➔ Microprocessor

CPU is stand-alone, RAM, ROM, I/O, timer are separate

Designer can decide on the amount of ROM, RAM and I/O ports.

Expensive

Versatility

General-purpose

➔ Microcontroller

CPU, RAM, ROM, I/O and timer are all on a single chip

Fixed amount of on-chip ROM, RAM, I/O ports

Not Expensive

Single-purpose

Special Purpose

➔ Harvard Architecture

In Harvard Architecture the data and instructions are stored in separate memory units each with their own bus.

Advantages:

Speeding up the data transfer rate,

Permits the designer to implement different bus widths and word sizes for program and data memory space.

➔ Von Neumann Architecture

In Von-Neumann architecture, there is no separate data and program memory. Instead, a single memory connection is given to the CPU.

Speed of execution is slower since it cannot fetch the data and instructions at the same time.

➔ Features of 8051

1. 8 bit Processor
2. 4KB Internal ROM
3. 128 Bytes Internal RAM
4. Four 8 BIT I/O PORTS (32 I/O LINES)
5. Two 16 Bit Timers/Counters
6. On Chip Full Duplex UART for Serial Communication
7. 5 Vector Interrupts (2 External, 3 Internal - Timer0,Timer1,Serial)
8. On Chip Clock Oscillator
9. 16 bit Address bus
   -->64k External Code Memory
   -->64k External Data Memory
10. 16-bit program counter to access external Code Memory and
11. 16 bit Data Pointer to access external Data Memory

    flags
12. 32 General Purpose Registers each of 8 bits

➔ 8051 Addressing Modes

The CPU can access data in various ways, which are called addressing modes

Immediate

Register

Direct

Register indirect

External Direct

➔ 8051 Instruction Set

8051 instructions have 8-bit opcode

There are 256 possible instructions of which 255 are

implemented

➔ INTERRUPT

In interrupt we learned

-->An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service

-->A single microcontroller can serve several devices by

two ways:

Interrupt

Polling

➔ INTERRUPT Vs POLLING

1)Interrupts:-

-->Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts

whatever it is doing and serves the device.

-->The program which is associated with the interrupt is called the interrupt

service routine (ISR) or interrupt handler.

2)Polling:-

-->The microcontroller continuously monitors the status of a given device. When the conditions met, it performs the service.

-->After that, it moves on to monitor the next device until every one is

serviced.

➔ There are six interrupts in 8051 as follows:

1) Reset – power-up reset.

2) Two interrupts are set aside for the timers.

one for timer 0 and one for timer 1

3) Two interrupts are set aside for hardware external interrupts.

P3.2 and P3.3 are for the external hardware interrupts INT0 (or EX1), and INT1 (or EX2)

4) Serial communication has a single interrupt that belongs to both receive and transfer.

➔ MBed account in running first LED blinking

-->We got a kit STM32. We discovered the datasheets of it, GPIO settings, clocks, timers, and other things of controller.

-->We Done our first program Hello world - Led Blinking.

-->We done this in online mbed Keil-ide, for that we folowed some steps,as folows:

1) We first register mbed account and run our first LED blink program,

we go to https://os.mbed.com/

2) There we sign up and made an account.

3) After that we select our board and import "mbed.h" library.

4) #include "mbed.h"

DigitalOut myled1(PA_5);

```
int main() {
while(1) {
myled1 = 1;
wait(0.2);
myled1 = 0;
wait(0.2);
}
}
```

We wrote this code and run it, and led was blinking

➔ Demystify STM32 Microcontroller

1. Nested means interrupts can be interrupted by another interrupt having a higher priority.

2. Here, Timer peripheral priority value is lesser than ADC, so TIMER interrupt is more urgent than ADC.

3. TIMER priority is HIGHER than ADC priority.

4. If both interrupt hit the NVIC at the same time, NVIC allows TIMER interrupt first

➔ Structe Padding

Code -

Code for MACROS

```
#define RCC_CLOCK_REG_ADDR      ((uint32_t*)0x4002104C)
#define PORT_A_MODE_ADDR        ((uint32_t*)0x48000000)
#define PORT_A_OUT_REG ((uint32_t*)0x48000014)
```

```
struct DataPacket
{
   char dataA;
   int          dataB;
   short dataC;
   char dataD;
};

int main(void)
{
   struct DataPacket data;
   data.dataA = 0xFF;
   data.dataB = 0x4F4C4C45;
   data.dataC = 0x48CD;
   data.dataD = 0x45;
}
```

<---OUTPUT--->

```
-O0
make -j4 all
arm-none-eabi-size   01blinkLedBare.elf
text             data      bss       dec       hex    filename
5400             108      1588     7096     1bb8  01blinkLedBare.elf
Finished building: default.size.stdout

-O1
arm-none-eabi-size   01blinkLedBare.elf
arm-none-eabi-objdump -h -S  01blinkLedBare.elf  > "01blinkLedBare.list"
```

text data bss dec hex filename

5140 108 1588 6836 1ab4 01blinkLedBare.elf

Finished building: default.size.stdout

-O2

arm-none-eabi-size 01blinkLedBare.elf

arm-none-eabi-objdump -h -S 01blinkLedBare.elf > "01blinkLedBare.list"

text data bss dec hex filename

5124 108 1588 6820 1aa4 01blinkLedBare.elf

Finished building: default.size.stdout

-O3

arm-none-eabi-size 01blinkLedBare.elf

arm-none-eabi-objdump -h -S 01blinkLedBare.elf > "01blinkLedBare.list"

text data bss dec hex filename

5124 108 1588 6820 1aa4 01blinkLedBare.elf

Finished building: default.size.stdout

➔ Priority Levels

1. It is vendor specific.

2. For STM32L4x5 MCU has 16 different priority levels.

3. TI TMC4C has 8 different priorit

➔ Real Time System

1) Real time doesn’t mean a fast computer or super

computer. It is a system of dealing with guarantee and timeliness rather than speed and performance.

2) Real time is human inference behavior rather than machine sense of time.

3) It is the system in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced.

4) If the timing constraints of the system are not met system failure is said to have occurred.

Create First Task and Schedular

Step1:

```
/* USER CODE BEGIN Includes */
    #include "FreeRTOS.h"
    #include "Task.h"
    #include <stdio.h>
```

Step3: Prototype to create first task is as following. Read about all inside pieces of task creation from following link https://freertos.org/a00125.html

```
/* USER CODE BEGIN 2 */
     status = xTaskCreate(task1_handler, "Task1", 200, "Hello World from Task-1", 2, &task1_handle);

     configASSERT(status == pdPASS);
```

Step4: Initialize return and status as following

```
/* USER CODE BEGIN 1 */
   TaskHandle_t task1_handle;

   BaseType_t status;
```

```
/* USER CODE END 1 */
```

Step5: Define and implement task handlers functions as following:

```
/* USER CODE BEGIN 4 */

static void task1_handler(void* parameters)
{
   while(1)
   {
      printf("%s\n", (char*)parameters);
   }

}

/* USER CODE END 4 */
```

Step6: Declare function prototype definition in Private Function Prototype (PFP) as following:

```
/* USER CODE BEGIN PFP */
static void task1_handler(void* parameters);
```

Step7:

One warning for Malloc failed will come which will be resolved by setting 0 in FreeRTOSConfig.h

```
#define configUSE_MALLOC_FAILED_HOOK                    0
```

Other error will be from memory allocation for tasks. Tasks are being assigned dynamic memory, not static memory, so static allocation should be made as zero.

```
#define configSUPPORT_STATIC_ALLOCATION                 0
```

Step8:

```
// Start the real time scheduler.
      vTaskStartScheduler();
```

// Will not get here unless there is insufficient RAM.

Follow the steps as mentioned below and resume the debug mode in step no 5

--->taskyeild() we can synshronize all the task.

➔ USART

1. Most UART are full duplex -- they have seprate pins and electronc hardware for the transmitter and reciever that allows serial i/o to take place simultaneously.
2. Based around shift registers and clock signals
3. UART clock determines baud rate
4. It frams the data with-
   --> Start bit to provide syn to the reciever
   --> One or more stop bits to signal end of data.
5. Most UART can also optionally generate parity bits in transmision and parity checking on reseption to provide simle error detection.

```
/* USER CODE BEGIN PV */
  uint8_t msg[] = "Hi, Welcome to UART demo!!\r\n";
  uint8_t msg1[] = "LED ON\r\n";
  uint8_t msg2[] = "LED OFF\r\n";
/* USER CODE END PV */

/* USER CODE BEGIN 2 */
  HAL_UART_Transmit(&huart2, msg, sizeof(msg), 1000);
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN 3 */
    HAL_GPIO_WritePin(myLED_GPIO_Port, myLED_Pin, 1);
    HAL_UART_Transmit(&huart2, msg1, sizeof(msg1), 1000);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(myLED_GPIO_Port, myLED_Pin, 0);
    HAL_UART_Transmit(&huart2, msg2, sizeof(msg2), 1000);
    HAL_Delay(1000);
/* USER CODE END 3 */
```

We can see output of it on "TERA Terminal"

➔ LCD

We learn to use UI of IDE STM32Cubel

We displyed Humidity, Temp and Pressure on LCD, using STM32 MCU's "hts221" and "lps22hb" sensor

We print something on LCD by making changes in "main.c" as follows -

```
  /* USER CODE BEGIN Includes */
    #include "lcd16x2_i2c.h"
  /* USER CODE END Includes */

  /* USER CODE BEGIN 2 */
    if(lcd16x2_i2c_init(&hi2c3))
    {
      HAL_GPIO_WritePin(MYLED1_GPIO_Port, MYLED1_Pin, GPIO_PIN_SET);
    }
    lcd16x2_i2c_printf("Hello World!");
  /* USER CODE END 2 */
```

➔ SPI

We also discover SPI communication using STM32 board.

We connected SPI ports like SPI_MISO and SPI_MOSI with jumper wire to establish SPI communication.

Code -

```
/* USER CODE BEGIN PV */
  uint8_t buffer_tx[10] = {10,23,14,16,17,22,34,67,23,67};
  uint8_t buffer_rx[10];
/* USER CODE END PV */

/* USER CODE BEGIN 2 */
  HAL_SPI_TransmitReceive(&hspi1, buffer_tx, buffer_rx, 10, 100);
/* USER CODE END 2 */
```

With jumper wire we can see some data in Reciever Buffer but after removing the wire the buffer is empty.

➔ Bluetooth

Advantages of Bluetooth -

Bluetooth is an amazing form of wireless technology that has taken “hands-free” communications to a new level. For those of you still fumbling around with just a handset, below is an introductory list of benefits you may be missing out on:

1. Wireless

Wireless capability is Bluetooth's claim to fame, making for a safer environment by eliminating those wires that do nothing but get in your way. It also works great with laptop computers and other portable devices as there is no need to worry about connecting cables.

2. Affordable

Despite its amazing functionality, the Bluetooth technology is actually inexpensive for developers to implement into their devices, resulting in cheaper costs which is passed down to you, the consumer.

3. Easy Automation

With Bluetooth you don't have to make any connections or tamper with any buttons. Two or more Bluetooth-enabled devices can automatically communicate when coming within a range of up 30 feet of one another.

4. The Wireless Standard

Bluetooth is the standardized wireless protocol, meaning that it is highly compatible with nearly any wireless device. It has the ability to connect various devices to each other, even if they are not of same model or brand.

5. Minimal Interference

Bluetooth-enabled devices offer excellent performance and are generally able to elude interference from the signals of other wireless devices. This is made possible by the use of low power signals and an underlying technology called frequency hopping.

6. Energy-efficient

Because the technology uses low power signals, Bluetooth requires a minimal amount of energy to function which results in less power consumption. This is a great benefit for mobile device users as additional battery power allows them to enjoy longer talk times.

7. Sharing of Data and Voice Communications

The protocol for Bluetooth is designed to enable compatible devices to share data and voice with each other. This is what allows you to safely drive around town with a mobile phone in your pocket while carrying out the conversation on a headset.

8. Your own Personal Area Network

Bluetooth technology has the ability to form a network with up to seven devices within a range of to 30 feet, allowing them to be connected to one another. Multiple PANs can also be set up in a single room of users.

9. Scalable

The protocol for Bluetooth allows it to be updated with ease. These upgrades offer new features and are typically compatible with older forms. Newer versions of the technology are currently being developed.

10. Universal

Bluetooth represents a universal standard that is recognized throughout the world. Because the technology is extremely popular, you can almost assure that your devices will be supported for years to come.

We learned BLE setup in STM32 CubeIDE for STML475 IOT kit to transmit and recieve data.

We also saw graph of Temp and Pressure in BLE scanner andriod app.

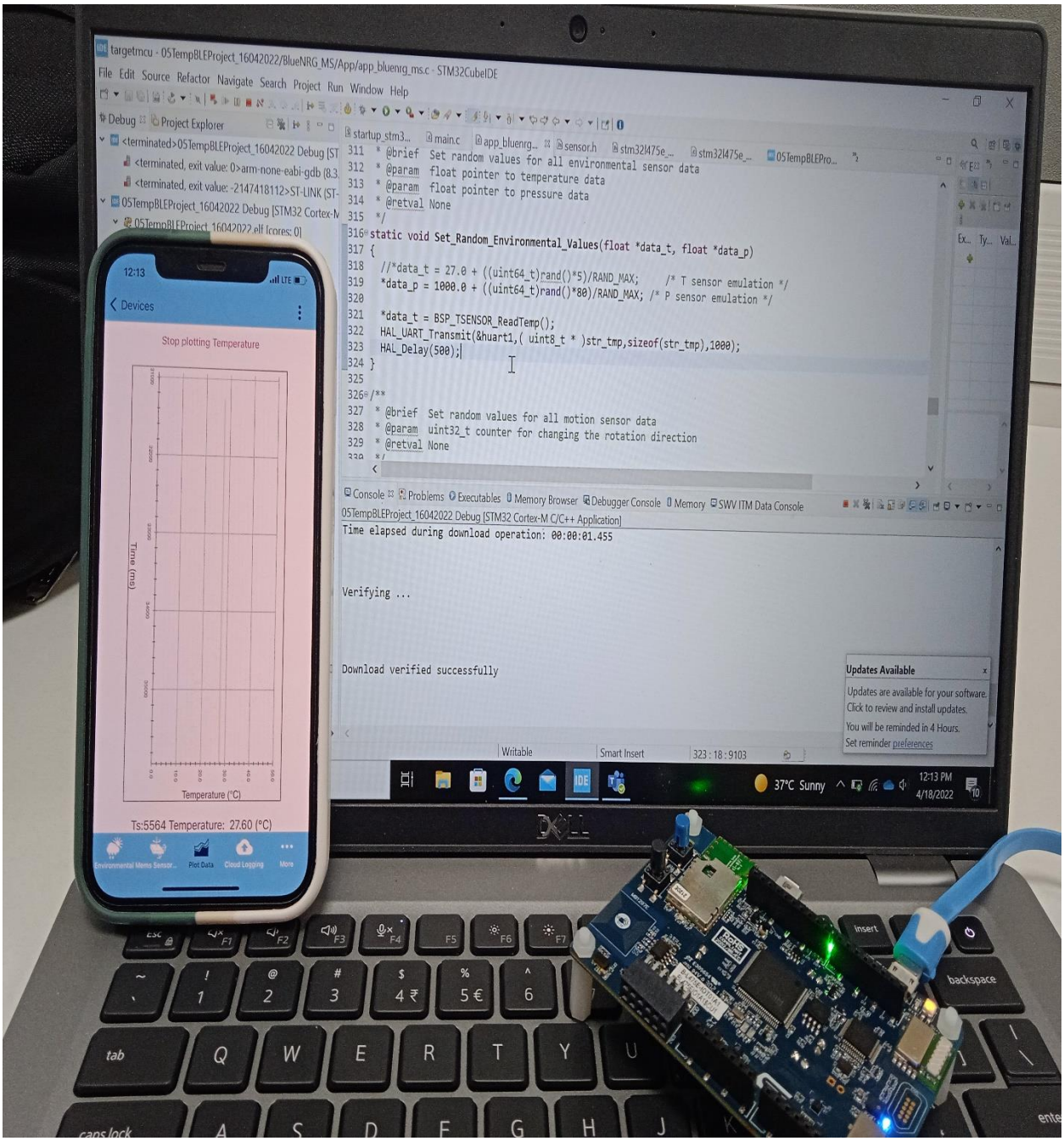

Figure 1 temperature program