Resumo ASE

Map Of Content

1. Type of Data Transfer
    1. Polling
    2. Interruptions
    3. DMA (Direct Memory Access)
2. Timers
    1. GPT (General Purpose Timer)
    2. HRT (High-Resolution Timer)
    3. RTC (Real-Time Clock)
    4. Watchdogs
3. Peripherals
    1. General Purpose Input/Output (GPIO)
        - PWM
    2. ADC (Analog-to-Digital Converter)
    3. DAC (Digital-to-Analog Converter)
    4. Wireless Communication
        - Wi-Fi
        - Bluetooth
4. Interfaces
    1. I2C
    2. SPI
    3. UART
5. C
    1. Type Of Variabes
    2. Stack and Heap
    3. Compilation
6. FreeRTOS

    1. FreeRTOS
    2. Power Management
    3. File System on Flash
    4. OTA (Updates Over The Air)
    5. Wireless Connectivity
7. Components used in class
    1. TC74 Sensor
    2. Display 7 segments
    3. SD card Reader
    4. Inversion of Logic Gate with MOSFETs

Type of Data Transfer

- Data Transfer is where you move data from internal components of the microcontroller itself, such as, moving data from registers of the CPU into or from the memory or peripherals.
- Besides you can transfer data between components without CPU interaction as long as the data bus is free.
- There are 3 type of data transfer essentially:
    1. Polling
    2. Interruptions
    3. DMA (Direct Access Memory)

Polling

- The CPU takes initiative, where it starts and controls the data transfer.
- In polling, the CPU actively check the status of a task or peripheral to see if the expecting data is ready to be transferred. However while it's waiting for the peripheral to be ready, it will steal clock cycles where it could be used for execution of instructions.
- **Advantages**:
    - Simple, to implement it, we use continuously loops, checking a flag or register in the peripheral to see if it has data available.
- **Disadvantages**:

    - Can be inefficient for slow peripherals or frequent data transfers
    - The processor wastes time constantly checking the peripheral, even if no data is ready. -> High Overhead

Interruptions

- In interruptions when the peripheral is ready to transfer data it will signal the CPU with a flag informing that the data in the peripheral is available.
- When the flag for interruptions in the CPU is signaled, it will abandon temporarily the execution of the program and execute the code that the interrupt handler is pointing to.
- The data transfer is made by the CPU but the busy waiting disappears, once it only occurs when the peripheral is ready.
- **Advantages:**
    - The CPU only spends time handling data transfer when necessary, improving overall performance.
- **Disadvantages**
    - Might introduce slight delays in handling the interrupt compared to polling continuously.

DMA (Direct Memory Access)

- Data is sent using DMA and DMA Controller, which is only supported by hardware and does not involve the CPU.
- During the data transfer, the DMA as the capability to control the address bus, the data bus and the control bus, because to transfer data it only needs to know:
    - source address where it will read the data
    - destination address where it will write the data
    - and the size in bytes of the data
- To transfer data, the most simple way, is to have a struct with content, # bytes/word that have been transferred and size of data to be transferred.
- **Data Transfer**
    1. DMAC ask for permission to be the bus master for the address,data and control bus
    2. wait for permission granted from the CPU
    3. while transfering:

      1. read a byte/word from the source address to a internal register of the DMA
      2. writes the data of the internal register into the destination address
      3. increments the # of bytes/word transferred
      4. loop while # bytes/words != size
   4. DMA remove the permission of the bus master and the buses and signal the CPU that they are free
- **Advantages:**
   - Reduces processor load significantly.
- **Disadvantages:**
   - Not suitable for all data transfer scenarios, particularly small data transfers.

Timers

- A timer is a hardware or software component where it is designed for precise timing and control within embedded systems. However hardware timers are more precise than software, since it doesn't depend on delays from the Operating System.
- Timers usually made with 3 components, a Count Register, a Comparator and a Reference Register.
   - **Counter:** the counter will count the number of cycle or time units that has passed. It will reset when it overflows or the reset flag is High.
   - **Comparator:** the comparator will compare 2 register where one of them is the time we want to achieve, and the other is output from the counter. When both register are equal it will send a High signal to the output.
   - **Max Count Register (Reference Register):** This register stores the max value that the counter must achieve. After that, it will set to high the reset flag of the counter.
- There are 3 type of timers within ESP32C3:
   1. GPT (General Purpose Timer)
   2. HRT (High-Resolution Timer)
   3. RTC (Real-Time Clock)
- Besides that there are Watchdogs Timers where it will be responsible to restart a program when there is a instruction that block the execution for a long time or

when there is an exception.

GPT (General Purpose Timer)

- GPTimers are 2 54-bits hardware timers, where with its driver we have a versatile set of functionalities for timing and control in your embedded systems.
- The behavior when the internal counter of a timer reaches a specific target value is called a timer alarm.
- When a timer alarms, a user registered per-timer callback would be called.
- GPTs have a built-in **pre-scaler**. This allows you to divide the clock signal feeding the timer, effectively slowing down the counter's increment/decrement rate. This provides more control over timing precision for specific applications.
- In ESP32 the Max Count Register is the Auto-Reload component.
- **Functionalities:**
  - Counting can be set to increment or decrement.
  - Can be set to create a delay within the program, where the program will wait for the interrupt signal from the GPTimer.
  - Creates periodic tasks in a continuous loop, triggering an interrupt at set intervals.
  - Can be used to create PWM (Pulse Width Generation) which based on the duty cycle configured sends a signal with different periods on and off.

HRT (High-Resolution Timer)

- The High-Resolution Timer (HRT) stands out as a powerful tool for applications demanding precise timing and rapid response times. Unlike GPTimer where accuracy and responsiveness are paramount.
- Ability to generate interrupts at high frequencies makes it ideal for real-time applications where quick responses are crucial.
- It is has a counter of 64 bits which provides a more precise reading of the time.

RTC (Real-Time Clock)

- Real-Time Clock (RTC) is a dedicated timer specifically designed to maintain accurate timekeeping even across various sleep modes and resets.

- RTC in the ESP32C3 is a crucial component for maintaining accurate system time, especially in applications that involve sleep modes, resets, or require time-based functionalities.

Watchdogs

- The Interrupt Watchdog is responsible for ensuring that ISRs (Interrupt Service Routines) are not blocked for a prolonged period of time. The TWDT is responsible for detecting instances of tasks running without yielding for a prolonged period.

Interfaces

I2C

- I2C due to is simplicity is used in several types os applications like:
  - Sensores, ADCs and DACs
  - External Memories of microcontrolores etc...
- **Basic Characteristics**
  - Bidirectional, half-duplex, byte oriented
  - Master/Slave transfers
  - Master-transmitter or Master-receiver
  - The communication bus only has 2 lines:
    - Serial data line(SDA)
    - Serial clock line(SCL)
  - Each device is connected to the bus is addressable by software using a single address
  - Addresses of 7 bits standard
- **Terminology**
  - **Transmitter:** device that sends data to the bus
  - **Receiver:** device that receives data from the bus
  - **Master:** device that starts the transfer, generates the clock signal and terminates the transfer
  - **Slave:** device addressed by the Master

    - **Multi-master:** multiple masters can try, at the same time, controlling the bus without corrupting the communications happening
    - **Refereeing:** procedure to assure that if several masters try to control the bus simultaneously, only one of them can is allowed to continue.
    - **Synchronization:** procedure to synchronize the signals of the clocks of two or more devices
- Master controls the SCL line, initiates the data transfer and the controls the addressing of other devices
- Slave is addressed by the master and conditions the the SCL line
- **Data Transfer**
    - The transfer is 8 bit (byte)oriented being transmitted, firstly the MSB
    - To initiate the transfer, the master:
        - Sends a START(S)
        - Next he sends the slave address(7 bit) and the qualification of the operation(RD/WR)
    - After the 8th bit (LSB), the addressed slave generates a ACK in the SDA, with a dominant bit(0)
    - Then the transmitter sends a byte of data
    - After the 8th bit (LSB), the receiver generates a ACK in the SDA, with a dominant bit(0)
    - This cycle of 9 bits repeats itself after each byte of data that is transferred
    - When the transfer is over the master sends a STOP
- **Data transfer- write**- master is the transmitter and the slave is the receiver(esta em cima a explicação)
- **Data transfer- read** -master is the receiver, slave is the transmitter, the only difference is that in the end of the transfer the master sends NACK so the slave stops the transfer
- **Multiple Master**
    - Two or more masters can initiate the transmission in a free bus(after a STOP) at the same time
    - It has to be predicted that the method of deciding which masters has control of the bus and completes the transmission- refereeing the access to the bus
    - The master that looses the process of refereeing it removes itself and only tries to access it again when it is free again
    - In the management of the bus is necessary:
        - Guarantees that the clocks of the masters are synchronized

        - The process that defines that the master that wins access to the bus, that controls the SDA line
    - The synchronization is based in recessive bit

SPI

- Master-Slave architecture E2E, full-duplex, synchronous communication
- SCK generated by master and provides it to the slaves
- Only one master with several slaves, only one slave is selected
- Master initiates and controls the data transfers
- **Pins**
    - **SCK(Clock):** Generated by master that synchronizes the transmission/reception of the data
    - **MOSI (Master Output and Slave Input,SDO on master):** Line of the master that sends data to the slave
    - **MISO (Master Input and Slave Output,SDI on master):** Line of the slave that sends data to the master
    - **SS (Slave Select):** Line of the master that selects the slave to communicate
- **Communication**
    - Master configures the clock to a frequency =/> that the slave he will communicate with handles
    - Master activates the SS\ of the slave he will communicate
    - In each clock cycle, with a postive transition:
        - Master puts in the MOSI line the a bit with the information of what is read by the slave in following opposite clock transition
        - Slave puts in the MOSI line the a bit with the information of what is read by the master in following opposite clock transition
    - Master deactivates the SS\ line and deactivates the clock
        - There is only clock when a transfer is happening
    - Finally, the master and the slave exchange content of the shift registers
- **Types of transfers**
    - SPI works in a data exchange mode, there is always has exchange between shift-registers of master and slave
    - The devices decide to use or discard the information
    - **Transfer Scenarios**

        - **Bidirectional**: Data exchanges in both ways
        - **Master -> Slave (write):** Master transfers data to the slave, accepts/ignores the data
        - **Slave -> Master (read):** masters wants to read the data of the slave, masters transfers to the slave irrelevant info and the slave accepts it or ignores it

---

FreeRTOS

Symetric Multiprocessing

- Symmetric multiprocessing is a computing architecture where two or more identical CPU cores are connected to a single shared main memory and controlled by a single operating system. In general, an SMP system:
    - has multiple cores running independently. Each core has its own register file, interrupts, and interrupt handling.
    - presents an identical view of memory to each core. Thus, a piece of code that accesses a particular memory address has the same effect regardless of which core it runs on.

Tasks

- FreeRTOS provides the following functions to create a task:
    - **Creation:**
        - xTaskCreate() creates a task. The task's memory is dynamically allocated.
        - xTaskCreateStatic() creates a task. The task's memory is statically allocated, i.e., provided by the user.
        - xTaskCreatePinnedToCore() creates a task with a particular core affinity. The task's memory is dynamically allocated.
        - xTaskCreateStaticPinnedToCore() creates a task with a particular core affinity. The task's memory is statically allocated, i.e., provided by the the user
    - **Execution**

    - The anatomy of a task in IDF FreeRTOS is the same as in Vanilla FreeRTOS. More specifically, IDF FreeRTOS tasks:
    - Can only be in one of the following states: Running, Ready, Blocked, or Suspended.
    - Task functions are typically implemented as an infinite loop.
    - Task functions should never return.
- **Deletion**
    - the dual-core nature, there are some behavioral differences when calling vTaskDelete() in IDF FreeRTOS:
        - When deleting a task that is currently running on the other core, a yield is triggered on the other core, and the task's memory is freed by one of the idle tasks.
        - A deleted task's memory is freed immediately if it is not running on either core.
- **SMP Scheduler**
    - Each task is given a constant priority upon creation. The scheduler executes the highest priority ready-state task.
    - The scheduler can switch execution to another task without the cooperation of the currently running task.
    - The scheduler periodically switches execution between ready-state tasks of the same priority in a round-robin fashion. Time slicing is governed by a tick interrupt

---

References

1. https://docs.espressif.com/projects/esp-idf/en/stable/esp32c3/get-started/index.html