Lab 1.1: PWM Hardware Design

In lab 1.0, we provided you with the hardware design and you did not have to bother with it. It is time to open the black box! You are now going to design an Avalon-MM slave interface that can be accessed by the Nios II processor to configure and generate the PWM signal. **We provide you with a template you need to complete.**

Some key concepts

- **I/O:** In a computer system, one can observe two main techniques when it comes to handling I/O between a processor and its peripherals:
  - *Memory-mapped I/O* where the same address bus is used by the processor to access memories and peripherals (e.g. Avalon); and
  - *Port-mapped I/O* where the processor provides the programmer with specific mechanisms, generally instructions, to access a peripheral (e.g. Intel).
- **Master/Slave:** In the bus terminology, we refer to a component that can initiate a transaction (read or write transfer) on the bus as a *master*. A *slave* component can only respond to a transaction initiated by a master. For instance, let's consider the classical case of a processor and a memory controller. When the processor accesses memory, it uses its *master* interface on the bus to access the *slave* interface of the memory controller.

Task 1: designing the PWM on PAPER

Apart from designing its interface on the bus, let's consider how to design the PWM generator itself. To help you figure this out, Figure 1 depicts a logic circuit diagram that basically divides the frequency of an input clock signal signal named *clk*.

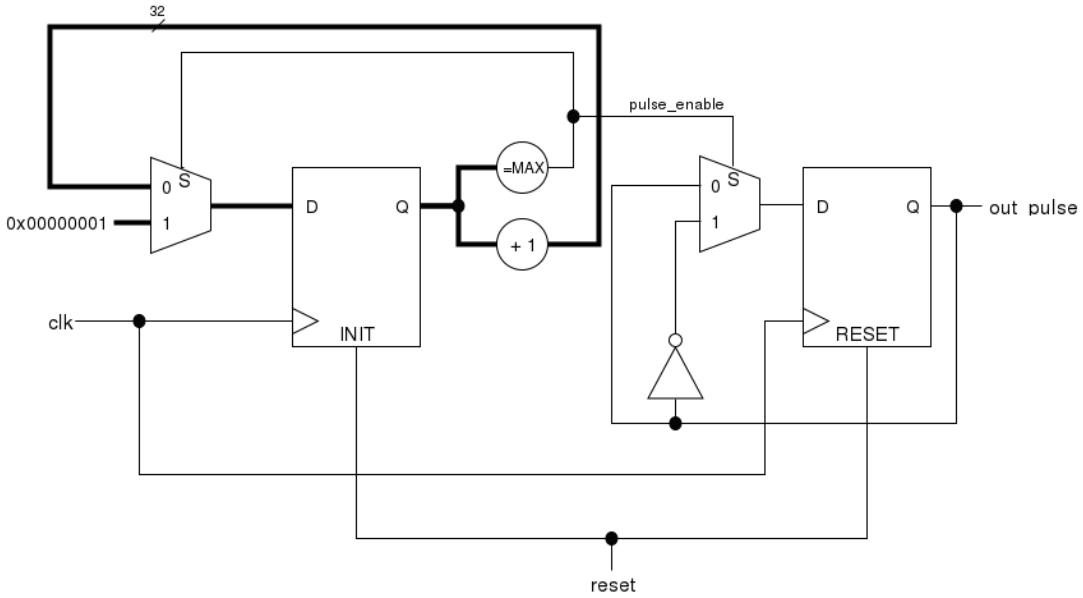

FIGURE 1: LOGIC DIAGRAM OF A STATIC FREQUENCY DIVIDING CIRCUIT. *MAX* IS THE DIVIDER.

Your task is to tweak this circuit to transform it into a PWM generator. To this end, you might consider that you have two additional 32-bit signals: *period* (to replace the *MAX* constant) and *duty* (to represent the duty cycle). As in lab 1.0, *duty* is supposed to be between 0 and *period*. You must also use a *start* signal to start and to stop the PWM generation.

Task 2: Write the code for your design

Complete the file named *<project_dir>/hw/hdl/pwm/hdl/pwm.vhd* with the code of what you designed in the previous task.

Task 3: Avalon-MM slave interface

It is now time to add support for the interface with the bus. Your interface should meet two criteria:

1. It should be compatible with the register map used in lab 1.0 so that your software still works. The register map is also shown in the *pwm.vhd* file.
2. It should meet the timing requirements of the Avalon bus (see slides).

To help you with this part, Figure 2 shows an example of the write circuitry.

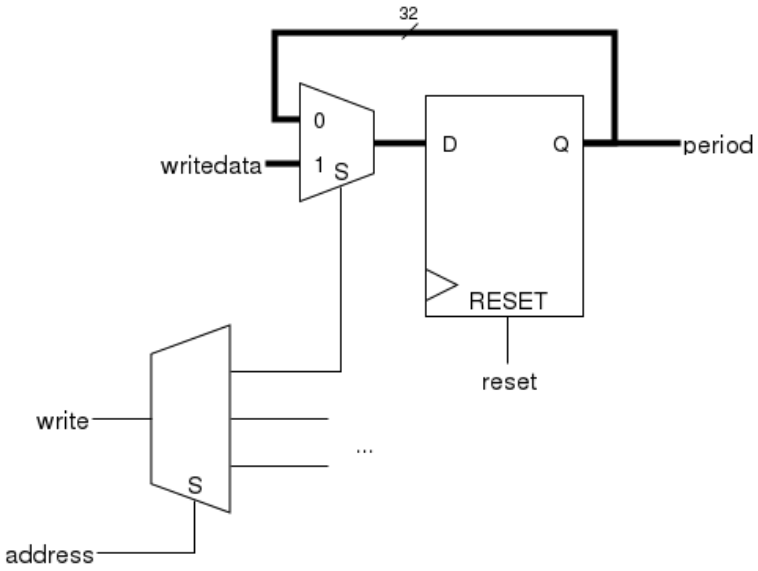

FIGURE 2: WRITE CIRCUITRY OF THE AVALON-MM SLAVE INTERFACE. ONLY THE *PERIOD* REGISTER IS SHOWN.

As you might have guessed, the signals *period, duty* and *start* we asked for in Task 1 are the registers written to by the processor through the bus.

Once the code is written, you can compile the *Quartus* project *<project_dir>/hw/quartus/lab1.qpf* and test it with your code of lab 1.0. You can also test your design in ModelSim with the provided testbench in *<project_dir>/hw/hdl/pwm/tb/tb_pwm.vhd.*