

A STARTER'S TUTORIAL FOR THE MSP430 ULTRA-LOW-POWER MCU EZ430-F2013 DEVELOPMENT TOOL

JONATHAN KIRWAN

Dedicated to all who continually struggle against conflation and ignorance and work diligently for knowledge that frees themselves and others from the limits into which they were born.

ABSTRACT. This is a series of small programming projects designed around the inexpensive and handy MSP430 Ultra-Low-Power MCU **eZ430-F2013** Development Tool. Each successive project builds concentricly upon earlier ones and is arranged to teach only one or two new concepts at a time. These lessons continue into much more than programming alone and will deal with many other things, such as human visual perceptions.

1. PROJECT INTRODUCTION

This project sequence is intended more as an introduction to embedded programming than as an introduction to Texas Instruments' microcontrollers. These lessons use the **eZ430-F2013** kit primarily because it is cheap and very easy to start using well.

Although these projects aren't designed to teach the **c** or **c++** languages, they do express ideas plainly and clearly so that a reader who already knows the basic elements should be able to make their own way. The projects assume familiarity with loops and conditional statements, as well as how they are written in the **c** or **c++** programming languages. Familiarity with integers and binary and hexadecimal notation is also assumed. The emphasis is to start learning microcontroller features and embedded programming right away, without having to spend lots of money.

Freshman computer science classes will often introduce students to **c** or **c++** (although **java** is avant-garde today) and to a fictitious assembly language made up for the purpose by some instructor. With the caveat that a real instruction set replaces a possibly fictitious one, these projects may even be used to help supplement some of the lecture sections. The **eZ430-F2013** kit includes both a well-featured simulator and FET hardware emulator, which may be used at the assembly level or at the **c** or **c++** language level.

Hopefully, there's a little something for everyone to enjoy.

Date: November 23, 2009.

Copyright © Jonathan Kirwan, 2009. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts and no Back-Cover Texts. The web site, <http://www.fsf.org/licensing/licenses/fdl-1.3.html>, currently has a copy of that license. It is not included within this document.

2. EZ430-F2013 OVERVIEW

The **eZ430-F2013** kit includes a USB stick that is small and light enough to carry in a pocket without hardly noticing it, while also reasonably safe against physical abuse. The retail price is about \$20, though during special promotions the price can be as little as half of that. (Look for special deals on April 30th of each year and sometimes running afterwards for as long as a week.)

Most of the included development software required in these lessons is from IAR Systems, located in Uppsala, Sweden, and is free to use so long as the programs use assembly or otherwise limit their `c` or `c++` language code to about four thousand bytes in size. That's enough to get through these lessons without difficulty. There are options for those interested in writing larger programs, including a distribution or two of the infamous GNU GCC compiler. Commercial compilers are also available from a handful of excellent vendors and will vary in price from roughly a few hundred dollars to perhaps more than a few thousand dollars. The point here is that the MSP430 is well supported by a spectrum of well-placed options to fit most needs and that the included development software is already well-suited to these lessons. For now, don't worry about it. Just use what comes with the kit.

There are other very good microcontroller choices for learning embedded programming, as well as quite a variety of pre-built board products that include a bewildering array of options and choices. Some include text and LCD displays, with backlighting, dozens of keys and LEDs, adjustable voltages, USB connections for learning how to do USB, and pretty much anything else imaginable. And the costs are often very reasonable, as well. But too many options can flummox many first starting out on embedded programming.

Where the **eZ430-F2013** excels is as a cheap, tiny, durable, and convenient USB device with a decent IDE and development toolset. The entire system of which can be attained for as little as \$20 (plus some shipping, perhaps.) It doesn't include a lot of built-in features, but it is a great place to start learning about embedded programming. And it is in that sense these lessons were also written.

3. GETTING FAMILIAR WITH THE USB STICK ITSELF

The **eZ430-F2013** development kit includes a box with basically two items: a *USB stick* with a pair of electronic circuit boards bound inside a plastic case and a USB connector at one end; and a *CD-ROM* that provides the necessary Windows drivers as well as the software development tools from IAR Systems used to write and test programs. The development tools can be used to edit, compile, download and debug programs into an MSP430F20x3 microcontroller sitting on one of the two tiny boards in the USB stick.

The USB stick that comes in the **eZ430-F2013** development kit can be opened to gain access to a pair of boards inside. The larger board in the USB stick mediates between the IBM PC software. It is there to help the development software residing on the IBM PC communicate properly and otherwise operate the tiny target board, which is the smaller one. Lessons (and projects) developed within the IDE are instead downloaded via that larger board to be placed into and then allowed to run on the small MSP430F2013 placed on that smaller (very tiny) target board. Just remember that these lessons will be downloaded into that smaller target board, not the larger one. That larger board is there only as an intermediary.

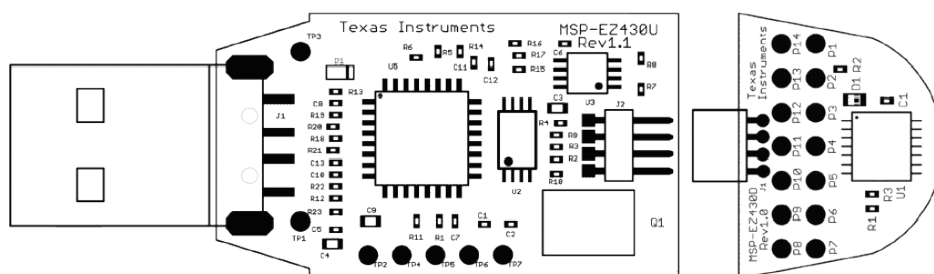


FIGURE 1. Pair of boards found in the eZ430-F2013 USB stick

The small target board in the eZ430-F2013 kit is based on the MSP430F2013 microcontroller. Although Texas Instruments hasn't yet seen fit to separately sell daughter boards using the MSP430F2013, they do sell some daughter boards based upon the MSP430F2012 microcontroller that can be purchased at a retail price of three boards for \$10. These can be easily attached to the eZ430-F2013 USB stick and used for special projects at very low cost and effort. (Of course, shipping usually isn't free.)

The plastic case of the USB stick can be easily opened. But take some care about it. The clear top half is connected to the clear lower half via four tiny plastic pins. Using a tiny flat-bladed screwdriver or penknife blade will help ease the two pieces apart. Fingernails can be used, as well, but carefully as it is pretty easy to overpower the pins and break them. Don't completely open one corner before moving to the next. Instead, loosen each corner a little bit at a time. Proceed around the plastic case, gradually separating the pieces. Soon, the two plastic halves will separate smoothly. Once that is done, the USB connector and two boards can be gently lifted out as a unit.

- Don't do this right away! This instruction is only included to provide a heads-up about possibilities after completing the lessons. For now, just read and try out the lessons.

There is a tiny black connector that separates the larger board from the smaller one. A gentle pull on the smaller board while holding onto the USB connector itself will remove the tiny MSP430F2013 target board. There are some through-holes on this tiny target board that can be used for soldering, if desired.

Fig. 1 is an illustration of what's inside the plastic case. The tiny circuit card on the right side is the target board that holds the MSP430F2013 microcontroller to be programmed.

As already mentioned earlier, replacements for this tiny target board can be purchased at three of them for \$10, but they will not have the MSP430F2013 — instead they include the MSP430F2012. There is a difference. So above all else, don't lose or break the one that comes in the kit. To replace it, another \$20 system has to be purchased. So if there is any chance of something happening to it, buy the extras for \$10 and use those, instead. For these projects, either the original or the replacement boards will work just fine.

The small target board also includes two rows of holes. A total of 14 holes, altogether. These holes provide access to the input/output pins of the MSP430F2013 processor. Wires can be soldered into these holes and brought out to some other board or device. But that is for later use in other lessons beyond this set, when switches and debouncing is addressed. There is also a tiny, green LED on this target board. It can be barely noticed just above the left side corner of the microcontroller, labeled as D1 and shown on the tiny target board in fig. 1 near the two rows of holes also shown there. The fact that the LED is already included makes it easy to get started right away with some important lessons without having to worry about soldering wires. There is always time for that, later.

There are some other holes on the larger board. These won't be used in these lessons, but it may be helpful to know what they are. See table 1.

TABLE 1. Test Points

TP1	Vcc
TP2	RESET
TP3	GND
TP4	TCK
TP5	TMS
TP6	TDI
TP7	TDO

Just note their presence and forget about them, for now.

4. INCLUDED DEVELOPMENT SOFTWARE

The CD-ROM includes the Windows drivers. Follow the instructions for installation.

The CD-ROM also includes a very useful but restrictive version of IAR Systems' full `c/c++` compiler. The restricted tool is called the *IAR Embedded Workbench Kickstart IDE*. This programming development environment includes automated program building, flexible workspaces, their `c/c++` compiler and assembler tools, a text editor, and debugging support to name the more important parts. But it is limited to developing smaller problems, those no more than about 4000 bytes in size. Even so, the *IAR Embedded Workbench Kickstart IDE* is fine and probably is the best one to start learning to use because it is so conveniently provided in the kit — unless it's already known that another one will be used for development later on.

Only one USB stick can be in use at a time and if two or more of the USB sticks are used in sequential order on the same Windows computer, a prompt from the "Found New Hardware" dialog occurs. If the drivers were installed the first time, don't search around — tell it NO — and make it use what drivers are already present. Otherwise, just avoid the problem by using only one particular USB stick on a given computer.

For these lessons, install the device drivers and install the IAR Systems IDE software. (Do that *before* inserting the USB stick into a USB port on the IBM PC computer!) The toolset uses a USB port of an IBM PC and requires the drivers to make use of the USB stick. The IAR IDE software allows the rest that is needed

to get started with embedded programming and it will use the Windows driver to help do that.

It's possible (and likely if enough time has gone by) that a more recent Windows driver or a more recent version of IAR's IDE becomes available. The Windows driver doesn't seem to need changing much and I can't recall ever changing it from the one that comes on the CD-ROM. But it's a good idea to check on either Texas Instruments or else IAR System's own web site for a more recent version of the compiler tools and IDE. It may include support for newer MSP430 microcontrollers.

One place to check is on the Texas Instruments website, though the exact location isn't always in the same place. A starting point, though, is at:

<http://www.ti.com/msp430>

Look for links related to supporting software. Another place is to go to IAR's website at:

<http://supp.iar.com/Download/SW/?item=EW430-KS4>

The newer version can be retrieved from there (after giving away some personal information, first.)

Use whichever web site is found more comfortable.

Regardless of approach, install the software. Once complete, the remaining focus can just be on the USB stick and programming it.

With the software installed and ready, plug the USB stick into a USB port on the IBM PC. If it is fresh out of the box and plugged into the USB port for the first time, the green LED on the USB stick will blink. It does this because some software has been preloaded to do that. As soon as power from the USB port is accessed, the microcontroller begins running that software. There may also be a message or two from the operating system, to notify that the drivers are being set up correctly when the USB stick is first seen. Also, each time it is inserted or removed, Windows often likes to do a special kind of *beep* sound to announce that fact.

The next important step is getting through the details required to actually start writing, compiling, downloading programs into the USB stick, and then running and debugging them. There is quite a lot yet to do.

5. SHORT NOTE ABOUT THE IAR IDE

The IAR IDE is more fully known as the *IAR Embedded Workbench IDE for MSP430*. And the free version and the combined IDE and compiler tools is the so-called *Kickstart* version of it.

When talking to others and asking questions, it's often helpful to let them know what's being used. (I've never paid for the full version, so I can't discuss fine details about it.) Sometimes, it's even important to communicate exact versions of specific programs. To get that information, start up the IAR IDE and select the *Help* item from the menu bar. Then select the *About/Product Info...* item from the drop-down menu that appears. A lot of detailed information will be displayed and can be selected and highlighted by using the mouse and then copied using the *ctrl-c* key, for pasting into some email message later.

As I mentioned before, there are two separate sources for the *Kickstart* edition of the IAR tools — IAR Systems and Texas Instruments. IAR Systems probably has the newer editions that may be available at any point in time, but they require

some personal information to get them. Texas Instruments probably updates their copies less often, but I've been able to secure copies from them without having to give away information about who I am. Choose your poison. (I keep all installation packages I download and never throw anything away. It's always possible that something gets worse, not better, in later editions.)

6. USING THE IAR IDE TOGETHER WITH THE LESSONS

The installation includes extensive documentation and I'd be a fool to try and replicate any significant part of it. Look for these documents in the installation directory and under the subdirectory 430\doc\. It's important to know about these. The first two shown in table 2 are the most important.

TABLE 2. IAR Documentation Sets

EW430_UserGuide.pdf	IAR Embedded Workbench IDE for MSP430 User Guide
EW430_CompilerReference.pdf	MSP430 IAR C/C++ Compiler Reference Guide
EW430_AssemblerReference.pdf	MSP430 IAR Assembler Reference Guide
xlink.ENU.pdf	IAR Linker and Library Tools Reference Guide

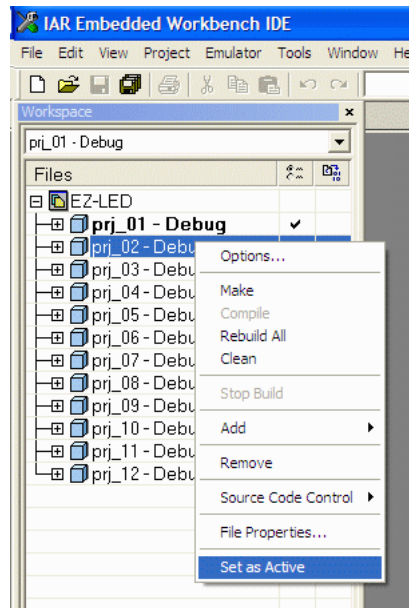


FIGURE 2. Partial view showing how to make a project active

The IAR Workbench IDE allows more than one project within what they call a *workspace*. I have used that capability to put all the projects in one convenient workspace. But this creates a problem for the IAR Workbench IDE. Which one is the current one the IDE uses? The workbench IDE needs to be told. That's done by right-clicking the mouse over the project name.

Fig. 2 illustrates what happens if this step is done right.

Note that the last item is highlighted and says *Set as Active*. When selecting that option, the project name will appear in bold text. In fact, fig. 2 shows *prj_01* in bold text because it is the active project. However, I've selected *prj_02* with the mouse, intending to make it the active project. When that happens, *prj_01* will cease to be indicated in bold text and *prj_02* will then appear with bold text, instead.

That's how the IAR Workbench IDE decides which of several projects within a workspace to use when compiling, downloading, and debugging code. Keep track of this. It can be *very* confusing to have one project marked as the active one while working on a different one. The IDE will happily compile and download the wrong code, with confusion resulting.

Make sure that the project marked as active is the right one!

The IAR Workbench supports either a simulator or a FET debugger, when testing developed code. The difference is immense. The simulator doesn't even use the USB stick! It just simulates the code without it. The FET debugger, on the other hand, will download code into the physical target device and debug it directly there. This is the difference between night and day. Under *no* circumstances use the simulator.

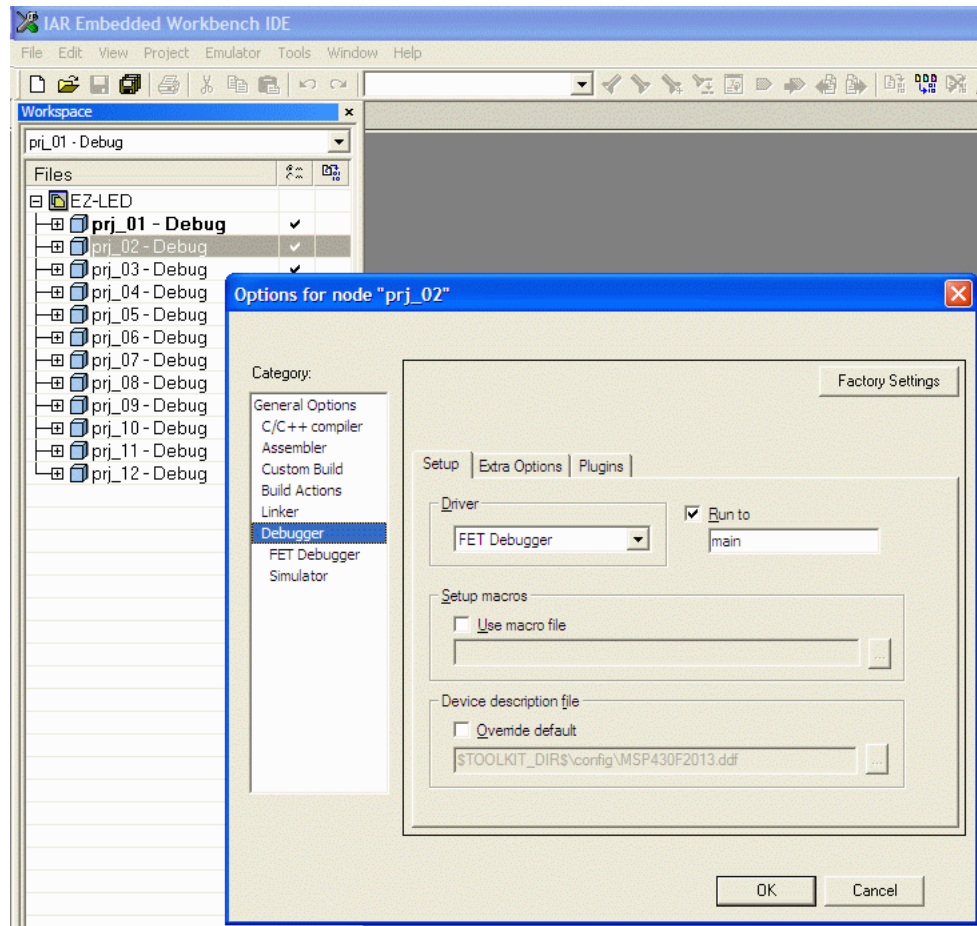
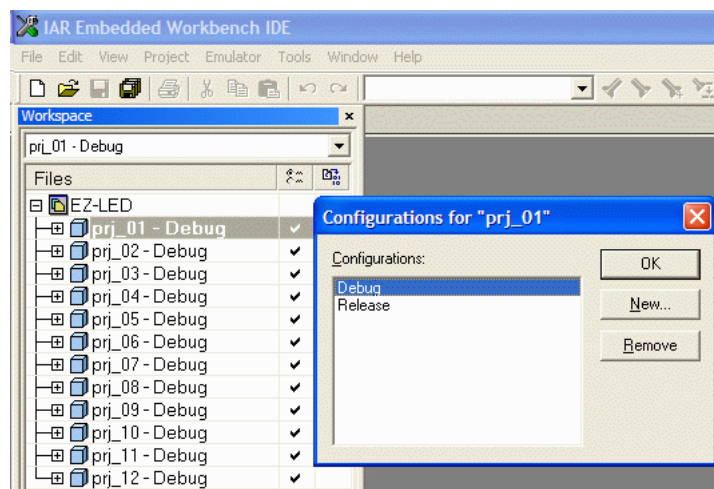


FIGURE 3. Verify that the FET debugger is selected

Refer back to fig. 2 and notice the first available option, oddly enough called *Options....*. Select that and the dialog box illustrated in fig. 3 pops up. Use the mouse to select *Debugger* on the panel on the left, make sure that the *Setup* tab is front and center, and observe the name in the drop-down box titled *Driver*. It should read *FET Debugger*. If it doesn't, change it and save the change.

In some cases, I'll ask that the project be switched from *Debug* to *Release*. Both fig. 2 and fig. 3 show all of the projects set to *Debug*. To change to *Release* simply ensure that the project name is highlighted by selecting it with the mouse, clicking

FIGURE 4. Choosing either *Debug* or *Release* modes

once with the left mouse button when the mouse is over the project name, and then use the menu option called *Project* and select the *Edit Configurations...* choice. A small dialog box will appear. Click on *Release* and select the OK button. Fig. 4 provides an example of this dialog box.

There are a couple of other complexities worth mentioning that otherwise might be a bit difficult to track down for those not entirely familiar with the way the IAR Workbench IDE does its job in communicating with the USB stick included in the eZ430-F2013 kit. One is making absolutely sure that the *right* FET debugger is selected. Take a look at fig.5 to see the correct selection.

There's still more to getting it all working well. Take a close look at fig.6 on the following page. It's possible that the selection of *Other* has been made with the linker's output. So make sure that the format is for C-SPY, as indicated there in fig.6.

As always, there's always more to learn. It's a never-ending story. One web address that provides some interesting and helpful information about the IAR Workbench software is located at IAR Systems own web site at:

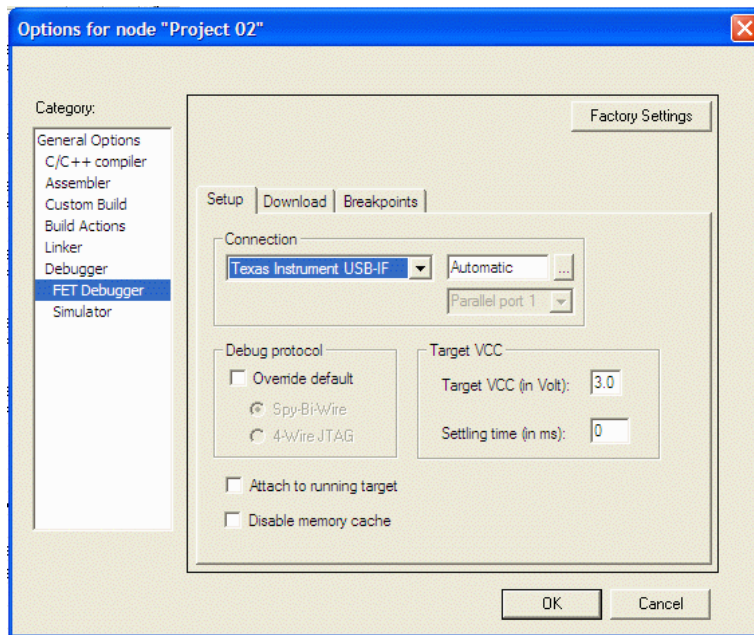
<http://www.iar.com/website1/1.0.1.0/473/1/>

The site looks a bit sparse at first glance, but some of the links are to pages with many more links on them. Many are worth serious reading time. Have a look.

Well, this provides snapshots of some important details. Read the manuals, scan through some of the web pages at IAR Systems, and otherwise just get familiar with the IAR Workbench. And move on to the projects ahead.

7. THE SERIES OF PROJECTS

Each project uses a single file (or *module*, for those who conflate the two concepts.) The beginning of each file contains important instructions to follow. Start with the first project, read the text in it, follow the steps indicated, and download and run the compiled program using the FET debugger. Then proceed to the next project in sequence.

FIGURE 5. Choosing the *right* FET debugger tool

These projects build upon concepts developed in earlier ones, proceeding from merely controlling an I/O line to using timer counters and interrupt handling to cause the LED to gradually brighten and dim.

I include almost all of the necessary documentation within the source code of each file. I chose this method instead of providing documentation here because it is all too easy for files to become separated and I'd like at least some possible utility to remain if that happens. Also, this way strongly encourages me to update the documentation in the same file I am busy modifying rather than putting it off until I can get to it, here.

Worth mentioning again is that these projects aren't designed to teach the `c` or `c++` language and I assume the reader knows the basic elements already or has access to a teacher or friend who can help when needed.

I delay the introduction of many concepts as long as possible, so that I can take on new issues in small steps. For example, I delay the introduction of interrupt handling because it requires new, hard to acquire mental concepts for some people — especially those more used to programming on workstation operating systems where a lot of this is handled for the programmer by libraries or the operating system, itself — and because it also requires quasi-standard or simply non-standard `c` code syntax that is often different for each compiler and needs to be learned. Those who are already very familiar with concepts introduced in earlier projects should feel free to skip over some (or all) of the lessons. I wanted to include a more gentle approach that would provide a concrete foundation before proceeding into new territory.

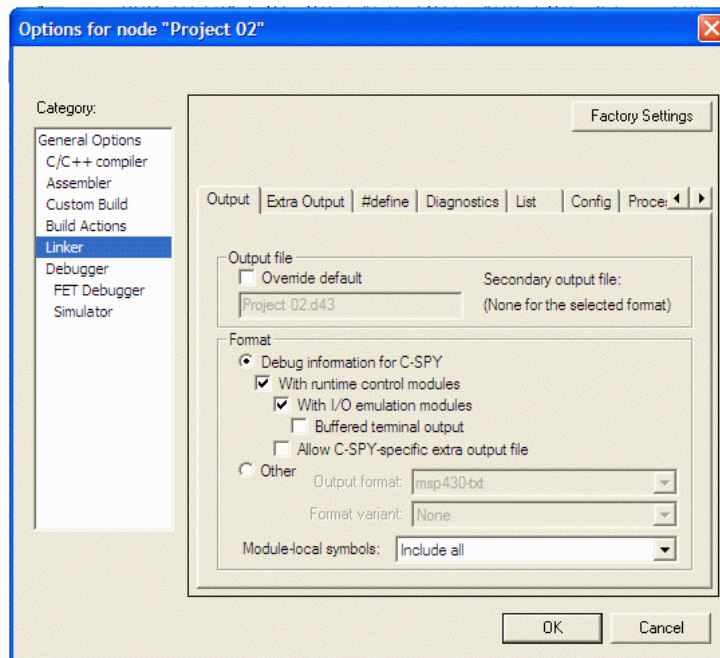


FIGURE 6. Selecting C-SPY linker output

These lessons continue into much more than programming alone and deal with such things as human visual perceptions. Hopefully, there's a little something for everyone to enjoy.

In these projects, I will sometimes place long discussions in the code at places other than at the top of the file. These discussions will be highlighted by,

*** READ THIS ***

which calls attention to them. It's wiser to pay particular attention to these sections and read the text there. In later projects, I often reduce these comments to much more terse ones, so that their bulk doesn't continually distract from new discussions being added. So check back on earlier projects, if needed. So if a few lines seem inadequately explained in a later file, take a look at some of the earlier ones. It's possible that a fuller explanation was already provided.

I expect readers to do their own work in securing appropriate documentation files from the two main sites I've already mentioned and finding and reading interesting sections. However, I'll include here some ancillary information on some of the projects, where inclusion in the text of the module itself would be impossible or where the information may otherwise be somewhat harder to come by than I have a right to expect from a reader.

And as I said, hopefully some fun will be had from these.

8. PROJECT NOTES

Project 1 There is a fair-sized hurdle to get over, just getting the first project running — for those who haven't tried using the IAR Workbench together with the rest of the eZ430-F2013 development kit. Drivers are needed so that the IAR software can

talk to the USB stick, the IAR software needs to be installed, a handy USB port needs to be found on the computer, the workspace I'm including needs to be placed on disk and loaded up correctly in the IAR workbench. Given that, I felt it best not to start right away getting deeper into some coding with the first project.

This document is included to help get past some of the problems, too, that may plague a newcomer. The focus of the first project is primarily to allow time and space to get onto the right track and running forward.

This project is very simple — make the LED turn on and stay on. That's it. However, there is a compiler error built into the source code that must be found and corrected. (Okay, I have commented out an example of a working line of code.)

The project's source code also includes a lot of writing that covers a wide range of subjects. Skip it, if it is a waste of reading time. For some, the context may help.

- Project 2** The second project assumes that the details of using the workspace, editing code, and getting that code loaded into the USB stick are successfully understood and takes the first step towards writing non-trivial code for the LED. In this case, the idea is to get the LED to flash at a visible rate somewhere between five times a second and once every five seconds.
- Project 3** This project takes a short detour and builds upon the last project to learn a little about compiler optimization and the *volatile* keyword of `c` and `c++`. It is still essentially just a blinking-LED program, though.
- Project 4** This project discusses the impact of varying cpu instruction rates on the *busy-wait* loops used in earlier projects and shows how to use calibrated instruction rates for processors like the MSP430F2013, which includes the capability.
- Project 5** This project expands upon the prior project, using calibrated clock rates, and takes the first step towards using the included Timer A2 hardware for doing some of the counting effort. It uses a *busy-wait* loop on the timer flags and does not use interrupts, yet.
- Project 6** This project introduces the idea of interrupt functions (using language extensions to `c` and `c++`) to the prior project. Most of the other details are the same, except that a hardware interrupt is enabled and the source code includes a function designed to respond to the interrupt event.
- Project 7** This project expands the interrupt function in order to completely off-load the work of blinking the LED, from the main code and into the interrupt code. This frees the main code to do other tasks.
Also, this project includes a short feature about using low-power mode in the main code instead of a *forever loop*.
- Project 8** In earlier projects blinking the LED, the on-time was 50% and the off-time was also 50%. In other words, a 50% duty cycle was used. This project introduces the idea of different duty cycles and encourages some exploration by observing the

LED, driven differently. It makes it easy to try different ratios to see how the LED appears.

Project 9 This project, for the first time in the series, increases the blinking rate of the LED to the point where the blinking cannot be seen, anymore. The critical flicker fusion frequency (CFF) is introduced and different duty cycles are experimented with. The effects upon the CFF, when moving the LED around quickly, is also explored.

Project 10 This project starts to use another compare register in the Timer A2 module to replace a software counter that was used in project 9. It also greatly expands on the use of pre-processor commands to isolate interesting parameters and place them in a convenient place in the code (near the top.) The interrupt code is greatly simplified, as a result.

This project also introduces the Talbot-Plateau and Weber-Fechner laws, regarding human vision.

Project 11 This project takes on a tougher task. The duty cycles were ‘*baked in*’ at compile-time in earlier projects. To change the duty cycle, it required a change to the code and re-compilation and downloading. In this project, software is added to automatically ramp the duty cycle up and down to vary the apparent LED brightness.

This project starts out solving this task by first placing new code to adjust the duty cycle into the main function code. The earliest version doesn’t work quite right, but is soon corrected. By the end of the project, this new code has been moved back into the interrupt code in order to free up the main function code, once again.

Project 12 The prior project left something to be desired, because it didn’t deal with the Weber-Fechner law in creating what would have been a smooth change in LED brightness. This project takes this law into account and develops a method to yield what appears to be a smoother change, up and down, in brightness. All of the code remains in the interrupt routine, leaving the main code to do other work.

This is the final project in this short series.

9. CONCLUSION

This series is non-invasive and requires no soldering. And this is a good stopping point. But much remains worth exploring, even operating a single LED in an embedded project. (And has already been trimmed and left on the cutting room floor of this series, too.)

With a willingness to modify the tiny board, an entire world opens.

Not today, not here, though.

(Jonathan Kirwan) NEW WORLD COMPUTING SERVICES
EMBEDDED SOFTWARE ENGINEER, HOBBYIST, AND OWNER
Current address: Hamlet of Boring, Oregon, USA
E-mail address: jonk@infinitefactors.org