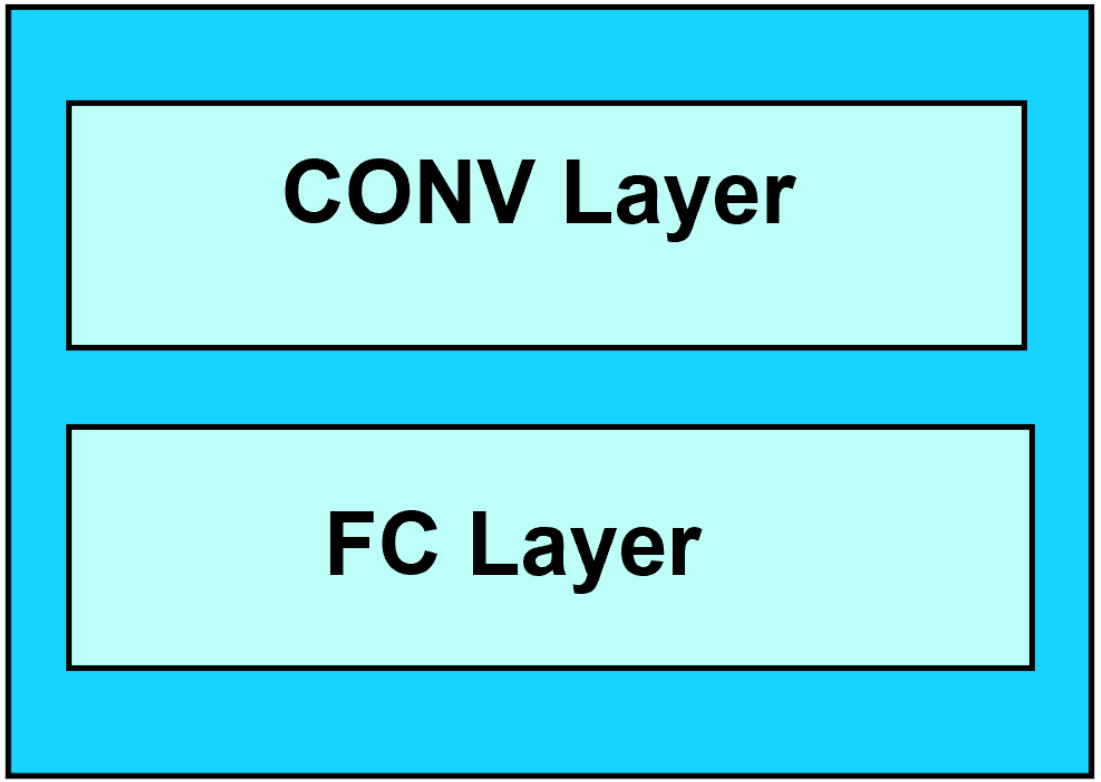
Week 2: A Big Picture?

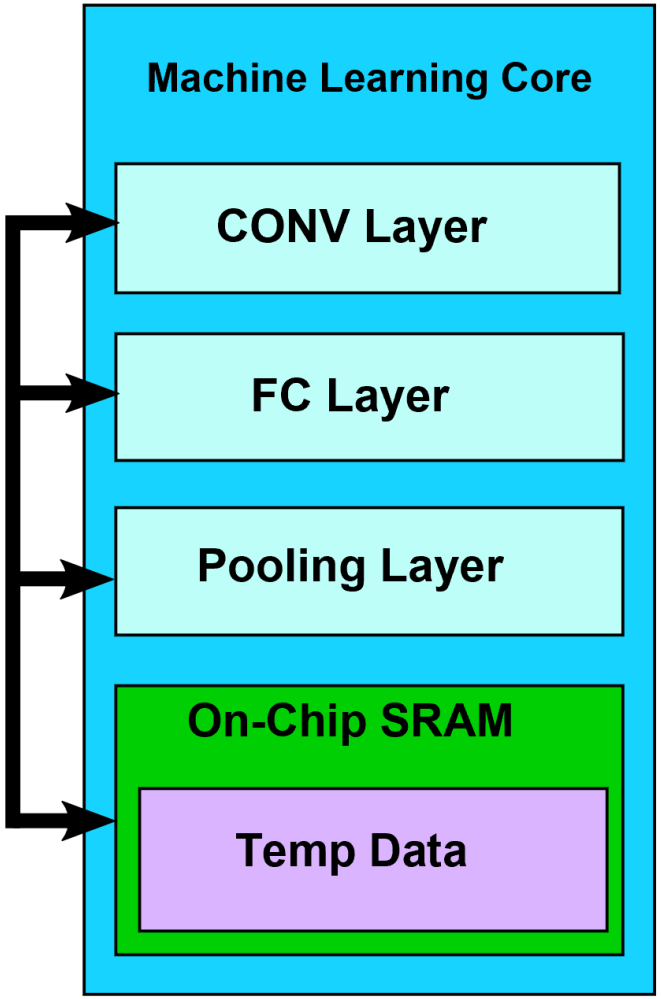
Last Week:

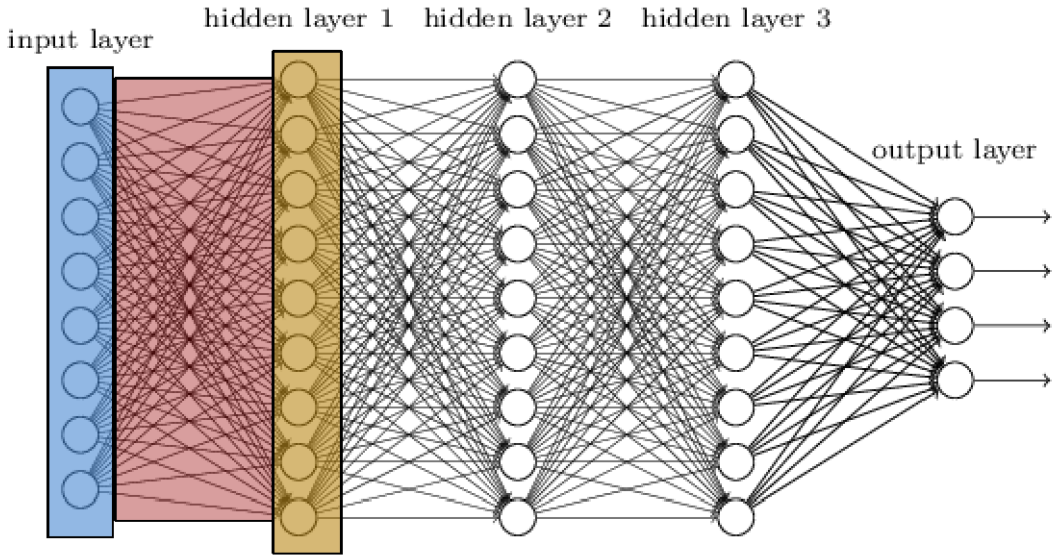


Questions:

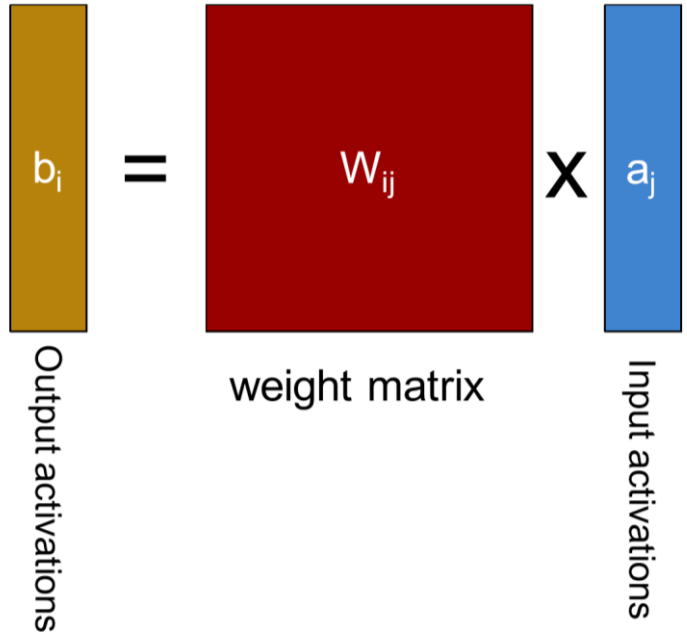
1. ***Why are they important?***
2. ***What are we doing?***
3. ***How are we going to use them?***
4. ***Give me 3 other solutions? (Can we do better?)***

Why they are important:

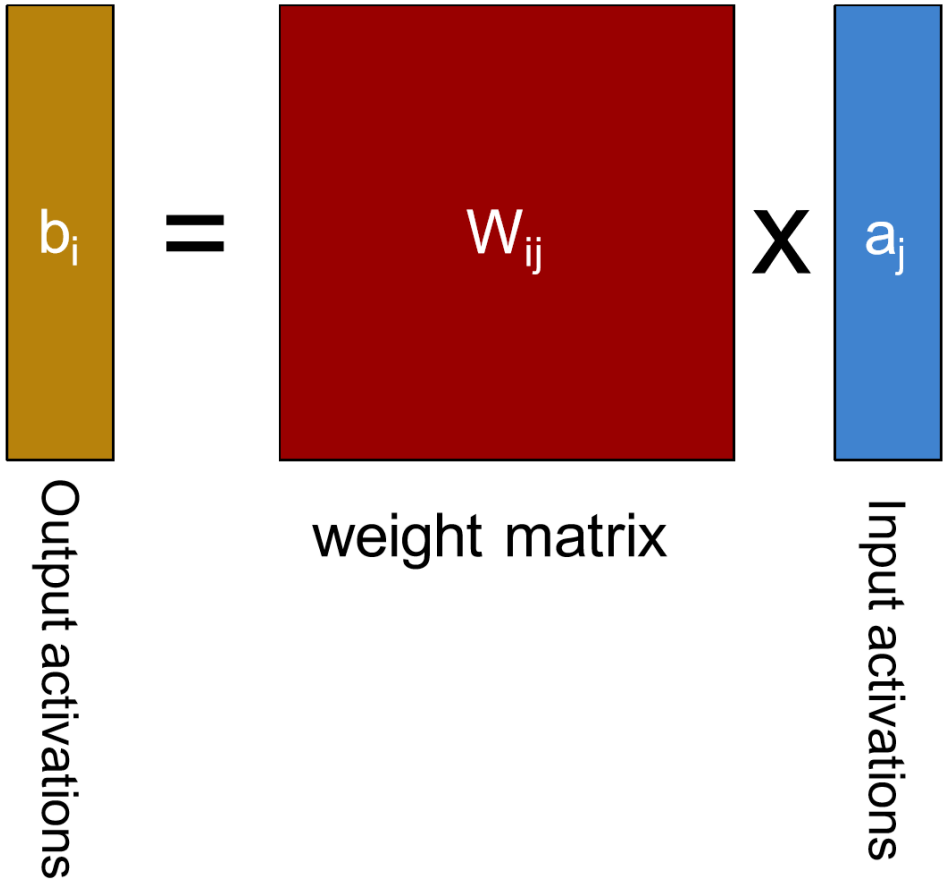


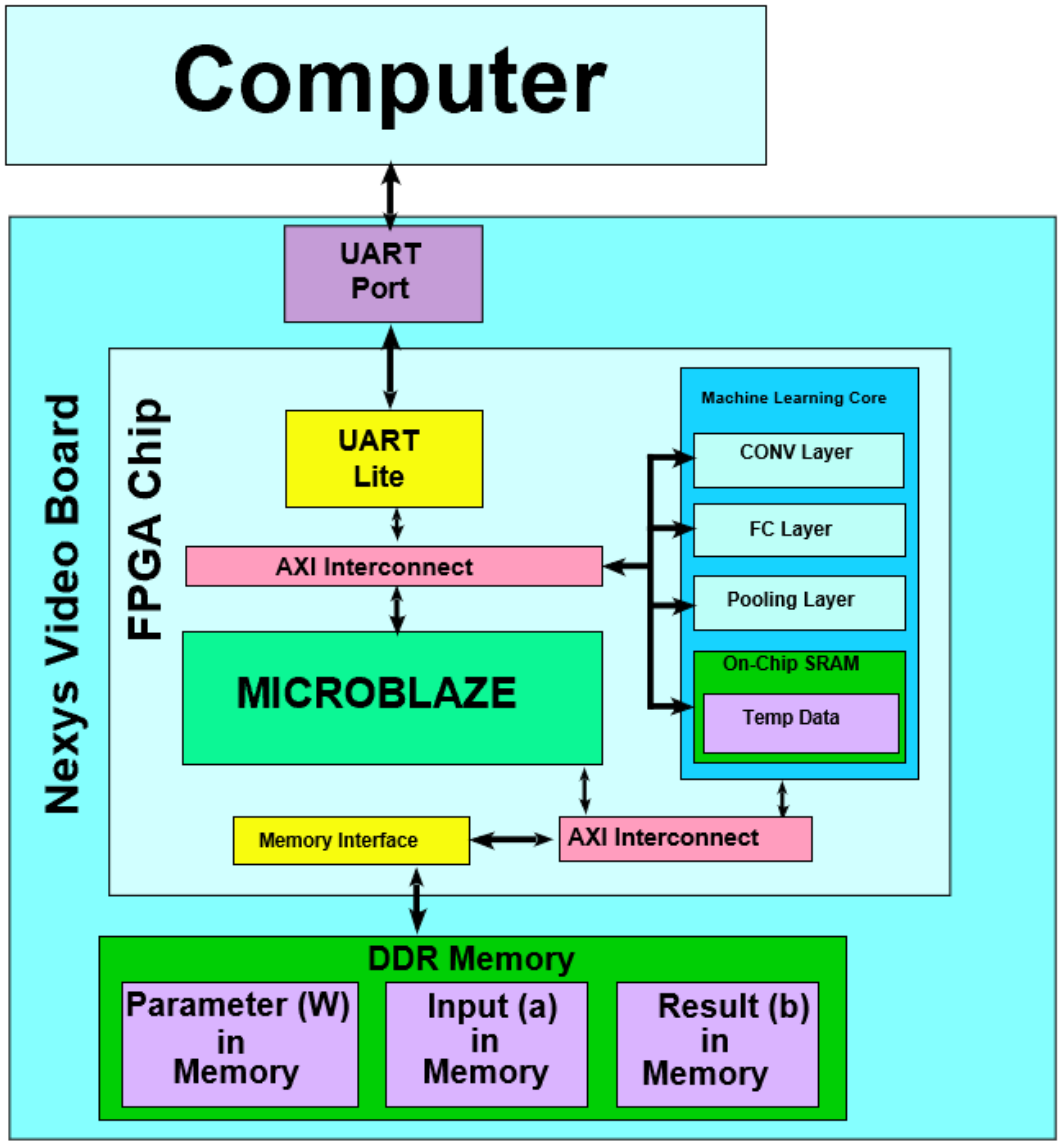


Key Operation is Dense M×V

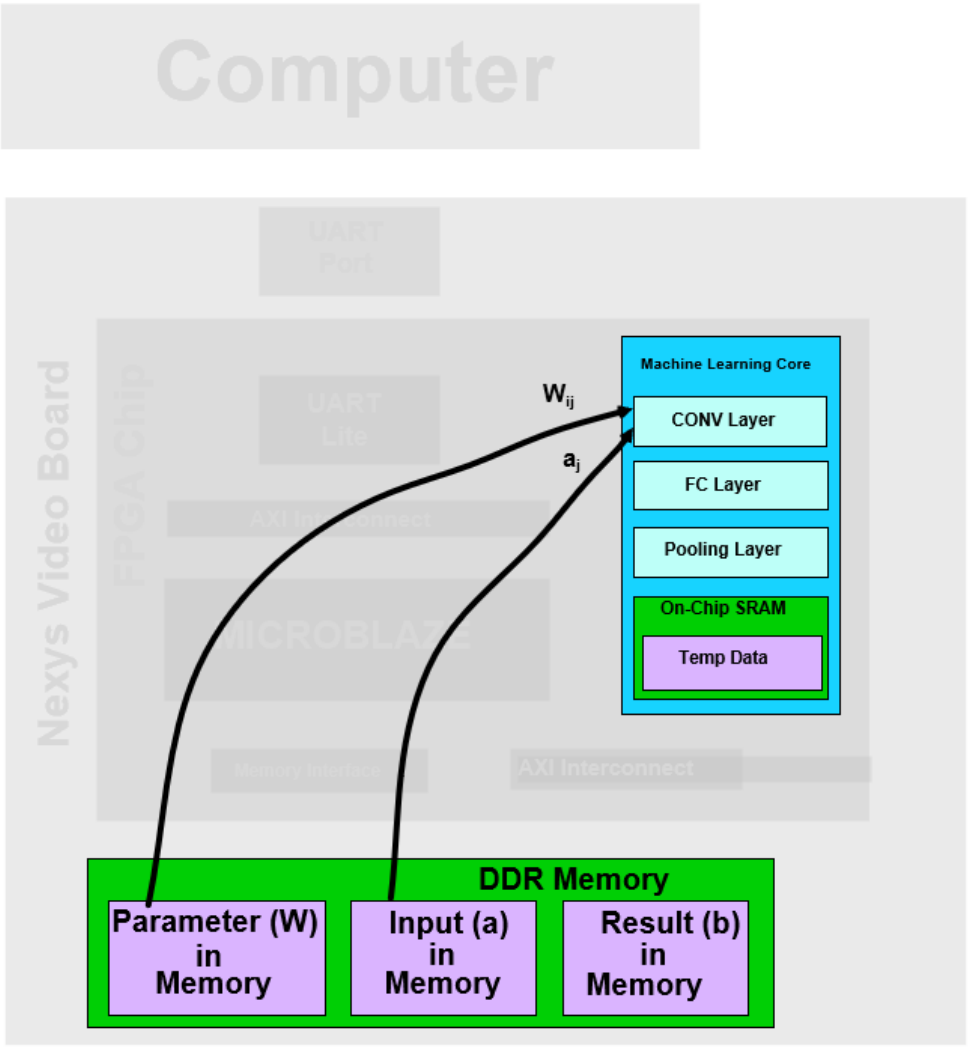


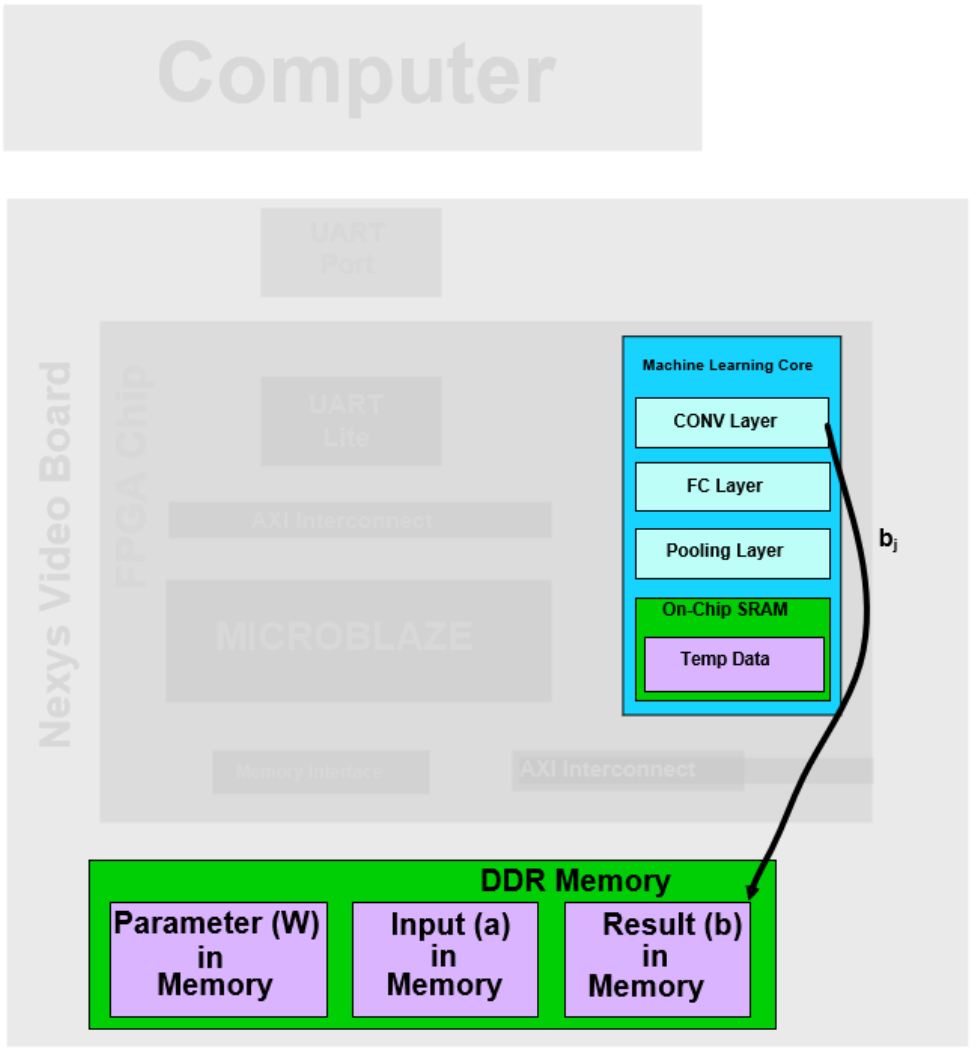
What we are doing:



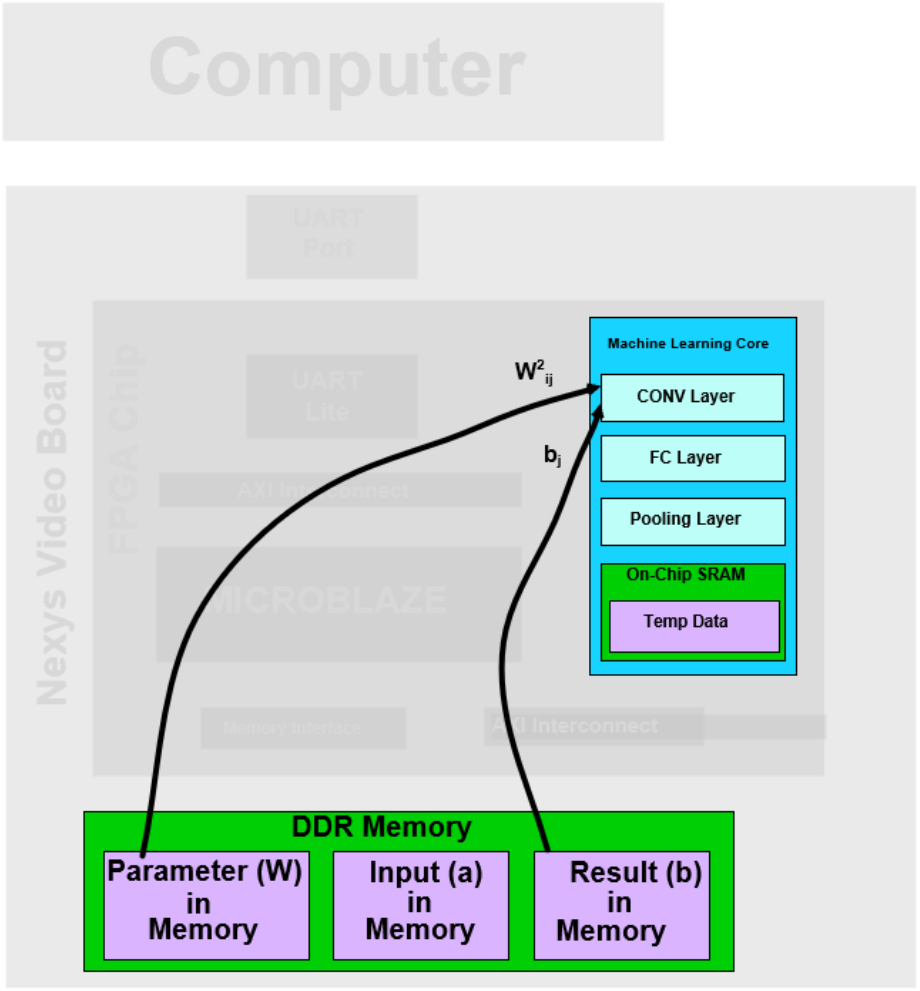


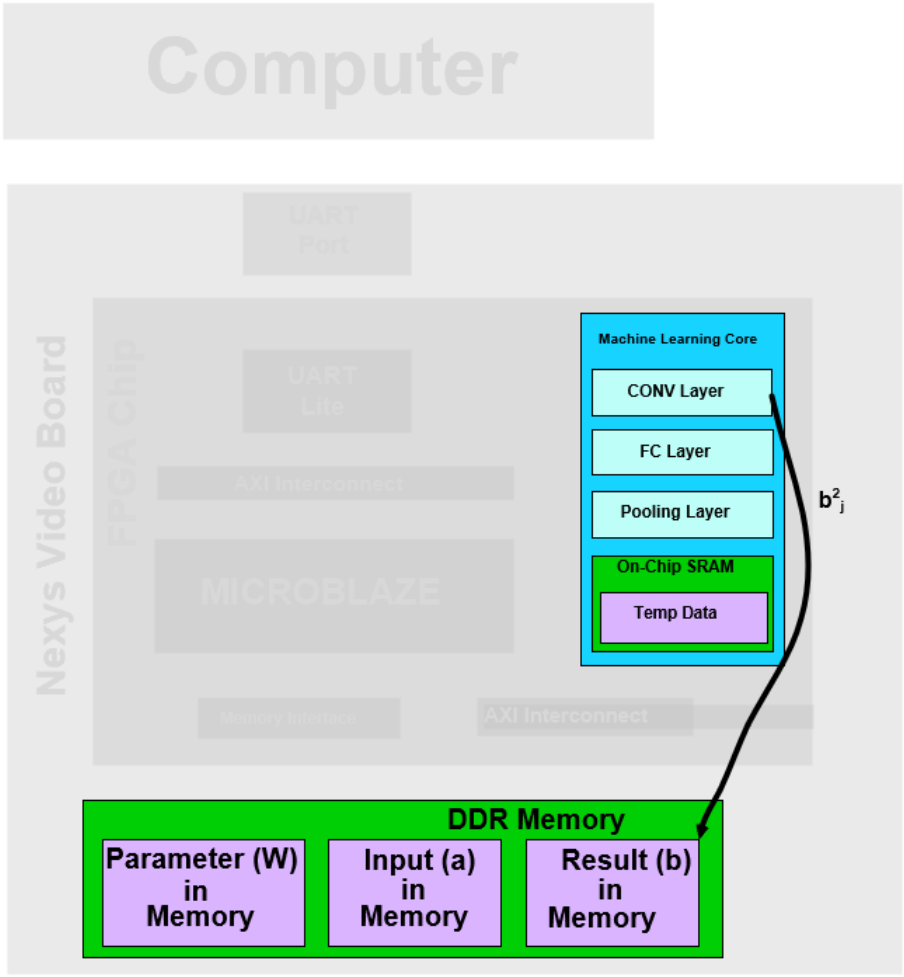
What we are doing:



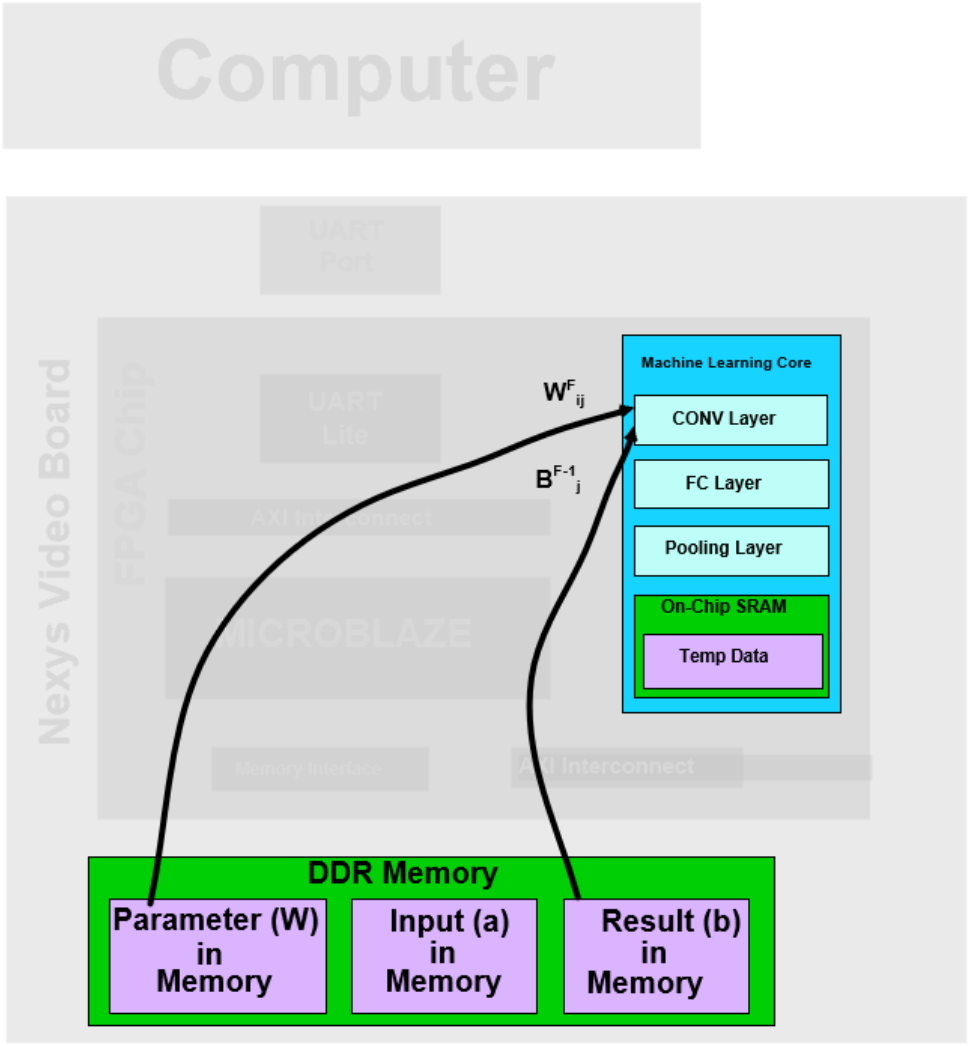


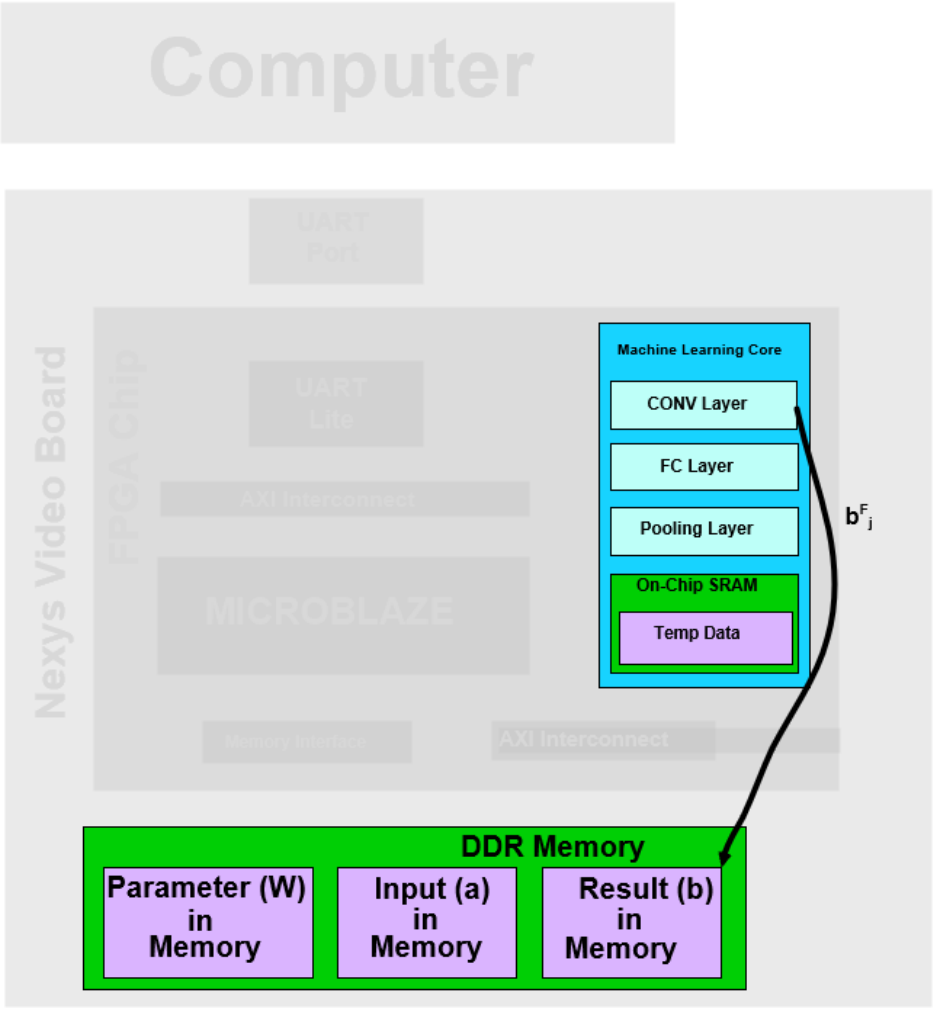
What we are doing:





What we are doing:





How we are using them:

Synthesis ‘C/C++’ to RTL.

Create blocks, Gate level design

Program Microprocessor such that it can interface with different IPs.

How we are using them:

VIVADO™

Micro-Blaze

ML Core

Mem Interface

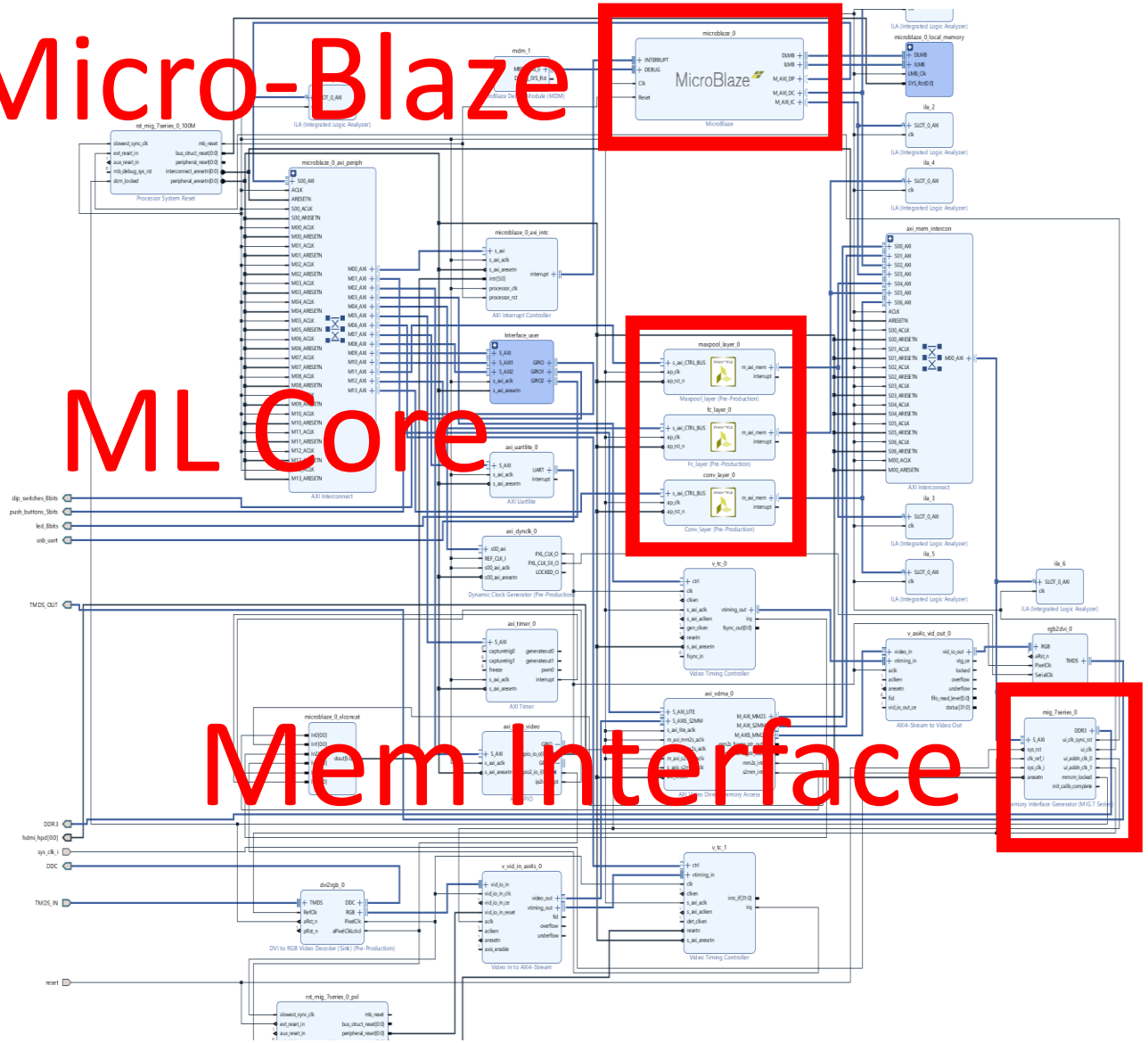


Diagram × **Address Editor** ×

Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
maxpool_layer_0					
Data_m_axi_mem (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF
conv_layer_0					
Data_m_axi_mem (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF
fc_layer_0					
Data_m_axi_mem (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF
axi_vdma_0					
Data_MM2S (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF
Data_S2MM (32 address bits : 4G)					
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF
microblaze_0					
Data (32 address bits : 4G)					
axi_dynclk_0	s00_axi	reg0	0x44A2_0000	64K	0x44A2_FFFF
Interface_user/axi_gpio_SW	S_AXI	Reg	0x4002_0000	64K	0x4002_FFFF
Interface_user/axi_gpio_Push	S_AXI	Reg	0x4003_0000	64K	0x4003_FFFF
Interface_user/axi_gpio_LED	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
axi_gpio_video	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
axi_timer_0	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
axi_uartlite_0	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
conv_layer_0	s_axi_CTRL_BUS	Reg	0x44A4_0000	64K	0x44A4_FFFF
fc_layer_0	s_axi_CTRL_BUS	Reg	0x44A5_0000	64K	0x44A5_FFFF
maxpool_layer_0	s_axi_CTRL_BUS	Reg	0x44A6_0000	64K	0x44A6_FFFF
microblaze_0_axi_intc	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF
v_tc_0	ctrl	Reg	0x44A1_0000	64K	0x44A1_FFFF
v_tc_1	ctrl	Reg	0x44A3_0000	64K	0x44A3_FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	32K	0x0000_7FFF
mig_7series_0	S_AXI	memaddr	0x8000_0000	512M	0x9FFF_FFFF

How we are using them:

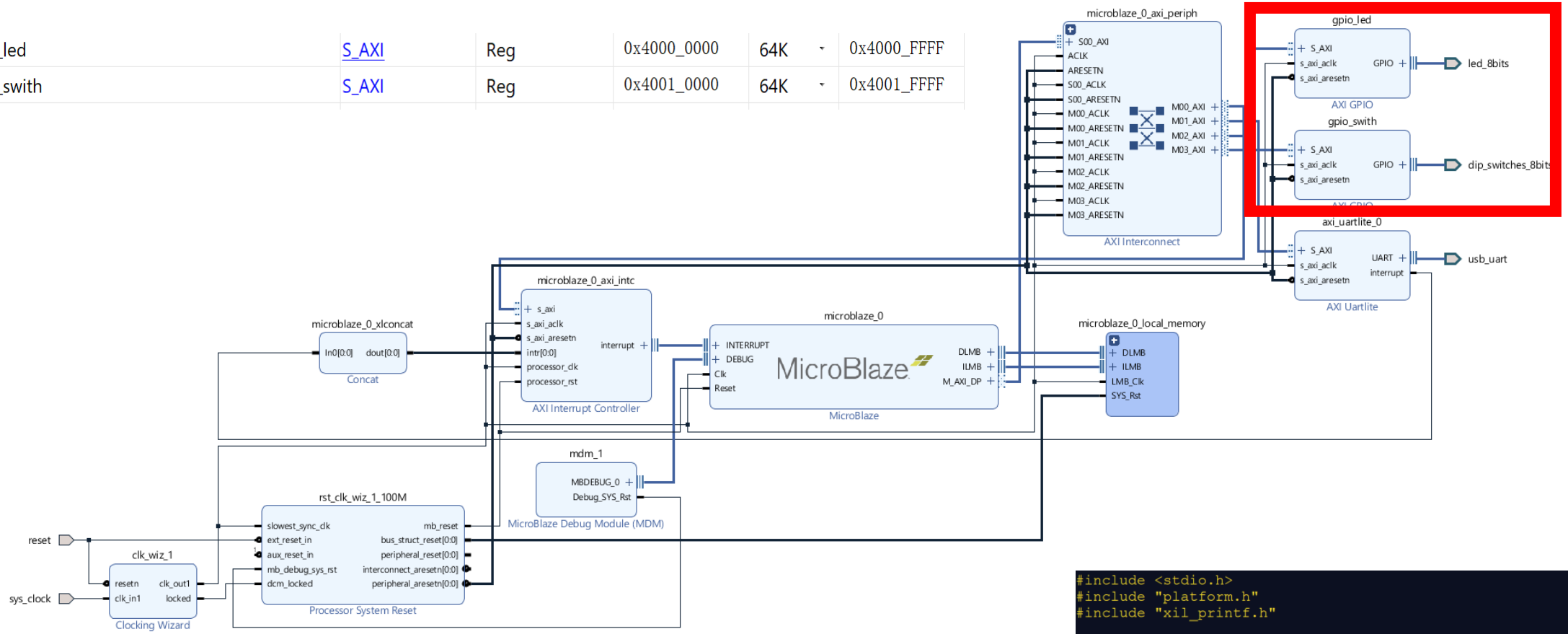

```
void hw_hls_core_processing ()
{
    printf("start hls core processing \n");
    //layer 1 - conv_1
       hw_conv_layer(2415919104, 2432696320,  1, 28, 28, 20, 24, 24, 1, 5);      //
       input offset 0x90000000 , output offset 0x91000000
    //layer 2 -pool_1
    hw_maxpool_layer(2432696320, 2449573736, 20, 24, 24, 20, 12, 12, 2, 2);      //
    input offset 0x91000000 , output offset 0x92018768  (conv2 bias + weight = 50
    + 25000 = 25050 = 0x18768)
    //layer 3 - conv_2
       hw_conv_layer(2449473536, 2466250752, 20, 12, 12, 50,  8,  8, 1, 5);      //
       input offset 0x92000000 , output offset 0x93000000
    //layer 4 -pool_2
    hw_maxpool_layer(2466250752, 2484629968, 50,  8,  8, 50,  4,  4, 2, 2);      //
    input offset 0x93000000 , output offset 0x941871D0  (fc1 bias + weight =500 +
    400000 = 400500 = 0x1871D0)
    //layer 5 -fc_1
        hw_fc_layer(2483027968, 2499825224, 800, 500, 1);                        //
        input offset 0x94000000 , output offset 0x95004E48  (fc2 bias + weight
        =10 + 5000 = 5010 = 0x4E48)
    //layer 6 -fc_2
        hw_fc_layer(2499805184, 2516582400, 500, 10, 0);                         //
        input offset 0x95000000 , output offset 0x96000000
    printf("done hls core processing \n");
}
```

Vivado™ HLS

```
#include <algorithm>
#include "fc_layer.h"

void fc_layer(float * mem,
              int input_offset,
              int output_offset,
              const int batch_size,
              const int num_inputs,
              const int num_outputs,
              const int enable_relu)
{
#pragma HLS INTERFACE m_axi port=mem depth=2147483648
#pragma HLS INTERFACE s_axilite port=input_offset bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=output_offset bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=batch_size bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=num_inputs bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=num_outputs bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=enable_relu bundle=CTRL_BUS
#pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS

  const int num_weights = num_inputs*num_outputs;
  const int num_biases =  num_outputs;


  for (int b = 0; b < batch_size; b++) {

    // Output Node Iterator
    for (int o = 0; o < num_outputs; o++) {

      // Set bias
      float output_element = mem[input_offset/sizeof(float) + num_weights + o];

      // Accumulate weighted sum
      for (int i = 0; i < num_inputs; i = i+1) {
//#pragma HLS PIPELINE
        float input_element = mem[input_offset/sizeof(float) + num_weights +
        num_biases + b*num_inputs+i];
        //if (input_element != 0.0f){
            float weight_element = mem[input_offset/sizeof(float) + o*num_inputs+i];
        //    if (weight_element != 0.0f){
                output_element += input_element * weight_element;
        //    }
        //}

        //float input_element = mem[input_offset/sizeof(float) + num_weights +
        num_biases + b*num_inputs+i];
        //float weight_element = mem[input_offset/sizeof(float) + o*num_inputs+i];
        //output_element += input_element * weight_element;

      }

      // Compute activation
      if (enable_relu)
        mem[output_offset/sizeof(float) + b*num_outputs+o] = std::max(0.0f,
        output_element);
      else
        mem[output_offset/sizeof(float) + b*num_outputs+o] = output_element;
    }
  }
}
```

Next Week Tutorial:

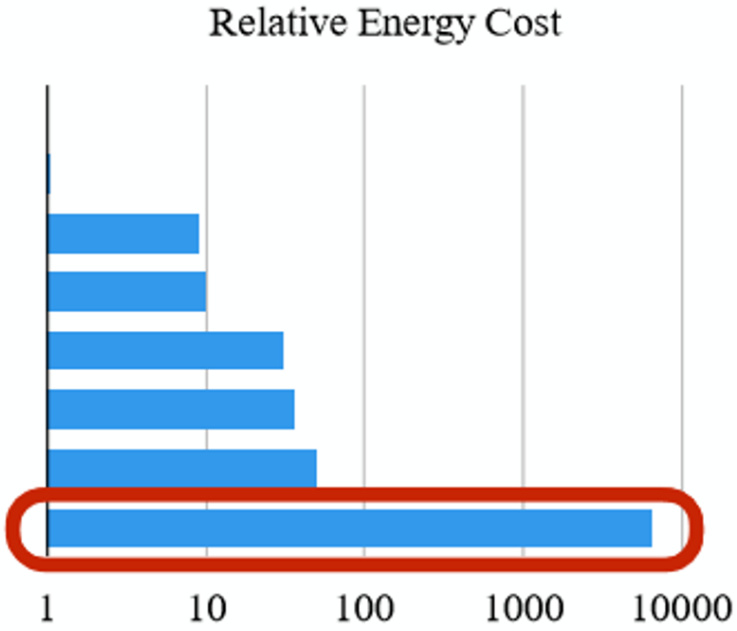
gpio_led	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
gpio_swith	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF

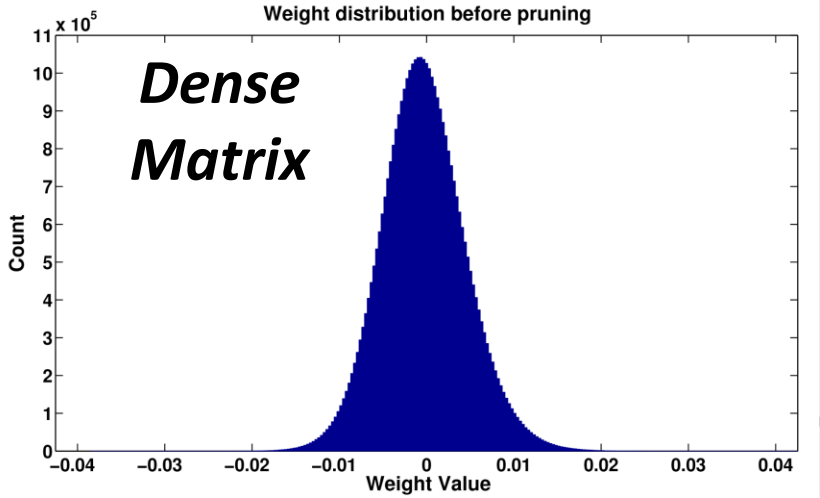


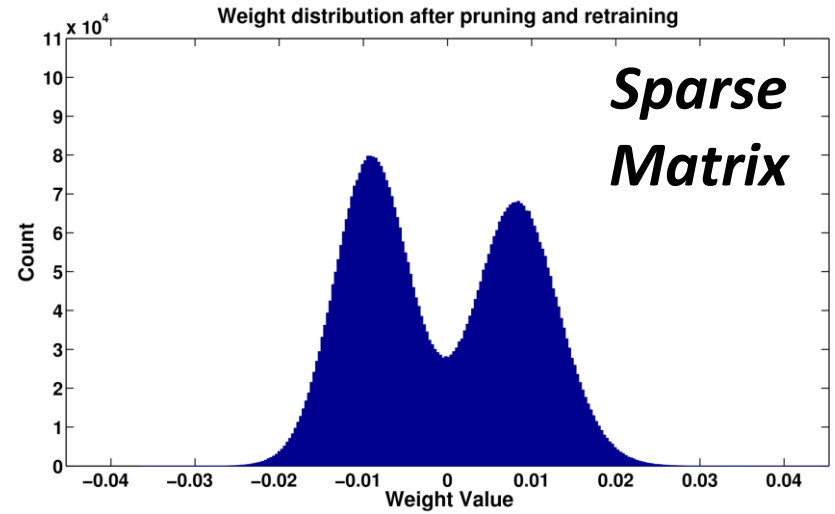

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"

volatile unsigned int * led              = (unsigned int *)0x40000000;
volatile unsigned int * switch_in          = (unsigned int *)0x40010000;

int main()
{
    init_platform();
    while(1)
    {
    * led = * switch_in ;
    }
    cleanup_platform();
    return 0;
}
```

Can we do better? -Deep Compression.

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**

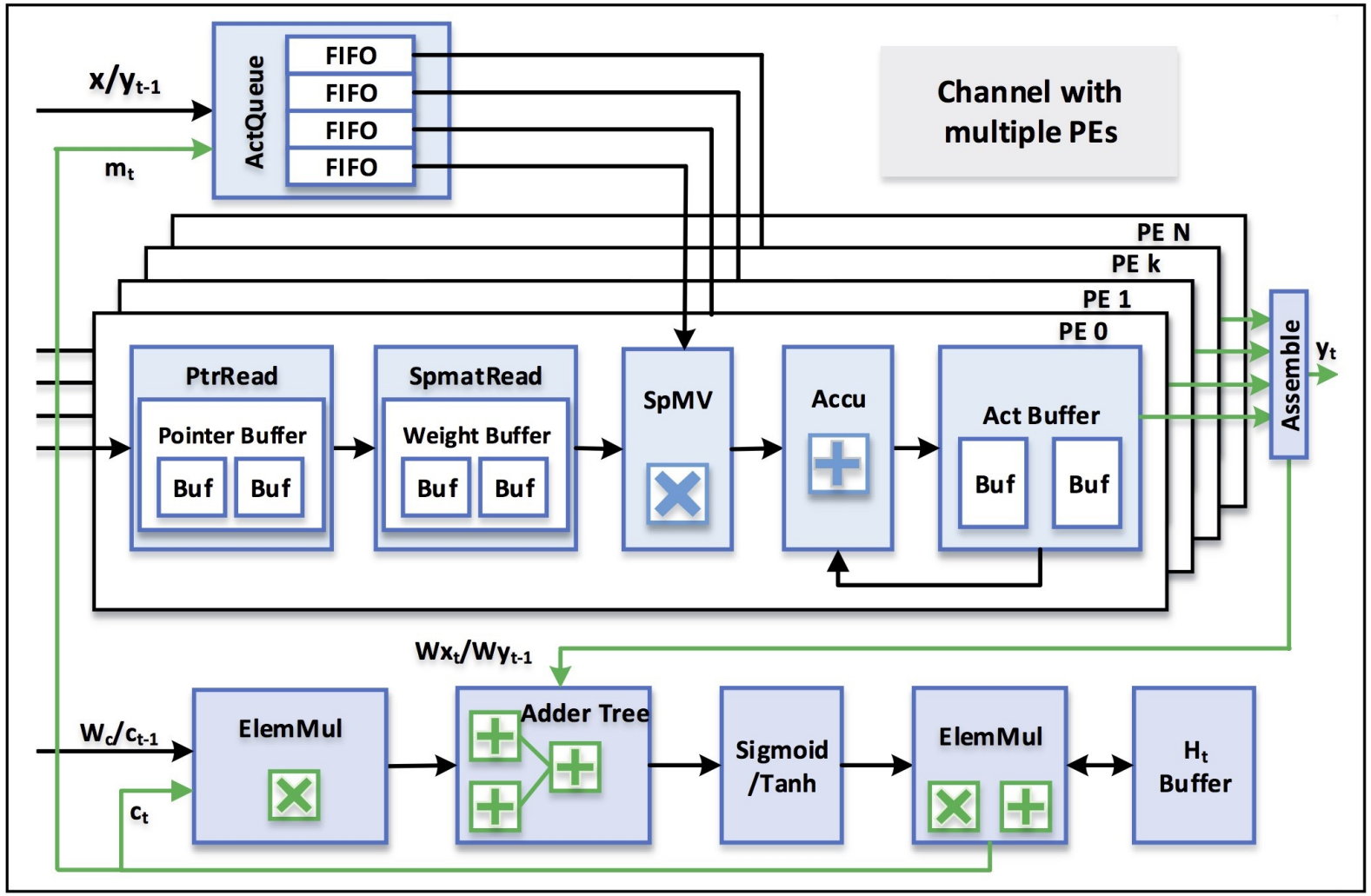






FROM: S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, W. Dally, "**EIE: Efficient Inference Engine on Compressed Deep Neural Network**", ISCA'16.

Can we do better? –Better Architecture.



From: S. Han, J. Kang, H. Mao, Y. Li, D. Xie, H. Luo, Y. Wang, H. Yang, W.J. Dally "**ESE: Efficient Speech Recognition Engine for Compressed LSTM**", to appear at FPGA'17

1. *Get something working.*
2. *Novel Architecture/Algorithm.*

Thank You!

Have a nice weekend!