Systolic Array Specification

Vinod Ganesan, Gokulan Ravi

July 12, 2019

1 Overview

The systolic-array based accelerator is used to accelerate Deep Learning Inference. It acts as a peripheral device alongside the Shakti C-Class processor. The accelerator is completely parameterized, hence is easily extensible. The entire accelerator is written in Bluespec System Verilog (BSV), a high-level hardware description language.

2 Packages

2.1 intMul_WS

`intMul_WS` package provides module definition for the smallest **M**utiply-**AC**cumulate (MAC) unit or Processing Element (PE). It forms the building block of systolic array. The module `mkintMul_WS` is polymorphic, and so is the interface for the module, `Ifc_intMul_WS`.

```
module mkintMulWS#(Int#(8) row, Int#(8) col, parameter Integer coord)(Ifc_intMul_WS#(bWidth))
    provisos(Mul#(bWidth, 2, twbWidth), Add#(a__, bWidth, twbWidth));
```

The `x,y` coordinates of the PE in the systolic array is specified in the `row` and `column` parameters. `coord` parameter specifies the y-coordinate of the PE, to limit weight flow. An overview of the implementation is presented in Figure 2 . The systolic array is a M $\times$ N grid of PEs. The North-West corner is indexed (0,0) and other PEs are indexed accordingly. In a weight stationary dataflow, weights are populated from the North side, and traverse North to South. Inputs are populated from West side and traverse West to East. Accumulator values (outputs) are populated from North side, and traverse North to South. Hence, the following are the traversals of input, weight and output in a PE.

1. **input:** Received from West through the `from_west` interface, and sent to East by `to_east` interface.
2. **weight:** Received from North through `from_north` interface, and sent to South by `to_south` interface.
3. **output:** Received from North through `acc_from_north` interface, and sent to South by `send_acc_to_south` interface.

The `bWidth` parameter specifies the number of bits in each input/weight value. Number of bits in output is twice `bWidth`. The respective interfaces for input and output carry input and output values respectively. Interface for weight has three values - the weight value, the coordinate upto which the weight should traverse and a dummy opcode. When a weight is received by a PE, it is propagated to the PE in the South `only if` the coordinate received is greater than the y-coordinate of the PE.

```
interface Ifc_intMul_WS#(numeric type bWidth);
  interface Put#(Tuple3#((Maybe#(Bit#(bWidth))),Bit#(8),Bit#(2))) from_north;
  interface Put#(Maybe#(Bit#(bWidth))) from_west;
  interface Put#(Bit#(TMul#(2,bWidth)))  acc_from_north;
  interface Get#(Bit#(TMul#(2,bWidth)))  send_acc_to_south;
  interface Get#(Tuple3#(Maybe#(Bit#(bWidth)),Bit#(8),Bit#(2))) to_south;
  interface Get#(Maybe#(Bit#(bWidth))) to_east;
endinterface
```

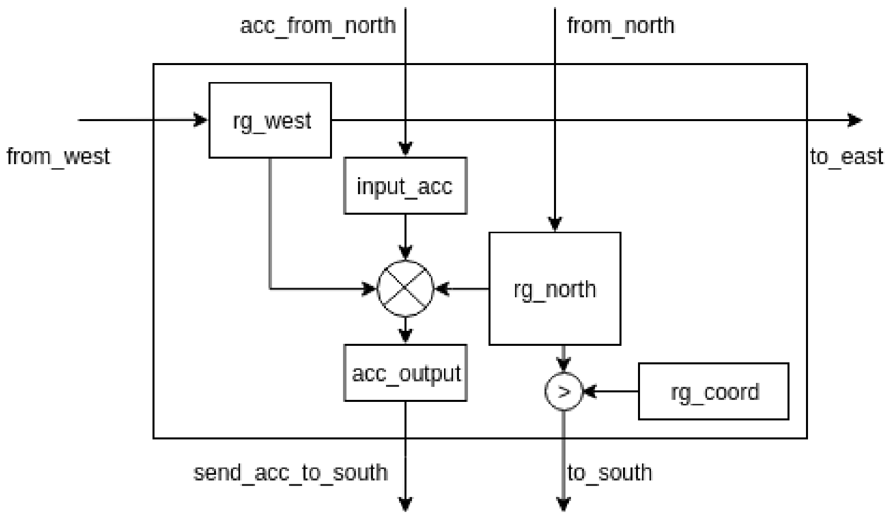

Figure 1: mkintMul_WS module, the smallest MAC unit

2.2 systolic

This package contains `mksystolic` module, which implements a `nRow` × `nCol` (parameterized) systolic grid, with each element in the grid being a Processing Element (PE), an instance of `mkintMul_WS` module. Taking care of edge conditions, each PE is connected with another PE in all four directions.

```
  module mksystolic(Ifc_systolic#(nRow,nCol,mulWidth))
      provisos (Add#(a__, mulWidth, TMul#(mulWidth, 2)));
```

In addition to `nRow` and `nCol`, the module takes in `mulWidth` parameter, which is used in the `bWidth` parameter for initializing every `mkintMul_WS` module. Except PEs present in the edges $Row0, Row$`nRow-1`$, Column0, Column$`nCol`-1 every PE is connected to four other PEs present in the North, South, West and East directions.

```
interface Ifc_RFIFO_Connections#(numeric type mulWidth);
  method Action send_rowbuf_value(Maybe#(Bit#(mulWidth)) value);
endinterface

interface Ifc_CFIFO_Connections#(numeric type mulWidth);
  method Action send_colbuf_value(Tuple4#(Maybe#(Bit#(mulWidth)),
    Bit#(TMul#(2,mulWidth)), Bit#(8),Bit#(2)) value);
  method Action send_acc_value(Bit#(TMul#(2, mulWidth)) accinput);
  method ActionValue#(Bit#(TMul#(2,mulWidth))) send_accumbuf_value;
endinterface

interface Ifc_systolic#(numeric type nRow, numeric type nCol,
                        numeric type mulWidth);
  interface Vector#(nRow, Ifc_RFIFO_Connections#(mulWidth)) rfifo;
  interface Vector#(nCol, Ifc_CFIFO_Connections#(mulWidth)) cfifo;
endinterface
```

Interface `Ifc_RFIFO_Connections` is a per-row interface, to send input values to the systolic array. It consists of `send_rowbuf_value` method, which sends input values to the PEs in Column 0, using their respective `from_west` interface. `Ifc_CFIFO_Connections` is a per-column interface - to send weights and to send & receive output values. It comprises of three methods - `send_colbuf_value` to send weights, `send_acc_value` to send outputs and `send_accumbuf_value` to receive outputs.

2.2.1 Internal Connections

Considering a systolic array of size `nRow` × `nCol`. A PE with index (i, j) has the following connections.

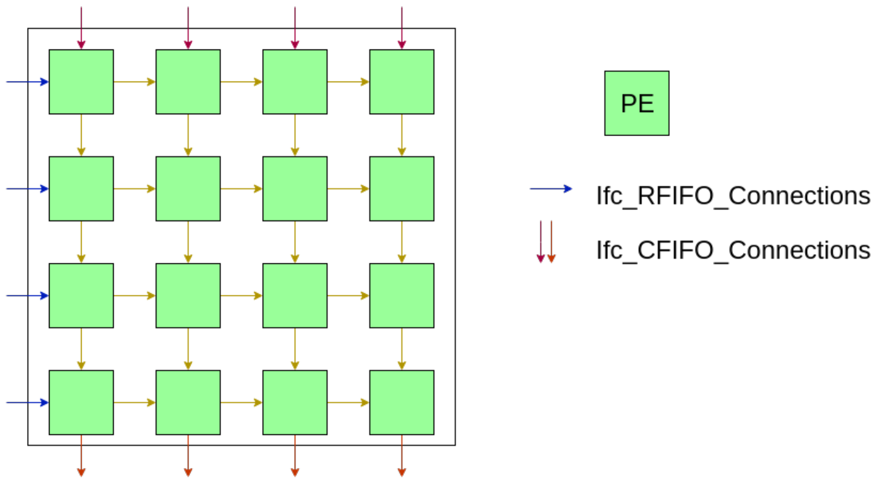

Figure 2: A 4 × 4 mksystolic module

- **i=0**: `from_north` and `acc_from_north` interfaces used in `cfifo[j].send_colbuf_value`.
- **j=0**: `from_west` interface is used in `rfifo[i].send_rowbuf_value`.
- **i=nRow-1**: `send_acc_to_south` interface is used in `send_accumbuf_value`.
- **i!=0 and i!=nRow-1**:

2.3 systolic_top

This package provides the module implementation for `mksystolic_top_axi4`, which acts as a peripheral alongside the processor. The module contains a AXI-4 slave module, which interacts with the `busWidth`-bit bus (64 for now).

The module contains an instance of `mksystolic` module, to which inputs and weights are fed, and outputs are received from. When a `nRow` × `nCol` systolic array is initialized, we add additional FIFOs so that fetch and access can be separated. There are `nRow` FIFOs `rowBuffers`, one per row, with each element of type `Bit#`*mulWidth*. Also, there are `nCol` FIFOs, one per column, with each element of type `Tuple4#(Bit#(mulWidth), Bit#(2*mulWidth), Bit#(8), Bit#(2))`. Each `Ifc_Rfifo_Connections` interface from the `mksystolic` instance connects to each of the FIFO, and fetches value from it. Similarly, each `Ifc_Cfifo_Connections` interface connects to each of the FIFO (`columnBuffers`), and fetches value from it.

The module also contains two buffers - Global Buffer, Accumulator Buffers and a set of configuration registers. Global Buffer is used for storing input values. Accumulator buffer is used for storing accumulator values. Every input/weight received from the bus is stored in these buffers. Configuration registers are used to set convolution parameters like input dimensions, weight dimensions, padding. In addition to that, few registers act as control signals for systolic array to start or stop execution. All three units - Global Buffer, Accumulator buffer and Configuration registers, are memory-mapped.

```
module mksystolic_top_axi4(Ifc_systolic_top_axi4#(addr_width,data_width,user_width,
                             sqrtnRow,sqrtnCol,gbufaddr,accumaddr,nFEntries,mulWidth));
endmodule
```

The following are the parameters for the polymorphic `mksystolic_top_axi4` module.

1. `addr_width`: Not used yet.
2. `data_width`: AXI-4 bus width.
3. `user_width`: Not used yet.
4. `sqrtnRow`: Square-root of number of rows in the systolic array.
5. `nCol`: Number of columns in the systolic array.

6. `gbufaddr`: Number of bits used to index into an entry in the Global buffer.
7. `abufaddr`: Number of bits used to index into an entry in the Accumulator buffer.
8. `nFEntries`: Number of entries in each of the `rowBuffer` and `columnBuffer` FIFOs.
9. `mulWidth`: Number of bits in each input/weight. The same is used to initialize `mksystolic` module.

```
interface Ifc_systolic_top_axi4#(numeric type addr_width, numeric type data_width,
                                numeric type user_width, numeric type nRow, numeric type nCol,
                                numeric type accumaddr,numeric type gbufaddr,
                                numeric type nFEntries,numeric type mulWidth);
    interface AXI4_Slave_IFC#(‘PADDR,data_width,0) slave_systolic;
    endinterface
```

Number of banks in the `columnBuffer` should be the same as the number of columns in the systolic array. Since there are `nCols` number of columns, each column producing one output value per cycle, each value would be written to corresponding bank each cycle.

However, number of banks in the `rowBuffer` should be square-root of the number of rows in the systolic array. For example, with a 3×3 weight, each input value will be multiplied by three different input values, for three different continous outputs. Hence, if the unrolled filter size is of size `sqrtnRow * sqrtnRow`, it will be multiplied by `sqrtnRow` different continous weights, which would be populated in different continous rows. Hence Global buffer contains `sqrtnRow` banks, and each value is sent to `sqrtnRow` banks in the same number of cycles, each time to a different row.

The memory map of different memory units is listed below.

Hardware	Description	Address range
Configuration Registers	IfmapDims	’hB0000000
	sysConfig	’hB0000002
	CoordCount	‘hB0000004
	weightCount	’hB0000006
Weights	WeightStart - WeightEnd	’hB0010000 - ’hB001ffff
Accumulator Buffer	AccumBufStart - AccumBufEnd	’hB0200000 - ’hB02fffff
Global Buffer	GBufStart - GBufEnd	’hB0030000 - ’hB003ffff

3 2D Convolution on systolic array

The number of rows in the systolic array should be fixed as the number of elements when the weight is unrolled. Each column computes convolution of different filter. Firstly, configuration registers are set with input and weight dimensions. Then, all the weights are sent from the host processor and populated in the array through `ColumnBuffers`. Similarly all inputs are populated in Global buffer. The host sets the `startBit` register to start the systolic execution. Inputs are then sent to the systolic array through `rowBuffer` and output values are sent through `columnBuffers`. After all MAC operations complete, the output is finally stored in the Accumulator buffer, which is read by the host.