POLYPUBLIE

Conception d'un logiciel de contrôle pour la plate-forme de prototypage waferboard TM

Mikaël Hervé Guillemot

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

Date	2013
Type	Mémoire ou thèse / Dissertation or Thesis Mémoire de maîtrise/Master's thesis
URL de PolyPublie PolyPublie URL	https://publications.polymtl.ca/1141/
Conditions d'utilisation Terms of use	© Tous droits réservés / All rights reserved
Référence Citation	Guillemot, M. H. (2013). Conception d'un logiciel de contrôle pour la plate-forme de prototypage waferboard TM [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1141/

Détails de la thèse
Thesis details

Directeurs de recherche Advisors	Yvon Savaria, & Yves Blaquière
Programme Program	génie électrique

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal
https://publications.polymtl.ca

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'UN LOGICIEL DE CONTRÔLE POUR LA PLATE-FORME DE PROTOTYPAGE WAFERBOARD™

MIKAËL HERVÉ GUILLEMOT

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

AVRIL 2013

© Mikaël Hervé Guillemot, 2013.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

CONCEPTION D'UN LOGICIEL DE CONTRÔLE POUR LA PLATE-FORME DE PROTOTYPAGE WAFERBOARD^{TM}

présenté par : GUILLEMOT Mikaël Hervé

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. SAVARIA Yvon, Ph.D., membre et directeur de recherche

M. BLAQUIERE Yves, Ph.D., membre et codirecteur de recherche

M. BOYER François-Raymond, Ph.D., membre

REMERCIEMENTS

Tout d'abord, je tiens à remercier mon directeur de recherche, M. Yvon Savaria, qui a su m'accorder sa confiance dès mon arrivée, mais aussi me guider dans ma découverte du milieu de la recherche. Merci d'avoir été à l'écoute et de m'avoir offert l'opportunité d'orienter mon travail vers le sujet qui aujourd'hui fait l'objet de ce mémoire.

Je tiens aussi à remercier mon codirecteur de recherche, M. Yves Blaquière pour ses conseils et sa disponibilité. J'ai beaucoup apprécié la possibilité que vous m'avez offerte de vous rencontrer régulièrement compte tenu de vos emplois du temps bien remplis. Je vous remercie de plus pour l'opportunité que vous m'avez offerte de venir travailler avec vous ici, au Québec.

J'ai aussi apprécié les défis que vous m'avez proposés et l'attention que vous avez portée à mes requêtes et mes propositions vis-à-vis du projet. Malgré le retour aux études je me suis retrouvé dans un environnement professionnel stimulant, et je vous en remercie. La qualité de cet environnement, je la dois aussi à notre partenariat avec l'industriel Gestion TechnoCap que je remercie pour sa contribution financière, notamment messieurs Richard Prytula, directeur et Richard Norman, inventeur du projet DreamWafer. Sans eux, le projet n'existerait simplement pas.

Je souhaite aussi remercier particulièrement mon ami Etienne Lepercq, car sans lui je n'aurais probablement jamais entendu parler de ce projet. Car je dois l'admettre, c'est bien toi qui as fait germé dans ma tête cette idée de venir faire une maîtrise au Québec. Tu as su me communiquer un peu de ta passion pour ce projet. Et sans cela, je serais probablement encore dans une lointaine ville de Bretagne ou pire, en région parisienne !

Un grand merci aussi à Nicolas Laflamme-Meyer et à Jean-Sébastien Turgeon qui, avec Etienne ont été capables de m'éclairer sur tous les aspects matériels du projet. Votre soutien m'a beaucoup apporté, et vous m'avez permis de voir ce projet sous un autre angle.

De plus, je tiens à remercier de tout cœur mes collègues chercheurs, professionnels et stagiaires car c'est grâce à eux que j'ai pu apprécier l'ambiance de travail de ces deux

dernières années. Et c'est grâce à notre travail commun, qu'un prototype du logiciel WaferConnect a pu voir le jour.

Je souhaite enfin remercier ma famille qui a su me soutenir tout au long de ces études et qui a aussi su respecter mes choix, ce qui m'a permis de vivre pleinement cette nouvelle expérience sans regrets ni remords.

RÉSUMÉ

L'évolution des domaines de l'électronique donne naissance à des circuits toujours plus complexes. Les phases de conception et de prototypage représentent une partie importante du coût de développement des produits. Le projet DreamWafer porte sur le prototypage rapide de systèmes électroniques. Il tente de proposer une nouvelle approche au prototypage et à la vérification fonctionnelle de systèmes électroniques, ce qui réduit ainsi leurs coûts de développement et les temps d'arrivé sur le marché. La plateforme s'appuie sur un circuit intégré à l'échelle de la tranche de silicium pour proposer l'équivalent d'un PCB reconfigurable. Une mer de contacts assure la connexion avec les composants déposés par l'usager sur la tranche de silicium et un réseau d'interconnexions configurables permet d'obtenir des routes équivalentes aux pistes de cuivre entre les billes des composants.

L'objectif général du projet de recherche présenté ici consiste à concevoir le logiciel nécessaire au support d'une telle plateforme. Le projet comprend l'élaboration d'une architecture générale favorisant l'intégration des ressources techniques existantes (comme les algorithmes de routage), la conception d'une architecture évolutive simplifiant les accès au matériel, et la réalisation d'un système d'affichage adapté à la complexité d'une telle plateforme. Cette conception doit se faire dans un environnement technique bipolaire (électronique et logiciel) connaissant un renouvellement important des personnes intervenant dans le projet.

L'architecture développée au cours de ces travaux est conçue pour prendre en charge les contraintes liées au pilotage de la plateforme électronique et de les isoler pour simplifier l'intégration matériel-logiciel et les développements à venir. Ce travail à fait l'objet d'un chapitre soumis pour le livre « *Novel Advances in Microsystems Technologies and their Applications* ».

À cette architecture s'ajoute une contribution technique publiée à la conférence CCECE 2013 (*Canadian Conference on Electrical and Computer Engineering*) dont l'objectif est de permettre l'affichage rapide du grand volume de données requis pour la représentation d'une telle plateforme.

Les résultats présentés dans ce mémoire mettent en avant l'aboutissement de l'architecture et du développement logiciel ainsi que l'efficacité des mécanismes d'affichage. Le logiciel résultant de ce travail est actuellement utilisé en démonstration et sert d'outil de qualification pour la plateforme matérielle.

ABSTRACT

Electronic system design methods are in constant evolution with access to technologies enabling the creation of increasingly complex circuits. The design and prototyping phases of today's systems represent a significant part of the overall cost of many products. The DreamWafer project attempts to propose a new approach to the rapid prototyping of Printed Circuit Board, reducing costs and delays. The platform is based on a wafer-scale silicon integrated circuit that works as a reconfigurable PCB. A sea of contacts on its top surface provides the connections with the user components deposited on it and a configurable interconnection network is used to create routes as in a traditional PCB.

We present the software written to support this platform. The project includes the development of general software architecture, the design of a scalable model simplifying access to equipment and the construction of a display system able to handle the dataset for such a platform.

The architecture developed in this work is designed to handle the constraints related to the hardware while assisting the integration of functional modules. This work was the subject of a chapter submitted for the book « Advances in Novel Microsystems Technologies and their Applications ».

In addition to this, an efficient new mechanism for the rapid display of large amounts of data representing the system is proposed. This technical contribution is going to be published in CCECE 2013 conference (Canadian Conference on Electrical and Computer Engineering).

Software resulting from this work is currently being used for demonstration and must serve as a validation tool for the hardware.

TABLE DES MATIÈRES

REMERCIEMENTS III

RÉSUMÉ V

ABSTRACT VII

TABLE DES MATIÈRES VIII

LISTE DES TABLEAUX X

LISTE DES FIGURES XI

LISTE DES SIGLES ET ABRÉVIATIONS XIII

LISTE DES ANNEXES XIV

INTRODUCTION 1

CHAPITRE 1 REVUE DE LITTERATURE 5

1.1 Contexte de recherche : le projet DreamWafer 6

1.1.1 Organisation du WaferIC 6

1.1.2 Flot de travail utilisateur 9

1.1.3 Environnement du WaferIC 10

1.1.4 Éléments logiciels existants 14

1.2 Prise en charge logicielle des systèmes électroniques 16

1.3 Modèle de conception logicielle 18

1.3.1 Architecture client-serveur 19

1.3.2 Architecture en couches 20

1.3.3 Architecture Modèle-Vue-Contrôleur 22

1.4 Techniques de programmation graphique 23

1.4.1 Comparaison entre dessin vectoriel immédiat et image matricielle 23

1.4.2 Technologie d'accélération graphique matérielle 26

1.5 Conclusion 28
CHAPITRE 2 CONCEPTION DU LOGICIEL WAFERCONNECT 29
2.1 Vue d'ensemble du logiciel WaferConnect 29
2.2 Contraintes sur flot de travail utilisateur 31
2.2.1 Adaptations réalisées 33
2.3 Architecture du logiciel WaferConnect 35
2.3.1 Représentation logicielle du WaferBoard 38
2.3.2 API d'accès au WaferIC et communication 41
2.4 Interface utilisateur 44
2.4.1 Spécifications de l'interface graphique utilisateur 45
2.4.2 Organisation et implémentation du GUI 47
2.4.3 Affichage du WaferIC et prise en charge des erreurs 51
2.5 Résultats 52
CHAPITRE 3 INTERFACE AVEC LE MATERIEL 55
3.1 Proposition d'architecture d'interface 55
3.2 Résultats 61
CHAPITRE 4 VISUALISATION DE L'ÉTAT DU CIRCUIT 62
4.1 Spécifications 63
4.2 Proposition d'une solution performante d'affichage 64
4.3 Résultats 68
CONCLUSION 71
RÉFÉRENCES 74
ANNEXES 77

LISTE DES TABLEAUX

Tableau 2.1: Commandes supportées par l'interpréteur de WaferConnect. 53

Tableau 2.2: État du développement et intervenants. 54

Tableau 4.1: Comparaison temporelle des techniques présentées. 69

LISTE DES FIGURES

Figure 1-1: Architecture du WaferIC. 7

Figure 1-2: Structure du réseau JTAG à l'échelle de la cellule. 8

Figure 1-3: Structure du WaferNet à l'échelle de la cellule. 9

Figure 1-4: Flot de travail utilisateur de la plateforme de prototypage. 10

Figure 1-5: Coupe transversale du WaferBoard. 11

Figure 1-6: Alimentation et indexation des images de réticule (A) et des PowerBlocks (B). 12

Figure 1-7: Distribution de l'information au sein du WaferBoard. 13

Figure 1-8: Organisation simplifiée d'un système informatique. 17

Figure 1-9: Exemple d'encapsulation au sein d'un pilote. 18

Figure 1-10: Architecture client serveur-simple. 20

Figure 1-11: Architecture à trois couches. 21

Figure 1-12: Interactions Modèle Vue et Contrôleur. 23

Figure 1-13: Exemple de distorsion induite par l'agrandissement d'une image matricielle, ici d'une cellule. A est l'original, B est l'agrandissement obtenu et C est l'image attendue idéalement. 25

Figure 1-14: Exemple d'agrandissement d'une image vectorielle à l'échelle du NanoPad. 27

Figure 2-1: Dépendances des étapes du flot de travail utilisateur. 33

Figure 2-2: Diagramme de classes du gestionnaire de flot de travail de l'utilisateur. 34

Figure 2-3: Architecture générale du logiciel WaferConnect et son découpage en couches. 37

Figure 2-4: Diagramme de classes UML du WBHR (*WaferBoard Hardware Representation*). 40

Figure 2-5: Exemple de trame à destination des PowerBlocks (A) et son évolution (B). 42

Figure 2-6: Exemple de chemin de configuration JTAG. 44

Figure 2-7: Interaction entre les sous modules GUI. 46

Figure 2-8: Environnement graphique de base de WaferConnect. 47

Figure 2-9: Environnement graphique avancé de WaferConnect. 48

Figure 2-10: Interpréteur de commandes de WaferConnect utilisé sous Windows. 50

Figure 3-1: Encapsulation des mécanismes génériques de pilotage. 57

Figure 3-2: Intégration de la structure de données. 59

Figure 3-3: Situation des simulateurs dans l'architecture. 60

Figure 4-1: Mécanismes du système de sélection. 67

LISTE DES SIGLES ET ABRÉVIATIONS

API	Application Programming Interface
AWT	Abstract Window Toolkit
CAD	Computer-Aided Design
CAO	Conception Assistée par Ordinateur
CPU	Central Processing Unit
CORBA	Common Object Request Broker Architecture
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HID	Human Interface Device
JTAG	Joint Test Action Group
LGPL	Lesser General Public License
MVC	Modèle-Vue-Contrôleur
PC	Personal Computer
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
RAM	Random Access Memory
SIG	Système d'Information Géographique
TSV	Through-Silicon Via
UIC	User Integrated Circuit
UML	Unified Modeling Language
USB	Universal Serial Bus
VBO	Vertex Buffer Object
WSI	Wafer-Scale Integration

LISTE DES ANNEXES

ANNEXE 1 – EXEMPLE D'UTILISATION DES VBO EN JAVA..... 77

INTRODUCTION

De nos jours, les technologies de miniaturisation sur silicium sont de plus en plus performantes et ne cessent de s'améliorer. Les circuits intégrés deviennent de plus en plus complexes à mesure que l'intégration augmente. Ils sont conçus pour être compacts et économes en énergie, tout en fournissant le plus de fonctionnalités possibles. Cette évolution vertigineuse présente toutefois quelques problèmes. Bien que de plus en plus de fonctionnalités soient intégrées au sein d'une même puce, les cartes de circuits imprimés (PCB) ont toujours un nombre de connexions important entre les composants.

De la complexité des puces récentes, associée à la volonté de miniaturisation des PCB, résulte un besoin de connexions très dense sur PCB. Cela rend le routage difficile et la validation complexe. En effet, pour réaliser le routage, il est souvent nécessaire de recourir à de nombreuses couches de cuivre, ce qui rend l'accès aux pistes internes (et donc la validation) problématique. Les phases de conception et de prototypage représentent une partie importante du coût des produits.

Ces différents facteurs combinés aux délais de fabrication rendent le prototypage de systèmes électroniques particulièrement onéreux. Il est de plus en plus courant de devoir réaliser plusieurs itérations avant l'obtention d'un prototype fonctionnel, utilisable pour l'intégration de la partie logicielle du système électronique. Certains outils existent pour réduire ces coûts. L'outil idéal devrait permettre la conception d'un prototype modifiable rapidement et à volonté tout en permettant la réutilisation ou le changement des composants du circuit et en assurant un contrôle et une observabilité de chacun des points du circuit.

Le projet de recherche DreamWafer™ [1] porte sur le développement d'un système de prototypage rapide pour les systèmes électroniques. L'objectif de ce projet consiste à réaliser l'équivalent d'un prototype d'un système électronique en seulement quelques minutes, réduisant ainsi leurs coûts de conception et leur temps d'arrivée sur le marché. La plateforme WaferBoard™ s'appuie sur un circuit intégré à l'échelle de la tranche de silicium (circuit Wafer-Scale Integration, WSI) nommé WaferIC™ pour proposer un PCB reconfigurable avec les outils pour le débogage.

Le principe de fonctionnement de cette plateforme est le suivant : une des surfaces du circuit présente une mer de contacts, appelés NanoPads. L'utilisateur dépose les composants de son système à prototyper sur cette surface. Chaque contact électrique entre les plots du composant et les NanoPads peut être observé, permettant au système de les détecter et d'identifier les composants. Il existe ainsi une connexion entre les plots du composant et un réseau d'interconnexions configurables interne au WaferIC. Ce réseau peut être programmé pour effectuer des liaisons entre les différentes plots de composants, liaisons se substituant ici aux pistes de cuivre d'un PCB.

Pour opérer le contrôle d'une telle plateforme matérielle, un outil logiciel de CAO (conception assisté par ordinateur) est essentiel. Cet outil doit proposer aux utilisateurs un ensemble de fonctions lui permettant d'interagir avec le matériel et une interface graphique permettant la visualisation des données à des fins de validation et de débogage de son système électronique prototypé.

Pour y arriver, de nouveaux algorithmes ont dû être élaborés pour configurer la plateforme, déterminer les routes à emprunter pour relier les composants ou simplement détecter la présence et la position des composants. En effet, même pour les puissants ordinateurs de bureau récents, certaines de ces tâches demandent un temps de calcul non négligeable. C'est la conception générale de ce logiciel, nommé WaferConnect, destiné à contrôler et configurer la plateforme de prototypage et à offrir à l'usager des outils de débogage qui fait l'objet de ce mémoire. Cela se résume en trois objectifs distincts : la conception du logiciel en vue de l'intégration, l'élaboration des mécanismes de communication avec le WaferBoard et la proposition de solutions d'optimisation.

Cette conception est assez particulière. En effet la partie logicielle a fait l'objet de recherches préliminaires ayant donné naissance à un certain nombre de ressources (algorithmes, organisation des données, programmes autonomes) permettant de prouver certains concepts et de tester le matériel. C'est seulement une fois ces ressources disponibles que s'est posée la question, voire le besoin du développement d'un logiciel unifié. La conception a donc été orientée par la nécessité de réutilisation, amenant deux contraintes majeures : proposer une architecture logicielle flexible, favorisant l'intégration des ressources existantes, mais aussi proposer et effectuer les adaptations nécessaires

lorsque l'existant n'est pas directement utilisable. Un besoin transverse s'est ajouté à ces contraintes : compte tenu de la nature du système à contrôler, la recherche de solutions performantes (en termes de délais d'exécution et de consommation de ressources) devait être privilégiée.

De plus, le volume d'informations à présenter à l'utilisateur potentiel du logiciel s'est avéré important. La définition d'une interface graphique permettant d'exposer ces informations de manière structurée s'est ajoutée à ce travail. Ces exigences ont été posées et consignées lors de la spécification du logiciel WaferConnect.

Le travail effectué se scinde selon trois axes distincts :

- La définition de l'architecture logicielle avec l'intégration des éléments existants;
- Le réagencement de la structure de données et des couches de communication;
- La conception d'une interface graphique adaptée à la présentation d'un grand nombre d'informations.

Après avoir présenté les éléments existant devant être intégrés, les méthodes d'architecture logicielle pouvant répondre à nos besoins et les techniques d'affichage usuelles permettant la prise en charge d'un grand nombre d'informations, chaque axe sera abordé de manière indépendante. Par souci de cohérence et de lisibilité, les résultats spécifiques seront présentés à la fin de chaque partie.

Le logiciel WaferConnect est destiné à offrir les outils nécessaires à la vérification fonctionnelle du système électronique à prototyper. Outre la recherche de performance temporelle, ce logiciel a été conçu pour offrir une prise en main simple et rapide, tout en proposant une grande quantité d'informations et une capacité de contrôle étendue. Un tel projet présente un travail de développement important, incompatible avec la quantité de développement attendue lors d'une maîtrise. Une fois la spécification rédigée, ce document de référence nous a permis de partager le travail de développement avec un certain nombre de professionnels et de stagiaires.

Le premier chapitre de ce mémoire a pour but de présenter les technologies utilisées par la suite ainsi que le fonctionnement de base de la plateforme. Le second chapitre introduit l'architecture du logiciel WaferConnect, dont les adaptations réalisées au développement

logiciel existant, les supports logiciels requis pour les interfaces de communication et l'interface utilisateur. Le troisième chapitre aborde l'interface de communication utilisée pour la plateforme et des outils logiciels pour le supporter. Enfin, le dernier chapitre propose une solution aux problèmes de performances rencontrés au niveau de l'affichage de l'état du circuit. Ces problèmes étant liés au volume d'information et à la vitesse d'affichage attendue, la solution proposée ne se limite pas à l'affichage d'un circuit électronique mais peut être utilisé de manière générale pour afficher rapidement un grand nombre de données vectorielles 2D.

Le logiciel obtenu est le fruit des efforts d'une équipe de développement dont les intervenants sont présentés au Tableau 2.2. Mon travail dans ce projet a consisté à établir les besoins en termes de logiciel et à les spécifier. Je me suis ensuite chargé de repartir et superviser les tâches de développement et d'intégration au sein de l'équipe, tout en assurant le rôle de référant technique. Grâce à ces efforts communs, le logiciel WaferConnect est présentement en phase d'intégration. L'interface utilisateur est stable, l'affichage est satisfaisant et les tests de la dernière version confirment les possibilités de pilotage du matériel alors que le prototype physique est en cours de finalisation. Des travaux sont encore en cours pour établir la carte des défauts matériels du circuit sur la tranche de silicium. Les possibilités d'utilisation de WaferConnect pour les tests et la qualification du matériel sont réelles.

CHAPITRE 1 REVUE DE LITTERATURE

L'objectif général de ce chapitre est de faire une revue de l'existant nécessaire à la compréhension des travaux présentés dans ce mémoire. Cette revue regroupe la présentation technique de la plateforme WaferBoard et la présentation des technologies envisagées pour concevoir et développer le logiciel WaferConnect. Ce chapitre comporte quatre parties structurées de la manière suivante :

La première partie a pour objectif de poser les contraintes techniques inhérentes à l'utilisation de la plateforme WaferBoard. Les fondements de la technologie et de l'architecture matérielle ainsi que le détail du fonctionnement de la plateforme présentée en première partie permettent une meilleure compréhension des décisions exposées par la suite. Cette partie introduit aussi les ressources préliminaires, antérieures à ce travail de maîtrise, à savoir les travaux de recherches et les développements effectués dans le cadre du projet DreamWafer. Ce sont ces ressources qui posent les bases des contraintes d'intégration.

La seconde partie présente les contraintes de pilotage d'un matériel électronique. Ces contraintes sont définies dans le cadre de l'utilisation du logiciel au sein d'un système d'exploitation avec des interfaces de communication complexes.

La troisième partie fait une revue des différentes technologies d'architecture logicielle ayant servi de base à la conception de l'architecture de WaferConnect.

Un portrait des différentes approches utilisées actuellement en programmation graphique sera dressé en dernière partie, en vu d'introduire les problématiques liées aux techniques d'affichage.

1.1 Contexte de recherche : le projet DreamWafer

Le projet DreamWafer est très vaste. Il couvre de multiples domaines et a fait l'objet de nombreux travaux de recherche [2]. Sa présentation détaillée n'est pas le but de cette partie. Nous présenterons ici les concepts et le fonctionnement général de la plateforme dans le but de mettre en évidence les éléments nécessaires à la compréhension des contraintes de conception et de développement de WaferConnect.

Le projet DreamWafer propose une nouvelle approche pour le prototypage rapide de systèmes électroniques avec une plateforme équivalent à un PCB reconfigurable. Le cœur de cette plate-forme est une tranche de silicium de 200 mm de diamètre sur laquelle est gravé un circuit intégré appelé WaferIC. Ce circuit présente un grand nombre de contacts (1 245 184) en surface : les NanoPads. Cette surface est recouverte d'un film anisotropique électriquement conducteur selon la verticale. L'utilisateur vient placer manuellement ses propres circuits intégrés (dénommés par la suite UIC) sur ce film; celui-ci permettant de protéger la surface du circuit tout en assurant un contact électrique entre les NanoPads et les plots (ou pins) des composants déposés par l'utilisateur. Le couvercle de l'appareil peut alors être refermé, permettant d'appliquer une pression mécanique sur les UIC, afin de les maintenir en place et d'assurer un bon contact électrique à travers le film anisotrope.

1.1.1 Organisation du WaferIC

Le circuit de la tranche de silicium est organisé de façon régulière selon une structure hiérarchique. Le cœur de la plateforme est composé d'un ensemble de 74 images de réticules (Figure 1-1). Ces réticules peuvent être configurés indépendamment, ce sont les plus grands éléments du circuit. Chaque réticule est composé d'une grille de 32 par 32 cellules (soit 1024 cellules par réticule). Ces cellules sont recouvertes par une grille de 4 par 4 NanoPads (soit 16 NanoPads par cellule) qui constituent l'interface physique avec la surface active de la tranche.

Chaque cellule contient un circuit configurable permettant un contrôle de ses entrées et ses sorties. Chacune est de plus reliée à deux réseaux d'interconnexions présentés ci-après. Des circuits assurant la configuration et l'alimentation de la tranche et des UICs viennent en contact avec la face arrière de la tranche. Ces circuits sont appelés PowerBlock (décrit à la

section 1.1.3) et permettent aussi d'établir la communication vers l'extérieur, jusqu'à l'ordinateur de contrôle destiné à héberger le logiciel. La Figure 1-1 montre de manière schématique l'architecture du circuit WaferIC.

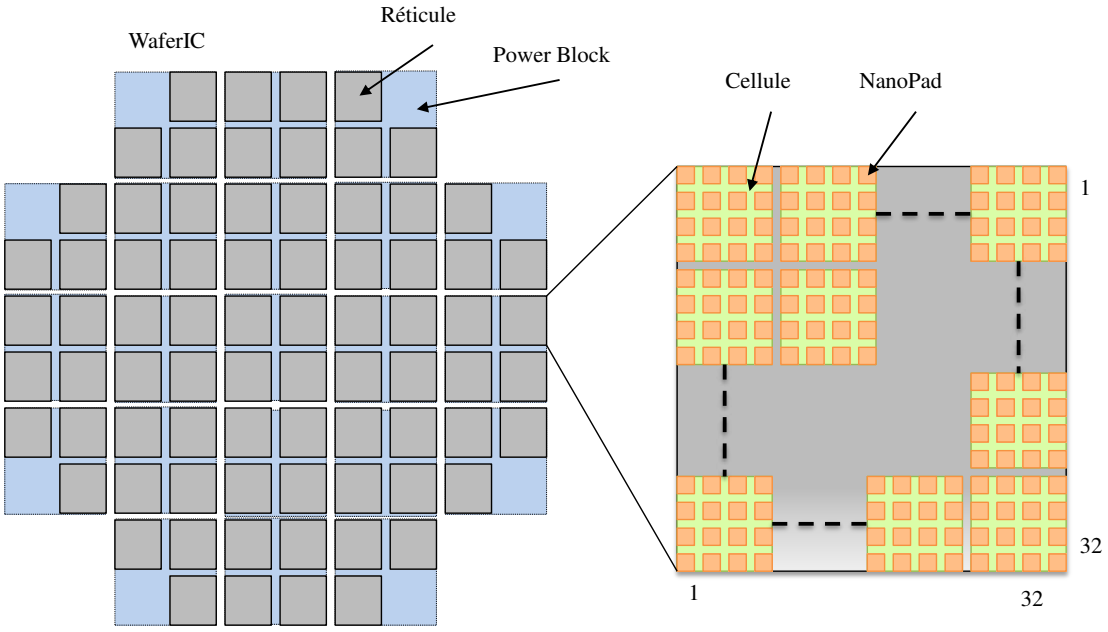


Figure 1-1: Architecture du WaferIC.

La cellule assure plusieurs fonctions. Elle permet entre autres de capturer et fournir les signaux sur ses propres NanoPads. Elle supporte jusqu'à deux connexions indépendantes entre ses NanoPads et l'extérieur via un système de multiplexage. Le circuit inclut deux réseaux d'interconnexions. Le premier permet la configuration de chaque cellule du circuit. Le second appelé WaferNet assure les connexions entre les NanoPads par l'intermédiaire des cellules.

Le réseau de configuration est un réseau de type JTAG [3] tolérant aux pannes. Ce réseau est critique car sans lui le circuit ne peut être programmé. Il est conçu pour résister aux défectuosités inhérentes à la fabrication de la tranche de silicium et comporte autant de groupe d'entrées/sorties que le circuit comporte de réticules. Il y a ainsi 5 signaux JTAG (TCK, TDI, TMS, TDO et TRST) par image de réticule, accessible par l'arrière de la tranche de silicium via les PowerBlocks. Ce réseau permet de transmettre l'information de

manière sérielle pour configurer le WaferNet et les NanoPads. Chaque NanoPad peut être en contact avec la bille d'un UIC et peut être configuré comme contact flottant, entrée ou sortie numérique, alimentation ou masse. La régularité de la matrice de cellule impose d'avoir un contrôleur JTAG par cellule. Le NanoPad étant un sous élément des cellules, le réseau JTAG est établi au niveau des cellules.

Chaque cellule du réseau est reliée à ses quatre voisines par deux connexions directionnelles comme présenté Figure 1-2. Cela permet de construire des chaînes de configuration pouvant emprunter des chemins variés et offre ainsi la redondance nécessaire pour que le réseau JTAG soit tolérant aux pannes [4].

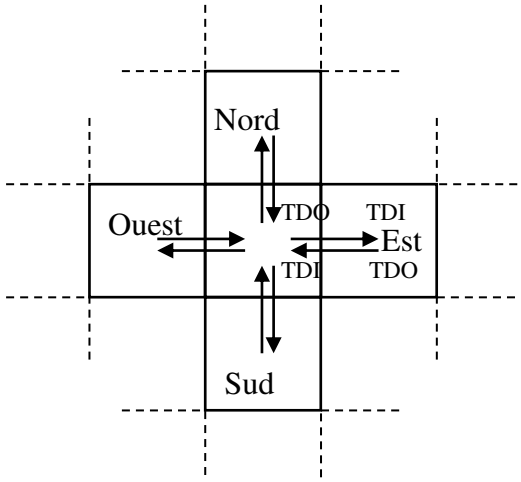


Figure 1-2: Structure du réseau JTAG à l'échelle de la cellule.

Le WaferNet est lui dédié à la configuration des routes [5]. Il s'appuie sur un maillage multi dimensionnel tolérant aux pannes. Comme l'autre réseau, il connecte les cellules les unes aux autres mais il est plus dense et plus étendu. Une fois configuré à l'aide d'une liste d'interconnexions fournie par l'utilisateur, des chemins spécifiques sont calculés, permettant d'établir une liaison électrique d'une cellule A à une cellule B comme le ferait une piste de cuivre sur un PCB. C'est ce réseau qui est utilisé pour le routage. Les cellules A et B sont ensuite configurées pour connecter cette piste aux NanoPads ciblés par le routage.

La structure du WaferNet est présentée à la Figure 1-3. Chaque cellule est liée à 6 autres cellules dans chaque orientation physique (Nord, Sud, Est et Ouest). La longueur des liaisons répond à la suite géométrique de raison 2, c'est à dire chaque cellule est liée aux 1^{e}, 2^{e}, 4^{e}, 8^{e}, 16^{e} et 32^{e} cellules voisines.

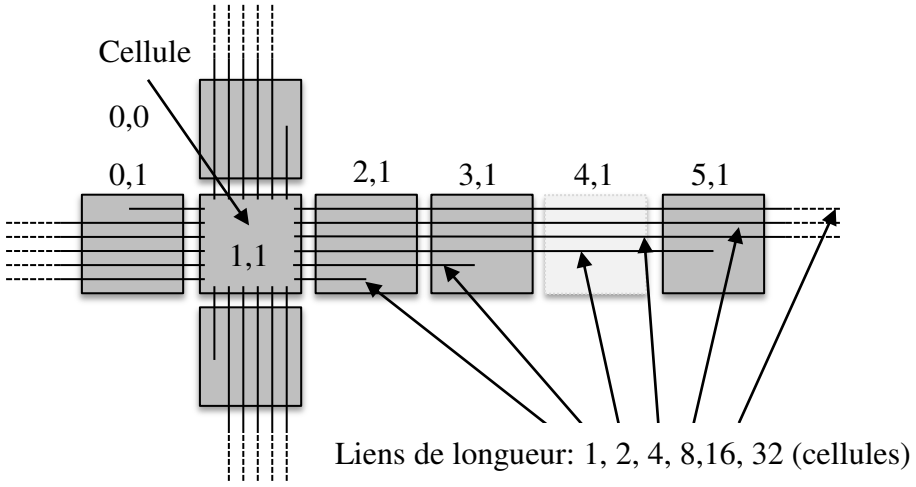


Figure 1-3: Structure du WaferNet à l'échelle de la cellule.

Les 1 245 184 NanoPads forment une surface de contacts. Lorsqu'un composant est déposé à la surface du wafer, chacune de ses plots va entrer en contact avec plusieurs NanoPads. Ce phénomène permet de détecter les plots à l'aide des courts-circuits ainsi créés entre les NanoPads.

1.1.2 Flot de travail utilisateur

La séquence théorique du flot de travail utilisateur [6] est présenté Figure 1-4. Tout d'abord, l'utilisateur place les composants électroniques (UIC) faisant partie du système à un prototype sur la WaferBoard, il ferme le couvercle, puis démarre la plateforme WaferBoard (mise sous tension). Une pression constante du couvercle sur les composants est maintenue afin d'assurer de bons contacts électriques entre les broches des UIC et les NanoPads. L'utilisateur importe ensuite les données de son circuit, à savoir sa netlist et les empreintes des UIC. Le logiciel effectue un diagnostic du réseau JTAG puis du WaferNet. Une fois ces étapes réalisées, il est possible d'accomplir l'étape de détection des plots.

L'opération de reconnaissance des boîtiers des UIC est exécutée, en prenant comme paramètres d'entrée la liste d'empreintes de boîtiers importée et les données obtenues lors de la détection de plots. Les UIC placés sur le WaferIC sont ainsi identifiés. La pression dans la poche est ajustée jusqu'à ce que tous les plots des uICs entrent en contact électrique avec les NanoPads ou jusqu'à ce que la pression maximale permise soit atteinte pour ne pas endommager la tranche de silicium. Au besoin l'utilisateur procède aux corrections et ajustements nécessaires. Le processus de routage peut alors être démarré, en utilisant la netlist et les contraintes importées. Le processus de routage génère les données de configuration du WaferNet. La dernière étape consiste à configurer réellement le WaferNet. Le circuit est enfin prêt à être prototypé et débogué.

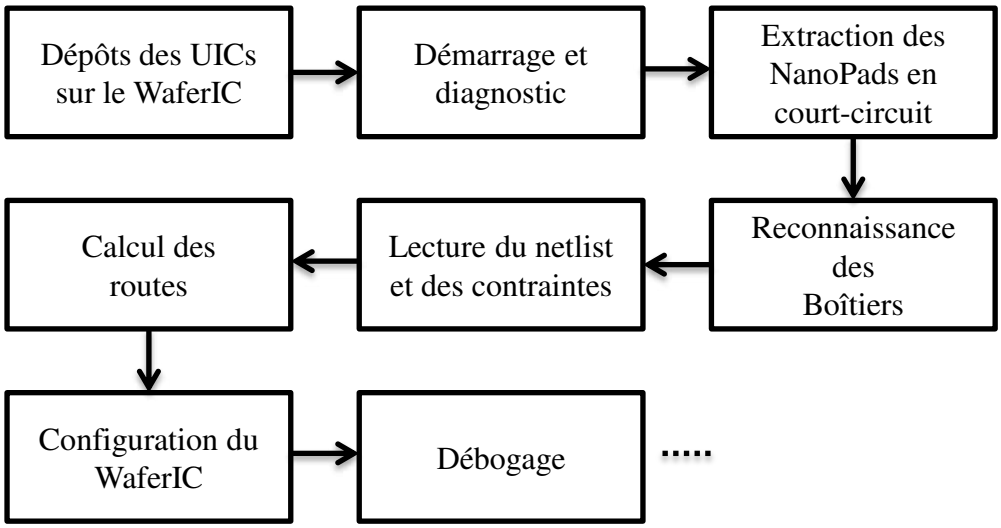


Figure 1-4: Flot de travail utilisateur de la plateforme de prototypage.

1.1.3 Environnement du WaferIC

La structure de base de la tranche et plus particulièrement la connexion entre les réticules et l'extérieur sont représentées Figure 1-5. Chaque groupe de quatre réticules est alimenté par un circuit imprimé miniature soudé à la tranche, appelé PowerBlock. Chaque PowerBlock assure l'alimentation électrique et la connexion des entrées/sorties de configuration avec

l'extérieur du WaferIC (via le BottomPCB). La Figure 1-5 présente une coupe transversale simplifiée permettant de situer le WaferIC dans le système.

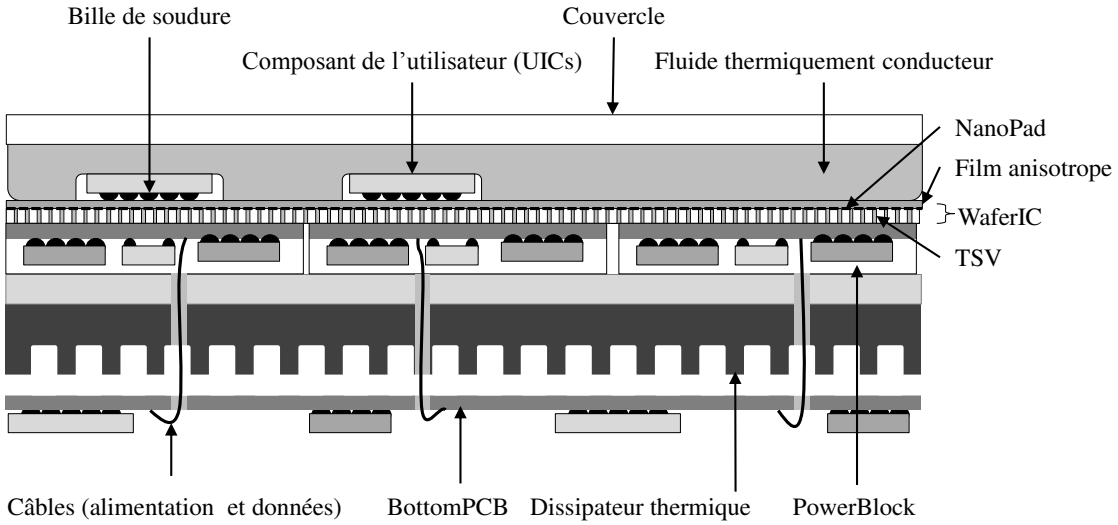


Figure 1-5: Coupe transversale du WaferBoard.

La Figure 1-6 présente la répartition des réticules aux PowerBlocks. Les carrés à bordure épaisse représentent les PowerBlocks et les carrés gris les images de réticules réellement reproduites sur la tranche. Le logiciel représente cette structure dans un tableau. Dans ce modèle, les images de réticules sont indexées comme indiqué sur la figure.

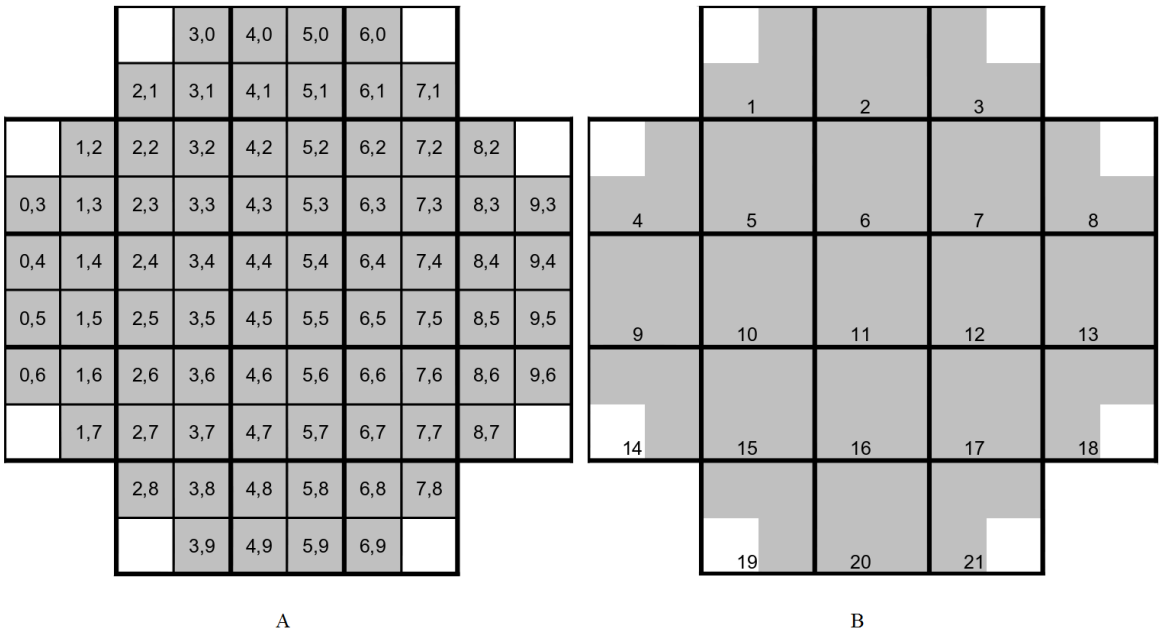


Figure 1-6: Alimentation et indexation des images de réticule (A) et des PowerBlocks (B).

Un circuit électronique nommé BottomPCB (en raison de son positionnement sous le WaferIC) se charge de la communication entre la station de travail (sur laquelle repose le logiciel WaferConnect) et le WaferIC, la gestion des capteurs, et la régulation de l'alimentation électrique, cela en passant par les PowerBlocks. Actuellement la communication entre le BottomPCB et la station de travail se fait via USB mais ce paramètre peut évoluer vers des ports plus rapides, dont GigE (Gigabit Ethernet) et PCIe (Peripheral Component Interconnect Express). La Figure 1-7 schématise l'organisation de la communication entre les modules.

Pour assurer une distribution des données entre la station de travail et les différents circuits de la plateforme, un protocole de communication a été établi [7]. Il permet l'empaquetage et le routage des informations qui peuvent êtres adressées à chaque carte de contrôle à savoir le BottomPCB et les PowerBlocks, ou au WaferIC.

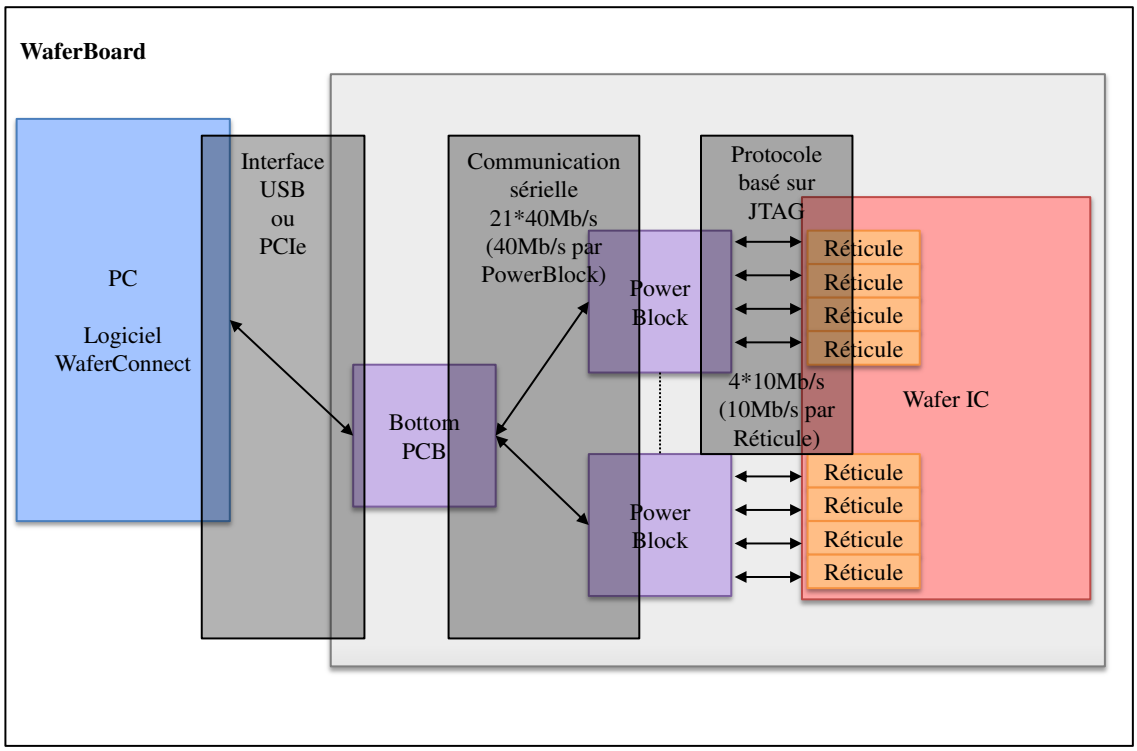


Figure 1-7: Distribution de l'information au sein du WaferBoard.

La configuration complète de la tranche peut nécessiter la circulation d'un grand nombre d'informations encapsulées dans des trames. La communication doit donc être rapide pour devenir transparente à l'utilisateur. Pour cela, les données de configuration sont transmises sous forme de commandes JTAG qui sont ensuite interprétées dans les PowerBlock. Dans le cas général, le protocole est efficace. Mais certaines erreurs critiques entrainent une perte de trame et le système peut alors perdre la connaissance de l'état du réticule, voir de la tranche. La configuration devra être refaite en conséquent.

Le BottomPCB propose des tampons (FIFO) pouvant contenir un certain nombre de trames. Pour des raisons de performances on ne peut observer en permanence l'état de ces tampons. On doit donc effectuer un contrôle logiciel du débit de données pour éviter une saturation physique, et donc des pertes de trames, tout en conservant un débit élevé.

Chaque PowerBlock permet de configurer directement jusqu'à quatre réticules. Les PowerBlocks sont indépendants, ce qui apporte une possibilité de configuration parallèle.

Le mécanisme de contournement de défaut de la tranche prévoit toutefois de configurer un réticule inaccessible de manière directe en passant par un autre réticule et donc potentiellement par un autre PowerBlock [8]. Ce cas particulier doit être pris en charge automatiquement par les mécanismes de formatage logiciel des données de telle sorte qu'ils soient transparents pour l'utilisateur et les développeurs des modules externes à la couche de communication. Cette tâche est prise en charge par le module JLIB [9] présenté dans la section 2.3.2.

1.1.4 Éléments logiciels existants

Comme évoqué en introduction, un certain nombre d'éléments logiciels étaient disponibles avant ce travail. Une fois les bases de l'architecture posées, ces éléments ont été adaptés en vue de leur intégration par une équipe de stagiaires et de professionnels. Ces éléments sont parfois qualifiés de modules externes car l'architecture de WaferConnect permet de considérer la plupart comme des plugins remplissant chacun une tâche spécifique.

Les principaux modules sont :

- Diagnostic JTAG : Un module critique dont le but est de déterminer les parties utilisables du réseau de configuration JTAG du WaferIC en validant les liens JTAG au niveau de la cellule. Des liens non fonctionnels sont inévitablement produits en raison des procédés de micro-fabrication du circuit. L'identification de ces liens est indispensable pour déterminer des chemins de configuration alternatifs si nécessaire, et garantir ainsi un réseau de configuration fiable. Une version préliminaire de ce module a été principalement implémentée par Jean-Sébastien Turgeon dans le cadre de son projet de maîtrise.

- Diagnostic WaferNet : Son objectif est d'établir la carte des liens fonctionnels du WaferNet. Ces données sont mémorisées et seront utilisées par le module de routage pour qu'il génère des routes passant que par des liens reconnus comme fonctionnels. Ce module est actuellement inexistant.

- Connexion aux outils CAD : Ce module se charge d'établir une connexion avec les logiciels de CAO « DxDesigner » et « PCB Expedition », tous deux édités par Mentor Graphics [10]. Cette connexion permet principalement d'importer les netlists et les empreintes des composants d'un circuit conçu avec ces outils. Ce module a été développé et intégré par l'équipe de professionnels peu après la spécification du WaferConnect.

- Détection de plots: Il permet de dresser la carte des NanoPads en court circuit et ainsi de localiser les pattes (ou plots) des composants en contact avec la surface du circuit. Cette détection se fait de manière massivement parallèle. Nous avons posé les bases d'un algorithme permettant de réaliser une détection complète en deux configurations du waferIC. Cette méthode a fait l'objet d'une publication [11] et le module a été développé et intégré par un de nos professionnels (Hai Nguyen) peu après la spécification de WaferConnect.

- Reconnaissance de boîtier de circuit intégré (*Package*) : La première fonctionnalité de ce module consiste à identifier les composants et leur position sur le WaferIC et est opérationnelle. Il se base sur les résultats de la détection de plots et la liste des empreintes importées pour proposer une identification automatique. Il doit de plus proposer une interface permettant à l'utilisateur de modifier ou d'approuver le résultat produit. Cette fonctionnalité est en cours de développement.

- Routage : De la même manière que les outils de conception de PCB, il est nécessaire de déterminer des routes physiques à partir des netlists (éventuellement importées du module Connexion avec DxDesigner) pour relier les composants. Cependant notre réseau est très différent de celui disponible sur un PCB réel. Ce module effectue le routage des netlists en se basant sur la position des composants fournie par le module de reconnaissance de package et en prenant en compte une liste de contraintes (délais à minimiser ou à équilibrer, liens à éviter,…). Ce module a fait l'objet de la thèse de doctorat d'Étienne Lepercq.

- Filtrage de Netlist : Ce module se situe entre le module de connexion aux outils CAD et le WaferConnect. Son objectif est de filtrer les netlists des composants inutiles (e.g.

résistances, condensateurs, connecteurs) pour les adapter au WaferBoard. Il a été implémenté par Karim Baratly dans le cadre de son projet de maîtrise.

Ces modules ont été écrit essentiellement en C++ à l'aide des librairies Qt, STL et Boost [12] [13] [14] pour assurer la portabilité Windows / Linux.

1.2 Prise en charge logicielle des systèmes électroniques

Le logiciel WaferConnect doit pouvoir piloter le matériel. Pour cela il est nécessaire de passer par plusieurs couches de communication : un ordre de configuration est transformé en commande JTAG puis empaqueté pour enfin être transmise aux cartes de contrôle (via USB ou Ethernet).

Un pilote de périphérique (ou *driver*) peut se définir comme un logiciel destiné à contrôler un périphérique matériel, interne ou externe [15]. On peut le voir comme une interface entre le logiciel et le matériel. Cela permet aux logiciels utilisant un tel pilote d'accéder au matériel sans avoir à se préoccuper de la manière dont il doit être physiquement contrôlé. Les pilotes sont généralement étroitement liés au système d'exploitation. Ils obtiennent ainsi les droits nécessaires à leur bon fonctionnement, contrairement aux logiciels en mode utilisateur. La Figure 1-8 rappelle brièvement l'organisation générale d'un système informatique commun afin de situer ces éléments par rapport au système d'exploitation.

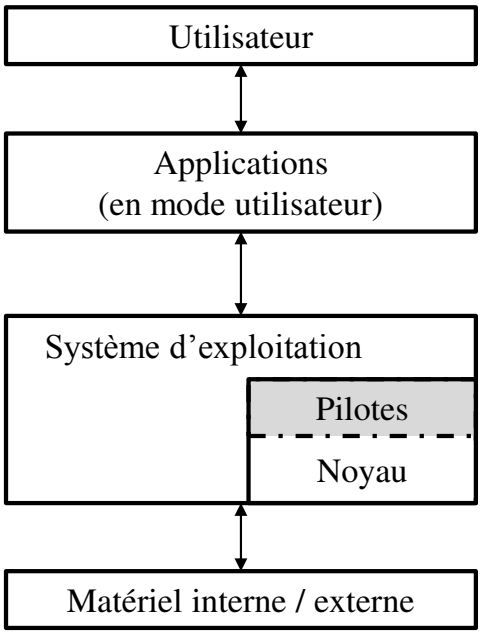


Figure 1-8: Organisation simplifiée d'un système informatique.

Pour des raisons de sécurité, les logiciels en mode utilisateur ne peuvent accéder directement au matériel. Cela permet de s'assurer que l'accès est effectué correctement, par l'intermédiaire du pilote adapté. Pour accéder directement au matériel, il est donc nécessaire de demander certaines autorisations au système d'exploitation.

Pour cela, le pilote est inscrit comme membre de confiance, voire comme module du noyau du système d'exploitation [15][16]. Pour augmenter cette sécurité il est de plus en plus fréquent que les pilotes soient signés électroniquement. Cela pour éviter qu'un logiciel malveillant puisse s'exécuter dans la zone de confiance du système d'exploitation.

Un cas particulier est celui du « *class driver* » qui est un pilote générique. Ce type de pilote est un pilote de base pouvant être utilisé pour un certain nombre de périphériques différents comme par exemple les pilotes USB HID [17].

Il est ainsi possible qu'un pilote s'appuie sur plusieurs autres pilotes pour fonctionner comme présenté à la Figure 1-9. Il est alors conçu comme une surcouche : il peut nécessiter de garder ou non ses privilèges vis-à-vis du système d'exploitation, et faire appel aux autres pilotes comme le ferait un logiciel en mode utilisateur. Dans tous les cas il doit

s'enregistrer comme pilote auprès du système d'exploitation car c'est en passant par celui-ci que les logiciels en mode utilisateur peuvent accéder aux fonctionnalités offertes par le pilote. Nous ne nous étendrons pas plus sur la programmation de pilotes car cette brève étude nous a guidés vers la réutilisation de pilotes existants.

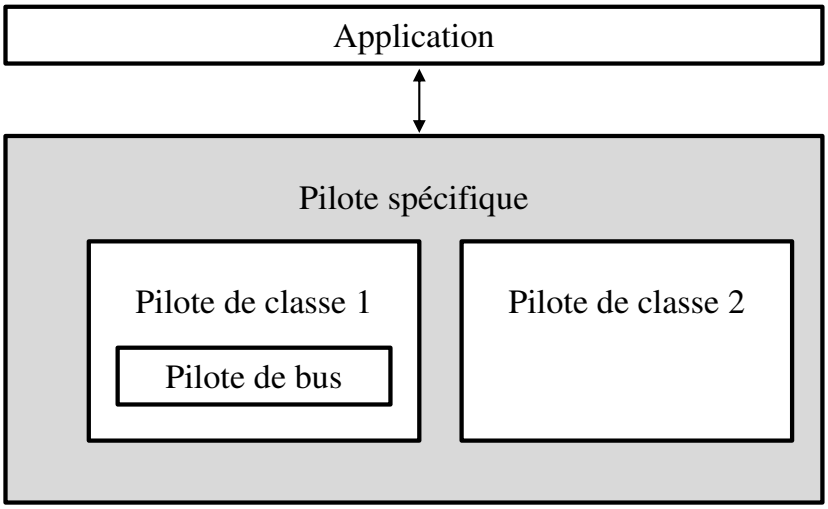


Figure 1-9: Exemple d'encapsulation au sein d'un pilote.

1.3 Modèle de conception logicielle

Lors de la réalisation d'un logiciel d'envergure, les problèmes à résoudre sont multiples. Un petit projet peut facilement être architecturé à la volée. Mais plus le projet grossit, plus la nécessité d'organisation devient importante. Il est, sinon difficile voire impossible, de continuer le développement et l'apport de corrections sans entraîner l'apparition de nouveaux problèmes. Ces aléas se manifestent très rapidement. Dans l'éventualité où le développement s'achève, il devient rapidement impossible d'assurer la maintenance et les évolutions d'un tel logiciel. Ce mémoire portant en grande partie sur la conception de l'architecture du logiciel WaferConnect, il est raisonnable d'introduire les outils disponibles comme supports à la conception d'une architecture.

Tout d'abord il est important de préciser que nous nous situons ici au niveau de l'architecture de l'application. Celle-ci nécessite de considérer le logiciel et ses composants dans leur globalité, ainsi que les besoins auxquels ils doivent répondre [18]. L'architecture

des algorithmes logiciels ne sera pas abordée ici. Bien que cela soit parfois lié au choix d'architecture au niveau applicatif, le choix d'un patron de conception pour la réalisation d'un algorithme donné a été laissé aux professionnels ayant réalisé l'implémentation.

L'abord scientifique du sujet est délicat car l'architecture est en partie une question de point de vue. Elle dépend fortement de la valeur que l'on attribue aux paramètres d'entrée (contraintes et objectifs). Un modèle de conception n'a de valeur scientifique que par sa généricité et l'obtention de résultats nécessite son utilisation dans un grand nombre d'applications. Ce n'est donc pas notre objectif. En effet, le besoin à l'origine de ce travail se résume à un seul logiciel : WaferConnect. Notre problématique est donc de proposer des choix de conception adaptés à ce besoin et non un nouveau modèle de conception générique dont on peut s'inspirer dans de multiples situations. Les prochaines sous-sections listent des modèles d'architecture qui ont été considérés dans l'élaboration du logiciel WaferConnect.

1.3.1 Architecture client-serveur

L'architecture client-serveur [19] est une architecture assez générique. Elle sert de base à la plupart des systèmes distribués et un grand nombre d'architectures en sont issues. Nous nous intéressons ici à l'organisation générale de ce modèle de manière à introduire l'architecture en couche. Une architecture client-serveur simple définit trois éléments : un serveur, un réseau et un ou plusieurs clients. Le serveur représente un fournisseur de services. Habituellement il est utilisé pour le calcul, le stockage et le traitement des données. Il est généralement connu du client car c'est le client qui va initier la communication. Le réseau assure la communication entre le serveur et le ou les clients. Le client, demandeur d'information va transmettre sa requête au serveur via le réseau. Le serveur traite la requête, renvois l'information demandée (si la requête est valide) et se charge habituellement de la présenter à l'utilisateur. La Figure 1-10 schématise l'architecture de base.

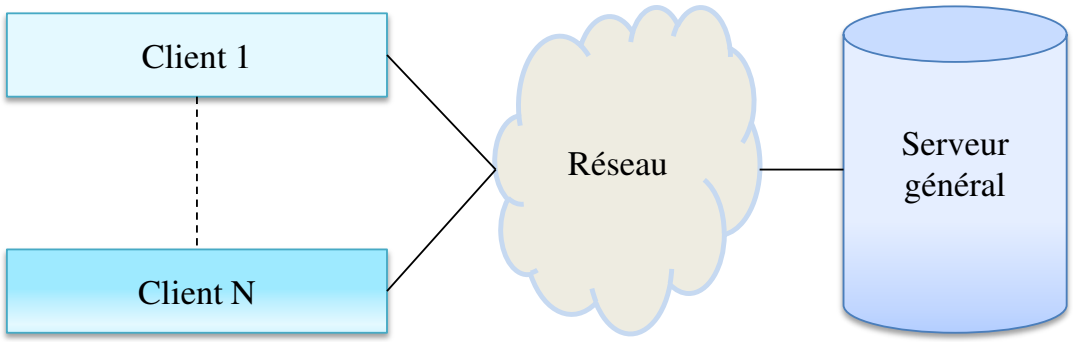


Figure 1-10: Architecture client serveur-simple.

C'est le modèle typique d'un serveur web simple : le serveur attend les requêtes client transitant à travers Internet et y répond. L'application client, par exemple le navigateur internet, se charge de présenter le résultat.

Le modèle s'élargit lorsqu'un serveur peut aussi faire office de client. En chaînant cette représentation il est possible d'obtenir des architectures très variées. Il est à noter que le concept du réseau utilisé ici n'implique pas nécessairement la présence de plusieurs machines physiques. Dans ce cas, la présence du réseau peut être optionnelle.

1.3.2 Architecture en couches

L'architecture en couches [20] est une extension de l'architecture client-serveur dans laquelle on insère des clients/serveurs intermédiaires. Chaque serveur intermédiaire est le client d'un ou plusieurs serveurs. Il est à noter qu'un client de la couche N ne peut accéder qu'à un serveur de la couche N-1. On obtient une chaîne linéaire, d'où la notion de couches. Cela peut tout de même donner des architectures très variées.

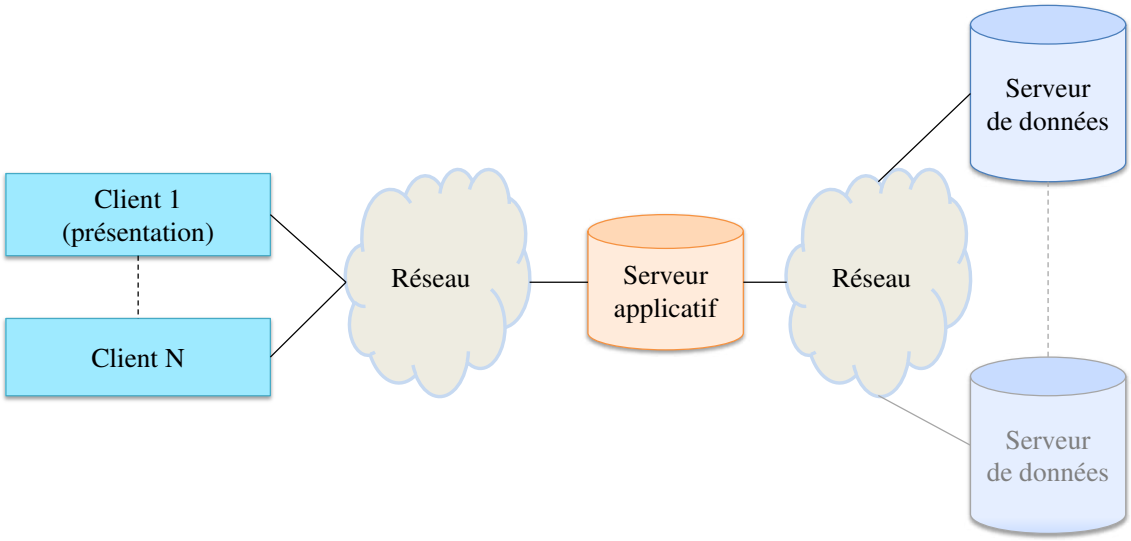


Figure 1-11: Architecture à trois couches.

La Figure 1-11 présente l'architecture en couche la plus utilisée : la « Three-tier ». Elle s'adapte à un grand nombre de situations. Elle est particulièrement adaptée à la conception de systèmes de services distribués. Ses trois couches sont formalisées ainsi :

- Couche présentation : c'est le client final, il ne fait pas office de serveur. Son rôle est d'assurer la présentation des données à l'utilisateur.

- Couche logique ou application : c'est la couche qui se charge du traitement des données, comme l'organisation des résultats. Cette couche est à la fois un serveur, qui communique avec la couche présentation, et un client qui communique avec la couche données.

- Couche données : c'est la couche « base de données », elle ne fait office que de serveur. Elle n'est pas accessible directement à la couche présentation car la couche logique se charge du contrôle d'accès.

Cette architecture est très répandue dans le développement d'applications web de commerce. En effet elle permet de regrouper plusieurs services distants via la couche

application. De plus les serveurs de la couche de données peuvent répondre à plusieurs clients ce qui rend l'architecture très flexible.

Le service bancaire illustre bien cette architecture. La couche présentation est assurée par le navigateur Internet, le serveur web de la banque est la couche logique qui s'appuie sur la couche données englobant la base de données d'informations de compte mais aussi les bases de données des fournisseurs de carte de crédit ou des informations boursières en fonction des services souscrits par l'utilisateur qui effectue la requête.

1.3.3 Architecture Modèle-Vue-Contrôleur

L'architecture Modèle-Vue-Contrôleur (MVC) [21] peut s'expliquer simplement. L'objectif est toujours de séparer la présentation, les données et leur traitement. Le modèle représente les données et l'intelligence de traitement de ces données. La vue est la représentation du modèle destinée à l'utilisateur. C'est la GUI d'une application. La vue peut interroger le modèle pour se mettre à jour. Ce point diffère beaucoup des architectures présentées précédemment. Enfin, le contrôleur joue le rôle d'interface entre l'utilisateur et le modèle. Il contrôle la vue et le modèle en fonction des requêtes utilisateur. La Figure 1-12 schématise les interactions entre les différents modules de l'architecture.

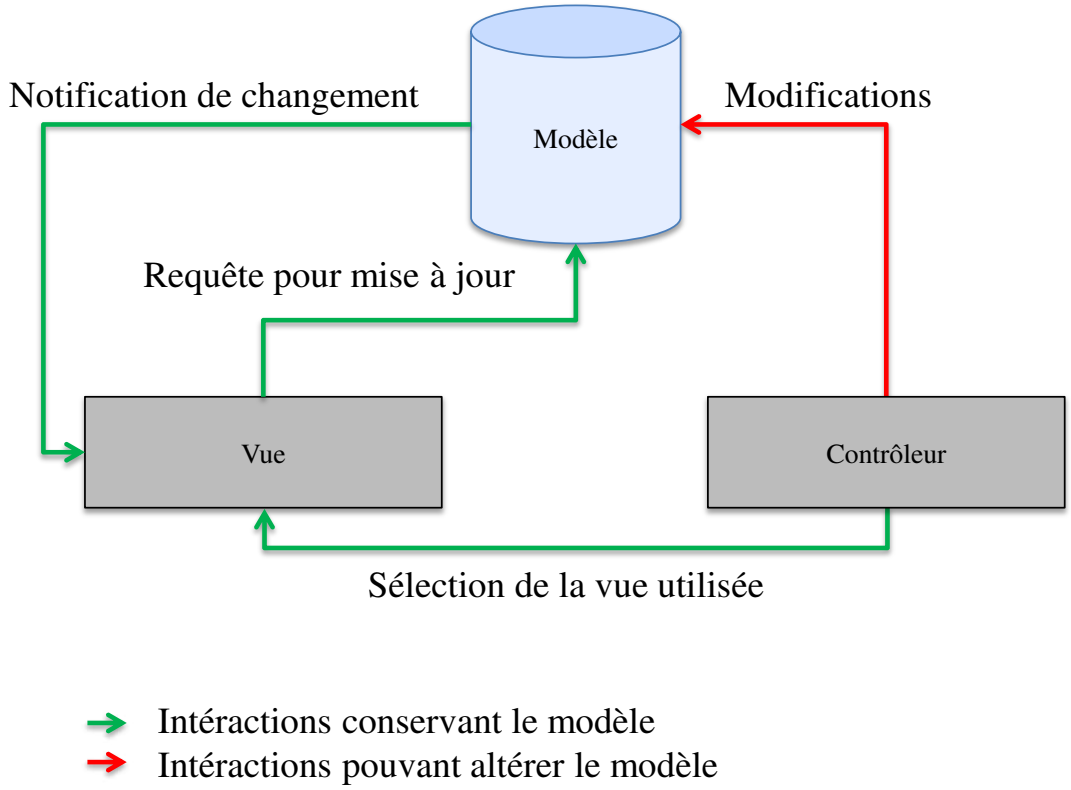


Figure 1-12: Intéractions Modèle Vue et Contrôleur.

1.4 Techniques de programmation graphique

L'affichage « temps réel » de données vectorielles volumineuses a toujours représenté un défi pour les développeurs logiciels. Dans le cadre du développement de l'outil CAD de pilotage pour le WaferBoard, le volume de données à prendre en charge est conséquent. Nous présentons ici les différentes techniques couramment utilisées en infographie, leurs avantages et leurs inconvénients.

1.4.1 Comparaison entre dessin vectoriel immédiat et image matricielle

La représentation vectorielle consiste à décrire l'image sous forme de points, de lignes, de courbes et de polygones. Un inconvénient majeur lié à la représentation vectorielle des images réside dans le fait que ce format de données ne se prête pas à l'application des techniques de traitement d'image. Cette particularité ne sera qu'évoquée car elle ne fait pas

partie de nos besoins et ne constitue donc pas un problème. Cela est donc la représentation la plus naturelle dans notre cas.

Une image matricielle (aussi appelée Raster) emmagasine les données sous forme de matrice de points (pixels). Chaque point est représenté par les informations d'intensité et de chrominance ou plus généralement par une valeur d'intensité pour les trois couleurs primaires par synthèse additive : le rouge, le vert et le bleu. Cette technique est utilisée pour les affichages possédant un tampon de données (frame buffer) [22] comme la plupart des moniteurs actuels. Elle est aussi employée comme technique d'encodage de base pour les photos numériques ou la représentation de cartes sur Internet (par exemple les cartes proposées par GoogleMap pour les cartes satellites ou aériennes). Dans un cas comme le nôtre, l'image matricielle n'est pas la technique de représentation native (nous possédons des données vectorielles), il est nécessaire de créer une image matricielle de « référence ». Cela peut prendre un certain temps et engendrer une consommation supplémentaire de mémoire pour stocker cette image matricielle. Une fois que l'image matricielle est générée à partir de sa représentation vectorielle, il est simple et rapide de l'afficher.

Pour assurer une modification aisée, l'image matricielle d'origine doit être stockée en mémoire. L'agrandissement et la réduction sont rapides et permettent d'obtenir des représentations à différentes échelles.

Par contre, ces modifications amènent une perte de précision (pixellisation sur un agrandissement, écart entre les objets incorrect) par rapport à une image définie vectoriellement. Ces déformations sont mises en évidence à la Figure 1-13. Une augmentation de la résolution de l'image de référence peut permettre de limiter ces effets, mais cela se traduit par une augmentation importante de l'empreinte mémoire de ladite image.

Dans le cas de notre application, la résolution minimale nécessaire pour maintenir l'information exacte est de 12 × 12 pixels par NanoPad (cela inclut le contour du NanoPad et la place nécessaire pour dessiner ses connexion au réseau JTAG et au WaferNet). Une résolution minimum de 16320 × 16320 pixels (266 342 400 pixels) est donc nécessaire pour représenter le WaferIC au complet. A titre de comparaison, les photos numériques actuelles ont une résolution de l'ordre de 10 000 000 pixels. Dans l'hypothèse où un pixel

est codé sur 24 bits, cela donne lieu à un volume de données brutes de 762 Méga Octets (Mo). En comparaison, les données vectorielles brutes (16 bits par coordonnée) occupent 58 Mo. L'utilisation de ce mode d'affichage mène à un compromis entre vitesse, qualité d'affichage et consommation mémoire.

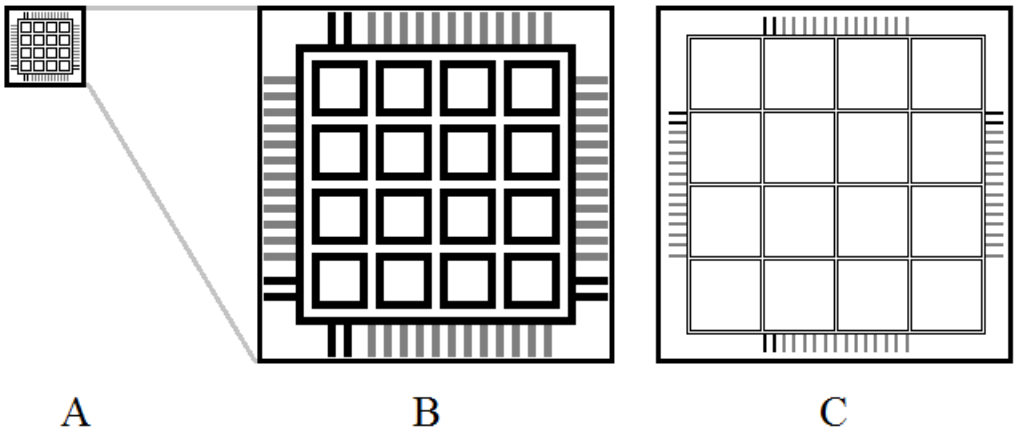


Figure 1-13: Exemple de distorsion induite par l'agrandissement d'une image matricielle, ici d'une cellule. A est l'original, B est l'agrandissement obtenu et C est l'image attendue idéalement.

Une technologie nommée Mipmapping [23] peut être utilisée pour limiter l'effort de calcul engendré par le redimensionnement d'une image matricielle. Plusieurs images sont générées correspondant chacune à un degré de l'agrandissement, puis une composition est réalisée afin d'afficher uniquement les fragments d'image nécessaires. Si un fragment n'est pas disponible, il est généré et stocké en mémoire. Cette technique est couramment utilisée dans les systèmes d'information géographique (SIG) [24]. Elle peut être à l'origine d'une consommation mémoire importante car ce n'est plus une mais plusieurs images qui sont à stocker. De plus les opérations de modification du contenu deviennent délicates parce que toutes les images existantes doivent être mises à jour. Il est en outre nécessaire de déterminer le niveau de précision souhaité pour définir à priori le nombre d'images intermédiaires à utiliser. Cette technique prend tout son intérêt dans le cadre d'applications client / serveur de consultation, où seul le serveur stocke l'image source et ses réductions. Le client récupère alors l'information nécessaire à la volée en fonction des besoins.

1.4.2 Technologie d'accélération graphique matérielle

Le rendu vectoriel consiste à dessiner chaque image à l'aide des informations vectorielles, et non à partir d'une image pré-calculée comme présenté ci-dessus. Ce type de dessin permet d'obtenir des agrandissements très fidèles : les proportions sont conservées et les contours des formes restent nets et précis. La Figure 1-14 en est un exemple. Cependant, pour parvenir à ces résultats, un grand nombre de calculs matriciels peuvent être nécessaires en fonction de la complexité de l'image.

Par exemple, une simple translation de l'image nécessite une multiplication matricielle par sommets présents dans l'image. Si l'on considère les liens (JTAG et WaferNet) comme des segments possédant 2 sommets et des éléments (réticules, cellules et NanoPads) comme des carrés définis par 4 sommets, la translation de l'image du circuit engendre 7 578 000 multiplications matricielles! Le fait que ces calculs fassent intervenir des nombres à virgules flottantes peut aussi avoir un impact sur le temps de calcul.

Les processeurs à usages généraux (CPU) sont peu efficaces pour effectuer ce genre de calculs mais s'avèrent suffisamment rapides pour effectuer ce travail pour des scènes simples. Pour des scènes plus complexes, notamment en 3D, l'utilisation de processeurs graphiques (GPU) dédiés est généralement requise. Des solutions très efficaces ont vu le jour au cours des dernières années, notamment pour la modélisation et l'affichage 3D. Mais la présence d'un tel matériel dans les ordinateurs de bureau n'était jusqu'à présent pas garantie.

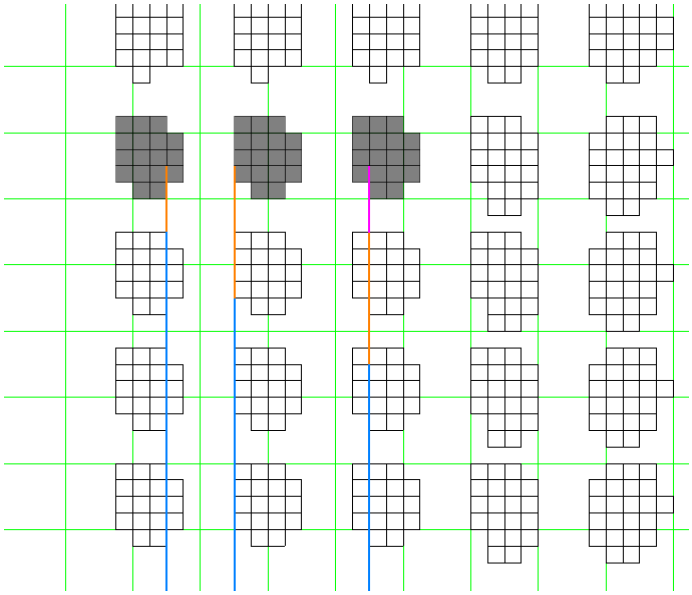


Figure 1-14: Exemple d'agrandissement d'une image vectorielle à l'échelle du NanoPad.

De nos jours, l'utilisation de matériel dédié est devenue beaucoup plus accessible, Intel produisant des CPU avec une unité de traitement graphique (GPU) intégré [25]. Les GPU travaillent directement avec un système de coordonnées cartésiennes à trois dimensions. Cela permet de dessiner nos données vectorielles de manière efficace. En travaillant dans le plan XY et en positionnant le point de vue le long de l'axe Z, il est possible d'obtenir l'affichage souhaité. Les GPU modernes sont capables de traiter plusieurs millions de polygones par seconde. Les calculs sont effectués à partir des données des sommets évoquées précédemment. Les objets à représenter étant soit des segments de droite, soit des rectangles, leur description dans ce contexte est adaptée. Une fois la géométrie traitée, chaque sommet peut se voir attribuer une couleur utilisée pour dessiner l'objet. Dans notre cas, la modification de la couleur d'un objet se limite à la modification de la couleur de 4 sommets.

À chaque dessin, le GPU cherchera les informations nécessaires dans la mémoire. Il est donc important de gérer les accès à la mémoire. Pour cela, il est proposé d'utiliser la technologie des « Vertex Buffer Object » (VBO) défini dans l'API OpenGL [26]. Cette technologie permet de définir les données visuelles directement dans la mémoire graphique au lieu d'utiliser la mémoire vive générale. Le processeur principal est donc moins sollicité

car les transferts de données ne sont effectués qu'en cas d'altération de la scène. Notez que dans ce contexte, la translation ou l'agrandissement de la scène ne change pas les données, mais seulement le point de vue.

1.5 Conclusion

Ce chapitre de revue de littérature a introduit le fonctionnement de la plateforme WaferBoard. Cette introduction bien que sommaire est nécessaire pour situer le contexte de développement du logiciel WaferConnect.

Ma contribution principale étant la conception de l'architecture générale de WaferConnect, les éléments de conception logicielle présentés posent les bases du chapitre suivant. L'architecture proposée ayant pour but de répondre efficacement aux besoins logiciels et aux contraintes du projet.

La section « prise en charge logicielle des système électroniques » présenté ici viendra supporter le chapitre présentant l'interface avec le matériel.

Enfin, la présentation des techniques de programmation graphiques a exposée les technologies servant de base à ma contribution technique.

CHAPITRE 2 CONCEPTION DU LOGICIEL WAFERCONNECT

Comme présenté dans le chapitre 1, un certain nombre de modules logiciels sont antérieurs à ce travail. Ils ont été développés indépendamment les uns des autres bien qu'un effort ait été fait pour qu'une majorité utilise la même structure de données pour représenter l'état du circuit sur tranche de silicium (hiérarchie, indexation et état des registres). Ce chapitre présente le processus de conception générale du logiciel WaferConnect ainsi que le résultat des travaux destinés à favoriser l'intégration des modules existants. La conception que nous proposons ici inclus l'architecture logicielle mais aussi l'adaptation des structures existantes comme le flot de travail et la structure de données. Le processus couvre aussi les besoins d'intégration spécifique au projet. Certains aspects techniques de ces modules ayant fait l'objet de publications scientifiques antérieures ne seront pas développés ici.

Le chapitre est organisé de la manière suivante : les besoins logiciels seront d'abord exposés, puis les adaptations du flot de travail existant à ces besoins seront détaillées. Une solution d'architecture générale prenant en compte les besoins des différents modules existants sera proposée. Enfin la création et la spécification d'une structure pour l'interface utilisateur seront présentées et discutées, suivie d'une synthèse des résultats obtenus.

2.1 Vue d'ensemble du logiciel WaferConnect

Le projet s'inscrit dans le cadre du prototypage rapide où l'objectif consiste à mettre en fonction un système électronique ou l'équivalent d'un PCB en seulement quelques minutes. Pour y parvenir, de nouveaux algorithmes ont dû être élaborés car même pour les ordinateurs de bureau actuels, le volume de données inhérent pour la représentation du WaferIC fait que certaines tâches demandent un temps de calcul non négligeable. Outre la recherche de performance temporelle, le logiciel a été conçu pour offrir une prise en main simple et rapide de la plateforme de prototypage, tout en offrant une gestion efficiente d'une grande quantité d'informations et une capacité de contrôle étendue.

Les choix effectués en termes d'architecture sont essentiellement une manière logique de répondre à un besoin et aux contraintes que ce besoin crée. Pour cerner ces choix, il est donc nécessaire de comprendre les contraintes qu'ils permettent d'intégrer. De nombreux modèles existent, parfois fondamentalement différents, parfois très proches. Certains concepteurs se réfèrent à l'emploi rigoureux de ces modèles qui deviennent des guides pour leurs développeurs. Cette pratique n'est pas toujours exactement adaptée à la résolution de leurs problèmes, mais le choix d'un modèle générique effectué judicieusement assure la cohérence et la maintenabilité d'un projet, ce qui minimise les risques et représente un gage de qualité logicielle. Ce cas de figure est courant dans les systèmes web qui ont la particularité d'être majoritairement centrés sur les données.

L'approche que nous proposons ici diffère légèrement. La généricité des modèles communs est un atout pour une utilisation massive mais peut rapidement devenir un handicap si les contraintes d'entrée du projet sont importantes. En effet, chaque projet répond à des besoins spécifiques. L'adoption d'un modèle générique ajoute des contraintes (cette fois de conception) au projet et il est souvent nécessaire d'effectuer des compromis. Cela peut se traduire par un assouplissement des contraintes de performance, la revue à la hausse des spécifications de l'ordinateur hôte (cas commun dans le développement de serveurs) ou simplement l'adaptation de certaines fonctionnalités.

L'architecture logicielle a été développée en fonction des contraintes et objectifs suivants:

1. Gérer les interactions complexes avec le matériel (WaferIC, PowerBlock, BottomPCB);
2. Proposer un temps de réponse optimal dépendamment des opérations à effectuer;
3. Effectuer l'intégration des outils de recherche existants;
4. S'adapter facilement à toute modification des spécifications matérielles;
5. Assurer la continuité du travail malgré les changements fréquents au sein de l'équipe de développement (outil développé essentiellement par des étudiants);

6. Fournir une interface graphique (GUI) adaptée à des utilisations occasionnelles ou avancées.

Ces contraintes seront reprises par la suite lors de la présentation de solutions. Certaines des solutions à ces contraintes et ces objectifs sont les suivantes:

- Les mécanismes d'interaction avec le matériel doivent être isolés (répond à la contrainte 1);

- Le logiciel doit être pensé pour être évolutif (répond aux contraintes 4 et 5);

- Les processus critiques doivent faire l'objet d'une étude de parallélisation et d'efforts d'optimisation pour obtenir des performances acceptables (répond à la contrainte 2);

Les points présentés ci-après mettent l'accent sur les choix techniques et architecturaux liés au contexte du projet : la recherche d'une solution logicielle évolutive et performante adaptée aux défis de la microélectronique moderne. Après avoir présenté l'architecture matérielle et logicielle, nous détaillerons les éléments ayant fait l'objet d'une conception particulière pour résoudre un problème logiciel hors du commun. Que ce problème soit lié à une contrainte technique ou un besoin de l'utilisateur.

2.2 Contraintes sur flot de travail utilisateur

Le flot de travail utilisateur proposé initialement est exposé dans la section 1.1.2. Les différentes étapes peuvent être regroupées comme suit.

Les deux premières étapes du flot (Figure 1-4, page 10) visent à obtenir une plateforme matérielle diagnostiquée et opérationnelle. L'étape de démarrage (*bootup*) va permettre de tester la communication, récupérer l'état de la plateforme (hors tranche) et initier la séquence de mise sous tension de la tranche de silicum. Le diagnostic va permettre de détecter les éléments fonctionnels de la tranche de silicium, dont les chemins de

configuration JTAG, le réseau d'interconnexion WaferNet, les unités fonctionnelles de chaque cellule, etc.

Les deux étapes suivantes visent à récupérer les informations sur les composants déposés sur la surface du wafer. Dans un premier temps, les plots des composants en contact avec aux moins deux NanoPads sont détectées. Ces plots sont regroupés en empreintes puis identifiés. Cette identification est assistée et validée par l'utilisateur. En cas de problème lors de cette étape (composant endommagé ou posé sur une surface défectueuse), les composants peuvent être changés ou déplacés.

Les deux étapes suivantes consistent à importer les informations nécessaires au routage (netlist et contraintes sur le netlist) puis à effectuer le routage. Une fois ces étapes effectuées la plateforme est prête pour être configurée et utilisée.

Les dépendances entre ces étapes sont présentées par la Figure 2-1. Ces dépendances représentent les contraintes réelles sur le flot de travail utilisateur. La reconnaissance de package ne peut se faire qu'avec l'importation d'empreinte des composants et les données issues de la détection des plots. Pour effectuer la détection, le réseau JTAG du WaferIC est activé pour configurer itérativement les NanoPads. Ce réseau doit donc avoir été préalablement diagnostiqué. Enfin, avec la connaissance de l'état du WaferNet et de la position des plots des UIC, le routage est effectué selon la netlist et ses contraintes.

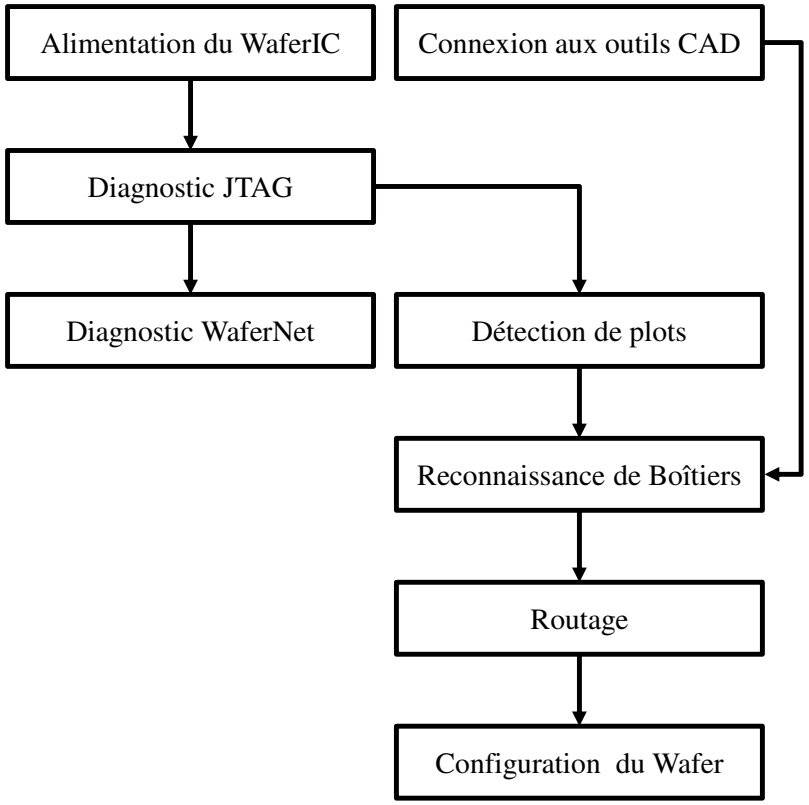


Figure 2-1: Dépendances des étapes du flot de travail utilisateur.

2.2.1 Adaptations réalisées

Pour répondre à cette problématique au niveau du logiciel, une architecture modulaire a été adoptée. Les dépendances sont prises en charge indépendamment de la partie fonctionnelle des étapes du flot de travail utilisateur (ou des commandes additionnelles comme l'exécution de scripts). Un nouveau module est ainsi défini : le module de gestion du flot de travail utilisateur (WorkflowManager). Ce module a pour but de garantir le respect des dépendances entre les étapes du flot de travail. Pour cela chaque étape est assimilée à une commande pouvant être autorisée (et donc exécutée lors de son appel) ou non. Chaque étape fonctionnelle du flot est encapsulée et construite pour demander une permission d'exécution au module de gestion du flot de travail. Sans cette permission, les ressources nécessaires à l'exécution de l'étape ne peuvent être utilisées. La Figure 2-2 propose une

représentation UML (Unified Modeling Language) du gestionnaire de flot de travail utilisateur tel qu'implémenté en C++.

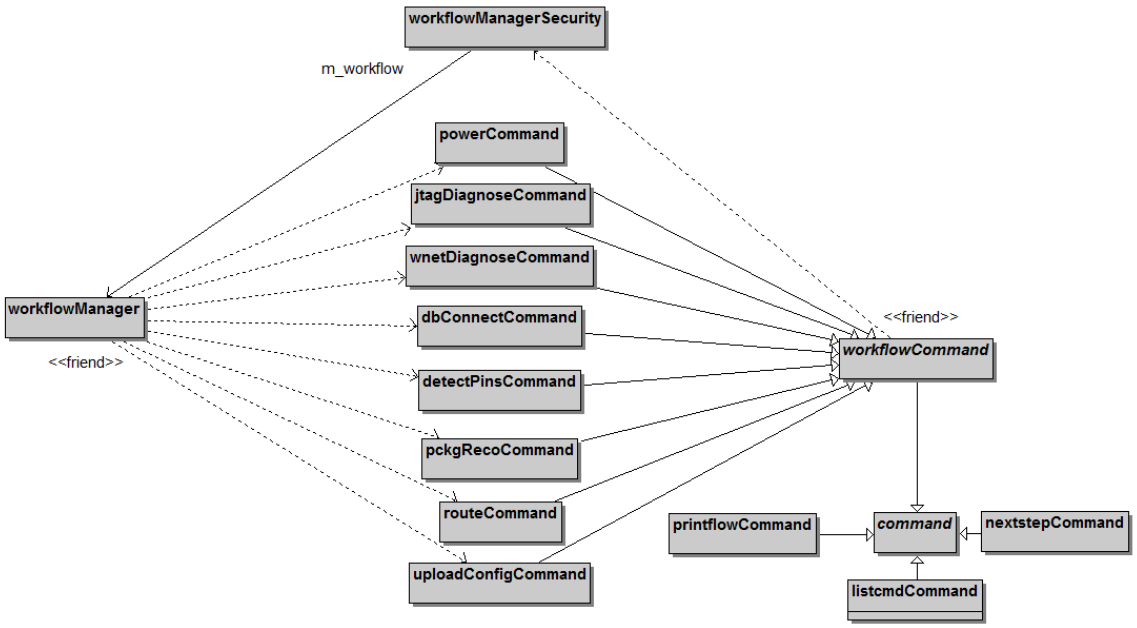


Figure 2-2: Diagramme de classes du gestionnaire de flot de travail de l'utilisateur.

Le gestionnaire de flot de travail permet un contrôle du flot de commandes et donc une sécurité d'utilisation. Le développeur ne pourra pas obtenir les ressources nécessaires sans passer par ce mécanisme. L'ajout d'une nouvelle commande est donc aussi sécurisé : car tant qu'elle n'est pas référencée, la commande n'obtiendra pas d'autorisation. Ceci a pour effet supplémentaire d'imposer une structure au développement de nouvelles commandes, tout en permettant un tel développement.

Enfin, à chaque invocation de commande, celle-ci est analysée et classée en fonction de son groupe. Parmi les groupes, on retrouve :

- Commande interdite (une liste de commandes à ne jamais exécuter est définie pour éviter les problèmes de sécurité du système, cela dans la mesure où les commandes de l'interpréteur système sont rendues disponibles)

- Commande système (relayée à l'interpréteur Linux/Windows)

- Commande interne (dont les commandes du flot de travail)

Une fois la commande identifiée, les contraintes associées à son groupe sont vérifiées et si tout est en règle, son exécution est effectuée. Par exemple, dans le cas d'une détection de plots (DetectPinsCommand, Figure 2-2), l'état du diagnostic est vérifié (l'étape de mise sous tension est testée implicitement si le diagnostic a été réalisé avec succès) et conditionne l'exécution de la détection de plots. Cette vérification assure ainsi la présence et la validité des informations d'entrée requises. Si cette assurance est absente, l'étape de détection de plots peut échouer ou proposer une carte des contacts erronée!

2.3 Architecture du logiciel WaferConnect

Une approche logicielle modulaire a été utilisée pour réduire la complexité du logiciel WaferConnect, tout en permettant une collaboration efficace entre les chercheurs et les développeurs. Cela répond en partie aux contraintes 1 et 3 présentées dans la section 2.1.

Le logiciel comprend des modules fonctionnels et des modules de contrôle. Cette répartition permet aux développeurs de logiciels ayant une connaissance limitée de la microélectronique de faire partiellement abstraction des contraintes matérielles. Ce besoin s'est fait ressentir très tôt dans le projet, certains développeurs nous ayant fait part de leurs difficultés à appréhender les contraintes de développement spécifiques aux besoins de configuration du WaferIC et aux protocoles de communication.

Ce découpage est rendu possible grâce à la création d'une API, nommée WAPI (Wafer Access API), qui encapsule les mécanismes de configuration de bas niveau [7][9]. Par exemple, WAPI permet aux développeurs de demander la configuration d'une route sans avoir à manipuler les registres d'état des cellules. Cette approche leur permet de penser au matériel en termes d'objets et de travailler ainsi à un niveau d'abstraction adapté au développement d'algorithmes sophistiqués sans se préoccuper des interactions complexes avec le matériel. L'utilisation de WAPI implique d'autres avantages et contraintes qui

seront présentés dans le chapitre suivant. L'architecture générale adoptée pour le logiciel WaferConnect est présentée à la Figure 2-3. Cette architecture est le fruit de plusieurs évolutions adoptées généralement à la suite de problèmes rencontrés lors du développement ou à l'apparition de nouvelles fonctionnalités matériel (par exemple le changement de la structure du WaferNet). Certaines évolutions de l'architecture ont été difficiles à adopter car elles représentaient une quantité de travail importante et l'apparition de nouvelles contraintes pour les développeurs, tout en répondant à un besoin à moyen ou long terme. Ce fut le cas de la formalisation de WAPI. De manière générale, l'architecture de WaferConnect est assimilable à une architecture à trois tranches (présentation – application – données). Cependant, comme expliqué précédemment (pour assurer l'adaptabilité du logiciel et l'intégrité des données), il nous a paru fondamental de distinguer le contrôle (validation des requêtes de l'utilisateur et des modules de traitement) et le traitement des données. Habituellement ces deux notions sont regroupées au sein de la couche application (présentées section 1.3). Le logiciel est donc découpé en quatre couches identifiées à la Figure 2-3 :

1. Présentation (composé de l'interface utilisateur présenté à la section 2.4);
2. Contrôle (représenté par le gestionnaire de flot de travail de l'utilisateur précédemment présenté);
3. Traitement (composé des modules externes présentés à la section 1.1.4);
4. Données (regroupant les modules encapsulés par WAPI détaillés à la section 3.1);

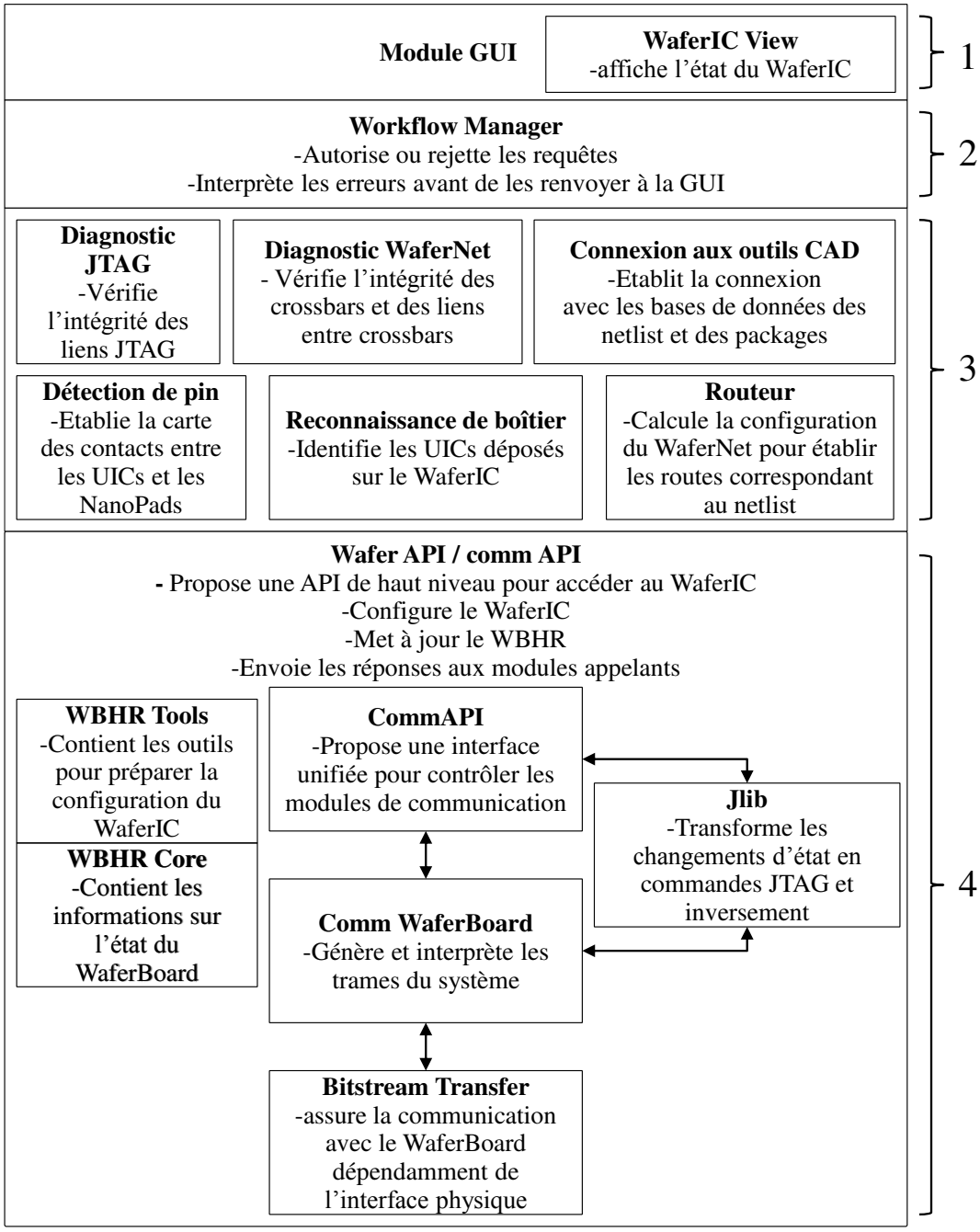


Figure 2-3: Architecture générale du logiciel WaferConnect et son découpage en couches.

Le modèle présenté n'est utilisé qu'à titre de guide. En effet, les fonctionnalités associées aux couches ne sont pas exclusives. La GUI va accéder périodiquement aux données pour

les afficher sans les dupliquer. Par rapport à une architecture trois-tiers, les accès entre les modules sont assouplis. Cet assouplissement permet d'isoler l'interprétation des résultats du matériel, tout en gardant ouverte la perspective d'une application client/serveur (utile dans le cas de coexistence de plusieurs prototypes sur la tranche de silicium). Les sous sections suivantes présentent la démarche adoptée pour structurer la couche « données » qui encapsule les mécanismes des communications avec le matériel et la structure de données.

2.3.1 Représentation logicielle du WaferBoard

La structure de données est un élément central du design du WaferConnect. Cette structure, nommée WBHR (*WaferBoard Hardware Representation*), vise principalement à permettre aux différents modules de se synchroniser et sera un passage obligé vers la configuration de la plateforme matérielle. Les attentes concernant le WBHR sont les suivantes :

- Structure partagée (la structure doit être accessible aux différents modules même si le développeur doit passer par WAPI pour y accéder);
- Gestion d'une grande quantité de données (les données propres à chaque cellule sont mémorisées par ce module);
- Rapidité d'accès (section 2.1, contrainte 2, p. 30, quelques secondes doivent suffire pour une lecture complète);
- Simplicité d'accès (ces données étant lues par plusieurs modules, il est intéressant de traiter la complexité d'accès une seule fois dans ce module);
- Disponibilité pour l'affichage des informations graphiques en temps réel (une latence est tolérable, mais il est important que l'utilisateur puisse visualiser rapidement l'état de la plateforme entre les différentes commandes qu'il demande, cela en raison du grand volume de donnée);

Le besoin d'une structure centralisée s'est rapidement manifesté (répondant à l'attente de structure partagée). Ce choix vient assez naturellement car le flot de travail étant séquentiel, les accès conflictuels sont limités. Il peut arriver qu'une mise à jour de l'affichage soit mise en attente pendant une écriture mais ce comportement n'est pas problématique.

Le WBHR est la troisième évolution de la structure de données utilisée à l'origine dans le projet de recherche DreamWafer. Cette dernière évolution permet de prendre en compte les besoins liés à l'intégration, l'affichage et la simplicité d'accès. Ces problématiques se sont posées lors de la spécification de l'architecture logicielle et suite à la refonte du système d'affichage. La structure a été étendue pour contenir toutes les informations nécessaires au suivi de la plateforme (température, pression, valeurs des courants d'alimentation, …) et à la configuration de la tranche de silicium, permettant ainsi d'élargir l'API pour mieux répondre aux besoins des autres modules, tout en conservant une possibilité de présentation visuelle des résultats.

Des optimisations supplémentaires ont eu lieu durant l'intégration. Ces optimisations visaient à réduire le temps de création de la structure de données au démarrage de l'application et les temps d'accès aux données, particulièrement dans le cas d'une lecture intégrale des données. En dépit des multiples évolutions, la structure du WBHR reste principalement basée sur la hiérarchie physique du WaferIC. La Figure 2-4 propose une représentation UML de cette structure.

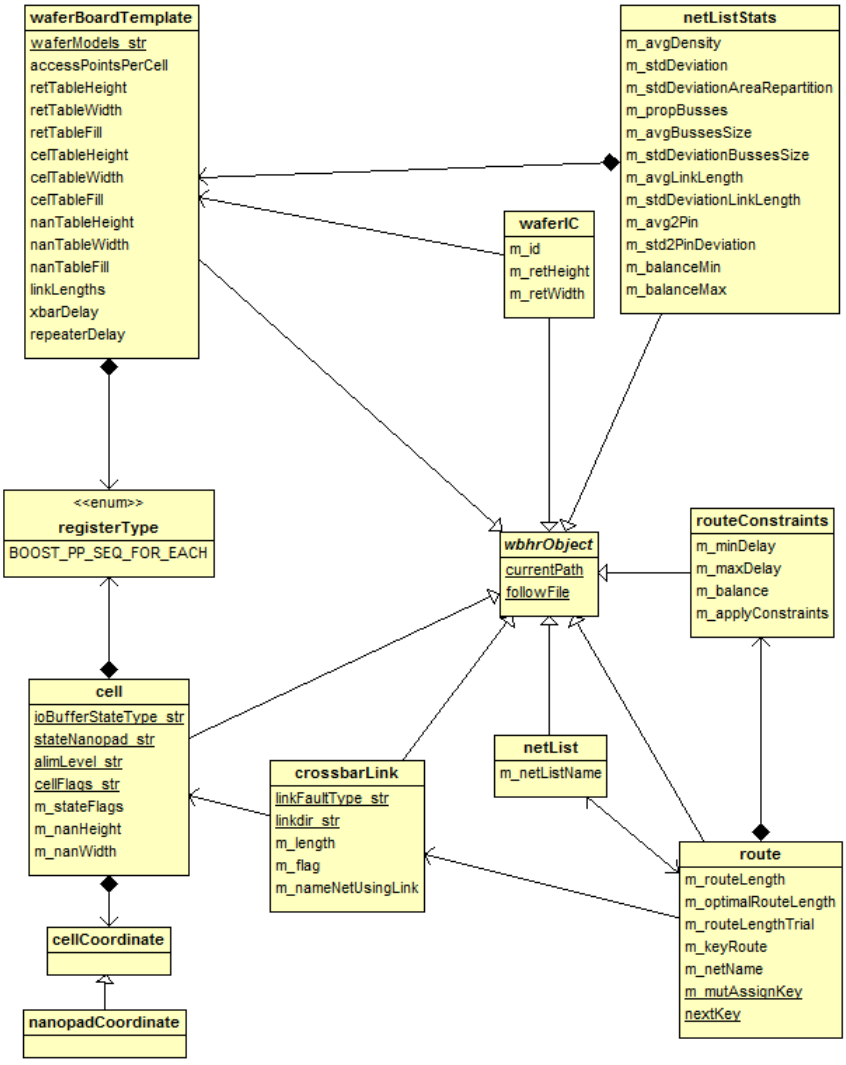


Figure 2-4: Diagramme de classes UML du WBHR (*WaferBoard Hardware Representation*).

La problématique de la conception d'une structure de données contenant un grand nombre d'enregistrements peut être résolue par l'utilisation d'un système de gestion de base de données. Bien que les systèmes de gestions de base de données soient très matures et très répandus, la solution qu'ils représentent n'est pas la plus adaptée à notre problème, cela pour plusieurs raisons. D'une part l'accès doit être rapide, pour cela un accès direct en mémoire est plus efficace qu'une requête à un système de gestion de base de données normalisée. D'autre part, la hiérarchie naturelle des données permet de se passer d'un autre

système d'indexation. Enfin la persistance des données à manipuler n'est pas critique. Dans le pire des cas, la perte des données entre deux utilisations peut être corrigée en effectuant à nouveau les étapes de diagnostic.

2.3.2 API d'accès au WaferIC et communication

La couche de communication du WaferBoard et son modèle logiciel sont rendus accessibles par un module dédié appelé Wafer Access API (WAPI). Il représente la frontière entre la partie logicielle abstraite (fonctionnelle) et la partie logicielle de pilotage (adaptation des accès au matériel). Le circuit sur la tranche de silicium est configuré grâce aux registres des cellules [1]. Le WAPI fournit des accès de haut niveau (commandes/actions) pour configurer la tranche et transforme ces demandes de configuration en état de registres. Il fait ensuite le nécessaire pour mettre à jour les registres de la tranche en utilisant le module de communication. Si la configuration est valide, elle est mise comme état « courant » dans la base de données (WBHR). Le module WAPI se charge également d'une partie du décodage des réponses de la tranche, retournant ainsi des résultats compréhensibles par le développeur. La configuration du matériel devient alors aisée pour les développeurs des modules au dessus de cette API.

Le module de communication englobe tous les éléments nécessaires pour accéder aux registres de la tranche et aux cartes de contrôle (BottomPCB et PowerBlocks). Il assure les fonctionnalités suivantes :

- Lecture des capteurs (température, pression, refroidissement);
- Lecture de la consommation électrique, tension et intensité de courant;
- Lecture de l'état de la tranche (lecture des registres);
- Commande du refroidissement (ventilation, contrôle des limites);
- Commande des régulateurs d'alimentation;
- Écriture dans les registres de la tranche;
- Coordination des séquences de lecture et d'écriture.

Le module de communication inclut plusieurs sous modules : l'API de communication unifiée (*comm API*), le module communication avec le WaferBoard (*Comm WaferBoard*), le module de transfert des bitstreams (Bitstream Transfer) et le module JLIB.

Le module *Comm API* constitue l'unique point de communication (entrée) avec le WaferBoard. Il est responsable de la récupération et du traitement de toutes les requêtes. La plupart des requêtes sont simplement redirigées vers *Comm WaferBoard* qui se charge de les formater en vu de leur routage vers l'élément matériel de destination. Si la requête est destinée au WaferIC, JLIB est mis à contribution pour produire le bitstream nécessaire qui sera envoyé à *Comm WaferBoard*.

Le module *Comm WaferBoard* est utilisé pour générer les trames de communication [7][27] qui permettent aux données d'être routées à travers le système : BottomPCB, PowerBlock et WaferIC. Le système utilisé consiste à créer des paquets dont l'entête décrit la destination suivante. Une trame peut donc encapsuler plusieurs entêtes dépendamment de la distance qu'elle doit parcourir. Un exemple de trame destinée aux PowerBlocks est présenté Figure 2-5, on y observe la trame (A) et son évolution au cours de son routage (B).

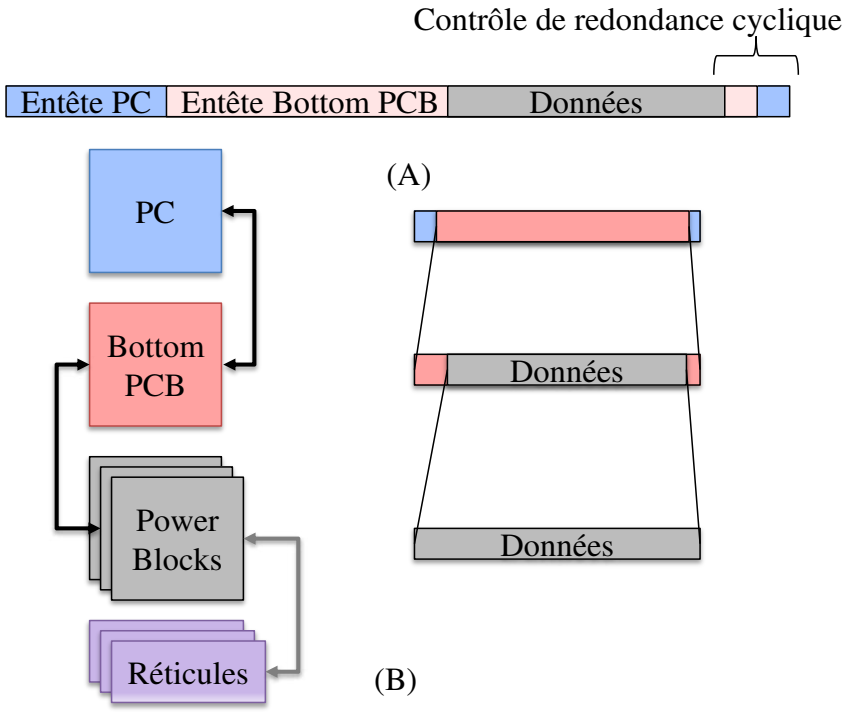


Figure 2-5: Exemple de trame à destination des PowerBlocks (A) et son évolution (B).

Le module *Bitstream Transfer* est lui chargé de transférer les données formatées au matériel via l'interface sélectionnée (USB, LAN, …). Il prend en charge la détection et la configuration de l'interface, l'évaluation de la version du matériel, la transmission des données, la régulation du trafic et la remontée des erreurs.

La configuration de la tranche se passe en deux étapes répétées au besoin, et cela par réticules. À chaque requête de configuration spécifique, un chemin contenant des cellules à configurer est déterminé. La bibliothèque JLIB contient les fonctions pour la conversion des informations de configuration de haut niveau contenues dans ce chemin en chaînes de commandes JTAG. Cette conversion génère deux types de chaînes JTAG. Les macro-chaînes spécifient le chemin permettant d'accéder aux cellules à configurer, alors que les micro-chaînes contiennent des instructions de configuration des cellules [9]. La chaîne de commandes JTAG résultante est ensuite compressée, encapsulée et envoyée au WaferIC par l'intermédiaire du BottomPCB puis des PowerBlocks. C'est le module de communication (*Comm API*) qui se charge de ces dernières étapes. La Figure 2-6 présente un exemple de chemin de configuration JTAG pour un réticule simplifié de 64 cellules (8*8).

Le chemin va de l'entrée (TSV TDI) à la sortie (TSV TDO) en passant par les cellules à configurer tout en évitant les cellules marquées comme inutilisable. On peut noter que certaines cellules font partie du chemin mais ne font pas parti des cellules à configurer. Ces cellules configurées en mode *bypass* (mode de la norme JTAG) permettent simplement d'accéder aux cellules voisines. S'il n'existe pas de chemin permettant de configurer toutes les cellules souhaitées dans le réticule, la configuration se fait en plusieurs étapes.

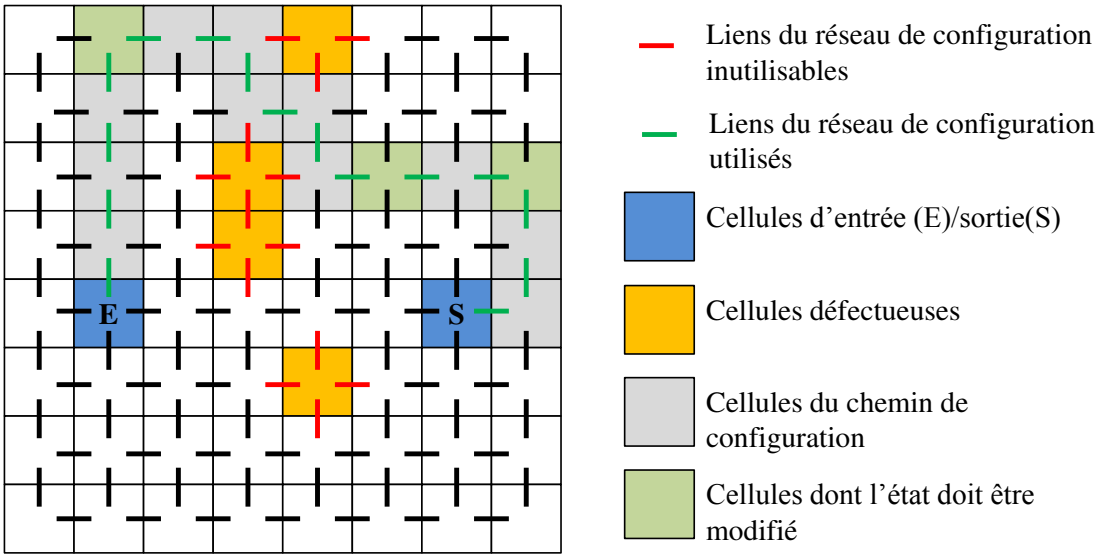


Figure 2-6: Exemple de chemin de configuration JTAG.

2.4 Interface utilisateur

Certains modules externes (routage et reconnaissance de boîtier) possédaient une interface graphique basée sur le kit de composants logiciels Qt. de Nokia. Cela a grandement guidé notre choix de kit de composants logiciels pour le développement de l'interface graphique de WaferConnect. De par l'utilisation de Qt., l'interface se veut très flexible. L'utilisateur aura le choix d'organiser son interface comme il l'entend.

Comme présenté ci-dessus, le logiciel doit permettre à l'usager un contrôle efficace de la plateforme matérielle. Les algorithmes disponibles, seuls, ne permettent pas un tel contrôle, même avec une interface rudimentaire comme l'accès en ligne de commande. L'utilisateur doit pouvoir gérer les défectuosités matérielles et les erreurs, qu'elles soient liées à la conception du circuit ou à une mauvaise manipulation. Il doit aussi pouvoir contrôler son flot de travail de manière intuitive et cohérente. Les contraintes étaient présentes, une des contributions de cette maîtrise a été de formaliser ces contraintes, de proposer une ébauche d'interface graphique y répondant pour enfin superviser sa réalisation.

2.4.1 Spécifications de l'interface graphique utilisateur

L'interface utilisateur graphique (GUI) est conçue dans l'objectif de satisfaire aux deux critères cruciaux : permettre à l'usager une utilisation rapide et simple avec la plate-forme matérielle grâce à des interactions intuitives et cohérentes avec le logiciel, et offrir une capacité de visualisation de données permettant une interprétation facile des données et une évaluation efficace des erreurs matérielles ou de manipulation.

Le module GUI du WaferConnect regroupe l'essentiel des fonctionnalités devant être accessibles à l'utilisateur. Pour simplifier son implémentation, le GUI a été découpé en sous modules :

- Une barre de menus standards permet d'accéder à la plupart des fonctionnalités et propose les options de configuration de l'interface graphique;

- Un espace réservé à l'affichage de l'état de la sélection, qui présente la sélection courante à l'utilisateur sous forme de listes avec une mise en évidence du « dernier objet sélectionné »;

- Un espace réservé à l'affichage de l'état des couches : les objets (cellules, réticules, UICs, …) sont regroupés par type suivant plusieurs couches distinctes. La liste des couches doit posséder une représentation graphique et doit être facilement accessible. Chaque couche possède deux propriétés modifiables par l'utilisateur dynamiquement : affichable et sélectionnable;

- L'explorateur d'objets fournit à l'utilisateur les informations disponibles sur un objet. Le mécanisme de sélection définit un objet comme « dernier objet sélectionné ». C'est cet objet qui doit être représenté dans l'explorateur. Certains objets sont liés hiérarchiquement. Il peut donc être intéressant de pouvoir explorer cette hiérarchie à travers l'explorateur d'objets;

- Le comparateur d'objets permet de comparer des objets de même type. Si plusieurs types d'objets sont présents dans la sélection, un système d'onglet doit permettre de regrouper les objets par type. Lors du défilement des propriétés des objets comparés, les propriétés sont mises en vis à vis;

- L'explorateur d'erreurs fournit à l'utilisateur les informations disponibles sur les problèmes qu'il a ou qu'il pourra rencontrer. Lors du clic sur une erreur, si la liste des objets en cause contient des objets affichables, l'explorateur d'erreurs doit mettre en évidence ces objets;

- La fenêtre de log/exécuteur de commandes affiche l'historique des commandes et permet la saisie d'une commande manuellement : c'est l'équivalent d'une console de commandes textuelles.

La présentation de l'état du flot de travail (présenté à la section 2.2), de l'état du WaferIC et de l'étape suivante dans le flot de travail viennent compléter la liste des éléments de l'interface graphique. Ce découpage est à l'origine du flot de données présenté à la Figure 2-7. On notera la présence du gestionnaire de flot utilisateur qui assure la validité des commandes passées au logiciel.

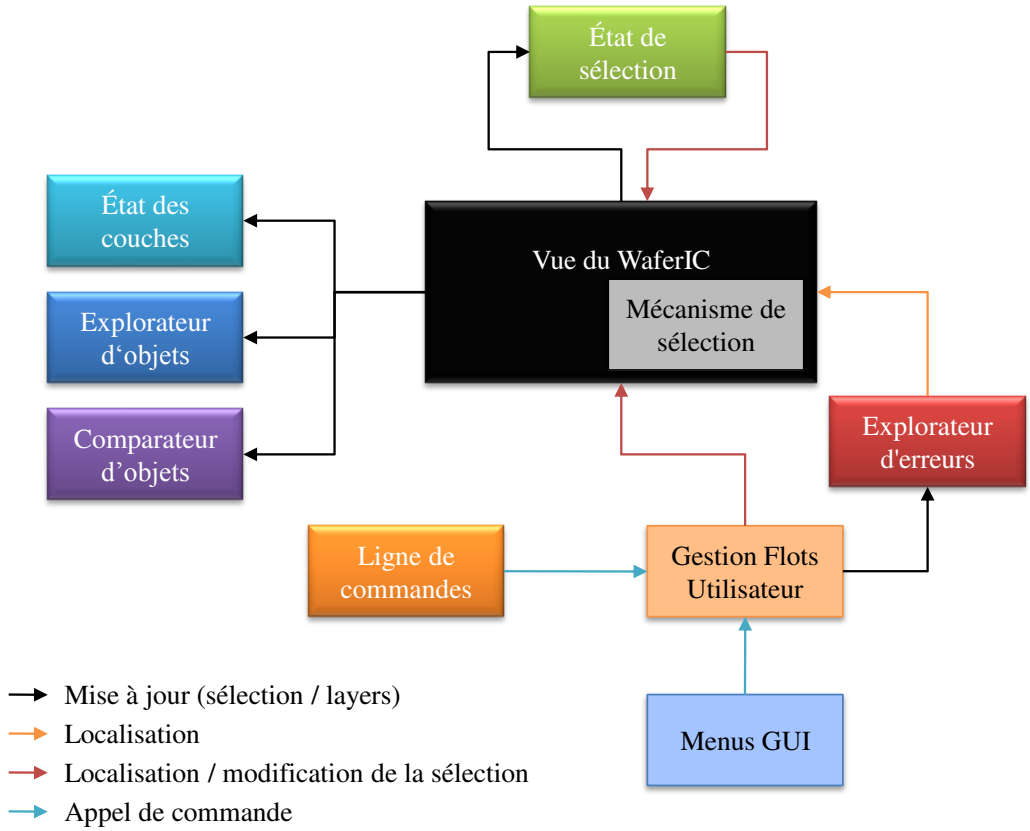


Figure 2-7: Interaction entre les sous modules GUI.

2.4.2 Organisation et implémentation du GUI

Comme la plupart des logiciels de CAO, WaferConnect va offrir de nombreuses possibilités. L'un des objectifs lors de la conception du logiciel est de réaliser une interface ergonomique tout en conservant une certaine simplicité malgré la complexité d'un tel logiciel. Pour cela, un menu commun et un système de vues sont utilisés. La vue principale (utilisée au premier démarrage) se veut claire et épurée : seules les interactions de base sont proposées. La gestion du flot de travail de l'utilisateur est mise en avant ainsi que l'affichage de la tranche et des problèmes potentiels. Cette vue présentée à la Figure 2-8 est complétée par une console de commande et un aperçu des objets sélectionnées. La conception du GUI a présumé que l'utilisateur était familiarisé avec les objets graphiques usuels tels la barre de menu, la barre d'état et les onglets.

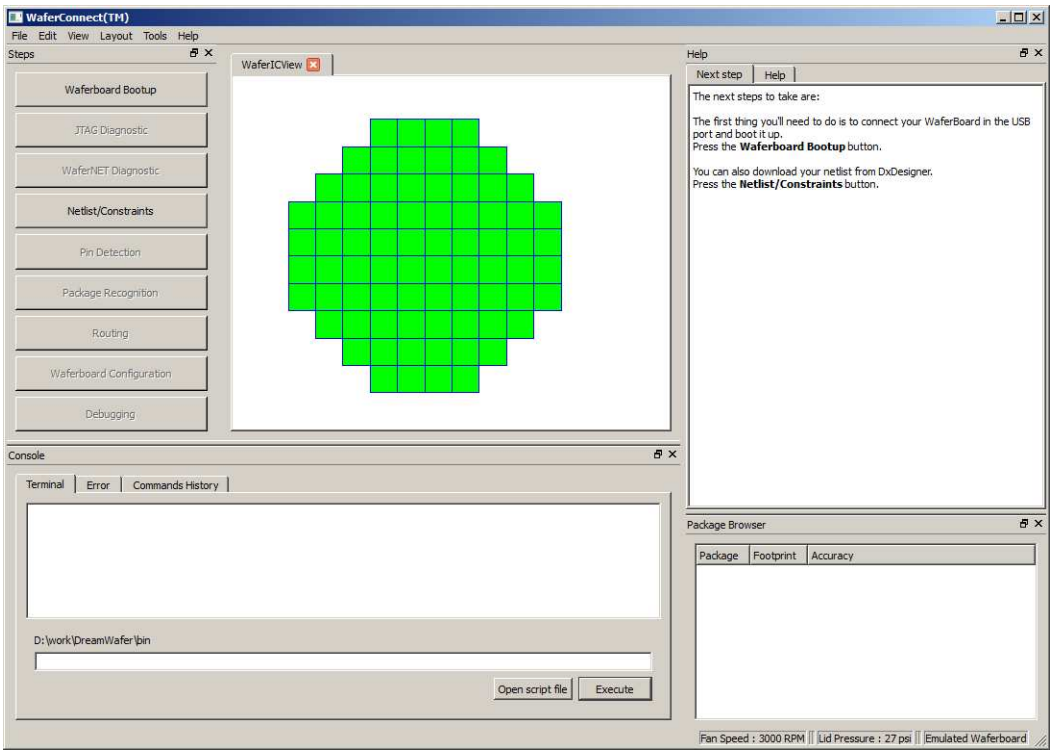


Figure 2-8: Environnement graphique de base de WaferConnect.

Les interactions avec la zone d'affichage sont limitées au nécessaire. Un clic gauche permet de sélectionner les éléments à étudier en détail. Un clic droit maintenu permet d'assurer la translation de la vue (fonction « *drag* ») et la molette permet de contrôler l'éloignement de

la représentation de la tranche (fonction « *zoom* »). Encore une fois l'objectif est de conserver des actions simples et intuitives.

Une vue « avancée » présentée à la Figure 2-9 est disponible. Celle-ci est conçue pour l'utilisateur expérimenté. Elle offre un accès rapide à tous les outils usuels et propose les informations concernant la sélection active. Enfin la vue peut être entièrement redéfinie et sauvegardée selon les goûts et habitudes des utilisateurs. Ces options de personnalisation de l'espace de travail sont courantes pour les logiciels modernes de grande envergure et font partie des attentes de l'utilisateur. La plateforme étant probablement partagée, la gestion et la sauvegarde de plusieurs environnements sont incontournables. Cette personnalisation permet à l'utilisateur de se retrouver rapidement dans un environnement familier et être ainsi rapidement opérationnel.

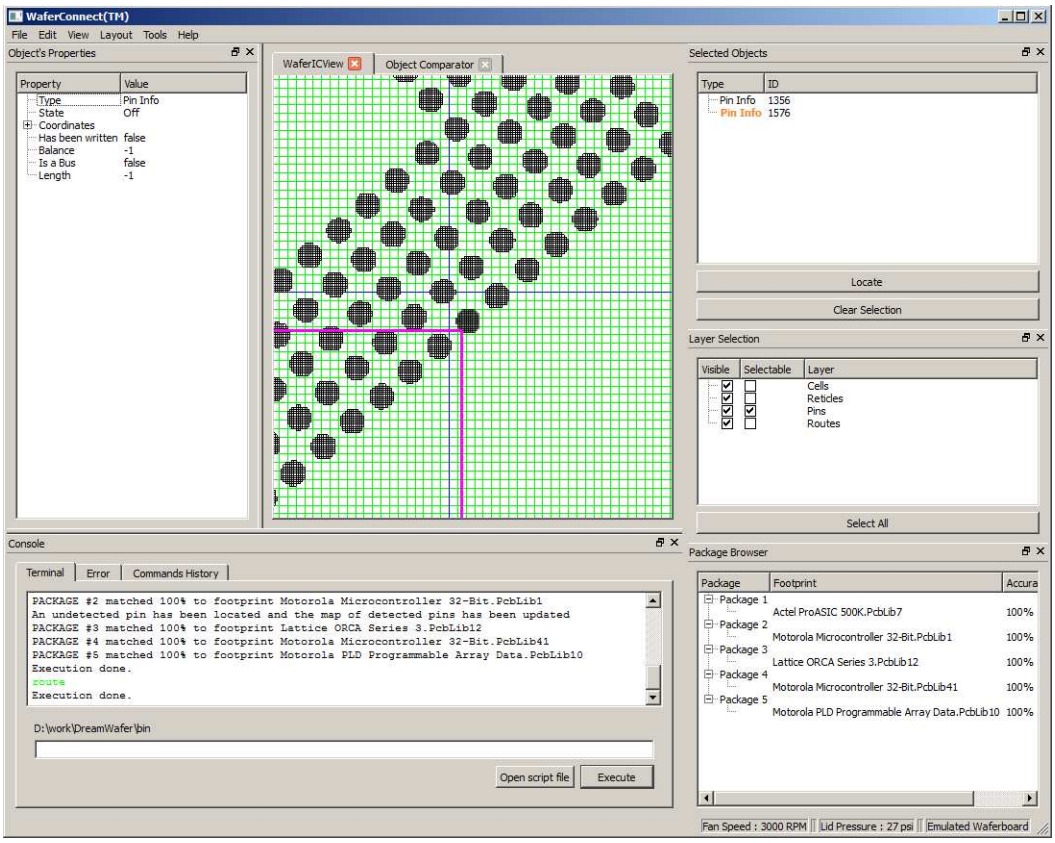


Figure 2-9: Environnement graphique avancé de WaferConnect.

Le logiciel WaferConnect s'adresse à deux catégories d'utilisateurs : les utilisateurs occasionnels et les utilisateurs expérimentés. La première catégorie est en attente d'une application rapide, simple d'utilisation et adaptable. La seconde catégorie attend une grande capacité de contrôle de l'outil et des informations les plus précises possibles.

Le but de l'utilisateur occasionnel est d'être rapidement efficace. Pour cela, il appréciera un outil qui s'intègre facilement dans son environnement de travail, sans le ralentir avec des détails qu'il jugera mineurs. Les mécanismes de génération et d'exécution de scripts répondent en partie à ce besoin. L'affichage complet de l'état de la tranche et de l'état du flot de travail contribue à un suivi visuel rapide de l'état de la plateforme. L'enregistrement de l'environnement et l'interface avec les outils de Mentor Graphics viennent compléter la réponse aux besoins de l'ingénieur.

Le point de vue de l'utilisateur expérimenté est sensiblement différent : celui-ci recherche en général un grand contrôle de l'outil afin de caractériser les paramètres de ses expérimentations pour les rendre reproductibles. Le moindre avertissement doit être maîtrisé. La connaissance de l'état de la plateforme est indispensable. Il attend de plus que les résultats fournis par l'application soient exhaustifs. Si la plateforme est incapable de fournir les données recherchées, elle devient inutile. Pour cela, l'état de tous les objets visibles de la tranche peut être affiché. Les routes peuvent être modifiées manuellement et un comparateur d'objets est disponible. Le système de gestion d'erreurs et un journal d'événements détaillés viennent achever la liste des outils disponibles.

Un interpréteur de commande vient s'ajouter à ces outils (Figure 2-10). Il permet d'exécuter toutes les actions du flot de travail à l'aide de commandes textuelles. Il permet de plus l'exécution de scripts et l'interprétation des commandes usuelles de l'interpréteur local (comme cmd pour les systèmes Windows, bash, sh, etc. pour les systèmes linux/Unix). Cet interpréteur permet à l'utilisateur de ne pas avoir à assimiler un autre système de commandes. L'application reste multiplateforme. En contrepartie, les scripts nécessiteront un portage s'ils contiennent des commandes destinées à l'interpréteur local et que celui-ci change.

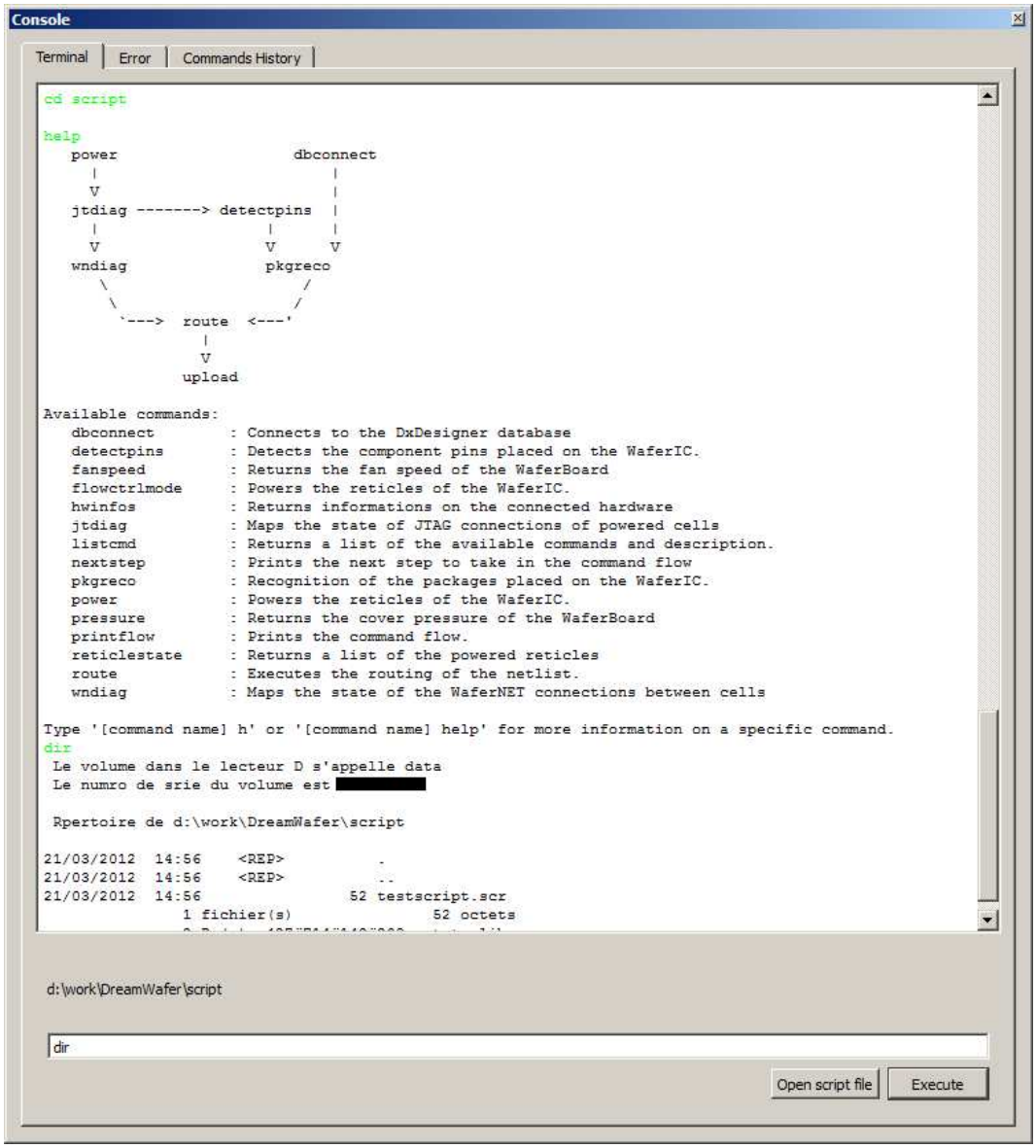


Figure 2-10: Interpréteur de commandes de WaferConnect utilisé sous Windows.

Les commandes spécifiques à l'application bénéficient d'une intégration large : en cas d'exécution à partir de la console, ou d'un script, la GUI est mise à jour et inversement, un appel à partir d'un élément de la GUI (par exemple un bouton) génère l'exécution d'une commande, son inscription dans la console et son ajout au script de session. L'application reste ainsi dans un état stable et l'information soumise à l'utilisateur reste cohérente malgré la redondance (console, boutons visuels et script). La Figure 2-10 présente un exemple du comportement de la console.

2.4.3 Affichage du WaferIC et prise en charge des erreurs

L'affichage du WaferIC fait l'objet d'un module à part entière dont les mécanismes sont présentés dans le quatrième chapitre. Ce module a pour objectif de présenter le WaferIC à l'utilisateur de manière graphique. C'est un élément majeur de l'interface utilisateur. Il est prévu pour venir s'intégrer au module GUI du WaferConnect. Les éléments visualisables du WaferIC comprennent les réticules, les cellules, les NanoPads, les plots des composants et leurs empreintes sur les NanoPads, les routes (qui sont représentées par un affichage partiel du WaferNet), les liens JTAG et les liens du WaferNet défectueux.

L'affichage et la sélection des éléments visuels du WaferIC reposent sur le système d'organisation par couches et le système de sélection du module GUI. Grâce à ce système de couches, seuls les objets présentant un intérêt pour l'utilisateur peuvent être affichés, évitant ainsi la surcharge de l'affichage. Cette technique permet ainsi d'offrir une information claire à l'utilisateur.

Contrairement aux PCB standards, la possibilité de configuration qu'offre le WaferBoard en fait un système destiné à plusieurs utilisateurs. Il est probable que l'utilisateur ait un accès temporaire à la plateforme. Il doit donc chaque fois reconfigurer son circuit, et cela le plus rapidement possible. Il doit être capable de réagir à l'état de la plateforme ou de ses composants.

Toute panne générale (matérielle ou non) doit être remontée rapidement à l'utilisateur (alimentation, refroidissement, réseau de communication sur PCB). En cas de défectuosité partielle de la tranche (cas normal d'utilisation), l'opérateur doit être averti de la présence des zones à problèmes ou inexploitables. Si ces zones sont importantes, cela lui permettra de les éviter au mieux lors de la réalisation de son circuit et du positionnement de ses composants.

Les pannes pouvant être corrigées humainement sont multiples. Par exemple, les cas de courts-circuits non attendus (poussières métalliques, composants trop proches,…) doivent apparaître clairement pour aider l'utilisateur à résoudre le problème. En effet, les algorithmes de protection lors de la mise sous tension de la plateforme ont pour but de limiter les risques, mais n'assurent pas un fonctionnement nominal quelle que soit la situation. De la même manière, les cas de mauvais contacts avec les composants doivent

être signalés avec précision (composants et numéros de plot en cause), pour aider à corriger la panne (augmentation de pression, repositionnement, nettoyage ou changement de composant endommagé physiquement).

Le choix adopté consiste d'une part à afficher une liste des problèmes rencontrés, avec leur sévérité, d'autre part à localiser les problèmes visuellement sur la représentation de l'état de la tranche.

2.5 Résultats

Les résultats de cette architecture sont nombreux bien que qualitatifs pour la plupart. Il n'est pas pertinent de présenter un temps de réalisation en raison de la diversité des intervenants et de leur qualification : le développement du logiciel WaferConnect a impliqué jusqu'à présent plus de 25 personnes, chercheurs, professionnels et stagiaires confondus.

Les sources du logiciel représentent 153 852 lignes de code réparties sur 691 fichiers. Bien que cet indicateur ne permette pas d'évaluer la performance ou la qualité du logiciel, il donne un ordre d'idée quant à la taille du logiciel développé.

Les figures présentées en partie 2.4 témoignent de l'aboutissement de l'interface utilisateur, ce sont en fait des captures du logiciel en fonctionnement et non un modèle préliminaire de design.

Le développement de couche « présentation » peut être considérée comme terminée : la GUI de base est complétée. Elle sera éventuellement enrichie lors de l'intégration de nouveaux modules.

Il en va de même pour la couche « contrôle ». Le logiciel est prêt pour l'intégration de nouvelles fonctionnalités. L'interpréteur de commandes supporte actuellement les commandes présentées par le Tableau 2.1. On peut noter que centaines commandes sont supportées par l'interpréteur mais pas leur fonctionnalité. C'est le cas par exemple de « jtdiag ». Cela signifie qu'un squelette a été développé pour préparer l'intégration du module fonctionnel.

La couche « traitement » qui regroupe les modules fonctionnels est la moins aboutie en raison de l'absence de certains modules. Cela ne représente pas de problèmes au niveau de cette couche, la problématique va se situer au niveau de l'accès à la couche « données ». Cela dans le cas où les méthodes d'accès aux données ne conviennent pas aux besoins du nouveau module à intégrer.

Pour cette raison, la couche « données » est considérée comme incomplète. Le développement initial est terminé mais l'intégration est en cours. Un autre facteur pouvant intervenir est l'évolution de la plateforme matérielle.

Tableau 2.1: Commandes définies et supportées par l'interpréteur de WaferConnect.

Commande	**Description**
dbconnect	Établie une connexion a la base de donnée DxDesigner.
detectpins	Effectue la détection de plots.
do	Exécute le fichier script passé en paramètre.
fanspeed	Affiche l'état des ventilateurs du WaferBoard.
flowctrlmode	Contrôle la mise sous tension des réticules.
help	Affiche l'aide.
hwinfos	Affiche l'état de la connexion et la version du matériel.
jtdiag	Effectue le diagnostic JTAG.
listcmd	Affiche la liste des commandes et leur description.
nextstep	Affiche l'étape suivante du flot de travail.
power	Met sous tension les réticules indiqués.
pressure	Affiche la pression exercée sur le WaferIC.
printflow	Affiche le flot de travail et les dépendances d'execution.
reticlestate	Affiche la liste des réticules sous tension.
wndiag	Effectue le diagnostic du WaferNet

Tableau 2.2: État du développement et intervenants.

Module logiciel	Auteurs	État
Bitstream Transfer	Nassir KASSANALY, Lloyd Ross SALVANT	Intégration en cours
Comm Waferboard	Thalie KEKLIKIAN	Complété
CommAPI	François RAJOTTE	Complété
Connexion outils CAD	Jérôme BARON	Complété
Détection de plots	Hai NGUYEN	Complété
Diagnostic JTAG	Jean Sébastien TURGEON	Développement en cours
Diagnostic WaferNet	-	Non développé
Gestionnaire de flot de travail	Thalie KEKLIKIAN, Jean-Christophe ST-PIERRE	Complété
GUI	Keven CHAUSSE, Hugo CARDIN, Yegor KHOMUTOV	Complété
JLIB	Jeremy TURCOTTE, Kada IM	Intégration en cours
Reconnaissance de Package	Marc-Aurèle CHARPENTIER-PODREZ	Intégration en cours
Routeur	Etienne LEPERCQ, Hai NGUYEN	Complété
WaferIC View	Keven CHAUSSE	Complété
WAPI	Fréderic BROCKOW, Long MA	Intégration en cours
WBHR Core	Jean Sébastien TURGEON, Etienne LEPERCQ, Yegor KHOMUTOV	Complété
WBHR Tools	Fréderic BROCKOW, Long MA	Complété

Le Tableau 2.2 présente l'état des modules développés ou intégrés sous ma direction technique et les personnes ayant effectuées leur développement. À ces travaux s'ajoute le développement d'un simulateur utilisé (décrit dans le prochain chapitre) pour valider le fonctionnement du logiciel.

Bien qu'encore incomplet, WaferConnect a été utilisé plusieurs fois avec succès en démonstration. En effet, plusieurs modules sont à présent intégrés. Le cœur du logiciel (gestion du flot de travail, structure de données et couche de communication) est fonctionnel de même que les modules GUI, visualisation du WaferIC, connexion CAD Tools, routage et détection de plots. La couche de communication est en cours de qualification en vue de son utilisation pour tester la prochaine version du WaferIC physique.

CHAPITRE 3 INTERFACE AVEC LE MATERIEL

Durant la phase de développement du logiciel WaferConnect, un des défis majeurs a été d'assurer la continuité du travail entre les différentes équipes de stagiaires notamment. Ce défi reste encore très présent.

Comme nous l'avons déjà présenté dans le Chapitre 2, les contraintes du projet sont nombreuses, tant au niveau matériel que logiciel. On remarque aussi que la majorité des intervenants sont spécialisés soit dans le matériel soit dans le logiciel, mais sont difficilement capables d'exceller dans les deux domaines. Or une grande partie du logiciel est très proche du matériel et demande donc des compétences dans ce domaine (notamment pour comprendre la manière dont il fonctionne). Cette constatation nous a amené assez rapidement à concevoir le logiciel de manière à découpler au maximum ces deux parties.

Pour faciliter le travail des informaticiens et la maintenabilité du logiciel, les parties communication et structure de données ont été isolées, et nous avons décidé de lier les modules de communication au module de pilotage de l'interface physique de communication (USB). Cette conception permet l'introduction d'un simulateur. Il devient alors possible de valider le fonctionnement du logiciel sans la présence du matériel.

Ce chapitre présente la démarche adopté pour concevoir, étendre et délimiter l'interface avec la matériel. Cela correspond à la couche « donnée » présenté section 2.3.

3.1 Proposition d'architecture d'interface

Habituellement, la manière d'arriver à une séparation des domaines matériel et logiciel consiste de concevoir un pilote proposant un accès simple au matériel. Cela fonctionne bien pour accéder à un périphérique USB, un micro, une webcam, un clavier, … qui finalement sont des périphériques relativement simples. L'API peut se limiter à quelques modes de configuration et des méthodes permettant d'accéder aux flots de données. Configurer un système comme le WaferIC est plus complexe de par le nombre d'éléments configurables

et la diversité des protocoles de communication. Il est donc nécessaire d'étendre les fonctionnalités de ce module logiciel de pilotage.

Dans le cas du WaferConnect, les contraintes de base sont lourdes. Le logiciel doit pouvoir être rapidement adapté a une nouvelle version du circuit. Il est aussi nécessaire de pouvoir supporter l'évolution de l'interface avec le matériel tout en assurant la compatibilité avec les précédentes versions, ce qui se traduit par la capacité de prendre en charge plusieurs interfaces (via USB, LAN, …). Enfin les contraintes de performance peuvent devenir critiques. L'application doit fonctionner sur un ordinateur de bureau, ce qui limite les ressources disponibles. Sachant que certains modules qui doivent faire partie du logiciel final consomment 90% des ressources mémoires et processeur disponibles durant leur exécution, nous avons décidé d'adapter les modèles de conception existants à nos besoins et non l'inverse.

La solution initiale vise à utiliser un pilote USB libre « libusb » (http://www.libusb.org). Ce choix est fait en raison de la compatibilité de la librairie « libusb » avec Linux et Windows et sa licence LGPL. Les pilotes prenant en charge plusieurs systèmes d'exploitation sont peu communs car leur écriture est fortement liée à un système d'exploitation donné [15][16]. Le développement effectué se greffe sur ces fondations. Le travail délicat du design de pilote et de ses interactions avec le système d'exploitation est ainsi laissé à la librairie.

Au niveau de son interface de développement, un pilote offre généralement les fonctionnalités suivantes : interrogation de l'état, lecture, écriture, commandes liées au protocole utilisé. Le but de l'implémentation est de regrouper les éléments nécessaires pour arriver à une structure d'accès similaire pour différentes interfaces physiques, tels l'USB ou l'Ethernet, maximisant ainsi la réutilisation de modules.

Bien que l'USB soit utilisé pour les développements initiaux et les exemples, notre problématique finale n'est pas de concevoir une architecture autour d'un pilote en particulier, mais de pouvoir prendre en charge différent pilotes en fonction de l'évolution de la spécification du WaferConnect et du WaferBoard. Pour y répondre, nous proposons d'utiliser une interface de programmation asynchrone. De cette manière, l'interface

supporte le traitement parallèle, les priorités d'envoi, et la gestion d'erreurs de communication. Cela offre une grande diversité de méthodes d'accès au matériel.

Comme présenté dans la section 2.3.2 la configuration du WaferIC repose sur le protocole JTAG et la construction d'un chemin permettant la configuration fonctionnelle du système. Ce mécanisme est unique, il est donc intéressant de l'intégrer au niveau du pilote pour limiter les risques d'implémentation redondante. Cette approche simplifie la tâche du développeur travaillant sur les fonctionnalités car il peut demander directement des modifications de registre au pilote. Le caractère asynchrone du pilote permet aussi d'envisager le regroupement des modifications de registres. On arrive alors à un système de transactions, permettant d'envoyer des demandes de modifications de plusieurs registres à la fois au pilote. Celui-ci se charge de déterminer les meilleurs chemins de configuration pour y arriver, et utilise le parallélisme offert par les 21 PowerBlocks. Le fait de travailler de cette manière peut générer une latence mais le nombre de chemins de configuration nécessaire pour arriver à l'état souhaité est moindre, ce qui le plus souvent se traduit par une diminution du volume de données de configuration et donc une amélioration des performances. Ce pilote « évolué » prenant en charge les transactions est présenté à la Figure 3-1.

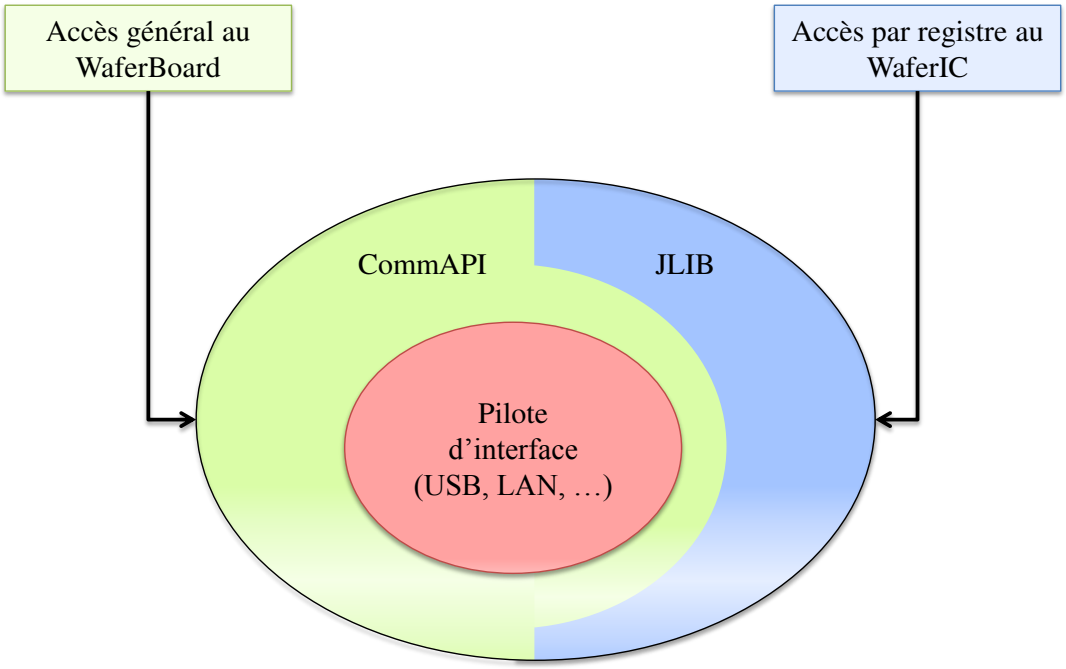


Figure 3-1: Encapsulation des mécanismes génériques de pilotage.

Une telle architecture permet un accès individuel au matériel: chaque module fonctionnel effectue ses transactions, mais il est le seul à connaitre la configuration qu'il a demandé. Si le module fonctionnel ne met pas correctement à jour le WBHR, la structure de donnée est corrompue et le système peut ne plus fonctionner. Une solution consiste à mettre à jour la totalité des registres, mais cela est très coûteux en temps. Une autre solution est d'étendre la portée du pilote en y englobant le WBHR et en le mettant à jour à chaque transaction. Cela permet d'assurer la cohérence des données.

Le matériel étant amené à évoluer, les registres à configurer pour obtenir un fonctionnement donné peuvent devenir plus nombreux d'une version à l'autre, cela pour supporter de nouvelles fonctionnalités dans le WaferIC. D'autre part, la structure de données WBHR a été conçue par des ingénieurs matériels. La structure adoptée est simple et logique de par sa hiérarchie, basée sur les éléments configurables du WaferIC à savoir les registres. Or pour les développeurs logiciels, travailler directement avec ces registre peut être complexe, notamment de par leur nombre. Par exemple, dans le cas du module de détection de plots (dont l'objectif est de détecter les courts-circuits entre les NanoPads), le développeur logiciel pourra profiter de la possibilité de demander le passage d'un NanoPad à un état logique 0 ou 1 en faisant abstraction des détails de configuration matériel [9].

Ces considérations introduisent le besoin de centraliser les mécanismes de configuration. Cela se fait au niveau du module WAPI. On obtient ainsi un ensemble permettant un accès au matériel simple et sécuritaire. Cet ensemble est schématisé à la Figure 3-2.

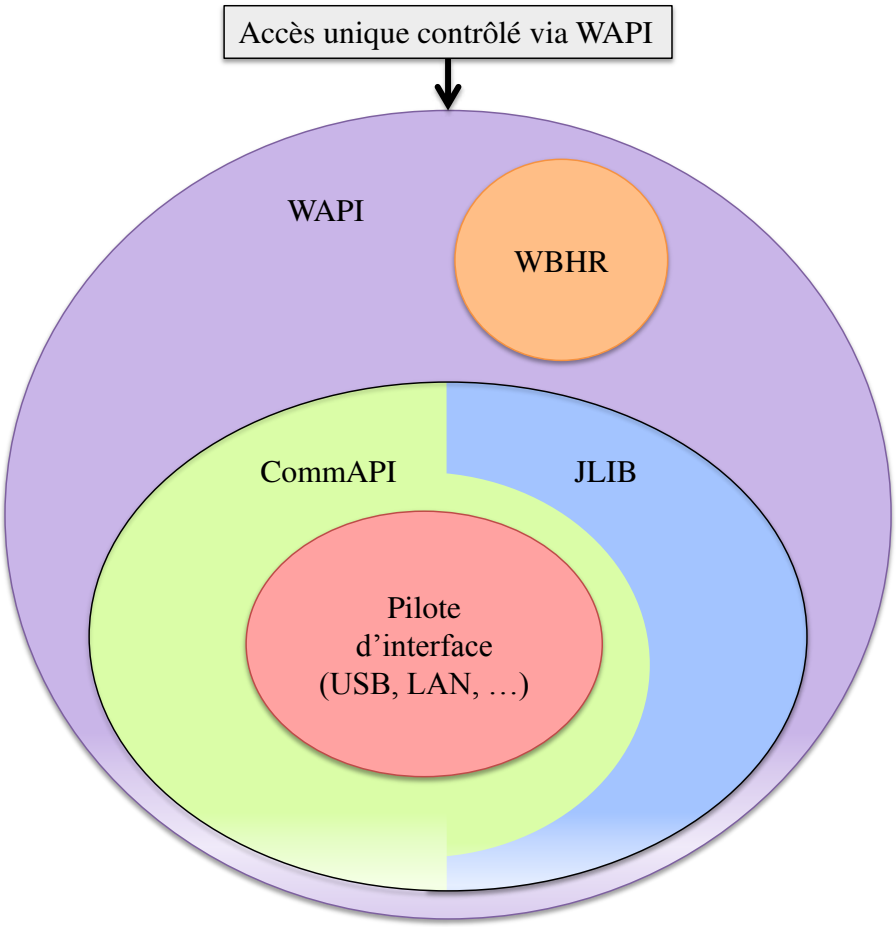


Figure 3-2: Intégration de la structure de données.

Bien que le but de cette approche soit l'isolement des notions de matériel et de logiciel, il en résulte des avantages sur deux plans différents. D'une part il devient possible de simuler le comportement du WaferBoard. En effet, nos couches fonctionnelles étant à présent bien définies, il est possible d'interfacer un module de simulation avec le logiciel au niveau de WAPI (simulateur V1), de *Comm API* (simulateur V2) ou du pilote (simulateur V3). Cela en remplaçant le module à simuler par le module de simulation correspondant. Ces deux cas sont présentés à la Figure 3-3. Pour pouvoir interagir avec une application à part entière, nous avons utilisé la technologie CORBA qui permet de transférer des objets entre deux applications via un réseau [28]. Le programme de simulation peut ainsi accéder aux API des modules souhaités.

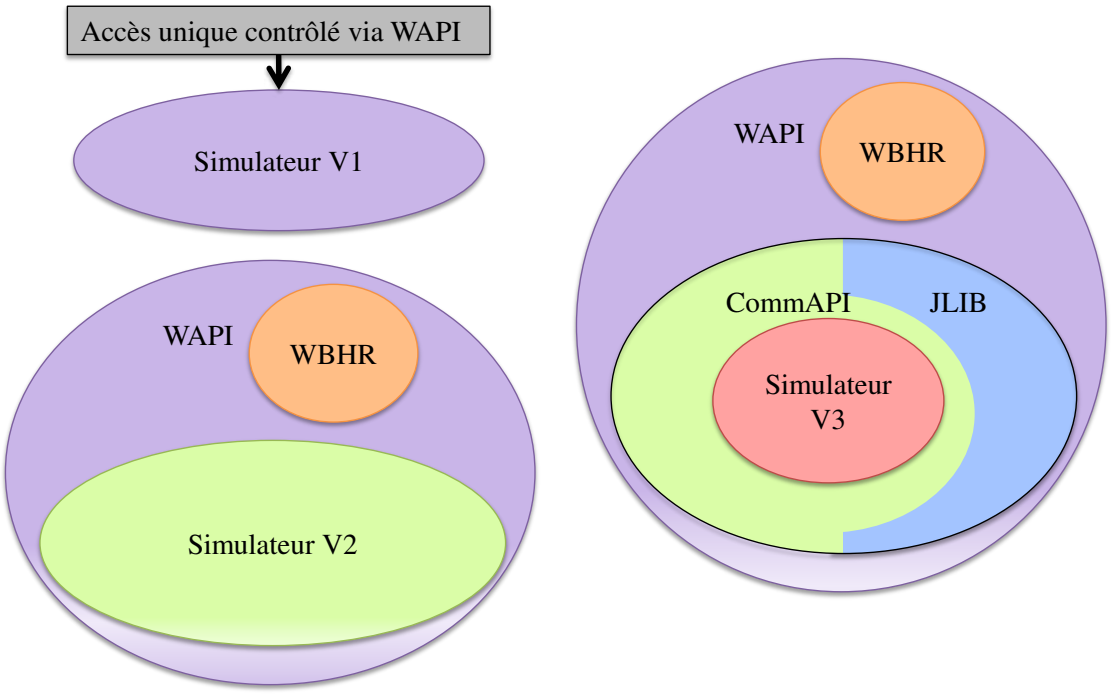


Figure 3-3: Situation des simulateurs dans l'architecture.

Cette architecture permet aussi d'apporter des options de fiabilité supplémentaires vis-à-vis de la plateforme matérielle bien qu'aucun code ne soit exécuté en mode noyau : le fait de travailler avec des commandes abstraites offre une visibilité à ce niveau d'abstraction. Une logique peut alors être appliquée pour filtrer les commandes pouvant mettre en danger la plateforme (à la manière d'assertions). Il devient aussi possible d'établir simplement des statistiques d'utilisation qui ne sont pas requises pour le moment mais pourront devenir intéressantes lors de l'optimisation et des premières commercialisations. Cela est lié au fait que tout module fonctionnel doit passer par WAPI pour interagir simplement avec le matériel. Le fait de pouvoir identifier les fonctionnalités les plus utilisées est un atout pour construire une stratégie d'optimisation. Que ce soit pour améliorer les performances sur le modèle courant en laboratoire, ou pour développer une nouvelle plateforme, la possession des données statistiques d'utilisation est un paramètre clef.

3.2 Résultats

Nous avons développé et testé les deux premières versions du simulateur (V1 et V2) qui interviennent à deux niveaux : tout d'abord dans la validation des modules fonctionnels puis dans la validation des modules WAPI et JLIB.

Une version du simulateur se substituant au module *Bitstream Transfer* (présenté ci-dessus comme simulateur V3) a permis de tester la couche de communication au complet. À ce stade il devient nécessaire de simuler le routage des commandes entre les éléments du WaferBoard (BottomPCB, PowerBlocks et WaferIC). Cette partie n'est pas encore opérationnelle.

Une fois le simulateur vérifié, à l'aide de la spécification exécutable du matériel, nous l'avons utilisé avec succès pour tester notre module de détection de plots. De plus, ce simulateur a permis de valider partiellement le fonctionnement des modules : WAPI, WBHR et JLIB.

CHAPITRE 4 VISUALISATION DE L'ÉTAT DU CIRCUIT

Dans ce chapitre nous présentons l'approche employée pour afficher l'état complet du circuit de la tranche. Le problème porte ici sur la représentation visuelle. En effet un tel affichage est nécessaire pour plusieurs raisons dont voici les deux principales :

D'une part, l'affichage de l'état précis du circuit offre un retour d'informations essentiel pour effectuer le suivi et la détection de bugs lors des phases de développement matériel. Il est difficile de monitorer une liste de plusieurs millions de registres de configuration en temps réel. Un affichage mettant en évidence les zones anormales (i.e. qui n'ont pas le comportement attendu, que ce soit lors d'un diagnostic, d'une détection de plots ou d'un routage) et permettant de sélectionner les éléments concernés offre un suivi rapide et précis des problèmes.

D'autre part, la présence de fragments de circuits non fonctionnels fait partie des problèmes à traiter lors d'une utilisation normale, de même que la validation de la position des composants du circuit à prototyper, leur identification et éventuellement la validation des routes. Il peut par exemple arriver qu'un composant soit déposé sur un groupe de NanoPads non fonctionnels. Dans ce cas l'utilisateur doit en être efficacement informé car il lui suffit de déplacer le composant pour résoudre son problème. L'idéal est de voir la totalité de l'état de la tranche de silicium de manière à repérer toutes les zones problématiques et connaître ainsi les positions valides pour accueillir les composants.

Plusieurs tentatives ont été faites pour assurer une telle représentation. Cela nécessite la prise en charge d'un grand nombre de données à afficher. De tels affichages représentent souvent un défi pour les développeurs logiciels. Les premières tentatives d'affichage ont confirmé cette difficulté en proposant un affichage précis et contrôlable mais extrêmement lent : plusieurs secondes étaient nécessaires pour générer une image de la tranche, l'image étant régénérée à chaque action effectuée par l'utilisateur; le résultat n'était pas satisfaisant.

Après l'introduction des spécifications du système d'affichage, ce chapitre présente une nouvelle approche née du compromis entre la rapidité d'affichage et la complexité d'implémentation des techniques utilisées.

La représentation attendue est celle du WaferIC et des éléments physiques qui le composent, tels l'état des NanoPads, des interconnexions du WaferNet, etc. Il est à noter que lorsque l'on parle d'affichage, il est question de la construction de l'image finale. En effet, les systèmes informatiques actuels se basent quasiment tous sur l'utilisation d'un tampon mémoire de la résolution de l'affichage nommé Frame buffer [22]. Ce Frame buffer est utilisé pour mémoriser une représentation matricielle de l'image finale à afficher.

Il est donc important de préciser que les images matricielles évoquées par la suite sont des images intermédiaires utilisées pour construire le Frame buffer. Dans ce contexte lorsqu'on parle d'affichage, il s'agit de la création de l'image destinée à être réellement affichée (à savoir le Frame buffer), et non de l'opération matérielle consistant à utiliser l'information du Frame buffer pour mettre à jour le périphérique d'affichage.

4.1 Spécifications

Pour rappel, la surface du WaferIC est composée de 74 réticules, contenant chacun 1024 cellules recouvertes de 16 NanoPads. Chaque cellule est reliée à des cellules voisines à l'aide 24 liens du WaferNet et 8 liens du réseau de configuration JTAG. Ce qui revient à 1 288 266 éléments et 1 212 416 liens, soit un total de 2 500 682 objets potentiellement affichables. L'affichage des réticules et des cellules permet à l'utilisateur de se repérer par rapport à la plateforme.

L'affichage des NanoPads est nécessaire en cas de défectuosités, ou de court-circuits avec un plot du composant déposé par l'utilisateur. Cela permet à l'utilisateur de détecter les problèmes de positionnement de ses composants.

L'utilisateur peut également être amené à effectuer une vérification visuelle des routes en cas de comportement non souhaité (par exemple dans le cas d'une route ne respectant pas un délai). Ces possibilités de visualisation sont également précieuses pour l'équipe de développement et l'équipe chargée de la qualification des prototypes du WaferIC.

L'affichage fonctionnel doit donc théoriquement être capable de gérer un maximum de 2,5 millions d'objets. Ce problème d'affichage de nombreux objets est rencontré dans d'autres domaines de la microélectronique, notamment la conception de circuits intégrés à très

grande échelle (VLSI) [29]. Il est à noter que la quantité réelle d'objets est inférieure car en cas d'utilisation normale, un grand nombre de NanoPads et de liens ne présentent ni défectuosité, ni activité. Les nombres obtenus représentent donc le pire scénario.

La première tentative de création d'une représentation visuelle du jeu de données matérielles s'est faite selon une approche classique. Cela consistait à utiliser les techniques de rendu logiciel standard disponibles à travers l'API de Qt [12]. Dans ces conditions, les éléments sont modélisés comme des objets à afficher. Ils sont référencés par un système de gestion d'événements [30] et l'opération de *clipping* (réduction de la scène à calculer à la partie visible uniquement) [31] est effectuée automatiquement par le kit de composants logiciels Qt. Ces mécanismes permettent un affichage relativement efficace en présence d'un nombre limité d'objets. L'affichage est précis: pas de pixellisation, un anticrénelage (anti-aliasing) est appliqué automatiquement, et le contrôle des objets tels que la sélection et la modification de ceux-ci est géré par le kit de composants logiciels, donc simple à implémenter et à maintenir. Cependant, suite aux expérimentations que nous avons effectuées, il s'avère qu'au-delà de 100 000 objets, le temps de calcul de l'image dépasse l'ordre de la seconde et le processeur est fortement utilisé. Cette solution n'est donc clairement pas viable si le nombre d'objets à afficher est de l'ordre de plusieurs millions. L'affichage doit consommer un minimum de ressources CPU et RAM de telle sorte que les parties de l'application critiques et gourmandes en ressources CPU comme le routage ou la détection de plots, puissent travailler efficacement.

4.2 Proposition d'une solution performante d'affichage

Il existe plusieurs techniques pour réaliser une accélération de l'affichage par rapport à la première implémentation proposée (présentée ci-dessus). Les bases sont présentées dans le chapitre 1. La combinaison de ces techniques est couramment utilisée pour obtenir les performances souhaitées. Cela aboutit généralement à l'élaboration d'une solution spécifique présentant un compromis entre la consommation de ressources, la vitesse, la complexité d'implémentation et par extension le coût de la solution mise au point. Hormis les applications proposant un affichage en trois dimensions, l'utilisation du matériel d'accélération 3D pour effectuer un affichage vectoriel simple est anecdotique. Une

solution pour obtenir un affichage rapide, proposée par J. Solomon [32] est d'utiliser une hiérarchie de données pour définir une frontière entre les informations traitées sous forme d'images matricielles et les informations affichées directement avec OpenGL.

La technique est bien adaptée pour résoudre le problème de la vitesse d'affichage, son implémentation est toutefois délicate à mettre en œuvre. Un intérêt majeur de cette technique est qu'elle tire parti d'un grand nombre de techniques, y compris l'accélération matérielle pour proposer un affichage des plus rapide. L'architecture de WaferIC est hiérarchique, donc cette technique [32] pourrait être appliquée.

Les performances obtenues avec la technologie disponible en 2002 [29] basée sur l'utilisation d'OpenGL et du mipmapping pourraient être suffisantes pour répondre à nos exigences de vitesse. Cependant, l'effort nécessaire à la mise en œuvre de cette technique est considérable. De plus, il n'y a rien de prévu concernant la prise en charge des interactions de l'utilisateur avec la scène. En effet, l'utilisateur doit avoir la possibilité de sélectionner les objets présents dans la scène sans passer par une liste annexe, cela pour garantir une bonne interactivité. Avec l'évolution de la technologie OpenGL et des GPU, afficher une telle quantité (de l'ordre de 2,5 millions) de données vectorielles en temps réel de manière simple devient possible.

En partant du principe qu'une utilisation efficace et intensive des capacités matérielles disponibles peut permettre d'obtenir des performances suffisantes, nous sommes amenés à étudier les technologies de prise en charge des données vectorielles d'OpenGL.

L'utilisation des « Vertex Buffer Object » (VBO) est une technique apparue avec la version 1.5 d'OpenGL disponible depuis Juillet 2003. Toutefois, son utilisation dans le contexte de notre application laisse entrevoir plusieurs obstacles pouvant limiter l'intérêt d'une telle utilisation. D'une part, il est beaucoup plus complexe d'utiliser les VBO par rapport à l'utilisation d'une grosse image matricielle. Le principe de base des VBO est d'allouer une partie de la mémoire graphique pour le stockage des sommets qui sont utilisés pour le tracé des figures (le type de figures à tracer est aussi communiqué). Cette zone mémoire est alors rendue accessible au programme via un pointeur. Dans ce contexte, le programmeur doit assurer la cohérence de ses données. La disponibilité des cartes vidéo 3D dédiées supportant la technologie VBO dans les ordinateurs de bureau peut être vue comme un

autre obstacle mais en comparaison avec le coût de la plateforme de prototypage, l'investissement qu'une telle carte représente est négligeable. De plus, des nouvelles spécifications d'OpenGL, seul OpenGL ES ne prend pas en charge les VBO. Par contre, cette technique prend uniquement en charge les opérations d'affichage. Dans certains cas, la manipulation des données est nécessaire (par exemple la sélection).

Le problème de la complexité d'implémentation peut être résolu par l'utilisation d'un kit de composants logiciels. Cela a un coût en termes de performances (la méthode OpenGL utilisée par le kit de composants logiciels n'est pas contrôlable). Le problème de la disponibilité du matériel tend à disparaître avec les nouvelles générations de processeurs qui embarquent de plus en plus couramment un GPU. Dans le cas de données d'entrées inadaptées (formes géométriques composées), une transformation en lignes et en polygones doit être effectuée pour que le tracé de la scène puisse être fait. Compte tenu des données à représenter dans notre cas, cette contingence est évitée. Le dernier obstacle, non des moindres, consiste à élaborer un mécanisme de sélection adapté. En effet OpenGL permet de gérer l'affichage mais ne permet pas de prendre en charge les interactions utilisateur comme le fait Qt. Bien que dans le cas de WaferConnect, la quantité de données à afficher soit particulièrement importante, ces premières considérations nous poussent à étudier les solutions immédiates. Le kit de composants logiciels Qt possède un mode de rendu utilisant automatiquement OpenGL. Ce fut donc la première solution envisagée. Malheureusement, le simple affichage des NanoPads ne permet pas de rencontrer l'objectif d'un affichage en moins de 100 ms.

Tel que précédemment implémenté, WaferConnect utilise deux fonctionnalités clés offertes par Qt : l'affichage et la prise en charge des événements d'entrée (dans notre cas les événements de la souris) générés par l'utilisateur correspondant à la sélection et l'agrandissement ou le déplacement de la zone visualisée. Si l'utilisation des mécanismes du kit de composants logiciels Qt est abandonnée au profit d'une solution basée sur les VBO et donc OpenGL, on peut utiliser une « vue » OpenGL [33] que l'on peut déplacer à la manière d'une caméra vis-à-vis de la scène (à savoir la représentation du WaferIC). Dans ce cas, l'agrandissement et le déplacement de la zone visualisée sont simples à implémenter. Il reste à résoudre le problème de la sélection d'un objet dans la scène.

Une approche triviale du mécanisme de sélection consiste à vérifier l'inclusion des coordonnées du point de sélection dans la zone recouverte par chacun des objets. Même avec une complexité linéaire, le traitement des 2 500 682 objets de la scène implique un grand nombre de calculs. Un pixel est sélectionné lorsqu'il est désigné par le pointeur de la souris à l'écran et qu'un clic est détecté. Par contre, le système de coordonnées de la scène et donc des objets n'est pas le même que celui du pointeur de la souris. Pour déterminer l'objet ou les objets à sélectionner, une projection des coordonnées du pixel sélectionné doit être effectuée dans l'espace de la scène 3D d'OpenGL [33]. Cette projection produit une demi-droite le long de l'axe Z comme présenté Figure 4-1.

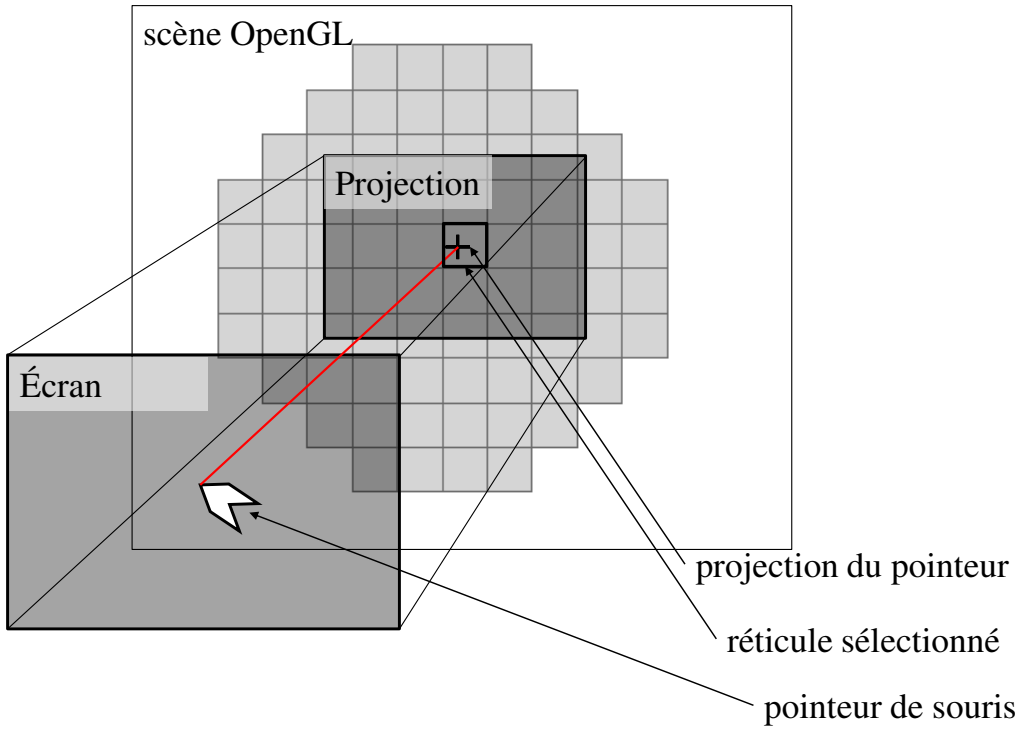


Figure 4-1: Mécanismes du système de sélection.

Une fois la projection connue, il est possible de calculer l'intersection entre cette droite et les objets présents dans le plan de la scène. Si une intersection est détectée, cela signifie que l'objet en question se trouve sous le pointeur de la souris et doit donc être sélectionné. Pour accélérer cette étape, certains objets peuvent être indexés en fonction des domaines qu'ils

recouvrent. Cela permet de segmenter l'espace et donc de réduire la quantité d'éléments faisant l'objet d'un calcul d'intersection. Dans ce cas, la sélection est plus rapide, mais la consommation de mémoire augmente avec la taille de l'index.

Dans notre application, le choix de l'utilisation d'un tel index ne se pose pas. En effet, les données sont naturellement indexées. Cela est dû à leur distribution régulière. Par exemple pour sélectionner un NanoPad, on détermine tout d'abord le réticule concerné (1 parmi 76). On descend ensuite dans la hiérarchie pour déterminer la cellule impliquée (1 parmi 1024), enfin le calcul d'intersection est effectué sur les 16 NanoPads de cette cellule. Cette approche limite les calculs d'intersection à 1 116 opérations plutôt que 1 245 184, ce qui correspond au nombre total de NanoPads.

Cette approche spécifique à notre cas permet une sélection simple et rapide. Nous avons donc décidé de composer l'utilisation de ce mécanisme de sélection avec l'utilisation des VBO et d'une vue OpenGL pour prendre en charge l'affichage du WaferIC dans WaferConnect.

Cette approche pourrait de plus présenter un intérêt pour afficher tous types de données vectorielles 2D structurées, où les seules interactions nécessaires sont: le déplacement de la vue, l'agrandissement et la sélection des éléments de la scène.

4.3 Résultats

L'utilisation d'une simple image matricielle simplifie grandement la mise en œuvre si on sait comment générer cette image à partir de la description du circuit. L'utilisation d'un matériel dédié à travers la technologie OpenGL et les VBO offre un rendu rapide et des images précises (même à fort grossissement). À cela s'ajoute une réduction de la charge du processeur principal et des besoins en mémoire.

Ces améliorations ont un prix: sans la présence d'un GPU (généralement une carte accélératrice graphique) prenant en charge les VBO, l'affichage ne peut être achevé. Cependant, il est toujours possible d'ajouter un tel matériel dans un ordinateur de bureau. En comparaison avec le coût de la plateforme de prototypage, l'investissement que cela représente est insignifiant.

Le Tableau 4.1 regroupe les résultats obtenus avec les différentes méthodes présentées précédemment. Ces résultats ont été obtenus avec une carte graphique ATI Radeon 5570. Ils sont basés sur la manipulation de 2 502 724 objets rectangulaires indexés sous forme de Quad, sur un ordinateur équipé d'un processeur Intel Core I5-750 et de 4Go de RAM, dans un environnement composé de Windows 7 64bits, Java 1.7 64bits (en utilisant la librairie JOGL 2.0) et du pilote graphique ATI Catalyst 12.8. Le logiciel de test a été écrit en Java avec les options « noddraw = true » pour le rendu vectoriel et matriciel non accéléré, et « opengl = true » pour l'accélération matérielle automatique (supporté par le kit de composants logiciels AWT de Java). Java est le langage utilisé pour l'étude expérimentale. Les résultats présentés dans le tableau 4.1 sont des valeurs moyennes obtenues sur la base de 100 expérimentations, cela pour minimiser l'influence du système d'exploitation et de la machine virtuelle Java. L'affichage est effectué de manière a ce que le WaferIC soit entièrement visible à l'écran.

Tableau 4.1: Comparaison des temps de calcul des techniques présentées.

Techniques d'affichage	**Résultats**		
	Initialisation (ms)	**Affichage (ms)**	**Accélération**
Image vectorielle	1000	6365	1
Image matricielle	7240	10	636
Accélération matérielle automatisée.	1012	890	7
Accélération matérielle spécifique (avec VBO)	268	25	254

Comme prévu, l'affichage basé sur une image matricielle est très rapide mais ne nécessite pas moins d'1 Go de RAM pour mémoriser l'image de base. Dans ce cas de figure, le déplacement de la vue est tout aussi rapide mais le redimensionnement de l'image peut prendre du temps (en restant inférieur à la seconde). Ce défaut est résolu en utilisant plusieurs images recalculées de tailles différentes, ce qui augmente l'empreinte mémoire.

La plupart des solutions disponibles à travers les kits de composants logiciels actuels offrent une grande flexibilité mais les vitesses d'affichage sont relativement faibles. L'accélération matérielle automatisée proposée par le kit de composants logiciels Java AWT est insuffisante pour répondre au problème initial (affichage en moins de 100 ms).

La solution proposée ici tire parti du matériel moderne via les VBO tout en conservant une implémentation abordable. C'est un compromis entre la solution proposée par J. Solomon et celle offerte par l'intermédiaire de Qt ou du kit de composants logiciels Java AWT.

Nos objectifs en matière de temps d'affichage sont remplis, et la méthode proposée permet de conserver une implémentation simple. De plus, la consommation de mémoire est déplacée vers le matériel. En revanche, la vitesse d'affichage, avec la méthode proposée n'est pas la plus rapide. Enfin, notre solution n'est pas intégrée dans un kit de composants logiciels existant, les mécanismes d'interaction doivent donc être réécrits.

Cette solution pourrait être utilisée dans d'autres domaines. La solution proposée est par exemple applicable dans le cadre des SIG [34] en envoyant des données vectorielles seules et en déléguant le travail de rendu à l'application cliente. Cela permettrait de réduire les quantités d'informations transférées entre le serveur et les clients et donc le temps de transfert des données, tout en conservant un affichage très rapide.

CONCLUSION

Dans ce mémoire, les différentes étapes de la conception du logiciel de contrôle pour le système de prototypage DreamWafer ont été présentées. Cet outil nommé WaferConnect permet à ses utilisateurs d'accéder au Wafer IC pour configurer leurs circuits, cela à l'aide d'une interface graphique complète et adaptable. WaferConnect doit supporter le cycle de travail suivant : une fois le démarrage et le diagnostic effectués, l'utilisateur dépose ses composants à la surface de la plateforme, demande une détection de plots, et importe les données de son circuit (netlists et footprint). Après avoir validé l'identification et l'orientation de ses composants que la fonction de détection de package lui propose, il est informé des erreurs éventuelles et peut demander un routage. Le circuit ainsi généré peut alors être vérifié et implanté sur le WaferIC. Le prototype est alors prêt à fonctionner et peut être débogué et modifié à souhait.

J'ai rédigé la spécification du logiciel WaferConnect en collaboration avec Etienne Lepercq. Le Tableau 2.2 (p. 54) liste les modules développés ou intégrés sous ma direction. La gestion du projet logiciel a été assurée par Pierre Popovic. Ces travaux m'ont permis de rédiger un chapitre de livre intitulé « A Wafer-Scale Rapid Electronic Systems Prototyping Platform – User Support Tools and Thermo-Mechanical Validation » en collaboration avec Hai Nguyen, Mohammed Bougataya, Yves Blaquière, Ahmed Lakhssassi, Mary Shields et Yvon Savaria.

Actuellement certaines fonctions du logiciel doivent encore être développées. Ces deux années de travaux ont permis d'obtenir un logiciel stable et évolutif assurant les fonctions suivantes :

- La détection de la plateforme et la communication complète avec celle-ci (que ce soit avec le BottomPCB, les PowerBlocks ou le WaferIC).

- Le diagnostic du réseau de configuration JTAG, fonction pré-requise pour effectuer une quelconque configuration du WaferIC que l'on sait fonctionnelle.

- La détection des plots des composants de l'utilisateur.

- L'import des données du circuit de l'utilisateur.
- L'identification automatique des empreintes des composants (l'interface permettant la correction et l'identification manuelle est en cours de développement).
- Le routage, partiellement fonctionnel car son intégration n'est pas encore finalisée.
- La configuration du WaferIC à partir de son modèle théorique.
- Le filtrage des commandes utilisateur en fonction de l'état de la plateforme.

Pour être en mesure de contrôler partiellement la plateforme de prototypage, le module assurant la séquence de démarrage ou « bootup » doit être finalisé. Le matériel nécessaire à la séquence de démarrage est actuellement en cours de conception. Les spécifications résultant de cette conception permettront de développer cette fonctionnalité.

Pour obtenir un système pleinement fonctionnel, l'identification manuelle des composants et le diagnostic du WaferNet devront être ajoutés. Ces deux fonctions doivent être fonctionnelles pour pouvoir effectuer le routage. De plus l'intégration du routage pourra être améliorée. D'autres améliorations peuvent certainement être apportées comme la possibilité d'exécuter plusieurs commandes indépendantes en parallèle, ou la sauvegarde des résultats du diagnostic.

Outre les fonctions présentées, la GUI actuelle représente une autre de mes contributions au logiciel. En effet, la technique utilisé (Chapitre 4) est publiée à la conférence CCECE 2013 (*Canadian Conference on Electrical and Computer Engineering*). Nous avons réussi à produire une interface intuitive et personnalisable adaptée à un large public. Le repérage des erreurs sur le circuit est simple et rapide, et l'observabilité de la plateforme matérielle est très détaillée, tout en conservant une réactivité suffisante pour garantir une utilisation fluide. Le module d'affichage est adaptable en cas d'évolution du matériel.

La technique utilisée pour visualiser le WaferIC peut être utilisé dans bien des domaines où l'affichage d'un grand nombre de données vectoriel peut poser problème. Cela peut être le cas d'un document SVG ou PDF contenant un grand nombre d'objets (Adobe Reader peut mettre plusieurs secondes à afficher une image vectorielle contenant 100 000 vecteurs), ou

simplement pour l'affichage de « layout » de circuits intégrés. A partir du moment où une représentation 3D est utilisée (infographie, mécanique, …) la solution proposée ne sera plus adaptée.

L'architecture que nous avons conçue pour le logiciel facilite l'évolution et la maintenabilité. À plus long terme, dans l'optique d'une commercialisation, certains aspects seront à revoir. WaferConnect est destiné avant tout à une utilisation en recherche. Cependant l'architecture étant évolutive, ce logiciel peut très bien servir de base au développement d'une application commerciale.

Le projet Dreamwafer représente maintenant 5 ans de travaux de recherche et de développement, répartis aussi bien sur les aspects logiciels que matériels. Mais nous nous approchons du succès : Le premier circuit WaferIC complet est en cours de fabrication et une fois l'électronique nécessaire à la séquence de démarrage prête, WaferConnect permettra d'effectuer les premiers tests du système au complet !

RÉFÉRENCES

[1] R. Norman, O. Valorge, Y. Blaquiere, E. Lepercq, Y. Basile-Bellavance, Y. El-Alaoui, R. Prytula, et Y. Savaria, « An active reconfigurable circuit board », in *Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference*, Montreal, QC, Canada, 2008, p. 351‑354.

[2] « DreamWafer ». [En ligne]. Disponible: http://www.dreamwafer.com/. [Consulté le 02-févr-2013].

[3] « JTAG - A Technical Overview - TAP Signals and Instructions ». [En ligne]. Disponible: http://www.xjtag.com/support-jtag/jtag-technical-guide.php. [Consulté le 02-févr-2013].

[4] Y. Basile-Bellavance, Y. Blaquiere, et Y. Savaria, « Faults diagnosis methodology for the WaferNet interconnection network », in *Circuits and Systems and TAISA Conference*, 2009, p. 1 ‑4.

[5] E. Lepercq, O. Valorge, Y. Basile-Bellavance, N. Laflamme-Mayer, Y. Blaquiere, et Y. Savaria, « An interconnection network for a novel reconfigurable circuit board », in *Microsystems and Nanoelectronics Research Conference*, 2009, p. 53 ‑56.

[6] E. Lepercq, Y. Blaquiere, R. Norman, et Y. Savaria, « Workflow for an electronic configurable prototyping system », in *IEEE International Symposium on Circuits and Systems*, 2009, p. 2005‑ 2008.

[7] R. Clochard, « WaferBoardTM Protocols Preliminary Specifications ». Document interne.

[8] Y. Basile-Bellavance, « Conception d'un système de test et de configuration numérique tolérant aux pannes pour la technologie WAFERIC », mémoire de maîtrise, École Polytechnique de Montréal, 2009.

[9] M. Gémieux, S. Tasmili, « DreamWafer Project: THE TEST CHAINS OF TDI ». Document interne.

[10] « The EDA Technology Leader - Mentor Graphics ». [En ligne]. Disponible: http://www.mentor.com/. [Consulté le 02-févr-2013].

[11] H. H. Nguyen, M. Guillemot, Y. Savaria, et Y. Blaquiere, « A new approach for pin detection for an electronic system prototyping reconfigurable platform », in *2012 23rd IEEE International Symposium on Rapid System Prototyping (RSP)*, 2012, p. 122‑127.

[12] J. Blanchette, M. Summerfield, *C++ GUI Programming with Qt 4 (2nd Edition)*, Prentice Hall, 2008.

[13] « Standard Template Library Programmer's Guide ». [En ligne]. Disponible: http://www.sgi.com/tech/stl/. [Consulté le 24-avr-2012].

[14] « Boost C++ Libraries ». [En ligne]. Disponible: http://www.boost.org/. [Consulté le 24-avr-2012].

[15] C. Cant, *Writing Windows WDM Device Drivers*. R&D Books, 1999, pp.11-29.

[16] J. Corbet, A. Rubini, et G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly Media, 2005, pp.1-41.

[17] « USB.org - HID Tools ». [En ligne]. Disponible: http://www.usb.org/developers/hidpage/. [Consulté le 02-févr-2013].

[18] J. Printz, *Architecture logicielle : Concevoir des applications simples, sûres et adaptables*. Dunod, 2006, pp.221-271.

[19] A. Berson, *CLIENT/SERVER ARCHITECTURE. 2nd edition*. McGraw-Hill, 1996, pp.35-52.

[20] W. Eckerson, « Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications », presented at the Open Information Systems 10, 1995.

[21] T. REENSKAUG, « The Model-View-Controller (MVC) Its Past and Present », *University of Oslo Draft*, 2003.

[22] R. Shoup, « SuperPaint: an early frame buffer graphics system », *IEEE Annals of the History of Computing*, vol. 23, n° 2, p. 32‑37, 2001.

[23] L. Williams, « Pyramidal parametrics », *SIGGRAPH Comput. Graph.*, vol. 17, n° 3, p. 1–11, juill. 1983.

[24] P. Dong, C. Yang, X. Rui, L. Zhang, et Q. Cheng, « An effective buffer generation method in GIS », in *Geoscience and Remote Sensing Symposium, IEEE International*, 2003, vol. 6, p. 3706‑3708.

[25] « Intel Core i5 Processor Specifications », *Intel*. [En ligne]. Disponible: http://www.intel.com/content/www/us/en/processors/core/core-i5-processor/specifications.html. [Consulté le 15-mars-2012].

[26] M. J. Kilgard, « Modern openGL usage: Using vertex buffer objects well », in *ACM SIGGRAPH ASIA 2008 Courses,* 2008.

[27] S. Charasse, « Projet DreamWafer: Bottom PCB Protocoles de communication v1.06 ». Document interne.

[28] « The OMG's CORBA Website ». [En ligne]. Disponible: http://www.corba.org/. [Consulté le 02-févr-2013].

[29] J. Solomon, « THE CHIPMAP™: VISUALIZING LARGE VLSI PHYSICAL DESIGN DATASETS », Citeseer, 2002.

[30] Nokia Corporation, « Qt 4.7: Graphics View Framework ». [En ligne]. Disponible: http://doc.qt.digia.com/4.7-snapshot/graphicsview.html. [24-avr-2012].

[31] J. F. Hughes, A. van Dam, M. McGuire, D. Sklar, J. D. Foley, S. K. Feiner, et K. Akeley, *Computer Graphics: Principles and Practice*, 3e éd. Addison Wesley, 2013 , pp.110-124.

[32] J. Solomon et M. Horowitz, « Using texture mapping with mipmapping to render a VLSI layout », in *Proceedings of the 38th annual Design Automation Conference*, New York, NY, USA, 2001, p. 500–505.

[33] F. S. H. Jr et S. M. Kelley, *Computer Graphics Using OpenGL*, 3e éd. Prentice Hall, 2006, pp.39-103.

[34] S. Yesilmurat et V. Isler, « Retrospective adaptive prefetching for interactive Web GIS applications », *GeoInformatica*, vol. 16, no 3, p. 435‑466, 2012.

ANNEXE 1 – Exemple d'utilisation des VBO en Java

Ce code source java a été utilisé pour une première évaluation des performances en utilisant les VBO. Il s'agit d'une classe exécutable qui a été choisie en raison de sa compacité. Une approche procédurale a été adoptée pour une meilleure lisibilité de l'annexe. Cette classe se nomme « VBOTest ».

```
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

import javax.media.opengl.GL;
import javax.media.opengl.GL2;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCapabilities;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.GLProfile;
import javax.media.opengl.awt.GLCanvas;
import javax.media.opengl.fixedfunc.GLMatrixFunc;
import javax.media.opengl.glu.GLU;

import com.jogamp.common.nio.Buffers;
import com.jogamp.opengl.util.FPSAnimator;
```

```
/**
 * cette classe represente une implémentation d'affichage utilisant les VBO.
Elle nous a
 * La librairie JOGL est necessaire pour compiler et executer cette source.
 *
 * @author   Mikael GUILLEMOT
 * @version  1.0
 *
 */
public class VBOTest implements GLEventListener, MouseListener,
MouseMotionListener, MouseWheelListener {

	private int[] list = new int[] { -1 };
	private int pointsParQuad = 4;
	private static int resolution = 1582; // 5 502 724 de rectangles
	private int nombreSommets = pointsParQuad * resolution * resolution;
	private static Frame frame;
	static float f = 0.0081f;
	static float xt = 0.0f;
	static float yt = 0.0f;
	static float xm = 0.0f;
	static float ym = 0.0f;
	static Point d0;
	static int logframe = 0;
	static long moy = 0;

	// redefinition (selection)
	boolean redefine = false;
	int xrd = 0;
	int yrd = 0;

	private static GLCanvas canvas;

	@Override
	public void reshape(GLAutoDrawable drawable, int x, int y, int w, int h)
	{
	}

	@Override
	public void dispose(GLAutoDrawable drawable) {
		GL2 gl = drawable.getGL().getGL2();
		gl.glDeleteBuffers(resolution * resolution, list, 0);

	}
```

```
public static void main(String[] args) {
	VBOTest vbov = new VBOTest();
	GLProfile glp = GLProfile.getDefault();
	GLCapabilities caps = new GLCapabilities(glp);

	// on active l'anti aliasing (multisampling)
	caps.setSampleBuffers(true);
	caps.setNumSamples(4);
	canvas = new GLCanvas(caps);

	frame = new Frame(resolution * resolution + " nanopads !");
	frame.setSize(800, 600);
	frame.add(canvas);
	frame.setVisible(true);
	frame.addMouseWheelListener(vbov);

	frame.addWindowListener(new WindowAdapter() {
		//
		public void windowClosing(WindowEvent e) {
			System.exit(0);
		}
	});

	canvas.addMouseListener(vbov);
	canvas.addMouseMotionListener(vbov);
	canvas.addGLEventListener(vbov);

	FPSAnimator animator = new FPSAnimator(canvas, 160);
	animator.add(canvas);
	animator.start();
	// on compte le nombre d'image affiché en 10s
	double t1 = System.nanoTime();
	try {
		Thread.sleep(10000);
	} catch (InterruptedException e1) {
		e1.printStackTrace();
	}
	double t2 = System.nanoTime();
	double f = animator.getTotalFPSFrames();
	System.out.println("fps = " + f / ((t2 - t1) / 1000000000));
}
```

```
	@Override
	public void init(GLAutoDrawable drawable) {
		GL2 gl = drawable.getGL().getGL2();
		// test du support des VBO
		if (!gl.isFunctionAvailable("glGenBuffers") ||
!gl.isFunctionAvailable("glBindBuffer")
				|| !gl.isFunctionAvailable("glBufferData") ||
!gl.isFunctionAvailable("glDeleteBuffers")) {
			System.err.println("Vertex buffer objects not supported.");
			System.exit(0);
		}
		gl.glClear(GL.GL_COLOR_BUFFER_BIT);
		gl.glHint(GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL.GL_NICEST);

		// on demande un VBO
		gl.glGenBuffers(1, list, 0);

		// on lie le buffer a une reference mémoire
		gl.glBindBuffer(GL.GL_ARRAY_BUFFER, list[0]);

		// on alloue le buffer nombreSommets * 3 * Buffers.SIZEOF_FLOAT * 2
		gl.glBufferData(GL.GL_ARRAY_BUFFER, nombreSommets * 3 *
Buffers.SIZEOF_FLOAT * 2, null, GL2.GL_DYNAMIC_DRAW);

		// on map le buffer a une table de float
		ByteBuffer bytebuffer = gl.glMapBuffer(GL.GL_ARRAY_BUFFER,
GL2.GL_WRITE_ONLY);
		FloatBuffer floatbuffer =
bytebuffer.order(ByteOrder.nativeOrder()).asFloatBuffer();

		// On a un buffer de float mappé, il reste plus qu'a remplire!
		genData3d(floatbuffer);
		// une fois les données pretes on peu relacher le buffer
		gl.glUnmapBuffer(GL.GL_ARRAY_BUFFER);

	}
```

```
@Override
public void display(GLAutoDrawable drawable) {
	GL2 gl = drawable.getGL().getGL2();
	int w = canvas.getWidth(), h = canvas.getHeight();

	// en cas de modifications des données(i.e. selection) on
	// les traites
	if (redefine)
		redefine(gl);

	// on prepare la vue avec une projection orthogonale en
	// fonction de la distance f de la scene.
	gl.glMatrixMode(GLMatrixFunc.GL_PROJECTION);
	gl.glLoadIdentity();
	new GLU().gluOrtho2D(-(f*w)/2.f,(f*w)/2.f,-(f*h)/2.f,(f*h)/2.f);
	gl.glMatrixMode(GLMatrixFunc.GL_MODELVIEW);
	gl.glLoadIdentity();
	gl.glViewport(0, 0, w, h);
	// on effectue la translation
	gl.glTranslatef(xt, yt, 0.0f);

	// on prepare les données a tracer
	gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
	gl.glBindBuffer(GL.GL_ARRAY_BUFFER, list[0]);
	gl.glEnableClientState(GL2.GL_VERTEX_ARRAY);
	gl.glEnableClientState(GL2.GL_COLOR_ARRAY);
	gl.glVertexPointer(3, GL.GL_FLOAT, 6 * Buffers.SIZEOF_FLOAT, 0);
	gl.glColorPointer(3, GL.GL_FLOAT, 6 * Buffers.SIZEOF_FLOAT,
			3 * Buffers.SIZEOF_FLOAT);
	gl.glPolygonMode(GL.GL_FRONT, GL2.GL_FILL);

	// on dessine
	gl.glDrawArrays(GL2.GL_QUADS, 0, nombreSommets);

	// on libere les ressources
	gl.glBindBuffer(GL.GL_ARRAY_BUFFER, 0);
	gl.glDisableClientState(GL2.GL_VERTEX_ARRAY);
	gl.glDisableClientState(GL2.GL_COLOR_ARRAY);
}
```

```
	/* modification de la couleur d'un polygone selectionné */
	private void redefine(GL2 gl) {
		redefine = false;
		// ici c'est le meme principe qu'a l'initialisation des données,
		// mais le buffer est deja défini.
		gl.glBindBuffer(GL.GL_ARRAY_BUFFER, list[0]);
		ByteBuffer bytebuffer = gl.glMapBuffer(GL.GL_ARRAY_BUFFER,
GL2.GL_WRITE_ONLY);
		FloatBuffer floatbuffer =
bytebuffer.order(ByteOrder.nativeOrder()).asFloatBuffer();
		int pos = (xrd + yrd * resolution) * 24; // 3*4 pts+3*4 color
		// on ne touche pas au coordonées
		floatbuffer.position(pos + 3);
		setColorBleue(floatbuffer);
		floatbuffer.position(pos + 3 + 6);
		setColorBleue(floatbuffer);
		floatbuffer.position(pos + 3 + 6 + 6);
		setColorBleue(floatbuffer);
		floatbuffer.position(pos + 3 + 6 + 6 + 6);
		setColorBleue(floatbuffer);
		gl.glUnmapBuffer(GL.GL_ARRAY_BUFFER);
		gl.glLoadIdentity();
	}

	private void setColorBleue(FloatBuffer floatbuffer) {
		floatbuffer.put(0);// rouge
		floatbuffer.put(0);// vert
		floatbuffer.put(1);// bleu
	}


	@Override
	public void mouseClicked(MouseEvent e) {
		// ici on addapte les coordonées de la souris au canvas
		Dimension d0 = canvas.getSize();
		float nb = resolution;
		Float x, y;
		float pitch = 2 / nb;
		x = (e.getX() - d0.width / 2) * f - xm;
		y = -(e.getY() - d0.height / 2) * f - ym;
		if (x > -1 && x < 1 && y > -1 && y < 1) {
			x = x / pitch;
			y = y / pitch;
			if (x > 0)
				x++;
			if (y > 0)
				y++;
			xrd = (x.intValue() + resolution / 2) - 1;
			yrd = (resolution / 2 + y.intValue()) - 1;
			redefine = true;
		}

	}
```

```
@Override
public void mouseWheelMoved(MouseWheelEvent e) {
	// gestion du recule de la camera
	if (e.getWheelRotation() < 0)
		f = f / 1.07f;
	else
		f = f * 1.07f;
}

@Override
public void mouseEntered(MouseEvent e) {
	// inutilisé
}

@Override
public void mouseExited(MouseEvent e) {
	// inutilisé
}

@Override
public void mousePressed(MouseEvent e) {
	// point de depart d'une éventuelle translation
	d0 = e.getPoint();
}

@Override
public void mouseReleased(MouseEvent e) {
	// destination d'une éventuelle translation
	xm = xt;
	ym = yt;

}

@Override
public void mouseDragged(MouseEvent e) {
	// deplacement en cours de translation
	xt = xm + (e.getX() - d0.x) * f;
	yt = ym + (-e.getY() + d0.y) * f;

}

@Override
public void mouseMoved(MouseEvent e) {
	// inutilisé
}
```

```
	private void genData3d(FloatBuffer fb) {

		float nb = resolution;
		float x = -1;
		float y = -1;
		float fullPitch = 2 / nb;
		float smalPitch = fullPitch / 20;
		float largePitch = smalPitch * 19;

		for (int i = 0; i < resolution; i++) {
			for (int j = 0; j < resolution; j++) {

				// vertex
				fb.put(x + smalPitch);
				fb.put(y + smalPitch);
				fb.put(0.0f);
				// color
				fb.put(1 - i / nb);
				fb.put(1 - j / nb);
				fb.put(1.0f);

				fb.put(x + smalPitch);
				fb.put(y + largePitch);
				fb.put(0.0f);
				fb.put(1 - i / nb);
				fb.put(1 - j / nb);
				fb.put(1.0f);

				fb.put(x + largePitch);
				fb.put(y + largePitch);
				fb.put(0.0f);
				fb.put(1 - i / nb);
				fb.put(1 - j / nb);
				fb.put(1.0f);

				fb.put(x + largePitch);
				fb.put(y + smalPitch);
				fb.put(0.0f);
				fb.put(1 - i / nb);
				fb.put(1 - j / nb);
				fb.put(1.0f);

				x = x + fullPitch;
			}
			y = y + fullPitch;
			x = -1;
		}
	}

}
```