December 2020

www.ccsinfo.com
262-522-6500

<Bits & Bytes> Newsletter

PRODUCT SPOTLIGHT

EMBEDDED ETHERNET DEVELOPMENT KIT

The Embedded Ethernet Development Kit supports development with Microchip's ENC28J60 Ethernet Transceiver

INSIDE THIS ISSUE:

- Configuration Management
- Linux Headaches
- Light Sensors to Calculate Speed of a Toy Car

Configuration Management with the CCS Compiler Tool Suite

There are many version control programs available to help keep track of software changes from one version to another as well as shared code between projects. We use a free SVN here at CCS, but there are many good free and commercial products out there. This article addresses how to best use features in the CCS compiler tool suite to augment the basic capabilities of a version control system.

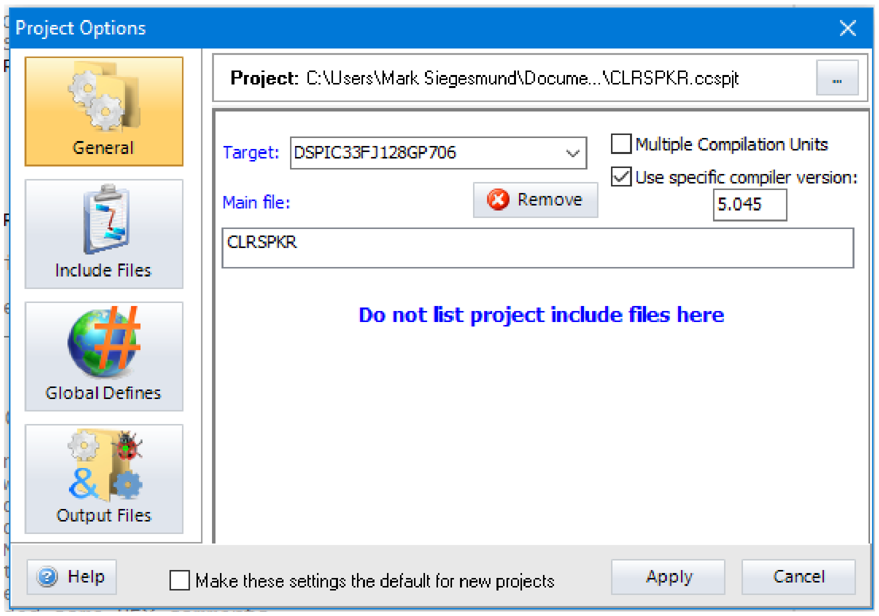
One problem that frequently comes up is attempting to rebuild a project that was built years ago. This is usually done before making an update to make sure the right code is being used as a starting point. Sometimes users even have trouble getting the same build result on different PC's even using the current software. There are three considerations for getting an exact hex file from source:

1. Compiler Version Optimization changes are made in almost every release, although the software may be functionally the same, the hex file can be different. If it is important that the hex file is identical, then including code like the following in the source will help:

```
#if (getenv("VERSION_STRING") != "5.045")
#error Wrong compiler version, use 5.045
#endif
```

Users should also archive the compiler installer for that version. If it is lost, users can always contact CCS with the reference number, compiler name (like PCW) and the version needed.

The IDE compiler allows for multiple versions to be available at the same time. Users can also fix the version for a specific project under OPTIONS > PROJECT OPTIONS so the IDE will always use that version for that project.



Project: C:\Users\Mark Siegesmund\Docume...\CLRSPKR.ccspjt

Target: DSPIC33FJ128GP706

☐ Multiple Compilation Units

☑ Use specific compiler version: 5.045

Main file: CLRSPKR

Do not list project include files here

☐ Make these settings the default for new projects

If there is a difference in builds between versions users can use the IDE feature TOOLS > FILE COMPARE to compare the old and new .lst files. This will identify all differences in the generated code. A review of the differences should help deciding if there is cause for concern. It is always a good idea to archive the .lst file with the source to help in this analysis. Note that the compiler version is at the top of the .lst file.

2. Include Files It is important to use the same include files to get the same result. It is best to archive all include files used for a project, including those provided by the compiler in the version control system. This is usually the cause of a program compiling differently between two PC's. The compiler has an output file that can help users identify all the included files and the exact file being used. In the .sym file there will be a section that looks like this:

```
Project Directory:
    C:\Users\Mark Siegesmund\Documents\CCS C Projects\CLRspkr\

Project Files:
    CLRSPKR.c                                                28-Feb-20 16:43  CRC=AB399B75
    CLRSPKR.h                                                24-Feb-20 10:47  CRC=33050B86
    hw.h                                                     25-Sep-19 09:46  CRC=C304F766
    virtual_eeprom.c                                         08-Jul-19 09:50  CRC=AF4920D9
    TLV320AIC12k.c                                           28-Aug-18 09:24  CRC=CA62A630
    addr_trap.h                                              20-Nov-12 15:48  CRC=9D66BD77
    dma_utility.c                                            30-Jul-19 07:31  CRC=35CEFA89
    fft.h                                                    11-Apr-13 09:47  CRC=D9E7B40A
    ..\..\..\..\..\Program Files (x86)\PICC\Drivers\math.h  22-Sep-16 08:36  CRC=9CF8C837
    globals.h                                                08-Aug-19 09:14  CRC=8E34464F
    sine_window.h                                            17-Dec-10 17:41  CRC=4D83B664
    ..\..\..\..\..\Program Files (x86)\PICC\Drivers\sw256.c 17-Dec-10 17:19  CRC=AA96313F
    ..\..\..\..\..\Progr...x86)\PICC\Devices\33fj128gp706.h 24-Feb-17 08:25  CRC=0E4843A5
    init.c                                                   11-Jan-11 09:56  CRC=34934064
    filter_utility.c                                         20-Nov-12 16:33  CRC=9F95FD98
    dsp_data_util.c                                          26-Oct-12 15:11  CRC=F724C45C
    filter.c                                                 16-Apr-19 12:20  CRC=49BA2659
    CLRSPKR_AGC.c                                            21-Feb-20 10:35  CRC=01391997
    tone.c                                                   14-Apr-15 21:18  CRC=F40A1B74
    tick.h                                                   10-Apr-13 12:07  CRC=F8B87750
    serial.c                                                 25-Sep-19 09:46  CRC=52ADD277
    morse.h                                                  10-Apr-13 12:02  CRC=CCBA2E6D
    tick.c                                                   27-Aug-12 09:12  CRC=F4C536AD
    morse.c                                                  12-Feb-20 15:50  CRC=A4F14A69
    CLRSPKR_PTT.c                                            24-Feb-20 10:39  CRC=716D54BE

Source signature=98990BC4
Program memory checksum=CCE7
Hex file CRC=0210
```

Under "Project Files" is listed every source file used to build the project along with the file date of the source file and its CRC. The files are relative paths with respect to the project directory shown above the list. This is a great help in making sure all the include files have been archived and that the same files are being used for a build. Archiving the .sym file will help in figuring out what was done.

Notice the .sym file also includes a CRC of the entire source code base and a CRC of the hex file. The CRC shown in the .sym file will match the CRC shown at the end of the .hex file. Users can open the .hex file in notepad to check it. The CCS device programmers will users if the hex file itself does not match the CRC due to a corrupted file.

3. Changes Outside the Source Code

There are a few things that can affect the generated code outside of the source code. The most obvious is compiler switches that might be included on the command line for a command line compile or from MPLAB. For example optimization level. The CCS IDE does not have code generation options as we encourage users to include such options in the source code (like: #opt 5).

Pre-processor defines can also be included on the command line or in the IDE under global defines. This is a great way to use the same code to produce different versions of the firmware. In the CCS IDE saving the .ccspjt file will have those defines saved. For command line compiles, saving the batch file that does the compiles might be enough. Users can also insert comments in the hex file to indicate what defines were used. For example:

```
#hexcomment\ Built with CCS PCD Compiler version __PCD__
#ifdef HAS_PTT
#hexcomment\ Built with "HAS_PTT" defined
#endif
```

The end of the .hex file will look like this:

```
; Built with CCS PCD Compiler version "5.098"
; Built with "HAS_PTT" defined
;DSPIC33FJ128GP802
;CRC=175B  CREATED="24-Nov-20 11:02"
```

Note that the \ puts the comments at the end of the file. Comments at the top of the file are displayed to the user each time a chip is programmed if the CCS device programmers are used. This can be helpful for example if different builds are for different products. For example:

```
#ifdef MAGIC_6000
#hexcomment This is for the MAGIC 6000 only
else
#hexcomment This is for the MAGIC 3000 only
#endif
```

Finally if the code uses features like __DATE__ then the generated hex might be expected to mismatch. For example:

```
      const  char PROGRAM_ID[] = {"CLRspkr built on " __DATE__};
```

will cause the date of the build to be part of the hex file.

In summary when archiving the source, make sure to include all the include files listed in the .sym file. Save the .lst, .sym, .hex and .ccspjt files with the source. Use #hexcoment to insert all configurable items in the .hex file. Save the compiler installer itself for versions used for production builds.

CCS Inc COMPILER Feature Focus

#serialize

can be used to add serial number instructions into the hex file

```
const int32 SerialNum = 0;
#serialize(id=serialNum, prompt="Enter the serial number")
...
printf("S/N: %LU \r\n",SerialNum);
```

The above code, when used with a CCS device programmer, will prompt the user for a serial number each time a chip is programmed. That serial number will be programmed into flash and can be accessed like any other constant in the program.

Linux Headaches

Recent versions of the Linux 32 bit LIBC library running on 64 bit machines have caused our compiler to crash with a segmentation fault. The problem started with version 2.28 of the library. You can find your version under Linux with this command:

```
ldd  --version
```

Users running on a 64 bit machine who keep their Linux kernel up to date have noticed the problem. The root of the problem is when the compiler launcher (ccsc) loads the compiler dynamic library (like pcm.so) memory is corrupted. We think, but can not be sure, the newer versions of the library loader assume the code segments are aligned on 64 bit boundaries even if the library is 32 bit.

CCS has been working on a number of fronts to find a solution that will work on everyone's system. The only success we have had is to compile, the compiler to generate 64 bit code. Since this changes all our internal data types we have needed to undertake a lot of testing to ensure the compiler is sound.

The new Linux software will be available before Christmas.

This means that there will be 32 bit and 64 bit downloads for the Linux compilers. The 32 bit will work on all 32 bit systems and the older 64 bit systems. The 64 bit version will work on all 64 bit systems.

Any Linux download rights that expired in 2020 have had a year of download rights added to their account. Contact support if you need new registration files.

Embedded Ethernet Development Kit

Kit Includes:

Embedded Ethernet Prototyping Board

In-Circuit Debugger/ Programmer

Exercise Tutorial

9V AC Adapter and Cables

PIC® MCU is a registered trademark of Microchip Technology Inc.

Supports Development with Microchip's ENC28J60 Transceiver

Light Sensors to Calculate Speed of a Toy Car

This project utilizes a Hot Wheels Set, phototransistor sensor (L14F2) and a PIC16F15354 device in order to display the speed of a toy car in units of inches per second on an LED display.

Materials for the Project

Begin a toy car and a track, such as a Hot Wheels set. Add the sensors into the track by drilling holes 4 inches apart and about 3/16ths of an inch in diameter, so the sensor fit through. It is important that these are as close to four inches apart as possible to ensure that the speeds that are determined are accurate. It is recommended the sensors have wires connected to the E and C pins, as this allows for one to bring much of the wiring away from the track, and allows for the sensors to be glued into place more easily without having to worry about a close connection to a board. Hot glue works best to mount the sensors in the holes. Make sure that the sensors are glued in low enough so that the movement of the car is not hindered by the car hitting the light sensors during its trip.

Take note that light sensors at the top of a loop may not work well if the car slightly falls off the track because it may not register the change in light.

In addition to the above materials, an in-circuit programmer unit, power source, and A to B USB and a proper connector for the device to the programmer is necessary for programming the development board.

Hardware Design

The development board has a PIC16F15354, four 7-segment LEDs to display the speed of the car after the car has passed over the two light sensors. The microcontroller has a built in timer and has six pins (B0-B5) available for the programmer, Extra pins are available to add features to the program.

The example programs only include the ability to make numbers appear on the LED display, but by adding additional entries in SEVENSEG, it would allow one to display letters as well.

The example programs only include the ability to make numbers appear on the LED display, but by adding additional entries in SEVENSEG, it would allow one to display letters as well.

The example program have the sensors to have their input return 1 (true) until they are covered up so that they are no longer exposed to light. At this point the sensor will then return 0 (false). The schematic for this sensor is illustrated below:

Program Example Code

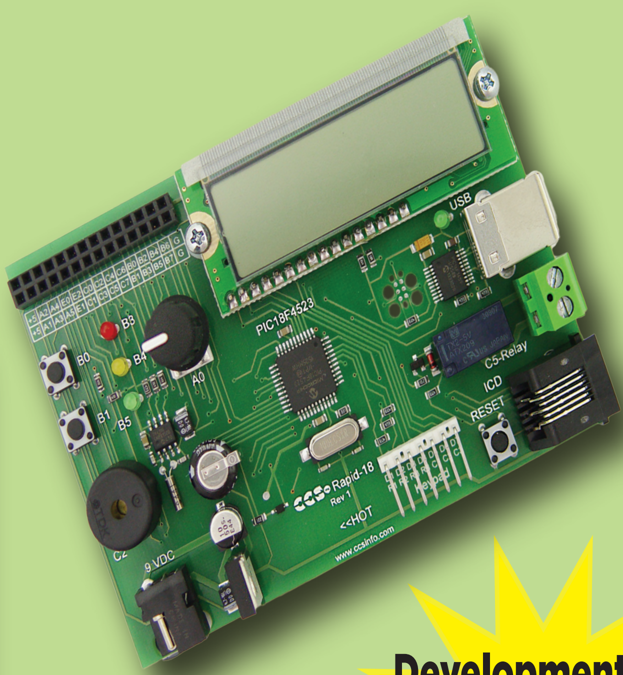
```
    #use timer(TIMER=2, ISR, TICK=1ms, BITS=32)


  do {
     speed = 0;
     if (!input(PIN_B1)) {
        set_ticks(0);
        while(input(PIN_B3));
        time = get_ticks();
        travelTime = (float32) time * 3 / 1000;
        speed = 12 / travelTime;
        delay_ms(250);
     } else if (!input(PIN_B3)) {
        set_ticks(0);
        while(input(PIN_B1));
        time = get_ticks();
        travelTime = (float32) time * 3 / 1000;
        speed = 12 / travelTime;
        delay_ms(250);
     }
  } while (speed == 0);
```

This is the code that is utilized to calculate the speed based on feedback from the light sensors. The "#use timer" preprocessor directive will set up the timer that is used to calculate the speed. In order to calculate the speed for the race car, the speed is initially set to 0 at the beginning of the loop to ensure that the speed is reset. Next, it checks to see if there is input coming from either sensor. Once there is a change in the input from one of these pins, the timer will start and will wait until the second pin detects an object running over it. At that point,the variable time will be set to the value returned by get_ticks(), which indicates the time in milliseconds that it took for the car to travel from one sensor to another. Since the light sensors are spaced four inches apart, the time to travel a full foot could be calculated by multiplying this time by three, and dividing by 1000 to convert milliseconds to seconds. This is stored as a variable called travelTime. Finally, to convert from the time that it takes to travel one foot to speed in inches per second, one can take 12 divided by the variable travelTime. The speed will then be displayed on the LED display in units of inches per second. This code can display the speed no matter the direction that the car is traveling.

Compile the program and connect the ICD unit to the development board and power supply. In the Compiler ribbon, select Compile -> Build & Run to program the board. Each of the four LEDs should display a “7” since that is what the default value at the start of the program was set to. Once the sensors are tripped for the first, the LED display should change to present the speed of the car. The speed should continue to be displayed until the power is disconnected, or until both of the sensors do not receive input / do not detect light. Sensors are in electronic products that people use daily. This demonstration would be a great way to introduce to students how sensors work and embedded programming in a fun way.

RAPID-18 DEVELOPMENT BOARD



FEATURES:

*PIC18F4523 Microchip PIC® MCU
*24 I/O Pins (11 can be 12-bit analog)
*One Potentiometer
*Two Pushbuttons
*Three LEDs
*USB Connector (with USB to PIC® UART interface)
*ICD Jack
*Keypad Socket (for 3x4 keypad)
*16x2 Character LCD
*Piezo Speaker/Buzzer
*Relay and Dry Contacts
*Real Time Clock/Calendar with Supercap

Development Kits Starting at $50.00

PIC® MCU is a registered trademark of Microchip Technology Inc

Bundle up and get back into coding projects with a C Compiler

$25 Off
a full Compiler
or Compiler Maintenance

USE CODE:
Winter20

COVID-19 RESPONSE

During this time of global uncertainty and change, we want to assure you that we are taking every precaution to ensure that we can safely support our customers during this time.

Despite these challenges, CCS staff is continuing to provide technical support, as well as processing orders. It is essential customers have the tools they need to provide the development of existing or new products that may be necessary in the fight of Covid-19.

Many of our existing customers are having to work from home and we want to remind everyone of our Software Licensing Agreement. We pre-register all compilers in a user's name. You can install your compiler on your home PC and laptops. If you do not have access to the registration files and installer, contact customer service for assistance.

Most importantly, as we work together in this unique and rapidly changing environment, we do so with confidence that we will overcome this challenge. Until then, we hold our enduring commitment to the health and well-being of our employees and customers.

Please let us know how we can help you. Stay healthy.

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!

You Tube