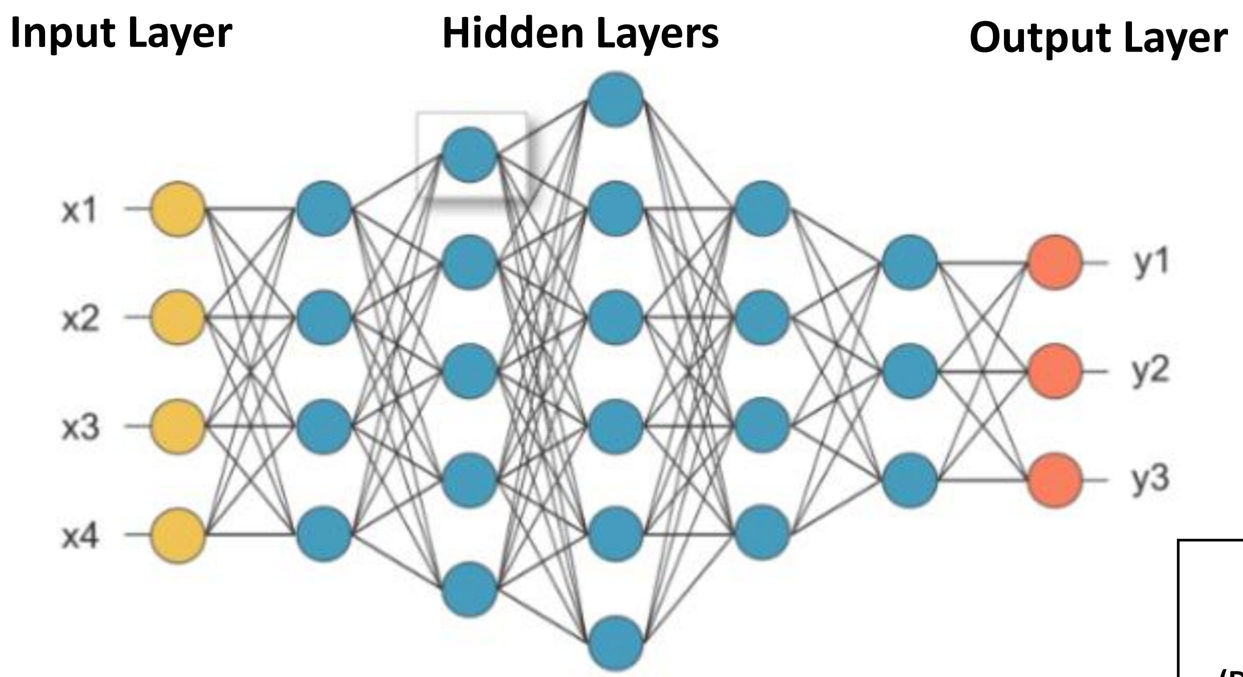
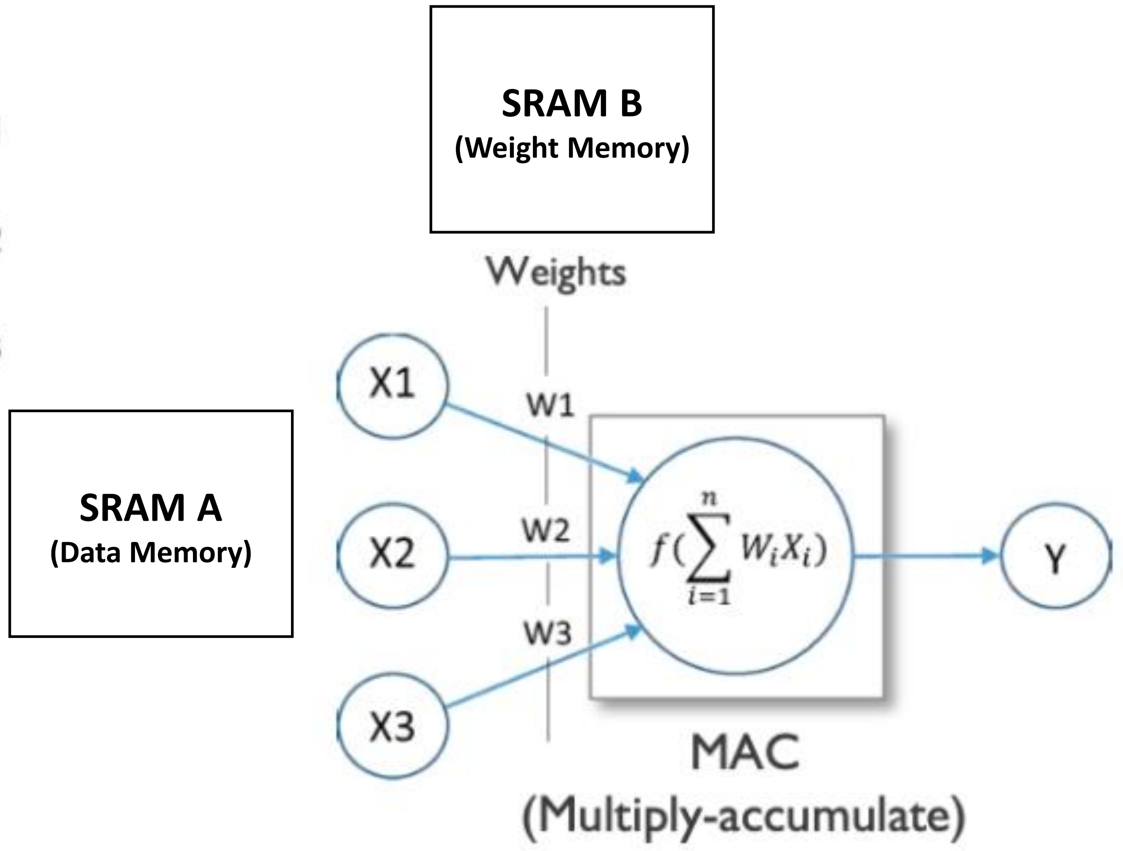
EE271 Project
Floating Point MAC for AI Engine

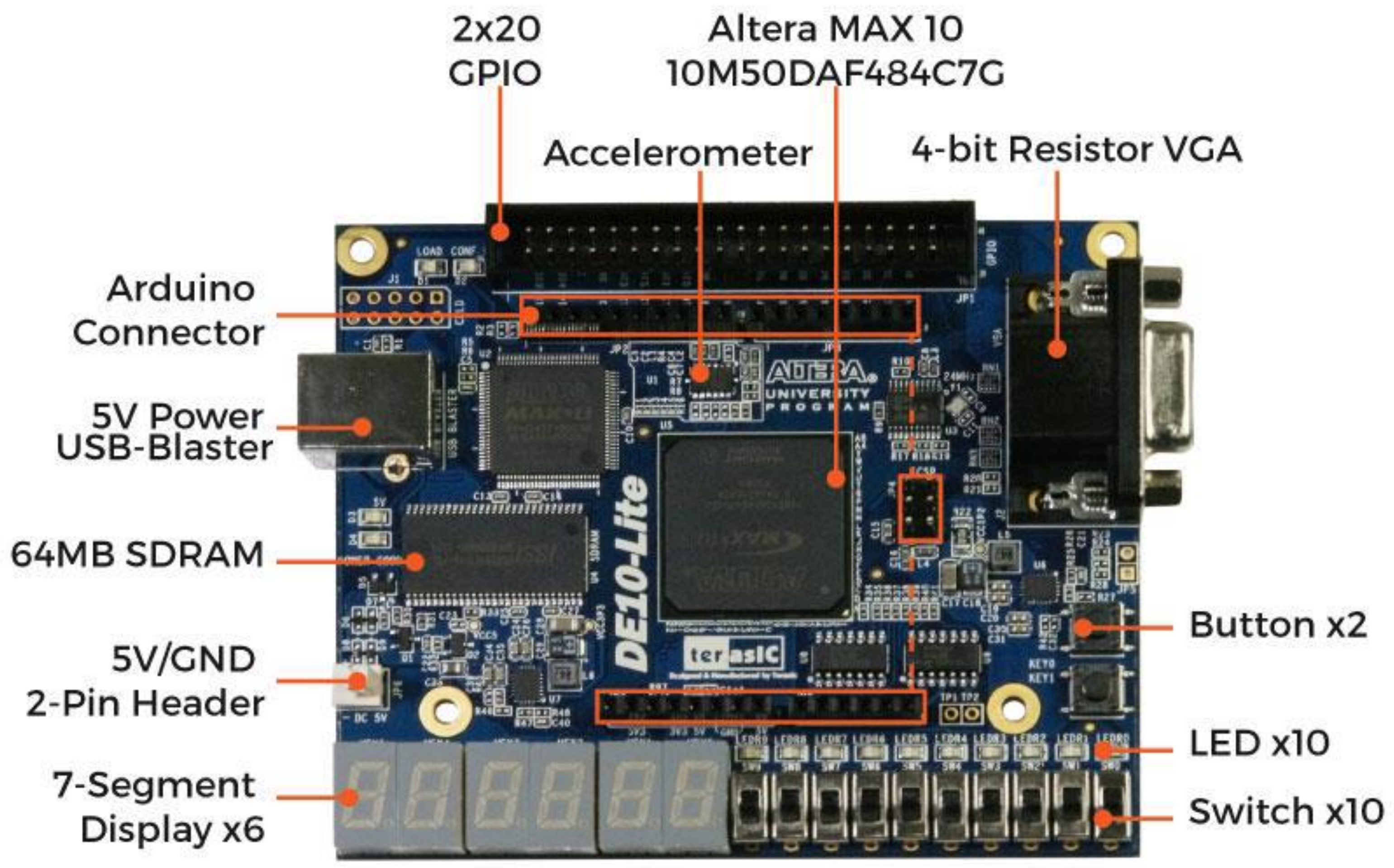
Design Specifications

MAC in AI Engines

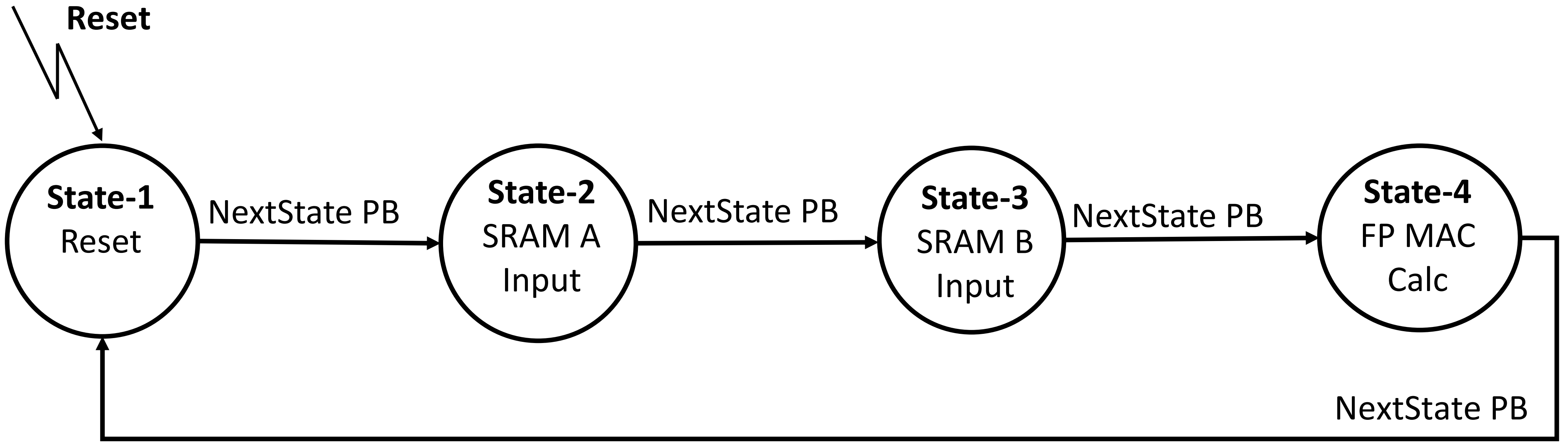




DE-10 Lite FPGA Board



Finite State Machine – FP MAC

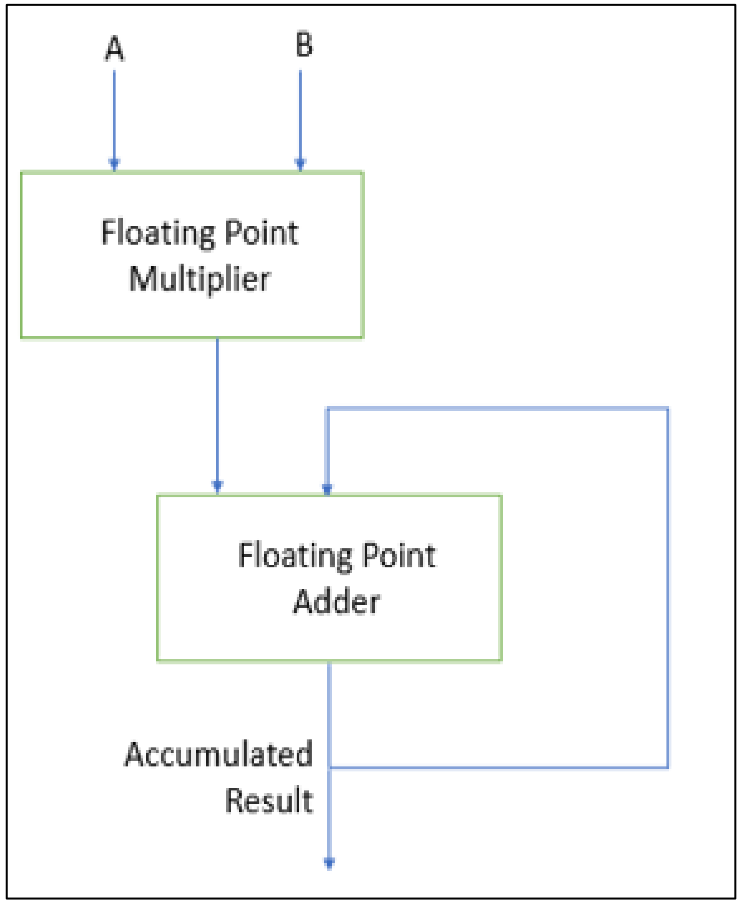


Working

- The Floating Point MAC should use 16-bit Half Precision FP numbers for the calculation.
- It should have two SRAMs (SRAM_A and SRAM_B) and it should allow user to store up to 8 FP values in both SRAMs using numerical keypad.
- In the final state, it should fetch each pair of FP numbers stored in SRAM A and SRAM B and multiply them using FP methods. Multiplied result should then be accumulated using FP Adder.

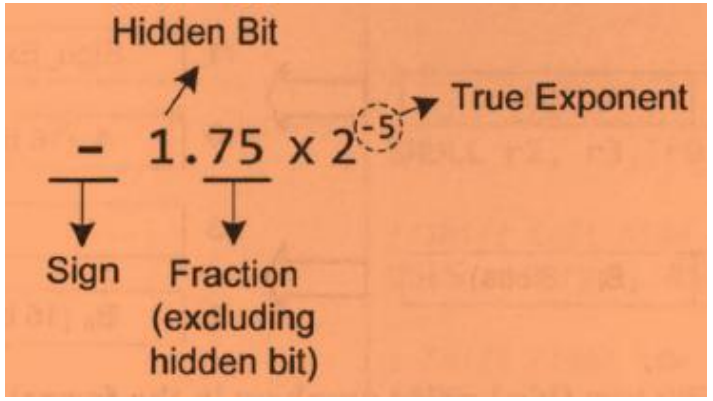
$$Result = \sum_{i=0}^{7} (A_i \times B_i)$$

- It should display the current state in LCD display controlled by Arduino.
- It should display the final result in hex in onboard 7-segment displays.
- FP MAC operation should be pipelined for better throughput.



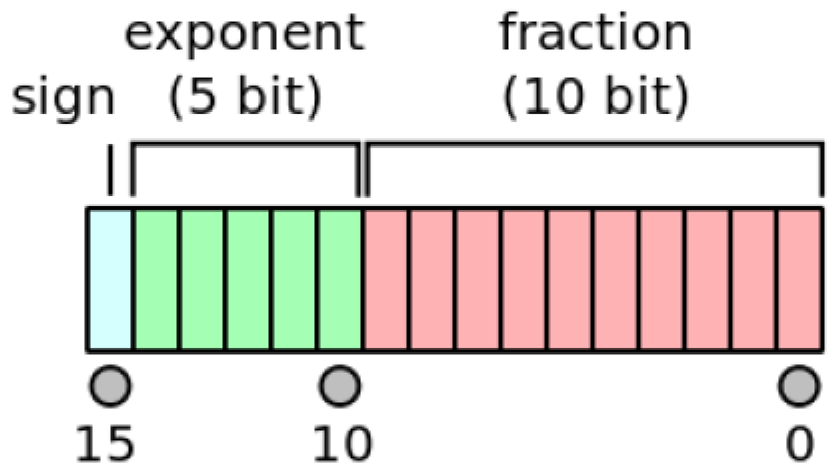
Floating Point Representation

- To understand Floating Point operations, first we need to know what are floating point numbers. IEEE represented a way to store larger set of numbers in fewer bits by creating a standard known as IEEE 754.
- You can represent numbers using 16,32,64 bits or custom bits following the standard. We will use 16 bits (Half Precision) for simplification.



16-bit Half Precision

- Half Precision uses 16 bits to represent numbers.
- It has 3 fields Sign, Exponent and Fraction.
- Mantissa is the fractional part, and exponent gives us the power.



Example: Representation of 5.5 in 16-bit FP

- Calculating Sign : Sign =0 (positive)
- Calculating Exponent and Fraction: divide or multiply number by 2 to achieve 1.XX format

  5.5 / 2 = 2.75

  2.75 / 2 = 1.375 (Note that we have reached 1.XX format, so we stop)

  Therefore,

  $5.5 \div 2^2 = 1.375$ or, $5.5 = 1.375 \times 2^2$

  Here we see **true exponent is 2** and **fraction is 0.375**

Calculating Biased Exponent

- Exponent was calculated as +2.
- To represent signed values, we could have either used two’s complement or biasing. Biased was selected because two’s complement makes it harder to compare.
- **Biased exponent= true exponent + Bias**
- Bias for Half Precision (16-bits) is **15**
- This means that the 5 exponent bits will now store biased values.
- So, our exponent 2 will be added to bias.
- It will now be added to 15 and stored as 17

Calculating Mantissa (Fraction)

- To convert Fraction to Mantissa, we need to convert 0.375 to binary.
- The conversion is achieved by repeatedly multiplying the fraction part with 2 until the product becomes 1. We need to store the leading digits of each multiplication as it becomes our binary value

$0.375 * 2 = 0.75 = 0 + 0.75$

$0.75 * 2 = 1.5 = 1 + 0.5$

$0.5 * 2 = 1.0 = 1 + 0$

- Therefore, our binary representation of 0.375 is 0.011

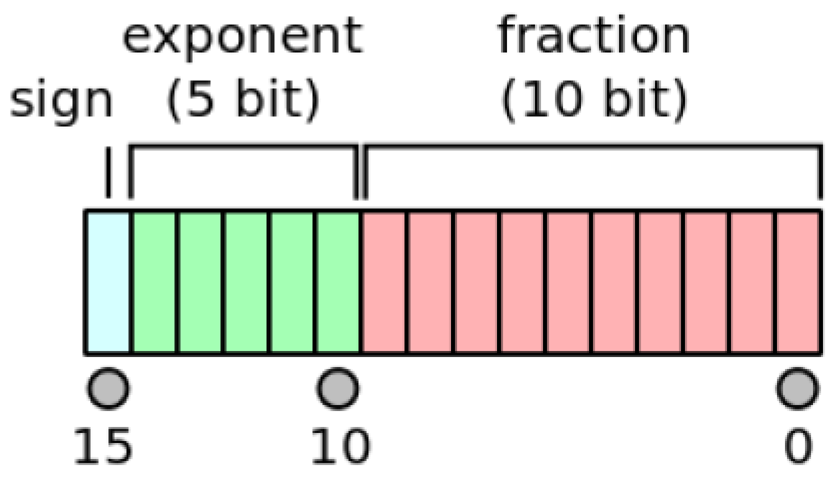
Converting to hexadecimal

- Finally, we can convert the number to hexadecimal for storage.
- 16-bits is 4 hexadecimal numbers = 0xHHHH in hexadecimal
- Sign is **0** which is 0 in binary
- Biased Exponent is **17** which is **10001** in 5-bit binary
- Mantissa is **011** and it is written from right to left (MSB to LSB)
- So, the 10 bits of mantissa will look like **0110_0000_00**

- In hexadecimal, the number would be

  0100_0101_1000_0000 or

  0x4580



What happens in each state?

- **State - 1 :**

  Reset SRAMs to Zero. Wait for the next state button. No need to display anything on 7-segment displays. Display “Reset” on LCD.

- **State - 2 :**

  Display “SRAM A Input” on LCD.

  Input 4 hex values (16-bit FP number) from 4x4 Keypad, display them on 7-segment display and store it to the corresponding mem location of SRAM_A. In this state, user should be able to store FP numbers to all 8 mem locations.

- **State - 3 :**

  Display “SRAM B Input” on LCD.

  Input 4 hex values (16-bit FP number) from 4x4 Keypad, display them on 7-segment display and store it to the corresponding mem location of SRAM_B. In this state, user should be able to store FP numbers to all 8 mem locations.

- **State - 4 :**

  Display “FP MAC Calc” on LCD.

  In each clock cycle, it should fetch one-one FP numbers from both SRAMs (starting from mem location 0), multiply them and accumulate the results of each pair. Display the final accumulated result in hex on 7-segment.

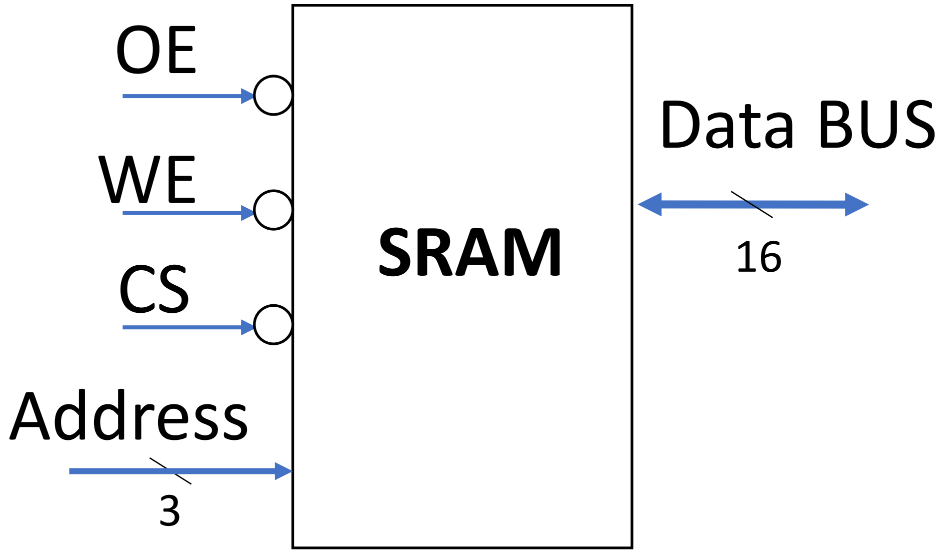
SRAM

Size : 8 x 16-bit (half word)

Control Pins :

**OE : **Output Enable (0- enable, 1- disable)
WE : Write Enable (0- enable, 1- disable)
CS : Chip Select (0- enable, 1- disable)

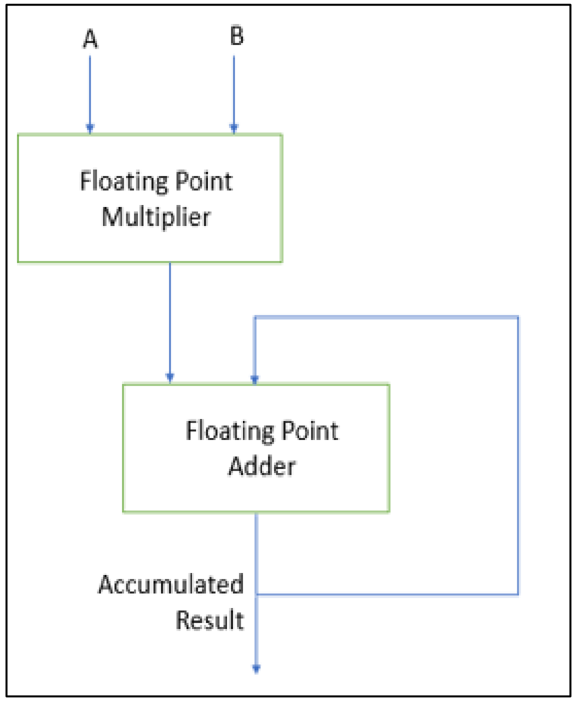
Data : 16-bit bi-directional data bus
Address : 3-bit input



Asynchronous read and write operation should be supported.

Pipelining (for FP MAC Calculation)

- Pipelining is the process of dividing the algorithm into several stages and executing all stages parallelly.
- This project implements pipelining in Floating Point Multiplier and Adder by dividing the calculation into Three stages.
- Floating Point Multiplier has two task – Multiply and Normalize .
- Floating Point Addition has three tasks - Align, Add and Normalize.

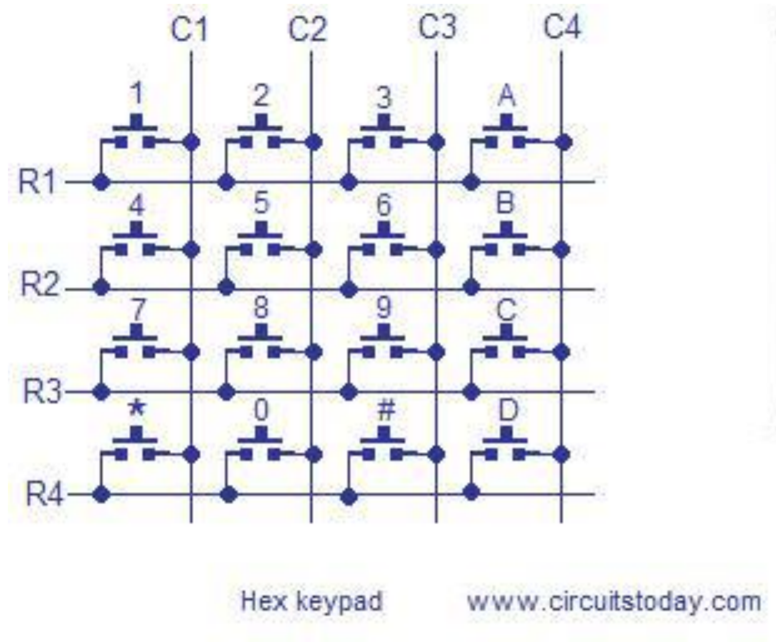


PP_Stage-1 (Multiply) → PP_Stage-2 (Normalize) → PP_stage-3 FP_Addition

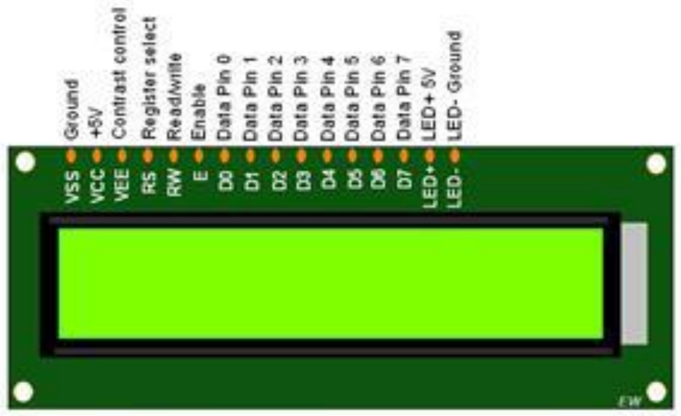
FP_Multiplication

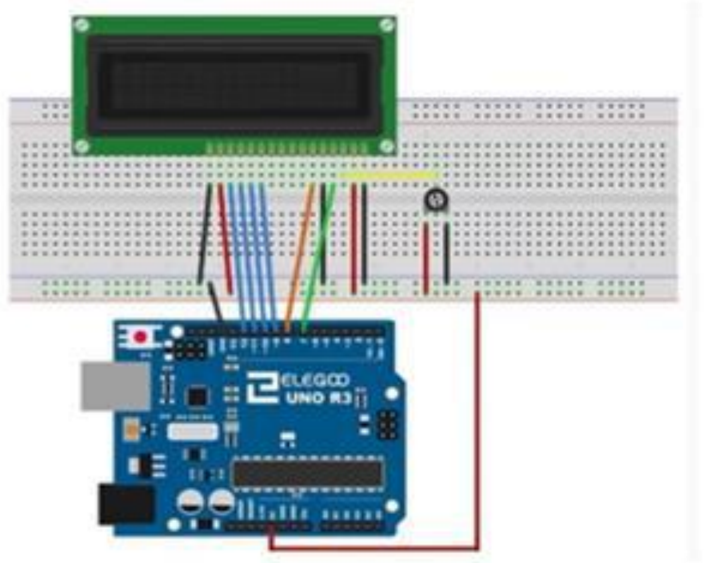
Interfacing with 4x4 Numerical Keypad

- For the Numerical Keypad, we are given four row wires and four column wires. This gives us a matrix of 4x4 points.
- We can use this to detect any button pressed on the Keypad.

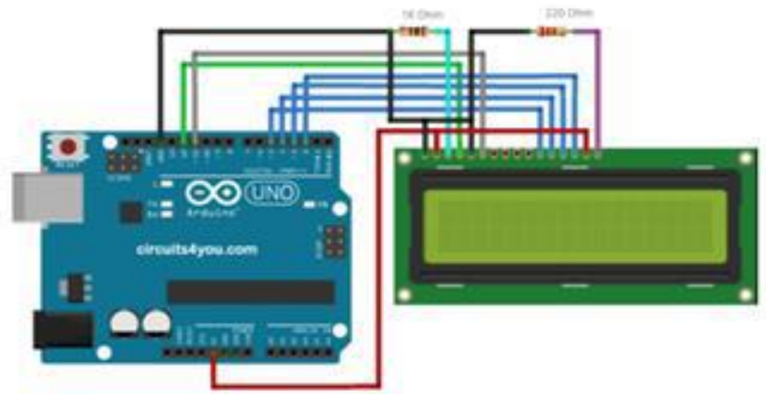


Interfacing with Arduino and LCD





E.g. 1



E.g. 2

Interfacing with Arduino and LCD

- We will also add an LCD interfaced with Arduino to tell us the current state. The Arduino code simply displays the name of the project at the top row and the current state (Reset, SRAM A Input, SRAM B Input or FP MAC Calc) state at the bottom row.
- The LCD configuration can be done using the circuit given by this link: lCD_Setup
- There are two videos for setup of Arduino Blink_LED and for configuring the LCD LCD_Display

FAQ

1. How to travel between the four states?

   You should use an onboard button to go to the next state.

2. How to reset?

   Use a switch for reset and reset everything to 0 from any state.

3. What is the size of the SRAMs?

   Your SRAMs can store up to 8 16-bit values.

Bill of Materials (per group)

S. No.	Component	Description	Quantity	Cost	Link
1	DE-10 Lite FPGA Board	FPGA Board for programming	1	$64	Link
2	4x4 Numerical Keypad	Keypad for input	1	4 for $7	Link
3	LCD	16*2 is recommended	1	$6	Link
4	Arduino Uno	(You can buy Open Source as well)	1	$14 ~$23	Link
5	Wires	buy M2M, M2F, and F2F for safety	~20	$7	
6	Breadboard	Any Breadboard will do	1	$8	Link
			Total cost per group:	**~ $105**	