ECE 3724/CS 3124 HW #6 PIC Hardware Introduction & LED/Switch IO – Reese Solutions

1. The –M*file* option to the Hi Tech PICC-18 compiler produces a map file that specifies the memory locations of subroutines and variables. Compile the `ledflash.c` code found on the course lab page with full optimization and give the locations of `main()`, `a_delay()`, and variable `i` of `a_delay()`.

Compiled with Linux version 8.30: picc18 –O –18F242 –Mledflash.map ledflash.c

Map file contains:

```
 _main        text       00005C   (location 0x005C in memory).
_a_delay      text       00001C   (location 0x001C in memory).
```

The variable 'i' is not in the map file because it is a local variable. Loading the .hex file into MPLAB and looking at the code, look beginning at location a_delay (location 0x001C) for some code that initializes a data location to 200 (0x00ca). I found:

```
movlw   0xc8     ;w = 0xc8
movlb   0        ; select bank 0
movwf   0xfc, BANKED        ; 0xfc is the LEAST significant byte of 'i'
clrf    0xfd,BANKED         ;oxfd is the MOST significant byte of 'i'.
```

So variable 'int i' is located in BANK 0, locations 0xfc (LSByte), 0xfd (MSByte).

If you compile with –a200 flag, main and a_delay is shifted by 0x200:

```
 _main        text       00025C   (location 0x025C in memory).
_a_delay      text       00021C   (location 0x021C in memory).
```

However, variable 'i' location is unchanged.

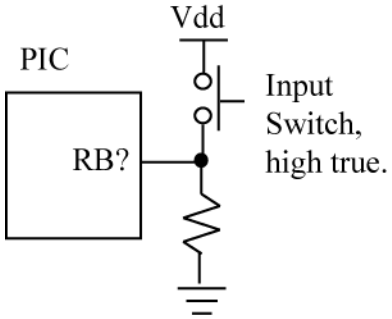
```
a_delay(){
  unsigned int i,k;
  /* change count values to alter delay */
  for (k=1800; --k;) {
    for(i = 200 ; --i;);
  }
}
```

variable 'i' is two bytes because it is an 'int'

2. Compile `ledflash.c` with (-O) and without optimization (without –O flag) and compare the code sizes.

With –O flag: 86 bytes, without –O flag: 110 bytes.

3. Draw a schematic for a pushbutton switch that implements a high-true function, i.e, provides a logic one when pressed and a logic zero when not pressed. You can only use a resistor and a pushbutton.



4. Modify the *reset.c* code on course lab page to provide a choice for software reset (the RESET instruction), and print out a message if this type of reset occurs. If this choice is selected, then perform a software reset. Read in the PIC18 datasheet about how to detect a software reset via the RCON register.

```
persistent char reset_cnt;

main(void){
  int i;
  char c;

  serial_init(95,1); // 19200 in HSPLL mode, crystal = 7.3728 MHz
  pcrlf();
  if (POR == 0){
    printf("Power-on reset has occurred."); pcrlf();
    POR = 1;  // setting to bit to 1 means that will
              // remain a '1' for other reset types
    reset_cnt = 0;
  }
  if (TO == 0) {
    SWDTEN = 0; // disable watchdog timer
    printf("Watchdog timer reset has occurred."); pcrlf();
  }
  if (RI == 0) {
    printf("Software reset has occurred!"); pcrlf();
  }
  reset_cnt;
  printf("Reset cnt is: %d",i);
  pcrlf();
  reset_cnt++;
  while(1) {
    printf("'1' to enable watchdog timer"); pcrlf();
    printf("'2' for sleep mode");  pcrlf();
    printf("'3 ' for both watchdog timer and sleep mode");  pcrlf();
    printf("'4 ' software reset ");  pcrlf();
    printf("Anything else does nothing, enter keypress: ");
    c = getch();
    putch(c);
    pcrlf();
    if (c == '1') SWDTEN = 1;  // enable watchdog timer
    else if (c == '2') asm("sleep");
    else if (c == '3') {
      SWDTEN = 1;     // enable watchdog timer
      asm("sleep");
    }
    else if (c == '4') {
      asm("reset");
    }
}
```

Detects software reset.

New choice for software reset.

Do software reset

5. From the PIC18Fxx2 datasheet, what is the typical Idd current in HS mode for a crystal frequency of 10 MHz, and a Vdd of 5 V? (Hint: Look at the DC Graphs and Tables section).

Between 4.5 mA and 5 mA

6. For an LED/switch configuration of LED on port RB4 and switch on port RB6, write code that blinks the LED once per second if the switch is pushed, else it blinks at twice per second. Use the DelayMs subroutine to implement the delays.

```
char i;

TRISB4 = 0; // RB4 output
TRISB6 = 1; //RB6 input
while(1) {
//toggle LED
 if (LB4) RB4 = 0; else RB4 = 1;
  if (!RB6) {
    // switch is pushed, so delay for ½ second
    DelayMs(200); DelayMs(200); DelayMs(100);
  } else {
   // switch is not pushed, delay for ¼ second
   DelayMs(200); DelayMs(50);
   }
 }
```

Two loop iterations will blink the led, so this value is one-half second as two loop iterations is 1 second (blink the LED once per second if switch is pushed). One-half second is 500 mS.

Two loop iterations will blink the led, so this value is one-quarter second as two loop iterations is one-half second (blink the LED twice per second if switch is not pushed).One quarter second is 250 ms.

7. Assume a low-true pushbutton input on RB4, and four high-true LEDs connected to pins RB0 through RB3. Write *C* code that configures PORTB for this operation, with the LEDs initially off. Then enter a loop, where each press and release of the switch turns the LEDs on in sequence, with the LED previously on being turned off (i.e. 1st press/release, RB0 is 1/RB3 is 0, 2nd press/release RB1 is 1/RB0 is 0, 3rd press/release RB2 is 1/RB1 is 0, 4th press/release RB3 is 1/RB2 is 0, repeat).

```
 TRISB = 0xF0;   //RB7-RB4 inputs, RB3:RB0 outputs
 PORTB = 0;      // all LEDs are off
while(1) {
 // wait for press/release
 while(RB4);  while(!RB4);
 RB0 =1; RB3 = 0;
while(RB4);  while(!RB4);
 RB1 =1; RB0 = 0;
while(RB4);  while(!RB4);
 RB2 =1; RB1 = 0;
while(RB4);  while(!RB4);
 RB3 =1; RB2 = 0;
  }
```

8. The HSPLL oscillator option multiplies the crystal frequency by 4 internally, while the HS option does not. If a PIC18 is drawing 16 mA using the HSPLL option, what is the expected current draw if the HS option is used with the same crystal?

Current varies linearly with frequency, so expected current draw would be 16 mA/4 ~ 4 mA

9. Assume RA4 is configured as an output and the assignment `RA4 = 0` is executed. What does a read of RA4 return? What does a read of LATA4 return? Now assume the assignment `RA4=1` is executed. What does a read of RA4 return? What does a read of LATA4 return?

Reading LATA4 returns the last thing written to LATA4, while reading RB4 reads the external pin value. A complication is that RA4 is an open-drain output, it can only pull low.

After `RA4 = 0` is done, the output pin is pulled low, so reading RA4 returns a '0'.
Reading LATA4 also returns a '0'.

After `RA4 = 1` is done, the output pin is now floating, so reading RA4 returns an unpredictable value, it could either read as a '0' or as a '1'. Reading LATA4 will return a '1' as that is the contents of the LATA4 latch.