

8-bit AVR® A More Complete PWM Driver

Last modified by Microchip on 2023/11/09 09:02

In this video

<https://www.youtube.com/watch?v=uA410-EC_Mo>

**Contents**

* [In this video](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/pwm-driver/#HInthisvideo)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/pwm-driver/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/pwm-driver/#HLearnMore)
* Create functions: PWM\_Increase\_duty(), PWM\_Decrease\_duty(), PWM\_BrightDim()
* Use enumerators to keep track of the direction of PWM duty cycle change.
* Declare our function prototypes.

Will will expand the options available for lighting our LED with the code changes outlined below.  We will include the ability to not only increase the brightness but to dim it as well.  We will clean up our code a bit by declaring function prototypes and using enumerators to keep track of our PWM duty cycle changes.

Procedure

Increase the LED Brightness

Take the logic that brightens the LED which is currently inside an ISR and copy and then delete it. Create a new function called PWM\_Increase\_duty just below the ISR and paste the copied logic into it.  It should look like the accompanying code.

void PWM\_Increase\_duty(void)  
{  
    uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty < period)  
    {  
        duty++;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}

Lower the LED Brightness

Copy this entire function and paste just below it and rename it PWN\_Decrease\_duty.  Now modify the logic so that it tests whether duty is greater than 0 and decrements Duty is true.  It should be like the accompanying code.

void PWM\_Decrease\_duty(void)  
{  
    uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty > 0)  
    {  
        duty--;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}

Determine the Direction of the Duty Cycle

Now we create a new function called PWM\_BrightDim with the following logic and place it into the TIMER0 ISR.  The code allows us to assign the direction of the duty cycle.  UP and DOWN will be declared as enums toward the top of the code which will be shown in the final code version.

void PWM\_BrightDim(void)  
{   uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
    **static** uint8\_t direction;  
      
    **switch** (direction)  
    {  
        **case** UP:  
            **if** (++duty == (period - 1))  
                    direction = DOWN;  
            **break**;  
        **case** DOWN:  
            **if** (--duty == 2)  
                direction = UP;  
            **break**;  
    }  
    OCR1B = duty;  
}

Declare the Functions

You may notice that MPLAB® X is showing errors in our code.  We need to declare functions toward the beginning to make the errors go away. Place the following code below the #defines which will declare our enums and our functions.

**enum** {UP, DOWN};  
  
void PWM\_Decrease\_duty(void);  
void PWM\_Decrease\_duty(void);  
void PWM\_BrightDim(void);  
void Timer\_Frequency(uint8\_t freq);  
void PWM\_Init(void);  
void milliS\_timer(uint8\_t milliS);

Final Code

Here is the final code that contains the elements shown above.

#define F\_CPU 16000000UL   
#include *<avr/io.h>*  
#include *<util/delay.h>*  
#include *<avr/interrupt.h>*  
  
#define LED\_ON  PORTB |= (1<<PORTB5)  
#define LED\_OFF PORTB &= ~(1<<PORTB5)  
#define LED\_TOGGLE  PINB |= (1<<PINB5)  
  
**enum** {UP, DOWN};  
  
void PWM\_Decrease\_duty(void);  
void PWM\_Decrease\_duty(void);  
void PWM\_BrightDim(void);  
void Timer\_Frequency(uint8\_t freq);  
void PWM\_Init(void);  
void milliS\_timer(uint8\_t milliS);  
  
  
ISR(TIMER1\_COMPA\_vect)  
{  
   LED\_ON;   
}  
  
ISR(TIMER1\_COMPB\_vect)  
{  
   LED\_OFF;   
}  
  
ISR(TIMER0\_COMPA\_vect)  
{  
    PWM\_BrightDim();  
}  
  
void PWM\_Increase\_duty(void)  
{  
  uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty < period)  
    {  
        duty++;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}  
  
void PWM\_Decrease\_duty(void)  
{  
  uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty > 0)  
    {  
        duty--;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}  
  
void PWM\_BrightDim(void)  
{   uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
    **static** uint8\_t direction;  
      
    **switch** (direction)  
    {  
        **case** UP:  
            **if** (++duty == (period - 1))  
                    direction = DOWN;  
            **break**;  
        **case** DOWN:  
            **if** (--duty == 2)  
                direction = UP;  
            **break**;  
    }  
    OCR1B = duty;  
}  
  
void Timer\_Frequency(uint8\_t freq)  
{  
    TCCR1B |= (1 << CS12) | (1 << WGM12);  *//Set clock source & set mode to CTC*    TIMSK1 |= (1 << OCIE1A); *//Enable the CTC interrupt*    OCR1A = (F\_CPU/(freq\*2\*256)-1);  
}  
  
void PWM\_Init(void)  
{  
    TCCR1B |= (1 << CS10) | (1 << WGM12);  *//No Prescaler & set mode to CTC*    TIMSK1 |= (1 << OCIE1A) | (1 << OCIE1B); *//Enable the CTC interrupt*    OCR1A = 800;  
    OCR1B = 400;  
}  
  
void milliS\_timer(uint8\_t milliS)  
{  
    TCCR0A |= (1 << WGM01);                *//set to CTC mode*    TCCR0B |= (1 << CS02) | (1 << CS00);   *//set prescaler to 1024*  
    OCR0A = milliS \* 7.8125 - 1;  
    TIMSK0 |= (1 << OCIE0A);               *//enable the interrupt*}  
             
             
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*    DDRB &= ~(1<<DDB7); *//set PB7 as an input pin*      
    milliS\_timer(2);      
  
    PWM\_Init();  
      
    sei();                  *//Enable global interrupts*      
    **while** (1) {  
                 
    }  
}

Program the Device

Press the Make and Program Device Main Project button at the top of the MPLAB X IDE.

Program the Device

You should observe the device increasing and decreasing brightness.

8-bit AVR® A More Complete PWM Driver

Last modified by Microchip on 2023/11/09 09:02

In this video

<https://www.youtube.com/watch?v=uA410-EC_Mo>

**Contents**

* [In this video](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/pwm-driver/#HInthisvideo)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/pwm-driver/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/pwm-driver/#HLearnMore)
* Create functions: PWM\_Increase\_duty(), PWM\_Decrease\_duty(), PWM\_BrightDim()
* Use enumerators to keep track of the direction of PWM duty cycle change.
* Declare our function prototypes.

Will will expand the options available for lighting our LED with the code changes outlined below.  We will include the ability to not only increase the brightness but to dim it as well.  We will clean up our code a bit by declaring function prototypes and using enumerators to keep track of our PWM duty cycle changes.

Procedure

Increase the LED Brightness

Take the logic that brightens the LED which is currently inside an ISR and copy and then delete it. Create a new function called PWM\_Increase\_duty just below the ISR and paste the copied logic into it.  It should look like the accompanying code.

void PWM\_Increase\_duty(void)  
{  
    uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty < period)  
    {  
        duty++;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}

Lower the LED Brightness

Copy this entire function and paste just below it and rename it PWN\_Decrease\_duty.  Now modify the logic so that it tests whether duty is greater than 0 and decrements Duty is true.  It should be like the accompanying code.

void PWM\_Decrease\_duty(void)  
{  
    uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty > 0)  
    {  
        duty--;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}

Determine the Direction of the Duty Cycle

Now we create a new function called PWM\_BrightDim with the following logic and place it into the TIMER0 ISR.  The code allows us to assign the direction of the duty cycle.  UP and DOWN will be declared as enums toward the top of the code which will be shown in the final code version.

void PWM\_BrightDim(void)  
{   uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
    **static** uint8\_t direction;  
      
    **switch** (direction)  
    {  
        **case** UP:  
            **if** (++duty == (period - 1))  
                    direction = DOWN;  
            **break**;  
        **case** DOWN:  
            **if** (--duty == 2)  
                direction = UP;  
            **break**;  
    }  
    OCR1B = duty;  
}

Declare the Functions

You may notice that MPLAB® X is showing errors in our code.  We need to declare functions toward the beginning to make the errors go away. Place the following code below the #defines which will declare our enums and our functions.

**enum** {UP, DOWN};  
  
void PWM\_Decrease\_duty(void);  
void PWM\_Decrease\_duty(void);  
void PWM\_BrightDim(void);  
void Timer\_Frequency(uint8\_t freq);  
void PWM\_Init(void);  
void milliS\_timer(uint8\_t milliS);

Final Code

Here is the final code that contains the elements shown above.

#define F\_CPU 16000000UL   
#include *<avr/io.h>*  
#include *<util/delay.h>*  
#include *<avr/interrupt.h>*  
  
#define LED\_ON  PORTB |= (1<<PORTB5)  
#define LED\_OFF PORTB &= ~(1<<PORTB5)  
#define LED\_TOGGLE  PINB |= (1<<PINB5)  
  
**enum** {UP, DOWN};  
  
void PWM\_Decrease\_duty(void);  
void PWM\_Decrease\_duty(void);  
void PWM\_BrightDim(void);  
void Timer\_Frequency(uint8\_t freq);  
void PWM\_Init(void);  
void milliS\_timer(uint8\_t milliS);  
  
  
ISR(TIMER1\_COMPA\_vect)  
{  
   LED\_ON;   
}  
  
ISR(TIMER1\_COMPB\_vect)  
{  
   LED\_OFF;   
}  
  
ISR(TIMER0\_COMPA\_vect)  
{  
    PWM\_BrightDim();  
}  
  
void PWM\_Increase\_duty(void)  
{  
  uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty < period)  
    {  
        duty++;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}  
  
void PWM\_Decrease\_duty(void)  
{  
  uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
      
    **if** (duty > 0)  
    {  
        duty--;  
    }  
    **else**  
    {  
        duty = 0;  
    }  
    OCR1B = duty;    
}  
  
void PWM\_BrightDim(void)  
{   uint16\_t period = OCR1A;  
    uint16\_t duty =  OCR1B;  
    **static** uint8\_t direction;  
      
    **switch** (direction)  
    {  
        **case** UP:  
            **if** (++duty == (period - 1))  
                    direction = DOWN;  
            **break**;  
        **case** DOWN:  
            **if** (--duty == 2)  
                direction = UP;  
            **break**;  
    }  
    OCR1B = duty;  
}  
  
void Timer\_Frequency(uint8\_t freq)  
{  
    TCCR1B |= (1 << CS12) | (1 << WGM12);  *//Set clock source & set mode to CTC*    TIMSK1 |= (1 << OCIE1A); *//Enable the CTC interrupt*    OCR1A = (F\_CPU/(freq\*2\*256)-1);  
}  
  
void PWM\_Init(void)  
{  
    TCCR1B |= (1 << CS10) | (1 << WGM12);  *//No Prescaler & set mode to CTC*    TIMSK1 |= (1 << OCIE1A) | (1 << OCIE1B); *//Enable the CTC interrupt*    OCR1A = 800;  
    OCR1B = 400;  
}  
  
void milliS\_timer(uint8\_t milliS)  
{  
    TCCR0A |= (1 << WGM01);                *//set to CTC mode*    TCCR0B |= (1 << CS02) | (1 << CS00);   *//set prescaler to 1024*  
    OCR0A = milliS \* 7.8125 - 1;  
    TIMSK0 |= (1 << OCIE0A);               *//enable the interrupt*}  
             
             
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*    DDRB &= ~(1<<DDB7); *//set PB7 as an input pin*      
    milliS\_timer(2);      
  
    PWM\_Init();  
      
    sei();                  *//Enable global interrupts*      
    **while** (1) {  
                 
    }  
}

Program the Device

Press the Make and Program Device Main Project button at the top of the MPLAB X IDE.

Program the Device

You should observe the device increasing and decreasing brightness.

AVR® Microcontrollers Introduction and Key Training Application

Last modified by Microchip on 2023/11/10 11:09

Hands-On Training

This is a series of both video and hands-on training focused on the fundamentals of AVR® microcontroller development. Projects are developed from scratch, using the datasheet, AVR LibC, and later app notes, as primary programming references (START is not used).

In the video series, several AVR peripherals are introduced, including GPIO, timer/counter, USART, and ADC. In the process, the training evolves toward building an example application that samples an analog light sensor with ADC, then proportionally updates a PWM duty cycle based on the ADC sensor reading. An averaged sensor value is then sent over USART to a PC terminal.

Supported Evaluation Kit

* [ATmega328PB Xplained Mini](https://www.microchip.com/en-us/development-tool/atmega328pb-xmini)

Next Lesson

* [Finding Documentation and Turning on an LED](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

# AVR®: Finding Documentation and Turning On an LED

Last modified by Microchip on 2023/11/09 09:02

**Contents**

* [In This Video](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HInThisVideo)
* [Reference Materials](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HReferenceMaterials)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HLearnMore)

## In This Video

## <https://youtu.be/6JEbnyz2Mac>

* Find the device datasheet, "Xplained Mini User's Guide", and its schematics.
* Start a new project in MPLAB® X IDE.
* Demonstrate how to efficiently use the datasheet to understand how to configure a pin and turn on an LED.

## Reference Materials

* [ATmega328PB Xplained Mini](https://www.microchipdirect.com/productsearch.aspx?keywords=ATMEGA328PB-XMINI)
* [MPLAB® X IDE](https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide)

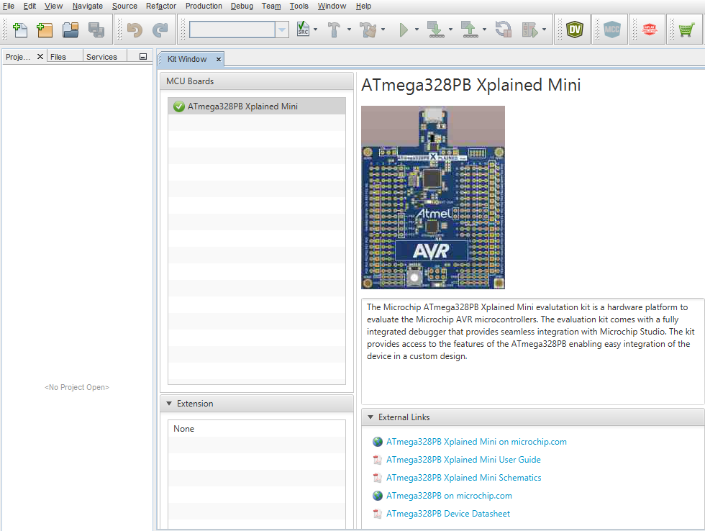
[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

## Procedure

##### Start MPLAB X

Start MPLAB X and plug in the Xplainined Mini to one of your USB ports.

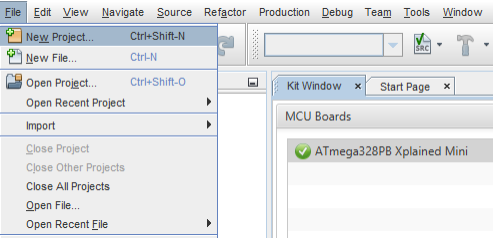
If the Xplained Mini is enumerated properly, you should see the following screen with the Kit Window opened showing the ATmega328PB Xplained Mini board.



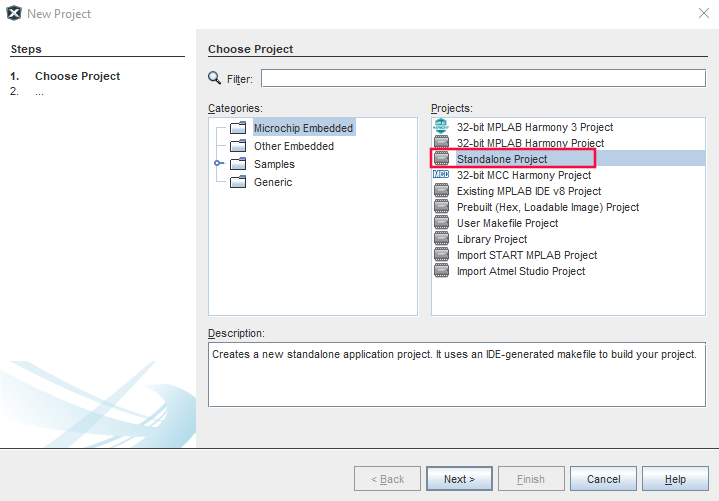
[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Create the Project

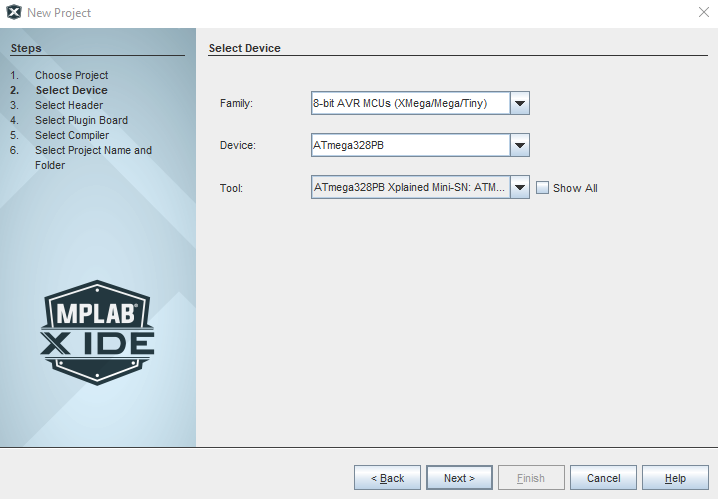
Select **File > New Project** from the menu at the top.



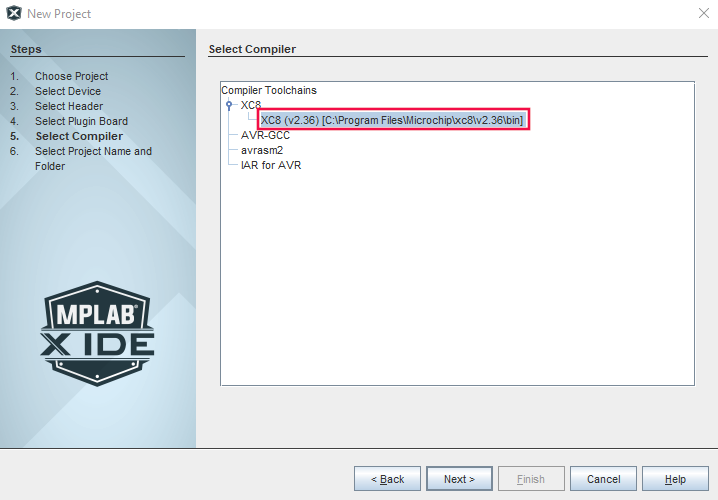
A New Project dialog box will appear.  Select **Standalone Project** from the options and then select **Next**.



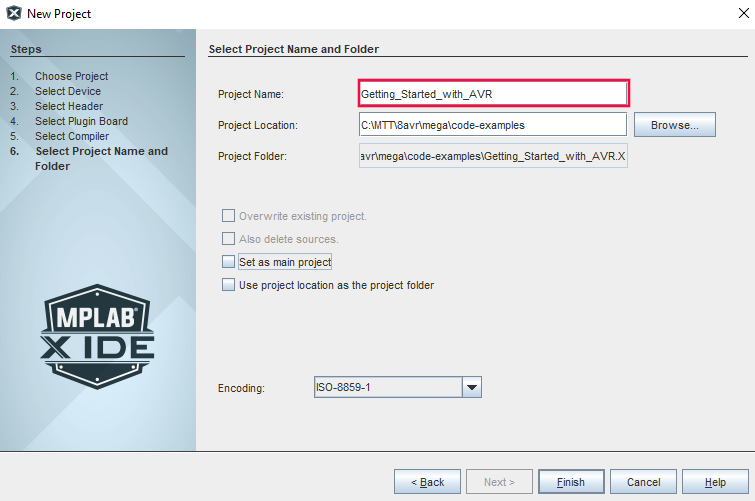
At the **Select Device** screen, select the following for the device **Family**, **Device**, and **Tool**.  Select **Next**.



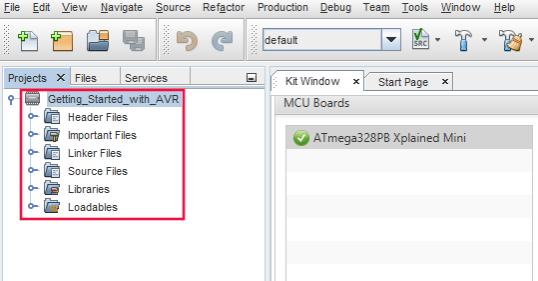
Now select the **XC8 Compiler Toolchain** and select**Next**



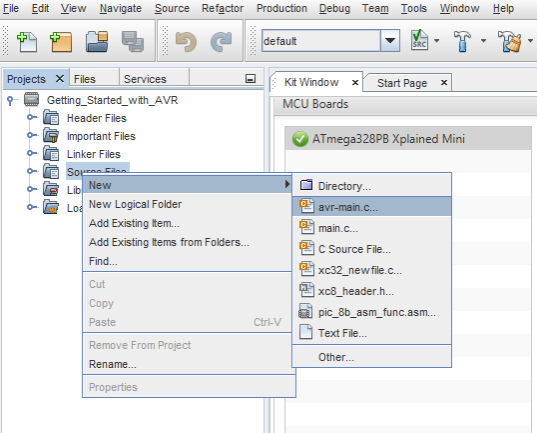
Next, we'll create a name for our project.  Enter "Getting\_Started\_with\_AVR" in the **Project Name** dialog box and then select **Finish**

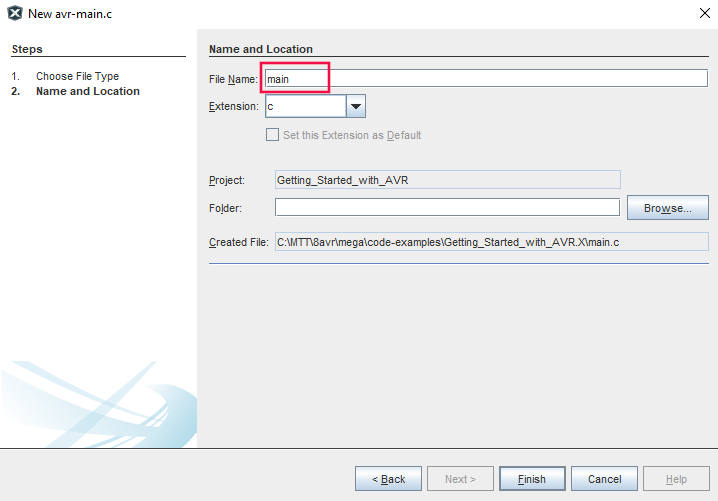


MPLAB X IDE has created our project and populated the user interface with several folders.

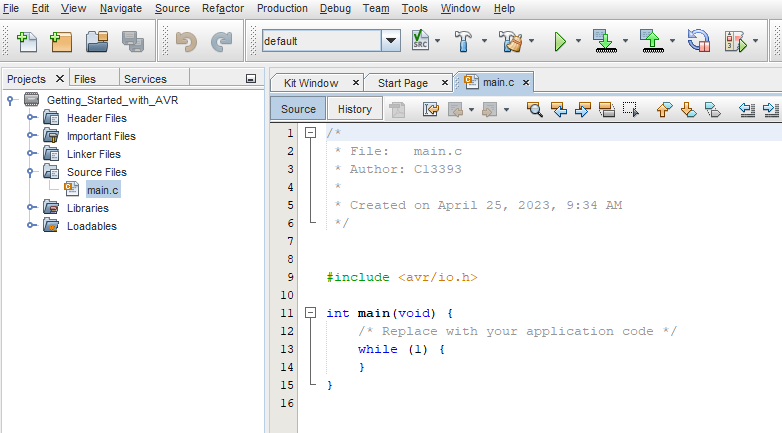


Right-click on the **Source Files** folder and hover over **New** and then select **avr-main.c**



Change the name of the File Name at the top of the dialog box to main and then select **Finish.**

MPLAB X has now created a main document as part of our project and it should be open for you to edit.  This file is where we'll enter our code to turn on the led of the ATmega328PB Xplained Mini.  As you can see, it populated the file with some of the things we will need in our project.  The avr/io.h header file has been included which contains definitions we'll need along with the main while loop for our application code to reside in.

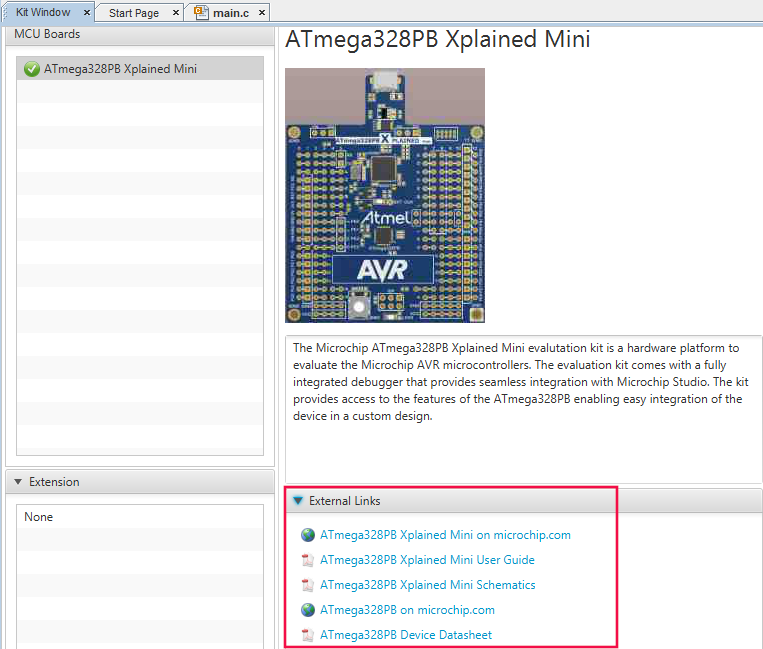


[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Find the documents

###### **the documents we need to get started programming the board**

In the kit window that opened when the ATmega328PB was connected, there is a section indicated by External Links located below the board image and description.  If not already expanded, select that section to expose the links to the ATMega328PB Mini User Guide, schematics, and datasheet.

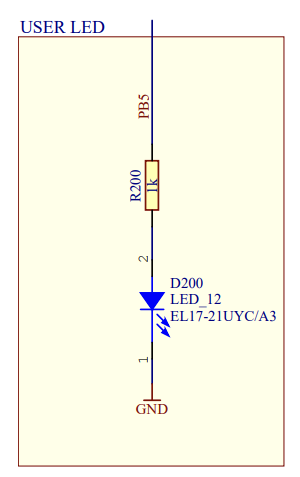


[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### LED PIN

* **Find what pin the LED is connected to so that we can turn on the correct pin.**

The ATmega328PB Xplained Mini User manual as well as the ATmega328PB Xplained Mini Schematic shows that the LED is connected to pin PB5 through a 1k ohm resistor. In order to turn the LED on, we will need to make PB5 high.



In order to determine how to do that, we will consult the ATmega328PB datasheet.  PB5 is a GPIO pin (General Purpose Input Output) and the best place to start in the datasheet is section 17 which discusses I/O-Ports and how to configure them.

From the datasheet, each port pin consists of three register bits:  DDxn, PORTxn, and PINxn.  The DDxn bit in the DDRx register selects the direction of this pin.

Note:  A lowercase "x" represents the numbering letter for the port, and the lowercase "n" represents the bit number.  For example, PORTB5 is bit number 5 on Port B.  In C Programming we can shorten this to PB5.

[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Set Pin Direction

First we set the direction of the pin and in this case, we want it to be an output. In C Programming, we can accomplish this with the following line of code:

DDRB |= (1 << PB5); *// set PB5 as output pin*

###### **Breakdown**

* DDRB is a register in the ATmega328PB microcontroller that controls the data direction of the digital I/O pins on port B. Each bit in the register corresponds to a pin on the port, with bit 0 corresponding to pin PB0, bit 1 to pin PB1, and so on.
* |= is the bitwise OR assignment operator. It performs a logical OR operation between the current value of PORTB and the value on the right-hand side of the operator and then stores the result back into PORTB.
* (1 << PB5) is a bit-shift operation that moves the value 1 to the left by the number of bits specified by the PB5 constant, which is 5. This evaluates to 0b00100000.

When we perform the bitwise OR operation between DDRB and (1 << PB5), the result will be that the bit corresponding to pin PB5 in DDRB is set to 1, while the other bits remain unchanged. This sets the data direction of pin PB5 to be an output.

Now we want to set the pin high so that it can drive the Led.

PORTB |= (1 << PB5); *// set PB5 high*

###### **Breakdown**

* PORTB is a register in the ATmega328PB microcontroller that controls the state of the digital output pins on port B. Each bit in the register corresponds to a pin on the port, with bit 0 corresponding to pin PB0, bit 1 to pin PB1, and so on.
* |= is the bitwise OR assignment operator. It performs a logical OR operation between the current value of PORTB and the value on the right-hand side of the operator, and then stores the result back into PORTB.
* (1 << PB5) is a bit-shift operation that moves the value 1 to the left by the number of bits specified by the PB5 constant. In this case, PB5 is defined as the value 5, so (1 << PB5) evaluates to 0b00100000.

This line of code sets the bit corresponding to pin PB5 in the PORTB register to 1 while leaving the other bits unchanged. This effectively sets the output voltage of pin PB5 to the logic HIGH level.

[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Combine the Code

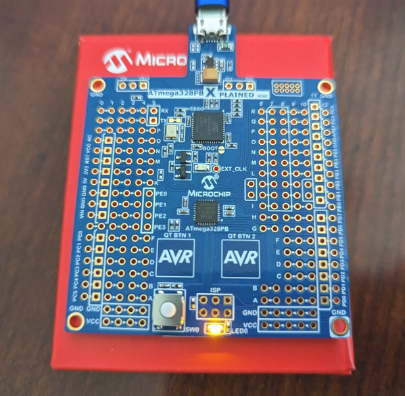
* **Let's put these lines of code together in MPLAB X.**

#include *<avr/io.h>*  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
       PORTB |= (1 << PB5); *// set PB5 high*      
    }  
}

Place the two lines of code into the main file and select the button indicated below to make and program the target device.

Program the Target

If everything was done as indicated, the led on the board should light continuously.



[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

## Learn More

* [Introduction and Key Training Application](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/introduction/)
* [Flashing an LED at a Specific Frequency](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/)

Better Coding Practice for USART Send Using a Sendflag

Last modified by Microchip on 2023/11/09 09:02

AVR®: Better Coding Practice for USART Send Using a Sendflag

In This Video:

<https://youtu.be/FHS5rvHLFIc>

* Debug the IRQ-driven Universal Synchronous Asynchronous Receiver Transmitter (USART) communications to understand what happens when a string is sent.
* Demonstrate that the USART Data Rate Empty IRQ has hit a number.
* Create a global flag that is checked in the while loop to determine when the average should be sent over the USART, enabling us to move the USART send function out of the Analog-to-Digital Converter (ADC) Interrupt Service Routine (ISR).
* Reveal that we still have the same functionality but have implemented better coding practices.

Learn More

* [Sending Averaged ADC Sample Over USART](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-avg-adc-over-usart/)
* [Understanding USART TX Pin Activity Using the Data Visualizer](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-tx-data-visualizer/)

Flashing an LED at a Specific Frequency

Last modified by Microchip on 2023/11/10 11:09

**Contents**

* [AVR®: Flashing an LED at a Specific Frequency](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/#HAVRAE:FlashinganLEDataSpecificFrequency)
* [Objectives](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/#HObjectives)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/#HLearnMore)

AVR®: Flashing an LED at a Specific Frequency

<https://youtu.be/aeXMPVQ3MuM>

Objectives

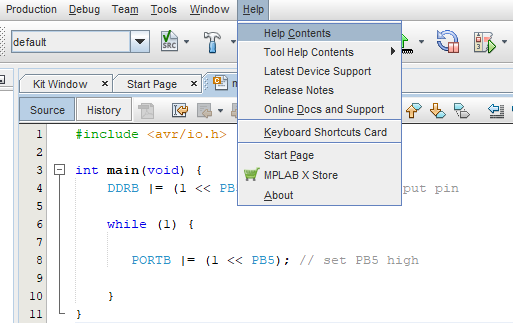
* Introduce XCA8 Library functions to set a precise delay based on the AVR CPU frequency.
* Find the clock source of the ATmega328P on the Xplained Mini.
* Writing to the PIN register to efficiently toggle a GPIO pin.

Procedure

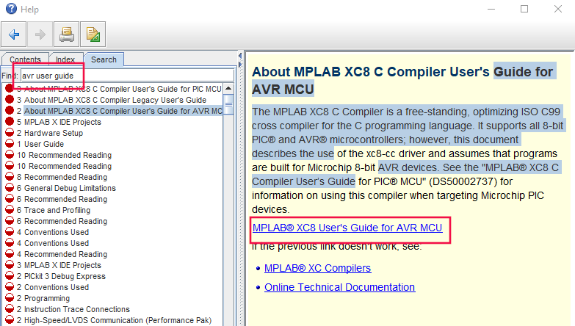
**Library Function**

Find the library function we need to set a delay in our code.

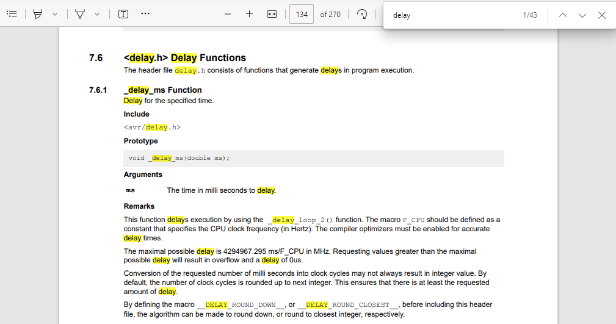
With the project from the [previous lesson](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/), open in MPLAB® X IDE, select **Help,** and then the **Help Contents** button in the top menu.



When the Help Contents opens, select the **Search**tab and type "avr user guide" in the **Find** field as shown. Then hit **Enter** and select **About MPLAB XC8 C Compiler User's Guide for AVR MCU**. Click the top link to the document on the right side of the panel.



This will open the XC8 user's guide for AVR MCUs where we will search for "delay" in the document.  This will bring us to section 7.6 of the guide which describes the use of the delay function



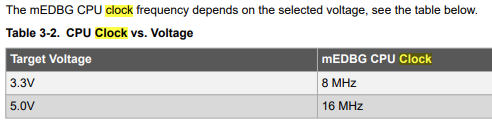
**Delay Function**

Add the Delay function.

The user guide tells us that in order to use this function, we'll need to include the header avr/delay.h file.  Place the following line of code towards the top of our Getting\_Started\_with\_AVR project like this:

#include *<avr/io.h>*  
#include *<avr/delay.h>*  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
       PORTB |= (1 << PB5); *// set PB5 high*      
    }  
}

The user guide also tells us that in order for this function to work properly, we will need to define the clock speed of the CPU.  This value will be assigned to the constant F\_CPU.  The clock speed of the ATmega328PB can be found in the "[*ATmega328PB Xplained Mini User Guide*](https://www.microchip.com/en-us/development-tool/ATMEGA328PB-XMINI#document-table)".  By searching for "clock" in the document, you'll find this page where it indicates that while configured to run at 5 volts, the clock speed is 16000000 Mhz.



Now we need to add this defined value to our code as an Unsigned Long constant as shown below on the first line.  The delay function has also been added to the main loop of the code as shown. It will cause a delay of 500ms.

If you compile and run this code, it will not toggle the LED.  It will turn on the LED and leave it on.

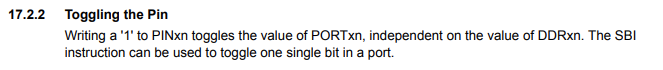
#define F\_CPU 16000000UL   
#include *<avr/io.h>*  
#include *<avr/delay.h>*  
  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
       PORTB |= (1 << PB5); *// set PB5 high*         
       \_delay\_ms(500);  
  
    }  
}

**Refer to Datasheet**

One more modification is needed.

Another look at the datasheet for the ATmega328PB shows in the I/O-Ports section, that the PORTB statement in the above code is not going to toggle our pin.  Our code will set the pin as HIGH and each time the code loops, it will keep bringing the pin HIGH so that it never goes LOW.

To toggle the pin, we need to review this information from the datasheet and write our code accordingly...



**Put the Code Together**

Putting it together in MPLAB X IDE.

Let's put these lines of code together in MPLAB X IDE. We'll comment out the PORTB line so that it doesn't affect our code. Note the addition of a PINB statement that does the toggling. Here's the final code version:

#define F\_CPU 16000000UL   
#include *<avr/io.h>*  
#include *<avr/delay.h>*  
  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
        *//PORTB |= (1<< PORTB5);*        PINB |= (1 << PINB5);   
         
       \_delay\_ms(500);  
    }  
}

**Program the Device**

Place this code into the main file of MPLAB X IDE and select the button indicated in the accompanying image to make and program the target device.

Program the Target

If everything was done as indicated, the LED on the board should blink.

Learn More

* [Finding Documentation and Turning On an LED](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)
* [Read an I/O as Input to Turn on LED](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-read-input/)

Sending "Hello World!" from the USART

Last modified by Microchip on 2023/11/09 09:02

**Contents**

* [In This Video](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#HInThisVideo)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#HLearnMore)

In This Video  
<https://youtu.be/XMTZHN23Ke8>

* Provide relevant community developed training materials for a useful USART\_putstring() function.
* Modify this to call our USART\_Transmit() function.
* Use delays for testing purposes.
* Put additional functionality into USART\_irq.c and the function prototype to USART\_irq.h, so main remains clean.

Procedure

Add Functionality to the USART Program

* Add the following function to our program just below the #include statements

void USART0\_Transmit(char\* StringPtr){  
   
**while**(\*StringPtr != 0x00){  *//Check if there is still more chars to send from the null char* USART\_send(\*StringPtr);    *//Send one char at a time* StringPtr++;}              *//Increment the pointer to read the next char*   
}

[Back to Top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#top)

Assign a Value to String[]

Add the following line to our main.c above the previous function

char String[]="Hello world!!";

[Back to Top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#top)

Call the function from main.c

* Add this line to the main.c file to send the String to the USART.  Place it just below the sie() line

 USART\_putstring(String);

[Back to Top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#top)

Compile and run the program.

* Open up your terminal program and if you are fast enough, you'll see the following...

​

[Back to Top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#top)

Make it easier to see in our Terminal program

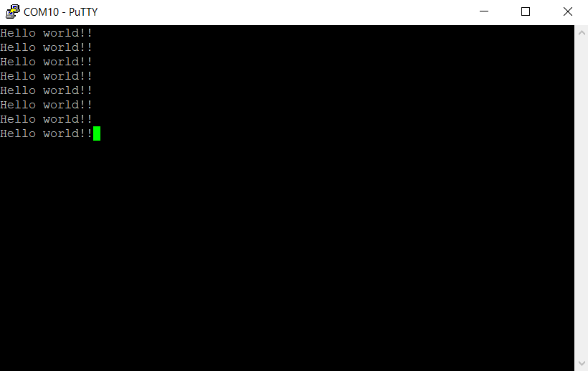
* Add a delay function and move the USART\_putstring(String) line into our main loop.  We will need to include the delay header file as well.  Here is the new main.c file contents with these changes.

#include *<avr/io.h>*  
#include *<avr/interrupt.h>  //ADDED*  
#include *"USART\_irq.h"*  
#include *<util/delay.h>*  
  
char String[]="Hello world!!";  
  
void USART\_putstring(char\* StringPtr){  
   
**while**(\*StringPtr != 0x00){  *//Check if there is still more chars to send from the null char* USART0\_Transmit(\*StringPtr);    *//Send one char at a time* StringPtr++;}              *//Increment the pointer to read the next char*   
}  
  
int main(void)  
{  
 */\* Set the baudrate to 9600 bps using 8MHz internal RC oscillator \*/*  
 USART0\_Init(MYUBRR);    *//CHANGED to PASS MYUBRR from calculation*  
 sei();  
 USART\_putstring(String);      
  
 **for**( ; ; ) {  
  */\* Echo the received character \*/*  
          
          
          
  USART0\_Transmit(USART0\_Receive());  
        USART\_putstring(String);  
        \_delay\_ms(1000);  
 }  
}

[Back to Top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#top)

Compile and run the program

* You should see the following in your terminal program.  Every time you click <Enter>, the terminal program will show what what is being sent from the UART.  Depending on how you set up the Terminal program, you might see the Hello world!! coming in every second as in the video.

​

[Back to Top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-hello-world/#top)

Clean up Main again

* Move our new function into the UART\_irq.c file
* Put the function definition into the USART\_irq.h file

Now retest the program to make sure the functionality is still there.

AVR®: Read an I/O as an Input to Turn on an LED

Last modified by Microchip on 2023/11/10 11:09

In this Lesson: <https://youtu.be/mXPyRF-_Y1E>  
[**https://youtu.be/mXPyRF-\_Y1E**](https://youtu.be/mXPyRF-_Y1E)

* Set up macros to make code more readable.
* Poll the status of the GPIO connected to the switch.
* Debug in MPLAB® X IDE using the I/O view, seeing bits in the PIN, DDRB, and PORTB registers change as we single step through the code.

Procedure

Macros

Define some macros to make our code more readable.

The previous lesson on flashing an LED gave us the following code...

#define F\_CPU 16000000UL   
#include *<avr/io.h>*  
#include *<avr/delay.h>*  
  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
        *//PORTB |= (1<< PORTB5);*        PINB |= (1 << PINB5);   
         
       \_delay\_ms(500);  
    }  
}

Let's add some macros to make it more readable.

In embedded C, the #define directive is used to define a macro, which is essentially a symbol or identifier that represents a certain value or expression.

The syntax for #define is as follows:

#define identifier value

where "identifier" is the name of the macro and "value" is the value or expression that the macro represents.

When the preprocessor encounters the #define directive, it replaces all occurrences of the identifier in the code with the corresponding value. This is done before the code is compiled, so it allows for simple substitution of values or expressions throughout the code.

For example, if you wanted to define a macro to represent the value of pi, you could use the following #define directive:

#define PI 3.14159

Then, anywhere in your code where you use the identifier PI, it will be replaced with the value 3.14159. This can be useful for simplifying code, making it more readable, and making it easier to make global changes to values or expressions throughout the codebase.  Note that we did this previously with the F\_CPU value.

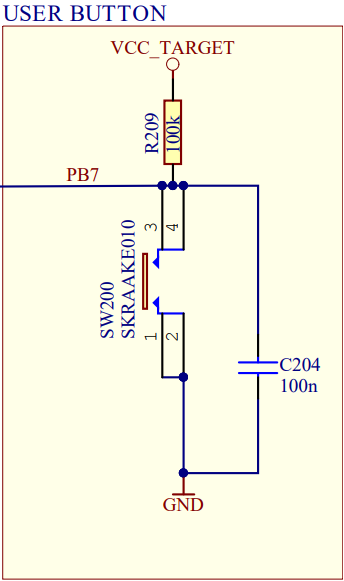
So, our LED flashing code with the #defines added, looks like below, and note that we added an LED off state that we'll need in the next iteration of code where we poll the status of a GPIO pin connected to a switch.

#define F\_CPU 16000000UL   
#include *<avr/io.h>*  
#include *<avr/delay.h>*  
  
#define LED\_ON  PORTB |= (1<<PORTB5)  
#define LED\_OFF PORTB &= ~(1<<PORTB5)  
#define LED\_TOGGLE  PINB |= (1<<PINB5)  
  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
        *//PORTB |= (1<< PORTB5);*        PINB |= (1 << PINB5);   
         
       \_delay\_ms(500);  
    }  
}

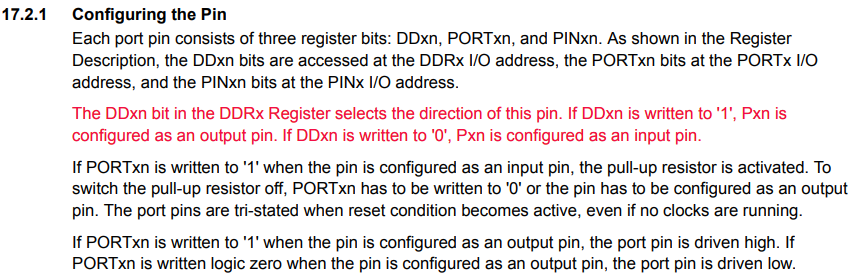
Switch Status

Poll the status of a GPIO-connected switch.

To find out which pin the switch is connected to, we will again review the schematic from the ATmega328PB Xplained Mini User Guide.  As shown below, the switch is connected to Pin PB7.  When the switch is pressed, the pin will read LOW because it will be directly connected to ground.  When not pressed, the pin will read HIGH because it will be connected to VCC\_TARGET through the 100k resistor.



To configure pin PB7 as an input, we will take a look at the datasheet.  The information we need is shown in red...



Translating this into code looks like...

DDRB &= ~(1<<DDB7); *//set PB7 as an input pin*

Now we need some control logic in our code so that when the button is pressed our code will turn on the LED and when not pressed, the LED will turn off.

**if**(!(PINB & (1<<PINB7))) *//If PINB7 is low*        {      
            LED\_ON;  
        }  
        **else**  
        {  
            LED\_OFF;  
        }

Combine the Code

Putting it all together and programming the device.

Here's the final version of the code for this lesson.  Enter it into the main.c file.  Note that the delay references have been removed because this example does not use the delay function.

#include *<avr/io.h>*  
  
#define LED\_ON  PORTB |= (1<<PORTB5)  
#define LED\_OFF PORTB &= ~(1<<PORTB5)  
#define LED\_TOGGLE  PINB |= (1<<PINB5)  
  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*    DDRB &= ~(1<<DDB7); *//set PB7 as an input pin*      
    **while** (1) {  
          
        **if**(!(PINB & (1<<PINB7))) *//If PINB7 is low*        {      
            LED\_ON;  
        }  
        **else**  
        {  
            LED\_OFF;  
        }        
         
    }  
}

Select the button indicated below to make and program the target device.

Program the Device

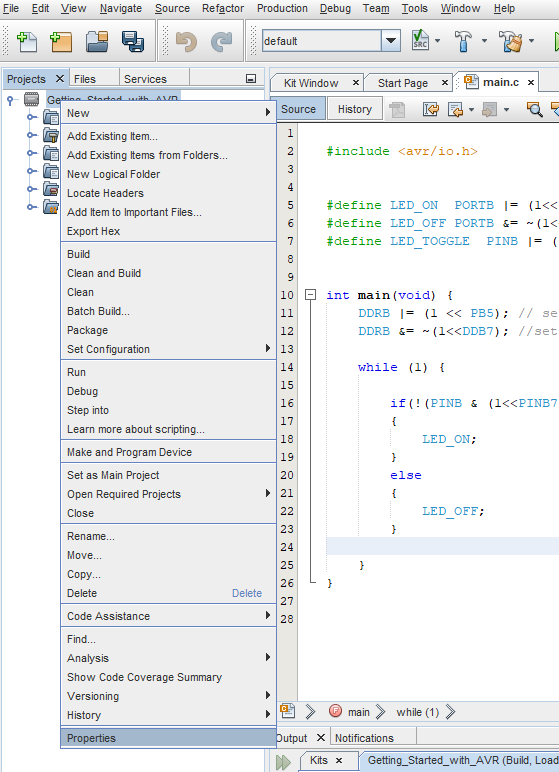
Your application code is working correctly if you press the button, the LED lights up.  When you release the button, the LED is off.

Debugging

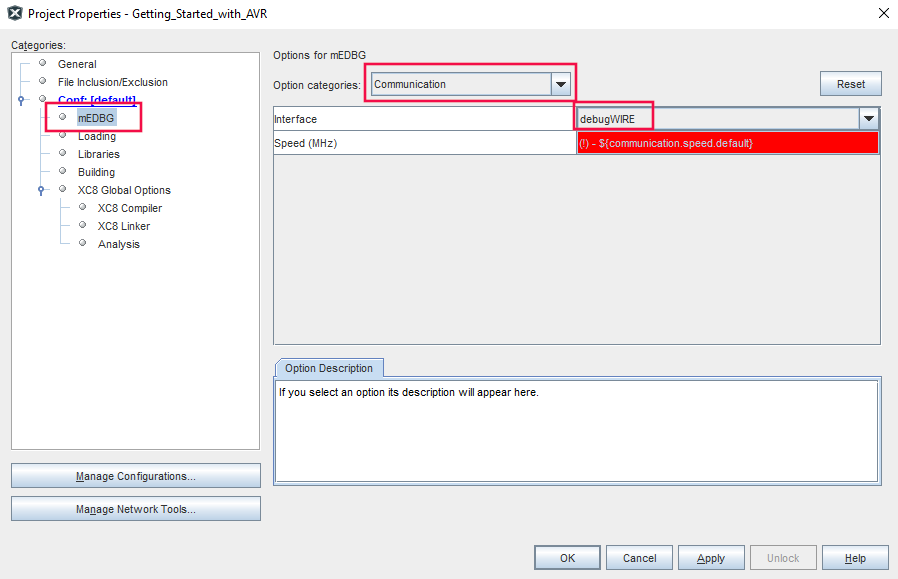
Use MPLAB X IDE debugging features to get another look at what is happening in our code.

To do this we will need to change the project properties to debug with debugWIRE.

Right-click on the project and select Properties at the bottom of the screen.



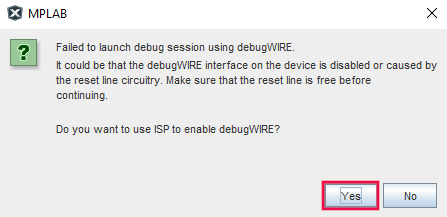
When the dialog box opens, select **mEDGB** on the left and then **Communications** in the Option categories.  Finally, if not already selected, choose **debugWIRE** as the interface.  Hit **Enter**



Now select the Debug Project button on the top menu.

Debug Button

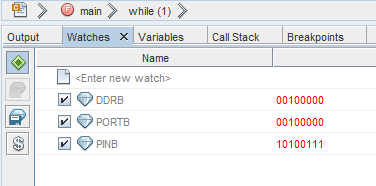
Select YES when the dialog box appears.



The application should now be running in Debug mode.  Press the **Pause** button to stop it temporarily.

Pause Debug Session

Finally, go to the Watch menu toward the bottom of your screen and create new watches for **DDRB**, **PINB**, and **PORTB**.  From here you can step through the program using **F7** and observe the values in the watch window change.  Try pressing the button while stepping as well so you can see that both branches of the control logic affect the registers.



 To take the MPLAB X IDE out of debug mode, select the Stop button.



Previous Lesson:

[Flashing an LED at a Specific Frequency](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/)

Next Lesson:

[Using Pin Change Interrupts](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-pin-change-interrupts/)

picoPower® Technology and Putting an Application to Sleep

Last modified by Microchip on 2023/11/09 09:02

AVR® Microcontroller (MCU): picoPower® Technology and Putting an Application to Sleep

Overview

In this video: <https://youtu.be/HaamYW3I72E>

* Revisit AVR® Libc support for sleep.
* Disable the Brown-out detector when sleeping.
* Provide an overview of the solution project to AVR hands-on megaAVR® device picoPower technology, which uses a very similar application to the one we have developed.
* Explore the techniques used to save power:
  1. General Purpose Input/Output (GPIO) set as inputs with pull-ups enabled.
  2. Use the Power Reduction register to turn off the clocks to unused peripherals.
  3. Use the Asynchronous Timer-Counter 2 which can be clocked directly from a Real Time Clock (RTC).
* Set the Sleep mode to Power-Save mode.
* In the while loop, if the send flag is set after sending the Universal Synchronous Asynchronous Receiver Transmitter (USART) string, it puts the AVR MCU to sleep.

Learn More

* [Understanding USART TX Pin Activity Using the Data Visualizer](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-tx-data-visualizer/)
* [Introduction to PTC on AVR](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/intro-ptc-on-avr/)

# Making Our Simple App Low Power

Last modified by Microchip on 2023/11/09 09:02

## In this Lesson

**Contents** <https://youtu.be/EvGHw6jSlLM>

* [In this Lesson](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/low-power/#HInthisLesson)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/low-power/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/low-power/#HLearnMore)
* Using sleep modes to reduce power consumption.
* Active clock domains and wake-up sources in the different sleep modes.
* Using the Power Reduction Register to turn off the clocks to unused peripherals.
* AVR® Libc support for sleep modes.
* Choosing and setting a sleep mode.
* Enabling sleep, putting the CPU to sleep, and disabling sleep on waking up.
* A recommended application note, AVR4013: picoPower Basics.

In the last lesson, we used a pin change interrupt to determine whether a switch was pressed or not.  The main benefit of using interrupts instead of polling a pin is that you are not keeping the microprocessor busy with one particular task and it gives us the opportunity to put the device into a low-power Sleep mode and dramatically reduce the power consumption of an application.  This benefit is particularly important for battery-driven applications.

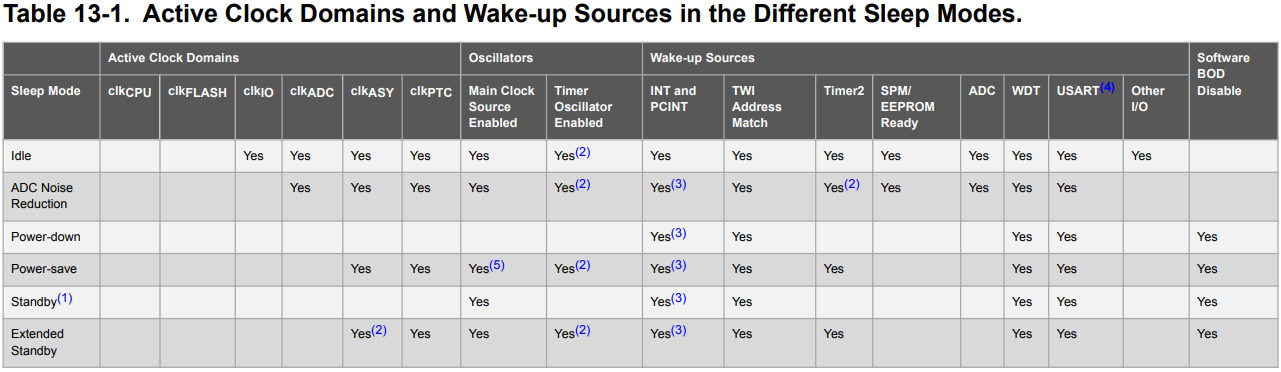
[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/low-power/)

## Procedure

##### ****1. Review the Datasheet****

###### **Review the datasheet for the  ATmega328PBto determine which sleep modes are available.**

Search for Sleep Modes in the datasheet and the section titled PM - Power Management and Sleep Modes will show the accompanying table. It shows what Sleep Modes are available to us based on which wake up sources we will be using.  In our case, we are using PCINT so all of the Sleep Modes are available to us.

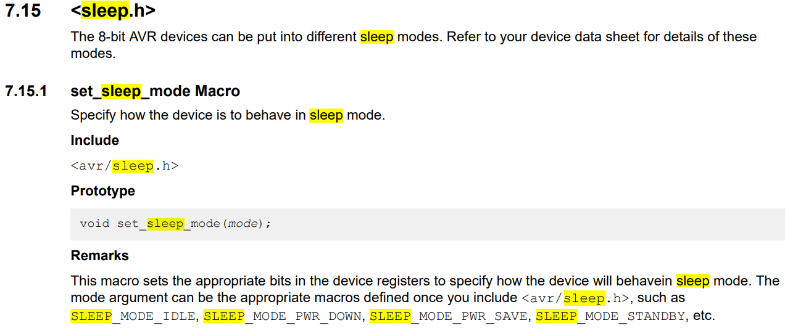


Of these Sleep Modes, the Power-down mode has the lowest current consumption. In this mode, the CPU and all peripherals are powered off, except for the asynchronous timer and external interrupts.  In Power-down mode, the current consumption is typically around 100 nA at 1.8V and room temperature, making it ideal for battery-powered applications that require long battery life. We'll focus on implementing this Sleep Mode in our application.

##### ****2. Review the Compiler Guide****

###### **Review the MPLAB XC8 C Compiler User Guide for AVR for how to implement Power-down Sleep Mode**

Open the [MPLAB XC8 C Compiler User Guide for AVR MCU](https://www.microchip.com/en-us/search?searchQuery=avr%20compiler%20guide&category=Product%20Documents|User%20Guides&fq=start%3D0%26rows%3D10) and search for "sleep".  You should land on the section that describes the Sleep mode mmacro.



##### ****3. Implement the Code Changes****

###### **Make the changes needed in the code to implement Power Down sleep mode.**

From Step 2 we see that we need to include the header file avr/sleep.h into our code which looks like this:

#include *<avr/sleep.h>*

Then we need to implement this line above our while() loop inside our main()

set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN);

Finally, we put the following code inside our while() loop so that we are always in Power-Down mode unless the interrupt occurs which wakes up the micro to service the ISR.

sleep\_mode();

Now let's put it all together

#include *<avr/io.h>*  
#include *<avr/interrupt.h>*  
#include *<avr/sleep.h>*  
  
#define LED\_ON  PORTB |= (1<<PORTB5)  
#define LED\_OFF PORTB &= ~(1<<PORTB5)  
#define LED\_TOGGLE  PINB |= (1<<PINB5)  
#define SWITCH\_PRESSED !(PINB & (1<<PINB7))  
  
  
ISR(PCINT0\_vect){  
    **if**(SWITCH\_PRESSED) *//If PINB7 is low*        {      
            LED\_ON;  
        }  
        **else**  
        {  
            LED\_OFF;  
        }  
      
}  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*    DDRB &= ~(1<<DDB7); *//set PB7 as an input pin*      
    set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN);  
      
    PCMSK0 |= (1<<PCINT7);  
    PCICR |= (1<<PCIE0);  
      
    sei();  
      
    **while** (1) {  
          
        sleep\_mode();  
  
           
    }  
}

##### ****4. Program the Device****

* + **Select the Program button from the top menu**

Program the Device

The application should display the same behavior when the button is pressed.  What you do not see, without some method to measure it, is that when the button is not pressed, the micro is drawing significantly less current.

For more information and details about low power, view the [app note AVR4013](https://www.microchip.com/en-us/application-notes/an8349).

## Learn More

* [Using Pin Change Interrupts](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-pin-change-interrupts/)
* [Using Timer Overflow IRQs](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-timer-overflow-irqs/)

# AVR®: Finding Documentation and Turning On an LED

Last modified by Microchip on 2023/11/09 09:02 <https://youtu.be/6JEbnyz2Mac>

**Contents**

* [In This Video](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HInThisVideo)
* [Reference Materials](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HReferenceMaterials)
* [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HProcedure)
* [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/#HLearnMore)

## In This Video

* Find the device datasheet, "Xplained Mini User's Guide", and its schematics.
* Start a new project in MPLAB® X IDE.
* Demonstrate how to efficiently use the datasheet to understand how to configure a pin and turn on an LED.

## Reference Materials

* [ATmega328PB Xplained Mini](https://www.microchipdirect.com/productsearch.aspx?keywords=ATMEGA328PB-XMINI)
* [MPLAB® X IDE](https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide)

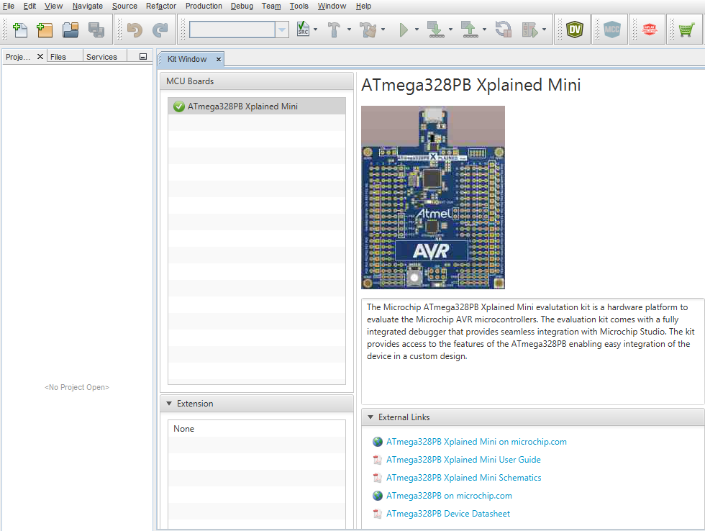
[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

## Procedure

##### Start MPLAB X

Start MPLAB X and plug in the Xplainined Mini to one of your USB ports.

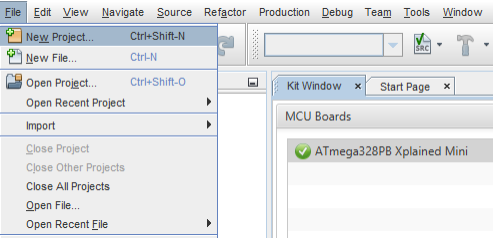
If the Xplained Mini is enumerated properly, you should see the following screen with the Kit Window opened showing the ATmega328PB Xplained Mini board.



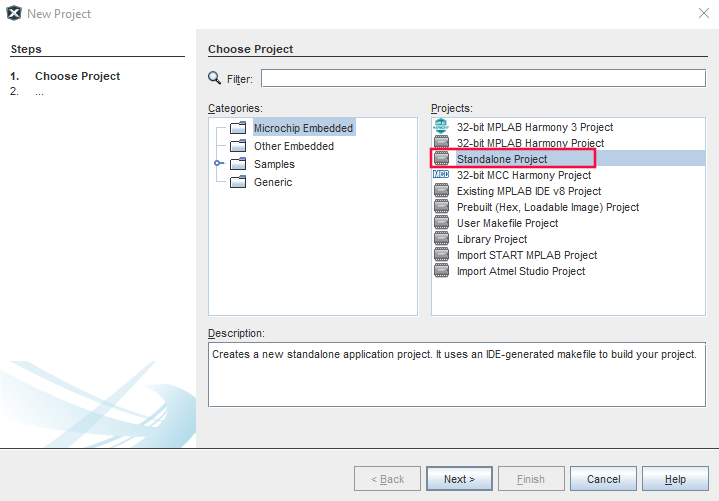
[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Create the Project

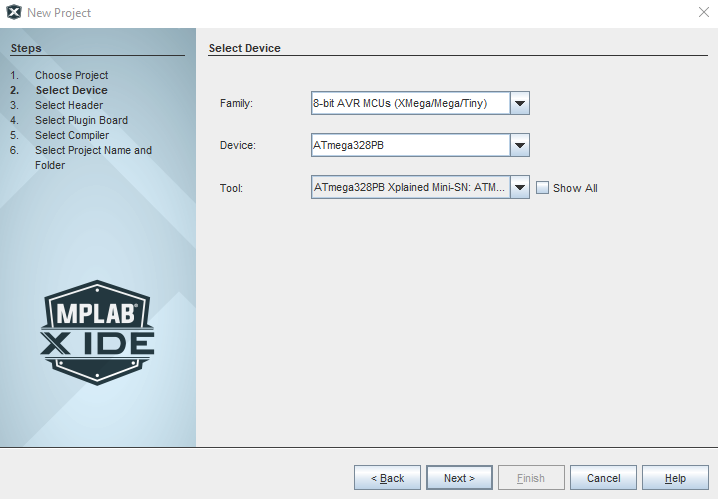
Select **File > New Project** from the menu at the top.



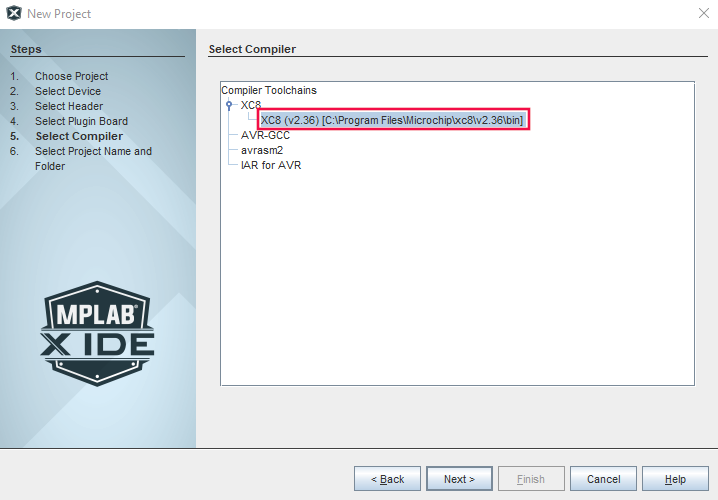
A New Project dialog box will appear.  Select **Standalone Project** from the options and then select **Next**.



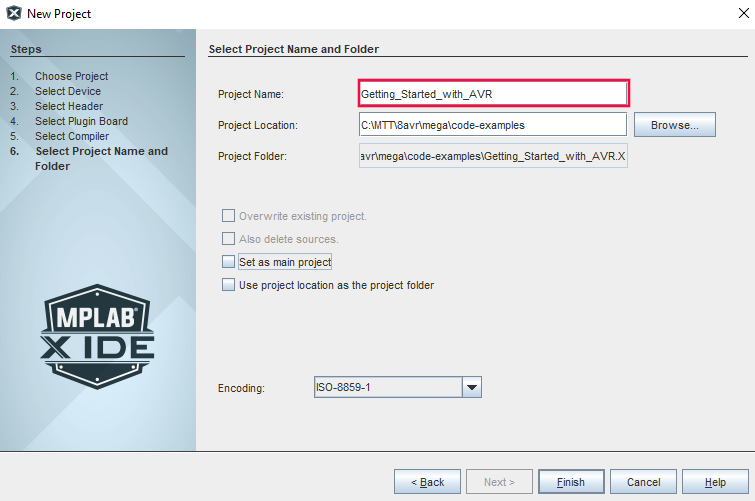
At the **Select Device** screen, select the following for the device **Family**, **Device**, and **Tool**.  Select **Next**.



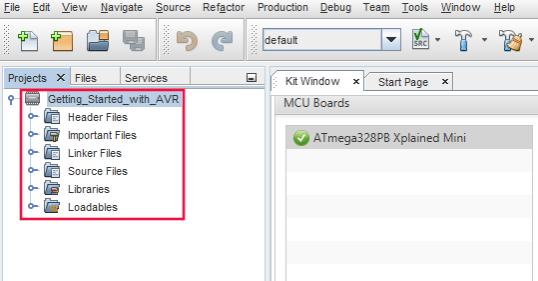
Now select the **XC8 Compiler Toolchain** and select**Next**



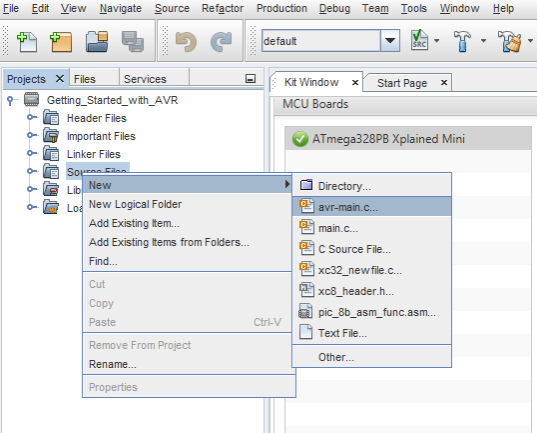
Next, we'll create a name for our project.  Enter "Getting\_Started\_with\_AVR" in the **Project Name** dialog box and then select **Finish**

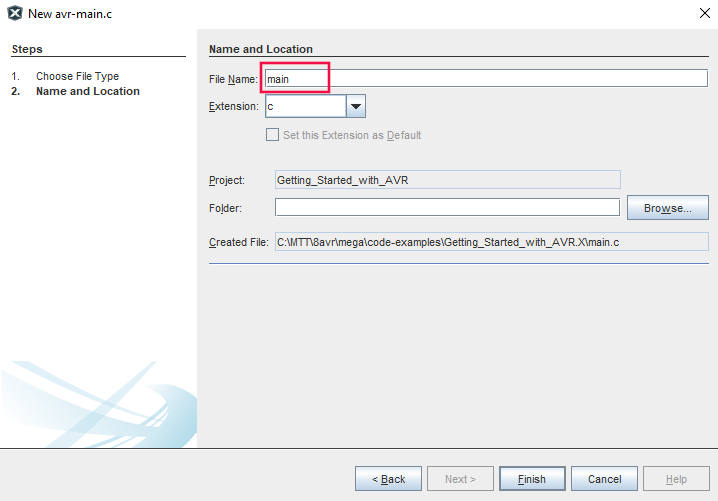


MPLAB X IDE has created our project and populated the user interface with several folders.

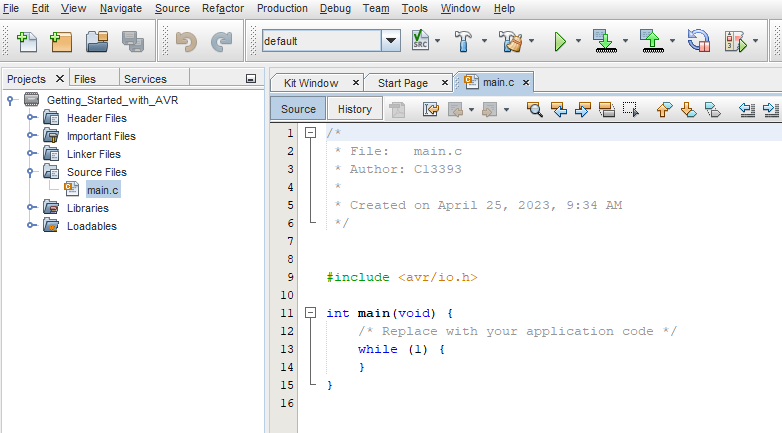


Right-click on the **Source Files** folder and hover over **New** and then select **avr-main.c**



Change the name of the File Name at the top of the dialog box to main and then select **Finish.**

MPLAB X has now created a main document as part of our project and it should be open for you to edit.  This file is where we'll enter our code to turn on the led of the ATmega328PB Xplained Mini.  As you can see, it populated the file with some of the things we will need in our project.  The avr/io.h header file has been included which contains definitions we'll need along with the main while loop for our application code to reside in.

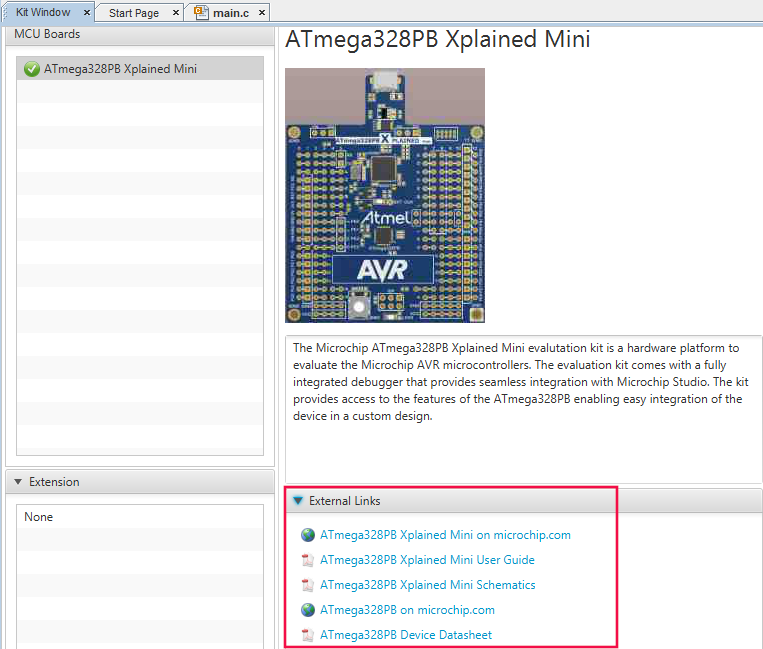


[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Find the documents

###### **the documents we need to get started programming the board**

In the kit window that opened when the ATmega328PB was connected, there is a section indicated by External Links located below the board image and description.  If not already expanded, select that section to expose the links to the ATMega328PB Mini User Guide, schematics, and datasheet.

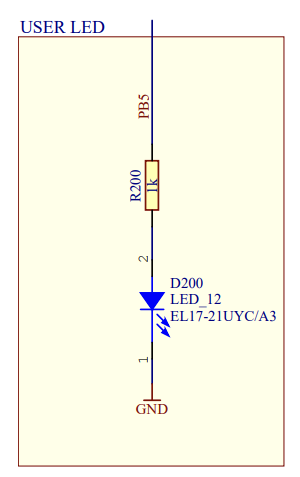


[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### LED PIN

* **Find what pin the LED is connected to so that we can turn on the correct pin.**

The ATmega328PB Xplained Mini User manual as well as the ATmega328PB Xplained Mini Schematic shows that the LED is connected to pin PB5 through a 1k ohm resistor. In order to turn the LED on, we will need to make PB5 high.



In order to determine how to do that, we will consult the ATmega328PB datasheet.  PB5 is a GPIO pin (General Purpose Input Output) and the best place to start in the datasheet is section 17 which discusses I/O-Ports and how to configure them.

From the datasheet, each port pin consists of three register bits:  DDxn, PORTxn, and PINxn.  The DDxn bit in the DDRx register selects the direction of this pin.

Note:  A lowercase "x" represents the numbering letter for the port, and the lowercase "n" represents the bit number.  For example, PORTB5 is bit number 5 on Port B.  In C Programming we can shorten this to PB5.

[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Set Pin Direction

First we set the direction of the pin and in this case, we want it to be an output. In C Programming, we can accomplish this with the following line of code:

DDRB |= (1 << PB5); *// set PB5 as output pin*

###### **Breakdown**

* DDRB is a register in the ATmega328PB microcontroller that controls the data direction of the digital I/O pins on port B. Each bit in the register corresponds to a pin on the port, with bit 0 corresponding to pin PB0, bit 1 to pin PB1, and so on.
* |= is the bitwise OR assignment operator. It performs a logical OR operation between the current value of PORTB and the value on the right-hand side of the operator and then stores the result back into PORTB.
* (1 << PB5) is a bit-shift operation that moves the value 1 to the left by the number of bits specified by the PB5 constant, which is 5. This evaluates to 0b00100000.

When we perform the bitwise OR operation between DDRB and (1 << PB5), the result will be that the bit corresponding to pin PB5 in DDRB is set to 1, while the other bits remain unchanged. This sets the data direction of pin PB5 to be an output.

Now we want to set the pin high so that it can drive the Led.

PORTB |= (1 << PB5); *// set PB5 high*

###### **Breakdown**

* PORTB is a register in the ATmega328PB microcontroller that controls the state of the digital output pins on port B. Each bit in the register corresponds to a pin on the port, with bit 0 corresponding to pin PB0, bit 1 to pin PB1, and so on.
* |= is the bitwise OR assignment operator. It performs a logical OR operation between the current value of PORTB and the value on the right-hand side of the operator, and then stores the result back into PORTB.
* (1 << PB5) is a bit-shift operation that moves the value 1 to the left by the number of bits specified by the PB5 constant. In this case, PB5 is defined as the value 5, so (1 << PB5) evaluates to 0b00100000.

This line of code sets the bit corresponding to pin PB5 in the PORTB register to 1 while leaving the other bits unchanged. This effectively sets the output voltage of pin PB5 to the logic HIGH level.

[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

##### Combine the Code

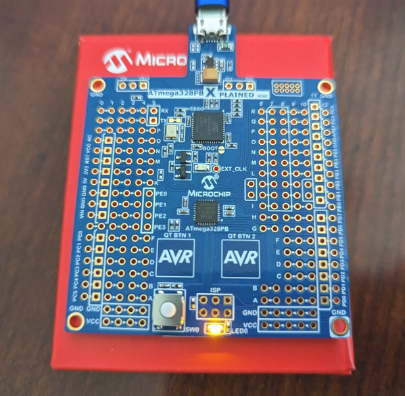
* **Let's put these lines of code together in MPLAB X.**

#include *<avr/io.h>*  
  
int main(void) {  
    DDRB |= (1 << PB5); *// set PB5 as output pin*      
    **while** (1) {  
          
       PORTB |= (1 << PB5); *// set PB5 high*      
    }  
}

Place the two lines of code into the main file and select the button indicated below to make and program the target device.

Program the Target

If everything was done as indicated, the led on the board should light continuously.



[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/led-on/)

## Learn More

* [Introduction and Key Training Application](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/introduction/)
* [Flashing an LED at a Specific Frequency](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/flashing-led/)

Better Coding Practice for USART Send Using a Sendflag

Last modified by Microchip on 2023/11/09 09:02

<https://youtu.be/FHS5rvHLFIc>

AVR®: Better Coding Practice for USART Send Using a Sendflag

In This Video:

* Debug the IRQ-driven Universal Synchronous Asynchronous Receiver Transmitter (USART) communications to understand what happens when a string is sent.
* Demonstrate that the USART Data Rate Empty IRQ has hit a number.
* Create a global flag that is checked in the while loop to determine when the average should be sent over the USART, enabling us to move the USART send function out of the Analog-to-Digital Converter (ADC) Interrupt Service Routine (ISR).
* Reveal that we still have the same functionality but have implemented better coding practices.

Learn More

* [Sending Averaged ADC Sample Over USART](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-avg-adc-over-usart/)
* [Understanding USART TX Pin Activity Using the Data Visualizer](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-usart-tx-data-visualizer/)

AVR® 10 ms Analog-to-Digital Converter (ADC) Samples Averaged Over 1 Second

Last modified by Microchip on 2023/11/09 09:02

**Contents**

* + [In This Video](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/adc-samples/#HInThisVideo)
  + [Procedure](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/adc-samples/#HProcedure)
  + [Learn More](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/adc-samples/#HLearnMore)

In This Video  
<https://www.youtube.com/watch?v=s6xND68cUIA>

* Modify a millisecond timer function, from [Updating Pulse Width Modulator (PWM) duty cycle using a millisecond timer](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-updating-pwm-millisecond-timer/), we create a 10 mS timer.
* Test the frequency, using the Data Visualizer, by toggling a pin in the timer IRQ.
* Change Analog-to-Digital Converter (ADC) auto-trigger source to Timer Counter 0 Compare Match A.
* Create volatile global variables accumulator, average and samples, using these to calculate an average in the ADC IRQ.
* Verify that we are now averaging ADC light sensor readings every second.

[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/adc-samples/)

Procedure

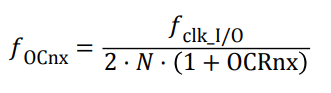
Pull up the Code from a Previous Project

Once again, we will pull some code from our [previous PWM project](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/8-bit-avr-adc-to-pwm-duty-cycle/) and add it to the AVR\_ADC project to create the 10mS timer.  Copy the milliS\_timer functiond() and paste it to just below the PWM\_Init function() in the AVR\_ADC project in the previous lesson.

Here is the code to be copied over...

void milliS\_timer(uint8\_t milliS)  
{  
    TCCR0A |= (1 << WGM01);                *//set to CTC mode*    TCCR0B |= (1 << CS02) | (1 << CS00);   *//set prescaler to 1024*  
    OCR0A = milliS \* 7.8125 - 1;  
    TIMSK0 |= (1 << OCIE0A);               *//enable the interrupt*}

In order to ADC samples every 10 ms, we need to revisit this equation...

​​​

OCR0A = (Required\_Time \* F\_CPU) / (Prescaler\_Value \* 1000) - 1

In this case, the required time is 10ms, the prescaler value is 1024, and the F\_CPU is 16,000,000 Hz.

Plugging these values into the formula, we get:

OCR0A = (10ms \* 16,000,000 Hz) / (1024 \* 1000) - 1 = (10 \* 16,000) - 1 = 160,000 - 1 = 159,999

Since OCR0A is an 8-bit register, it can hold values from 0 to 255. Therefore, we need to choose the closest possible value to 159,999 that fits within the 8-bit range. In this case, the closest value is 156, which can be used as the OCR0A value to achieve a 10ms interrupt with a prescaler of 1024.

void milliS\_timer0\_10()  
{  
    TCCR0A |= (1 << WGM01);    *//set to CTC mode*    TCCR0B |= (1 << CS02) | (1 << CS00);    *//set prescaler to 1024*      
  
    OCR0A = 156; *// 10mS at 1024*    TIMSK0 |= (1 << OCIE0A);               *//enable the interrupt*}

Test our Sample Frequency

We will test our sample frequency by setting a pin high each time the code steps into the TIMER0 ISR.  Place the following code in the main and newly created ISR for TIMER0.

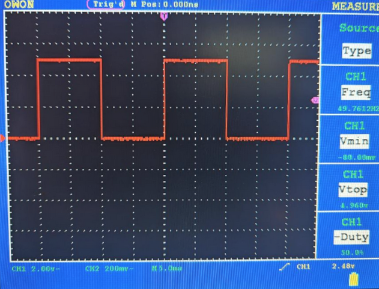
Place in main

DDRD |= (1 << DDRD2);

Place the ISR below the ISR for the ADC

ISR(TIMER0\_COMPA\_vect)  
{  
    PIND |= (1 << PIND2);  
}

Every time the TIMER0 interrupt occurs, PD2 will change states. We can see this in the scope plot below.

​​​

Change ADC Trigger Source

Now that we know our TIMER0 is counting down properly, we will set the TIMER0 interrupt as a Trigger Source for the ADC.  This will change the ADC Trigger Source from Free Running Mode to Timer/Counter0 Compare Match A.

To make this happen, add the following line to the ADC\_init() function just above the sei(); line.

ADCSRB |= (1 <<ADTS1) | (1 << ADTS0);

Now every time TIMER0 counts down, it generates and interrupt, and an ADC sample is taken.

Modify the ADC Interrupt Service Routine

Create the following variables just below the #define macros. These will be used to calculate the averages.

**volatile** uint32\_t accumulator = 0;  
**volatile** uint16\_t average = 0;  
**volatile** uint16\_t samples = 0

Here is the modified ISR(ADC\_vect) which will gather the data and perform calculations for the average.

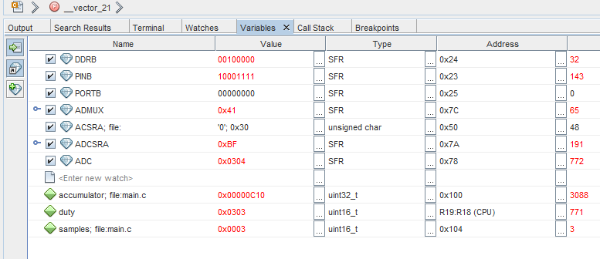
ISR(ADC\_vect)  
{  
    uint16\_t duty = ADC;  
    OCR1B = duty;  
      
    accumulator += duty;  
    samples++;  
    **if** (samples == 100)  
    {  
        average = accumulator/100;  
        accumulator = 0;  
        samples = 0;  
    }  
}

Complete Code

Here is the complete code for the lesson.

#include *<avr/io.h>*  
#include *<avr/interrupt.h>*  
#define F\_CPU 16000000UL  
  
#define LED\_ON  PORTB |= (1<<PORTB5)  
#define LED\_OFF PORTB &= ~(1<<PORTB5)  
#define LED\_TOGGLE  PINB |= (1<<PINB5)  
  
**volatile** uint32\_t accumulator = 0;  
**volatile** uint16\_t average = 0;  
**volatile** uint16\_t samples = 0;  
  
ISR(TIMER1\_COMPA\_vect)  
{  
   LED\_ON;   
}  
  
ISR(TIMER1\_COMPB\_vect)  
{  
   LED\_OFF;   
}  
  
ISR(ADC\_vect)  
{  
    uint16\_t duty = ADC;  
    OCR1B = duty;  
      
    accumulator += duty;  
    samples++;  
    **if** (samples == 100)  
    {  
        average = accumulator/100;  
        accumulator = 0;  
        samples = 0;  
    }  
}  
  
ISR(TIMER0\_COMPA\_vect)  
{  
    PIND |= (1 << PIND2);  
}  
  
void PWM\_Init(void)  
{  
    TCCR1B |= (1 << CS10) | (1 << WGM12);  *//No Prescaler & set mode to CTC*    TIMSK1 |= (1 << OCIE1A) | (1 << OCIE1B); *//Enable the CTC interrupt*    OCR1A = 800;  
    OCR1B = 400;  
}  
  
void milliS\_timer0\_10()  
{  
    TCCR0A |= (1 << WGM01);    *//set to CTC mode*    TCCR0B |= (1 << CS02) | (1 << CS00);    *//set prescaler to 1024*      
  
    OCR0A = 156; *// 10mS at 1024*    TIMSK0 |= (1 << OCIE0A);               *//enable the interrupt*}  
  
void ADC\_init(void)  
{  
   ADMUX |= (1 << REFS0) | (1 << MUX0);   *//AVCC equal to VCC and ADC1 input*   ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);  
   ADCSRB |= (1 <<ADTS1) | (1 << ADTS0);  
   sei();  
}  
  
  
int main(void) {  
    DDRD |= (1 << DDRD2);  
    DDRB |= (1 << DDRB5);  
    ADC\_init();  
    PWM\_Init();  
    milliS\_timer0\_10();  
      
    **while** (1) {  
    }  
}

Set a breakpoint within the ADC interrupt service routine, enter debug mode, and watch the accumulator and average values.  You should these update every time the application hits the breakpoint and is restarted.

​​​

[Back to top](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8-bit-avr/getting-started/adc-samples/)