Alex Williams

Project 2 Report

**Introduction:**

Create a calculator that does operations on 8-bit numbers. Display the values on VGA display and use the LCD to prompt the user with instructions.

Interview with client

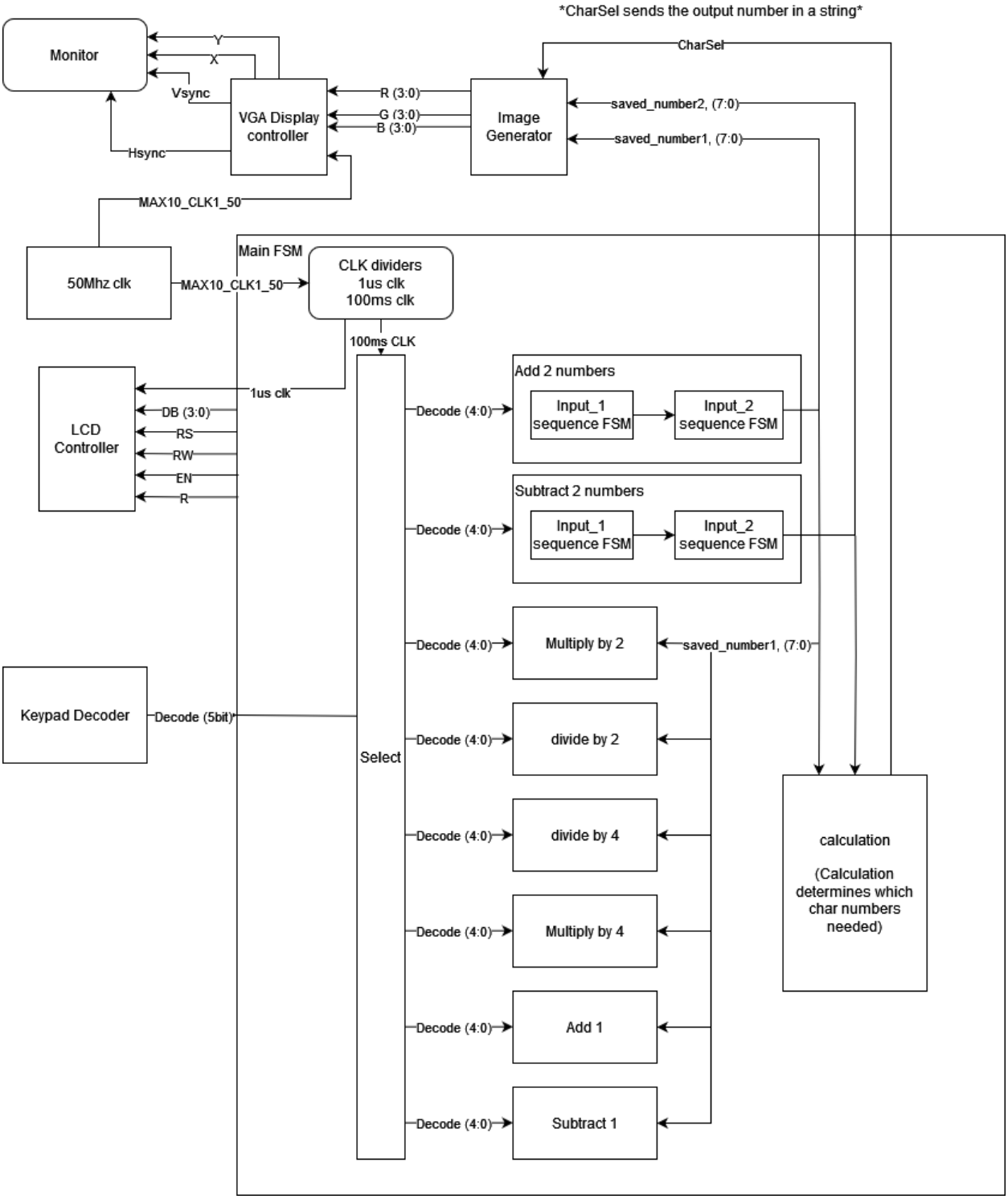
- Client wants user to press 2 keys on the keypad to represent an 8-bit hex number.
- The inputted 8-bit number should be displayed to the VGA display as it's decimal equivalent.
- The LCD should give instructions to the user.
- The client wants the following operations:
    - Add two numbers
    - Subtract two numbers
    - Multiply by 2
    - Divide by 2
    - 4 other operations

Assumptions

- LCD will say:
    - Enter first number
    - Enter OP or solve
    - Press KEY0 to solve
- Operations:
    - 4 other operations:
        - Multiply by 4
        - Divide by 4
        - Add one
        - Subtract one
    - When operations requiring only one inputted number is selected, the first number will be chosen to perform operations on.
    - A: add
    - B: subtract
    - C: multiply by 2
    - D: divide by 2
    - E: add 1
    - F: subtract 1
    - A and C: multiply by 4
    - A and D: divide by 4
- Math operations:
    - Negative numbers are not supported.
    - Integer division only.
    - Max value displayed on screen will be 999.

**Block Diagram, FSM, and component descriptions:**

Block diagram:



The Block diagram consists of the following components:

Keypad Decoder

- Decodes keypad press into a 5-bit output
- Input: Row (5-bit), Col (4-bit), CLK(1-bit)
- Output: DecodeOut (5 bit) - The MSB determines if a button is pressed

LCD display (x4 The others are exact replicas)

- DB (4-bit out) - data bus to send commands to the LCD
- RS (1-bit out) - register select
- RW (1-bit out) - Read/Write bit
- EN (1-bit out) - Enable bit
- 1USCLK (1-bit in) - clock
- R (1-bit in) - reset the LCD

VGA controller

- Input: saved_number1, saved_number2(8-bit)- saved numbers for calculation
- Output: Vsync, HSync(1-bit), X ,Y(10-bit) - VGA signal control

Image Generator (characters)

- Input: X, Y, CharX, CharY(10-bit) - Determines the location of the images
- Input: CharSel (integer) - Determines which image to display depending on the number (Applies to final_output)
- Output: red,green,blue (all 4-bit) - Colors for the chars

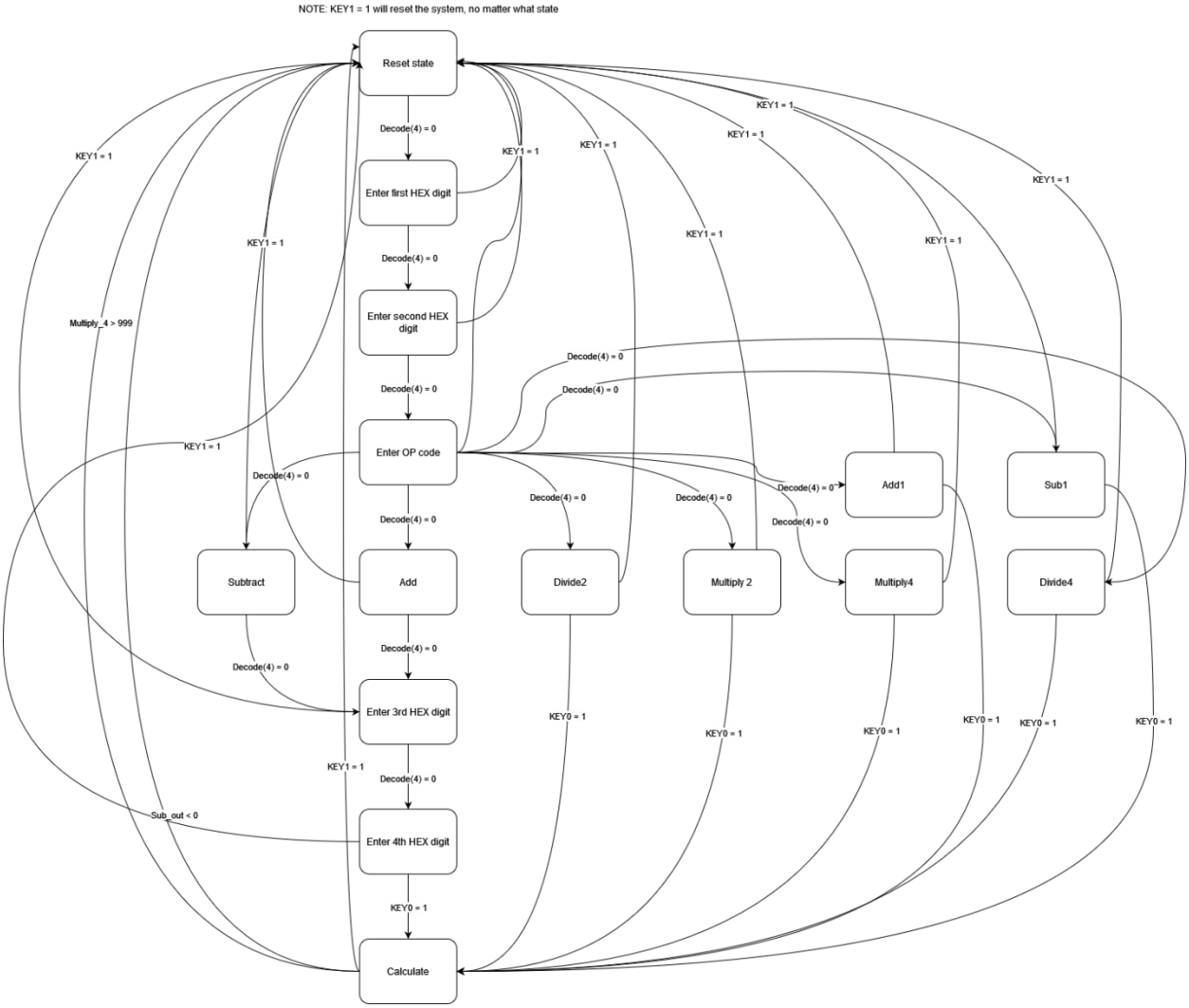
Main FSM

- ROW (4-bit in)
- COL (5-bit out) - MSB determines if a button is pressed (Active low)
- KEY (2 bit in) - used to press enter KEY[0] = 1 or reset KEY[1] = 1. One-hot encoded
- CLK (1-bit in) - system clock
- VGA_HS (1-bit out) - Vertical sync
- VGA_VS (1-bit out) - Horizontal sync
- VGA_R (4-bit out) - Red value of the characters
- VGA_G (4-bit out) - Red value of the characters
- VGA_B (4-bit out) - Red value of the characters
- DB (4-bit out) - Data bus used to send commands and or ascii to the LCD
- RS (1-bit in) - Register select. Used in conjunction with RS, RW, DB to send commands
- RW (1-bit in) - Read/Write. Used in conjunction with RS, RW, DB to send commands
- EN (1-bit in) - Enable, used to update the data sent from RS,RW,DB

<u>Main FSM Flow:</u>

- User inputs the first two 4-bit HEX digits (00-FF) to create an 8-bit number (saved_number1)
- User selects a opcode by selecting A-F or AC and AD simultaneously (8-bit one hot encoded)
- If adding or subtracting, the user enters in two more 4-bit HEX digits (00-FF) for the second 8-bit number (saved_number2)
- User presses the KEY1 button to display the output (Final_output, 10-bits)
- User presses the KEY0 button to reset the system

FSM:


NOTE: KEY1 = 1 will reset the system, no matter what state
Reset state
Enter first HEX digit
Enter second HEX digit
Enter OP code
Enter 3rd HEX digit
Enter 4th HEX digit
Calculate
Subtract
Add
Divide2
Multiply 2
Multiply4
Divide4
Add1
Sub1
Decode(4) = 0
KEY1 = 1
KEY0 = 1
Multiply_4 > 999
Sub_out < 0

**States and descriptions:**

*All states that require a user to enter digits or OP code will require DecodeOut(4) = 0*
*All states will return to the reset state if KEY1 = 1*

Reset:

- Reset all signals to initial states
- All characters are 0
- Waits for user to enter a digit (DecodeOut(4) = 0)
    - When pressed, moves to "Enter first HEX digit" state

Enter first HEX digit:

- User enters the first HEX digit
- Moves to "Enter second HEX digit" state
- Corresponding character has been placed on the screen

Enter second HEX digit:

- User enters the second HEX digit
- Corresponding character has been placed on the screen
- 8-bit HEX number is saved to memory, saved_number1
- Moves to "Enter OP code" state

Enter OP code:

- LCD prompts user to enter an op code. It also prompts user to calculate (KEY0 = 1) if the user chooses an op code that does not require a second number
- User enters OP code (A – F) or AC and AD. A for add, B for subtract, etc.
- Corresponding op code has been placed on the screen
- OP code is saved to memory
- Moves to state depending on the op code

Add:

- A has been pressed. The LCD prompts to enter a new number
- Add symbol has been displayed to the screen
- Moves to "Enter 3rd HEX digit" state

Subtract:

- B has been pressed. The LCD prompts to enter a new number
- Subtract symbol has been displayed to the screen
- Moves to "Enter 3rd HEX digit" state

Multiply2:

- C has been pressed
- Multiply by 2 has been displayed on the screen
- Moves to calculate state

Divide2:

- The D button has been pressed.
- Divide by 2 is displayed on the screen.
- Moves to the calculate state.

Add1:

- The E button has been pressed.
- Add 1 is displayed on the screen.
- Moves to the calculate state.

Sub1:

- The F button has been pressed.
  Subtract 1 is displayed on the screen.
- Moves to the calculate state.

Multiply4:

- The A and C buttons have been pressed
- Multiply by 4 is displayed on the screen
- Moves to the calculate state

Divide4:

- The A and D buttons have been pressed.
- Divide by 4 is displayed on the screen.
- Moves to the calculate state.

Enter 3rd HEX digit:

- User enters the third HEX digit
- Corresponding character has been placed on the screen
- Moves to "Enter 4th Hex digit" state

Enter 4th Hex digit:

- User enters the third HEX digit
- Corresponding character has been placed on the screen
- 8-bit HEX number is saved to memory, saved_number2
- Moves to "calculate" state

Calculate:

- LCD prompts user to press KEY0
- When KEY0 is pressed, the OP code is checked and calculation is done based on the op code
- If the calculation is greater than 999 or less than 0, the system moves to the reset state
- The final answer is displayed on the screen

**Updates to the block diagram:**

The block diagram is similar the calculator block diagram in project 1, but there have been modifications to give support for the LCD display and the VGA controller. This has also led to many changes in the main FSM. The decoder can support repeated inputs, the FSM sequence detector has the LCD and VGA character ROM attached, and the main FSM has proper reset logic. This is better because it allows for a better understanding of the FSM and the flow of the system. This also allows the designer to make changes to the systems quickly and easily.

**Pitfalls and achievements:**

Pitfalls:

- DE10 lite did not have ready-made PMODs easily accessible
- Lack of working LCD controller specifically for the DE10-Lite
    - Xilinx example LCD code was unnecessarily complex
    - Unconstrained arrays are not supported by Quartus software
    - LCD display did not properly transition
- VGA controller implementation was difficult to understand at first
- LCD cannot be reprogrammed. The board must be unplugged and plugged in again. This is an electrical issue or a possible malfunction with LCD
- Issues with consecutive presses on the keypad

Achievements:

- Wired my own PMOD connections and forced compatibility with the DE10 Lite
- Modified the example Xilinx code by removing the unconstrained array and set a constant
    - Changed delay timings to fit the slower DE10 clock
    - Utilized component instantiation to have the LCD display different patterns
- Converted a Verilog implementation of the controller to VHDL and built a custom ROM code display from the examples provided in Blackboard
- Functional overall project
- Updated the calculations

**References:**

VGA controller (Verilog) - https://habr.com/en/articles/707224/
The original code is written in Verilog, so it had to be converted to VHDL

VGA image generations - Blackboard
The ROM code (in the VGA tutorial with the bit map of the number '1') was used to write the rest of this component

LCD display controller - Digilent example code
The example has been modified to support the DE10 lite by:

- Removing the unconstrained array and setting a value
- Setting the delay timings to support 50Mhz instead of 100Mhz

Keypad Decoder - Digilent example code
The example has been modified by:

- Removing the bidirectional JA pins and assigning ROW and COL ports in a direction for easier reading of the code
- Modifying the pin assignments to support the inputs and outputs