GUI Controls

Table 1: List of Controls in the main page

Controls	Function
1 – Upload file	When 'upload file' button is pressed, directory of the system is shown, and VHDL design file can be selected. If testbench file is selected, error message will be shown.
2 – Toggle page	Toggle between main page and truth table page
3 – Menu list for Input ports	Variable menu list shows the extracted input ports where user can select the input ports
4 – Type of Input	Signal, Reset, and Clock input type can be selected.
5 – Vector Information	Display vector information of the respective port with "STD_LOGIC" or "STD_LOGIC_VECTOR"
6 – Signal Timing Input	When signal type is selected, user can input the number of cycle and saved it. Single Bits, Vector Bits and Truth table signal input type can be selected, and 'timing details' button will request for input parameters correspond to the signal input type and number of cycles.
7 – Reset Timing Input	When reset type is selected, user can input the number of cycle and saved it. 'timing details' button will request input parameters based on number of cycles.
8 – Clock Timing Input	When clock type is selected, user can input clock timing, duty cycle and time metric correspond to the clock
9 – Saved and Generate	After all the parameters are inputted for given input port, 'saved' button is pressed to save the parameters. After all the all the input ports have been saved with parameters, 'generate' button is pressed to generate testbench. Testbench will be automatically added in the directory of where VHDL design file is uploaded.

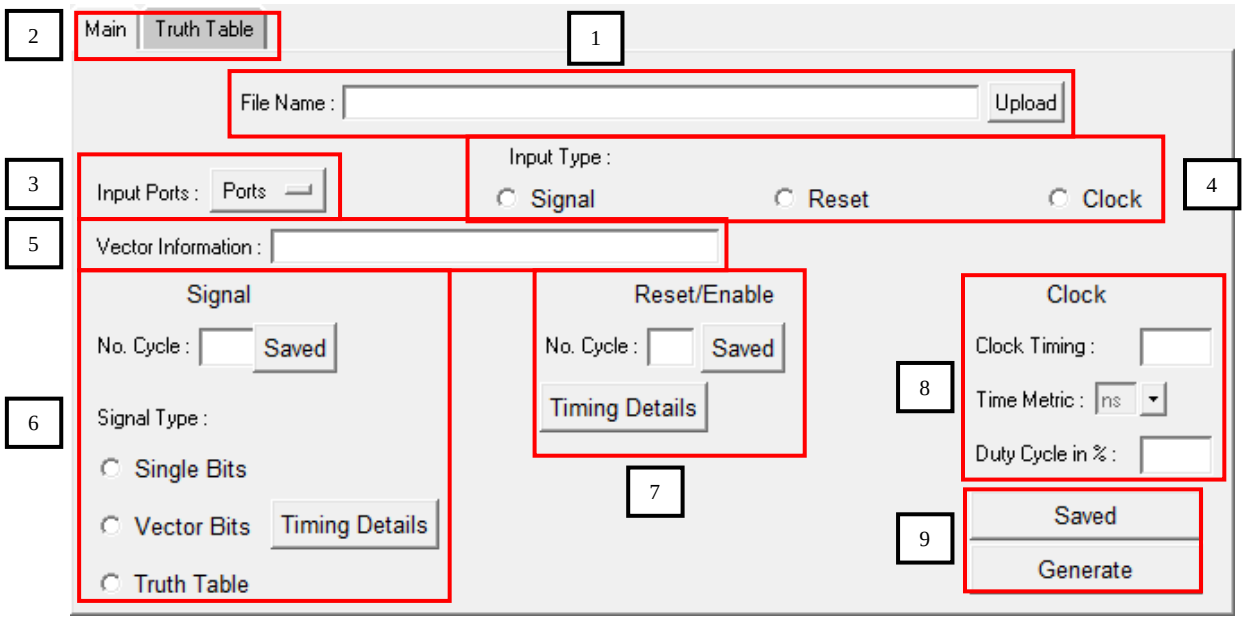
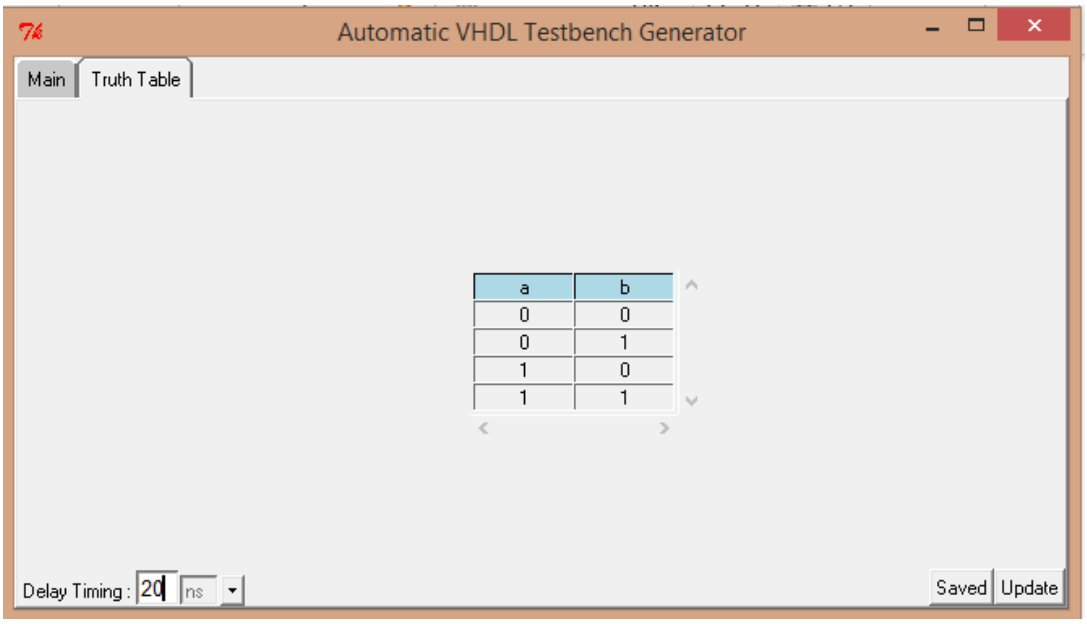


Figure 15: GUI main page

For truth table input, name of input port is saved to truth table variable. After all the desired truth table variable is saved, 'Truth Table' page can be toggled. 'update' button is pressed to update all the desired truth table variable into truth table. The input of truth table is automatically filled with all the possible combination for a truth table based on number of truth table variable. The input of truth table can be changed to 0 or 1 which allow flexibility in the design. Delay timing can be inputted at the bottom of GUI and truth table input can be saved. 'Generate' button in the main page can be pressed to generate testbench with truth table input.



Case Study

A. Asynchronous Design (Half-adder circuit)

Listing 1. Half-adder VHDL design file

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity half_adder is
6      Port ( a : in  STD_LOGIC;
7             b : in  STD_LOGIC;
8             sum : out  STD_LOGIC;
9             carry : out  STD_LOGIC);
10 end half_adder;
11
12 architecture Behavioral of half_adder is
13
14 begin
15
16    sum <= a xor b;
17    carry <= a and b;
18
19 end Behavioral;
```

As explained earlier, the user only needs to input VHDL design file, input type and parameters correspond to the input type like clock timing, duty cycle, bits data etc. The following are simple steps for user to generate VHDL testbench of half adder design using this tool.

1) Users need to upload the half_adder.vhd file shown in **Code Listing 1** from any local directory by pressing the 'upload' button in the GUI. The director of half_adder.vhd file will be shown in the text box.
2) User can select input ports via the menu list and select the input type via the check box. Parameters correspond to input type can be filled. In the case of half adder, inputs would include *a* and *b* with both having same types *signal* and *truth table*. Parameters are saved by pressing the 'Saved' button.

3) After all the input and their corresponding parameters are saved, the next step is to toggle to truth table page since truth table input type is chosen and pressed the 'update' button. Truth table input with all the possible combination is generated for input *a* and *b*. Delay timing of 20ns is inputted. 'saved' button is pressed to save truth table input and delay timing.
4) The final step is to press the 'generate' button back at the main page. The half adder testbench will be generated in the same directory the user inputted the half_adder.vhd file. 'half_adder_tb.vhd' testbench file is shown in **Code Listing 2**.

Listing 2. Generated test-bench for Half-adder VHDL design file

```
1  Library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4
5  -- Declare module entity. Declare module i
6  entity half_adder_tb is
7  end half_adder_tb;
8
9  -- Begin module architecture/code.
10 ARCHITECTURE behavior OF half_adder_tb IS
11
12 COMPONENT half_adder
13  PORT(
14 a : in  STD_LOGIC;
15    b : in  STD_LOGIC;
16    sum : out  STD_LOGIC;
17    carry : out  STD_LOGIC);
18
19 END COMPONENT;
20
21 -- Inputs & Outputs
22    signal tb_a :  STD_LOGIC;
23    signal tb_b :  STD_LOGIC;
24    signal tb_sum :  STD_LOGIC;
25    signal tb_carry :  STD_LOGIC;
26
27 -- *** Instantiate Constants ***
28 BEGIN
29
30 -- Instantiate the UUT module.
31 uut : half_adder
32 port map (
33    a  => tb_a,
34    b  => tb_b,
35    sum   => tb_sum,
36    carry    => tb_carry);
```

```
39  -- Insert Processes and code here.
40  -- Stimulus process
41  stim_proc: process
42  begin
43  wait for 20 ns;
44
45  tb_a <= '0';
46  tb_b <= '0';
47  wait for 20 ns;
48
49  tb_a <= '0';
50  tb_b <= '1';
51  wait for 20 ns;
52
53  tb_a <= '1';
54  tb_b <= '0';
55  wait for 20 ns;
56
57  tb_a <= '1';
58  tb_b <= '1';
59  wait;
60  end process;
61
62  END behavior; -- architecture
```

B. Synchronous Design (Clock divider Design)

Listing 3. Clock divider VHDL design file

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   Use ieee.std_logic_unsigned.all;
4
5   entity Counter is
6       Port ( Clk : in  STD_LOGIC;
7              Reset : in  STD_LOGIC;
8              clk_divider2 : out STD_LOGIC;
9              clk_divider4 : out STD_LOGIC;
10             countout : out  STD_LOGIC_VECTOR (3 downto 0));
11  end Counter;
12
13  architecture Behavioral of Counter is
14  Signal int_count: std_logic_vector(3 downto 0):= (others => '0');
15  begin
16     Process (Clk, reset)
17     begin
18           if (reset = '0') then
19              int_count <= (others => '0');
20           elsif rising_edge (clk) then
21              --if (int_count = "1001") then
22                 --int_count <= (others => '0');
23              --else
24                 int_count <= int_count + '1';
25              --end if;
26           end if;
27
28     end Process;
29     countout <= int_count;
30
31     clk_divider2 <= int_count(0);
32
33     clk_divider4 <= int_count(1);
34
35  end Behavioral;
```

Code listing 3 demonstrate code for clock divider VHDL design and Counter.vhd is uploaded to the automatic testbench generator. Parameters are inputted for *Clk* and *Reset* input ports are as follow:

Clk input port
Input type: Clock
Clock Timing: 20

Time Metric: ns
Duty Cycle: 50%

Reset input port
Input type: Reset
No. Cycle: 2
Timing Details:
Don't tick the box with "Timing based on Clock ports"
Time Metric: ns
Clock On Timing: 5
Clock Off Timing: 25
Clock On Timing: 100
Clock Off Timing: 5

Generated test-bench for clock divider is shown in **code listing 4**.

Listing 4. Generated test-bench for Clock Divider VHDL design file

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Counter_tb is
end Counter_tb;

-- Begin module architecture/code.
ARCHITECTURE behavior OF Counter_tb IS

COMPONENT Counter
 PORT(
Clk : in  STD_LOGIC;
   Reset : in  STD_LOGIC;
   clk_divider2 : out STD_LOGIC;
   clk_divider4 : out STD_LOGIC;
   countout : out  STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;

-- Inputs & Outputs
   signal tb_Clk :  STD_LOGIC;
   signal tb_Reset :  STD_LOGIC;
   signal tb_clk_divider2 : STD_LOGIC;
   signal tb_clk_divider4 : STD_LOGIC;
   signal tb_countout :  STD_LOGIC_VECTOR (3 downto 0);

-- *** Instantiate Constants ***
constant Clk_PERIOD : time := 20 ns;

BEGIN
-- Instantiate the UUT module.
uut : Counter
port map (
   Clk   => tb_Clk,
   Reset    => tb_Reset,
   clk_divider2   => tb_clk_divider2,
   clk_divider4   => tb_clk_divider4,
   countout    => tb_countout);
```

```
41  -- Generate necessary clocks.
42  Clk_process1: process
43  begin
44     tb_Clk <= '1';
45     wait for Clk_PERIOD*0.5;
46     tb_Clk <= '0';
47     wait for Clk_PERIOD*0.5;
48  end process;
49
50  -- Toggle the resets.
51  reset1: process
52  begin
53     tb_Reset <= '1';
54     wait for 5 ns;
55     tb_Reset <= '0';
56     wait for 25 ns;
57     tb_Reset <= '1';
58     wait for 100 ns;
59     tb_Reset <= '0';
60     wait for 5 ns;
61     wait;
62  end process;
63
64  END behavior; -- architecture
```

C. Finite-State Machine Implementation Design (Receiver of RS232 design)

Listing 5. Receiver of RS232 VHDL design file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity Rs232Rxd is
Port ( Reset : in  STD_LOGIC;
       Clock16x : in  STD_LOGIC;
       Rxd : in  STD_LOGIC;
       DataOut1 : out  STD_LOGIC_VECTOR (7 downto 0));
end Rs232Rxd;

architecture Rs232Rxd_Arch of Rs232Rxd is

   attribute enum_encoding: string;
   --state definitions
   type stateType is (stIdle, stData, stStop, stRxdCompleted);
   attribute enum_encoding of statetype: type is "00 01 11 10";

   signal presState: stateType;
   signal nextState: stateType;
   signal iReset, iRxd1, iRxd2: std_logic ;
   signal iClock1xEnable, iClock1x, iEnableDataOut: std_logic ;
   signal iClockDiv: std_logic_vector (3 downto 0) ;
   signal iDataOut1, iShiftRegister: std_logic_vector (7 downto 0) ;
   signal iNoBitsReceived: std_logic_vector (3 downto 0) ;
begin

   process (Clock16x)
   begin

      if Clock16x'event and Clock16x = '1' then
         if Reset = '1' or iReset = '1' then
            iRxd1 <= '1';
            iRxd2 <= '1';
            iClock1xEnable <= '0';
            iClockDiv <= (others=>'0');
         else
            iRxd1 <= Rxd;
            iRxd2 <= iRxd1;
               if iClock1xEnable = '1' then
                  iClockDiv <= iClockDiv + '1';
               elsif iRxd1 = '0' and iRxd2 = '1' then
                  iClock1xEnable <='1';
               end if;
         end if;
      end if;

   end process;
   iClock1x <= iClockDiv(3);

   process (iClock1xEnable, iClock1x)
   begin

      if iClock1xEnable = '0' then
         iNoBitsReceived <= (others=>'0');
         iShiftRegister <= (others=>'0');
         presState <= stIdle;
      elsif iClock1x'event and iClock1x = '1' then
         iNoBitsReceived <= iNoBitsReceived + '1';
         presState <= nextState;
            if iEnableDataOut = '1' then
               iDataOut1 <= iShiftRegister;
            else
               iShiftRegister <= Rxd & iShiftRegister(7 downto 1);
            end if;
      end if;

   end process;
   DataOut1 <= iDataOut1;
```

```
73    process (presState, iClock1xEnable, iNoBitsReceived)
74    begin
75       --signal defaults
76       iReset <= '0';
77       iEnableDataOut <= '0';
78       case presState is
79          when stIdle =>
80             if iClock1xEnable = '1' then
81                nextState <= stData;
82             else
83                nextState <= stIdle;
84             end if;
85
86          when stData =>
87             if iNoBitsReceived = "1001" then
88                iEnableDataOut <= '1';
89                nextState <= stStop;
90             else
91                iEnableDataOut <= '0';
92                nextState <= stData;
93             end if;
94
95          when stStop =>
96             nextState <= stRxdCompleted;
97
98          when stRxdCompleted =>
99             iReset <= '1';
100            nextState <= stIdle;
101      end case;
102
103   end process;
104
105 end Rs232Rxd_Arch;
```

Code listing 5 demonstrate code for receiver of Rs232 design and RS232Txd.vhd is uploaded to the automatic testbench generator. Parameters are inputted for *Send*, *DataIn*, *Reset* and *Clock16x* input ports are as follow:

Clock16x input port

Input type: Clock
Clock Timing: 6.5
Time Metric: us
Duty Cycle: 50%

Reset input port

Input type: Reset
No. Cycle: 1

Timing Details:
Tick the box “Timing based on clock ports”
Choose Clock16x as clock ports
Ignore Time Metric
Clock on Timing: 2
Clock Off Timing: 1

Rxd input port

Input type: Signal
No. Cycle: 2
Signal type: Single Bits

Timing Details:
Tick the box “Timing based on clock ports”
Choose Clock16x as clock ports
Ignore Time Metric
Single Bits: 1
Delay between each bit: 0
Delay Timing between each data: 5.5
Signal Bits: 0010001101
Delay between each bit: 16

Generated test-bench for receiver of RS232 is shown in **code listing 6**.

Listing 6. Generated test-bench for receiver of RS232 VHDL design file

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;


-- Declare module entity. Declare module inputs, inouts, and outputs.
entity tb_Rs232Rxd is
end tb_Rs232Rxd;

-- Begin module architecture/code.
ARCHITECTURE behavior OF tb_Rs232Rxd IS

COMPONENT Rs232Rxd
 PORT(
Reset : in STD_LOGIC;
   Clock16x : in STD_LOGIC;
   Rxd : in STD_LOGIC;
   DataOut1 : out STD_LOGIC_VECTOR (7 downto 0) );

END COMPONENT;

-- Inputs & Outputs
   signal tb_Reset : STD_LOGIC;
   signal tb_Clock16x : STD_LOGIC;
   signal tb_Rxd : STD_LOGIC;
   signal tb_DataOut1 : STD_LOGIC_VECTOR (7 downto 0) ;

-- Local parameter, wire, and register declarations go here.
-- N/A
-- general signals
-- N/A

-- *** Instantiate Constants ***
constant Clock16x_PERIOD : time := 6.5 us;
BEGIN

-- Instantiate the UUT module.
uut : Rs232Rxd
port map (
   Reset    => tb_Reset,
   Clock16x    => tb_Clock16x,
   Rxd   => tb_Rxd,
   DataOut1    => tb_DataOut1);


-- Generate necessary clocks.
Clk_process1: process
begin
   tb_Clock16x <= '1';
   wait for Clock16x_PERIOD*0.5;
   tb_Clock16x <= '0';
   wait for Clock16x_PERIOD*0.5;
end process;

-- Toggle the resets.
reset1: process
begin
   tb_Reset <= '1';
   wait for 2*Clock16x_PERIOD;
   tb_Reset <= '0';
   wait for 1*Clock16x_PERIOD;
   wait;
end process;
```

```
65  -- Insert Processes and code here.
66  -- Stimulus process1
67  Rxd: process
68  begin
69     tb_Rxd <= '1';
70     wait for 0*Clock16x_PERIOD;
71
72     wait for 5.5*Clock16x_PERIOD;
73
74     tb_Rxd <= '0';
75     wait for 16*Clock16x_PERIOD;
76     tb_Rxd <= '0';
77     wait for 16*Clock16x_PERIOD;
78     tb_Rxd <= '1';
79     wait for 16*Clock16x_PERIOD;
80     tb_Rxd <= '0';
81     wait for 16*Clock16x_PERIOD;
82     tb_Rxd <= '0';
83     wait for 16*Clock16x_PERIOD;
84     tb_Rxd <= '0';
85     wait for 16*Clock16x_PERIOD;
86     tb_Rxd <= '0';
87     wait for 16*Clock16x_PERIOD;
88     tb_Rxd <= '1';
89     wait for 16*Clock16x_PERIOD;
90     tb_Rxd <= '1';
91     wait for 16*Clock16x_PERIOD;
92     tb_Rxd <= '0';
93     wait for 16*Clock16x_PERIOD;
94     tb_Rxd <= '1';
95     wait for 16*Clock16x_PERIOD;
96  wait;
97  end process;
98
99  END behavior; -- architecture
```