Will Krska and Conor Naughton

EC551 Fall 2022

Lab 2

Our overall design is shown in the following block diagram

Terminal Interface Top

UART\_TX\_data

UART\_RX\_data

Key\_select\_data

Data Loader

Key\_select\_flag

Key\_select\_data

Mode\_select\_data

Mode\_select\_flag

Keyboard Interface

UART Interface

UART Transmitter

UART Receiver

PS/2 Controller

As shown, the design can be broken down into three main segments, a processing element and interfaces for handling both the keyboard and UART signals. The following is a description of these segments.

**Processing Element/Data Loader**

This block

**PS/2 Keyboard Controller**

To build our keyboard controller, we started with the demo provided in the lab manual as a base and built the additional functionality needed for this lab. In particular, within the PS/2 receiver module, we added a new block which is triggered on the positive edge of the clock and checks the second byte of the keycode variable to identify whether a key release has occurred (this happens when the second byte is F0). When this happens, we set a keystroke flag high and send the PS/2 byte value of the keystroke up to the top keyboard module. When we are coming from the initial state, the next step in this block is to check to see if the current keystroke was the enter key (this happens when the first byte is 5A). This indicates that the user has entered a mode selection value and triggers a modeflag to go high as well as sending the byte value of the mode to the top module. Once a valid mode byte (I, L, A or B) has been sent, the module continues to send keystrokes to the top module as they appear which will be accumulated for instruction formation. In the keyboard top module, there are finite state machines for both the keystroke and mode data which are entered on each clock cycle and stay in the initial state until they receive a high flag. Once they get the high flag, they hold a corresponding flag low for another clock cycle and then put it high and sent that signal up to the overall terminal interface module. The reason for this sequence is because the PS2 receive module runs with a 50MHz clock while the top module runs with the system 100MHz clock so if we only passed through the high flag, it would get sent twice before the flag going low was recognized.

**UART Interface**

Our UART interface also built upon the demo code which was linked in the lab manual. This example code simply loaded in an introductory message and then sent another message whenever a button was pressed. It accomplishes this using a case statement which controls when messages are sent to UART and adds delays in order to provide time for the message send process to complete. These states include a Wait\_Mode which is where the state machine sits until it gets a flag identifying that it has received a mode select or a keystroke. We modified the introductory message as directed in the lab manual and then added logic within the case statement which controlled the messages being sent over UART to identify which mode has been entered based on the mode\_select signal originating from the keyboard controller. Based on which mode was selected, a predefined array of strings is loaded into the UART using the same byte by byte method used for the introductory message. In the case that the mode selected is not amongst those which we intend to use for this lab, a invalid mode message is sent and the state machine returns to the loop looking waiting for the correct mode to arrive.

**Terminal Interface**

Our terminal interface

Will summary of almost every source (ignoring the very obvious ones and the ones in the datapath)

1. alu/hex/mat/ps2\_to\_ascii – takes the values and translates them to ascii values
2. data\_loader: a very large FSM. Takes in a given key presses translated from PS/2 and from UART. On startup, it is in idle, and is looking for the mode selection. Any keypress will assign to the appropriate mode output and mode flag (0->4). These are used by the UART controller. It will then go into the appropriate sequence for the specified mode. This can be surmised from the code. The state of the FSM is represented on the LEDS. This also contains 2 main modules: mat3mult, and mini\_ALU. These correspond to the operations done by the benchmark and ALU mode respectively (will be described later). The results of these values are output to the top file to be processed by the uart module.
   1. I mode: takes in 4 valid chars (0-F) then goes into wait state for enter. On enter, the address and instruction are asserted and written to the datapath memory. After enter, if R is pressed, goes to run state and ap\_start is asserted for the datapath, then returns to idle
   2. L mode: same, but with UART input instead of keyoard
   3. A: takes in valid char (0-F), then next state takes in shift ONLY, then next state takes in keypress associated with the operation, then next state takes in valid char 0-F, then last state writes back ALU output
   4. B: takes in 9 valid chars (0-F), and ignores everything else (even thought the user is supposed to input a space in between each num, and enter at the end of the row). After those , it moves on to the next matrix and does the same thing. After that it goes to the writeback state and outputs the result
3. datapath: almost the same datapath from lab 1
4. mat3mult: basically a container for 9 instances of vect3mac. Takes in 2 vectors of 9 hex values representing a 3x3 matrix, and outputs a 3x3 matrix of bytes. In a generate block, it assigns the appropriate rows and columns to vect3mac
5. mini\_ALU: purely combinational ALU for the 4 operations (xor, add, sub, mult). Takes in 2 4bit, outputs 8bit
6. PS2 reciever: modified, now on a keypress the FSM detects the xF0 then latches the keystroke output to whatever byte follows and asserts flag (NOT a pulse)
7. Temp\_top: ignore this one
8. Term int top: mostly unchanged, just contains everything
9. Top\_ keyboard: modified, now passes through the keypress from PS2reciever, but turns the flag into a 1 clk (at 100mhz) pulse.
10. Uart arbiter: takes in the uart signal, and asserts the keystroke and and 1 clock (100 mhz) pulse
11. Vect3mac: takes in 2 vectors of 3 hex vals (12 bits) and performce MAC/vector multiplication, and outputs a 8bit value