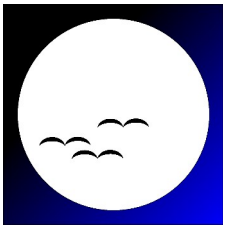


The QMdbSim Project

A Microchip Device Simulator Framework For QSpice

Document History

2025.02.22 – Initial documentation. Caveat emptor.

About The Project

The QMdbSim Project is a framework for using Microchip micro-controller devices in QSpice simulations. Under the covers, it uses Microchip's software simulator (MDB). Using the framework, developers can easily create and distribute "QMdbSim device driver components" for specific Microchip devices. Others can then use those driver components to load/execute device-specific binary files in QSpice simulations.

The project includes all schematics and code for an example PIC16F15213 device. This document provides some minimal information about using the example and how to create "drivers" for other Microchip devices.

Note: I assume that you have basic familiarity with QSpice as well as Microchip's MPLabX IDE and micro-controller devices.

Requirements

As mentioned, the project uses the Microchip software simulator (MDB). This tool is installed automatically with the MPLabX IDE. I presume that you have the IDE installed.

The project code executes MDB.bat and expects to find it here:

```
"C:\Program Files\Microchip\MPLABX\v6.20\mplab_platform\bin\mdb.bat"
```

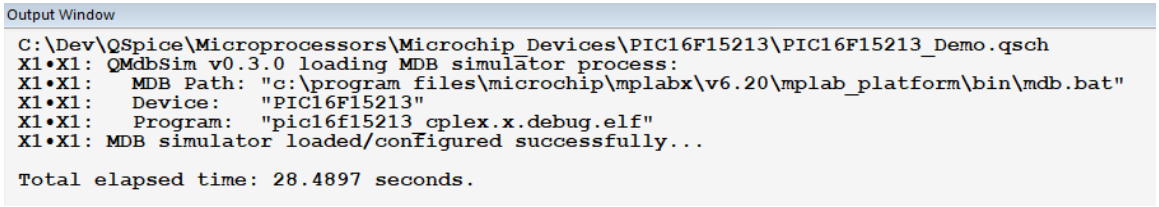
If you don't have MPLabX v6.20 installed or it is installed elsewhere, you'll need to change the path in `PIC16F15213.qsch` for your version/location.

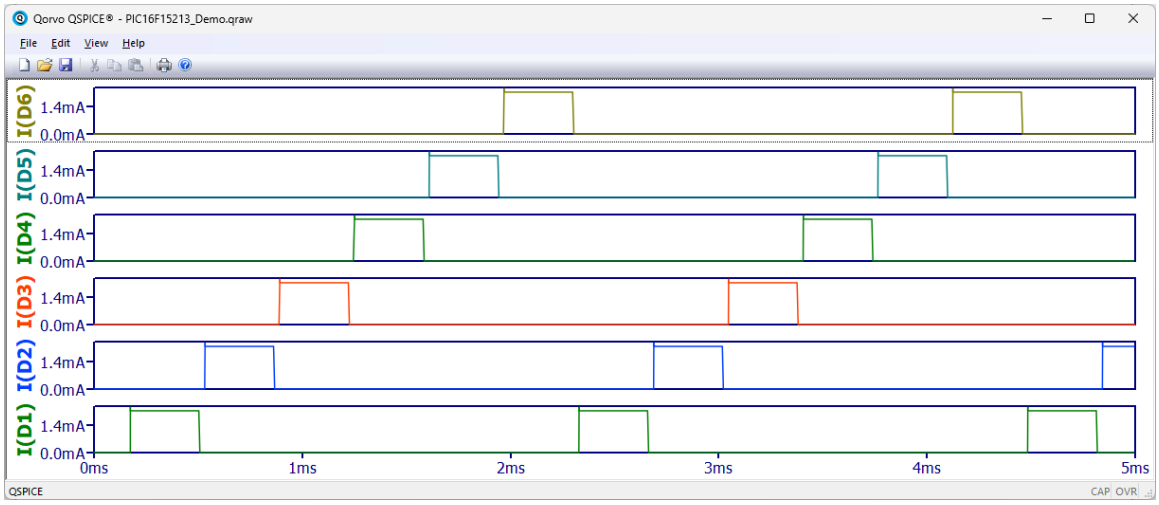
Using The Example PIC16F15213 Device

To test out the PIC16F15213 example, you need only copy the following project files from the repository:

File	Description
PIC16F15213_Demo.qsch	Top-level, user schematic containing the circuitry for an LED Charlie-Plexing project. It contains a hierarchical schematic block for PIC16F15213.qsch.
PIC16F15213.qsch	Device-specific schematic which contains the C-Block DLL component. It should not be changed (except for the MDB path mentioned above).
PIC16F15213.dll	The compiled device-specific C-Block code.
PIC16F15213_CPlex.X.debug.elf	The PIC binary "hex file" compiled with MPLabX for debugging. You can recompile the code for release (*.hex) if desired – either *.hex or *.elf files will work with the simulator.
PIC16F15213_CPlex.zip	Source code for PIC16F15213_CPlex.X.debug.elf. (Not required for simulation but available for your editing pleasure.)

Open `PIC16F15213_Demo.qsch` and run a simulation. Be patient – on my rather average laptop, the simulation takes about 30 seconds to complete. (MDB takes about 10 seconds just to initialize the simulator for the device and program.)





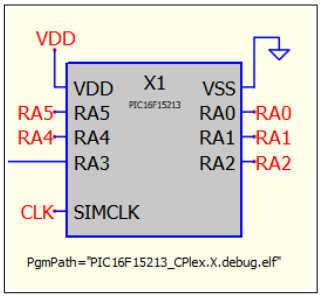
You can now change the PIC device code and recompile in MPLabX to produce a revised binary. Just change the X1 `PgmPath` attribute to match your binary file location/name if changed.

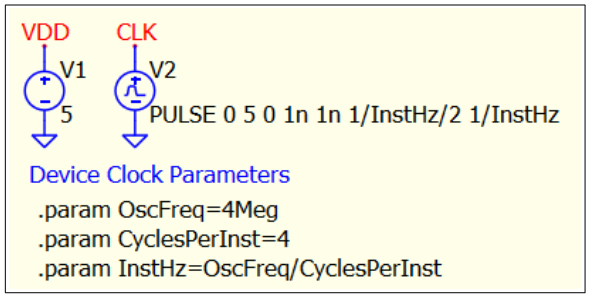
You can, of course, also make any changes you like to the top-level schematic. But you'll need to know a few things….

Internal Oscillators

The X1 ports match the actual PIC16F15213 pins with an added "SIMCLK" pin that doesn't exist on the physical device. We need SIMCLK to match the *instruction frequency* set in the device code "configuration bits."

On this device, one instruction takes four oscillator cycles. The example device code enables the 4MHz internal oscillator so we set `OscFreq` to 4Meg and let the `.param` directives calculate the 1MHz instruction frequency for us.





Note: To be clear, while the physical device has an internal oscillator that can be configured for various frequencies, the simulator has no way to map PIC internal oscillator time to QSpice simulation time. The QMdbSim code simply advances the PIC simulation by one instruction cycle on each falling edge of SIMCLK.

Weak Pull-Ups

The PIC16F15213 supports weak pull-ups on some pins. Control over pull-ups is configured in the binary code and can by changed dynamically in the code. Until someone convinces me that it's truly important, it's not implemented in the simulations. You can add pull-ups in your top-level schematic to match the (static) configuration in your device code for "better" electrical simulation. (See Electrical Fidelity below.)

Peripherals

Microchip devices have "integrated peripherals" such as ADC, DAC, SPI, I2C, USB, etc. The MDB simulator may or may not support a given peripheral for a given device. If MDB doesn't support it, well, this project doesn't support it. See the Resources section for a link to Microchip's list of peripherals supported by MDB.

According to that list, the MDB simulator supports the following peripherals for the PIC16F15213: CCP1/2, EEFlash, INT/IOC, FVR, OSC, PORTA, PPS, PWM3/4, Pull-Up, TMR0-2, UART1, WDT. Notably, ADC isn't in that list.

Electrical Fidelity

The simulation doesn't attempt to accurately replicate the electrical behavior of a physical device. `PIC16F15213.qsch` contains the electrical circuitry seen by QSpice. You can change that (for example, to add internal pull-up resistors) but keep in mind that the simulator is a "digital-behavior tool" and not really suitable for electrical analysis. That is, you shouldn't expect to use the QSpice simulation to accurately measure currents on the device pins.

Creating Other Microchip Devices

Creating other Microchip devices isn't hard using the QMdbSim framework. Well, it's as easy as I can make it….

You'll definitely need a modern C++ compiler. I've provided project files for Microsoft Visual Studio. If you use these, you'll be able to use the MSVS debugger to run simulations in a terminal window. (See Resources below.)

First, download the rest of the files from the project repository. Here are the steps to create a "PICxyz" device:

- Rename `PIC16F15213.qsch` to `PICxyz.qsch` and edit it. This schematic provides the electrical mapping between the component code and the top-level schematic (i.e., `PIC16F15213_Demo.qsch`).
  - For every tri-state GPIO pin, you'll need three QSpice component ports (in/out/ctrl) and a GPIO_Pin tri-state "connector."
  - For input-only device pins, see RA3 in the example.
  - Change the DLL name attribute to "PICxyz" and generate a template (`PICxyz.cpp`) from QSpice to get the correct "data[]" mappings to use below.
  - Change the DLL name attribute to PICxxx.
- Copy `PIC16F15213.cpp` to `PICxxx.cpp`.
  - Edit the evaluation function name to "picxxx".
  - Copy the "`data[]`" mappings from the temporary `PICxyz.cpp` above into the evaluation function.
  - Replace all "PIC16F15213" literals with "PICXYZ".
  - In the evaluation function initialization, edit/add/remove the pin/port/name mappings to match the new device. It looks like this:

```
// add all pin/port/name mappings to list
inst->mdb.addPinPortMap("RA0", &RA0_I, &RA0_O, &RA0_C);
inst->mdb.addPinPortMap("RA1", &RA1_I, &RA1_O, &RA1_C);
inst->mdb.addPinPortMap("RA2", &RA2_I, &RA2_O, &RA2_C);
inst->mdb.addPinPortMap("RA3", &RA3_I);
inst->mdb.addPinPortMap("RA4", &RA4_I, &RA4_O, &RA4_C);
inst->mdb.addPinPortMap("RA5", &RA5_I, &RA5_O, &RA5_C);
```

- Create a top-level demo schematic similar to `PIC16F15213_Demo.qsch`. (Note: We'll eventually supply the "devices" as symbols but, for now, we're using hierarchical schematics.)
  - Open `PICxxx.qsch` and right-click in an open area of the schematic.
  - Select "Open Parent Schematic" to generate the demo schematic.
  - Add the SIMCLK bits from `PIC16F15213_Demo.qsch` (V2/CLK signal and related `.param` directives).
  - Add the remaining electrical circuitry for your demo.
- Create device-specific demonstration code in MPLabX.

Finally, package up the various bits and – most importantly – *share!*

Do let me know if I missed something or if you have questions.

Next Steps

I think that the biggest issue is simulation speed. At this point, the project loads MDB as a console application. It sends/receives commands via `stdin`/`stdout`/`stderr`. There's much formatting and parsing which is inherently slow.

For faster simulation, we need to interact directly with the MDB API. The API is Java stuff. Although I can do Java, it's not my strongest suit.

So, if you're a Java dude/dudette and would like to contribute, please let me know. You'll be my hero.

Resources

My GitHub QSpice Repository

https://github.com/robdunn4/QSpice

In addition to this project, you'll find information about C-Block component concepts ("C-Block Basics"), various C-Block component examples, tips on using VSCode and Visual Studio IDEs for C-Block development and debugging, and links to other useful QSpice-related sites/repositories.

Microchip

MPLabX IDE (free)

PIC16F15213/14/23/24/43/44 Low Pin Count Micro-Controllers (datasheet)

Microchip Debugger (MDB) User's Guide (one)

Microchip Debugger (MDB)User's Guide (another one)

Simulator Peripheral Support By Device

Microsoft

Visual Studio Downloads (2022 Community Edition is free)

Conclusion

I hope that you find this project to be useful or – at least – interesting. Please let me know if you find problems or suggestions for improving the code or this documentation.

You can find me on the QSpice Forums as @RDunn.

--robert