Digital Systems Design II

COEN 313 (Section FJ-X)

Experiment 5:
A Serial Two's Complementer Usuing ASM Chart and Dataphath/Control Unit Design Methodology

Andre Hei Wang Law

4017 5600

Performed on Monday, November 15, 2021

Due on Tuesday, December 14, 2021

“I certify that this submission is my original work and meets the Faculty’s Expectations of Originality”.

Tuesday, December 14, 2021

Andre Hei Wang Law

4017 5600

1) Objectives

For the final experiment of the course Coen 313, students have a main objective of becoming familiar with the ASM chart, control unit and datapath design methodology. Then, they will also learn to be well-versed in VHDL coding of datapath components and finite state machine at the RTL level.

2) VHDL Code

```
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);
                  reset : in std_logic ;
                  clk :    in std_logic ;
                  done_out :              out std_logic ;
                  reg_out :               out std_logic_vector(7 downto 0));
end twos_complementer ;

architecture arch_twos of twos_complementer is

type state_type is (start, loading, lsb_0, lsb_1, invert);
```

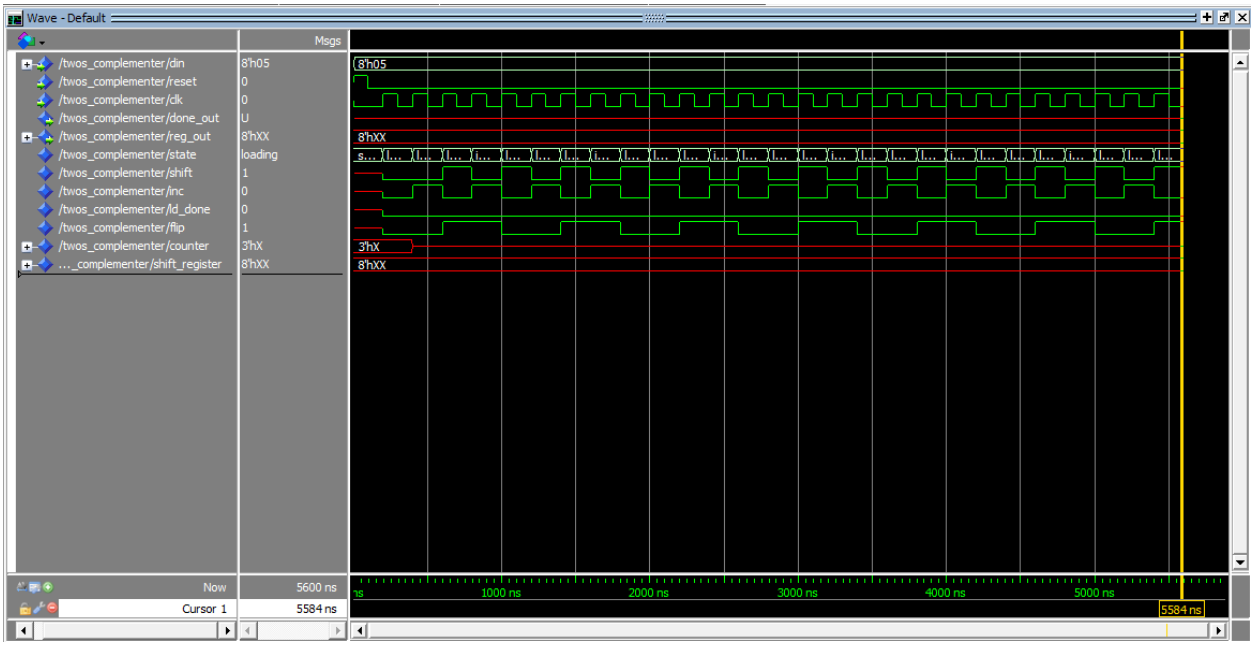
```
signal state : state_type;
signal d, shift, load, inc, clr, ld_done, clr_done, flip : std_logic;
signal counter : std_logic_vector(2 downto 0);
signal shift_register, temp_reg : std_logic_vector(7 downto 0);

begin
process(reset, clk, counter, shift_register)
begin

if (reset = '1') then
        counter <= "000";
        state <= start;
elsif (clk'event and clk='1') then
        case state is
                when start =>
                        counter <= "000";
                        shift_register <= "00000000";
                        load <= '1';
                        shift <= '0';
                        inc <= '0';
                        clr <= '1';
                        ld_done <= '0';
                        clr_done <= '1';
                        flip <= '0';
                        state <= loading;
                when loading =>
                        shift <= '0';
                        load <= '0';
                        clr <= '0';
                        clr_done <= '0';
```

```
            if (counter = "111") then
                ld_done <= '1';
                done_out <= '1';
            else
                if (flip = '1') then
                    inc <= '1';
                    state <= invert;
                else
                    inc <= '1';
                    if (shift_register(0) = '1') then
                        state <= lsb_1;
                    else
                        state <= lsb_0;
                    end if;
                end if;
            end if;
        when lsb_0 =>
            d <= shift_register(0);
            inc <= '0';
            shift <= '1';
            flip <= '1';
            state <= loading;
        when lsb_1 =>
            d <= shift_register(0);
            inc <= '0';
            shift <= '1';
            flip <= '0';
            state <= loading;
        when invert =>
            d <= not(shift_register(0));
```

```
                    inc <= '0';
                    shift <= '1';
                    flip <= '0';
                    state <= loading;
                when others =>
        end case;
end if;
end process;

process (clk, inc, clr)
begin
        if (clr = '1') then
                counter <= "000";
                clr <='0';
        end if;
        if (inc = '1') then
                counter <= std_logic_vector(unsigned(counter) + 1);
        end if;
end process;


process (clk, din, d, shift, load)
begin
        if (load = '1') then
                shift_register <= din;
                load <= '0';
        end if;
        if (shift ='1') then
                temp_reg <= shift_register(6 downto 1) & d & shift_register(7);
        end if;
```

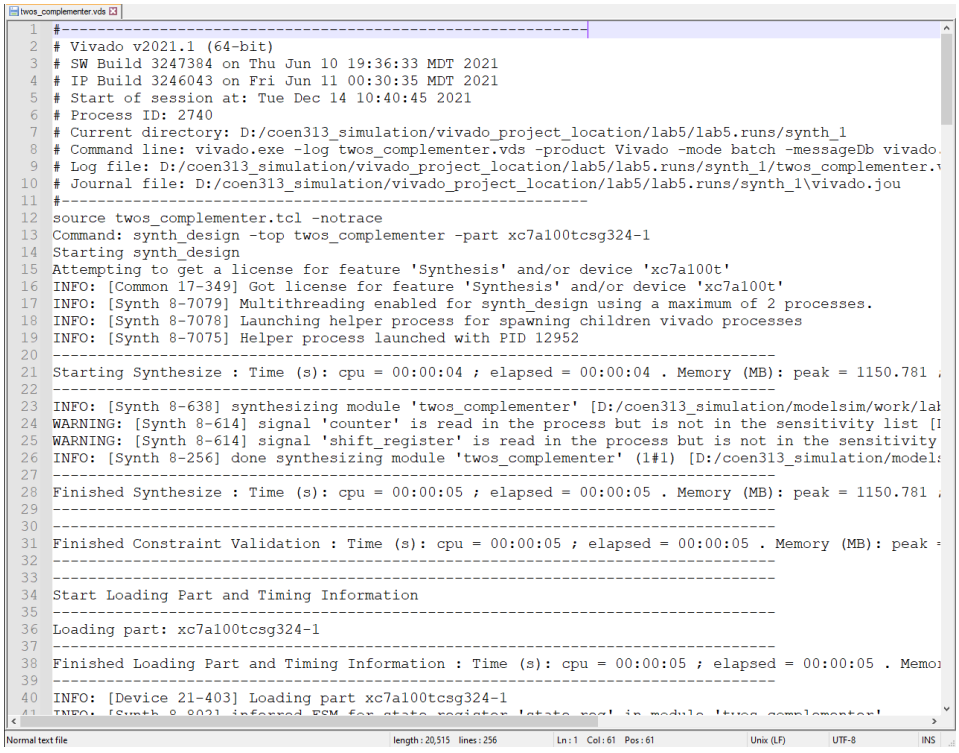
```
end process;

reg_out <= not (temp_reg);

process (clk, clr_done, ld_done)
begin
        if (clr_done = '1') then
                done_out <= '0';
                clr_done <= '0';
        end if;
end process;
end architecture;
```

3)Simulation Results



3.1 Modelsim Simulation Results

4) Synthesis and Implementation Log Files



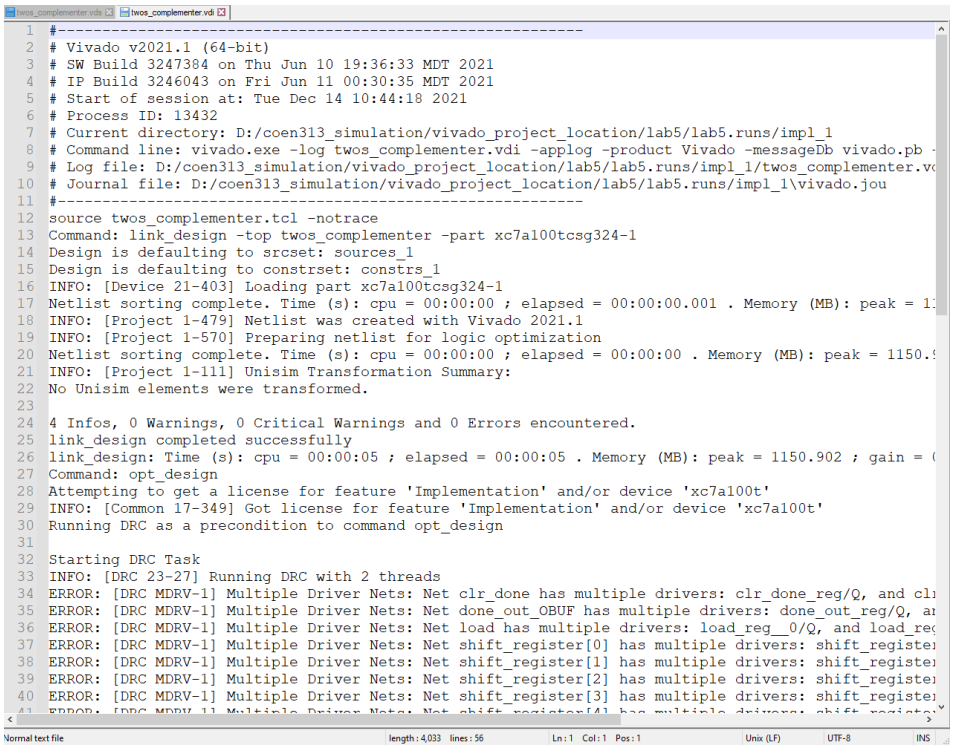
```
#-----------------------------------------------------------
# Vivado v2021.1 (64-bit)
# SW Build 3247384 on Thu Jun 10 19:36:33 MDT 2021
# IP Build 3246043 on Fri Jun 11 00:30:35 MDT 2021
# Start of session at: Tue Dec 14 10:40:45 2021
# Process ID: 2740
# Current directory: D:/coen313_simulation/vivado_project_location/lab5/lab5.runs/synth_1
# Command line: vivado.exe -log twos_complementer.vds -product Vivado -mode batch -messageDb vivado.
# Log file: D:/coen313_simulation/vivado_project_location/lab5/lab5.runs/synth_1/twos_complementer.v
# Journal file: D:/coen313_simulation/vivado_project_location/lab5/lab5.runs/synth_1\vivado.jou
#-----------------------------------------------------------
source twos_complementer.tcl -notrace
Command: synth_design -top twos_complementer -part xc7a100tcsg324-1
Starting synth_design
Attempting to get a license for feature 'Synthesis' and/or device 'xc7a100t'
INFO: [Common 17-349] Got license for feature 'Synthesis' and/or device 'xc7a100t'
INFO: [Synth 8-7079] Multithreading enabled for synth_design using a maximum of 2 processes.
INFO: [Synth 8-7078] Launching helper process for spawning children vivado processes
INFO: [Synth 8-7075] Helper process launched with PID 12952
---------------------------------------------------------------------------------
Starting Synthesize : Time (s): cpu = 00:00:04 ; elapsed = 00:00:04 . Memory (MB): peak = 1150.781 ;
---------------------------------------------------------------------------------
INFO: [Synth 8-638] synthesizing module 'twos_complementer' [D:/coen313_simulation/modelsim/work/lab
WARNING: [Synth 8-614] signal 'counter' is read in the process but is not in the sensitivity list [D
WARNING: [Synth 8-614] signal 'shift_register' is read in the process but is not in the sensitivity
INFO: [Synth 8-256] done synthesizing module 'twos_complementer' (1#1) [D:/coen313_simulation/models
---------------------------------------------------------------------------------
Finished Synthesize : Time (s): cpu = 00:00:05 ; elapsed = 00:00:05 . Memory (MB): peak = 1150.781 ;
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Finished Constraint Validation : Time (s): cpu = 00:00:05 ; elapsed = 00:00:05 . Memory (MB): peak =
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Start Loading Part and Timing Information
---------------------------------------------------------------------------------
Loading part: xc7a100tcsg324-1
---------------------------------------------------------------------------------
Finished Loading Part and Timing Information : Time (s): cpu = 00:00:05 ; elapsed = 00:00:05 . Memor
---------------------------------------------------------------------------------
INFO: [Device 21-403] Loading part xc7a100tcsg324-1
INFO: [Synth 8-802] inferred FSM for state register 'state_reg' in module 'twos_complementer'
```

Normal text file | length: 20,515 lines: 256 | Ln: 1 Col: 61 Pos: 61 | Unix (LF) | UTF-8 | INS

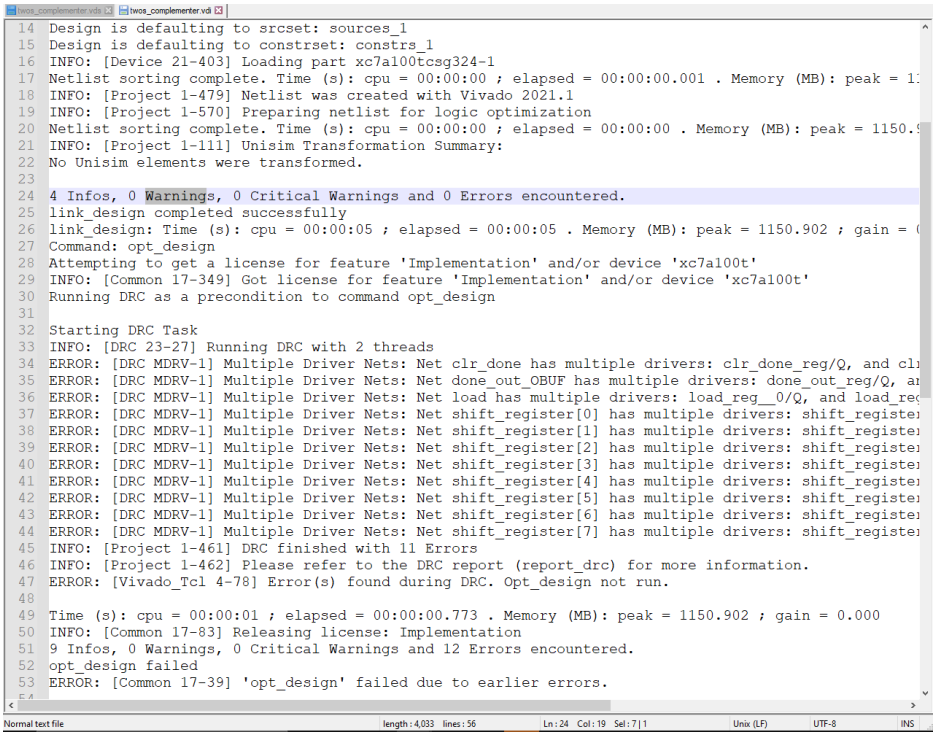
4.1.a Beginning of Synthesis Log File

```
|12    |IBUF |    10|
|13    |OBUF |     9|
+------+-----+------+

Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |    79|
+------+---------+-------+------+
---------------------------------------------------------------------------------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:12 ; elapsed = 00:00:12 . Memory (MB): pea
---------------------------------------------------------------------------------
Synthesis finished with 0 errors, 31 critical warnings and 9 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:12 ; elapsed = 00:00:12 . Memory (MB): peak =
Synthesis Optimization Complete : Time (s): cpu = 00:00:12 ; elapsed = 00:00:12 . Memory (MB): peak
INFO: [Project 1-571] Translating synthesized netlist
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 1150.7
INFO: [Netlist 29-17] Analyzing 19 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 1150.7
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 19 instances were transformed.
  LD => LDCE: 19 instances

Synth Design complete, checksum: 99dbca64
INFO: [Common 17-83] Releasing license: Synthesis
16 Infos, 9 Warnings, 31 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:14 ; elapsed = 00:00:15 . Memory (MB): peak = 1150.781 ; gain =
INFO: [runtcl-6] Synthesis results are not added to the cache due to CRITICAL_WARNING
INFO: [Common 17-1381] The checkpoint 'D:/coen313_simulation/vivado_project_location/lab5/lab5.runs/
INFO: [runtcl-4] Executing : report_utilization -file twos_complementer_utilization_synth.rpt -pb tw
INFO: [Common 17-206] Exiting Vivado at Tue Dec 14 10:41:03 2021...
```

4.1.b Example of a Synthesis Log File Warning Messages



4.2.a Beginning of Implementation Log File



4.2.b Example of an Implementation Log File Warning Messages

The messages, errors and warnings generated by Vivado can be useful in pinpointing problems and help explaining why this area has an issue. These log files are found in the Vivado project directory in which the synthesis log file is under "/.runs/synth_1/entity_name.vds" while the implementation log file is under "/.runs/impl_1/entity_name.vdi". The above figure 4.1.b and 4.2.b are some examples of warning messages encountered during implementation and synthesis, respectively. The importance of these generated messages or warning is to aide the user to locate potential problems and solve them easily by reading the type of error messages. In the long run, unresolved minor warnings can cause unwanted results, thus promoting the user to always solve their "warnings", "critical warning" and "errors encountered". The best-case scenario is when the log file indicates that there are zero problems. Notice that in this case, the Modelsim simulation results had issues while still being able to be synthesised and implemented. The cause of the unwanted red line (no signal) from figure 3.1 can be explained by the many critical warnings and warnings such as shown in figure 4.1.b. Overall, it is a good practice to revise the VHDL code in order to solve all these errors, critical warnings and warnings.

5) Bit File

6.1 Bit File of the Two's Converter (from NotePad++)

6) Questions

6.1) 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.

```
INFO: [Device 21-403] Loading part xc7a100tcsg324-1
INFO: [Synth 8-802] inferred FSM for state register 'state_reg' in module 'twos_complementer'
WARNING: [Synth 8-327] inferring latch for variable 'done_out_reg' [D:/coen313_simulation/modelsim/work/lab_5.vhdl:115]
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |               Previous Encoding
---------------------------------------------------------------------------------------------------
                   start |                            00001 |                              000
                 loading |                            00010 |                              001
                  invert |                            00100 |                              100
                   lsb_1 |                            01000 |                              011
                   lsb_0 |                            10000 |                              010
```

Figure 6.1 Encoding State from the Synthesis Log File

This is a one-hot state encoding. This can be deduced based on the fact that the “new encoding” section has as many encodings as there are of bits in which only one of them are “ON” while the rest remains “OFF”.

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

The advantage of binary state encoding is that the encoding is a straightforward. The assigned values go sequentially to the states in which as few bits are used to encode these states. On the other hand, a one-hot state encoding uses one ON bit and remaining OFF bits to represent the current state. The overall total of bits used is excessively larger than the binary state encoding. However, its strength lies with its simplification where the stimulus logic for the flip flips doesn’t need to be decoded as the bits itself are the states. Yes, there exists another encoding technique and it is called the “Gray Encoding” which consists of a sequency of one-bit changes between one value to the next one. Its advantage is that this technique uses less amounts of bits.

6.3) Consider a datapath which contains a register which has a control signal called “LOAD_MICK” which acts as a synchronous load control signal for the register. Explain by means of a timing diagram and in words the error contained in the following ASM chart (Figure 3):

The error of figure 3 is that “LOAD_MICK” should not be inside Keith while its condition expression, “MICK = 6”, is below. Rather, we should load the signal first by means the synchronous load control signal, then go through the Keith state and its condition expressions.