Timers and Interrupts

Instructor: Deepak Gangadharan

Introduction

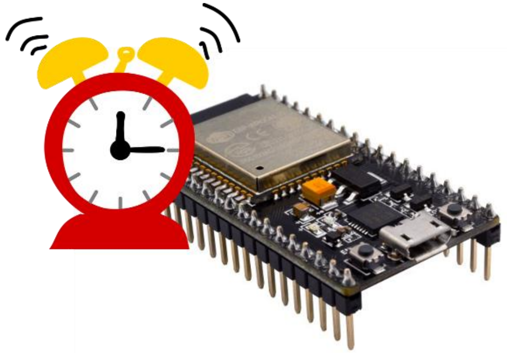
- Suppose that data needs to be collected from a sensor at a fixed frequency
- Assume data needs to be collected from analog sensor connected to ESP32
- Generally the analogRead() execution time on ESP32 is about 10 microseconds
- If the desired data collection interval is much larger, say 1 second, we can introduce a delay of 1 second in the loop for data collection
- The collection interval will be 1.00001 seconds, which may hardly affect the calculations

Introduction (contd...)

- If the desired collection interval is, say 1 millisecond, then introducing a delay will result in a collection interval of 1.01 millisecond, an error of 1%
- As we reduce the data collection interval to 10 microsecond, error percentage keeps increasing
- What can be done to avoid this problem?

Timers

- Used to calculate time using a known clock frequency obtained from the oscillator output
- In effect, timers are just counters provided with a particular clock frequency
- After a particular count is reached, a timer generates an interrupt that can trigger an application or even other timers to register their ticks
- Many Arduino functions use timers, for example:
  - delay()
  - millis()
  - micros()
  - delayMicroseconds()



Timers

- If you are given a counter which counts from 0 to MAX_VALUE, and an interrupt is generated every time the count reaches MAX_VALUE, then by adjusting the frequency of the count, can get accurate time intervals
- Typically MAX_VALUE = 2^{N-1}, N is the number of bits of the counter

General Purpose Timer in ESP32

- ESP32 contains 2 hardware timer groups
- Each group has 2 general purpose 64-bit hardware timers based on 16-bit pre-scalers and 64-bit up/down counters which are capable of being auto-reloaded
- What are pre-scalers?
- They help divide the base clock frequency
- For a 16-bit pre-scaler Min divisor = ? , Max divisor = ?
- ESP32 has a base clock frequency of 80 MHz → Frequency can be adjusted from 1.22 KHz to 80 MHz → Can it be varied continuously?

Configuring a Timer

- Timer Initialization
  - A timer should be initializaed by calling **timer_init()** and passing a structure **timer_config_t** to define how the timer should operate
  - Timer parameters that can be set
    - Clock source – Along with divider, defines the resolution of the timer
    - Divider
    - Mode – Incrementing or Decrementing → Defined using **counter_dir**
    - Counter Enable – If enables, will start incrementing or decrementing immediately after calling timer_init(). Can change behavior with **counter_en**
    - Alarm Enable – Can be set using **alarm_en**
    - Auto Reload

Configuring a Timer

- Timer Control
  - To enable the timer, call the function timer_init() with counter_en set to true, or call timer_start()
  - Specify timer's initial counter value by calling timer_set_counter_value()
  - To check the timer's current value, call timer_get_counter_value() or timer_get_counter_time_sec()
  - To pause the timer at any time, call timer_pause(). To resume, call timer_start().
  - To reconfigure the timer, call timer_init()

Configuring a Timer

- Alarms
  - To set an alarm, call timer_set_alarm_value(), and then enable the alarm using timer_set_alarm()
  - Can also be enabled during initialization when calling timer_init()
  - After the alarm is enabled, and the timer reaches the alarm value, two actions can occur depending on the configuration
    - An interrupt will be triggered if previously configured
    - When auto_reload is enabled, the timer's counter will be automatically reloaded

Interrupts

- A natural way to respond to unexpected events or random events
- An interrupt is automatic transfer of software execution in response to an event that is asynchronous with the current software execution
- This event is called a trigger
- Interrupts the flow of embedded systems program to another set of instructions
- These instructions are called Interrupt Service Routine (ISR)
- Once the ISR is completed, the code is expected to execute at the point of break

Need for interrupts

- Often used to capture events that are random
- Most common example is button press
- Software for what to do is known but do not know when to execute it
- Interrupts can be used to separate non-time-critical functions from the time-critical functions

Thank You