Faculty of Engineering
University of Ottawa

Lab 2: Hardware Interfacing – Keypad

Computer Architecture II
CEG3136

Jonathan Wong - 8801941
Tae Kim - 8391046
Group 18

Lab performed: October 4th & 11, 2018
Lab due: October 18, 2018

Table of Content

Objectives:

To introduce the interfacing of the Motorola 9S12DG256 through an implementation of a keypad unit.

Equipment Used:

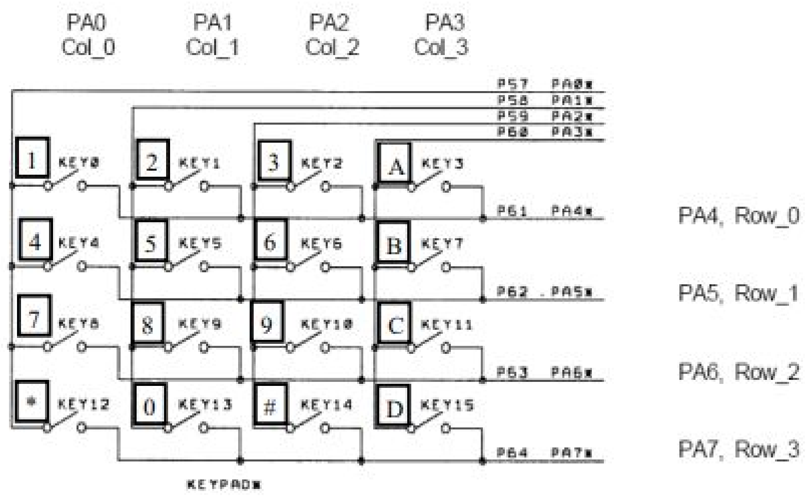
- Windows PC
- Dragon 12 Plus Trainer

Purpose

This lab presents the design of an alarm system simulation dependent on the assembler and executed on the Dragon 12 Plus Trainer. Based upon on modular and structured programming , the system will have a variant amount of features.

Discussion

Prelab: Operation of the KeyPad



Keypad connections:
PA0 connects COL0 of the keypad
PA1 connects COL1 of the keypad
PA2 connects COL2 of the keypad
PA3 connects COL3 of the keypad
PA4 connects ROW0 of the keypad
PA5 connects ROW1 of the keypad
PA6 connects ROW2 of the keypad
PA7 connects ROW3 of the keypad

Figure 1: Connection of Keypad to Port A Taken from the Lab Module 2, Switch Module Design

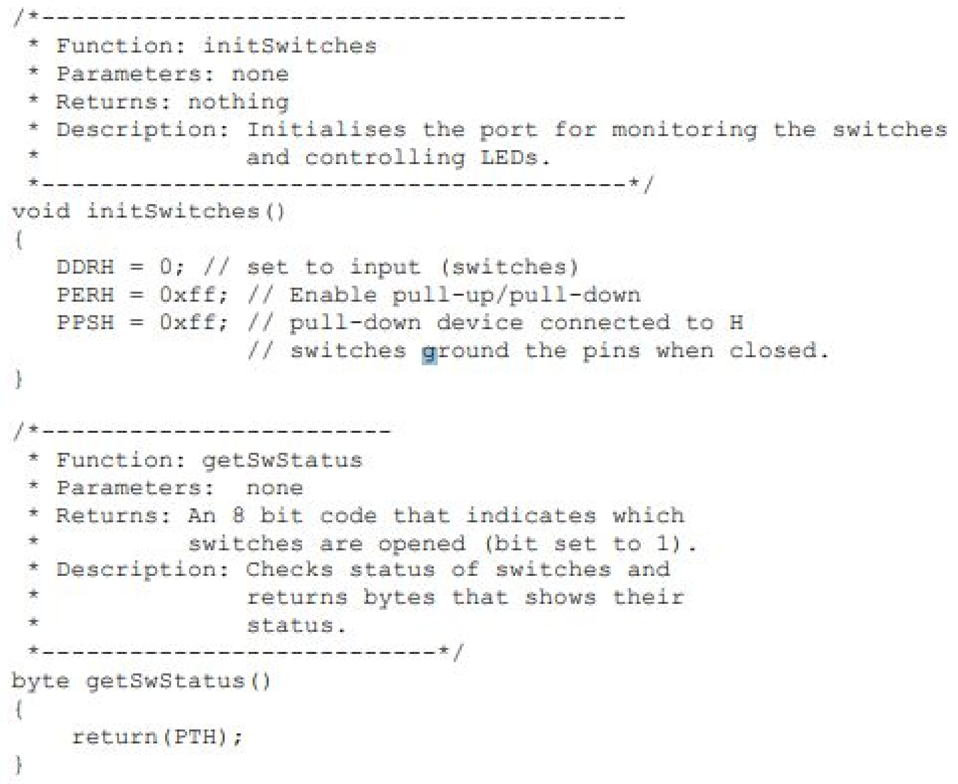
Part 1: KeyPad

The first task was implementing KeyPad.asm, which was the middle piece that was used to communicate the keyboard on the Dragon12 Board with the user. The feature of this is used to scan the inputs and associate them with a numerical value of ASCII code nature.

There are multiple functions of KeyPad.asm that required modification;

- initSwitches: Initialises Port H pins as input pins for reading the status of the switches. Pull-up resistors are enabled since the switches pull the voltage on the pins to ground (Taken from the Lab Module 2).

- getSwStatus: This subroutine simply returns the contents of the PORT H data register which reflects the status of the pins. When a pin is high, the switch is open, when a pin is low, the switch is closed (Taken from the Lab Module 2, Switch Module Design).



```
/*-----------------------------------------
 * Function: initSwitches
 * Parameters: none
 * Returns: nothing
 * Description: Initialises the port for monitoring the switches
 *              and controlling LEDs.
 *-----------------------------------------*/
void initSwitches()
{
   DDRH = 0; // set to input (switches)
   PERH = 0xff; // Enable pull-up/pull-down
   PPSH = 0xff; // pull-down device connected to H
                // switches ground the pins when closed.
}

/*------------------------
 * Function: getSwStatus
 * Parameters:  none
 * Returns: An 8 bit code that indicates which
 *          switches are opened (bit set to 1).
 * Description: Checks status of switches and
 *              returns bytes that shows their
 *              status.
 *---------------------------*/
byte getSwStatus()
{
    return(PTH);
}
```

Figure 2: Pseudo C Code of initSwitches() and getSwStatus(), taken fromthe Lab Module 2, Switch Module Design

pollReadKey was a function that allowed for the calling program to test whether or not a key was pressed and if pressed returning its value. This could be determined regardless of whether or not the key were pressed.

```
char pollReadKey() {
      char ch = NOKEY;            // Initialized as no key.
      int count = POLLCOUNT;      // Number of checks to be done
      PORTA = 0x0f;               // Resets PORTA
      do {
      if (PORTA != 0x0f) {        // Checks for key press
            delayms(1);
             if (PORTA != 0x0f) {  // Check again for key press
                  ch = readKey();   // Reads key and converts to
ASCII
                  break;          // End loop
            }
      }
      count--;
      } while (count);
      return ch;
}
```

Figure 3: Pseudo C Code of pollReadKey()

Another crucial function that was implemented is the readKey. The purpose of this function reads the value associated with the pressed key, guaranteeing that debouncing occurs for both pressing and releasing of set key. After a 10 ms wait, confirming a key was pressed and once the keypress signal is stable, the key is read using readKeyCode() subroutine which then it’s outputs are translated into an ASCII character using the readKeyCode() subroutine. After the key is released, the program then returns transitioning to the calling function.

```
byte readKey(){
      byte ch;
      do
      {
            PORTA = 0x0F; //set all output pins PA7-PA4 to 0
            while(PORTA == 0x0F) // Checking for the leading edge
//bits PA3-PA0 are 1 until a key is pressed)
            {
                  code = PORTA; // get the keycode from the user
                  delayms(10); //Delay for the debounce of button
            }
      }while(code != PORTA); //start again when PORTA changes
      code = readKeyCode(); //call readKeyCode to get keycode
      PORTA = 0x0F; // set pins PA7-PA4 to 0
      while(PORTA != 0x0F) // wait for trailing edge
      {
            delayms(10); //delay for the debounce of the key
      }
      ch = translate(code);//call translate to get ASCII code
      return(ch);
}
```

Figure 4: Pseudo C Code of readKey()

Part 2: Delay.asm

In part 2 , we were tasked with creating a subroutine called delayms, that functions as a implementing a delay of 10ms. Inputs were a unit of ms while the output is void. Using two loops, the outer loop delays in multiples of milliseconds while the inner loop causes a delay of 1 millisecond. The inner loop takes into account the number of cycles required by the outer loop to perform its functions.

```
void delayms(unsigned int milli) {
     unsigned int delayCycles;

     while (milli--) {
     delayCycles = 0x12BD; // Number of cycles for a 1 MS delay.

     do {
          // NOP
     } while(--delayCycles)
     // Few extra NOP
     }
     return;
}
```

Figure 5: Pseudo C Code of delayms()

Conclusion

This lab taught us how inputs can be read from the DragonBoard as well the challenges of coding software to define denounces and scanning techniques. Not only that, we were able to implement and control the system's delay in assembly using subroutines and modular code, teaching us real life lessons of code modification to fit into a desired result._After much trial and error, and lots of translation of C to assembly, all functions of the alarm.asm were viable as they produced their hypothetical function.