DATE: 12/10/2024
TO: Prof. Riadul Islam, TA Dhandeep Challagundla
FROM: Joaquin Calilao, Bardan Kafle
SUBJECT: CMPE 413 Project Report Final

---

1 Project Objective

The CMPE 413 Project involves developing a **byte-addressable cache memory chip**, by utilizing structural VHDL. The cache memory consists of 4 blocks each containing 4 bytes for a total of 16 addressable bytes that can be read and written to. In order to achieve read/write functionality, control logic was developed and implemented structurally in VHDL. Signals provided by the CPU entering the chip will drive the control logic and perform the appropriate operations on the memory.

Additionally, the control logic is able to determine whether a certain memory address can be validly accessed and modified, and will perform further operations to accommodate those invalid accesses to certain memory locations.

2 Design Overview

2.1 Cache Memory

As aforementioned, the cache memory table consists of 4 blocks each containing 4 bytes. In order to create memory, 1-bit cache cells were developed per the specifications provided in the lab description. The decoder consists of a 2-to-4 decoder, D-latch, and a transmission gate provided in the lab project blackboard folder.

After developing the cache cell design, an 8-bit cache-byte was designed and consisted of 8 parallel cache cells. Then a 4-byte cache block consisting of 4 cache-bytes, and finally a 16-byte cache table consisting of 4-cache blocks.

A matrix at the end of this section describes the hardware interconnect of the signals for each layer of the cache memory hierarchy (starting from cell up to the cache top). Additionally, the discussion for the implementation of control logic to read/write to the cache memory is discussed in further detail in Section 2.2

This space is intentionally left blank.

Below are the RTL schematics for each layer of the cache memory hierarchy. Starting with the top level and going through to the cache cell schematic.

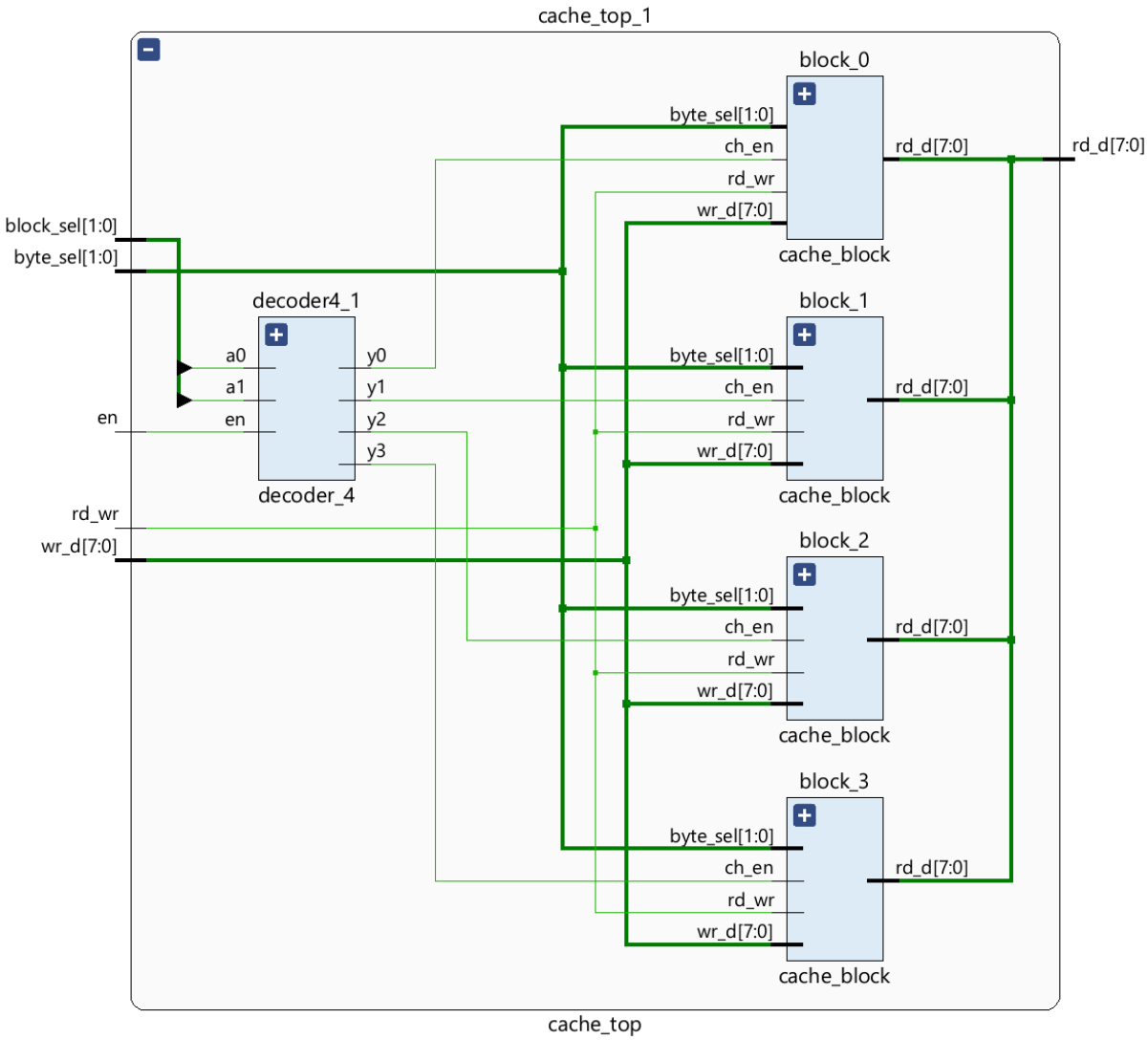

Figure 1: RTL Schematic of cache_top.vhd

Notice that the cache top contains a decoder which is primarily responsible for selecting one of the 4 cache blocks to be accessed at a particular time.

Also, notice that the rd_wr and wr_d signal is distributed amongst all cache blocks, showing that read/write mode is set for the entire cache, but only the block selected (and byte selected for that matter) is actually read from or written to at any given time, thanks to the decoder driving the ch_en signal.

The following diagram shows the inner schematic of 1 cache block.

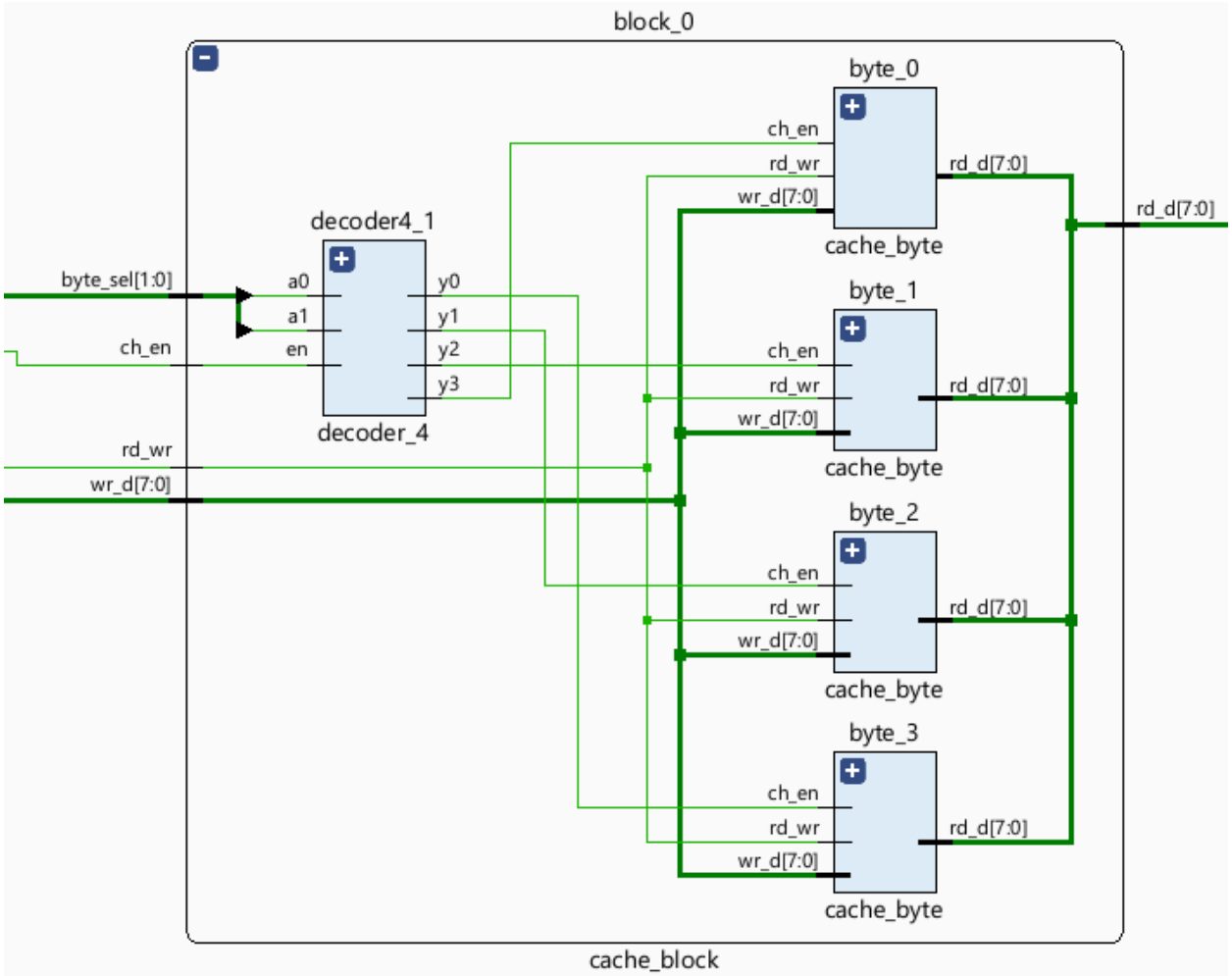

Figure 2: RTL Schematic of cache_block.vhd

Recall the byte select signal from the top level cache which is also being used here. Each cache block gets its own decoder and uses the byte select signals to select which specific byte to access for the block.

Since this byte_select signal is distributed to all 4 blocks, essentially byte selected is selecting the entire column of bytes, but only the specific byte that is in the selected block is actually accessed.

This is because the output of the decoder in the top level cache is connected to the en pin of the decoder of @ the cache_block level. Thus, if the block is not selected, it cannot select a bit to access. Only the block selected can select a byte. Remember however that the entire cache is set to read or write at any given time.

This space is intentionally left blank.

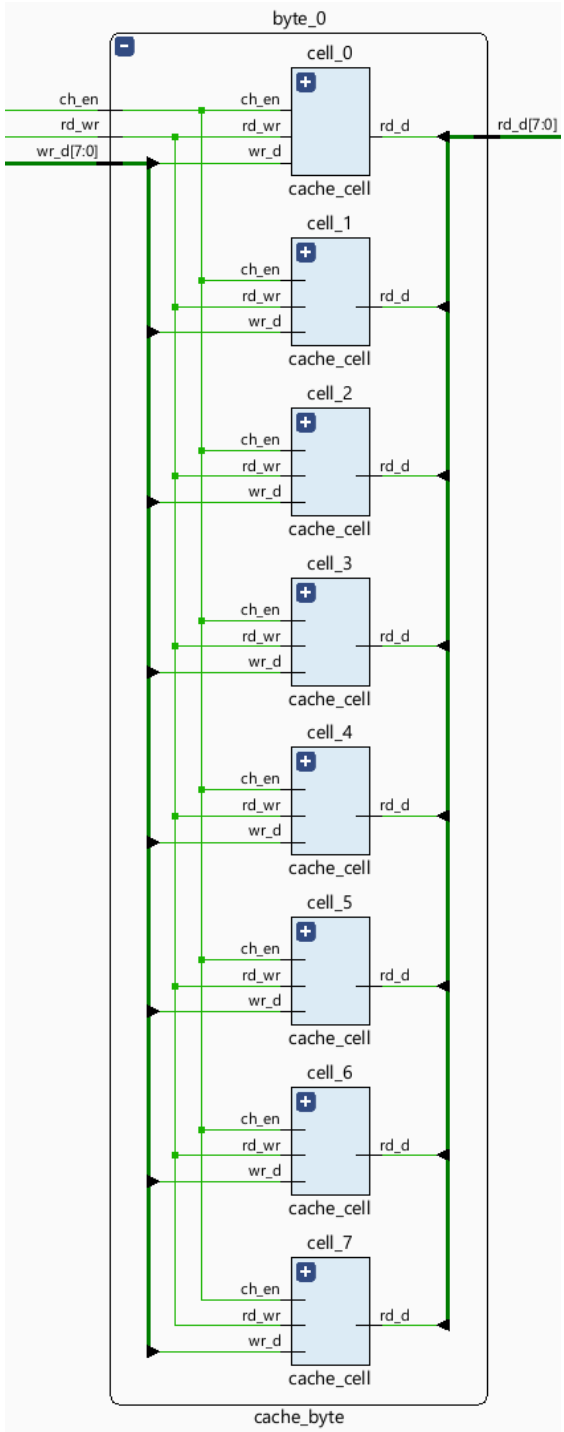

Figure 3: RTL Schematic of cache_byte.vhd

Notice that for the cache byte, the ch_en signal is distributed to all cells to access the whole byte when it is selected. The same can be said about rd_wr signals. However, wr_d (write data) is sent in as an 8 bit vector, with one bit being sent to each respective cell.

There are some notable control signals on the cache memory that are wired in a specific way in order to make the memory byte-addressable.

Table 1: Cache Memory Hardware Interconnect (Full Hierarchy)

Signal	**Direction**	**Bit Width**	**Description**	**Associated Component**
rd_d	output	1	1-bit data that can be read from the cell	*cache_cell.vhd*
ch_en	input	1	signal that enables access to the cell	*cache_cell.vhd*
rd_wr	input	1	signal that sets read or write mode for the cell	*cache_cell.vhd*
wr_d	input	1	1-bit data that can be written into the cell	*cache_cell.vhd*
rd_d	output	8	8-bit data that can be read from the cell	*cache_byte.vhd*
ch_en	input	8	signal that enables access to all bits in the byte	*cache_byte.vhd*
rd_wr	input	8	signal that sets read or write mode for all bits	*cache_byte.vhd*
wr_d	input	8	8-bit data that can be written into the cell	*cache_byte.vhd*
rd_d	output	8	8-bit data that can be read from the cell	*cache_block.vhd*
byte_sel	input	2	2-bit address input that selects a byte via a decoder	*cache_block.vhd*
ch_en	input	1	enables the decoder that selects a byte	*cache_block.vhd*
rd_wr	input	1	signal that sets read or write mode for all bytes	*cache_block.vhd*
wr_d	input	8	8-bit data that can be written into the cell	*cache_block.vhd*
rd_d	output	8	1-byte output data from the cache memory	*cache_top.vhd*
block_sel	input	2	2-bit address for selecting which block to access via a 2-to-4 decoder	*cache_top.vhd*
byte_sel	input	2	2-bit address for selecting which byte to access via a 2-to-4 decoder	*cache_top.vhd*
wr_d	input	32	32-bit input data that is written into an accessed byte (only the selected byte is overwritten)	*cache_top.vhd*

This space is intentionally left blank.

Below are the simulation views that verify the correct operation of **cache_top.vhd** and **cache_cell.vhd**. The testbench files are included in the submission along with the VHDL module code.

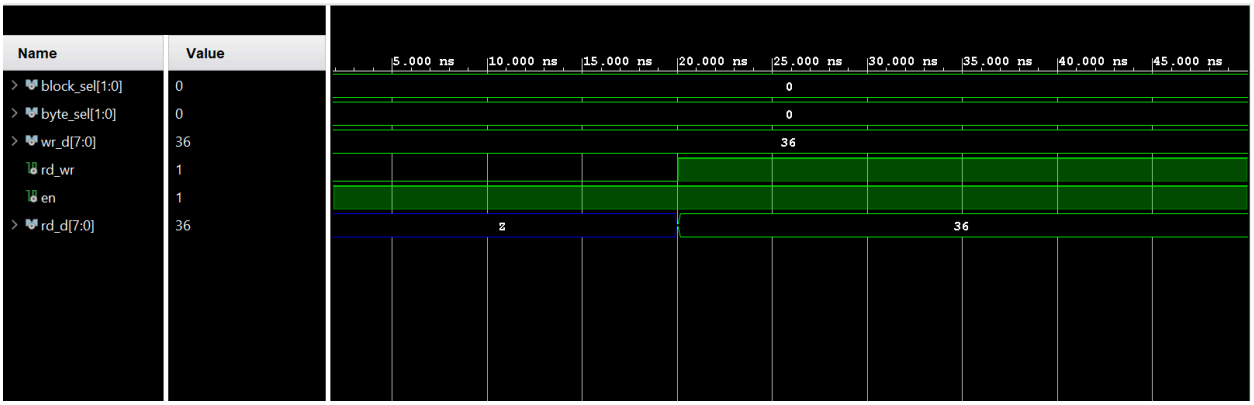

Figure 4: cache_top.vhd simulation waveform

In the snippet of cache_top test shown above, it can be seen that block and byte select are both 0, meaning that we select byte address 0000 in the cache memory. Then, from t = (0, 20)ns we see that en = 1 and rd_wr = 0, meaning the cache is enabled in write mode. The value of 36 in hex is written to address 0000 in this time. Then at 20 ns, rd_wr = 1 indicating that the cache is in read mode. From here the data is expected to output the same value that we entered during the write operation which it shows as rd_d goes from Z to 36 at the 20 ns time step. So the cache_top module works as expected.

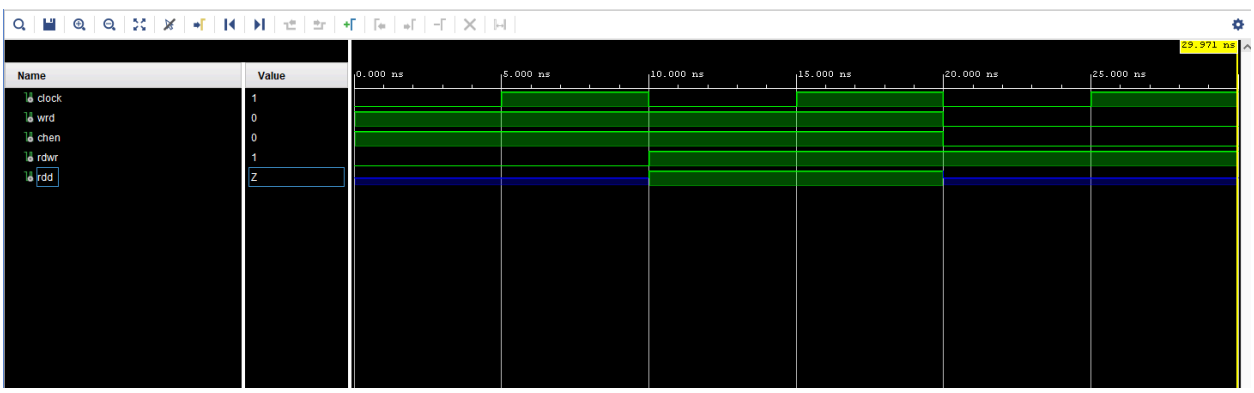

Figure 5: cache_cell.vhd simulation waveform

In the snippet of cache_cell test shown above, it can be seen that initially, wr_d = 1, ch_en = 1, and rd_wr = 0 when t = (0, 10) ns. Since this means the cell is in write mode, and we want to write a 1 into the D-latch of the cache cell. To verify that the write operation was successful, at t = (10, 20) ns, the cache cell is set to read mode indicated by rd_wr = 1. At this point, the output of the cell should be the 1 that we wrote to it during t = (0, 10) ns, since we now want to read from it. This is correctly indicated by the output of 1 by rd_d when t = {10, 20} ns. At t = (20, 30)ns, ch_en is set low, meaning that the cell is disabled. Now, no output should be read at all, and this is correctly indicated by the high Z from rd_d.

2.2 Control Logic

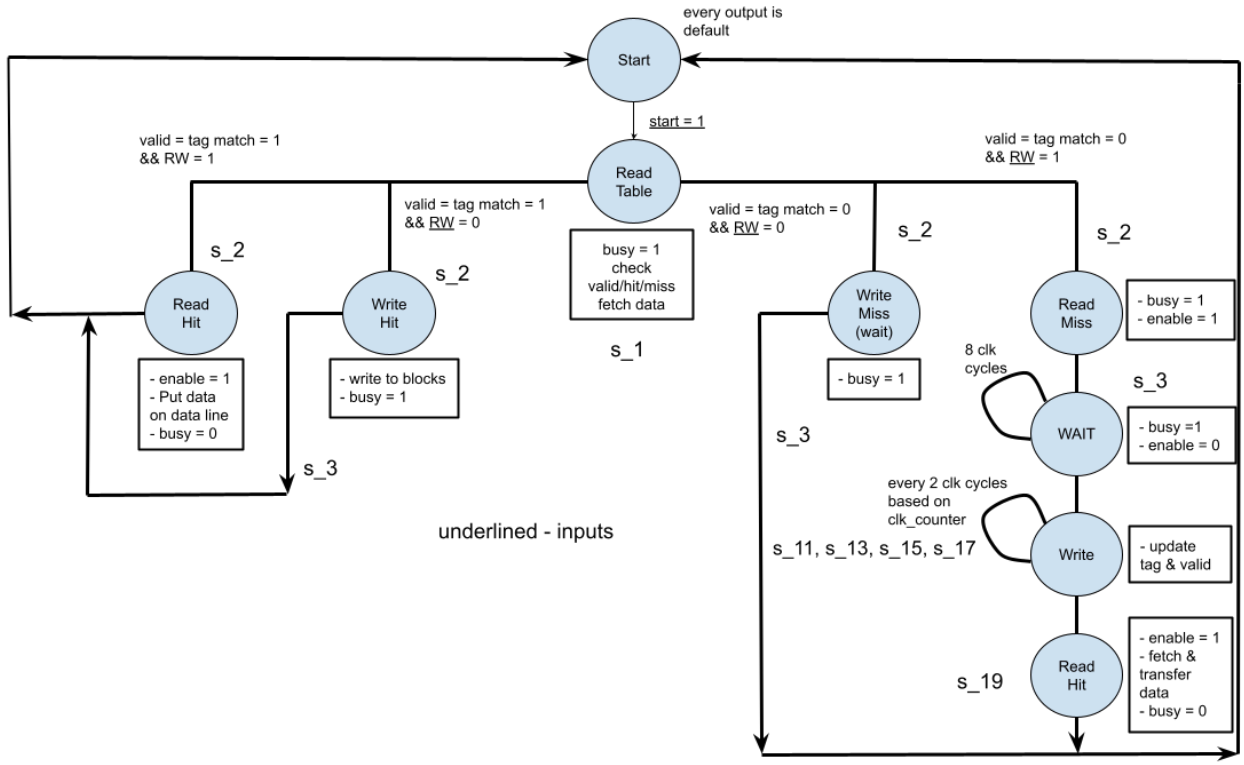

Figure 6: State Machine Flow Diagram for Control Logic

The control logic which involves writing and reading to the cache table as well as sending signals to the CPU revolves around a shift register. In our case, this comes in the form of a ring counter where when the 'busy' signal is turned on, the position of the program based on each negative clock cycle can be tracked. The figure below shows the individual waveform for the ring counter.

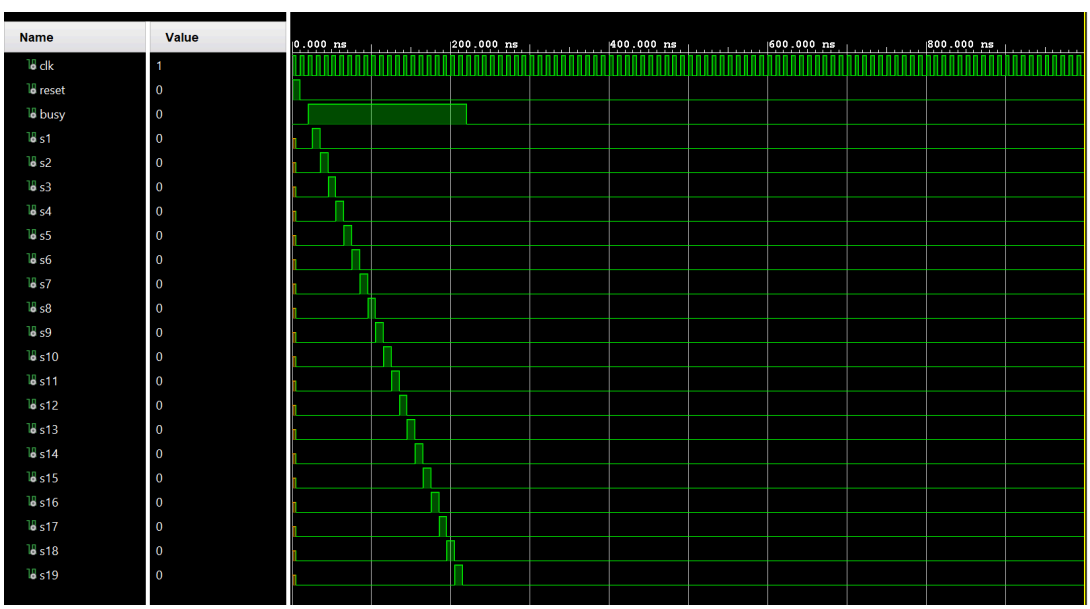

Figure 7: clk_counter.vhd simulation waveform

As observed above, the counter allowed us to perform operations based on the exact clock cycle they should happen. This made the project easier to implement than trying to make a dynamic state machine.

2.1.1 Clock Cycles

Assume cycle 1 starts on the negative edge of the clock immediately after the 'start' signal is asserted.

Cycle 1:

The busy signal becomes active. This starts the first action of the cache program which is the Hit/Miss check. The clock counter also begins and this cycle is referenced as signal s1 from the counter.

Cycle 2:

At this point, the operation specific functionality begins based on whether the CPU_ADD was a hit or miss and the given read/write state:

CLOCK COUNTER STATE : S2	
Cache Operation	Function
Read Hit	- Data is fetched from cache architecture and transmitted out to CPU_DATA - Busy is set to low and output enable signal becomes active
Read Miss	MEM_EN is set active and MEM_ADD is provided back to the CPU
Write Hit	Addressed block is written to using latched CPU_DATA
Write Miss	Idle

Cycle 3:

Read Hit operation is finished.

CLOCK COUNTER STATE : S3	
Cache Operation	Function
Read Miss	MEM_EN and MEM_ADD reset
Write Hit	Busy is set to low
Write Miss	Busy is set to low

Cycle 4 - 10:

Write Hit and Write Miss are finished on cycle 4.

CLOCK COUNTER STATE : S4 - S10	
Cache Operation	Function
Read Miss	Idle, waiting for data from memory (First byte of data arrives on cycle 10, however we stay idle)

Cycle 11 - 18:

CLOCK COUNTER STATE : S11 - S18	
Cache Operation	Function
Read Miss	Bytes received from memory are written to the cache (each byte is held for two clock cycles therefore, writing happens on S11, S13, S15, and S17)

Cycle 19:

CLOCK COUNTER STATE : S19	
Cache Operation	Function
Read Miss	Busy signal is set to low and the desired byte is written back to the CPU using CPU_DATA

2.3 Top Level Design Hierarchy and Simulation

Below is a hierarchical block diagram of the final design for the cache memory implementation. Next to the arrows are the associated signals being passed across each block.

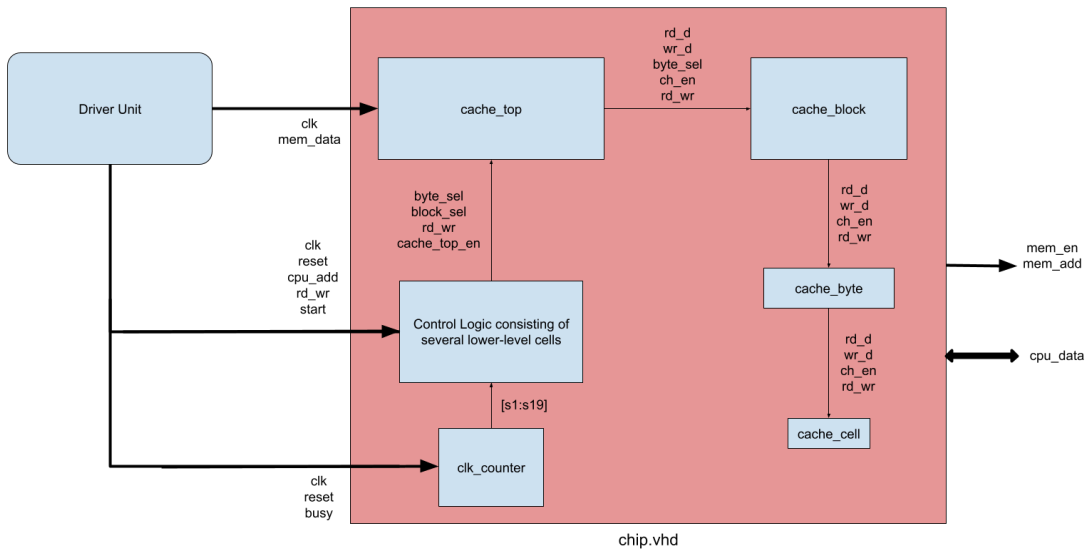

Figure 8: Top level block diagram of cache memory design w/ signals

The figure above shows the final top-level architecture for the cache memory design. "**chip.vhd**" acts as a HDL wrapper for all the higher-level cells as well as some lower-level cells and gate-level primitives that constitute the control logic. In practice, the signals passed to the chip are driven by a standalone CPU, however a testbench provided with the lab description acts as the driver to test the design.

clk and **mem_data[7:0]** are fed directly to the cache top. **mem_data** can be written to a selected byte in the cache if it is in write mode. The bulk of the control logic works in tandem with the **clk_counter** to determine for how many clock cycles each operation should last (Read Hit/Miss or Write Hit/Miss). The **clk_counter** module will reset its count whenever reset=1 or busy=0.

<u>The control logic mainly consists of lower-level cells such as decoders, a mux, and several gate-level primitives.</u> Most of the control logic is structurally coded directly under **chip.vhd** and does not belong to an independent module. It is responsible for instructing the **cache_top** module to perform certain operations depending on the **rd_wr** signal. The control logic itself determines whether the operation is a hit or a miss by checking the valid and tag bits associated with the cache.

Furthermore, the control logic essentially parses the incoming bits from cpu_add[5:0] (CPU Address signal) into the bit order specified in the project document to be sent to **cache_top** (refer to Section 2.1 for detail regarding how block/byte offset bits are used):

2	2	2
Tag	**Block Offset**	**Byte Offset**

During read operations, 3 outputs are derived from **chip.vhd**: **mem_add**, **mem_en**, and **cpu_data**. **cpu_data** is the data read from a cache memory byte that was specified by **cpu_add**. **mem_add** and **mem_en** are signals that are associated with accessing the main memory, separate from the cache.

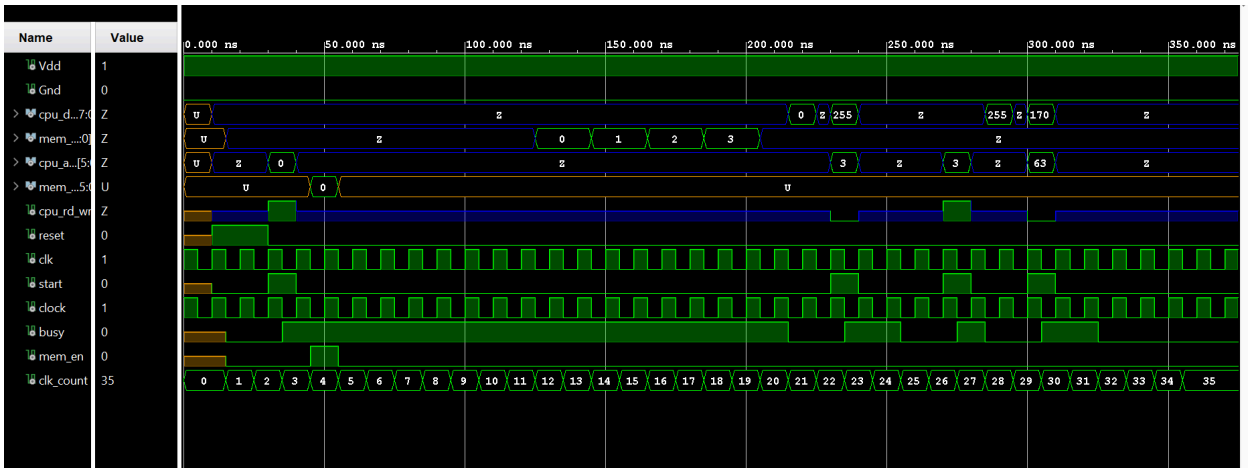

Figure 9: chip.vhd simulation waveform

The snippet of chip_test shown above is derived from the provided testbench along with the project description. Based on the waveforms provided in the project description, it can be seen that a Read Miss occurs at address 00000000 operation is happening first

clk_count 22 to 29 test Write Hit and Read Hit. Write Hit and Read Hit work as intended, based on the results shown in the waveform provided in the lab project description.

Finally, starting from **clk_count** 29, Write Miss is being tested. When compared to the provided waveform, the simulation view shows that a write attempt is being made at clock cycle 30, since its a write miss, no other operation should occur after this point, and that is also shown in the waveform above as well as the provided waveform in the project description.

This space is intentionally left blank.

2.3 Lower Level Cells

The following section provides simulation results for lower-level cells: **comparator2.vhd** and **mux_1bit_4to1.vhd**.

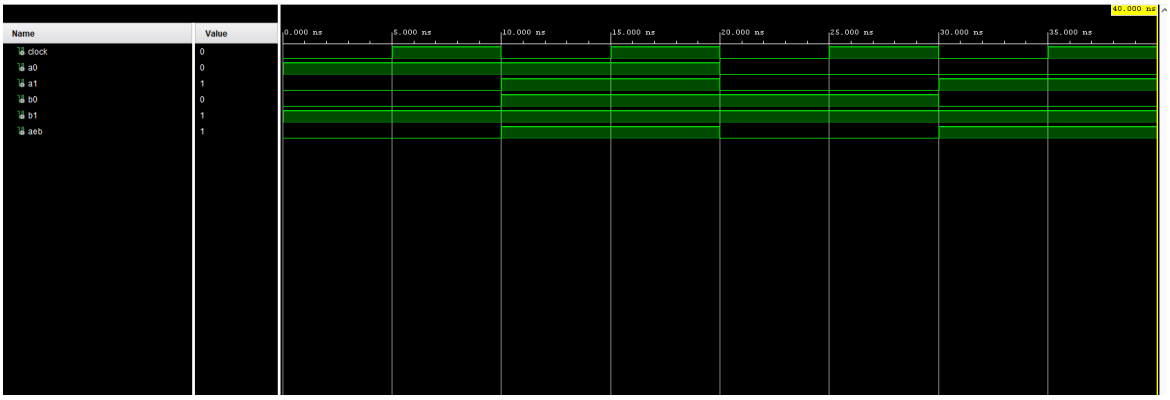

Figure 10: comparator2.vhd simulation waveform

In the comparator2_test simulation screenshot shown above, we compare two, 2-bit inputs **a** and **b**. **aeb** represents the "a equals b" signal. Notice at t = 5 ns, **a** = 01, **b** = 10. Since **a** doesn't equal **b**, we don't expect aeb to go high, which the waveform demonstrates. Then at t = 10 ns, **a** = 11, **b** = 11. Since the inputs are equal, we expect to see **aeb** = 1, which the waveform shows for the duration of time for which a is the same value as **b**. Two comparators are used in the control logic. One of them is used to determine if the tag bits sent by cpu_add are correct for the address in the cache memory being specified. The other comparator handles the valid bit. The output of these two comparators is ANDed together and is used to determine if the operation is a Hit or Miss.

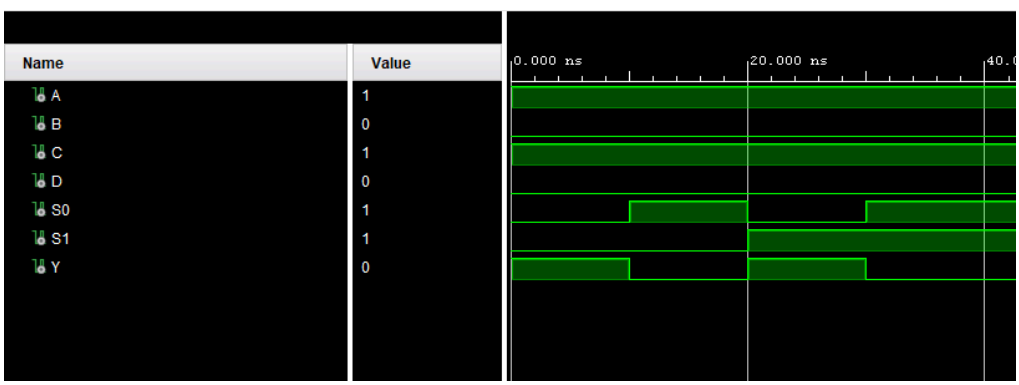

Figure 11: mux_1bit_4to1.vhd simulation waveform

In the mux_1bit_4to1_tb, we test every single combination of select bits S. By testing this way, we see that the output Y shows the values of each address (A[00], B[01], C [10], D[11]) as it is selected. Thus, the operation of the mux is verified. Recall that muxes in the design are used primarily for control logic, and not for selecting blocks and bytes of the cache to be accessed.

Figure 12: mux_8bit_2to1.vhd simulation waveform

In the mux_8bit_2to1 test, we test a mux with an 8-bit input/output data bus. There are 2 inputs to select from: **A0** and **A1**. The select bit is **S**. As can be seen from the waveform above, when switching S from 0 to 1, the output of the mux also changes from A0 to A1. Thus the function of the module is verified.

3 Layouts and Schematics

Below is a compilation of all the cell schematics and layouts with accompanying LVS verification.

Layouts are in alphabetical order. Area computations are shown in the chip.vhd section. Please refer there.

and2.vhd

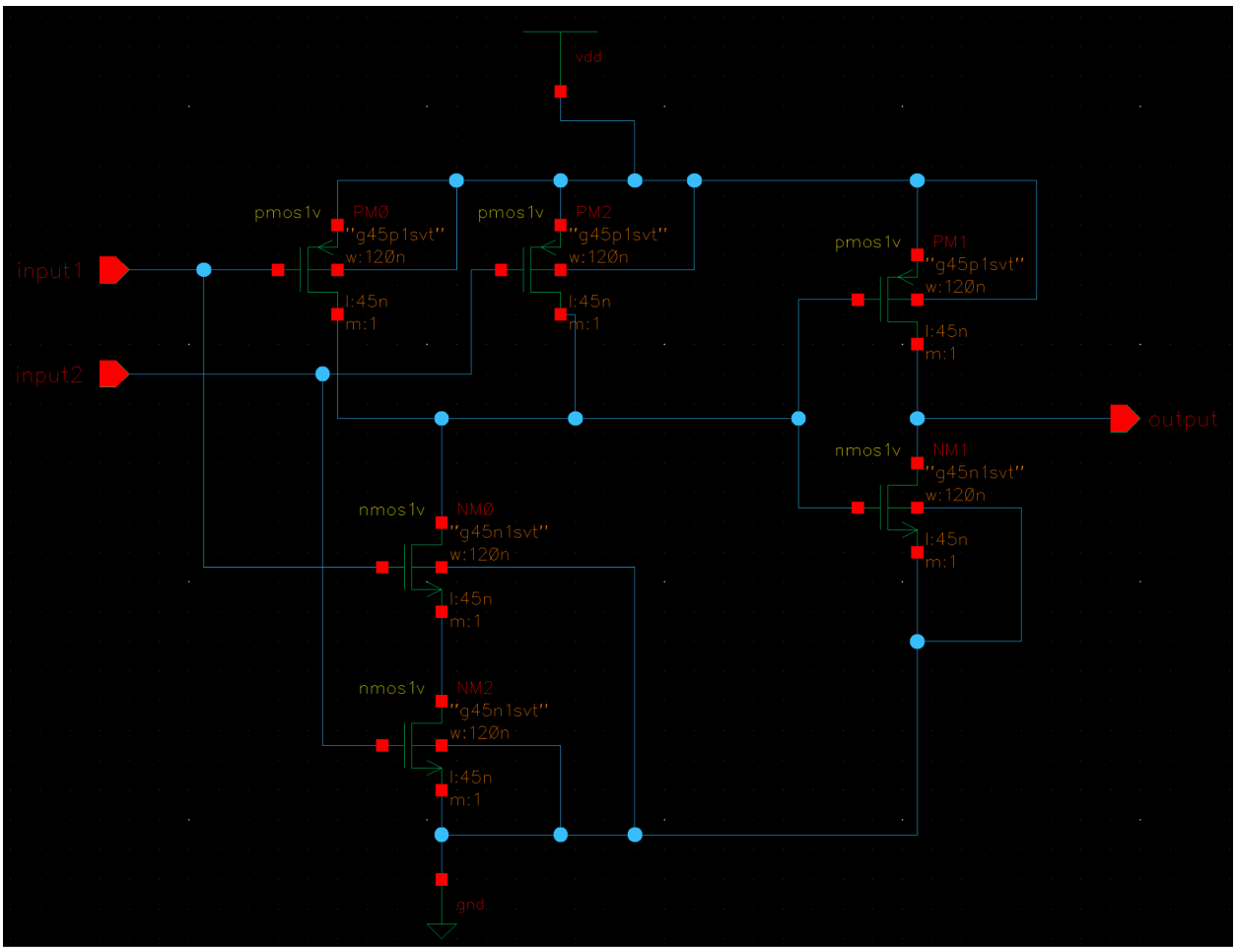

Figure 13: and2.vhd schematic view

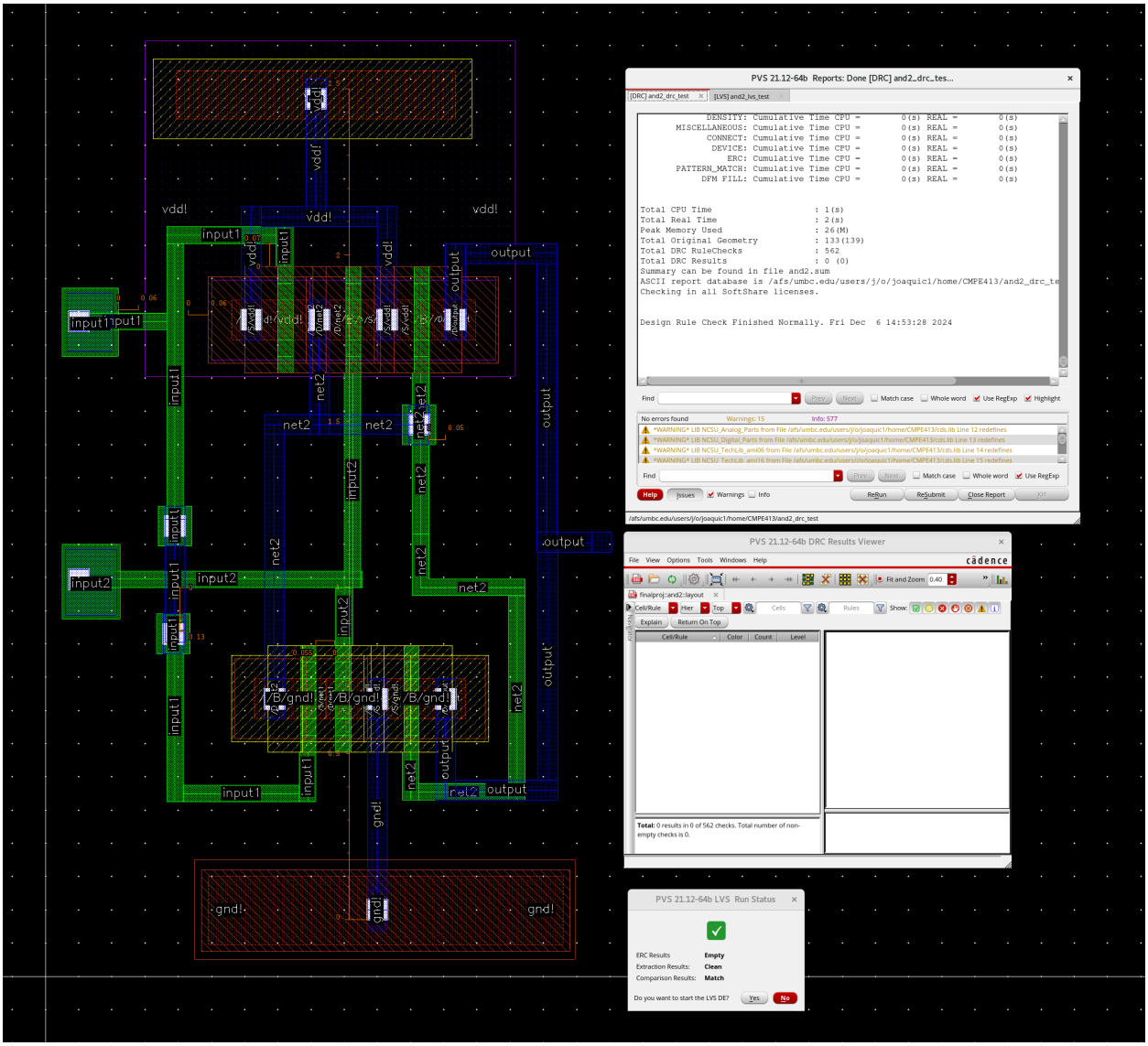

Figure 14: and2.vhd layout view (DRC + LVS verified)

or3.vhd

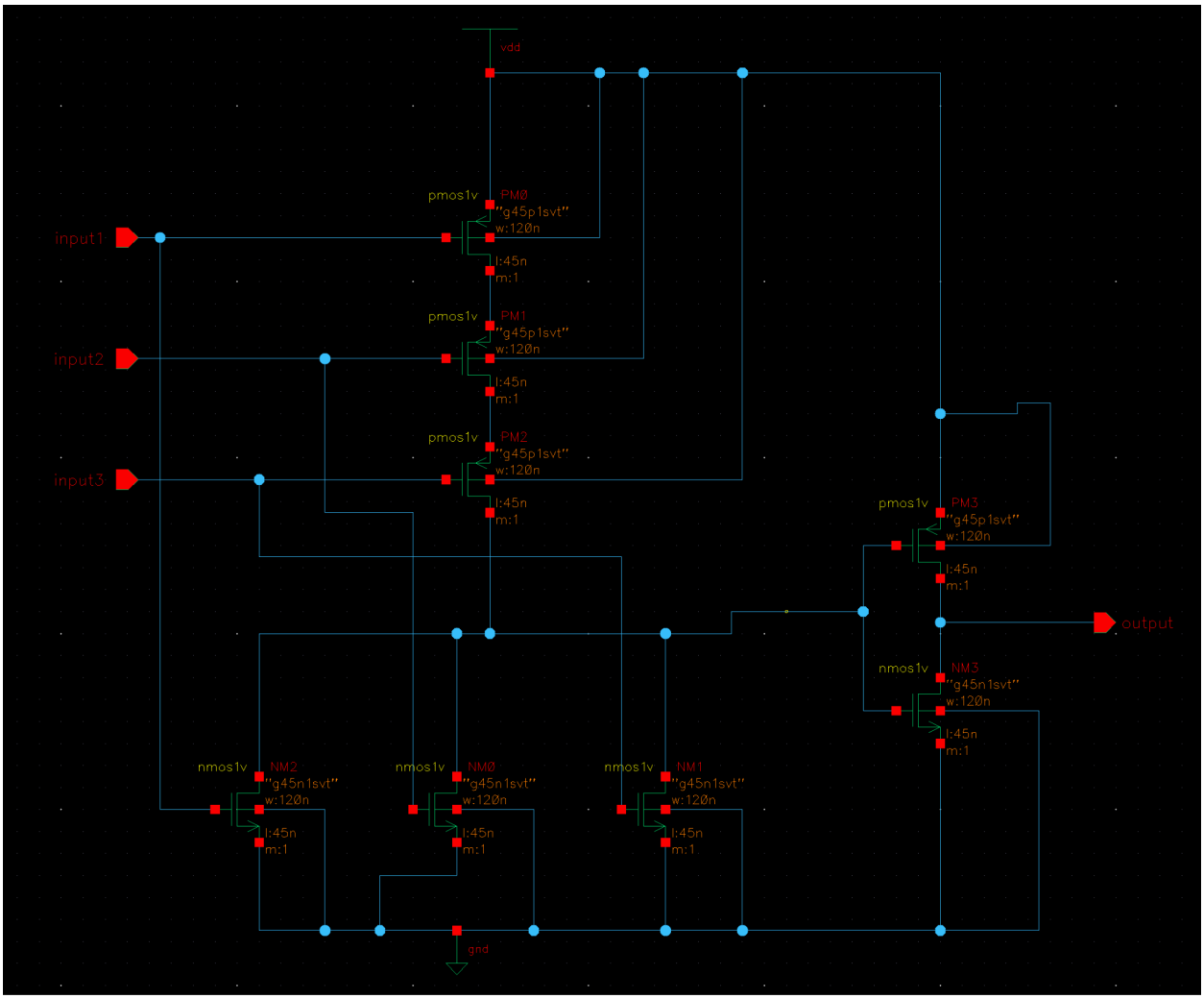

Figure 15: or3.vhd schematic view

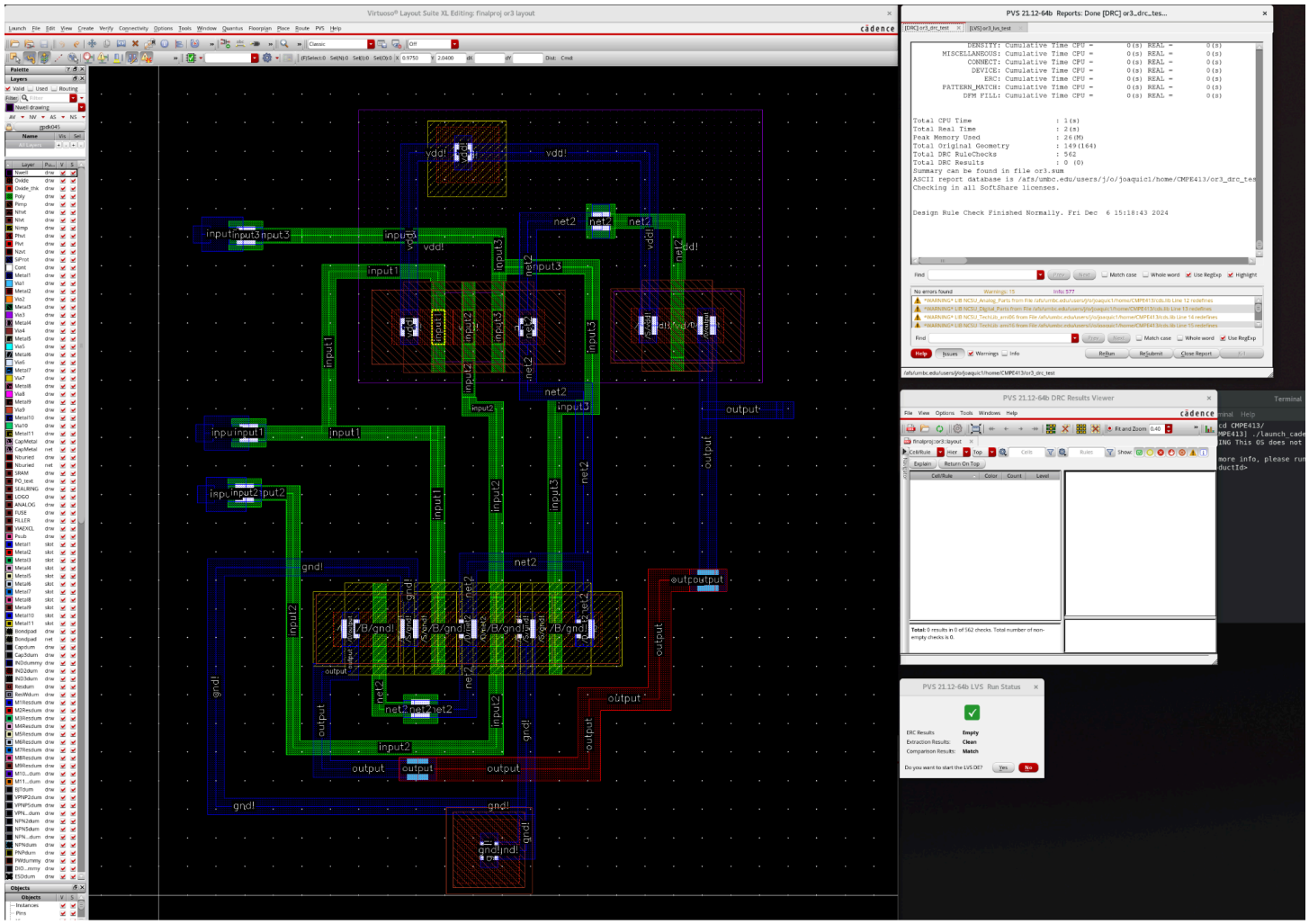

Figure 16: or3.vhd layout view (DRC + LVS verified)

tx.vhd

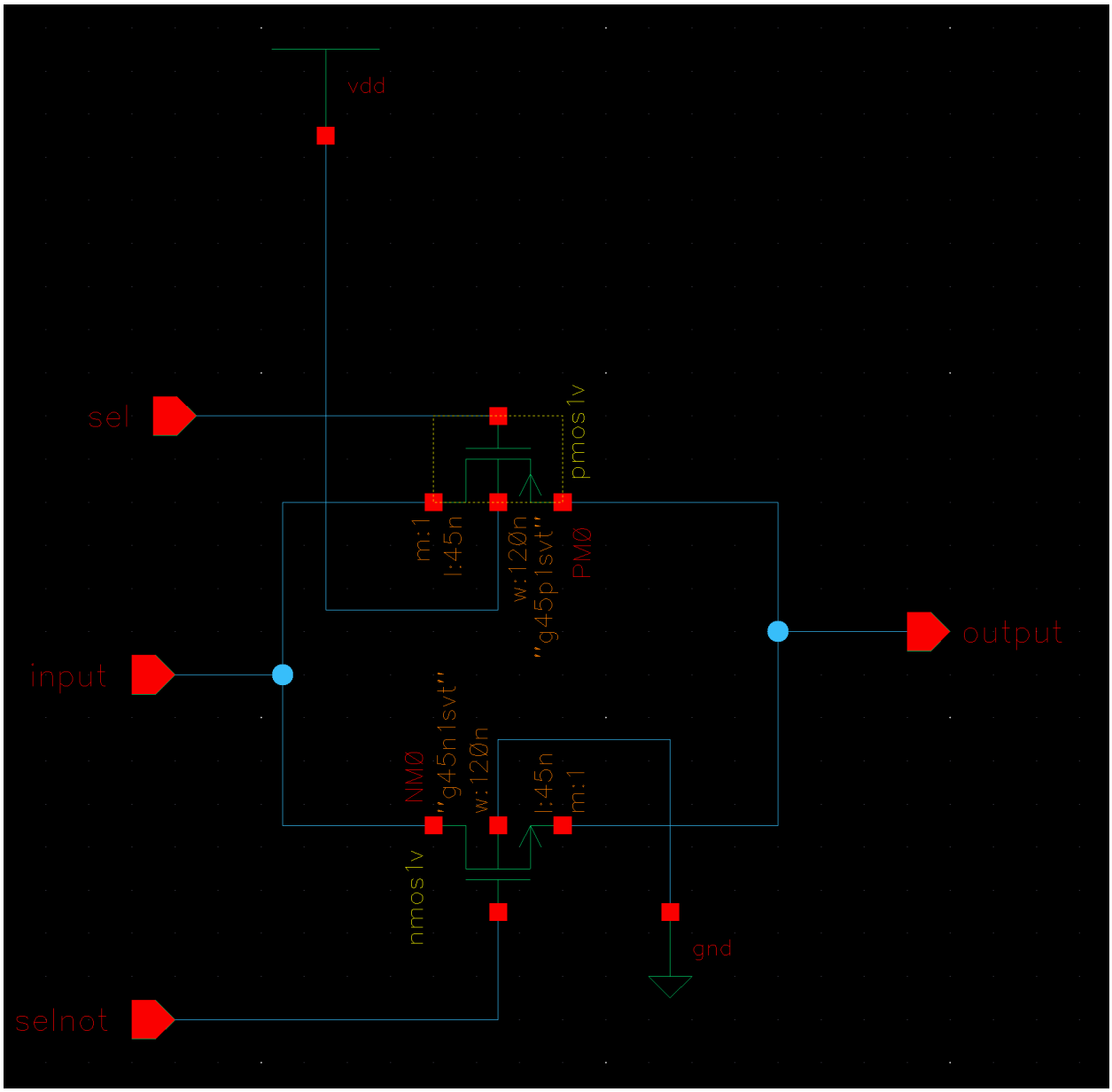

Figure 17: tx.vhd schematic view

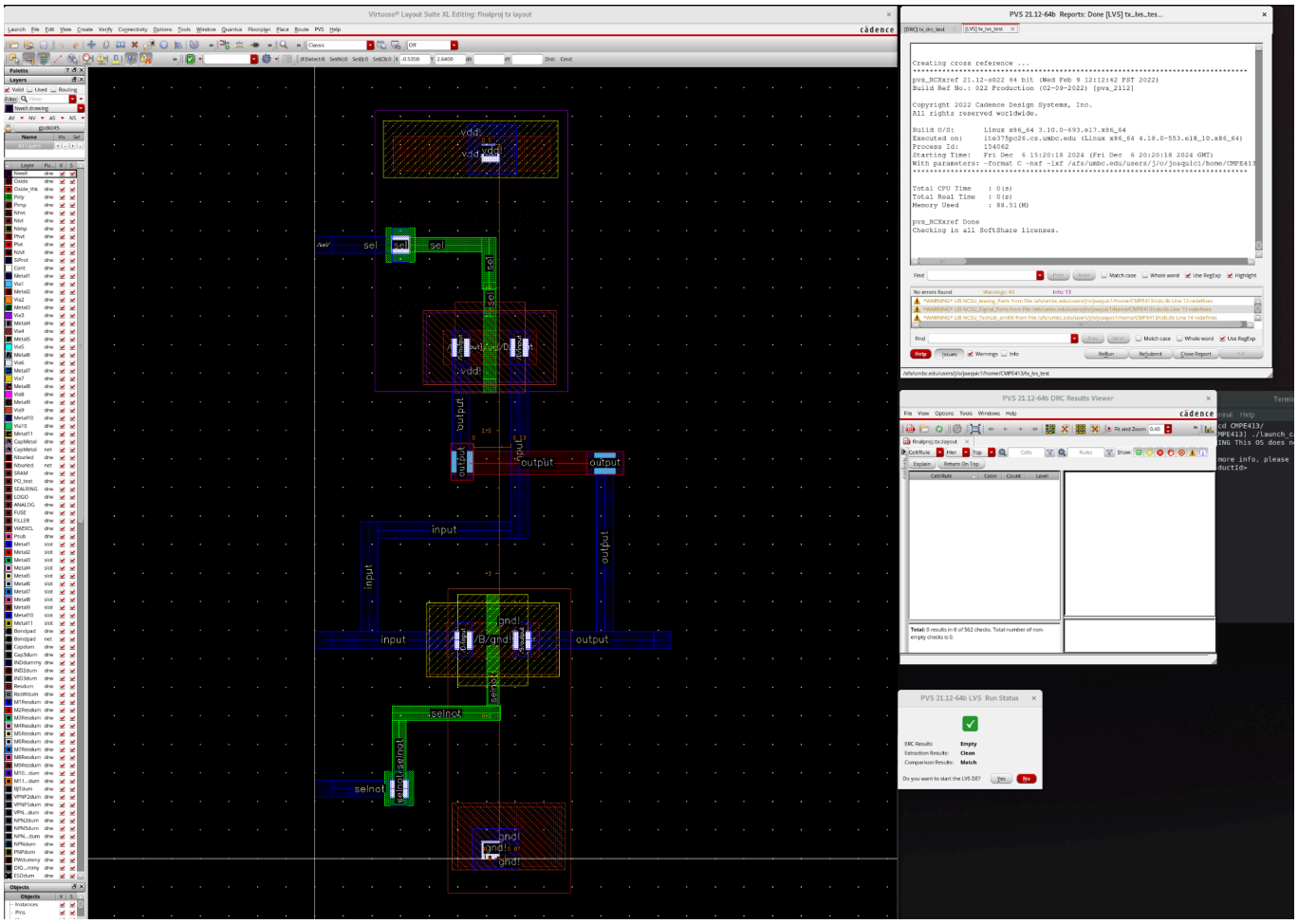

Figure 18: tx.vhd layout view (DRC + LVS verified)

xnor2.vhd

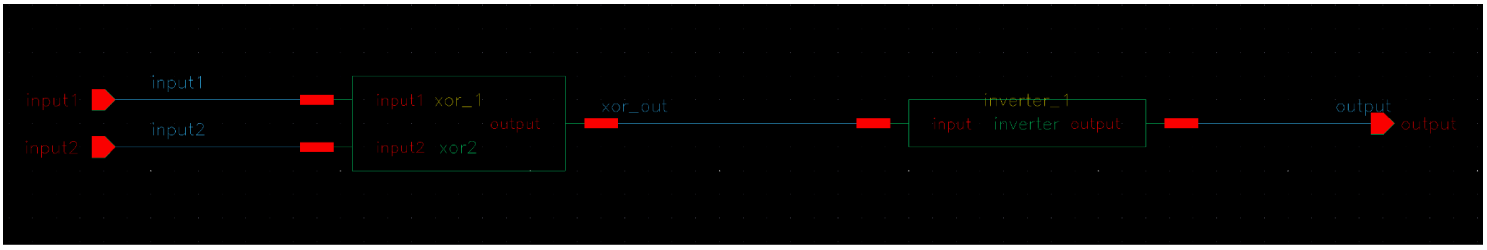

Figure 19: xnor2.vhd schematic view

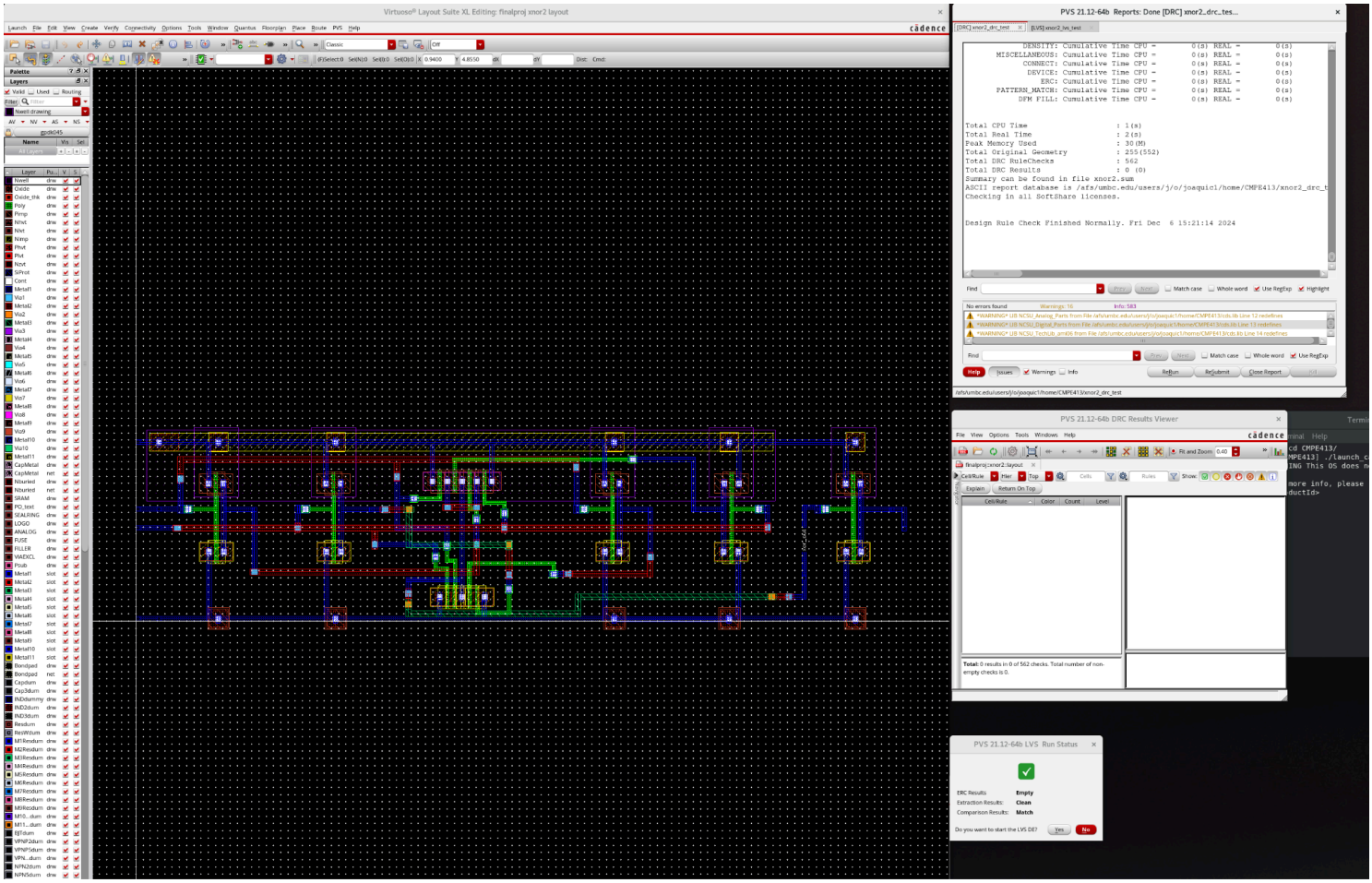

Figure 20: xnor2.vhd layout view (DRC + LVS verified)

xor2.vhd

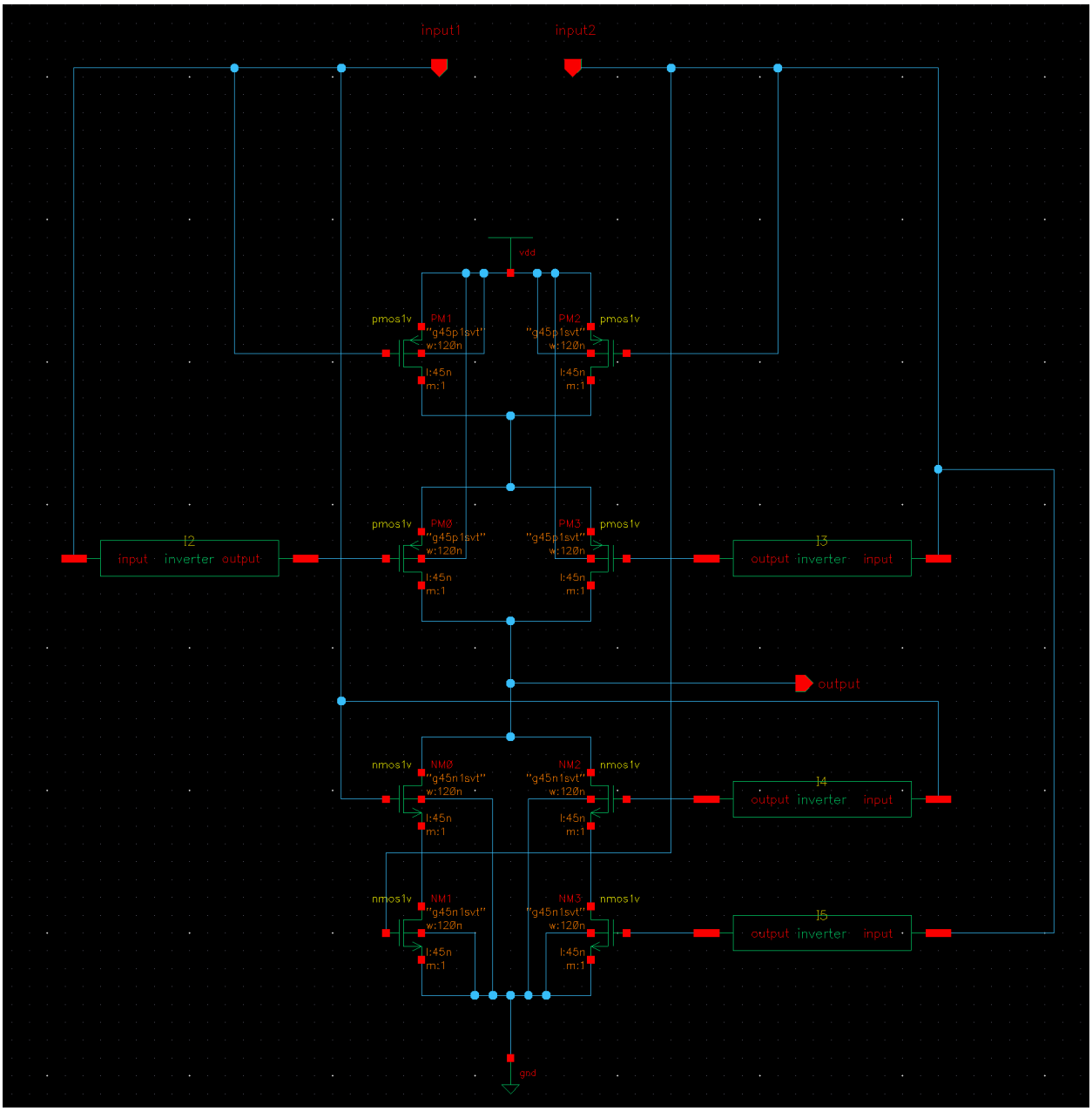

Figure 21: xor2.vhd schematic view

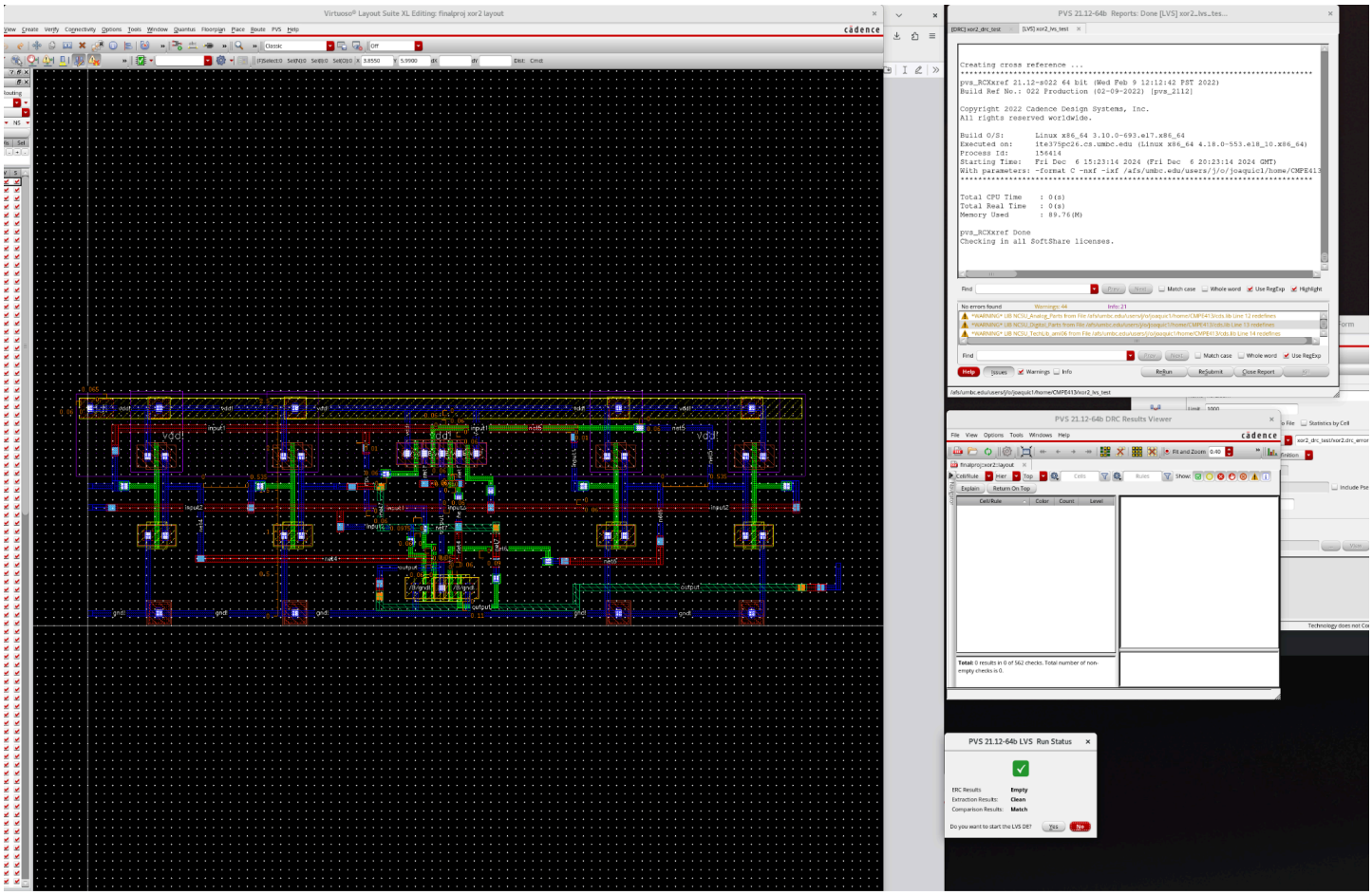

Figure 22: xor2.vhd layout view (DRC + LVS verified)

and10.vhd

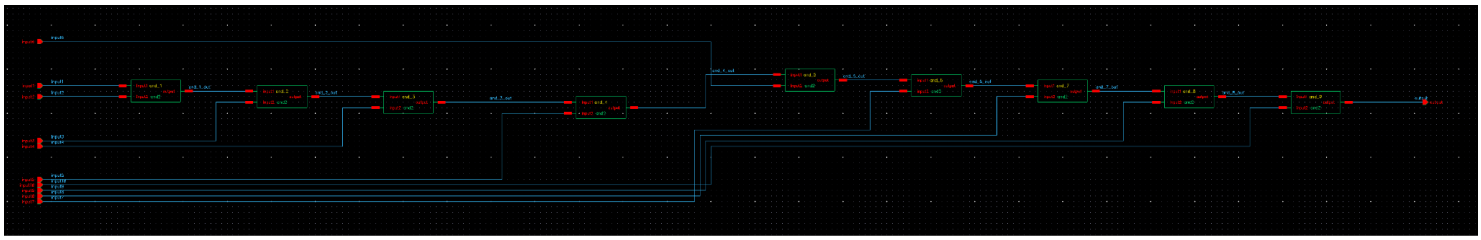

Figure 23: and10.vhd schematic view

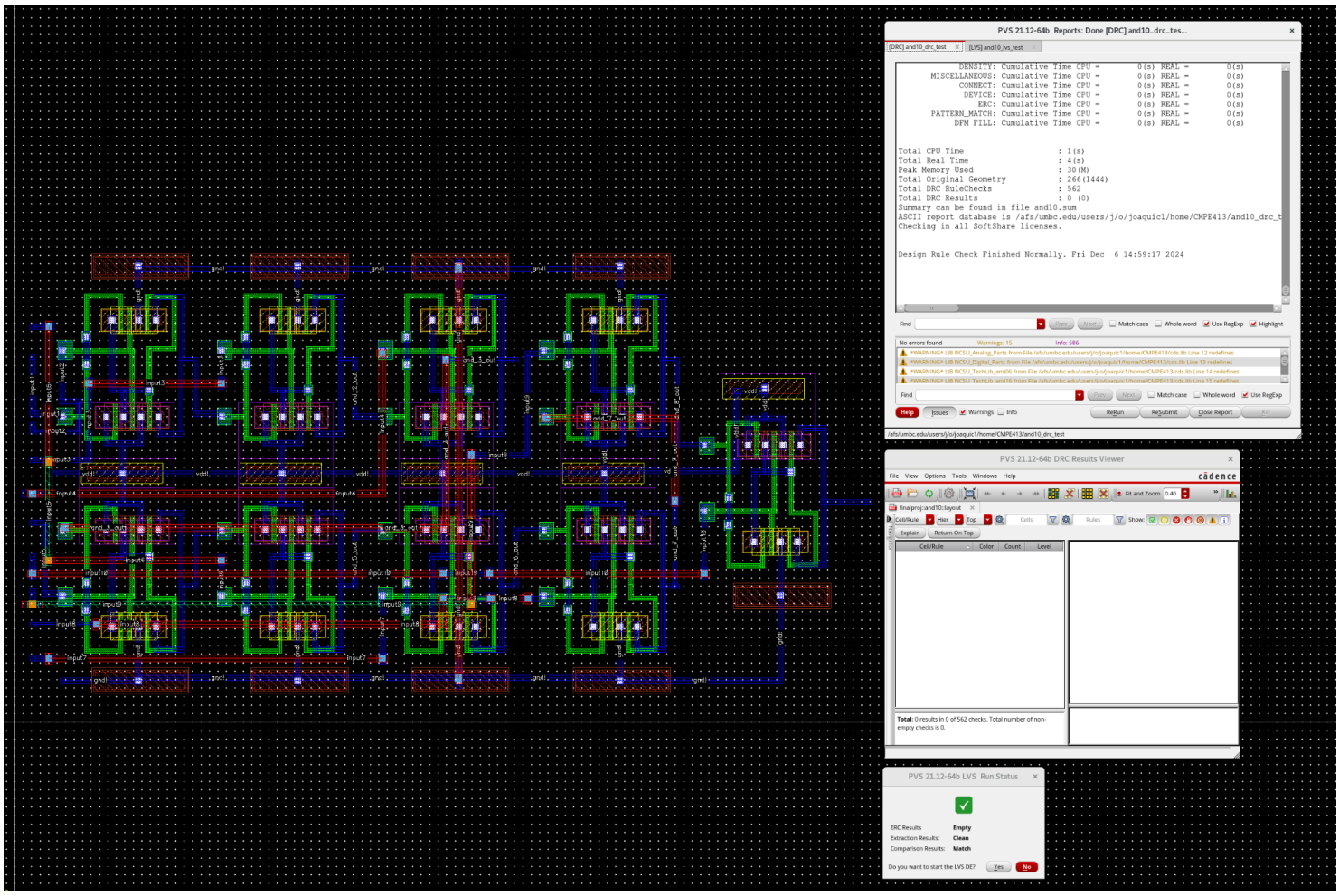

Figure 24: and10.vhd layout view (DRC + LVS verified)

cache_top.vhd

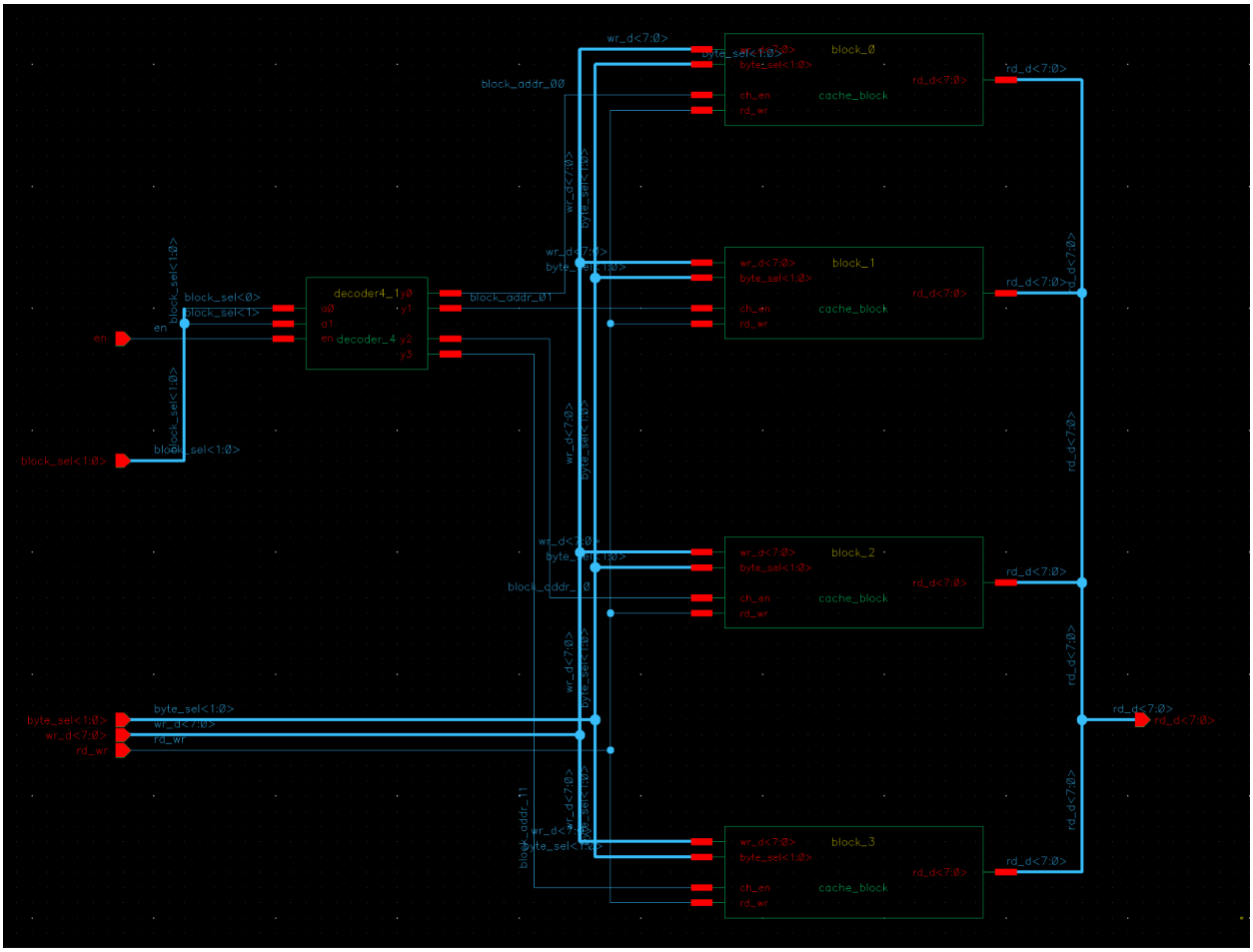

Figure 25: cache_top.vhd schematic view

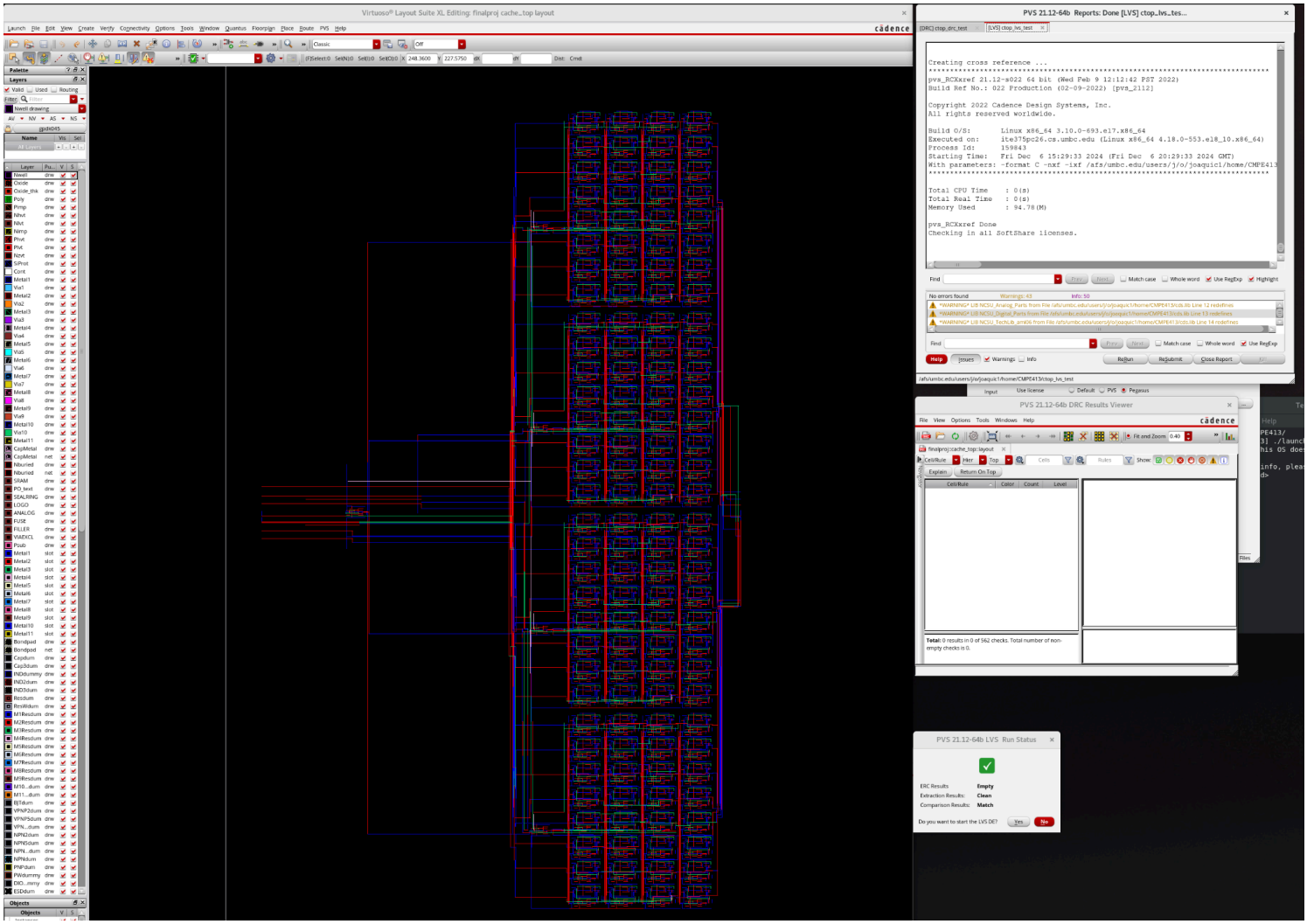

Figure 26: cache_top.vhd layout view (DRC + LVS verified)

cache_block.vhd

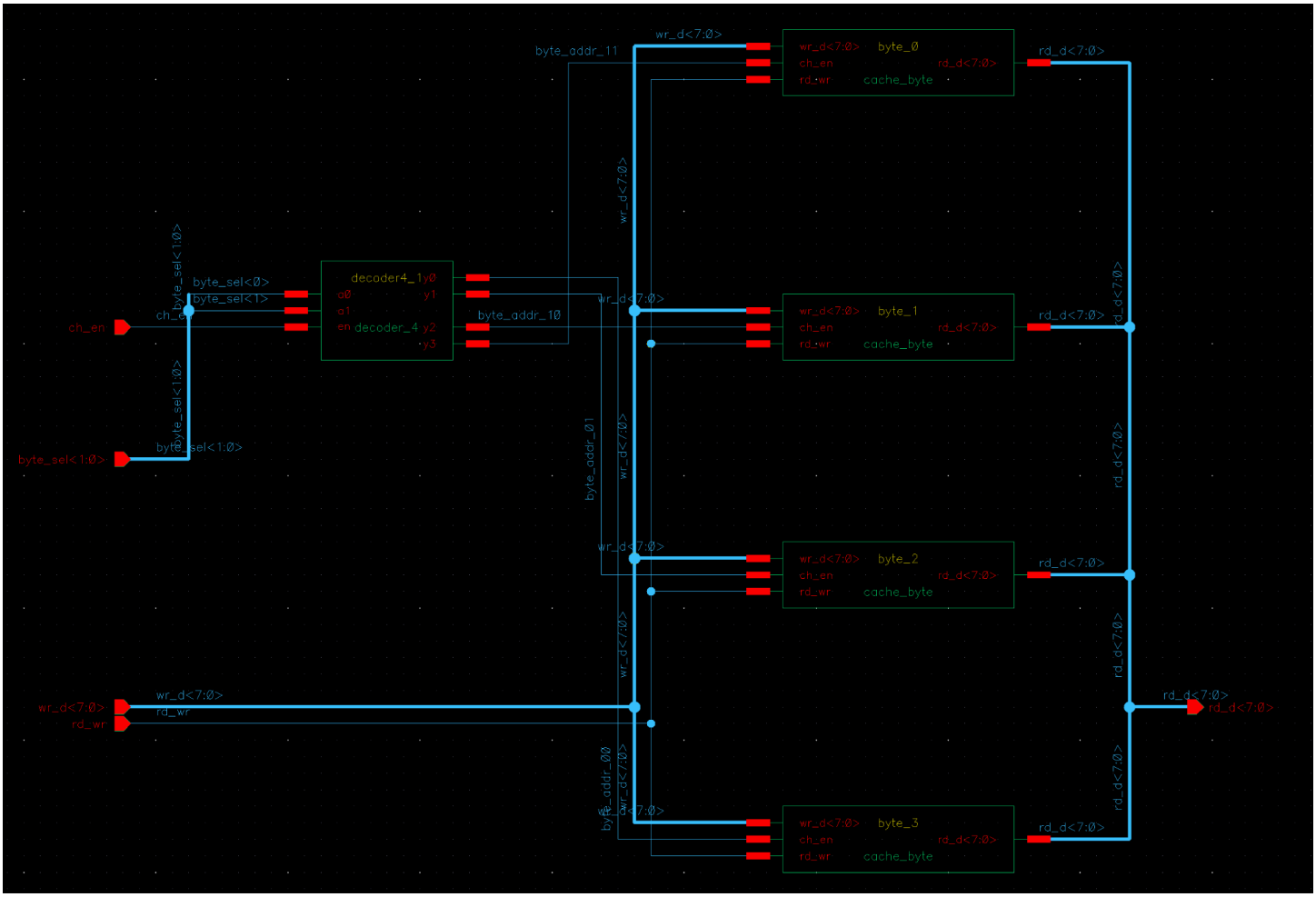

Figure 27: cache_block.vhd schematic view

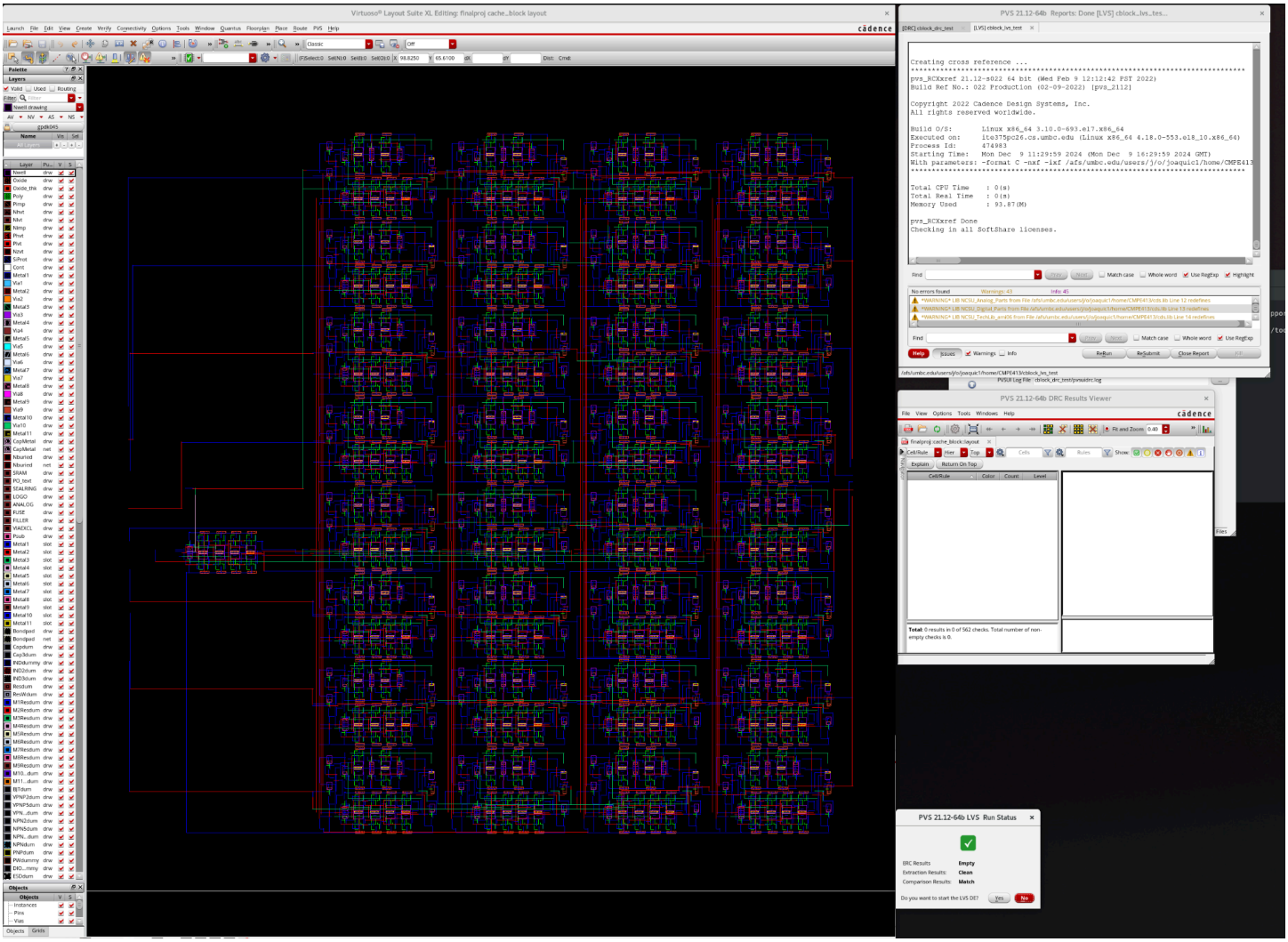

Figure 28: cache_block.vhd layout view (DRC + LVS verified)

cache_byte.vhd

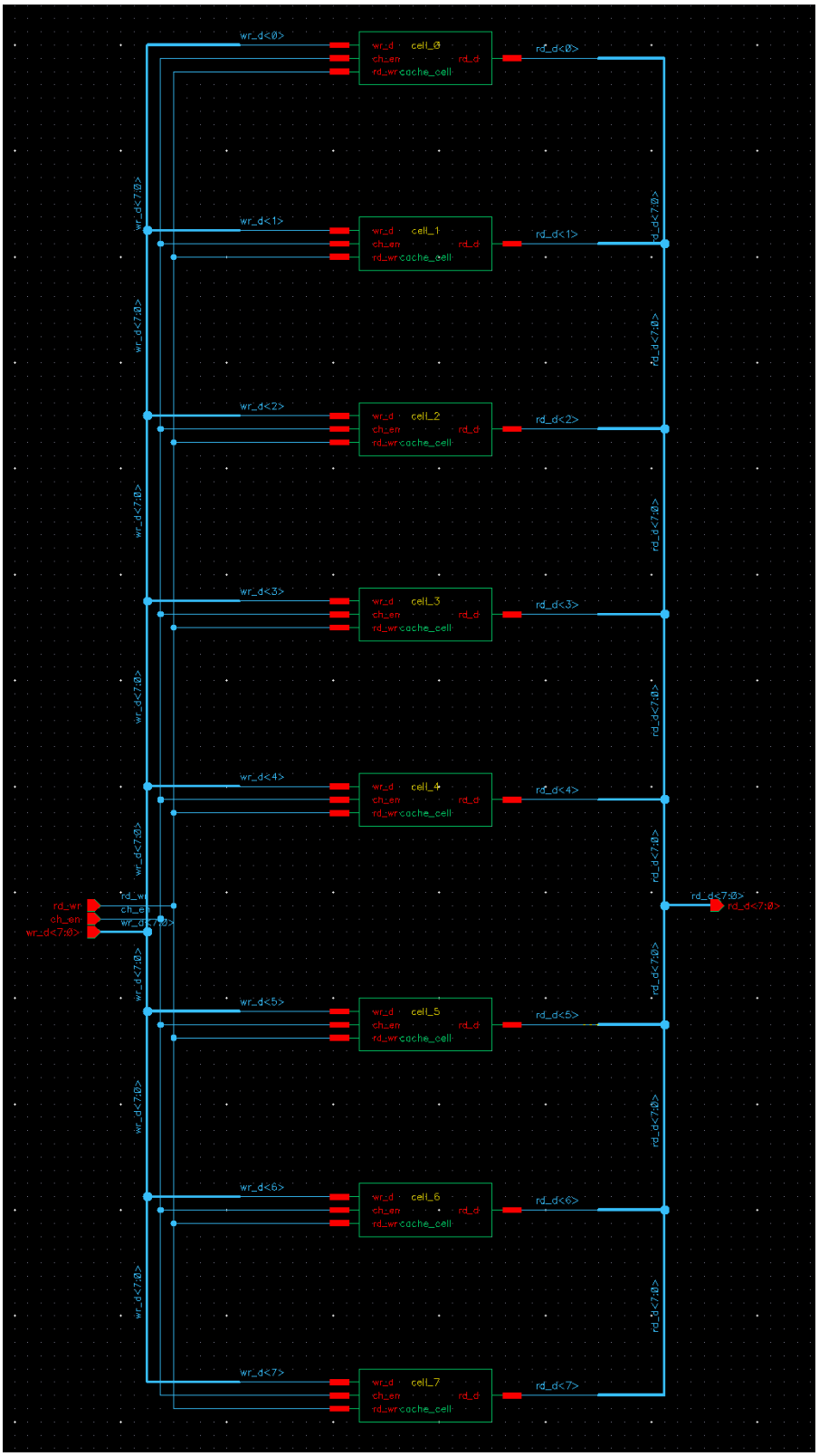

Figure 29: cache_byte.vhd schematic view

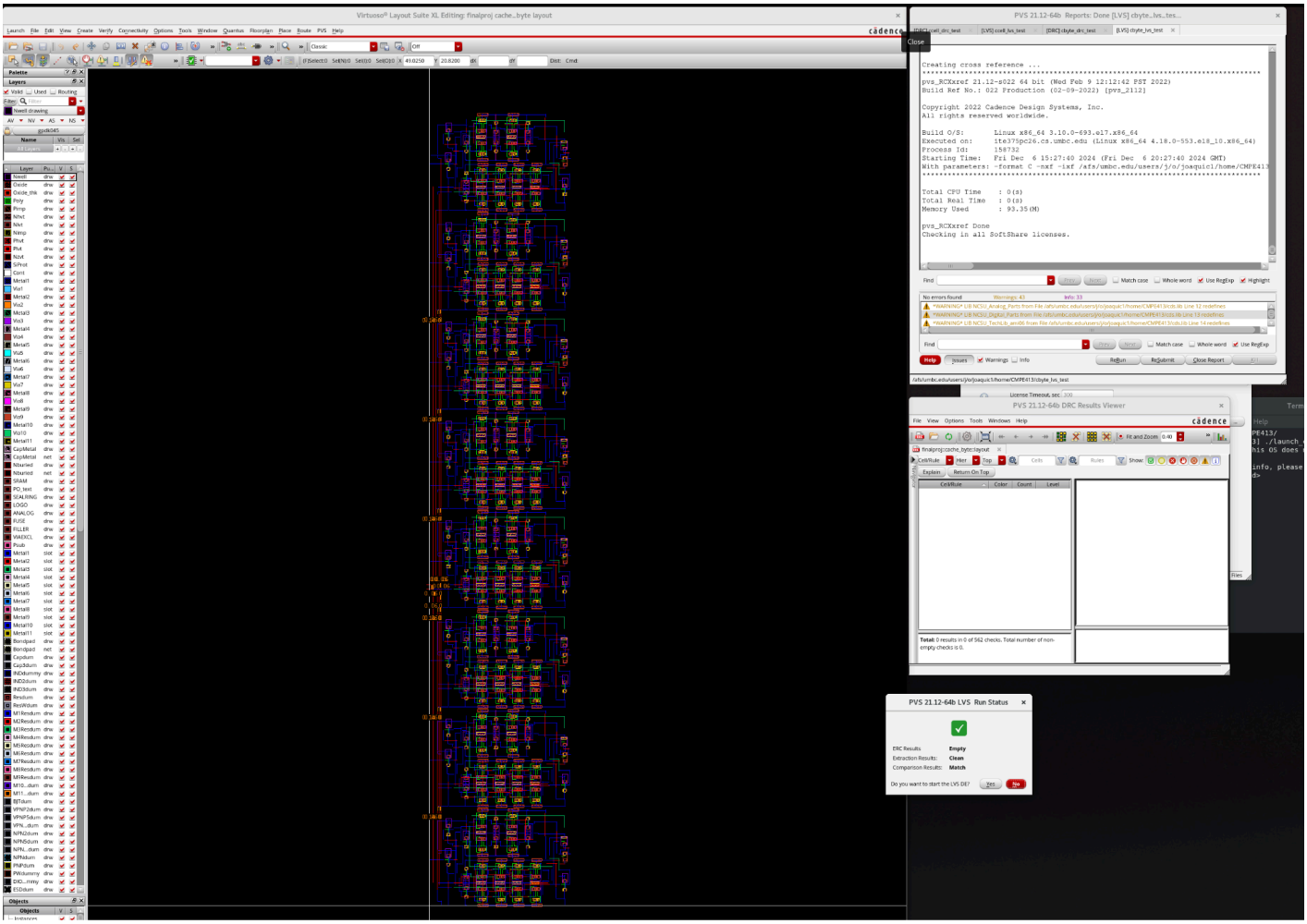

Figure 30: cache_byte.vhd layout view (DRC + LVS verified)

cache_cell.vhd

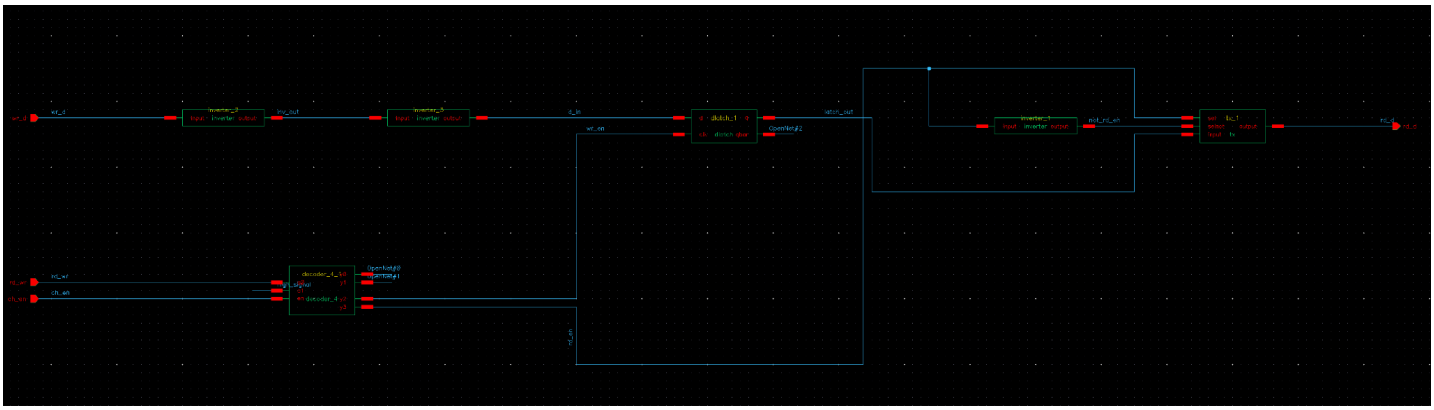

Figure 31: cache_cell.vhd schematic view

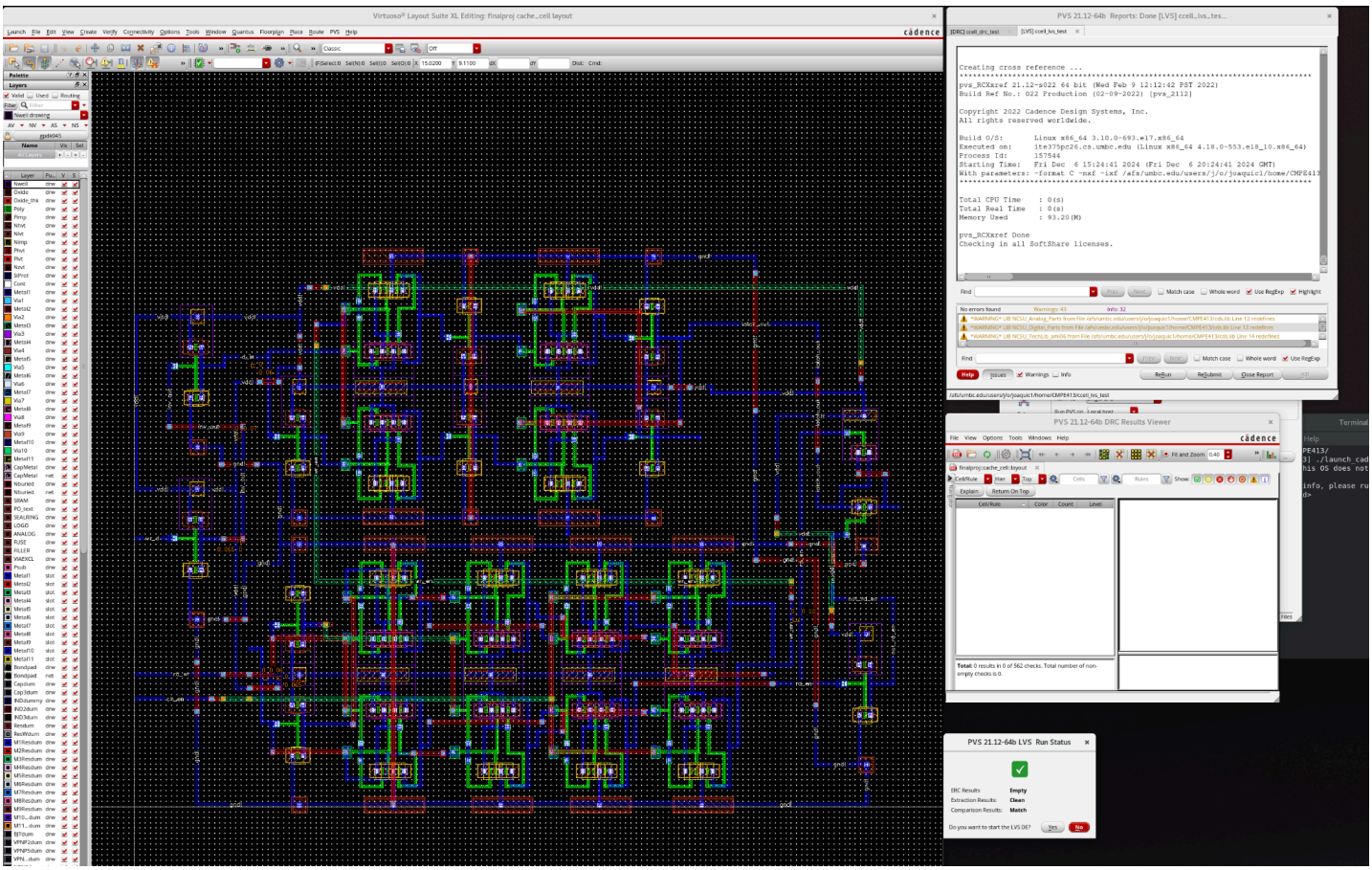

Figure 32: cache_cell.vhd layout view (DRC + LVS verified)

chip.vhd (AREA CALCULATION INCLUDED)

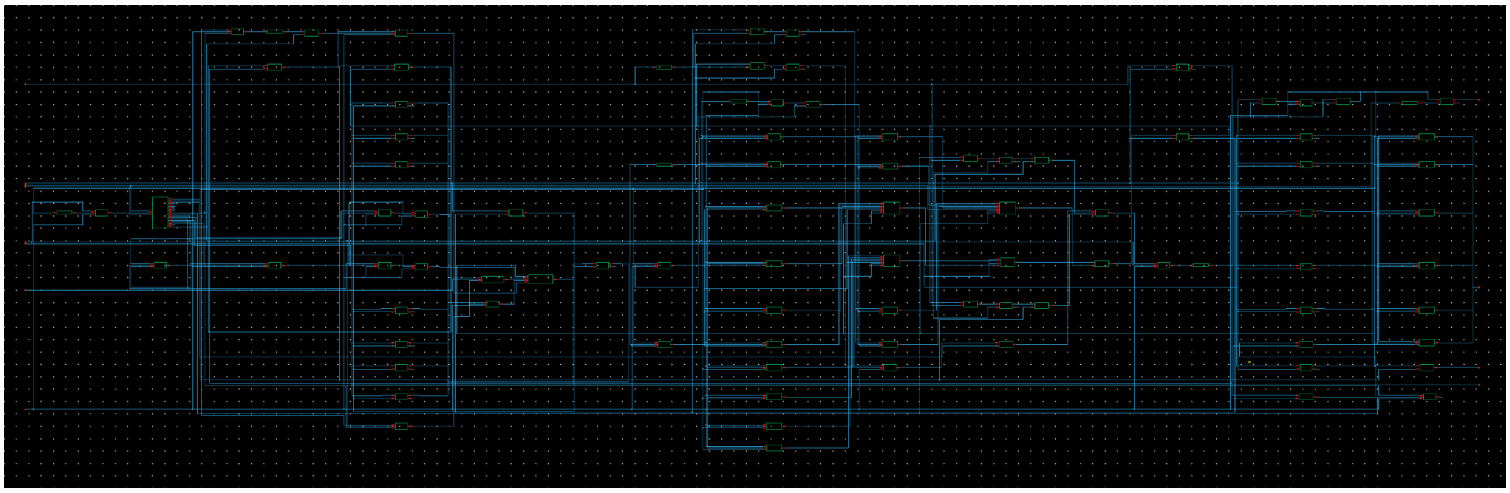

Figure 33: chip.vhd schematic view

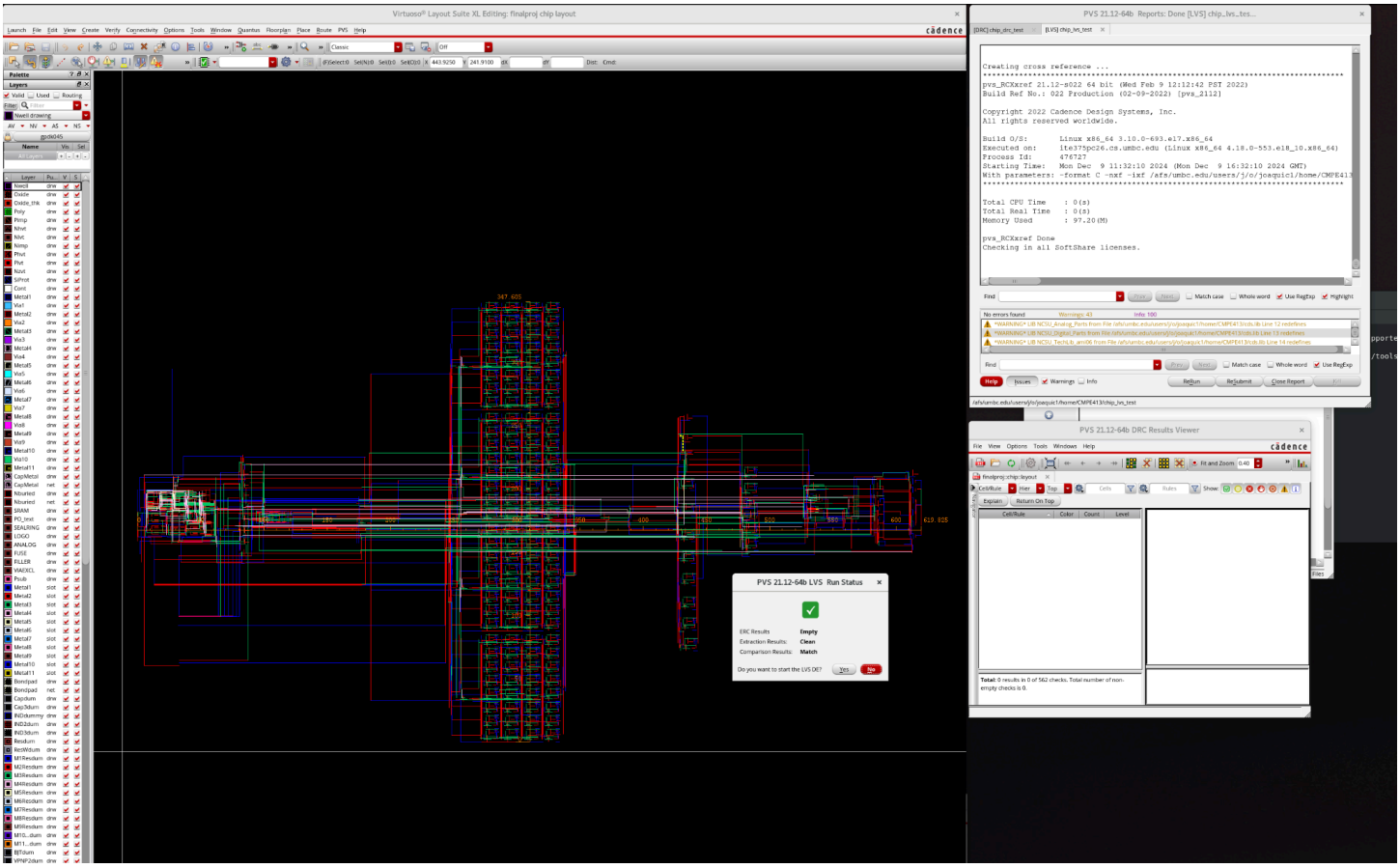

Figure 34: chip.vhd layout view (DRC + LVS verified)

Length: 619.825 um
Width: 347.605 um

Area of chip.vhd layout: (619.825um x 347.605um) = $215.454\ x\ 10^3\ um^2$

clk_counter.vhd

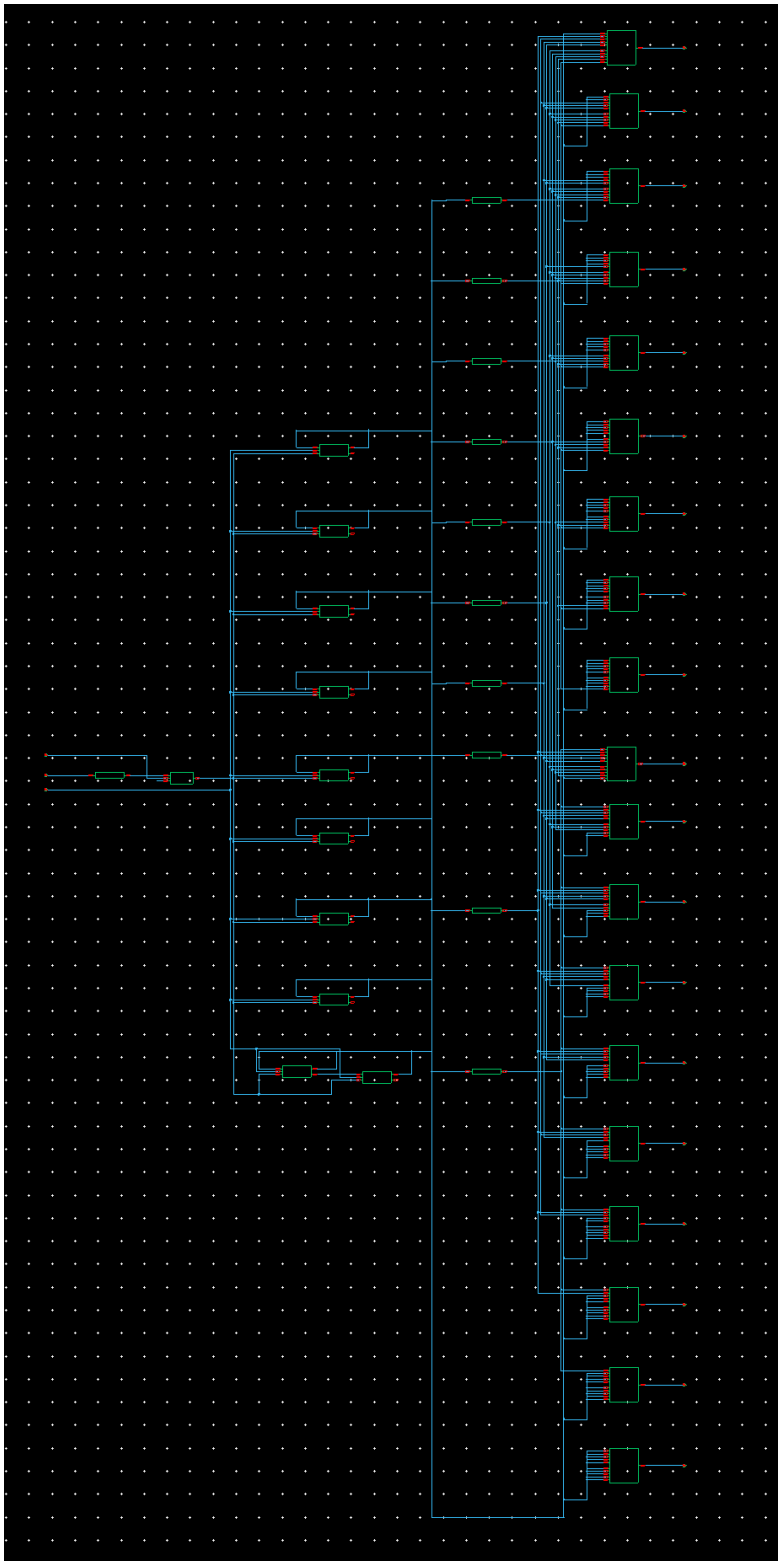

Figure 35: clk_counter.vhd schematic view

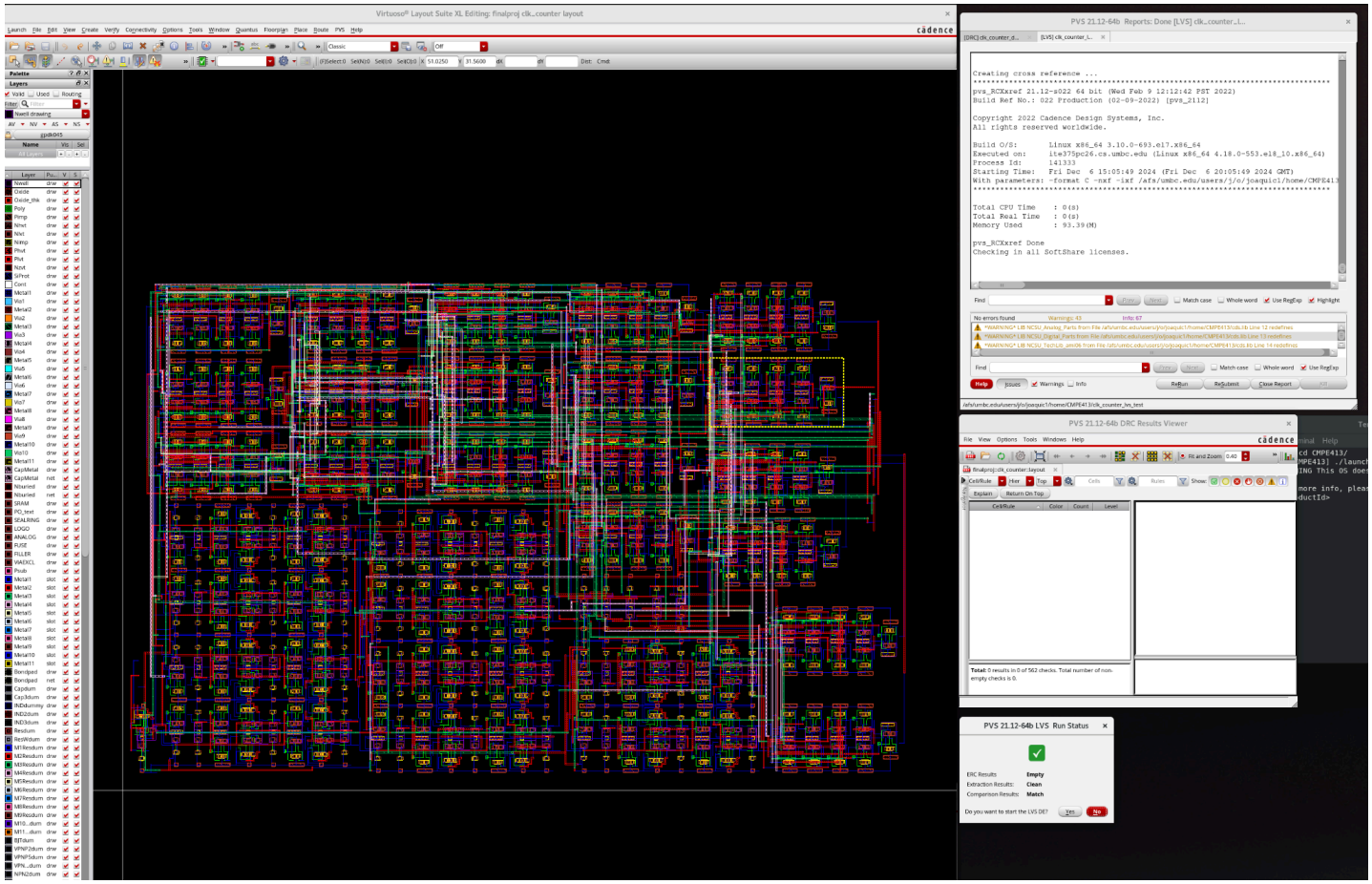

Figure 36: clk_counter.vhd layout view (DRC + LVS verified)

comparator2.vhd

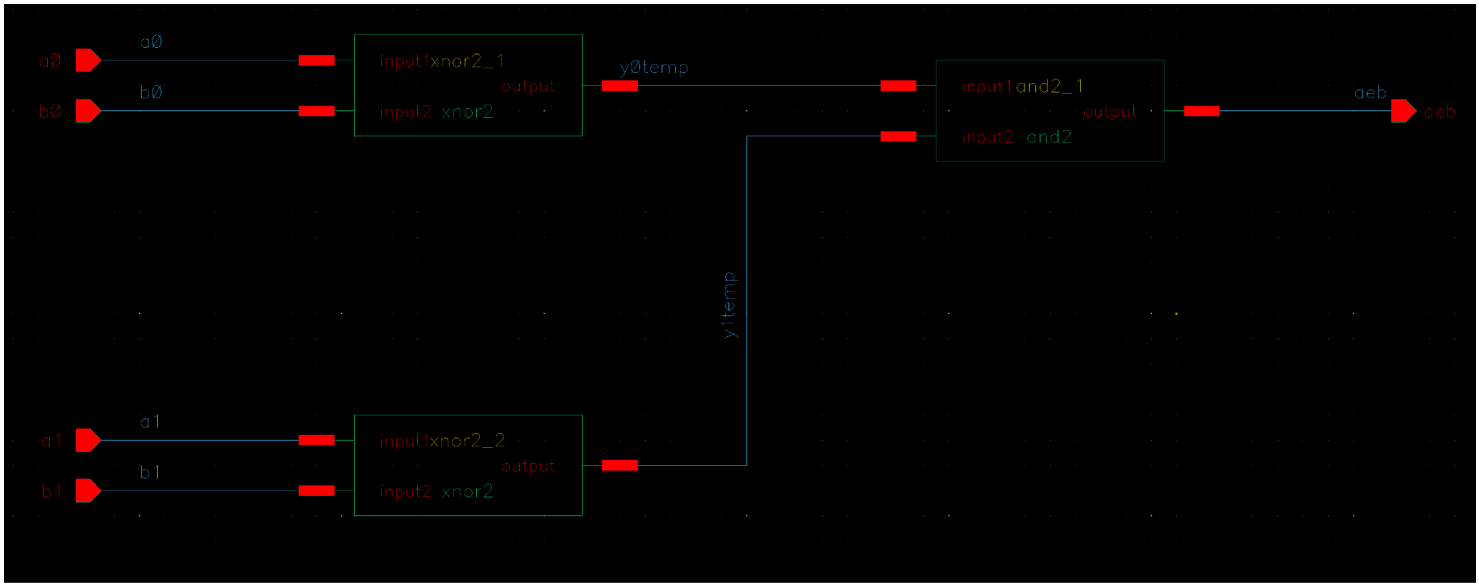

Figure 37: comparator2.vhd schematic view

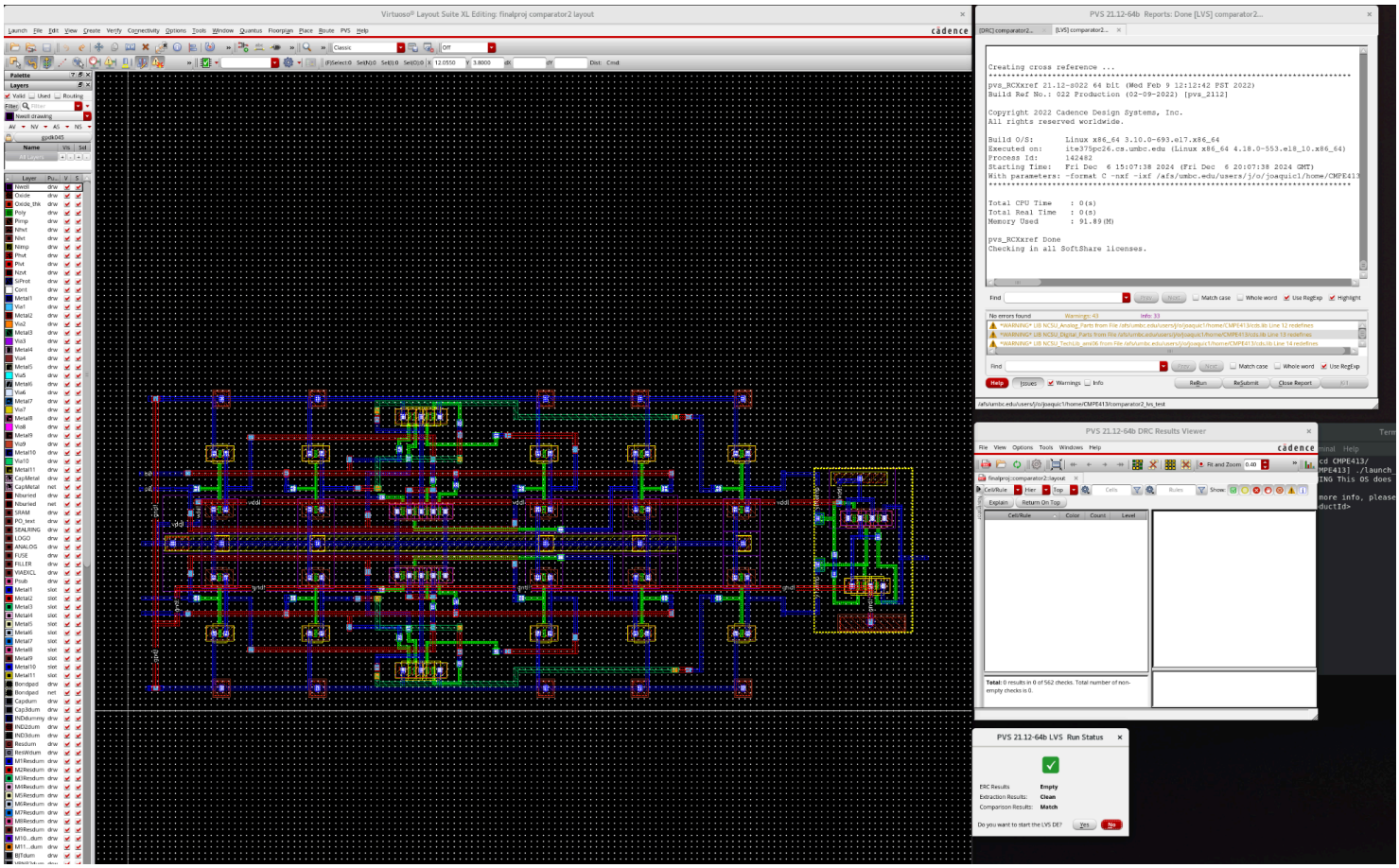

Figure 38: comparator2.vhd layout view (DRC + LVS verified)

decoder_4.vhd

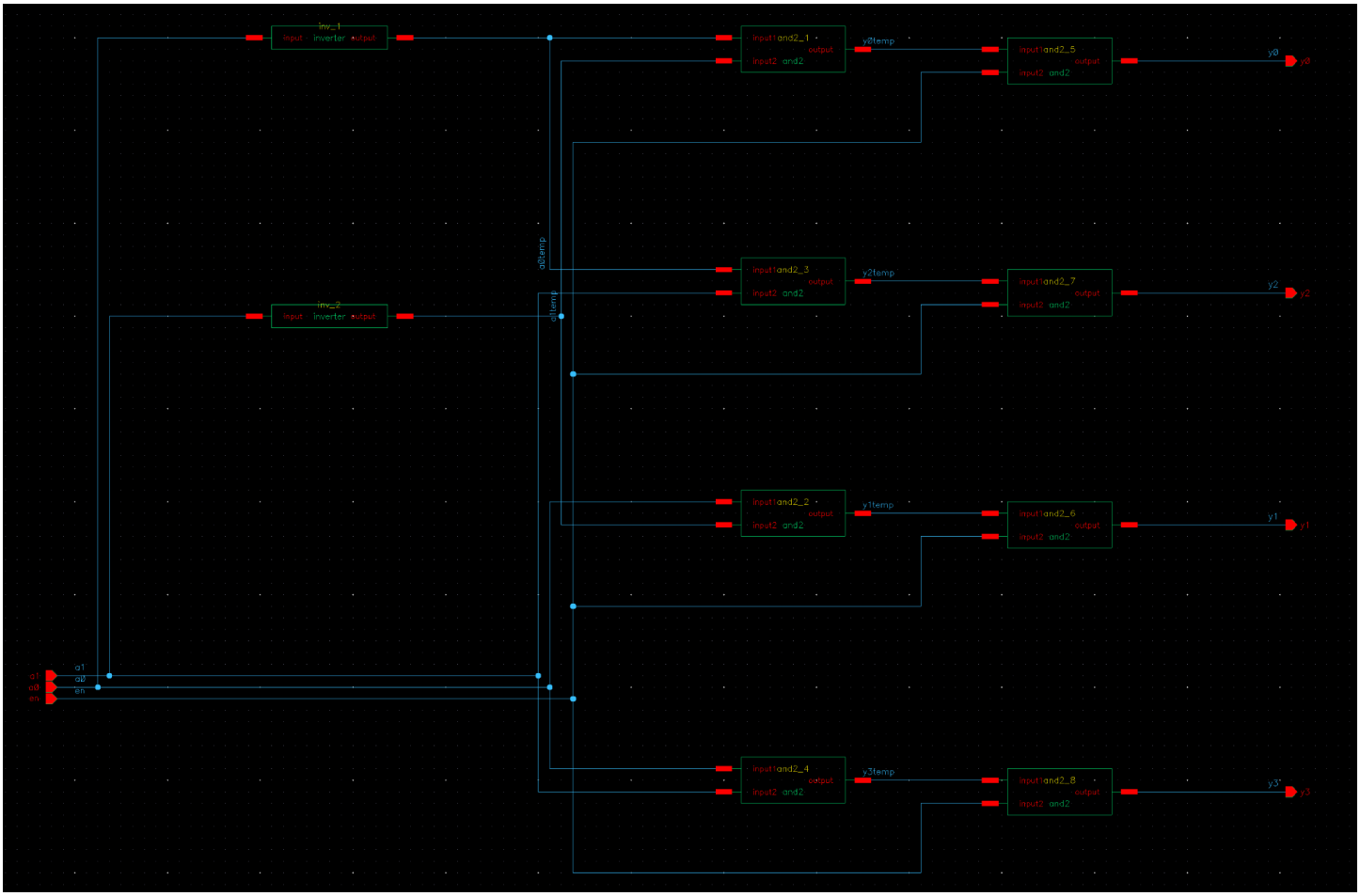

Figure 39: decoder_4.vhd schematic view

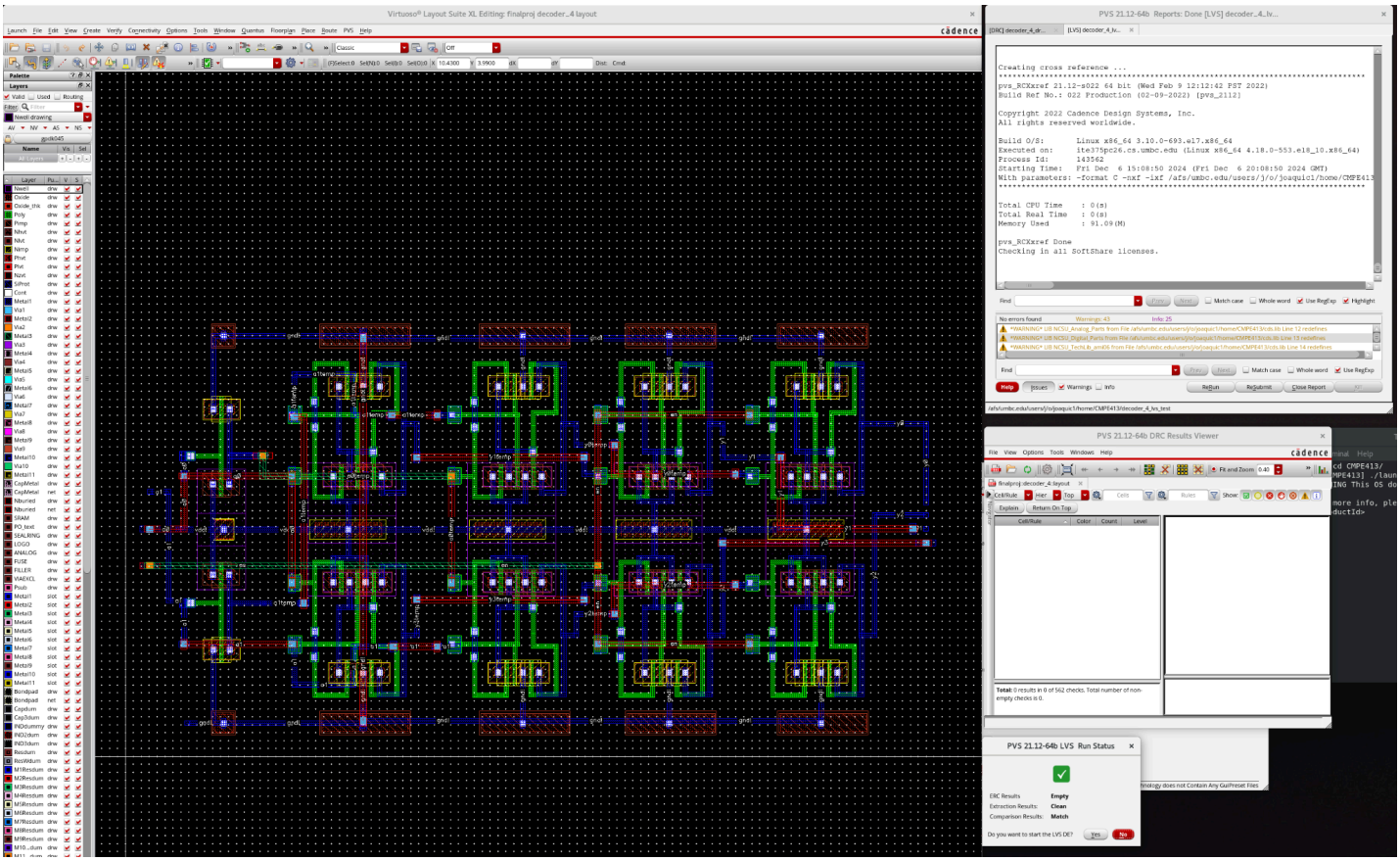

Figure 40: decoder_4.vhd layout view (DRC + LVS verified)

dff_reset.vhd

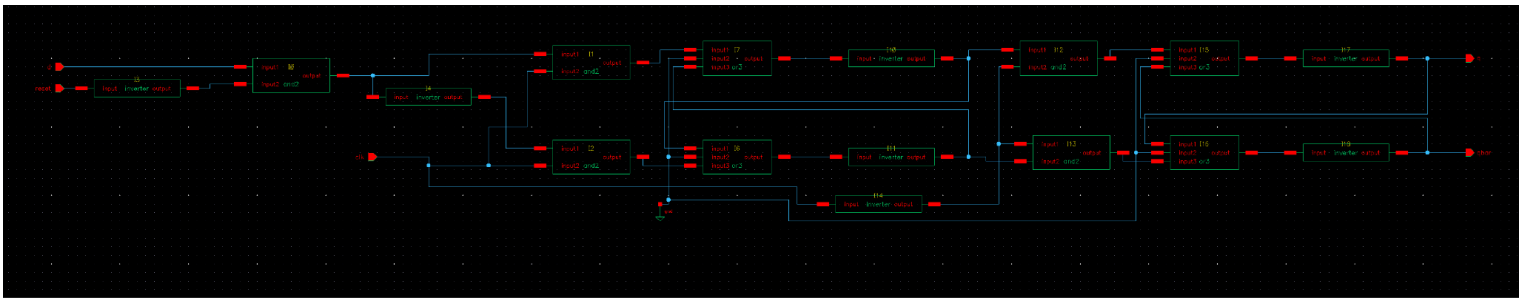

Figure 41: dff_reset.vhd schematic view

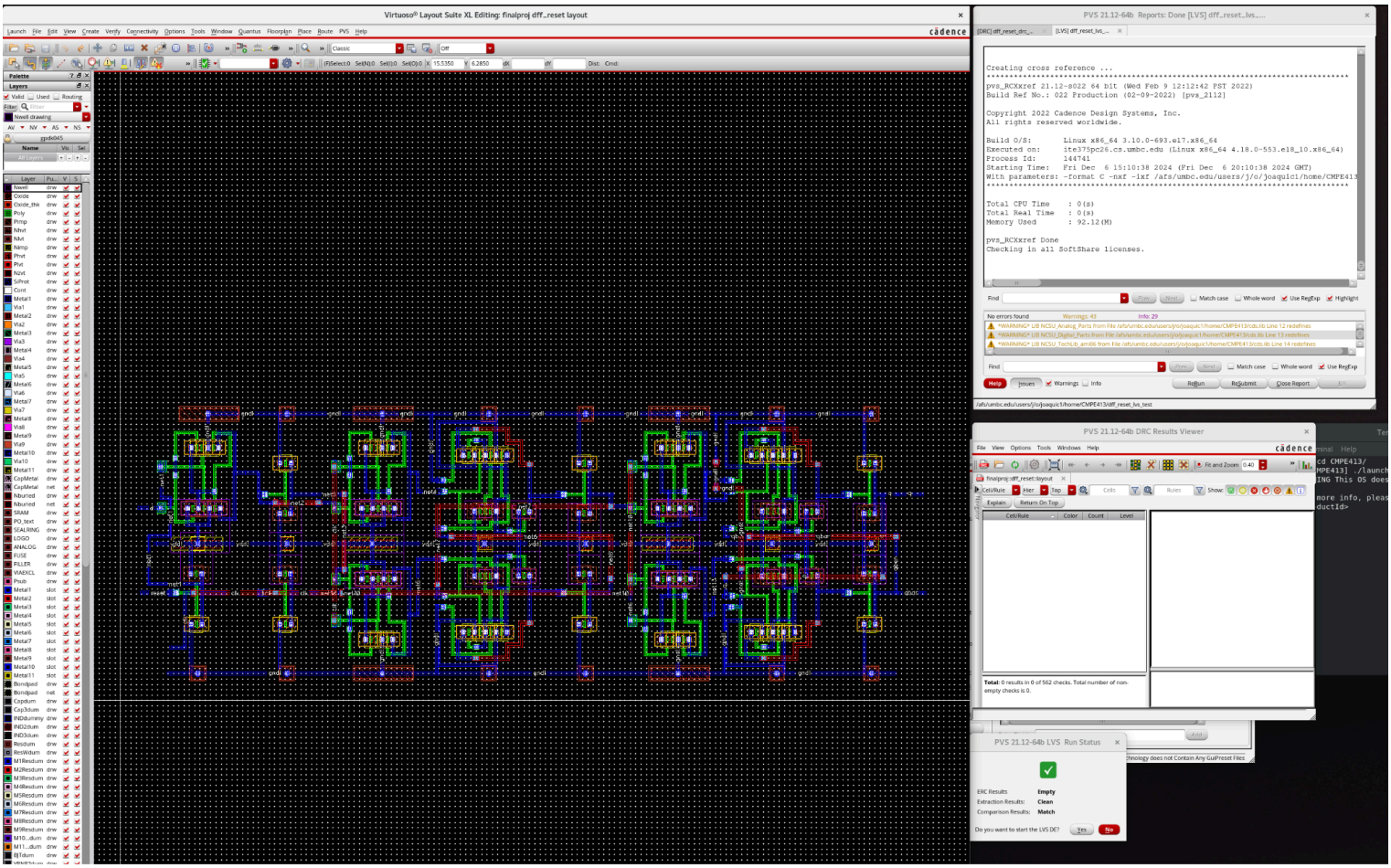

Figure 42: dff_reset.vhd layout view (DRC + LVS verified)

dlatch.vhd

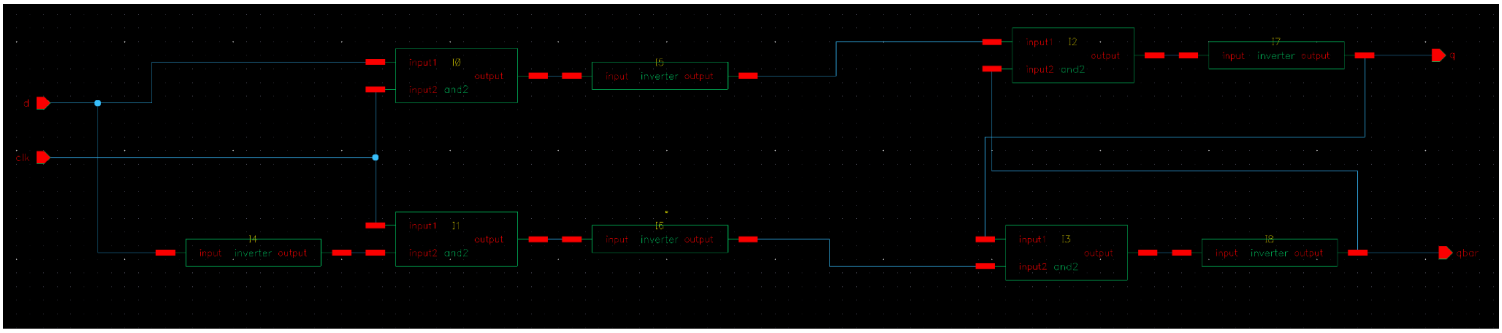

Figure 43: dlatch.vhd schematic view

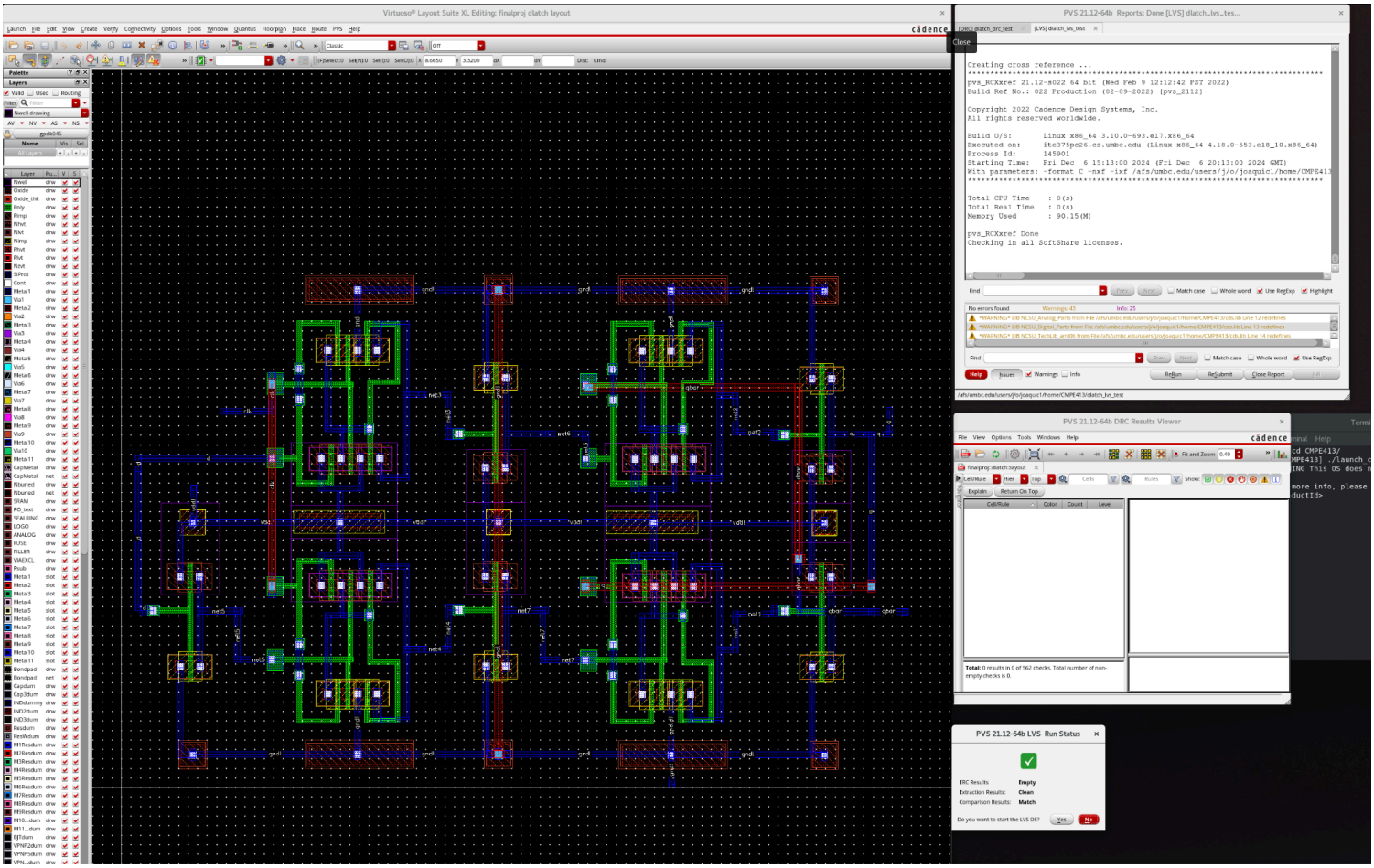

Figure 44: dlatch.vhd layout view (DRC + LVS verified)

inverter.vhd

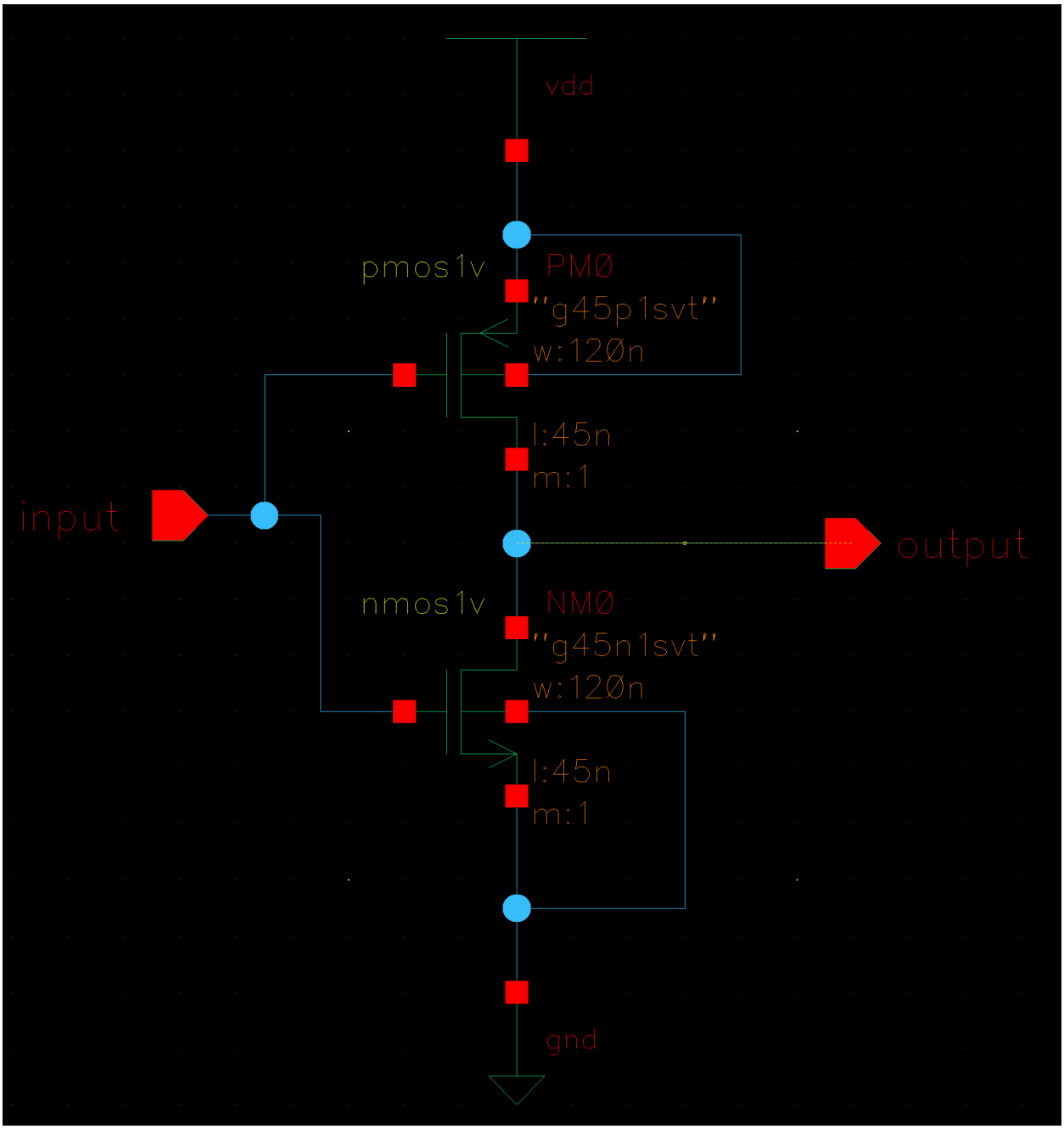

Figure 45: inverter.vhd schematic view

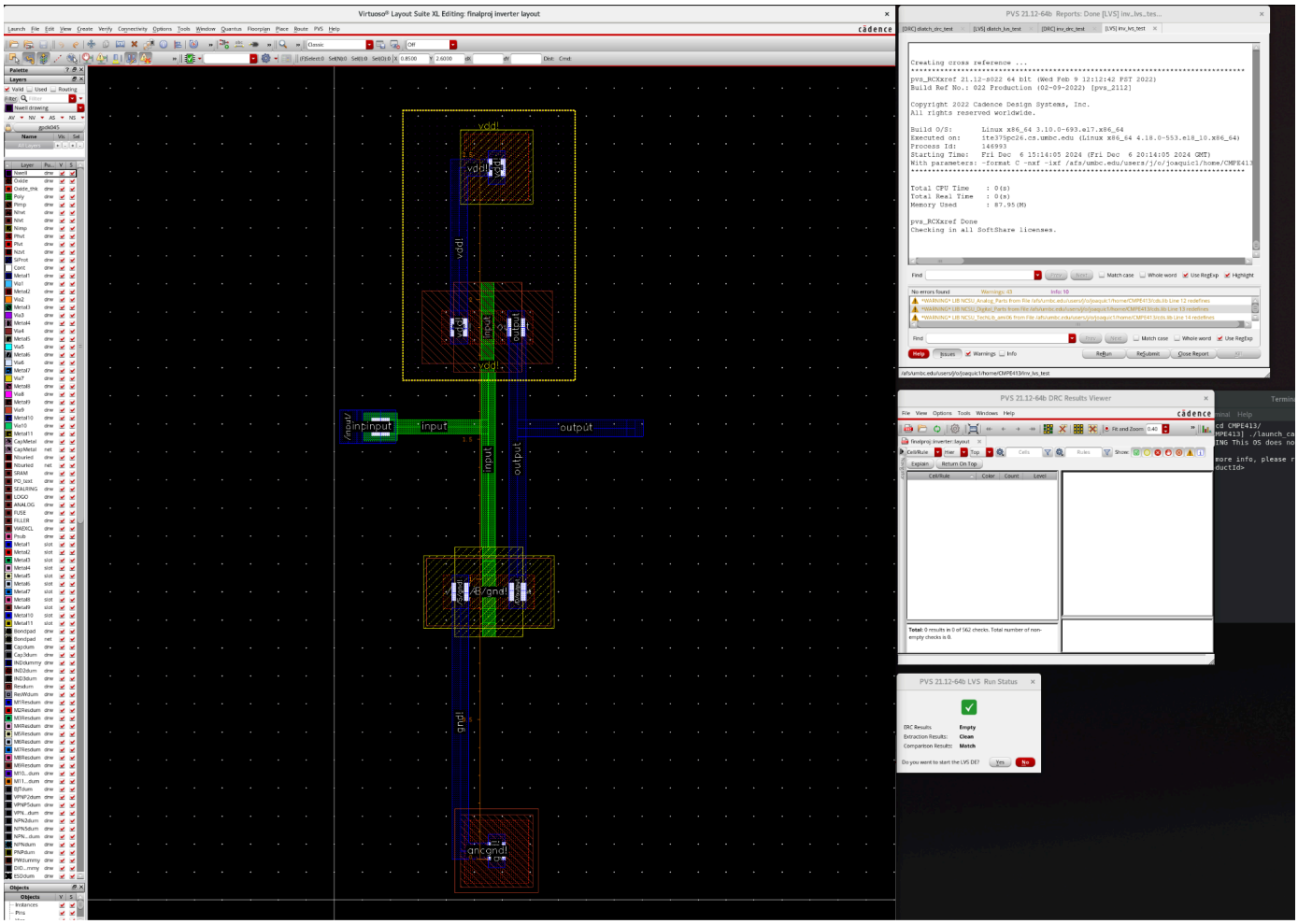

Figure 46: inverter.vhd layout view (DRC + LVS verified)

mux_1bit_2to1.vhd

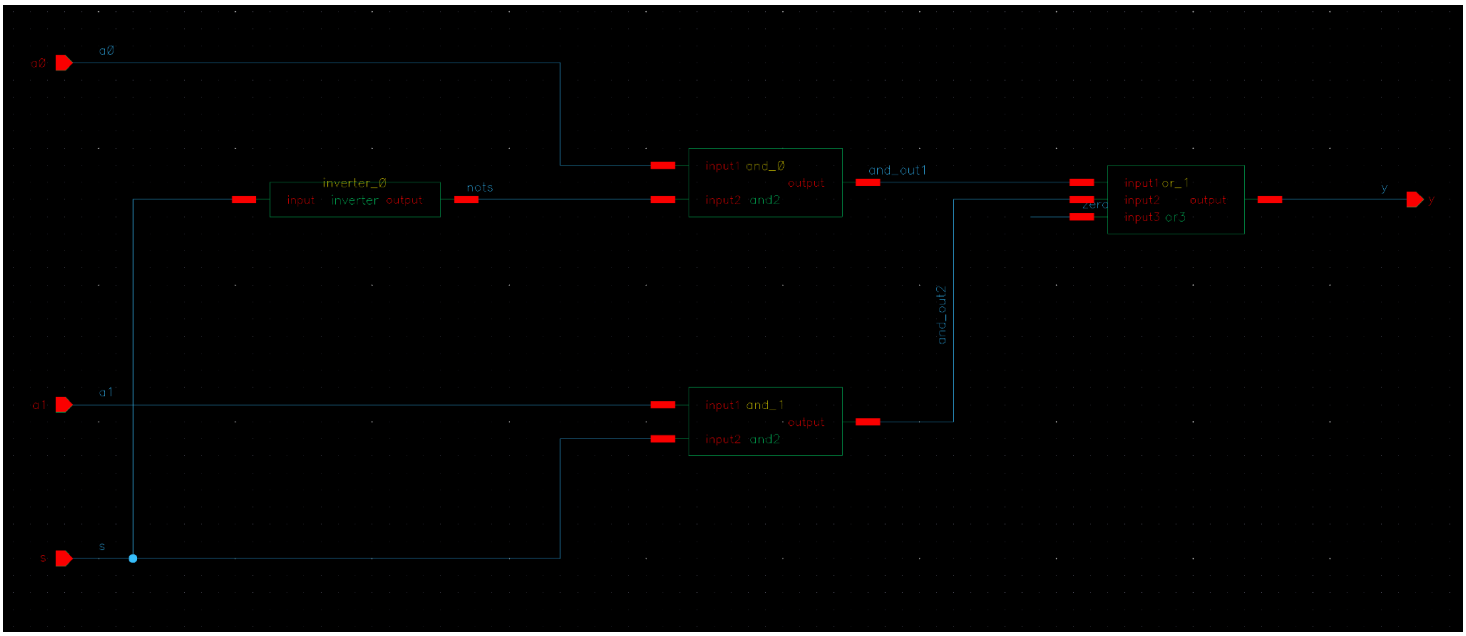

Figure 47: mux_1bit_2to1.vhd schematic view

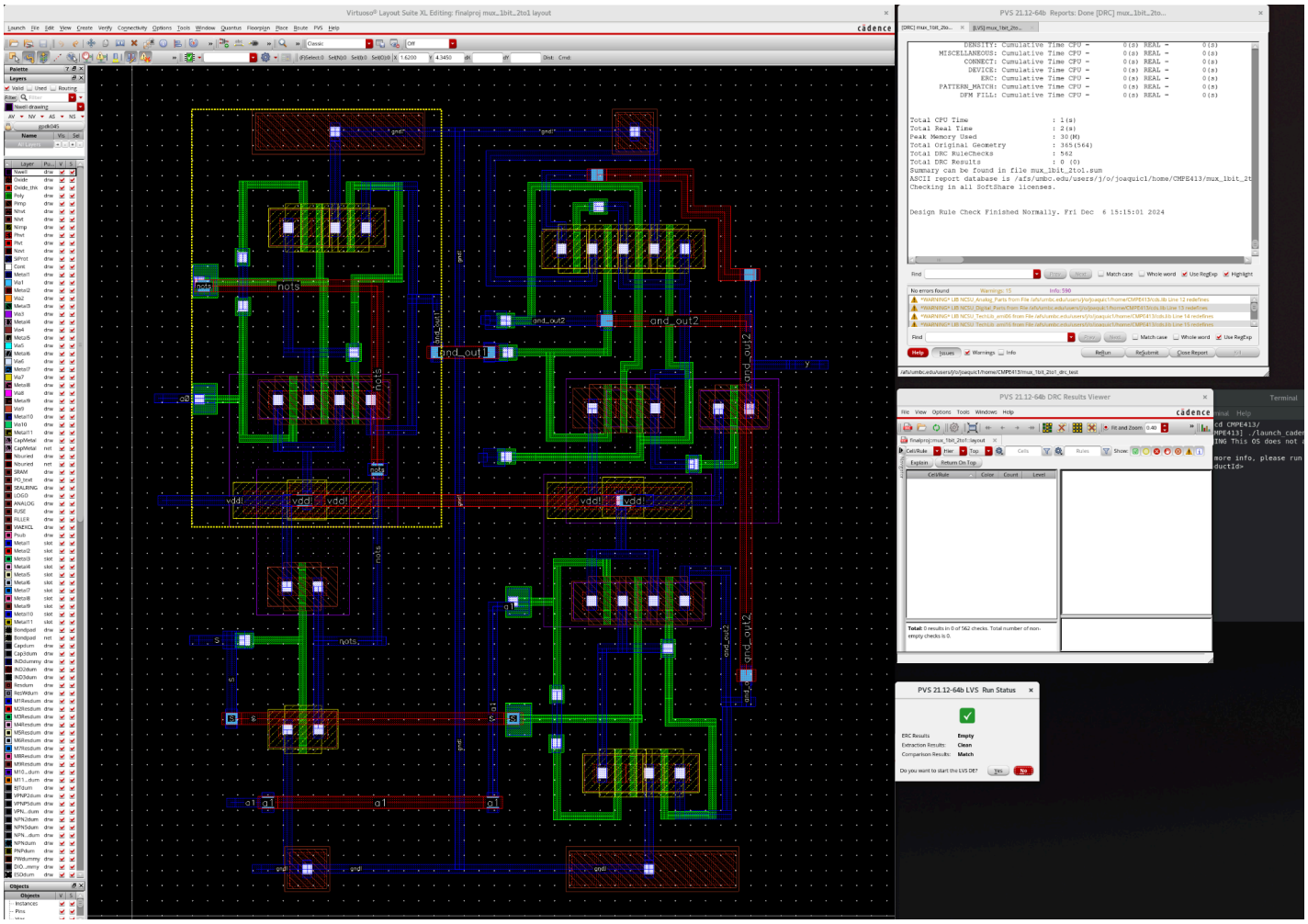

Figure 48: mux_1bit_2to1.vhd layout view (DRC + LVS verified)

mux_1bit_4to1.vhd

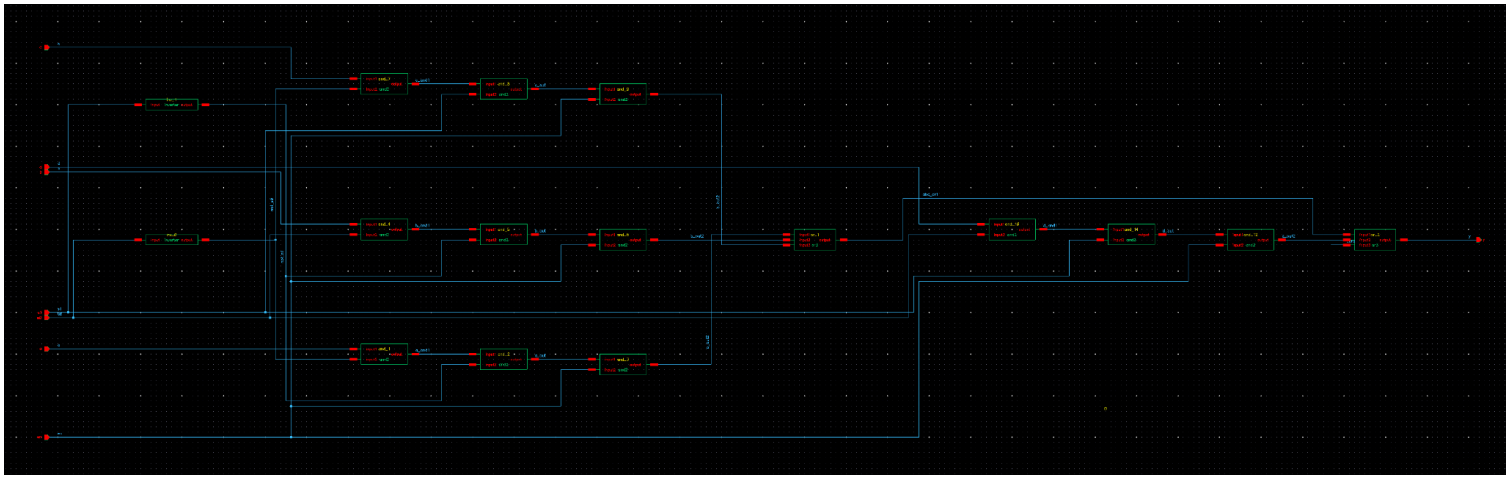

Figure 49: mux_1bit_4to1.vhd schematic view

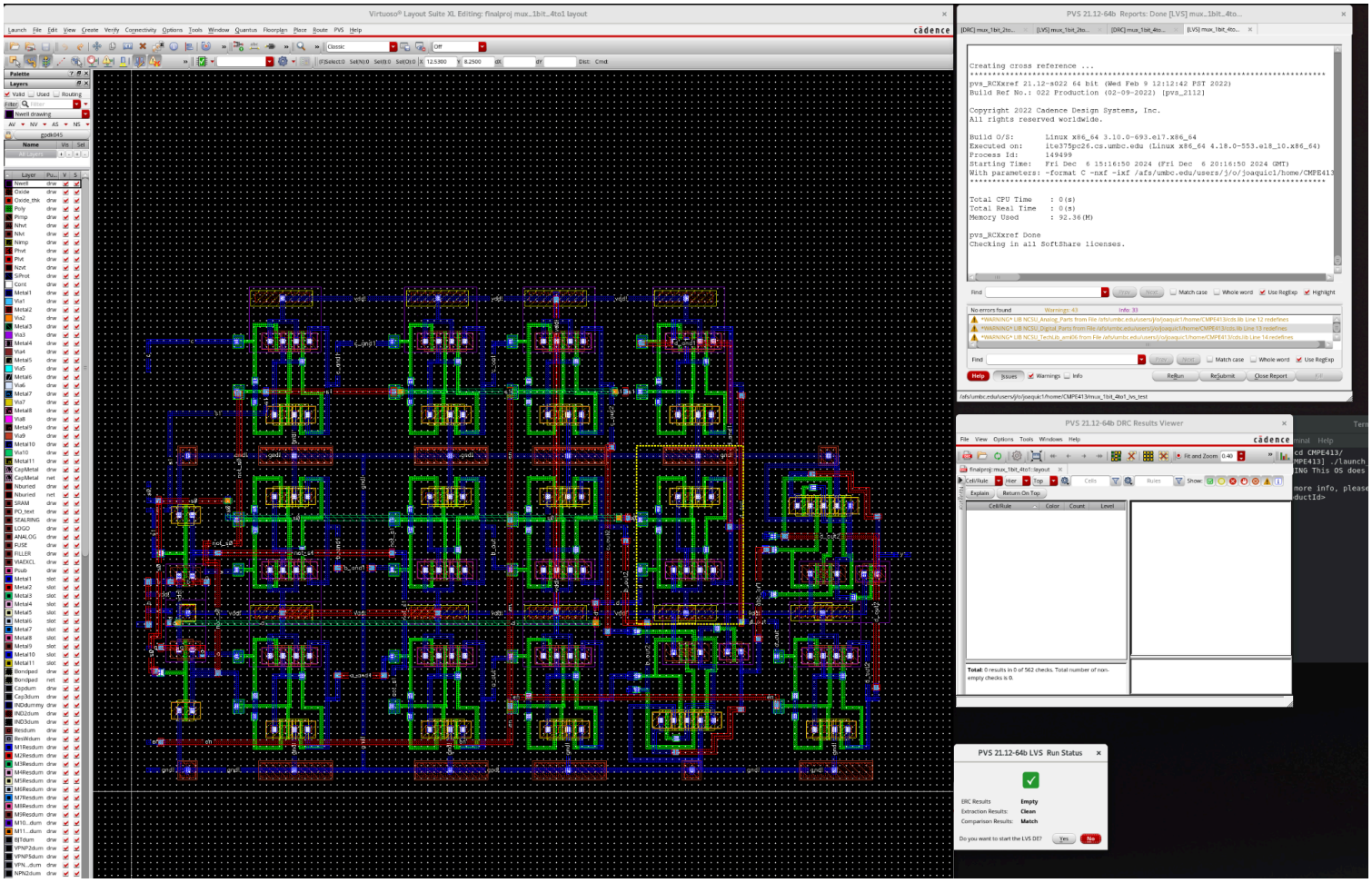

Figure 50: mux_1bit_4to1.vhd layout view (DRC + LVS verified)

mux_8bit_2to1.vhd

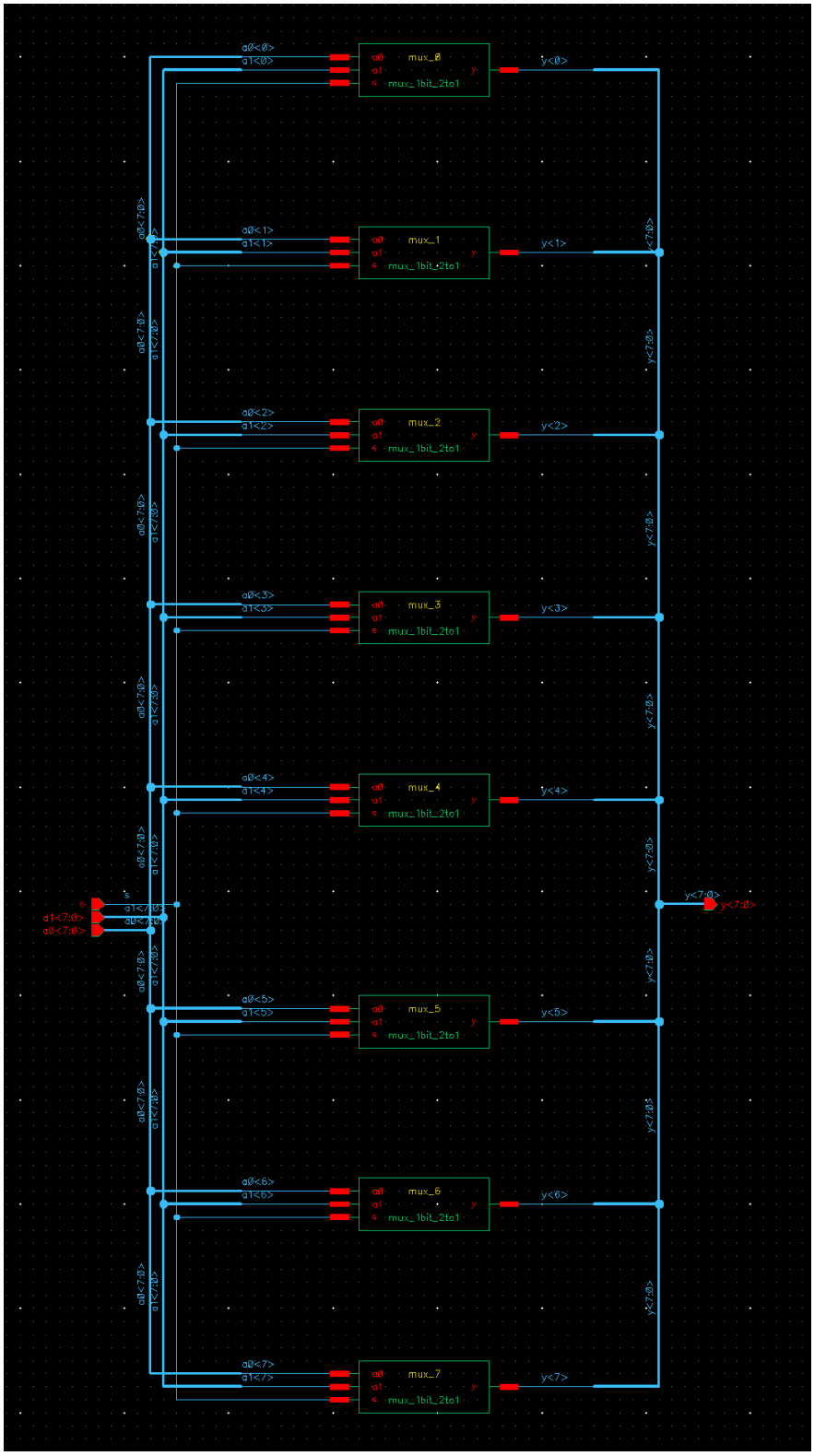

Figure 51: mux_8bit_2to1.vhd schematic view

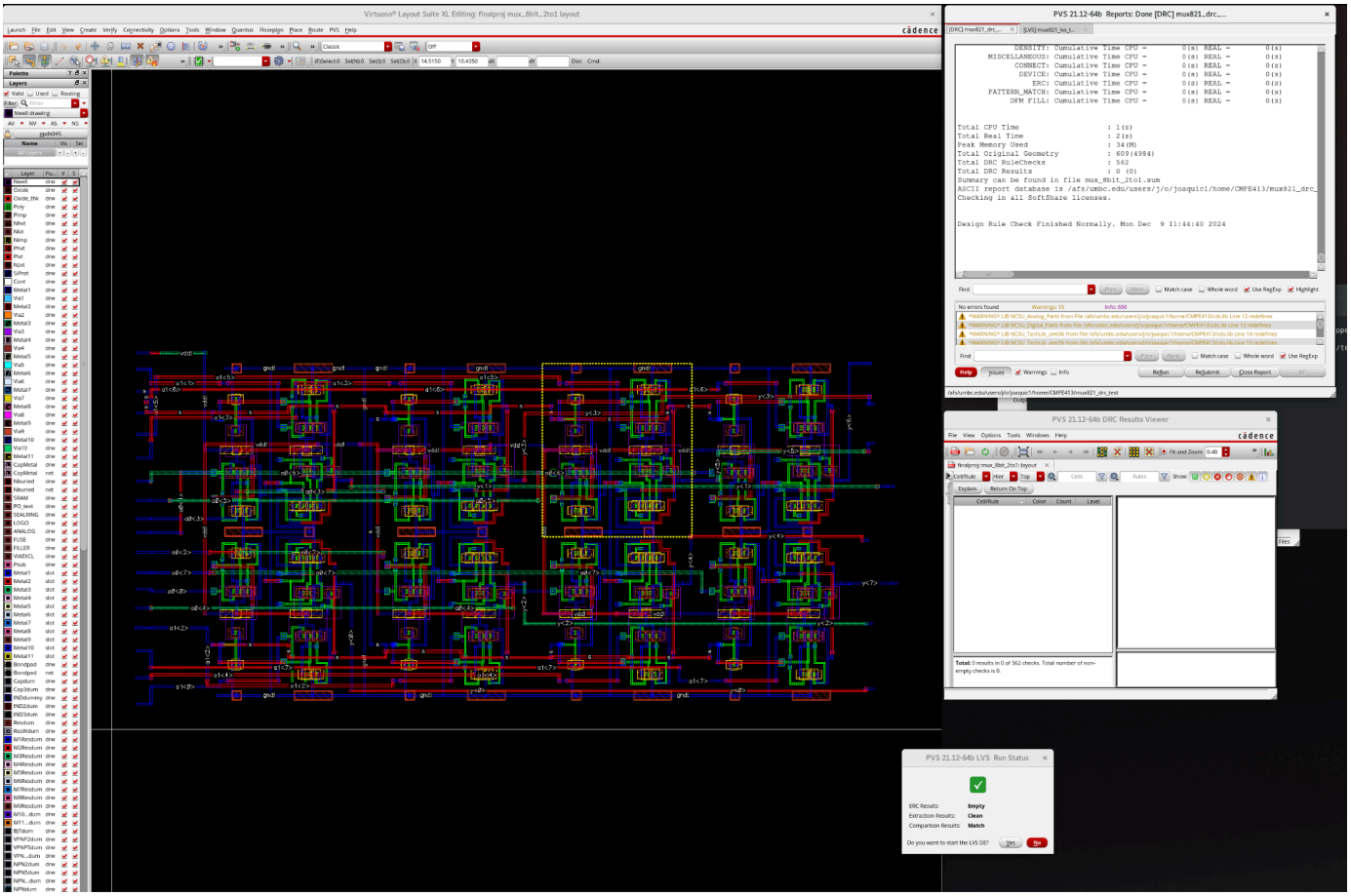

Figure 52: mux_8bit_2to1.vhd layout view (DRC + LVS verified)

LVS Reports are included with the submission of the project directory.

4 Executive Summary

Below is a table describing the work distribution amongst the group.

Work	**Joaquin**	**Bardan**
cache_cell.vhd	%60	%40
cache_byte.vhd	%60	%40
cache_block.vhd	%45	%55
cache_top.vhd	%40	%60
comparator2.vhd	%50	%50
clk_counter.vhd	%30	%70
mux_1bit_4to1.vhd	%60	%40
8bit_2to1_mux.vhd	%40	%60
decoder_4.vhd	%60	%40
chip.vhd	%45	%55
Additional Testbench Files	%50	%50
and2 schematic + layout	%70	%30
and10 schematic + layout	%30	%70
cache_block schematic + layout	%30	%70
cache_byte schematic + layout	%70	%30
cahce_cell schematic + layout	%70	%30
cache_top schematic + layout	%70	%30
chip schematic + layout	%30	%70
clk_counter schematic + layout	%30	%70
comparator2 schematic + layout	%70	%30
decoder_4 schematic + layout	%30	%70

dff_reset schematic + layout	%30	%70
dlatch schematic + layout	%30	%70
inverter schematic + layout	%70	%30
mux_1bit_2to1 schematic + layout	%30	%70
mux_1bit_4to1 schematic + layout	%30	%70
mux_8bit_2to1 schematic + layout	%30	%70
or3 schematic + layout	%70	%30
tx schematic + layout	%70	%30
xnor2 schematic + layout	%70	%30
xor2 schematic + layout	%70	%30
Final Project Report	%50	%50

5 VHDL Module Code

Below is a compilation of all the VHDL Module code used in the cache memory design:

chip.vhd

```
None
--
-- Entity: chip
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity chip is
       Port (
       cpu_add      : in  std_logic_vector(5 downto 0);
       cpu_data   : inout std_logic_vector(7 downto 0);
       cpu_rd_wrn : in  std_logic;
       start        : in  std_logic;
       clk          : in  std_logic;
       reset        : in  std_logic;
       mem_data   : in  std_logic_vector(7 downto 0);
       Vdd          : in  std_logic;
       Gnd          : in  std_logic;
       busy         : out std_logic;
       mem_en       : out std_logic;
       mem_add      : out std_logic_vector(5 downto 0)
       );
end chip;

architecture structural of chip is

component clk_counter
       port (
       clk   : in  std_logic;
       reset : in  std_logic;
       busy : in std_logic;
       s1 : out std_logic;
       s2 : out std_logic;
       s3 : out std_logic;
       s4 : out std_logic;
       s5 : out std_logic;
```

```
       s6 : out std_logic;
       s7 : out std_logic;
       s8 : out std_logic;
       s9 : out std_logic;
       s10 : out std_logic;
       s11 : out std_logic;
       s12 : out std_logic;
       s13 : out std_logic;
       s14 : out std_logic;
       s15 : out std_logic;
       s16 : out std_logic;
       s17 : out std_logic;
       s18 : out std_logic;
       s19 : out std_logic
       );
end component;

component comparator2
       port (
       a0 : in std_logic;
       a1 : in std_logic;
       b0 : in std_logic;
       b1 : in std_logic;
       aeb : out std_logic
       );
end component;

component and2

  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       output   : out std_logic);
end component;

component or3

  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       input3   : in  std_logic;
       output   : out std_logic);
end component;
```

```
component inverter
  port (
       input  : in  std_logic;
       output   : out std_logic);
end component;

component Dlatch
  port ( d   : in  std_logic;
       clk : in  std_logic;
       q   : out std_logic;
       qbar: out std_logic);
end component;

component cache_top
       port (
       block_sel : in std_logic_vector(1 downto 0);
       byte_sel : in std_logic_vector(1 downto 0);
       wr_d : in std_logic_vector(7 downto 0);
       en : in std_logic;
       rd_wr : in std_logic;
       rd_d : out std_logic_vector(7 downto 0)
       );
end component;

component mux_1bit_4to1
       Port (
       A: in std_logic;
       B: in std_logic;
       C: in std_logic;
       D: in std_logic;
       en: in std_logic;
       S0: in std_logic;
       S1: in std_logic;
       Y: out std_logic
       );
end component;

component mux_8bit_2to1
       Port (
       A0 : in std_logic_vector(7 downto 0);
       A1 : in std_logic_vector(7 downto 0);
       S : in std_logic;
       Y : out std_logic_vector(7 downto 0)
       );
```

```
end component;

component mux_1bit_2to1
       Port (
       A0 : in std_logic;
       A1 : in std_logic;
       S : in std_logic;
       Y : out std_logic
       );
end component;

component cache_cell
       port (
       wr_d   : in std_logic;
       ch_en  : in std_logic;
       rd_wr  : in std_logic;
       rd_d   : out std_logic
       );
end component;

-- signals for latching signals at start
signal store_cpu_add : std_logic_vector(5 downto 0) := "000000";
signal block_offset, byte_offset : std_logic_vector(1 downto 0);
signal rd_wr_from_cpu, start_and_not_hit : std_logic;
signal cpu_data_save : std_logic_vector(7 downto 0);

-- signals related to clk counter
signal s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_10, s_11, s_12, s_13,
s_14, s_15, s_16, s_17, s_18, s_19 : std_logic;

-- signals related to busy signal
signal not_busy, reset_or_notbusy, busy_reset, busy_reset0, busy_off_sig :
std_logic;
signal not_clk, busy_and_out, busy_or_out, busy_d, busy_q : std_logic;
signal wr_wire1, rh_wire1, rm_wire1, wr_wire2 : std_logic;

-- signals for hit/miss
signal hit, hit_comparator_out, hit_wValid : std_logic;

-- signals for operation state
signal not_cpu_rd_wrn, not_hit, rd_hitq, wr_hitq, wr_missq, rd_missq :
std_logic;
signal rd_hit, rd_miss, wr_hit, wr_miss : std_logic := '0';
```

```
-- signals related to mem_en output
signal wire_for_mem_en, mem_en_internal : std_logic;
signal undefined : std_logic := 'U';

-- signals for valid bit rd/wr
signal not_s19, not_reset, wd_d_valid, s_1_or_s_19_or_reset, s_1_or_not_s_19,
rd_wr_valid : std_logic;
signal valid_read_1, valid_read_2, valid_read_3, valid_read_4, valid_read :
std_logic;

-- signals for tag bits rd/wr
signal s_1_or_reset, s_1_or_not_reset : std_logic;
signal row1_tag0, row2_tag0, row3_tag0, row4_tag0, row1_tag1, row2_tag1,
row3_tag1, row4_tag1, tag0_read, tag1_read : std_logic;

-- signals for cache rd/wr
signal s10_or_s11, s12_or_s13, s14_or_s15, s16_or_s17, s_2_and_hit, temp,
temp2, temp3 : std_logic;
signal read_miss_en, cache_rd_wr_rm, s_2_and_rd_wr, cache_top_en, cache_rd_wr :
std_logic;
signal wr_d_for_cache : std_logic_vector(7 downto 0);

-- Instantiations
for and_1, and_2, and_3, and_4, and_5, and_6, and_7, and_8, and_9, and_10,
and_11, and_12, and_13, and_14, and_15, and_16, and_17, and_18 : and2 use
entity work.and2(structural);
for or_1, or_2, or_3, or_4, or_5, or_6, or_7, or_8, or_9, or_10, or_11, or_12,
or_13, or_14, or_15, or_16, or_17, or_18, or_19, or_20, or_21 : or3 use entity
work.or3(structural);
for inverter_0, inverter_1, inverter_2, inverter_3, inverter_4, inverter_5,
inverter_6 : inverter use entity work.inverter(structural);

for latch_cpu_add0, latch_cpu_add1, latch_block0, latch_block1, latch_cpu_add4,
latch_cpu_add5, latch_rd_wr_from_cpu, latch_busy_signal, latch_hit,
latch_rd_hit, latch_rd_miss, latch_wr_hit, latch_wr_miss : Dlatch use entity
work.Dlatch(structural);
for latch_cpu_data0, latch_cpu_data1, latch_cpu_data2, latch_cpu_data3,
latch_cpu_data4, latch_cpu_data5, latch_cpu_data6, latch_cpu_data7 : Dlatch use
entity work.Dlatch(structural);

for cc1: clk_counter use entity work.clk_counter(structural);
for comparator_1 : comparator2 use entity work.comparator2(structural);
for valid_1, valid_2, valid_3, valid_4: cache_cell use entity
work.cache_cell(structural);
```

```
for cache_top_1 : cache_top use entity work.cache_top(structural);

for mem_add_1_mux, mem_add_2_mux, mem_add_3_mux, mem_add_4_mux, mem_add_5_mux :
mux_1bit_2to1 use entity work.mux_1bit_2to1(structural);
for valid_mux, tag_mux0, tag_mux1 : mux_1bit_4to1 use entity
work.mux_1bit_4to1(structural);
for cache_mux : mux_8bit_2to1 use entity work.mux_8bit_2to1(structural);

begin

	-------------------------------------------------------
	-- Latching cpu_address

	latch_cpu_add0 : Dlatch
	port map (
		d => cpu_add(0),
		clk => start,
		q => store_cpu_add(0),
		qbar => open
	);

	latch_cpu_add1 : Dlatch
	port map (
		d => cpu_add(1),
		clk => start,
		q => store_cpu_add(1),
		qbar => open
	);

	latch_block0 : Dlatch
	port map (
		d => cpu_add(2),
		clk => start,
		q => block_offset(0),
		qbar => open
	);

	latch_block1 : Dlatch
	port map (
		d => cpu_add(3),
		clk => start,
		q => block_offset(1),
		qbar => open
	);
```

```
latch_cpu_add4 : Dlatch
port map (
       d => cpu_add(4),
       clk => start,
       q => store_cpu_add(4),
       qbar => open
);

latch_cpu_add5 : Dlatch
port map (
       d => cpu_add(5),
       clk => start,
       q => store_cpu_add(5),
       qbar => open
);

-----------------------------------------------------------
-- Latching cpu_rd_wrn

latch_rd_wr_from_cpu : Dlatch
port map (
       d => cpu_rd_wrn,
       clk => start,
       q => rd_wr_from_cpu,
       qbar => open
);

and_1 : and2
port map (
       input1 => start,
       input2 => not_hit,
       output => start_and_not_hit
);

latch_cpu_data0 : Dlatch
port map (
       d => cpu_data(0),
       clk => start_and_not_hit,
       q => cpu_data_save(0),
       qbar => open
);

latch_cpu_data1 : Dlatch
```

```
port map (
       d => cpu_data(1),
       clk => start_and_not_hit,
       q => cpu_data_save(1),
       qbar => open
);

latch_cpu_data2 : Dlatch
port map (
       d => cpu_data(2),
       clk => start_and_not_hit,
       q => cpu_data_save(2),
       qbar => open
);

latch_cpu_data3 : Dlatch
port map (
       d => cpu_data(3),
       clk => start_and_not_hit,
       q => cpu_data_save(3),
       qbar => open
);

latch_cpu_data4 : Dlatch
port map (
       d => cpu_data(4),
       clk => start_and_not_hit,
       q => cpu_data_save(4),
       qbar => open
);


latch_cpu_data5 : Dlatch
port map (
       d => cpu_data(5),
       clk => start_and_not_hit,
       q => cpu_data_save(5),
       qbar => open
);


latch_cpu_data6 : Dlatch
port map (
       d => cpu_data(6),
```

```
        clk => start_and_not_hit,
        q => cpu_data_save(6),
        qbar => open
);

latch_cpu_data7 : Dlatch
port map (
        d => cpu_data(7),
        clk => start_and_not_hit,
        q => cpu_data_save(7),
        qbar => open
);

-----------------------------------------------------------
-- Determines the reset condition for the clk counter
-- clk counter resets when reset = 1 OR busy = 0

inverter_0 : inverter
port map (
        input => busy_d,
        output => not_busy
);

or_1: or3
port map (
        input1 => reset,
        input2 => not_busy,
        input3 => Gnd,
        output => reset_or_notbusy
);
-----------------------------------------------------------
-- clk counter instantiation

cc1: clk_counter
port map (
        clk => clk,
        reset => reset_or_notbusy,
        busy => busy_d,
        s1 => s_1,
        s2 => s_2,
        s3 => s_3,
        s4 => s_4,
        s5 => s_5,
        s6 => s_6,
```

```
        s7 => s_7,
        s8 => s_8,
        s9 => s_9,
        s10 => s_10,
        s11 => s_11,
        s12 => s_12,
        s13 => s_13,
        s14 => s_14,
        s15 => s_15,
        s16 => s_16,
        s17 => s_17,
        s18 => s_18,
        s19 => s_19
);

-------------------------------------------------------
-- Keeps Busy on until a clk signal goes high and resets it back to 0
-- use busy_d if you need to access the busy state

or_2 : or3
port map (
        input1 => busy_off_sig,
        input2 => reset,
        input3 => Gnd,
        output => busy_reset0
);

inverter_1 : inverter
port map (
        input => busy_reset0,
        output => busy_reset
);

inverter_2 : inverter
port map (
        input => clk,
        output => not_clk
);

and_2 : and2
port map (
        input1 => start,
        input2 => busy_reset,
        output => busy_and_out
```

```
);

or_3 : or3
port map (
       input1 => busy_and_out,
       input2 => busy_d,
       input3 => Gnd,
       output => busy_or_out
);

and_3 : and2
port map (
       input1 => busy_reset,
       input2 => busy_or_out,
       output => busy_d
);

latch_busy_signal : Dlatch
port map (
       d => busy_d,
       clk => not_clk,
       q => busy,
       qbar => open
);

------------------------------------------------------------
-- The logic for determining a hit/miss

comparator_1 : comparator2
port map (
       a0 => cpu_add(4), -- take the cpu_add tag bits
       a1 => tag0_read,
       b0 => cpu_add(5), -- take the cpu_add tag bits
       b1 => tag1_read,
       aeb => hit_comparator_out -- comparison of tag bits
);

and_4 : and2
port map (
input1 => hit_comparator_out,
input2 => valid_read,
output => hit_wValid
);
```

```
latch_hit : Dlatch
port map (
       d => hit_wValid,
       clk => start,
       q => hit, -- latch hit/miss state for reference
       qbar => not_hit
);

-------------------------------------------------------
-- Establishes control signals for each state of a read/write hit/miss

inverter_3 : inverter
port map (
       input => cpu_rd_wrn,
       output => not_cpu_rd_wrn
);

-- read hit
and_5 : and2
port map (
       input1 => hit,
       input2 => cpu_rd_wrn,
       output => rd_hit
);

latch_rd_hit : Dlatch
port map (
       d => rd_hit,
       clk => start,
       q => rd_hitq,-- latch rd_hit signal
       qbar => open
);

-- write hit
and_6 : and2
port map (
       input1 => hit,
       input2 => not_cpu_rd_wrn,
       output => wr_hit
);

latch_wr_hit : Dlatch
port map (
       d => wr_hit,
```

```
        clk => start,
        q => wr_hitq, -- latch wr_hit signal
        qbar => open
    );

    -- write miss
    and_7 : and2
    port map (
        input1 => not_hit,
        input2 => not_cpu_rd_wrn,
        output => wr_miss
    );

    latch_wr_miss : Dlatch
    port map (
        d => wr_miss,
        clk => start,
        q => wr_missq, -- latch wr_miss signal
        qbar => open
    );

    -- read miss
    and_8 : and2
    port map (
        input1 => not_hit,
        input2 => cpu_rd_wrn,
        output => rd_miss
    );

    latch_rd_miss : Dlatch
    port map (
        d => rd_miss,
        clk => start,
        q => rd_missq, -- latch rd_miss signal
        qbar => open
    );

    ---------------------------------------------------------
    -- when to reset busy based on hit/miss

    or_4 : or3
    port map (
        input1 => wr_hitq,
        input2 => wr_missq,
```

```
        input3 => Gnd,
        output => wr_wire1
    );

    and_9 : and2
    port map (
        input1 => rd_missq,
        input2 => s_19,
        output => rm_wire1
    );

    and_10 : and2
    port map (
        input1 => rd_hitq,
        input2 => s_2,
        output => rh_wire1
    );

    and_11 : and2
    port map (
        input1 => wr_wire1,
        input2 => s_3,
        output => wr_wire2
    );

    or_5 : or3
    port map (
        input1 => wr_wire2,
        input2 => rm_wire1,
        input3 => rh_wire1,
        output => busy_off_sig
    );

    -------------------------------------------------------
    -- mem_en signal

    and_12 : and2
    port map (
        input1 => rd_wr_from_cpu,
        input2 => not_hit,
        output => wire_for_mem_en
    );

    and_13 : and2
```

```
port map (
       input1 => wire_for_mem_en,
       input2 => s_2,
       output => mem_en
);

and_14 : and2
port map (
       input1 => wire_for_mem_en,
       input2 => s_2,
       output => mem_en_internal
);

mem_add_0_mux : mux_1bit_2to1
port map (
       A0 => undefined,
       A1 => store_cpu_add(0),
       S => mem_en_internal,
       Y => mem_add(0)
);

mem_add_1_mux : mux_1bit_2to1
port map (
       A0 => undefined,
       A1 => store_cpu_add(1),
       S => mem_en_internal,
       Y => mem_add(1)
);

mem_add_2_mux : mux_1bit_2to1
port map (
       A0 => undefined,
       A1 => block_offset(0),
       S => mem_en_internal,
       Y => mem_add(2)
);

mem_add_3_mux : mux_1bit_2to1
port map (
       A0 => undefined,
       A1 => block_offset(1),
       S => mem_en_internal,
       Y => mem_add(3)
);
```

```
mem_add_4_mux : mux_1bit_2to1
port map (
       A0 => undefined,
       A1 => store_cpu_add(4),
       S => mem_en_internal,
       Y => mem_add(4)
);

mem_add_5_mux : mux_1bit_2to1
port map (
       A0 => undefined,
       A1 => store_cpu_add(5),
       S => mem_en_internal,
       Y => mem_add(5)
);

-----------------------------------------------------------
-- reading and writing valid bit logic

inverter_4 : inverter
port map (
       input => s_19,
       output => not_s19
);

inverter_5 : inverter
port map (
       input => reset,
       output => not_reset
);

or_6 : or3
port map (
       input1 => s_19,
       input2 => not_reset,
       input3 => Gnd,
       output => wd_d_valid
);

or_7 : or3
port map (
       input1 => s_19,
       input2 => s_1,
```

```
        input3 => reset,
        output => s_1_or_s_19_or_reset
);

or_8 : or3
port map (
        input1 => not_s19,
        input2 => s_1,
        input3 => Gnd,
        output => s_1_or_not_s_19
);

and_15 : and2
port map (
        input1 => s_1_or_not_s_19,
        input2 => not_reset,
        output => rd_wr_valid
);

valid_1: cache_cell
port map (
        wr_d  => wd_d_valid,
        ch_en => s_1_or_s_19_or_reset,
        rd_wr => rd_wr_valid,
        rd_d => valid_read_1
);

valid_2: cache_cell
port map (
        wr_d  => wd_d_valid,
        ch_en => s_1_or_s_19_or_reset,
        rd_wr => rd_wr_valid,
        rd_d => valid_read_2
);

valid_3: cache_cell
port map (
        wr_d  => wd_d_valid,
        ch_en => s_1_or_s_19_or_reset,
        rd_wr => rd_wr_valid,
        rd_d => valid_read_3
);

valid_4: cache_cell
```

```
port map (
       wr_d  => wd_d_valid,
       ch_en => s_1_or_s_19_or_reset,
       rd_wr => rd_wr_valid,
       rd_d => valid_read_4
);

valid_mux : mux_1bit_4to1
port map (
       A => valid_read_1,
       B => valid_read_2,
       C => valid_read_3,
       D => valid_read_4,
       en => Vdd,
       S0 => block_offset(0),
       S1 => block_offset(1),
       Y => valid_read
);

-------------------------------------------------------
-- reading and writing tag bits logic

or_9 : or3
port map (
       input1 => s_1,
       input2 => reset,
       input3 => Gnd,
       output => s_1_or_reset
);

or_10 : or3
port map (
       input1 => s_1,
       input2 => not_reset,
       input3 => Gnd,
       output => s_1_or_not_reset
);

row1_tag_0: cache_cell
port map (
       wr_d  => Gnd,
       ch_en => s_1_or_reset,
       rd_wr => s_1,
       rd_d => row1_tag0
```

```
);

row1_tag_1: cache_cell
port map (
       wr_d  => Gnd,
       ch_en => s_1_or_reset,
       rd_wr => s_1_or_not_reset,
       rd_d => row1_tag1
);

row2_tag_0: cache_cell
port map (
       wr_d  => Gnd,
       ch_en => s_1_or_reset,
       rd_wr => s_1,
       rd_d => row2_tag0
);

row2_tag_1: cache_cell
port map (
       wr_d  => Gnd,
       ch_en => s_1_or_reset,
       rd_wr => s_1_or_not_reset,
       rd_d => row2_tag1
);

row3_tag_0: cache_cell
port map (
       wr_d  => Gnd,
       ch_en => s_1_or_reset,
       rd_wr => s_1,
       rd_d => row3_tag0
);

row3_tag_1: cache_cell
port map (
       wr_d  => Gnd,
       ch_en => s_1_or_reset,
       rd_wr => s_1_or_not_reset,
       rd_d => row3_tag1
);

row4_tag_0: cache_cell
port map (
```

```
        wr_d  => Gnd,
        ch_en => s_1_or_reset,
        rd_wr => s_1,
        rd_d => row4_tag0
    );

    row4_tag_1: cache_cell
    port map (
        wr_d  => Gnd,
        ch_en => s_1_or_reset,
        rd_wr => s_1_or_not_reset,
        rd_d => row4_tag1
    );

    tag_mux0 : mux_1bit_4to1
    port map (
        A => row1_tag0,
        B => row2_tag0,
        C => row3_tag0,
        D => row4_tag0,
        en => s_1_or_not_reset,
        S0 => block_offset(0),
        S1 => block_offset(1),
        Y => tag0_read
    );

    tag_mux1 : mux_1bit_4to1
    port map (
        A => row1_tag1,
        B => row2_tag1,
        C => row3_tag1,
        D => row4_tag1,
        en => s_1_or_not_reset,
        S0 => block_offset(0),
        S1 => block_offset(1),
        Y => tag1_read
    );

    ---------------------------------------------------------
    -- reading and writing to cache logic (unfinished)

    or_11 : or3
    port map (
        input1 => s_10,
```

```
        input2 => s_11,
        input3 => Gnd,
        output => s10_or_s11
);

or_12 : or3
port map (
        input1 => s_12,
        input2 => s_13,
        input3 => Gnd,
        output => s12_or_s13
);

or_13 : or3
port map (
        input1 => s_14,
        input2 => s_15,
        input3 => Gnd,
        output => s14_or_s15
);

or_14 : or3
port map (
        input1 => s_16,
        input2 => s_17,
        input3 => Gnd,
        output => s16_or_s17
);


or_15: or3
port map (
input1 => s14_or_s15,
input2 => s16_or_s17,
input3 => Gnd,
output => byte_offset(1)
);

or_16: or3
port map (
input1 => s12_or_s13,
input2 => s16_or_s17,
input3 => Gnd,
output => byte_offset(0)
```

```
);

and_16 : and2
port map (
       input1 => s_2,
       input2 => hit,
       output => s_2_and_hit
);

or_17: or3
port map (
input1 => s_12,
input2 => s_14,
input3 => s_16,
output => temp
);

or_18: or3
port map (
input1 => temp,
input2 => Gnd,
input3 => s_10,
output => temp2
);

or_19: or3
port map (
input1 => temp,
input2 => s_19,
input3 => s_10,
output => read_miss_en
);

inverter_6 : inverter
port map (
       input => temp2,
       output => temp3
);

and_17: and2
port map (
       input1 => temp3,
       input2 => s_19,
       output => cache_rd_wr_rm
```

```
);

and_18 : and2
port map (
       input1 => s_2_and_hit,
       input2 => rd_wr_from_cpu,
       output => s_2_and_rd_wr
);

or_20 : or3
port map (
       input1 => s_2_and_hit,
       input2 => read_miss_en,
       input3 => Gnd,
       output => cache_top_en
);

or_21 : or3
port map (
       input1 => s_2_and_rd_wr,
       input2 => cache_rd_wr_rm,
       input3 => Gnd,
       output => cache_rd_wr
);


cache_mux : mux_8bit_2to1
port map (
       A0 => cpu_data_save,
       A1 => mem_data,
       S => read_miss_en,
       Y => wr_d_for_cache
);

cache_top_1: cache_top
port map (
block_sel => block_offset,
byte_sel => byte_offset,
en => cache_top_en,
wr_d => wr_d_for_cache,
rd_wr => cache_rd_wr,
rd_d => cpu_data
);
```

```
end structural;
```

cache_top.vhd

```
None
-- Entity: Cache Top Level
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity cache_top is
  port (
       block_sel : in std_logic_vector(1 downto 0);
       byte_sel : in std_logic_vector(1 downto 0);
       wr_d : in std_logic_vector(7 downto 0);
       en : in std_logic;
       rd_wr : in std_logic;
       rd_d : out std_logic_vector(7 downto 0)
  );
end cache_top;

--define architecture
architecture structural of cache_top is

component cache_block
  port (
       wr_d : in std_logic_vector(7 downto 0);
       byte_sel : in std_logic_vector(1 downto 0);
       rd_wr : in std_logic;
       ch_en : in std_logic;
       rd_d : out std_logic_vector(7 downto 0)
  );
end component;

component decoder_4
  port (
       a0   : in  std_logic;
       a1   : in  std_logic;
       en   : in std_logic;
```

```
      y0   : out std_logic;
      y1   : out std_logic;
      y2   : out std_logic;
      y3   : out std_logic
  );
end component;

--gate/signal declarations
for block_0, block_1, block_2, block_3: cache_block use entity
work.cache_block(structural);
for decoder4_1: decoder_4 use entity work.decoder_4(structural);

--intermediate block ch_en signals
signal block_addr_00 : std_logic;
signal block_addr_01 : std_logic;
signal block_addr_10 : std_logic;
signal block_addr_11 : std_logic;

--pull-high signal
signal one : std_logic := '1';
signal not_byte_sel : std_logic;

begin
  -- select lines
  decoder4_1: decoder_4 port map (a0 => block_sel(0), a1 => block_sel(1), en =>
en, y0 => block_addr_00, y1 => block_addr_01, y2 => block_addr_10, y3 =>
block_addr_11);

  -- cache array
  block_0: cache_block port map (wr_d => wr_d, byte_sel => byte_sel, rd_wr =>
rd_wr, ch_en => block_addr_00, rd_d => rd_d);
  block_1: cache_block port map (wr_d => wr_d, byte_sel => byte_sel, rd_wr =>
rd_wr, ch_en => block_addr_01, rd_d => rd_d);
  block_2: cache_block port map (wr_d => wr_d, byte_sel => byte_sel, rd_wr =>
rd_wr, ch_en => block_addr_10, rd_d => rd_d);
  block_3: cache_block port map (wr_d => wr_d, byte_sel => byte_sel, rd_wr =>
rd_wr, ch_en => block_addr_11, rd_d => rd_d);

end structural;
```

cache_block.vhd

```
None
--
-- Entity: cache_block
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity cache_block is
  port (
       wr_d : in std_logic_vector(7 downto 0);
       byte_sel: in std_logic_vector(1 downto 0);
       ch_en : in std_logic;
       rd_wr : in std_logic;
       rd_d : out std_logic_vector(7 downto 0)
  );
end cache_block;

--define architecture
architecture structural of cache_block is

component cache_byte
  port (
       wr_d : in std_logic_vector(7 downto 0);
       ch_en : in std_logic;
       rd_wr : in std_logic;
       rd_d : out std_logic_vector(7 downto 0)
  );
end component;

component decoder_4
  port (
       a0   : in  std_logic;
       a1   : in  std_logic;
       en   : in std_logic;
       y0   : out std_logic;
       y1   : out std_logic;
       y2   : out std_logic;
```

```
      y3   : out std_logic
  );
end component;

--gate/signal declarations
for byte_0, byte_1, byte_2, byte_3: cache_byte use entity
work.cache_byte(structural);
for decoder4_1 : decoder_4 use entity work.decoder_4(structural);

signal byte_addr_00 : std_logic;
signal byte_addr_01 : std_logic;
signal byte_addr_10 : std_logic;
signal byte_addr_11 : std_logic;

begin

      decoder4_1: decoder_4 port map (a0 => byte_sel(0), a1 => byte_sel(1), en
=> ch_en, y0 => byte_addr_00, y1 => byte_addr_01, y2 => byte_addr_10, y3 =>
byte_addr_11);

      byte_0: cache_byte port map (wr_d => wr_d, ch_en => byte_addr_11, rd_wr
=> rd_wr, rd_d => rd_d);
      byte_1: cache_byte port map (wr_d => wr_d, ch_en => byte_addr_10, rd_wr
=> rd_wr, rd_d => rd_d);
      byte_2: cache_byte port map (wr_d => wr_d, ch_en => byte_addr_01, rd_wr
=> rd_wr, rd_d => rd_d);
      byte_3: cache_byte port map (wr_d => wr_d, ch_en => byte_addr_00, rd_wr
=> rd_wr, rd_d => rd_d);
end structural;
```

cache_byte.vhd

```
None
--
-- Entity: cache_byte
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity cache_byte is
  port (
       wr_d : in std_logic_vector(7 downto 0);
       ch_en : in std_logic;
       rd_wr : in std_logic;
       rd_d : out std_logic_vector(7 downto 0)
  );
end cache_byte;

architecture structural of cache_byte is

component cache_cell
  port (
       wr_d : in std_logic;
       ch_en : in std_logic;
       rd_wr : in std_logic;
       rd_d : out std_logic
  );
end component;

--gate/signal declarations
for cell_0, cell_1, cell_2, cell_3, cell_4, cell_5, cell_6, cell_7: cache_cell
use entity work.cache_cell(structural);

begin
       cell_0: cache_cell port map (wr_d(0), ch_en, rd_wr, rd_d(0));
       cell_1: cache_cell port map (wr_d(1), ch_en, rd_wr, rd_d(1));
       cell_2: cache_cell port map (wr_d(2), ch_en, rd_wr, rd_d(2));
       cell_3: cache_cell port map (wr_d(3), ch_en, rd_wr, rd_d(3));
       cell_4: cache_cell port map (wr_d(4), ch_en, rd_wr, rd_d(4));
       cell_5: cache_cell port map (wr_d(5), ch_en, rd_wr, rd_d(5));
       cell_6: cache_cell port map (wr_d(6), ch_en, rd_wr, rd_d(6));
       cell_7: cache_cell port map (wr_d(7), ch_en, rd_wr, rd_d(7));
```

```
end structural;
```

cache_cell.vhd

```
None
-- Entity: 1-bit cache cell
-- Architecture : structural
-- Author: Joaquin Calilao; Bardan Kafle;
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;


entity cache_cell is
       port (
       wr_d   : in std_logic;
       ch_en  : in std_logic;
       rd_wr  : in std_logic;
       rd_d   : out std_logic
       );
end cache_cell;

architecture structural of cache_cell is

component decoder_4
       port (
       a0   : in std_logic;
       a1   : in std_logic;
       en   : in std_logic;
       y0   : out std_logic;
       y1   : out std_logic;
       y2   : out std_logic;
       y3   : out std_logic
       );
end component;

component dff
  port ( d   : in  std_logic;
       clk : in  std_logic;
       q   : out std_logic;
       qbar: out std_logic);
```

```
end component;

component Dlatch
       port (
       d   : in  std_logic;
       clk : in  std_logic;
       q   : out std_logic;
       qbar: out std_logic
       );
end component;

component tx
       port (
       sel    : in std_logic;
       selnot : in std_logic;
       input  : in std_logic;
       output : out std_logic
       );
end component;

component inverter
       port (
       input : in std_logic;
       output : out std_logic
       );
end component;

component and2
       port (
       input1 : in std_logic;
       input2 : in std_logic;
       output : out std_logic
       );
end component;

-- gate/signal declarations
for decoder_4_1: decoder_4 use entity work.decoder_4(structural);
for tx_1: tx use entity work.tx(structural);
for inverter_1, inverter_2, inverter_3 : inverter use entity
work.inverter(structural);
for Dlatch_1 : Dlatch use entity work.Dlatch(structural);

-- intermediate signals
signal latch_out, d_in, inv_out: std_logic;
```

```
signal wr_en, rd_en : std_logic;
signal high_signal : std_logic := '1';
signal not_rd_en : std_logic;

begin

       decoder_4_1 : decoder_4 port map(rd_wr, high_signal, ch_en, open, open,
wr_en, rd_en);

       inverter_1 : inverter port map (rd_en, not_rd_en);

       inverter_2 : inverter port map (wr_d, inv_out);
       inverter_3 : inverter port map (inv_out, d_in);

       Dlatch_1 : Dlatch port map(d_in, wr_en, latch_out, open);

       tx_1 : tx port map(rd_en, not_rd_en, latch_out, rd_d);

end structural;
```

clk_counter.vhd

```
None
--
-- Entity: clk_counter
-- Architecture : structural
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity clk_counter is
       port (
       clk   : in  std_logic;
       reset : in  std_logic;
       busy : in std_logic;
       s1 : out std_logic;
       s2 : out std_logic;
       s3 : out std_logic;
       s4 : out std_logic;
       s5 : out std_logic;
       s6 : out std_logic;
       s7 : out std_logic;
       s8 : out std_logic;
       s9 : out std_logic;
       s10 : out std_logic;
       s11 : out std_logic;
       s12 : out std_logic;
       s13 : out std_logic;
       s14 : out std_logic;
       s15 : out std_logic;
       s16 : out std_logic;
       s17 : out std_logic;
       s18 : out std_logic;
       s19 : out std_logic
       );
end clk_counter;

architecture structural of clk_counter is

       component dff_reset
       port (
```

```
            d      : in  std_logic;
            clk    : in  std_logic;
            reset  : in  std_logic;
            q      : out std_logic;
            qbar   : out std_logic
      );
      end component;

      component and10
      port(
            input1   : in  std_logic;
            input2   : in  std_logic;
            input3   : in  std_logic;
            input4   : in  std_logic;
            input5   : in  std_logic;
            input6   : in  std_logic;
            input7   : in  std_logic;
            input8   : in  std_logic;
            input9   : in  std_logic;
            input10  : in  std_logic;
            output   : out std_logic
      );
      end component;

      component inverter
      port (
            input  : in  std_logic;
            output : out std_logic
      );
      end component;

      component or3

      port (
            input1   : in  std_logic;
            input2   : in  std_logic;
            input3   : in  std_logic;
            output   : out std_logic);
      end component;

      signal q_signals : std_logic_vector(9 downto 0);
      signal d_signals : std_logic_vector(9 downto 0);
      signal q0bar, q1bar, q2bar, q3bar, q4bar, q5bar, q6bar, q7bar, q8bar,
q9bar : std_logic;
```

```
	signal q_bar_last : std_logic;
	signal zero : std_logic := '0';
	signal not_busy_or_reset, not_busy : std_logic;

	for inverter_1, inverter_2, inverter_3, inverter_4, inverter_5,
inverter_6, inverter_7, inverter_8, inverter_9 , inverter_10, inverter_11:
inverter use entity work.inverter(structural);
	for and_1, and_2, and_3, and_4, and_5, and_6, and_7, and_8, and_9,
and_10, and_11, and_12, and_13, and_14, and_15, and_16, and_17, and_18, and_19
: and10 use entity work.and10(structural);
	for or_1 : or3 use entity work.or3(structural);
	for dff1, dff2, dff3, dff4, dff5, dff6, dff7, dff8, dff9, dff10 :
dff_reset use entity work.dff_reset(structural);


begin

	inverter_1 : inverter
	port map (
		input => busy,
		output => not_busy
	);

	or_1 : or3
	port map (
		input1 => not_busy,
		input2 => reset,
		input3 => zero,
		output => not_busy_or_reset
	);

	dff1: dff_reset
	port map (
		d      => q_bar_last,
		clk   => clk,
		reset => not_busy_or_reset,
		q      => q_signals(0)
	);

	dff2: dff_reset
	port map (
		d      => q_signals(0),
		clk   => clk,
		reset => not_busy_or_reset,
```

```
        q      => q_signals(1)
    );

    dff3: dff_reset
    port map (
        d      => q_signals(1),
        clk    => clk,
        reset => not_busy_or_reset,
        q      => q_signals(2)
    );

    dff4: dff_reset
    port map (
        d      => q_signals(2),
        clk    => clk,
        reset => not_busy_or_reset,
        q      => q_signals(3)
    );

    dff5: dff_reset
    port map (
        d      => q_signals(3),
        clk    => clk,
        reset => not_busy_or_reset,
        q      => q_signals(4)
    );

    dff6: dff_reset
    port map (
        d      => q_signals(4),
        clk    => clk,
        reset => not_busy_or_reset,
        q      => q_signals(5)
    );

    dff7: dff_reset
    port map (
        d      => q_signals(5),
        clk    => clk,
        reset => not_busy_or_reset,
        q      => q_signals(6)
    );

    dff8: dff_reset
```

```
port map (
       d      => q_signals(6),
       clk   => clk,
       reset => not_busy_or_reset,
       q      => q_signals(7)
);

dff9: dff_reset
port map (
       d      => q_signals(7),
       clk   => clk,
       reset => not_busy_or_reset,
       q      => q_signals(8)
);

dff10: dff_reset
port map (
       d      => q_signals(8),
       clk   => clk,
       reset => not_busy_or_reset,
       q      => q_signals(9),
       qbar  => q_bar_last
);

inverter_2: inverter
port map (
       input  => q_signals(0),
       output => q0bar
);

inverter_3: inverter
port map (
       input  => q_signals(1),
       output => q1bar
);

inverter_4: inverter
port map (
       input  => q_signals(2),
       output => q2bar
);

inverter_5: inverter
port map (
```

```
        input  => q_signals(3),
        output => q3bar
    );

    inverter_6: inverter
    port map (
        input  => q_signals(4),
        output => q4bar
    );

    inverter_7: inverter
    port map (
        input  => q_signals(5),
        output => q5bar
    );

    inverter_8: inverter
    port map (
        input  => q_signals(6),
        output => q6bar
    );

    inverter_9: inverter
    port map (
        input  => q_signals(7),
        output => q7bar
    );

    inverter_10: inverter
    port map (
        input  => q_signals(8),
        output => q8bar
    );

    inverter_11: inverter
    port map (
        input  => q_signals(9),
        output => q9bar
    );

    and_1: and10
    port map (
        input1  => q_signals(0),
        input2  => q1bar,
```

```
        input3  => q2bar,
        input4  => q3bar,
        input5  => q4bar,
        input6  => q5bar,
        input7  => q6bar,
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s1
    );

    and_2: and10
    port map (
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q2bar,
        input4  => q3bar,
        input5  => q4bar,
        input6  => q5bar,
        input7  => q6bar,
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s2
    );

    and_3: and10
    port map (
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q3bar,
        input5  => q4bar,
        input6  => q5bar,
        input7  => q6bar,
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s3
    );

    and_4: and10
    port map (
        input1  => q_signals(0),
```

```
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q_signals(3),
        input5  => q4bar,
        input6  => q5bar,
        input7  => q6bar,
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s4
    );

    and_5: and10
    port map (
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q_signals(3),
        input5  => q_signals(4),
        input6  => q5bar,
        input7  => q6bar,
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s5
    );

    and_6: and10
    port map (
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q_signals(3),
        input5  => q_signals(4),
        input6  => q_signals(5),
        input7  => q6bar,
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s6
    );

    and_7: and10
    port map (
```

```
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q_signals(3),
        input5  => q_signals(4),
        input6  => q_signals(5),
        input7  => q_signals(6),
        input8  => q7bar,
        input9  => q8bar,
        input10 => q9bar,
        output  => s7
    );

    and_8: and10
    port map (
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q_signals(3),
        input5  => q_signals(4),
        input6  => q_signals(5),
        input7  => q_signals(6),
        input8  => q_signals(7),
        input9  => q8bar,
        input10 => q9bar,
        output  => s8
    );

    and_9: and10
    port map (
        input1  => q_signals(0),
        input2  => q_signals(1),
        input3  => q_signals(2),
        input4  => q_signals(3),
        input5  => q_signals(4),
        input6  => q_signals(5),
        input7  => q_signals(6),
        input8  => q_signals(7),
        input9  => q_signals(8),
        input10 => q9bar,
        output  => s9
    );

    and_10: and10
```

```
port map (
       input1  => q_signals(0),
       input2  => q_signals(1),
       input3  => q_signals(2),
       input4  => q_signals(3),
       input5  => q_signals(4),
       input6  => q_signals(5),
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s10
);

and_11: and10
port map (
       input1  => q0bar,
       input2  => q_signals(1),
       input3  => q_signals(2),
       input4  => q_signals(3),
       input5  => q_signals(4),
       input6  => q_signals(5),
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s11
);

and_12: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q_signals(2),
       input4  => q_signals(3),
       input5  => q_signals(4),
       input6  => q_signals(5),
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s12
);
```

```
and_13: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q2bar,
       input4  => q_signals(3),
       input5  => q_signals(4),
       input6  => q_signals(5),
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s13
);

and_14: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q2bar,
       input4  => q3bar,
       input5  => q_signals(4),
       input6  => q_signals(5),
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s14
);

and_15: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q2bar,
       input4  => q3bar,
       input5  => q4bar,
       input6  => q_signals(5),
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s15
);
```

```
and_16: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q2bar,
       input4  => q3bar,
       input5  => q4bar,
       input6  => q5bar,
       input7  => q_signals(6),
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s16
);

and_17: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q2bar,
       input4  => q3bar,
       input5  => q4bar,
       input6  => q5bar,
       input7  => q6bar,
       input8  => q_signals(7),
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s17
);

and_18: and10
port map (
       input1  => q0bar,
       input2  => q1bar,
       input3  => q2bar,
       input4  => q3bar,
       input5  => q4bar,
       input6  => q5bar,
       input7  => q6bar,
       input8  => q7bar,
       input9  => q_signals(8),
       input10 => q_signals(9),
       output  => s18
```

```
      );


      and_19: and10
      port map (
            input1  => q0bar,
            input2  => q1bar,
            input3  => q2bar,
            input4  => q3bar,
            input5  => q4bar,
            input6  => q5bar,
            input7  => q6bar,
            input8  => q7bar,
            input9  => q8bar,
            input10 => q_signals(9),
            output  => s19
      );

end structural;
```

comparator2.vhd

```
None
-- Entity: 2 bit comparator (non-magnitude, only equality)
-- Architecture : structural
-- Author: Joaquin Calilao
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity comparator2 is
  port ( a0   : in  std_logic;
      a1: in  std_logic;
      b0: in std_logic;
      b1   : in std_logic;
      aeb   : out std_logic);
end comparator2;

--define architecture
architecture structural of comparator2 is
```

```
component xnor2
  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       output   : out std_logic);
end component;

component and2
  port (
       input1 : in std_logic;
       input2 : in std_logic;
       output : out std_logic);
end component;

--gate/signal declarations
for and2_1: and2 use entity work.and2(structural);
for xnor2_1, xnor2_2: xnor2 use entity work.xnor2(structural);

--inverter output label
signal y0temp: std_logic;
signal y1temp: std_logic;

--intermediate AND output label

begin
       xnor2_1: xnor2 port map (a0, b0, y0temp);
       xnor2_2: xnor2 port map (a1, b1, y1temp);
       and2_1: and2 port map (y0temp, y1temp, aeb);

end structural;
```

Dlatch.vhd

```
None
-- Entity: positive level triggered D latch
-- Architecture : structural
-- Author:
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity Dlatch is
  port ( d   : in  std_logic;
       clk : in  std_logic;
       q   : out std_logic;
       qbar: out std_logic);
end Dlatch;

architecture structural of Dlatch is

begin

  output: process (d,clk)

  begin
       if clk = '1' then
       q <= d;
       qbar <= not d ;
 end if;
 end process output;

end structural;
```

and10.vhd

```
None
--
-- Entity: and10
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity and10 is
       Port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       input3   : in  std_logic;
       input4   : in  std_logic;
       input5   : in  std_logic;
       input6   : in  std_logic;
       input7   : in  std_logic;
       input8   : in  std_logic;
       input9   : in  std_logic;
       input10  : in  std_logic;
       output   : out std_logic
       );
end and10;

architecture structural of and10 is

component and2
       port (
       input1 : in std_logic;
       input2 : in std_logic;
       output : out std_logic
       );
end component;

for and_1, and_2, and_3, and_4, and_5, and_6, and_7, and_8, and_9 : and2 use
entity work.and2(structural);
signal and_1_out, and_2_out, and_3_out, and_4_out, and_5_out, and_6_out,
and_7_out, and_8_out : std_logic;
```

```
begin

       and_1 : and2 port map (input1 => input1, input2 => input2, output =>
and_1_out);
       and_2 : and2 port map (input1 => and_1_out, input2 => input3, output =>
and_2_out);
       and_3 : and2 port map (input1 => and_2_out, input2 => input4, output =>
and_3_out);
       and_4 : and2 port map (input1 => and_3_out, input2 => input5, output =>
and_4_out);
       and_5 : and2 port map (input1 => and_4_out, input2 => input6, output =>
and_5_out);
       and_6 : and2 port map (input1 => and_5_out, input2 => input7, output =>
and_6_out);
       and_7 : and2 port map (input1 => and_6_out, input2 => input8, output =>
and_7_out);
       and_8 : and2 port map (input1 => and_7_out, input2 => input9, output =>
and_8_out);
       and_9 : and2 port map (input1 => and_8_out, input2 => input10, output =>
output);

end structural;
```

and2.vhd

```
None
--
-- Entity: and2
-- Architecture : structural
-- Author: cpatel2
-- Created On: 11/11/2003
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity and2 is

  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       output   : out std_logic);
```

```
end and2;

architecture structural of and2 is

begin

  output <= input2 and input1;

end structural;
```

decoder_4.vhd

```
None
-- Entity: 2 to 4 decoder
-- Architecture : structural
-- Author: Joaquin Calilao
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity decoder_4 is
  port ( a0   : in  std_logic;
       a1: in  std_logic;
       en: in std_logic;
       y0   : out std_logic;
       y1   : out std_logic;
       y2   : out std_logic;
       y3   : out std_logic);
end decoder_4;

--define architecture
architecture structural of decoder_4 is

component and2
  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       output   : out std_logic);
end component;
```

```
component inverter
  port (
       input  : in  std_logic;
       output   : out std_logic);
end component;

--gate/signal declarations
for inv_1, inv_2: inverter use entity work.inverter(structural);
for and2_1, and2_2, and2_3, and2_4, and2_5, and2_6, and2_7, and2_8: and2 use
entity work.and2(structural);

--inverter output label
signal a0temp: std_logic;
signal a1temp: std_logic;

--intermediate AND output label
signal y0temp: std_logic;
signal y1temp: std_logic;
signal y2temp: std_logic;
signal y3temp: std_logic;

begin

       inv_1: inverter port map (a0, a0temp);
       inv_2: inverter port map (a1, a1temp);

       and2_1: and2 port map (a0temp, a1temp, y0temp);
       and2_2: and2 port map (a0, a1temp, y1temp);
       and2_3: and2 port map (a0temp, a1, y2temp);
       and2_4: and2 port map (a0, a1, y3temp);

       and2_5: and2 port map (y0temp, en, y0);
       and2_6: and2 port map (y1temp, en, y1);
       and2_7: and2 port map (y2temp, en, y2);
       and2_8: and2 port map (y3temp, en, y3);

end structural;
```

dff_reset.vhd

```
None
--
-- Entity: dff_reset
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity dff_reset is
       port (
       d      : in  std_logic;
       clk    : in  std_logic;
       reset  : in  std_logic;
       q      : out std_logic;
       qbar   : out std_logic
       );
end dff_reset;

architecture structural of dff_reset is

begin

       output: process
       begin
       wait until ( clk'EVENT and clk = '0' );
       if reset = '1' then
              q <= '0';
              qbar <= '1';
       else
              q <= d;
              qbar <= not d;
       end if;
       end process output;

end structural;
```

inverter.vhd

```
None
--
-- Entity: inverter
-- Architecture : structural
-- Author: cpatel2
-- Created On: 11/11/2003
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity inverter is
  port (
       input  : in  std_logic;
       output   : out std_logic);
end inverter;

architecture structural of inverter is

begin

  output <= not (input);

end structural;
```

mux_1bit_2to1.vhd

```
None
--
-- Entity: mux_1bit_2to1
-- Architecture : structural
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_1bit_2to1 is
```

```
        Port (
        A0 : in std_logic;
        A1 : in std_logic;
        S : in std_logic;
        Y : out std_logic
        );
end mux_1bit_2to1;

architecture structural of mux_1bit_2to1 is

component or3
  port (
        input1   : in  std_logic;
        input2   : in  std_logic;
        input3   : in  std_logic;
        output   : out std_logic);
end component;

component and2
  port (
        input1   : in  std_logic;
        input2   : in  std_logic;
        output   : out std_logic);
end component;

component inverter
  port (
        input  : in  std_logic;
        output   : out std_logic);
end component;

for inverter_0 : inverter use entity work.inverter(structural);
for and_0, and_1 : and2 use entity work.and2(structural);
for or_1 : or3 use entity work.or3(structural);

signal notS, and_out1, and_out2 : std_logic;
signal zero : std_logic := '0';

begin

        inverter_0 : inverter
        port map (
                input => S,
                output => notS
```

```
        );

        and_0 : and2
        port map (
                input1 => A0,
                input2 => notS,
                output => and_out1
        );

        and_1 : and2
        port map (
                input1 => A1,
                input2 => S,
                output => and_out2
        );

        or_1 : or3
        port map (
                input1 => and_out1,
                input2 => and_out2,
                input3 => zero,
                output => Y
        );
end structural;
```

mux_1bit_4to1.vhd

```
None
--
-- Entity: mux_1bit_4to1
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_1bit_4to1 is
        Port (
```

```
       A: in std_logic;
       B: in std_logic;
       C: in std_logic;
       D: in std_logic;
       en : in std_logic;
       S0: in std_logic;
       S1: in std_logic;
       Y: out std_logic
       );
end mux_1bit_4to1;

architecture structural of mux_1bit_4to1 is

component or3
  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       input3   : in  std_logic;
       output   : out std_logic);
end component;

component and2
  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       output   : out std_logic);
end component;

component inverter
  port (
       input  : in  std_logic;
       output   : out std_logic);
end component;

signal not_S0, not_S1, A_out, B_out, C_out, D_out, A_out2, B_out2, C_out2,
D_out2, A_and1, B_and1, C_and1, D_and1, ABC_or1 : std_logic;
signal zero : std_logic := '0';

for inv_0, inv_1 : inverter use entity work.inverter(structural);
for and_1, and_2, and_3, and_4, and_5, and_6, and_7, and_8, and_9, and_10,
and_11, and_12 : and2 use entity work.and2(structural);
for or_1, or_2 : or3 use entity work.or3(structural);

begin
```

```
inv_0 : inverter
       port map (
       input => S0,
       output => not_S0
       );

inv_1 : inverter
       port map (
       input => S1,
       output => not_S1
       );

and_1 : and2
       port map (
       input1 => A,
       input2 => not_S0,
       output => A_and1
       );
and_2 : and2
       port map (
       input1 => A_and1,
       input2 => not_S1,
       output => A_out
       );

and_3 : and2
       port map (
       input1 => A_out,
       input2 => en,
       output => A_out2
       );

and_4 : and2
       port map (
       input1 => B,
       input2 => S0,
       output => B_and1
       );
and_5 : and2
       port map (
       input1 => B_and1,
       input2 => not_S1,
       output => B_out
```

```
        );

and_6 : and2
        port map (
        input1 => B_out,
        input2 => en,
        output => B_out2
        );

and_7 : and2
        port map (
        input1 => C,
        input2 => not_S0,
        output => C_and1
        );
and_8 : and2
        port map (
        input1 => C_and1,
        input2 => S1,
        output => C_out
        );

and_9 : and2
        port map (
        input1 => C_out,
        input2 => en,
        output => C_out2
        );

and_10 : and2
        port map (
        input1 => D,
        input2 => S0,
        output => D_and1
        );
and_11 : and2
        port map (
        input1 => D_and1,
        input2 => S1,
        output => D_out
        );

and_12 : and2
        port map (
```

```
       input1 => D_out,
       input2 => en,
       output => D_out2
       );

or_1 : or3
       port map (
       input1 => A_out2,
       input2 => B_out2,
       input3 => C_out2,
       output => ABC_or1
   );

or_2 : or3
       port map (
       input1 => ABC_or1,
       input2 => D_out2,
       input3 => zero,
       output => Y
       );

end structural;
```

mux_8bit_2to1.vhd

```
None
--
-- Entity: mux_8bit_2to1
-- Architecture : structural
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_8bit_2to1 is
       Port (
       A0 : in std_logic_vector(7 downto 0);
       A1 : in std_logic_vector(7 downto 0);
```

```
	S : in std_logic;
	Y : out std_logic_vector(7 downto 0)
	);
end mux_8bit_2to1;

architecture structural of mux_8bit_2to1 is

component mux_1bit_2to1
	Port (
	A0 : in std_logic;
	A1 : in std_logic;
	S : in std_logic;
	Y : out std_logic
	);
end component;

for mux_0, mux_1, mux_2, mux_3, mux_4, mux_5, mux_6, mux_7 : mux_1bit_2to1 use
entity work.mux_1bit_2to1(structural);

begin

	mux_0 : mux_1bit_2to1
	port map (
		A0 => A0(0),
		A1 => A1(0),
		S => S,
		Y => Y(0)
	);

	mux_1 : mux_1bit_2to1
	port map (
		A0 => A0(1),
		A1 => A1(1),
		S => S,
		Y => Y(1)
	);

	mux_2 : mux_1bit_2to1
	port map (
		A0 => A0(2),
		A1 => A1(2),
		S => S,
		Y => Y(2)
	);
```

```
        mux_3 : mux_1bit_2to1
        port map (
                A0 => A0(3),
                A1 => A1(3),
                S => S,
                Y => Y(3)
        );

        mux_4 : mux_1bit_2to1
        port map (
                A0 => A0(4),
                A1 => A1(4),
                S => S,
                Y => Y(4)
        );

        mux_5 : mux_1bit_2to1
        port map (
                A0 => A0(5),
                A1 => A1(5),
                S => S,
                Y => Y(5)
        );

        mux_6 : mux_1bit_2to1
        port map (
                A0 => A0(6),
                A1 => A1(6),
                S => S,
                Y => Y(6)
        );

        mux_7 : mux_1bit_2to1
        port map (
                A0 => A0(7),
                A1 => A1(7),
                S => S,
                Y => Y(7)
        );

end structural;
```

or3.vhd

```
None
--
-- Entity: or3
-- Architecture : structural
-- Author: cpatel2
-- Created On: 11/11/2003
--
library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity or3 is

  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       input3   : in  std_logic;
       output   : out std_logic);
end or3;

architecture structural of or3 is

begin

  output <= input3 or input2 or input1;

end structural;
```

tx.vhd

```
None
-- Entity: tx -- transmission gate
-- Architecture : structural
-- Author:
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity tx is
  port ( sel   : in std_logic;
       selnot: in std_logic;
       input : in std_logic;
       output:out std_logic);
end tx;

architecture structural of tx is

begin

  txprocess: process (sel, selnot, input)
  begin
       if (sel = '1' and selnot = '0') then
       output <= input;
       else
       output <= 'Z';
       end if;
  end process txprocess;

end structural;
```

xnor2.vhd

```
None
-- Entity: xnor2
-- Architecture : structural
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/24

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity xnor2 is
       port (
       input1 : in std_logic;
       input2 : in std_logic;
       output : out std_logic
       );
end xnor2;


architecture structural of xnor2 is

begin
       output <= not(input2 xor input1);
end structural;
```

xor2.vhd

```
None
--
-- Entity: xor2
-- Architecture : structural
-- Author: cpatel2
-- Created On: 11/11/2003
--
library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity xor2 is

  port (
       input1   : in  std_logic;
       input2   : in  std_logic;
       output   : out std_logic);
end xor2;

architecture structural of xor2 is

begin

  output <= input2 xor input1;

end structural;
```

chip_test.vhd

```
None
-- Entity: chip_test
-- Architecture : test
-- Author: cpatel2
-- Created On: 11/01/05
--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use STD.textio.all;

entity chip_test is

end chip_test;

architecture test of chip_test is

  component chip
      port (
      cpu_add      : in  std_logic_vector(5 downto 0);
      cpu_data   : inout  std_logic_vector(7 downto 0);
      cpu_rd_wrn : in  std_logic;
      start        : in  std_logic;
      clk          : in  std_logic;
      reset        : in  std_logic;
      mem_data   : in  std_logic_vector(7 downto 0);
      Vdd     : in  std_logic;
      Gnd          : in  std_logic;
      busy         : out std_logic;
      mem_en       : out std_logic;
      mem_add      : out std_logic_vector(5 downto 0));
  end component;



  for c1 : chip use entity work.chip(structural);

  signal Vdd, Gnd: std_logic;
  signal cpu_data, mem_data: std_logic_vector(7 downto 0);
  signal cpu_add, mem_add: std_logic_vector(5 downto 0);
  signal cpu_rd_wrn, reset, clk, start, clock, busy, mem_en: std_logic;
```

```
  signal clk_count: integer:=0;

procedure print_output is
   variable out_line: line;

   begin
   write (out_line, string' (" Clock: "));
   write (out_line, clk_count);
   write (out_line, string'(" Start: "));
   write (out_line, start);
   write (out_line, string'(" Cpu Read/Write: "));
   write (out_line, cpu_rd_wrn);
   write (out_line, string'(" Reset: "));
   write (out_line, reset);
   writeline(output, out_line);

   write (out_line, string' (" CPU address: "));
   write (out_line, cpu_add);
   write (out_line, string'(" CPU data: "));
   write (out_line, cpu_data);
   writeline(output, out_line);

   write (out_line, string'(" Memory data: "));
   write (out_line, mem_data);
   writeline(output, out_line);
   writeline(output, out_line);

   write (out_line, string'(" Busy: "));
   write (out_line, busy);
   write (out_line, string'(" Memory  Enable: "));
   write (out_line, mem_en);
   writeline(output, out_line);

   write (out_line, string'(" Memory  Address: "));
   write (out_line, mem_add);
   writeline(output, out_line);

   write (out_line, string'("
-----------------------------------------------"));
   writeline(output, out_line);


end print_output;
```

```
begin

  Vdd <= '1';
  Gnd <= '0';
  clk <= clock;

  c1 : chip port map (cpu_add, cpu_data, cpu_rd_wrn, start, clk, reset,
mem_data, Vdd, Gnd, busy, mem_en, mem_add);

  clking : process
  begin
       clock<= '1', '0' after 5 ns;
       wait for 10 ns;
  end process clking;

  io_process: process

       file infile  : text is in "./chip_in.txt";
       variable out_line: line;
       variable buf: line;
       variable value: std_logic_vector(7 downto 0);
       variable value0: std_logic_vector(5 downto 0);
       variable value1: std_logic;

  begin

       while not (endfile(infile)) loop

       wait until rising_edge(clock);
       print_output;

       readline(infile, buf);
       read(buf, value0);
       cpu_add <= value0;

       readline(infile, buf);
       read(buf, value);
       cpu_data <= value;

       readline(infile, buf);
       read(buf, value1);
```

```
        cpu_rd_wrn <= value1;

        readline(infile, buf);
        read(buf, value1);
        start <= value1;

        readline(infile, buf);
        read(buf, value1);
        reset <= value1;

        wait until falling_edge(clock);

        readline(infile, buf);
        read(buf, value);
        mem_data <= value;

        clk_count <= clk_count+1;

        print_output;

        end loop;
        wait;

   end process io_process;


end test;
```

chip_test_stdout.vhd

None

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use STD.textio.all;

entity chip_test is
end chip_test;

architecture test of chip_test is

  component chip
       port (
       cpu_add      : in  std_logic_vector(5 downto 0);
       cpu_data   : inout  std_logic_vector(7 downto 0);
       cpu_rd_wrn : in  std_logic;
       start        : in  std_logic;
       clk          : in  std_logic;
       reset        : in  std_logic;
       mem_data   : in  std_logic_vector(7 downto 0);
       Vdd          : in  std_logic;
       Gnd          : in  std_logic;
       busy         : out std_logic;
       mem_en       : out std_logic;
       mem_add      : out std_logic_vector(5 downto 0)
       );
  end component;

  for c1 : chip use entity work.chip(structural);

  signal Vdd, Gnd: std_logic := '1';
  signal cpu_data, mem_data: std_logic_vector(7 downto 0) := (others => 'Z');
  signal cpu_add, mem_add: std_logic_vector(5 downto 0) := (others => 'Z');
  signal cpu_rd_wrn, reset, clk, start, clock, busy, mem_en: std_logic := '0';

  signal clk_count: integer := 0;

  procedure print_output is
       variable out_line: line;
  begin
       write(out_line, string'(" Clock: "));
       write(out_line, clk_count);
       write(out_line, string'(" Start: "));
```

```
      write(out_line, start);
      write(out_line, string'(" Cpu Read/Write: "));
      write(out_line, cpu_rd_wrn);
      write(out_line, string'(" Reset: "));
      write(out_line, reset);
      writeline(output, out_line);

      write(out_line, string'(" CPU address: "));
      write(out_line, cpu_add);
      write(out_line, string'(" CPU data: "));
      write(out_line, cpu_data);
      writeline(output, out_line);

      write(out_line, string'(" Memory data: "));
      write(out_line, mem_data);
      writeline(output, out_line);

      write(out_line, string'(" Busy: "));
      write(out_line, busy);
      write(out_line, string'(" Memory Enable: "));
      write(out_line, mem_en);
      writeline(output, out_line);

      write(out_line, string'(" Memory Address: "));
      write(out_line, mem_add);
      writeline(output, out_line);

      write(out_line, string'("
----------------------------------------------"));
      writeline(output, out_line);
  end print_output;

begin
  Vdd <= '1';
  Gnd <= '0';
  clk <= clock;
  c1 : chip port map (cpu_add, cpu_data, cpu_rd_wrn, start, clk, reset,
mem_data, Vdd, Gnd, busy, mem_en, mem_add);

  -- Clock Generation
  clking: process
  begin
      clock <= '1', '0' after 5 ns;
      wait for 10 ns;
```

```
  end process clking;

  -- Internal Input Generation and Logging
  io_process: process
  begin
      -- Simulating input values from `chip_in.txt`
      -- Replace each sequence below with patterns derived from the file
cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '1'; mem_data <= "ZZZZZZZZ";
      clk_count <= clk_count + 1;
      wait for 10 ns;
      print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '1'; mem_data <= "ZZZZZZZZ";
      clk_count <= clk_count + 1;
      wait for 10 ns;
      print_output;

cpu_add <= "000000"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= '1'; start <= '1';
reset <= '0'; mem_data <= "ZZZZZZZZ";
      clk_count <= clk_count + 1;
      wait for 10 ns;
      print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
      clk_count <= clk_count + 1;
      wait for 10 ns;
      print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
      clk_count <= clk_count + 1;
      wait for 10 ns;
      print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
      clk_count <= clk_count + 1;
      wait for 10 ns;
      print_output;
```

```
cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000000";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000000";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000001";
```

```
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000001";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000010";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000010";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000011";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "00000011";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
```

```
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "000011"; cpu_data <= "11111111"; cpu_rd_wrn <= '0'; start <= '1';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "000011"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= '1'; start <= '1';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
        clk_count <= clk_count + 1;
        wait for 10 ns;
        print_output;
```

```
cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "111111"; cpu_data <= "10101010"; cpu_rd_wrn <= '0'; start <= '1';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;

cpu_add <= "ZZZZZZ"; cpu_data <= "ZZZZZZZZ"; cpu_rd_wrn <= 'Z'; start <= '0';
reset <= '0'; mem_data <= "ZZZZZZZZ";
       clk_count <= clk_count + 1;
       wait for 10 ns;
       print_output;
  end process io_process;

end test;
```

chip_in.vhd

```
None
ZZZZZZ
ZZZZZZZZ
Z
0
1
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
1
ZZZZZZZZ
000000
ZZZZZZZZ
1
1
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
```

```
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000000
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000000
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000001
ZZZZZZ
```

```
ZZZZZZZZ
Z
0
0
00000001
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000010
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000010
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000011
ZZZZZZ
ZZZZZZZZ
Z
0
0
00000011
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
```

```
0
0
ZZZZZZZZ
000011
11111111
0
1
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
000011
ZZZZZZZZ
1
1
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
```

```
ZZZZZZZZ
111111
10101010
0
1
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
ZZZZZZ
ZZZZZZZZ
Z
0
0
ZZZZZZZZ
```

cache_top_tb.vhd

```
None
--
-- Entity: cache_top_tb
-- Architecture : test
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity cache_top_tb is
end cache_top_tb;

architecture test of cache_top_tb is

	component cache_top
	port (
		block_sel : in std_logic_vector(1 downto 0);
		byte_sel : in std_logic_vector(1 downto 0);
		wr_d : in std_logic_vector(7 downto 0);
		en : in std_logic;
		rd_wr : in std_logic;
		rd_d : out std_logic_vector(7 downto 0)
	);
	end component;

	for cache_top_0 : cache_top use entity work.cache_top(structural);

	signal block_sel : std_logic_vector(1 downto 0);
	signal byte_sel : std_logic_vector(1 downto 0);
	signal wr_d : std_logic_vector(7 downto 0);
	signal rd_wr : std_logic;
	signal en : std_logic;
	signal rd_d : std_logic_vector(7 downto 0);

begin

	cache_top_0 : cache_top
	port map (
		block_sel => block_sel,
```

```
            byte_sel => byte_sel,
            wr_d => wr_d,
            rd_wr => rd_wr,
            en => en,
            rd_d => rd_d
      );

      stim_process : process
      begin
      block_sel <= "00";
      byte_sel <= "00";
      wr_d <= "00100100";
      rd_wr <= '0';
      en <= '1';

      wait for 20ns;

      rd_wr <= '1';

      wait for 20ns;

      wait;
      end process;

end test;
```

cache_cell_test.vhd

None

```
--
-- Entity: cache_test_test
-- Architecture : test
-- Author: Bardan Kafle & Joaquin Calilao
-- Created On: 11/16/2024
--
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
```

```
entity cache_cell_test is

end cache_cell_test;

architecture test of cache_cell_test is

component cache_cell
       port (
       wr_d         : in  std_logic;
       ch_en        : in std_logic;
       rd_wr        : in std_logic;
       rd_d         : out std_logic);
end component;

for cc1 : cache_cell use entity work.cache_cell(structural);
   signal wrd, chen, rdwr, rdd : std_logic;
   signal clock : std_logic;

begin

cc1 : cache_cell port map (wrd,chen,rdwr,rdd);

clk : process
   begin  -- process clk

       clock<='0','1' after 5 ns;
       wait for 10 ns;

  end process clk;

io_process: process

  file infile  : text is in "cache_cell_in.txt";
  file outfile : text is out "cache_cell_out.txt";
  variable wrdx, chenx, rdwrx, rddx : std_logic;
  variable buf : line;

begin

  while not (endfile(infile)) loop


       readline(infile,buf);
       read (buf,wrdx);
```

```
      wrd<=wrdx;

      readline(infile,buf);
      read (buf,chenx);
      chen<=chenx;

      readline(infile,buf);
      read (buf,rdwrx);
      rdwr<=rdwrx;

      wait until falling_edge(clock);

      rddx:=rdd;

      write(buf,rddx);
      writeline(outfile,buf);

  end loop;

end process io_process;

end test;
```

cache_cell_in.vhd

```
None
1
1
0
1
1
1
0
0
1
```

clk_counter_tb.vhd

```
None
--
-- Entity: clk_counter_tb
-- Architecture : test
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity clk_counter_tb is
end clk_counter_tb;

architecture test of clk_counter_tb is

	component clk_counter
	port (
		clk   : in  std_logic;
		reset : in  std_logic;
		busy : in std_logic;
		s1 : out std_logic;
		s2 : out std_logic;
		s3 : out std_logic;
		s4 : out std_logic;
		s5 : out std_logic;
		s6 : out std_logic;
		s7 : out std_logic;
		s8 : out std_logic;
		s9 : out std_logic;
		s10 : out std_logic;
		s11 : out std_logic;
		s12 : out std_logic;
		s13 : out std_logic;
		s14 : out std_logic;
		s15 : out std_logic;
		s16 : out std_logic;
		s17 : out std_logic;
		s18 : out std_logic;
		s19 : out std_logic
	);
```

```
        end component;

        for clk_c : clk_counter use entity work.clk_counter(structural);

        signal clk : std_logic := '0';
        signal reset : std_logic := '0';
        signal busy : std_logic := '0';
        signal s1, s2, s3, s4, s5, s6, s7, s8, s9 : std_logic;
        signal s10, s11, s12, s13, s14, s15, s16, s17, s18, s19 : std_logic;

begin

        clk_c : clk_counter
        port map (
                clk => clk,
                reset => reset,
                busy => busy,
                s1     => s1,
                s2     => s2,
                s3     => s3,
                s4     => s4,
                s5     => s5,
                s6     => s6,
                s7     => s7,
                s8     => s8,
                s9     => s9,
                s10   => s10,
                s11   => s11,
                s12   => s12,
                s13   => s13,
                s14   => s14,
                s15   => s15,
                s16   => s16,
                s17   => s17,
                s18   => s18,
                s19   => s19
        );

        generate_clock : process
        begin
                clk <= '1', '0' after 5ns;
                wait for 10ns;
        end process generate_clock;
```

```
        stim_process : process
        begin

        reset <= '1';
        wait for 10ns;
        reset <= '0';

        wait for 10ns;

        busy <= '1';
        wait for 200ns;
        busy <= '0';

        wait;
        end process;

end test;
```

comparator2_test.vhd

```
None
--
-- Entity: 2-bit comparator test bench
-- Architecture : structural
-- Author: Joaquin Calilao

--define libaries here
library STD;
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;

--entity name no ports since its test bench
entity comparator2_test is

end comparator2_test;

architecture test of comparator2_test is

component comparator2
       port ( a0   : in  std_logic;
              a1: in  std_logic;
              b0: in std_logic;
              b1   : in std_logic;
              aeb   : out std_logic);
end component;

--define an instance of the comparator2
for comparator2_1: comparator2 use entity work.comparator2(structural);

--define signals of the test bench
signal clock : std_logic;

signal a0   : std_logic;
signal a1 : std_logic;
signal b0   : std_logic;
signal b1   : std_logic;
signal aeb : std_logic;

begin

--assign i/p o/p to the ports of the comparator2
```

```
comparator2_1 : comparator2 port map (a0, a1, b0, b1, aeb);

--control inputs
clk : process
   begin  -- process clk

       clock<='0','1' after 5 ns;
       wait for 10 ns;

  end process clk;

io_process: process

file infile : text is in "comparator2_in.txt";
file outfile : text is out "comparator2_out.txt";

variable a0x   : std_logic;
variable a1x   : std_logic;
variable b0x   : std_logic;
variable b1x   : std_logic;
variable aebx   : std_logic;
variable buf : line;

begin

       while not (endfile(infile)) loop
       readline(infile, buf);
       read(buf,a0x);
       a0 <= a0x;

       readline(infile, buf);
       read(buf,a1x);
       a1 <= a1x;

       readline(infile, buf);
       read(buf,b0x);
       b0 <= b0x;

       readline(infile, buf);
       read(buf,b1x);
       b1 <= b1x;

       wait until falling_edge(clock);
       aebx:=aeb;
```

```
        write(buf, aebx);
        writeline(outfile, buf);
        end loop;

        end process io_process;
end test;
```

comparator2_test_in.txt

```
None
1
0
0
1
1
1
1
1
0
0
1
1
0
1
0
1
```

decoder_4_test.vhd

```
None
--
-- Entity: 2 to 4 decoder
-- Architecture : structural
-- Author: Joaquin Calilao

--define libaries here
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;
```

```
--entity name no ports since its test bench
entity decoder_4_test is

end decoder_4_test;

architecture test of decoder_4_test is

component decoder_4
       port ( a0   : in  std_logic;
              a1: in  std_logic;
              en: in std_logic;
              y0   : out std_logic;
              y1   : out std_logic;
              y2   : out std_logic;
              y3   : out std_logic);
end component;

--define an instance of the alu_4
for decoder_4_1: decoder_4 use entity work.decoder_4(structural);

--define signals of the test bench
signal clock : std_logic;

signal a0   : std_logic;
signal a1 : std_logic;
signal en : std_logic;
signal y0   : std_logic;
signal y1   : std_logic;
signal y2   : std_logic;
signal y3   : std_logic;

begin

--assign i/p o/p to the ports of the alu_4
decoder_4_1 : decoder_4 port map (a0, a1, en, y0, y1, y2 ,y3);

--control inputs
clk : process
   begin  -- process clk

       clock<='0','1' after 5 ns;
       wait for 10 ns;
```

```
  end process clk;

--Read inputs from input file called alu_4_in.txt and write output to
alu_4_out.txt

io_process: process

file infile : text is in "decoder_4_in.txt";
file outfile : text is out "decoder_4_out.txt";

variable a0x   : std_logic;
variable a1x : std_logic;
variable enx : std_logic;
variable y0x   : std_logic;
variable y1x   : std_logic;
variable y2x   : std_logic;
variable y3x   : std_logic;
variable buf : line;

begin

       while not (endfile(infile)) loop
       readline(infile, buf);
       read(buf,a0x);
       a0 <= a0x;

       readline(infile, buf);
       read(buf,a1x);
       a1 <= a1x;

       readline(infile, buf);
       read(buf,enx);
       en <= enx;

       wait until falling_edge(clock);
       y0x:=y0;
       y1x:=y1;
       y2x:=y2;
       y3x:=y3;

       write(buf, y0x);
       writeline(outfile, buf);
       write(buf, y1x);
       writeline(outfile, buf);
```

```
        write(buf, y2x);
        writeline(outfile, buf);
        write(buf, y3x);
        writeline(outfile, buf);
        end loop;

        end process io_process;

end test;
```

decoder_4_in.vhd

```
None
0
0
1
1
0
1
0
1
1
1
1
1
0
0
0
1
0
0
0
1
0
1
1
0
```

mux_1bit_4to1_tb.vhd

```
None
--
-- Entity: mux_1bit_4to1_tb
-- Architecture : test
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--


library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_1bit_4to1_tb is
end mux_1bit_4to1_tb;

architecture test of mux_1bit_4to1_tb is

	component mux_1bit_4to1
	port (
		A: in std_logic;
		B: in std_logic;
		C: in std_logic;
		D: in std_logic;
		S0: in std_logic;
		S1: in std_logic;
		Y: out std_logic
	);
	end component;

	for mux_0 : mux_1bit_4to1 use entity work.mux_1bit_4to1(structural);

	signal A, B, C, D, S0, S1, Y : std_logic;

begin

	mux_0 : mux_1bit_4to1
	port map (
		A => A,
		B => B,
		C => C,
		D => D,
```

```
            S0 => S0,
            S1 => S1,
            Y => Y
        );

        stim_process : process
        begin

        A <= '1';
        B <= '0';
        C <= '1';
        D <= '0';

        S0 <= '0';
        S1 <= '0';

        wait for 10ns;

        S0 <= '1';
        S1 <= '0';

        wait for 10ns;

        S0 <= '0';
        S1 <= '1';

        wait for 10ns;

        S0 <= '1';
        S1 <= '1';

        wait;
        end process;

end test;
```

mux_8bit_2to1_tb.vhd

```
None
--
-- Entity: mux_8bit_2to1_tb
-- Architecture : test
-- Author: Bardan Kafle
-- Created On: 11/16/2024
--

library STD;
library IEEE;
use IEEE.std_logic_1164.all;

entity mux_8bit_2to1_tb is
end mux_8bit_2to1_tb;

architecture test of mux_8bit_2to1_tb is

	component mux_8bit_2to1
	port (
		A0: in std_logic_vector(7 downto 0);
		A1: in std_logic_vector(7 downto 0);
		S: in std_logic;
		Y: out std_logic_vector(7 downto 0)
	);
	end component;

	for mux_0 : mux_8bit_2to1 use entity work.mux_8bit_2to1(structural);

	signal A0, A1, Y : std_logic_vector (7 downto 0);
	signal S : std_logic;

begin

	mux_0 : mux_8bit_2to1
	port map (
		A0 => A0,
		A1 => A1,
		S => S,
		Y => Y
	);

	stim_process : process
```

```
        begin

        A0 <= "00000000";
        A1 <= "11111111";
        S <= '0';

        wait for 10ns;

        S <= '1';

        wait;
        end process;

end test;
```