Getting Started with the STM32G031J6

Step 1: Build the circuit

Figure 15. **STM32G0 Series reference schematic**

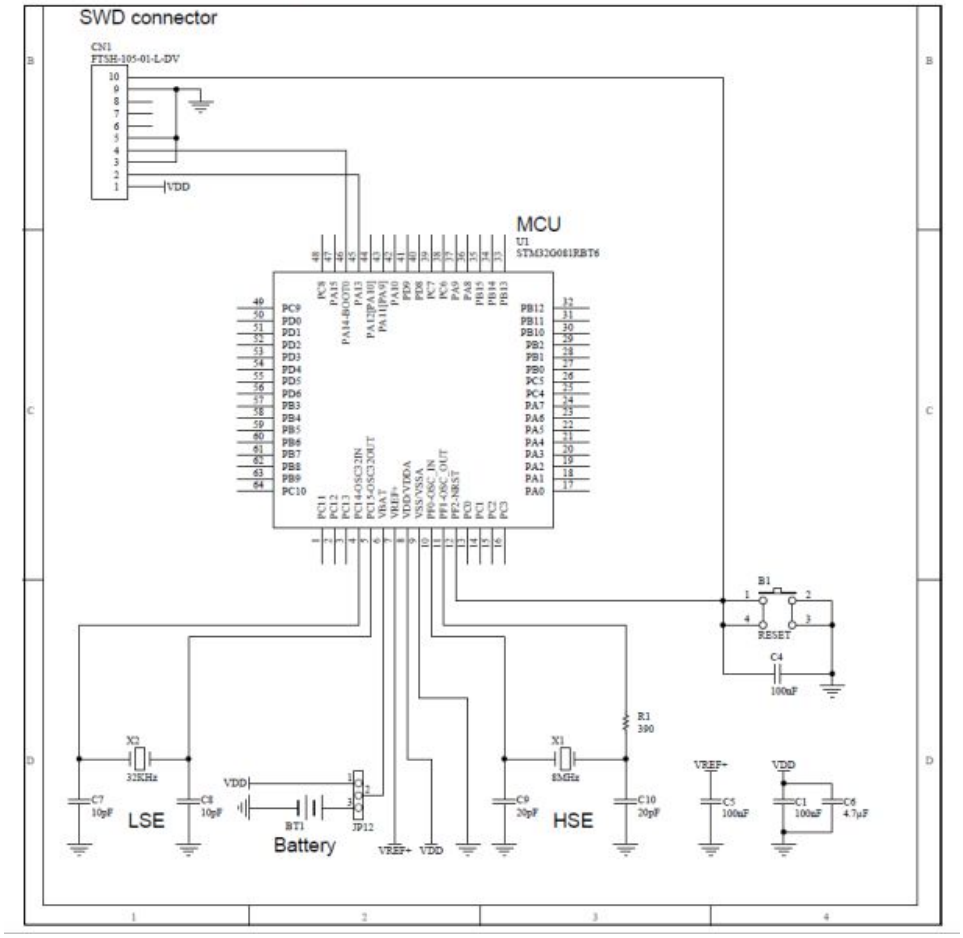


Figure 1: Recommended schematic for the minimum required circuit to program STM32G0x devices. Image was taken from the document [Getting started with STM32G0 Series hardware development (AN5096)], Figure 15 on page 25.

Figure 10. STM32G031Jx SO8N pinout

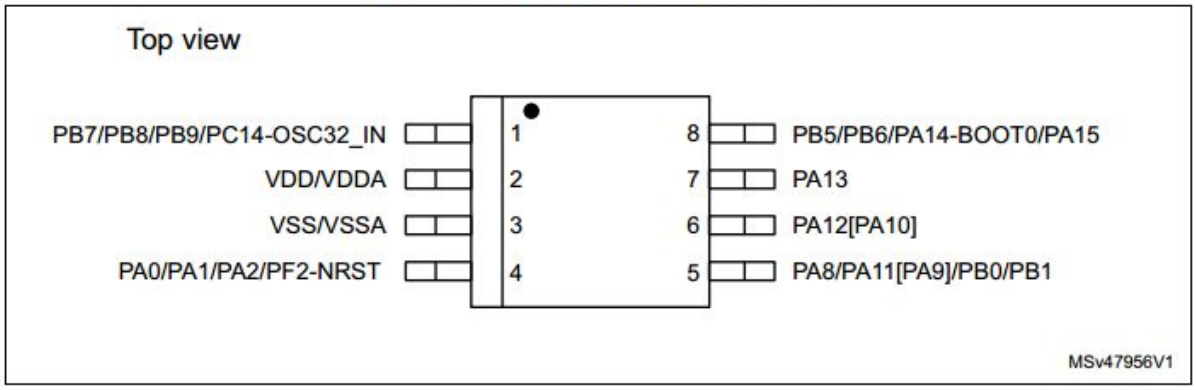


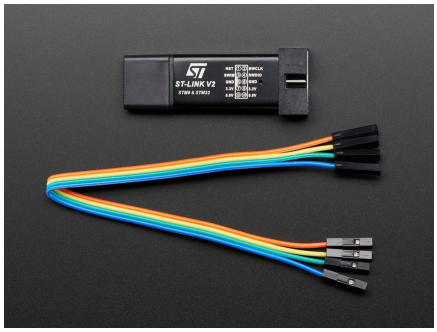
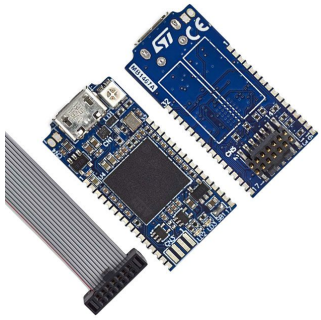
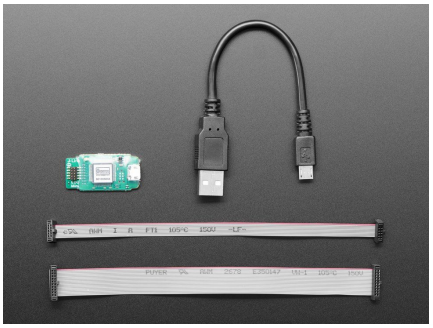
Figure 2: Pinout for the STM32G031J6. Image was taken from the document Datasheet for STM32G031 (DS12992), Figure 10 on page 34.

- **Options:**
    - (1) Buy an 8-SOIC breakout board[1] or create one with a PCB mill or PCB fabrication service on which you solder the STM32G031. Put the breakout board and any other necessary components on a breadboard for temporary prototyping or on a stripboard or protoboard[2] for more permanent prototyping.
    - (2) Use a PCB mill or PCB fabrication service to create the minimal development board that I designed for the STM32G031 (see here for more information about how to order a PCB from JLCPCB). Hand-solder all of the components.
    - (3) Order the minimal development board that I designed for the STM32G031 fully assembled from JLCPCB (see here for more information about how to order an assembled PCB from JLCPCB).
- The STM32G031J6 was chosen to prototype with because the STM32 line is a highly popular family of MCUs and as this variant comes in an SOIC package, it allows the hobbyist to hand-solder it to a PCB, if they so choose. This allows the hobbyist the options of 1 and 2 (above), whereas more common MCUs in tighter packages would only have afforded the hobbyist option 3.
- The primary resource for how to design the required circuit came from the document Getting started with STM32G0 Series hardware development (AN5096). There is an example circuit in figure 15 on page 25 and the document, as a whole, offers great information about various circuit requirements, such as what the supply voltage needs to be and how to connect a debug adapter.
- Schematic notes for the STM32G031J6:
    - Pin 2 (VDD/VDDA) should be connected to 1.7-3.6V.
    - Pin 3 (VSS/VSSA) should be connected to GND.
    - The following components or sections of the circuit from the Getting started with STM32G0 Series hardware development (AN5096) document are not critical (or available, in all cases, on the STM32G0) and so were left out of the development board that I designed:
        - LSE (low-speed external oscillator circuit)
        - HSE (high-speed external oscillator circuit)
        - Battery
        - VREF+
    - C1 (100 nF) and C6 (4.7 uF) are power filtering capacitors and should be placed as close to the VDD/VDDA pin as possible.
    - B1 and C4 (100 nF) are the reset button and a smoothing capacitor (to prevent switch bounce from triggering multiple resets in a row).
- Additional notes for the minimal development board that I designed:
    - LED1 and R1 are a power indicator sub-circuit. The value of R1 should be adjusted for the desired brightness level, based on the supply voltage and the forward voltage of LED1.

[1] Options from Adafruit and Sparkfun
[2] Options that I've designed for PCB fabrication can be found here

- LED2 and R2 are a user-defined output, to enable a basic “Blinky” program. The value of R2 should be adjusted for the desired brightness level, based on the supply voltage and the forward voltage of LED2. J# is provided should you want to disconnect this LED from your circuit; simply cut the trace between the terminals of J# with a small Dremel or hobby knife to do so. Should you wish to put the LED back into the circuit, simply solder two male headers to the pads of J# and install a jumper across them (or find another similarly clever means of reconnecting the LOWER/UPPER pad of J# [the pad to which LED is connected] to one of the GPIO pins on the STM32G0).
- D# and D# are blocking diodes to prevent circuit damage in the event that the STM32G0 is connected to power supplies both from J# via an external circuit and via J# from an ST-Link debug adapter.
- J# is specifically designed to work with the J-Link EDU Mini debug adapter and is also pin compatible with pins 3-12 of the ST-Link v3 Mini. However, neither of those debug adapters provide power to the MCU (the “Vref” pin connected to the supply voltage is simply used by the J-Link/ST-Link v3 Mini to determine what the operating voltage of the MCU is) and you’ll need to find another means to do so. If you are still breadboarding your circuit at this point, you could use a breadboard power supply[3] connected to the V_IN pin on J#.
- There is no overvoltage protection on this development board so I cannot guarantee that your circuit will survive accidentally being connected to any power supply greater than 4V (Table 18 on page 44 of the Datasheet for STM32G031 (DS12992) lists this as the absolute maximum rating for VDD/VDDA). If you wish to include this in your circuit, I recommend building this simple circuit.

Step 2: Pick a debug adapter

- **Options:**
  - **ST-Link v2 (reduced)**
  - **ST-Link v2 (full)**
  - **ST-Link v3 Mods**
  - **ST-Link v3 Set**
  - **ST-Link v3 Mini**
  - **J-Link EDU Mini (and all other J-Link models)**
  - **ST-Link on-board (OB) reprogrammed as J-Link OB**

[3] One option here.

- The difference between the ST-Link v2 "reduced" and "full" (my own terms) seems to be their form factor and pinouts. There is no reference made to the “reduced” model on ST’s website, making it likely that these debug adapters are only manufactured by third-parties. However, they seem fully supported by ST’s software. Some of the “reduced” models seem to support JTAG programming (see the Seeed model in footnote 5, below) while others do not (the Adafruit model in the same footnote).
- The ST-Link v3 Mods and Set are listed above for completeness, but deemed unsuitable for hobbyists for either their cost (\$35 for the ST-Link v3 Set and \$70 or more for other J-Link models) or availability (I could find no distributors with the ST-Link v3 Mods in stock at the time of this writing). However, of note is that the ST-Link v3 Set includes the ability to control SPI, UART, I2C, CAN, and GPIO peripherals using an adapter board (MB1440), which many may deem a worthwhile initial expense.
- See here for notes about how to reflash the on-board ST-Link debug adapter for many popular ST development boards with J-Link firmware. This can be an inexpensive way to enable J-Link debugging, however, the license terms state that the resulting debug adapter may only be used with ST products, which limits its usefulness. If you are short on cash, only plan to develop for ST MCUs, and want to be able to use a J-Link debug adapter, then this may be the best option for you. For everyone else, see below.
- The remaining four options are compared below. To me, the data suggests the following:
    - Get the ST-Link v2 (reduced) if you are short on cash and want to develop for the STM8[4].
    - Get the ST-Link v2 (full) if you aren’t short on cash and want to develop for the STM8.
    - Get the J-Link EDU Mini if you think you might develop for multiple different MCUs and want a powerful debug adapter.
        - If you do, I would still recommend getting an inexpensive ST-Link v2 (reduced). The STM32G0 seems to allow developers the ability to reconfigure the programming pins, effectively locking them out from any further development and an ST-Link seems critical to be able to erase the device for any further programing. See the WARNING note under “Write a program”, below, for further details.
        - I would also recommend finding a breakout board for the 10-pin J-Link connector that will make prototyping much easier (since not all development boards will have the 10-pin connector).
    - Get the ST-Link v3 Mini if you think you’ll develop entirely for the STM32 and desire a very powerful debug adapter.

	ST-Link v2 (reduced)	ST-Link v2 (full)	ST-Link v3 Mini	J-Link EDU Mini

[4] This MCU isn’t covered much here, but it’s performance is on par with the ATMega328 from what I can tell and I can be bought on a development board for as little as \$1.10! (Note: I have reason to believe that these are genuine MCUs but have not yet confirmed this.)

Cost	~$8-12[5]	~$22[6]	~$10-15[7]	~$20[8]
Compatible MCUs	All STM8 & STM32	All STM8 & STM32	Only STM32	So many
Protocols supported	● SWIM (STM8) ● SWD (STM32) ● JTAG (some models)	● SWIM (STM8) ● SWD (STM32) ● JTAG (STM32)	● SWD ● JTAG	● SWD ● JTAG
Provides power to MCU?	Yes	No	No	No
Terminal options [9]	None	SWO[10]	● SWO ● UART	● SWO ● RTT[11]
Data rate	● SWIM: 12.8 Kbytes/s ● SWD: 4 MHz [12]	● SWIM: 12.8 Kbytes/s ● SWD: 4 MHz [13]	● SWD: 24 MHz[14] ● JTAG: 21 MHz[15] ● UART (also called “VCP”, virtual COM port): 15 Mbps[16]	● SWD/JTAG: 4 MHz

[5] From Seeed or Adafruit
[6] From Digikey, see here for a full list of suppliers
[7] From Digikey, see here for a full list of suppliers
[8] From Adafruit
[9] These are ways you can communicate from your code back to a computer. A UART terminal is the same way an Arduino sends messages back to a computer. SWO performs a similar function, but requires a SWO-compatible debug adapter and software. It has the advantage of not requiring a UART connection on the MCU (since some applications may need that for other purposes) but is only available for more expensive MCUs (specifically, Cortex-M3/M4/M7).
[10] Only available on Cortex-M3/M4/M7.
[11] Works for all devices supported by Segger (i.e. not limited to only the Cortex-M3/M4/M7). For more about RTT, see here.
[12] Based on the highest connection setting available for this device in any software I tested. I could not find this specification listed anywhere in ST’s documentation for the ST-Link v2, including the ST-Link v2 User Manual (UM1075).
[13] See above.
[14] Listed on page 10 of the ST-Link v3 User Manual (UM2502)
[15] See above.
[16] See above.

Step 3: Pick an IDE/code editor

- **Options (all are IDEs unless otherwise noted):**
    - **STM32CubeIDE**
    - **Keil MDK-Arm**
    - **Segger Embedded Studio**
    - **IAR Embedded Workbench**
    - **Atollic TrueStudio**
    - **Eclipse**
    - **CXSTM32**
    - **SW4STM32**
    - **Your favorite text editor (NOT an IDE)**
- STM32CubeIDE
    - Download and install STM32CubeIDE.
    - May need to create an account with ST first.
- Keil MDK-Arm Lite
    - Download and install the free version of MDK-Arm (called “MDK-Arm Lite”). With this version, you’ll be limited to creating binaries that are no greater than 32kB (which works fine for this MCU, since that’s the amount of flash we have anyways).
    - There is no Linux or Mac version of this software, that I am aware of. On Linux, I downloaded and installed this in a Windows 10 virtual machine using Virtual Box.
- Segger Embedded Studio
    - Download and install Embedded Studio.

Step 4: Write a program or open an example program

Open an example program

- The only example programs I could find were located in the STM32G0 SDK and then were specific to a few of ST’s development boards. I can’t guarantee that they will work for our MCU, but there are some examples for the “NUCLEO-STM32G031K8” which is close enough to our MCU that it seems like they should work without too much effort.
- First, download the STM32G0 SDK (called “STM32CubeG0”).
- The examples for the Nucleo board mentioned above are in /Projects/NUCLEO-STM32G031K8 of the downloaded folder.
- For STM32CubeIDE, the example files that you’ll want to import or open are located in the folder “SW4STM32”. Each separate example has a folder with this name inside of it.
- For Keil MDK-Arm and Segger Embedded Studio, the example files that you’ll want to import or open are located in the folder “MDK-ARM”. Each separate example has a folder with this name inside of it.

Write a program

- WARNING: It seems possible to configure pins 4, 7, and 8 of the STM32G031J6 as GPIOs or other peripherals, even with the Device Configuration Tool in STM32CubeIDE. HOWEVER:
    - Pins 7 and 8 are used for the SWD interface that programs the MCU. Configuring these pins as anything other than the default prevents you from debugging or reprogramming your MCU. The only way to reload a new program in that scenario is to perform a full Flash erase (see below) of the device.
    - Pin 4 is the MCU's reset pin. Ostensibly you can configure this as a GPIO or other peripheral. However, I couldn't achieve that functionality. When I tried to do so, the pin seemed inoperable and, in every case but one, wouldn't even function to reset the MCU. I would caution against using this for any alternate function, even if you manage to figure out how to do so, since having a functional reset pin is CRITICAL to performing a Flash erase of the MCU and in the unfortunate event that you accidentally reconfigure both the reset pin and the SWD pins for another peripheral, it is my understanding that there is NO way to reprogram that MCU.
    - Performing a Flash erase of the STM32G031:
        - Using ST-Utility (also known as "STSW-LINK004")
            - Download and install the ST-Utility.
            - Connect the STM32G0 to your ST-Link and the ST-Link to your computer.
            - Open up the ST-Utility.
            - Click on "Target" > "Settings...".
            - Change "Mode" to "Connect under reset".
            - Change "Reset mode" to "Hardware reset" (if it didn't change automatically).
            - Click "Cancel" (This sounds counterintuitive but in my testing the modified settings are still saved. Clicking "Okay" is also fine, you might just get an error message about ST-Utility not being able to connect to the MCU.)
            - Hold the STM32G0 in reset.
            - Press the "Flash erase" button (icon shaped like a stick of dynamite). (It may also work to click "Target" > "Erase chip" or to press Ctrl+E [shortcut for "Erase chip"]).
            - Click "Okay" on the warning dialog box that comes up.
            - Immediately release the MCU from reset.
        - Using STM32CubeProgrammer CLI
            - Download and install the STM32CubeProgrammer. The following steps only require the command line interface (CLI) for this program, but there is a GUI also. The CLI worked for me on Linux

immediately after installation. However, to get the GUI to work I had to follow the steps in section 1.2.1 of the STM32CubeProgrammer User's Manual (UM2237).
    - Connect the STM32G0 to your ST-Link and your ST-Link to your computer.
    - Open up a terminal window from the folder $INSTALL_DIR/STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/.
    - Hold the STM32G0 in reset.
    - Run either of the following commands:
        - ./STM32_Programmer_CLI -c port=swd -e all
        - ./STM32_Programmer_CLI -c port=swd mode=UR reset=HWrst
    - Immediately after executing those commands, release the STM32G0 from reset.
- STM32CubeIDE
    - Click "File" > "New" > "STM32 project".
    - Use the search bar in the upper-left quadrant of the newly opened window to find "STM32G031J6". Select it from the "MCU/MPUs" list and click "Next".
    - Give your project a name, leaving the remaining options as they are, and click "Finish".
    - Write your application code, or use the Device Configuration Tool to create code for you (see "STM32CubeIDE Device Configuration Tool" under "Software Development Tools", below).
- Keil MDK-Arm
    - If you've already exported a project from STM32CubeMX for MDK-Arm, then import that by clicking "Project" > "Open project" and navigating to the MDK-Arm project file created by STM32CubeMX. This was how I was able to work with the STM32G031J6 in MDK-Arm, since the process below (probably the preferred one) didn't work for me.
    - Otherwise, start a new project by doing the following.
        - Ensure that you have STM32CubeMX downloaded along with the latest version of Java (STM32CubeMX will prompt you during installation if you need to update and provide you with a valid hyperlink to download that).
        - Open up the Pack Installer and search for "STM32G031J6" in the search bar on the left-hand side of the newly opened window. Ensure that, at a minimum, you have installed a device-specific pack for the STM32G0; mine was called "Keil::STM32G0xx_DFP".
        - Start a new project by clicking "Project" > "New uVision project".
        - Select a folder location and project name and click "OK".
        - In the next window, search or navigate the menu tree until you can select "STM32G031J6Mx" and click "OK".

        - In the next window, called “Manage Runtime Environment”, you’ll select the SDK components that you want to include in your project.
            - Clicking the check box next to “Device” > “STM32Cube Framework (API)” > “STM32CubeMX” is the easiest way to do this.
            - Otherwise, you’ll need to add individual components to make a working project.
        - Configure your project using STM32CubeMX as described above, then click “Generate Code”.
            - Right before launching, STM32CubeMX warned me about not running 64-bit Java (I was only running 32-bit since I was on a Windows 10 VM) and then wouldn’t let me generate code. Perhaps you’ll have better luck.
    - Lastly, open up “Target options” (icon looks like a magic wand over top of three small boxes) and go to the “Debug” tab.
    - Select the debug adapter you’re using (ST-Link or J-Link) and click “Settings”.
    - Make sure that MDK-Arm sees your debug adapter. You may need to change the port to “SW” or click “Scan” (if you’re using a J-Link) to do this.
    - Once connected, go to the tab called “Flash download”.
    - If there is no programming algorithm listed, click “Add” and find the listing that matches the STM32G031J6 most closely. For me, that was the listing with the following information:
        - Description: STM32G0xx 32 KB Flash
        - Flash size: 32k
        - Device type: On-chip flash
        - Origin: Device family package
    - Click “Add” to return to the “Flash download” tab.
    - Make sure the box next to “Reset and run” is checked (this will allow the J-Link to reset our MCU after programming, instead of requiring us to manually reset it). This worked fine with my J-Link, but I couldn’t get it to work with my ST-Link. I could reset the application just fine from inside a debugging session, but after each time I downloaded code to the MCU I needed to perform a manual reset.
    - Return back to the main project.
- Segger Embedded Studio
    - The recommended way to start a project for STM32G031J6 in Embedded Studio is to initialize the pin and project settings using STM32CubeMX (see below).
        - After the project has been created, click “File” > “Import Project” > “Import Keil MDK Project”.
    - If you wish to not use STM32CubeMX and want to start a native project in Embedded Studio, perform the following steps.
        - 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 “STM32G031J6” 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 the STM32G0 interface library, STM32CubeG0. All of the files you will need are located within the downloaded folder, but you’ll have to dig for them.
- Text editor
    - Initialize the pin and project settings using STM32CubeMX, with the toolchain set to “makefile” (see below).

Software Development Kits[17]

- STM32CubeIDE Device Configuration Tool
    - Double-click “[PROJECT_NAME].ioc” from the “Project Explorer”
    - Configure each pin under “Pinout & Configuration” > “Pinout view” by clicking the pin and selecting the appropriate function.
    - Alternatively, select the peripheral you’d like to configure from the list on the left-hand side of the window.
    - Right-click each pin to give it a unique name.
    - Configure the clock speed and routing under the “Clock Configuration” tab.
    - When you’re finished, click the “Generate code” button from the toolbar (icon of a spoke and gear in between the “Build All” and “Debug” buttons).
    - I ran into build errors if I tried to change the functionality of a pin after generating code without changing the pin’s name. The only workaround I found was to rename the pin.
    - After generating code has completed, you’ll need to look at “main.h” and inside the “Drivers/STM32G0xx_HAL_Driver” folder of your project to identify the API you’re going to use. For instance, after I configured one of my pins as an output, I found pin and port definitions in “main.h” and functions like “HAL_GPIO_TogglePin” and “HAL_GPIO_ReadPin” in “Drivers/STM32G0xx_HAL_Driver/Src/stm32g0xx_hal_gpio.c”.

[17] Software Development Kits include any piece of software that makes it easier for the developer to write application code. Many such SDKs exist, so the kind focused on here relates to managing the MCU hardware (startup code, MCU core, peripherals, etc). This is typically the first thing a developer needs to start working with their target MCU. Some common examples of this type of SDK include makefile or IDE-specific project configuration files, header files for referencing the MCU registers, or a HAL (hardware abstraction layer) which is sometimes called low-level driver. Any of these components may be made available either directly as source code or indirectly through a GUI.

    - Alternatively (and probably more efficiently), you could peruse the user manuals provided here. These documents provide a description of how to use this SDK. For instance, the HAL and LL drivers (ADC, GPIO, etc) are described in UM2319 and use of the USB device library is described in UM1734. Many names (for functions, structs, etc) are hyperlinked to make navigation slightly easier than with a standard PDF.
- STM32CubeMX
    - Download and install STM32CubeMX.
    - Open the program.
    - Click “File” > “New project” or the “Access MCU selector” button under “I need to…” > “Start my project from MCU”.
    - Use the search bar in the upper-left quadrant of the newly opened window to find “STM32G031J6”. Select it from the “MCU/MPUs” list and click “Start Project”.
    - Follow the steps above for “STM32CubeIDE Device Configuration Tool” to configure your project.
        - Before clicking “Generate code”, click on “Project Manager” and select the appropriate “Toolchain/IDE” option in the “Code Generator” field.
        - “STM32CubeIDE” will generate the .ioc file that was mentioned previously which STM32CubeIDE uses to configure project settings.
        - “MDK-Arm” will generate project files for Keil MDK-Arm and Segger Embedded Studio.
        - “makefile” will generate a makefile-based build suitable for a command-line build or for importing to IDEs such as Eclipse or CLion.
- STM32CubeG0
    - The full library for the code generated by STM32CubeMX is also available for download here. However, since the GUI is capable of producing makefiles from STM32CubeMX projects and since it is so much more user-friendly than the library source files, I would discourage using this library directly. Instead, create a project in STM32CubeMX and change the project settings to export to a makefile.
- libopencm3
    - Download libopencm3 from git. It’s easiest if this ends up being placed in or near your project folder.
    - From the main libopencm3 folder, run “make TARGETS=stm32/g0”
    - To use libopencm3 in your projects, refer to the online API documentation here. Click on the link at either the top or left-hand side of the page for “STM32G0”. The ensuing page describes all of the functions available to you via this library.
    - However, the documentation seems to lack crucial details, like mentioning that peripheral clocks need to be enabled before they will work. What helped me was to follow along with the book “Beginning STM32” by Warren Gay, which documents how to use libopencm3 for the STM32F103C8T6. The API isn’t exactly the same for the G0 as for the F1, but it’s close enough to serve as a starting point. He provides a git repo with source code for the book here.

Step 5: Compile the program

- This will simply compile the program without downloading it to your STM32G0 or starting a debug session.
    - To compile the program and download it to your STM32G0, see “Flash the program to the MCU” below.
    - To compile the program, download it to your STM32G0, and start a debug session, see “Start a debug session” below.
- STM32CubeIDE
    - Click one of the build options under “Project”, or
    - Press Ctrl + B (shortcut for “Build All”), or
    - Click the icon in the shape of a hammer, or
    - Click the icon in the shape of a page with “010” in front of it (shortcut for “Build All”), or
    - Right click the project folder in the “Project Explorer” window and click “Build Project”
- Keil MDK-Arm
    - Click “Project” > “Build target” or
    - Press F7 (hotkey for “Build target”) or
    - Click the “Build target” icon (shape of a single downward pointing blue arrow overtop a square) or
    - Click “Project” > “Rebuild all target files” or
    - Click the “Rebuild” icon (shape of two downward pointing blue arrows overtop a square)
- 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)
- Text editor
    - **Options:**
        - **Using STM32CubeMX**
        - **Using libopencm3**
    - STM32CubeMX
        - If you generated your project using STM32CubeMX and specified the toolchain as “makefile”, then all you need to do to recompile your code is open a terminal window in the project’s root folder and run “make”.
    - libopencm3
        - Compilation
            - Use gcc.

        - Make sure to specify the libopencm3 “include” directory in the shell command (for me, “-I../libopencm3-master/include”) and to define the symbol “STM32G0” (“-DSTM32G0”).
        - The following command worked for me: “arm-none-eabi-gcc -Os -g -mthumb -c -DSTM32G0 -I../libopencm3-master/include miniblink.c”
    - Linking
        - Use either gcc or ld.
        - Make sure to specify the location and filename of the libopencm3 library (for me, “-L../libopencm3-master/lib -lopencm3_stm32g0”) and the location of the linker script (for me, “-T../STM32G031J6Mx_FLASH.ld”).
        - libopencm3 has a generic linker file for Cortex-M devices, but it requires another linker file to specify the location and size of the flash memory and RAM. I can think of 2 ways to get this:
            - Preferred method: Use STM32CubeMX to create a makefile project. Doing this created a file called “STM32G031J6Mx_FLASH.ld” in the project’s root folder.
            - Look elsewhere in the libopencm3 folder for an example of what this file needs to look like and copy it. Edit the flash location and size to be 0x08000000 and 32K and the RAM location and size to be 0x20000000 and 8K[18].
        - The following command worked for me: “arm-none-eabi-ld -Map miniblink.map -T../STM32G031J6Mx_FLASH.ld miniblink.o -L../libopencm3-master/lib -lopencm3_stm32g0 -N -o miniblink.elf”
    - Convert the .elf file to a .bin: “arm-none-eabi-objcopy -Obinary miniblink.elf miniblink.bin”

Step 6: Flash the program to the MCU

- This will compile your program and download it to your STM32G0.
    - To simply compile your program without downloading it to your STM32G0, see “Compile the program” above.
    - To compile the program, download it to your STM32G0, and start a debug session, see “Start a debug session” below.
- STM32CubeIDE
    - To the best of my knowledge, STM32CubeIDE 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.

[18] The flash location and the RAM location and size are located in the STM32G0x1 reference manual in Table 3 on page 56. This table would lead us to believe that the flash size for our device is 64K, but it’s fairly clear from the subtitle on the product page for the STM32G031J6 that our device only has 32K of flash.

- Keil MDK-Arm
    - Click “Flash” > “Download” or
    - Press F8 (hotkey for “Download”) or
    - Click the “Download” icon (shape of two downward pointing blue arrows underneath the word “LOAD”)
- Segger Embedded Studio
    - Click “Target” > “Download [PROJECT NAME]” or
    - Click “Build” > “Build and Run”
- Text editor
    - **Options:**
        - **J-Flash (executable)**
        - **Ozone**
        - **J-Link GDB server & GDB**
        - **ST-Link GDB server & GDB**
        - **STM32CubeProgrammer**
    - 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 (STM32G031J6).
        - 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 0x08000000.
        - Click “Program Device”.
        - There is also a J-Flash CLI.
    - Ozone (to flash the program to the MCU from INSIDE a debugging session)
        - See below.
    - J-Link GDB Server & GDB (to flash the program to the MCU from INSIDE a debugging session)
        - See below.
    - ST-Link GDB Server & GDB (to flash the program to the MCU from INSIDE a debugging session)
        - See below.
    - STM32CubeProgrammer
        - Not tested.

Step 7 (Optional): Start a debug session

- This will compile your program, download it to your STM32G0, and start a debug session.

    - To simply compile your program without downloading it to your STM32G0, see “Compile the program” above.
    - To simply compile the program and download it to your STM32G0, see “Flash the program to the MCU” above.
- STM32CubeIDE
    - Click the “Debug” button (the icon looks like a green bug), or
    - Press F11 (shortcut for “Debug”), or
    - Click “Run” > “Debug”, or
    - Click “Run” > “Debug As” > “Local C/C++ application”, or
    - Click “Run” > “Debug As” > “STM32 MCU C/C++ application”, or
    - Right-click on the project in the “Project Explorer” and select “Debug As” > “Local C/C++ application”, or
    - Right-click on the project in the “Project Explorer” and select “Debug As” > “STM32 MCU C/C++ application”
- Keil MDK-Arm
    - Click “Debug” > “Start/stop debug session” or
    - Press Ctrl+F5 (hotkey for “Start/stop debug session”) or
    - Click the “Start/stop debug session” icon (shape of magnifying glass with a small, red, lowercase “d” in the middle)
- Segger Embedded Studio
    - Click “Build” > “Build and Debug” or
    - Click “Debug” > “Debug with Ozone” or
    - Press Alt+F5 (“Debug with Ozone” hotkey)
- Text editor
    - **Options:**
        - **Ozone**
        - **J-Link GDB server & GDB**
        - **ST-Link GDB server & GDB**
    - Ozone
        - Download, install, and run Ozone.
        - In the window that pops up, select the “STM32G031J6” 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 J-Link 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 “STM32G031J6” under “Target device” and set “Target interface” to SWD.

        - If you're connecting to your J-Link 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 "STM32G031J6 (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
    - ST-Link GDB server & GDB
        - Download and install STM32CubeIDE. This includes the ST-Link GDB server as well as the STM32CubeProgrammer (which is required to run the GDB server).
        - Locate the ST-LINK_gdbserver executable and the STM32_Programmer_CLI executable. Hints about where to look can be found in section 1.3 of the STM32CubeIDE ST-LINK GDB server User Manual (UM2576).
        - Start the ST-Link GDB server with, at a minimum, "-cp LOCATION_OF_STM32_PROGRAMMER_CLI_EXECUTABLE". You may also need to specify that the format is "SWD" ("-d"). The example below also include "-v", meaning that the output should be verbose.
        - The following worked for me (I had installed STM32CubeProgrammer separately and directed the GDB server to that install location, instead of the one inside STM32CubeIDE, where the ST-LINK_gdbserver executable was located): ./ST-LINK_gdbserver -d -v -cp "/home/nathancharlesjones/STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin"
        - Look at the output for the port on which the server is open. The default is 61234.
        - 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 :61234

Helpful tools

- STM-STUDIO-STM32: Data visualization tool for STM32 (sort of like uC-Probe)
- ST-Utility (also known as “STSW-LINK004”)
- 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 STM32G031J6
- Datasheet for STM32G031 (DS12992)
- Software Development Tools for STM32 and STM8 MCUs
- Embedded Software Libraries for STM32 and STM8 MCUs
- STM32 Microcontroller Debug Toolbox (AN4989)
- Getting started with STM32G0 Series hardware development (AN5096)
- How to reset the STM32G0 for reprogramming if your application code requisitioned the SWDIO/SWDCLK pins (NewbieHack)
- Making the open-source STM32 adapter driver "st-utility" (by texane) work with STM32G0
- Using Keil MDK with STM32CubeIDE (AN323)
- Description of STM32G0 HAL and low-layer drivers (UM2319)
- STM32CubeIDE ST-LINK GDB server User Manual (UM2576)
- Description of STM32G0 HAL and low-layer drivers User Manual (UM2319)

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
STM32	STM32G031J6M6	Self-built	ST-Link	STM32CubeIDE	STM32CubeMX	STM32CubeIDE	STM32CubeIDE	STM32CubeIDE	Yes	Ubuntu 18.04.03	Almost bricked the MCU when I reprogrammed the SWDCLK pin to be a GPIO. Succeeded by erasing the entire MCU and changing pins.
STM32	STM32G031J6M6	Self-built	J-Link EDU mini	Text editor	STM32CubeMX	gcc	Ozone	Ozone	Yes	Ubuntu 18.04.03	Used STM32CubeMX to create a project and exported to a makefile. Added my two lines of code to toggle an LED and then loaded using Ozone. Incredibly easy. Hardest part is learning the API.
STM32	STM32G031J6M6	Self-built	J-Link EDU mini	Text editor	libopencm3	gcc	Ozone	Ozone	Yes	Ubuntu 18.04.03	Very difficult to navigate libopencm3 and configure for new target without the help of Warren Gay's git repo (for F1). Some functions change between families so I'm not sure anymore how useful the library would be.
STM32	STM32G031J6M6	Self-built	J-Link EDU mini	Embedded Studio	STM32CubeMX	gcc	J-Flash	Ozone	Yes	Ubuntu 18.04.03	Built project in STM32CubeMX, exported as Keil MDK-Arm project, and imported that project into Embedded Studio. Downloading worked fine from Embedded Studio but I needed to switch to Ozone ("Debug" > "Debug with Ozone" or Alt+F5) to be able to debug.
STM32	STM32G031J6M6	Self-built	ST-Link	Keil MDK-Arm	STM32CubeMX	Keil MDK-Arm	Keil MDK-Arm	Keil MDK-Arm	Yes	Windows 10 VM	Needed to explicitly add programming algorithm. ST-Link would download and debug just fine, but could not reset target after download.
STM32	STM32G031J6M6	Self-built	J-Link EDU mini	Keil MDK-Arm	STM32CubeMX	Keil MDK-Arm	Keil MDK-Arm	Keil MDK-Arm	Yes	Windows 10 VM	Needed to explicitly add programming algorithm. ST-Link would download and debug just fine, but could not reset target after download.
STM32	STM32G031J6M6	Self-built	ST-Link	Text editor	STM32CubeMX	gcc	ST-LINK_gdbserver	gdb	Yes	Ubuntu 18.04.03	