Developing LPCXpresso Applications with CooCox

This document shows how to put together a truly open-source software development system for the LPCXpresso LPC1769 evaluation board. Using the free CooCox Eclipse-based IDE (**I**ntegrated **D**evelopment **E**nvironment), the inexpensive CooCox CoLinkEx debugger module ($28), and the LPCXpresso board ($30), a low cost editor, compiler, linker, flash programmer, integrated GDB debugger, and target board can be realized for $65 in parts and still be capable of supporting other manufacturer's ARM-Cortex microprocessors with no limits whatsoever on program size, etc.

The LPC1769 LPCXpresso Board

There's been a lot of interest lately in the LPCXpresso board ($29.95), retailed by Adafruit, Element14, Digikey, and others. This is a LPC1769 ARM-Cortex evaluation board that includes a debugger circuit. Figure 1 shows this board (actually the very similar LPC1768 version) and you can see that the left-hand part is a USB-based debugger called LPC-Link while the right-hand part is a LPC1769 target board.

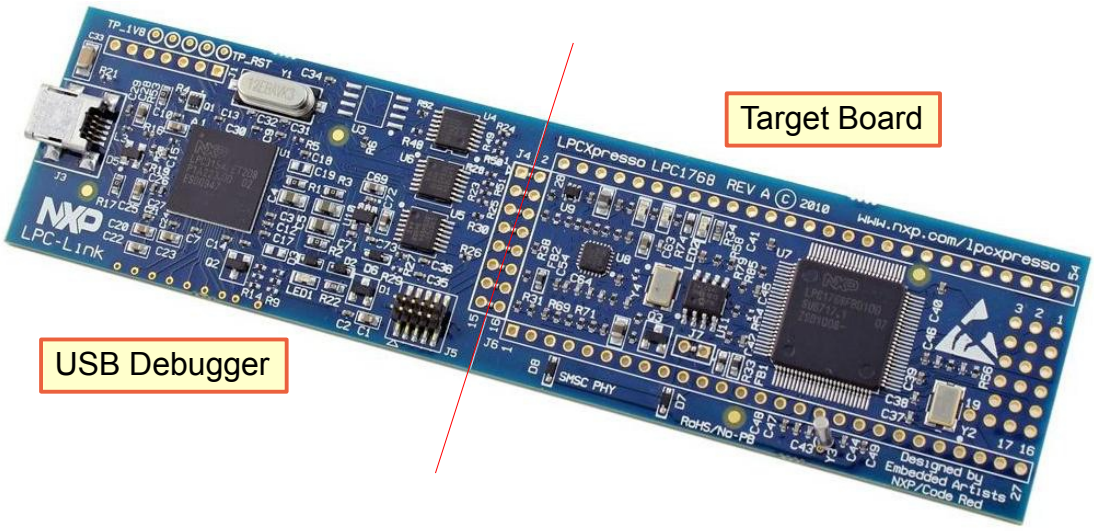


Figure 1: LPCXpresso Board - $29.95 from Adafruit, Digikey, and others.

This LPC1769 evaluation board was designed by Embedded Artists from Malmo, Sweden. They also designed the Mbed board which is quite popular (it has a cloud-based development system).

Wedded to the LPCXpresso board is the Code Red LPCXpresso IDE (**I**ntegrated **D**evelopment **E**nvironment), a complete software development system for this board. The download is free, easy to install, and includes a GNU tool chain, Eclipse IDE platform, and the LPC-Link driver to connect to the on-board debugger. The LPCXpresso IDE was developed by Code Red Technologies from Cambridge, England and San Francisco, California.

This all sounds very good and it is. I downloaded the Code Red LPCXpresso IDE, installed it with very little effort, and within minutes had it single-stepping through C code that I had developed. Wow, this thing is free, fairly easy to use – it's just too good to be true, Right?

Like all "*free*" Eclipse/GNU commercial tool chains released recently, this one has limitations. Namely a limit of 128k for the executable (the LPC1769 includes 512k of flash memory).

It also won't support any other manufacturer's ARM-CORTEX chips and does not support C++ language. Figure 2 shows a screen shot of their own table showing the capabilities of this Code Red LPCXpresso IDE versus their other more costly offerings.

Feature	LPCXpresso	Precision32 IDE	Red Suite NXP Edition	Red Suite Full License
Price (Perpetual license)	Free	Free	$256 (256k) $512 (512k)	$999 [2]
Evaluations	No (free product)	No (free product)	No (Use Red Suite Full to evaluate)	Yes 60 day Non-expiring 128k limit when used with Freescale FRDM boards
IDE Features				
C/C++ Support	C	C/C++	C/C++	C/C++
Support for 3rd party plugins	Yes	Yes	Yes	Yes
Red State	Yes (NXP SCT only)	No	Yes	Yes
Memory configuration editor and External Flash support	Yes	No	Yes	Yes
Debug features				
Download Limit	128k	256k	256k/512k	Unlimited
JTAG/SWD interfaces supported	Red Probe+, RDB-Link, Redlink, LPC-Link,	Silabs USB Debug Adapter, Red Probe+,	Red Probe+, Redlink, RDB-Link, LPC-Link,	Red Probe+, Redlink, RDB-Link, LPC-Link, Freescale FRDM TI Stellaris ICDI Silabs USB Debug Adapter, and selected Evaluation boards
Peripheral displays	Yes	Yes	Yes	Yes
Red Trace	No	No	Yes [1]	Yes [1]
Instruction Trace with ETB/MTB (when available on target MCU)	No	No	Yes [1]	Yes [1]
Standalone flash programming	Yes	Yes	Yes	Yes
Inclusive Support	Web-based forum	Via Silabs website	up to 90 days No support after 31 December 2013	Until 31 December 2013
Supported MCU Families	LPCXpresso	Precision32 IDE	Red Suite NXP 256k/512K	Red Suite Full License

Figure 2: Comparison of Code Red Eclipse\GNU-based Tool Chains

If you read the above chart, you're looking at $512 to go to a 512k executable or $999 to go to an unlimited executable size. Added to that, the $999 Red Suite Full License supports debuggers that are fairly expensive (e.g. the Red Probe+ is $150.00). To be fair, the Red Suite Full License does support just about every ARM-Cortex embedded microprocessor available for sale.

Starter Kit Software Development Packages are Teasers

Microprocessor manufacturers used to feel compelled to offer compilers, linkers, editors, and debuggers for their products. This side business required lots of staff and telephone support for users so these systems were very expensive. A few manufacturers now offer free, open-source development systems – Atmel has a nice Eclipse-based system called Atmel Studio 6 that is completely free with no restrictions, but it only works with Atmel microprocessor chips (ARM and AVR) and the Atmel debugger (J-Link).

Other chip manufacturers have outsourced software development systems to third parties, such as Code Red, Atollic, Keil, IAR, and others. The usual scenario is when a new chip set is developed, such as the NXP LPC1769, a starter evaluation board is outsourced to someone (in this case, Embedded Artists) and a

free software development system is outsourced to someone else (Code Red Technologies). The problem is that the starter IDE is usually a "limited" version of the standard software development package. The limitation is usually code size; you can't create a program bigger than 128k. When you need the extra space, you'll have to upgrade and this usually runs $1000 or more. I've seen some of these starter packages limit the number of breakpoints, not support floating point, and other ruses to entice you to buy the professional version.

Is a $999 Eclipse/GNU Development Platform Fair?

The answer is yes and no, depending on your point of view. The Open Source movement, pioneered by Richard Stallman, was intended to make software development in the digital age available to the masses, not just the elites who could afford thousands of dollars for the tools and operating systems. Open Source contributors, led by Richard Stallman and Linus Torvalds, developed the GNU tool chain, the Linux operating system, the Eclipse IDE, and a host of other software goodies. They did this without pay and offered it to anyone on Earth for free.

The idea for commercial use of open source software is that while you can't charge for the open source parts (the GNU tool chain or Eclipse, for example), you can charge for any added value. Code Red didn't develop Eclipse, but they did create a few plug-ins that make Eclipse quite a bit easier to use. They have a right to charge for these improvements.

I have used professionally a microprocessor software development package from Green Hills Software where they themselves developed in-house the compiler, IDE, and debugger software and hardware. They charged about $15000 for this and support ran about $2400 a year. So for commercial applications, I don't think any company would object to paying $999 for a complete software development system from Code Red Technologies (you get pretty good support at that price).

The problem is that the $999 price tag cuts out students, hobbyists, and entrepreneurs who wish to start a commercial development on a shoestring budget.

A Free Open Source ARM-Cortex Development System

There is a way to put together a free open source ARM-Cortex software development system by using the **GCC ARM Embedded** tool chain, the **CooCox IDE** and their little $28 **CoLinkEx** JTAG/SW debugger board.

The **GCC ARM Embedded** tool chain is a GNU-based compiler/linker for the ARM-Cortex microprocessors and is actually maintained by engineers from ARM, who update the tools package several times a year. They offer it as a free download and monitor a forum for those who have questions or bug reports.

CooCox is an unusual name, but it stands for **COO**perate on **CO**rte**X**. The CooCox open source embedded tools project was started in 2009 by Embest and Wuhan University in China. Two years ago, Embest and CooCox were purchased by Farnell and Newark/Element14. Their web site states "*We have been committed to providing free and open ARM Cortex M development tools to users over the years*". I take them at their word that these development tools will always be free and they will profit instead from added services and advertising.

The heart of the CooCox IDE is a version of Eclipse that they have stripped down and added new features for embedded programming. Figure 3 shows a typical CooCox display screen showing the editing window, the project manager window and so forth.

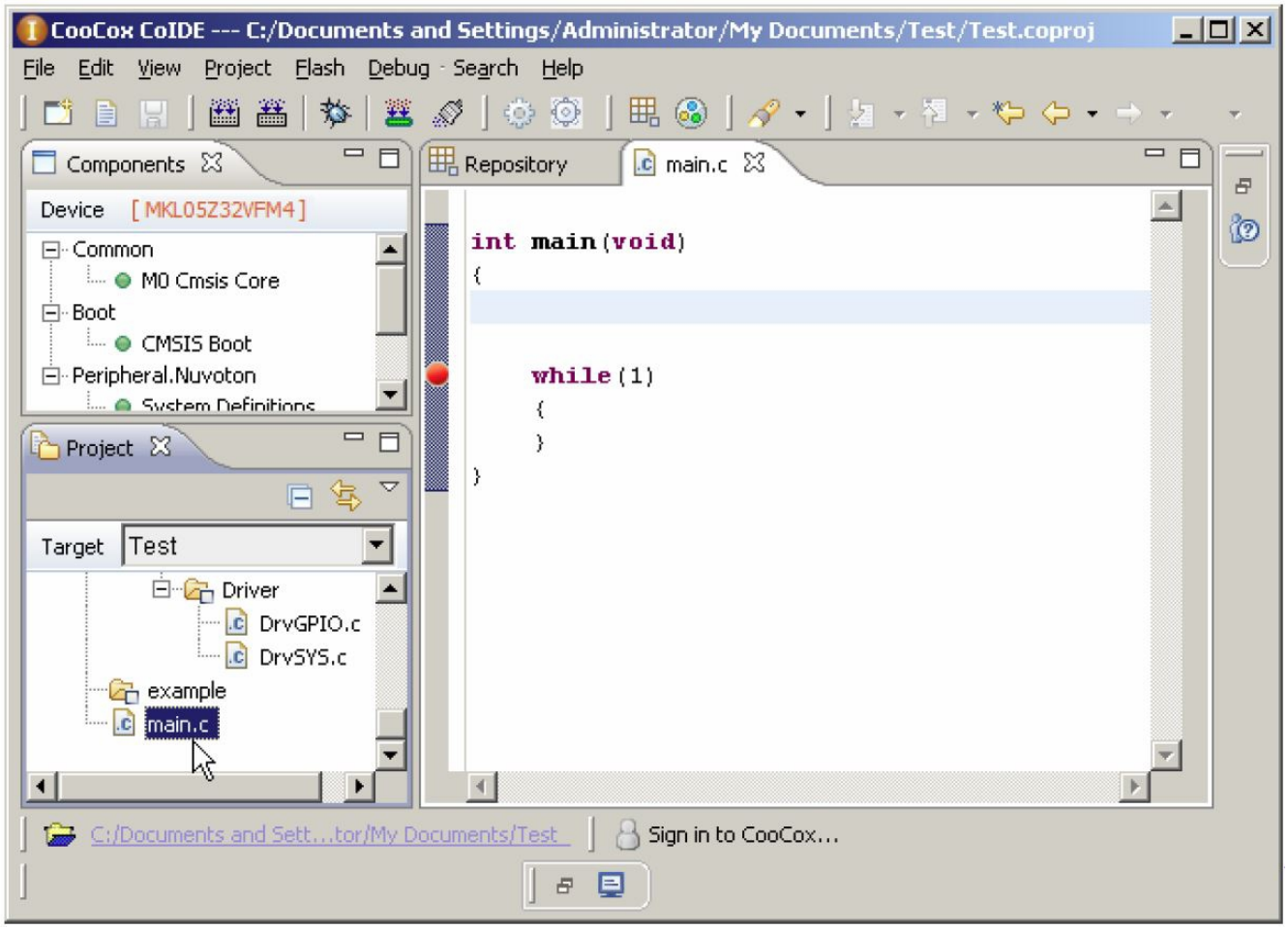


Figure 3: Typical CooCox Display

Experienced Eclipse users will immediately note that while Figure 3 is an Eclipse screen, the number of pull-down menus and action buttons have been greatly reduced. This is probably a good thing since the Arduino folks have led the way in keeping the development system as simple as possible. When you click within one of the sub-windows (called perspectives in Eclipse jargon), a “right_click” with the mouse will open up an extensive Eclipse context menu that has all the normal Eclipse activities such as “delete”, “rename”, “find declaration” and so on.

When you create a new project, you have to specify the chip that you will be using. In Figure 4, the NXP LPC1769 microprocessor chip is selected for the project. CooCox has most of the ARM-Cortex chips available for selection.

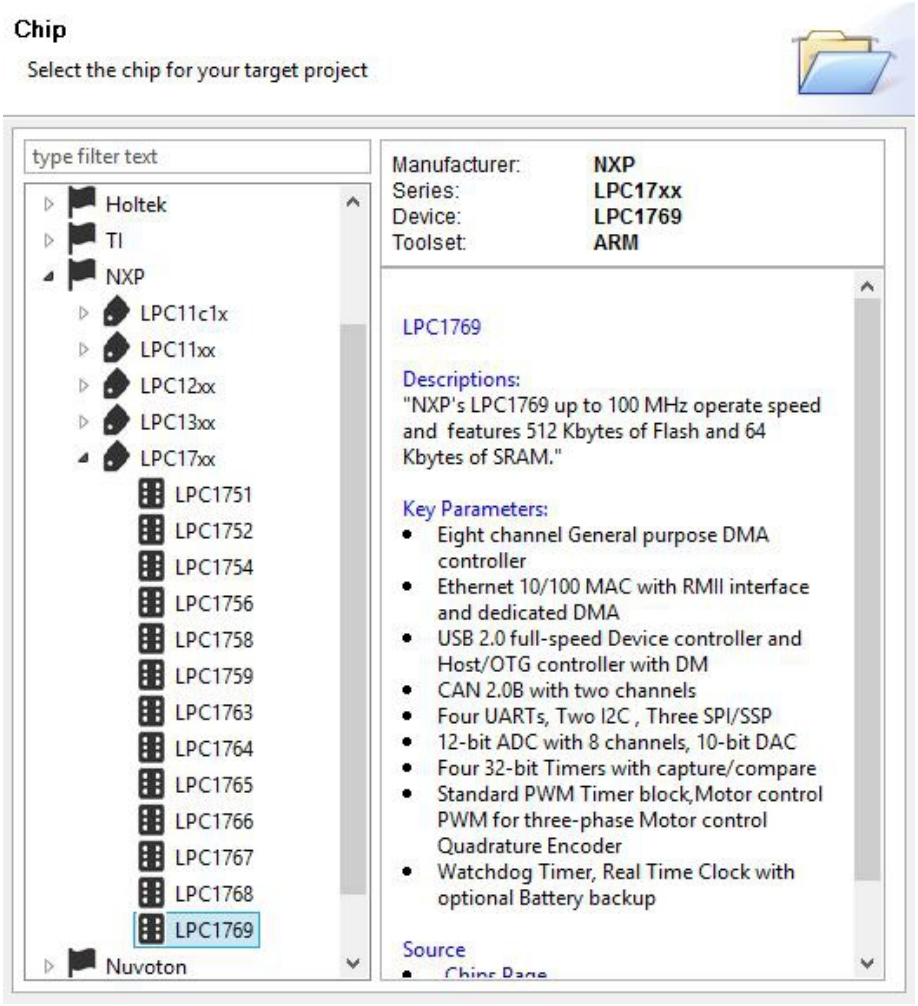


Figure 4: Selecting the Microprocessor Chip

One really impressive CooCox feature is the Repository, which allows you to click on the peripherals you need and it will automatically add the required driver software to your project. Figure 5 shows this repository view for the LPC1769 chip.

In Figure 5, the project definition was started by selecting the **GPIO** peripheral and clicking its check box.

What happens next is a really nice feature of CooCox. The system ascertains that you are going to need additional software modules to have a complete project. These modules are the things that support the interrupt vector table, start-up code, clock setup and the like.

Therefore the Repository automatically initiates the selection of the **CMSIS core**, **Common_Header**, **CMSIS_boot**, and **CLKPWR** components and thereby adding their drivers to the project. Now we have a complete project with a “do nothing” main program that will actually compile and link and could be downloaded to flash.

The list of peripherals in the Repository matches exactly the peripheral devices supported by your selected ARM-Cortex chip.

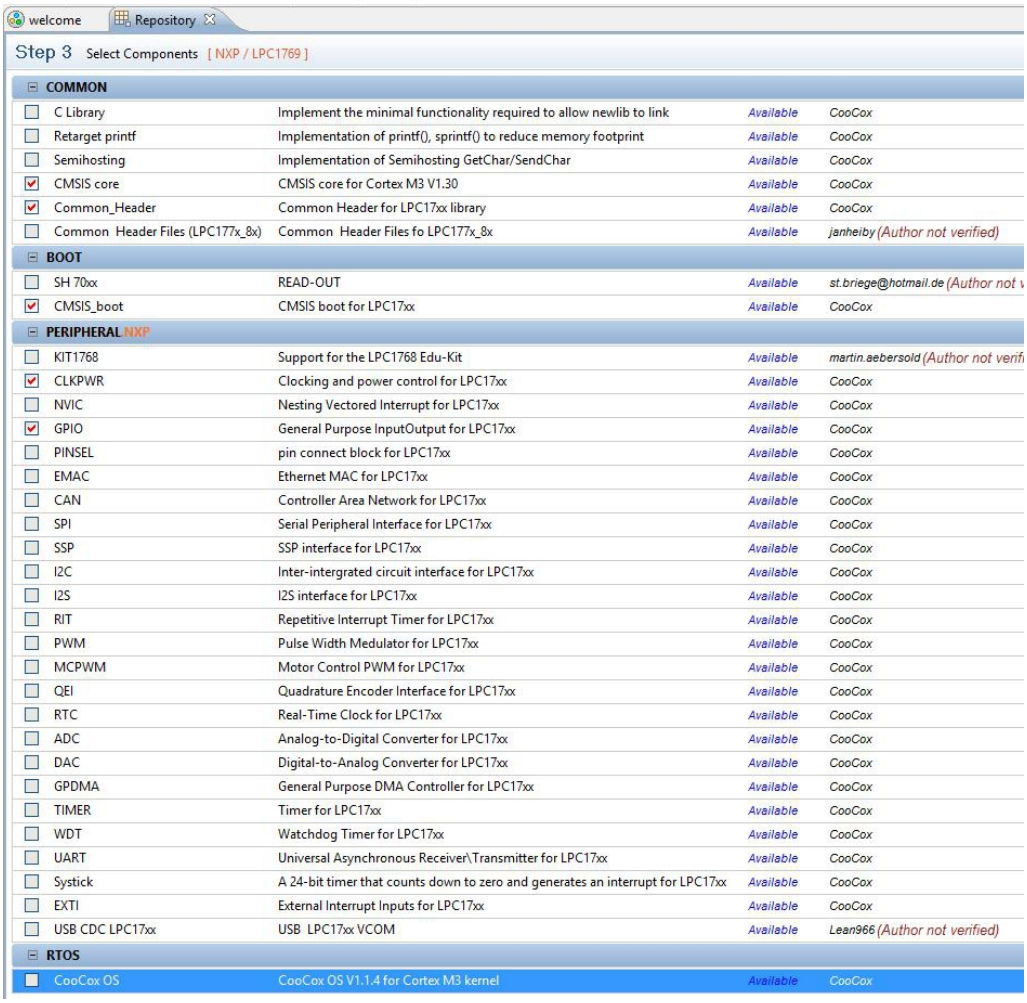


Figure 5: CooCox Repository Includes Drivers for the Peripherals You Select

There's also a “Components” view that you can display in CooCox. It enumerates the software components you have currently selected, as shown in Figure 6.

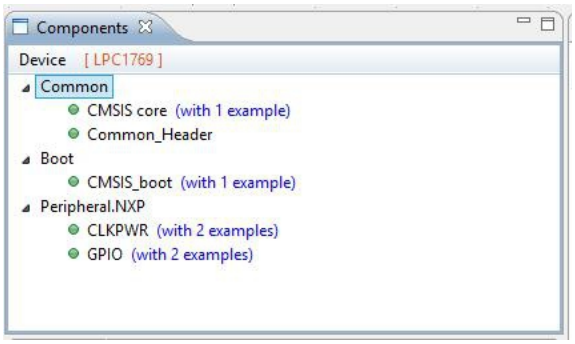


Figure 6: CooCox Components View

Notice that in Figure 6, some of the selected components have “examples” that you can link to. If you like the example (you can view it), you can opt to have it added to your project. Understand that these examples are generic and may need some modification to function. For example, I had to change the GPIO port number and bit position to target the example code to the specific LED that is on the LPCXpresso target board. Still, this is a very nice feature.

When it is time to compile, link, download, and debug your project, CooCox has simplified these actions down to five tool bar buttons with the simplicity of the Arduino system. Figure 7 shows these toolbar buttons.

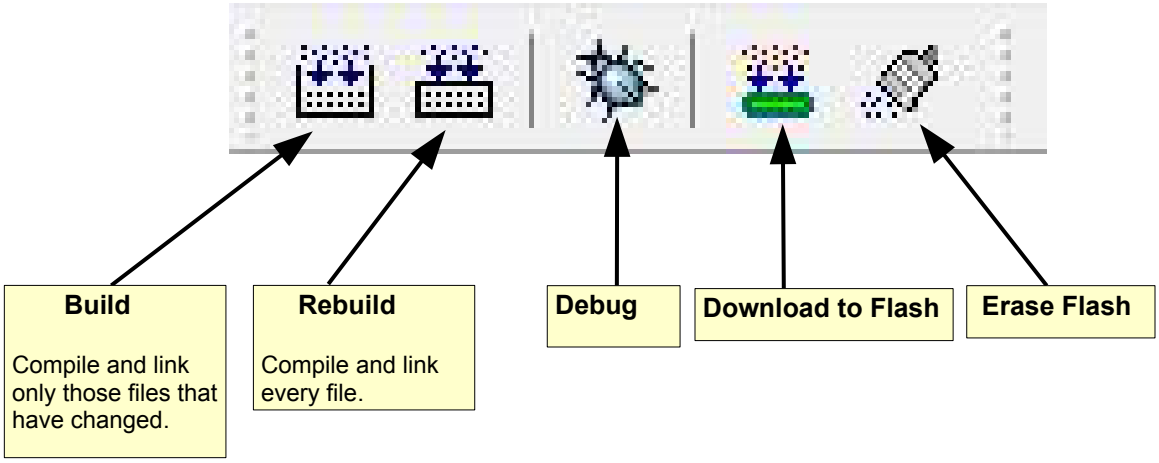


Figure 7: CooCox Compile and Debug Buttons

In 2007, I wrote a tutorial for the Atmel Corporation titled “**Using Open Source Tools for AT91SAM7S Cross Development**”. This showed all the setups and codes required to configure an Eclipsed-based ARM cross development system. The tutorial was 145 pages and was read by hundreds of thousands of people. The number of pages required just to explain how to set up a functioning Eclipse debug launch configuration was daunting!

CooCox has simplified all that detail to one very simple window; you just have to identify your debugger, whether it is SWD or JTAG, and its speed. Once done, their plug-ins take care of everything. Figure 8 shows the configuration window for the debugger setup.

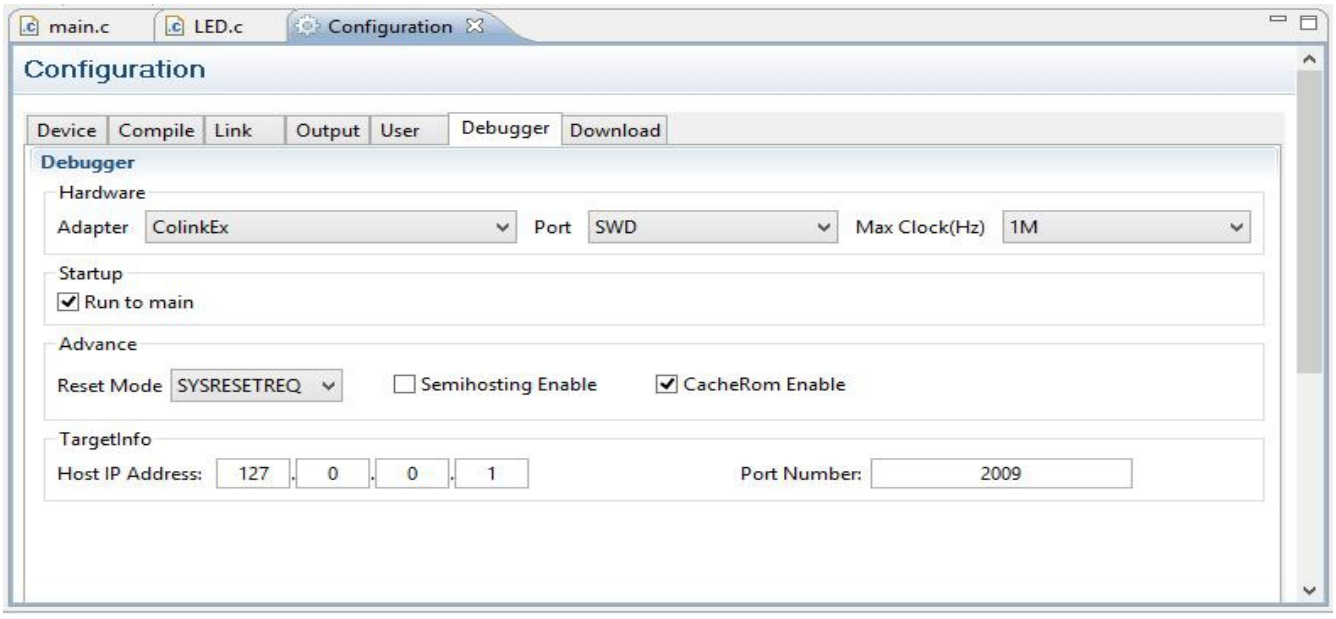


Figure 8: CooCox Debugger Configuration

In Figure 8 above, I selected the CooCox **CoLinkEx** debugger board and selected **SWD** debugging in the Port pull-down menu box. Figure 9 shows all the debug hardware adapter choices currently available in CooCox. Note that there are many debuggers supported and some of them are reasonably priced.

Note the “Max Clock(Hz)” on the far right. If you have trouble making the debugger work, try selecting a slower max clock rate. I had to reduce this rate to 100 Khz for the LPC1769Xpresso board.

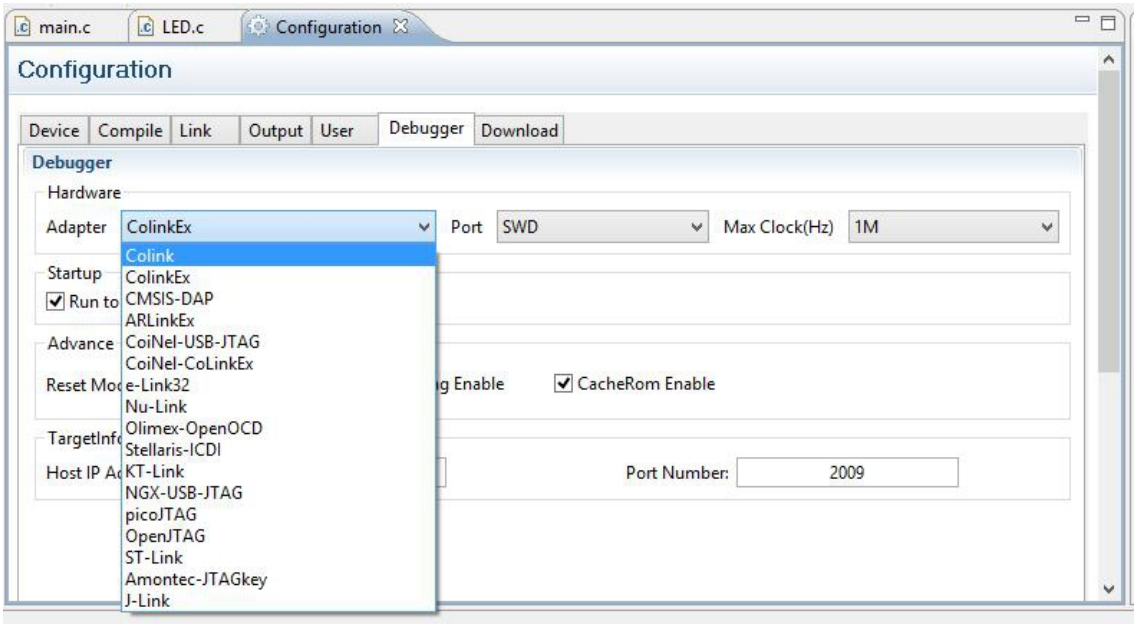


Figure 9: Hardware Debug Adapters Supported by CooCox.

Installing the CooCox ARM-Cortex Software Development System

To create a CooCox software development system on your computer, there are four steps required.

1. **Download and Install the GCC ARM Embedded tool chain**

2. **Download and Install the CooCox Eclipse-based IDE**

3. **Tell CooCox where the GCC ARM Embedded tool chain is located**

4. **Download and Install the CoLinkEx Debugger Firmware and USB Drivers**

Fortunately, most of the steps involve Windows installers where you can just take the “default” settings. The only exception is step 3 where you have to know where the compiler is located.

Download and Install the GCC ARM Embedded tool chain

The GNU tool chain was developed by the Free Software Foundation (http://www.gnu.org/) and they maintain the master set of compiler, linker, and debugger source codes.

A group of engineers at ARM, using the GNU source codes, periodically update the GNU tool chain to support new chips that ARM and its partner companies develop. They upgrade GNU several times a year and provide "binaries" that anyone can freely download and install. The project is called "***GNU Tools for ARM Embedded Processors***" and their web site (shown in Figure 10) is:

https://launchpad.net/gcc-arm-embedded

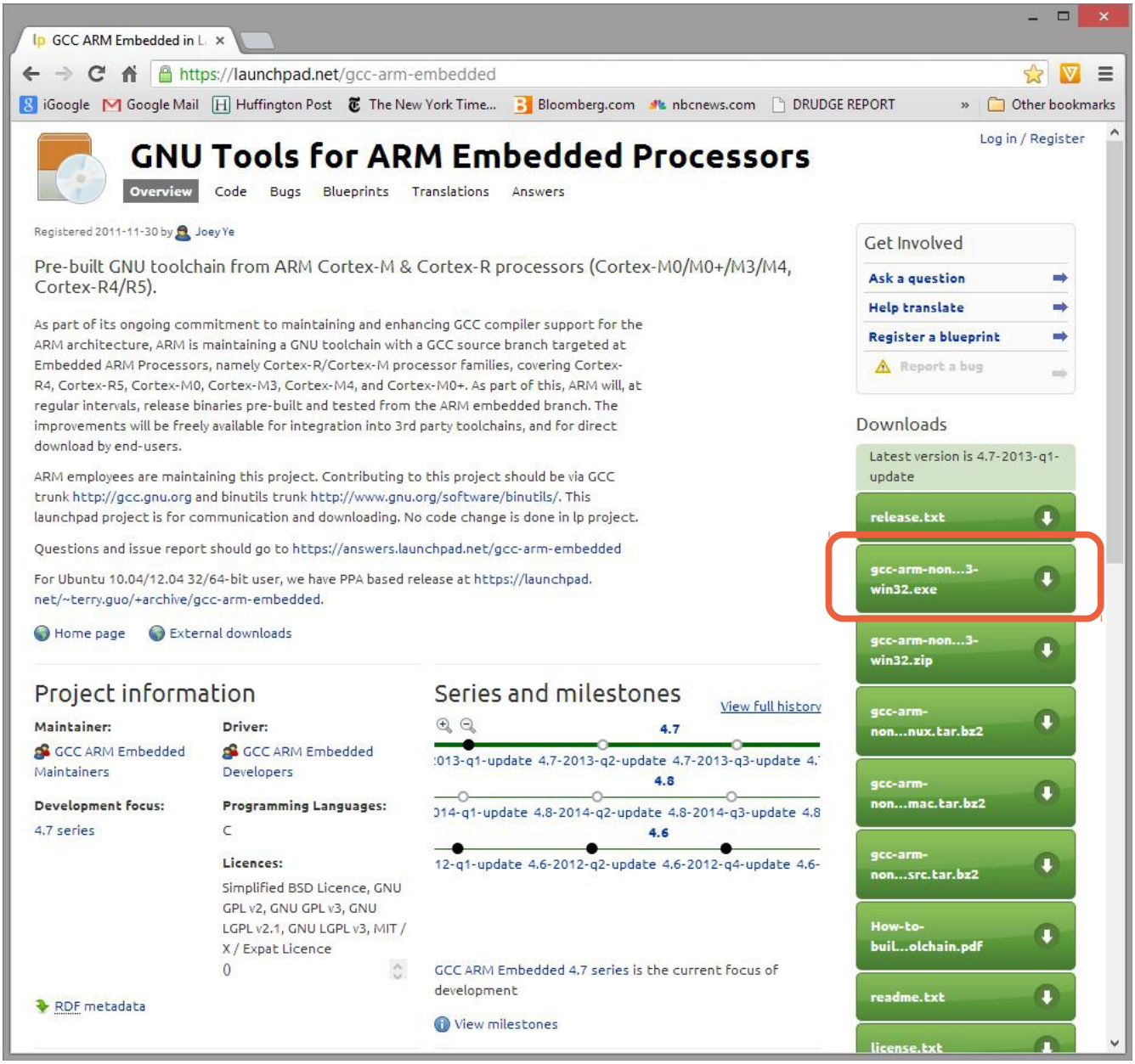


Figure 10: GNU Tools for ARM Embedded Processors Web Site

Double-click on the **win32.exe** button as shown in Figure 10 to start the download.

Figure 11 shows the download in progress using the Google Chrome Browser. It's about 95 megabytes.

Figure 11: Downloading the Tool Chain (compiler, linker, debugger)

When the download finishes, select "**Open**" in the Google Chrome download pull-down menu, as shown in Figure 12, and that will start the tool chain installation process

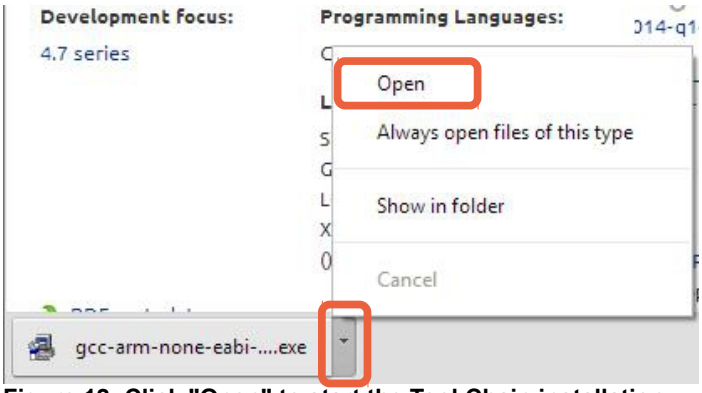


Figure 12: Click "Open" to start the Tool Chain installation

Fortunately, you can take the default choice on every installation screen. In Figure 13, English is the default. Figure 14 is the authorization to proceed.

Figure 13: Select the Desired Language

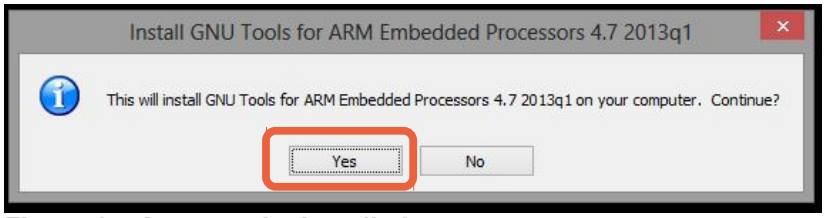


Figure 14: Agree to the Installation

Figure 15 is a “welcome” screen while Figure 16 is the “license agreement”. Be sure to click the “**I accept...**” check box and proceed.

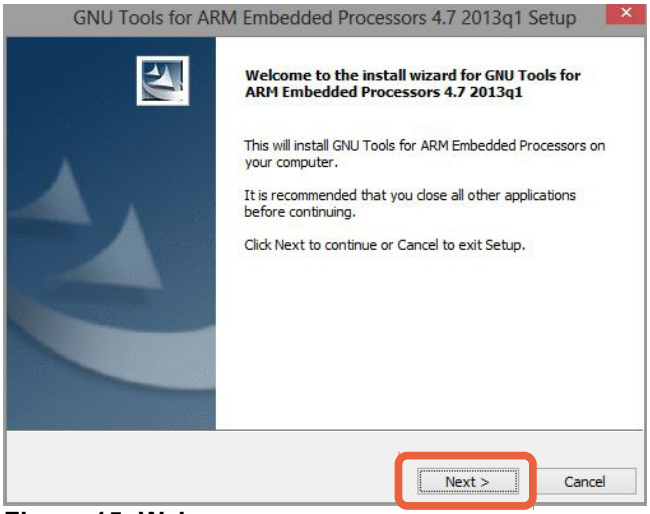


Figure 15: Welcome screen

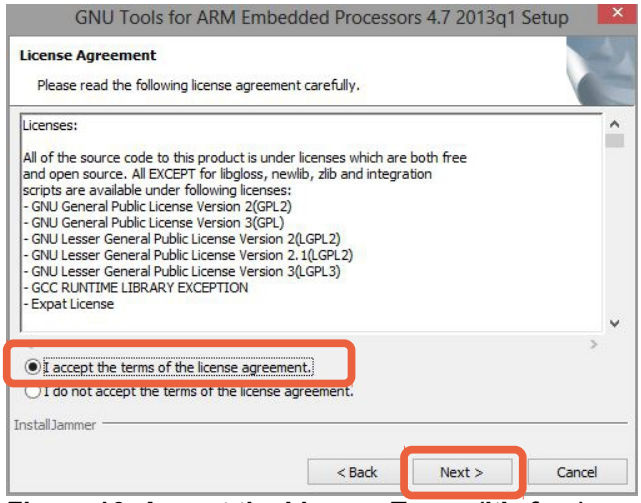


Figure 16: Accept the License Terms (it's free)

Figure 18 is very important – it indicates where the tool chain will be installed (what folder). You should write this down somewhere. When we install CooCox, they will ask for the path to the compiler executable, which is in a **\bin** sub-folder of the path in Figure 18.

Figure 17 is just another authorization to proceed.

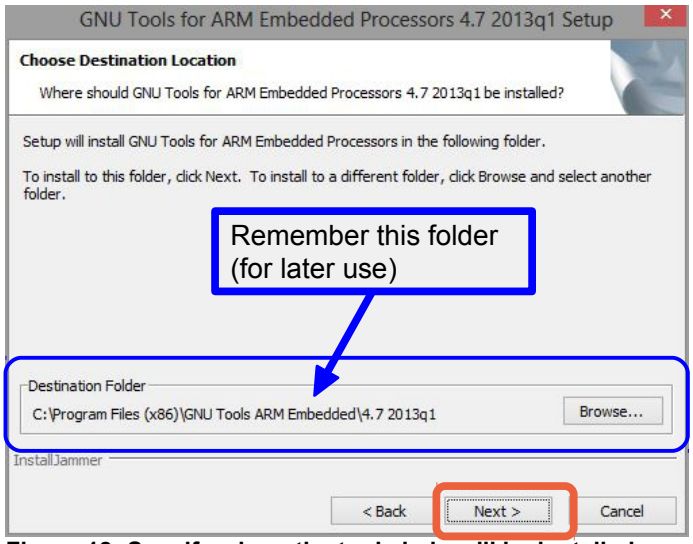


Figure 18: Specify where the tool chain will be installed

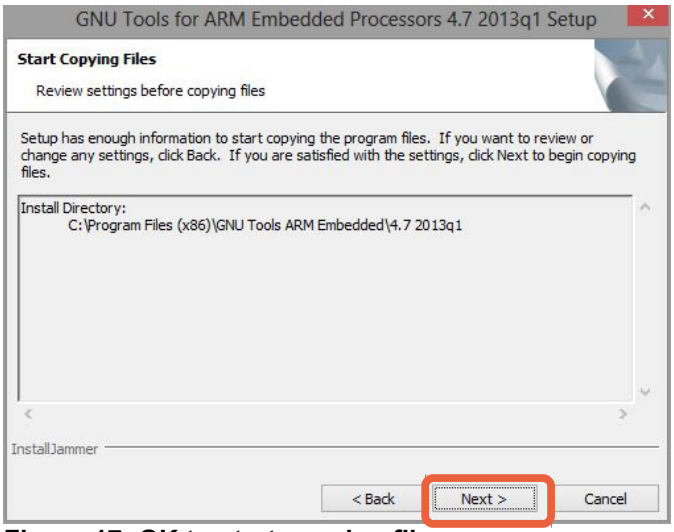


Figure 17: OK to start copying files

Figure 20 shows the installation in progress. When completed, the final screen is shown in Figure 19. Hit “**Finish**” to complete the installation of the GNU ARM Embedded Tool Chain.

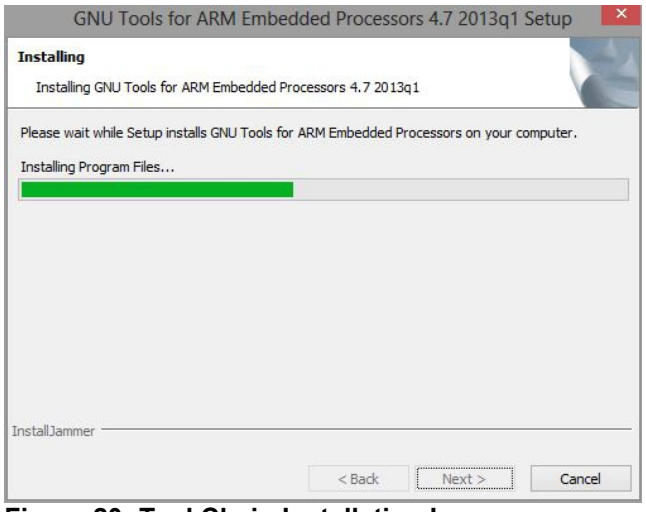


Figure 20: Tool Chain Installation In-progress

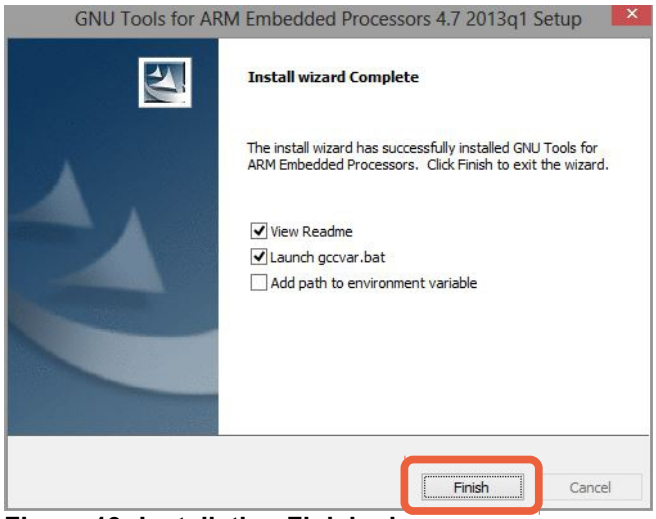


Figure 19: Installation Finished

The Tool Chain installation is finished; there is one bit of information that you must figure out: "**Where is the C Compiler?**". The installation indicates this location (see Figure 18), the compiler will be in a sub-folder called **\bin**. The reason why we must know the path is that CooCox will specifically need this.

On my computer (Windows 8), the folder holding the executables for the compiler, linker, and other utilities is:

C:\Program Files (x86)\GNU Tools ARM Embedded\4.7 2013q1\bin

The C compiler we're looking for is: **arm-none-eabi-gcc.exe**

Figure 21 is a Windows Explorer display that shows the **\bin** folder that holds the compiler CooCox will be using. When you install CooCox, they will ask you to connect to the compiler by identifying its containing folder. So it behooves you to know how to locate or specify it. Be smart, write it down!

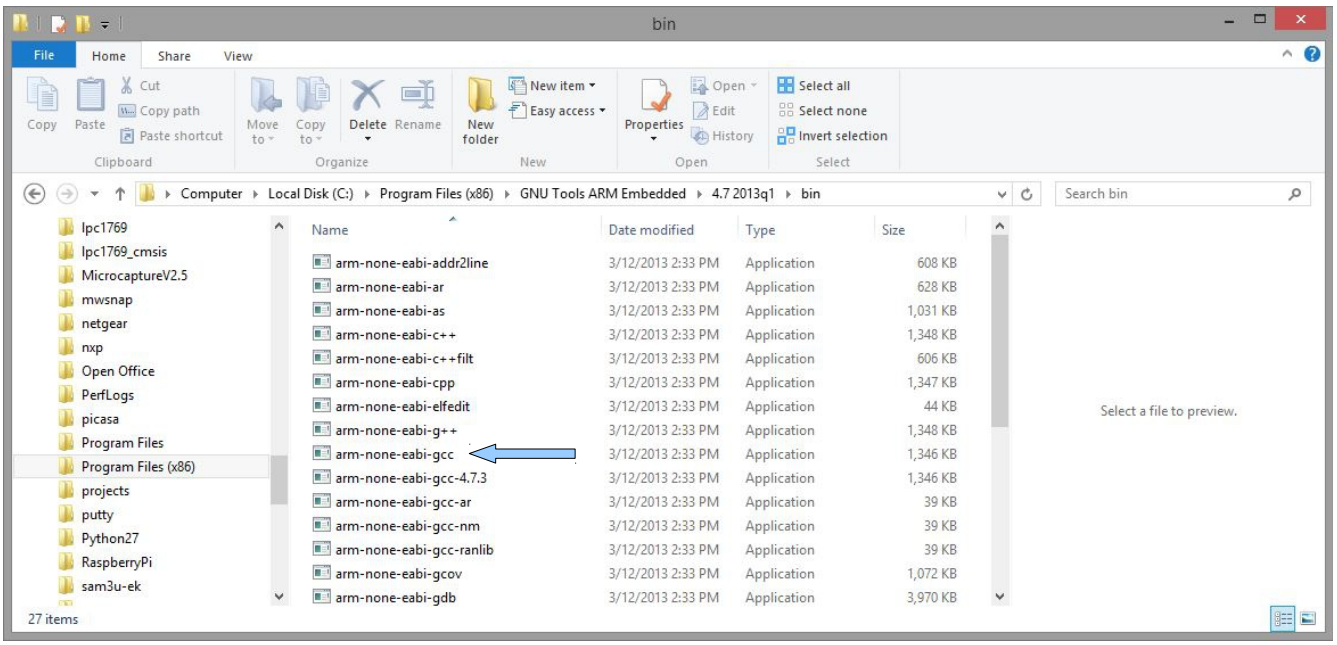


Figure 21: Where the C Compiler is Located

Download and Install CooCox IDE

The CooCox IDE can be installed from their web site: http://www.coocox.org/CooCox_CoIDE.htm#

When you enter their web site (it's somewhat cluttered), you should see something similar to Figure 22.

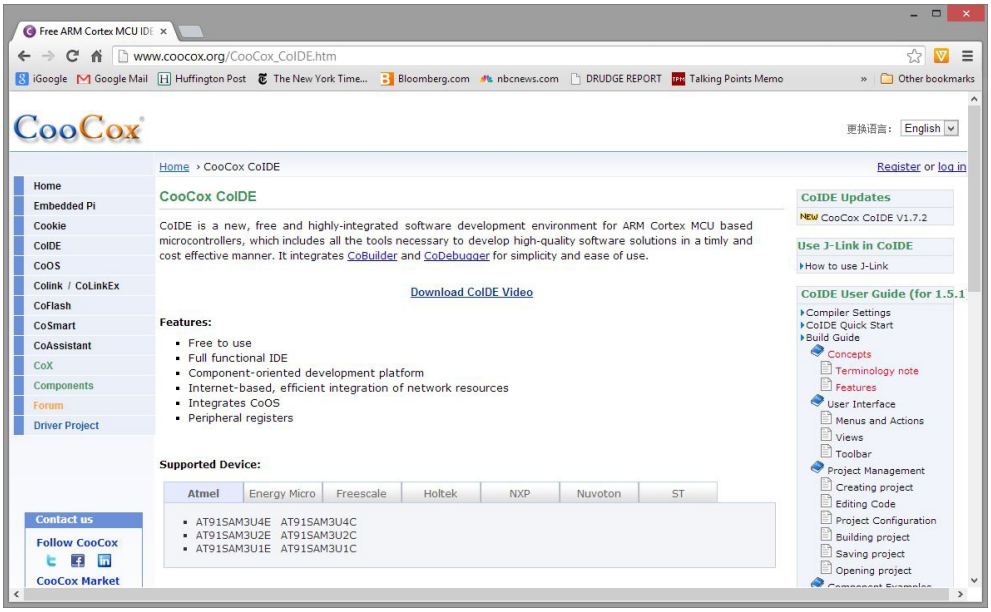


Figure 22: CooCox Web Site

If you scroll down to the bottom of the web site home page, you will see the download link as shown in Figure 23. Click on "**Download the latest CoIDE directly**".

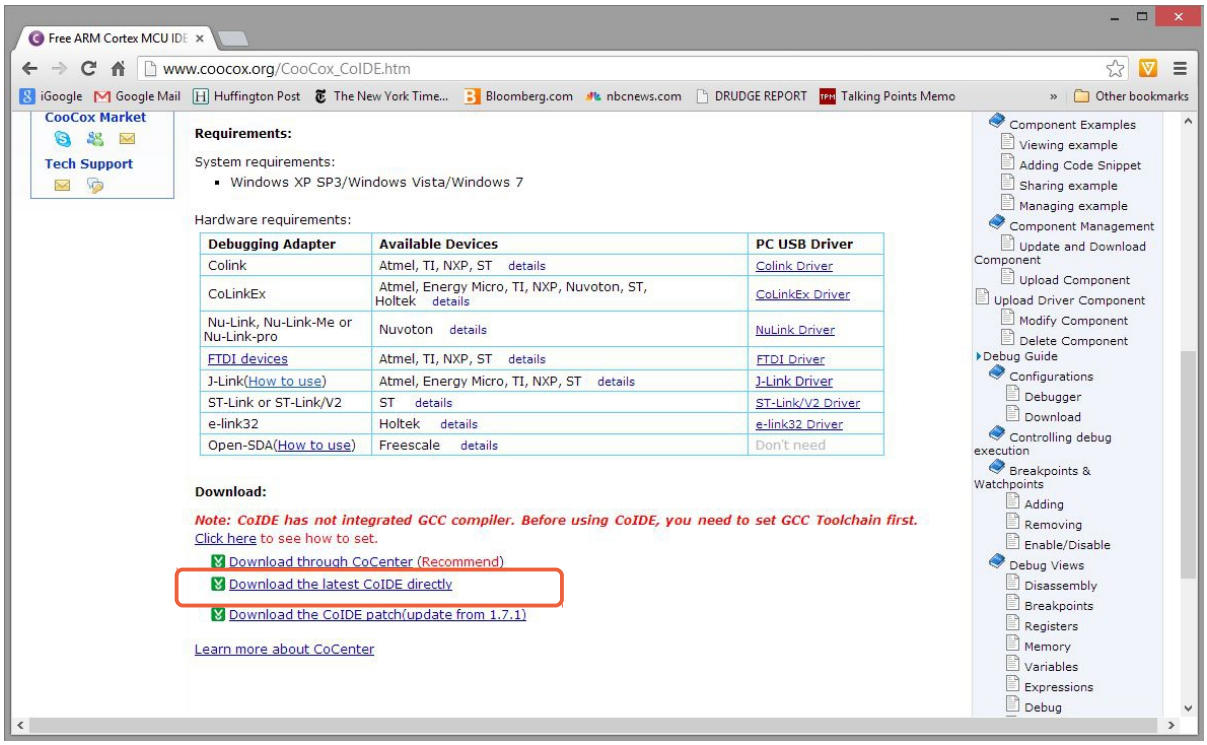


Figure 23: Download CooCox Directly

CooCox will ask you to identify yourself so you'll have to devise a user-name and password and give your email address. Don't worry, it is still free. Click "**Register**" in Figure 24 to initiate the download.

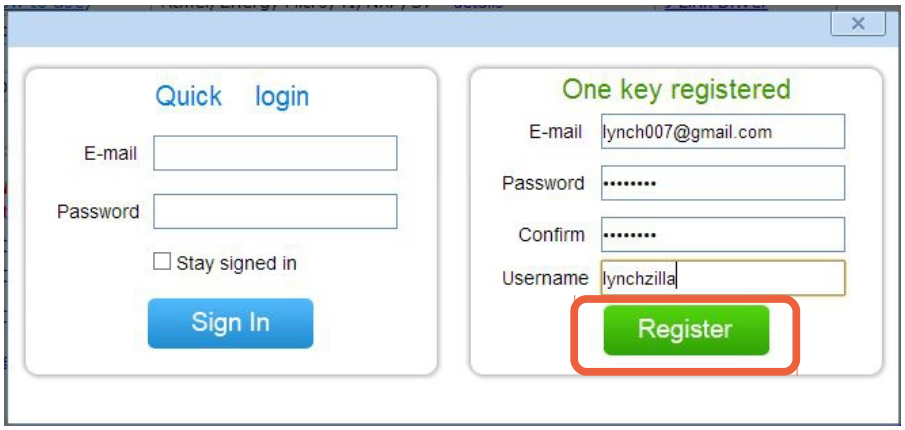


Figure 24: CooCox Registration Screen

The download (Figure 25) is massive (259 MB) so it will take a few minutes. Figure 26 shows the Google Chrome Browser indicating that it's finished. In the Chrome browser, clicking "**Open**" in the pull-down menu will start the CooCox IDE installation.

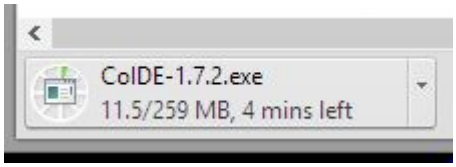


Figure 25: Download Is Quite Large

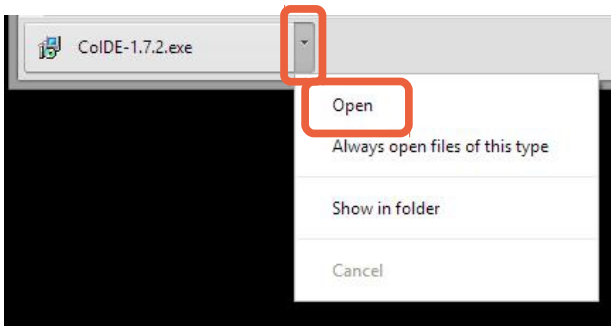


Figure 26: Click "Open" to Start Installation

Installation of the CooCox IDE is simple; just take the defaults on every screen. Figure 27 is just a welcome screen, click "**Next**" to continue.

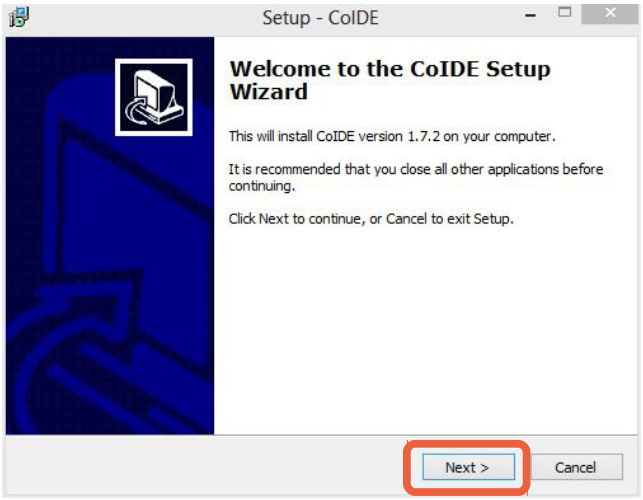


Figure 27: Welcome Screen

Figure 28 shows where the CooCox IDE will be installed on your computer. If this folder does not exist, CooCox will create it for you. Click “**Next**” to proceed.

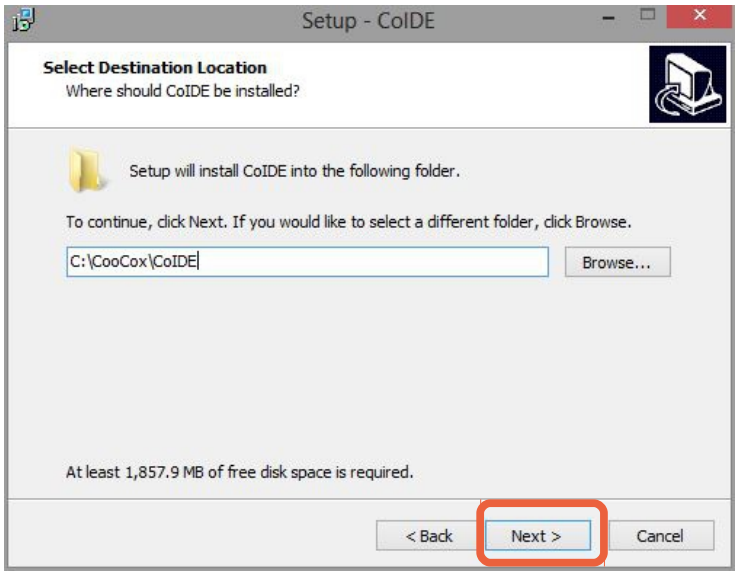


Figure 28: CooCox Destination Folder Location

CooCox will place an entry in the Start Menu for you as shown in Figure 29. Click “**Next**” to proceed.

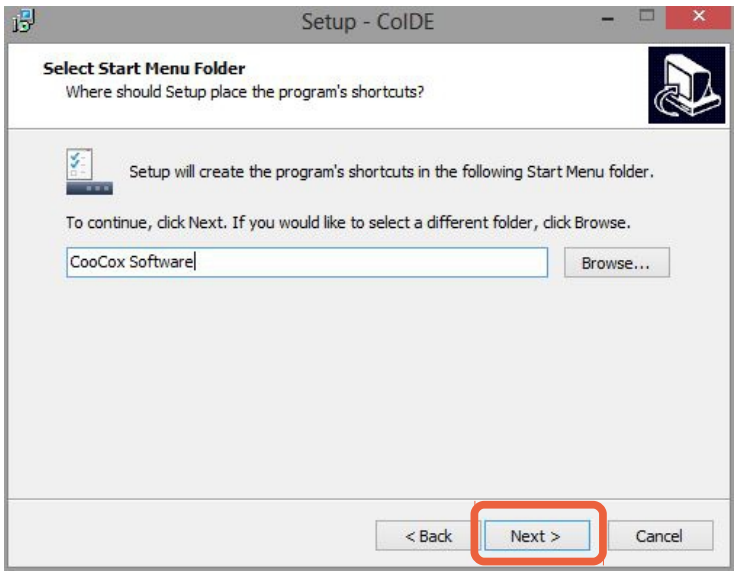


Figure 29: Place a Shortcut on the Start Menu

CooCox has a feature called **CoCenter** which is a Windows application that simplifies maintenance of the IDE and other utilities since it automates the download and installation of software updates, etc. This didn't work on my Windows 8 computer, but there's no harm done in taking the default and letting **CoCenter** install as shown in Figure 30. Click “**Next**” to proceed.

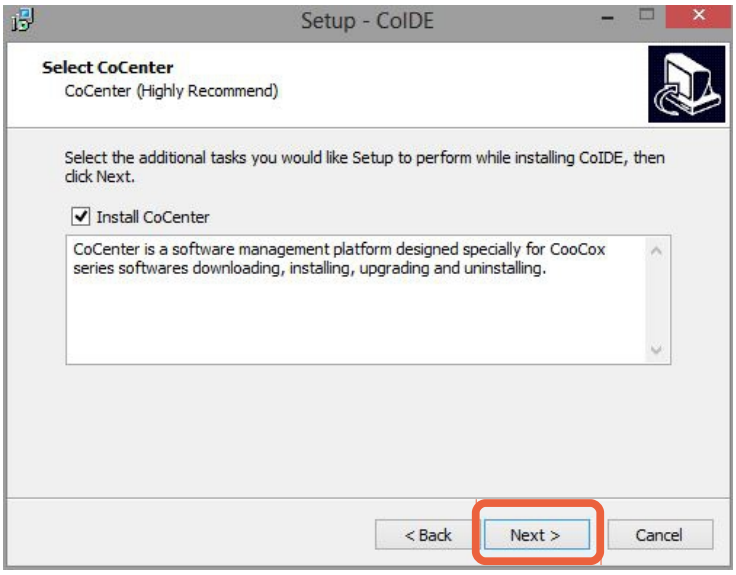


Figure 30: Install CoCenter

The CooCox installer gives one last look at where it will install, what the Start Menu folder looks like, and if CoCenter was chosen. Click "**Install**" to actually start the installation.

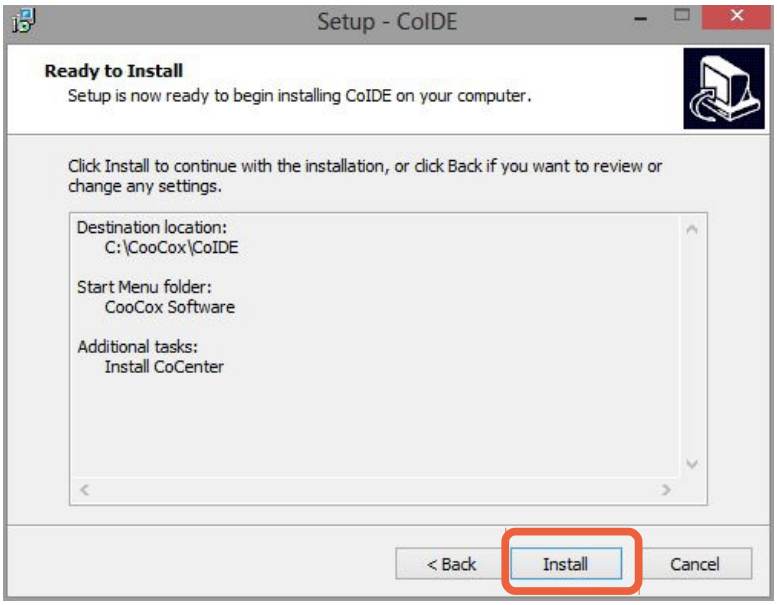


Figure 31: CooCox IDE is Ready to Install

Figure 32 shows the CooCox installer in action. The CooCox IDE is huge since it includes Eclipse, the Eclipse CDE C development plug-ins, and CooCox's own plug-ins. Installation will take several minutes. Figure 33 shows the Installation has completed, click "**Finish**" to exit the installer.

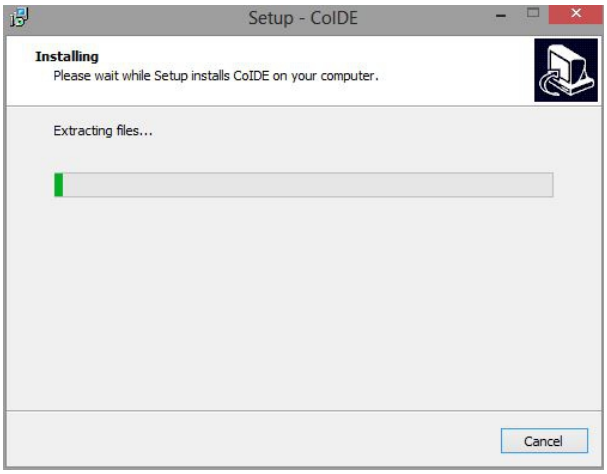


Figure 32: CooCox IDE Installation In Progress

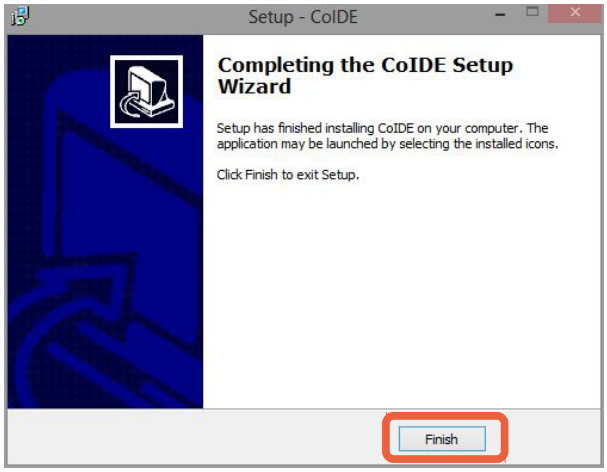


Figure 33: CooCox IDE Installation Complete

Now we can check if CooCox IDE actually works. Click on the desktop icon as shown in Figure 34.

Figure 34: Desktop Icon

CooCox has a Splash Screen shown in Figure 35.

Figure 35: CooCox Spash Screen

If the CooCox IDE was installed properly, the CooCox Eclipse main screen will appear as shown in Figure 36. This is all standard Windows stuff; there's a pull-down menu line, a “toolbar button” line and a series of sub-windows (called “perspectives” in Eclipse-speak). The very bottom of the Eclipse main screen is a status line.

Figure 36 shows 4 current perspectives: Components perspective, Project perspective, Help perspective and a “Welcome” perspective. You click anywhere within the perspective to give it focus (make it active).

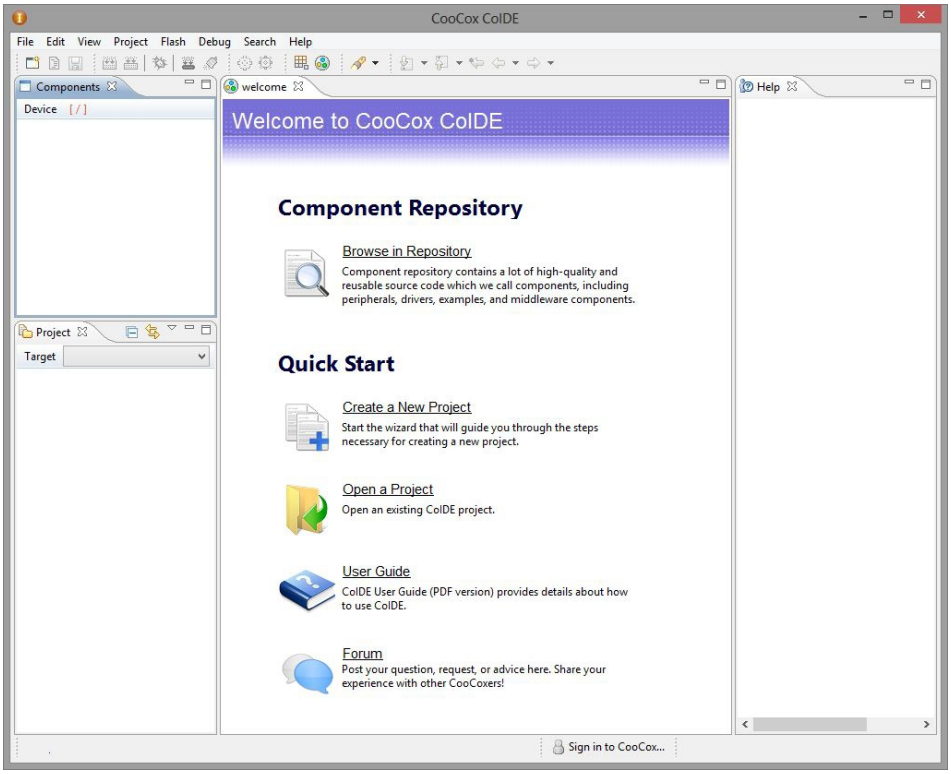


Figure 36: CooCox Home Screen

The very first thing that must be done is to “connect” the CooCox IDE to the GNU ARM Embedded Tool Chain.

Connect CooCox IDE to the GNU Tool Chain

The final bit of magic required to get CooCox editing and building software is to inform CooCox where the tool chain is located. In the CooCox home window, click “**Project**” followed by “**Select Toolchain Path**” to specify the folder location of the GNU compiler, etc (refer to Figure 37).

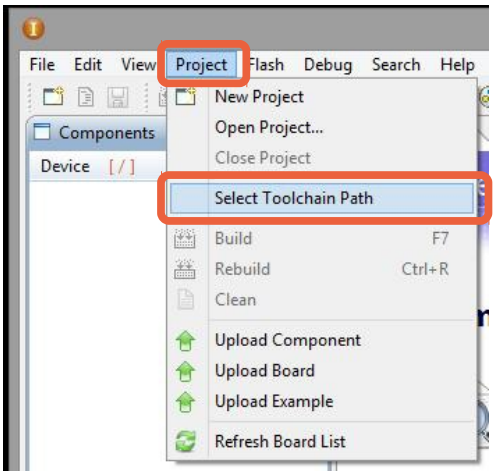


Figure 37: Click "Select Toolchain Path" to connect CooCox to the Compiler

A text box appears where you could just type the path to the GNU compiler, assuming that you wrote it down or pasted the path into the Windows Clipboard. Most people will just select “Browse” and find the

compiler that way, as shown in Figure 38.

Figure 38: Specifying the Path to the Compiler

On my computer (Windows 8), the folder holding the executables for the compiler, linker, and other utilities is: **C:\Program Files (x86)\GNU Tools ARM Embedded\4.7 2013q1\bin**

Figure 39 shows the result of browsing for that specific folder. Click "**OK**" to establish the connection.

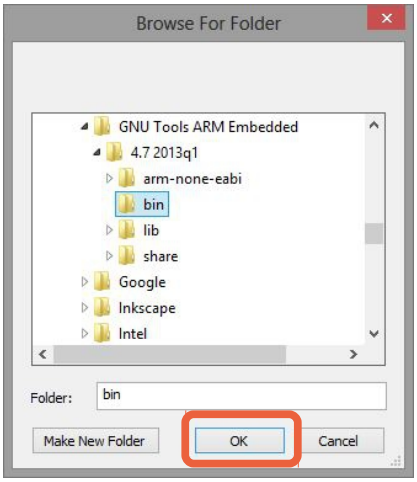


Figure 39: Browse to the Folder Containing the Compiler

CooCox will give you one last chance to approve the Toolchain path, as shown in Figure 40. Click "**OK**" to approve the selection.

Figure 40: Final Approval of the Toolchain Path

Modify The LPCXpresso Board

The LPCXpresso board is built in two parts: the hardware debugger and a LPC1769 target board. If you refer back to Figure 1 at the beginning of this document, there is a line drawn through the J4 header (2x8) that separates the two parts. The J4 header transfers the SWD debugger signal lines from the on-board debugger to the target section. To do this by default, there is a factory-installed solder bridge on each pin-pair on the J4 header. Figure 41 is a close-up of this J4 header and you can see the solder bridges that come with the board.

To connect the CooCox CoLinkEx debugger to the LPC1769 target section, these bridges have to be removed. Figure 42 shows three of the bridges removed (work in progress). You can do this with a narrow point soldering iron tip and a rosin Flux-Pen. Just apply a little flux, heat up the solder bridge, and “swipe” it with the tip until the bridge is dislodged enough to remove the short circuit, as shown in Figure 42. Figure 42 shows the work in progress; you must remove **all** the solder bridges on the J4 header!

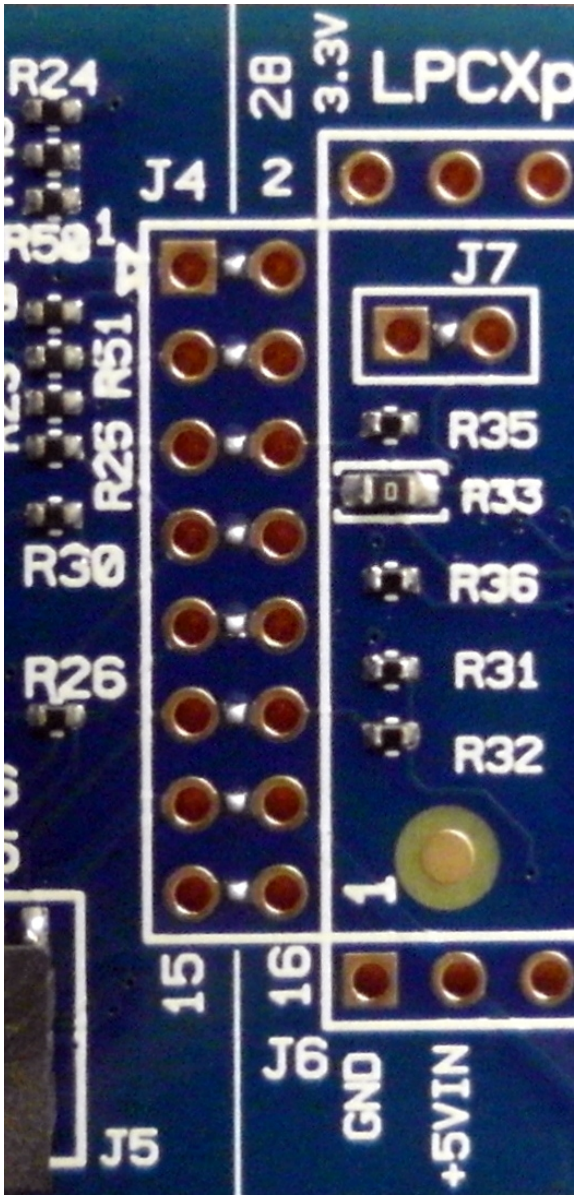


Figure 41: Solder bridges of Header J4

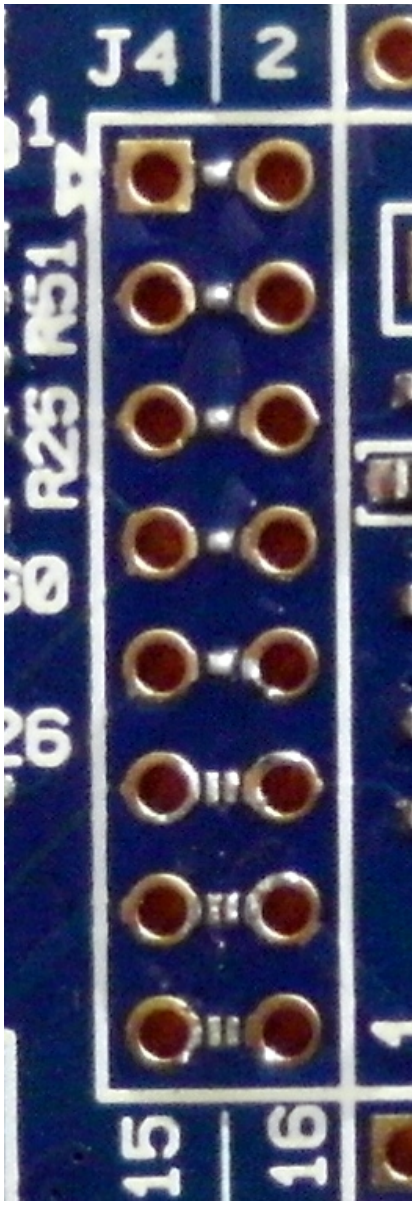


Figure 42: Bottom 3 solder bridges removed

Once the factory-installed J4 solder bridges have been removed, the next modification is to install a 2x8 header at the J4 location. In Figure 43, you can see that the 2x8 header pins are inserted into the top and soldered on the bottom of the board. With the factory-installed solder bridges removed, the LPC-Link debugger on the left in Figure 43 is actually disconnected. Now we can connect a third-party debugger, such as the CoLinkEx, to the right hand pins of header J4 (pins 2, 4, 6, 8, 10, 12, 14, 16).

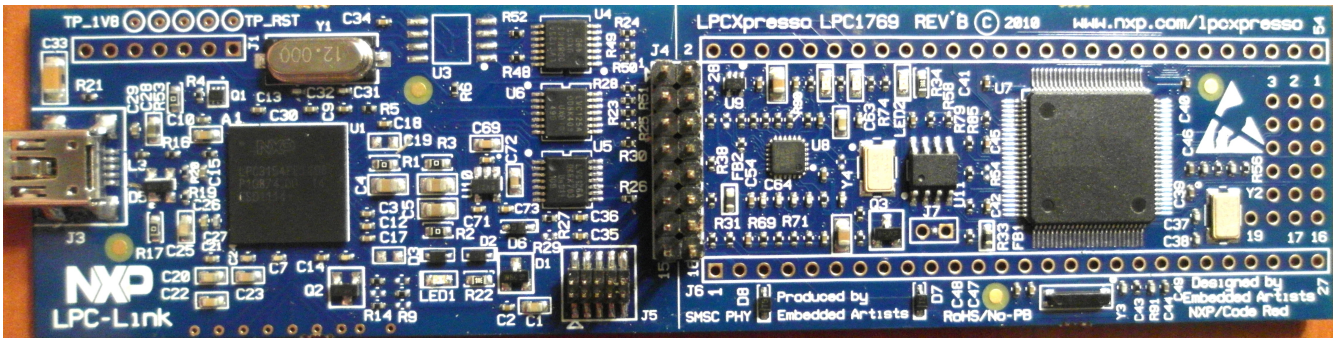


Figure 43: Two 8-pin headers soldered to the J4 pads

These headers are inexpensive and can be ordered from:

www.sparkfun.com	Break-away headers – straight – 40 pins one piece	$1.50	PRT-10158
www.adafruit.com	Break-away 0.1" 36-pin strip male header -10 pieces	$7.50	ID:392

Setting Up the CoLinkEx Debugger

The CooCox CoLinkEx debugger is $27.91 and connects the host desktop PC to the target board via USB. Figure 44 shows this debugger, you can purchase it through the CooCox web site: http://www.coocox.com/

Their web site vectored me to the Newark/Element14 USA web site where they had the board in stock.

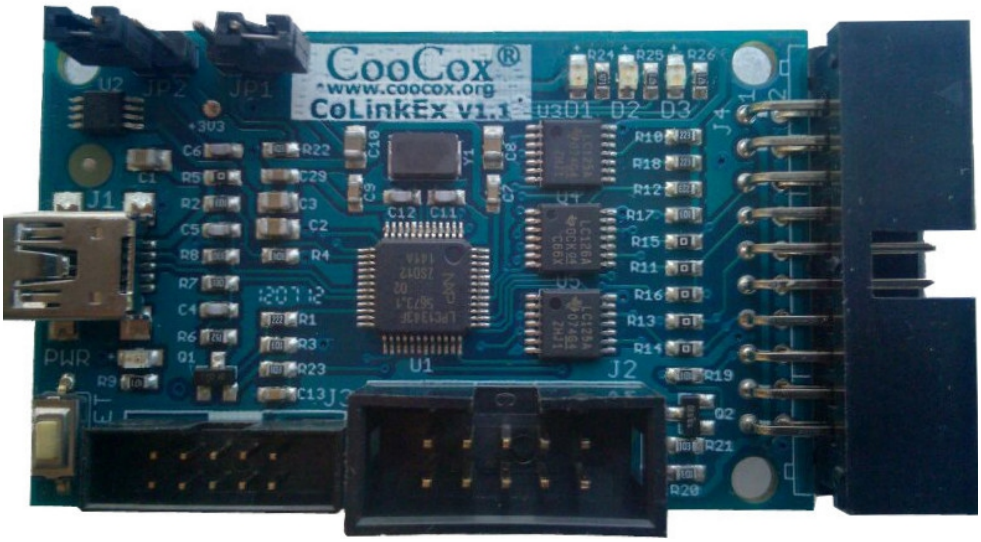


Figure 44: CooCox CoLnkEx Debugger

CooCox provides very good documentation (in English). The CoLinkEx User Manual can be downloaded from here: http://www.coocox.org/downloadfile/Colink/CoLinkEx_User_Manual.pdf

Upgrade the CoLinkEx Firmware

The first thing that should be done is to upgrade the CoLinkEx firmware with the latest and greatest revision from CooCox. The CooCox web site has a link to download this firmware and the User Manual has extensive instructions on how to do this.

Here's the link to the CooCox CoLinkEx page (see Figure 45): http://www.coocox.org/colinkEx.htm

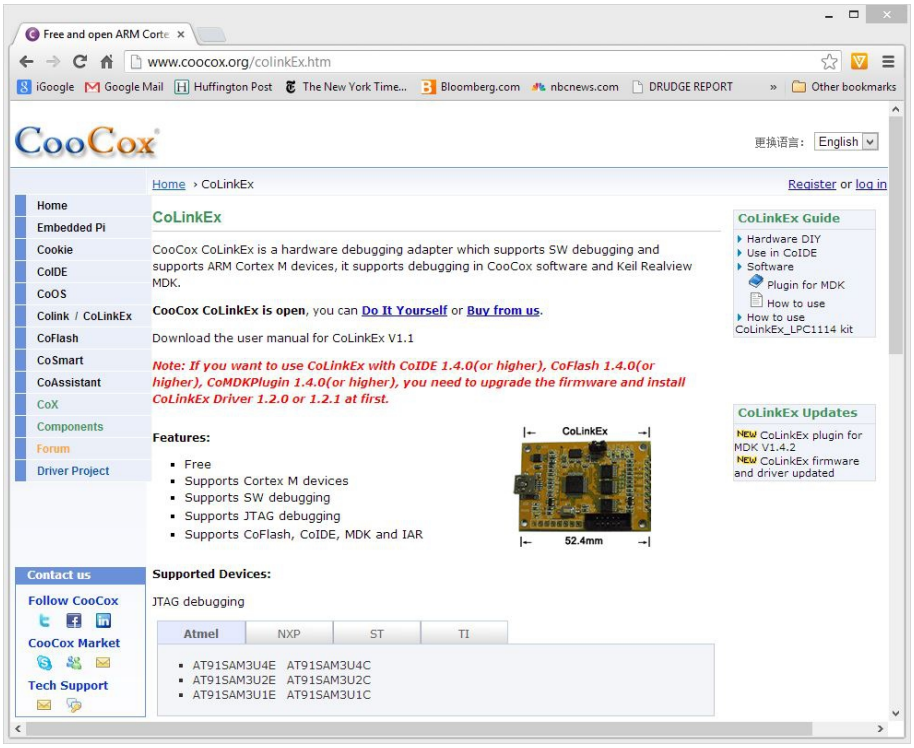


Figure 45: CooCox CoLinkEx Web Site

Scroll to the bottom of the CoLinkEx page as shown in Figure 46 below.

Figure 46: Click on CoLinkEx Firmware

Click on the **CoLinkEx firmware** as shown in Figure 46 above and download it. When the download is complete, you should see it in your browser's download folder, as show in Figure 47. Note that it is a "**.bin**" file.

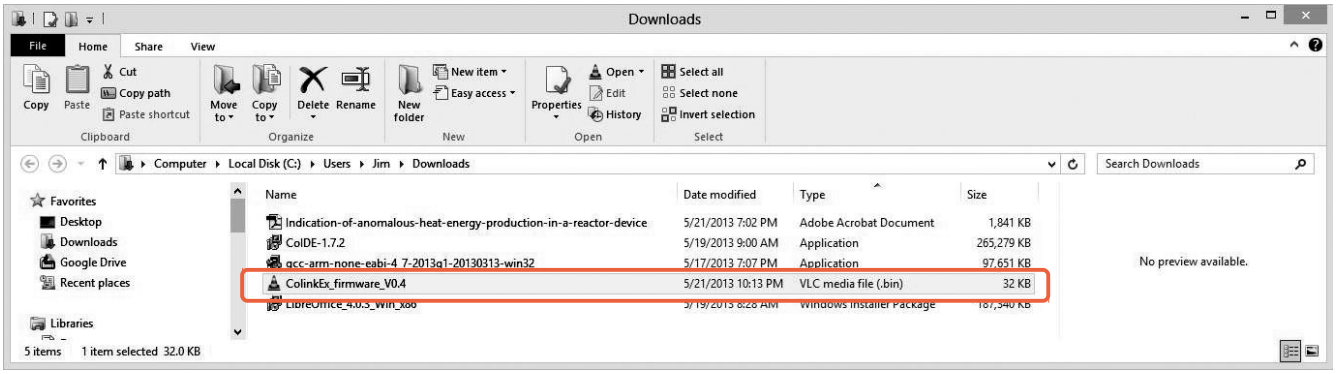


Figure 47: CoLinkEx Firmware downloaded

To upgrade the firmware, you have to connect the CoLinkEx board to your computer via the USB cable **with the JP1 jumper installed** (Figure 48).

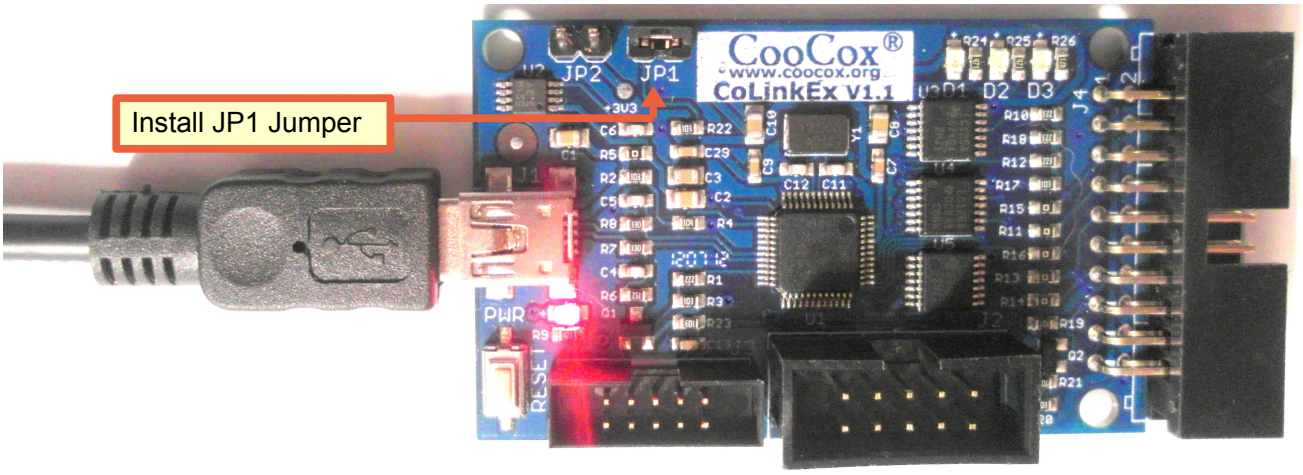


Figure 48: To Upgrade CoLinkEx firmware, Install the JP1 jumper.

When you plug in the CoLinkEx USB cable to your computer, you should hear a "beep" indicating that the computer has recognized the CoLinkEx board as an external hard drive (just like a thumb drive or flash card). When you use Windows Explorer to see all the storage devices on your system, the CoLinkEx board will be present as an external hard drive and you can browse to see what files are on the drive, as shown in Figure 49.

Note that in Figure 49 below, the CoLinkEx board on my computer is listed as "**CRP DISABLD (L:)**", an external hard drive that you can freely delete and add files into.

The firmware supplied by the factory is "**L:/firmware.bin**"

Before installing the new firmware, **delete the old firmware** (firmware.bin).

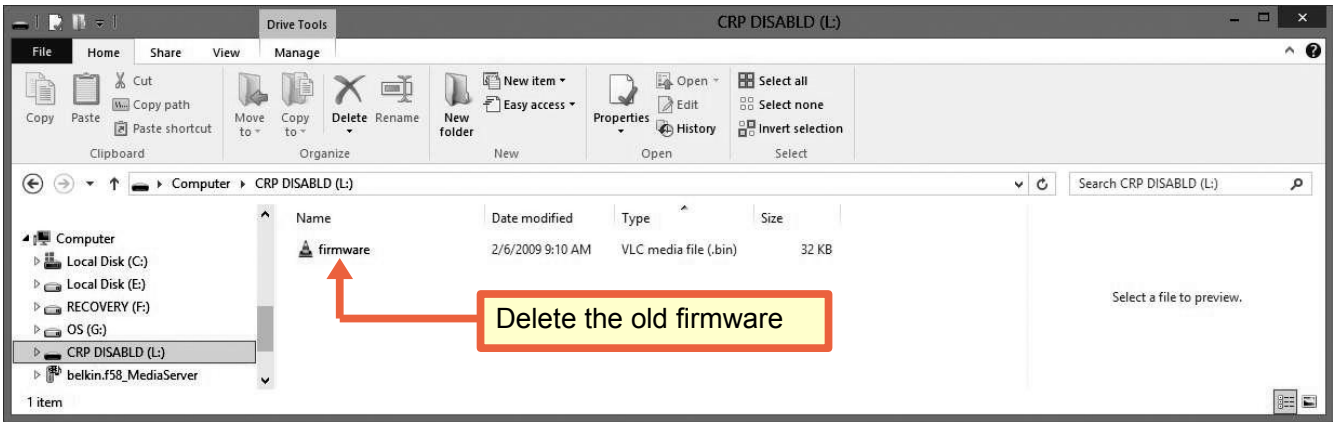


Figure 49: CoLinkEx Appears as a Hard Drive on Your System

Upgrading the CoLinkEx firmware is now very easy, just copy and paste the downloaded new firmware shown in Figure 47 to the "**CRP DISABLD L:**" drive. Figure 50 shows the result of that operation. In my case, I now have revision V0.4 of the CoLinkEx firmware installed.

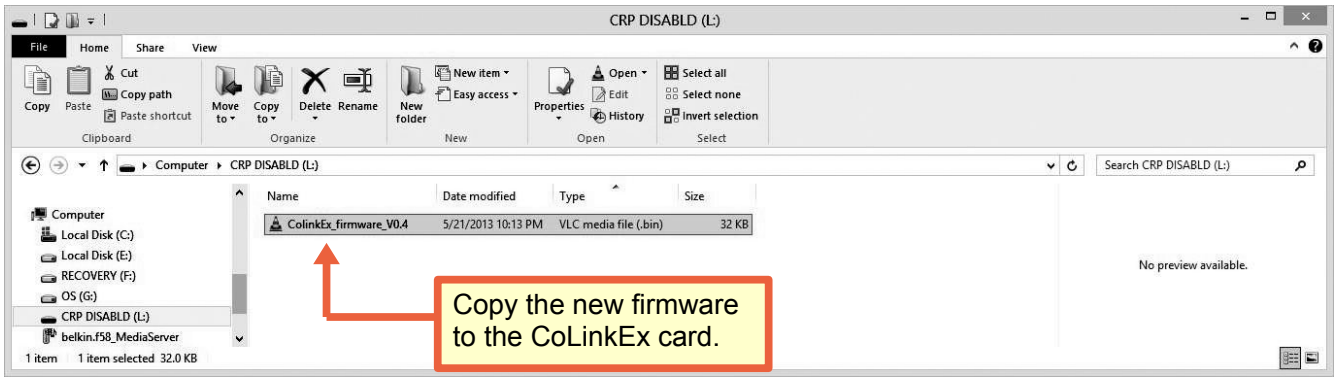


Figure 50: Latest and Greatest CoLinkEx Firmware Copied to Card

Fixing a Windows 8 Problem – Signed Driver Enforcement

If you have a Windows 8 operating system, attempts to install USB drivers to support the **CoLinkEx** debugger card will fail. Why? The reason is that default Windows 8 will not allow installation of device drivers that are not "signed" - installation of the **CoLinkEx** driver will abort and you probably won't have a clue as to why.

The solution is to run through a rather involved procedure to "turn off" signed driver enforcement. If you don't have Windows 8, you'll just get a pop-up display during USB driver installation that asks "driver is unsigned, proceed?". You can answer "Yes" and finish the job.

Nonetheless, for Windows 8 users, please follow the ensuing steps to turn off "*signed driver enforcement*".

- First, bring up the Charm Bar (click on the lower-right corner of the screen or simply hit the “Windows flag key and the letter “I” simultaneously). The Charm bar is shown in Figure 51.

  In the Charm bar, click “**Settings**”.

- Now click on “**Change PC Settings**” as shown in Figure 52.

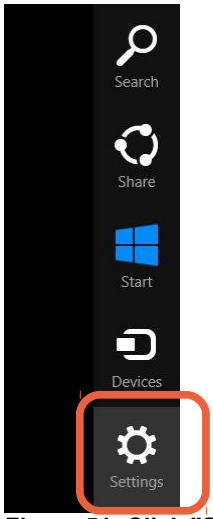


Figure 51: Click "Settings" on the Charm Bar

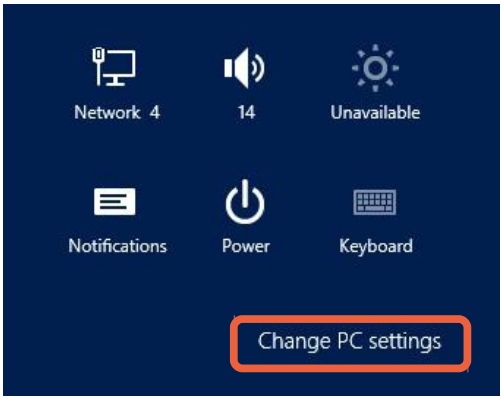


Figure 52: Click "Change PC Settings"

- Under the “**Change PC Settings**” section, click the “**General**” button (see Figure 54).

- In the “**General**” setup section, click the “**Restart now**” button just below the “**Advanced Setup**” section (Figure 53).

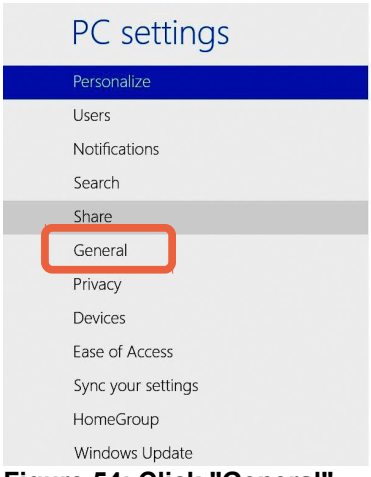


Figure 54: Click "General"

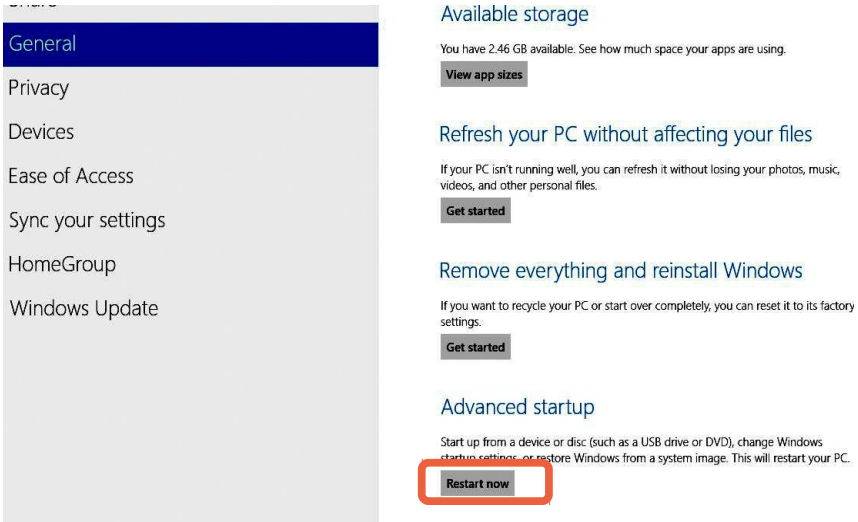


Figure 53: Under "Advanced Startup”, click "Restart now"

- The Windows 8 computer will now restart and you will be presented with a screen shown in Figure 56. Click on the "**Troubleshoot**" option.

- In the Troubleshoot screen that appears next, click on "**Advanced Options**" as shown in Figure 55.

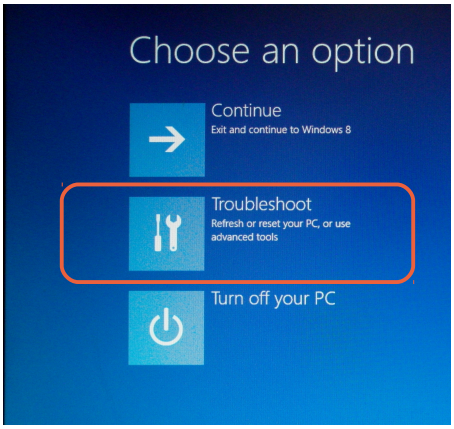


Figure 56: Click "Troubleshoot"

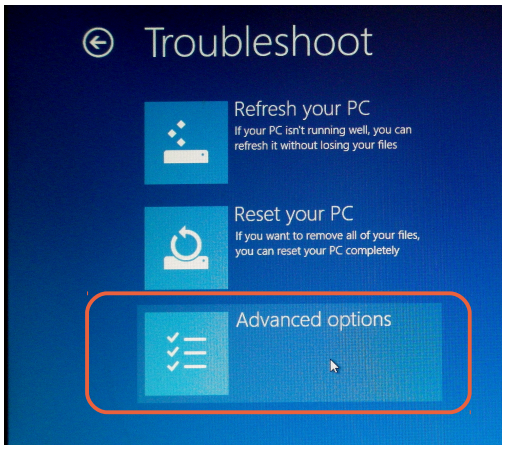


Figure 55: Click "Advanced Options"

Under the "**Advanced Options**" screen, click the "**Startup Settings**" choice as shown in Figure 57.

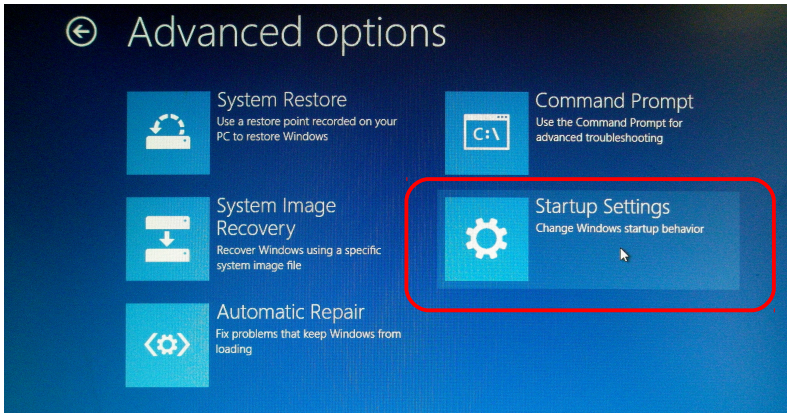


Figure 57: Click "Startup Settings"

In the "**Startup Settings**" screen (Figure 58), click "**Restart**".

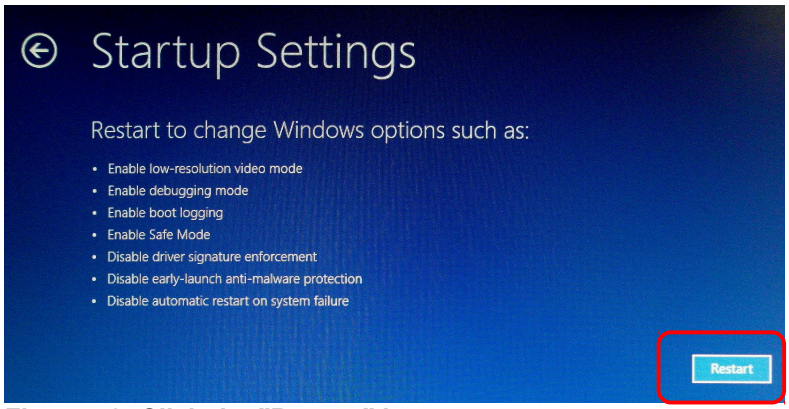


Figure 58: Click the "Restart" button.

- In the “Startup Settings” screen shown in Figure 59, you have 9 possible choices. The option we want to select is 7, labeled as “**Disable driver signature enforcement**”.

  Click either the **7** key on your keyboard or the **F7** function key and your Windows 8 computer will reboot again with driver signature enforcement now disabled.

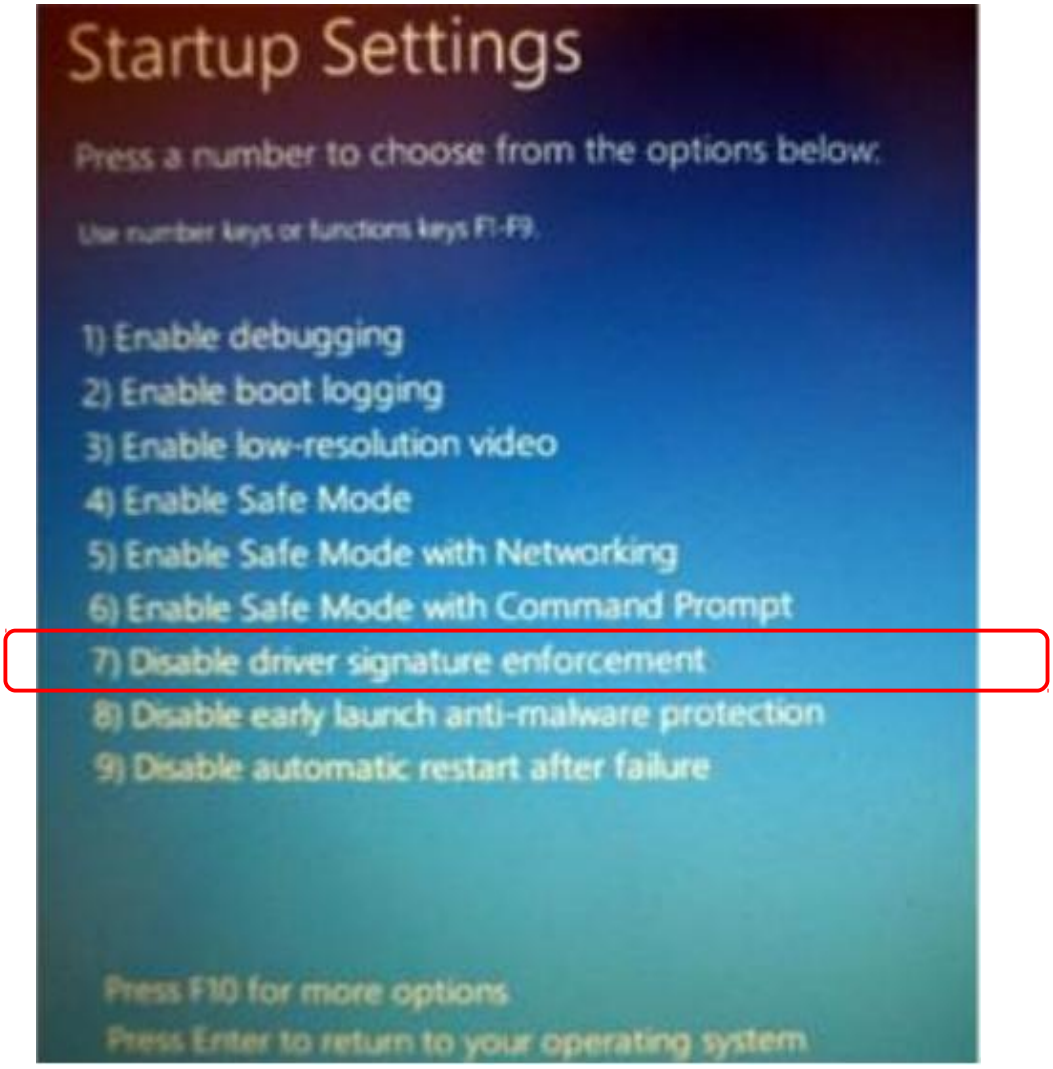


Figure 59: Click the 7 key or the F7 function key.

With that final Windows 8 reboot, you are good to go in terms of USB driver installation.

Note that you should install the CoLinkEx USB driver right now while the “signed driver enforcement” is disabled. If you restart your Windows 8 computer, the “signed driver enforcement” will be enabled again and you'll have to repeat the above steps.

Install the Latest CoLinkEx USB Drivers

Installing the CoLinkEx USB drivers is a two-step process: Download the USB driver components and then actually install the USB driver.

- **Download the USB Driver Components**

  Referring back to the CooCox web page for the CoLInkEx debugger (Figure 60), you must choose either the 32-bit USB driver or the 64-bit USB driver. For my Windows 8 setup, I choose to download the 64-bit version (Version 1.2.1 in Figure 60).

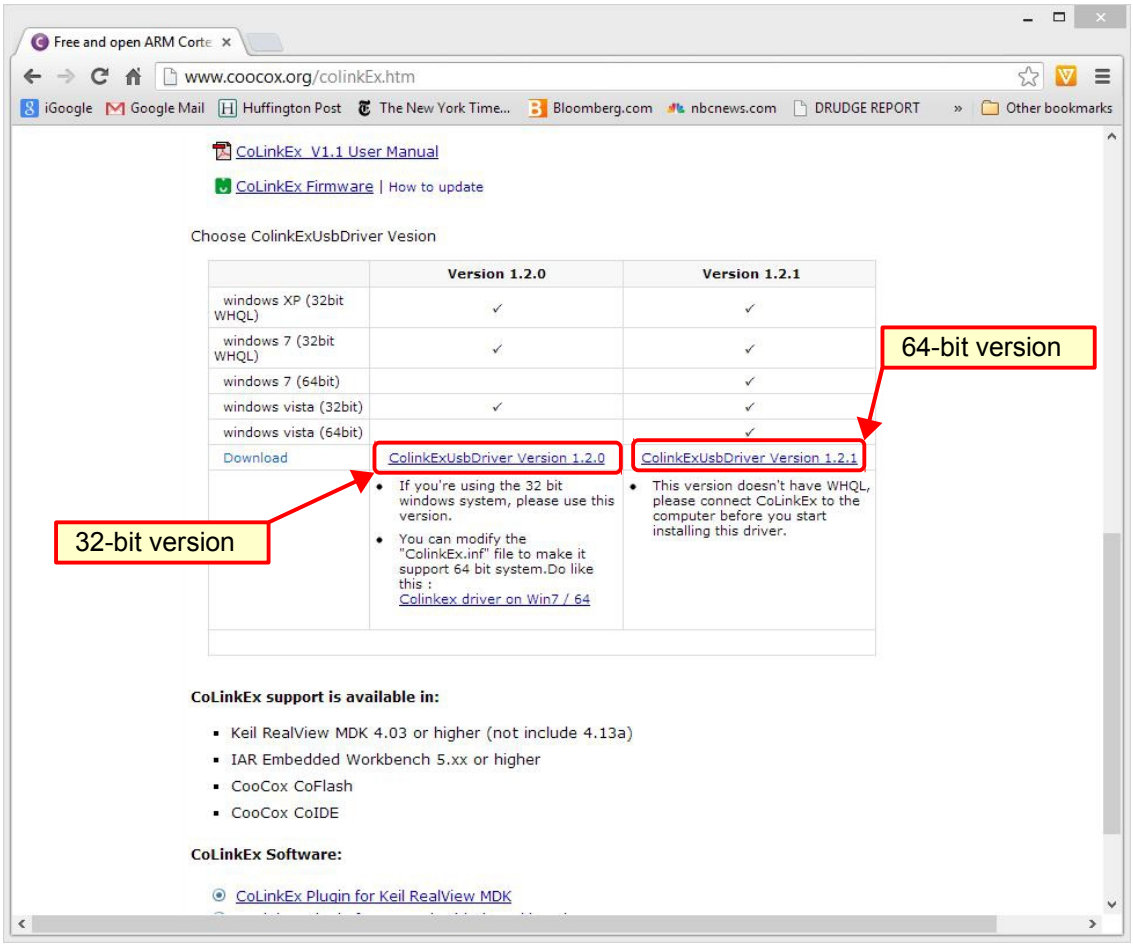


Figure 60: Download the 32-bit or the 64-bit USB Driver

Figure 61 shows the USB driver being downloaded.

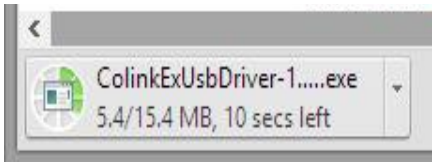


Figure 61: Downloading the USB Driver

When the USB driver has completed downloading, use the pull-down menu and click "**Open**" as shown in Figure 62 to initiate the installation. Remember that this only copies the needed software components to your hard drive; it does not install the USB driver (that comes next).

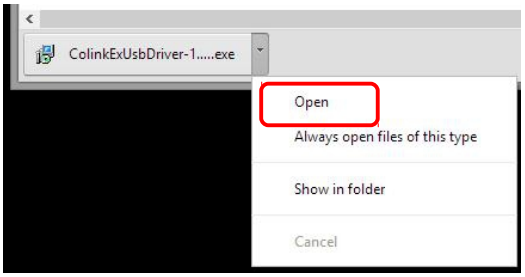


Figure 62: Click "Open" to commence installation

CooCox will present a “welcome” screen as shown in Figure 63. Click “**Next**” to continue.

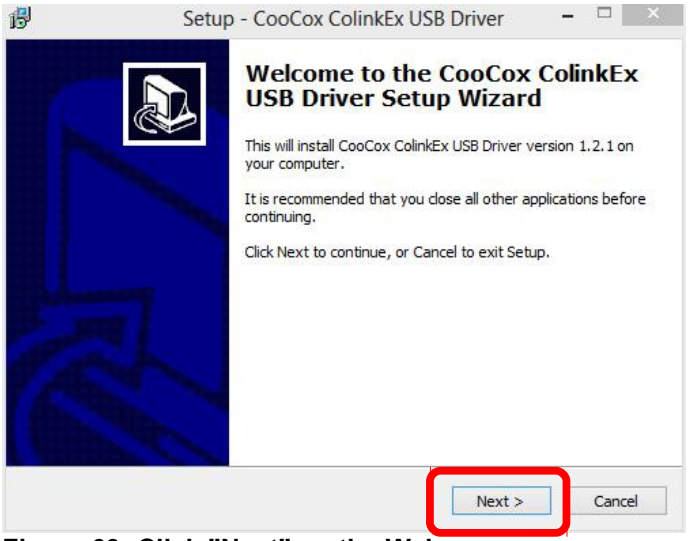


Figure 63: Click "Next" on the Welcome screen

Take the default on the Destination Location (Figure 64).

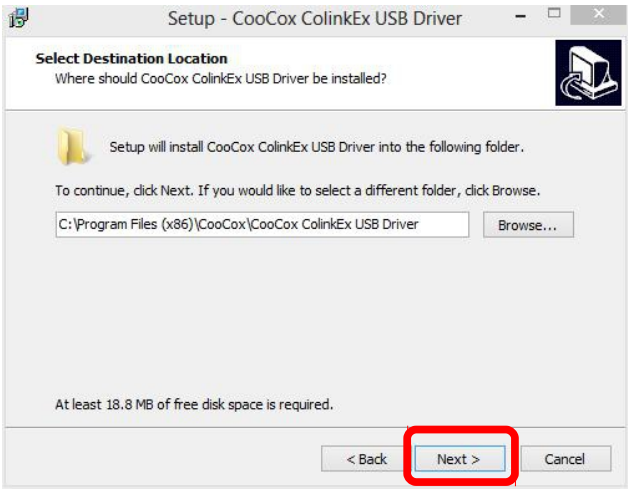


Figure 64: Take the default on the Destination Location

CooCox will give you one last look on the Destination Location. Click “**Install**” as shown in Figure 65 to commence copying of the driver components.

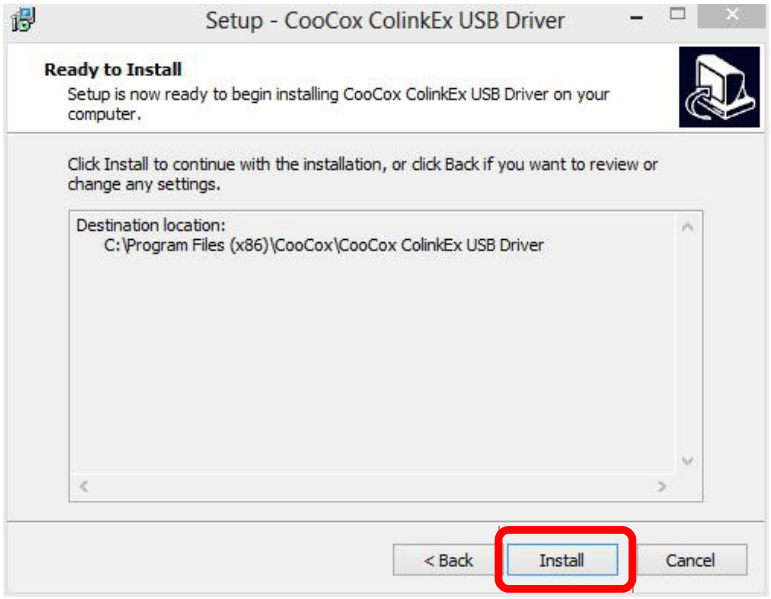


Figure 65: Click "Install" to Commence Installation

When the installation finishes, you'll get the window shown in Figure 66. Click “**Finish**” to complete the installation.

Figure 66: Click "Finish" to conclude installation.

- **CoLinkEx USB Driver Installation**

USB drivers are usually installed the very first time you plug the USB device in. Before you connect your CoLinkEx debugger to your computer's USB port, make sure the JP2 jumper is installed and the JP1 jumper is removed. Jumper JP2 passes the 3.3 volt Vcc supply to the target board.

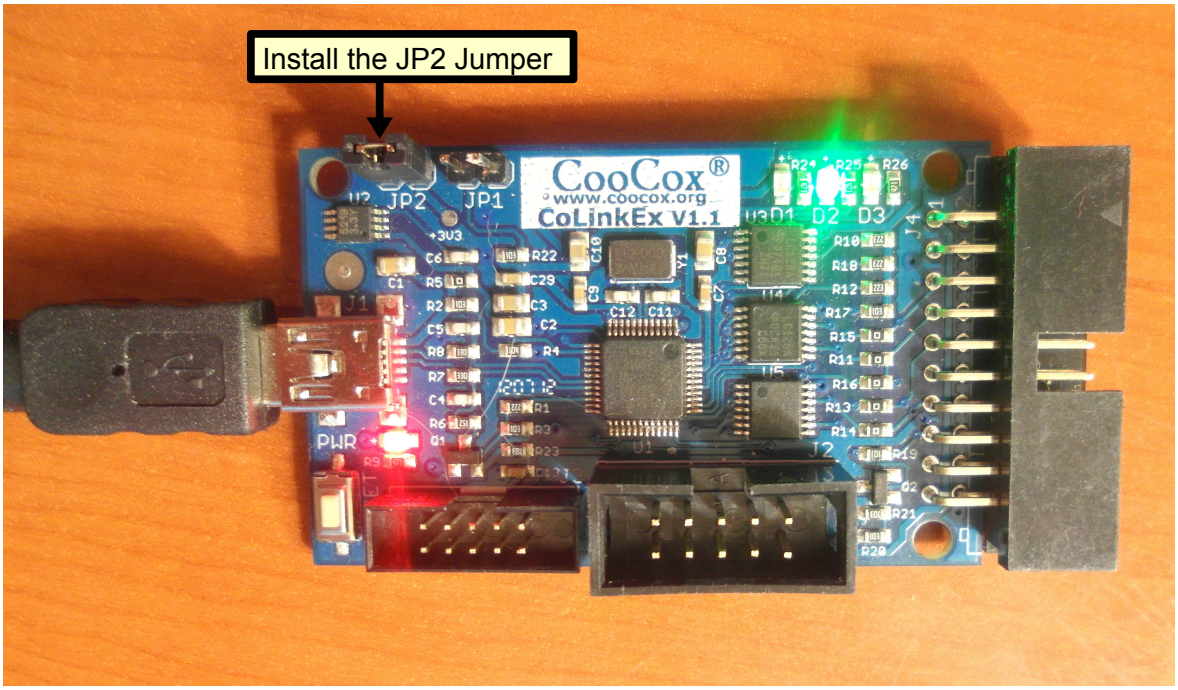


Figure 67: Install the JP2 jumper before connecting the CoLinkEx Debugger to your computer

Plug the CoLinkEx Debugger's USB cable into your computer. Hit the [**flag and x**] buttons simultaneously and select the **Device Manager** as shown in Figure 68. Note that CoLinkEx device appears in the "**Other Devices**" category. Right-click on the **COOCOX COLINKEX** device and select "**Properties**".

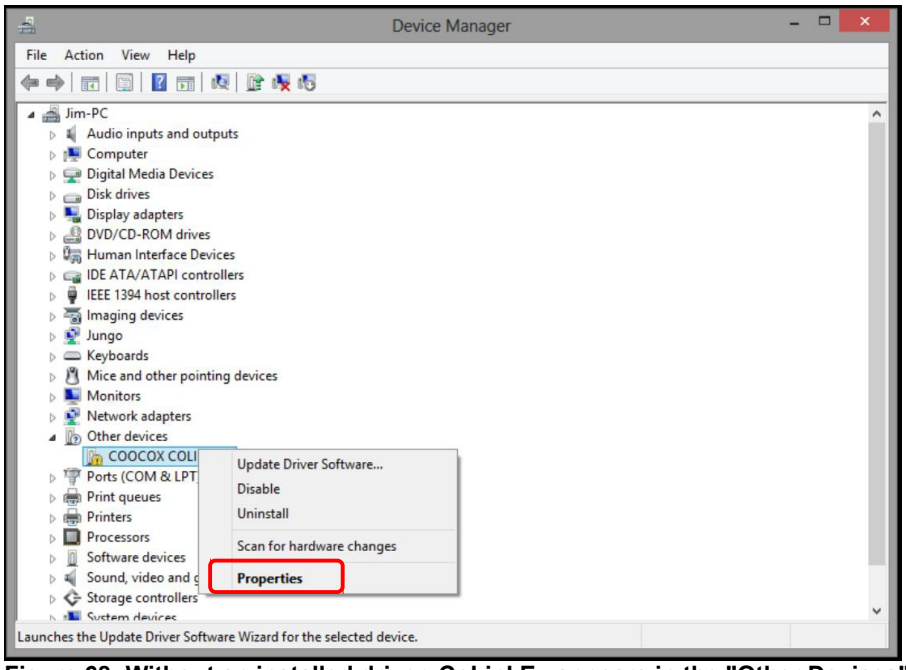


Figure 68: Without an installed driver, CoLinkEx appears in the "Other Devices" category

The properties display window (Figure 69) indicates that no device driver has been installed. Click on “**Update Driver**” to begin the installation.

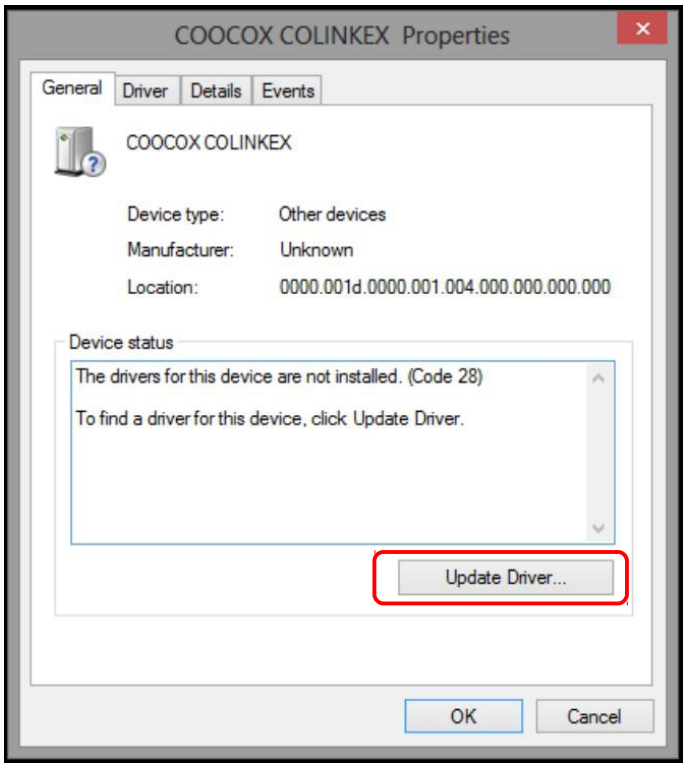


Figure 69: Click "Update Driver" to install the USB driver.

In Figure 70, click the “**Browse my computer..**.” option.

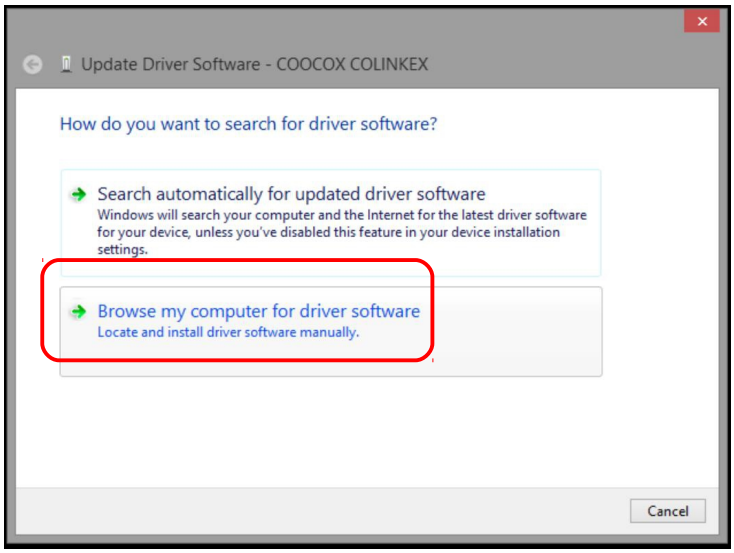


Figure 70: Select "Browse my Computer for driver software"

In Figure 71, click the “**Browse...**” button.

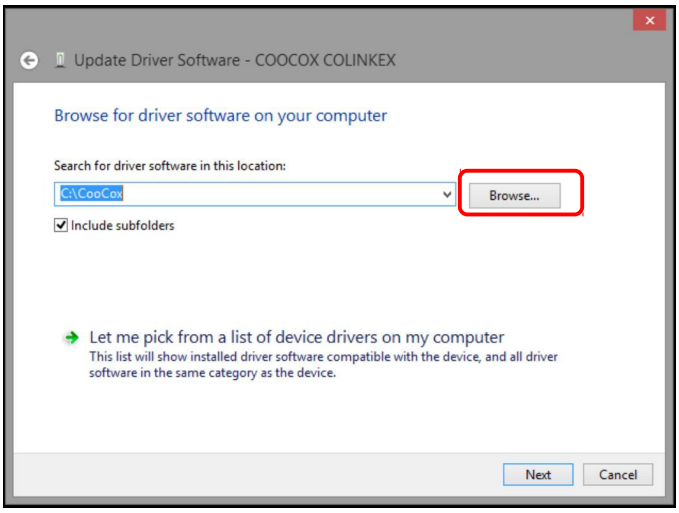


Figure 71: Click the "Browse" button.

In Figure 72 below, we have browsed to the location where the CooCox USB drivers can be found. Note that the location on my computer is “**C:\Program Files (x86)\CooCox\CooCox CoLinkEx USB Driver**”.

Make sure that the “Include Subfolders” check box is checked. Click “**Next**” in Figure 72 to proceed.

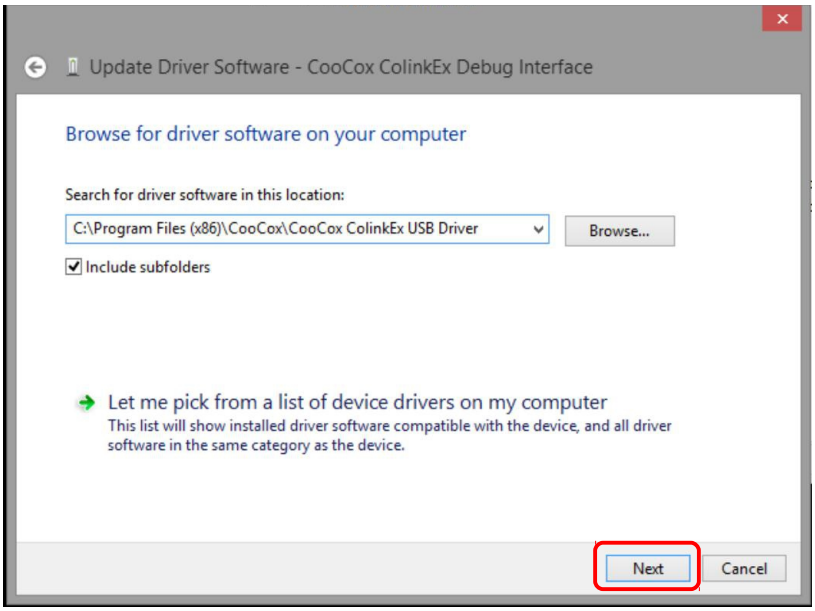


Figure 72: Click "Next" to start USB Driver Installation

The CoLinkEx USB driver installation will start, as shown in Figure 73 below.

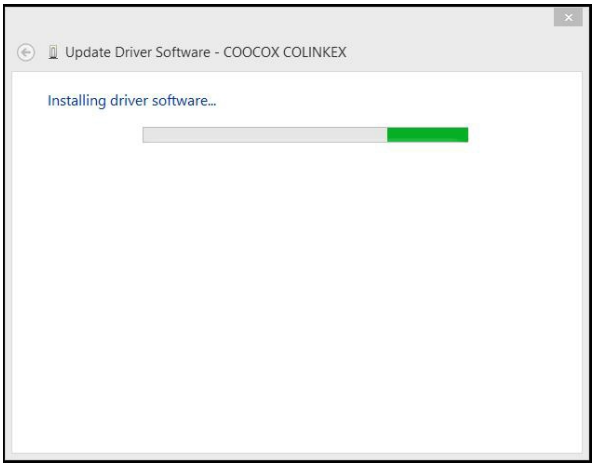


Figure 73: Driver installation in progress

At some point during the installation you may get a Windows Security warning as shown in Figure 74. Just click the "**Install this driver software anyway**" option to allow the driver installation to complete.

Figure 74: Click the "Install this driver software anyway" option

When the USB driver installation completes, you should get a report of success as shown in Figure 75 below. Click "**Close**" in Figure 75 to exit the installer.

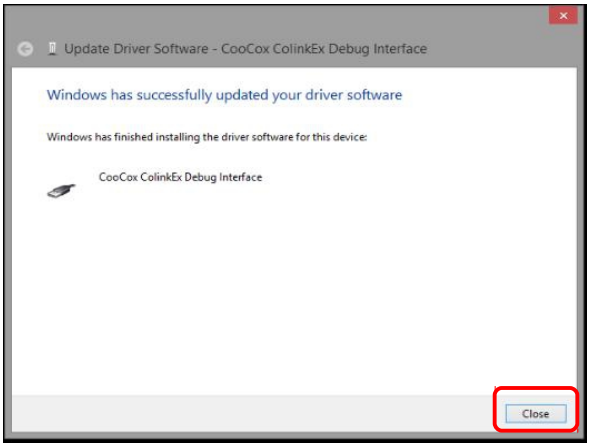


Figure 75: Driver installation successful

To double-check that our driver installation was successful, bring up the Device Manager again (hit the flag and x keys simultaneously and select Device Manager). Note that the CoLinkEx USB driver is now listed under the “Universal Serial Bus Controllers” heading (Figure 76).

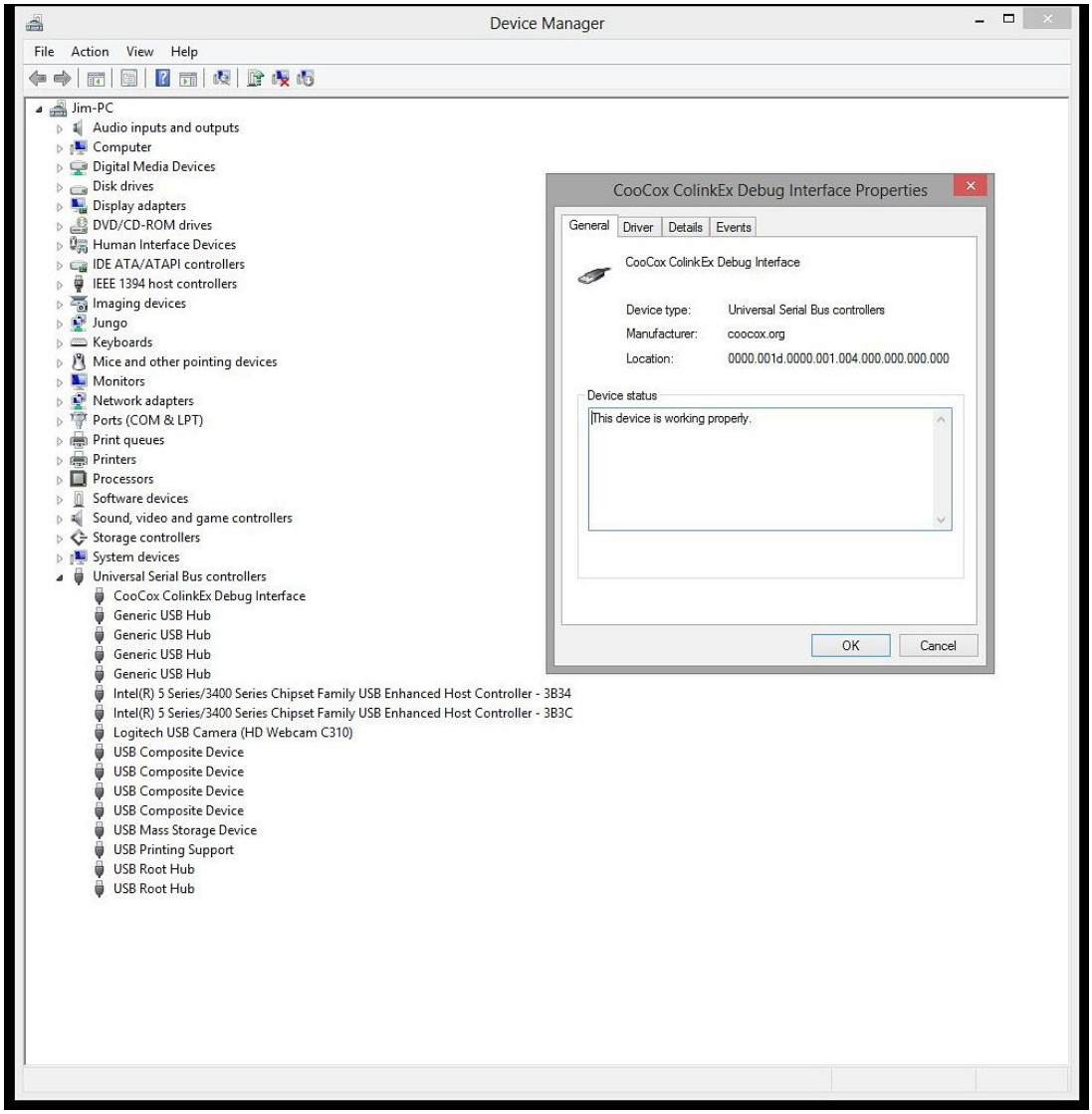


Figure 76: CoCox USB Driver is working properly

Building a SWD Cable

The CoLinkEx debugger board has three SWD/JTAG connectors; two 10-pin male connectors and one 20-pin male connector. We will use the larger 10-pin connector **J2** (10 pins (2x5) in 0.1"x0.1" spacing) as shown in Figure 77 below.

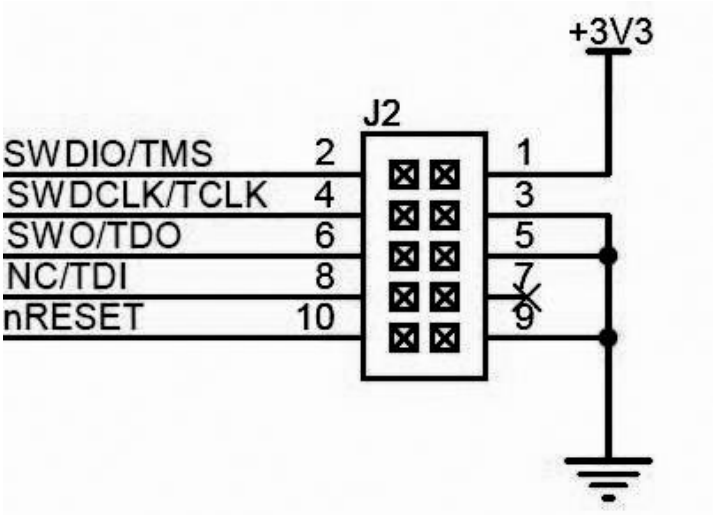


Figure 77: J2 Connector has the SWD Signals We Need

The actual alignment of this J2 connector (overhead view) is shown in Figure 79 and 78.

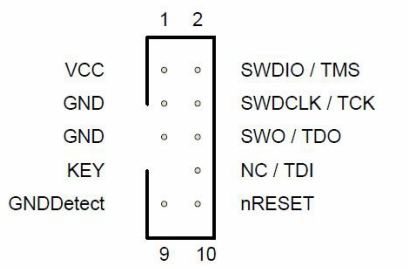


Figure 79: J2 SWD Connector

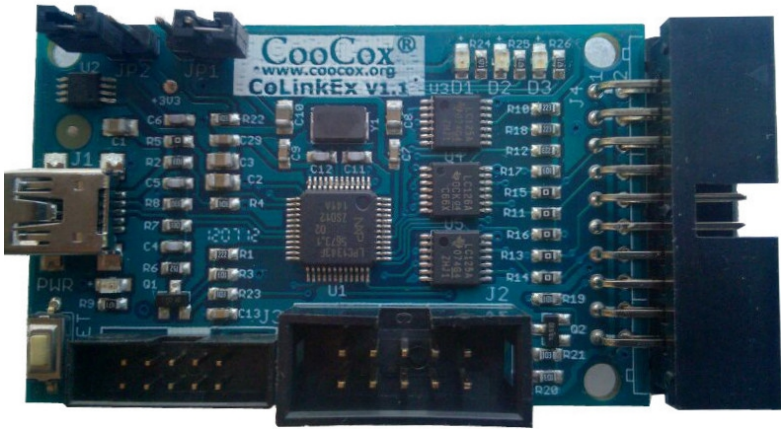


Figure 78: Overhead view of J2 Connector

On the LPC1769Xpresso board, the schematic fragment in Figure 80 shows the pin-out of the J4 board header. We will be connecting to pins 2,4,6,8,10,12,14, and 16 of the J4 header (the target side).

Note that if you have followed the instructions, the solder bridges labeled R59 through R66 will have been removed.

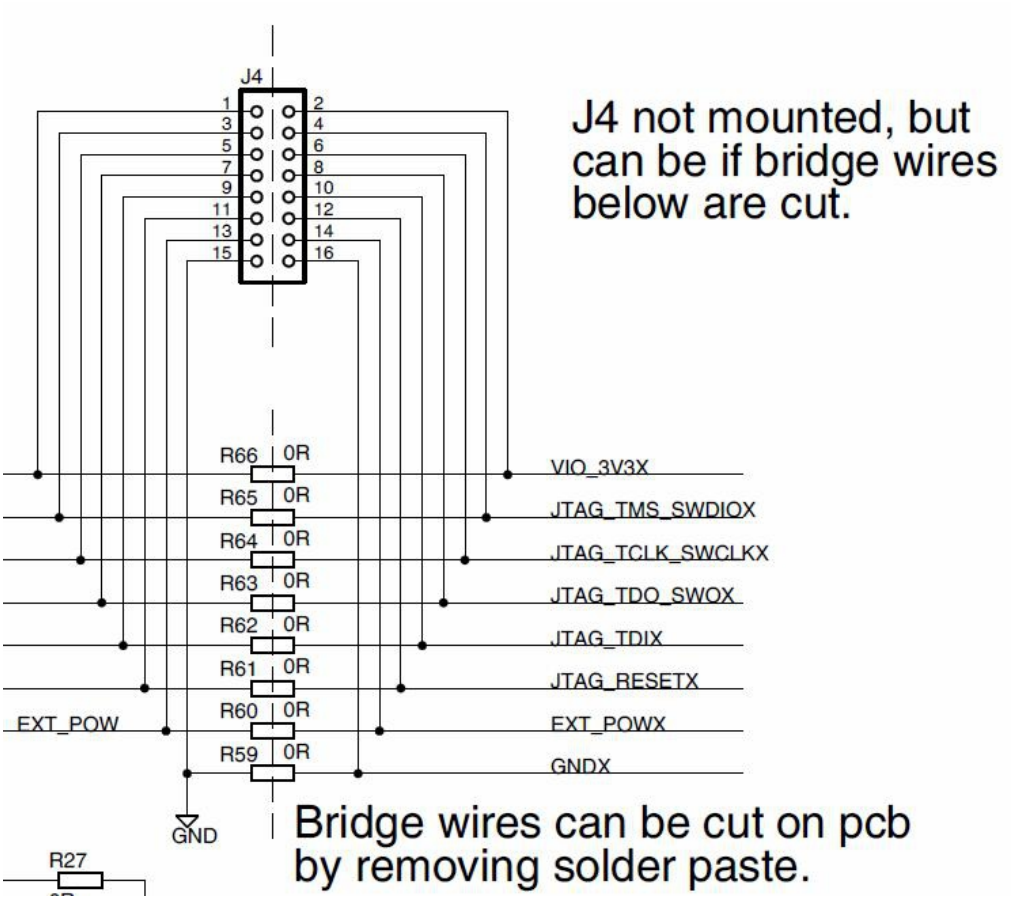


Figure 80: Partial schematic diagram of LPC1769 Xpresso board.

Based of the schematic diagrams above, the following connections are required:

Signals	CoLinkEx J2	LPC1769XPRESSO J4	Color
Vcc (3.3 volts)	1	2	Brown
SWDIO	2	4	Red
SWCLK	4	6	Yellow
SWO	6	8	Grey
GND	9	16	White
RESET	10	12	Black

Figure 81: SWD Cable Details

Based on the connection details shown in Figure 81, the cable can be assembled by using two cable assemblies from Adafruit, thus requiring no soldering or IDC crimping.

10-pin IDC Socket Rainbow Breakout Cable -
ID: 1199

$3.95

Qty.	Discount
1-9	$3.95
10-99	$3.56
100+	$3.16

Add to Cart: 1

ADD TO CART

96 IN STOCK

ADD TO WISHLIST

Figure 82: The individual .1" female sockets go to J4 on the LPC1769Xpresso Board

10-pin Socket/Socket IDC cable - Short 1.5" -
ID: 556

$1.50

Qty.	Discount
1-9	$1.50
10-99	$1.35
100+	$1.20

Add to Cart: 1

ADD TO CART

IN STOCK

ADD TO WISHLIST

Figure 83: This short cable mates to the J2 male connector on the CoLinkEx Debugger

By mating these two cables together, we have a ready-to-go cable assembly for driving the SWD debugging connector J4 on the LPC1769Xpresso board. Just follow the color coded wires as shown in Figure 81.

Figure 82 shows the completed assembly. Note that in this arrangement, the LPC-LINK half of the LPC1769 Xpresso board is not powered and inert.

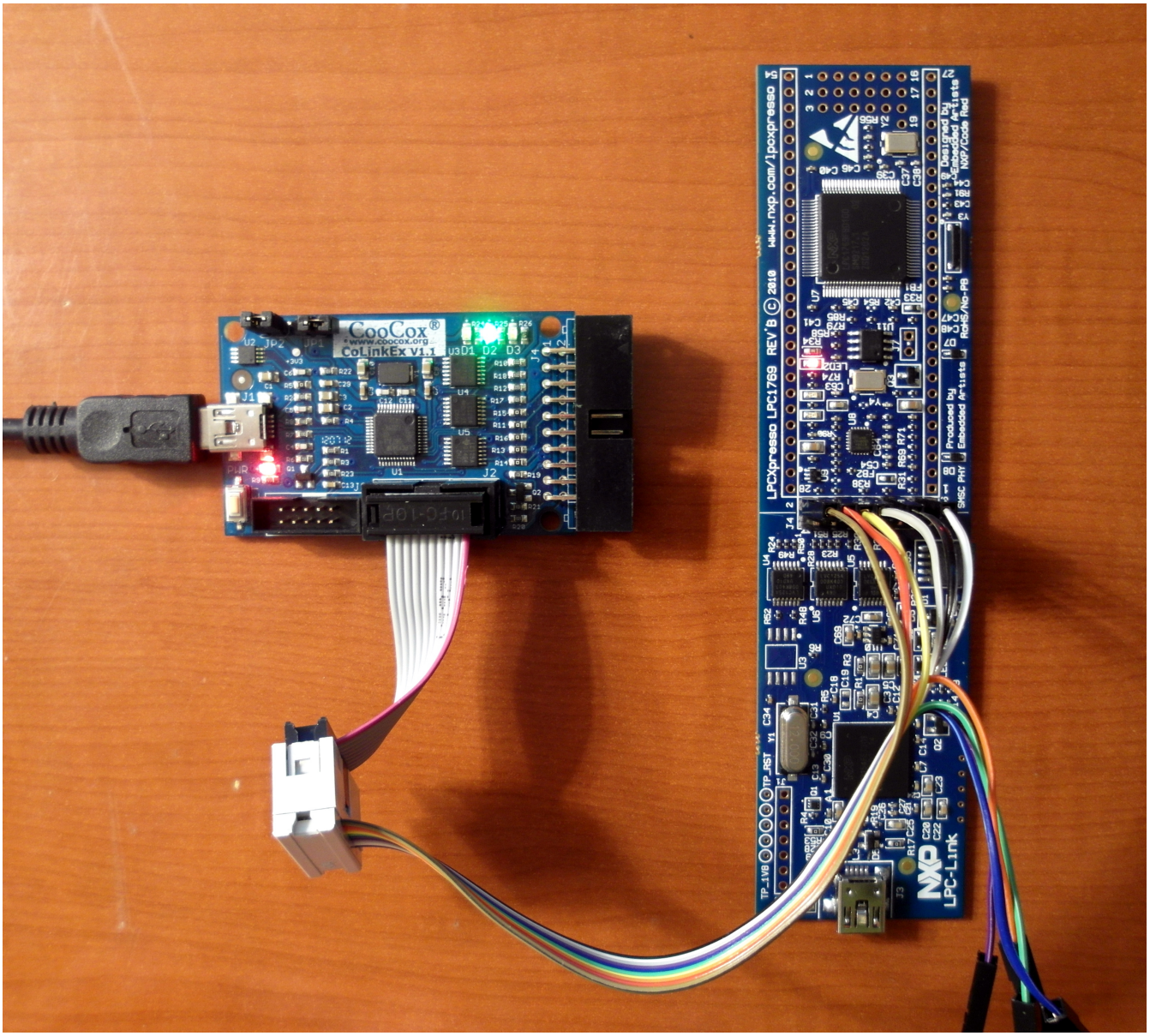


Figure 84: Simple SWD Cable connects CoLinkEx to LPC1769Xpresso Board

Create a Simple LPCXpresso Project

The very first novice project in the desktop computer arena is always the “Hello World” project where the aforementioned phrase is printed on the console screen using a simple print statement. In the micro-controller arena, the first novice project is invariably the “Blinker” project where a LED mounted on the board is flashed on and off in an endless loop.

The LPCXpresso board (target side) has a user-writable red LED2 shown in Figure 85. This LED is connected to a digital input/output pin, specifically Port 0 – pin 22.

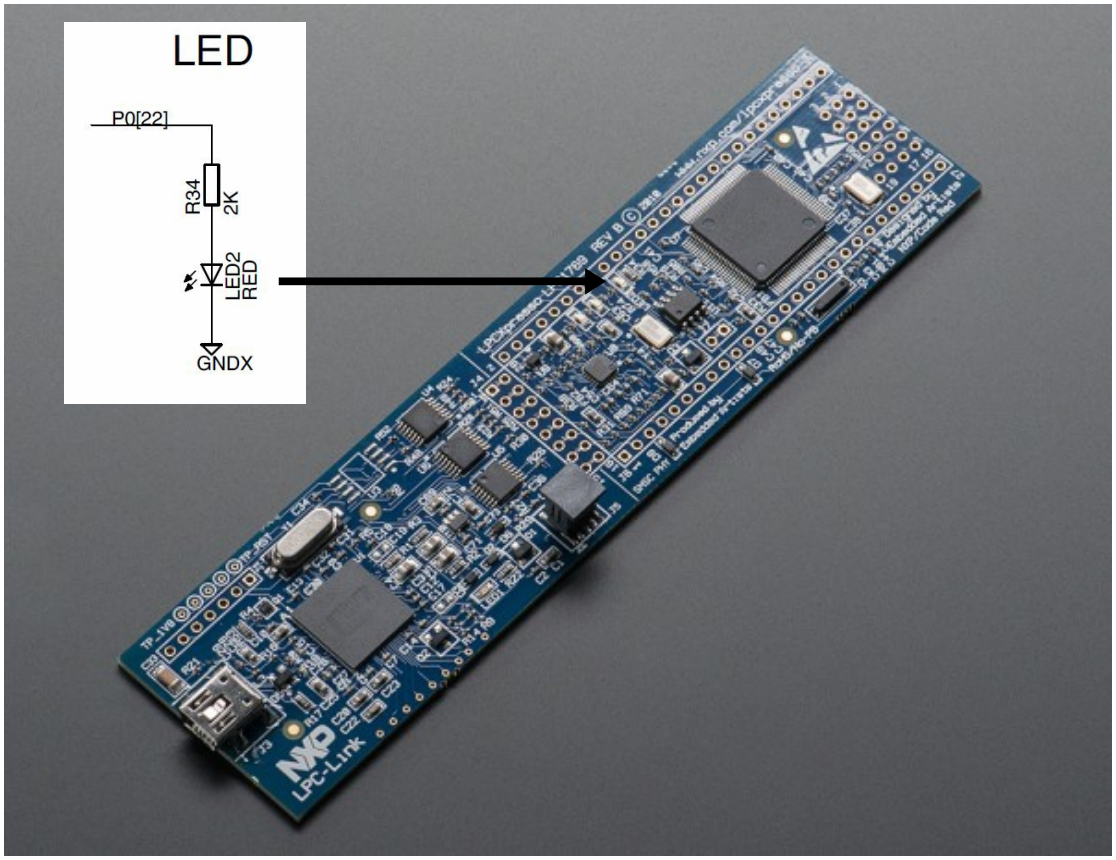


Figure 85: LPCXpresso Board has a User-writable LED (P0.22)

To create a project, first click on the COIDE icon on your desktop, as shown in Figure 86.

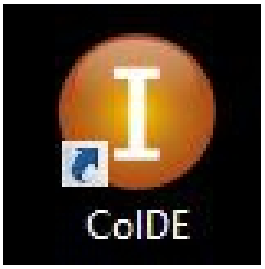


Figure 86: Click CoIDE icon to start

The COIDE Eclipse main window will display, shown in Figure 87. Note it has the usual Windows pull-down menu bar, a toolbar, a status line at the bottom and three perspectives (sub-windows) in the center of the screen; namely the Project, Welcome, and Console views.

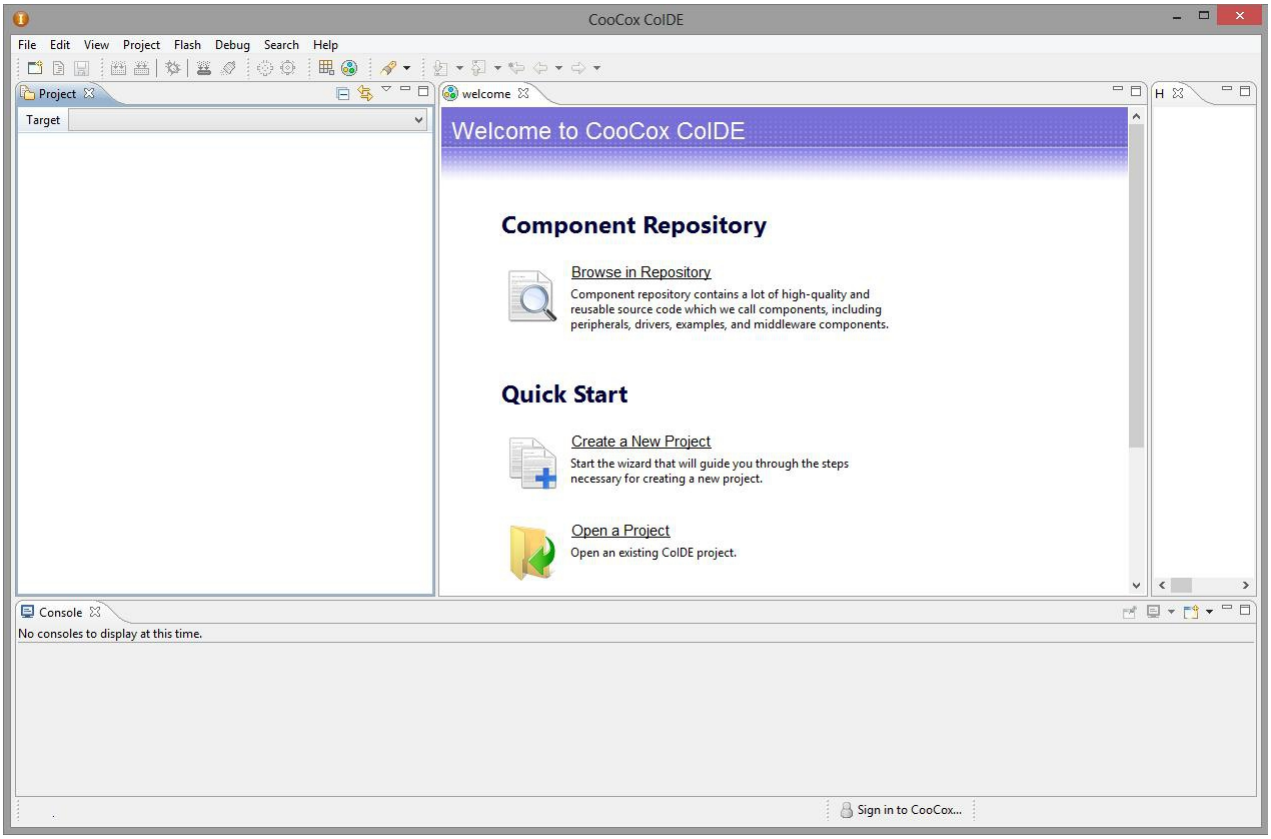


Figure 87: CooCox IDE Start-up Display

To create a project, select "**New Project**" from the **Project** pull-down menu (Figure 88).

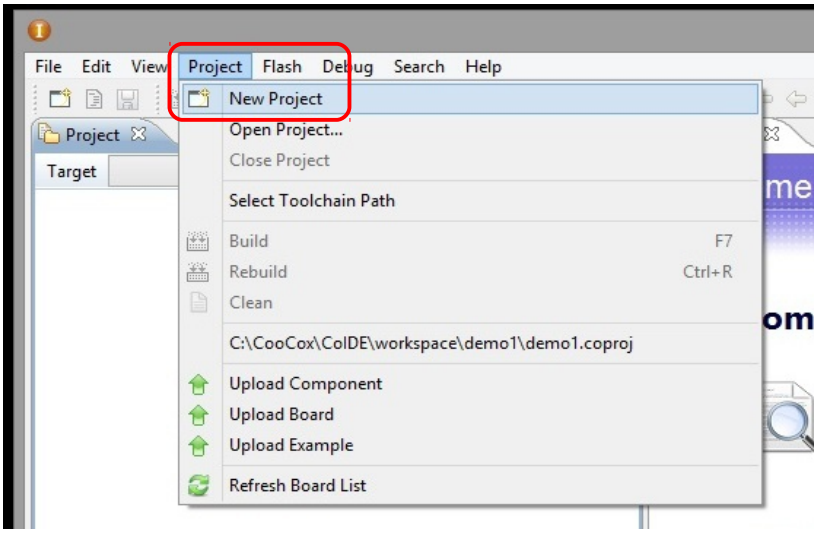


Figure 88: To start a new project, click Project – New Project

In the Project window that pops up (Figure 89), enter a project name such as "**demo1**" and click "**Next**" to continue. Note that the path to the workspace containing the project is identified in this example as:

C:\CooCox\CoIDE\workspace

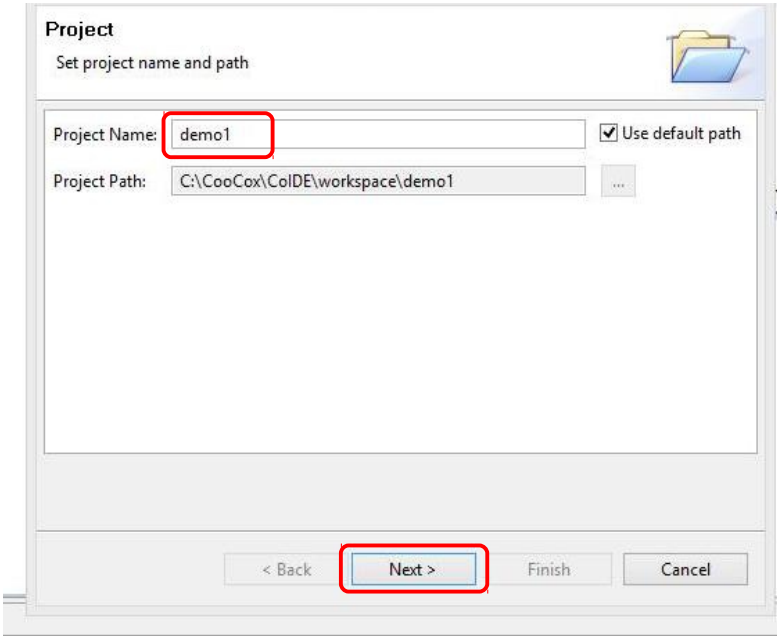


Figure 89: Creating the Project Name

Now CooCox can create a project either tailored to a particular Arm-Cortex microprocessor chip or tailored to a specific evaluation board. In this simple example, we will be tailoring the project to a particular chip: the **NXP LPC1769** device. As shown in Figure 90, click on the chip icon and click "**Next**" to continue.

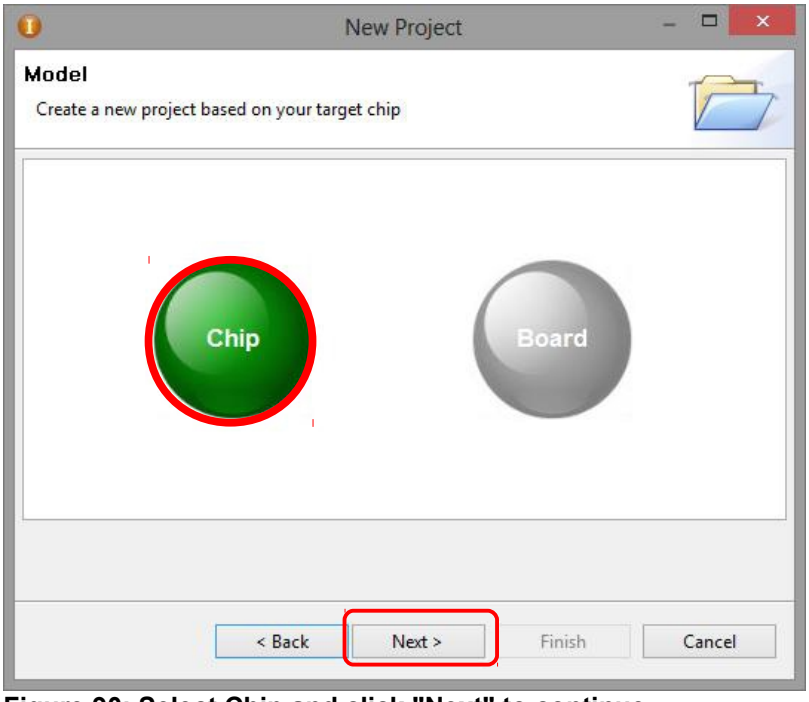


Figure 90: Select Chip and click "Next" to continue

First a list of manufacturers is displayed (Figure 92), click on **NXP**. This will automatically display a list of NXP ARM-Cortex chips (Figure 91). Click on **LPC1769**.

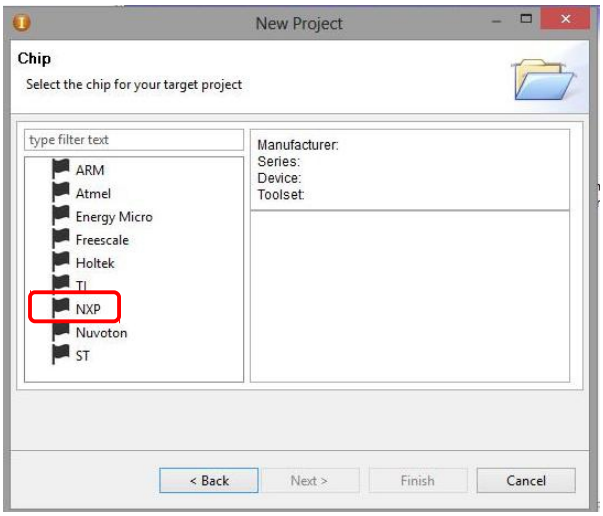


Figure 92: Select the manufacturer: NXP

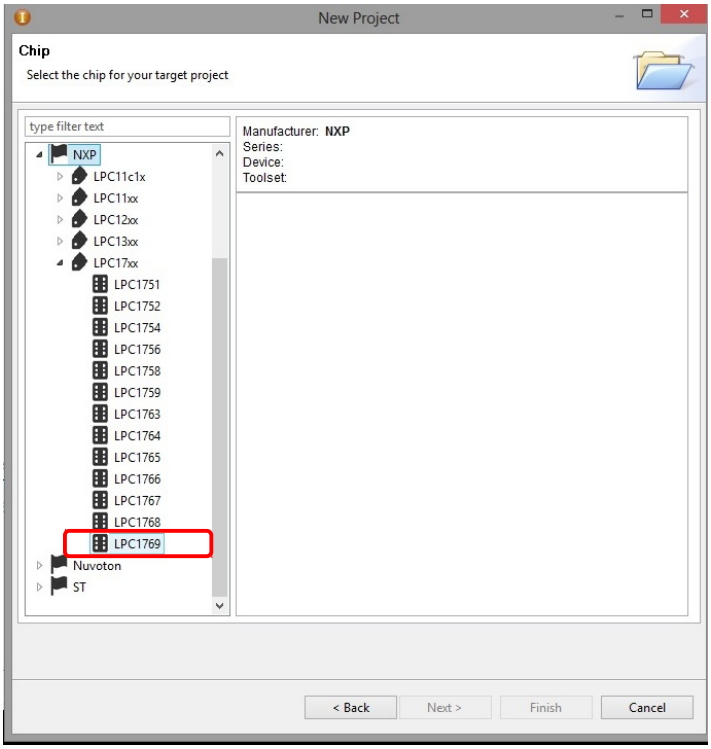


Figure 91: Select the Chip model: LPC1769

Some details about the LPC1769 chip will be displayed (Figure 93). Click on "**Finish**" to complete creation of a template project for the LPC1769 ARM-Cortex chip.

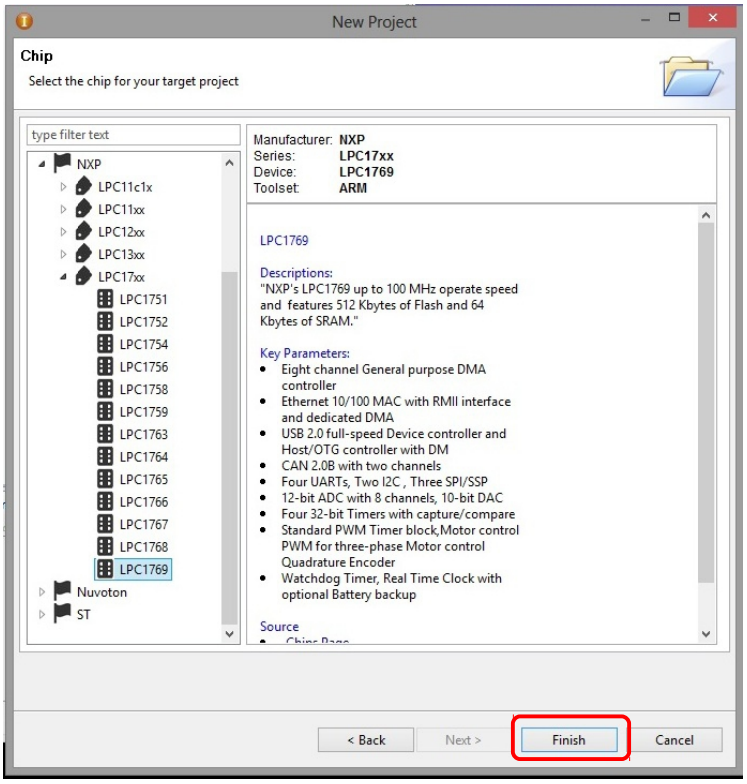


Figure 93: Click "Finish" to create a template project

CoIDE will now display an actual project; named demo1 and containing only one file (main.c). Figure 94 shows this starter project. Also, a list of software components called the **Repository** will appear.

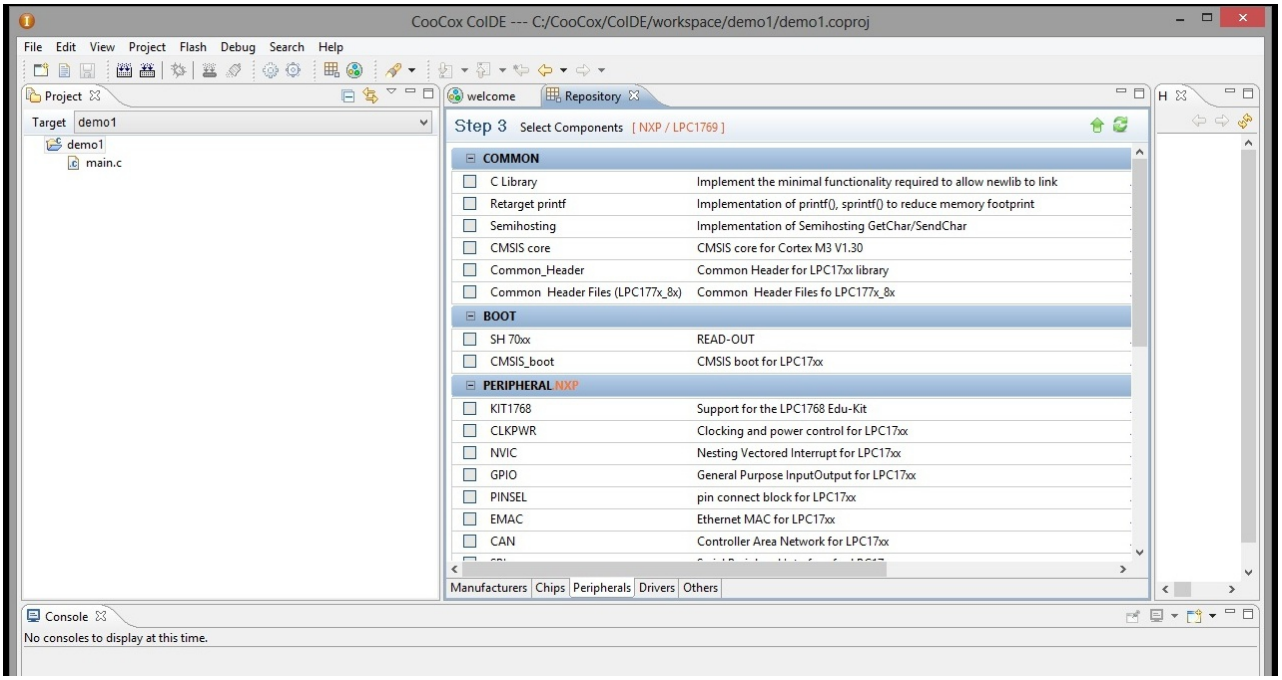


Figure 94: demo1 project with only one file.

What CoIDE does next is really quite wonderful. We know that we want to blink that LED and it is connected to a digital input/output pin. So it is obvious that we need to check the **GPIO** box in Figure 94. GPIO stands for **G**eneral **P**urpose **I**nput **O**utput peripherals.

When we do that, CoIDE automatically checks and thereby includes the following necessary software packages:

Software package	Function	
CMSIS Core	CMSIS core for Cortex M3 V1.30	(supplied by ARM)
Common_Header	Common Header for LPC17xx library	(supplied by NXP)
CMSIS_boot	Start up functions and data structures	(supplied by ARM and NXP)
CLKPWR	Functions to set up the chip's clock circuits	(supplied by NXP)

There has been several mentions of something called CMSIS. This stands for **C**ortex **M**icrocontroller **S**oftware **I**nterface **S**tandard and is defined by ARM for the ARM-Cortex chips. This CMSIS standardizes the interrupt vector tables, peripheral device register names, memory addresses of all peripherals, and so forth.

Getting a bare-metal microprocessor application to do anything used to be a very difficult task. With this included CMSIS and vendor-supplied device drivers, starting your project by including these things means that you can “jump to main()” and get your application going in minutes rather than weeks.

In Figure 95 below, clicking the **GPIO** check box automatically includes all the necessary CMSIS and device driver software components to create a fully functional project that can be compiled, linked, and downloaded to flash memory and executed. It wouldn't do very much, you need to add some statements to blink the LED2.

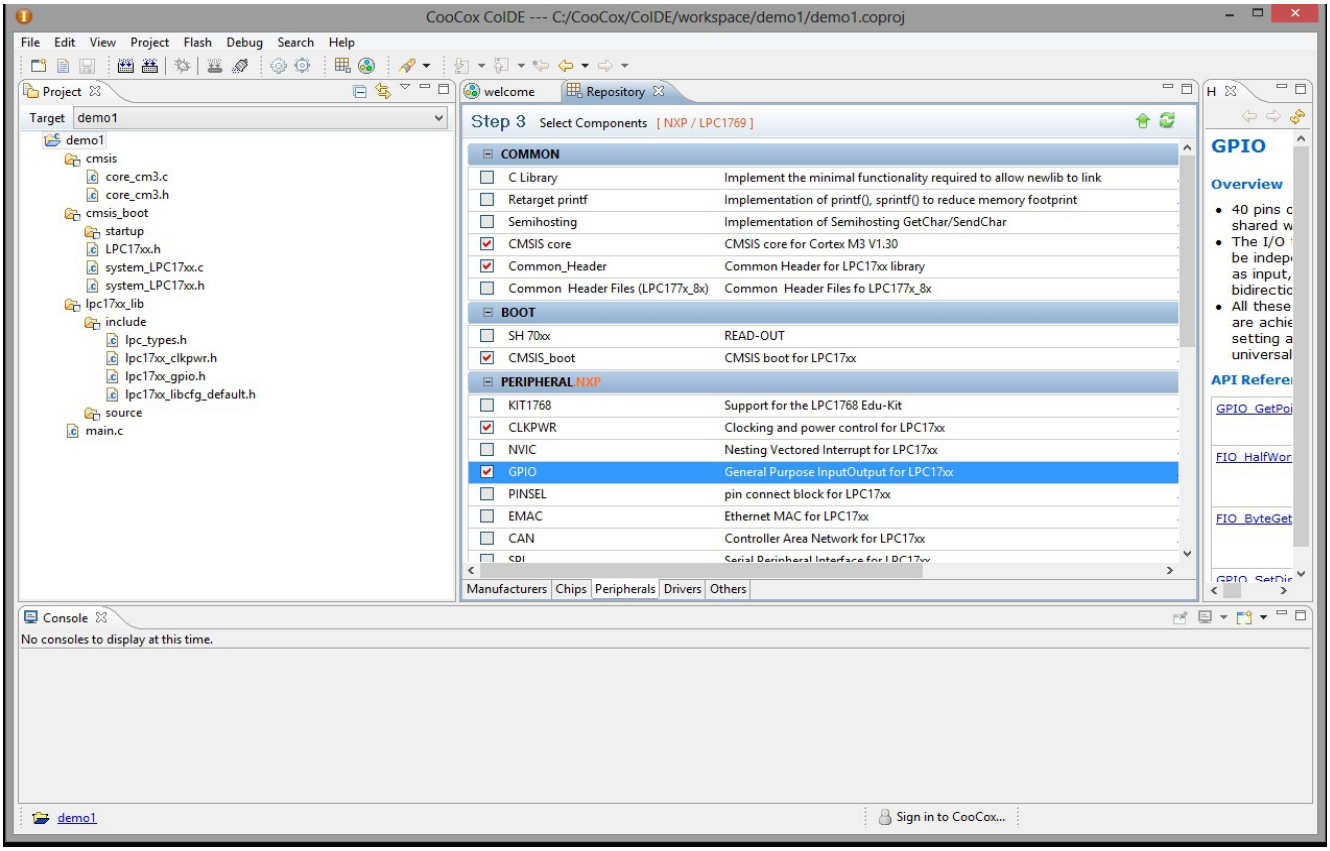


Figure 95: Selecting GPIO will automatically select CMSIS core, Common_Header, CMSIS_Boot and CLKPWR

Adding an Example to the Project

Now we will make use of CoIDE “examples” to get a blink LED code template. Bring up the Components view by clicking “**View**” followed by “**Components**” as shown in Figure 96.

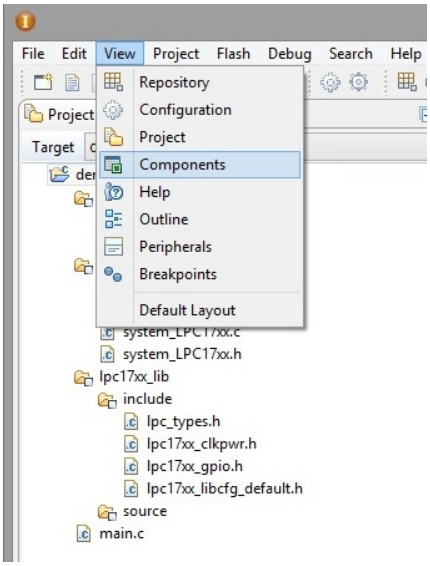


Figure 96: Add the Components view to the Editing Area

The Components view will be added to the Project view on the far left as shown in Figure 97. Notice that this is just a list of the software components currently selected for inclusion into the project.

What's more interesting is that the GPIO component has a link you can click to see an example.

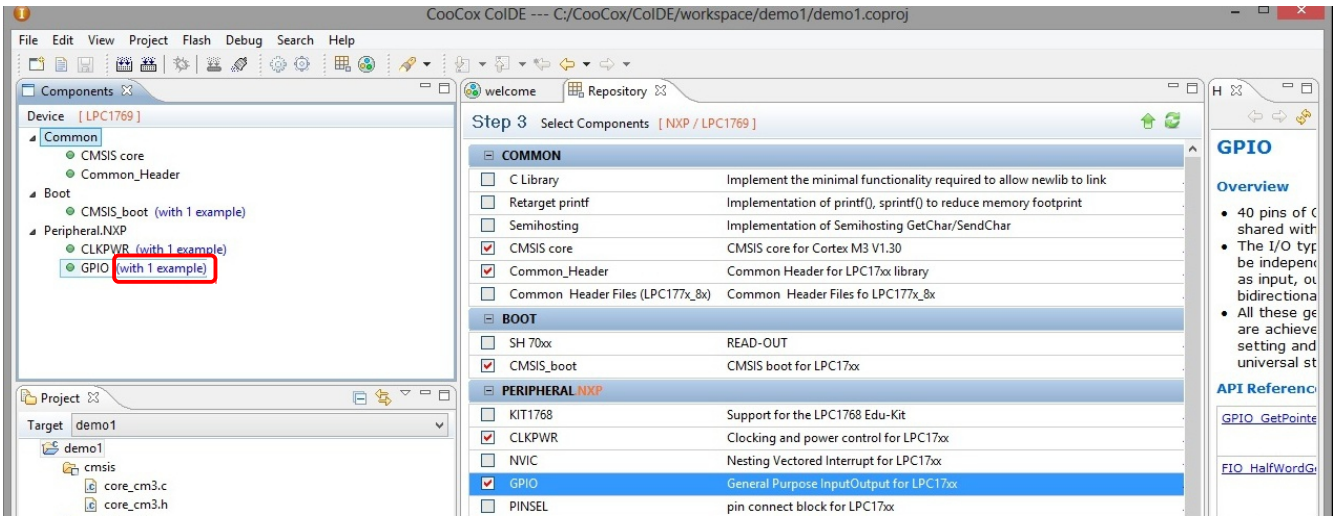


Figure 97: Components View lists the software packages currently selected

If you click on the GPIO example in Figure 97 above, a new addition to the editing area describes the available example (Figure 98).

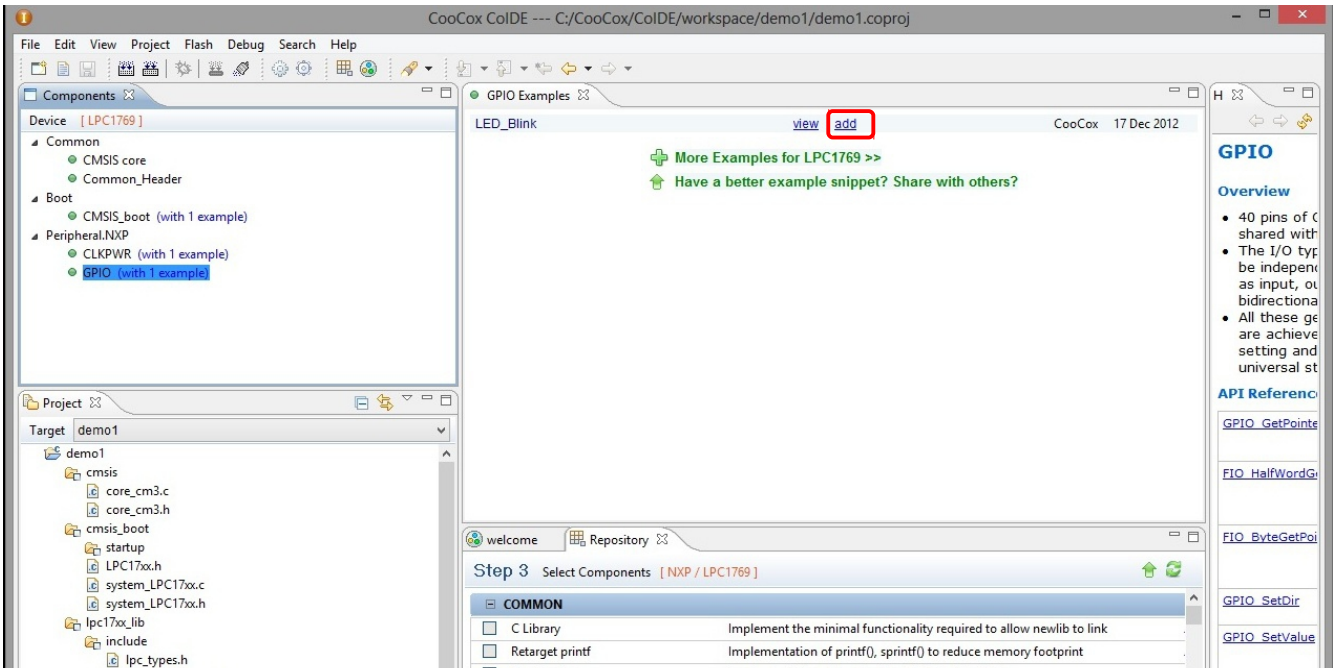


Figure 98: GPIO Examples include a LED_Blink code sample

In the GPIO Examples window, you can view the proposed code or add it to the project. As shown in Figure 98, click the **ADD** link to add the **LED_Blink** code to the project. A confirmation text box will appear (Figure 99), click "**Yes**" to confirm adding the Blink_LED example to the project.

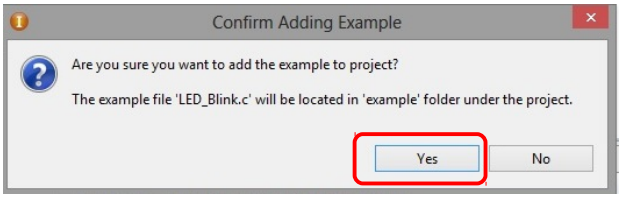


Figure 99: Click "Yes" to add the example

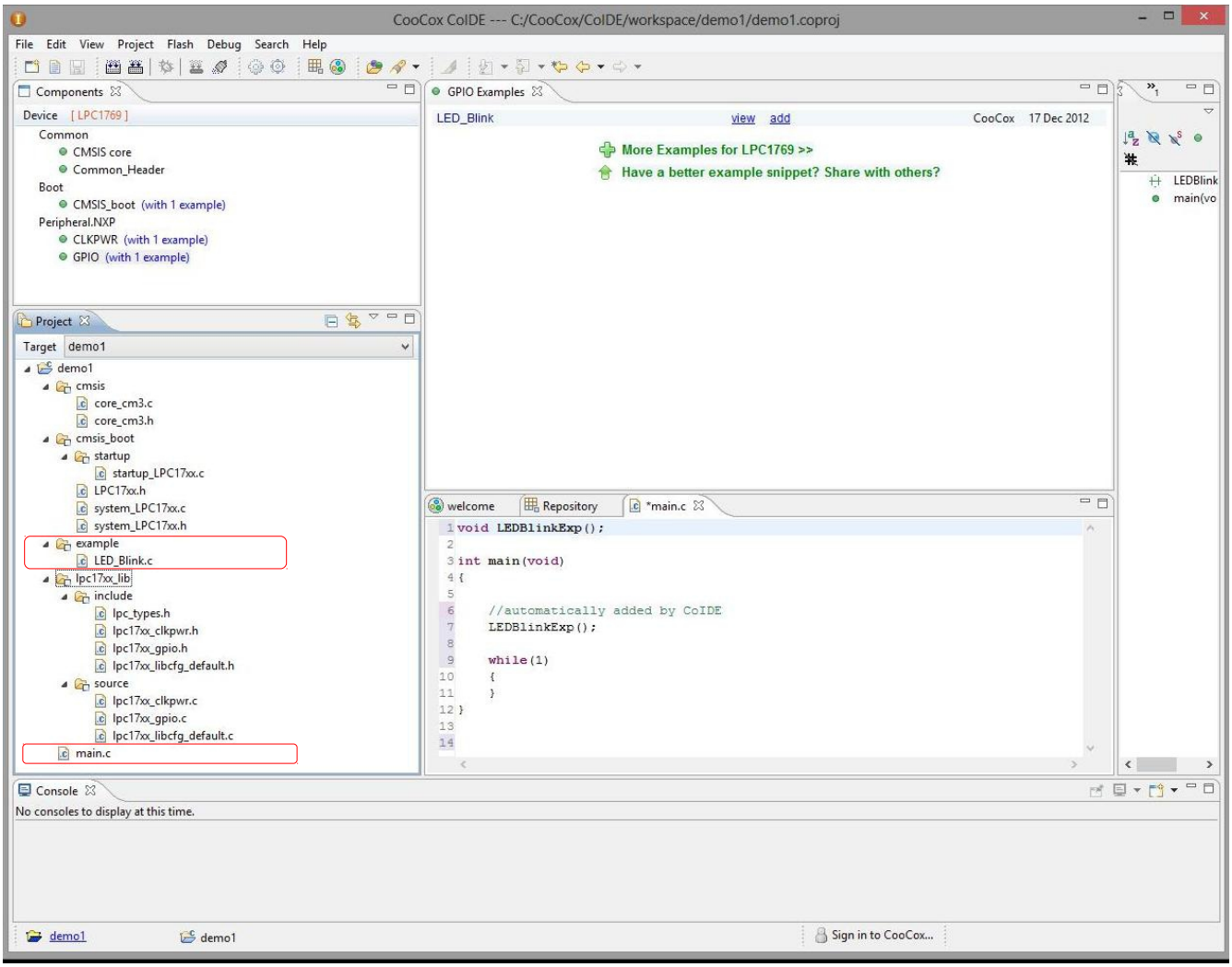


Figure 100: LED_Blink Example added to the Project

Observe in Figure 100 the two things have changed in the Project view: a new folder called "**example**" has been added and within that folder is a LED blinker C function called **LED_Blink.c**.

The main function has been replaced with one that prototypes and calls the blinker C function **LEDBlinkExp** (Figure 101).

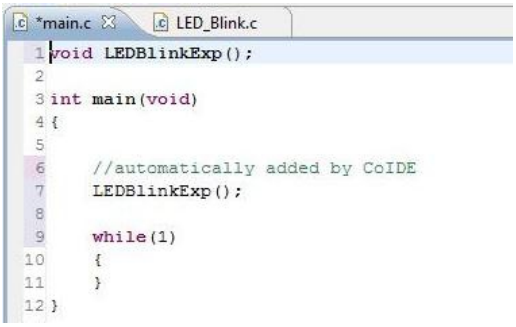


Figure 101: Main Program has Blinker Call

If the blinker source file **LED_Blink()** is not displayed in the center editing area, bring it up by clicking on that function in the Project view on the far left (see Figure 102 below).

Before you get too excited by all this software automation, be forewarned that the chances are high that the LED Blinker example will not work without some modification. This example LED_Blink() was probably created for some other ARM-Cortex microprocessor or some other evaluation board.

Remember that our LED on the LPCXpresso board is **P0.22** (Port 0, bit 22). Figure 102 shows the LEDBlinkExp() function as supplied by CoIDE. The file **lpc17xx_gpio.c** contains sufficient documentation to enable you to understand each GPIO function's parameters and return values.

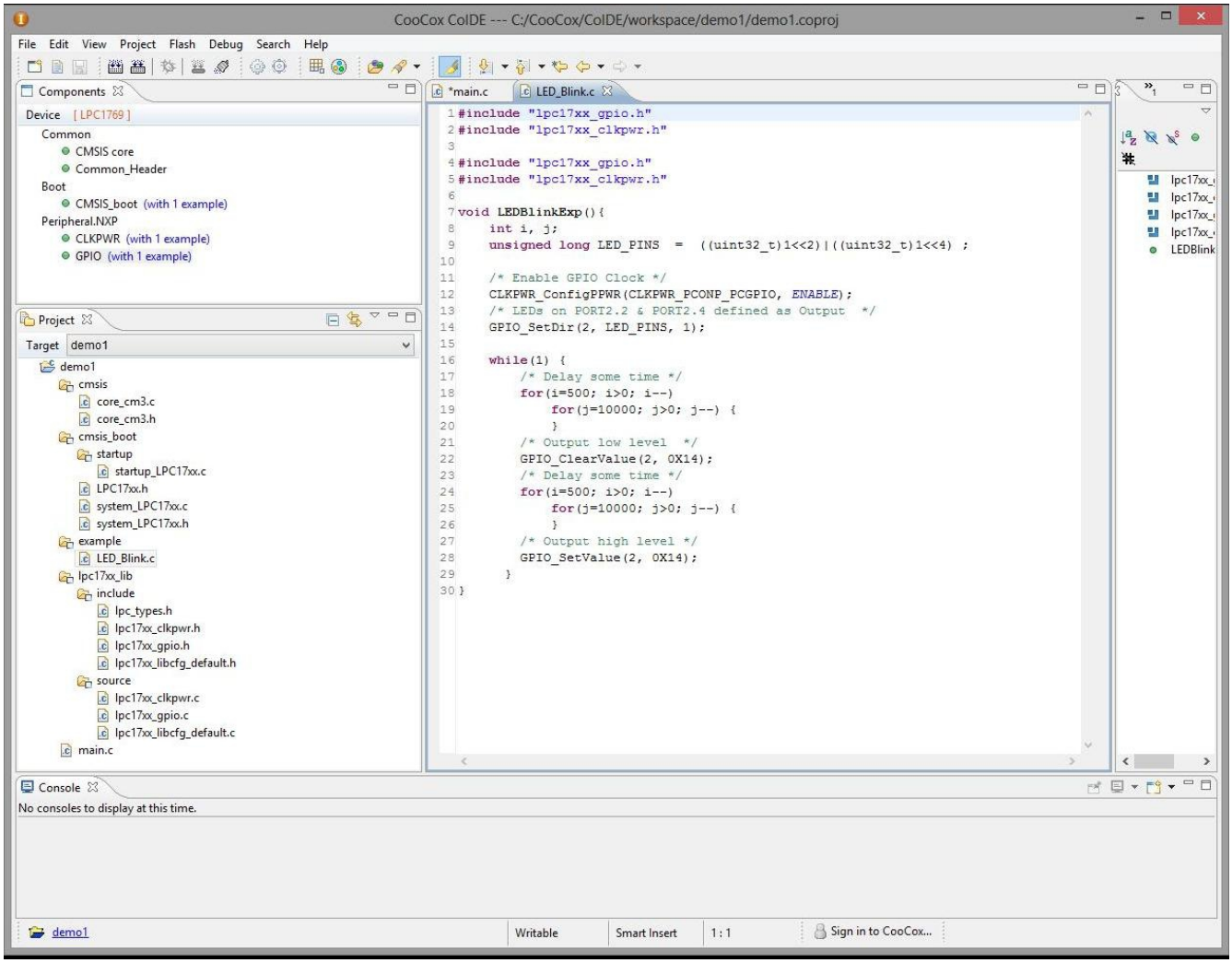


Figure 102: The example function LEDBlinkExp() will not work

Fixing this up to work with the LPCXpresso board is straightforward. This example was set up to blink two LEDs P2.2 and P2.4 – our LPCXpresso board has a single LED at P0.22.

Studying the example blinker function, we can spot several required changes. First, the #includes have been unnecessarily repeated (delete one pair). The mask LED_PINS should be a constant since that puts it into flash EPROM rather than precious RAM memory. Also, the mask should be modified to access only one I/O bit (I/O bit 22). The setting of the I/O port direction (output to drive an LED) needs to specify GPIO port 0 and bit 22. For setting and clearing GPIO port 0 - bit 22, the GPIO port should be changed to 0 and the constant LED_PINS substituted for better readability.

The following is a summary of the changes made to tailor the example to the LPCXpresso board.

1. **The redundant #includes (lines 1 - 5)**

```
#include "lpc17xx_gpio.h"
#include "lpc17xx_clkpwr.h"

#include "lpc17xx_gpio.h"                    /* unnecessary                    */
#include "lpc17xx_clkpwr.h"                  /* delete both redundant #includes */
```

2. **The constant LED_PINS (line 9)**

```
unsigned long LED_PINS =    ((uint32_t)1<<2) | ((uint32_t)1<<4);   /* incorrect, masked for bits 2 and 4 */

const unsigned long LED_PINS = ((uint32_t)1<<22);                  /*  correct, use const and mask for just bit 22  */
```

3. **Setting the Port Direction (lines 13 - 14)**

```
/*  LEDs on P2.2 & P2.4  defined as Output  */      /* incorrect, comment calls out two pins  */

GPIO_SetDir(2, LED_PINS, 1);                       /*  incorrect, port is 2  */

/* LED on PORT0.22 defined as Output  */           /* correct, comment refers to P0.22  */

GPIO_SetDir(0, LED_PINS, 1);                       /*  correct, port is now 0  */
```

4. **Clearing the LED I/O Bit (line 22)**

```
GPIO_ClearValue(2, 0x14);                          /* incorrect, port is 2 and two bits are cleared */

GPIO_ClearValue(0, LED_PINS);                      /* correct, port is 0 and bits defined by constant */
```

5. **Setting the LED I/O Bit (line 28)**

```
GPIO_SetValue(2, 0x14);                            /* incorrect, port is 2 and two bits are set */

GPIO_SetValue(0, LED_PINS);                        /* correct, port is 0 and bits defined by constant */
```

Editing these changes into the **LED_Blink.c** function in Figure 102 will give the revised file shown in Figure 103 below.

main.c | *LED_Blink.c

```
1 #include "lpc17xx_gpio.h"
2 #include "lpc17xx_clkpwr.h"
3
4 void LEDBlinkExp(){
5     int i, j;
6     const unsigned long LED_PINS = ((uint32_t)1<<22);
7
8     /* Enable GPIO Clock */
9     CLKPWR_ConfigPPWR(CLKPWR_PCONP_PCGPIO, ENABLE);
10
11    /* LED on PORT0.22 defined as Output  */
12    GPIO_SetDir(0, LED_PINS, 1);
13
14    while(1) {
15        /* Delay some time */
16        for(i=500; i>0; i--)
17            for(j=10000; j>0; j--) {
18            }
19
20        /* Output low level  */
21        GPIO_ClearValue(0, LED_PINS);
22
23        /* Delay some time */
24        for(i=500; i>0; i--)
25            for(j=10000; j>0; j--) {
26            }
27
28        /* Output high level */
29        GPIO_SetValue(0, LED_PINS);
30    }
31 }
```

Figure 103: LEDBlinkExp( ) Function Modified for LPCXpresso Board

Figure 104 shows the completed project, ready to be compiled and debugged. While CoIDE automatically saves your source files every few minutes, it's still a good idea to get into the habit of hitting the "**Save**" toolbar button periodically.

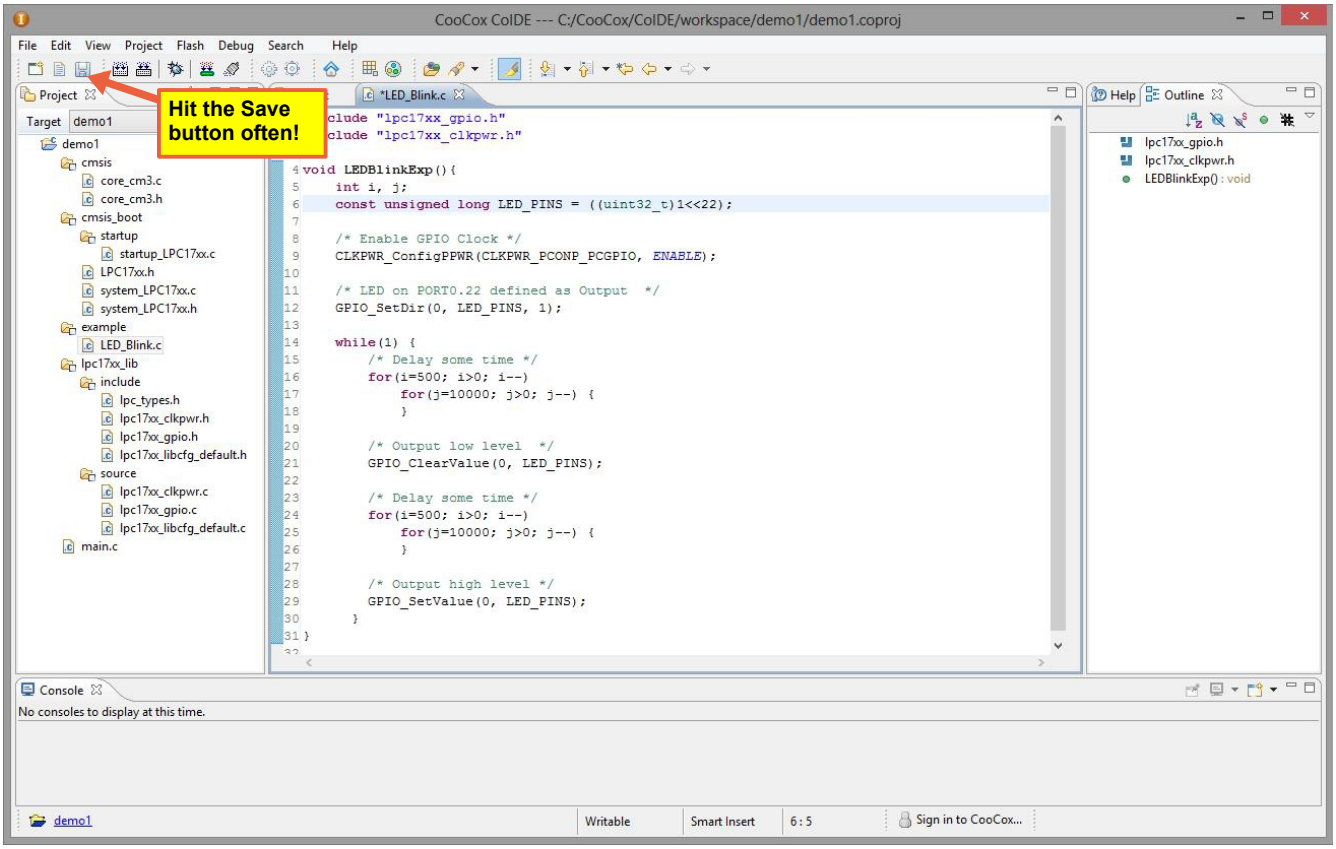


Figure 104: Completed, ready-to-compile blinker project

Build (Compile and Link) the Project

To build the project, there are two toolbar buttons: the "**Rebuild**" button (highlighted in Figure 105) will compile every C and assembler file in the project. This would obviously take longer in a really large project. The "**Build**" button to the left in Figure 105 will only compile those C and assembler files that you have changed. For a very large project, "**Build**" is more efficient.

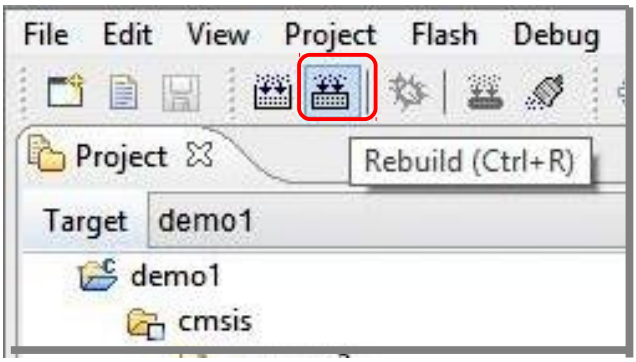


Figure 105: Hit "Rebuild button to compile and link all files

Figure 106 shows the result of the "**Rebuild**" operation. The console display at the bottom shows the messages emitted by the compiler and linker. The important bit is the "*BUILD SUCCESSFUL*" notation, indicating that no errors were detected and the program built successfully. Note that the program size (what will be loaded into the LPC1769 microprocessor's flash EPROM memory) is just 2476. bytes.

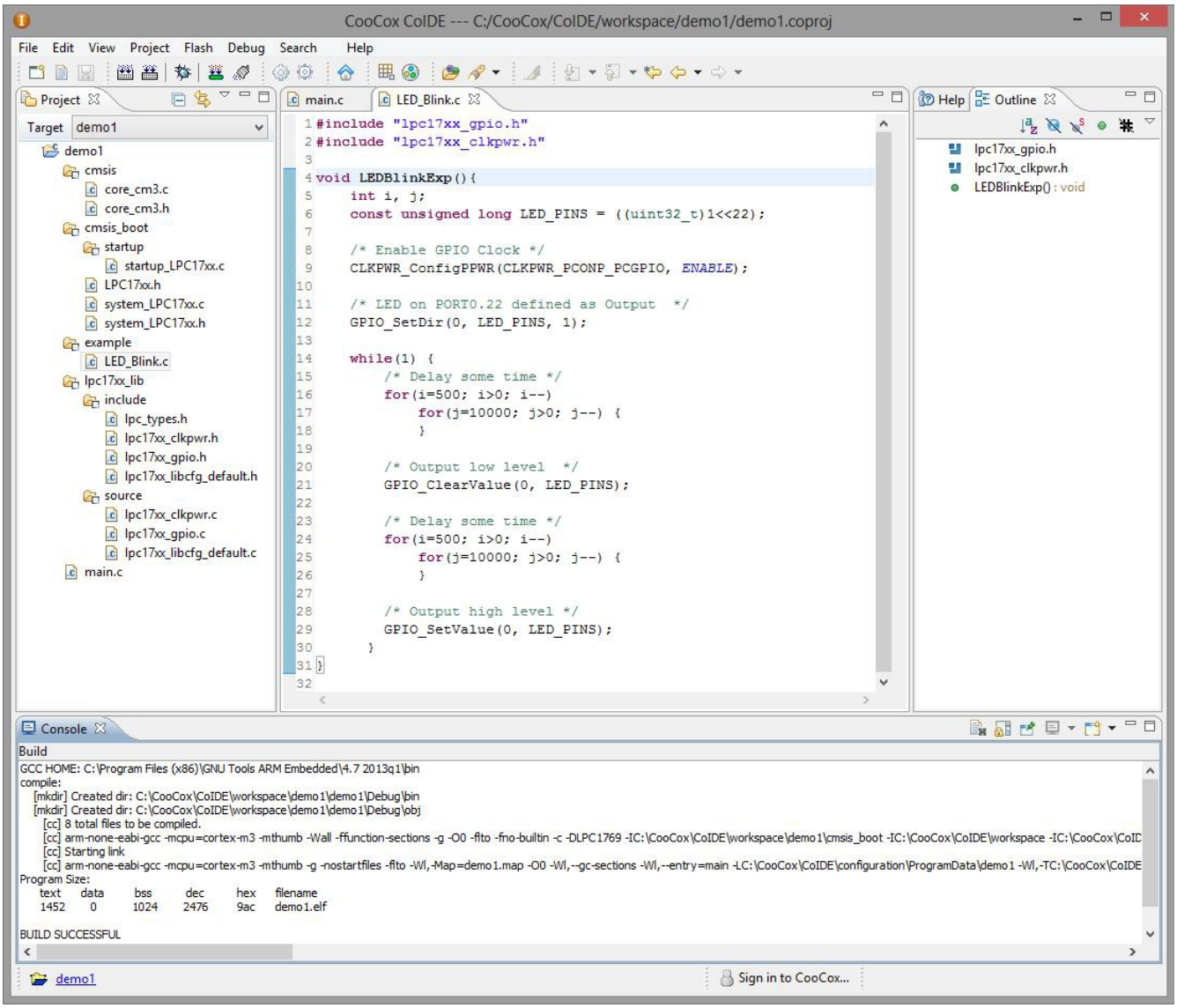


Figure 106: Build-All was Successful

Setting Up the Debugger

The next step is to download the program to the target board (LPC Xpresso) and run and/or debug it. Before attempting this for the first time on any project, **it's imperative that the debugger configuration be set up properly**.

As shown in Figure 107 on the right, click on "**View – Configuration**" to add the configuration screen to the center editing area.

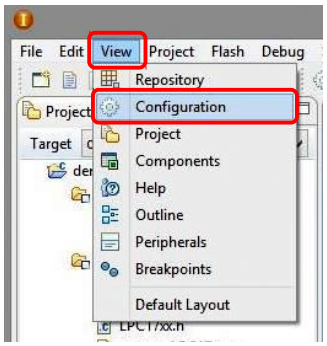


Figure 107: Access the Configuration Screen

When the configuration screen appears, select the "**Debugger**" tab as shown in Figure 108. Note that "**CoLinkEx**" was chosen as the debug adapter and the port is "**SWD**". I had to slow the "Max Clock(Hz)" to **100K** to get the debug adapter to work reliably.

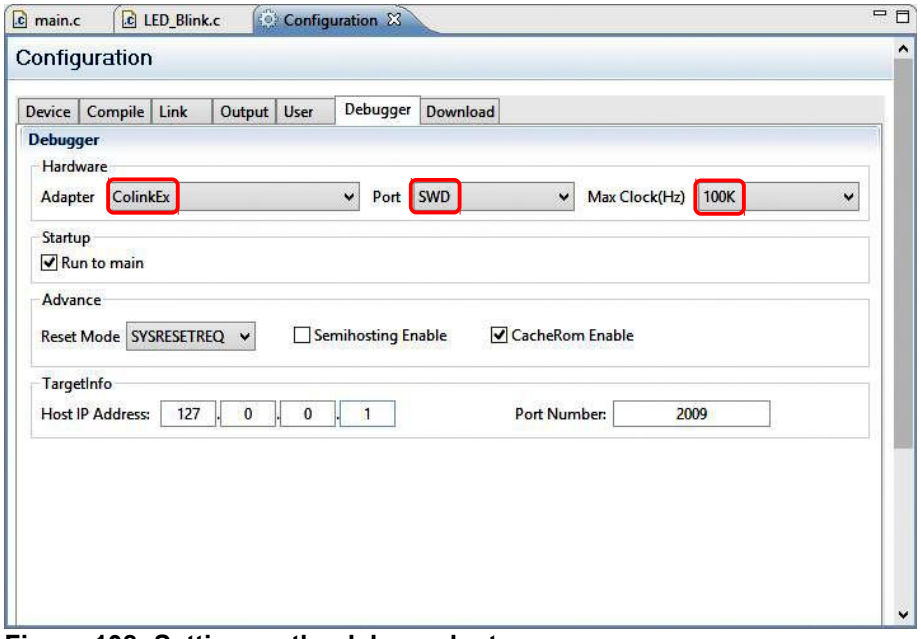


Figure 108: Setting up the debug adapter

Click the "**Start Debug**" toolbar button to start up the Eclipse/GDB debugger (Figure 109). CoIDE is defaulted so that by hitting the "**Start Debug**" button, your project will be automatically rebuilt and the resultant binary file will be automatically downloaded to the LPCXpresso target flash memory before the debugger runs. You can change this behavior, if desired, by using the "Configuration" window again.

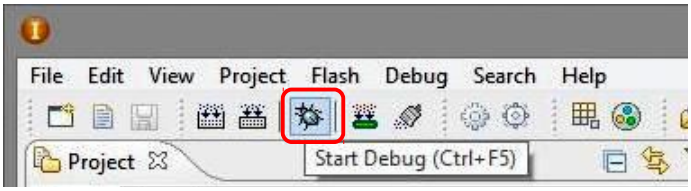


Figure 109: Click "Start Debug" toolbar button

Testing the Eclipse GDB Debugger

The Eclipse GDB debugger takes several seconds to start up. Consider what it has to accomplish: rebuild your project (in case you forgot), download and burn your program into flash EPROM, start up the LPC1769 CPU, set up the interrupt vector tables, initialize all required variables, and then jump to main(). The debugger will stop before the first executable line after the main() statement, as shown in Figure 110.

This is a “temporary” breakpoint and as is the case with ALL breakpoints, the debugger will stop just before the statement containing the breakpoint and thereby not execute that statement.

Note that the source file main.c has focus in the center editing area and the yellow arrow points to the statement where the debugger has stopped (it has not executed that statement).

The debug tab, on the bottom right of the screen, shows that the thread has suspended and a breakpoint has been encountered at line 7 of the C language function main().

The top right of the screen in Figure 110 shows a disassembly view of mixed C source lines and their ARM-Cortex assembler statements (you can see that the program will execute a branch-and-link instruction to jump to the LED blinker function).

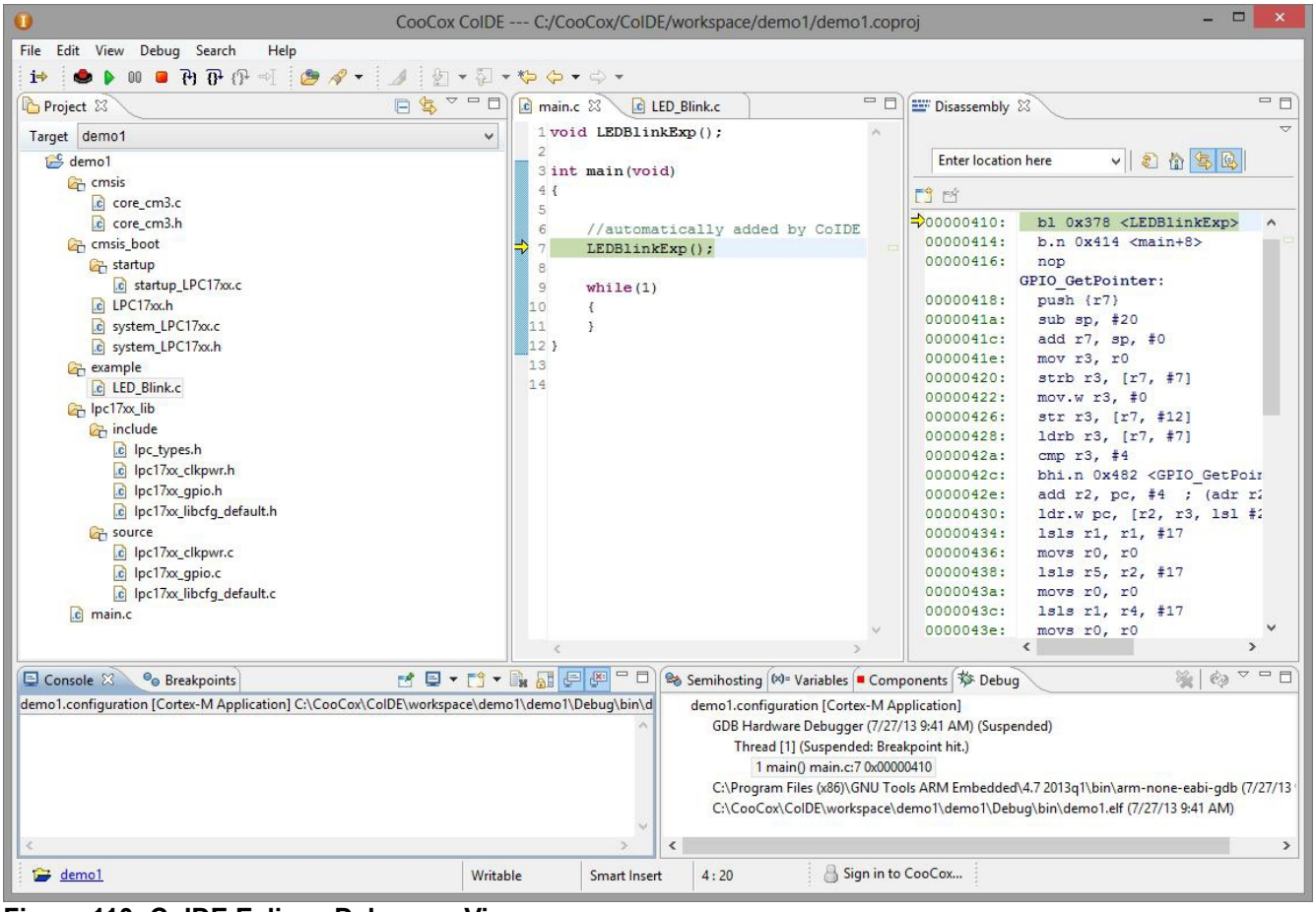


Figure 110: CoIDE Eclipse Debugger View

At this point we could probably say that the CooCox IDE is completely functional for the LPC1969 ARM-Cortex microprocessors. However, to be sure let's test a few of the debugger functions.

First, let's set a breakpoint. Click on the tab for the file **LED_Blink.c** and give it focus. If that file (tab) is not present in the editing area, just double-click on it in the project manager tree on the left and it will pop up in the editing area.

Set a breakpoint on the statement that turns on the LED (line 29 in Figure 111). Just double click on that source line on the far left (just left of the line numbers). When the breakpoint is set, a red circle will appear in the break point column with a check mark. This means that the breakpoint is “armed” and ready to go.

You can set five of these hardware breakpoints (Eclipse GDB uses an additional three breakpoints for single-stepping, etc). Remember that these breakpoints are built-in to the ARM-Cortex CPU hardware and allow the microprocessor to run at full speed.

There is an additional version of the hardware breakpoint called the “watchpoint”. The watchpoint monitors a memory location for a change in state or a specific value to appear. At the moment, CooCox does not implement this feature.

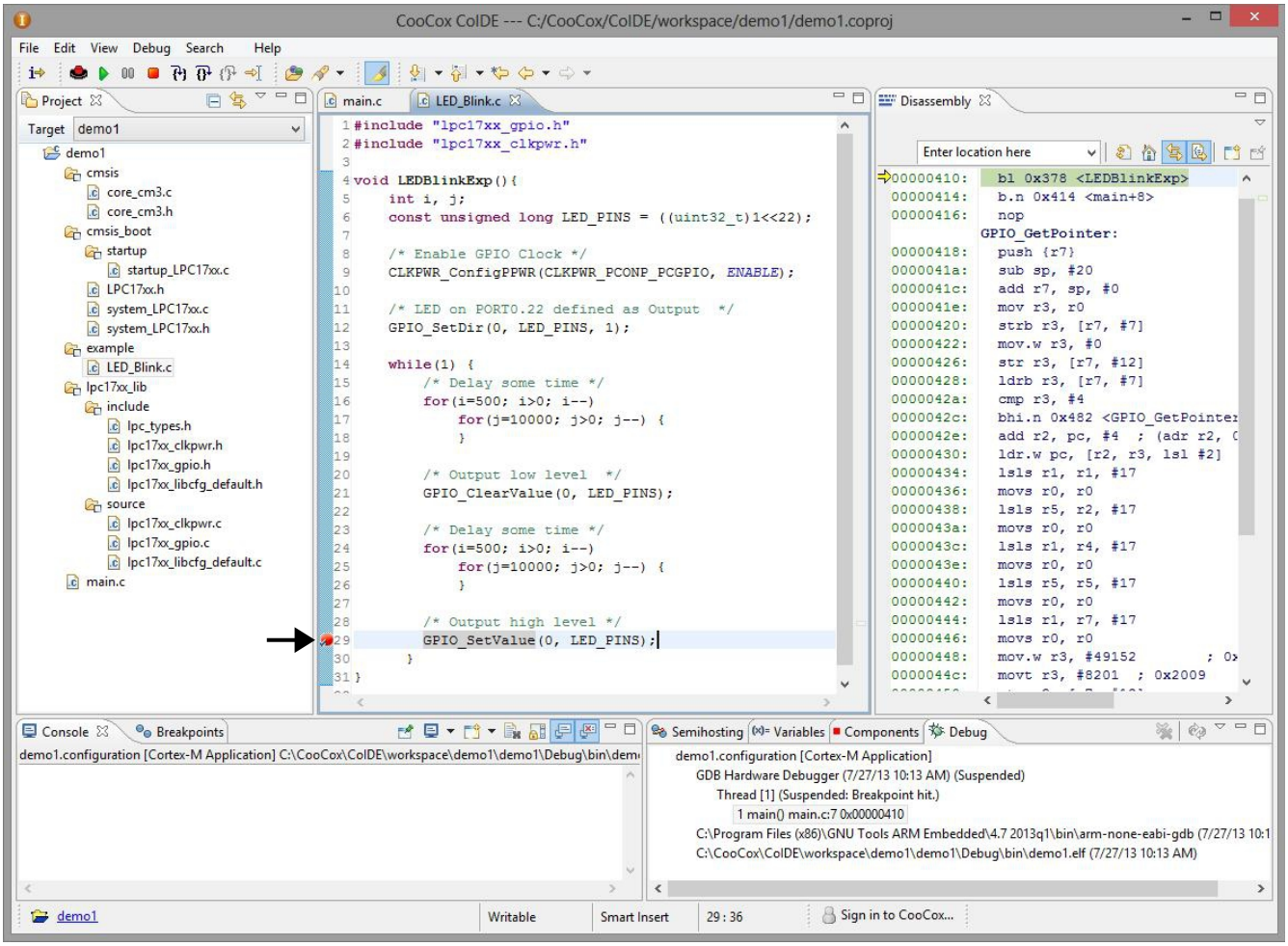
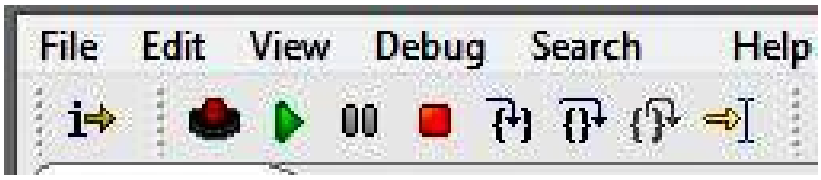


Figure 111: Setting a hardware break point

When the Eclipse GDB debugger is running, there are convenient toolbar buttons to do just about everything (Figure 112). It would be wise to study Figure 112 intensely and commit to memory the function of each debug toolbar button!

break point set



RUN - resumes execution at full speed. The program will thus run forever or halt at the next breakpoint encountered.

Suspend - stops execution wherever it is.

Step Into - executes just one source line. If the source line is a function call, it will enter that function and stop there.

Step Over - executes just one source line. If the source line is a function call, it will execute that function completely and stop at the line following the function call.

Step Out - if you are single-stepping within a function, the step out button will execute the remaining statements in the function and leave that function, stopping at the line following the function call.

RUN to Line - resumes execution at full speed and will halt at the source line where the cursor is located.

Reset CPU - restarts the debugger (rebuilds the application, downloads the binary to EPROM, starts the Eclipse GDB debugger, and resets the CPU, and then finally stops at the main function).

Terminate - stops the debugger and returns to normal Eclipse editing.

Instruction Stepping Mode - toggles between C source line stepping and assembler language source line stepping.

Figure 112: Debugger Toolbar Buttons

If we click the “Run” toolbar button  with the debugger currently stopped at main(), the target CPU will start execution from that point and continue until it hits the breakpoint at line 29 of the LED_BlinkExp() function. Note in Figure 113 below that the file LED_Blink.c now has focus and a yellow arrow locates where the debugger has stopped (line 29).

The Debug summary view at the bottom right indicates that the execution thread has suspended with a breakpoint hit at line 29 of function LEDBlinkExp().

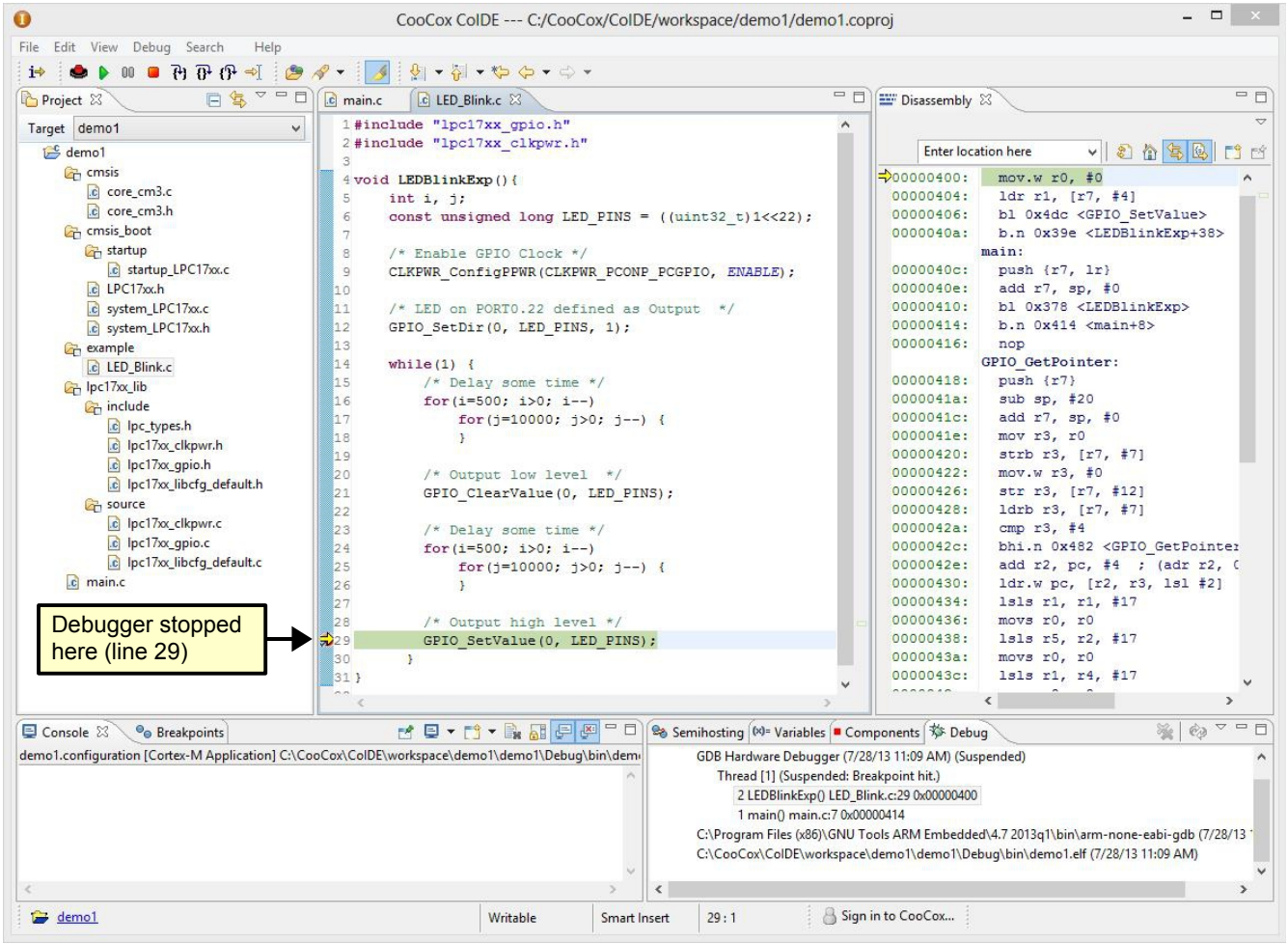


Figure 113: Debugger has hit breakpoint at line 29 of LED_Blink() function.

The breakpoint can be removed by simply clicking on it again (in the breakpoint column on the far left of the editing area).

If you right-click within the breakpoint column, there's a special pop-up menu that allows you to enable / disable breakpoints or to change their properties. For example, you can set the breakpoint to ignore the first three occurrences and break at the fourth hit.

If you are currently at a breakpoint, you can inspect variables for their current value by simply hovering the cursor over the variable name until the pop-up window appears that displays the current value.

Conclusions

If you have gotten to this point and your CooCox IDE performs as indicated in this document, then congratulations, you have a fully functional CooCox software development system for the LPCXpresso LPC1769 board. By just changing the selected chip, you'll be able to develop and debug software for the entire LPCXpresso line.

How much did all this cost?

LPC1769 LPCXpresso Evaluation board	$29.95	(Adafruit Industries)
CooCox CoLinkEx debugger board	$27.91	(Element14 / Newark)
40-pin break-away header pins	$1.50	(Sparkfun)
10-pin IDC socket rainbow breakout cable	$3.95	(Adafruit Industries)
10-pin socket/socket IDC cable – short 1.5 inches	$1.50	(Adafruit Industries)
CooCox COIDE software	free – open source	
gcc arm embedded tool chain	free – open source	
Total	$64.81	

If you take away the cost of the LPCXpresso board, then the cost of a complete software development system with debugger is about $35 (US). For that expense, you get a development system similar to the Code Red LPCXpresso IDE without the limitations. You can develop code for just about any ARM-Cortex chip with no limitation on program size and so forth.

In the interest of honest reporting, CooCox is a “work in progress”. The CooCox IDE does have bugs as any perusal of their online support forum will demonstrate. However, I really like what they are trying to do – select your chip and desired peripherals and CooCox will set up a functioning starter project for you.

CooCox also has a fairly complete real-time operating system that you can select for inclusion into your project. It's well-documented and relatively efficient.

If you are interested in developing applications for ARM-Cortex chips, CooCox is proving with each passing day that it is a viable and rich open-source software development environment. Give it a try, you can't beat the price!

Jim Lynch

July, 2013

About the Author

Jim Lynch lives in Grand Island, New York and is a software developer for Control Techniques, a subsidiary of Emerson Electric. He develops embedded software for the company's industrial drives (high power motor controllers) which are sold all over the world.

Mr. Lynch has previously worked for Mennen Medical, Calspan Corporation, and the Boeing Company. He has a BSEE from Ohio University and a MSEE from State University of New York at Buffalo.

Jim is a single father and has two grown children and four grandchildren who now live in Florida and Nevada.

Jim is very proud of his two brothers, one is a Viet Nam veteran in Hollywood, Florida and the other is the Bishop of St. Petersburg, also in Florida. Jim enjoys playing the guitar, woodworking, and going to the movies.

Lynch can be reached via e-mail at: lynch007@gmail.com