HUB75 SYNOPSYS PROJECT

Cell Generation & Testbench Comparison

SEPTEMBER 24, 2024

Mark Ivey
UTD EECT 6325, VLSI Design

Design Vision Flipflop Output

```
Inferred memory devices in process
  in routine hub75_g_hub_mux4_g_clock_freq200000000_g_color_depth8_g_num_row4_g_num_col4 line
107 in file
    '/home/eng/m/mxi240020/6325/projects/hub75/project-1-library_pair/hub75.vhd'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|   hub_blank_o_reg   | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|   hub_latch_o_reg   | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|    hub_clk_o_reg    | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|    hub_mux_o_reg    | Flip-flop |   4   |  Y  | N  | N  | N  | N  | N  | N  |
|    bcd_time_reg     | Flip-flop |   3   |  Y  | N  | N  | N  | N  | N  | N  |
|      h_row_reg      | Flip-flop |  32   |  Y  | N  | N  | N  | N  | N  | N  |
|      h_col_reg      | Flip-flop |  32   |  Y  | N  | N  | N  | N  | N  | N  |
|   disp_timer_reg    | Flip-flop |  32   |  Y  | N  | N  | N  | N  | N  | N  |
|    clk_timer_reg    | Flip-flop |  32   |  Y  | N  | N  | N  | N  | N  | N  |
|    hub_reg_f_reg    | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|    hub_mem_f_reg    | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|   hub_out_sm_reg    | Flip-flop |   3   |  Y  | N  | N  | N  | N  | N  | N  |
|   hub_piso_r0_reg   | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|   hub_piso_g0_reg   | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|   hub_piso_b0_reg   | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|    hub_r0_o_reg     | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|    hub_g0_o_reg     | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
|    hub_b0_o_reg     | Flip-flop |   1   |  N  | N  | N  | N  | N  | N  | N  |
===============================================================================
Presto compilation completed successfully.
Information: Building the design 'bram_xfer' instantiated from design
'input_mem_g_color_depth8_g_num_row4_g_num_col4' with
  the parameters "g_addr_width=4,g_color_depth=8". (HDL-193)
Inferred memory devices in process
  in routine bram_xfer_g_addr_width4_g_color_depth8 line 29 in file
    '/home/eng/m/mxi240020/6325/projects/hub75/project-1-library_pair/bram_xfer.vhd'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|   data_out_o_reg    | Flip-flop |   8   |  Y  | N  | N  | N  | N  | N  | N  |
|   block_ram_r_reg   | Flip-flop |  256  |  Y  | N  | N  | N  | N  | N  | N  |
===============================================================================
```

Last Lines of the Cell Report:

```
registers/s_mem_sipo_r0_reg[7]
                                                dff            library        7.000000   n
--------------------------------------------------------------------------------
Total 4321 cells                                          11668.000000
```

All code can be found here: https://github.com/Mivey/HUB75_Fork

bram_xfer.vhd file

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use ieee.math_real.all;

entity bram_xfer is
  generic (
    g_addr_width : integer := 8;
    g_color_depth : integer := 8
  );
  port (
    clk         : in    std_logic;
    en          : in    std_logic;
    we_i        : in    std_logic;
    addr_i      : in    unsigned(g_addr_width - 1 downto 0);
    data_in_i   : in    std_logic_vector(g_color_depth - 1 downto 0);
    data_out_o  : out   std_logic_vector(g_color_depth - 1 downto 0)
  );
end entity bram_xfer;

architecture rtl of bram_xfer is
  type t_ram is array(2 ** g_addr_width - 1 downto 0) of std_logic_vector(g_color_depth - 1
downto 0);
  signal block_ram_r : t_ram;

begin
  ram_operation : process (clk) is
  begin
    if rising_edge(clk) then
      if (en = '1') then
        if (we_i = '1') then
          block_ram_r (to_integer(addr_i)) <= data_in_i;
          data_out_o <= data_in_i;
        else
          data_out_o <= block_ram_r(to_integer(addr_i));
        end if;
      end if;
    end if;
  end process ram_operation;
end architecture rtl;
```

hub75.vhd file

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
```

```vhdl
  use ieee.math_real.all;

entity hub75 is
  generic (
    g_hub_mux     : integer := 4;
    g_clock_freq  : integer := 200_000_000;
    g_color_depth : integer := 8;
    g_num_row     : integer := 4;
    g_num_col     : integer := 4
  );
  port (
    clk         : in    std_logic;
    hub_mux_o   : out   unsigned(g_hub_mux - 1 downto 0);
    hub_clk_o   : out   std_logic;
    hub_latch_o : out   std_logic;
    hub_blank_o : out   std_logic;
    hub_g0_o    : out   std_logic;
    hub_b0_o    : out   std_logic;
    hub_r0_o    : out   std_logic;
    mem_sipo_r0 : in    bit_vector(g_color_depth - 1 downto 0);
    mem_sipo_g0 : in    bit_vector(g_color_depth - 1 downto 0);
    mem_sipo_b0 : in    bit_vector(g_color_depth - 1 downto 0);
    hub_reg_f   : out   bit;
    hub_mem_f   : out   bit;
    mem_hub_f   : in    bit;
    reg_hub_f   : in    bit
  );
end entity hub75;
architecture rtl of hub75 is

  type hub_sm is (sm_reset, sm_clk_low, sm_clk_hi, sm_delay, sm_latch_hi);
  signal hub_out_sm : hub_sm := sm_reset;
  constant hub_clk_tick : integer := (g_clock_freq / 20_000_000) - 1;
  type tick_array is array (natural range<>) of integer;
  constant disp_tick : tick_array(g_color_depth - 1 downto 0) :=
  ( 13056 / (2 ** 7) - 1, 13088 / (2 ** 6) - 1, 13088 / (2 ** 5) - 1,
    13088 / (2 ** 4) - 1, 13088 / (2 ** 3) - 1, 13088 / (2 ** 2) - 1,
    13088 / (2 ** 1) - 1, 13088 / (2 ** 0) - 1 );
  signal h_row        : integer;
  signal h_col        : integer;
  signal disp_timer   : integer;
  signal clk_timer    : integer;
  signal bcd_time     : integer range 0 to g_color_depth - 1;
  signal hub_piso_r0 : bit_vector(g_color_depth - 1 downto 0);
  signal hub_piso_g0 : bit_vector(g_color_depth - 1 downto 0);
  signal hub_piso_b0 : bit_vector(g_color_depth - 1 downto 0);

begin
```

```
hub_state_machine : process (clk) is
begin
  if rising_edge(clk) then
    case hub_out_sm is
      -----------------------reset---------------------
      when sm_reset =>
        hub_blank_o <= '1';
        hub_latch_o <= '0';
        hub_clk_o   <= '0';
        hub_mux_o   <= (others => '1');
        bcd_time    <= 0;
        h_row       <= 0;
        h_col       <= 0;
        disp_timer  <= disp_tick(0);
        clk_timer   <= hub_clk_tick;
        hub_reg_f   <= '0';
        hub_mem_f   <= '1';
        if (mem_hub_f = '1') then
          if (reg_hub_f = '1') then -- do I need reg_comp_f?
            hub_piso_r0 <= mem_sipo_r0;
            hub_piso_g0 <= mem_sipo_g0;
            hub_piso_b0 <= mem_sipo_b0;
            hub_out_sm  <= sm_clk_low;
            hub_reg_f   <= '1';
          elsif (reg_hub_f = '0') then
            hub_reg_f  <= '0';
            hub_out_sm <= sm_reset;
          end if;
        end if;
      ----------------------- clk low----------------------
      when sm_clk_low =>
        hub_r0_o <= to_stdulogic(hub_piso_r0(0));
        hub_g0_o <= to_stdulogic(hub_piso_g0(0));
        hub_b0_o <= to_stdulogic(hub_piso_b0(0));
        disp_timer <= disp_timer - 1;
        clk_timer  <= clk_timer - 1;
        if (clk_timer <= (hub_clk_tick / 2)) then
          hub_out_sm <= sm_clk_hi;
          hub_clk_o  <= '1';
        elsif (clk_timer > (hub_clk_tick / 2)) then
          hub_out_sm <= sm_clk_low;
          hub_reg_f  <= '0';
          hub_clk_o  <= '0';
        end if;
      ----------------------- clk high----------------------
      when sm_clk_hi =>
        if (clk_timer = 0) then
          clk_timer  <= hub_clk_tick;                            -- reset timer
          disp_timer <= disp_timer - 1;
          hub_clk_o  <= '0';
          if (h_col < g_num_col - 1) then
            hub_out_sm <= sm_clk_low;                    -- go back and send next bit
            h_col      <= h_col + 1;
            for i in 0 to g_color_depth - 2 loop
              hub_piso_r0(i) <= hub_piso_r0(i + 1);
              hub_piso_g0(i) <= hub_piso_g0(i + 1);
              hub_piso_b0(i) <= hub_piso_b0(i + 1);
            end loop;
          else                                         -- just sent the MSB, so now wait
            hub_out_sm <= sm_delay;
          end if;
        elsif (clk_timer > 0) then
          disp_timer <= disp_timer - 1;
          clk_timer  <= clk_timer - 1;
          hub_out_sm <= sm_clk_hi;
          hub_clk_o  <= '1';
        end if;
      ----------------------- timer delay---------------------
      when sm_delay =>
        hub_clk_o <= '0';
        if (disp_timer > 0) then
          disp_timer <= disp_timer - 1;
          hub_out_sm <= sm_delay;
        elsif (disp_timer = 0) then
          bcd_time    <= bcd_time - 1;
          disp_timer  <= disp_tick(bcd_time - 1);
          clk_timer   <= 20;
          hub_latch_o <= '1';
          hub_out_sm  <= sm_latch_hi;
          hub_piso_r0 <= mem_sipo_r0;
          hub_piso_g0 <= mem_sipo_g0;
          hub_piso_b0 <= mem_sipo_b0;
          if (bcd_time = g_color_depth - 1) then
            h_row       <= h_row + 1;
            hub_blank_o <= '1';
            hub_mux_o   <= to_unsigned (h_row, hub_mux_o'length);
          end if;
        end if;
      ----------------------- latch high---------------------
      when sm_latch_hi =>
        disp_timer <= disp_timer - 1;
        if (clk_timer > 0) then
          clk_timer   <= clk_timer - 1;
          hub_out_sm  <= sm_latch_hi;
          hub_latch_o <= '1';
        elsif (clk_timer = 0) then
          if (mem_hub_f = '0') then
            hub_out_sm <= sm_reset;
```

```
          elsif (reg_hub_f = '0') then
            hub_out_sm <= sm_latch_hi;
          else
            hub_out_sm <= sm_clk_low;
            hub_reg_f <= '1';
          end if;
          hub_latch_o <= '0';
          hub_blank_o <= '0';
        end if;
      when others =>
        null;
    end case;
  end if;
  end process hub_state_machine;
end architecture rtl;
```

input_mem.vhd file

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use ieee.math_real.all;

entity input_mem is
  generic (
    g_color_depth : integer := 8;
    g_num_row     : integer := 8;
    g_num_col     : integer := 8
  );
  port (
    clk          : in    std_logic;
    reset        : in    bit;
    top_data     : in    std_logic_vector(24 - 1 downto 0);
    top_dv_i     : in    std_logic;
    top_u_l      : in    std_logic;
    top_ack      : out   std_logic;
    mem_out_r0   : out   std_logic_vector(g_color_depth - 1 downto 0);
    mem_out_g0   : out   std_logic_vector(g_color_depth - 1 downto 0);
    mem_out_b0   : out   std_logic_vector(g_color_depth - 1 downto 0);
    addr_i       : in    unsigned(4 - 1 downto 0);
    mem_hub_f    : out   bit;
    hub_mem_f    : in    bit;
    reg_mem_f    : in    bit;
    mem_reg_f    : out   bit
  );
end entity input_mem;

architecture rtl of input_mem is
  component bram_xfer
```

```
    generic (
      g_addr_width : integer;
      g_color_depth : integer
    );
    port (
      clk : in std_logic;
      en : in std_logic;
      we_i : in std_logic;
      addr_i : in unsigned(g_addr_width - 1 downto 0);
      data_in_i : in std_logic_vector(g_color_depth - 1 downto 0);
      data_out_o : out std_logic_vector(g_color_depth - 1 downto 0)
    );
  end component;

  type mem_sm is (sm_load, sm_complete, sm_read);
  signal simple_mem_sm : mem_sm := sm_complete;
  type temp_mem is array (0 to 2) of std_logic_vector(g_color_depth - 1 downto 0);
  signal lazy_out : temp_mem;
  signal lazy_in : temp_mem;
  signal mem_timer   : integer;
  signal we_i : std_logic;
  signal s_addr : unsigned(addr_i'range);
  signal mem_full : bit;

  begin
  RGB_Array : for i in lazy_out'range generate
    RGB_Array: bram_xfer
      generic map (
        g_addr_width => 4,
        g_color_depth => g_color_depth
      )
      port map (
        clk   => clk,
        en => '1',
        we_i => we_i,
        addr_i => addr_i,
        data_in_i => lazy_in(i),
        data_out_o => lazy_out(i)
      );
  end generate;
  mem_loader : process (clk) is
  begin
---- this process needs to see that both reg and hub are in a state of reset before it
releases its own reset.
    if rising_edge(clk) then
      if (reset = '0') then
        simple_mem_sm <= sm_complete;
        mem_reg_f     <= '0';
        mem_hub_f     <= '0';
        we_i          <= '0';
      elsif (reset = '1') then
        case simple_mem_sm is

          when sm_load =>
            if (top_dv_i = '0' or (mem_full = '1')) then
              top_ack       <= '1';
              simple_mem_sm <= sm_complete;
              we_i <= '0';
              mem_timer <= 0;
            else
              lazy_in(2) <= top_data(3 * g_color_depth - 1 downto 2 * g_color_depth);
              lazy_in(1) <= top_data(2 * g_color_depth - 1 downto 1 * g_color_depth);
              lazy_in(0) <= top_data(1 * g_color_depth - 1 downto 0 * g_color_depth);
              if (s_addr = "111111") then
                mem_full <= '1';
              else
                mem_full <= '0';
              end if;
              s_addr <= s_addr + 1;
              simple_mem_sm <= sm_load;
            end if;

          when sm_complete =>
            if (top_dv_i = '1') then
              if (mem_timer < 7) then
                simple_mem_sm <= sm_complete;
                mem_timer <= mem_timer + 1;
              elsif (mem_timer >= 7) then
                simple_mem_sm <= sm_load;
                mem_full <= '0';
                s_addr <= (others => '0');
                mem_timer <= 0;
                top_ack <= '0';
                mem_hub_f <= '0';
                mem_reg_f <= '0';
                we_i <= '1';
              end if;
            elsif (top_dv_i = '0') then
              mem_timer <= 0;
              mem_hub_f <= '1';
              mem_reg_f <= '1';
              simple_mem_sm <= sm_read;
              s_addr <= addr_i;
              we_i <= '0';
            end if;

          when sm_read =>
            if (top_dv_i = '0') then
```

```vhdl
              s_addr <= addr_i;
              mem_out_r0 <= lazy_out(2);
              mem_out_g0 <= lazy_out(1);
              mem_out_b0 <= lazy_out(0);
              mem_hub_f <= '1';
              mem_reg_f <= '1';
            elsif (top_dv_i = '1') then
              simple_mem_sm <= sm_complete;
              mem_hub_f <= '0';
              mem_reg_f <= '0';
            end if;
          when others =>
            null;
        end case;
      end if;
    end if;
  end process mem_loader;
end architecture rtl;
```

sipo_reg.vhd file

```vhdl
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use ieee.math_real.all;

entity sipo_reg is
  generic (
    g_color_depth : integer := 8;
    g_num_row     : integer := 4;
    g_num_col     : integer := 4
  );
  port (
    clk          : in    std_logic;
    mem_sipo_r0 : out   bit_vector(g_color_depth - 1 downto 0);
    mem_sipo_g0 : out   bit_vector(g_color_depth - 1 downto 0);
    mem_sipo_b0 : out   bit_vector(g_color_depth - 1 downto 0);
    mem_out_r0  : in    std_logic_vector(g_color_depth - 1 downto 0);
    mem_out_g0  : in    std_logic_vector(g_color_depth - 1 downto 0);
    mem_out_b0  : in    std_logic_vector(g_color_depth - 1 downto 0);
    addr_i      : out   unsigned(4 - 1 downto 0);
    reg_hub_f   : out   bit;
    hub_reg_f   : in    bit;
    reg_mem_f   : out   bit;
    mem_reg_f   : in    bit
  );
end entity sipo_reg;

architecture rtl of sipo_reg is
```

```
  type reg_sm is (sm_stop, sm_load, sm_done);
  signal mem_to_reg : reg_sm := sm_stop;
  signal reg_col_inc   : integer range 0 to g_num_col - 1;
  signal reg_row       : integer range 0 to g_num_row - 1;
  signal reg_cd        : integer range 0 to g_color_depth - 1;
  signal s_mem_sipo_r0 : bit_vector(g_color_depth - 1 downto 0);
  signal s_mem_sipo_g0 : bit_vector(g_color_depth - 1 downto 0);
  signal s_mem_sipo_b0 : bit_vector(g_color_depth - 1 downto 0);
  signal s_delay       : unsigned(1 downto 0);
begin
  reg_loader : process (clk) is
  begin
   if rising_edge(clk) then
      case mem_to_reg is
        when sm_stop =>
          reg_hub_f <= '0';
          if (mem_reg_f = '1') then
            mem_to_reg  <= sm_load;
            reg_row     <= 0;
            reg_cd      <= 0;
            reg_col_inc <= 0;
            addr_i      <= (others => '0');
            addr_i <= to_unsigned(reg_row, 2) & to_unsigned(reg_col_inc, 2);
          elsif (mem_reg_f <= '0') then
            mem_to_reg <= sm_stop;
          end if;

        when sm_load =>
          if (mem_reg_f = '1') then
            for i in g_color_depth - 2 downto 0 loop
              s_mem_sipo_r0(i) <= s_mem_sipo_r0(i + 1);
              s_mem_sipo_g0(i) <= s_mem_sipo_g0(i + 1);
              s_mem_sipo_b0(i) <= s_mem_sipo_b0(i + 1);
            end loop;
            s_mem_sipo_r0(7) <= to_bit(mem_out_r0(reg_cd));
            s_mem_sipo_g0(7) <= to_bit(mem_out_g0(reg_cd));
            s_mem_sipo_b0(7) <= to_bit(mem_out_b0(reg_cd));
            addr_i <= to_unsigned(reg_row, 2) & to_unsigned(reg_col_inc, 2);
            if (reg_col_inc < g_num_col - 1) then
              mem_to_reg  <= sm_load;
              reg_hub_f   <= '0';
              reg_col_inc <= reg_col_inc + 1;
            elsif (reg_col_inc >= g_num_col - 1) then
              mem_to_reg  <= sm_done;
              s_delay <= "11";
              reg_hub_f   <= '0';
              reg_col_inc <= 0;
              if (reg_cd < g_color_depth - 1) then
                reg_cd <= reg_cd + 1;
              elsif (reg_cd >= g_color_depth - 1) then
                reg_cd  <= 0;
                if (reg_row < g_num_row - 1) then
                  reg_row <= reg_row + 1;
                elsif (reg_row >= g_num_row - 1) then
                  reg_row <= 0;
                end if;
              end if;
            end if;
          elsif (mem_reg_f = '0') then
            mem_to_reg  <= sm_stop;
            reg_hub_f   <= '0';
            reg_col_inc <= 0;
          end if;

        when sm_done =>
          mem_sipo_r0 <= s_mem_sipo_r0;
          mem_sipo_g0 <= s_mem_sipo_g0;
          mem_sipo_b0 <= s_mem_sipo_b0;
          if (s_delay = "00") then
            if (mem_reg_f = '1') then
              if (hub_reg_f = '0') then
                mem_to_reg <= sm_done;
                reg_hub_f  <= '1';
              elsif (hub_reg_f = '1') then
                mem_to_reg <= sm_load;
                reg_hub_f  <= '0';
              end if;
            elsif (mem_reg_f = '0') then
              mem_to_reg <= sm_stop;
              reg_hub_f  <= '0';
            end if;
          else
            s_delay <= s_delay - 1;
            reg_hub_f   <= '1';
          end if;

        when others =>
          null;
      end case;
    end if;
  end process reg_loader;
end architecture rtl;
```

<u>top.vhd file</u>

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
```

```vhdl
  use ieee.math_real.all;

entity top is
  generic (
    g_hub_mux     : integer := 4;
    g_clock_freq  : integer := 200_000_000;
    g_color_depth : integer := 8;
    g_num_row     : integer := 4;
    g_num_col     : integer := 4
  );
  port (
    clk         : in    std_logic;
    reset       : in    bit;
    top_data    : in    std_logic_vector(24 - 1 downto 0);
    top_dv_i    : in    std_logic;
    top_u_l     : in    std_logic;
    top_ack     : out   std_logic;
    hub_mux_o   : out   unsigned(g_hub_mux - 1 downto 0);
    hub_clk_o   : out   std_logic;
    hub_latch_o : out   std_logic;
    hub_blank_o : out   std_logic;
    hub_g0_o    : out   std_logic;
    hub_b0_o    : out   std_logic;
    hub_r0_o    : out   std_logic
  );
end entity top;

architecture rtl of top is
  component hub75
    generic (
      g_hub_mux     : integer;
      g_clock_freq  : integer;
      g_color_depth : integer;
      g_num_row     : integer;
      g_num_col     : integer
    );
    port (
      clk         : in    std_logic;
      hub_mux_o   : out   unsigned(g_hub_mux - 1 downto 0);
      hub_clk_o   : out   std_logic;
      hub_latch_o : out   std_logic;
      hub_blank_o : out   std_logic;
      hub_g0_o    : out   std_logic;
      hub_b0_o    : out   std_logic;
      hub_r0_o    : out   std_logic;
      mem_sipo_r0 : in    bit_vector(g_color_depth - 1 downto 0);
      mem_sipo_g0 : in    bit_vector(g_color_depth - 1 downto 0);
      mem_sipo_b0 : in    bit_vector(g_color_depth - 1 downto 0);
      hub_reg_f   : out   bit;
      hub_mem_f   : out   bit;
      mem_hub_f   : in    bit;
      reg_hub_f   : in    bit
    );
  end component;
  component sipo_reg
    generic (
      g_color_depth : integer;
      g_num_row     : integer;
      g_num_col     : integer
    );
    port (
      clk         : in    std_logic;
      mem_sipo_r0 : out   bit_vector(g_color_depth - 1 downto 0);
      mem_sipo_g0 : out   bit_vector(g_color_depth - 1 downto 0);
      mem_sipo_b0 : out   bit_vector(g_color_depth - 1 downto 0);
      mem_out_r0  : in    std_logic_vector(g_color_depth - 1 downto 0);
      mem_out_g0  : in    std_logic_vector(g_color_depth - 1 downto 0);
      mem_out_b0  : in    std_logic_vector(g_color_depth - 1 downto 0);
      addr_i      : out   unsigned(4 - 1 downto 0);
      reg_hub_f   : out   bit;
      hub_reg_f   : in    bit;
      reg_mem_f   : out   bit;
      mem_reg_f   : in    bit
    );
  end component;
  component input_mem
    generic (
      g_color_depth : integer;
      g_num_row     : integer;
      g_num_col     : integer
    );
    port (
      clk        : in    std_logic;
      reset      : in    bit;
      top_data   : in    std_logic_vector(24 - 1 downto 0);
      top_dv_i   : in    std_logic;
      top_u_l    : in    std_logic;
      top_ack    : out   std_logic;
      mem_out_r0 : out   std_logic_vector(g_color_depth - 1 downto 0);
      mem_out_g0 : out   std_logic_vector(g_color_depth - 1 downto 0);
      mem_out_b0 : out   std_logic_vector(g_color_depth - 1 downto 0);
      addr_i     : in    unsigned(4 - 1 downto 0);
      mem_hub_f  : out   bit;
      hub_mem_f  : in    bit;
      reg_mem_f  : in    bit;
      mem_reg_f  : out   bit
    );
  end component;
  signal mem_sipo_r0 :  bit_vector(g_color_depth - 1 downto 0);
  signal mem_sipo_g0 :  bit_vector(g_color_depth - 1 downto 0);
  signal mem_sipo_b0 :  bit_vector(g_color_depth - 1 downto 0);
  signal reg_hub_f   :  bit;
  signal hub_reg_f   :  bit;
  signal mem_out_r0  :   std_logic_vector(g_color_depth - 1 downto 0);
  signal mem_out_g0  :   std_logic_vector(g_color_depth - 1 downto 0);
  signal mem_out_b0  :   std_logic_vector(g_color_depth - 1 downto 0);
  signal addr_i      :   unsigned(4 - 1 downto 0);
  signal mem_hub_f   :   bit;
  signal hub_mem_f   :   bit;
  signal reg_mem_f   :   bit;
  signal mem_reg_f   :   bit;
begin
  registers: sipo_reg
    generic map (
      g_color_depth => g_color_depth,
      g_num_row     => g_num_row,
      g_num_col     => g_num_col
    )
    port map (
      clk   => clk,
      mem_sipo_r0 => mem_sipo_r0, mem_sipo_g0 => mem_sipo_g0, mem_sipo_b0 => mem_sipo_b0,
      mem_out_r0  => mem_out_r0, mem_out_g0  => mem_out_g0, mem_out_b0  => mem_out_b0,
      addr_i => addr_i, reg_hub_f => reg_hub_f, hub_reg_f => hub_reg_f,
      reg_mem_f => reg_mem_f, mem_reg_f => mem_reg_f
    );
  load_n_store: input_mem
    generic map (
      g_color_depth => g_color_depth,
      g_num_row     => g_num_row,
      g_num_col     => g_num_col
    )
    port map (
      clk        => clk,
      reset      => reset,
      top_data   => top_data,
      top_dv_i   => top_dv_i,
      top_u_l    => top_u_l,
      top_ack    => top_ack,
      mem_out_r0 => mem_out_r0,
      mem_out_g0 => mem_out_g0,
      mem_out_b0 => mem_out_b0,
      addr_i     => addr_i,
      mem_hub_f  => mem_hub_f,
      hub_mem_f  => hub_mem_f,
      reg_mem_f  => reg_mem_f,
      mem_reg_f  => mem_reg_f
      );
```

```vhdl
  display_out: hub75
    generic map (
      g_hub_mux     => g_hub_mux,
      g_clock_freq  => g_clock_freq,
      g_color_depth => g_color_depth,
      g_num_row     => g_num_row,
      g_num_col     => g_num_col
    )
    port map (
      clk   => clk,
      hub_mux_o   =>  hub_mux_o,
      hub_clk_o   =>  hub_clk_o,
      hub_latch_o =>  hub_latch_o,
      hub_blank_o =>  hub_blank_o,
      hub_g0_o    =>  hub_g0_o,
      hub_b0_o    =>  hub_b0_o,
      hub_r0_o    =>  hub_r0_o,
      mem_sipo_r0 =>  mem_sipo_r0,
      mem_sipo_g0 =>  mem_sipo_g0,
      mem_sipo_b0 =>  mem_sipo_b0,
      hub_reg_f   =>  hub_reg_f,
      hub_mem_f   =>  hub_mem_f,
      mem_hub_f   =>  mem_hub_f,
      reg_hub_f   =>  reg_hub_f

    );
end architecture;
```

Top_tb.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_tb is
end;

architecture bench of top_tb is
  -- Clock period
  constant clk_period : time := 5 ns;
  -- Generics
  constant g_hub_mux : integer := 3;
  constant g_clock_freq : integer := 200_000_000;
  constant g_color_depth : integer := 8;
  constant g_num_row : integer := 4;
  constant g_num_col : integer := 8;
  constant g_u_row  : integer := 2;
  constant g_u_col  : integer := 3;
  constant g_addr_width  : integer := 5;
```

```
    -- Ports
  signal clk : std_logic;
  signal reset : bit;
  signal top_data : std_logic_vector(24 - 1 downto 0);
  signal top_dv_i : std_logic;
  signal top_u_l : std_logic;
  signal top_ack : std_logic;
  signal hub_mux_o : unsigned(g_hub_mux - 1 downto 0);
  signal hub_clk_o : std_logic;
  signal hub_latch_o : std_logic;
  signal hub_blank_o : std_logic;
  signal hub_g0_o : std_logic;
  signal hub_b0_o : std_logic;
  signal hub_r0_o : std_logic;
  signal finished : boolean := false;
begin
  top_inst : entity work.top
  generic map (
    g_hub_mux => g_hub_mux,
    g_clock_freq => g_clock_freq,
    g_color_depth => g_color_depth,
    g_num_row => g_num_row,
    g_num_col => g_num_col,
    g_u_row => g_u_row,
    g_u_col => g_u_col,
    g_addr_width => g_addr_width
  )
  port map (
    clk => clk,
    reset => reset,
    top_data => top_data,
    top_dv_i => top_dv_i,
    top_u_l => top_u_l,
    top_ack => top_ack,
    hub_mux_o => hub_mux_o,
    hub_clk_o => hub_clk_o,
    hub_latch_o => hub_latch_o,
    hub_blank_o => hub_blank_o,
    hub_g0_o => hub_g0_o,
    hub_b0_o => hub_b0_o,
    hub_r0_o => hub_r0_o
  );
process is
begin
  clk <= '1';
  wait for clk_period;
  clk <= '0';
  wait for clk_period;
  if finished then
```

```
    wait;
  end if;

end process;
process is
  variable temp : unsigned(3 downto 0);
begin
  reset <= '0';
  wait for clk_period * 10;
  reset <= '1';
  wait for clk_period * 2;
  top_dv_i <= '1';
  top_data <= X"AA55AA";
  wait for clk_period * 16;
  for i in 0 to 7 loop
    temp := to_unsigned(i, 4);
    if (temp(0) = '0' ) then
      wait for clk_period * 2;
      top_data <= X"55AA55";
    else
      wait for clk_period * 2;
      top_data <= X"AA55AA";
    end if;
  end loop;
    wait for clk_period * 2;
  top_data <= X"dec0de";
  wait for clk_period * 2;
  top_data <= X"c0ffee";
  for i in 0 to 7 loop
    temp := to_unsigned(i, 4);
    if (temp(0) = '0' ) then
      wait for clk_period * 2;
      top_data <= X"55AA55";
    else
      wait for clk_period * 2;
      top_data <= X"AA55AA";
    end if;
    -- wait for clk_period * 2;
  end loop;
    wait for clk_period * 2;
  top_dv_i <= '0';
  wait on top_ack for 1000 * clk_period;
  wait for clk_period * 20;
  for i in 0 to 31 loop
    -- reg_hub_f <= '0';
    wait for clk_period * 120880;
  end loop;
for i in 0 to 31 loop
    wait for 2 * clk_period;
  end loop;
    wait for 2 * clk_period;
    wait for clk_period * 2;
    top_dv_i <= '1';
    top_data <= X"AA55AA";
    wait for clk_period * 16;
    for i in 0 to 7 loop
      temp := to_unsigned(i, 4);
      if (temp(0) = '0' ) then
        wait for clk_period * 2;
        top_data <= X"55AA55";
      else
        wait for clk_period * 2;
        top_data <= X"AA55AA";
      end if;
      -- wait for clk_period * 2;
    end loop;
      wait for clk_period * 2;
    top_data <= X"beef1e";
    wait for clk_period * 2;
    top_data <= X"b0d1e5";
    for i in 0 to 7 loop
      temp := to_unsigned(i, 4);
      if (temp(0) = '1' ) then
        wait for clk_period * 2;
        top_data <= X"a5a5a5";
      else
        wait for clk_period * 2;
        top_data <= X"969696";
      end if;
      -- wait for clk_period * 2;
    end loop;
      wait for clk_period * 2;
    top_dv_i <= '0';
    wait on top_ack for 1000 * clk_period;
    wait for clk_period * 20;
    for i in 0 to 31 loop
      wait for 2 * clk_period;
    end loop;
      wait for 2 * clk_period;
  finished <= true;
  wait for clk_period * 2;
  wait;
 end process;
end;
```

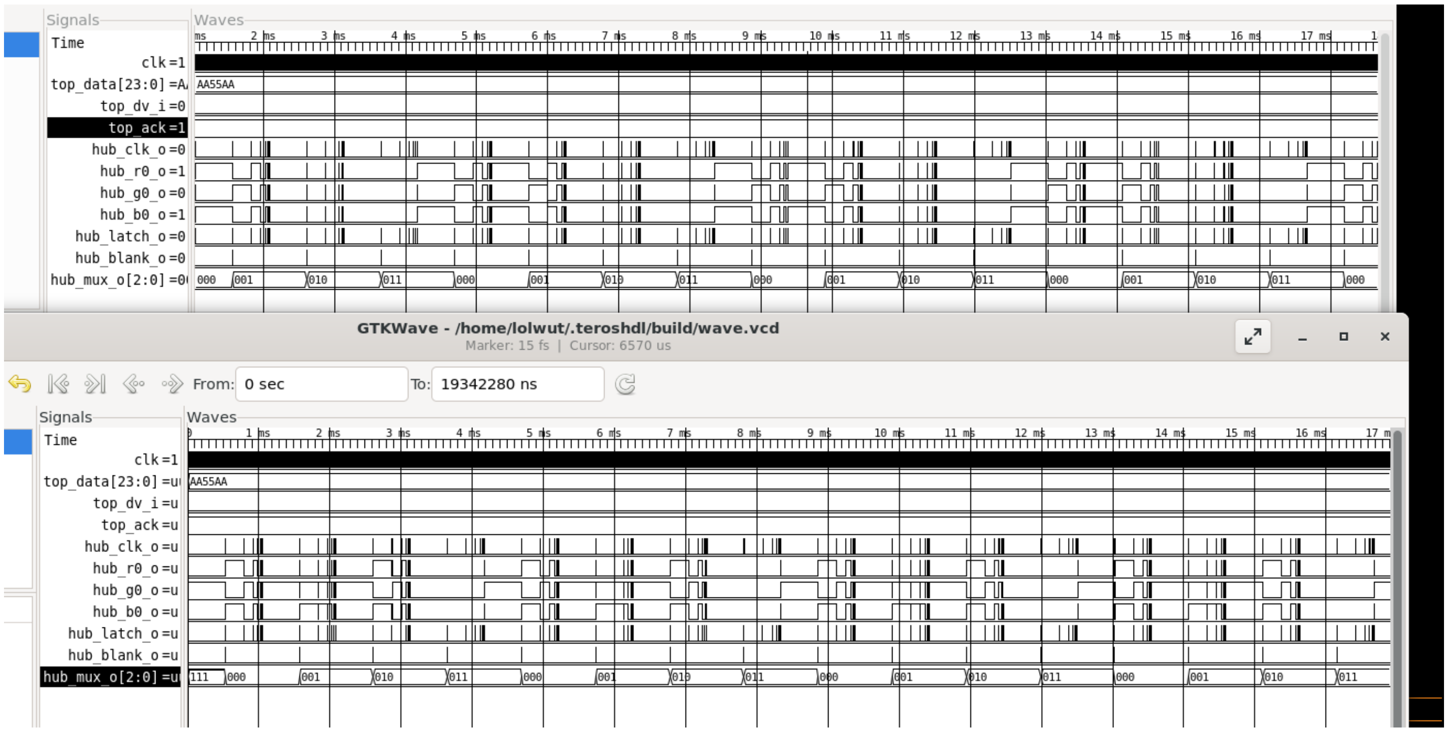

Figure 1: Identical full panel output from behavioral and Synopsys netlist.

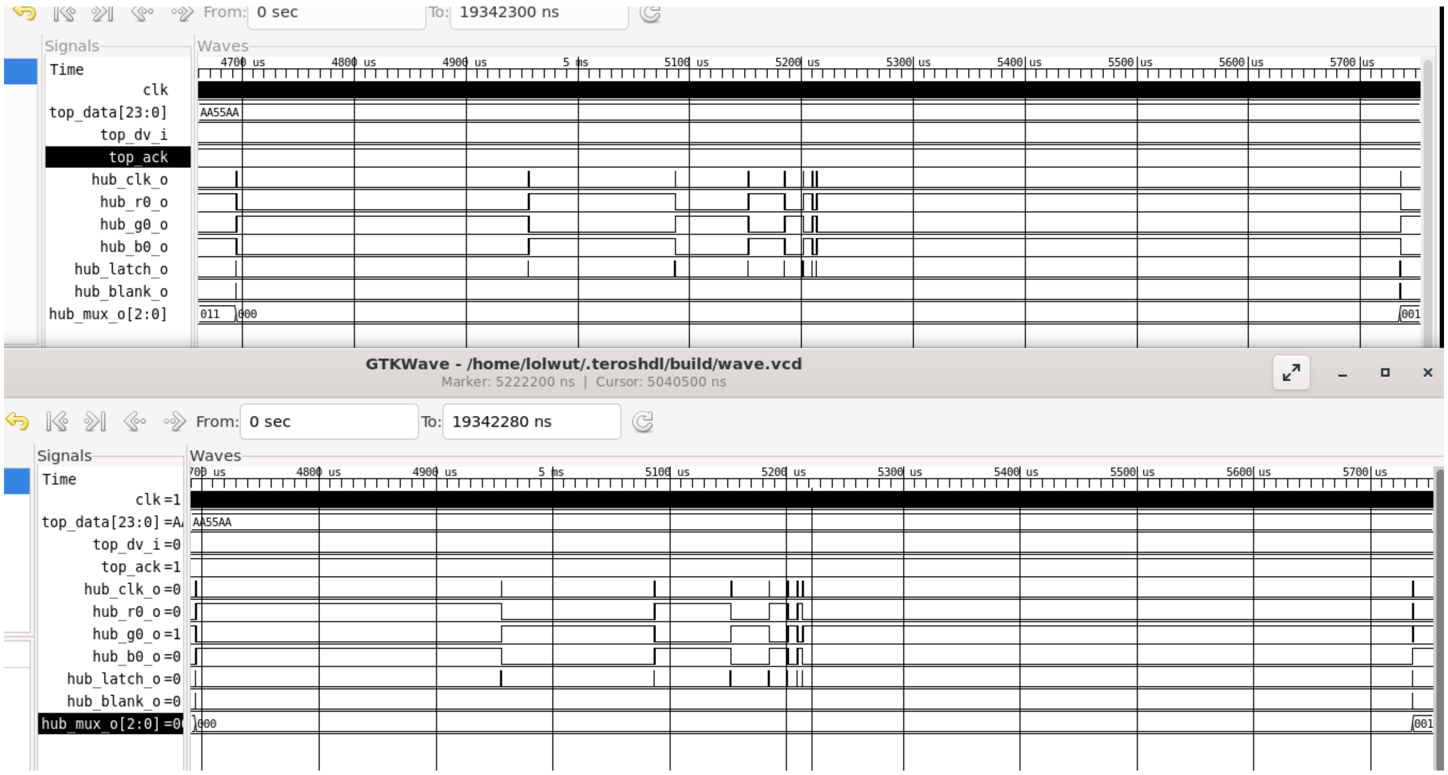

Figure 2: Output of one row of LED's, behavioral vs netlist.

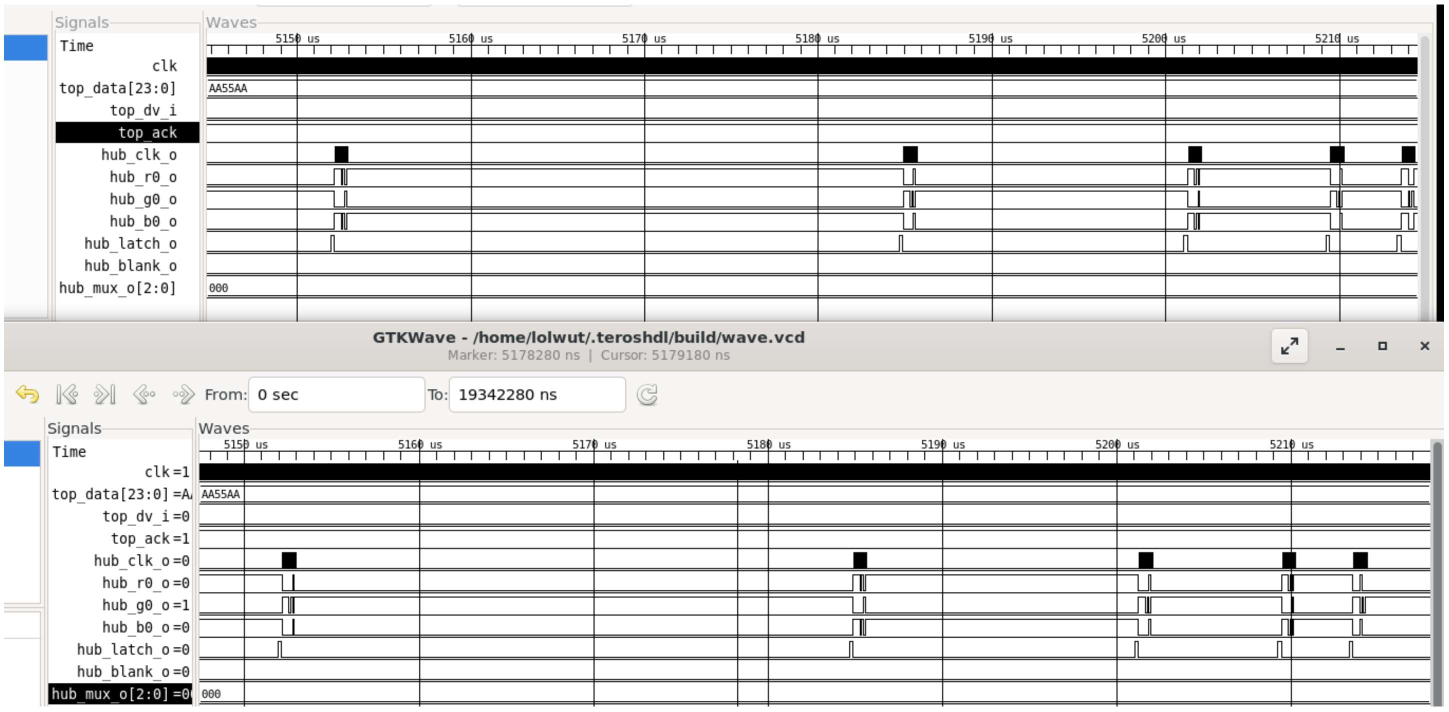

Figure 3: Zoom in on data transfer, Binary Coded Modulation currently being displayed. The wait time between clock and data activity is halved as expected.

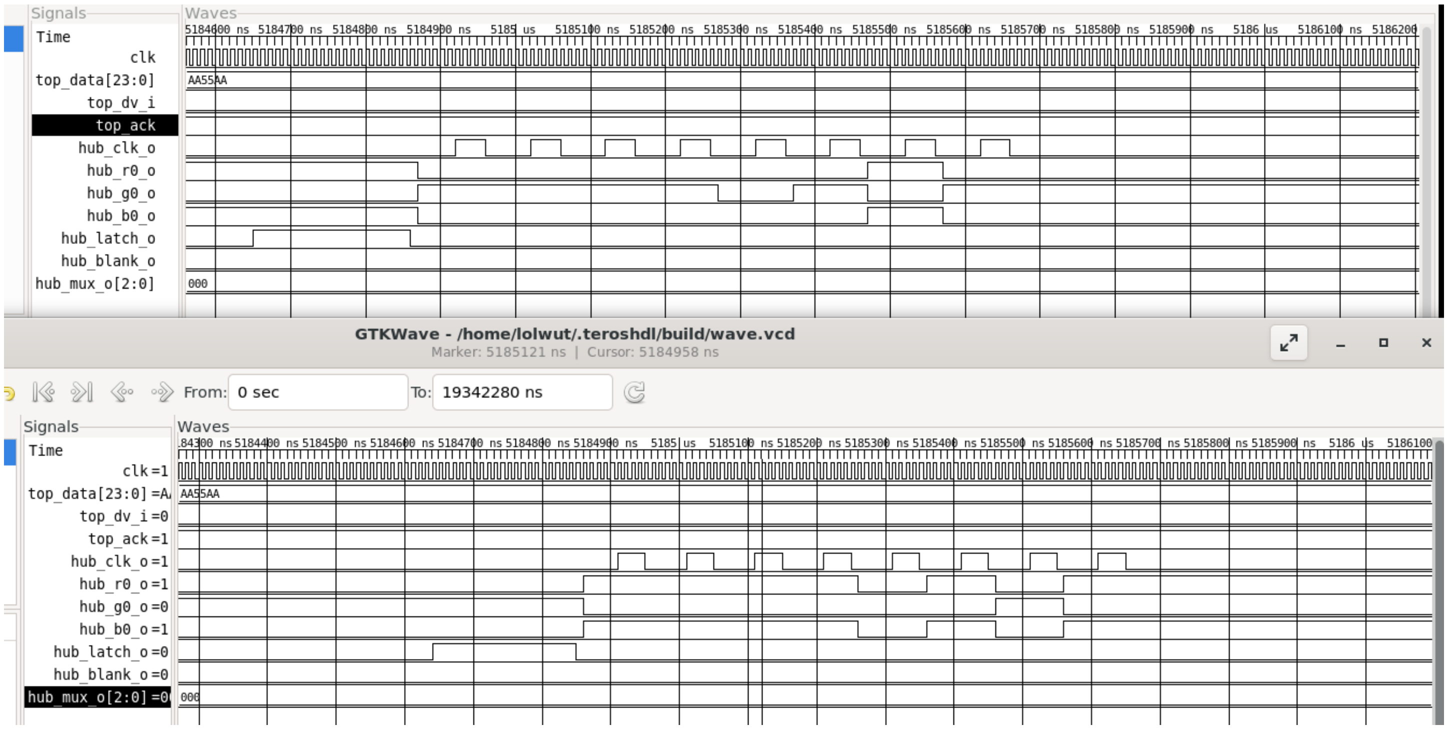

Figure 4: Zoom in on one clock and data transfer.