

Ejemplo de temporizador MCU AVR® de 8 bits



Esta página ilustra varios métodos para configurar los temporizadores en un MCU AVR ® de 8 bits . Se proporciona una descripción general de los temporizadores en un AVR antes de presentar el ejemplo paso a paso.

Requisitos para ejecutar los ejemplos prácticos

Requisitos de hardware




- ATmega328PB Mini tablero explicado
- Cable micro USB

Herramienta	Sobre		
 (http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)	Mini kit de evaluación ATmega328PB Xplained	 (http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)	 (https://www.microchipdirect.com/prc)

El kit de evaluación ATmega328PB Xplained Mini es una plataforma de hardware para evaluar el microcontrolador Atmel ATmega328PB. NO se necesita un kit de depurador externo para ejecutar estos laboratorios. El ATmega328PB tiene un depurador incorporado completamente integrado.

Este ejercicio práctico demuestra cómo configurar los diferentes temporizadores.

Herramientas de software

Herramienta	Sobre	Instaladores				Instrucciones de instalación
		Windows	linux	Mac OS X		
 Atmel® Studio Integrated Development Environment	 (/atstudio:start)	 (http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)				 (/install:atstudio)

Additional Files

Files



Xplained Board User Guide (http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide_UserGuide.pdf)

Overview of Timers on 8-bit AVR MCUs

ATmega328PB microcontroller has five Timer/Counters:

Timer 0	TC0	8-bit Timer/counter with PWM
Timer 1	TC1	16-bit Timer/counter with PWM and Asynchronous Operation.
Timer 2	TC2	8-bit Timer/counter with PWM and Asynchronous Operation.
Timer 3	TC3	16-bit Timer/counter with PWM and Asynchronous Operation.
Timer 4	TC4	16-bit Timer/counter with PWM and Asynchronous Operation.

Definitions:**Register Nomenclature**

BOTTOM	The counter reaches the BOTTOM when it becomes zero (0x00 for 8-bit counters and 0x0000 for 16bit counters)
MAX	The counter reaches its maximum value when it becomes 0x0F (15 decimal) for 8-bit counters and 0x00FF (255 decimal) for 16-bit counters
TOP	The counter reaches the TOP when its value becomes equal to the highest value possible. The TOP value can be assigned fixed value MAX or the value stored in the OCRxA register. This assignment is dependent upon the mode of operation

Timer 0 - 8-bit Timer/Counter with PWM

Timer/Counter0 (TC0) is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and PWM support.

TC0 Registers

The Timer/Counter 0 register (TCNT0) and Output Compare TC0x registers (OCR0x) are 8-bit registers. Interrupt request signals are all visible in the Timer Interrupt Flag Register 0 (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register 0 (TIMSK0).

Name: TCCR0B
Offset: 0x45
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x25

Bit	7	6	5	4	3	2	1	0
	FOC0A	FOC0B			WGM02	CS02	CS01	CS00
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

(/local--files/8avr:avrtimer/TCCR0B.PNG)

TC0 Timer/Counter Clock Sources

TC0 can be clocked by an internal or an external clock source. The clock source is selected by writing to the Clock Select (CS02:0) bits in the Timer/Counter Control Register (TCCR0B).

Bits 2:0 – CS0n: Clock Select [n = 0..2]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

TC0 Counter Unit

Depending on the mode of operation used, T0 is cleared, incremented, or decremented at each timer clock (clkT0). clkT0 can be generated from an external or internal clock source, selected by the Clock Select bits (CS0[2:0]).

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the T0 Control Register A (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B).

Bits 1:0 – WGM0n: Waveform Generation Mode [n = 1:0]

Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) Mode, and two types of Pulse Width Modulation (PWM) modes.

Table 1-2 Waveform Generation Mode Bit Description

Mode	WGM2:0	Mode of Operation	TOP	OCR0x Update	TOV flag set on
0	0 0 0	Normal	0xFF	Immediate	MAX
1	0 0 1	PWM Phase Correct	0xFF	TOP	BOTTOM
2	0 1 0	CTC	OCRA	Immediate	MAX
3	0 1 1	Fast PWM	0xFF	BOTTOM	MAX
4	1 0 0	Reserved	~	~	~
5	1 0 1	PWM Phase Correct	OCRA	TOP	BOTTOM
6	1 1 0	Reserved	~	~	~
7	1 1 1	Fast PWM	OCRA	BOTTOM	MAX

**Note:**

1. MAX = 0xFF
2. BOTTOM = 0x00

Modes of Operation for TC0

The mode of operation determines the behavior of TC0 and the Output Compare pins. It is defined by the combination of the Waveform Generation mode bits and Compare Output mode bits in the Timer/Counter control Registers A and B (TCCR0B.WGMn2, TCCR0A.WGM01, TCCR0A.WGM00, and TCCR0A.COM0x[1:0]).

Available modes of operation for TC0 are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode

Clear Timer on Compare Match Mode

In Clear Timer on Compare mode (WGM0[2:0]=0x2), the OCR0A Register is used to manipulate the counter resolution: the counter is cleared to ZERO when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution.

The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared. An interrupt can be generated each time the counter value reaches the TOP value by setting the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value.

The waveform frequency is defined by the following equation:

$$f_{\text{OCnx}} = \frac{f_{\text{clk.I/O}}}{2 \cdot N \cdot (1 + \text{OCRnx})}$$

(/local--files/8avr:avrtimer/freqz0.PNG)

N represents the prescaler factor (1, 8, 64, 256, or 1024).

TC1, TC3, & TC4 - 16-bit Timer/Counters with PWM

The 16-bit Timer/Counter units allow accurate program execution timing (event management), wave generation, and signal timing measurement.

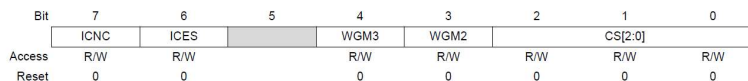
Registers (TC1, TC3, TC4)

- The Timer/Counter (TCNTn), Output Compare Registers (OCRA/B), and Input Capture Register (ICRn) are all 16-bit registers.
- The Timer/Counter Control Registers (TCCRnA/B) are 8-bit registers and have no CPU access restrictions.
- Interrupt requests (abbreviated to Int.Req. in the block diagram) signals are all visible in the Timer Interrupt Flag Register (TIFRn). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSKn).

Timer/Counter Clock Sources (TC1, TC3, TC4)

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select bits in the timer/Counter control Register B (TCCRnB.CS[2:0]).

Name: TCCR1B, TCCR3B, TCCR4B
Offset: 0x81 + n*0x10 [n=0..2]
Reset: 0x00
Property: -



(/local--files/8avr:avrtimer/TCCR1B.PNG)

Bits 2:0 – CS[2:0]: Clock Select [n = 0..2]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

Counter Unit (TC1, TC3, TC4)

TC1, TC3, and TC4 are programmable 16-bit bi-directional counters.

Each 16-bit counter is mapped into two 8-bit I/O memory locations: Counter High (TCNTnH) containing the upper eight bits of the counter, and Counter Low (TCNTnL) containing the lower eight bits.

Depending on the selected mode of operation, the counter is cleared, incremented, or decremented at each timer clock (clkTn). The clock clkTn can be generated from an external or internal clock source, as selected by the Clock Select bits in the Timer/Counter Control Register B (TCCRnB.CS[2:0]).

The counting sequence is determined by the setting of the Waveform Generation mode bits in the Timer/Counter Control Registers A and B (TCCRnB.WGM[3:2] and TCCRnA.WGM[1:0]).

Modes of Operation (TC1, TC3, TC4)

The mode of operation is determined by the combination of the Waveform Generation mode (WGM[3:0]) and Compare Output mode (TCCRnA.COMx[1:0]) bits.

Available modes of operation are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode
- Phase and Frequency Correct PWM Mode

Fast PWM Mode

The Fast Pulse Width Modulation or Fast PWM modes (modes 5, 6, 7, 14, and 15, WGM[3:0]= 0x5, 0x6, 0x7, 0xE, 0xF) provide a high frequency PWM waveform generation option. The Fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM.

In Fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM[3:0] = 0x5, 0x6, or 0x7), the value in ICRn (WGM[3:0]=0xE), or the value in OCRnA (WGM[3:0]=0xF). The counter is then cleared at the following timer clock cycle.

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot (1 + TOP)}$$

(/local--files/8avr:avrtimer/freqz1.PNG)



Note:

- The “n” in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the “x” indicates Output Compare unit (A/B).
- N represents the prescale divider (1, 8, 64, 256, or 1024).

TC2 - 8-bit Timer/Counter2 with PWM and Asynchronous Operation

Timer/Counter2 (TC2) is a general purpose, dual channel, 8-bit Timer/Counter module.

TC2 Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2A and OCR2B) are 8-bit registers. Interrupt request signals are all visible in the Timer Interrupt Flag Register (TIFR2). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK2).

Name: TCCR2B
Offset: 0xB1
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	FOC2A	FOC2B			WGM22	CS22	CS21	CS20
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

(/local--files/8avr:avrtimer/TCCR2B.PNG)

TC2 Clock Sources

TC2 can be clocked by an internal synchronous or an external asynchronous clock source:
The three Clock Select bits (CS2:CS0) select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

TC2 Counter Unit

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clkT2). clkT2 can be generated from an external or internal clock source, selected by the Clock Select bits (CS2[2:0]).

The counting sequence is determined by the setting of the WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2A) and the WGM22 bit located in the Timer/Counter Control Register B (TCCR2B).

TC2 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM2[2:0]) and Compare Output mode (COM2x[1:0]) bits.

Available modes of operation are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode

Normal Mode

In the Normal mode (WGM22:0 = 0) the counting direction is always up (incrementing), without having the counter cleared. The counter will roll over to 0x00 when it passes its maximum 8-bit value (TOP = 0xFF).

In normal operation, the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 Flag, in this case, behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV2 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written any time.



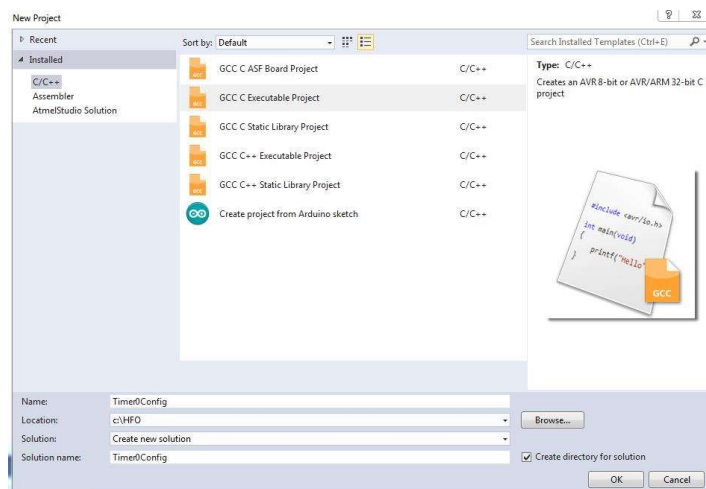
Regardless of which mode is used the programmer needs to remember two things:

1. The timer has to be started by selecting the clock source.
2. If interrupts are used they must be enabled.

! Step-by-Step

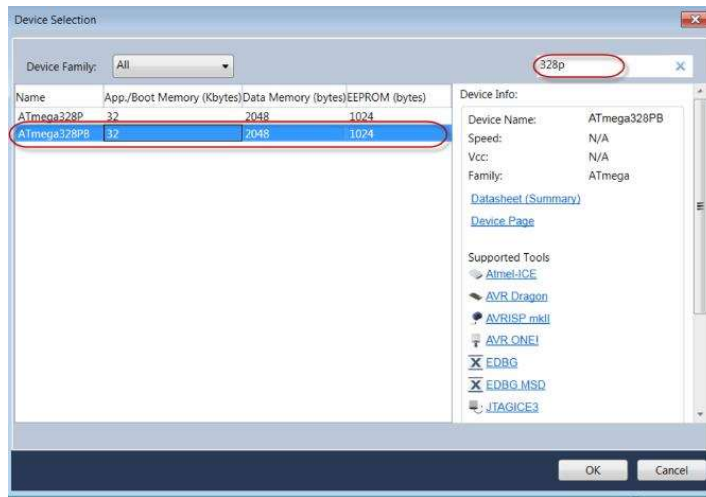
Project Creation

- 1 Open Atmel Studio 7
- 2 Select **File > New > Project**
- 3 In the **New Project** window select **GCC C executable Project** and give the project name to "Timer0Config". Set the location of the project files. (This example uses "C:\HFO\TimersConfigurations"). Click **OK**.



(/local--files/8avr:avrtimer/new-project.jpg)

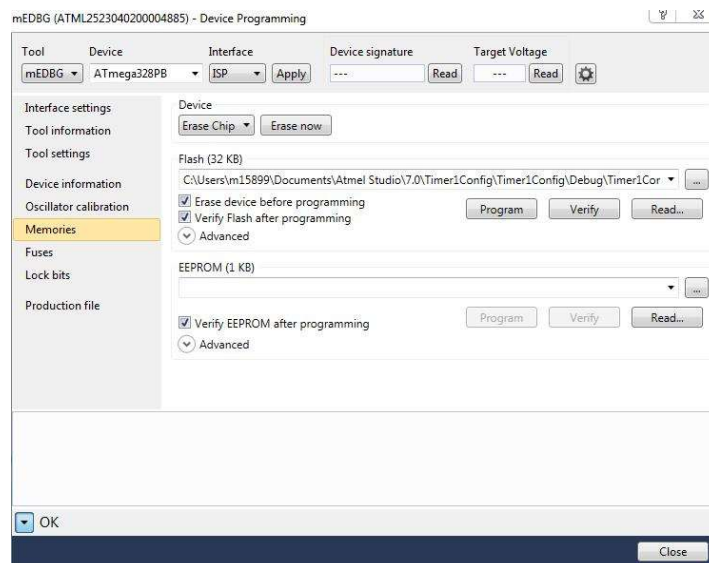
- 4 In the search bar of the **Device Selection** window type "328p" then select the device **Atmega328PB**, click **OK**.



(/local--files/8avr:avrtimer/select-device.jpg)

Programming the board

- 5 Select **Build > Build Solution** from the top menu to compile the code. You will see a "BUILD SUCCEEDED" message in the output window of Studio 7.
- 6 Select **Tools > Device Programming**, make sure the settings are as in the picture below and click **Apply**.



(/local--files/8avr:avrtimer/program.jpg)

- 7 Select **Memories > Program**. Wait until the "Verifying Flash...OK" message to appear.

Project Modifications



avr/io.h and avr/interrupt.h must be **"#included"** in main.c for this project to compile and run.

- io.h provides for the definition of all of the AVR peripherals.
- interrupt.h is required for all applications using interrupts.

8 Modification 1 - Blinking a LED using TC0

In this example Timer 0 blinks the LED connected to PB5 every 32 ms.

- a Modify the main function by adding the following code:

```

1  <font></font>?
2  int main(void)<font></font>
3  <font></font>
4  {<font></font>
5      //set RB5 as output<font>
6      DDRB |= 1 << DDRB5; <font>
7  <font></font>
8      //call TMR0 initializati
9      init_TC0(); <font></font>
10 <font></font>
11     //enable interrupt<font>
12     sei(); <font></font>
13 <font></font>
14     while (1)<font></font>
15     {<font></font>
16         //main loop<font></f
17     }<font></font>
18 }<font></font>
19 <font></font>

```

b Create the function `init_TC0 ()` in `main.c`. Add the following code:

```

1  <font></font>?
2  void init_TC0(void)<font></for
3  {<font></font>
4  <font></font>
5      // Set the Timer Mode to (
6      TCCR0A |= (1 << WGM01);<font>
7  <font></font>
8      // Set the value that you
9      OCR0A = 0xF9; //249<font>-
10 <font></font>
11     //Set the ISR COMPA vect<
12     TIMSK0 |= (1 << OCIE0A);
13 <font></font>
14     // set prescaler to 1024 ;
15     TCCR0B |= (1 << CS00) | (
16 }<font></font>
17 <font></font>

```

The preceding code does the following:

- Configures Timer 0 in CTC mode by setting the corresponding WGM bit in TCCR0A register;
- Sets the TOP value in the OCR0A register calculated using the equation 1 to blink the LED at 30 kHz;
- Enables the Timer Interrupt;
- Sets the pre-scaler to 1024 by configuring the CS00 and CS02 bits in TCCR0B register;

c Add the interrupt service routine to `main.c`:

```

1  <font></font>?
2  ISR (TIMER0_COMPA_vect) <font>
3  {<font></font>
4      //event to be executed e
5      counter++;<font></font>
6      if (counter == MyTimerCo
7          counter = 0;<font></
8          PORTB ^= 1 << PORTB5
9      }<font></font>
10 }<font></font>
11 <font></font>

```



The use of the constant *MyTimerConstant* and the variable *counter* is optional. If used, changing the value of MyTimer Constant will alter the time it takes the LED to blink.

9 Modification 2 - Using TC1 in PWM mode to dim the LED

This example uses TC1 to generate a PWM signal which is fed through an I/O pin to drive an LED. Changing the PWM duty-cycle will result in a change in the brightness of the LED.

a Modify `main()` by adding the following code.


```

1  <font></font>
2  int main(void) <font></font>
3  {
4      //set direction of pin F
5      DDRB |= 1 << DDRB1; <font></font>
6      <font></font>
7      init_TC1_pwm(); <font></font>
8      <font></font>
9      //enable global interrupt
10     sei(); <font></font>
11     <font></font>
12     while (1) <font></font>
13     {
14         //main loop <font></font>
15     } <font></font>
16     <font></font>
17     <font></font>
18     <font></font>

```

For configuring the direction bit of pin PB1, write the DDB1 bit to logic 1 in the DDRB register. On the ATmega328PB Xplained Mini Board, LED is connected to PB5 and PWM is generated on PB1. To view the LED dimming, connect a wire from PB5 to PB1 on the ATmega328PB Xplained Mini Board, as shown below.

- b Create the `init_TC1_pwm()` function before the main loop with the following code:

```

1  <font></font>
2  void init_TC1_pwm(void) <font></font>
3  {
4      //clear OCnA on compare match
5      TCCR1A = (1 << COM1A1); <font></font>
6      <font></font>
7      //the counting sequence is
8      TCCR1A |= (1 << WGM10); <font></font>
9      TCCR1B |= (1 << WGM12); <font></font>
10     <font></font>
11     //256 prescaler clock selected
12     TCCR1B |= (0 << CS10) | (0 << CS11); <font></font>
13     <font></font>
14     //enable interrupts
15     TIMSK0 |= (1 << OCIE1A); <font></font>
16     <font></font>
17     <font></font>

```

`init_TC1_PWM()` performs the following"

- Sets the bits COMA1=1 and COMA0=0 (non-inverting mode from datasheet table Compare Output Mode, Fast PWM in the TCCR1A register;
- Configures Bits 1:0 – WGM[1:0]:Waveform Generation Mode accordingly for Fast PWM, 8-bit mode;
- Verifies the configuration for the clock source and prescaler to be used by the Timer/Counter, which decides the frequency of the PWM. Internal clock source divided by 256 is configured by writing to the CS10 and CS12 bits in TCCR1B register;
- Enables the Timer Interrupt;

- c Add the ISR function after the main loop to execute dimming LED event:

```

1  <font></font>
2  ISR (TIMER1_COMPA_vect) //
3  {
4      duty_cycle--; <font></font>
5      if (duty_cycle == 0) {
6          duty_cycle = 0xFF; <font></font>
7      } <font></font>
8      OCR1A = duty_cycle; <font></font>
9      <font></font>
10     <font></font>
11     <font></font>

```

The register OC1RA decides the duty cycle of PWM. To dim the LED gradually decrease the duty cycle.

10 Blinking an LED using TC2

For this example, Timer 2 (TC2) is configured to blink the LED connected to PB5 at a period of 3 seconds.

- a Modify the main function by adding the following code:

```
1  <font></font>
2  int main(void)<font></font>
3  {<font></font>
4      //set direction of PB5 a
5      DDRB |= 1 << DDRB5;<font>
6  <font></font>
7      /* Timer clock = I/O clc
8      TCCR2B = (1 << CS22) | (
9  <font></font>
10     /* Clear overflow flag *
11     TIFR2 = 1 << TOV2;<font>
12  <font></font>
13     /* Enable Overflow Inter
14     TIMSK2 = 1 << TOIE2;<font>
15  <font></font>
16     // enable global interr
17     sei();<font></font>
18  <font></font>
19     while (1)<font></font>
20     {<font></font>
21         // Main loop<font></font>
22     }<font></font>
23 }<font></font>
24 <font></font>
```

- Configures the direction bit of pin PB5, write the DDB5 bit to logic 1 in the DDRB register;
- Sets the prescaler to 1024 by configuring the CS20, CS21 and CS22 bits in TCCR2B register;
- Clears Overflow flag - TOV2 is cleared by writing a logic one to the flag.
- Enables global interrupts by calling `sei();`

- b Enable the Overflow Timer Interrupt;

Add the ISR function after the main loop to execute blinking LED event:

```
1  <font></font>
2  ISR (TIMER0_COMPA_vect) <font></font>
3  {<font></font>
4      counterTimer2++;<font></font>
5      if(counterTimer2 == MyTi
6      {<font></font>
7          counterTimer2 = 0;<font>
8          PORTB ^= 1 << PORTB5
9      }<font></font>
10 <font></font>
```

Set the PORTB register to blink the LED



MyTimer2Constant is optional. This constant can be changed to alter the time it takes the LED to blink.