

BHARAT ELECTRONICS LIMITED



NAME: TARUN SAI GAMPLA

TRAINEE-ID: PT – 430

UNIVERSITY ROLL NO: 21BEC7115

DEPARTMENT: Product Development and Innovation Centre (PDIC) , Embedded and Display Systems.

COLLEGE / UNIVERSITY NAME: Vellore Institute of Technology - AP

<p>Mr. Vijay Kumar U Guide, Senior Engineer, Embedded Systems, Bharat Electronics Limited</p>	<p>Mr. Sant Kumar Deputy General Manager, Embedded Systems, Bharat Electronics Limited.</p>
--	--

A Public Sector Undertaking under the Ministry of Defence, Government of India

INDEX

S.No	Title	Pg.No
1	Introduction To GPIO's	1
2	Understanding AM5728	4
3	Basic Communication Protocols	6
4	Analog to Digital Converters (ADC)	11
5	Understanding Basic Linux Commands	12
6	Raspberry pi 3 model B	13
7	Basic Terminologies (Pull up pull down, open drain, Push pull)	15
8	STM32F030R8 GPIO configuration and Register base Addresses	16
9	STM32F030R8 Bus interfaces	24
10	STM32F030R8 RCC	29
11	Semi Hosting	30
12	Difference between uint and volatile uint	31
13	Basic Program STM32F030R8	32
14	Basic GPIO Driver building F030R8	36
15	References	42

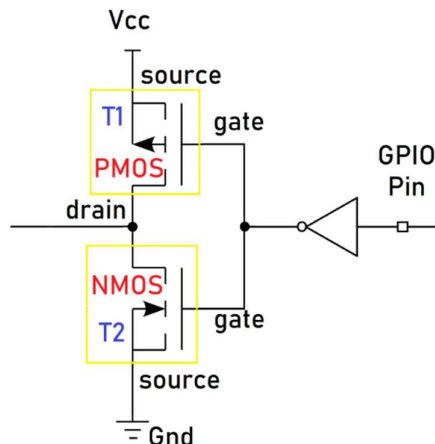
INTRODUCTION TO GPIO

GPIO [General Purpose Input Output]:

- Reads input when configured as input.
- Acts as outputs when configured as output.
- These pins can also be used to issue interrupt to the processor when configured as external hardware interrupt pins.
- GPIO pins are also used for communication.

GPIO as Input:

The input buffer is involved in the input mode. If you make the enable line as 1, this will deactivate the output buffer and activate the input buffer. GPIO Input buffer also has CMOS where NMOS and PMOS are connected like the output buffer. But here, the gate is connected to the GPIO pin, and the drain is connected to the processor.



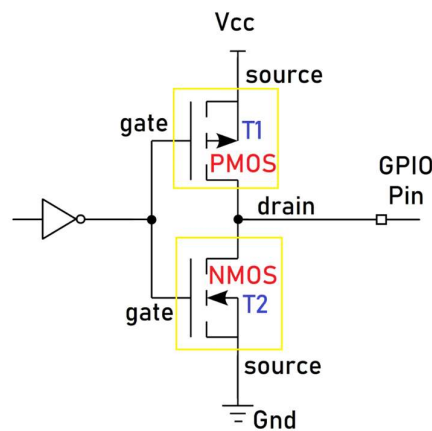
A high signal at gate of NMos will trigger the NMOS will trigger the NMOS, and a low signal will not trigger it. This is opposite for PMOS. Triggering this allows current to flow drain to source [PMOS] / source to drain [NMOS]. Initially as we power on the Microcontroller, all the GPIO's by default are in high impedance state, causes leakage. To avoid the leakage, we can use pull-down or pull-up logic, the same logic that will be used for the push buttons.

When the external device is writing 1 to the GPIO pin, the first inverter will make it as 0. So, the PMOS transistor (T1) will be activated and the NMOS transistor (T2) will be deactivated. Finally, T1 will take the Vcc and give it to the processor. So, we will get 1 in our software. When the external device is writing 0 to the GPIO pin, the first inverter will make it as 1. So,

the PMOS transistor (T1) will be deactivated and the NMOS transistor (T2) will be activated. Finally, T2 will take the Gnd and give it to the processor.

GPIO as Output:

First, we will see how the GPIO works in output mode. If you make the enable line as 0, this will activate the output buffer and deactivate the input buffer. The output buffer is connected to the two CMOS Transistors. CMOS technology uses both N-type (NMOS) and P-type (PMOS) transistors, and it is connected.

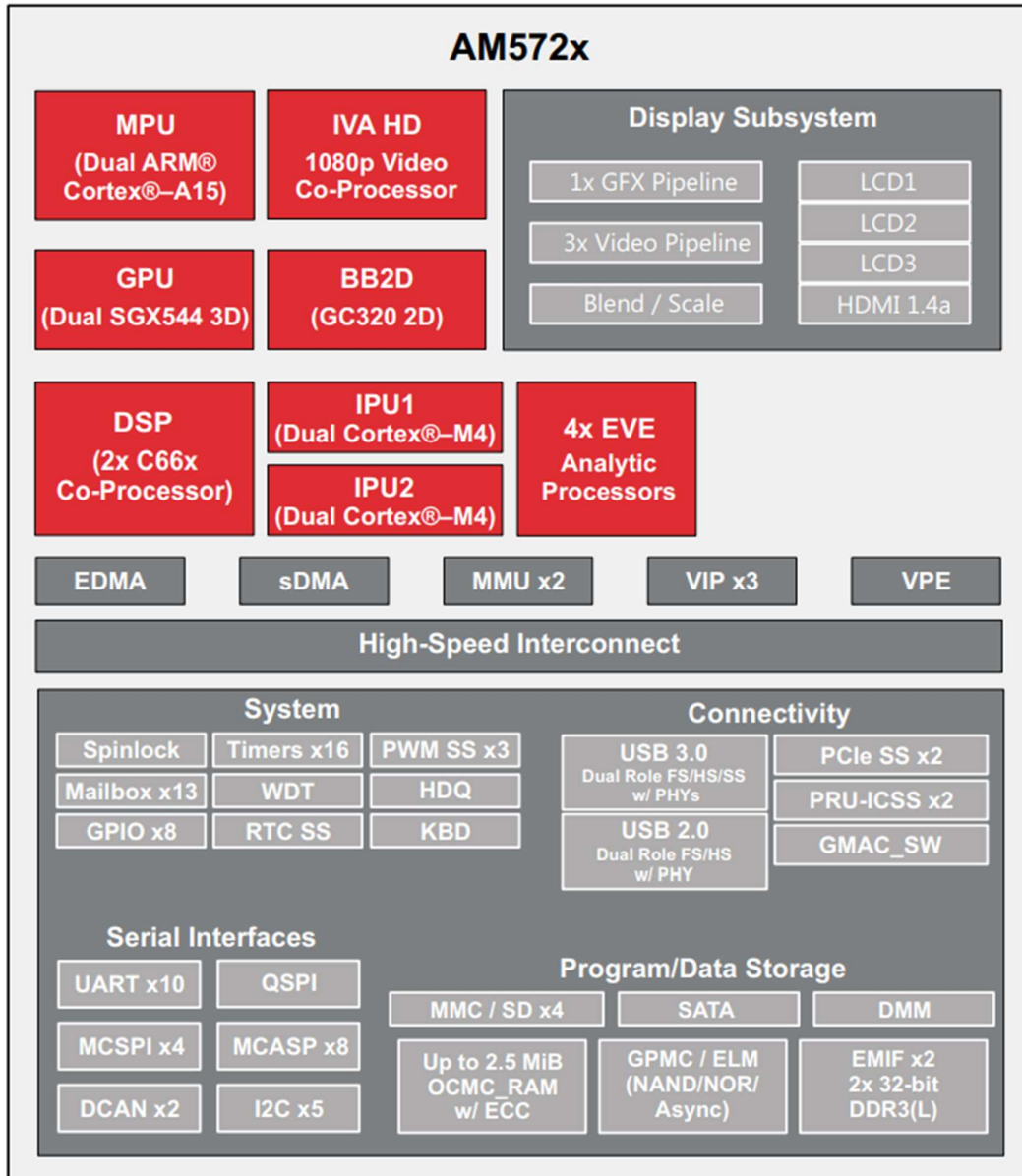


In the above image, T1 is the PMOS transistor and T2 is the NMOS transistor. When we write 1 from our code, the first inverter will make it as 0. So, the PMOS transistor (T1) will be activated and the NMOS transistor (T2) will be deactivated. Finally, T1 will take the Vcc and give it to the GPIO pin. If we connect the LED now, it will be ON.

When we write 0 from our code, the first inverter will make it as 1. So, the PMOS transistor (T1) will be deactivated and the NMOS transistor (T2) will be activated. Finally, T2 will take the Gnd and give it to the GPIO pin. If we connect the LED now, it will be turned OFF.

UNDERSTANDING AM5728

The AM572x is a high-performance, Sitara™ device, integrated on a 28-nm technology. The architecture is designed for embedded applications including industrial communication, Human Machine Interface (HMI), automation and control, and other high performance and general use applications, and best-in-class CPU performance, video, image, and graphics processing sufficient to support:



This Processor Contains:

1. MPU Subsystem.
2. DSP Subsystems
3. EVE Subsystems
4. PRU-ICSS
5. IPU Subsystems

6. IVA-HD Subsystem
7. Display Subsystem
8. Video Processing Subsystem
9. Video Capture

10. 3D GPU Subsystem
11. BB2D Subsystem, and some more.

The general-purpose interface combines eight GPIO modules for a flexible, user-programmable, general-purpose input/output (I/O) controller. The general-purpose interface implements functions that are not implemented with the dedicated controllers in the device and require simple input and/or output softwarecontrolled signals. The GPIO allows a variety of custom connections and expands the I/O capabilities of the system to the real world.

GENERAL PURPOSE INTERFACE SIGNALS

Signal	I/O ⁽¹⁾	Description	Reset Value ⁽²⁾
gpio1_[0:3]	I	GPIO (inputs only)	Hi-Z
gpio1_[4:31]	I/O	GPIO	Hi-Z
gpio2_[0:31]	I/O	GPIO	Hi-Z
gpio3_[0:31]	I/O	GPIO	Hi-Z
gpio4_[0:31]	I/O	GPIO	Hi-Z
gpio5_[0:31]	I/O	GPIO	Hi-Z
gpio6_[4:31]	I/O	GPIO	Hi-Z
gpio7_[0:19]	I/O	GPIO	Hi-Z
gpio7_[22:31]	I/O	GPIO	Hi-Z
gpio8_[0:23]	I/O	GPIO	Hi-Z
gpio8_27	I	GPIO (input only)	Hi-Z
gpio8_[28:31]	I/O	GPIO	Hi-Z

BASIC COMMUNICATION PROTOCOLS

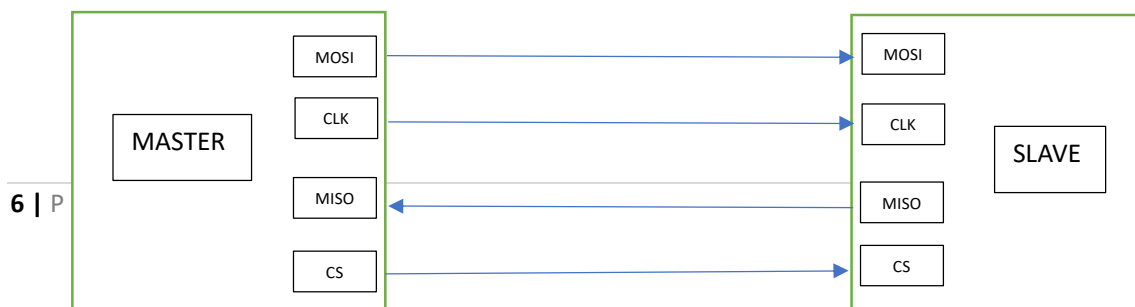
SPI COMMUNICATION PROTOCOL (SPI):

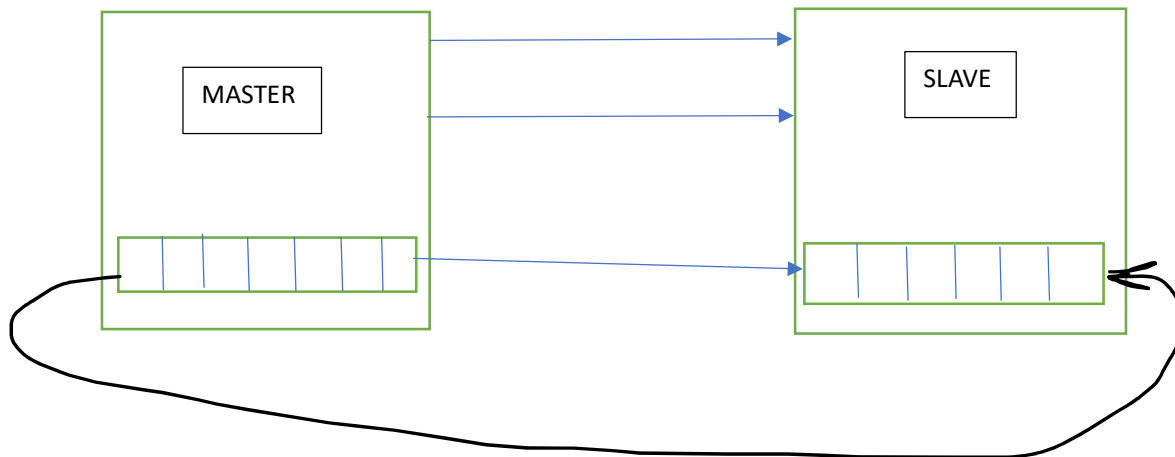
Developed by Motorola to provide full duplex synchronous serial communication between master and slave devices.

I2C – Half Duplex [Only transmission or Receiving at a time]

SPI – Full Duplex [Both transmission and receiving at same time]

For every clock cycle 1-bit is transferred.

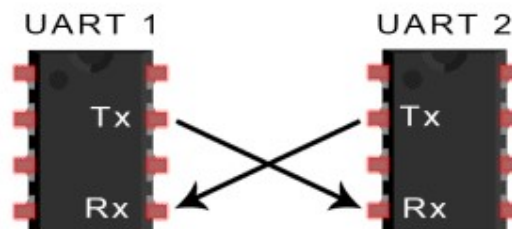




- When Master wants to send the data, it loads the data into shift register first and then selects a slave using SS.
- The serial clk line (SCK) is then enabled and one bit of data is shifted on MOSI line with each clock pulse.
- If both Master and slave want to transfer data simultaneously then MISO will shift a bit from slave to Master at the start of shift register and MOSI will shift a bit from master to slave at the beginning
- Here both MOSI and MISO shifts for parallel transmission.

UART COMMUNICATION PROTOCOL:

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART.



UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet

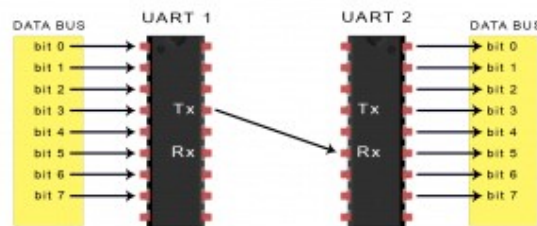
being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

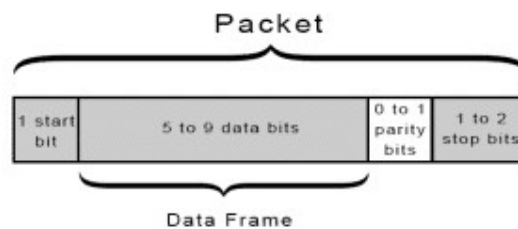
In Universal Asynchronous Receiver / Transmission is no incoming clk signal associated with the data, so the receiver needs to know the baud rate. In which require only dateline and data is transmitted byte by byte. UART cannot work like USART.

HOW UART WORKS:

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end.



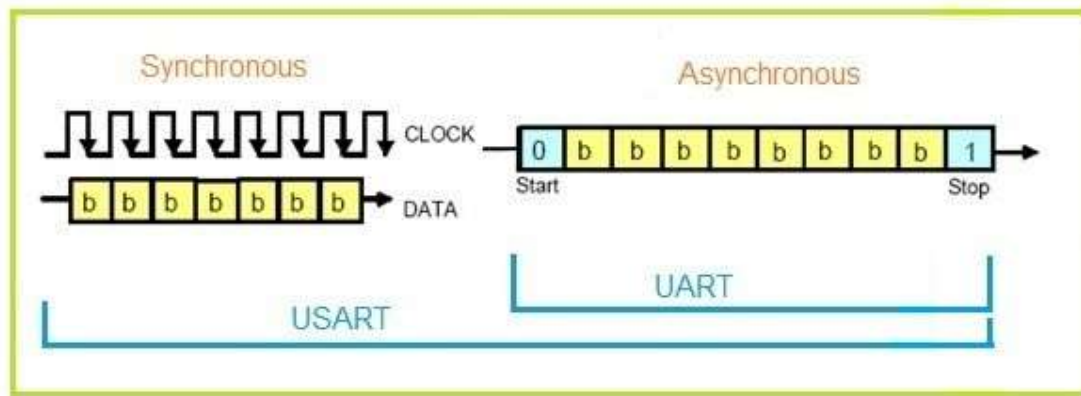
UART transmitted data is organized into packets. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits.



USART (Universal Synchronous/Asynchronous receiver/transmitter) PROTOCOL:

A USART (universal synchronous/asynchronous receiver/transmitter) is hardware that enables a device to communicate using serial protocols. It can function in a slower asynchronous mode, like a universal asynchronous receiver/transmitter (UART), or in a faster synchronous mode with a clock signal. USARTs are no longer common in consumer PCs but are still used in industrial equipment and embedded systems.

A UART device can use asynchronous communication protocols. A USART device can use both asynchronous and synchronous communication protocols. Therefore, a USART can do anything a UART can do and more. Because a USART requires more complex circuitry and more communication lines to fully implement, many devices may only implement a UART to save on cost, complexity or power usage.

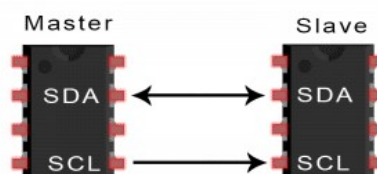


The internal clock is derived from the internal CPU clock. Both transmitter and receiver use the same clock to operate from. The transmitter is synchronous to the clock. In coming data to the receiver is not. The baud rates are not exact power of 2 multiples of the CPU clock, thus baud rate inaccuracies occur at some settings. This is why you see some "funny" crystal oscillator frequencies.

I2C COMMUNICATION PROTOCOL

I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

Like UART communication, I2C only uses two wires to transmit data between devices:



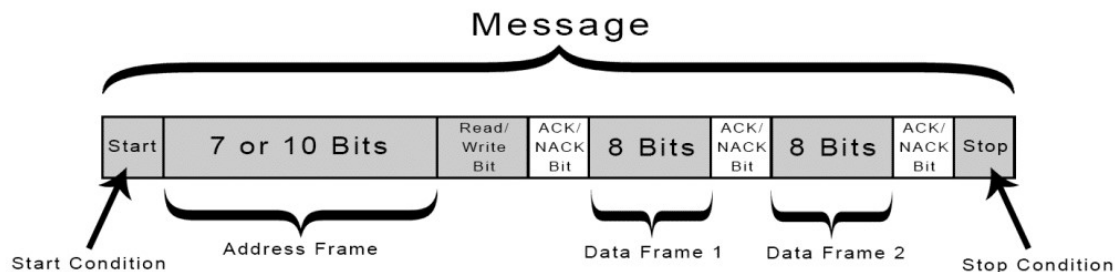
SDA (Serial Data) – The line for the master and slave to send and receive data.

SCL (Serial Clock) – The line that carries the clock signal.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line). Like SPI, I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

WORKING OF I2C:

With I2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame.



Start Condition: The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.

Stop Condition: The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

Address Frame: A 7- or 10-bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

Multiple masters can be connected to a single slave or multiple slaves. The problem with multiple masters in the same system comes when two masters try to send or receive data at the same time over the SDA line. To solve this problem, each master needs to detect if the SDA line is low or high before transmitting a message. If the SDA line is low, this means that another master has control of the bus, and the master should wait to send the message. If the SDA line is high, then it's safe to transmit the message. To connect multiple masters to multiple slaves, use the following diagram, with 4.7K Ohm pull-up resistors connecting the SDA and SCL lines to Vcc.

ANALOG TO DIGITAL CONVERTER

- Analog to digital converter (ADC) converts an analog signal a digital signal.
- Digital output is a 2's complement binary number proportional to input (Generally)
- ADC converts continuous time and continuous amplitude to discrete time and discrete amplitude digital signal.
- Instead of continuously performing the conversion, ADC does conversion periodically, sampling the input, and limiting the allowable band width of the input signal.
- Performance is taken up by bandwidth (sampling rate) & signal to noise ratio.
- If any ADC operates at a sampling rate greater than twice the bandwidth of the signal, then as per the Nyquist – Shannon sampling theorem, near perfect reconstruction is possible.

RESOLUTION:

- Indicates the no of discrete values it can produce over the allowed range of analog input values.
- Resolution determines the magnitude of quantization error and therefore determines the maximum possible signal to noise ratio for an ideal ADC without sampling.

QUANTIZATION ERROR:

- Rounding error b/t the analog input voltage to the ADC and the output digitized value.

DITHER:

- In ADC's, the performance can usually be improved using dither. A very small amount of random noise (eg: white noise), which is to be added to input before conversion.
- It extends the effective range of signals that the ADC can convert, at the expense of slight noise increase.
- Dither only increases resolution of a sampler, cannot improve linearity, thus accuracy does not necessarily improve.

SAMPLING RATE:

- The rate at which the new digital values are sampled from the analog signal.
- The rate of new values is called sampling rate and sampling frequency of the converter.
- A continuously varying bandlimited signal can be sampled and then the original signal can be reproduced from the discrete time values by a reconstruction filter.

CONVERSION TIME: The time the converter performs conversion.

LINUX COMMAND LIST BASICS

HARDWARE INFORMATION:

Dmesg – Show bootup messages.
Cat /proc/cpuinfo – Show Cpu info
Free -h – show free and used memory
Lshw – hardware info
lsblk – Block devices into
lspci -tv – free diagram of PCI devices – same can be done to USB.
dmidecode – show bios hardware into
hdparm -i /dev/[disk] – show disk data into
hdparm -tT/dev/[disk] – Disk read speed test

FILE COMPRESSION:

Tar cf[file.tar] [file] – create a tarfile from a file
Tar xf [file.tar] – extract archived file
Tarczf [file.tar.gz] – create a gzip tar file
Gzip [file] – create a gz compressed file.

PACKAGE INSTALLATION:

apt install
yum search [keyword]
yum info [package]
yum install [package.rpm]
rpm -i [package.rpm]

tar zxvf source_code.tar.gz
cd source_code
./configure
Make

SSH LOGIN

ssh user@host

ssh host

ssh -p port user@host

telenet host

BASIC RASPBERRY PI 3 MODEL B

Python library installation:

```
$sudo apt-get install python-rpi.gpio python-rpi.gpio
```

For configuring Raspberry Pi in Raspbian, we are using Raspbian with PIXEL desktop. It is one of the best ways to get Raspbian started with the Raspberry Pi. Once we finish booting, we will be in the PIXEL desktop environment.

Now to open the menu, you need to click the button that has the Raspberry Pi logo on it. This button will be in the top left. After clicking the button, choose Raspberry Pi configuration from the preferences.

Configuration tool

Following is the configuration tool in PIXEL desktop –

Configuration tool

By default, the configuration tool opens to its system tab which has the following options –

Change Password – The default password is raspberry. You can change it by clicking the change password button.

Change the hostname – The default name is raspberry pi. You can also change it to the name, which you want to use on the network.

Boot – You can choose from the two options and control whether Raspberry Pi boots into the desktop or CLI i.e., command line interface.

Auto Login – With the help of this option, you can set whether the user should automatically log in or not.

Network at Boot – By choosing this option, you can set whether the pi user is automatically logged in or not.

Splash screen – You can enable or disable it. On enabling, it will display the graphical splash screen that shows when Raspberry Pi is booting.

Resolution – With the help of this option, you can configure the resolution of your screen.

Underscan – There are two options, enable or disable. It is used to change the size of the displayed screen image to optimally fill the screen. If you see a black border around the screen, you should disable the underscan. Whereas, you should enable the underscan, if your desktop does not fit your screen.

There are three other tabs namely Interfaces, Performance, and Localization. The job of interface tab is to enable or disable various connection options on your Raspberry Pi.

You can enable the Pi camera from the interface tab. You can also set up a secure connection between computers by using SSH (short for Secure Shell) option.

If you want to remote access your Pi with a graphical interface then, you can enable RealVNC software from this tab. SPI, I2C, Serial, 1-wire, and Remote GPIO are some other interfaces you can use.

There is another tab called Performance, which will give you access to the options for overclocking and changing the GPU memory.

The localization tab, as the name implies, enable us to set –

The character set used in our language.

Our time zone.

The keyboard setup as per our choice.

Our Wi-Fi country.

Configure Wi-Fi

You can check at the top right, there would be icons for Bluetooth and Wi-Fi. The fan-shaped icon is on the Wi-Fi. To configure your Wi-Fi, you need to click on that icon. Once clicked, it will open a menu showing the available networks. It also shows the option to turn off your Wi-Fi.

Among those available networks, you need to select a network. After selecting, it will prompt for entering the Wi-Fi password i.e., the Pre Shared Key.

If you see a red cross on the icon, it means your connection has been failed or dropped. To test whether your Wi-Fi is working correctly, open a web browser and visit a web page.

PULLUP, PULL DOWN, OPEN DRAIN AND PUSH/PULL

Pull-up Resistor:

Imagine you have a digital input pin on a microcontroller. When the pin is not actively driven to a high or low state by some external device, it might "float" to an undefined voltage. Connecting a pull-up resistor between the pin and the high voltage (V_{cc}) ensures that the pin reads a logical high (1) when there's no other active connection.

It's like having a gentle force (the pull-up resistor) that keeps the pin pulled towards a high state even when nothing else is actively pulling it low.

Pull-down Resistor:

Conversely, a pull-down resistor is connected between the pin and the low voltage (ground). This ensures that when there's no active connection pulling the pin high, the pull-down resistor pulls the pin to a logical low (0) state.

It's like having a gentle force that keeps the pin pulled towards a low state when nothing else is actively pulling it high.

Open Drain:

In an open-drain configuration, the output device (like a transistor) can actively pull the signal line low, but it doesn't actively drive it high. When inactive, the line is left unconnected (open).

This configuration is often used in scenarios where multiple devices need to share a common bus. By having open-drain outputs with pull-up resistors, you can create a wired-AND logic. If any device pulls the line low, it brings the overall voltage down; otherwise, the pull-up resistor keeps the line high.

Push-Pull:

In a push-pull configuration, you have two active devices (transistors) working together to drive a signal line.

For example, in the output stage of an amplifier or a microcontroller pin configured for push-pull output, one transistor actively pulls the line high by connecting to the positive power supply (V_{cc}) during one state, and the other transistor actively pulls the line low by connecting to ground during the opposite state.

This configuration is often used for strong and efficient driving of signals.

Understanding these concepts is fundamental when designing and troubleshooting digital circuits. They play a crucial role in signal stability, noise immunity, and efficient signal transmission in electronic systems.

General-purpose I/Os (GPIO)

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) and a 32-bit set/reset register (GPIOx_BSRR). Ports A and B also have a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection registers (GPIOx_AFRH and GPIOx_AFRL). On STM32F030xB and STM32F030xC devices, also ports C and D have two 32-bit alternate function selection registers (GPIOx_AFRH and GPIOx_AFRL).

GPIO MAIN FEATURES

- 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 port A or B I/O port configuration.
- Analog function
- Alternate function selection registers (at most 16 AFs possible 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.

GPIO port mode register (GPIOx_MODER)

(x = A to D, F)

Address offset: 0x00

Reset value: 0x2800 0000 for port A

Reset value: 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:0 MODER[15:0][1:0]: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O mode.

00: Input mode (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

8.4.2 GPIO port output type register (GPIOx_OTYPER)

(x = A to D, F)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 OT[15:0]: Port x configuration I/O pin y (y = 15 to 0)

These bits are written by software to configure the I/O output type.

0: Output push-pull (reset state)

1: Output open-drain

GPIO port output speed register (GPIOx_OSPEEDR)

(x = A to D, F)

Address offset: 0x08

Reset value: 0x0C00 0000 (for port A)

Reset value: 0x0000 0000 (for other ports)

Base address: 0x4800 0000

Physical address: 0x4800 0008

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Bits 31:0 OSPEEDR [15:0] [1:0]: Port x configuration I/O pin y (y = 15 to 0) These bits are written by software to configure the I/O output speed.

x0: Low speed

01: Medium speed

11: High speed

4) GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A to, D, F):

Address offset: 0x0C

Reset value: 0x2400 0000 (for port A)

Reset value: 0x0000 0000 (for other ports)

Base address: 0x4800 0000

Physical address: 0x4800 000C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Bits 31:0 PUPDR [15:0] [1:0]: Port x configuration I/O pin y (y = 15 to 0) These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down

01: Pull-up

10: Pull-down

11: Reserved

5)GPIO port input data register (GPIOx_IDR) (x = A to D, F):

Address offset: 0x10

Reset value: 0x0000 XXXX

Base address:0x4800 0000

Physical address:0x4800 0010

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 IDR [15:0]: Port x input data I/O pin y (y = 15 to 0) These bits are read-only. They contain the input value of the corresponding I/O port.

6)GPIO port output data register (GPIOx_ODR) (x = A to D, F):

Address offset: 0x14

Reset value: 0x0000 0000

Base address:0x4800 0000

Physical address:0x4800 0014

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 ODR[15:0]: Port output data I/O pin y (y = 15 to 0) These bits can be read and written by software.

7)GPIO port bit set/reset register (GPIOx_BSRR) (x = A to D, F):

Address offset: 0x18

Reset value: 0x0000 0000

Base address:0x4800 0000

Physical address:0x4800 0018

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 BR [15:0]: Port x reset I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx.

Bits 15:0 BS [15:0]: Port x set I/O pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

8)GPIO port configuration lock register (GPIOx_LCKR) (x = A to B)

This register is used to lock the configuration of the port bits when a correct write sequence is applied to bit 16 (LCKK).

Each lock bit freezes a specific configuration register (control and alternate function registers).

Address offset: 0x1C

Reset value: 0x0000 0000

Base address:0x4800 0000

Physical address:0x4800 001C



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LCKK
															rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

Bit 16 LCKK: Lock key

This bit can be read any time. It can only be modified using the lock key write sequence.

0: Port configuration lock key not active

1: Port configuration lock key active.

Bits 15:0 LCK[15:0]: Port x lock I/O pin y (y = 15 to 0)

These bits are read/write but can only be written when the LCKK bit is 0.

0: Port configuration not locked

1: Port configuration locked

9)GPIO alternate function low register (GPIOx_AFRL) (x = A to D,):

Address offset: 0x20

Reset value: 0x0000 0000

Base address:0x4800 0000

Physical address:0x4800 0020

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL7[3:0]				AFSEL6[3:0]				AFSEL5[3:0]				AFSEL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFSEL3[3:0]				AFSEL2[3:0]				AFSEL1[3:0]				AFSEL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 AFSELy[3:0]: Alternate function selection for port x pin y (y = 0..7)

0000: AF0

0001: AF1

0010: AF2

0011: AF3

0100: AF4

0101: AF5

0110: AF6

0111: AF7

10)GPIO alternate function high register (GPIOx_AFRH) (x = A to D, F):

Address offset: 0x24

Reset value: 0x0000 0000

Base address:0x4800 0000

Physical address:0x4800 0024

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL15[3:0]				AFSEL14[3:0]				AFSEL13[3:0]				AFSEL12[3:0]			
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFSEL11[3:0]				AFSEL10[3:0]				AFSEL9[3:0]				AFSEL8[3:0]			
rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW	rW

Bits 31:0 AFSELY[3:0]: Alternate function selection for port x pin y (y = 8..15)

AFSELY selection:

0000: AF0

0001: AF1

0010: AF2

0011: AF3

0100: AF4

0101: AF5

0110: AF6

0111: AF7

11)GPIO port bit reset register (GPIOx_BRR) (x = A to D, F):

Address offset: 0x28

Reset value: 0x0000 0000

Base address:0x4800 0000

Physical address:0x4800 0028



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 BR[15:0]: Port x reset IO pin y (y = 15 to 0)

These bits are write-only. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODx bit

1: Reset the corresponding ODx bit

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOA_MODER	MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]	MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]																
	Reset value	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00	GPIOx_MODER (where x = B..D, F)	MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]	MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	GPIOx_OTYPER (where x = A..D, F)	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0																
	Reset value																																
0x08	GPIOA_OSPEEDR	OSPEEDR15[1:0]	OSPEEDR14[1:0]	OSPEEDR13[1:0]	OSPEEDR12[1:0]	OSPEEDR11[1:0]	OSPEEDR10[1:0]	OSPEEDR9[1:0]	OSPEEDR8[1:0]	OSPEEDR7[1:0]	OSPEEDR6[1:0]	OSPEEDR5[1:0]	OSPEEDR4[1:0]	OSPEEDR3[1:0]	OSPEEDR2[1:0]	OSPEEDR1[1:0]	OSPEEDR0[1:0]																
	Reset value	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	GPIOx_OSPEEDR (where x = B..D, F)	OSPEEDR15[1:0]	OSPEEDR14[1:0]	OSPEEDR13[1:0]	OSPEEDR12[1:0]	OSPEEDR11[1:0]	OSPEEDR10[1:0]	OSPEEDR9[1:0]	OSPEEDR8[1:0]	OSPEEDR7[1:0]	OSPEEDR6[1:0]	OSPEEDR5[1:0]	OSPEEDR4[1:0]	OSPEEDR3[1:0]	OSPEEDR2[1:0]	OSPEEDR1[1:0]	OSPEEDR0[1:0]																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOA_PUPDR	PUPDR15[1:0]	PUPDR14[1:0]	PUPDR13[1:0]	PUPDR12[1:0]	PUPDR11[1:0]	PUPDR10[1:0]	PUPDR9[1:0]	PUPDR8[1:0]	PUPDR7[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR0[1:0]																
	Reset value	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx_PUPDR (where x = B..D, F)	PUPDR15[1:0]	PUPDR14[1:0]	PUPDR13[1:0]	PUPDR12[1:0]	PUPDR11[1:0]	PUPDR10[1:0]	PUPDR9[1:0]	PUPDR8[1:0]	PUPDR7[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR0[1:0]																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_IDR (where x = A..D, F)	IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0																
	Reset value																																
0x14	GPIOx_ODR (where x = A..D, F)	ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0																
	Reset value																																
0x18	GPIOx_BSRR (where x = A..D, F)	BSR15	BSR14	BSR13	BSR12	BSR11	BSR10	BSR9	BSR8	BSR7	BSR6	BSR5	BSR4	BSR3	BSR2	BSR1	BSR0																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

AHB APB BASE ADDRESSES (BUS INTERFACES)

Table 17. STM32G074xx/xxKxx peripheral register boundary addresses

Bus	Boundary address	Size	Peripheral
-	0x4800 1800 - 0x5FFF FFFF	~384 MB	Reserved
AHB2	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF
	0x4800 1000 - 0x4800 13FF	1 KB	Reserved
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA
-	0x4002 4400 - 0x47FF FFFF	~128 MB	Reserved
AHB1	0x4002 3400 - 0x4002 43FF	4 KB	Reserved
	0x4002 3000 - 0x4002 33FF	1 KB	CRC
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH Interface
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved
	0x4002 1000 - 0x4002 13FF	1 KB	RCC
	0x4002 0400 - 0x4002 0FFF	3 KB	Reserved
	0x4002 0000 - 0x4002 03FF	1 KB	DMA
-	0x4001 8000 - 0x4001 FFFF	32 KB	Reserved
APB	0x4001 5C00 - 0x4001 7FFF	9 KB	Reserved
	0x4001 5800 - 0x4001 5BFF	1 KB	DBGMCU
	0x4001 4C00 - 0x4001 57FF	3 KB	Reserved
	0x4001 4800 - 0x4001 4BFF	1 KB	TIM17
	0x4001 4400 - 0x4001 47FF	1 KB	TIM16
	0x4001 4000 - 0x4001 43FF	1 KB	TIM15 ⁽¹⁾
	0x4001 3C00 - 0x4001 3FFF	1 KB	Reserved
	0x4001 3800 - 0x4001 3BFF	1 KB	USART1
	0x4001 3400 - 0x4001 37FF	1 KB	Reserved
	0x4001 3000 - 0x4001 33FF	1 KB	SPI1
	0x4001 2C00 - 0x4001 2FFF	1 KB	TIM1
	0x4001 2800 - 0x4001 2BFF	1 KB	Reserved
	0x4001 2400 - 0x4001 27FF	1 KB	ADC
	0x4001 1800 - 0x4001 23FF	3 KB	Reserved
	0x4001 1400 - 0x4001 17FF	1 KB	USART6 ⁽²⁾
	0x4001 0800 - 0x4001 13FF	3 KB	Reserved
	0x4001 0400 - 0x4001 07FF	1 KB	EXTI
	0x4001 0000 - 0x4001 03FF	1 KB	SYSCFG

Bus	Boundary address	Size	Peripheral
-	0x4000 8000 - 0x4000 FFFF	32 KB	Reserved
APB	0x4000 7400 - 0x4000 7FFF	3 KB	Reserved
	0x4000 7000 - 0x4000 73FF	1 KB	PWR
	0x4000 5C00 - 0x4000 6FFF	5 KB	Reserved
	0x4000 5800 - 0x4000 5BFF	1 KB	I2C2 ⁽¹⁾
	0x4000 5400 - 0x4000 57FF	1 KB	I2C1
	0x4000 5000 - 0x4000 53FF	1 KB	USART5 ⁽²⁾
	0x4000 4C00 - 0x4000 4FFF	1 KB	USART4 ⁽²⁾
	0x4000 4800 - 0x4000 4BFF	1 KB	USART3 ⁽²⁾
	0x4000 4400 - 0x4000 47FF	1 KB	USART2 ⁽¹⁾
	0x4000 3C00 - 0x4000 43FF	2 KB	Reserved
	0x4000 3800 - 0x4000 3BFF	1 KB	SPI2 ⁽¹⁾
	0x4000 3400 - 0x4000 37FF	1 KB	Reserved
	0x4000 3000 - 0x4000 33FF	1 KB	IWDG
	0x4000 2C00 - 0x4000 2FFF	1 KB	WWDG
	0x4000 2800 - 0x4000 2BFF	1 KB	RTC
	0x4000 2400 - 0x4000 27FF	1 KB	Reserved
	0x4000 2000 - 0x4000 23FF	1 KB	TIM14
	0x4000 1800 - 0x4000 1FFF	2 KB	Reserved
	0x4000 1400 - 0x4000 17FF	1 KB	TIM7 ⁽²⁾
	0x4000 1000 - 0x4000 13FF	1 KB	TIM6 ⁽¹⁾
	0x4000 0800 - 0x4000 0FFF	2 KB	Reserved
	0x4000 0400 - 0x4000 07FF	1 KB	TIM3
	0x4000 0000 - 0x4000 03FF	1 KB	Reserved

AMBA:

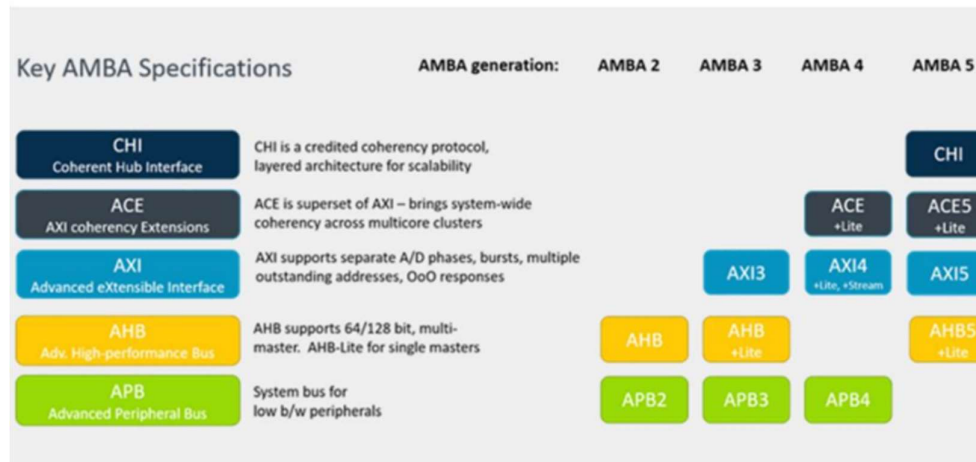
- Advance microcontroller bus architecture.
- Connection Management
- Open standard, how to connect and manage different components or blocks within on SOC.
- 3-distinct busses are described for facilitating communication [AHB, APB and ASB].

AHB [ADVANCED HIGH-PERFORMANCE BUS]

- Backbone of the system designed specifically for high performance, high frequency components. Includes connections of processors, on-chip memories and memory interfaces among others.
- When the Master need to take control of the bus it must first send a request to the arbiter.
- Arbiter grants access based on periodization scheme. This partition scheme is not defined by AMBA and will differ b/t designs.

APB [ADVANCED PERIPHERAL BUS]

- APB is simplified interface for low-frequency system components.
- APB consists of single bus master called APB bridge.
- The bridge is the interface b/t high performance bus and the low frequency peripherals.



AMBA (Advanced Microcontroller Bus Architecture) is a freely-available, open standard for the connection and management of functional blocks in a system-on chip (SoC). It facilitates right-first-time development of multi-processor designs, with large numbers of controllers and peripherals.

AMBA specifications are royalty-free, platform-independent and can be used with any processor architecture. Due to its widespread adoption, AMBA has a robust ecosystem of partners that ensures compatibility and scalability between IP components from different design teams and vendors.

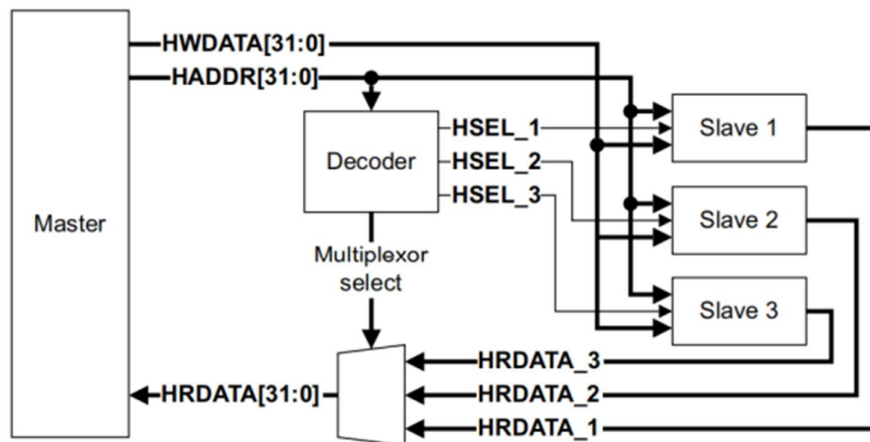
AHB (AMBA High-performance Bus): -

AMBA AHB is a bus interface suitable for high-performance synthesizable designs. It defines the interface between components, such as masters, interconnects, and slaves. AMBA AHB implements the features required for high-performance, high clock frequency systems including:

- Burst transfers.

- Single clock-edge operation.
- Non-tristate implementation.
- Wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB slaves are internal memory devices, external memory interfaces, and high-bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB slaves, for system performance reasons, they typically reside on the AMBA Advanced Peripheral Bus (APB). Bridging between the higher performance AHB and APB is done using an AHB slave, known as an APB bridge.



A single master AHB system design with the AHB master and three AHB slaves. The bus interconnect logic consists of one address decoder and a slave-to-master multiplexor. The decoder monitors the address from the master so that the appropriate slave is selected and the multiplexor routes the corresponding slave output data back to the master. AHB also supports multimaster designs by the use of an interconnect component that provides arbitration and routing signals from different masters to the appropriate slaves.

APB (Advanced Peripheral Bus): -

The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. The APB protocol is not pipelined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol.

The APB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow.

Every transfer takes at least two cycles. The APB can interface with:

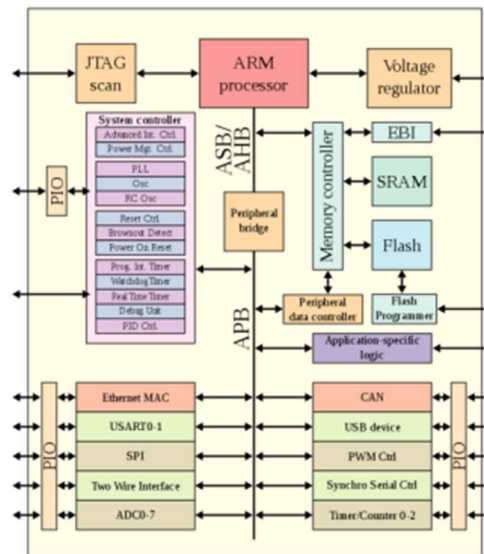
- AMBA Advanced High-performance Bus (AHB)

- AMBA Advanced High-performance Bus Lite (AHB-Lite)
- AMBA Advanced Extensible Interface (AXI)
- AMBA Advanced Extensible Interface Lite (AXI4-Lite)

AHB to APB Bridge: -

The AHB to APB bridge interface is an AHB slave. When accessed (in normal operation or system test) it initiates an access to the APB. APB accesses are of different duration (three HCLK cycles in the EASY for a read, and two cycles for a write). They also have their width fixed to one word, which means it is not possible to write only an 8-bit section of a 32-bit APB register. APB peripherals do not need a PCLK input as the APB access is timed with an enable signal generated by the AHB to APB bridge interface. This makes APB peripherals low power consumption parts, because they are only strobed when accessed.

Importance: -

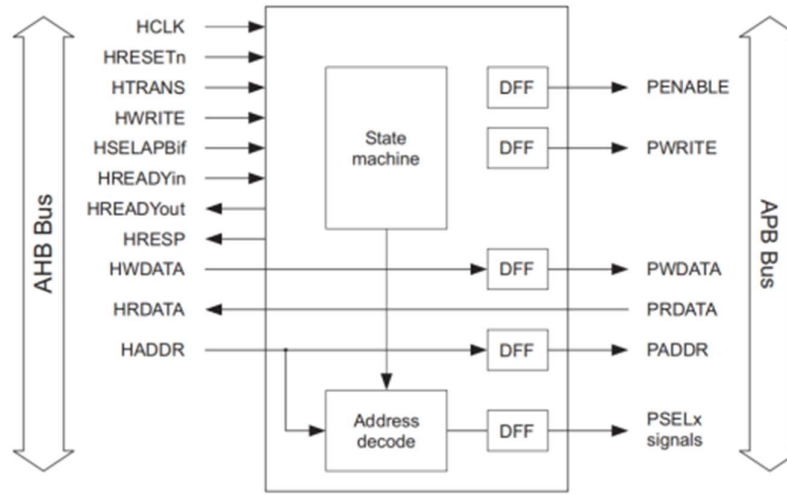


The AHB to APB bridge is an AHB slave, providing an interface between the high speed AHB and the low-power APB. Read and write transfers on the AHB are converted into equivalent transfers on the APB. As the APB is not pipelined, then wait states are added during transfers to and from the APB when the AHB is required to wait for the APB.

It is required to bridge the communication gap between low bandwidth peripherals on APB with the high bandwidth ARM Processors and/or other high-speed devices on AHB. This ensures that there is no data loss between AHB to APB or APB to AHB data transfers. AHB2APB interfaces AHB and APB. It buffers address, controls and data from the AHB, drives the APB peripherals and return data along with response signal to the AHB.

The AHB2APB interface is designed to operate when AHB and APB clocks have the any combination of frequency and phase. The AHB2APB performs transfer of data from AHB to APB for write cycle and APB to AHB for Read cycle. Interface between AMBA high performance bus (AHB) and AMBA peripheral bus (APB). It provides latching of address, controls and data signals for APB peripherals.

Architecture:



RCC:

Reset: There are three types of reset, defined as system reset, power reset and RTC domain reset.

Power reset: A power reset is generated when one of the following events occurs: 1. Power-on/power-down reset (POR/PDR reset) 2. When exiting Standby mode A power reset sets all registers to their reset values.

System reset: System reset sets each register to its reset value, unless otherwise specified in the register description. A system reset is generated when one of the following events occurs:

- A low level on the NRST pin
- Window watchdog event
- Independent watchdog event
- A software reset
- Low-power management reset
- Option byte loader reset
- A power reset

The reset source can be identified by checking the reset flags in the Control/Status register, RCC_CSR

These sources act on the NRST pin and it is always kept low during the delay phase. The RESET service routine vector is fixed at address 0x0000_0004 in the memory map.

The system reset signal provided to the device is output on the NRST pin. The pulse generator guarantees a minimum reset pulse duration of 20 μ s for each internal reset source. In case of an external reset, the reset pulse is generated while the NRST pin is asserted low.

SEMI HOSTING

- Semi hosting is a mechanism that enables code running on an ARM target to communicate and use the input/output facilities on a host computer that is running a debugger.
- Common I/O operations facilitated by semi hosting include printing to the console (stdout) and reading from the console (stdin).
- Semihosting requires interaction with the debugging host environment and it relies on specific fns provided by the debugging tools. These functions allow the embedded system to request services from the host environment.
- Semihosting involves instrumenting the code with special calls or instructions that indicate when the embedded system should use semihosting services.
- Semihosting is useful when the embedded system needs to output debug information, interact with the developer or perform certain operations that are more easily handled by the host environment.
- Semihosting is used to print the debugging messages, logging data and reading input during debugging.
- While semihosting is convenient for debugging purposes. It may introduce dependencies on the host environment and might not be suitable for resource-constrained or real-time systems.

ENABLING SEMIHOSTING:

- Exclude syscalls.c from build
- Head towards debug configuration, instead of GDB select OCD.
- Now go to startup and write in run command [monitor arm semihosting enable] apply close
- Goto Project -> properties -> C/C++ build -> Settings -> MCU GCC linker -> Miscellaneous -> add flags -> -specs=rdimon.specs -lc-lrdimon
- Include a fn in main.c
extern void initialize_monitor_handles(void);
- Now call
initialize_monitor_handles();
- Run the debugger now.

DIFFERENCE BETWEEN UINT32 AND VOLATILE UINT32

uint32_t:

- `uint32_t` is a specific data type defined in the C or C++ programming languages.
- It stands for an unsigned 32-bit integer.
- "Unsigned" means that it only represents non-negative integers (zero and positive values).
- It has a size of 32 bits, allowing it to represent values from 0 to $2^{32} - 1$.
- When you declare a variable as `uint32_t`, you're telling the compiler to allocate 32 bits of memory for that variable.

`uint32_t myNumber = 42;`

- In this example, `myNumber` is a variable of type `uint32_t` that holds the value 42.

volatile uint32_t:

- Adding `volatile` to a variable declaration is a qualifier that informs the compiler about the volatility of the variable.
- It is often used when dealing with variables that can change unexpectedly or outside the normal flow of the program.
- For example, when working with hardware registers or variables modified by interrupt service routines (ISRs).

`volatile uint32_t`

In this case, `sensorValue` is declared as a `volatile uint32_t`. Here's why:

- **Optimization Concerns:**

- Without `volatile`, the compiler might optimize code assuming that the variable doesn't change between certain points in the program.
- This optimization could lead to unexpected behavior when dealing with hardware or external events.

- **Interrupts and Multithreading:**

- In a scenario where an interrupt service routine (ISR) can modify `sensorValue`, declaring it as `volatile` ensures that the compiler doesn't make assumptions about its value outside the regular program flow.
- This is crucial because ISRs can interrupt the normal flow of the program, and the variable can change at any time.

- **Preventing Caching:**

- ### Example Usage:

In summary, using `uint32_t` alone declares a 32-bit unsigned integer, while adding `volatile` to it (`volatile uint32_t`) signals to the compiler that the variable's value may change at any time, often due to external factors such as hardware interrupts. This ensures proper behavior when working with dynamic or asynchronous changes to variables.

```

/* USER CODE BEGIN Header */
/**
 *
 *
 *
 * @file : main.c
 * @brief : Main program body
 *
 *
 *
 *
 * @attention
 *
 * Copyright (c) 2023 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"

```




```

/* Private includes -----
---*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----
---*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----
---*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----
---*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----
---*/

/* USER CODE BEGIN PV */
GPIO_PinState btn_st;

/* USER CODE END PV */

/* Private function prototypes -----
---*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* 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)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----
    ---*/

```



```

/* Reset of all peripherals, Initializes the Flash interface and the
SysTick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    btn_st = HAL_GPIO_ReadPin(GPIOC, Button1_Pin);
    if(btn_st == GPIO_PIN_RESET){
        HAL_GPIO_WritePin(GPIOA, LED_Pin, GPIO_PIN_SET);
    }
    else{
        HAL_GPIO_WritePin(GPIOA, LED_Pin, GPIO_PIN_RESET);
    }
    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Initializes the RCC Oscillators according to the specified parameters
     * in the RCC_OscInitTypeDef structure.
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != 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_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
    Error_Handler();
}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOF_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin : Button1_Pin */
    GPIO_InitStruct.Pin = Button1_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(Button1_GPIO_Port, &GPIO_InitStruct);

    /*Configure GPIO pin : LED_Pin */
    GPIO_InitStruct.Pin = LED_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);

    /* USER CODE BEGIN MX_GPIO_Init_2 */
    /* USER CODE END MX_GPIO_Init_2 */
}

/* 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 */

```

PROGRAMMING DRIVERS FOR GPIO PORTS STM32F030R8

HEADER FILES:

Filename: stm32f03xx.h

```

/*
 * stm32f03xx.h
 *
 * Created on: Nov 23, 2023
 * Author: gampa
 */

#ifndef INC_STM32F03XX_H_
#define INC_STM32F03XX_H_
#include <stdint.h>
#define SRAM_BASE 0x20000000
#define PERI_BASE 0x40000000

//APB ADDRESS
#define APB_BASE_ADD (PERI_BASE + 0x00)
#define AHB1_BASE_ADD (PERI_BASE + 0x20000)
#define AHB2_BASE_ADD (PERI_BASE + 0x8000000)
#define TIM3_BASE_ADD (APB_BASE_ADD + 0x400)
#define TIM6_BASE_ADD (APB_BASE_ADD + 0x1000)
#define TIM7_BASE_ADD (APB_BASE_ADD + 0x1400)

//AHB1 BASE ADDRESS
#define RCC_BASE_ADD (AHB1_BASE_ADD + 0x1000)
#define RCC (RCC_TypeDef_t*)RCC_BASE_ADD

```

```
//AHB2 BASE ADDRESS
#define GPIOA_BASE_ADD (AHB2_BASE_ADD + 0x00)
#define GPIOB_BASE_ADD (AHB2_BASE_ADD + 0x400)
#define GPIOC_BASE_ADD (AHB2_BASE_ADD + 0x800)
#define GPIOD_BASE_ADD (AHB2_BASE_ADD + 0x1000)
#define GPIOF_BASE_ADD (AHB2_BASE_ADD + 0x1400)

#define GPIOA (GPIO_RegDef_t*)GPIOA_BASE_ADD

//Define Pins Mode
#define GPIO_input_mode 0
#define GPIO_output_mode 1
#define GPIO_altfxn_mode 2
#define GPIO_analog_mode 3

//Other Defs

#define SET 1
#define RESET 0

typedef struct
{
    uint32_t CR;
    uint32_t CFGR;
    uint32_t CIR;
    uint32_t APB2RSTR;
    uint32_t APB1RSTR;
    uint32_t AHBENR;
    uint32_t APB2ENR;
    uint32_t APB1ENR;
    uint32_t BDCR;
    uint32_t CSR;
    uint32_t AHBRSTR;
    uint32_t CFGR2;
    uint32_t CFGR3;
    uint32_t CR2;
} RCC_TypeDef_t;

//Structure main
typedef struct
{
    uint32_t MODER;
    uint32_t OTYPER;
    uint32_t OSPEEDR;
    uint32_t OPUPDR;
    uint32_t IDR;
    uint32_t ODR;
    uint32_t BSSR;
}GPIO_RegDef_t;

typedef struct
{
    uint8_t Gpio_Moder;
    uint8_t out_type;
    uint8_t pinNumber;
```



```
uint8_t OSpeed;
uint8_t PuPd;
}GPIO_Config_t;
```

```
#endif /* INC_STM32F03XX_H_ */
```

File name: stm32f03xx_gpio.h

```
/*
 * stm32f03xx_gpio.h
 *
 * Created on: Nov 23, 2023
 * Author: gampa
 */

#ifndef INC_STM32F03XX_GPIO_H_
#define INC_STM32F03XX_GPIO_H_

#include <stm32f03xx.h>
#define OUTPUT_SPEED_LOW 0;
#define OUTPUT_SPEED_HIGH 1;
#define NO_PUPD 0;
#define PU 1;
#define PD 2;
#define reserved 3;
#define OPUPD 0;
#define OOD 1;

typedef struct{
    GPIO_RegDef_t *GPIOx;
    GPIO_Config_t GPIO_Config;
}GPIO_Handle_t;

void GPIO_Init(GPIO_Handle_t GPIO_Handle);
void GPIO_Write_Bit(GPIO_RegDef_t *pGPIOx, uint8_t pinNumber, uint8_t
enod);
int GPIO_Read_Bit(GPIO_RegDef_t *pGPIOx, uint8_t pinNumber);
void GPIO_Toggle_bit(GPIO_RegDef_t *pGPIOx, uint8_t pinNumber);

#endif /* INC_STM32F03XX_GPIO_H_ */
```

SOURCE FILES:

Filename: stm32f03xx_gpio.c

```
/*
 * gpio.c
 *
 * Created on: Nov 23, 2023
 * Author: gampa
 */

#include <stdint.h>
#include <stm32f03xx.h>
#include <stm32f03xx_gpio.h>

RCC_TypeDef_t *GPIO_RCC = RCC;

void GPIO_Init(GPIO_Handle_t GPIO_Handle){
    GPIO_RCC->AHBENR |= (1 << 17);
```



```

    GPIO_Handle.GPIOx->MODER |= (GPIO_Handle.GPIO_Config.Gpio_Moder <<
(2*GPIO_Handle.GPIO_Config.pinNumber));
    GPIO_Handle.GPIOx->OTYPER |= (GPIO_Handle.GPIO_Config.out_type <<
(GPIO_Handle.GPIO_Config.pinNumber));
    GPIO_Handle.GPIOx->OSPEEDR |= (GPIO_Handle.GPIO_Config.OSpeed <<
(2*GPIO_Handle.GPIO_Config.pinNumber));
    GPIO_Handle.GPIOx->OPUPDR |= (GPIO_Handle.GPIO_Config.PuPd <<
(2*GPIO_Handle.GPIO_Config.pinNumber));
}

void GPIO_Write_Bit(GPIO_RegDef_t *pGPIOx, uint8_t pinNumber, uint8_t
enod){
    if(enod == SET)
    {
        pGPIOx -> ODR |= (1 << pinNumber);
    }
    else if(enod == RESET)
    {
        pGPIOx -> ODR |= ~(1 << pinNumber);
    }
}

int GPIO_Read_Bit(GPIO_RegDef_t *pGPIOx, uint8_t pinNumber){
    uint32_t temp;
    temp = pGPIOx -> IDR;
    if(temp & (1<<pinNumber)){
        return SET;
    }
    return RESET;
}

void GPIO_Toggle_bit(GPIO_RegDef_t *pGPIOx, uint8_t pinNumber){
    //Using set reset
    pGPIOx -> ODR ^= (1 << pinNumber);
}

```

APPLICATION:

```

/**

*****
***
* @file           : main.c
* @author        : Auto-generated by STM32CubeIDE
* @brief         : Main program body

*****
***
* @attention
*
* Copyright (c) 2023 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.
*

*****
***

```

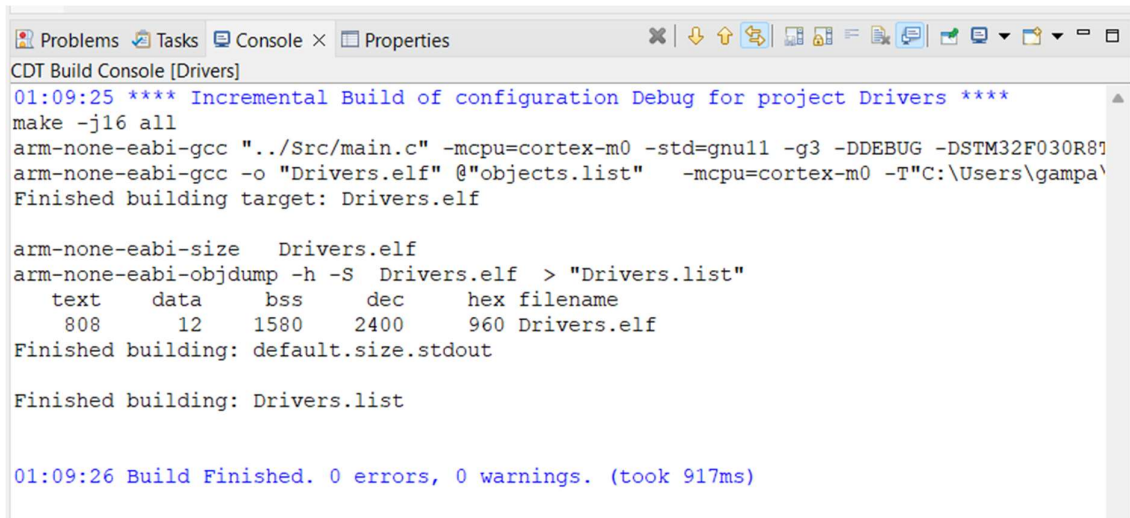
```
#if !defined(__SOFT_FP__) && defined(__ARM_FP)
    #warning "FPU is not initialized, but the project is compiling for an
    FPU. Please initialize the FPU before use."
#endif

#include <stdint.h>
#include <stm32f03xx.h>
#include <stm32f03xx_gpio.h>

GPIO_Handle_t Handle;
void GPIO_Configuration(void) {
    Handle.GPIOx = GPIOA;
    Handle.GPIO_Config.pinNumber = 5;
    Handle.GPIO_Config.Gpio_Moder = GPIO_output_mode;
    Handle.GPIO_Config.OSpeed = OUTPUT_SPEED_LOW;
    Handle.GPIO_Config.PuPd = PU;
    GPIO_Init(Handle);
}

int main(void)
{
    GPIO_Configuration();
    GPIO_Write_Bit(GPIOA, 5, SET);
    GPIO_Write_Bit(GPIOA, 5, RESET);
    /* Loop forever */
    for(;;);
}
```

CONSOLE LOG:



The screenshot shows the IDE's console window with the following output:

```
CDT Build Console [Drivers]
01:09:25 **** Incremental Build of configuration Debug for project Drivers ****
make -j16 all
arm-none-eabi-gcc "../Src/main.c" -mcpu=cortex-m0 -std=gnu11 -g3 -DDEBUG -DSTM32F030R81
arm-none-eabi-gcc -o "Drivers.elf" @"objects.list" -mcpu=cortex-m0 -T"C:\Users\gampa\
Finished building target: Drivers.elf

arm-none-eabi-size Drivers.elf
arm-none-eabi-objdump -h -S Drivers.elf > "Drivers.list"
text    data    bss    dec    hex filename
808     12     1580    2400    960 Drivers.elf
Finished building: default.size.stdout

Finished building: Drivers.list

01:09:26 Build Finished. 0 errors, 0 warnings. (took 917ms)
```


MEMORY REGIONS:

Build Analyzer × Static Stack Analyzer Cyclomatic Comple...

Drivers.elf - /Drivers/Debug - Nov 29, 2023, 1:09:26 AM

Memory Regions Memory Details

Region	Start address	End address	Size	Free	Use
RAM	0x20000000	0x20001fff	8 KB	6.45 KB	1.5%
FLASH	0x08000000	0x0800ffff	64 KB	63.2 KB	82.0%

MEMORY DETAILS:

Build Analyzer × Static Stack Analyzer Cyclomatic Comple...

Drivers.elf - /Drivers/Debug - Nov 29, 2023, 1:09:26 AM

Memory Regions Memory Details

Search

Name	Run address (V...	Load address (LM...
FLASH	0x08000000	
> .isr_vector	0x08000000	0x08000000
> .text	0x080000c0	0x080000c0
> .rodata	0x08000328	0x08000328
> .preinit_array	0x08000328	0x08000328
> .init_array	0x08000328	0x08000328
> .fini_array	0x0800032c	0x0800032c
> .data	0x20000000	0x08000330
RAM	0x20000000	
> .data	0x20000000	0x08000330
> .bss	0x20000004	
> ._user_heap_stack	0x2000002c	



REFERENCES

- [1] https://www.st.com/resource/en/reference_manual/rm0360-stm32f030x4x6x8xc-and-stm32f070x6xb-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
- [2] <https://www.st.com/resource/en/datasheet/stm32f030f4.pdf>
- [3] <https://www.circuitbasics.com/basics-uart-communication/>
- [4] javatpoint.com
- [5] <https://developer.arm.com/documentation/dui0471/g/Bgbjjgij#:~:text=Semihosting%20is%20a%20mechanism%20that,%2C%20and%20disk%20I%2FO.>
- [6] <https://www.ti.com/lit/ug/spruhz6l/spruhz6l.pdf>
- [7] <https://projects.raspberrypi.org/en/pathways/python-intro>
- [8] Wikipedia.com