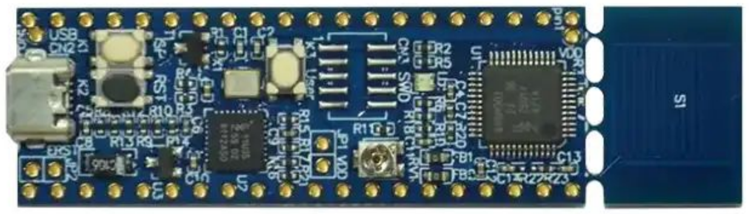
Getting Started with the LPC845-BRK

Step 1: Procure the LPC845-BRK development board[1]



Step 2 (Optional): Pick a different debug adapter

- **Options:**
  - **CMSIS-DAP (on-board)**
  - **J-Link**
  - **Black Magic Probe**
- The biggest advantage of the on-board CMSIS-DAP debug adapter (other than being on-board) is that it is trace enabled, meaning that in addition to being able to set breakpoints and step through code, the CMSIS-DAP debug adapter will capture every single instruction that's executed by the MCU and provide that log to the developer. This is useful, for instance, when you're attempting to debug why your MCU keeps finding its

[1] https://www.digikey.com/products/en?keywords=lpc845-brk

way to the HardFault Handler. By setting a breakpoint in the HardFault Handler, the program will halt as soon as it's reached and the trace output will show you exactly which instructions were being executed immediately prior to the HardFault Handler being called. The MCUXpresso SDK for the LPC845 (see below) also provides modules to easily set up semihosting, allowing for bi-directional communication between the MCU and any debug software that supports semihosting.

- The Segger J-Link is available for around $20[2] and is also a powerful debug adapter. The biggest reasons you might want to switch to a J-Link are to maintain consistency with other MCUs that you're developing using the J-Link or to use Segger's Real-Time Transfer (RTT)[3] protocol (allows bi-directional serial transfers over the J-Link).
    - You'll also need either a surface-mount connector to solder onto CN3 ("suggested part number Samtec FTSH-105-01-L-DV-K" from section 4.1 of UM11181: User Manual for LPC845 Breakout board[4]) or a J-Link breakout board ($2 from Adafruit[5]) to be able to connect the J-Link to the LPC845-BRK.
    - If you're using the J-Link breakout board, make the connections below. You can double-check which pins these are by viewing the schematic for the LCP845-BRK[6].
        - SWIO <-> PIO0_02 (pin 37)
        - CLK <-> PIO0_03 (pin 36)
        - nRST <-> PIO0_05 (pin 34)
        - Vref <-> VDD (pin 40)
        - GND <-> GND (pin 20)
    - In either case, the board will need to be externally powered to work, since the J-Link will not supply power to the board. You can use the micro USB connector on the LPC845-BRK to provide power or you can use a breadboard power supply [7].
    - Lastly, make sure to solder a pin header at JP2 and to place a jumper across it, to hold the on-board CMSIS-DAP debug adapter in reset (from section 4.1 of UM11181: User Manual for LPC845 Breakout board[8]).
- The Black Magic Probe[9] is another powerful and also completely open-source debug adapter. The biggest reason you might want to use a Black Magic Probe is to be able to debug and also have a serial terminal open on the same device (similar to using a J-Link with RTT) or to use an IDE that might not work with CMSIS-DAP or J-Link (since the Black Magic Probe runs it's own GDB server in its firmware). The Black Magic Probe can be bought for around $60[10] or you can "build your own", so to speak: the Black Magic

[2] From Adafruit (https://www.adafruit.com/product/3571)
[3] https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/
[4] https://www.nxp.com/webapp/Download?colCode=UM11181
[5] https://www.adafruit.com/product/2743
[6] https://www.nxp.com/downloads/en/schematics/SCH_LPC845_BRK.zip
[7] One option here (https://www.digikey.com/short/z3wb7z).
[8] https://www.nxp.com/downloads/en/schematics/SCH_LPC845_BRK.zip
[9] https://github.com/blacksphere/blackmagic/wiki
[10] From 1BitSquared (https://1bitsquared.com/products/black-magic-probe)

Probe firmware can be flashed to an STM32F103 (which is available for around $10 on Mouser[11]), flashed to an ST-Link (which are available for around $8[12]), run from your host computer which connects to your target via an FT232 protocol adapter (which are available for around $15 from Digi-Key[13]), or others. You can learn more about which targets are compatible with the Black Magic Probe on their "Hardware"[14] page and more about how to build the proper firmware from the "Hacking"[15] page.

Step 3: Pick an IDE/code editor

- **Options (all are IDEs unless otherwise noted):**
  - **MCUXpresso**
  - **Keil MDK-Arm**
  - **IAR Embedded Workbench**
  - **Segger Embedded Studio**
  - **Your favorite text editor (NOT an IDE)**
- MCUXpresso
  - Download the version of MCUXpresso[16] for your OS.
  - Follow the installation instructions found in the MCUXpresso IDE Installation Guide[17].
- Segger Embedded Studio
  - Download and install Embedded Studio[18].

Step 4: Write a program or open an example program

Open an example program

- Go to the NXP MCUXpresso SDK Builder[19] and download the SDK for the LPC845.
- In the SDK folder, demo apps, driver examples, and a program template can be found in /boards/lpc845breakout/. Each one (except the program template) has a "readme.txt" file with instructions about how to load it with common IDEs such as IAR Embedded Workbench, Keil MDK-Arm, and NXP MCUXpresso. It also seems possible to compile

[11] https://www.mouser.com/ProductDetail/ROBOTIS/902-0084-010?qs=sGAEpiMZZMtw0nEwywcFgJjuZv55GFNmg73WEmdgwPMTdTT6QQZQ1w%3D%3D
[12] From Seeed (https://www.seeedstudio.com/Sipeed-USB-JTAG-TTL-RISC-V-Debugger-p-2910.html) or Adafruit (https://www.adafruit.com/product/2548)
[13] https://www.digikey.com/product-detail/en/2264/1528-1449-ND/5761217/?itemSeq=320611313
[14] https://github.com/blacksphere/blackmagic/wiki/Debugger-Hardware
[15] https://github.com/blacksphere/blackmagic/wiki/Hacking
[16] https://nxp.flexnetoperations.com/control/frse/download?element=11239507
[17] https://www.nxp.com/docs/en/quick-reference-guide/MCUXpresso_IDE_Installation_Guide.pdf
[18] https://www.segger.com/products/development-tools/embedded-studio/
[19] https://mcuxpresso.nxp.com/en/welcome

these programs using the Makefile setup (included below in the “Blinky_v3” project) and to download and debug them using one of the text editor options listed below.
- See also the PDF titled “Getting Started with MCUXpresso SDK” (located in /docs) and the online API documentation for the MCUXpresso SDK[20].

Write a program

- MCUXpresso
    - Go to the NXP MCUXpresso SDK Builder[21] and download the SDK for the LPC845.
    - From the PDF titled “Getting Started with MCUXpresso SDK” (located in /docs): “Drag and drop the SDK zip file into the “Installed SDKs” view to install an SDK. In the window that appears, click the “OK” button and wait until the import has finished.”
    - Click “File” > “New” > “Project”.
    - Select “New C/C++ Project” (with the MCUXpresso logo) and click “Next”.
    - Type “LPC845” into the “Available boards” search bar and select the “lpc845breakout” board. Click “Next”.
    - Give your project a unique name and leave all default options the same (unless you know what you’re doing). Ensure all checkboxes are checked under “Drivers” to add support for each peripheral into your project. Click “Finish”.
    - Certain default functions are set up for you at this point, including manipulating the LEDs on the development board and utilizing the UART. “board.h” (located in the folder /board/boards) has many macros defined which give you an idea of how to quickly use some of these functions. For instance, I was able to easily add “LED_BLUE_INIT(LOGIC_LED_OFF);” at the start of my main and “LED_BLUE_TOGGLE();” inside of it (along with a hasty for-loop delay) to start turning the blue LED on and off.
    - Add your code to either the main file or another file placed inside the “source” folder, and any headers in “Includes”.
    - See also the PDF titled “Getting Started with MCUXpresso SDK” (located in /docs) and the online API documentation for the MCUXpresso SDK[22] for more information about how to use the MCUXpresso SDK.
- Segger Embedded Studio
    - Follow the instructions above to get and connect a Segger J-Link.
    - 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”.

[20] https://mcuxpresso.nxp.com/api_doc/dev/1156/
[21] https://mcuxpresso.nxp.com/en/welcome
[22] https://mcuxpresso.nxp.com/api_doc/dev/1156/

- Enter “LPC845M301” 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 SDK for the LPC845 from the NXP MCUXpresso SDK Builder[23].
- Add the files to your project that you specifically want to use (or you could just add them all).
    - All of the actual API files you’ll need should be in /devices/LPC845/drivers, but you’ll also need some setup files in /CMSIS and elsewhere in /devices/LPC845/utilities.
    - See also the PDF titled “Getting Started with MCUXpresso SDK” (located in /docs) and the online API documentation for the MCUXpresso SDK[24].
    - 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, to be able to use GPIO_PortInit(), GPIO_PinInit(), and GPIO_PortToggle() (located in “fsl_gpio.c” and “fsl_gpio.h”), I needed to include the following files:
        - fsl_clock.c/.h
        - fsl_common.c/.h
        - fsl_device_registers.h
        - fsl_reset.c/.h
        - LPC845.h
        - LPC845_features.h
        - core_cm0plus.h
        - cmsis_version.h
        - cmsis_gcc.h
        - cmsis_compiler.h
        - LPC845_flash.ld
        - startup_LPC845.S
        - system_LPC845.c/.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 “fsl_device_registers.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.
        - Right-click your project’s name under “Project Items”
        - Select “Options”

[23] https://mcuxpresso.nxp.com/en/welcome
[24] https://mcuxpresso.nxp.com/api_doc/dev/1156/

        - Under the pull-down menu, select “Public configurations” > “Common”
        - Select “Preprocessor”
        - Click the three dots next to “Preprocessor definitions”
        - Add “CPU_LPC845M301JBD48” on its own line in the text box that comes up.
        - “OK” out of everything back to your project.
- Text editor
    - Copy the “Blinky_v3” folder. Put your source code in “source” and your headers in “include” (the “CMSIS”, “components”, and “devices” folders were copied directly from the SDK for the LPC845[25]).
    - You could also download the SDK for the LPC845[26]. Copy the “CMSIS”, “components”, and “devices” folders from the SDK and the Makefile from “Blinky_v3” into your project. Don’t forget to double-check and update all files and file paths.
    - See the PDF titled “Getting Started with MCUXpresso SDK” (located in /docs) and the online API documentation for the MCUXpresso SDK[27] for information about which functions are available to you and how to use them.

Software Development Kits[28]

- NXP MCUXpresso SDK Builder[29]
    - Clink “Select development board” on the page above to get started building the proper SDK for your build environment. That page also has links to an online MCUXpresso Config Tool[30] and what appears to be an offline version of the MCUXpresso SDK Builder[31].
    - See the PDF titled “Getting Started with MCUXpresso SDK” (located in /docs) and the online API documentation for the MCUXpresso SDK[32] for information about which functions are available to you and how to use them.

[25] https://mcuxpresso.nxp.com/en/welcome
[26] https://mcuxpresso.nxp.com/en/welcome
[27] https://mcuxpresso.nxp.com/api_doc/dev/1156/
[28] 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 a low-level driver. Any of these components may be made available either directly as source code or indirectly through a GUI.
[29] https://mcuxpresso.nxp.com/en/welcome
[30] https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools/mcuxpresso-config-tools-pins-clocks-peripherals:MCUXpresso-Config-Tools?tab=Design_Tools_Tab
[31] http://boards/lpc845breakout/
[32] https://mcuxpresso.nxp.com/api_doc/dev/1156/

    - Demo apps, driver examples, and a program template can be found in the downloaded SDK folder in /boards/lpc845breakout/. Each one (except the program template) has a “readme.txt” file which describes its basic function. It also seems possible to compile these programs using the Makefile setup (included below in the “Blinky_v3” project) and to download and debug them using one of the text editor options listed below.

Step 5: Compile the program

- This will simply compile the program without downloading it to your LPC845 or starting a debug session.
    - To compile the program and download it to your LPC845, see “Flash the program to the MCU” below.
    - To compile the program, download it to your LPC845, and start a debug session, see “Start a debug session” below.
- MCUXpresso
    - Click “Project” > “Build All”, or
    - Press Ctrl + B (shortcut for “Build all”), or
    - Click “Project” > “Build project”, or
    - Click “Build” (in the “Quickstart” panel)
- 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
    - If you’ve copied the “Blinky_v3” folder, run “make” to build (all of the output files will go into the “build” folder; the executables are named “Blinky_v3”). Run “make clean” to delete the “build” folder.
    - Alternatively, you could look in the “Blinky_v2” folder for a very minimal implementation. This folder only includes the support code you’ll need to blink an LED. You can build this project by running either “make” (in which case all of the output files will go into the “build” folder; the executables are named “Blinky_v2”) or running “./Build-minimal-for-LPC845.sh” (in which case all of the output files will go into the “output-of-Build-minimal-for-LPC845” folder; the executables are named “main”). Run “make clean” or “./Clean-minimal-for-LPC845.sh” to remove the appropriate output folder.
    - An explanation of the “Build-minimal-for-LPC845” script is below.
        - From a shell in the same directory as the project you’re trying to build, run the following gcc compilation command: GCC -DCPU_LPC845M301JBD48 -mthumb -mcpu=cortex-m0 -Os -g -c -I ./ main.c clock_config.c fsl_clock.c fsl_reset.c fsl_gpio.c system_LPC845.c fsl_common.c fsl_power.c startup_LPC845.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 "fsl_gpio.c" and "fsl_gpio.h", and their dependencies. You should include whatever other source code you need for your 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 my folder:
            - clock_config.h
            - pin_mux.h
            - fsl_power.h
            - fsl_clock.h
            - fsl_common.h
            - fsl_device_registers.h
            - fsl_reset.h
            - LPC845.h
            - LPC845_features.h
            - core_cm0plus.h
            - cmsis_version.h
            - cmsis_gcc.h
            - cmsis_compiler.h
            - system_LPC845.h
    - Then run the following gcc linker command: GCC -Wl,-Map=main.map -specs=nosys.specs -mthumb -mcpu=cortex-m0 -T LPC845_flash.ld main.o clock_config.o fsl_clock.o fsl_reset.o fsl_gpio.o system_LPC845.o fsl_common.o fsl_power.o startup_LPC845.o -o main.elf
        - Make sure the linker script, "LPC845_flash.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").
    - I put these three shell commands into a file called "Compile-for-LPC845.sh" so, alternatively, you could edit that script (to make sure it correctly points to your ARM GCC binaries and also your project files) and simply run that script, instead of each of the above commands individually. There's a smarter way to do this with make, but I'm not quite that cool yet.

Step 6: Flash the program to the MCU

- This will compile your program and download it to your LPC845.
    - To simply compile your program without downloading it to your LPC845, see “Compile the program” above.
    - To compile the program, download it to your LPC845, and start a debug session, see “Start a debug session” below.
- NXP MCUXpresso
    - To the best of my knowledge, MCUXpresso 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”
- Text editor
    - **Options:**
        - **J-Flash (executable)**
        - **Ozone**
        - **J-Link GDB server & GDB/gdbgui**
        - **OpenOCD GDB server & GDB/gdbgui**
    - 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 (LPC845M301JBD48).
        - 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 0x00000000.
        - 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/gdbgui (to flash the program to the MCU from INSIDE a debugging session)
        - See below.
    - OpenOCD GDB server & GDB/gdbgui (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 LPC845, and start a debug session.
    - To simply compile your program without downloading it to your LPC845, see “Compile the program” above.
    - To simply compile the program and download it to your LPC845, see “Flash the program to the MCU” above.
- NXP MCUXpresso
    - Click “Debug” (in the “Quickstart” panel) and select your debug adapter in the window that opens up, or
    - Click “Run” > “Debug last launched”, or
    - Press F11 (shortcut for “Debug last launched”), or
    - Click “Run” > “Debug as” > (select your debug adapter)
- 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/gdbgui**
        - **OpenOCD GDB server & GDB/gdbgui**
    - Ozone
        - Download, install, and run Ozone[33].
        - In the window that pops up, select the LPC845M301JBD48 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 file for your program and click “OK”.
    - J-Link GDB Server & GDB
        - Find and run the file “JLinkGDBServerExe”.
        - Select “LPC845M301JBD48” under “Target device” and set “Target interface” to SWD.
        - If you’re connecting to your LCP845-BRK with a USB cable, ensure that “USB” is selected under “Connection to J-Link”.

[33] https://www.segger.com/products/development-tools/ozone-j-link-debugger/

        - 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 “LPC845M301 (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
            - Press Ctrl-x-a from inside GDB to start TUI mode.
        - Alternatively, you could install and run gdbgui, if you prefer a GUI to a CLI (It invokes "gdb" by default, so don't forget to point it to the right one on start-up (e.g. "gdbgui -g arm-none-eabi-gdb").
    - OpenOCD GDB server & GDB
        - Download OpenOCD[34]. I also needed HIDAPI[35] on my Linux machine to be able to use OpenOCD with CMSIS-DAP (see “Compiling And Installing OpenOCD With CMSIS-DAP Support”[36] from Hash Define Electronics for more information).
        - Run one of the commands below based on which debug adapter you’re using (notice that the “interface” configuration file is the only thing that changes between the commands for the two debug adapters):
            - J-Link
                - openocd -f path-to-OpenOCD/tcl/interface/jlink.cfg -c "transport select swd" -f path-to-OpenOCD/tcl/target/lpc84x.cfg -c "cortex_m reset_config sysresetreq" -c "adapter_khz 4000"
                - Or simply run “./OpenOCD_and_J-Link.sh”
            - CMSIS-DAP
                - openocd -f path-to-OpenOCD/tcl/interface/cmsis-dap.cfg -c "transport select swd" -f path-to-OpenOCD/tcl/target/lpc84x.cfg -c "cortex_m reset_config sysresetreq" -c "adapter_khz 4000"
                - Or simply run “./OpenOCD_and_CMSIS-DAP.sh”

[34] https://sourceforge.net/p/openocd/code/ci/master/tree/
[35] https://github.com/libusb/hidapi
[36] https://www.hashdefineelectronics.com/compiling-and-installing-openocd-with-cmcsis-dap-support/

        - Once OpenOCD starts a server, in a different shell, start GDB (i.e. /path/to/gdb /path/to/executable.elf).
        - Connect GDB to the OpenOCD session with “target remote :3333”.
        - Load your code onto the LPC845 with “load /path/to/executable.elf”.
        - Type “mon reset halt” and press Enter.
        - Use GDB normally.
            - Press Ctrl-x-a from inside GDB to start TUI mode.
        - Alternatively, you could install and run gdbgui, if you prefer a GUI to a CLI (It invokes "gdb" by default, so don't forget to point it to the right one on start-up (e.g. "gdbgui -g arm-none-eabi-gdb").

Helpful tools

- Freescale FreeMaster[37]
- Micrium uC/Probe[38]
- Segger Utilities[39] (if using J-Link/J-Link OB debug adapters)
    - Listed under “More information” > “Technology” at the bottom of the webpage.
- Third-party utilities[40] that integrate with Segger J-Link (if using J-Link/J-Link OB debug adapters)

References

- Product page[41]
- LPC84x product page[42]
- NXP: Get Started with the LPC845-BRK[43]
- LPC845-BRK User Manual[44] (requires free NXP account)
- LPC84x User Manual[45]
- LPC84x User Manual Errata Sheet[46]

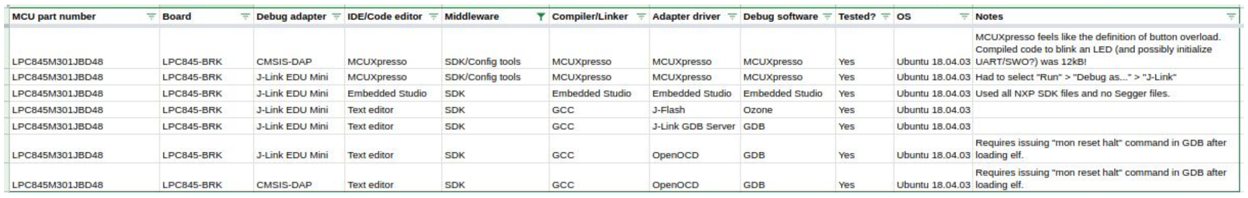
37 https://www.nxp.com/design/software/development-software/freemaster-run-time-debugging-tool:FREEMASTER
38 https://www.micrium.com/ucprobe/about/
39 https://www.segger.com/products/debug-probes/j-link/models/j-link-ob/
40 https://www.segger.com/products/debug-probes/j-link/tools/third-party-applications/
41 https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc800-cortex-m0-plus-/lpc845-breakout-board-for-lpc84x-family-mcus:LPC845-BRK
42 https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc800-cortex-m0-plus-/low-cost-microcontrollers-mcus-based-on-arm-cortex-m0-plus-cores:LPC84X
43 https://www.nxp.com/document/guide/-get-started-with-the-lpc845-brk-:GS-LPC845-BRK
44 https://www.nxp.com/webapp/Download?colCode=UM11181
45 https://www.nxp.com/webapp/Download?colCode=UM11029
46 https://www.nxp.com/docs/en/errata/ES_LPC84X.pdf

- MCUXpresso IDE Installation Guide[47]
- MCUXpresso SDK API Reference Manual[48]
- Keil Application Note 237: LPCXpresso LPC800 Cortex-M0+[49]

[47] https://www.nxp.com/docs/en/quick-reference-guide/MCUXpresso_IDE_Installation_Guide.pdf
[48] https://mcuxpresso.nxp.com/api_doc/dev/1156/
[49] http://www.keil.com/appnotes/docs/apnt_237.asp

Tested Toolchain Combinations

See Compatible Toolchains[50] or something more legible.



MCU part number	Board	Debug adapter	IDE/Code editor	Middleware	Compiler/Linker	Adapter driver	Debug software	Tested?	OS	Notes
LPC845M301JBD48	LPC845-BRK	CMSIS-DAP	MCUXpresso	SDK/Config tools	MCUXpresso	MCUXpresso	MCUXpresso	Yes	Ubuntu 18.04.03	MCUXpresso feels like the definition of button overload. Compiled code to blink an LED (and possibly initialize UART/SWO?) was 12kB!
LPC845M301JBD48	LPC845-BRK	J-Link EDU Mini	MCUXpresso	SDK/Config tools	MCUXpresso	MCUXpresso	MCUXpresso	Yes	Ubuntu 18.04.03	Had to select "Run" > "Debug as..." > "J-Link"
LPC845M301JBD48	LPC845-BRK	J-Link EDU Mini	Embedded Studio	SDK	Embedded Studio	Embedded Studio	Embedded Studio	Yes	Ubuntu 18.04.03	Used all NXP SDK files and no Segger files.
LPC845M301JBD48	LPC845-BRK	J-Link EDU Mini	Text editor	SDK	GCC	J-Flash	Ozone	Yes	Ubuntu 18.04.03	
LPC845M301JBD48	LPC845-BRK	J-Link EDU Mini	Text editor	SDK	GCC	J-Link GDB Server	GDB	Yes	Ubuntu 18.04.03	
LPC845M301JBD48	LPC845-BRK	J-Link EDU Mini	Text editor	SDK	GCC	OpenOCD	GDB	Yes	Ubuntu 18.04.03	Requires issuing "mon reset halt" command in GDB after loading elf.
LPC845M301JBD48	LPC845-BRK	CMSIS-DAP	Text editor	SDK	GCC	OpenOCD	GDB	Yes	Ubuntu 18.04.03	Requires issuing "mon reset halt" command in GDB after loading elf.

[50] https://github.com/nathancharlesjones/Embedded-for-Everyone/tree/master/Getting-Started-Guides