Timer Interrupts Explained with Examples

There are times when you need something to happen, on time, every time.... which becomes virtually impossible without using Timer Interrupts.

These are similar to external interrupts, but instead of firing on an external event, they fire on a timer.

They are so called as they will intterupt the thread of execution after the current instruction completes, and run their code, returning to the next instruction from where it left off when it has finished.

NOTE - the configuration of these varies from platform to platform, and may involve very different code to the below. The concepts of how they work is the same however accross all platforms.

Timer Interrupt Considerations

These routines

- need to be extremely fast to execute, and it is often best to simply set a number of flags or states within the **I**nterrupt **S**ervice **R**outine, then evaluate them when required in your normal thread code in loop().
- only fire when the configured timer overflows
- are restricted to a small number of timers depending on the MCU hardware in use.
- can only set values to variables declared with VOLATILE, which ensures they arent optimized away, and can be used reliably in the ISR and in the main loop() code

NOTE - Timer interrupts may interfere with other functionality (PWM for example) depending on the timer chosen to configure.

e.g. ESP8266 gas 2 x Timers available:

0 (Used by WiFi), 1 is available to configure.

e.g. Arduino Uno has 3 x Timers available:

Timer0 - An 8 bit timer used by Arduino functions delay(), millis() and micros().

Timer1 - A 16 bit timer used by the Servo() library

Timer2 - An 8 bit timer used by the Tone() library

The Mega boards have Timers 3,4,5 which may be used instead

Calculations

As these timers are hardware based, all timing is related to the clock of the timer. These vary board to board, and some common boards are listed below:

ESP32 - 80Mhz

ESP8266 - 80Mhz

ATMega328 - CPU Clock speed (16Mhz)

timer speed (Hz) = Timer clock speed (Mhz) / prescaler

The **prescaler / divider** is what the above frequency is divided by to form a "tick" of the timer (increment its counter). The ISR is then configured to fire after a specific number of ticks.

The prescaler is used, as the timers can only store up to 8/16 bits in their counters, meaning they would overflow every 256/16000000 s (16us) for 8 bit counters, and 65536 / 16000000 s (4us) for 16 bit counters, which is often far more than needed. The prescaler allows this to be scaled to allow longer intervals.

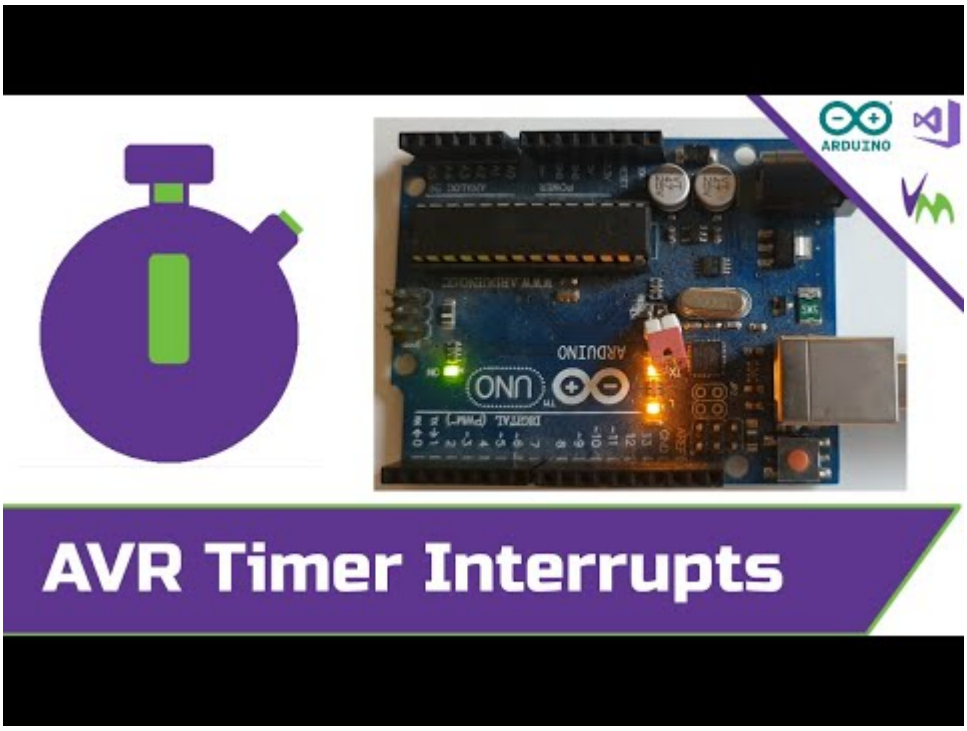
For AVR boards the compare match register also needs configuring, and is worked through in the code below fully.

compare match register = [16,000,000Hz/ (prescaler * desired interrupt frequency)] - 1

! Remember ! that when you use timers 0 and 2 this number must be less than 256, and less than 65536 for timer1

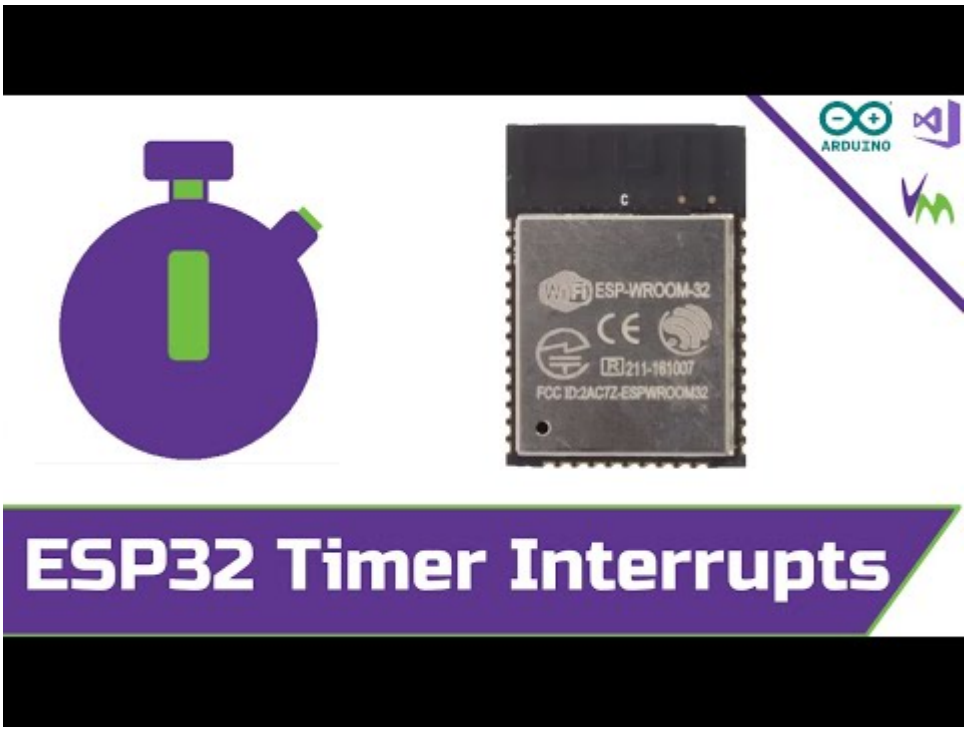
Quick Videos

AVR Timer Interrupts



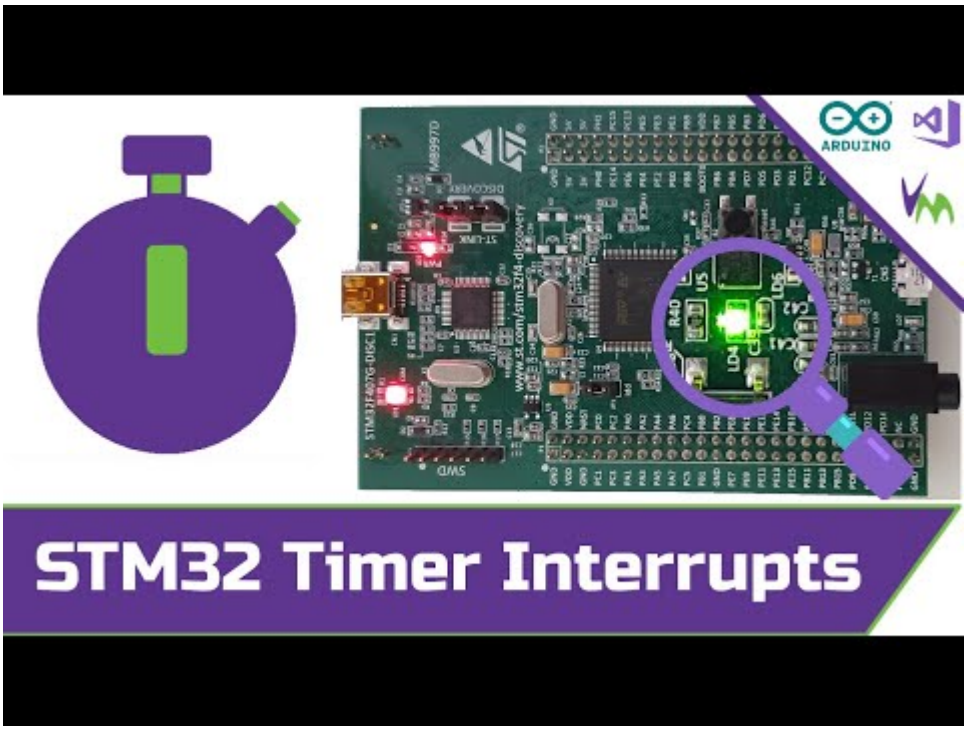
Watch Video At: https://youtu.be/K1yxILe8ivk

ESP32 Timer Interrupts



Watch Video At: https://youtu.be/TE72iO-aDlw

STM32 Timer Interrupts



Watch Video At: https://youtu.be/Z1FE6OtdiSM

Example Software

The software example below will simply show the count of times it has fired, in the Serial Monitor, and is configured to fire once per second.

The code in loop is simply to output to the user, and like with External Interrupts, loop can simply inspect the *interrupts* flag, and perform an action based on this as needed

AVR Example

```
int compareMatchReg;
volatile int interrupts;

void setup()
{
        Serial.begin(115200);
        // initialize timer1
        noInterrupts();           // disable all interrupts
        TCCR1A = 0;
        TCCR1B = 0;

        // Set compareMatchReg to the correct value for our interrupt interval
        // compareMatchReg = [16, 000, 000Hz / (prescaler * desired interrupt
frequency)] - 1

        /* E.g. 1Hz with 1024 Pre-Scaler:
                compareMatchReg = [16, 000, 000 / (prescaler * 1)] - 1
                compareMatchReg = [16, 000, 000 / (1024 * 1)] - 1 = 15624

            As this is > 256 Timer 1 Must be used for this..
        */
        compareMatchReg = 15624;   // preload timer from calc above
        TCNT1 = compareMatchReg;   // preload timer


        /*
        Prescaler:
                (timer speed(Hz)) = (Arduino clock speed(16MHz)) / prescaler
                        So 1Hz = 16000000 / 1 --> Prescaler: 1
                        Prescaler can equal the below values and needs the
relevant bits setting
                        1    [CS10]
                        8    [CS11]
                        64   [CS11 + CS10]
                        256  [CS12]
                        1024 [CS12 + CS10]
        */
        TCCR1B |= (1 << CS12);    // 256 prescaler

        TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
        interrupts();             // enable all interrupts
}

ISR(TIMER1_OVF_vect)        // interrupt service routine
{
        TCNT1 = compareMatchReg;   // preload timer
        interrupts++;
        Serial.print("Total Ticks:");
        Serial.println(interrupts);
}

void loop()
{
        // your program here...
}
```

ESP8266 Example

```
#include <ESP8266WiFi.h>
#include <Ticker.h>

Ticker timer;

volatile int interrupts;

// ISR to Fire when Timer is triggered
void ICACHE_RAM_ATTR onTime() {
        interrupts++;
        Serial.print("Total Ticks:");
        Serial.println(interrupts);
        // Re-Arm the timer as using TIM_SINGLE
        timer1_write(2500000);//12us
}

void setup()
{
        Serial.begin(115200);
        //Initialize Ticker every 0.5s
        timer1_attachInterrupt(onTime); // Add ISR Function
        timer1_enable(TIM_DIV16, TIM_EDGE, TIM_SINGLE);
        /* Dividers:
                TIM_DIV1 = 0,   //80MHz (80 ticks/us - 104857.588 us max)
                TIM_DIV16 = 1,  //5MHz (5 ticks/us - 1677721.4 us max)
                TIM_DIV256 = 3  //312.5Khz (1 tick = 3.2us - 26843542.4 us max)
        Reloads:
                TIM_SINGLE      0 //on interrupt routine you need to write a new
value to start the timer again
                TIM_LOOP        1 //on interrupt the counter will start with the
same value again
        */

        // Arm the Timer for our 0.5s Interval
        timer1_write(2500000); // 2500000 / 5 ticks per us from TIM_DIV16 ==
500,000 us interval
}

void loop()
{
}
```

ESP32 Example

```
volatile int interrupts;
int totalInterrupts;

hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR onTime() {
        portENTER_CRITICAL_ISR(&timerMux);
        interrupts++;
        portEXIT_CRITICAL_ISR(&timerMux);
}

void setup() {

        Serial.begin(115200);

        // Configure Prescaler to 80, as our timer runs @ 80Mhz
        // Giving an output of 80,000,000 / 80 = 1,000,000 ticks / second
        timer = timerBegin(0, 80, true);
        timerAttachInterrupt(timer, &onTime, true);
        // Fire Interrupt every 1m ticks, so 1s
        timerAlarmWrite(timer, 1000000, true);
        timerAlarmEnable(timer);
}

void loop() {
        if (interrupts > 0) {
                portENTER_CRITICAL(&timerMux);
                interrupts--;
                portEXIT_CRITICAL(&timerMux);
                totalInterrupts++;
                Serial.print("totalInterrupts");
                Serial.println(totalInterrupts);
        }
}
```

STM32 Example

This and more examples for the STM32Duino core can be found at this GitHub Repository.

The PWM function can also be used to achieve this in one line of code (if a simple LED On/Off Operation)

```
#if defined(LED_BUILTIN)
#define pin  LED_BUILTIN
#else
#define pin  D2
#endif

void Update_IT_callback(void)
{ // Toggle pin. 10hz toogle --> 5Hz PWM
  digitalWrite(pin, !digitalRead(pin));
}


void setup()
{
#if defined(TIM1)
  TIM_TypeDef *Instance = TIM1;
#else
  TIM_TypeDef *Instance = TIM2;
#endif

  // Instantiate HardwareTimer object. Thanks to 'new' instanciation,
HardwareTimer is not destructed when setup() function is finished.
  HardwareTimer *MyTim = new HardwareTimer(Instance);

  // configure pin in output mode
  pinMode(pin, OUTPUT);

  MyTim->setOverflow(10, HERTZ_FORMAT); // 10 Hz
  MyTim->attachInterrupt(Update_IT_callback);
  MyTim->resume();
}


void loop()
{
  /* Nothing to do all is done by hardware. Even no interrupt required. */
}
```

Further Uses

Due to the speed of execution interrupts can be used for setting up clocks for data lines (I2C for example), or to configure LED fading or any other number of time specific events or communications.

Other Types of Interrupt

Interrupts are also used to create realtime input handlers through External Interrupts, so the ISR code can fire when an event occurs, irrelevant of where in your code it is at the time.