بسم الله الرحمن الرحيم

washing machine controller

Created by :

أحمد مازن الحلو

إبراهيم ربحي الخولي

مايو

2023

1. Introduction :

The objective of this project is to develop a versatile washing machine controller capable of controlling various wash cycles, such as washing, rinsing, spinning, and idle, while supporting multiple pre-programmed wash programs tailored to specific fabric types or clothing requirements. The controller is designed to provide a seamless user experience by incorporating intuitive control panel buttons and a seven-segment display for visual feedback.

2. Entities :

1. washing_machine :

"washing_machine," is responsible for controlling the operation of a washing machine. It takes inputs such as "clk" (clock signal), "reset" (reset signal), "start" (start signal), and "userProg" (user program selection signal). The output "state" represents the current state of the washing machine.

1.1- Architecture and Components :

The architecture "washing_machine_arch" defines the behavior of washing_machine It includes the following components:

1.2-Constants :

WASH: Represents the washing state.
SPIN: Represents the spinning state.
RINSE: Represents the rinsing state.
IDLE: Represents the idle state.
COTTON: Represents the user program for cotton.
RAPID: Represents the user program for rapid.
SLOW: Represents the user program for slow.
DRAIN: Represents the user program for draining.
WHITE: Represents the user program for white clothes.
COTTON_CYCLES_COUNT: Array of integers representing the number of cycles for each stage in the cotton program.

RAPID_CYCLES_COUNT: Array of integers representing the number of cycles for each stage in the rapid program.
SLOW_CYCLES_COUNT: Array of integers representing the number of cycles for each stage in the slow program.
DRAIN_CYCLES_COUNT: Array of integers representing the number of cycles for each stage in the drain program.
WHITE_CYCLES_COUNT: Array of integers representing the number of cycles for each stage in the white program.

2.3-Signals :

wash_counter: Integer signal representing the current cycle count for the washing stage.
spin_counter: Integer signal representing the current cycle count for the spinning stage.
rinse_counter: Integer signal representing the current cycle count for the rinsing stage.
curr_program: 5-bit vector signal representing the current user program selected.
curr_state: 2-bit vector signal representing the current state of the washing machine.

2. Test Bench for washing_machine

The test bench for the washing machine controller is implemented in the washing_machine_tb entity. It includes the following components:

2.1-Component Declaration :

The test bench instantiates the washing_machine component, which represents the washing machine controller being tested. It has input and output ports to interface with the controller.
Testbench Signals

Several signals are declared within the test bench architecture to control the inputs and monitor the outputs of the controller. These signals include:

tb_clk: Represents the clock signal for the test bench.
tb_reset: Controls the reset signal for the controller.
tb_start: Controls the start signal for the controller.
tb_userProg: Represents the user program input to the controller.
tb_state: Monitors the state output of the controller.

2.2-Clock Process :

The clock process generates the clock signal for the test bench. It oscillates between '0' and '1' with a specified clock period.

2.3-Stimulus Process :

The stimulus process applies various test scenarios to the controller and monitors its state output for verification. Each scenario involves setting the tb_userProg and tb_start signals to specific values and asserting the expected states of the controller.

2.4-Test Results

The test bench was executed, simulating the behavior of the washing machine controller under different scenarios. The results of the simulation are as follows:

★ Test Scenario: COTTON Program

Expected behavior:
State 1: WASH
State 2: WASH
State 3: SPIN
State 4: RINSE (repeated three times)
State 7: IDLE
Observed behavior: [Include any observed deviations from the expected behavior]

★ Test Scenario: RAPID Program

Expected behavior:
State 1: WASH
State 2: WASH
State 3: SPIN (repeated three times)
State 6: RINSE
State 8: IDLE
Observed behavior: [Include any observed deviations from the expected behavior]

- ★ Test Scenario: SLOW Program

  Expected behavior:
  State 1: WASH (repeated two times)
  State 3: SPIN
  State 4: RINSE
  State 5: IDLE
  Observed behavior: [Include any observed deviations from the expected behavior]

- ★ Test Scenario: DRAIN Program

  Expected behavior:
  State 1: WASH
  State 2: SPIN (repeated two times)
  State 4: RINSE
  State 5: IDLE
  Observed behavior: [Include any observed deviations from the expected behavior]

- ★ Test Scenario: WHITE Program

  Expected behavior:
  State 1: WASH
  State 2: SPIN (repeated three times)
  State 4: RINSE
  State 5: IDLE
  Observed behavior: [Include any observed deviations from the expected behavior]

3. seven_segment :

The "seven_segment" entity controls the seven-segment display and takes inputs such as "clk" (clock signal), "reset" (reset signal), "start" (start signal), "userProg" (user program selection signal), and "state" (current state signal). The entity provides outputs for "sevSeg_data" (seven-segment display data) and "sevSeg_driver" (seven-segment display driver).

3.1-Architecture and Components :

The architecture "seven_segment_arch" defines the behavior of the seven-segment display controller. It includes the following components:

3.2-Constants :

Characters: Constants representing the binary encoding of different characters to be displayed on the seven-segment display.
States: Constants representing the different states of the system.
Programs: Constants representing different user programs.

3.4-Signals :

sevSegIndex: An integer signal that keeps track of the current character index to be displayed on the seven-segment display.

4. Test Bench for seven_segment :

The test bench for the seven-segment display controller, named seven_segment_tb, has been implemented. It includes the necessary signals and processes to test the functionality of the controller.

4.1-Components and Signals :

The test bench instantiates the seven_segment component, which represents the seven-segment display controller being tested. The test bench includes the following signals:

clk: Represents the clock signal for the test bench.
reset: Controls the reset signal for the controller.
start: Controls the start signal for the controller.
userProg: Represents the user program input to the controller.
state: Monitors the state output of the controller.
sevSeg_data: Represents the data output to the seven-segment display.
sevSeg_driver: Represents the driver output for the seven-segment display.

4.2-Clock Process :

The clock process generates the clock signal for the test bench. It oscillates between '0' and '1' with a specified clock period of 100000 ns.

4.3-Stimulus Process :

The stimulus process applies various test scenarios to the controller and monitors its outputs for verification. It includes test cases for different states of the controller and checks the expected values of sevSeg_data and sevSeg_driver at each state.

4.4-Test Results :

During the simulation of the seven-segment display controller using the test bench, the following results were observed:

★ State 00:
Expected sevSeg_data value: "1010101"
Expected sevSeg_driver value: "100000"
Observed sevSeg_data value: [Include the observed value]
Observed sevSeg_driver value: [Include the observed value]

★ State 11:
Expected sevSeg_data value: "1111001"
Expected sevSeg_driver value: "100000"
Observed sevSeg_data value: [Include the observed value]
Observed sevSeg_driver value: [Include the observed value]

★ State 10:
Expected sevSeg_data value: "0011001"
Expected sevSeg_driver value: "100000"
Observed sevSeg_data value: [Include the observed value]
Observed sevSeg_driver value: [Include the observed value]

★ State 01:
Expected sevSeg_data value: "0100100"
Expected sevSeg_driver value: "100000"
Observed sevSeg_data value: [Include the observed value]
Observed sevSeg_driver value: [Include the observed value]

5. clock_devider :

The clock divider component plays a crucial role in digital design by generating a slower clock signal from a higher-frequency clock signal. This slower clock signal is often required to synchronize operations and ensure proper timing within the design. The purpose of this report is to provide an overview of the clock_devider entity, its architecture, and functionality.

The clock_devider entity defines the interface and ports of the clock divider component. It includes the following ports:

clk: An input port representing the original clock signal.
reset: An input port for the reset signal.
desired_freq: An input port specifying the desired frequency for the slower clock signal.
out_clk: An output port providing the slower clock signal.

5.1- Architecture and Functionality

The clock_devider_arc architecture describes the behavior and implementation details of the clock divider component. It utilizes internal signals and processes to generate the slower clock signal.

The architecture includes the following internal signals:

counter: An integer signal used to keep track of the current count.
curr_out: A std_logic signal representing the current value of the output clock signal.

Within the architecture, there is a process sensitive to changes in the clk and reset signals. The process implements the functionality of the clock divider by incrementing the counter on each rising edge of the clk signal. It also checks if the counter has reached or exceeded a threshold value calculated based on the desired frequency.

If the counter exceeds the threshold, the curr_out signal is toggled using the not operator, and the counter is reset to zero. This toggling behavior generates the slower clock signal with a frequency proportional to the desired frequency specified by the desired_freq input port.

Finally, the curr_out signal is assigned to the out_clk output port, allowing the slower clock signal to be used in other components of the design.

6. Main Component: Controller :

The main entity, named "main," serves as the top-level entity that integrates various components to control the washing machine's operation. The main entity receives input signals such as "userProg" (user program selection), "clk" (clock signal), "reset" (reset signal), and "start" (start signal). It provides output signals including "state" (current state of the washing machine), "sevSeg_data" (data for a seven-segment display), and "sevSeg_driver" (driver for enabling specific characters on the display).

5.1- Architecture and Components :

The architecture "main_arch" defines the behavior of the main entity and incorporates multiple components to control the washing machine. The key components utilized in the architecture are:

★ **washing_machine:**
The washing_machine component is responsible for managing the core functionality of the washing machine. It receives the "clk" (clock signal), "reset" (reset signal), "start" (start signal), and "userProg" (user program selection) inputs. The component outputs the "state" signal, representing the current state of the washing machine.

★ **seven_segment:**
The seven_segment component handles the display functionality of the washing machine controller. It receives the "clk" (clock signal), "reset" (reset signal), "start" (start signal), "userProg" (user program selection), and "state" (current state of the washing machine) inputs. The component generates the appropriate "sevSeg_data" signal, which contains the data to be displayed on a seven-segment display. It also produces the "sevSeg_driver" signal, which determines which display segments are enabled at a given clock pulse.

★ **clock_devider:**
The clock_devider component is utilized to divide the input clock signal into slower clock signals for specific components. It takes the "clk" (clock signal), "reset" (reset signal), and "desired_freq" (desired frequency) inputs. The component generates the "out_clk" signal, which is a slower clock signal used by the associated component.

6.2-Signals :

In the main_arch architecture, the following signals are employed:

sev_seg_clock: A signal representing the slower clock signal generated by the clock_devider component for the seven_segment component.
washing_machine_clock: A signal representing the slower clock signal generated by the clock_devider component for the washing_machine component.
state_signal: A signal representing the current state of the washing machine, obtained from the washing_machine component.
sevSeg_data_signal: A signal containing the data to be displayed on a seven-segment display, obtained from the seven_segment component.
sevSeg_driver_signal: A signal determining which segments of the seven-segment display are enabled at a particular clock pulse, obtained from the seven_segment component.

References

project file : Washing Machine Controller.pdf
Code source : [Ahmed-elhelou/Washing-Machine (github.com)](https://github.com/Ahmed-elhelou/Washing-Machine)

Conclusion

The main entity serves as the central component in the design and implementation of the washing machine controller. It integrates the washing_machine and seven_segment components to control the washing machine's operation and display the relevant information. The clock_devider component is used to generate slower clock signals for synchronization purposes. By leveraging these components and their interactions, the main entity provides a functional and efficient solution for controlling a washing machine's cycles and displaying the relevant information on a seven-segment display. The washing machine controller can be further enhanced and expanded to incorporate additional features and functionalities to cater to different washing machine models and user preferences.