COEN 313 Lab 5 (UR-X)

by

Mamadou Diao Kaba

27070179

Submitted on

April 8th, 2023

"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"

Objective

The main objective of this lab is to become familiar with the ASM chart and control unit/datapath design methodology and in VHDL coding of datapath components and finite state machines at the RTL level.

Results

serial2scomplementer.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity twos_complementer is
port (din: in std_logic_vector(7 downto 0);
clk: in std_logic;
reset: in std_logic;

done_out: out std_logic;
counter_out: out std_logic_vector (3 downto 0);
reg_out: out std_logic_vector (7 downto 0)
);
end twos_complementer;

architecture Circuit of twos_complementer is

--define the internal signals
signal ld_done:std_logic;
signal clr_done:std_logic;
signal done: std_logic;
signal clr:std_logic;
```

```
signal inc:std_logic;
signal d,shift,load:std_logic;

signal counter:std_logic_vector (3 downto 0);
signal shift_reg:std_logic_vector (7 downto 0);



--define the state type
type state_type is
(load_state,test_bit,first_one,still_zero,test_count0,test_count1,flip_0,flip_1,finish,next_bit);
signal state:state_type;

begin
--Done Register
process(clk)
begin

if(clk'event and clk='1') then
if clr_done='1' then done <='0';
end if;
if ld_done='1' then done <='1';
end if;
end if;
end process;

--Counter Register
process(clk)
begin
if(clk'event and clk='1') then
if clr='1' then counter<="0000";
end if;
if inc='1' then counter<=std_logic_vector(unsigned(counter)+1);
```

```
end if;
end if;
end process;

--Shift Register
process (clk)
begin
if (clk'event and clk='1') then
if load='1' then shift_reg<=din;
else
if shift='1' then shift_reg<=D&shift_reg(7 downto 1);
end if;
end if;
end if;
end process;

--Next state logic
process (reset, clk)

begin

if (reset='1') then state<=load_state;
else
if (clk'event and clk='1') then
case state is
when load_state=>
state<=test_bit;

when test_bit=>
if shift_reg(0)='1' then
state<=first_one;
else state<=still_zero;
```

```
end if;

when first_one=>
state<=test_count1;

when still_zero=>
state<=test_count0;

when test_count0=>
if counter="1000" then
state<=finish; --It is possible that the din is “00000000”
else state<=test_bit;
end if;

when test_count1=>
if counter="1000" then
state<=finish; --It is possible that the din is “10000000”
else state<=next_bit;
end if;

when next_bit=>
if counter="1000" then
state<=finish;
else
if shift_reg(0)='0' then
state<=flip_0;
else state <=flip_1;
end if;
end if;

when flip_0=>
if counter="1000" then
```

```
state<=finish;
else state<=next_bit;
end if;

when flip_1=>
if counter="1000" then
state<=finish;
else state<=next_bit;
end if;

when finish=>
state<=finish;

end case;

end if;
end if;

end process;

--Moore output logic
process(state)
begin
case state is
when load_state=>

d<='0';
shift<='0';
ld_done<='0';
clr_done<='1';
clr<='1';
inc<='0';
```

```
load<='1';

when test_bit=>

d<='0';
shift<='0';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='0';
load<='0';

when first_one=>

d<='1';
shift<='1';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='1';
load<='0';

when still_zero=>

d<='0';
shift<='1';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='1';
load<='0';
```

```
when test_count0=>

d<='0';
shift<='0';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='1';
load<='0';

when test_count1=>

d<='0';
shift<='0';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='0';
load<='0';

when next_bit=>

d<='0';
shift<='0';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='0';
load<='0';

when flip_0=>
```

```
d<='1';
shift<='1';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='1';
load<='0';

when flip_1=>

d<='0';
shift<='1';
ld_done<='0';
clr_done<='0';
clr<='0';
inc<='1';
load<='0';

when finish=>

d<='0';
shift<='0';
ld_done<='1';
clr_done<='1';
clr<='1';
inc<='0';
load<='0';

end case;
end process;

end Circuit;
```

serial2scomplementer.xdc

```
# Vivado does not support old UCF syntax
# must use XDC syntax

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk];
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [ get_ports { din[0] }  ] ;
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [ get_ports { din[1] }  ] ;
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [ get_ports { din[2] } ] ;
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { din[3] }];
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { din[4] }];
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { din[5] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { din[6] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { din[7] }];
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { reset }];

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [ get_ports { reg_out[0] }  ] ;
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [ get_ports { reg_out[1] }  ] ;
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [ get_ports { reg_out[2] }  ] ;
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { reg_out[3] }];
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { reg_out[4] }];
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { reg_out[5] }];
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { reg_out[6] }];
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { reg_out[7] }];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { counter_out[0] }];
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { counter_out[1] }];
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { counter_out[2] }];
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { counter_out[3] }];
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { done_out }];
```

Questions

1. Examine the .vds log file contained in the synth_1 directory created by the Xilinx Vivado tools. Your design contains a finite state machine. Comment on the state encoding used by the synthesis tool.

The .vds log file shows that the synthesis tool uses one-hot encoding. Each state takes a separate flip flop, and the combination of bits has only a single high and all the others are low.

2. Comment on the advantages/disadvantages of a binary state encoding vs. a one-hot state encoding. Do other state encoding techniques exist?

The advantages of binary state encoding compared to a one-hot state encoding are that it uses m number of flip-flop where 2^m >=n number of states (therefore, highest coding efficiency) compared to m=n for a one-hot encoding and it has potentially few illegal states (easy to detect) compared to many illegal states (complex circuit to detect) for the one-hot encoding. Its main disadvantage compared to the one-hot encoding is that it is mainly used for smaller state machines. Whereas the one-hot encoding can be used for larger state machines.

There also exists the Gray encoded state machine and the modified One-hot finite state machine

Conclusion

To conclude, this final lab offered a well-versed understanding in VHDL coding of datapath components and finite state machines at the RTL level.