Lab 02

Developed by: Muhammad Imran and Peng Cheng

Date : 2015-02-26

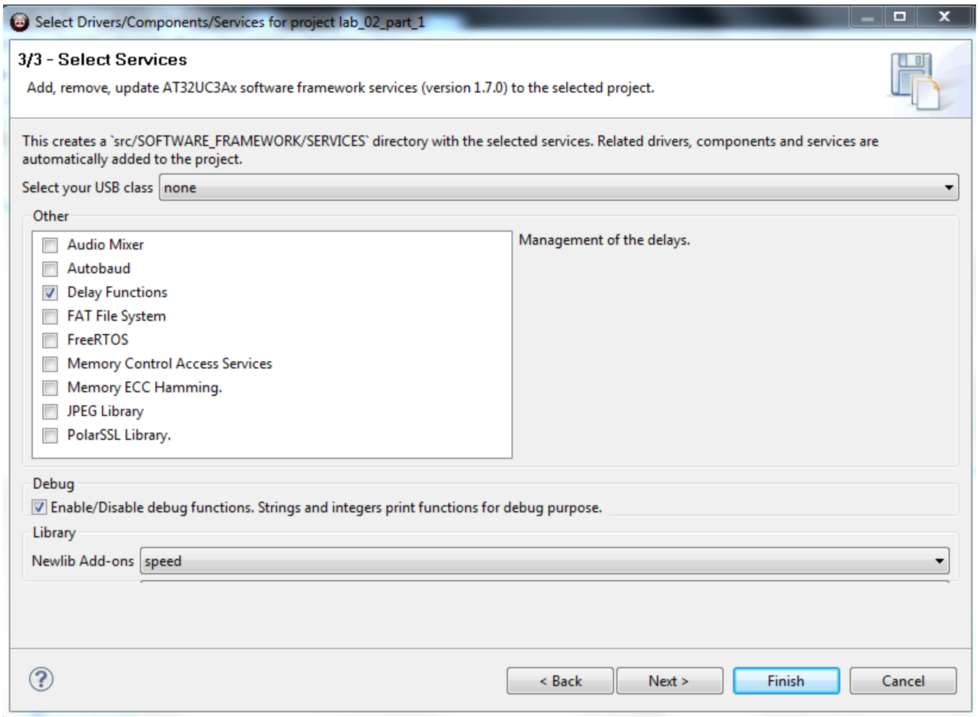
Course : Programming Embedded Systems (ET014G) Mid Sweden University, Sweden.

This lab has two tasks. Task1 is preparatory and will help you in debugging the complex designs of the labs.

Task 1

Debugging is essential in any coding project in order to make a robust system. Each software has aiding tool for making the debugging process easier. In this lab, you can use the AVR studio debugging functionality.

1. You can add this functionality while selecting driver, components and services. An example can be seen in Figure 1.
2. Add header file #include "print_funcs.h" in the main program and then use the different functions in your code from this file.



Select Drivers/Components/Services for project lab_02_part_1

3/3 - Select Services

Add, remove, update AT32UC3Ax software framework services (version 1.7.0) to the selected project.

This creates a `src/SOFTWARE_FRAMEWORK/SERVICES` directory with the selected services. Related drivers, components and services are automatically added to the project.

Select your USB class: none

Other

- □ Audio Mixer
- □ Autobaud
- ☑ Delay Functions
- □ FAT File System
- □ FreeRTOS
- □ Memory Control Access Services
- □ Memory ECC Hamming.
- □ JPEG Library
- □ PolarSSL Library.

Management of the delays.

Debug

☑ Enable/Disable debug functions. Strings and integers print functions for debug purpose.

Library

Newlib Add-ons: speed

Figure 1. Adding debug function

3. For using the debug functionality you can attach the board to computer with a serial cable and use any RS232 terminal to see the messages.

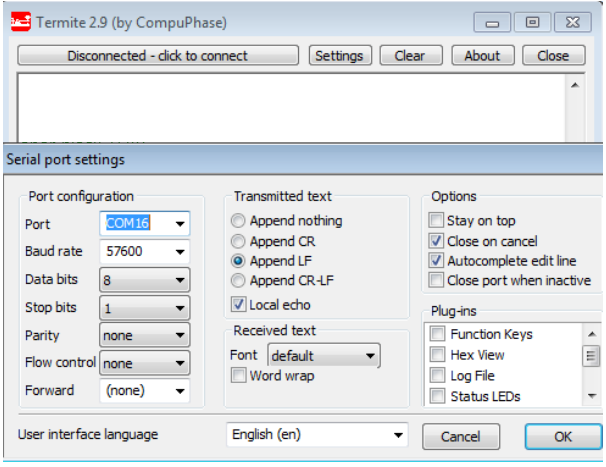
How to get terminal software:

Which software: Anyone!

For example a putty or termite can be downloaded from
http://www.putty.org/

http://www.compuphase.com/software_termite.htm

The terminal can configured with setting shown in Figure 2. Use USART_1 for the debugging process. The port can be set according to your computer.



Termite 2.9 (by CompuPhase)

Serial port settings

Port configuration

Port COM16
Baud rate 57600
Data bits 8
Stop bits 1
Parity none
Flow control none
Forward (none)

Transmitted text

○ Append nothing
○ Append CR
● Append LF
○ Append CR-LF
☑ Local echo

Received text

Font default
□ Word wrap

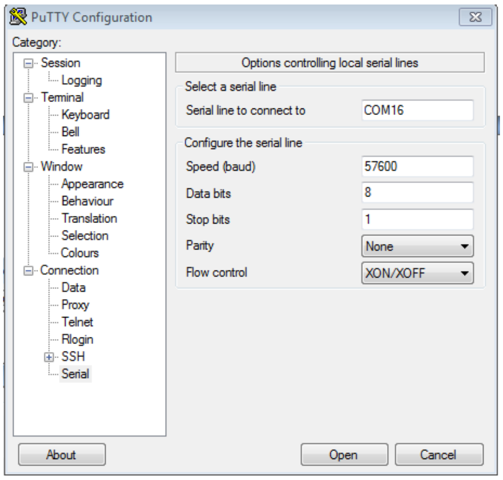
Options

□ Stay on top
☑ Close on cancel
☑ Autocomplete edit line
□ Close port when inactive

Plug-ins

□ Function Keys
□ Hex View
□ Log File
□ Status LEDs

User interface language English (en)



PuTTY Configuration

Options controlling local serial lines

Select a serial line

Serial line to connect to COM16

Configure the serial line

Speed (baud) 57600
Data bits 8
Stop bits 1
Parity None
Flow control XON/XOFF

Figure 2. Terminal setting.

➡ The following program controls the LEDs with push buttons by using interrupt handling method. Modify the program for the following mentioned functionality.
When nothing is pressed, LED1 is on and a corresponding status message of LED1 is displayed on the terminal. When a push button1 is pressed, LED1 is off, LED2 is on and status message of LED2 is displayed on the terminal.
Make sure to add the drivers of GPIO and INTC.

```
/*********************************************************
 Name           : main.c
 Author         : Imran
 Copyright      : Not really
 Description    : EVK1100 template
 *********************************************************/

// Include Files
#include "board.h"
#include "gpio.h"
//#include "intc.h"
#include "avr32/io.h"

__attribute__((__interrupt__))
static void int_handler (void)
{
  if( gpio_get_pin_interrupt_flag( GPIO_PUSH_BUTTON_0 ) )
  {
                LED_On( LED1 );
                LED_Off( LED0 );
                gpio_clear_pin_interrupt_flag(GPIO_PUSH_BUTTON_0);
  }
}
int main(void) {

              /* TODO: replace this comment with your code */
               gpio_enable_pin_interrupt(GPIO_PUSH_BUTTON_0 , GPIO_PIN_CHANGE);
               INTC_init_interrupts ();

// "In every port there are four interrupt lines connected to the interrupt
// controller. Every eigth interrupts in the port are ored together to form an
// interrupt line."
// AVR32_INTC_INT0 is for the interrupt priority level.
// Every eight interrupts in the port are stored together to form an interrupt line. That
// means each interrupt line can handles
// 8 pins. GPIO 0-7 ((PA0-PA07)) are handled by AVR32_GPIO_IRQ_0 and GPIO 104-109 (PX36-PX11)
// by AVR32_GPIO_IRQ_13.
// Simple way is to use formula "AVR32_GPIO_IRQ_0 + (GPIO to be registered/8)".
// in the formular AVR32_GPIO_IRQ_0 will act as a base address and (GPIO to be registered/8)
// will point to the specific line
// for Pushbutton 0 (PX16- GPIO 88) you can use the formula to have 75 address or use
AVR32_GPIO_IRQ_11 which is assigned 75 in uc3a0512.h
               INTC_register_interrupt(&int_handler,  (AVR32_GPIO_IRQ_0+88/8),
AVR32_INTC_INT0);
               Enable_global_interrupt ();
               while (1)
               {
                LED_On(LED0);
               }
              return 0;
}
```

Task2

On your computer, download and install this HxD freeware hex editor and disk editor from:
http://mh-nexus.de/en/hxd/
Read the contents of SD card by using by a menu *open disk*. Remember to open it as administrator.

For EVK1100:

For this lab, consult SD/MMC card example and SDRAM example. SD/MMC card example shows how to read from SD/MMC to internal RAM by using PDCA. In this lab, we do the opposite by writing to the SD card and involve external SDRAM.

Initialize the whole microprocessor into the fastest clock speed possible.

> You can use power manger driver for setting the frequency. Specifically the function pm_configure_clocks() will help you in setting the frequency.
> While setting frequency for peripheral bus A, considers errata on Page 813 in the datasheet which describes the limitation of PBA clock. Following function gives the some further help.

```
pm_freq_param_t System_Clock = {
                .cpu_f = CPU_HZ,
                .pba_f = PBA_HZ,
                .osc0_f = FOSC0,
                .osc0_startup = OSC0_STARTUP
            };
```

Initialize the sdramc module in order to access the external 32MB SDRAM.

```
volatile unsigned char *sdram = SDRAM; //SDRAM address
// Initialize the external SDRAM chip.
sdramc_init(FOSC0);
print_dbg("SDRAM initialized\n");
//for detail, see SDRAM example
```

Within HSB bus matrix, set EBI slave to have default master as PDCA.

> “the 4-bit FIXED_DEFMSTR field selects a fixed default master provided
> that DEFMSTR_TYPE is set to fixed default master. Please refer to the Bus Matrix user
> interface description”. Page 133

```
// Setting EBI slave to have fixed default master
AVR32_HMATRIX.SCFG[AVR32_HMATRIX_SLAVE_EBI].defmstr_type     =
AVR32_HMATRIX_DEFMSTR_TYPE_FIXED_DEFAULT;

Setting EBI slave to have PDCA as a master
```

```
AVR32_HMATRIX.SCFG[AVR32_HMATRIX_SLAVE_EBI].fixed_defmstr    =
AVR32_HMATRIX_MASTER_PDCA;
```

Initialize the PDCA with SPI TX mode and SPI modules in order to access the external 2GB SD card that is provided by us.

See example SD/MMC card example

Pre-write the following dummy data into the whole 32MB SDRAM: Data byte sequence start from 0x00 increment to 0xFF and repeat again.

Using PDCA to read the whole 32MB SDRAM and write the data into the SD card.

For PDCA configuration, consult" SD/MMC card example". For writing to SD/MMC card, use the function *sd_mmc_spi_ram_2_mem* from "sd_mmc_spi_mem.c" or *sd_mmc_spi_write*_sector_from_ram from "sd_mmc_spi.c". Both functions write one MMC sector (512 bytes) from a ram buffer.

The SD card can be accessed 512 byte (or one block) at a time.

After this is done, power off the EVK1100 and show the SD card content using the HxD editor.

Drivers:

DRIVERS/PM
DRIVERS/EBI/SDRAMC
COMPONENTS/MEMORY/SDRAM/
DRIVERS/HMATRIX
DRIVERS/PDCA
DRIVERS/SPI
COMPONENTS/MEMORY/SD_MMC/SD_MMC_SPI/

References:

1. AVR32UC3A Datasheet Rev.K
2. EVK1100_Schematics_RevC

Useful Tip:

Quick references to datasheet and schematic can be found
http://www.avrfreaks.net/wiki/index.php/Documentation:EVK1100/Hardware_Reference