**Lab 3**

**Programmable Clock Divider**

**Prepared by:**

**Steven Blair and Daniel Olsen**

**Prepared for:**

**Dr. Wang**

**ECE 428/593B-001**

**Programmable ASIC Design**

**March 14, 2015**

# Specification

In this lab, students designed and constructed a programmable clock divider. The divider accepts a 100 MHz input clock and provides either a 4 MHz or a 5 MHz output clock based on the position of a toggle switch used to select a frequency and a push button used to latch in a frequency.

# Background

A digital clock manager (DCM) is a circuit that manipulates clock signals; in practice, it is often used to multiply/divide a clock signal, to recondition the duty cycle of a clock signal, to change the phase of a clock signal, or to eliminate clock skew across a clock domain.

In this lab, a specific implementation of a digital clock manager DCM\_CLKGEN that is provided by Xilinx is used. This digital clock manger has four inputs: CLKIN, the input clock; PROGCLK, the programming clock; PROGEN, the programming enable signal; and PROGDATA, the programming data. This digital clock manager also has three outputs: CLOCKFX, the output clock; PROGDONE, the signal that represents when programming is done; and LOCKED, the signal that represents when clock adjustment is done.

The input clock frequency and the output clock frequency for the digital clock manager are related by the equation

|  |  |  |
| --- | --- | --- |
|  |  | Equation |

where the values of the multiplier and the divider are programmable. To program the circuit, a clock signal must be provided on the PROGCLK pin and the following signals must be provided on the PROGDATA and PROGEN pins:

1. A 2-bit LoadD command ‘10’ followed by an 8-bit value representing with the least significant bit sent first must be shifted into the PROGDATA pin; during these ten cycles, the PROGEN pin must be asserted.
2. A 2-bit LoadM command ‘11’ followed by an 8-bit value representing with the least significant bit sent first must be shifted into the PROGDATA pin; during these ten cycles, the PROGEN pin must be asserted.
3. A 1-bit command ‘0’ must be shifted into the PROGDATA pin; during this one cycle, the PROGEN pin must be asserted.

Step 1 and Step 2 must be separated by at least two cycles; Step 2 and Step 3 must be separated by at least one cycle. Following this procedure, the digital clock manager is internally programmed with the specified values; at the conclusion of programming, the PROGDONE pin is asserted. It takes a finite amount of time for the digital clock manager to adjust the output clock according to specifications; at the conclusion of adjustment, the LOCKED pin is asserted.

# Design

## Design at the High Level

The digital clock manager provides a convenient mechanism to divide the 100 MHz clock available on the Atlys Development Board to either a 4 MHz or a 5 MHz clock through its programmable interface; if this interface is used, the design problem for the programmable frequency divider is reduced to the problem of designing a circuit that programs the digital clock manager with the appropriate values of the multiplier and the divider.

The programming process requires a strict procedure that defines the required inputs and the expected outputs to be followed; the required inputs are in the form of a flag that must be set on the PROGEN pin and a variety of commands and arguments that must be set on the PROGDATA pin. The LoadD command is 2 bits; the LoadD argument is 8 bits; the LoadM command is 2 bits; the LoadM argument is 8 bits; and the Go command is 1 bit. Since the PROGEN pin must be set for a sequence of a command and an argument, it is convenient to combine the command and argument where appropriate. The LoadD command and argument can be combined into a 10 bit vector; the LoadM command and argument can be combined into a 10 bit vector; and the Go command remains a 1 bit vector.

The programming process can be modeled as a finite state machine with each state in the machine representing either an entire step or a partial step in the programming process. A simple breakdown of the process reveals that states are needed for (1) the LoadD command and argument, (2) the delay between the LoadD command and the LoadM command, (3) the LoadM command and argument, (4) the delay between the LoadM command and the Go command, (5) the Go command, (6) the delay needed for programming, and (7) the delay needed while idle.

Each command and argument can be loaded into a shift register prior to programming and shifted out of the shift register during programming; each shift of the shift register can be counted by a counter to ensure that the variable length commands and arguments are handled correctly. Since the longest string of bits that needs to be sent at a single time is 10 bits, the shift register that is used can be limited to a width of 10 bits and the counter can be limited to a width of 4 bits.

The primary inputs to the circuit, the outputs of the digital clock manager, and the counter will be used to determine transitions in the finite state machine. The finite state machine will determine what value needs to be loaded into the shift register, when the value needs to be loaded into the shift register, when the shift register needs to be enabled, when the counter needs to be reset, and when the programming interface on the digital clock manager needs to be enabled. This fully describes the values needed on PROGEN and PROGDATA and satisfies the design problem.

Table : Enumeration of States in Finite State Machine

|  |  |
| --- | --- |
| **State** | **Description** |
| LoadD1 Preparation | The command for LoadD and the divider argument for 4 MHz operation are loaded into the shift register. The circuit transitions into the LoadD state. |
| LoadD2 Preparation | The command for LoadD and the divider argument for 5 MHz operation are loaded into the shift register. The circuit transitions into the LoadD state. |
| LoadD | The command for LoadD and the divider argument for either 4 MHz or 5 MHz operation are shifted out of the shift register into the programming data pin of the digital clock manager; the programming enable pin is also asserted. After everything is shifted out in ten cycles, the circuit transitions into the Delay state. |
| Delay | A single clock cycle is wasted. The cycle spent in the Delay state and the cycle spent in the LoadM Preparation state satisfy the two cycles of delay required between the LoadD operation and the LoadM operation. The circuit transitions into the LoadM Preparation state. |
| LoadM Preparation | The command for LoadM and the constant multiplier argument are loaded into the shift register. The circuit transitions into the LoadM state. |
| LoadM | The command for LoadM and the constant multiplier argument are shifted out of the shift register into the programming data pin of the digital clock manager; the programming enable pin is also asserted. After everything is shifted out in ten cycles, the circuit transitions into the Go Preparation state. |
| Go Preparation | The command for Go is loaded into the shift register. The cycle spent in the Go Preparation state satisfies the one cycle of delay required between the LoadM operation and the Go operation. The circuit transitions into the Go state. |
| Go | The command for Go is shifted out of the shift register into the programming data pin of the digital clock manager; the programming enable pin is also asserted. After everything is shifted out in one cycle, the circuit transitions into the Wait for Programming state. |
| Wait for Programming | The circuit waits for the internal programming of the digital clock manager to complete; this serves to prevent further programming to occur until the digital clock manager is ready. Once the digital clock manager indicates the programming is done, the circuit transitions into the Idle state. |
| Idle | The circuit waits for a programming operation to be requested. Based on the value of the frequency select switch, the circuit transitions into the LoadD1 Preparation state or the LoadD2 Preparation state when the programming request button is pressed. |

## Defining the Programming Data

In the digital clock manager, the multiplier must have a minimum value of 2 and the divider must have a minimum value of 1. Since the circuit is primarily concerned with the dividing process, the multiplier is set at its minimum value of 2 and the divider is used to divide the resulting 200 MHz clock to the required value. To yield 4 MHz operation, the 200 MHz clock must be divided by a divider of 50; to yield 5 MHz operation, the 200 MHz clock must be divided by a divider of 40.

The arguments shifted into the digital clock manager must take the value given by the value required for the argument minus one. The multiplier of 2 corresponds to an argument of 1 shifted in; the divider of 50 corresponds to an argument of 49 shifted in; and the divider of 40 corresponds to an argument of 39 shifted in. It must be noted that the commands must be shifted into the digital clock manager in order, while the arguments must be shifted with the least significant bit first. It should also be noted that the digital clock manager can be initialized with values for the multiplier and the divider to ensure that the circuit starts in a valid state.

## Defining the Programming Data and Programming Enable Process

Given the programming process for the digital clock manager, it is possible to provide an enumeration of states and a description of state transitions for a finite state machine modeling the programming process. Since the values to be shifted into the programming data pin of the digital clock manager must be loaded into a shift register first, additional states have been added to the finite state machine to allow preparation of the shift register. Occasionally, these extra states may be useful as they may absorb the delay needed between sending commands to the digital clock manager; most of the time, though, these extra states add overhead to the design.

An enumeration of all the states is provided in Table 1; a state transition diagram relating the states with the state transitions is provided at the conclusion of this report.

# Implementation

## Description of Modules and Sub-Modules

The programmable clock divider consists of four modules: a finite state machine module, a shift register module, a counter module, and a digital clock management module.

The finite state machine module was written in VHDL using a behavioral style. It accepts as input a clock signal, a reset signal, the program request signal, the frequency select signal, the counter count signal, and the digital clock manager programming done signal; it provides as output the shift register shift enable signal, the shift register load enable signal, the shift register load value signal, the counter reset signal, and the digital clock manager programming enable signal. Each state in the finite state machine represents a step or a partial step in the programming process.

The shift register module was written in VHDL using a behavioral style. It accepts as input a clock signal, a shift enable signal, a load enable signal, and a load value signal; it provides as output a shift out signal. The shift register is 10 bits wide and is primarily responsible for holding the bits used to program the digital clock manager.

The counter module was written in VHDL using a behavioral style. It accepts as input a clock signal, an enable signal, and a reset signal; it provides as output a count signal. The counter is 4 bits wide and is primarily responsible for counting the number of bits used to program the digital clock manager.

The digital clock management module was implemented as a Xilinx DCM\_CLKGEN primitive. It accepts as input an input clock signal, a programming data signal, a programming enable signal, a programming clock signal, and a reset signal; it provides as output an output clock signal, a programming done signal, and a locked signal. It is initialized to provide 5 MHz operation at boot time, but it can be reprogrammed at run time to provide either 4 MHz or 5 MHz operation.

The four modules have been implemented together in a top-level module implemented in VHDL using a structural style. The shift enable, load enable, and load value of the finite state machine are connected to the shift register; the count reset signal of the finite state machine is connected to the counter; and the programming enable signal of the finite state machine is connected to the digital clock manager. The shift out signal of the shift register is connected to the programming data pin of the digital clock manager. The count signal of the counter and the programming done signal of the digital clock manager are both connected to the finite state machine. The input clock is used to synchronize all modules and is used as the input of the digital clock manager; the input reset is used to reset all modules.

## Mapping of Signals to Development Board

The inputs to the programmable clock divider consist of the input clock signal, the reset signal, the program request signal, and the frequency select signal. The input clock is mapped to the 100 MHz CMOS oscillator available on pin L15. The reset signal and the program request signal are mapped to the push button on pins P3 and N4, respectively. The frequency select signal is mapped to the slide switch on pin A10.

The outputs of the programmable clock divider consist of the output clock signal, the programming done signal, and the locked signal. The output clock signal is mapped to the PMOD pin available on pin T3. The programming done signal and the locked signal are mapped to the LEDs on pins U18 and M14, respectively.

# Simulation

To ensure proper operation of the design, a testbench was written in VHDL that tests each of the major transitions through the finite state machine. The testbench consists of two tests: (1) configuration for 4 MHz operation and (2) configuration for 5 MHz operation. During all tests, a 100 MHz clock must be provided to the testbench to replicate the 100 MHz oscillator provided by the development board.

Proper operation of the design can be verified by checking the waveforms generated by the testbench against the specifications provided in the Specification section. The waveforms corresponding to a post-translation simulation of the testbench are included at the conclusion of this report; careful analysis will reveal that these waveforms do indeed meet specifications.

Immediately following simulation, a full test of the design was run on the Atlys Development Board. This design was verified by the TA to perform according to specification.