Programming Embedded Systems 2018 / JB

Exercise 2 / 17/24.9.2018 / Deadline for submitting report 8.10.2018

Return report electronically on address: https://abacus.abo.fi/ro.nsf. If you do not have an ÅA account, please email jerker.bjorkqvist@abo.fi

Advisor: Jerker Björkqvist, Agora

Equipment and tools

Equipment used:

a) Texas Instruments LaunchPad **MSP430G2** development card
b) Own laptop

Task

Using the requirements from ex1, rewrite the program to support basic Embedded Operating System (EOS), by

a) Rewriting software into tasks (and init-functions) , writing O-O embedded C-code
b) Creating variable delays (in ms) by using programmable timers and interrupts
c) Basic reading for switches

Details

Tasks. Create tasks (~=functions) for

d) blinking the led
e) reading input from a input pin (led stops blinking when button pressed)
…
f) creating a functionality for variable delay (in ms), using timers and interrupts

O-O embedded C-code means that software modules should be clearly divided into separate files / and functions. Settings specific for microcontroller / topic area should be separated in include (.h) files.

Programmable timers on the MSP430. MSP430 has five types of timer modules; Basic Timer, RTC, Watchdog Timer, Timer_A, Timer_B. In this exercise, we use Timer_A.

Timer_A is a 16 bit timer, and can be configured in different ways. It can count up / down, use different clock sources, use different dividers. Here are the following important registers:

TACTL (alt. TA0CTL)

TASSEL_X → Set clock source (ACLK, **SMCLK,** TACLK, INCL)
ID_X → Divider from clock source (1/2/4/8)
MC_X → counting mode

CCTL0 (alt. TACCTL0) – Capture /compare control 0

CCIE → enable interrupts

CCR0 (TACCR0) – Capture / Compare control register
Put here the value to count to

By configuring the timer (TACTL), configuring interrupts (CCTL0) and setting the correct capture / compare value (CCR0) we can generate a interrupt with predefined intervals.

In order run some code when the interrupt occurs, we have to tell the compiler to place some code at a specific place and that it is an interrupt. This varies with different compilers, but in TI GCC:

```
__attribute__((interrupt(TIMER0_A0_VECTOR))) void Timer_A(void) {
  timer_run();
}
```

The code to be run is now timer_run()

In order to enable interrupts in the system, you also have to have a statement

```
__enable_interrupt();
```

Registers for I/O:

PxDIR – register for controlling I/O-port direction

P1DIR |= 0b00100000; // pin 6 on port 1 is set as output

PxOUT – register: latch for output port

P1OUT |= 0b00100000; // pin 6 on port 1 is set high

PxIN – value on port (for reading input)

myval = P1IN & 0b00100000; //Value on Port 1 pin 6

PxREN – enable pullup / pulldown

P1REN |= 0b00100000; // enable pullup/pulldown,
pullup/pulldown dependent on P1OUT

Note also that your “ports.h” file can (should) be written like this

#define RED_LED PORT P1OUT
#define RED_LED_PIN (0b00100000)

Document what you have done, and submit the documentation and the code you have produced electronically to the address give above.

General rules for documenting projects:

Each report should include:

- Title
- Name
- Date / timeframe when exercise performed
- Group (if not done individually)
- Assumptions on knowledge of the reader
- Own contribution (if performed in group)
- Description of the task / exercise
- Description of the equipment used
- Description of performed work
- Achieved results