HLS Lab-B

Binarized Neural Network

R09944013
林芍甫

Outline

- FINN: A Framework for Fast, Scalable Binarized Neural Network Inference [FPGA ‘17]
- Reproducing results with PYNQ-Z2

FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

25th International Symposium on Field-Programmable Gate Arrays, February 2017
Yaman Umuroglu*† , Nicholas J. Fraser*‡ , Giulio Gambardella* , Michaela Blott* , Philip Leong‡ , Magnus Jahre† and Kees Vissers*
*Xilinx Research Labs; †Norwegian University of Science and Technology; ‡University of Sydney yamanu@idi.ntnu.no

FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

- Contribution
  - Optimizations for mapping BNNs onto FPGA more efficiently
  - A range of prototypes that demonstrate the potential of BNNs on an off-the-shelf FPGAs platform
- Methods
  - The Matrix–Vector–Threshold Unit
  - BNN-specific Operator Optimizations
    - Popcount for Accumulation
    - Batchnorm-activation as Threshold
    - Boolean OR for Max-pooling
  - Convolution: The Sliding Window Unit

BNN-specific Operator Optimizations

- Popcount for Accumulation
- Batchnorm-activation as Threshold
- Boolean OR for Max-pooling

The Matrix–Vector–Threshold Unit (top.cpp)

- The Matrix–Vector–Threshold Unit
  - SIMD = Number of input columns computed in parallel
  - PE = Number of output rows computed in parallel
- Streaming FIFO

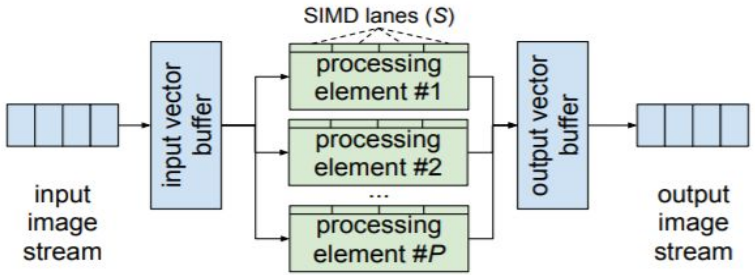


Figure 5: Overview of the MVTU.

The Matrix–Vector–Threshold Unit (top.cpp)

```
void DoCompute(ap_uint<64> *in, ap_uint<64> *out, const unsigned int numReps) {

hls::stream<ap_uint<64>>     memInStrm("DoCompute.memInStrm");
hls::stream<ap_uint<L0_PE * (L0_API + L0_APF)>>  inter0("DoCompute.inter0");
hls::stream<ap_uint<64>>     memOutStrm("DoCompute.memOutStrm");


#pragma HLS stream depth=1024 variable=memInStrm
#pragma HLS stream depth=L0_DEPTH variable=inter0
#pragma HLS stream depth=L1_DEPTH variable=inter1
#pragma HLS stream depth=L2_DEPTH variable=inter2
#pragma HLS stream depth=1024 variable=memOutStrm

Mem2Stream_Batch<64, inBytesPadded>(in, memInStrm, numReps);
StreamingFCLayer_Batch
  <L0_MW, L0_MH, L0_SIMD, L0_PE, Recast <XnorMul>, Slice<ap_uint<1>>>
  (memInStrm, inter0, weights0, threshs0, numReps,   ap_resource_lut());
```

The Matrix–Vector–Threshold Unit (mvau.hpp)

- Popcount/XNOR Operation in MAC
- Batchnorm-activation as Threshold
- Boolean OR for Max-pooling

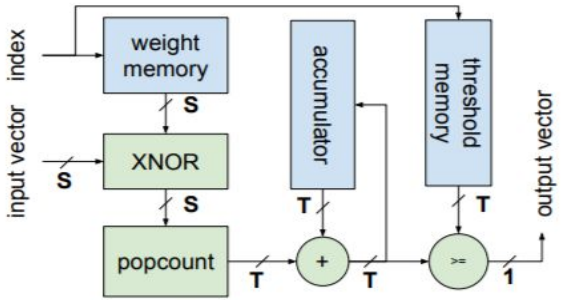
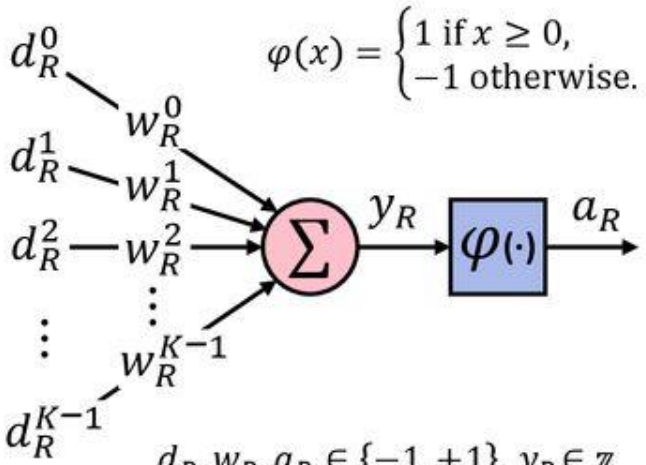


Figure 6: MVTU PE datapath. **Bold** indicates bitwidth.

A Binary Neuron

Simulated using real numbers:

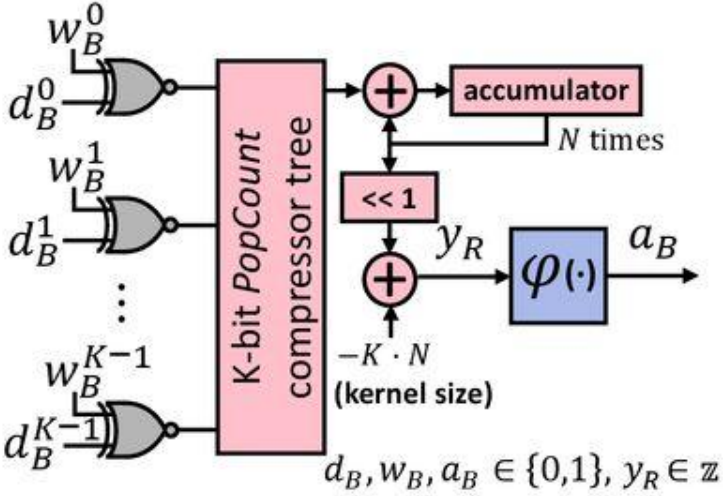


Encoding:

-1 → 0

+1 → 1

Implemented using binary encoding:

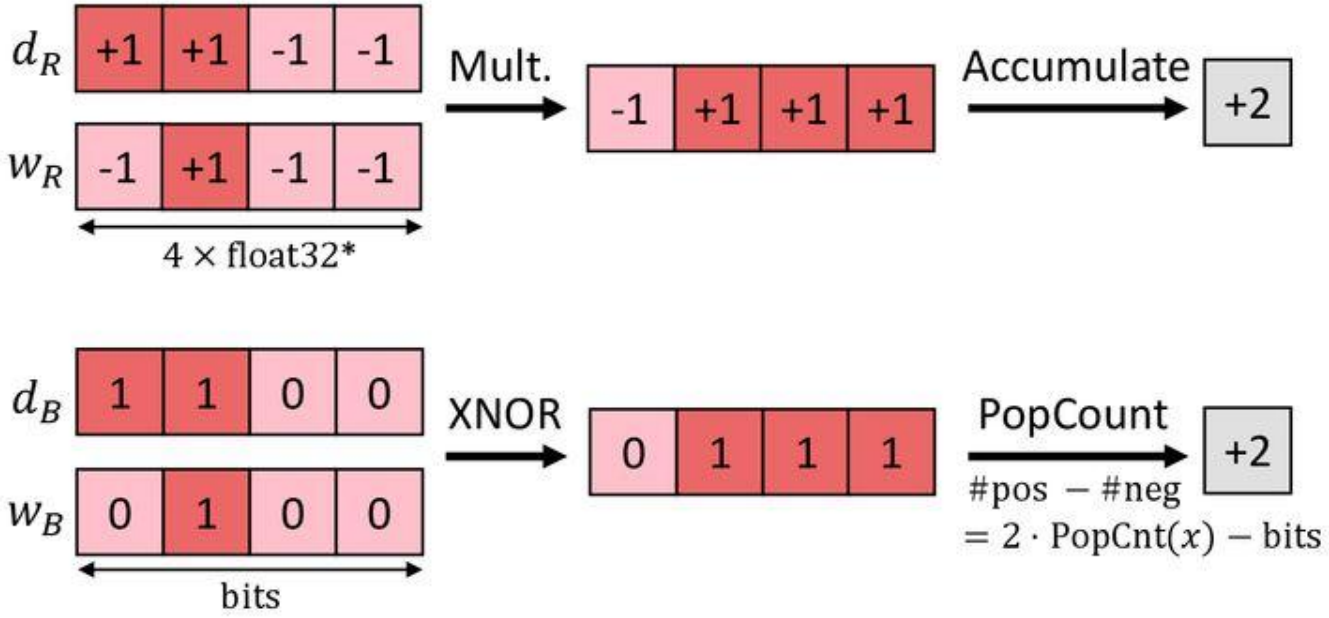


$$a_B = \varphi\left(\sum_{i=0}^{K-1} w_R^i \cdot d_R^i\right)$$

$$a_B = \varphi\left(2 \cdot PopCnt\left(\sim\left(w_B^i \hat{} d_B^i\right)\right) - K \cdot N\right)$$

Convolution with bitwise operations

Multiplication and addition are replaced by bitwise XNOR and PopCount.



** To enable usage of fast floating-point GPU kernels in PyTorch/Tensorflow during BNN training.*

The Matrix–Vector–Threshold Unit (mvau.hpp)

```
// compute matrix-vector product for each processing element
auto const &w = weights.weights(tile);
for (unsigned pe = 0; pe < PE; pe++) {
#pragma HLS UNROLL
 auto const wgt = TWeightI()(w[pe]);
 for (unsigned mmv = 0; mmv < MMV; mmv++) {
   auto const act = TSrcI()(inElem, mmv);
   accu[mmv][pe] = mac<SIMD>(accu[mmv][pe], wgt, act, r, mmv);
 }
}
```

```
class XnorMul {
 ap_uint<1> const  m_val;
public:
 XnorMul(ap_uint<1> const  val) : m_val(val) {
#pragma HLS inline
 }

public:
 int operator*(ap_uint<1> const &b) const {
#pragma HLS inline
   return  m_val == b? 1 : 0;
 }
};
inline int operator*(ap_uint<1> const &a,
XnorMul const &b) {
#pragma HLS inline
 return  b*a;
}
```

The Matrix–Vector–Threshold Unit (mvau.hpp)

- Let the output of dot product be a_k , the batch normalization parameters $\Theta_k = (\gamma_k, \mu_k, i_k, B_k)$
- Batch normalization $\text{BatchNorm}(a_k, \Theta_k) = \gamma_k \cdot (a_k - \mu_k) \cdot i_k + B_k$
- prior to the activation function $\text{Sign}(\text{BatchNorm}(a_k, \Theta_k)$
- We could combine BatchNorm and Activation as a single threshold τ_k
  - $\tau_k = \mu_k - (\frac{B_k}{\gamma_k \cdot i_k})$

The Matrix–Vector–Threshold Unit (mvau.hpp)

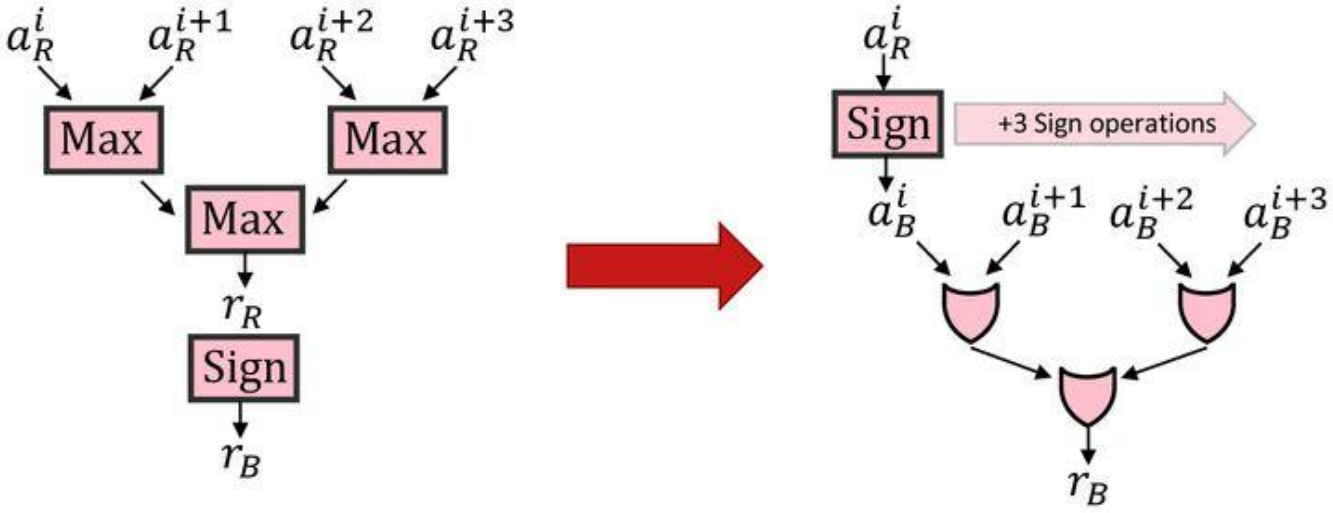
```
for (unsigned pe = 0; pe < PE; pe++) {
#pragma HLS UNROLL
  for (unsigned mmv = 0; mmv < MMV; mmv++) {
#pragma HLS UNROLL
    outElem(pe, mmv, 1) =
        activation.activate(nf, pe, accu[mmv][pe]);
  }
}
```

```
template<unsigned NF, unsigned PE, unsigned NumTH,
  typename TA, typename TR, int ActVal = 0, typename Compare = std::less<TA>>
class ThresholdsActivation{
public:
 TA m_thresholds[PE][NF][NumTH];
 public:
 TA init(unsigned const  nf, unsigned const  pe) const {
#pragma HLS inline
   return  TA(0);
 }

public:
 TR activate(unsigned const  nf, unsigned const  pe,  TA const &accu) const {
#pragma HLS inline
   TR result=ActVal;
 for(unsigned int i=0; i< NumTH; i++){
#pragma HLS unroll
     result+=Compare()(m_thresholds[pe][nf][i], accu);
   }
   return result;
 }
};
```

Max-pooling with bitwise operations

- During training the Max-pooling layer was put before binary activation.
- During inference, the max-pooling layer should be put after the binary activation to optimize the computation:*

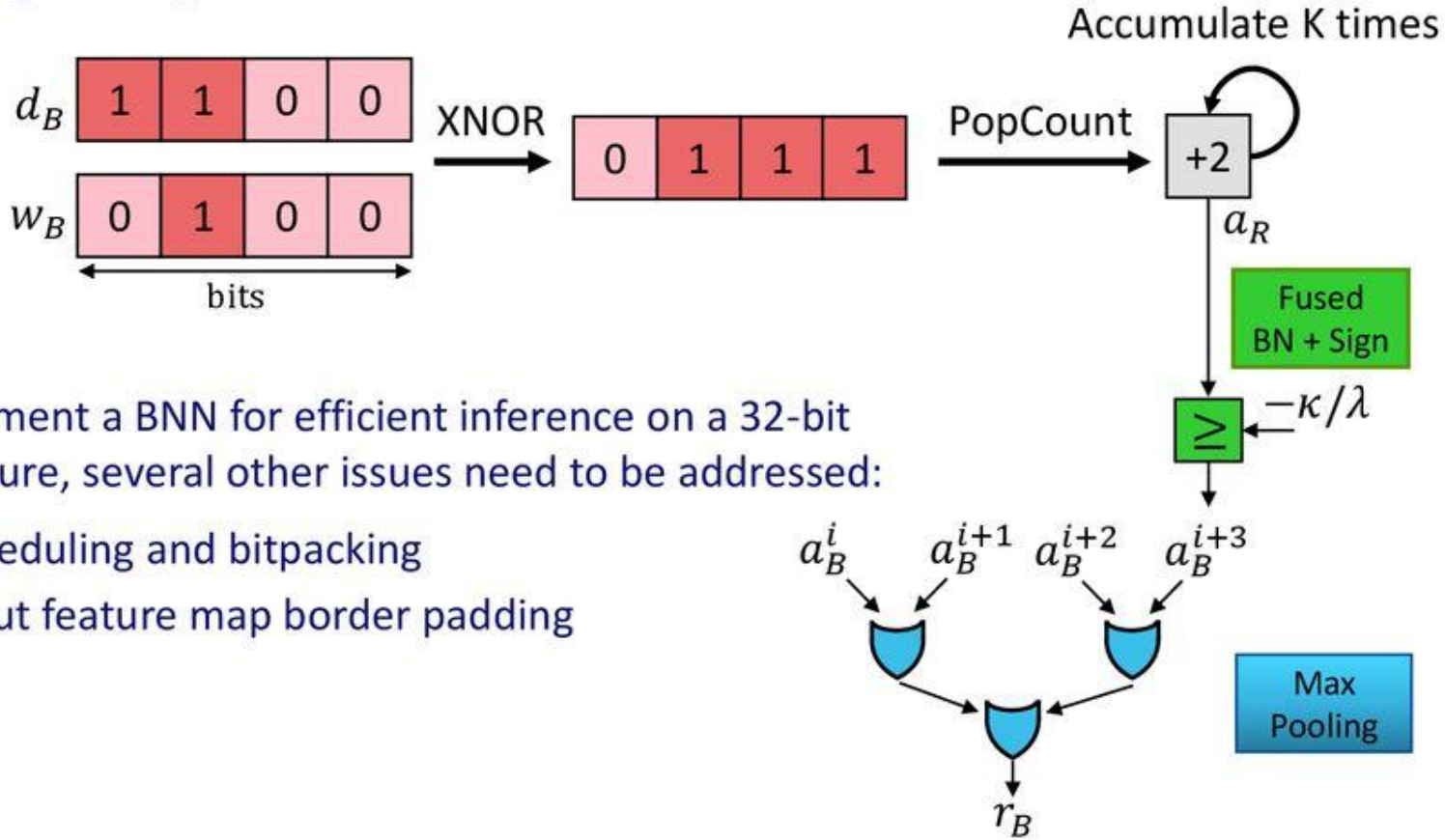


* Finn: A framework for fast, scalable binarized neural network inference – (FPL 2017)

The Matrix–Vector–Threshold Unit (mvau.hpp)

```
template<unsigned int ImgDim, unsigned int PoolDim, unsigned int NumChannels>
void StreamingMaxPool(stream<ap_uint<NumChannels> > & in,
   stream<ap_uint<NumChannels> > & out) {
...
 for (unsigned int yp = 0; yp < ImgDim / PoolDim; yp++) {
   for (unsigned int ky = 0; ky < PoolDim; ky++) {
     for (unsigned int xp = 0; xp < ImgDim / PoolDim; xp++) {
#pragma HLS PIPELINE II=1
       ap_uint<NumChannels> acc = 0;
       for (unsigned int kx = 0; kx < PoolDim; kx++) {
         acc = acc | in.read();
       }
       // pool with old value in row buffer
       buf[xp] |= acc;
     }
   }
}
...
}
```

Putting it together



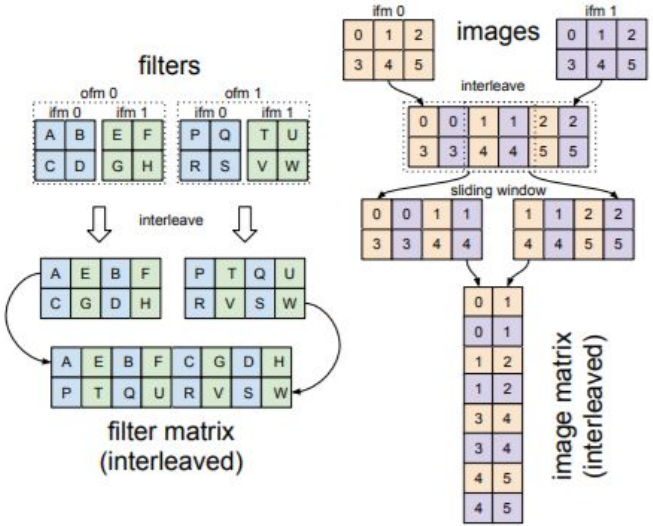
To implement a BNN for efficient inference on a 32-bit architecture, several other issues need to be addressed:

- Scheduling and bitpacking
- Input feature map border padding

Convolution: The Sliding Window Unit

```
ConvolutionInputGenerator <ConvKernelDim, IFMChannels,  TSrcI::width, IFMDim,
          OFMDim, SIMD, 1>(wa_in, convInp, reps);

Matrix_Vector_Activate_Batch <MatrixW, MatrixH, SIMD, PE,  1, TSrcI, TDstI,
TWeightI>
   (static_cast< hls::stream<ap_uint<SIMD*TSrcI::width>>&>(convInp),
    static_cast< hls::stream<ap_uint<PE*TDstI::width>>&>  (mvOut),
    weights, activation, reps * OFMDim * OFMDim, r);
```



Reproducing FINN on PYNQ-Z2

Models and Datasets

- Models
  - LFC Model
    - three-layer fully connected network
  - CNV
    - Convolution layer
      - 3x3 convolution, 3x3 convolution, 2x2 maxpool
    - Convolution layers repeated three times with 64-128-256 channels
    - two fully connected layers of 512 neurons
- Datasets
  - MINST
  - CIFAR-10

Implementations

- 1 bit weights and 1 bit activation (W1A1) for CNV and LFC
- 1 bit weights and 2 bit activation (W1A2) for CNV and LFC
- 2 bit weights and 2 bit activation (W2A2) for CNV

cnvW1A1 - HLS synthesis

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	6	-
FIFO	-	-	-	-	-
Instance	62	24	178116	122454	0
Memory	195	-	2912	160	0
Multiplexer	-	-	-	7002	-
Register	-	-	160	-	-
Total	257	24	181188	129622	0
Available	280	220	106400	53200	0
Utilization (%)	91	10	170	243	0

cnvW1A1 - RTL Synthesis

Site Type	Used	Fixed	Available	Util%
Slice LUTs	29635	0	53200	55.70
LUT as Logic	27197	0	53200	51.12
LUT as Memory	2438	0	17400	14.01
LUT as Distributed RAM	1578	0		
LUT as Shift Register	860	0		
Slice Registers	42053	0	106400	39.52
Register as Flip Flop	42053	0	106400	39.52
Register as Latch	0	0	106400	0.00
F7 Muxes	2404	0	26600	9.04
F8 Muxes	589	0	13300	4.43

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	124	0	140	88.57
RAMB36/FIFO*	76	0	140	54.29
RAMB36E1 only	76			
RAMB18	96	0	280	34.29
RAMB18E1 only	96			

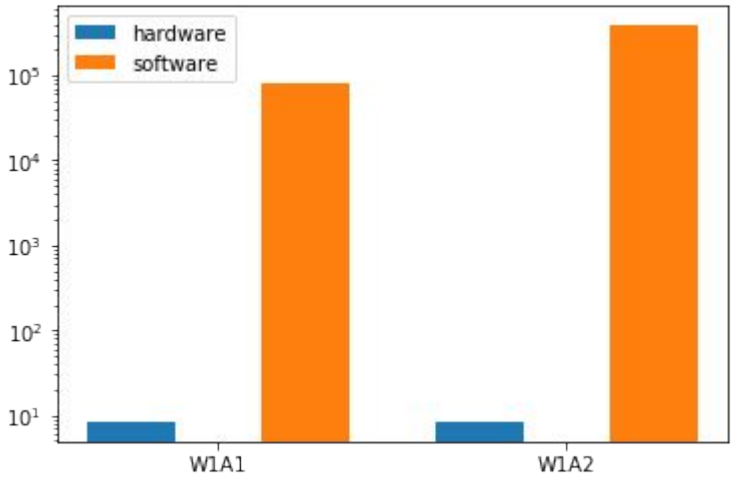
Site Type	Used	Fixed	Available	Util%
DSPs	24	0	220	10.91
DSP48E1 only	24			

Compare cnvW1A1 and cnvW1A2

		FF	LUT
cnvW1A1	HLS	181188	129622
	RTL	42053	29635
cnvW1A2	HLS	202908	152857
	RTL	Place Design ERROR!	

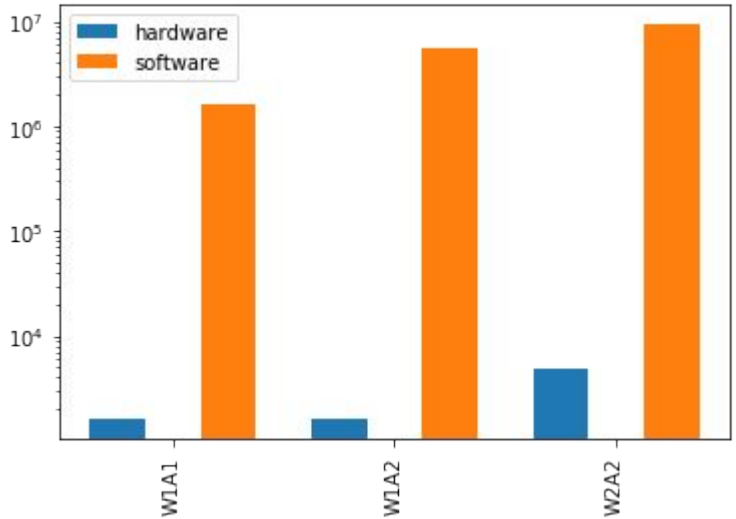
LFC model

- MNIST + LFC
  - Compare HW inference and SW inference elapsed time
  - Accuracy: 98.4%



CNV model

- Cifar10 + CNV
  - Compare HW inference and SW inference elapsed time



BinaryNets for Pynq

- Try to train BinaryNets with https://github.com/MatthieuCourbariaux/BinaryNet
- However, it is deprecated and hard to setup environment (Theano, PyLearn2, ...).

Takeaway Questions

1. XNOR gate is used in which operation in the BNN?
2. Which one operates last in the design of FINN?
    a. popcount
    b. xnor
    c. threshold