ChipGen User Manual

Neuroengineering Research Lab
University of Pennsylvania

10 January 2006

Abstract

This is a brief document to help organize your design files to use the AER layout compiler developed in UPenn's Brains in Silicon lab.

Contents

List of Figures

List of Tables

Chapter 1

ChipGen Specifications

1.1 Required Files & Cells

The following files are required for proper execution of ChipGen:

1. Layout Files
   (a) AER cell library (typically named chiplib.tdb).
   (b) Metapixel[1] layout.
2. Spice Files
   (a) AER circuits library (typically named genlib.sp).
   (b) Metapixel netlist.
   (c) Subcircuits file.
   (d) Layout definition file.

The AER layout cell library and spice circuits library will not be discussed for they pertain to word-serial AER transmitter[2] and receiver[3] implementation and are beyond the scope of this document.

[1]The term **metapixel** is used to indicate there is more than one basic element within the layout cell, e.g. a mini 2x2 array of neurons. Thus, a 32x32 chip generated using a metapixel with a 2x2 neuron group actually contains 64x64 neurons.

[2]K Boahen, A Burst-Mode Word-Serial Address-Event Channel-I: Transmitter Design, *IEEE Transactions on Circuits and Systems I*, vol 51, no 7, pp 1269-1280, July 2004.

[3]K Boahen, A Burst-Mode Word-Serial Address-Event Channel-II: Receiver Design, *IEEE Transactions on Circuits and Systems I*, vol 51, no 7, pp 1281-1291, July 2004.

1.1.1 Metapixel Layout Cells

The metapixel layout file is a single Tanner L-Edit file (.tdb) with the following cells:

1. PixelFootPrint—the main metapixel cell to be arrayed in the chip.
2. PixelFootLeft—the cell located at the left edge of every array row.
3. PixelFootRight—the cell located at the right edge of every array row.
4. PixelFootTop—the cell located at the top edge of every array column.
5. PixelFootBottom—the cell located at the bottom edge of every array column.
6. PixelFootAll—a cell with PixelFootPrint and all the edge cells (PixelFootLeft, -Right, -Top, and -Bottom) together.
7. Additional Cells—cells within the PixelFootPrint.

The naming convention is important—the layout generator looks for those specific names (case-sensitive). The specifications for each cell is found within section 2.

1.1.2 Metapixel Netlist

Three spice netlist (.sp) files are required; within these files, it is important that digital and analog power supplies be named **DVdd/DGnd** and **AVdd/AGnd** to insure proper netlist connectivity. Appendix A contains sample netlist files.

Main metapixel netlist: pixel.sp

This file is the top-level circuit for the metapixel. There should be no subcircuits defined in this file (only instanced), and all the node names should match the names within the layout definition file (see below). The default filename is **pixel.sp**, however it need not be named so.

Subcircuit definition file: subckts.sp

This spice netlist file contains all of the subcircuit definitions found within the main metapixel file. If there are no subcircuit instances within **pixel.sp**, then this file is not necessary. However, if required, the filename must be **subckts.sp**.

Layout definition file: layout.sp

This file links the netlist file to the metapixel layout file, and must be named **layout.sp**. The PixelFootPrint cell has signals exiting all four edges to connect to neighboring cells or edge cells. This file is used to define the signals and their appropriate edges so that the netlist generator can correctly wire the pixels and edge cells together.

Each line in this file defines an edge and its signals, in the following format:

edge={*Signal1*,*Signal2*,*Signal3*,...}

Edge can be one of (case sensitive):

- LEFT, RIGHT, TOP or BOTTOM (required)
- GLOBAL (required)
- LEFTEDGE, RIGHTEDGE, TOPEDGE or BOTTOMEDGE (optional)

For *edge* = LEFT, RIGHT, TOP or BOTTOM, the braces define the list of signals leaving the cell in order from top to bottom (for LEFT and RIGHT edges) and from left to right (for TOP and BOTTOM edges). **It is important that the order of the signals is correct**, otherwise the pixels will be incorrectly wired. For example, the first signal on the BOTTOM edge of a metapixel is connected to the first signal on the TOP edge of the metapixel immediately below. Obviously, opposite edges **MUST** have the same number of signals; there is no restriction on the number of signals. However, the naming convention of the signals is important. A *row* or *column* signal, i.e. a signal that is the same along a row or column, is defined by using the same signal name on opposing edges. For example, if the file contained the following two lines:

```
.layout
LEFT={TSelY,OutL}
RIGHT={TSelY,OutR}
END
```

TSelY is a *row* signal, as it has the same label on the left and right side of the array. The different names for the next signal imply that there is circuitry between the two labels within the pixel. However, OutR of one pixel will be connected to OutL of the pixel to its right.

GLOBAL is used to define global biases and power signals. Signals within this group still need to be defined within LEFT, RIGHT, TOP or BOTTOM to ensure proper netlist connectivity. It is important that these signals are distinguished as global so that they will be defined in the top-level instance and correctly connected to pads.

Edge = LEFTEDGE, RIGHTEDGE, TOPEDGE or BOTTOMEDGE defines how to connect the signals at the end of the pixel array. **This is only really useful for local connections.**

Remember all global signals and transmitter or receiver signals are already connected to circuitry outside the array. The default state (i.e. the case where LEFTEDGE, RIGHTEDGE, TOPEDGE or BOTTOMEDGE categories are left out) is to leave all remaining signals hanging at the edge of the pixels. However, in specific circumstances, it may be required to tie these hanging signals to **Vdd** or **Gnd**. This is performed in the following manner:

- Copy one of the edge statements (e.g. LEFT) and relabel the category (i.e. change LEFT to LEFTEDGE).
- Change only the signals that you wish to tie to a specific node by changing the name in the list to the desired nodename. Using the example above, another line may be added LEFTEDGE={TSelY, Vdd}. This indicates that the **OutL** signal along the left side of the array will be connected to **Vdd**. If you wish to leave the node hanging, **keep the original name**.
- Note: the EDGE categories must be placed *after* the inter-pixel category (e.g. LEFTEDGE must be defined after LEFT)

Finally, the last line in each file must be *END*.

See Appendix A for sample files.

1.2 Layout Compilation

This section provides a more in-depth description of the layout rules that must be followed in order to properly generate the chip. First, a brief description of the general layout of the chip is needed in order to fully understand the overall structure of the chip.

1.2.1 Chip Layout

Figure 1.1 shows the basic layout of the chip. The following will be descriptions of signals used by the AER receiver and transmitter. See Table 1.1 for a summary.

Receiver signals

The receiver circuitry is located to the top and left of the metapixel array (see Figure 1.1). As such, receiver signals should only be found within the PixelFootTop and PixelFootLeft cells. Receiver signals, and their edge entrance, are as follows:

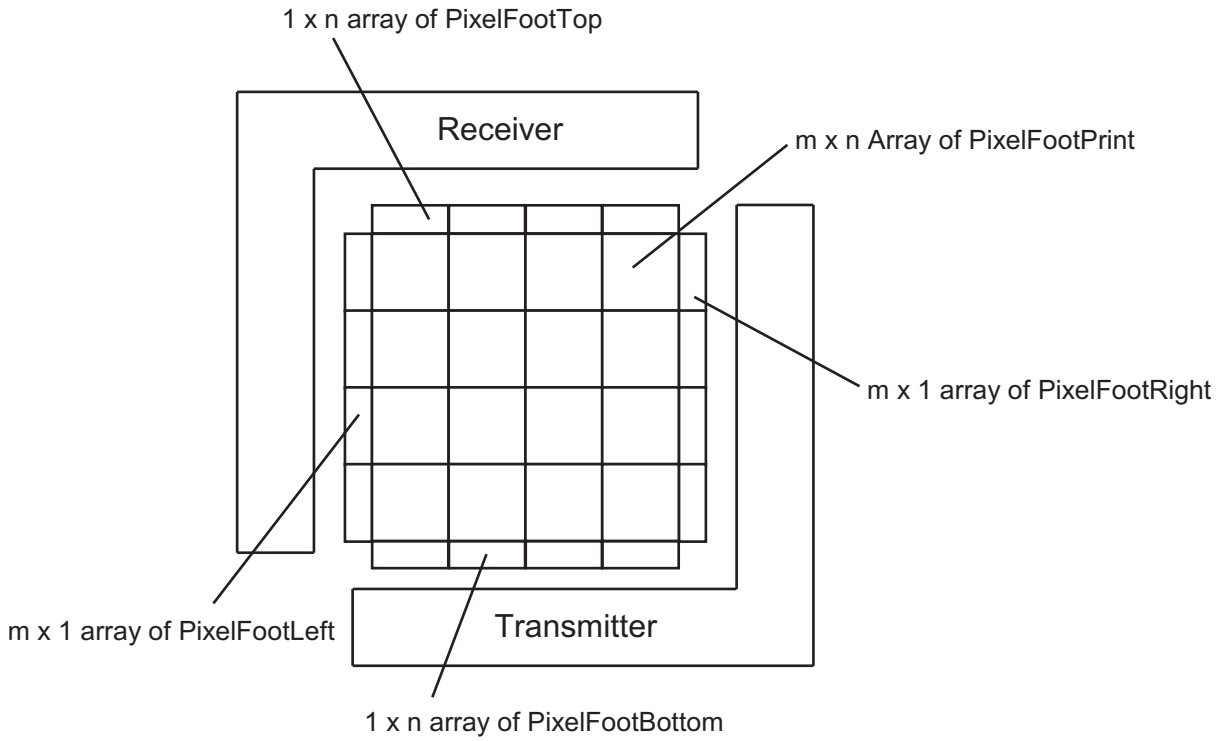

Figure 1.1: Chip Layout

Overall layout of the generated AER chip (without pads). The metapixel cells are tiled with their abutment boxes abutting. Edge cells abut the PixelFootPrint and other edge cells.

1. Column Select signal—This signal is Active Low and enters the array through the PixelFoot-Top cell. Typically named ∼RSelX (not a requirement)[4].

2. Row Select signal—This signal is Active Low and enters the array through the PixelFootLeft cell. Typically named ∼RSelY.

3. Pixel Acknowledge signal—This signal is also Active Low and exits the array through the PixelFootLeft cell. Typically named ∼PixAck.

Note that there are restrictions on transistor sizes and on the input logic circuitry for receiver signals. See Figure 1.4 for details.

Transmitter signals

The transmitter circuitry is located to the bottom and right of the pixel array (Figure 1.1). As such, transmitter signals should only be found within the PixelFootBottom and PixelFootRight cells. Transmitter signals, and their edge entrance, are as follows:

[4]When a signal's "typical" name is specified, it will be referred throughout the document as such. However, when designing your cell, it need not be named so.

Signal	Active	Edge Cell	Default Name
Receiver Column Select	Low	PixelFootTop	∼RSelX
Receiver Row Select	Low	PixelFootLeft	∼RSelY
Receiver Row Acknowledge	Low	PixelFootLeft	∼PixAck
Transmitter Column Request	Low	PixelFootBottom	∼TReqX
Transmitter Row Request	Low	PixelFootRight	∼TReqY
Transmitter Row Select	High	PixelFootRight	TSelY

Table 1.1: AER Signal Descriptions

1. Column Request signal—This signal is Active Low and exits the array through the PixelFootBottom cell. Typically named ∼TReqX.
2. Row Request signal—This signal is Active Low and exits the array through the PixelFootRight cell. Typically named ∼TReqY.
3. Row Select signal—This signal is Active High and enters the array through the PixelFootRight cell. Typically named TSelY.

Note that there are restrictions on transistor sizes and on the output logic circuitry for transmitter signals. See Figure 1.4 for details.

Vdd and Ground signals

Analog Vdd and Gnd signals enter the metapixel array through the PixelFootTop and PixelFootBottom cells. Digital Vdd and Gnd signals enter the metapixel array through the PixelFootLeft and PixelFootRight cells (Figure 1.2).

Global bias signals

Global bias signals may exist in any metapixel edge cell. If present, biases along the bottom and left edge cells are routed to the bottom of the chip and biases along the top and right are routed to the top (Figure 1.3A). There also exists an option to route all biases to the top of the chip. Exercising this option will remove the bottom padframe, thus allowing more space for the metapixel array (Figure 1.3B).

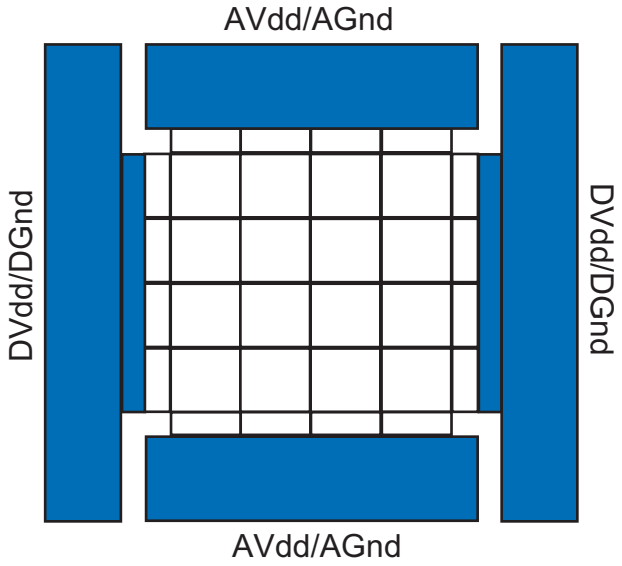

Figure 1.2: Power Bus Layout

DVdd (Metal5) and DGnd (Metal4) abut the entire left edge of the PixelFootLeft abutment box and the entire right edge of the PixelFootRight abutment box. AVdd (Metal5) and AGnd (Metal4) abut the entire top edge of the PixelFootTop abutment box and the entire bottom edge of the PixelFootBottom abutment box.

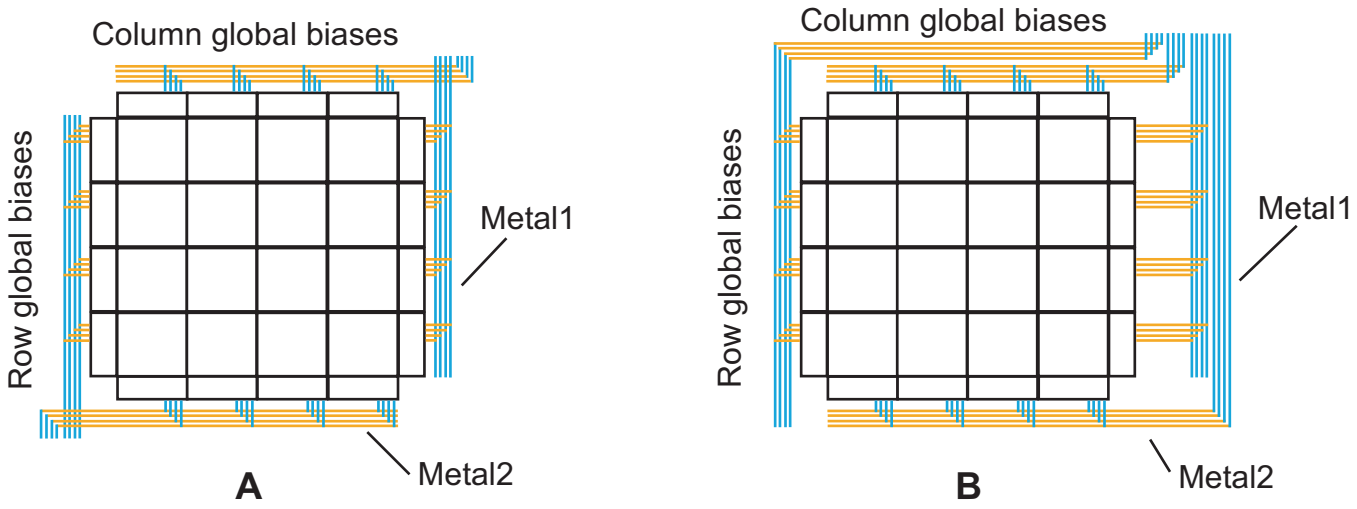

Figure 1.3: Global Bus Layout

A. Global bus. Global row biases are collected on Metal1 (blue) lines running vertically to the left and right of the metapixel array, connecting to Metal2 (orange) ports on the left edge of PixelFootLeft and right edge of PixelFootRight. Global column biases are collected on Metal2 lines running horizontally at the top and bottom of the metapixel array, connecting to Metal1 ports on the top edge of PixelFootTop and bottom edge of PixelFootBottom. B. Global row and column biases routed to the top of the chip.

1.2.2 MetaPixel Layout

This section describes specifications for the metapixel layout.

PixelFootPrint

The PixelFootPrint is the core of the metapixel array: it contains the neural circuitry that is tiled in the chip.

There are restrictions on the interface circuitry between neurons and AER receiver and transmitter circuitry. The circuitry required for a neuron to communicate with the transmitter and receiver are shown in Figure 1.4 (see caption for details). Note that these circuits are required within the neuron, as well as transistor size requirements for those transistors drawn in red (see Figure 1.5).

Some comments on the PixelFootPrint:

1. The abutment box for this cell is set by placing a port called 'Abut' on the Icon/Outline layer. THIS IS VERY IMPORTANT! The metapixels are arrayed with their abutment boxes abutting (Figure 1.6).
2. The PixelFootPrint needs to be rectangular.
3. Ports must be placed on ALL the signals exiting the metapixel. The port must be the same width as the wire leaving the cell.
4. Ports with the same name must be differentiated using an index, even if they will be the same signal (e.g. a global bias). For example, suppose you have a metapixel with a 2x2 array of neurons, each sharing a global bias called 'Vbias'. Due to layout issues, you have two signals labeled 'Vbias' exiting the cell, one for each row. Though these two signals will eventually be connected, they must be differentiated with an index (starting at zero). The port labels must be modified to 'Vbias_0' and 'Vbias_1'; the '_' between signal name and index is **NECESSARY**.
5. There are no restrictions on what level metals are needed exiting the metapixel. The edge cells (see Section 1.2.3) are used to route the signals to the required layers.
6. Minimum height (or width) must be:

$$\max(r \times 73\lambda, t \times 84\lambda)$$

where r and t represent, respectively, the number of receiver and transmitter rows (or columns) per metapixel.

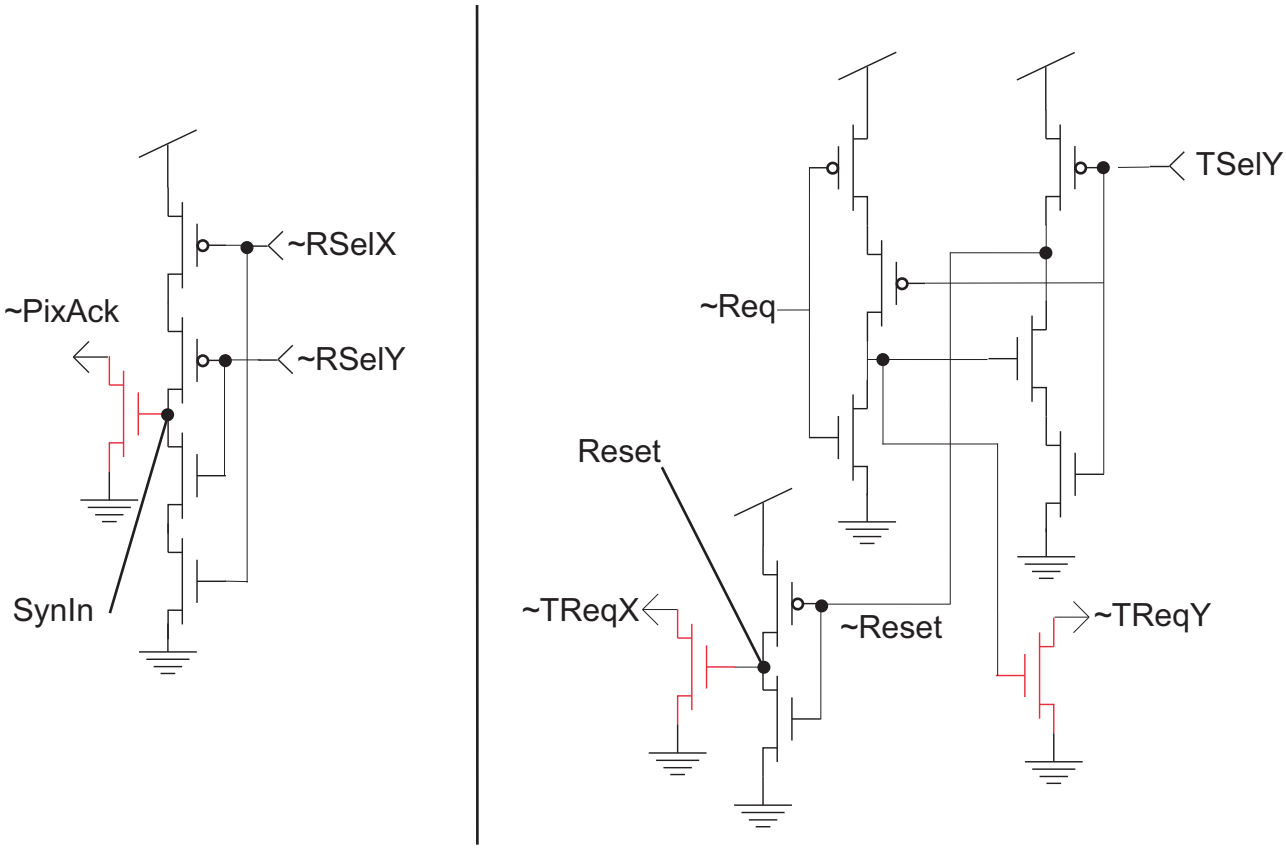

Figure 1.4: Required Interface Circuitry Between AER Receiver/Transmitter and Neurons. Transistors in red (∼PixAck, ∼TReqX, and ∼TReqY) indicate a required transistor size (see Figure 1.5). Active low signals are indicated by '∼'. Left: AER neuronal input circuitry. The receiver signals are ∼RSelX, ∼RSelY and ∼PixAck. SynIn is the input into the 'synapse' of the neuron (e.g. current-mirror integrator). Right: AER neuronal output circuitry. The transmitter signals are ∼TReqX, ∼TReqY and TSelY. ∼Req is the active low signal from the neuron (e.g. spike from neuron). Reset and ∼Reset are the signals used to reset the neuron.

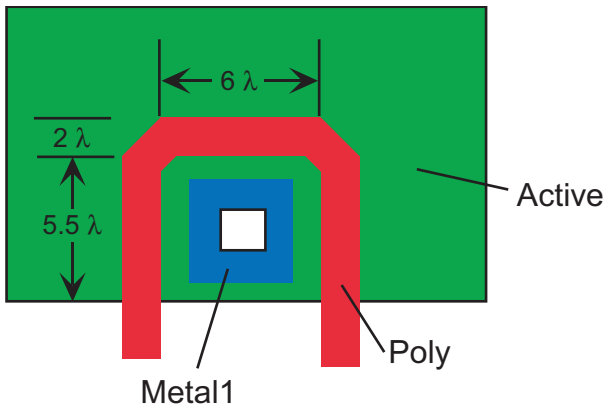

Figure 1.5: Transistor Sizing

The transistor driving Receiver signal ∼PixAck and Transmitter signals ∼TReqX and ∼TReqY must have minimum channel length and be U-shaped as shown in the figure. The Metal1 layer represents the drain (receiver or transmitter signal) of the transistor, while the source is attached to Ground.

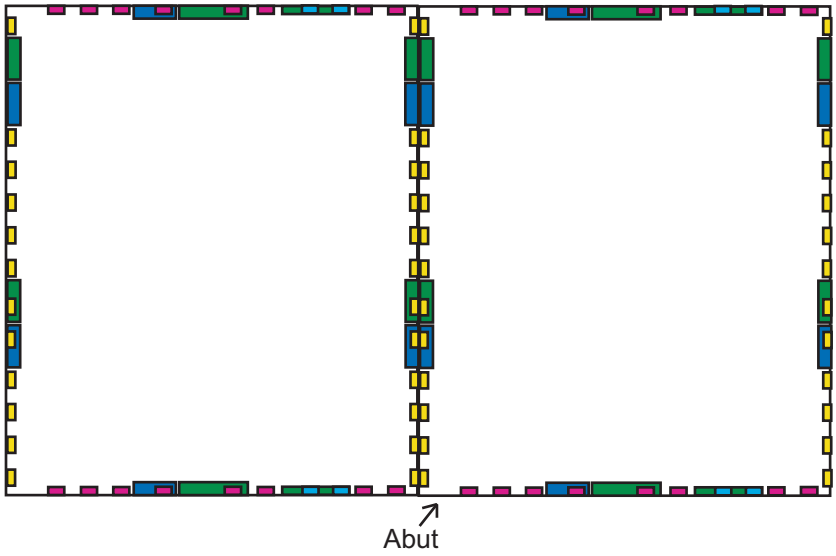

Figure 1.6: 2x1 Array of Metapixels
Cells are arrayed according to their abutment box, which is a port called 'Abut' on the Icon/Outline layer.

Layout recommendations:

1. Digital Vdd and Gnd enter the metapixel array through the left and right sides of the metapixel array, while Analog Vdd and Gnd enter the array through the top and bottom (Figure 1.2). It is strongly recommended that your metapixel has all these signals (or at least 3) entering the metapixel from the appropriate side. Any signals that cannot be routed as such from within the metapixel can then use the edge pixels to route them around. For example, your layout may require you to have Analog Gnd exiting the right side of the metapixel. In that case, PixelFootRight can be used to route Analog Gnd to PixelFootTop; it will extend beyond PixelFootRight's abutment box.
2. From the perspective of the automatic compiler, the metapixel is a black box whose only interface are ports on the edges. The compiler looks for ports corresponding to signals identified in the "chip.txt" parameter file (Section 2.2). These signals can run on any layer in PixelFootPrint, but must be converted to Metal1 (for vertical signals) or Metal2 (for horizontal signals) in the edge cells in order for the compiler to connect them appropriately. See Figure 1.7 for an example.

1.2.3 Edge Cells

This section describes layout specifications for PixelFootLeft, PixelFootRight, PixelFootTop and PixelFootBottom. Comments applying to all edge cells:

1. The edge cells have two primary purposes:
   (a) Divert various signals or power supplies traversing the cell to the appropriate layers. See the appropriate subsection below for specific details.

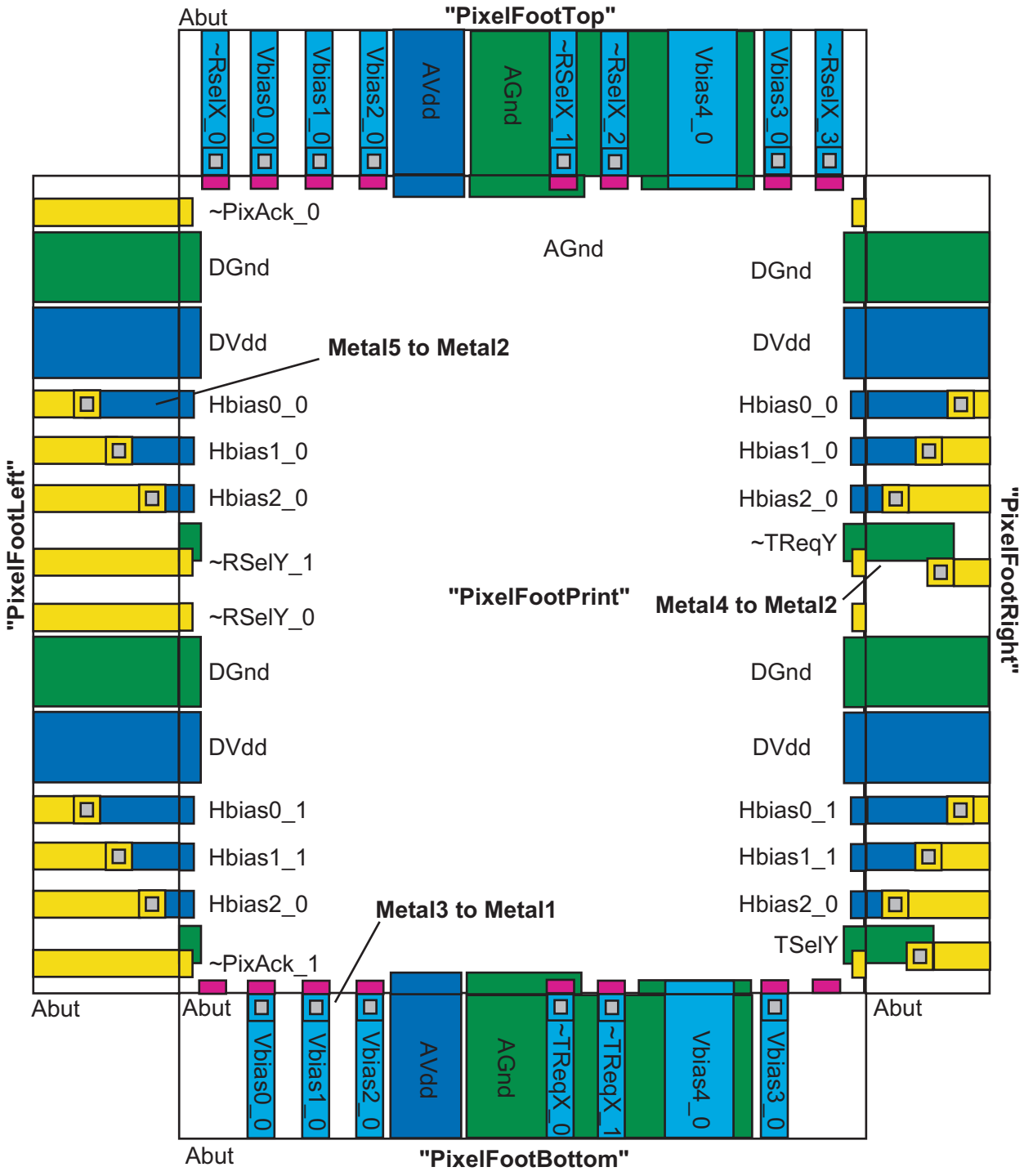

Figure 1.7: Sample PixelFootPrint Connectivity

Ports on the left and top edges of PixelFootPrint should align with ports on the right and bottom edges. Edge cells convert PixelFootPrint ports to the correct metal layers. Metal layers 1,2,3,4, and 5 are represented, respectively, by the colors light blue, yellow, purple, green, and dark blue.

(b) Handle local connections along the array edge. Many metapixels have local connections between them within the array. These local connections by default will be hanging along the edge of the array. However, depending on the application, it may be necessary to tie the local connections to Vdd or Gnd.

2. The abutment box for this cell is set by placing a port called 'Abut' on the Icon/Outline layer (similar to PixelFootPrint). The abutment box for the Top and Bottom edge cells must have the same width as the PixelFootPrint. The abutment box for the Left and Right edge cells must have the same height as the PixelFootPrint. **THIS IS VERY IMPORTANT!**

3. All signals exiting the edge cells on the outer edge (except power supplies) must have a port on the appropriate layer. The ports have the following restrictions:

   (a) The port must be along (but within) the outer edge of the cell.

   (b) The port width (or height, depending on the edge) must match the pitch of the wire exiting the cell.

   (c) The port name must match the name used within the PixelFootPrint (see Section 1.2.2).

   (d) The port must be on the top level of the cell (i.e. it cannot be just contained within an instance).

NOTE: The **inner edge** of each edge cell refers to the edge abutting the metapixel array. The **outer edge** is the opposite side of the abutment box.

Horizontal Edge Cells: PixelFootLeft and -Right

The horizontal edge cells share the following properties:

1. Abutment box height must match that of the PixelFootPrint. There is no restriction on the width.

2. All signals entering from the inner edge must exit the cell on **Metal 2** layer. **THIS IS NECESSARY**. Make sure the signal connects to the outer edge.

3. All signals exiting the cell must have a port on the **Metal 2** layer.

4. Digital Vdd and Gnd supplies should be connected to the outer edge on Metals 5 and 4, respectively.

The PixelFootLeft cell is placed at the left edge of every row in the metapixel array. Signals exiting the left edge of the metapixel array—and that need to be handled within this cell—are:

- Receiver signals ~RSelY and ~PixAck
- Global bias signals

- Digital Vdd and Gnd supplies

See Figure 1.8 for sample automatic wiring. The PixelFootRight cell is placed at the right edge of every row. Signals that need to traverse the right edge cell are:

- Transmitter signals ~TReqY and TSelY
- Global bias signals
- Digital Vdd and Gnd supplies

Vertical Edge Cells: PixelFootTop and -Bottom

The vertical edge cells share the following properties:

1. Abutment box width must match that of the PixelFootPrint. There is no restriction on the height.
2. All signals entering from the inner edge must exit the cell on **Metal 1** layer. **THIS IS NECESSARY**. Make sure the signal connects to the outer edge.
3. All signals exiting the cell must have a port on the **Metal 1** layer.
4. Analog Vdd and Gnd supplies should be connected to the outer edge on Metals 5 and 4, respectively.

The PixelFootTop cell is placed at the top edge of every column. Signals within the edge cell are:

- Receiver signal ~RSelX
- Global bias signals
- Analog Vdd and Gnd supplies

See Figure 1.9 for sample automatic wiring. The PixelFootBottom cell is placed at the bottom edge of every column. Signals within:

- Transmitter signal ~TReqX
- Global bias signals
- Analog Vdd and Gnd supplies

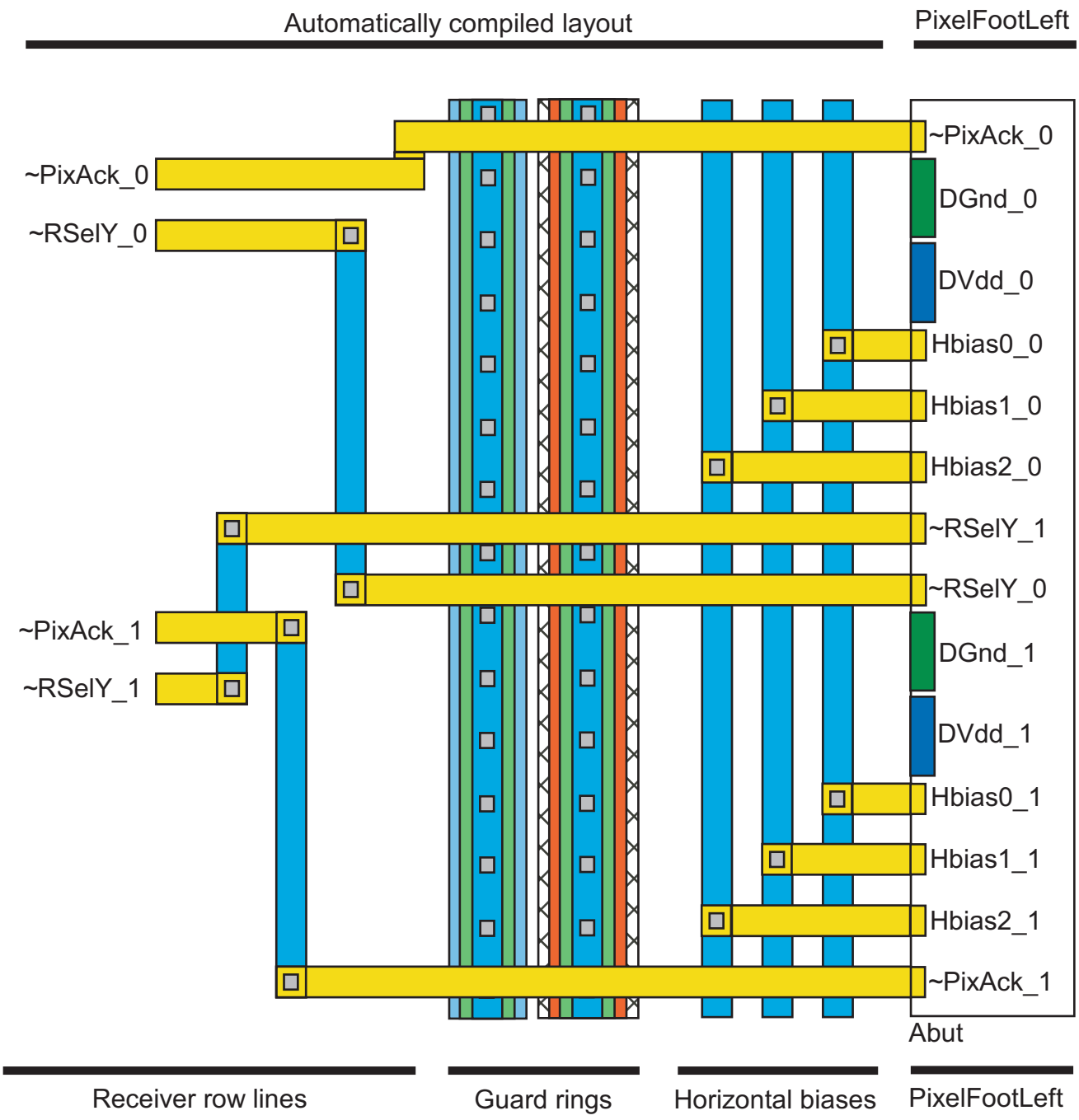

Figure 1.8: Sample PixelFootLeft Connectivity

PixelFootLeft contains ports for digital power, global biases running horizontally across the metapixel array, and row handshaking lines for the receiver. Global bias ports and receiver lines should be separated by at least the minimum width of a via and the surrounding metal, and should be placed on Metal2 (yellow). The entire left edge of PixelFootLeft connects to DGnd on Metal4 (green) and DVdd on Metal5 (dark blue).

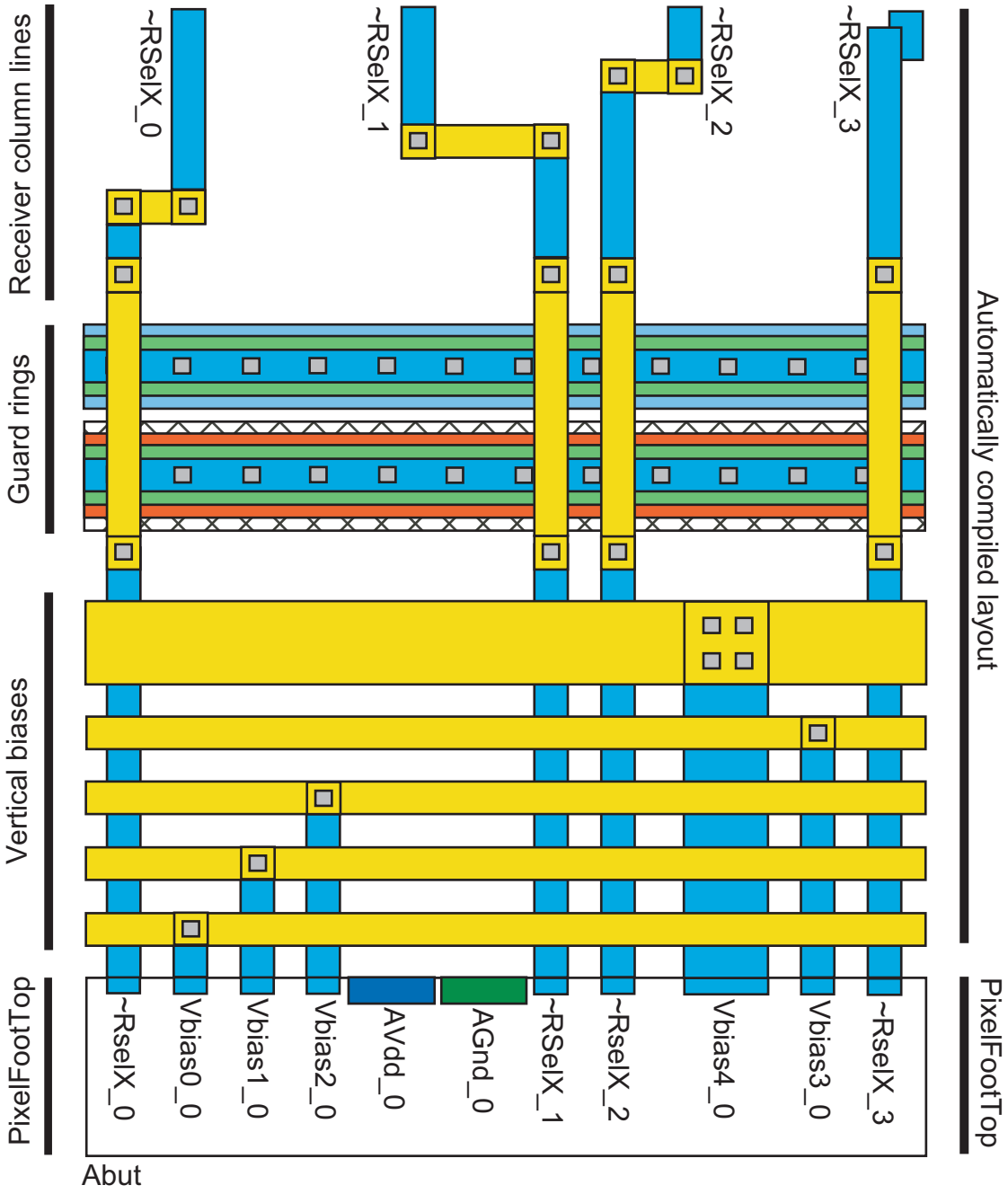

Figure 1.9: Sample PixelFootTop Connectivity

PixelFootTop contains ports for analog power, global biases running vertically across the metapixel array, and column select lines for the receiver. Global bias ports and receiver lines should be separated by at least the minimum width of a via and the surrounding metal, and should be placed on Metal1 (light blue). The entire top edge of PixelFootTop connects to AGnd on Metal4 (green) and AVdd on Metal5 (dark blue). Bias width is maintained in the routing, as seen in Vbias4_0.

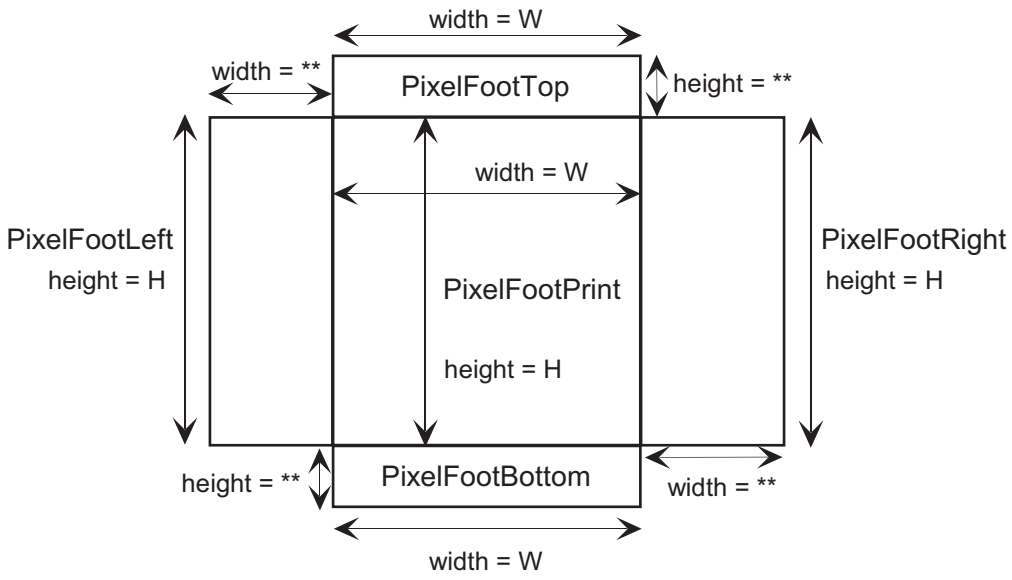

Figure 1.10: PixelFootAll

PixelFootPrint with edge cells within the PixelFootAll cell. The inner edge of each edge cell abut the abutment box of PixelFootPrint (e.g. the right edge of PixelFootLeft abuts the left edge of PixelFootPrint). The '**' denotes there is no size restriction.

1.2.4 DRC Requirements

It is recommended that the following items all pass DRC (Design Rule Check):

1. PixelFootAll—This cell is not necessary for layout compilation, however it is useful for checking the correctness of the layout. It is constructed by placing the edge cells (PixelFootLeft, -Right, -Top, and -Bottom) around PixelFootPrint. The inner edge of the abutment boxes for all the edge cells should be lined up with the abutment box of PixelFootPrint (Figure 1.10). This is also a good way to check the size restrictions in the edge cells (see Section 1.2.3). The PixelFootPrint should not cause any DRC errors with the edge cells; DRC on PixelFootAll will check this.

2. 3x3 Array of PixelFootPrint—This is to ensure that no DRC errors appear when you array the metapixel. Edge cells (PixelFootLeft, -Right, -Top, and -Bottom) should be placed along the edge of the array.

Chapter 2

ChipGen User Interface

2.1 ChipGen GUI

Use ChipGen's Graphical User Interface (GUI) to specify parameters for layout and netlist generation. Below is a description of the buttons and options that are available through the GUI (see Figure 2.1 for a picture of the GUI itself).

1. **General Chip Info**

   This section is where you input general chip parameters:

   - Rows—number of rows in the metapixel array.
   - Columns—number of columns in the metapixel array.
   - Directory—the full path of the directory where all the required files are located (the ChipGen DLL, AER cell library, metapixel library, etc.).
   - Library—the name of the AER layout or spice library.
   - Pixel—the name of the file that contains the metapixel cell layout or netlist.
   - Output—the name of the file to save the generated layout or netlist.
   - Biases on Top—set this option if you wish to route all the biases to the top of the chip. Note that this will eliminate the bottom pad frame.
   - Padframe—set this option if you wish to generate a padframe along with the core.
   - Short Substrate—set this option to short the substrate to GND in the spice netlists. This option is normally set, however, clear it if you wish to LVS the layout to check for unconnected GND nets.
   - Guard Pads—set this option to include Pads for the Guard Ring. Two pairs of Vdd and Gnd pads will placed on the top and bottom frame. If this option is clear then Guard Vdd/Gnd is connected to Digital Vdd/Gnd.

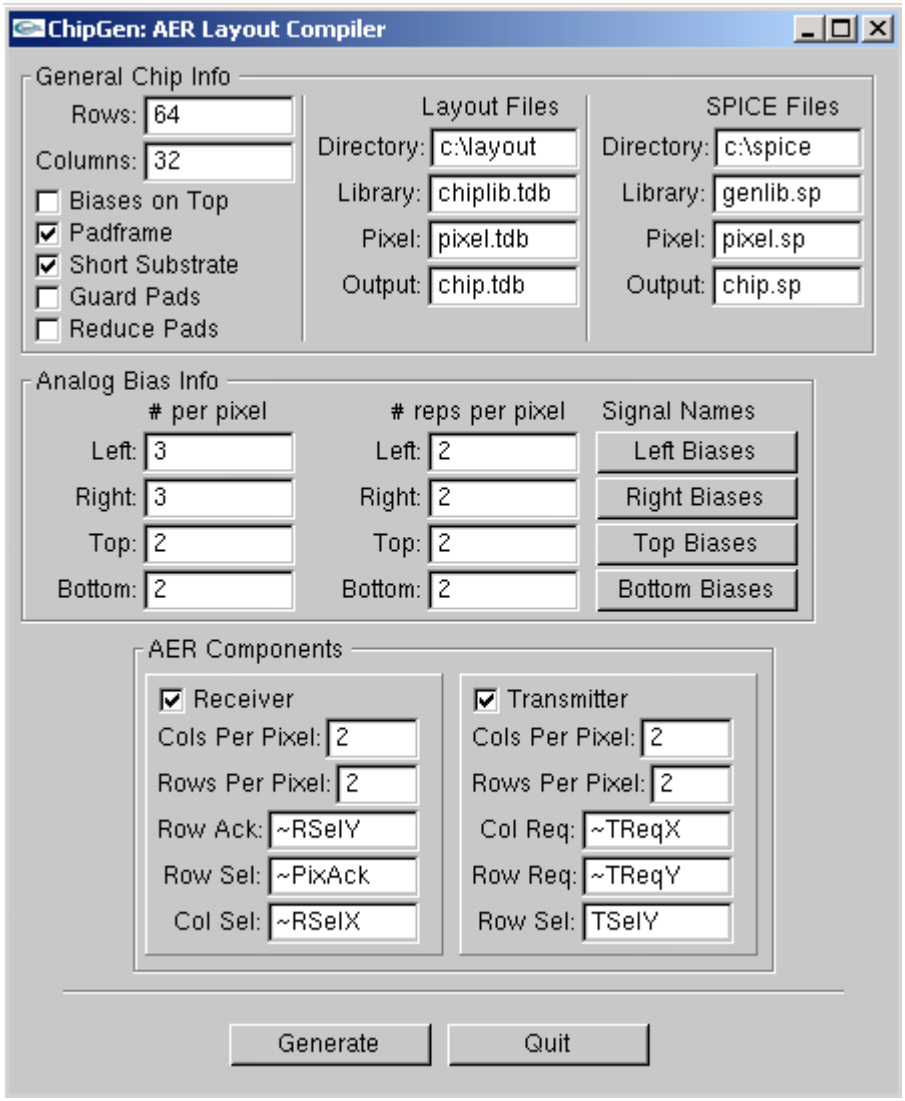

Figure 2.1: ChipGen's GUI

Enter parameters for layout and netlist generation through the GUI.

- Reduce Pads—set this option to reduce the total pad count of the chip. This will roughly eliminate 4 pads on each side of the chip by eliminating a pair of Analog Vdd/Gnd and Guard Vdd/Gnd from the top and bottom pad frames, a pair of Digital Vdd/Gnd from the left and right pad frames, and eliminating the Vdd or Gnd pad between Reset and Acknowledge.

2. **Analog Bias Info**

   This section is where you input info regarding the biases. Each row (Left,Right,Top,Bottom) corresponds to the respective side of the metapixel array. The following is a description of the columns:

   - # per pixel—number of distinct biases per metapixel.
   - # reps per pixel—number of times each distinct bias is repeated in one PixelFootPrint. For example, if you have seven global row biases and you use five of them twice in a PixelFootPrint, and the other two only once, you need to set this parameter to 2 and, in the layout, make dummy ports in PixelFootLeft for the other two biases that are not repeated so that they all appear twice. On the other hand, if you use one bias three times and the remaining six once, you should just wire the three repetitions together in the edge pixel, and set this parameter to 1.
   - Signal Names—press this button to open a window to enter the names of the biases.

3. **AER Components**

   This section is where you specify the AER components to include in the chip layout. You can select combinations of Receiver and Transmitter.

   - Receiver
     - Cols Per Pixel—number of columns per metapixel.
     - Rows Per Pixel—number of rows per metapixel.
     - Col Select—column select signal.
     - Row Select—row select signal.
     - Row Ack—row acknowledge.
   - Transmitter
     - Cols Per Pixel—number of columns per metapixel.
     - Rows Per Pixel—number of rows per metapixel.
     - Col Request—column request signal.
     - Row Request—row request signal.
     - Row Select—row select signal.

4. Command Buttons

   - Generate—creates the chip.txt file and spice netlist files
   - Quit—exits GUI

2.2 Parameter File

Parameters for automatic compilation are listed in a file named "chip.txt", which is automatically generated by the ChipGen GUI. The parser is case insensitive and ignores whitespace, however the headings must be spelled exactly as shown. Comments can be delineated with a % symbol. Note: if a heading is missing, a default value will be used. Below is a sample chip.txt. In the following example, left and right biases and top and bottom biases share the same name. Since the "WIRE_BIASES_UP" option is not checked, distinct pads for each bias will be generated on the top and bottom of the chip. If the option were checked, though, only one pad for each bias would be generated since they share the same name.

```
% This is general chip info
DIRECTORY           c:\chipgen\layout
AER_FILE            chiplib.tdb
PIX_FILE            pixel.tdb
OUT_FILE            chip.tdb
SP_DIR              c:\chipgen\spice
SP_LIB_FILE         genlib.sp
SP_PIX_FILE         pixel.sp
SP_OUT_FILE         chip.sp
CHIP_PIXEL_ROWS     64
CHIP_PIXEL_COLS     32
PADFRAME            YES
WIRE_BIASES_UP      NO
SUBSTRATE           YES
REDUCE_PADS         NO
GUARD_PADS          NO

% AER receiver info
RECEIVER            YES
RCVR_ROWS_PER_PIX   2
RCVR_COLS_PER_PIX   2
RCVR_ROW_SIGS
     ~PixAck
     ~RSelY
RCVR_COL_SIGS
     ~RSelX

% AER transmitter info
TRANSMITTER         YES
XMIT_ROWS_PER_PIX   2
XMIT_COLS_PER_PIX   2
XMIT_ROW_SIGS
     TSelY
     ~TReqY
XMIT_COL_SIGS
     ~TReqX

% Left analog bias info
LEFT_BIAS_NUM       3
LEFT_BIAS_REPS      2
LEFT_BIAS_NAMES
     Hbias0
     Hbias1
     Hbias2

% Right analog bias info
RIGHT_BIAS_NUM      3
RIGHT_BIAS_REPS     2
RIGHT_BIAS_NAMES
     Hbias0
     Hbias1
     Hbias2

% Top analog bias info
TOP_BIAS_NUM        2
TOP_BIAS_REPS       2
TOP_BIAS_NAMES
     Vbias0
     Vbias1

% Bottom analog bias info
BOTTOM_BIAS_NUM     2
BOTTOM_BIAS_REPS    2
BOTTOM_BIAS_NAMES
     Vbias0
     Vbias1

END
```

Chapter 3

Pad-Frame Layout

When the 'Padframe' option is checked, ChipGen generates layout with pads in a cell named 'chip' (Figure 3.1). The top and bottom pads (FrameTop and FrameBottom) are reserved for analog power and biases. The left pads (FrameLeft) are reserved for digital power and receiver signals. The right pads (FrameRight) are reserved for digital power and transmitter signals. Below is a detailed description of each element of the padframe:

FrameTop

1. Generated if there are right and top biases; not generated otherwise.
2. The 'Guard Pads' option places two pairs of Guard Vdd/Gnd pads (see Figure 3.1) on the left and right.
3. The 'Reduce Pads' option removes a pair of Analog Vdd/Gnd pads and a pair of Guard Vdd/Gnd pads (see Figure 3.1).

FrameBottom

1. Generated if there are left and bottom biases; not generated otherwise.
2. Excluded if 'Biases on Top' option is checked (any left and bottom biases are routed to 'FrameTop').
3. The 'Guard Pads' option places two pairs of Guard Vdd/Gnd pads (see Figure 3.1) on the left and right.
4. The 'Reduce Pads' option removes a pair of Analog Vdd/Gnd pads and a pair of Guard Vdd/Gnd pads (see Figure 3.1).

FrameLeft

1. Generated if there is a receiver.
2. Power and ground (RcvPVdd/Gnd) pads are interleaved between the address and control signals.
3. Address bits are named RcvAdr0' to RcvAdr$\{N-1\}$ where N is the maximum address bits of the receiver.
4. The control signals are labeled (from bottom to top):
   - RcvRst—reset
   - RcvAck—acknowledge
   - RcvReqY—active-high row-request
   - ∼RcvReqX—active-low column-request
5. The 'Reduce Pads' option removes a pair of Digital Vdd/Gnd pads and a pair of RcvPVdd/Gnd pads (see Figure 3.1).
6. The 'Biases on Top' option places a pair of Analog Vdd/Gnd pads in the bottom corner.

FrameRight

1. Generated if there is a transmitter.
2. Power and ground (XmtPVdd/Gnd) pads are interleaved between the address and control signals.
3. Address bits are named XmtAdr0 to XmtAdr$\{N-1\}$ where N is the maximum address bits of the transmitter.
4. The control signals are labeled (from bottom to top):
   - XmtRst—reset
   - XmtAck—acknowledge
   - XmtReqY—active-high row-request
   - ∼XmtReqX—active-low column-request
5. The 'Reduce Pads' option removes a pair of Digital Vdd/Gnd pads and a pair of XmtPVdd/Gnd pads (see Figure 3.1).
6. The 'Biases on Top' option places a pair of Analog Vdd/Gnd pads in the top corner.

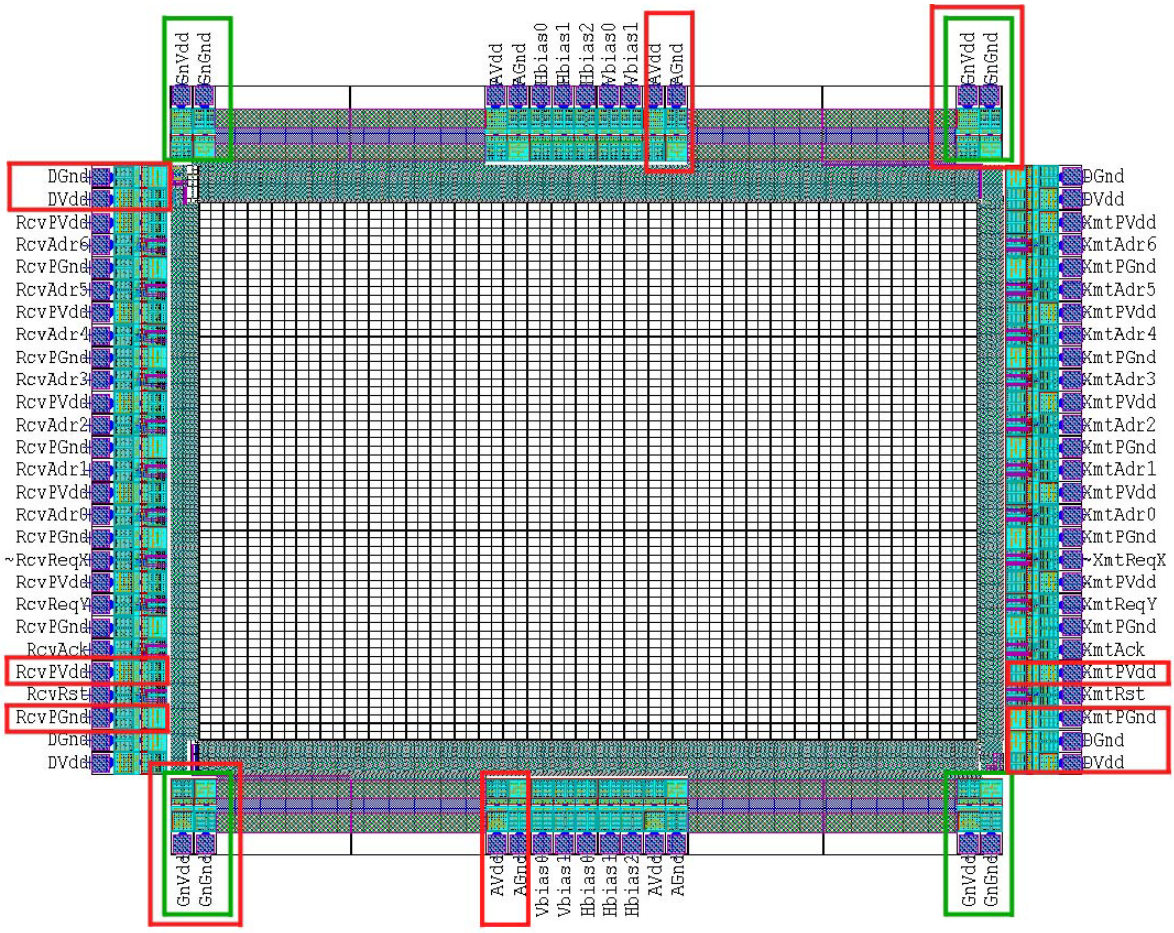

Figure 3.1: Example chip layout with pads.

A 64x64 array of PixelFootPrint from example_pixel.tdb; layout was generated with the 'Padframe' and 'Guard Pads' options set. Pads with a red outline are removed when the 'Reduce Pads' option is checked; pads with a green outline are removed when the 'Guard Pads' option is unchecked. All ground pads are labeled with 'DGnd' when the 'Short Substrate' option is checked.

Chapter 4

Simulation

In this chapter we describe how to simulate your PixelFootPrint and the Core or Chip that ChipGen generates. We will start with PixelFootPrint.

4.1 Simulating PixelFootPrint

It is important to simulate PixelFootPrint so that you can confirm that your pixel's event-generation circuitry generates a request signal (Figure 4.1) with rise and fall times of less than 5 nanoseconds (0.25μm technology). Typically, some form of positive feedback is required to achieve this.[1]

An S-Edit schematic file containing the PixelFootPrint testbench is in the file 'pixeltest.sdb' (Figure 4.2). This testbench will supply synaptic input to your neuron circuit and readout its spikes. It includes the following modules:

- PixelRcvEnv—simulates receiver environment, appropriately following the handshaking protocol (Figure 4.1).
- PixelRcvIntfc—interfaces neuron with receiver (see Figure 1.4).
- TestNeuron—receives synaptic input and generates spikes (see Figure 4.3).
- PixelXmtIntfc—interfaces neuron with transmitter (see Figure 1.4).
- PixelXmtEnv—simulates transmitter environment, appropriately following the handshaking protocol (Figure 4.1).

The four interfaces between these five modules use the following signals:

[1] E Culurciello, R Etienne-Cummings, and K Boahen, A Biomorphic Digital Image Sensor, *IEEE Journal of Solid State Circuits*, vol 38, no 2, pp 281-294, 2003.

PixelRcvEnv to PixelRcvIntfc

- `nrselx`—active-low column-select (i.e., request)
- `nrsely`—active-low row-select (i.e., request)
- `npixack`—active-low pixel-acknowledge

PixelRcvIntfc to TestNeuron

- `synin`—active-high synaptic event signal

TestNeuron to PixelXmtIntfc

- `nreq`—active-low pixel-request
- `npixrst`—active-low pixel-reset (i.e., acknowledge)

PixelXmtIntfc to PixelXmtEnv

- `ntreqx`—active-low column-request
- `ntreqy`—active-low row-request
- `tsely`—active-high row-select (i.e., acknowledge)

In addition, `vpl` controls how frequently PixelRcvEnv sends synaptic events to PixelRcvIntfc, and `nRST` (active-low) resets PixelXmtEnv at the beginning of the simulation.

To illustrate the correct sequencing of these signals, we present results from simulating the testbench with an example neuron circuit (Figure 4.3).[2] This conductance-based design includes an excitatory synapse, a refractory channel, a soma, and an axon-hillock. The synapse includes a pulse-extender that extends the nanosecond-wide pulse supplied by PixelRcvIntfc. The axon-hillock uses positive-feedback to generate a sufficiently fast `nreq` signal (PixelXmtIntfc requires rise and fall times under 5ns).

After the soma integrates a couple of synaptic events, the axon-hillock starts generating spikes (Figure 4.4a). These synaptic events are delivered by a three-nanosecond handshake involving `nsely`, `nselx` (shorted to `nsely`), and `npixack` (Figure 4.4b). The spikes are read out by a ten-nanosecond handshake involving `ntreqy`, `ntreqx`, and `tsely` (Figure 4.4c).

[2] JV Arthur and K Boahen, Learning in Silicon: Timing is Everything, *Advances in Neural Information Processing Systems 18*, 2006.

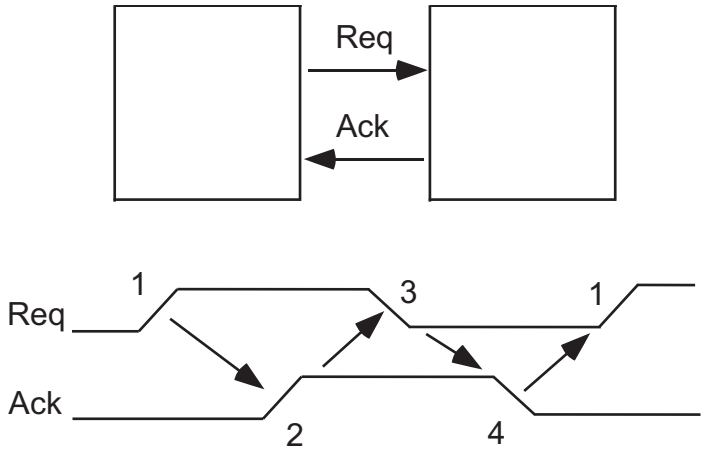

Figure 4.1: Four-phase Handshake Protocol

The left block **requests** by asserting **Req** (1) and the right block **acknowledges** by asserting **Ack** (2). The left block signals that it received the acknowledge by de-asserting **Req** (3) and the right block terminates the communication by de-asserting **Ack** (4).

4.2 Simulating the Core

Use 'testbench.sp' to ensure your ChipGen-generated neuron array functions correctly with the word-serial AER circuits. This testbench (Figure 4.5) instances the following subcircuits:

- ReceiverEnv—simulates the receiver environment by repeatedly sending the receiver a burst consisting of a row address followed by two column addresses.
- TransmitterEnv—simulates the transmitter environment by acknowledging the transmitter approriately.

The signals used in the testbench are:

ReceiverEnv

- RcvAdr{0,1}—receiver address-bits
- RcvReqY—active-high row-request
- ∼RcvReqX—active-low column-request
- RcvAck—active-high acknowledge

TransmitterEnv

- XmtAdr{0,1}—transmitter address-bits
- XmtReqY—active-high row-request
- ∼XmtReqX—active-low column-request
- XmtAck—active-high acknowledge

In addition, RcvRst and XmtRst reset the receiver and transmitter at the beginning of the simulation.

To illustrate the correct sequencing of these signals, we present results from simulating the testbench using an example neuron array (Figure 4.6).[3] The example neuron array simply consists of buffers so the transmitter output follows the receiver input.

You can simulate the chip without pads by extracting the 'core' layout cell (extract 'chip' to simulate with pads). Include the extracted netlist into the testbench using the '.include' statement. The signal names for 'core' and 'chip' are the same.

[3]By convention, ~XmtReqX should go low after the data becomes valid, however a delay was not necessary here because the receiver adds delay between the request signal and its address latch.

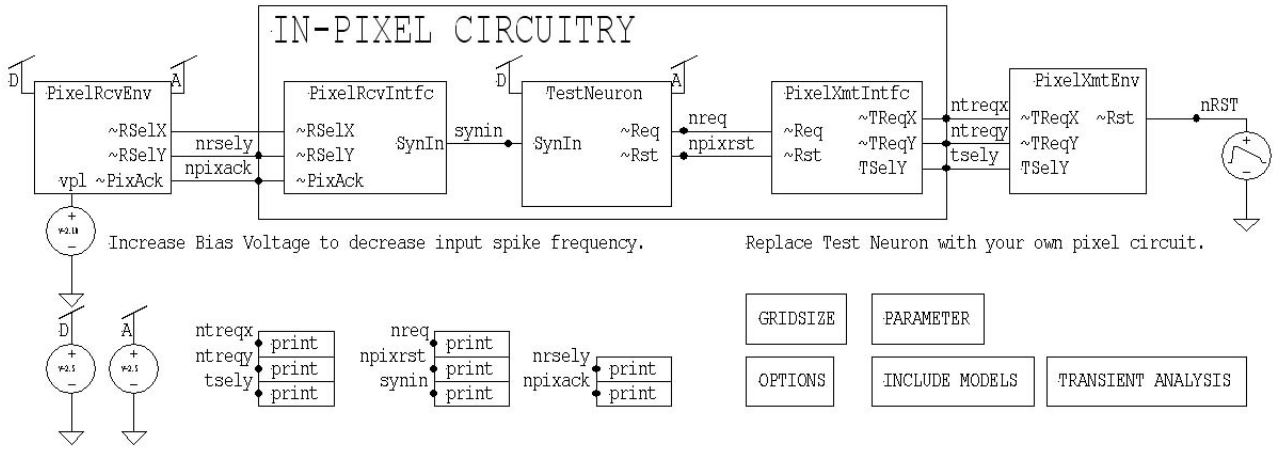

Figure 4.2: PixelFootPrint TestBench Schematic

Use this testbench to simulate your neuron circuit by replacing `TestNeuron` with your own module. `PixelRcvEnv` will deliver spikes to `TestNeuron` and `PixelXmtEnv` will readout spikes from `TestNeuron`. You can increase the input spike rate by lowering the bias voltage `vpl` applied to `PixelRcvEnv`.

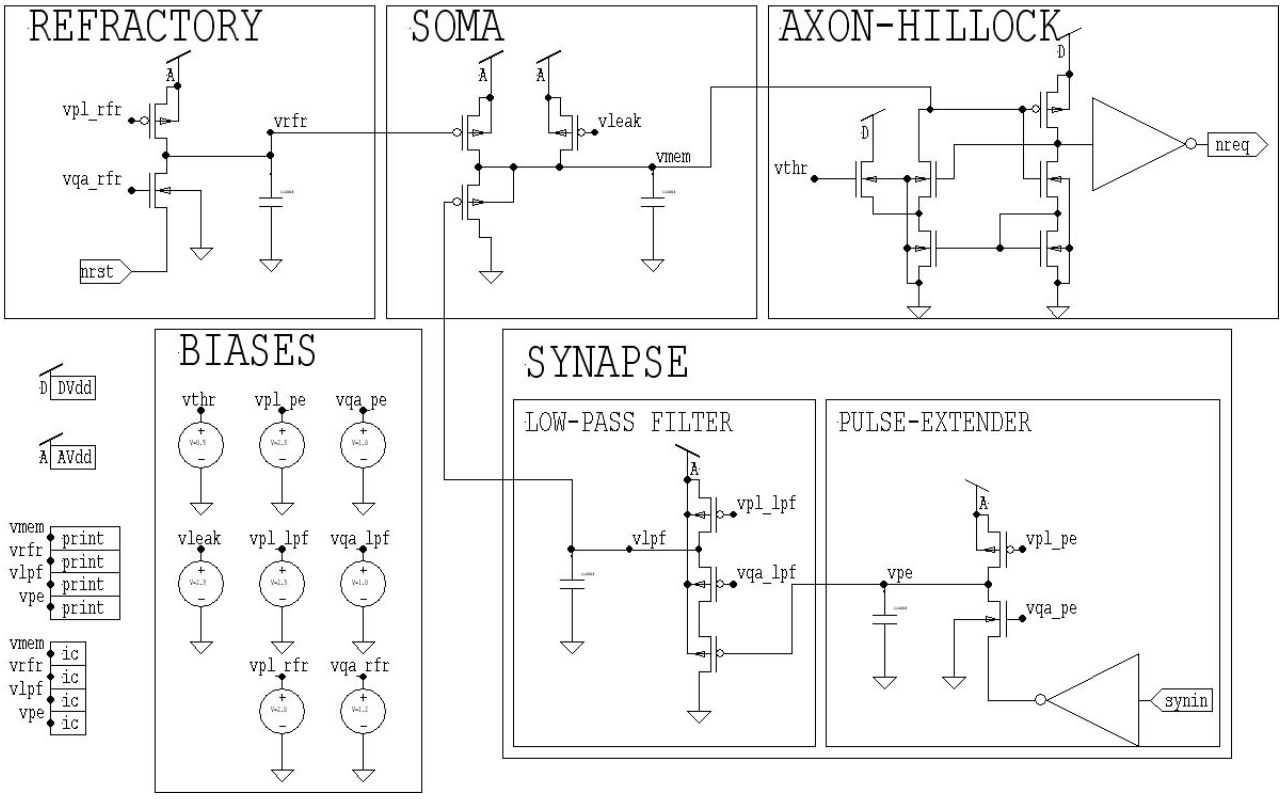

Figure 4.3: Example TestNeuron Schematic

An example conductance-based neuron with an excitatory synapse, a refractory channel, a soma, and an axon-hillock. Each successive synaptic event at `synin` leaks charge off the pulse-extender's node `vpe`. This in turn leaks charge off `vlpf` and hence `vmem`. Once `vmem` passes a threshold, positive-feedback through the axon-hillock's current mirror kicks in to give `nreq` a fast slew rate. This request is acknowledged by `nrst`, which leaks charge off `vrfr`, making the neuron refractory.

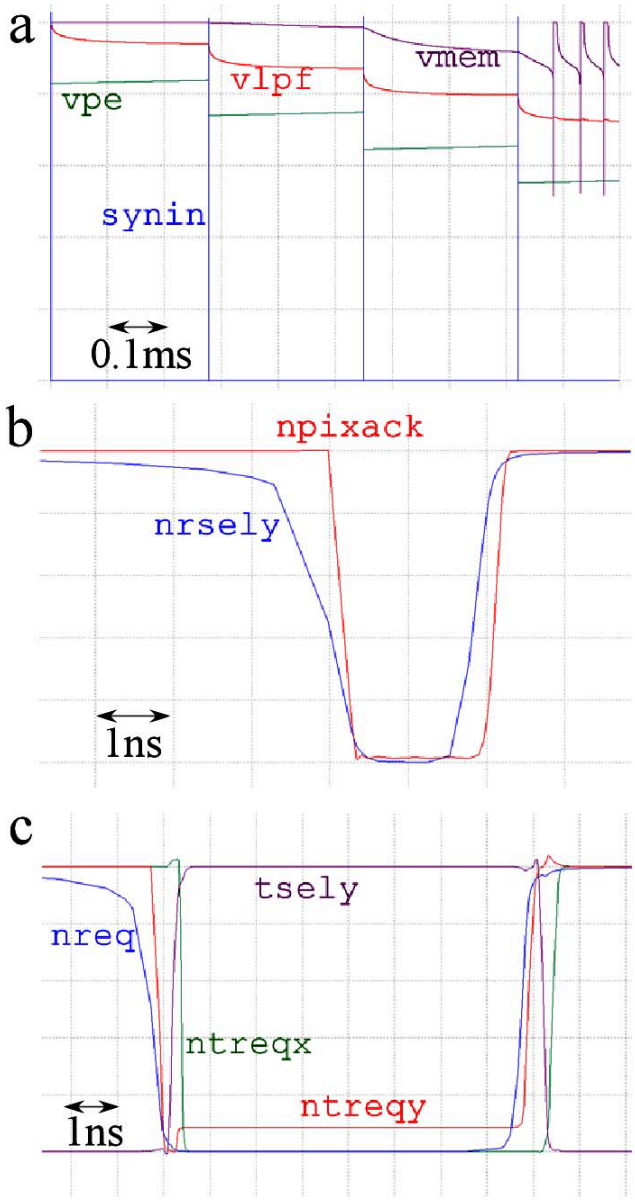

Figure 4.4: PixelFootPrint TestBench Waveforms

W-Edit plots of 'pixeltest.sp' simulation. (a) Each successive `synin` leaks charge off `vpe`, which leaks charge off `vlpf` and ultimately `vmem`. Once `vmem` passes a threshold, positive-feedback kicks in to generate spikes. (b) Following the four-phase handshake protocol (see Figure 4.1), `PixelRcvEnv` selects `PixelRcvIntfc` by pulling `nrsely` and `nrselx` (not shown) low, which prompts `PixelRcvIntfc` to acknowledge with `npixack`; it also pulls `synin` high (see (a)). (c) When `PixelXmtIntfc` sees `nreq` go low (due to spikes shown in (a)), it makes a request to `PixelXmtEnv` by pulling `ntreqy` low. `PixelXmtEnv` acknowledges by taking `tsely` high. This prompts `PixelXmtIntfc` to pull `ntreqx` low; it asserts `npixrst` (not shown) concurrently to clear `nreq` by terminating the spike. This prompts `ntreqy` to go high, which is acknowledged by `tsely` going low, and that prompts `ntreqx` to go high.

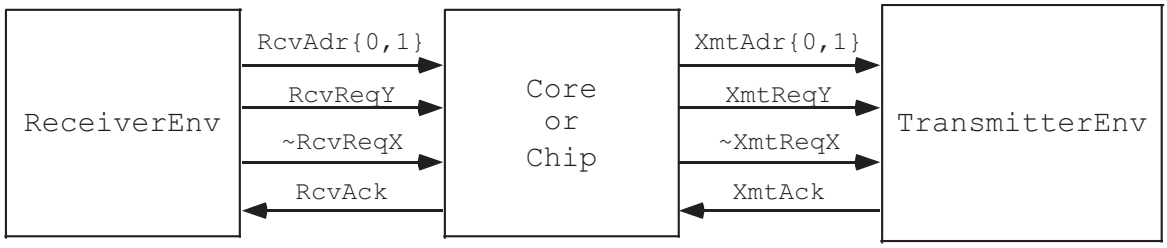

Figure 4.5: Core or Chip Testbench Schematic

`ReceiverEnv` sends address-events to `Core or Chip` via four-phase handshaking, and `Core or Chip` sends address-events to `TransmitterEnv` via four-phase handshaking.

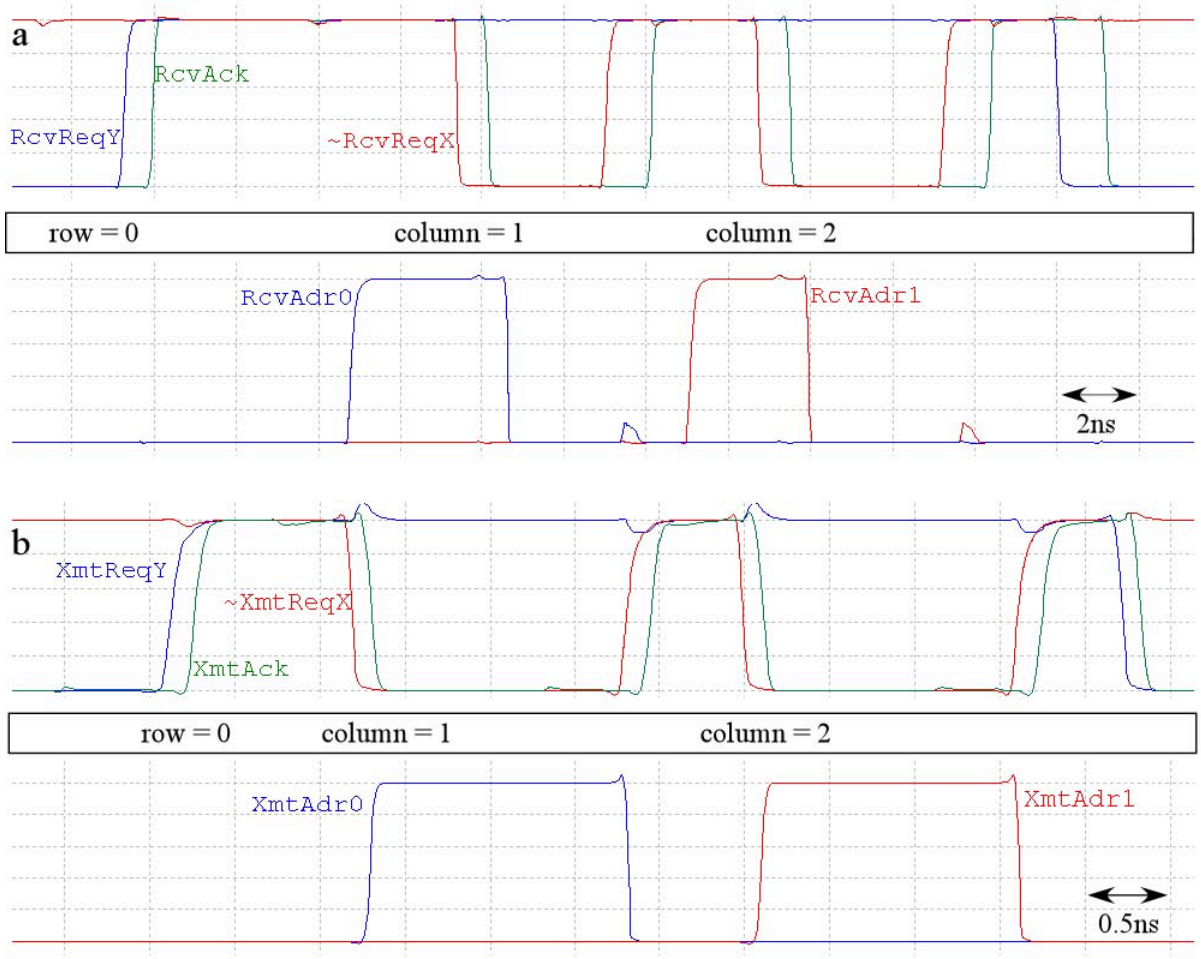

Figure 4.6: Core TestBench Waveforms

W-Edit plots of 'testbench.sp' simulation. (a) `ReceiverEnv` sends the row address (row = 0) by driving `RcvReqY` high, which prompts the receiver to acknowledge by taking `RcvAck` high, a two-phase handshake. `ReceiverEnv` then sends the column address (column = 1) by driving ~`RcvReqX` low, and the receiver acknowledges by driving `RcvAck` low; subsequent transitions on these signals complete the four-phase handshake. The second column address (column = 2) is sent by another four-phase handshake. The burst terminates when `ReceiverEnv` drives `RcvReqY` low, prompting `RcvAck` to go low. (b) The `XmtReqY`, ~`XmtReqX`, and `XmtAck` signals between the transmitter and `TransmitterEnv` follow the same handshaking sequence as in (a).

Appendix A

Sample Netlist Files

Below are samples of netlist files needed for Netlist Generation (see Section 1.1.2). This example is for a metapixel with a 2x3 subarray of neurons, each with one input synapse. The netlists do not represent any usable circuit. They are designed to illustrate various points.

A.1 layout.sp

This file defines the signals entering/exiting the pixel. The sample file is presented first (in small font) and a description of the circuit follows. NOTE: The first line of the file should not contain the **.layout** statement.

```
*
* layout.sp
*

.layout
LEFT={TSelY_0,~TReqY_0,~RSelY_0,~PixAck_0,Vbr_0,OutL_0,TSelY_0,~TReqY_1,~RSelY_1,~PixAck_1,Vbr_1,OutL_1}
RIGHT={TSelY_0,~TReqY_0,~RSelY_0,~PixAck_0,Vbr_0,OutR_0,TSelY_0,~TReqY_1,~RSelY_1,~PixAck_1,Vbr_1,OutR_1}
TOP={~RSelX_0,~TReqX_0,Vinc_0,OutT_0,~RSelX_1,~TReqX_1,Vinc_1,OutT_1,~RSelX_2,~TReqX_2,Vinc_2,OutT_2}
BOTTOM={~RSelX_0,~TReqX_0,Vinc_0,OutB_0,~RSelX_1,~TReqX_1,Vinc_1,OutB_1,~RSelX_2,~TReqX_2,Vinc_2,OutB_2}
GLOBAL={Vbr,Vinc,AGnd,AVdd,DGnd,DVdd}
LEFTEDGE={TSelY_0,~TReqY_0,~RSelY_0,~PixAck_0,Vbr_0,Vdd,TSelY_0,~TReqY_1,~RSelY_1,~PixAck_1,Vbr_1,Vdd}
RIGHTEDGE={TSelY_0,~TReqY_0,~RSelY_0,~PixAck_0,Vbr_0,Gnd,TSelY_0,~TReqY_1,~RSelY_1,~PixAck_1,Vbr_1,Gnd}
END

** end layout.sp **
```

Remarks:

- Outside the receiver and transmitter signals, the only other signals exiting the metapixel horizontally are **Vbr**, **OutL**, and **OutR** (remember, the indices are used to differentiate between the various neurons within the metapixel). **Vbr** is defined as GLOBAL, and thus has the same node name on left and right sides of the metapixel. **OutL** and **OutR** are local connections between metapixels. Vertically, the only other signals are the global signal **Vinc** and local signals **OutT** and **OutB**.
- The local signal pairs (**OutL**/**OutR** and **OutT**/**OutB**) are located in the same position within each list, as is required to correctly connect them.
- **OutL** is connected to **Gnd** along the left side of the array (as defined by LEFTEDGE) and **OutR** is connected to **Vdd** along the right side of the array (as defined by RIGHTEDGE).
- Since TOPEDGE and BOTTOMEDGE statements do not exist, the local signals **OutT** and **OutB** are left hanging by default.
- All global biases and power signals need to be defined in GLOBAL.

A.2 pixel.sp

This file is the main file for the pixel.

```
*
* pixel.sp
*

Xsp0_0 OutL_0 OutR00 OutT_0 OutB00 TSelY_0 ~TReqY_0 ~RSelY_0 ~PixAck_0 ~RSelX_0 ~TReqX_0 Vinc_0 Vbr_0 DGnd AGnd DVdd AVdd subpixel
Xsp0_1 OutR00 OutR01 OutT_1 OutB01 TSelY_0 ~TReqY_0 ~RSelY_0 ~PixAck_0 ~RSelX_1 ~TReqX_1 Vinc_1 Vbr_0 DGnd AGnd DVdd AVdd subpixel
Xsp0_2 OutR01 OutR_0 OutT_2 OutB02 TSelY_0 ~TReqY_0 ~RSelY_0 ~PixAck_0 ~RSelX_2 ~TReqX_2 Vinc_2 Vbr_0 DGnd AGnd DVdd AVdd subpixel

Xsp1_0 OutL_1 OutR10 OutB00 OutB_0 TSelY_1 ~TReqY_1 ~RSelY_1 ~PixAck_1 ~RSelX_0 ~TReqX_0 Vinc_0 Vbr_1 DGnd AGnd DVdd AVdd subpixel
Xsp1_1 OutR10 OutR11 OutB01 OutB_1 TSelY_1 ~TReqY_1 ~RSelY_1 ~PixAck_1 ~RSelX_1 ~TReqX_1 Vinc_1 Vbr_1 DGnd AGnd DVdd AVdd subpixel
Xsp1_2 OutR11 OutR_0 OutB02 OutB_2 TSelY_1 ~TReqY_1 ~RSelY_1 ~PixAck_1 ~RSelX_2 ~TReqX_2 Vinc_2 Vbr_1 DGnd AGnd DVdd AVdd subpixel

** end pixel.sp **
```

Remarks:

- All the signals found within **layout.sp** must have the same name as in **pixel.sp**.
- This file only contains instances of subcircuits or individual components (e.g. transistors). There should be no subcircuit definitions within this circuit.
- Local connections within this level of the netlist may exist (e.g. OutR00) that need not be defined in **layout.sp**.
- Note how the index numbering corresponds to the position of the neuron within the 2x3 subarray in the metapixel. This need not be the case but is STRONGLY recommended.

A.3 subckts.sp

This file defines the subcircuits found within **pixel.sp**.

```
*
* subckts.sp
*

.SUBCKT subpixel OutL OutR OutR OutB TSelY ~TReqY ~RSelY ~PixAck ~RSelX ~TReqX Vinc Vbr DGnd AGnd DVdd AVdd

{******** component list or other subcircuit instances ***************}

.ENDS

{**************** other subcircuit definitions ******************}

** end subckts.sp **
```

The subcircuit contains components, such as transistors and capacitors, or other subcircuit instances. In the latter case, the other subcircuit definitions need be defined within this file as well.

Appendix B

Walkthrough

Let us go through a simple example of ChipGen in action.

1. Make sure you have all the required files as specified in Chapter 1. Example files that have been included:
   - chiplib.tdb—AER cell layout library. Note that without Tanner's Mixed-Signal Design Kit[1], most pads will appear as outlines.
   - example_pixel.tdb—An example metapixel layout.
   - genlib.sp—AER circuits netlists.
   - example_pixel.sp—An example metapixel spice netlist.
   - subckts.sp—The subcircuit netlist definitions for AER receiver and transmitter circuitry used in the example metapixel.
   - layout.sp—Layout definition for example metapixel.
2. Use chipgen_gui.exe to generate chip.txt and spice files. Make sure that the directory fields contain the correct path (e.g. if you are working from the example files, the layout directory field should be c:\{folder_you_extracted_to}\chipgen\layout)
3. Load and run "ChipGen.DLL" in L-Edit: Go to the menu bar and click on Tools → Macro → Load... → Run. Before you click the "Run" button, though, click on "Setup" to make sure that, in the UPI tab in the console that pops up, the "Update display" checkbox in the "While UPI code executes" section is clear.
4. After a few minutes, the layout will be created. Now you can DRC to detect any layout errors: Go to the menu bar and click on Tools → DRC.
5. Once DRC has completed without any errors, you can extract a netlist of the layout: Go to the menu bar and click on Tools → Extract... → Run. If you selected the "Short Substrate" option in the ChipGen GUI, use the file "mTSMd026.ext" as the extract definition file, otherwise

[1] http://www.tanner.com/EDA/products/designkit/tsmc.htm

use “mTSMd026_open”; this file is located in c:\{folder_you_extracted_to}\chipgen. Make sure you save the extracted netlist into the same folder as the generated spice files (e.g. c:\{folder_you_extracted_to}\chipgen\spice) so that Tanner’s LVS program can find all the required files.

6. Now you can LVS the extracted layout netlist with the generated netlist. In Tanner’s LVS program go to the menu bar and click on File → New... → LVS Setup. In the console that opens, the Input tab is used to specify the locations of the netlists; the generated chip netlist is named example_chip.sp. In the Options tab there is a checkbox named “Consider M bulk terminals and B,J,Q,Z substrate terminals” in the “Device Terminals” section; check this box if the “Short Substrate” option was selected in the ChipGen GUI, otherwise leave this clear.

Appendix C

Frequently Asked Questions

Why is the ChipGen GUI interface not displaying on my computer?

The GUI is written using the OpenGL Utility Toolkit (GLUT)[1], originally written by Mark Kilgard, ported to Win32 (Windows 95,98,Me,NT,2000,XP) by Nate Robins. Make sure you have the file "glut32.dll" in the same directory as the GUI executable or in the Windows system directory.

Why does ChipGen generate the same layout after I have changed the parameters?

You must reload the L-Edit macro for the changes to take effect.

Why does ChipGen open up multiple cell windows?

This is an option in L-Edit that must be turned off. Go to the menu bar and click on Tools → Macro... → Setup. In the UPI tab there will be a section called "While UPI code executes;" make sure the "Update display" option is unchecked.

Why do I get the message "Error while copying {your pixel name}.tdb?"

Make sure that your .tdb file with metapixel layout has the same setup information as the chiplib.tdb file that contains the AER cell library. To change setup information, in the menu bar go to File → Replace Setup..., then use chiplib.tdb file as the file that you want setup information to come from.

[1] http://www.xmission.com/~nate/glut.html

After loading ChipGen, some windows pop up then L-Edit becomes unresponsive—has the program crashed?

The ChipGen macro can take up to twenty minutes to run for large chips (e.g. 256x256 of the example_pixel) so you may have to wait a bit. However, if your chip is not large and L-Edit remains unresponsive for more than ten minutes, the macro may have encountered an error. If such is the case, try checking that the metapixel layout obeys the rules mentioned in Chapter 1, and that the parameters you set in the GUI match your layout.

Why do pads appear as just outlines?

Our lab uses the pads that are included in Tanner's TSMC Mixed-Signal Design Kit[2] so we cannot give away these pads due to license restrictions. However, we have included "PadOut," which has been modified to reduce switching noise.

I have Tanner's TSMC Mixed-Signal Design Kit—how do I get ChipGen to compile these pads?

The pad layouts that need to be copied over into the AER layout library (chiplib.tdb) are "PadVdd", "PadGnd", "PadInC", and "PadAref". Make sure both the AER layout library and the layout file that contains the pad layouts are open in L-Edit, and that the AER layout library is the top-level window. Go to the menu bar and click on Cell → Copy...; in the "File:" pulldown, select the layout file that contains the pad layouts. Select the name of the pad layout that you want to copy, and press the "OK" button; make sure that "Copy cell to current file" is selected in the "Reference type" section at the bottom of the window. A "Conflict Resolution" window will open; for each cell listed, in the pulldown menu under "Action," select "Overwrite old cell." Make sure you do not overwrite the existing "PadOut" in the AER layout library with Tanner's "PadOut".

Why do I receive errors when I LVS the chip (core and padframes) but no errors when I LVS the core alone?

The spice files that are generated contain netlists for the padframes so there will be element mismatches if the required pad layouts (see above question) are not present.

Why is my simulation failing even though my layout passed DRC and LVS?

Make sure your pixel's event-generation circuitry generates the ∼Req signal (see Figure 1.4) with rise and fall times of less than 5 nanoseconds ($0.25\mu m$ technology). Typically, some form of positive

[2] http://www.tanner.com/EDA/products/designkit/tsmc.htm

feedback is required to achieve this.[3]

[3]E Culurciello, R Etienne-Cummings, and K Boahen, A Biomorphic Digital Image Sensor, *IEEE Journal of Solid State Circuits*, vol 38, no 2, pp 281-294, 2003.