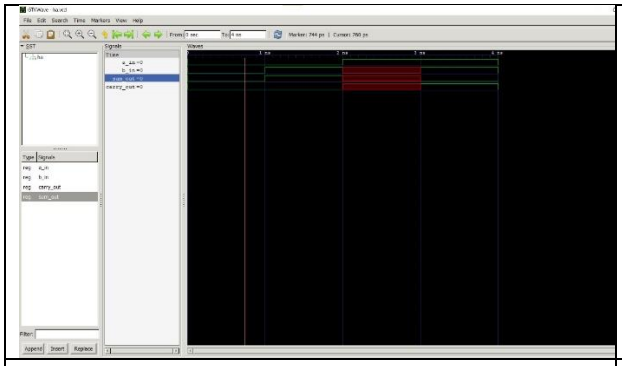
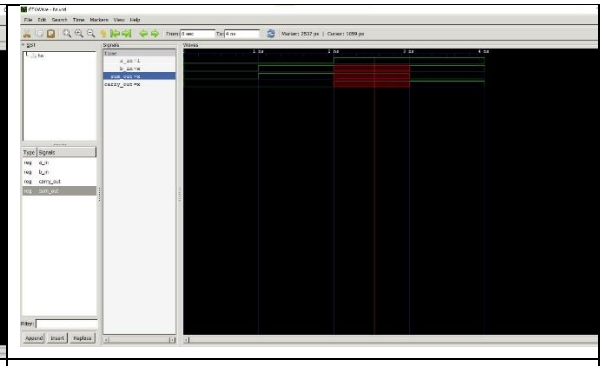
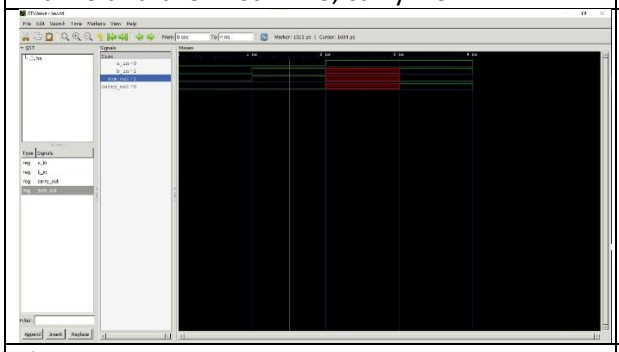
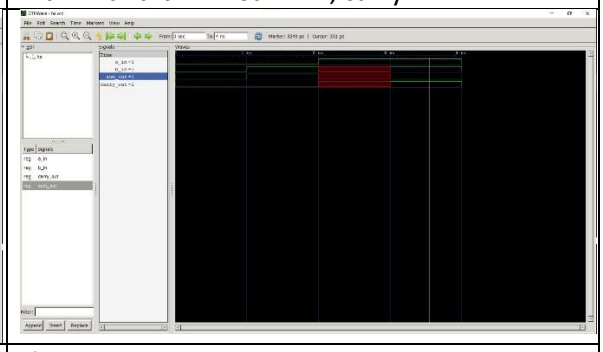
Computer Systems 414 12 May 2021

Walt Deyzel: 21750793

Half Adder:

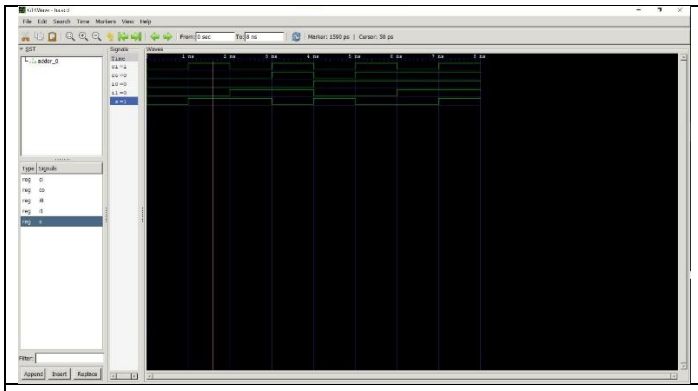
	
If a = 0 and b=0 -> sum = 0, carry = 0	If a = 1 and b=x -> sum = x, carry = x
	
If a = 0 and b=1 -> sum = 1, carry = 0	If a = 1 and b=1 -> sum = 0, carry = 1

Full Adder:

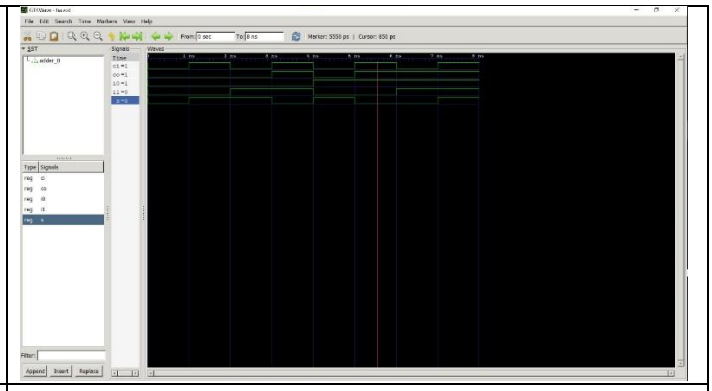
The **entity is boxed in orange** while the **architecture is boxed in yellow.** See code block below.

The block diagram of the full adder is on the left. The truth table is listed below. It is clear from comparing the truth table with the GTKwave output that they match.

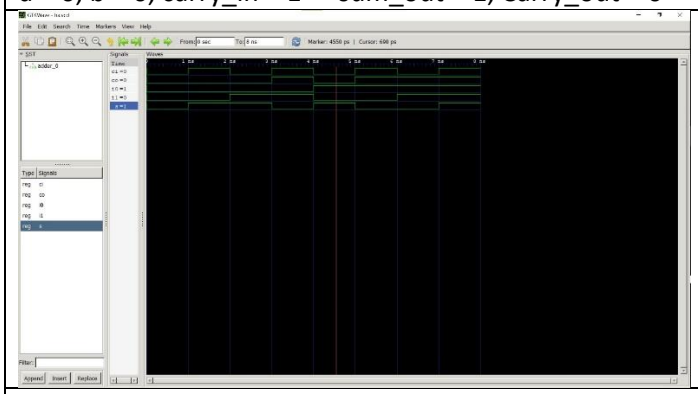
GTKwave output examples.



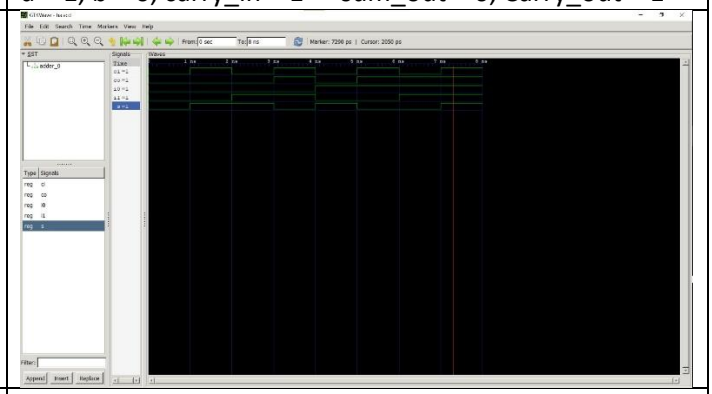
a = 0, b = 0, carry_in = 1 -> sum_out = 1, Carry_out = 0



a = 1, b = 0, carry_in = 1 -> sum_out = 0, Carry_out = 1



a = 1, b = 0, carry_in = 1 -> sum_out = 0, Carry_out = 1



a = 1, b = 1, carry_in = 1 -> sum_out = 1, Carry_out = 1

Input			Output	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig. 2 Truth table

https://electricalvoice.com/full-adder-truth-table-logic-diagram/

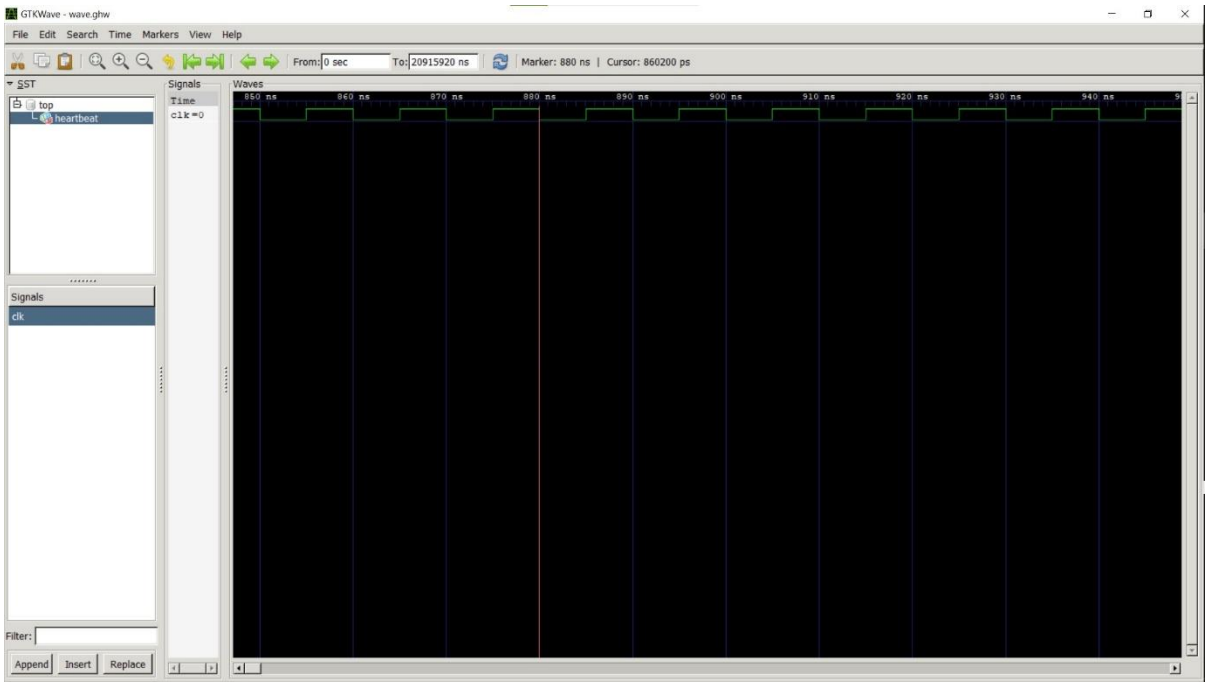
adder.vhdl

```vhdl
entity adder is
    -- `i0`, `i1`, and the carry-in `ci` are inputs of the adder.
    -- `s` is the sum output, `co` is the carry-out.
    port (i0, i1 : in bit; ci : in bit; s : out bit; co : out bit);
end adder;

architecture rtl of adder is
begin
    --  This full-adder architecture contains two concurrent assignments.
    --  Compute the sum.
    s <= i0 xor i1 xor ci;
    --  Compute the carry.
    co <= (i0 and i1) or (i0 and ci) or (i1 and ci);
end rtl;
```

Orange -> Entity; Yellow-> Architecture

Heartbeat:



heartbeat.vhdl

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

entity heartbeat is
  port ( clk: out std_logic);
end heartbeat;

architecture behaviour of heartbeat is
  constant clk_period : time := 10 ns;
begin
  -- Clock process definition
  clk_process: process
  begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
  end process;
end behaviour;
```

The **entity is boxed in orange** while the **architecture is boxed in yellow.**

The signal is held high for one half of the clock period and low for the other half creating a square wave.

Full Adder Test Bench

adder_tb.vhdl

```vhdl
-- A testbench has no ports.
entity adder_tb is
    end adder_tb;

    architecture behav of adder_tb is
      -- Declaration of the component that will be instantiated.
      component adder
        port (i0, i1 : in bit; ci : in bit; s : out bit; co : out bit);
      end component;

      -- Specifies which entity is bound with the component.
      for adder_0: adder use entity work.adder;
      signal i0, i1, ci, s, co : bit;
    begin
      -- Component instantiation.
      adder_0: adder port map (i0 => i0, i1 => i1, ci => ci, s => s, co => co);
      -- This process does the real job.
      process
        type pattern_type is record
          -- The inputs of the adder.
          i0, i1, ci : bit;
          -- The expected outputs of the adder.
          s, co : bit;
        end record;
        -- The patterns to apply.
        type pattern_array is array (natural range <>) of pattern_type;
        constant patterns : pattern_array :=
          (('0', '0', '0', '0', '0'),
           ('0', '0', '1', '1', '0'),
           ('0', '1', '0', '1', '0'),
           ('0', '1', '1', '0', '1'),
           ('1', '0', '0', '1', '0'),
           ('1', '0', '1', '0', '1'),
           ('1', '1', '0', '0', '1'),
           ('1', '1', '1', '1', '1'));
      begin
        -- Check each pattern.
        for i in patterns'range loop
          -- Set the inputs.
          i0 <= patterns(i).i0;
          i1 <= patterns(i).i1;
          ci <= patterns(i).ci;
          -- Wait for the results.
          wait for 1 ns;
          -- Check the outputs.
          assert s = patterns(i).s
            report "bad sum value" severity error;
          assert co = patterns(i).co
            report "bad carry out value" severity error;
        end loop;
        assert false report "end of test" severity note;
        -- Wait forever; this will finish the simulation.
        wait;
      end process;

    end behav;
```

Component