4

High-Performance Coding Techniques

This chapter contains examples utilizing various high-performance coding techniques.

Data-Path Duplication

The following examples illustrate how to duplicate logic in HDL to improve timing.

In Example 4-1 and Example 4-2, `CONTROL` is a late arriving input signal. The goal is to reduce the logic from `CONTROL` to the output port `COUNT`.

Example 4-1 Original Verilog Before Logic Duplication

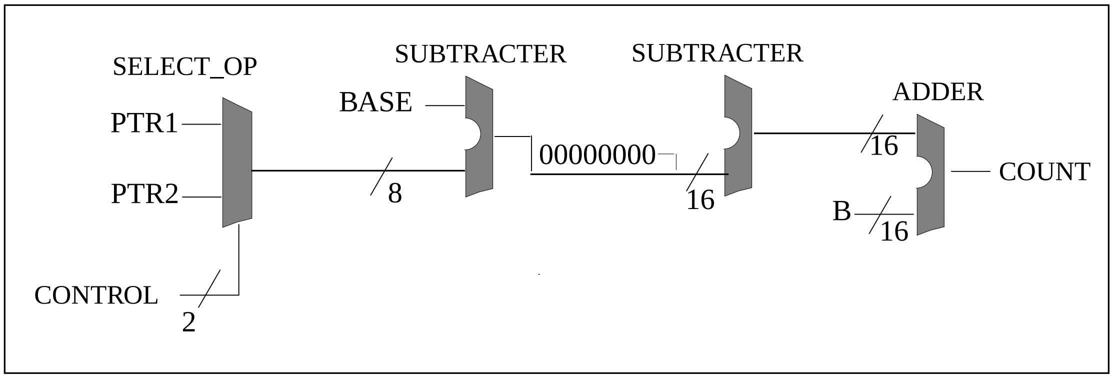
```
module BEFORE (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);
input [7:0] PTR1,PTR2;
input [15:0] ADDRESS, B;
input CONTROL;       // CONTROL is late arriving
output [15:0] COUNT;

parameter [7:0] BASE = 8'b10000000;
wire [7:0] PTR, OFFSET;
wire [15:0] ADDR;

assign PTR = (CONTROL == 1'b1) ? PTR1 : PTR2;
assign OFFSET = BASE - PTR; //Could be any function f(BASE,PTR)
assign ADDR = ADDRESS - {8'h00, OFFSET};
assign COUNT = ADDR + B;

endmodule
```

Example 4-2 Original VHDL Before Logic Duplication

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

entity BEFORE is
port(ADDRESS, B : in std_logic_vector (15 downto 0);
        PTR1, PTR2 : in std_logic_vector (7 downto 0);
  CONTROL    : in  std_logic; -- CONTROL is late arriving
        COUNT      : out std_logic_vector (15 downto 0));
end BEFORE;

architecture RTL of BEFORE is
begin
        process (B, CONTROL, ADDRESS, PTR1, PTR2)
        constant BASE : std_logic_vector (7 downto 0) := ”10000000”;
        variable PTR, OFFSET : std_logic_vector (7 downto 0);
        variable ADDR        : std_logic_vector (15 downto 0);
begin
              if CONTROL = ’1’ then
                   PTR := PTR1;
              else
                   PTR := PTR2;
              end if;

              OFFSET := BASE - PTR;  -- Could be any function f(BASE,PTR)
              ADDR := ADDRESS - (”00000000” & OFFSET);

              COUNT <= ADDR + B;
        end process;
end RTL;
```

Figure 4-1 shows the structure implied by the original HDL.

Figure 4-1 Structure Implied by Original HDL Before Logic Duplication



In Figure 4-1, notice that there is a SELECT_OP next to a subtracter. When you see a SELECT_OP next to an operator, there is a good chance that you can move the SELECT_OP to after the operator. You might want to do this if the control signal for the SELECT_OP is late arriving. You can move the SELECT_OP by duplicating the logic in the branches of the conditional statement that implied the SELECT_OP.

In Figure 4-1, you can also see the signal that CONTROL selects between two inputs. The selected input drives a chain of arithmetic operations (the data path) and ends at the output port COUNT. If CONTROL arrives late, you want to move the selection closer to the output port COUNT.

Example 4-3 and Example 4-4 show the improved HDL for Example 4-1 and Example 4-2. The improved HDL shows the data-path duplication described previously.

Example 4-3 Improved Verilog With Data Path Duplicated

```
module PRECOMPUTED (ADDRESS, PTR1, PTR2, B, CONTROL, COUNT);
input [7:0] PTR1, PTR2;
input [15:0] ADDRESS, B;
input CONTROL;
output [15:0] COUNT;

parameter [7:0] BASE = 8’b10000000;
wire [7:0] OFFSET1,OFFSET2;
wire [15:0] ADDR1,ADDR2,COUNT1,COUNT2;

assign OFFSET1 = BASE - PTR1;  // Could be f(BASE,PTR)
assign OFFSET2 = BASE - PTR2;  // Could be f(BASE,PTR)
assign ADDR1 = ADDRESS - {8’h00 , OFFSET1};
assign ADDR2 = ADDRESS - {8’h00 , OFFSET2};
assign COUNT1 = ADDR1 + B;
assign COUNT2 = ADDR2 + B;
assign COUNT = (CONTROL == 1’b1) ? COUNT1 : COUNT2;

endmodule
```

Example 4-4 Improved VHDL With Data Path Duplicated

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

entity PRECOMPUTED is
port (ADDRESS, B : in std_logic_vector (15 downto 0);
        PTR1, PTR2 : in std_logic_vector (7 downto 0);
        CONTROL    : in  std_logic;
        COUNT      : out std_logic_vector (15 downto 0));
end PRECOMPUTED;

architecture RTL of PRECOMPUTED is
begin
        process (CONTROL, ADDRESS, B, PTR1, PTR2)
        constant BASE : std_logic_vector (7 downto 0) := ”10000000”;
        variable OFFSET2, OFFSET2 : std_logic_vector (7 downto 0);
        variable ADDR1, ADDR2     : std_logic_vector (15 downto 0);
        variable COUNT1, COUNT2   : std_logic_vector (15 downto 0);
        begin

             OFFSET1 := BASE - PTR1;  -- Could be f(BASE,PTR)
```

```
        OFFSET2 := BASE - PTR2;  -- Could be f(BASE,PTR)

        ADDR1 := ADDRESS - ("00000000" & OFFSET1);
        ADDR2 := ADDRESS - ("00000000" & OFFSET2);

        COUNT1 := ADDR1 + B;
        COUNT2 := ADDR2 + B;

        if CONTROL = '1' then
             COUNT <= COUNT1;
        else
             COUNT <= COUNT2;
        end if;

    end process;

end RTL;
```

When you duplicate the operations that depend on the inputs PTR1 and PTR2, the assignment to COUNT becomes a selection between the two parallel data paths. The signal CONTROL selects the data path. The path from CONTROL to the output port COUNT is no longer the critical path, but this change comes at the expense of duplicated logic.

In Example 4-3 and Example 4-4, the entire data path is duplicated because CONTROL arrives late. Had CONTROL arrived earlier, you could have duplicated only a portion of the logic, thereby decreasing the area expense. The designer controls how much logic is duplicated.

In addition, the amount of duplication is proportional to the number of branches in the conditional statement. For example, if there were four PTR signals in Example 4-1 and Example 4-2 instead of two (PTR1 and PTR2), the area penalty would be larger, because you would have two more duplicated data paths.

Figure 4-2 shows the structure implied by the improved HDL.

Figure 4-2 Structure Implied by Improved HDL With Data Path Duplication

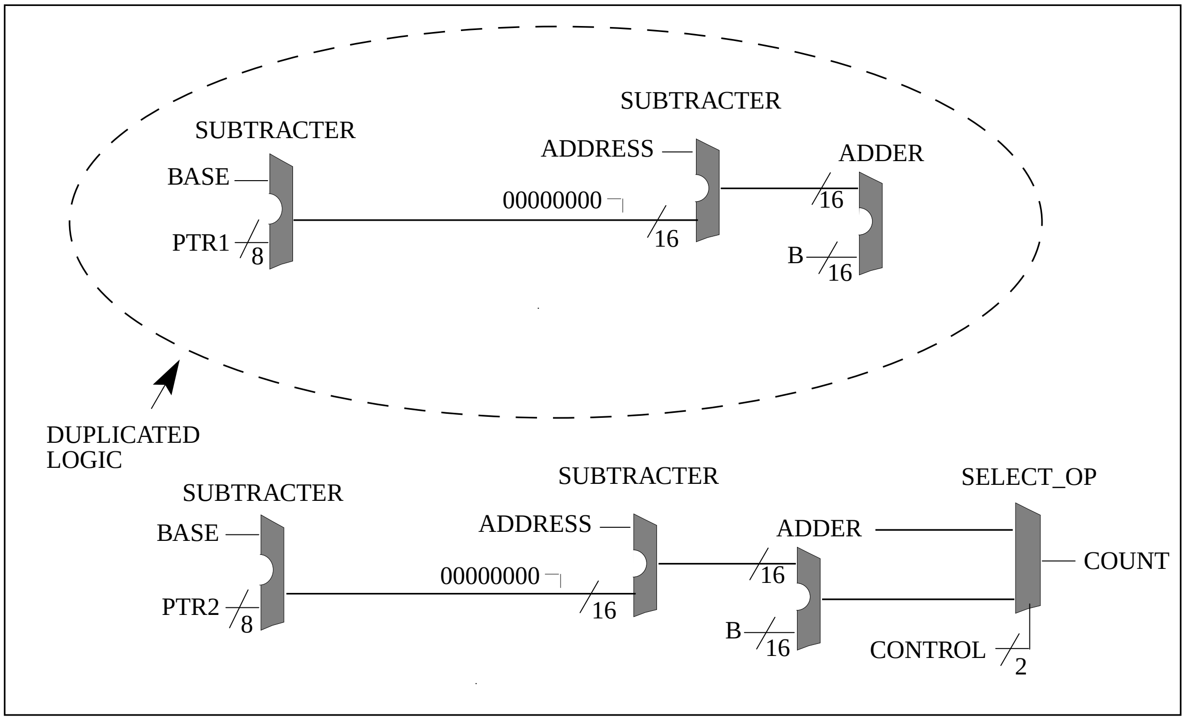


Table 4-1 shows the timing and area results for the original and the improved HDL shown in Example 4-1, Example 4-2, Example 4-3, and Example 4-4. The timing numbers are for the path from CONTROL to COUNT[9], which was the worst path in the original design.

Table 4-1 Timing and Area Results for Data-Path Duplication

	Data Arrival Time	**Area**
Original Design	5.23	1057
Improved Design	2.33	1622

In conclusion, the improved design with the data path duplicated is much better with respect to timing. As expected, the area is worse for the improved design. If you want to optimize your design for timing and are less concerned about area, data-path duplication is the recommended methodology. Note that logic duplication also increases the load on the input pins.

Operator in if Condition

Example 4-5 and Example 4-6 show Verilog and VHDL designs that contain operators in the conditional expression of an if statement. The signal A in the conditional expression is a late arriving signal, so you should move the signal closer to the output.

Example 4-5 Original Verilog With Operator in Conditional Expression

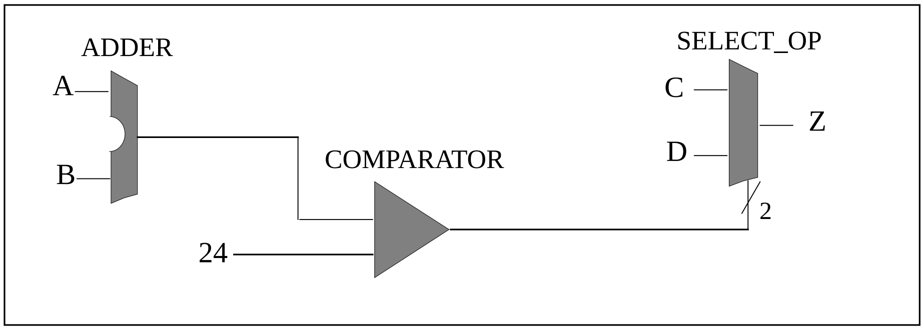
```
module cond_oper(A, B, C, D, Z);
parameter N = 8;
input [N-1:0] A, B, C, D; //A is late arriving
output [N-1:0] Z;

reg [N-1:0] Z;

always @(A or B or C or D)
begin
    if (A + B < 24)
        Z <= C;
    else
        Z <= D;
end

endmodule
```

Example 4-6 Original VHDL With Operator in Conditional Expression

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

entity cond_oper is
generic(N: natural := 8);
port(A, B: in std_logic_vector(N-1 downto 0);
  -- A is late arriving
    C, D: in std_logic_vector(N-1 downto 0);
    Z: out std_logic_vector(N-1 downto 0));
end cond_oper;

architecture one of cond_oper is
begin
    process(A, B, C, D)
    begin
        if (A + B < 24) then
            Z <= C;
        else
            Z <= D;
        end if;
    end process;
end one;
```

Figure 4-3 shows the structure implied by the original HDL in Example 4-5 and Example 4-6. The signal A is an input to the adder in Figure 4-3.

Figure 4-3 Structure Implied by Original HDL With Late Arriving A Signal



You want to reduce the number of operations that have the signal A in their `fanin` cone. Example 4-7 and Example 4-8 show the improved HDL for Example 4-5 and Example 4-6.

Example 4-7 Improved Verilog With Operator in Conditional Expression

```
module cond_oper_improved (A, B, C, D, Z);
parameter N = 8;
input [N-1:0] A, B, C, D; // A is late arriving
output [N-1:0] Z;

reg [N-1:0] Z;

always @(A or B or C or D)
begin
     if (A < 24 - B)
          Z <= C;
     else
          Z <= D;
     end
     endmodule
```

Example 4-8 Improved VHDL With Operator in Conditional Expression

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

entity cond_oper_improved is
generic (N : natural := 8);
port (A, B : in std_logic_vector(N-1 downto 0);
  -- A is late arriving
    C, D : in std_logic_vector(N-1 downto 0);
    Z    : out std_logic_vector(N-1 downto 0));
end cond_oper_improved;

architecture one of cond_oper_improved is
begin
    process(A, B, C, D)
    begin
        if (A < 24 - B) then
            Z <= C;
        else
            Z <= D;
        end if;
    end process;
end one;
```

Figure 4-4 shows the structure implied by the improved HDL. The signal A is an input to the comparator in Figure 4-4.

Figure 4-4 Structure Implied by Improved HDL With A as Input to Comparator

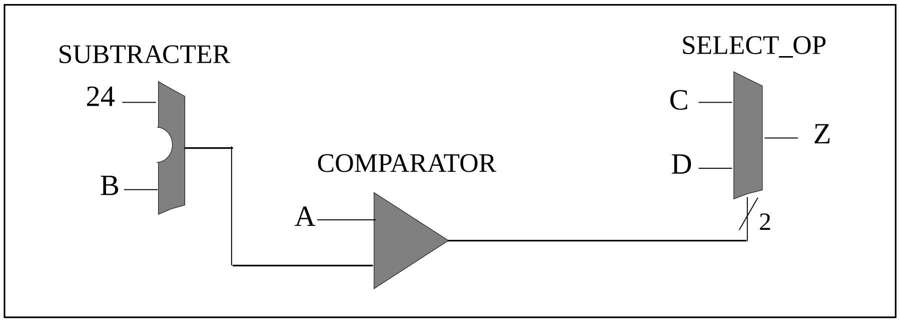


Table 4-2 shows the timing and area results (given that A is a late arriving input) for the original and improved HDL shown in Example 4-5, Example 4-6, Example 4-7, and Example 4-8. The timing results are for the worst path in the design.

Table 4-2 Timing and Area Results for Conditional Operator Examples

	Data Arrival Time	**Area**
Original Design	4.33	411.1
Improved Design	3.89	271.0