University of Puerto Rico
Mayaguez Campus
Electrical and Computer Engineering Department.
ICOM5217 - Microprocessor Interfacing

Experiment ## - Report

Low-power Modes, LED Display Techniques & Keypads

Juan Lebrón Betancourt
Anthony Llanos Velázquez
Profesor: Manuel Jimenez
Instructor: Luis Malavé
October 22, 2013

Exercise:

Low Power Mode Operation

•

Implementation Details

Tiva Low-Power Mode Current Consumption

The LCD Interface mounted on the first experiment was tested again but this time using the Tiva's low power mode feature. For this, the TIva's User Guide and Datasheet were followed, and using the method in the Tiva Libraries named CPUwfi(), the Tiva automatically enters in a low power state in which consumes less current and waits for an interrupt (wfi) to wake it up again. Hence the Tiva is only in non-low-power mode when running the designated interrupts. While in low-power mode, **current consumption in the Tiva is lowered *to* 0.031A or 31mA *from* 40mA** which is the current that is normally consumed.

The low-power mode feature is implemented in the exercise done for the fist experiment of the course. This experiment involved the task of providing an interface for a user in which they saw a list of items and could scroll through the list using two up and down buttons. In the original implementation, the microprocessor completed the setup of the ports and entered a polling state in which it waited for user input resulting in energy being wasted. For the implementation of this experiment, interrupts were configured such that the microprocessor could be put in a low power state after setting up all of the ports.

The microprocessor successfully entered low power mode after setting up the ports, and the current consumption reported was lower as specified above. When the microprocessor entered the low power state after finishing the interrupt, it continued execution at the main routine of the program and if there was no other instruction the program ended. Therefore, the program was written such that when the control returned from the interrupt to the main, the processor was in an infinite loop that called again the low power state successfully.

Below is the piece of code that correctly returned the processor to low-power state:

```
int main(void){

	//..
	//Setup instructions ommitted.
	//..

	//Call forever low power state when returning from interrupts.
	while(1){
		CPUwfi();
	}
}
```

The same interrupt and control structure from experiment 2 was used except that the interrupt routine from the sensors were changed to the buttons:

```
void switchPressed(){
      SysCtlDelay(300000);
      //Raw Interrupt Status.
      uint32_t ris = HWREG(0x40024414);
      //Pressed Right.
      if(ris & 0x02){
            //Move down in circular array.
            menuDown(cursor);
      }
      //Pressed Left.
      else if(ris & 0x04){
            //Move up in circular array.
            menuUp(cursor);
      }
      SysCtlDelay(300000);
      IntFinish();
      //On return to main, CPUwfi() is called.
}
```

Battery Exercise:
With its current consumption the Tiva would last the following time in normal and in low power mode, using a 1200 mAh battery with nominal voltage until 25%.

Normal
Current Consumption: 0.040A or 40mA
Battery: 1200 mAh

Time: (1200mAh*0.75)/40mA = **22.5 hours**

Low Power
Current Consumption: 0.031A or 31mA
Battery: 1200 mAh

Time: (1200mAh*0.75)/31mA = **29.03 hours**

The Tiva would last 29-22.5 = **6.5 hours** more in low power mode.

Exercise:

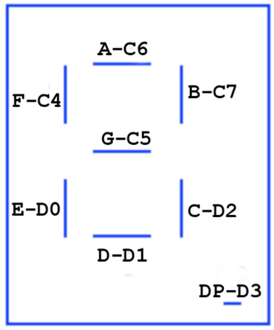
Using 7-segment Displays

•

Implementation Details

Understanding 7-segment displays.

To understand 7-segment displays, a table was completed that showed the hexadecimal result of showing the numbers 0-9 and the letters A through F in a display. The assignment of the letter segments is also show in the image below.



#	B-C7	A-C6	G-C5	F-C4	DP-C3	C-D2	D-E1	E-D0	7-seg
0	1	1	0	1	0	1	1	1	0xD7
1	1	0	0	0	0	1	0	0	0x84
2	1	1	1	0	0	0	1	1	0xE3
3	1	1	1	0	0	1	1	0	0xE6
4	1	0	1	1	0	1	0	0	0xB4
5	0	1	1	1	0	1	1	0	0x76
6	0	1	1	1	0	1	1	1	0x77
7	1	1	0	0	0	1	0	0	0xC4
8	1	1	1	1	0	1	1	1	0xF7
9	1	1	1	1	0	1	0	0	0xF4
A	1	1	1	1	0	1	0	1	0xF5
B	1	1	1	1	1	1	1	1	0xFF
C	0	1	0	1	0	0	1	1	0x53
D	1	1	0	1	1	1	1	1	0xDF
E	0	1	1	1	0	0	1	1	0x73
F	0	1	1	1	0	0	0	1	0x71

Exercise:

Dynamic Display Using Two 7-segment Displays

•

Implementation Details

Counter

A counter was implemented using the table above such that it displayed the numbers in an incrementing way. The implementation uses a lookup table in which the digits from the table in the previous exercise are entered. A counter in software was implemented with a timer such that the timer incremented the counter every second while updating the display. Below is the code of the timer that handles the display and the timer that handles the counter. The reason for this was that there were two displays connected and the value was show using both:

```
uint32_t values[] = { ~0xD7, ~0x84, ~0xE3, ~0xE6, ~0xB4, ~0x76, ~0x77, ~0xC4,
                      ~0xF7, ~0xF4, ~0xF5, ~0xFF, ~0x53, ~0xDF, ~0x73, ~0x71 };
//Counting Timer.
void timer0A(){
    //Clear the interrupt.
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
      //Circular loop.
      if(++counter1 > 15){
            counter1 = 0;
            counter2++;
      }
      //For counting with both digits.
      if(counter2 > 15){
            counter2 = 0;
            counter1 = 0;
      }
}
//Display timer.
void timer1A(){
    //Clear interrupt.
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
    //Toggle between both timers with their respective values.
    if(state) {
      gpioSetData(GPIO_PORTA,0x08);
      gpioSetData(GPIO_PORTD, values[counter1]&0x0F);
      gpioSetData(GPIO_PORTC, values[counter1]&0xF0);

    }
    else {
      gpioSetData(GPIO_PORTA,0x04);
      gpioSetData(GPIO_PORTD, values[counter2]&0x0F);
      gpioSetData(GPIO_PORTC, values[counter2]&0xF0);
    }
    state = !state;
}
```

The timer was setup to have an execution rate of 120Hz. This means that both displays are refreshed at a 60Hz frequency.

```
void initTimerModule(){
      //...Omitted
      TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/120);
      //...Omitted
}
```

Exercise:

Using a Keypad

•

Implementation Details

Reading from a Keypad

A keypad was used to enter numbers which were displayed on the 7-segment display from the exercise above. The values were stored such that it displayed the last two numbers entered automatically on the function that updated the display.

Reading from the keypad required us to know how the keypad was structured. The structure of the keypad consisted lines that covered rows and columns. to detect whether there was a button pressed, a scan has to be done by setting to high a row, and checking each column to detect which one returns high. Each one of the 4 rows had three columns/keys to read, either 1-2-3, 4-5-6, 7-8-9, or #-0-*. The function had a map that told which number was read.

```
//Read data from a port’s pin.
uint32_t keypadReadPortPins(uint32_t port, uint32_t pins){
      return gpioGetData(port) & pins;
}

//Read a selected row by checking the three columns
int keypadReadRow(uint32_t rport, uint32_t rpin, int d1, int d2, int d3){

       //Read the port’s pin or row.
       gpioSetData(rport,rpin);
       if(keypadReadPortPins(GPIO_PORTA,0xE0)){
            //Read column 1 by checking input.
             if(keypadReadPortPins(GPIO_PORTA, 0x80)){
                  msb=lsb;
                  lsb=d1 ;
                  gpioSetData(rport,0x00);
                  return 1;
             }
             //Read Column 2
             else if(keypadReadPortPins(GPIO_PORTA,0x40)){
                   msb=lsb; //Least significant becomes most significant.
                   lsb=d2; //New number is least significiant.
                   gpioSetData(rport,0x00);
                   return 1 ;
             }
             //Read Column 3
             else if(keypadReadPortPins(GPIO_PORTA,0x20)){
                   msb=lsb;
                   lsb=d3;
                   gpioSetData(rport,0x00);
                   return 1;
             }

       }
       //End read to allow next read.
       gpioSetData(rport,0x00);
       return 0;
}
```

```c
//TImer that continously read the keypad. Contains the map to the keys.
void timer0A(){
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    //PB1 PE4 PE5 PB4 PA5 PA6 PA7
    //SysCtlDelay(200000);

    //Mapping for the keypads is done here.
    if(!keypadReadRow(GPIO_PORTB, 0x02,3,2,1))
    if(!keypadReadRow(GPIO_PORTE, 0x10,6,5,4))
    if(!keypadReadRow(GPIO_PORTE, 0x20,9,8,7))
    keypadReadRow(GPIO_PORTB, 0x10,11,0,10);
    TimerEnable(TIMER0_BASE,TIMER_A);
}
```

Exercise:

Homework - Calculator

•

Implementation Details

Implementation

A calculator was implemented using the components developed in the previous exercises. The calculator allowed for the input of numbers from which you could see in the display. Once the star or pound signs were pressed, the current value in the displays was stored and the used could enter another number. Should the user press an additional implemented button for the system, they could clear the number (setting it to 0, or compute the operation that was entered before with the current number displayed, and the stored number, displaying the result on the displays. This logic was implemented in the programming of the system using the same logic of the previous exercise but changing the interpretation of the pound and star signs and adding the two push buttons.

```
void keypadTimer0A(){
    //Clear interrupt flag.
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    //Read the rows.
    int operation=-1;
    if(!keypadReadRow(GPIO_PORTB, 0x02,3,2,1))
    if(!keypadReadRow(GPIO_PORTE, 0x10,6,5,4))
    if(!keypadReadRow(GPIO_PORTE, 0x20,9,8,7))
      operation = keypadReadRow(GPIO_PORTB, 0x10,11,0,10);

    //If the operation is either add or substract, store current
    //number to allow input of the nex one.
    if(operation == 10){
      op = 11 ;
      stack = msb*10+lsb;
    }
    else if(operation == 11){
      op = 10;
      stack = msb*10+lsb;
    }

    TimerEnable(TIMER0_BASE,TIMER_A);
}

//Handles the input of the push buttons.
void keypadControlHandler(){
      SysCtlDelay(200000);
      //If clear button, clear both digits.
      if((gpioGetInterruptRawStatus(GPIO_PORTE)&0x04)){
            msb=0;
            lsb=0;
      }
      //If enter button, perform operation:
      else if(gpioGetInterruptRawStatus(GPIO_PORTE)&0x02){
            int result ;
            //Substract operation.
            if(op == 10){
                  result = stack - (msb*10+lsb);
                  if(result<0) result *= -1;
                  msb = result / 10;
                  lsb = result % 10;
                  if(result>99){
```

```
                    msb = 0;
                    lsb = 15;
                }
            }
            //Add operation.
            else if(op == 11){
                result = stack + (msb*10+lsb);

                msb = result / 10;
                lsb = result % 10;
                if(result>99){
                    msb = 0;
                    lsb = 15;
                }
            }
        }
        SysCtlDelay(200000);
        gpioSetInterruptClear(GPIO_PORTE,0x06);
}
```