



www.ccsinfo.com

262-522-6500

<BITS & BYTES>

Newsletter

INSIDE THIS ISSUE:

Pg 2-5
ICD Programming Information
and Diagnostics

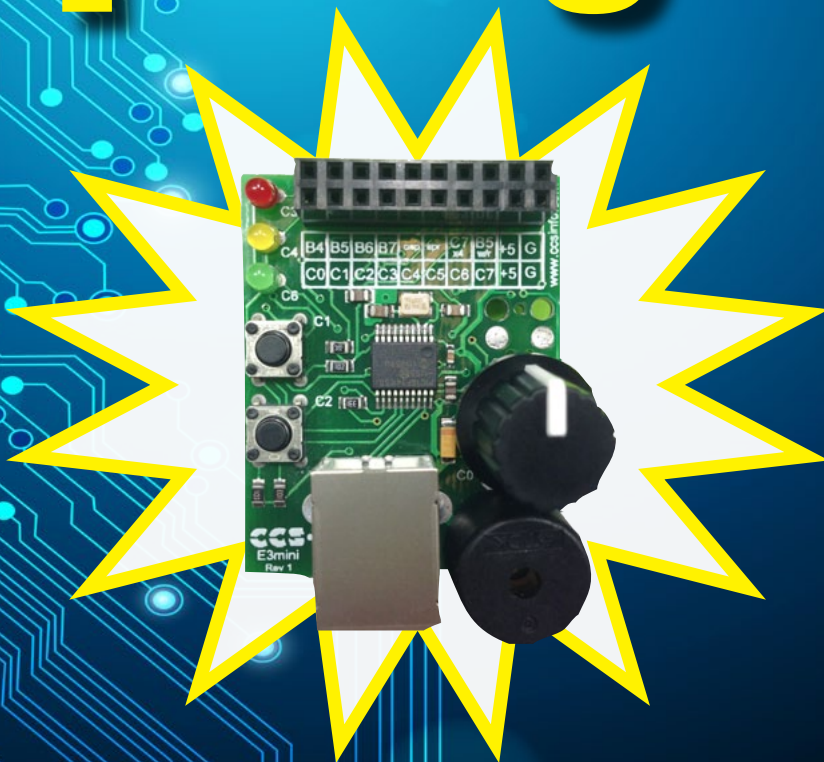
Pg 6-7
Using SPI with the CCS
Compiler

Pg 7
Programmer/Debugger Drivers
for Windows on Arm®
Processors

Pg 8
VS Code Extension for CCS C
Compiler

Pg 9
Programming RGB LEDs

Product Spotlight



E3-MINI PROTOTYPE BOARD

This board uses the PIC18F14K50 processor. Because the board has a bootloader, no device programmer is required to reprogram the board with new software. It also has a USB port that can be used to communicate between a PC and the user program running on the PIC® MCU.

This is a guide to successful In-Circuit Serial Programming (ICSP). To get started, the power connections, the supply voltage, and the bypass capacitors must be suitable for the device that is being programmed. If the ICD will be used for debugging, then the target device must also have an active clock oscillator.

ICSP Cable

Excessive cable length can cause various problems such as: noise pickup, crosstalk, signal reflections, or excessive voltage drop. Be sure to use a cable that is as short as practical for the application. Keep in mind that the circuit board's trace length will add to the overall wiring length. Therefore, place the ICSP connector close to the PIC on the circuit board.

ICSPCLK/PGC (clock) and ICSPDAT/PGD (data):

These are bidirectional data lines. The circuit must be designed in a way that does not significantly alter the voltage levels, rise time, or fall time of the signals. Avoid diodes, pull-up resistors, pull-down resistors, and large value capacitors. Please see Figure 1 for an example of normal PGC (top) and PCD (bottom) signals. It is best to dedicate these pins to ICSP. If the ICD will not be used for debugging, then additional circuitry can be connected to these pins. This allows the pins to be used for other functions after programming has completed, but the circuitry must present a high impedance during programming. If the additional circuitry does not present a sufficiently high impedance, a resistor can typically be added between the PIC® MCU pin and the additional circuitry. This can minimize the undesired loading affects of the additional circuitry and allow successful programming. If a resistor is not suitable, then the circuit can include a switch or a jumper instead. After programming has been completed, the switch or jumper would be used to connect the additional circuitry to the PIC® MCU.

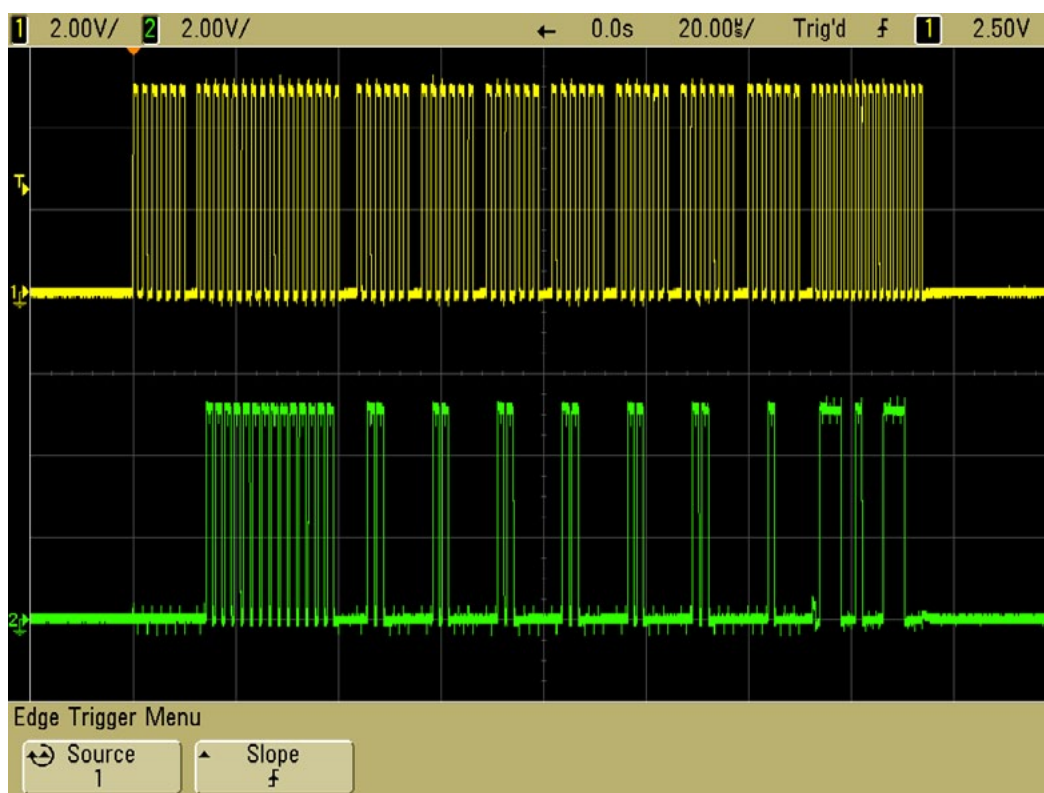


Figure 1

In general, we do not advise the use of capacitors on PGC or PGD. However, there is an exception for electrically-noisy environments and/or unusually long ICSP cables. In these situations, it can be helpful to include a 68pF capacitor from PGC to ground and a second 68pF capacitor from PGD to ground. This can help with noise suppression.

To help with design flexibility, some PIC® MCU devices have more than one pair of PGC/PGD pins. Any pair can be used for programming. There are configuration bits to select the pair that is used for debugging.

/MCLR (Master Clear, active lo)

The circuit must be designed in a way that does not significantly alter the voltage levels, rise time, or fall time of the signal during programming or debugging. In many situations, only a pull-up resistor is required here. CCS recommends a 47K pull-up resistor. Please see Figure 2 for an example of a normal /MCLR signal.

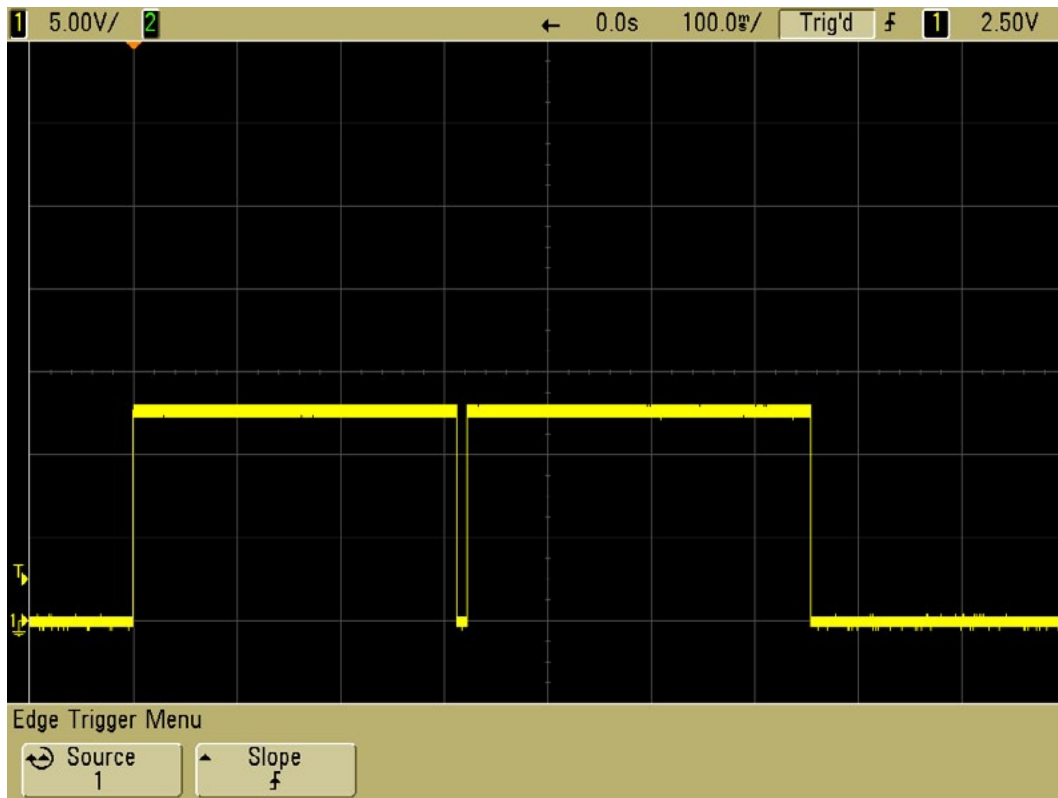


Figure 2

Many PIC® MCU devices include a Power-up Timer (PUT) which can be enabled in the device's configuration bits. For most applications, the PUT eliminates the need for a capacitor on the /MCLR pin. If a capacitor is connected between the pull-up resistor and ground, include a resistor or a diode between the capacitor and the /MCLR pin on the PIC® MCU. This will minimize the affect of the capacitor during programming / debugging. Microchip literature recommends a Schottky diode or a 470 Ohm resistor. If a Schottky diode is used, the cathode would connect to the /MCLR pin on the PIC® MCU. The /MCLR pin on the PIC® MCU will always connect directly to the /MCLR pin on the circuit board's ICSP connector when an ICD-U80 (or other CCS programmer) is used.

For some PIC® MCU devices, up to +13V is supplied to the /MCLR pin during programming. The circuit must be designed in a way that is tolerant of that voltage. The 47K pull-up resistor is a good way to deal with that voltage.

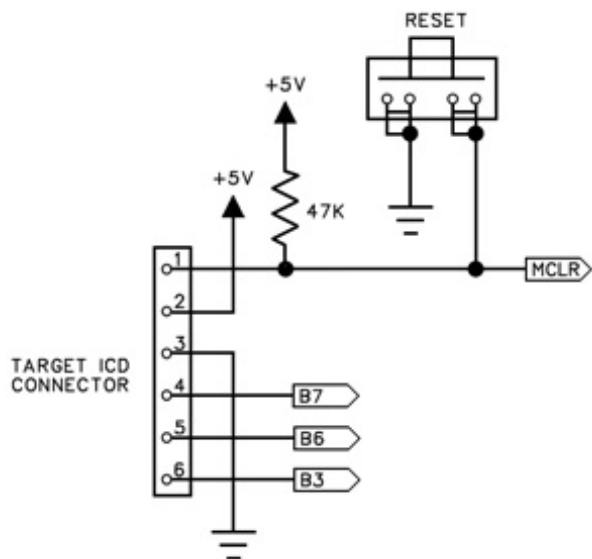


Figure 3

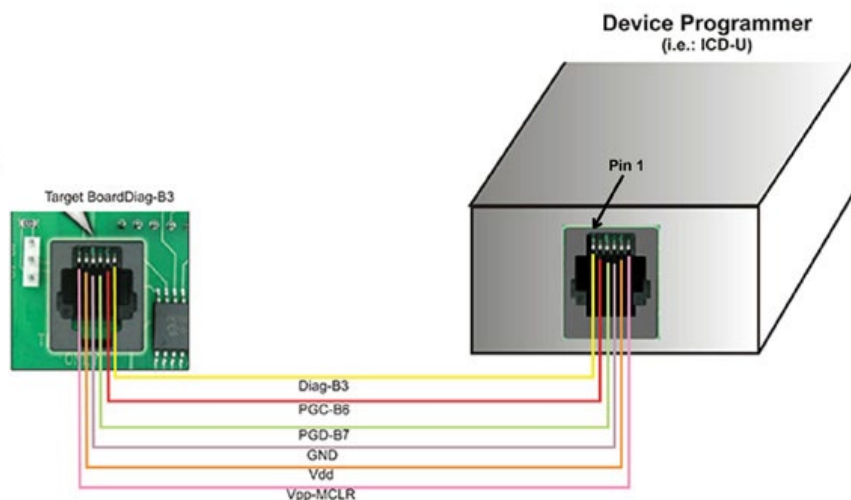


Figure 4

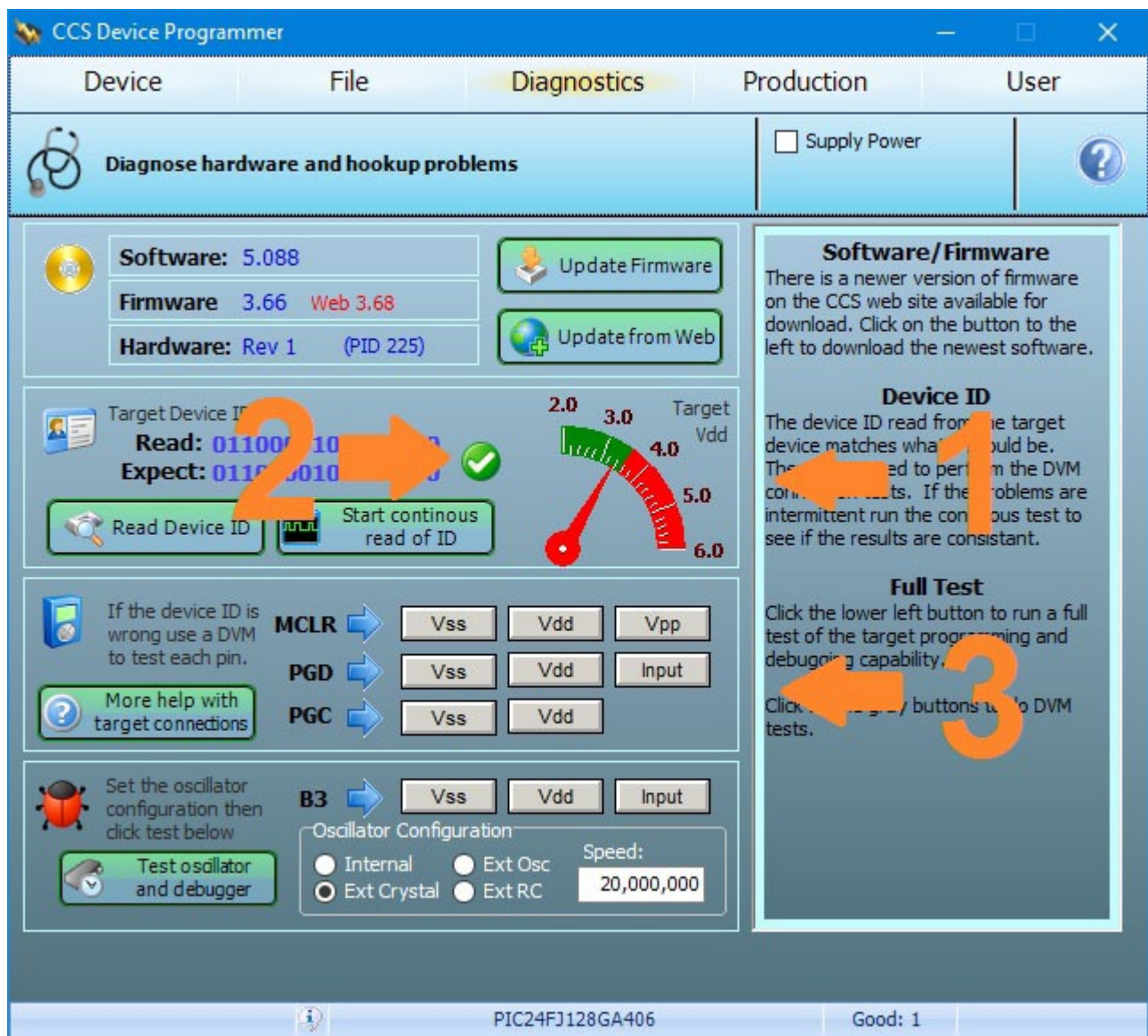
Help with Diag-B3, Pin 6 on Target Board ICSP Connector

During debugging, Pin 6 may be used to interact with a monitor inside of the debug window. This allows for printf()'s and getch()'s in the code to show up in the debugger. Any PIC® MCU I/O pin may be selected and the use of pin 6 is optional. Note that the CCS ICD units can also be used outside of a debugger to have this same capability with the PGC/PGD pins. In that configuration, the ICD acts like a TTL-to-serial converter.

Use of the Troubleshooting Aids on the CCSLOAD Diagnostics Tab

When the Diagnostics Tab is selected, the software will report the measured supply voltage in the target circuit and it will report the device ID that is read from the target device. Note: 12-bit PIC® MCU devices do not report a device ID.

1. Check the voltage on the screen to be sure it matches the target chip. If not, check the cable connections for power and verify the power source is set right.
2. Check to see if there is green checkmark indicating the device ID is right. If so, the hardware connections are likely good. To verify, click on "Start continuous read of ID" and watch the lower right to see if there are intermittent problems that may be caused by noise. If there seems to be a problem, use a scope to view the signals real time.
3. With no green checkmark, you should check the connections for PGC, PGD and MCLR using the gray buttons in the lower part of the window. Use a DVM connected directly to the chip to verify that the voltage at each pin can be changed by the software.



For additional help, please contact CCS Technical Support:
<https://www.ccsinfo.com/contactEmail.php?dept=ts>

References:

Microchip PICkit 2 User Guide

Microchip MPLAB PICkit 5 User Guide

Microchip In-Circuit Serial Programming (ICSP) Guide

CCS FAQ, How do I connect the ICD-S/U to my PIC® MCU hardware?:

https://www.ccsinfo.com/faq.php?page=connect_icd

Using SPI with the CCS Compiler

SPI, or Serial Peripheral Interface, is a synchronous serial communication protocol used to enable data exchange between a master device (like a microcontroller) and one or more peripheral (slave) devices. Microchip's PIC® microcontrollers often come with built-in SPI modules (sometimes labeled as MSSP – Master Synchronous Serial Port). These can be configured either as master or slave via software.

SPI operates using four main lines: SCK (Serial Clock), MOSI (Master Out Slave In), MISO (Master In Slave Out), and SS or CS (Slave Select or Chip Select). The master generates the clock signal on the SCK line and initiates communication by selecting a specific slave using the CS line. Data is then transmitted bit by bit, with the master sending data on the MOSI line and receiving data on the MISO line simultaneously, allowing for full-duplex communication. This simple and fast protocol is commonly used in embedded systems for connecting devices like sensors, memory chips, and displays.

CCS provides a support library for taking advantage of both hardware and software based SPI functionality. Hardware SPI uses the built-in SPI peripheral inside the microcontroller. Software SPI bit-bangs the I/O pins. The `setup_spi()` library is for hardware SPI, while the `#use spi()` library can be used for both hardware and software SPI. The following is an example showing how to use `#use spi()` for software SPI:

```
#use spi(MASTER, SPI_CLOCK=PIN_C3, SPI_DI=PIN_C4, SPI_DO=PIN_C5, MODE=0, \
        BITS=8, STREAM=SPI_STREAM)

BYTE wData[128];
BYTE rData[128];

// Fill wData with the data to transfer

// Send 128 bytes and save the received data to rData
spi_transfer(SPI_STREAM, wData, rData, 128);

// Send 0xAA and receive a byte at the same time
int8 response = spi_xfer(SPI_STREAM, 0xAA, 8);
```

In this example, we are setting the master device. After setting all of the proper parameters in `#use spi()`, either the `spi_transfer()` function or the `spi_xfer()` function can be used to both transfer and receive data on the SPI bus. The `spi_xfer()` function lets you set how many bits of data will be transferred. The `spi_transfer()` function is designed for sending one or more bytes at a time. It is generally faster for larger data transfers, as it handles multiple bytes in a single call. The following is an example of how to set up `#use spi()` for hardware SPI:

```
#use spi(MASTER, SPI1, MODE=0, BITS=8, STREAM=SPI_STREAM, ENABLE=PIN_C0, \
        DO=PIN_C5, DI=PIN_C4)
```

The `setup_spi()` library should be used when you want to configure hardware SPI manually and use register-level access. The compiler has functions to directly access the MSSP hardware SPI for users with special needs. The following is an example showing how to use `setup_spi()`:

```
setup_spi(SPI_MASTER | SPI_CLK_DIV_16 | SPI_L_TO_H | SPI_XMIT_L_TO_H);  
  
spi_write(0xAA);  
int data = spi_read();
```

This example sets the master device and uses the `spi_write()` function to send the data and the `spi_read()` function to read the data.

8-Bit AVR® Support for Programmers



Programming support for all 8-bit AVR® microcontrollers. LOAD-n-GO, Prime8 and ICD-U80 supported. Programming adapter and cables available as separate purchase.

8-bit AVR®
Programming Adapter
53505-1867 | \$25.00

sales@ccsinfo.com
262-522-6500 EXT 35
www.ccsinfo.com/NL0422



Programmer/Debugger Drivers for Windows on Arm® Processors

CCS recently came out with new programmer/debugger drivers. These drivers allow all CCS programmers/debuggers to work on computers running Windows on Arm® processors.

CCS device programmers only work with 32 or 64-bit Windows on Intel®, AMD or Arm® processors, or with Linux on Intel® or AMD processors.

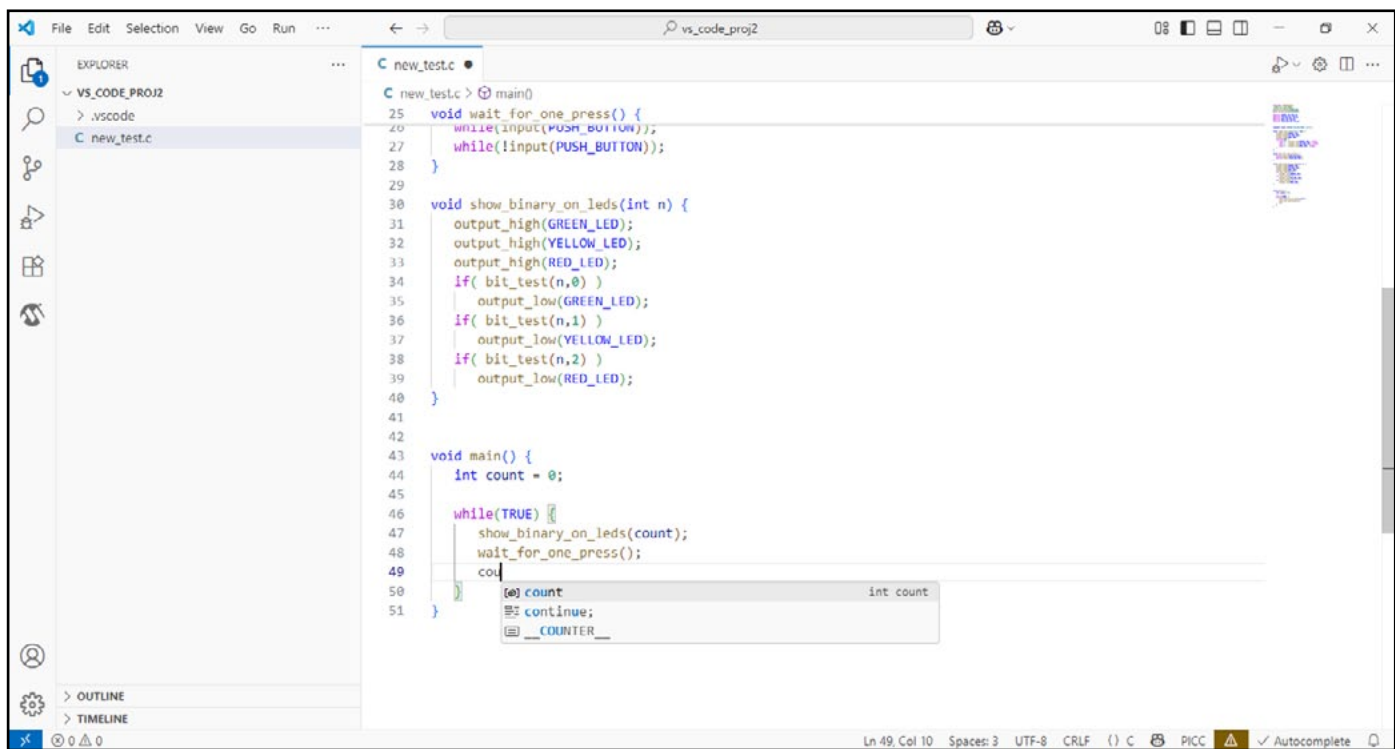
Installation directions for the new drivers can be found on the following FAQ page:
https://www.ccsinfo.com/faq.php?page=arm_drivers

VS Code Extension for CCS C Compiler

Thanks to a gracious CCS customer, a VS Code extension is now publicly available for the CCS C Command-line Compilers. It is an alpha-stage, non-official and non-affiliated VS Code extension, so use at your own risk. The extension is called PICC CCS Support and can be found in the Visual Studio Code Marketplace. Learn more or install from this page:

<https://marketplace.visualstudio.com/items?itemName=pgvt.picc-ccs-support>

The CCS IDE comes with many more features including a debugger, but there is one notable feature that the VS Code extension offers – automatic intellisense. The CCS IDE comes with a code completion feature that is turned on by default. It automatically brings up code completion suggestions for things like elements of a structure after typing the decimal. However, for things like variables and functions, you must start typing the name and then use Ctrl + Space to bring up suggestions. The VS Code extension automatically brings up code completion suggestions for everything as you are typing.



If you are using Visual Studio Code, you may also want to install the MPLAB AI Coding Assistant extension from Microchip. This extension can give detailed answers to questions about a specific Microchip PIC® MCU device that you are programming.

If you are interested in having AI assist you with reading and writing CCS C code, there are at least a few AI models that partially understand code written for the CCS C compiler. These include ChatGPT, Copilot, and Claude.

Programming RGB LEDs

RGB LEDs work by combining three different colors of light - red, green, and blue - to create a wide range of colors. An RGB LED has three tiny light-emitting diodes (one for each color) inside a single unit. By adjusting the brightness of each diode, the RGB LED can produce various colors, including white. Below is the schematic for an RGB LED.

With the specific RGB LED that we used, the red, green and blue LED chips are different brightnesses when operated at the same current. To achieve similar brightnesses for the red, green, and blue chips in the LED package, we chose to use three different resistance values for R1, R2, and R3.

The brightness can also be adjusted by feeding a PWM signal to the LED and varying the duty cycle. The signal is always 0 to V_{dd} but the time the signal is at V_{dd} will vary depending on the duty. For example, 100% duty is full brightness and 0% is off.

Initialization of the PWMs on a PIC18F55Q43:

```
//0 Turns LED on (100%), 32000 turn LED off
#define PWM_RGB_DEFAULT_DUTY 0
//Setup RGB PWM pins
set_pwm1_duty(1, PWM_RGB_DEFAULT_DUTY);
set_pwm2_duty(1, PWM_RGB_DEFAULT_DUTY);
set_pwm3_duty(1, PWM_RGB_DEFAULT_DUTY);
```

```
//Setup one of the 2 output slices that
// the Dual 16-bit PWM module has
setup_pwm1_slice(1, PWM_STANDARD);
setup_pwm2_slice(1, PWM_STANDARD);
setup_pwm3_slice(1, PWM_STANDARD);
```

```
setup_pwm1(PWM_ENABLED | PWM_CLK_FOSC, PWM_RGB_PERIOD - 1, 1); //1KHz PWM
setup_pwm2(PWM_ENABLED | PWM_CLK_FOSC, PWM_RGB_PERIOD - 1, 1);
setup_pwm3(PWM_ENABLED | PWM_CLK_FOSC, PWM_RGB_PERIOD - 1, 1);
```

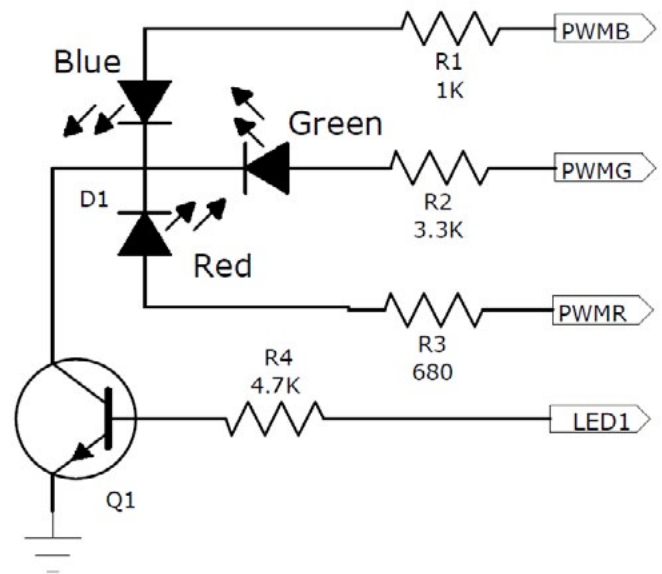
Setting the color and brightness:

```
duty1 = ((uint32_t)PWM_RGB_PERIOD * g_LEDColor.red) / 255;
duty2 = ((uint32_t)PWM_RGB_PERIOD * g_LEDColor.green) / 255;
duty3 = ((uint32_t)PWM_RGB_PERIOD * g_LEDColor.blue) / 255;

duty1 = (PWM_RGB_PERIOD - (((uint32_t)duty1 * g_LEDBrightness) / 100));
duty2 = (PWM_RGB_PERIOD - (((uint32_t)duty2 * g_LEDBrightness) / 100));
duty3 = (PWM_RGB_PERIOD - (((uint32_t)duty3 * g_LEDBrightness) / 100));

set_pwm1_duty(1, duty1);
set_pwm2_duty(1, duty2);
set_pwm3_duty(1, duty3);
```

Watch a video of a project containing programmable RGB LEDs at:
<https://www.ccsinfo.com/content.php?page=led-board>



**\$25 Off a Full
Compiler or Compiler
Maintenance**

CCS C Compiler Savings!



Use Code: News26

More than 30 years experience in software, firmware and hardware design and over 500 custom embedded C design projects using a Microchip PIC® MCU device. We are a recognized Microchip Third-Party Partner.

www.ccsinfo.com



Follow Us!

