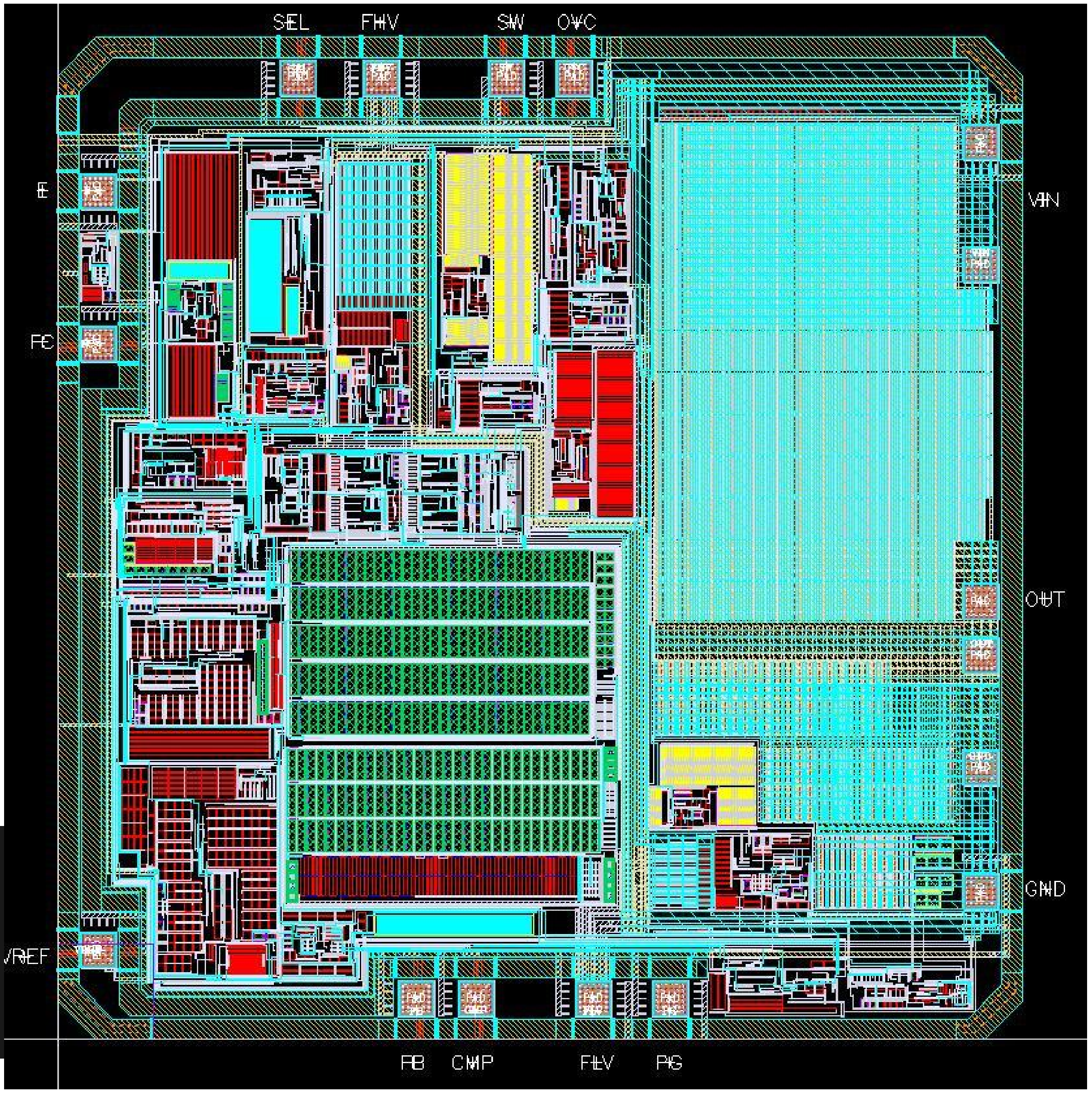
# ECE4514
# DIGITAL DESIGN 2

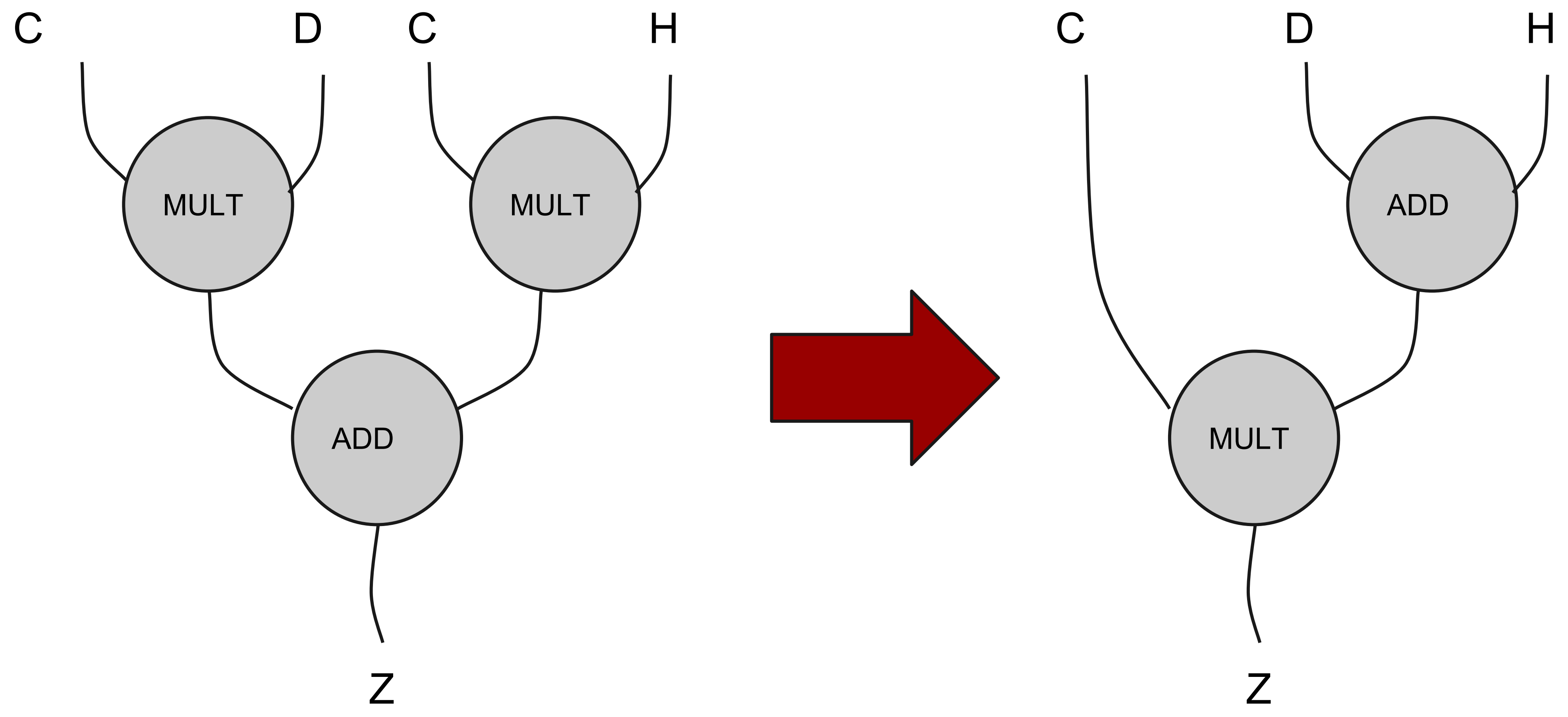


# AREA
## CHAPTER 2

## Scope

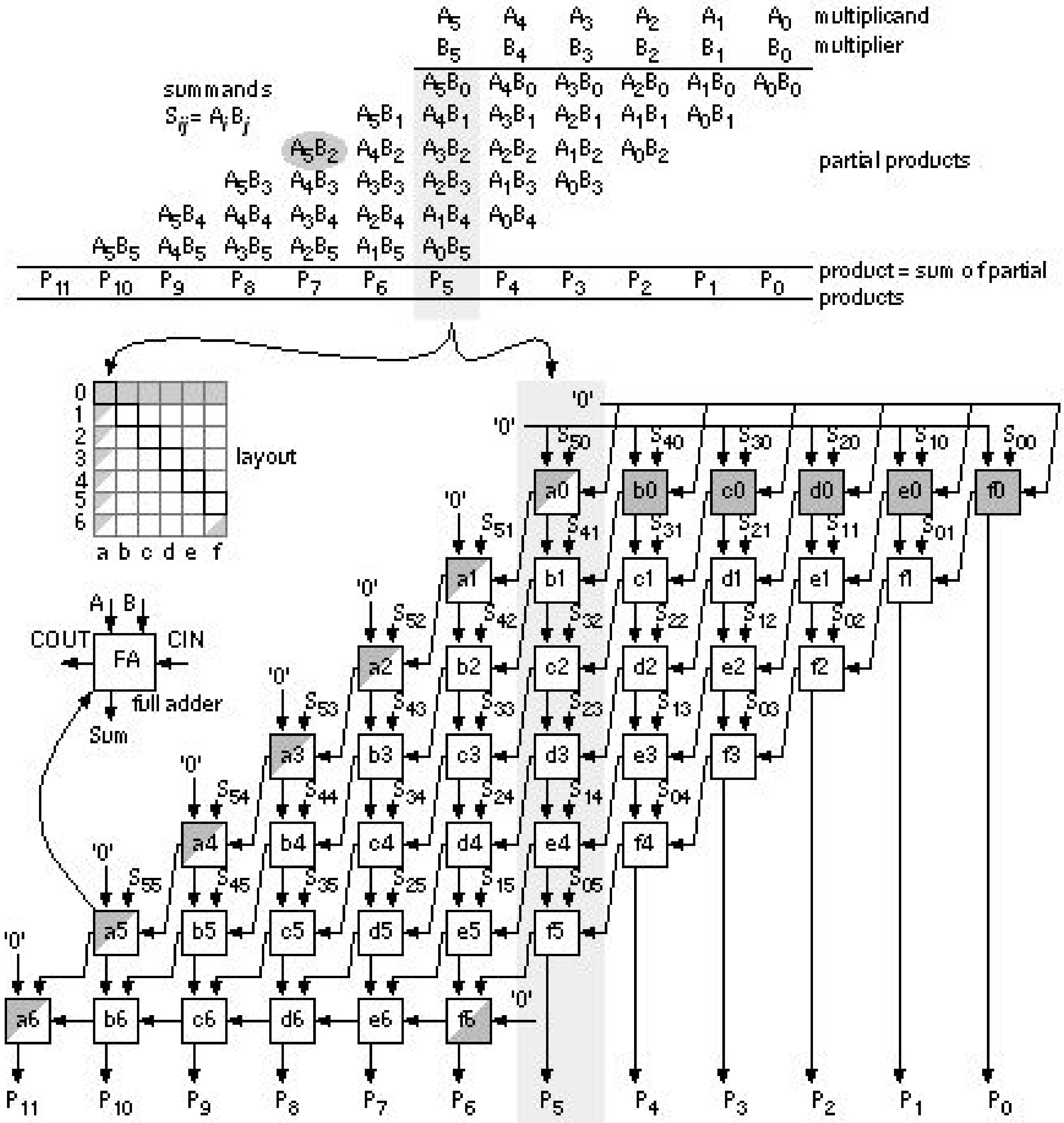
1. Rolling up the pipeline to reuse logic resources in different stages of a computation.
2. Controls to manage the reuse of logic when a natural flow does not exist.
3. Sharing logic resources between different functional operations.
4. The impact of reset on area optimization.
    - Impact of FPGA resources that lack reset/set capability.
    - Impact of FPGA resources that lack asynchronous reset capability.
    - Impact of RAM reset.

# Topology: z = c*d+c*h



Topology refers to the higher-level organization of the design and is not device specific.

# Multiplication



## Rolling Up the Pipeline

```
module mult8(
   output [7:0] product,
   input [7:0] A,
   input [7:0] B,
   input clk);
reg [15:0] prod16;

assign product = prod16[15:8];

always @(posedge clk)
      prod16 <= A * B;

endmodule
```

A is an integer, B is a fractional quantity

B scales A by [0..1)

# Weakening Mult to Serial

```
module mult8(
   output done,
   output reg [7:0] product,
   input [7:0] A,
   input [7:0] B,
   input clk,
   input start);

reg [4:0] multcounter; // counter for number of shift/adds
reg [7:0] shiftB; // shift register for B
reg [7:0] shiftA; // shift register for A

assign adden = shiftB[7] & !done;
assign done = multcounter[3];
```

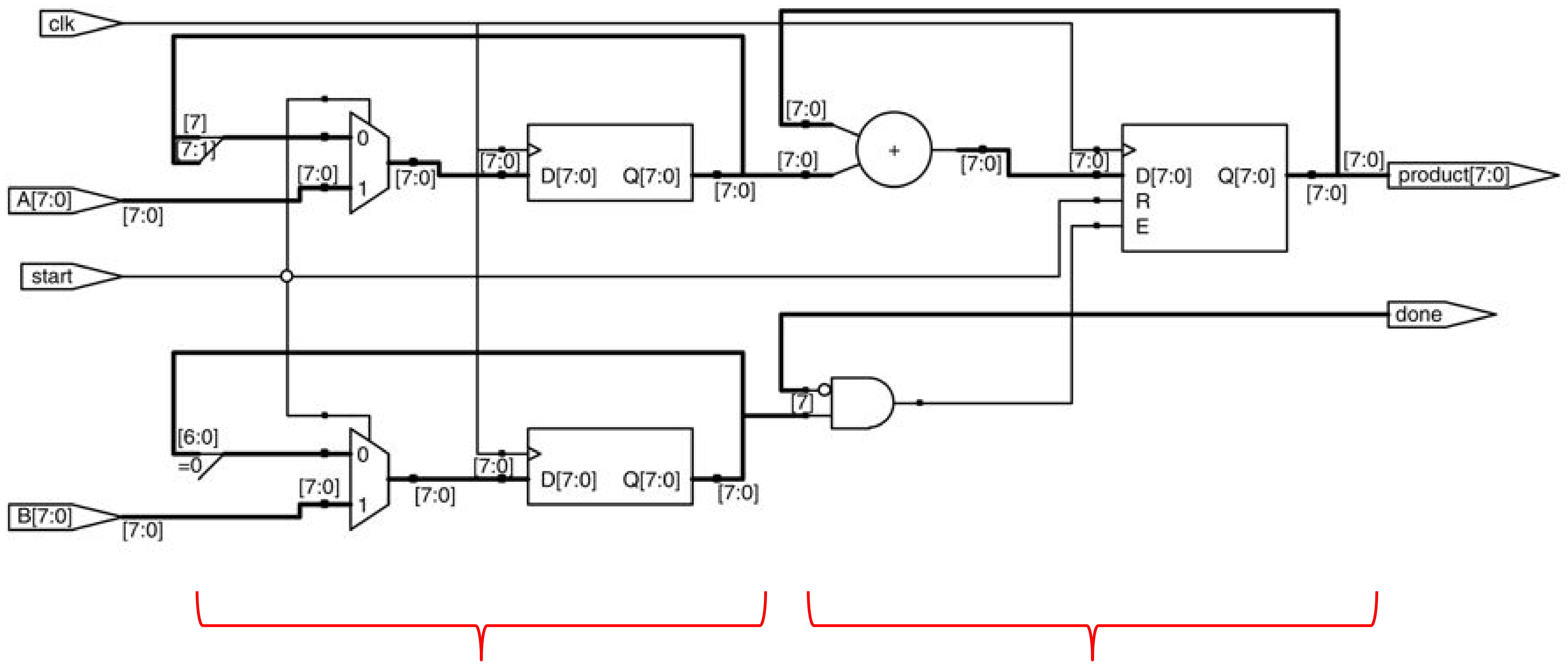
- 
- 
-

## Weakening Mult to Serial

- 
- 
-

```
always @(posedge clk) begin
// increment multiply counter for shift/add ops
   if(start) multcounter <= 0;
   else if(!done) multcounter <= multcounter + 1;
   // shift register for B
   if(start) shiftB <= B;
   else shiftB[7:0] <= {shiftB[6:0], 1’b0};
   // shift register for A
   if(start) shiftA <= A;
   else shiftA[7:0] <= {shiftA[7], shiftA[7:1]};
   // calculate multiplication
   if(start)      product <= 0;
   else if(adden) product <= product + shiftA;
end
endmodule
```

# Reduced Multiplier



SHIFT REGISTERS

ACCUMULATOR

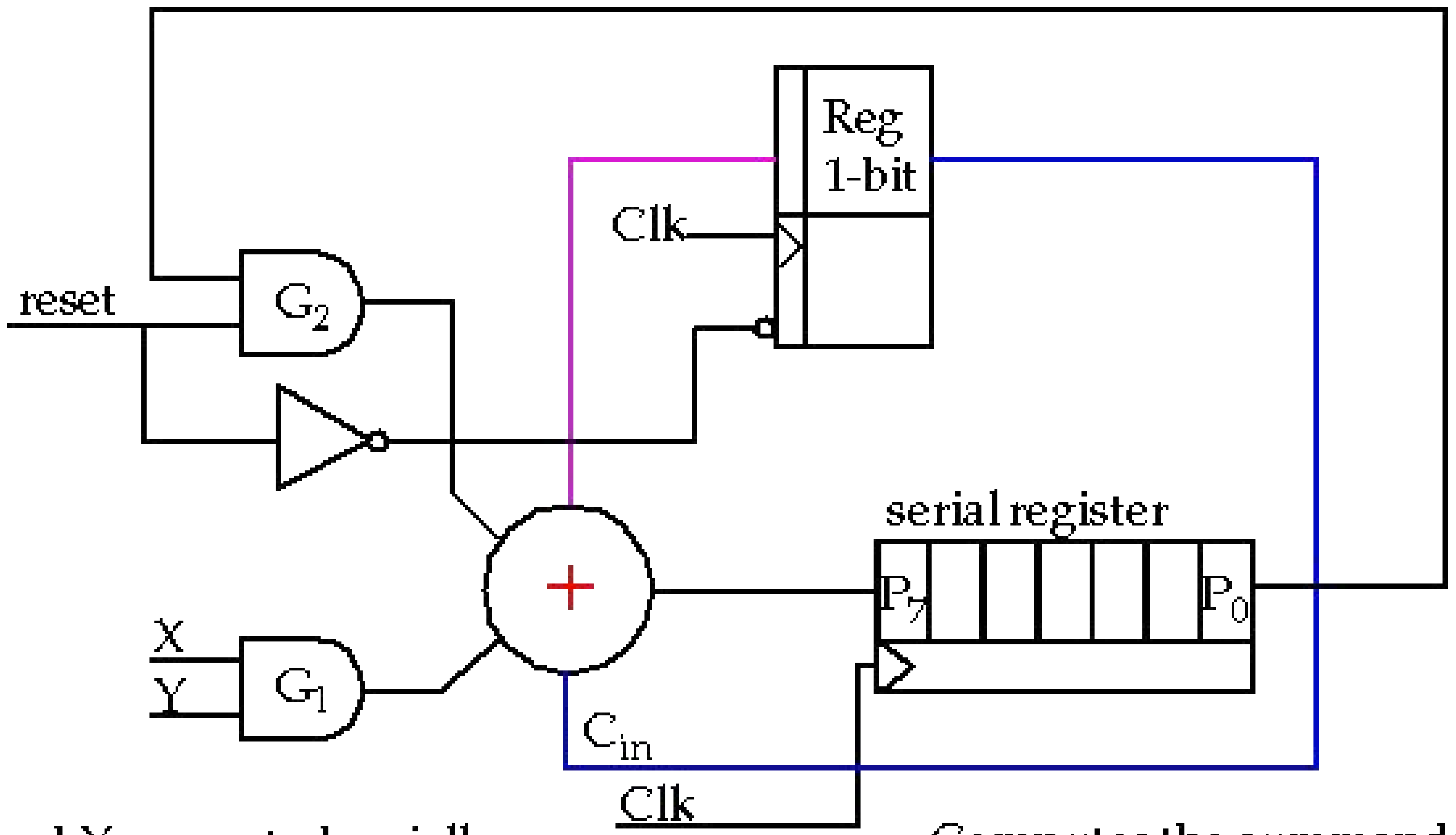
## Reduced Multiplier

Eliminated tree of adders with

- Two shift registers
- Adder
- Simple control

But requires 8 clock cycles instead of 0.

# Serial Multiplier



X and Y presented serially to the inputs of $G_1$ at different rates.

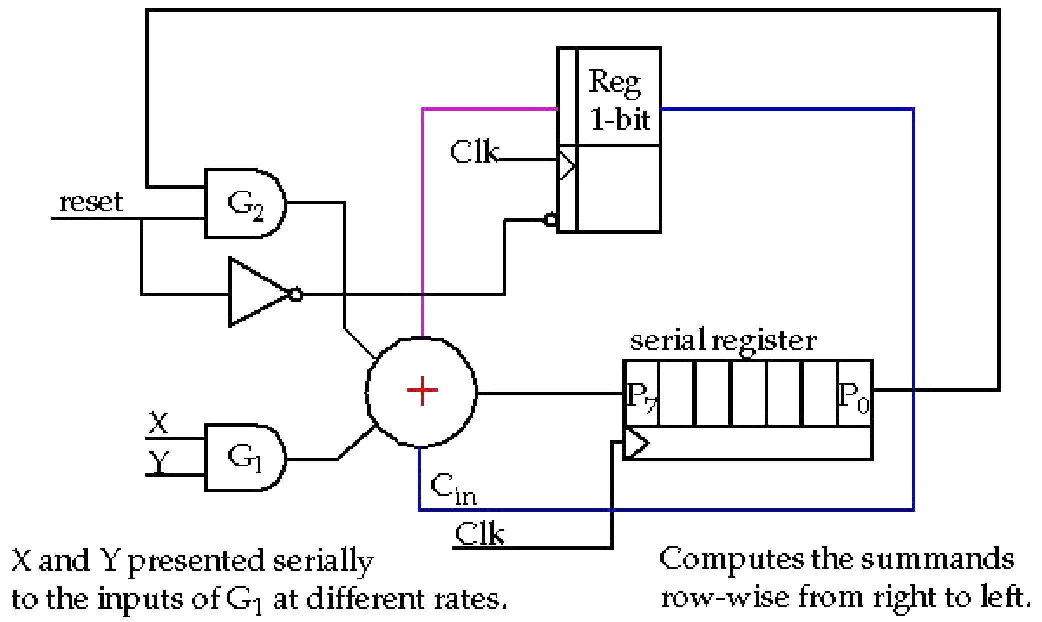
Computes the summands row-wise from right to left.

# Serial Multiplier

Full serial multiplier

Really small:

- One full-adder cell
- Some flip/flops
- Multiply performed by AND gate



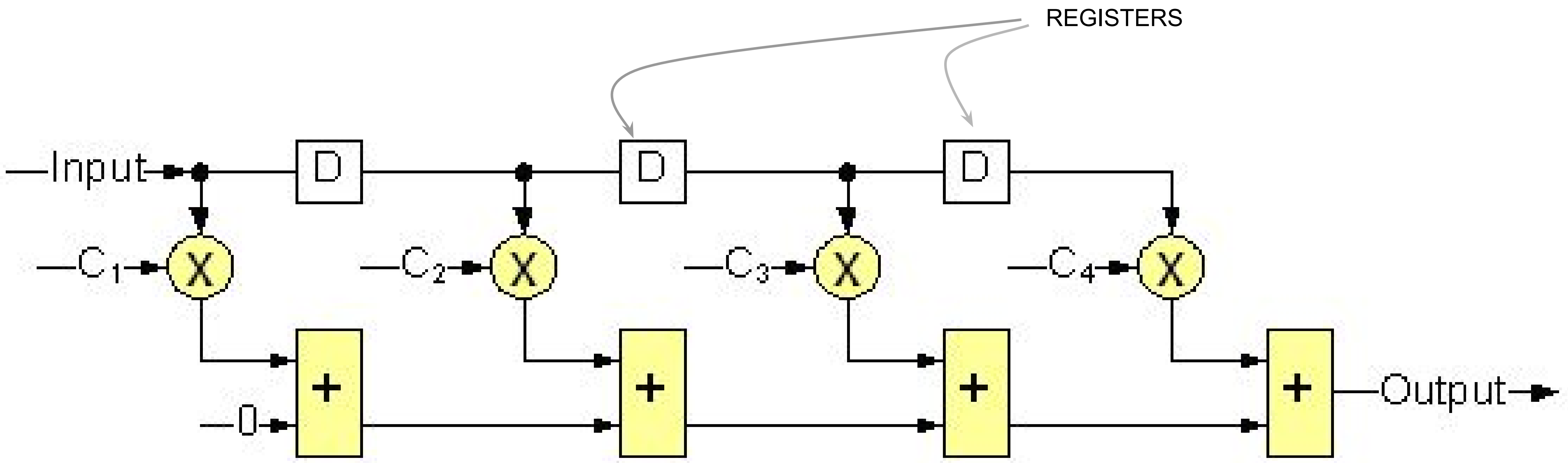
Reg 1-bit
Clk
reset
$G_2$
serial register
$P_7$
$P_0$
X
Y
$G_1$
+
$C_{in}$
Clk
X and Y presented serially to the inputs of $G_1$ at different rates.
Computes the summands row-wise from right to left.

Requires *N*N* clock cycles to compute

## CONTROL-BASED LOGIC REUSE

# FIR Filter

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$



$$Y = \text{coeffA} * X[0] + \text{coeffB} * X[1] + \text{coeffC} * X[2]$$

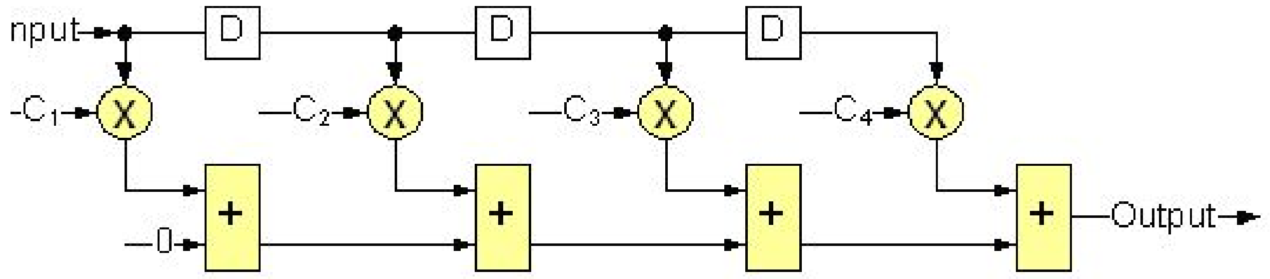
THIS WORKS, BUT WHY ISN'T THIS A GOOD DESIGN?

# FIR Filter

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

FROM CHAPTER 1

```
module fir(
  output [7:0] Y,
  input  [7:0] A, B, C, X,
  input        clk,
  input        validsample);
  reg    [7:0] X1, X2, Y;
  reg    [7:0] prod1, prod2, prod3;

  always @ (posedge clk) begin
    if(validsample) begin
      X1    <= X;
      X2    <= X1;
      prod1 <= A * X;
      prod2 <= B * X1;
      prod3 <= C * X2;
    end
    Y <= prod1 + prod2 + prod3;
  end
endmodule
```

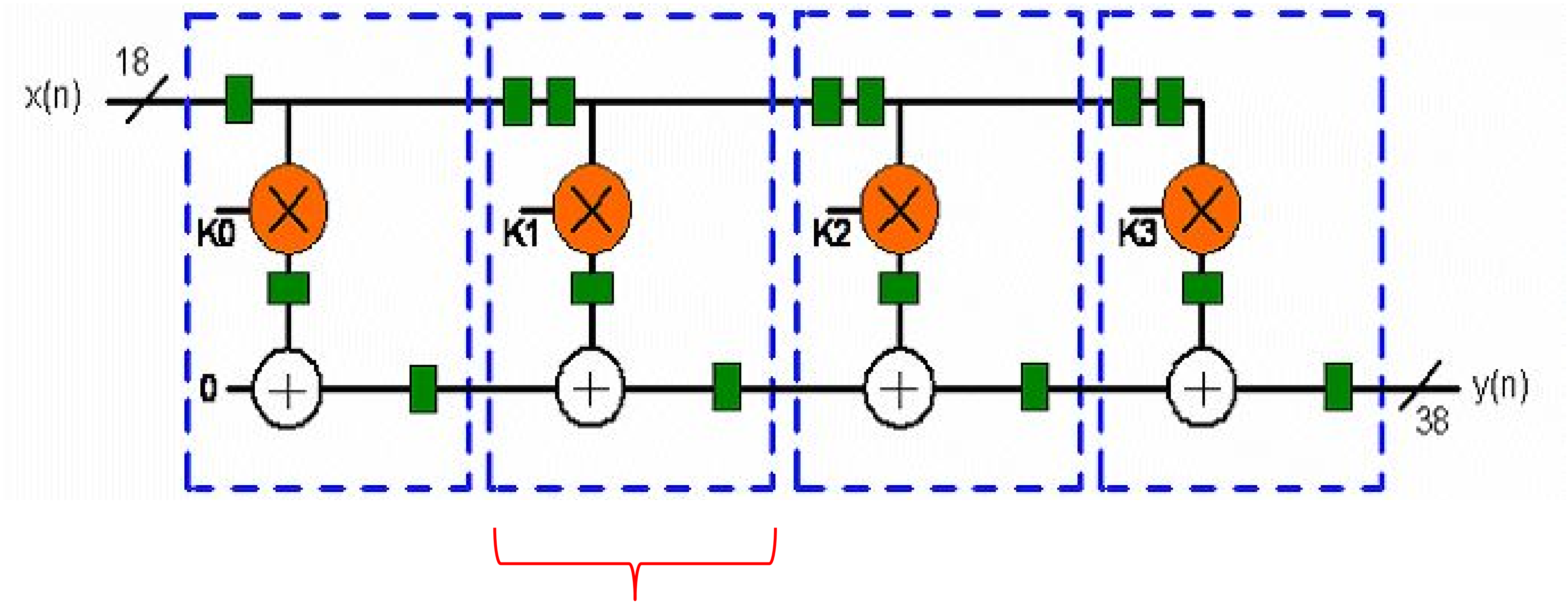


IN GENERAL, *N*-ORDER FIR REQUIRES:

- *N* multipliers
- *N* adders
- *N-1* registers

Throughput: 1 point per clock

# FIR: Trying to Get Smaller

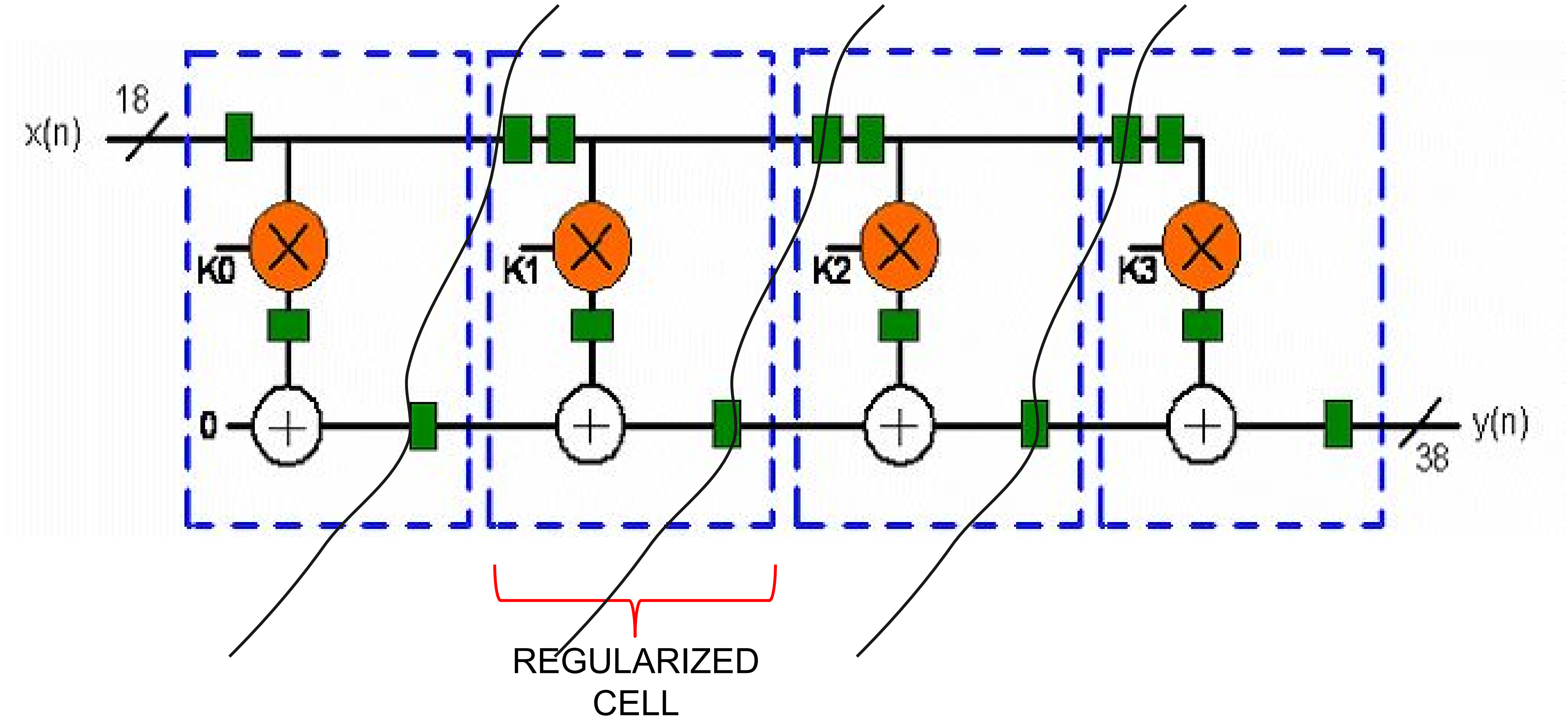


REGULARIZED
CELL

OBJECTIVE:

1. Regularize the overall structure into repeatable components
2. Reduce it to one component
3. Add control logic / multiplexers to sequence the data through it

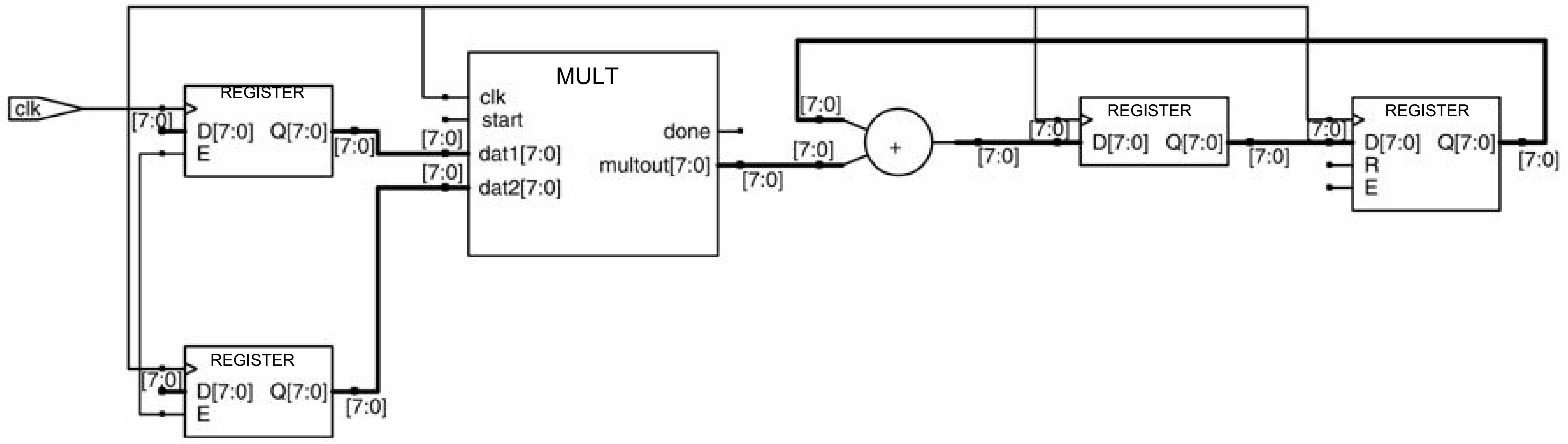
# FIR: Trying to Get Smaller



OBJECTIVE:

1. Regularize the overall structure into repeatable components
2. Reduce it to one component
3. Add control logic / multiplexers to sequence the data through it

## FIR: With One MAC



Refer to textbook for code.
Much more complex, requires FSM.

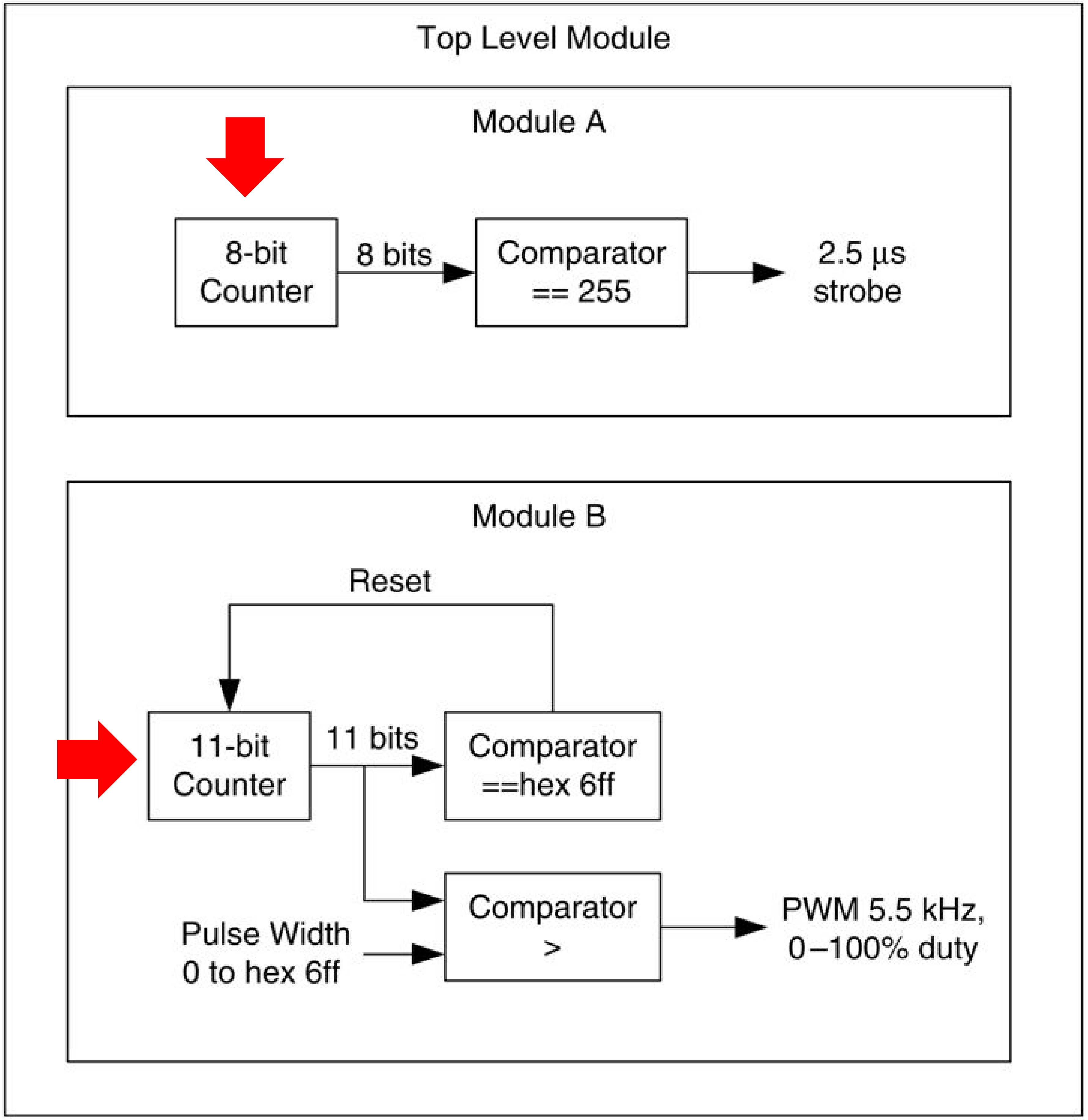
NOTE: MAC = Multiply-ACcumulate

## RESOURCE SHARING

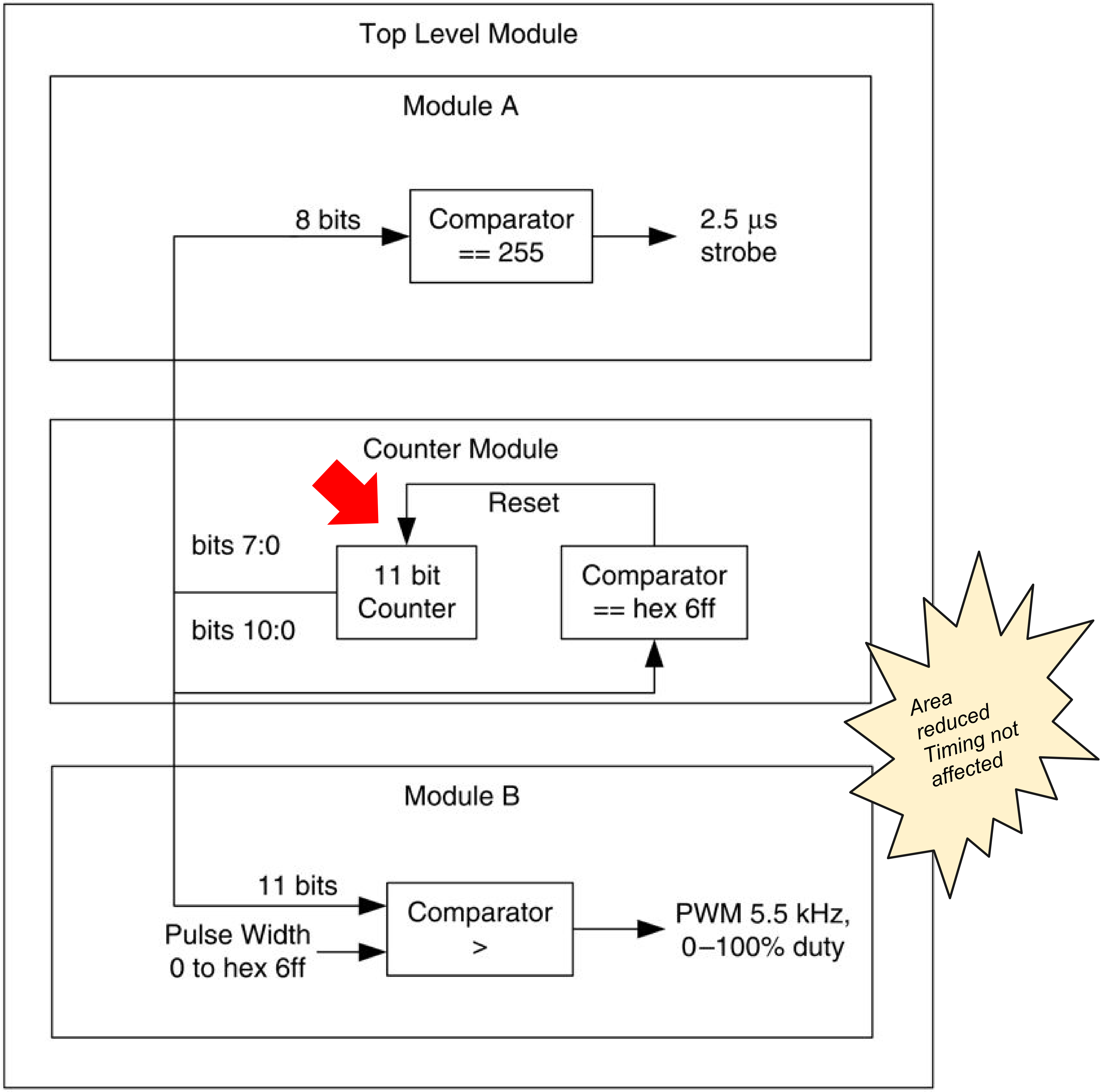
## Resource Sharing

*Higher-level architectural resource sharing where different resources are shared across different functional boundaries.*

## Common Counter



# Common Counter



## Resource Sharing

*For compact designs where area is the primary requirement, search for resources that have similar counterparts in other modules that can be brought to a global point in the hierarchy and shared between multiple functional areas.*

*If done properly, resource sharing will not impact temporal properties of a design.*

## EFFECTS OF RESET ON AREA

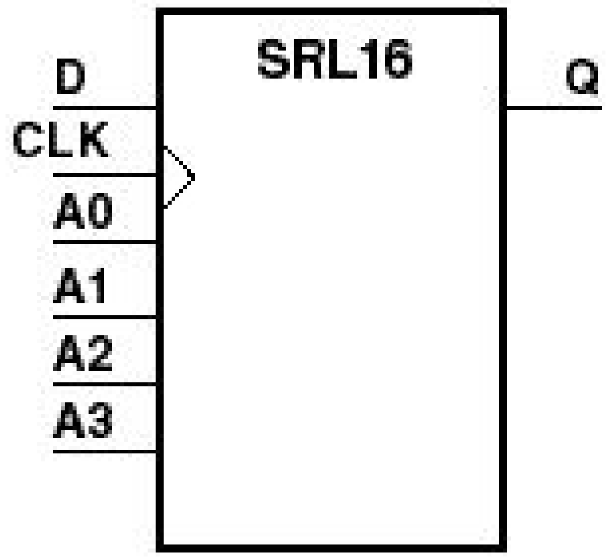
## Technology Dependencies

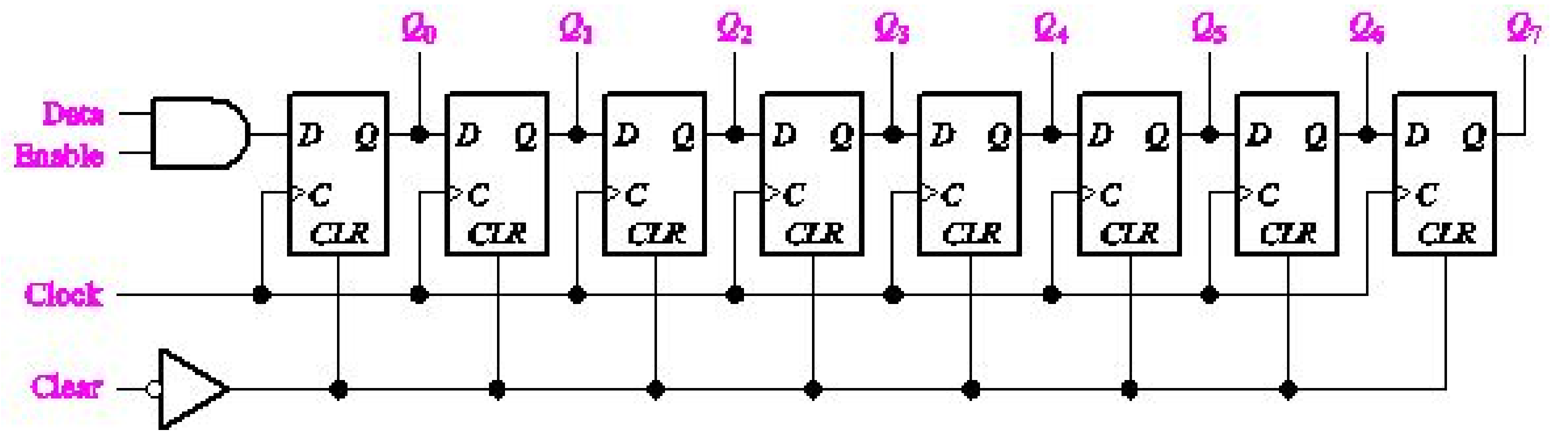
*In general, it is <u>always</u> a good idea to initialize the state of a machine (style violation if you don't).*

*This discussion pertains to the mapping of HDL code to particular target-dependent components. When the mapping is not a good fit, the amount of logic explodes.*

# Example 1: Shift Registers

SHIFTING IS COMMONPLACE IN DIGITAL DESIGN. VENDORS CREATED SPECIFIC COMPONENT TO DO THIS COMPACTLY:





ALMOST EQUIVALENT CIRCUIT

## Example 1: Shift Registers

IMPLEMENTATION 1 : *Synchronous Reset*

```
always @(posedge iClk)
 if(!iReset) sr <= 0;
 else sr        <= {sr[14:0], iDat};
```

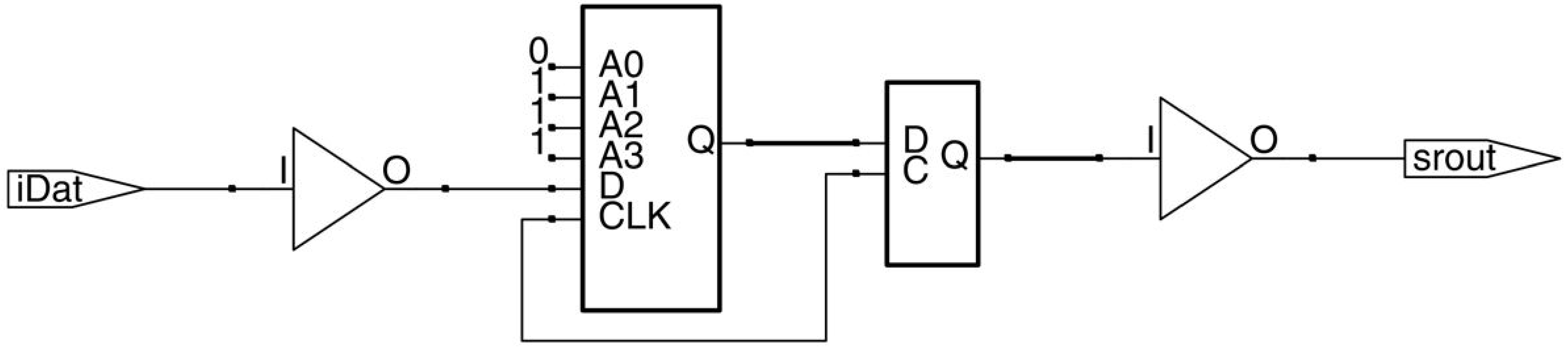
IMPLEMENTATION 2 : *No Reset*

```
always @(posedge iClk)
 sr <= {sr[14:0], iDat};
```

# Example 1: Shift Registers

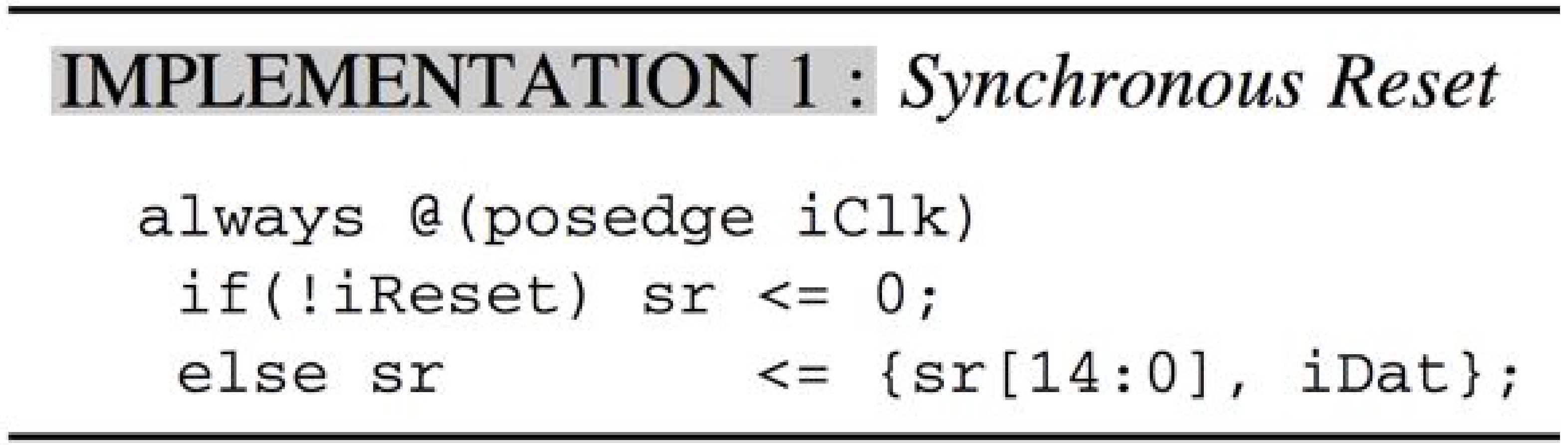
IMPLEMENTATION 2 : *No Reset*

```
always @(posedge iClk)
 sr <= {sr[14:0], iDat};
```



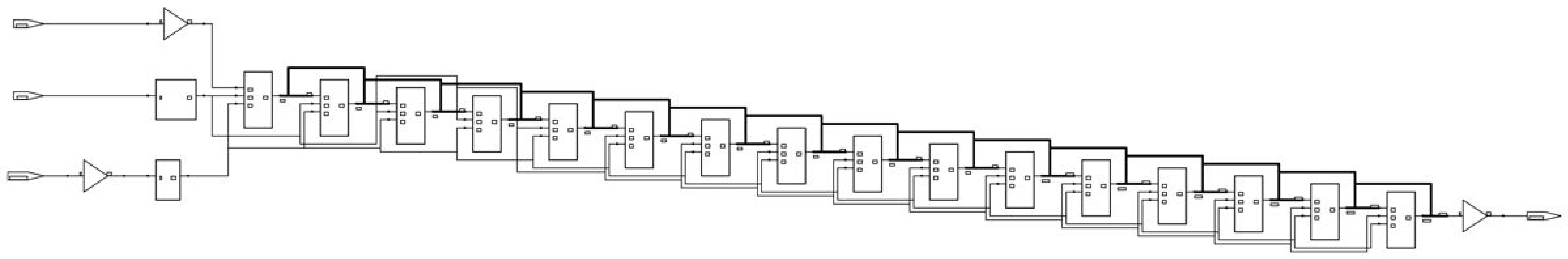
VERY SMALL AND COMPACT

# Example 1: Shift Registers



IMPLEMENTATION 1 : *Synchronous Reset*

```
always @(posedge iClk)
 if(!iReset) sr <= 0;
 else sr         <= {sr[14:0], iDat};
```



NOT SO SMALL AND COMPACT

Example 2: RAM

# LH5116/H

**CMOS 16K (2K × 8) Static RAM**

## FEATURES

- 2,048 × 8 bit organization
- Access time: 100 ns (MAX.)
- Power consumption:
  Operating: 220 mW (MAX.)
  Standby: 5.5 µW (MAX.)
- Single +5 V power supply
- Fully-static operation
- TTL compatible I/O
- Three-state outputs
- Wide temperature range available
  LH5116H: -40 to +85°C
- Packages:
  24-pin, 600-mil DIP
  24-pin, 300-mil SK-DIP
  24-pin, 450-mil SOP

## DESCRIPTION

The LH5116/H are static RAMs organized as 2,048 × 8 bits. It is fabricated using silicon-gate CMOS process technology. It features high speed access in read mode using output enable ($t_{OE}$).

## PIN CONNECTIONS

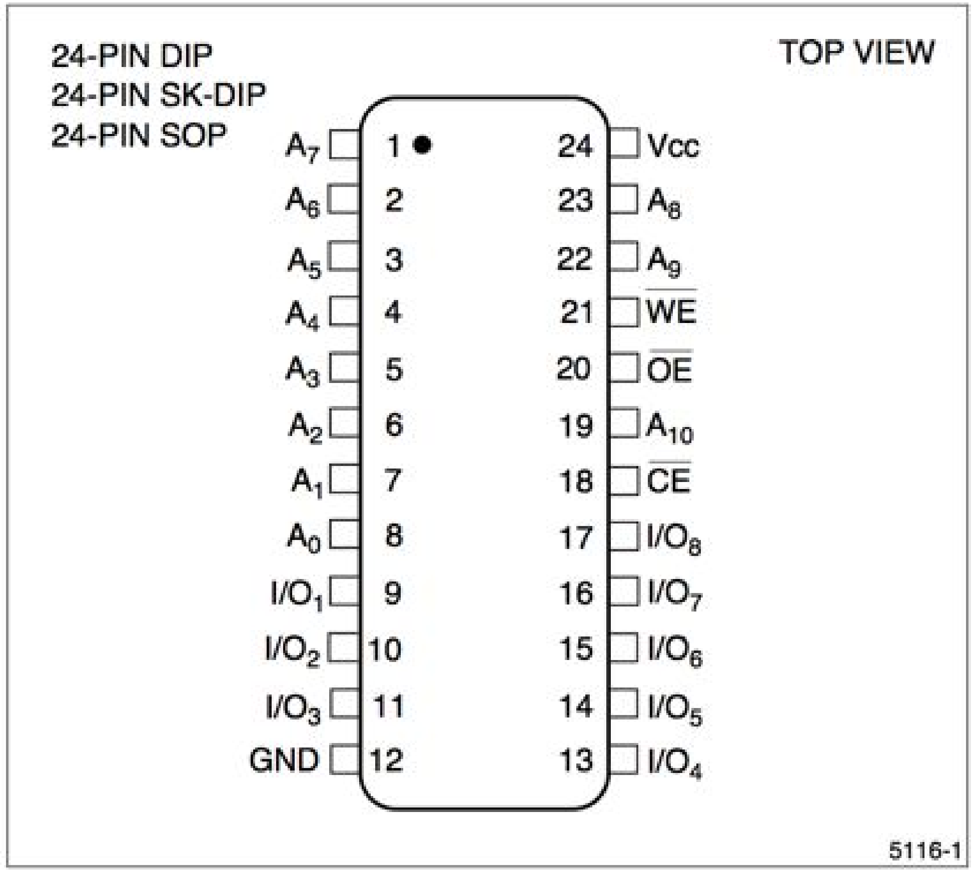


**Figure 1. Pin Connections for DIP, SK-DIP, and SOP Packages**

SO, HOW DO YOU RESET IT?

# Example 2: RAM

```
module resetckt(
output reg [15:0] oDat,
input iReset_n, iClk, iWrEn,
input [7:0] iAddr, oAddr,
input [15:0] iDat);
reg [15:0] memdat [0:255];

always @(posedge iClk or negedge iReset_n)
    if(!iReset_n) memdat <= 0;
    else begin
    if(iWrEn)
       memdat[iAddr] <= iDat;
    end
    oDat <= memdat[oAddr];
end
endmodule
```

WOW!

## Example 2: RAM

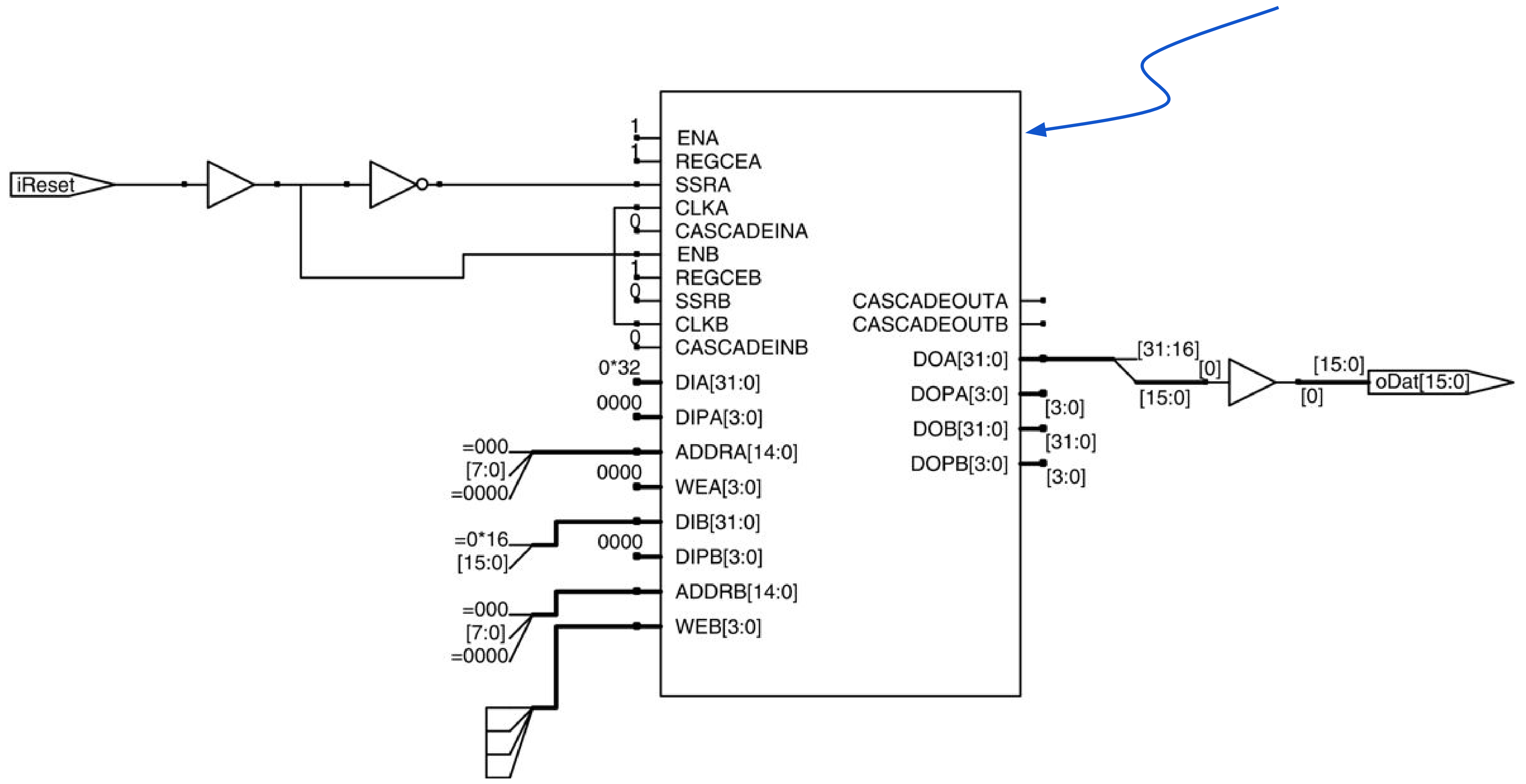
```
module resetckt(
output reg [15:0] oDat,
input iReset_n, iClk, iWrEn,
input [7:0] iAddr, oAddr,
input [15:0] iDat);
reg [15:0] memdat [0:255];

always @(posedge iClk)
    if(!iReset_n) oDat <= 0;
    else begin
    if(iWrEn)
       memdat[iAddr] <= iDat;
    end
    oDat <= memdat[oAddr];
end
endmodule
```

NOTE: Synchronous Reset

Maybe reasonable

# Example 2: RAM

FPGA-specific RAM Block



WITH SYNCHRONOUS RESET

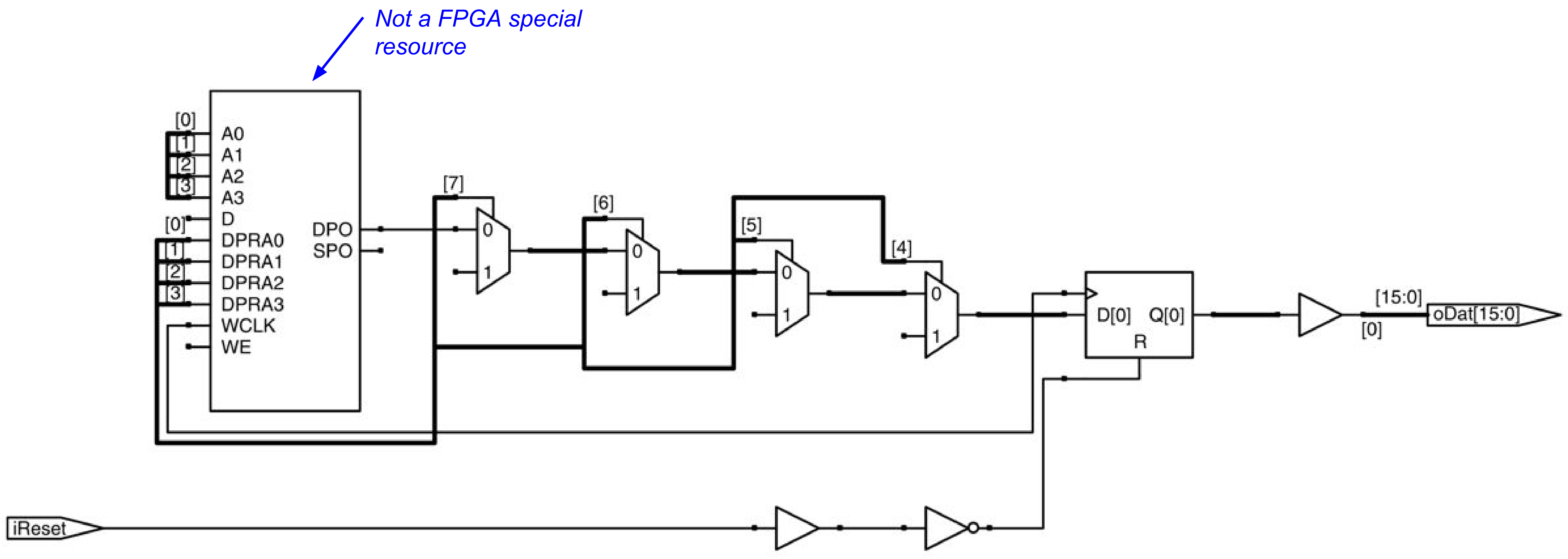
# Example 2: RAM

```
module resetckt(
output reg [15:0] oDat,
input iReset_n, iClk, iWrEn,
input [7:0] iAddr, oAddr,
input [15:0] iDat);
reg [15:0] memdat [0:255];

always @(posedge iClk or negedge iReset_n)
    if(!iReset_n) oDat <= 0;
    else begin
    if(iWrEn)
        memdat[iAddr] <= iDat;
    end
    oDat <= memdat[oAddr];
end
endmodule
```

NOTE: Asynchronous Reset

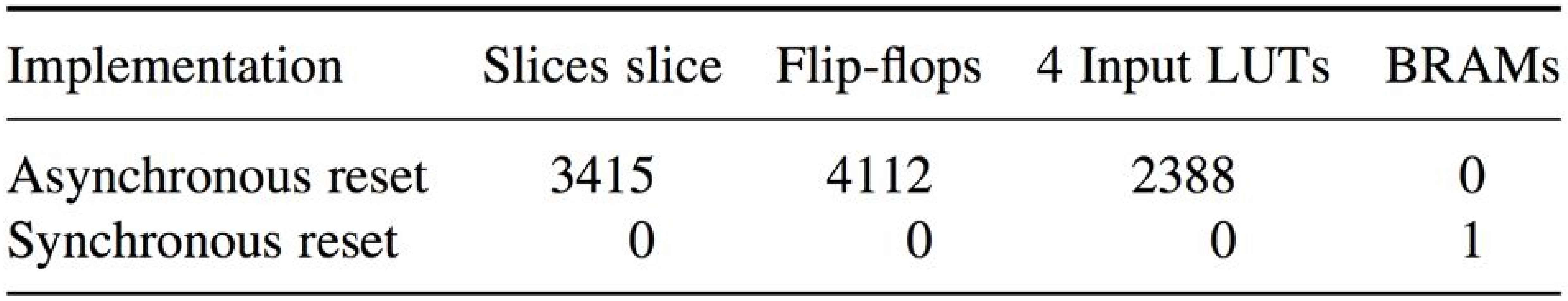
Maybe reasonable

# Example 2: RAM



WITH ASYNCHRONOUS RESET

# Example 2: RAM



| Implementation | Slices slice | Flip-flops | 4 Input LUTs | BRAMs |
|---|---|---|---|---|
| Asynchronous reset | 3415 | 4112 | 2388 | 0 |
| Synchronous reset | 0 | 0 | 0 | 1 |

RESOURCE EXPLOSION!

Generalizing: use synchronous reset vs asynchronous

## Summary

1. *Rolling up the pipeline can optimize the area of pipelined designs with duplicated logic in the pipeline stages.*
2. *Controls can be used to direct the reuse of logic when the shared logic is larger than the control logic.*
3. *For compact designs where area is the primary requirement, search for resources that have similar counterparts in other modules that can be brought to a global point in the hierarchy and shared between multiple functional areas.*
4. *An improper reset strategy can create an unnecessarily large design and inhibit certain area optimizations.*
5. *An optimized FPGA resource will not be used if an incompatible reset is assigned to it. The function will be implemented with generic elements and will occupy more area.*
6. *Improperly resetting a RAM can have a catastrophic impact on the area.*
7. *Using set and reset can prevent certain combinatorial logic optimizations.*

THE END

# Weakening Mult to Serial

-
-
-

```
always @(posedge clk) begin
   if (start) begin    // Begin multiplication
      multcounter <= 0;
      shiftB <= B;  shiftA <= A;
      product <= 0;
   else
      // increment multiply counter for shift/add ops
      if(!done) multcounter <= multcounter + 1;
      // shift register for B
      shiftB[7:0] <= {shiftB[6:0], 1'b0};
      shiftA[7:0] <= {shiftA[7], shiftA[7:1]};
      // calculate multiplication
      if(adden) product <= product + shiftA;
   end
endmodule
```