EE214: Experiment 2 (Thursday Batch)

Wadhwani Electronics Lab, IIT Bombay

10th March 2021

Instructions:

1. So far we have dealt with structural and data-flow kind of design styles. However it may become tedious to optimize boolean expressions manually for complex designs. Behavioral description comes handy when exact design schematic is not known or too large to implement structurally. However one should not take this availability for granted and write C/C++ style codes without considering the synthesizability and timing of the design.
2. You should not use any other library & package (except ieee.std_logic_1164.all)
3. A "Latch" or Register or combinational loop should not be inferred from your VHDL code by Quartus since we are making a combinational circuit.
4. We encourage students to use generics appropriately and avoid hard-coding as much as possible
5. Please upload well –commented– code

Problem Statement

1. Design and describe 4-bit divider in VHDL using Behavioral modeling. You can use the Skeleton code attached or you can code from scratch. (5)
2. Generate tracefile with the following format(Set mask bits to all zeros when divisor is 0). (2)
(N3 N2 N1 N0 D3 D2 D1 D0) (R7 R6 R5 R4 Q3 Q2 Q1 Q0) (1 1 1 1 1 1 1 1)
Here N, D, R, Q represents dividend, divisor, remainder and quotient respectively.
3. Verify working of your design by performing RTL and Gate-level simulation. Show the results to your TA.(3)
4. Verify working of your design using scanchain. Show the results to your TA. (5)

Hints:

- 1D X 1D array can be declared as
type pr_type is array (*natural range* <>) *of std_logic_vector*(7 *downto* 0);
variable pr : *pr_type*(0 *to* 3) := (*others* => (*others* =>' 0'));

  OR

  type pr_type is array (0 *to* 3) *of std_logic_vector*(7 *downto* 0);
variable pr : *pr_type* := (*others* => (*others* =>' 0'));
- Values can be assigned to 1D X 1D array variable as
pr(2)(3) := '1';
pr(0)(3 downto 0) := "1010";

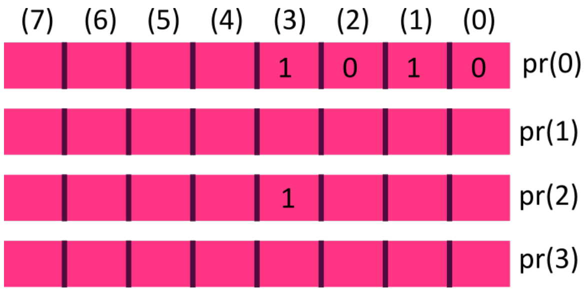

Figure 1: 1D x 1D array

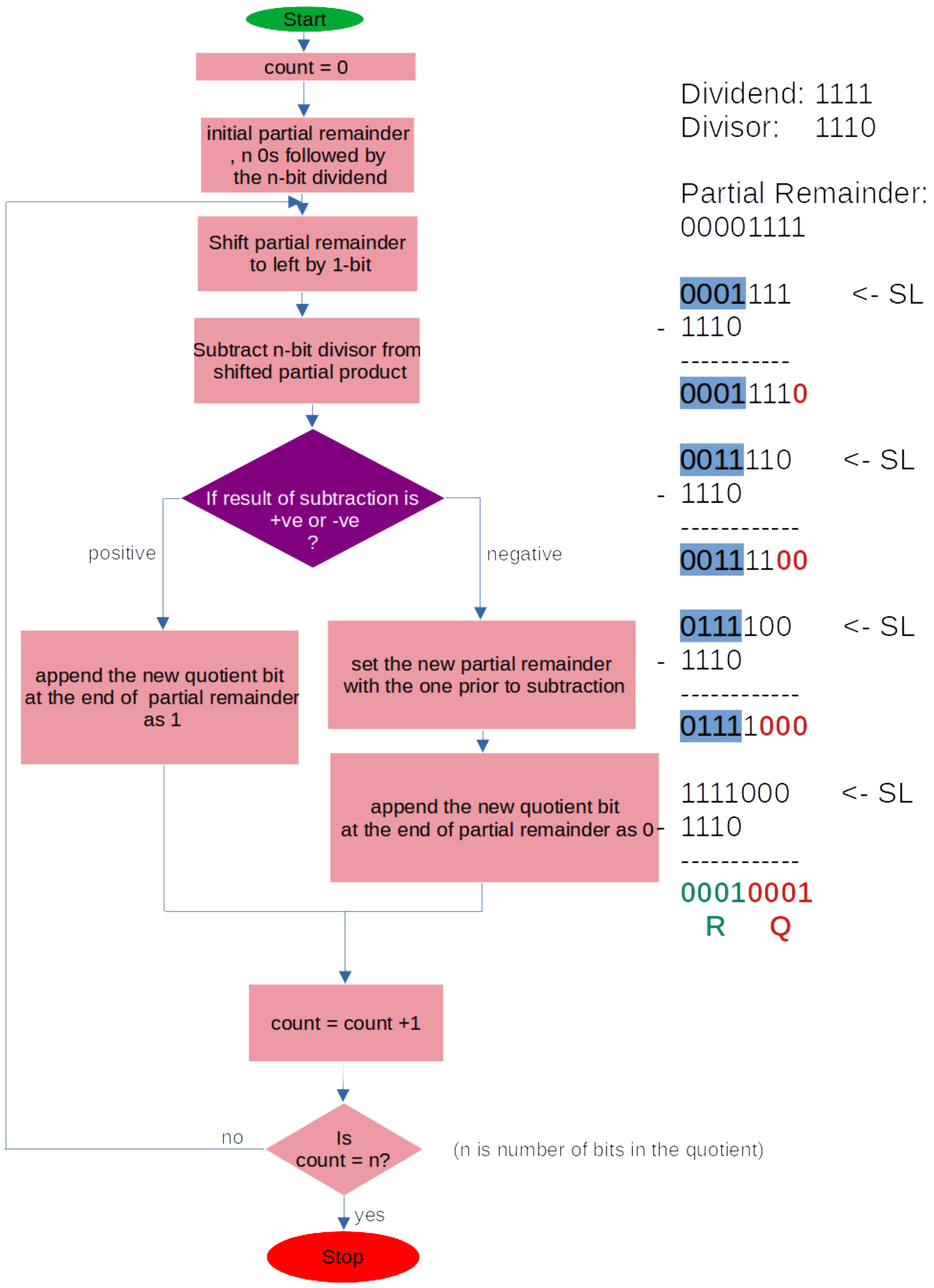
Start
count = 0
initial partial remainder , n 0s followed by the n-bit dividend
Shift partial remainder to left by 1-bit
Subtract n-bit divisor from shifted partial product
If result of subtraction is +ve or -ve ?
positive
negative
append the new quotient bit at the end of partial remainder as 1
set the new partial remainder with the one prior to subtraction
append the new quotient bit at the end of partial remainder as 0
count = count +1
Is count = n?
no
yes
Stop
(n is number of bits in the quotient)
Dividend: 1111
Divisor: 1110
Partial Remainder: 00001111
0001111 <- SL
- 1110
00011110
0011110 <- SL
- 1110
00111100
0111100 <- SL
- 1110
01111000
1111000 <- SL
- 1110
00010001
R Q

Figure 2: Divider algorithm

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

entity div is
    generic(
        N : integer:=4; -- operand width
        NN : integer:=8 -- result width
        );
    port (
        Nu: in std_logic_vector(N-1 downto 0);-- Nu (read numerator) is dividend
        D: in std_logic_vector(N-1 downto 0);-- D (read Denominator) is divisor
        RQ: out std_logic_vector((NN)-1 downto 0)--upper N bits of RQ will have remainder and
                                                 --lower N bits will have quotient
    ) ;
end div;

architecture beh of div is
    -- unconstrained 1D x 1D array
    type pr_type is array (natural range<>) of std_logic_vector(NN-1 downto 0);

    -- subtractor function. [Usage: var := sub(X,Y) where var is a variable
    --                       and X,Y are two 4-bit inputs for subtractor]
    function sub(A: in std_logic_vector; B: in std_logic_vector)
        return std_logic_vector is
            -- variable declaration
            variable W : integer := A'length;
            variable diff : std_logic_vector(W downto 0):= (others=>'0');
            variable borrow : std_logic_vector(W downto 0):= (0 => '1', others=>'0');
            variable B_sign: std_logic_vector(A'length-1 downto 0):=(others=>'0');
        begin
            B_sign(B'length-1 downto 0) := not B;
            for i in 0 to W-1 loop
                diff(i) := A(i) xor B_sign(i) xor borrow(i);
                borrow(i+1) := (A(i) and B_sign(i)) or (borrow(i) and (A(i) xor B_sign(i)));
            end loop;
            diff(W) := not borrow(W);
            return diff;
    end sub;

begin


division : process(Nu, D)
-- Here Nu (read numerator) is dividend and D (read denominator) is divisor
-- variable k holds length of dividend
variable k : integer := Nu'length;

-- 1D x 1D array should be used, instead of reading and writing to same variable
-- (This is a limitation of VHDL synthesizer)
-- declare variable to hold partial remainder for subsequent iteration
--////////////TODO

-- declare variable to hold difference from subtractor
--////////////TODO

-- declare temporary variable to hold prior partial product in case difference is negative
--////////////TODO
begin
    -- sequential statements to calculate quotient and remainder
    --////////////TODO

    RQ <= --////////////TODO-- final result assignment
end process ; -- division
end beh ; -- beh
```