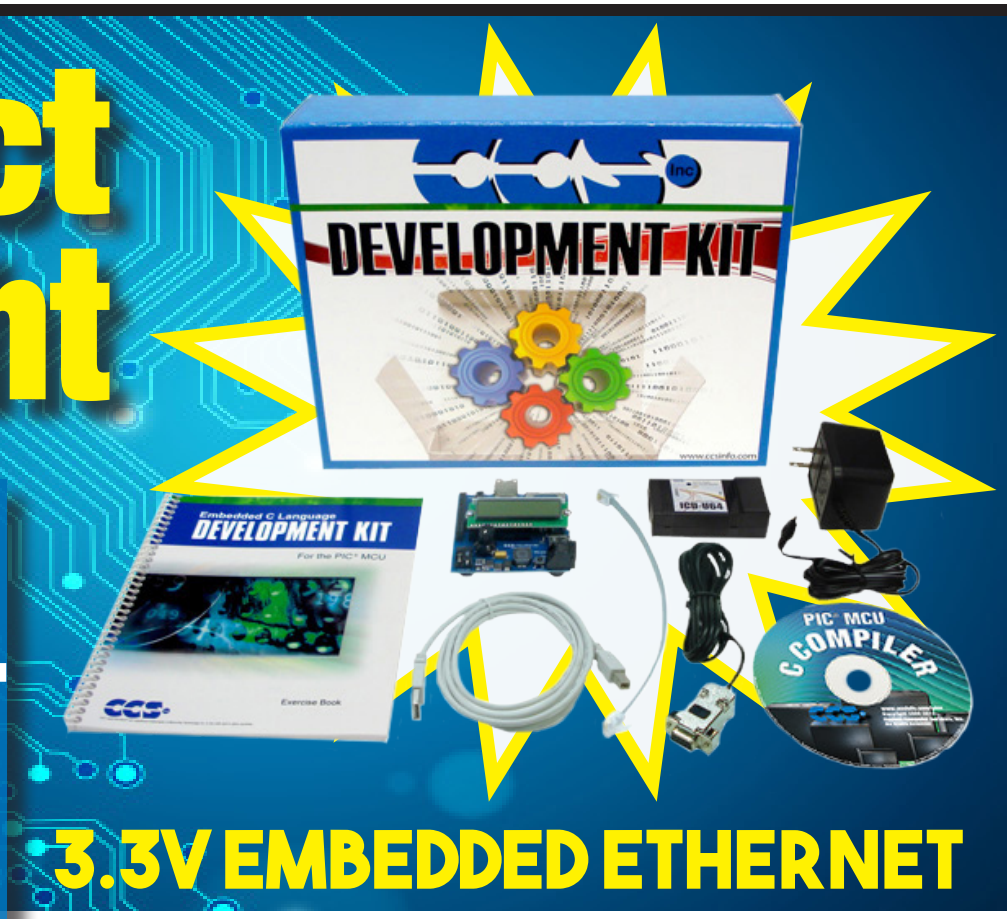
April 2021

www.ccsinfo.com
262-522-6500

<Bits & Bytes> Newsletter

Product Spotlight



3.3V EMBEDDED ETHERNET

The 3.3V Embedded Ethernet Development Kit features Microchip's PIC® PIC18F67J60. This kit includes the powerful PCWH Integrated Development Environment with compiler support for Microchip's PIC® PIC10, PIC12, PIC16 and PIC18 families and an ICD-U64 in-circuit programmer/debugger that supports C-aware real time debugging.

INSIDE THIS ISSUE:

Using Overloading to Deal with Mixed ROM and RAM Data

By: Mark Siegesmund

The CCS C Compiler supports C++ like function overloading. Function overloading allows the user to develop two functions with the same name, but have different parameters or return types. The CCS C Compiler can distinguish between RAM and ROM pointers when overloading. This is useful on the PIC® MCU which uses the Harvard architecture, meaning accessing the RAM and ROM is different. Since accessing RAM and ROM is different, the compiler needs to know which one the user is accessing so it can use the proper method.

Here is a simple example of function overloading using ROM and RAM pointers:

```
int Checksum(rom char * ptr) {
   int cs;
   while(*ptr!=0) // *ptr reads from ROM
      cs += *ptr;
   return cs;
}

int Checksum(char * ptr) {
   int cs;
   while(*ptr!=0)  // *ptr reads from RAM
      cs += *ptr;
   return cs;
}

...

char data[10] = "Foo Bar";
const char hex[] = "0123456789ABCDEF";

label_cs = checksum("Hello World");  // Calls first funct
hex_cs = checksum(hex);              // Calls first funct
data_cs = checksum(data);            // Calls second funct
```

Checksum() has two functions, one for handling ROM pointers and one for handling RAM pointers. In the CCS C Compiler the 'rom' keyword specifies the ROM flash memory, if this is not specified the CCS C Compiler will use RAM. When Checksum() is called with the "Hello World" as a 'rom char *'

Some compiler built in functions deal with conversions from ROM to RAM pointers. For example strcpy() has two forms:

```
strcpy(ramstring, "Hello World");
strcpy(ramstring, buffer);
```

This is so the compiler can easily deal with awkward syntactical issues like this:

```
char  string[] = "ABCDEFGHIJ";
```

ROM string moved to RAM and string is a RAM pointer. The CCS C Compiler also supports this syntax with all the appropriate conversions:

```
char string[10];
string = "Hello World";
```

Note that %s in a printf() is able to figure out if the string is in RAM or ROM however most string functions in string.h do not have overloaded versions for ROM data. For example if you want to do strcat(string,"...") then you must first make a function like this:

```
char *strcat(char *s1, rom char *s2)
{
   unsigned char *s;

   for (s = s1; *s != '\0'; ++s);
   while(*s2 != '\0')
   {
      *s = *s2;
      ++s;
      ++s2;
   }

   *s = '\0';
   return(s1);
}
```

Here is a more complex example of function overloading based on ROM and RAM pointers:

```
struct {
   union {
      rom char* pRom;
      char* pRam;
   };
   int1 isRom;
} savedString;

void SetSavedString(rom char* pRom) {
   savedString.pRom = pRom;
   savedString.isRom = TRUE;
}

void SetSavedString(char* pRam) {
   savedString.pRam = pRam;
   savedString.isRom = FALSE;
}

void PrintSavedString(void) {
   if(savedString.isRom)
      printf("%s", savedString.pRom);
   else
      printf("%s", savedString.pRam);
}

SetSavedString((rom char*)"Hello World");
SetSavedString((char*) "foo bar");
```

SetSavedString() saves a (char*) or (rom char*) depending on what was passed in by the user to memory, and sets a flag denoting the type of pointer that is being used. The PrintSavedString() checks the flag to select the proper pointer, and prints the string.

Long Range RF Communications

By: CCS Staff

Introducing the new CCS Long Range RF Development Kit. LoRa® is low-power wide-area network protocol that can be used to periodically communicate sensor data. LoRa® uses a proprietary spread spectrum modulation that is similar to and a derivative of Chip Spread Spectrum (CSS) modulation. This allows for long-range transmission, up to 9 miles, with low power consumption. This makes it ideal for battery applications that only needs to periodically transmit sensor data.

The CCS Long Range RF Development Kit comes with two development boards with Microchip's RN2903 modules. The RN2903 modules are designed to communicate using LoRa® modulation using the US frequency band. This is everything needed to develop a LoRa® peer-to-peer (P2P) implementation. The development kit's exercise manual has several exercises for setting up testing different types of LoRa® P2P networks using the provided RN2903 and LoRa® P2P drivers.

Additionally, the development kit can be used the to develop code to communicate on a LoRaWAN® network. LoRaWAN® is a upper level network protocol, developed by the LoRa® Alliance, that uses LoRa® modulation for creating a cloud-based network. In LoRaWAN® networks end-devices periodically send messages to a gateway, which forwards it to a network server which in turn forwards the message to an application server. Any response for the end-device is then forward back in the reverse order. The development kit can be used to develop and test end-devices for a LoRaWAN® implementation, using the provided RN2903 and LoRaWAN® driver. Testing the LoRaWAN® end-device will require at minimum an external LoRaWAN® gateway, not provided with the Long Range RF Development Kit.

CCS COMPILER FEATURE FOCUS

The addressmod feature allows creating new address spaces for variables that can use user defined functions to perform the memory access. The compiler allows a developer to create a new address space for C variables. This address space need not be in PIC memory. It could be an external EEPROM, an LCD screen or even another processor.

```
void EEread(int32 address, int8 * ram, int bytes) {
   for(int i=0;i<bytes;i++,ram++,addr++)
      *ram=read_eeprom(address);
}

void EEwrite(int32 addrress, int8 * ram, int bytes) {
   for(int i=0;i<bytes;i++,ram++,addr++)
      write_eeprom(address,*ram);
}

addressmod (EEPROM, EEread, EEwrite,
            getenv("EEPROM_ADDRESS"),
            getenv("EEPROM_ADDRESS")+0xFF);

...

EEPROM int32 myvar;

oldvalue=myvar;
myvar++;
```

Here is a simple example to create an address space for EEPROM. First we need to write read and write functions for our virtual memory and then tell the compiler about the new space. Finally, we can define and use variables in that space.

The CCS Compiler Comes with DMX512 Bus Lighting Drivers

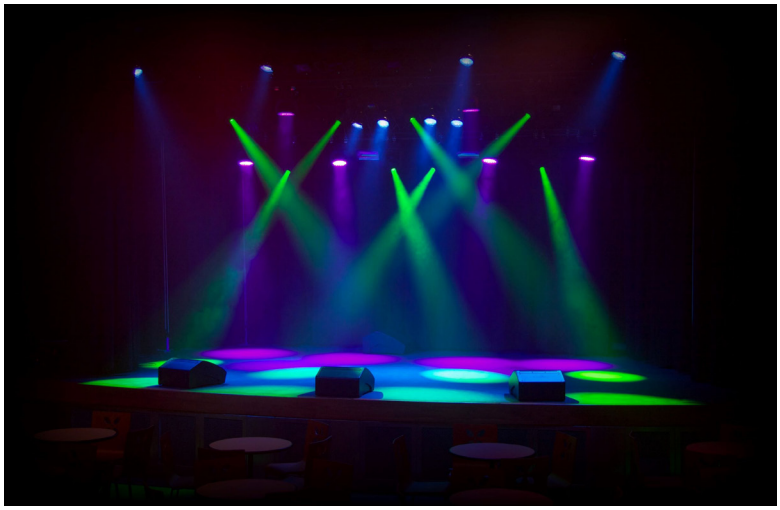
By: Richard Ackerman

DMX512 is a serial protocol that is commonly used to control stage lighting and effects. It was originally intended as a standardized method for controlling light dimmers. It soon became the primary method for linking controllers, a lighting console for example, to dimmers and special effect devices such as fog machines and intelligent lights.

A DMX512 network employs a multi-drop bus topology with nodes strung together in a daisy chain. A network consist of a single DMX512 controller, which is the master of the network and only transmitter, and one or more slave devices. Each slave device has an IN connector and usually an OUT or THRU connector, were as the controller only has an OUT connector. The controller's OUT connector is connected via a DMX512 cable to the first slave's IN connector. A second cable then links the OUT or THRU connector of the first slave to the IN connector of the next slave in the chain, and so on until all slaves are connected. The specification requires a terminator to be connected to the final OUT or THRU connector of the last slave on the daisy chain.

A DMX512 network is called a "DMX universe." The OUT connector on a DMX512 controller can control a single universe. Each universe operates up to 512 channels with each channel's parameter ranging between 0 and 255. A controller simply changes the values of these parameters.

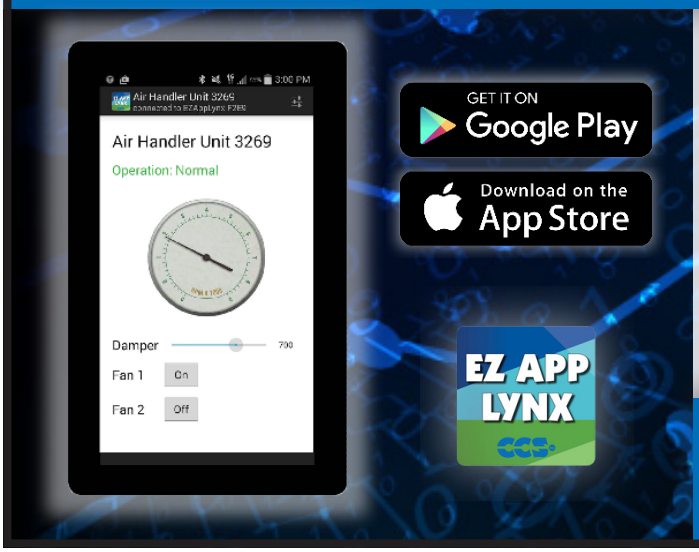
For the DMX512 protocol, the controller transmits asynchronous serial data at 250 kbits/s. The data format is fixed at one start bit, eight data bits, and two stop bits. Each data frame consists of the following, a break, mark-after-break, slot 0 (Start Code) and up to 512 slots of channel data, each containing one byte. The break, which signals the end of one packet and the start of another, cause the slaves to start reception and also serves as a position reference for the data bytes within in the packet. The first slot, slot 0, is reserved for a start code that specifies the type of data in the packet.



The CCS C Compiler comes with a DMX512 driver, dmx.c, which can be used to develop code for either a DMX512 controller or a DMX512 slave device. When setup for a DMX512 controller the following functions are provided, DMXInit() to initialize the driver, DMXSetChannel() to set the specified channel to the specified value, DMXGetChannel() to get the current set value for the specified channel, and DMXCommit() to transmit the DMX512 channel data to the slave device. When setup for a DMX512 slave device the following functions are provided, DMXInit() to initialize the driver, DMXKbhit() to determine if new data as been received, and DMXGetd() to retrieve the DMX data.

In addition to the driver, the CCS C Compiler also provides the following two examples: ex_dmx_controller.c and ex_dma_slave.c, showing how to use the DMX512 driver to build code for a DMX512 controller and a DMX512 slave device.

EZ APP LYNX



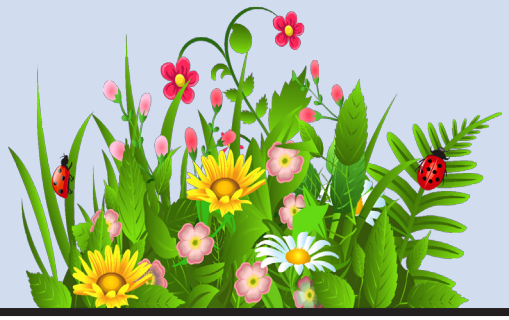
PIC® MCU C Library can interface your device to smart phones and tablets with Bluetooth®

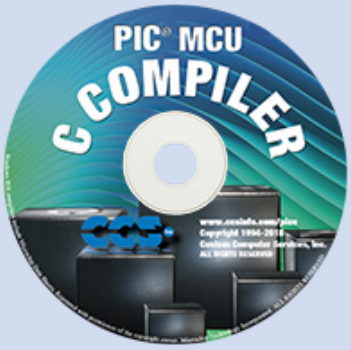
sales@ccsinfo.com
262-522-6500 EXT 35
www.ccsinfo.com/NL421

CCS Inc

Spring into Coding Projects with a C Compiler

$25 Off a Full Compiler or Compiler Maintenance





Use Code: Spring2021

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

CCS Inc

Follow Us!