Lab 2:_IO Handling

Revised by : Muhammad Imran (Developed by: Muhammad Amir Yousaf)

Date : 2014-10-24

Course : ET032G, Elektroteknik GR (A), Mikrodatorteknik at Mittuniversitetet, Sweden.

IO handling is important in microcomputer based system design in order to establish communication with external devices. In XMEGA, several kinds of IO handling techniques are offered. It has interrupt enabled IO mapped ports, 'DMA controller' to handle communication through IO peripherals without the intervention of CPU and Event System.

In this lab, you will use two techniques in order to get familiarity with IO handling. These techniques include

- Polling: where CPU have to closely monitor event on an IO port.
- Interrupt: CPU may sleep or keep doing other things; the event will interrupt the normal flow of program.

Task 1:

In this task, you will write a program that uses a CPU polling technique in which CPU is continuously monitoring an event on IO (a touch button). Identify which port share the touch buttons. When a button (for example CS0) is pressed or timer count (for example timer type 0) value is reached to the specified limit, the time (number of seconds) showing on the LCD is reset and counting is restarted.

Modify the given program for the stated functionality. The program has LCD related functions but you need to integrate the drivers and components in ASF wizard.

- LCD drivers
- YMCC42048AAAYDCL LCD components

If you are interested in more details related to LCD, please access this page
http://eewiki.net/display/microcontroller/Basic+LCD+Example+for+Xmega-B1
Note! 16 bit timer type 0 and touch button 0 are used in the program

```
#include <compiler.h>
#include <asf.h>
int main (void) {

               board_init();

               // LCD setting///
               CLK.RTCCTRL=0x01;
               c42048a_init();
               #ifndef CONF_BOARD_LCD_BACKLIGHT_PWM
               backlight_on();
               #endif
               lcd_set_contrast(30);
               // Timer 0 setting //
               //Lab task
               TCC0.PER=
               TCC0.CTRLA =

               // Touch button setting
               //Lab task
               PORTE.DIR =
               //initialization//
               const uint8_t message[] = "LAB_02";
               c42048a_write_alpha_packet(message);
               c42048a_set_numeric_dec(0);
               uint16_t counter = 0;

               // Actual coding section for the lab
               while (true) {

                              //Lab tasks
                              //c42048a_set_numeric_dec(seconds);  // seconds is user defined variable
                              TCC0.INTFLAGS = 0x01;               // clear the interrupt flag
                              }

                // condition for checking the touch button with given function
               }
           }
```

Write in report the scenarios when polling technique can be used.

TIPS
These functions/statements will help you in the lab
is_touch0();
Use a counter to show the number of seconds and this counter should either be reset when a button is pressed or when a counter values reach to its limit.

Task 2:

"An interrupt is a signal to the processor by a peripheral indicating an event or a change of state that needs immediate attention. The processor responds by suspending its current activities, saving its state, and executing a small program called an interrupt handler (or interrupt service routine, ISR) to deal with the event. The processor resumes execution after finishing the interrupt handler."

Peripherals can have one or more interrupts in XMEGA controller, and all are individually enabled and configured. It will generate an interrupt request when there will be an interrupt condition. The programmable multilevel interrupt controller (PMIC) controls the handling and prioritizing of interrupt requests. When an interrupt request is acknowledged by the PMIC, the program counter is set to point to the interrupt vector, and the interrupt handler can be executed. The advantage of having such a feature allows the CPU doing normal operation while there is no high priority events. Each IO pin in XMEGA controller implements the interrupt feature and is able to alert CPU to level change at input pin.

Note! There are useful hints in the Appendix for this task

- In this task you will use interrupt handling feature in order to achieve the stated functionality of task 1. Modify the program in Task 1 such that CPU polling is replaced with the interrupt handling.
- When the Xmega board is initialized, a message "TASK_02" is displayed on LCD and LED0 is switched on. Values of time (number of seconds) are not displayed.
- When a button (CS0) is pressed, a message "INT_RCD" is displayed and LED0 is switched off and LED1 is switched on. Values of time (number of seconds) are displayed.

Write in report the scenarios when interrupt based technique can be used.

Appendix A

Configuring and Using Port Interrupts I/O port interrupts can be used to generate interrupt on pin change or pin level, and for waking the device from sleep modes. This section gives an overview of the I/O port interrupt system and how it is used.

Configuration of Port Interrupts and Interrupt service routine

Each I/O port on the XMEGA has two interrupts. It is possible to map each interrupt to be triggered by an arbitrary combination of the pins in the I/O port.

Setting up the pin interrupts is done in the first three steps (For task01 and task02).

i. Configure the input/sense part of the PINnCTRL register for each pin that can trigger the interrupt.
ii. Write the bit mask corresponding to the pins that can trigger the interrupt to the INT0MASK register.
iii. Select the interrupt priority level for the port by medium by setting the INTCTRL – Interrupt Control register of the port E.

Interrupt Level Configuration	Group Configuration	Description
00	OFF	Interrupt disabled.
01	LO	Low-level interrupt
10	MED	Medium-level interrupt
11	HI	High-level interrupt

Define interrupt service routine (ISR) (For task02)

Define an ISR out of the main function and specify the corresponding Interrupt vector in Interrupt service routine. For this lab in task02, PORTE interrupt vector 0 can be used. The interrupt vectors are a list of hardwired addresses in program memory which are executed after the interrupt is triggered. After finishing the tasks, the CPU resume operations from where interrupt was called.

For ISR to be working, you need to set relevant interrupt control register for the peripherals you are using (steps i-iii), enable the global interrupt flag (step iv-v) and the condition for the interrupt (for example pressing a touch button).

iv. For enabling global interrupt, the global interrupt enable (I) bit is set in the CPU status register (SREG). You can use the already available functions like SEI(), enable_global_interrupt() for this.
v. Enable medium level interrupts in the PMIC by writing in its control (CTRL) register.

http://www.atmel.com/Images/doc8312.pdf
http://www.atmel.com/Images/doc8043.pdf
http://www.atmel.com/Images/doc8401.pdf
http://www.avrfreaks.net/forum/tut-newbies-guide-avr-interrupts