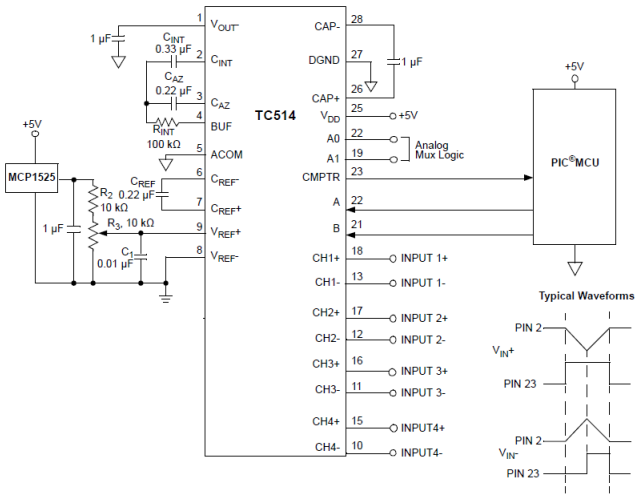
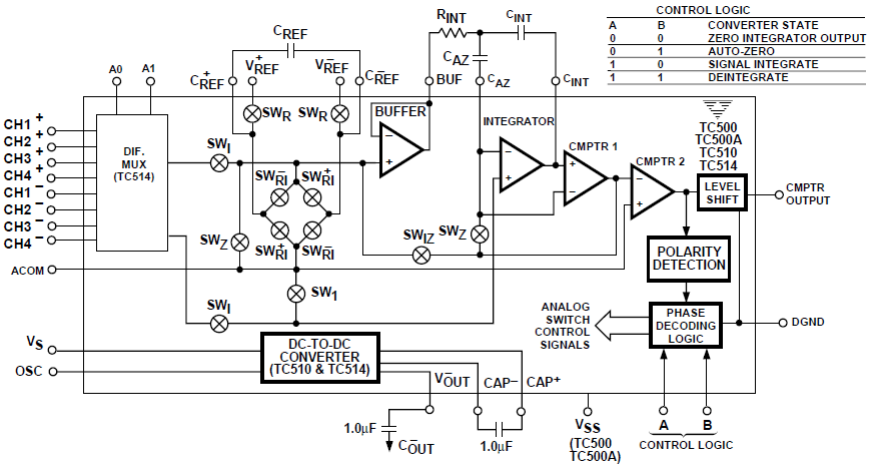
Controller for TC514 Precision Analog Front End

Ken Short

TC514 Front End Design Example



Older TelCom TC514 Diagram with Correct SW_{RI} Labeling



Conversion Phases

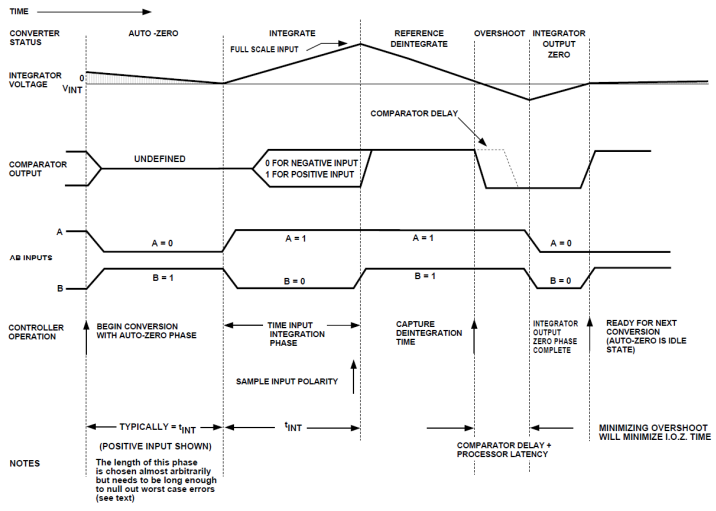
- There are four phases that make up a single conversion of an analog input to a binary output
  - Auto Zero
  - Input Signal Integration
  - Reference Voltage Deintegration
  - Integrator Output Zero

Switch Positions vs. Conversion Phase

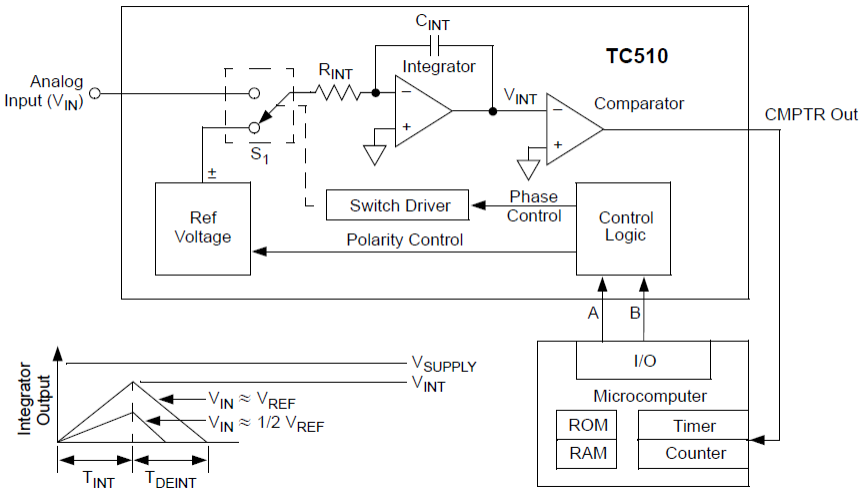
Conversion Phase	SW_I	SW_R+	SW_R-	SW_Z	SW_R	SW_1	SW_{IZ}
Auto-zero (A = 0, B = 1)	—	—	—	Closed	Closed	Closed	—
Input Signal Integration (A = 1, B = 0)	Closed	—	—	—	—	—	—
Reference Voltage De-integration (A =1, B = 1)	—	* Closed	—	—	—	Closed	—
Integrator Output Zero (A = 0, B = 0)	—	—	—	—	Closed	Closed	Closed

* Assumes a positive polarity input signal. SW^{-}_{RI} would be closed for a negative input signal.

Typical System Timing



Basic Dual-Slope Analog-to-Digital Converter



Relationship Between Integrate and Deintegrate Phases

$$\frac{1}{R_{INT}\ C_{INT}} \int_0^{t_{INT}} V_{IN}(t)\ dt = \frac{V_{REF}\ t_{DEINT}}{R_{INT}\ C_{INT}}$$

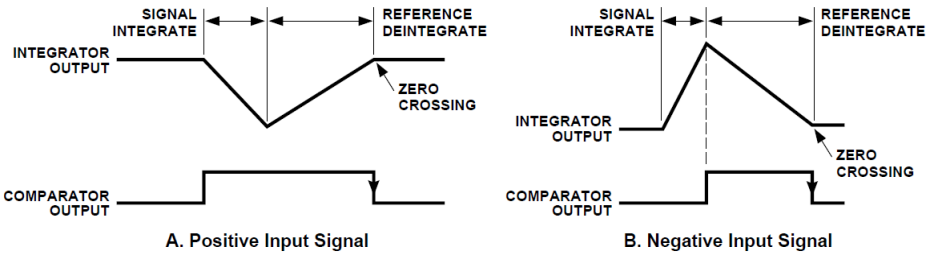
where:

V_{REF} = Reference Voltage
t_{INT} = Signal Integration time (fixed)
t_{DEINT} = Reference Voltage Integration time (variable)

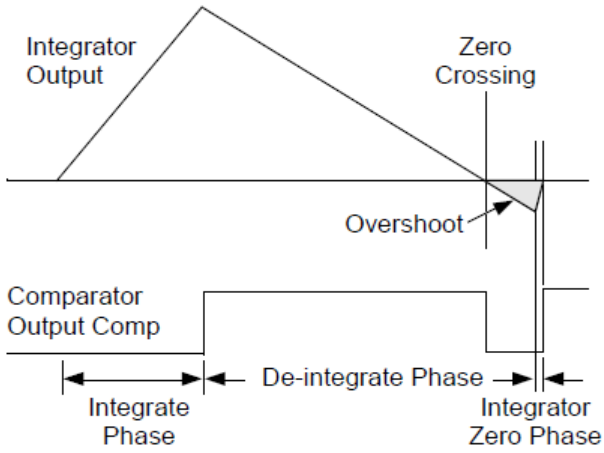
For a constant V_{IN}:

$$V_{IN} = V_{REF}\ \frac{t_{DEINT}}{t_{INT}}$$

Comparator Output



Integrator Output Zero Phase



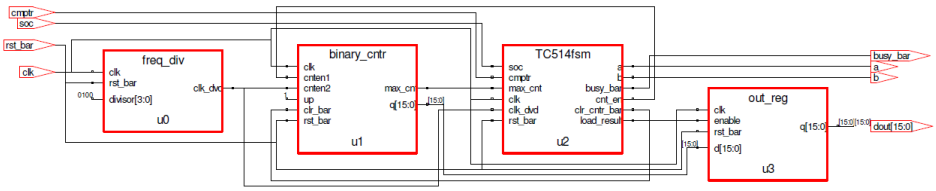
A and B Phase Control

A	B	Phase	Purpose	Duration
0	1	auto zero	correct for offset voltages	2**16 clocks minimum
1	0	signal integrate	integrate unknown input voltage	2**16 clocks exactly
1	1	reference deintegrate	integrate - Vref	until neg. edge of CMPTR
0	0	integrator zero	bring the integrator's output to 0	until pos. edge of CMPTR

Measured Voltage vs Count

$$\text{Vunknown} = \frac{\text{deintegrate count}}{2^{16}} \times \text{Vref}$$

System Block Diagram



Frequency Divider

```
entity freq_div is
    port (
        clk : in std_logic;          -- system clock
        rst_bar: in std_logic;       -- synchronous reset
        divisor: in std_logic_vector(3 downto 0);    -- d
        clk_dvd: out std_logic);     -- output
end freq_div;
```

Binary Counter

```
entity binary_cntr is
    generic (n : integer := 16);
    port (clk : in std_logic;        -- system clock
        cnten1 : in std_logic;       -- acitve high count enable
        cnten2 : in std_logic;       -- acitve high count enable
        up : in std_logic;           -- count direction
        clr_bar : in std_logic;      -- synchrounous counter clear
        rst_bar: in std_logic;       -- synchronous reset
        q: out std_logic_vector (n-1 downto 0); -- count
        max_cnt: out std_logic);     -- maximum count indication
end binary_cntr;
```

4/18/2022 © Copyright 2022 Kenneth L. Short 15

Out Register

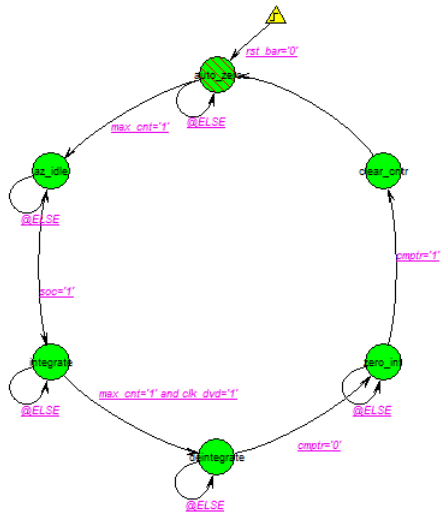
```
entity out_reg is
    generic (n : integer := 16);
    port (
        clk: in std_logic;       -- system clock
        enable: in std_logic;    -- parallel load enable
        rst_bar: in std_logic;   -- synchronous reset
        d: in std_logic_vector (n-1 downto 0);   -- data in
        q: out std_logic_vector (n-1 downto 0)   -- data out
        );
end out_reg;
```

4/18/2022 © Copyright 2022 Kenneth L. Short 16

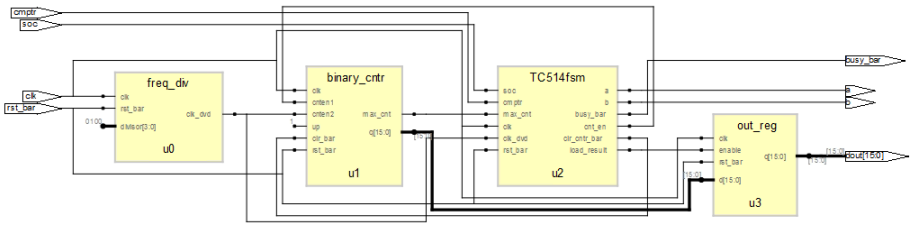
TC514 Controller FSM

```
entity TC514fsm is
    port (
        soc : in std_logic;          -- start conversion control input
        cmptr : in std_logic;        -- TC 514 comparator status input
        max_cnt : in std_logic;      -- maximum count status input
        clk : in std_logic;          -- system clock
        clk_dvd : in std_logic;      -- clock divided down
        rst_bar: in std_logic;       -- synchronous reset
        a  : out std_logic;          -- conversion phase control
        b : out std_logic;           -- conversion phase control
        busy_bar : out std_logic;    -- active low busy status
        cnt_en : out std_logic;      -- counter enable control to counter
        clr_cntr_bar : out std_logic;   -- signal to clear counter
        load_result : out std_logic);   -- load enable
end;
```


State Diagram Generated From Code by Code2Graphics



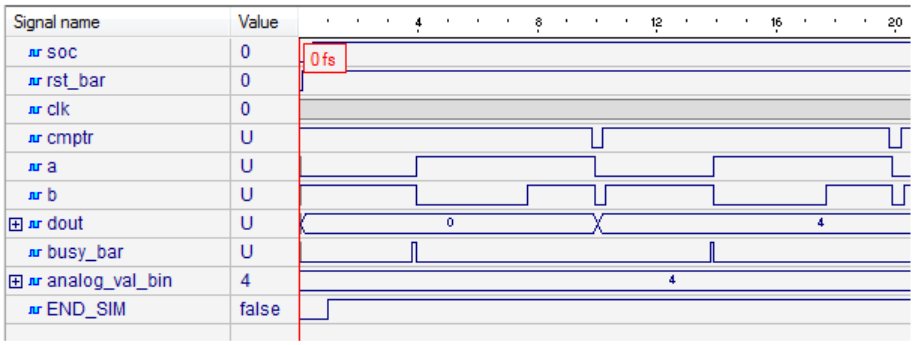
TC514cntrl: Top-Level Block Diagram



Use of Generics in This Design

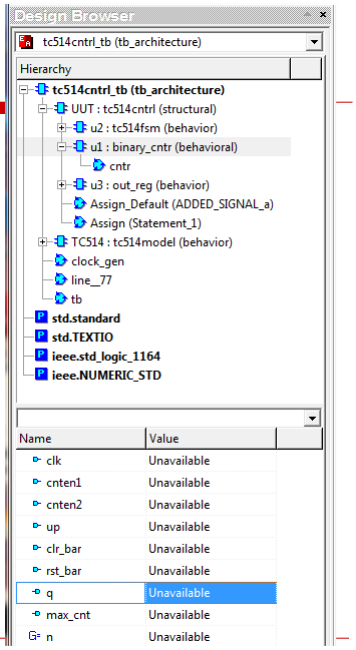
- Appropriate entities use a generic *n* to specify the width of some of the design registers.
- This allows the resolution of the analog-to-digital converter to be changed in one place to change the entire design.
- Use of a small value of *n* is particularly useful during simulation.

Using Generic in Simulation, with n = 4



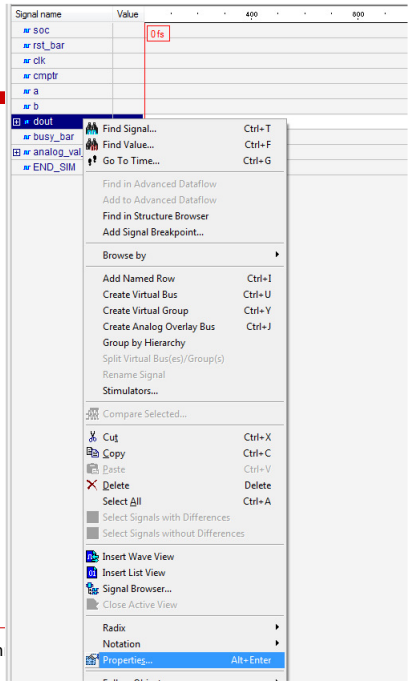
Looking at Internal Signals and Variables

- With simulation we can look at internal signals and variables in component entities



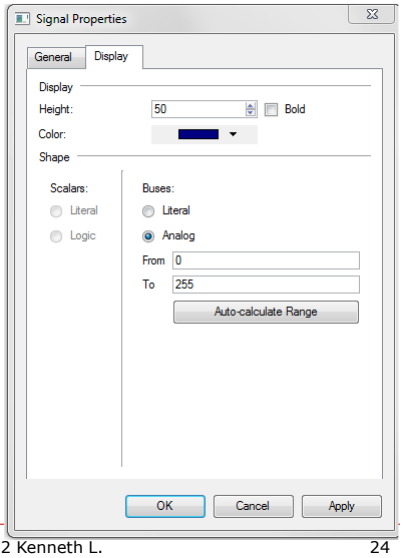
Looking at the Binary Counter's Value

- Add signal q from the tc514cntrl.
- Right click on q and select properties.

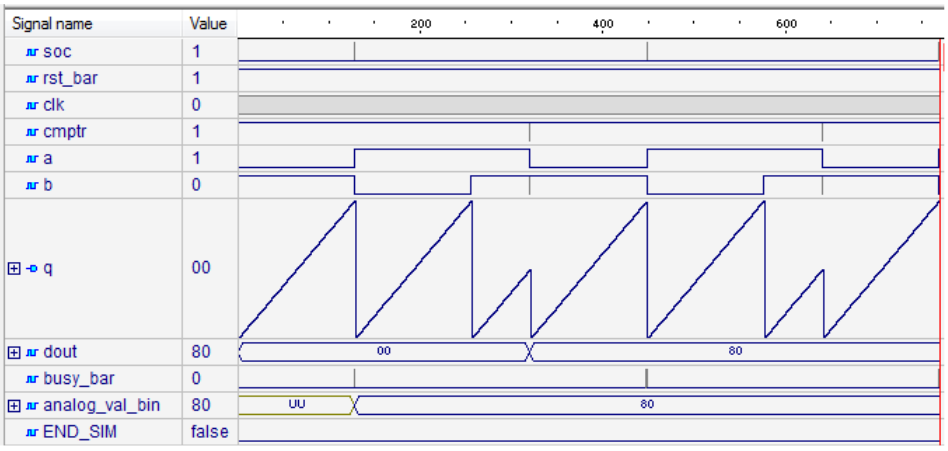


Displaying a Vector as an Analog Value

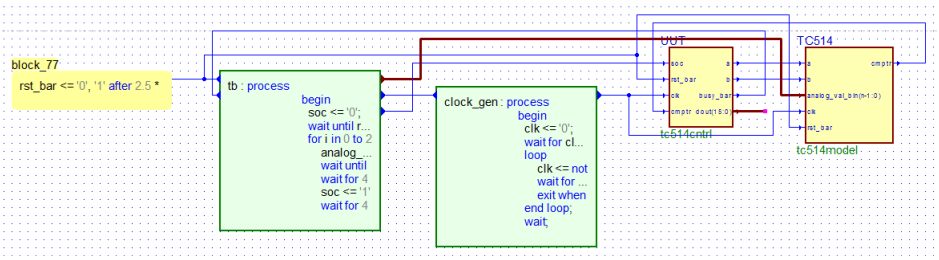
- In the properties box select the Display tab.
- Set Height to 100.
- Select Analog for Buses



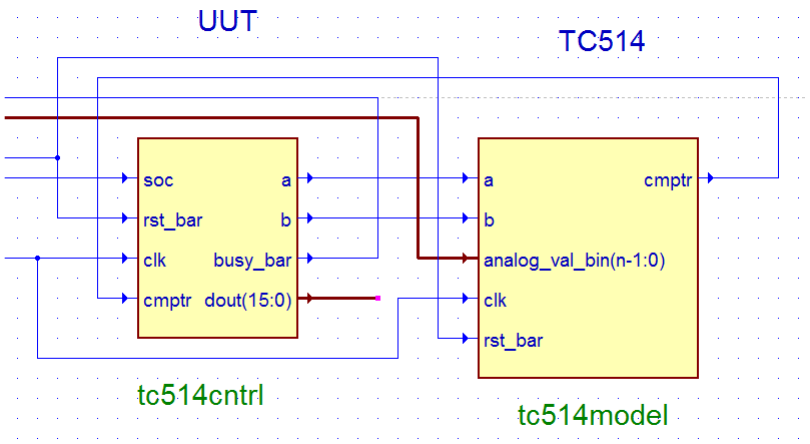
q Vector Displayed as Analog Signal (n = 8)



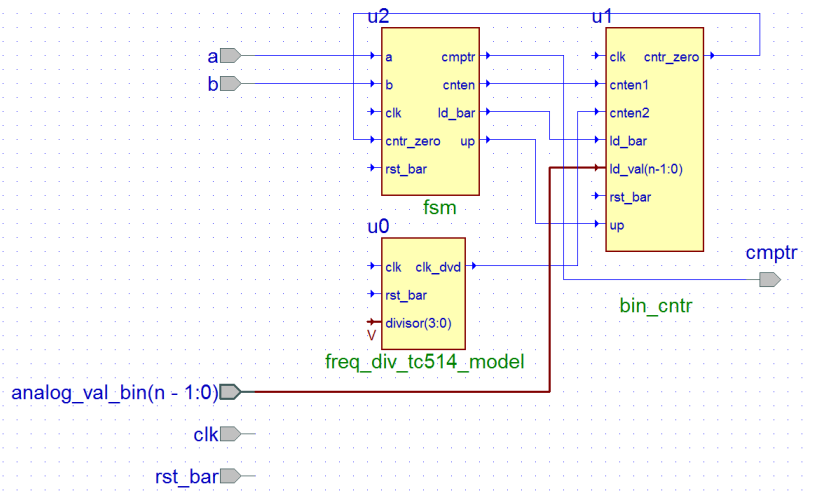
Basics for a Self-Checking Testbench



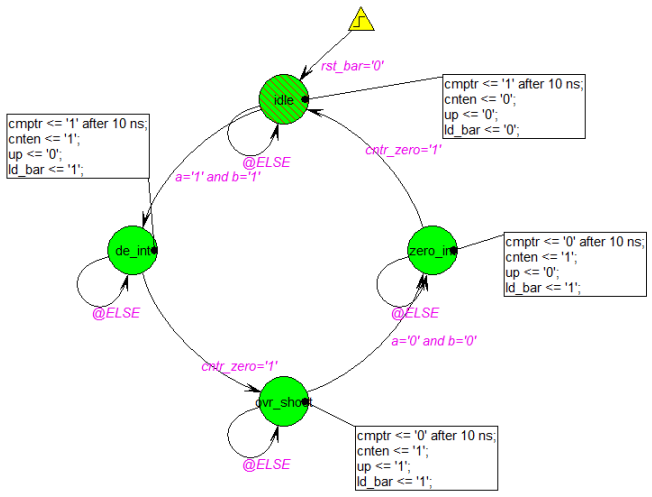
Testbench's TC514 Model



TC514 Model Diagram



State Diagram for TC514 Model FSM



State Assignment Approaches for Moore FSMs

```
-- to automtically create state assignment the following signal declarati
-- and constant declarations must be commented out and states enumerated:

-- (load_result, clr_cntr_bar, cnt_en, busy_bar, a, b) <= present_state;

    signal present_state, next_state : std_logic_vector(5 downto 0);
    constant auto_zero: std_logic_vector (5 downto 0) := "011001";
    constant az_idle: std_logic_vector (5 downto 0) := "000101";
    constant integrate: std_logic_vector (5 downto 0) := "011010";
    constant deintegrate: std_logic_vector (5 downto 0) := "011011";
    constant zero_int: std_logic_vector (5 downto 0) := "110000";
    constant clear_cntr: std_logic_vector (5 downto 0) := "000001";

-- to automtically create state diagram the following type declaration
-- and signal declaration must be uncommented:

-- type state is (auto_zero, az_idle, integrate, deintegrate, zero_int,
--  clear_cntr);
-- signal present_state, next_state : state;
```

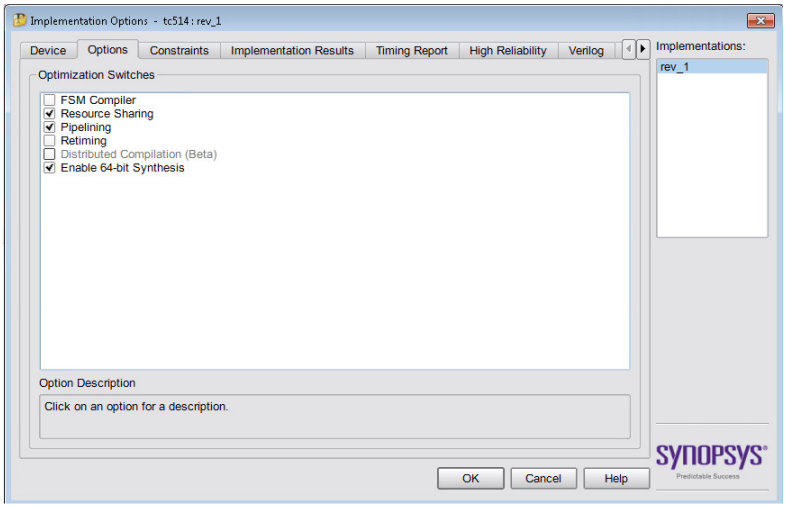

What is the Synplify Symbolic FSM Compiler

- The Symbolic FSM Compiler is an advanced state machine optimizer, which automatically recognizes state machines in your design and optimizes them.
- Unlike other synthesis tools that treat state machines as regular logic, the FSM Compiler extracts the state machines as symbolic graphs, and then optimizes them by re-encoding the state representations and generating a better logic optimization starting point for the state machines.
- The FSM Explorer uses the state machines extracted by the FSM Compiler when it explores different encoding styles.
- The FSM Explorer option is only available in the Synplify Pro and Synplify Premier tools.

State Assignment

- If your design does not specify state assignment, the Synplifier synthesizer contains a feature called FSM Compiler that identifies FSMs in your description and optimizes their state assignments based on the target PLD
- If you specified state assignment in your FSM description, make sure Synplify's FSM Compiler is OFF when you run the synthesis.

Turning OFF FSM Compiler



4/18/2022 © Copyright 2022 Kenneth L. Short 33