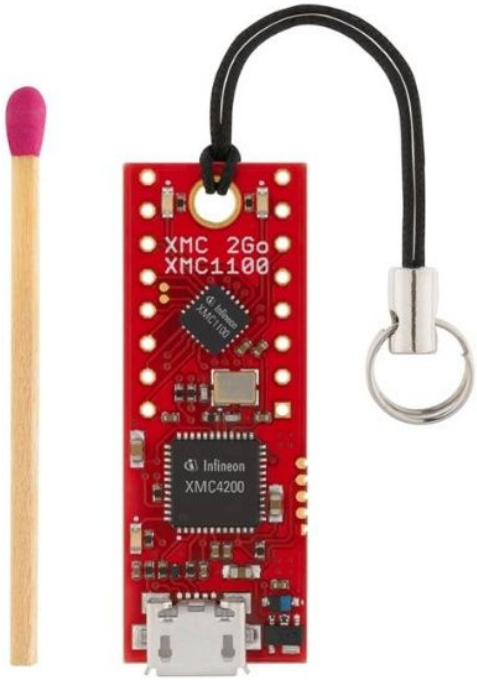
Getting Started with the XMC2GO (XMC1100)

Getting Started with the XMC2GO (XMC1100) **1**
- Step 1: Procure the XMC2GO 1
- Step 2: Pick an IDE/code editor 2
- Step 4: Write a program or open an example program 2
  - Open an example program 2
  - Write a program 3
  - Middleware 4
- Step 5: Compile the program 5
- Step 6: Flash the program to the MCU 5
- Step 7 (Optional): Start a debug session 7
- Helpful tools 8
- References 8
- Tested Toolchain Combinations 9

Step 1: Procure the XMC2GO



Step 2: Pick an IDE/code editor

- **Options (all are IDEs unless otherwise noted):**
    - **Infineon DAVE**
    - **Segger Embedded Studio**
    - **Visual Studio (VS) Code (with Platform IO; see "Middleware" below)**
    - **Eclipse**
    - **Altium, Atollic, IAR, Arm/Keil**[1]
    - **Arduino**[2]
    - **Your favorite text editor (NOT an IDE)**
- Infineon DAVE
    - Download DAVE (Note: There is no installation process beyond unzipping the download folder). You'll need to request access, though approval happens within a few hours or a day. That access only lasts for so long, however, so if you ever need to download the DAVE program again, you'll have to "re-request" access.
        - Unzip DAVE download; inside that folder is another zipped folder. Unzip THAT folder (may need to unzip to location with very short filepath; I had to unzip to C:\DAVE, otherwise I would get errors about filepaths being too long when I tried to unzip anywhere else).
    - Download and install Segger J-Link.
    - Make sure that the Segger J-Link option in DAVE ("Window" > "Preferences" > "Run/Debug" > "SEGGER J-Link") points to the SEGGER J-Link installation folder.
- Segger Embedded Studio
    - Download and install Embedded Studio.
- VS Code
    - Download and install VS Code. Follow the instructions below to get PlatformIO.

Step 4: Write a program or open an example program

Open an example program

- Infineon DAVE
    - I could not find any example programs in the normal DAVE download, but there are a few in the download for XMC Lib. I was able to import these examples into DAVE (by navigating to their folders using "File" > "Import" or "File" > "Open Projects from File System") but, unfortunately, I could not figure out how to get these programs to download the XMC2GO, which made their usefulness extremely limited.

[1] View DAVE information page, under "Tools & Software" > "DAVE Version 4" > "3rd Parties"
[2] https://github.com/arduino/Arduino/wiki/Unofficial-list-of-3rd-party-boards-support-urls

Write a program

- Infineon DAVE
    - Click “File” > “New” > “New DAVE Project”.
    - Give your project a name.
    - Select either “Easy Start Project”, “Simple Main Project”, or “DAVE CE Project” under “Project Type”.
        - “Easy Start Project” is a piece of Infineon demo code and may be overwhelming at first; it employs a number of inline functions and register writes to toggle some GPIOs and output a few strings over the UART port to a serial terminal.
        - “Simple Main Project” and “DAVE CE Project” are more blank to start, and may be easier to take in.
    - Select “ARM-GCC Application” under “Tool Chain” (if another is selected).
    - Click “Next”.
    - Find “XMC2GO” on the “Microcontroller Selection Page” and click “Finish”.
    - Write your application code or use DAVE Apps to create code for you (see “DAVE Apps” under “Middleware”, below).
- Segger Embedded Studio
    - Click “File” > “New project”.
    - In the window that opens up, select “Create the project in a new solution”.
    - Select the third option (“A C/C++ executable for a Cortex-M processor executing from FLASH memory”), give the project a name, and click “Next”.
    - Enter “XMC1100-0064” in the search bar and select the MCU that shows up. Click “Next”.
    - Leave the project settings as they are and click “Next”.
    - I left all “Project file” options as is.
    - I left all “Debug/Release” options as is.
    - Select “Finish”.
    - Download XMC Lib (under “Tools and Software” > “DAVE Version 4, DAVE Apps, XMC Lib and example projects”).
    - Add the files to your project that you specifically want to use.
        - I couldn’t find great documentation for XMC Lib. To figure out which files I needed I had to look in the “inc” and “src” folders within the “XMC Lib” folder. Those two folders contain the function calls and port definitions that you’ll want to use. For example, “xmc_gpio.c” contains the function “XMC_GPIO_SetMode” and “xmc_gpio.h” contains the inline function “XMC_GPIO_ToggleOutput”, which is all you need to get an LED to blink.
        - Once you find the functions you want to use, follow the trail of headers to figure out all of the dependencies you’ll need to include in the project to make everything work. For instance:
            - “xmc_gpio.c” includes “xmc_gpio.h”

                - “xmc_gpio.h” includes “xmc_common.h” and “xmc1_gpio.h”
                - “xmc_common.h” includes “xmc_device.h”
                - “xmc1_gpio.h” includes “xmc_common.h” and “xmc1_gpio_map.h”
            - Thus, to toggle a GPIO you’ll need to add the following files to your project:
                - “xmc_gpio.c”
                - “xmc_gpio.h”
                - “xmc_common.h”
                - “xmc1_gpio.h”
                - “xmc_device.h”
                - “xmc1_gpio_map.h”
            - Add those files (and any others you’ll need) to your project by clicking “Project” > “Add existing file…”. In my limited searching, I was unable to find a way to add entire folders, so you’ll have to add each file individually.
            - Since “xmc_device.h” relies on the exact MCU in question being #define’d, you’ll also need to #define that exact MCU either in your source code or as part of your build options in Embedded Studio (possibly, Embedded Studio will do this for you as part of specifying the MCU in creating the project).
        - The “XMC Lib” download has a few examples for reference.
        - Many of the examples include a function call to SysTick_Config (i.e. “SysTick_Config(SystemCoreClock >> 1);”). I was able to track this function to the file “core_cm0.h” and the reference to SystemCoreClock to the file “system_XMC1100.c”. However, I was not able to get my project in Embedded Studio to build by merely adding these files.
- VS Code + PlatformIO
    - Navigate to the PlatformIO Home.
    - Click “New Project” under “Quick Access”
    - Create a project name and search for “XMC1100 XMC2GO (Infineon)” under “Boards”.
    - Leave the Framework as is; the “Arduino” framework is the only one offered for the XMC2GO board.
    - Click “Finish”.
    - Write a sketch as if you were writing it for the Arduino. Pin definitions can be found in the “mapping_port_pin” array in pins_arduino.h.
        - I ultimately found that by entering a normal Arduino function call (such as “digitalToggle”), right-clicking that function name, and selecting “Go to definition”. Then I did the same thing for the “mapping_port_pin” struct that shows up in that function.
        - From that array, we can see that our analog inputs are on pins 12 and 13 (which map to XMC pins 2.9 and 2.7) and that our on-board LEDs are on pins 14 and 15 (mapped to XMC pins 1.0 and 1.1).
- Text editor

    - Download XMC Lib (under “Tools and Software” > “DAVE Version 4, DAVE Apps, XMC Lib and example projects”).
    - Follow the steps above under “Segger Embedded Studio” (above) to find all of the supporting files you’ll need and the example programs that may help you figure out how to use them.

Middleware[3]

- DAVE Apps (requires DAVE IDE)
    - Might come preinstalled with DAVE.
    - If not, download DAVE apps from DAVE support page
    - Start new DAVE project.
    - Click “DAVE” > “Add New APP…”
    - Add the desired apps, one for each usage (e.g. add a Digital_IO app for each GPIO pin you want to use)
    - Configure each app as desired; double-clicking on the app icon in the “App Dependency” window will open up the editable properties. Can also allocate pins graphically (Manual Pin Allocator) and can wire signals between apps (HW Signal Connectivity).
    - Right-click on the app icon and select “Rename Instance Label…” to give the app a meaningful name.
    - Need to dig into the included files to see what the API is.
    - Click “DAVE” > “Generate Code” to push settings to your project code.
    - Could possibly use these outside DAVE, but I don’t want to have to figure out all the dependencies. You’d also lose the usefulness of configuring via the GUI.
- XMC Lib (HAL)
    - Primarily for use with Altium, ARM/KEIL, Atollic, and IAR, though some functionality can be gained by manually sifting through source code (see above).
- PlatformIO
    - Uses “Arduino” framework to employ XMC1100 peripherals

Step 5: Compile the program

- This will simply compile the program without downloading it to your XMC1100 or starting a debug session.
    - To compile the program and download it to your XMC1100, see “Flash the program to the MCU” below.

---

[3] Middleware is any piece of software which makes it easier for the developer to configure the MCU and interface with its various peripherals. Common examples include a header file (which simply provides convenient handles for each of the registers), an API (which provides functions for the developer with which they can configure and manipulate the MCU and peripherals), and a GUI (which present the configurable parts of the MCU and peripherals in a more human-readable and intuitive fashion, though they are typically only provided inside the MCU manufacturer’s branded IDE).

    - To compile the program, download it to your XMC1100, and start a debug session, see “Start a debug session” below.
- Infineon DAVE
    - Click “Project” > “Build” or
    - Press Ctrl+B (“Build” hotkey) or
    - Click the icon of the blue hammer over a white background (“Build” shortcut) or
    - Click “Project” > “Rebuild” or
    - Click the icon of the white hammer over a blue background (“Rebuild” shortcut)
- Segger Embedded Studio
    - Click “Build” > “Build [PROJECT NAME]” or
    - Press F7 (“Build” hotkey) or
    - Click “Build” > “Rebuild [PROJECT NAME]” or
    - Press Alt+F7 (“Rebuild” hotkey)
- VS Code
    - Select the “PlatformIO” icon (looks like an alien’s head) on the left navigation pane.
    - Click “Build” under “Project tasks”.
- Text editor
    - From a shell in the same directory as the project you’re trying to build, run the following gcc compilation command: GCC -mthumb -mcpu=cortex-m0 -Os -g -c -I ./ main.c xmc_gpio.c system_XMC1100.c startup_XMC1100.S
        - “GCC” is the filepath to the gcc binary configured for ARM targets (mine was “/opt/gcc-arm/bin/arm-none-eabi-gcc”).
        - Here, my program in main simply toggles a GPIO and so only needs the API included in “xmc_gpio.c” and “xmc_gpio.h”. You should include whatever other source code you need for you application.
        - For simplicity, I placed all of the necessary header files in the same folder as the source and startup files listed above (hence the “-I ./”). I needed the following from the XMC Lib folder:
            - xmc_gpio.h
            - xmc_device.h
            - xmc_common.h
            - xmc1_gpio.h
            - xmc1_gpio_map.h
            - XMC1100.h
            - cmsis_gcc.h
            - core_cm0.h
            - core_cmFunc.h
            - core_cmInstr.h
            - system_XMC1100.h
    - Then run the following gcc linker command: “GCC -mthumb -mcpu=cortex-m0 -Wl,-Map=main.map -Wl,--gc-sections -T XMC1100x0064.ld main.o xmc_gpio.o system_XMC1100.o startup_XMC1100.o -o main.elf

        - Make sure the linker script, “XMC1100x0064.ld”, is in the same folder as the rest of the files (or provide the correct filepath if it isn’t).
        - This creates an executable in ELF format.
    - Finally, convert the ELF to a binary that we can load onto the MCU with the following shell command: “OBJCOPY -O binary main.elf main.bin”
        - “OBJCOPY” is the filepath to the gcc binary configured for ARM targets (mine was “/opt/gcc-arm/bin/arm-none-eabi-objcopy”).

Step 6: Flash the program to the MCU

- This will compile your program and download it to your XMC1100.
    - To simply compile your program without downloading it to your XMC1100, see “Compile the program” above.
    - To compile the program, download it to your XMC1100, and start a debug session, see “Start a debug session” below.
- Infineon DAVE
    - To the best of my knowledge, DAVE doesn’t have this option. The only way to flash the MCU is to use one of the options below that also start a debug session.
- Segger Embedded Studio
    - Click “Target” > “Download [PROJECT NAME]” or
    - Click “Build” > “Build and Run”
- VS Code
    - Select the “PlatformIO” icon (looks like an alien’s head) on the left navigation pane.
    - Click “Upload” under “Project tasks”.
- Text editor
    - **Options:**
        - **J-Flash executable**
        - **J-Flash command line**
        - **J-Link GDB server & GDB**
        - **OpenOCD GDB server & GDB**
    - J-Flash executable
        - Find and run the file “JFlashLiteExe”
        - In the window that opens, select the button with three dots to set the correct device (XMC1100-0064 for the XMC2GO).
        - Make sure the interface is SWD.
        - I set the speed to 4000 kHz and this worked for me; I have no idea if faster download speeds are possible.
        - Click “OK”.
        - In the “Data file” portion of the resultant window, click the button with three dots again to select the binary (.bin) file created above.
        - Ensure that the programming address (“Prog. addr.”) is 0x10001000.
        - Click “Program Device”.

    - J-Flash command line
        - I did not prototype with this.
    - J-Link GDB server & GDB (to flash the program to the MCU from INSIDE a debugging session)
        - See below.
    - OpenOCD GDB server & GDB (to flash the program to the MCU from INSIDE a debugging session)
        - See below.

Step 7 (Optional): Start a debug session

- This will compile your program, download it to your XMC1100, and start a debug session.
    - To simply compile your program without downloading it to your XMC1100, see “Compile the program” above.
    - To simply compile the program and download it to your XMC1100, see “Flash the program to the MCU” above.
- Infineon DAVE
    - Right-click the project and select “Run as” > “DAVE C/C++ Application” or
    - Right-click the project and select “Debug as” > “DAVE C/C++ Application” or
    - Click the icon in the shape of a bug
- Segger Embedded Studio
    - Click “Build” > “Build and Debug” or
    - Click “Debug” > “Debug with Ozone” or
    - Press Alt+F5 (“Debug with Ozone” hotkey)
- VS Code
    - Click “Debug” > “Start Debugging” or
    - Press F5 (“Start Debugging” hotkey)
- Text editor
    - **Options:**
        - **Ozone**
        - **J-Link GDB Server & GDB**
        - **OpenOCD GDB server & GDB**
    - Ozone
        - Download, install, and run Ozone.
        - In the window that pops up, select the XMC1100-0064 for the “Device”.
        - Leave “Peripherals” blank.
        - Click “Next”.
        - Set the target interface as SWD and the speed to whatever you want (4 MHz was the default I used).
        - Select your emulator from the ones listed; if you’re connecting to your XMC2GO with a USB cable, ensure that “USB” is selected under “Host interface”.

        - Select the ELF or binary file for your program and click “OK”.
    - J-Link GDB Server & GDB
        - Find and run the file “JLinkGDBServerExe”.
        - Select “XMC1100-0064” under “Target device” and set “Target interface” to SWD.
        - If you’re connecting to your XMC2GO with a USB cable, ensure that “USB” is selected under “Connection to J-Link”.
        - Leave other options as they are and click “OK”.
        - The next window should show you:
            - a red bar next to “GDB” with the message “Waiting for connection”,
            - a green bar next to “J-Link” with the message “Connected”, and
            - a green bar next to “Device” with the message “XMC1100-0064 (Halted)”.
            - Some of the messages may be cut off.
        - The log should list no errors and the last line should be “Waiting for GDB connection…”
        - Take note of the line that starts “Listening on TCP/IP port ####”, as this will be the port we connect to from GDB.
        - Now run GDB from a shell and connect to the J-Link GDB server. For me, this looked like this:
            - $ /opt/gcc-arm/bin/arm-none-eabi-gdb main.elf
            - (gdb) target remote :2331
    - OpenOCD GDB server & GDB
        -

Helpful tools

- XMC Pinout tool
- uC/Probe XMC
    - Under “Tools & Software” > “uC/Probe XMC”
- Segger Utilities
    - Listed under “More information” > “Technology” at the bottom of the webpage.
- Third-party utilities that integrate with Segger J-Link

References

- Product page for XMC2GO
- Product page for XMC1100
- DAVE Quick Start Guide
- DAVE information booklet
- DAVE SDK Introduction

Tested Toolchain Combinations

See HERE for something more legible.

MCU family	MCU part number	Board	Debug adapter	IDE/Code editor	Middleware	Compiler/Linker	Adapter driver	Debug software	Tested?	OS	Notes
XMC1100	XMC1100Q024X0064ABXUMA1	XMC2GO	J-Link Lite OB	Embedded Studio	XMCLib	Embedded Studio	Embedded Studio	Embedded Studio	Yes	Ubuntu 18.04.03	Felt like I used XMCLib only clumsily. I had to manually add each source and header file I wanted to use (and each of their dependencies) and fix any broken references.
XMC1100	XMC1100Q024X0064ABXUMA1	XMC2GO	J-Link Lite OB	DAVE	DAVE Apps	DAVE	DAVE	DAVE	Yes	Windows 10	Had to install DAVE to C:\DAVE since filepaths were so long. DAVE apps were a touch confusing to figure out.
XMC1100	XMC1100Q024X0064ABXUMA1	XMC2GO	J-Link Lite OB	Text editor	XMCLib	GCC	J-Link GDB Server	GDB	Yes	Ubuntu 18.04.03	All necessary files found inside XMCLib download.
XMC1100	XMC1100Q024X0064ABXUMA1	XMC2GO	J-Link Lite OB	Text editor	XMCLib	GCC	J-Flash	Ozone	Yes	Ubuntu 18.04.03	Feature-rich graphical debugging tool.
XMC1100	XMC1100Q024X0064ABXUMA1	XMC2GO	J-Link Lite OB	VS Code	PlatformIO	VS Code	VS Code	VS Code	Yes	Ubuntu 18.04.03	Uses Arduino API. Really painless, though I have had much trouble with PlatformIO in the past on different platforms.
XMC1100	XMC1100Q024X0064ABXUMA1	XMC2GO	J-Link Lite OB	Text editor	XMCLib	GCC	OpenOCD	GDB	Yes	Ubuntu 18.04.03	OpenOCD documentation does not have very clear getting started instructions.