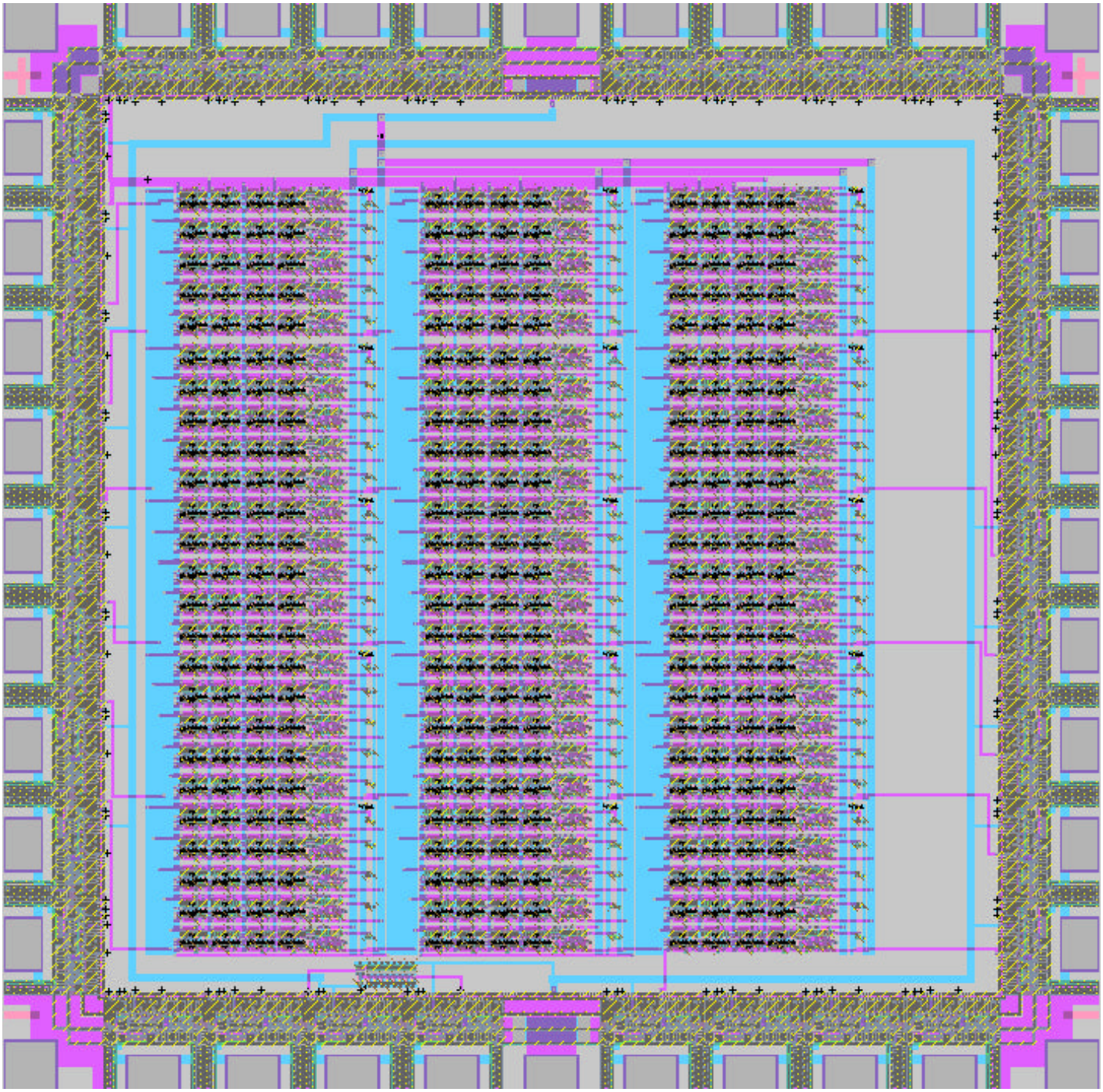
Final Report:
Design and Implementation of a Binary Neural Network



Daniel Smith & Shamik Maitra
E158 Intro to CMOS VLSI Design
Prof. Harris

Background:

In a typical neural network, a set of inputs is operated on by a series of nonlinear elements called neurons. This network of neurons produces a set of outputs based on these inputs. Neural networks have many practical applications, which mostly stem from the fact that the strength of a neural network is its ability to identify salient features from large data sets. Typical applications are data compression, facial recognition, trend analysis/prediction, etc.

The neurons are usually arranged in architectures that support the function that they are expected to perform. On a basic level each neuron's connection to another neuron or input carries with it some strength, or weight. Whenever an input arrives at a neuron, it is multiplied by this connection weight.

Typical neurons (in the computational sense) multiply each input by a connection weight and add up all of these weighted inputs. This sum is usually sent through an activation function to determine whether or not a neuron will fire (or to what degree).

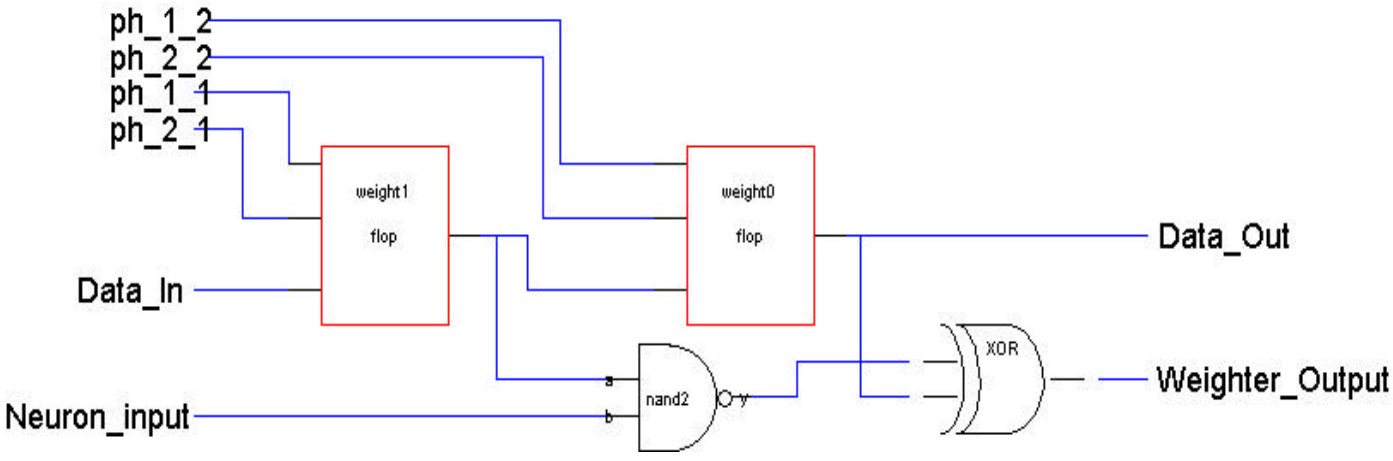
While such networks exist primarily as software and computer models, some have been hardwired into circuits as well as IC's. Several neural net chips exist on the market today. Some of these chips operate as analog devices by running below threshold on the transistors thereby gaining continuous properties instead of discrete properties afforded by CMOS transistor logic. Other chips, however, do use purely the digital capacity of CMOS to execute a network; and yet others use hybrids between the two to create a more robust design.

Functional Overview:

We accomplished the design and implementation of our neural network through a good degree of analogizing processes that take place in a neuron to processes that we felt we could replicate efficiently on a chip. We felt efficiency, in terms of space, was important because we wanted to fit as many neurons as possible onto this chip.

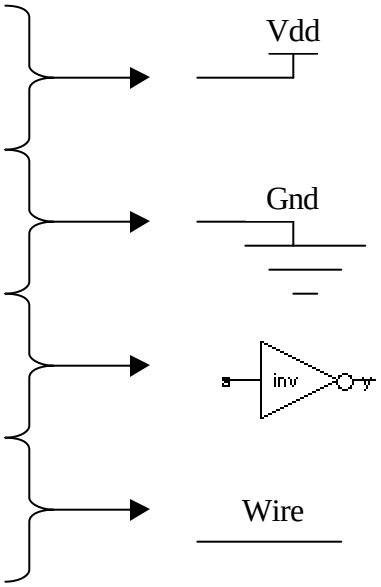
Since we wanted our neural network to be able to be used for different applications, we decided to fully connect each layer to the next layer. This means that each of the five inputs to the network is connected to each of the 5 neurons in the first layer.

To accomplish the weighting function performed by typical neurons, we decided to perform logical operations on each of the inputs to the neuron with known values (weights, in other words). We accomplished this by storing our binary connection weights in flops. We decided to implement these "connection weights" at the individual neuron because it eliminated the need to have one central memory holding all weights for the whole network. Each input to each neuron has two weights associated with it. The first weight is NAND-ed with the input, and the output of this operation is XOR-ed with the second weight.



The reason for including both of these operations is that they provide us with more flexibility in how the neuron deals with the input. The above schematic shows the hardware we are using to weight each input to each input. The following truth table illustrates the advantages of using two weights instead of one.

Weight1	Weight0	Input	Output
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



As you can see, each independent set of two weights passes a different input to the neuron's activation function.

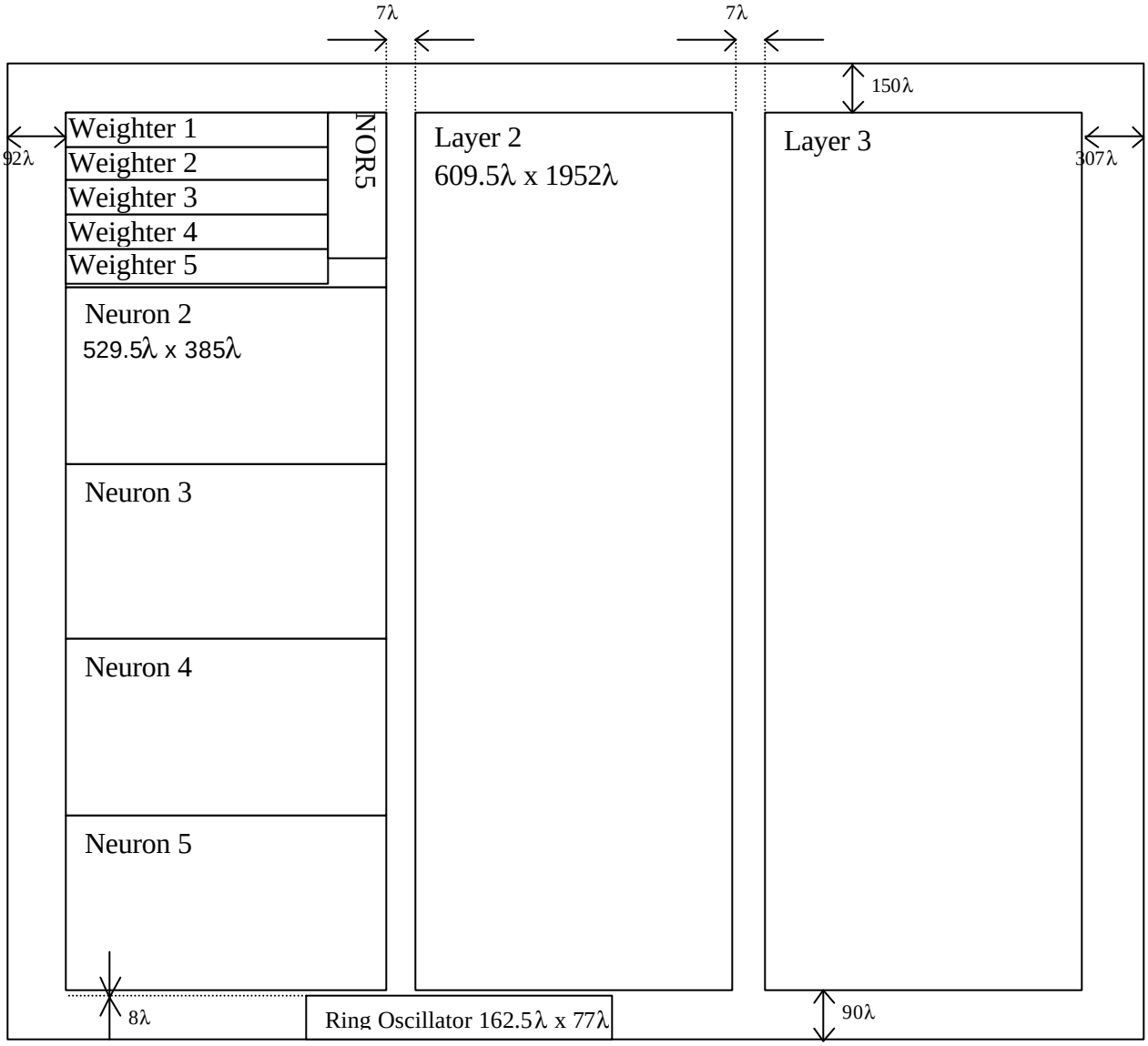
Our neuron's activation function is the NOR5. Each neuron weights each of its five inputs separately and then NOR5-s them all together. Now, given the above scheme for the four different possible weights that an input can be subject to, we can develop some insight as to how all the neurons should be weighted in order to produce a desired output from a given input.

For example, since the NOR is only high when all inputs to it are low, if we want to tell it to ignore an input, we tie the weighter corresponding to that input to ground (not physically but by giving it a weight of 01). Another example would be if we wanted to turn a neuron into a NOR5, we would just give all the weighters in the neuron the weight 11. This would tell the weighter to pass all five inputs to the neuron on to the NOR5 at the end. More details about this functionality are given in the simulation section.

So far we have ignored how the weights are given to the neural network. Each flop (storing one weight) in addition to having its output connected to a logic gate also has its output connected to the next flop. In this way, all of the flops all over the chip form a distributed shift register. The first flop on the chip will accept the weights to be shifted in, and as the chip (shift register) is clocked, the data will move through until all 150 weights have been shifted into the chip. Since the clock is spread over almost the entire chip, we are using 2-phase clocking with no overlap to ensure that all weights are properly shifted in without any data corruption. The downside of this is that to change a single weight you must shift in all the weights again, however, since there are only 150 weights, this shouldn't be too laborious using any sort of controller with a clock rate of a couple of hertz or more.

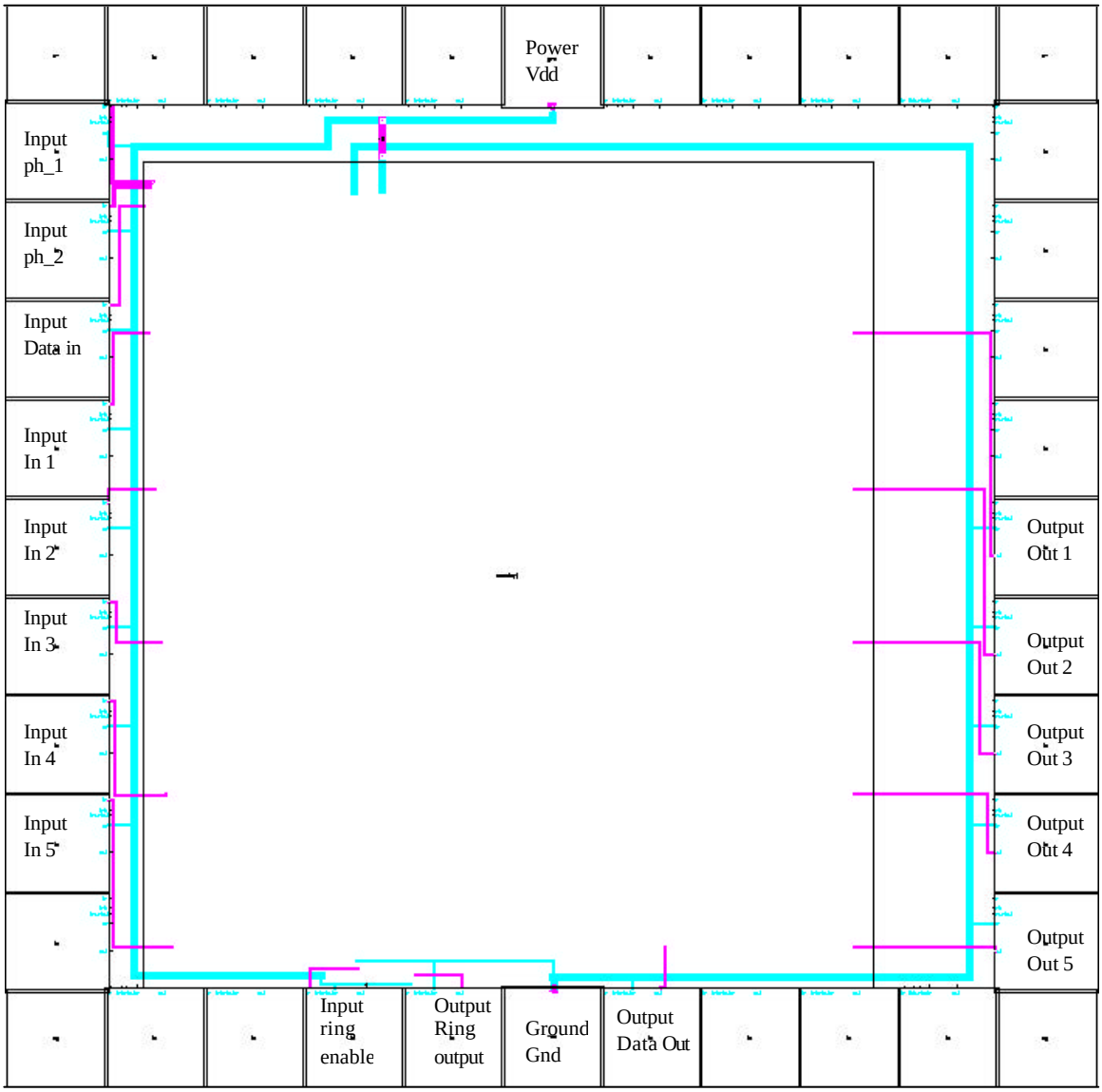
In short our design uses several abstractions from a continuous model of neural networks. Through appropriate choices for our weights we can perform various functions with our neuron. Our network consists of 3 layers (stack of 5 neurons) that are fully connected in order. That is, the inputs are fully connected to layer 1; layer 1 is fully connected to layer 2; etc. Each neuron's output is the NOR of the weighted inputs. The weights are input to the chip by means of a distributed shift register.

Chip Floorplan:



This floor plan excludes the pad frame for the chip. The outer dimensions of this floorplan include all of the facets in the core of our chip. This floorplan shows the biggest facets in the core. The first layer is dissected to show the neurons, and the first neuron is dissected to show the components. The weighter is laid out in a data path style. Within the data path are two flops, one NAND, and one XOR, which are not shown. Also, within the flop are two latches clocked separately. Thee are also not shown to keep the floorplan readable.

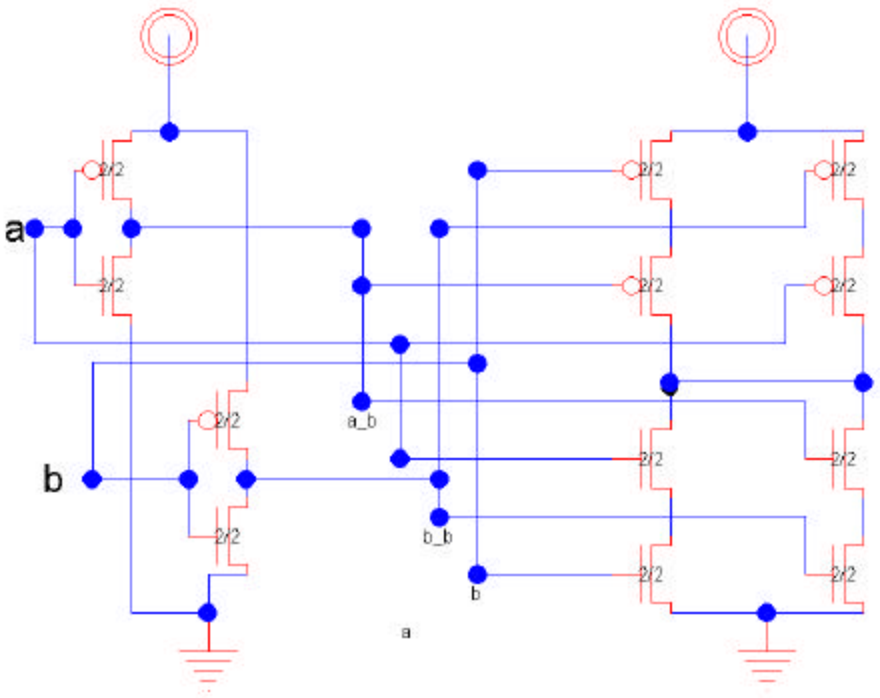
Chip Pinout:

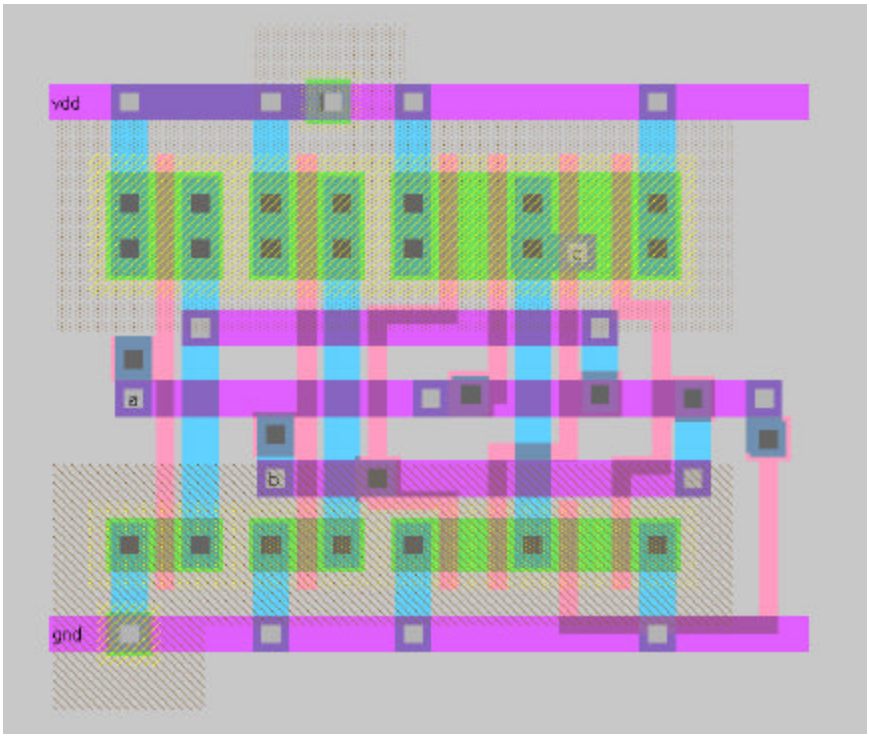


Description: To input weights put the weights (last weight first) on Data in while clocking ph_1 and ph_2. The five inputs to the network are inputs In 1 – In 5. The output of the weight shift register comes on data out. The five outputs of the network are Out 1 – Out 5. At the bottom, a small ring oscillator is provided for test purposes. Set ring enable to high to turn on the oscillator. We expect the frequency of the oscillator be around 230 MHz.

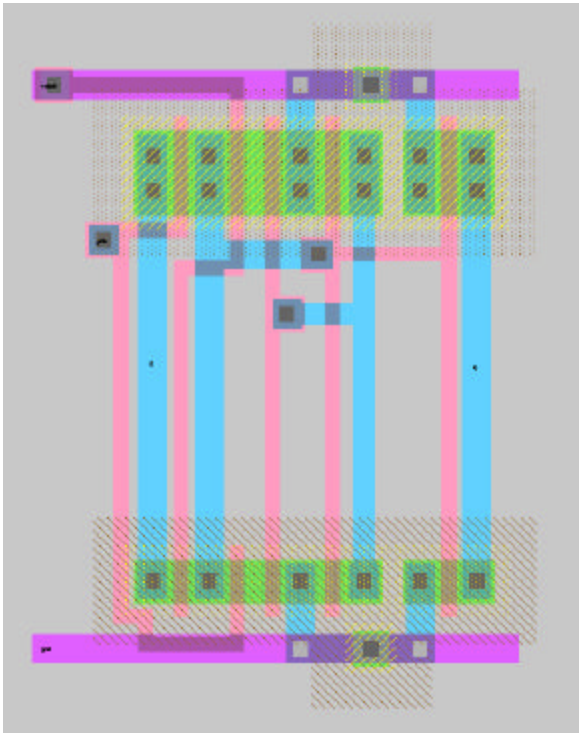
Leaf Cell Details:

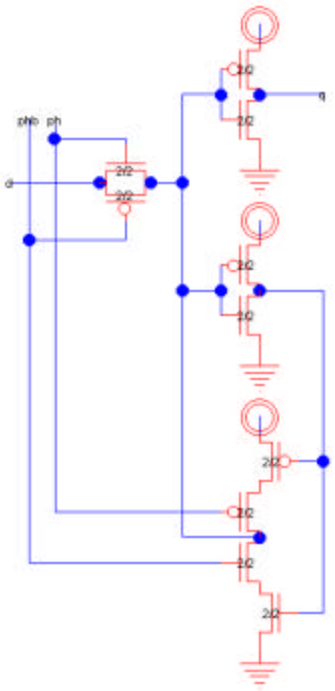
XOR



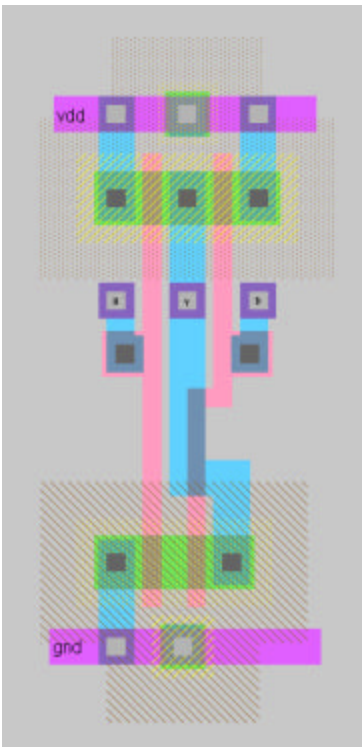


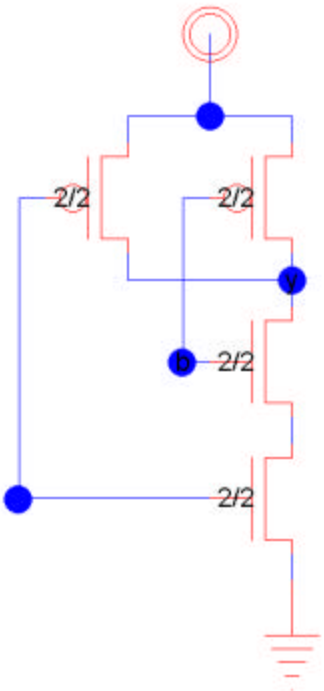
LATCH



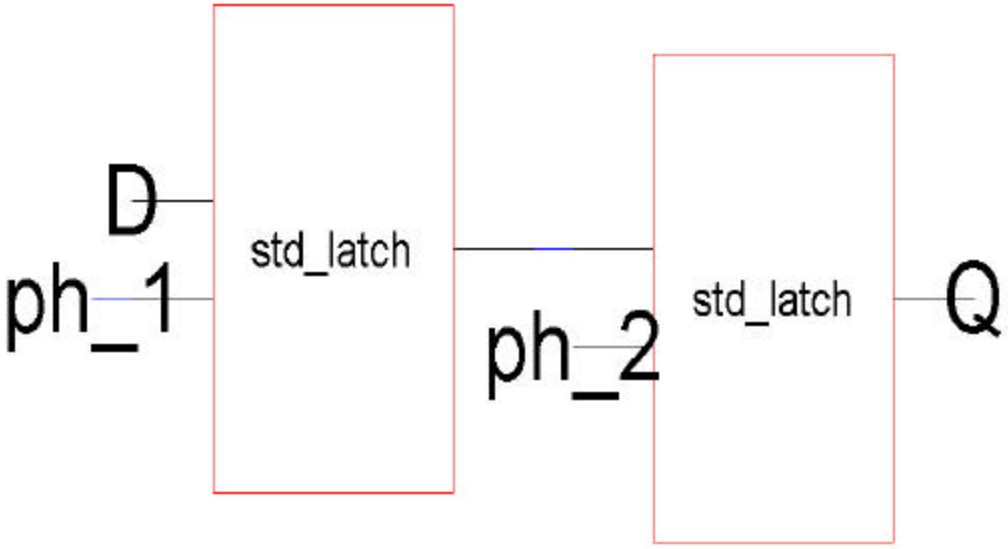


NAND2

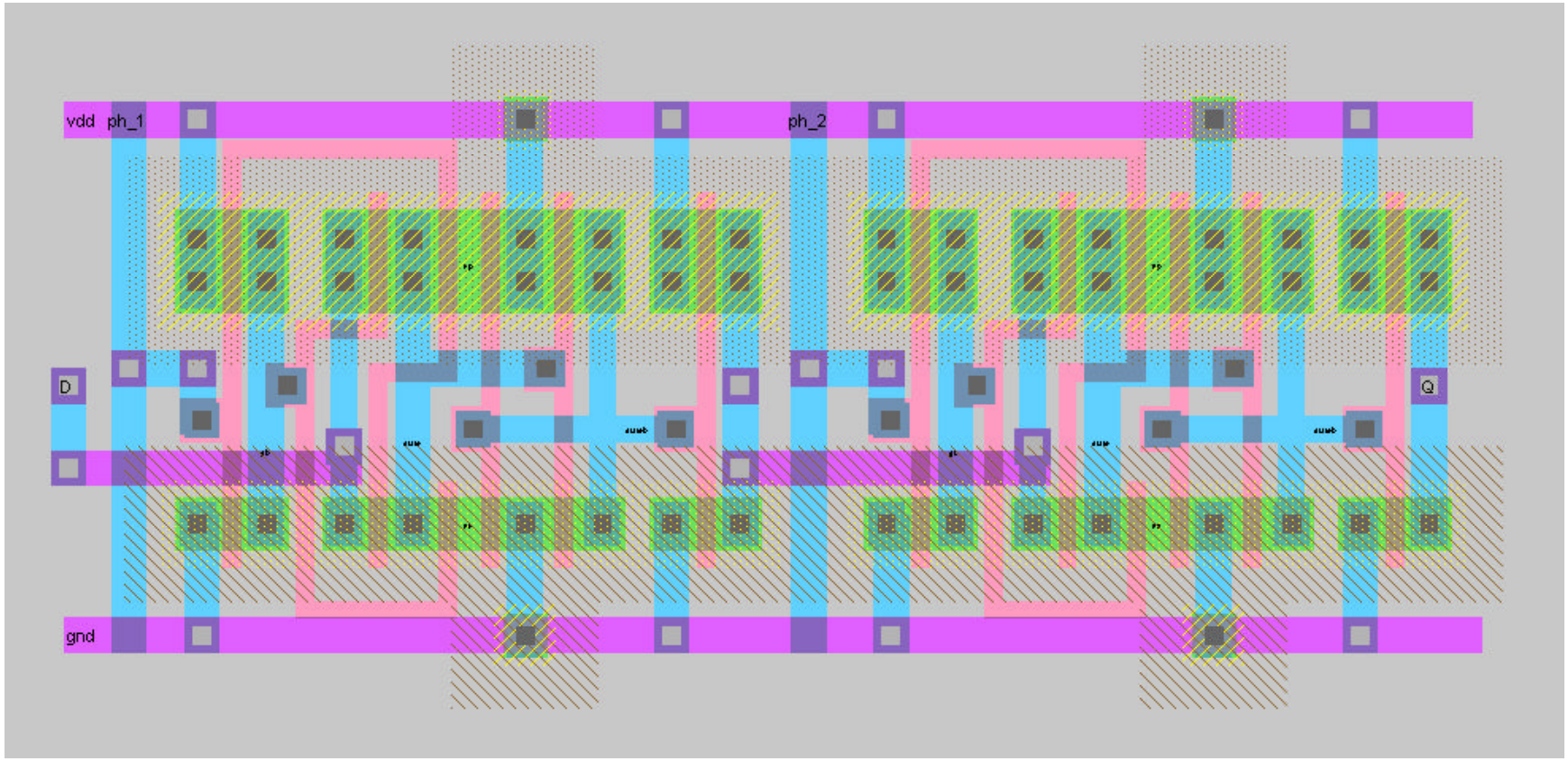




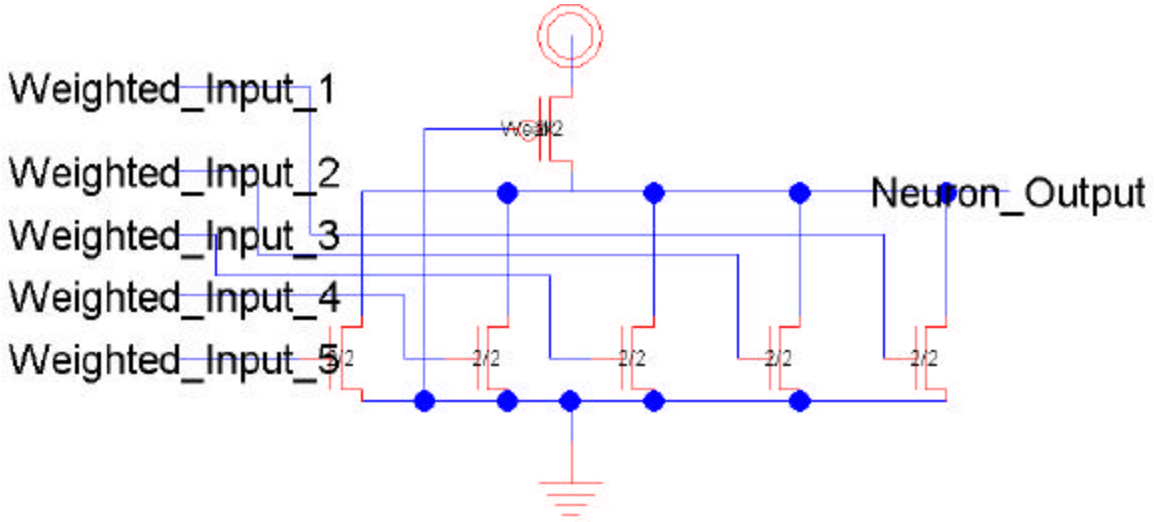
FLOP

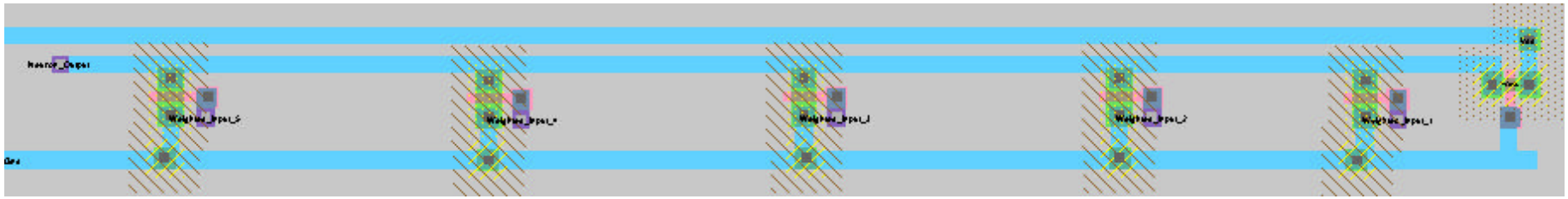


Flop Layout



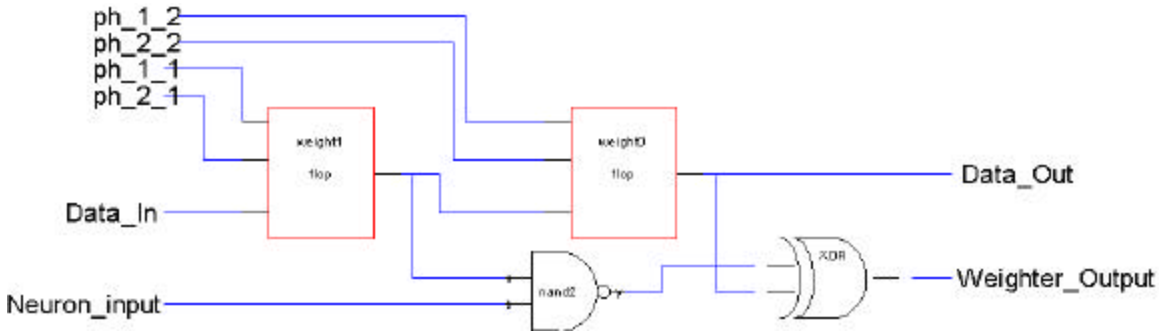
PSEUDO_NOR5

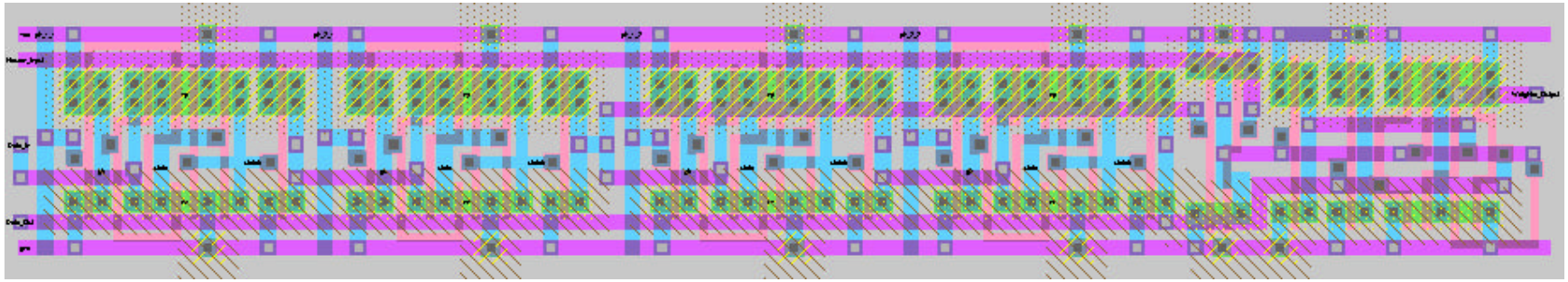




Description: This facet is a pseudo nmos gate. There's one weak pullup nmos transistor, far left, which is always on. The rest of the transistors are four times its size. If any of them turn on, the output (left side) will go low, making a NOR gate.

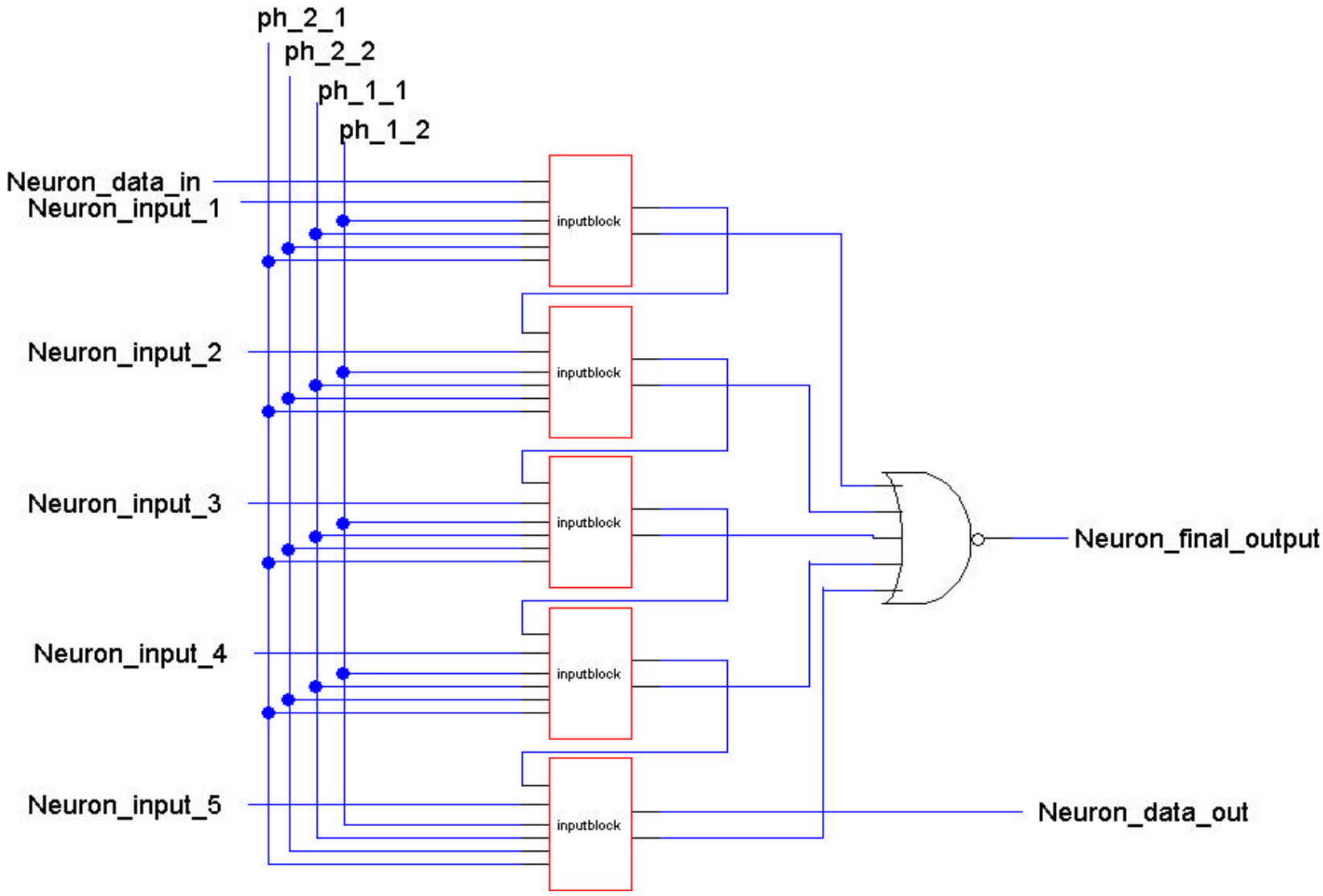
Weighter



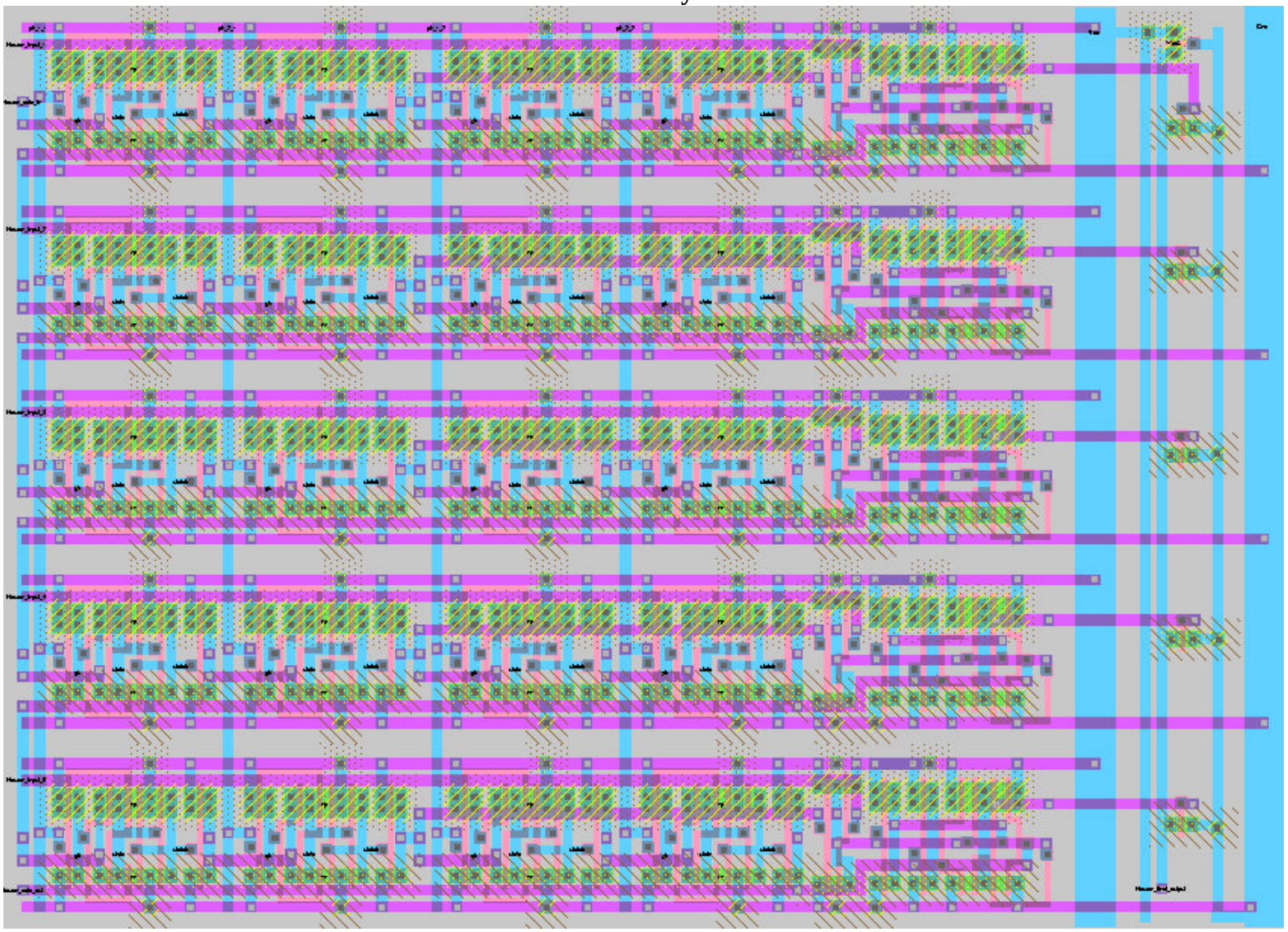


Description: This facet takes two ph_1 clocks and two ph_2 clocks to match the layout. The layout had two clocks so that each weighter can snap together with the weighter above, making the clocks run as four parallel lines over each layer. The input is exported on the far left in the middle, data in on the left at the top, and data out on the left at the bottom. The output is exported on the right.

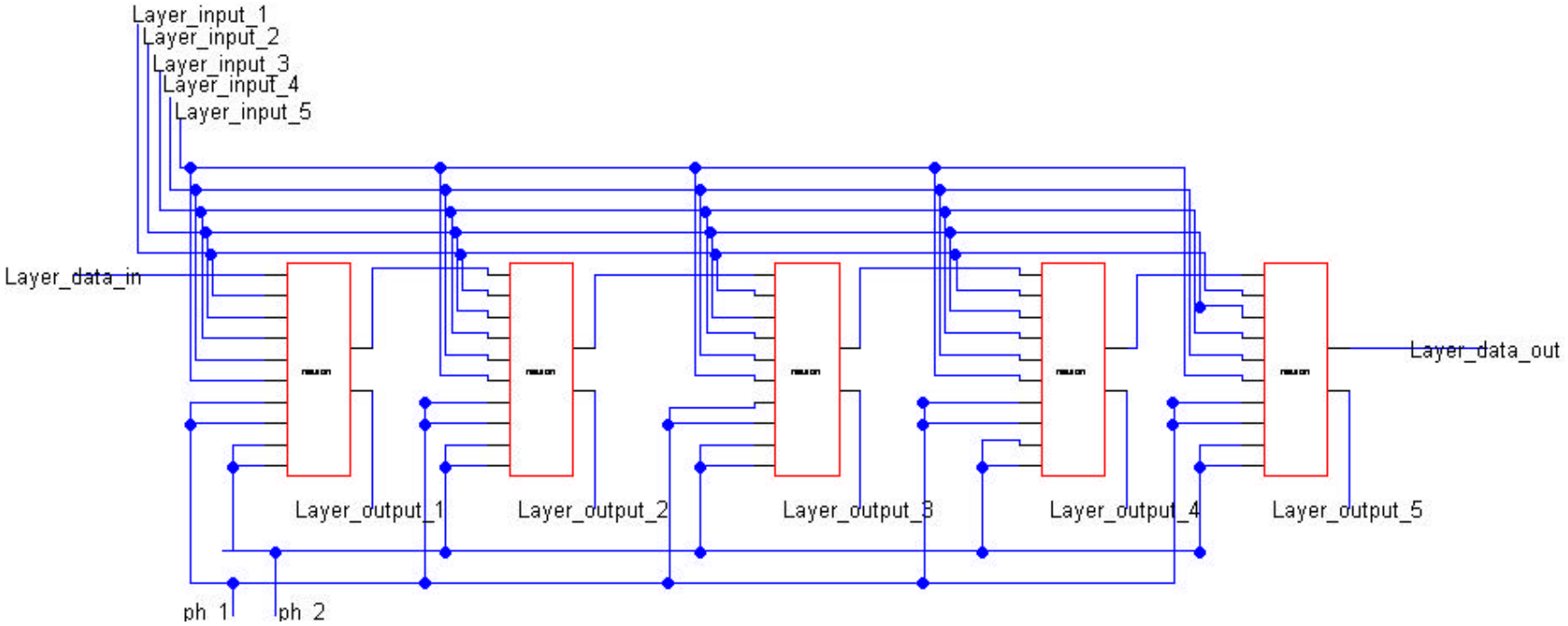
Neuron


ph_2_1
ph_2_2
ph_1_1
ph_1_2
Neuron_data_in
Neuron_input_1
Neuron_input_2
Neuron_input_3
Neuron_input_4
Neuron_input_5
inputblock
inputblock
inputblock
inputblock
inputblock
Neuron_final_output
Neuron_data_out

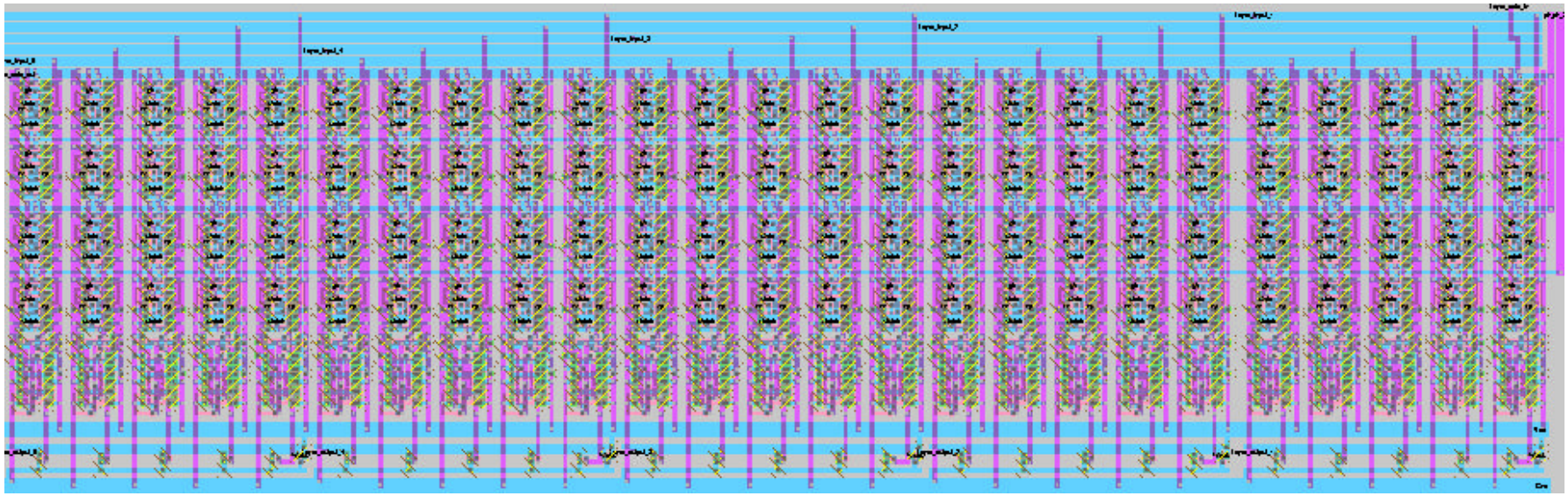
Neuron Layout



Layer

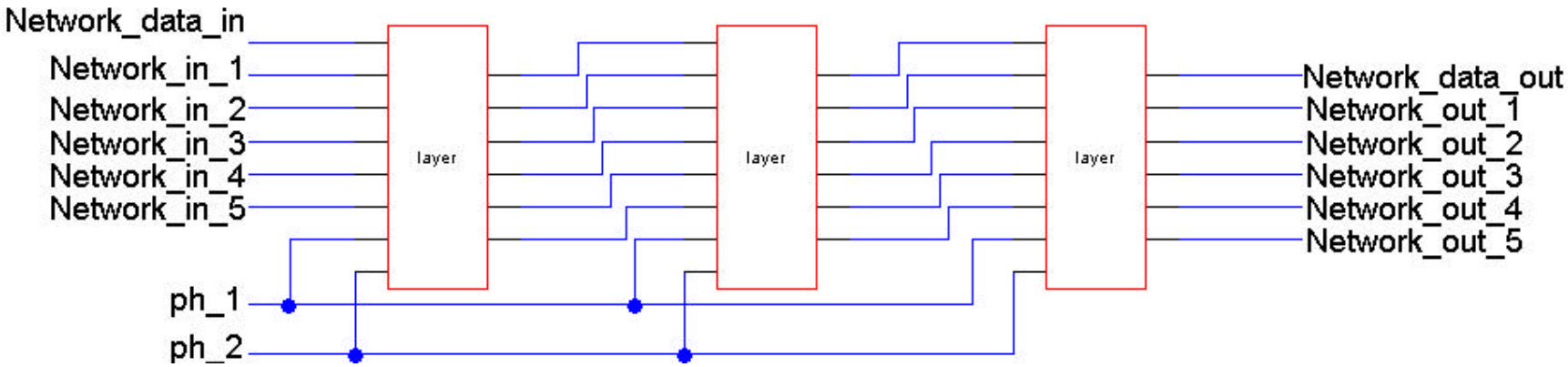

Layer_input_1
Layer_input_2
Layer_input_3
Layer_input_4
Layer_input_5
Layer_data_in
Layer_data_out
Layer_output_1
Layer_output_2
Layer_output_3
Layer_output_4
Layer_output_5
ph_1
ph_2

Layer Layout

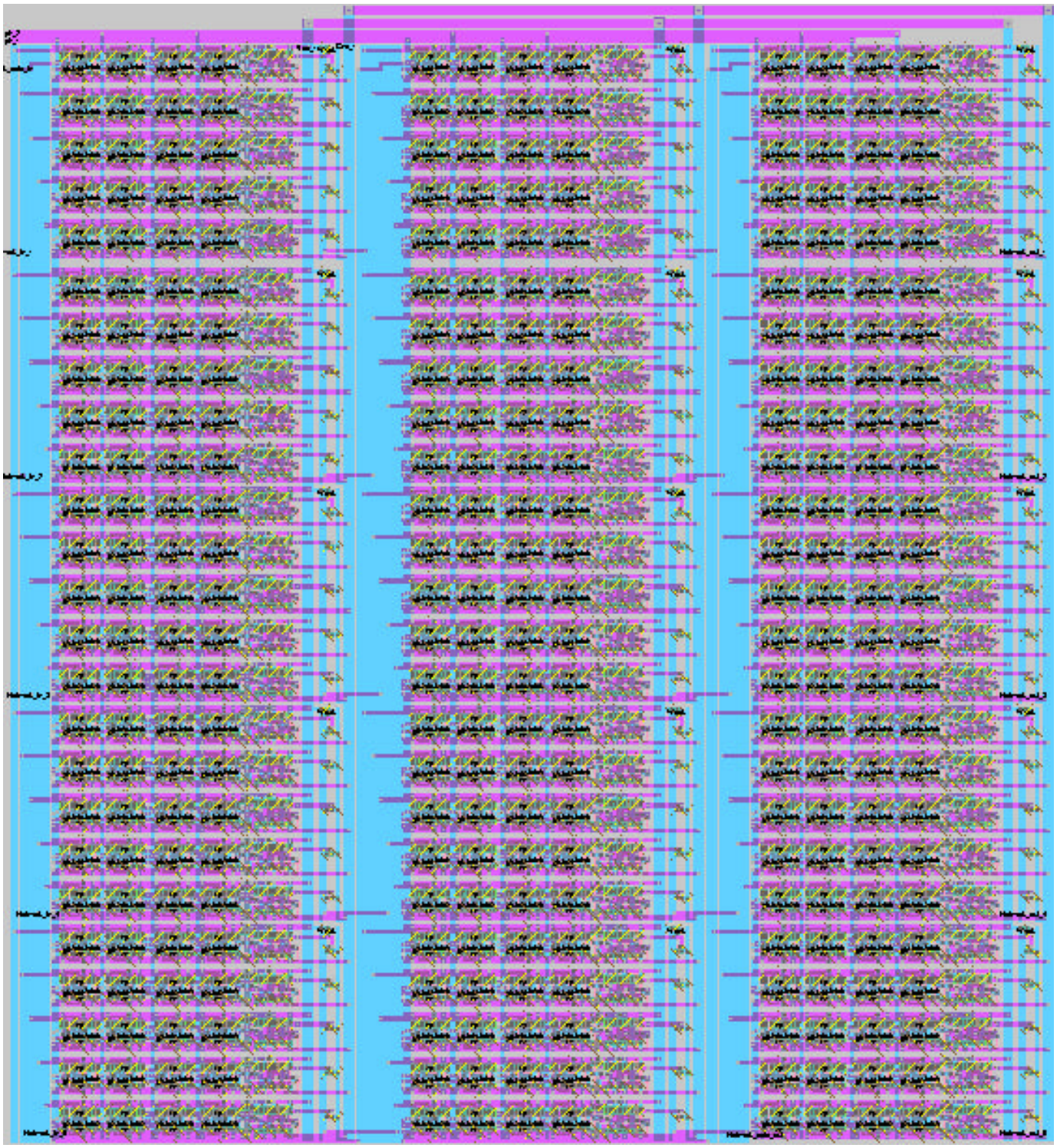


Note: This is rotated to be horizontal. The inputs come in the top and the outputs come out the bottom. It is five neurons stacked.

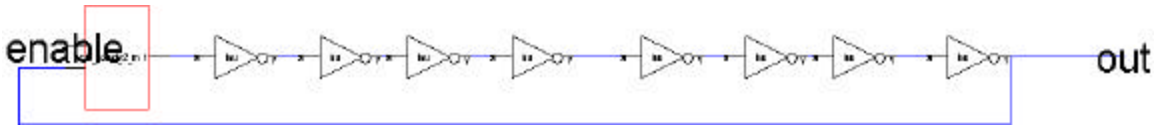
Network

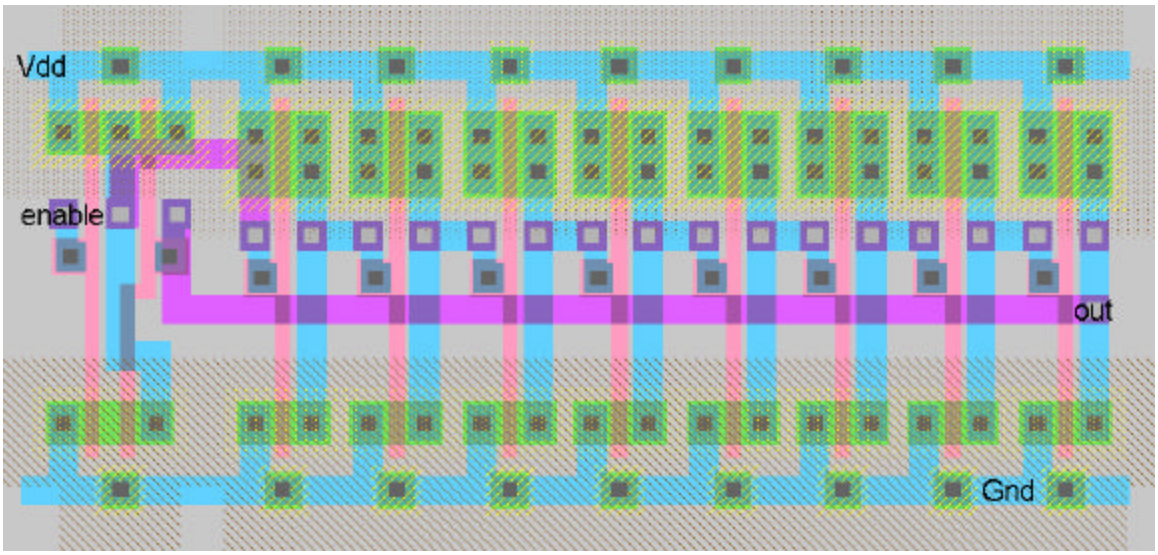

Network_data_in
Network_in_1
Network_in_2
Network_in_3
Network_in_4
Network_in_5
ph_1
ph_2
layer
layer
layer
Network_data_out
Network_out_1
Network_out_2
Network_out_3
Network_out_4
Network_out_5

Network Layout

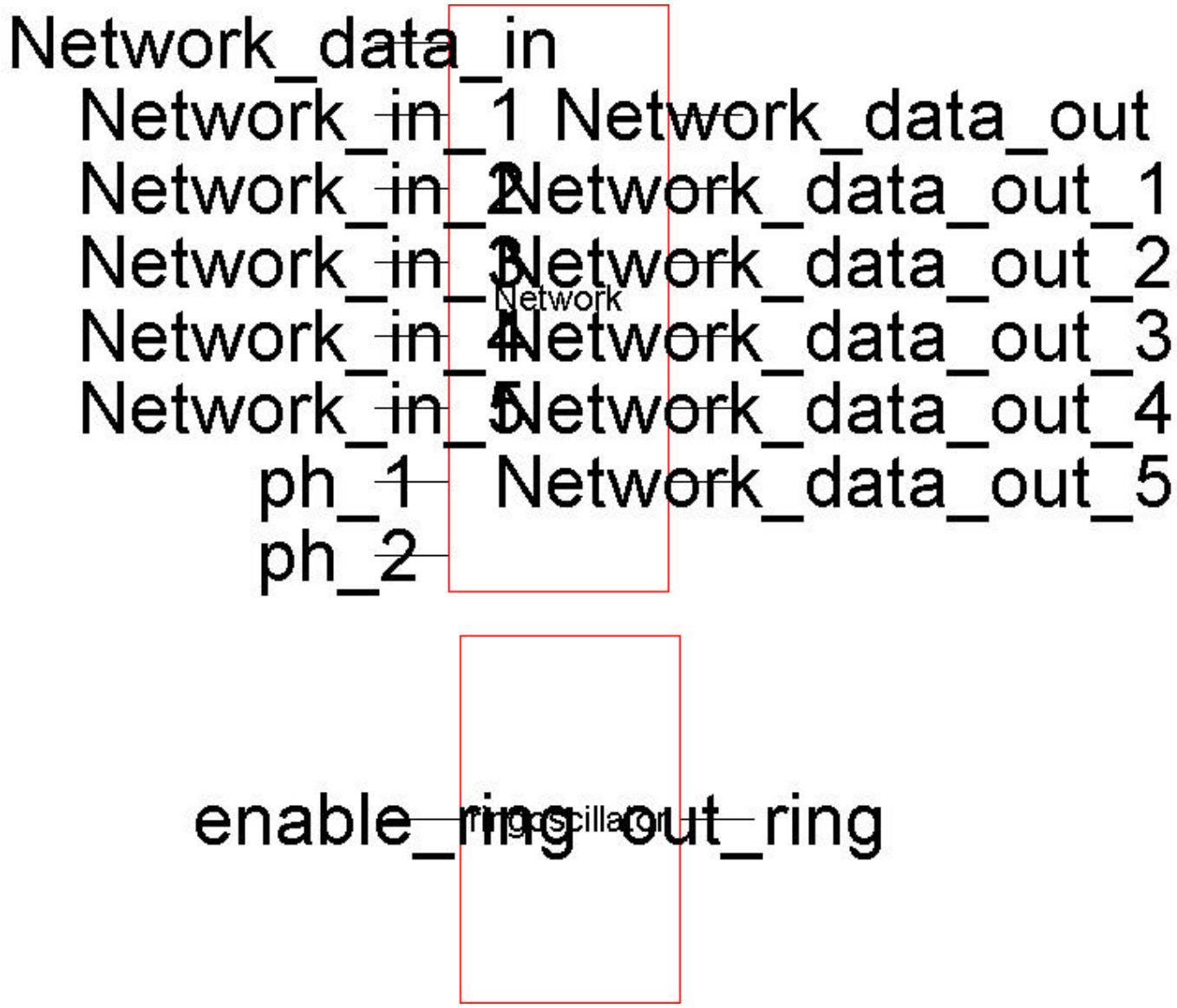


Ring Oscillator

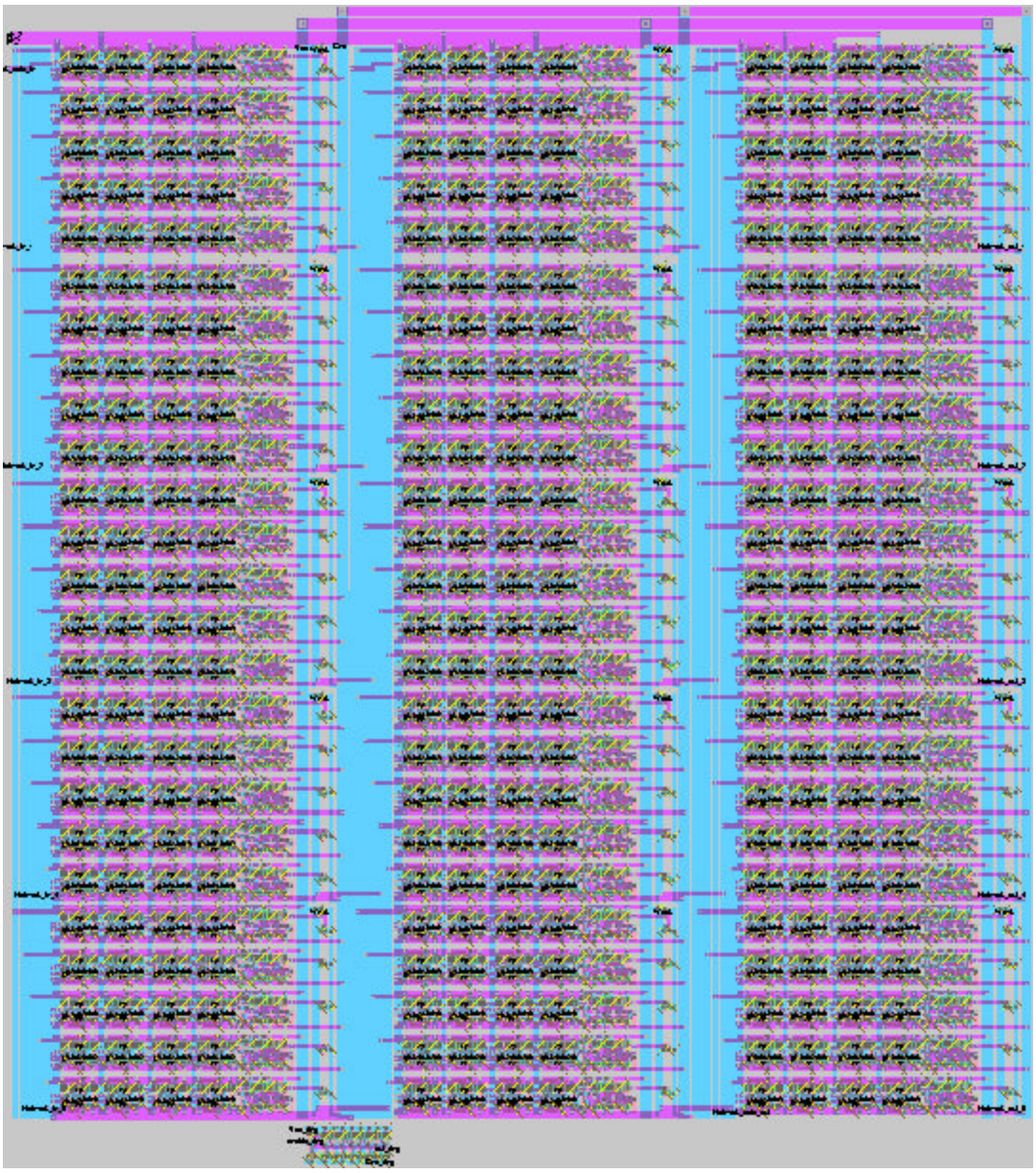




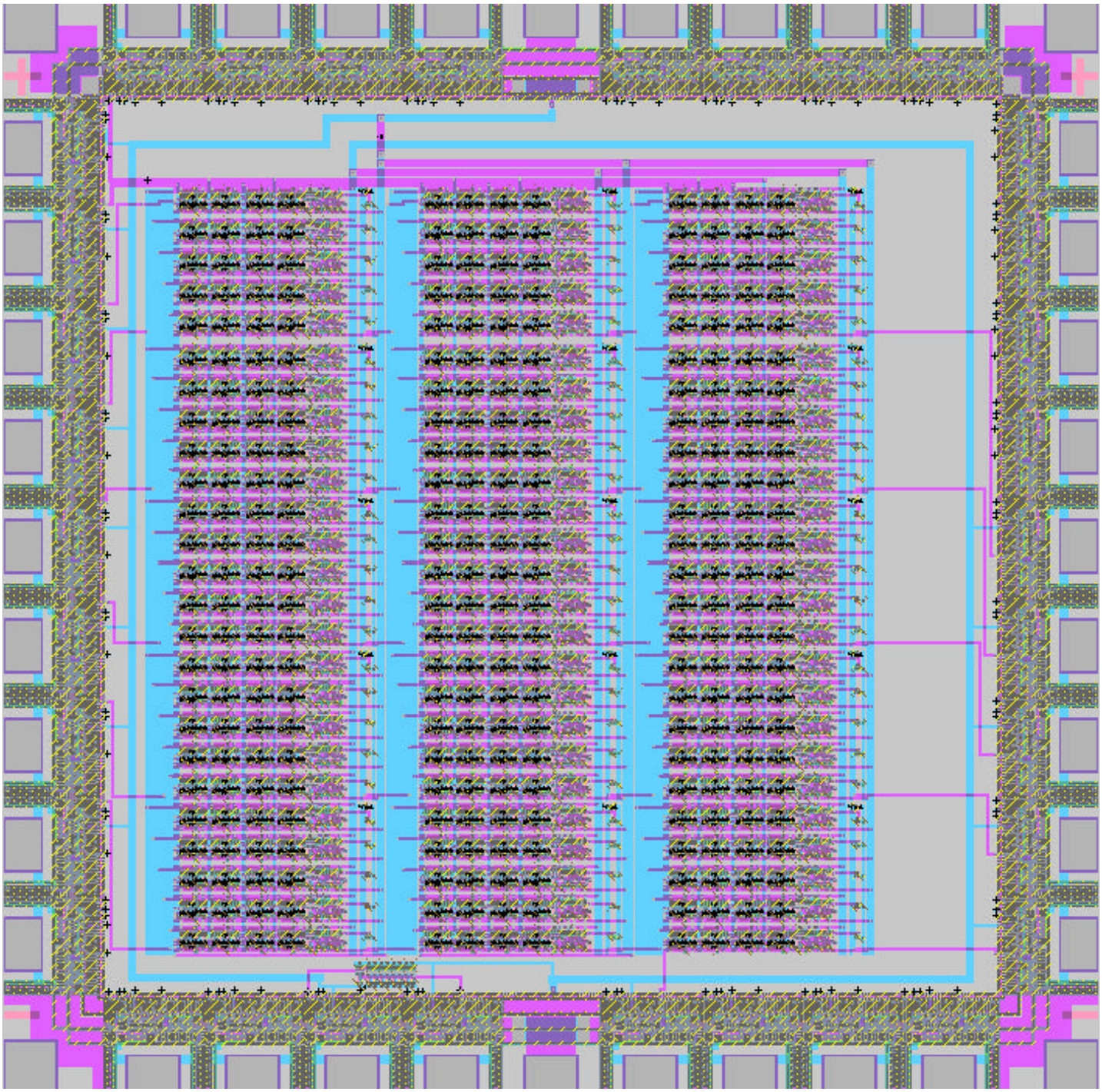
Core


Network_data_in
Network_in_1
Network_in_2
Network_in_3
Network_in_4
Network_in_5
ph_1
ph_2
Network
Network_data_out
Network_data_out_1
Network_data_out_2
Network_data_out_3
Network_data_out_4
Network_data_out_5
enable_ring
ring_oscillator
out_ring

Core layout



Top Level

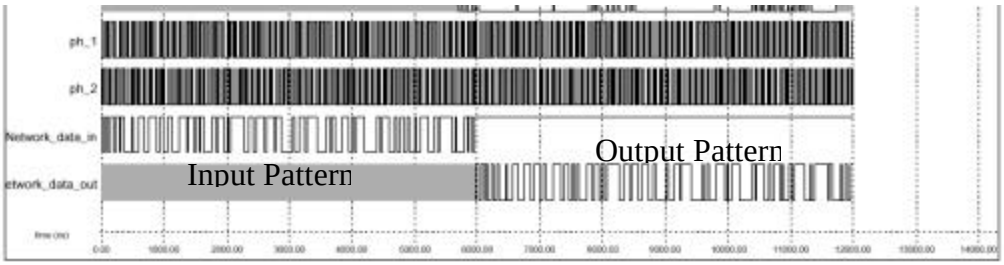


Summary of Design:

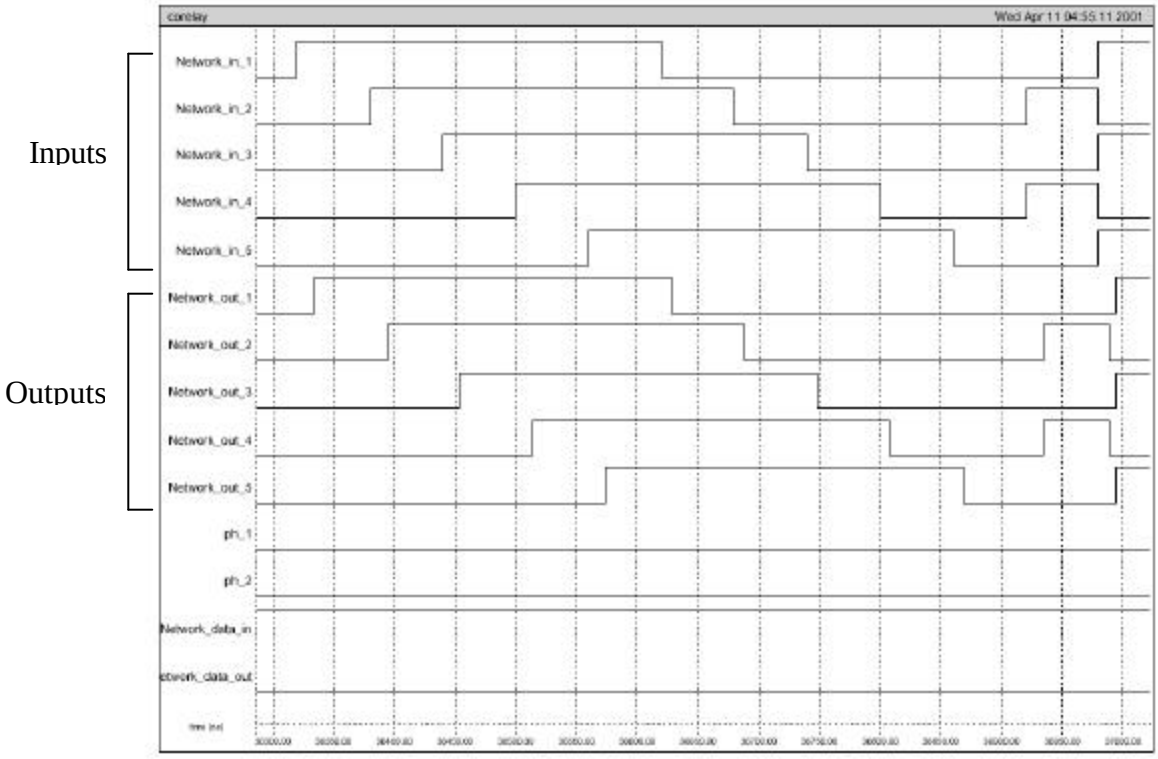
	Cells	Actual Cell Size	Actual Cell Area	Transistor Count	Area / Transistor	Design Time Sch (hrs.)	Lay (hrs.)	Tests (hrs.)	Comments
1	std_latch	80 x 77	6,160	12	513.3333333	0	0.5	0.5	modification of standard cell to use metal 2 as horizontal
2	flop	169 x 77	13,013	24	542.2083333	0.5	1	1	using two modified std_latches back to back to form a flop
3	std_nand2	33 x 77	2,541	4	635.25	0	0.5	0.5	modification of standard cell to use metal 2 as horizontal
4	XOR	85.5 x 77	6,584	12	548.625	1.5	3	0.5	some layout work required to keep within floorplan estimates
5	Weighter	435.5 x 77	33,534	64	523.9609375	0.5	6	2	quite a lot of layout work to ensure snap-together-ness
6	Psudo-NOR5	46.5 x 380.5	17,693	6	2948.875	0.5	2	0.5	weird size and shape to fit the height of the whole neuron
7	Neuron	529.5 x 385	203,858	326	625.3297546	0.75	4	2.5	also had to ensure the ability of neurons to snap together
8	Layer	609.5 x 1952	1,189,744	1630	729.9042945	0.5	2.5	0.5	pretty quick, involved zipping together neurons
9	Network	1841 x 2001	3,683,841	4890	753.3417178	0.25	2	0	three layers placed side by side, with a few connections
10	Ring Oscillator	162.5 x 77	12,513	20	625.625	0.25	0.5	0.5	
11	Core	1841 x 2086.5	3,841,247	4910	782.3312627	0.25	0.75	1.5	
12	Top	2754 x 2754	7584516		-		1.25		
	Totals:					5	24	10	Total Design Time: 39 hrs.

This is the summary of how much time we spent actually working with electric for the chip. We actually spent quite a bit of time on developing the concept of how we could best perform the functions of a theoretical neural network given our limited hardware, experience, and time. We were unable to actually figure out the transistor count of the pad frame and so those entries are left blank on our table. Part of what contributes to the slightly high (for a datapath) area per transistor is the fully connected nature of our network. This involved quite a few big wires, which do wonders at increasing the area of our layout without changing our transistor count.

Simulation Details:



Result I: This is the graphical output of the shifttest script (see appendix) on the core layout. None of the assertions in shifttest failed. This test fills all 150 flops in the network with random values. It then clocks the chip 150 times to get out all the values again and checks to make sure that they are all correct. You can see that the output pattern (left half of the simulation, matches in the input pattern (right half of the simulation. This indicates that our shift register at least is working properly.



Result II: This is the output of the identtest1 script on the core layout. This test shifts in a pattern of weights which makes the identity network, each output value equals its corresponding input values. Prior to the waveforms above, the 150 bits were shifted in.

Here we can see that after the shifting, the outputs are indeed matching the inputs to the network. None of the assertions in idnettest1 failed on this test. There are five different identity tests, each one of which rotates the output by a different amount. Identtest2 rotates the output by 3 (1 rotation per layer).

Design Verification:

	Cells	Sub-Cells	Complexity	Simulates	DRC	ERC	NCC	Estimated Cell Size	Actual Cell Size
1	std_latch	-	N/A	x	x	x	x	80 x 80	80 x 77
2	flop	1	1	x	x	x	x	80 x 160	169 x 77
3	std_nand2	-	N/A	x	x	x	x	80 x 40	33 x 77
4	XOR	-	3	x	x	x	x	80 x 160	85.5 x 77
5	Weighter	1,2,3,4	4	x	x	x	x	80 x 520	435.5 x 77
6	PsudoNMOS NOR5	-	2	x	x	x	x	80 x 240	46.5 x 380.5
7	Neuron	5,6	5	x	x	x	x	400 x 600	529.5 x 385
8	Layer	7	4	x	x	x	x	2000 x 600	609.5 x 1952
9	Network	8	3	x	x	x	x	2000 x 1900	1841 x 2001
10	Ring Oscillator	-	2	XXX	x	x	x	XXX	162.5 x 77
11	Core	9,10	2	x	x	XXX	Gem	XXX	1841 x 2086.5
12	Top	10,11	3		XXX	XXX	XXX	XXX	2754 x 2754

The different components of our system all simulate fine individually and all pass DRC, ERC, and NCC. However, the ring oscillator, which we placed on the chip as a test structure to be used once the chip is fabricated doesn't pass ERC when placed in the overall core of the chip because of its placement. Since it is crammed beneath our network, it didn't make sense to connect it to the power and ground in the core-level layout. We connected it, instead, to power and ground in the top-level schematic where power and ground are needed to enable the appropriate input and output pads. Since power and ground were not connected to Network power and ground in the core, ERC saw the P wells and N wells not connected to any source of power or ground to properly bias them against leakage into the substrate. Hence the ring oscillator made our overall core layout fail ERC. It also failed Electric's NCC however, Gemini said that it was equivalent to the schematic we provided for this layer.

Additionally, the ring oscillator was unable to simulate properly on IRSIM. We believe that the reason for this is that it might be a difficult circuit for IRSIM to analyze in terms of timing, since it is really a loop that doesn't end and is dependent on the state of the oscillator system at any given time.

Another point of interest is the top-level cell (facet). It failed DRC because the pad frame is imported from Caltech Interchange Format to Electric and is hence described in terms of pure layers. This does not fit with Electric's schema for performing DRC. ERC was never able to go through to completion without crashing Electric on the Top-level layout, and NCC was impossible to run as well since we were not provided with a schematic for the pad frame.

When we proposed the project, we hadn't proposed to put a ring oscillator on the core and so there is no estimated size for this. The size of the top-level layout was what we were designing for, and our design fits comfortably inside it with enough room to insert a test structure such as our ring oscillator.

On the basis of simulations and the checkers in electric and IRSIM, however, we are fairly confident that our design is sound and will be able to perform its functions in a satisfactory manner.

Test Plan:

The plan for the testing procedure of our chip after fabrication is as follows:

1. Use the ring oscillator structure on the chip with the enable tied to Vdd and measure the output with an oscilloscope. We expect to see a (probably deformed) square wave output at approx 230 MHz. We are using a 9-stage oscillator that has a pretty high frequency, but it is still within the upper limits of the measuring capacity of the oscilloscopes available to us in the electronics lab at Harvey Mudd College.
2. Upon confirmation that the chip is not defective (as confirmed by step 1), the shift register's functionality should then be tested by physically performing the functions of the program **shifttest** which is included in Appendix A. **Shifttest** shifts into the large register a random sequence of 150 weights, and tests them to make sure that they are shifted out properly and uncorrupted.
3. Once the functionality of the shift register is assured, the first step will be to try to replicate the results of the program **identtest1** (see Appendix A). This program shifts in the appropriate weights to allow the network to replicate an input pattern. It should be noted that the chip is entirely combinational once the weights have been shifted in. The clock's sole purpose is for the purpose of moving data through the shift register.

   This is a good starting point because it is a relatively simple pattern to diagnose troubles in individual neurons' circuitry. If the given input vector doesn't match the given output vector, there is a problem. The neuron, but not the layer at fault can be pinpointed using one-hot encoded vectors. Further

application of the proper weights and vectors should allow the tester to diagnose the neuron/layer at fault. If necessary, weights in other neurons may be adjusted to accommodate the failure of a few neurons. We can treat the output of the neuron as an "ignore" by weighted its respective input at all connected neurons with 01.

4. Once appropriate neurons are determined to be functional, executing[1] any of the following programs on the chip will more thoroughly test functionality of the combinational logic on the chip: **identtest2, identtest3, identtest4, identtest5** (see Appendix A). These all cause each layer to barrel shift the input pattern by a fixed amount. **Identtest2** barrel shifts the input by 1 place at each layer for a total shift of three slots. Likewise, **identtest3** barrel shifts the input pattern by 2 places at each layer for a total of 6 places. This pattern of incrementing the shift amount per layer continues through **identtest5**.
5. Finally, upon successful completion of these tests, we may look at more logically complex functions. It should not be hard to write a program to teach the neural network to accomplish a task using a Boltzmann learning algorithm. This involves providing an input and desired output, and randomly switching weights to decrease the overall "Energy" function of the system. More investigation is definitely required to do this kind of training. However, provided in Appendix B is a Matlab description of a Neural Network that was written to describe this particular network. So, if one wanted to simulate the results provided by such a training program without actually having to load

weights onto the chip, they should run the set of weights given by their training program into the provided Matlab code and try different test vectors to see whether or not the training program has given an accurate set of weights to perform the particular application.

[1] It should be noted that by program execution, we mean to actually hook up a processor to shift in the appropriate weights given in the program and then physically providing the necessary inputs to the chip and measuring the outputs.

Appendix A:Test Files

```
----------------------------------inv-----------------------------------------------------------------
l Network_data_in
c
h Network_data_in
c

------------------------buff-----------------------------------------------
l Network_data_in
c
l Network_data_in
c

------------------------one------------------------------------------------
h Network_data_in
c
l Network_data_in
c

------------------------zero-----------------------------------------------
h Network_data_in
c
h Network_data_in
c

------------------------checkbuff------------------------------------------
assert Network_data_out 1
c
assert Network_data_out 1
c

------------------------checkinv-------------------------------------------
assert Network_data_out 0
c
assert Network_data_out 1
c

------------------------checkone-------------------------------------------
assert Network_data_out 0
c
assert Network_data_out 0
c

------------------------checkzero------------------------------------------
assert Network_data_out 1
c
assert Network_data_out 0
c

------------------------initclock------------------------------------------
clock ph_1 1 0 0 0
clock ph_2 0 0 1 0

------------------------shifttest------------------------------------------
```

```
@ initclock

@ zero
@ zero
@ zero
@ zero
@ zero

@ one
@ zero
@ inv
@ zero
@ inv

@ buff
@ inv
@ zero
@ inv
@ one

@ inv
@ buff
@ inv
@ zero
@ zero

@ zero
@ one
@ buff
@ inv
@ zero

@ zero
@ buff
@ buff
@ zero
@ inv

@ zero
@ buff
@ zero
@ buff
@ zero

@ buff
@ one
@ one
@ zero
@ inv

@ zero
@ buff
@ buff
@ one
@ one

@ zero
```

```
@ zero
@ one
@ buff
@ zero

@ zero
@ buff
@ zero
@ one
@ one

@ zero
@ buff
@ zero
@ inv
@ inv

@ zero
@ zero
@ inv
@ inv
@ buff

@ zero
@ inv
@ inv
@ buff
@ buff

@ inv
@ one
@ inv
@ inv
@ inv

@ checkzero
@ checkzero
@ checkzero
@ checkzero
@ checkzero

@ checkone
@ checkzero
@ checkinv
@ checkzero
@ checkinv

@ checkbuff
@ checkinv
@ checkzero
@ checkinv
@ checkone

@ checkinv
@ checkbuff
@ checkinv
@ checkzero
```

```
@ checkzero

@ checkzero
@ checkone
@ checkbuff
@ checkinv
@ checkzero

@ checkzero
@ checkbuff
@ checkbuff
@ checkzero
@ checkinv

@ checkzero
@ checkbuff
@ checkzero
@ checkbuff
@ checkzero

@ checkbuff
@ checkone
@ checkone
@ checkzero
@ checkinv

@ checkzero
@ checkbuff
@ checkbuff
@ checkone
@ checkone

@ checkzero
@ checkzero
@ checkone
@ checkbuff
@ checkzero

@ checkzero
@ checkbuff
@ checkzero
@ checkone
@ checkone

@ checkzero
@ checkbuff
@ checkzero
@ checkinv
@ checkinv

@ checkzero
@ checkzero
@ checkinv
@ checkinv
@ checkbuff

@ checkzero
```

```
@ checkinv
@ checkinv
@ checkbuff
@ checkbuff

@ checkinv
@ checkone
@ checkinv
@ checkinv
@ checkinv


----------------------------layerident1---------------------------------
@ inv
@ zero
@ zero
@ zero
@ zero

@ zero
@ inv
@ zero
@ zero
@ zero

@ zero
@ zero
@ inv
@ zero
@ zero

@ zero
@ zero
@ zero
@ inv
@ zero

@ zero
@ zero
@ zero
@ zero
@ inv

------------------------------layerident2----------------------------
@ zero
@ inv
@ zero
@ zero
@ zero

@ zero
@ zero
@ inv
@ zero
@ zero

@ zero
```

```
@ zero
@ zero
@ inv
@ zero

@ zero
@ zero
@ zero
@ zero
@ inv

@ inv
@ zero
@ zero
@ zero
@ zero

---------------------------layerident3---------------------------------
@ zero
@ zero
@ inv
@ zero
@ zero

@ zero
@ zero
@ zero
@ inv
@ zero

@ zero
@ zero
@ zero
@ zero
@ inv

@ inv
@ zero
@ zero
@ zero
@ zero

@ zero
@ inv
@ zero
@ zero
@ zero

------------------------------layerident4---------------------------------
@ zero
@ zero
@ zero
@ inv
@ zero

@ zero
@ zero
```

```
@ zero
@ zero
@ inv

@ inv
@ zero
@ zero
@ zero
@ zero

@ zero
@ inv
@ zero
@ zero
@ zero

@ zero
@ zero
@ inv
@ zero
@ zero

-------------------------layerident5----------------------------------
@ zero
@ zero
@ zero
@ zero
@ inv

@ inv
@ zero
@ zero
@ zero
@ zero

@ zero
@ inv
@ zero
@ zero
@ zero

@ zero
@ zero
@ inv
@ zero
@ zero

@ zero
@ zero
@ zero
@ inv
@ zero

-----------------------netident1--------------------------------------
@ layerident1
@ layerident1
@ layerident1
```

```
------------------------netident2-------------------------------------
@ layerident2
@ layerident2
@ layerident2

------------------------netident3-------------------------------------
@ layerident3
@ layerident3
@ layerident3

------------------------netident4-------------------------------------
@ layerident4
@ layerident4
@ layerident4

------------------------netident5-------------------------------------
@ layerident5
@ layerident5
@ layerident5

------------------------identtest1-------------------------------------
@ initclock
@ netident1

l Network_in_1 Network_in_2 Network_in_3 Network_in_4 Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_1
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_2
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_3
s
s
assert Network_out_1 1
assert Network_out_2 1
```

```
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

h Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

h Network_in_5
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1



l Network_in_1
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1

l Network_in_2
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1

l Network_in_3
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

l Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
```

```
assert Network_out_4 0
assert Network_out_5 1

l Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_1 Network_in_3 Network_in_5
h Network_in_2 Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 0

h Network_in_1 Network_in_3 Network_in_5
l Network_in_2 Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 1

--------------------------identtest2-----------------------------------
@ initclock
@ netident2

l Network_in_1 Network_in_2 Network_in_3 Network_in_4 Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_1
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 0

h Network_in_2
s
s
```

```
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

h Network_in_3
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

h Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

h Network_in_5
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1



l Network_in_1
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 1

l Network_in_2
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

l Network_in_3
s
s
assert Network_out_1 0
```

```
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

l Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

l Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_1 Network_in_3 Network_in_5
h Network_in_2 Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 1

h Network_in_1 Network_in_3 Network_in_5
l Network_in_2 Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

------------------------------identtest3---------------------------------
@ initclock
@ netident3

l Network_in_1 Network_in_2 Network_in_3 Network_in_4 Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_1
```

```
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_2
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

h Network_in_3
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

h Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1

h Network_in_5
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1



l Network_in_1
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1

l Network_in_2
s
```

```
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

l Network_in_3
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 1

l Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_1 Network_in_3 Network_in_5
h Network_in_2 Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 1

h Network_in_1 Network_in_3 Network_in_5
l Network_in_2 Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 0

----------------------identtest4----------------------------------------
@ initclock
@ netident4
```

```
l Network_in_1 Network_in_2 Network_in_3 Network_in_4 Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_1
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 1

h Network_in_2
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 1

h Network_in_3
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 1

h Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 1

h Network_in_5
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1
```

```
l Network_in_1
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

l Network_in_2
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

l Network_in_3
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

l Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 0

l Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_1 Network_in_3 Network_in_5
h Network_in_2 Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

h Network_in_1 Network_in_3 Network_in_5
l Network_in_2 Network_in_4
```

```
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

----------------------------identtest5-----------------------------------
@ initclock
@ netident5

l Network_in_1 Network_in_2 Network_in_3 Network_in_4 Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

h Network_in_1
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 0

h Network_in_2
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 0

h Network_in_3
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1

h Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1
```

```
h Network_in_5
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 1
assert Network_out_5 1



l Network_in_1
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 1

l Network_in_2
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 1

l Network_in_3
s
s
assert Network_out_1 1
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_5
s
s
assert Network_out_1 0
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 0
assert Network_out_5 0

l Network_in_1 Network_in_3 Network_in_5
```

```
h Network_in_2 Network_in_4
s
s
assert Network_out_1 1
assert Network_out_2 0
assert Network_out_3 0
assert Network_out_4 1
assert Network_out_5 0

h Network_in_1 Network_in_3 Network_in_5
l Network_in_2 Network_in_4
s
s
assert Network_out_1 0
assert Network_out_2 1
assert Network_out_3 1
assert Network_out_4 0
assert Network_out_5 1

--------------------testall-----------------------------------
@ shifttest
print ************************************************************
print shift test complete
print ************************************************************

@ identtest1
print ************************************************************
print identtest1 complete
print ************************************************************

@ identtest2
print ************************************************************
print identtest2 complete
print ************************************************************

@ identtest3
print ************************************************************
print identtest3 complete
print ************************************************************

@ identtest4
print ************************************************************
print identtest4 complete
print ************************************************************

@ identtest5
print ************************************************************
print identtest5 complete
print ************************************************************
```

Appendix B: Matlab Simulation

We wrote a simulation of our network in matlab. The hope was that we would be able to train this simulation in matlab and then write the learned weights onto the chip, thus getting around the fact that our chip has no hardware learning. Unfortunately, we were not able to easily adapt any of matlab's neuron net training functions to our network. We had hoped that we could train the network through some sort of stochastic model, or through genetic programming, however we were unable to implement the training due to time constraints. We are providing the simulation here for future use.

Notes: The simulation represents each two bit weight in the hardware as an integer ranging from 0 to 3. The most significant bit of the integer is the bit that is NANDed in hardware, the least significant bit is the bit that is XORed. Thus for an input a:

Weight	Output
0	1
1	0
2	~a
3	A

To create a network, use:

```
net = newbin(#inputs, layervector);
```

where layervector is of the form [#inputs #inputs #inputs …]

To create the network we defined in hardware, you call:

```
net = newbin(5,[5 5 5]);
```

To simulate, use:

```
results = sim(net, [in1_1 in1_2; in2_1 in 2_2; in3_1 in3_2; in4_1 in
4_2; in5_1 in5_2]);
```

The values separated by spaces are different trials, the values separated by semicolons are different inputs.

This simulation is built in the neural nets package in matlab. The weights for the first layer are stored in the matrix net.IW{1}. The weights for subsequent layers are stored in net.LW{layer, layer+1}. This is because these weights specify the weights connecting that layer to the next layer.

To use this simulation, place the following files in one directory and start matlab from that directory (unix).

File: ourweight.m

```
function output = ourweight(weights, inputs)
%produce an output using our special little weight function

if isstr(weights)
  switch(weights)
    case 'deriv'
      output = 'undefined';
    otherwise
      error('unrecognized code.')
  end
  return
end

[wrows wcols] = size(weights);
[irows icols] = size(inputs);
for row=1:wrows
  for col = 1:icols
    oredWeights = 0;
    for i=1:wcols
      switch(weights(row, i))
        case 0
          weightedOutput = 1;
        case 1;
          weightedOutput = 0;
        case 2;
          weightedOutput = not(inputs(i, col));
        case 3;
          weightedOutput = inputs(i,col);
        otherwise
          msg = sprintf('invalid weight: %f', weights(row,i))
          error(msg);
      end
      oredWeights = oredWeights | weightedOutput;
    end
    output(row,col) =  not(oredWeights);
  end
```

File:netnor.m

```
function n = netnor(varargin)

n = varargin{1};
if isstr(n)
  switch n
    case 'deriv',
      n = 'undefined';
    otherwise
      error('Unrecognized code.')
  end
  return
end

for i=2:length(varargin)
```

```
  n = n | varargin(i);
end
%n = not(n);
```

File: newbin.m

```
function net = newbin(numInputs, layers)
%create a binary neural network
%net = newbin(numInputs, layers)
%s layers is a 1 by n matrix where each element is the size of a layer
numLayers = length(layers)
if isa(layers,'cell') & (prod(size(layers)) == length(layers))
  layers = [layers{:}];
end

%structure
net = network(1,numLayers);
net.biasConnect = zeros(numLayers,1);
net.inputConnect(1,1) = 1;
[j,i] = meshgrid(1:numLayers,1:numLayers);
net.layerConnect = (j == (i-1));
net.outputConnect(numLayers) = 1;
net.targetConnect(numLayers) = 1;

%simulation
net.inputs{1}.range = repmat([0 1], numInputs,1);
for i=1:numLayers
  net.layers{i}.size = layers(i);
  net.layers{i}.transferFcn = 'purelin';
end

net.performFcn = 'mse';

%Adaption
%I dunno figure this out laters

%training
%neh

%Initialization
net.initFcn = 'initlay';
for i=1:numLayers
  net.layers{i}.initFcn = 'initwb';
  net.layers{i}.netInputFcn = 'netnor';
end
net.inputWeights{1,1}.initFcn='initzero';
net.inputWeights{1,1}.weightFcn='ourweight';
for i=2:numLayers
  net.layerWeights{i,i-1}.initFcn='initzero';
  net.layerWeights{i,i-1}.weightFcn='ourweight';
end
net = init(net);
```