I want to create a group of FFs which power-up in a certain state. How can I do this in HDL without creating an extra port in my design using Alliance1.4?

The ROCBUF was created for synthesis users who needed to create FFs which would power-up in a '1' or '0' state, but the FF would not have an asynchronous reset or set pin. FFs in XC4000 type devices with an asynchronous reset pin will power-up as a '0'. FF's in XC4000 type devices with an asynchronous set pin will power-up as a '1.'

By describing FFs with an asynchronous set or reset pin, you can create a group of FFs that power-up in a known pattern, like "10101111." If you want FFs with asynchronous reset or set pins, this is an easy task, because the HDL will describe this behavior. But, if you do not want or do not need these FFs to have a asynchronous reset or set pins, you must still describe, in the RTL code for the FFs, an asynchronous reset or set pin.

By connecting the HDL code which describes the asynchronous reset or set pin of an RTL described FF to the ROCBUF, you can create FFs that power-up in a known state. The ROCBUF will not synthesize to logic. So, even though the ROCBUF is connected to a top-level port, no extra pin will be added to a design. The top-level port the ROCBUF is connected to will not be implemented.

HDL State Machine Technique

Continued on the following page

HDL State Machine Technique

Continued from the previous page

VERILOG EXAMPLE OF USING THE ROCBUF:

```
module
stmchine(CLK,RESET,STRTSTOP,CLKEN,RST);

input CLK;
input RESET;
input STRTSTOP;
output CLKEN;
output RST;

reg CLKEN;
reg RST;
wire rstWire;

parameter [5:0] //synopsys enum
STATE_TYPE
 clear=6'b000001,
 zero=6'b000010,
 start=6'b000100,
 counting=6'b001000,
 stop=6'b010000,
 stopped=6'b100000;

reg [5:0] current_state;
reg [5:0] next_state;

always@(current_state or STRTSTOP)
begin
    case(current_state) //synopsys
full_case parallel_case
    clear:begin
        next_state<=zero;
        CLKEN<=1'b0;
        RST<=1'b1;
        end
    zero:begin

 next_state<=(STRTSTOP)?start:zero;
        CLKEN<=1'b0;
        RST<=1'b0;
        end
    start:begin

next_state<=(STRTSTOP)?start:counting;
        CLKEN<=1'b0;
        RST<=1'b0;
        end
    counting:begin

next_state<=(STRTSTOP)?stop:counting;
        CLKEN<=1'b1;
        RST<=1'b0;
    end
    stop:begin

next_state<=(STRTSTOP)?stop:stopped;
        CLKEN<=1'b0;
        RST<=1'b0;
    end
    stopped:begin

next_state<=(STRTSTOP)?start:stopped;
        CLKEN<=1'b0;
        RST<=1'b0;
    end
    endcase
 end

 always@(posedge CLK or posedge
rstWire)
 begin
    if(rstWire==1'b1)
    current_state = clear;
    else
    current_state = next_state;
 end

ROCBUF U1 (.I(RESET),.O(rstWire));

endmodule
```

VHDL EXAMPLE OF USING THE ROCBUF:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_unsigned.all;

entity smallcntr is

 port ( CE : in STD_LOGIC;
    CLK : in STD_LOGIC;
    CLR : in STD_LOGIC;
    QOUT : out STD_LOGIC_VECTOR(9
downto 0)
    );
end smallcntr;

architecture inside of smallcntr is

signal qoutsig : STD_LOGIC_VECTOR(9
downto 0);
signal clrSig: STD_LOGIC;

component ROCBUF
port(I: in STD_LOGIC; O: out
STD_LOGIC);
end component;

begin

process(CE,CLK,clrSig)
 begin

 if(clrSig='1') then
   qoutsig <="0100010001";
 elsif(CE='1') then
   if(CLK'event and CLK='1') then
      qoutsig<=qoutsig +
"0000000001";
   end if;
 end if;
 end process;

QOUT<=qoutsig;

U1: ROCBUF port
map(I=>CLR,O=>clrSig);

end inside;
```