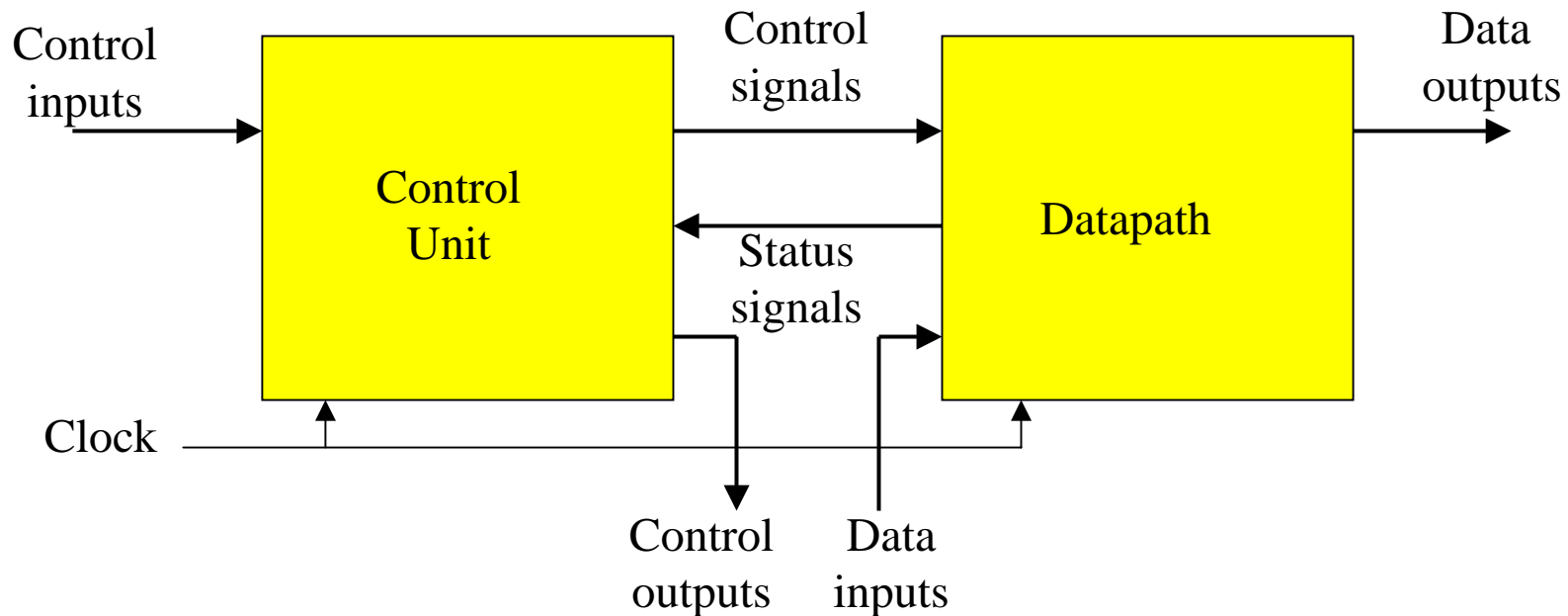


ECSE-323

Digital System Design

Datapath/Controller Lecture #2

Datapath/Controller architecture



General Approach to Design of Datapath/Controller Systems

1. Describe the function to be performed
2. Determine what datapath elements are needed
3. Specify the interconnections of the datapath elements
4. Identify the controller input and output signals
5. Sketch the sequence of control signal values needed to carry out the desired function
6. Design a Finite State Machine that will implement the required sequence
7. Simulate (by hand or by computer) the complete system to verify the proper execution of the desired function. Go back to step if there are any problems.
8. Implement and test complete system. Go back to step 2 if there are any problems with the implementation

Case Study #1 - Digital Filter

We wish to design recursive digital filter that implements the equation:

$$y(n) = a * y(n-1) + b * x(n) + c * x(n-1)$$

where the coefficients a, b, c are stored in registers.

You can have as many registers and multiplexers as you want, but assume that you only have available one multiplier and one adder block. Assume that an external input named *Sample* goes high whenever a new sample value on x is available.

STEP 1: Describe the function to be performed

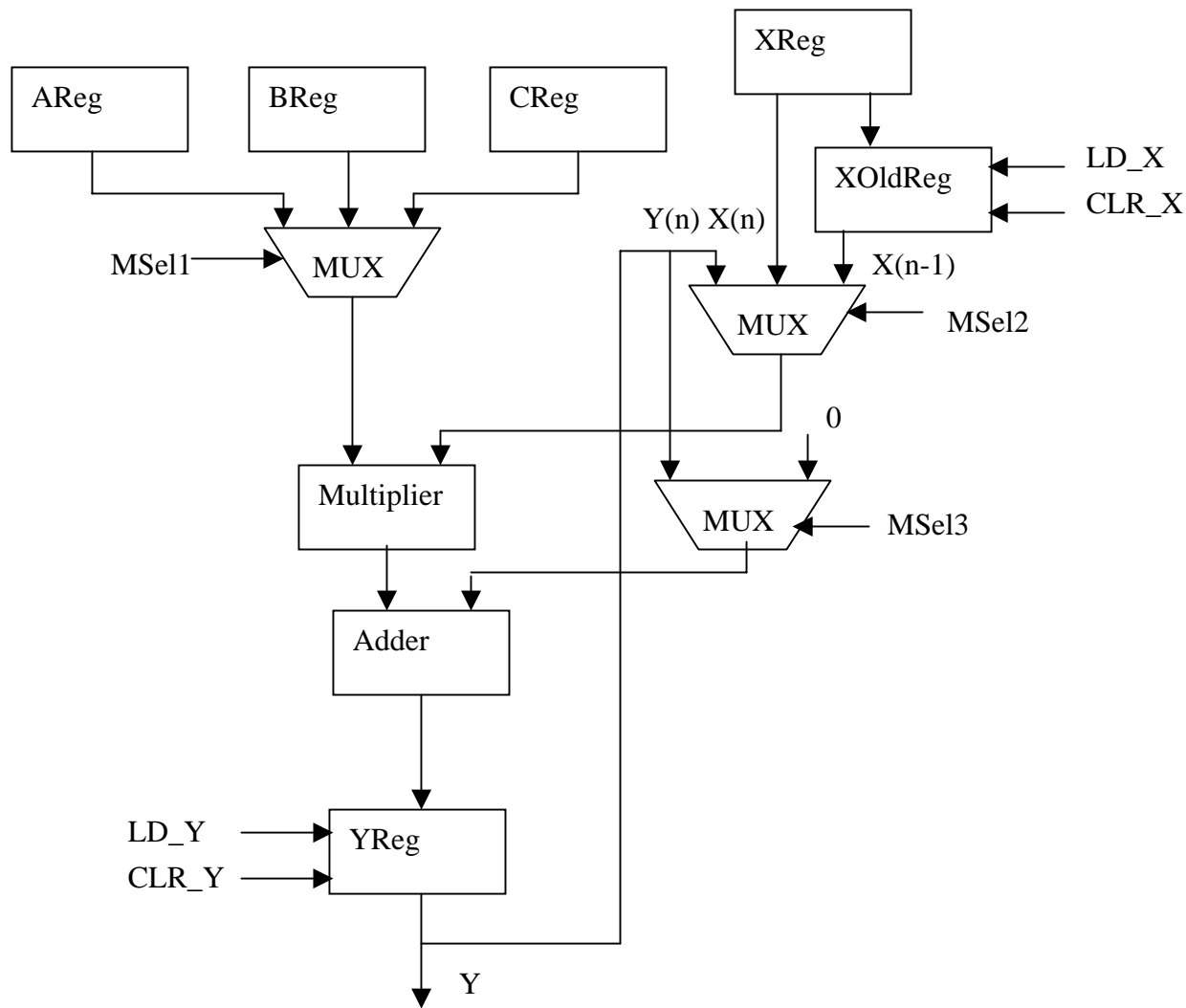
$$y(n) = a * y(n-1) + b * x(n) + c * x(n-1)$$

1. `y = y` (from previous cycle)
2. `y = 0 + a*y`
3. `y = y + b*x(n)`
4. `y = y + c*x(n-1)`

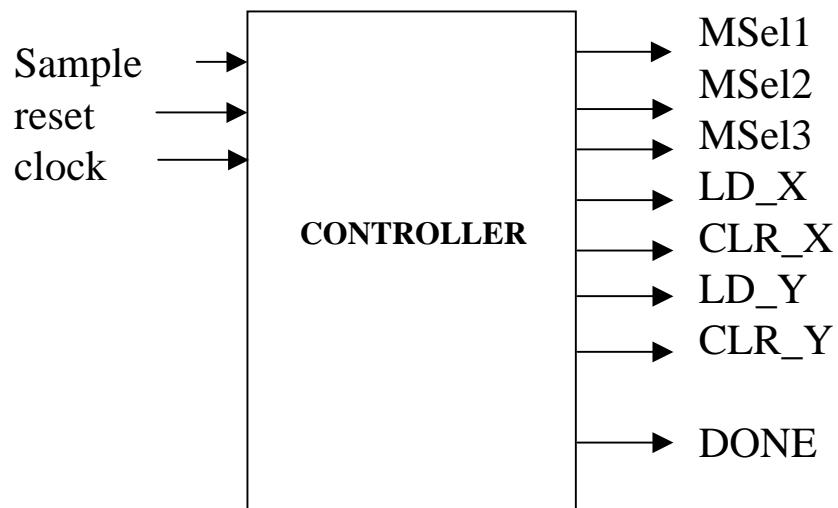
STEP 2: Determine the Datapath elements needed

- *Registers* to store $x(n), x(n-1), y(n), a, b, c$
- *Adder*
- *Multiplier (or combination Multiply-Accumulator)*
- *Multiplexers*

STEP 3: Specify the interconnections of the datapath



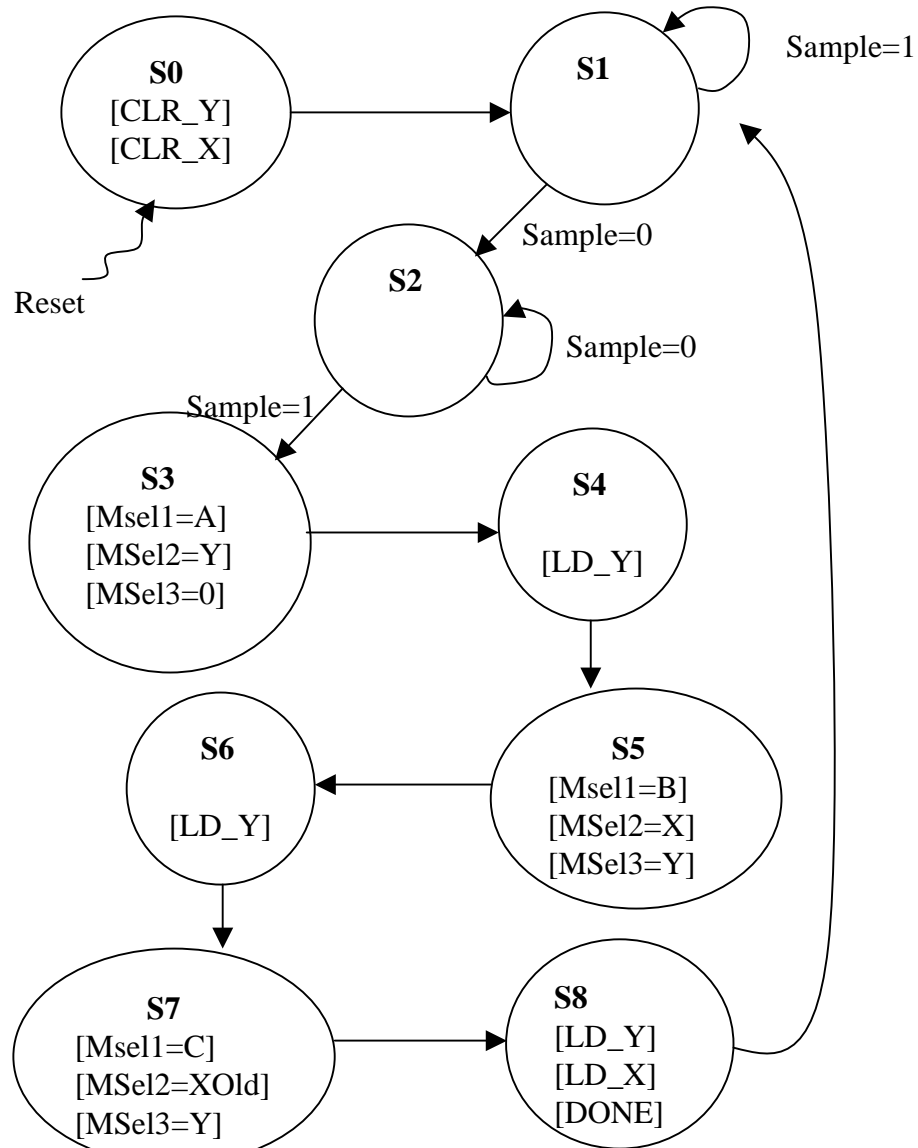
STEP 4: Identify the controller inputs and outputs



STEP 5: Sketch the sequence of control signal values

1. Wait for Sample to go low.
2. Wait for Sample to go high.
3. Set $Msel1$ to the a input, $Msel2$ to the Y input, and $MSel3$ to the 0 input (thus, the output of the adder will be $a*y(n-1)$).
4. Assert LD_Y to store the result of the multiply-accumulate into the output (Y) register.
5. Set $Msel1$ to the b input, $Msel2$ to the $X(n)$ input, and $MSel3$ to the Y input (thus, the output of the adder will be $y+b*x(n)$).
6. Assert LD_Y to store the result of the multiply-accumulate into the output (Y) register.
7. Set $Msel1$ to the c input, $Msel2$ to the $X(n-1)$ input, and $MSel3$ to the Y input (thus, the output of the adder will be $y+c*x(n-1)$).
8. Assert LD_Y to store the result of the multiply-accumulate into the output (Y) register. This will be the final result, so we also assert the $DONE$ signal. Finally, we assert the LD_X signal, so that the current X value becomes the old X value.

STEP 6: Design the FSM



Note: DONE will be high for only one clock cycle

STEP 7: Simulate the System

Left as an exercise for the student...

STEP 8: Debug and Modify the Design if needed

Is this design correct?

Case Study #2 - GCD Computation

We want to design a system that can compute the *Greatest Common Divisor* (GCD) of two binary numbers $(u,v) : g = \text{GCD}(u,v)$.

Example: $u=42 \quad v=99$
 $g = \text{GCD}(42,99) = 3$

the divisors of 42 are $(1,2,3,6,7,14,21)$

the divisors of 99 are $(1,3,9,11,33)$

3 is the largest divisor common to both numbers.

STEP 1: Describe the function to be performed

Here is an algorithm for finding the GCD:

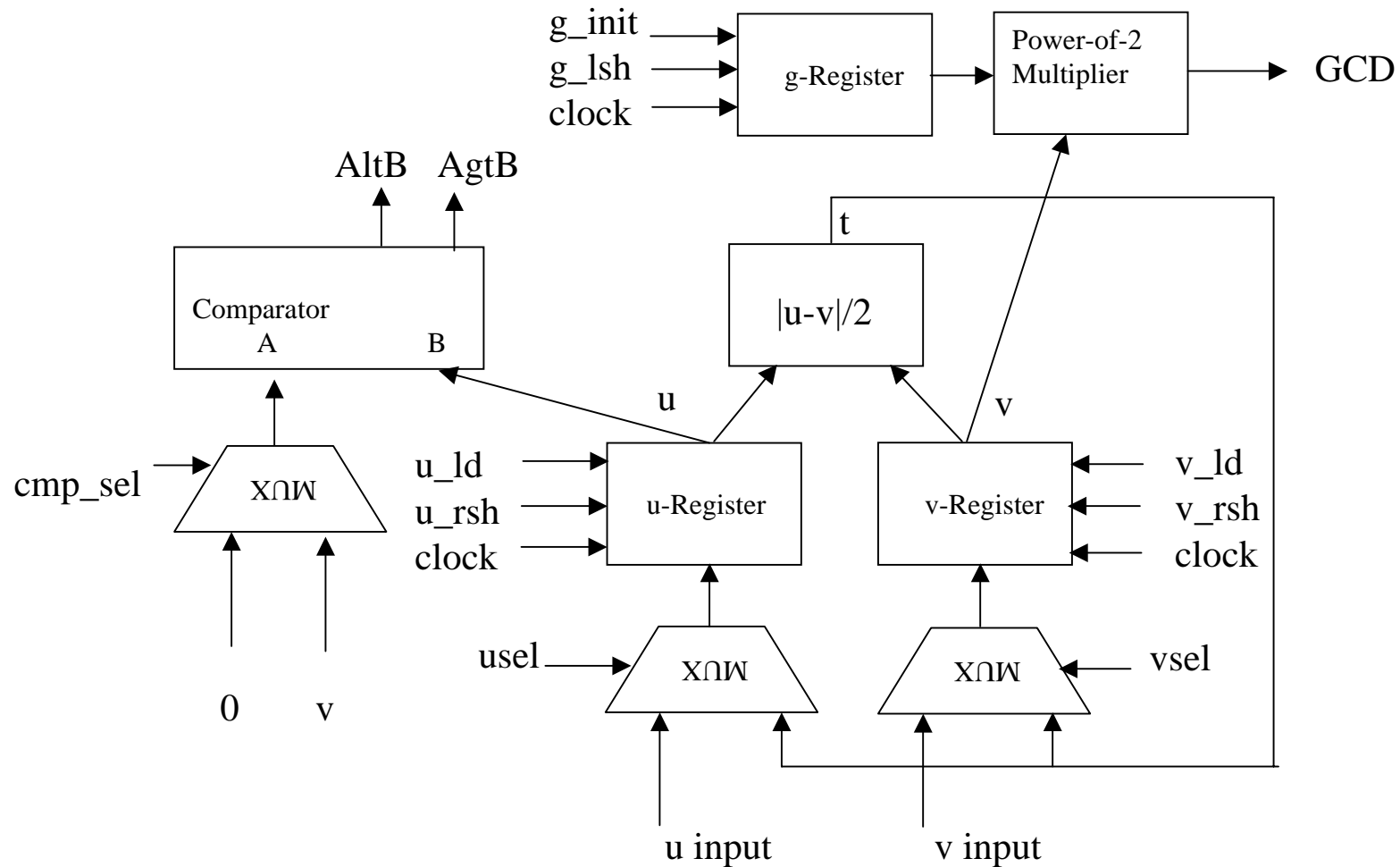
1. initialize $g = 1$
2. while ((u is even) and (v is even))
 - $u = u/2$ (right shift)
 - $v = v/2$
 - $g = 2*g$ (left shift)[now u or v (or both) are odd]
3. while ($u > 0$)
 - if (u is even), $u = u/2$
 - else if (v is even), $v = v/2$
 - else
 - $t = |u-v|/2$
 - if ($u < v$), then $v = t$ else $u = t$
4. return $g*v$

STEP 2: Determine the Datapath elements needed

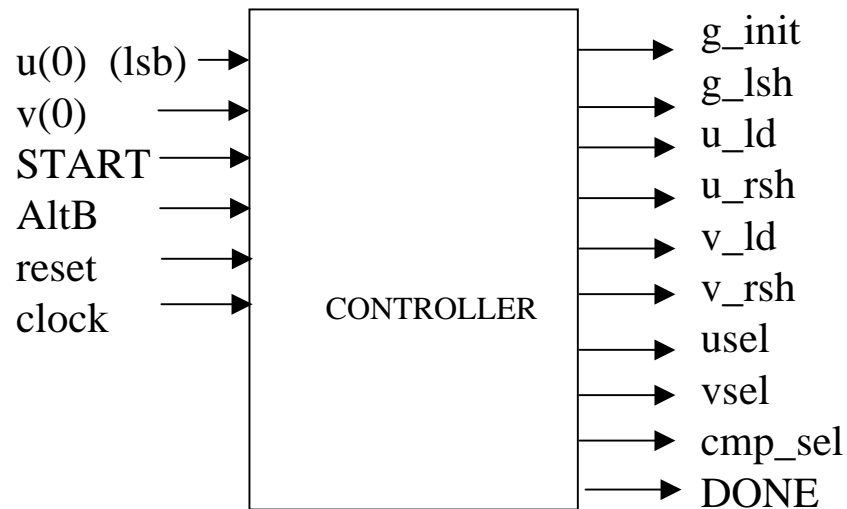
- *Shift registers* to store and divide u, v by 2
- *Shift register* to store and multiply g by 2
- *Comparator* to determine if $u < v$ and if $u < 0$
- *Absolute Difference module* to compute $|u - v|/2$
- *Multiplier* to compute $g * v$
- *Multiplexers* to select inputs to the u, v registers and to the comparator

We also need a way to determine if a data value is even. This can be done simply, by taking the inverse of the LSB.

STEP 3: Specify the interconnections of the datapath



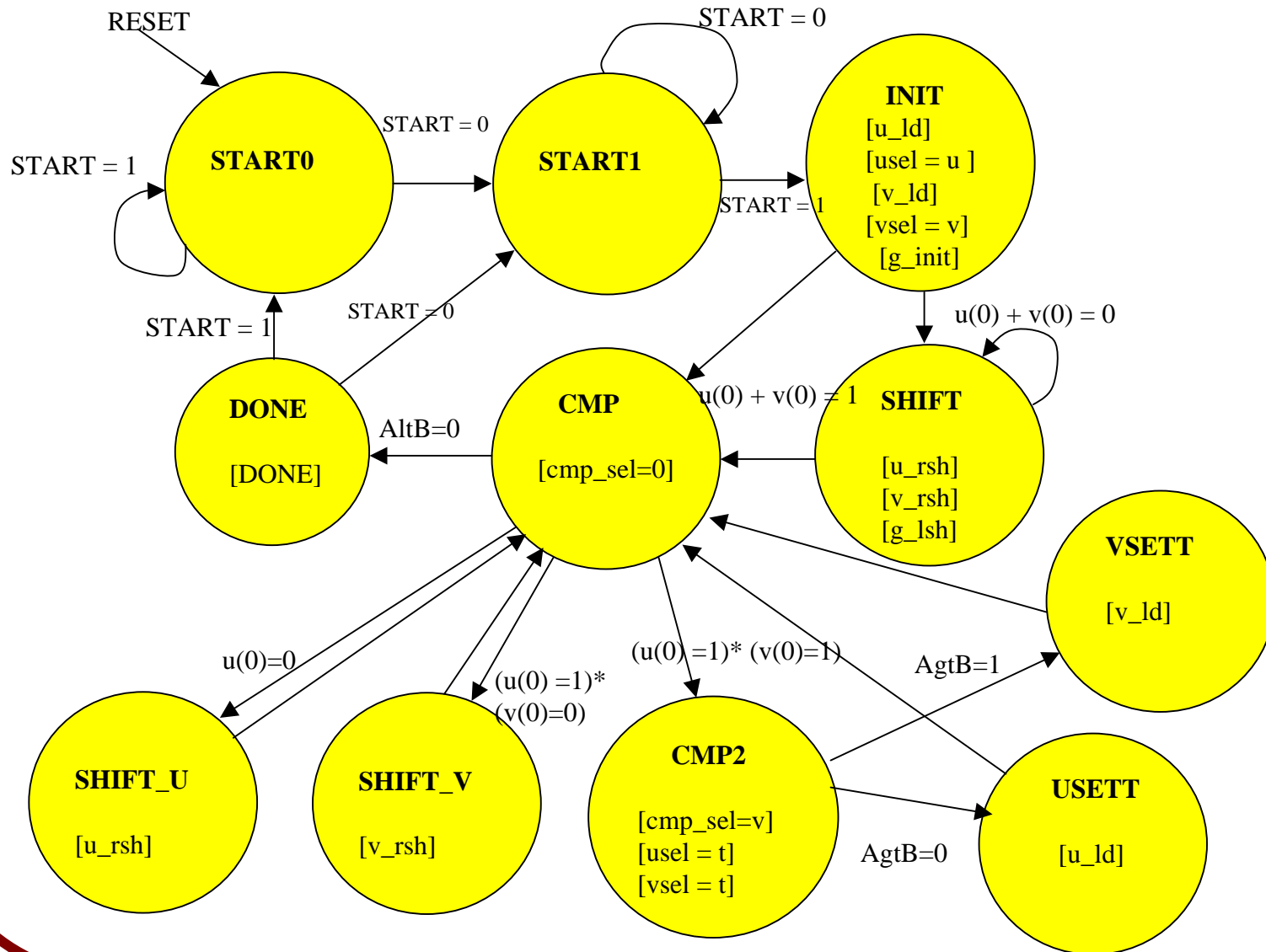
STEP 4: Identify the controller inputs and outputs



STEP 5: Sketch the sequence of control signal values

1. Wait for START to go low.
2. Wait for START to go high.
3. Initialize the g register to a value of 1 (assert g_init), load u and v inputs into u and v registers (set $usel$ to u input, $vsel$ to v input, assert u_ld and v_ld).
4. If ($u_lsb=0$ and $v_lsb=0$) then assert u_rsh , v_rsh , g_lsh and repeat step 4 else continue to step 5.
5. Set cmp_sel to select zero at input A of the comparator.
6. If $AltB=0$ (i.e. $u<0$) then assert $DONE$ (GCD output is now valid) else continue to step 7.
7. If $u_lsb=0$ then assert u_rsh and go to step 5 else go to step 8.
8. If $v_lsb=0$ then assert v_rsh and go to step 5 else go to step 9.
9. Set cmp_sel to select v at input A of the comparator, and set $usel$ and $vsel$ both to the t input.
10. If $AgtB=1$ (i.e. $u>v$) then assert v_ld , else assert u_ld . Go back to step 5.

STEP 6: Design the FSM



STEP 7: Simulate the System

Left as an exercise for the student...

STEP 8: Debug and Modify the Design if needed

Should be correct, but it is important to check...

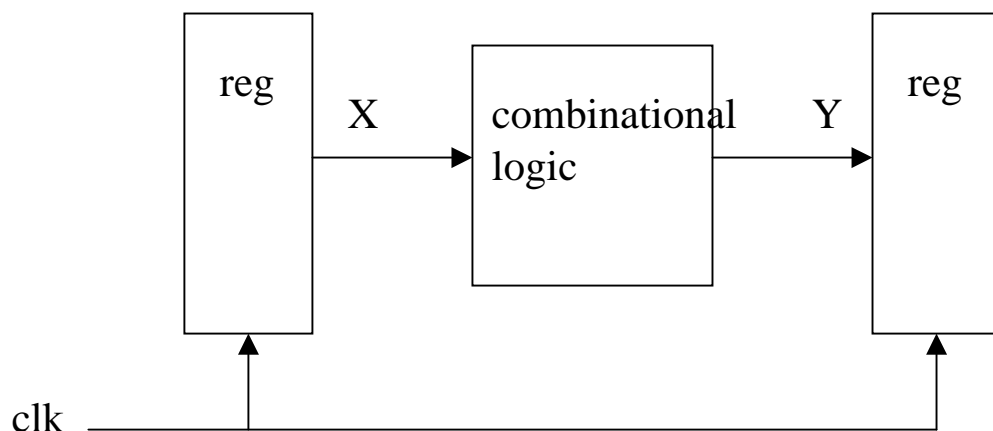
General Approach to Design of Datapath/Controller Systems

1. Describe the function to be performed
2. Determine what datapath elements are needed
3. Specify the interconnections of the datapath elements
4. Identify the controller input and output signals
5. Sketch out the sequence of control signal values needed to carry out the desired function
6. Design a Finite State Machine that will implement the required sequence
7. Simulate (by hand or by computer) the FSM to verify the proper execution of the desired function
8. Go back to step two if there are any problems with the implementation

Determining the Maximum Clock Rate

Generally, we want our digital system to run as fast as possible. This means running with as high a clock rate as possible.

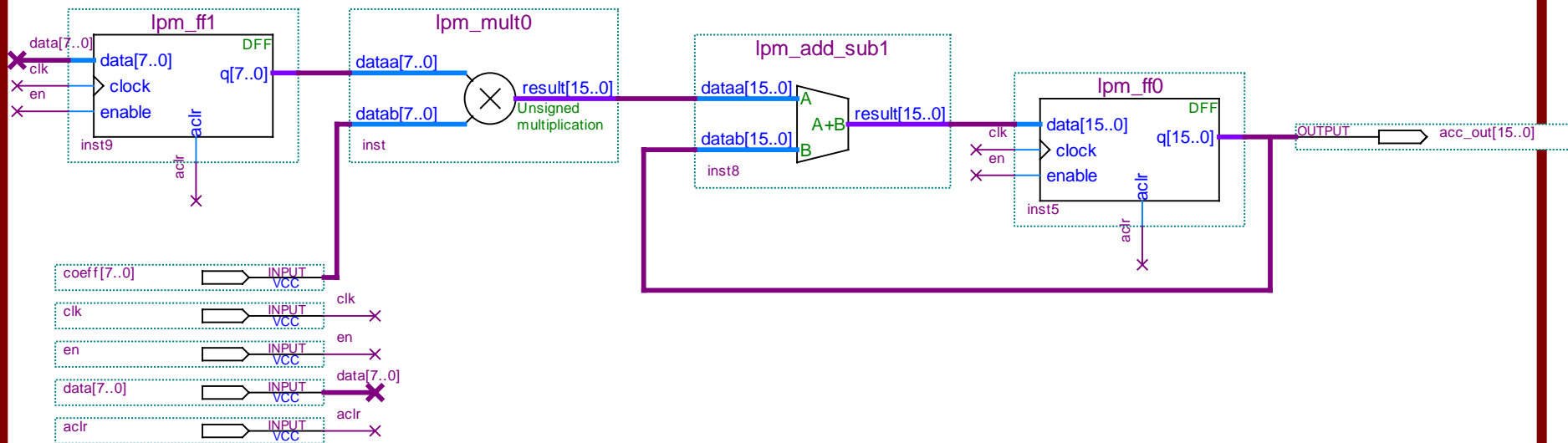
The maximum clock rate is determined by the longest propagation delay in the datapath (between two sequential elements).



Following a clock transition, the value of X may change. The next clock transition should not occur until the value of Y is stable.

Determining the Maximum Clock Rate

Example: Multiply-Accumulate circuit.



Timing Analyzer results for FLEX10K70RC240-4

The screenshot shows the 'Timing Analyzer Summary' window. The table below is a transcription of the data shown in the window.

Type	Slack	Required Time	Actual Time	From	To	From Clock
1 Worst-case tsu	N/A	None	87.100 ns	coeff[0]	lpm_ff0:inst5 lpm_ff:lpm_ff_component dffs[15]	--
2 Worst-case th	N/A	None	-0.400 ns	en	lpm_ff1:inst9 lpm_ff:lpm_ff_component dffs[2]	--
3 Worst-case tco	N/A	None	19.100 ns	lpm_ff0:inst5 lpm_ff:lpm_ff_component dffs[6]	acc_out[6]	clk
4 Total number of failed paths						
5 Clock Setup: 'clk'	N/A	None	13.64 MHz (period = 73.300 ns)	lpm_ff1:inst9 lpm_ff:lpm_ff_component dffs[0]	lpm_ff0:inst5 lpm_ff:lpm_ff_component dffs[15]	clk

The maximum propagation delay is **73.3 nsec** from the multiplier input to adder output.

The maximum clock rate is **13.64 MHz**.

The propagation delay of 73.3 nsec means that the clock period must be at least 73.3 nsec to allow the output of the adder enough time to settle before it is stored in the output register.

Thus, the maximum clock frequency is 13.64MHz.

In a synchronous sequential systems, *all* sequential elements should be clocked with the same signal (multiple clock domain techniques are available, but this is a topic outside the scope of this course).

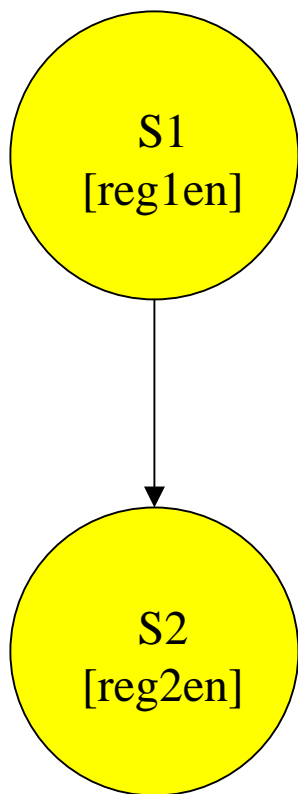
This means that all circuits must be clocked at 13MHz, even if their propagation delays are less than 73nsec.

It would be nice to have a way to let some parts of the system run faster than others, but still use a single clock.

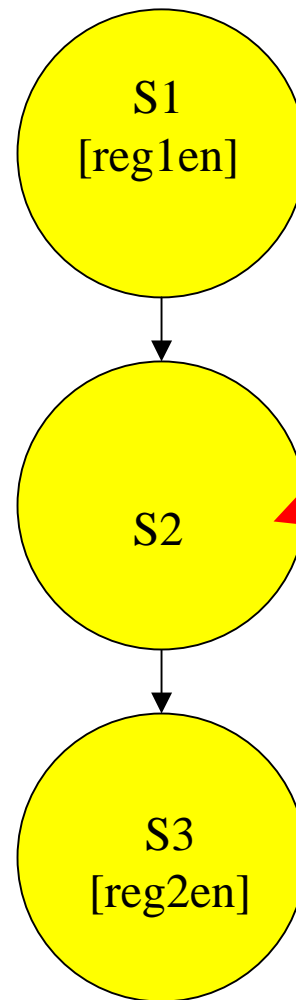
The way to do this is through the use of *Wait States*.

A wait state is an extra state in which nothing happens, and whose purpose is to provide an extra clock period's worth of time to allow signals to settle before they are operated on.

For example, suppose that most of our system could run off of a 40nsec clock, but the multiply-accumulate circuit needs 73nsec between the setting of the input multiplexer select and the loading of the output register. This can be done by using a 25MHz global clock, and adding an extra state between the enabling of the first register and the enabling of the second register.



runs at 13MHz



runs at 25MHz

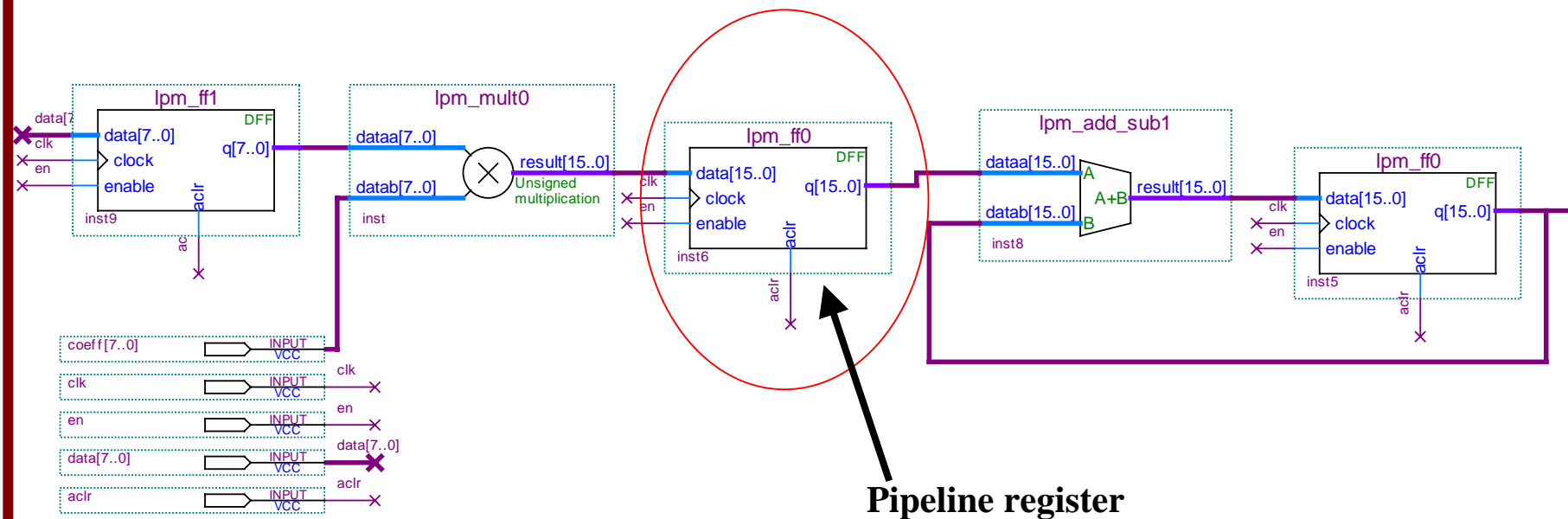
Pipelining

Another approach to speeding up the system is pipelining.

This technique is based on the fact that the maximum clock speed depends on the longest propagation delay between two registers (the so-called critical path).

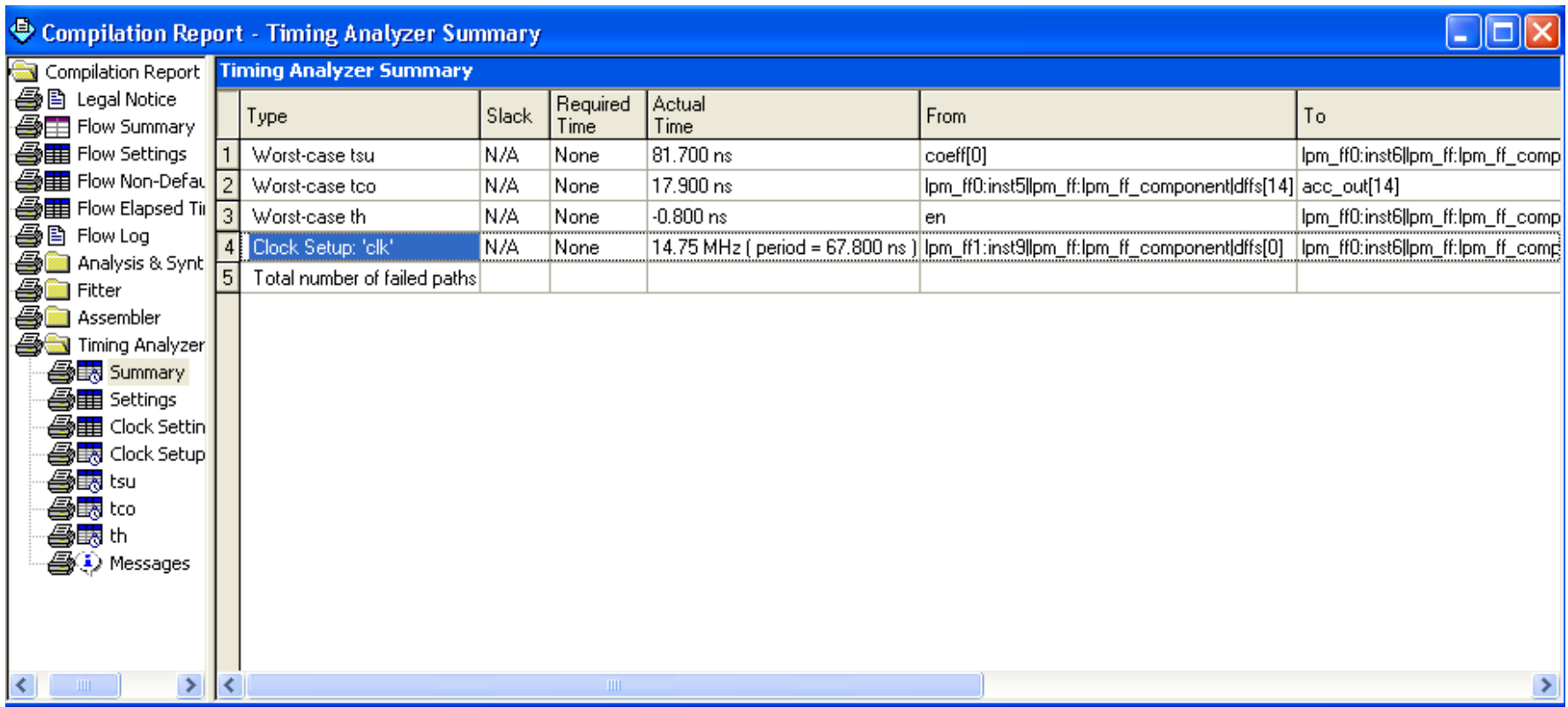
If the critical path contains circuit that can be divided into two (as in our multiply-accumulate example) then we can insert a register at the dividing point. This will reduce the maximum delay time, thereby increasing the maximum clock speed.

Example: Pipelined Version of the Multiply-Accumulate circuit.



The extra register introduces a *latency*, but actually results in a speedup

Timing Analyzer results for FLEX10K70RC240-4



Compilation Report - Timing Analyzer Summary

Timing Analyzer Summary

Type	Slack	Required Time	Actual Time	From	To
1 Worst-case tsu	N/A	None	81.700 ns	coeff[0]	lpm_ff0:inst6 lpm_ff:lpm_ff_comp
2 Worst-case tco	N/A	None	17.900 ns	lpm_ff0:inst5 lpm_ff:lpm_ff_component dffs[14]	acc_out[14]
3 Worst-case th	N/A	None	-0.800 ns	en	lpm_ff0:inst6 lpm_ff:lpm_ff_comp
4 Clock Setup: 'clk'	N/A	None	14.75 MHz (period = 67.800 ns)	lpm_ff1:inst9 lpm_ff:lpm_ff_component dffs[0]	lpm_ff0:inst6 lpm_ff:lpm_ff_comp
5 Total number of failed paths					

The maximum propagation delay has dropped to ***67.8 nsec.***

The maximum clock rate has increased to ***14.75 MHz.***

In this example the maximum clock rate did not increase very much, since the multiplier has most of the delay in the critical path.

For other circuits the delay might be more evenly spread and there may be more places in which we could insert registers. This may provide a significant increase in the maximum clock speed.