Lab AVR32 Intro

Aim
Getting started with AVR32: the IDE and the microcontroller

Literature
AVR32 manuals, tutorials (getting started with HW and SW, creating your own program, debugging)

Brief summary
AVR32 intro, GPIO using registers and drivers. Peripheral units; PWM, A/D.

Reporting
Demonstrate all programs for the teaching assistants. For this lab, no written report is necessary (an exception from the general rule that all labs should be reported in writing).

0. Introduction
The EVK1100 is equipped with three push buttons and 6 LEDs. The buttons and LEDs are connected to the following ports:

Item	**Pin**	**GPIO pin**	**GPIO port/pin**
Push button PB0	PX16	GPIO 88	2/24
Push button PB1	PX19	GPIO 85	2/21
Push button PB2	PX22	GPIO 82	2/18
LED0 (PCB label LED1)	PB27	GPIO 59	1/27
LED1 (PCB label LED2)	PB28	GPIO 60	1/28
LED2 (PCB label LED3)	PB29	GPIO 61	1/29
LED3 (PCB label LED4)	PB30	GPIO 62	1/30
LED4/5 (PCB label LED5)	PB19/PB20	GPIO 51/52	1/19 / 1/20

	(Bicolored LED, Red/Green)		
LED6/7 (PCB label LED6)	PB21/PB22 (Bicolored LED, Red/Green)	GPIO 53/54	1/21 / 1/22

Refer to the AVR32-manual for how to translate between pin number, GPIO pin number and GPIO port/pin numbers, or see the below.

GPIO port = floor((GPIO number) / 32), example: floor((36)/32) = 1
GPIO pin = GPIO number mod 32, example: 36 mod 32 = 4

1. GPIO with registers

The following code lights one LED.

```
int main(void){
AVR32_GPIO.port[1].gpers = 1 <<27; //enable GPIO control
AVR32_GPIO.port[1].oders = 1 <<27; //enable output driver
AVR32_GPIO.port[1].ovrc = 1 <<27; //clear value of pin
while(1){
}
}
```

The following lines can be used to read pins, i.e. the push buttons

```
AVR32_GPIO.port[2].gpers = 1 <<24; //enable GPIO control
int i=(AVR32_GPIO.port[2].pvr >> 24) & 0x01; //read port value
register
```

Exercises

1.1 Turn on all LEDs.

1.2 Turn on one LED when a push button is pressed

1.3 Make one bicolored LED change color every time a push button is being pushed.

1.4 Make one bicolored LED change color every time a push button is pressed (and released)

More exercises

1.5 Read all push buttons at once, with one line of code

1.6 Control all LEDs with one command, one line of code

2. GPIO with low-level drivers

Use the drivers in the AVR32 Software Framework.

```
#include "gpio.h"
```

The API for the GPIO-module is easiest found in the header-file (gpio.h). Use, for example, the following functions:

```
gpio_enable_gpio_pin(unsigned int pin)
gpio_set_gpio_pin(unsigned int pin)
gpio_get_pin_value(unsigned int pin)
```

Exercises (same as above)

2.1 Turn on all LEDs.

2.2 Turn on one LED when a push button is pressed

2.3 Make one bicolored LED change color every time a push button is being pushed.

2.4 Make one bicolored LED change color every time a push button is pressed (and released)

More exercises (not same as above)

2.5 The EVK1100 is also equipped with a small joystick. Investigate which ports/pins this is connected to, and write a small program that turns on/off LEDs with different settings on the joystick.

3. GPIO with board-specific drivers

Use the board-specific drivers from the AVR32 Software Framework. Check out especially the files `board.h`, `led.c` and `led.h`.

Make sure to include the following:

```
#include "board.h"
```

These drivers (`led.c`) contain functions such as:

```
extern void LED_Toggle(U32 leds);

extern void LED_Display(U32 leds);
```

Exercises

3.1 Write a program that turns on and off LEDs using the push-buttons using these drivers (use `LED_On(), LED_Off(), LED_Display()`).

3.2 Do you understand the differences between `LED_Display()` and `LED_On()`? If not, experiment further, and be prepared to explain the differences to the lab assistants!

4. A/D – Analog to Digital conversion

With the peripheral units, we mainly use the low-level drivers and the examples found in the API and documentation of the AVR32 Software Framework.

Use functions such as:

```
extern void adc_configure(volatile avr32_adc_t * adc);
extern void adc_start(volatile avr32_adc_t * adc);
extern unsigned long adc_get_value(volatile avr32_adc_t * adc,
unsigned short channel);
```

The EVK1100 contains three analogue sensors; one potentiometer, one photoresistor (light sensor) and one temperature sensor. These are connected to the AVR32 according to the following table:

Item	**AVR32 port/pin / ADC channel**
Temperature sensor	PA21 / ADC0
Potentiometer	PA22 / ADC1
Photoresistor	PA23 / ADC2

Exercises

4.1 What is the resolution of the A/D-converter? (Check the manual)

4.2 Write a small program that reads the value of the photoresistor and turns on all LEDs when there is low light shining on the photoresistor (covered by the hand), and turns off all LEDs when it's lighter. Demonstrate for the lab assistants.

5. PWM

The PWM module can be configured so that a GPIO pin acts as PWM output. On EVK1100, one PWM channel, PWM3, is connected to PB22, same as one of the bicolored LEDs (PB22 = GPIO number 54).

The following example configures a PWM output on PB22.

```
#include "pwm.h"
#include "gpio.h"

#define EXAMPLE_PWM_PIN                AVR32_PWM_3_PIN
#define EXAMPLE_PWM_FUNCTION           AVR32_PWM_3_FUNCTION

int main(void) {
        unsigned int channel_id;
        avr32_pwm_channel_t pwm_channel = { .ccnt = 0 };  // One
channel config.
        gpio_enable_module_pin(EXAMPLE_PWM_PIN,
EXAMPLE_PWM_FUNCTION);
        pwm_channel.CMR.cpre = AVR32_PWM_CPRE_MCK_DIV_256;  //
Channel prescaler.
        pwm_channel.cdty = 1;   // Channel duty cycle, should be <
CPRD.
        pwm_channel.cprd = 20;  // Channel period.
        // With these settings, the output waveform period will be :
        // (115200/256)/20 == 22.5Hz == (MCK/prescaler)/period, with
MCK == 115200Hz,
        // prescaler == 256, period == 20.

        pwm_channel_init(channel_id, &pwm_channel); // Set channel
configuration to channel 0.
        pwm_start_channels(1 << channel_id);  // Start channel 0.

        while(1){

        }
        return 0;
}
```

Exercises

5.1 How many PWM channels exists?

5.2 Read the manual, look at the API and the example program, and make sure that you are capable of configuring any PWM channel, for any frequency and duty cycle. Demonstrate this for the lab assistants.