Ronnie Whyrick

ECE – 431

Dr. Devore

8th of December 2017

The first thing I did was include the enable interrupts macro and peripheral declarations. I then defined a macro of 125 for second count, which when counted at 8ms intervals adds to a second. I set, for my getkey(); function a COL1, and COL2 O/P. then my RED and green LEDs and set my different states for my two state machines. I then wrote the declaration for all of my functions throughout my code. Following that I declared my variables and initialized some of them to a value.

Inside my main function I established and enabled all of the different CPU, BusClock, Port, and Timer regulations (both MTIM and TPM). I set the oscillator to trim to 8MHz, had it delay 2ms at 4MHz, then set my speed up to 8 MHz again. I set all of my PTAD0-3 ports to outputs. I set every port B pull-up enable. I next set the drive strength of my port A pins that would be driving my LEDs. I set my MTIMER to the bus clock divided by the 256, with this I made my period exactly 8ms. I also set the timer and enabled TOF ISRs. For my TPM timer I enabled the period to instead by a little over 1 second with the maximum PS allowed. Following that I configured that TPM to be an ISR based output compare – Output Toggle function. Lastly I set the initial state of my REDLED and GREENLED to zero, and enabled interrupts.

My forever loop was empty, as it would have only hindered the shifting process. My Modulo Timer, which is set to activate every 8ms is stated simply that whenever the TOF flag is full, clear It and call the keystate function. Inside my keystate function I have 3 states: wait for a key to be pressed, decode that key when it is pressed, then wait for that key to be released. To trigger the press and release I set check to see if any of the pulled-up Port B pins (initially set as 1) have changed values to a zero, and to not allow key presses during lit LEDs I also demanded that both LEDs be shut off. If the conditions are true the switch statement being pre-defined Macros held to the switching variable of unsigned char combostate = the next case – which is decode. Inside of the same case however I have a Boolean failsafe to keep my case it the correct place every 8ms, as if no key is pressed when the LEDs are off, then the case will remain the same. Inside the decode state the function Getkey is called, inside this function is the brute force determining algorithm to decipher between columns and rows of the keypad which key was pressed. Upon doing this I will reference the the ASCIItable above which is a character string of all the keys in order from left-right then top-bottom. Upon acquiring whatever key was pressed the decode case comes back and calls the final state machine codestate. Once codestate is called the variable key is checked in the initial case statement as 'good'. This is where it checks to see if a * or # is pressed if either is pressed the program will reset nextdigit and send the cdstate variable back to 'good' or if the pointer nextdigit is pointing to a null it will light the GREENLED and reset the counting and nextdigit parameters, respectively. If neither of these keys are the ones pressed the program looks to see if the variable key, which hold the current defined ASCII character is the next digit that the pointer *nextdigit is pointing to. If it is, *nextdigit will be incremented pointing to the next character in the string. But if key is not equal to this next character string character then the state bad is called. Upon doing this the program will hop back to the first state machine and sets combostate = wait4release where the only escape out of this case is the release of a the button, this is checked the same way a press is checked (one of the electrical signals to one of the rows has been connected & the pulled up port reads a 0). If the case was changed to bad, the next time that codestate is called (ie another key was pressed) the bad function will be read. If the bad function is read, the only escape out (without incrementing a badcount) is the reset symbol *. If * is not pressed the bad case will be held there until key becomes the # or confirm key. In this case one of two things will happen. If you have messed up the code less than 3 times, the REDLED will just turn on for 2 seconds and light RED, if you have exceeded 3 counts, then the badcount will reset and the REDLED will turn on for 10 seconds.

The final part of the program is the TPM timer. This timer is called a little over every second and checks the tpmcountdown variable which is set by the setting of 1 or more of the LEDs if either LED is set it sets a tpmcountdown time. Inside the ISR if tpmcountdown is equal to 0, the timer will just return to the programs original protocol and reset, if not, the ISR will de-increment this variable until it satisfies the former statement.