ECE 464 / 564 Project : Fall 2020:
Multi-precision Artificial Neural Network

This project is to be conducted individually. You can collaborate on the paper version of the design, including discussion of ideas, design approach, etc. However, you are forbidden to share code or to reuse code of others. We will be running code comparison tools on your submitted code.

All 564-001 and 564-601 students are expected to undertake the 564 project. 464 students can take the 564 project if they want at their own risk. Both projects are simpler than in past years to account for the current circumstances. The normal distinction between EOL and on campus students no longer exists. Hence this change from past years.

ECE 564: Your task is to implement a BitFusion-like architecture that can handle multiple levels of precision in a neural network. The BitFusion concept is described in the attached paper (https://arxiv.org/pdf/1712.01507.pdf) and the attached independent study report.

The general idea is to change the arithmetic hardware to match the precision needed in the neural network stage. The hardware can perform a smaller number of larger multiply-adds per cycle or a larger number with smaller words. Doing this can save power and area versus the alternative of sizing the arithmetic units for the largest word sizes.

You are also to implement the interface to the test fixture provided. In the test fixtures, three memories are provided, one for weights, one for inputs and one for results. You can read one 16-bit word from the input and weight memories per clock cycle, and you can write one 16-bit word to the result memory per clock cycle. Assume that the memory cycle time can match or better your logic clock cycle time.

Note, I am NOT asking you to implement the BitFusion unit as described in the paper. It motivates and informs this project – it's not the specification. An ideal solution will incorporate enough arithmetic resources to keep pace with the memories – and no more. It is not required that you support multiple bit widths directly. **The simplest solution to this project is to implement one or more Multiply Accumulate (MAC) units that support the widest bit width and simply sign extend narrower inputs.** That solution is acceptable but it is not the most efficient solution.

ECE 464: Your task is to design a multiply-accumulate unit, or set of units, that can support the following specific combination.

- 2-bit weights, 8-bit inputs, 16-bit outputs. Up to 64 inputs.

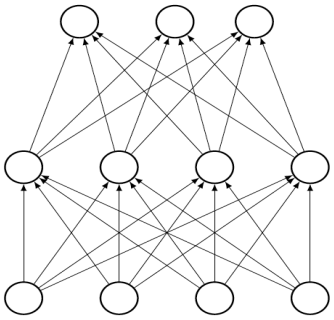
You are also to implement the interface to the test fixture provided. In the test fixtures, three memories are provided, one for weights, one for inputs and one for results. All three modes of operation discussed in the independent study report should be implemented.

In both projects you will be implementing one layer of a fully connected DNN to be computed at a time. The input memories will come with multiple single layer computations to be completed.

This is a technical description. Separately we will also be providing the following:

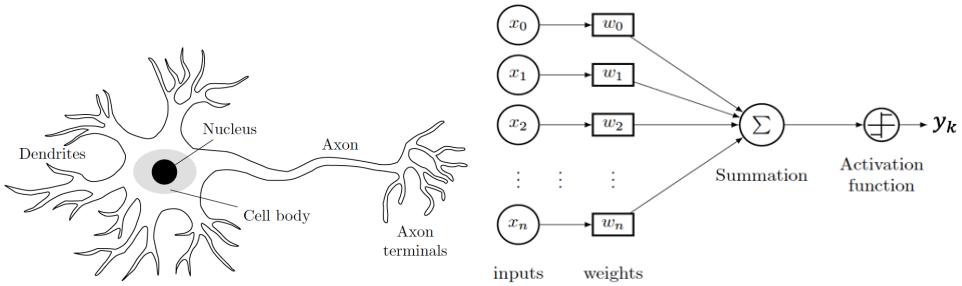
1. Sample inputs and expected outputs.
2. A test fixture precisely specifying all connections and how to feed the inputs and verify the outputs.
3. A general description of how the project will be conducted from a non-technical perspective.

A Brief Introduction to Artificial Neural Networks



Artificial neural networks are a set of algorithms inspired by the neural networks in biological system. Such algorithms are modeled base on a collection of connected perceptrons as in a biological brain. The network has to "learn" to perform tasks through a training process. During the training process, the associated weights of each connection can be adjusted to better match the desired result as training proceeds. The weight affects the strength of the connection between perceptron, as greater weight magnitude usually means the connection "conducts" more and will have more influence to the next stage.

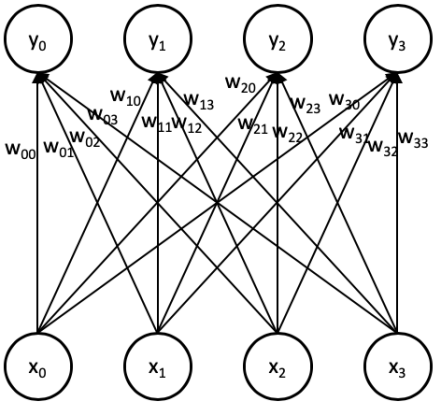
Perceptron Analogy



The artificial neuron is typically modeled as a perceptron as shown in the figure above. The output of the perceptron will be the dot products of the input vector and weight vector passing through an activation function as shown in the following equation, where $\emptyset(v_i)$ is the activation function.

$$y_k = \emptyset(w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n)$$

Weight Matrix and Connected Layer



$$\begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\begin{bmatrix} \emptyset(v_0) \\ \emptyset(v_1) \\ \emptyset(v_2) \\ \emptyset(v_3) \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

As we start to build up a layer of fully connected layer of the neural network, we can create a collection including all weight vectors and arrange (append and transpose) them into a matrix. With this weight matrix form, we can use matrix multiplication to write the computation as the evaluation above.

To simplify this project, we won't be implementing an activation function, i.e. [*y*]=[*v*].

Arithmetic Issues

These are all signed numbers and the outputs are signed too. (This is a simplification on the paper.)

Often in machine learning the output range is smaller than that specified by the problem and a saturating MAC is needed (i.e. instead of overflowing, the (here 8-bit) output stays at 8 or -7). However to simplify the problem this year we will have outputs that are sufficiently wide to capture the complete range. (This is a simplification on the paper and actual inference engines.)

For a discussion on implementation on signed arithmetic, please see https://moodle-courses2021.wolfware.ncsu.edu/mod/folder/view.php?id=272980 "Coding Guidelines". In short please use the signed variable declarations. Note, assigning to a wider signed variable automatically does sign extension.

e.g.

```
input  signed  [7:0] a, b;
output signed [15:0] z;

assign z = a * b;
// -> signed 8x8=16 bit multiply
```

Source: Synopsys

Overall Schema

The overall schema is a little artificial but is as follows: For each run

Control Interface

There will be three control signals for the unit.

input	reset_b	: Active low reset signal, will clear the machine state
input	clk	: System clock forwarded from the test fixture.
output	busy	: The test bench will halt when busy is high, waiting for the computation.
input	run	: The test bench will set run signal high after all data has been loaded.

ECE 564 Project – Memory structure

There are three SRAMs provided, one for weights, one for input data, and one for output results. Each memory is 16 bits wide, and is 16-bit word addressable.

Weights and inputs

These memories will be PACKED 16-bit memories, i.e. there will be multiple weights or inputs in a 16-bit word if the input or weight is 8-bits wide or less

- two for 8-bit inputs,
- four for 4-bit inputs or weights, and
- eight for 2-bit inputs or weights.

Outputs

All results are 16-bit words. Please use sign extension.

The maximum number of inputs (and outputs) is 128 decimal. All data sets are sized so that all words are fully used. E.g. 2-bit entries have 8, 16, 24, 32 etc. datums, 4-bits have 4, 8, 12, etc. datums, and 8-bits enties have 2, 4, 6, 8, etc. datums

> The first two lines of each block of memory will specify the number of inputs and the size of the input (2, 4 or 8 bits**). If the number of inputs is FF then there are no more samples to run in the memory.**

Below are some examples. Note these only describe an initial set of examples. Your hardware should be able to handle other examples within the parameters as follows:

- Weights : 2, 4, or 8 bits
- Inputs : 2, 4, or 8 bits
- Up to 128 inputs
- 16-bit outputs .

Examples:

Input set of the following sets. Each set is to be done and completed before moving to the next set.

- Problem 1: 16 inputs supporting 2-bit weights with 8 bit inputs (14-bit results, sign extended to 16 bits) (
- Problem 2: 16 inputs supporting 4-bit weights and 8-bit inputs (16-bit results)
- Problem 3: 32 inputs supporting 2-bit weights and 2-bit inputs (9 bit results, sign extended to 16 bits).

i.e. three runs total

All entries are in hex. (here "x" means an input, not that the number is in hex).

Data Input SRAM

Address	Data	Comment
0000	10	# of entries (decimal 16)
0001	08	Word size
0002	x1 x0	High order Byte is x1, Low order x0
0003	x3 x2	
…		
0009	x15 x14 // Problem 2	
000A	10	#entries, second set
000B	08	#word size
000C	x1 x0	
…		
0014	x15 x14	
// Problem 3		
0015	20	# entries (32 – four words of 8)
0016	02	Word size
0017	x7 x6 x5 x4 x3 x2 x1 x0	
…		
001A	x31 x30 x29 x28 x27 x26 x25 x23	
001B	FF	No more samples to run

Weight input SRAM

Address	Data	Comment
// Problem 1		
0000	10	# of inputs (16) # of weights are 16^2 = 256 weight – 32 memory locs

0001 02 word size
0002 w07 w06 w05 w04 w03 w02 w01 w00
0003 w0F w0E w0D w0C w0B w0A w09 w08
0004 w17 w16 w15 w14 w13 w12 w11 w10
…
0021 wFF wFE wFD wFC wFB wFA wF9 wF8
// Problem 2
0022 10 16 inputs, 256 weights – 64 locs
0023 04 word size
0024 w03 w02 w01 w00
..
0063 wFF wFE wFD wFC
// Problem 3
0064 20 # of inputs (32) # of weights is 32^2 = 1024 in 128 locs
0065 2 word size
0066 w00,07 w00,06 w00,05 w00,04 w00,03 w00,02 w00,01 w00,00
…
00E5 w1F,1F w1F,1E w1F,1D w1F,1C w1F,1B w1F,1A w1F,19 w1F,18

Output SRAM
Address Data Comment
// 16 16-bit results
0000 y0
0001 y1
…
000F yF
// 16 12-bit results, sign extended to 16-bits, one entry per word
0010 y0
…
001F yF
// 32 9-bit results, sign extended to 16-bits, one entry per word
0020 y0
…
003F y31

ECE 464 Project

Weights and inputs

These memories will be PACKED 16-bit memories, i.e. there will be multiple weights or inputs in a 16-bit word as the input is 8-bits wide and weight is 2-bits wide

- two for 8-bit inputs,
- eight for 2-bit inputs or weights.

Outputs

All results are 16-bit words. Please use sign extension.

The maximum number of inputs (and outputs) is 128 decimal. All data sets are sized so that all words are fully used. E.g. 2-bit entries have 8, 16, 24, 32 etc. datums, 4-bits have 4, 8, 12, etc. datums, and 8-bits enties have 2, 4, 6, 8, etc. datums

> The first line of each block of memory will specify the number of inputs**. If the number of inputs is FF then there are no more samples to run in the memory.**

Below are some examples. Note these only describe an initial set of examples. Your hardware should be able to handle other examples within the parameters as follows:

- Weights : 2 bits
- Inputs : 8 bits
- Up to 64 inputs
- 16-bit outputs .

Examples:

Input set of the following sets. Each set is to be done and completed before moving to the next set.

- Problem 1: 16 inputs supporting 2-bit weights with 8 bit inputs (14-bit results, sign extended to 16 bits)
- Problem 2: 32 inputs supporting 2-bit weights and 8-bit inputs (14-bit results, sign extended to 16 bits)
  i.e. two runs total

All entries are in hex. (here "x" means an input, not that the number is in hex).

Data Input SRAM

Address	Data	Comment
0000	10	# of entries (16)
0001	x1 x0	High order Byte is x1, Low order x0
0002	x3 x2	
…		
0008	x15 x14	
0009	20	# of entries (32) , second set
000A	x1 x0	
…		

0019	x31 x30	
001A	FF	No more samples to run

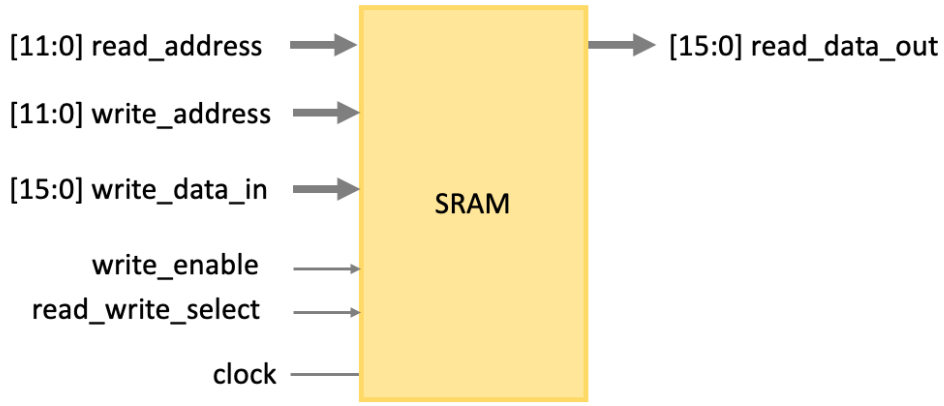
Weight input SRAM

Address	Data	Comment
0000	10	# of inputs (16) □ # of weights are 16^2 = 256
0001	w07 w06 w05 w04 w03 w02 w01 w00	
0002	w0F w0E w0D w0C w0B w0A w09 w08	
0003	w17 w16 w15 w14 w13 w12 w11 w10	
…		
0020	wFF wFE wFD wFC wFB wFA wF9 wF8	
0021	20	# of inputs (32) □ # of weights are 32^2 = 1024
0022	w07 w06 w05 w04 w03 w02 w01 w00	
..		
00A1	w3FF w3FE w3FD w3FC w3FB w3FA w3F9 w3F8	

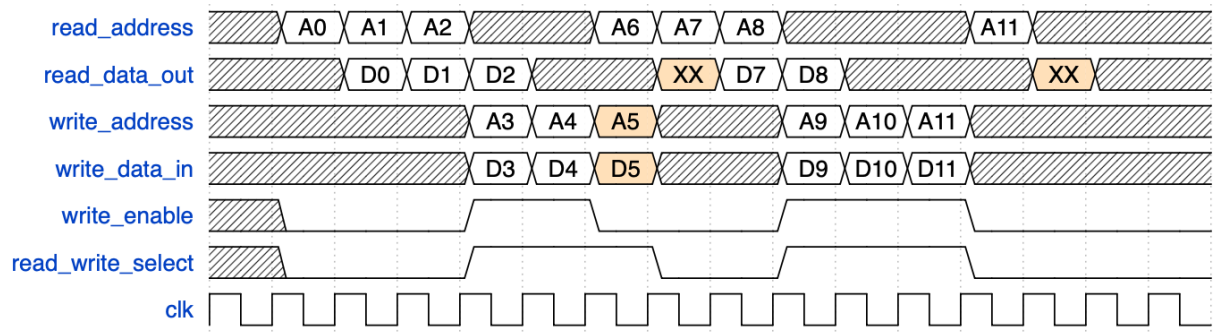
Output SRAM – one 16-bit word per SRAM word

// Problem 1 - 16 outputs	
0000	y0
..	
000F	yF
// Problem 2 - 32 outputs	
0010	y0
..	
002F	y1F

SRAM interface



The SRAM is byte addressable and has a one cycle delay between address and data. When writing to the SRAM, you would have to set the "write_enable" to high and the "read_write_select" signal to high. The SRAM will write the data in the next cycle. The "read_data_out" is valid when only "read_write_select" is set to low when requesting the data. Note that for cell state $C^{(t)}$, the read_data_out has 18 bits.



As shown in the example above, since "write_enable" is set to low when A5 and D5 is on the write bus, D5 will not be written to the SRAM. Also, because "read_write_select" is set to high, the read request for A6 will not be valid.
Note that the SRAM cannot handle consecutive read after write (RAW) to the same address (shown as A11 and D11 in the timing diagram). You would have to either manage the timing of your access, or write the data forwarding mechanism yourself. As long as the read and write address are different, the request can be pipelined.

Metrics
10 points (out of 100) are reserved for a competitive score in each class for performance per unit of area. Performance is measured as 1/(clock cycles * clock period). Area is just cell area.

IO Constraints
Apply the input/output timing constraints of the class standard synthesis script to all IO, including those connected to the SRAMs.