EE-421: Digital System Design

Doing MATHS on an FPGA

Arithmetic Circuits: Serial Multiplier (aka Shift-and-Add Multiplier)

Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST
SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE (SEECS)

FPGA Implementation of a Combinational Multiplier

- In Verilog:

```
P = A * B;
```

(make sure **P** has twice the number of bits of **A** and **B**)

- Synthesis tool will create a combinational multiplier if DSP Block inference is turned off.

F_{max} of a Combinational Multiplier

- Measured on our Cyclone FPGA:

Multiplier Width (in bits)	F_{max} (MHz)
8 x 8	464
16 x 16	396
32 x 32	104
64 x 64	66

Serial (Multi-cycle) Multiplier

Multi Cycle Multiplier

How do you multiply by hand?

$$
\begin{array}{rl}
 & 1101 \quad A \\
\times & 1011 \quad B \\
\hline
 & 1101 \\
 & 11010 \\
 & 000000 \\
+ & 1101000 \\
\hline
 & 10001111 \quad P
\end{array}
$$

One Possible Algorithm

```
    1101   A
  × 1011   B
  ------
    1101
   11010
  000000
+ 1101000
  -------
10001111   P
```

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Note: This is NOT Verilog

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$

A

B

0 **P_{old}**

P

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$

A

B

0 **P_{old}**

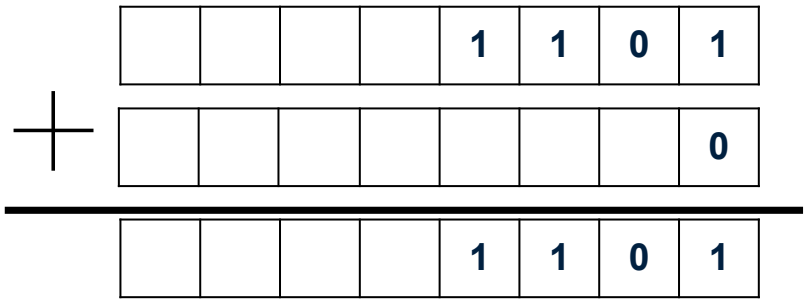
P

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$

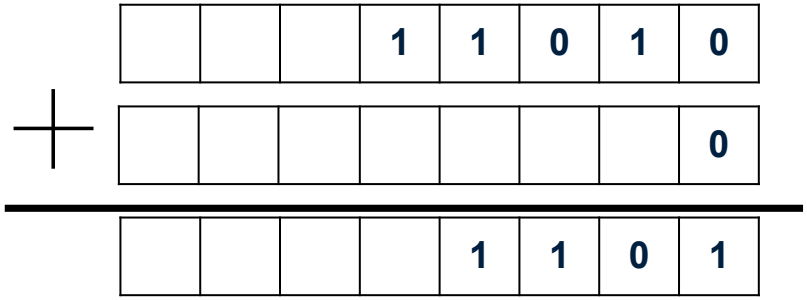


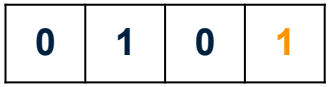

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$



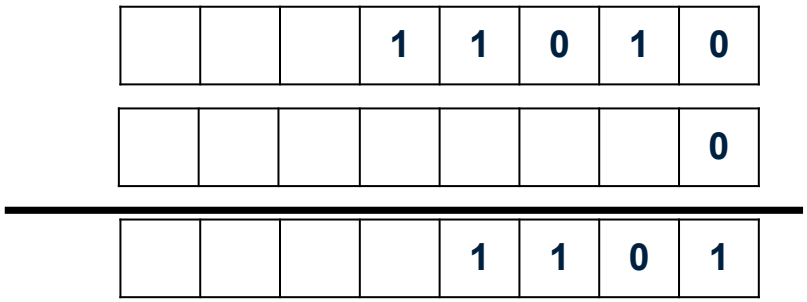


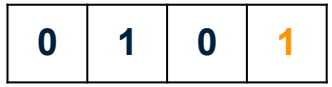

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$



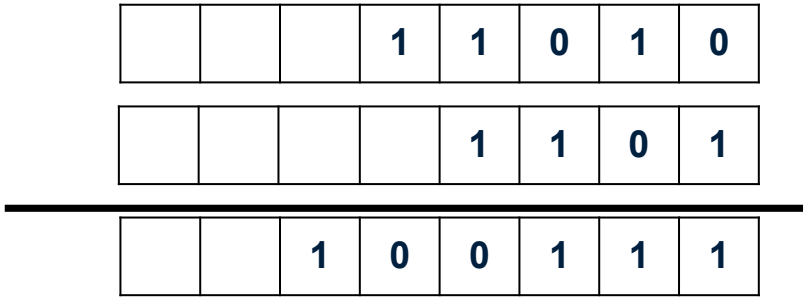



```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$

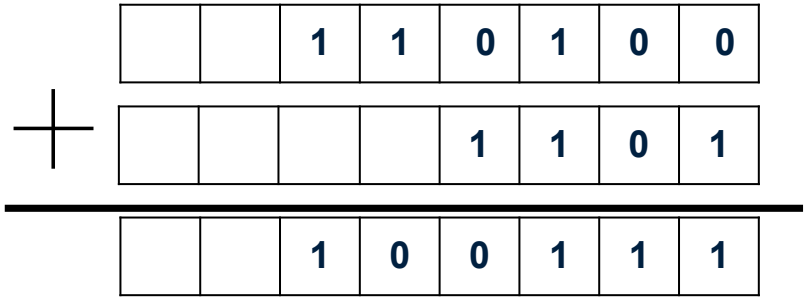



```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Example

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$



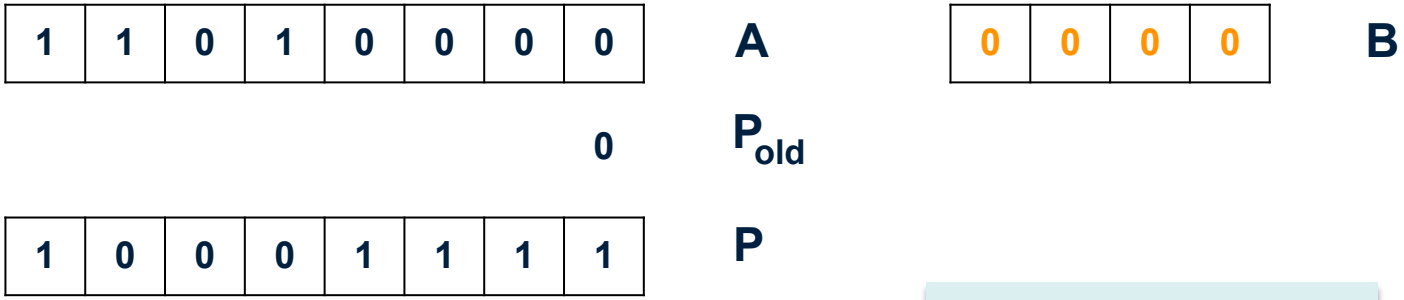

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Continue until B is ZERO ….

Final Result

$$13 \times 11 = 143$$

$$1101 \times 1011 = 10001111$$



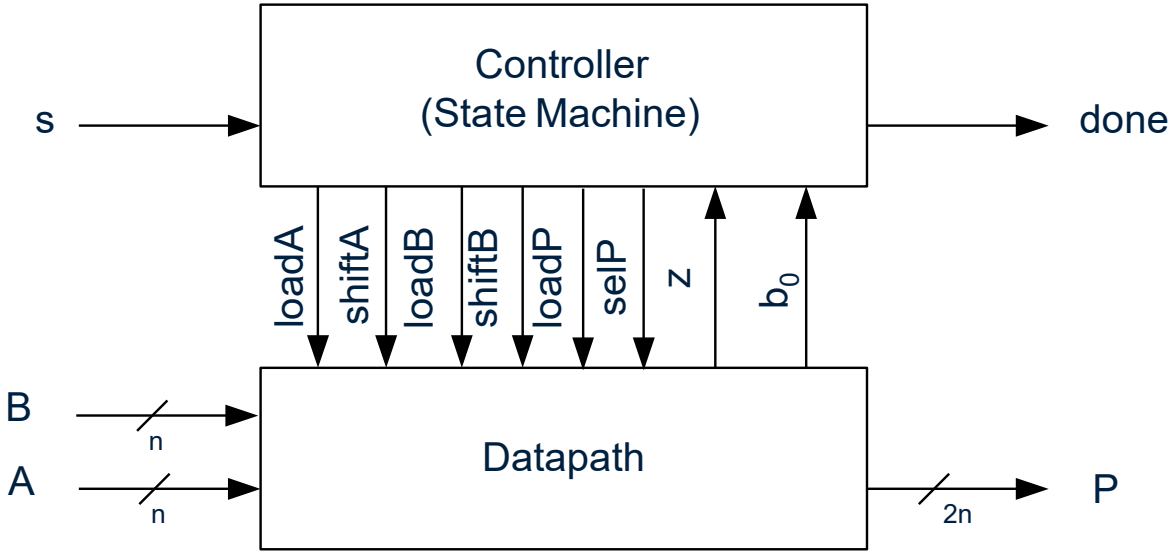

```
P = 0
while B != 0:
  if B(0) == 1:
    P = P + A
  A = A << 1
  B = B >> 1
```

Alternate Algorithm Representation

```
P = 0
while B != 0:
    if B(0) == 1:
        P = P + A
    A = A << 1
    B = B >> 1
```

$P = 0$;
for $i = 0$ to $n - 1$ do
    if $b_i = 1$ then
        $P = P + A$;
    end if;
    Left-shift A ;
end for;

Top – Level Schematic

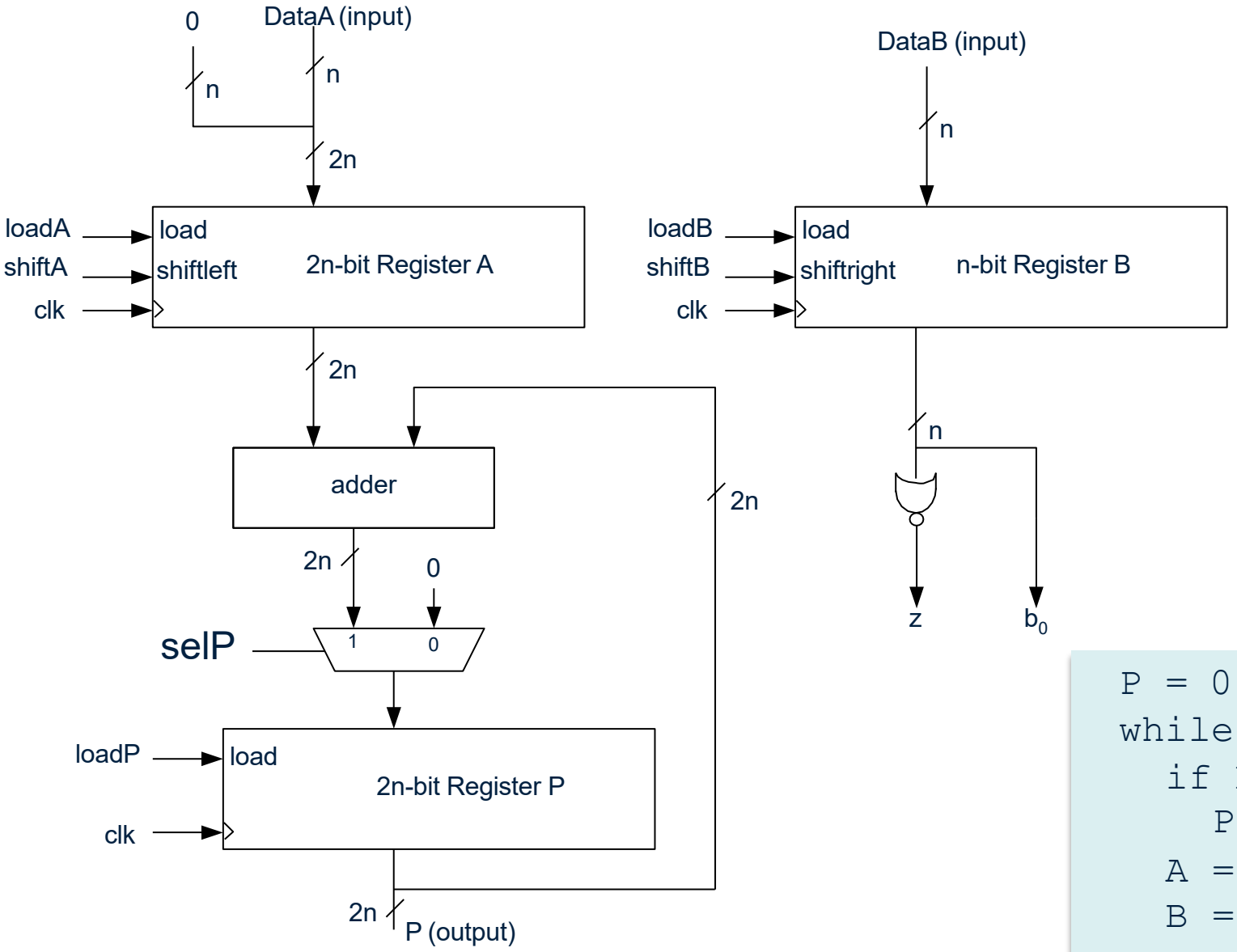


When ***s*** goes high, n-bit values are available on ***A*** and ***B.***

The machine then multiplies, and when it is finished, asserts ***done*** and puts the result on ***P***.

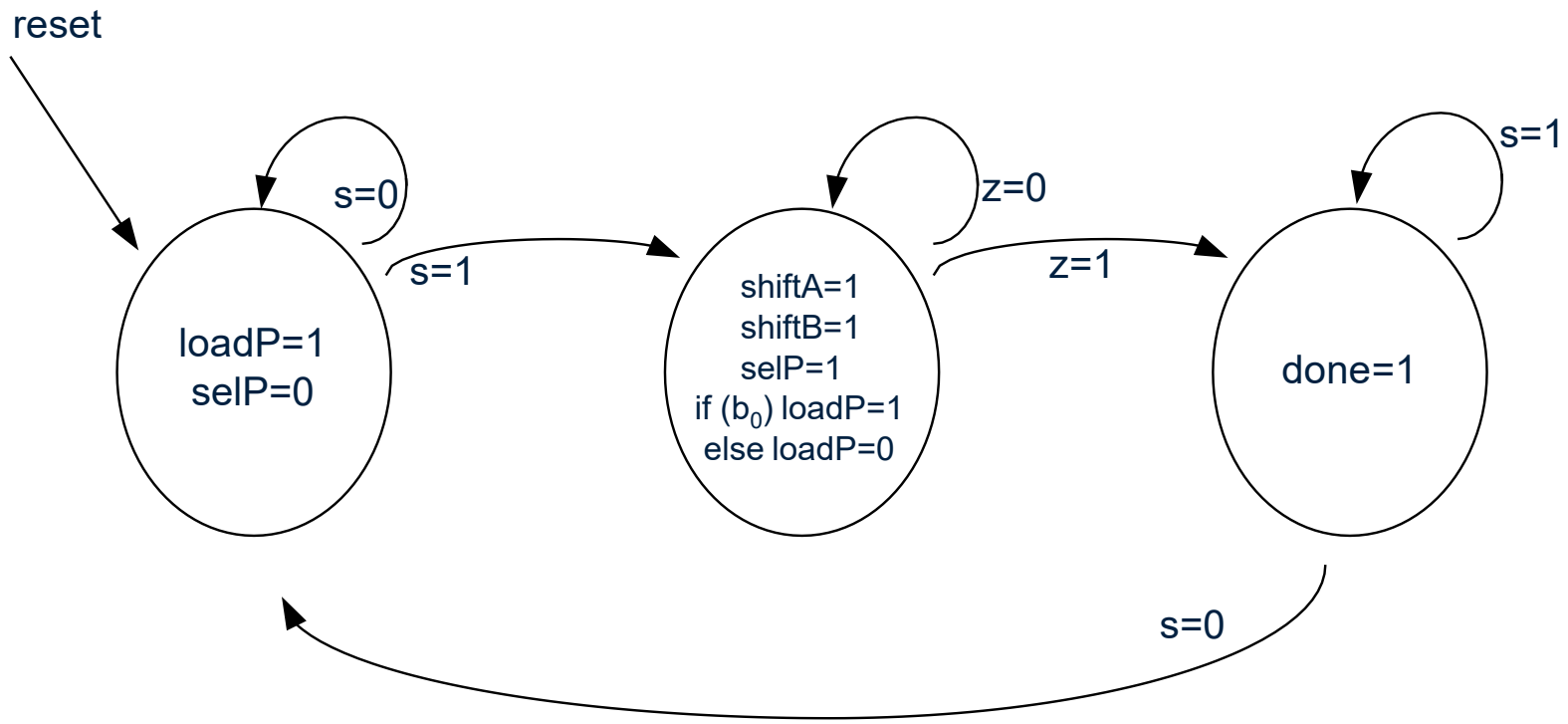
It then waits for **s** to go low.

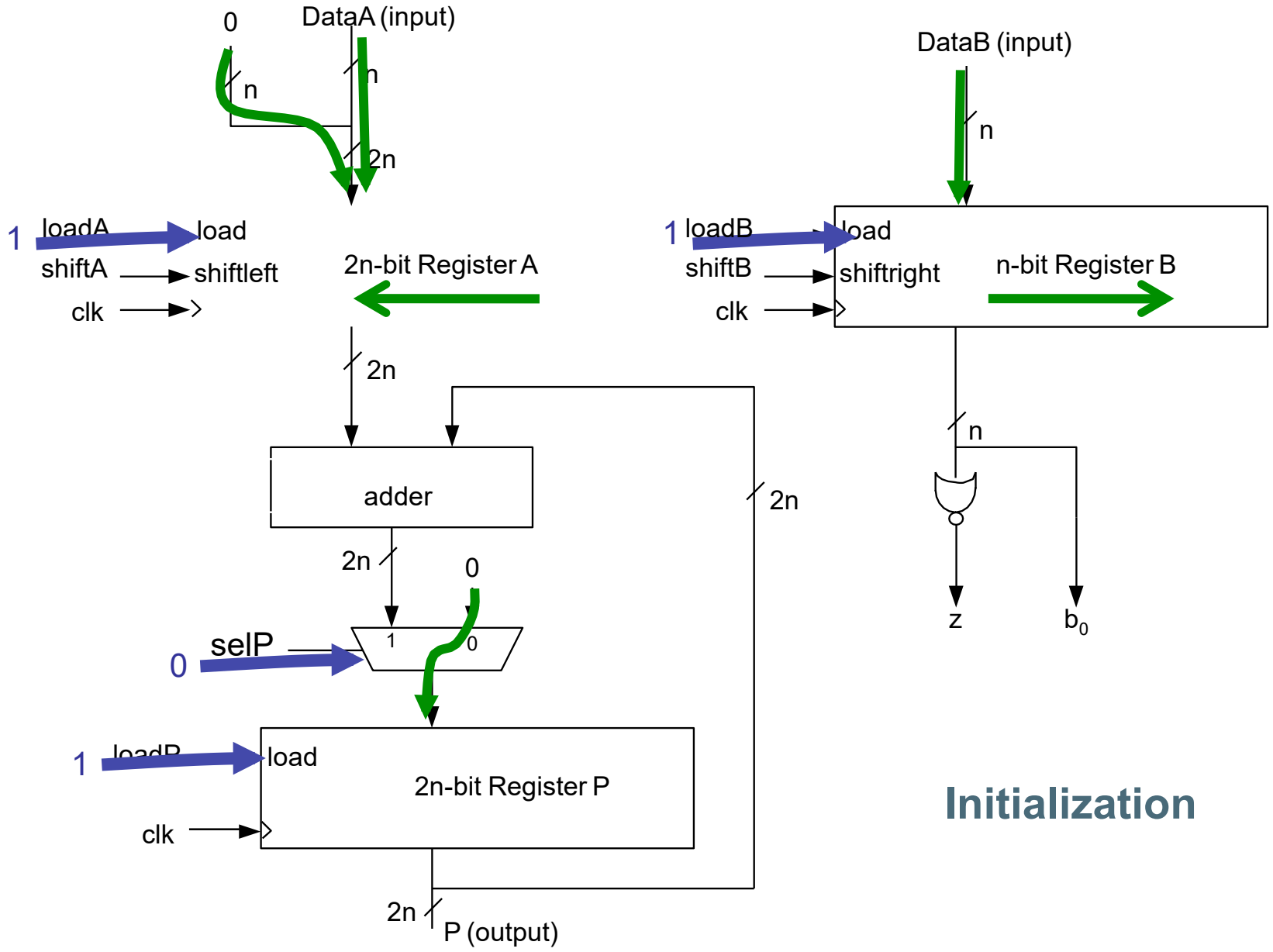
Datapath



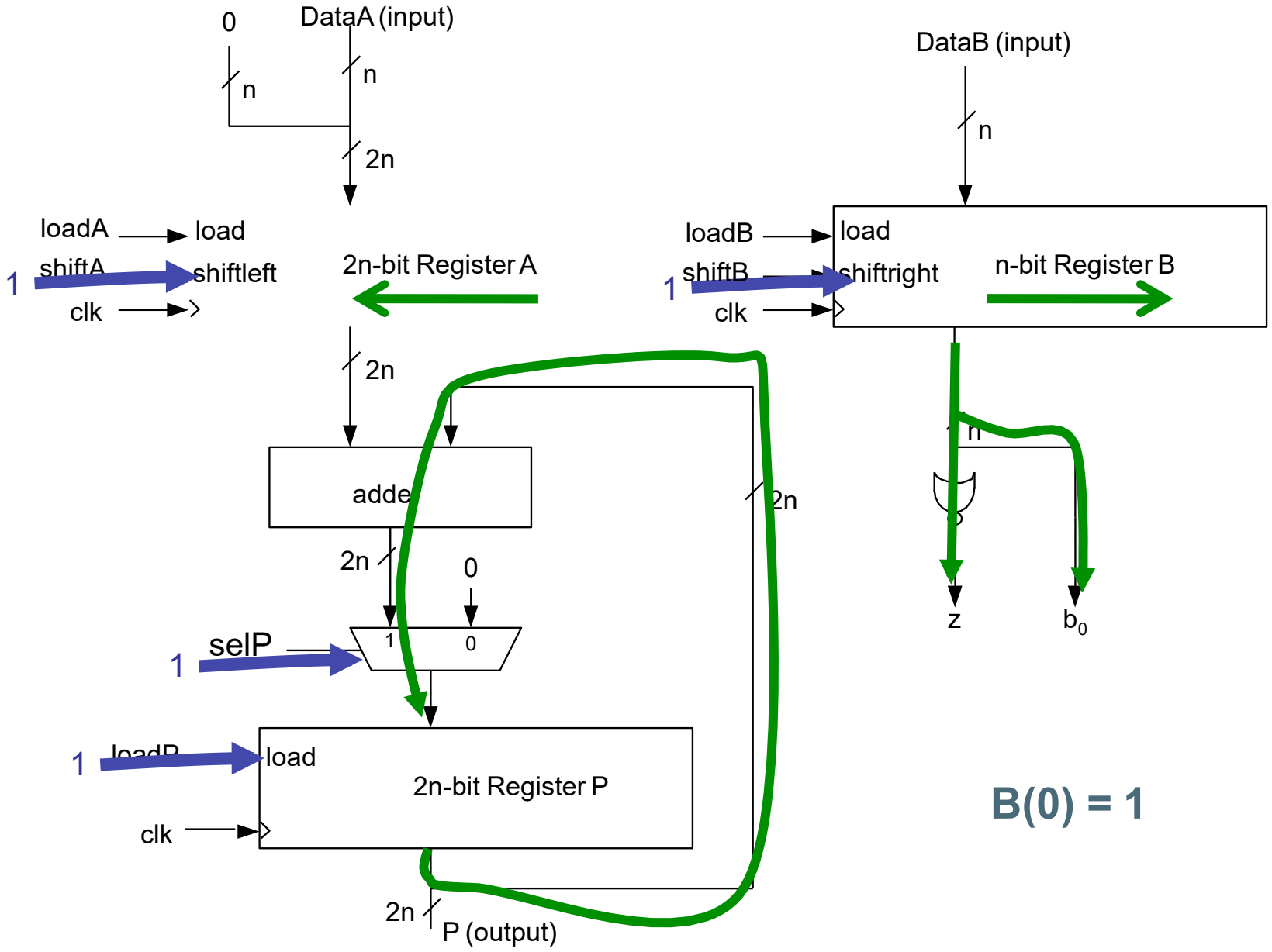

```
P = 0
while B != 0
  if B(0) == 1
    P = P + A
  A = A << 1
  B = B >> 1
```

Controlpath

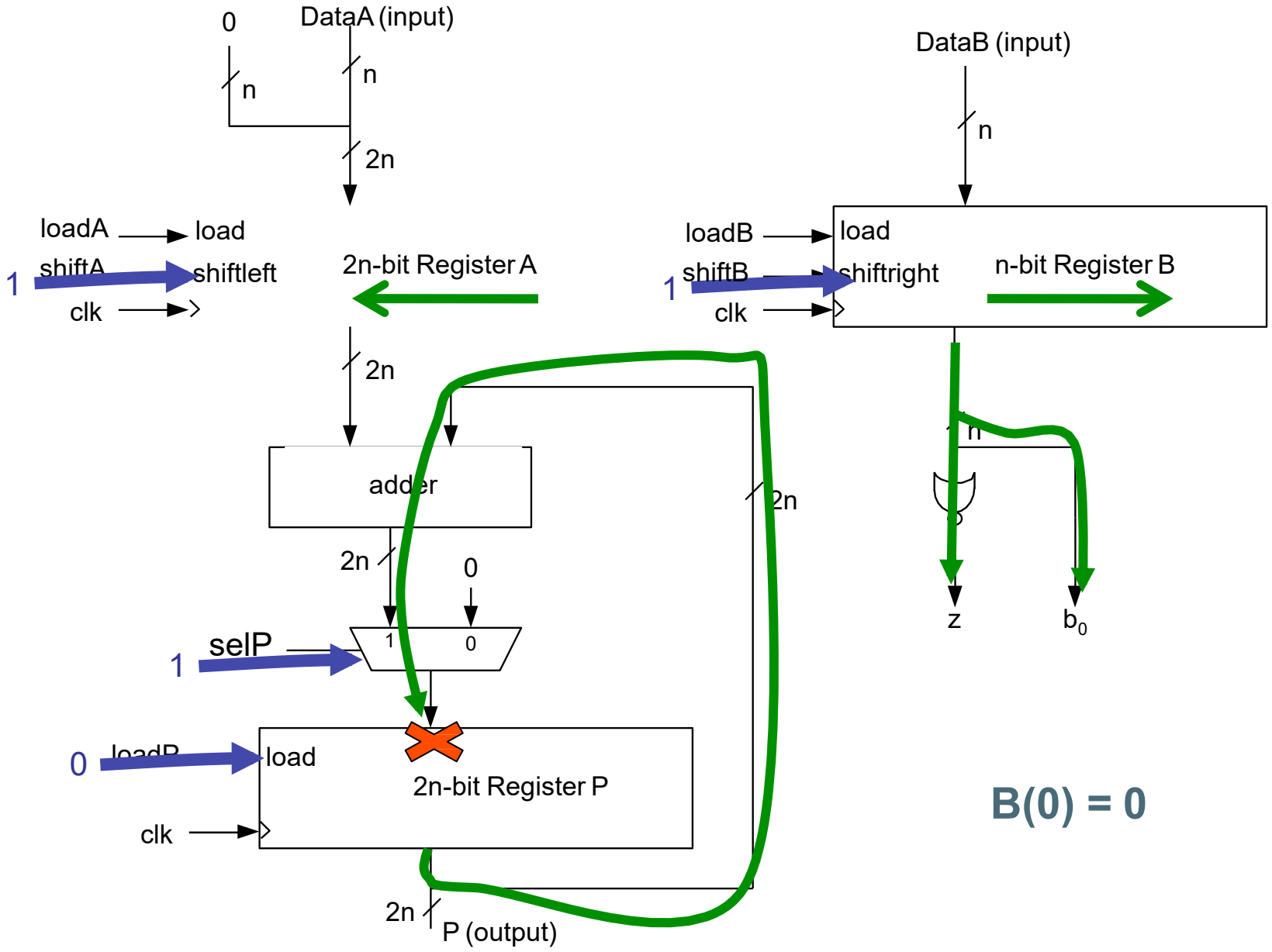




Initialization


0
DataA (input)
n
n
2n
DataB (input)
n
loadA
load
shiftA
shiftleft
clk
2n-bit Register A
loadB
load
shiftB
shiftright
clk
n-bit Register B
1
1
2n
adde
2n
2n
0
n
z
b_0
selP
1
1
0
loadP
load
2n-bit Register P
clk
2n
P (output)

B(0) = 1


0
DataA (input)
n
n
2n
DataB (input)
n
loadA
load
shiftA
shiftleft
clk
1
2n-bit Register A
loadB
load
shiftB
shiftright
clk
1
n-bit Register B
2n
adder
n
2n
2n
0
1
0
selP
1
load
0
2n-bit Register P
clk
2n
P (output)
z
b0

B(0) = 0

How to describe the cct?

- You can describe the Datapath and FSM in Verilog based on these schematics just like we have been doing throughout the course.

- Or, now that we understand the details, we can leverage the Synthesis tool's capabilities of inferring the datapath for us, and describe the algorithm instead.

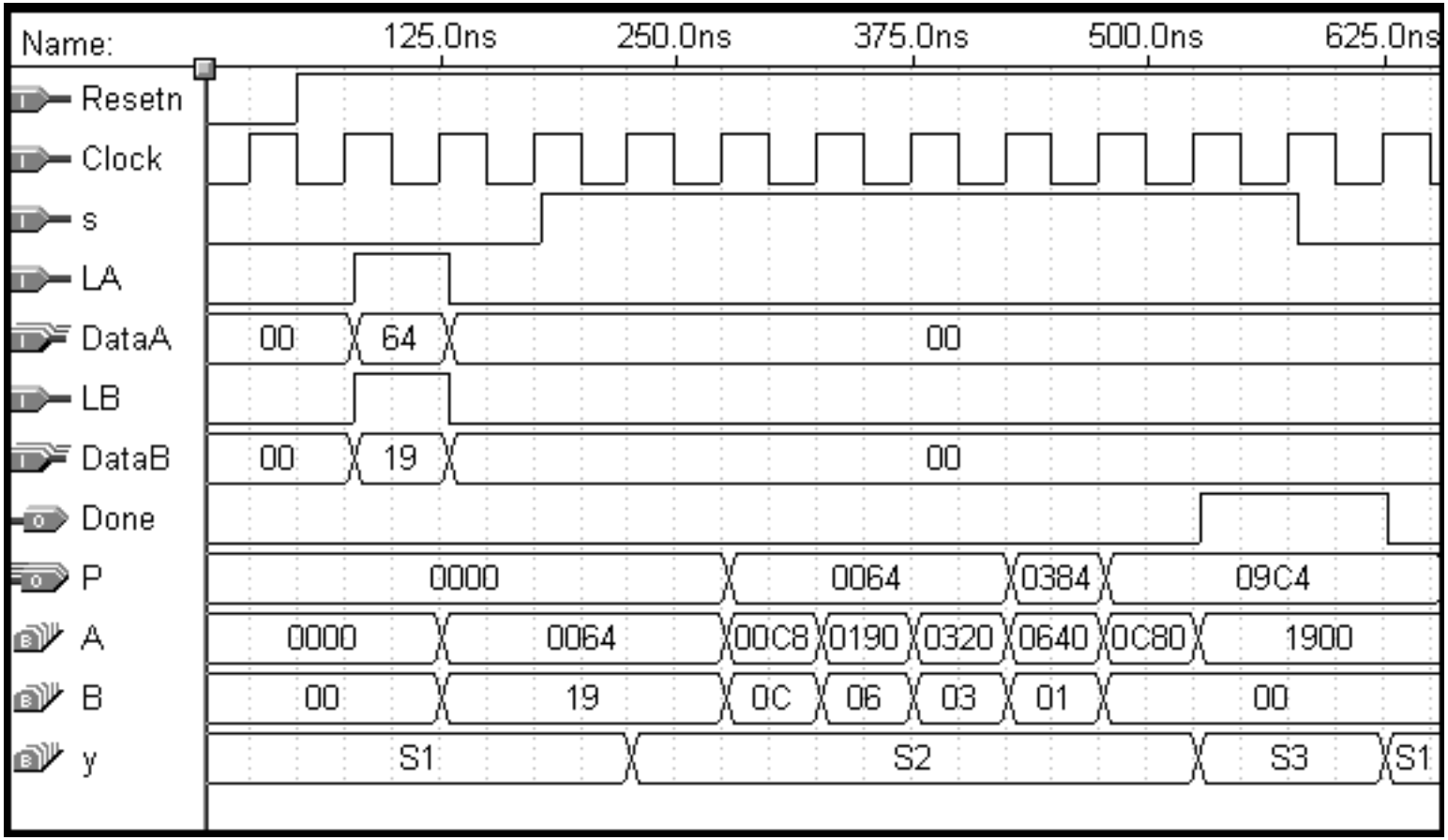
```
module multiply (Clock, Resetn, LA, LB, s, DataA, DataB, P, Done);
    parameter n = 8;
    input Clock, Resetn, LA, LB, s;
    input [n-1:0] DataA, DataB;
    output [n+n-1:0] P;
    output reg Done;
    wire z;
    reg [n+n-1:0] A, DataP;
    wire [n+n-1:0] Sum;
    reg [1:0] y, Y;
    reg [n-1:0] B;
    reg EA, EB, EP, Psel;
    integer k;

// control circuit

    parameter S1 = 2'b00, S2 = 2'b01, S3 = 2'b10;

    always @(s, y, z)
    begin: State_table
            case (y)
                S1:    if (s == 0) Y = S1;
                       else Y = S2;
                S2:    if (z == 0) Y = S2;
                       else Y = S3;
                S3:    if (s == 1) Y = S3;
                       else Y = S1;
                default: Y = 2'bxx;
            endcase
    end
```

```
always @(posedge Clock, negedge Resetn)
        begin: State_flipflops
                if (Resetn = = 0)
                        y <= S1;
                else
                        y <= Y;
        end
```

```
always @(s, y, B[0])
begin: FSM_outputs
    // defaults
    EA = 0; EB = 0; EP = 0; Done = 0; Psel = 0;
    case (y)
        S1: EP = 1;
        S2: begin
                EA = 1; EB = 1; Psel = 1;
                if (B[0]) EP = 1;
                else EP = 0;
            end
        S3: Done = 1;
    endcase
end
```

… continued in next slide…

```
//datapath circuit

        shiftrne ShiftB (DataB, LB, EB, 0,
Clock, B);
                defparam ShiftB.n = 8;
        shiftlne ShiftA ({{n{1'b0}}, DataA}, LA,
EA, 1’b0, Clock, A);
                defparam ShiftA.n = 16;

        assign z = (B = = 0);
        assign Sum = A + P;

        // define the 2n 2-to-1 multiplexers
        always @(Psel, Sum)
                for (k = 0; k < n+n; k = k+1)
                        DataP[k] = Psel ?
Sum[k] : 1’b0;

        regne RegP (DataP, Clock, Resetn, EP,
P);
                defparam RegP.n = 16;

endmodule
```



Simulation results for the multiplier circuit

F_{max} For A Multi-Cycle Multiplier

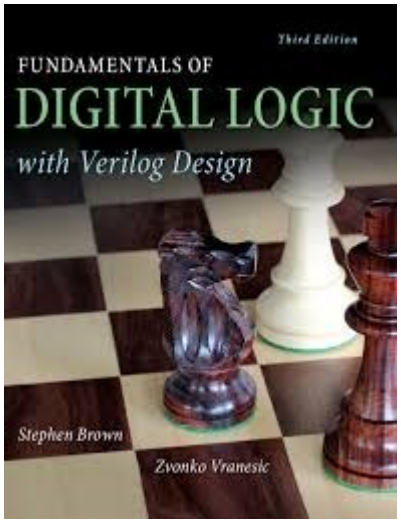
- Multi-Cycle Multiplier Fmax:
  - 64 bits x 64 bits: Fmax = 400 MHz
  - **Faster Fmax, but there is now LATENCY Takes up to N clock cycles to compute product**

- Compare with Combinational Multiplier

Multiplier Width (in bits)	F_{max} (MHz)
8 x 8	464
16 x 16	396
32 x 32	104
64 x 64	66

Recommended Reading

- Digital System Design with Verilog HDL, 3/e, b **S**tephen Brown and **Z**vonko Vranesic. [**S&Z**]
  - S&Z,
    - Chapter-3
      - 3.2



THANK YOU