CONVOLUTION ACCELERATION USING DYNAMIC HARDWARE

Designed/Presented by David Cain

Special thanks to Omar Eddash and Adam Frost for mentorship

Overview

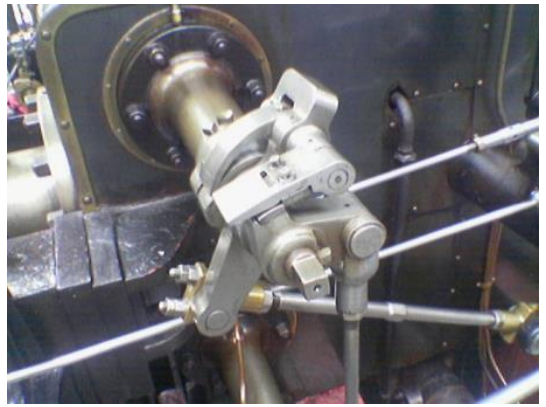
- Design Objective
- Project Introduction
  - Learn Verilog & Intro to Vivado
  - Enabling Partial Reconfig
- Design Components
  - Partially Reconfigurable Multipliers
  - Crossbar Data Switch
  - Partially Reconfigurable Adder
  - Matric Accelerator
  - Asynchronous FIFO
  - Matrix Controller
  - Design High Level Wrapper
- Design Resources
  - Floorplan
  - Resource Utilization
  - Power Report
- Conclusions & Future Work

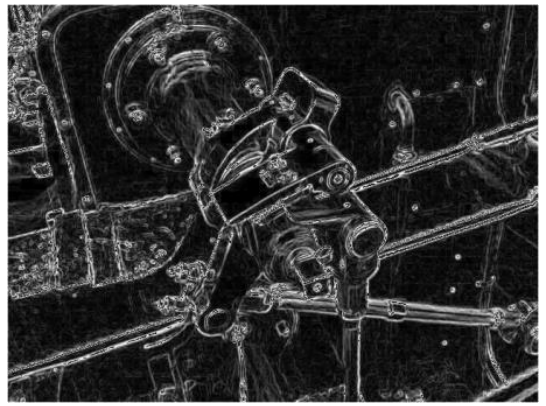
Design Objective

Will update this after looking over your slides

Develop an FPGA based reconfigurable design to accelerate matrix convolution

- This is a computationally intensive step used in many image processing algorithms
- The design must accept multiple data types
  - Integer data values
  - Floating point data values
  - Fixed point data values
- Each of these data types can be dynamically reconfigured on device





Design Objective

Project Introduction

- **Learn Verilog & Intro to Vivado**

  Enabling Partial Reconfiguration

Design Components

- Partially Reconfigurable Multipliers
- Crossbar Data Switch
- Partially Reconfigurable Adder
- Matric Accelerator
- Asynchronous FIFO
- Matrix Controller
- Design High Level Wrapper

Design Resources

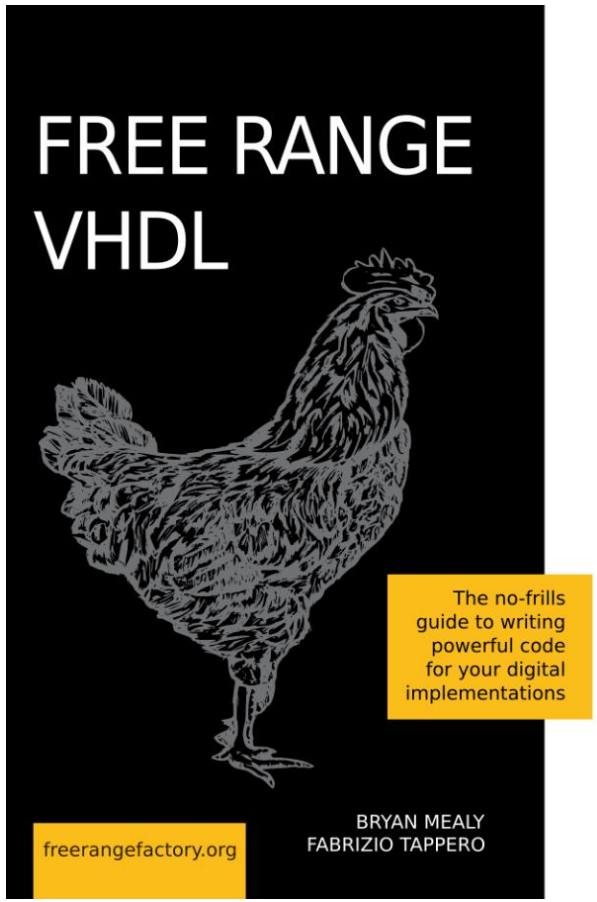
- Floorplan
- Resource Utilization
- Power Report

Conclusions & Future Work

Project Introduction:

Learn Verilog & Intro to Vivado

- Upon entering the project, I was referenced this book and several tutorials to get a feeling of Verilog
  - *Free Range VHDL*, Bryan Mealy, 2018

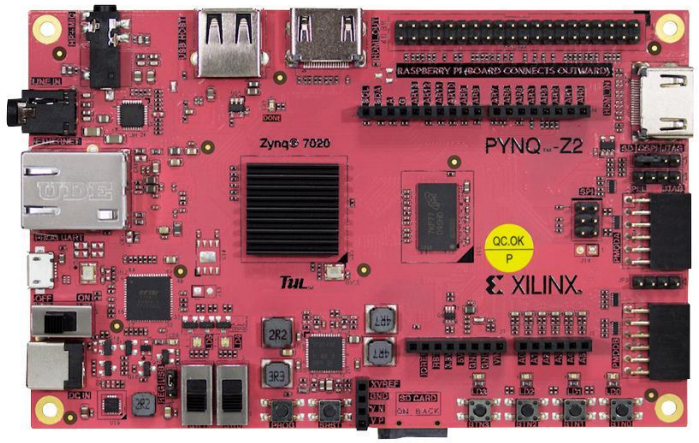


- Xilinx also has tutorials with pre-built designs to be imported and demonstrate concepts of how to use the hardware available

- YouTube tutorials were also helpful for workflow within Vivado

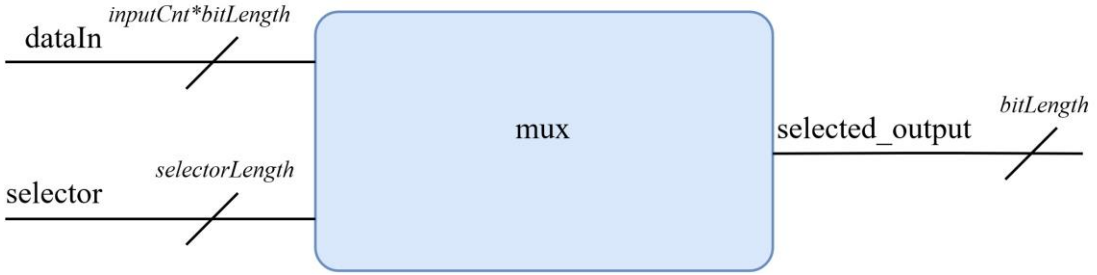
- Vivado is the primary software suite used for RTL design when implementing on Xilinx based FPGAs such as the Pynq-Z2 & Ultra96 V2

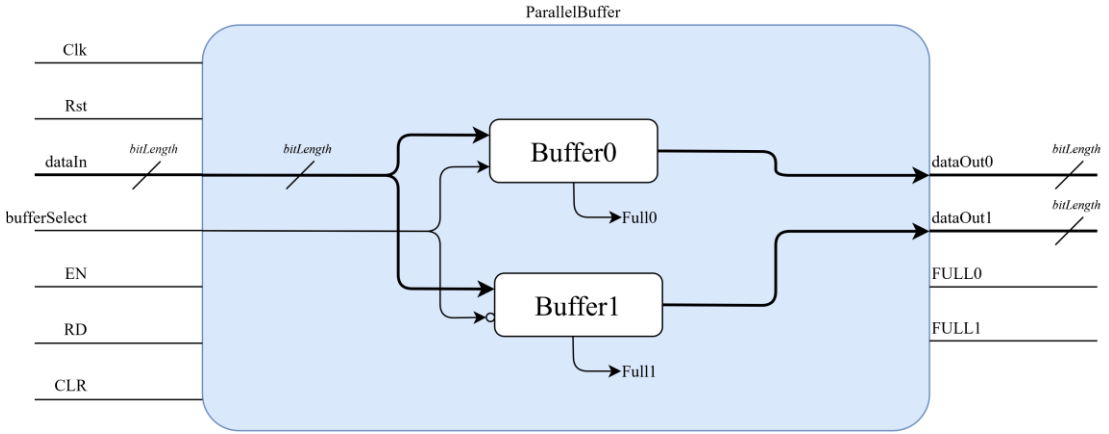
- Vivado offers multiple development forms:
  - HDL support for Verilog or VHDL
  - Built in IP Integrator using block diagrams



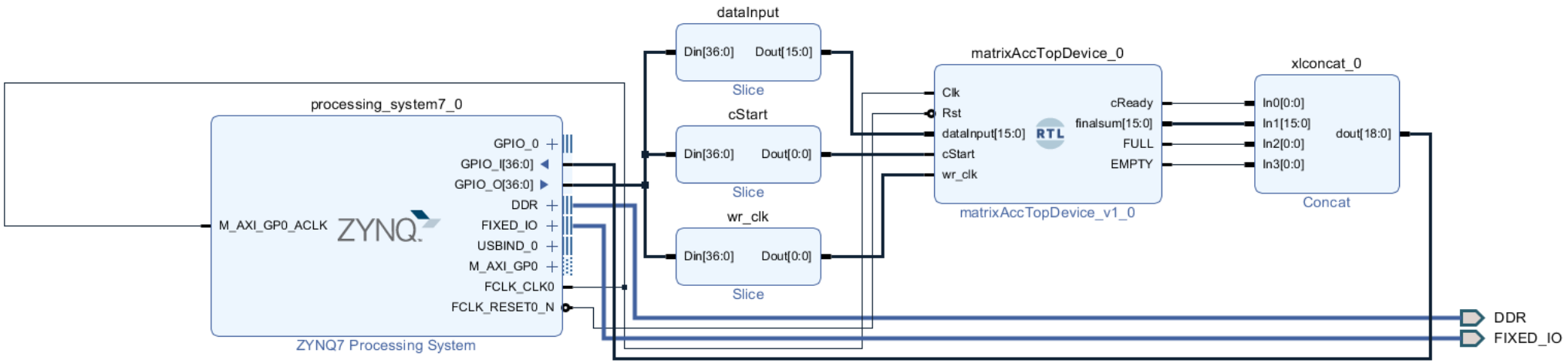
With a simple understanding of Verilog, basic designs were created and tested:

-Multibit port MUX
-Multibit flipflop register
-Parallel multibit flipflop
-Etc…





- Example of block diagram interface
  **For synthesis, block diagrams must be packaged with HDL wrappers generated by Vivado*



Design Objective

Project Introduction

- Learn Verilog & Intro to Vivado
- **Enabling Partial Reconfiguration**

Design Components

- Partially Reconfigurable Multipliers
- Crossbar Data Switch
- Partially Reconfigurable Adder
- Matric Accelerator
- Asynchronous FIFO
- Matrix Controller
- Design High Level Wrapper

Design Resources

- Floorplan
- Resource Utilization
- Power Report

Conclusions & Future Work

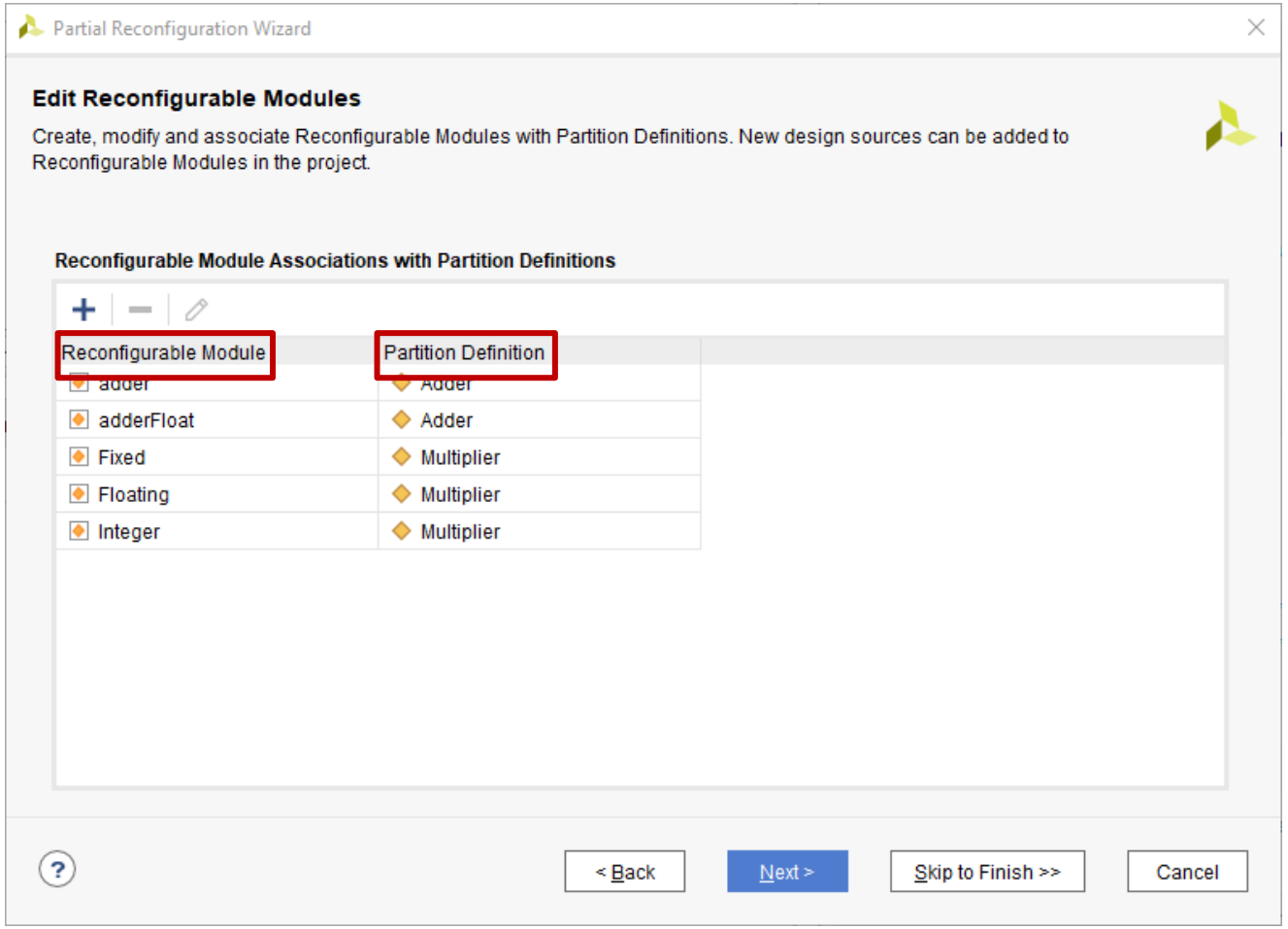
Project Introduction:

Enabling Partial Reconfiguration

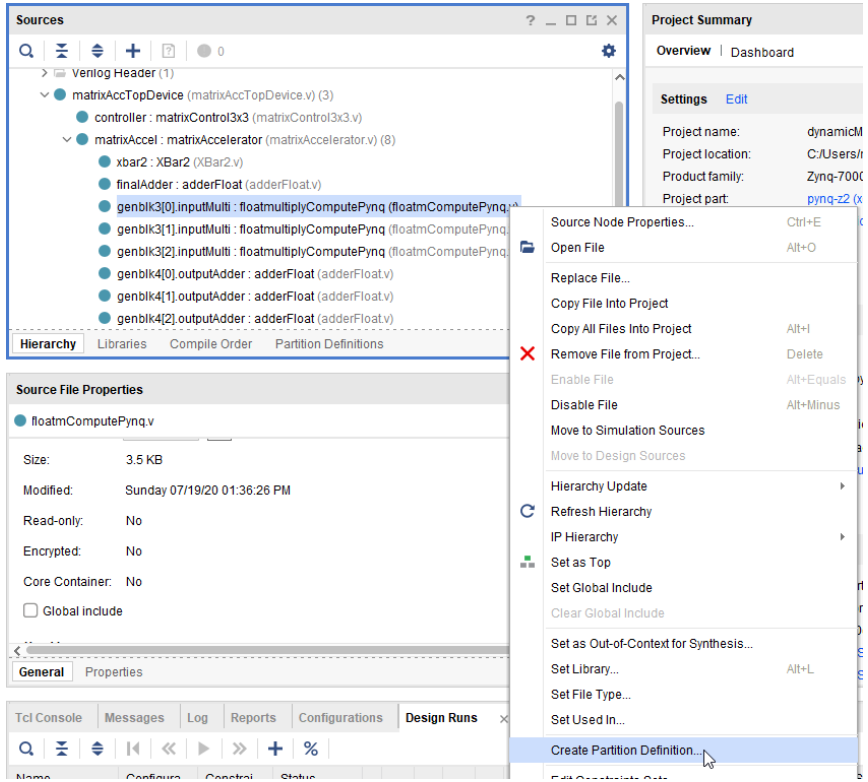
- Using Vivado's partial reconfiguration wizard, designs can become dynamically reconfigurable

- This allows for partitions of the FPGA to be reprogrammed during runtime

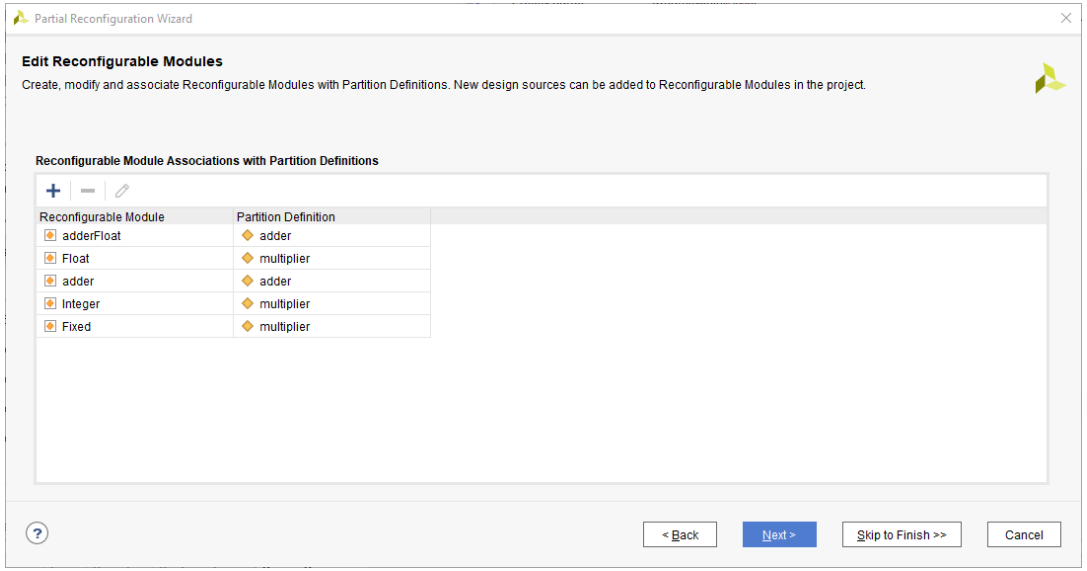
- When converting a project to partially reconfigurable, simulation is no longer available

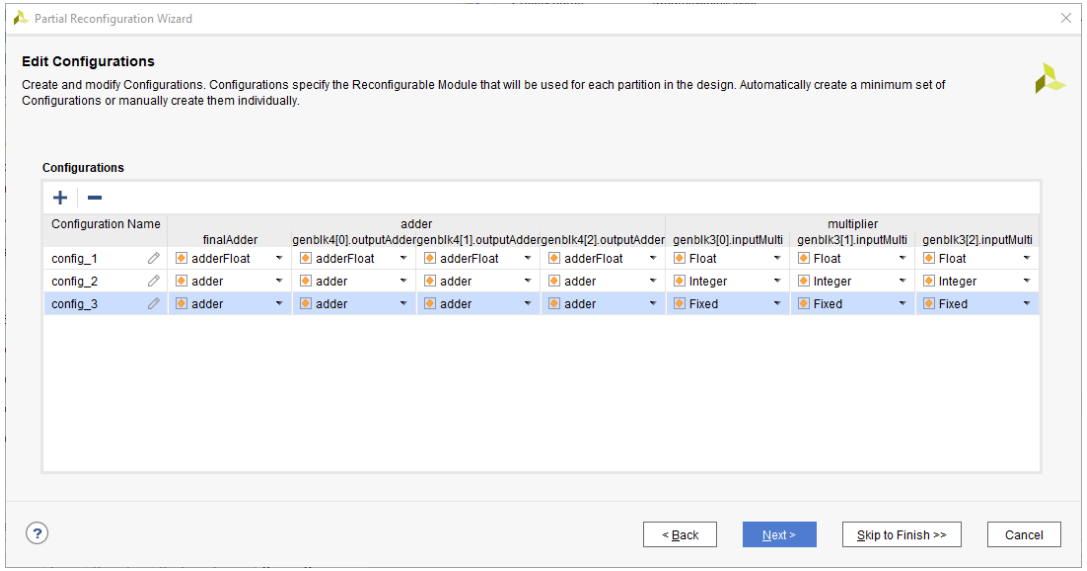


- Backup project
- Enable Partial Reconfiguration
  - Tools->Enable Partial Reconfiguration…
- Select a device in top to create a partition definition

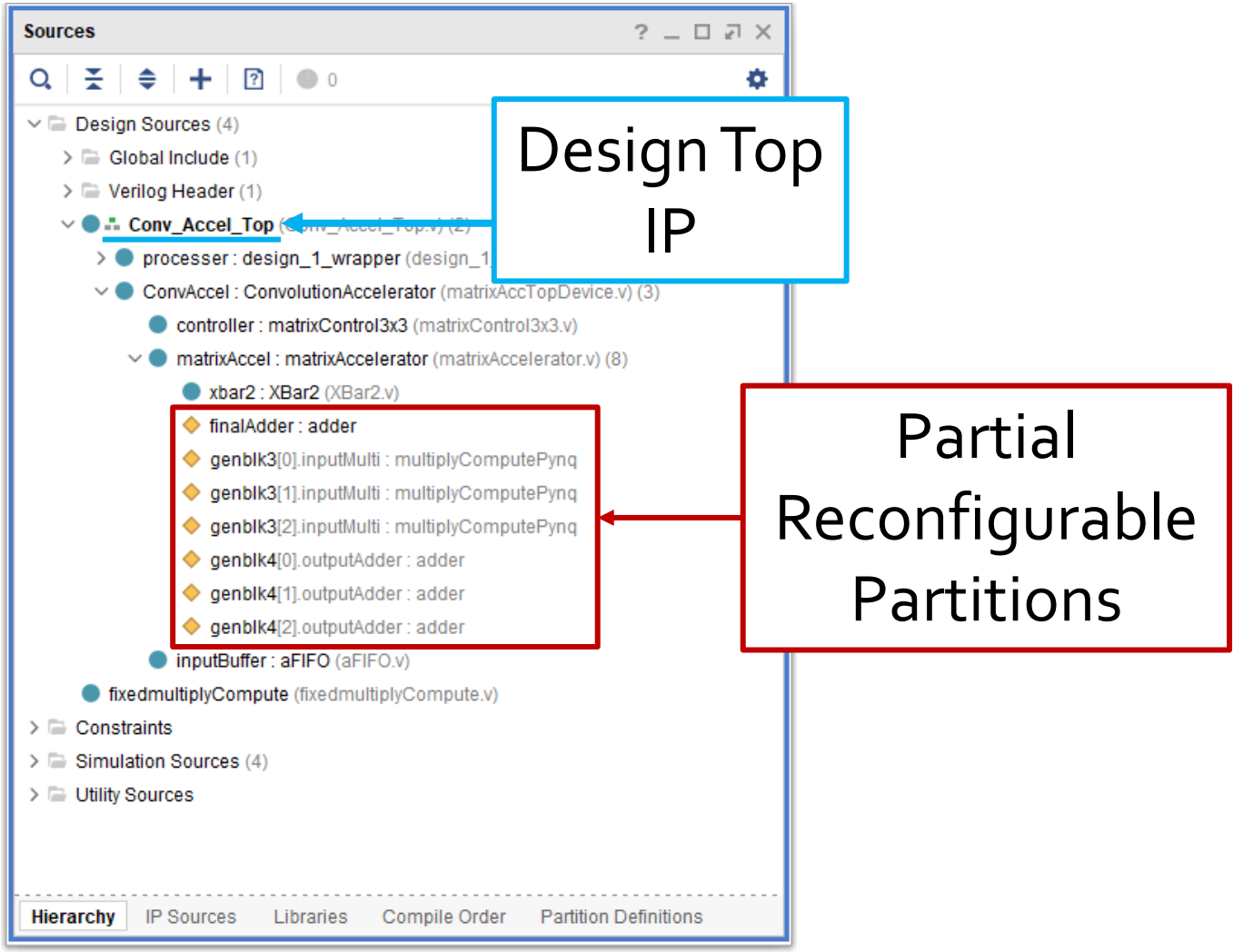


- Add any needed partitions and modules
- Create a configuration with modules set as needed



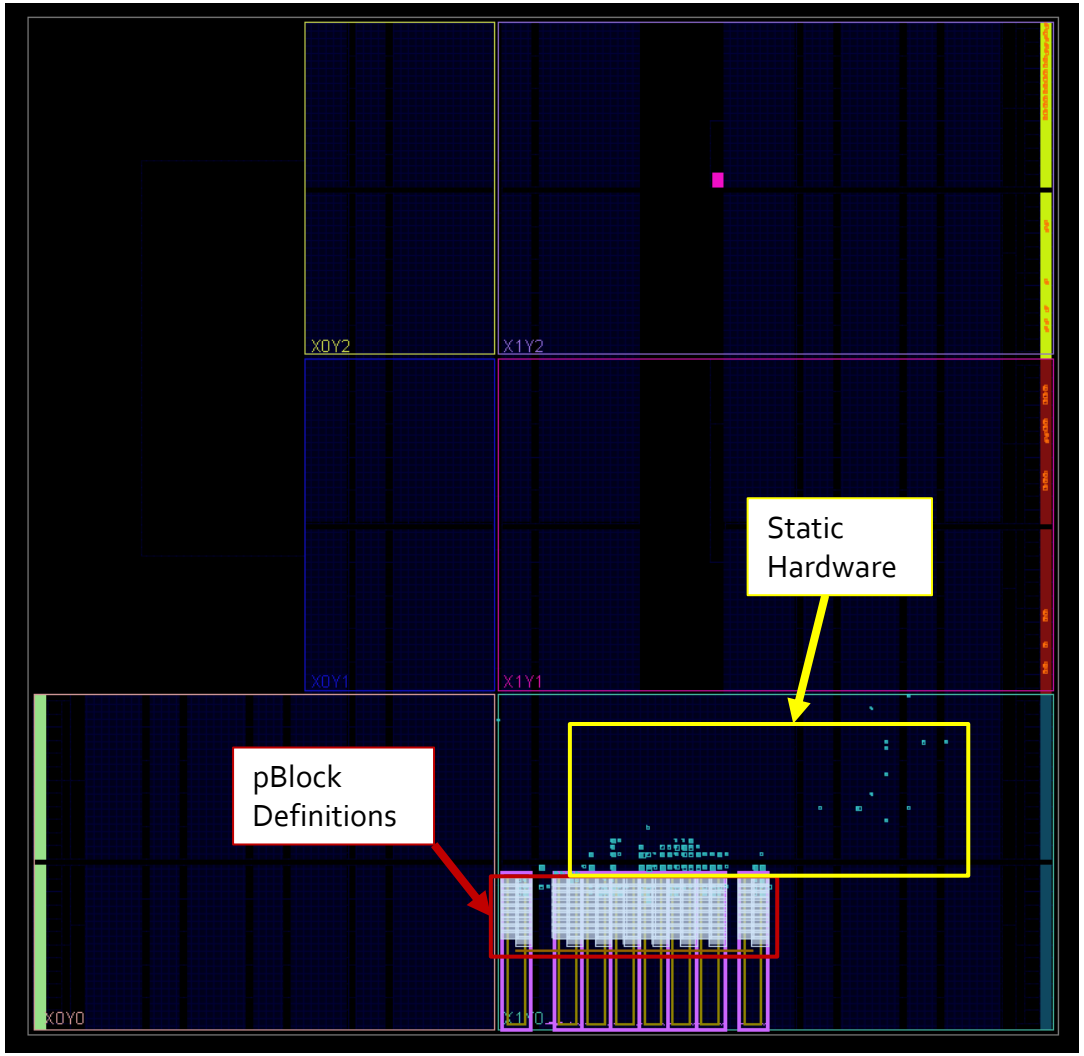


- Hierarchy view of design wrapped together with partial reconfiguration enabled



- Each reconfigurable partition requires a dedicated pBlock to dictate the hardware available for reconfiguration

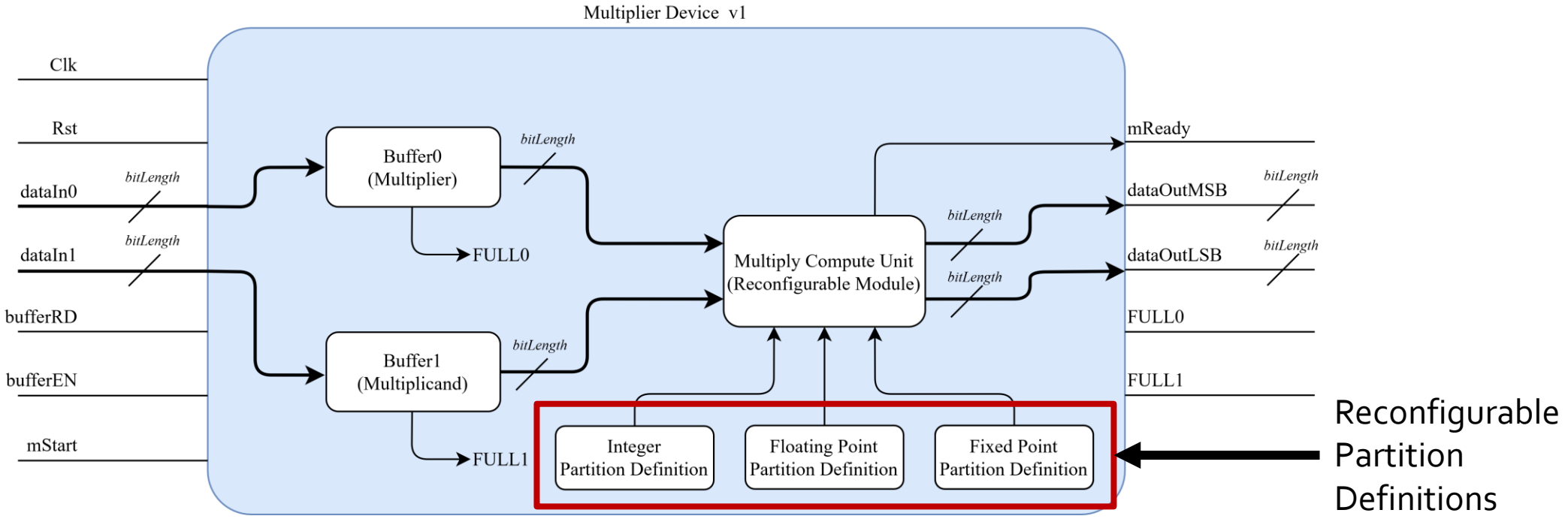
- This is a screenshot of the floorplan for a design with 8 pBlocks to enable 8 separate reconfigurable multiply compute partitions.



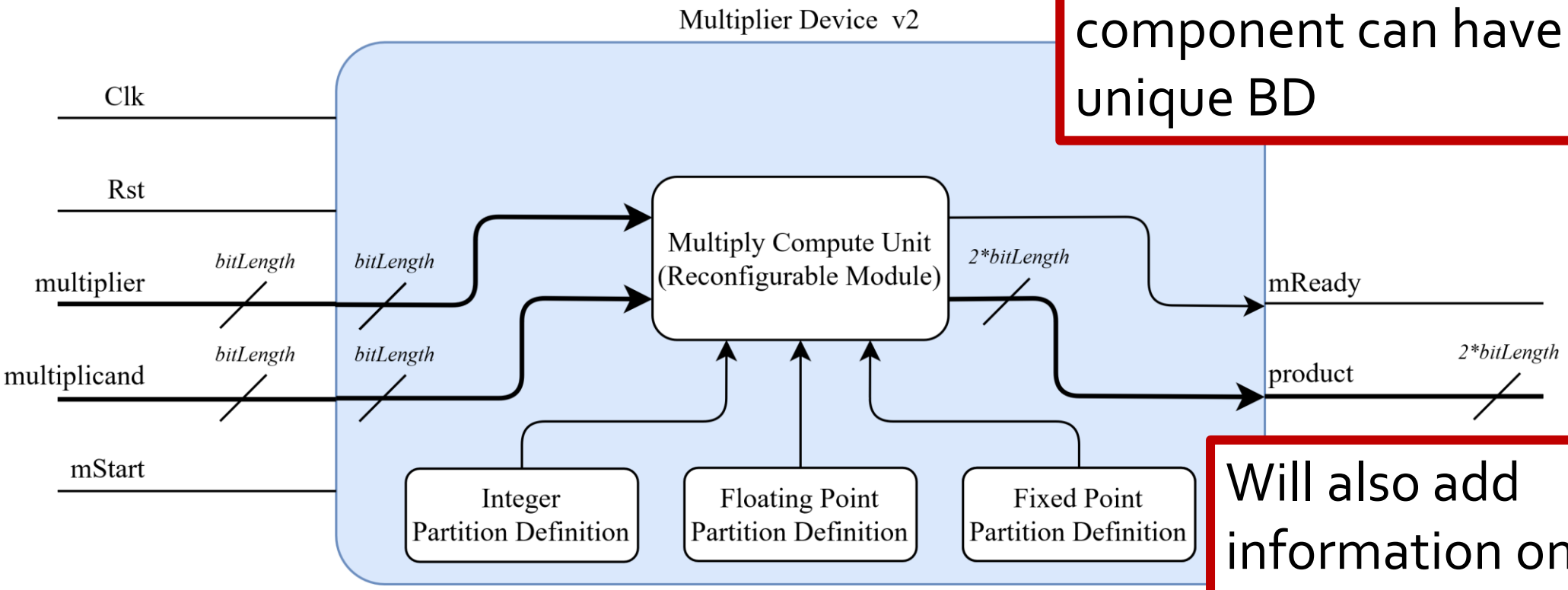
Design Components

Partially Reconfigurable Multipliers

- A first step was to implement devices that could intake 2 data values, multiply them, then return the product
- Decided to change layout later in development
- Block diagram of multiplier v1



- Block diagram of multiplier v2, updated design in use



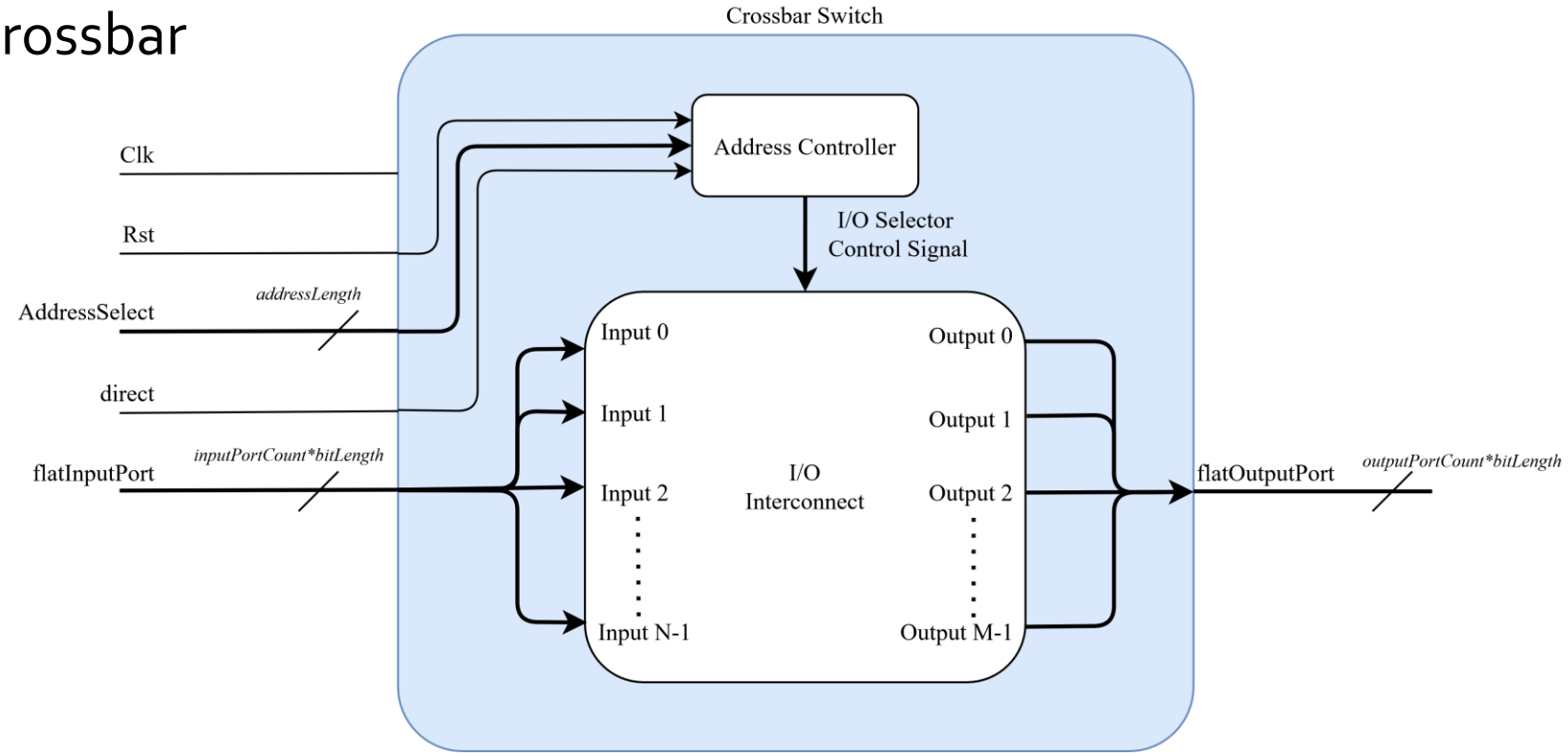
Will try to create a new block diagram so the component can have unique BD

Will also add information on DSP implementation of multipliers

Design Components

Crossbar Data Switch

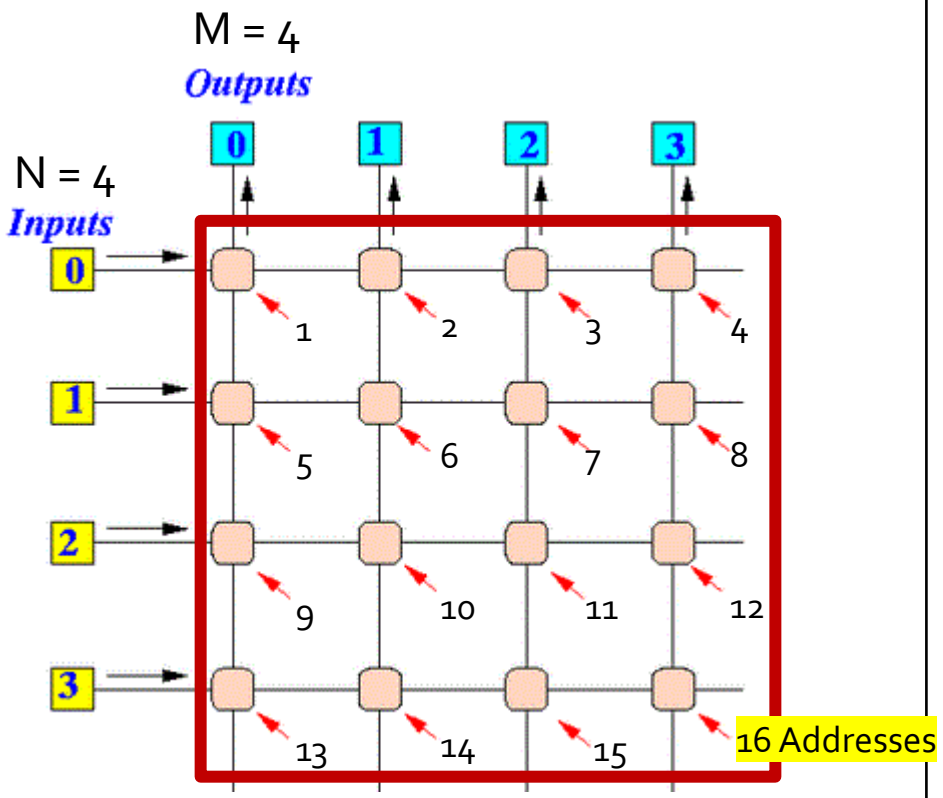
- A crossbar switch was designed and implemented to handle product outputs
- Basis for a crossbar switch is to allow any input port to be connected to any output port
- Block diagram of crossbar



- The crossbar operates on a grid connection concept. With N inputs and M outputs, this generates NxM valid address selections

- At positive edge clocks, the address on AddressSelect port is toggled

- If direct is HIGH, equal I/O ports will be connected.
  - i.e. Output0 = Input0; Input1 = Output1;…

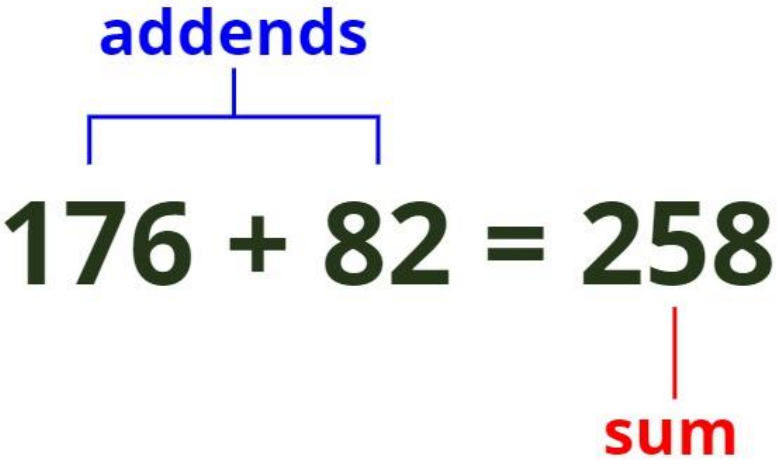


Design Components

Partially Reconfigurable Adder

- With a method to now compute many products in parallel and sort them, a device was needed to accumulate the sum of products

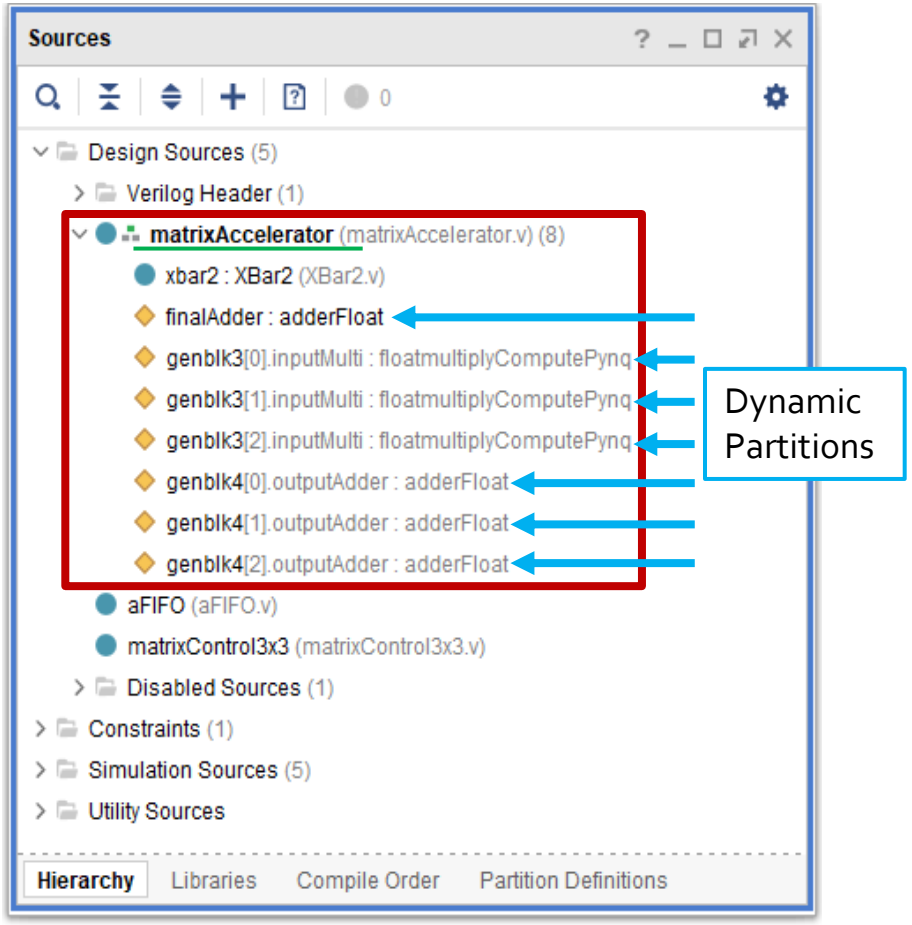
- 2 Partition definitions were created:
  - Integer/Fixed Point Adder
  - Floating Point Adder



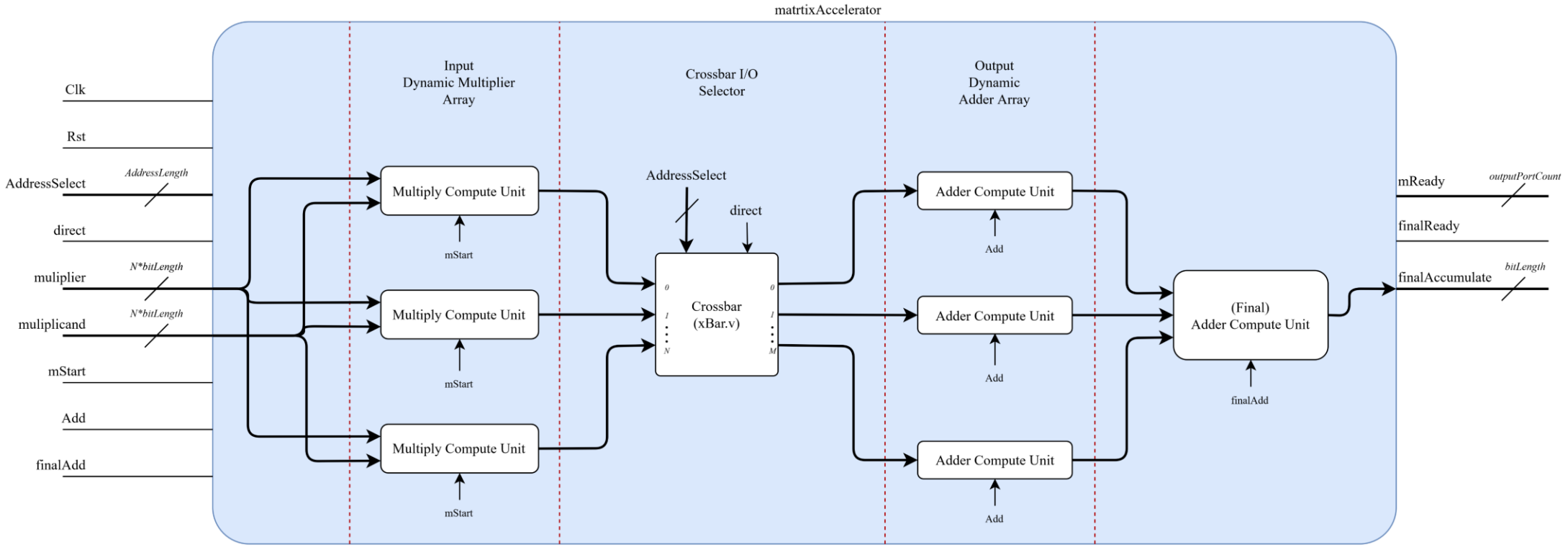
Matrix Accelerator

- These components were then packaged into a single IP
  - This device has a lot of I/O and no built-in control logic
  - This means it must be packaged with a controller IP for operation

Hierarchy View



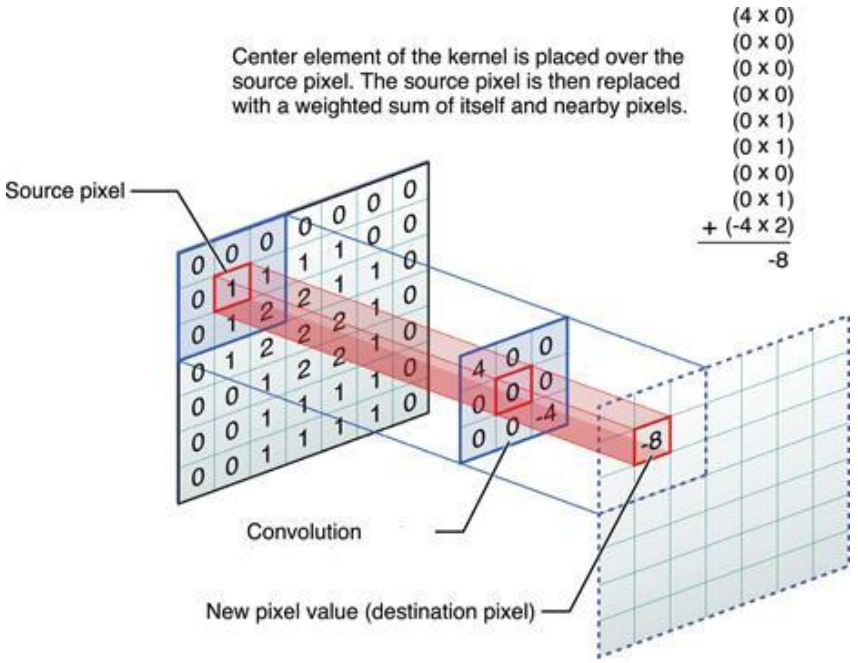
Block diagram of matrix accelerator



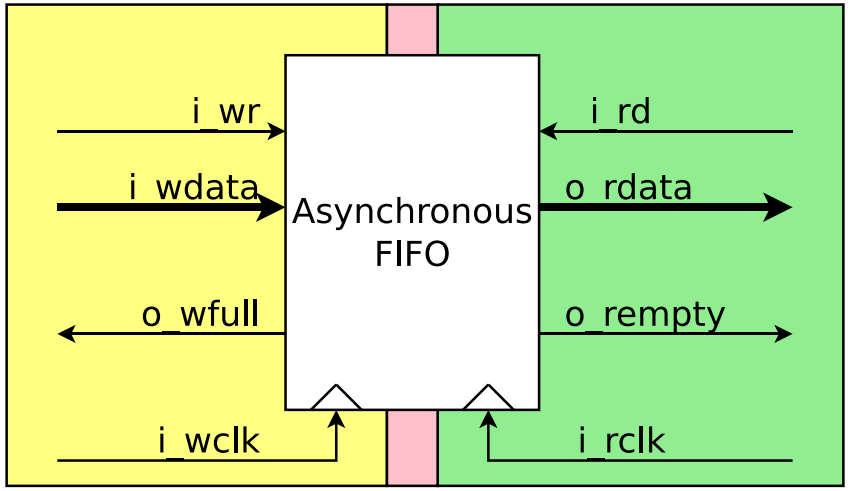
Design Components

Asynchronous FIFO

- The convolution accelerator needs much data input
  - Data Set (Image Values)
  - Filter Set (Filter Values)

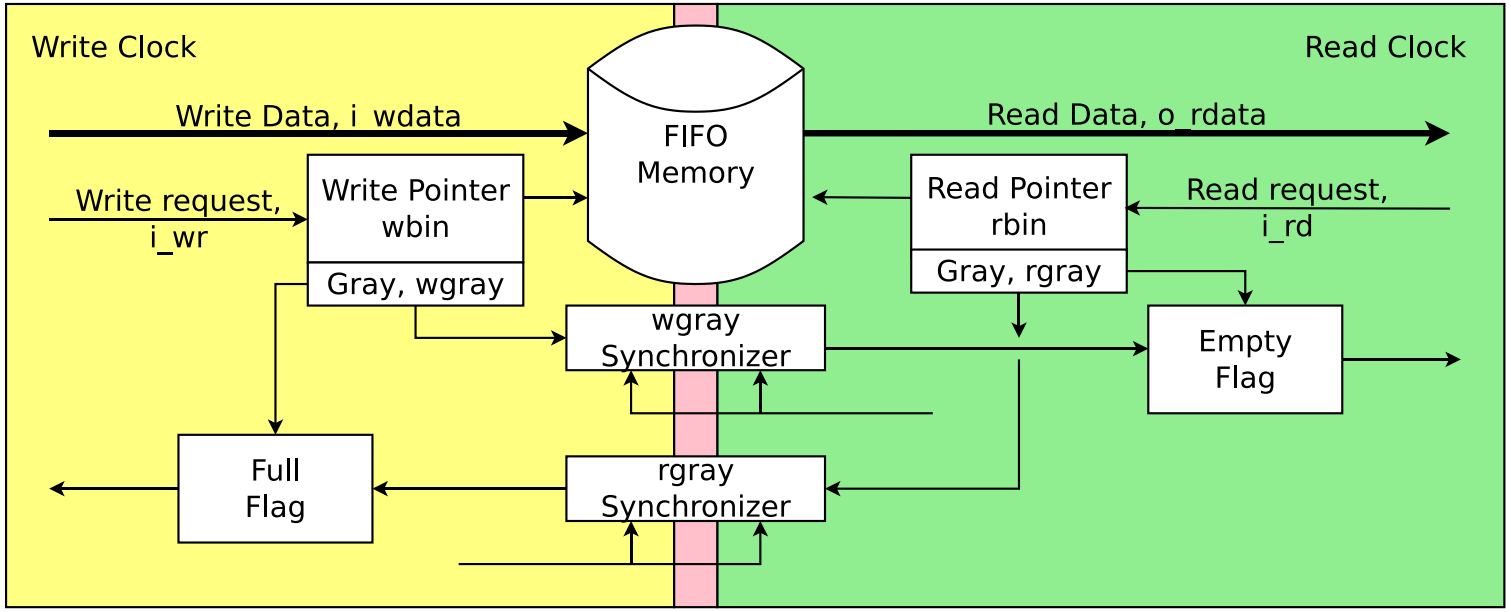


- Since the program will not be able to load values and operate the FPGA properly, an asynchronous FIFO buffer was used to input a data set across the clock domains



- There was a helpful online project:
  - *Crossing Clock Domains with an Asynchronous FIFO*, Dan Gisselquist, 2018
  - https://zipcpu.com/blog/2018/07/06/afifo.html

- I attempted to write this IP from scratch, but the problem was difficult
- This write-up by Gisselquist was an extensive overview of the device problems, behavior, and similar work

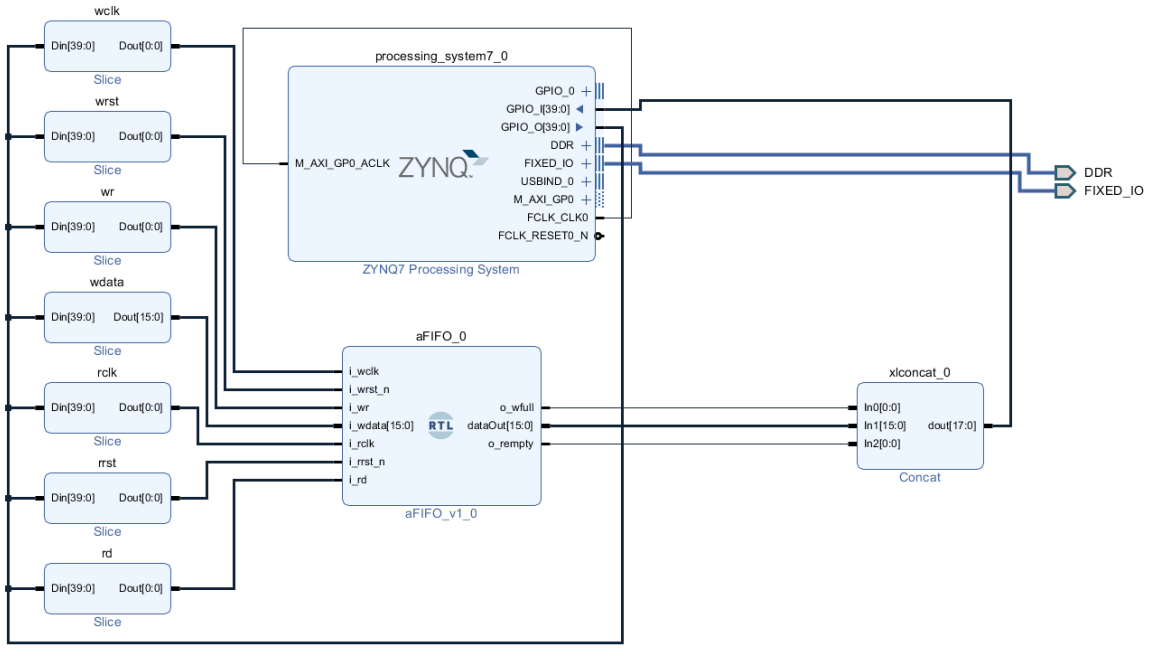


- To verify this FIFO design was suitable, it was first tested in Vivado simulation

- In the Python environment, the buffer would be filled with a random data set

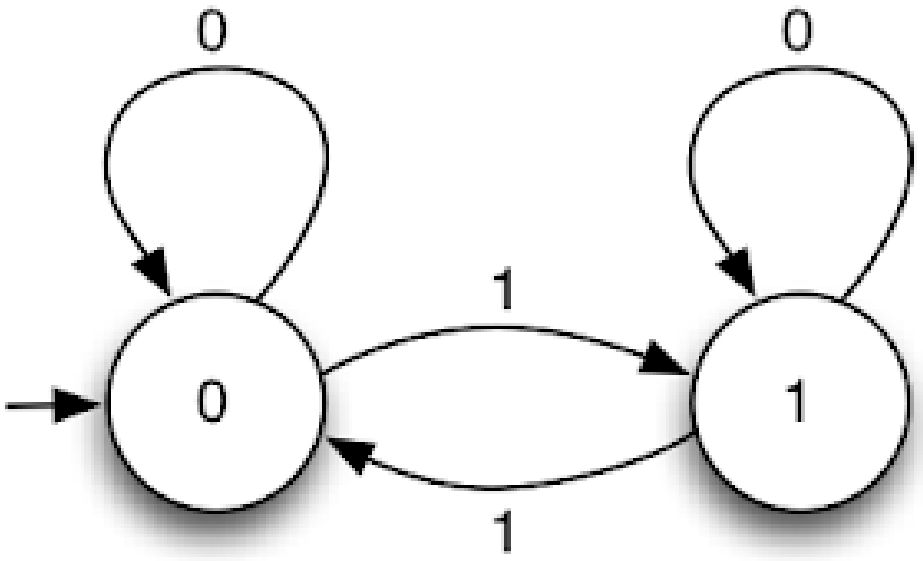
- Next, the buffer data would be extracted and compared with original input

- The load/compare times were also tracked



Matrix Controller

- The controller needs to:
  - Read data in the input buffer
  - Pipeline data into the matrix accelerator
  - Use matrix accelerator control signals to generate convolution sum

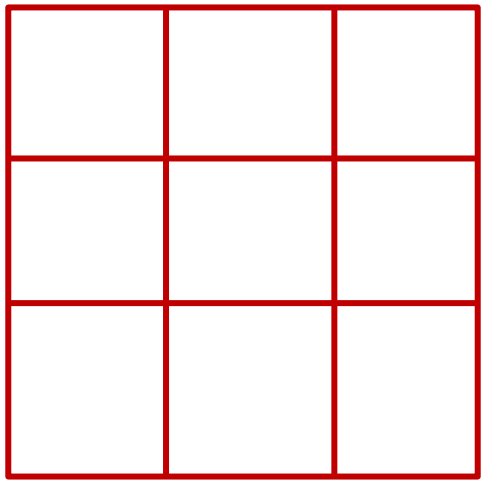


- The controller has three main states:
  - Reading data from the buffer
  - Multiplying the input values
  - Add the resulting products

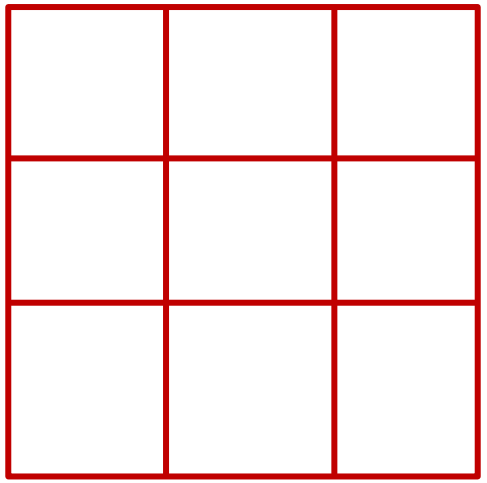
```
In [65]: testCnt = 1000
         print("Running %d tests" %testCnt)
         start_time = time.time()
         for i in range(testCnt):
             #print("\nRunning test",(i+1))
             result = testConvolution(0)
             if(result!=1):
                 print("**** Test fail at",i)
                 break
             if(i == (testCnt-1)):
                 stop_time = time.time()
                 print("**** All test passed")

         print("Total runtime(ms):",round((stop_time-start_time)*1000,2))

         Running 1000 tests
         **** All test passed
         Total runtime(ms): 140412.16
```

- Buffers are used to temporarily hold input values

Data Input Buffer



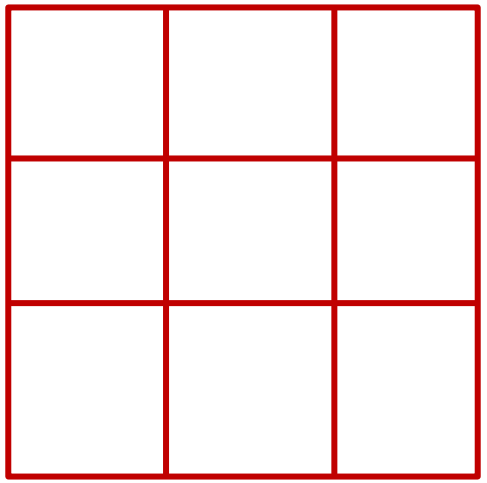
Filter Kernel Buffer



- Buffers are used to temporarily hold input values

- The first full input set is stored as the filter

Data Input Buffer



Filter Kernel Buffer

1	4	7
2	5	8
3	6	9

- Buffers are used to temporarily hold input values

- The first full input set is stored as the filter
- The next full input set is temporarily stored as the data to be operated on

Data Input Buffer

1	4	7
2	5	8
3	6	9

Filter Kernel Buffer

1	4	7
2	5	8
3	6	9

- Buffers are used to temporarily hold input values

- The first full input set is stored as the filter
- The next full input set is temporarily stored as the data to be operated on
- After the convolution is completed, the data set is shifted to the left

Data Input Buffer

4	7	
5	8	
6	9	

Filter Kernel Buffer

1	4	7
2	5	8
3	6	9

- Buffers are used to temporarily hold input values

- The first full input set is stored as the filter
- The next full input set is temporarily stored as the data to be operated on
- After the convolution is completed, the data set is shifted to the left
- The next column of data fills the empty buffers, acting as a slide on the original feature set
- The next convolution can now be computed

Data Input Buffer

4	7	10
5	8	11
6	9	12

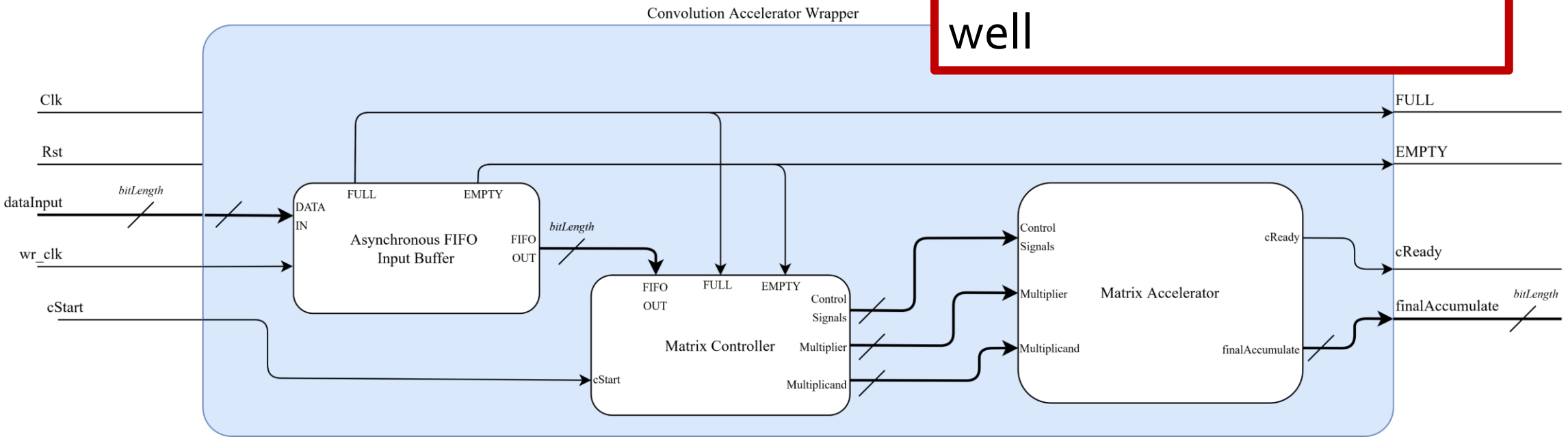
Filter Kernel Buffer

1	4	7
2	5	8
3	6	9

Design High Level Wrapper

- A top-level package was created to wrap the input buffer, matrix accelerator, and controller into one design
- Convolution Accelerator block diagram

Need to update this BD as output is not buffered as well

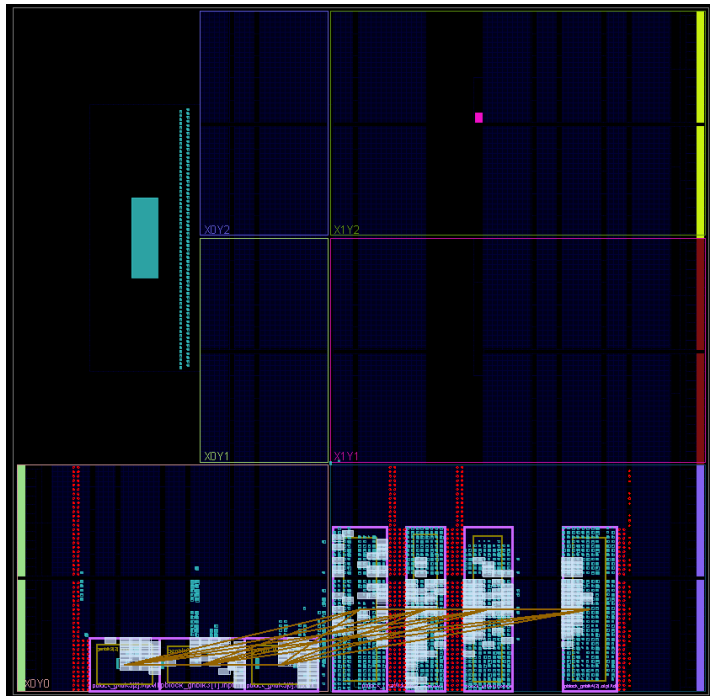


- Design Objective
- Project Introduction
  - Learn Verilog & Intro to Vivado
  - Enabling Partial Reconfiguration
- Design Components
  - Partially Reconfigurable Multipliers
  - Crossbar Data Switch
  - Partially Reconfigurable Adder
  - Matric Accelerator
  - Asynchronous FIFO
  - Matrix Controller
  - Design High Level Wrapper
- **Design Resources**
  - **Floorplan**
  - Resource Utilization
  - Power Report
- Conclusions & Future Work

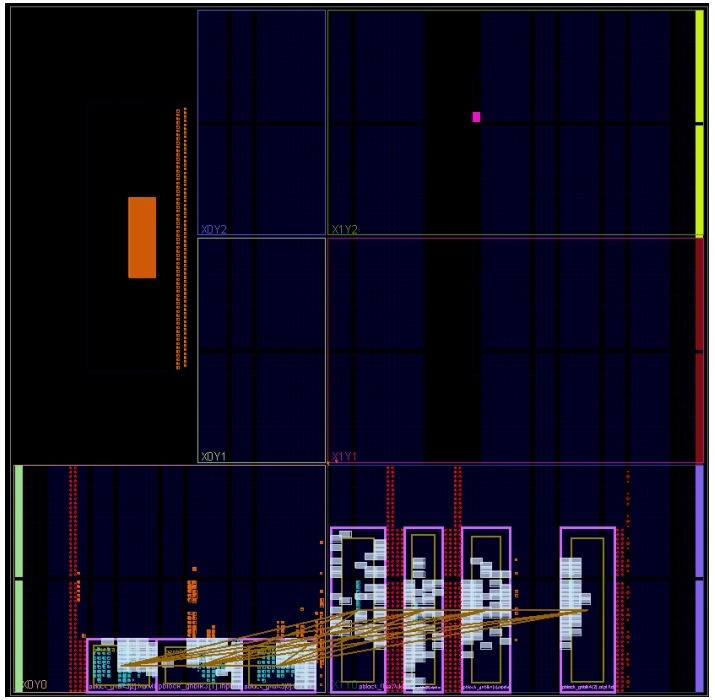
Design Resources

Floorplan

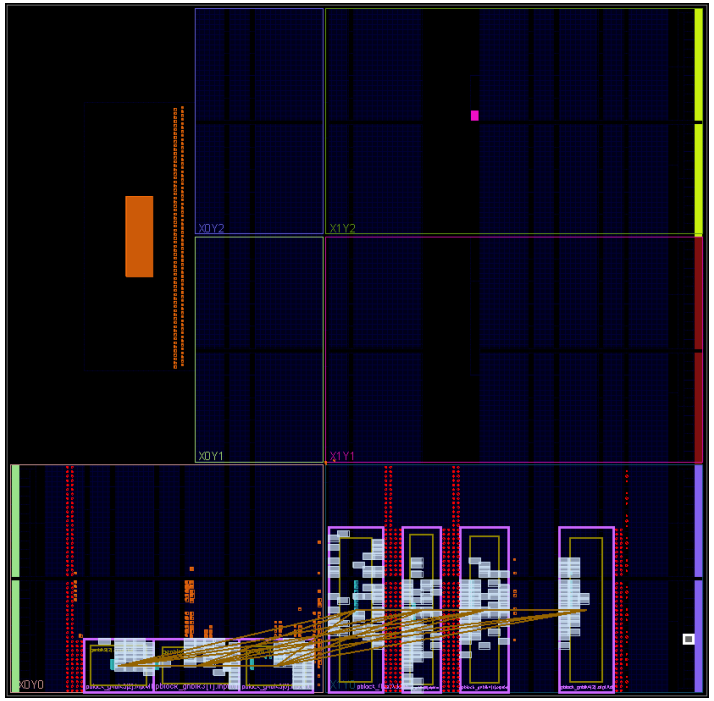
Floating Point Configuration Floorplan



Fixed Point Configuration Floorplan



Integer Configuration Floorplan



Resource Utilization

Floating Point Configuration Resource Utilization

!!!!

Name	Slice LUTs (51040)	Slice Registers (102080)	Slice (12760)	LUT as Logic (51040)	LUT as Memory (16864)	DSPs (220)	BUFGCTRL (32)
matrixAccTopDevice	5199	459	1386	5175	24	3	3
controller (matrixControl3x3)	120	117	33	120	0	0	0
inputBuffer (aFIFO)	59	64	17	35	24	0	0
matrixAccel (matrixAccelerator)	5020	278	1336	5020	0	3	0
processer (design_1_wrapper)	1	0	1	1	0	0	2

Fixed Point Configuration Resource Utilization

Name	Slice LUTs (51040)	Slice Registers (102080)	Slice (12760)	LUT as Logic (51040)	LUT as Memory (16864)	BUFGCTRL (32)
matrixAccTopDevice	1177	483	369	1153	24	3
controller (matrixControl3x3)	120	117	33	120	0	0
inputBuffer (aFIFO)	59	64	17	35	24	0
matrixAccel (matrixAccelerator)	998	302	319	998	0	0
processer (design_1_wrapper)	1	0	1	1	0	2

Integer Configuration Resource Utilization

Name	Slice LUTs (51040)	Slice Registers (102080)	Slice (12760)	LUT as Logic (51040)	LUT as Memory (16864)	DSPs (220)	BUFGCTRL (32)
matrixAccTopDevice	381	411	133	357	24	3	3
controller (matrixControl3x3)	120	117	33	120	0	0	0
inputBuffer (aFIFO)	59	64	17	35	24	0	0
matrixAccel (matrixAccelerator)	202	230	83	202	0	3	0
processer (design_1_wrapper)	1	0	1	1	0	0	2

- Design Objective
- Project Introduction
  - Learn Verilog & Intro to Vivado
  - Enabling Partial Reconfiguration
- Design Components
  - Partially Reconfigurable Multipliers
  - Crossbar Data Switch
  - Partially Reconfigurable Adder
  - Matric Accelerator
  - Asynchronous FIFO
  - Matrix Controller
  - Design High Level Wrapper
- **Design Resources**
  - Floorplan
  - Resource Utilization
  - **Power Report**
- Conclusions & Future Work

Design Resources

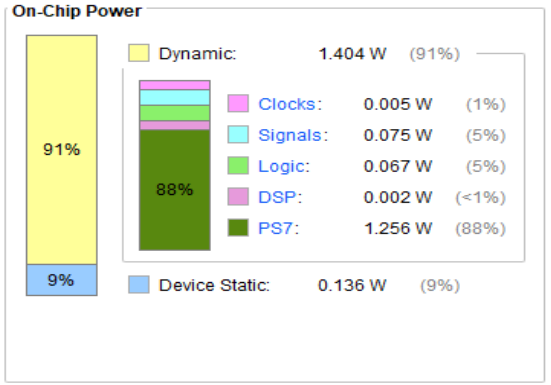
Power Report

Floating Point Configuration Power Report

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.541 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**42.8°C**
Thermal Margin:	42.2°C (3.5 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

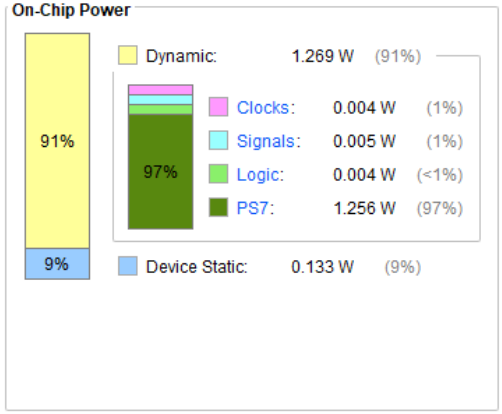


Fixed Point Configuration Power Report

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.402 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**41.2°C**
Thermal Margin:	43.8°C (3.7 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

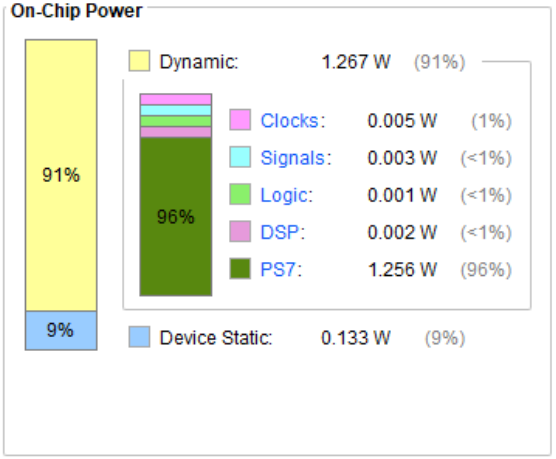


Integer Configuration Power Report

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.4 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**41.1°C**
Thermal Margin:	43.9°C (3.7 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



Conclusions & Future Work

Conclusions

- While much of the development process for RTL has streamlined by Xilinx, many times development and debugging could be quite tedious
- Currently software to control the FPGA is python based and seems to result in long runtimes for simple computation
- Floating point utilizes much more resources compared to integer or fixed point

Future Work

- New software for loading data sets will likely be written in Xilinx's SDK suite
- Floating point needs to be overlooked due to failing synthesis with timing issues
- Begin generating feature maps

Will likely add more here

Thank You