CS223 Laboratory Project
Two's Complement Checksum for Memory

Groups: Each student will do the project individually. **Group size = 1**

Important dates:
Report submission and project upload: 18.5.2020 Monday, until **09:00 AM**.
You need to upload your design files in order to make your project demonsration.
Demo presentation day: 18, 19, 20 and 21 May 2020 during lab time.

Description:

The purpose of this project is to build a data input system and memory integrated with Checksum algorithm by using an FPGA on Basys 3 board. The project is an example of creating a system to perform sophisticated computations by combining different components on the FPGA.

Data corruption might occur whenever digital data is stored or transmitted. Checksum calculations are used in different types of communication systems and packets sent over a network are verified with a checksum. Checksum is simple, powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

In this project, the idea behind checksum algorithm is to consider whole data as one large array. Each array item is repeatedly added starting from the first index until the last one is reached to find out the overall sum. Afterwards two's complement of this total sum is calculated and the resulting value will be the checksum (details of the algorithm are given in Appendix and in Figure 6).

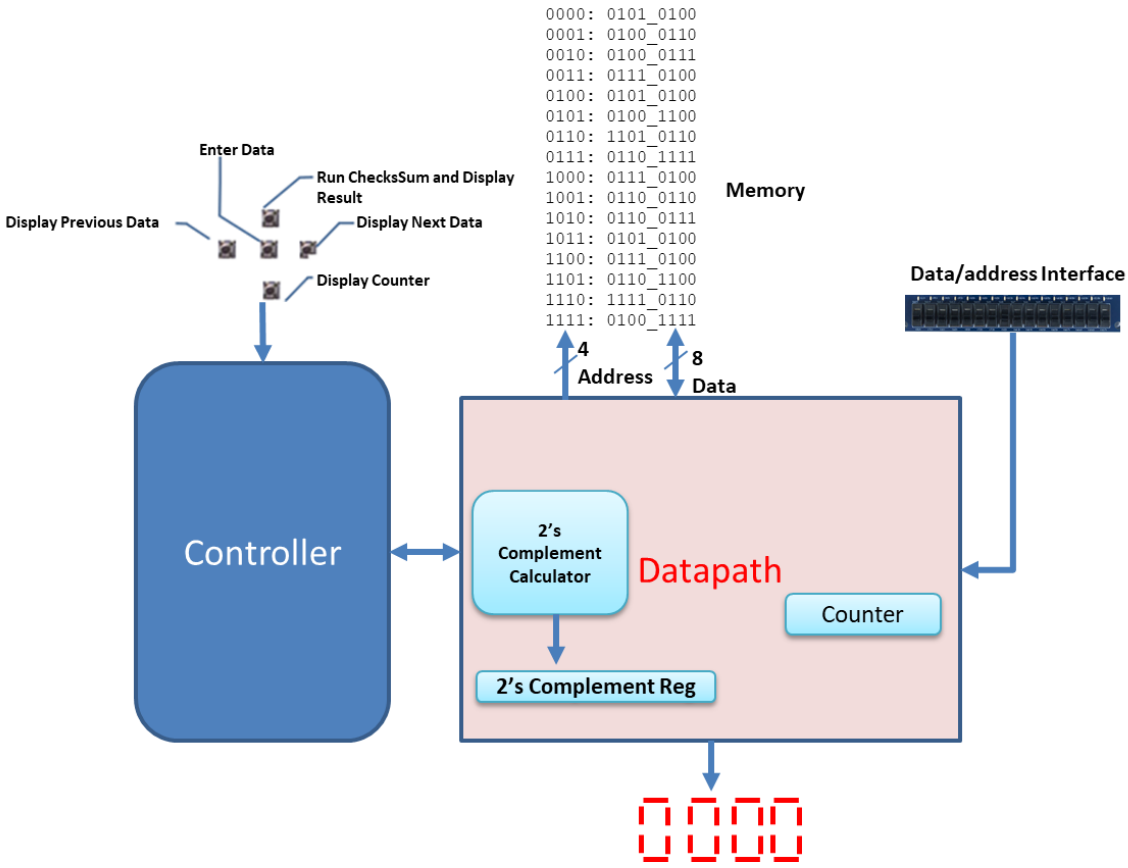

Figure 1: Operation Flow Architecture

A memory block will be filled with 16 data words with 8-bit width and then several functions will be executed through push-buttons. Data and address bits for memory interface will be taken from the user switches (Figure 1 and Figure 2).

Data Input interface: 12 user switches and their corresponding 12 user LEDs will be used in order to input 8-bit data and 4-bit address for each memory word. As shown in Figure 2, eight user switches will be used in order to input the unsigned 8-bit data for the memory, last four user switches will be used in order to specify the target memory address. Memory size in this project is 16 words with 8-bit width. Once data and address inputs are set you should press on the Enter Data button in order to store the adjusted data value to the addressed memory location.

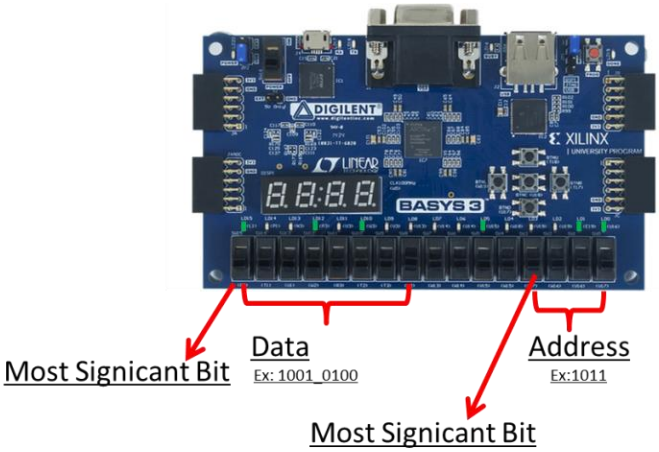

Figure 2: User Switches and LEDs for Data and Address Inputs

1. **Enter Data**

   The status of switches SW15-SW8 are registered as the data input and the status of switches SW3-SW0 are used as the address information. If a LED is ON then the corresponding bit is 1, on the other hand if the LED if OFF then the corresponding bit defined by the switch is 0.

2. **Run Checksum and Display Result**

   Once the memory is filled totally, CheckSum will be calculated by using two's complement checksum algorithm for the whole memory content. By pressing on Run CheckSum push button, checksum value will be calculated and displayed on the four digit 7-segment display for 10 seconds (Figure 4) and then the display will return back to memory display mode (Figure 5).

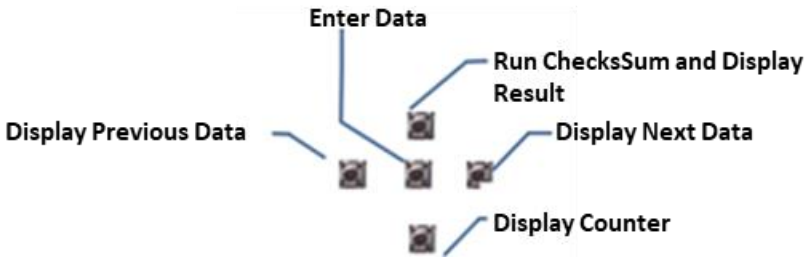

Figure 3: Push Button Operations

In memory display mode, 7-segment display will show the content of the memory location in the address specified by slide switches SW3-SW0. Address information will be displayed on the most significant digit of the seven-segment display and data information will be displayed on the least significant two digits of the seven-segment display.

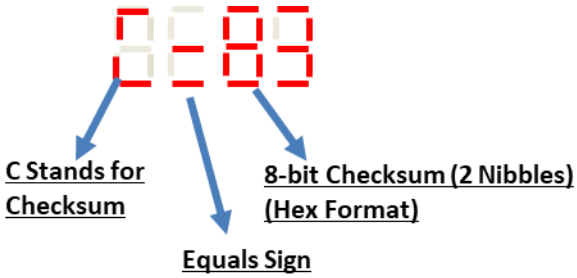

Figure 4: ***Checksum Display Mode*** *(Ex: Checksum C=0x83)*

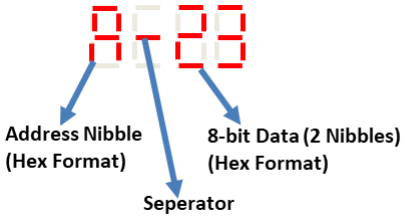

Figure 5: ***Memory Display Mode*** *Address-Data Values on Seven Segment (Address: 0xa – Data: 0x23)*

3. Display Previous Data

The previous memory address and corresponding data will be displayed on the seven segment upon pressing on this button. If the address displayed is 0x0 then with a press on display Previous Data button 0xf address (will circulate) and data will be shown on the display (Figure 5).

4. Display Next Data

Next memory address and corresponding data will be displayed on the seven segment upon pressing on this button. If the address displayed is 0xf then with a press on display Next Data button 0x0 address (will circulate) and data will be shown on the display (Figure 5).

5. Display Counter

Upon pressing on the **Run CheckSum** button a counter will be initialized and the counter value will be incremented by one on each clock cycle until the CheckSum calculation ends. Thus, the total number of clock cycles will be counted during CheckSum operation. With a press on **Display Counter** button the counter value will be shown on the display for 10 seconds and then the display will return back to memory display mode (Figure 5).

6. Reset State

On reset, memory will be initialized with the following content 0x00, 0x01, 0x02, 0x03, 0x04 0x05, 0x06, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f and the seven-segment display will show the memory content according to the address adjusted by the slide switches SW3-SW0.

Notes:

1. Study Basys 3 board documents carefully to avoid connecting different wires to each other and damaging your Basys 3.
2. You need to have a report for your project and show it to your TAs during demo. Report at least should include:
    - Give explanation about which physical modules you used on Basys 3, such as push buttons or switches etc.

- Describe your own design, Systemverilog modules and codes, timer, FSMs, datapath (*do not include your code in the report*).
- Give detailed diagram of the main FSM, datapath in your design, and clear explanation for the function of each state.
- If you use any ready code from internet, mention it at the end of your report.

APPENDIX:

The checksum value contains the two's complement of the sum of all words in the memory. In this project, memory has 16 x 8-bit data and sum of these 16 words will be stored in an 8-bit register so that **overflow** during addition operation **will be ignored**. Once the addition operation including 16 words is finished, Checksum value will be calculated by adding 1 to the complement of calculated sum register.

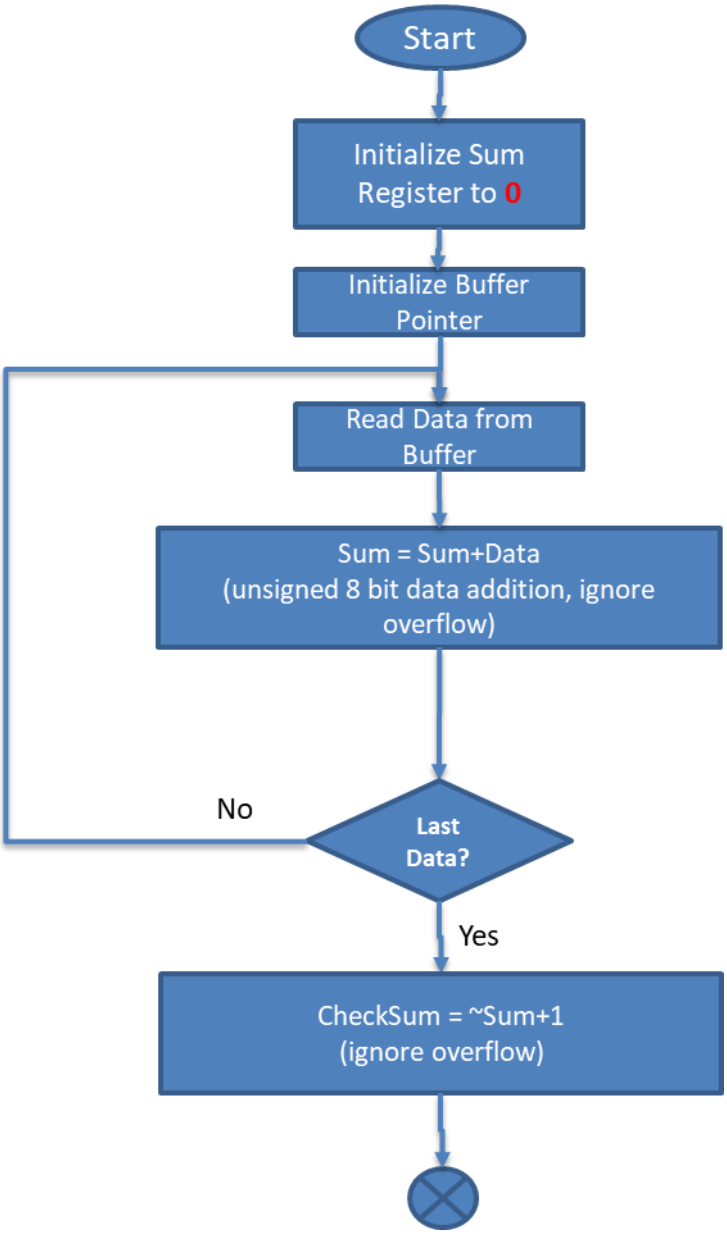

Figure 6: Two's Complement Checksum Algorithm

```
#include <stdio.h>

unsigned  char twoscomplement(unsigned char *pMsg, unsigned char size)
{
     unsigned char i,j;
     unsigned char checksum,sum=0;
     unsigned char temp;
     for(j=0;j<size;j++){
          temp = *(pMsg+j);
          sum = sum+temp;
     }
     checksum = ~sum+1;
     return (checksum);
}

unsigned  char twoscomplement(unsigned char *pMsg, unsigned char size);

int main()
{
     unsigned char csprjRam[16]={0X00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06,
     0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e ,0x0f};
     unsigned char csum;
     csum = twoscomplement((unsigned char *)&csprjRam[0], 16);
     printf("2's complement = %2X ", csum);
     return 0;
}
```

This algorithm will generate a checksum **= 88** with the memory content (`csprjRam[16]`) given in the example code above.

For example, the algorithm will calculate a checksum **= 0C** for the memory content including 16 words given in Figure 1.