TECHNICAL ARTICLE

Ultra Low Power Microcontroller Module Opens New Doors for Engineers—Part 1: Eclipse Project Setup

Simon Bramble, Staff Engineer

Abstract

This article introduces an ultra low power, feature rich microcontroller module and explains how to program and debug it using popular, free of charge tools. Unlike many other high end microcontroller modules, this one is available in a DIP footprint, allowing easy prototyping for professional engineers and hobbyists alike. Part 1 describes how to create a project in Eclipse, and Part 2 discusses how to configure Eclipse to work with the microcontroller module.

Introduction

While the constant pursuit of miniaturization pushes the boundaries of electronics, it can leave engineers and hobbyists struggling to prototype with the ever-shrinking packages. While there is little doubt that including a microcontroller in a design adds a new dimension to the circuit's capabilities, many engineers still hanker for the days when more components were available in DIP packages. The more advanced microcontrollers are only available in packages that are impossible to prototype with, or only come with an evaluation kit that is huge and includes many unnecessary components.

This article introduces the prototype friendly MAX32625PICO microcontroller module that is incredibly small yet still has a DIP footprint so is extremely easy to integrate into both experimental and production PCBs. This article also informs the reader how to program and debug using Eclipse, the extremely popular, free integrated development environment (IDE). It will gently encourage engineers to leave behind the world of 8-bit processors, while still keeping a toe in the waters of the DIP package. For a complete guide on how to get started, see the following instructions.

Meet the MAX32625PICO

Figure 1 shows the MAX32625PICO, which is also known as the PICO. It contains the MAX32625 microcontroller, which is an ultra low power, 32-bit Arm® Cortex®-M4 processor with 512 kB of Flash and 160 kB of SRAM, while working at up to 96 MHz. The most useful of the microcontroller's peripherals have been made available to the pins on the PICO, including an SPI main, an SPI sub, an I²C port, two inputs to a 10-bit ADC, a 1-Wire interface, and two UARTs. The PICO also integrates the MAX1 4750 power management IC, an RGB LED, a push button switch, a USB Micro B connector, and a 10-pin Cortex debug header, and it can be powered directly from the USB port or an external 5 V supply. Moreover, it has 0.1 in header connectors on either side of the PCB so pins can be soldered to the PICO, making it mountable in a prototype breadboard. If a surface-mount solution is required, the header connectors have copper going right to the edge of the board, so the PICO can be used as a surface mount component too. The PICO is small enough to be slotted into any design, yet large enough to allow easy prototyping. See Figure 1.

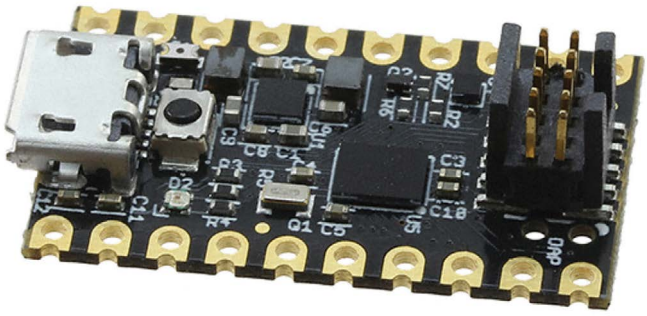

Figure 1. The MAX32625PICO.

Below are the components needed to program and debug the PICO:

- Two PICOs
- Two Micro B USB cables (normally supplied with the PICOs)
- A programming cable, serial number TC2050-IDC-NL-050

The TC2050 cable has a 10-pin connector on one end and spring loaded pogo pins on the other end. This cable is only used during debug or to reprogram the PICO if the bootloader gets overwritten. Normal programming is simply a drag and drop.

Creating a Project in the Eclipse IDE

The PICO was originally designed to be used with the online Mbed compiler platform, but this platform has now been discontinued. The open-source Eclipse IDE is a development platform and is extremely popular with both professionals and hobbyists and presents a free of charge alternative to Mbed. This article will instruct the programmer on how to configure Eclipse to generate a binary file that can be dragged to the target hardware using Windows Explorer. The configuration files can be downloaded in a zip file linked at the end of this article.

The Eclipse IDE can be downloaded from the landing page of the MAX32625 microcontroller. Navigate to the **Tools and Simulations** section and click on the **Low Power ARM Micro SDK (Win)** download link. From here the Arm Cortex Toolchain IDE can be downloaded and installed. Use the default installation configuration.

Once installed, select:

File -> New -> Maxim Microcontrollers

as shown in Figure 2.

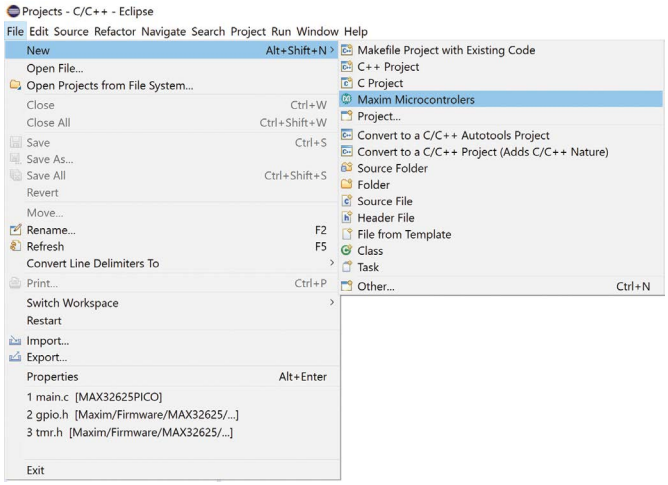

Figure 2. Creating the first project.

When the **Create Project** box appears, ensure the project name has no spaces.

Fill in the **Select Project Configuration** window as shown in Figure 3.

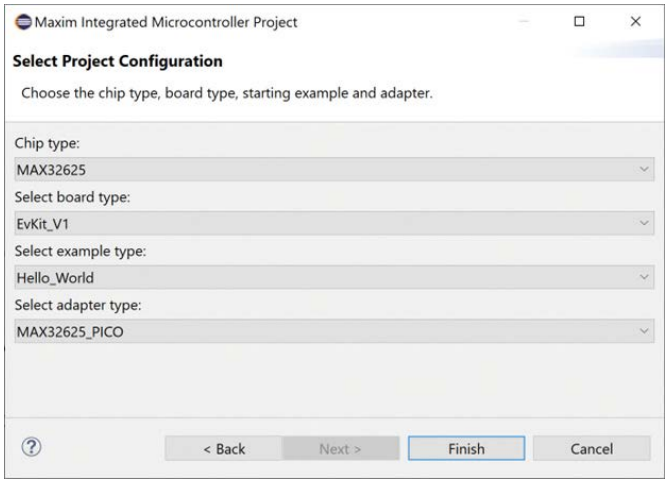

Figure 3. Project configuration.

How Does the IDE Communicate with the Target Hardware?

It is worth describing what is going on inside Eclipse to help explain any error messages that might be flagged up in the debugging process. The IDE communicates with the target

microcontroller (in this case, the PICO) using two programs: GNU debugger (GDB) and open on-chip debugger (OCD). GDB is a high level debugging tool that allows the user to step through code, execute breakpoints, and examine register values. GDB communicates with OpenOCD, which is a tool that translates high level commands into something closer to what the target microcontroller understands. Both GDB and OpenOCD run on the PC, inside Eclipse. Connected between the PC and the final hardware is a debug adapter that translates commands into electrical signals that the microcontroller's debug port understands. This debug port can take one of two forms: a JTAG debug port or a lower pin count version of JTAG, called single wire debug (SWD) port. All of the above allow the programmer to execute the code, line by line on the actual target hardware, seeing how the register values change, rather than on a simulator.

The Eclipse IDE talks to GDB, GDB talks to OpenOCD, and OpenOCD talks via the debug adapter, to the debug port on the target PICO. GDB and OpenOCD reside inside the software development kit (SDK) are installed automatically with the SDK, and are called by Eclipse. Once a debug adapter is connected between the PC and the target hardware, end to end communication between Eclipse and the PICO is seamless.

The project configuration shown in Figure 3 assumes the Eclipse IDE is going to be connected to a MAX32625EVKIT, which uses a JTAG port. However, the PICO uses the lower pin count SWD port instead of the JTAG debugger. To interface the Eclipse IDE with the PICO, a second PICO can be connected between the host PC and the target PICO, in place of the JTAG debugger. This second PICO will be called the programmer PICO, to distinguish it from the target PICO on which the final code will run. The programmer PICO needs to be loaded with a program called DAPLink, which translates USB signals from the PC into signals that the PICO understands. This is a simple procedure and is outlined in the Loading a Binary File section in Part 2 of this article.

Since a programmer PICO is being used instead of the JTAG debugger, under the **Select adapter type**, select **MAX32625_PICO**. See Figure 4.

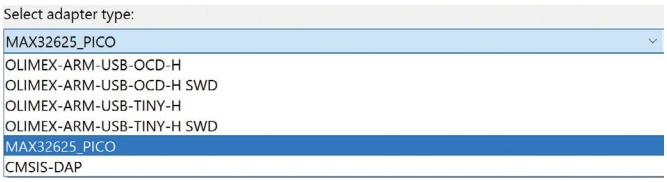

Figure 4. Select adapter type.

Modifying Eclipse to Work with the PICO

Download the zip file from the link at the end of this article and extract its contents to a convenient location. Locate the file called

MAX32625PICO.cfg

and copy it into the following directory:

C:\Maxim\Toolchain\share\openocd\scripts\target

This is a modification of the original file (MAX32625.cfg) and it enables a second PICO to be used in place of the JTAG debugger. This file overrides the reset command typically issued by the JTAG debugger and resets some of the registers of the target processor as well as the program counter and stack pointer.

Once the new project has been created, from within Eclipse's **Project Explorer** tab (top left corner), right click over the project name and select:

Debug As -> Debug Configurations...

Navigate to **GDB OpenOCD Debugging** on the left menu and select the current project.

In the **Debugger** tab, change the CFG file from **MAX32625.cfg** to MAX32625pico.cfg, as shown in Figure 5.

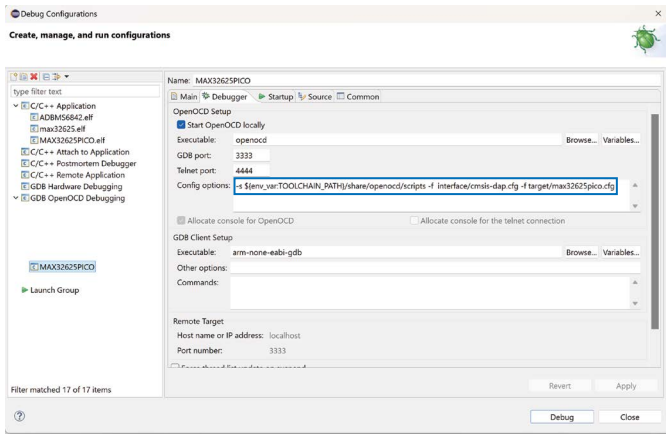

Figure 5. Pointing to the new config file.

Click **Apply**, then **Close**.

In the **Project Explorer** window, right click again over the project name and select **Properties**. Left click on the **C/C++ Build** heading and under the **Builder Settings** tab, ensure the **Build command** reads:

make ECLIPSE=1 release

as shown in Figure 6. This instructs Eclipse to generate the binary file (the executable program) that can be loaded into the PICO.

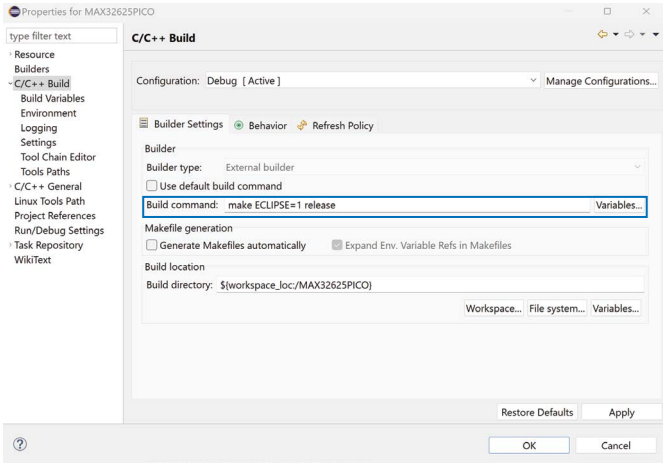

Figure 6. Generating the binary file.

Click **Apply** and **OK**.

Configuring Eclipse to Include Other Files

To make Arm processors easier to program, the microcontroller vendor provides low level drivers to enable the programmer's code to communicate with the many registers and peripherals inside the microcontroller. Therefore, the programmer does not need to worry about register level control and can concentrate on the higher level functionality of the end application. These drivers are called the Common Microcontroller Software Interface Standard (CMSIS) and are simply a hierarchy of files needed to program the MAX32625. At the heart of this microcontroller is the Arm core, which is common to each particular family of Arm-based microcontrollers. This has a set of files that are needed to configure it. Built around the Arm core is a set of peripherals (ADCs, GPIO ports, timers, counters, SPI ports, etc.), which is what distinguishes the device from other Arm-based devices. These peripherals need a set of files to configure them. The Arm core and its peripherals make up the microcontroller. The device is then mounted on a board and connected to, for example, a display, port headers, switches, LEDs, and Bluetooth® transceivers. There is a set of files that describe how it is connected to the surrounding components on the board.

Starting from the heart of the MAX32625 (the Arm core) and working outwards, there is a set of files to configure the Arm core itself, then a set of files to configure the peripherals surrounding the Arm core, and finally a set to configure the components on the evaluation kit that surround the processor. Eclipse needs to be configured so it can find these files when compiling the program.

Inside Eclipse, in the **Project Explorer** window, right click over the project name and select **Properties**, then expand the **C/C++ General** menu and navigate to the **Paths and Symbols** section. Under the **Includes** tab, select **GNU C**. Add the directories shown in Figure 7 using the **Add** and **File system** buttons.

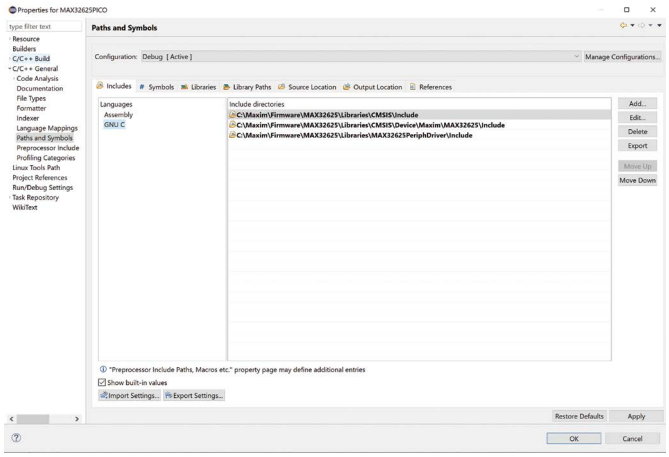

Figure 7. Including external files.

Click **Apply** and **OK** once the above have been added.

Below is an explanation of what each included directory does.

C:\Maxim\Firmware\MAX32625\Libraries\CMSIS\Include
This directory includes the files used by the Arm core of the MAX32625.

C:\Maxim\Firmware\MAX32625\Libraries\CMSIS\Device\Maxim\MAX32625\Include
This directory includes the peripheral/register definitions of the MAX32625.

C:\Maxim\Firmware\MAX32625\Libraries\MAX32625PeriphDriver\Include
This directory includes the files that enable the peripherals of the MAX32625 to be used.

Conclusion

Part 1 of this article has described how to create a project in Eclipse and the software needed to program a microcontroller. Part 2 discusses how to configure Eclipse to work with the PICO.

Download the software files here: https://www.analog.com/media/en/software/software-configuration/eclipse-configuration.zip

About the Author

Simon Bramble graduated from Brunel University in London in 1991 with a degree in electrical engineering and electronics, specializing in analog electronics and power. He has spent his career in analog electronics and worked at Linear Technology (now part of Analog Devices).