FH JOANNEUM
System Test Engineering

Digital Electronics – Laboratory 4

PWM circuit with automatic test bench and
a vending machine logic coded as a state machine

Patrick Lampl

12.11.2021

1 CONFIGURABLE PWM CIRCUIT

Work in teams of two, you can split coding efforts within your team or do co-programming to avoid programming mistakes (four eyes see more than two).

1.1 Specification of PWM circuit

Create a pulse width modulation generator *PWMgen* with the following features:

- 16-bit PWM resolution
- Configurable PWM frequency
- Two operation modes:
    - Continuous PWM generation
    - n-cycles mode
        - Configurable number of PWM cycles
        - Trigger Input signal to start generation of n-cycles

Consider the following design constraints:

- When the duty cycle is at its minimum value (0%) the PWM output should always stay low.
- When the duty cycle is at its maximum value (100%) the Output should be always high.
- When in n-cycles mode, the PWM output should stay low after all cycles were generated.
- The inputs and outputs should be according to the table below.

direction, bit-width and name of signal	Functional description
input [15:0] i_duty	PWM duty cycle control register
input [15:0] i_limit	PWM period control register
input [7:0] i_n	number of cycles to be generated within n-cycle mode
input i_trig	trigger the generation of n-PWM-cycles on rising edge of i_trig
input i_mode	Mode selection bit: 0 : selects continuous PWM, 1 : selects n-PWM-cycles mode
input i_clk	100MHz clock
input i_rst,	Active high synchronous reset: Resets all internal memories to zero
output logic o_pwm	PWM output signal

1.2 Initial manual tests

Create a simple, manually checked simulation which tests the following scenarios:

- Test only a single frequency for now: 0x7FFF
- Test 5 different Duty cycles: 0% 25% 50% 75%and 100%
- Test both modes, continuous and n-cycles mode.
    - Test the n-cycles mode by generating 2, 3 and 4 pulses

This should give you a total of 4*5 = 20 different test cases.

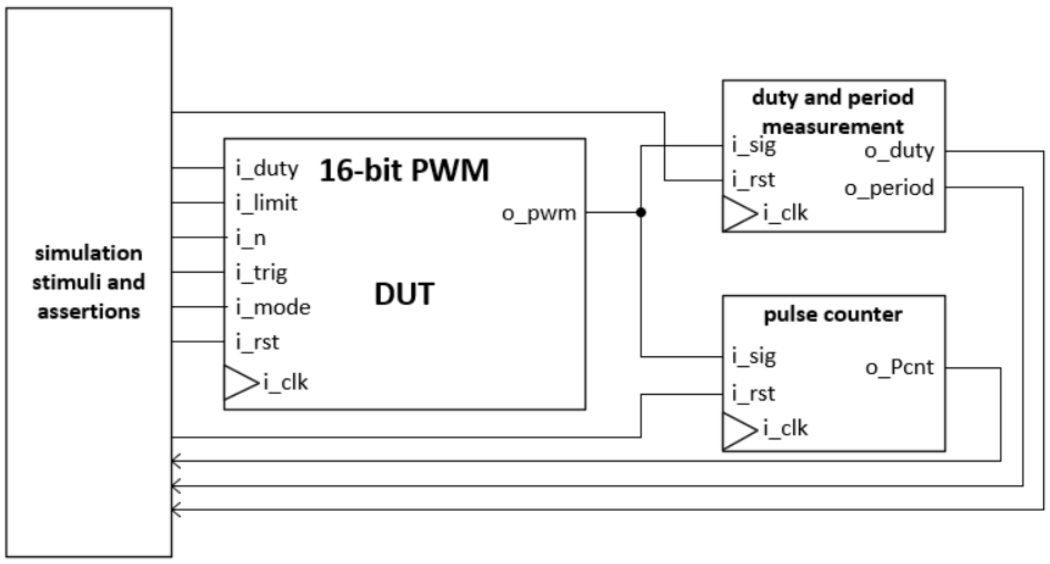
Verify the correct PWM frequency and duty cycle by measurements on the simulation waveforms.

1.3 Automatic test environment

Create an automatic test bench, where no visual analysis of signal waveform is necessary to assess the success or failure of a test run.

Automatic checks can be created by using *assert* statements within the testbench. These assertions typically check whether a signal is high or low, or whether the value of a vector is within a certain accepted range. With the $*display* system function, text messages about the success or failure of a test run can be printed to the console.

To automatically check the PWM's period, duty cycle and the number of generated pulses you will need to create two small measurement circuits:



Example test bench setup with the PWM circuit as the device under test. The test bench stimuli will configure the PWM block and let the simulation run for a certain amount of time. Then the results from the measurement circuits is evaluated and if the measurement matches the configuration, the test was successful and the next PWM configuration can be selected and the process repeats.

1.3.1 Specification of period and duty cycle measurement circuit

Create a circuit that can measure the period of an arbitrary 1-bit signal. It should output a new measurement value after every observed signal period. It should be possible to measure periods longer than those that can be generated by the PWM circuit.

Additionally, the high time of the input signal should be measured and output after every signal period. The high time and the period measurements will allow us to compute the signals duty cycle for each individual signal period.

The duty and period measurement must only provide valid measurements for the continuous PWM mode.

Verify that the circuit can properly measure signal periods by
applying some test signals. These can be created in a similar way as the simulation test clock:

```
logic test_sig = 0;
always #Xns test_sig = ~test_sig; #Yns test_sig = ~test_sig;
```

Where X is the signal's LOW time and Y the signals HIGH time.

1.3.2 Specification of the pulse counter circuit

Create a second test circuit that can measure the number of signal periods of a 1-bit signal. The circuit should increment the period count continuously until it either overflows or it is reset by its *i_rst* input.

1.3.3 Test runs

Create two separate test runs, one for the **continuous** and one for the **n-PWM-cycles** mode.
In each of the test runs an integer variable should define the number of tests to be executed.
For each test the inputs i_duty, i_limit and i_n should be defined by a random number generator. ($urandom or $urandom_range)
Count the number of successful and failed test runs and print them after all tests completed.

things to consider for the test runs

- i_duty is supposed to be smaller than i_limit, otherwise you mostly test the case of 100% duty cycle instead of values between 0-100%.
- consider keeping the PWM logic in reset while you modify i_limit and i_duty to make sure the initial conditions of each test are the same.
- how long do you have to simulate each test case so that also the lowest PWM frequencies can be successfully evaluated?
- in case it is necessary, chose a precision of 1% for your measurements

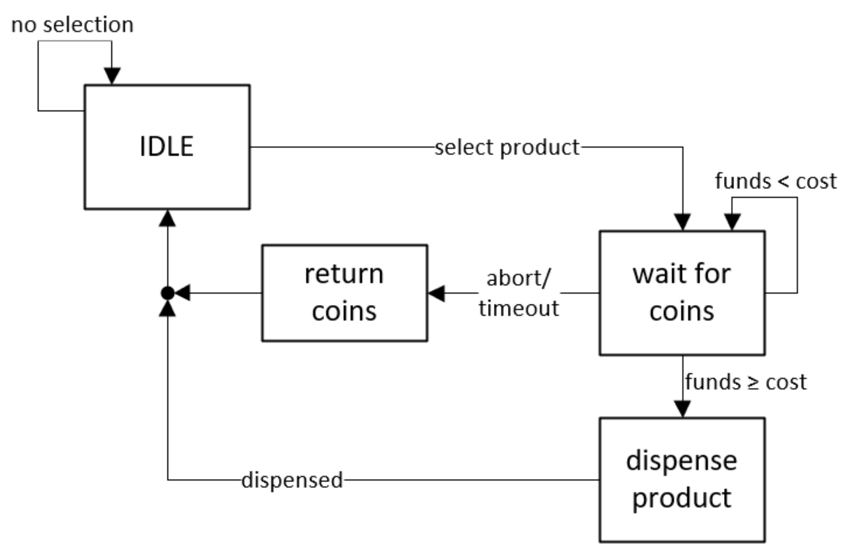
1.4 Report

Document the following:

- your duty and period measurement logic
- your pulse counter logic
- your PWM logic
  - the initial manual testbench and its results
  - the automatic testbench with its two test runs.
- the results of the automatic test bench with 100 tests (per run)
- the results of the automatic test bench where you purposely inserted errors (to see if the test is reporting the errors). This can be done by disconnecting one of the measurements signals and replacing it with a constant or maybe sometimes by the right value and sometimes by a wrong value.

2 STATEMACHINE

Create simple vending machine logic via modelling it as a state machine:



States and state transition conditions of vending machine

The vending machine can dispense three products and only accepts one-euro coins to stock up funds. It is supposed to have the following inputs and outputs:

direction, bit-width and name of signal	Functional description
input i_coin	pulse signal from coin receiver when a 1 euro coin was inserted
input [2:0] i_sel	signals of three buttons which select one of three products: bit 2 : energy drink 3€ bit 1 : soft drink 2€ bit 0 : sparkling water 1€
i_abort	abort and give back money
output o_coin_retn	signal for the coin receiver to return all currently inserted coins.
output [1:0] o_prod	2-bit binary signal to telling the dispenser which product to dispense
output o_disp	pulse signal for dispenser to dispense the product now.
input i_clk	100MHz clock
input i_rst,	Active high synchronous reset: Resets all internal memories to zero

Create a simple manual test bench where you test three cases:

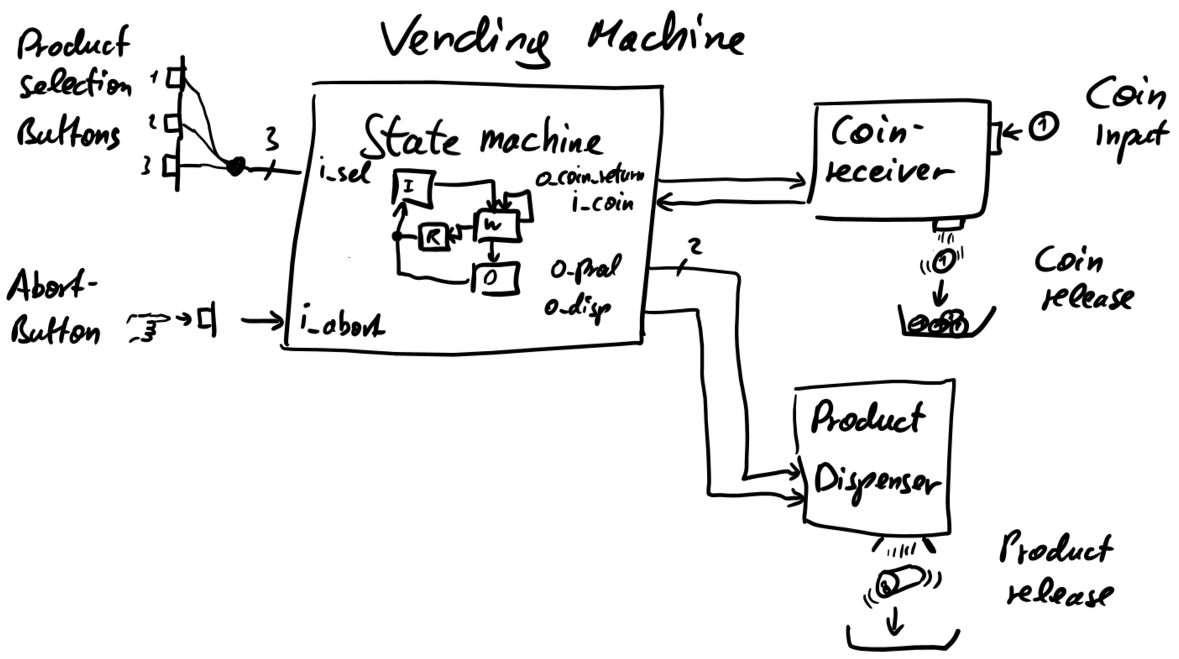
- dispensing of a soft drink
- dispensing of an energy drink interrupted by timeout after inserting 2 euros
- dispensing of water interrupted by a user abort, before inserting coins.

2.1 Documentation

Document your state machine logic, the test bench for your three test cases and the results of the simulation. (interpret you results! does the state machine behave as expected?)

2.2 Diagram of full vending machine

for better understanding how the state machine is supposed to interact with the different modules of the vending machine:



only the state machine must be coded!
It controls the interaction between the buttons, the coin receiver and the dispenser.

Remember to code the state machine in a way that the state change is coded separately from the actions performed within said states (as shown within the lecture slides).

Every state that has an immediate follower can directly switch to the next state.
Waiting, where the “next_state = current_state” is only required within the “wait” and “idle” state.
All other states immediately go to the next.