For memory map details check in the reference manual:

A memory map with text and images

AI-generated content may be incorrect.

A close-up of a table

AI-generated content may be incorrect.

Bus domains:

* There are 4 peripheral busses: APB1, APB2, AHB1, AHB2. They have their own base addresses
* AHB is used for peripherals that need high speed data communication
* APB is used for peripherals that need low speed communication

To reset all the peripheral registers in one shot, use the following register:

A computer error message

AI-generated content may be incorrect.

**GPIO Pin interrupt Configuration:**

1. Pin must be in input mode
2. Configure the edge trigger (RE, FE, RFE)
3. Enable interrupt delivery from peripheral to the processor
4. Identify the IRQ number
5. Configure IRQ priority for the identified IRQ number
6. Enable interrupt reception on that IRQ number
7. Implement the ISR handler

Not: The ISR handlers should be implemented in the application layer of the program by overriding the default handler. The ISR handler then should call the driver supported function to process the request

**STM32 GPIO Pin Specifications**

* Vdd is the main power supply of the microcontroller, provided externally
* Standard operating voltage of Vdd: 1.8<=Vdd<=3.6V. Max voltage on Vdd pin: 4 V
* Vss is ground reference of the microcontroller and should be 0V
* Minimum Vss = -0.3V

A screenshot of a computer

AI-generated content may be incorrect.

A white and black text on a black background

AI-generated content may be incorrect.

STM32 pins:

1. Three-volt tolerant (TT)
2. Five volt tolerant (FT)
3. Three-volt tolerant directly connect to the ADC (TTa)

General operating condition for TT & TTa pin: 0.3 V <= Vin <= Vdd + 0.3 V

FT pin:

* 0.3 V <= Vin <= 5.5 V
* GPIO is FT in input mode (not applicable when GPIO is in output or analog mode)
* FT GPio, max Vin is Vdd + 0.4 V

A screenshot of a table

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Vil: maximum voltage level that is interpreted as a logical 0 by a digital input

ViH: minimum voltage level that is interpreted as as a logical 1 by a digital input

VoL: maximum output voltage low level voltage which can be interpreted as logic low

VoH: the minimum output high level voltage which can be interpreted as logic high

**SPI Details**

* Allows communication between one master and one or more slave devices
* 4 I/O pins for communication with external device
* MISO: Master In/Slave out data – pin used to transmit data in slave mode and receive data in master mode
* MOSI: Master Out/ Slave in data – pin used to transmit data in master mode and receive date in slave mode
* SCK: serial clock output pin for SPI master and input pin for SPI slaves
* NSS: slave select pin – depending on SPI and NSS settings, this pin can be used to select an individual slave device for communication
* Slave’s data comm lines (MISO, MOSI) will be in high impedance state until SS line is pulled to ground.

Slave management:

1. Software slave management

* Set SSM bit in the SPIx\_CR1 reg as 1
* Pin is grounded internally

1. Hardware slave management

* Set SSM bit in the SPI\_CR1 reg as 0
* The NSS pin must be grounded manually

SPI Communication Format:

* Format depends on the clock phase, clock polarity, and data frame format
* CPOL (clock polarity) – controls the idle state value of the clock when noi data is being transferred
* CPOL is reset, the SCLK pin has low-level idle state. If set to high, SCLK pin has high-level idle state
* CPHA controls at which clock edge of the SCLK the data should be sampled by the slave
* Combination of CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge

**A close-up of a data register

AI-generated content may be incorrect.**

A black and white diagram

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

* Note: In SPI communication when master or slave sends 1 byte, it also receives 1 byte in return
* The transmission of 1 byte results in 1 garbage byte collection in the Rx buffer of the master and RXNE flag is set. Do a dummy read and clear the flag

**SPI interrupts:**

During SPI communication, interrupts can be generated by events:

* Transmit Tx buffer ready to loaded; txbuffer is empty and txe flag is set
* Data received in rxbuffer; rxbuffer is full and rxne is set
* Master mode fault (in single master case avoid this error happening)
* Overrun error
* Interrupts can be enabled and disabled (default) separately

A table of events with text

AI-generated content may be incorrect.

**Things to check for SPI debugging steps:**

* Master mode bit is enabled in the configuration register
* SPI peripheral enable bit must be enabled
* SPI peripheral clock must be enabled

**I2C Protocol**

* Serial data communication protocol between integrated circuits
* Based on specification defined by nxp
* Has multi-master capability
* Needs 2 pins for communication
* Master talks to slaves based on slave addresses
* Half duplex
* Max speed is 4Mhz in ultra speed plus only supported in some microcontrollers
* Slave can make master wait by holding the clock down if its busy by clock stretching feature of I2C
* Data rate (number of bits transferred from sender to receiver in 1 sec) is much lesser in I2C compared to SPI
* All I2C peripherals are on APB1 bus

SDA and SCL signals:

* Bidirectional lines connected to positive supply voltage via pull up resistors. When the bus is free both lines are high
* The output stages of devices connected to the bus must have an open-drain or open-collector configuration
* Bus capacitance limits the number of interfaces connected to the bus
* All transaction begin with start (S) and are terminated by a STOP(S)
* A high to low transition on SDA line while SCL is high defines a start condition
* A low to high transition on SDA line while SCL is high defines a stop condition
* START and STOP conditions are always generated by the master. The bus is considered busy after the start condition
* Bus is considered to be free again at a certain time after the stop condition
* When the bus is free another master (if present) can get the chance to claim the bus
* The bus stays busy if a repeated start is generated instead of a stop condition
* Most of MCU’s I2C peripherals support both master and slave mode. Non need to configure the mode because when peripheral generates the start condition, it automatically becomes the master and when it generates the stop condition it goes back to slave mode

I2C protocol: Ack/Nack

* Acknowledge takes place after every byte
* Acknowledge bit allows receiver to signal the transmitter that the byte was successfully received and another byte may be sent
* Master generates all clock pulses, including the acknowledge ninth clock pulse
* When slave receives the ACK from master, that’s an indication for the slave to send on more byte to the master
* When master receives the ACK from the slave, that’s an indication for the master that slave has received data successfully

Data Transition

* Data on the SDA line must be stable during the high period of the clock. The high or low state of the data line can only change when the clock signal on the SCL line is low
* One clock pulse is generated for each data bit transferred
* The start and stop conditions are exceptions. They can transition during the high period of the clock; what differentiates them from the data transfer face of protocol

I2C Clock stretching

* Holding the clock to 0 or ground level
* The moment clock is held at low, whole I2C interface pauses until clock is given up to its normal operation level
* Used when I2C slave is not able to cooperate with the clock speed and the slave needs time to process data

A screenshot of a document

AI-generated content may be incorrect.  
A diagram of a circuit

AI-generated content may be incorrect.

A screenshot of a black and white page

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A black and white document with text

AI-generated content may be incorrect.

The enable control bit must be enabled to generate interrupts for the event flags above

BTF flag and TX and preventing underrun:

* During Txing of data byte, if TXE=1, then data register is empty
* If firmware has not writer any byte to data register before shift register becomes empty (previous byte transmission), then BTF flag will be set and clock will be stretched to prevent the underrun

BTF flag and TX and preventing underrun:

* If RXNE=1, new data is waiting in the data register, and if firmware has not read the data byte yet before shift register is filled with another new data, then BTF flag will be set and clock will be stretched to prevent the overrrun