Lab 1: Accessing Configuration Registers

Developed by : Muhammad Amir Yousaf

Revised by : Muhammad Imran

Date : 2014-10-02

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

Goal:

The goal of this lab is to have an in depth understanding of the architecture of modern microcontrollers and validation of concepts with the help of simulators. XMEGA family of microcontrollers is selected for this purpose. The microcontroller XMEGA B1 128 has various flexible and configurable peripherals to interface with external electronic devices. You will use C language for writing a program to access its configuration registers and to implement simple computational tasks. The exercises motivate to consult the datasheet, manual and tutorials provided at the manufacturer web page in order to get insight of device.

Platform used:

Development Kit: XMEGA Xplained B1

Programming environment: AVR Studio 6.2

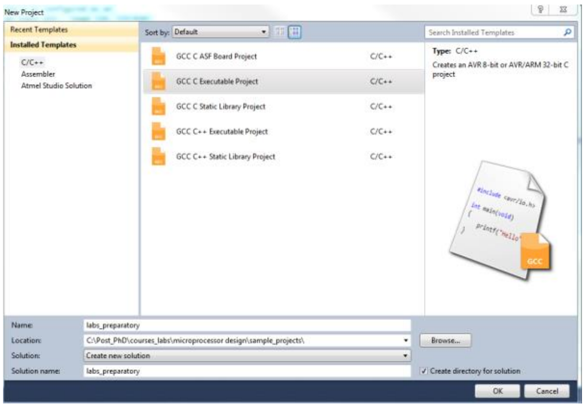
Preparation for Exercises:

Various text and video tutorials are available in order to get basic understanding of tools required for performing these lab exercises. The link for download is given here

http://www.atmel.com/devices/ATXMEGA128B1.aspx?tab=documents

In preparatory, you will run a simple program to understand the basics of AtmelStudio and XMEGA Xplained B1. Follow the steps to run a preparatory task

1. Start AtmelStudio 6.2, go to File in menu bar, click on *New Project* to create a new project with name *labs_preparatory*
2. Select *GCC C Executable Project* (for tasks without ASF) or *GCC C ASF Board Project* (for tasks with ASF). You would have window as shown in following figure.



GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++

Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

```
#include <avr/io.h>
int main(void)
{
printf("Hello"
}
```

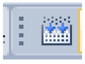
Name: labs_preparatory

Location: C:\Post_PhD\courses_labs\microprocessor design\sample_projects\

Solution: Create new solution

Solution name: labs_preparatory

☑ Create directory for solution

3. Select Device Family in drop menu as *AVR XMEGA, 8-bit* and choose *ATxmega128B1*
4. Paste the provided C code in the file *labs_preparatory.c* and click build (F7)  and

   you would see the build results in *Output window* which shows the status and location of output file. You can find errors and warning in this window.

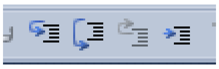
   Code: read the comments with code

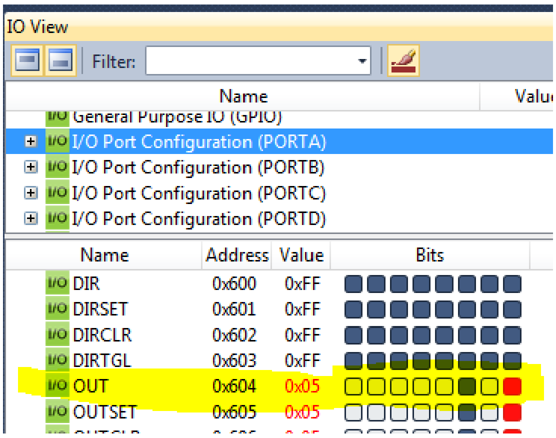
```
unsigned char* const PORTA_DIR_REG= (unsigned char*)0x600;
unsigned char* const PORTA_OUT_REG= (unsigned char*)0x604;

int main()
{
       *PORTA_DIR_REG = 0xFF;
       for (unsigned char counter = 0; ;)
         {
          *PORTA_OUT_REG = counter++;
             if(counter >= 10)
               counter = 0;
           }
}
```

   Comments for the above code:

   PORTA *direction* and *out* registers' addresses are mapped onto PORTA_DIR_REG and PORTA_OUT_REG respectively. PORTA_DIR_REG has been assigned 1s (0xFF) in order to make all pins as output. For more details, read the following text from manual. (see page 124, 133/410 in XMEGA B MANUAL)

   "Each port has one data direction (DIR) register and one data output value (OUT) register that are used for port pin control. The data input value (IN) register is used for reading the port pins. Direction of the pin is decided by the DIRn bit in the DIR register. If DIRn is written to one, pin n is configured as an output pin. If DIRn is written to zero, pin n is configured as an input pin".
5. Right click on the project folder in the *solution explorer, go to toolchain* tab and set complier optimization level –O0.
6. In the *Tool* tab, select debugger as Simulator and save the project.
7. Now press the *Start debugging and break* Debug and run the program in different steps 
8. You will see the counter values in the *OUT – Data Output Value register* having address 0x604. Familiarize yourself with code, datasheet and manual in order to use it in Lab tasks. General Purpose IO ports can be accessed and configured through the registers mapped in IO memory (see Appendix). You can enable the IO view by clicking on this button



IO View

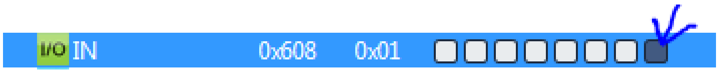
Filter:

Name	Valu
General Purpose IO (GPIO)	
⊞ I/O Port Configuration (PORTA)	
⊞ I/O Port Configuration (PORTB)	
⊞ I/O Port Configuration (PORTC)	
⊞ I/O Port Configuration (PORTD)	

Name	Address	Value	Bits
DIR	0x600	0xFF	■■■■■■■■
DIRSET	0x601	0xFF	■■■■■■■■
DIRCLR	0x602	0xFF	■■■■■■■■
DIRTGL	0x603	0xFF	■■■■■■■■
OUT	0x604	0x05	□□□□□■□■
OUTSET	0x605	0x05	□□□□□■□■
OUTCLR	0x606	0x05	□□□□□■□■

Task 1:

In this task, you will use AVR Studio 6 Editor and AVR Simulator. Modify the preparatory program such that it takes input from PIN0 of PORT B and then increment the counter values which are assigned to PORT A as output. This program needs to be written without AVR Software Framework (asf.h). Take snapshots for report. You can give inputs to pins by clicking on the square space as shown in the figure.



Task 2:

XMEGA has three 16 bit timer/counters that can be configured to produce time delays. Use a Real time counter to generate one second delay and modify the counter to count the time's ticks. You may use AVR Software Framework (ASF) to make the program but you should have an understanding.

Is it possible to verify one second delay with AVR Simulator in AVR Studio 6? Motivate?A video tutorial of ASF is given here: [Atmel Software Framework, Getting Started]

Switch the Optimization to level O0 and O1 and note the difference.

Tasks3:

In this task and next coming tasks, you can use the ASF. The example code in preparatory can be translated into following lines of code

```
#include <asf.h>
int main(void)
{
    char counter=0;
    PORTA.DIR=0xFF;
    while(1)
        {
        PORTA.OUT=counter++;
        if(counter >= 10)
          counter = 0;
        }
}
```

Identify the difference and write in report. Modify the above code to take input from a PIN0 of PORT B and reset the counter when a positive edge is detected. Assign the counter values to PORT C.

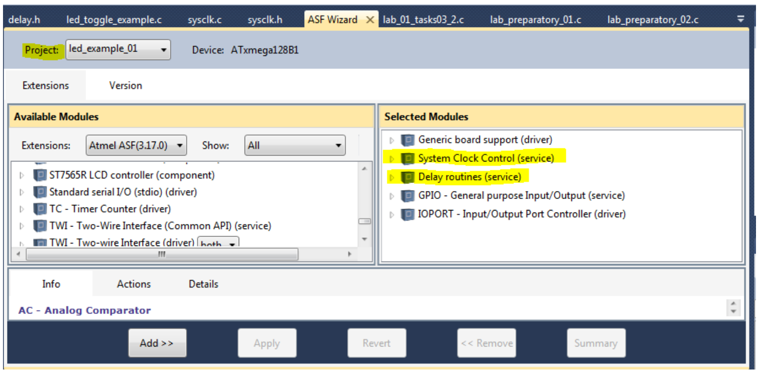
TIPS: For sensing rising edge you will need to use PINnCTRL – Pin n Configuration Register, INT1MASK – Interrupt 1 Mask register, and INTFLAGS – Interrupt Flag register. (see page 136-138 in XMEGA B MANUAL)

Task 4:

This task involves a real microcontroller instead of software simulator. XMEGA B1 Xplained board has four LEDs i.e. LED0 to LED3. Display some flash pattern over them. You can program more than one pattern replacing each other after regular interval time (using timer/counter). See appendix to know about the programming of the device.

TIPS:

- LEDS are connected to PINS of PORTB and you can find the information in Table 4-5. J3 I/O expansion header of user guide “Atmel AVR1912: Atmel XMEGA-B1 Xplained Hardware User Guide”. The LEDS and other peripherals can be found in the header file *xmega_b1_xplained.h.*
- If you are using delay functions such as *delay_ms* from *delay.h* , add the relevant modules in ASF wizard as shown in following figure. Make sure that you have selected the right project and add the dependent modules.



Project: led_example_01

Device: ATxmega128B1

Extensions: Atmel ASF(3.17.0)

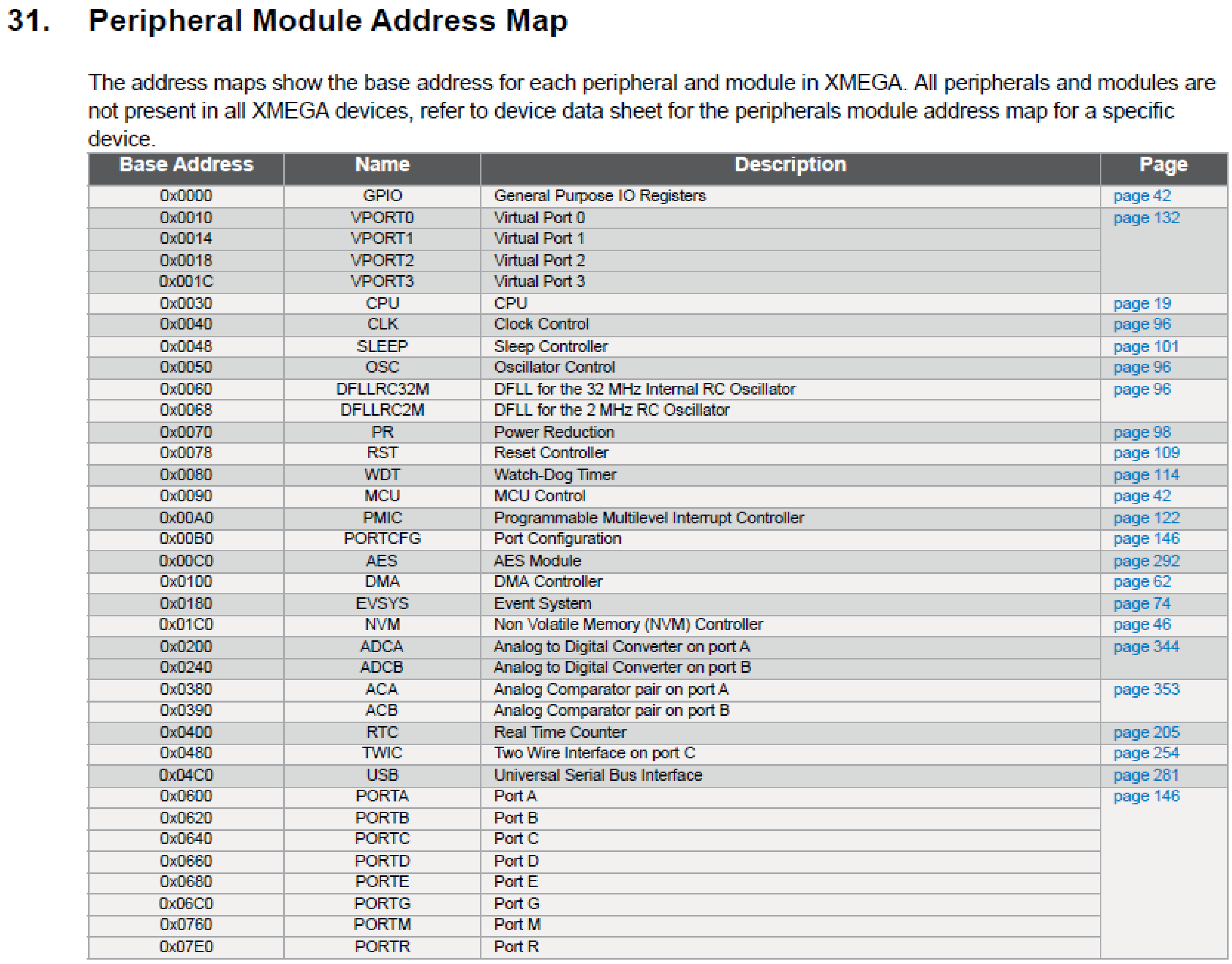
Show: All

ST7565R LCD controller (component)
Standard serial I/O (stdio) (driver)
TC - Timer Counter (driver)
TWI - Two-Wire Interface (Common API) (service)
TWI - Two-wire Interface (driver) both

Generic board support (driver)
System Clock Control (service)
Delay routines (service)
GPIO - General purpose Input/Output (service)
IOPORT - Input/Output Port Controller (driver)

AC - Analog Comparator

Appendix



31. Peripheral Module Address Map

The address maps show the base address for each peripheral and module in XMEGA. All peripherals and modules are not present in all XMEGA devices, refer to device data sheet for the peripherals module address map for a specific device.

Base Address	Name	Description	Page
0x0000	GPIO	General Purpose IO Registers	page 42
0x0010	VPORT0	Virtual Port 0	page 132
0x0014	VPORT1	Virtual Port 1	
0x0018	VPORT2	Virtual Port 2	
0x001C	VPORT3	Virtual Port 3	
0x0030	CPU	CPU	page 19
0x0040	CLK	Clock Control	page 96
0x0048	SLEEP	Sleep Controller	page 101
0x0050	OSC	Oscillator Control	page 96
0x0060	DFLLRC32M	DFLL for the 32 MHz Internal RC Oscillator	page 96
0x0068	DFLLRC2M	DFLL for the 2 MHz RC Oscillator	
0x0070	PR	Power Reduction	page 98
0x0078	RST	Reset Controller	page 109
0x0080	WDT	Watch-Dog Timer	page 114
0x0090	MCU	MCU Control	page 42
0x00A0	PMIC	Programmable Multilevel Interrupt Controller	page 122
0x00B0	PORTCFG	Port Configuration	page 146
0x00C0	AES	AES Module	page 292
0x0100	DMA	DMA Controller	page 62
0x0180	EVSYS	Event System	page 74
0x01C0	NVM	Non Volatile Memory (NVM) Controller	page 46
0x0200	ADCA	Analog to Digital Converter on port A	page 344
0x0240	ADCB	Analog to Digital Converter on port B	
0x0380	ACA	Analog Comparator pair on port A	page 353
0x0390	ACB	Analog Comparator pair on port B	
0x0400	RTC	Real Time Counter	page 205
0x0480	TWIC	Two Wire Interface on port C	page 254
0x04C0	USB	Universal Serial Bus Interface	page 281
0x0600	PORTA	Port A	page 146
0x0620	PORTB	Port B	
0x0640	PORTC	Port C	
0x0660	PORTD	Port D	
0x0680	PORTE	Port E	
0x06C0	PORTG	Port G	
0x0760	PORTM	Port M	
0x07E0	PORTR	Port R	

Programming the device

You can program the device with your application in two ways

- 1. Programming via Atmel AVR Studio 5 and Atmel AVR tools.
- 2. Programming via the DFU boot loader.

But in the labs, you will program the device visa DFU boot loader. For this purpose DFU programming software **FLIP** has already been installed in the lab computer. Therefore, go to Start-> FLIP 3.4.7 and open it. Select the device according to instruction in the XMEGA-B1 getting started Guide (section 5.2.2). It is important that FLIP uses USB interface to program the ATxmega128B1 but the boot loader has to be evoked by shorting **pin 6 on J1 to GND** if it hasn't been done already**.**