Playing Around with Project Template

The project template repo has an example of adding a peripheral to a rocket-chip instance. Reading through the example code might give you some idea of what's going on, but you'll learn a lot by modifying it.

Project-template includes an example peripheral that adds a programmable pulse-width modulation output. Wikipedia has a pretty good description of PWM. It is often used for motor control. For the purposes of the lab, what you need to know is that PWM is a single-bit output that has a period and duty cycle, illustrated below.

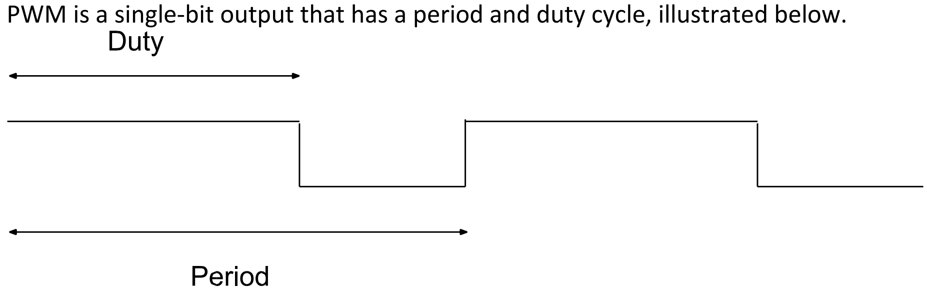

Figure 1: PWM Waveform

Build the example

Follow the instructions in the README. The instructions here are incomplete and meant to summarize the README.

RISC-V Tools

You'll need to run git submodule update --init --recursive. This will download riscv-tools, which includes a C compiler for the RISC-V architecture. You'll need to build these tools to run test programs on rocket. Building these tools can take a while. There are also known issues on OS X if you are using a filesystem that is case sensitive (Windows is problematic too). Workarounds exist, talk to Sijun (sijun@berkeley.edu) if you run into problems.

Build the simulator

Build the debug target.

make CONFIG=PWMConfig debug

Run tests

Build the tests in the tests/ directory with make. Run tests/pwm.riscv Use the -vout.vcd option to generate a VCD file to out.vcd.

./simulator-example-PWMConfig-debug -vout.vcd ../tests/pwm.riscv

View waves

Open out.vcd with your preferred waveform viewer and find the pwmout signal.

Modifying PWM

You will add a PWM "demodulator" to the PWM example. The idea of the demodulator is that it will be able to tell you the "duty cycle" of an input. We'll build a very simple demodulator, so it will not figure out what the input period is. Instead, the period will be a value you can set from the processor. A block diagram is shown below:

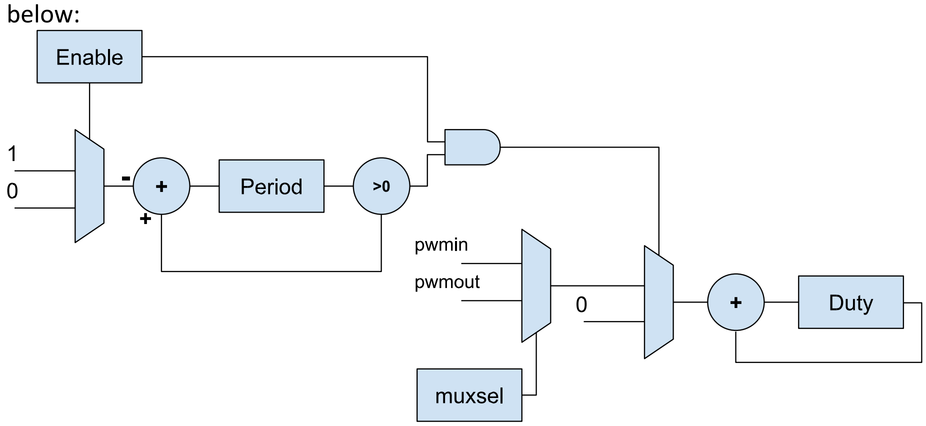

Figure 2: PWM demodulator block diagram

The demodulator will add a top level input, called pwmin. A register, called muxsel, controls whether the demodulator uses pwmin or instead feeds back the output of the unmodified PWM block (called pwmout).

A register called period will control how long the selected input is integrated. The result of the integration will be stored in a register called duty. A register called enable will control when the integration is performed. The value of period will decrement as the integration is performed until its value reaches zero. Software will poll on period until its value is zero, after which it knows the integration is complete and the value of duty is valid.

C programs

Add a test based on pwm.c. Call it demod.c. It should:

1) Enable PWM output using the code from the original pwm.c
2) Set muxsel to choose the output of the PWM block as the input of the demodulator
3) Set a demodulation period and enable the demodulator
4) Poll until the integration is complete and check that the integration result is consistent with the duty cycle of the PWM output

Look at waves

Look at the waves and prepare some printouts. Show the following:

For the un-modified project-template, run pwm.riscv and show: 1) pwmin and pwmout 2) The memory mapped registers getting updated by the program

For your modified project-template, run demod.riscv and show: 1) pwmout and pwmin 2) The memory mapped registers getting updated by the program

Deliverables

1) Any Chisel files you modify. Most of the changes you'll have to make will probably be in PWM.scala. You will probably also have to modify TestHarness.scala to set the value of pwmin.
2) Your C test, called demod.c.
3) Screenshots of waves, detailed in the section above.

Extras

If you finish this lab and have extra time, feel free to make your demodulator more sophisticated. Some ideas:

- PWM → PAM converter and low pass filter
- Add queue to store samples so processor doesn't drop inputs
- Modify the test harness so you feed a creative input into pwmin