THE NELSON MANDELA

AFRICAN INSTITUTION OF SCIENCE AND TECHNOLOGY ;

ARUSHA, TANZANIA

Embedded Systems Engineering (ESE) Lab

TRAINING MANUAL FOR THE EMBEDDED SYSTEMS ENGINEERING LABORATORY

Contents

1. Introduction

This document serves as the training manual for the students and research in the Embedded System Engineering laboratory.

The Embedded Systems Engineering laboratory aims to equip Masters students specialising in Embedded and Mobile Systems to develop smart-based-application applications using STM32 Microcontrollers and Raspberry Pi.

2. Installation and configuration of the STM32 microcontroller board

2.1. Description and specifications of STM32 microcontroller board

Embedded systems engineering is a highly interdisciplinary approach, requiring a tight integration of core principles of computer science and electrical engineering with application-specific knowledge. Embedded systems engineering is continuously keeping pace with the development in other fields of information processing and networking (E.g. Security, the Internet of Things, etc.). An embedded system must be able to work with limited resources (as required by the application). Unused resources are uselessly increasing the form size, the weight and the costs of the final product.

The Cortex-M4 processors use a 32-bit architecture. Internal registers in the register bank, the data path, and the bus interfaces are all 32-bit wide. The Instruction Set Architecture (ISA) in the Cortex-M processors is called the Thumb ISA and is based on Thumb-2 Technology which supports a mixture of 16-bit and 32-bit instructions.

The Cortex-M4 processors have:

• Three-stage pipeline design

• Harvard bus architecture with unified memory space: instructions and data use the same address space

• 32-bit addressing, supporting 4GB of memory space

• On-chip bus interfaces based on ARM AMBA (Advanced Microcontroller Bus Architecture) Technology, which allow pipelined bus operations for higher throughput

• An interrupt controller called NVIC (Nested Vectored Interrupt Controller) supporting up to 240 interrupt requests and from 8 to 256 interrupt priority levels (dependent on the actual device implementation)

• Support for various features for OS (Operating System) implementation such as a system tick timer, shadowed stack pointer

• Sleep mode support and various low-power features

• Support for an optional MPU (Memory Protection Unit) to provide memory protection features like programmable memory, or access permission control

• Support for bit-data accesses in two specific memory regions using a feature called Bit Band

• The option of being used in single-processor or multi-processor designs

The ISA used in Cortex-M3 and Cortex-M4 processors provides a wide range of instructions:

• General data processing, including hardware divide instructions

• Memory access instructions supporting 8-bit, 16-bit, 32-bit, and 64-bit data, as well as instructions for transferring multiple 32-bit data

• Instructions for bit field processing

• Multiply Accumulate (MAC) and saturate instructions

• Instructions for branches, conditional branches and function calls

• Instructions for system control, OS support, etc.

In addition, the Cortex-M4 processor also supports:

• Single Instruction Multiple Data (SIMD) operations

• Additional fast MAC and multiply instructions

The Cortex-M4 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including:

- outstanding processing performance combined with fast interrupt handling
- enhanced system debug with extensive breakpoint and trace capabilities
- efficient processor core, system and memories
- ultra-low power consumption with integrated sleep mode and an optional deep sleep mode
- platform security robustness, with optional integrated *Memory Protection Unit* (MPU).

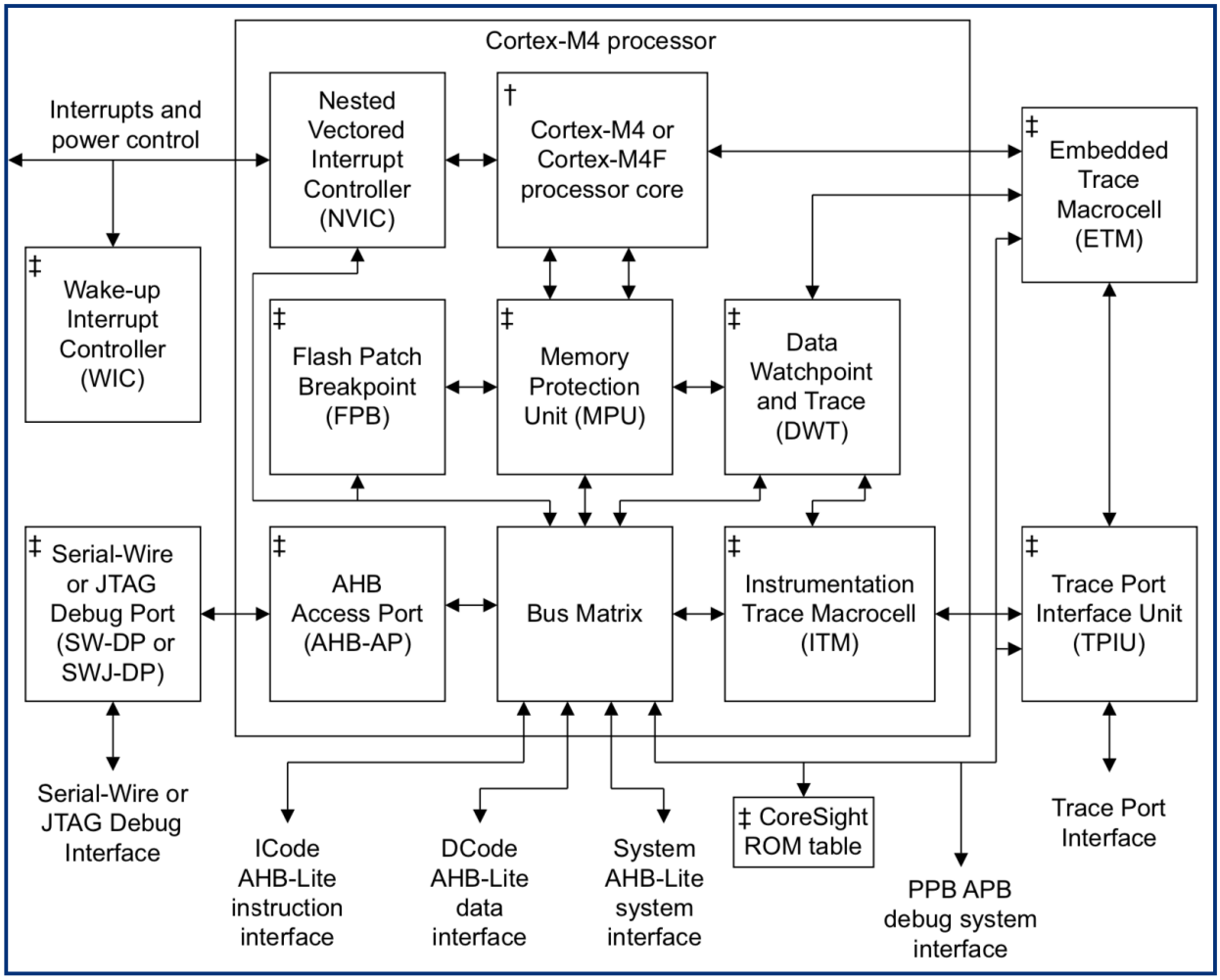

Fig 1. Block diagram for Cortex-M4 processor

Among the other things Cortex-M4 processor from ARM has the following components:

- CPU – Central Processing Unit may have multiple cores. For example Cortex-M4 processor has only one core, therefore it is a single core Processor.
- In short, Processor is Core + Surronding Peripherials specific such as NVIC, FPB, DWT, MPU, ITM etc.
- The ARM processor communicates with external word via different bus interfaces such as Icode interface, Dcode interface and System interface.

2.2. Installing an Integrated Development Environment (IDE)

- IDE is a software that contains all the essential tools to develop, compile, link, and debug your code.
- In some cases, an embedded developer has to integrate compiler and debugger tools to the IDE manually.
- Throughout this course we will be using Eclipse-based STM32CubeIDE which is developed by ST Microelectronics to write, compile, debug applications on STM32 ARM-based microcontrollers.
- STM32CubeIDE is an eclipse IDE with STM32 related customization
- Download STM32CubeIDE
- Link: https://www.st.com/en/development-tools/stm32cubeide.html
- Developer has to register and login in order to download the software

(a). Required documents for embedded systems development

a. Two sets of documents

1. Documents related to a processors
2. Document s related to microcontrollers
3. Document related to a development board

(b). Documents related to microcontroller will be downloaded from st.com

a. Datasheet
b. Reference Manual

(c). Board related documents

a. User Manual

b. Schematic diagrams

(d). Processor related documents

a. Cortex-M Generic User Guide

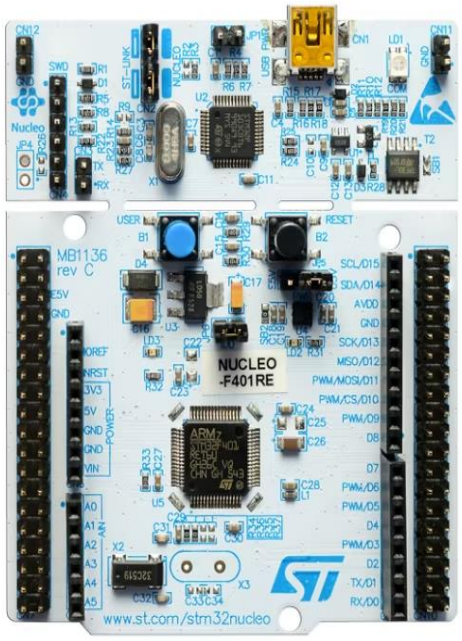

Fig 2. NUCLEO-F401RE Microcontroller board

3. Locating the Ports and Pins of Components on the Nucleo board

The NUCLEO-F401RE Microcontroller board is used in the laboratory setup. This microcontroller has 64 pins arranged in ports. All ports and pins used in STM32F401re microcontroller can found in User Manual (UM1724).

The STM32 Nucleo-64 boards based on the MB1136 reference board (NUCLEO-F401RE and NUCLEO-F411RE) provide an affordable and flexible way for developer to try out new concepts and build prototypes with the STM32 microcontrollers in the LQFP64 package, choosing from the various combinations of performance, power consumption, and features. The ARDUINO Uno V3 connectivity support and the ST morpho headers provide an easy means of expanding the functionality of the Nucleo open development

platform with a wide choice of specialized shields. The STM32 Nucleo boards do not require any separate probe as they integrate the ST-LINK/V2-1 debugger and programmer. The STM32 Nucleo boards come with the comprehensive free software libraries and examples available with the STM32Cube MCU Packages, as well as direct access to the Arm® Mbed™ online resources at http://mbed.org/.

From the User Manual, developer can locate all ports and their respective pins ready to connect external devices such as sensors, actuators, LCD, etc. Moreover, the document shows build-in LED of the board and user button.

LED

The tricolor LED (green, orange, red) LD1 (COM) provides information about ST-LINK communication status. LD1 default color is red. LD1 turns to green to indicate that communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

- Slow blinking Red/Off: at power-on before USB initialization
- Fast blinking Red/Off: after the first correct communication between the PC and ST-LINK/V2-1 (enumeration)
- Red LED On: when the initialization between the PC and ST-LINK/V2-1 is complete
- Green LED On: after a successful target communication initialization
- Blinking Red/Green: during communication with the target
- Green On: communication finished and successful
- Orange On: Communication failure

User LD2: the green LED is a user LED connected to ARDUINO signal D13 corresponding to STM32 I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Refer to *Table 11* to *Table 23* when:

- the I/O is HIGH value, the LED is on
- the I/O is LOW, the LED is off

LD3 PWR: the red LED indicates that the STM32 part is powered and +5V power is available.

Push-buttons

B1 USER: the user button is connected to the I/O PC13 (pin 2) of the STM32 microcontroller.

B2 RESET: this push-button is connected to NRST, and is used to RESET the STM32 microcontroller.

4. Setting up header file

The header file STM32f4xx.h is not automatically when you are using STM32CubeIDE. However, developer has to download header file from the official website of STMicroelectronics (www.st.com) by searching STM32CubeF4. STM32Cube MCU Package for STM32F4 series (HAL, Low-Layer APIs and CMSIS, USB, TCP/IP, File system, RTOS, Graphic - and examples running on ST boards). STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards. It also comprises the STM32CubeF4 MCU Package composed of the STM32Cube hardware abstraction layer (HAL) and the low-layer (LL) APIs, plus a consistent set of middleware components (RTOS, USB, FAT file system, Graphics and TCP/IP). TouchGFX graphic software stack is also included in the STM32CubeF4 MCU Package as a part of the STM32Cube ecosystem. It is available free of charge for production and redistribution on STM32 microcontrollers. All embedded software utilities are delivered with a full set of examples running on STMicroelectronics boards.

All features

- Consistent and complete embedded software offer that frees the user from dependency issues
- Maximized portability between all STM32 Series supported by STM32Cube
- Hundreds of examples for easy understanding
- High quality HAL and low-layer API drivers using CodeSonar® static analysis tool
- TouchGFX graphics software stack
- STM32F4-dedicated middleware including USB Host and Device, and TCP/IP

- Free user-friendly license terms
- Update mechanism that can be enabled by the user to be notified of new releases

5. Developing the GPIO output driver

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection register (GPIOx_AFRH and GPIOx_AFRL).

GPIO main features

• Up to 16 I/Os under control

• Output states: push-pull or open drain + pull-up/down

• Output data from output data register (GPIOx_ODR) or peripheral (alternate function output)

• Speed selection for each I/O

• Input states: floating, pull-up/down, analog

• Input data to input data register (GPIOx_IDR) or peripheral (alternate function input)

• Bit set and reset register (GPIOx_BSRR) for bitwise write access to GPIOx_ODR

• Locking mechanism (GPIOx_LCKR) provided to freeze the I/O configuration

• Analog function

• Alternate function input/output selection registers (at most 16 AFs per I/O)

• Fast toggle capable of changing every two clock cycles

• Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

Steps to develop GPIO output driver to blink LED

Step 1. Create STM32 project in STM32CubeIDE

Step 2. Create a blank .c file (main.c)

Step 3. Include STM32f4xx.h header file and write main function and infity loop inside using while(1) {}

Step 4. Locate user LED in PA5 (pin 5 of port A)

Step 5. Set direction of pin as output: **GPIO port mode register (GPIOx_MODER) (x = A..E and H)**

Step 6. GPIO port output type register (GPIOx_OTYPER) (x = A..E and H)

Step 7. GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..E and H)

Step 8. GPIO port output data register (GPIOx_ODR) (x = A..E and H)

Step 9. GPIO port bit set/reset register (GPIOx_BSRR) (x = A..E and H). This register can used as an alternative to ODR.

Step 10. Set the time delay between the action i.e. ON and OFF.

```
#include "stm32f4xx.h"

#define PIN_LED (1U<<5)
#define GPIOEN        (1<<0)
int main() {
      /*1. Enable clock access to GPIOA*/
      RCC->AHB1ENR |= GPIOEN;
      /*2. Configure PA5 as an output*/
      GPIOA->MODER |=  (1U<<10);
      GPIOA->MODER &= ~(1U<<11);

      while(1) {
            /*3. Blink LED*/
            GPIOA->ODR |= PIN_LED;
            for(int delay = 0; delay < 1000000; delay++){}
            GPIOA->ODR &= ~PIN_LED;
            for(int delay = 0; delay < 1000000; delay++){}
      }
}
```

Step 2. Create

6. Developing the GPIO input driver for push button

The same steps from above will be used and incorperating the IDR register (**GPIO port input data register (GPIOx_IDR) (x = A..E and H)**).

```
#include "stm32f4xx.h"

#define GPIOAEN             (1U<<0)
#define GPIOCEN             (1U<<2)

#define PIN5                (1U<<5)
#define PIN13               (1U<<13)

#define LED_PIN             PIN5
#define BTN_PIN             PIN13

int main(void)
{

      /*Enable clock access to GPIOA and GPIOC*/
      RCC->AHB1ENR |=GPIOAEN;
      RCC->AHB1ENR |=GPIOCEN;

    /*Set PA5 as output pin*/
      GPIOA->MODER |=(1U<<10);
      GPIOA->MODER &=~(1U<<11);

      /*Set PC13 as input pin*/
      GPIOC->MODER &=~(1U<<26);
      GPIOC->MODER &=~(1U<<27);

      while(1)
      {

            /*Check if BTN is pressed*/
            if(GPIOC->IDR & BTN_PIN)
            {
                  /*Turn on led*/
            GPIOA->BSRR = LED_PIN;
            }
            else{
                  /*Turn off led*/
            GPIOA->BSRR = (1U<<21);
            }

      }
}
```

7. Developing the UART transmitting and receiving Drivers

Universal synchronous asynchronous receiver transmitter (USART)

The universal synchronous asynchronous receiver transmitter (USART) offers a flexible means of full-duplex data exchange with external equipment requiring an industry standard NRZ asynchronous serial data format. The USART offers a very wide range of baud rates using a fractional baud rate generator.

It supports synchronous one-way communication and half-duplex single wire

communication. It also supports the LIN (local interconnection network), Smartcard Protocol and IrDA (infrared data association) SIR ENDEC specifications, and modem operations (CTS/RTS). It allows multiprocessor communication.

High speed data communication is possible by using the DMA for multibuffer configuration.

USART functional description

The interface is externally connected to another device by three pins (see *Figure 167*). Any USART bidirectional communication requires a minimum of two pins: Receive Data In (RX) and Transmit Data Out (TX):

RX: Receive Data Input is the serial data input. Oversampling techniques are used for data recovery by discriminating between valid incoming data and noise.

TX: Transmit Data Output. When the transmitter is disabled, the output pin returns to its I/O port configuration. When the transmitter is enabled and nothing is to be transmitted, the TX pin is at high level. In single-wire and smartcard modes, this I/O is used to transmit and receive the data (at USART level, data are then received on SW_RX).

Through these pins, serial data is transmitted and received in normal USART mode as

frames comprising:

- An Idle Line prior to transmission or reception
- A start bit
- A data word (8 or 9 bits) least significant bit first
- 0.5,1, 1.5, 2 Stop bits indicating that the frame is complete
- This interface uses a fractional baud rate generator - with a 12-bit mantissa and 4-bit fraction
- A status register (USART_SR)

- Data Register (USART_DR)
- A baud rate register (USART_BRR) - 12-bit mantissa and 4-bit fraction.
- A Guardtime Register (USART_GTPR) in case of Smartcard mode.

Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M-bit status.

When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

Procedure:

(a). Enable the USART by writing the UE bit in USART_CR1 register to 1.
(b). Program the M bit in USART_CR1 to define the word length.
(c). Program the number of stop bits in USART_CR2.
(d). Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
(e). Select the desired baud rate using the USART_BRR register.
(f). Set the TE bit in USART_CR1 to send an idle frame as first transmission.
(g). Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
(h). After writing the last data into the USART_DR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

Receiver

During an USART reception, data shifts in the least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

Procedure:

(a). Enable the USART by writing the UE bit in USART_CR1 register to 1.
(b). Program the M bit in USART_CR1 to define the word length.
(c). Program the number of stop bits in USART_CR2.

(d). Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication. STEP 3

(e). Select the desired baud rate using the baud rate register USART_BRR

(f). Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

When a character is received

- The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
- An interrupt is generated if the RXNEIE bit is set.
- The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
- In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to the Data Register.
- In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be cleared before the end of the reception of the next character to avoid an overrun error.

8. Analog-to-digital converter (ADC)

The 12-bit ADC is a successive approximation analog-to-digital converter. It has up to 19 multiplexed channels allowing it to measure signals from 16 external sources, two internal sources, and the VBAT channel. The A/D conversion of the channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored into a leftor right-aligned 16-bit data register. The analog watchdog feature allows the application to detect if the input voltage goes beyond the user-defined, higher or lower thresholds.

ADC main features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- Interrupt generation at the end of conversion, end of injected conversion, and in case of analog watchdog or overrun events
- Single and continuous conversion modes
- Scan mode for automatic conversion of channel 0 to channel ‘n’

- Data alignment with in-built data coherency
- Channel-wise programmable sampling time
- External trigger option with configurable polarity for both regular and injected conversions
- Discontinuous mode
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed
- ADC input range: VREF– ≤ VIN ≤ VREF+
- DMA request generation during regular channel conversion

9. Developing System Tick Timer Driver

- Found in all ARM Cortex Microcontrollers regardless of the Silicon manufacturer.
- Used for taking actions periodically.

Often used as time-base for real-time operating systems

- The Systick is a **24-bit down counter** driven by the processor clock.
- A **24-bit down counter** is used to work as a system timer in STM32F4 ARM Cortex M4 Microcontroller.
- The down-counting starts from a preloaded/set value using Control Registers of the SystTick Timer.
- The rate of decrement depends on the system clock frequency.
- Maximum value that can be loaded to the Load Register of the System Timer is $2^{24} - 1$.
- It can generate an interrupt when the value reaches zero. The value of the counter decrements on every positive edge of the clock cycle.
- The NVIC manages interrupts generated by the SysTick and transfers to the control of CPU to related interrupts service routine.
- System timer can measure the time elapsed. By using this feature we can generate precise delays.

- We can execute tasks periodically such as periodic polling and CPU scheduling.
- System timer can be used to generate delays at the regular time interval.
- It can be used to generate delays between two events.
- It can be used to call periodic tasks after a specific time, such as RTOS tick timer.
- To implement time-shared scheduling in RTOS.
- To generate SysTick interrupts at a fixed time interval.

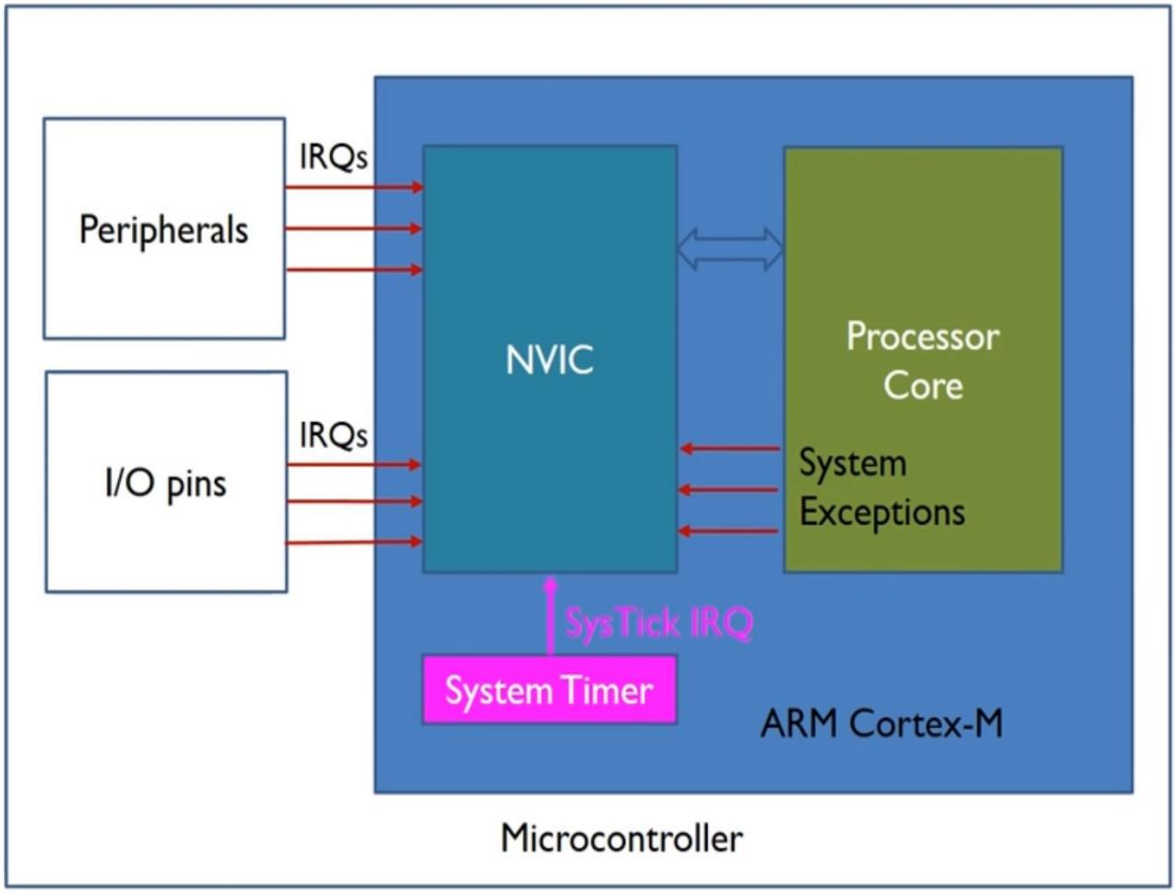

Fig 3. System Tick Timer for ARM Cortex-M processor

Systick registers

- Systick Current Value Register (STCVR)
  - This register contains the count value
- Systick Control & Status Register (STCSR)

- This register allows us to configure the systick clock source, enable/disable interrupts and enable/disable the systick counter

- Systick Reload Value Register (STRVR)
  - This is where the initial count value is placed

Systick Count Value computations

- Compute the delay achieved by **loading 10** in the Systick Reload Value Register (STRVR) given system clock = 16MHz
- *Written as*

» ***Systick -> LOAD = 9***

» ***Written in the CMSIS standard, we write 9 instead of 10 because the counter starts counting from 0***

Solution

System clock = 16MHz = 16 000 000 cycles/second

If 1 second executes 16 000 000 cycles

How many seconds execute 1 cycle?

$\Rightarrow \frac{1}{16000000} = \mathbf{62.5ns} = \mathbf{62.5x10^{-9}s}$

$\Rightarrow$ ***Then 10 cycles*** **=>** $\mathbf{10x62.5x10^{-9}s = 625x10^{-9}s = 625ns}$

System Clock (SYSCLK) is chosen as clock source

If:

$$Systick \rightarrow LOAD = N$$

$$Delay\ achieved = Nx\frac{1}{SYSCLK} = \frac{N}{SYSCLK}$$

Compute **N** value for achieving a **1 ms** delay given **SYSCLK** as **16MHz**

Solution

$$1ms = 0.001s$$

$$Delay = \frac{N}{SYSCLK}$$

$$0.001 = \frac{N}{16000000}$$

$$N = 0.001x16000000$$

$$N = 16000$$

10.Other areas in the plan of the ESE laboratory

The following are the areas also covered in the laboratory using STM32F401re and STM32F411re microcontrollers.

A. Developing the General Purpose Timer driver
B. Developing the Timer Output Compare driver
C. Developing GPIO Interrupt driver
D. Developing the UART transmitter and receiver interrupt drivers
E. Developing the ADC interrupt driver
F. Developing the Systick interrupt driver
G. SPI, I2C etc.