CAD TOOL EMULATION FOR A TWO-LEVEL RECONFIGURABLE DSP ARCHITECTURE

by

DANIEL SKARPAS

a thesis submitted in partial fullfillment of

the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

School of Electrical Engeneering and Computer Science

WASHINGTON STATE UNIVERISTY

May 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of DANIEL SKARPAS find it satisfactory and recommend that it be accepted

_____

Chair:

_____

_____

Acknowledgements

This research was conducted with the High Performance Computer Systems (HiperCops) research group under the guidance of Dr. José G. Delgado-Frias at Washington State University. I would like to thank both Michell Myjak for developing the architecture used and Jonathan Karl Larson for he's previous work on the CAD-tool and their invaluable help through the development. I would also like to express my appreciation of the support and help provided by Dr. Dr. José G. Delgado-Frias.

CAD TOOL EMULATION FOR A TWO-LEVEL RECONFIGURABLE DSP ARCHITECTURE

Abstract

by Daniel Skarpas M.S.

Washington State University

May 2007

Chair: José G. Delgado-Frias

The use of reconfigurable hardware has become more attractive for usage in Digital Signal processing the last few years as well as the use of CAD tools for manipulation of specific DSP or as a mean for simulation. To be more specific, the CAD-tools enable the users to create, modify, simulate and provide valuable feedback about the respective hardware long before creation. This research deals with the design of CAD Tools for the creation and manipulation of medium-grained DSP algorithms by encapsulating minor components and merging them into larger modules by detailed feedback from the CAD0-tool throughout the programming and simulation. The architecture used is a result of several years' research by the High Performance Computer Systems (HiperCops) research group at Washington State University to accelerate digital signal processing.

Furthermore the CAD-tools includes features to create and design complicated DSP algorithms through the graphical interface, and creating generic modules by abstracting the functionality of designed sections and mapping them to other sections of the architecture. As a tool for evaluation and benchmarking, CAD tools can greatly enhance the usage of the final product and provide valuable feedback from the design of the hardware architecture through the graphical user interface.

Table of Contents

Dedication

This thesis is dedicated to my parents for their continuous support through the years and for always being there.

Chapter 1

Introduction

All around us we encounter Digital Signal Processing (DSP) in one form or another, often presented in minor hardware through cell phones, PDA's GPS etc, and the need for hardware research in this field continues as requirements and a competitive marked increase the usage of DSP's every day. Many now look toward reconfigurable hardware as an option for DSP, as they combine the flexibility of field-programmable gate arrays (FPGAs) and the real-time requirements provided by purpose-designed application-specific integrated circuits (ASICs).

The problem with FPGAs is apparent as we look at basic operations such as multiplications, and most FPGA provides dedicated multipliers to enhance the overall l performance of the system. Several new approached has been proposed for DSP Computation [1], such as the one-dimensional RaPiD array[2], the two-dimensional KressArray[3] and architectures that support both fine grained and medium-grained components by MONITUIM[4] and others. One of the problems with the alternatives listed above is that some are limited to by the length of the predefined word, and thus creating restrictions for the coarse-grained architecture they support. As an alternative to the options mentioned above, HiperCops Research group developed a Medium-grained architecture [5] and [6] and a CAD-Tool [7] to provide the flexibility of a FPGA while still maintaining a certain degree of real-time performance. This is provided by a medium-grained architecture that operates by using cells of size 4-bit or 8-bit data. Operations such as a 32-bit multiplier would require a range of cells combined into producing the required computational power, but the architecture also provides

a wider range of operations, not being limited in the same way as some coarse-grained architectures. The CAD-Tool provided by J.Larson[7] explored several approaches on the architecture behind the CAD-Tool. We used this architecture to create a new version of the CAD-Tool and provided the designer with multiple options when creating algorithms and simulations.

1.1 Overview

The architecture suggested in [6] describes an architecture provided by 3 different layers.

1. Elemental Layer
2. Local Network
3. Global Network

The CAD-Tool however will add the Module layer as this provides additional functionality and may be considered as a different layer as the once mentioned above. This last layer is not part of the proposed architecture provided in [6], but simply a layer provided for the designer to enhance the usability of the system. The architecture uses a matrix of 64x64 cells connected by pipelined interconnections. Algorithms and computational operations are provided by cells that are combined into modules. Chapter 2 and 3 provides detailed description of each layer in the architecture.

1.2 Elements

Every cell consists of 4x4 elements that are configured through a lookup table. They can either provide a mathematic operation(Figure 1.1) or function as a memory unit (Figure 1.2).

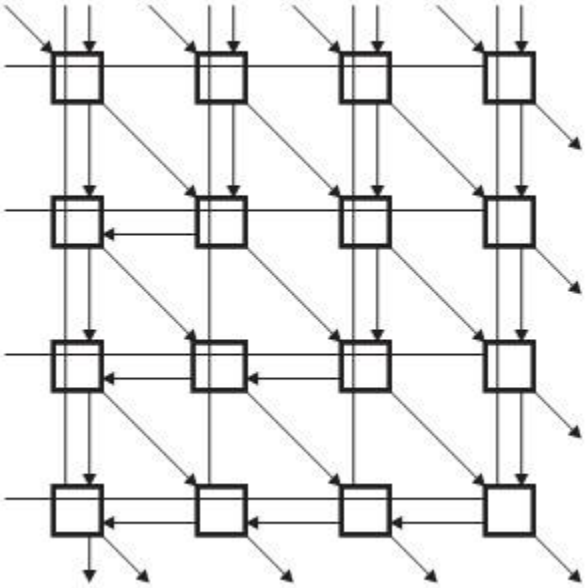


Figure 1.1: Cell in mathematics mode

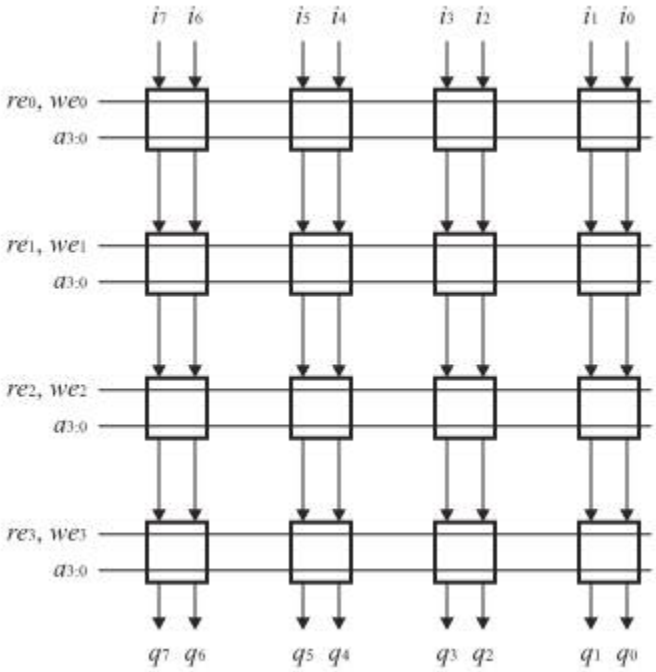


Figure 1.2: Cell in memory mode

1.3 Cells

The cells provides the encapsulation of the 4x4 elements and connect adjacent cells or directly to the global network.

1.3.1 Local Network

Between all adjacent cells lies the local network. They provide the structure for connecting input/output between each cell and their respective elements. Figure 1.3 shows cells connected over the local network.

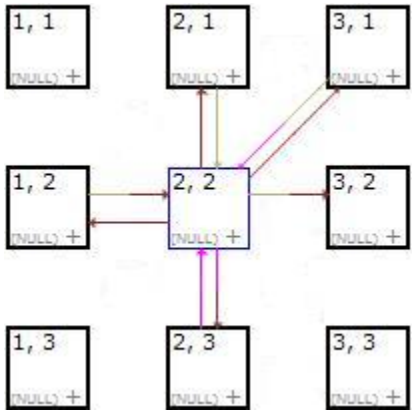


Figure 1.3: Cell-to-cell

1.3.2 Global Network

The global network provides an overlying network, enabling cells to communicate outside the adjacent network, and transfer data quickly across the network instead of routing cell-by-cell. Every 2 cells connect to one mutual global switch and thus connecting to the global network. The H-Tree structure of the global network (Figure 1.4) is very similar to a binary tree. [8] Describes the conceptual architecture in detail.

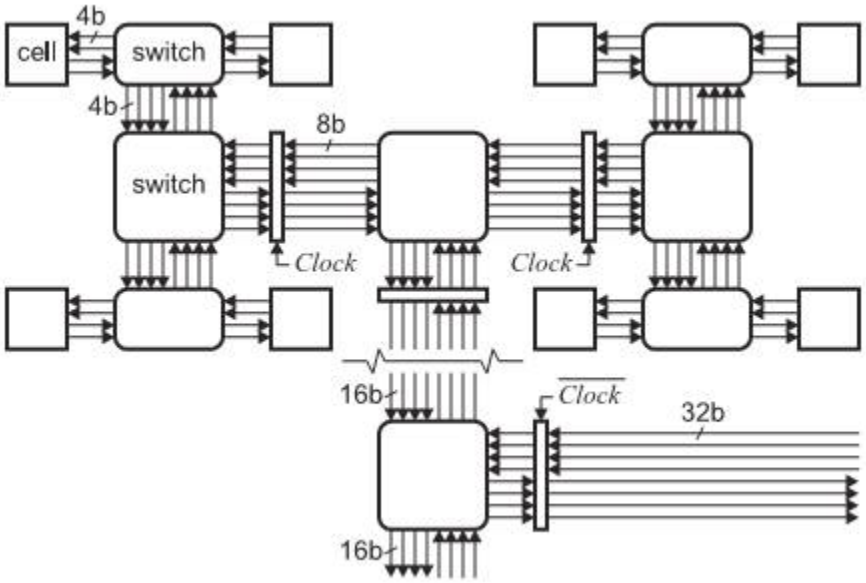


Figure 1.4: Global H-Tree

1.4 Module Layer

To be able to use the CAD-Tool in an efficient way, we designed the tools to create modules from sections of the software already designed, thus creating a blueprint we can apply to other sections of the proposed mapping. The creation and usage of the modules will be addressed in chapter 5.

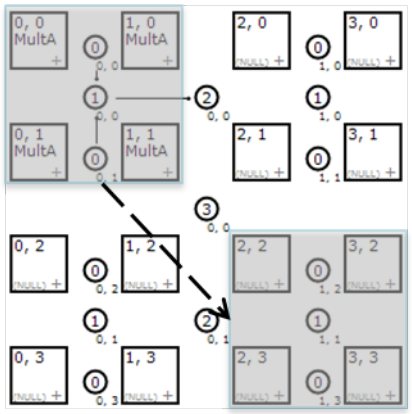


Figure 1.5: Module placement

Chapter 2

CAD-Tool Architecture

The reason for the creation of this kind of CAD-Tool is to provide the designer with the means to create a simulation of the actual model before creating it through its hardware. HiperCops reconfigurable architecture provides the designer with the capability of designing a DSP algorithm from the basic level, such as programming the elements inside each cell, to creating larger modules by combining cells into larger sections. However to maintain the relation between the hardware and emulator, we model the software as closely to the hardware as possible, making the hardware design our primary blueprint.

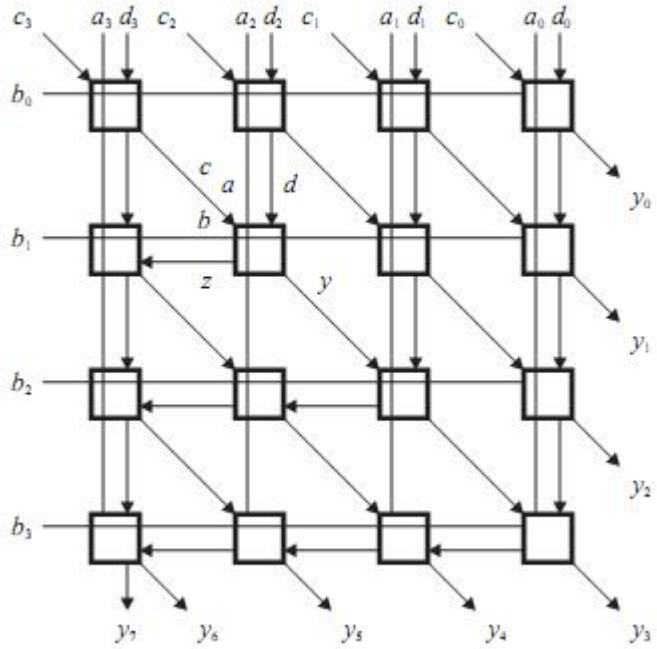
2.1 Cell

The cell is the foundation of the HiperCops architecture. Every cell is a matrix of 4x4 elements as described in detail in [06]. These elements provides either (1) An arithmetic operation or (2) Operates as an matrix for memory storage.

1. Memory Mode: Memory cells are needed to store data and results throughout any compilations inn most DSP operations.

2. Mathematic Mode: The arithmetic operators are needed to create algorithms for both simples operations such as multipliers, or more complex operations are required by Fast Fourier Transform (FFT) or filters such as FIR. As given by Figure 2.1 below

we have a,b,c,d as inputs to the matrix while *y0-y7* provides the output.



2.1.1 Cell elements

The elements provide the reconfigurable operational functionality in the system, including operations such as:

- Multiplication
- Addition
- Subtraction
- Bit shifting
- Memory access

- And control logic.

Chapter 5, Modules, describes the creation of these components. For more details about the conceptual design see[06].

2.1.2 Cell mapping

Table 2.1 shows the mapping between cells. E.g. If cell(0,0) connects to cell(1,0) through wire 4, the corresponding input wire in cell(1,0)would be 7.

Table 2.1: Cell-to-cell routing

Cell Output	Cell Input
0	5
1	4
2	7
3	6
4	1
5	0
6	3
7	2

Table 2.2: Routing table entry for cell

A	D1	D0	r	r	r	r	r	r	r	r	r	r	r	r	r
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
G	G	G	G	W	W	W	W	Y	Y	Y	Y	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2.1.3 Local interconnections

Cells connect to other cells through wires, namely input wires and output wires. These wires ether connect the output from the elements to another cell or directly to the closest

global switch. As we notice from figure 2.1 any cell can connect to the directly adjacent cells through a mesh of wires.

Field	Definition
A	Declares whether the input is active
D1, D0	Delay bits. Used during pipeline for the synchronization
r	Reserved bits
G	Contains the global output bits
W	Denotes a connection to the diagonal cell
Y	1. Input. Maps the input wire to the elements 2 according to Table(2.4). 2. Output. Maps the output wire to 1 of the 4 global input buses.

Table 2.3: Fields in routing entry

a	b	c	d	w	x	y	z
7	6	5	4	3	2	1	0

Table 2.4: Cell Input/Output mapping

2.1.4 Global interconnections

The global network provides the cell with access to parts of the architecture that is not directly adjacent to the cell, thus enabling the network with more flexibility as any output can be directed to any cell in the network.

2.2 Global Switches

The global network is a series of switches enabling cells to make connections across the network i.e. outside its adjacent cells. There is no processing inside each switch, only routing mechanisms that route date from the input wires to the output wires. Figure 2.2 gives us a detailed picture of the global network and the connectivity. First (figure 2.2, left) we see how the global network is created in the emulator, giving a fully connected global network up to level 4, where level 0 provides the link between cells and the global network. The second figure (figure 2.2, right) shows the connectivity on which H-Tree and the relationship between every level in the global network and the size of the buses creating the up and down path between every switch.

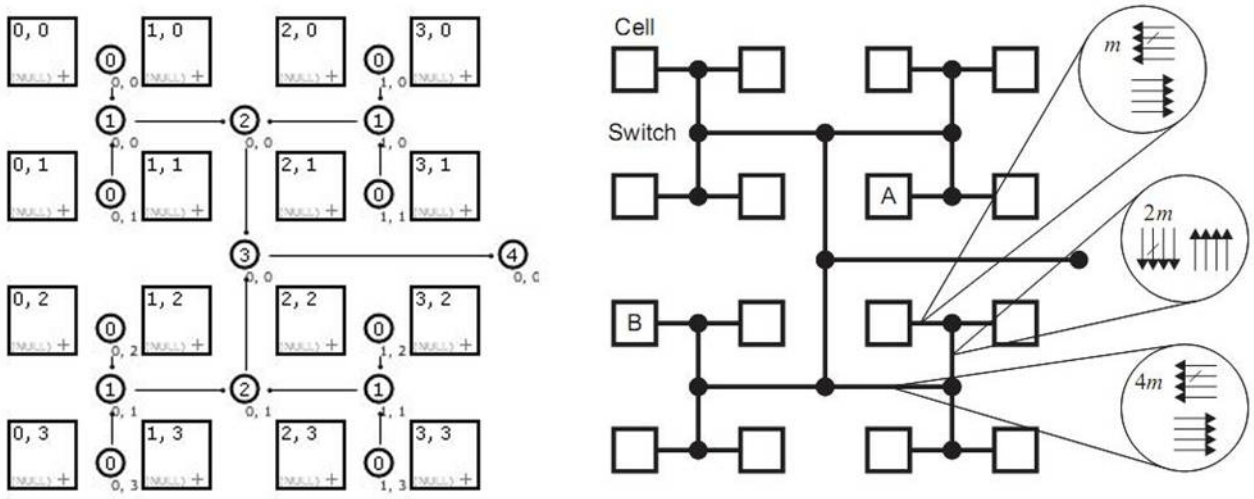


Figure 2.1: H-Tree

2.2.1 Global switch information

Consisting of 12 layers (0-11), the global network provides local network, i.e. cell-to-cell interaction, the ability to connect cells/modules that are not adjacent. Every cell can access

the global network and connect to any cell by accessing by connecting to the adjacent global switch (level 0 of the global network) and up connecting the global network to a switch on a higher level, then routing down another direction to reach the designated switch that connects to the destination cell. Chapter 3 describes the interaction between the different layers in more detailed.

Table 2.5: Global switch mapping

Layer	Number of Global switches	Bits on bus to previous layer	Bits on bus to next layer
0	$64x32$	4	8
1	$32x32$	8	16
2	$16x32$	16	32
3	$16x16$	32	64
4	$8x16$	64	128
5	$8x8$	128	256
6	$4x8$	256	512
7	$4x4$	512	1024
8	$2x4$	1024	2056
9	$2x2$	2056	4096
10	$1x2$	4096	8192
11	$1x1$	8192	-

Table 2.4 show the number of global switches on each level and their corresponding number of buses. Starting on level 0, every switch connects to two adjacent cells through its 2 buses, 4 bits on each, and given that if both cells connect all available wires to the designated switch, the unlink to level 1 will have to be 16 bits.

As the terms output and input are relative in accordance with the direction of the current connection, we can deduct the following:

1. Any connection UP the global network, will give OUTPUT bits = 2 x INPUT bits.

2. Any connection DOWN the global network will give INPUT bits = 2 x OUTPUT bits.

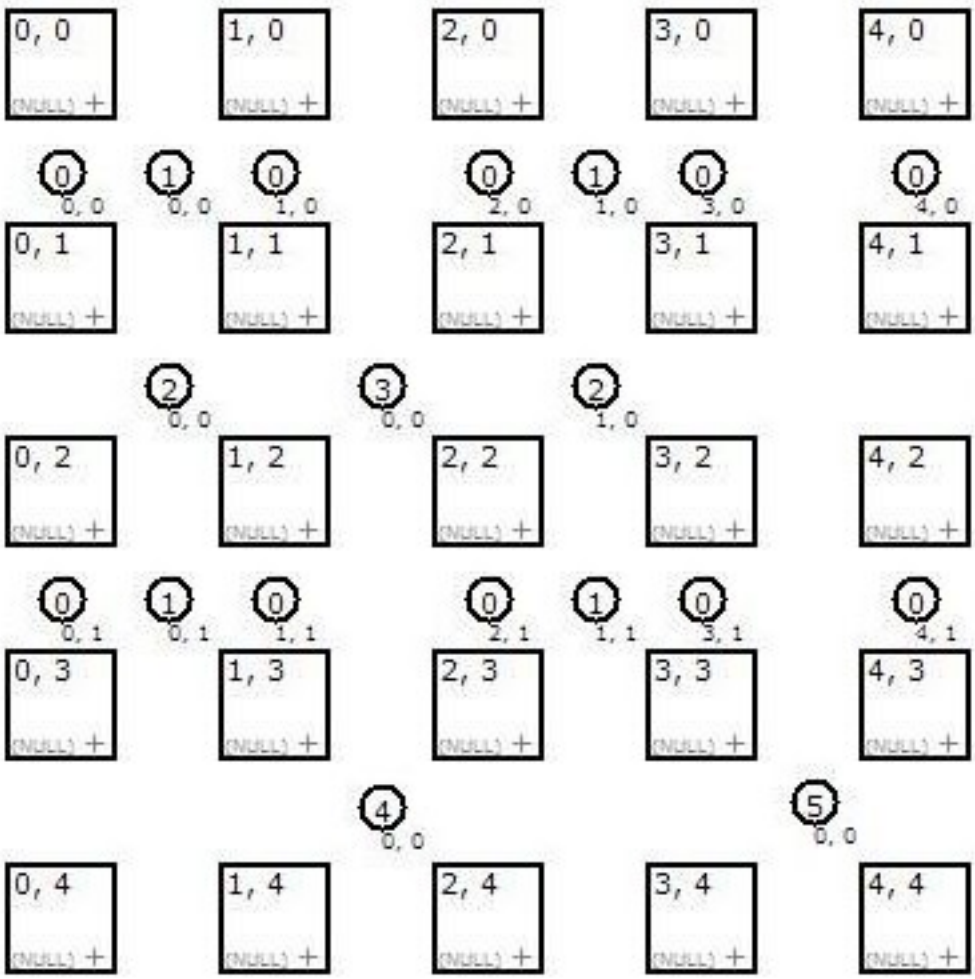


Figure 2.2: Global switch network

The distinction between input and output bits provides the foundation for the upper and lower connections described in chapter 3.

2.2.2 Orientation

The relation between the global switch and the adjacent switches/cells are given in figure 2.3.

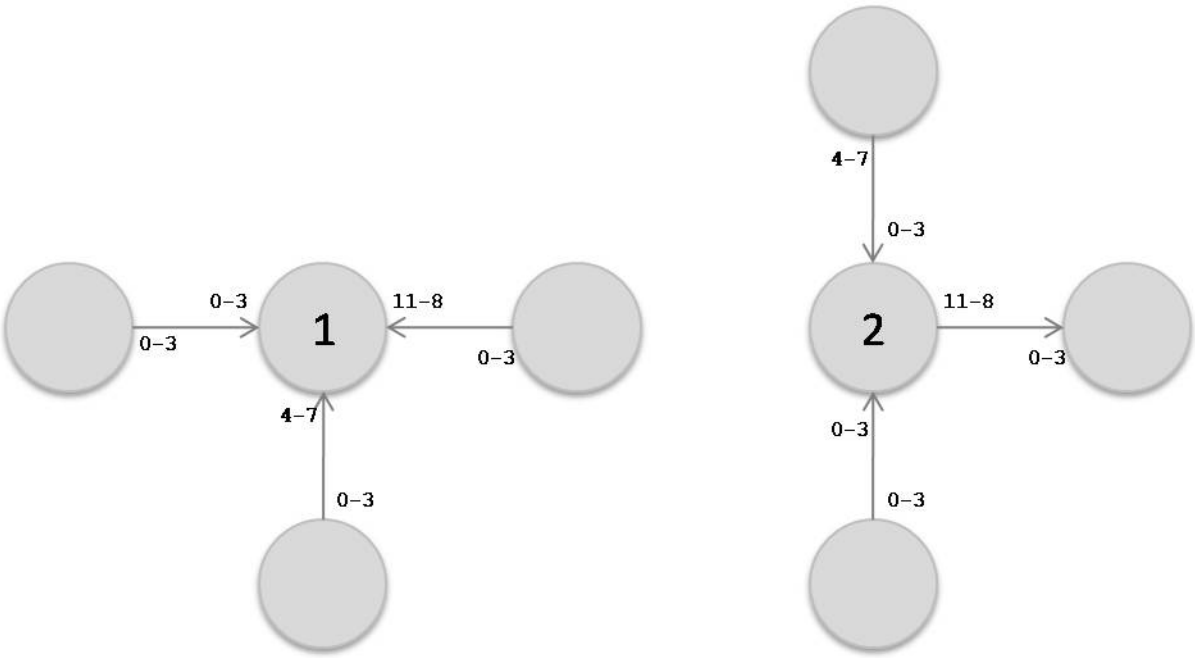


Figure 2.3: Relational layout, global switches

2.3 Buses

As mentioned above, interactions between the cells occur either through the local network or the global. In both cases they use the buses to reach another cell or switch. On the lowest level, i.e. "cell-to-cell" or "cell to global switch (level 0)", 4-bit connections connects adjacent cells and connects cells to the global network.

2.3.1 Visual Mapping

Figure 2.4 shows the conceptual design of the bus mapping, provided in [6]. Chapter 4 provides more detailed information about the visual feedback and programming of the buses.

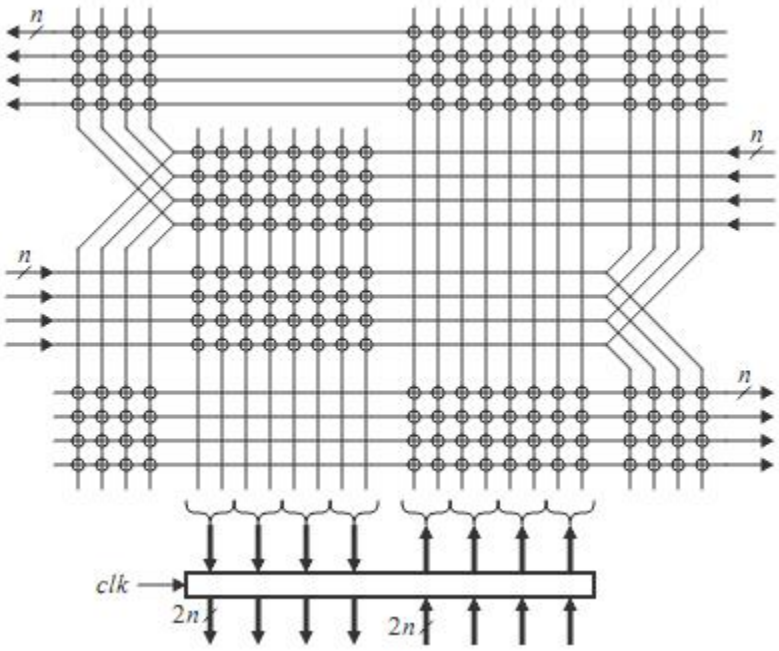


Figure 2.4: Global Switch, bus routing

Chapter 3

Connections

Chapter 2 described in detail the inner workings of the cells and the global network; we will now look further into the relationship between these two components. To simplify the picture, we might say that the global network is a as mesh of unconnected switches, where each switch is of different size (see chapter 2.3. buses) depending on the location of this switch inside the network, but where every switch has the exact same structure and purpose; creating routing between cells. One realistic example would be to create an FFT, as illustrated by figure 3.1, using 3 multipliers where each of the multipliers would be a module created by connecting 4x4 cells. But first, we'll look at the how each of the underlying sections of the system creates both global and local connections.

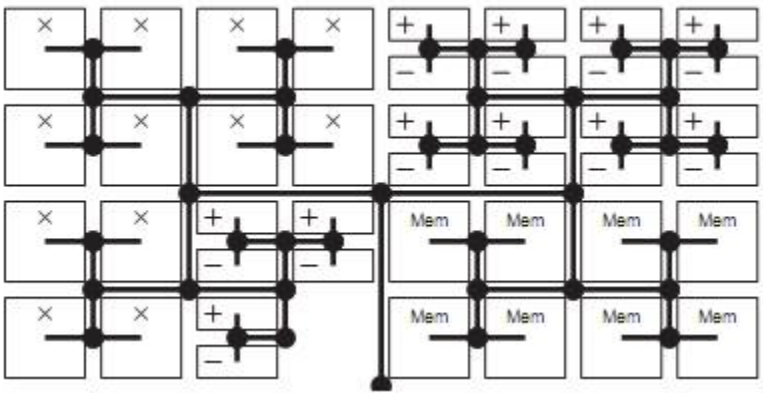


Figure 3.1: Fast Fourier Transform

3.1 Cell connections

As mentioned in chapter 2, any cell can connect to any adjacent cell (see figure 3.1) through 4-bits output/input buses. However there is a difference between a connection to cell and one connecting the switch to the global network. We will discuss both connections below.

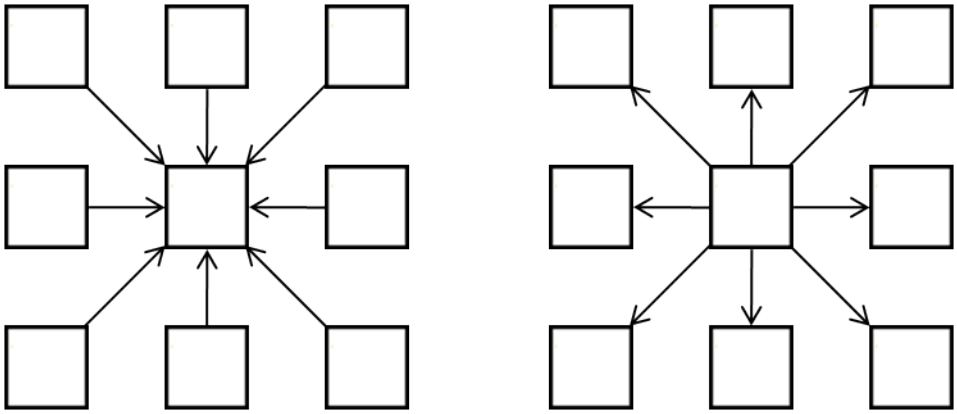


Figure 3.2: Cell to Cell

3.1.1 Local

Figure 3.3 displays how a user connects one cell to another. The process can be broken down into 3 pieces:

1. Select source cell, and select one of the following *outputs: A, B, C, D, W, X, Y, Z.*

2. Select destination cell. This cell must be adjacent to the source cell. The designer selects one of the following *inputs: a, b, c, d, w, x, y, z*

3. CAD-Tools then maps the connecting according to the selected wire input/output.

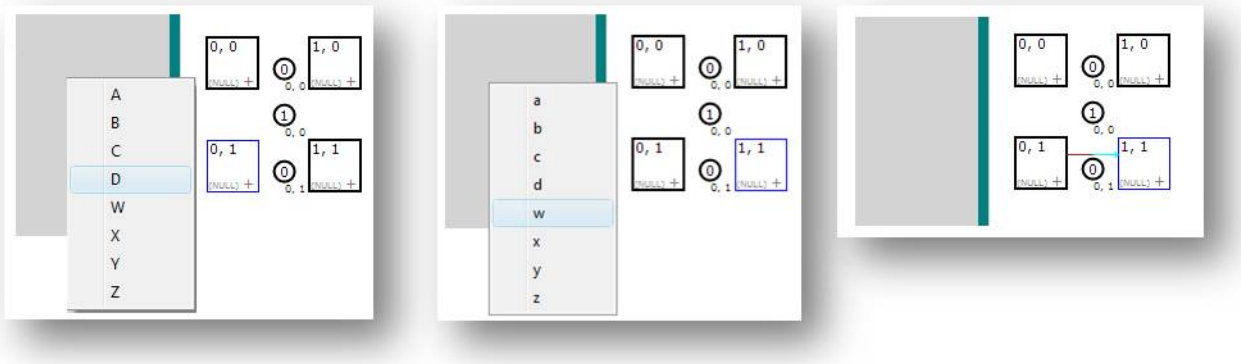


Figure 3.3: Cell-To-Cell

3.1.2 Global

When a cell connects to the global network, the exact wiring will depend on the position of the selected cell according to the adjacent global switch. The distinction relates to the selected cells position in regards to its adjacent global switch, figure 3.4 shows how we separate between these two distinctions in the CAD-Tool. The wiring will be directed accordingly:

1. Cell connecting from the left side of the global switch: *0-3*
2. Cell connecting form the right side of the global switch: *8-11*

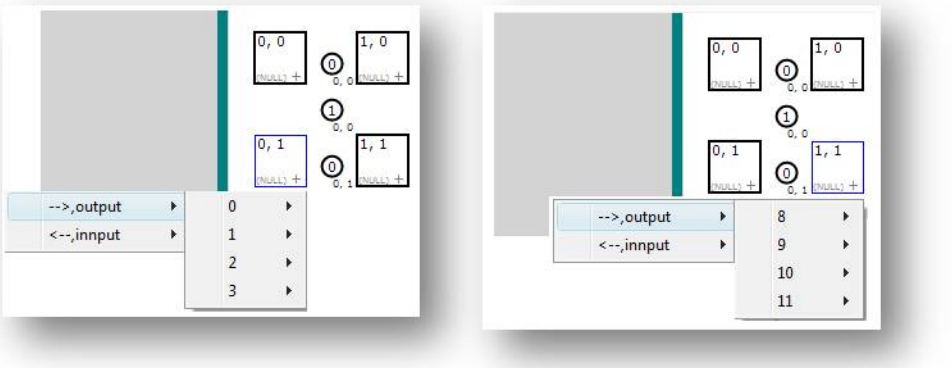


Figure 3.4: Cell to Global switch

3.2 Global Switches

The global network provides the system with the possibility of connecting cells to non-adjacent cells across the network by accessing a higher level en thus enabling faster communication then any cell-to-cell interaction. There are however several different types of connections that can be made from one cell/switch to another cell/switch.

- Output to output. Routing wires up the H-tree from a switch or cell to a switch on a higher level.
- Input to Input. Routing wires down the H-tree from a switch to a switch or cell.
- Output to input. Routing wires down the H-tree to a cell or switch, bridging the output and the input lines.

3.2.1 Output to output

Figure 3.5 shows a connection from a switch to another switch connected by the *output-to-output* connection. This kind of connection is used when a designer wants to set up a wiring up the network to a switch of a higher.

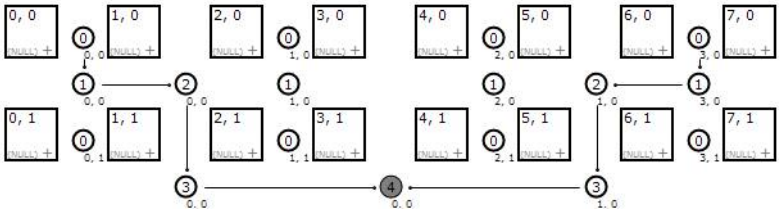


Figure 3.5: output-to-output

Another example (see figure 3.6) on when to use the *output-to-output* would be during a simulation and we wanted to monitor the output on certain wires going out of global switch(5,0,0) providing the results between two multiplier and two adders.

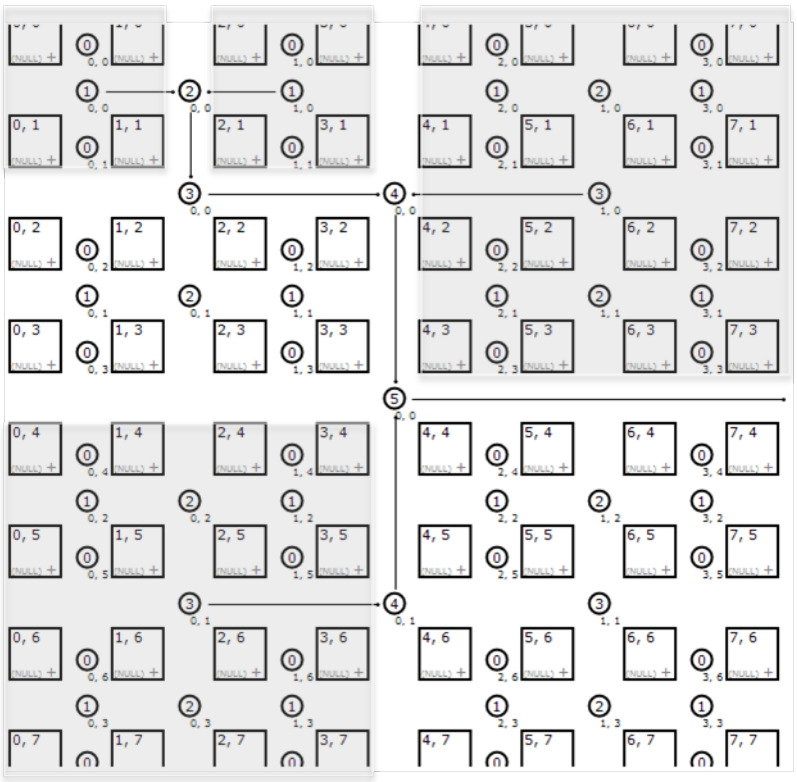


Figure 3.6: output-to-output

3.2.2 Input to input

Figure 3.7 shows wiring going down the network, using *input-to-input* connection. Whenever results from on section of the network would provide input to another, this kind of connection would be used by wiring from a higher level switch down to a switch then to the destination cell.

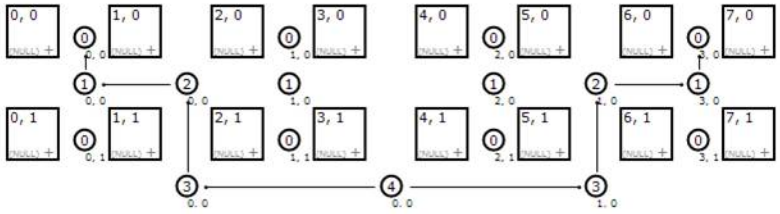
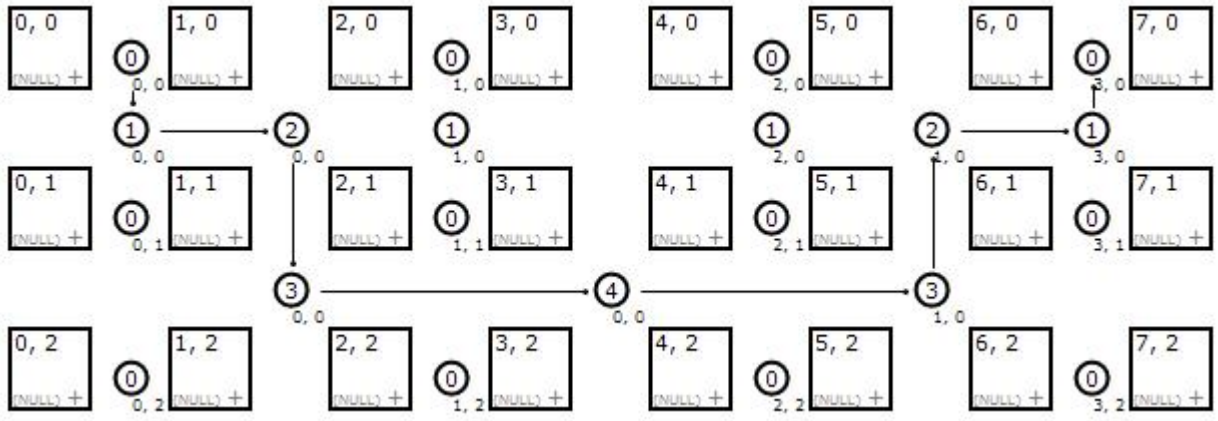


Figure 3.7: input to input

3.2.3 Input to output

To bridge the two connections we use *output-to-input* as show in figure 3.2.3.



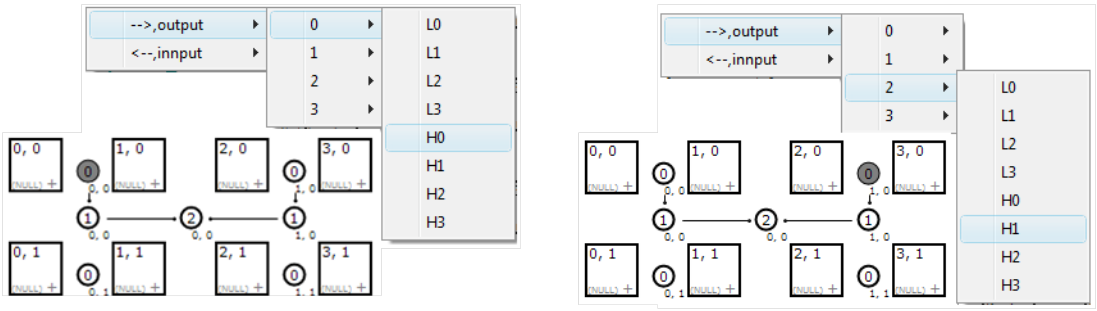


Figure 3.8: Bus selection

3.2.4 Editing the global network

To ensure that the designer access the correct sections of the global network and does not select any switches that are already taken, any menu will always show the current state of the bus currently accessed. In figure 3.8 the designer creates two connections. One from global switch (0, 0, 0) and another from (0,1,0), both connecting to global switch (2,0,0) and both are output connections. As we can see from figure 3.9 after the connections are made, both the buses are taken, and cannot be selected. Future development change this and make it possible to override buses that are taken and thus eliminating the connections previously connected to this bus if this would benefit the designer.

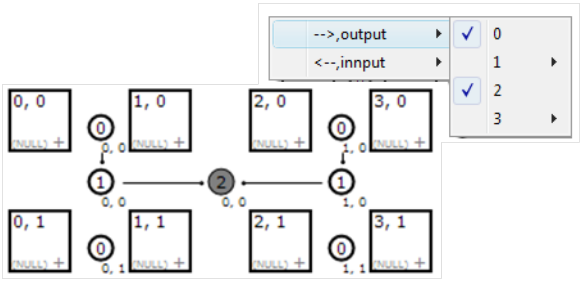


Figure 3.9: Bus selection, taken

3.2.5 Graph search

To create any connection in the global network whether making a *input-to-input* or *output-to-output* connection, we use variation of the procedure described below. The procedure $nodeSearch(sourceNode, destNode)$ traverses the global network from one the source switch to another switch on a higher level. **procedure** $nodeSearch(sourceNode, destNode)$

1: $currentNode \leftarrow sourceNode$
2: **while** $currentNode \neq destNode$ **do**
3: $\quad nextNode \leftarrow getNextNode(currentNode)$
4: $\quad createPath(currentNode, nextNode)$
5: $\quad currentNode \leftarrow nextNode$
6: **end while**

The procedure $nextNode(node)$ returns the next switch on the next level by looking at the field where the current switch is located. As the global switch structure shifts every even/odd times,i.e.

1. If current switch level is even, we need to find the switch on level+ 1 with with the same Y-coordinate.

2. If current switch level is odd, we need to find the switch on level+ 1 with with the same X-coordinate.

procedure $nextNode(node)$

1: **if** $ODD(node_switchLevel)$ **then**
2: $\quad y \leftarrow lowestDistance(node.y, node(level + 1).y$
3: $\quad returnnode(node_x, y)$
4: **end if**

5: **if** $EVEN(currentNode_switchLevel)$ **then**

6: $x \leftarrow lowestDistance(node.x, node(level+1).x$

7: $returnnode(x, node_y)$

8: **end if**

Chapter 4

Simulator

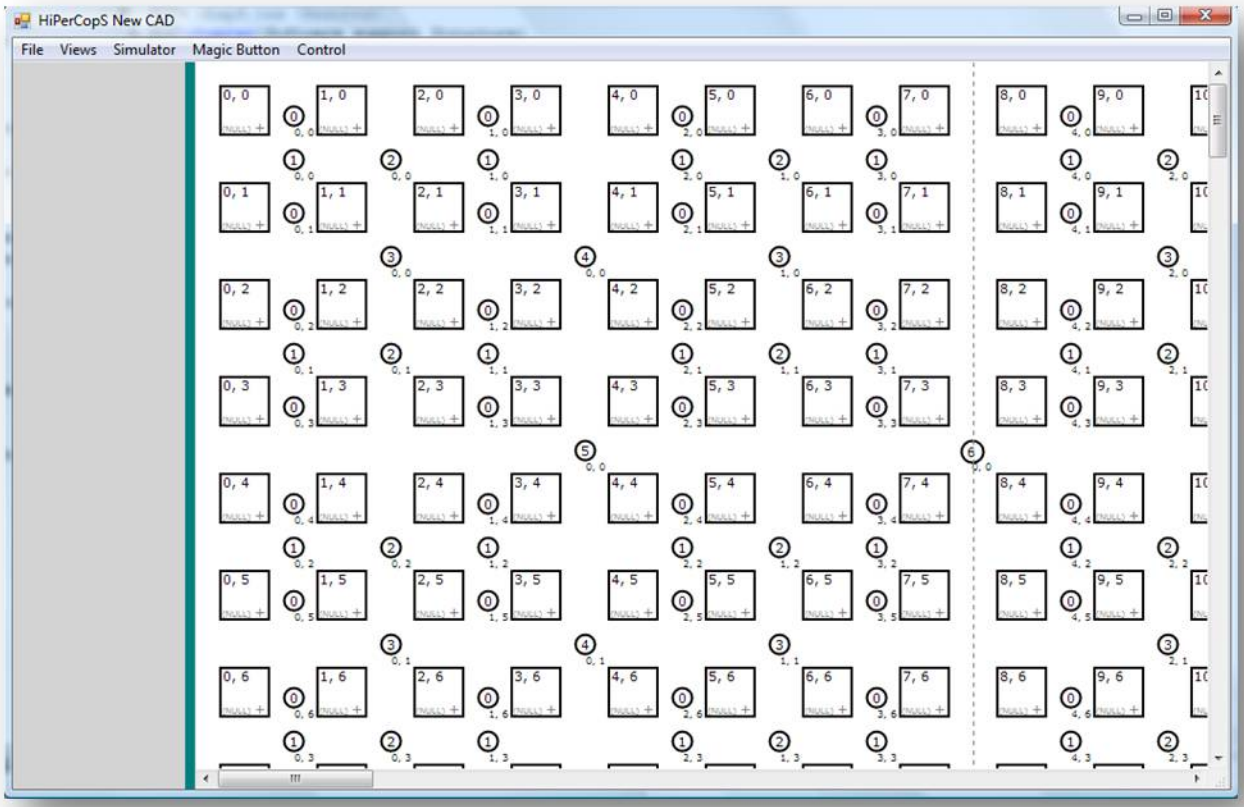


Figure 4.1: HiperCops CADTOOL

One of reasons we wanted to create a more functional interface was to create an easy to use user-interface that enables the designer to use the software as a tool to test and create simulations before creating the hardware, and thus being able to simulate the algorithms on

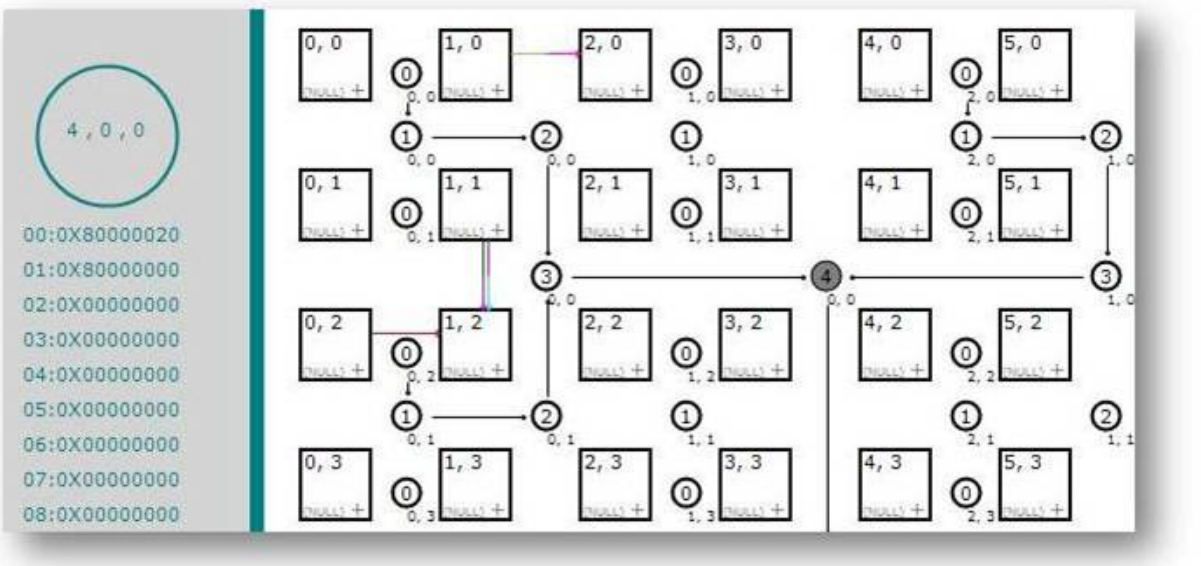


Figure 4.2: HiperCops CAD-Tools

the emulator and using this as a tool for testing may enhance the development time greatly. As a ground rule before creating any application that involves complicated interaction with a user, one should consider the field of HCI (Human Computer Interaction) before creating the graphical User interface. The following ideas were used throughout the creation:

1. The first module, GOMS, presented in [8], expressed how to achieve a certain action, e.g. "create a wire from cell(1,0) to cell (1,1)" by listing the steps needed to start and complete a task.

2. Another model, the KEYSTROKE model, also presented in [8] predicts the time it takes to access and use every key, mouse action;move the mouse from one object to another by selecting an item etc.

The purposes of both models are to discover what approach to a certain scenario would benefit the user, i.e. given several ways to conduct a task, what would be the fastest and most efficient way to do it. To be brief, the purpose is to make the software more usable

and efficient by applying the steps above to the software development.

One of the major changes to the software from the previous version is the move from a floating tool pane to a permanent "'infoPanel"' ever present to the designer. Figure 4.2 shows the old tool pane to the left and the new infoPanel to the right. We discovered that such a permanent panel increased the response time of the designer considerably.

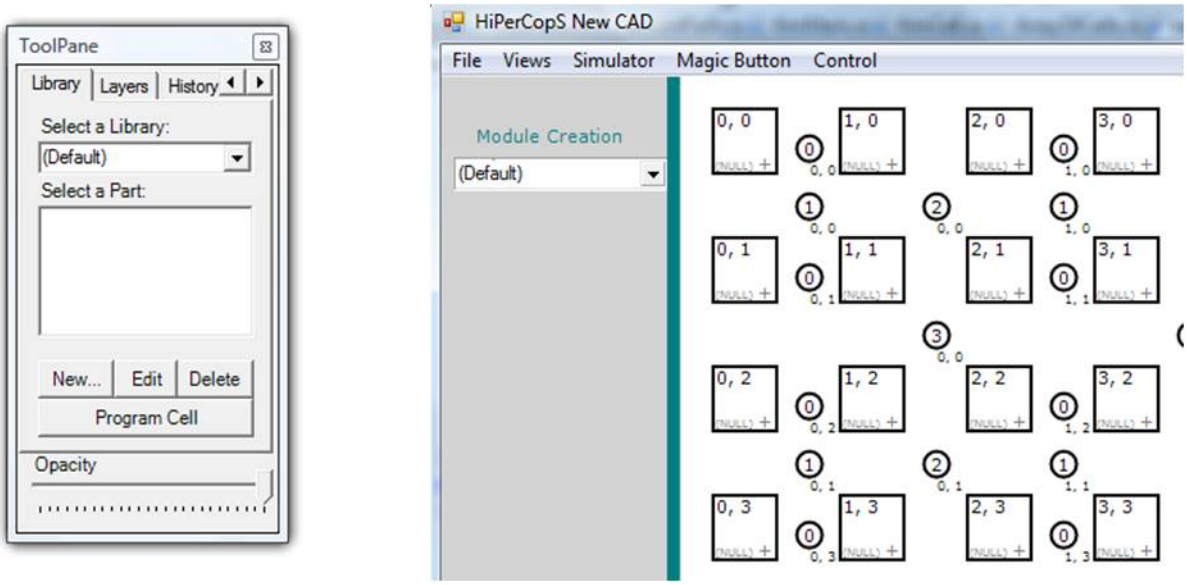


Figure 4.3: Toolpane and InfoPanel

The "infoPanel" has 3 differ

1. Global network: Switch. Provides the designer with detailed information about the contents of the selected switch, such as the available wires and bus capacity. to create a more expedient usage of the CAD-Tool as the user can identify used and available buses..

2. Simulation.

3. Module Creation. This mode is described in detail in chapter 5.

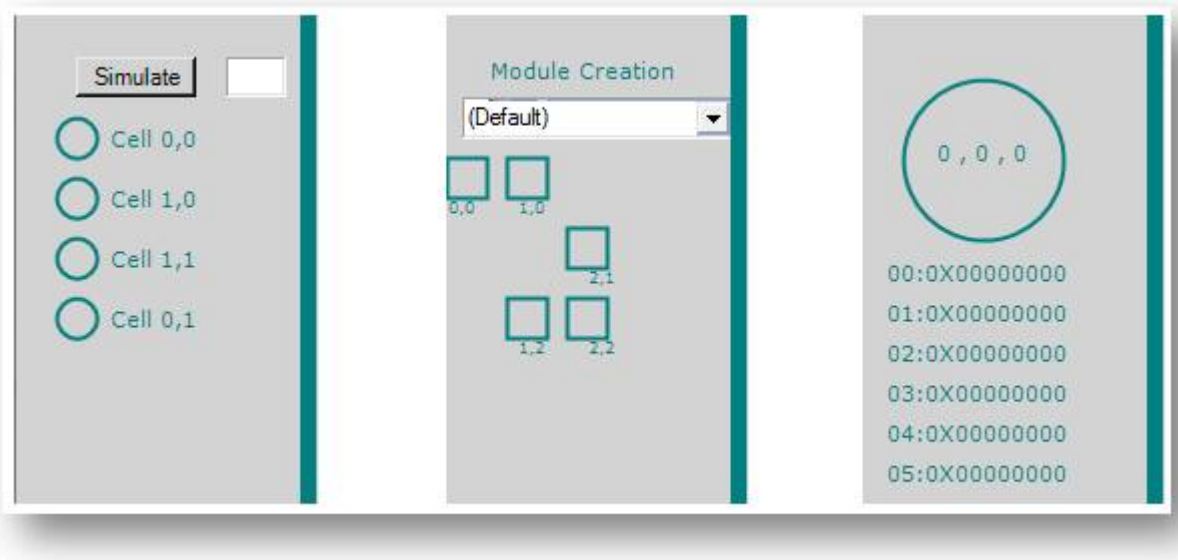


Figure 4.4: Information Panel

4.1 Wire control

To create a visual picture of the simulation and relate the information back to the designer, any given switch can be monitored by selecting this switch and adding it to the monitor panel on the left before running the simulation.

4.1.1 Design

We designed the control: "Add wires" to be as simple as possible. Figure 4.5 shows a screenshot from the display.

4.2 Storage

Given the complex structure of the system, we needed to create a storage that saves all the vital information from the three major part, cell-array, switch-switch array and not

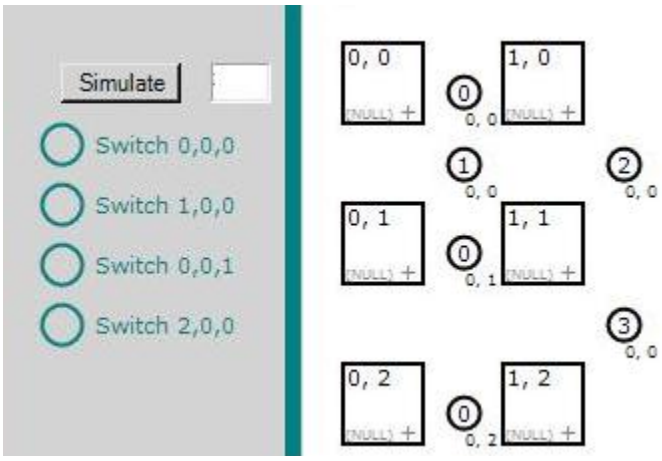


Figure 4.5: Simulator, Wire Trace

least the graphics array. Figure 4.6 describes the save-process while figure 4.7 the loading process. Figure 4.8 shows how a saved object will be stored on disk.

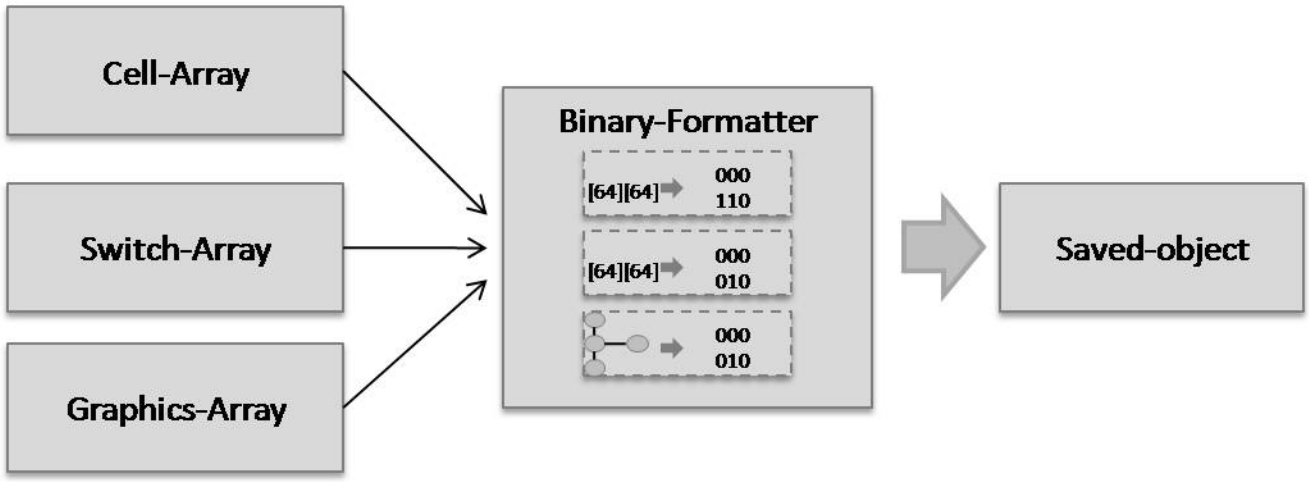


Figure 4.6: Save to File

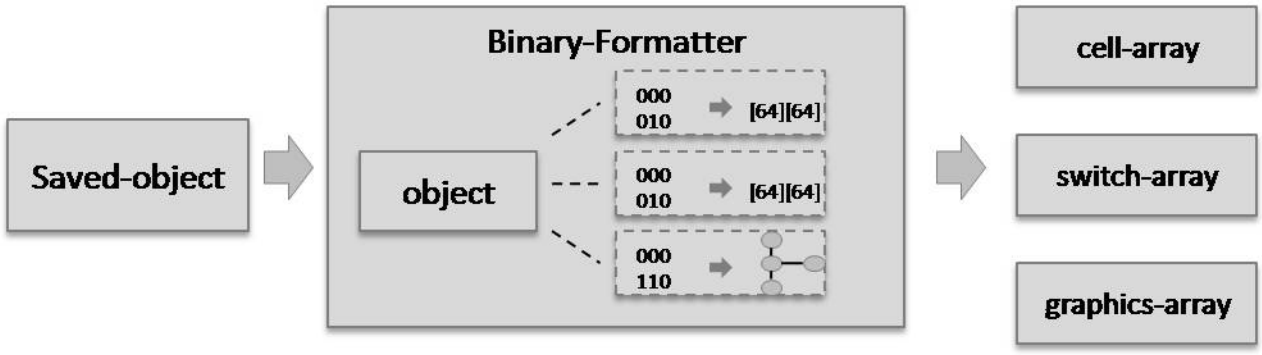


Figure 4.7: Load from file

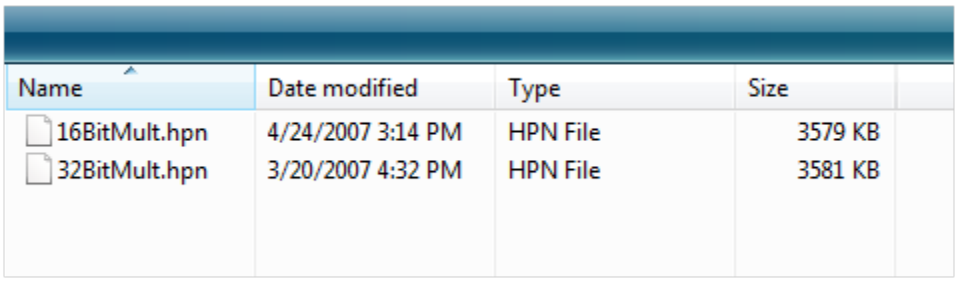


Figure 4.8: File saved

Chapter 5

Modules

The previous chapters described the system from the most basic part provided by the elements, the connections given by cells and to the global network connecting all the elements together. However, in order to use the software in a more efficient manner, the need for more complicated elements are needed.
Furthermore, the current system can simulate multiplications, additions etc, but given a more complicated scenario, such as setting up, compiling and manipulating an FFT by programming each element, connecting all the wires, and finally setting up the global network will provide the designer with a formidable task. Modules provides a solution to this problem.

5.1 Modules

The notion of encapsulation is not a new one, in fact it is on that has been used for years, and become more and more popular over the years with JAVA, C++ and C Sharp. We use modules in much that same way, by encapsulating specific objects and rescuing them elsewhere in the application, while still maintaining the same functionality. The creation of a module can be divided into 3 different sections. Selection, creation and placement.

Figure 7.1 illustrated the use of modules as the designer selects the specific cells and created the module.

1. Controls: Create Module

2. Right-Click on the desired cells. Select "Add cell".

3. Keep adding until all desired cells are present

4. Save the module.

5. Placing a module: Select one module from the list, and placing it by selecting a cell.

A detailed description of the module creation is presented below.

5.1.1 Selection

Before we can create any new module we need to declare the cells we intend to use in the new module. Given the specific structure of the system, there need to be strict rules under which cells can be selected. To scenario in figure 5.2 below describes how we select and the assumptions we operate under.

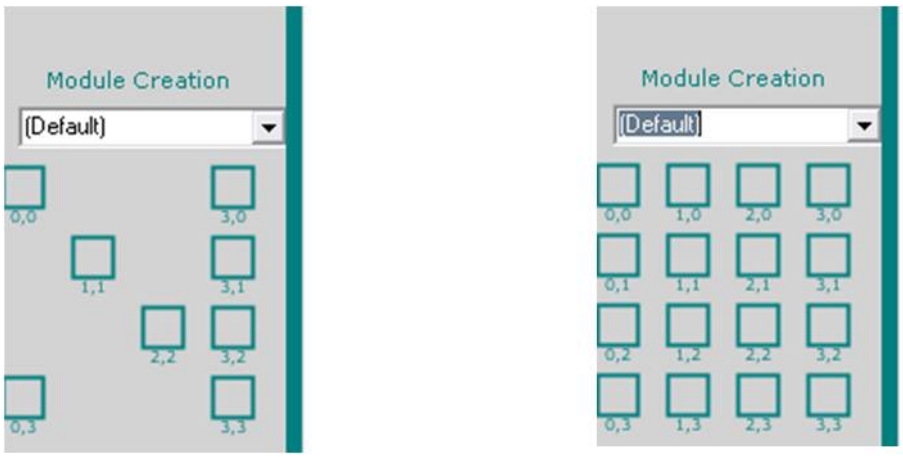


Figure 5.1: Selecting Cells

1. Selecting cells: The selected cells can only have wires inside the designated area i.e. if wires connect to the outside world, these will be ignored since including these wires would greatly complicate the Creation and Placement methods. It is also assumed

that if the designer wishes to include the functionality under certain cells, he will include them in the selection.

2. As cells are added to the matrix, they are aligned according to the their relative position to the rest of the cells. Figure 7.1 provides an example on how these cells are added to the matrix.

5.1.2 Creation

Once all the item has been selected, the designer must save the compilation and thus creating the module. There are currently no limitations/restrictions on the actual creation, except the ones mentioned in Selecting cells. Later it might be necessary to create a compiler that determines if the module follows specifications declared by the designer e.g. 32 bit multiplier using only a certain amount of memory.

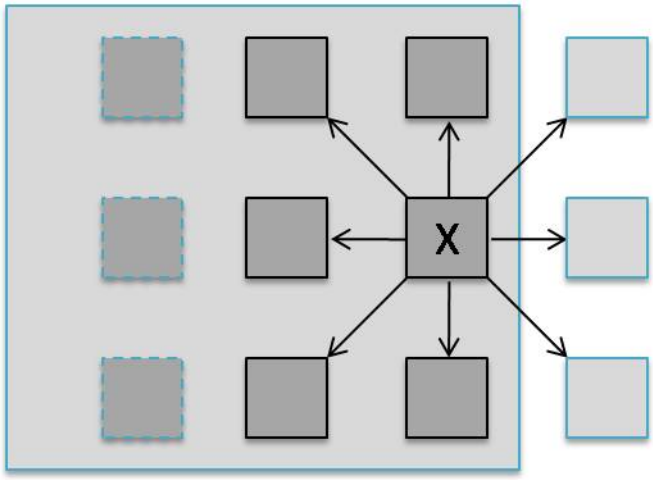


Figure 5.2: Create Module

The creation of the module can be summarized accordingly:

Once the cells have selected, we need to create a generic module in such a way that it can

be applied anywhere in the system as long as an identical structure is present. Therefore, what we need from the matrix of cells is the relationship between them; the wiring from every one cell within the module area. Figure 5.2, Create Module, shows how these cells are selected. Remember, the global switches will not be considered as they can be added to the new connection after the module has been placed.

Furthermore, we need to retrieve the compilation of the elements within each cell. However, since they are not related to anything outside the cell, i.e. beside the wires they are connected to, we only need to add them to the structure, without manipulation. The module is then stored to the library by selecting the store menu.

5.1.3 Placement

Since we now have a working module present in the library we can apply this module almost anywhere in the emulator. There are a few limitations however.

1. The location of the "Source Cell", the cell from which we apply the module and thus the upper left of the original module, and the "Destination Cell", the lower right of the module must be applied within type of cell structure as the module was created; the structure of the module in relation with the connecting global network must be compatible with the area we are applying it to.

2. Any previous programming of the destination cells will be removed. A future edition to this function might be to create a tool that merge the module with an already programmed cell and still maintaining the previous structure.

Chapter 6

Conclusion

This thesis has provided tools to create, modify and observe medium-grained DSP algorithms through a robust graphical interface. The CAD-tool enables the designer to create large algorithms and benchmark any specific section of the system.

6.1 Contribution

The CAD-tool now provides the designer with a full-scale simulation tool such as:

1. The creation of algorithms on a large scale through module creation and applying these modules to other sections of the software.
2. Creation modules and storing them for future use and development.
3. Simulation observation. Observe the simulation by selecting observations point in the architecture.

6.2 Future work

There are numerous additions that can be made to any software of this kind, and will only be limited to usage it is intented for, but some of the additions may be:

1. Scripting tool. Create a scripting tool that programs the cell architecture by using a set of predefined modules, cell configuration etc. This might enhance the performance

of the CAD-tool.

2. Research the design of CAD-tools and produce a more efficient and usable designer-tool by applying usability studies commonly used in Human computer interaction (HCI) research.

References

[1] R. Hartstein, "Coarse grain reconfigurable architecture,"in *Proc. 6th Asia South Pacific Automation Conferance*, Yokohama, Japan, pp. 564-570, 2001.

[2] C. Ebeling, D. Cronquist,P. Franklin and C. Fisher "RaPiD- a configurable computing architecture for compute-intensive applications," *University of Washington, Department of Computer Science and Engineering Tech Report TR-96-11-03*, Nov, 1996.

[3] R. Hartstein, M. Hertz, T. Hoffman, U. Nagledinger, "Using the KressArray for reconfigurable computing," *Proc. SPIE, vol 3526, pp. 150-161*, Oct 1998.

[4] P. Heyster and G. Smith, "Mapping of DSP algorithms on the MONITUM architecture,"in *Proc. Internationl Parallel and Distributed Processing Symposium*, pp. 180-185, Apr 1003.

[5] M. Myjak "A two-level reconfigurable cell array for digital signal processing" *M.S. Thesis*, Whashington State University,May 2004.

[6] M. Myjak "A medium-grained reconfigurable architecture for digital signal processing" *PhD. Dissertation*, Whashington State University,May 2006.

[7] J. Larson, "CAD TOOL EMULATION FOR A TWO-LEVEL RECONFIGURABLE ARRAY FOR DIGITAL SIGNAL PROCESSING," *M.S. Thesis*, Whashington State University,2005.

[8] S.K. Card, T.P. Moran and A. Newell "The psychology of Human-Computer Interaction" *PhD. Dissertation* Hillsdale, NJ, Lawrence Erlbaum Associates, 1983.

Appendix A

```
public class sectionPath
   {
       public ArrayOfCells aOfCells;
       public float[,] gpPoints;
       public sectionPath()
       {
           this.gPath = new GraphicsPath();
           this.aOfCells = new ArrayOfCells();
       }
       public sectionPath(GraphicsPath GP)
       {
           this.gPath = GP;
       }
       public sectionPath(GraphicsPath GP, ArrayOfCells AC)
       {
           this.aOfCells = AC;
           this.gPath = GP;
       }
       public void format(GraphicsPath gp)
       {
           gpPoints = new float[gp.PointCount, 3];
           int length = gp.PointCount;
```

```
    for (int i = 0; i < length; i++)
    {
        gpPoints[i, 0] = (float)gp.PathData.Points[i].X;
        gpPoints[i, 1] = (float)gp.PathData.Points[i].Y;
        gpPoints[i, 2] = (float)gp.PathData.Types[i];
    }
    int a = new int();
}
public HiperCopsControls.ArrayOfCells connectGlSwToGlSw(ArrayOfCells cSarray, S
{
    SwitchBlock curr_node = new SwitchBlock();
    SwitchBlock next_node = new SwitchBlock();
    curr_node = src;
    if (src.switch_level > dest.switch_level)
    {
        curr_node = dest;
        dest = src;
    }
    while (next_node.switch_level < dest.switch_level)
    {
        next_node = nextSwitch(curr_node, glBSw_all);
        cSarray.CreateGlobalSwitchToGlobalSwitch(curr_node.switch_level, curr_n
            next_node._x, next_node._y, wireInd, upper, wirePos);
        curr_node = next_node;
    }
```

```
    return cSarray;
}
private SwitchBlock nextSwitch(SwitchBlock CurrentSw, SwitchBlock[][,] glBSw_al
{
    SwitchBlock next = new SwitchBlock();
    int L0 = glBSw_all[CurrentSw.switch_level + 1].GetLength(0);
    int L1 = glBSw_all[CurrentSw.switch_level + 1].GetLength(1);
    int dist1 = 100;
    int dist2 = 100;
    int loc1 = 0;
    int loc2 = 0;
    //Locate on X plane(same Y)
    if (cadPanel.IsOdd(CurrentSw.switch_level))
    {
        for (int i = 0; i < L0; i++)
        {
            if (distance(CurrentSw._cellX, glBSw_all[CurrentSw.switch_level + 1
            {
                dist1 = distance(CurrentSw._cellX, glBSw_all[CurrentSw.switch_l
                loc1 = i;
                loc2 = CurrentSw._y;
            }
        }
    }
    //locate on Y plane(identical X)
```

```
    else if (cadPanel.IsEven(CurrentSw.switch_level))
    {
        for (int i = 0; i < L1; i++)
        {
            if (distance(CurrentSw._cellY, glBSw_all[CurrentSw.switch_level + 1]
            {
                dist2 = distance(CurrentSw._cellY, glBSw_all[CurrentSw.switch_l
                loc1 = CurrentSw._x;
                loc2 = i;
            }
        }
    }
    next = glBSw_all[CurrentSw.switch_level + 1][loc1, loc2];
    return next;
}
public int distance(int a, int b)
{
    if (a < b)
        return b - a;
    else
        return a - b;
}

public void connectGlSwToGlSw(ArrayOfCells cSarray, SwitchBlock sbStart, Cell c
{
```

```
}
public void reCreate()
{
    int length = gpPoints.GetLength(0);
    GraphicsPath gpNew = new GraphicsPath();
    PointF [] pts = new PointF[length];
    byte [] bts = new byte[length];

    for (int i = 0; i < length; i++)
    {
        pts[i] = new PointF(gpPoints[i, 0], gpPoints[i, 1]);
        bts[i] = (byte)gpPoints[i, 2];
    }
    if (length > 0)
    {
        this.gPath = new GraphicsPath(pts, bts);
    }
    else
        this.gPath = new GraphicsPath();
}

[NonSerialized]
public GraphicsPath gPath;
```