**Microcontroller Basics**

**By Shrey Bechara**

**GID-158312**

**Understand clock architecture in STM32L4**

**What is Clock?** Clock is a square signal and it is used by the microcontroller to synchronize the execution of instructions and the operation of peripherals such as timers, serial ports, and analog-to-digital converters.

**Clock Sources?** 1-Crystal Oscillator (HSE) (External to MCU)

2-RC Oscillator (HSI) (Internal to MCU)

3-Phase locked loop (PLL) (External to MCU)

Different types of clock signals available in STM32L4 are HSE, HSI, LSE, LSI, PLL.

1. HSE (High-Speed External): HSE is an external clock source that is used to drive the system clock. It can be a crystal oscillator or an external clock generator.
2. HSI (High-Speed Internal): HSI is an internal clock source that is used as an alternative to the HSE. It has a fixed frequency of 16 MHz and is usually used when the HSE is not available.
3. LSE (Low-Speed External): LSE is another external clock source that is used for low-power applications such as real-time clock (RTC) or watchdog timer. It is usually a 32.768 kHz crystal oscillator.
4. LSI (Low-Speed Internal): LSI is an internal clock source that is used for low-power applications such as backup SRAM retention. It has a fixed frequency of 32 kHz.
5. PLL (Phase-Locked Loop): PLL is a clock multiplier that is used to generate a high-frequency clock from a low-frequency clock source. It can be used to generate the system clock from the HSE or HSI.

HSE and HSI are external and internal clock sources used to drive the system clock, respectively.LSE and LSI are external and internal clock sources used for low-power applications. PLL is a clock multiplier that can be used to generate a high-frequency clock from a low-frequency clock source. The choice of clock source and clock frequency depends on the application requirements and the availability of external clock sources.

**Types of Memory used in Microcontroller.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Flash memory** | **RAM** | **EEPROM** | **ROM** | **CACHE** |
| **TYPE** | Non-volatile | volatile | Non- volatile | Non- volatile | volatile |
| **Used for** | Storing Firmware and application code | Storing temporary data and program code during execution | Storing small amount of data that need to be retained even when the power is turned off, such as calibration data or config. setting | Store bootloader  other critical firmware. | Store frequently accessed data and instruction. |
| **Reprogrammable** | yes |  | yes | no |  |
| **Speed** | Slower than RAM | Faster than Flash | Faster than Flash | Faster and more reliable than Flash | Faster than Flash |

**Difference between Flash memory and EEPROM**

|  |  |  |
| --- | --- | --- |
|  | **Flash** | **EEPROM** |
| **Programming and erase cycle** |  | More compare to flash. |
| **Block size** | Large block | Small block therefore making it suitable for storing small amount of data that need to be updated frequently. |
| **Access Time** | Slower access time due to large block and it can be accessed on block by block basis. | It can be accessed on byte by byte basis. |
| **Read and write data speed** |  | Faster, especially when dealing with small amount of data. |
| **Cost** | Cheaper due to its large block and limited no of programming and erase cycle. |  |

**Linker Script File in controller**

A linker script file, also known as a linker command file or a linker map file, is a text file that is used by the linker during the software build process to specify how the various object files and libraries are linked together to create the final executable code that will run on the microcontroller.

The linker script file contains a set of directives that define the memory regions of the microcontroller, such as the program memory, data memory, stack memory, and other memory-mapped peripherals. It also specifies the location and size of each object file and library that is linked to the executable code, as well as the order in which they are linked.

Some of the typical tasks performed by the linker script file include:

1. Memory allocation
2. Section placement
3. Optimization

Overall, the linker script file is an essential part of the software build process for microcontrollers and plays a critical role in ensuring that the final executable code is generated correctly and efficiently and meets the requirements of the target hardware platform and application.

**NVIC and Interrupt Vector table**

**What is a vector table?** It is a table of addresses of system exceptions + interrupts.

NVIC- nested vector interrupts controller- It is responsible for managing & prioritizing interrupt request from various sources in the microcontroller and providing an efficient mechanism for handling them. It uses vector table.

We will try to understand the vector table from reference manual.

NVIC also supports nested interrupts.

What are nested interrupts?

**What happens when the reset button is pressed in microcontroller?**

When reset button is pressed series of operation takes places…

1. Reset circuitry of microcontroller creates a signal and send to processor to start a reset operation.
2. Processer will stop executing instructions.
3. It will reset its internal state including program counter, state register and stack pointer.
4. Processor will also perform series of initialized routines, such as configuring the clock + other system peripherals + resetting the NVIC. + setting up memory & I/O interfaces.
5. Bootloader or startup code will executes.
6. Processor loads the reset vector, which is pointing to the instruction that needs to be executed after the reset operation and that specific instruction is placed at the top most position in the code section in memory.

NOTE: when reset or booting operation is executed the data stored in volatile memory like RAM, register contents will be erased. So it’s advisable that data to be preserved must be stored in flash or EEPROM, or backed up externally.

**Optimization in microcontroller**

Optimization in microcontroller means a process of improving

the performance & efficiency of a program by

reducing size

‘’ memory usage

‘’ execution time

while preserving its functionality and correctness.

Why it is important? As we know that microcontroller have

Limited memory

‘’ processing power

‘’ energy

and always work in real time or strict time requirement constraints. Therefore with the help of optimization we will try to maximize the use of all available resources and try to generate expected performance in most efficient way.

NOTE: At time of optimization we have to keep in mind that the functionality and reliability of the code can’t be compromised.

**Volatile Keyword**

When we declare the keyword as volatile we tell compiler that it value can be also changed outside the program scope using external interrupts or by other task or thread etc

Generally, when we declare any variable and assign value to it, for optimization the compiler will copy the value of data assign to variable in a register and the copied data in register will be used in a further operation + compiler thinks that the value of that variable can be updated using code only.

But we know in microcontroller, due to external sources value of a variable can be changed.

So we declare that specific variable as volatile so when in program execution, variable data is needed, compiler will have latest updated data because it will go to memory instead of register to take the value as it is declared as volatile.

So it will help us in avoiding unexpected output and not needed optimization.

Hence we use volatile keyword.

**Role of Start up file or Bootloader**

The startup file is responsible for performing several essential tasks that are required to initialize the microcontroller's hardware and software components and prepare the system for running the user application.

Some of the most common tasks performed by the startup file is as follows:

1. Initializing the microcontroller's stack pointer.
2. Setting up the microcontroller's interrupt vector table
3. Configuring the microcontroller's clock system
4. Setting up the microcontroller's memory interfaces
5. Initializing the microcontroller's system and peripheral components
6. Jumping to the main user application

Overall, the startup file is a critical component of any embedded software application, and plays a key role in initializing the microcontroller's hardware and software components and preparing the system for running the user application.

**Protocol**A picture containing timeline

Description automatically generated

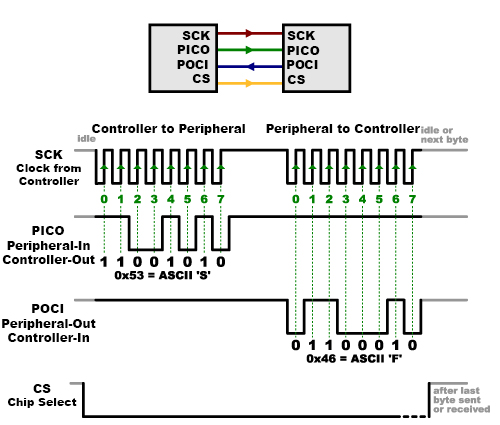
Data size is configurable meaning **number of bits in a single frame can be changed according to requirement(from 5 to 9 bits).**

As we can see UART is a serial communications therefore it will transfer data bit by bit from device 1 to device 2, But device 1 will receive data as parallel from source, similarly device 2 will transfer data to destination parallel.

Text

Description automatically generated with medium confidence

In SPI, it can have 2 configurations 1- independent Chain configuration.

 2- Daisy Chain configuration.

Why SPI mode required it defines the timing of the data transfer and must be the same for both the master and the slave devices for data to be properly transmitted and received.

Timeline

Description automatically generated with medium confidence

Timeline

Description automatically generated

What is pull up or pull down? In pull up, the signal line is connected to voltage Vcc and signal is pulled up i.e. high (1) and make its default or initial state as high (1) when signal line is not driven. This is used to avoid unexpected behavior or noise. Similarly in pull down.

Table

Description automatically generated

What if we connect Multiple slaves with same address? I2C devices may support an alternative address that can be set through hardware pins or software configuration. This allows multiple devices to share the same physical address, but still be individually addressed by the master by using their alternative addresses.

NOTE: It is generally best practice to avoid connecting multiple devices with the same address on a communication bus to avoid potential conflicts and errors.

ADC: Analog to Digital Converter

* It is an electronic component which converts analog signals to digital signals.
* It involves of a 2 main steps:

1. Sampling of an analog signal -- How much sampling rate is required? Ans: Nyquist theorem stats Bandwidth of a signal -> 2\*Sampling rate else it will show quantization error.
2. Quantization of an analog signal.

Watchdog Timer

* It is used to reset the processor, but when?
* Basically, WDT is connected with a processor and processor will periodically send a signal regarding its health to a WDT and it will monitor that.
* If signal from the processor is not received by the WDT in predefined time, then it will understand that processor is misbehaving due to software or hardware bug.
* So, it will reset it.

Real time clock

* It basically gathers the data of all the components in real time and store in very detailed manner i.e. it will have data of a any components behavior with time, date, year etc.
* So, its attached with almost every embedded systems to monitor and to take necessary steps to trigger or to terminate any process at real time for max accuracy and efficiency.

**Pulse-Width Modulation**

* PWM stands for Pulse-Width Modulation, which is a method of digital signal processing used in microcontrollers to generate analog output signals. PWM signals are used to control the speed of motors, the brightness of LEDs, and other applications that require variable output voltages.
* In a microcontroller, PWM signals are generated by output pins that are configured for PWM mode. These pins are connected to a timer/counter peripheral, which generates a series of pulses with a fixed frequency.

**Random Number Generator**- Confusing and have doubt.

**Timer**

Timers are a type of hardware module in a microcontroller that can be used to measure time intervals, generate periodic signals, and synchronize events.

Timers can be used in various applications, such as:

1. Measuring time intervals between events, such as in a pulse width modulation (PWM) system.
2. Generating periodic signals, such as in a real-time clock (RTC) or a system clock.
3. Synchronizing events, such as in a communication protocol or control system.
4. Implementing time-based operations, such as in a delay function or a timeout function.

**FreeRTOS**

Why it is needed when we have general purposes OS like windows, mac os, Linux etc.?

1. It helps us in running multiple tasks concurrently.
2. It guarantees us that it will meet a timing deadline by using scheduler.

Main difference between FreeRTOS and GPOS lies in their real-time capabilities, resource utilization, task management, programming model, and intended applications. FreeRTOS is optimized for real-time processing, with a small code footprint and efficient task management, while GPOS systems prioritize throughput and performance, with a more complex programming model and a broader range of applications.