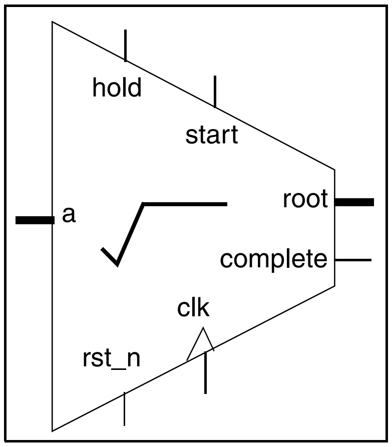
DW_sqrt_seq

Sequential Square Root

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized word length
- Parameterized number of clock cycles
- Unsigned and signed (two's complement) square roots
- Registered or un-registered inputs and outputs
- Provides minPower benefits with the DesignWare-LP license. (Get the minPower version of this datasheet.)



Description

DW_sqrt_seq is a sequential square root designed for low area, area-time trade-off, or high frequency (small cycle time) applications. Note that data input is taken as absolute value. Two's complement input is converted into unsigned magnitude. Output is unsigned (positive).

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Clock
rst_n	1 bit	Input	Reset, active low
hold	1 bit	Input	Hold current operation (=1)
start	1 bit	Input	Start operation (=1). A new operation is started by setting start = 1 for one clock cycle.
a	*width* bit(s)	Input	Radicand
complete	1 bit	Output	Operation completed (=1)
root	(*width* +1)/2 bit(s)	Output	Square root

Table 1-2 Parameter Description

Parameter	Values	Description
width	≥ 6	Word length of a
tc_mode	0 or 1 Default: 0	Two's complement control 0 = unsigned 1 = two's complement
num_cyc[a]	3 to int((*width*+1)/2 Default: 3	User-defined number of clock cycles to produce a valid result. The real number of clock cycles depends on various parameters and is given in Table 1-6 on page 4 and the topic titled "Formula" on page 4.
rst_mode	0 or 1 Default: 0	Reset mode 0 = asynchronous reset 1 = synchronous reset
input_mode[b]	0 or 1 Default: 1	Registered inputs 0 = no 1 = yes
output_mode	0 or 1 Default: 1	Registered outputs 0 = no 1 = yes
early_start	0 or 1 Default: 0	Computation start 0 = start computation in the second cycle 1 = start computation in the first cycle See Table 1-6 on page 4 for the dependency of *early_start* on *input_mode*.

a. Note that the num_cyc specification indicates the actual throughput of the device. That is, if a new input is driven before the num_cyc number of cycles are complete, the results are undetermined.
b. When configured with the parameter *input_mode* set to '0', input 'a' MUST be held constant from the time `start` is asserted until `complete` has gone high to signal completion of the calculation. Conversely, if a configuration with the parameter *input_mode* set to '1' is used, the 'a' input will be captured when `start` is high and otherwise ignored.

Table 1-3 Synthesis Implementations

Implementation	Function	License Feature Required
cpa	Carry-propagate adder synthesis model	DesignWare

Table 1-4 Simulation Models

Model[a]	Function
DW03.DW_SQRT_SEQ_CFG_SIM	Design unit name for VHDL simulation
dw/dw03/src/DW_sqrt_seq_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_sqrt_seq.v	Verilog simulation model source code

a. Note that during the computation phase (after start and before complete is asserted), the simulation models output X values and therefore cannot be used as a compare for gate-level simulations.

Table 1-5 Operation Truth Table

start	hold	Operation
0	0	Idle or Running
0	1	Hold
1	X	Start

DW_sqrt_seq computes the integer square root of radicand `a` in a user-defined number of clock cycles (*num_cyc*). As long as `start`=1 the square root operation is in the initialization state. Once `start`=0, the calculation begins followed by valid output flagged when `complete` =1 or an intervening setting of `start` to 1.The square root operation is stalled when `hold`=1. For theory of square root operation, refer to the datasheet DW_sqrt.

The parameter *tc_mode* determines whether the data of input (`a`) is interpreted as unsigned (*tc_mode*=*0*) or two's complement (*tc_mode*=*1*) number. The input is converted into unsigned absolute value for calculation of square root.

The internal registers can either have an asynchronous (*rst_mode* =*0*) or synchronous reset (*rst_mode* = *1*) that is connected to the reset signal `rst_n`.

After reset conditions are released (`rst_n =1`) there are not restrictions on when start can be set to 1 and then to 0. However, if `start` is set to 0 immediately after `rst_n` goes to 1 and `start`=0 continues through the first *num_cyc* clock cycles, then `complete` will go to 1. This first `complete` =1 when no start is initiated following reset may yield invalid results and should be disregarded.

The parameter *input_mode* determines whether the inputs are to be registered inside DW_sqrt_seq (*input_mode*=*1*) or not (*input_mode*=*0*). If configured without input registers (*input_mode*=*0*), then the logic that drives input `a` must hold the input value constant for the entire time it takes to calculate the result (from the cycle before `start` drops until `complete` goes high). When configured with input registers (*input_mode*=*1*) inputs `a` is captured when `start` is high and ignored until `start` goes high again.

Note When configured with no input registers, changes on input `a` while `complete` is low (calculation cycle) will produce unpredictable output values. Simulation models will produce unknown output values (Xs) and post an error message indicating the instance that violated this rule and the simulation time when the violation was detected.

The parameter *output_mode* determines whether the outputs are registered (*output_mode* =*1*) or not (*output_mode*=*0*). When the parameter *early_start*=*1*, computation starts immediately after setting `start` to 1. This saves one extra cycle to store the data (*early_start*=*0*), but feeds the inputs directly into the components critical path. Table 1-6 on page 4 shows the *input_mode*, *output_mode*, and *early_start* parameter combinations and corresponding actual number of cycles required to perform an operation.

Table 1-6 Actual Cycles Based on *input_mode*, *output_mode*, and *early_start*

input_mode	output_mode	early_start	Actual Number of Cycles
0	0	0	*num_cyc*–2
0	0	1	Invalid parameter setting
0	1	0	*num_cyc*–1
0	1	1	Invalid parameter setting
1	0	0	*num_cyc*–1
1	0	1	*num_cyc*–2
1	1	0	num_cyc
1	1	1	*num_cyc*–1

Note that the num_cyc specification indicates the actual throughput of the device. That is, if a new input is driven before the num_cyc number of cycles are complete, the results are undetermined.

Formula

The following formula describes the number of radicand bits processed per cycle:

bits processed per cycle = `ceil (`*width*/*num_cyc*`)`

where:

width is the bit width of the radicand, and
num_cyc is the number of clock cycles required for square root.

The actual number of clock cycles required by a computation are calculated using the following formula:

actual number of cycles = `num_cyc-(1-output_mode)-(1-input_mode)-early_start`

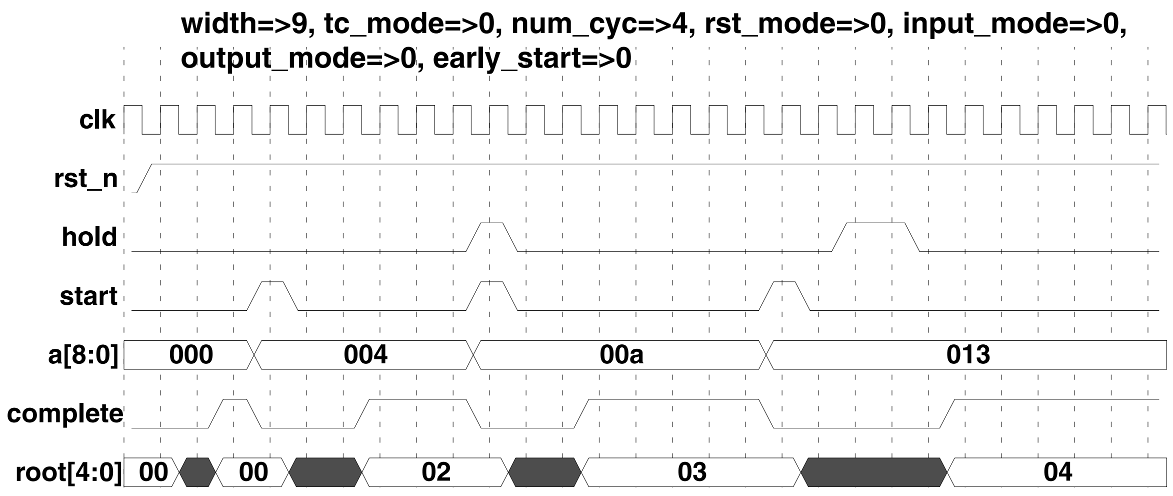
Note that there is a restriction of the relationship between *width* and *num_cyc* such that `num_cyc <= int((width+1)/2)`. In other words, *num_cyc* cannot exceed 1/2 of width.

Timing Waveforms

The following timing waveforms show a 9-bit unsigned sequential square root for specific inputs of `hold` and `start` and their corresponding outputs. The parameter settings for each simulation are shown at the top of each figure. When `hold`=1 and `start`=0, the result is delayed by the same number of clock cycles for which `hold` is 1. For example, if `hold`=1 for two clock cycles, then the result is delayed by two clock cycles.

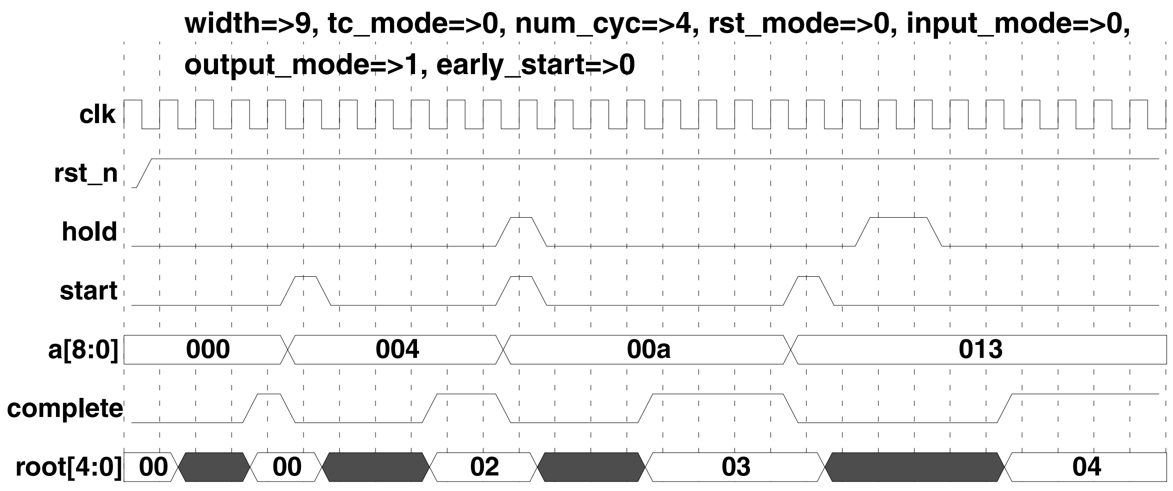
For the parameter settings shown in Figure 1-1, Table 1-6 on page 4 specifies that the result is produced after two cycles. However, data is available on the rising edge of the fourth clock following the clock that asserts the `start` signal.

Figure 1-1 Simulation Waveform 1


width=>9, tc_mode=>0, num_cyc=>4, rst_mode=>0, input_mode=>0, output_mode=>0, early_start=>0
clk
rst_n
hold
start
a[8:0]
000
004
00a
013
complete
root[4:0]
00
00
02
03
04

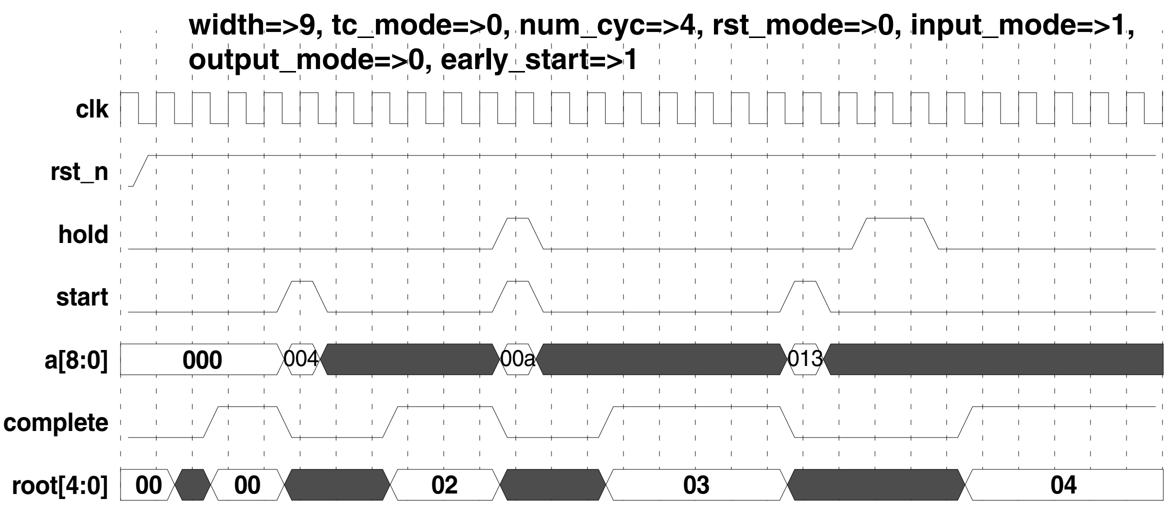
For the parameter settings shown in Figure 1-2, Table 1-6 specifies that the result is produced after three cycles.

Figure 1-2 Simulation Waveform 2


width=>9, tc_mode=>0, num_cyc=>4, rst_mode=>0, input_mode=>0, output_mode=>1, early_start=>0
clk
rst_n
hold
start
a[8:0]
000
004
00a
013
complete
root[4:0]
00
00
02
03
04

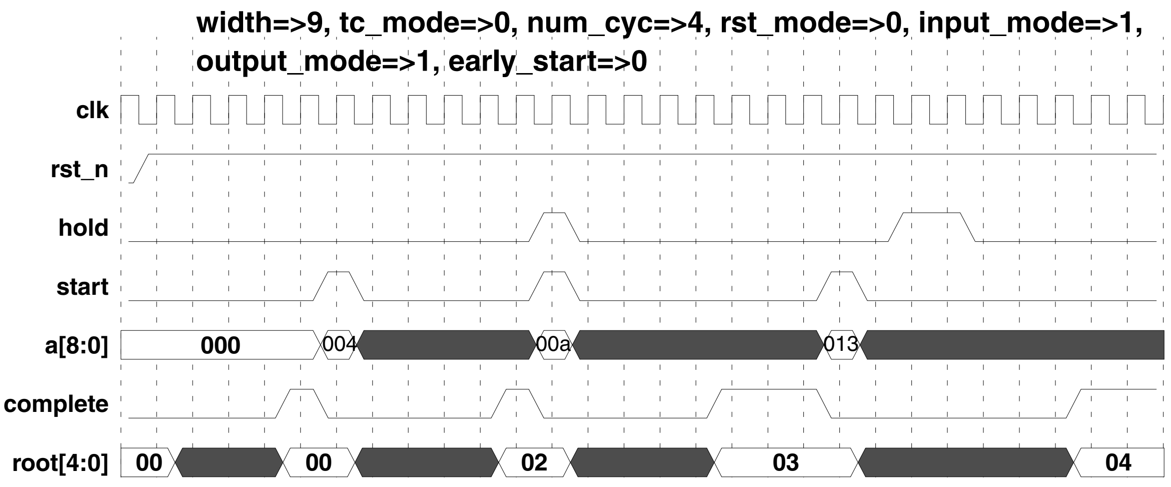
For the parameter settings shown in Figure 1-3, Table 1-6 on page 4 specifies that the result is produced after two cycles. Since *input_mode=1* (registered input) the input data can be removed after the first cycle.

Figure 1-3 Simulation Waveform 3


width=>9, tc_mode=>0, num_cyc=>4, rst_mode=>0, input_mode=>1, output_mode=>0, early_start=>1
clk
rst_n
hold
start
a[8:0]
000
004
00a
013
complete
root[4:0]
00
00
02
03
04

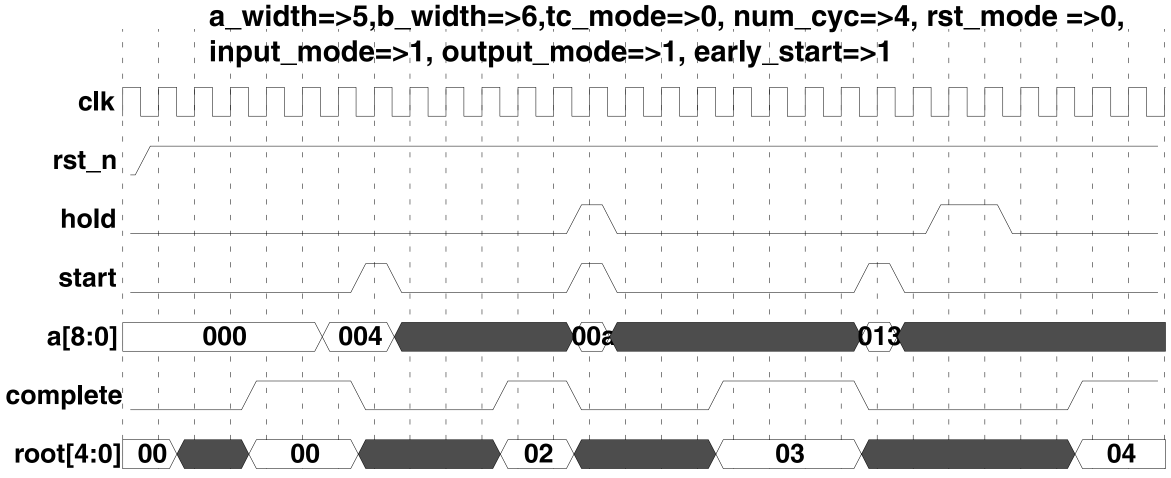
For the parameter settings shown in Figure 1-4, Table 1-6 specifies the result is produced after four cycles. Since *input_mode=1* (registered input) the input data can be removed after the first cycle.

Figure 1-4 Simulation Waveform 4


width=>9, tc_mode=>0, num_cyc=>4, rst_mode=>0, input_mode=>1, output_mode=>1, early_start=>0
clk
rst_n
hold
start
a[8:0]
000
004
00a
013
complete
root[4:0]
00
00
02
03
04

For the parameter settings shown in Figure 1-5, Table 1-6 on page 4 specifies that the result is produced after three clock cycles. Since *input_mode=1* (registered input), the input data can be removed after the first cycle. With `hold`=1 and `start`=1, the result is delayed by the same number of cycles that `hold`=1. Note that the data will be available on the *num_cyc* number of clocks after the data is registered. Note that the data is registered in the clock cycle immediately preceding `start`=1.

Figure 1-5 Simulation Waveform 5


a_width=>5,b_width=>6,tc_mode=>0, num_cyc=>4, rst_mode =>0,
input_mode=>1, output_mode=>1, early_start=>1
clk
rst_n
hold
start
a[8:0]
000
004
00a
013
complete
root[4:0]
00
00
02
03
04

Related Topics

- Math – Sequential Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_sqrt_seq_inst is
  generic (inst_width       : POSITIVE := 8; inst_tc_mode     : INTEGER := 0;
           inst_num_cyc     : INTEGER := 3;  inst_rst_mode    : INTEGER := 0;
           inst_input_mode  : INTEGER := 1;  inst_output_mode : INTEGER := 1;
           inst_early_start : INTEGER := 0 );
  port (inst_clk      : in std_logic;  inst_rst_n : in std_logic;
        inst_hold     : in std_logic;  inst_start : in std_logic;
        inst_a        : in std_logic_vector(inst_width-1 downto 0);
        complete_inst : out std_logic;
        root_inst     : out std_logic_vector((inst_width+1)/2-1 downto 0)  );
end DW_sqrt_seq_inst;

architecture inst of DW_sqrt_seq_inst is
begin
-- Instance of DW_sqrt_seq
  U1 : DW_sqrt_seq
    generic map (width => inst_width,   tc_mode => inst_tc_mode,
                 rst_mode => inst_rst_mode,   input_mode => inst_input_mode,
                 output_mode => inst_output_mode,
                 early_start => inst_early_start   )
    port map (clk => inst_clk,   rst_n => inst_rst_n,
              hold => inst_hold,   start => inst_start,   a => inst_a,
              complete => complete_inst,   root => root_inst   );
end inst;

-- pragma translate_off
configuration DW_sqrt_seq_inst_cfg_inst of DW_sqrt_seq_inst is
  for inst
  end for;
end DW_sqrt_seq_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_sqrt_seq_inst(inst_clk, inst_rst_n, inst_hold, inst_start, inst_a,
                        complete_inst, root_inst );

  parameter inst_width = 8;
  parameter inst_tc_mode = 0;
  parameter inst_num_cyc = 3;
  parameter inst_rst_mode = 0;
  parameter inst_input_mode = 1;
  parameter inst_output_mode = 1;
  parameter inst_early_start = 0;

  // Please add +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator
  // command line (for simulation).

  input inst_clk;
  input inst_rst_n;
  input inst_hold;
  input inst_start;
  input [inst_width-1 : 0] inst_a;
  output complete_inst;
  output [(inst_width+1)/2-1 : 0] root_inst;

  // Instance of DW_sqrt_seq
  DW_sqrt_seq #(inst_width, inst_tc_mode, inst_num_cyc, inst_rst_mode,
              inst_input_mode, inst_output_mode, inst_early_start)
    U1 (.clk(inst_clk),   .rst_n(inst_rst_n),   .hold(inst_hold),
        .start(inst_start),   .a(inst_a),   .complete(complete_inst),
        .root(root_inst) );
endmodule
```