May/June 2022

www.ccsinfo.com
262-522-6500

<Bits & Bytes> Newsletter

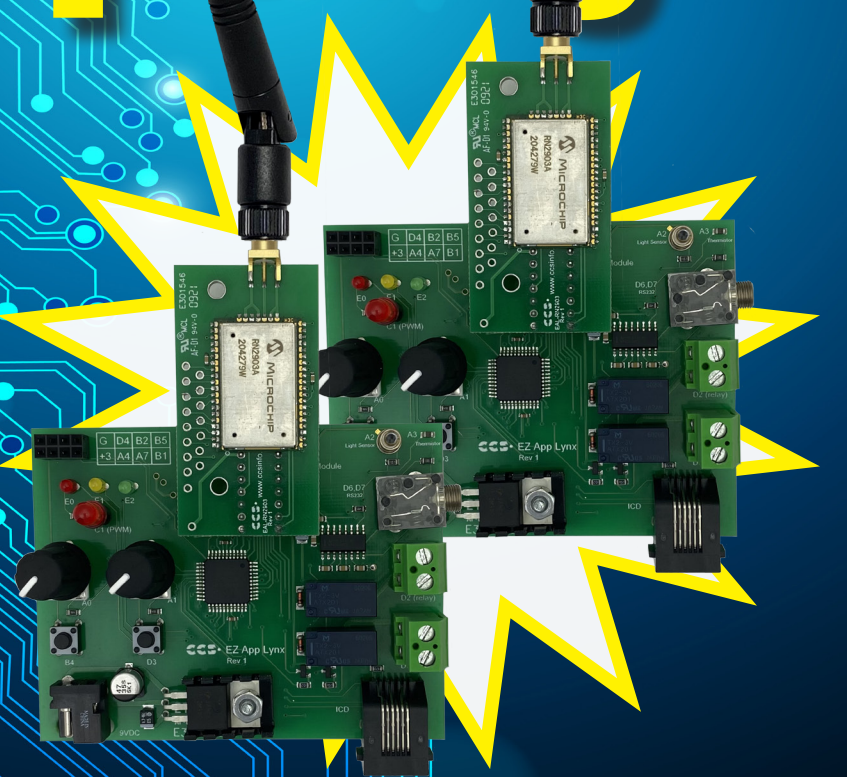
INSIDE THIS ISSUE:

Pg 2
Adding Serial Numbers to Production Images

Pg 4
Archiving, Moving or Branching a Project

Pg 5
PIC® MCU Reprogramable Pins (RPxxx)

Product Spotlight



LONG RANGE RF PROTOTYPING BOARD

The Long Range RF Development Boards can have a 9 mile RF link with a PIC® MCU and LoRaWAN®. The included module is compatible with LoRaWAN® networks.

support@ccsinfo.com
sales@ccsinfo.com

Adding Serial Numbers to Production Images

The CCS tool suite has a number of features to help adding serial numbers to product firmware. In your source code the primary method uses the pre-procssor directive #serialize. The CCS device Adding Serial Numbers to Production Images. The CCS tool suite has a number of features to help adding serial numbers to product firmware. In your source code the primary method uses the pre-processor directive #serialize. The CCS device programmers must be used to take advantage of this feature. It works by adding a special comment line to the hex file.

Location of the serial number

The usual way to handle the location is to use a const declaration like one of the following:

```
const  int32  serial_number;
const  char   sn[7];
```

Then in #serialize add something like id=serial_number.

You can also place the serial number in EEPROM by specifying an address like:

```
dataee=0x10
```

Format

Usually the format can be figured out from the const however you can specify it using one of:

`binary=x` - x is the number of bytes.

`string=x` - x is the number of characters.

`unicode=n` - If n is a 0, the string format is normal unicode.
For n>0 n indicates the string number in a USB descriptor.

Serial number source

There are four ways to specify where the serial number comes from:

`file=`"filename.txt" - The file x is used to read the initial serial number from, and this file is updated by the ICD programmer. It is assumed this is a one line file with the serial number. The programmer will increment the serial number.

`listfile=`"filename.txt" - The file x is used to read the initial serial number from, and this file is updated by the ICD programmer. It is assumed this is a file one serial number per line. The programmer will read the first line then delete that line from the file.

`next=`"x" - The serial number X is used for the first load, then the hex file is updated to increment x by one.

`prompt=`"text" - If specified the user will be prompted for a serial number on each load. If used with one of the above three options then the default value the user may use is picked according to the above rules.

Logging

`log=xxx` - A file may optionally be specified to keep a log of the date, time, hex file name and serial number each time the part is programmed. If no id=xxx is specified then this may be used as a simple log of all loads of the hex file.

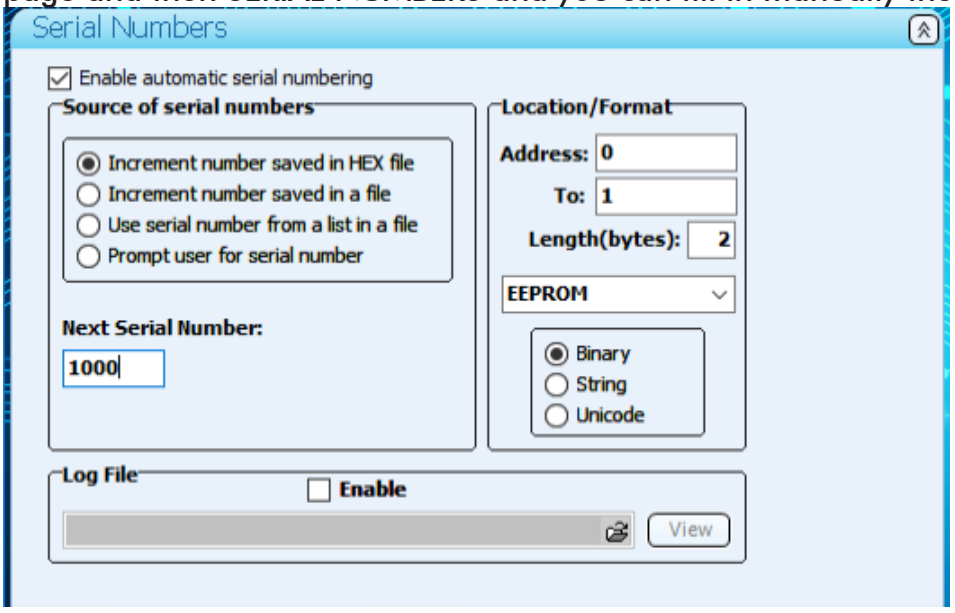
Examples:

```
//Prompt user for serial number to be placed at address of serialNumA
//Default serial number = 200

int8 const serialNumA=100;
#serialize(id=serialNumA,next="200",prompt="Enter S/N")

//Adds serial number log in seriallog.txt
#serialize(id=serialNumA,next="200",prompt="Enter S/N",
           log="seriallog.txt")

//Retrieves serial number from serials.txt
#serialize(id=serialNumA,listfile="serials.txt")

//Place serial number at EEPROM address 0, reserving 1 byte
#serialize(dataee=0,binary=1,next="45",prompt="Put in S/N")

//Place string serial number at EEPROM address 0, reserving 2 bytes
#serialize(dataee=0, string=2,next="AB",
           prompt="Put in S/N")
```

USB_STRING_DESC is a table of USB strings and is read by the CCS C Compiler's USB stack when the host PC reads a string from the PIC®. A value of 3 is passed to the Unicode parameter, to tell the #serialize that the serial number should be encoded as a Unicode string and that it should overwrite the 4th string in **USB_STRING_DESC** (the first string in **USB_STRING_DESC** is 0). In the device descriptor for your USB device, the field identifying which string to use for the serial number should also have been set to 3.

```
#serialize(id=USB_STRING_DESC, unicode=3, prompt="SN#")
```

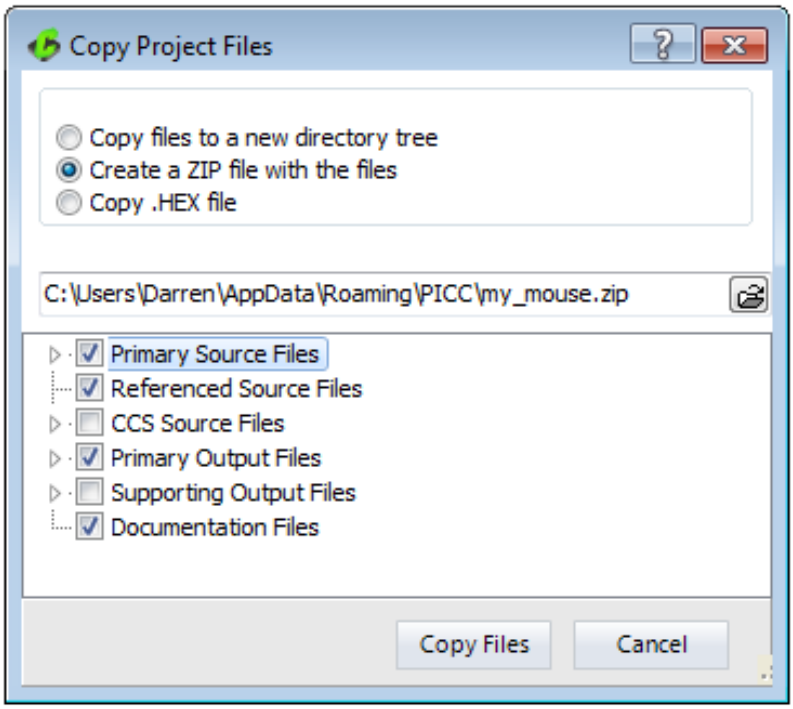
<u>Using a CCS device programmer but not the compiler?</u>
The same features can be used by editing the hex file using the CCSload utility. Select the FILE page and then SERIAL NUMBERS and you can fill in manually the parameters:



Save or Save-As the hex file and the serial number works as it did with the compiler directives.

Archiving, Moving or Branching a Project

The CCS IDE has an easy way to make a copy of a project. The feature is accessed by using FILE > COPY PROJECT. The dialog box looks like this:



The lower selection allows you to select an entire group of files or expand the group and select specific files. The references source files include files supplied with the compiler like stdlib.h. The supporting output files are files used by the IDE and not usually opened outside the IDE. Documentation files are any files attached to the project using the navigation bar.

The location entry in the center allows you to specify the destination. Use the folder icon on the right to browse and/or create a new location.

The functions are as follows:

Copy files to a new directory tree
This simply makes a copy of the designated files to a new directory location. The .ccspjt file usually references the project files using a relative path so the project should compile right in the new location.

Create a ZIP file with the files
This makes a zip file with the designated files. This is ideal for a archive or if you need to send a project to CCS for analysis.

Copy .HEX file
Despite the name this function can quickly make a copy of any combination of files to another location. For example you may need to make a copy of the project hex file to a public location where others can use it for testing or production. By default just the project hex file is selected however if you select some other files that now becomes the default so you can quickly use this function to make the same copy when needed.

PIC® MCU Reprogramable Pins (RPxxx)

Many newer Microchip PIC® microcontrollers have re-programmable peripheral pins (RP). These pins allow the user to dynamically allocate peripherals to these pins, such as external interrupts, input capture, PWM, serial, timers and more. This offers the designer great flexibility when designing a product since the functionality of these pins can be changed at run-time. The data sheet for a device will list he pin assignments and these pins are denoted as either RPxx or RPIxx, where xx is the RP pin number. Pins that are RPIxx can only be programmed as an input (timer input, serial input, interrupt input, etc), whereas RPxx pins can be programmed either as an input or output (PWM output, serial output, etc).

Static Assignments in C

The static method for assigning I/O pins to a peripheral is the #pin_select directive. The #pin_select directive is a preprocessor directive for assigning I/O pins to peripherals and is executed before main() starts. The syntax for this command is as follows:

```
#pin_select function=pin
```

A list of functions and pins that can be used with the #pin_select directive is located in the device's header file near the top of the file, opening the device's header file (like 18F25K42.h) and searching for #pin_select is the quickest way to find them. The following is an example of how to assign pins to the UART1 RX and TX pins:

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

When using RP pins with a peripheral library, such as #use rs232(), the #pin_select must come before the #use directive, for example:

```
#pin_select U1TX=PIN_C6
#pin_select U1RX=PIN_C7
#use rs232(UART1, baud=9600, stream=U1)
```

There is a special method for assigning the peripheral pins is inside the #use pwm and #use capture directives. Future compiler release may allow this in other #use directives as well. Here is an example:

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

The above will make the assignment of PIN_B0 as the CCP1 output pin.

Dynamic Pin assignments

In addition to #pin_select the CCS C Compiler also provides the pin_select() function for assigning pins to a peripheral. The pin_select() function can be used to assign, reassign and unassign pins to/from a peripheral at run-time. This allows the flexibility of using the same pin for multiple peripherals or using pins as both peripheral pins and I/O pins. The basic pin_select() function uses the following syntax: pin_select("function", pin);. The functions and pins are the same as what is used with the #pin_select directive, the only difference being that the function is passed as a constant string. The following is an example of how to assign pins to the UART1 peripheral:

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

To unassign a pin from a peripheral depends on whether it an input peripheral or an output peripheral. To unassign a pin from an output peripheral is done as follows:

```
pin_select("NULL", PIN_C6); //unassign PIN_C6 from output peripheral.
```

To unassign a pin from an input peripheral is done as follows:

```
pin_select("U1RX", FALSE);  //unassign pin from U1RX input peripheral.
```

Because of how output peripherals are assigned to RP pins it is possible to assign multiple pins to the same output peripheral when using the pin_select() directive. For example the following will assign multiple pins to the CCP1 peripheral:

```
pin_select("CCP1OUT", PIN_B0);
pin_select("CCP1OUT", PIN_B1);
```

This method of tying several pins to the same output can only be performed with the pin_select() function, #pin_select cannot be used to do this.

A more advanced form of the pin_select() directive is as follows:

```
pin_select("function", pin,  unlock,  lock);
```

In order to change the pin assignments at run time the pins must be first specifically unlocked to prevent run away code from changing a pin assignment. The optional unlock and lock are 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. When the lock/unlock parameters are not specified in the function both are 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 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);
```

Alternate pin assignments

Before the RP pins came out some chips allowed select peripherals to have multiple (usually just two) pins that can be assigned. This is done either by a fuse (like CCP2B3 and CCP2C1) or using an internal register.

In the case the selection is by fuse the #fuse directive must be inserted in the code and then the compiler will treat that pin as a peripheral. For example:

```
#fuses CCP2C1
```

In the case that the register assignments are made by register the built in functions will have an option for the assignment. See the header file for the device. The UART assignments are made with the #use rs232 by specifying one of the alternate pins.

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

CCS Inc

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.

CCS Inc

Follow Us!

www.ccsinfo.com

AVR® is a registered trademark of Microchip Technology Inc.