Titre: Title:	Design and implementation of a high resolution, multi-hit time-to-digital converter (TDC) on FPGA
Auteur: Author:	Amir Mohammad Amiri
Date:	2007
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Amiri, A. M. (2007). Design and implementation of a high resolution, multi-hit time-to-digital converter (TDC) on FPGA [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7955/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7955/
Directeurs de recherche: Advisors:	Abdelhakim Khouas, & Mounir Boukadoum
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

DESIGN AND IMPLEMENTATION OF A HIGH RESOLUTION, MULTI-HIT TIME-TO-DIGITAL CONVERTER (TDC) ON FPGA

AMIR MOHAMMAD AMIRI

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME MAÎTRISE ÈS SCIENCES APPLIQUÉES

GÉNIE ÉLECTRIQUE

AVRIL 2007

Your file Votre référence
ISBN: 978-0-494-29210-5
Our file Notre référence
ISBN: 978-0-494-29210-5

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

DESIGN AND IMPLEMENTATION OF A HIGH RESOLUTION, MULTI-HIT TIME-TO-DIGITAL CONVERTER (TDC) ON FPGA

Présenté par: AMIRI, Amir Mohammad
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. DAVID Jean-Pierre, Ph.D., président
M. KHOUAS Abdelhakim, Ph.D., membre et Directeur
M. BOUKADOUM Mounir, Ph.D., membre et Codirecteur
M. LANGLOIS Pierre, Ph.D., Membre

To My Parents
& All Family Members and Friends

ACKNOWLEDGEMENT

I would like to say special thanks and offer special gratitude to professors Mounir Boukadoum and Abdelhakim Khouas for their excellent technical and moral support and guidance as my supervisors throughout the time that this research project was being carried out.

I would like to also pay my gratitude and appreciations to my parents, and all family members and friends that have been of enormous help for the duration of this research work. My success in carrying out and completion of this project would certainly not have been possible without their generous support.

I would like to also thank École Polytechnique de Montréal, the GRM (Groupe de Recherche en Microélectronique), L'université du Québec à Montréal (UQÀM), the Regroupement Stratégique en Microsystèmes du Québec (RESMIQ), and the National Science and Engineering Research Council of Canada (NSERC) for making it possible for me to carry out this research activity.

Further special appreciations are also offered to my friend, Mr. Jacque L. Athow, for his support in our mutual technical discussions related to FPGA architectures and associated issues.

And finally, my special thanks and appreciations go to all the stuff members of the Department of Electrical Engineering, and the GRM personnel for their excellent assistance and services in providing an excellent working environment for research students.

RESUMÉ

La notion et la mesure du temps ont leurs significations importantes dans divers domaines de la science et de l'ingénierie. Un intervalle de temps est défini comme le temps écoulé entre les occurrences de deux événements physiques, soit le retard de propagation d'un bloc logique dans les circuits électroniques numériques, ou les fluctuations entre les périodes consécutives d'un signal périodique tel que l'horloge du système, ou bien la distance dans le temps entre deux particules adjacentes libérées du point de collision dans les expériences de temps de vol (Time Of Flight). Un Convertisseur temps-numérique (Time-to-Digital Converter ou TDC) est un circuit électronique conçu pour mesurer le délai entre deux événements temporels. En général, un circuit TDC se compose d'un interpolateur grossier notamment un compteur binaire avec une résolution de mesure égale à la période de l'horloge de référence (T_{REF}) du système et d'un autre interpolateur divisant T_{REF} en plusieurs plus petits pas. Les différents paramètres caractéristiques d'un circuit TDC incluent la résolution de la mesure, la précision/exactitude des mesures, la gamme dynamique, les erreurs de non linéarité, le temps mort, ainsi que la *résolution paire d'impulsion* (Pulse-Pair-Resolution ou PPR) pour les circuits TDC capable de mesurer plus qu'un intervalle de temps dans une seule phase de mesure. Les méthodes d'interpolation du temps comprennent des architectures analogiques et numériques.

Ce projet de recherche propose un circuit TDC basé sur un nouvel interpolateur de temps utilisant le principe de Vernier afin d'atteindre une haute résolution de mesure. L'architecture de l'interpolateur est un tableau de lignes à délais de Vernier (VDL) utilisant des éléments tampon et des bascules à verrouillage D (*D-Latch*) comme élément à délai des lignes de propagation. Le tableau de lignes à délais peut aussi être vu comme une matrice à cellules de retard Vernier avec l'interconnexion entre les rangs successifs accomplis par les premières cellules de retard de chacune des lignes. Toutes les lignes du tableau partagent une horloge de référence commune. La matrice de retard mesure la distance dans le temps entre le front montant du signal de donnée et celui de l'horloge

appliqués aux entrées respectives de la matrice. L'interconnexion verticale de la matrice fournit un chemin de propagation avec le plus grand pas temporel τ_Y tandis que la résolution élevée τ_X de mesure est atteinte par la propagation horizontale à travers les lignes de retard de Vernier. Le circuit TDC mesure un intervalle de temps entre deux impulsions entrantes en prenant la différence des temps d'arrivée correspondants par rapport à un point de référence. En incorporant deux matrices de retard fonctionnant en parallèle dans les deux phases d'une période d'horloge, le temps morts du circuit est amélioré. Ainsi, le temps de mesure/interpolation du circuit est minimisé grâce au chemin de propagation vertical. Le TDC proposé est aussi capable de mesurer plus qu'un intervalle de temps dans une seule phase de mesure. Pour exploiter la portabilité et les aspects faible coût de conception et de fabrication, le prototype du circuit TDC a été implémenté sur la plateforme peu coûteuse de SPARTAN^{TM}-3 FPGA de la famille SPARTAN^{TM} de XILINX. En simulation, la résolution de mesure atteinte est de τ_X = 113ps, alors que les résultats expérimentaux donnent $\tau_X \sim$ 75ps, avec un PPR de l'ordre de 7.5ns. La déviation moyenne d'un intervalle mesuré par rapport à sa valeur idéale est de l'ordre de ~1.5 % pour les intervalles plus grands que 12.0ns. Cependant, cette déviation augmente à 3.5 % pour les intervalles dont les longueurs sont inférieurs à 12.0ns. Les sources principales d'erreur de mesure sont attribuées à l'existence du biais sur les lignes de l'horloge, à des variations dans les paramètres d'horloge de référence telles que les fluctuations périodiques (Jitter), l'erreur de phase et du cycle, la non uniformité du retard des cellules dû à la non uniformité de l'interconnexion des cellule adjacentes, ainsi qu'aux problèmes de mésappariement entre les éléments logiques identiques sur la surface du FPGA.

ABSTRACT

The notion of timing and time measurement has its significance in many areas of science and engineering. A time interval is defined as the time elapsed between occurrences of two physical events, be it the propagation delay of a logic block in digital electronic circuits from an input to the output, the jitter in a periodic signal such as the system clock, or the distance in time between two or more adjacent particles liberated from the collision point in Time-Of-Flight experiments. A Time-to-Digital Converter (TDC), also referred as Time Interval Meter (TIM), is the electronic circuit designed to measure the time elapsed between two timing events/pulses. A TDC circuit usually consists of two time interpolating modules namely the fine time interpolator module used to slice the period of the reference clock into smaller time steps, and the coarse interpolator (binary counter) used to achieve coarse measurement of time with a resolution equal to the reference clock period. The various characteristic parameters of a TDC circuit include the resolution of the measurement, the precision/accuracy of the measurements, the maximum measurable interval (dynamic range), non-linearity of the conversion process, and the dead time. For multi-hit TDC circuits capable of measuring more than one time interval sequentially, pulse pair resolution is another parameter of importance.

Over the years, a lot of research efforts have been devoted to come up with new or more improved TDC structures. The many TDC solutions proposed by researchers involve design methods that vary from the early analog-based solutions to the recent partial or fully digital ones. The time-to-amplitude conversion and the time interval stretching are the famous analog-based time interpolation methods used in many TDC circuits. On the digital side, the interpolation solutions are mainly achieved by means of delay lines with resolutions proportional to single gate delay or the variants of delay lines such as the Vernier principle with sub-gate delay resolutions. Other complex structures with improved characteristics have also been proposed.

The proposed TDC circuit in this research work is based on a new time interpolator utilizing the Vernier principle in order to achieve the high measurement resolution. The time interpolator is an array of Vernier delay lines (VDL), also seen as matrix of Vernier delay cells, with the interconnection between successive rows achieved through the first delay cells of each line with a common reference clock line shared among all. The delay matrix measures the distance in time between the rising edges of the data and the subsequent clock signals applied at the respective data and clock inputs of the matrix. The matrix structure breaks this interval into two-level time bins of $\tau_Y = t_L$ and $\tau_X = t_L - t_B$ with t_B and t_L as the propagation delays of the buffer and latch elements of the matrix delay cells. The vertical interconnection provides a propagation path with larger time bin τ_Y whereas the single-shot measurement resolution τ_X is achieved by horizontal propagation throughout the Vernier delay lines. The TDC circuit measures an interval of time between two incoming pulses by taking the difference in corresponding arrival times with respect to some reference point.

The TDC circuit is implemented on a low-cost SPARTAN-3 FPGA from the SPARTAN^{TM} family by XILINX in an attempt to exploit the recent advances in programmable logic devices. The measurement resolution attained is τ_X = 113ps in simulation, whereas the experimental results give $\tau_X \sim$ 75ps. The TDC circuit is capable of measuring multiple sequential intervals with a PPR ~ 7.5 ns. The average deviation of a measured time interval T from its ideal value is ~1.5% for intervals larger than 12.0 ns and 3.5% for intervals with ideal lengths less than 12.0ns. The main sources of measurement error are attributed to existence of clock skew on clock lines in the delay matrices, variations in reference clock parameters in forms of jitter, phase and duty cycle, non-uniformity of cell delays due to existence of non-uniform routings, and the on-chip delay mismatch among identical logic elements on FPGA surface.

CONDENSÉ EN FRANÇAIS

0.1 Introduction

Un convertisseur temps-numérique (Time-to-Digital Converter ou TDC) est un circuit électronique conçu pour mesurer le délai entre deux événements (impulsions) détectés à l'entrée du TDC. En utilisant un circuit TDC, l'intervalle de temps est converti en un mot binaire pour le traitement numérique. Avec l'évolution rapide du monde numérique, comme plusieurs systèmes électroniques, le circuit TDC a migré d'une implémentation analogique vers une implémentation numérique.

Les applications nécessitant la mesure d'intervalles de temps se retrouvent dans divers domaines scientifiques et d'ingénierie. Dans un circuit électronique, l'intervalle de temps à mesurer pourrait représenter le retard de propagation d'un bloc logique entre le front montant/descendant d'un signal de test à l'entrée du circuit et le front correspondant à la sortie. De même, les fluctuations entre les périodes consécutives (Jitter) d'un signal périodique tel que l'horloge d'un système est un autre exemple concret pour lequel une mesure d'intervalle est utile. Dans les expériences de temps de vol (Time-of-Flight) et de spectrométrie massive, la distance dans le temps entre deux particules adjacentes déchargées du point de collision est mesurée pour trouver la masse moléculaire des particules relâchées. Dans l'instrumentation, les oscilloscopes numériques et les analyseurs logiques à haute résolution exigent la mesure précise d'intervalles de temps pour échantillonner les formes d'onde numériques. Parmi les autres applications de mesure d'intervalle de temps, on retrouve la spectroscopie de vie de positron et les appareils de viseur laser utilisés pour trouver la distance entre l'objet et le point de référence (position d'appareil), où la mesure du temps est un des éléments clés du système général.

Un intervalle de temps à mesurer (T_m) est défini entre les fronts montants de deux impulsions ordinairement nommées comme START et STOP. Étant plus grand que la période d'horloge de référence, T_m peut être divisé en sous intervalles fins et grossiers. La section grossière de T_m est divisible par la période d'horloge du système, donc mesurable par un compteur avec une résolution de T_{REF_CLK}. Cependant, pour mesurer les sections fines de T_m, il est nécessaire d'interpoler l'horloge du système en divisant la période d'horloge du système en plusieurs plus petits pas (voir Figure 1.2).

Les diverses métriques de performance d'un circuit TDC incluent la *résolution de mesure*, *la précision de mesure, la gamme dynamique*, *le temps de mesure*, *le temps mort*, *le multi-hit*, *la résolution paire d'impulsion* (pulse-pair resolution ou *PPR*), et *la non linéarité de conversion.* La *résolution de mesure* est le plus petit pas de temps qui peut être discriminé pour une mesure de temps, elle représente la finesse ou la granularité de la mesure. La *précision de mesure* représente le degré d'exactitude de la valeur mesurée par rapport à la véritable valeur. La *gamme dynamique* est l'intervalle maximal de temps mesurable par un circuit TDC. Le *temps de conversion/mesure* peut être défini comme le temps de traitement de l'intervalle à mesurer, c'est-à-dire de la détection de l'impulsion STOP jusqu'à l'obtention du mot binaire correspondant. L'appellation *temps de mesure* est également utilisée. Le *temps mort* est l'intervalle de temps le plus court qu'un TDC peut exiger entre la fin d'une mesure et la possibilité de démarrer la mesure suivante. Le *multi-hit* est la capacité d'un circuit de TDC de mesurer plus qu'un intervalle de temps dans une seule phase de mesure sans retourner à un état initial. La *résolution paire d'impulsion* (*PPR*) est la distance minimale dans le temps entre deux impulsions reçues qui peut être mesurée par un circuit TDC à multi-hit. La déviation de la réponse du convertisseur par rapport aux valeurs prévues est due aux erreurs de non linéarité intégrale et différentielle.

En prenant en considération les diverses caractéristiques des circuits TDC, le travail actuel vise l'amélioration de la résolution de mesure, du temps mort, et de la capacité de multi-hit, et une résolution paire d'impulsion (*PPR*) raisonnable. De plus, inspirés par les avancements récents dans les circuits logiques programmables tels que les FPGA, nous avons choisi de nous concentrer sur la portabilité et sur les aspects faible coût de conception et de fabrication. Nous avons donc ciblé la technologie FPGA en implémentant notre prototype sur la plateforme peu coûteuse de SPARTAN^{TM}-3 FPGA de la famille SPARTAN^{TM} de XILINX.

0.2 Revue de littérature des méthodes d'interpolation de temps existantes

Au cours des dernières années, des efforts de recherche énormes ont été consentis dans le but de proposer des structures nouvelles et plus robustes pour les méthodes de mesure des intervalles de temps. L'objectif est de surmonter les diverses limitations affectant le comportement des circuits existants. Les solutions proposées comprennent des méthodes de conception analogiques et numériques.

Les fameuses techniques d'interpolation analogique incluent la conversion temp-à-amplitude (Time-to-Amplitude or TAC) et la méthode d'allongement d'intervalle (Time Interval Stretching or TIS). Avec la technique du TAC, l'intervalle de temps à mesurer (T_m) est transformé en une amplitude de tension V par chargement/déchargement d'un condensateur à l'aide d'un courant constant I pour une durée de T_m. Par la suite, V est converti en un mot binaire numérique utilisant un convertisseur analogique-numérique (CAN). Avec cette méthode, l'intervalle de temps à mesurer dépend linéairement de l'amplitude de la tension aux bornes du condensateur. La tension V aux bornes du condensateur devrait être maintenue jusqu'à ce que le mot binaire correspondant soit obtenu à la sortie du CAN. Le temps de conversion du TAC dépend principalement du temps requis par le CAN pour convertir la tension V, alors que la résolution de mesure du TAC dépend de la résolution de conversion du CAN utilisé.

TIS est une autre méthode analogique d'interpolation d'intervalle de temps. Ici, le petit pas de mesure est obtenu en convertissant T_m en un intervalle plus grand T donné par $T = T_m + T_{str.}$ Cet intervalle est obtenu en additionnant T_m avec une version tendue de T_m par un facteur k $(T_{str} = kT_m)$. Par la suite, T peut être mesuré par un compteur avec une horloge de fréquence $f_0 = 1/T_0$ menant à $T = T_m + T_{str} = (k + 1)\ T_m = N\ T_0$. Donc, l'intervalle T_m à mesurer peut être calculé comme: $T_m = NT_0 / (k + 1)$ avec une résolution de mesure de $T_0 / (k + 1)$. Avec la méthode TIS, une résolution de mesure de l'ordre de 10 ps a été rapportée [11]. Malgré sa haute résolution de mesure, TIS souffre d'un temps de conversion élevé dû au temps d'allongement kT_m.

Dans le domaine numérique, il existe plusieurs solutions TDC incorporant différentes méthodes d'interpolation de temps. Une des techniques numériques les plus simples pour atteindre une grande résolution est l'interpolateur de ligne de retard qui utilise une chaîne d'éléments de retard [17][18] (voir Figure 2.3). Conduite par un signal s, une chaîne de retard à N-éléments avec un délai de propagation t_d pour chacun des éléments de retard retardera le front principal de s par t_d lorsque le signal traversera la chaîne. En calculant le nombre d'étages propagés par s, l'intervalle (T_m) peut être mesuré avec une résolution de mesure de t_d. La gamme dynamique pour un interpolateur de ligne de retard dépend du nombre d'éléments de retard utilisés dans la chaîne. Cependant, la gamme dynamique peut être augmentée à l'aide d'un compteur binaire incorporé dans le système en parallèle. En remplaçant la chaîne à délai par une chaîne à délai d'un DLL (Delay-Locked-Loop), une interpolation plus efficace est obtenue [20][21][22]. Le DLL peut être verrouillé à un retard fixe avec les subdivisions égales au délai du retard configurable t_d des éléments de la chaîne à délai du DLL. L'utilisation d'un DLL pour l'interpolation de temps est une méthode efficace du point de vue de la stabilité, cependant, la résolution de mesure est toujours limitée au retard minimum d'un élément de délai.

Pulse Shrinking (PS) est une autre technique utilisée dans la mesure d'intervalle de temps [17][23][24][25][26]. Elle consiste à mesurer une largeur W d'une impulsion par la réduction successive de W par une valeur constante ε, jusqu'à la disparition complète de l'impulsion lorsque elle traverse la chaîne d'éléments à délai (voir Figure 2.6). Les éléments de retard de la ligne à délai sont contrôlés par une tension avec les temps de montée et de descente variables qui raccourciront la largeur d'une impulsion d'entrée au noeud de la sortie par la valeur constante de ε (voir Figure 2.8). De cette façon, la largeur d'impulsion est mesurée avec une résolution de mesure égale à la valeur de réduction ε. Le rétrécissement de l'impulsion dans la mesure d'intervalle de temps impliquant les impulsions START et STOP exige une conversion à deux niveaux. Premièrement, l'intervalle de temps T_m est transformé en une largeur W d'impulsion suivie d'une conversion subséquente de W dans le mot binaire. Bien que la résolution de mesure des solutions TDC précédentes à base de PS était dans la gamme de centaines de picosecondes [17][25], les nouvelles méthodes améliorées de réduction d'impulsion (Cyclic Pulse Shrinking) ont atteint des résolutions de mesure dans la gamme de 20 ps à 70 ps [23][24][26].

Avec les avancements récents dans la technologie rendant des systèmes fonctionnels à haute fréquence, les circuits TDC à basse résolution, notamment les TDC utilisant des chaînes à délai qui ont une résolution de mesure de l'ordre du délai des éléments de délai, pratiquement inutilisable. De nouvelles méthodes de conception visant à atteindre des résolutions de temps beaucoup plus élevées devaient être développées. Une des méthodes largement utilisées pour atteindre une résolution plus élevée est basée sur le principe de Vernier [27] (voir Figure 2.9). Le petit pas de temps par le principe de Vernier est atteint par une petite différence entre les périodes de deux signaux périodiques produits par deux oscillateurs en anneau (Ring Oscillator). Les deux oscillateurs sont déclenchés par les signaux de START et STOP et produisent des signaux périodiques de périodes T_1 et T_2, respectivement avec $T_1 > T_2$. Les deux oscillateurs sont remis à l'état initial dès que la coïncidence entre les fronts de START et STOP est détectée à l'aide d'un circuit de détection de front (Edge Detector). L'intervalle $T_m = t_{START} - t_{STOP}$ est mesuré en considérant la différence entre le nombre cumulatif des périodes produites par les deux oscillateurs, comptées par leurs compteurs respectifs. De cette façon, la plus petite unité de temps qui peut être discriminée pour trouver T_m est $t_r = T_1 - T_2$ [11].

D'autres variantes de la méthode de Vernier permettent d'atteindre une résolution de mesure élevée en exploitant la différence dans les délais de propagation des éléments logiques [27][29]. Pour de telles méthodes, les signaux concernés sont propagés à travers deux chaînes à délai différentes A et B contenant des éléments logiques identiques. Dépendamment de la méthode de conception, les chaînes A et B peuvent être composées d'éléments tampon, de bascules, ou d'autres éléments logiques. Par défaut, les délais des éléments dans une des deux chaînes, par exemple A, sont légèrement plus élevés que ceux de l'autre chaîne B. La plus petite unité de temps discriminable est donnée par la différence de retard $\Delta t = t_a - t_b$, où t_a et t_b sont les délais de propagation des éléments dans les lignes A et B respectivement.

Afin d'augmenter la résolution de mesure, d'autres circuits TDC combinent plusieurs chaînes d'interpolation de temps dans des topologies plus complexes. Une telle structure combine M lignes de retard à N étages dans une topologie de tableau [18] (voir Figure 2.13). Dans cette topologie, les signaux de données sont propagés à travers M échantillonneurs de temps. Les entrées de données et d'horloge des échantillonneurs adjacents sont retardés par les petits retards respectifs de t_d et de t_c, par lesquels la résolution simple d'interpolateur t_d est divisée en M, donnant une résolution finale de $\Delta t_r = t_d/M$. Dans le même ordre d'idée, une autre solution de TDC combine des circuits DLL multiples dans une structure de tableau pour fournir une résolution de mesure améliorée [voir section 2.7]. Avec cette méthode, une résolution RMS de 34,3 ps, une gamme dynamique de 3,2 ns, et un LSB de 89,3 ps ont été obtenus avec une fréquence d'horloge de référence de 80 MHz [32].

0.3 Architecture TDC proposée

L'objectif principal de ce projet de recherche consiste à réaliser un circuit TDC permettant de surmonter certaines limitations de performance qui affectent les solutions TDC précédentes. En particulier, les objectifs initiaux de ce projet comprennent la réduction du temps de mesure, la capacité de mesurer plusieurs intervalles de temps dans une seule phase de mesure, l'obtention d'une résolution de mesure améliorée et la réalisation du circuit sur une plateforme peu coûteuse de FPGA. D'après le travail de conception cité dans [29], une résolution de mesure acceptable de 100 à 200 ps est atteignable par la méthode de VDL proposée. Pour une ligne de VDL à M étages divisant un intervalle de temps T en N petits pas t_r ($t_r = t_a - t_b$ où t_a et t_b sont les délais des éléments de logique dans les deux lignes à délai de VDL A et B et $t_a > t_b$), l'intervalle maximale à mesurer est donné par $T_{max} = M\ t_r$. Le temps requis pour mesurer T_{max} est donné par le nombre d'étages propagé multiplié par le plus grand délai des élément dans la ligne VDL notamment t_a. C'est-à-dire, pour mesurer $T_{max} = M\ t_r$, il faut que les impulsions concernées propagent M étages avec un délai cumulatif de $M\ t_a$. Dans ce cas, le temps perdu est donné par $(M\,t_a - M\,t_r) = M\,t_b$. Si l'intervalle d'intérêt est grand et qu'il exige une ligne de VDL avec beaucoup plus d'étages, le temps supplémentaire de mesure requis peut devenir un défi de la conception. La topologie proposée est basée sur la technique de VDL présenté dans [29]. Pour réduire le temps de mesure, la méthode proposée utilise deux chemins de propagations pour les impulsions d'entrée dont le délai de propagation τ_Y dans le premier chemin est plus grand que le délai τ_X du deuxième chemin. En utilisant des pas plus grands, plusieurs propagations coûteuses de l'impulsion sont évitées. Ceci réduit significativement le nombre d'étapes de VDL exigées et, par conséquent, réduit considérablement le temps de mesure supplémentaire. Pour atteindre la détection et le traitement continus des données de mesure, le travail actuel utilise le parallélisme. En incorporant plus d'un interpolateur de temps en parallèle, le temps mort du circuit est amélioré. La plate-forme visée pour la réalisation du circuit TDC est basée sur le FPGA faible coût SPARTAN^{TM}-3 de XILINX.

Tel que mentionnée ci-haut, le circuit TDC proposé est basé sur un interpolateur de temps utilisant le principe de Vernier afin d'atteindre une grande résolution de mesure. L'interpolateur de temps est un tableau de lignes à délai de Vernier (VDL) utilisant des éléments tampon et des bascules à verrouillage D (D-Latch) comme élément à délai des lignes de propagation (voir Figure 4.2). L'interconnexion verticale entre deux rangs successifs est fournie par les premières cellules de retard de chacune des lignes. Toutes les lignes du tableau partagent une seule horloge de référence commune. L'architecture proposée peut aussi être vu comme une matrice de cellules à retard. La matrice de retard mesure la distance dans le temps entre les fronts montants des signaux de données et d'horloge subséquentes provenant des entrées respectives de la matrice. Cet intervalle est brisé en petits pas de temps à deux niveaux, notamment $\tau_X = t_L - t_B$ et $\tau_Y = t_L$ avec t_B et t_L étant les délais de propagation des éléments tampon et des loquet-D, respectivement. L'interconnexion verticale de la matrice fournit un chemin de propagation avec le plus grand pas de temps tandis que la résolution élevée de mesure est atteinte par la propagation horizontale à travers les lignes de retard de Vernier.

Le circuit TDC mesure un intervalle de temps entre deux impulsions entrantes en prenant la différence des temps d'arrivée correspondants par rapport à un point de référence. La matrice de cellules de retard sert donc à mesurer le temps d'arrivée des impulsions par rapport à l'horloge de référence qui, par la suite, sont utilisées pour déterminer la distance dans le temps entre deux impulsions consécutives.

À chaque saut vertical effectué par l'impulsion de donnée qui se propage, la distance dans le temps entre l'impulsion et le front d'horloge suivant est réduit par τ_Y. Ce décalage continue jusqu'au moment où le sous intervalle résultant est plus court que $\tau_{Y.}$ À ce moment, l'impulsion commence à propager horizontalement avec des pas de temps plus petits.

L'intervalle mesurable maximal obtenu en utilisant la matrice de retard est donné par les délais de propagation de N-1 Loquet-D verticaux et de M délais différentiels horizontaux. C'est-à-dire : $T_{max} = (N\text{-}1)\ \tau_Y + M\ \tau_X$. Pour mesurer T_{max}, l'impulsion en question doit se propager à travers (N+M-1) Loquet-D, correspondant aux coordonnées (N-1, M), ce qui exige un temps de mesure total de $T_{total} = (N + M - 1)\ t_L$. Pour mesurer T_{max}, le temps perdu maximal T_{ex} est donnée par $T_{ex} = T_{total} – T_{max} = (N + M - 1)\ t_L – [(N\text{-}1)\ \tau_Y + M\ \tau_X] = M\ t_B$.

Une autre caractéristique importante est l'implémentation de deux matrices fonctionnant en parallèle, avec deux signaux d'horloge qui sont déphasés de 180 degrés. La présence de deux matrices de retard fonctionnant chacune dans une phase d'une période d'horloge permet la détection et le traitement continu d'impulsions successives, tout en subissant un PPR minimal. Ceci réduit considérablement le temps mort requis pour réactiver le système pour les prochaines mesures.

Idéalement, quand une impulsion fait un saut d'une rangée à l'autre, elle ne devrait exister nulle part dans la rangée précédente. Pour éviter la problématique de multi détection des mêmes impulsions sur plus d'un rang, chaque rangée de la matrice, à l'exception de la première, est équipée d'un circuit de réinitialisation qui, lors de la détection d'une nouvelle impulsion, génère un signal RESET d'une largeur constante, ce qui remet toutes les cellules de retard dans la rangée précédente à l'état initial. De plus, un générateur d'impulsions interne générant des impulsions d'une largeur constante est incorporé pour éviter la re-propagation d'une impulsion d'une largeur plus grande que la phase de l'horloge de référence à la phase active prochaine, ce qui autrement produirait des lectures erronées.

Le choix de taille de la matrice dépend principalement de la fenêtre de temps d'observation T_{OBS} ciblé et de la résolution verticale τ_Y. Avec T_{OBS} et τ_Y donnés, le nombre N de rangées à cellules de retard est donné par $N \geq (T_{OBS}/\tau_Y)$ où $\tau_Y = t_L$ et $T_{OBS} = T_{REF}/2$. De même, le nombre M d'étages est donné par $M \geq (\tau_Y / \tau_X)$. La valeur de la fenêtre de temps d'observation T_{OBS} dépend du temps requis pour qu'une impulsion se propage à travers la matrice au complet ainsi que de l'intervalle minimal de temps entre l'arrivée du front du signal d'échantillonnage et le prochain front d'horloge indiquant la prochaine phase de détection. Ce dernier garantit le temps requis par les processus d'encodage et de sauvegarde temporaire des données acquises avant le début d'une nouvelle phase active de détection.

Donc, la taille de la matrice de retard dépend de la résolution de mesure recherchée, des délais des éléments tampon et des Loquet-D utilisés, ainsi que de la valeur de T_{OBS}. Une taille minimale de matrice est obtenue si la résolution τ_X est atteinte avec des éléments de tampon et de Loquet-D ayant un délai de propagation minime, ce qui amène une petite fenêtre d'observation et, par conséquent, une petite taille de la matrice.

0.4 Résultats

Le circuit de test est composé de deux matrices de retard fonctionnant en parallèle, une unité mémoire temporaire FIFO (First-In-First-Out), un encodeur permettant d'encoder les données détectées, un contrôleur basé sur des machines à états finis (FSM), un décaleur (Shifter), une mémoire de taille 512 X 32 octets, et un contrôleur d'affichage (LCD driver) pour l'affichage des données sur l'afficheur LCD (Liquide Crystal Display) (voir Figure 4.12). Le système a été implémenté sur une plateforme de prototypage utilisant un circuit FPGA de type SPARTAN^{TM}-3 de XILINX. Les matrices de retard, synchronisées par deux horloges déphasées de 180 degrés, génèrent les données à chacune des phases de l'horloge de référence.

Pour atteindre l'uniformité exigée des délais et des retards des éléments tampon et des loquet-D, chaque cellule de retard des matrices a été manuellement placée. Les résultats de simulation de ModelSim et de l'outil d'analyse de temps de ISE de XILINX (Timing Analyser) donnent des valeurs moyennes de $\tau_Y \sim 1.4$ ns et de $\tau_X \sim 100$ ps. La taille de chaque matrice de retard a été définie à 25 x 15 (N=25 et M = 15).

Les résultats expérimentaux donnent des délais de propagation qui sont légèrement différents de ceux obtenus en simulation. Afin d'estimer les valeurs des délais exactes des lignes de propagation verticales et horizontales, des lignes de calibration à base d'un oscillateur en anneau (Ring Oscillator ou RO) ont été implémentées. Pour les délais de propagation verticaux, le retard est estimé par la différence entre les périodes d'un RO de référence et d'un RO contenant trois cellules supplémentaires placées et connectées de façon identiques aux connexions de cellules verticales dans les matrices. De même, les lignes de calibration horizontales ont été placées et connectées comme les lignes de VDL horizontales des matrices. Les résultats de ces lignes de calibration donnent une résolution verticale d'une valeur de ~1.0 ns, et la résolution horizontale varie entre 80 et 120 ps. Ainsi, le circuit est capable de détecter les impulsions séquentielle avec une PPR de 8.0 ns. La détection continuelle des impulsions d'une façon séquentielle élimine le temps mort associé à un circuit TDC conventionnel. Le circuit a été aussi testé avec différentes valeurs d'intervalles. Les résultats mesurés montrent une erreur de mesure moyenne de l'ordre de ~1.5 % pour les intervalles plus grands que 12.0ns. Cependant, cette déviation augmente à 3.5 % pour les intervalles dont les longueurs sont inférieurs à 12.0ns. La valeur du skew maximal dans les lignes de l'horloge a été estimée de l'ordre de 400 ps.

0.5 Contraintes de performance

Pendant la réalisation et l'implémentation du circuit TDC proposé, plusieurs contraintes ont dû être respectées. Les contraintes les plus importantes sont : le placement manuel des éléments logiques, le routage identique des cellules de la matrice dans le but d'obtenir des délais uniformes, la minimisation du biais des lignes d'horloge, la limitation de la surface occupée par la structure symétrique de la matrice, et l'obtention d'un routage identique pour le chemin de données à partir de l'entrée principale du circuit jusqu'aux entrées de données respectives des matrices correspondantes. Au cours de la réalisation du projet et afin de respecter les conditions et limitations imposées et d'arriver à un système fonctionnel sur la plateforme FPGA, l'outil de synthèse et d'implantation de XILINX ont été guidés en spécifiant plusieurs contraintes de synthèse et d'implantation pour chacune des phases correspondantes.

Les principales sources d'erreurs sont liées à l'incertitude dans l'uniformité des délais des cellules à retard et la présence de biais dans les lignes d'horloge. De plus, la performance du système est aussi affectée par d'autres facteurs tels que les variations dans les paramètres de l'horloge comme les variations de la période (Jitter), de la phase, et du duty cycle, ainsi que les fluctuations des délais de propagation des éléments identiques sur une même puce, qui sont principalement causées par les fluctuations des paramètres du procédé de fabrication et de l'environnement.

0.6 Conclusion et travaux futurs

Ce projet de recherche propose un circuit TDC à base d'une nouvelle méthode d'interpolation de temps. La technique d'interpolation proposée divise la période d'horloge de référence en tranches de N pas de temps et chacun des pas est ensuite divisé en M plus petits pas. Cette interpolation à deux niveaux réduit le temps de mesure en éliminant beaucoup de propagation successive de données à travers les lignes de retard pour couvrir la gamme dynamique de l'interpolateur. Les impulsions à l'entrée sont propagées avec des pas de temps plus grands jusqu'à ce que le temps restant à propager devienne plus petit que le pas en question. Ensuite, les plus petits pas de temps sont utilisés pour obtenir une meilleure résolution. Le circuit prototype a été réalisé sur une plate-forme SPARTAN^{TM}-3 FPGA de Xilinx. La résolution de mesure atteinte est de l'ordre de 100-120 ps.

Les travaux futurs sur ce projet consisteraient en l'amélioration de la performance du circuit TDC par, possiblement, une re-implémentation du circuit sur un prototype de FPGA plus récent avec les chemins d'horloges améliorés. Ainsi, l'amélioration des interconnexions entre les éléments des cellules adjacentes par un algorithme approprié garantissant un routage équilibré et uniforme afin de respecter l'uniformité des délais est fortement suggérée. De plus, il serait très intéressant de réaliser une implémentation sur ASIC du système permettant d'avoir des routages symétriques, et en conséquence, des délais de propagation plus uniformes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SIGNS AND ABREVIATION

A/D	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
ASM	Algorithmic State Machine
BEL	Placement constraint used by Xilinx PAR tool
BRAM	Block Random Access Memory
CDT	Code Density Test
CLB	Configurable Logic Block
DCM	Digital Clock Manager
DLL	Delay-Locked Loop
DNL	Differential Non-Linearity
FF	Flip-Flop
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GCD	Greatest Common Divisor
INL	Integral Non-Linearity
LCD	Liquid Crystal Display
LOC	Placement and synthesis constraint in Xilinx Synthesis Tool
LSB	Least Significant Bit
LUT	Look-up Table
MAXDELAY	XILINX timing constraint for maximum delay on a net
MAXSKEW	XILINX timing constraint for maximum skew among shared paths
MSB	Most Significant Bit
NCD	Native Generic Database
NGC	Native Circuit Description
PPR	Pulse Pair Resolution
PRPG	Pseudo-Random Pulse Generation/Generator
RAM	Random Access Memory

RO	Ring Oscillator
TAC	Time-to-Amplitude Conversion
t_B	Propagation delay of a buffer logic element
TDC	Time-to-Digital Converter
TIM	Time Interval Meter
TIS	Time Interval Stretching
t_L	Propagation delay of a latch
T_m	Interval-to-be-measured
T_{OBS}	Observation Time Window (Detection time window for Measurement)
TOF	Time-of-Flight
T_{REF}	Period of the reference clock
T_{RO}	Period of Ring Oscillator
τ_X	Single shot measurement resolution
τ_Y	Vertical measurement resolution (larger time step)
UCF	User Constraint File
VCDL	Voltage-Controlled Delay Line
VDL	Vernier Delay Line
HDL	Hardware Description Language
XST	Xilinx Synthesis Tool

CHAPTER 1 - INTRODUCTION

Many applications in science and engineering require precise measurement of time intervals or the time elapsed between occurrences of two physical events. Depending on the nature and the requirements set by the application of concern, some applications need precise measurements for shorter intervals of time while some others may necessitate relatively longer time interval measurements. Regardless of the magnitude, it is often desirable to achieve timing measurements that are accurate and well representative of the actual elapsed time.

Applications involving time interval measurement are wide-spread in various fields of science and engineering. In electronic circuits, the time interval of interest could represent the propagation delay of a logic block from the rising/falling edge of a test signal at the input to the corresponding rising/falling edge at the output. Similarly, the jitter present in consecutive periods of a periodic signal such as the system clock is a good candidate for the interval of interest [1]. In Time-Of-Flight (TOF) and mass spectrometry experiments [2][3][4]][5][6], the distance in time between two or more adjacent pulses or particles liberated from the collision point are to be measured for finding the molecular mass of the particles released. In instrumentation, the high resolution digital oscilloscopes and logic analyzers require precise measurements of time intervals for sampling digital waveforms and signal timings [7]. Other applications where timing measurement is one of the key elements of the overall process include applications such as positron lifetime spectroscopy [8] and laser range finder devices [9] used in finding an object's distance from the reference point (the device location).

A Time-to-Digital Converter (TDC), also referred as Time Interval Meter (TIM), is an electronic circuit designed to measure the time elapsed between two timing events/pulses. With the rapidly advancing digital world, many of the originally analog-domain parameters have been embedded into the digital domain for better and easier data processing. A TIM circuit is one of such structures that have evolved from its earlier analog domain time measurement methods to the most recent digital ones [10][11]. Using a TDC, the time interval of interest is converted into a binary word representation that can easily be used in other digital blocks for further processing.

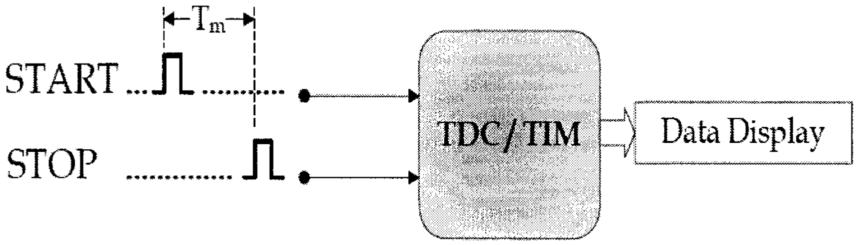


Figure 1.1: Abstract view of a TDC circuit

A time interval to be measured (T_m) is defined between rising edges of two pulses commonly named as START and STOP. The interval T_m is usually described in terms of fine and coarse subintervals. The coarse section of T_m (T_2 in Figure 1.2) is divisible by the system clock period, hence measurable with a resolution of T_{REF} by a binary counter clocked with a frequency of $1/T_{REF}$. However, to measure the fine constituents of T_m (T_1, T_3 in Figure 1.2), further interpolation of the system clock (division of system clock period into many smaller time bins) is needed. From Figure 1.2, it is seen that with the fine and coarse subintervals measured, T_m can be given as:

$$T_m = T_1 + T_2 - T_3 \tag{1.1}$$

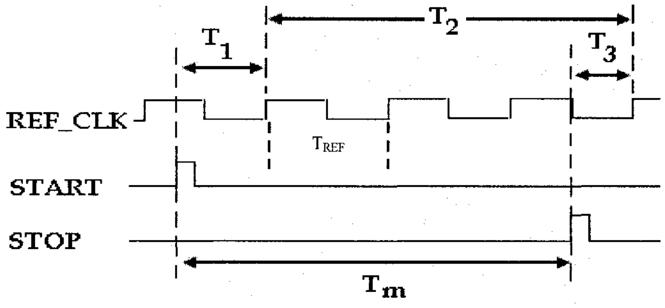


Figure 1.2: The interval (T_m) in terms of its constituent sub-intervals

With T_m defined by eq. 1.1, typical TDC circuits consist of a coarse time counter running with some reference clock and a fine interpolator as depicted in Figure 1.3.

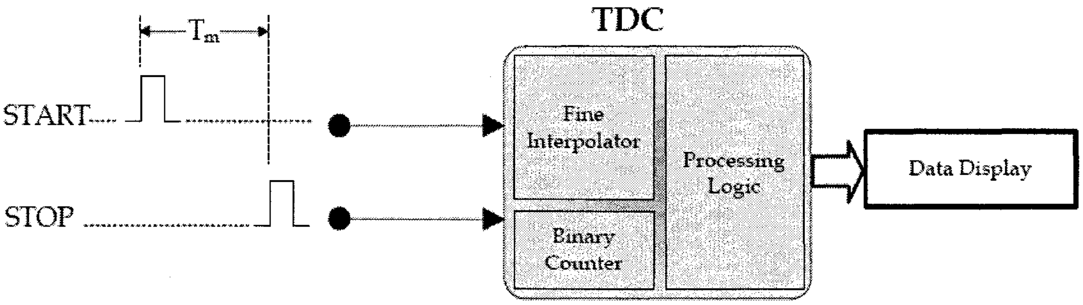


Figure 1.3: Further refined view of a TDC circuit

While digital counter design methods have been pretty much well established to address coarse timing measurements, the focus of the recent research efforts have mostly been in the area of time interpolators to provide finer resolutions. In theory, a particular time interpolator slicing the reference clock period into N equal intervals will achieve a resolution of T_{REF} / N. In practice, however, the finer resolution of the interpolator is affected by factors such as the quantization error, signal jitter in reference clock signal, offset errors and other design-dependent factors. Throughout the years, researchers have come up with many TDC solutions, each targeting a particular performance metric of the TDC. Some of the early-proposed solutions are further discussed in the next chapter.

1.1 General Terms and Characteristics

Similar to any electronic circuit, a TDC comes with its own performance metrics that are of interest and concern to designers during the design phase. Some of the important terms and characteristics associated with TDC circuits are as follows:

Measurement Resolution: The smallest time unit that a time interval T_m can be described with is referred to as the measurement resolution. It is the smallest time quantity that can be discriminated in a timing measurement. For instance, a measured time interval $T_m = 25.6$ ns has a resolution of 0.1ns or 100 picoseconds.

***Dynamic/Measurement Range*:** Usually extended by a binary counter, the dynamic or measurement range is the maximum time interval measurable by a TDC.

***Conversion/Measurement Time*:** Conversion time of a TDC can be defined as the processing time of the interval to be measured from the detection of the STOP pulse up to the binary word or any other form used to represent the time interval.

***Dead time*:** The shortest time required by a TDC from the end of a measurement to come back to the initial ready state is commonly referred as conversion dead time.

Multi-hit: This is the capability of a TDC circuit to measure more than one time interval, possibly from a common reference point in time, in a single measurement phase without having to go back to an initial ready state.

Pulse-Pair Resolution (PPR): *PPR* is the minimum distance in time between two incoming data pulses that can be measured by a multi-hit capable TDC circuit.

Non-Linearity Errors: Similar to analog-to-digital and digital-to-analog converters, the transfer curve of a TDC circuit may deviate from the expected response. Non-linearity of a converter is a measure of output deviation from the expected values under differential and integral non-linearity errors. *Differential non-linearity* refers to the difference between the actual time width corresponding to a digital code and the ideal one. *Integral non-linearity* of a TDC refers to the difference between the actual code transition points and a line connecting the ideal end points of the conversion.

Measurement Precision: Precision of the measurement refers to the degree of coherence among several time interval measurements.

Measurement Accuracy: Accuracy portrays the degree of agreement of the measured value with respect to the actual value of the unknown.

While all the associated characteristics are important to meet the requirements for proper operation of a TDC circuit, researchers have given more attention to certain parameters such as resolution, dead time, dynamic range, and multi-hit capability. Timing resolution is important as it reduces the measurement uncertainty while increasing the accuracy of the measurement. In Time-of-Flight experiments, an improved timing resolution has a direct impact on the effectiveness of the particle identification [12]. Reducing the dead time increases performance of the TDC circuit by increasing the number of applicable measurements in a given time window. Similarly, having a large dynamic range, a TDC is no longer confined to short time intervals and it can process relatively larger time intervals.

1.2 Project Intent

Understanding the importance of the various characteristics of the TDC circuits, the current work focuses on achieving high measurement resolution, low dead time, and multi-hit capability with a reasonable pulse-pair resolution. Furthermore, inspired by the recent advances in the programmable logic devices such as Field Programmable Gate Array (FPGA), we have chosen to focus on portability and low cost aspects of designs targeting FPGA technology by implementing our prototype on the low-cost SPARTAN-3 FPGA from the SPARTAN^{TM} family by XILINX. A partial objective of the project is also to explore the degree of feasibility of timely constraint designs such as a TDC on an FPGA platform. FPGA-targeted designs have several advantages such as low cost, fast prototyping and implementation, re-usability of silicon, easy integration within a larger system, and the flexibility of upgrading and modification of the design over the ASIC counterpart. However, despite the various advantages, designs targeting FPGA platforms may suffer from relatively lower performance compared to the corresponding ASIC solution.

With a general introduction to TDC circuits in this chapter, the organization of the subsequent chapters is as follows: Chapter 2 compiles a study of several time interpolation methodologies used in accomplishing high resolution TDC circuits. In chapter 3 some important performance metrics are discussed. Chapter 4 presents the proposed TDC solution by discussing all details relating to the design and implementation. The test strategies along with simulation/experimental results are presented in chapter 5. Finally, the concluding remarks and the future work are contained in chapter 6.

CHAPTER 2 - OVERVIEW OF TIME INTERPOLATION TECHNIQUES

Over the years there have been enormous efforts by researchers in the field to come up with new and more robust structures for time interpolation in an attempt to overcome the various limitations that may have affected the behaviour and performance of existing time interpolators. The many solutions proposed by researchers encompass both analog and digital design methods. While the early TDC solutions were mostly analog-based, partial or full digital design methods have pretty much dominated for the past two decades. In general, digital methods are preferred over the analog counterpart due to their relative design simplicity, less susceptibility to external disturbance, and less sensitivity to ambient temperature [11]. In this chapter, some of the many existing TDC architectures are touched upon in order to provide more insight about time interpolation methodologies.

2.1 Time–to–Amplitude Converter (TAC)

TAC is an analog solution for time interpolation where the time interval T_m is converted into a voltage amplitude V through charging/discharging a capacitor by a constant current I for the duration of T_m. Subsequently, V is converted to a digital binary word using a conventional high-speed analog-to-digital (A/D) converter. With the TAC method, the time interval to be measured is linearly dependent on the voltage amplitude across the charging/discharging capacitor. The voltage V across the capacitor needs to be maintained until the corresponding binary word is obtained from the A/D converter. The conversion time of TAC method is mainly dependent on the time required by the A/D to convert V into a corresponding binary representation. Figure 2.1 depicts the TAC circuit diagram.

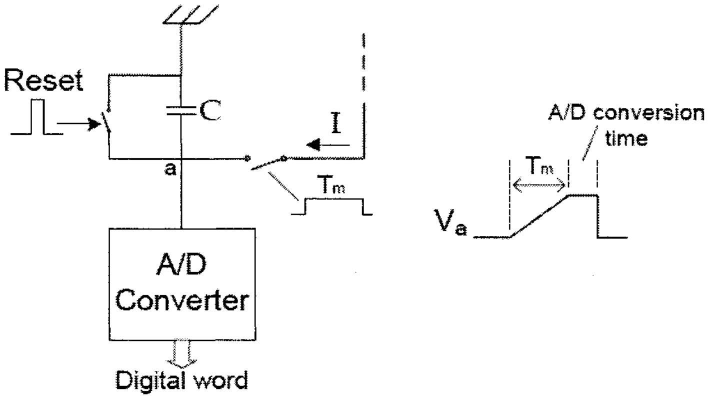


Figure 2.1: TAC circuit diagram

While the TAC method is based on relating an interval in time (T_m) to some voltage amplitude, the actual realization of the time to voltage correspondence may vary from one TAC-based circuit to the other. In some TAC methods the voltage V to be converted to binary representation is the remaining voltage across a fully pre-charged capacitor that is discharged by a constant discharging current ($I_{discharge}$) [13]. The discharging action is kicked off and terminated by the arrivals of the START and STOP pulses, respectively. This way, the interval T_m between the START and STOP pulses is made to correspond to the remaining voltage V across the capacitor through discharging principle. On the other hand, some other TAC realizations may use the voltage level V across a capacitor that is reached after being charged with some constant charging current (I_{charge}) [14]. In this case, the START pulse would trigger charging of a capacitor with I_{charge} until the arrival of the STOP pulse which opens the path, disabling any further charging of the capacitor in concern. Hence, the voltage V to represent T_m is reached as a result of charging the capacitor. Other variations may also exist, but the essential of TAC method is the same; that is, create a correspondence between the interval of time T_m and some voltage V, followed by A/D conversion.

In TAC method of time interpolation, the measurement resolution depends on the conversion resolution of the A/D converter used to convert V into a binary representation. Hence, exploiting the recent advances in the design and implementation of fast and high resolution A/D converters, high measurement resolutions for TAC-based TDC circuits may be achievable. Using TAC, the timing resolutions reported vary in the range of 10 to 100 picoseconds [3][10][11][13][14][15]. It is to note that a TAC-based TDC circuit may experience higher power consumption relative to a digital counterpart, and have larger dead time due to A/D conversion process involved.

2.2 Time Interval (TI) Stretching

Time interval stretching is another well-known analog time interval interpolation method where the small discriminating time bin is obtained by converting T_m into a much larger time interval T followed by averaging. The time interval T, given by $T = T_m + T_{str}$, is obtained by summing T_m with a stretched version of T_m by some factor k (i.e. $T_{str} = kT_m$). Subsequently, the new interval $\boldsymbol{T}$ can be accounted for using a counter clocked at some frequency $f_0 = 1 / T_0$. A counter reading of N would indicate $T = T_m + T_{str} = N\,T_0$. By simple mathematic manipulations, we get:

$$T_m = N\,T_0 / (k + 1) \tag{2.1}$$

From eq. 2.1 it is seen that T_m is given with a final resolution of $T_0 / k{+}1$.

The definition of factor k depends on the actual realization. In [11], k is given as the ratio of two constant currents, namely, I_{charge} and $I_{discharge}$, respectively. A capacitor C is charged by the charging current (I_{charge}) for the duration of T_m. Next, the voltage across C can be discharged using some small discharge current ($I_{discharge}$) such that the condition $k = I_{charge} / I_{discharge}$ is satisfied. The new time interval T is given as the total charging and discharging time intervals and is counted by a counter running with a clock signal with period T_0. Therefore, $T = T_{charge} + T_{discharge} = N\,T_0$, and with $T_m = T_{charge} = T_{discharge} / k$ eq. 2.1 is obtained. Figure 2.2 depicts the circuit schematic for a TI stretching methodology.

Another variation may find the stretch factor k as the combination of two ratios namely the current and capacitor ratios [16]. Using TI stretching a measurement resolution as low as 10ps has been reported [11]. With its high measurement resolution achievement, TI stretching methodology suffers from the long conversion time due to time stretching component kT_m.

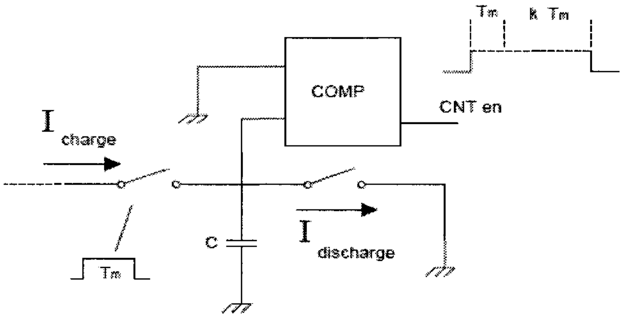


Figure 2.2: Time Interval Stretching Circuit Diagram

2.3 Delay Line as Time Digitizer

Possibly one of the simplest digital methods to achieve smaller time bins is by means of a chain of delay elements [17][18]. When driven by a digital signal s, an N-element delay chain with a propagation delay of t_d for the chain components will delay the leading edge of s by t_d as the signal propagates through the chain, providing N tap lines distant by t_d. Counting the number of stages propagated by s, the interval (T_m) is found with a time resolution of t_d. The schematic in Figure 2.3 shows the basic principle of time interval measurement using a delay chain.

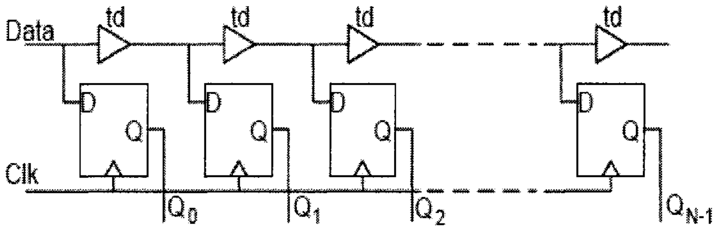


Figure 2.3: Delay line as time interpolator

The time interval between the START signal applied at the *Data* input and the STOP signal applied at the *Clk* input of the time interpolator of Figure 2.3 is measured by multiplying the delay t_d by the number of flip-flops having logic high at their Q outputs. The output Q_i of FF_i with a high value indicates that the START signal is leading the STOP signal by at least $i\ t_d$. The time interval T_m is interpreted as $t_d \leq T_m < (i+1)\ t_d$ if output Q_i is high, or as $T_m < i\ t_d$ if output Q_i is low. The dynamic range for a delay line interpolator depends on the number of delay elements used in the chain. However, the dynamic range can always be increased if a binary counter is incorporated into the system in parallel.

The actual realization of the buffer/delay elements can be achieved by different means such as two cascaded inverters, simple CMOS buffer implementation, a controlled logic element such as an AND gate with one input connected to *VCC*, a look-up-table configured as buffer, and etc. The choice of a specific implementation may depend on the requirements set by the application of concern. Regardless of which buffer realization is used, the measurement resolution using delay line interpolation is limited to the propagation delay of the logic element used as the delay component. A disadvantage of this method lies on the sensitivity to delay mismatch among identical elements of the line, due to process parameter variations and other factors such as the variation in power supply and temperature [19].

A more effective delay line interpolation is achieved by using the tap lines out of a delay-locked loop (DLL) [20][21][22]. Typical DLL circuits as the one in Figure 2.4 consist of a *phase detector* to assure the input and output signals being in phase, a *charge pump* that sources or sinks current for a duration of pulse widths at UP or DN outputs of phase detector (generating a net charge proportional to UP and DN pulse widths), a *loop filter* driven by the charge pump that integrates and filters the charge current to produce a control voltage that in turn drives the *voltage controlled delay line* (VCDL). The DLL can be locked to a fixed overall delay, with subdivisions equal to the delay of the configurable delay of a single buffer t_d. For instance, if the DLL is locked to one period of the reference clock T_{REF}, then the tap outputs would be distant by $t_d = T_{REF} / N$. The individual delays for the delay cells (current starved inverters) in the line are controlled by a control voltage increasing/decreasing the charge/discharge currents. Using a DLL for time interpolation is an effective method from the stability point of view as it is self-calibrated through the negative feedback. However, the effective time resolution is still limited to the minimum delay of a single buffer element in the VCDL.

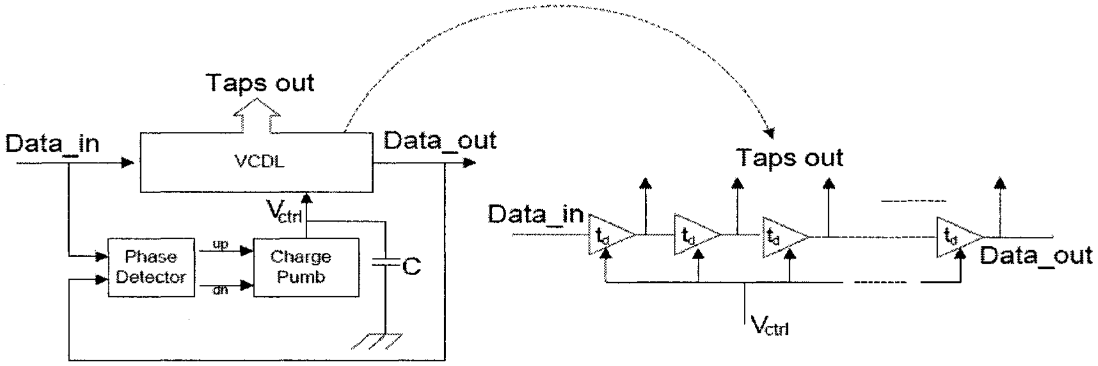


Figure 2.4: DLL circuit schematic

2.4 Carry-Ripple Multi-bit Adder as Time interpolator

An FPGA-based TDC circuit proposed in [41] uses an N-bit carry-ripple adder circuit for achieving fine time interpolation. The fine interpolation exploits the fast carry lines embedded within the FPGA fabric for propagation of the incoming data pulse.

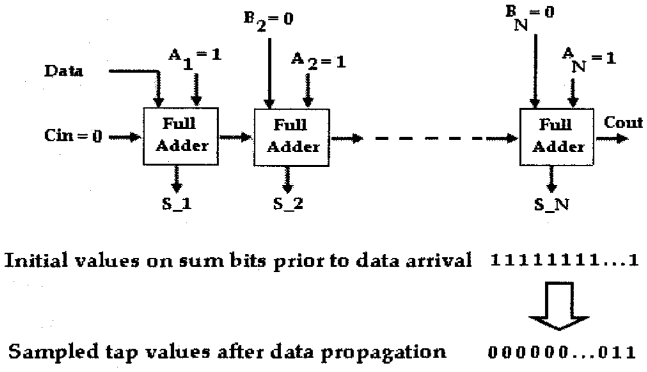

Data
$A_1=1$
$B_2=0$
$A_2=1$
$B_N=0$
$A_N=1$
Cin = 0
Full Adder
Full Adder
Full Adder
Cout
S_1
S_2
S_N
Initial values on sum bits prior to data arrival 11111111...1
Sampled tap values after data propagation 000000...011

Figure 2.5: N-bit carry ripple adder used as fine time interpolator

As shown in Figure 2.5 the adder's A-inputs are tied to logic high while the B-inputs are tied to logic low with the exception of the least significant B-input that is used as the input for the incoming pulse. The output code (sum bits) contains all logic-one. Upon arrival of the data pulse and the subsequent propagation through the full adder modules, the sum bits start to change to logic-0 forming a thermometric output code. The subsequent clock edge will sample the state of the sum output bits (thermometric code). The pulse arrival time with respect to the next rising edge of the clock is given by the number of stages or delay cells the pulse has propagated (number of sum bits changing to logic-o). Of course, the number of stages in the adder circuit should cover a propagation time that is larger than the period of the reference clock. Using this method, the RMS resolution achieved in [41] was reported to be 65.8ps and 91.5ps for the respective implementations on VIRTEX-II (XILINX) and ALTERA ACEX 1K (ALTERA) FPGA platforms. However, the TDC system shows relatively high non-linearity values that are attributed to the architecture of the FPGA platforms.

2.5 Pulse Shrinking Technique

Time interval measurement can also be achieved by means of pulse shrinking. Pulse shrinking is a technique of measuring the width W of a pulse signal through successive pulse width reduction by some constant value, say ε, until complete disappearance as the pulse propagates through the elements of the delay line. The elements in the delay line are voltage-controlled delay elements with variable rise or fall times that will shorten the width of the input pulse at output node by ε. This way the pulse width in concern is measured by a resolution equal to the width reduction value (ε). It is important that the maximum pulse width reduction achievable by the delay line is larger or at least equal to the input pulse width (W). An alternative pulse shrinking method is cyclic; that is, a single pulse shrinking element in the line achieves the constant pulse width reduction while the pulse in concern propagates the line in circular manner until it is disappeared. Both of these methods are illustrated below.

To implement the pulse shrinking delay cell, there have been several proposals by researchers in the field. Figure 2.8 shows one of the many methods used for the pulse shrinking delay cell [23].

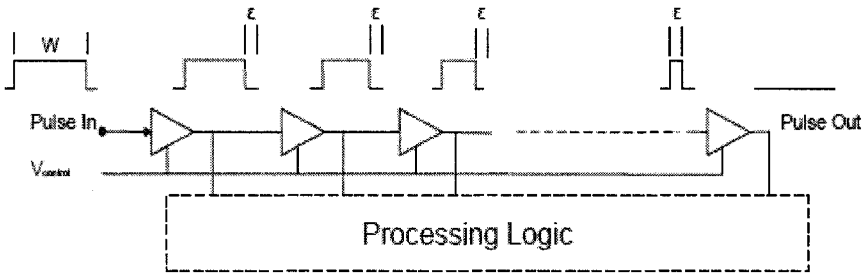


Figure 2.6: simple pulse shrinking circuit topology

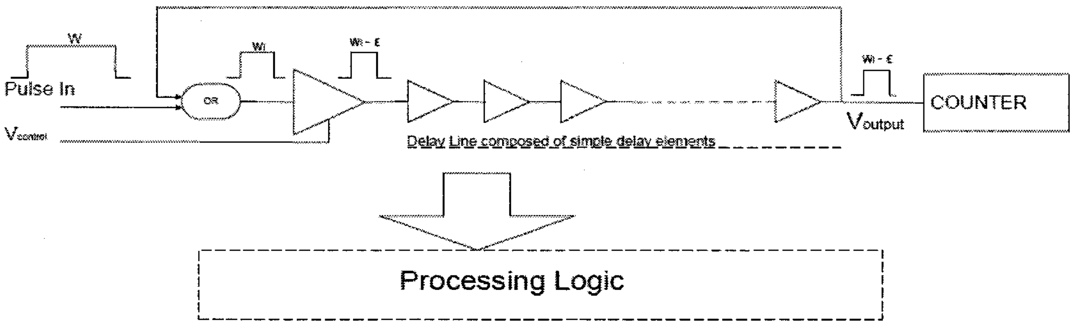


Figure 2.7: Cyclic pulse shrinking circuit topology

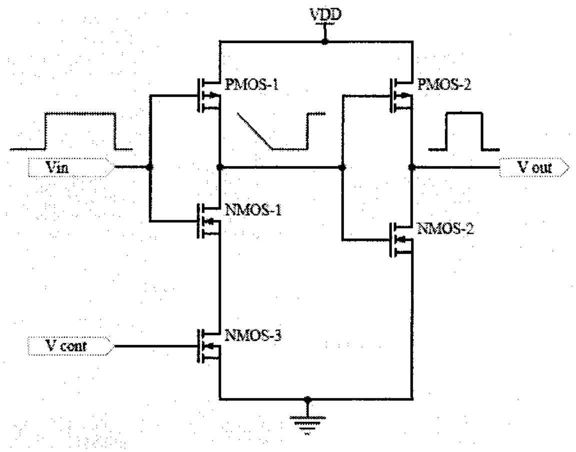


Figure 2.8: A pulse shrinking delay cell

The basic pulse shrinking delay element of Figure 2.8 consists of a CMOS inverter driven by a CMOS current-starved inverter. By controlling the discharge path of the output node in the current-starved inverter through NMOS-3, the high to low propagation delay of the inverter is made longer than the corresponding low to high propagation delay. This causes the voltage at the input node of the second inverter to be stretched longer before reaching the low threshold voltage. Consequently, the output voltage V_{out} will have a pulse width that is shorter than the corresponding pulse width of the input voltage V_{in}. The difference in pulse width of the input and the output voltages is proportional to the difference in the low-to-high and high-to-low propagation delays of the current-starved inverter [24]. The use of pulse shrinking in time interval measurement involving *START* and *STOP* incoming pulses requires two-level conversion. First the time interval to be measured (T_m) is converted into a single pulse width (W) followed by a subsequent conversion of W into binary word through tap readings of the delay line. While the measurement resolution for earlier TDC solutions based on pulse shrinking were in the range of hundreds of picoseconds [17][25], more improved TDC solutions based on cyclic pulse shrinking [23][24][26] have achieved measurement resolutions in the range of 20ps to 70ps.

2.6 The Vernier Principle

Some applications in nuclear science require a TDC with time resolution much smaller than the delay of a single buffer [2][3][4][12]. Furthermore, with the advances in technology, the trend is towards systems operating with system clocks running at much higher frequencies. These make the TDC circuits with single buffer delay time resolution almost impractical, leading towards new design methods aiming to achieve much higher time resolutions. One of the methods widely used to achieve high resolution is the Vernier principle [27]. The small timing resolution through the basic Vernier principle is achieved through a small incremental difference in periods of two periodic signals generated by two oscillators running in parallel [11]. Triggered by the START and the

STOP signals to generate periodic signals with respective periods T_1 and T_2 where $T_1 > T_2$, the oscillators are reset once the coincidence in the START and STOP edges are detected using an edge detector circuit. The time interval $T_m = t_{START} - t_{STOP}$ is measured by taking the difference in the number of cumulative periods generated by the START and STOP-*triggered* oscillators, counted by their respective counters. This way the smallest time unit that can be discriminated in finding the interval T_m is $t_r = T_1 - T_2$.

Other variations of implementing the Vernier method exploit the difference in component propagation delays. In such methods, the signals in concern are propagated through different delay lines A and B composed of *ideally identical elements*. Depending on the actual design topology used, the components in lines A and B may be buffers, latches, or other logic elements. The propagation delay of elements in line-A is slightly *larger* than those of line-B. The smallest time unit that can be discriminated is given by the delay difference $\Delta t = t_a - t_b$ with t_a and t_b being the propagation delay of elements in delay lines A and B respectively.

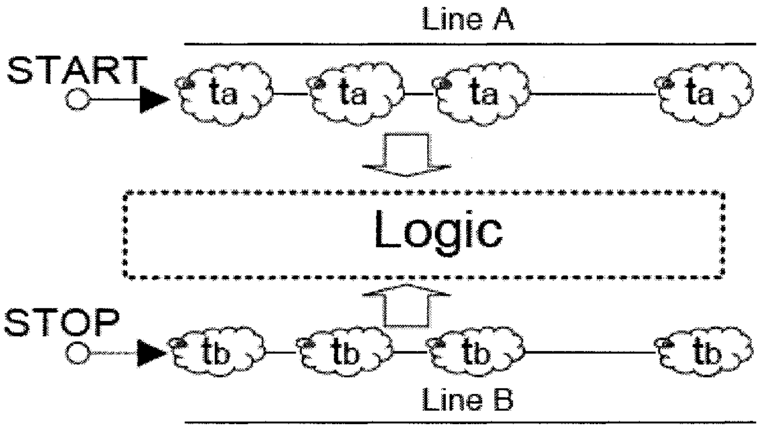


Figure 2.9: VDL principle exploiting the difference in propagation delays of logic elements

2.6.1 Buffer-Buffer Vernier Delay Line (VDL)

Conventionally, the components in delay lines A and B in VDL of Figure 2.9 are implemented using logic buffer elements. The START and STOP signals both propagate through the delay chains composed of identical buffer elements with the propagation delays of buffer elements along the START path being larger than those propagated by the STOP signal. Figure 2.10 depicts a buffer-based VDL schematic.

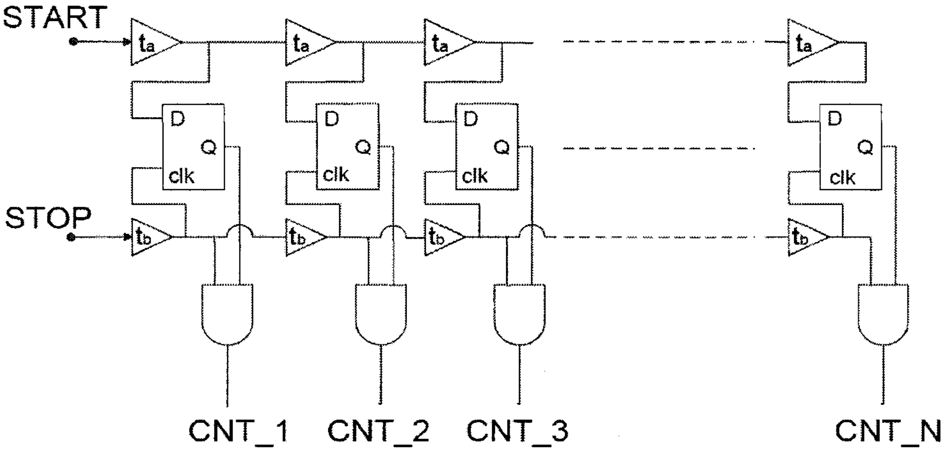


Figure 2.10: Buffer-implemented VDL used in jitter characterization circuit [28]

By design, the START signal is made to lead the STOP signal. After each stage the time difference between both the START and STOP signals is decreased by an amount Δt, detected and recorded by a corresponding *D-latch with* a logic-1 resulting when *START* is in lead from *STOP* and a logic-0 for vice versa. The VDL topology in Figure 2.10 was used to characterize the jitter in the input signal. The *counters* count the number of times (for several test cases in order to be able to get enough data for averaging) the *START* signal leads the *STOP* signal with a delay difference set by its position (i.e. at position K, the time difference is $K\ \Delta t$). This is to say that CNT_1 will increment counter_1 that keeps track of the number of times the START signal *leads* the STOP signal by, at least, a delay greater than Δt and counter_2 incremented by CNT_2 counts the number of times the START signal *leads* STOP by, at least, 2 Δt and so forth. Using the VDL interpolation topology, the interval T_m is found with a sub-gate timing resolution of $\Delta t = t_a - t_b$, where the value of Δt has been reported as low as 30 ps [27]. The major disadvantage of this topology, however, may be the increased number of counters needed and the sensitivity of the delay lines to mismatch in propagation delays among the identical components of the lines.

2.6.2 Buffer-Latch Vernier Delay Line (VDL)

An alternative implementation of the VDL method can be achieved by utilizing a chain of buffer elements in conjunction with a chain of latch components [29]. In this topology, the START signal that is in lead from the STOP signal is propagated through the chain of latch components followed by the STOP signal making its way through the buffer elements. The STOP pulse disables the CLK inputs of the latch components as it reaches a particular stage, attempting to catch the START signal. At some stage *k*, the STOP signal eventually catches the START signal by disabling the CLK input of the latch component before the START signal can reach the D-input of the latch. In this case, the outputs Q of all first *k* stages are high. The initial time difference between the two edges of START and STOP signals (The interval to be measured T_m) is governed by:

$$k\, t_L = T_m + k\, t_B \tag{2.2}$$

And subsequently T_m is given by:

$$T_m = k\, (t_L - t_B) \tag{2.3}$$

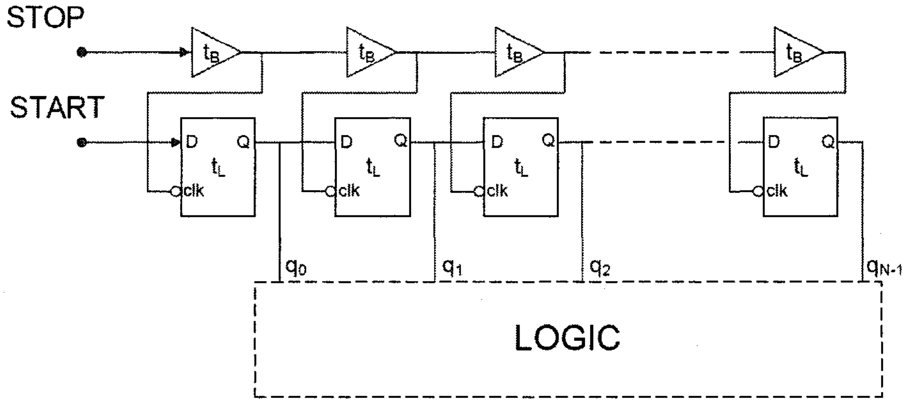


Figure 2.11: VDL implemented by Latch and Buffer lines

From eq. 2.2 and eq. 2.3 we see that a sub-gate timing resolution of $\Delta t = t_L - t_B$ is achieved. An FPGA implementation of this technique with a single shot timing resolution of 200 ps was reported in [29].

2.7 Multiple Sampling Techniques

2.7.1 Delay Locked Loop (DLL) and Adjustable RC line

The time resolution of a TDC circuit that uses the tap lines of a DLL for time interpolation is limited by the basic cell delay $t_d = T_{REF} / N$ with T_{REF} as the reference clock period and N as the number of tap lines of DLL. Some interpolation scheme [22][30] further improves this limited resolution by means of multiple phase shifted sampling. In this scheme, a DLL interpolates the reference clock period into increments of $t_d = T_{REF} / N$ and an M-stage RC delay line provides the equidistant finer time bins for sampling. The block diagram of this scheme is shown in Figure 2.12. The time interval measurement is based on time stamping of the arrival time of the hit signals.

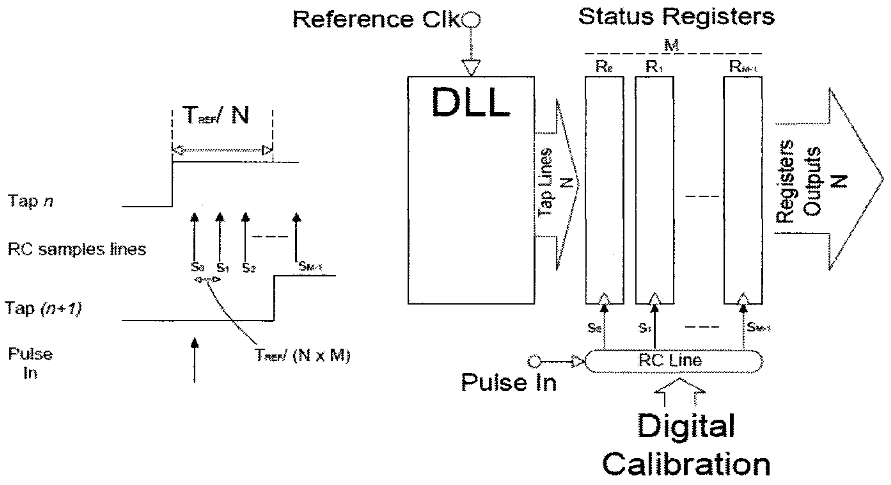


Figure 2.12: Time interpolator based on a DLL and an RC delay line

The arrival time of an incoming pulse (hit signal) is measured as follows: Upon arrival of the hit signal, the states of the DLL tap lines are sampled by the hit signal and its *M-1* delayed versions. Of all, if the first *m* status registers have the same values for the corresponding tap lines of the DLL (say logic-1 at first *n* tap lines and logic-0 for the rest), hence a high-to-low transition from tap line *n* to *n+1*, no such transition may be observed on the corresponding tap lines of the remaining *M-m* status registers starting with the status register *m*. This is to say that by sampling the states of the tap lines in

DLL for M times at an interval $\delta t = t_d/M$, the tap lines of DLL has not changed since the arrival of the hit signal for m samples. The arrival time of the hit signal, hence, can be induced as the time elapsed from the reference clock edge ($t_{hit} = nt_d + (M\text{-}m)\ \delta t$). Therefore, the small time bin achieved is the sampling interval $\delta t = T_{REF}/\ (N\ X\ M)$.

While the RC delay line can be implemented using both active and passive elements, in [22], passive elements were chosen due to their robustness to temperature and supply variations. However, the parasitic resistance and capacitance in passive elements cause variable delay along the line requiring calibration. Using this interpolation technique, an RMS resolution of less than 25 ps was reported in [22].

2.7.2 Array of Time Samplers

An alternative implementation of the time interpolation technique based on multiple sampling can be achieved using an array of time samplers [18]. In this scheme, the basic gate delay resolution obtained by the time sampler described in section 2.3 is further broken into M smaller units through multiple sampling. Figure 2.13 depicts the block diagram of the topology [18]. Similar to the interpolation scheme described in previous section, the basic principle of this topology as well lies on several phase-shifted sampling. A particular time event can be caught by several time samplers with shifted clocks.

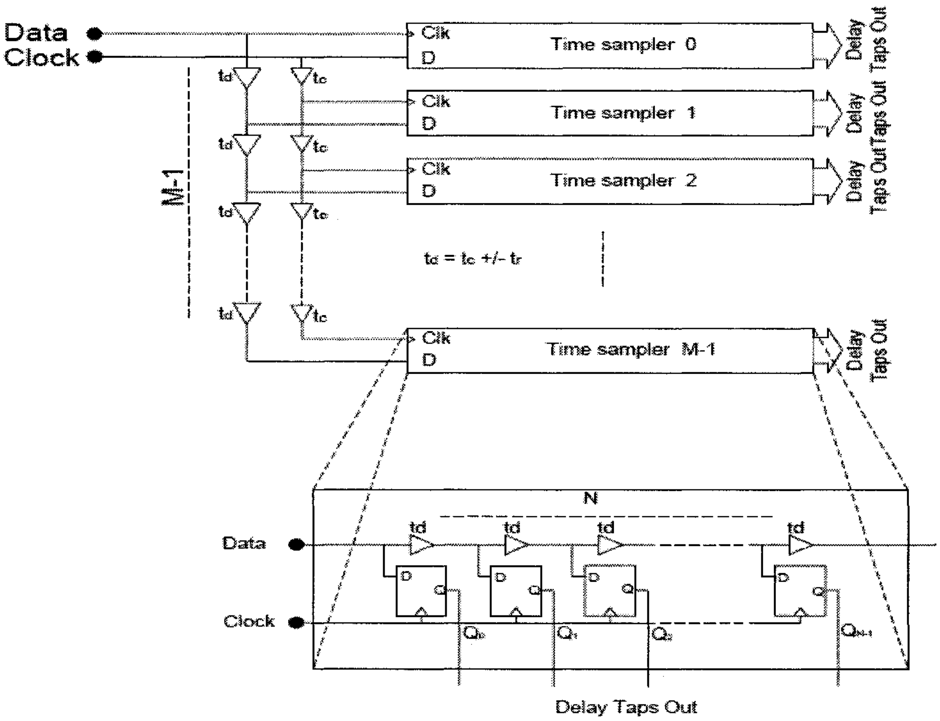


Figure 2.13: Time interpolation using multiple time sampling

The interval T_m between the START (data) and STOP (clock) signals using a single time sampler is found with basic gate delay resolution by the number of stages that the START signal has propagated before it is caught by the STOP signal. By adding more time samplers operating with shifted clock and data signals, T_m can be described in terms of both the number of time samplers and the number of stages that the START signal has propagated. That is, if the start signal has made its way to the i^{th} stage of first j time samplers before it is caught, the interval T_m is described as:

$$T_m = (i+j) \bullet \Delta t_r \tag{2.4}$$

The phase shift amount Δt_r is given by $\Delta t_r = t_d / M = t_d - t_c$ with t_d and t_c being the delay amount along the data and clock lines of two consecutive time samplers. In [18] an ASIC implementation of a TDC architecture based on four 10-stage time samplers (N=10, M= 4) was reported with a conversion time of less than 2.5 ns and a measurement resolution of 62.5ps at clock speed of 400 MHz.

2.8 Array of DLL

The limited single gate delay resolution (t_d) of a delay line interpolator utilizing a delayed locked loop (DLL) can be improved if several DLL circuits are placed in an array topology (Figure 2.14) with a small offset between consecutive DLL inputs [31]. Here the clock period is broken into N x M time bins with magnitude of t_d/M. The small phase shift value between consecutive DLL structures is achieved by another M-stage DLL circuit ($M < N$ and therefore, $t_{dn} < t_{dm}$) driving the N-stage DLL input lines. The desired time bin to be achieved is $t_{bin} = t_{dm} - t_{dn}$ with t_{dm} and t_{dn} as the basic gate delay for delay cells in M-DLL and N-DLL respectively. With t_{dn} and t_{dm} replaced by their respective values of T_{REF}/N and T_{REF}/M, and expressing t_{bin} as a fractional part of t_{dn}, a relation between M and N is established as $M = N \, x \, F/(F+1)$ with $F > 1$.

The time interval measurement is based on time stamping. That is upon arrival of hit signals (incoming pulses), the state of the DLL tap lines in the array are sampled and stored for processing. The interval between two hit signals is then calculated by subtracting one arrival time from the other. Using the topology in Figure 2.14, an RMS resolution of 34.3 ps, a dynamic range of 3.2 us, and an LSB (time bin) of 89.3 ps have been obtained with a reference clock frequency of 80 MHz [32]. A major drawback of this scheme is cited as its inability to produce number of tap readings that are power of 2, leading to the requirement of a special and more complex encoding scheme [31].

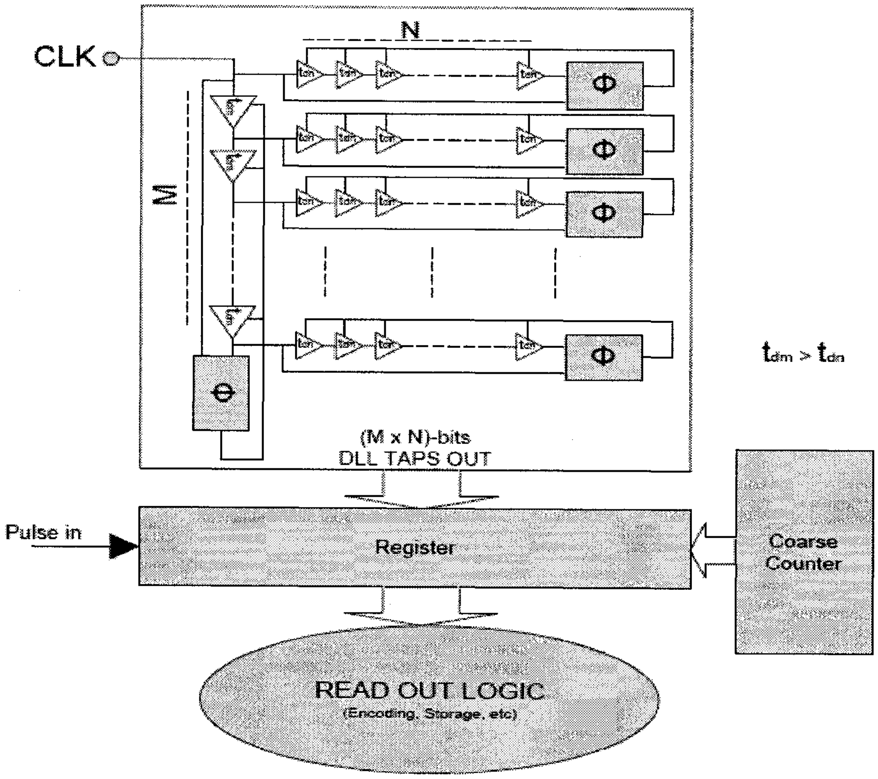


Figure 2.14: TDC circuit diagram based on a DLL array

2.9 Chapter Summary

This chapter presented several time interpolation techniques achieved in analog and digital domains. The various time interpolation techniques tend to achieve high measurement resolution using different design methodologies. The TAC and TIS are the two analog interpolation techniques widely used in TDC circuits. On the digital side, most of the solutions discussed use delay lines for delaying of the propagating input pulses. With an insight of the various interpolation techniques discussed in this chapter, chapter 3 will touch upon important performance metrics related to TDC circuits.

CHAPTER 3 - PERFORMANCE ANALYSIS IN TDC CIRCUITS

In the field of electronics, an electronic device is often associated with important performance metrics that are vital to characterize for the operation of the unit as a single or as an embedded component within a broader system. Circuit designers usually characterize an electronic circuit with nominal values for the associated performance metrics with some degrees of tolerance. While the nominal values are what circuit designers aim the device to operate with, the actual circuit performance may exhibit some deviation in metric values from their nominal ones. This deviation in metric values may be caused by inevitable systematic errors such as skew in clock line of a clocked system that is inherent in the design, or by some extrinsic errors caused by environmental factors such as variations in temperature and supply voltage limiting the circuit operation under nominal metric values.

In applications that involve some sort of data conversion before the final output is processed, accuracy of conversion plays an important role in the overall behaviour of the device. This includes any system designed to process an analog quantity in digital domain, that require data converter circuits for input and output data using the respective Analog-to-Digital Converter (ADC) and Digital-to-Analog Converter (DAC) circuits. A TDC circuit also converts the time that is an analog quantity into a digital binary word. The very first process in a data converter circuit is the *quantization* step where mapping between the various amplitude values of the analog quantity at the input is made to a corresponding digital level value represented by a binary word. The length of the binary word depends on the resolution (number of distinct analog levels corresponding to the different digital words) of the converter. If a data converter can break the input analog quantity into 2^N distinct amplitude values, then the converter is said to have N-bit resolution with a digital word containing N bits.

Quantizing a value introduces an error that is proportional to the conversion step. The various conversion steps are discriminated by the Least Significant Bit (*LSB*) of the digital binary word representing a quantized level. A transition in the *LSB* of a binary word corresponding to some analog amplitude causes a move to the next digital level corresponding to a subsequent different analog amplitude value. Depending on the input analog quantity (signed or unsigned), the dynamic range (from the smallest or minimum value to the maximum value) of a converter consists of 2^N digital levels of equal magnitude referred as *LSB*. Hence, by default, an ideal data converter introduces quantization errors that vary between $\pm LSB/2$ to a maximum of one *LSB* because a single digital word serves to represent a range of valid analog input values.

Similar to ADC and DAC, a TDC circuit is also characterized by the LSB (smallest time increment that can be discriminated) and thus suffers from errors associated with the quantization step. The accuracy of measurement in TDC circuits is evaluated from the difference between the expected and the actual transfer response of the system. The ideal transfer curve of a TDC, as shown in Figure 3.1, is a monotonically increasing staircase-shape response where distinct binary words are associated to a range of valid analog time values falling within the range of values contained by the resolution. The actual response of a system, however, may deviate from the ideal response. The following section describes some of the terms commonly used to characterize and evaluate a converter's response.

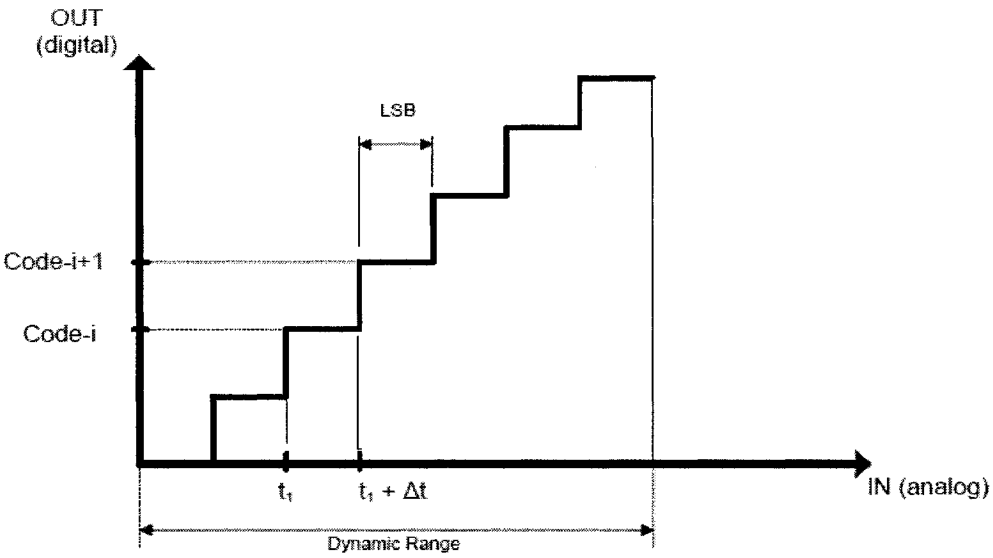


Figure 3.1: Input-output transfer response for an ideal converter

3.1 Performance Parameters

While the ideal quantization error with a maximum magnitude of single LSB is expected in a TDC circuit, other factors may affect the system response from the expected values. To characterize the actual system response, the system needs to be evaluated for the non-linearity errors that are defined as *the deviation of the actual response from the ideal staircase-shape curve* described in terms of differential and integral non-linearity characteristics.

3.1.1 Differential Non-Linearity (DNL)

If the actual transfer response of a converter (TDC) is obtained using experimental results, the curve may look as sketched in Figure 3.2. It is seen that at some parts of the curve, the staircase-shape steps have deviated from the ideal single *LSB* value, causing a non-uniform staircase curve with step widths either shorter or longer than the expected value of one *LSB*. This variation of the output time bin size from the ideal value of a single *LSB* is referred to as Differential non-linearity. The less the *DNL* value the more monotonic the system's response with no missing code, and consequently the less the measurement error. As pointed out in [12], depending on the type of converter (ADC or TDC), the definitions for various metrics may differ. The *DNL* of a time bin in a TDC circuit can be evaluated by eq. 3.1 where D_i is the cumulative delay from the reference point to tap_i [12].

$$DNL = [D_{i+1} - D_i - LSB]/LSB, \quad i=0...N-1 \tag{3.1}$$

3.1.2 Integral Non-Linearity (INL)

Another measure to evaluate a converter's performance with respect to the ideal response is to characterize the input/output characteristic with a straight line of ideal gain. The deviation of the transition points on the curve from the ideal transition points represented by a straight line is referred as Integral non-linearity and commonly it is defined as a maximum value. *INL* represents the cumulative *DNL* error of a converter [33].

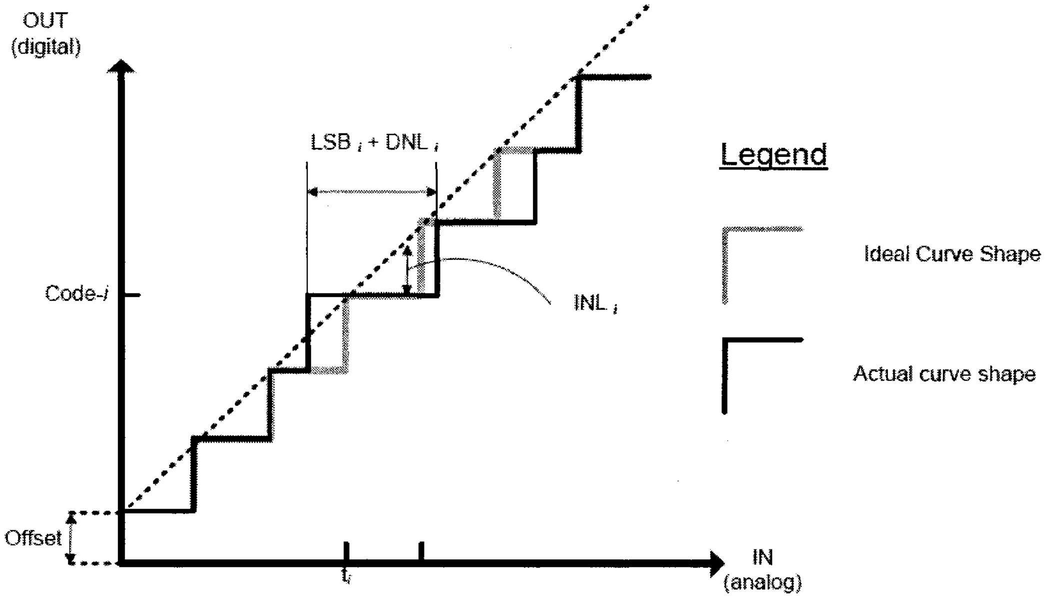


Figure 3.2: Transfer curve for a non-ideal converter

The magnitude of the *INL* error depends on the actual position of the straight line chosen. Two commonly used methods of choosing a straight line is the *best fit* and *the end point* straight lines. A *best fit straight line* gives a good approximation of the converter's transfer function with no actual position of the line defined, providing information about the gain, offset and position of the actual transfer function. It is said to be the true representation of a converter's linearity. On the other hand, *the end point straight line* has a precise line position by passing through the converter's transfer function end points. In general, the best fit straight line method is preferred over the end point straight line because it produces better results [34]. In [12] the *INL* value of a tap output in the context of a TDC has been defined as:

$$INL_i = [\, D_i - i \; x \; LSB - D_{offset} \,] \,/\, LSB \;\;,\; i = 0,\ 1,\ \ldots N\text{-}1 \qquad (3.2)$$

In eq. 3.2 D_i is defined as for *DNL* in above, and D_{offset} is the delay corresponding to zero time analog input.

Other terms such as the *gain* and *offset* errors often used in the context of ADC and DAC circuits may not have any significance as far as a TDC is concerned. The *offset* error is the intercept of the transfer curve corresponding to zero time input and the exact value depends on the straight line used to analyze the *INL*. Similarly the *gain* error is defined as the deviation of the slope of the straight line used from the ideal line and may not have any significance in the context of a TDC.

3.2 Error Sources

The main source of error in measurement results produced by a converter circuit is the *quantization* error caused by the non-linear response of the converter during the quantization process of the analog input. This section will focus on other sources of errors that cause the deviations of the converter response from the ideal one.

3.2.1 Reference clock jitter

One of the important sources of error in synchronous systems is the jitter present in the reference clock signal. Signal jitter is the variation of the edges of a periodic signal such as a digital clock from their ideal expected locations. In a jittery periodic signal, the rising/falling edges of two adjacent periods may either come earlier or later than the expected point in time. A synchronous system designed to function with an ideal clock signal may deviate from its expected response if the actual clock signal has fluctuations in its period from one cycle to another.

In a TDC circuit, the time interval measurement often depends on a reference clock signal whose edge is used as the STOP edge for the data pulse (in case of a START/STOP time measurement principle) or as an observation time window in case of a multi-hit. The jitter in the clock period affects the measurement results causing unexpected addition or reduction of the actual time interval to be measured due to displacement of the observation time window or simply the STOP edge. The jitter in the clock signal can be thought of an added random noise to the conversion function.

3.2.2 Skew in clock paths

In any system where the flow of data along the data path is synchronized by a periodic clock signal, the frequency of system operation may be affected if the clock paths from the clock signal source to the destination points (clocked elements of the system such as registers) have non-identical delays. The matter is worse if the operation of the system depends on the arrival of the clock edges at various points of the system at particular expected points in time. The difference in clock path delays from the clock source to two adjacent clocked elements in a system is referred as *clock skew*. While the official definition of clock skew may be described using two sequentially adjacent registers with some combination logic in between (see Figure 3.3), clock skew can exist between any two registers in a data path and affect expected system behaviour.

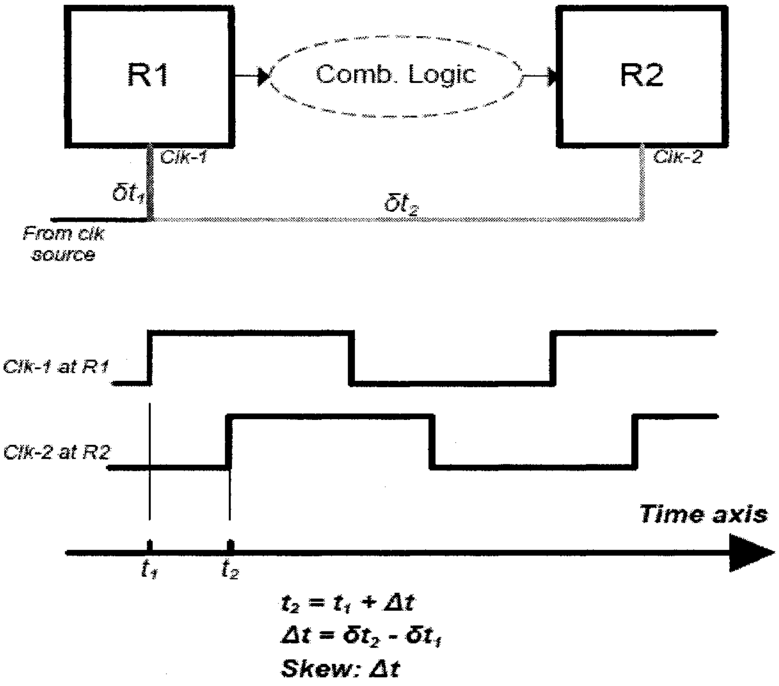


Figure 3.3: Clock skew

Clock skew can either be *positive* or *negative* depending on the register of reference. A clock skew is said to be *positive* if the arrival of the clock signal at the final register of a data path leads the arrival of the clock signal at the initial register of the same data path. However, the inverse scenario with the clock signal arriving at the final register after arrival at the initial register of the same data path is said to be *negative skew*. Some of the causes of clock skew include difference in lengths of lines from clock source to various destination points, difference in delays of buffers within the clock distribution network, difference in passive interconnect parameters, and the difference in active device parameters such as MOS threshold voltages and channel mobility. Depending on the application, clock skew may affect system performance in limiting the overall system clock frequency or the actual behaviour of the system. For further discussion of clock distribution and clock skew the reader is urged to refer to [35]. For the TDC circuit under study, clock skew among various lines of the delay matrices introduces some inaccuracy in measurements.

3.2.3 Variation in delays

Many TDC circuits use interpolators that are based on delay lines composed of identical delay cells. The ideal behaviour of these interpolators assumes identical delays for all delay elements. However, the actual propagation delays are non-identical due to factors such as variations in the process parameters. The non-identical delays among these cells introduce inaccuracy in the measurement results that contribute to the INL and DNL non-linearity errors. In addition to process parameter deviations, other environmental factors such as variations in the power supply, and temperature may also affect component delays causing deviation in the system response from the expected one.

The generally accepted errors and their sources as described in this chapter affect the performance of the TDC circuit under study. Further analysis of the various sources of errors that are specific to the current design is discussed in the analysis and discussion part of this report in chapters 4 and 5.

In the next chapter, the proposed TDC circuit is discussed in details. The TDC is based on matrices of delay cells, aiming to improve the timing overhead while achieving relatively high measurement resolution. The chapter will encompass all details and issues regarding the design, implementation, and performance limitations.

CHAPTER 4 - PROPOSED TDC ARCHITECTURE

4.1 Introduction

Given the importance of time interval measurement in various fields of science, this research project consists of a new improved method of time interpolation used in TDC circuits. The proposed time interpolation technique tends to provide high measurement resolution while improving the time overhead that is a bottleneck in many designs by minimizing the dead time and the overall measurement time required for a measurement phase. The TDC under study also aims to achieve multi-interval time measurement by implementing the ability to detect and process more than one sequentially incoming data pulse. In order to exploit the recent development in the programmable logic devices such as the Field Programmable Gate Array (FPGA) and explore possible pros and cons of designs targeting the latter, the TDC system is implemented on a low-cost FPGA platform.

This chapter covers all the various steps taken in realizing the TDC architecture we propose. First, the initial project objectives are restated in the form of requirements. Next, design specifications are proposed based on the stated requirements, and subsequently, a detailed design and implementation description for each component of the circuit is made. The chapter will also discuss the calibration schemes required for the estimation of delay of the logic elements used in the design.

4.2 Presentation of The TDC Architecture

4.2.1 Architecture Requirements

The primary aim of the project in realizing a TDC circuit was to overcome some of the performance limitations by earlier TDC solutions. In particular, initial project objectives include reduced time overhead, the ability to measure several time intervals in a single measurement phase, improved single shot measurement resolution, and the realization on a low-cost programmable device such as an FPGA.

The time overhead in TDC designs is mainly due to the time lost in between measurement phases (dead time) and the extra time that is required to obtain measurement data (measurement or conversion time). Thus, in an effort to decrease the time overhead, the TDC under study should consider reducing the dead time and the measurement time associated with the circuit. Similarly, in order to be able to measure more than one time interval per phase of measurement, the TDC circuit should consider a topology that achieves multi-hit capability. That is, the circuit should be able to detect and handle more than one timing event in a sequential manner with respect to some common reference point.

As for the improved single shot resolution, a particular time interpolation technique should be used that slices the reference clock period into desired small time increments. Since the project targets FPGA platforms, feasible single shot resolutions are initially set to be around 100 to 200 picoseconds in order to be immune from metastability effects of FF/latch elements [37] . And lastly, a particular low-cost FGPA platform is to be chosen for implementing the circuit in order to demonstrate design feasibility. The choice of the FPGA platform should be made mainly based on the criteria of availability, low-cost, user-friendliness, and the availability of the resources needed.

4.2.2 Proposed Design Specifications

With the design goals described in the previous section as requirements, the design specifications are defined as follows:

Measurement Resolution: Inspired by the design work in [29], the acceptable measurement resolution of 100 to 200 picoseconds is achievable by the VDL method used by the former. Hence, the time interpolation method chosen for the current design is to be based on the VDL. However, the VDL technique as used in [29] suffers from large measurement time, particularly when the VDL line is composed of a large number of stages.

Measurement time reduction: The small time increment using a VDL line is achieved at the expense of some extra time required by the line in order to propagate an incoming pulse throughout all its stages. Each stage of a VDL line provides a differential time bin that is given by the difference in delay of the elements of both lines. However, the cost of each differential time bin is the propagation of the pulse through the element with the largest delay in the line. In other words, for an M-stage VDL line, in order to break an interval T into N smaller time bins of magnitude t_r ($t_r = t_a - t_b$ with t_a and t_b being the delay of logic elements in both lines of VDL and $t_a > t_b$), the extra time wasted is given by $(M\, t_a - M\, t_r) = M\, t_b$. If the interval of concern is large enough and requires a VDL line with many stages, the extra overhead may become a bottleneck of the design. Therefore, while considering the VDL technique used in [29], a better topology aiming to reduce the time overhead needs to be introduced.

One way to reduce the measurement time is to reduce the number of stages in the VDL line which is possible if the interval to be sliced off into smaller time bins can be small. The interval to be interpolated can become small if a two-level timing resolution is achieved. Instead of directly slicing the reference clock period into smallest time bins (t_r), the reference clock period should be broken into relatively larger time steps of magnitude τ_1, with subsequent interpolation of those larger time steps into smaller Vernier time bins τ_2. This way an interval to be measured is described by two units of time, by analogy to distance measurement using a ruler equipped with units of millimeter and centimeter. By larger time bin propagation, pulse propagation through many costly Vernier stages is avoided. This will significantly reduce the number of VDL stages required, which in turn considerably reduces the extra measurement time. The two-level time interpolation diagram is shown in Figure 4.1.

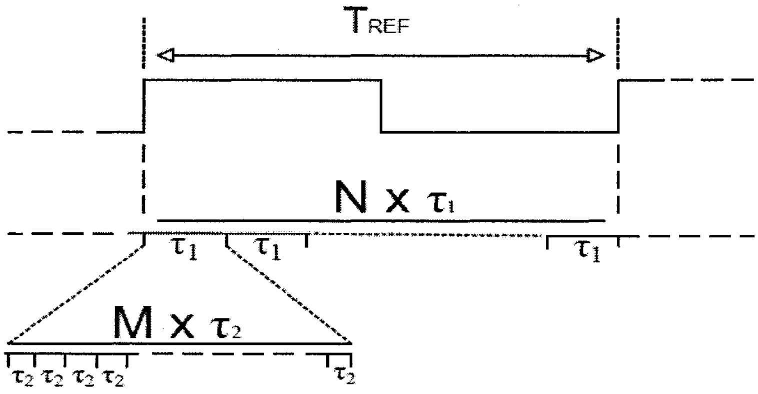


Figure 4.1: Two-level time interpolation

Dead time reduction: To reduce the dead time (time wasted between measurement phases), a scheme or algorithm is needed to make the pulse detection and data acquisition continuous, that is the processing of detected data events should not be a setback for further acquisition of data. To achieve the continuous detection and processing of measurement data, the current TDC work uses *parallelism*. The incorporation of multiple out of phase time interpolators operating in parallel will definitely improve the overall dead time of the circuit. The aim is to use two time interpolators in both phases of the clock. This way, the detection of measurement data is possible at any time.

Low-cost FPGA platform: The FPGA platform used for realizing the TDC circuit is based on the low-cost XS31000 FPGA from the XILINX SPARTAN^{TM}-3 family of FPGAs. The XC3S1000 SPARTAN^{TM}-3 FPGA has about one million system gates corresponding to 17280 logic cells and is equipped with other useful primitive resources such as digital clock managers for clock generation, block RAM units for temporary storage, and optimized built-in multiplier units for possible computations. The platform offers a user-friendly FPGA environment containing the necessary components needed for carrying out projects such as the current project.

4.2.3 The Time Interpolator

The core part of the design is the time interpolator circuit to provide time steps that are much smaller than the reference clock period. This section discusses details of the various design components, including the time interpolator.

Topology: As stated in the specifications, the proposed time interpolator is based on the Vernier principle. The Time interpolator is a matrix of delay cells composed of logic buffers and latch elements. Alternatively the matrix of delay cells can be seen as an array of buffer-latch Vernier Delay Lines (VDL) described in earlier section with row-wise interconnection between successive rows achieved through the first delay cells of each line and a common reference clock line shared by all matrix rows. The proposed matrix topology is shown in Figure 4.2.

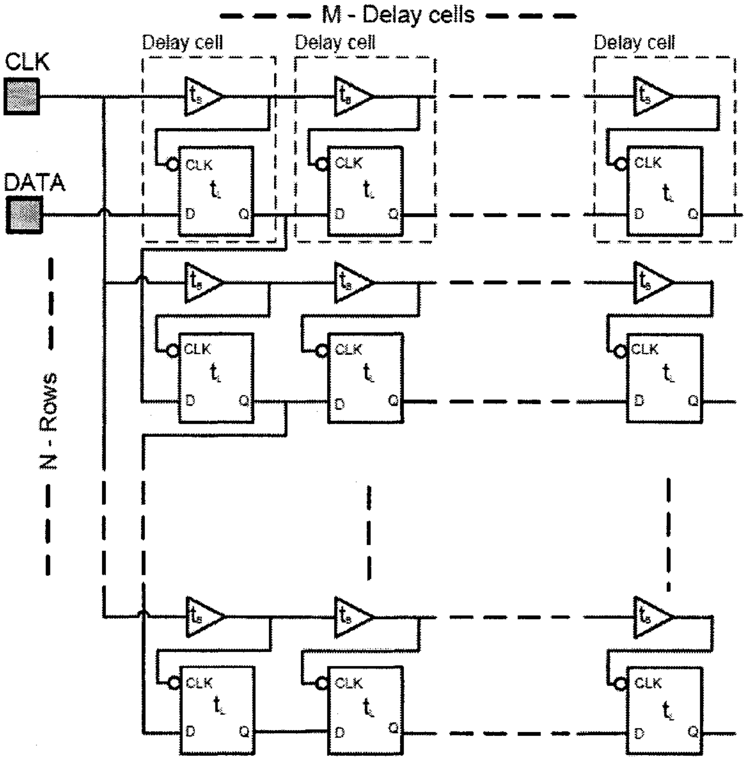


Figure 4.2: Delay matrix topology

The structure of the delay matrix as depicted in Figure 4.2 measures the distance in time between the rising edges of the data and the following clock signals applied at the respective data and clock inputs of the matrix. This distance defines the *event interval* (T_e) for an incoming pulse/event which is the interval from the incoming data edge to the subsequent reference clock edge. T_e is broken into two-level time bins namely $\tau_Y = t_L$ and $\tau_X = t_L - t_B$ with t_B and t_L as the propagation delay of the buffer and latch elements in the delay matrix, *assuming that the propagation delay across the D-latch element is t_L along vertical and horizontal directions*. The vertical interconnection provides a propagation path with larger time bin τ_Y whereas the single shot resolution τ_X is achieved by horizontal propagation throughout the Vernier delay lines. Once the event interval T_e for an incoming pulse is found, it can be used in combination with the coarse count value from the binary counter to determine the arrival time of the pulse in concern with respect to some reference point.

Matrix Operation: Initially, all latch elements of the matrix are transparent with a logic low value at their Q outputs. Similar to a VDL, a data pulse arriving through the data input continues to propagate throughout the delay cells in the matrix as long as the latch elements of the delay cells remain transparent. Upon arrival of the rising edge of the clock signal at the clock input of latch elements any further propagation of the data pulse is blocked. The event interval (T_e) corresponding to the propagating data pulse is measured by counting the number of rows (n) and the number of delay cells (m) through which it propagates in vertical and horizontal directions. A data pulse whose arrival time from the next rising edge is distant by more than τ_Y will propagate vertically from one row to the next. Thus with each vertical pulse propagation the remaining sub-interval is reduced by τ_Y. The vertical propagation continues until the sub-interval is less than τ_Y at which point the horizontal propagation comes into play.

Measurement Principle: Despite the START/STOP principle of operation of the delay matrix, time interval measurement in the TDC circuit is based on time stamping of the pulse arrival time. Depicted in Figure 4.3, the interval of interest is the distance between the arrival times of the incoming data pulses.

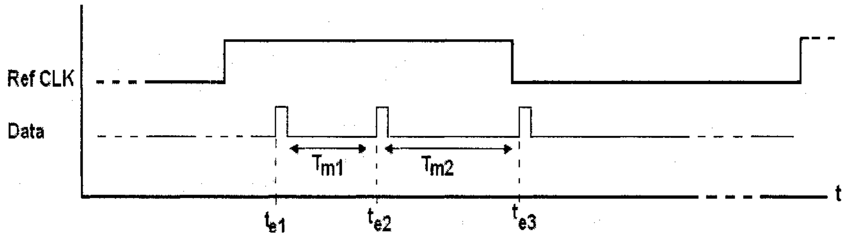
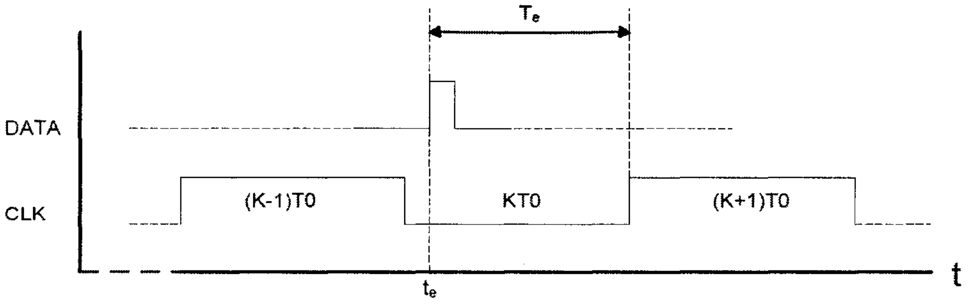


Figure 4.3: Interval of Interest

The matrix topology in combination with a coarse counter provides the means for detecting more than one data pulse in a single measurement phase. By time stamping the arrival times for several sequentially incoming pulses and subsequently taking the difference in the measured arrival times the time intervals of interest are found. For instance, the interval between data pulses e_i and e_{i+1} can be calculated using their respective arrival times t_i and t_{i+1}, that is: $T_i = t_{i+1} - t_i$. As illustrated in Figure 4.4 and given by eq. 4.1, the arrival time t_e for an incoming event ***e*** in the active phase K is computed using the interval T_e and the coarse count value $(K+1)$ corresponding to the next phase of the reference clock. T_e is determined by reading the (n, m) coordinates of the delay matrix.

$$t_e = (k+1)T_0 - T_e = (k+1)T_0 - (n\,\tau_Y + m\,\tau_X) \tag{4.1}$$



$$t_e = (K+1)TO - \underbrace{(n\tau_Y + m\tau_X)}_{T_e}$$

Figure 4.4: Pulse arrival time

Matrix Size: The number of delay cells in the matrix is related to the observation time window (T_{OBS}) used for pulse detection. *T_{OBS} is defined as the interval of time for which the interpolator (delay matrix) is active for detecting and propagating the incoming data pulses throughout its delay cells.* The primary use of the delay matrix is to provide smaller time bins for the duration of T_{OBS} in parallel with the coarse counter that counts the number of clock periods. Hence, an important consideration while choosing the matrix size is that the delay to be interpolated by the matrix should be at least equal or larger than the T_{OBS} in order to have smaller time bins for the whole duration of $T_{OBS.}$ Therefore, as shown in Figure 4.5, the delay matrix needs to break the observation time window or the reference clock period into a minimum of N smaller time bins, N being a positive integer number.

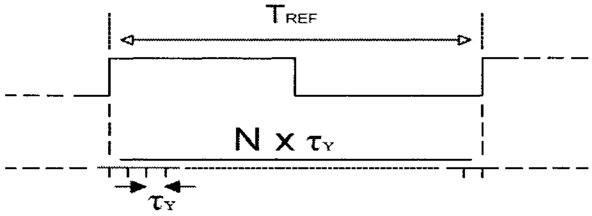


Figure 4.5: Interpolation of T_{REF} into N smaller time bins of magnitude τ_Y

The accuracy of the time measurement may increase if the smaller time increments of magnitude τ_Y can be further broken into much smaller time bins of magnitude $\tau_{X.}$ Thus further interpolation of τ_Y should also be achieved, as shown in Figure 4.6 in below.

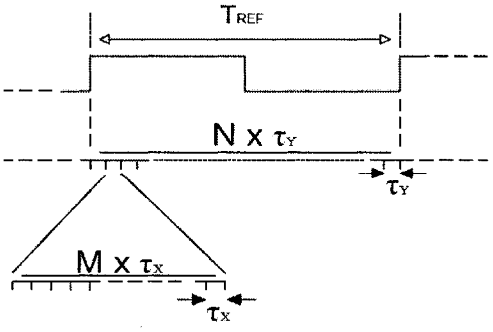


Figure 4.6: Interpolation of T_{REF} into smaller time bins of magnitudes τ_Y and τ_X

Using the topology of the matrix of delay cells, the first level time interpolation with time increments of magnitude τ_Y is achieved by the vertical propagation path provided through the inter-line interconnections. Whenever a data pulse propagates to a subsequent row of the matrix, the time distance between the data and clock edges is further decreased by a single latch delay. Therefore, the time increment τ_Y is given by the vertical single latch delay. Moreover, as the maximum data-clock edge distance in a measurement phase is T_{OBS}, the number of lines in the matrix relates to the number of time bins of magnitude τ_Y in T_{OBS}. Thus, the number of lines N in the matrix should obey:

$$N \geq T_{OBS} / \tau_Y \tag{4.2}$$

As the latch elements of a delay cell are negative level sensitive, the observation time window (T_{OBS}) in the matrix corresponds to half period of the reference clock. That is:

$$T_{OBS} = T_{REF} / 2 \tag{4.3}$$

Therefore, from eq. 4.2 and eq. 4.3, the number N of matrix lines is given by:

$$N \geq T_{REF} / 2\ \tau_Y \tag{4.4}$$

With N given by eq. 4.4, it should be noted that T_{REF} itself has to obey some restrictions that depends on the number of delay cells per row of the matrix. This restriction will be discussed more once the conditions for the number M of delay cells per row have been presented.

The smallest time increment of magnitude τ_Y is achieved by the horizontal propagation along a row of the matrix that is nothing but a buffer-latch implementation of the Vernier delay line (VDL). As discussed in section 2.6.2, in a buffer-latch VDL topology, the data (START) and clock (STOP) pulses propagate through the respective latch and buffer chains with the propagation delay (t_L) of elements in the latch chain slightly larger than the propagation delay (t_B) of the elements in the buffer line. This way the interval of time between the data and clock edges can be measured by small time increments that are given by the difference in propagation delays of the logic elements in the lines ($t_L - t_B$). Therefore, when the remaining distance between the data and clock pulses are less than τ_Y, the data pulse can not propagate to a subsequent row. However, it can continue to propagate horizontally with tiny time bins of magnitude $\tau_X = t_L - t_B$ until it is caught by the delayed clock edge that keeps on disabling the clock inputs of the latch elements on the lines. Hence, the smallest time increment τ_X is the measurement resolution achieved by the VDL line. That is: $\tau_X = t_L - t_B$. Knowing τ_X, τ_Y, the number of stages of logic elements in each line is determined as:

$$M \geq \tau_Y / \tau_X \tag{4.5}$$

Assuming the same propagation delay across the D-latch in horizontal and vertical directions (see section 5.1.3) and, hence, replacing τ_Y and τ_X by their values gives:

$$M \geq t_L / (t_L - t_B) \tag{4.6}$$

As pointed earlier, T_{OBS} has to obey one more restriction. The state of the matrix tap lines can be sampled only when the clock edge has propagated through all the stages of a line in order to provide a complete propagation throughout the matrix by some data pulse. From Figure 4.7, it is seen that T_{REF} is governed by two parameters, namely, $T_{delayed}$ and T_{proc}. $T_{delayed}$ is the minimum interval of time needed before the state of the matrix tap out lines can be sampled. This is given by:

$$T_{delayed} \geq M\, t_B \tag{4.7}$$

Similarly, T_{proc} is the minimum time needed for the *processing* of the sampled data from the matrix before a new phase of measurement starts. This may include the time required for data encoding and temporary storage. It should be noted that T_{proc} is implementation-dependent and may vary depending on the particular implementation scheme used for data processing.

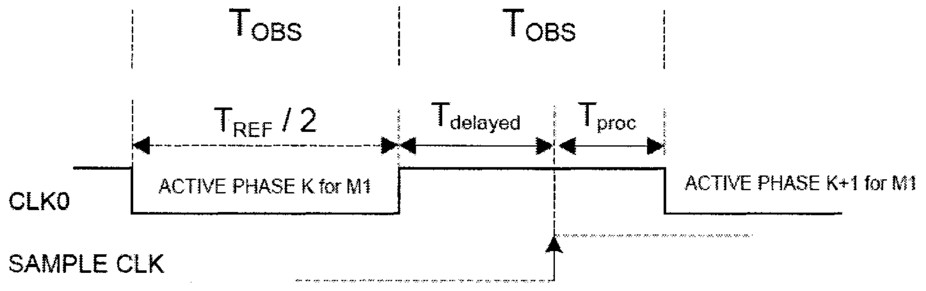


Figure 4.7: T_{OBS} decomposition

Thus, the size of the matrix is governed by eq. 4.4, eq. 4.6, and eq. 4.7. That is, the size of the delay matrix depends on the available propagation delay values for the buffer and latch elements, the observation time window T_{OBS} itself depending on the reference clock period used. Should the small time increments τ_Y and τ_X can be achieved through small-delay buffer and latch elements, a smaller observation time window T_{OBS} can be used which would subsequently imply ideally smaller matrix size.

Back-Resetting: Ideally, a data input pulse is to propagate throughout the delay matrix as expected. That is to say that an incoming data pulse is supposed to propagate through a vertical stage if the distance between the respective edges of the data pulse and the clock is larger than the single latch delay, otherwise, the pulse will propagate horizontally through the VDL line of concern. Under such ideal scenario, when a pulse moves to subsequent rows, it should no longer exist anywhere on any precedent rows. However, this assumption may fail due to the small non-zero difference in time between the vertical and the cumulative horizontal resolutions $(\Delta t = t_L - M\, t_r)$ caused by factors such as the mismatch in delays of delay cell components along a single line. This is mostly critical for incoming pulses distant from the reference clock edge with delays comparable to vertical resolution t_L of the matrix. Under such circumstances, the same data pulse may reside on more than one VDL line of the matrix, leading to erroneous tap readings or multiple detection of the same pulse.

In order to remedy the problem of multiple detection, a row-wise back resetting scheme is incorporated inside the delay matrix that would serve to clear all elements of a matrix line once the data pulse makes its way to the subsequent one. For instance when the pulse reaches stage $(i+1,\ 1)$ from $(i,\ 1)$ all the delay cells in row$_i$ should be cleared. This will ensure that no successive two rows will contain the same data pulse. The new matrix topology including the back resetting scheme is shown in Figure 4.8. The reset signal used to clear the elements of a matrix line is generated by a constant pulse generator unit that produces a constant-width reset pulse once it detects the edge of the incoming data pulse at the output of first delay cell of a matrix line. Therefore, each row on the matrix with the exception of the first is equipped with a constant reset generator that, upon detection of a new pulse, will assert a constant reset signal of ~2.5 ns to reset all the delay cells on the previous row. The choice of a constant reset signal is made as to minimize the effects of back resetting on the pulse pair resolution. Shown in Figure 4.9, the reset pulse generator is implemented by a self-reset configured flip flop driven by the output of the first cells of the lines.

Tap Out Lines Code Representation: The output of the tap lines were tested using two output code representations, namely, *thermometric* and *one-out-of-many.* In thermometric coding, the tap out lines will be at logic high up to the tap line where the pulse was caught. For example if a particular pulse makes its way to m^{th} delay cell of row n, then all tap lines of the row n starting from DC_{n0} up to DC_{nm} will have a logic-1. However, in one-out-of-many encoding, only the output of a single delay cell out of M cells of a row will have logic-1. Respective examples of thermometric and one-out-of-many binary words for an 8-bit word are *11111110* and *00000010.*

Side Effects on PPR: Ideally the *PPR* for the proposed circuit is τ_Y or a single pulse per matrix line. This is due to the choice of thermometric coding of the matrix tap lines. However, *PPR* is affected by the back-resetting scheme as the constant-width reset pulse keeps any incoming data pulse from entering the line presently reset for the duration of the reset signal.

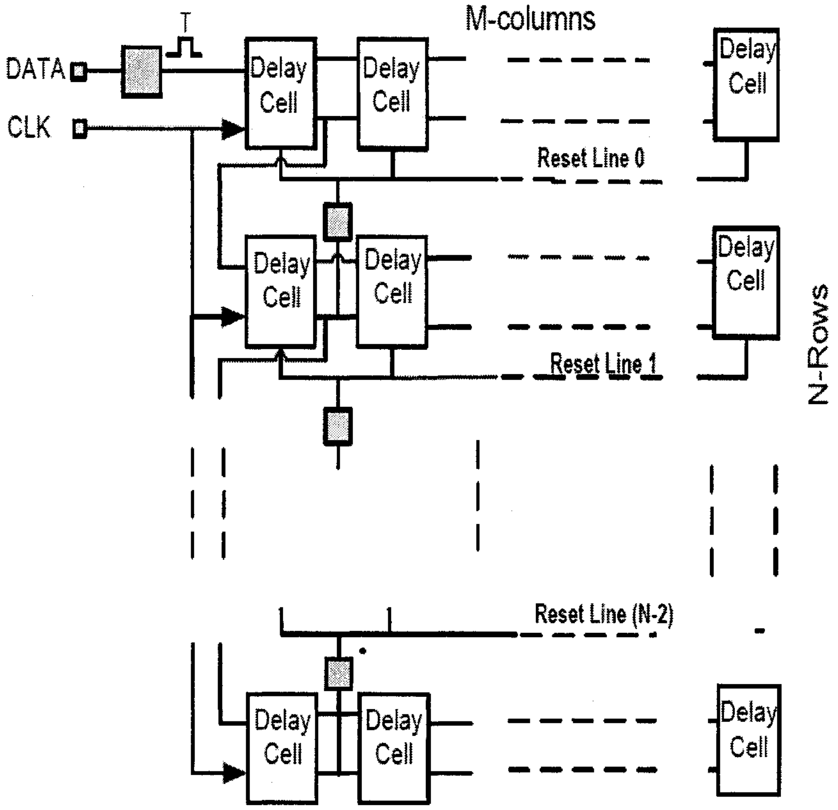


Figure 4.8: New matrix topology including the reset generators

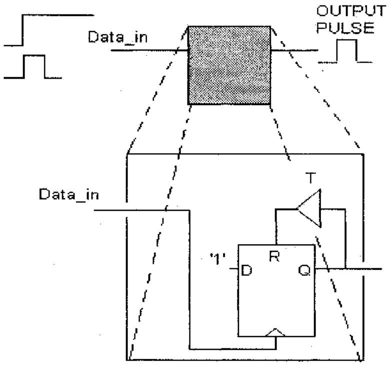


Figure 4.9: Self-reset FF used as constant-width pulse generator

Input Pulse*:* When the interval of interest is the time distance between two data pulses whose widths are shorter than the observation time window, the time interpolator (Delay matrix) will naturally achieve the time measurement with no errors. However, if the time interval of concern is the distance between two larger-width input pulses such as a clock phase with its width larger than the observation time window, erroneous dummy measurement results may be produced. This is the result of erroneous propagation of the lagging part (see Figure 4.10) of the input pulse that is still high in the next phase of measurement. In such cases, an input pulse will be interpreted as two sequential pulses. First the actual arrival time taken from the rising edge location of the incoming pulse, and second the propagation of the lagging part at the start of next phase of measurement.

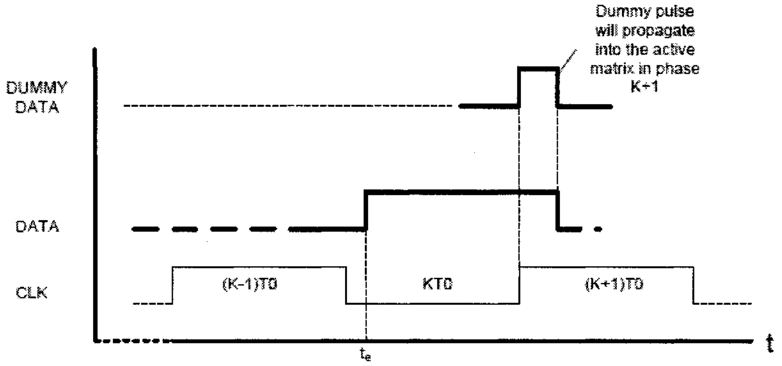

Dummy pulse will propagate into the active matrix in phase K+1
DUMMY DATA
DATA
CLK
(K-1)T0
KT0
(K+1)T0
t
t_e

Figure 4.10: Erroneous dummy pulse propagation

This multi-detection problem is removed by an internal pulse generator unit that, similarly to constant reset pulse generation, would generate a constant-width data pulse based on the rising edge of the incoming data pulse. This way, upon detecting the rising edge of the input pulse, a constant-width data pulse is made to propagate throughout the delay matrix. Regardless of the width of the incoming pulse, a constant-width data pulse propagates throughout the matrix, removing any ambiguity on the detected pulses on matrix delay lines. It should be noted that as the interval measurement is between data pulses that arrive through the same data input and follow the same propagation path, the offset delay introduced by the pulse generator does not affect the measurement results.

Measurement time: The time interpolators based on a single VDL achieve smaller time interpolation with a measurement time overhead that is relatively costly. For instance, consider a single VDL with differential measurement resolution $\tau_X = \tau_A - \tau_B$ where τ_A and τ_B are the propagation delays of elements in lines A and B of the VDL with the assumption that $\tau_A > \tau_B$. In order to measure an interval that is given by $T_m = K{\cdot}\tau_X$ where K is in the order of 50 – 100, the input pulse needs to propagate through a K-stage VDL. The time overhead in this case would be $(K{\cdot}\tau_A) - (K{\cdot}\tau_X) = K{\cdot}\tau_B$. When the factor K is large, the corresponding time overhead is large as well, especially if τ_X is very small.

The matrix of delay cells reduces the measurement time by providing a vertical propagation path with larger time bins of magnitude τ_Y. In this case, an interval $T = K\,\tau_X$ is measured with two timing resolutions, namely, τ_Y and τ_X. If the distance between the data pulse and the clock signal is larger than τ_Y, the pulse propagates vertically with the larger time step τ_Y, avoiding the costly horizontal propagation with smaller time bins. Once, the distance between the data and the clock pulses is reduced to less than τ_Y, horizontal propagation comes into play. This practice significantly reduces the measurement time overhead. Each time a data pulse propagates in the vertical direction, the measurement time is reduced by $M{\cdot}t_B$ (t_B as the buffer delay) for an $N\ X\ M$ matrix with M-stage VDL lines.

The maximum measurable interval using the delay matrix is given by the propagation delays of N-1 vertical latches and the M horizontal differential delay τ_X. That is: $T_{max} = (N\text{-}1)\ \tau_Y + M\ (t_L\text{-}\ t_B)$. To measure T_{max}, the pulse needs to propagate through (N-1) D-latches vertically (denoted by τ_Y) and through M D-latches horizontally (denoted by t_L) at coordinates (N-1, M) requiring a total measurement time of $T_{total} = (N - 1)\ \tau_Y + M\ t_L$. The maximum extra time overhead T_{ex} to measure T_{max} is:

$$T_{ex}\ = [T_{total}]–[T_{max}] = [(N\text{-}1)\ \tau_Y + M{\cdot}t_L] - [(N\text{-}1)\ \tau_Y + M\ (t_L\text{-}\ t_B)] = M\ t_B \qquad (4.8)$$

Replacing T_{max} by T_m with coordinates (m, n) gives a measurement time overhead $m{\cdot}t_B$. Therefore, the topology of the delay matrix achieves a significant reduction in the measurement time overhead.

Parallelism: One of the major drawbacks of TDC circuits is the dead time, a period of time from the end of current phase of measurement until the start of a new one during which the system can not achieve any time measurements. In the current TDC design, the conventional dead time is improved by parallelism. The TDC system under study implements two delay matrices operating with clocks that are 180 degrees out of phase. Having two delay matrices operating in both phases of the reference clock period allows detection of the hit signals in every clock cycle. At any given time while matrix-1 is busy watching for incoming events in *phase-1* of the clock period, matrix-2 is busy processing the event that it captured in the preceding clock phase. Similarly while matrix-2 is busy watching for incoming events in *phase-2*, matrix-1 is busy processing the event(s) that it captured in *phase-1*. Hence, each matrix can detect and process one or more hit signals subject to PPR in both phases of the same clock cycle. Therefore, in contrast to earlier solutions, the conventional dead time required to re-activate the system for next round of time measurement has been eliminated as the system is always active and detecting. Using this technique, several time intervals among sequential data pulses with respect to a common/non-common reference point can be detected. The only limiting factor in this scheme would be the storage capacity for the results of the continuous data pulses.

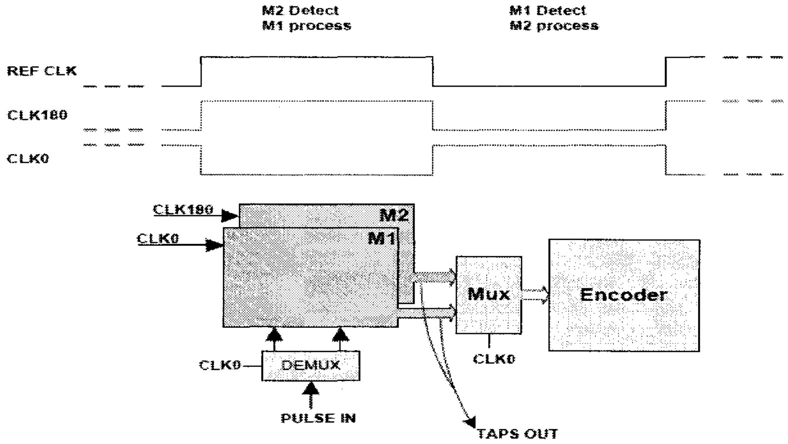


Figure 4.11: Parallel operation of delay matrices

4.2.4 Overall TDC Architecture

In addition to the delay matrix as the time interpolator, the proposed TDC architecture is composed of supporting logic blocks such as the coarse binary counter for coarse time measurement, an encoder unit for measurement data encoding, intermediate storage components, and a calibration logic unit to improve measurement accuracy along with the control logic managing the overall operation of the system. The overall system is shown in Figure 4.12.

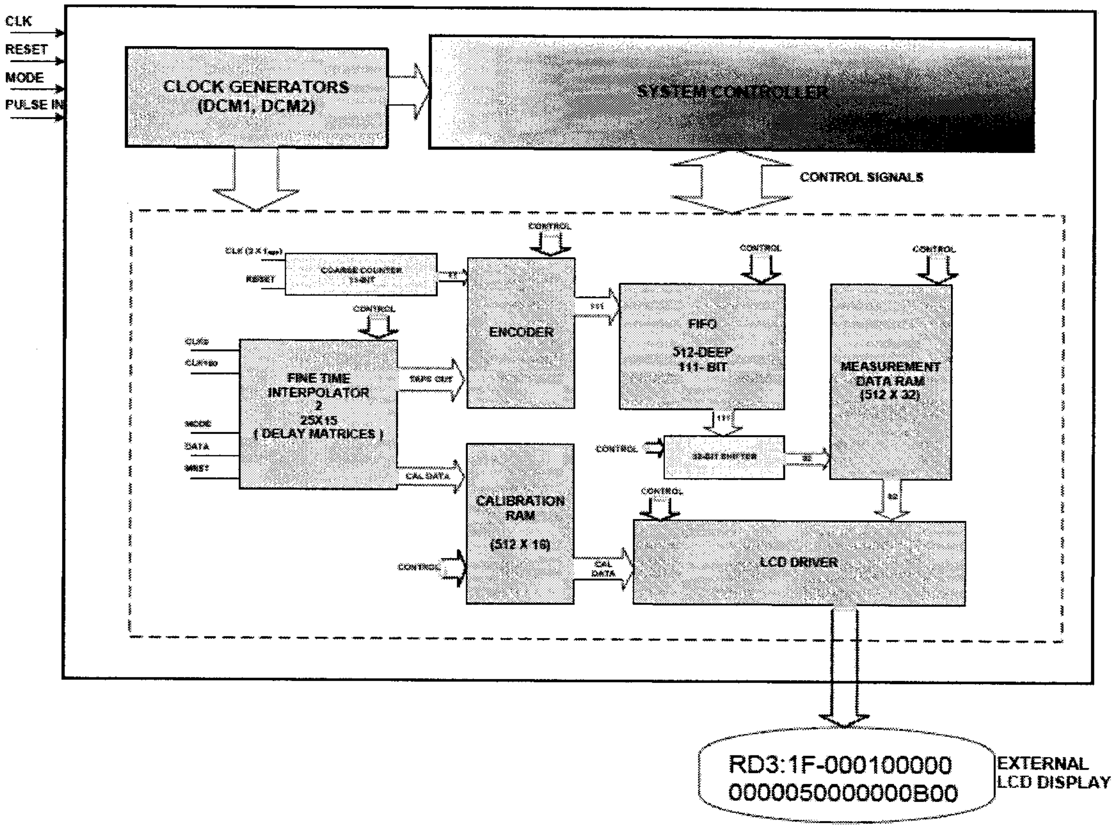


Figure 4.12: Overall TDC system

A brief overview of the overall system operation follows, with the detailed description of each system component provided in the next section. Under normal operation the tap lines' readings corresponding to the active delay matrix in phase *K* along with the coarse count value from the binary counter corresponding to phase *K+1* are sampled into the data encoder unit. The encoded word from the data encoder is then transferred onto a custom-size FIFO for temporary storage and control is returned back for the next phase of detection. The clock frequency for the binary counter is twice the frequency of the reference clock signal used for the observation time window. This way in each phase of the reference clock (for each tap readings from both matrices) a corresponding coarse count value is available. The system continuously detects and stores the encoded data regarding the incoming data pulses with respect to the reference clock edges as long as enough storage area is available in the FIFO unit. Once the FIFO is full, the data is transferred to the Data RAM for permanent storage visible to outside world through some simple RAM interface. In the system architecture of Figure 4.12, the data stored in Data RAM are displayed on an LCD display. A snapshot of a sample measurement data displayed on the LCD display when implemented on the Spartan^{TM}-3 FPGA platform is shown in Figure 4.13 below.

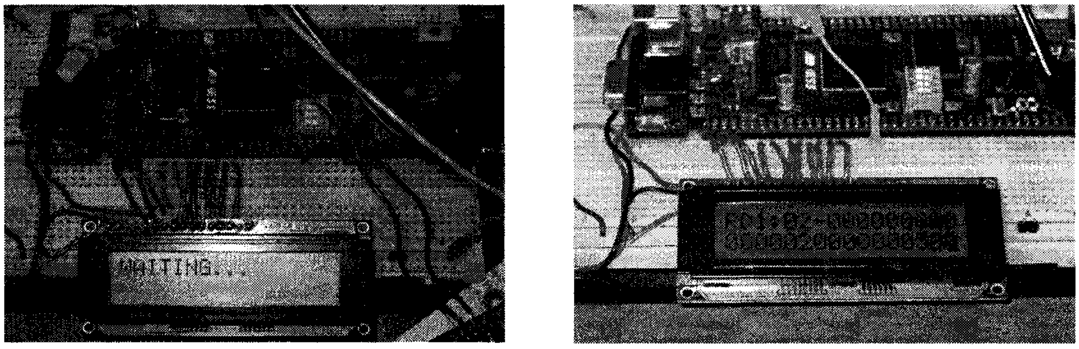


Figure 4.13: Snapshots of the test system on Spartan^{TM}-3 FPGA Platform

4.3 Components And System Integration

4.3.1 Target Implementation Platform

Recent advances in sub-micron silicon technology have overcome early limitations imposed on electronic circuit designers in the areas of Application-specific Integrated Circuits (ASIC) and the programmable logic devices such as Field Programmable Gate Arrays (FPGA). Current FPGA vendors provide designers with many built-in common-use logic blocks such as memory blocks, multiplier units, delay-locked loop (DLL), designated delay lines, and optimized logic primitives as libraries. These embedded blocks are optimally designed and placed within the FPGA devices in an attempt to equip designers with the necessary tools in order to come up with robust and complex designs incorporating several circuit blocks into a single functional system.

Logic mapping onto an FPGA is achieved using the Configurable Logic Blocks (CLB) and Input/Output Blocks (IOB) within the FPGA technology. Each of these blocks is equipped with logic resources such as registers and other basic logic elements. While the actual mapping of logic onto these blocks may not be vital in all applications, some applications such as the TDC circuit under study or in particular the time interpolator (delay matrix) requires logic mappings onto CLBs. This is due to the strict requirement of the delay matrix to provide time increments that are uniform among successive delay cells. Therefore, despite the simplistic theoretical description of the matrix topology, the actual design component mappings onto FPGA blocks requires good knowledge of the target FPGA technology and some implementation efforts. For this project, the FPGA device chosen is the Xc3s1000 from the SPARTAN^{TM}-3 family of FPGA by XILINX, which is briefly described below.

Spartan^{TM}-3 FPGA Family: Spartan^{TM}-3 is a low-cost, high density family of FPGA from XILINX. The logic density in Spartan^{TM}-3 family ranges from 50000 system gates in the least-dense Xc3s50 device up to highest density of 5 million system gates in Xc3s5000. The Spartan^{TM}-3 family is an enhanced version of the earlier Spartan-IIE family in terms of logic resources, internal RAM blocks, and an increased number of input/output pins, improved global clock lines and improved clock management functions, along with several IP cores including a MicroBlaze^{TM} processor. With the features enlisted in Table 4.1, Xc3s1000 is a suitable low-cost FPGA device for realizing the TDC circuit under study. In particular, features such as the DCM units for multiple in- or out-of-phase clock signals, the BRAM units for temporary storage, wide number of user I/Os, enhanced global buffers for clock signals and the high logic density are vital to the realization of the TDC circuit.

Table 4.1: Summary of features of Xc3s1000 FPGA Device [36]

Feature	**Qty**
Logic Density	1 Million system gates
Configurable Logic Blocks (CLB)	1920
Logic Cell (1 LUT + 1 DFF ; 8/CLB)	15360
Block RAM Bits (K = 1024)	432 K
Distributed RAM Bits (K = 1024)	120 K
Digital Clock Manager (DCM)	4
Global Clock Buffer	8
Maximum User I/O	391
Dedicated (18 x 18) Multipliers	24

Spartan^{TM}-3 CLB Overview: The general structure of Spartan^{TM}-3 device is shown in Figure 4.14. The surface of the device is divided into a grid of CLBs indexed by column and row coordinates. Each CLB is further divided into four identical interconnected units referred as slices. Hence, four coordinate pairs are reserved per CLB. That is: X_iY_j, X_iY_{j+1}, $X_{i+1}Y_j$, and $X_{i+1}Y_j$ where each coordinate pair specifies a particular slice within the CLB. The common logic elements within the slices of a CLB include: carry logic, two function generators (LUT), two storage elements (FF/Latch), multiplexers and arithmetic gates. Each pair of slices per CLB column (i.e. X_iY_j and X_iY_{j+1}) is equipped with independent fast carry chains. In addition, the slices with even x-coordinates support Distributed RAM and shifting data by 16-bit registers [36]. The main resource for implementing logic functions is the Look-Up Table (LUT) that acts as a four-input and single output function generator. The storage elements within a slice share the common clock and clock enable input paths with separate data inputs. These are programmable elements that act either as a D-flip flop or level sensitive latch. For more information on the Spartan^{TM}-3 family of FPGA, see [36].

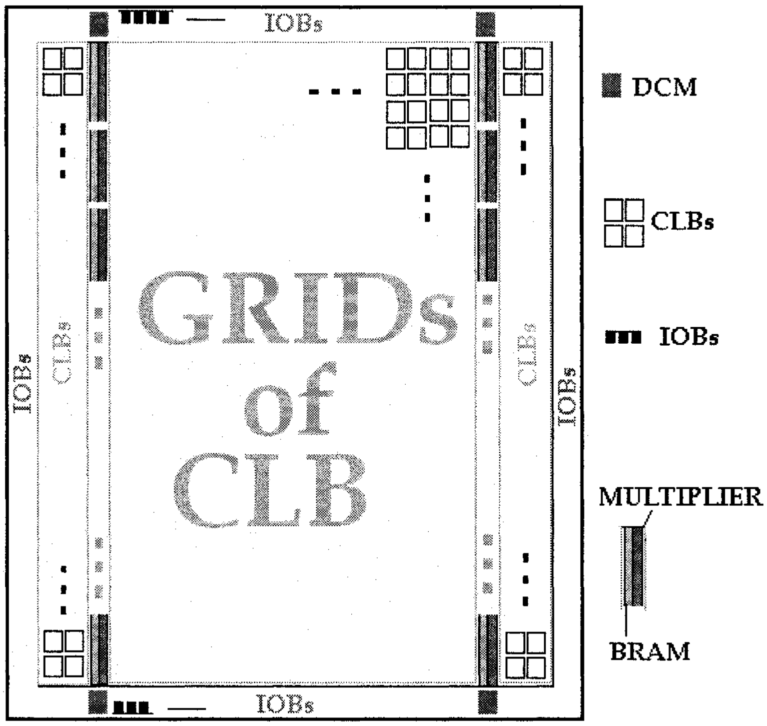


Figure 4.14: Spartan^{TM}-3 general surface view

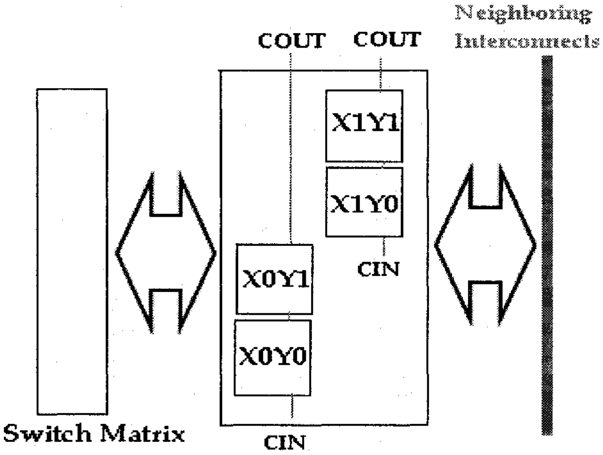


Figure 4.15: CLB slice structure [36]

All design components in the TDC system under study were described in VHDL. The synthesis tool used for synthesizing the design components is the XILINX Synthesis Tool (XST) embedded in the ISE^{TM} Foundation ^{TM}, the logic design environment from XILINX. The design entry point is the Project Navigator graphical user interface embedded in ISE^{TM} Foundation ^{TM}.

4.3.2 Delay Matrix

From design, it was seen that the basic component of the proposed delay matrix is a delay cell composed of a buffer and a negative level sensitive latch element. With the symbolic representation of a buffer being as the standard symbol for an inverter with no circle on the tip of the triangular shape, the actual realization of the latter in the gate level may vary. Depending on the application and the corresponding requirements and specifications, some may implement a buffer as an even number of cascaded inverters, whereas others may use a controlled gated logic element such as an AND gate with one input tied to *VCC*. When targeting FPGA devices, the choice of implementation may also be influenced by the internal structure of the basic CLB element. Most logic functions in FPGAs are implemented using LUTs (function generators). With the CLB structure of Spartan^{TM}-3 as briefly described earlier, the schematic of the buffer elements is depicted in Figure 4.16.

The buffer element is implemented as the combination of a one-input look-up table driving the wide-function multiplexer within a CLB slice that combines the upper and lower LUT outputs into a single driver net. Therefore, a first order delay approximation ignoring the interconnect delays indicates the buffer gate delay roughly given by the propagation delays for a single LUT and the wide-function multiplexer. On the other hand, the latch element of the delay cell can be implemented using the *FF* elements available within the CLB slices, namely, *FFX* and *FFY*.

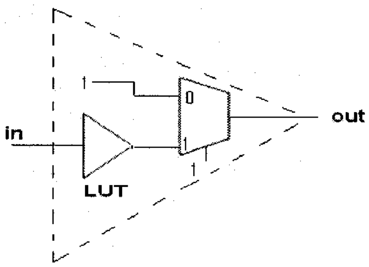


Figure 4.16: Buffer implementation

The choice of the buffer implementation as shown in Figure 4.16 was mainly governed by the propagation delay sought. Given that the propagation delay of the latch element within the CLB slice is fixed (single FF path) and that the timing resolution depends on the difference in propagation delays of the latch and buffer elements, a variation of the buffer implementations had to be tested before an acceptable delay difference (~100ps) between the latch and the achieved buffer configuration was obtained. With a single LUT used as a buffer, the difference in delays was in the range of ~400ps, leading to a timing resolution that largely deviates from the intended resolution. Therefore, with the choice of buffer and latch implementations as in above, a single CLB was dedicated per delay cell. Moreover, as the buffer and latch delays may also be affected by interconnect delay, the delay cells of a single matrix need to be equidistant in order for the cell delays to be uniform. Hence, all delay cells need to be manually placed at equal distances.

Using the eq. 4.2, eq. 4.6 and eq. 4.7 and the simulation values for $t_B \sim 1.475$ns and $t_L \sim 1.588$ns outlined in Table 6.1, the respective values of N and M for the delay matrix needed to be larger than 22 and 13 for $T_{OBS} = 35$ns. The size of the delay matrix was chosen to be 25 x 15 with two extra cells per row. The extra cells per row are added as to avoid loss of data pulse along a row due to possibly non-uniform cell delays that may undesirably let a pulse go undetected. This happens when a data pulse propagates completely through the row before the sample clock arrives.

In VHDL, the matrix was implemented as an array of interconnected delay lines using structural VHDL with each components (buffer, delay cell) being implemented separately and then combined into a single unit using port mapping. In addition to the basic delay cells, the matrix topology implements other logic elements such as the back resetting units, the thermometric and one-out-of-many coding of tap lines, and constant-width pulse generator. The back resetting units are implemented using port mapped flip-flop primitives and buffer elements configured in a topology as in Figure 4.9. These units are then placed at CLB locations that are close to the first delay cells of each row (see the floorplanned design view in APPENDIX-II).

The delay matrix source code also implements logic to enable pulling the internal RESET lines low upon arrival of the master reset signal. Internal RESET signals are used in back-resetting scheme. They are asserted high when a pulse makes its way to the first delay cell of a row in concern. Upon assertion of system master reset signal, these signals need to be deactivated; otherwise the lines will remain reset preventing any incoming signals to propagate through.

4.3.3 Coarse Counter

Initially the TDC contained an 8-bit coarse counter. This was updated to an 11-bit binary counter to increase the dynamic range or maximum measurable time interval. In parallel with the matrix of delay cells as the fine time interpolator, the binary counter operates with a clock frequency ($f = 2\ f_{\text{REF}}$) twice that of the delay matrices in order to provide coarse count values in both phases of the reference clock for both delay matrices.. With 11-bit output the maximum count value for the counter is 2^{11}, leading to a maximum dynamic range of $2^{11}\ x\ T_{OBS}$. This dynamic range can always be increased by utilizing a larger size binary counter.

4.3.4 Data Encoder

The data on the matrix tap out lines are to be encoded into a data word before it is stored in memory. The matrix tap out lines in TDC system are sampled into the encoder unit where they are encoded into a data word using the simple encoding scheme described next. The output of the encoder unit reflects the encoded word corresponding to the registered matrix tap lines by the sampling clock and the coarse counter value appended as *MSB*.

Encoding Scheme: A data word out of the encoder unit represents the tap out lines arranged as a stream of hexadecimal values. The value of a hexadecimal digit in the data word indicates the number (*m*) of delay cells propagated by the data pulse along the matrix row of concern. The location of the hexadecimal value indicates the number (*n*) of rows of the matrix where the pulse is residing. The respective domains of ‘n’ and ‘m’ are given as: n:*0,1,...N-1* and m:*1,2,...,M*. An (*n, m*) coordinate indicates that a data pulse exists on the m^{th} stage of the n^{th} row of the matrix and the corresponding interval (T_e) is given by ($n\tau_Y + m\tau_X$). Consider the sample encoded word "10000 02000 00900 00005 0000D" corresponding to the tap out lines (25 X 15 delay matrix). In this encoded word, the coordinates (*n, m*) for the five data pulses residing in the delay matrix are (*0, D*), (*5, 5*), (*12, 9*), (*18, 2*) and (*24, 1*). Taking the coordinate (*5, 5*) we see that the pulse in concern has propagated five rows vertically and five stages horizontally, hence representing the corresponding interval $T_e = 5\tau_Y + 5\tau_X$. In addition to the matrix tap out lines the coarse counter value corresponding to a tap reading can also be attached to the encoded word. If an *N*-bit coarse counter is used, the *N* counter bits will be concatenated to the most significant bit of the encoded data word as: "CCCC-10000020000009000005 0000D" with CCCC as the 4-bit coarse count value for a 4-bit binary counter.

The encoding scheme was embedded in an asynchronous encoder. The data from the tap out lines of the delay matrices unit is sampled into the encoder registers by the sampling clock. The sampling clock is provided by the matrices unit. Once registered, the data asynchronously are encoded into a 100-bit word (25 rows of 4-bit encoded value). The VHDL code (see APPENDIX-III) for encoding uses CASE-SELECT statements combined in a single process. The latter is asserted when tap data are sampled into internal input registers.

It is worthwhile to mention that prior to using the CASE-SELECT clause the encoder was implemented using IF-THEN-ELSE statements. While both method of encoder implementation were functional, the acquired data using CASE-SELECT implementation were more immune to timing errors. This is because when using the IF-THEN-ELSE clause, a priority encoder is obtained. A priority encoder always goes through all the conditions preceding the condition of interest. In the case of the TDC system under study, there is no need for a priority encoder. Therefore, using CASE-SELECT is preferable in terms of improved timing.

4.3.5 Synchronous FIFO Unit

The TDC system incorporates a FIFO (First-In-First-Out) unit for temporary storage of the measurement data out of the encoder unit. The main purpose of using the FIFO unit is to be able to quickly store encoded data before the next phase of detection starts. Since the size of the encoded data word depends on the actual size of the matrix, single clock data storage is not possible with the available fixed data width RAM units. In order to store the encoded word from the encoder directly into the data RAM more than a single clock would be required with smaller burst size data. This would further delay the arrival of the next phase of detection. To remedy this problem a custom-size FIFO unit that can provide single clock cycle data storage is used.

4.3.6 32-bit Shifter

In order to transfer the larger-width data words from the FIFO to the smaller-width Data RAM in multiple data chunks, a shifter is needed. This unit simply reads data from the FIFO and transfers it to the Data RAM in multiple smaller data words. The data stored in the FIFO is 111 bits wide (11-bit coarse count value + 100-bit encoded tap data), whereas the RAM unit used has accessible word locations of 32-bits width. A single entry of the FIFO unit requires 4 clock cycles in order to be transferred to the RAM unit.

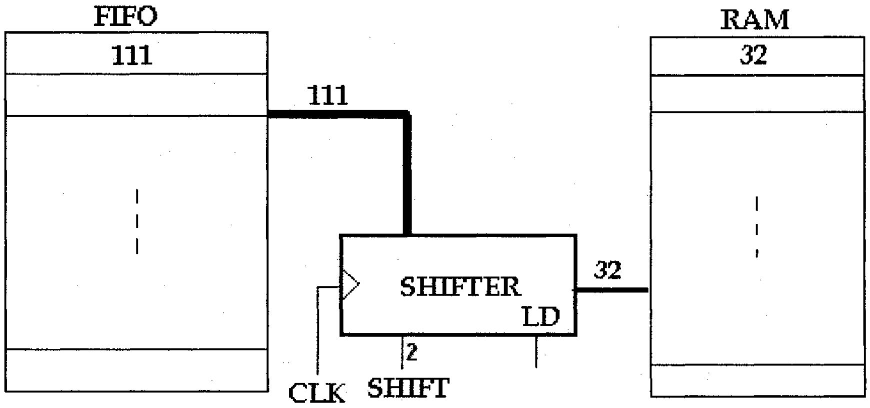


Figure 4.17: 32-bit Shifter in Operation

4.3.7 Data RAM

The measurement data stored in the FIFO is transferred to a RAM memory unit that can be accessed by some external interfaces for further processing. The RAM memories used in the Test System were implemented using the built-in Block RAM units inside the Spartan^{TM}-3 FPGA. FPGA manufacturers provide a variety of built-in RAM blocks with different size organizations. The BRAM used in this project is a synchronous dual port 512 x 32-bit where data READ and WRITE are made with the rising edge of the clock should the corresponding control signals be asserted. The RAM locations are accessible through both ports and different clock signals can be used for each port. For instance, the data are stored with the faster controller clock and read with the slower clock to interface the LCD display. For more info on BRAM and its variants see [36].

4.3.8 Digital Clock Manager (DCM) for Clock Generation

Probably one of the most important built-in components inside FPGA devices, the digital clock manager unit encompasses a DLL unit, a frequency synthesizer, and a phase shifter to achieve three main functions namely phase shifting, frequency synthesis (wide range of output frequencies), and skew elimination in the output clock signals. Use of a DCM in this project is vital as the system uses several clock signals with various frequencies. For instance, the reference clock frequency for the delay matrices is half the frequency of the coarse counter. Similarly, the controller unit runs at a frequency that is much faster (~ x 5) than the reference clock frequency used for clocks CLK0 and CLK180 for the respective matrices 1 and 2. The 180-degree out of phase signals are also outputs of DCM unit. Below is the block diagram of a built-in DCM inside Spartan^{TM}-3 FPGA.

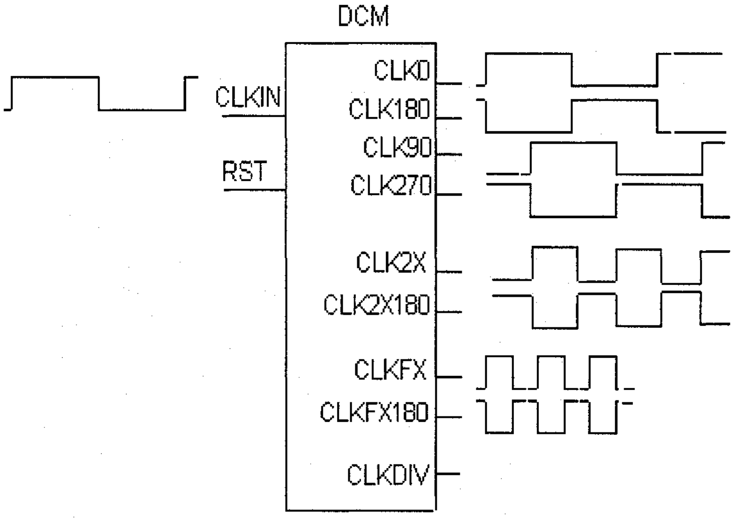


Figure 4.18: Simplified DCM block diagram

4.3.9 LCD Driver

The system interface to the outside world (user) is achieved through a 16-character × 2-line Dot Matrix Liquid Crystal Display (LCD) capable of displaying alphanumeric characters and symbols. Manufactured by HITACHI, the HD44780U LCD display comes with a built-in controller that can be configured to drive the dot matrix display area to display up to two lines of 16 characters (alphanumeric and symbols). In order to embed the HD44780U LCD display into the system, the system designer needs to provide control (3-bit) and data (8-bit) signals that will be used to configure the built-in controller to drive the dot matrix LCD to display the characters of interest.

The steps required to provide the timely control and data values for the control and data lines of the HD44780U LCD display unit are encompassed into a single VHDL file comprising of multiple finite state machines (FSM) for normal and calibration data display. This file in cooperation with the main system controller unit generates the timely control and data values that drive the control and data lines of the LCD display for displaying the requested characters/symbols representing the calibration or normal test data read from corresponding block RAM memories in the system. Data from RAM are read one-by-one into internal registers using a synchronous handshake managed by the main system controller and then the corresponding LCD control and data bits are sent to the built-in LCD controller of the HD44780U unit for display.

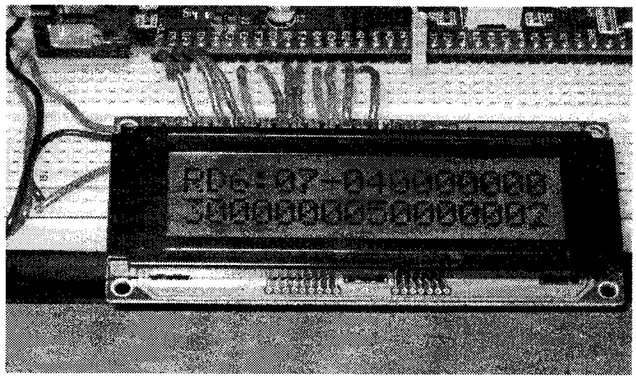


Figure 4.19: Typical data display on LCD with an 8-bit coarse count value

The LCD display exhibits the encoded matrix taps lines exactly as they are encoded. The reading in Figure 4.19 shows the existence of four pulses within the same time window with matrix coordinates (*0, 2*), (*7, 5*), (*15, 3*), and (*23, 4*) of the matrix. The coarse counter value for the current phase of measurement is the hexadecimal value of 7.

The LCD display is character mapped: in order to display a character on the display, the corresponding ASCII representation needs to be written in the internal memory of the LCD controller. The character sequence on the LCD display is a sequence of states with each character being handled by one state. For instance, to display the character "*F*" on the LCD display the corresponding ASCII representation (*x"46"*) needs to be written into the internal RAM of the LCD controller by driving the LCD data and control lines. This can be handled by designating a state , say *S_F*, where the output value for data lines of the LCD display are the bit-composition (*x"46"*) corresponding to the character "*F*". This way for each character on display, a single state is required to output the corresponding ASCII code. Once data is written in the internal RAM of the LCD unit, the embedded controller within the LCD unit will handle the actual steps required for driving the dot matrix display. Therefore, as far as the end user is concerned, proper character ASCII data and control values are to be provided based on the functional and timing requirements outlined in the data sheet of HD44780U. In addition to the FSMs for generating the bit-composition of characters to be displayed, other logic for custom display control of calibration and experimental data such as timers, counters, and control flags for alternate displays are also embedded inside the LCD driver unit. The VHDL file is appended in APPENDIX-III.

4.3.10 System Control Unit

The main controlling body of the system is a module based on finite state machines (FSM) that manages the overall operation of the various components of the TDC system. Having a main controlling module is a common practice in designing digital systems. As such, the interactions among various system components to achieve the expected system behaviour are managed by controlling the routing of data along the various data paths that connect the many system modules together. Such an approach breaks the digital system into two main modules, namely, the data path consisting of the functional modules and the corresponding interconnects, and the control unit (FSM or Microcontroller –based) consisting of control circuitry to assert timely control signals. The control unit in the TDC system manages system operation under normal and calibration modes. During normal operation, the several tasks to be administered include the generation of control signals during the detection, processing, and display phases of operation. This has been achieved by dividing the operation of the control unit into three main finite state machines, namely, *DETECT*, *PROCESS*, and *DISPLAY*. The system operation in calibration mode is discussed in the next section.

The *DETECT* FSM is responsible for asserting notification signals for storage of detected and encoded data. Upon system reset, the control moves to the initial state where based on the mode of operation (Calibration or Normal) the next state transition is determined. As depicted in Figure 4.20, when in normal mode, the control moves to a wait state until new encoded data is ready to be stored. Upon detection of the sample clock edge, the *ACTIVATE* signal is asserted which forces a transition to the temporary state at the next rising edge of control clock. Meanwhile, a reset pulse *RST_M* is generated to clear the contents of the active matrix from any residues of the propagating pulses. When in *TMP* state, *START_STORE* is asserted to trigger storing of the data handled by the *PROCESS* FSM. Subsequently the control returns back to waiting state for next round of data acquisition.

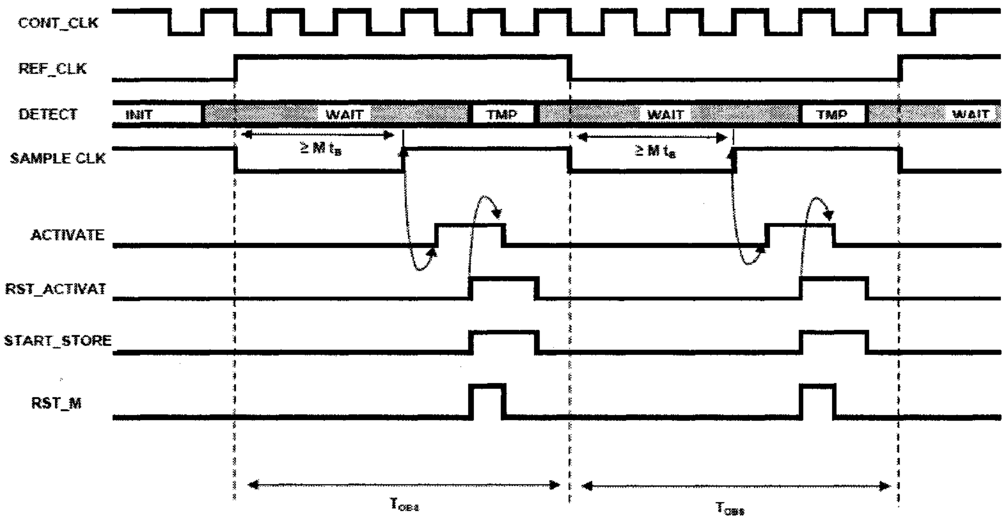


Figure 4.20: Waveforms showing DETECT state transitions

The *PROCESS* machine is in charge of generating control signals required for temporary and permanent storage of encoded data into the respective FIFO and RAM units. Upon system reset, the *PROCESS* FSM enters an initial state with a subsequent unconditional transition to the *PROCESS* waiting state where the state transition blocks on the *START_STORE* flag to be asserted by the *DETECT* FSM. Upon assertion of the *START_STORE*, *PROCESS* FSM enters the subsequent state where appropriate FIFO control signals are asserted for temporary storage of the encoded data. This is done in two states. One for generating the control signals for *FIFO_WRITE* operation, and the other for verifying whether the FIFO has reached its maximum storage capacity. If the maximum capacity is not reached, the *PROCESS* machine returns back to its waiting state for more data to be stored and the process continues as before. However, if the FIFO unit has no vacant storage slot, the control moves to states responsible for transferring the data into the RAM unit for permanent storage. The steps required for transferring data from

FIFO to the RAM include generation of control signals for the SHIFTER unit (for data shifting in chunks of 32-bits), the RAM (for write operations), and the FIFO (for *FIFO_READ* operation). The 32-bit data shifting is required as the RAM block used is of size 512 x 32. The data transfer operation is the typical case of data transfer among modules of an arbitrary digital system. The state diagrams and corresponding algorithmic state machine (ASM) charts are shown in Figure 4.21and Figure 4.23.

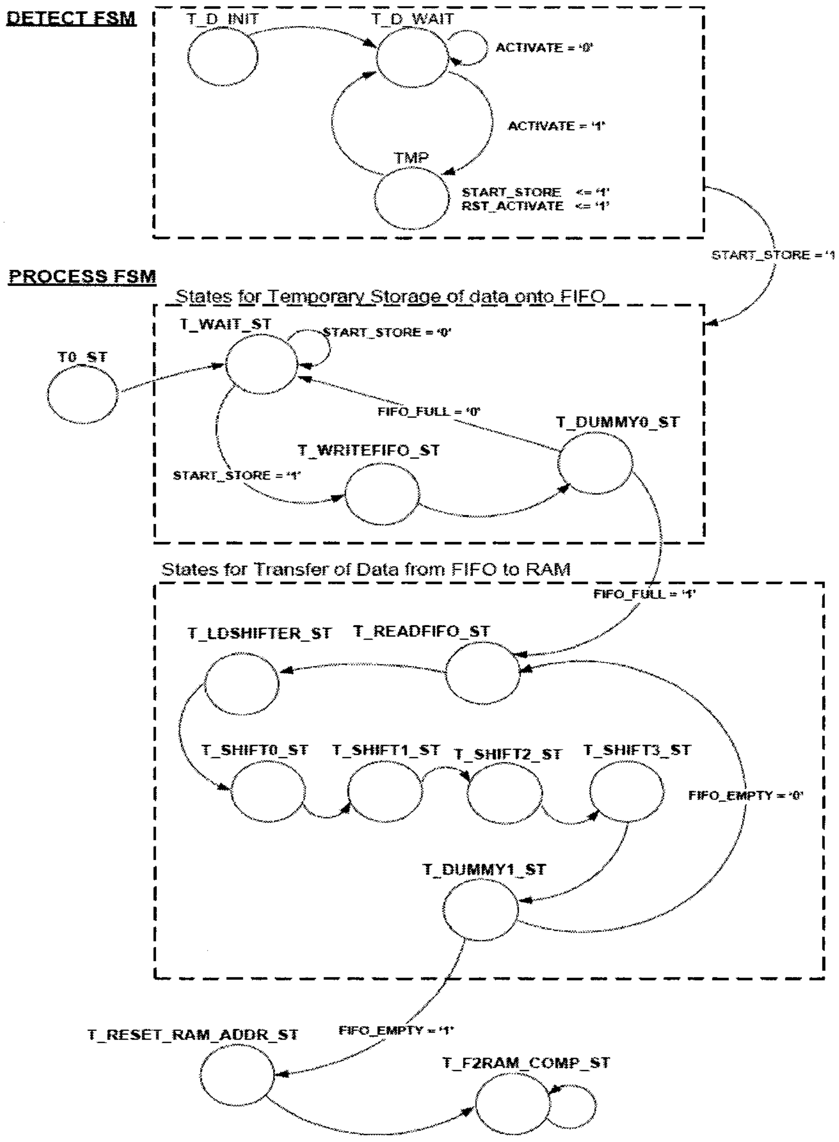


Figure 4.21: State Diagrams for DETECT and PROCESS FSMs

The *DISPLAY* machine comes into play once all the data from the FIFO is transferred into the RAM block. At the last data transfer state, the *START_DISP* signal is asserted to initiate the start of data display from RAM. The *DISPLAY* machine is responsible to generating the handshake control signals required between the RAM and the LDC driver. The LCD driver unit reads data from the RAM block one-by-one for display. It is a continuous read and display operation for all the data in the RAM block. The displaying of data is the last and final operation of the TDC prototype system where the stored data in the RAM are to be continuously displayed on the 16-character x 2-line Dot Matrix Liquid Crystal Display. The VHDL code is annexed in APPENDIX-III.

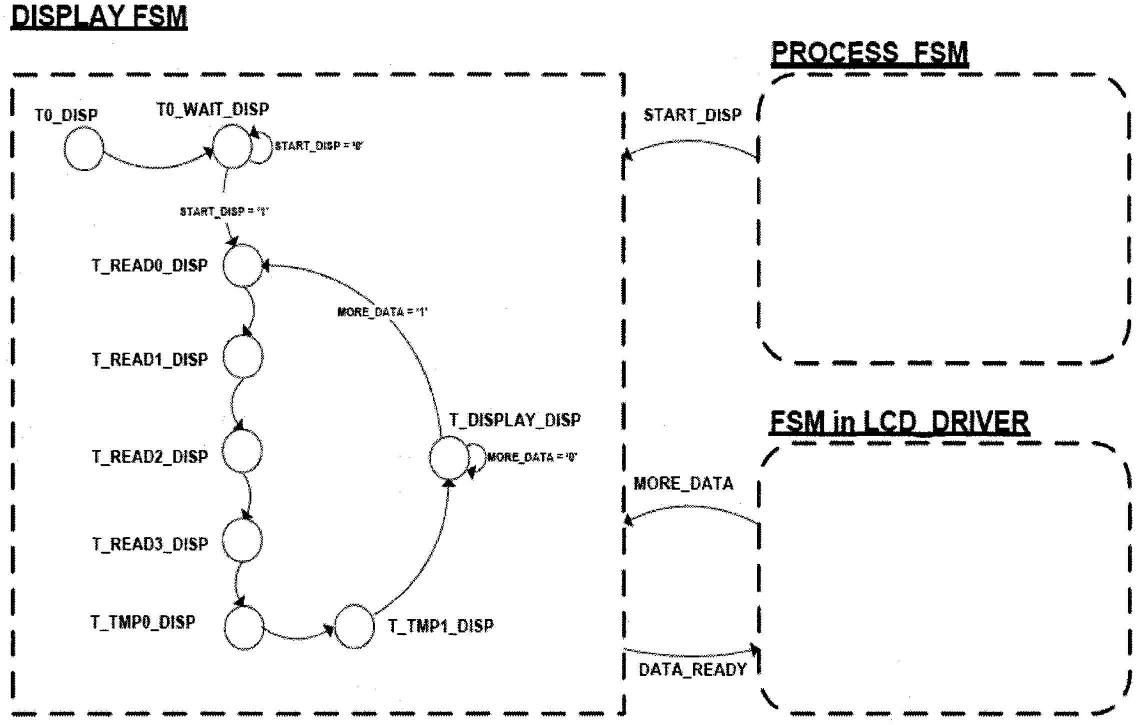


Figure 4.22: State diagram for DISPLAY FSM

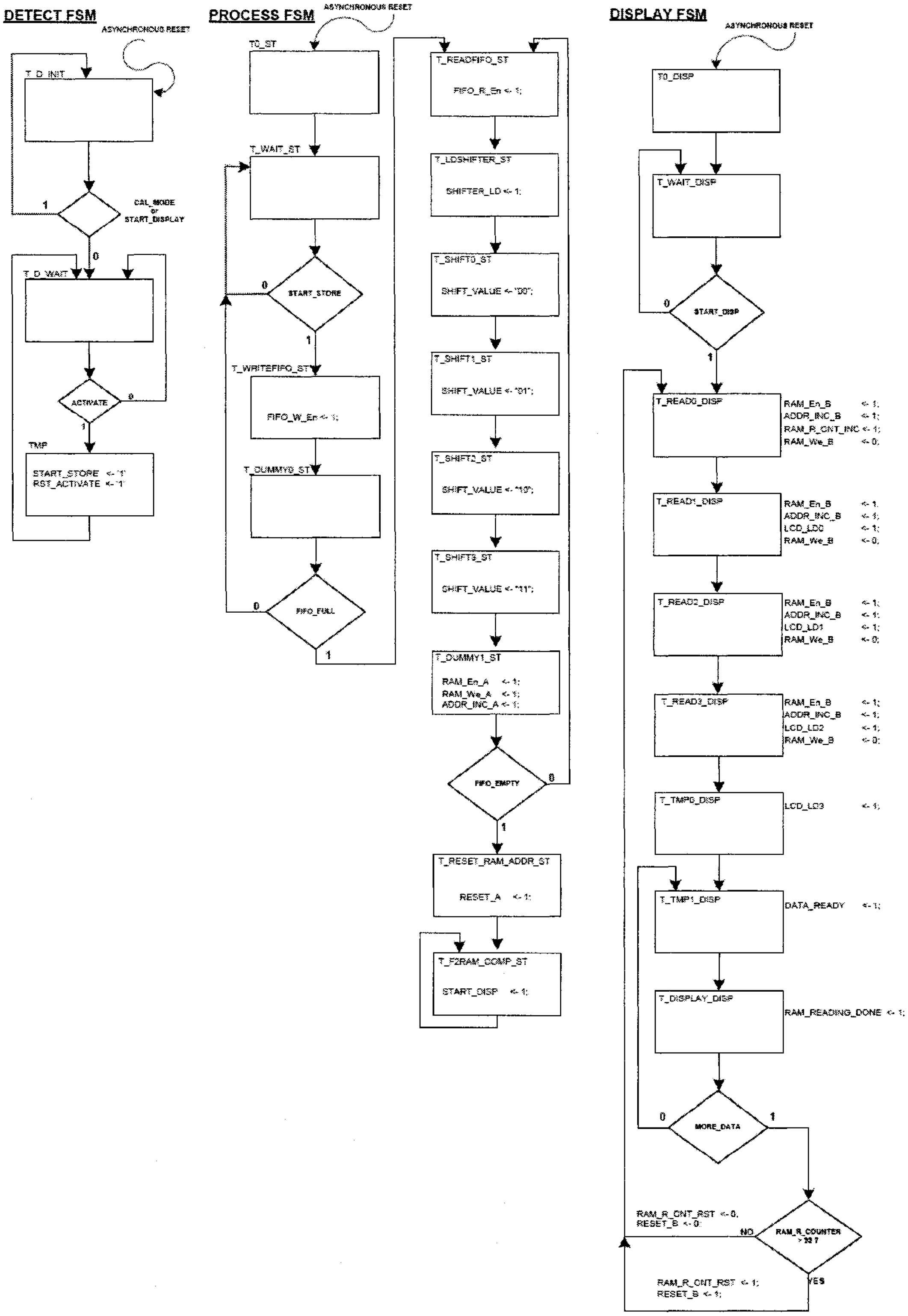


Figure 4.23: ASM-CHART for NORMAL OPERATION

4.4 Delay Calibration

While the static timing values provided by the simulation tools can be used to analyze system functionality and static timing, they may not accurately represent the actual delay values on the die. In order to precisely characterize the performance of the circuit, the TDC circuit needs a calibration scheme to find an estimation of the actual delay values for cell components. To estimate τ_Y, the vertical propagation delay of the latch elements needs to be investigated, whereas estimation of τ_X requires finding an estimate for the horizontal propagation delay by the buffer and latch elements of the lines. To do so, two calibration methods are proposed. One aims to estimate the resolution, while the other aims to estimate the component delays along the horizontal and vertical paths within the FPGA surface.

4.4.1 RO-based Calibration Scheme

This scheme aims to estimate component delays. The method suggests reconfiguration of a buffer and a latch line into ring oscillators (*RO*) by inserting identical inverting cells at the beginning of each line. The schematic for this approach is shown in Figure 4.24.

The periods of the ring oscillators (T_{RO}) can be found from the value of the corresponding binary counters incremented by the *RO*-generated signals for the duration of $T_{CAL.}$ The error in the calculated periods of *RO* becomes negligible if a large count interval (T_{CAL}) is used. Having found the corresponding periods of the buffer and latch – based *RO*, component delays of the line elements (buffer, latch) along the horizontal path of the FPGA surface are found. The accuracy of this method depends on careful placement and routing of the line components in both lines such that the feedback line for both *RO* is equal. This way, the feedback delay in both lines would be a constant value, having little or no effects on the final calculation of the horizontal resolution $\tau_{X.}$ Estimation of τ_Y on the other hand involves having two *N*-element *RO* circuits placed vertically (see Figure 4.25) somewhere along the FPGA surface. One of the *RO* circuits acts as the reference *RO* by generating a periodic signal with a period of $T_{REF\text{-}RO}$. The second *RO* has a device under test (*DUT*) inserted somewhere along the ring path generating a periodic signal with period of $T_{DUT\text{-}RO}$. The addition of the *DUT* makes the periods of the *RO* circuits differ by twice the delay of the *DUT*. That is: $2\Delta t = T_{DUT\text{-}RO} - T_{REF\text{-}RO}$ with Δt being the added propagation delay by the *DUT*. The *DUT* in this case would consist of vertically cascaded matrix cells similar to the vertical interconnection of the matrix rows through their first cells. Knowing the number of cells in the *DUT*, the vertical propagation of the latch elements are found from the difference in *RO* periods obtained experimentally. Again, for proper estimation the ring oscillators must be carefully placed as to have equal inter-cell and feedback net delays.

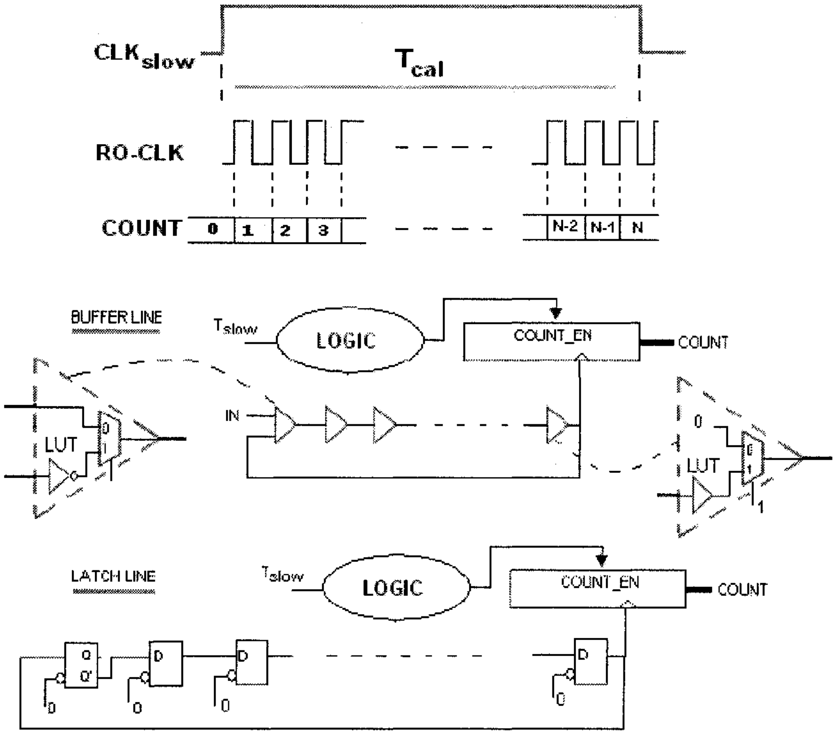


Figure 4.24: Buffer and latch line reconfigured as RO

Originally, each row of the matrix was the target for reconfiguration into a *RO* to estimate the cell propagation delay along the corresponding horizontal path. This way when in calibration mode, the matrices' rows one after the other would be reconfigured into *RO* and corresponding calibration data would be stored into calibration RAM. After completion of the test, the data would be read for processing. Unfortunately, the implementation of this strategy was found to affect the normal operation of the TDC due to adding dummy buffer cells along the common clock path in order to cancel the effects of the inverting cells required to reconfigure lines into *RO*. However, having a separate implementation of this strategy that is functional uniquely under calibration mode would definitely help in investigating the delay mismatch along the target area within the FPGA surface.

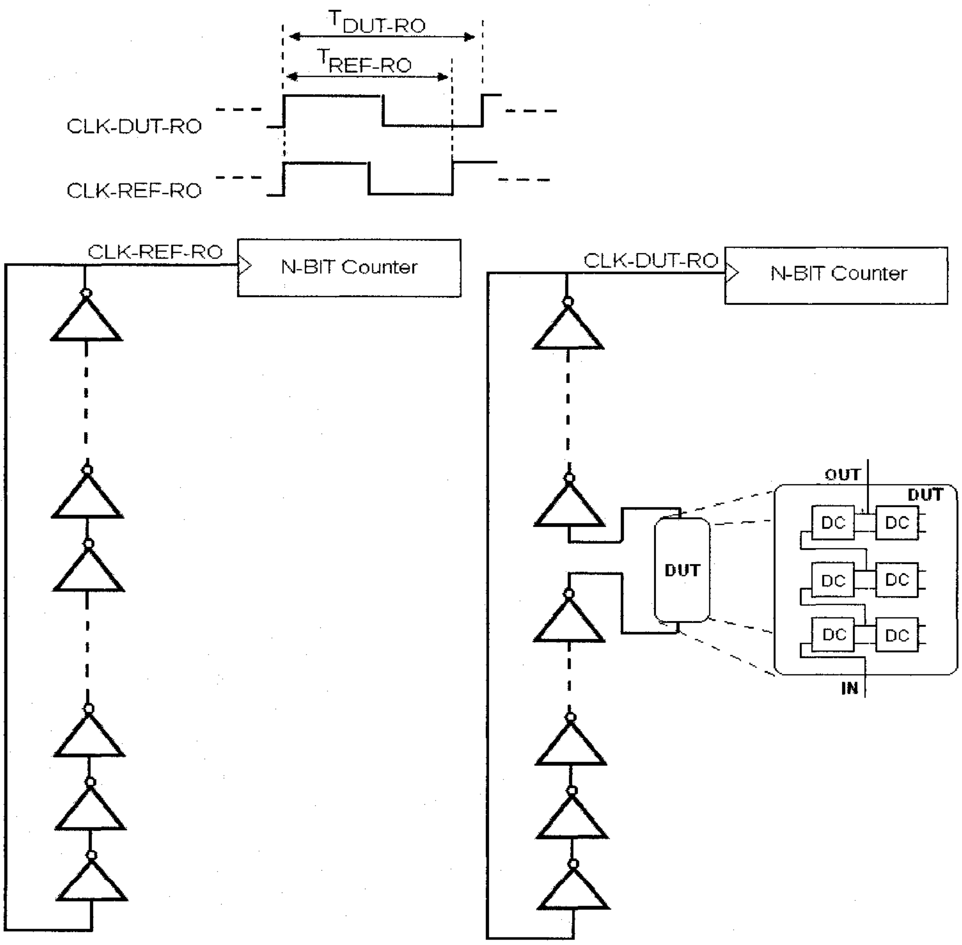


Figure 4.25: Cell delay estimation for vertical propagation

With the relatively smaller values of standard deviations for delay mismatch along the target area of the FPGA surface that were obtained from full matrix calibration scheme, a partial calibration scheme was embedded into the delay matrices as shown in Figure 4.26. Contrary to reconfiguration of all the matrix lines into *RO* in calibration mode, under partial calibration unit separate *RO*-configured vertical and horizontal lines are added to the delay matrices unit as shown in Figure 4.26 without any side effects on the structure of the matrices.

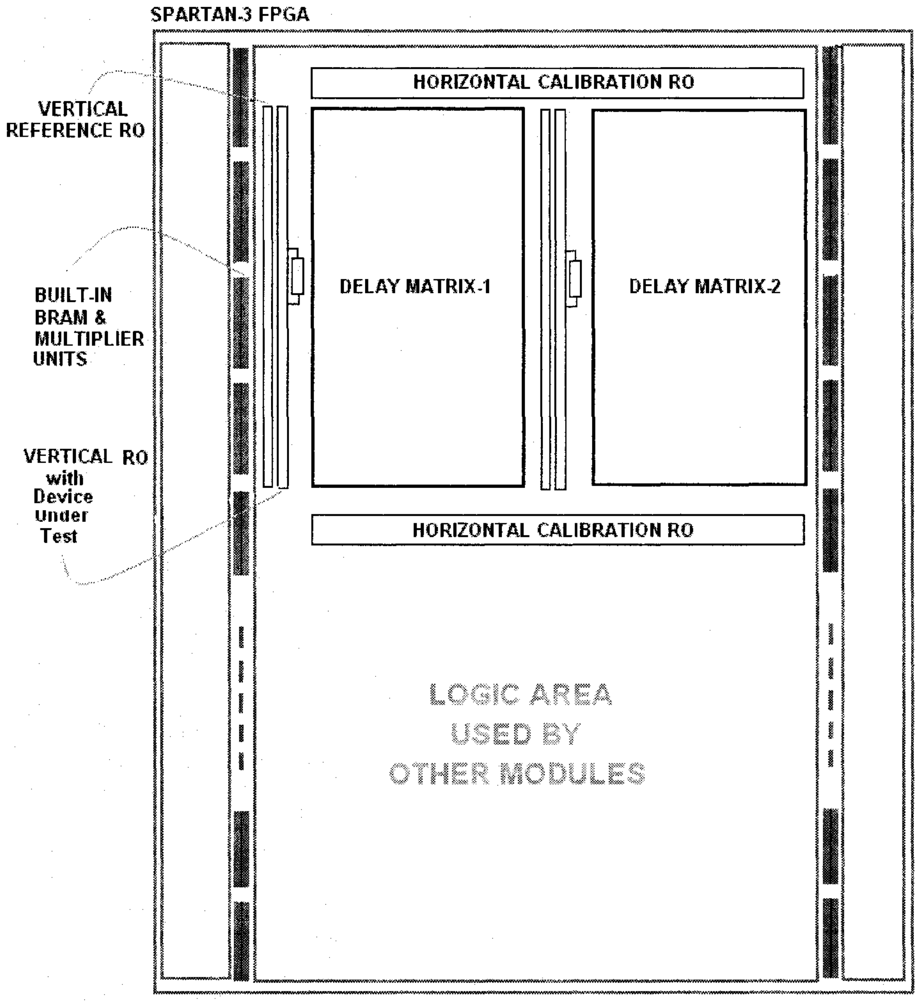

SPARTAN-3 FPGA
VERTICAL REFERENCE RO
BUILT-IN BRAM & MULTIPLIER UNITS
VERTICAL RO with Device Under Test
HORIZONTAL CALIBRATION RO
DELAY MATRIX-1
DELAY MATRIX-2
HORIZONTAL CALIBRATION RO
LOGIC AREA USED BY OTHER MODULES

Figure 4.26: Partial calibration lines of the TDC system on FPGA surface

Therefore, using the embedded calibration lines the desired delays of the components are estimated to ultimately be used to determine the resolution values of the delay matrices.

Calibration data acquisition and display: The operation of the TDC system is switched between calibration and normal modes using an external mode input. The system control continuously verifies for the status of the mode input while in waiting state and accordingly follows the appropriate flow. Handling of the normal operation of the TDC system by the control unit was discussed in section 4.3.10. The steps required to obtain data from the calibration topology discussed above are as follow:

Each calibration run has been divided into two sequential phases with each phase dedicated to acquire calibration data from specific calibration lines. In *phase-1*, the obtained calibration data come from the two vertical REF-RO and the two horizontal calibration *ROs* (upper and lower) composed of the buffer lines of the calibration VDL. *Phase-2*, on the other hand, covers calibration data from the two vertical *RO-DUT* lines and the upper and lower horizontal latch-based *RO* lines. A single calibration run is pictorially described in Figure 4.27.

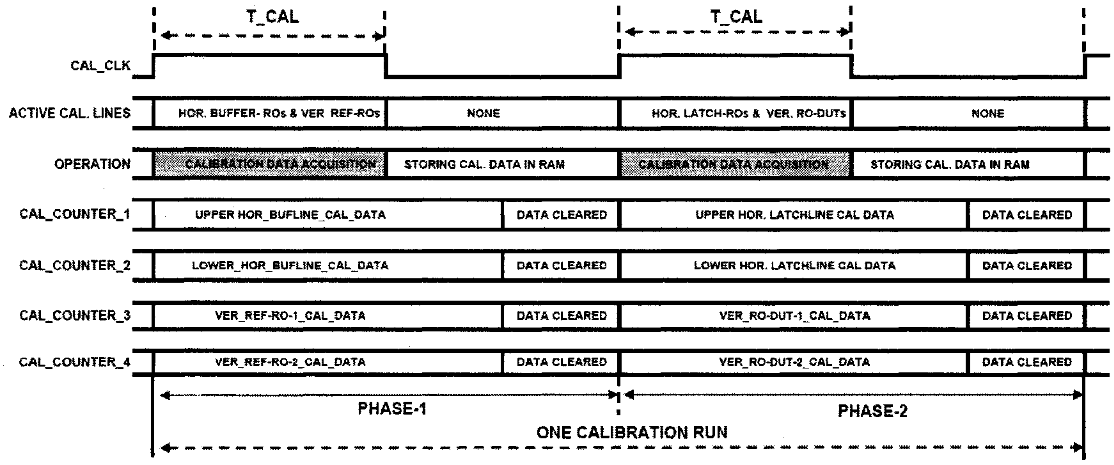


Figure 4.27: A single calibration run divided in two phases

Similar to normal operation, the calibration steps are handled by means of two FSMs, namely the *CAL_CONFIG* and the *CAL_STORE. CAL_CONFIG* comprises the states required to determine the active calibration lines (horizontal buffer/latch lines and vertical REF_RO/RO_DUT) for the current phase and accumulate corresponding calibration data. The steps used to store/display calibration data in/from the calibration RAM are encompassed in *CAL_STORE* states. When in calibration mode, calibration data obtained from RO-based partial calibration logic discussed earlier is stored into the calibration data RAM. Both FSMs interact with each other using common handshake signals. Upon completion of calibration data storage for the desired number of calibration trials, the data are displayed on the LCD display. Below are the ASM charts for *CAL_CONFIG* and *CAL_STORE* FSMs.

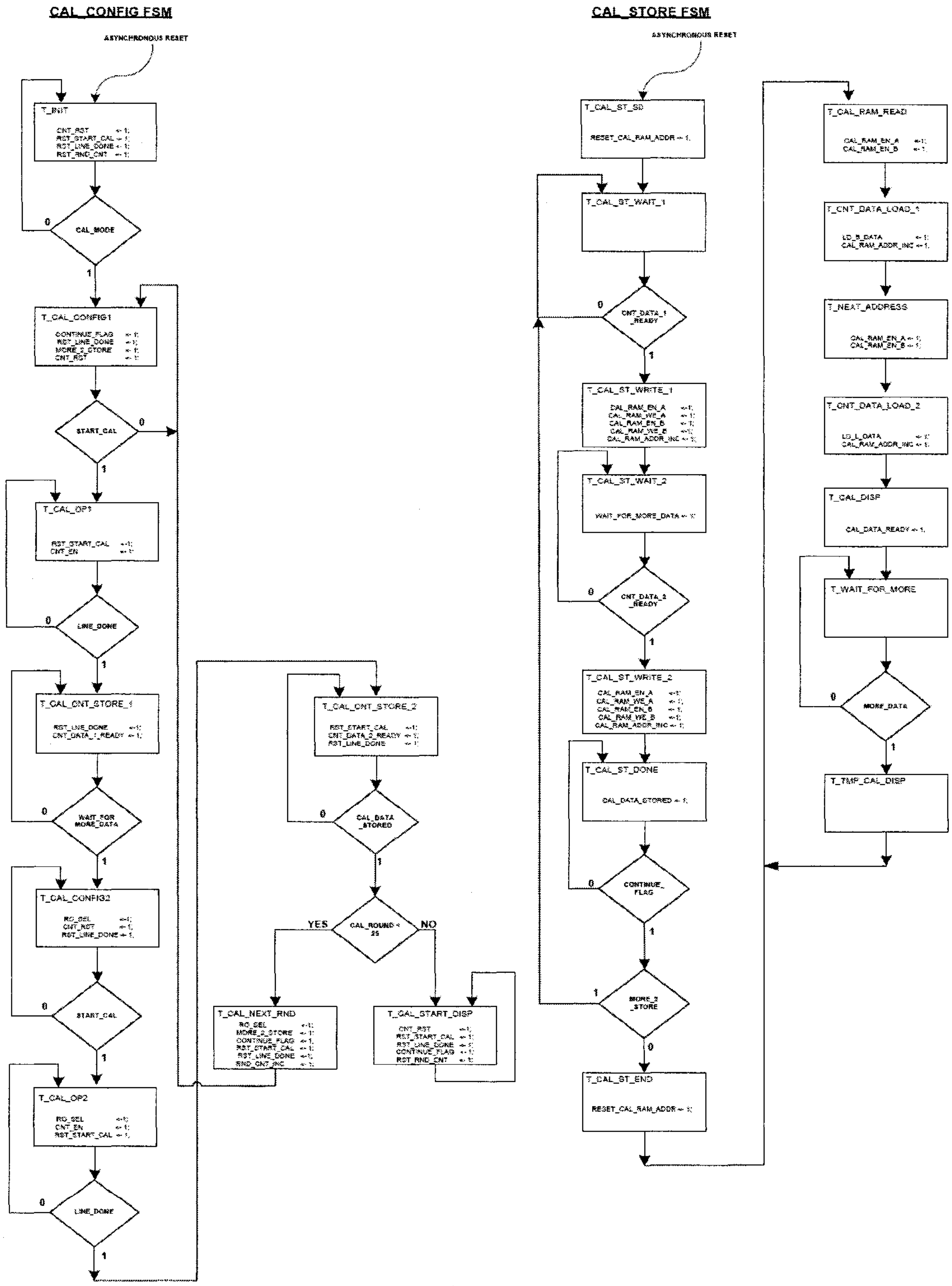


Figure 4.28: ASM charts for calibration operation involving CAL_CONFIG and CAL_STORE FSMs

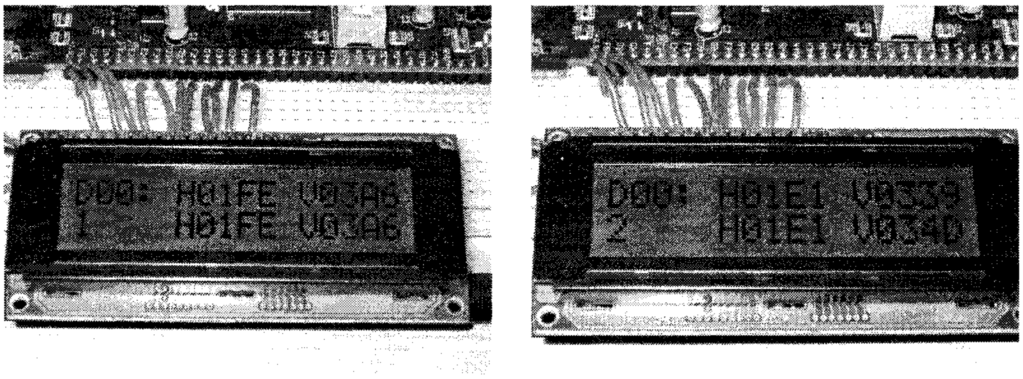


Figure 4.29: Sample calibration data display

Typical calibration data display for a first calibration run is shown in Figure 4.29. The displaying of the calibration data is also done in carried out in two phases. In the first phase (left picture in Figure 4.29), the display shows calibration counters' data from the two buffer-based horizontal calibration lines for horizontal buffer delay estimation and the data from two vertical reference ring oscillators. After a short time, the data switches to calibration data (shown on the right side in Figure 4.29) from the latch-based horizontal calibration line for horizontal latch delay estimation and the data from the vertical ring oscillator with the device under test.

4.4.2 Analytic method of Calibration

Another proposed calibration method for determining the horizontal and vertical resolutions of the TDC system takes an analytical approach. This method involves applying known test intervals of equal magnitude, say T_{CAL}, and manipulation of eq. 4.1 for pulse arrival times to obtain a system of equation with two unknown quantities, namely, τ_Y and τ_X. Consider the waveforms shown in below:

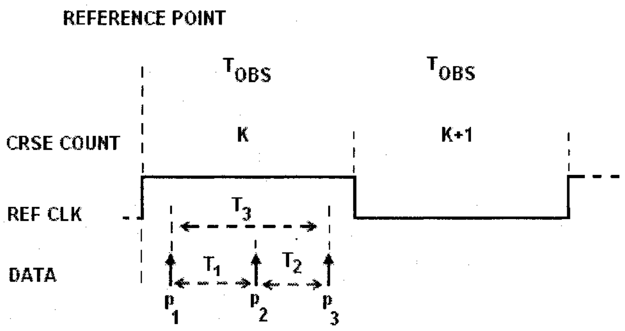


Figure 4.30: Equidistance pulses within the same time window

Using eq. 4.1, the arrival times for incoming pulses 1, 2 and 3 are given as:

$$t_{e1} = (k+1)\, T_{OBS} - T_{e1} = (k+1)\, T_{OBS} - (n_1\, \tau_Y + m_1\, \tau_X) \tag{4.9}$$

$$t_{e2} = (k+1)\, T_{OBS} - T_{e2} = (k+1)\, T_{OBS} - (n_2\, \tau_Y + m_2\, \tau_X) \tag{4.10}$$

$$t_{e3} = (k+1)\, T_{OBS} - T_{e3} = (k+1)\, T_{OBS} - (n_3\, \tau_Y + m_3\, \tau_X) \tag{4.11}$$

From the diagram and eq. 4.9 to eq. 4.11, we get:

$$T_1 = t_{e2} - t_{e1} \tag{4.12}$$

$$T_3 = t_{e3} - t_{e1} \tag{4.13}$$

Replacing t_{e1}, t_{e2}, and t_{e3} by their values will yield, after some arithmetic manipulations:

$$\tau_Y = [T_1\, (m_1 - m_3) - T_2\, (m_1 - m_2)] \,/\, [(m_1 - m_3)\, (n_1 - n_2) - (m_1 - m_2)\, (n_1 - n_3)] \tag{4.14.a}$$

$$\tau_X = [T_1 - \boldsymbol{\tau}_{\mathbf{Y}}\, (n_1 - n_2)] \,/\, (m_1 - m_2)] \tag{4.14.b}$$

$$\tau_X = [T_1\, (n_1 - n_3) - T_3\, (n_1 - n_2)] \,/\, [(n_1 - n_3)\, (m_1 - m_2) - (n_1 - n_2)\, (m_1 - m_3)] \tag{4.15.a}$$

$$\tau_Y = [T_1 - \tau_Y\, (m_1 - m_2)] \,/\, (n_1 - n_2)] \tag{4.15.b}$$

If the data input is sources from a periodic signal with period T_{CAL}, eq. 4.14 and eq. 4.15 are reduced to:

$$\tau_Y = [T_{CAL}\, (2m_2 - m_1 - m_3)] \,/\, [(m_1 - m_3)\, (n_1 - n_2) - (m_1 - m_2)\, (n_1 - n_3)] \tag{4.16.a}$$

$$\tau_X = [T_{CAL} - \tau_Y (n_1 - n_2)] / (m_1 - m_2)] \qquad (4.16.b)$$

$$\tau_X = [T_{CAL} (2n_2 - n_1 - n_3)] / [(n_1 - n_3)(m_1 - m_2) - (n_1 - n_2)(m_1 - m_3)] \qquad (4.17.a)$$

$$\tau_Y = [T_{CAL} - \tau_X (m_1 - m_2)] / (n_1 - n_2)] \qquad (4.17.b)$$

From eq.4.14 to eq. 4.17, it is seen that the vertical and horizontal resolution values can be determined once the required pulse coordinates are found experimentally. It is to note that while this method seems ideal, it may not provide accurate estimates due to the undesirable effects such as clock skew, delay mismatch, and jitter in reference clock period. Those effects influence the linearity of the TDC operation, making the proposed analytical calibration selective. That is, the method may produce good results based on select number of test intervals and the matrix coordinates obtained.

4.5 Chapter Summary

This chapter mainly discussed the theory and principle of operation for the proposed TDC circuit. The core part of the TDC circuit is the time interpolator that was implemented using two matrices of delay cells operating in parallel. The matrices can also be seen as arrays of buffer-latch VDL with the interconnection between successive rows achieved through the first delay cells of each line and a common reference clock line shared by all lines. The delay matrix measures the distance in time between the rising edges of the data and the subsequent clock signals applied at the respective data and clock inputs of the matrix. The matrix structure breaks this interval into two-level time bins of $\tau_Y = t_L$ and $\tau_X = t_L - t_B$ with t_B and t_L as the propagation delays of the buffer and latch elements of the matrix delay cells. The vertical interconnection provides a propagation path with larger time bin τ_Y whereas the single-shot high resolution τ_X is achieved by horizontal propagation throughout the Vernier delay lines. The TDC circuit measures an interval of time between two incoming pulses by taking the difference in corresponding arrival times with respect to some reference point.

In addition to the fine time interpolator, a binary counter is used as the coarse counter to increase the measurement range of the TDC system. Similarly other supporting modules include a data encoder to encode the tap line readings out of the delay matrices, a FIFO and Data RAM for storage of measurement data, a shifter module to help transfer data from the FIFO to the data RAM, and an interface driver for the LCD Display that is used to display the measurement data. The whole system is controlled by an FSM-based control unit. The several clock signals are provided by the digital clock manager primitive module within the Spartan$^{\text{TM}}$-3 FPGA.

The next chapter discusses the various important issues and considerations relative to the design and implementation of the TDC system. The chapter will touch upon logic mapping and placement onto the FPGA surface, design constraints, and important performance limiting factors.

CHAPTER 5 - ISSUES AND DISCUSSIONS

5.1 Logic Placement onto FPGA surface

The standard procedure for creating and implementing a design for programmable logic devices such as FPGA and CPLD devices comprises various steps known as the design flow. The design flow consists of design entry and synthesis, design implementation, and design verification. Since the FPGA platform used in this project is based on a Spartan-3 FPGA from XILINX, the discussions that follow are XILINX-specific.

In the *design entry and synthesis* step, designers begin with a conceptual description of the design under study by means of combining schematic components using a graphic interface or by means of functional behaviour described in hardware description language (HDL) using a text editor interface. The functional description using either of the two methods is synthesized (synthesis is the process of converting the high level of abstraction described in HDL or schematic into lower level library logic primitives) to obtain a design netlist. A netlist is nothing but a text description of circuit connectivity comprising instances, nets, and some attribute information. The created netlist is the input to NGDBuild program that performs the necessary steps to create a Native Generic Database (NGD) file describing the logical design in terms of XILINX logic primitives. Logic primitives are the simplest design elements in the XILINX libraries.

Design implementation comprises the steps required to *fit* the synthesized logical design into a specific XILINX FPGA. The process starts by *mapping* the logical description of the design contained in NGD file generated in the synthesis step to the physical components within the target FPGA device. The output of the map process is an NCD (Native Circuit Description) file containing a physical description of the design in terms of the primitives of the target FPGA device. The next step is to place and route the design within the target FPGA device. In the placement step the design primitives in the

NCD file are placed in specific locations within the target device. Next is the routing step where the interconnections among various logic primitives are defined. The place and route operation is achieved by XILINX PAR tool. The output from PAR is a new NCD file defining the placement and routing description for the design on the target device. The last step in the implementation is the bit file generation where a file containing the FPGA configuration file in binary is generated for the NCD file from place and route step. The bit file can then be downloaded into the FPGA through a software program interface (IMPACT from XILINX) to configure the FPGA device for the desired design.

Design verification is the process of testing the functionality of the design. Design verification can be performed at various levels of the design phase in the form of simulation (functional, timing), static timing analysis (using Timing Analyzer), and in-circuit verification through custom test cases using ad-hoc test methods.

5.1.1 Design Optimization

Design optimization for FPGA designs is achieved by means of constraints applied at various levels of the design flow. A constraint introduced during the synthesis step may produce an NGD file that is optimized for the intended behaviour in terms of the logic primitives. Similar constraints can be applied at map, place, and route processes. Map constraints may define specific logic primitives within the target FPGA device to be used for the low level logic defined in the NGD file from the synthesis step. Placement constraint may define particular logic element at a particular location of the target device to be used, whereas the routing constraints may define desired net interconnections among logic elements out of many available interconnections possible. The form of constraints to be applied is, to some extent, design-dependent. For instance, if the overall system clock frequency is a designer's main concern, a global constraint for the system clock period can be introduced. In such cases the automatic place and route process is performed based on the given constraints. On the other hand, some designs may require a local constraint by defining the exact placement and routing of a logic element at a particular location of the device for proper functionality. Under such circumstance, the designer needs to place and route the design manually.

For the TDC design under study, it is very important that the placement of logic elements for the interpolator be carried out manually. This is due to the equally distant time slices required for fine time bins provided by the delay cells in the Vernier Delay Lines of the matrix. The next section discusses the various synthesis and mapping constraints that are vital for the operation of the TDC circuit.

5.1.2 Why Manual Logic Placement?

As pointed out in the previous section, manual logic placement may be a necessary step for some components of time constrained applications such as the delay matrix. The TDC matrix requires uniformly time spaced clock and data signals throughout its architecture in order to properly operate with the desired measurement resolution. Automatic placement by the commercially available synthesis tools may physically map and place delay components in any place within the FPGA, depending on the optimization option and the algorithm used. Similarly, automatic routing may route the interconnections among logic components with unpredictable delays that may not be acceptable for particular applications. Therefore, automatic place and route options are of no use in applications requiring uniform and predictable delays such as the TDC matrix. In such cases, the designers have no choice but to go through the (tedious) job of manually placing and routing the design. Still, it may be hard to achieve the desired/expected delays and performance despite the tedious manual placement and routing. The more timely critical the application, the more challenging is the FPGA implementation. In the current work, a resolution of ~(100-120) ps was targeted and achieved by manually placing each delay cell of the matrix in order to attain the required uniform and predictable delays for the latch and buffer components of each Vernier cell. Many trial and error rounds are required in order to find out an optimized placement topology that would fulfill the desired routing and component delays for high resolutions. In fact, the measurement resolution using FPGA circuits are limited and not any desired resolution may be achieved due to the fixed delays of the pre-determined routing paths among the logic block components within the FPGA.

5.1.3 Directional-Dependent D-latch Propagation Delays

By design, a signal propagates throughout the delay matrix in vertical and horizontal directions. In section 4.2.3, the vertical and horizontal propagation delays through the latch elements were assumed to be the same. Within the FPGA surface, however, this assumption may not be correct, since the routing in different directions of the FPGA surface may show different propagation delays. Simulation and experimental results (see section 6.2) for the chosen placement topology of the delay matrices show that the propagation delays through the D-latch vertically is slightly different than the horizontal direction due to the different routings. Thus, it is more correct to change the term in the numerator of eq. 4.6 from t_L to τ_Y.

5.1.4 Delay Matrix Constraints

In order to meet the specified measurement resolution, for each delay cell of the delay matrix a single CLB is dedicated to contain both the LUT-based buffer and the latch elements. Furthermore, adjacent CLB components within the FPGA architecture have been chosen to map consecutive delay cells of the lines. This way the buffer and latch elements are uniformly distant from each other along any delay line providing equal logic delays along the clock and data signal paths. The schematic in Figure 5.1 shows this placement scheme.

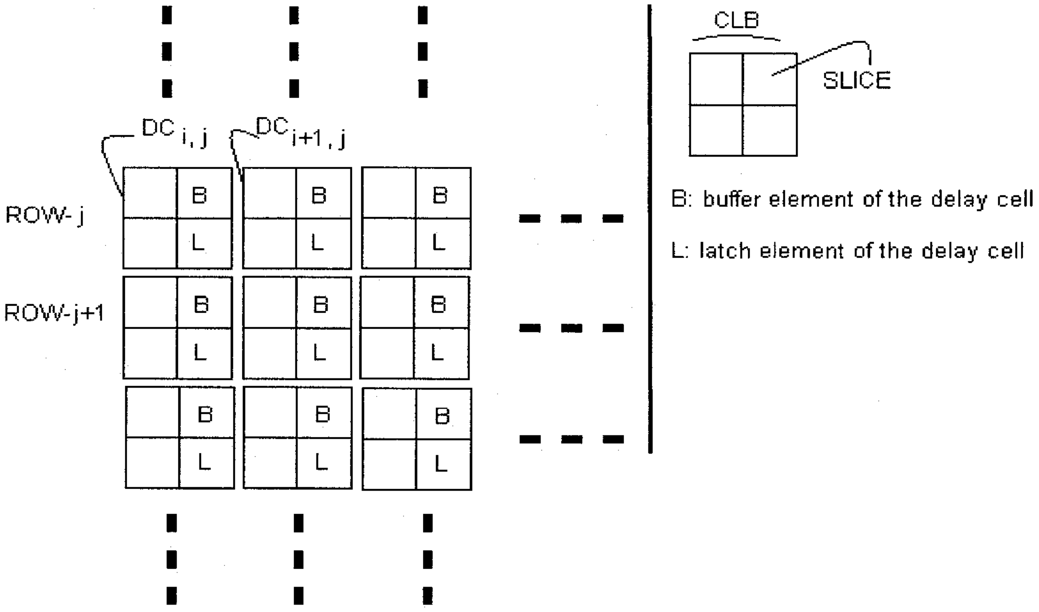

DC i, j
DC i+1, j
ROW- j
ROW-j+1
B
L
CLB
SLICE
B: buffer element of the delay cell
L: latch element of the delay cell

Figure 5.1: CLB placement scheme for delay cells of the delay matrix

The achievement of various optimizations such as the delay cell placements are made by means of constraints in the design. Design constraints can be specified either in a user constraint file (UCF) containing all the constraints and attached to the design project or in the HDL design file using constraint syntax. The synthesis, and place and route tools consider all the constraints contained in the UCF or the design file while performing the respective operation of synthesis, placement and routing.

The placement scheme described previously was achieved using the "*LOC*" constraint. "*LOC*" is a basic placement and synthesis constraint that defines an absolute location within the FPGA die where a design element can be placed. Although the "*LOC*" constraint defines the location of the logic elements within the FPGA, the choice of which specific primitive within a slice remains up to the tools. To remedy this problem additional constraints need to be applied. Choosing a particular element within the slice is important due to the uniform routing delay required among the delay cells. If different slice elements are used for logic components in a delay line, there may be slight difference in routing delays from one CLB to the other, depending on the particular input/output pins of the used slices. Using the same element within the slices for all delay cells of the lines may infer the same net routing and consequently uniform inter-cell delays composed of logic and routing delays may be obtained. Therefore, specific placement constraints are to be used to define the basic elements within a slice to be used.

"*BEL*" is an advanced placement constraint that has been used to select a basic element in a CLB slice such as a flip-flop or a LUT to be mapped to a design element such as a latch or a buffer component. For instance, to map a design latch, the designer has the freedom of choosing either *FFX* or *FFY* within a slice. Similarly, for logic functions, either of the F or G function generators (LUT) can be specified. If "*LOC*" is used alone for component placement, the tool will still have the freedom of choosing either of the available logic or FF elements within a slice. Hence, using the combination of *LOC* and *BEL*, particular logic element within a slice is selected to map the component of concern. This way the buffer and latch elements of the delay cells follow identical placements, having almost uniform inter-cell routing.

5.1.5 Other Constraints

In addition to the constraints used for the placement of the delay cells in the interpolator in order to achieve predictable uniform logic and routing delays, other design constraints have also been used. One of such constraints is the "*KEEP_HIERARCHY*" constraint that prevents the tools from flattening the design in the optimization step. The *XST* (*XILINX Synthesis Tool*) attempts to flatten (removing logic that seem redundant to *XST*) the design by optimizing the module boundaries. "*KEEP_HIERARCHY*" is a synthesis and implementation constraint that helps maintain design hierarchy for various modules in the design. In designs such as the TDC system, it is important to keep the hierarchy of the system modules such as the delay matrix intact.

Another important constraint widely used in several modules of the design is "*KEEP*". "*KEEP*" is also an advanced mapping and synthesis constraint that prevents absorption of the nets into a logic block. If two logic elements are mapped onto a single logic block of the FPGA, the interconnecting signals may be absorbed and no longer visible in the physical design database. One discomfort of signal absorption is the inability to track the signal in simulation as it is no longer visible. Another discomfort occurs in logic implementation where, in the absence of the signal, the intermediate logic may be undesirably optimized out. For instance, a buffer element is nothing but a delay in the signal path. If the input/output signals of a buffer are not constrained with "*KEEP*" or other similar existing constraints, the intended buffer element will be seen as a delay and will be optimized to a single net. "*KEEP*" has been extensively used throughout system modules to prevent undesirable optimization and improve signal traceability in simulation.

In addition to synthesis, map, and placement constraints, some critical parts of the system are given timing constraints. Timing constraints are guidance for the *PAR* (*place and routing tool*) to select routings with minimized net delay and clock skew. Handling skew for clock and other critical signals in the design is as vital as the placement of the logic elements of the interpolator. From the design step in this document, it was seen that the delay matrices are equipped with back resetting modules that serve to reset an earlier row once the data pulse makes its way to the subsequent one. It is very important that the constant reset signal reaches the reset input of the delay cells in the earlier row either at the same time or with a minimized skew among the various destination points. One way to control this is to apply the "*MAXSKEW*" constraint to all inter-row reset signals. The *PAR* will try to route the reset signals in a way as to minimize the signal skew among various destination delay cells.

Another timing constraint used in the system is "*MAXDELAY*" which sets a maximum delay for a net. This constraint in combination with the buffering technique has been used to control the clock path to the destination delay matrices from the clock source. Within the FPGA die, there are various routing possibilities to route one logic cell to the other. For every synthesis and implementation step the *PAR* may route two logic cells in different manners. This process is costly as far as routing of the clock is concerned. The two delay matrices in the interpolator use two 180 degree out of phase reference clock signals. In theory the assumption was that the clock signals reach the matrices on the die at an exact instance of time. In practice this is not the case. Depending on the exact location of matrices on the FPGA die, the clock signals may reach the entry points at different instances of time. The larger the gap in arrival time between the reference clock edges at the matrices entry points, the more error prone will the system be. Therefore, it is vital to somehow control the delay from the global clock buffers to the matrix entry points.

To minimize clock skew, FPGA device manufacturers usually embed specific clock nets to feed the clock inputs of logic blocks. In designs such as the delay matrix, where the clock signal feeds components other than a latch/FF, the clock skew is relatively high, heavily affecting the expected behaviour. Under such circumstances, either special design measures need to be taken to exploit the existence of special clock nets or other means of controlling the clock skew need to be implemented. The solution used in this work is to subdivide the clock paths from the clock sources (global clock buffers) to their respective destination points by means of buffer insertion. By placing the intermediate buffer elements on the clock paths in a topology such that every resulting sub-segments are of equal net delays, the two clock signals are forced to propagate through equidistant net paths reaching the matrices clock entry point at equal time. The net delays were analyzed using the static timing analyzer and the FPGA Editor tools included within the XILINX ISE suite.

With the problem of clock net delays for both out of phase reference clock signals solved, there yet remains another problem of clock skew among the various lines of the matrices. In theory the clock signals are assumed to reach the various delay lines of the matrices at equal points in time, which in practice is not achievable. This problem has been addressed by means of tight logic placement for the last clock buffer before the matrix entry point and the "*MAXSKEW*" and "*MAXDELAY*" constraints applied to the clock signals of the matrices. After observing equal delays for all the sub-segments of the clock paths and minimized clock skew among the clock signals of the delay lines in the matrices, *directed routing constraints* have been applied to the nets in order to keep the same routing topology even if the design is re-synthesized. *Direct routing constraint* is a way of keeping an achieved desirable routing intact throughout many synthesis and implementation runs. The routing will remain the same as long as the logic placement for the elements of concern remains the same.

Lastly, it is worth mentioning that group constraints have also been applied to place the system modules in specific areas of the FPGA die. The area group constraint reserves a range of CLB locations for a module to which it is attributed. In the TDC system, separate group constraints have been dedicated to system modules. In this way their associated logic can be combined together in specific areas on the FPGA die, achieving a more modular logic placement for the system. The group constraints appear as rectangles in the floor planned view of the design in APPENDIX-II, occupying a range of CLB.

5.2 Design & Implementation Limiting Factors

Design and implementation of any electronic system comes with some sort of associated limitations and issues. While some limitations may be easy to overcome, some others may turn into a major challenge for the designers to conquer. The TDC system under study as well demonstrated some limitations during the design and implementation phases. These limitations are discussed in the following subsections.

5.2.1 Skew In Clock Lines

Discussed in Chapter 3 - , the existence of skew in the clock lines of a system is a major performance limiting factor in synchronous systems. The delay lines of the interpolator module share a common clock signal line. This requires a careful skew-free routing of the clock signal to all the delay line entry points. Since the current design is implemented on an FPGA platform, managing of clock skew may be harder than an ASIC implementation due to the fixed architecture of the FPGA devices. The designated global clock lines within the FPGA devices usually drive the clock input of the clocked elements in a CLB or in an IOB. Driving combinational logic adds load capacitance to the lines that causes clock jitter and skew along the lines. Therefore in designs such as the delay matrix where the clock signal feeds components other than clocked elements, the elimination of clock skew is almost unachievable. However, using tight placement constraints along with some buffering techniques a minimized skew along the clock lines may be obtained.

The existence of skew along the common clock path of the delay matrix causes enlargement/shrinkage in the ideal value ($\tau_Y = t_L$) of the larger bins in the delay matrix. Consider two sequential rows R_i and R_j ($j = i + 1$) of the delay matrix. The vertical time slot (τ_Y) achieved by data propagation from R_i to R_j is enlarged when the clock line on R_j suffers from negative skew. With a negative skew value of Δt on R_j, the distance between the data and the clock edges further decreased by Δt leading to a corresponding vertical time slot value of $\tau_Y = t_L + \Delta t$. Similarly with positive skew along the clock line of R_j, the distance between the propagating data and the clock edges is further increased by Δt leading to a time slot value $\tau_Y = t_L - \Delta t$. This enlargement/shrinkage in the value of τ_Y causes a shift in the expected final pulse locations, ultimately bringing about non-linearity in the transfer curve of the converter. This scenario is shown in Figure 5.2 where the negative skew on row-1 causes enlargement in the bin value corresponding to row-0. Similarly, existence of positive skew on row-3 causes shrinkage in bin value corresponding to row-2.

The interline skew for the clock signal in the delay matrix was minimized using timing, map and placement constraints along with the buffering of the clock path. Using these techniques and after many repeated trials, the results from the FPGA Editor tool show a minimized interline skew ranging between (10-400) picoseconds.

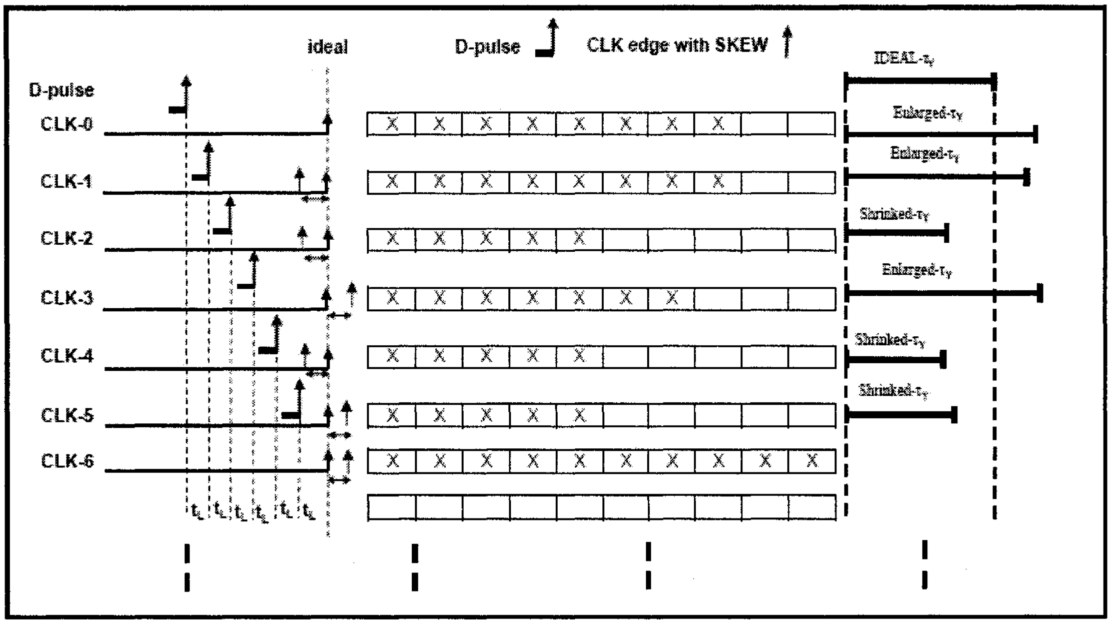


Figure 5.2: Side effects of skew on vertical time bin (τ_Y)

5.2.2 Meta-stability In Flip-Flop Elements

Meta-stability is an important phenomenon to be considered in designs with cascaded flip-flop elements. The meta-stability behaviour of a latch or FF component refers to the malfunction of the FF in terms of stretched propagation delay or an unpredictable final state that is caused by a change in data at the data input in the vicinity of the clock edge. This phenomenon is more apparent in designs with cascaded FF topologies where the change in data at the data input of the second FF, which is also the output of the first FF is more vulnerable to being within the critical short interval from the clock edge.

One way to improve the meta-stability in cascaded FF circuits is to use two or more cascaded FF sharing a common clock instead of a single one [38]. Shown in Figure 5.3, the shaded region represents the timing zone where any change in data will make FF-1 output unstable. FF-1 acts as the synchronizer that makes FF-2 more immune to metastability effects. Any unstable data at the output of FF-1 is tolerated by FF-2 for the duration of up to one complete clock cycle. At time t_0, the changing input data is sampled into FF-1 while FF-2 samples the old value on *Q_int* which is *logic-0.* The output of FF-1 takes some time to stabilize and upon arrival of the subsequent edge of clock at time t_1, the stabilized value is sampled into FF-2. This way the output *Q_OUT* remains unaffected by the changing data on *Q_int*.

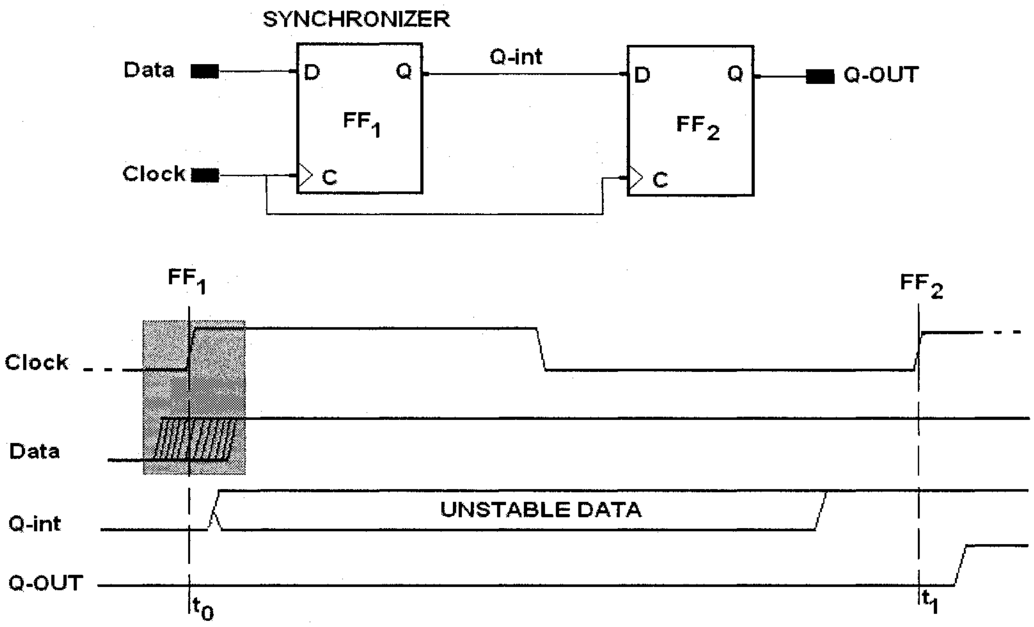


Figure 5.3: Double-FF used to improve metastability

In the matrix topology of the current TDC, however, using double latch components has the negative implications of an increase in size and observation time window given the target resolution. Meta-stability effects in FPGA devices and corresponding test circuits have been investigated in [37] where the meta-stability effects were found to be more apparent in designs seeking higher measurement resolution. Experimental test results from the implementation of the TDC circuit onto the Spatan-3 FPGA platform did not show any misbehaviour due to meta-stability.

5.2.3 Delay Mismatch In Identical Components

In applications such as the TDC circuit under study where a delay chain composed of identical elements are used, delay mismatch among the delay cells of the lines are a major limiting factor as far as achievable accuracy of the results is concerned. Delay mismatch is caused by the variations in physical and electrical parameters of an IC that are the direct result of the environmental factors such as power supply and temperature variations as well as the physical factors such as die-to-die and intra-die (within-die) physical variations. The physical factors of delay mismatch are caused by imperfections in mask, processing and layout. While the environmental factors are highly design-dependent, the physical factors are independent of design, causing inevitable variations in the electrical performance of the integrated circuits [19].

With the major design issues outlined in these sections, it is worthwhile to mention that other design issues such as the availability of two jitter-free clocks with an exact 180 degrees of phase difference and the dependence of DCM locking on input signal frequency range have also been acting as limiting factors throughout the implementation phase. When these out of phase clocks are jittery, there exist overlapped regions where either both clocks are high or both are low (see Figure 5.4). In the former case, both matrices are disabled and any incoming pulse remains undetected. However in the latter case, the same pulse may be detected by both matrices as both of them are active. Therefore, any misalignment in the theoretically 180-degree out-of-phase clocks may either cause loss or multi-detection of the incoming pulses arriving in the vicinity of these clock edges.

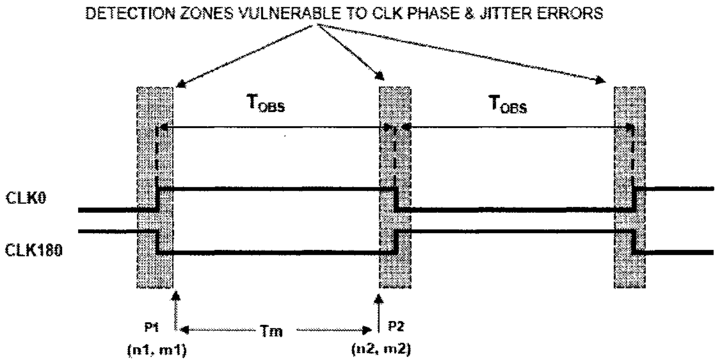


Figure 5.4: Vulnerable zones of the reference clocks for pulse detection

5.3 Chapter Summary

This chapter outlined important considerations and issues regarding the design, implementation, behaviour and performance of the TDC circuit. Despite the advantage of using an FPGA platform for rapid prototyping, this chapter emphasized on the importance of the manual logic placement of delay cell elements for the correct behaviour of the TDC system. Similarly the chapter touched upon performance limitations such as skew in clock lines of the delay matrix, metastability in cascading flip-flops, and the delay mismatch in identical components.

With the important design considerations discussed in this chapter, chapter 6 will encompass the test strategies and methods used in verifying the functionality and performance of the TDC circuit along with the presentation and elaboration of the obtained results.

CHAPTER 6 - SIMULATION & EXPERIMENTAL RESULTS

To investigate the various performance characteristics of the TDC circuit several test strategies applying different test cases in simulation (functional & timing) as well as in hardware using the FPGA prototyping board from XESS were carried out. This chapter encompasses the details of the test strategies used along with the analysis of the obtained simulation and experimental results.

6.1 Test Strategies

6.1.1 Functional Simulation

The TDC system was verified for functional and timing behaviour using ad-hoc testing methods. After completing circuit implementation, the first attempt was to verify the *functional* behaviour of the system. Since the TDC system utilizes delaying process to measure time intervals, un-timed functional simulation was mainly used to individually verify the behaviour of the supporting modules to the system and the overall flow of control from detection to the storage and the final display on the LCD.

The system controller unit was functionally verified for the proper assertion of controlling signals required by other modules of the system. This process involved testing the controller unit with proper test vectors to investigate correct state transition of the various FSM where the controlling signals are asserted. For instance, to verify the temporary storage process of detected data in an active phase of measurement, appropriate test vectors representing the sampling clock and other input signals such as handshake interface signals from the FIFO (*FIFO_FULL, FIFO_EMPTY*) had to be applied through the controller test bench circuit. By observing the expected state transitions the controller unit was tested for generating the control signals required by the supporting modules such as the ENCODER and the FIFO involved in the temporary storage process. Similar test cases were performed to investigate the steps required for permanent storage of data and the displaying process of data on the LCD unit.

On the same token, the FIFO unit was functionally tested for appropriate data read/write operations. Similarly, the encoder unit was verified for properly encoding the tap readings out of the delay matrices using the encoding scheme described in chapter 4. Other modules such as the shifter, the RAM, and the LCD driver units were also individually tested for transferring the data from FIFO to the RAM, and displaying of the stored data onto the LCD display.

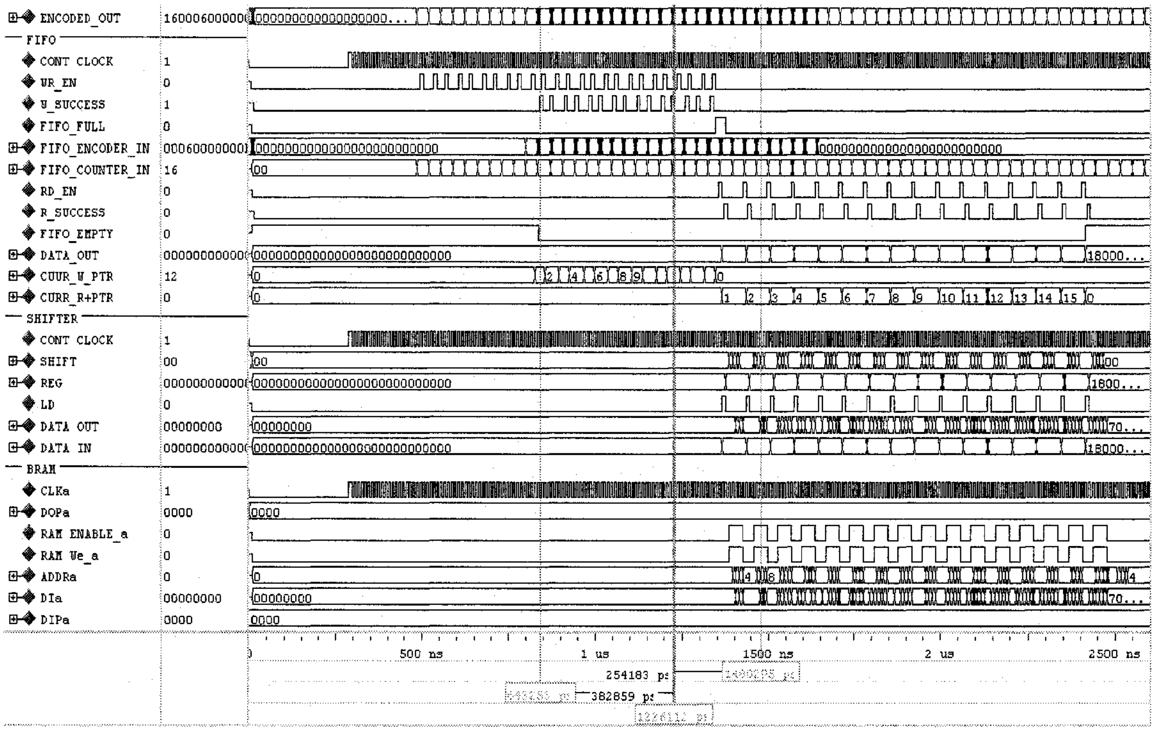


Figure 6.1: Sample waveforms for data storage onto FIFO & BRAM

Being delay-dependent, the matrix itself was tested for correct functional behaviour experimentally. The test intervals in this step was obtained using binary counter output bits clocked with an arbitrary clock frequency of $f_{\mathrm{CLK}} = 1/T_{CLK}$. Each output bit of an N-bit binary counter divides the input clock frequency by 2^{n+1} with n being the output bit in concern having values of *0, 1, 2, ..., N-1*. By selecting different output bits of the binary counter to drive the data input of the TDC circuit, different fixed intervals were obtained. For instance, when the input pulses were distant by 10ns from each other (i.e. when the data input frequency was 100MHz), three distinct pulses were to be detected in the observation time window (T_{OBS}) of 30 ns. Similarly, by using larger inter-pulse time distances, the combination of data acquired by both the coarse counter and the matrix delay cells was tested. Therefore, the TDC was functionally tested by observing the results using several test cases.

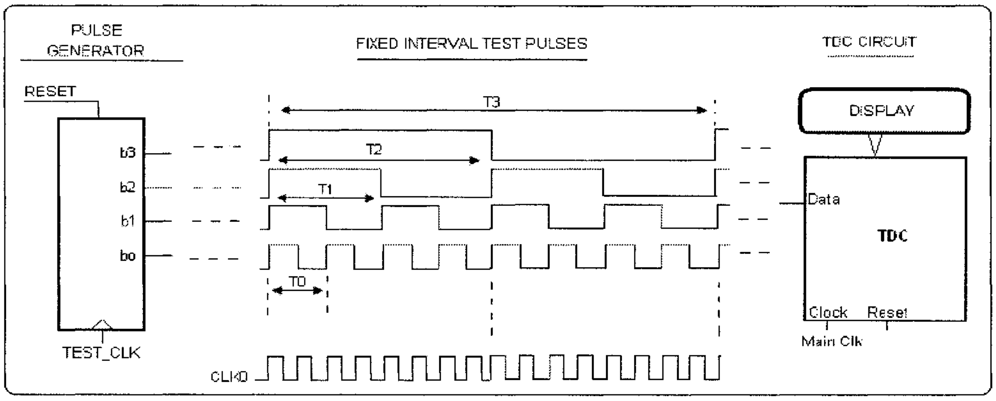


Figure 6.2: Binary counter as fixed interval generator

6.1.2 Timing Simulation

To test the timing behaviour of the TDC system, post place and route simulations were carried out using the MODELSIM simulator. In post place and route simulation the logic and net delays are included in the simulation, leading to a better verification of system behaviour.

Timing simulation for the TDC circuit was important due to the strict timing requirement for the time bins in the delay matrix to be uniformly spaced. Moreover, timing simulation helped find timing constraints such as the arrival of the sampling edge to sample the active matrix tap lines before the start of the next phase of measurement. Additionally, resetting of the matrix tap lines prior to entering an active measurement phase is also critical and had to be verified to assure that no valid data is lost from the matrix tap lines. Data loss may occur if the matrix reset signal remains active longer than expected when the matrix enters into the active measurement phase. This will keep the matrix delay cells reset and ultimately prevent the incoming data pulses from propagating through the matrix delay cells. Above all, since the observation time window (T_{OBS}) mainly depends on cell delays and the minimum interval needed for performing the sample, encode and storage processes prior to entering a new phase of measurement, timing simulation was the first approximating element in determining the latter. Based on timing values observed during timing simulation, the minimum required time window was found to be 35ns.

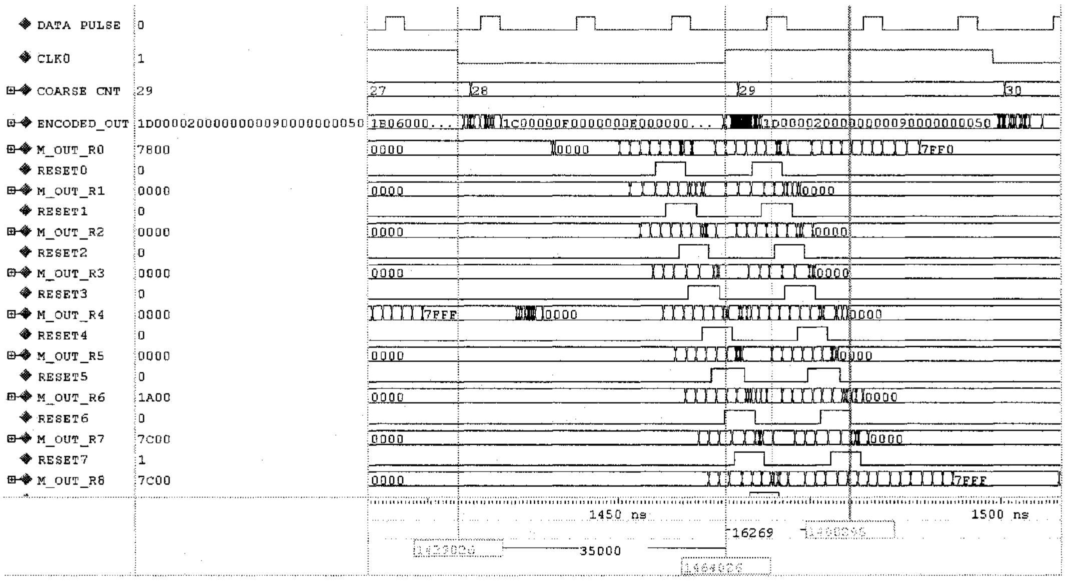


Figure 6.3: Sample timing waveforms for a detected pulse & corresponding encoding

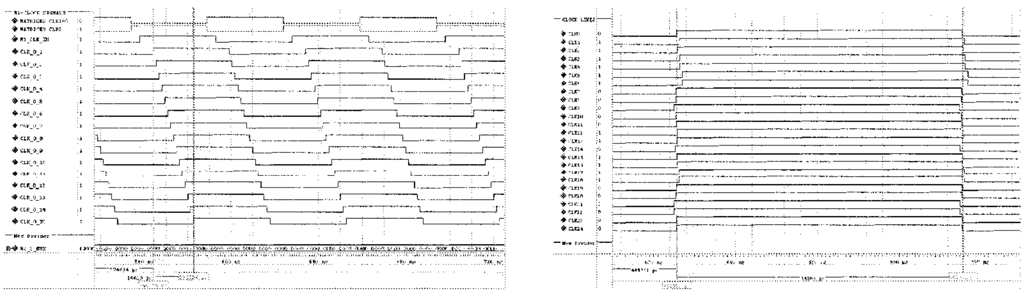


Figure 6.4: Clock delaying waveform and clock misalignment due to skew

6.1.3 Experimental Testing

In addition to simulations, functional and characterization tests were also carried out on the TDC system in hardware. The TDC was tested using fixed and variable –length time intervals with fixed intervals coming from the outputs of the internal DCM clock outputs and a binary counter output bits, and the variable-length intervals generated by an external pulse generator (HP-81130A) .

Test Intervals Measurements: A large number of test intervals of known lengths varying from 8.0ns up to the extended dynamic range T_d (2^{11} × T_{OBS} for 11-bit coarse counter) were measured by the TDC circuit. The diagrams in Figure 6.5 depict interval calculation for the input test intervals.

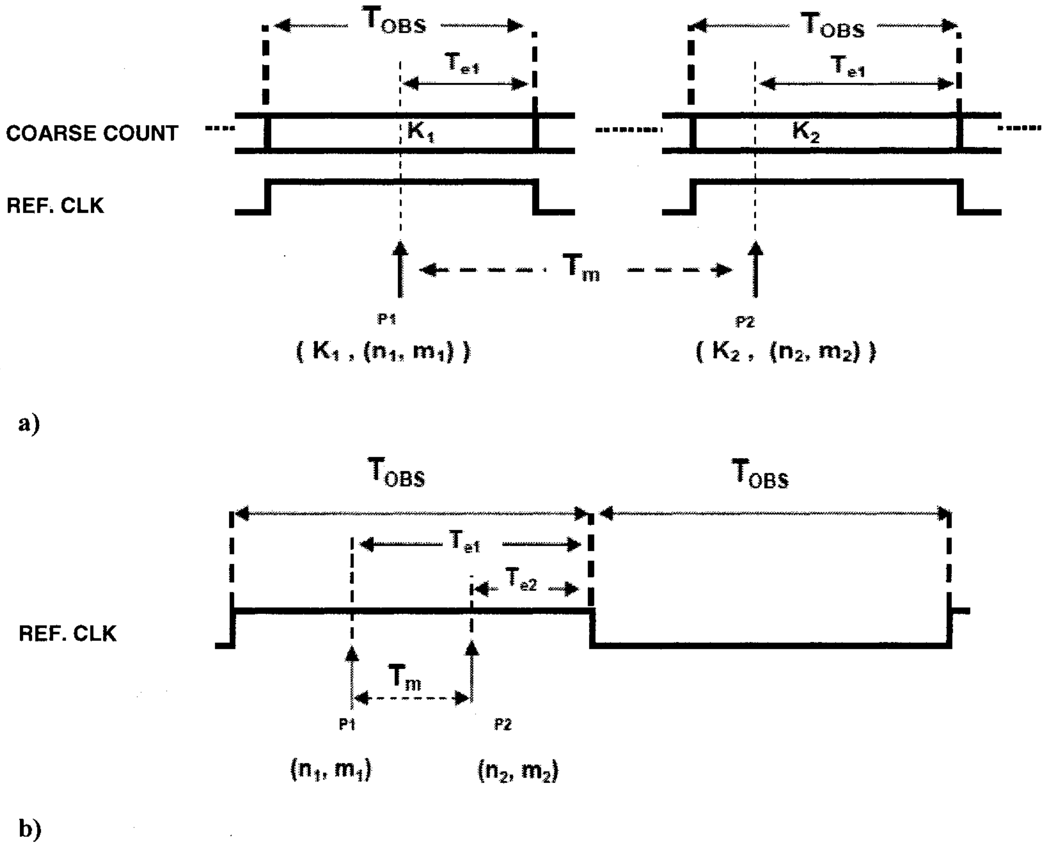


Figure 6.5: Demonstration of interval Calculation a) $T_m > T_{OBS}$ b) $T_m < T_{OBS}$

The calculations for the interval to be measured (T_m) in both cases of Figure 6.5 are given by:

$$T_m = t_{e1} - t_{e2} = \tau_Y (n_1 - n_2) + \tau_X (m_1 - m_2) \qquad T_m < T_{OBS} \qquad (6.1)$$

$$T_m = t_{e1} - t_{e2} = (K_2 - K_1)\, T_{OBS} + \tau_Y (n_1 - n_2) + \tau_X (m_1 - m_2) \qquad T_m > T_{OBS} \qquad (6.2)$$

As seen in Figure 6.5, the distance in time between the two incoming pulses p_1 and p_2 is found from the corresponding arrival times, t_{e1} and t_{e2}. The variables K_1 and K_2 are the respective coarse count values for p_1 and p_2 for the case $T_m > T_{OBS}$. Test results are presented in section 6.2.

Estimation of Component Delays*:* In addition to timing simulation, the propagation delays for the buffer and latch components of a delay cell were estimated using the analytical and RO-based calibration methods described in section 4.4.

Code Density Testing*:* To investigate the non-linearity errors (*DNL* and *INL*) of the TDC circuit the method of Code Density Test (CDT) was applied. CDT is a statistical method for testing data converters based on random data. The random data are obtained by applying a large number of random test pulses to obtain a uniformely distributed histogram showing the frequency of occurrence of each output code within the dynamic range of the converter [39]. Even though CDT is widely used to characterize analog-to-digital converters (ADC), its use in characterizing TDC is also significant.

The application of CDT in TDC testing requires a large number of hit pulses arriving at random intervals of time. By accumulating random hit pulses corresponding to each time bin of the TDC dynamic range a histogram is obtained with histogram bins corresponding to converter time bin. Due to the randomness of the incoming pulse arrival times, the incoming pulses will be uniformly distributed within the dynamic range of the converter with the number of hit pulses accumulated in each bin of the histogram proportional to the actual size of the bin of concern. Once the histogram is obtained, TDC characteristics such as the differential and integral non-linearity parameters can be determined.

Consider the sample histogram for a TDC with 25 time bins. The horizontal bins of the histogram correspond to converter time bins whereas the histogram values represent the number of pulses detected per bin.

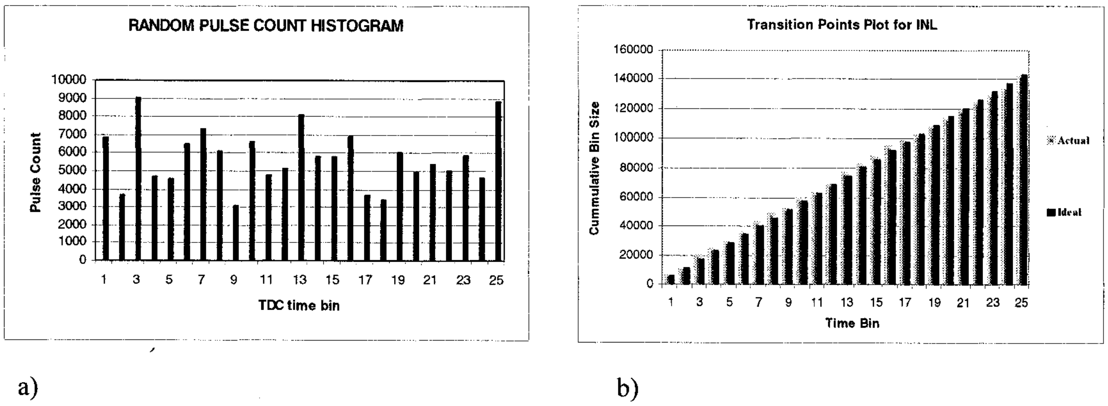


Figure 6.6: a) Data histogram b) Histogram representing transition points

In an ideal converter, the number of hit pulses accumulated in a bin should ideally be equal for all the time bins. Denoting this number as n_s and the total number of detected random pulses by N, the ideal bin weight is given by: $\omega_{\text{-}ideal} = n_s / N$. The actual weight of a bin in the TDC is obtained from the histogram by dividing the actual count values corresponding to a time bin by the total number of pulses applied. That is: $\omega_{\text{-}i} = n_i / N$ for $i=1, 2, 3... 25$. The ideal width of a time bin is given by the ideal bin weight multiplied by the dynamic range T_d ($\Delta t_{\text{ideal}} = \omega_{\text{-}ideal}\ x\ T_d$). The actual time bin width of the converter can then be given by the actual weight of the time bin of concern multiplied by the dynamic range. That is: $\Delta t_i = \omega_{\text{-}i}\ x\ T_d$.

The differential non-linearity of the i$^{\text{th}}$ time bin is given by subtracting one from the ratio of the actual bin weight to the ideal one. Likewise, *DNL* could also be given by dividing the difference between actual and ideal pulse counts corresponding to a bin by the ideal pulse count, that is: $DNL_i = (n_i - n_s) / n_s = [(\omega\text{-}_i / \omega\text{-}_{ideal}) - 1]$. Integral non-linearity which is the deviation of the transfer curve from the ideal case can be obtained from a cumulative histogram obtained by summing consecutive bin sizes in the original histogram (see Figure 6.6.b). The cumulative bin sizes represent transition points, and hence, *INL* can be obtained by subtracting the actual transition points from the ideal ones.

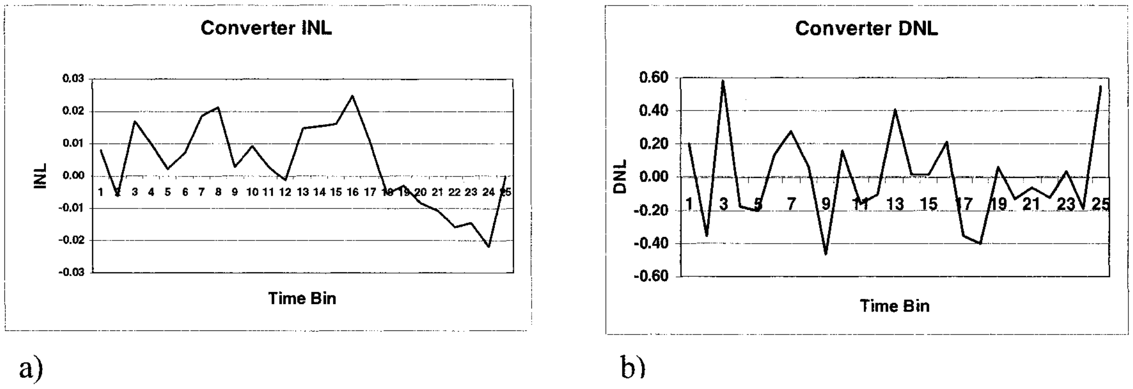
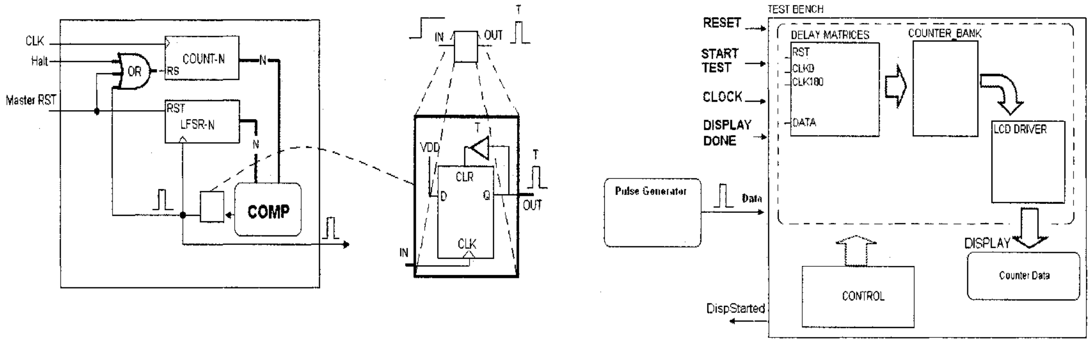


Figure 6.7: a) Sample Converter INL b) Sample Converter DNL

CDT Circuitry: The block diagrams of the circuitry required to carry out Code Density Testing is shown in Figure 6.8. The complete test system consists of two major modules, namely, the pulse generator and the test bench circuitry driven by the former. The test bench circuit consists of the delay matrices under test, a bank of counters from which an n-bit counter is dedicated to a single time bin of the delay matrices, a control unit, along with the LCD driver unit that serves to display data onto the LCD display. The Test Bench circuit assigns a counter for each time bin of the delay matrices unit. A counter corresponding to a time bin is incremented if upon arrival of the sample clock, its count increment input that is driven by the time bin value is high. Hence, a counter is incremented when the incoming pulse reaches that bin before the sampling clock samples the state of the matrix.

Since, the occupied FPGA surface increases with larger counter sizes, the Test Bench was configured to carry out pulse detection in smaller chunks. The number of pulses to be received by the Test Bench is handled by the control unit. When the desired numbers of pulses is received, the test bench notifies the pulse generator through the DISP_STARTED handshake signal so that any further pulse generation is halted until the current counters' data are output to the LCD display for reading. Pulse generation can resume once the current data are registered. The pulse generation continues from the same LFSR value where it was stopped and no pulse repetition occurs. This way by running several trials with chunks of ~30000 pulses at a time, the amount of data required for code density analysis is obtained. The placement of matrix logic cells in the Test Bench follows exactly the same topology as the TDC circuit itself.



a) Pseudo-random pulse generator (PRpG)

b) Test bench circuit

Figure 6.8: Code density test circuit block diagrams

The CDT was applied in two ways. First, a custom pseudo-random pulse generator (PRpG) was designed to apply random pulses. The second method involved sweeping the TDC matrices by pulses distant by ($m\ T_{OBS} + \partial t$) with $\partial t \sim 50\ ps$ and m being an integer. This is actually sweeping of the delay matrices by $\partial t \sim 50ps$ since the direct sweep of 50ps distant pulses is impractical. Both of these methods are described below.

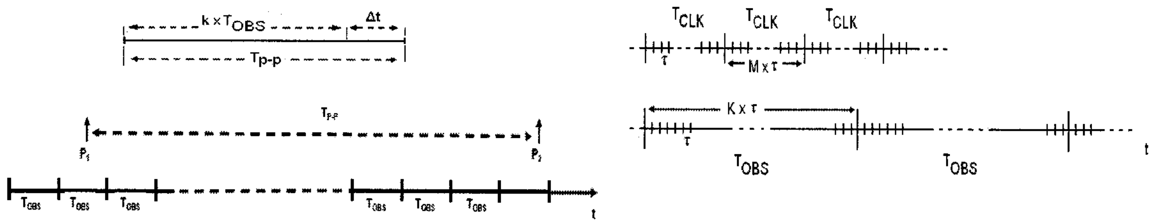
A. *Test Application Using PRpG*: The PRpG is composed of an N-bit binary counter clocked with a clock of frequency $f_{\text{CLK}} = 1\ /\ T_{CLK}$ and an N-bit Linear Feedback Shift Register (LFSR). Further, an N-bit XOR-based comparator is used to compare the binary values contained in both the LFSR and the binary counter and upon detection of a match, a fixed-width pulse is output, the LFSR is clocked for a new value and the binary counter is reset. The Halt input halts any pulse generation for as long as it is kept high. The variable interval $T_{p\text{-}p}$ among subsequent pulses is controlled by the pseudo-randomly changing contents of LFSR, and is given by:

$$T_{p\text{-}p} = (c\ x\ T_{CLK}) \qquad c \in \{1,\ 2...,\ 2^N\text{-}1\} \tag{6.3}$$

The factor c is the cumulated counter value for each run and T_{CLK} is the period of the clock used by the counter. The interval $T_{p\text{-}p}$ is also related to the target observation window (T_{OBS}) by:

$$T_{p\text{-}p} = i\,T_{OBS} + \Delta t = i\,T_{OBS} + k\boldsymbol{\tau} \qquad i \geq 0 \;\; ; \quad k \in \{0, 1, 2..., K\text{-}1\} \tag{6.4}$$

The residue sub-interval $\Delta t = k\boldsymbol{\tau}$ results when the greatest integer multiple of T_{OBS} is removed from $T_{p\text{-}p}$. Since the inter-pulse interval $T_{p\text{-}p}$ depends on the value of LFSR, the value of k will also be pseudo-randomly varying within its domain. However, it is important that k assumes all K values of its domain in order to cover all bins of T_{OBS} with equal probability. In order for k to avoid partial coverage of its domain values, the period T_{CLK} of the clock needs to be defined in terms of τ or a multiple of τ; that is, if T_{OBS} has been divided into K time bins of magnitude τ, T_{CLK} should also be given as $T_{CLK} = M\,\tau$ for some integer value M. This is shown below:



a) Inter-pulse interval $T_{p\text{-}p}$

b) T_{OBS} and T_{CLK} defined in terms of $\boldsymbol{\tau}$

Figure 6.9: $T_{p\text{-}p}$, T_{OBS} and T_{CLK}

With $T_{OBS} = K\,\tau$ and $T_{CLK} = M\,\tau$, the values of Δt will cover all the time bins within T_{OBS} if and only if:

$$GCD\,(K, M) = 1 \tag{6.5}$$

where GCD stands for the greatest common divisor.

Condition (6.5) was algorithmically validated in C++ (see APPENDIX-IV) where all $2^N\text{-}1$ values corresponding to an N-bit LFSR were swept for any missing domain values of k within $\{0, 1, 2... K\}$. For various test values of T_{OBS} each broken into K bins of τ, any period T_{CLK} meeting the outlined condition produced a pulse density uniformly distributed across the bins of T_{OBS}, whereas periods not meeting the latter condition indicated some periodicity in the values of k ultimately not covering all the domain values. Figure 6.10 shows the results of a sample test case for 25000 pulses with K=270, M=59 (T_{OBS} = 27 ns, τ =100ps and T_{CLK} = 5.9ns) and hence GCD (270, 59) = 1.

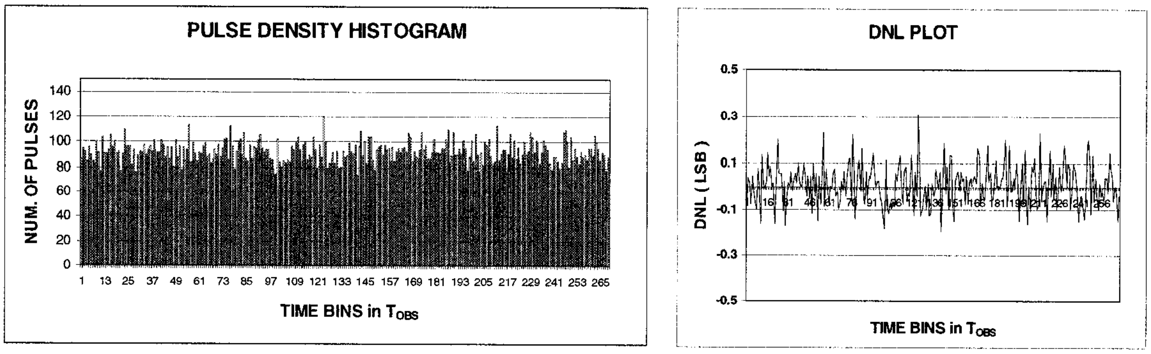


Figure 6.10: Emulation results for pulse distribution of 25000 pulses within T_{OBS} for K= 270, τ = 100ps and T_{CLK} =5.9ns

Therefore, as long as condition (6.5) is met, no correlation between $T_{p\text{-}p}$ and T_{OBS} will exist. For an N-bit LFSR with a primitive polynomial, the counter values will range from 1 to 2^N-1, leading to inter-pulse intervals pseudo-randomly varying among 2^N-1 multiples of $T_{CLK.}$ Obviously, the maximum desired number of pseudo-random pulses should not exceed 2^N-1. Fed by a clock signal with period of T_{CLK} = 5.71 ns, the PRpG circuit was implemented on a separate FPGA prototyping board, driving the data input of the TDC test bench board (see Figure 6.11).

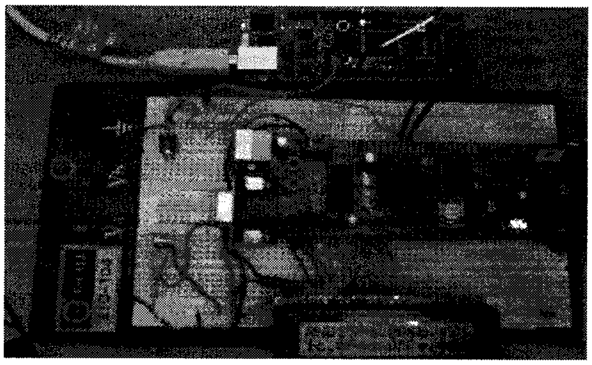


Figure 6.11: FPGA platforms implementing the TDC system and CDT Test Circuitry

B. ***Test Application Using Pulse Sweeping***: In this scheme, the PRpG was replaced by the HP-81130A pulse/pattern generator. As shown below, the sweep pulses were generated at sequential intervals that were distant by (T_{OBS} + ∂t).

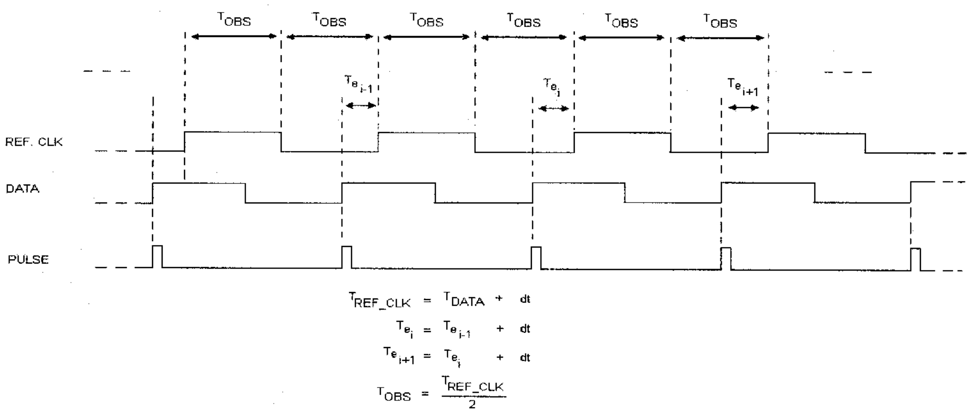

T_{OBS}
T_{OBS}
T_{OBS}
T_{OBS}
T_{OBS}
T_{OBS}
$T_{e_{i-1}}$
T_{e_i}
$T_{e_{i+1}}$
REF. CLK
DATA
PULSE
$T_{REF\_CLK} = T_{DATA} + dt$
$T_{e_i} = T_{e_{i-1}} + dt$
$T_{e_{i+1}} = T_{e_i} + dt$
$T_{OBS} = \frac{T_{REF\_CLK}}{2}$

Figure 6.12: Pulse sweep generation for $T_{REF\_CLK} = T_{DATA} + \partial t$

By feeding the data input of the test bench circuit with a periodic signal that has a slightly smaller/larger period than the reference clock (T_{REF_CLK} = T_{DATA} + ∂t), the resulting incoming pulses were swept through the cells of the matrix lines. Pulse sweeping was also used to sweep the actual TDC circuit as another alternative to investigate the value of the vertical resolution τ_Y where a systematic sweep from one row of the matrix to another was observed. The histograms obtained by using both methods of CDT application are presented and discussed next.

6.2 Acquired Data and Analysis

6.2.1 Timing Analyzer Data

Following the logic placement and routing for the chosen matrix topology in terms of CLB elements, the propagation delay of the logic elements, namely the latch and buffer components of a cell in the delay matrix, were examined using the Timing Analyzer tool embedded in the ISE Tools. The Timing Analyzer can perform point-to-point static timing analysis of a design network on the FPGA surface with no stimulus vectors. Using

Timing Analyzer, the delay along a given path can be verified for meeting the specific timing requirements of a design. While calculating the worst-case timing of a design, Timing Analyzer takes into account delays for combinational logic, routing, and other delays such as clock-to-Q and setup times of FF/Latches.

To determine the value of the horizontal resolution τ_X, the reference logic and routing delays for the buffer and latch components of the delay cells in the matrix lines were taken from the partial RO-based calibration lines that were placed on top and bottom of the delay matrices. These reference delay values were observed to be uniform throughout the reference delay lines. The value of the vertical resolution τ_Y which is the combination of the logic delay (latch) and a vertical interconnect between successive rows, was observed by tracing the vertical path of the matrices using Timing Analyzer.

Table 6.1: Delay analysis using Timing Analyzer

Component	**Delay**		
	Logic (ps)	**Interconnect (ps)**	**Net Delay**
Latch	633	955	t_L =1.588 ns
Buffer	911	564	t_B =1.475 ns
Horizontal Vernier Resolution $\tau_X = (t_L - t_B)$			**113 ps**
Latch logic Delay			633 ps
Vertical interconnect			631 ps
Vertical Time Step/Resolution τ_Y = vertical t_L			**1.264 ns**

Therefore, as shown in table 6.1, the respective simulation values for vertical resolution τ_Y and horizontal resolution τ_X were observed to be *1.264 ns and 113 ps.*

6.2.2 Calibration Data

In addition to the static timing analysis using the Timing Analyzer tool, two calibration schemes discussed in section 4.4 were also used to estimate the actual values of the vertical resolution (τ_Y) and horizontal resolution (τ_X).

RO-based Calibration: To determine τ_Y, the difference in the periods of the reference and the corresponding ring oscillator with the embedded device under test (*DUT*) was used as explained in section 4.4. Similarly, to determine τ_X the difference in obtained periods of the buffer and latch –configured horizontal ring oscillators was used. Shown in tables 6.2 and 6.3, the values of τ_Y and τ_X were estimated to be 1.0 ns and 74 ps respectively.

Table 6.2: Determination of τ_Y using Ring Oscillator-based Calibration

Ring Oscillator	**T_{CAL} (ns)**	**CNT**	**T_{RO} (ns)**	**ΔT (ns)**	**$\tau_Y = \Delta T / (2 \times 3)$** *(3-stage DUT)*
RO-REF-1	42240	920	45.913	6.043	1.007 ns
RO-DUT-1	42240	813	51.956		
RO-REF-2	42240	913	46.265	5.948	0.991 ns
RO-DUT-2	42240	809	52.212		
AVERAGE					**1.00 ns**

Table 6.3: Determination of τ_X using Ring Oscillator-based Calibration

Ring Oscillator	**T_{CAL} (ns)**	**CNT**	**T_{RO} (ns)**	**BUF/LATCH DELAY (ns)**	**$\tau_X = t_L - t_B$ (ps)**
Buffer-RO-1	52800	636	82.98	1.221	73
Latch-RO-1	52800	600	88.00	1.294	
Buffer-RO-2	52800	636	83.01	1.221	73
Latch-RO-2	52800	602	88.00	1.294	
Buffer-RO-1	42240	509	82.98	1.220	74
Latch-RO-1	42240	480	88.00	1.294	
Buffer-RO-2	42240	509	83.01	1.220	74
Latch-RO-2	42240	480	87.70	1.294	
				AVERAGE	**73.5**

Analytical Calibration: The values of $\boldsymbol{\tau_Y}$ and $\boldsymbol{\tau_X}$ were also evaluated using the analytical method of cell delay estimation discussed in section 4.4. Table 6.4 tabulates sample calibration data obtained by applying test intervals of variable lengths in which case equations 4.14 and 4.15 are used to estimate τ_Y and τ_X. On the other hand, table 6.5 presents sample calibration data obtained from a periodic test signal of constant period T_{CAL}, hence, using equations 4.16 and 4.17.

Table 6.4: Analytical calibration results using different intervals T1 and T2

Trial	**Test Intervals**	**Coordinates**				**Estimated Values**	
	(ns)	**m_1**	**m_2**	**n_1**	**n_2**	**τ_Y (ns)**	**τ_X (ps)**
1	8.5	3	6	24	15	0.971	81
	10.2	1	7	24	13		
2	8.0	3	2	15	8	1.092	355
	19.0	2	10	24	4		
3	8.0	3	2	22	14	0.981	154
	11.0	1	6	15	3		
4	13.5	2	3	21	7	1.000	500
	17.0	1	1	22	5		
5	13.5	7	2	22	9	1.000	100
	9.5	7	2	16	7		
6	15.0	2	1	15	0	0.992	121
	13.5	7	2	22	9		
7	9.5	3	6	23	13	0.989	129
	15.0	10	1	24	10		
8	8.0	5	2	7	0	1.017	293
	21.5	2	5	24	2		
9	10.0	2	6	12	1	1.018	300
	21.5	2	5	24	2		
10	24.5	2	1	24	1	1.049	383
	10.0	2	6	12	1		
11	8.0	5	2	7	0	1.029	265
	9.5	3	6	23	13		
12	15.0	10	1	24	10	1.000	111
	17.0	1	1	22	5		
13	24.5	2	1	24	1	1.054	260
	19.0	2	10	24	4		
14	23.5	3	1	24	1	1.000	250
	20.5	10	8	24	4		
15	10.2	1	7	24	13	1.158	423
	12.0	6	5	24	14		
16	21.5	6	3	21	0	1.010	95
	15.0	10	1	24	10		
17	9.8	3	6	19	9	1.018	126
	22.0	6	1	22	1		
18	7.5	12	6	24	17	1.000	83
	17.0	1	1	22	5		
19	24.5	2	1	24	1	1.064	21
	8.6	6	2	24	16		
20	17.0	6	5	19	2	0.991	158
	8.6	2	4	16	7		
			AVERAGE			***1.022***	***210***
			STANDARD DEVIATION			***0.044***	***132***
			MAXIMUM			***1.158***	***500***
			MINIMUM			***0.971***	***21***

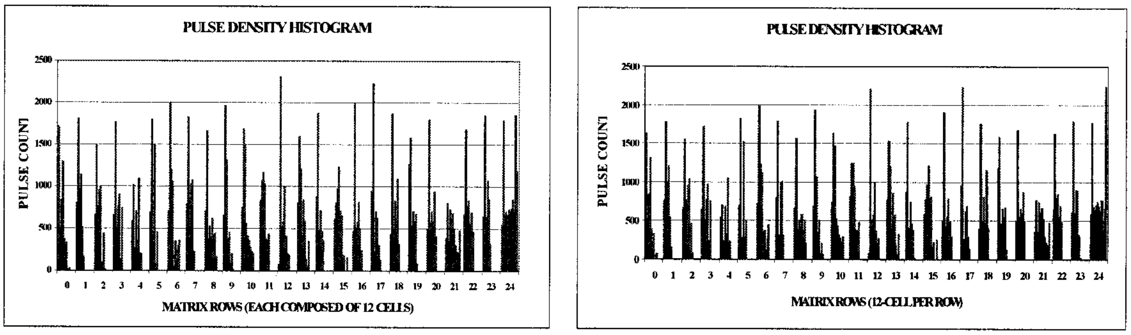
Table 6.5: Analytical calibration results using T_{CAL}

Trial	Column			Row Coordinates			T_{CAL}	Estimated Values	
	m_1	m_2	m_3	n_1	n_2	n_3	(ns)	τ_X (ns)	τ_Y (ns)
1	2	7	2	22	13	5	8.5	0.100	1.000
2	7	2	6	20	12	3	8.5	0.110	0.994
3	6	2	6	22	14	5	8.5	0.125	1.000
4	4	2	4	16	8	0	8.5	0.000	1.063
5	1	6	6	21	10	0	10.2	0.204	1.020
6	6	7	1	23	13	4	10.2	0.148	1.035
7	3	1	6	24	14	3	10.2	0.142	0.992
8	3	1	4	22	12	1	10.2	0.196	0.981
9	5	1	7	24	15	4	9.8	0.200	1.000
10	3	6	1	19	9	0	9.8	0.127	1.018
11	2	5	1	22	13	5	8.7	0.145	1.015
12	10	4	2	24	16	7	8.7	0.229	0.916
13	2	9	4	17	8	0	8.7	0.086	1.034
14	5	2	1	24	14	4	10.3	0.000	1.030
15	3	6	2	22	11	1	10.3	0.139	1.000
16	2	1	7	23	13	2	10.3	0.145	0.994
17	4	6	1	24	15	7	8.6	0.141	1.000
18	6	2	4	24	16	7	8.6	0.165	1.063
19	5	2	4	16	9	1	7.5	0.197	1.020
20	12	6	4	24	17	9	7.5	0.221	1.035
							Average	0.141	0.997
							Standard Deviation	0.063	0.040
							Maximum	0.229	1.063
							Minimum	0.000	0.882

The results of tables 6.4 and 6.5 show that the obtained values obtained for vertical bin $\tau_Y$are close enough to the RO-based value with a magnitude of ~ 1.0 ns. On the other hand, to estimate the smaller time bin τ_X, it is seen that only a select number of test runs produces desirable results. One of main reasons for the irregularities in the estimated values of τ_X comes from the deviation of the converter response from the expected theoretical linear response. Since the calibration equations in section 4.4 are based on a theoretical linear response, any non-linearity in the TDC response produces matrix coordinates that will adversely affect the estimated value of τ_X with respect to the one expected from the system of linear equations. In some cases, the value of τ_X was found to be zero, whereas in other extreme cases, not shown above, negative values of $\tau_X$were also observed. The case of $\tau_X \sim 0$ is more frequent when a periodic test interval with period T_{CAL} was used. This is because with equidistant test pulses the corresponding row coordinates (n_i) mostly fall on equidistant rows of matrices in terms of number. Under such circumstances equations 4.16 and 4.17 lead to a zero-value estimation for τ_X. However, when using equations 4.14 and 4.15 with variable-length test intervals the obtained coordinates are less prone to produce $\tau_X \sim 0$ and only certain combinations of test intervals used along with the corresponding coordinates may lead to better results. As far as the deviation in estimated values of τ_X is concerned, both cases produce estimated values that are biased by the non-linear actual response of the TDC for τ_X.

6.2.3 Code Density Test Data

As mentioned earlier, two methods of pulse application were used to collect code density data. The collected data from both methods are presented in tables 6.6 and 6.7. The table entries correspond to the accumulated number of pulses per time bin indexed by the row and column numbers. For instance, the data enlisted in the table entry with table coordinates *(n, m)* = *(0, 2)* indicates the number of pulses accumulated in the corresponding time bin of the delay matrix with coordinates of *(0, 2)*. Furthermore, it should be noted that using CDT only the first twelve cells out of fifteen cells per row are shown since no data were detected in cells 13 to 15.



a)

b)

Figure 6.13: Pulse distribution histograms across finer time bins (τ_X) a) Pulse sweeping method b) Pseudo-random pulse generation

Table 6.6: CDT results from application of pseudo-randomly generated pulses

R_i	Time Bin indices												sum_i	w_i	DNL_i	INL_i
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**				
0	1630	476	839	1315	404	188	328	0	78	0	0	0	**5258**	0.0380	-0.0491	-0.0020
1	754	1779	978	1195	561	165	0	0	0	0	0	0	**5432**	0.0393	-0.0177	-0.0027
2	672	1553	767	953	1037	94	461	69	0	0	0	0	**5606**	0.0406	0.0138	-0.0021
3	656	1712	512	775	966	231	752	0	0	0	0	0	**5604**	0.0405	0.0134	-0.0016
4	544	706	238	686	321	1050	242	228	0	0	0	0	**4015**	0.0290	-0.2739	-0.0125
5	696	1818	280	1518	284	499	0	0	0	0	0	0	**5095**	0.0369	-0.0786	-0.0157
6	714	1988	1229	1126	40	363	120	268	447	1	0	0	**6296**	0.0455	0.1386	-0.0101
7	795	1784	317	988	1007	315	0	0	0	0	0	0	**5206**	0.0377	-0.0585	-0.0125
8	659	1567	371	509	422	579	383	507	211	0	0	0	**5208**	0.0377	-0.0582	-0.0148
9	692	1937	1072	350	508	72	212	59	0	0	0	0	**4902**	0.0355	-0.1135	-0.0193
10	731	1641	1464	522	443	320	270	217	291	0	0	0	**5899**	0.0427	0.0668	-0.0167
11	813	1240	1237	949	371	378	480	0	0	0	0	0	**5468**	0.0396	-0.0111	-0.0171
12	85	2206	514	579	993	366	197	268	6	0	0	0	**5214**	0.0377	-0.0571	-0.0194
13	774	1538	1195	400	866	576	193	58	329	0	0	0	**5929**	0.0429	0.0722	-0.0165
14	871	1777	412	742	462	381	22	0	0	0	0	0	**4667**	0.0338	-0.1560	-0.0228
15	591	776	971	1209	739	811	161	221	3	248	0	0	**5730**	0.0414	0.0362	-0.0213
16	503	1904	428	542	785	496	303	0	0	0	0	0	**4961**	0.0359	-0.1028	-0.0254
17	958	2230	265	624	692	290	132	0	0	0	0	0	**5191**	0.0376	-0.0612	-0.0279
18	396	1754	493	804	475	1152	410	0	0	0	0	0	**5484**	0.0397	-0.0083	-0.0282
19	1181	1590	458	663	549	674	126	0	0	0	0	0	**5241**	0.0379	-0.0522	-0.0303
20	499	1671	559	662	598	874	496	0	0	0	0	0	**5359**	0.0388	-0.0309	-0.0315
21	358	763	356	745	613	671	522	295	216	190	468	0	**5197**	0.0376	-0.0602	-0.0339
22	650	1630	797	849	549	697	496	0	0	0	0	0	**5668**	0.0410	0.0250	-0.0329
23	610	1781	597	903	894	320	22	0	0	0	0	0	**5127**	0.0371	-0.0728	-0.0358
24	588	1762	687	641	706	750	678	627	768	371	667	2239	**10484**	0.0758	0.8960	0.0000
Total Collected Pulses: **138241**																
												AVERAGE	5323.21	0.0385	-0.0373	-0.0189
Ideal number of pulses per row: **5529.6**												**STANDARD DEVIATION**	455.52	0.0033	0.0824	0.0106
												MINIMUM	4015.00	0.0290	-0.2739	-0.0358
Ideal weight for each row: **0.04**												**MAXIMUM**	6296.00	0.0455	0.1386	-0.0016

Table 6.7: CDT results obtained from pulse sweeping

R_i	Time Bin indices												sum_i	w_i	DNL_i	INL_i
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**				
0	1707	493	844	1290	373	237	334	6	28	0	0	0	**5312**	0.0381	-0.0486	-0.0019
1	798	1804	977	1139	519	165	0	0	0	0	0	0	**5402**	0.0387	-0.0325	-0.0032
2	667	1492	763	957	1003	94	446	24	0	0	0	0	**5446**	0.0390	-0.0246	-0.0042
3	670	1758	512	776	909	134	743	0	0	0	0	0	**5502**	0.0394	-0.0146	-0.0048
4	593	1010	274	711	346	1092	205	203	0	0	0	0	**4434**	0.0318	-0.2059	-0.0130
5	696	1793	296	1496	91	450	0	0	0	0	0	0	**4822**	0.0345	-0.1364	-0.0185
6	701	1998	1201	1049	53	346	133	301	355	0	0	0	**6137**	0.0440	0.0992	-0.0145
7	791	1824	355	1025	1067	230	0	0	0	0	0	0	**5292**	0.0379	-0.0522	-0.0166
8	702	1660	366	531	396	619	411	443	163	0	0	0	**5291**	0.0379	-0.0524	-0.0187
9	655	1960	1316	377	458	86	207	13	0	0	0	0	**5072**	0.0363	-0.0916	-0.0224
10	754	1684	1488	556	412	357	279	218	215	0	0	0	**5963**	0.0427	0.0680	-0.0197
11	829	1071	1164	1035	376	354	432	0	0	0	0	0	**5261**	0.0377	-0.0577	-0.0220
12	75	2308	527	575	995	408	199	184	0	0	0	0	**5271**	0.0378	-0.0559	-0.0242
13	797	1589	1204	404	838	592	134	61	344	0	0	0	**5963**	0.0427	0.0680	-0.0215
14	883	1873	487	711	459	353	0	0	0	0	0	0	**4766**	0.0341	-0.1464	-0.0273
15	614	802	978	1222	717	660	179	178	2	151	0	0	**5503**	0.0394	-0.0144	-0.0279
16	461	1993	511	580	813	506	238	0	0	0	0	0	**5102**	0.0366	-0.0862	-0.0314
17	944	2234	230	706	630	310	121	0	0	0	0	0	**5175**	0.0371	-0.0731	-0.0343
18	435	1863	505	827	455	1095	314	0	0	0	0	0	**5494**	0.0394	-0.0160	-0.0349
19	1266	1582	543	708	586	673	84	0	0	0	0	0	**5442**	0.0390	-0.0253	-0.0359
20	496	1796	569	707	576	933	407	0	0	0	0	0	**5484**	0.0393	-0.0178	-0.0367
21	390	799	344	735	595	682	498	311	225	209	481	0	**5269**	0.0377	-0.0563	-0.0389
22	677	1682	770	832	562	697	467	0	0	0	0	0	**5687**	0.0407	0.0186	-0.0382
23	644	1841	635	1075	845	333	2	0	0	0	0	0	**5375**	0.0385	-0.0373	-0.0397
24	552	1787	673	703	664	734	669	739	852	708	1849	1189	**11119**	0.0797	0.9915	0.0000
Total Collected Pulses: **139584**																
												AVERAGE	5352.71	0.0383	-0.0413	-0.0229
Ideal number of pulses per row: **5583.36**												**STANDARD DEVIATION**	375.78	0.0027	0.0673	0.0119
												MINIMUM	4434.00	0.0318	-0.2059	-0.0397
Ideal weight for each row: **0.04**												**MAXIMUM**	6137.00	0.0440	0.0992	-0.0019

Table 6.8: Estimation of τ_Y using CDT results for T_{OBS} = 27.5ns

Row	Pseudo-random Method		Pulse sweeping Method	
	W_i	$\tau_Y = W_i X$	W_i	$\tau_Y = W_i X$
0	0.0381	1.046 ns	0.0380	1.047 ns
1	0.0387	1.081 ns	0.0393	1.064 ns
2	0.0390	1.115 ns	0.0406	1.073 ns
3	0.0394	1.115 ns	0.0405	1.084 ns
4	0.0318	0.799 ns	0.0290	0.874 ns
5	0.0345	1.014 ns	0.0369	0.950 ns
6	0.0440	1.252 ns	0.0455	1.209 ns
7	0.0379	1.036 ns	0.0377	1.043 ns
8	0.0379	1.036 ns	0.0377	1.042 ns
9	0.0363	0.975 ns	0.0355	0.999 ns
10	0.0427	1.173 ns	0.0427	1.175 ns
11	0.0377	1.088 ns	0.0396	1.036 ns
12	0.0378	1.037 ns	0.0377	1.038 ns
13	0.0427	1.179 ns	0.0429	1.175 ns
14	0.0341	0.928 ns	0.0338	0.939 ns
15	0.0394	1.140 ns	0.0414	1.084 ns
16	0.0366	0.987 ns	0.0359	1.005 ns
17	0.0371	1.033 ns	0.0376	1.020 ns
18	0.0394	1.091 ns	0.0397	1.082 ns
19	0.0390	1.043 ns	0.0379	1.072 ns
20	0.0393	1.066 ns	0.0388	1.080 ns
21	0.0377	1.034 ns	0.0376	1.038 ns
22	0.0407	1.128 ns	0.0410	1.120 ns
23	0.0385	1.020 ns	0.0371	1.059 ns
24	0.0797	2.086 ns	0.0758	2.191 ns
AVERAGE		**1.0589** ns		**1.0546** ns
STANDARD DEVIATION		**0.0906** ns		**0.0740** ns
MINIMUM		**0.7987** ns		**0.8736** ns
MAXIMUM		**1.2525** ns		**1.2091** ns

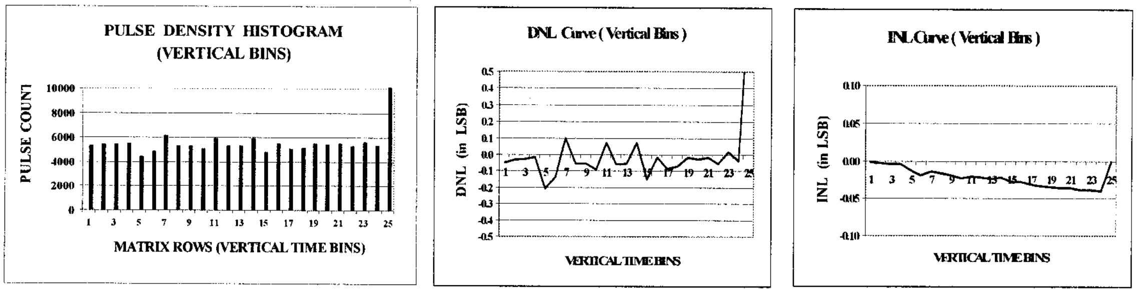


Figure 6.14: Row-wise pulse density, DNL, & INL curves from pulse sweeping

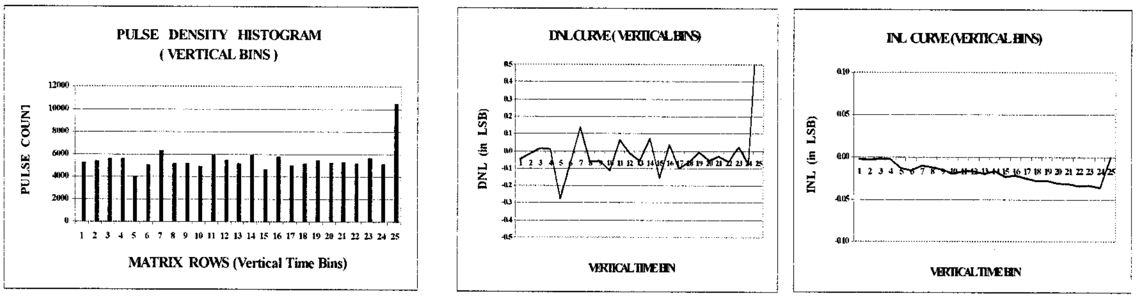


Figure 6.15: Row-wise pulse density, DNL & INL curves from PRpG

Table 6.8 presents the estimated vertical resolution from CDT data. Again, it is seen that the estimated value of τ_Y is in agreement with the estimated τ_Y values using the RO-based and analytical calibration methods. To estimate the horizontal time steps τ_X (single shot measurement resolution), CDT data also explored irregularities in the magnitude of τ_X. The CDT data show that while some bins accumulate number of pulses that are somehow close to the expected uniformly distributed pulse count, other bins show unexpected non-uniform pulse accumulation. This can be seen in the CDT data of tables 6.6 and 6.7 and the corresponding histograms in Figure 6.13.

A first source of these irregularities is attributed to the existence of clock skew along the clock paths of the matrix lines that causes enlargement/shrinkage in the vertical bins of the matrices. In addition to mismatch in the number of pulses to be accumulated in each row of the matrix, the enlargement/shrinkage in vertical bins affects data accumulation in smaller bins contained within the latter by undesirably dislocating incoming pulses from their expected cell locations. For instance, depending on the type (positive/negative) and magnitude of clock skew along two sequential rows R_i and R_j, some pulses near farther cells of R_i may end up on the early cells of the subsequent row R_j. Similarly, pulses distant that are to fall on early cells of R_i may undesirably fall on farther cells of the earlier row R_i. The estimated values of clock skew are tabulated in tables 6.11 and 6.12.

A second major source of irregularities in the small time bins of the matrices is attributed to delay mismatch among cells that can be mainly attributed to mismatch in the corresponding routing delays. Observations of logic cell placement and routings using the FPGA EDITOR and the TIMING ANALYZER tools from XILINX revealed that while the interconnect routing among buffer components along cells of a row are uniform with negligible irregularities, there exists some non-uniformity in inter-latch routings. That is, despite the symmetric logic cell placement of latch elements uniform logic routings is guaranteed, despite application of timing constraints such as *MAXDELAY* which serves to guide the tool to choose available routings that would meet the given *MAXDELAY* values as net delays. Depending on the chosen programmable interconnection points by the routing tool, the routing delays may differ to some considerable extent. For instance, static timing analysis from Timing Analyzer shows that some latch-interconnecting nets such as the routings that connect the second latch elements of the rows to the third have a static delay of ~1.35ns compared to ~1.01 ns for the majority of other cells in a row. The CDT results also show that the second cells of each row accumulates a great number of pulses compared to its adjacent cells. This is mainly because the corresponding time bin is larger than the expected ~80ps as obtained by the embedded calibration lines. It is to note, however, that the static timing analysis for the calibration lines show exact uniform delay values as outlined in Table 6.1 for routing and logic delays of the buffer and latch components throughout both calibration lines. A sample static timing analysis result obtained from Timing Analyzer for row-1 of matrix-1 driven by CLK0 is presented in table 6.9.

In view of the foregoing, the CDT results may not serve to determine the single shot high measurement resolution and, hence, the latter is estimated using the values obtained from calibration lines and simulations outlined in earlier section. However, the CDT data does serve to estimate/evaluate the values of the vertical time bin (τ_Y) and the interline clock skew. Additionally, the CDT data proved helpful as an alternate tool to the timing analyzer tools in identifying the non-linearity limitations imposed on the small time bins of the delay matrices.

Table 6.9: Sample Timing Analyzer data for ROW-1 of delay matrix M-1

CELL	BUFFER		LATCH	
	DELAY (ns)		DELAY (ns)	
	LOGIC	Driven-NET	LOGIC	Driven-NET
1	0.911	0.564	0.633	1.006
2	0.911	0.564	0.650	1.354
3	0.911	0.564	0.650	1.322
4	0.911	0.564	0.650	1.014
5	0.911	0.575	0.650	1.039
6	0.911	0.564	0.650	1.030
7	0.911	0.564	0.650	1.030
8	0.911	0.564	0.650	1.014
9	0.911	0.564	0.650	1.030
10	0.911	0.572	0.650	1.322
11	0.911	0.564	0.650	1.014
12	0.911	0.572	0.650	1.322
13	0.911	0.909	0.650	0.687
14	0.911	0.909	0.650	0.687
15	0.911	-	0.650	0.593

6.2.4 Test Interval Measured Data

As mentioned earlier, many test intervals with lengths randomly selected from 8ns to the full dynamic range were measured. A select number of measurements are tabulated below:

Table 6.10: Sample Interval Measurement Data

$\tau_Y = 1.0$ ns $\tau_X = 100$ ps			
Test Interval (ns)	Measured Interval	% error	Absolute
8.000	7.630	4.63%	370
9.000	9.074	0.82%	74
13.500	13.926	3.16%	430
15.000	14.666	2.23%	333
17.000	17.780	4.58%	780
20.500	20.148	1.72%	350
33.000	32.230	2.33%	770
41.000	40.860	0.34%	140
188.000	188. 320	0.17%	320
650.000	649.270	0.11%	730

The estimated value for τ_X varies between ~75ps in RO-calibration and 113ps in simulation, and the obtained results corresponding to the applied test intervals produced similar percentage errors using any values in the range (75-150) ps. The values of τ_X and τ_Y used to calculate the intervals in table 6.10 were 100ps and 1.0ns respectively.

6.2.5 Clock Skew Estimation

The values of skew along the common clock lines of the delay matrices obtained using the FPGA Editor Tool of XILINX are presented in below:

Table 6.11: Clock net delays from source to destination cells of the matrices

DESTINATION ROW	Matrix-1 (CLK0)		Matrix-2 (CLK180)	
	CLK NET DELAY (SRCE -TO-DEST)	Relative Skew (DELAY – AVG_DELAY)	CLK NET DELAY (SRCE -TO-DEST)	Relative Skew (DELAY – AVG_DELAY)
0	1.974	0.012	2.168	0.198
1	1.982	0.020	1.982	0.012
2	1.835	-0.127	1.835	-0.135
3	1.860	-0.102	1.860	-0.110
4	1.841	-0.121	1.841	-0.129
5	1.835	-0.127	1.835	-0.135
6	1.854	-0.108	1.854	-0.116
7	1.839	-0.123	1.839	-0.131
8	1.848	-0.114	1.848	-0.122
9	1.851	-0.111	1.851	-0.119
10	1.992	0.030	1.992	0.022
11	1.818	-0.144	1.818	-0.152
12	1.824	-0.138	1.824	-0.146
13	1.965	0.003	1.965	-0.005
14	1.815	-0.147	1.815	-0.155
15	1.917	-0.045	1.917	-0.053
16	1.972	0.010	1.972	0.002
17	1.929	-0.033	1.929	-0.041
18	1.939	-0.023	1.939	-0.031
19	2.088	0.126	2.088	0.118
20	2.218	0.256	2.218	0.248
21	2.210	0.248	2.210	0.240
22	2.218	0.256	2.218	0.248
23	2.210	0.248	2.210	0.240
24	2.218	0.256	2.218	0.248
AVG_DELAY	**1.962** ns		**1.970** ns	
MAX	**2.218** ns		**2.218** ns	
MIN	**1.815** ns		**1.815** ns	
MAX_SKEW = MAX – MIN	*403 ps*		*403 ps*	

Results from code density testing can also be used to estimate skew on the clock lines of the delay matrices. However, since CDT results are obtained from combination of both matrices operating in parallel, the skew values will reflect the combined skew effects sourced from clock lines in both matrices. Described in section 5.2, the existence of skew causes an enlargement/shrinkage in the vertical time bins of the interpolator (delay matrices). By estimating the values of τ_Y obtained from CDT, a good approximation of skew is obtained. Of course, this method assumes that other limiting factors such as variations in the reference clock period are random and their effects are cancelled out during averaging. Table 6.12 presents the estimated maximum and intermediate skew values using CDT data for both methods of pulse application.

Table 6.12: Estimation of Skew from CDT data

ROW	PULSE SWEEPING CDT			PSEUDO-RANDOM CDT		
	τ_Y	$\Delta t = (\tau_Y - AVG)$ ns	SKEW	τ_Y	$\Delta t = (\tau_Y - AVG)$ ns	SKEW
0	1.047 ns	-0.008	-	1.046 ns	-0.013	-
1	1.064 ns	0.010	**0.008**	1.081 ns	0.022	**0.013**
2	1.073 ns	0.018	**-0.010**	1.115 ns	0.056	**-0.022**
3	1.084 ns	0.029	**-0.018**	1.115 ns	0.056	**-0.056**
4	0.874 ns	-0.181	**-0.029**	0.799 ns	-0.260	**-0.056**
5	0.950 ns	-0.105	**0.181**	1.014 ns	-0.045	**0.260**
6	1.209 ns	0.154	**0.105**	1.252 ns	0.194	**0.045**
7	1.043 ns	-0.012	**-0.154**	1.036 ns	-0.023	**-0.194**
8	1.042 ns	-0.012	**0.012**	1.036 ns	-0.023	**0.023**
9	0.999 ns	-0.055	**0.012**	0.975 ns	-0.084	**0.023**
10	1.175 ns	0.120	**0.055**	1.173 ns	0.115	**0.084**
11	1.036 ns	-0.018	**-0.120**	1.088 ns	0.029	**-0.115**
12	1.038 ns	-0.016	**0.018**	1.037 ns	-0.022	**-0.029**
13	1.175 ns	0.120	**0.016**	1.179 ns	0.121	**0.022**
14	0.939 ns	-0.116	**-0.120**	0.928 ns	-0.131	**-0.121**
15	1.084 ns	0.030	**0.116**	1.140 ns	0.081	**0.131**
16	1.005 ns	-0.049	**-0.030**	0.987 ns	-0.072	**-0.081**
17	1.020 ns	-0.035	**0.049**	1.033 ns	-0.026	**0.072**
18	1.082 ns	0.028	**0.035**	1.091 ns	0.032	**0.026**
19	1.072 ns	0.018	**-0.028**	1.043 ns	-0.016	**-0.032**
20	1.080 ns	0.026	**-0.018**	1.066 ns	0.007	**0.016**
21	1.038 ns	-0.017	**-0.026**	1.034 ns	-0.025	**-0.007**
22	1.120 ns	0.066	**0.017**	1.128 ns	0.069	**0.025**
23	1.059 ns	0.004	**-0.066**	1.020 ns	-0.039	**-0.069**
24	2.191 ns	1.136	**-0.004**	2.086 ns	1.027	**0.039**
AVG	**1.0546** ns	-		**1.0589** ns	-	
Max Skew = Most Positive + | Most Negative |			**301 ps**			**454 ps**

The values of skew corresponding to the clock lines of the rows of delay matrices are estimated by first finding the shrinkage/enlargement of the vertical bins. In Table 6.12, the shrinkage/enlargement of the time bins is represented by negative and positive values of Δt respectively. From the skew analysis of section 5.2, it is seen that a shrinkage in value of bin-i by Δt represents a positive skew of magnitude Δt on clock line of the subsequent row (R_{i+1}) whereas an enlargement of bin-i by Δt reflects a negative skew of magnitude Δt on the clock line of the subsequent row (R_{i+1}). The worst case skew value, i.e. the gap between the most positive and most negative values is found to be **301ps** using pulse sweeping CDT data, and **454ps** using pseudo-random pulses CDT data. Further, it should be noted that the skew on the first row is determined by CDT in last row (R_{24}), but due to clock edge misalignment, the CDT value for row-24 is further affected, and can not actually be used for skew estimation.

6.2.6 Accuracy/Precision Analysis

Shown in Figure 6.16, measurement accuracy is taken in this report as the average measured value deviation from the ideal result. Measurement precision, on the other hand, refers to the dispersion of the measured values from the mean value of measurements.

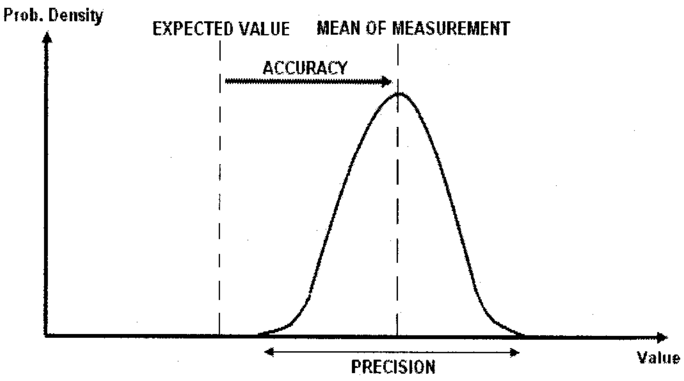


Figure 6.16: Measurement Accuracy and Precision definitions

The TDC circuit was tested by measuring many test intervals of fixed and variable-lengths to be measured. From the test interval measurements, it was observed that the maximum relative error of ~ 11% was observed for intervals less than 12.0 ns. To analyze measurement accuracy and precision parameters two analysis methods were carried out in terms of measured values and relative errors. For the first case, 30 measurements corresponding to an 8.0 ns interval were made. The 8.0 ns interval was chosen because the relative measurement error is highest with the smaller intervals. For the test intervals of 8.0 ns, the mean value of measurements was found to be 8.033 ns with a standard deviation (σ) value of 0.366ns. The accuracy of measurement in this case is 33 ps. The measurement precision, however, is widely spread between $\pm 3\sigma$ values from the measurement mean. Hence, the measured values vary in a range of $(8.033 \pm 3\sigma)$ ns with $\sigma = 0.366$ns.

Through the application of over 300 test intervals of variable lengths, the accuracy and precision of the measurements were also analyzed in terms of relative percentage error. That is, the accuracy of measurements given as the average percentage error from the expected value was observed to be ~3.5% for intervals of lengths less than or equal to 12.0 ns and ~ 1.5% for intervals larger than 12.0 ns respectively. The measurement precisions given in terms of the standard deviation of the relative percentage errors from the mean error values were observed to be ~3.0% and 1.0% for the respective accuracy values of 3.5% and 1.0%.

Therefore, using the TDC circuit the average deviation of a measured time interval *T* from its ideal value may be ~1.5% for intervals larger than 12.0 ns and 3.5% for intervals with ideal lengths less than or equal to 12.0ns.

6.2.7 FPGA Area Utilization

The results of the XILINX synthesis report are presented below, showing the number of resources occupied by the current implementation of the TDC system.

Table 6.13: SPATRAN^{TM-}3 FPGA Resource Utilization Summary

Delay Matrices		Complete TDC including Matrices	
Number of Slices	1572 out of 7680 (20%)	Number of Slices:	2320 out of 7680 (30%)
Number of Slice Flip Flops	927 out of 15360 (6%)	Number of Slice Flip Flops:	2194 out of 15360 (14%)
Number of 4 input LUTs	3143 out of 15360 (20%)	Number of 4 input LUTs:	4640 out of 15360 (30%)
		Number of BRAMs:	7 out of 24 (29%)
		Number of GCLKs:	8 out of 8 (100%)
		Number of DCMs:	2 out of 4 (50%)

6.3 Discussions

The main sources of measurement error are attributed to clock skew on the clock lines of the delay matrices, variations in reference clock parameters in the form of jitter, phase and duty cycle, non-uniformity of cell delays due to existence of some non-uniform routings, and the on-chip delay mismatch among identical logic elements on FPGA surface.

The TDC circuit is vulnerable to clock skew due to its sharing of a common clock line among various rows of the core interpolator (delay matrices). While FPGA vendors provide dedicated global clock routings and other interconnects such as long wires in an attempt to have minimized skew along clock paths, the TDC circuit can not exploit the latter. This is due to the fact that, by design, the clock signals in this design enter logic elements other than the clock elements (FF) within the FPGA Configurable Logic Blocks. However, despite these systematic limitations, the clock skew was minimized by means of buffer insertion along the clock path from clock source (DCM Unit) to the expected destination points on the delay matrix and through the application of directed routing constraints to lock the best case routing achieved by several test and try cases. The maximum skew values along clock lines of both matrices were estimated to be **403ps** using static timing analysis. On the other hand, using CDT data the maximum skew value was estimated to be **301ps** and **454ps** by using the pulse sweeping and pseudo-random pulses methods, respectively.

The variations in parameters of the clock signal generated from the digital clock manager (DCM) unit affect the system performance as well. XILINX reports jitter errors of up ~150 ps on the DCM outputs CLK0 and CLK180. These are the main clock signals used as the out-of-phase clocks of the delay matrices in parallel. These variations in the reference clock edges affect the observation time window (T_{OBS}) and ultimately the pulse locations on the delay matrix of concern. The displacements in expected pulse locations introduce errors when the distance between two displaced pulses is measured. Similarly, XILINX reports uncertainty of up to ~400 ps in duty cycle errors for the DCM outputs. Uncertainty in duty cycle also affects the observation time window in the form of shrinkage and enlargement of the latter, also leading to pulse location displacements. Existence of phase error between CLK0 and CLK180 also causes measurement uncertainty in the incoming pulses in the vicinity of the misaligned clock edges.

Non-uniformity in routing delays causes differential non-linearity among time interpolator bins. From CDT data it was seen that the linearity of the interpolator was good for the vertical time bins. However, the non-linearity of the smaller bins within the vertical time steps was observed to be large. Non-uniform routing delays in the latch components of the delay cells and the clock skew were identified to be the main source of the non-linearity errors. While manual logic placement was heavily used to guarantee symmetric placement of logic elements on the FPGA surface, manual routing is impractical. The routing of the logic cells were mostly controlled by designating specific pins in the source and destination cells. Additionally, several XILINX-specific design and implementation constraints such as *MAXDELAY*, *MAXSKEW*, and *BEL* were used in an attempt to guide the XILINX tools to achieve uniform logic placement and routing. Static timing analysis showed that while the tools respected uniformity of routing and logic delays in clock buffer lines, attempts to do the same for latch components failed to some extent. This is mainly due to the availability of programmable interconnecting points (*PIP*) in the switch matrix associated to each CLB block.

Lastly, the other major factor in limiting performance of the TDC is the on-chip delay mismatch among theoretically identical logic elements. Delay mismatch also affects the theoretically uniform delay values among delay cells of the matrices. In [40], delay mismatch values were estimated to be of the order of picoseconds.

6.4 Chapter Summary

In summary, this chapter discussed the various test strategies used to evaluate various performance characteristics of the proposed TDC circuit. Functional simulation was used to verify the functionality of supporting system modules such as the controller, the FIFO, the encoder, and the controller unit. Timing simulation was carried out to verify the behaviour of pulse propagation throughout the delay matrices as well as other critical timings vital in the correct functionality of each module within the system. In addition to simulations, on board experimental testing was also carried out using the FPGA prototyping board to verify system functionality. To investigate the vertical (τ_Y) and the horizontal (τ_X) measurement resolutions, static timing analysis, experimental calibration lines based on ring oscillators, as well as an analytical method of calibration were used. Furthermore, many test intervals were measured using the found values of resolutions. Code density testing based on two different methods of pulse application, namely the pseudo random pulse generation and pulse sweeping was used to investigate the static behaviour of the TDC circuit. In addition to static characteristics, the skew along the clock lines of the matrix was also estimated using static timing analysis results and the code density testing.

Table 6.14 summarizes all the characteristic values of the TDC circuit with the delay matrices sizes of 25 x 15, $T_{OBS} = 27.5$ ns.

Table 6.14: Summary of TDC characteristic parameters

Parameter	**Value**					
	Simulation	**Experimental Calibration**		**Code Density Test**		**Direct Interval Measurement**
		RO-based	**Analytic**	**PRpG**	**P. Sweep**	
Vertical time step (τ_Y)	1.264 ns	~1.0 ns	~1.0 ns	1.06 ns	1.05 ns	N/A
			DNL	Within ± 0.3LSB		
			INL	Within ± 0.1LSB		
Meas. Resolution (τ_X)	113 ps	74 ps	(134 – 229) ps *	N/A**	N/A**	N/A
			DNL and INL	N/A**		
PPR	8.0 ns	N/A				7.5 ns
Dynamic Range	2^{11} x 27.5 ns (using an 11-bit Coarse counter)					
Avg. Meas. Error (%	N/A					2.6 %
Avg. Meas. Error (abs)	N/A					320 ps
Extra Meas. Time	*M x t_B* => *Simulation*: 15 x 1.48 ns *Experimental*: ~ 15 x 1.2 ns					

Estimated Measurement Accuracy and Precision

		$T_m \leq 12.0$ ns	**$T_m \geq 12.0$ ns**
Accuracy	**AVG**	~3.5 %	~1.5 %
	MAX	~11.25 %	~5.7 %
Precision		σ ~3.0 %	σ ~1.0 %

* The average values of τ_X using analytical calibration is affected by the actual non-linear response of TDC

** Due to undesirable effects, leading to high DNL and INL values, CDT data not suitable to estimate τ_X

6.4.1 Performance evaluation with comparable designs

As pointed out in section 4.2.2, the interpolator of the proposed TDC is based on the work reported in [29] where an FPGA implementation had achieved a measurement resolution of 200ps. In addition to improving the measurement resolution from 200ps to 75ps, the proposed TDC further improves the time overhead in terms of the measurement and the dead times. Moreover, the proposed TDC is capable of measuring more than one time interval per phase of measurement. However, from area point of view, the proposed TDC does occupy more surface area within the FPGA device due to its design structure (two matrices of Vernier delay cells). On the same token, the FPGA-implemented TDC design in [41] reports an RMS measurement resolution of ~50ps (actual resolution ~ 91ps). While exploiting the fast carry lines within the FPGA fabric for time interpolation, similarly to the proposed TDC, the design in [41] still suffers from relatively higher non-linearity errors that are attributed to the fixed structure of the FPGA devices used. Additionally, the pulse detection in the TDC design proposed in [41] is limited to single pulse detection per reference clock period which is not the case for the proposed TDC. Thus, in comparison with the comparables designs such as [29][41][42], the proposed TDC can be seen as a design with an state-of-the-art time interpolation technique, outlining the various limiting issues relating to FPGA structure.

CHAPTER 7 - CONCLUSION AND FUTURE WORK

This research work involved design and implementation of a multi-hit time-to-digital converter (TDC) circuit on a low cost FPGA platform. The core part of the project was the time interpolator used to slice the reference clock period into smaller time steps. Based on the Vernier principle, the time interpolator combined multiple (N) of M-stage Vernier Delay Lines (VDL) in an array topology (also seen as an $N \times M$ matrix of Vernier delay cells) as to provide two-way propagation path for the incoming pulses. The two-way data pulse propagation path in turn provides two types of time steps with different path magnitudes. The larger time steps are obtained by the vertical pulse propagation whereas the smaller time bins are obtained by the horizontal propagation path.

With the larger time paths provided, pulse propagation through many costly Vernier stages is avoided for intervals of relatively larger magnitude. This significantly reduces the number of VDL stages required leading to considerable reduction of the measurement time. Using the new interpolator with M-stage Vernier delay lines, the maximum extra measurement/interpolation time is given by $M\ t_B$ with t_B being the propagation delay of the buffer element in the Vernier delay cell. This is considerably smaller than the extra time wasted when a single VDL with many stages is used. For the implemented TDC circuit with 25 x 15 delay cell matrices, this maximum extra interpolation time is ~ (15 x 1.2ns = 18ns). In addition to the improved measurement time, parallelism was used to reduce the dead time of the circuit. Having two interpolators operating in parallel in both phases of the reference clock period provide continuous detection and processing of incoming data pulses.

Similar to any Vernier-based time interpolators, the theoretical single shot measurement resolution of the TDC circuit is given by the difference in propagation delays of the elements of the Vernier cells. In this case, the measurement resolution was given by $\tau_X = t_L - t_B$ with t_B and t_L as the respective propagation delays of the buffer and latch elements. The larger time steps (vertical resolution) of the TDC circuit is theoretically given by $\tau_Y = t_L$. However, due to the difference in routing delays in horizontal and vertical directions within the FPGA surface, the value of τ_Y will not be equal to t_L as used in determining τ_X. In simulation, the respective values of τ_X and τ_Y were found to be 113ps and 1.264ns, whereas the experimental estimations were measured to be 75ps and 1.05ns respectively.

The proposed TDC system was implemented on XSA-3S1000 FPGA platform from XESS which is based on XILINX SPARTAN^{TM}-3 FPGA chip. While the implemented prototype validated the theoretical aspect of the design by correct functionality, several performance limitations, mainly attributed to the fixed FPGA structure, were observed throughout the realization of the project. These limitations were imposed mainly by existence of clock skew on clock lines of the delay matrices, variations in reference clock parameters in forms of jitter, phase and duty cycle, non-uniformity of cell delays due to existence of non-uniform routings, and the on-chip delay mismatch among identical logic elements on FPGA surface. As discussed in chapters 4 and 5, several optimization strategies were used to minimize the negative effects of these circuits. However, as consequence of certain limiting factors such as non-uniformity in routing delays, the measurement results carried out by the TDC circuit were, to some extent, affected.

With a functional prototype of the proposed TDC circuit on the Spartan-3 FPGA platform, this research work meets its initial objectives by accomplishing time interval measurements with a resolution of ~100ps with the intended multi-hit mode of operation. In contrast to VDL-based TDC designs reported in [27][28][29], the TDC circuit ideally eliminates the conventional dead time through parallelism. Similarly, by incorporating two-level time interpolation, the measurement time is significantly improved in comparison to single VDL-based designs. Additionally, from the discussions in chapter 5 it was seen that this project work also touched upon important performance limiting issues regarding the feasibility of timely critical systems on programmable logic devices with fixed internal structure. With the achieved performance characteristics and a novel time interpolator, the proposed TDC well fits among the comparable designs, as discussed in section 6.4.1.

In order to correct the non-uniformity in cell delays caused by inter-cell routing, several attempts were made for manually routing of the design. One of the ways to achieve this was to have a manually-routed module for delay matrices and make it as a design macro. Having a macro module avoids repeated synthesis, placement, and routing for the delay matrices. Subsequently, including this macro as a routed module would keep the placement and routing intact. Despite several trials and attempts no success was made. One main factor was the large number of I/Os involved in the delay matrices, in fact larger than the available I/O pads on the target FPGA technology. When synthesized with no I/Os, the corresponding macro would have no pins and hence the tools were not able to instantiate a pin-free module. After several failed trials, the uniformity of net delays for the inter-cell connections remains to be a major factor and is left as future work for the current design.

The future work would also consist of overcoming errors related to clock skew. Existence of skew on the common clock line of the delay lines cause non-linearity in TDC response. A re-implementation of the TDC circuit on a more advanced FPGA with improved clock lines would possibly be a simple solution to investigate. A second practical choice would consist of implementing the TDC on an ASIC with improved placement and routings through careful layout design. This way, not only the clock lines would be carefully designed with minimum skew, the inter-cell routing delays may also be uniform due to custom layout design.

Improvement of other factors such as clock jitter and the undesirable variations in phase and duty cycle of the system clock is heavily suggested. One way to improve these factors might involve designing of an external clock generator module that would overcome the limitations of the DCM unit within the FPGA. Having jitter-free clock signal eliminates unwanted cycle-to-cycle enlargement/shrinkage of the detection window (T_{OBS}), which in turn improves the delectability of the interpolator. Similar improvements are also obtained by clock signals of reduced phase and duty cycle errors.

Finally, similar to any delay-line based interpolator, the delay lines remain vulnerable to delay mismatch caused by variations in process parameters. Any improved modified structure of the interpolator that would overcome the sensitivity of the matrix lines to delay mismatch would definitely improve the linearity of the TDC.

REFERENCES

[1] Antonio H. Chan, Gordon W. Roberts; "A Jitter Characterization System Using a Component-Invariant Vernier Delay Line"; IEEE Transactions on (VLSI) Systems, Vol. 12, NO. 1, Jan. 2004.

[2] Ealgoo Kim, Hansang Lim, Taeyon Lee, Dongbum Choi and Jaehong Park; "Time of Flight (TOF) measurement of adjacent pulses"; IEEE Nuclear Science Symposium Conference Record, Vol. 2, 4-10 Nov.2001 Page(s):812-815

[3] Maatta, K.; Kostamovaara, J.; "A high-precision time-to-digital converter for pulsed time-of-flight laser radar applications; Instrumentation and Measurement"; IEEE Transactions on Volume 47, Issue 2, April 1998 Page(s):521 – 536

[4] N. Paschalidis et al., "A Time-Of-Flight System On A Chip Suitable For Space Instrumentation"; IEEE Nuclear Science Symposium Conference Record, Vol. 2, 4-10 Nov.2001 Page(s):750-754

[5] Pasi P., Kari M., and Juha Kostamovaara; "Integrated Time-Of-Flight Laser Radar"; IEEE Transactions on Instrumentation and Measurement, VOL. 46, NO. 4, August 1997; pages: 996-999

[6] H. Brockhaus and A. Glasmachers; " Single Particle Detector System for High Resolution Time Measurements"; IEEE transactions on Nuclear Science, Vol. 39, issue 4, Aug. 1992, pages: 707-711.

[7] Keunoh Park; Jaehong Park; "20 ps resolution time-to-digital converter for digital storage oscilloscopes"; IEEE Nuclear Science Symposium Conference Record, Vol. 2, 8-14 Nov. 1998; Pages: 876-881.

[8] Brian K. Swann, Benjamin J. Blalock, Lloyd G. Clonts, David M. Binkley, James M. Rochelle, Eric Breeding, and K. Michelle Baldwin; "A 100-ps Time-Resolution CMOS Time-to-Digital Converter for Positron Emission Tomography

Imaging Applications"; Solid-State Circuits, IEEE Journal of Volume 39, Issue 11, Nov. 2004 Page(s):1839 – 1852

[9] Pasi P., Kari M., Kostamovaara J.; "Pulsed Time-of-Flight Laser Radar Module With Millimeter-Level Accuracy Using Full Custom Receiver and TDC ASICs" ; IEEE Transactions on Instrumentation and Measurement; Vol. 51, Issue 5, Oct. 2002, Pages: 1102-1108.

[10] Dan I. Porat; "Review Of Sub-Nanosecond Time-Interval Measurements"; IEEE transactions on Nuclear Science. Vol. 20, pages: 36-51; 1973.

[11] Kalisz Josef, "Review Of Methods For Time Interval Measurements With Picosecond Resolution", Institute of Physics Publishing, Metrolgia Vol. 41, 17-32, 2004.

[12] Manuel Mota; "Design and Characterization of CMOS High-Resolution Time-to-Digital Converters"; PHD Thesis.

[13] Pouxe, J.; "A time to amplitude converter ASIC"; Nuclear Science Symposium and Medical Imaging Conference, 1991, Conference Record of the 1991 IEEE; 2-9 Nov. 1991 Page(s):626 - 628 vol.1.

[14] Wan-Li Sun et al; "A time to amplitude conversion chain (TACC) for time interval measurement"; Measurement Science Technology; 2000, vol. 11 N36-N38

[15] Elvi Ra¨ Isa Nen-Ruotsalainen, Timo Rahkonen And Juha Kostamovaara; "Integrated Time-to-Digital Converters Based on Interpolation"; Analog Integrated Circuits and Signal Processing, Vol. 15, 49–57 (1997).

[16] R.-Ruotsalainen E, Rahkonen T and Kostamovaara J; "A BiCMOS time-to digital converter with time stretching interpolators"; Proc. European Solid-State Circuit Conf. ESSCIRC'96 (Neuchatel, 17–18 September 1996) p 4.

[17] Rahkonen, T.E.; Kostamovaara, J.T.; "The use of stabilized CMOS delay lines for the digitization of short time intervals"; Solid-State Circuits, IEEE Journal of. Volume 28, Issue 8, Aug. 1993 Page(s):887 – 894.

[18] Chorng-Sii Hwang, Poki Chen, Hen-Wai Tsao, "A High-Resolution and Fast-Conversion Time-To-Digital Converter," Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on Volume 1, 25-28 May 2003 Page(s): I-37 - I-40 vol.1.

[19] Nassif, S.R; "Modeling and analysis of manufacturing variations"; Custom Integrated Circuits, 2001, IEEE Conference on.6-9 May 2001 Page(s):223 – 228.

[20] Arai Y. and Ikeno M., "A Time Digitizer CMOS Gate-Array with a 250 ps Time Resolution", 1996 IEEE J. Solid State Circuits 31, 212-20.

[21] Santos, D.M.; Dow, S.F. Flasck, J.M.; Levi, M.E.; "A CMOS delay locked loop and sub-nanosecond time-to-digital converter chip"; Nuclear Science, IEEE Transactions on Volume 43, Issue 3, Part 2, June 1996 Page(s):1717 – 1719.

[22] Manuel Mota, Jorgen Christiansen; "A High-Resolution Time Interpolator Based on a Delay Locked Loop and an RC Delay Line"; Solid-State Circuits, IEEE Journal of Volume 34, Issue 10, Oct. 1999 Page(s):1360 – 1366.

[23] A. Epstein et *al.*, "A CMOS Pulse-Shrinking Delay Element For Time Interval Measurement"; IEEE Transactions on Circuits and Systems-II Analog and Digital Signal Processing, Vol. 47, No. 9, Sept. 2000, pages: 954-958.

[24] Kostas Karadamoglou et *al.*; "A CMOS Time-to-Digital Converter for Space Science Instruments"; Proc. of the 28th European Solid-State Circuits Conference, 2002; 24-26 2002, pages: 707-710.

[25] Elvi Raisanen-Ruotsalainen, et *al.*, "A Low-Power CMOS Time-to-Digital Converter", IEEE Journal of Solid State Circuits, Vol. 30, No.9, Sept. 1995, pages: 984-990.

[26] S. Tisa, et al., "Monolithic Time-to-Digital Converter with 20 ps Resolution"; Proceedings of the 29th European Solid State Circuits Conference, 16-18 Sept.2003; pages: 465-468.

[27] Dudek, P.; Szczepanski, S.; Hatfield, J.V.; "A high-resolution CMOS time-to-digital converter utilizing a Vernier delay line"; Solid-State Circuits, IEEE Journal of Volume 35, Issue 2, Feb. 2000 Page(s):240 – 247.

[28] N. Abaskharoun and G. W. Roberts, "Circuits for on-chip sub-nanosecond signal capture and characterization," Proc. Of IEEE Conference on Custom Integrated Circuits, 2001, pp. 251–254.

[29] Kalisz, J., Szplet, R. Pasierbinski, J. and Poniecki, A.,"Field-Programmable-Gate-Array-Based Time-to-Digital Converter with 200-ps Resolution"; IEEE Transactions on Instrumentation and Measurement, vol. 46-1, pp.51-55, 1997.

[30] M. Mota, J. Christiansen, S. Débieux, V. Ryjov, P. Moreira, and A. Marchioro"; A flexible multi-channel high-resolution Time-to-Digital Converter ASIC"; Nuclear Science Symposium Conference Record, 2000 IEEE Volume 2, 15-20 Oct. 2000 Page(s):9/155 - 9/159 vol.2.

[31] C. Jorgan; "An Integrated High Resolution CMOS Timing Generator Based on an Array of Delay Locked Loops"; IEEE Journal of Solid State Circuits, Vol. 31, No, 7, July 1996.

[32] M. Mota, J. Christiansen; "A four channel, self-calibrating, high resolution, Time to Digital Converter"; Electronics, Circuits and Systems, 1998 IEEE International Conference on Volume 1, 7-10 Sept. 1998 Page(s):409 - 412 vol.1.

[33] Microchip; “Analog-to-Digital Converter Design Guide”; Application Notes.

[34] Dallas Semiconductor MAXIM; “INL/DNL Measurements for high-Speed Analog-to-Digital Converters”; Application Note 283.

[35] Friedman, E.G; “Clock distribution networks in synchronous digital integrated circuits”; Proceedings of the IEEE, Volume 89, Issue 5, May 2001; pages: 665-692.

[36] Xilinx SPARTAN^{TM}-3 Datasheets on www.xilinx.com

[37] Kalisz Josef, Z. Jachna, “Metastability tests of flip-flops in programmable digital circuits”; Microelectronics Journal, Volume 37, Issue 2, pages 174-180, Feb.2006

[38] Kalisz J., Zbigniew Jachna, “Dual-Edge Late-Transition Detector for Testing the Metastability Effect in Flip-Flops”; Proceedings of 16th International Conference on Microelectronics, Tunis, Dec. 2004.

[39] Joey Doernberg; “Full-Speed Testing of A/D Converters”; IEEE Journal of Solid State Circuits, vol. sc-19, No. 6, December 1984, pages: 820-827.

[40] Amir M. Amiri, Abdelhakim Khouas, Mounir Boukadoum, “On the Timing Uncertainty in Delay-Line-Based Time Measurement Applications targeting FPGAs; accepted paper to International Symposium on Circuit and Systems , New Orleans, 2007.

[41] Jian Song, Qi An, Shubin Liu; “A high-Resolution Time-to-Digital Converter Implemented in Field-Programmable-Gate-Arrays”; Nuclear Science, IEEE Transactions on, Vol. 53, Issue 1, Feb. 2006, Pages: 236-241.

[42] Jinyuan Wu et al., “Firmware-only Implementation of Time-to-Digital Converter (TDC) in Field-Programmable Gate Array (FPGA)”; IEEE Nuclear Science Symposium Conference Record, vol. 1, 19-25 Oct. 2003; pages: 177-181.

BIBLIOGRAPHY

M.Bolt, E.Cantatore, M.Socha, C.Aussems, J.Solo ; "Matching Properties of MOS Transistors and Delay Line Chains with Self-Aligned Source/Drain Contacts"; Proceedings of the 1996 IEEE International Conference on Microelectronic Test Structures, Vol. 9, March 1996

A.Pergoot, B.Graindourze, Er.Janssens, J.Bastos, M.Steyaert, P. Kinet, R.Roovers, W.Sansen; "Statistics for Matching"; Proceedings of the 1995 IEEE International Conference on Microelectronic Test Structures, Vol. 8, March 1995

Bo Zhou, Abdelhakim Khouas; "Measurement of Delay Mismatch Due to Process Variations by Means of Modified Ring Oscillators"; IEEE International Symposium on Circuits and Systems, Vol.5, ISCAS, 2005, pp. 5246-5249

Chorng-Sii Hwang, Poki Chen, and Hen-Wai Tsao, "A High-Precision Time-to-Digital Converter Using a Two-Level Conversion Scheme"; Nuclear Science, IEEE Transactions on Volume 51, Issue 4, Aug. 2004 Page(s):1349 - 1352

M. S. Gorbics, J. Kelly, K. M. Roberts and R. L. Sumner; "A High Resolution Multihit Time to Digital Converter Integrated Circuit"; Nuclear Science, IEEE Transactions on Volume 44, Issue 3, June 1997 Page(s):379 – 384

Elvi Räisänen-Ruotsalainen, Timo Rahkonen, and Juha Kostamovaara; "An Integrated Time-to-Digital Converter with 30-ps Single-Shot Precision"; Solid-State Circuits, IEEE Journal of Volume 35, Issue 10, Oct. 2000 Page(s):1507 – 1510.

Y. Araj; "A cmos time-to-digital converter VLSI for high-energy physics"; VLSI Circuits, 1988. Digest of Technical Papers. 1988 Symposium on Aug 22-24, 1988 Page(s):121 – 122.

Alex Kirichenko, Saad Sarwana, Deep Gupta, Irwin Rochwarger, and Oleg Mukhanov; "Multi-Channel Time Digitizing Systems"; Applied Superconductivity, IEEE Transactions on Volume 13, Issue 2, June 2003 Page(s):454 – 458

Ryszard Pelka et *al.* , "Nonlinearity Correction of the Integrated Time-to-Digital Converter with Direct Coding"; IEEE Transaction on Instrumentation and Measurement, Vol. 46, No. 2, April 1997,pages: 449-453

Jinyuan Wu et al., "Firmware-only Implementation of Time-to-Digital Converter (TDC) in Field-Programmable Gate Array (FPGA)"; IEEE Nuclear Science Symposium Conference Record, vol. 1, 19-25 Oct. 2003; pages: 177-181.

Stanley L. Hurst; "VLSI Testing: digital and mixed analogue/digital techniques"; the institution of Electrical Engineers, London, United Kingdom; 1998.

Volnei A. Perdroni; "Circuit Design with VHDL"; MIT Press, 2004

Saeed Ghahramani; "Fundamentals of probability with Stochastic Processes"; 3rd Edition, Prentice Hall 2005

XILINX SPARTAN^{TM}-3/VIRTEX application notes (www.xilinx.com)

APPENDIX-I

Conference papers

1. Proceeding of NEWCAS-2006, Gatineau, Qc, Canada
2. Accepted paper for ISCAS-2007, New Orleans, US

Low Dead Time, Multi-hit FPGA-Based Time-to-Digital Converter

Amir Mohammad Amiri[1], Mounir Boukadoum[2], Abdelhakim Khouas[1]
1. École Polytechnique Montréal, Canada
2. Université du Québec À Montréal, Canada
amir-mohammad.amiri@polymtl.ca, boukadoum.mounir@uqam.ca, Abdelhakim.Khouas@polymtl.ca

***Abstract* – This paper presents improvements on a novel FPGA-based multi-hit Time-to-Digital Converter (TDC) to measure time intervals with a resolution of 100ps and a variable dynamic range controlled by a binary coarse counter. We use a matrix topology to provide a two-level resolution, aiming to minimize the overall measurement time. The conventional dead time is eliminated by the continuous detection and processing of data by two delay matrices operating in parallel. A back-resetting scheme eliminates the erroneous multi-detection of an event along matrix tap lines. The circuit was tested on a XILINX SPARTAN-3 FPGA platform.**

I. Introduction

Precise measurement of time intervals is required in many applications of science and engineering. The time interval to be measured (T_m) is usually defined between the rising edges of two pulses commonly named as START and STOP. T_m typically consists of fine and coarse sub-intervals as shown in Fig. 1. The coarse section, T_2 is divisible by the system clock period, T_{ref}, but to measure the fine sections, T_1 and T_3, further interpolation of the reference clock (division of T_{ref} into many smaller time bins) is needed.

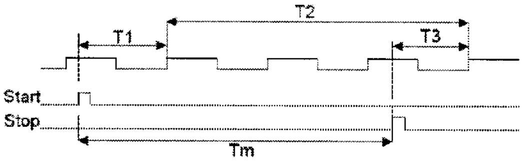


Fig. 1: Representation of time interval to be measured

Many research efforts have been devoted to develop time interpolators, both in the analog and the digital domains. The well known time-to-amplitude conversion (TAC) method [8] is an analog solution where T_m is converted into a voltage amplitude v by charging a capacitor with a constant current for the duration of T_m. Practical resolutions of 1ps to 20ps have been reported using the TAC method. However, such analog solutions suffer from the long conversion time to convert v into a binary representation. On the other hand, a Time-to-Digital Converter (TDC) circuit directly measures and converts T_m to a binary word for digital processing.

While most TDC solutions were previously realized in ASIC, recent advances in programmable logic devices, such as FPGAs, have enabled researchers to focus on the flexibility, the low cost, and the shorter prototyping time aspects of a design supported by the latter. This paper presents an FPGA-based design with a 100ps of measurement resolution. The organization of the paper is as follows: Section 2 provides a brief review of popular digital TDC methods; section 3 recaps the architecture in [1] and presents our modifications to improve performance. Simulation and experimental results are presented in section 4, followed by a conclusion in section 5.

II. Some existing time interpolators

A. Simple Delay Lines

The basic digital time sampler uses a delay chain to slice the clock period into N equal time bins. The delay line illustrated in Fig. 2 measures the time interval between the START signal, applied at the *data* input, and the STOP signal applied at the *clk* input. This simple time interpolation method is easy to implement, but it is limited to a measurement resolution of single gate delay t_d and sensitive to delay mismatches between successive stages. A more effective interpolation is achieved if the delay line is replaced by a delay-locked loop (DLL), which improves stability through the negative feedback [2].

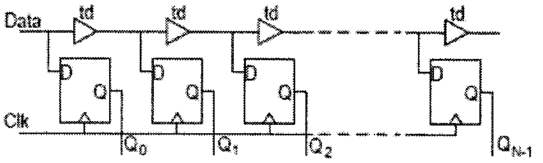


Fig. 2: Simple time interpolator

B. Vernier Delay Line (VDL)

VDL interpolators measure T_m by propagating the data and clock signals through two delay lines, A and B, where the propagation delay of elements in line A is slightly larger than those of line B [3]. The VDL depicted in Fig. 3.a) achieves a resolution of $\Delta t = t_a - t_b$ with t_a and t_b as the respective propagation delay of the elements in lines A and B. An alternative VDL technique uses a chain of buffer elements and a chain of latches as depicted in Fig. 3.b). The START signal is applied at the *in* input and propagates through the chain of latches, while the STOP signal is applied at the *clk* input and propagates through the chain of buffers. The STOP signal disables the latches when in phase or in lead from the START signal. Hence, T_m is measured with a resolution $\Delta t = t_L - t_B$. In [4], an FPGA-implementation achieves 200ps resolution.

Fig. 3.c depicts a single-inverter implementation of the VDL principle to overcome the increased number of counters and the sensitivity of the delay line to mismatch in identical components [5]. Here, the *data* and *clk* edges activate two single-inverter ring oscillators generating waveforms with respective periods of $2T_s$ and $2T_f$ with $T_s > T_f$. This improved VDL achieves a high resolution of $\Delta T = T_s – T_f$.

C. Array Structures

In an effort to further achieve high measurement resolutions, some TDC circuits combine multiple single time interpolators in an array topology [6] [7].

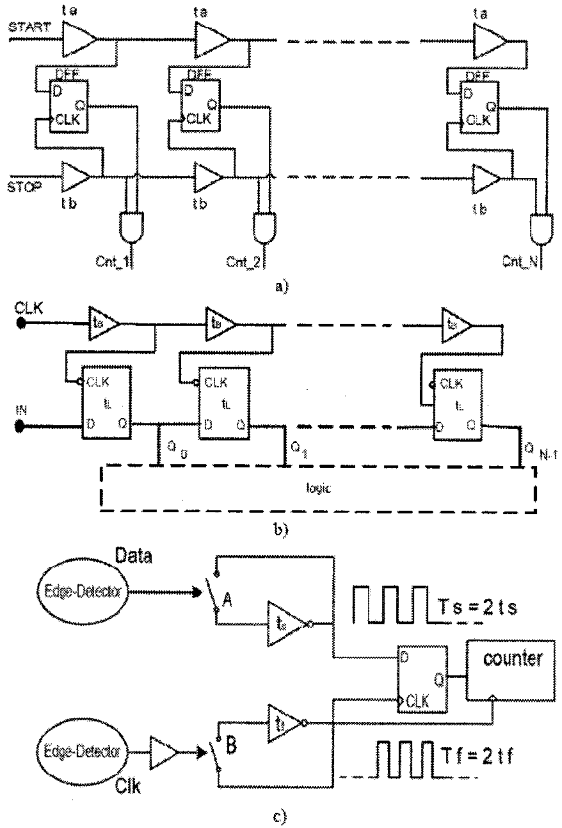


Fig. 3: a) Buffer-only VDL; b) Latch and buffer VDL; c) Single-inverter VDL

Ref. [6] presents a fine time interpolator that combines M simple time interpolators with N stages such as the one in fig. 2. In this topology, the data pulse is made to propagate through M time samplers with the clock and data inputs of any two adjacent time samplers delayed by respective small delays of t_c and t_d, through which the single interpolator resolution of t_d is broken into M, giving a final resolution of $\Delta t_r = t_d / M$. Ref. [6] reports an ASIC implementation with M=4, and a conversion time of less than 2.5 ns at 400 MHz clock speed.

In [7], multiple phase-shifted DLLs were combined in an array structure to provide improved measurement resolution. The small phase shift among subsequent DLLs was achieved using the difference in fixed delay buffer elements in a driving DLL and the buffer elements in the DLL components of the array. The high measurement resolution in this topology was achieved at the expense of a complicated encoding scheme for the tap line readings. The authors reported an RMS resolution of 34.3 ps, a dynamic range of 3.2 us, and an LSB (time bin) of 89.3 ps at a clock frequency of 80 MHz.

III. Proposed Architecture

A multi-hit TDC circuit targeting FPGA technology was proposed in [1] to address the time overhead (dead time) associated with the single VDL reported in [4].

A. Delay Matrix Operation

Shown in Fig. 4, the circuit uses a matrix of delay cells composed of N individual VDL lines interconnected through their first delay cells. The propagation of an incoming pulse along a single VDL line is disabled by the rising edge of the clock. The row-wise interconnection provides for the data pulse to jump to the subsequent VDL line should its arrival time with respect to the next clock edge be larger than t_L. Therefore, by analogy to a ruler with both *cm* and *mm* ticks, the matrix topology breaks the active phase of the reference clock into N time bins of resolution R_y=t_L, with each bin further broken into M smaller bins of resolution R_x=t_L–t_B, where t_B and t_L are the buffer and latch propagation delays, respectively.

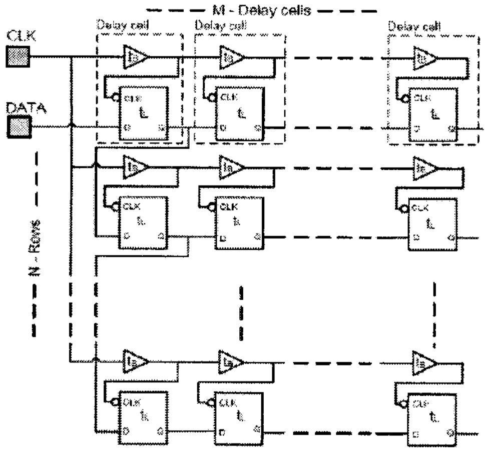


Fig. 4: Matrix of delay cells

Fig. 5 depicts the offsetting of the pulse arrival time by t_L each time it moves from one row of the matrix to the next. At the last row, the sub-interval that results is shorter than t_L and the pulse is propagated horizontally with finer time steps.

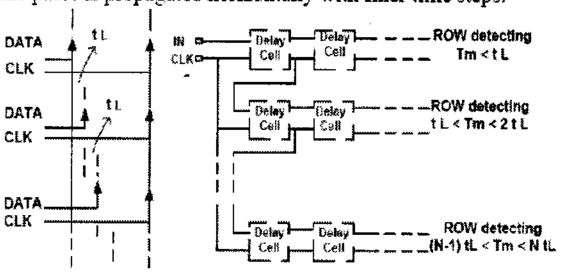


Fig. 5: Pulse forwarding to subsequent row

Illustrated in Fig. 6, the pulse arrival time, t_e, in the active phase is computed using the interval T_e it takes for the pulse to reach *(K+1)* T_0. T_e is determined by reading the (n,m) coordinates of the delay matrix. We have:

$$t_e = (k+1)T_0 - T_e = (k+1)T_0 - (nR_y + mR_x) \quad (1)$$

The time intervals between sequentially incoming pulses are measured by stamping the pulses' arrival times relative to the next rising/falling edge of the reference clock. Taking the difference in arrival times will result in the intervals of interest, that is $T_m = t_{e2} - t_{e1}$.

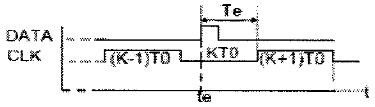


Fig. 6: Event time presentation

The maximum measurable interval using the delay matrix is given by the propagation delays of N-1 vertical latches and the M horizontal differential delay R_x. That is: $T_{max} = (N\text{-}1)\ R_y + MR_x$. To measure T_{max}, the pulse needs to propagate through (N+M-1) latches, at coordinates (N-1, M), requiring a total measurement time of $T_{total} = (N + M\ \text{-}1)\ t_L$. The maximum time overhead T_{ex} to measure T_{max} is:

$$T_{ex} \;= T_{total}–T_{max} = (N + M\text{-}1)\ t_L - [(N\text{-}1)R_y + MR_x] = M\,t_B \quad (2)$$

This relationship still holds if T_{max} is replaced by T_m with coordinates (m, n) leading to measurement time overhead mt_B. Another important feature is the implementation of two matrices operating in parallel, with clock signals 180 degrees out of phase. Having two delay matrices operating in both phases of a clock period allows for the continuous detection and processing of sequential hit signals subject to the minimum pulse pair resolution (PPR). This considerably reduces the conventional dead time needed to reactivate the system for next round of measurement.

B. Proposed Improvements

Despite the aforementioned advantages, the matrix structure described in [1] may produce erroneous readings by detecting the same pulse in more than one row. Also, no explicit analysis of how to choose matrix size along with the involved trade-offs was made in [1]. There also exist design and performance issues such as meta-stability in cascaded latches, and managing of clock skew and component delays in FPGA implementation of the matrix that designers need to be aware of.

Multi-Detection Correction: Ideally, when a pulse moves to a subsequent row, it should no longer exist on the previous one. In the delay matrix proposed in [1], this assumption fails due to the delay mismatch of cell components along a single line and the small non-zero difference in time between the vertical and cumulative horizontal resolutions ($\Delta t = R_y – MR_x$). This is particularly critical for pulses distant from the reference clock edge by delays comparable to t_L.

To remedy this problem, we propose to reset the earlier row as the pulse traverses to the next, thus ensuring that not two adjacent rows will contain the same data pulse. The new topology is shown in Fig. 7. Each row of the matrix, with the exception of the first, is equipped with a reset circuit that, upon detection of a new pulse, will assert a constant pulse width reset signal to clear all the delay cells in the previous row. Moreover, an internal constant data pulse generator was incorporated to prevent any re-propagation of the lagging part of a larger-width incoming pulse to the next active phase that otherwise would produce erroneous readings.

Criterion for Matrix Size: The choice of matrix size primarily depends on the target observation time window T_{obs} and the vertical resolution R_y. Given T_{obs} and R_y, the number of rows N and M of delay cells in a single row obey:

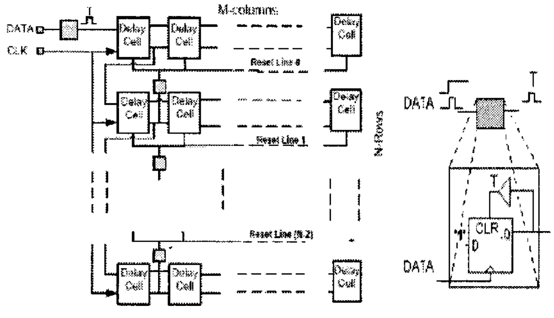


Fig. 7: a) New matrix topology b) Constant data and reset pulse generator

$$N \geq T_{obs} / R_y \quad with\ Rv = t_L\ and\ T_{obs} = T_{ref}/2 \quad (3)$$

$$M \geq R_y / R_x = t_L /(t_L - t_B) \quad (4)$$

T_{obs} is governed by $T_{delayed} = Mt_B$ to allow complete propagation of a data pulse throughout all the matrix delay cells, and the minimum interval from sample edge to the next clock edge ($T_{proc} = T_{ref}/2 - T_{delayed}$) required by the active matrix to reset its tap lines before the next active phase starts. Hence, T_{obs} is given by:

$$T_{obs} = T_{delayed} + T_{proc} \quad (5)$$

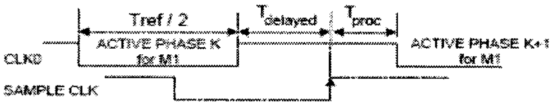


Fig. 8: Sampling of delay matrix tap lines for a matrix M1 clocked by CLK0

Thus, the size of the delay matrix depends on the measurement resolution sought, the buffer and latch delays present, and the value of T_{obs}. The ideal size is obtained if the desired resolution $R_x = t_L – t_B$ can be achieved by small-delay buffer and latch components, leading to a smaller observation window, and subsequently a smaller matrix size.

FPGA design considerations: Due to the fixed architecture of FPGA circuits, controlling of component delays is a major challenge for the designer. Automatic placement and routing with unpredictable delays become useless in applications such as the TDC matrix. In such cases, the designers must place and route the design manually. Some FPGA circuits may have built-in delay lines in order to address the uniformity and predictability of delays. However, such delay lines confine designers to limited design sizes for applications such as the TDC matrix, making the performance architecture-dependent.

Another important design consideration regards the meta-stability phenomenon in cascaded latch/FF components and the clock skew among subsequent VDL lines that affect the behavior of the matrix structure. The meta-stability behavior of a latch/FF component is observed when the data at the D input changes in the vicinity of the clock edge, resulting in stretching of the propagation delay of the Latch/FF or even in an unpredictable final state of the latter. FPGA-implemented designs seeking higher measurement resolutions are found to be more susceptible to meta-stability effects [9]. Meta-stability effects may be reduced if each FF in the cascaded lines is replaced by two FF sharing a common clock. However, using double latches in the matrix topology may increase matrix size and observation time window, given the target resolution.

In addition to meta-stability effects, the behavior of the delay matrix is vulnerable to clock skew between subsequent rows. To minimize clock skew, FPGA device manufacturers usually embed specific clock routings to feed the clock inputs of logic blocks. In designs such as the delay matrix, where the clock signal feeds components other than a latch/FF, the clock skew is relatively high, heavily affecting the expected behavior. Under such circumstances, special design measures need to be taken to exploit the existence of special clock nets.

IV. Simulation and Experimental Results

Illustrated in Fig. 9, the test circuit is composed of two delay matrices operating in parallel, a custom–size FIFO to provide single cycle data storage, a data encoder, a FSM-based controller, a shifter, a 512 X 32 RAM, and a LCD display driver. The system was implemented on a SPARTAN-3 FPGA platform to testimony low-cost feasibility. The delay matrices, driven by two 180-degree out-of-phase clocks, generate tap outputs in either phase of the common clock. To attain the required uniform and predictable delays for the latch and buffer components, each delay cell of the matrices was placed manually. Results from timing simulation and the timing analyzer of ISE tools from XILINX show average values of R_y = 1.4 ns and R_x=100 ps. For T_{obs} = 35ns, using (3) and (4) the size of each delay matrix was chosen to be 25 x 15 (N=25 and M=15) including the extra cell padded at the end of each line to minimize (R_y–MR_x>0).

The system operates as follows: The matrices' tap outputs and the coarse counter values are sampled into the data encoder registers where they are encoded as 108-bit words (100-bits for encoding matrices' tap lines plus 8 counter bits) using the encoding scheme in [1]. The encoded word is then stored onto a 16-deep, 108-bit word line FIFO unit. To ensure the validity of data in the matrix, the delay cells are reset prior to entering a new active phase of detection. The system continuously detects and stores non-zero data as long as enough storage area is present in the FIFO (We used a small FIFO for demonstration purpose.) Once the FIFO is full, the stored data is transferred to the RAM and then displayed on the LCD display. The test data pulses and the various variable frequency clock signals were generated internally by digital clock managers (DCM), and counters. Simulation results provided tap readings that agree with the given test intervals. Additionally, the system was also tested with an early implementation by observing tap readings on LCD display for the given pulse sequences.

Table I tabulates a sample inter-pulse interval calculation for the reading "14000030000000500000000000B" in RAM with B as the value *m* in hexadecimal notation corresponding to first row in the matrix. The hexadecimal value 14 is the coarse counter value corresponding to phase (K+1). The thermometric coding of the output taps along a row implies a theoretical PPR value of R_y, and a single pulse per row. However, PPR is affected by the internal reset lines that keep earlier rows reset for the duration of the reset pulse, blocking pulse propagation. The actual observed PPR was 8.0 ns.

V. Conclusion

We presented improvements to our earlier work [1] by: 1) introducing an inter-row back-resetting scheme to eliminate the erroneous tap readings caused by multi-detection of a single pulse; 2) reducing the measurement resolution to 100ps; 3) eliminating the erroneous propagation of a wider incoming data pulse through a constant pulse generator. We also developed an analytic criterion for selecting the size of the delay matrices. Furthermore, the paper touched upon important considerations relative to designs that use cascaded FF/latch elements and targeting FPGA devices. The tradeoff of our improvements was a reduced PPR of 8.0ns. Our current focus is to improve and characterize the early implementation by further testing and incorporating a calibration scheme to account for component delay mismatch.

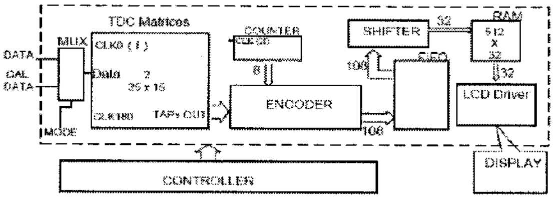


Fig. 9: Overall test system for the new delay cell matrix

TABLE I
Sample Inter-pulse Interval Calculation

E_i	(n_i, m_i)	T_e (ns)	*Event Time t_i (ns)	T_m
1	(20,3)	28.3	$\tau_2 - T_e = \tau_1 + 6.7$	t2 - t1 = 11.0 ns
2	(12,5)	17.3	$\tau_2 - T_e = \tau_1 + 17.7$	t3 - t2 = 16.2 ns
3	(0,B)	1.1	$\tau_2 - T_e = \tau_1 + 33.9$	t3 - t1 = 27.2 ns

* $\tau_2 = (K+1) \times T_{obs}$ = 20 (hex 14) x 35ns = 700 ns; $\tau 1 = (K) \times T_{obs}$ = 19 ($13) x 35ns = 665ns;

References

[1] M. S. Andaloussi, et al., "A novel time-to-digital converter with 150 ps time resolution and 2.5 ns pulse-pair resolution", in Microelectronics, The 14th International Conference on 2002 – ICM, 2002, pp. 123-126.

[2] M. Mota, J. Christiansen, "A High-Resolution time Interpolator Based on a delay Locked Loop and an RC Delay Line", Solid-State Circuits, IEEE Journal of, Vol 43, Issue 10, Oct.1996, Pages: 1360 - 1366.

[3] Dudek, P.; Szczepanski, S.; Hatfield, J.V.; "A high-resolution CMOS time-to-digital converter utilizing a Vernier delay line," Solid-State Circuits, IEEE Journal Volume 35, Issue 2, Feb. 2000 Page(s):240-247.

[4] Kalisz, J., Szplet, R. Pasierbinski, J. and Poniecki, A.,"Field-Programmable-Gate-Array-Based Time-to-Digital Converter with 200-ps Resolution"; IEEE Transactions on Instrumentation and Measurement, vol. 46-1, pp.51-55, 1997.

[5] Antonio H. Chan, Gordon W. Roberts, "A Jitter Characterization System Using a Component-Invariant Vernier Delay Line,", IEEE Transactions On (VLSI) Systems, VOL. 12, NO. 1, JANUARY 2004.

[6] Chorng-Sii Hwang, Poki Chen*, Hen-Wai Tsao , "A High-Resolution and Fast-Conversion Time-To-Digital Converter," Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on Volume 1, 25-28 May 2003 Page(s):I-37 - I-40 vol.1.

[7] M. Mota, J. Christiansen, "A four channel, self-calibrating, high resolution, Time to Digital Converter," Electronics, Circuits and Systems, 1998 IEEE International Conference on Volume 1, 7-10 Sept. 1998 Page(s):409 - 412 vol.1.

[8] Kalisz Josef, "Review Of Methods For Time Interval Measurements With Picosecond Resolution", INSTITUTE OF PHYSICS PUBLISHING, METROLGIA VOL. 41, 17-32, 2004.

[9] Kalisz Josef, Z. Jachna, "Metastability tests of flip-flops in programmable digital circuits"; Microelectronics Journal, Volume 37, Issue 2, pages 174-180, Feb.2006

On the Timing Uncertainty in Delay-Line-based Time Measurement Applications Targeting FPGAs

Amir M. Amiri, Abdelhakim Khouas
Department of Electrical Engineering
École Polytechnique de Montréal
Montreal, Canada
amiri@grm.polymtl.ca, akhouas@polymtl.ca

Mounir Boukadoum
Computer Science Department
Université du Québec à Montréal
Montreal, Canada
boukadoum.mounir@uqam.ca

***Abstract*— this paper addresses important performance issues in delay-line-based timing applications targeting FPGA devices. The circuit under test is a TDC circuit implemented on a low-cost FPGA from XILINX. Various performance limitations such as uncertainty and non-uniformity in cell delays are described and corresponding optimization and improvement suggestions are made. Experimental results were obtained using ring oscillator-based test structures to inspect intra-die delay mismatches along the target FPGA's surface.**

I. Introduction

Recent advances in the sub-micron silicon technology have defeated the early technology limitations imposed on circuit designers in the areas of Application-Specific Integrated Circuits (ASIC) and the programmable logic devices such as Field Programmable Gate Arrays (FPGA). The current trend in transistor density per logic device has augmented the popularity of FPGAs more than ever. Furthermore, the many embedded common-use logic blocks within FPGA devices such as memory blocks, multiplier units, delay-locked loops (DLL), designated delay lines, and other optimized logic primitives as library components have greatly simplified system design, almost eliminating the need for external resources. These optimally designed and placed embedded blocks within the FPGA devices equip system designers with the necessary tools in order to come up with robust and complex designs incorporating several circuit blocks into a single functional system.

In some applications, it is important to have precise control over propagation delays within the FPGA fabric. This is the case of FPGA-targeted designs involving delay line-based time measurement application such as those reported in [1]-[4]. Given the importance of accuracy in timing applications, we discuss bottlenecks and suggest optimizations in those designs targeting FPGA devices. Although the results and discussions in this paper are based on the FPGA-implemented Time-to-Digital Converter (TDC) circuit in [1], the importance of limiting factors in restricting performance goes beyond, affecting any delay-line-based timing application targeting FPGA devices. The organization of this paper is as follows: Section II briefly recaps the FPGA-implemented TDC design under test. The various performance limitations imposed on the TDC design are discussed in section III. Section IV outlines optimization and improvement guidelines. We conclude in section V.

II. Recap of TDC under Test

The circuit under test is based on a time interpolator utilizing the vernier principle in order to achieve single-shot high measurement resolution. Shown in Fig. 1, the time interpolator is an array of buffer-latch vernier delay lines (VDL) with the interconnection between successive rows achieved through the first delay cells of each line and a common reference clock line shared by all lines. The delay matrix measures the distance in time between the rising edges of the data and the subsequent clock signals applied at the respective data and clock inputs of the matrix. The matrix structure breaks this interval into two-level time bins of $\tau_Y = t_L$ and $\tau_X = t_L - t_B$ with t_B and t_L as the propagation delays of the buffer and latch elements of the matrix delay cells. The vertical interconnection provides a propagation path with larger time bin τ_Y whereas the single-shot high resolution τ_X is achieved by horizontal propagation throughout the vernier delay lines. The TDC circuit measures an interval of time between two incoming pulses by taking the difference in corresponding arrival times with respect to some reference point. The circuit was implemented onto a low-cost XILINX Spartan^{TM}-3 FPGA platform.

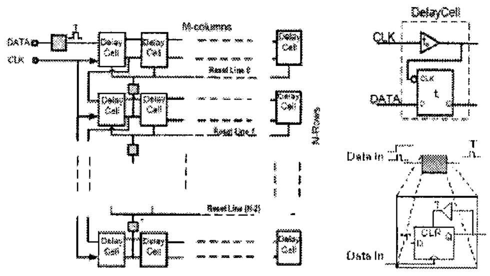


Figure 1. Delay Matrix as Time Interpolator

This work was partially supported by grants from NSERC and ReSMiQ in Canada.

III. Performance Bottlenecks

Logic mapping onto FPGAs is mainly achieved using Configurable Logic Blocks (CLB) and Input/Output Blocks (IOB) within the FPGA technology. Timing applications utilizing delay lines may not afford the automatic logic mapping and placement due to the tight timing requirements, emphasizing the need for manual placement. For instance, most of the delay-line-based time interpolators achieve small time steps by propagating the incoming pulse(s) through delay line(s) ideally composed of uniformly time-spaced delay components. The actual delay of these components may depend upon factors such as the logic elements forming the delay component, the target silicon technology used (FPGA or ASIC), the intra-die delay mismatch, the uniformity of the inter-cell routing delays, and other design-dependent factors.

A. *Uncertainty in Logic Delays*

Any application involving a delay line composed of identical logic elements is sensitive to delay mismatch. Delay mismatch is the direct cause of variations in electrical and physical characteristics of devices caused by process and mask imperfections during fabrication and the design-dependent environmental factors such as the variations in power supply and temperature [5]. To investigate the effects of intra-die variations on cell delays in the target FPGA used in [1], several experiments were conducted. The experiments involved reconfiguration of the buffer-latch VDL lines into ring oscillators (RO) by inserting identical inverting cells at the beginning of each line as shown in Fig 2. An effective test circuit in characterizing delay mismatch was presented in [6], but due to the symmetrical topology requirement and the internal structure of the FPGA it may not be applicable to characterize the delay mismatch in the TDC under study. In the current test structure, the RO lines were carefully placed and routed onto FPGA logic elements by following placement guidelines (described in section 4) to remove any topology-dependent delay non-uniformity. The periods of the ring oscillators can be found from the count value of a binary counter incremented by the RO-generated periodic signals for the duration of count interval **T**. The error in the calculated periods of RO becomes negligible if a large count interval is used. The results in the first three rows of Table-I were obtained by turning the VDL lines of both matrices of the design into ROs. This way, all matrix rows, one by one, are reconfigured into RO and corresponding incremented count values are stored in RAM before control of the counter is given to the subsequent RO of the matrix of concern. In order to investigate delay mismatch along the entire surface of the FPGA, the experiment was repeated by placing the 25 x 15 matrices' logic on the upper, middle, and bottom sections of the FPGA surface. By design, matrix-1 (M1) occupies the left half of the FPGA surface area whereas matrix-2 (M2) is placed on the right half. The delay data corresponding to Full Matrix RO Configurations, shown in Table 1 are the average values taken from 25 rows of the matrices with the enlisted standard deviations. The last row of Table I indicates delay values that were obtained by placing a single RO-configured VDL at 25 random locations of the FPGA.

TABLE I. Cell delay mismatch within FPGA surface

RO-Configuration	Matrix	Buffer delay (t_B)		Latch delay (t_L)	
		μ (ns)	σ (ps)	μ (ns)	σ (ps)
Full-Matrix (Top)	M1	1.173	7.33	1.229	12.9
	M2	1.1681	5.54	1.2227	7.93
Full-Matrix (Mid)	M1	1.1788	8.6	1.2278	15.93
	M2	1.1713	4.77	1.2233	11.97
Full-Matrix (Bot)	M1	1.173	5.77	1.228	16.2
	M2	1.1776	6.35	1.2332	14.42
Single-VDL at random locations	-	1.1271	6.95	1.2419	19.375

TABLE II. Inter-cell delay mismatch

Data Flow Direction	Buffer delay (t_B)		Latch delay (t_L)	
	μ (ns)	σ (ps)	μ (ns)	σ (ps)
Left-to-Right	1.1347	8.87	1.35	22.2
Right-to-Left	1.12	5.03	1.26	8.41

The test results for inter-cell delay mismatch using a single VDL are tabulated in Table II. The data were obtained by a cell-removal procedure where the constituent cells of a line were removed one by one and corresponding effects on the period of the ring oscillator observed. This process was conducted on all 15 cells of the VDL used, at several random locations on the surface and in both data flow directions. As the mean and standard deviation values in Tables I and II show, the component delays in the matrix are acceptable for designs seeking measurement resolutions around 80-100 ps. It is to note that the buffer/latch placement scheme in this experiment was only to investigate cell delay mismatch and obtained delay values do not serve to determine the final single-shot resolution of the TDC.

B. *Non-Uniformity of Cell Delays*

The theoretical uniform propagation delay of cells in delay-line-based designs may also be affected by design-dependent factors as well. In [1] the basic component of the proposed delay matrix is composed of a buffer logic element and a negative level sensitive latch. The element delay consists of the logic and net delays. The logic delay of a cell depends on logic mapping of the component while routing delays are location-dependent with respect to adjacent cell.

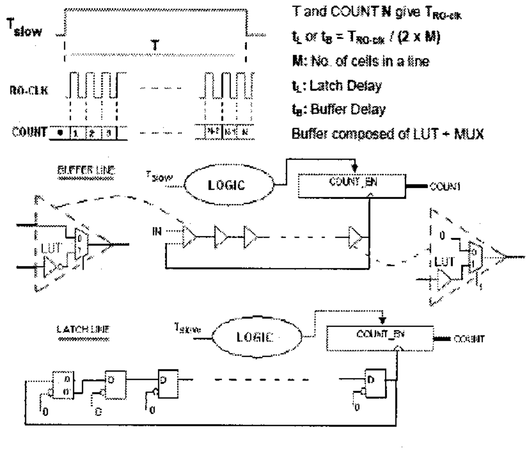


Figure 2. Ring Oscillator configuration of the buffer and latch lines

The choice of the actual mapping from a symbolic buffer onto a logic element varies. Depending on the design specification, some designs may implement a buffer as a cascaded even number of inverters, whereas other designs may use a controlled logic element such as an AND gate with one input tied to V_{DD}. Regardless of the actual logic mapping, while the non-uniformity of basic cell delays is a matter of concern for both ASIC and FPGA designs, the choice of the logic implementation in the latter is further influenced by *the fixed internal structure of a CLB slice in terms of basic logic elements, the limited number of general I/O pins of a CLB slice, the pre-determined routing possibilities*, and *the structure of the CLB switch matrix that selects one out of several interconnects*. Due to the great number of cells in the matrix and the critical dependence of matrix performance on symmetric placement, the best choice for mapping a buffer in [1] seemed to be a single-input Look-Up Table (LUT) contained in the CLB slice. However, although placing the mapped logic elements at equidistant CLB units on the FPGA surface is helpful, the actual uniformity of delays is not guaranteed. Uniformity may be achieved if the designer controls both the logic placement and the interconnecting media among delay cells such that the routing structure is also uniform.

C. *Skew along Clock Paths*

Defined as the difference in clock path delays from the source to adjacent clocked elements, clock skew is a major issue in the design of any synchronous system. In general, the sensitivity of designs to clock skew is relatively higher in FPGA-oriented designs than in the ASIC counterparts with careful custom layout design. The problem worsens in timely critical designs where the clock lines must drive asynchronous logic rather than the optimally designed global clock lines within the FPGA that are pre-routed to drive the clock input of FF in CLBs. In designs such as in [1], clock skew is problematic as the elements of the delay line(s) share the same clock with the assumption of equal arrival time of the clock edge at the destination cells. The existence of skew along the common clock path of a delay matrix in [1] causes enlargement/shrinkage in the ideal latch delay value ($\tau_Y = t_L$) of the larger time bins in the delay matrix. Consider two sequential rows R_i and R_j ($j = i + 1$) of the delay matrix. The vertical time slot (τ_Y) achieved by data propagation from R_i to R_j is enlarged when the clock line on R_j suffers from negative skew. With a negative skew value of Δt on R_j, the distance between the data and the clock edges further decreased by Δt leading to a corresponding vertical time slot value of $\tau_Y = t_L + \Delta t$. Similarly with positive skew along the clock line of R_j, the distance between the propagating data and the clock edges is further increased by Δt leading to a time slot value $\tau_Y = t_L - \Delta t$. This enlargement/shrinkage in the value of τ_Y causes a shift in the expected final pulse locations, ultimately bringing about non-linearity in the transfer curve of the converter.

D. *Placement and Routing Issues*

The emphasis on manual logic placement and routing comes from the fact that automatic place and route tools become useless for critical components as they may place and route logic elements with unpredictable delays. Logic mapping, placement, and routing processes in FPGA designs are normally optimized by means of constraints applicable at the corresponding stages of the design flow. During the mapping process, constraints can be used to guide the tool to select specific logic primitives to be used for the low level logic in the synthesis netlist. Similarly, placement constraints can be used to define particular logic element at a specific location of the target FPGA, whereas routing constraints may define the desired net interconnections among logic elements out of several available routing possibilities. Therefore, to achieve optimal solutions for timely critical applications, designers need first to exploit the resources available to efficiently apply the applicable synthesis, map, place and route constraints. That being said, it is worth mentioning that regardless of the relatively large number of available applicable constraints, not all applications targeting FPGAs may be optimally designed. The task of reaching an optimal design using manual place and route could then become the only alternative, albeit a challenging and time consuming process requiring many test-and-run cases. In [1] uniformity of logic and routing delays were achieved by testing all combinations of buffer/latch placements within CLB slices, for several dataflow directions within the FPGA.

Timing applications with symmetric topologies may also be limited by the number of available resources within the target FPGA surface. For instance, if the delay matrix in [1] required a number of cells greater than the available number of CLBs per row of the FPGA surface, then the symmetry of the design would be affected. Similar limitations occur when constituent elements of a module need to be placed in a contiguous section of the FPGA surface, for example the FF elements in a binary counter for fast carry chain logic. This problem is of more concern in designs where the required area for the symmetric logic placement is large, confining other modules to smaller dispersed surface of the chip.

The variation in edge locations of the reference clock signal from their expected ones also referred as signal jitter, acts as a performance bottleneck as well. In delay-line-based measurement circuits, clock jitter causes unexpected additions or reductions to the actual time interval to be measured. In [1] jitter may shrink/enlarge the detection time window of the TDC, leading to edge misalignment of the out-of-phase clocks used for two matrices in parallel. This creates dead zones in the detection window, deactivating both matrices and causing loss of hit signals in their vicinity.

IV. SUGGESTED OPTIMIZATION STRATEGIES

Although most of the imposed restrictions on FPGA designs may originate from the predetermined structure and possibly the limited application-dependent resources, a good portion of restrictions may be self-imposed by the designer due to insufficient knowledge of the target technology. While, at first glance, custom design for system components appears to be appealing, using built-in primitives to implement the component of interest may result in great savings in terms of space and temporal resources. In the TDC under study, built-in resources such as the DCM proved helpful in providing the multiple phase shifted clocks. Similarly, enormous amount of CLB registers may be saved if temporary storage units such as First-In-First-Out (FIFO) are mapped onto available block RAM units with minimum control logic. Thus, when working with FPGAs, a good design practice is to acquire adequate knowledge of the target technology by reviewing the vendor application notes that outline the functionality and the optimal methods of using built-in primitives.

Another way to improve the performance of a system on FPGA is through application of design constraints that are guidelines for the design tools to select one out of many available alternatives related to map, place and route processes based on the desired system behavior. Given the fixed surface structure, designs may be optimized if system components are mapped onto a logic resource that is optimal for the component of concern. On the same token, the placement of the best-fit mapped logic element in an optimal location within the FPGA surface may further improve system behavior. In time measurement designs, the application of constraints is a necessary step to meet the timing requirements of timely critical components such as the logic elements of the delay lines. The uniformly time-spaced delaying process for the clock and data edges by means of buffer or latch lines requires firm control of the routing and logic placement processes. One strategy to achieve uniformly-spaced delaying by means of delay lines is to map the design element onto a CLB logic element such that the component has single entry and exit points, leaving a single routing possibility among delay cells of the line(s). Next, one has to make sure that all delay cells share the same single-in single-out topology using the same primary input and output pins of their corresponding CLB slices. This strategy has a double advantage. First, it guarantees that all the identical elements of a line are mapped onto the same logic elements of the CLB slices, with equal logic delays, and secondly, in the absence of applicable constraints for inter-cell routing and specific CLB pin selection, it leaves no choice for the automatic routing tool but to connect the elements of the line using the only possible routing. Since the placement and routing for all the line(s) elements are the same, equal logic and routing delays are obtained. Applying this strategy will also benefit the designer by relieving him/her from the tedious manual routing process to achieve equal routing delays. In large designs, manual routing in FPGAs is a major bottleneck requiring lots of work at the lowest level of logic, defeating the reduced prototyping time purpose in FPGA designs.

Figure 2 shows the logic mapping corresponding to the buffer and latch elements, for the delay matrix' vernier lines of [1]. The buffer is mapped onto a LUT driving a MUX element of the slice. Hence, the logic delay for the buffer is the sum of the LUT and multiplexer delays. In addition to supporting the delay matching with the corresponding latch delay of a vernier cell for the small differential resolution τ_X, the multiplexer helps keep the latch line transparent by driving the clock input of the latch elements by a logic-0 when the latch line is configured as a RO in calibration mode for on-chip delay estimation. On the other hand, the latch element of a cell has the flexibility of being mapped onto any one of the two memory cells of the CLB slice for uniformity of the latch delays, provided the desired differential resolution value can be achieved. In [1], the uniformity of cell delays were achieved using combinations of XILINX-specific design constraints such as BEL for basic element mapping, LOCK_PINS to select same LUT input pins for equal inter-cell routings, ROUTE for locking the optimized routing, and LOC for component placement.

Finally, the clock skew problems described previously make attempts to minimize them necessary. In [1], the possibility of using the global buffer lines for the clock lines are removed by design due to the structure of a delay cell as the clock signal passes through a buffer before reaching the clock input of a CLB slice. In such cases, skew minimization may be achieved by routing the clock signal from a global source buffer to destination points by means such as local buffering. That is to say, clock buffers need to be implemented and placed within the FPGA surface from source to destination in a topology similar to the conventional H-tree or any symmetric topology as to guarantee equal routing possibilities from source to all critical destination points. This process can be repeated for several clock buffer placement strategies with the help of applicable timing constraints such as MAXDELAY and MAXSKEW. The final optimized routed clock path can then be locked using directed routing constraint. When applied, the latter maintains the routing of the placed design intact even if the design is re-synthesized.

V. Conclusion

In this paper the cause and effects of several technology- and design-dependent bottlenecks such as delay mismatch, clock skew, manual place and route, jittery clock signal, and area constraints on delay-line-based time measurement application targeting FPGA devices were addressed. Also several optimization strategies were discussed. Experimental results were obtained by custom test structures related to a TDC on FPGA. Our conclusion is that, while application of the numerous constraints at various stages of the design flow, namely the map, place, and route processes, is vital in improving the performance of timely critical systems targeting FPGAs, the latter may not be the optimal choice as target technology for systems involving multiple delay lines.

References

[1] Amir M. Amiri, et *Al.*, "Low Dead Time, Multi-hit, FPGA-based Time-to-Dgital Converter"; proc. IEEE NEWCAS 2006, Gatineau (Canada), June 2006, pp. 29-32.

[2] kalisz, J et *Al*, "Field-Programmable-Gate-Array-Based Time-to-Digital Converter with 200-ps Resolution"; IEEE Trans. on Instrumentation and Measurement, 46-1, pp.51-55, 1997.

[3] Jinyuan Wu, et al.; "Firmware-only Implementation of Time-to-Digital Converter (TDC) in Field-Programmable Gate Array (FPGA)"; Nuclear Science Symp. Conference Record, 2003 IEEE Volume 1, 19-25 Oct. 2003 Page(s):177 - 181

[4] Antonio H. Chan, *et Al*, "A Jitter Characterization System Using a Component-Invariant Vernier Delay Line", IEEE Transactions On (VLSI) Systems, VOL. 12, NO. 1, Jan, 2004.

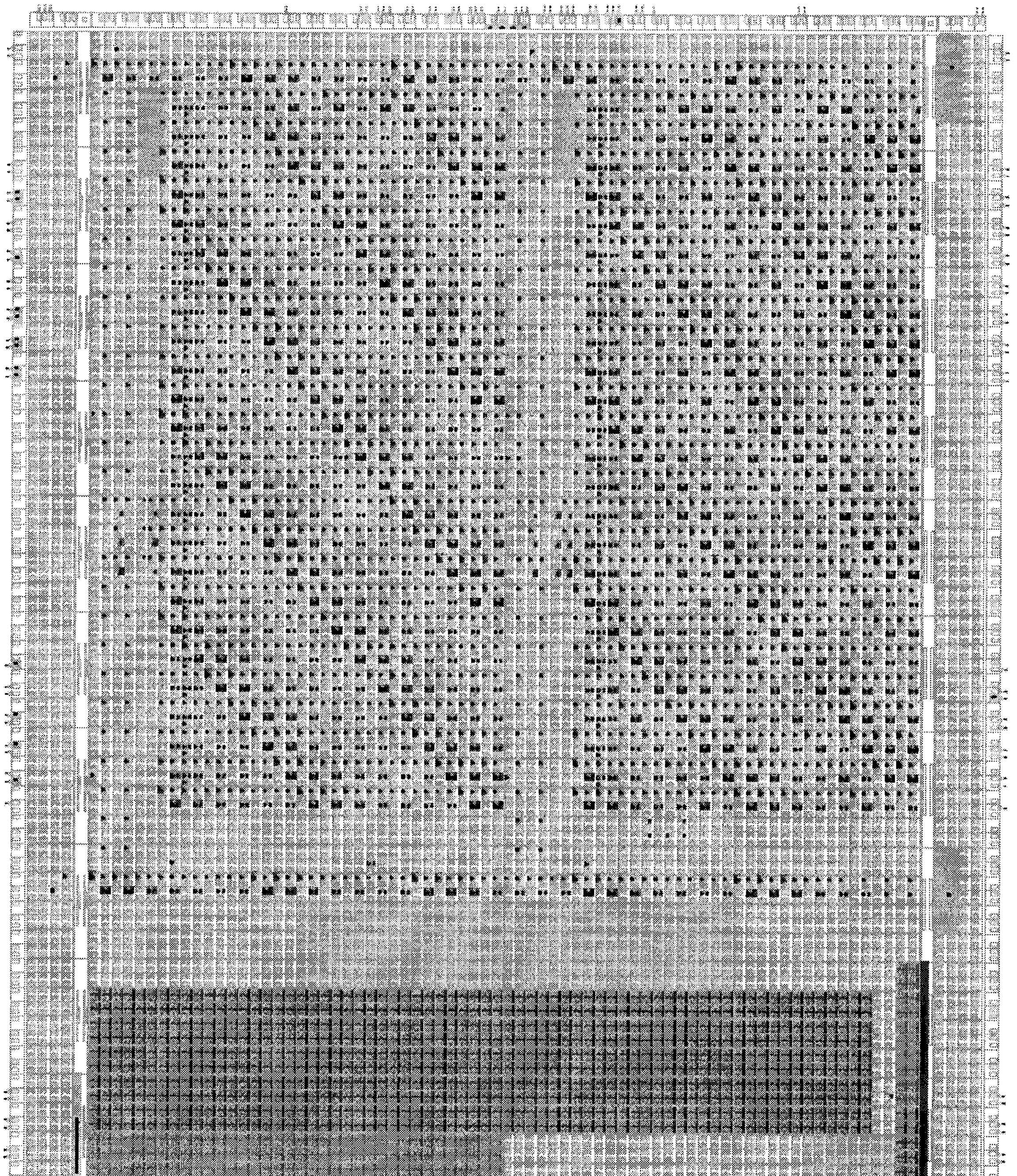
[5] Nassif, S. R.; "Modeling and analysis of manufacturing variations"; IEEE Conference on Custom Integrated Circuits, 6-9 May 2001 Page(s):223 – 228

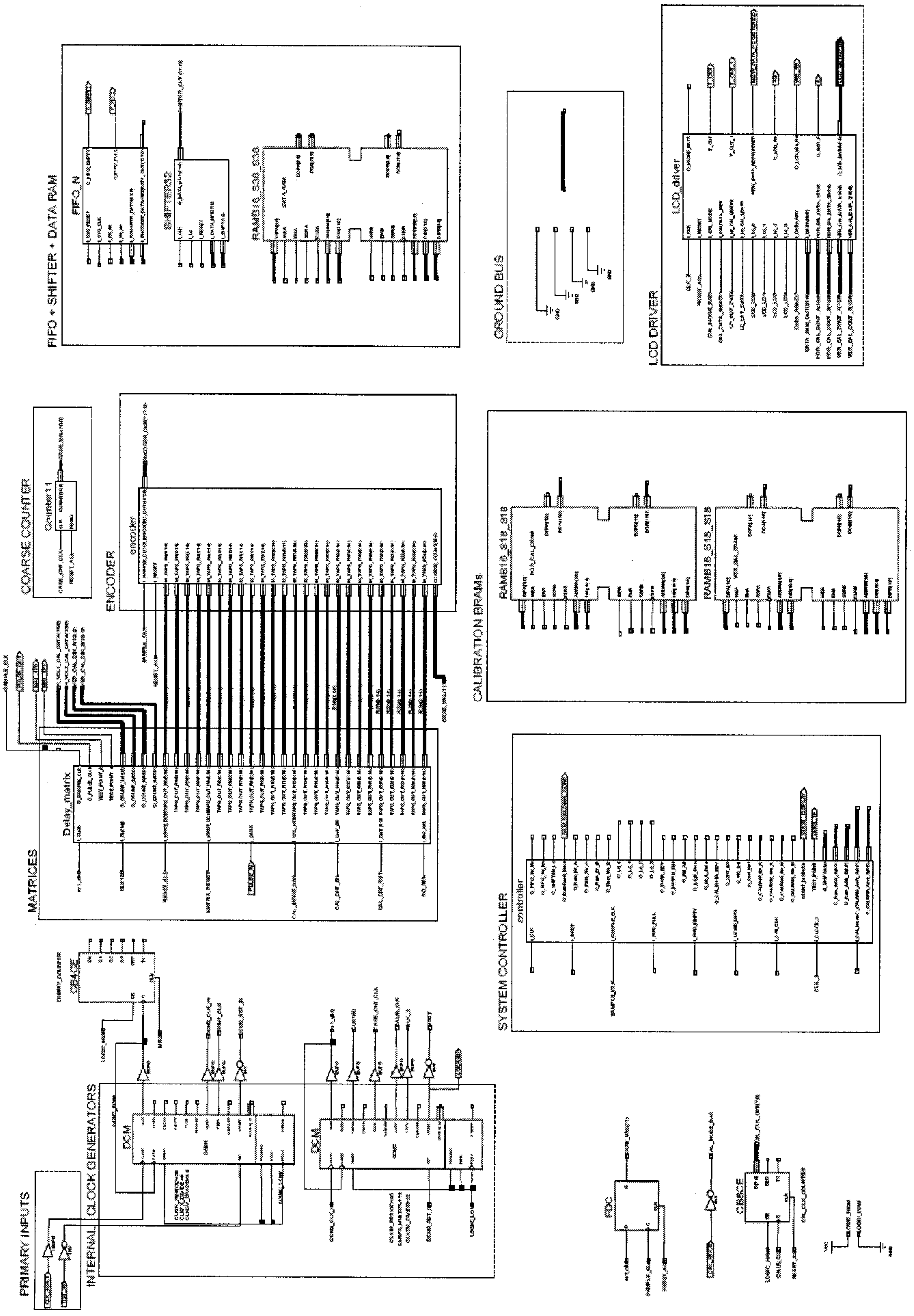
[6] B. Zhou, A. Khouas, "Measurement of delay mismatch due to process variations by means of modified ring oscillators", IEEE International Symposium on Circuits and Systems, 2005, Vol.5, pp. 5246-5249

APPENDIX-II

Schematic Of Floorplanned TDC Logic on Xc3s1000 -Spartan^{TM}-3 FPGA Surface

TDC Top-Level Schematic




PRIMARY INPUTS
INTERNAL CLOCK GENERATORS
DCM
DCM
CB4CE
MATRICES
Delay_matrix
COARSE COUNTER
Counter11
ENCODER
encoder
FIFO + SHIFTER + DATA RAM
FIFO_N
SHIFTER32
RAMB16_S36_S36
SYSTEM CONTROLLER
controller
CALIBRATION BRAMs
RAMB16_S18_S18
RAMB16_S18_S18
GROUND BUS
LCD DRIVER
LCD_driver
FDC
CB8CE

APPENDIX-III

VHDL SOURCE CODE

Only a select number of VHDL files are appended here. For complete source code, please refer to the submitted CD-ROM

REFERENCED VHDL SOURCE CODE

In this appendix, some VHDL files for TDC_SYSTEM is attached. It should be noted that not all VHDL files are appended due to the resulting large number of pages. For complete source code, including the TEST BENCH FILES the reader can refer to the zip file submitted with this document. The appended VHDL source codes in their respective order are:

- buf_element.vhd -- THE BUFFER ELEMENT
- DelayCell.vhd -- THE VERNIER DELAY CELL
- Delay_matrix.vhd-- THE INTERPOLATOR composed of 2 matrices + calibration logic
- Controller.vhd -- SYSTEM MAIN CONTROLLER UNIT
- Encoder.vhd -- ENCODER unit for encoding of tap lines (partial source code shown)
- LCD_DRIVER.vhd —LCD DRIVER for displaying of data display

For other VHDL files, such as: DC_matrix.vhd, Cal_Unit.vhd, and others, along with VHDL files used for the Test Bench Project, please refer to the zipped file submitted with this document.

♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦BUFFER ELEMENT♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦

```
-- **************************************************************** --
-- Module: Buffer implemented with a LUT driving a mux
-- AUTHOR: AMIRI AMIR MOHAMMAD
-- DATE:   27, OCTOBER, 2005
-- Modified: 05/05/2006
-- REMARKS: . The original 2-input buffer modified TO 3-input buffer
--     to support CALIBRATION configuration
--     ADDED LUT BEFORE MUX FOR BETTER CONTROL OF I/O PIN ASSIGNMENT (26/JAN/2007)
-- **************************************************************** --
--     LIBRARIES
-- **************************************************************** --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;
-- **************************************************************** --
--        ENTITY
-- **************************************************************** --
entity buf_element is
    Port ( I : IN   STD_LOGIC;
      M : IN    STD_LOGIC;
           O : OUT  STD_LOGIC );
end buf_element;
-- **************************************************************** --
--        ARCHITECTURE
-- **************************************************************** --
ARCHITECTURE STRUCTURAL OF buf_element IS
-- *** ATTRIBUTES
   ATTRIBUTE KEEP           : STRING;
   ATTRIBUTE KEEP_HIERARCHY   : STRING;
   ATTRIBUTE LOCK_PINS      : STRING;

-- *** SIGNALS
   SIGNAL BUF_IN, BUF_OUT, BUF_BETWEEN   : STD_LOGIC;

   ATTRIBUTE KEEP_HIERARCHY OF STRUCTURAL  : ARCHITECTURE IS "YES";
   ATTRIBUTE LOCK_PINS OF BUF_COMPONENT1   : LABEL IS "I0:A2";
   ATTRIBUTE KEEP OF BUF_IN, BUF_OUT     : SIGNAL IS "TRUE";
BEGIN
-- *** CSA
```

```
    BUF_IN <= I;
    O  <= BUF_OUT;

    -- I0 = '1' sends a '1' to G-input of LATCH ELEMENTS WHEN IN CAL MODE
    BUF_COMPONENT0: MUXF5  -- LUT USED AS A BUFFER
    port MAP(I0=>'0',  I1=>BUF_BETWEEN, S=>M, O=>BUF_OUT );

    BUF_COMPONENT1: LUT1      -- IMPORTANT WHEN LOCK_PINS ARE USED.
    GENERIC MAP(INIT => "10" )-- THIS IS BECAUSE, THE INPUTS OF THE MUX IS LOCAL
    PORT MAP(        -- AND CAN NOT BE SET WITH LOCK_PINS
       O  => BUF_BETWEEN,
       I0 => BUF_IN
    );
end STRUCTURAL;
```

♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦DELAY CELL♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦

```
-- ****************************************************************************************
-- VHDL code FOR A DELAY CELL ELEMENT TO BE USED IN DELAY MATRIX using STRUCTURAL VHDL
-- AUTHOR: AMIRI AMIR MOHAMMAD
-- DATE:   27, OCTOBER, 2005
-- REMARKS: NONE
-- ****************************************************************************************
-- *************************************************************** --
--       LIBRARIES
-- *************************************************************** --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;
-- *************************************************************** --
--       ENTITY
-- *************************************************************** --
entity DelayCell is
    Port (  D_IN   : in std_logic;
            CLK_IN    : in std_logic;
            MRST   : in std_logic;
      MODE   : IN STD_LOGIC;
            INTRST    : in std_logic;
      CLK_OUT    : OUT STD_LOGIC;
            Q_OUT  : out std_logic;
      INTRST_1: IN STD_LOGIC
        );
end DelayCell;
-- *************************************************************** --
--       ARCHITECTURE
-- *************************************************************** --
architecture STRUCTURAL of DelayCell is
-- *** ATTRIBUTES
   ATTRIBUTE BEL    : STRING;
   ATTRIBUTE KEEP   : STRING;
   ATTRIBUTE KEEP_HIERARCHY : STRING;

   ATTRIBUTE KEEP_HIERARCHY OF STRUCTURAL: ARCHITECTURE IS "YES";

-- *** COMPONENTS
   -- BUFFER
   COMPONENT BUF_ELEMENT
   PORT (
      I : IN STD_LOGIC; M : IN STD_LOGIC; O : OUT STD_LOGIC
   );
   END COMPONENT;

-- *** SIGNALS
   SIGNAL    LATCH_CLR        : STD_LOGIC;
   SIGNAL    Q_INT, MODE_BAR, BUF_OUT   : STD_LOGIC:= '0';

   ATTRIBUTE KEEP OF  BUF_OUT, Q_INT: SIGNAL IS "TRUE";

   ATTRIBUTE BEL  OF LATCH_COMPONENT: LABEL IS "FFY";
   SIGNAL GATE_ENABLE_VALUE : STD_LOGIC;
BEGIN
-- *** CSA
        --  OUTPUTS
   CLK_OUT   <= BUF_OUT   ;
   Q_OUT     <= Q_INT     ;

   -- CLK DISABLE LOGIC
   GATE_ENABLE_VALUE <= '1';

   LATCH_CLR <= INTRST OR MRST OR INTRST_1;      -- LATCH CLEAR FUNCTION
   MODE_BAR  <= NOT MODE;

   -- NEGATIVE LEVEL LATCH   for DATA
   LATCH_COMPONENT : LDCE_1
   GENERIC MAP('0')
   PORT MAP  (CLR=>LATCH_CLR, D=>D_IN, G=>BUF_OUT, GE=>GATE_ENABLE_VALUE ,Q=>Q_INT );

   -- BUFFER ELEMENT for CLK DELAY
   BUFFER_COMPONENT: BUF_ELEMENT PORT MAP ( I=>CLK_IN, M=> MODE_BAR, O=>BUF_OUT  );

end STRUCTURAL;
```

♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦ INTERPOLATOR (DELAY_MATRIX.VHD) ♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦

```
-- *******************************************************************************************
-- STRUCTURAL VHDL CODE FOR AN INTERPOLATING UNIT COMPOSED OF 2 MATRICES OF DELAY CELL 25 X 15
-- AUTHOR:    AMIRI AMIR M.
-- DATE:      01, JANUARY, 2006
-- REMARKS: NONE
--            . code cleanup on:22march2007
-- *******************************************************************************************
-- *************************************************************** --
--       LIBRARIES
-- *************************************************************** --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;
-- *************************************************************** --
--       ENTITY DECLARATION
-- *************************************************************** --
ENTITY Delay_matrix IS
    Port (
   I_CLK0    : IN std_logic;-- REFERENCE CLOCK-0
   I_CLK180  : IN STD_LOGIC;-- REFERENCE CLOCK-180

   I_MRST_DCM   : IN std_logic; -- MASTER RESET
   I_MRST_CONT  : IN std_logic; -- LOCAL RESET, TO CLEAR ACTIVE MAT. ENTRIES AT THE END OF MEAS.
PHASE
   I_DATA       : IN std_logic; -- PULSE DATA INPUT

   -- CAL. CONFIG. I/Os
   I_CAL_MODE   : IN STD_LOGIC;  -- FROM EXTERNAL SWITCH

   -- CONROLLER INTERFACE
   I_CNT_EN  : IN STD_LOGIC;
   I_CNT_RST : IN STD_LOGIC;
   I_RO_SEL     : IN STD_LOGIC;
   -- CALIBRATION DATA RAM INTERFACE
   O_COUNT_1    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
   O_COUNT_2    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
   O_COUNT_3    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
   O_COUNT_4    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);

      -- ENCODER INTERFACE
   O_SAMPLE_CLK   : out std_logic;

   TAPS_OUT_R0  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R1  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R2  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R3  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R4  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R5  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R6  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R7  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R8  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R9  : out std_logic_vector(0 TO 14);
   TAPS_OUT_R10 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R11 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R12 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R13 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R14 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R15 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R16 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R17 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R18 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R19 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R20 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R21 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R22 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R23 : out std_logic_vector(0 TO 14);
   TAPS_OUT_R24 : out std_logic_vector(0 TO 14);

   -- OUTPUTS
   O_PULSE_OUT  : OUT STD_LOGIC;
   TEST_POINT_0 : OUT STD_LOGIC;
   TEST_POINT_1 : OUT STD_LOGIC
  );
END Delay_matrix;

-- ************************************************************** --
--              ARCHITECTURE DECLARATION
-- ************************************************************** --
ARCHITECTURE Behavioral OF Delay_matrix IS
-- ATTRIBUTES
   ATTRIBUTE KEEP            : STRING;
   ATTRIBUTE MAXSKEW         : STRING;
   ATTRIBUTE MAXDELAY        : STRING;
   ATTRIBUTE KEEP_HIERARCHY  : STRING;
   ATTRIBUTE CLOCK_SIGNAL    : STRING;
   ATTRIBUTE BEL          : STRING;
   ATTRIBUTE LOCK_PINS       : STRING;

   ATTRIBUTE KEEP_HIERARCHY OF BEHAVIORAL : ARCHITECTURE IS "YES";
-- *** COMPONENT DECLARATION
   -- MATRIX COMPONENT
   COMPONENT dc_matrix
   PORT(
      D_PULSE   : IN std_logic;
      CLK_IN    : IN std_logic;
      MRST      : IN std_logic;
      CAL_MODE  : IN std_logic;

      SAMPLE_CLK : OUT std_logic;
      TAPS_R0   : OUT std_logic_vector(0 TO 14);
      TAPS_R1   : OUT std_logic_vector(0 TO 14);
      TAPS_R2   : OUT std_logic_vector(0 TO 14);
      TAPS_R3   : OUT std_logic_vector(0 TO 14);
      TAPS_R4   : OUT std_logic_vector(0 TO 14);
      TAPS_R5   : OUT std_logic_vector(0 TO 14);
      TAPS_R6   : OUT std_logic_vector(0 TO 14);
      TAPS_R7   : OUT std_logic_vector(0 TO 14);
      TAPS_R8   : OUT std_logic_vector(0 TO 14);
      TAPS_R9   : OUT std_logic_vector(0 TO 14);
      TAPS_R10  : OUT std_logic_vector(0 TO 14);
      TAPS_R11  : OUT std_logic_vector(0 TO 14);
      TAPS_R12  : OUT std_logic_vector(0 TO 14);
      TAPS_R13  : OUT std_logic_vector(0 TO 14);
      TAPS_R14  : OUT std_logic_vector(0 TO 14);
      TAPS_R15  : OUT std_logic_vector(0 TO 14);
      TAPS_R16  : OUT std_logic_vector(0 TO 14);
      TAPS_R17  : OUT std_logic_vector(0 TO 14);
      TAPS_R18  : OUT std_logic_vector(0 TO 14);
      TAPS_R19  : OUT std_logic_vector(0 TO 14);
      TAPS_R20  : OUT std_logic_vector(0 TO 14);
      TAPS_R21  : OUT std_logic_vector(0 TO 14);
      TAPS_R22  : out std_logic_vector(0 TO 14);
      TAPS_R23  : out std_logic_vector(0 TO 14);
      TAPS_R24  : out std_logic_vector(0 TO 14)  );
   END COMPONENT;

   -- CLK REGISTERS FOR EQUALIZING THE CLOCK PATH FOR BOTH MATRICES..
   COMPONENT CLOCK_REGISTER
   PORT ( CLK_IN: IN STD_LOGIC;
          CLK_OUT : OUT STD_LOGIC
          );
   END COMPONENT;

   -- 10-BIT COUNTER
   COMPONENT counter10
   PORT(
      CLK       : IN std_logic;
      CNT_EN    : IN std_logic;
      RESET  : IN std_logic;
      COUNT  : OUT std_logic_vector(9 downto 0)  );
   END COMPONENT;

   COMPONENT cal_unit
   PORT(
      CAL_EN    : IN std_logic;
      RO_SEL    : IN std_logic;
      RESET  : IN std_logic;
      HOR_RO_CLK1 : OUT std_logic;
      HOR_RO_CLK2 : OUT std_logic;
      VER_RO_CLK1 : OUT std_logic;
      VER_RO_CLK2 : OUT std_logic
      );
   END COMPONENT;

-- SIGNALS
   SIGNAL PULSE_CLEAR      : STD_LOGIC := '0';
   SIGNAL DATA_PULSE       : STD_LOGIC := '0';
   SIGNAL BUF_DATA_PULSE   : STD_LOGIC := '0';
   SIGNAL RO_CLK_1         : STD_LOGIC;
   SIGNAL RO_CLK_2         : STD_LOGIC;
   SIGNAL RO_CLK_3         : STD_LOGIC;
   SIGNAL RO_CLK_4         : STD_LOGIC;

   -- RESET SIGNAL
   SIGNAL RESET_M1  : STD_LOGIC := '0';
   SIGNAL R1     : STD_LOGIC := '0';
   SIGNAL RESET_M2  : STD_LOGIC := '0';
   SIGNAL R2       : STD_LOGIC := '0';

   -- SAMPLE CLK SIGNAL FOR SAMPLING
   SIGNAL SAMPLE_CLK_M1   : STD_LOGIC;
   SIGNAL SAMPLE_CLK_M2   : STD_LOGIC;

   SIGNAL
      CLK0_BUFFERED_0, CLK0_BUFFERED_1, CLK0_BUFFERED_2, CLK0_LASTBUFFER, CLK_R_0, CLK_R_1,
      CLK180_BUFFERED_0, CLK180_BUFFERED_1, CLK180_BUFFERED_2, CLK180_LASTBUFFER, CLK_R_180
   : STD_LOGIC;

   -- MUX CONTROL SIGNALS
   TYPE OUTPUT_MUX IS ARRAY (0 TO 24) OF STD_LOGIC_VECTOR(0 TO 14);
   SIGNAL M1_2_MUX, M2_2_MUX : OUTPUT_MUX :=
          ( (others => '0'),(others => '0'), (others => '0'), (others => '0'), (others => '0')
          , (others => '0'),(others => '0'), (others => '0'), (others => '0'), (others => '0')
          , (others => '0'),(others => '0'), (others => '0'), (others => '0'), (others => '0')
          , (others => '0'),(others => '0'), (others => '0'), (others => '0'), (others => '0')
          , (others => '0'),(others => '0'), (others => '0'), (others => '0'), (others => '0') );

   -- ATTRIBUTE VALUES
   ATTRIBUTE MAXSKEW  OF CLK_R_0, CLK_R_180 : SIGNAL IS "500 ps";
   ATTRIBUTE MAXDELAY OF CLK_R_0, CLK_R_180   :  SIGNAL IS "2000 ps";

   ATTRIBUTE MAXSKEW  OF DATA_PULSE                : SIGNAL IS "100PS";
   ATTRIBUTE MAXDELAY OF SAMPLE_CLK_M1, SAMPLE_CLK_M2  : SIGNAL IS "2000 PS";
   ATTRIBUTE MAXSKEW  OF O_SAMPLE_CLK              : SIGNAL IS "1000 PS";
   ATTRIBUTE KEEP     OF O_SAMPLE_CLK, data_pulse     : SIGNAL IS "TRUE";

   -- THIS SIGNAL IS USED FOR MUX SELECTION..SO SKEW MAY BE LARGE DEPENDING ON LOCATION OF MUXES
   ATTRIBUTE MAXSKEW OF CLK_R_1 : SIGNAL IS "1500 PS";

   ATTRIBUTE MAXDELAY OF CLK0_BUFFERED_0, CLK180_BUFFERED_0 : SIGNAL IS "800 ps";
   ATTRIBUTE MAXDELAY OF CLK0_BUFFERED_1, CLK180_BUFFERED_1 : SIGNAL IS "1500 ps";
   ATTRIBUTE MAXDELAY OF CLK0_BUFFERED_2, CLK180_BUFFERED_2   :  SIGNAL IS "1500 ps";

   ATTRIBUTE BEL OF CONSTANT_PULSE_GENERATOR: LABEL IS "FFY"; -- UPPER FF OF THE CLB
   ATTRIBUTE BEL OF BUF0              : LABEL IS "F";  -- LOWER LUT OF THE CLB
   ATTRIBUTE MAXDELAY OF BUF_DATA_PULSE      : SIGNAL IS "1 ns";

   ATTRIBUTE BEL OF BUFFER_BG0  : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_0    : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_1    : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_2    : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_LAST1 : LABEL IS "F";

   ATTRIBUTE BEL OF BUFFER_BG180 : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_3    : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_4    : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_5    : LABEL IS "G";
   ATTRIBUTE BEL OF BUFFER_LAST2 : LABEL IS "F";

   ATTRIBUTE
      LOCK_PINS OF BUFFER_5, BUFFER_2 , BUFFER_0,  BUFFER_3, BUFFER_LAST2, BUFFER_LAST1
   : LABEL IS "I0:A1";

   ATTRIBUTE KEEP  OF
      CLK0_BUFFERED_0, CLK0_BUFFERED_1, CLK0_BUFFERED_2, CLK0_LASTBUFFER, CLK_R_0,
      CLK180_BUFFERED_0, CLK180_BUFFERED_1, CLK180_BUFFERED_2, CLK180_LASTBUFFER, CLK_R_180
   : SIGNAL IS "TRUE";
BEGIN

-- ***CONCURRENT SIGNAL ASSIGNMENTS (CSA)*** --
   -- TEST POINT ASSIGNMENT
   TEST_POINT_0 <= R1;
   TEST_POINT_1 <= R2;
   -- CSA FOR DETERMINING THE RESET SIGNALS FOR EACH MATRIX
   RESET_M1 <= I_MRST_DCM OR R1;
   RESET_M2 <= I_MRST_DCM OR R2;

   -- OUTPUT PULSE for TESTING PURPOSES
   O_PULSE_OUT  <= DATA_PULSE;

--  BUFFERS TO MINIMIZE CLOCK SKEW ALONG CLOCK PATH FROM BUFG TO AN IDEAL LOC CLOSE TO MATRICES
   BUFFER_BG0: LUT1-- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>I_CLK0        , O=>CLK0_BUFFERED_0  );
   BUFFER_0: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>CLK0_BUFFERED_0   , O=>CLK0_BUFFERED_1  );
   BUFFER_1: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>CLK0_BUFFERED_1   , O=>CLK0_BUFFERED_2  );
   BUFFER_2: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>CLK0_BUFFERED_2   , O=>CLK0_LASTBUFFER  );
   BUFFER_LAST1: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>CLK0_LASTBUFFER   , O=>CLK_R_0  );

   BUFFER_BG180: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>I_CLK180          , O=>CLK180_BUFFERED_0  );
   BUFFER_3: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>CLK180_BUFFERED_0 , O=>clk180_BUFFERED_1  );
   BUFFER_4: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>clk180_BUFFERED_1 , O=>clk180_BUFFERED_2  );
   BUFFER_5: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>clk180_BUFFERED_2 , O=>CLK180_LASTBUFFER  );
   BUFFER_LAST2: LUT1  -- LUT USED AS A BUFFER
   GENERIC MAP (INIT => "10")PORT MAP ( I0=>CLK180_LASTBUFFER , O=>CLK_R_180  );

   -- FOR MATRIX SELECTION BASED ON I_CLK0 (MEANT TO REDUCE FAN OUT FOR I_CLK0)
   CLK_REG_SELECT : CLOCK_REGISTER PORT MAP(CLK_IN=>I_CLK0, CLK_OUT=>CLK_R_1);

   --CALIBRATION COUNTERS, INCREMENTED BY THE RO-GENERATED CLK. THESE ARE SHARED BY ALL 8 RO
   --COUNTER 1 is USED for first HORZ. VDL
   CAL_COUNTER_1: counter10 PORT MAP
   (CLK => RO_CLK_1,CNT_EN => I_CNT_EN, RESET => I_CNT_RST, COUNT =>O_COUNT_1(9 DOWNTO 0) );

   --COUNTER 2 is USED for 2ND HORZ. VDL
   CAL_COUNTER_2: counter10 PORT MAP
   (CLK => RO_CLK_2,CNT_EN => I_CNT_EN, RESET => I_CNT_RST, COUNT =>O_COUNT_2(9 DOWNTO 0) );

   --COUNTER 3 is USED for VERTICAL REF-RO1 AND RO-DUT-1
   CAL_COUNTER_3: counter10 PORT MAP
   (CLK => RO_CLK_3,CNT_EN => I_CNT_EN, RESET => I_CNT_RST, COUNT =>O_COUNT_3(9 DOWNTO 0) );

   --COUNTER 4 is USED for VERTICAL REF-RO2 AND RO-DUT-2
   CAL_COUNTER_4: counter10 PORT MAP
   (CLK => RO_CLK_4,CNT_EN => I_CNT_EN, RESET => I_CNT_RST, COUNT =>O_COUNT_4(9 DOWNTO 0) );

   --ASSIGNING ZEROS TO UPPER 6 BITS OF COUNTER OUTPUTS
   O_COUNT_1(15 DOWNTO 10) <= (OTHERS => '0');
   O_COUNT_2(15 DOWNTO 10) <= (OTHERS => '0');
   O_COUNT_3(15 DOWNTO 10) <= (OTHERS => '0');
   O_COUNT_4(15 DOWNTO 10) <= (OTHERS => '0');

   -- CALIBRATION UNIT
   CALIBRATION_LINES: cal_unit
   PORT MAP(
      CAL_EN      => I_CAL_MODE,
      RO_SEL      => I_RO_SEL,
      RESET       => I_MRST_DCM,
      HOR_RO_CLK1 => RO_CLK_1,
      HOR_RO_CLK2 => RO_CLK_2,
      VER_RO_CLK1 => RO_CLK_3,
      VER_RO_CLK2 => RO_CLK_4
   );
   -- ***  FIXED-WIDTH PULSE GENERATION  *** --
   CONSTANT_PULSE_GENERATOR: FDCP
   GENERIC MAP('0') PORT MAP (PRE=>'0', CLR => PULSE_CLEAR, C=> I_DATA, D=>'1', Q=>DATA_PULSE );
   -- LUT USED AS A BUFFER
   BUF0 : LUT1
   generic map (INIT => "10")port map (O => BUF_DATA_PULSE,   I0 => DATA_PULSE  );

PULSE_CLEAR <= BUF_DATA_PULSE;

-- MATRIX 1 SENSITIVE TO I_CLK0 ( NO SHIFTED CLK)
   M1:dc_matrix
   PORT MAP(
      D_PULSE      => DATA_PULSE,
      CLK_IN       => CLK_R_0    ,
      MRST     => RESET_M1   ,
      CAL_MODE => I_CAL_MODE,

      SAMPLE_CLK   => SAMPLE_CLK_M1,
      TAPS_R0      =>  M1_2_MUX(0),
      TAPS_R1      =>  M1_2_MUX(1),
      TAPS_R2      =>  M1_2_MUX(2),
      TAPS_R3      =>  M1_2_MUX(3),
      TAPS_R4      =>  M1_2_MUX(4),
      TAPS_R5      =>  M1_2_MUX(5),
      TAPS_R6      =>  M1_2_MUX(6),
      TAPS_R7      =>  M1_2_MUX(7),
      TAPS_R8      => M1_2_MUX(8),
      TAPS_R9  =>  M1_2_MUX(9),
      TAPS_R10 =>  M1_2_MUX(10),
      TAPS_R11 =>  M1_2_MUX(11),
      TAPS_R12 =>  M1_2_MUX(12),
      TAPS_R13 =>  M1_2_MUX(13),
      TAPS_R14 => M1_2_MUX(14),
      TAPS_R15 => M1_2_MUX(15),
      TAPS_R16 => M1_2_MUX(16),
      TAPS_R17 => M1_2_MUX(17),
      TAPS_R18 => M1_2_MUX(18),
      TAPS_R19 => M1_2_MUX(19),
      TAPS_R20 => M1_2_MUX(20),
      TAPS_R21 => M1_2_MUX(21),
      TAPS_R22 => M1_2_MUX(22),
      TAPS_R23 => M1_2_MUX(23),
      TAPS_R24     => M1_2_MUX(24)
   );

   -- MATRIX 2  SENSITIVE TO CLK0_180
   M2:dc_matrix
   PORT MAP(
      D_PULSE  => DATA_PULSE,
      CLK_IN   => CLK_R_180 ,
      MRST   => RESET_M2   ,
      CAL_MODE=> I_CAL_MODE,

      SAMPLE_CLK   => SAMPLE_CLK_M2,
      TAPS_R0      =>  M2_2_MUX(0),
      TAPS_R1      =>  M2_2_MUX(1),
      TAPS_R2      =>  M2_2_MUX(2),
      TAPS_R3      =>  M2_2_MUX(3),
      TAPS_R4      =>  M2_2_MUX(4),
      TAPS_R5      =>  M2_2_MUX(5),
      TAPS_R6      =>  M2_2_MUX(6),
      TAPS_R7      =>  M2_2_MUX(7),
      TAPS_R8      => M2_2_MUX(8),
      TAPS_R9  =>  M2_2_MUX(9),
      TAPS_R10 =>  M2_2_MUX(10),
      TAPS_R11 =>  M2_2_MUX(11),
      TAPS_R12 =>  M2_2_MUX(12),
      TAPS_R13      =>  M2_2_MUX(13),
      TAPS_R14  => M2_2_MUX(14),
      TAPS_R15  => M2_2_MUX(15),
      TAPS_R16  => M2_2_MUX(16),
      TAPS_R17  => M2_2_MUX(17),
      TAPS_R18  => M2_2_MUX(18),
      TAPS_R19  => M2_2_MUX(19),
      TAPS_R20  => M2_2_MUX(20),
      TAPS_R21  => M2_2_MUX(21),
      TAPS_R22  => M2_2_MUX(22),
      TAPS_R23  => M2_2_MUX(23),
      TAPS_R24  => M2_2_MUX(24)
   );

-- *** PROCESSES *** --
   -- OUTPUT MUX FOR CURRENT/ACTIVE MATRIX, BASED ON "I_CLK0"
   OUTPUT_BUS:
   PROCESS( CLK_R_1, M1_2_MUX, M2_2_MUX )
      BEGIN
         CASE CLK_R_1 IS
            WHEN '0' =>  TAPS_OUT_R0  <= M2_2_MUX(0);
                      TAPS_OUT_R1     <= M2_2_MUX(1);
                      TAPS_OUT_R2     <= M2_2_MUX(2);
                      TAPS_OUT_R3     <= M2_2_MUX(3);
                      TAPS_OUT_R4     <= M2_2_MUX(4);
                      TAPS_OUT_R5     <= M2_2_MUX(5);
                      TAPS_OUT_R6     <= M2_2_MUX(6);
                      TAPS_OUT_R7     <= M2_2_MUX(7);
                      TAPS_OUT_R8     <= M2_2_MUX(8);
                      TAPS_OUT_R9     <= M2_2_MUX(9);
                      TAPS_OUT_R10    <= M2_2_MUX(10);
                      TAPS_OUT_R11 <= M2_2_MUX(11);
                      TAPS_OUT_R12 <= M2_2_MUX(12);
                      TAPS_OUT_R13 <= M2_2_MUX(13);
                      TAPS_OUT_R14 <= M2_2_MUX(14);
                      TAPS_OUT_R15 <= M2_2_MUX(15);
                      TAPS_OUT_R16 <= M2_2_MUX(16);
                      TAPS_OUT_R17 <= M2_2_MUX(17);
                      TAPS_OUT_R18 <= M2_2_MUX(18);
                      TAPS_OUT_R19 <= M2_2_MUX(19);
                      TAPS_OUT_R20 <= M2_2_MUX(20);
                      TAPS_OUT_R21 <= M2_2_MUX(21);
                      TAPS_OUT_R22 <= M2_2_MUX(22);
                      TAPS_OUT_R23 <= M2_2_MUX(23);
                      TAPS_OUT_R24 <= M2_2_MUX(24);

            WHEN OTHERS =>  TAPS_OUT_R0     <= M1_2_MUX(0);
                        TAPS_OUT_R1     <= M1_2_MUX(1);
                        TAPS_OUT_R2     <= M1_2_MUX(2);
                        TAPS_OUT_R3     <= M1_2_MUX(3);
                        TAPS_OUT_R4     <= M1_2_MUX(4);
                        TAPS_OUT_R5     <= M1_2_MUX(5);
                        TAPS_OUT_R6     <= M1_2_MUX(6);
                        TAPS_OUT_R7     <= M1_2_MUX(7);
                        TAPS_OUT_R8     <= M1_2_MUX(8);
                        TAPS_OUT_R9     <= M1_2_MUX(9);
                        TAPS_OUT_R10    <= M1_2_MUX(10);
                        TAPS_OUT_R11 <= M1_2_MUX(11);
                        TAPS_OUT_R12 <= M1_2_MUX(12);
                        TAPS_OUT_R13 <= M1_2_MUX(13);
                        TAPS_OUT_R14 <= M1_2_MUX(14);
                        TAPS_OUT_R15 <= M1_2_MUX(15);
                        TAPS_OUT_R16 <= M1_2_MUX(16);
                        TAPS_OUT_R17 <= M1_2_MUX(17);
                        TAPS_OUT_R18 <= M1_2_MUX(18);
                        TAPS_OUT_R19 <= M1_2_MUX(19);
                        TAPS_OUT_R20 <= M1_2_MUX(20);
```

```
                          TAPS_OUT_R21 <= M1_2_MUX(21);
                          TAPS_OUT_R22 <= M1_2_MUX(22);
                          TAPS_OUT_R23 <= M1_2_MUX(23);
                          TAPS_OUT_R24 <= M1_2_MUX(24);
         END CASE;
   END PROCESS;

   -- MUXING THE SAMPLE CLK BASED ON ACTIVE MATRIX (M1 OR M2)
   SAMPLE_CLK_MUXING:
   PROCESS(CLK_R_1, SAMPLE_CLK_M1, SAMPLE_CLK_M2 )
      BEGIN
         IF CLK_R_1 = '1' THEN  O_SAMPLE_CLK <= SAMPLE_CLK_M1;
         ELSE                   O_SAMPLE_CLK <= SAMPLE_CLK_M2;
         END IF;
   END PROCESS;

   -- ROUTING THE CONTROLLER-BASED RESET TO THE APPROPRIATE ACTIVE MATRIX
   RESET_DEMUX:
   PROCESS(SAMPLE_CLK_M1, I_MRST_CONT)
      BEGIN
         CASE SAMPLE_CLK_M1 IS
            -- WHEN I_CLK0 IS LOW, M1 IS ACTIVE, AND M2 IS IN PROCESS AFTER WHICH NEEDS TO BE
RESET
            WHEN '0'  => R2 <= I_MRST_CONT;
                         R1 <= '0';
            WHEN OTHERS =>  R1 <= I_MRST_CONT;
                            R2 <= '0';
            END CASE;
   END PROCESS;
END Behavioral;
```

♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦ CONTROL UNIT♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦

```
-- ******************************************************************************************
-- MODULE: CONTROL UNIT                                                           *
-- AUTHOR: AMIRI AMIR MOHAMMAD                                                    *
-- DATE:   27 OCTOBER 2005                                                        *
-- REMARKS:                                                                   *
--          . MODIFIED 06/07 MAY, 2006 TO INCORPORATE CALIBRATION CONTROLL          *
--          . LAST UPDATE: 21-MARCH-2007 (CODE CLEANUP)                             *
-- ******************************************************************************************
-- ************************************************************** --
--                           LIBRARIES
-- ************************************************************** --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;
-- ************************************************************** --
--                           ENTITY DECLARATION
-- ************************************************************** --
ENTITY Controller IS
   port (
      -- SYSTEM CLK AND RESET
      I_CLK         : IN STD_LOGIC; -- MASTER CLOCK ( ~5 X MATRIX CLOCK )
      I_MRST        : IN STD_LOGIC;  -- (FROM DCM LOCKED)
      I_SAMPLE_CLK     : IN STD_LOGIC;  -- FROM ENCODER; ASSERTED WHEN DATA IS SAMPLED

      -- FIFO INTERFACE
      I_FIFO_FULL   : IN STD_LOGIC;  -- FROM FIFO
      I_FIFO_EMPTY  : IN STD_LOGIC;  -- FROM FIFO
      O_FIFO_Rd_En  : OUT STD_LOGIC;-- TO FIFO
      O_FIFO_Wr_En  : OUT STD_LOGIC;-- TO FIFO

      -- SHIFTER INTERFACE
      O_SHIFT       : OUT STD_LOGIC_VECTOR(1 DOWNTO 0); -- TO SHIFTER
      O_SHIFTER_Ld : OUT STD_LOGIC;                -- TO SHIFTER

      -- RAM INTERFACE ---( NORMAL DATA )-----------
      O_RamRead_Done  : OUT STD_LOGIC;

      -- PORT A
      O_Ram_Addr_A    : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
      O_Ram_Addr_B    : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
      O_Ram_En_A      : OUT STD_LOGIC;
      O_Ram_We_A      : OUT STD_LOGIC;

      --_PORT B
      O_Ram_En_B      : OUT STD_LOGIC;
      O_Ram_We_B      : OUT STD_LOGIC;

      -- LCD INTERFACE
      I_MORE_DATA     : IN STD_LOGIC;
      O_Ld_0       : OUT STD_LOGIC;
      O_Ld_1       : OUT STD_LOGIC;
      O_Ld_2       : OUT STD_LOGIC;
      O_Ld_3       : OUT STD_LOGIC;
      O_DATA_RDY      : OUT STD_LOGIC;

      -- GENERAL OUTPUTS
      O_MATRIX_Rst : OUT STD_LOGIC; -- TO MATRICES
      O_Rst_All    : OUT STD_LOGIC;

      -- CALIBRATION I/O ---------------------------
      I_CAL_CLK    : IN STD_LOGIC;  -- USED for START_CAL and LINE_DONE conditions
      I_CLOCK_2    : IN STD_LOGIC;  -- THE SLOWER CLOCK FOR LCD, RAM, CONTROLLER INTERFACE
      I_CAL_Mode      : IN STD_LOGIC;  -- EXTERNAL MODE INPUT

      -- LCD INTERFACE ----- ( CALIBRATION DATA )------
      O_Ld_B_Data     : OUT STD_LOGIC;-- FOR LAODING OF FIRST 4 CAL. COUNTER VALUES FROM RAM TO
LCD CONT. REGs
      O_Ld_L_Data     : OUT STD_LOGIC;-- FOR LOADING OF SECOND 4 CAL. COUNTER VALUES FROM RAM TO
LCD CONT. REGs
      O_CAL_DATA_RDY  : OUT STD_LOGIC;-- HANDSHAKE signal used to interact with LCD CONTROLLER

      -- TO DELAY MATRIX FOR CAL_UNIT
      O_CNT_En     : OUT STD_LOGIC;
      O_RO_Sel     : OUT STD_LOGIC;
      O_CNT_Rst : OUT STD_LOGIC;

      -- RAM INTERFACE ----- ( CALIBRATION DATA )------
      -- PORT A
      O_CALRAM_Addr_A : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      O_CALRAM_En_A   : OUT STD_LOGIC;
      O_CALRAM_We_A: OUT STD_LOGIC;

      -- PORT B
      O_CALRAM_Addr_B : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
      O_CALRAM_En_B   : OUT STD_LOGIC;
      O_CALRAM_We_B: OUT STD_LOGIC;

      -- GENERAL TEST POINTS FOR DISPLAY
      START_DISPLAY: OUT STD_LOGIC;
      TEST_POINT      : OUT STD_LOGIC
    );
END Controller;
-- ************************************************************* --
--                            ARCHITECTURE
-- ************************************************************* --
ARCHITECTURE Behavioral of controller is

   ATTRIBUTE KEEP_HIERARCHY : STRING;
   ATTRIBUTE KEEP_HIERARCHY OF BEHAVIORAL : ARCHITECTURE IS "TRUE";
```

```
-- SIGNALS
   SIGNAL START_STORE        : STD_LOGIC;
   SIGNAL RST_M, RST_BUF : STD_LOGIC;
   SIGNAL ACTIVATE       : STD_LOGIC;
   SIGNAL RST_ACTIVATE   : STD_LOGIC;
   SIGNAL START_DISP         : STD_LOGIC;
   SIGNAL RAM_ADDR_RST       : STD_LOGIC;
   SIGNAL RESET_A            : STD_LOGIC;
   SIGNAL RESET_B            : STD_LOGIC;
   SIGNAL ADDR_INC_A         : STD_LOGIC;
   SIGNAL ADDR_INC_B     : STD_LOGIC;

   SIGNAL RAM_ADDR_REG_A     : STD_LOGIC_VECTOR(8 DOWNTO 0);
   SIGNAL RAM_ADDR_REG_B        : STD_LOGIC_VECTOR(8 DOWNTO 0);
   SIGNAL WAIT_FOR_MORE_DATA : STD_LOGIC;

   SIGNAL RAM_W_CNT_INC      : STD_LOGIC;
   SIGNAL RAM_R_CNT_INC      : STD_LOGIC;
   SIGNAL RAM_R_CNT_RST      : STD_LOGIC;

   SIGNAL
      RAM_W_COUNTER, RAM_R_COUNTER
   : INTEGER RANGE 0 TO 128 := 0;   -- ASSUMING FIFO WILL BE 128 ENTRY DEEP

   SIGNAL TEST_SIGNAL  : STD_LOGIC;
   SIGNAL TEST_SIGNAL_1 : STD_LOGIC;

-- FINITE STATE MACHINE
   TYPE     DETECT_FSM IS (T_D_INIT, TMP, T_D_WAIT);
   SIGNAL   CURR_D_STATE, NEXT_D_STATE : DETECT_FSM := T_D_INIT;

   -- PROCESS_FSM         T_FIFO_WE_ST,T_FIFO_RE_ST ,
   TYPE PROCESS_FSM IS (
      T0_ST        , T_WAIT_ST     , T_WRITEFIFO_ST, T_READFIFO_ST ,
      T_SHIFT0_ST  , T_LDSHIFTER_ST, T_DUMMY1_ST    , T_RESET_RAM_ADDR_ST,
      T_DUMMY0_ST  , T_SHIFT1_ST    , T_SHIFT2_ST    , T_SHIFT3_ST,
      T_F2RAM_COMP_ST
   );
   SIGNAL CURR_ST_STATE, NEXT_ST_STATE : PROCESS_FSM := T0_ST;

   -- DISPLAY_FSM
   TYPE DISPLAY_FSM IS (
      T0_DISP      , T_WAIT_DISP , T_READ0_DISP  , T_READ1_DISP   ,
      T_READ2_DISP, T_READ3_DISP, T_TMP0_DISP, T_TMP1_DISP, T_DISPLAY_DISP
   );
   SIGNAL CURR_DISP_STATE, NEXT_DISP_STATE : DISPLAY_FSM := T0_DISP;

-- CALIBRATION-related SIGNALS
   -- CAL_CONFIG_FSM
   TYPE   CAL_CONFIG_FSM IS (
      T_INIT       , T_CAL_CONFIG1, T_CAL_OP1       , T_CAL_CNT_STORE_1,
      T_CAL_CONFIG2, T_CAL_OP2      , T_CAL_CNT_STORE_2 , T_CAL_NEXT_RND  ,
      T_CAL_START_DISP
   );
   SIGNAL CURR_C_STATE, NEXT_C_STATE : CAL_CONFIG_FSM := T_INIT;

   -- CAL_STORE_FSM
   TYPE CAL_STORE_FSM IS (
      T_CAL_ST_S0     , T_CAL_ST_WAIT_1   , T_CAL_ST_WRITE_1, T_CAL_ST_WAIT_2,
      T_CAL_ST_WRITE_2 , T_CAL_ST_DONE  , T_CAL_ST_END   , T_WAIT_FOR_MORE,
      T_CAL_DISP       , T_CAL_RAM_READ , T_TMP_CAL_DISP , T_CNT_DATA_LOAD_1,
      T_NEXT_ADDRESS   , T_CNT_DATA_LOAD_2
   );
   SIGNAL CURR_CAL_ST_STATE, NEXT_CAL_ST_STATE :  CAL_STORE_FSM := T_CAL_ST_S0;

   SIGNAL START_CAL       : STD_LOGIC;
   SIGNAL RST_START_CAL      : STD_LOGIC;
   SIGNAL LINE_DONE       : STD_LOGIC;
   SIGNAL RST_LINE_DONE   : STD_LOGIC;
   SIGNAL RST_RND_CNT     : STD_LOGIC;
   SIGNAL RND_CNT_INC     : STD_LOGIC;
   SIGNAL RESET_CAL_RAM_ADDR : STD_LOGIC;
   SIGNAL CAL_RAM_ADDR_RST      : STD_LOGIC;
   SIGNAL CAL_RAM_ADDR_INC      : STD_LOGIC;

   SIGNAL CAL_ROUND          : INTEGER RANGE 0 TO 30 := 0;
   SIGNAL CAL_RAM_ADDR_REG_A : STD_LOGIC_VECTOR(9 DOWNTO 0);
   SIGNAL CAL_RAM_ADDR_REG_B : STD_LOGIC_VECTOR(9 DOWNTO 0);

   SIGNAL CNT_DATA_1_READY   : STD_LOGIC;
   SIGNAL CNT_DATA_2_READY   : STD_LOGIC;
   SIGNAL CAL_DATA_STORED : STD_LOGIC;
   SIGNAL MORE_2_STORE    : STD_LOGIC;
   SIGNAL CONTINUE_FLAG      : STD_LOGIC;
-- ************************************************************ --
--                           ARCHITECTURE BODY
-- ************************************************************ --
BEGIN -- ARCHITECTURE BODY

-- **************************** NORMAL OPERATION CONTROL LOGIC ****************************
-- *** CONCURRENT SIGNAL ASSIGNMENTS (CSA)*** --
   TEST_POINT      <= TEST_SIGNAL;
   START_DISPLAY   <= TEST_SIGNAL_1;
   O_Ram_Addr_A    <= RAM_ADDR_REG_A;
   O_Ram_Addr_B    <= RAM_ADDR_REG_B;
   O_Rst_All       <= I_MRST;
   O_MATRIX_Rst    <= RST_M  ;
   RAM_ADDR_RST <= RESET_A OR RESET_B OR I_MRST;

-- ****** PORT MAPPED LOGIC ***** --
   -- FIXED-WIDTH MATRIX RESET SIGNAL
   CONSTANT_RESET: FDCP
   GENERIC MAP('0') PORT MAP (PRE=>'0', CLR => RST_BUF, C=> ACTIVATE, D=>'1',  Q=>RST_M );
   RST_M_BUFFER : LUT1 -- LUT USED AS A BUFFER
   generic map (INIT => "10") port map ( O => RST_BUF,   I0 => RST_M  );

-- *** CALIBRATION PROCESSES *** --
   --PROCESS USED TO TRIGGER STATE TRANSITION AFTER RISING EDGE OF I_SAMPLE_CLK
   DATA_LATCH_PROCESS:
   PROCESS(I_SAMPLE_CLK, I_MRST,RST_ACTIVATE) --,PULSE_FLAG)
   BEGIN
      IF ( I_MRST = '1' OR RST_ACTIVATE = '1') THEN
         ACTIVATE <= '0';
      ELSIF (I_SAMPLE_CLK'EVENT AND I_SAMPLE_CLK = '1') THEN
         ACTIVATE <= '1';
      END IF;
   END PROCESS;

   --COUNTERS TO KEEP TRACK OF NUMBER OF DATA BEING WRITTEN OR READ TO/FROM RAM
   RAM_WRITE_COUNTER:
   PROCESS(I_CLK, I_MRST,RAM_W_CNT_INC)
      BEGIN
         IF I_MRST = '1' THEN
               RAM_W_COUNTER <= 0;
         ELSIF I_CLK'EVENT AND I_CLK = '1' THEN
               IF RAM_W_CNT_INC = '1' THEN
                  RAM_W_COUNTER <= RAM_W_COUNTER + 1;
               END IF;
         END IF;
    END PROCESS;

   -- COUNTERS TO KEEP TRACK OF NUMBER OF DATA BEING WRITTEN OR READ TO/FROM RAM
   RAM_READ_COUNTER:
```

```
   PROCESS(I_CLOCK_2, I_MRST,RAM_R_CNT_INC, RAM_R_CNT_RST)
      BEGIN
          IF I_MRST = '1' OR RAM_R_CNT_RST = '1' THEN
                RAM_R_COUNTER <= 0;
          ELSIF I_CLOCK_2'EVENT AND I_CLOCK_2 = '1' THEN
                IF RAM_R_CNT_INC = '1' THEN
                   RAM_R_COUNTER <= RAM_R_COUNTER + 1;
                END IF;
          END IF;
    END PROCESS;

    -- 9-BIT ADDRESS COUNTER FOR (PORT A)
   RAM_ADDRESS_A_COUNTER:
   PROCESS (I_CLK, RAM_ADDR_RST, ADDR_INC_A)
   BEGIN
          IF (RAM_ADDR_RST = '1') THEN
             RAM_ADDR_REG_A <= (OTHERS => '0');
          ELSIF (I_CLK'EVENT AND I_CLK = '1') THEN
             IF (ADDR_INC_A = '1' ) THEN
                RAM_ADDR_REG_A <= RAM_ADDR_REG_A + 1;
             END IF;
          END IF;
   END PROCESS;

   -- PORT B USED TO READ DATA FROM RAM, CLOCKED WITH A SLOWER CLOCK
   RAM_ADDRESS_B_COUNTER:
   PROCESS (I_CLOCK_2, RAM_ADDR_RST, ADDR_INC_B)
   BEGIN
          IF (RAM_ADDR_RST = '1') THEN
             RAM_ADDR_REG_B  <= (OTHERS => '0');
          ELSIF (I_CLOCK_2'EVENT AND I_CLOCK_2 = '1') THEN
             IF (ADDR_INC_B = '1') THEN
                RAM_ADDR_REG_B <= RAM_ADDR_REG_B + 1;
             END IF;
          END IF;
   END PROCESS;

-- **** NORMAL OPERATION FSMs **** --
   -- DETECT FSM
   DETECT_STATES_ASSIGNMENT:
   PROCESS( CURR_D_STATE, NEXT_D_STATE, ACTIVATE, I_CAL_Mode, START_DISP)
      BEGIN

          RST_ACTIVATE <= '0';

          START_STORE  <= '0';

          CASE CURR_D_STATE IS
             WHEN T_D_INIT
                   => IF I_CAL_Mode = '1' OR START_DISP = '1' THEN
                            NEXT_D_STATE <= T_D_INIT;
                      ELSE  NEXT_D_STATE <= T_D_WAIT;
                      END IF;

             WHEN TMP -- FIFO FULL and DISPLAY starts? Then lock in T_D_INIT after few cycles
                   => IF START_DISP = '0' THEN
                            NEXT_D_STATE <= T_D_WAIT;
                      ELSE  NEXT_D_STATE <= T_D_INIT;
                      END IF;
                      START_STORE  <= '1';
                      RST_ACTIVATE <= '1';

             WHEN T_D_WAIT
                   => IF (ACTIVATE = '1') THEN
                         NEXT_D_STATE <= TMP;
                      ELSE
                         NEXT_D_STATE <= T_D_WAIT;
```

```
                       END IF;
      END CASE;
   END PROCESS;

   -- *** PROCESS FSM
   -- STATE ASSIGNMENTS FOR STORING DATA IN FIFO LATER TRANSFERED TO RAM
   STORE_TRANSFER_STATE_ASSIGNMENT:
   PROCESS(CURR_ST_STATE,  START_STORE, NEXT_ST_STATE, I_FIFO_FULL, I_FIFO_EMPTY)
      BEGIN
         -- OUTPUTS
         O_FIFO_Rd_En <= '0';     -- FIFO SIGNALS
         O_FIFO_Wr_En <= '0';
         O_SHIFTER_Ld <= '0';     -- SHIFTER SIGNALS
         O_SHIFT         <= "00";
         O_Ram_En_A      <= '0';     -- RAM SIGNALS
         O_Ram_We_A      <= '0';

         -- SIGNALS
         TEST_SIGNAL_1   <= '0';
         START_DISP      <= '0';    -- INTERNAL SIGNALS
         ADDR_INC_A      <= '0';
         RESET_A         <= '0';
         RAM_W_CNT_INC<= '0';

         CASE CURR_ST_STATE IS
            WHEN T0_ST
                  => NEXT_ST_STATE <= T_WAIT_ST;

                     -- SIGNALS
                     RESET_A   <= '1';

            WHEN T_WAIT_ST
                  => IF (START_STORE = '1' ) THEN
                        NEXT_ST_STATE <= T_WRITEFIFO_ST;
                     ELSE
                        NEXT_ST_STATE <= T_WAIT_ST;
                     END IF;

            WHEN T_WRITEFIFO_ST
                  => NEXT_ST_STATE    <= T_DUMMY0_ST;

                     --OUTPUTS
                     O_FIFO_Wr_En <= '1';

            WHEN T_DUMMY0_ST
                  => IF I_FIFO_FULL = '1' THEN
                           NEXT_ST_STATE    <= T_READFIFO_ST;
                     ELSE  NEXT_ST_STATE    <= T_WAIT_ST;
                     END IF;

            WHEN T_READFIFO_ST
                  => NEXT_ST_STATE    <= T_LDSHIFTER_ST;-- LOAD THE DATA FROM FIFO OUTPUT TO
SHIFTER REGISTER

                     --OUTPUTS
                     O_FIFO_Rd_En <= '1';     -- FIFO SIGNALS

            WHEN T_LDSHIFTER_ST
                  => NEXT_ST_STATE   <= T_SHIFT0_ST;

                     -- OUTPUTS
                     O_SHIFTER_Ld <= '1';

            WHEN T_SHIFT0_ST
                  => NEXT_ST_STATE    <= T_SHIFT1_ST;

                     -- OUTPUTS
```

```
                   O_SHIFT      <= "00";-- SHOWN EXPLICITLY

         WHEN T_SHIFT1_ST
                => NEXT_ST_STATE   <= T_SHIFT2_ST;

                   -- OUTPUTS
                   O_SHIFT      <= "01";
                   O_Ram_En_A   <= '1';
                   O_Ram_We_A   <= '1';

                   --SIGNALS
                   ADDR_INC_A      <= '1';

         WHEN T_SHIFT2_ST
                => NEXT_ST_STATE   <= T_SHIFT3_ST;

                   --OUTPUTS
                   O_SHIFT         <= "10";
                   O_Ram_En_A      <= '1';
                   O_Ram_We_A      <= '1';

                   --SIGNALS
                   ADDR_INC_A      <= '1';

         WHEN T_SHIFT3_ST
                => NEXT_ST_STATE   <= T_DUMMY1_ST;

                   --OUTPUTS
                   O_SHIFT         <= "11";
                   O_Ram_En_A      <= '1';
                   O_Ram_We_A      <= '1';

                   --SIGNALS
                   ADDR_INC_A      <= '1';
                   RAM_W_CNT_INC<= '1';

         WHEN T_DUMMY1_ST
                => IF I_FIFO_EMPTY = '1' THEN
                             NEXT_ST_STATE    <= T_RESET_RAM_ADDR_ST;
                   ELSE      NEXT_ST_STATE    <= T_READFIFO_ST;
                   END IF;

                   -- OUTPUTS
                   O_Ram_En_A      <= '1';
                   O_Ram_We_A      <= '1';

                   --SIGNALS
                   ADDR_INC_A      <= '1';

         WHEN T_RESET_RAM_ADDR_ST
                => NEXT_ST_STATE   <= T_F2RAM_COMP_ST;

                   --SIGNALS
                   RESET_A   <= '1';

         WHEN T_F2RAM_COMP_ST
                => NEXT_ST_STATE   <= T_F2RAM_COMP_ST;

                   --SIGNALS
                   START_DISP      <= '1';    -- INTERNAL SIGNALS
      END CASE;
END PROCESS;

-- ***** DISPLAY FSM **** --
-- DISPLAY STATE ASSIGNMENTS
DISPLAY_STATE_ASSIGNMENT:
   PROCESS(CURR_DISP_STATE, NEXT_DISP_STATE, START_DISP, I_MORE_DATA, RAM_R_COUNTER,
RAM_W_COUNTER)
       BEGIN
          --OUTPUT SIGNALS
          O_Ram_En_B      <= '0';
          O_Ram_We_B      <= '0';
          O_RamRead_Done  <= '0';
          O_Ld_0       <= '0';
          O_Ld_1       <= '0';
          O_Ld_2       <= '0';
          O_Ld_3       <= '0';
          O_DATA_RDY      <= '0';

          --SIGNALS
          ADDR_INC_B      <= '0';
          RESET_B         <= '0';
          RAM_R_CNT_INC<= '0';
          RAM_R_CNT_RST   <= '0';

          CASE CURR_DISP_STATE IS
             WHEN  T0_DISP
                => NEXT_DISP_STATE <=  T_WAIT_DISP;

                   --SIGNALS
                   RAM_R_CNT_RST   <= '1';

             WHEN   T_WAIT_DISP
                => IF START_DISP = '0' THEN
                         NEXT_DISP_STATE <= T_WAIT_DISP;
                   ELSE  NEXT_DISP_STATE <= T_READ0_DISP;
                   END IF;

             WHEN   T_READ0_DISP
                => NEXT_DISP_STATE <= T_READ1_DISP;

                   --OUTPUTS
                   O_Ram_En_B      <= '1';

                   --SIGNALS
                   ADDR_INC_B      <= '1';
                   RAM_R_CNT_INC<= '1';

             WHEN   T_READ1_DISP
                => NEXT_DISP_STATE <= T_READ2_DISP;

                   --OUTPUT
                   O_Ram_En_B      <= '1';
                   O_Ld_0       <= '1';

                   -- SIGNALS
                   ADDR_INC_B      <= '1';

             WHEN   T_READ2_DISP
                => NEXT_DISP_STATE <= T_READ3_DISP;

                   --OUTPUTS
                   O_Ram_En_B   <= '1';
                   O_Ld_1    <= '1';

                   --SIGNALS
                   ADDR_INC_B   <= '1';

             WHEN   T_READ3_DISP
                => NEXT_DISP_STATE <= T_TMP0_DISP;

                   --OUTPUTS
                   O_Ram_En_B   <= '1';
                O_Ld_2    <= '1';

                --SIGNALS
                ADDR_INC_B   <= '1';

            WHEN T_TMP0_DISP
              => NEXT_DISP_STATE <= T_TMP1_DISP;

                --OUTPUTS
                O_Ld_3 <= '1';

            WHEN T_TMP1_DISP
              => NEXT_DISP_STATE <= T_DISPLAY_DISP;

                --OUTPUTS
                O_DATA_RDY          <= '1';

            WHEN T_DISPLAY_DISP
              => IF I_MORE_DATA = '0' THEN
                    NEXT_DISP_STATE <= T_DISPLAY_DISP;
                    RAM_R_CNT_RST    <= '0';
                    RESET_B              <= '0';
                ELSE
                    NEXT_DISP_STATE <= T_READ0_DISP;
                    RAM_R_CNT_RST    <= '0';
                    RESET_B          <= '0';
                    -- CHANGED TO HARD CODED TO READ ONLY 100 OUT OF 128 DATA IN RAM
                    IF (RAM_R_COUNTER > 99) THEN --IF NEED TO RESTART FROM FIRST DATA IN RAM
                        RAM_R_CNT_RST  <= '1';
                        RESET_B         <= '1';
                    END IF;
                END IF;

                --OUTPUTS
                O_RamRead_Done   <= '1';
            END CASE;
    END PROCESS;

    DETECT_STORE_STATE_TRANSITIONS:
    PROCESS( I_MRST, I_CLK )
    BEGIN
        IF (I_MRST = '1') THEN
            CURR_D_STATE     <= T_D_INIT;
            CURR_ST_STATE       <= T0_ST;
        ELSIF (I_CLK'EVENT AND I_CLK = '1' ) THEN
            CURR_D_STATE        <= NEXT_D_STATE;
            CURR_ST_STATE       <= NEXT_ST_STATE;
        END IF;
    END PROCESS;

    DISPLAY_STATE_TRANSITIONS:
    PROCESS( I_MRST, I_CLOCK_2 )
    BEGIN
        IF (I_MRST = '1') THEN
            CURR_DISP_STATE <= T0_DISP;
        ELSIF (I_CLOCK_2'EVENT AND I_CLOCK_2 = '1' ) THEN
            CURR_DISP_STATE     <= NEXT_DISP_STATE;
        END IF;
    END PROCESS;


-- ******************************** CALIBRATION CONTROL LOGIC *********************************

    -- ***** CALIBRATION RAM ADDRESS ***** --
    O_CALRAM_Addr_A <= CAL_RAM_ADDR_REG_A;
    O_CALRAM_Addr_B <= CAL_RAM_ADDR_REG_B;
    CAL_RAM_ADDR_RST<= RESET_CAL_RAM_ADDR OR I_MRST;
```

```
CAL_RAM_ADDRESS_COUNTER_A:
PROCESS (I_CLOCK_2, CAL_RAM_ADDR_RST, CAL_RAM_ADDR_INC)
BEGIN
   IF (CAL_RAM_ADDR_RST = '1') THEN
      CAL_RAM_ADDR_REG_A <= (OTHERS => '0');-- ADDR DECOD VDL1/vREF1 => 0000000000 -
   0FFFFFFFFF
      CAL_RAM_ADDR_REG_B <= "1000000000"; --VDL-2 / vREF-2 => 1000000000 - 1FFFFFFFFF
   ELSIF (I_CLOCK_2'EVENT AND I_CLOCK_2 = '1') THEN
      IF (CAL_RAM_ADDR_INC = '1') THEN
         CAL_RAM_ADDR_REG_A <= CAL_RAM_ADDR_REG_A + 1;
         CAL_RAM_ADDR_REG_B <= CAL_RAM_ADDR_REG_B + 1;
      END IF;
   END IF;
END PROCESS;

-- ***** CAL_STORE_FSM ***** --
CAL_NEXT_STATE_TRANSITION:
PROCESS(NEXT_CAL_ST_STATE, NEXT_C_STATE, I_CLOCK_2, I_MRST)
BEGIN
   IF I_MRST = '1' THEN
      CURR_CAL_ST_STATE <= T_CAL_ST_S0;
      CURR_C_STATE     <= T_INIT;
   ELSIF I_CLOCK_2'EVENT AND I_CLOCK_2 = '1' THEN
      CURR_CAL_ST_STATE <= NEXT_CAL_ST_STATE;
      CURR_C_STATE        <= NEXT_C_STATE;
   END IF;
END PROCESS;

-- CAL_STORE_FSM STATE/OUTPUT ASSIGNMENT
CAL_STORE_STATE_ASSIGNMENT:
PROCESS( CURR_CAL_ST_STATE, NEXT_CAL_ST_STATE, CONTINUE_FLAG, MORE_2_STORE,
         CAL_RAM_ADDR_REG_A, I_MORE_DATA, CNT_DATA_1_READY, CNT_DATA_2_READY  )
   BEGIN

   --OUTPUTS
   O_CAL_DATA_RDY  <= '0';
   O_Ld_B_Data        <= '0';
   O_Ld_L_Data        <= '0';
   O_CALRAM_En_A      <= '0';
   O_CALRAM_We_A      <= '0';
   O_CALRAM_En_B      <= '0';
   O_CALRAM_We_B      <= '0';

   -- SIGNALS
   CAL_RAM_ADDR_INC   <= '0';
   RESET_CAL_RAM_ADDR <= '0';
   CAL_DATA_STORED    <= '0';
   WAIT_FOR_MORE_DATA <= '0';
   -- TEST_SIGNAL_1   <= '0';

   CASE CURR_CAL_ST_STATE IS
         WHEN  T_CAL_ST_S0
                  => NEXT_CAL_ST_STATE <= T_CAL_ST_WAIT_1;
                     --OUTPUTS

                     -- SIGNALS
                     RESET_CAL_RAM_ADDR<= '1';

         WHEN  T_CAL_ST_WAIT_1
                  => IF (CNT_DATA_1_READY = '1') THEN

                           NEXT_CAL_ST_STATE   <= T_CAL_ST_WRITE_1;

                     ELSE  NEXT_CAL_ST_STATE   <= T_CAL_ST_WAIT_1;
```

```
                   END IF;

      WHEN  T_CAL_ST_WRITE_1
                => NEXT_CAL_ST_STATE   <= T_CAL_ST_WAIT_2;

                   --OUTPUTS
                O_CALRAM_En_A <= '1';-- DATA FROM BUF COUNTERS (M1, M2) INTO PORTA & PORTB
                   O_CALRAM_We_A    <= '1';
                   O_CALRAM_En_B    <= '1';
                   O_CALRAM_We_B    <= '1';

                   -- SIGNALS
                   CAL_RAM_ADDR_INC<= '1';

      WHEN  T_CAL_ST_WAIT_2
                => IF (CNT_DATA_2_READY = '1') THEN
                          NEXT_CAL_ST_STATE   <= T_CAL_ST_WRITE_2;
                   ELSE   NEXT_CAL_ST_STATE   <= T_CAL_ST_WAIT_2;
                   END IF;

                   -- SIGNALS
                   WAIT_FOR_MORE_DATA  <= '1';

      WHEN  T_CAL_ST_WRITE_2
                => NEXT_CAL_ST_STATE   <= T_CAL_ST_DONE;
                   --OUTPUTS
O_CALRAM_En_A      <= '1';-- DATA FROM BOTH LATCH COUNTERS (M1, M2) INTO PORTA AND PORTB
                   O_CALRAM_We_A       <= '1';
                   O_CALRAM_En_B       <= '1';
                   O_CALRAM_We_B       <= '1';

                   -- SIGNALS
                   CAL_RAM_ADDR_INC    <= '1';   -- INCREMENT BOTH ADDRESS REGISTERS

      WHEN  T_CAL_ST_DONE
                --The Cal state has to either be in Next_row or Final state
                => IF CONTINUE_FLAG = '1' THEN
                       IF (MORE_2_STORE = '1') THEN
                             NEXT_CAL_ST_STATE   <= T_CAL_ST_WAIT_1;
                       ELSE  NEXT_CAL_ST_STATE   <= T_CAL_ST_END;
                       END IF;
                   ELSE   NEXT_CAL_ST_STATE   <= T_CAL_ST_DONE;
                   END IF;

                   -- SIGNALS
                   CAL_DATA_STORED <= '1';

      WHEN  T_CAL_ST_END
                => NEXT_CAL_ST_STATE   <= T_CAL_RAM_READ; --T_CAL_DISP;

                   -- SIGNALS
                   RESET_CAL_RAM_ADDR  <= '1';-- RESET ADDRESS REGISTER READY FOR READING

      WHEN T_CAL_RAM_READ
                => NEXT_CAL_ST_STATE <= T_CNT_DATA_LOAD_1;

                   --OUTPUTS
                   O_CALRAM_En_A       <= '1';     -- READ FROM RAM
                   O_CALRAM_En_B       <= '1';


          WHEN T_CNT_DATA_LOAD_1
                => NEXT_CAL_ST_STATE <= T_NEXT_ADDRESS;
                   -- OUTPUTS

                   O_Ld_B_Data  <= '1'; -- LOAD first batch of DATA FROM RAM INTO LCD REGS
```

```
                          -- SIGNALS
                          CAL_RAM_ADDR_INC   <= '1';

             WHEN T_NEXT_ADDRESS
                       => NEXT_CAL_ST_STATE <= T_CNT_DATA_LOAD_2;

                          --OUTPUTS
                          O_CALRAM_En_A   <= '1';
                          O_CALRAM_En_B   <= '1';

             WHEN T_CNT_DATA_LOAD_2
                       =>  NEXT_CAL_ST_STATE <= T_CAL_DISP;

                          --OUTPUTS
                          O_Ld_L_Data     <= '1';-- LAOD DATA FROM RAM  INTO LATCH REGISTER OF LCD

                          -- SIGNALS
                          CAL_RAM_ADDR_INC   <= '1';

             WHEN T_CAL_DISP
                       => NEXT_CAL_ST_STATE <= T_WAIT_FOR_MORE;

                          --OUTPUTS
                          O_CAL_DATA_RDY <= '1';

             WHEN T_WAIT_FOR_MORE
                       => IF (I_MORE_DATA = '1') THEN
                                NEXT_CAL_ST_STATE <=    T_TMP_CAL_DISP;
                          ELSE  NEXT_CAL_ST_STATE <=    T_WAIT_FOR_MORE;
                          END IF;

             WHEN T_TMP_CAL_DISP
                       => NEXT_CAL_ST_STATE <= T_CAL_RAM_READ;
                          IF CAL_RAM_ADDR_REG_A > "0000110001" THEN
                                RESET_CAL_RAM_ADDR <= '1';
                          ELSE  RESET_CAL_RAM_ADDR <= '0';
                          END IF;

     END CASE;
   END PROCESS;

   -- ***** CAL_CONFIG_FSM ****** --
   CAL_CONFIG_STATE_ASSIGNMENTS:
   PROCESS( CURR_C_STATE, NEXT_C_STATE,I_CAL_Mode, START_CAL, LINE_DONE, CAL_DATA_STORED,
WAIT_FOR_MORE_DATA, CAL_ROUND)
   BEGIN

          -- OUTPUTS --
          O_CNT_En     <=  '0';
          O_RO_Sel     <=  '0';
          O_CNT_Rst <=  '0';

          -- SIGNALS
          RST_START_CAL   <= '0';
          CONTINUE_FLAG<= '0';
          MORE_2_STORE    <= '0';
          CNT_DATA_1_READY<= '0';
          CNT_DATA_2_READY<= '0';
          RST_LINE_DONE<= '0';
          RND_CNT_INC     <= '0';
          RST_RND_CNT     <= '0';
          TEST_SIGNAL     <= '0';   -- FOR DEBUGGING PURPOSES


          CASE CURR_C_STATE IS
                WHEN T_INIT
                       => IF I_CAL_Mode = '1' THEN
```

```
                                NEXT_C_STATE <= T_CAL_CONFIG1;
                          ELSE  NEXT_C_STATE <= T_INIT;
                          END IF;

                          -- OUTPUTS --
                          O_CNT_Rst <=  '1';

                          -- SIGNALS
                          RST_START_CAL   <= '1';
                          RST_LINE_DONE<= '1';
                          RST_RND_CNT     <= '1';

                WHEN T_CAL_CONFIG1
                       => IF (START_CAL = '1' ) THEN    -- IN THIS STATE, CONTROL WAITS FOR
START_cAL PULSE
                                NEXT_C_STATE <= T_CAL_OP1;
                          ELSE  NEXT_C_STATE <= T_CAL_CONFIG1;
                          END IF;

                          -- OUTPUTS --
                          O_RO_Sel     <=  '0';     -- O_RO_Sel = 0 SELECTS FIRST GROUP OF 4 RING
OSCI.
                          O_CNT_Rst <=  '1';

                          -- SIGNALS
                          CONTINUE_FLAG<= '1';
                          MORE_2_STORE    <= '1';      -- KEPT HIGH FOR 2 STATES
                          RST_LINE_DONE<= '1';

                WHEN T_CAL_OP1
                       => IF LINE_DONE = '1' THEN      -- LINE DONE is asserted when T_CAL makes
a H-2-L transition
                                NEXT_C_STATE <= T_CAL_CNT_STORE_1;
                          ELSE  NEXT_C_STATE <= T_CAL_OP1;
                          END IF;

                          -- OUTPUTS --
                          O_CNT_En     <= '1';
                          O_RO_Sel     <= '0';

                          -- SIGNALS
                          RST_START_CAL<= '1';   -- BRING START_CAL DOWN

                WHEN T_CAL_CNT_STORE_1
                       => IF WAIT_FOR_MORE_DATA = '1' THEN
                                NEXT_C_STATE <= T_CAL_CONFIG2;
                          ELSE  NEXT_C_STATE <= T_CAL_CNT_STORE_1;
                          END IF;

                          -- SIGNALS
                          RST_LINE_DONE<= '1';     -- desactivate LINE_DONE
                          CNT_DATA_1_READY<= '1';

                WHEN T_CAL_CONFIG2
                       => IF (START_CAL = '1' ) THEN
                                NEXT_C_STATE <= T_CAL_OP2;
                          ELSE  NEXT_C_STATE <= T_CAL_CONFIG2;
                          END IF;

                          -- OUTPUTS --
                          O_RO_Sel     <=  '1';
                          O_CNT_Rst <=  '1';   -- RESET ALL COUNTERS FOR NEW CAL DATA FROM NEXT 4
RO

                          -- SIGNALS
                          RST_LINE_DONE<= '1';
```

```
             WHEN T_CAL_OP2
                    => IF (LINE_DONE = '1') THEN
                              NEXT_C_STATE <= T_CAL_CNT_STORE_2;
                       ELSE   NEXT_C_STATE <= T_CAL_OP2;
                       END IF;

                       -- OUTPUTS --
                       O_CNT_En      <=   '1';
                       O_RO_Sel      <=   '1';

                       -- SIGNALS
                       RST_START_CAL    <= '1';

             WHEN T_CAL_CNT_STORE_2
                    => IF (CAL_DATA_STORED = '1') THEN
                          IF CAL_ROUND < 25 THEN  -- IF MORE LINES TO COFIGURE FOR CALIBRATION
                                NEXT_C_STATE <= T_CAL_NEXT_RND;
                          ELSE  NEXT_C_STATE <= T_CAL_START_DISP;
                          END IF;
                       ELSE   NEXT_C_STATE <= T_CAL_CNT_STORE_2;
                       END IF;

                       -- OUTPUTS --
                       O_RO_Sel      <=   '1';

                       -- SIGNALS
                       RST_START_CAL    <= '1';
                       RST_LINE_DONE<= '1';
                       CNT_DATA_2_READY<= '1';

             WHEN T_CAL_NEXT_RND
                    => NEXT_C_STATE <= T_CAL_CONFIG1;
                       -- OUTPUTS --
                       O_RO_Sel  <=   '1';

                       -- SIGNALS
             -- THIS FLAG DIRECTS STORE MACHINE TO WAIT STATE FOR MORE DATA STORAGE

                       MORE_2_STORE <= '1';
                       -- THIS FLAG DIRECTS STORE MACHINE TO EITHER wait_store OR end_store

                       CONTINUE_FLAG<= '1';
                       RST_START_CAL    <= '1';
                       RST_LINE_DONE<= '1';
                       RND_CNT_INC    <= '1'; -- TAKE ACCOUNT OF NUMBER OF CALIBRATION RUNS

             WHEN T_CAL_START_DISP
                    => NEXT_C_STATE <= T_CAL_START_DISP;
                          -- OUTPUTS --
                       O_CNT_Rst <=   '1';

                       -- SIGNALS
                       RST_START_CAL    <= '1';
                       RST_LINE_DONE<= '1';
                       CONTINUE_FLAG<= '1';

                       RND_CNT_INC    <= '0';
                       RST_RND_CNT    <= '1';
       END CASE;
END PROCESS;

-- COUNTER USED FOR NUMBER OF CALIBRATION ROUNDS
SELECT_VALUE_COUNTER:
PROCESS(I_CLOCK_2, I_MRST, RND_CNT_INC, RST_RND_CNT)
BEGIN
   IF (I_MRST = '1' OR RST_RND_CNT = '1') THEN CAL_ROUND <= 0;
   ELSIF (I_CLOCK_2'EVENT AND I_CLOCK_2 = '1') THEN
```

```
         IF RND_CNT_INC = '1' THEN
            CAL_ROUND <= CAL_ROUND + 1;
         END IF;
      END IF;
   END PROCESS;


   -- CONDITIONS FOR BUF_LINE_DONE AND LAT_LINE_DONE (END OF CAL. RUN)
   LINE_CAL_DONE_CONDITION:
   PROCESS( I_MRST, I_CAL_CLK, RST_LINE_DONE)
   BEGIN
      IF I_MRST = '1' OR RST_LINE_DONE = '1' THEN
         LINE_DONE <= '0';
      ELSIF I_CAL_CLK'EVENT AND I_CAL_CLK = '0' THEN
         LINE_DONE <= '1';
      END IF;
   END PROCESS;

   -- START CAL CONDITION (START OF CAL. RUN)
   START_CAL_CONDITION:
   PROCESS( I_MRST, I_CAL_CLK, RST_START_CAL, I_CAL_Mode)
   BEGIN
      IF I_MRST = '1' OR RST_START_CAL = '1' THEN
         START_CAL <= '0';
      ELSIF I_CAL_CLK'EVENT AND I_CAL_CLK = '1' THEN
         IF I_CAL_Mode = '1' THEN
            START_CAL <= '1';
         END IF;
      END IF;
   END PROCESS;

end Behavioral;
```

♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦Encoder ELEMENT♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦

```
-- **********************************************************************************
-- VHDL code for an encoder used to latch and encode the tap lines of the TDC matrix
-- Author: Amiri Amir Mohammad
-- Date: SEPT 22, 2005
-- Remarks: for ERRONEOUS TAP READINGs ENCODED VALUE IS "F"
--        . UPDATE: code clean up on :22march2007
-- **********************************************************************************
-- *************************************************************** --
--                            LIBRARIES
-- *************************************************************** --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- *************************************************************** --
--                            ENTITY DECLARATION
-- *************************************************************** --
entity encoder is
      Port (
         M_TAPS_R0: IN std_logic_vector(0 TO 14);
         M_TAPS_R1: IN std_logic_vector(0 TO 14);
         M_TAPS_R2: IN std_logic_vector(0 TO 14);
         M_TAPS_R3: IN std_logic_vector(0 TO 14);
         M_TAPS_R4: IN std_logic_vector(0 TO 14);
         M_TAPS_R5: IN std_logic_vector(0 TO 14);
         M_TAPS_R6: IN std_logic_vector(0 TO 14);
         M_TAPS_R7: IN std_logic_vector(0 TO 14);
         M_TAPS_R8 : IN std_logic_vector(0 TO 14);
         M_TAPS_R9 : IN std_logic_vector(0 TO 14);
         M_TAPS_R10   : IN std_logic_vector(0 TO 14);
         M_TAPS_R11   : IN std_logic_vector(0 TO 14);
```

```
          M_TAPS_R12    : IN std_logic_vector(0 TO 14);
          M_TAPS_R13    : IN std_logic_vector(0 TO 14);
          M_TAPS_R14    : IN std_logic_vector(0 TO 14);
          M_TAPS_R15    : IN std_logic_vector(0 TO 14);
          M_TAPS_R16    : IN std_logic_vector(0 TO 14);
          M_TAPS_R17    : IN std_logic_vector(0 TO 14);
          M_TAPS_R18    : IN std_logic_vector(0 TO 14);
          M_TAPS_R19    : IN std_logic_vector(0 TO 14);
          M_TAPS_R20    : IN std_logic_vector(0 TO 14);
          M_TAPS_R21: IN std_logic_vector(0 TO 14);
          M_TAPS_R22    : IN std_logic_vector(0 TO 14);
          M_TAPS_R23    : IN std_logic_vector(0 TO 14);
          M_TAPS_R24    : IN std_logic_vector(0 TO 14);

          COARSE_COUNT : IN STD_LOGIC_VECTOR(11 DOWNTO 0);

          I_SAMPLE_CLOCK : IN std_logic;   -- DELAYED SAMPLE CLOCK FROM MATRICES
          I_RESET          : IN std_logic; -- MASTER RESET OUT OF CONTROLLER
          ENCODED_OUT   : OUT std_logic_vector(111 DOWNTO 0)
   );
end encoder;
-- ************************************************************** --
--                     ARCHITECTURE
-- ************************************************************** --
ARCHITECTURE Behavioral OF encoder IS

-- *** ATTRIBUTES *** --
   ATTRIBUTE KEEP_HIERARCHY : STRING;
   ATTRIBUTE KEEP : STRING;

   ATTRIBUTE KEEP_HIERARCHY OF BEHAVIORAL : ARCHITECTURE IS "YES";

-- INTERNAL REGISTER TO HOLD INCOMING TAP VALUES
 SIGNAL TAPS_R0_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R1_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R2_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R3_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R4_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R5_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R6_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R7_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R8_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R9_REGISTER    : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R10_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R11_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R12_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R13_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R14_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R15_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R16_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R17_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R18_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R19_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R20_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R21_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R22_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R23_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');
 SIGNAL TAPS_R24_REGISTER   : STD_LOGIC_VECTOR(0 TO 14) := (others => '0');

 SIGNAL ENCODED : STD_LOGIC_VECTOR(99 DOWNTO 0);
 SIGNAL COUNTER_VALUE : STD_LOGIC_VECTOR(11 DOWNTO 0);
 SIGNAL TAPS_REGISTERED     : STD_LOGIC;

BEGIN
-- ****** CSA ******
   ENCODED_OUT <= COUNTER_VALUE & ENCODED;
```

```
-- **** INTERNAL_REGISTERS
   PROCESS (I_SAMPLE_CLOCK, I_RESET)
   BEGIN
   IF I_RESET = '1' THEN
       TAPS_R0_REGISTER  <=(others => '0');
       TAPS_R1_REGISTER  <=(others => '0');
       TAPS_R2_REGISTER  <=(others => '0');
       TAPS_R3_REGISTER  <=(others => '0');
       TAPS_R4_REGISTER  <=(others => '0');
       TAPS_R5_REGISTER  <=(others => '0');
       TAPS_R6_REGISTER  <=(others => '0');
       TAPS_R7_REGISTER  <=(others => '0');
       TAPS_R8_REGISTER    <=(others => '0');
       TAPS_R9_REGISTER    <=(others => '0');
       TAPS_R10_REGISTER <=(others => '0');
       TAPS_R11_REGISTER <=(others => '0');
       TAPS_R12_REGISTER <=(others => '0');
       TAPS_R13_REGISTER <=(others => '0');
       TAPS_R14_REGISTER <=(others => '0');
       TAPS_R15_REGISTER <=(others => '0');
       TAPS_R16_REGISTER <=(others => '0');
       TAPS_R17_REGISTER <=(others => '0');
       TAPS_R18_REGISTER <=(others => '0');
       TAPS_R19_REGISTER <=(others => '0');
       TAPS_R20_REGISTER <=(others => '0');
       TAPS_R21_REGISTER <=(others => '0');
       TAPS_R22_REGISTER <=(others => '0');
       TAPS_R23_REGISTER <=(others => '0');
       TAPS_R24_REGISTER <=(others => '0');

       TAPS_REGISTERED    <= '0';
       COUNTER_VALUE      <= (OTHERS => '0');

   ELSIF (I_SAMPLE_CLOCK'EVENT AND I_SAMPLE_CLOCK = '1') THEN
      TAPS_R0_REGISTER <= M_TAPS_R0;
      TAPS_R1_REGISTER <= M_TAPS_R1;
      TAPS_R2_REGISTER <= M_TAPS_R2;
      TAPS_R3_REGISTER <= M_TAPS_R3;
      TAPS_R4_REGISTER <= M_TAPS_R4;
      TAPS_R5_REGISTER <= M_TAPS_R5;
      TAPS_R6_REGISTER <= M_TAPS_R6;
      TAPS_R7_REGISTER <= M_TAPS_R7;
      TAPS_R8_REGISTER<= M_TAPS_R8;
      TAPS_R9_REGISTER<= M_TAPS_R9;
      TAPS_R10_REGISTER <= M_TAPS_R10;
      TAPS_R11_REGISTER <= M_TAPS_R11;
      TAPS_R12_REGISTER <= M_TAPS_R12;
      TAPS_R13_REGISTER <= M_TAPS_R13;
      TAPS_R14_REGISTER <= M_TAPS_R14;
      TAPS_R15_REGISTER <= M_TAPS_R15;
      TAPS_R16_REGISTER <=  M_TAPS_R16;
      TAPS_R17_REGISTER <=  M_TAPS_R17;
      TAPS_R18_REGISTER <=  M_TAPS_R18;
      TAPS_R19_REGISTER <=  M_TAPS_R19;
      TAPS_R20_REGISTER <=  M_TAPS_R20;
      TAPS_R21_REGISTER <=  M_TAPS_R21;
      TAPS_R22_REGISTER <=  M_TAPS_R22;
      TAPS_R23_REGISTER <=  M_TAPS_R23;
      TAPS_R24_REGISTER <=  M_TAPS_R24;
      TAPS_REGISTERED     <= '1';
      COUNTER_VALUE   <= COARSE_COUNT;
   END IF;
   END PROCESS;

-- ************* ENCODERS ****************
   -- PROCESS FOR THE ENCODING PROCEDURE OF THE INPUT TAPS BASED ON:
   --                           BIT POSITIONS    => INDICATE ROW NUMBER    (n)
   --                            4-BIT VALUE  => INDICATE THE DELAY CELLS HAVING (m) (0 - E)
   --               THEREFORE:    (25X15)-BITS INPUT => 100-BIT ENCODED VALUE (25X4)
   --                            ALSO: 11-MSB-BITS FOR COARSE CNT
    -- PROCESS FOR THE REGISTERED ENCODER
   ENCODERS:
   PROCESS (
          TAPS_R0_REGISTER  , TAPS_R1_REGISTER  , TAPS_R2_REGISTER  ,
          TAPS_R3_REGISTER  , TAPS_R4_REGISTER  , TAPS_R5_REGISTER  ,
          TAPS_R6_REGISTER  , TAPS_R7_REGISTER  , TAPS_R8_REGISTER  ,
          TAPS_R9_REGISTER  , TAPS_R10_REGISTER , TAPS_R11_REGISTER ,
          TAPS_R12_REGISTER , TAPS_R13_REGISTER , TAPS_R14_REGISTER ,
          TAPS_R15_REGISTER , TAPS_R16_REGISTER , TAPS_R17_REGISTER ,
          TAPS_R18_REGISTER , TAPS_R19_REGISTER , TAPS_R20_REGISTER ,
          TAPS_R21_REGISTER , TAPS_R22_REGISTER, TAPS_R23_REGISTER,
          TAPS_R24_REGISTER , I_RESET, TAPS_REGISTERED                  )
      BEGIN
          IF (I_RESET = '1' ) then
             ENCODED <=   (others => '0');
          ELSE
             IF (TAPS_REGISTERED = '1') THEN

                -- ENCODER 1 (1-OUT-OF-MANY-ENCODING)
                CASE TAPS_R0_REGISTER IS
                   WHEN "000000000000001" => ENCODED(3 DOWNTO 0) <= x"F";
                   WHEN "000000000000010" => ENCODED(3 DOWNTO 0) <= x"E";
                   WHEN "000000000000100" => ENCODED(3 DOWNTO 0) <= x"D";
                   WHEN "000000000001000" => ENCODED(3 DOWNTO 0) <= x"C";
                   WHEN "000000000010000" => ENCODED(3 DOWNTO 0) <= x"B";
                   WHEN "000000000100000" => ENCODED(3 DOWNTO 0) <= x"A";
                   WHEN "000000001000000" => ENCODED(3 DOWNTO 0) <= x"9";
                   WHEN "000000010000000" => ENCODED(3 DOWNTO 0) <= x"8";
                   WHEN "000000100000000" => ENCODED(3 DOWNTO 0) <= x"7";
                   WHEN "000001000000000" => ENCODED(3 DOWNTO 0) <= x"6";
                   WHEN "000010000000000" => ENCODED(3 DOWNTO 0) <= x"5";
                   WHEN "000100000000000" => ENCODED(3 DOWNTO 0) <= x"4";
                   WHEN "001000000000000" => ENCODED(3 DOWNTO 0) <= x"3";
                   WHEN "010000000000000" => ENCODED(3 DOWNTO 0) <= x"2";
                   WHEN "100000000000000" => ENCODED(3 DOWNTO 0) <= x"1";
                   WHEN OTHERS => ENCODED(3 DOWNTO 0) <= x"0";
                END CASE;

-- ENCODER 1 (THEROMOMETRIC ENCODING)
--IF   ( TAPS_R0_REGISTER(14) = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"F";  "111111111111111"
--ELSIF ( TAPS_R0_REGISTER(13) = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"E"; "111111111111110"
--ELSIF ( TAPS_R0_REGISTER(12) = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"D";   "111111111111100"
--ELSIF ( TAPS_R0_REGISTER(11) = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"C";   "111111111111000"
--ELSIF ( TAPS_R0_REGISTER(10) = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"B";   "111111111110000"
--ELSIF ( TAPS_R0_REGISTER(9)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"A";   "111111111100000"
--ELSIF ( TAPS_R0_REGISTER(8)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"9";   "111111111000000"
--ELSIF ( TAPS_R0_REGISTER(7)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"8";      "111111110000000"
--ELSIF ( TAPS_R0_REGISTER(6)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"7";      "111111100000000"
   --ELSIF ( TAPS_R0_REGISTER(5)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"6";   "111111000000000"
   --ELSIF ( TAPS_R0_REGISTER(4)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"5";   "111110000000000"
   --ELSIF ( TAPS_R0_REGISTER(3)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"4";   "111100000000000"
   --ELSIF ( TAPS_R0_REGISTER(2)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"3";   "111000000000000"
   --ELSIF ( TAPS_R0_REGISTER(1)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"2";   "110000000000000"
   --ELSIF ( TAPS_R0_REGISTER(0)  = '1' ) THEN ENCODED(3 DOWNTO 0) <= x"1";   "100000000000000"
   --ELSE    --   ENCODED(3 DOWNTO 0) <= x"0";
             --END IF;

- - - NOT ALL CASES SHOWN..ALL 25 ENCODERS ARE SIMILAR TO THE CASES IN ABOVE FOR DIFF TAP LINES

            ELSE
              ENCODED <= (OTHERS=> '0');
          END IF;
      END IF;
   END PROCESS;
```

```
end Behavioral;
```

```
♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦ LCD DRIVER ♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦
-- ******************************************************************************************
-- MODULE: LCD_DRIVER, used to generate character ASCII for 16X2 LCD DISPLAY
-- Author: Amiri Amir Mohammad
-- Date: 4-MARCH-2007
-- Remarks:
-- ******************************************************************************************
-- ************************************************************** --
--                  LIBRARIES
-- ************************************************************** --
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;
-- ************************************************************** --
--                  ENTITY DECLARATION
-- ************************************************************** --
ENTITY LCD_driver IS
    PORT (  I_CLK       : IN STD_LOGIC;
             I_RESET       : IN STD_LOGIC;
             I_DATA     : IN STD_LOGIC_VECTOR(31 DOWNTO 0);

          -- CALIBRATION INTERFACE
          I_CAL_MODE       :IN STD_LOGIC;
          I_CALDATA_RDY:IN STD_LOGIC;
          I_Ld_CAL_BDATA   :IN STD_LOGIC;
          I_Ld_CAL_LDATA   :IN STD_LOGIC;

          -- DATA INTERFACES TO CALIBRATION RAM DATA BUS
          HOR_CAL_DATA_1 :IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          HOR_CAL_DATA_2 :IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          VER_CAL_DATA_1 :IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          VER_CAL_DATA_2 :IN STD_LOGIC_VECTOR(15 DOWNTO 0);

          -- HANDSHAKE SIGNALS WITH CONTROLLER
          I_Ld_0     : IN STD_LOGIC;
          I_Ld_1     : IN STD_LOGIC;
          I_Ld_2     : IN STD_LOGIC;
          I_Ld_3     : IN STD_LOGIC;
          I_DATA_RDY    : IN STD_LOGIC;
          O_MORE_DATA   : OUT STD_LOGIC; -- WHEN NEW I_DATA IS REQUESTED

          -- OUTPUT ON LEDs
          T_OUT                : OUT STD_LOGIC;
          T_OUT_1              : OUT STD_LOGIC;
             NEW_DATA_REGISTERED    : OUT STD_LOGIC;

          -- LCD DISPLAY INTERFACE
          O_LCD_DATA    : OUT STD_LOGIC_VECTOR(7 downto 0);
             O_LCD_RS      : OUT STD_LOGIC;
             O_LCD_We_Re   : OUT STD_LOGIC;
             O_LCD_E       : OUT STD_LOGIC
   );
END LCD_driver;
-- ************************************************************** --
--                            ARCHITECTURE
-- ************************************************************** --
ARCHITECTURE Behavioral OF LCD_driver IS
-- *** ATTRIBUTES *** --
   ATTRIBUTE KEEP_HIERARCHY: STRING;
   ATTRIBUTE KEEP_HIERARCHY OF BEHAVIORAL: ARCHITECTURE IS "YES";

-- *** FLAGs
   -- CONTROL FLAGS used to DETERMINE CURRENT DISPLAY STATE
   TYPE CONTROL_FLAGS IS (WAIT_FLAG, DISP_FLAG, REQ_FLAG);
   SIGNAL CURR_FLAG : CONTROL_FLAGS := WAIT_FLAG;

-- *** LCD FSMs *** --
   -- CONTROL FSM, used to CONTROL STATE OF DISPLAY on LCD
   TYPE CONTROL_STATES IS
   (  T_0, T_WAIT_A, T_WAIT_B, T_DISPLAY, T_TMP, T_REQUEST_A, T_REQUEST_B  );
   SIGNAL CURR_CONT_STATE, NEXT_CONT_STATE : CONTROL_STATES := T_0;

   -- FSM used to determine valid ascii to be sent TO LCD
   TYPE LCD_STATES IS (
      S0, S1, S2, S3, S4, S5, S4_1, S5_1,  -- REQUIRED FOR LCD INITIALIZATION
      S_W, S_A, S_I0, S_T, S_I1, S_N, S_G,-- "WAITING"
      S_DOT0, S_DOT1, S_DOT2,              -- "..."
      S_F, S_C, S_H, S_E,                  -- "FETCHING"
      S_DOTS, S_NEXT_LINE,                 -- ":" and NEW LINE
      S_CRS_1 , S_CRS_2 ,S_CRS_3,          -- FOR DISPLAYING COARSE VALUE(4 HEX DIGITS)
      S_MINUS,                          -- "-"

      -- CALIBRATION DATA DISPLAY
      S_D_CAL, S_DIGIT_1, S_DIGIT_2,
      S_H1_CAL,                            -- "H"
      S_VDL1_BUF_CNT1, S_VDL1_BUF_CNT2, S_VDL1_BUF_CNT3, S_VDL1_BUF_CNT4,
      S_VDL1_LAT_CNT1, S_VDL1_LAT_CNT2, S_VDL1_LAT_CNT3, S_VDL1_LAT_CNT4,

      S_V1_CAL,                         -- "V"
      S_VREF1_CNT1, S_VREF1_CNT2, S_VREF1_CNT3, S_VREF1_CNT4,
      S_V_RO1_CNT1, S_V_RO1_CNT2, S_V_RO1_CNT3, S_V_RO1_CNT4,

      S_PHASE_DIGIT, S_SPACE_0, S_SPACE_1 ,S_SPACE_2 ,
      S_SPACE_3, S_SPACE_4, S_SPACE_5, S_SPACE_6,

      S_H2_CAL,
      S_VDL2_BUF_CNT1, S_VDL2_BUF_CNT2, S_VDL2_BUF_CNT3, S_VDL2_BUF_CNT4,
      S_VDL2_LAT_CNT1, S_VDL2_LAT_CNT2, S_VDL2_LAT_CNT3, S_VDL2_LAT_CNT4,

      S_V2_CAL,
      S_VREF2_CNT1, S_VREF2_CNT2, S_VREF2_CNT3, S_VREF2_CNT4,
      S_V_RO2_CNT1, S_V_RO2_CNT2, S_V_RO2_CNT3, S_V_RO2_CNT4,

      -- STATES TO DISPLAY LSB CONTENTS (ENCODED DATA CORRESPONDING TO EACH ROW OF THE MATRIX)
      S_LSB_0 , S_LSB_1 , S_LSB_2 , S_LSB_3 , S_LSB_4 ,
      S_LSB_5 , S_LSB_6 , S_LSB_7 , S_LSB_8 , S_LSB_9 ,
      S_LSB_10, S_LSB_11, S_LSB_12, S_LSB_13, S_LSB_14,
      S_LSB_15, S_LSB_16, S_LSB_17, S_LSB_18, S_LSB_19,
      S_LSB_20, S_LSB_21, S_LSB_22, S_LSB_23, S_LSB_24
   );
   SIGNAL CURR_LCD_STATE, NEXT_LCD_STATE : LCD_STATES;

-- *** SIGNALS *** --
   SIGNAL SEC_2_CNT       : UNSIGNED(31 DOWNTO 0) := (OTHERS => '0');
   SIGNAL MORE_COUNTER    : INTEGER RANGE 0 TO 3;
   SIGNAL MORE_CNT_RST :STD_LOGIC;
   SIGNAL CNT_RST         :STD_LOGIC;
   SIGNAL MORE_FLAG       :STD_LOGIC;


   -- COUNTER used to switch display data on LCD DISPLAY
   SIGNAL ROUND_COUNTER : INTEGER RANGE 0 TO 100 := 0;

   -- A PRESCALER SIGNAL USED TO PRODUCE LOW FREQUENCY CLOCK FOR THE LCD
   SIGNAL PRESCALER : STD_LOGIC_VECTOR(18 downto 0);
   SIGNAL LCD_CLK     : STD_LOGIC;

   -- INTERNAL REGISTERS
   SIGNAL REG0, REG1, REG2, REG3 : STD_LOGIC_VECTOR(31 DOWNTO 0) :=(OTHERS => '0');

   -- REGISTERS FOR HOLDING COARSE AND FINE PARTS OF THE READINGS
   SIGNAL MSB_REG   : STD_LOGIC_VECTOR(11  DOWNTO 0):= (OTHERS => '0');
   SIGNAL LSB_REG   : STD_LOGIC_VECTOR(99 DOWNTO 0) := (OTHERS => '0');

   -- INTERNAL SIGNAL FOR HOLDING ASCII CORRESPONDING TO LETTERS OF HEX SYSTEM
   SIGNAL DECODED_ASCII : STD_LOGIC_VECTOR(0 TO 7) := (OTHERS => '0');

   -- INTERNAL SIGNAL HOLDING THE I_DATA WHOSE EQUIVALENT ASCII IS TO BE SENT TO LCD DISPLAY
   SIGNAL DECODED_HEX : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');

   -- DELAY COUNTERS
   SIGNAL DELAY_COUNTER, PHASE_FLAG_DELAY_COUNTER : UNSIGNED(31 DOWNTO 0) := (OTHERS => '0');

   SIGNAL DELAY_COUNTER_RESET :STD_LOGIC := '0';
   SIGNAL TIME_OUT           :STD_LOGIC;
   SIGNAL TIME_OUT_1         :STD_LOGIC;
   signal INPUT_REGISTERED   :STD_LOGIC;
   SIGNAL RST_INPUT_REGISTERED:STD_LOGIC;
   SIGNAL RESET_1            :STD_LOGIC;

   SIGNAL RESET_INT : STD_LOGIC;

-- *** CALIBRATION LOGIC SIGNALS ********
    -- REGISTERS TO HOLD CAL. I_DATA FOR BUFFER LINE IN VDL-1 and VDL-2
   SIGNAL VDL1_BUF_CAL_DATA, VDL2_BUF_CAL_DATA : STD_LOGIC_VECTOR(15 DOWNTO 0);

   -- REGISTERS TO HOLD CAL. I_DATA FOR LATCH LINE IN VDL-2    and VDL-2
   SIGNAL VDL1_LAT_CAL_DATA, VDL2_LAT_CAL_DATA : STD_LOGIC_VECTOR(15 DOWNTO 0);

   -- REGISTERS TO HOLD CAL. I_DATA FOR VERTICAL REFERENCE RO-1 AND RO-2
   SIGNAL REF_RO1_CAL_DATA, REF_RO2_CAL_DATA  : STD_LOGIC_VECTOR(15 DOWNTO 0);

   -- REGISTERS TO HOLD CAL. I_DATA FOR VERTICAL RO-1 AND RO-2
   SIGNAL V_RO1_CAL_DATA, V_RO2_CAL_DATA      : STD_LOGIC_VECTOR(15 DOWNTO 0);

   -- FLAGs
   TYPE   PHASE_FLAG IS (PHASE_1, PHASE_2);   -- flag used for flipping digit '1' or '2'
   SIGNAL    CURR_PHASE_FLAG : PHASE_FLAG := PHASE_1;

-- **************************************************************** --
--                            ARCHITECTURE BODY BEGINS
-- **************************************************************** --
BEGIN
-- *** CSA ***
   RESET_1              <= RST_INPUT_REGISTERED or I_RESET;
   NEW_DATA_REGISTERED     <= INPUT_REGISTERED;
   DELAY_COUNTER_RESET     <= I_RESET OR RESET_INT;

   O_LCD_E              <= LCD_CLK;
   O_LCD_We_Re          <= '0';
   LCD_CLK              <= PRESCALER(16);
   T_OUT                <= TIME_OUT;
   T_OUT_1              <= TIME_OUT_1;

   -- DELAY COUNTERS USED TO CONTROL THE DURATION OF DISPLAY OF VARIOUS DATA ON LCD
   SEC_2_DELAY_COUNTER:
   PROCESS(LCD_CLK,I_RESET, CNT_RST)
          VARIABLE TMP : UNSIGNED(31 DOWNTO 0) := (OTHERS => '0');
      BEGIN
          IF I_RESET = '1' OR CNT_RST = '1' THEN
             SEC_2_CNT  <= (OTHERS => '0');
             TIME_OUT_1 <= '0';
          ELSIF LCD_CLK'EVENT AND LCD_CLK = '1' THEN
             TMP := SEC_2_CNT;
             TMP :=  TMP + 1;
             IF (TMP > 100) THEN
                TIME_OUT_1 <= '1';
             END IF;
```

```
            SEC_2_CNT <= TMP;
         END IF;
   END PROCESS;

   MORE_CNT:
   PROCESS(I_CLK,I_RESET, MORE_CNT_RST)
      VARIABlE TMP : INTEGER RANGE 0 TO 3 := 0;
   BEGIN
      IF I_RESET = '1' OR MORE_CNT_RST = '1' THEN
         MORE_COUNTER <= 0;
      ELSIF I_CLK'EVENT AND I_CLK = '1' THEN
         TMP := MORE_COUNTER;
         TMP :=  TMP + 1;
         IF TMP = 3 THEN     -- keeping O_MORE_DATA high for 3 cycles
            MORE_FLAG <= '1';
         ELSE
            MORE_FLAG <= '0';
         END IF;
         MORE_COUNTER <=  TMP;
      END IF;
   END PROCESS;

-- *** CONTROL_FSM
   CONTROL_STATE_TRANSITION:
   PROCESS(I_CLK, I_RESET)
   BEGIN
      IF I_RESET = '1' THEN
         CURR_CONT_STATE <= T_0;
      ELSIF I_CLK'EVENT AND I_CLK='1' THEN
         CURR_CONT_STATE <= NEXT_CONT_STATE;
      END IF;
   END PROCESS;

   -- CONTROL_FSM NEXT STATE ASSIGNMENT
   NEXT_CONT_STATE_ASSIGNMENT:
   PROCESS( CURR_CONT_STATE, NEXT_CONT_STATE, INPUT_REGISTERED,
            TIME_OUT, MORE_FLAG, TIME_OUT_1  )
   BEGIN
      CASE CURR_CONT_STATE IS
         WHEN T_0      => NEXT_CONT_STATE <= T_WAIT_A;
         WHEN T_WAIT_A   => IF (INPUT_REGISTERED = '1' ) THEN
                                  NEXT_CONT_STATE <= T_WAIT_B;
                            ELSE  NEXT_CONT_STATE <= T_WAIT_A;
                            END IF;

         WHEN T_WAIT_B   => IF (TIME_OUT_1 = '1' ) THEN
                                  NEXT_CONT_STATE <= T_DISPLAY;
                            ELSE  NEXT_CONT_STATE <= T_WAIT_B;
                            END IF;

         WHEN T_DISPLAY  => IF (TIME_OUT = '1') THEN
                                  NEXT_CONT_STATE <= T_TMP;
                            ELSE  NEXT_CONT_STATE <= T_DISPLAY;
                            END IF;
         WHEN T_TMP      => IF (MORE_FLAG = '1') THEN
                               NEXT_CONT_STATE <= T_REQUEST_A;
                            ELSE  NEXT_CONT_STATE <= T_TMP;
                            END IF;

         WHEN T_REQUEST_A =>   IF INPUT_REGISTERED = '1' THEN
                                     NEXT_CONT_STATE <= T_REQUEST_B;
                               ELSE  NEXT_CONT_STATE <= T_REQUEST_A;
                               END IF;

         WHEN T_REQUEST_B =>   IF TIME_OUT_1 = '1' THEN
                                     NEXT_CONT_STATE <= T_DISPLAY;
                               ELSE  NEXT_CONT_STATE <= T_REQUEST_B;
```

```
                                END IF;
         WHEN OTHERS      => NULL;
      END CASE;
   END PROCESS;

   -- OUTPUT ASSIGNMENT FOR CONTROL_FSM
   CONT_STATES_OUTPUT_ASSIGNMENTS:
   PROCESS(CURR_CONT_STATE)
   BEGIN

      O_MORE_DATA         <= '0';
      CNT_RST                <= '1';
      MORE_CNT_RST        <= '1';
      RST_INPUT_REGISTERED <= '0';
      RESET_INT           <= '0';

      CASE CURR_CONT_STATE IS

         WHEN T_0     => CURR_FLAG <= WAIT_FLAG;    RESET_INT <= '1';
         WHEN T_WAIT_A=> CURR_FLAG <= WAIT_FLAG;    RESET_INT <= '1';
         WHEN T_WAIT_B=> CURR_FLAG <= WAIT_FLAG; CNT_RST     <= '0';

         WHEN T_DISPLAY =>   CURR_FLAG<= DISP_FLAG;
                          CNT_RST    <= '0';  -- KEEP TIME_OUT_1 ON LONGER FOR LCD DISPLAY
OBSERVATION
                          RST_INPUT_REGISTERED <= '1';

         WHEN T_TMP       => CURR_FLAG     <= REQ_FLAG;
                          O_MORE_DATA  <= '1';   -- kept for about 3 cycles...
                          MORE_CNT_RST <= '0';

         WHEN T_REQUEST_A=> CURR_FLAG     <= REQ_FLAG;
                          MORE_CNT_RST<= '0';
         WHEN T_REQUEST_B=>     CURR_FLAG     <= REQ_FLAG;
                          CNT_RST       <= '0';
      END CASE;
   END PROCESS;

-- CLOCKED PROCESS TO DECREMENT INDEX OF COORDINATESAFTER SOME LONGER PERIOD FOR DISPLAY
-- DELAY COUNTER USED TO CONTROL AMOUNT OF TIME individual TAP row READINGS SHOULD STAY ON DISPLAY
   INDEX_DETERMINATION:
   PROCESS (LCD_CLK, DELAY_COUNTER_RESET, DELAY_COUNTER)
      VARIABLE V_DELAY_COUNTER : UNSIGNED(31 DOWNTO 0) := (OTHERS =>'0');
      VARIABLE V_PHASE_FLAG_DELAY_COUNTER : UNSIGNED(31 DOWNTO 0) := (OTHERS =>'0');
   BEGIN
      IF (DELAY_COUNTER_RESET = '1') THEN
         DELAY_COUNTER          <= (OTHERS => '0');
         ROUND_COUNTER       <= 0;
         CURR_PHASE_FLAG        <= PHASE_1;
         PHASE_FLAG_DELAY_COUNTER<= (OTHERS => '0');

      ELSIF ( LCD_CLK'EVENT AND LCD_CLK= '1' ) THEN
         TIME_OUT <= '0';

         V_DELAY_COUNTER := DELAY_COUNTER;
         V_DELAY_COUNTER := V_DELAY_COUNTER + 1;

         V_PHASE_FLAG_DELAY_COUNTER := PHASE_FLAG_DELAY_COUNTER;
         V_PHASE_FLAG_DELAY_COUNTER:= V_PHASE_FLAG_DELAY_COUNTER + 1;

         -- CHANGING OF CURR_PHASE_FLAG FOR CALIBRATION DATA DISPLAY
         IF (V_PHASE_FLAG_DELAY_COUNTER > 500) THEN
            IF CURR_PHASE_FLAG = PHASE_1 THEN
               CURR_PHASE_FLAG <= PHASE_2;
            ELSE
               CURR_PHASE_FLAG <= PHASE_1;
            END IF;
            V_PHASE_FLAG_DELAY_COUNTER := (OTHERS => '0');
         END IF;
         PHASE_FLAG_DELAY_COUNTER <= V_PHASE_FLAG_DELAY_COUNTER;

         -- CONTROL SWITCHING OF NEXT TAP READING
         IF (V_DELAY_COUNTER > 1000) THEN
            TIME_OUT <= '1';
            IF (I_CAL_MODE = '1') THEN
               IF ROUND_COUNTER < 24 THEN
                     ROUND_COUNTER <= ROUND_COUNTER + 1;
               ELSE  ROUND_COUNTER <= 0;
               END IF;
            ELSE
               IF ROUND_COUNTER < 99 THEN
                     ROUND_COUNTER <= ROUND_COUNTER + 1;
               ELSE  ROUND_COUNTER <= 0;
               END IF;
            END IF;

            V_DELAY_COUNTER := (OTHERS =>'0');
         END IF;
         DELAY_COUNTER <= V_DELAY_COUNTER;
      END IF;
   END PROCESS;

-- NEXT STATE ASSIGNMENT PROCESS
   NEXT_STATE_ASSIGNMENT:
   PROCESS(CURR_LCD_STATE, CURR_FLAG, CURR_PHASE_FLAG, I_CAL_MODE)
   BEGIN
      CASE CURR_LCD_STATE IS
         WHEN S0 =>  NEXT_LCD_STATE <= S1;

         WHEN S1 =>  NEXT_LCD_STATE <= S2;
         WHEN S2 =>  NEXT_LCD_STATE <= S3;
         WHEN S3 =>  NEXT_LCD_STATE <= S4;
         WHEN S4 =>  NEXT_LCD_STATE <= S5;
         WHEN S5 =>  IF (CURR_FLAG = WAIT_FLAG) THEN
                        NEXT_LCD_STATE <= S_W;
                     ELSIF (CURR_FLAG = REQ_FLAG) THEN
                        NEXT_LCD_STATE <= S4_1;
                     ELSE
                        IF (I_CAL_MODE = '0') THEN
                              NEXT_LCD_STATE <= S_DIGIT_1;
                        ELSE  NEXT_LCD_STATE <= S_D_CAL;
                        END IF;
                     END IF;

         -- *** CALIBRATION DATA DISPLAY
         -- STATE SEQUENCE FOR DISPLAYING "RD--:H----V----"
         --                        P   :H----V----"         P: phase
         WHEN S_D_CAL    => NEXT_LCD_STATE <= S_DIGIT_1;
         WHEN S_DIGIT_1  => NEXT_LCD_STATE <= S_DIGIT_2;
         WHEN S_DIGIT_2  => NEXT_LCD_STATE <= S_DOTS;

         WHEN S_DOTS  => IF I_CAL_MODE = '0' THEN
                        NEXT_LCD_STATE <= S_CRS_1;
                     ELSE  NEXT_LCD_STATE <= S_SPACE_1;
                     END IF;

         WHEN S_SPACE_1 => NEXT_LCD_STATE  <= S_H1_CAL;
         WHEN S_H1_CAL=> IF (CURR_PHASE_FLAG = PHASE_1) THEN
                              NEXT_LCD_STATE <= S_VDL1_BUF_CNT1;
                        ELSE     NEXT_LCD_STATE <= S_VDL1_LAT_CNT1;
                        END IF;

         -- DISPLAYING HOR. CAL. BUFFER DATA
         WHEN S_VDL1_BUF_CNT1 => NEXT_LCD_STATE  <= S_VDL1_BUF_CNT2;
   WHEN S_VDL1_BUF_CNT2 => NEXT_LCD_STATE  <= S_VDL1_BUF_CNT3;
   WHEN S_VDL1_BUF_CNT3 => NEXT_LCD_STATE  <= S_VDL1_BUF_CNT4;
   WHEN S_VDL1_BUF_CNT4 => NEXT_LCD_STATE  <= S_SPACE_0;

   WHEN S_SPACE_0        => NEXT_LCD_STATE  <= S_V1_CAL;

   -- DISPLAYING HOR. CAL. LATCH DATA
   WHEN S_VDL1_LAT_CNT1 => NEXT_LCD_STATE  <= S_VDL1_LAT_CNT2;
   WHEN S_VDL1_LAT_CNT2 => NEXT_LCD_STATE  <= S_VDL1_LAT_CNT3;
   WHEN S_VDL1_LAT_CNT3 => NEXT_LCD_STATE  <= S_VDL1_LAT_CNT4;
   WHEN S_VDL1_LAT_CNT4 => NEXT_LCD_STATE  <= S_SPACE_0;


   WHEN S_V1_CAL   => IF (CURR_PHASE_FLAG = PHASE_1) THEN
                            NEXT_LCD_STATE <= S_VREF1_CNT1;
                      ELSE      NEXT_LCD_STATE <= S_V_RO1_CNT1;
                      END IF;

   WHEN S_VREF1_CNT1  => NEXT_LCD_STATE <= S_VREF1_CNT2;
   WHEN S_VREF1_CNT2  => NEXT_LCD_STATE <= S_VREF1_CNT3;
   WHEN S_VREF1_CNT3  => NEXT_LCD_STATE <= S_VREF1_CNT4;
   WHEN S_VREF1_CNT4  => NEXT_LCD_STATE <= S_NEXT_LINE;

-- DISPLAYING DATA FOR VERTICAL CAL LATCH LINE (RO-1)
   WHEN S_V_RO1_CNT1  => NEXT_LCD_STATE <= S_V_RO1_CNT2;
   WHEN S_V_RO1_CNT2  => NEXT_LCD_STATE <= S_V_RO1_CNT3;
   WHEN S_V_RO1_CNT3  => NEXT_LCD_STATE <= S_V_RO1_CNT4;
   WHEN S_V_RO1_CNT4  => NEXT_LCD_STATE <= S_NEXT_LINE;


   -- CAL. DATA SECOND LINE DISPLAY
   WHEN S_PHASE_DIGIT=> NEXT_LCD_STATE  <= S_SPACE_2;
   WHEN S_SPACE_2  => NEXT_LCD_STATE  <= S_SPACE_3;
   WHEN S_SPACE_3  => NEXT_LCD_STATE  <= S_SPACE_4;
   WHEN S_SPACE_4  => NEXT_LCD_STATE  <= S_SPACE_5;
   WHEN S_SPACE_5  => NEXT_LCD_STATE  <= S_H2_CAL;
   WHEN S_SPACE_6     => NEXT_LCD_STATE  <= S_V2_CAL;

   WHEN S_H2_CAL   => IF (CURR_PHASE_FLAG = PHASE_1) THEN
                            NEXT_LCD_STATE <= S_VDL2_BUF_CNT1;
                      ELSE      NEXT_LCD_STATE <= S_VDL2_LAT_CNT1;
                      END IF;

   WHEN S_VDL2_BUF_CNT1 => NEXT_LCD_STATE  <= S_VDL2_BUF_CNT2;
   WHEN S_VDL2_BUF_CNT2 => NEXT_LCD_STATE  <= S_VDL2_BUF_CNT3;
   WHEN S_VDL2_BUF_CNT3 => NEXT_LCD_STATE  <= S_VDL2_BUF_CNT4;
   WHEN S_VDL2_BUF_CNT4 => NEXT_LCD_STATE  <= S_SPACE_6;

   WHEN S_VDL2_LAT_CNT1 => NEXT_LCD_STATE  <= S_VDL2_LAT_CNT2;
   WHEN S_VDL2_LAT_CNT2 => NEXT_LCD_STATE  <= S_VDL2_LAT_CNT3;
   WHEN S_VDL2_LAT_CNT3 => NEXT_LCD_STATE  <= S_VDL2_LAT_CNT4;
   WHEN S_VDL2_LAT_CNT4 => NEXT_LCD_STATE  <= S_SPACE_6;

   WHEN S_V2_CAL      => IF (CURR_PHASE_FLAG = PHASE_1) THEN
                               NEXT_LCD_STATE <= S_VREF2_CNT1;
                         ELSE      NEXT_LCD_STATE <= S_V_RO2_CNT1;
                         END IF;

   WHEN S_VREF2_CNT1  => NEXT_LCD_STATE <= S_VREF2_CNT2;
   WHEN S_VREF2_CNT2  => NEXT_LCD_STATE <= S_VREF2_CNT3;
   WHEN S_VREF2_CNT3  => NEXT_LCD_STATE <= S_VREF2_CNT4;

   WHEN S_VREF2_CNT4  => NEXT_LCD_STATE <= S5;

   WHEN S_V_RO2_CNT1  => NEXT_LCD_STATE <= S_V_RO2_CNT2;
   WHEN S_V_RO2_CNT2  => NEXT_LCD_STATE <= S_V_RO2_CNT3;
WHEN S_V_RO2_CNT3   => NEXT_LCD_STATE <= S_V_RO2_CNT4;

WHEN S_V_RO2_CNT4   => NEXT_LCD_STATE <= S5;

-- STATE SEQUENCE FOR DISPLAYING "WAITING..."
WHEN S_W  => NEXT_LCD_STATE <= S_A;
WHEN S_A  => NEXT_LCD_STATE <= S_I0;
WHEN S_I0 => NEXT_LCD_STATE <= S_T;
WHEN S_T     => IF CURR_FLAG = WAIT_FLAG THEN
                      NEXT_LCD_STATE <= S_I1;
                ELSE  NEXT_LCD_STATE <= S_C;
                END IF;
WHEN S_I1   => NEXT_LCD_STATE <= S_N;

WHEN S_N     => NEXT_LCD_STATE <= S_G;
WHEN S_G     => NEXT_LCD_STATE <= S_DOT0;
WHEN S_DOT0 => NEXT_LCD_STATE <= S_DOT1;

WHEN S_DOT1 => NEXT_LCD_STATE <= S_DOT2;
WHEN S_DOT2 => NEXT_LCD_STATE <= S5;

-- STATES FOR F, C, H IN "FETCHING...". THE REST IS SHARED WITH "WAITING..."
WHEN S4_1 => NEXT_LCD_STATE <= S5_1;
WHEN S5_1 => NEXT_LCD_STATE <= S_F;
WHEN S_F     => NEXT_LCD_STATE <= S_E;
WHEN S_E    => NEXT_LCD_STATE <= S_T;
WHEN S_C     => NEXT_LCD_STATE <= S_H;
WHEN S_H     => NEXT_LCD_STATE <= S_I1;

WHEN S_CRS_1 => NEXT_LCD_STATE <= S_CRS_2;
WHEN S_CRS_2 => NEXT_LCD_STATE <= S_CRS_3;
WHEN S_CRS_3 => NEXT_LCD_STATE <= S_MINUS;

WHEN S_MINUS =>  NEXT_LCD_STATE <= S_LSB_24;

WHEN S_LSB_24=>  NEXT_LCD_STATE <= S_LSB_23;
WHEN S_LSB_23=>  NEXT_LCD_STATE <= S_LSB_22;
WHEN S_LSB_22=>  NEXT_LCD_STATE <= S_LSB_21;
WHEN S_LSB_21=>  NEXT_LCD_STATE <= S_LSB_20;
WHEN S_LSB_20=>  NEXT_LCD_STATE <= S_LSB_19;
WHEN S_LSB_19=>  NEXT_LCD_STATE <= S_LSB_18;
WHEN S_LSB_18=>  NEXT_LCD_STATE <= S_LSB_17;
WHEN S_LSB_17=>  NEXT_LCD_STATE <= S_LSB_16;
WHEN S_LSB_16=>  NEXT_LCD_STATE <= S_NEXT_LINE;

WHEN S_NEXT_LINE    =>  IF I_CAL_MODE = '0' THEN
                          NEXT_LCD_STATE   <= S_LSB_15;
                    ELSE      NEXT_LCD_STATE   <= S_PHASE_DIGIT;
                    END IF;

WHEN S_LSB_15=>  NEXT_LCD_STATE  <= S_LSB_14;
WHEN S_LSB_14=>  NEXT_LCD_STATE  <= S_LSB_13;
WHEN S_LSB_13=>  NEXT_LCD_STATE  <= S_LSB_12;
WHEN S_LSB_12=>  NEXT_LCD_STATE  <= S_LSB_11;
WHEN S_LSB_11=>  NEXT_LCD_STATE  <= S_LSB_10;
WHEN S_LSB_10    =>  NEXT_LCD_STATE  <= S_LSB_9;
WHEN S_LSB_9     =>  NEXT_LCD_STATE  <= S_LSB_8;
WHEN S_LSB_8     =>  NEXT_LCD_STATE  <= S_LSB_7;
WHEN S_LSB_7     =>  NEXT_LCD_STATE  <= S_LSB_6;
WHEN S_LSB_6     =>  NEXT_LCD_STATE  <= S_LSB_5;
WHEN S_LSB_5     =>  NEXT_LCD_STATE  <= S_LSB_4;
WHEN S_LSB_4     =>  NEXT_LCD_STATE  <= S_LSB_3;
WHEN S_LSB_3     =>  NEXT_LCD_STATE  <= S_LSB_2;
WHEN S_LSB_2     =>  NEXT_LCD_STATE  <= S_LSB_1;
WHEN S_LSB_1     =>  NEXT_LCD_STATE  <= S_LSB_0;
--WHEN S_LSB_0   =>  NEXT_LCD_STATE  <= S5;
WHEN OTHERS  =>  NEXT_LCD_STATE  <= S5;
```

```
    END CASE;
   END PROCESS;

   -- STATE OUTPUT ASSIGNMENT STATES
   LCD_DATA_ASSIGNMENTS:
   PROCESS( CURR_LCD_STATE, DECODED_ASCII)
   BEGIN
      O_LCD_RS <= '0';

      case CURR_LCD_STATE is
         -- COMMANDS
         WHEN S0 => O_LCD_DATA<=x"00"; -- D7 ......... D0
         WHEN S1 => O_LCD_DATA<=x"3c"; -- 00111100 setting 8-bit DATA length, 2 lines of 5X10
display dots
         WHEN S2 => O_LCD_DATA<=x"0C"; -- 00001100 sets entire display on, but no cursor and no
blinking
         WHEN S3 => O_LCD_DATA<=x"60"; -- 01100000 sets cursor move direction to Right and
display shift
         WHEN S4 => O_LCD_DATA<=x"01"; -- 00000001 clearing the entire display
         WHEN S5 => O_LCD_DATA<=x"80";


         -- setting address of DDRAM to $00   0000 0001

         WHEN S4_1 => O_LCD_DATA<=x"01"; -- clearing the entire display00000001

         WHEN S5_1 => O_LCD_DATA<=x"80"; -- setting address of DDRAM to $00   0000 0001

         WHEN S_NEXT_LINE => O_LCD_DATA <= x"c0";


         -- CALIBRATION STATES OUTPUTS

         WHEN S_DOTS  => O_LCD_DATA <= x"3A";   O_LCD_RS<='1';

         -- THE HEX VALUES ARE ASCII CORRESPONDING TO EACH CHARACTER
         WHEN S_E        => O_LCD_DATA <= x"45";      O_LCD_RS<='1';
         WHEN S_D_CAL    => O_LCD_DATA <= x"44";      O_LCD_RS<='1';
         WHEN S_MINUS    => O_LCD_DATA <= x"2D";   O_LCD_RS<='1';
         WHEN S_W        => O_LCD_DATA <= x"57";   O_LCD_RS<='1';
         WHEN S_A        => O_LCD_DATA <= x"41";   O_LCD_RS<='1';
         WHEN S_I0       => O_LCD_DATA <= x"49";   O_LCD_RS<='1';
         when S_I1    => O_LCD_DATA <= x"49";   O_LCD_RS<='1';
         WHEN S_T        => O_LCD_DATA <= x"54";   O_LCD_RS<='1';
         WHEN S_N        => O_LCD_DATA <= x"4e";   O_LCD_RS<='1';
         WHEN S_G        => O_LCD_DATA <= x"47";   O_LCD_RS<='1';
         WHEN S_DOT0     => O_LCD_DATA <= x"2e";   O_LCD_RS<='1';
         when S_DOT1     => O_LCD_DATA <= x"2e";   O_LCD_RS<='1';
         when S_DOT2  => O_LCD_DATA <= x"2e";   O_LCD_RS<='1';
         WHEN S_F        => O_LCD_DATA <= x"46";   O_LCD_RS<='1';
         WHEN S_C        => O_LCD_DATA <= x"43";   O_LCD_RS<='1';
         WHEN S_H  |S_H1_CAL | S_H2_CAL    => O_LCD_DATA <= x"48";   O_LCD_RS<='1';

         WHEN S_V1_CAL | S_V2_CAL  => O_LCD_DATA <= x"56";   O_LCD_RS<='1';

         WHEN S_SPACE_0 | S_SPACE_1 | S_SPACE_2 |
              S_SPACE_3 | S_SPACE_4 | S_SPACE_5 |  S_SPACE_6
              => O_LCD_DATA <= x"20";   O_LCD_RS<='1';

         -- DISPLAYING DECODED DATA FOR OTHER STATES (COARSE AND FINE )
         WHEN OTHERS   => O_LCD_DATA <= DECODED_ASCII;   O_LCD_RS<='1';
      END CASE;
   END PROCESS;

-- PROCESS FOR ASSIGNING THE HEX DIGITS AT APPROPRIATE STATES
   -- UPDATE DECODED_HEX AT RISING EDGE, WHILE SENDING IT TO LCD AT FALLING EDGE
   PROCESS( LCD_CLK,CURR_LCD_STATE, CURR_PHASE_FLAG, ROUND_COUNTER)      -- , CURR_MATRIX_FLAG
```

```
      VARIABLE  COARSE_INTEGER : integer := 0;
   BEGIN
      IF (LCD_CLK'EVENT AND LCD_CLK ='1') then
         CASE CURR_LCD_STATE IS
            WHEN S_CRS_1 => DECODED_HEX <= MSB_REG  (11 DOWNTO 8);
            WHEN S_CRS_2 => DECODED_HEX <= MSB_REG  (7 DOWNTO 4);
            WHEN S_CRS_3 => DECODED_HEX <= MSB_REG (3 DOWNTO 0);

            WHEN S_LSB_0  => DECODED_HEX <= LSB_REG(3  DOWNTO 0 );
            WHEN S_LSB_1  => DECODED_HEX <= LSB_REG(7  DOWNTO 4 );
            WHEN S_LSB_2  => DECODED_HEX <= LSB_REG(11 DOWNTO 8 );
            WHEN S_LSB_3  => DECODED_HEX <= LSB_REG(15 DOWNTO 12);
            WHEN S_LSB_4  => DECODED_HEX <= LSB_REG(19 DOWNTO 16);
            WHEN S_LSB_5  => DECODED_HEX <= LSB_REG(23 DOWNTO 20);
            WHEN S_LSB_6  => DECODED_HEX <= LSB_REG(27 DOWNTO 24);
            WHEN S_LSB_7  => DECODED_HEX <= LSB_REG(31 DOWNTO 28);
            WHEN S_LSB_8  => DECODED_HEX <= LSB_REG(35 DOWNTO 32);
            WHEN S_LSB_9  => DECODED_HEX <= LSB_REG(39 DOWNTO 36);
            WHEN S_LSB_10 => DECODED_HEX <= LSB_REG(43 DOWNTO 40);
            WHEN S_LSB_11 => DECODED_HEX <= LSB_REG(47 DOWNTO 44);
            WHEN S_LSB_12 => DECODED_HEX <= LSB_REG(51 DOWNTO 48);
            WHEN S_LSB_13 => DECODED_HEX <= LSB_REG(55 DOWNTO 52);
            WHEN S_LSB_14 => DECODED_HEX <= LSB_REG(59 DOWNTO 56);
            WHEN S_LSB_15 => DECODED_HEX <= LSB_REG(63 DOWNTO 60);
            WHEN S_LSB_16 => DECODED_HEX <= LSB_REG(67 DOWNTO 64);
            WHEN S_LSB_17 => DECODED_HEX <= LSB_REG(71 DOWNTO 68);
            WHEN S_LSB_18 => DECODED_HEX <= LSB_REG(75 DOWNTO 72);
            WHEN S_LSB_19 => DECODED_HEX <= LSB_REG(79 DOWNTO 76);
            WHEN S_LSB_20 => DECODED_HEX <= LSB_REG(83 DOWNTO 80);
            WHEN S_LSB_21 => DECODED_HEX <= LSB_REG(87 DOWNTO 84);
            WHEN S_LSB_22 => DECODED_HEX <= LSB_REG(91 DOWNTO 88);
            WHEN S_LSB_23 => DECODED_HEX <= LSB_REG(95 DOWNTO 92);
            WHEN S_LSB_24 => DECODED_HEX <= LSB_REG(99 DOWNTO 96);


            -- CALIBRATION STATES

-- VDL-1 I_DATA
            -- BUFFER LINE

            WHEN S_VDL1_BUF_CNT1   => DECODED_HEX <= VDL1_BUF_CAL_DATA(15 DOWNTO 12);
            WHEN S_VDL1_BUF_CNT2   => DECODED_HEX <= VDL1_BUF_CAL_DATA(11 DOWNTO 8);
            WHEN S_VDL1_BUF_CNT3   => DECODED_HEX <= VDL1_BUF_CAL_DATA(7 DOWNTO 4);
            WHEN S_VDL1_BUF_CNT4   => DECODED_HEX <= VDL1_BUF_CAL_DATA(3 DOWNTO 0);


            -- LATCH LINE
            WHEN S_VDL1_LAT_CNT1   => DECODED_HEX <= VDL1_LAT_CAL_DATA(15 DOWNTO 12);
            WHEN S_VDL1_LAT_CNT2   => DECODED_HEX <= VDL1_LAT_CAL_DATA(11 DOWNTO 8);
            WHEN S_VDL1_LAT_CNT3   => DECODED_HEX <= VDL1_LAT_CAL_DATA(7 DOWNTO 4);
            WHEN S_VDL1_LAT_CNT4   => DECODED_HEX <= VDL1_LAT_CAL_DATA(3 DOWNTO 0);



-- VDL-2 I_DATA
            -- BUFFER LINE

            WHEN S_VDL2_BUF_CNT1   => DECODED_HEX <= VDL2_BUF_CAL_DATA(15 DOWNTO 12);
            WHEN S_VDL2_BUF_CNT2   => DECODED_HEX <= VDL2_BUF_CAL_DATA(11 DOWNTO 8);
            WHEN S_VDL2_BUF_CNT3   => DECODED_HEX <= VDL2_BUF_CAL_DATA(7 DOWNTO 4);
            WHEN S_VDL2_BUF_CNT4   => DECODED_HEX <= VDL2_BUF_CAL_DATA(3 DOWNTO 0);

            -- LATCH LINE
            WHEN S_VDL2_LAT_CNT1   => DECODED_HEX <= VDL2_LAT_CAL_DATA(15 DOWNTO 12);
            WHEN S_VDL2_LAT_CNT2   => DECODED_HEX <= VDL2_LAT_CAL_DATA(11 DOWNTO 8);
            WHEN S_VDL2_LAT_CNT3   => DECODED_HEX <= VDL2_LAT_CAL_DATA(7 DOWNTO 4);
            WHEN S_VDL2_LAT_CNT4   => DECODED_HEX <= VDL2_LAT_CAL_DATA(3 DOWNTO 0);
```

```
-- VERTiCAL LINE-1 I_DATA
             WHEN S_VREF1_CNT1   => DECODED_HEX <= REF_RO1_CAL_DATA(15 DOWNTO 12);
             WHEN S_VREF1_CNT2   => DECODED_HEX <= REF_RO1_CAL_DATA(11 DOWNTO 8);
             WHEN S_VREF1_CNT3   => DECODED_HEX <= REF_RO1_CAL_DATA(7 DOWNTO 4);
             WHEN S_VREF1_CNT4   => DECODED_HEX <= REF_RO1_CAL_DATA(3 DOWNTO 0);

             WHEN S_V_RO1_CNT1   => DECODED_HEX <= V_RO1_CAL_DATA(15 DOWNTO 12);
             WHEN S_V_RO1_CNT2   => DECODED_HEX <= V_RO1_CAL_DATA(11 DOWNTO 8);
             WHEN S_V_RO1_CNT3   => DECODED_HEX <= V_RO1_CAL_DATA(7 DOWNTO 4);
             WHEN S_V_RO1_CNT4   => DECODED_HEX <= V_RO1_CAL_DATA(3 DOWNTO 0);


-- VERTiCAL LINE-2 I_DATA
             WHEN S_VREF2_CNT1   => DECODED_HEX <= REF_RO2_CAL_DATA(15 DOWNTO 12);
             WHEN S_VREF2_CNT2   => DECODED_HEX <= REF_RO2_CAL_DATA(11 DOWNTO 8);
             WHEN S_VREF2_CNT3   => DECODED_HEX <= REF_RO2_CAL_DATA(7 DOWNTO 4);
             WHEN S_VREF2_CNT4   => DECODED_HEX <= REF_RO2_CAL_DATA(3 DOWNTO 0);

             WHEN S_V_RO2_CNT1   => DECODED_HEX <= V_RO2_CAL_DATA(15 DOWNTO 12);
             WHEN S_V_RO2_CNT2   => DECODED_HEX <= V_RO2_CAL_DATA(11 DOWNTO 8);
             WHEN S_V_RO2_CNT3   => DECODED_HEX <= V_RO2_CAL_DATA(7 DOWNTO 4);
             WHEN S_V_RO2_CNT4   => DECODED_HEX <= V_RO2_CAL_DATA(3 DOWNTO 0);
             WHEN S_DIGIT_1      =>
                IF  (ROUND_COUNTER < 10)                           THEN   DECODED_HEX <=
(OTHERS=>'0');
                ELSIF (ROUND_COUNTER > 9  AND ROUND_COUNTER < 20)    THEN   DECODED_HEX <= "0001";
                ELSIF (ROUND_COUNTER > 19 AND ROUND_COUNTER < 30)    THEN   DECODED_HEX <= "0010";
                ELSIF (ROUND_COUNTER > 29 AND ROUND_COUNTER < 40)    THEN   DECODED_HEX <= "0011";
                ELSIF (ROUND_COUNTER > 39 AND ROUND_COUNTER < 50)    THEN   DECODED_HEX <= "0100";
                ELSIF (ROUND_COUNTER > 49 AND ROUND_COUNTER < 60)    THEN   DECODED_HEX <= "0101";
                ELSIF (ROUND_COUNTER > 59 AND ROUND_COUNTER < 70)    THEN   DECODED_HEX <= "0110";
                ELSIF (ROUND_COUNTER > 69 AND ROUND_COUNTER < 80)    THEN   DECODED_HEX <= "0111";
                ELSIF (ROUND_COUNTER > 79 AND ROUND_COUNTER < 90)    THEN   DECODED_HEX <= "1000";
                ELSE                                                 DECODED_HEX <= "1001";
                END IF;

             WHEN S_DIGIT_2   =>
                IF  (ROUND_COUNTER < 10) THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER, 4);
                ELSIF (ROUND_COUNTER > 9 AND ROUND_COUNTER < 20) THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-10, 4);
                ELSIF (ROUND_COUNTER > 19 AND ROUND_COUNTER < 30)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-20, 4);
                ELSIF (ROUND_COUNTER > 29 AND ROUND_COUNTER < 40)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-30, 4);
                ELSIF (ROUND_COUNTER > 39 AND ROUND_COUNTER < 50)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-40, 4);
                ELSIF (ROUND_COUNTER > 49 AND ROUND_COUNTER < 60)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-50, 4);
                ELSIF (ROUND_COUNTER > 59 AND ROUND_COUNTER < 70)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-60, 4);
                ELSIF (ROUND_COUNTER > 69 AND ROUND_COUNTER < 80)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-70, 4);
                ELSIF (ROUND_COUNTER > 79 AND ROUND_COUNTER < 90)    THEN
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-80, 4);
                ELSE
                   DECODED_HEX <= CONV_STD_LOGIC_VECTOR(ROUND_COUNTER-90, 4);
                END IF;

             WHEN S_PHASE_DIGIT=> IF    (CURR_PHASE_FLAG = PHASE_1) THEN
                                    DECODED_HEX <= "0001"; -- 1
                             ELSE      DECODED_HEX <= "0010"; -- 2
                             END IF;
             WHEN OTHERS      => DECODED_HEX <= X"F";
          END CASE;
       END IF;
END PROCESS;

-- INPUTTING I_DATA INTO INTERNAL REGISTERS
INTERNAL_REGISTERS:
PROCESS( I_CLK, I_RESET, I_Ld_0, I_Ld_1, I_Ld_2, I_Ld_3)
BEGIN
   IF I_RESET = '1' THEN
      REG0 <= (OTHERS =>'0');
      REG1 <= (OTHERS =>'0');
      REG2 <= (OTHERS =>'0');
      REG3 <= (OTHERS =>'0');
   ELSIF I_CLK'EVENT AND I_CLK = '1' THEN
      IF (I_Ld_0 = '1') THEN
         REG0 <= I_DATA;
      END IF;
      IF (I_Ld_1 = '1') THEN
         REG1 <= I_DATA;
      END IF;
      IF (I_Ld_2 = '1') THEN
         REG2 <= I_DATA;
      END IF;
      IF (I_Ld_3 = '1') THEN
         REG3 <= I_DATA;
      END IF;
   END IF;
END PROCESS;

-- CALIBRATION I_DATA REGISTERS
INTERNAL_CAL_REGISTERS:
PROCESS( I_CLK, I_RESET, I_Ld_CAL_BDATA, I_Ld_CAL_LDATA)
BEGIN
   IF I_RESET = '1' THEN
      VDL1_BUF_CAL_DATA <= (OTHERS =>'0');
      VDL2_BUF_CAL_DATA <= (OTHERS =>'0');
      VDL1_LAT_CAL_DATA <= (OTHERS =>'0');

      VDL2_LAT_CAL_DATA <= (OTHERS =>'0');

   ELSIF I_CLK'EVENT AND I_CLK = '1' THEN
         -- ALL INPUT DATA LINES are SHARED BY INTERNAL REGISTERS
         -- Depending on Cont. Inputs, proper DATA from RAMs is loaded into internal registers

      IF (I_Ld_CAL_BDATA = '1') THEN
         VDL1_BUF_CAL_DATA <=HOR_CAL_DATA_1;

         VDL2_BUF_CAL_DATA <=HOR_CAL_DATA_2;
         REF_RO1_CAL_DATA<=VER_CAL_DATA_1;
         REF_RO2_CAL_DATA<=VER_CAL_DATA_2;
      END IF;
      IF (I_Ld_CAL_LDATA = '1') THEN
         VDL1_LAT_CAL_DATA <=HOR_CAL_DATA_1;
         VDL2_LAT_CAL_DATA <=HOR_CAL_DATA_2;
         V_RO1_CAL_DATA  <=VER_CAL_DATA_1;
         V_RO2_CAL_DATA  <=VER_CAL_DATA_2;

      END IF;
   END IF;
END PROCESS;

-- LOADING DATA_REGISTERS WITH INPUT DATA
PROCESS(I_CLK, I_RESET, I_DATA_RDY)
BEGIN
   IF I_RESET = '1' THEN
      LSB_REG <=(OTHERS => '0');
      MSB_REG <=(OTHERS => '0');
   ELSIF I_CLK'EVENT AND I_CLK = '1' THEN
      IF (I_DATA_RDY = '1') THEN
```

```
            LSB_REG <= REG3(3 DOWNTO 0) & REG2 & REG1 & REG0;
            MSB_REG <= REG3(15 DOWNTO 4);
         END IF;
      END IF;
   END PROCESS;

   -- PROCESS TO INDICATE NEW DATA BEING RESGISTERED.
   process(I_CLK, RESET_1)
   begin
         if RESET_1 = '1' then
               INPUT_REGISTERED <= '0';
         elsif I_CLK'event and I_CLK = '1' then
               if (I_DATA_RDY = '1' OR I_CALDATA_RDY='1') then
                  INPUT_REGISTERED <= '1';
               end if;
         end if;
   end process;

   -- STATE TRANSITION
   STATE_TRANSITION:
   PROCESS(LCD_CLK, I_RESET) BEGIN
      IF (I_RESET='1') THEN
         CURR_LCD_STATE<=s0;
      ELSIF (LCD_CLK'EVENT AND LCD_CLK='0') THEN-- CHANGING STATE AT FALLING EDGE OF INTERNAL
CLOCK
         CURR_LCD_STATE<=NEXT_LCD_STATE;
      END IF;
   END PROCESS;

   -- PROCESS FOR A SCALER COUNTER USED FOR CLOCKING OF LCD
   PRESCALER_UNIT:
   PROCESS(I_CLK, I_RESET) BEGIN
      IF (I_RESET = '1') THEN
         PRESCALER <= (OTHERS => '0');
      ELSIF (I_CLK'event and I_CLK='1') then
         PRESCALER<=PRESCALER+'1';
      END IF;
   END PROCESS;

   -- COMBINATIONAL PROCESS OR LUT FOR ASCII REPRESENTATION OF THE HEX DIGITS;
   HEX_2_ASCII_LUT:
   PROCESS( DECODED_HEX )
   BEGIN
      CASE DECODED_HEX IS
         WHEN x"0" => DECODED_ASCII <= X"30";
         WHEN x"1" => DECODED_ASCII <= X"31";
         WHEN x"2" => DECODED_ASCII <= X"32";
         WHEN x"3" => DECODED_ASCII <= X"33";
         WHEN x"4" => DECODED_ASCII <= X"34";
         WHEN x"5" => DECODED_ASCII <= X"35";
         WHEN x"6" => DECODED_ASCII <= X"36";
         WHEN x"7" => DECODED_ASCII <= X"37";
         WHEN x"8" => DECODED_ASCII <= X"38";
         WHEN x"9" => DECODED_ASCII <= X"39";
         WHEN x"A" => DECODED_ASCII <= X"41";
         WHEN x"B" => DECODED_ASCII <= X"42";
         WHEN x"C" => DECODED_ASCII <= X"43";
         WHEN x"D" => DECODED_ASCII <= X"44";
         WHEN x"E" => DECODED_ASCII <= X"45";
         WHEN x"F" => DECODED_ASCII <= X"46";
         WHEN OTHERS => DECODED_ASCII <= x"00";
      END CASE;
   END PROCESS;
end Behavioral;
```

APPENDIX-IV

C++ Source Code For PRpG Emulation Presented in Chapter 6

```
/* THIS C++ FILE SERVES TO VERIFY A GIVEN T_CLK TO BE USABLE IN THE PULSE GENERATOR FOR RANDOMNESS
*/
/*
  ALGORITHM:
    USER INPUT: T_CLK, T_OBS, t_BIN, LFSR_SIZE
    - FROM USER INPUT
        . find the number of bins in T_OBS (0,1,...A) and store in an array
        . find the max_count value fron LFSR_SIZE (2^LFSR_SIZE)
        . find the integer factor between T_OBS and T_p
        . FOR THE RANGE OF POSSIBLE T_P (CNT X T_CLK) FOR CNT: 1,2,..,2^LFSR_SIZE
        . CALCULATE all resulting values of 'a' element of [0-A]
        . Verify if the calculated value is in the Range stored in the array
        . IF A VALUE not found, Then the generated T_p does not produce all 'a' values.

*/
#include <iostream>
#include <cmath>

using namespace std;

#define T_OBS   27
#define LFSR_SIZE  15

// fucntion to search for a value
bool allCovered(int a[], int size){
    for (int i=0; i<size; i++){
        if ( a[i] == 0 )
            return false;
    }
    return true;
}
/* functino to calculate MAX value in an array */
int maxVal (int a[], int size){
    int maxValue =0;
    for (int i= 0; i<size; i++){
        if (a[i] > maxValue)
            maxValue = a[i];
    }
    return maxValue;
}

/* functino to calculate MIN value in an array */
int minVal (int a[], int size){
    int minValue =10000;
    for (int i= 0; i<size; i++){
        if (a[i] < minValue)
            minValue = a[i];
    }
    return minValue;
}

/* functino to calculate AVG value in an array */
float avgVal (int a[], int size){

    float totalNum = 0.0;
    for (int i= 0; i<size; i++){
        totalNum = totalNum + a[i];
    }
    return totalNum/size;
}

// FUNCTION TO PRINT OUT THE CONTENT OF COUNTERS ARRAY
void printArray(int BIN_ARRAY[], int ARRAY_SIZE){
int k = 1;
int CUM_COUNTER = 0;
int totalPulses = 0;
for (int i=1; i<=ARRAY_SIZE; i++){
```

```
cout<<"BIN: "<<k<<"-"<<( (i % 10 == 0) ? 10 : i % 10)<<"      COUNT:     "<<BIN_ARRAY[ i-1 ]<<endl;
   //cout<<"BIN: "<<i<<"      COUNT:     "<<BIN_ARRAY[ i-1 ]<<endl;
   if ( i % 10 == 0 ){
      CUM_COUNTER = 0;
      for (int j=0;j<10;j++){
         CUM_COUNTER += BIN_ARRAY[i-10+j];
      }
      totalPulses += CUM_COUNTER;
      cout<<"Total Pulses In Row-"<<k<<": "<<CUM_COUNTER<<endl;
      k ++;
      cout<<endl;
      }
   }
   cout<<"Total Pulses: "<<totalPulses<<endl;
}

int main(){
   // DECLARED VARIABLES
   int A, CNT_MAX;
   int a, k;
   double d_Tp, d_Tclk, d_Tobs, d_tBIN, N;
   char dummy_char, more = '1';

   while (more != '0'){
      d_Tobs = T_OBS;
      cout<<"Enter T_CLK: ";
      cin>>d_Tclk;
      cout<<endl;
      N = LFSR_SIZE;

      d_tBIN = 0.1;
      // Calculate range A
      A = d_Tobs/d_tBIN;// decimal part truncated by casting from double to integer
      cout<<"A Range =  "<<A<<endl;

      int *A_array = new int[A];// index flag array
      int *C_array = new int[A];// index count array..how many times each index detected
      // fill the array with all zeros
      for (int j=0; j<A; j++ ){
         A_array[j] = 0;
         C_array[j] = 0;
      }

      CNT_MAX = pow(2.0,N);
      cout<<"Max Count 2 ^ "<<N<<" = "<<CNT_MAX<<endl;
      for (int i = 1; i <= CNT_MAX; i++){
         // calculate T_p
         d_Tp = i * d_Tclk;
         k = d_Tp / d_Tobs;  // obtain the factor k
         //cout<<"k = "<<k<<endl;
         a = (d_Tp - k*d_Tobs)/d_tBIN;
         //cout<<"a = "<<a<<endl;
         // write a flag to the corresponding bin location. Only the bins covered will have a 1
         A_array[a] = 1;
         C_array[a] = C_array[a] + 1;
      }
      cout<<"The last Tp-p: "<<d_Tp<<endl;
      cout<<"Last k = "<<k<<endl;
      cout<<"Last Sub-interval: "<<d_Tp - k*d_Tobs<<endl;
      // search for zeros
      if ( ! allCovered(A_array, A) )
         cout<<"Not All Values Covered!!"<<endl;
      else
         cout<<"All Values Covered!!"<<endl;
      cout<<"Press a letter followed by RETURN KEY to continue...";
      cin>>dummy_char;
      printArray(C_array, A);
```

```
        cout<<endl;
        //MAX ARRAY VALUE
        cout<<"Max Value: "<<maxVal( C_array, A )<<endl;

        //MIN ARRAY VALUE
        cout<<"Min Value: "<<minVal(  C_array, A)<<endl;

        //AVERAGE VALUE
        cout<<"Avg Value: "<<avgVal(  C_array, A)<<endl;
        cout<<endl;
        cout<<endl;
        cout<<"Continue testing? press a number key followed by RETURN KEY...(0 to exit)";
        cin>>more;
        cout<<endl;
        delete A_array;
    }
    return 0;
}


♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦ PATTERN GENERATION EMULATION ♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦♦
/**** THIS C++ CODE FOR TESTING LFSR-BASED PULSE GENERATION FOR RANDOMNESS *****/
/* AUTHOR: AMIRI AMIR M. */
/* ALGORITHM:

        . FROM THE FIRST PULSE, ACCUMULATE THE TIME UNTIL LAST
        . FOR EACH PULSE TAKE THE COARSE PART OF TIME (K x T_OBS )
        . VERIFY WHERE THE FINE PART ENDS UP
        . OUT OF 300 TIME BINS OF IDEAL WIDTH (100ps = 30ns/300-BINS), INCREMENT CORRESPONDING
PULSE COUNTERS
          WHENEVER A PULSE ENDS UP IN THAT BIN. (i.e. IF A PULSE ENDS IN BIN=239, COUNTER-239 is
INCREMENTED)

  ** --- IN ALGORITHM FORM --- **

        - ACCUMULATE TIME FROM FIRST PULSE
        - GET THE BIN
           . DO "CUMMULTAED_TIME % T_OBS" (STORED IN A DOUBLE)
           . ASSIGN TO AN INTEGER (DECIMAL POINTS TRUNCATED HERE)
           . MAKE THE DECIMAL POINTS TO ONE DIGIT ONLY
              - MULTIPLY BY 10.0 AND CAST TO AN INTEGER
              - RE-DIVIDE BY 10.0 CAST BACK TO DOUBLE
           . CALCULATE THE INDEX (BIN) OF THE ARRAY
              - ADD REMAINDER (from modulus operation) TO ONE-DIGIT PRECISION DECIMAL POINT
              - CAST THE RESULTS TO DOUBLE,
              - MULTIPLER BY 10.0
              - ASSIGN TO AN INTEGER to be able to use as ARRAY_BIN[ INDEX ];
        - INCREMENT THE CORRESPONDING COUNTER
              - BIN_ARRAY[ index ]++
        - REPEAT FOR DESIRED_NUM_OF_PULSES
        - PRINT BIN COUNTER CONTENTS

        - IF MORE TEST DESIRED
           . RE-INITIALIZE LFSR USING SEED FUNCTION
           . REPEAT THE ABOVE

/* 999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
*/

#include <iostream>
#include <stdio.h>
#include <iomanip>
#include <cmath>
#include <ctime>

// INITIALIZED VALUES ARE FOR EVERY 50K VALUES..
//#define INITIAL_VALUE   6 //, 9166074 , 12819109 , 12641911 , 4544461 , 4847540
```

```
//#define T_REF        2.277 //11.42857 //5.71

#define T_OBS       27 // 30 ns time window
#define ARRAY_SIZE     270
#define NUM_OF_PULSE      25000
// GLOBAL VARIABLE FOR COUNTERS ARRAY
int BIN_ARRAY[ARRAY_SIZE];

using namespace std;

/* functino to calculate MAX value in an array */
int maxVal (int a[], int size){
   int maxValue =0;
   for (int i= 0; i<size; i++){
      if (a[i] > maxValue)
         maxValue = a[i];
   }
   return maxValue;
}

/* functino to calculate MIN value in an array */
int minVal (int a[], int size){
   int minValue =10000;
   for (int i= 0; i<size; i++){
      if (a[i] < minValue)
         minValue = a[i];
   }
   return minValue;
}
/* functino to calculate AVG value in an array */
float avgVal (int a[], int size){
   float totalNum = 0.0;
   for (int i= 0; i<size; i++){
      totalNum = totalNum + a[i];
   }
   return totalNum/size;
}

/* FUNCTION TO DISPLAY BITS OF THE INTEGER ARGUMENT*/
void displayBits(unsigned LFSR)
{
   const int SHIFT = 8 * sizeof( unsigned ) - 9;
   const unsigned MASK = 1 << SHIFT;
   cout<<setw(12) <<LFSR << " = ";
   for (unsigned i = 1; i <= SHIFT + 1; i++){
      cout<< (LFSR & MASK ? '1' : '0' );
      LFSR <<= 1;
      if (i % 8 == 0)
         cout<<' ';
   }
   //cout << "                 ";
}

/* FUNCTION TO CLOCK LFSR */
unsigned int nextValue (unsigned LFSR)
{
   unsigned int xored_1 = 0, xored_2 = 0, xored_3 = 0;
   unsigned int SHIFT_IN = 0;
    /*** 16-BIT LFSR { p(X) = 1 + X2 + X3+ X5 + X16 } **   */
   xored_1 = (LFSR & 0x00000001) ^ ( (LFSR & 0x00000004) >> 2);  // XORING 0-TH and second TAPPED
OUT BITS
   xored_2 = xored_1 ^ ( (LFSR & 0x00000008) >> 3 );          // XORING the above with third
TAPPED OUT
   xored_3 = xored_2 ^ ( (LFSR & 0x00000020) >> 5 );          // XORING the above with fifth
tapped out bit
```

```
   LFSR      = LFSR >> 1;                   // SHIFT THE LOWER BITS TO THE RIGHT BY 1
   LFSR = LFSR & 0x0000FFFF;
   SHIFT_IN  = xored_3 << 15;
   LFSR      = LFSR | SHIFT_IN;
   return LFSR;
}

// FUNCTION TO PRINT OUT THE CONTENT OF COUNTERS ARRAY
void printArray(){
   int k = 1;
   int CUM_COUNTER = 0;
   int totalPulses = 0;

   for (int i=1; i<=ARRAY_SIZE; i++){
      cout<<"BIN: "<<k<<"-"<<( (i % 10 == 0) ? 10 : i % 10)<<"      COUNT:    "<<BIN_ARRAY[ i-1
]<<endl;
      //cout<<"BIN: "<<i<<"      COUNT:    "<<BIN_ARRAY[ i-1 ]<<endl;
      if ( i % 10 == 0 ){
         CUM_COUNTER = 0;
         for (int j=0;j<10;j++){
            CUM_COUNTER += BIN_ARRAY[i-10+j];
         }
         totalPulses += CUM_COUNTER;
         cout<<"Total Pulses In Row-"<<k<<": "<<CUM_COUNTER<<endl;
         k ++;
         cout<<endl;
      }
   }
   cout<<"Total Pulses: "<<totalPulses<<endl;
}
int main(){
   // DEFINE VARIABLES
   int    index = 0;
    //unsigned int seed;
   unsigned int temp;
   double T_REF;
   char dummy_char, more = '1';
   unsigned int LFSR; //          = INITIAL_VALUE;
   unsigned int timeCounter_int = 0;
   unsigned int remainder     = 0;
   double decimal_points   = 0.0;
   double COUNTER_d;
   double timeCount_d = 0;
   unsigned int i      = 0;
   double CUMMULATED   = 0.0;

   while (more != '0'){
      //cout<<"ENTER a seed value to GENERATE A RANDOM SEED for LFSR..";
      //cin>>seed;
      cout<<"ENTER T_CLK: ";
      cin>>T_REF;

      srand( time(0) );
      LFSR = 1 + rand();
      // initialize array to zeros
      for (int l=0; l<ARRAY_SIZE;l++)
         BIN_ARRAY[l] = 0;

      cout<<" LFSR INITIALIZED TO:  "<<endl;
      displayBits( LFSR );
      cout<<endl<<endl;
      cout<<"Press a letter followed by RETURN KEY to continue...";
      cin>>dummy_char;

      i = 0;
      while (i < NUM_OF_PULSE){
            LFSR = nextValue( LFSR );     // CLK LFSR
            COUNTER_d = LFSR;               // ASSIGNING TO THE COUNTER (CASTING TO A DOUBLE
            //ACCUMULATE TIME IN ns
            timeCount_d = (COUNTER_d * T_REF); // CALCULATE TIME        timeCount_d +
            timeCounter_int = timeCount_d; // TRUNCATE TO INTEGER
                        //cout<<"TCounter Untruncated: "<<fixed<<timeCount_d<<" "<<"TCounter
               Truncated: "<<fixed<<timeCounter_int<<endl<<endl;
            remainder    = timeCounter_int % T_OBS;
            decimal_points  = timeCount_d - timeCounter_int;
            //cout<<"DECIMAL POINT before Round off: "<<decimal_points<<endl;

            // set precision to only 1 decimal point TO GET in 0.1 ns (100PS)
            temp = (unsigned int)(decimal_points * 100.0);
            //cout<<"Temp: "<<temp<<endl;

            decimal_points = (temp % 10 >= 5 ) ? ( (temp/10)+1 )/10.0 : (temp/100.0);
            //cout<<"DECIMAL POINT after round off: "<<decimal_points<<endl<<endl;

            //cout<<"Remainder: "<<remainder<<"       "<<"DECIMAL VALUES:
"<<decimal_points<<endl;
            //cout<<"DECIMAL POINT MULT.BY 10: "<<(int)(decimal_points * 10)<<endl;

            // Calculate the bin index
            index =  ( ( (double)remainder + decimal_points ) * 10) ;
            //cout<<"INTEGER BIN # :"<<index<<endl;
            //cout<<"DOUBLE BIN #: "<< ((double)(remainder) + decimal_points ) * 10.0 <<endl;
            //cout<<"Remainder: "<<remainder<<"    INDEX INT: "<<index<<endl;

            BIN_ARRAY[ index ] ++; // INCREMENT BIN COUNTERS
            i++;
         }
         printArray();
         cout<<endl;
         //MAX ARRAY VALUE
         cout<<"Max Value: "<<maxVal( BIN_ARRAY, ARRAY_SIZE )<<endl;
         //MIN ARRAY VALUE
         cout<<"Min Value: "<<minVal( BIN_ARRAY, ARRAY_SIZE  )<<endl;
         //AVERAGE VALUE
         cout<<"Avg Value: "<<avgVal( BIN_ARRAY, ARRAY_SIZE )<<endl;
         cout<<endl;
         cout<<endl;
         cout<<"Continue testing? press a number key followed by RETURN KEY...(0 to exit)";
         cin>>more;
         cout<<endl;
   }
   cout<<endl;
   return 0;
}
```