Lab 4 report – Reading the temperature

Student name: Vu Nguyen (71141)

Date: 17.03.2015

Assumptions on knowledge of the reader

It is assumed that a reader should know some basic programming skills in C/C++ in order to understand the source code. A basic knowledge of hardware such as processors, LEDs, pin ports, interrupt, cooperative scheduler, ADC converter, and hardware timers should be known.

Description of the task

The task of lab 4 is to implement a cooperative scheduler for an embedded system. A built-in ADC converter will be used to convert the ambient temperature using a negative temperature coefficient (NTC) resistor. The code in this lab will implemented based on lab 3. In this lab, an interrupt driven cooperative embedded scheduler will be implemented so that it can scheduler tasks for the application. The tasks in lab 4 consist of three tasks: read values from the built-in ADC converter, make the led on the expansion board (daughterboard) blink with a frequency correlating to the ADC value read, check the status of the button on the expansion board to enable/disable the blinking by pressing a button (e.g, pressing the button will disable led blinking).

Description of the equipment used

The equipment used consists of a Modtronix SBC65EC embedded platform with built in web server and bootloader. A PC with Microchip MPLAB IDE/MCC18 compiler is installed to write and compile the C source code. Besides, Modtronix Network Bootloader must be installed in the development PC to program the executable to the PIC microcontroller of the platform. An Ethernet cable is used to connect the platform to the local area network. A regular 2.1mm jack plug for power supply (7-35VDC) is used to power up the platform. The blue button on the expansion board will be used to enable/disable the led blinking. This button is configured as pull-up. Since the NTC resistor on the expansion board is difficult to use for testing purpose, an alternative is to use a 10K potentiometer on the expansion board and have the built-in ADC converter read the analog voltages from this potentiometer.

Description of the performed work

Two components are reused from lab 3, PES_leds and PES_ports. The PES_leds component is used to control the blinking of the LEDs on the expansion board. The PES_ports component is used to control the port pins of the microcontroller, for example, reading pin status and set a port pin high/low. Three new components implemented in this lab are tcc_scheduler (.c and .h), mytask (.c and .h), and PES_adc (.c and .h). The tcc_scheduler component implements the embedded cooperative scheduler. The cooperative scheduler is based on the work of Pete Vidler [1]. However, it is modified a bit to suit the requirements of the lab. The mytask component is used to define three tasks required: reading ADC

values, checking the status of the blue button, and blinking the led on the expansion board. Finally, the PES_adc component is responsible for reading analog values from the potentiometer on the expansion board and returns the converted values.

Implementation of cooperative scheduler

The cooperative scheduler consists of a scheduler data structure and five functions for initializing the scheduler, adding tasks to the scheduler, dispatching the tasks and entering power saving mode as shown below

```
//Scheduler data structure
typedef struct {
  void  (*ptask)(void); // Pointer to the task function.
  uint32_t period;      // Period to execute with.
  uint32_t delay;       // Delay before first call.
} task_type_t;
```

Listing 1. Cooperative scheduler data structure

```
void SCH_Init(uint32_t tick_interval_ms);
void SCH_Start(void);
void SCH_Create_Task(void (*function_pointer)(void),
                     uint32_t period,
                     uint32_t delay);

void SCH_Dispatch_Tasks(void);
void SCH_Enter_Idle_Mode(void);
```

Listing 2. Functions offered by the cooperative scheduler

Function SCH_Init sets up interrupt interval for timer 0. The argument passed into the function is *tick_interval_ms* which indicates the interrupt interval of time 0 in ms. After setting up timer0 function SCH_Start should be used to enable time 0. Function SCH_Create_Task adds tasks to an array of tasks which is managed the scheduler. Function SCH_Dispatch_Tasks checks if a certain task managed by the scheduler is due to run. If any, it will execute that task. When the scheduler has nothing to do, it should enter power saving mode. This is achieved by calling function SCH_Enter_Idle_Mode.

Implementation of the ADC conversion function

The ADC conversion component is responsible for initializing the built-in ADC converter, converting the analog values and returning the converted values in digital format. It consists of two functions PES_ADC_init and PES_ADC_read. Since the potentiometer on the expansion board is connected to port pin A2 of the microcontroller, function PES_ADC_init configures this pin as input and selects AN2 as input channel. Furthermore, it enables the built-in ADC module and chooses clock, format, and acquisition time. Another function is the PES_ADC_read. This function starts ADC conversion process, waits until the conversion is done, and returns the converted value. Implementation of this component can be seen as follows:

```
void PES_ADC_init(void) {
  TRISAbits.TRISA2 = 1;          //Configure AN2 as input port

  ADCON0bits.CHS = 0b0010;       //Select AN2
  ADCON2bits.ACQT = 0b000;
  ADCON2bits.ADCS = 0b111;
  ADCON2bits.ADFM = 1;           //Right justified
  ADCON0bits.ADON = 1;           //Enable the ADC module
}

uint16_t PES_ADC_read(void) {
  uint16_t temperature;

  //Start ADC conversion
  ADCON0bits.GO_DONE = 1;

  //Wait until ADC conversion is done
  while(ADCON0bits.GO_DONE != 0);

  temperature = (((uint16_t)ADRESH)<<8)|(ADRESL);

  return temperature;
}
```

Listing 3. Implementation of ADC conversion component

Implementation of the task component

The task component mytask (.c and .h) implements three tasks for the application: task_adc, task_led, and task_button. Task task_adc reads the values from the ADC converter by calling PES_ADC_read. The value read is then mapped to a range of 20 to 200 ticks. The mapped value in turn is assigned to the blinking period of the LED on the expansion board. The task task_button checks the status of the blue button on the expansion board. If the button is pressed, it set the *button_blue_state* flag. Otherwise, it clears the flag. The task task_led is pretty simple. If the *button_blue_state* flag is set, it makes a call to blink_led function which is offered by PES_leds component to blinking the red LED. Implementation of this component can be seen in Listing 4.

```
#include "main.h"
#include "mytask.h"
#include "PES_ports.h"
#include "PES_leds.h"
#include "PES_adc.h"

//Buttons on the expansion board
#define BUTTON_BLUE_PRESSED 0
#define BUTTON_BLUE_UNPRESSED 1
#define BUTTON_BLUE PB1

//Global variable
static uint8_t button_blue_state = BUTTON_BLUE_UNPRESSED;
```

```
uint32_t led_period = 0;

void task_adc(void) {
  uint16_t temperature;

  temperature = PES_ADC_read();

  //Map adc reading from 10-bit ADC converter to the
  //range of 20 - 200
  //period = (adc reading * 200 / 1024) + offset
  led_period = (uint32_t)(temperature * (uint32_t)200 / (uint32_t)1024) + 20;
}

void task_button(void) {
  if (readPin(BUTTON_BLUE) == BUTTON_BLUE_PRESSED) {
    button_blue_state = BUTTON_BLUE_PRESSED;
  } else {
    button_blue_state = BUTTON_BLUE_UNPRESSED;
  }
}

void task_led(void) {
  if (button_blue_state == BUTTON_BLUE_UNPRESSED) {
    blink_led(LED_RED);
  }
}
```

Listing 4. Implementation of mytask component

Implementation of the application

The application in this lab is pretty straightforward. It initializes the system at the beginning. Then it initializes the cooperative scheduler and the tasks. Next, it adds all the tasks to the scheduler and starts the scheduler. Once the initialization process is done, it enter the infinite loop where to repeatedly calls SCH_Dispatch_Tasks and put the microcontroller into idle mode if the scheduler has no tasks to be done. Tasks are scheduled in such a way that ADC reading will get the highest priority. The next highest priority is the button checking task. The lowest priority task is the LED blinking task. The following listing shows the implementation of the application.

```
#include "projdefs.h"
#include "p18f6627.h"
#include "PES_ports.h"
#include "ttc_scheduler.h"
#include "mytask.h"
#include "PES_ports.h"
#include "PES_adc.h"

static void System_Init(void);

//Main application
void main(void) {
  //Initialize the system
```

```
  System_Init();

  //Initialize cooperative scheduler with tick interval of 5ms
  SCH_Init(5);

  //Initialize tasks
  PES_ADC_init();

  //Create tasks with name, period, and initial delay in ticks (NOT ms)
  SCH_Create_Task(task_adc, 10, 0);
  SCH_Create_Task(task_button, 20, 5);
  SCH_Create_Task(task_led, 40, 10);


  //Start cooperative scheduler
  SCH_Start();

  while(1) {
    SCH_Enter_Idle_Mode();
    SCH_Dispatch_Tasks();
  }
}

static void System_Init(void) {
  //Disable watchdog timer
  WDTCONbits.SWDTEN = 0;
}
```

Listing 5. Implementation of the application

Achieved results

As mentioned in the Description of the task section, a potentiometer is used instead of the NTC resistor for easy testing purposed. When the knob of the potentiometer is rotated counter clockwise so that the values read get smallest, the led blinking frequency is slow. Rotating the potentiometer clockwise gradually will increase the blinking frequency of the red LED on the expansion board. The red LED on the expansion board blinks when the blue button is not pressed. When the blue button is pressed, the red LED stops blinking immediately.

References

[1] Pete Vidler. Simple Co-Operative Scheduling [online]; Pete's webpage, 16 March 2011. http://petevidler.com/2011/03/simple-co-operative-scheduling/ Accessed 17 March 2015.