August 2021

CCS Inc

www.ccsinfo.com
262-522-6500

<BITS & BYTES> Newsletter

INSIDE THIS ISSUE:

Product Spotlight

CAN BUS FD DEVELOPMENT KIT

This kit enables users to begin CAN FD network development with Microchip's PIC® PIC16 and dsPIC® DSC families. The development kit includes the powerful PCWHD Integrated Development Environment with compiler support for Microchip's PIC® PIC10, PIC12, PIC16, PIC18, PIC24 and dsPIC® DSC families and an ICD-U64 in-circuit programmer/ debugger that supports C-aware real time debugging. The prototyping board has four nodes. CAN drivers and example code are also included.

support@ccsinfo.com sales@ccsinfo.com

The .sym compiler output file provides a good overview of the build. When you archive your source code and hex file saving the .sym file as well can be helpful. The .sym file has the following information:

RAM allocation
ROM allocation
User memory space allocation (when needed)
Source file list with CRC's
Object signatures
Unit list
Compiler settings that affect compilation
Task scheduling (when needed)
Output file list

RAM allocation

```
000     @SCRATCH
001     @SCRATCH
001     _RETURN_
002     @SCRATCH
003     @SCRATCH
004-007 main.a
008-00B main.b
00C-00F main.x
010-013 @ADDFF.P1
010     @PRINTF_X_809.P2
010     @delay_ms1.P3
```

The RAM locations are listed in order with the allocated variable names. Names with an @ before them are compiler generated variables. The main.a notation is used to indicate a local variable x in the function main(). A range indicates multiple bytes for example 004-007 is a four byte variable. For compiler built in functions frequently the .P1, .P2 notation is used for parametter 1 and 2.

Notice there are three variables all sharing location 010. This is because those three functions are never active at the same time so they can use the same memory location.

If there is a out of RAM error, the .sym file is still created and variables that will not fit in memory will have "na" instead of an address.

ROM allocation

```
000322  USB_CLASS_DESCRIPTORS.call
00032E  USB_CLASS_DESCRIPTORS.data
000336  USB_DEVICE_DESC.call
000342  USB_DEVICE_DESC.data
000392  usb_clear_isr_reg
0003A0  usb_cdc_init
```

Again each area of ROM used is listed with the function or data name. Notice the data areas are shown as .call and .data. The .call address is the function that is called to retrieve data and the actual data is saved at the .data address. The .lst file will show the actual ROM data for each allocated area.

User memory space allocation (when needed)

```
     User Memory space: eeprom
       007FFE-007FFE  signature
       007FFC-007FFC  productid
       007FF6-007FFA  serialnumber
```

User memory can be anywhere the read/write algorithm and address range are defined in the code. This shows a memory space called “eeprom” and the data areas defined.

Source file list with CRC’s

```
     Project Directory:
           D:\Projects\

     Project Files:
          ex_usb_bootloader.c           [05-Jul-18 12:26  CRC=4EE7C16D]
          ..\fe\ex_usb_common.h         [26-Jan-17 17:12  CRC=79BAE87A]
          ..\fh\24FJ256GB206.h          [14-Apr-21 15:34  CRC=A1FBF743]
          ..\fe\usb_bootloader.h        [01-Apr-19 11:57  CRC=A5BB5FCA]
          ..\fe\usb_cdc.h               [01-Apr-19 11:39  CRC=D21D1DD7]
```

The project files are listed relative to the show project directory. The file date/times are shown along with the file CRC. This can be valuable in comparing builds on different machines to make sure the right source files are being used. This data is one reason to save the .sym files to ensure future builds are done right.

Object signatures

```
     Source signature=0C7E5CD2
     Program memory checksum=0000
     Hex file CRC=9AF1
```

More data to compare builds. Note that is you use #rom xxx=checksum then the program memory will be 0 because it saves the checksum itself at location xxx.

Unit list

```
     Units:
      project_mcu (main)
      D:\Projects\mcu\report_mcu.o
      D:\Projects\mcu\filter_mcu.o
      D:\Projects\mcu\main_mcu.o
```

For a single compilation unit program only one file will be listed here, your main program. Above is an example of a three unit program.

Compiler settings that affect compilation

```
     Compiler Settings:
     Processor:          PIC18F6722
     Pointer Size:       16
     ADC Range:          0-255
     Opt Level:          9
     Short,Int,Long:     UNSIGNED: 1,8,16
     Float,Double:       32,32
     ICD Provisions:     Yes
```

This section shows key compiler settings that can affect the compilation. Some lines are only shown if they are different from the compiler default.

Task scheduling (when needed)

```
      Task Info:
        Ticks: 100 ns
        Cycle: 100 ms

      Tasks:
        The_first_rtos_task
        The_second_rtos_task
        The_third_rtos_task

      Task Schedule:
         100 ms  640:The_second_rtos_task
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  633:The_first_rtos_task
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  640:The_second_rtos_task
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
         100 ms  643:The_third_rtos_task
      Sync to next 100 ms
```

This shows a deterministic RTOS build with a 100ms time slice and timer resolution of 100ns. There are three tasks shown and the schedule of what happens in each 100ms time slice.

Output file list

```
      Output Files:
         XREF file:          ex_usb_bootloader.xsym
         Errors:             ex_usb_bootloader.err
         Ext Symbols:        ex_usb_bootloader.esym
         INHX8:              ex_usb_bootloader.hex
         Symbols:            ex_usb_bootloader.sym
         List:               ex_usb_bootloader.lst
         Debug/COFF:         ex_usb_bootloader.cof
         Project:            ex_usb_bootloader.ccspjt
         Call Tree:          ex_usb_bootloader.tre
```

Each file created by the compiler is shown again relative to the project directory. Note that the .ccspjt file is both an input and output file. The .xsym and .esym files are only created when compiling in the IDE and those files are only used by the IDE.

Windows 11

By: Mike Primeau

The latest version of Windows, Windows 11, will be rolling out to the public in the near future. Here is what you need to know in order to take advantage of the newest version of the OS while continuing to use CCS, Inc. products.

Most PCs on the market right now will be able to support Windows 11. For those that want to know if their current PC is supported, Microsoft has released the minimum system requirements for installing Windows 11 which are as follows:

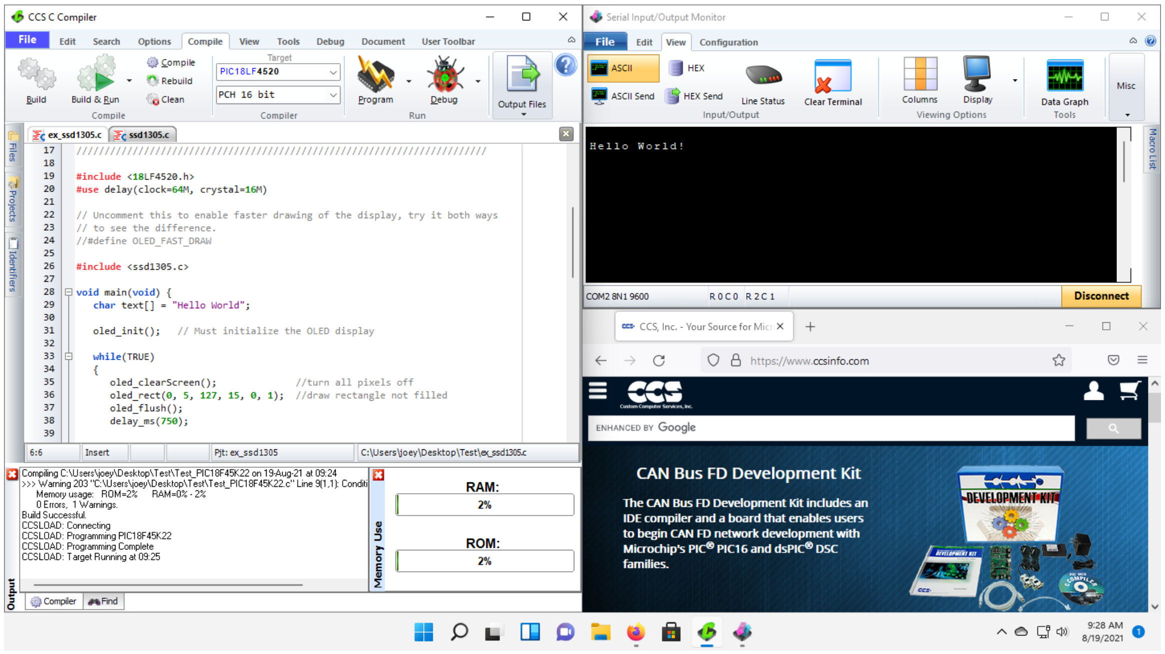
- Processor with a clock speed of at least 1 GHz and has 2 or more cores
- At least 4 GB of RAM
- 64 GB of available storage
- UEFI and Secure Boot capable
- A graphics processor that is compatible with DirectX 12 or later (This can be checked by typing dxdiag into windows search and running the preinstalled app)
- A high definition display (720p) with a screen that is at least 9”
- Trusted Platform Module (TPM) support (Most computers that meet the above requirements and are less than five years old will support TPM)

The majority of PCs that were manufactured in the last few years should meet these requirements. Most of these system requirements are geared towards increasing security on the new OS, partially achieved by preventing older hardware from running Windows 11. Microsoft plans to release tools to help determine hardware eligibility once Windows 11 is widely available.

Windows 11 brings new features and a sleek new look while still functioning very similar to Windows 10 which will make the transition fairly simple. The new Windows includes new features like Widgets and Snap layouts and Desktops. Android apps will also be making an appearance in the new OS.

The Start menu returns in Windows 11, which can be accessed in the middle of the Taskbar. It still includes the ability to search for installed apps as well as view an alphabetized list of apps, just like Windows 10. The new Start menu will be very recognizable to those upgrading from Windows 10.

The new snap layouts feature allows for easier and more predictable window snapping. This allows users to better utilize their screen space for multitasking purposes. Desktops also make a return to Windows 11 allowing users to maintain multiple desktops, allowing for separation between work, school, or personal desktops.



```
////////////////////////////////////////////////////////////////////////////

#include <18LF4520.h>
#use delay(clock=64M, crystal=16M)

// Uncomment this to enable faster drawing of the display, try it both ways
// to see the difference.
//#define OLED_FAST_DRAW

#include <ssd1305.c>

void main(void) {
   char text[] = "Hello World";

   oled_init();   // Must initialize the OLED display

   while(TRUE)
   {
      oled_clearScreen();              //turn all pixels off
      oled_rect(0, 5, 127, 15, 0, 1);  //draw rectangle not filled
      oled_flush();
      delay_ms(750);
```

```
6:6    Insert    Pjt: ex_ssd1305    C:\Users\joey\Desktop\Test\ex_ssd1305.c
```

```
Compiling C:\Users\joey\Desktop\Test\Test_PIC18F45K22 on 19-Aug-21 at 09:24
>>> Warning 203 "C:\Users\joey\Desktop\Test\Test_PIC18F45K22.c" Line 9(1,1): Conditi
    Memory usage:   ROM=2%      RAM=0% - 2%
    0 Errors,  1 Warnings.
Build Successful.
CCSLOAD: Connecting
CCSLOAD: Programming PIC18F45K22
CCSLOAD: Programming Complete
CCSLOAD: Target Running at 09:25
```

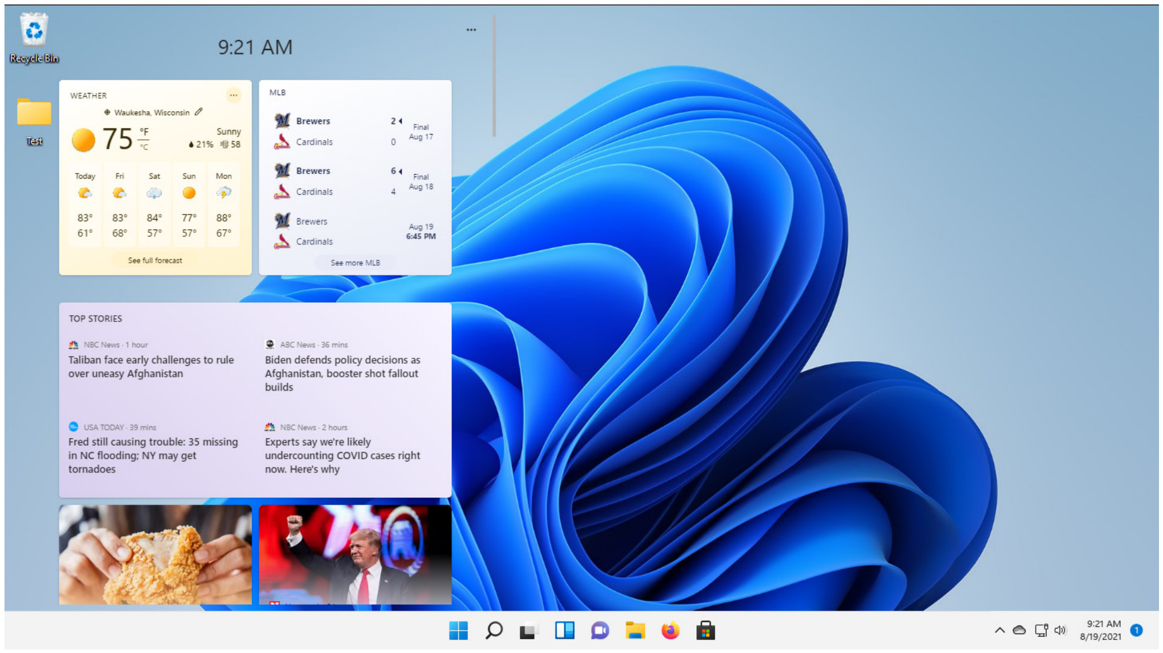
```
Hello World!
```

```
COM2 8N1 9600    R0 C0  R2 C1
```

CAN Bus FD Development Kit

The CAN Bus FD Development Kit includes an IDE compiler and a board that enables users to begin CAN FD network development with Microchip's PIC® PIC16 and dsPIC® DSC families.

The Widgets feature is a new way to view news and information in Windows 11. Widgets can be curated to show you personalized content, news and info that you want to see. The Widgets button is located on the Taskbar and can be hidden if desired, allowing users to choose if they want to take advantage of this feature.



NBC News · 1 hour
Taliban face early challenges to rule over uneasy Afghanistan

ABC News · 36 mins
Biden defends policy decisions as Afghanistan, booster shot fallout builds

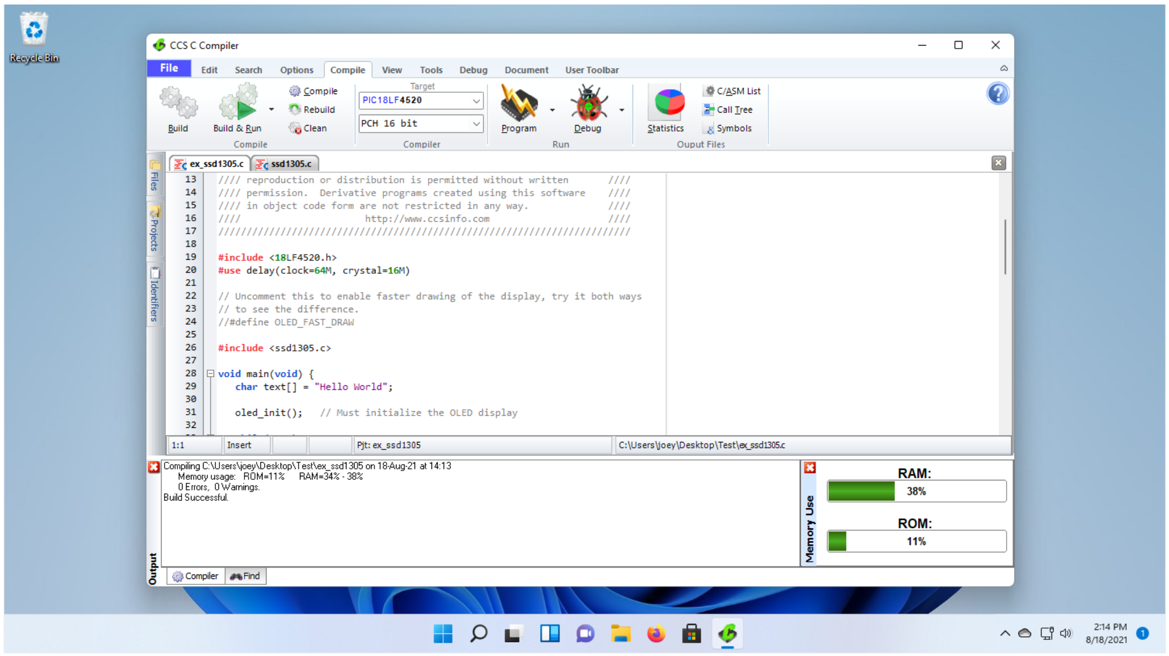
USA TODAY · 39 mins
Fred still causing trouble: 35 missing in NC flooding; NY may get tornadoes

NBC News · 2 hours
Experts say we're likely undercounting COVID cases right now. Here's why

Windows 11 will be introducing Android apps to your PC. Microsoft is planning to allows users to display Android apps in the Microsoft store and download them through the Amazon Appstore. Using common mobile apps on your PC will be an interesting new feature for Windows that Microsoft plans to implement later this year.

For our customers that make use of the CCS C Compiler as well as our programmers/debuggers, there should be little to no difference going from Windows 10 to Windows 11. Microsoft has committed to maintaining compatibility between Windows 10 and 11, this means that the CCS C Compiler will install and run without the need for any extra changes.

This also includes our USB drivers for CCS programmers/debuggers and other software tools. This will allow customers to continue programming Microchip PIC devices on Windows 11 with the same hardware and software they currently use.



```
//// reproduction or distribution is permitted without written      ////
//// permission.  Derivative programs created using this software    ////
//// in object code form are not restricted in any way.              ////
////                         http://www.ccsinfo.com                  ////
/////////////////////////////////////////////////////////////////////////

#include <18LF4520.h>
#use delay(clock=64M, crystal=16M)

// Uncomment this to enable faster drawing of the display, try it both ways
// to see the difference.
//#define OLED_FAST_DRAW

#include <ssd1305.c>

void main(void) {
   char text[] = "Hello World";

   oled_init();   // Must initialize the OLED display

```

```
Compiling C:\Users\joey\Desktop\Test\ex_ssd1305 on 18-Aug-21 at 14:13
    Memory usage:   ROM=11%     RAM=34% - 38%
    0 Errors,  0 Warnings.
Build Successful.
```

Windows 11 preview builds can be installed by becoming a part of the Windows Insider Program. The Insider Program is free and it's a great way for excited Windows users to try out the new OS and check out its new features ahead of time. For those who want to wait, Microsoft plans to begin rolling out upgrades to Windows 10 users in early 2022.

Windows 11 brings a new sleek and clean redesign to Windows while still keeping much of the functionality of the previous Windows 10. Most users should find an easy transition to the new version and should find compatibility with all the apps they are used to using on Windows 10. For the CCS customers that are looking to upgrade, the switch should be fairly easy while allowing you to continue using CCS products.

Pin_Select

By: Richard Ackerman

Almost every new PIC® microcontroller Microchip has released in the last few years has a peripheral called the Peripheral Pin Select (PPS) peripheral. This peripheral is used to assign what pins are used as the peripheral pins for most digital peripheral. This means instead of a specific pin or pins that has to be used with a peripheral you can choose the pin or pins to use with it. This is very useful on low pin count devices were a single pin would have had multiple peripheral functions on a single pin causing you to choose which peripheral to use. The way the PPS peripheral works for peripheral pins that are outputs, the UART TX pin for example, the peripheral is assigned to the pins, and for peripheral pins that are inputs, the UART RX pin for example, the pin is assigned to the peripheral. Because of this it's possible to assign multiple pins to the same output peripheral, but only one pin can be assigned to an input peripheral.

To assign a pin as a peripheral pin the CCS C Compiler has several methods to do this. The primary method is the #pin_select directive. The #pin_select directive is a preprocessor directive used to assign pins at compile time. The format for the #pin_select directive is as follows:

```
#pin_select xx=yy
```

With xx being the peripheral to assign the pin to, see the PIN_SELECT section of the device's header file for a list of the peripherals pins can be assigned to, and yy being one of the pin defines in the device's header file. The following is an example of how to assign the UART1 TX and RX pins using the #pin_select directive:

```
#pin_select U1TX=PIN_C6
#pin_select U1RX=PIN_C7
```

However not all pins can be assigned as a peripheral pin, to determine which can be assigned under the PIN_SELECT section of the device's header is a list of pins that can be assigned as peripheral pins. Additionally not all valid pins can be assigned to every peripheral, some device only allow certain port pins to be assigned to a specific peripheral were as some other devices only allow some pins to be assigned as input peripheral pins. This is device dependent and the device's data sheet should be checked carefully when laying out the peripheral pins for device with a PPS peripheral. The #pin_select directive does have checks to make sure a pin can be assigned to a peripheral, and will generate a compiler error if a pin can't be assigned to a peripheral or if the pin isn't a remappable peripheral pin.

Another method the CCS C Compiler has for assigning pins as peripheral pins is the pin_select() function. The pin_select() function can be used to assign pins at run time. This is useful in cases were more then one pin needs to be assigned to a single output peripheral, if a single pin needs to be used by multiple peripherals or if the pins assignments need to changed, for example, in a case were the code needs to support multiple hardware configurations. The format the pin_select() directive is as follows:

```
pin_select("xx", yy, aa, bb);
```

With xx being the same peripheral names used by #pin_select and yy being one of the pin defines in the device's header file. Additionally aa and bb are optional parameters used to specify whether to do or not to do the unlock and lock procedures, TRUE does the procedure and FALSE doesn't to the procedure. With aa specifying the unlock procedure and bb specifying the lock procedure. When the aa and bb parameters are not specified in the function call the lock and unlock procedures are both performed by default. These optional parameters are most useful when using the pin_select() function to assign multiple peripheral pins sequentially. For example, the following is an example of of how to assign the UART1 TX and RX pins at run time:

```
pin_select("U1TX", PIN_C6, TRUE, FALSE);
pin_select("U1RX", PIN_C7, FALSE, TRUE);
```

Another useful feature of the pin_select() function is that it can be used to unassign the peripheral pins. The format of the function call depends on the whether unassignment is for an output peripheral or an input peripheral pin. To unassign an output peripheral pin the following format is used:

```
pin_select("NULL", yy);
```

With yy being the one of the pin defines in the device's header file. To unassign an input peripheral pin the following format is used:

```
pin_select("xx", FALSE);
```

With xx being same peripheral names used by #pin_select. For example, the following is an example of unassigning the previously assigned UART1 TX an RX pins:

```
pin_select("NULL", PIN_C6, TRUE, FALSE);
pin_select("U1RX", FALSE, TRUE);
```

The final method has for assigning the peripheral pins is some for the #use directive can make the peripheral pin assignments, the #use pwm() directive, for example. For #use directive that this is supported with the format is similar to the following, see the help file entry for the #use directive for the exact format:

```
#use pwm(CCP1, output=PIN_B0)
```

The above will make the assignment of PIN_B0 as the CCP1 output pin. This method is currently not supported by all of the #use directives, currently only the #use pwm() and #use capture() directives support this feature. For the other #use directives #pin_select should be used to assign the peripheral pins before the #use directive line. The #use rs232() is an example were the pins should be assigned using the #pin select directive before the #use rs232() line if the hardware UART pins are to be used on a PPS device were the UART pins are remappable peripheral pins. Without the assignments the compiler will implement a software RS232 instead. For example, the following will assign the UART1 TX and RX pins and setup the RS232 to use the hardware UART peripheral:

```
#pin_select U1TX=PIN_C6
#pin_select U1RX=PIN_C7

#use rs232(xmit=PIN_C6, rcv=PIN_C7, baud=115200, stream=U1_STREAM)
```

USB Development Kit

Includes

- USB Prototyping Board
- In-Circuit Debugger/ Programmer
- Exercise Tutorial
- 9V AC Adapters and Cables

sales@ccsinfo.com
262-522-6500 EXT 35
www.ccsinfo.com/NL521

Summer is for C Compiler Savings

$25 Off a Full Compiler or Compiler Maintenance

Use Code:Summer2021

More than 25 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!