April,2023 Semester

ICT 6641
Advanced Embedded System Design

Lecture#2:
Getting Started With STM32 MCUs

S. M. Lutful Kabir, *PhD*
Professor, BUET

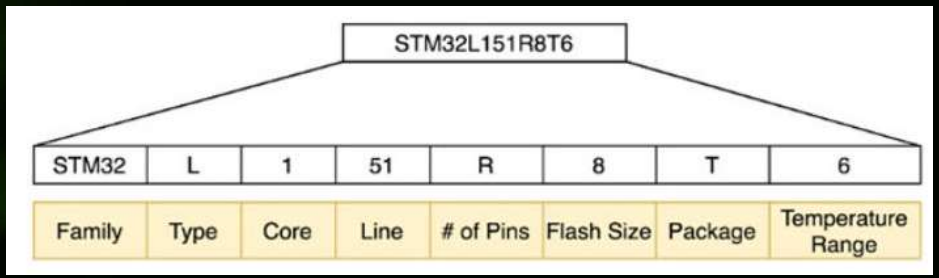
A Comparison between STM32 and AVR IC

Features	STM32F103	ATMEGA328
Clock Frequency	72 Mhz	16 Mhz
I2C Buses	2	1
SPI Buses	2	1
CAN Bus	Yes	No
Analog Channel	10	8
PWM Channel	15	6
USART Buses	3	1
GPIO's	32	24
On Board RTC	Yes	No
Architecture	ARM Cortex M3 32 bit	AVR RISC 8 bit
ADC Resolution	12 bit	10 bit
Quantization Level	4096	1024
Flash Memory	64KB	32KB
SRAM	20KB	2KB
Debugging	Serial, JTAG	Serial
PWM Resolution	16 bit	10bit
Price	**110**	**115**

STM32 ARM®-Based Microcontrollers

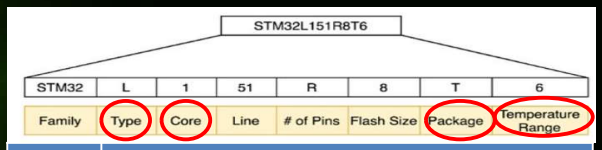
- The STM32 series of microcontrollers are one of the most popular ICs among the 32-Bit microcontrollers.
- STMicroelectronics provides multiple of product lines for the STM32 parts. There are low-power, mainstream, and high-performance product lines.
- And a more application-specific wide variety of parts that enables you to pick the right part for your project.
- There are low-cost FullSpeed USB solutions, CAN, LIN, Ethernet, DCMI (Digital Camera Memory Interface), and CryptoEngine for cryptographic applications, and much more powerful peripherals.
- Both digital and analog such as ADCs, DAC, OPamp, Comparators, etc.

Naming Convention of STM32 part numbers



https://www.digikey.com/en/maker/blogs/2020/understanding-stm32-naming-conventions

Naming Convention of STM32 part numbers (continued)



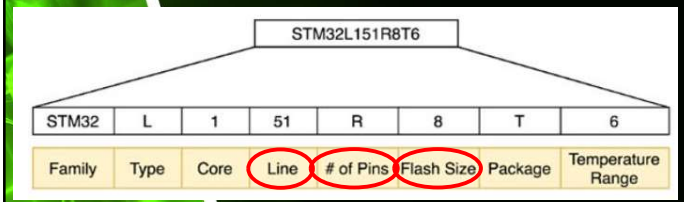
Type	
F	Foundation; Sometimes: High Performance (for example, STM32F2 SERIES)
G	Mainstream
L	Low-power
H	High-Performance
W	Wireless

Core	
0	ARM Cortex M0
1	ARM Cortex M3
2	ARM Cortex M3
3	ARM Cortex M4
4	ARM Cortex M4
7	ARM Cortex M7

Temperature	
6	-40^{0}C to 85^{0}C
7	-40^{0}C to 105^{0}C

Package	
P	TSSOP
H	BGA
U	VFQFPN
T	LQFP
Y	WLCSP

Naming Convention of STM32 part numbers (continued)



The two numbers under "line" describe the line of the MCU. The line describes the features of this particular device, for example, the peripherals, and the speed.

Number of pins	
F	20
G	28
K	32
T	36
S	44
C	48
R	64 or 56
V	100
Z	144
I	176

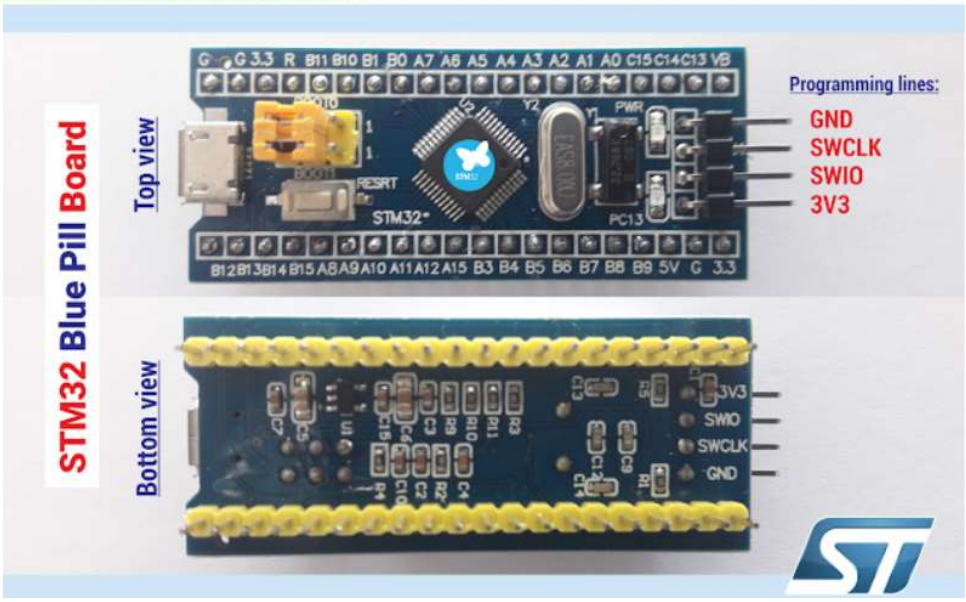
Flash-memory Size	
4	16 Kbyte
6	32 Kbyte
8	64 Kbyte
B	128 Kbyte
C	256 Kbyte
D	384 Kbyte
E	512 Kbyte
F	768 Kbyte
G	1024 KByte
H	1536 KByte
I	2048 KByte

The board that will be used

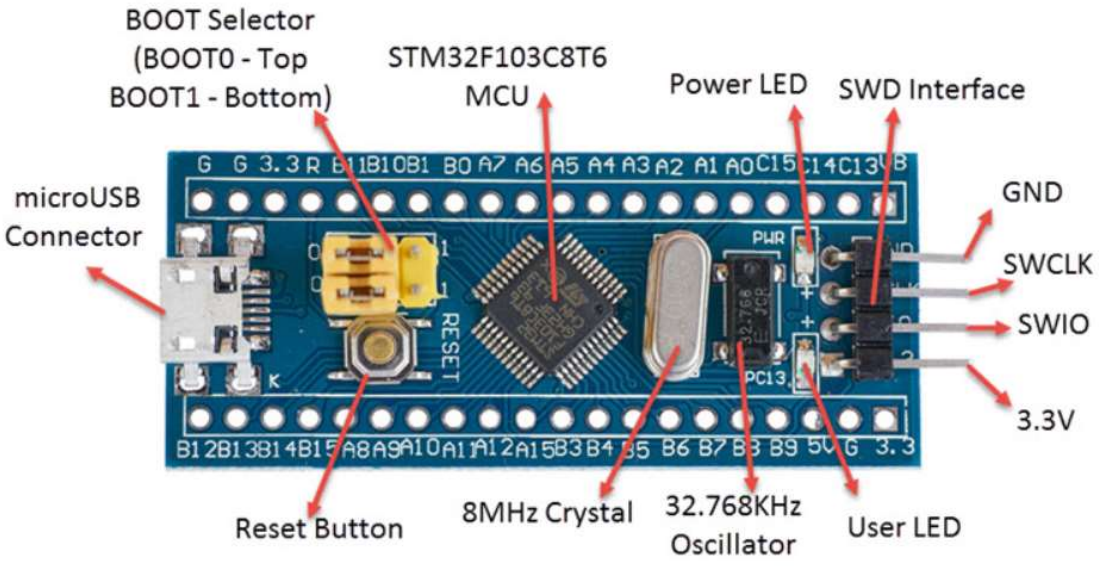
- The primary development board we've selected for this course is Blue Pill.
- The blue pill uses STM32F103C8 microcontroller
- The STM32F103C8 microcontroller has a Cortex-M3 core.
- But, it lacks the FPU and DSP operations which can be a huge miss in certain applications.
- However, it's much cheaper target MCU at the end of the day.
- We can set the desired SYSCLK speed up to 72MHz for F103C8 microcontroller.

STM32 Blue Pill Board

The two sides of STM32 Blue Pill Board



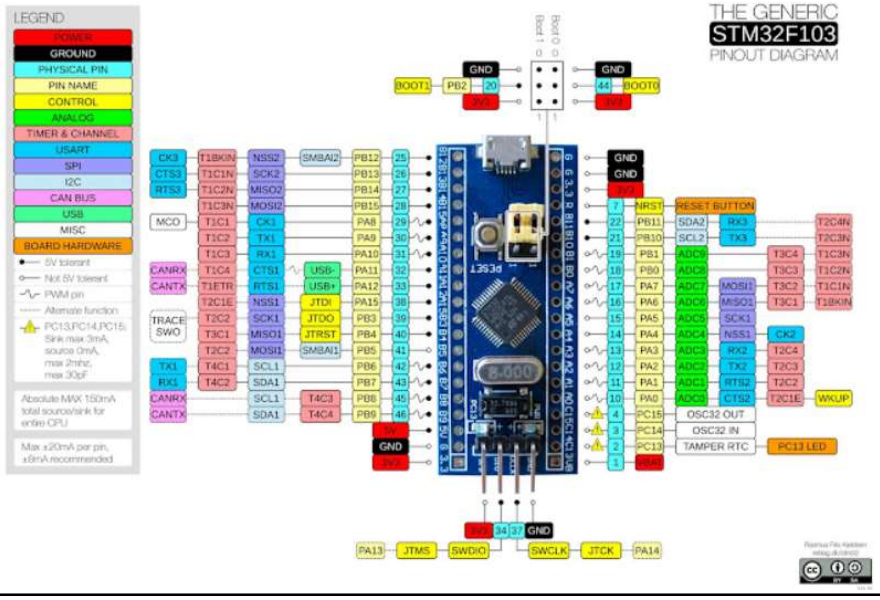
Different Components in the Blue Pill Board



Features of the Different Components

- Features of the different components are as follows:
  - It contains the main MCU – the STM32F103C8T6 in a Quad Flat Package.
  - A Reset Switch – to reset the Microcontroller.
  - microUSB port – for serial communication and power.
  - BOOT Selector Jumpers – BOOT0 and BOOT1 jumpers for selecting the booting memory.
  - Two LEDs – User LED (with PC13) and Power LED.
  - 8 MHz Crystal – Main Clock for MCU.
  - 32.768KHz Oscillator – RTC Clock.
  - SWD Interface – for programming and debugging using ST-Link.
  - 3.3V regulator (on the bottom) – converts 5V to 3.3V for powering the MCU.
- On either long edge of the board, there are pins for connecting various Analog and Digital IO and Power related stuff.

The pin configuration of the board along with their different functions



The Features of STM32F103xx microcontrollers

- The STM32F103xx medium-density performance line family incorporates
  - the high-performance ARM®Cortex®-M3 32-bit RISC core
  - operating at a 72 MHz frequency,
  - Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes.
  - Two 12-bit ADCs,
  - Three general purpose 16-bit timers plus
  - One PWM timer,
  - Two I^2Cs and SPIs, three USARTs, an USB and a CAN.
- The devices operate from a 2.0 to 3.6 V power supply.
- They are available in wide temperature range.
- A comprehensive set of power-saving mode.
- Six different package types: from 36 pins to 100 pins.

All features

- ARM®32-bit Cortex®-M3 CPU Core
  - 72 MHz maximum frequency,1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - 64 or 128 Kbytes of Flash memory
  - 20 Kbytes of SRAM
- Clock, reset and supply management
  - 2.0 to 3.6 V application supply and I/Os
  - POR, PDR, and programmable voltage detector (PVD)
  - 4-to-16 MHz crystal oscillator
  - Internal 8 MHz factory-trimmed RC
  - Internal 40 kHz RC
  - PLL for CPU clock
  - 32 kHz oscillator for RTC with calibration
- Low-power
  - Sleep, Stop and Standby modes
  - VBAT supply for RTC and backup registers
- 2 x 12-bit, 1 µs A/D converters (up to 16 channels)
  - Conversion range: 0 to 3.6 V
  - Dual-sample and hold capability
  - Temperature sensor
- DMA
  - 7-channel DMA controller
  - Peripherals supported: timers, ADC, SPIs, I^2Cs and USARTs
- Up to 80 fast I/O ports
  - 26/37/51/80 I/Os, all mappable on 16 external interrupt vectors and almost all 5 V-tolerant
- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
- 7 timers
  - Three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
  - 16-bit, motor control PWM timer with dead-time generation and emergency stop
  - 2 watchdog timers (Independent and Window)
  - SysTick timer 24-bit downcounter
- Up to 9 communication interfaces
  - Up to 2 x I^2C interfaces (SMBus/PMBus)
  - Up to 3 USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
  - Up to 2 SPIs (18 Mbit/s)
  - CAN interface (2.0B Active)
  - USB 2.0 full-speed interface
- CRC calculation unit, 96-bit unique ID
- Packages are ECOPACK®

See the datasheet for more elaborate description → https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

Applications of STM32F103xx microcontrollers

- These features mentioned in the previous slide make the STM32F103xx medium-density performance line microcontroller family suitable for a wide range of applications such as
  - motor drives,
  - application control,
  - medical and handheld equipment,
  - PC and gaming peripherals,
  - GPS platforms,
  - industrial applications,
  - PLCs,
  - inverters,
  - printers,
  - scanners,
  - alarm systems,
  - video intercoms, and
  - HVACs.

Reference manual for detail use of the features

https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

ARM Cortex-M4+FPU

- The Arm® Cortex®-M4 with FPU processor is the latest generation of Arm® processors for embedded systems.
- It was developed to provide a low-cost platform that meets the needs of MCU implementation, with
  - a reduced pin count,
  - low-power consumption,
  - delivering outstanding computational performance,
  - an advanced response to interrupts.
- The Arm® Cortex®-M4 with FPU 32-bit RISC processor features exceptional code efficiency, delivering the high-performance expected from an Arm® core.
- The processor supports a set of DSP instructions that allow efficient signal processing and complex algorithm execution.

Discussion on the GPIOs

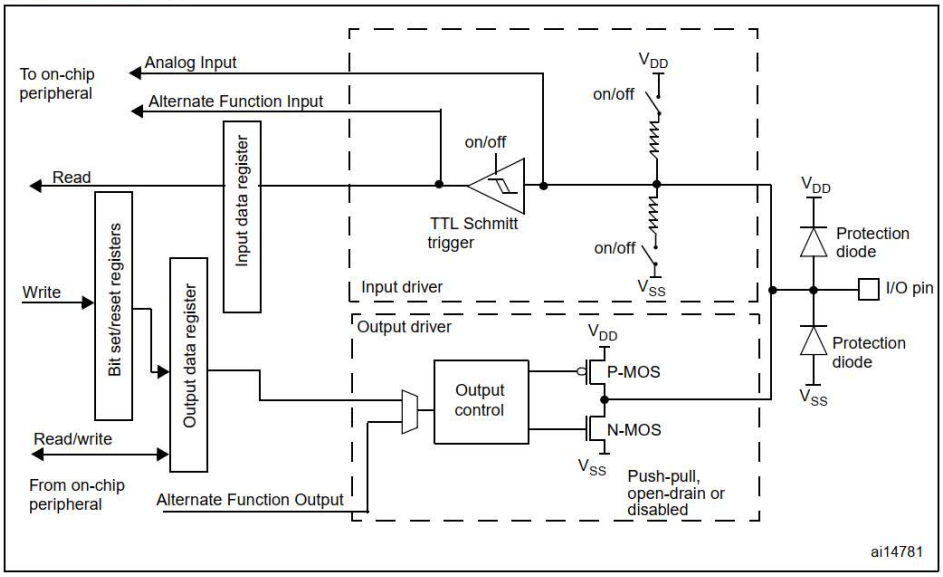
- Each of the general-purpose I/O ports has
  - two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH),
  - two 32-bit data registers (GPIOx_IDR, GPIOx_ODR),
  - a 32-bit set/reset register (GPIOx_BSRR),
  - a 16-bit reset register (GPIOx_BRR) and
  - a 32-bit locking register (GPIOx_LCKR).
- **To understand the use of the above registers, read the reference manual**

Discussion on the Ports
(continued)

- Subject to the specific hardware characteristics of each I/O port listed in the *datasheet*, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:
  - Input floating
  - Input pull-up
  - Input-pull-down
  - Analog
  - Output open-drain
  - Output push-pull
  - Alternate function open-drain
  - Alternate function push-pull

Discussion on the Ports (continued)

- Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed).
- The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers.
- This way, there is no risk that an IRQ occurs between the read and the modify access.



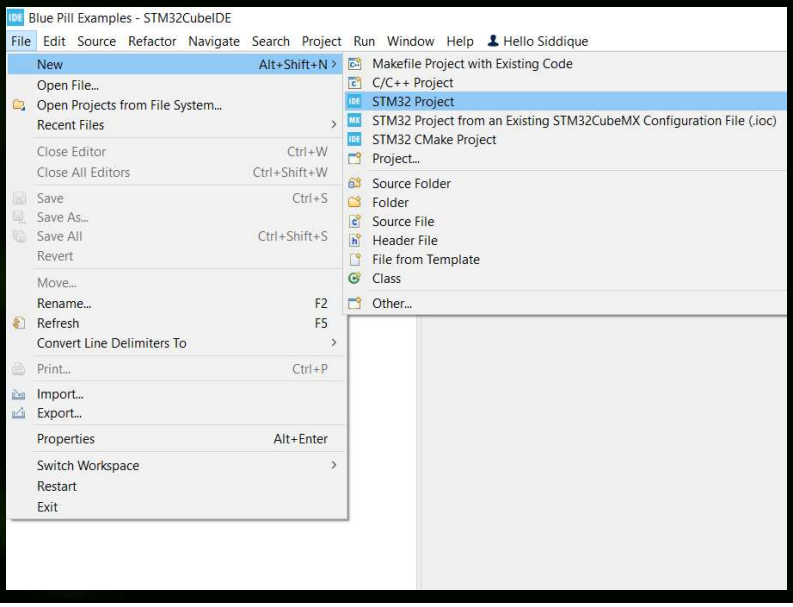
Discussion on the Ports (continued)

- During and just after reset, the alternate functions are not active and the I/O ports are configured in Input Floating mode
- When configured as output, the value written to the Output Data register (GPIOx_ODR) is output on the I/O pin.
- It is possible to use the output driver in Push-Pull mode or Open-Drain mode (only the N-MOS is activated when outputting 0).
- The Input Data register (GPIOx_IDR) captures the data present on the I/O pin at every APB2 clock cycle.
- All GPIO pins have an internal weak pull-up and weak pull-down that can be activated or not when configured as input.

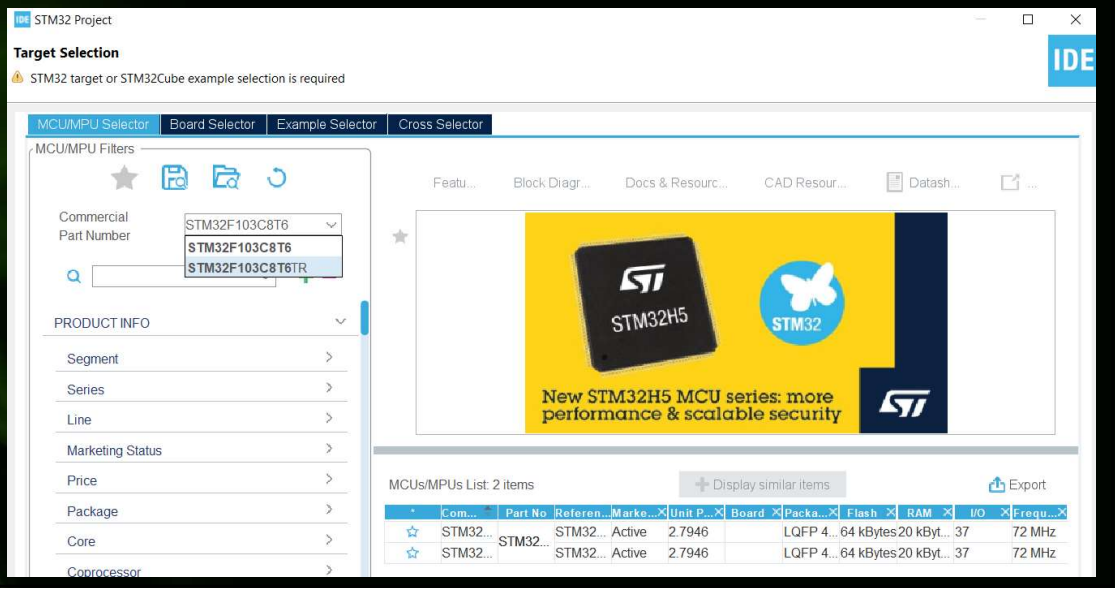
An Example: STM32 GPIO Port as Output

- We'll configure a GPIO pin to be output.
- Then, we'll do the first LED blinking with the STM32 blue pill board.
- You'll learn all the steps to configure STM32 and generate the code.
- Finally you'll compile the program and generate the Hex file

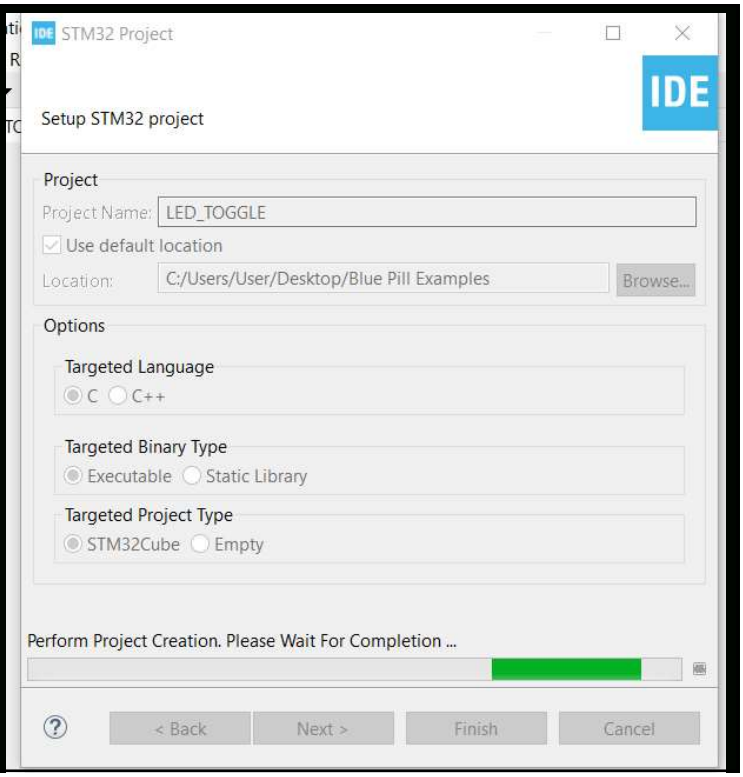
Step#1: Start STM32CubeIDE and Create a new STM32 Project



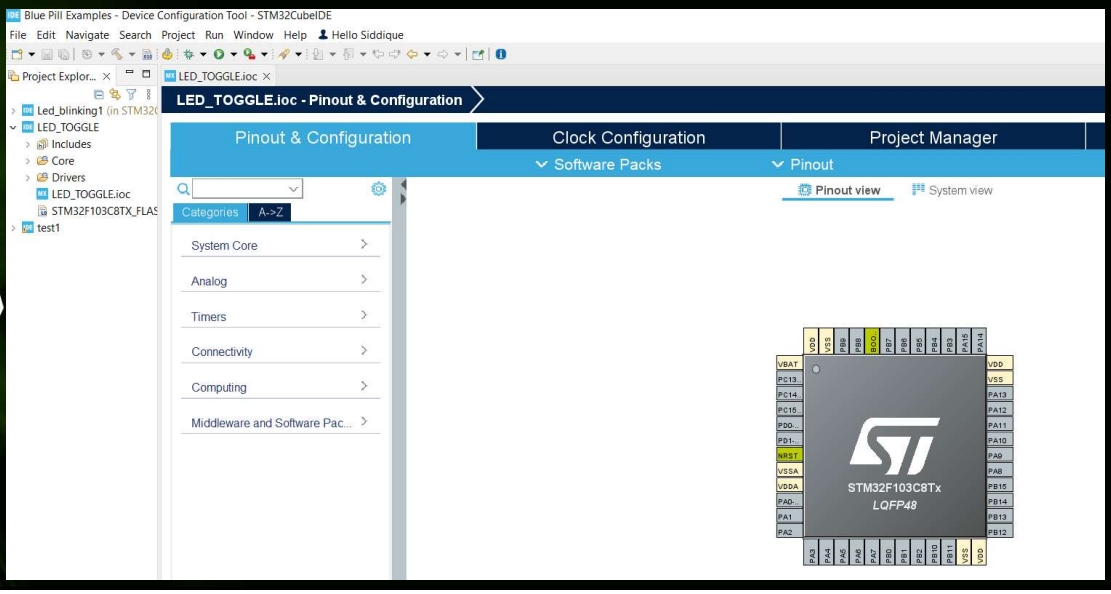
Step#2: Select the IC



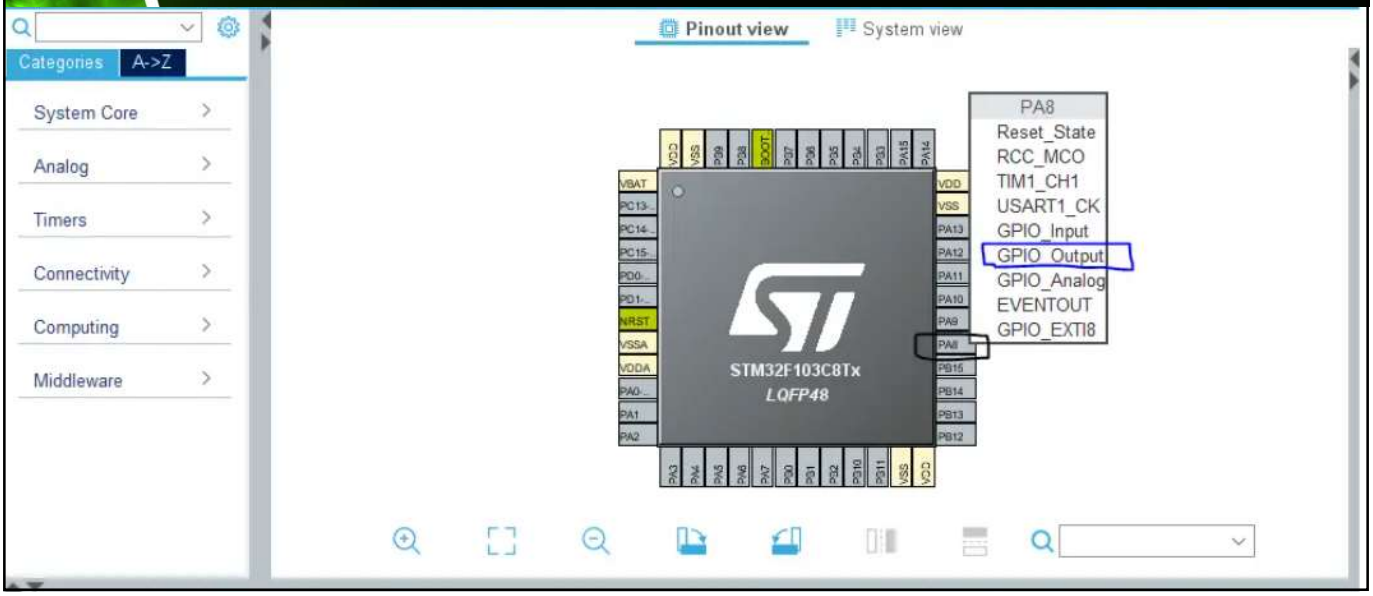
Step#3: Setting up



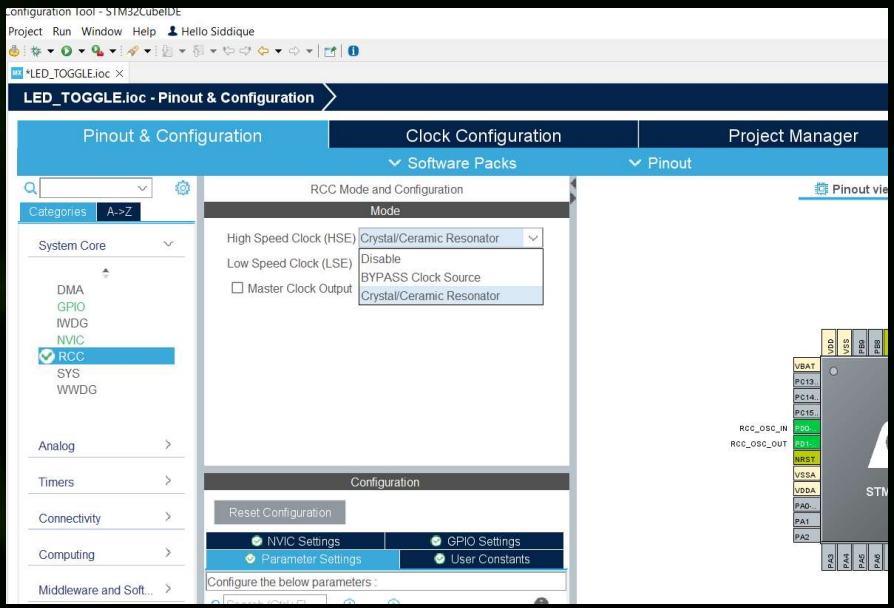
Step#4: Pin Configuration



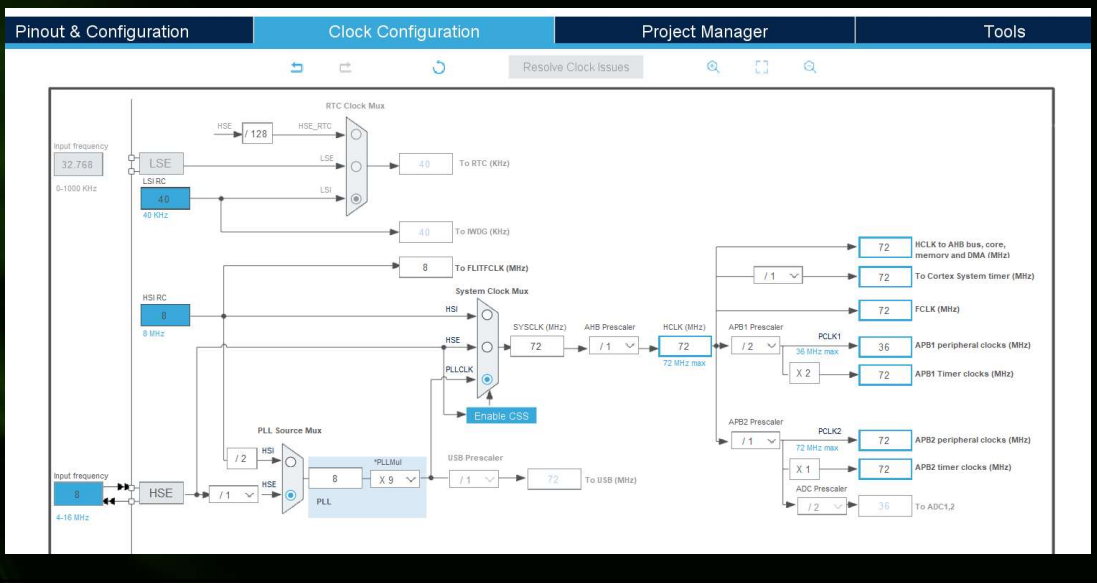
Step#5: Configuration of the Pins



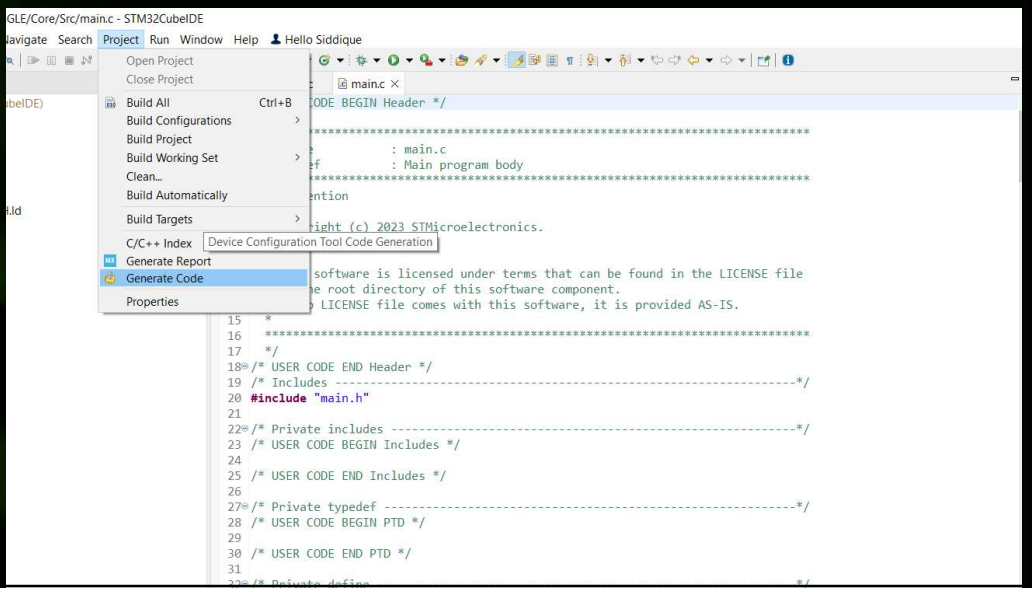
Step#6: Defining the Crystal



Step#7: Setting the Speed



Step#8: Generating the Code



Step#9: Inserting the User Code

```c
#include "main.h"


void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{

  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();

  while (1)
  {

  }

}
```

- Both functions SystemClock_Config() and MX_GPIO_Init() are generated to configure the system clock as we've done in the GUI before and the GPIO pin which we've selected to be an output pin.
- The implementation of both functions is found in the file after the main function
- We call each of them before the main loop while(1) as well as the HAL_Init function.
- The HAL_Init must be called at the beginning of your application.
- Its functionality is clarified in the HAL Documentation.

```c
while (1)
{
  /* USER CODE END WHILE */
        // LED ON
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
        HAL_Delay(100);
        // LED OFF
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
        HAL_Delay(100);
  /* USER CODE BEGIN 3 */
}
```

HAL_Init() function

- HAL_Init(): this function must be called at application startup to
  - initialize data/instruction cache and pre-fetch queue
  - set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
  - call HAL_MspInit() user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). HAL_MspInit() is defined as "weak" empty function in the HAL drivers.
- And most importantly it initializes the SysTick timer, whose ticks are used by the HAL_Delay().
- The SysTick timer is set to tick @ 1000Hz or every 1mSec. So the HAL_Delay function will give you multiples of milliseconds delay.
- HAL_Delay(). this function implements a delay (expressed in milliseconds) using the SysTick timer.
  Care must be taken when using HAL_Delay() since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR.

For explanation of all libraries ➡ https://www.st.com/resource/en/user_manual/um1725-description-of-stm32f4-hal-and-lowlayer-drivers-stmicroelectronics.pdf

HAL functions (continued)

- Besides the delay function, we also need to know the HAL APIs for controlling the GPIO pins.
- To do basic stuff like pin read or write or port read/write, and so on.
- So we'll head over again to the HAL documentation and search for the GPIO chapter, where we'll find this listing for the available APIs.
- The APIs are hyperlinked in the documentation file, so you can click the name of the function to go directly to its detailed description.

IO operation functions

This section contains the following APIs:

- *HAL_GPIO_ReadPin()*
- *HAL_GPIO_WritePin()*
- *HAL_GPIO_TogglePin()*
- *HAL_GPIO_LockPin()*
- *HAL_GPIO_EXTI_IRQHandler()*

- So, let's take a closer look at the GPIO_WritePin() function as we'll be using it as well.

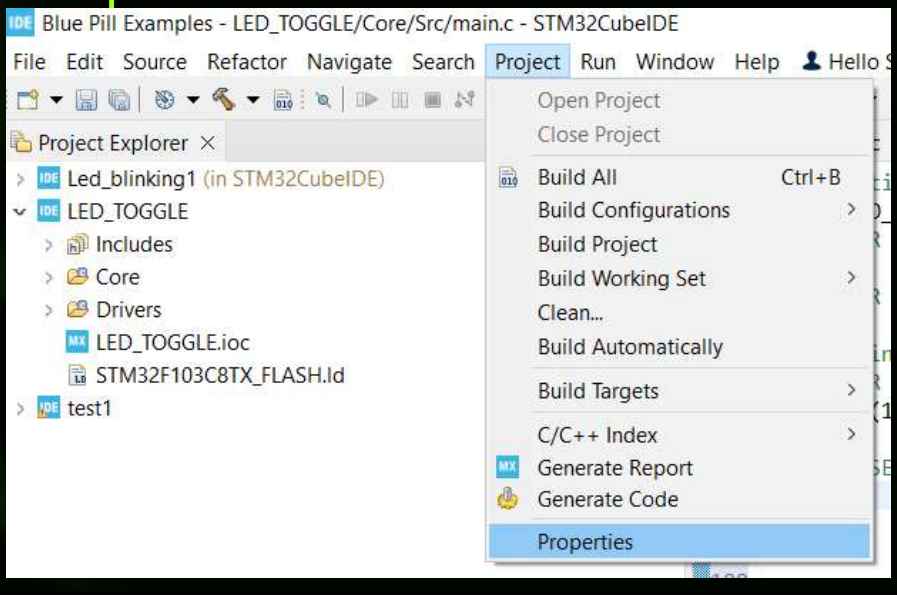
HAL_GPIO_WritePin

Function name	**void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)**
Function description	Set or clear the selected data port bit.
Parameters	• **GPIOx:** where x can be (A..H) to select the GPIO peripheral for STM32L4 family • **GPIO_Pin:** specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).

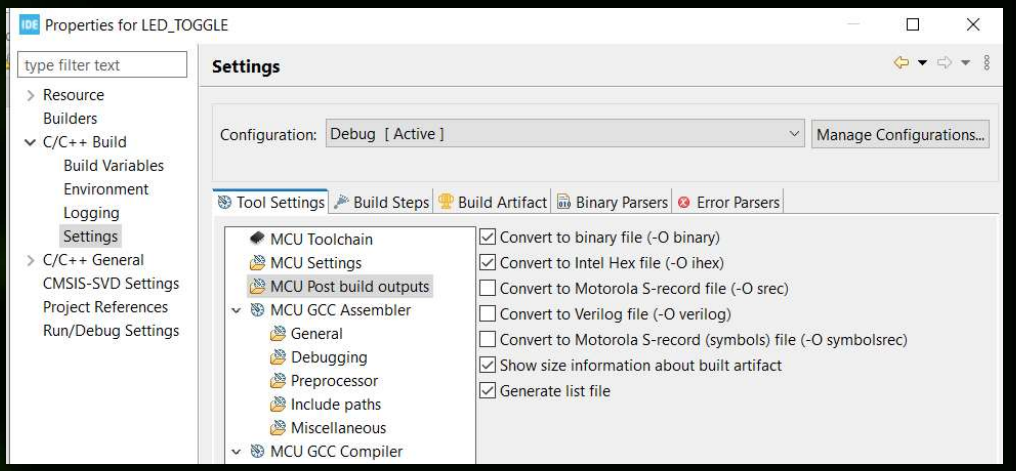
- After reading the documentation and getting familiar with the available APIs, you are ready to go.
- In our toggling LED example, we won't need more than the GPIO_WritePin and HAL_Delay functions.
- And here is the full application code.

```
int main(void)
{

  HAL_Init();

  SystemClock_Config();

  MX_GPIO_Init();

  while (1)
  {
      // LED ON
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
      HAL_Delay(100);
      // LED OFF
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
      HAL_Delay(100);
  }

}
```

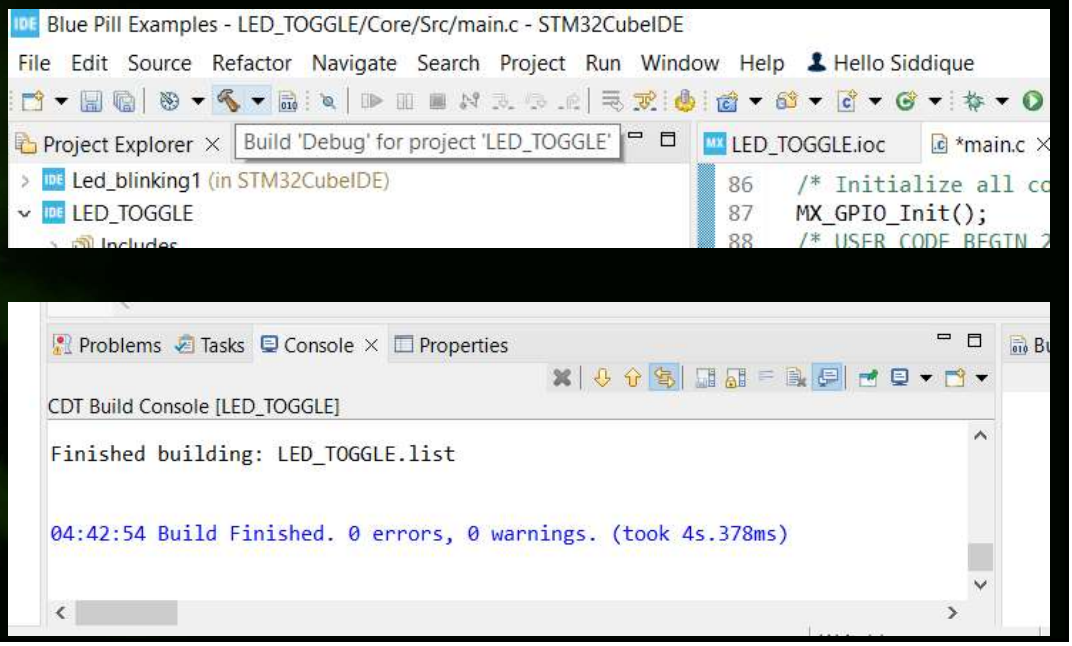
Step#10: Generating binary and hex output-1



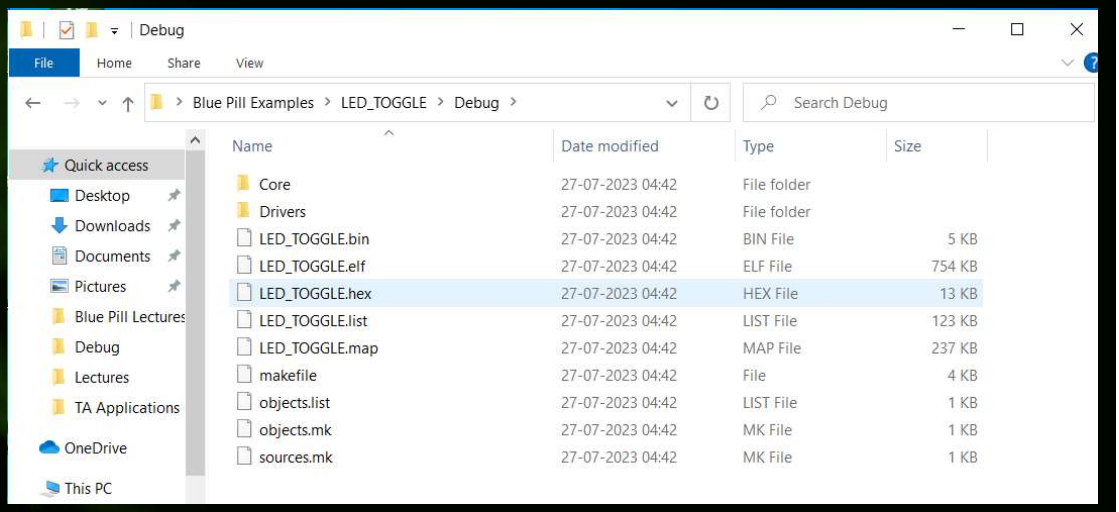
Step#10: Generating binary and hex output-2



Step-11: Building the code

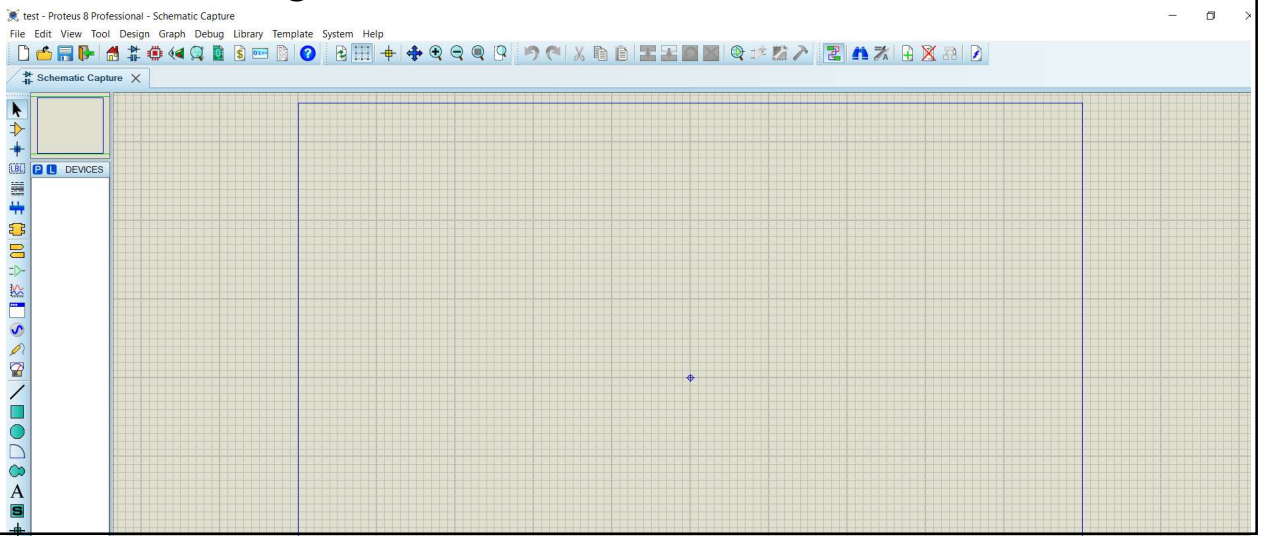


Output files of the process

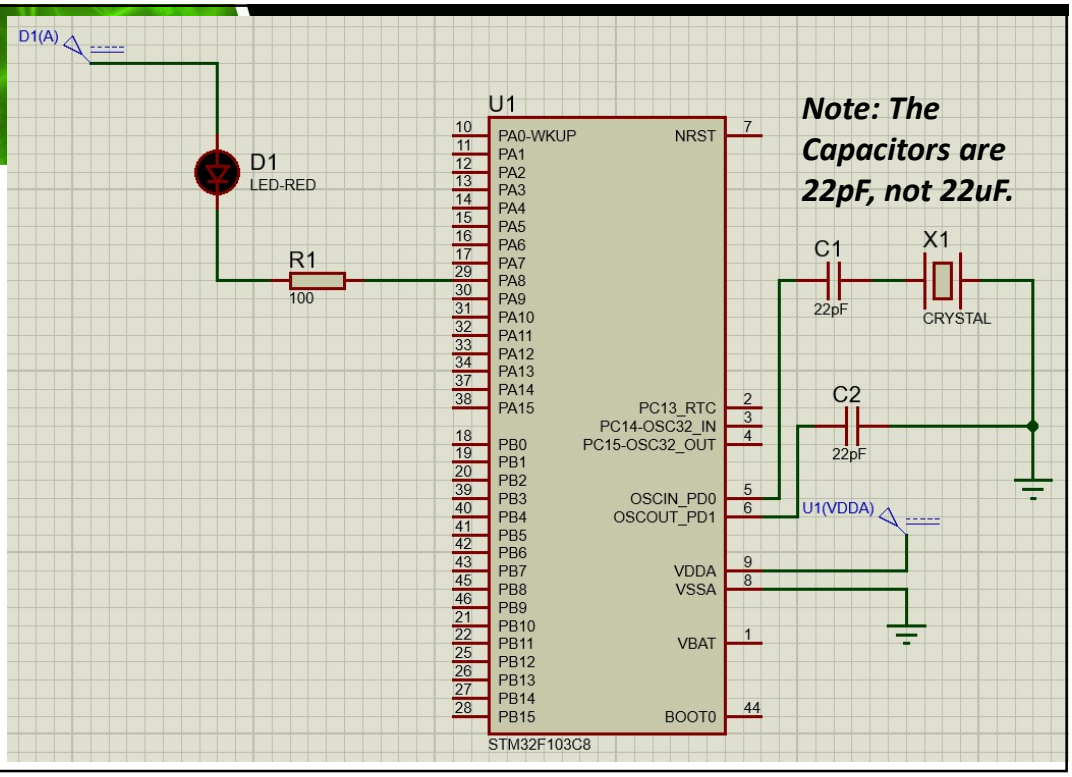


Simulation Using Proteus

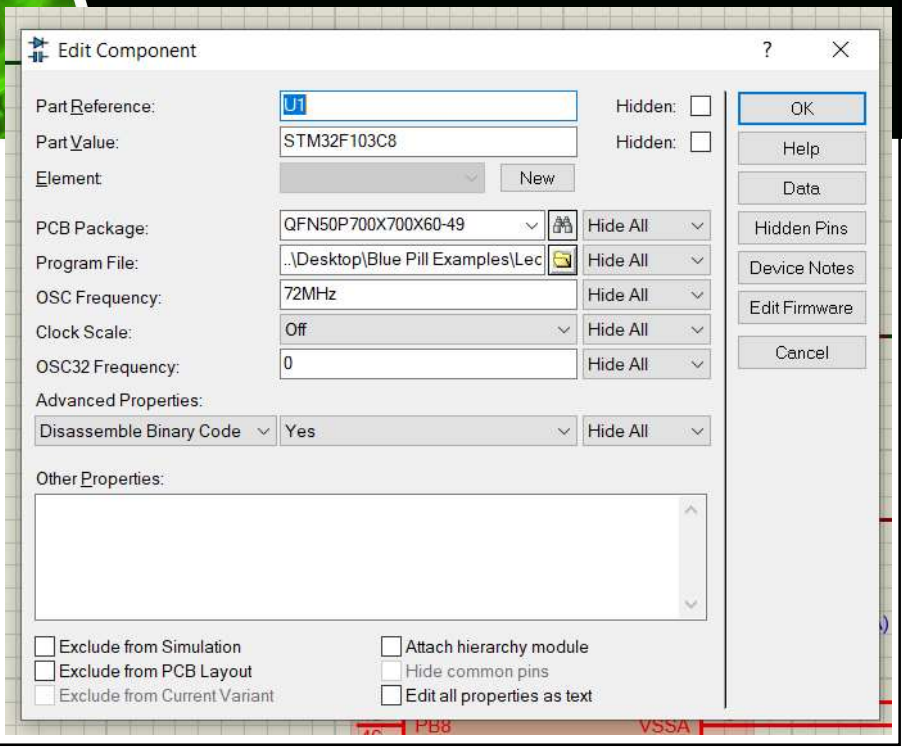
- We will be using Proteus 8.15



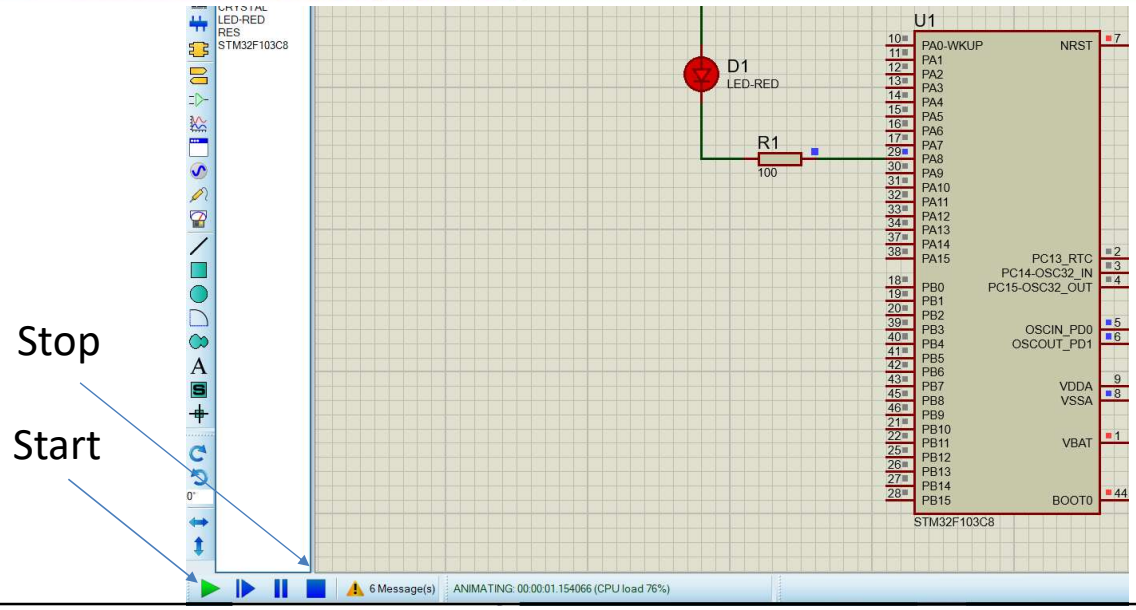
Schematic Diagram



Linking Hex File

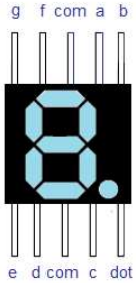


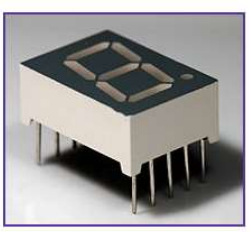
Play and Replay with Different Delay time

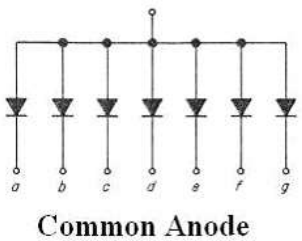


Single 7-Segment Display

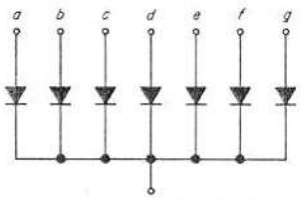
- Lit the digits “0” to “9” in a single 7-segment display at an interval of 1 second.
- There are two types of 7-segment display.
  - Common Cathode
  - Common anode
- Assume that the module is a common-cathode one.





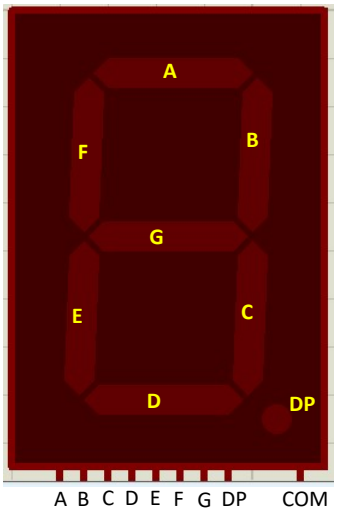


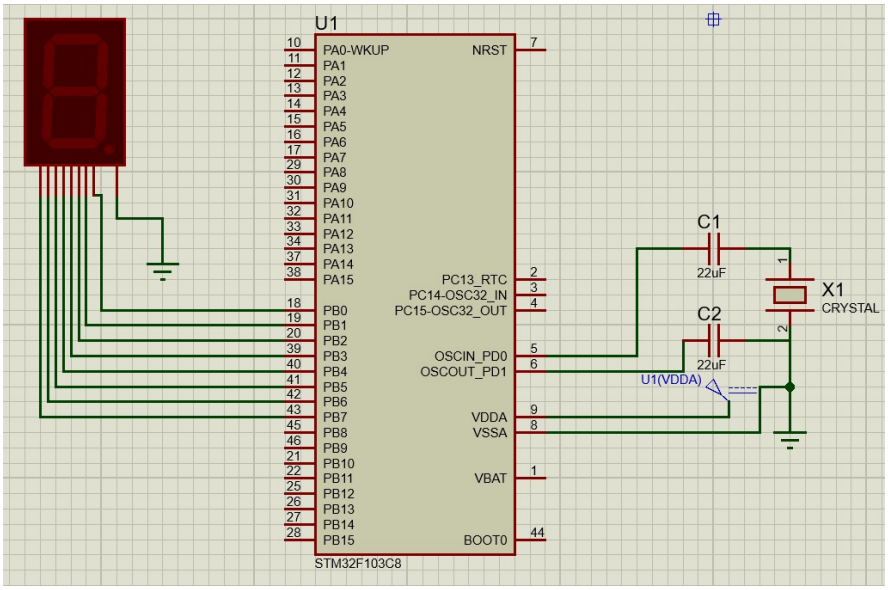
Common Anode



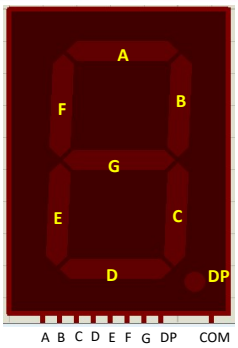
Common Cathode

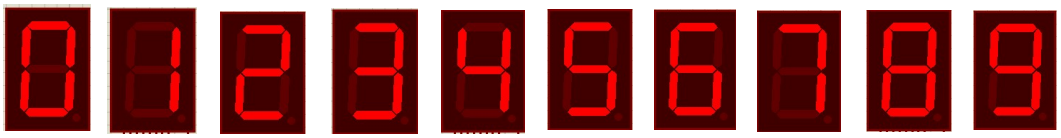
Connection Diagram

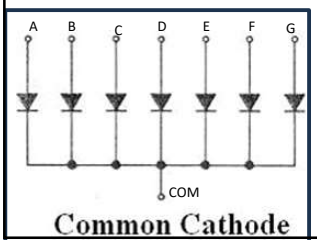




Hex Value to be sent







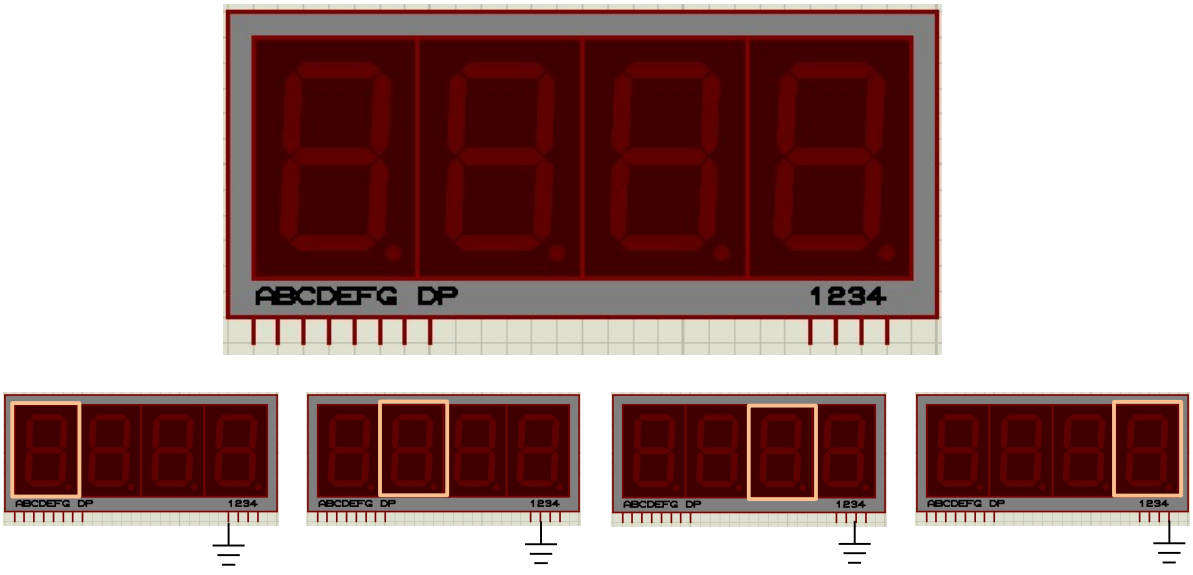
Our Case
=> CC

Digit	Segments								HEX
	A	B	C	D	E	F	G	DP	
0	1	1	1	1	1	1	0	0	FC
1	0	1	1	0	0	0	0	0	60
2	1	1	0	1	1	0	1	0	DA
3	1	1	1	1	0	0	1	0	F2
4	0	1	1	0	0	1	1	0	66
5	1	0	1	1	0	1	1	0	B6
6	1	0	1	1	1	1	1	0	BE
7	1	1	1	0	0	0	0	0	E0
8	1	1	1	1	1	1	1	0	FE
9	1	1	1	1	0	1	1	0	F6

Relevant portion of the code

```
double DigHex[] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xF6};


        while (1)
        {
                for (int i=0; i<=9; i++) {
                        GPIOB->ODR=DigHex[i];
                        HAL_Delay(500);
                }
        }
```

Multi-digit 7-segment display



Home Assignment-1

- Display a 4-digit number in a multi digit 7-segment display.
- Date of submission: Before the next class.

Thanks