Thoughts on Exam 1

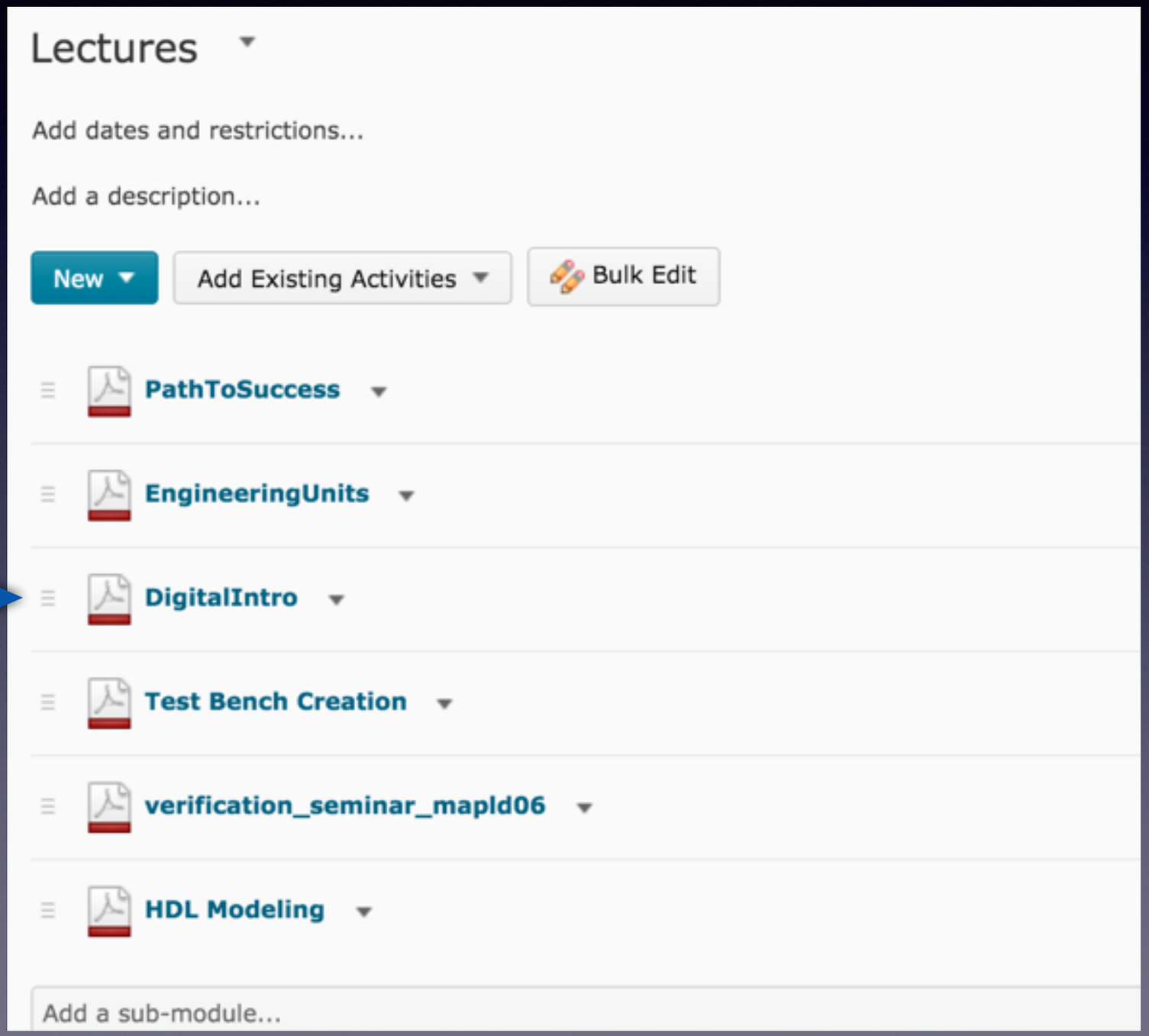
CECS 460
Spring 2016

Purpose of Presentation

- I created this for my 360 students after grading their first exam
- Since the story is so similar at this point between the two classes I am giving you the same review along with 460 specific information at the end
- You would do well to make sure that all the information contained herein has been added to your knowledge bank

Have Your Read The Course Supplied Material?

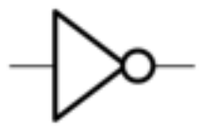
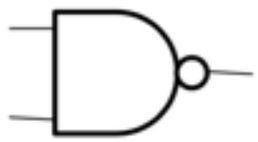
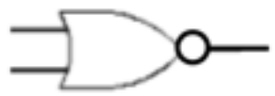
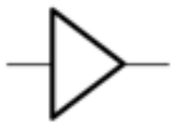
Supplied
For Your Review
In Case You Need Help



201 Gate Concepts

At this time you should know all of the gates and how to draw them and what is the Verilog operator is for each operation

Basic Logic Gates

Name	Symbol	Verilog
and		y = a & b;
or		y = a | b;
not		y = ~b;
exclusive or		y = a ^ b;
nand		y = ~(a & b);
nor		y = ~(a | b);
buf		y = a;

a	b	and	or	not	xor	nand	nor	buf
0	0	0	0	1	0	1	1	0
0	1	0	1	0	1	1	0	0
1	0	0	1	1	1	1	0	1
1	1	1	1	0	0	0	0	1

Gates define relationships. How does the output relate to the inputs? Gates and the expressions that are built by combining them together are referred to as **COMBINATIONAL LOGIC**. When the inputs switch the expression is evaluated and the output is updated.

Soldiers, forty centuries are looking down upon you!

Napoleon - when entering Egypt

Students, forty years are looking down upon you!

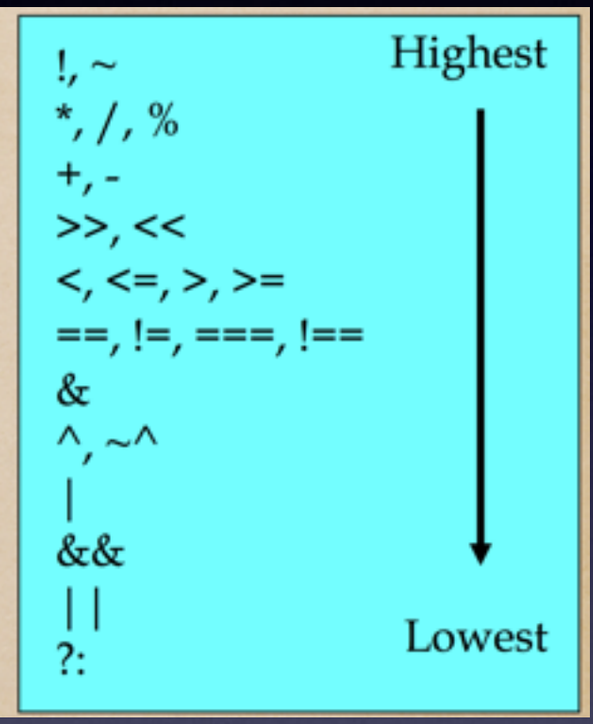
Tramel - when starting CECS 440

Bit Wise vs. Logical Operators (Not the same)

Bit-wise vs Logical Operations

- Application of Boolean Algebra has resulted in two types of operations
  - Bit-wise: the operation is performed on operands of one or more bits with the result being equal to the width of the operands
  - Logical: the operation is performed on operands with the value of 1/0, TRUE/FALSE and the result is always 1/0, TRUE/FALSE

{}	Concatenation
+, -, *, /	Arithmetic
%	Modulus
~	bit-wise Negation
&	bit-wise/reduction AND
|	bit-wise/reduction OR
^	bit-wise/reduction XOR
^~, ~^	bit-wise/reduction XNOR
<<	Left shift
>>	Right shift
?:	Conditional
>, >=, <, <=	Relational
!	Logical Negation
&&	Logical AND
||	Logical OR
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality



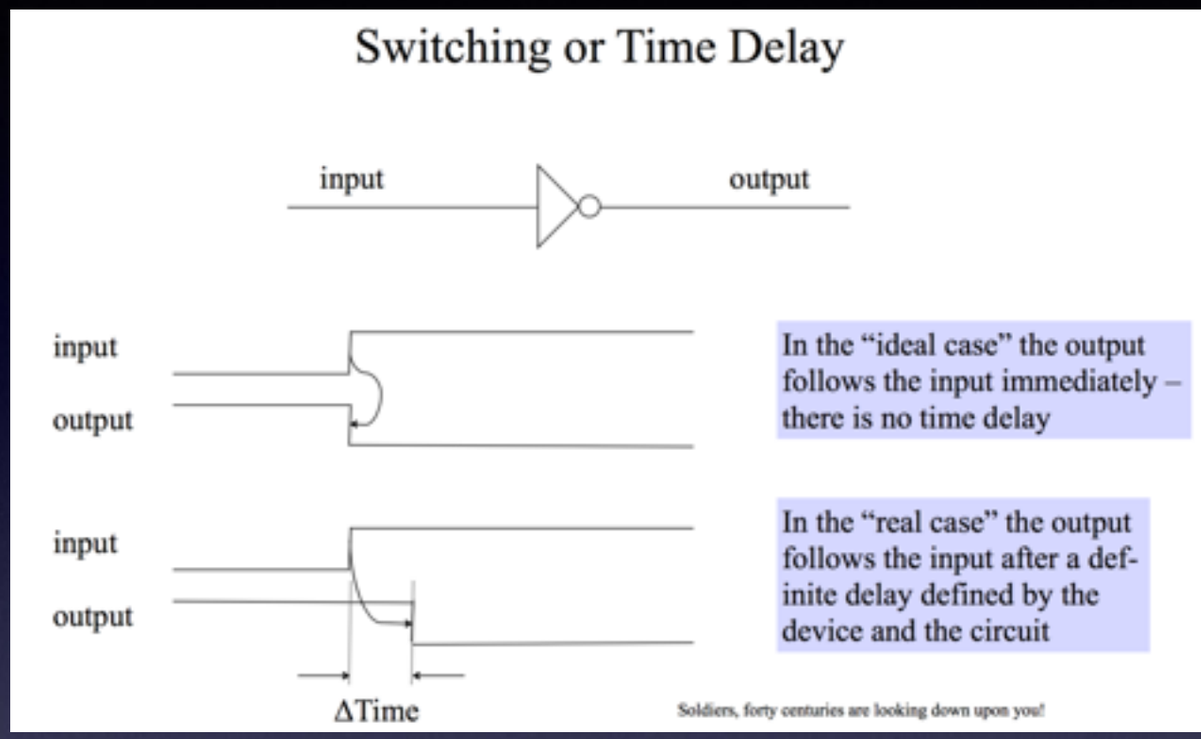
When manipulating bits you should use the “bit-wise” operators: ‘|’, ‘&’. When dealing with logical equations you should use “logical”operators.

```
reg [15:0] A, B;
y = A | B;  // good or
y = A & B;   //good and

y = A || B;  // bad or
y = A && B; // bad and
```

Switching Characteristics - Gates

Combinational Circuit Issues

- In the *ideal circuit* the output transition follows the input transition immediately (no delay)
- Physical devices exhibit delays when switching
  - t_{phl}: time to propagate from high to low
  - t_{plh}: time to propagate from low to high
- Classroom timing diagrams follow the switching characteristics of the *ideal circuit* with switching delays assumed.



We discussed in class the fact that even light is limited in the distance it can travel in time. (~1 ft/ns). Same is true with signals in our circuits. Combinational logic should be the easiest to understand but it sets the stage for sequential circuits (flops).

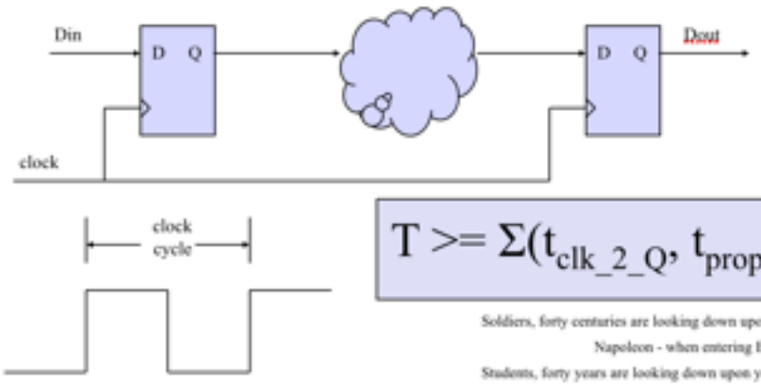
Switching Characteristics - Flops

Sequential Circuit Issues

- In the *ideal circuit* the output follows the D input immediately following the active clock edge (no delay)
- Physical devices exhibit delays when switching
  - t_{ck2Q}: time from clock edge to Q update
- Physical devices have requirements when switching
  - Setup: Time the D input must be stable prior to the clock edge
  - Hold: Time the D input must be stable after the clock edge
- Physical delays are assumed in our circuit analysis (although our diagramming techniques don't show them) - they allow synchronous circuits to behave orderly

Clocking Methodology

- We assume all sequential devices change states on the same active edge of the clock (rising or falling)
- We assume the clock cycle is long enough to allow the propagation delay through the combinational logic
- The clock cycle (T, or period) sets the time budget: how much time is available to accomplish the desired function?



Soldiers, forty centuries are looking down upon you!
Napoleon - when entering Egypt
Students, forty years are looking down upon you!
Tramel - when starting CECS 440

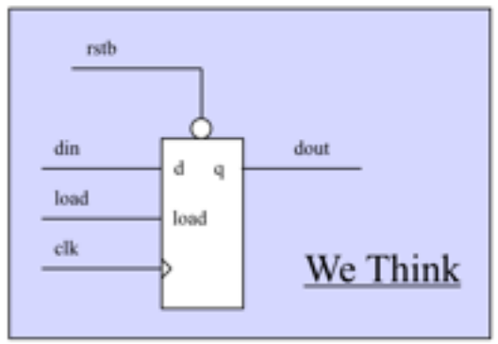
We discussed in class the fact that even light is limited in the distance it can travel in time. (~1 ft/ns). Same is true with signals in our circuits. Combinational logic should be the easiest to understand but it sets the stage for sequential circuits (flops).

Thinking Sequential Design

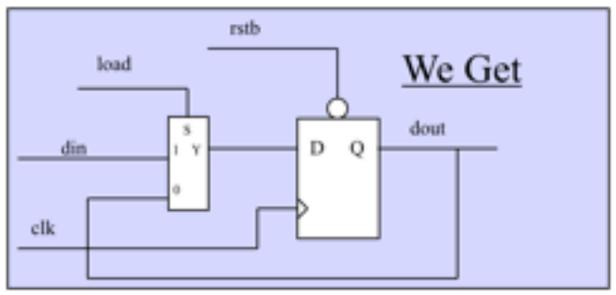
Register Transfer Logic (RTL)

```
module register (clk, rstb, load, din, dout);
input clk, rstb, load;
input [n-1:0] din;
output [n-1:0] dout;

reg [n-1:0] dout;

always @(posedge clk or negedge rstb)
   if (!rstb)
         dout <= 1'b0;
   else if (load)
         dout <= din;
endmodule
```

We Write



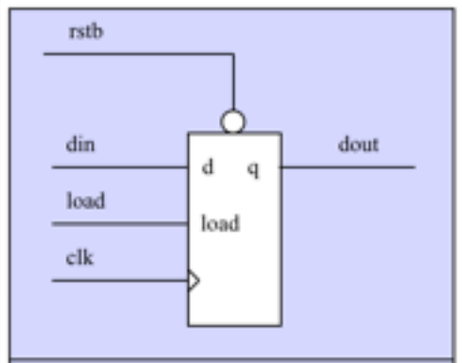
We Think



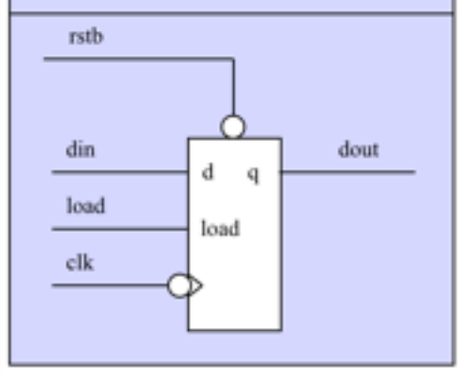
We Get

Soldiers, forty centuries are looking down upon you!
Napoleon - when entering Egypt
Students, forty years are looking down upon you!
Tramel - when starting CECS 440

Graphical Conventions - Sequential



Asynchronous/Synchronous RESET (must be defined)
Synchronous Load
Positive edge triggered clock input



Asynchronous/Synchronous RESET (must be defined)
Synchronous Load
Negative edge triggered clock input

Soldiers, forty centuries are looking down upon you!
Napoleon - when entering Egypt
Students, forty years are looking down upon you!
Tramel - when starting CECS 440

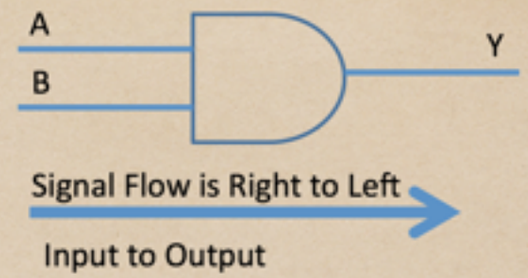
The older slides have low active rest. Everyone should be able to envision sequential design in terms of thinking, writing and getting. The slide on the right emphasizes how you should be able at this time to read the graphical symbols

Combinational Delays

Combinational Delays

- Signals move through electronic devices at speeds that rival the speed of light
- Light travels at 186,000 miles per second
  - At that rate it will circumnavigate the globe over 7 times in one second
  - When broken down into nanoseconds that translates to about one foot per nanosecond
- Every device (gate) will exhibit a finite delay
- More complex gate combinations exhibit a greater delay proportional to the levels of logic

Delay of a Gate

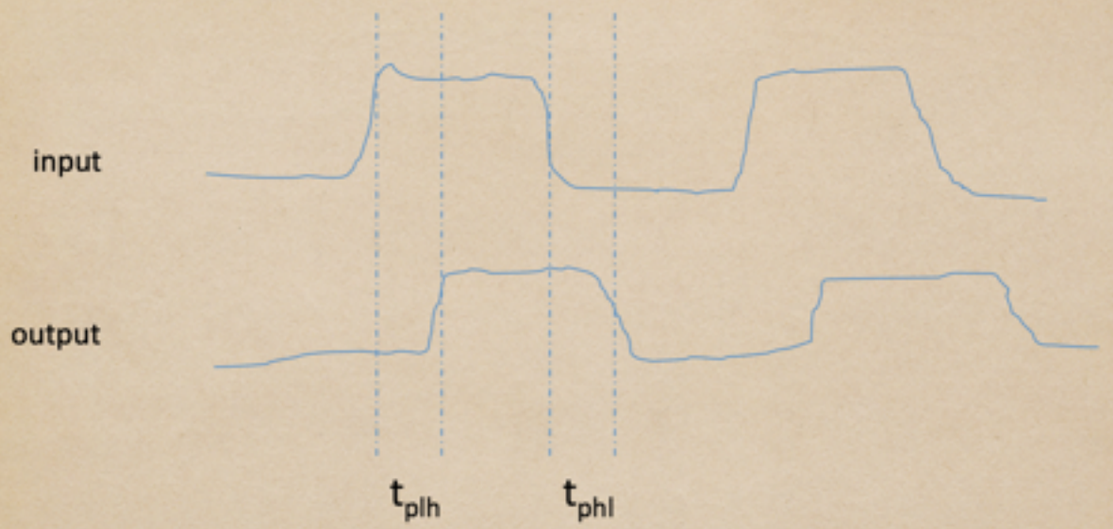


- Delays are identified by the final state of the output
  - If the output ends up a 1 the delay is t_{plh}
  - If the output ends up a 0 the delay is t_{phl}
- When the input switches there will be a delay until the output switches – that is called the

Make sure you understand what these slides present.

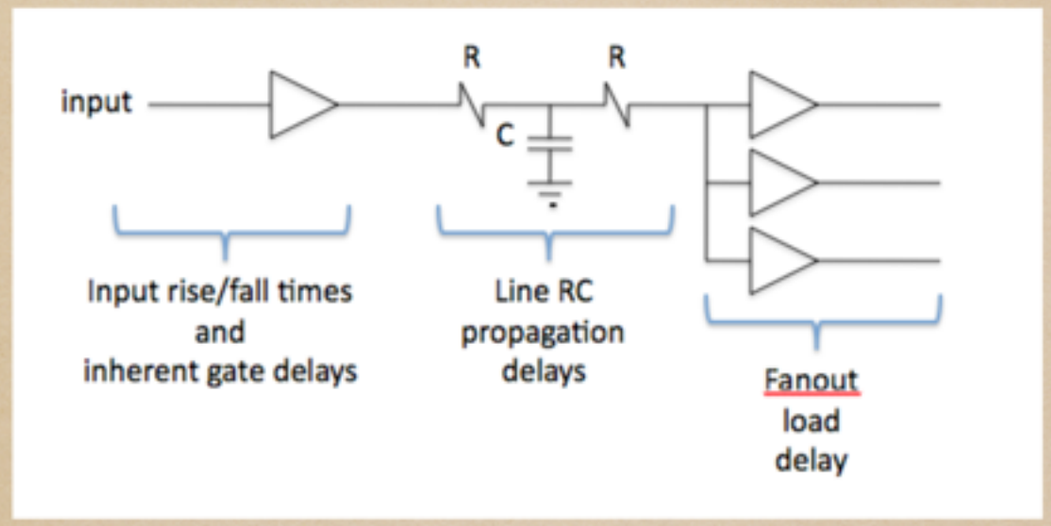
Combinational Delays 2

Switching Outputs



- Typically it takes longer to charge a net than to discharge it. This means $t_{plh} > t_{phl}$

Real World Considerations



- The speed the input rises or falls affects the response time of the gate
- The metal interconnects will have an RC delay
- The number of loads will also affect the time required for the signal to switch

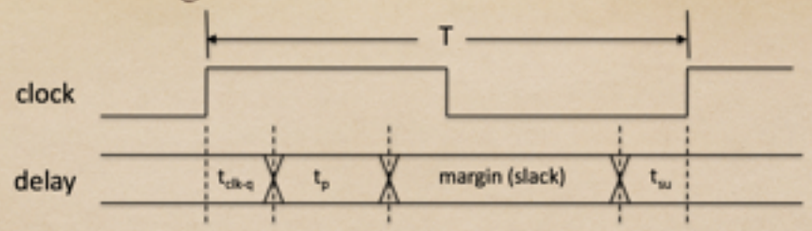
Make sure you understand what these slides present.

Further Thoughts on Timing

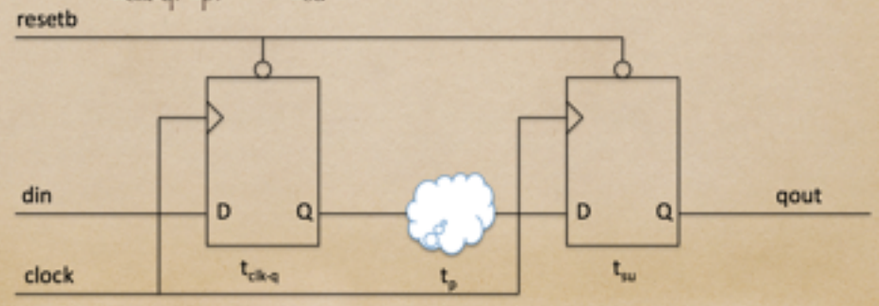
Timing - Real or Imagined

- When considering timing there are two distinct worlds to keep clear in your mind: the simulated world and the physical world
- When modeling with RTL (@(posedge clk)) there are no references to time other than waiting for the active edge of the clock
- Simulation is used to prove functional (logical) correctness, not to prove proper timing
- Confirmation of proper timing is the responsibility of the timing analysis tools - although even still all delays must be modeled (or approximated)
- It is not until the circuit is run at speed, voltage and temperature variations that absolute confidence may be established

Synchronous Considerations



- The timing budget is established by the period of the clock
- Most managers insist on a margin (5-10%) to be used in performing timing analysis - this means that if your period was 30ns you would be allocated 27ns (-10%) for your budget
- Consumers are t_{clk-q}, t_p, and t_{su}



Without the physical delays in our electronics we would never be able to build synchronous digital circuits. Simulation does not require you to understand this - but when you want circuits working on an FPGA you do.

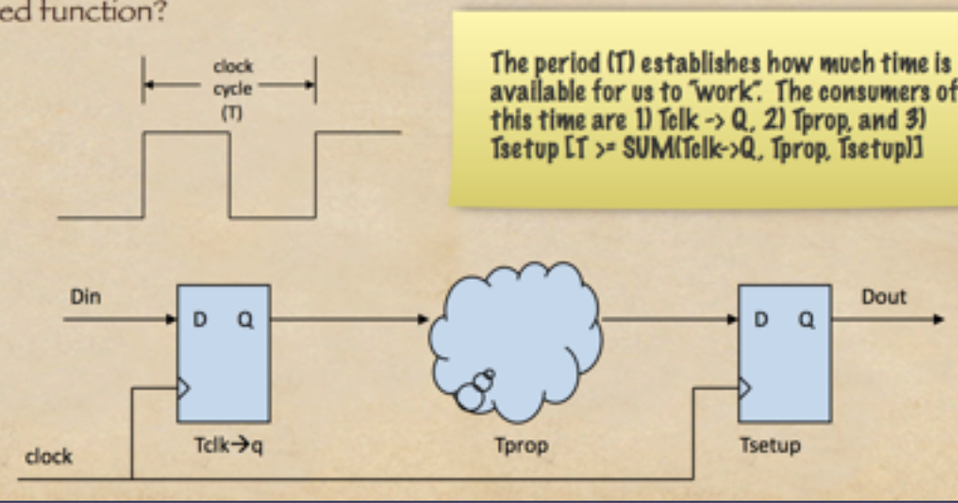
More Further Thoughts on Timing

Issues and Concerns

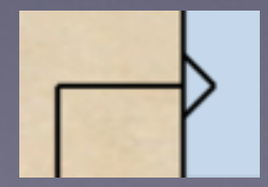
- The fundamental concept of setup and hold are fairly simple to comprehend
- Setup is the time that the data input to the flop must be stable prior to the active edge of the clock
- Hold is the time that the data input to the flop must be stable after the active edge of the clock
- The implications and applications of these concepts appears to be the source of the confusion

Clocking Methodology

- We design so that all sequential devices change states on the same active edge of the clock (rising or falling) ~ ergo "Synchronous Design"
- We design so that the clock cycle is long enough to allow the propagation delay through the combinational logic ~ verify with Static Timing Analysis (STA)
- The clock cycle set the time budget: how much time is available to accomplish the desired function?



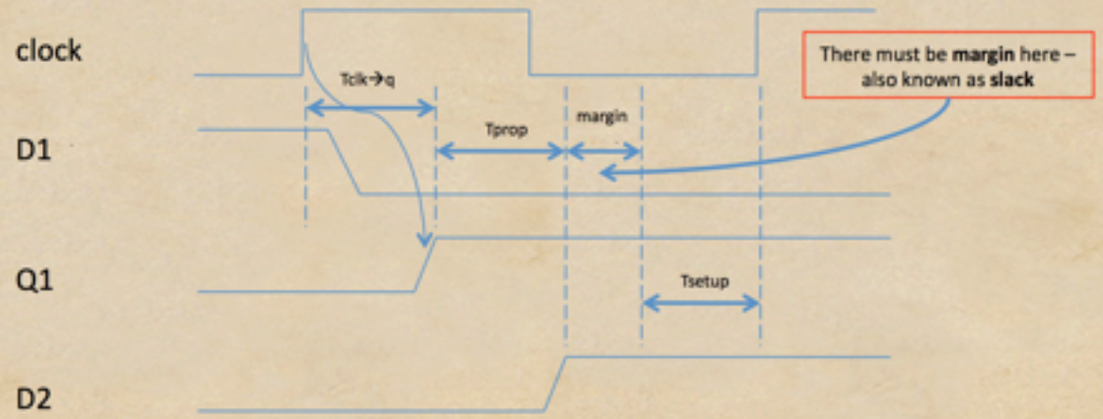
Make sure you know the active edge of the clock. For us it will always be "posedge". The symbol on the drawing is meant to designate that.



More Further Thoughts on Timing 2

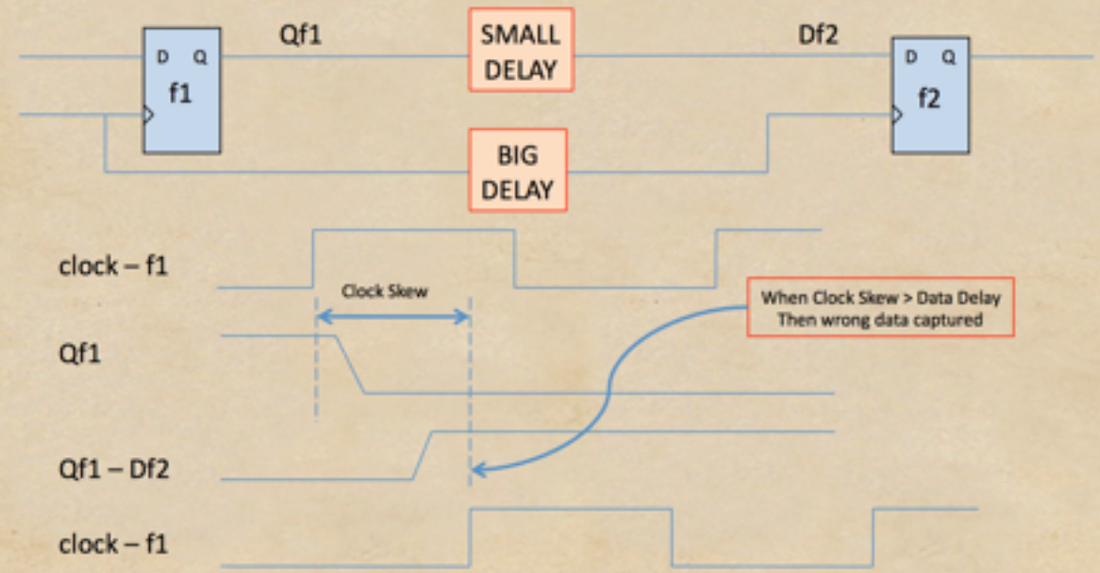
Setup

- The sum of the three consumers must be less than or equal to the budget of the period
- There are other contributors to delay on the nets but we will use this simplified view for our consideration
- The same is true for clock skew
- Setup is a concern when it comes to design construction and effects the worst-case analysis



Hold

- Hold is a concern when it comes to design implementation
- It is an issue in chip design that is exposed by excessive clock skew



We are concerned with setup delays when we are performing “worst case analysis” - will our circuit work? Hold is not used there, it comes into play with clock skew.

More Further Thoughts on Timing 3

Hold Solution

- The practical solution in ASIC/SOC design is to ensure that the clock trees distribute the clocks with minimal skew
- FPGAs typically have "routed clocks" (low-skew clock routing resources) where the manufacturer will guarantee a maximum clock skew
- There are times when delays are added to the data paths to ensure that they will exceed the clock skew (not recommended but sometimes necessary)
- Design practices/methodologies avoid these issues by preventing the use of clock dividers that result in clocks being distributed on a non-low skew routing resource
- This is a very rudimentary presentation - the well is deep
- Remember
  - Clocks go to clocks
  - Resets go to resets
  - Data goes to D inputs

More 201 Slides: Introduction to Sequential Circuits (Flops)

The Big Transition

From: Combinational To: Sequential

- Up till this time the logic we have discussed was implemented functions as in the equation $y = f(x)$. Whenever an input switched, the function was re-evaluated, and the output was updated as necessary.
- Now we will consider circuits that are able to retain their value even though the inputs have switched.
- This retained value is referred to as a "state"

State

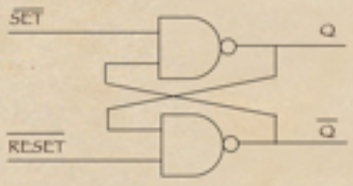
- In the world of computer science and engineering the word "State" has a particular meaning
- State refers to a unique configuration of information in a program or machine
- In order to have state you must have some form of "storage devices" - Something that can retain a value
- In computer science these "devices" are known as variables. In computer engineering these "devices" are known as registers, or flip-flops, or flops
- An initial step toward a flop is what we call a "latch"

First concept is that of "state" - digital electronics is able to 'hold' a value (no more just y=f(x))

More 201 Slides: Latches (unfavored but sometimes necessary)

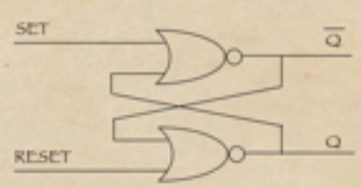
Latches

- A simple technique to build a latch is to use the gates we have learned about and interconnect them in what may be considered an unusual manner: feedback
- We will only use NAND and NOR gates for this



$\overline{S}$ $\overline{R}$	Q	$\overline{Q}$
00	X	X
01	1	0
10	0	1
11	-	-

This means no change - the output keeps the value it had before the inputs switched to this combination

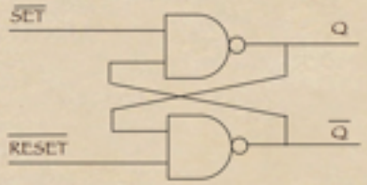


SR	Q	$\overline{Q}$
00	-	-
01	0	1
10	1	0
11	X	X

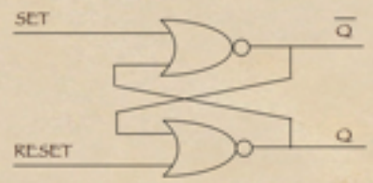
XX means do not let the inputs be found in this configuration. This is an illegal condition meaning the latch will not work properly

Latch Behavior

- Let us study the truth tables to understand the behavior of the latch



$\overline{S}$ $\overline{R}$	Q	$\overline{Q}$
00	X	X
01	1	0
10	0	1
11	-	-



SR	Q	$\overline{Q}$
00	-	-
01	0	1
10	1	0
11	X	X

* The active state of the input is indicated by heading of the LHS - bar means active LOW, no bar means active HIGH
* In no case should both inputs be active at the same time (X = don't do this)
* When SET is active, Q goes to 1 and $\overline{Q}$ goes to 0
* When RESET is active, Q goes to 0 and $\overline{Q}$ goes to 1
* When neither input is active the outputs remain the same - retain their "state" (ie. LATCHED)

Latches lay the foundation for understanding flip-flops.

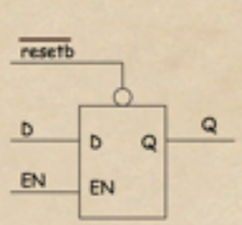
More 201 Slides: Latches (unfavored but sometimes necessary)

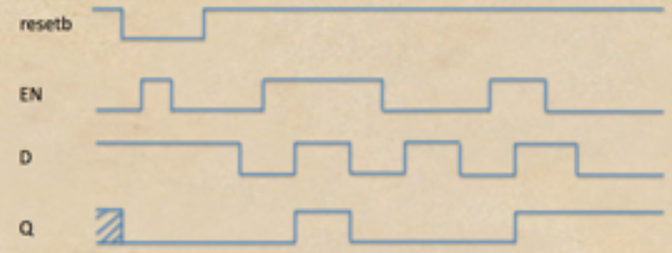
D Latches

- In the textbook and in device libraries there are elements called "D latches" (that look like flops – but they are not!)
  - A latch has the concept of "state" – it will retain its value
  - A latch has an "enable" that controls when the state of the latch may be changed (0->1 or 1->0)
  - A latch may also have an asynchronous reset that will force the output of the latch to its inactive state
- The behavior of a D latch is defined as: "when the latch is enabled the output follows the input, when the latch is disabled the latch will retain the value on D at the time it was disabled."

This is a good sentence to memorize!

D Latches Continued





- On the left is the block symbol for a latch that behaves exactly the same as we have seen with the exception that we now have a data input (D) and an enable (EN)
- Notice that when the latch is enabled (EN=1) that the output follows the data input and when the latch is disabled (EN=0) then the latch retains its value
- Notice that before we assert reset or enable that we do not know the value of Q so we mark it with hashed lines

Documenting the behavior of digital circuits using timing diagrams should be something you are beginning to feel comfortable doing. Of course, it assumes you understand how the digital circuits behave.

More 201 Slides: Latches (unfavored but sometimes necessary)

Modeling D Latches in Verilog

```
module latch (D, Q, EN, RSTb);
input  D, EN, RSTb;
output Q;

reg Q;

// always @(*)

always @(D or EN or RSTb)
   if (!RSTb)
      Q = 1'b0; else
   if (EN)
      Q = D;    //notice no "else"

endmodule
```

Remember that you can 'infer unwanted latches' if you do this in your combinational logic design. Remember to define what you want to happen on every conditional branch (if/else, case, etc.)

More 201 Slides: Sequential Logic

The Big Turn

- When the subject of Digital Electronics was introduced at the beginning of the semester we discussed how the material would be divided into two sections: 1) Combinational Logic 2) Sequential Logic
- Up till now we have dealt with gates and considered how they may be combined to build a variety of fundamental building blocks
- Now we move on to Sequential logic to learn about clocks, flip-flops (flops), registers, counters, and many more mysterious constructs

Sequential Circuits

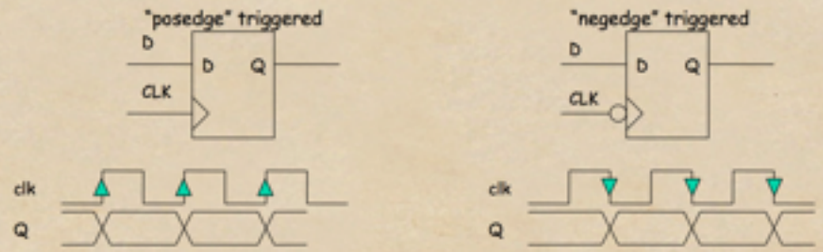
- Sequential circuits introduce two new items
  - The concept of a "state"
  - The concept of a "clock"
- The basic building block for sequential circuits is the flip-flop. Although there are different variations: D, JK, T we will concentrate on the D flip-flop, or flop
- "State" implies that the circuit will retain a value
  - This is different from combinational logic where we are always defining expressions, ie. $y = f(x)$. Here whenever the input switches the function is applied and the output is updated.
  - The "state" of a design will be defined as the contents of the D flops at any given point in time
- "Clock" implies there will be a system clock and that the flops will only be updated on the active edge of the clock

To pass 360 these concepts must be clear to you.

More 201 Slides: Sequential Logic

Sequential Circuit Behavior

- Sequential circuits behave differently than combinational circuits in that the outputs only switch at the active edge of the clock



- At the "active" edge of the clock (either posedge or negedge) the value on D is copied to Q
- Between active edges the contents (therefore the output) of the D-flop remain constant

This is a good sentence to memorize

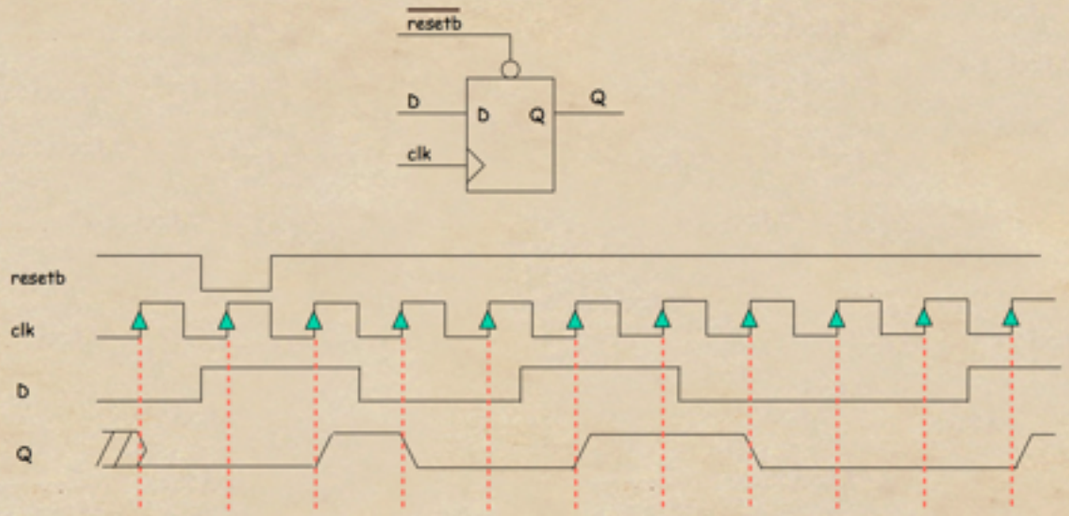
Asynchronous Reset

- For synchronous circuits to work predictably there is the need to be able to bring the circuit to a "known" state. This is accomplished through a "reset" signal
  - Resets will either be HIGH or LOW active
  - For our class we will define our reset input as "asynchronous"
    - This means that when reset is asserted the flop is immediately cleared
    - To synchronously clear the flop you must utilize the D input
  - As long as reset is held active the flop will be initialized. It is only after reset is removed that the flop will respond to the clock

Although flops may be clocked on posedge and negedge we will always use the former. This means the outputs of flops ONLY CHANGE AFTER AN ACTIVE CLOCK EDGE! All our flops reset asynchronously (although we drive the resets with a synchronous signal (AISO).

More 201 Slides: Sequential Logic

Simple Example of a D Flop



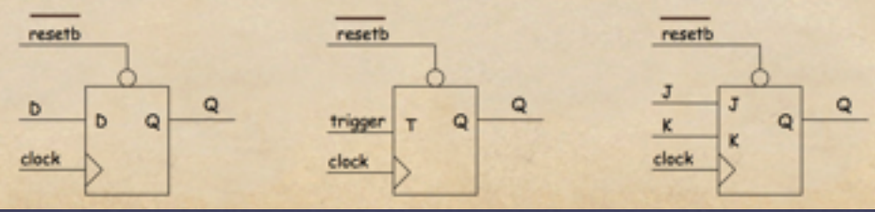
- Reset always takes precedence over the clk and D inputs
- At the ACTIVE edge of the clock, the value on D goes to Q
- Whenever reset is ACTIVE the Q goes to zero

General Flip-Flops (Flops)

- As mentioned we will deal only with D flip flops
  - At the active edge of the clock, the value on D is copied to Q
- Another flip flop type is the T
  - Every active edge of the trigger flips Q (0,1,0,1 …)
- Another flip flop type is the JK

J	K	At Active Edge of Clock	
0	0	No Change	
0	1	Clear flop	(Q←0)
1	0	Set flop	(Q←1)
1	1	Flip the Flop	(Q←~Q)

Although we speak about 3 types of flops: D, T & JK we are primarily interested in the D

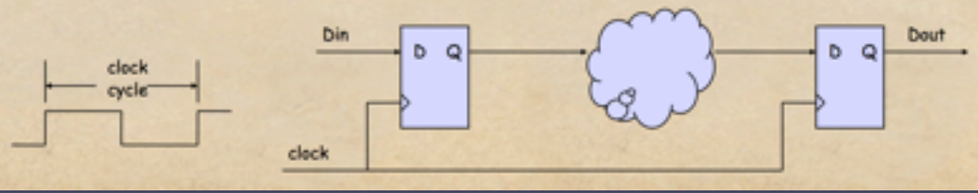


You must understand how a D Flop behaves. We will use it as a building block to create many other more complex functions that designs require.

More 201 Slides: Sequential Logic

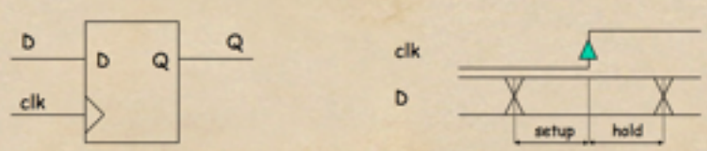
Clocking Methodology

- We assume all sequential devices change states on the same active edge of the clock (In system design there will be multiple clock frequencies but that will be dealt with differently)
- We assume the clock cycle is long enough to allow the propagation delay through the combinational logic cloud (gates)
- The clock cycle sets the time budget: how much time is available to accomplish the desired function?
  - Examples: 10 MHz => 100 nsec; 25 MHz => 40 nsec
- Delays encountered
  - Clock to Q on flop; propagation delay through cloud; setup time to flop



Synchronous Design Requirements

- Flip flops impose timing requirements on the designer in order to produce a circuit that will work properly
- These requirements impose timing relationships between the D input and the active edge of the clock
  - Setup: this is the period of time that the D input must be stable (no switching) prior to the active edge of the clock
  - Hold: this is the period of time that the D input must be stable (no switching) after the active edge of the clock
- If you violate these timing relationships then your circuit will not work predictably.



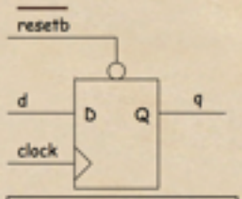
Sorry for repeating myself - but these concepts must be clear to you.

More 201 Slides: Sequential Logic Modeling in Verilog

Modeling Sequential Logic using Verilog

- There is a particular style that should be followed when modeling flops in Verilog
- The following example is a positive edge triggered, asynchronously reset D flop

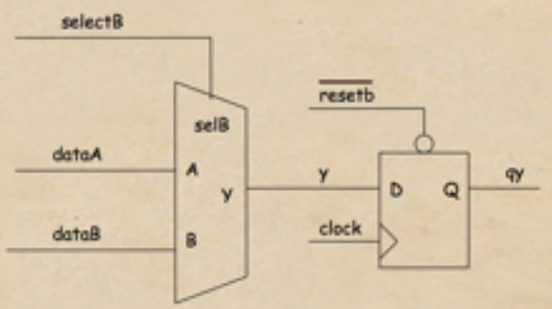
```
module dflop (clock, resetb, d, q);
input clock;  // positive edge clock
input resetb; // low active, asynchronous reset
input d;      // data input
output q;     // flop output
reg q;        // procedural block - register
data type

always @(posedge clock or negedge resetb)
   if (resetb == 1'b0)
      q <= 1'b0;
   else
      q <= d;
endmodule
```



This is the symbol of what we modeled in Verilog

Another Example - Registered Multiplexer

```
module regmux (clock, resetb, selB, dataA,
dataB, qy);
input clock, resetb, selB, dataA, dataB;
output qy;      // flop output
reg qy;         // procedural block - reg data
type
always @(posedge clock or negedge resetb)
   if (resetb == 1'b0)
      qy <= 1'b0;
   else
      if (selectB == 1'b1) qy <= dataB;
      else                 qy <= dataA;
endmodule

--- OR ---

wire dy;
assign dy = selectB ? dataB : dataA;
always @(posedge clock or negedge resetb)
   if (resetb == 1'b0)
      qy <= 1'b0;
   else
      qy <= dy;
```



I know we are now using HIGH ACTIVE RESET but these slides are old. Please adapt. It is the same methodology we are practicing. Writing correct Verilog is a skill you need to be developing.

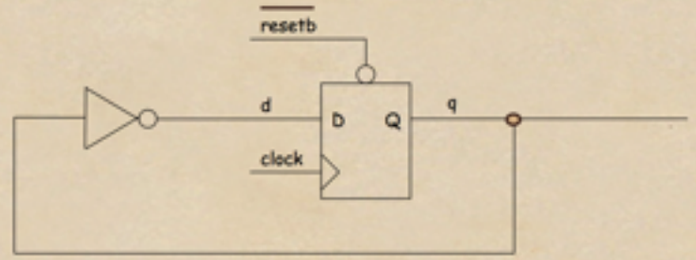
More 201 Slides: The D Flop

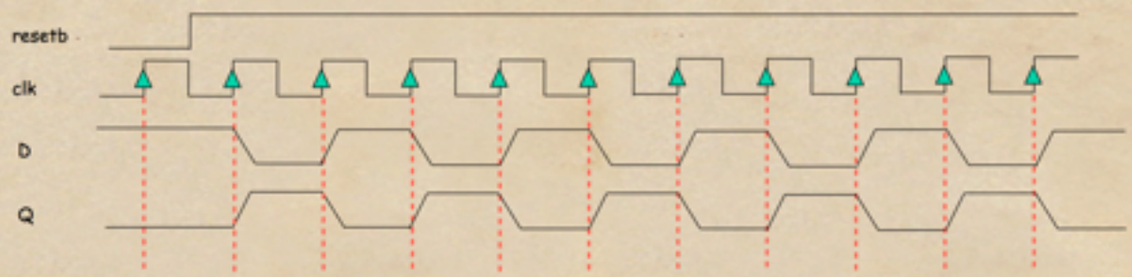
The D-Flop - Our Foundation

- We have been introduced to the D-Flop as the "work horse" for implementing sequential (clocked) circuits
- The next two slides will show how we can use the D-Flop to implement both the T-Flop and the JK-Flop

The T-Flop

- This is rather straight-forward: the definition of a T-Flop (Toggle Flop) is that Q should "toggle" (0->1, 1->0) on every active edge of the clock





We use the D flop to create other types of flops. It is how the flops are controlled that varies in the different types.

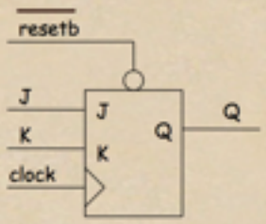
More 201 Slides: The D Flop

The JK-Flop

- This is little more complex in that we have two inputs that will control whether we set, reset or toggle the flop - create and implement the truth table

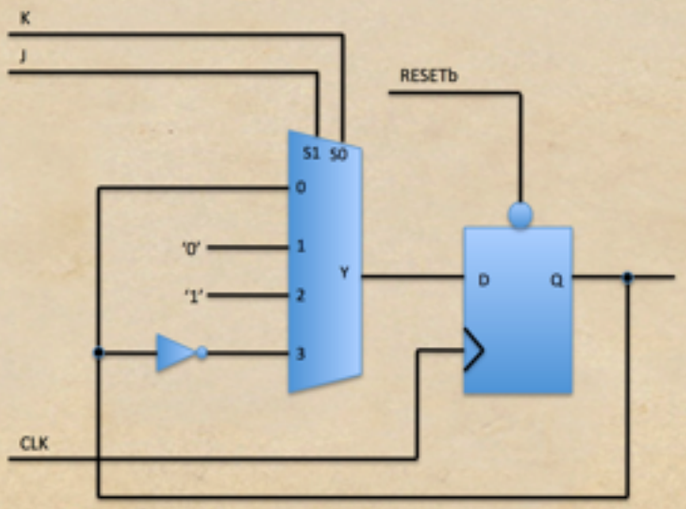
JK Flop Truth Table

J	K	Q	
0	0	Q_{t-1}	No Change
0	1	0	Clear
1	0	1	Set
1	1	$\overline{Q}_{t-1}$	Toggle (Flip)



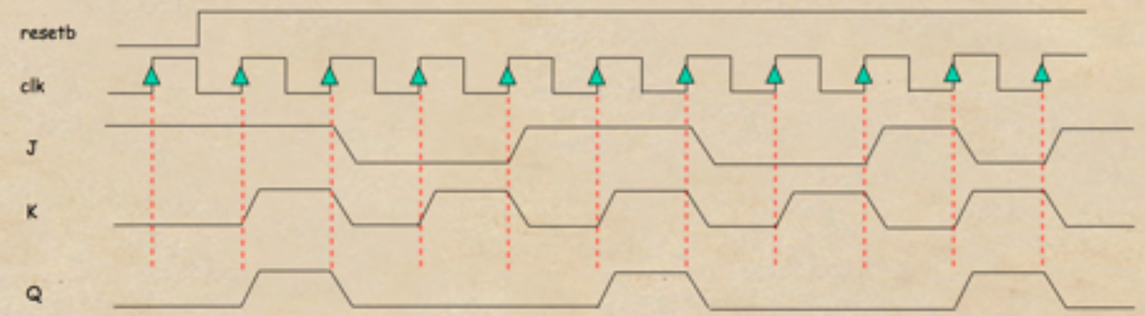
JK Flop Truth Table (D Flop)

J	K	D
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$



The JK-Flop

- This demonstrates the behavior of the JK Flop
- Follow along with the truth table

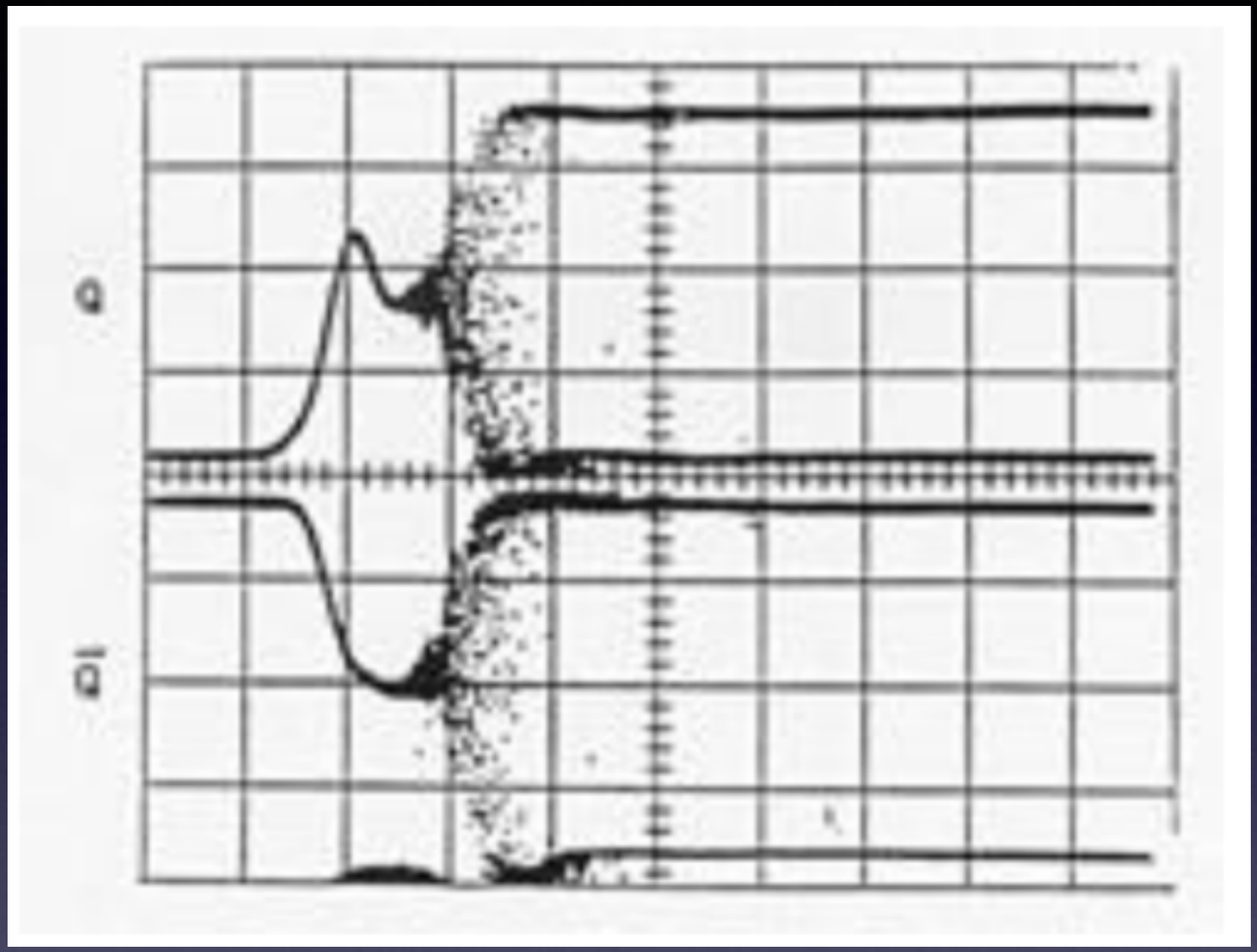


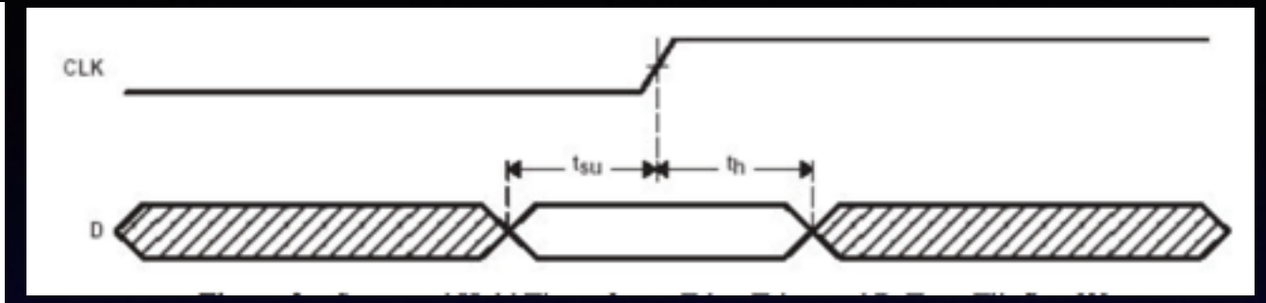
JK Flop Truth Table

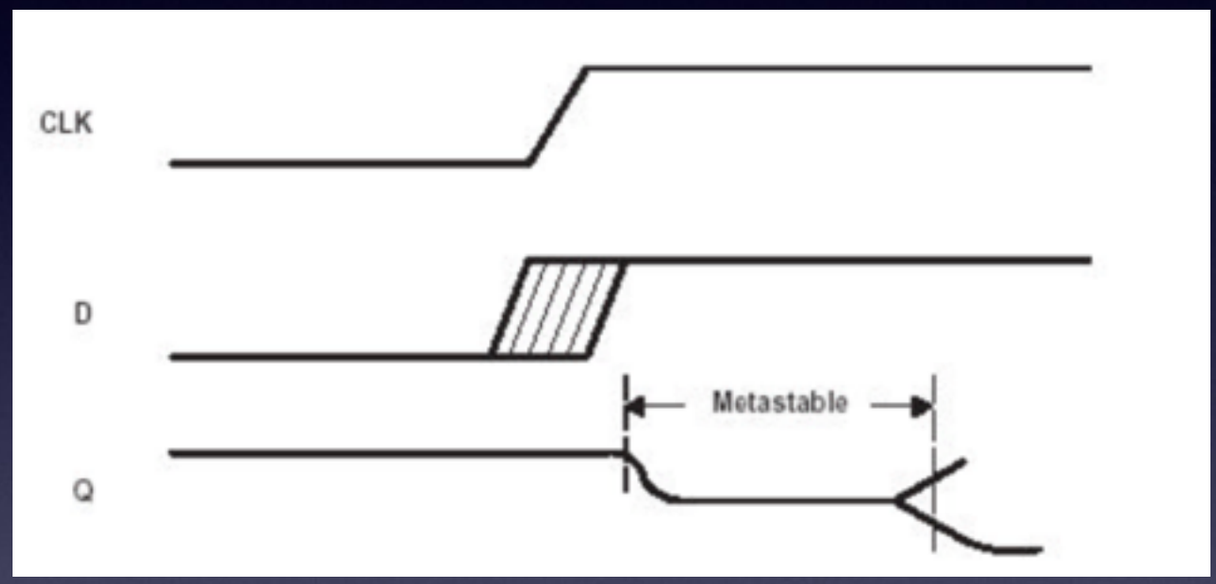
J	K	Q	
0	0	Q_{t-1}	No Change
0	1	0	Clear
1	0	1	Set
1	1	$\overline{Q}_{t-1}$	Toggle (Flip)

We should all be able to generate these timing diagrams.

Metastability - Paper on BeachBoard





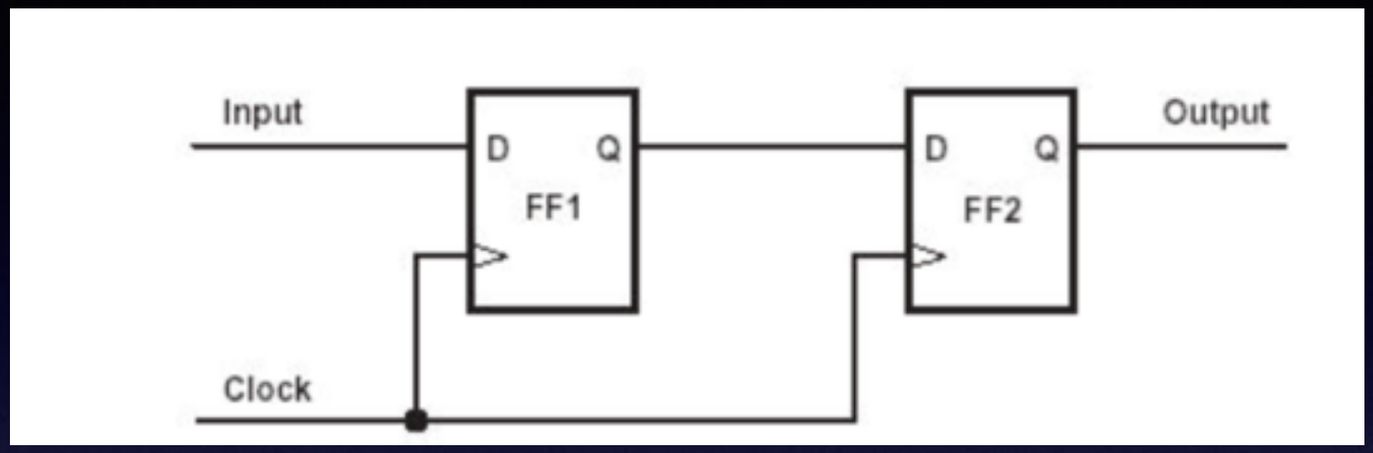


What is metastability?

What causes metastability?

Metastability is when the output of a flop “oscillates” and settles in an unpredictable state. This settling should occur within one clock period. Each flop has timing constraints - violating those constraints it the reason the flop enters the metastable state.

Dealing with Metastability



Conclusion

Digital circuit designers must determine the metastability characteristics of their circuit in order to ensure reliability. The degree to which metastability occurs within a circuit can to a great degree determine the reliability of a circuit. Designers must be versed in knowing about metastability, predicting the occurrence of metastability, and limiting the frequency of metastable outputs. Ways of limiting metastability include using only one clock, using faster flipflops, decrease the asynchronous input frequency, and use synchronization hardware. These steps can easy be taken by designers to increase the reliability of a circuit.

We identified two causes for metastability - 1) Bad design - in this case slowing the clock, faster technology, redesign with pipelining should resolve the issue. 2) Crossing clock domains - here is a design necessity. To do this the receiving domain must synchronize the incoming signal using a double flop technique. The caveat here is that the incoming signal must be able to be sampled by the receiving clock.

Metastability Lectures

Cross Clock Domains

If everyone in the world had the same internal clock then perhaps there wouldn't be so much misunderstanding among them.

Synchronous Design

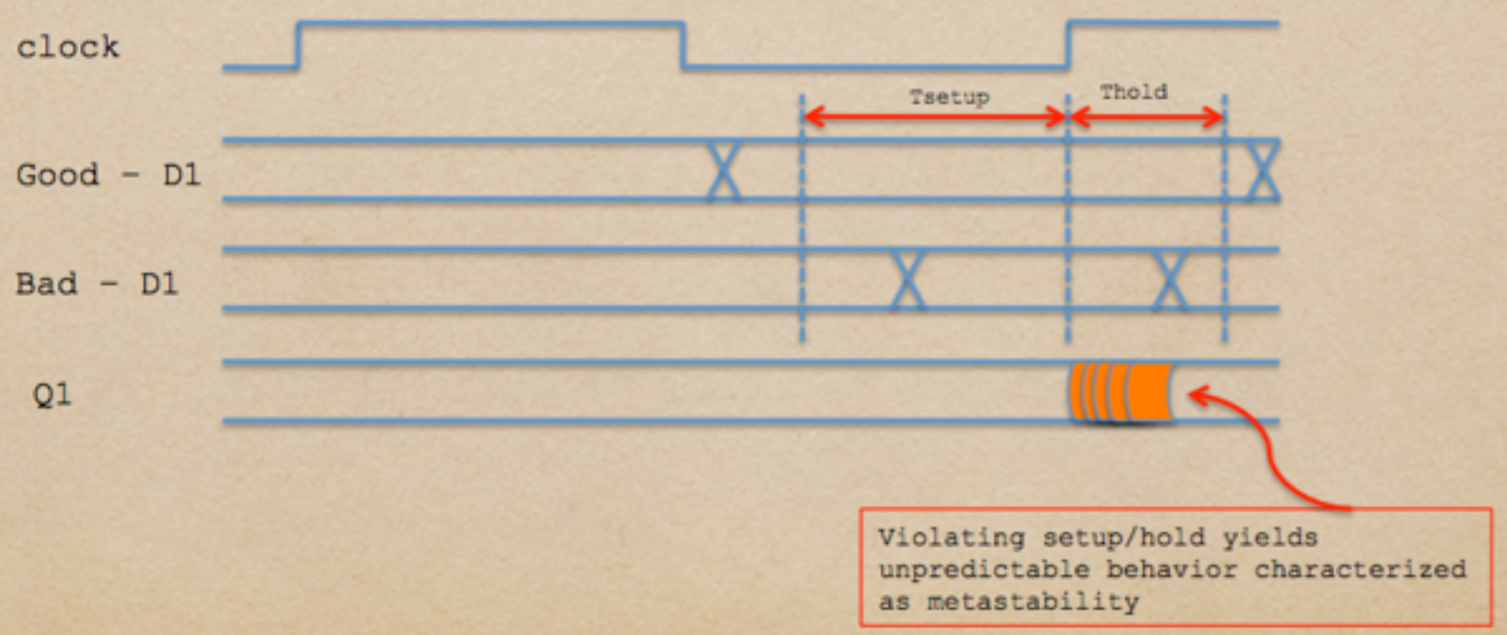
- In the methodology taught in class there is always one clock and every flop in the design receives the same clock. This prevents the issues discussed in the following slides.
- In practicality there is the need to provide multiple clock sources in order to meet the standards required by external interfaces
- There are guidelines to follow that allow the interfacing of signals sourced in one clock domain and need to be received in another.
- Each clock domain still has the requirement of one clock source to every flop in the block.

- When a signal is sourced from one clock domain and received by another there is no way to ensure that the receiving flop has met its setup and hold requirements
- Violating setup/hold has the potential of placing the flop into a "metastable" state where the output vacillates for a period of time before settling to an unpredictable state

Metastability Lectures

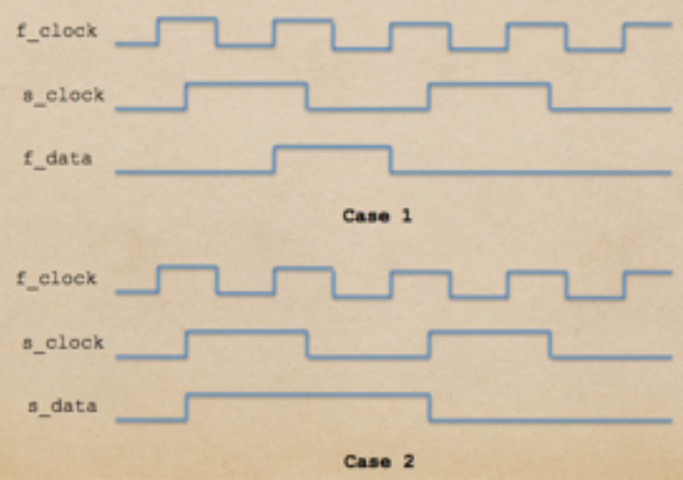
Metastability

- The data input must be stable during the period of time defined as **setup** and that defined as **hold**
- Metastability is when the output may oscillate and settle in at an unpredictable state
- The settling time is defined by manufacturer and is typically assumed to be less than the clock period.

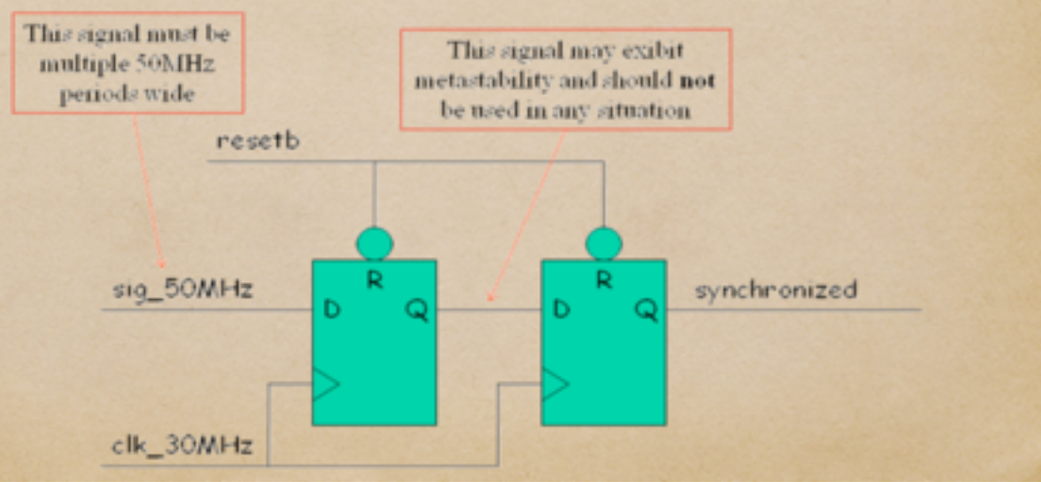


Metastability Lectures

- For single pulse signals that sourced at a faster frequency may not be sampled by the receiving clock – case 1
- Signals from a slower frequency typically may be sampled at the receiving end – case 2
- In both cases the signals must be treated as asynchronous and must be synchronized



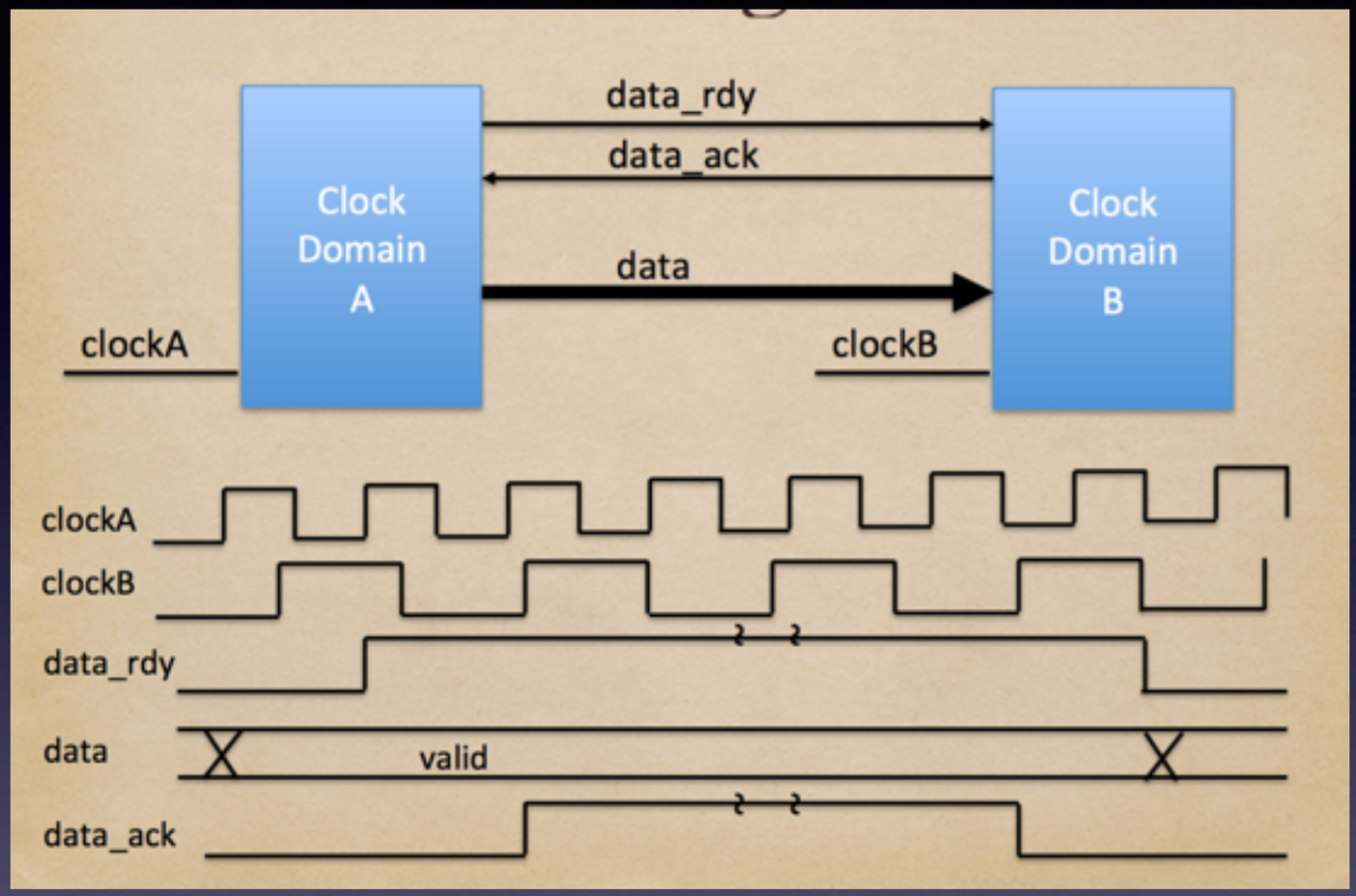
- The simplest solution is to generate a pulse that is longer than the clock period of the receiving domain. This allows synchronization using "double flop"



The signals sourced in one clock domain and expected in another clock domain are considered to be "crossing-clock domains." Typically only the control signals need to be synchronized (not the data). To properly synchronize signals there is the need to be able to meet Nyqvist Sampling where the signal is oversampled at the receiving end.

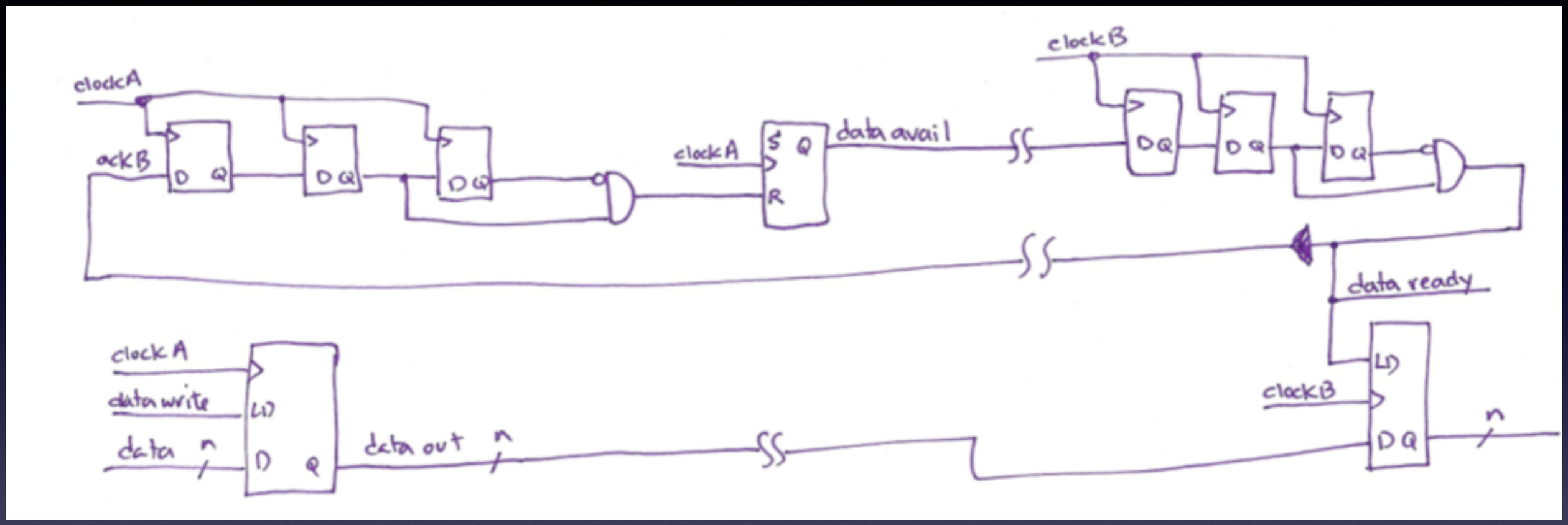
Metastability - Recommended Approach

- Moving data from one clock domain to another needs to be accomplished with much care
- Remember that the control signals are what need to be synchronized, not the data (this may save significant logic)
- A recommended approach, where time allows, is that there be a handshake on each data exchange thus ensuring the proper receipt of the data



The recommended approach is to use “feedback.” This means that the sender holds the signal active until the receiver acknowledges that it has receive it. This approach fits into the “Digital Synchronous Design Methodology” where there is never anything left to chance (or assumptions) - the signal is active till it is confirmed that it has been received.

Metastability - Implementation Sketch



This case demonstrates a case where the sender is sending “n” bits of data (8/16/32) to the receiver. It will load the data into the data_out register and then will set the data_available signal to the receiving domain. The data is held constant until the receiving domain responds with an acknowledge. The acknowledge captures the data on the receiving end and also notifies the sender that the data has been received. It will also clear the data_available signal.

Metastability - What Does it Have to do with Reset?

Reset

[v. ree-set; n. ree-set] verb, -set, -set·ting, noun-verb (used with object)
1. to set again: to reset an alarm clock. 2. to set back the odometer on (an auto or other vehicle) to a lower reading: a used-car dealer charged with resetting his cars.

- For anything to work properly it needs to get off to a good start
- For this purpose we introduce the concept of the reset signal
- I believe that its application to digital circuits may be traced to the bowling machines that would allow the player to set up the pins again or to "reset" the pins.
- The definition in our world of reset it to bring the state of our machine to a KNOWN STATE (typically zeros)

Basic Concepts

- Digital circuits use the reset signal to bring the machine to a "known state" [although technically one might think to "reset" a signal is to set it to a "1" but this is not the case - think "KNOWN STATE"]
- The reset inputs on flops should only be used for initialization – never for control
- Depending on the definition of the flop reset may either put a zero in the flop (reset) or a one (set)
- The reset signal is sourced external to the device and is typically a system-wide signal
- By definition the reset signal is "asynchronous" with respect to the clock within the device
- A typical external implementation is a simple RC circuit with a time constant measured in milliseconds

Reset is an extremely critical signal in the digital design world. In the theme of always being in complete control the designer needs to know that the state of the machine will be 'known' whenever there is reset. Resets occur at power-up and whenever a catastrophic event occurs that demands the system to "start over." When this happens the proper design of reset will ensure the machine always starts up properly.

Metastability - What Does it Have to do with Reset?

Implementations

Systems may not need a reset if the characteristic behavior is to "flush out" the design prior to dependent operation

- An example would be a shift register that would shift in all zeros before the operation begins
- In certain applications this is desirable because the flop without a reset is smaller (less area)

Historically reset was asserted and de-asserted asynchronously to the internal clocks

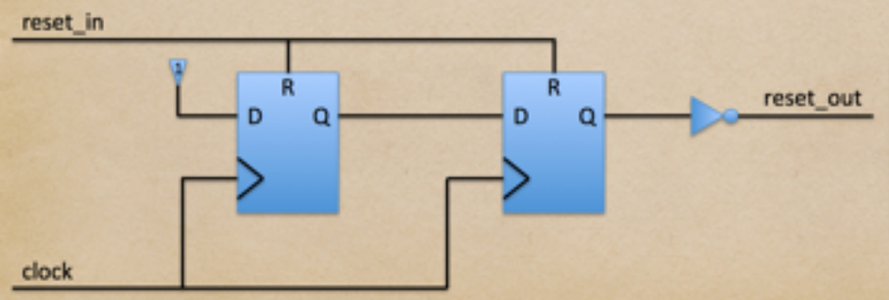
This implementation ignored a requirement defined by most flops that requires the reset signal to be stable at the activation of the clock with adverse results being potential metastability

Recent design methodologies have imposed a requirement for each clock domain within the design called "asynchronous-in, synchronous-out"

- This makes the release path of reset a synchronous event that STA tools can confirm proper operation

Asynchronous-In, Synchronous Out

- This implementation requires a similar circuit for every clock domain
  - Reset signal to each clock domain unique and synchronized to the clock domain
- The assertion of reset is passed on directly to the flops but the de-assertion is "delayed" and synchronized to the clock domain



Author Today, 9:26 AM

If our design also had a 66 MHz clock domain then we would require a similar circuit with the same resetb input but having the 66 MHz clock and deriving sresetb_66MHz

The purpose of the AISO circuit in our design is to prevent any of the flops in our design from entering a metastable state when reset is released. Just as there is a timing constraint between the data input and the active edge of the clock - so also with the release of reset. Synchronously releasing reset ensures all of the flops are reset at the same time and that they are stable (no metastable) when reset is released.

Debouncing Inputs from Mechanical Switches

The Problem

- Interfacing to mechanical devices exposes the dramatic time differential between the movement of mechanical devices and the digital electronics expected to respond to them
- When a switch moves there apparently is a clean 0 - 1 or 1 - 0 transition but in reality there is a transition period where many transitions occur which may be detected by the electronics.

A General Rule and a Proposed Solution

- A general rule is that when a mechanical switch is being moved that there will be approximately a 20 ms period of uncertainty when the value being read from the switch might be transitioning.
- Pong presents two solutions: 1) to move the output as soon as a transition is detected, and 2) to wait until the transition has been stable before detecting the value
- I do not like the first solution because a decision is being made before it has been clearly established that a transition is being made (could be noise)
- My preference is to wait until the output has been stable for 20 ms before determining the value of the switch

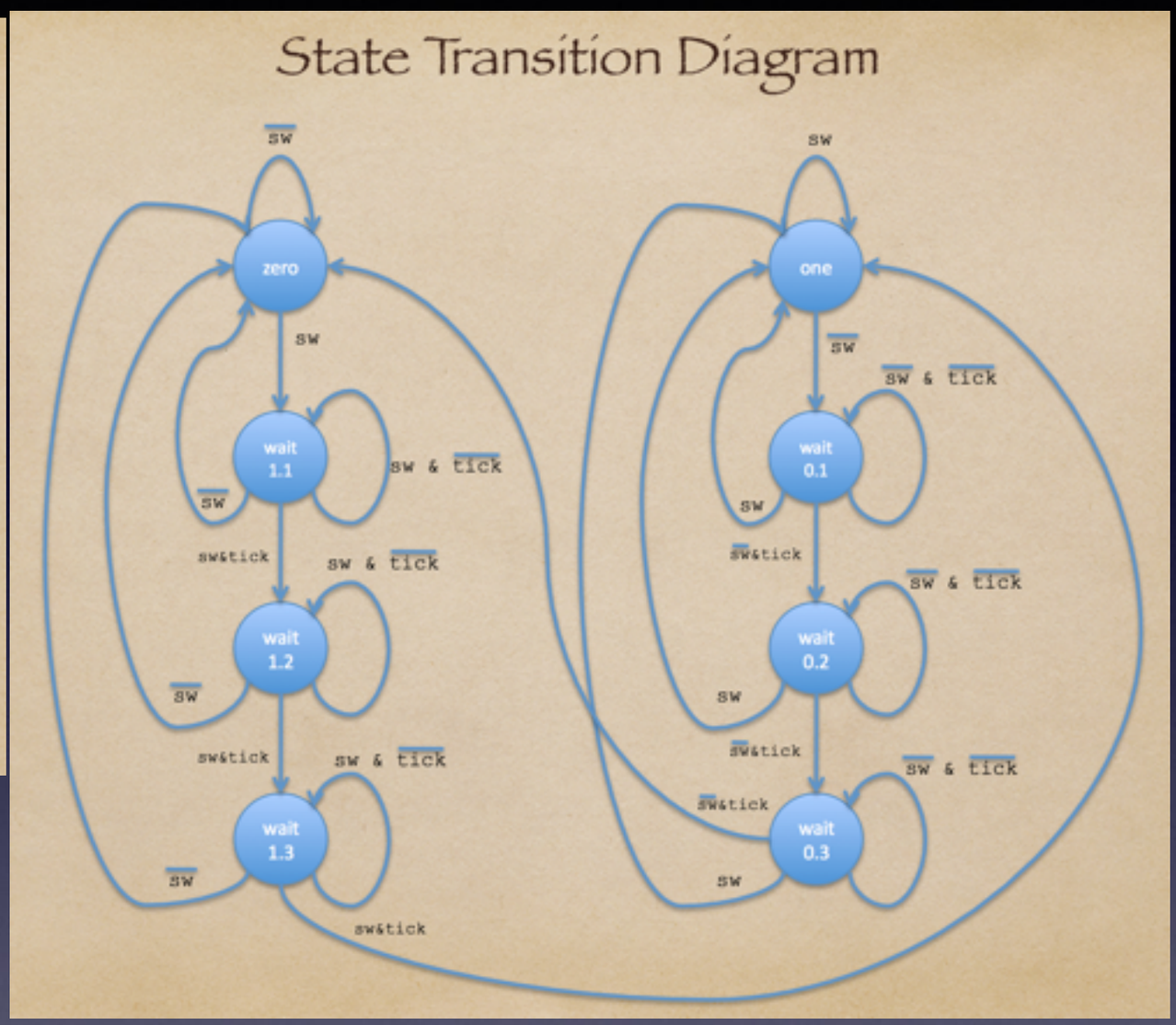
This is not a metastability issue. Remember that metastability is when the timing constraints of a flop are violated and then the output of the flop oscillates. The oscillation will stop in less than one clock period (20ns?). Here we are two orders of magnitude larger - 20ms. This is why our solution to debounce includes a state machine.

Debouncing Inputs from Mechanical Switches

A General Rule and a Proposed Solution

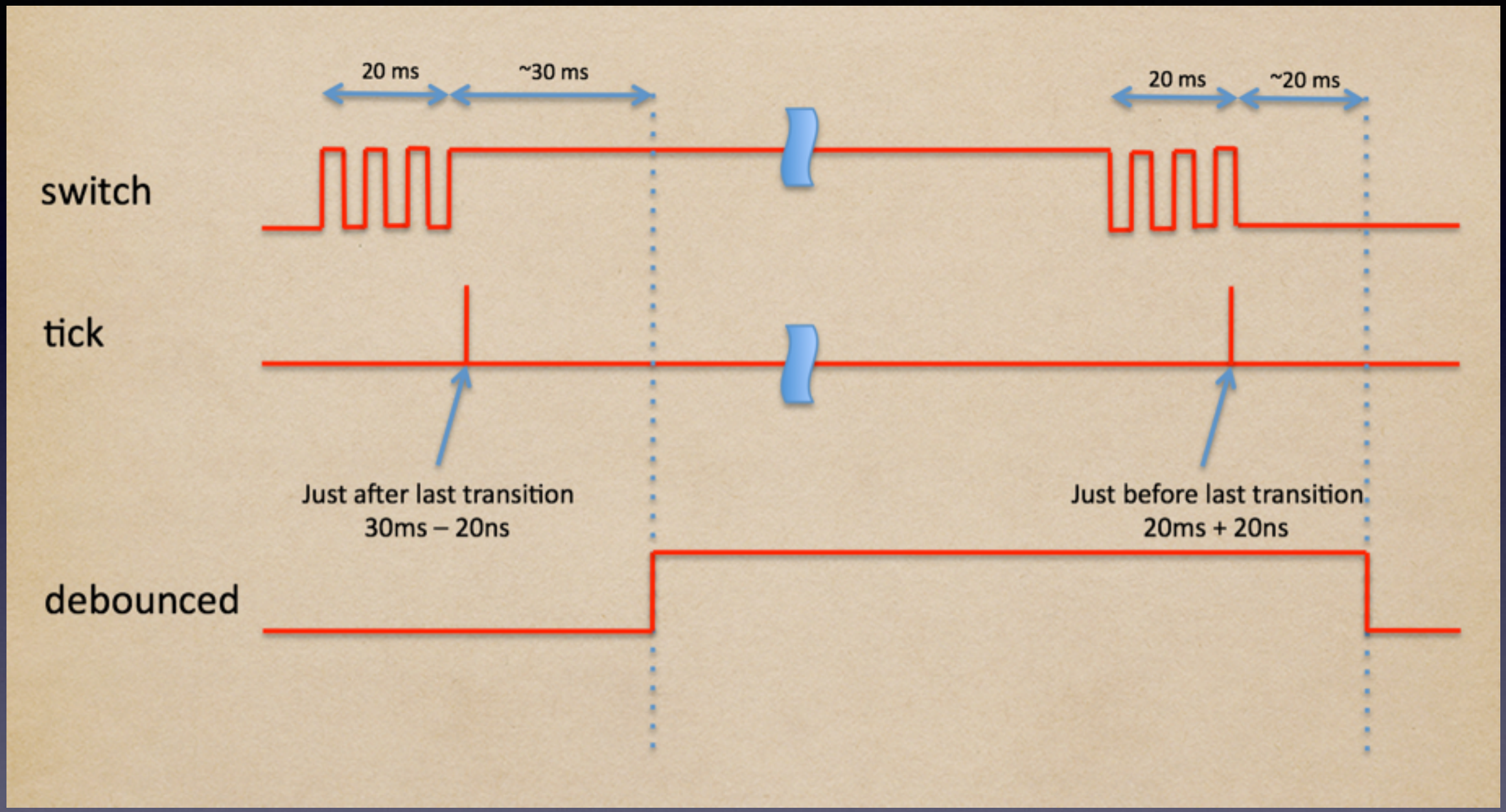
- After detecting the movement and then once the movement stops, wait for 20 ms before passing the switch movement on

State Transition Diagram



The slide on the left documents the desired behavior of our debounce circuit. The STD on the right documents how we will implement the debounce circuit using a state machine. In documenting the STD it is important to stress the role that ‘tick’ plays in the circuit. Its 10ms frequency allows the state machine to keep track of elapsed time in order to meet the desired behavior documented on the left.

Debouncing Inputs from Mechanical Switches



This documents the final implementation of the design. Please note that the relationship between the generation of tick and the input switch is not guaranteed. Sometimes switch comes just before the tick and sometimes it comes just after. This is why the end result is that the debounce takes from 20ms to 30ms.

Engineering Units (Our Language)

Proper use of units allow proper communication of information

The vocabulary of computer engineering assumes familiarity with Engineering Units

Scientific notation is a way to write numbers that are too big or small to be conveniently written in decimal form

- Scientific notation is: $a \times 10^b$
- Where a is {0..9}

We do not use Scientific Notation!

Standard decimal notation	Scientific notation
2	2×10^0
300	3×10^2
4,321.768	$4.321\ 768 \times 10^3$
−53,000	-5.3×10^4
6,720,000,000	6.72×10^9
0.2	2×10^{-1}
0.000 000 007 51	7.51×10^{-9}

- Engineering units
  - a is in {000 .. 999}
  - b is always in a power of 3
- Examples
  - 19×10^6 // also may be 19E6
  - 237×10^{-3} // also may be 237E-3

Some commonly used prefixes for the international system of units

Prefix	Symbol	Power of 10	Prefix	Symbol	Power of 10
exa	E	10^{18}	~~deci~~	~~d~~	~~10^{-1}~~
peta	P	10^{15}	~~centi~~	~~c~~	~~10^{-2}~~
tera	T	10^{12}	milli	m	10^{-3}
giga	G	10^9	micro	μ	10^{-6}
mega	M	10^6	nano	n	10^{-9}
kilo	k	10^3	pico	p	10^{-12}
~~hecto~~	~~h~~	~~10^2~~	femto	f	10^{-15}
~~deka~~	~~da~~	~~10^1~~	atto	a	10^{-18}

You must know how to communicate and manipulate numbers represented as engineering units. To read 10 ms and write 10E-6 is unacceptable. You must learn what the familiar terms represent. By this I mean kilo, mega, giga, tera, milli, micro, nano and pico. The world keeps improving its engineering capability so it it reasonable to assume you should know peta, exa, femto and atto as well.

Engineering Units - Learn to THINK

Simple Table for Frequencies (Clocks)

Frequency (Hz)	Engineering Unit	Period (Sec)	Engineering Unit
1	1 Hz	1	1 sec
10	10 Hz	.1	100 ms
100	100 Hz	.01	10 ms
1,000	1 kHz	.001	1 ms
10,000	10 kHz	.0001	100 us
100,000	100 kHz	.00001	10 us
1,000,000	1 MHz	.000001	1 us
10,000,000	10 MHz	.0000001	100 ns
100,000,000	100 MHz	.00000001	10 ns
1,000,000,000	1 GHz	.000000001	1 ns

kHz = E3	ms = E-3
MHz = E6	us = E-6
GHz = E9	ns = E-9

Simple Observation

Key

Fraction	Decimal
1/2	.5000
1/3	.3333
1/4	.2500
1/5	.2000
1/6	.1667
1/7	.1428
1/8	.1250
1/9	.1111
1/10	.1000

Examples

Frequency	Period
200 Hz	5 ms
33 MHz	30.30 ns
4 MHz	250 ns
50 MHz	20 ns
6 MHz	166.67 ns
70 MHz	14.28 ns
8 GHz	125 ps
9 Hz	111.11 ms
10 MHz	100 ns

The goal here is that you would be able to intuitively convert from frequency to period and vice-versa

This is your chosen profession - moving in the world of frequencies and periods is your future. You will be a better engineer if you are comfortable to how to move from frequencies to periods (and vice versa) without always reaching for a calculator.

Engineering Units - Application to Problems

(15) In our project we needed a circuit to generate a one clock wide pulse every 10ms. Assuming a 50MHz clock. Using WE THINK-WE COMPUTE-WE WRITE- WE GET approach, document this design.

Make sure you can read a problem description and understand the steps you need to take to correctly answer the question. Here there are two key items: a 50MHz clock and a one clock wide pulse every 10ms. The first question is how many 50MHz clocks are there in 10ms.

WE COMPUTE

$f = 50E6 \therefore T = \frac{1}{50E6} = .02E-6$

$= 20E-9$

FIND # CLOCKS IN 10ms

$\frac{10E-3}{20E-9} = .5E6 = 500E3$

$= 500\,000$

$\therefore 19$ bits

To answer this you need to find the period of the clock and then perform the division to determine how many clock periods are in 10ms. Once you have the count you need to know the powers of 2 to determine how many bits you require to represent this number.

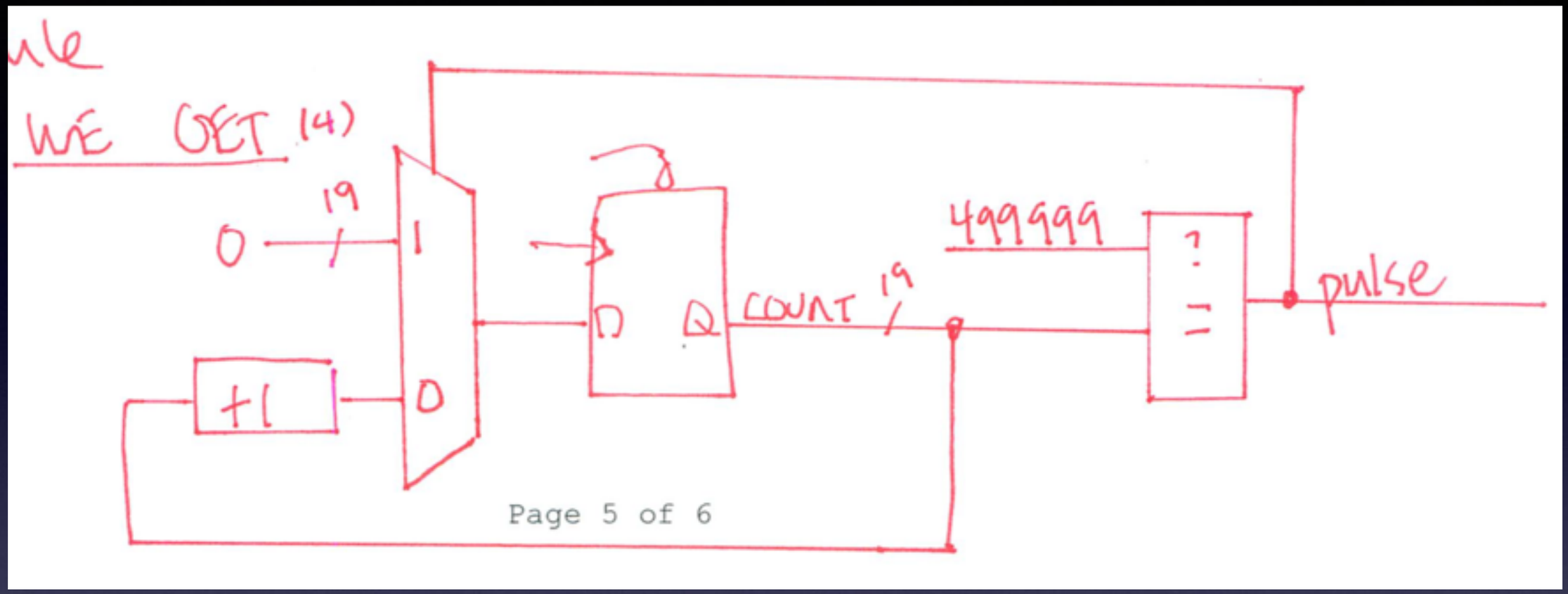
Engineering Units - Application to Problems

$2^{10} = 1024$ // 1k - this is a fundamental starting point
$2^{10} * 2^{10} = 2^{20}$ // 1M – you now know to represent 1M you need 20 bits

In this example the number you need to count to is 500,000. The decimal equivalent of 2^{20} is 1,048,576 so one half that, or 2^{19}, would be 524,288. It is readily apparent now that 500,00 is less than 524,288 so that the number of bits required to represent 500,000 would be 19.

The math is really not so complicated. Knowing the approximate values of the powers of 2 is a great assistance when determining what structures you need in your design to accommodate those values.

Anticipating the Synthesized Result of our Verilog



The circuit that is required to implement the required function is fairly straight forward. The pulse making function of this circuit is seen repeatedly in digital design. The symbology used here is not universal but I believe it is clear. There is a counter that must be wide enough to hold the maximum count of 500,000. That is why it is noted to be 19 bits wide. There is a comparator that indicates when the maximum count has been reached. The multiplexer on the input to the register is used to either increment the counter or clear the counter when the counter has reached its maximum count (rolled-over).

Writing the Verilog to Generate the Circuit

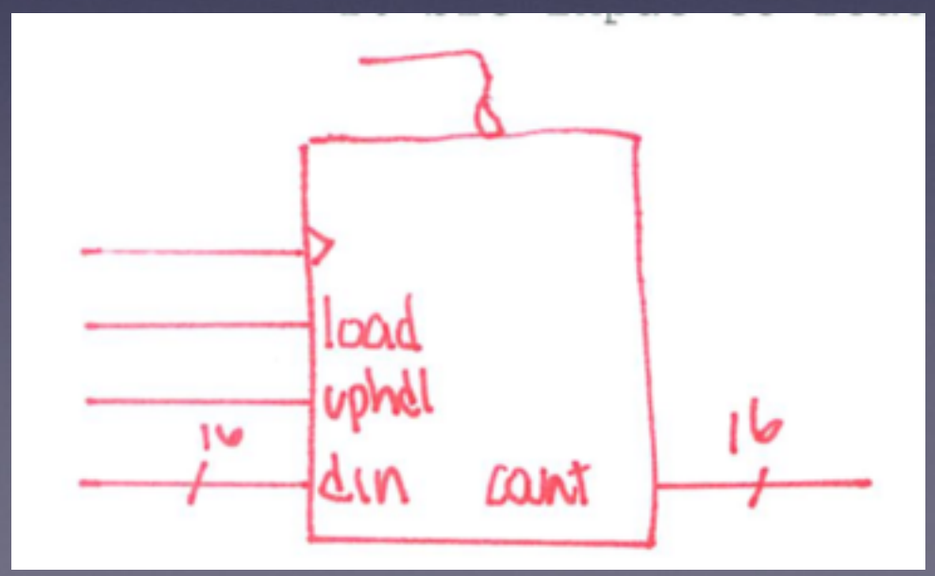
```
module veri (clk, rsth, pulse);
 input clk, rsth;
 output pulse;
 reg [18:0] count;
 wire pulse;
 assign pulse = (count == 499999);
 always @(↑clk, ↓rsth)
    if (! rsth)  count <= 19'b0; else
    if (pulse)   count <= 19'b0;
    else         count <= count + 1;
endmodule
```

The code matches the desired circuit. The if/else provides the reset to the flops as well as the multiplexer on the input to the flops. The comparison operation provides the comparing operation. Keep in mind the data type of the LHS of an ‘assign’ is always a wire and the data type of the LHS of an ‘always’ block is always a register (reg here)

Designing a Circuit to a Specification

(10) Write the Verilog code to implement a 16-bit up/down counter. The inputs are: clock, reset, load, uphdnl (1=count up/0=count down), and a 16-bit input to load (when load = 1).

You are not always guaranteed that a circuit description will be written in a perfect form. What you should be able to do is to break down a description and understand what you will need and how to implement it. In this case 1) a 16-bit register must be defined to serve as the counter. 2) Since there is a register (an array of flops) in the design you know that there must be reset and clock controlling the register. 3) Since there is a 'load' input that tells you that there must be a 16-bit input that will be loaded into the register when the 'load' signal is active. 4) When the load signal is not active then the counter will count. If 'uphdnl' is HIGH the counter should count up, if it is LOW the counter should count down.



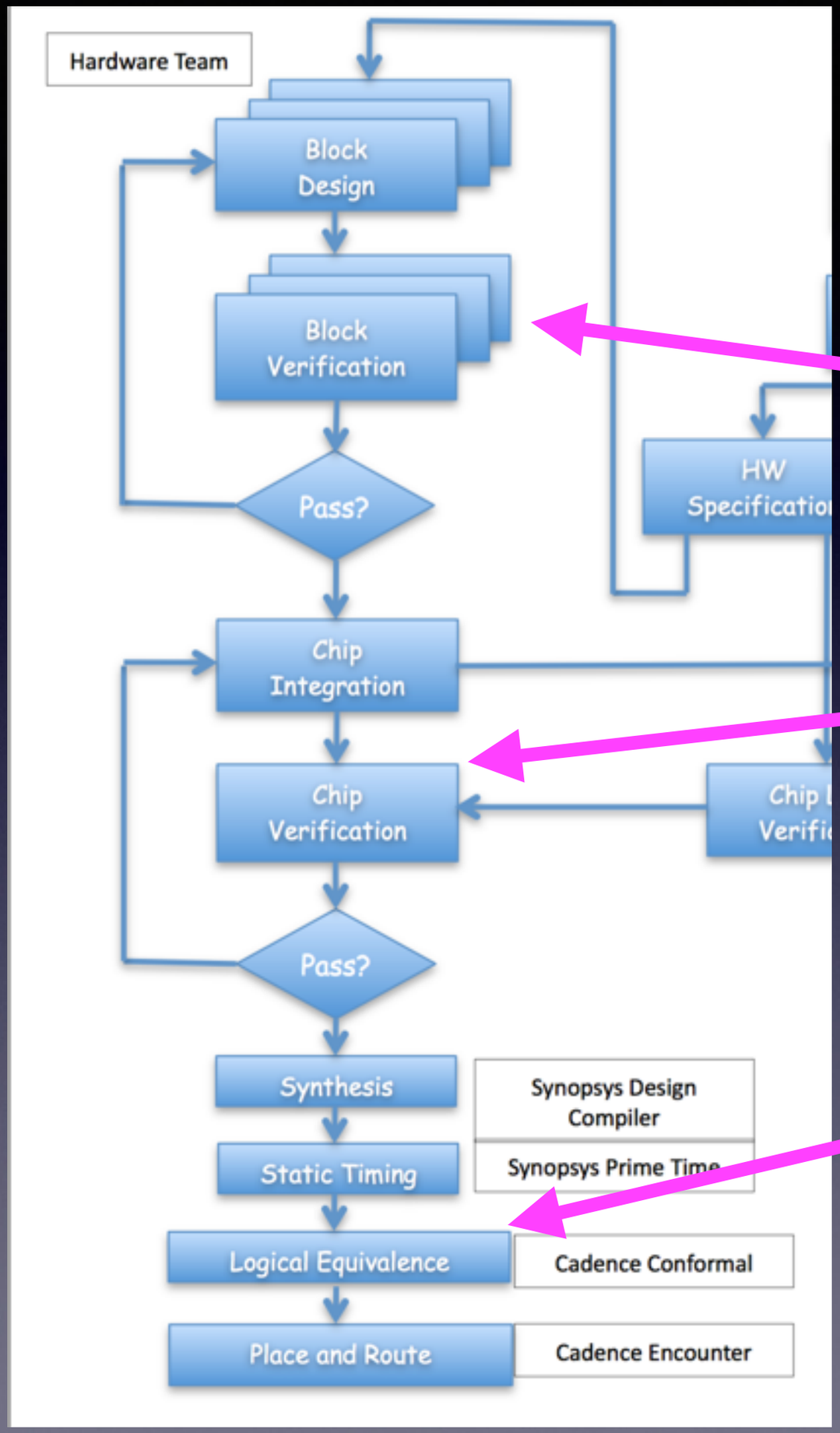
Before coding your module it is sometimes helpful to draw the top-level block diagram (we think) in order to make sure you have all the inputs/outputs accounted for. Here the reset is LOW ACTIVE, for us from now on it will be HIGH ACTIVE.

Designing a Circuit to a Specification

```
module countit (clk,rsth,load,uphdnlo,
                din, count);
input clk, rsth, load, uphdnl;
input [15:0] din;
output [15:0] count;                    7.3

always @(↑clk,↓rsth)
   if (!rsth) count <= 16'b0; else
   if (load)  count <= din ; else
   if (uphdnl) count <= count +1 ;
   else       count <= count -1 ;
endmodule
```

Writing the Verilog flows out from the top-level block diagram and the functional description created from the problem statement. Remember that simple is typically better (usually easier to understand). Once you have checked for a signal being HIGH the else case assumes it is LOW - you don't need to check again. The sequence that the if/else is constructed with sets the precedence for the operation of the circuit - here (as always) reset has the highest priority, then load, then if load is not active count either up or down.

460 Specific Topics



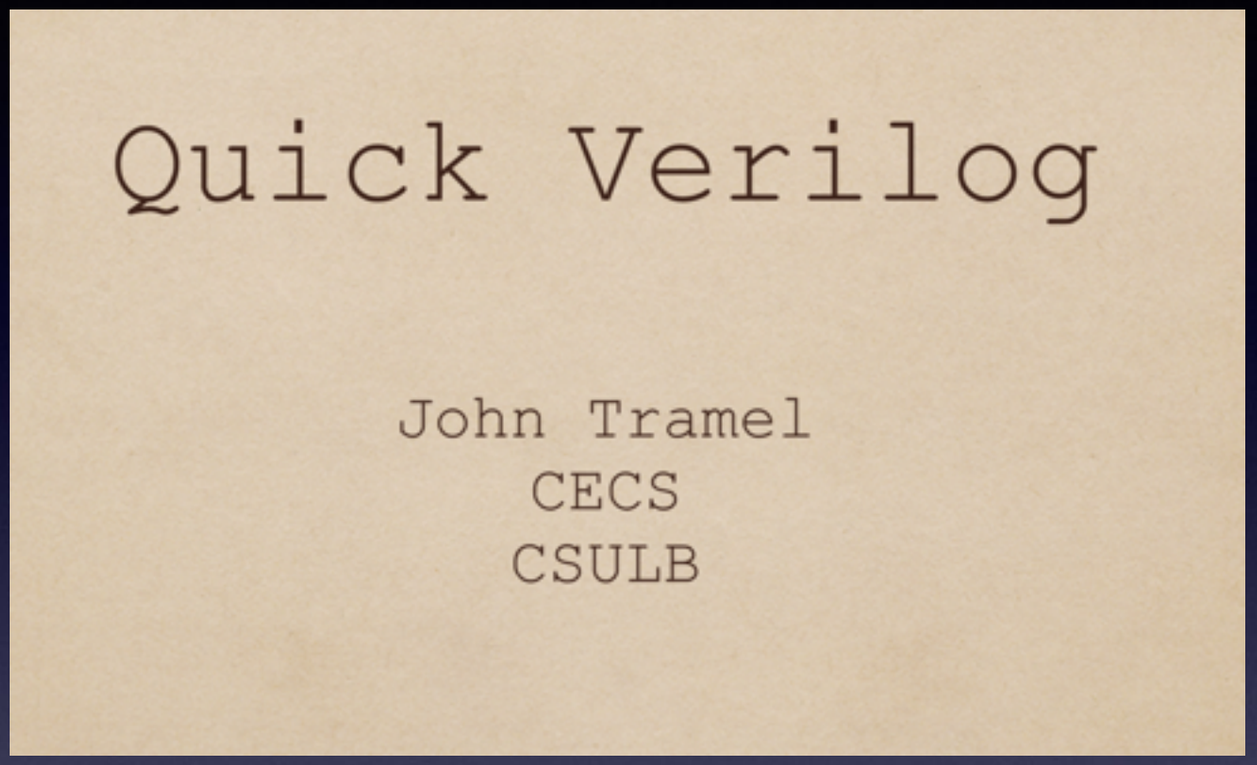
Verification Stages in the SOC Flow

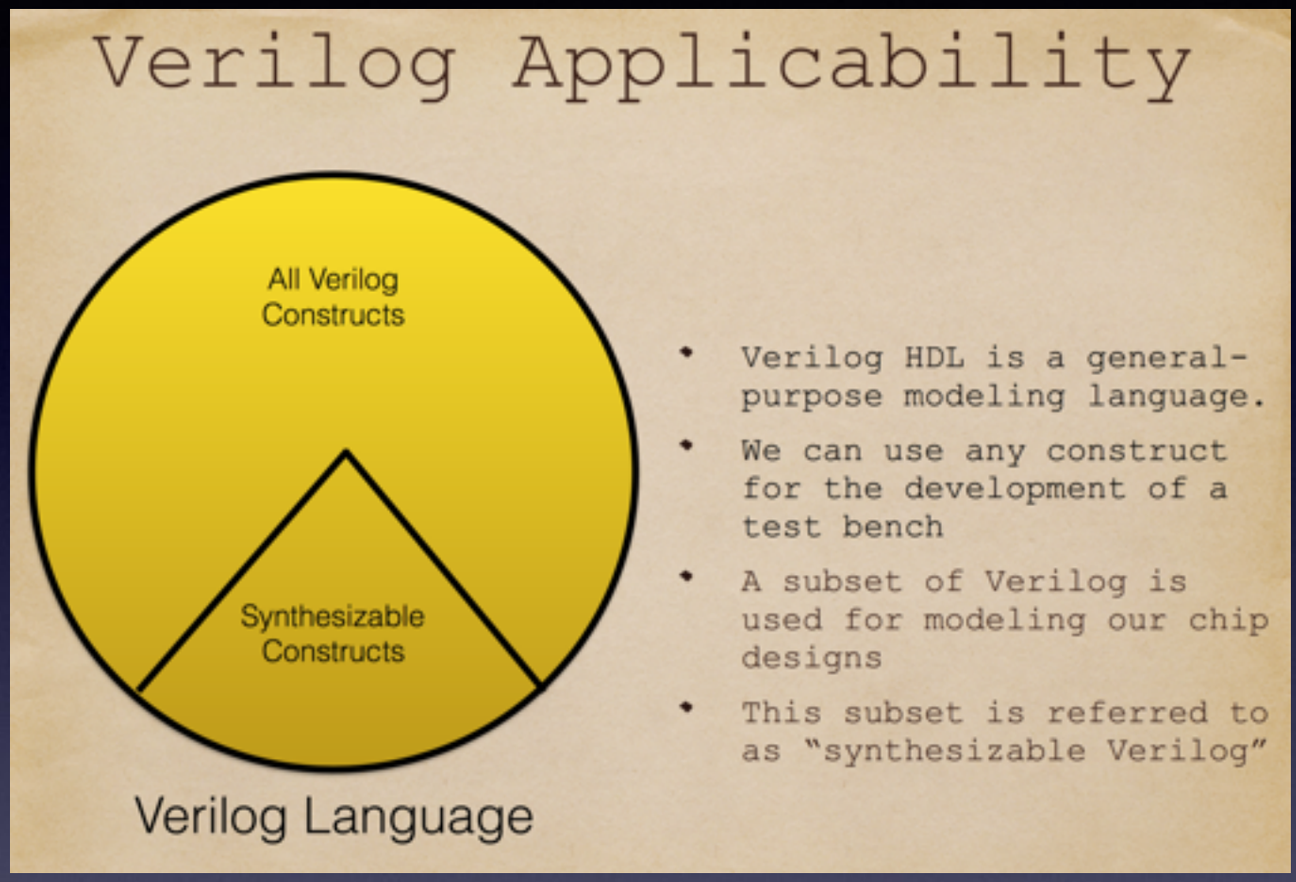
There are three main verification stations in the SOC flow presented in class. The first is the verification of block designs. The designers write their own test bench and simulate their design.

Once the team has completed verifying their blocks they integrate them together into to the top-level design. Here the verification is also performed by simulation but this time the test environment has been created by an independent team.

The last verification is making sure the netlist generated by the synthesis tool matches the ‘golden verilog’ code that was the input to synthesis. The verification is performed by Logical Equivalence Checking (LEC), or Formal Verification. This is a mathematical analysis not a simulation.

Verilog





I have encouraged you to study the documentation that has been available for 360 and now 460. The contents of these presentation contain important concepts for you to master the realm of Synchronous Digital Design

Verilog Operators

{}	Concatenation
+, -, *, /	Arithmetic
%	Modulus
~	bit-wise Negation
&	bit-wise/reduction AND
|	bit-wise/reduction OR
^	bit-wise/reduction XOR
^~, ~^	bit-wise/reduction XNOR
<<	Left shift
>>	Right shift
?:	Conditional
>, >=, <, <=	Relational
!	Logical Negation
&&	Logical AND
||	Logical OR
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality

Highest

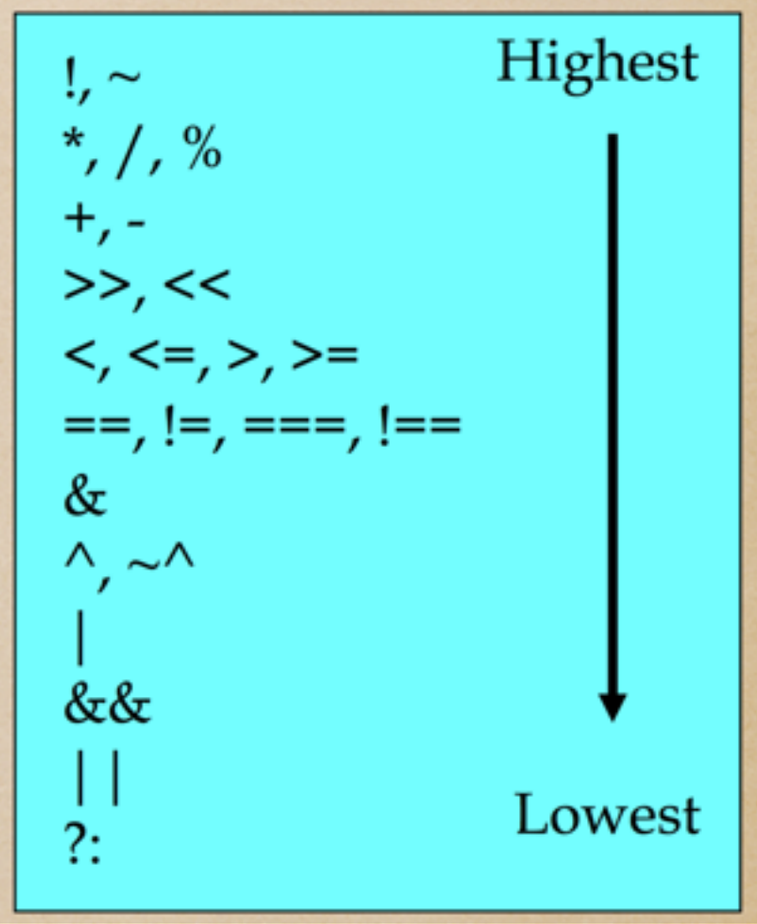
!, ~
*, /, %
+, -
>>, <<
<, <=, >, >=
==, !=, ===, !==
&
^, ~^
|
&&
||
?:

Lowest

You should know the operators that Verilog utilizes. It is of note that many of them are found in C/Java/Python - any modern programming language. Bitwise operators twiddle bits (0110 | 0100 = 0110). These include AND ‘&’, OR ‘|’, and EXCLUSIVE OR ‘^’. Logical operators deal with logical operands (T/F). Any operand fed to a logical operator is reduced to (T/F) and the result will always be (T/F). How do the tools treat TRUE/FALSE with respect to vectored operands? If any bit in the vector is a one the vector is considered to be TRUE. So - (0001 && 1000) will result in TRUE.

Verilog Operators

{}	Concatenation
+, -, *, /	Arithmetic
%	Modulus
~	bit-wise Negation
&	bit-wise/reduction AND
|	bit-wise/reduction OR
^	bit-wise/reduction XOR
^~, ~^	bit-wise/reduction XNOR
<<	Left shift
>>	Right shift
?:	Conditional
>, >=, <, <=	Relational
!	Logical Negation
&&	Logical AND
||	Logical OR
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality



'reduction' operators are unique to Verilog. They have only one operator and produce a scalar (1 bit wide) result. Let A = 10101011. ^A is an 'exclusive or reduction' and will exclusive or every bit in A together to produce a result of 1. |A is an 'or reduction' and will or all the bits together to produce a result also of 1. &A is an 'and reduction' and will and all the bits of A together to produce a 0.

If any of these concepts are not clear you should play around with the Xilinx simulator until you understand how they behave.

tramelblaze

Generating the Memory File for tb_rom

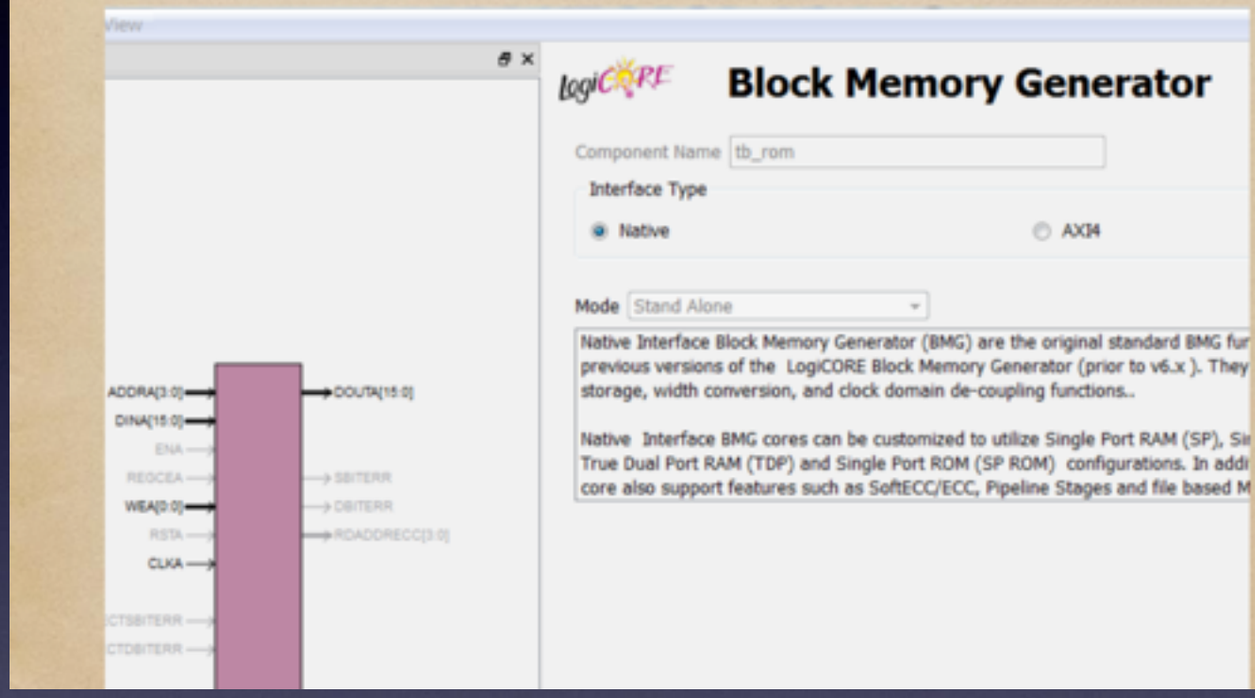
- With your .tba file in the same directory as the assembler – double click on the assembler and when prompted enter the full name of your .tba file (i.e. DemoOne.tba)
- You will see the following files generated in the directory

DemoTwo.coe	2/23/2016 9:34 AM	COE File	21 KB
DemoTwo.lst	2/23/2016 9:34 AM	LST File	2 KB
DemoTwo.sim	2/23/2016 9:34 AM	SIM File	1 KB
DemoTwo.tba	2/23/2016 9:29 AM	TBA File	1 KB

- The .lst file is meant to assist you in debugging your design. In the simulation you can follow along as the processor fetches and executes instructions
- The .sim file is for $readmemh and is not currently in our method
- The .coe file is the file we will program into the memory tb_rom

Block Memory Generator

- The Block Memory Generator window opens with our defined name (tb_rom) and we now will create our memory
- Next



I have a saying “that what I present in class I expect you to know.” The next portion of our project development will require the utilization of the embedded processor. I have presented the information you need to utilize the processor. It is in your best interest to work with it as soon as possible. The design of the UART will be complex, the utilization of the processor is not meant to be a complicated task.

tramelblaze - A 16-bit Emulator of the PicoBlaze


PicoBlaze 8-bit Embedded Microcontroller User Guide

for Spartan-3, Virtex-II, and Virtex-II Pro FPGAs

UG129 (v1.1.2) June 24, 2008

PicoBlaze™

As an emulator of the PicoBlaze the tramelblaze implements the same instruction set as the PicoBlaze and supports the same interface timing. To be comfortable using the tramelblaze you need to spend time reading the Xilinx document, UG129.

tramelblaze - A 16-bit Emulator of the PicoBlaze

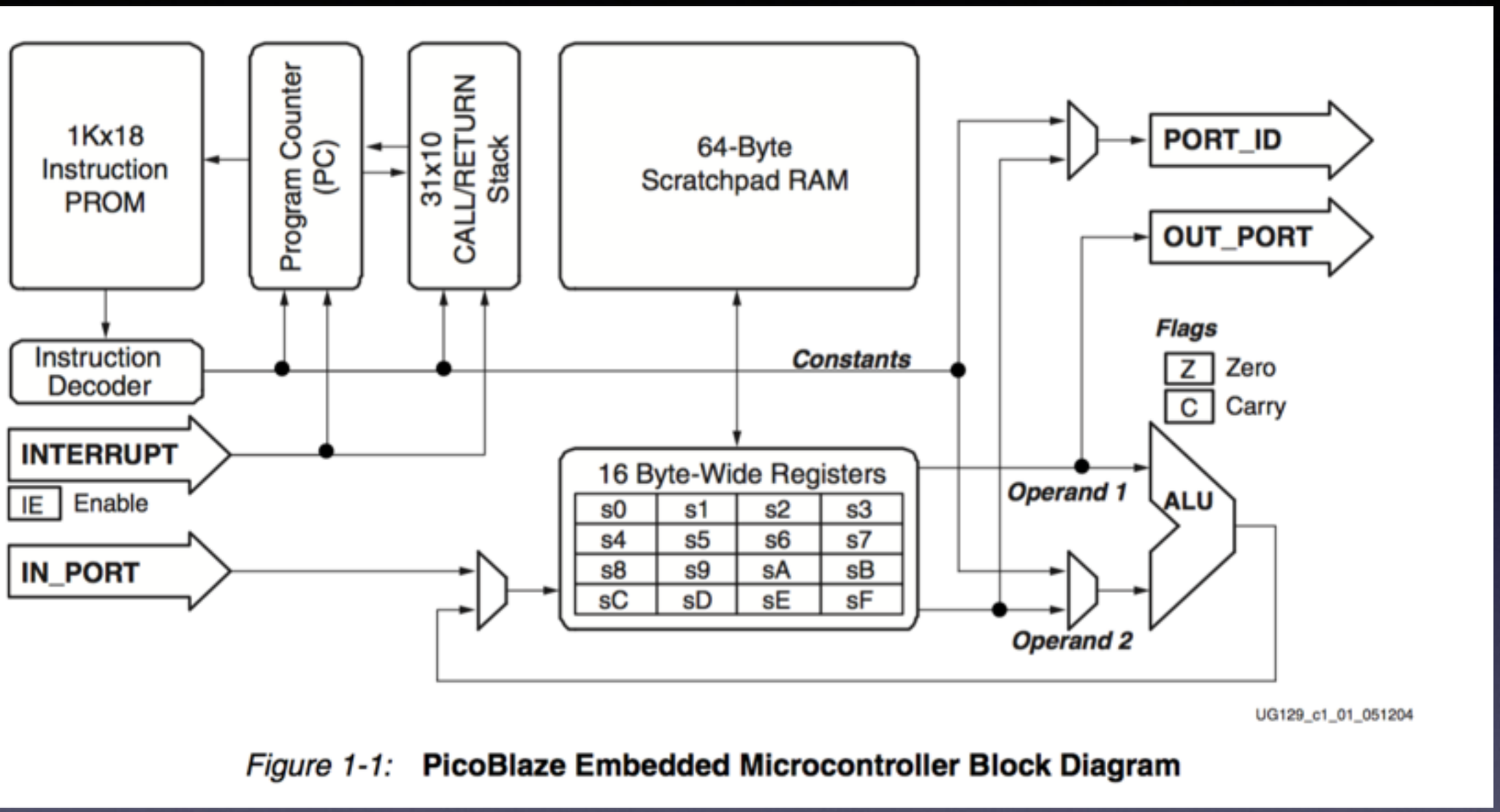
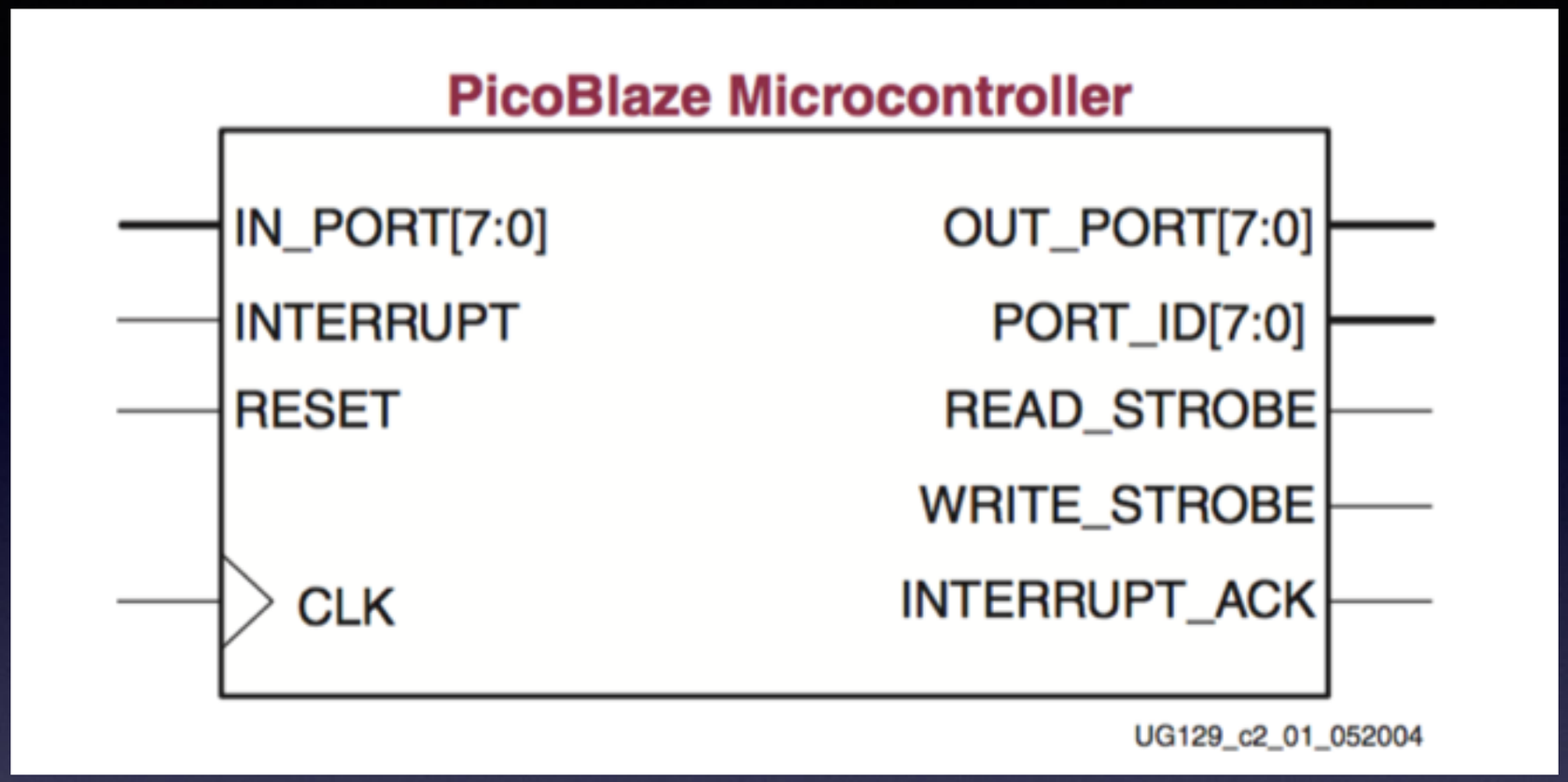


Figure 1-1: **PicoBlaze Embedded Microcontroller Block Diagram**

The processor architecture is the same. The difference is that the registers are 16 bits wide and the instruction memory is 4Kx16. The Scratchpad RAM will soon be implemented. What is particularly of interest to you is the external interface of the processor and the Instruction Set Architecture implemented.

tramelblaze - A 16-bit Emulator of the PicoBlaze



When you instantiate the tramelblaze_top into your design this is the interface that you will see. The instruction memory will be ‘hidden’ within the design. The ports on the diagram above (16 bits for us) are the signals you need to utilize to communicate with the other logic within your design.

tramelblaze - A 16-bit Emulator of the PicoBlaze

Table 3-1: **PicoBlaze Instruction Set (alphabetical listing)**

Instruction	Description	Function	ZERO	CARRY
ADD sX, kk	Add register sX with literal kk	sX ← sX + kk	?	?
ADD sX, sY	Add register sX with register sY	sX ← sX + sY	?	?
ADDCY sX, kk (ADDC)	Add register sX with literal kk with CARRY bit	sX ← sX + kk + CARRY	?	?
ADDCY sX, sY (ADDC)	Add register sX with register sY with CARRY bit	sX ← sX + sY + CARRY	?	?
AND sX, kk	Bitwise AND register sX with literal kk	sX ← sX AND kk	?	0
AND sX, sY	Bitwise AND register sX with register sY	sX ← sX AND sY	?	0
CALL aaa	Unconditionally call subroutine at aaa	TOS ← PC PC ← aaa	-	-
CALL C, aaa	If CARRY flag set, call subroutine at aaa	If CARRY=1, {TOS ← PC, PC ← aaa}	-	-
CALL NC, aaa	If CARRY flag not set, call subroutine at aaa	If CARRY=0, {TOS ← PC, PC ← aaa}	-	-

The instructions contained within the document and the explanation of how they operate apply to the tramelblaze. Any question regarding how an instruction operates will be answered by reading the document.

tramelblaze - A 16-bit Emulator of the PicoBlaze

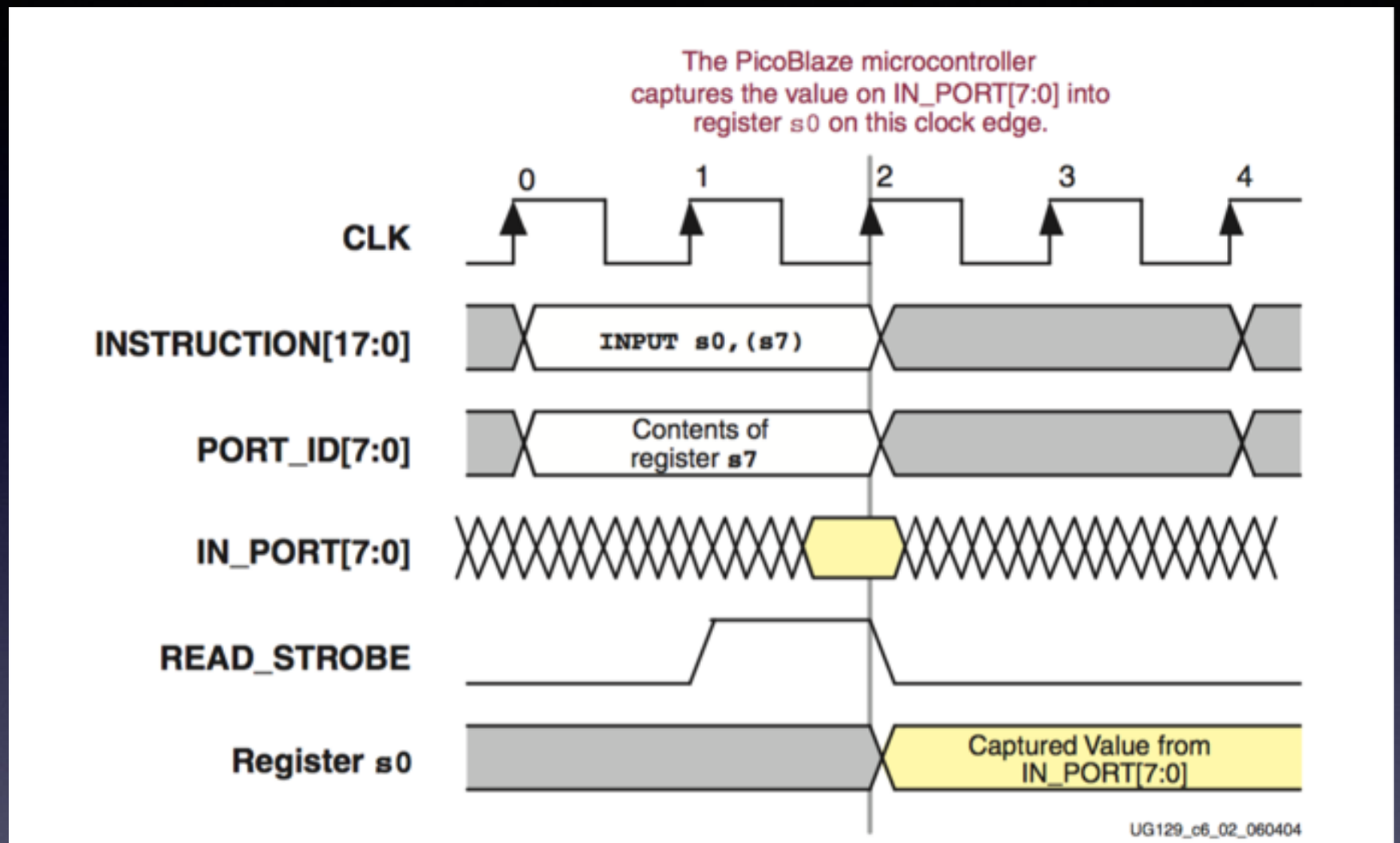


Figure 6-2: **Port Timing for INPUT Instruction**

The timing for interfacing to the tramelblaze is contained within the document. It clearly documents the timing of the processor. This will determine how you interface to the processor. This is for an INPUT. There is a similar slide for the OUTPUT.