TE3059 Embedded Systems

Project Title:

Calculator with PS/2 keyboard and LCD display

Final Project Report

Andrés Antonio Bravo Orozco A01630783
Rodolfo Casillas Sánchez A01229493
Jean Carlo Álvarez Guerra A01635182

3/12/2021

Revision History

Rev. #	Date	Changes By	Description
0.1	18/11/21	Andrés	Preliminary
0.2	22/11/21	Rodolfo	Introduction, scope, and intended audience redacted
0.3	27/11/21	Jean Carlo	Completion of theoretical frame work and project description
0.4	29/11/21	Jean Carlo	Update on design description
0.5	30/11/21	Andrés	System requirements and test results finalized
1.0	1/12/21	Rodolfo	Design description redacted

Table of Contents

Table of Figures

1.0 Introduction

This document describes the final project that we propose to implement as part of the final evaluation of the Embedded Systems Laboratory course. The main goal of this document is to describe the main details of the calculator that we will implement, the general objectives of the project (which include communicating the FPGA and de ARM processor), the approach that we will follow to achieve these objectives, the hardware and materials required to develop the project in a successful manner and the acceptance criteria that will be used to ensure a correct performance calculator.

1.1 Scope

This document has the purpose of providing a high-level view of the final project for the Embedded Systems Laboratory course. No technical specifications will be discussed, rather, an overview of what the final product will be is going to be described, things such as the key elements, design flow and necessary information to understand the project will be present in this document.

1.2 Intended Audience

This document is intended to be read by the professor of the course and classmates that may need it as reference for future projects, either personal or academic.

1.3 Terminology

Acronym	Description
ARM	Advanced RISC Machine
CLK	Clock
FPGA	Field Programmable Gate Array
GPIO	General Purpose I/O
HPS	Hard Processor System
I/O	Input/Output
LCD	Liquid-Crystal Display
LSB	Least Significant Bit
PS/2	Personal System/2
RISC	Reduced Instruction Set Computing
SoC	System on Chip
SPI	Serial Peripheral Interface

Table 1 Acronyms

1.4 Related Documents

Terasic. “DE10-Standard User Manual”, V1.0.6, 2018.

2.0 Project Description

The final project for the Embedded Systems Laboratory course will consist of a calculator that is divided into two parts to meet the criteria of involving the FPGA and the processor of the DE10-Standard Cyclone V development board. The first part, the FPGA part, will consist of designing a controller for the PS/2 interface so a keyboard can be connected to it and be used as an input device. The second part requires the interface of the LCD display that is available on the development board, the LCD will be configured using C code and driven by the ARM processor that is embedded in the board, also, the logic for the mathematical operations regarding the calculator will be configured using high level programming language.

Since we took our digital systems laboratory we wanted to implement a controller for a PS/2 interface, but for some reasons we were not able to fit it into the course time, and now we see a really good opportunity to finally implement it alongside the LCD in the development board. We believe that this a very complete project in terms of the knowledge it will give us, embedded systems design is an area with a huge professional potential, and being able to understand and implement our own project, it is a good way to make the most out of the course and the boards we have there in order for use to use as best as we can.

A diagram depicting the HPS and FPGA division on the DE10-Standard board is shown on figure 1 and figure 2. This figures help understand how the development board is distributed and can be confirmed that the PS/2 interface and the LCD display are on separate sections of the board, therefore communication between them should be established in order to fulfil both, the most important requirement of the project, and also the proper functionality of the calculator.

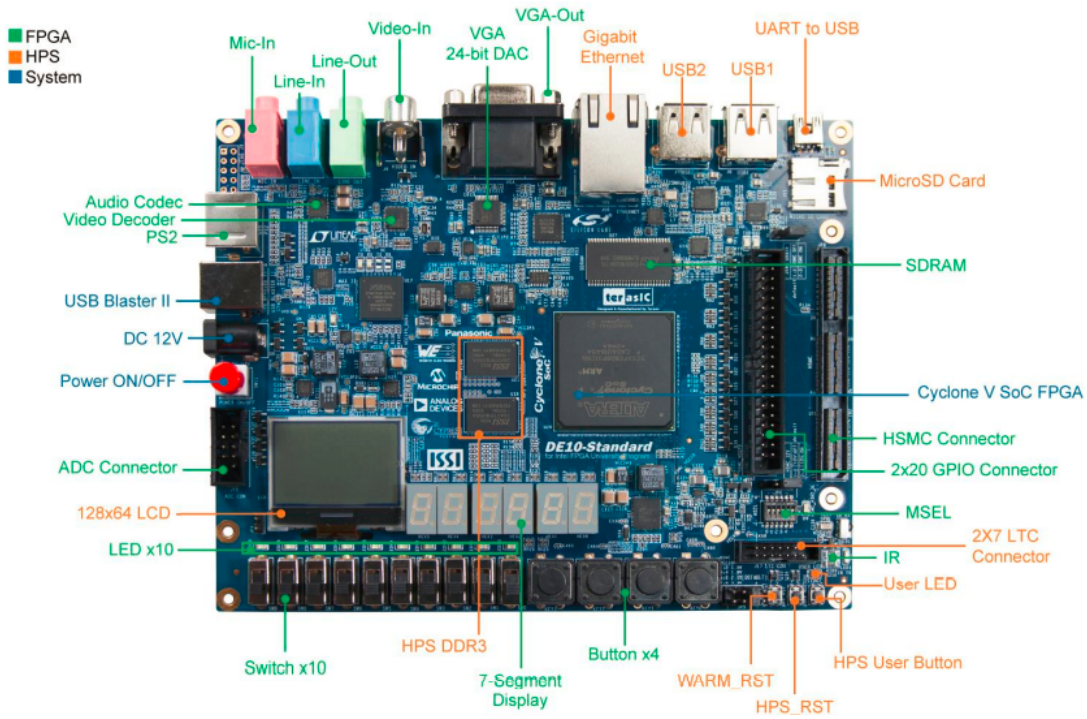

Figure 1: HPS and FPGA structure of the DE10-Standard Cyclone V board

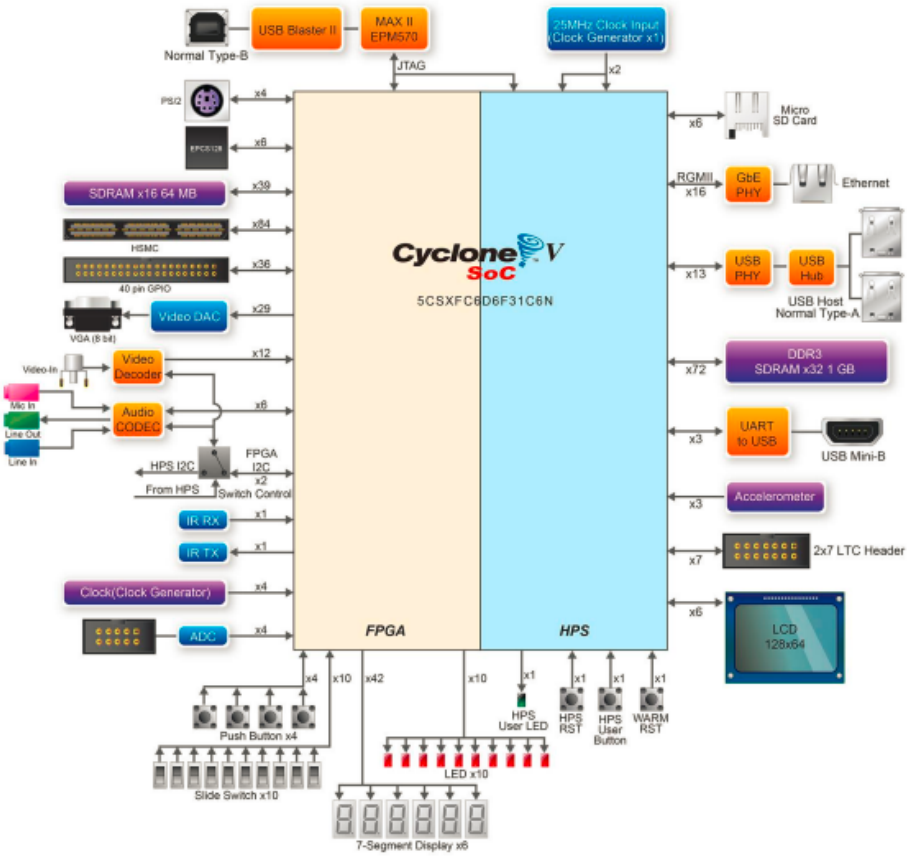

Figure 2: DE10-Standard Cyclone V board peripheral distribution

2.1 System Block diagram

Figure 7 contains a block diagram portraying the system components and their desired interconnections.

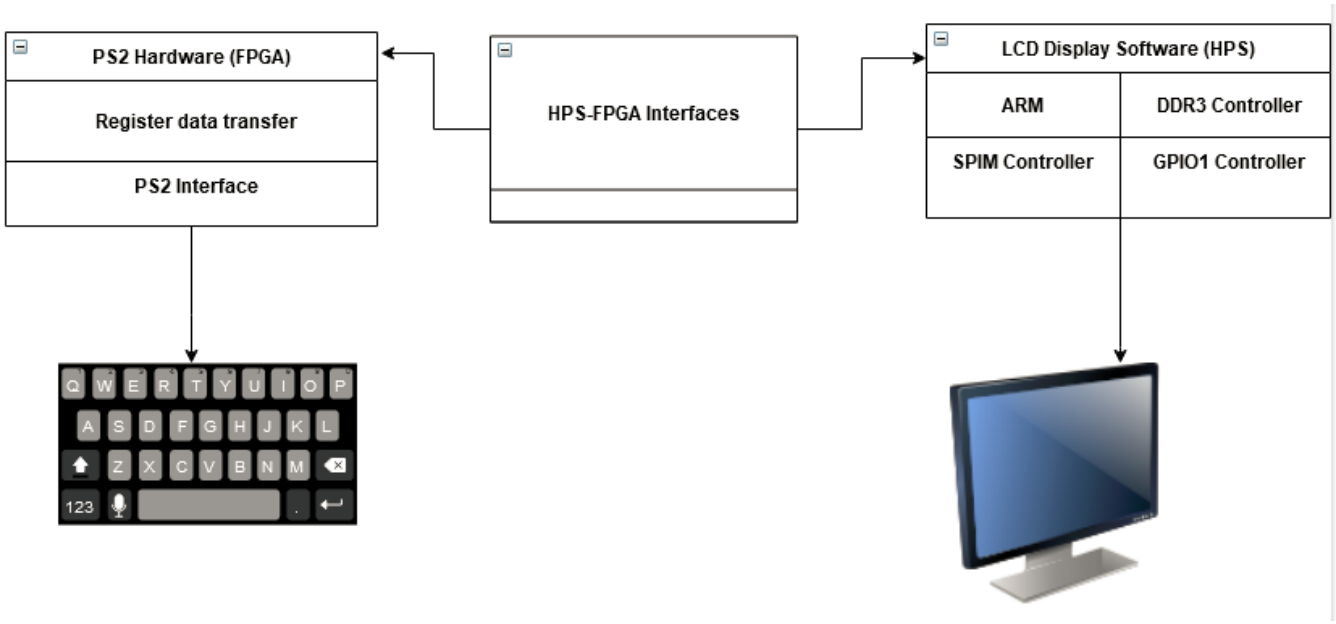

Figure 3: System block diagram

2.2 Project Goals

The main goals for this project are defined below

- Understand and properly implement the interface between the FPGA and HPS elements of the SoC.
- Comprehend the functionality of the PS/2 keyboard and its interface with an FPGA
- Understand the LCD connections to the ARM processor and their interactions.
- Being able to process operations with more than one digit
- Understand the low level interaction of a processor with hardware.

3.0 Theoretical Background

PS/2 Keyboard

For the FPGA section of the project, the PS/2 serial port of the DE10-Standard Cyclone V development board will be used to connect a keyboard and use it as an input device.

The interface allows two PS/2 devices to be connected at a time with a corresponding coupler, for this project only a keyboard will be connected, so the signals that are going to be used will be *PS2_CLK* and *PS2_DATA*. The corresponding connections can be seen on figure 3.

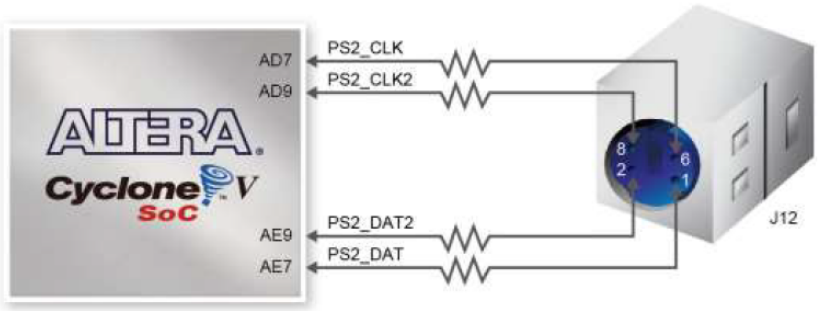

Figure 4: Connections between the FPGA and PS/2 interface

The basics of PS/2 communication consists of two signals, clock and data (as stated before), the controller has absolute control over the transmission signal, and the PS/2 device is the one that generates the clock signal for data transmission. There is bidirectional communication between the controller and the device. If the device is to send data to the controller, if should receive first the enable signal in stream mode, after that, data is going to start to be transmitted, said data consists of words of 11 bits, each frame consists of three sections, they contain a start bit that is always low, eight bits corresponding to data starting with the LSB, one parity bit using odd check, and one stop bit that is always high. This way the controller is able to sample the incoming data on the falling edge of the clock signal, implementing a 11-bit shift register [1].

The other type of communication is from the controller to the device, in order to indicate the start of a new transmission, the controller pulls the clock line to a low state for more than one clock cycle, once this is done, the data line is also pulled down so now the clock line can be released, this makes the system go from inhibit state to release state. The transmission of data consists of words that are 12-bits long, it contains a start bit that is always low, eight bits containing the data, one parity bit with odd checking, a stop bit that is always high and an acknowledged bit set to always be low. Now the data line is released and the device can detect any change that may appear on that said line, if no change is detected in one clock cycle, the device will pull low the data line and that action will be interpreted as an acknowledge that the data was received in a correct manner [1]. Figure 4 shows wave diagramas for the communication between controller and device.

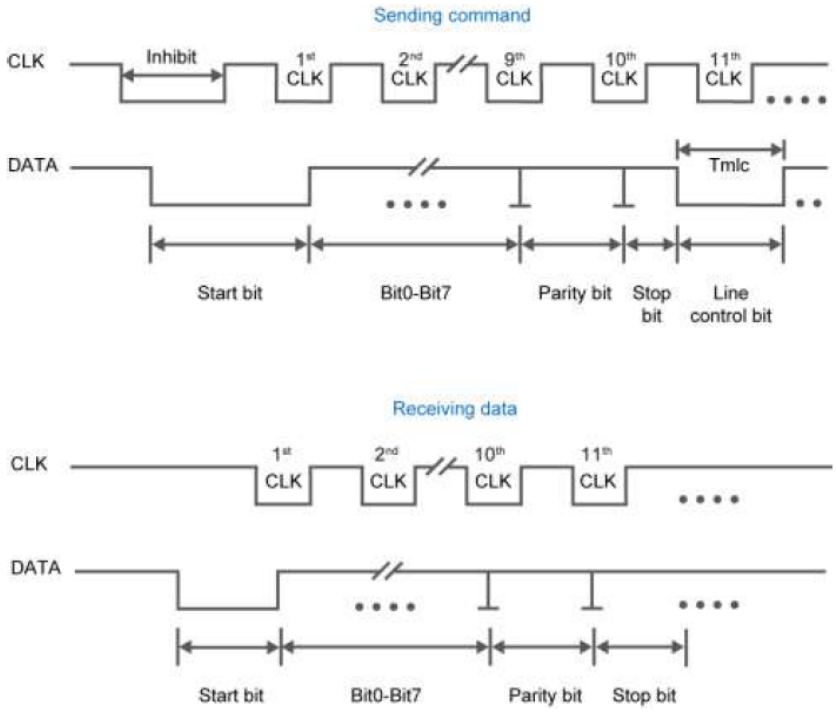

Figure 5: Wave diagram for data transmission using the PS/2 interface

LCD Display

The project will also consist of incorporating the PS/2 keyboard with an LCD display. The LCD display to be used is the one found on the HPS side of the DE10-Standard Cyclone V development board.

The LCD is connected to the SPIM0, GPIO1 controller in HPS on this DE10-Standard board. The built-in virtual memory-mapped device driver in the system is used to access the registers in the HPS SPIM and GPIO controllers. The SPI interface is used to transfer Data or Command from HPS to LCD. Because the LCD is write-only, only three SPI signals *LCM_SPIM_CLK, LCM_SPIM_SS*, and *LCM_SPIM_MOSI* are required. The *LCM_D_C* signal is used to indicate the signal transferred on the SPI bus is Data or Command. When *LCM_D_C* signal is pulled high, it means the signal on SPI bus is Data. When *LCM_D_C* signal is pulled low, it means the signal on SPI bus is Command. The *LCD_RST_n* is the reset control signal of LCD. This signal is low active. The *LCM_BK* signal is used to turn on/off the black light of the LCD. When this signal is pulled high, LCD backlight is turned on [1]. A diagram of the connections between the LCD and the HPS is provided in figure 5.

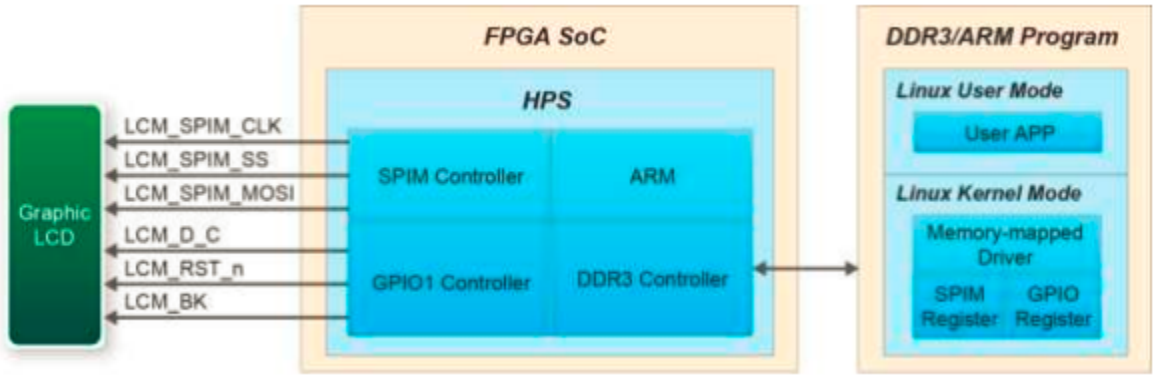

Figure 6: LCD and HPS connection diagram

LCD Control

To control the LCD display, certain initialization measures must take place beforehand:

- Common output mode select (Code: 0xC0~0xCF)
- Power control set (Code: 0x28~0x2F)
- Display start line set (Code: 0x40~0x7F)
- Page address set (Code: 0xB0~0xB8)
- Column address set (Code: 0x00 to 0x18)
- Display ON/OFF (Code: 0xAE~0xAF)

Once the display is initialized the user can start interacting with the LCD. The display area of the LCD is divided into 8 pages, so to correctly display data the developer must specify the target page and column address. The relationship between the data bits and the LCD pixels can be seen on figure 6.

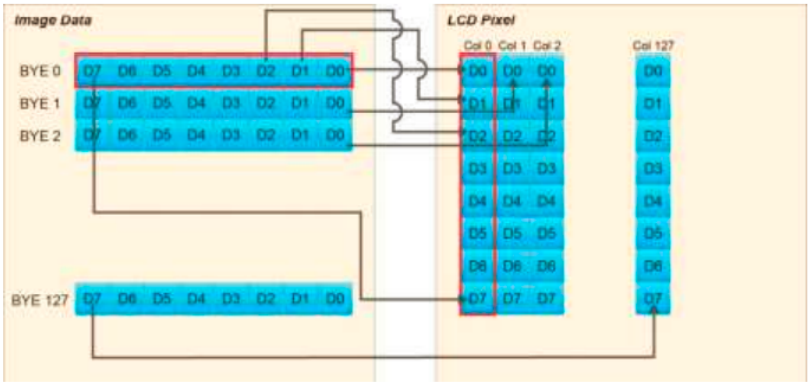

Figure 7: Relation between LCD display pixel and image data bits

The LCD display code will be in charge of all the mathematical computations involving the calculator. The LCD will obtain the raw data from the ps2 keyboard and decode it in order to obtain the necessary numbers and symbols for the calculation. After obtaining the data, decoding, and processing it, the display will show the operands, the operation and the result inserted by the user [1].

4.0 System Requirements

State	ID	Name	Description	Importance level	Priority	Comments
Active	FR-01	Use of PS/2 keyboard	The device should be able to use the PS/2 keyboard as input to the system	Required	1	
Active	FR-02	Correct identification of keys	The device should b able to correctly identify the key being pressed and ignored the unused ones	Required	2	
Active	FR-03	Mathematical operations	The device should be able to perform the four basic operations: sum, substracion, multiplication and division	Required	3	
Active	FR-04	LCD display implementation	The device must be able to send operands, the operator and the result to the LCD display.	Required	4	
Active	FR-05	Correct results	The device must display the correct operation result when the enter key is pressed.	Required	5	
Active	FR-06	Delete	The system must be able to reset when the delete key is pressed.	Required	6	
Active	FR-07	Size of operands	Operands must be at least 4 digits.	Desirable	7	

5.0 Design Description

5.1 PS/2

For the PS/2 keyboard, the PS/2 interface consists of a register named PS2_Data, which is a read/write register where the data from the keyboard is stored, and the user can read the information stored in there to retrieve the desired data. In order for the user to know when new data is available, bit 15 of the register is set to logic HIGH and the user can go ahead and read the value. Once that is known, a mask of the eight least significant bits can be used to extract the key code stored in the register [2]. Figure 8 shows a diagram of register PS2_data.

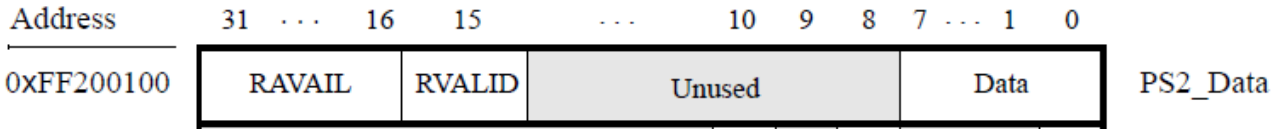

Figure 8. PS2_Data register

The code for each key is transmitted repeatedly until the key is released, then, an end of transmission code with value 0xF0 is sent just before one last transmission of the key code, by capturing the end of transmission code, is a way of debouncing the key press and make sure only one key press is detected even if the key is held down for a long time. This was implemented in the code by constantly checking the data sent by the keyboard, and once the 0xF0 is detected, the very next byte of information received (which is the actual key code being sent one last time) is used for the whole program, and no other key value is accepted until the current value (key) is done being processed with the corresponding code.

Figure 9 shows the table with the PS/2 code for each key in the keypad of the keyboard used for this project; those codes were translated into integer numbers using a switch case statement.

6C/F0 6C	Keypad 7
6B/F0 6B	Keypad 4
69/F0 69	Keypad 1
77/F0 77	Keypad /
75/F0 75	Keypad 8
73/F0 73	Keypad 5
72/F0 72	Keypad 2
70/F0 70	Keypad 0
7E/F0 7E	Keypad *
7D/F0 7D	Keypad 9
74/F0 74	Keypad 6
7A/F0 7A	Keypad 3
71/F0 71	Keypad .
84/F0 84	Keypad -
7C/F0 7C	Keypad +
79/F0 79	Keypad Enter

Figure 9. PS/2 codes for keypad numbers and operations

5.2 Mathematical logic

Regarding the mathematical logic involving the calculator, several things needed to be considered in order to get a correctly functioning calculator. We needed to filter out all the unknown keys so that they didn't interfere with the regular flow. To do this, all keys not belonging to the keypad were assigned the arbitrary value of sixteen so that we could recognize them easily. Once those keys are ignored, if the delete key is pressed, all necessary counters and flags are cleared as well as the LCD screen, the previous configurations are shown on the code in figure 10.

```
if(num == 16){ // if unknown char is detected
    count--; // keep previous value of counter
}
else if(num == 0xF){ // if delete is pressed
    count = 0;
    operand_flag = 0;
    enter_flag = 0;
    HEX_PS2(0xD, 0xD);
    // call delete function
    clear_screen();
}
```

Figure 10. Code for unknown keys and delete key

If the program flow doesn't enter in the previous conditions, it means that a valid key (aside from delete) was detected. The first thing that needs to be done is to verify that the maximum number of digits per operand (4) hasn't been reached, if that is not the case then we check for the operation that was pressed and when it was pressed, if it was pressed when the second operand (operand_flag = 1) is being computed everything needs to stay the same, on the other hand, when the first operand is being computed (operand_flag = 0), an operation can't be entered with an empty first operand, that is why the value of the counter is checked, if there is something in the operand, the number of digits entered for the first operand is saved, the operation is saved, the counter is reset, and the operand_flag changes value, see figure 11 for the detailed code.

```
if(count >= N+1){ // if maximun number of digits are entered
    count = N+1; // avoid overwritting of the operand
}
if(num==0xA || num==0xB || num==0xC || num==0xD){ // detect operation
    count --;
    if(operand_flag){ // if operand_b is being computed
        operand_flag = 1; // do nothing
    }
    else{ // if operand_a is being computed
        if(count == 0){ // avoid an operation being detected with no operand_a
            count = 0; // do nothing
        }
        else{
            count_a = count; // save number of digits in operand_a fo calculation
            count = 0;
            operand_flag = 1;
            operation = num; // save operation entered by the user
            HEX_PS2(0xA, 0xA);
            // send to LCD
            send_operation();
        }
    }
```

Figure 11. Code for valid operation key detection

Another condition is to check if the enter key was pressed, if it was pressed during the detection of the first operand, nothing happens, if it was pressed when the second operand was being detected, enter_flag is raised so the next condition is not met, if the enter_flag is set to zero, means the operands are being computed. We need to make sure we haven't reached the limit on the number of digits per operand and also that the user isn't trying to enter zero as the most significant digit, this helps to avoid errors such as trying to divide by zero. Next, the process of saving the operand is the same regarding if it is the first or the second, the only thing that changes is the array were the numbers are saved, since the counter increments when the key is detected before checking which one is it, the digit is saved in the position count-1 in the arrays. This is shown in figure 12.

```
else if(num==0xE){ // if enter is pressed
    count --;
    if(operand_flag){ // if operand_b is being computed
        enter_flag = 1;
    }
    else{ // if operand_a is being computed
        operand_flag = 0; // do nothing
    }
}
else if(enter_flag == 0){
    if(count < N+1){ // operands have valid number of digits
        if(num == 0 && count == 1){ // avoid entering zeros as the most significant digits of operands
            count --;
        }
        else{
            if(operand_flag){ // operand_b
                operand_b[count-1] = num;
                //printf("%d\n", (int)operand_b[count-1]);
                HEX_PS2(operand_b[count-1], operand_flag);
                // send to LCD
                print_num(operand_b[count-1],count_a+2+count);

            }
            else{ // operand_a
                if(count == 1){
                    clear_screen();
                }
                operand_a[count-1] = num;
                //printf("%d\n", (int)operand_a[count-1]);
                HEX_PS2(operand_a[count-1], operand_flag);
                // send to LCD
                print_num(operand_a[count-1],count-1);
            }
        }
    }
}
```

Figure 12. Code for enter and operand detection

If the enter key was pressed in the proper time (meaning there is something in the second operand), the calculation of the result takes place as it can be seen on figure 13. The digits of each operand are concatenated into just a single number using the corresponding power of 10 for each digit using the counters to know how many digits per operand are there, which is not necessarily the same for both operands. Once that is done a switch-case statement is used to make the proper calculation of the result depending on the operation entered by the user, and finally, both arrays for the operands are set to zero in every possible index so that if the next operation uses less digits as before, a previous value doesn't affect the next result. Once the result is computed, figure 14 shows that the counters and flags are reset to be ready for the next input and calculation.

```
void calculation(){
    int num_a = 0, num_b = 0;
    char i;
    int powers[N] = {1, 10, 100, 1000};

    // compute corresponding powers of 10 to operands
    for(i=0; i<count_a; i++){
        num_a += operand_a[count_a-1-i]*powers[i];
    }
    for(i=0; i<count; i++){
        num_b += operand_b[count-1-i]*powers[i];
    }

    switch(operation){
        case 0xA:
            result = num_a + num_b;
            break;
        case 0xB:
            if(num_b > num_a){ // handle negative result
                // append minus sign to LCD print
            }
            result = num_a - num_b;
            break;
        case 0xC:
            result = num_a * num_b;
            break;
        case 0xD:
            if(num_b == 0){ // handle division by zero
                result = 0;
                //printf("MATH ERROR\n");
                // send error msg to LCD
                LCD_text(math_error_lcd, 3);
                refresh_buffer();
            }
            else{
                result = num_a / num_b;
            }
            break;
        default:
            result = 0;
            break;
        }
```

Figure 13. Calculation process

```
if(enter_flag){// calculate result
    if(count > 0){ // enter was pressed with something in operand_b
        HEX_PS2(0xF, 0xF);
        calculation();
        // reset flags and counters
        count = 0;
        count_a = 0;
        operand_flag = 0;
        enter_flag = 0;
        //printf("result: %d\n", result);
        // send to LCD
        print_result(result);
    }
    else{ // if enter was pressed with no operand_b
        enter_flag = 0; // reset flag
    }
}
```

Figure 14. Enter key code process

5.3 LCD display

The DE10-Standard Computer has a liquid crystal display (LCD) port which is connected to the 128 × 64 pixel display on the DE10-Standard board. Data is written to the LCD in pages where each page consists of a 128 × 8 pixel row of the display for a total of 8 pages. Each column of a page is presented as a byte, where each bit writes to one of the 8 pixels. In order to write to the display, the page address must be written first to the data register of SPIM0 in order to then write the column addresses. Then the 8-bits of pixel data can be written to the screen. After writing one 8-bit column, the column address is incremented automatically, so that consecutive writes to the data register will be ready for the next columns [2].

In order to initialize the LCD screen, the SPIM0 with address 0xFFF00000 must be taken out of reset and put into transfer only mode, the baud rate is then set to 0x40 and the slave register is enabled. Interrupts should be turned off before SPIM0 is turned on. To turn on the backlight of the LCD, the output direction of register GPIO1 with address 0xFF709000 is set to point to the LCD. Finally, the LCD registers are initialized in order to send data to the display [2].

The DE10-Standard Computer LCD example includes functions to print lines of text, but not single characters. In order to print each individual number as the character was pressed, a new function had to be developed for this purpose. This function passes the desired position of the character inside a certain page to the frame buffer in order to print the required character. The described function can be seen on figure 15.

```
void LCD_num(char * num, int page, int pos)
{
    int c,j;
    c = (unsigned int)num;
    for (j                              = 0; j < 8; j++)
        frame_buffer[page][pos+j] = chars[c][j];
}
```

Figure 15. Custom LCD function for character print.

Some other functions were added on the main program to correctly display numbers, operations, and results

```
void print_num(char num, char pos){
    LCD_num(num+'0', 0, pos*8);
    refresh_buffer();
}

void print_operation(char num, char pos){
    LCD_num(num, 0, pos*8);
    refresh_buffer();
}

void send_operation(){
    switch(operation){
        case 0xA:
            print_operation('+', count_a+1);
            break;
        case 0xB:
            print_operation('-', count_a+1);
            break;
        case 0xC:
            print_operation('*', count_a+1);
            break;
        case 0xD:
            print_operation('/', count_a+1);
            break;

    }
}
```

Figure 16. Main program functions for displaying numbers and operations.

6.0 Tests and Results

6.1 Test Strategy

First the ps2 keyboard was tested independently using the seven segment displays available on the fpga board. The keyboard should print in the display the hexadecimal code inserted by the keypad on the side of the keyboard, since this is the only part that will be used. Numbers 0-9 are kept the same while operators +,-,*,/, enter, and delete were marked with the hexadecimals 0xA, 0xB, 0xC, 0xD, 0xE, 0xF respectively.

As a second test another seven segment display was added to inform the user of the state of the system. The system consists of four stages: input of operand A, input of operation, input of operand B and result calculation. This stage 1 was shown on the second display as a 0, stage 2 made both displays show a value of 0xAA, stage 3 made the second display show a value of 1, and stage 4 showed both displays as 0xFF. Once this flow was completed and all of the cases were debugged the fifth possible stage of delete was added as showing 0xDD on both displays and restarting the system.

The third test was made displaying on the terminal the operands and the result of operation. Some extra information was displayed to the terminal for debug purposes like internal counters, this was made in order to correctly compute the numbers provided through the keyboard with the required operation.

The final test was displaying the numbers and results on the LCD screen. This was made in order to see the correct operands on the display as well as the operation and result. The correct functionality of the system was tested.

6.2 Test Results

The first test passed without problem, the correct numbers were displayed on the seven segment display including the operations.

For the second test there were a lot of issues. There were issues related to the correct accumulation of operands, which was solved with careful usage of internal counters. There were other problems related to the advancement of the stages and special cases between stages (only accepting operation once the first operand is inserted, enter key is only available once the operand and operations were entered, only accepting four digits for operand A and B, correct reset of arrays and counters, etc). All of these issues were solved by taking into consideration all of the special cases and correctly coding them in the program, making sure the flow followed the desired logic throughout the flow of the system.Figure 17 shows the different stages represented in the seven segment displays.

Figure 17. Representation of the system stages on the seven segment displays.

The third test showed some issues regarding the correct transfer of the entered numbers into workable numbers to make operations. The issue was debugged and a wrong for loop was coded, where the loop condition was being changed by the loop itself, thus giving erroneous results. Once this issue was solved the system was working correctly.

The final test proved to have some difficulties pertaining to the correct display of the number. This was solved by using custom functions that let each individual number be written on the screen separately, with careful notice of the data type translated into the screen. Figure 18 shows the final functionality of the system.

Figure 18. Working calculator.

7.0 Verification

Criteria	Was it met?	Comments
Execution of basic mathematical operations	YES	The basic mathematical operations can be entered and seen correctly on the display
Correct display of full operation on LCD screen	YES	The operations are done correctly when entered
Clear/reset button is able to reset calculation and screen	YES	The clear button can be pressed at any time during execution to reset the system
Accepts at least 4-digit operands	YES	Operand A and operand B can both be entered with lengths of four digits
Ignores 0s left of the most significant digit.	YES	0s cannot be entered on the right side of an operand
Executes the operation (using the enter key) only if both operands and the operator are captured.	YES	The enter key has no effect on the flow of execution until both operands have been entered
Only takes into account the numeric keypad of the keyboard (ignores alphabetic keys)	YES	All keys which are not part of the extended keyboard have no effect on the display
Correctly captures data from keyboard	YES	The keys entered are correctly captured on the display and the right operations are done

8.0 Conclusions

Andrés: In conclusion, this project was an introduction into embedded system design, where a calculator using ps2 hardware, the processor and a LCD screen peripheral connected to the processor was developed. The project proved to be challenging at times, we had a lot of trouble with the custom design of a ps2 controller interface since the signals were not responding like we had anticipated. This was resolved by using the DE10-standard computer Arm setup available on the fpga monitor program. This gave us a solid base and understanding of the use of the ps2 with the processor for us to work upon and develop the necessary logic to make our project.

Jean Carlo: For me personally, this project proved to be challenging, for I had to begin working on it with little understanding or context of what our design should do or how my team was undertaking the project. I had to ask a lot of questions, while our team kept working, of basic functionalities of the keyboard, how the board captured data, performed operations, etc. in order to actually contribute to problem solving. I found this to be very useful and in the end helped me be more aware of what was going on, how I could be of use and how all of the embedded systems knowledge learned during the semester was being applied into our work.

Rodolfo: This project was an interesting and challenging way to conclude the embedded systems laboratory, it was great to be able create projects using the FPGA, NIOS and see examples with the HPS, and to have this project to make the interconnection with the FPGA and HPS was a great proof of the power embedded systems may have and also the advantage over pure HW systems, since debugging is way easier in a high-level language than in an HDL, we can confirm this, since it was easier to find the bugs about getting the operand's digits and calculating the proper result by using C language and only having Verilog to the PS72 interface rather than doing everything with Verilog.

9.0 References

[1] Terasic. "DE10-Standard User Manual", V1.0.6, 2018.

[2] Terasic. (2017). DE-10-Standard computer System with ARM Cortex A9