

ECE 3724 Test #3 – Sumer 2006 You may use only the provided reference materials. All figures are on the last pages; questions are on pages 1-8 (four sheets, double sided).

Part I: (80 pts)

- a. (5 pts) Write C code that configures PORTB for the IO shown in the figure for problem (a) on the Figure sheet. The internal weak pullup must be enabled. Do not assume any default bit values.

```
//one solution
RBPU = 0;
TRISB2 = 1;
TRISB0 = 1;
TRISB3 = 0;
```

- b. (15 pts) Assuming the IO configuration of the previous problem, write a *while(1){}* loop that implements the LED/Switch IO state machine shown for problem (b) in the figures. Either use a *switch()* statement approach or a *if-then-else* approach. Assume you have available the *DelayMs()* function for blinking the LED; you do NOT have to include debounce delays for the switch input.

```
#define START_STATE 0
#define BLINK_STATE 1
#define TOGL_STATE 2

// This code sits inside main(), assume configuration from problem 1a.
unsigned char state = START_STATE;
while (1)
{
    switch (state)
    {
        case START_STATE :
            RB3 = 0;          // LED off
            while (RB2);     // Wait until a press occurs
            state = BLINK_STATE;
            break;

        case BLINK_STATE :
            LB3 = !LB3;      // Toggle LED
            DelayMs(250);    // Delay to make toggle visible
            if (RB2) state = TOGL_STATE;        // If button released
            break;

        case TOGL_STATE :
            LB3 = !LB3;      // Toggle LED for blinking
            while (RB2);     // Wait until a press occurs
            while (!RB2);    // Wait until a release occurs

            if (RB0) START_STATE;
            break;
    }
}
```

- c. (20 pts) For the switch configuration shown in Figure (a), implement the flowchart of Figure(b) using an interrupt to handle the button on the RB2 input. Use semaphores controlled by the ISR to tell the *while(1)* loop in the main() program to blink, turn on or turn off the LED.

1. (13 pts) ISR code

```
volatile unsigned char blink = 0;
volatile unsigned char led_on = 0;
volatile unsigned char state = START_STATE

void interrupt myISR()
{
    if (INT2IF){
        DelayMS(30);    //debounce
        INT2IF = 0;
        switch(state) {
            case START_STATE :
                INTEDGE2 = 1;    // rising edge
                blink = 1;
                state = BLINK_STATE;
                break;
            case BLINK_STATE :
                blink = 0;
                state = TOGL_STATE;
                break; //no need to toggle led on first release
                    //since LED state is unknown (either off or on)
            case TOGL_STATE:
                if (!RB0) {
                    led_on = !led_on;
                } else {
                    INTEDGE2 = 0; //falling edge
                    led_on = 0;
                    state = START_STATE;
                }
                break;
        }
    }
}
```

2. (7 pts) main() code, be sure to configure and enable your interrupt.

```
main(){
    IPEN = 0;        // Turn off interrupt priority system
    INT2IF = 0;     // Clear any pending pushbutton presses
    INT2IE = 1;     // Enable pushbutton interrupt
    INTEDG2 = 0;    // Look for a falling edge initially
    GIE = 1;        // Enable interrupts
    while (1) {
        if (led_blink) {
            LATB3 = !LATB3; DelayMs(200);
        } else if (led_on) LATB3 = 1;
        else LATB3 = 0;
    }
}
```

- d. (7 pts) Assume an asynchronous serial channel with a data format of 1 start bit, 8 data bits, and 2 stop bits between characters. Assume the PIC has an FOSC of 40 MHz. How many instruction cycles will take for the UART on the PIC to overflow for a baud rate of 19200?

After three characters, will overflow (not counting leading edge of 4th start bit).

So $3 * (8+2+1) / 19200 = 1718.8$ us to overflow

One instruction cycle = $4/40$ MHz = 0.1 us

Total instruction cycles to overflow = $1718.8/0.1 = 17188$ instructions

- e. (7 pts) Write C code that implements the *char getch(void)* function (read one character from the serial port). *No interrupts are enabled.*

```
char getch(void) {
    while(!RCIF);           //wait for character
    return(RCREG);         //return
}
```

- f. (9 pts) Assume the definitions of a circular buffer that we have used in lab (i.e, the head pointer is used to place data into the buffer, the tail pointer is used to take data out of the buffer, the buffer is empty when head is equal to tail, and that pointers are incremented and wrapped before used to access the buffer).

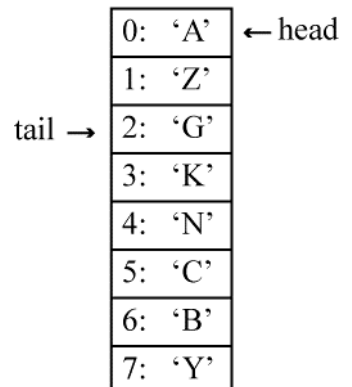
f1. From figure F, how many characters are currently *available* in the buffer? (this is not the total number of locations in the buffer) 6

f2. From figure F, what character is returned if the buffer is read?

K

f3. From figure F, what location is modified if one character is written to the buffer? 1

Problem (f)



g. (9 pts) Given the code below and Figure (g), answer the questions – assume that the switches do NOT bounce. You MUST JUSTIFY your answer or no credit is given.

g.1 Assume the PIC is powered on, and then Button A is pressed and released, then Button B is pressed and released. At this point, what is the value of count?

The press/release of ButtonA sets the INT1IF flag, but does not cause an interrupt. The press/release of Button B sets the INT2IF flag, and causes an interrupt. Both flags are set, so count is incremented twice. After the ISR is exited, count =2 .

g.2 Assume the PIC is powered on, and then Button B is pressed and released *twice*, then Button A is pressed and released. At this point, what is the value of count?

The first press/release of ButtonB sets the INT2IF flag, and causes an interrupt, so count =1. The second press/release of Button B sets the INT2IF flag, but does not cause an interrupt. The press/release of Button A sets the INT1IF flag, and causes an interrupt. In the ISR, both flags are set, so count is incremented twice. Final count value is 3.

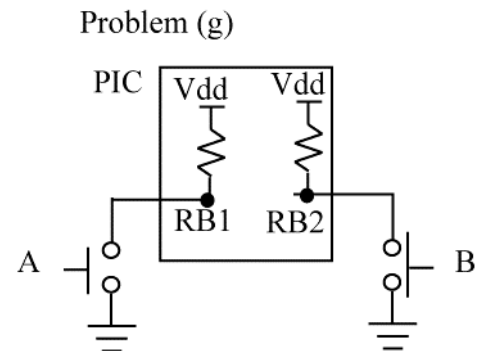
g.3 Assume the PIC is powered on, and then Button A is pressed and released *twice*, then Button B is pressed and released. At this point, what is the value of count?

Same as (1) (final count = 2), because the multiple presses of ButtonA only sets the flag and does not cause an interrupt.

```

interrupt isr() {
    if (INT1IF) {
        INT1IF = 0; INT1IE = 0; INT2IE = 1;
        count++;
    }
    if (INT2IF) {
        INT2IF = 0; INT2IE = 0; INT1IE = 1;
        count++;
    }
}
main() {
    RBPU = 0; TRISB = 0xFF;
    IPEN = 0; INTEDG1 = 1; INTEDG2 = 1;
    INT1IF = 0; INT2IF = 0;
    INT1IE = 0; INT2IE = 1;
    count = 0;
    PEIE = 1; GIE = 1;
    while (1){
    }
}

```



- h. (8 pts) In the code below, there are two possibilities about what can happen, depending upon how fast the user hits a key after the message is printed. Explain both scenarios.

```
main() {
    char c;
    serial_init(95,1); // 19200 in HSPLL mode, crystal = 7.3728 MHz
    printf("Howdy!"); pcrLf();
    while (1) {
        printf("Hit any key: ");
        SWDTEN = 1;
        getch();
        asm("sleep");
        SWDTEN=0;
        pcrLf();
    } //end while()
} //end main()
```

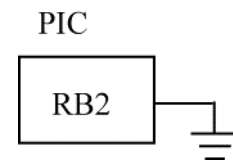
If the user presses a key before the watchdog timer expires, the PIC18 enters sleep mode, and then the WDT wakes it up, and 'hit any key' is reprinted.

If the user waits long enough, the watchdog timer expires, resetting the PIC18, causing the 'Howdy' message to printed.

Part II: (20 pts) Answer 5 out of the next 7 questions. Cross out the 2 questions that you do not want graded. Each question is worth 4 pts.

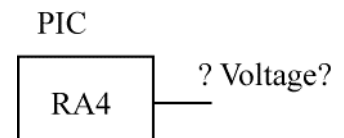
1. In the diagram below, Port RB2 is configured as an OUTPUT. If I write a '1' to RB2, using "RB2 = 1", what is returned (0 or 1) if I read RB2? if I read LATB2?

Reading RB2 returns a '0' since it returns the state of the pin (0 V). Reading LATB2 returns a '1' since it returns the last value written.



2. Assume that RA4 has been configured to be a digital output. RA4 is an OPEN-DRAIN output on the PIC. If I write a '0' to RA4 (RA4 = 0), what voltage will I read on the output? If I write a '1' to RA4 (RA4 = 1), what voltage will I read on the output?

An open drain output cannot pull high. Writing a '0' causes the output to be pulled to ground, so the output will measure as 0V. Writing a '1' causes the output to float, so the voltage could be anywhere between 0 and Vdd.



3. Assume a crystal oscillator with a value of 8 MHz is connected to the PIC, and that the HSPLL option is enabled. What value has to be written to the SPBRG register to generate a baud rate of 38400 using high-speed mode? Show your calculations.

HSPLL option sets FOSC = 4 * crystal freq = 32 MHz
SPBRG = [(32 MHz)/(16 * 38400)] - 1 = 51

4. Explain the difference between a HALF-DUPLEX communication channel and a FULL-DUPLEX communication channel. Which of these terms describe the RS232 standard?

Half-duplex – communication both ways, but only one direction at a time
duplex – communication both directions, simultaneously
RS232 has both TX, RX pins, is a duplex link.

5. Assume that a guy from Ole Miss decided that sending only 8 bits + 1 start + 1 stop bit was too slow for him in asynchronous serial transfer, and he modified the asynchronous serial protocol so that it would send 64 bits + 1 start + 1 stop bit (assume we do indeed have enough data to fill up the 64 bits. What is the problem with this?

Asynchronous communication does not send a clock with the data, so mismatches in clocks (generated by some oscillator source on each CPU) on both sides accumulate with the number of bits that is sent in one transmission. Increasing the number of bits reduces the amount of clock mismatch that can be tolerated.

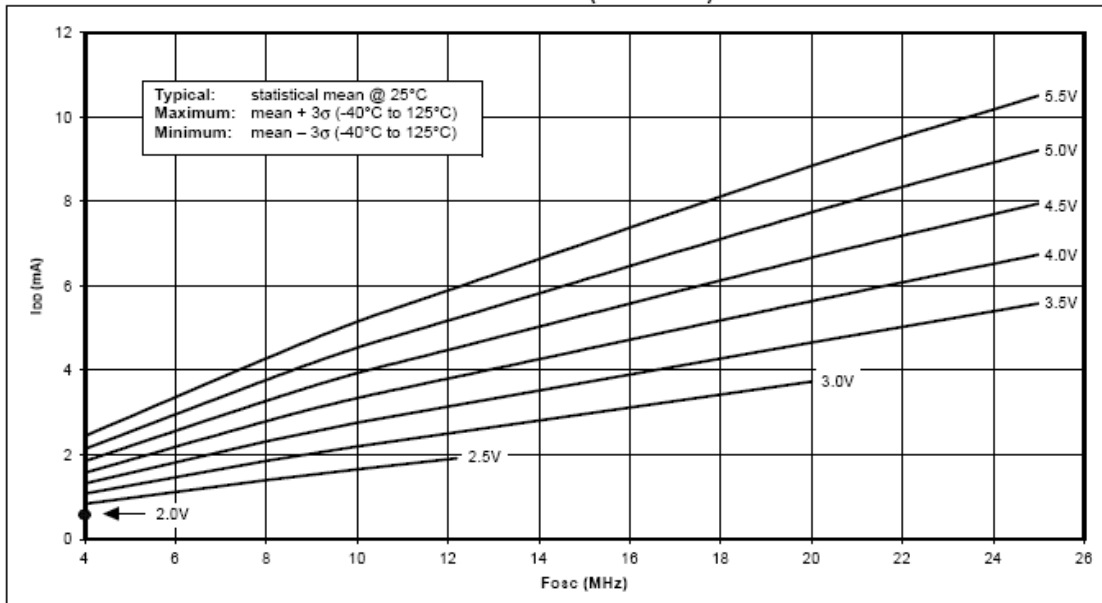
6. Draw a diagram that clearly shows how you monitor TOTAL current draw in your breadboard PIC18 system.

See the lab manual.

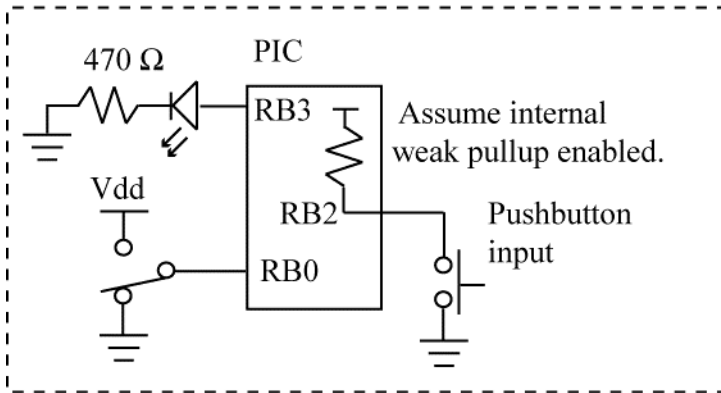
7. In the diagram below, it is clear that lowering the voltage and lowering the clock frequency reduces current draw. Why does the graphs for 2.5V, 3.0V not extend past 24MHz like the other curves? Explain.

With lower voltages, gates (transistors) cannot switch as fast, thus limiting the maximum obtainable clock frequency.

FIGURE 23-1: TYPICAL I_{DD} vs. F_{OSC} OVER V_{DD} (HS MODE)



Problem (a)



Problem (b)

