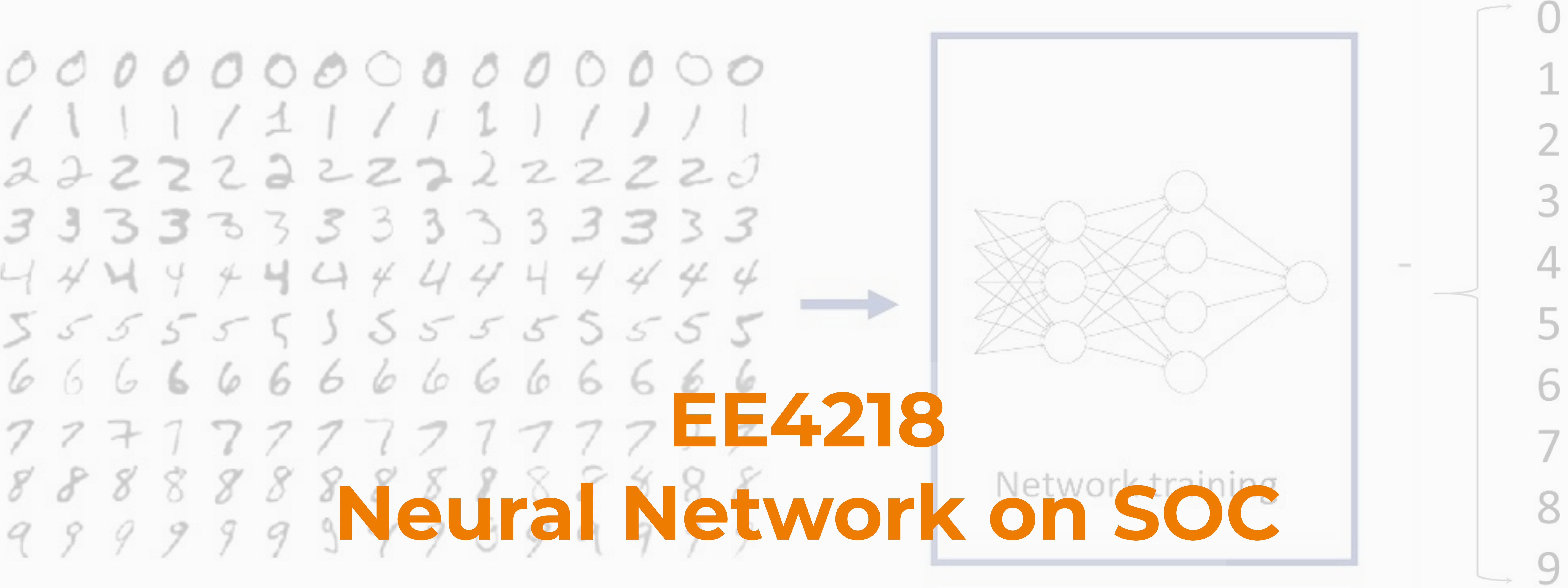
EE4218
Neural Network on SOC

Group 2   Liu Zichen and Liu Xingsi

2019

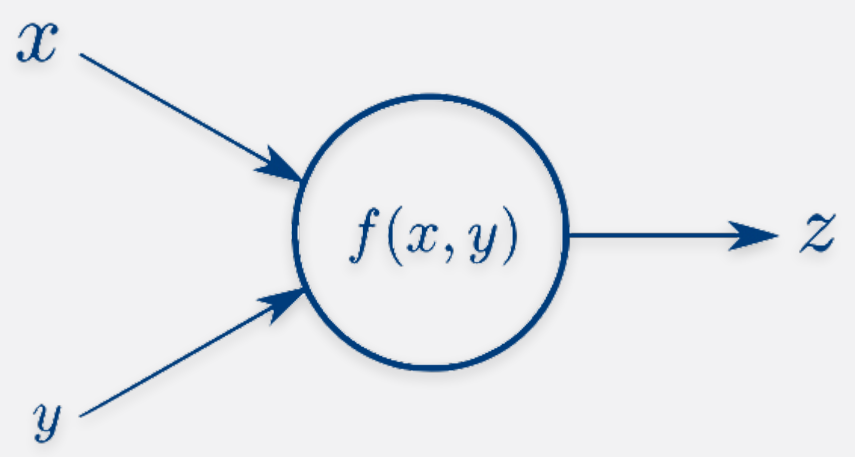


National University of Singapore

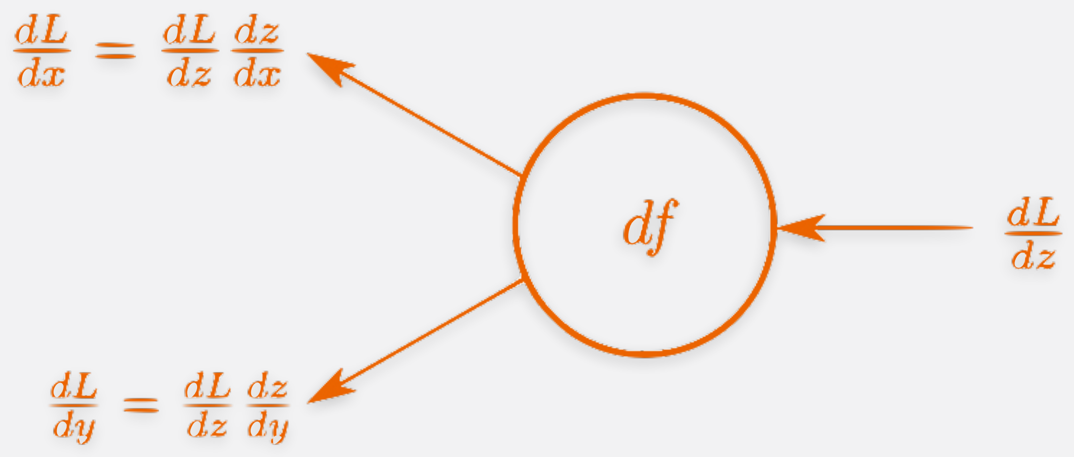
Prelude: Python and C

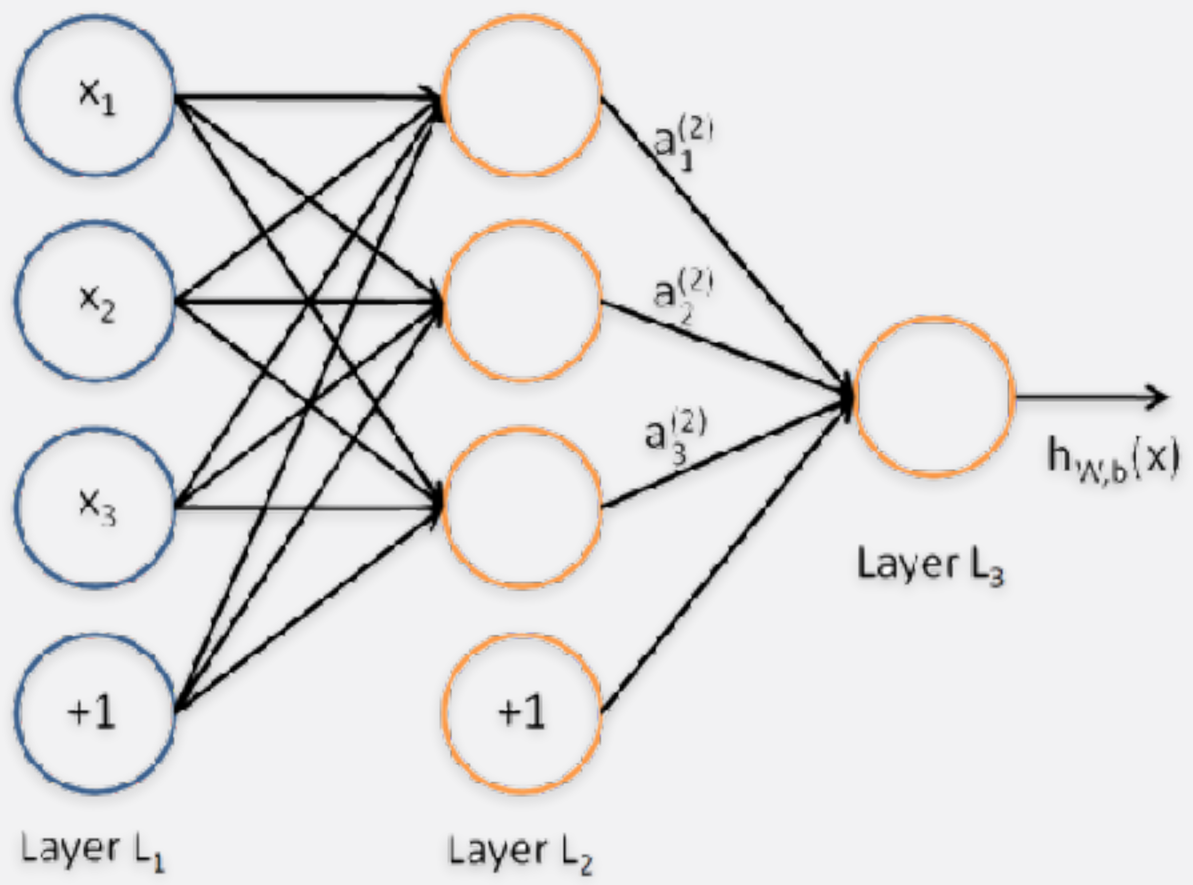
Data & Network Design & Insights

Forwardpass



Backwardpass


$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
df
$\frac{dL}{dz}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$

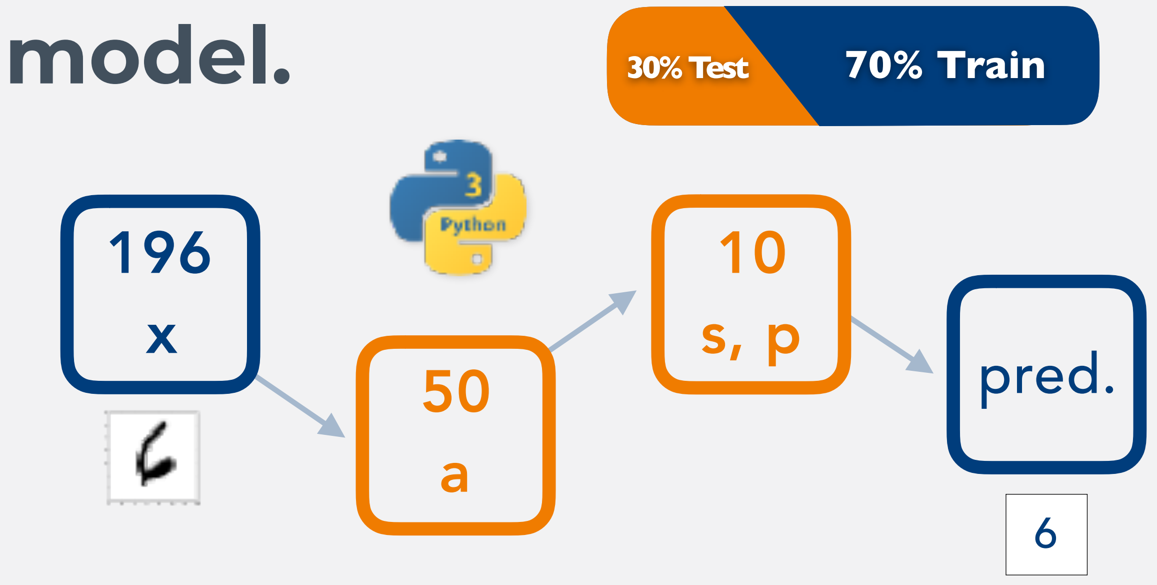


$$\hat{a}_i = \max\left(W_1\hat{x}_i + \hat{b}_1\ , 0\right)$$
$$\hat{s}_i = W_2\hat{a}_i + \hat{b}_2$$
$$\hat{p}_i = softmax(\hat{s}_i)$$
$$y_i = argmax(\hat{p}_i)$$

where $\hat{x}_i \in R^D, \hat{a}_i \in R^H, \hat{s}_i, \hat{p}_i \in R^C, y_i \in R$

for $i \in 1,2,\ldots,N$

model.



result.

Feasibility

Accuracy: **93.7%** @ 5000 Data

W/b distribution (resolution)

5th : **~10^{-5}**

95th : **~10^{-3}**

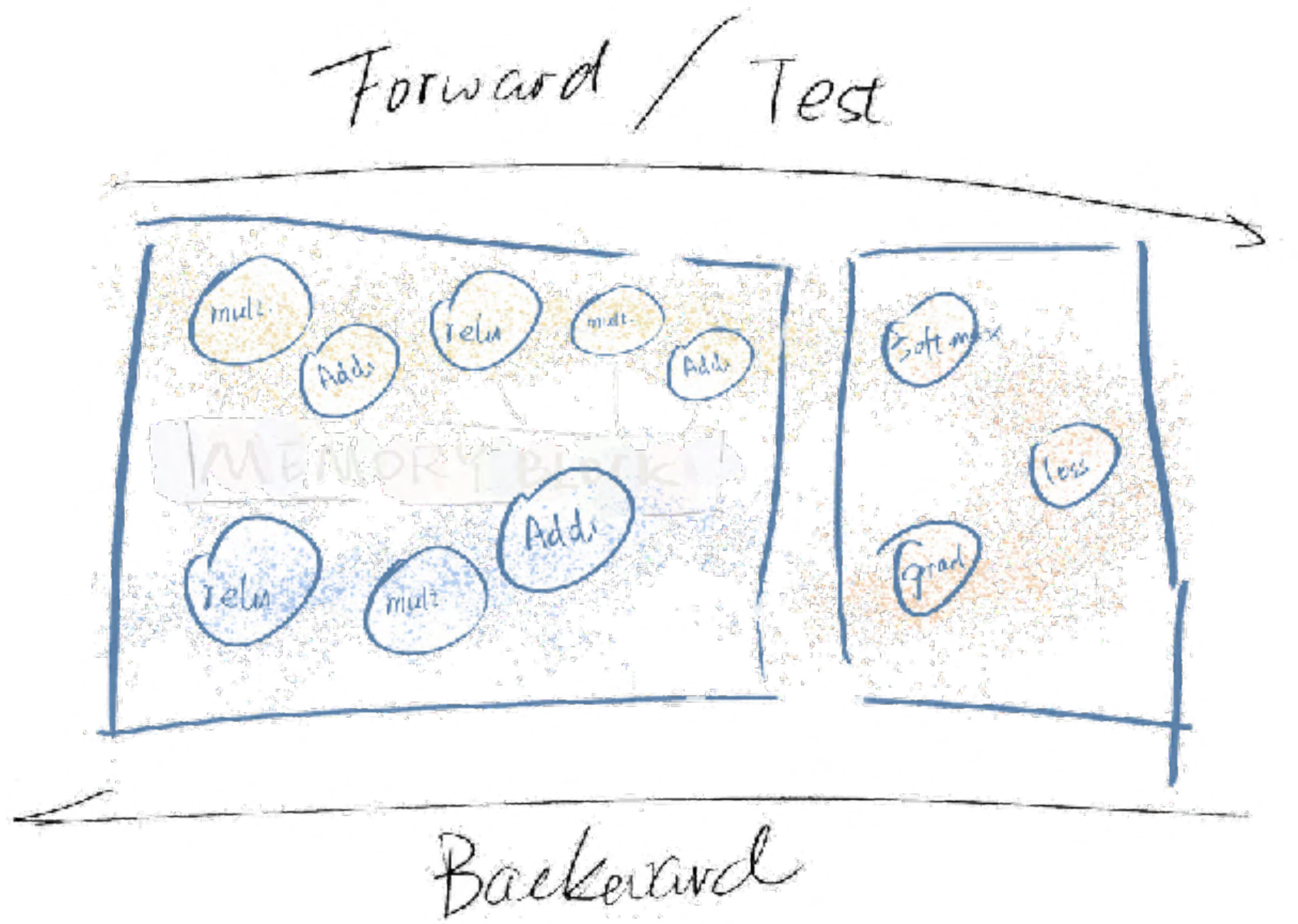
Test Time @3000 Data

1.462ms → gcc -Ofast → **0.3ms**

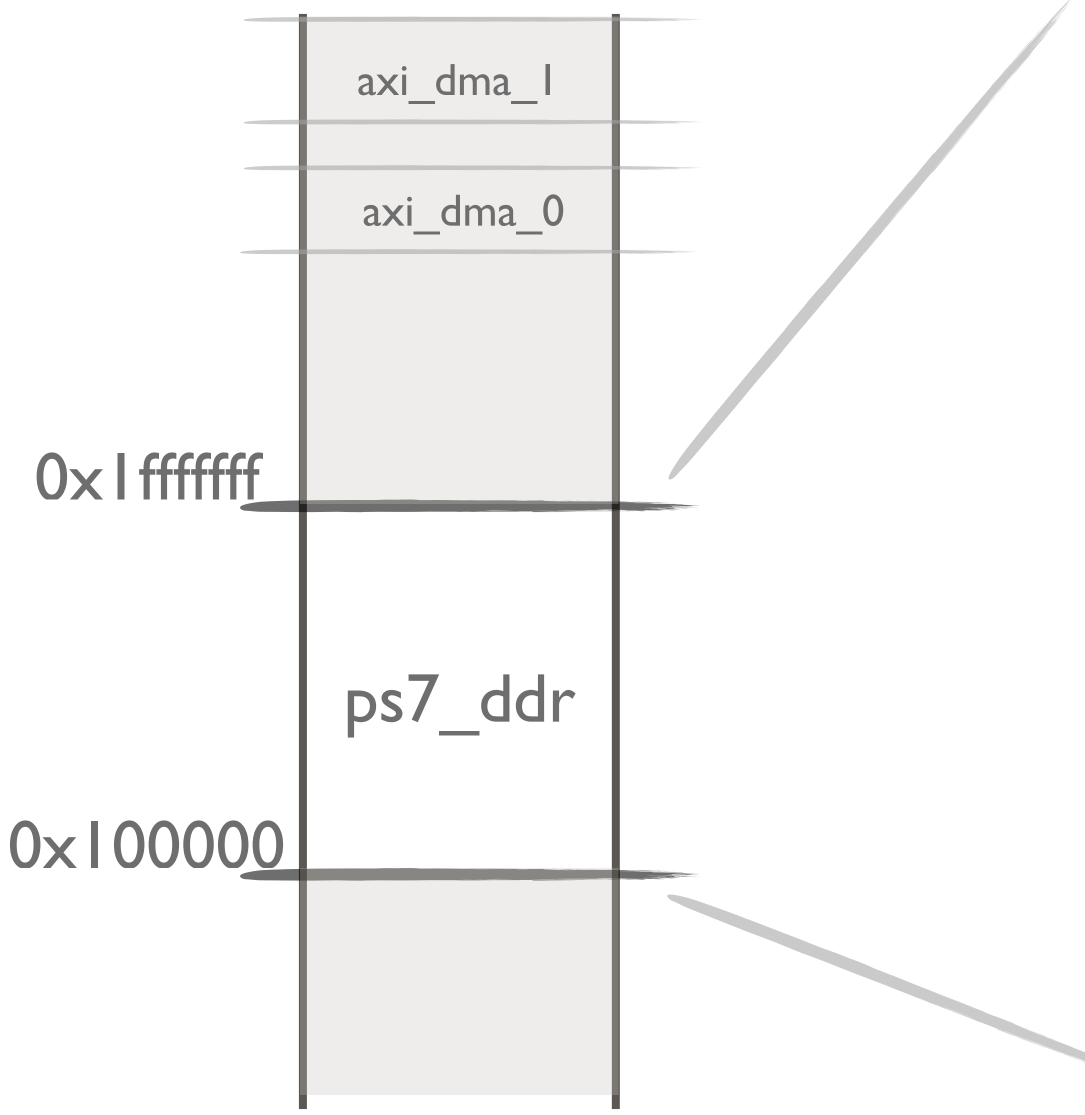
C on ARM

Port C on PC to C on ARM

data flow.



address management.



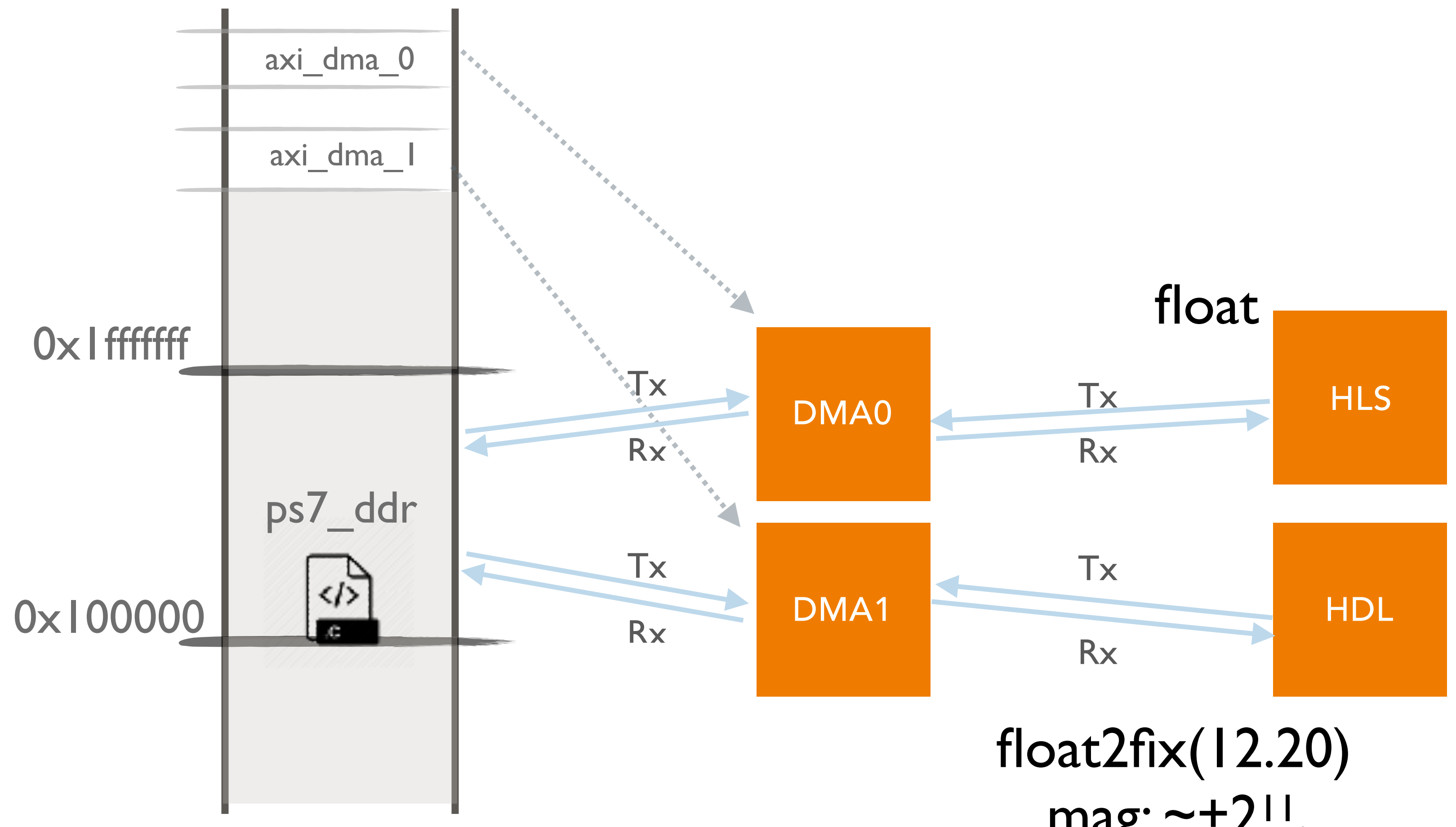
MEM_BASE_ADDR	(DDR_BASE_ADDR + 0x100000
X_TRAIN_BASE	(MEM_BASE_ADDR + 0x100000
Y_TRAIN_BASE	(X_TRAIN_BASE + 0xdf680)
b1_BASE	(W1_BASE + 0xb920)
…	…
W2_BASE	(b1_BASE + 0xf8)
b2_BASE	(W2_BASE + 0x9d0)
X_BATCH_BASE	(b2_BASE + 0x48)
X_T_BASE	(h_T_BASE + 0x4710)
grads_W2_BASE	(X_T_BASE + 0xb920)
…	…
RX_BUFFER_BASE	19800000
TX_BUFFER_BASE	19800100

!!!KEEP HARDWARE IN MIND!!!

SYSTEM OVERVIEW

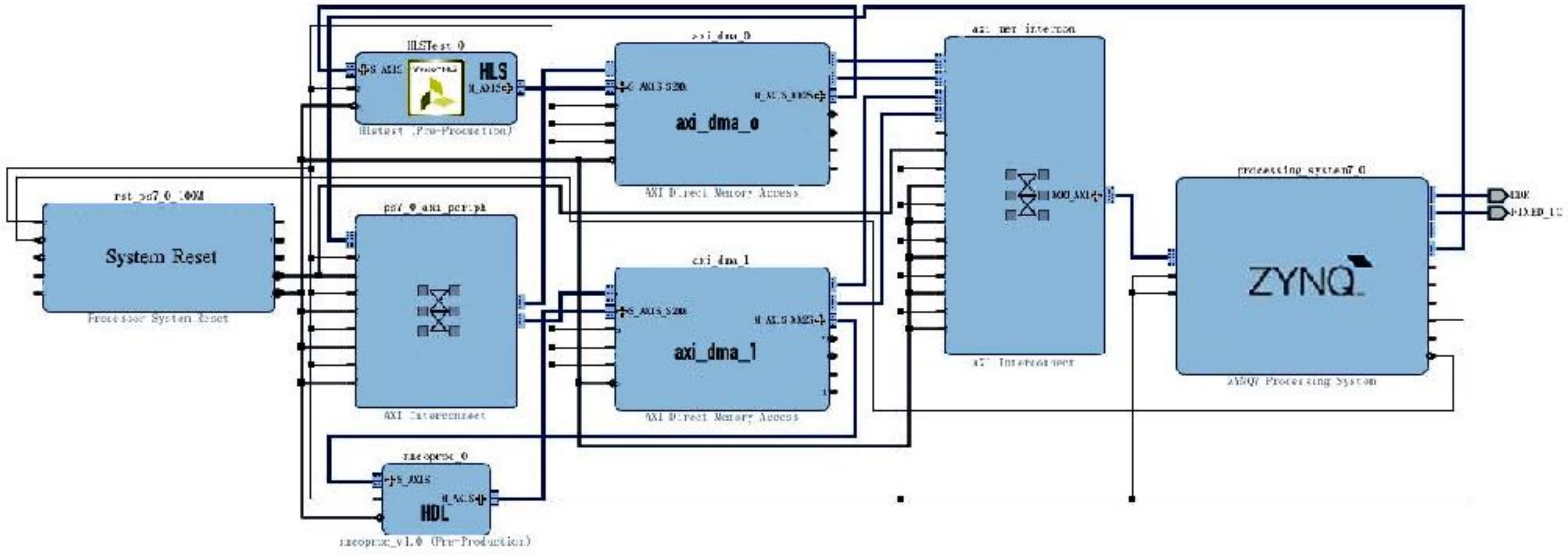
Integrated FPGA-Accelerated Inference Engine

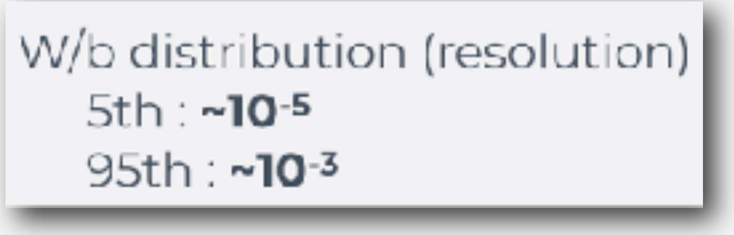
abstract structure.



float2fix(12.20)
mag: ~±2^{11},
res: 2^{-20}(~10^{-6}).

block design.

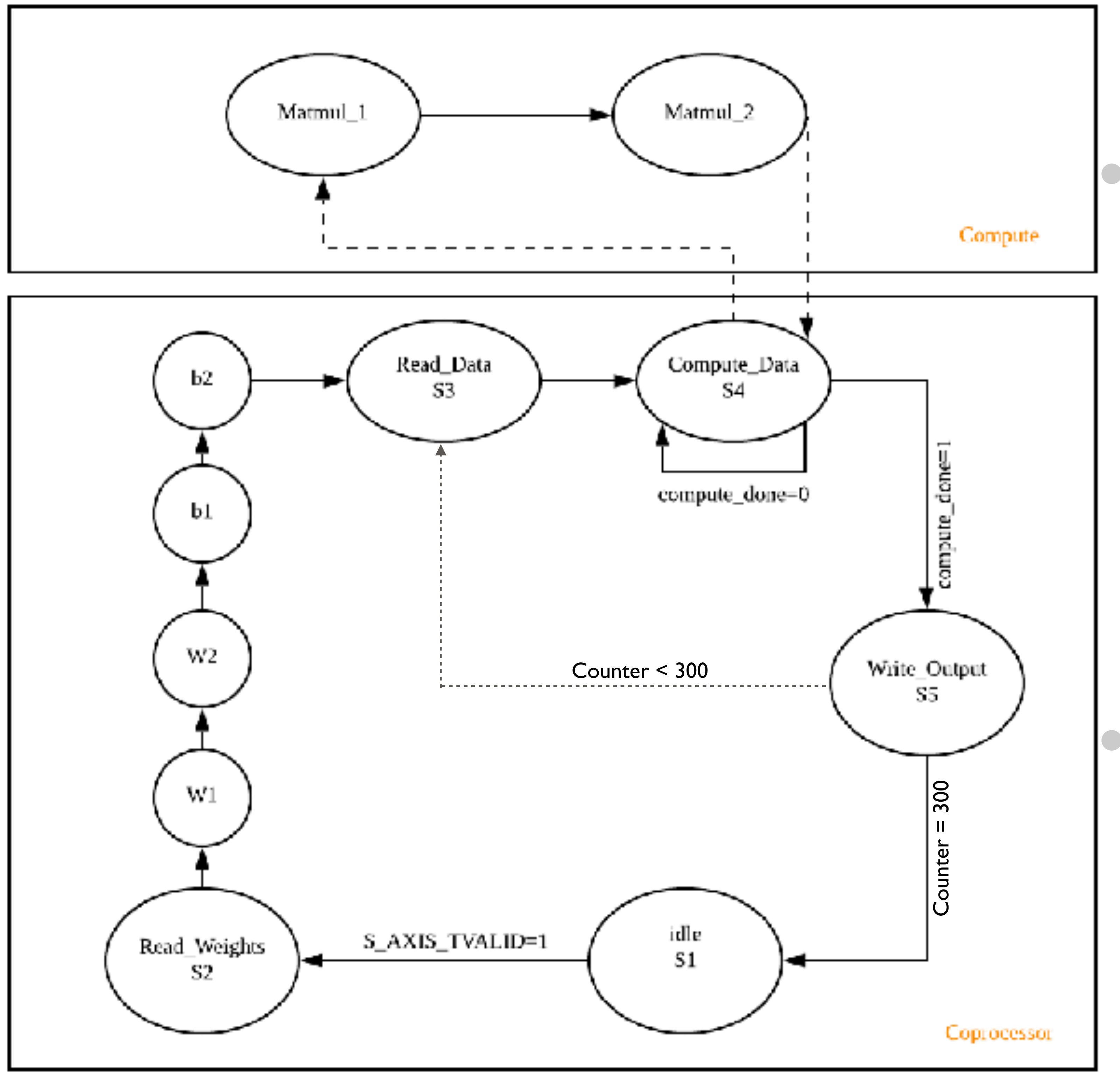


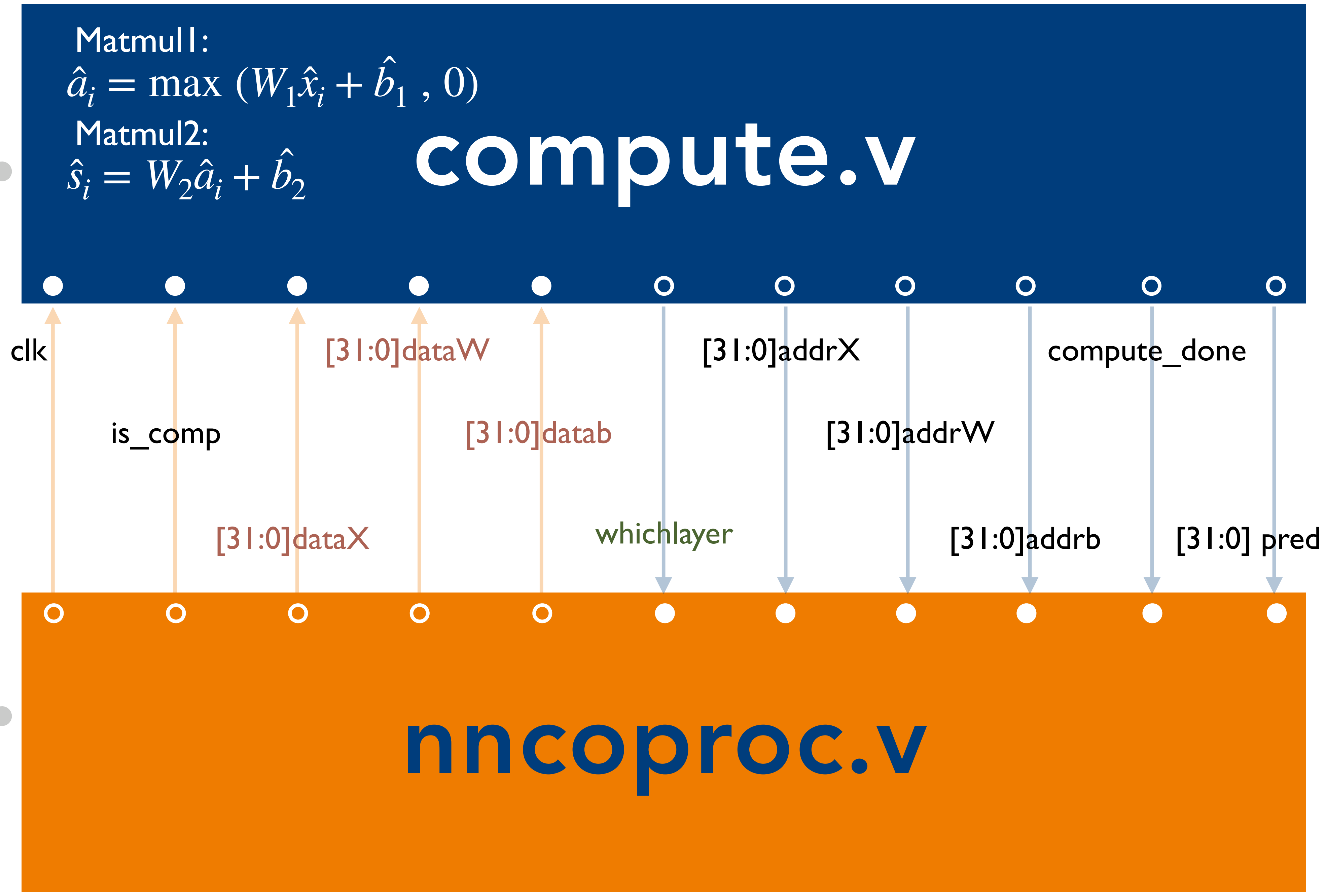


W/b distribution (resolution)
5th : ~10^{-5}
95th : ~10^{-3}

HDL DESIGN WITH SIMULATION†

State Transition Diagram & Module Interface





Matmul1:

$\hat{a}_i = \max\,(W_1\hat{x}_i + \hat{b}_1\ ,\, 0)$

Matmul2:

$\hat{s}_i = W_2\hat{a}_i + \hat{b}_2$

*Colour indicates signed

† See appendix

HLS DESIGN WITH PIPELINE

Latency and Resource Comparison

When port from ARM C: use dedicated instead of generic, for *pipelining*

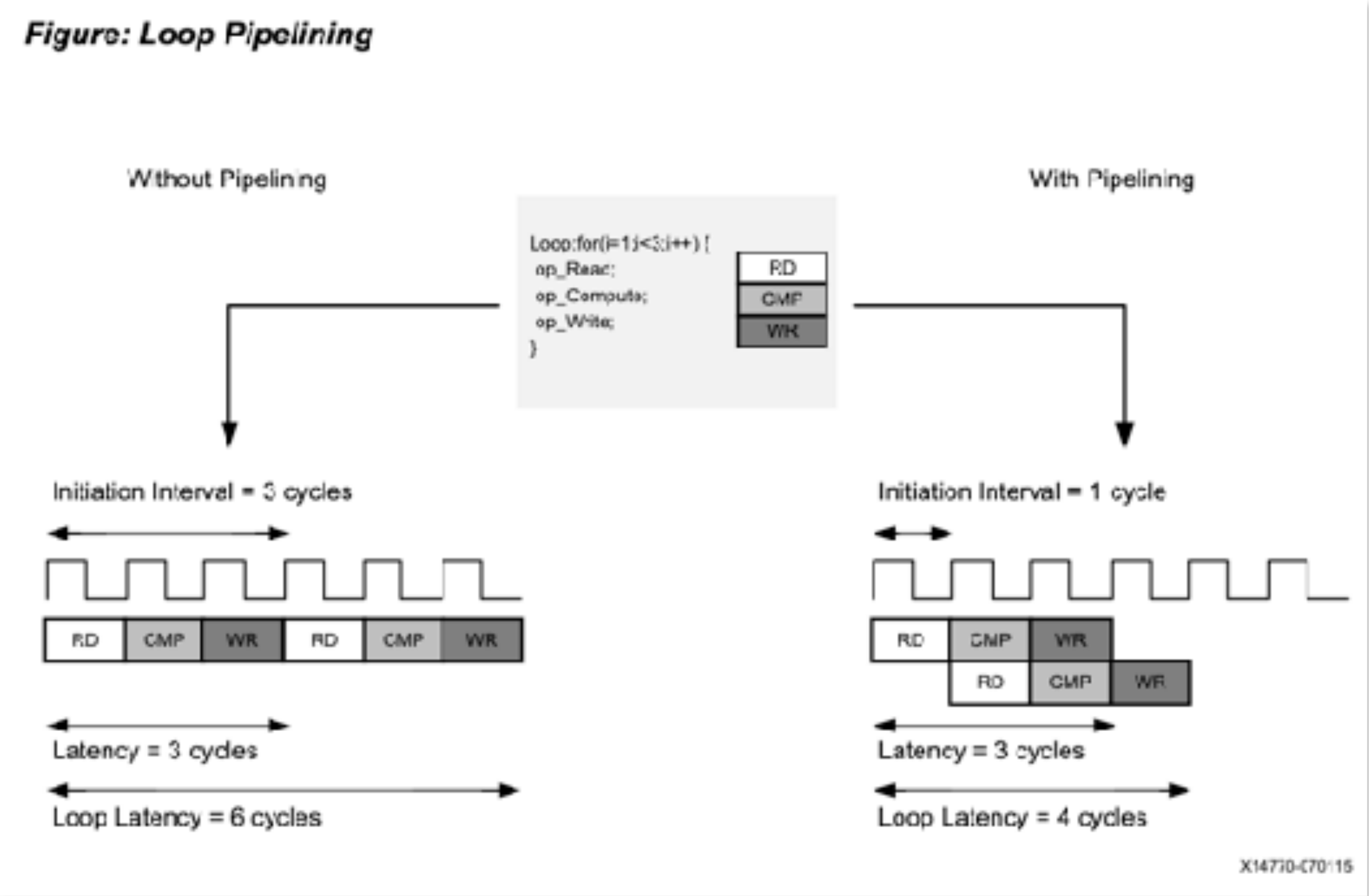
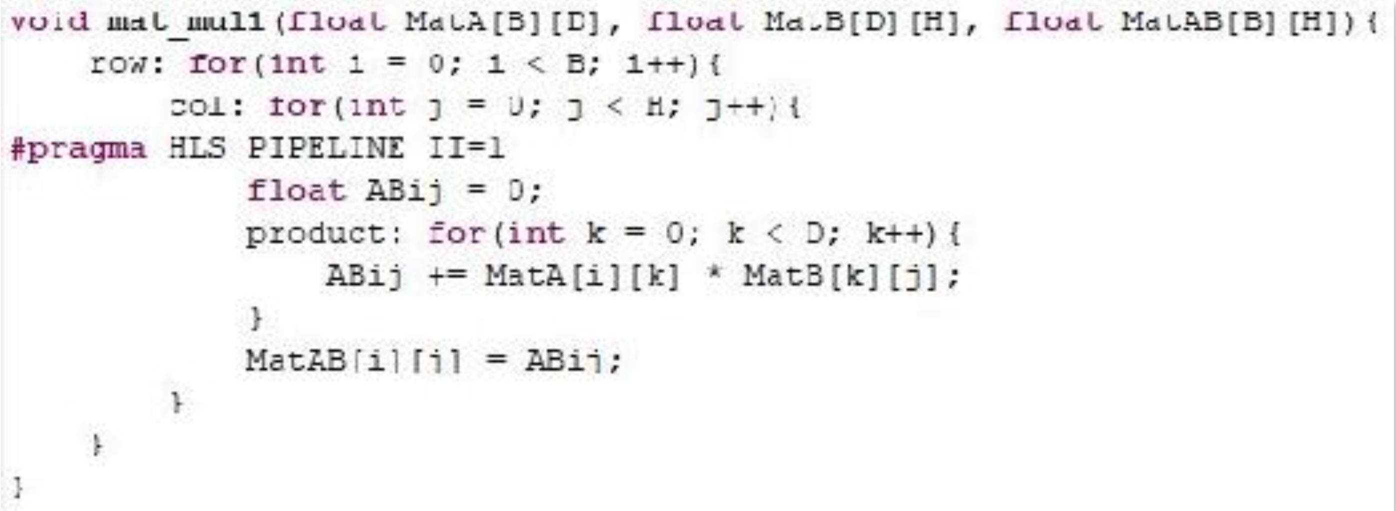


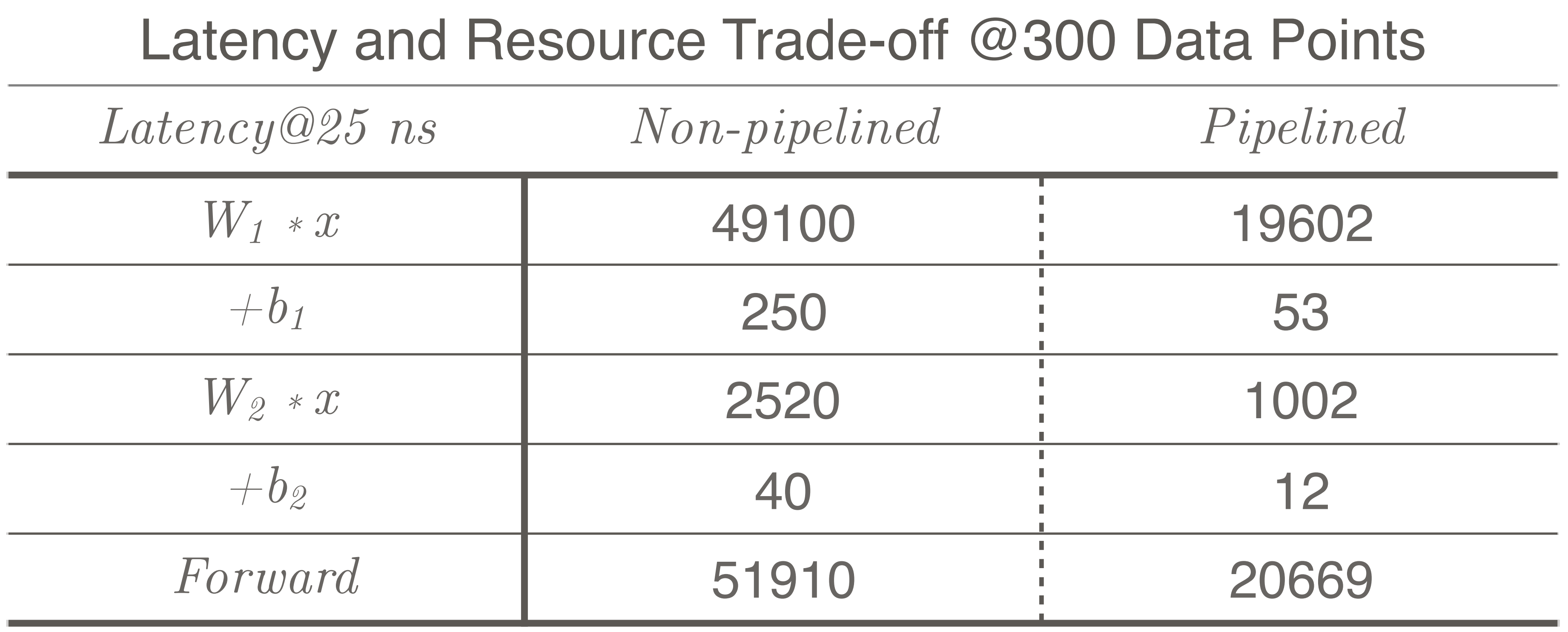
Figure: Loop Pipelining

```
Loop:for(i=1;i<3;i++) {
 op_Read;
 op_Compute;
 op_Write;
}
```

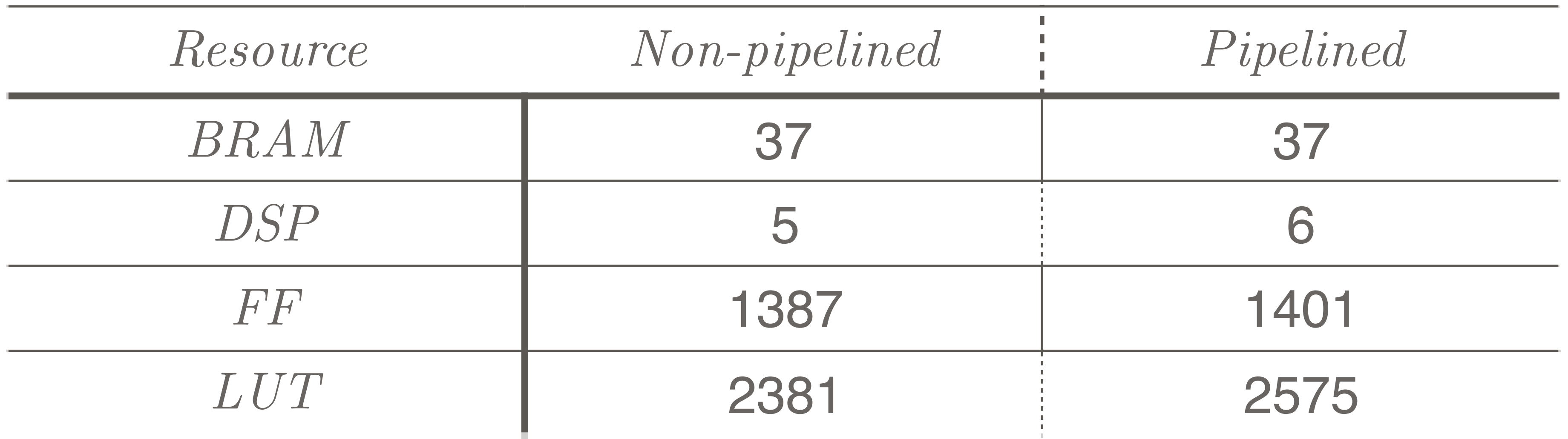


```
void mat_mul1(float MatA[B][D], float MatB[D][H], float MatAB[B][H]){
    row: for(int i = 0; i < B; i++){
        col: for(int j = 0; j < H; j++){
#pragma HLS PIPELINE II=1
            float ABij = 0;
            product: for(int k = 0; k < D; k++){
                ABij += MatA[i][k] * MatB[k][j];
            }
            MatAB[i][j] = ABij;
        }
    }
}
```

Latency and Resource Trade-off @300 Data Points



Latency@25 ns	*Non-pipelined*	*Pipelined*
$W_1 * x$	49100	19602
$+b_1$	250	53
$W_2 * x$	2520	1002
$+b_2$	40	12
Forward	51910	20669

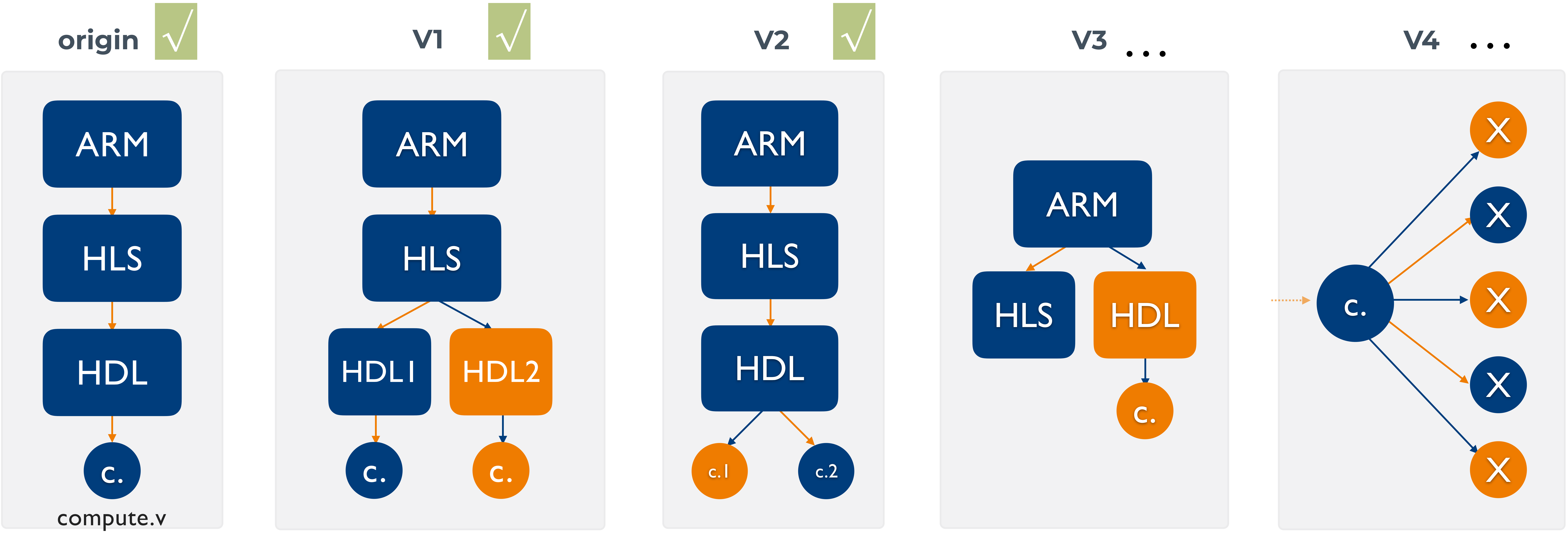


Resource	*Non-pipelined*	*Pipelined*
BRAM	37	37
DSP	5	6
FF	1387	1401
LUT	2381	2575

DATA PARALLELISM

Trade Hardware for Speed

NUS
National University
of Singapore



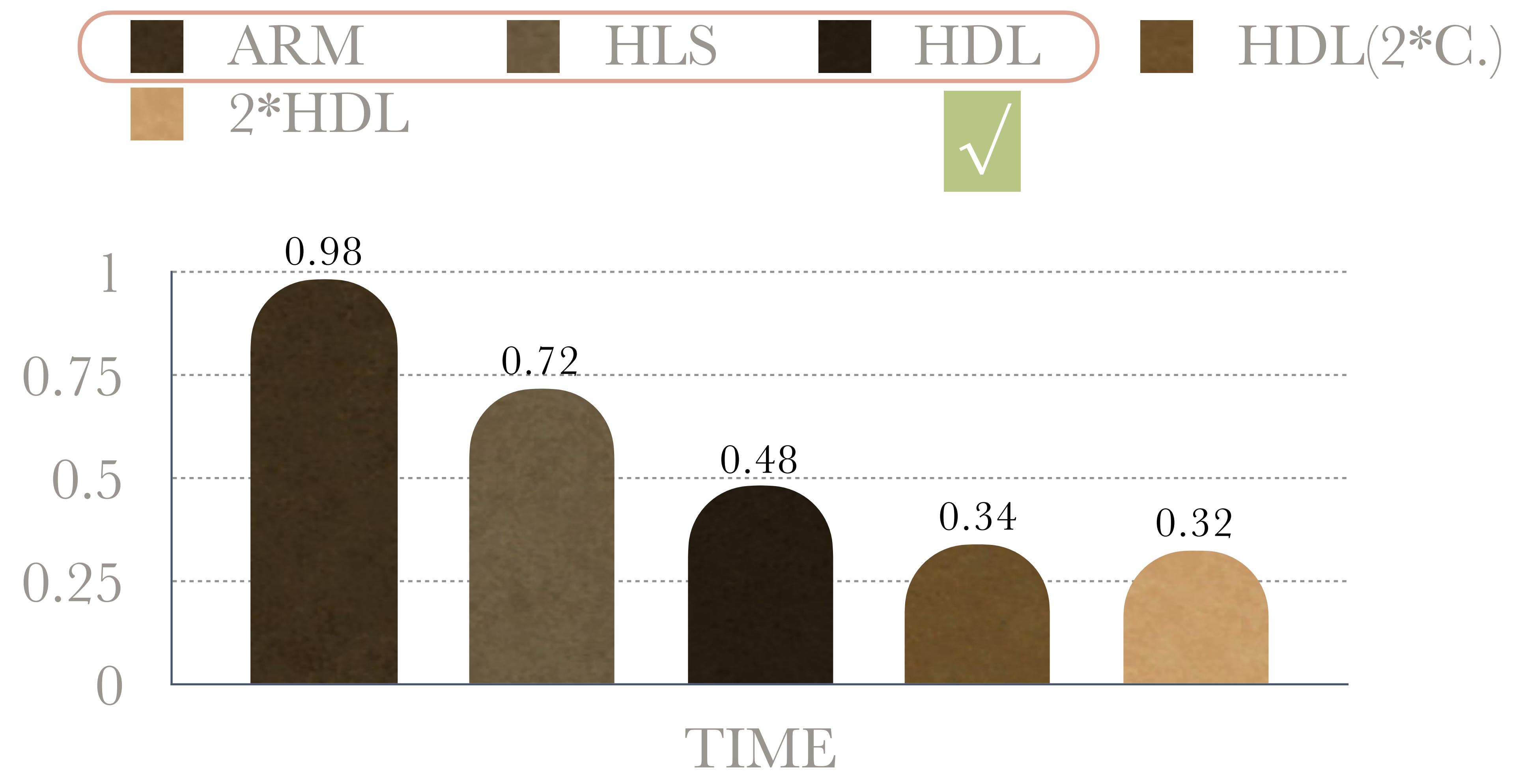
Combine V1-V4 to achieve better performance

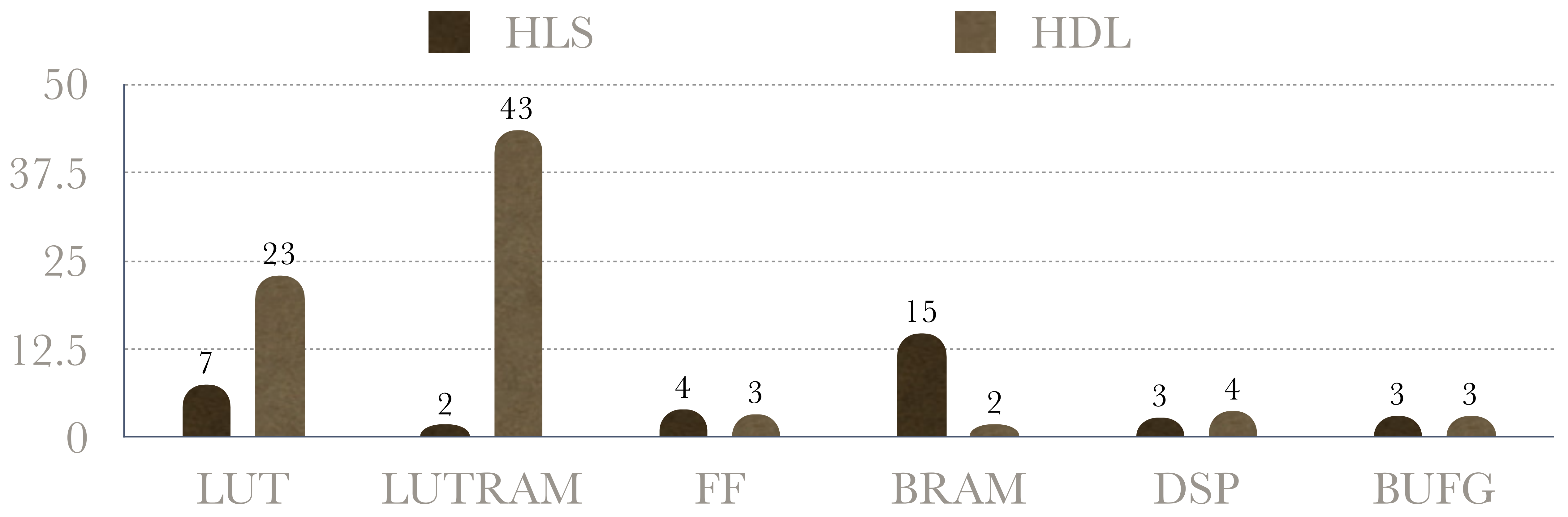
RESULT and DEMO

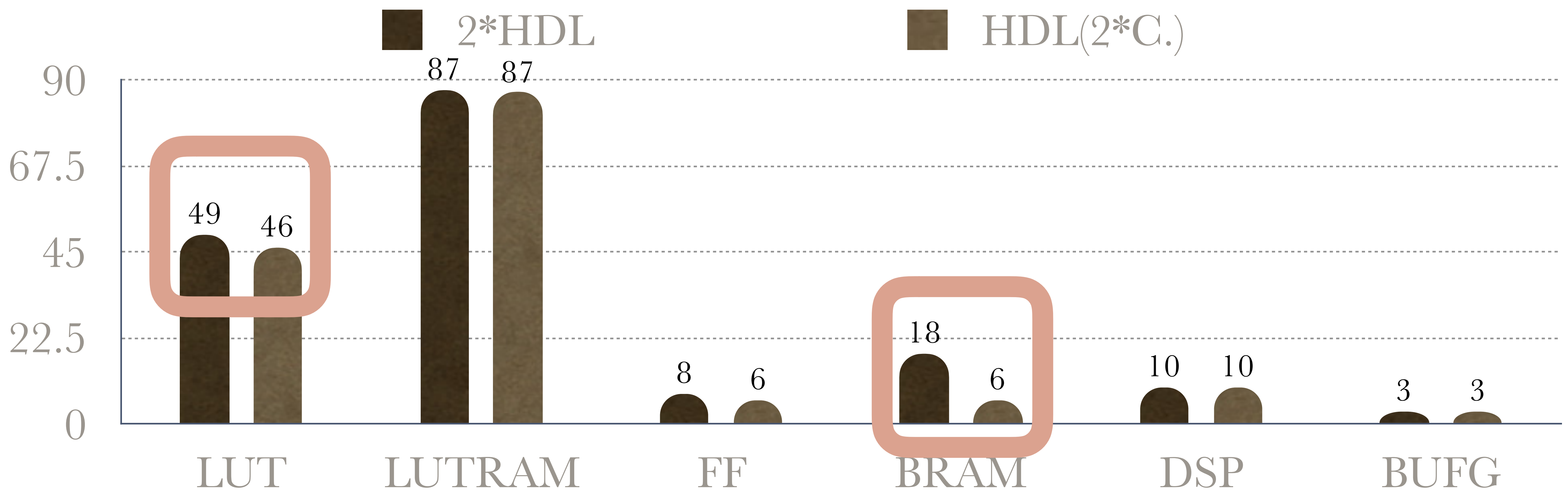
Comparison on Time, Accuracy and Resource Usage

TEST @
training data set = 700
testing data set = 300
iteration @777 iters
Accuracy @ test: 80.0%

Units:
Time: ms/data;
Others: % of total available

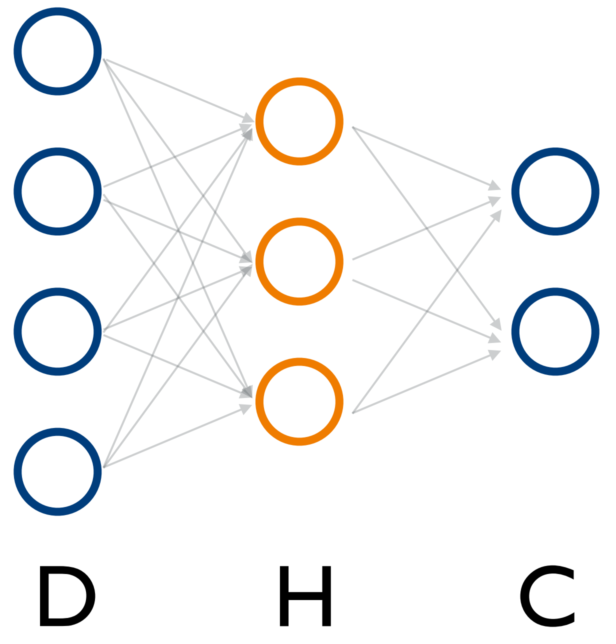


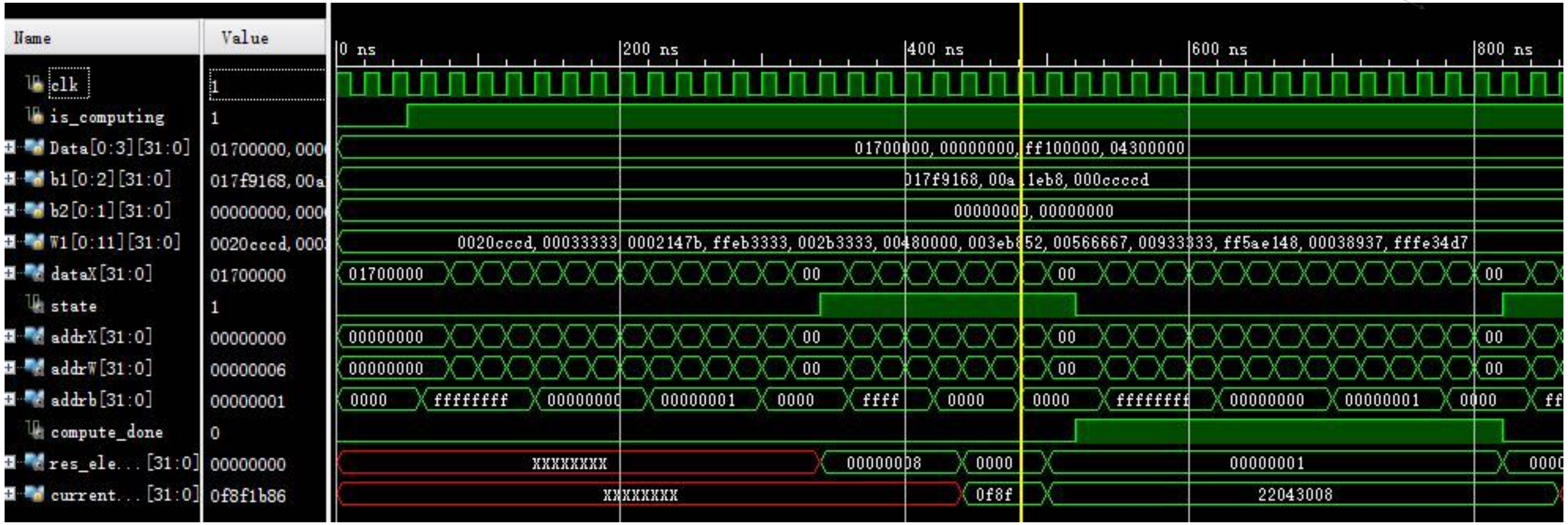




APPENDICES

Simulation (compute.v)



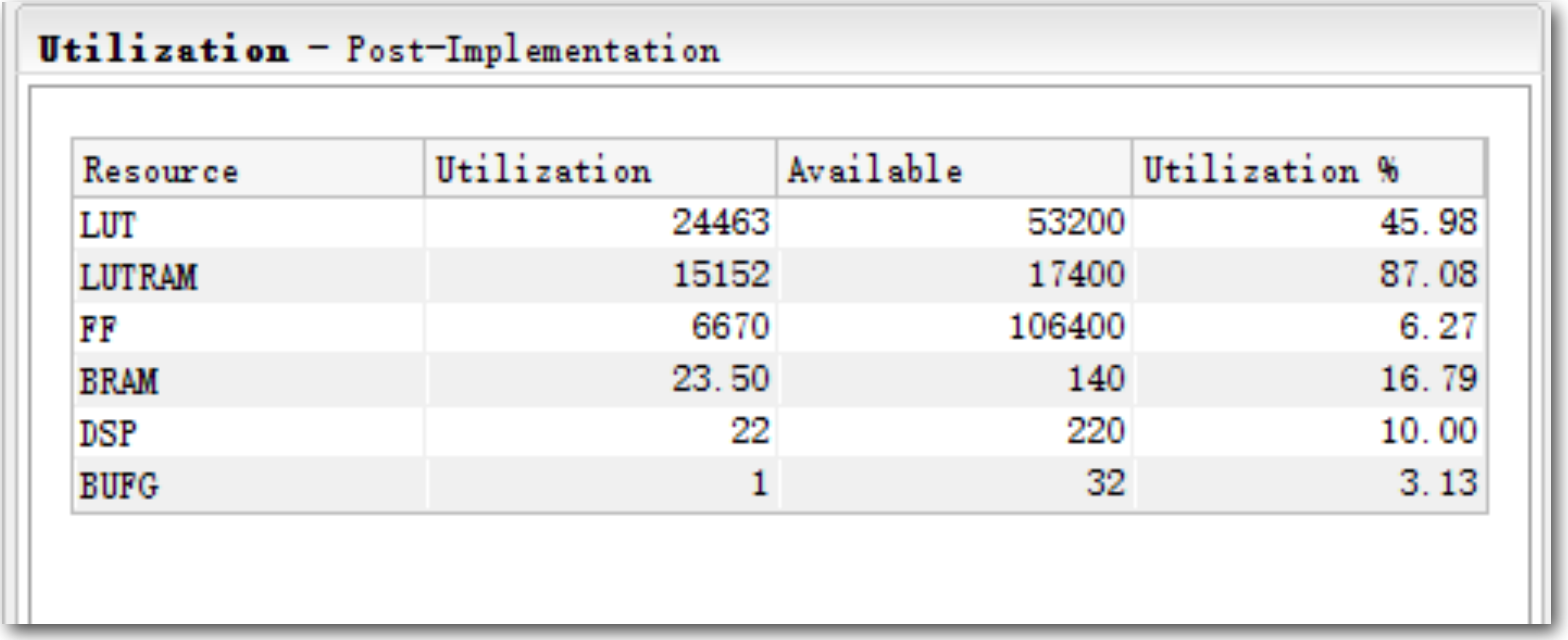


Name	Value
clk	1
is_computing	1
Data[0:3][31:0]	01700000, 000
b1[0:2][31:0]	017f9168, 00a
b2[0:1][31:0]	00000000, 000
W1[0:11][31:0]	0020cccd, 000
dataX[31:0]	01700000
state	1
addrX[31:0]	00000000
addrW[31:0]	00000006
addrb[31:0]	00000001
compute_done	0
res_ele...[31:0]	00000000
current...[31:0]	0f8f1b86

	Class	Score 1	Score 2
Actual Result (fix-point)	1	0f8.f1b86	220.43008
Expected Result (fix-point)	1	0f8.f1bb0	220.43080
Expected Result (float)	1	248.9442605	544.261841

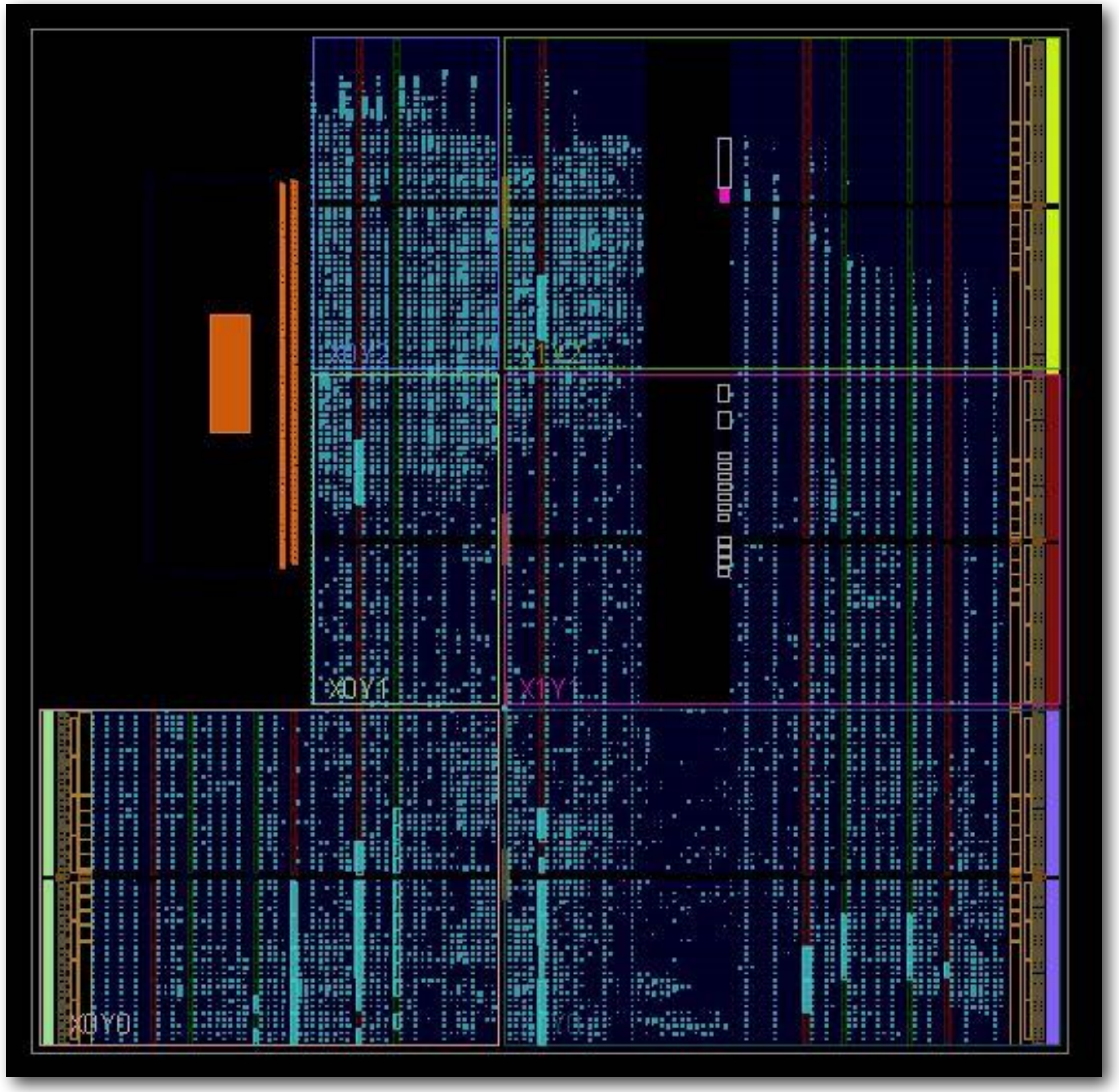
APPENDICES

Resource for Complete System: Pipelined HLS and DataParallel HDL.



Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	24463	53200	45.98
LUTRAM	15152	17400	87.08
FF	6670	106400	6.27
BRAM	23.50	140	16.79
DSP	22	220	10.00
BUFG	1	32	3.13



DEBRIEF

Some Take-aways

- When port from PC C to ARM C: allocate memory with caution and **don't** expect heap (global) or stack (local) is enough.

- When port from ARM C to HLS CPP: use dedicated functions with fixed i/o size instead of generic, re-usable functions, for *pipelining*.

- When writing HDL: start small and scale up, simulate before implement. If use asynchronous data read, usually we infer Distributed RAM (combinational), which make the critical path longer and slow down the system frequency; on the other had, if use synchronous data read, usually Block RAM is inferred and the critical path will be shorter.

- Timing report will show the failed endpoints, which, at the same time, indicates our critical path. We can use multi-cycle-path to deal with it, or consider make it synchronous.