Dynamic Hardware Matrix Acceleration Design Reflection

Designed/Presented by David Cain

Special thanks to Omar Eddash and Adam Frost for mentorship

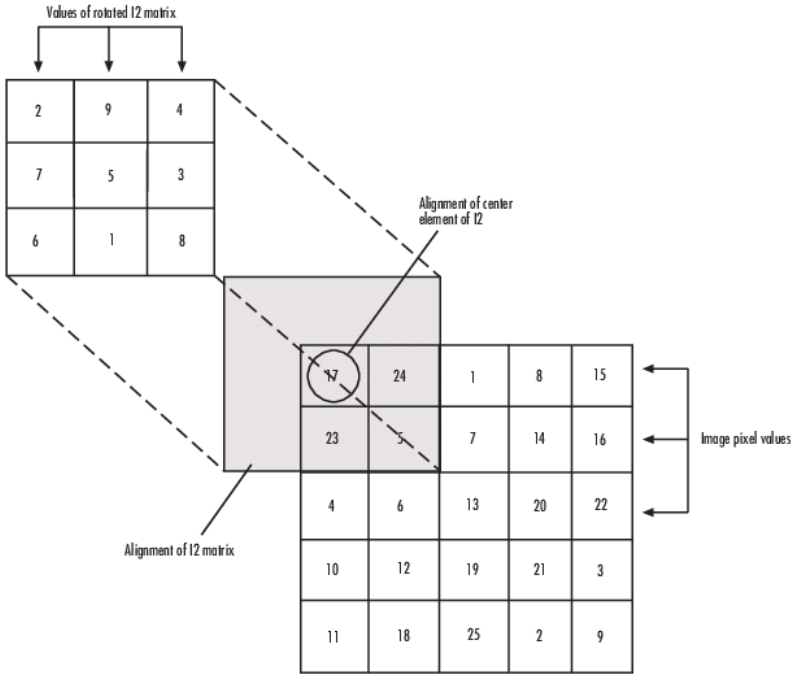
Design Objective

Develop an FPGA based reconfigurable design to accelerate matrix convolution

- This is a computationally intensive process used in many image processing algorithms that can benefit from hardware acceleration

- 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



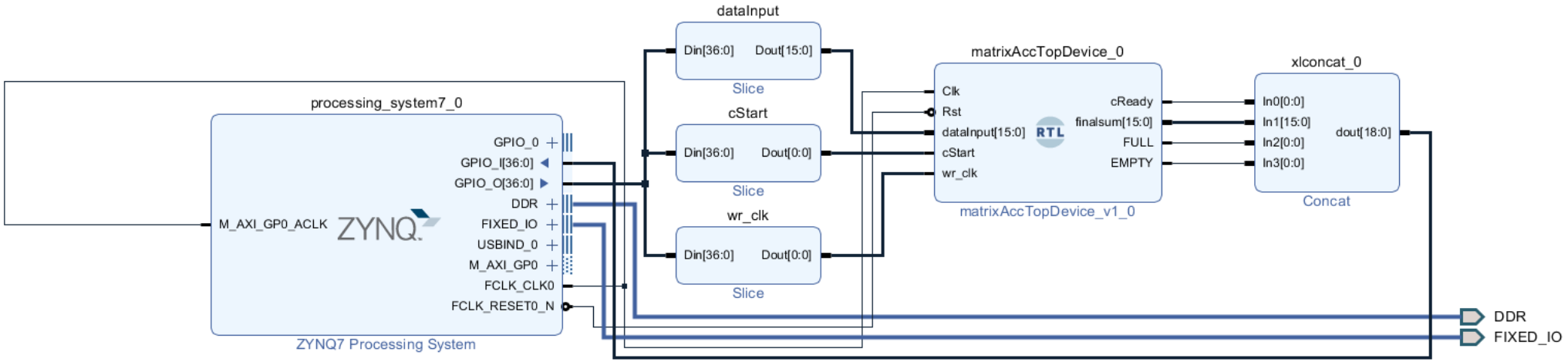
Intro to Vivado

- Vivado is the primary software suite used when working with Xilinx based FPGAs such as the Pynq-Z2
- Vivado offers design development using HDL or the built in IP Integrator using block diagrams
  - These block diagrams can compose of either default IPs provided by Xilinx or user custom IPs that are imported/created within the project
  - The HDL defining these Xilinx based IPs are not viewable by end-user
- Often both development forms are used when working on a design

Intro to Vivado (cont.)

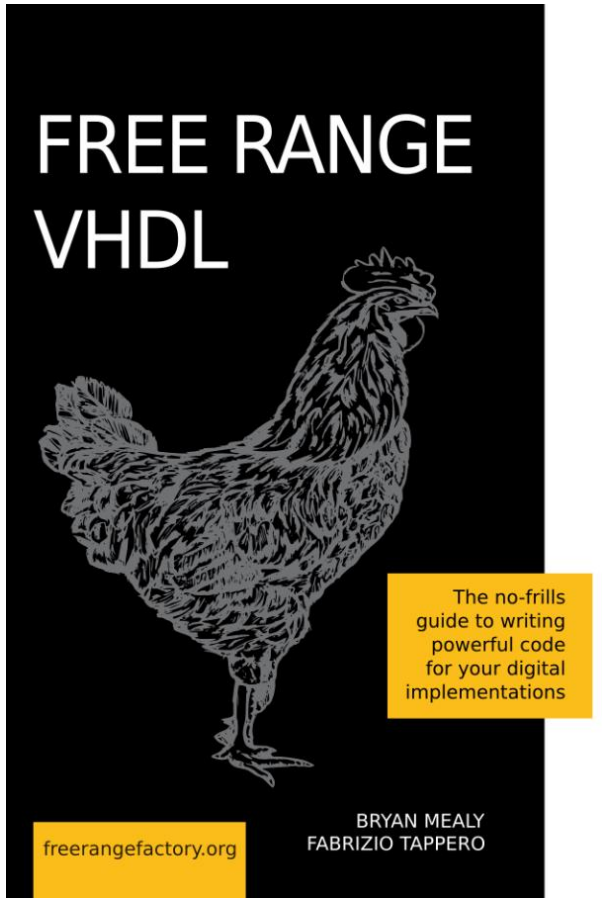
- Example of block diagram interface

  **For synthesis, block diagrams must be packaged with HDL wrappers generated by Vivado*



Design Step 1: Learn Verilog

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

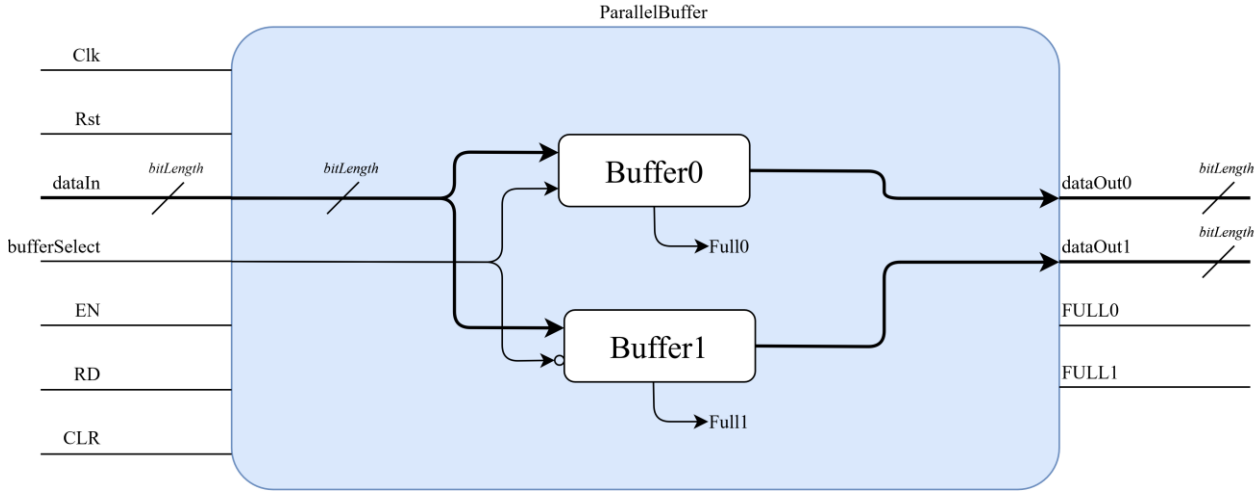


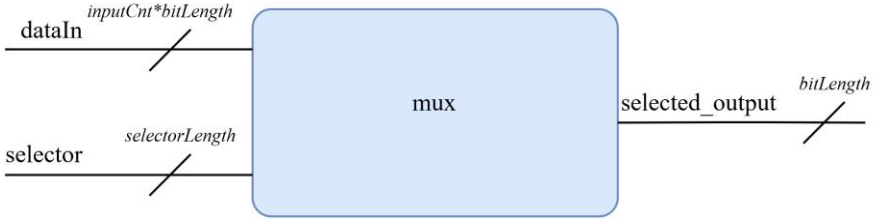
- Xilinx also has tutorials with pre-built designs to be imported and demonstrate concepts of how to use the hardware available

- YouTube tutorials were also helpful for workflow within Vivado

Design Step 1: Learn Verilog (cont.)

- With a simple understanding of Verilog, basic designs were created and tested to verify understanding:
  1. Multibit flipflop register (FlipFlop.v)
  2. Parallel multibit flipflop (ParallelBuffer.v)
  3. Multibit port MUX (mux.v)
  4. Variable data slicer (dataSplit.v)





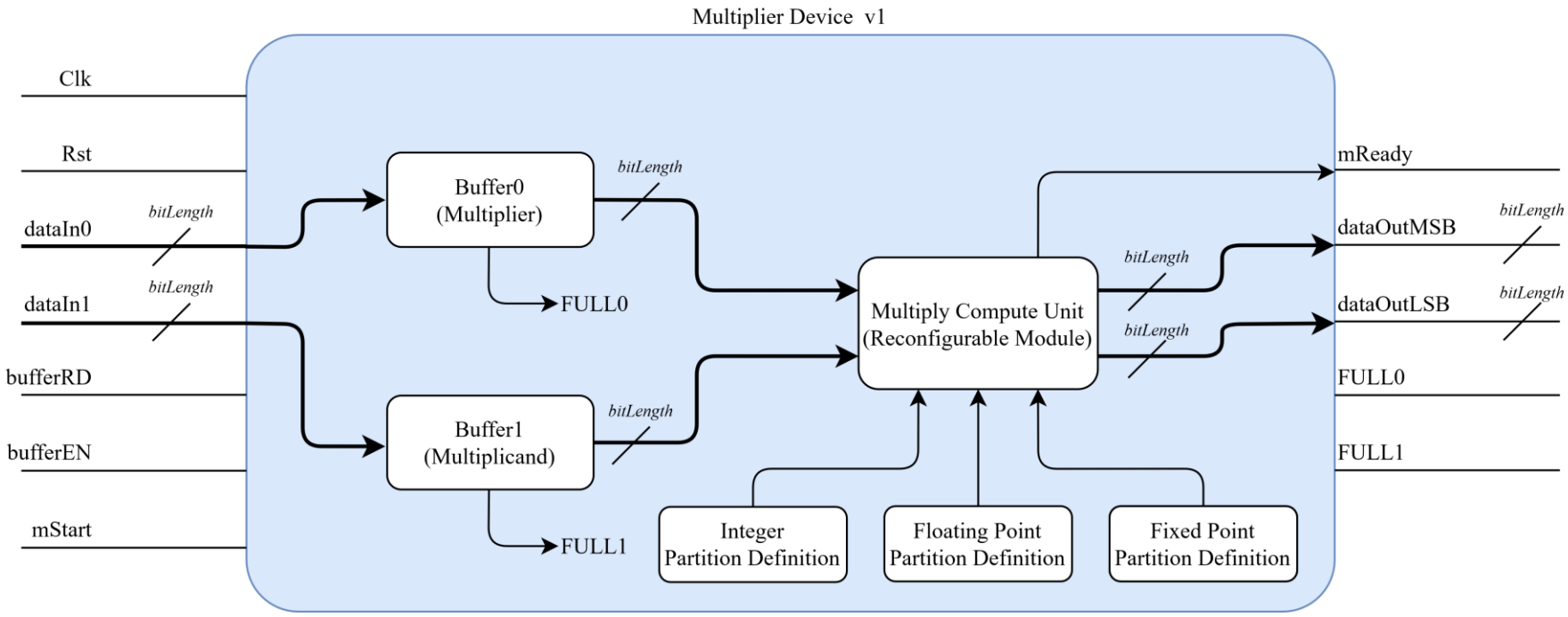
Design Step 2: Partial Reconfig Multipliers

- A first step was to implement devices that could intake 2 data values, multiply them, then return the product
  - This needed to be designed for three separate data types, with separate algorithms to complete each computation type

- This was initially designed to buffer input data internally, allowing for parallel computations. The buffered input was removed due to adding unnecessary delay while pipelining data streams

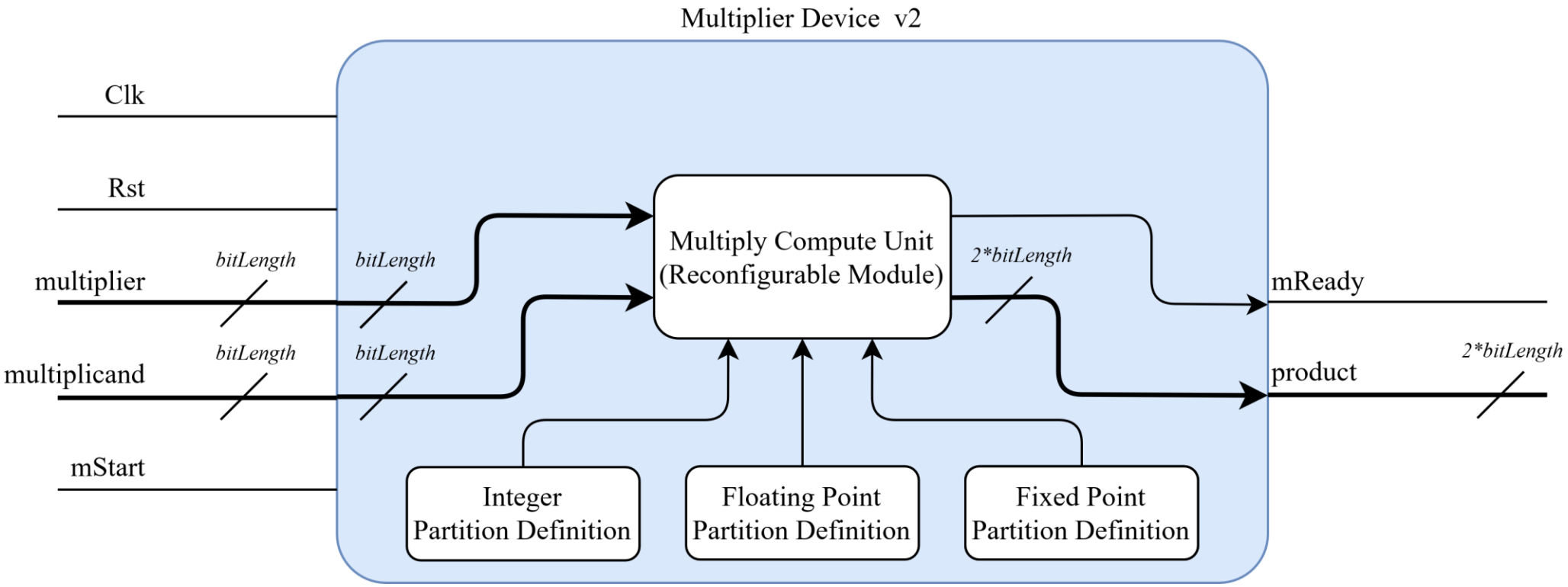
Design Step 2: Partial Reconfig Multipliers (cont.)

- Block diagram of multiplier v1



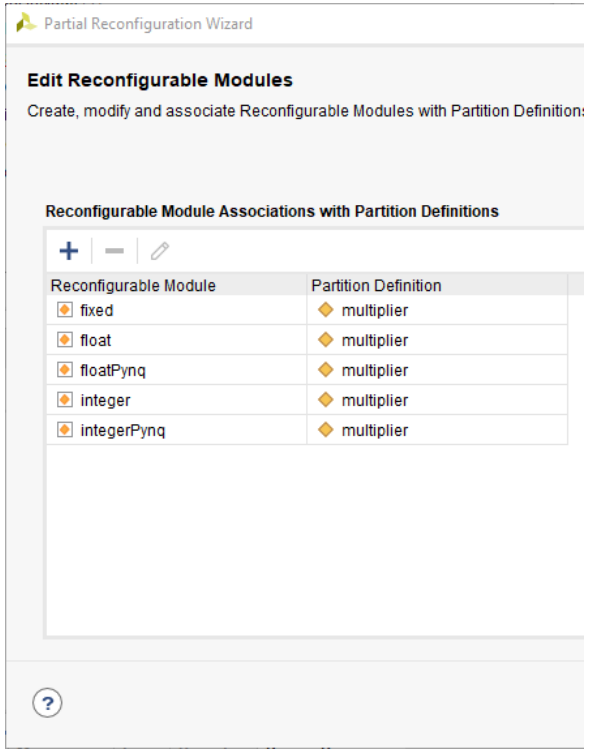
Design Step 2: Partial Reconfig Multipliers (cont.)

- Block diagram of multiplier v2



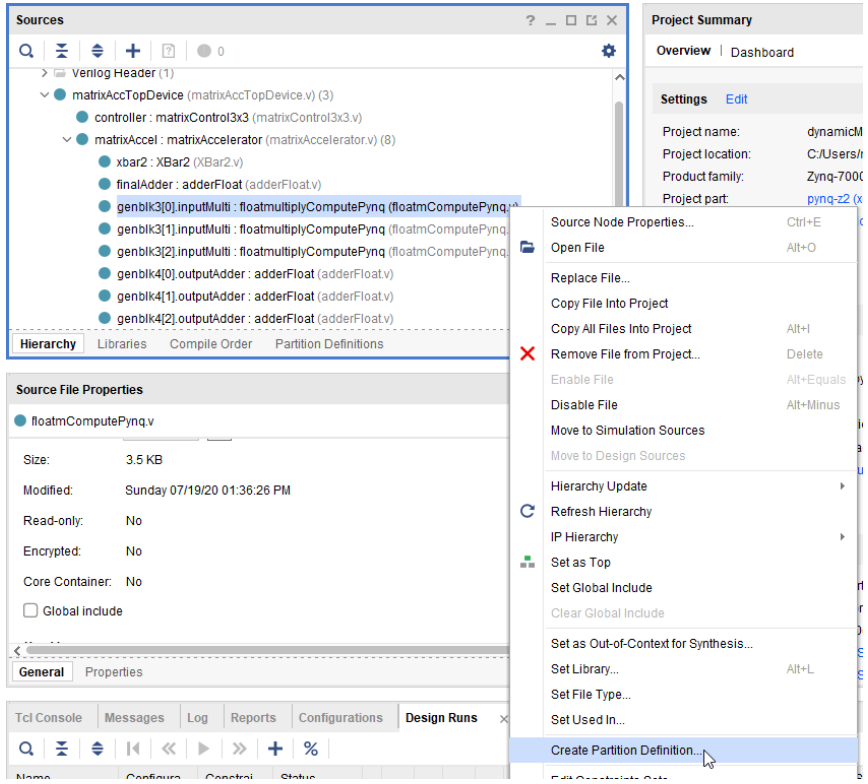
Design Step 3: Enabling Partial Reconfig

- Using Vivado's partial reconfiguration wizard, designs can become dynamically reconfigurable
- This allows for parts of the FPGA to be reprogrammed and swapped out with reconfigurable modules of the same partition definition
- When converting a project to partially reconfigurable, simulation is no longer available
  - It is recommended to flesh out much of a PR design while static, then create a new project for a dynamic version



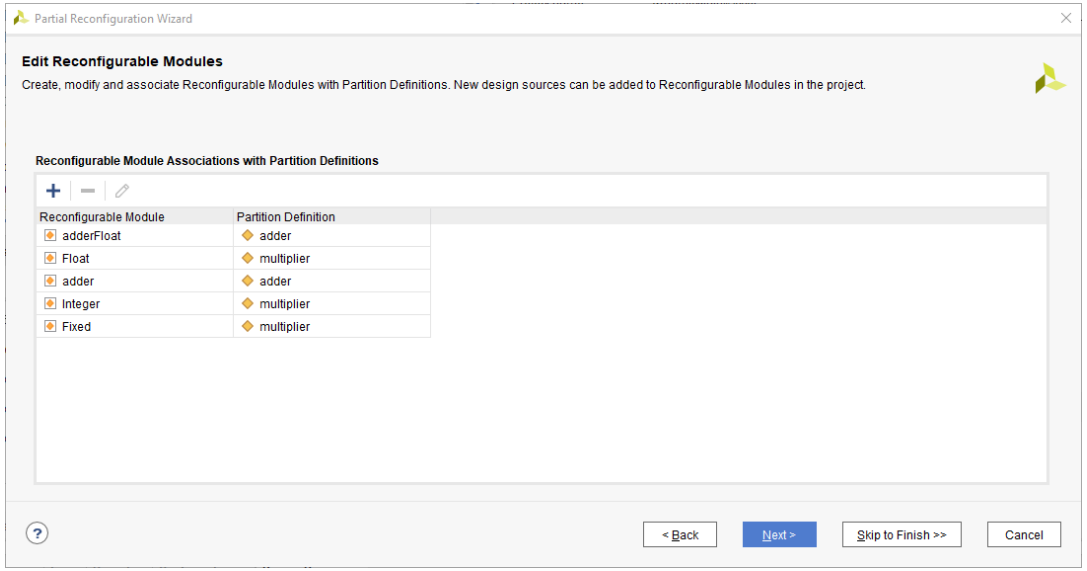
Design Step 3: Enabling Partial Reconfig (cont.)

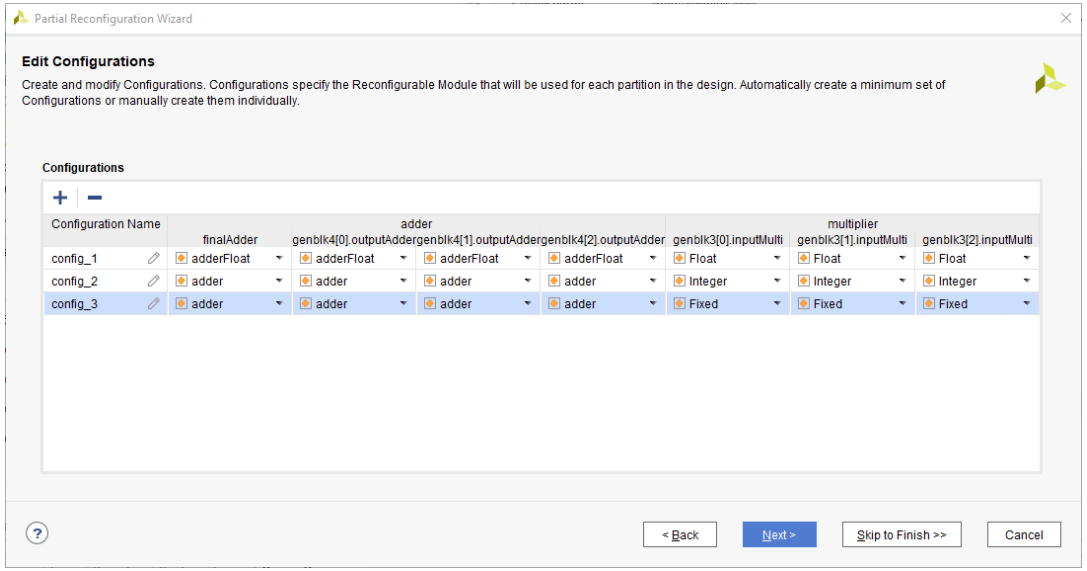
- Backup project
- Enable Partial Reconfiguration
  - Tools->Enable Partial Reconfiguration…
- Select a device in top to create a partition definition



Design Step 3: Enabling Partial Reconfig (cont.)

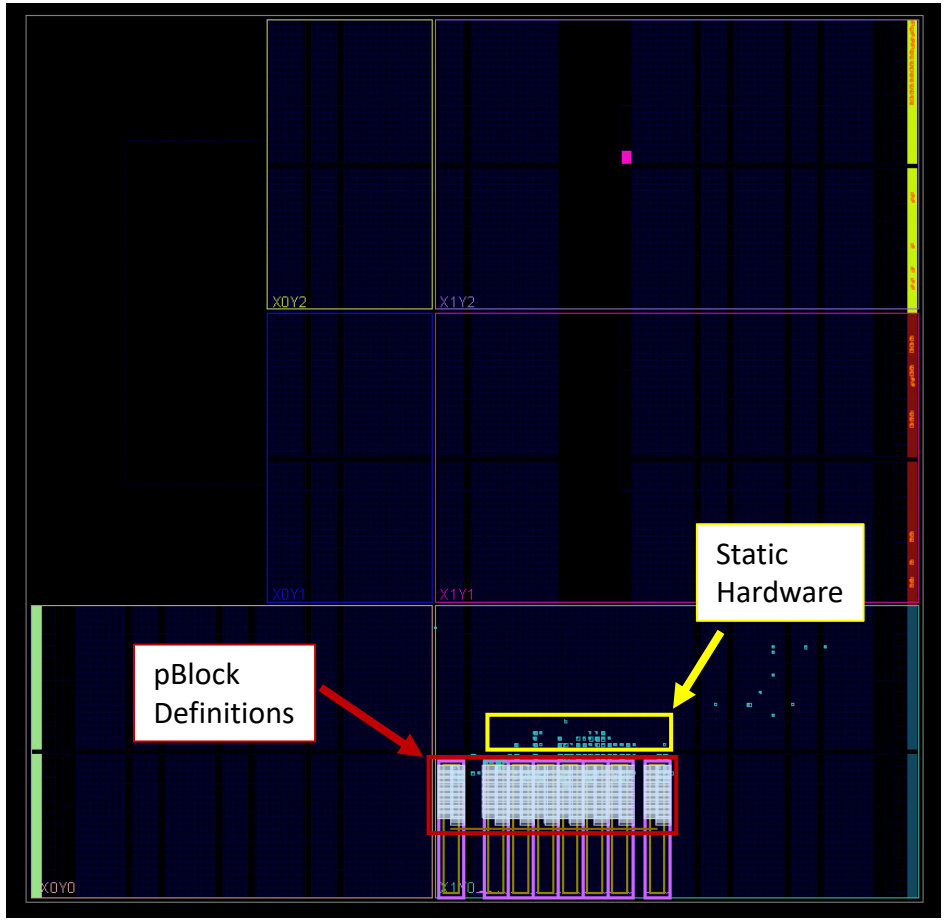
- Add any needed partitions and modules
- Create a configuration with modules set as needed





Design Step 2: Partial Reconfig Multipliers (cont.)

- Each reconfigurable partition requires a dedicated pBlock to dictate the hardware available for reconfiguration
- This pBlock should contain only the dynamic module. Static modules will be placed and routed automatically
- This is a screenshot of the floorplan for a design with 8 pBlocks to enable 8 multiply compute modules.



Design Step 3: Crossbar Data Switch

- A data link was needed to interface many parallel multiplier outputs to many parallel adder devices

- A crossbar switch was designed and implemented. This switch allows for any input port to be selected and connected to any output port.
  - The crossbar was written from scratch to have variable input and output port counts
  - These ports are also variable bit length

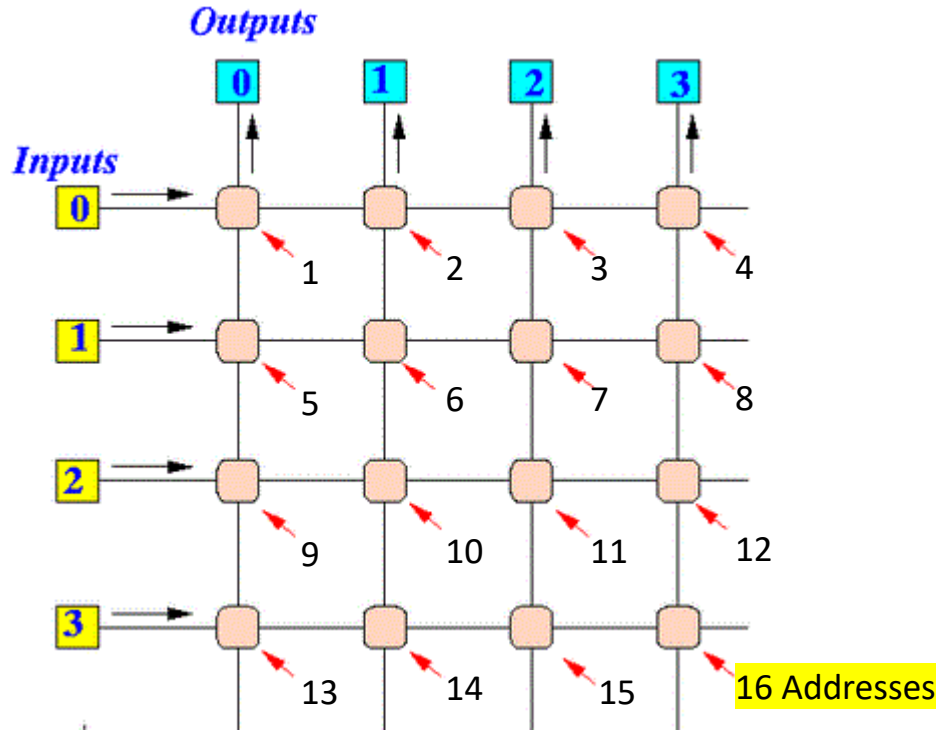
- The data connection is asynchronous, mimicking wire connections

Design Step 3: Crossbar Data Switch (cont.)

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

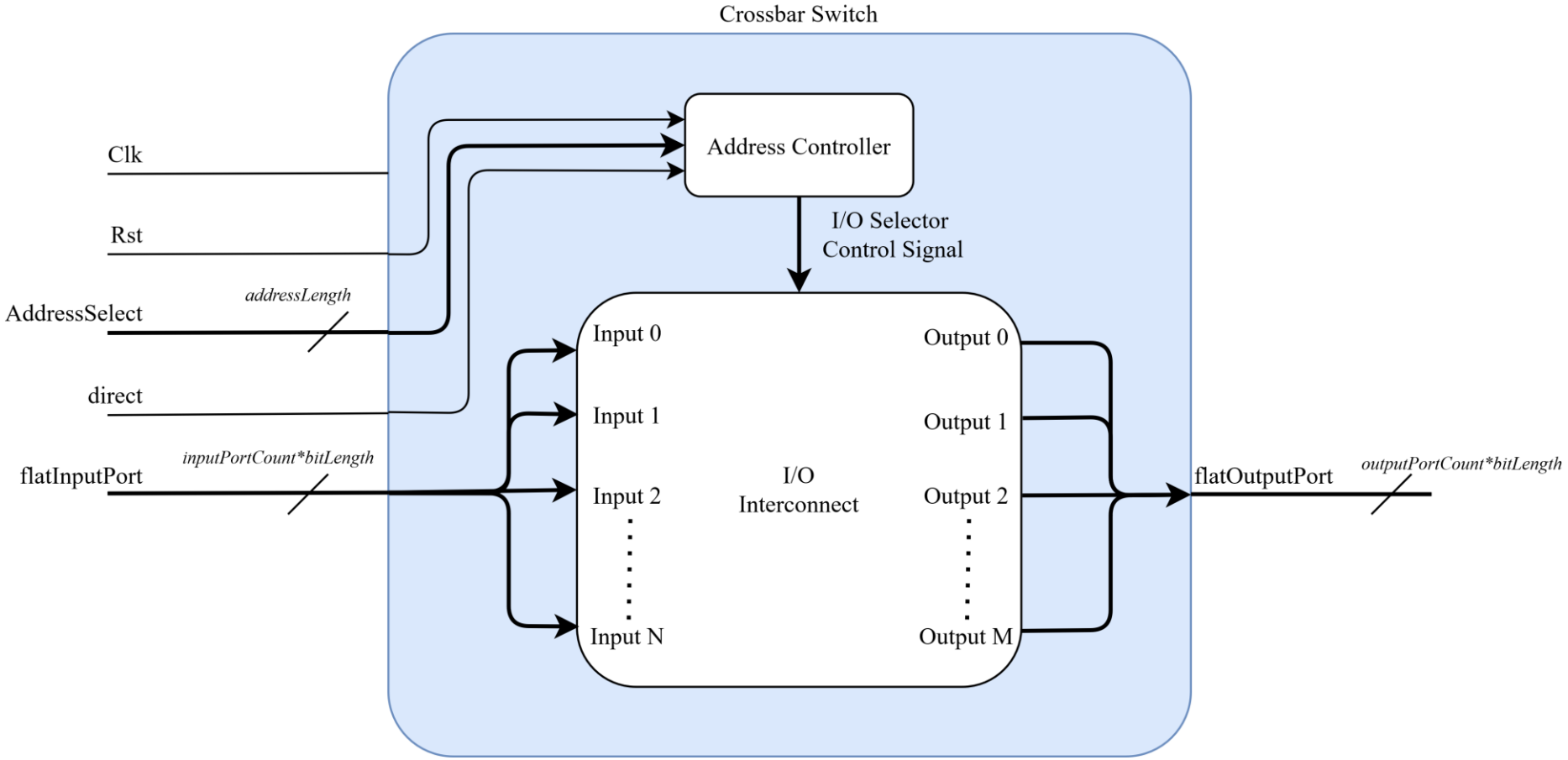
- At positive edge clocks, the address on AddressSelect port is toggled

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



Design Step 3: Crossbar Data Switch (cont.)

- Block diagram of crossbar



Design Step 4: Partial Reconfig Adder

- With a method to now compute many products in parallel and sort them, a device was needed to accumulate the sum of products
  - For the adders, only 2 reconfigurable definitions were needed to handle the three data types. Integer and fixed point can operate with the same definition, but floating point needed a separate definition

- This