Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics

Department of Automation and Applied Informatics
Embedded and control systems specialization

Embedded and Ambient Systems Laboratory

Laboratory guide

ARM microcontroller programming in C language

Written by: Zoltán Szabó (zoltan.szabo@aut.bme.hu)
Viktor Kovacs (viktor.kovacs@aut.bme.hu)
Gergely Kardos (gergely.kardos@aut.bme.hu)
Norbert Koszó (norbert.koszo@aut.bme.hu)

Last modified: April 13, 2016

Introduction

The main goal of the laboratory is to get familiar with the STM32F4 development kit and its programming in C language, using an Eclipse-based development environment.

Description of the laboratory

Through the measurement tasks, students get an overview and get familiar with programming the SM32F4Discovery development kit and its extension board in C language. It is recommended to use the available peripheral library, which provides descriptively named functions to modify the registers of the hardware. The peripheral library's source code is available for free, and it contains useful comments, making the development easier.

The extension board

An extension board has been created for the F4Discovery developer kit that contains numerous input and display devices to extend the functionality of the developer kit. The following peripherals can be found on the extension board (if the signal name starts with # - means the signal is low active):

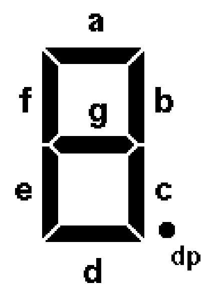
- 4 pieces of pushbuttons with common enable input (BTNEN – PD11; BTN1 – PD12; …; BTN4 – PD15)
- 4 pieces of switches with common enable input (SWEN – PC6; SW1 – PE2; SW2 – PE4; SW3 – PE5; SW4 – PE6)
- Display peripherals: LEDs, 7 segment displays, graphic LCD display. Each one of these has an 8 bit bus driver which is connected to the same 8 bit bus. The LEDs and the 7 segment displays use D flip-flops with clock and enable inputs, the LCD display is driven by a bidirectional bus driver with direction and enable inputs.
    - 8 pieces of LEDs, with common enable and clock signal (#LEDEN – PC11; LEDCLK - PA15; LD1 – PE8 (DB0); …; LD8 – PE15 (DB7))
    - 4 pieces of 7 segment displays, with enable, clock, and two selection signals (#7SegEN – PD10; 7SegCLK – PD2; display selector inputs: 7SEL0 - PB14, 7SEL1 – PB15; segments: „a" – DB0 – PE8, „b" – DB1 – PE9, … „DP" – DB7 – PE15)
    - 64x128 pixel monochrome graphic LCD display with enable (#GLCDEN – PB7; backlight enable: BL_PWM – PC8; control: GLCD_RESET – PD3, GLCD_R/#W – PE7; GLCD_CS1 – PB4; GLCD_CS2 – PB5; GLCD_E – PD7; GLCD_DI – PD3; data: DB0 – PE8; …; LCD_DB7 – PE15)
- Analog potentiometer (PB1)
- Analog input (PB0) (Directly or through a follower differential amplifier, or through a comparator with hysteresis.)
- I^2C temperature meter (SDA- PB9; SCL – PB6; T_INT – PB8)

- Digital SPI potentiometer which sets the parameters of the comparator of the analog input (SPI_CS_POT – PA3; MOSI – PA7; SPI_SCK – PA5)
- RS232 port (USART3_TX - PD8; USART3_RX - PD9)
- CAN (CAN1_RX - PD0; CAN1_TX - PD1)
- Micro servo PWM output (PA8)
- Ethernet (JP3 needs to be shorted to operate, which also disables the MEMS sensor on the F4Discovery kit)

Preparing for the laboratory

Review the schematics of the extension board! In the laboratory we will use the push buttons, the switches, the seven segment displays and the LEDs, so focus on these peripherals. During the laboratory the push buttons, switches, LEDs and 7 segment displays will be used.

Fill the following table! The table specifies the content of the control byte for each displayable character on the 7 segment display

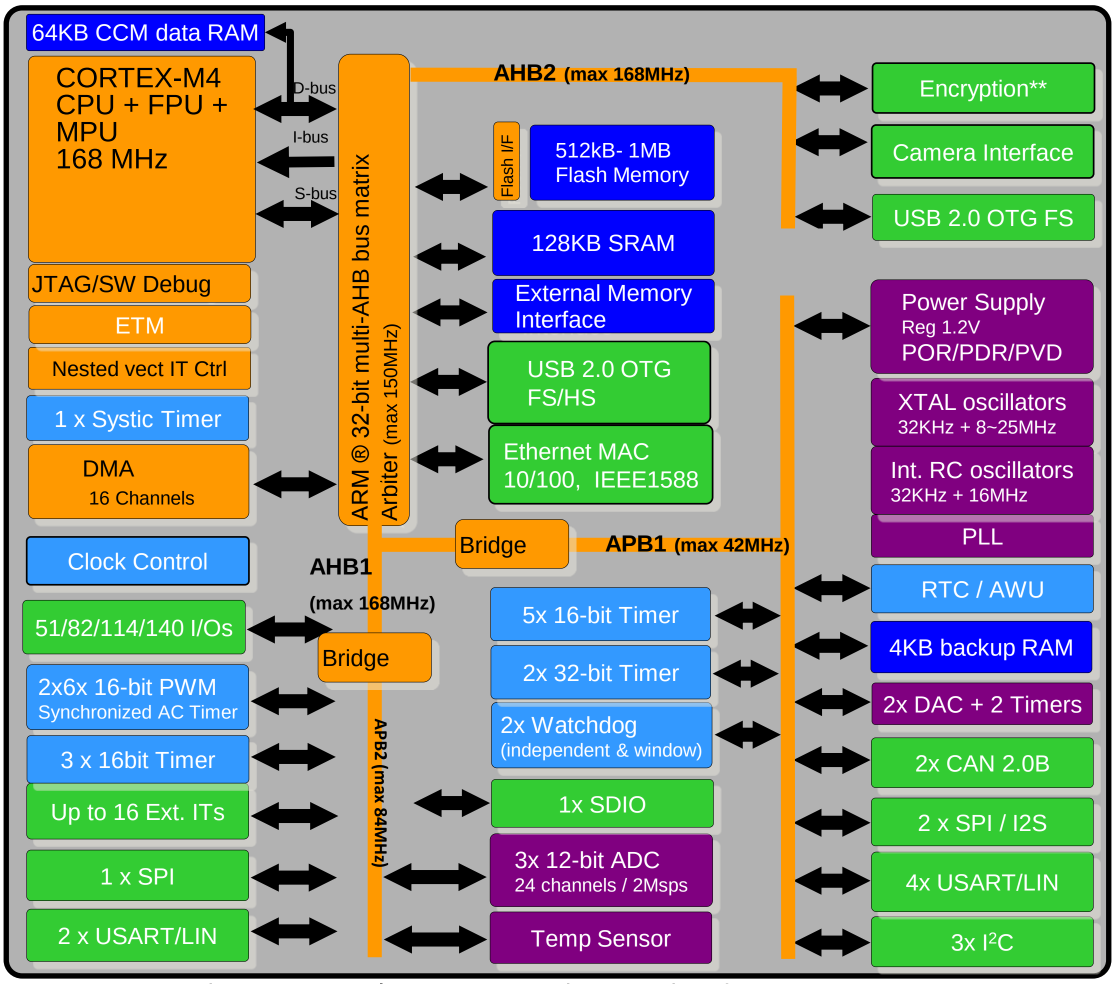


	a PE8 (DB0)	b PE9 (DB1)	c PE10 (DB2)	d PE11 (DB3)	e PE12 (DB4)	f PE13 (DB5)	g PE14 (DB6)	dp PE15 (DB7)	Hexa érték
0	X	X	X	X	X	X	-	-	3F
1									
2									
3									
4									
5									
6									
7									
8									
9									
A									
b									
C									
D									
E									
F									

Review the parts of the peripheral library which are relevant for the laboratory!

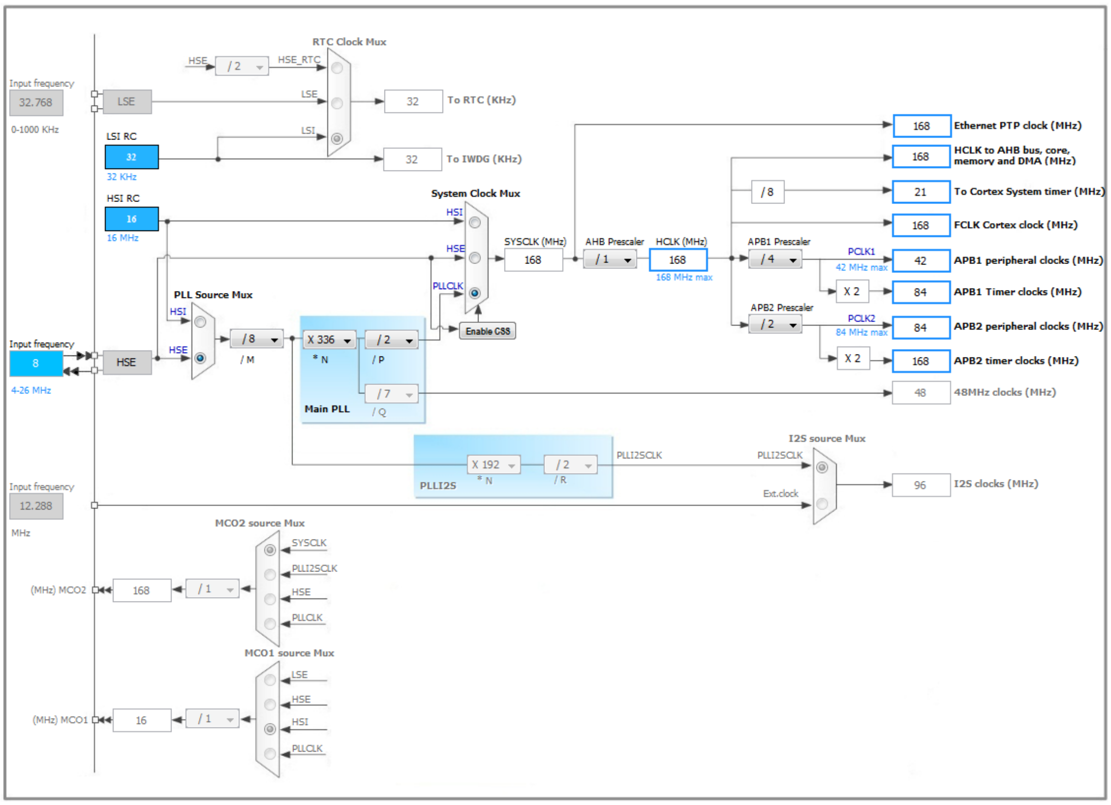
Introduction to the microcontroller and it's programming

Block structure of the MCU:



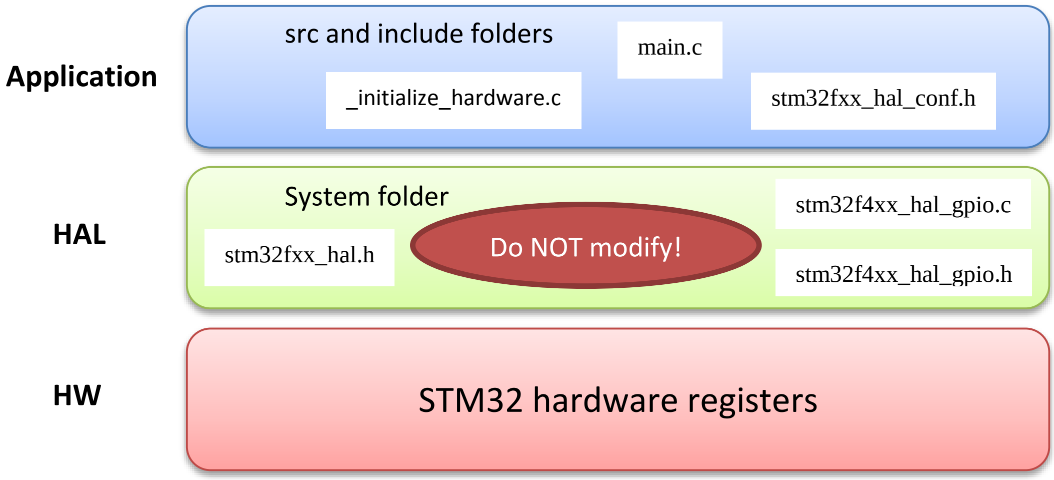
HS requires an external PHY connected to ULPI interface,
** Encryption is only available on STM32F415 and STM32F417

The MCU has a lot of peripherals, but only a few of them will be used during the laboratory (GPIO, Timer, NVIC…).
To start the MCU, two essential things must be done: the stack pointer needs to be initialized and the entry point of the main function must be set at the reset vector. Additionally it is recommended to implement the error handler functions. These functions are already implemented in the library for the STM32F407VGT6 microcontroller and the STM32 HAL library. Finally the system clock must be configured, but the default initialization also accomplishes this task for this microcontroller
The clock structure of the system can be seen on the image below.



The clock system of the MCU is complicated and has many options. The manufacturer provides an application note which describes the clock system in details and makes understanding easier.
After reset, the MCU operates from its internal 16MHz RC oscillator (HSI), and almost all the peripheries switched off from the clock network. During the laboratory, a template program will be given, which sets up the external crystal oscillator and configures the PLL to work on the maximal 168MHz clock. But it does not enable the clock of any periphery. These clocks must be enabled manually where it is necessary.

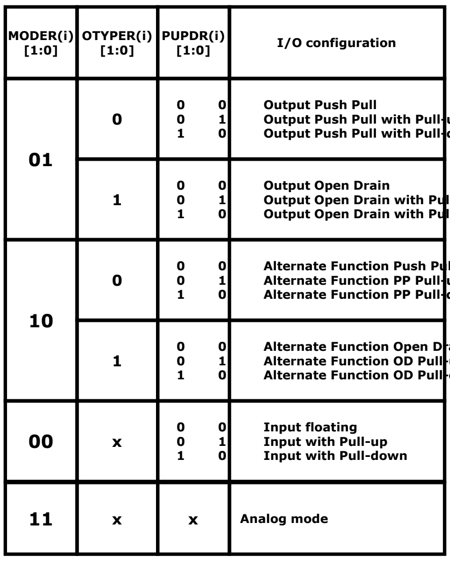
Using the peripheral library



The peripheral library has been written in ANSI C language with separate source and header files to each periphery (e.g. stm32f4xx_hal_gpio.c, stm32f4xx_hal_gpio.h). To use the peripheral library the stm32f4xx_hal.h and the stm32f4_hal_cortex.h header files must be included. The header included in the project (stm32f4xx_hal_conf.h) file includes all the HAL header files. The unnecessary includes may be removed to decrease compilation time. Full description of the HAL libraries can be found in the referenced document. Also the source files are well commented and contain a description at the beginning. Types, constants, symbols and functions defined in the HAL header files are named self-describing as all start with prefixes of the peripheries: HAL_GPIO_Init(), NVIC_PriorityGroup_0 etc.

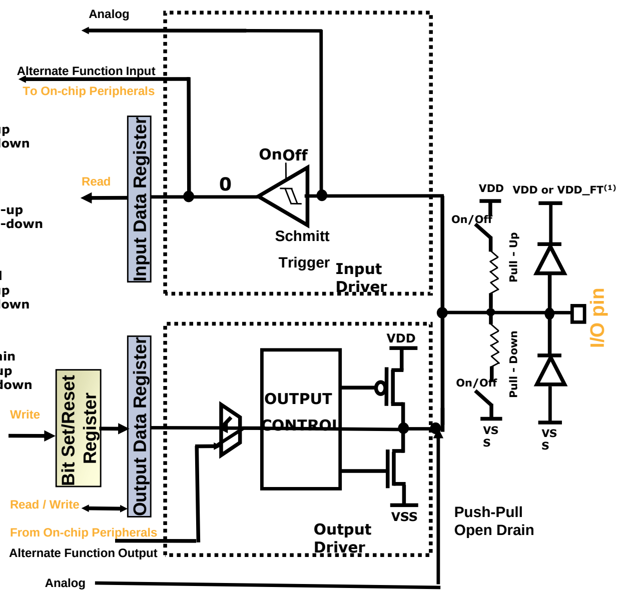
GPIO configuration

The following block diagram shows the structure of one GPIO:



MODER(i) [1:0]	OTYPER(i) [1:0]	PUPDR(i) [1:0]		I/O configuration
01	0	0	0	Output Push Pull
		0	1	Output Push Pull with Pull-up
		1	0	Output Push Pull with Pull-down
	1	0	0	Output Open Drain
		0	1	Output Open Drain with Pull-up
		1	0	Output Open Drain with Pull-down
10	0	0	0	Alternate Function Push Pull
		0	1	Alternate Function PP Pull-up
		1	0	Alternate Function PP Pull-down
	1	0	0	Alternate Function Open Drain
		0	1	Alternate Function OD Pull-up
		1	0	Alternate Function OD Pull-down
00	x	0	0	Input floating
		0	1	Input with Pull-up
		1	0	Input with Pull-down
11	x	x		Analog mode

* In output mode, the I/O speed is configurable through OSPEEDR register: 2MHz, 25MHz, 50MHz or 100 MHz



(1) VDD_FT is a potential specific to five-volt tolerant I/Os and different from VDD.

First clock must be enabled for a given GPIO port with the following macro: __GPIOx_CL_ENABLE() where x denoted the port identifier (GPIOA, GPIOB etc).
After reset, every GPIO pin is configured in input floating mode. This can be modified by the HAL_GPIO_Init(…) function. The first parameter of the function is the port (GPIOA, GPIOB, etc.) the second is a GPIO_InitTypeDef structure which contains the following variables:

- Pin -> GPIO_Pin_0 .... 15, GPIO_Pin_All, GPIO_Pin_None (Multiple selection is possible with bitwise OR)
- Mode:
  - GPIO_MODE_INPUT // Input Floating Mode
  - GPIO_MODE_OUTPUT_PP // Output Push Pull Mode
  - GPIO_MODE_OUTPUT_OD // Output Open Drain Mode
  - GPIO_MODE_AF_PP // Alternate Function Push Pull Mode
  - GPIO_MODE_AF_OD // Alternate Function Open Drain Mode
  - GPIO_MODE_ANALOG // Analog Mode

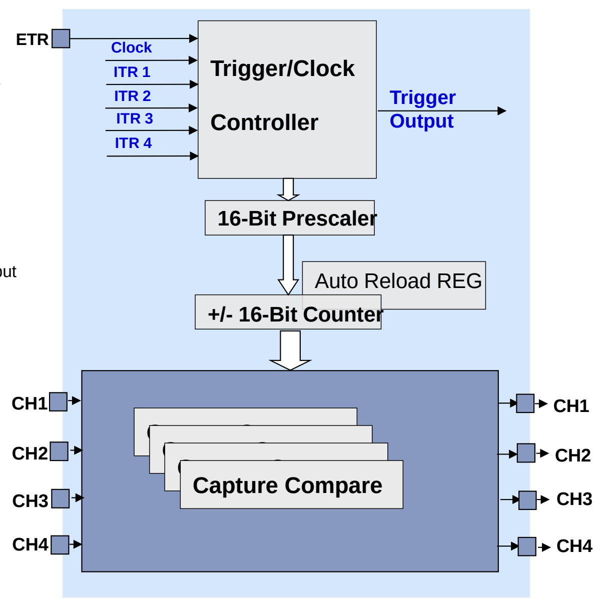
  - GPIO_MODE_IT_RISING // External Interrupt Mode with Rising edge trigger detection
  - GPIO_MODE_IT_FALLING // External Interrupt Mode with Falling edge trigger detection
  - GPIO_MODE_IT_RISING_FALLING // External Interrupt Mode with Rising/Falling edge trigger detection
  - GPIO_MODE_EVT_RISING // External Event Mode with Rising edge trigger detection
  - GPIO_MODE_EVT_FALLING // External Event Mode with Falling edge trigger detection
  - GPIO_MODE_EVT_RISING_FALLING // External Event Mode with Rising/Falling edge trigger detection
- Speed:
  - GPIO_SPEED_LOW //lowest EMI -> softer edges
  - GPIO_SPEED_MEDIUM
  - GPIO_SPEED_FAST
  - GPIO_SPEED_HIGH //highest EMI -> sharper edges
- Pull:
  - GPIO_ NOPULL
  - GPIO_PULLUP
  - GPIO_PULLDOWN
- Alternate
  - GPIO_AF0_RTC_50Hz
  - …
  - GPIO_AF15_EVENTOUT

Basic timer handling

This microcontroller includes several timers with different capabilities, but we will only cover the basic functionalities. The following figure presents the functionalities of a general purpose timer:

General Purpose timer Features overview

- TIM2, 3, 4 and 5 on Low Speed APB (APB1)
- Internal clock up to **84 MHz** (if AHB/APB1 prescaler distinct from 1)
- 16-bit Counter for TIM3 and 4
- 32-bit Counter for TIM2 and 5
  - Up, down and centered counting modes
  - Auto Reload
- 4 x 16 High resolution Capture Compare Channels
  - Programmable direction of the channel: input/output
  - Output Compare
  - PWM
  - Input Capture, PWM Input Capture
  - One Pulse Mode
- Synchronization
  - Timer Master/Slave
  - Synchronisation with external trigger
  - Triggered or gated mode
- Encoder interface
- 6 Independent IRQ/DMA Requests generation
  - At each Update Event
  - At each Capture Compare Events
  - At each Input Trigger



Functions used for configuration:

- __TIMx_CLK_ENABLE() – enabling the timer clock source
- TIM_Base_InitTypeDef:
    - Period – last value of the counter
      f = TIM_counter_clk/(Period+1)
    - Prescaler - [0 -> 2^{16}-1] – clock division value
      TIM_counter_clk=APBx_clk/(prescaler+1)
    - ClockDivision – used for digital filters, let's set it to 1
    - CounterMode – TIM_COUNTERMODE_ [UP/DOWN/CENTERALIGNED(1..3)]
- HAL_TIM_Base_Init(…) – initialize the timer based on the structure
- HAL_TIM_Base_Start_IT() – start the timer
- HAL_TIM_Base_Stop_IT() – stop the timer

During the exercises Timer4 and Timer 6 are used, these connect to bus APB1, thus the base clock for these timers is 84MHz.

Interrupt handling

An interrupt controller (NVIC) is responsible for interrupt handling in this microcontroller. After reset all peripheral interrupts is disabled. To enable interrupts, the interrupt groups and priority levels must be set up in the interrupt controller and the interrupt handler function must be implemented. In case of external interrupts, the external interrupt controller (EXTI) and the system configuration controller (which maps the ports to interrupt channels) also must be configured.

Interrupts may be grouped into "preemption priority" or "sub priority" group. The main difference between the two groups is that interrupts in preemption priority (lower number, higher priority) may interrupt sub priority interrupts. In case of interrupts with the same preemption priority the handling order is defined by sub priority. The number of preemption and sub priority levels depends on the priority group settings. The following table defines the relation between the group settings and the number of levels in each group.

PRIGROUP (3 bits)	Binary Point (group.sub)		Preemting Priority (Group Priority)		Sub-Priority	
			Bits	Levels	Bits	Levels
011 (NVIC_PriorityGroup_4)	4.0	gggg	4	16	0	0
100 (NVIC_PriorityGroup_3)	3.1	gggs	3	8	1	2
101 (NVIC_PriorityGroup_2)	2.2	ggss	2	4	2	4
110 (NVIC_PriorityGroup_1)	1.3	gsss	1	2	3	8
111 (NVIC_PriorityGroup_0)	0.4	ssss	0	0	4	16

Five priority models are available, which determine the number of bits available for priority groups and subpriorities. Lower priority numbers denote higher priority.

- HAL_NVIC_SetPriorityGrouping(…) – set up priority model

- HAL_NVIC_SetPriority(…) – set the priority of a selected interrupt
- HAL_NVIC_EnableIRQ(…) – enable a selected interrupt
- HAL_NVIC_DisableIRQ(…) – disable a selected interrupt

In order to handle a given interrupt the native interrupt handler function must first be written. The function name associated to a given interrupt can be found in vectors_stm32f4xx.c, ie. TIM4_IRQHandler is associated to TIM4:

```
  TIM4_IRQHandler,         // TIM4
```

In this handler routine the task is only to call the predefined handler function of the HAL library, which will later call back the final handler function. The greatest benefit of this method is that the builtin handler of the library does all the checks and clears the interrupt flag. The developer must only care about the application logic in the callback function. The name of the callback function can be found in the documentation or the related header file. In the example the following function is called by at the overflow of a timer:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
```

Laboratory tasks

A number of IDE software available for the STM32 microcontrollers (MDK-ARM, IAR, TASKING). We are going to use **Eclipse** CDT with Sourcery CodeBench Lite Edition for ARM EABI GCC compiler and OpenOCD debugger to avoid the limitations of proprietary IDEs.
(A short tutorial to the most popular IDEs can be found in UM1467 documentation on the ST webpage).

Create a folder under "D:\users" and unzip the file under it. The source code will contain comments, which helps solving the tasks. At some parts functions from the periphery library has to be used. Only the name of the function will be given, the parameter list has to be looked up. The signals of the peripheries can be easily read from the schematics of the board.

Task 0: Project and IDE setup

Let's start the Eclipse IDE, which will ask where the workspace should be. Select the previously created folder and do NOT make it default.
Select File|Import|General/Existing Projects into Workspace. Select archive file (select the previously downloaded template) and press Finish.
Setting up the debug configuration:
Roll down the debug button and select Debug configurations. Select GDB OpenOCD Debugging and add a New Debug Configuration. On the Debugger tab, select the openocd executable. In the other options box, paste:

```
-f board/stm32f4discovery.cfg
```

Debugging should work now. Close the window.
Go to Window/Preferences/General/Workspace, turn on Save automatically before build option.

Go to Window/Preferences/ C/C++/Editor/Content Assist/Advanced, turn on all options. Close. Right click the name of the project in the Project explorer, select index/Rebuild.

Task 1: Basic port handling, timers

1.1 Chase light with software timing

Write a program, which create running lights on the LD1-LD8 LEDs of the F4-Ext board. Use software timing.

Files to modify: main.c, bsp_led.c, bsp.c
Functions to implement/modify:

- main (modify)
- bsp_led.c (implement all functions)
- HAL_MspInit (modify)

Recommended functions from the library:

- __GPIOx_CLK_ENABLE() – enable periphery clock source
- HAL_GPIO_Init() – initialize GPIO ports
- HAL_GPIO_WritePin(), HAL_GPIO_TogglePin() – setting, modifying PGIO pin state
- GPIOx->ODR – GPIO port output register

The LEDs are driven by a flip-flop circuit, which also needs to be enabled (LED_CLK, #LEDEN, DB0-7).

1.2 Running light with hardware timing

Modify the previous program. Use Timer4 with interrupt for timing. Each LED have to light for 250ms.

Files to modify: main.c, bsp_timer.c, bsp.c
Functions to implement/modify:

- main (modify)
- Timer4_Init, Timer4_Start, Timer4_Stop, HAL_TIM_PeriodElapsedCallback (implement)
- HAL_MspInit (modify)

Recommended functions from the library:

- HAL_NVIC_SetPriorityGrouping() – NVIC priority model selection
- __TIMx_CLK_ENABLE() – enable timer clock
- HAL_TIM_Base_Init() – timer initialization
- HAL_NVIC_SetPriority() – setup interrupt priority
- HAL_NVIC_EnableIRQ() – enable interrupt
- HAL_TIM_Base_Start_IT() – start timer with interrupt mode
- HAL_TIM_Base_Stop_IT() – stop timer with interrupt mode

Task 2: Stop watch

In this task a stop watch have to be implemented, which shows the time with 0.1s precision on the seven-segment displays.

1.3 Seven-segment display

Write a function (DisplayDigit) which displays a single digit on one of the seven-segment displays. The function has to accept three parameters: which display to write, what hexadecimal value to write (0…F), and light up the decimal point or not. To assign the hexadecimal numbers with the seven-segment displays hexadecimal code set the SegmentTable array in the code. To try your DisplayDigit implementation, display the "F" hexadecimal value on the 2nd display. Call DisplayDigit() from the main() function.

Files to modify: main.c, bsp_7seg.c, bsp.c
Functions to implement/modify:

- main (modify)
- implement all methods in bsp_7seg.c
- HAL_MspInit (modify)

Recommended functions from the library:

- __GPIOx_CLK_ENABLE() – enable periphery clock source
- HAL_GPIO_Init() – initialize GPIO ports
- HAL_GPIO_WritePin(), HAL_GPIO_TogglePin() – setting, modifying PGIO pin state
- GPIOx->ODR – GPIO port output register

All GPIO ports have 16 bits. GPIO_Write() modifies all of these bits at once, keep it in mind if you want to modify only few of them.

1.4 Displaying with time multiplexing

Write the interrupt handler function of the Timer4 to display the last 4 hexadecimal digits of the global "counter" variable on the seven-segment displays. On the schematic of the F4-Ext board it can be clearly seen, that the seven-segment displays must be driven with time multiplexing. The numbers on the displays must look stationary (they should not flicker); keep it in mind when setting up timer4. The displays will show time in tenth of a second precision (e.g. 123.9), turn on the appropriate decimal point!

Files to modify: main.c, bsp_timer.c
Functions to implement/modify:

- main (modify)
- Timer4_init (modify)
- HAL_TIM_PeriodElapsedCallback (modify)

1.5 Counter with 0.1s precision

Set up the Timer6 to generate interrupts with 0.1s intervals. Write the interrupt handler function to increase the "counter" variable. The "counter variables value is only valid in the range of [0…9999].

Files to modify: main.c, bsp_timer.c, bsp.c
Functions to implement/modify:

- main (modify)
- HAL_MspInit (modify)
- Timer4_init (modify)
- HAL_TIM_PeriodElapsedCallback (modify)

1.6 Control the timer with buttons

Set the corresponding GPIOs, the EXTI and NVIC controller to generate interrupt every time, when BTN[123] is pressed. The BTN1 has to start, BTN2 has to stop the measurement, and BTN3 has to clear the counter.

Files to modify: main.c, bsp_button.c, bsp.c
Functions to implement/modify:

- main (modify)
- HAL_MspInit (modify)
- Implement all functions in bsp_button.c

Task 3. Reaction game (optional task)

Based on the previous task, create a reaction game. In the game after BTN4 pressed, lit up one of the LEDs from LD[123]. At the same time, start a stop watch on the seven-segment displays with hundred of a second precision. The player has to press the appropriate button from BTN[123] as fast as he can. The stop watch has to stop, when the appropriate button is pressed. Display the elapsed time in seconds dot hundreds of seconds format (e.g. 01.89), on the seven-segment displays. After the BTN4 is pressed, wait 1-4 seconds randomly before turning on one of the LEDs. The internal random number generator can be used to generate random numbers,

Questions for preparation

- What clock source is used by the microcontroller after reset?
- What is the maximal system clock of the controller?
- What operating modes does a GPIO port have?
- What setup parameters does a basic timer have?
- What are the typical steps of a periphery initialization? What kind of functions are available for this purpose?
- What are priority groups and sub priority groups, what are the main differences?
- What is needed to handle an interrupt using the STM32 HAL library?
- What steps are required using the STM32 platform to enable a timer peripheral and handle its interrupts?

Related documents

Information about the MCU from STMicroelectronics:
http://www.st.com/internet/mcu/subclass/1521.jsp
Useful site about the MCU:
http://www.emcu.it/STM32F4xx/STM32F4xx.html

Appendix

Mapping of peripheries on the F4-Ext board:

Name	PortPin	
DB0	PE8	7seg_a/LD1/GLCD_DB0
DB1	PE9	7seg_b/LD2/GLCD_DB1
DB2	PE10	7seg_c/LD3/GLCD_DB2
DB3	PE11	7seg_d/LD4/GLCD_DB3
DB4	PE12	7seg_e/LD5/GLCD_DB4
DB5	PE13	7seg_f/LD6/GLCD_DB5
DB6	PE14	7seg_g/LD7/GLCD_DB6
DB7	PE15	7seg_dp/LD8/GLCD_DB7
#LDEN	PC11	
LED_CLK	PA15	
#7SEN	PD10	
7SEL0	PB14	
7SEL1	PB15	
7SEG_CLK	PD2	
BTNEN	PD11	
BTN1	PD12	
BTN2	PD13	
BTN3	PD14	
BTN4	PD15	