UNIVERSITI TEKNOLOGI MALAYSIA

Blinky project

Milistone 4

SEMESTER2
ACADEMIC SESSION 2021/2022

GROUP MEMBER:

Name:	Matric No:
Hawraa Khaleel Makki Fajo	MKE191186
Alaa Kadhem mukheef	MKE191113

ABSTRACT

The world becomes digital and everything transferred to be monitored along with the Internet and to be controlled using programable devices called embedded systems. In some cases, these systems work on time so they can perform real-time applications. These systems can be considered as computer systems that monitor, respond to, or control an external environment. They connected to the environment using sensors and/or other input-output interfaces. These systems can be used in many fields such as military, medical, communication, and educational as in this project. Challenges of using embedded systems are how to meet the environmental conditions concerning the timing requirements and the real-time behavior how to choose the best-embedded devices to be used for a certain application and how to use the suitable sensor connected with the embedded device. Because of this, this report aims to provide a full understanding of how to use embedded system devices to perform some applications. The report represents the use of the STM32F446 microcontroller as an embedded system capable to be used with Internet-of-Things (IoT) applications. It is evaluated using flashing the internal LED to be familiar with the toolkit. Simulation results show that the STM32F446 is very useful to be used in a wide range of applications due to its efficient functionality and its applicability of programming and integration with other devices.

I. Introduction

An embedded system is a microprocessor-based computer hardware system with software that is designed to perform a dedicated function, either as an independent system or as a part of a large system. At the core is an integrated circuit designed to carry out computation for real-time operations (Belleza&De Freitas 2018). Complexities range from a single microcontroller to a suite of processors with connected peripherals and networks; from no user interface to complex graphical user interfaces. The complexity of an embedded system varies significantly depending on the task for which it is designed. Embedded system applications range from digital watches and microwaves to hybrid vehicles and avionics. As much as 98 percent of all microprocessors manufactured are used in embedded systems (Buysse et al. 2021).

Embedded systems are managed by microcontrollers or digital signal processors (DSP), Application-Specific Integrated Circuits (ASIC), Field-Programmable Gate Arrays (FPGA), and gate arrays. These processing systems are integrated with components dedicated to handling electric and/or mechanical interfacing (Huo et al. 2021). Embedded systems programming instructions, referred to as firmware, are stored in read-only memory or flash memory chips, running with limited computer hardware resources. Embedded systems connect with the outside world through peripherals that link input and output devices (Koppermann et al. 2018).

The basic structure of an embedded system includes the following components:

- Sensors: The sensor measures and converts the physical quantity to an electrical signal, which can then be read by an embedded systems engineer or any electronic instrument. A sensor stores the measured quantity in the memory.
- Analog-to-Digital Conversion (ADC): An ADC converts the analog signal sent by the sensor into a digital signal. This allows the embedded system to understand sensors data.
- Processor: Processors assess the data to measure the output and store it in the memory (Xiao 2018).
- Digital-to-Analog Converter (DAC): A DAC changes the digital data fed by the processor to analog data to be understandable with the physical environment.
- Actuators: An actuator compares the output given by the DAC to the actual output stored and stores the approved output.

The industry for embedded systems is expected to continue growing rapidly, driven by the continued development of Artificial Intelligence (AI), Virtual Reality (VR) and Augmented Reality (AR), machine learning, deep learning, and the Internet of Things (IoT) (Lee&Shin 2020). The cognitive embedded system will be at the heart of such trends as reduced energy consumption, improved security for embedded devices, cloud connectivity, and mesh networking, deep learning applications, and visualization tools with real-time data (Shirley et al. 2021).

This project aims to have a full practical simulation of the use of an Embedded System especially the use of STM32F446. The objective of this project can be as:

- Exploring and experiencing the integration of the Embedded Systems.
- Exploring and understanding the use of the STM32F446 toolkit.

- Exploring and understanding the LED blinking application on the STM32F446 discovery board.

The remainder of this report is section II which describes the hardware system design of the STM32F446. Section III describes the work procedure of this project. Section IV represents the simulation results and discussion. Section V talks about the challenges behind this project. Finally, section VI concludes the work and gives some ideas for future development.

II. Hardware System Design

The STM32 Nucleo board as Shown in Figure 1 provides an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller line, choosing from the various combinations of performance, power consumption, and features. The Arduino™ connectivity support and ST Morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger/programmer (Sioutis&Tan 2019).

It leverages ST's proprietary ART Accelerator™, smart architecture, advanced Flash technology, and its embedded ARM Cortex-M4 core to achieve the leading performance of 225 DMIPS and 608 CoreMark® at 180MHz executing from embedded Flash (Villarroel et al. 2019).

Efficient simultaneous communication via multiple interfaces enables smarter and more interactive industrial, scientific, medical, and Internet-of-Things (IoT) applications, while the advanced process technology, together with dynamic voltage scaling, extensive clock gating, and flexible sleep modes allow for significant power savings.

- Performance: At 180 MHz, the STM32F446 delivers 225 DMIPS/608 CoreMark performance executing from Flash memory, with 0-wait states thanks to ST's ART Accelerator. The DSP instructions and the floating-point unit enlarge the range of addressable applications.
- Power efficiency: ST's 90 nm process, ART Accelerator, and the dynamic power scaling enable the current consumption in run mode and executing from Flash memory to be as low as 200 µA/MHz at 180 MHz. In Stop mode, the power consumption is 50 µA typical.
- Integration:
  - 2 dedicated audio PLL, SPDIF input, 3 half-duplex, and 2 serial audio interfaces (SAI) supporting full-duplex as well as Time Division Multiplex (TDM) mode.
  - Up to 20 communication interfaces (including 4x USARTs plus 2x UARTs running at up to 11.25 Mbit/s, 4x SPI running at up to 45 Mbit/s, 3x I²C with a new optional digital filter capability, 2x CAN, SDIO, HDMI CEC, and camera interface).
  - Two 12-bit DACs, three 12-bit ADCs reaching 2.4 MSPS or 7.2 MSPS in interleaved mode Up to 17 timers: 16- and 32-bit running at up to 180 MHz.
  - Easily extendable memory range using the flexible 90 MHz memory controller with a 32-bit parallel interface, and supporting Compact Flash, SRAM, PSRAM, NOR, NAND, and SDRAM memories.
  - Cost-effective NOR flash extension thanks to the 90MHz Dual quadSPI interface supporting memory-mapped mode

The STM32F446 product line provides from 256-Kbyte to 512-Kbyte Flash, 128-Kbyte SRAM and from 64 to 144 pins in packages as small as 3.85 x 3.728 mm.

Figure 1 STM32F446 toolkit (St 2021)

The NUCLEO-F446RE development board enables anyone to take advantage of the STM32F446 features smoothly and flexibly with unlimited expansion capabilities through Arduino™ and Morpho connectors. Developers benefit from STM32Cube firmware, direct support from a large ecosystem of IDE partners, and access to the Arduino and ARM® mbed™ online communities (Sioutis et al. 2019). This intuitive board gives easy access to the STM32F446 high performance at 225 DMIPS. With up to 512 Kbytes of Flash memory and 128 Kbytes of SRAM, Dual Quad-SPI interface, SPDIF, and SAI audio connectivity, the Nucleo-F446RE enables a wide diversity of interactive applications.

Its features can be summarized as:

- Two types of extension resources
  - Arduino Uno Revision 3 connectivity
  - STMicroelectronics Morpho extension pin headers for full access to all STM32 I/Os
- On-board ST-LINK/V2-1 debugger/programmer with SWD connector
  - Selection-mode switch to using the kit as a standalone ST-LINK/V2-1
- Flexible board power supply
  - USB VBUS or external source (3.3 V, 5 V, 7 - 12 V)
  - Power management access point
- User LED (LD2)
- Two pushbuttons: USER and RESET
- USB re-enumeration capability: three different interfaces supported on USB
  - Virtual Com port
  - Mass storage (USB Disk drive) for drag-drop programming
  - Debug port

The STM32 Nucleo boards are the official Development Boards from STMicroelectronics. It features the ARM Cortex M4 32-bit which is in the LQFP64 package. The Boards pinout is similar to Arduino UNO and has many other additional pins to expand performance. This board also comes with an integrated ST-LINK/V2-1 programmer and debugger; hence it is very easy to get started with this board as shown in Figure 2.

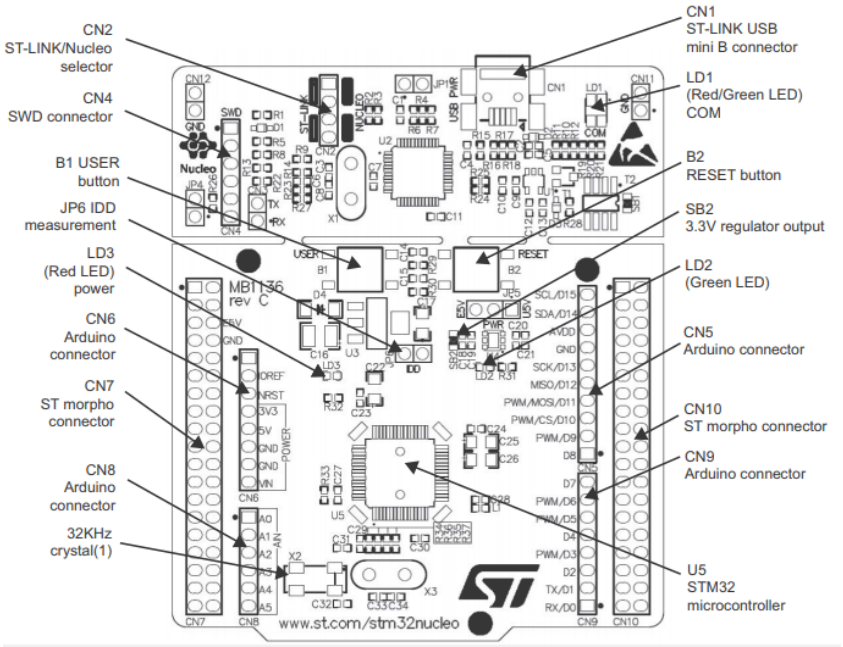

Figure 2 STM32F446 pins description (Components 2018)

As shown in Figure 2, there are three LEDs, where LD1 is for indicating USB communication, LD2 is a programmable LED, and LD3 indicates power. Similarly, there are two push buttons where one is user programmable, and the other is to reset the Microcontroller. The Board operates with a 3.3V supply but a wide voltage range of 7-12V can be provided to the VIN pin since it has an onboard voltage regulator.

There are two sets of pins. The pin one resembles the Arduino UNO and the blue one is the STM32 style. The Arduino-like pins are female connector pins that exactly match the order and position of Arduino UNO pins and hence any Arduino shield can be used with these development boards.

The Arduino pins are split into categories CN5, CN6, CN8, and CN9. Each category pin can be tabulated as in Table 1.

Table 1: Arduino pins category

Pin Category	**Pin Type**	**Pin names**	**Description**
CN6	Power	IOREF	3.3V Reference

			Voltage pin
RESET	Resets the Microcontroller		
+3.3V	Provides 3.3V as output can also be used to power the MCU		
+5V	5V output only pin		
GND	Ground pins		
CN8	Analog Pins and I2C	A0-A1	Used to measure Analog Voltage
A4 and A5	These pins can also be used for I2C Communication A4 is SDA and A5 is SCL		
CN5	Digital Pins and SPI	D8-D15	Digital GPIO pins
AVDD	Can be used to provide an analog reference voltage		
GND	Ground Pins		
D13, D12. D11 and D10	Acts as SCK, MISO, MOSI, and CS pins respectively for SPI communication		
CN9	Digital Pins and USART	D0 to D7	Digital GPIO pins
D0 and D1	Acts as Rx and Tx pins respectively for USART communication		

Apart from the Arduino pins, the board also has 76 (38+38) GPIO pins as male headers on either side of the board as shown above. These pins are classified into CN7 and CN10 with each having 38 Pins. They comprise GPIO pins, Analog Pins, Timer Pins, and Power pins. The name of the pins can be found in Figure 2. They are also categorized in Table 2.

Table 2: GPIO pins discription

Pin Category	**Pin Type**	**Pin names**	**Description**
CN7	Port pins	PC0, PC1, PC2, PC3, PC10, PC11, PC12, PC13, PC14, PC15	Port C digital I/O pins
PD2	Port D I/O pin		
PA0, PA1, PA4, PA13, PA14, PA15	Port A I/O pins		
PB7, PB8, and PB9	Port B I/O pin		
PH0 and PH1	Port H I/O pins		
Power	VBAT	Can be used to power the module from the battery	
+3.3V	Provides 3.3V as output can also be used to power the MCU		
+5V	5V output only pin		

VIN	The unregulated input power pin		
RESET	Resets the MCU		
IOREF	Reference Voltage Pin		
CN10	Port Pins	PC4, PC5, PC6, PC7, PC8, PC9	Port C I/O Pins
PA2, PA3, PA4, PA6, PA7, PA10, PA11 and PA12	Port A I/O Pins		
PB1, PB2, PB3, PB4, PB5, PB6, PB8, PB9, PB10, PB12, PB14, PB15	Port B I/O Pins		
Power	U5V	5V Power Pin	
GND	System Ground of the MCU		
AGND	Analog Ground Pin		

In our work, the LED blinking application on the STM32F446 discovery board has been simulated as shown in Figure 3. STM32F446 Discovery board has onboard LEDs. We are using Keil IDE to build the LED blinking application. STM32F446 Discovery board has an on-board ST-link debugger.

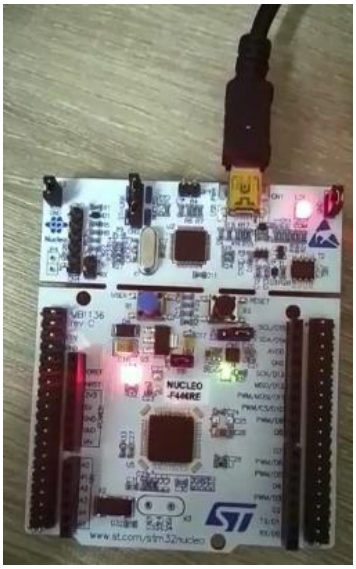

Figure 3 Real Hardware used

Keil MDK is the most comprehensive software development solution for the various STM32 microcontroller families and provides everything you need for creating, building, and debugging embedded applications (Verslype et al. 2020). MDK includes the genuine Arm Compiler and the easy-to-use Keil uVision IDE/Debugger that interfaces to STM32CubeMX and

Software Packs. MDK also offers various professional middleware components as shown in Figure 4.

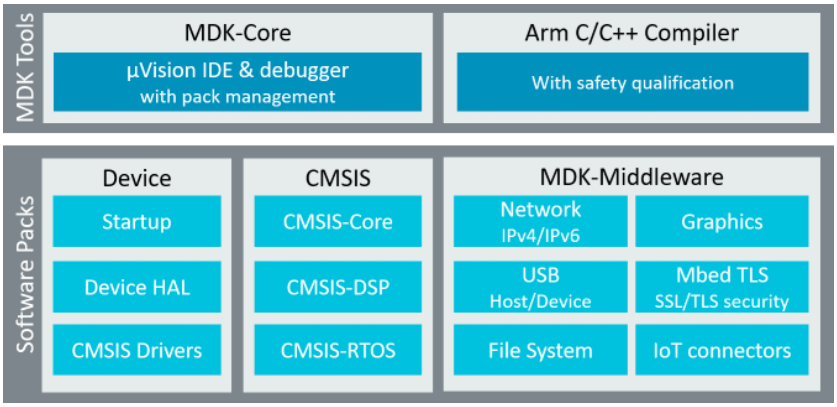

Figure 4 Keil MDK middleware components (Mdk 2021)

Figure 5 shows the functionality description of the Keil IDE main window that is used to design our work.

- The Project window shows application source files and selected software components. Below the components, you will find corresponding library and configuration files. Projects support multiple targets. They ease configuration management and may be used to generate debug and release builds or adoptions for different hardware platforms.
- The Manage Run-Time Environment window shows all software components that are compatible with the selected device. Inter-dependencies of software components are identified with validation messages.
- The Configuration Wizard is an integrated editor utility for generating GUI-like configuration controls in assembler, C/C++, or initialization files.
- The Functions window gives fast access to the functions in each C/C++ source code module.
- The Code Completion list and Function Parameter information help you to keep track of symbols, functions, and parameters.
- Dynamic Syntax Checking validates the program syntax while you are typing and provides real-time alerts to potential code violations before compilation.

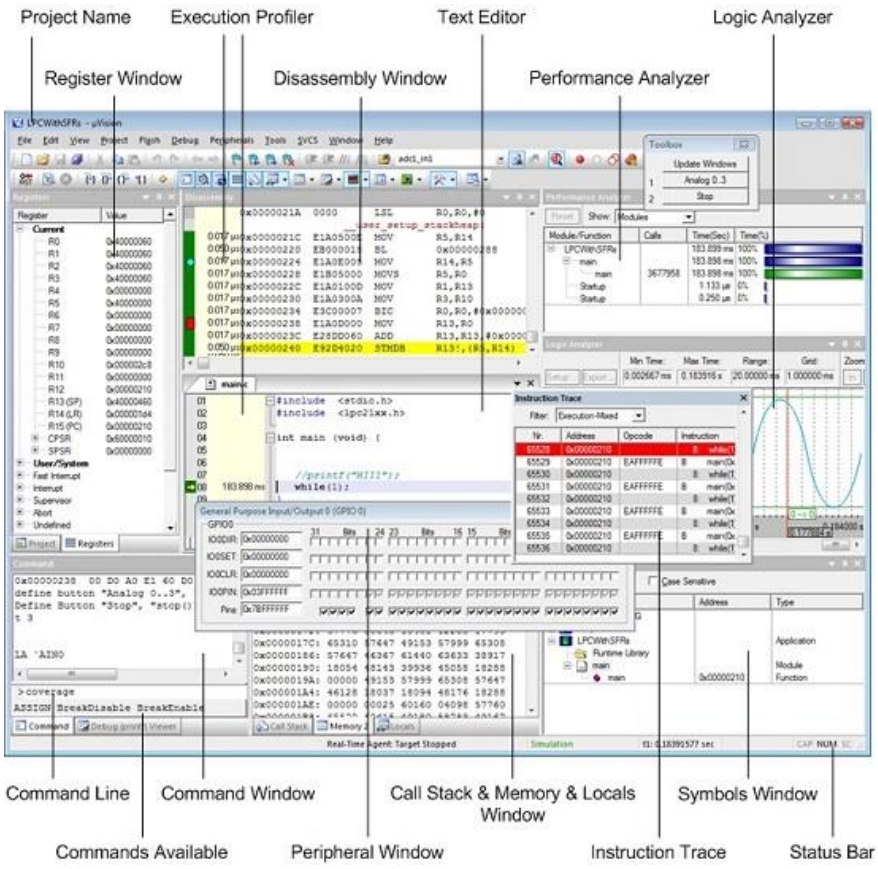

Figure 5 Keil IDE main window (Mdk 2021)

III. Work Procedures

As discussed in section II. The technical specification of the toolkit used can be summarized as in Table 3.

Table 3: Technical Specifications

Architecture	ARM Cortex M4 CPU with FPU
Power consumption	2.4uA at standby without RTC
CPU Frequency	84 MHz
Crystal Oscillator Range	4 to 26 MHz
MCU Operating Voltage (VDD)	1.7V to 3.6V
Board Operating Voltage (VIN)	7V to 15V
Flash Memory	512KB
SRAM	96 KB
GPIO Pins	50
ADC	12-bit 16Channel
RTC	In-built 32kHz with calibration
Timers	16-bit (6)

	32-bit (2)
Watchdog Timers	2
USART/UART Communication	4
I2C Communication	3
SPI Communication	3
USB2.0 Support	Yes
Internal Crystal Oscillator	Yes, 16MHz
On Board Debugger	Yes, Serial Wire and JTAG

Figure 6 shows the block diagram of the work. It consists of the use of the STM32F446 toolkit to flash the internal LED of the toolkit.

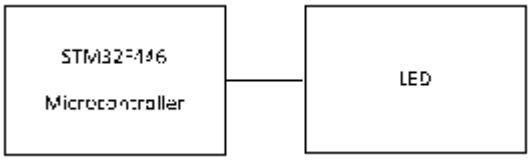

Figure 6 Block diagram

To open Keil software, click on the start menu, then program, and then select keil2. Figure 7 window will appear on your screen.

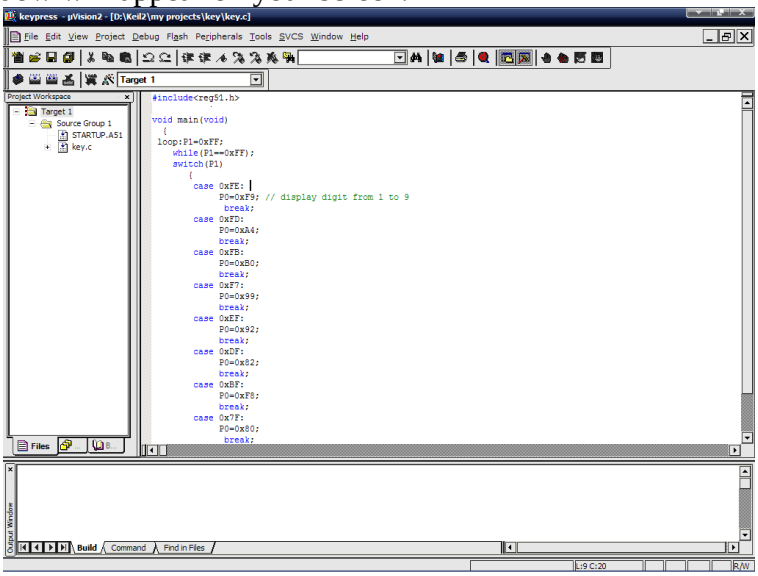

Figure 7 main window of Keil IDE

Starting new project as:

- Click on the Project menu and select new project
- you will be asked to create a new project in a specific directory
- just move to your desired directory and there create a new folder for your project.

After saving the new project, follow the following procedures:

- Chose your target device for which you want to write the program.
- Scroll down the cursor and select STM32F446 as shown in Figure 8.

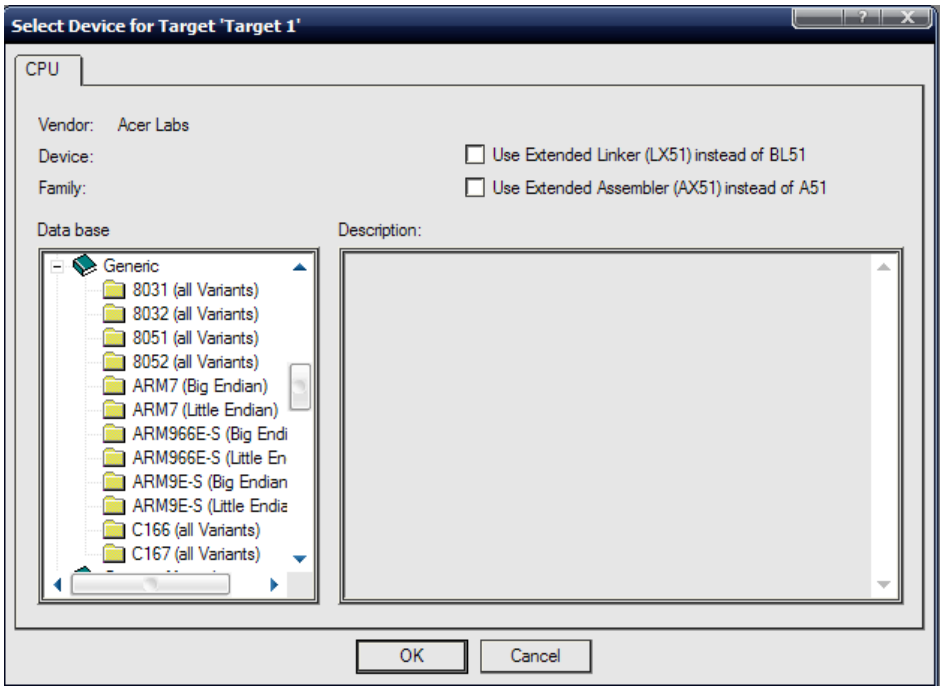

Figure 8 Selecting target device on Keil IDE

When you click OK in Figure 8, you will be asked to add startup code and file to your project folder. click yes. Now on your screen expand the target1 list fully. Now click on the file menu and select a new file. an editor window will open. Now you can start writing your code.

- As you start writing the program in C, same way here also you have to first include the header file. Because our target is stm32 our header file will be “stm32f4xx_it.h”
- after including this file. just right click on the file and select open document < stm32f4xx_it.h >.
- Now we can write our program the same as c language starting with void main()
- After completing the code save the file in the project folder with the “.c” extension.
- If you want to see the output on ports go to the peripheral menu and select I/O ports. select the desired port. you can give input to port pins by checking or unchecking any check box.

The structure of our code to flash the internal LED of the toolkit is separated into three sections. The first one is the GPIO configuration of the toolkit as shown in Figure 9. The second section is the description of all functions used in the total code as shown in Figure 10. The third section is the main code for flashing the internal LED of the toolkit as shown in Figure 11.

GPIO Configuration

__HAL_RCC_GPIOA_CLK_ENABLE () – This function enables the clock for the PORTA.

typedef struct{

uint32_t Pin; uint32_t Mode; uint32_t Pull; uint32_t Speed; uint32_t Alternate; }GPIO_InitTypeDef;

The above structure is GPIO configuration structure, which is used to configure the GPIO pins.

/*Configure GPIO pin : PA5 */

GPIO_InitStruct.Pin = GPIO_PIN_5;

GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;

GPIO_InitStruct.Pull = GPIO_NOPULL;

GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

The GPIO pin configured in Push pull output mode.

Figure 9 GPIO configuration code

Figure 10 shows the functions used in the total code.

Function Definitions

SystemClock_Config() – This function is used to generate the System clock frequency 180 MHz.

HAL_GPIO_Init() – This function is used to configured the GPIO pin as per user requested configuration.

HAL_GPIO_WritePin() – This function is used to set the pin voltage to HIGH or LOW (3.3V or 0V).

Delay() – This function is used to provide delay. This function uses, systick timer flags to provide the delay.

Figure 10 Functions used

Figure 11 shows the main function of the code that is used to flash the internal LED of the toolkit.

Application Code

```
int main(void)
{
  /* MCU Configuration----------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */   HAL_Init();
  /* Configure the system clock to 180 MHz */
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* Infinite loop to run blink application */   while (1)
  {
/* Turn On the LED Pin */
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
        Delay(100); // 100 ms delay
        /* Turn Off the LED Pin */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
        Delay(100); // 100 ms delay  } }
```

Figure 11 main code

IV. Simulation Results and Discussion

This project aims to deal with the STM32F446 toolkit. the basic work is to flash the internal LED of the toolkit using the code available in the appendix. LED2 is the output of the work. Figure 12 shows the OFF status of the LED when starting the work.

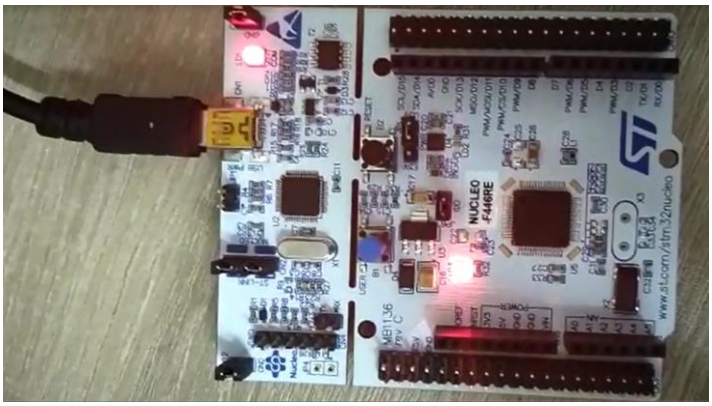

Figure 12 LED2 is OFF

Figure 13 shows when the LED2 status changed to ON.

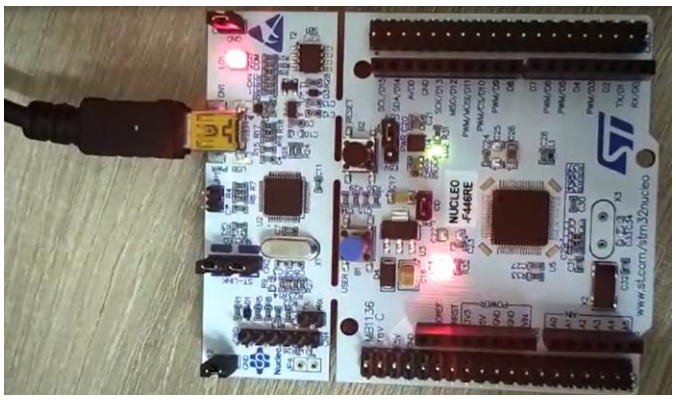

Figure 13 LED2 is ON

Figure 14 shows when LED2 returns into OFF again.

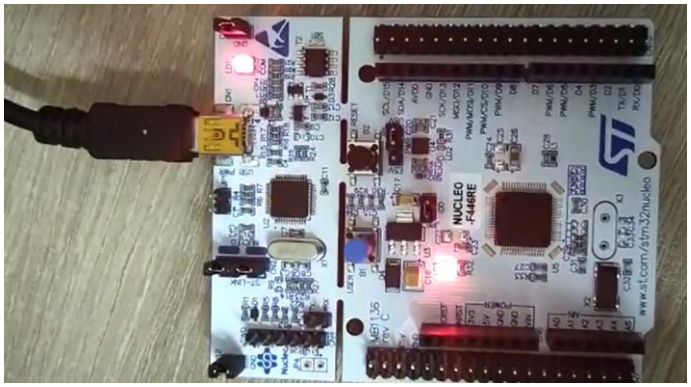

Figure 14 LED2 is OFF again

V. Key challenges

Many challenges are facing this project. It is requiring a good knowledge of Embedded Systems applications and a solid background of how to choose the best hardware devices to be used. At first, we face some difficulties in choosing the Embedded System board Another challenge comes from the coding knowledge which requires having a full overview of all the hardware requirements and operations. This takes some time to be familiar with. Figure 15 can summarize the key challenges of this project.

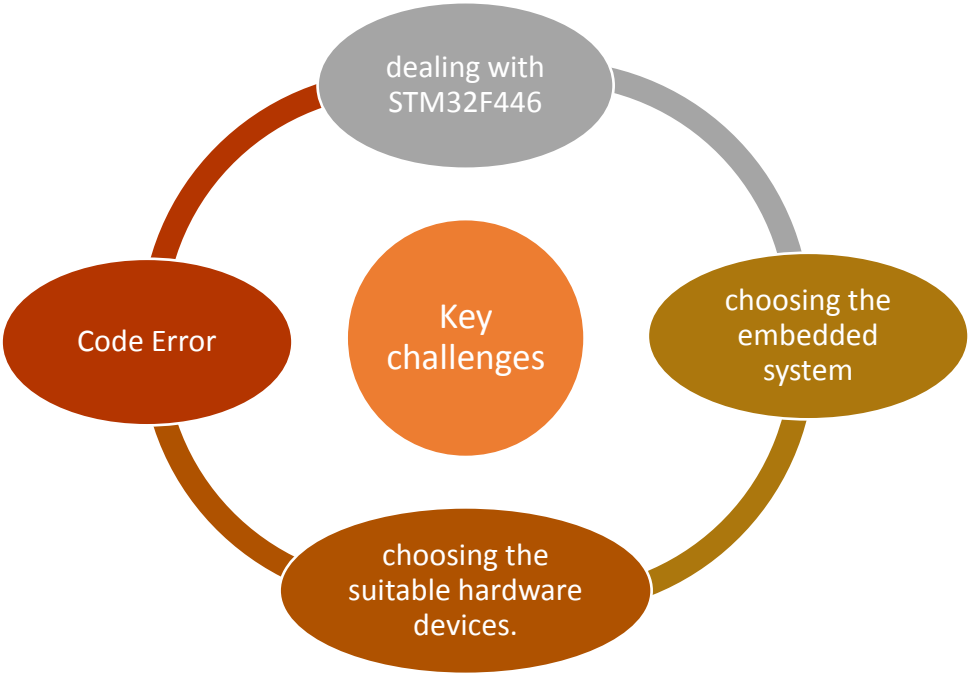

Figure 15 Key challenges

VI. Conclusion

Through the information collected and the experiences that have been done on this project, the piece of STM32F446 is a wide world of experiments and projects that can be used in all fields and various disciplines. As the goal of this project was LED blinking, the experiment was successfully implemented.

Appendix:

```
 /* USER CODE BEGIN Header */
/**
******************************************************************************
* @file : main.c
* @brief : Main program body
******************************************************************************
* @attention
*
* Copyright (c) 2021 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
******************************************************************************
*/
/* USER CODE END Header */
```

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void Delay(uint32_t u32_lDelayInms);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
/**
* @brief The application entry point.
* @retval int
*/
int main(void)
{

/* MCU Configuration--------------------------------------------------------*/
/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();
/* Configure the system clock to 180 MHz */
SystemClock_Config();
/* Initialize all configured peripherals */
MX_GPIO_Init();
/* Infinite loop to run blink application */
while (1)
{
/* Turn On the LED Pin */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
Delay(100); // 100 ms delay
/* Turn Off the LED Pin */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
Delay(100); // 100 ms delay
}
}
/**
* @brief Delay in milli seconds
* @retval None
*/

void Delay(uint32_t u32_lDelayInms)
{
/* Reset the 1ms counter, it will increment in systick handler */
u8_g1msCounter = 0;
while(u8_g1msCounter <= u32_lDelayInms);
}
/**
```

```
* @brief System Clock Configuration
* @retval None
*/
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = (Wireless);
/** Configure the main internal regulator output voltage
*/
__HAL_RCC_PWR_CLK_ENABLE();
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
/** Initializes the RCC Oscillators according to the specified parameters
* in the RCC_OscInitTypeDef structure.
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 4;
RCC_OscInitStruct.PLL.PLLN = 180;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 2;
RCC_OscInitStruct.PLL.PLLR = 2;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
Error_Handler();
}
/** Activate the Over-Drive mode
*/
if (HAL_PWREx_EnableOverDrive() != HAL_OK)
{
Error_Handler();
}
/** Initializes the CPU, AHB and APB buses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
{
Error_Handler();
}
}

/**
* @brief GPIO Initialization Function
* @param None
* @retval None
*/
static void MX_GPIO_Init(void)
{
GPIO_InitTypeDef GPIO_InitStruct = {0};
/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
```

```
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET);
/*Configure GPIO pin : PA5 */
GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pin : PB13 */
GPIO_InitStruct.Pin = GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
/**
* @brief This function is executed in case of error occurrence.
* @retval None
*/
void Error_Handler(void)
{
/* USER CODE BEGIN Error_Handler_Debug */
/* User can add his own implementation to report the HAL error return state */
__disable_irq();
while (1)
{
}
/* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
* @brief Reports the name of the source file and the source line number
* where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None
*/
void assert_failed(uint8_t *file, uint32_t line)
{
/* USER CODE BEGIN 6 */
/* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
/* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Belleza, R. R. & De Freitas, E. P. 2018. Performance Study of Real-Time Operating Systems for Internet of Things Devices. *IET Software* 12(3): 176-182.

Buysse, L., Van Den Broucke, Q., Verslype, S., Peuteman, J., Boydens, J. & Pissoort, D. 2021. Fpga-Based Digital Twins of Microcontroller Peripherals for Verification of Embedded Software in a Distance Learning Environment. *2021 XXX International Scientific Conference Electronics (ET)*, hlm. 1-4.

Components. 2018. Stm32 Nucleo F401re Development Board https://components101.com/microcontrollers/stm32-nucleo-f401re-pinout-datasheet

Huo, D., Cao, C., Liu, P., Wang, Y., Li, M. & Xu, Z. 2021. Commercial Hypervisor-Based Task Sandboxing Mechanisms Are Unsecured? But We Can Fix It! *Journal of Systems Architecture* 116(102114.

Koppermann, P., Pop, E., Heyszl, J. & Sigl, G. 2018. 18 Seconds to Key Exchange: Limitations of Supersingular Isogeny Diffie-Hellman on Embedded Devices. *IACR Cryptol. ePrint Arch.* 2018(932.

Lee, S.-C. & Shin, D. 2020. Tcp/Ip Using Minimal Resources in Iot Systems. *Journal of the Korea Society of Computer and Information* 25(10): 125-133.

Mdk, A. K. 2021. C/C++ Compiler, Ide/Debugger, Cmsis, Rtos, Middleware for Stm32 https://www.st.com/en/partner-products-and-services/arm-keil-mdk.html

Shirley, D., Sundari, V. K., Sheeba, T. B. & Rani, S. S. 2021. Analysis of Iot-Enabled Intelligent Detection and Prevention System for Drunken and Juvenile Drive Classification. Dlm. (pnyt.). *Automotive Embedded Systems,* hlm. 183-200. Springer.

Sioutis, M. & Tan, Y. 2019. Open Source Implementation of Htip for Embedded Devices. *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, hlm. 106-110.

Sioutis, M., Tan, Y., Oura, K., Nakagawa, J. & Ryuichi, N. 2019. Deployment and Evaluation of Elite, an Open Source Implementation of Echonet Lite for Micro-Controllers. *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, hlm. 794-798.

St. 2021. Stm32 Nucleo-64 Board. https://www.st.com/en/evaluation-tools/nucleo-f446re.html

Verslype, S., Buysse, L., Peuteman, J., Pissoort, D. & Boydens, J. 2020. Remote Laboratory Setup for Software Verification in an Embedded Systems and Mechatronics Course. *2020 XXIX International Scientific Conference Electronics (ET)*, hlm. 1-4.

Villarroel, A., Zurita, G. & Velarde, R. 2019. Development of a Low-Cost Vibration Measurement System for Industrial Applications. *Machines* 7(1): 12.

Wireless, G. 2019. 60ghz Wireless: The Ieee 802.11ad Standard. https://www.60ghz-wireless.com/60ghz-technology/60ghz-wireless-the-ieee-802-11ad-standard/

Xiao, P. 2018. *Designing Embedded Systems and the Internet of Things (Iot) with the Arm Mbed.* John Wiley & Sons.