EE222 Lecture 15 Feb. 26, 2019

Low Power Design of Logic Circuits:

- Combinational circuits
  - Mixed signal (digital and analog) circuits for processing in memory (PIM)
- Sequential circuits
- Clock distribution

Combinational Vs. Sequential Circuits

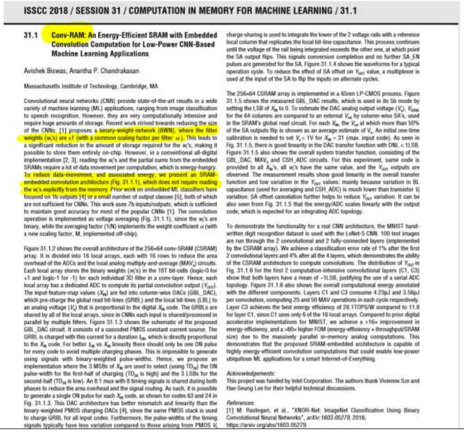
Combinational (or Combinatorial) Networks/Circuits

- Circuit without storage
- Outputs depend only on its current inputs
- Examples: NAND gate, look-up table (LUT)

Vs.

Sequential Networks/Circuits

- Circuit with storage elements
- Outputs depend on present inputs and also on history of inputs
- Examples: RAM, finite state machine (FSM)
- Normally, combinational cells AND storage elements



ISSCC 2018 / SESSION 31 / COMPUTATION IN MEMORY FOR MACHINE LEARNING / 31.1

31.1 Conv-RAM: An Energy-Efficient SRAM with Embedded Convolution Computation for Low-Power CNN-Based Machine Learning Applications

Avishek Biswas, Anantha P. Chandrakasan

Massachusetts Institute of Technology, Cambridge, MA

Convolutional neural networks (CNN) provide state-of-the-art results in a wide variety of machine learning (ML) applications, ranging from image classification to speech recognition. However, they are very computationally intensive and require huge amounts of storage. Recent work strived towards reducing the size of the CNNs: [1] proposes a binary-weight-network (BWN), where the filter weights (w_i's) are ±1 (with a common scaling factor per filter: α). This leads to a significant reduction in the amount of storage required for the w_i's, making it possible to store them entirely on-chip. However, in a conventional all-digital implementation [2, 3], reading the w_i's and the partial sums from the embedded SRAMs require a lot of data movement per computation, which is energy-hungry. To reduce data-movement, and associated energy, we present an SRAM-embedded convolution architecture (Fig. 31.1.1), which does not require reading the w_i's explicitly from the memory. Prior work on embedded ML classifiers have focused on 1b outputs [4] or a small number of output classes [5], both of which are not sufficient for CNNs. This work uses 7b inputs/outputs, which is sufficient to maintain good accuracy for most of the popular CNNs [1]. The convolution operation is implemented as voltage averaging (Fig. 31.1.1), since the w_i's are binary, while the averaging factor (1/N) implements the weight-coefficient α (with a new scaling factor, M, implemented off-chip).

Figure 31.1.2 shows the overall architecture of the 256×64 conv-SRAM (CSRAM) array. It is divided into 16 local arrays, each with 16 rows to reduce the area overhead of the ADCs and the local analog multiply-and-average (MAV_a) circuits. Each local array stores the binary weights (w_i's) in the 10T bit-cells (logic-0 for +1 and logic-1 for -1) for each individual 3D filter in a conv-layer. Hence, each local array has a dedicated ADC to compute its partial convolution output (Y_{OUT}). The input-feature-map values (X_{IN}) are fed into column-wise DACs (GBL_DAC), which pre-charge the global read bit-lines (GRBL) and the local bit-lines (LBL) to an analog voltage (V_a) that is proportional to the digital X_{IN} code. The GRBLs are shared by all of the local arrays, since in CNNs each input is shared/processed in parallel by multiple filters. Figure 31.1.3 shows the schematic of the proposed GBL_DAC circuit. It consists of a cascoded PMOS constant current source. The GRBL is charged with this current for a duration t_{ON}, which is directly proportional to the X_{IN} code. For better t_{ON} vs X_{IN} linearity there should only be one ON pulse for every code to avoid multiple charging phases. This is impossible to generate using signals with binary-weighted pulse-widths. Hence, we propose an implementation where the 3 MSBs of X_{IN} are used to select (using TD_{56}) the ON pulse-width for the first-half of charging (TD_{56} is high) and the 3 LSBs for the second-half (TD_{56} is low). An 8:1 mux with 8 timing signals is shared during both phases to reduce the area overhead and the signal routing. As such, it is possible to generate a single ON pulse for each X_{IN} code, as shown for codes 63 and 24 in Fig. 31.1.3. This DAC architecture has better mismatch and linearity than the binary-weighted PMOS charging DACs [4], since the same PMOS stack is used to charge GRBL for all input codes. Furthermore, the pulse-widths of the timing signals typically have less variation compared to those arising from PMOS V_t charge-sharing is used to integrate the lower of the 2 voltage rails with a reference local column that replicates the local bit-line capacitance. This process continues until the voltage of the rail being integrated exceeds the other one, at which point the SA output flips. This signals conversion completion and no further SA_EN pulses are generated for the SA. Figure 31.1.4 shows the waveforms for a typical operation cycle. To reduce the effect of SA offset on Y_{OUT} value, a multiplexer is used at the input of the SA to flip the inputs on alternate cycles.

The 256×64 CSRAM array is implemented in a 65nm LP-CMOS process. Figure 31.1.5 shows the measured GBL_DAC results, which is used in its 5b mode by setting the LSB of X_{IN} to 0. To estimate the DAC analog output voltage (V_a), V_{GRBL} for the 64 columns are compared to an external V_{ref} by column-wise SA's, used in the SRAM's global read circuit. For each X_{IN}, the V_{ref} at which more than 50% of the SA outputs flip is chosen as an average estimate of V_a. An initial one-time calibration is needed to set V_a = 1V for X_{IN} = 31 (max. input code). As seen in Fig. 31.1.5, there is good linearity in the DAC transfer function with DNL < 1LSB. Figure 31.1.5 also shows the overall system transfer function, consisting of the GBL_DAC, MAV_a and CSH_ADC circuits. For this experiment, same code is provided to all X_{IN}'s, all w_i's have the same value, and the Y_{OUT} outputs are observed. The measurement results show good linearity in the overall transfer function and low variation in the Y_{OUT} values: mainly because variation in BL capacitance (used for averaging and CSH_ADC) is much lower than transistor V_t variation. SA offset cancelation further helps to reduce Y_{OUT} variation. It can be also seen from Fig. 31.1.5 that the energy/ADC scales linearly with the output code, which is expected for an integrating ADC topology.

To demonstrate the functionality for a real CNN architecture, the MNIST handwritten digit recognition dataset is used with the LeNet-5 CNN. 100 test images are run through the 2 convolutional and 2 fully-connected layers (implemented by the CSRAM array). We achieve a classification error rate of 1% after the first 2 convolutional layers and 4% after all the 4 layers, which demonstrates the ability of the CSRAM architecture to compute convolutions. The distribution of Y_{OUT} in Fig. 31.1.6 for the first 2 computation-intensive convolutional layers (C1, C3) show that both layers have a mean of ~1LSB, justifying the use of a serial ADC topology. Figure 31.1.6 also shows the overall computational energy annotated with the different components. Layers C1 and C3 consume 4.23pJ and 3.56pJ per convolution, computing 25 and 50 MAV operations in each cycle respectively. Layer C3 achieves the best energy efficiency of 28.1TOPS/W compared to 11.8 for layer C1, since C1 uses only 6 of the 16 local arrays. Compared to prior digital accelerator implementations for MNIST, we achieve a >16× improvement in energy-efficiency, and a >60× higher FOM (energy-efficiency × throughput/SRAM size) due to the massively parallel in-memory analog computations. This demonstrates that the proposed SRAM-embedded architecture is capable of highly energy-efficient convolution computations that could enable low-power ubiquitous ML applications for a smart Internet-of-Everything.

Acknowledgements:
This project was funded by Intel Corporation. The authors thank Vivienne Sze and Hae-Seung Lee for their helpful technical discussions.

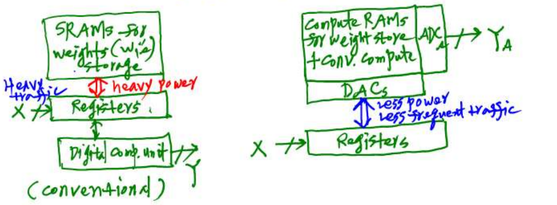
References:
[1] M. Rastegari, et al., "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks", *arXiv:1603.05279*, 2016. https://arxiv.org/abs/1603.05279

Computation in Memory (CIM)

(Also called Processing in Memory (PIM))

Ref. Convolution-RAM: An Energy Efficient SRAM with Embedded Convolution Computation for Low Power CNN (Convolution Neural Network) - based Machine Learning Applications

ISSCC 2018 A. Biswas, et al. MIT



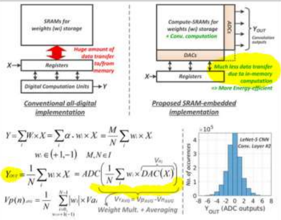

SRAMs for weights (wi) storage
Huge amount of data transfer to/from memory
X
Registers
Digital Computation Units
Y
Conventional all-digital implementation
Compute-SRAMs for weights (wi) storage + Conv. computation
ADCs
Y_{OUT}
Convolution outputs
DACs
Much less data transfer due to in-memory computation => More Energy-efficient
X
Registers
Proposed SRAM-embedded implementation
$Y=\sum_i W_i\times X_i=\sum_i \alpha\cdot w_i\times X_i=\frac{M}{N}\sum_i w_i\times X_i$
$w_i\in(+1,-1)\quad M,N\in I$
$Y_{OUT}=\frac{1}{N}\sum_i w_i\times X_i=ADC\left(\frac{1}{N}\sum_i w_i\times DAC(X_i)\right)$
V_{Ai}
$Vp(n)_{AVG}=\frac{1}{N}\sum|w_i|\times Va_i$
$Vr_{AVG}=Vp_{AVG}-Vn_{AVG}$
Weight Mult. + Averaging
$\times 10^5$
LeNet-5 CNN Conv. Layer #2
No. of occurences
Y_{OUT} (ADC outputs)
-8 -6 -4 -2 0 2 4 6 8
0 1 2 3 4

Figure 31.1.1: Concept of embedded convolution computation, performed by averaging in SRAM, for binary-weight convolutional neural networks.

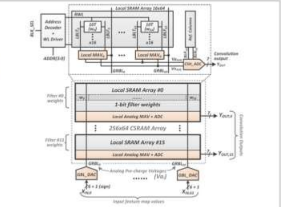


Figure 31.1.2: Overall architecture of the Convolution-SRAM (CSRAM) showing local arrays, column-wise DACs and row-wise ADCs to implement convolution as weighted averaging.

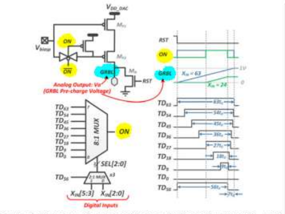


Figure 31.1.3: Schematic and timing diagram for the column-wise GBL_DAC, which converts the convolution digital input to an analog pre-charge voltage for the SRAM.

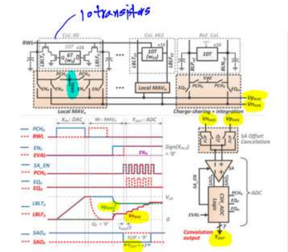
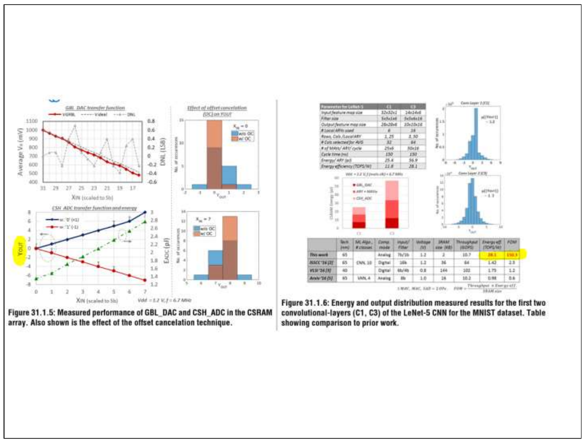


Figure 31.1.4: Architecture for the row-wise multiply-and-average (MAV_a) and CSH_ADC. Operational waveforms for convolution computation.



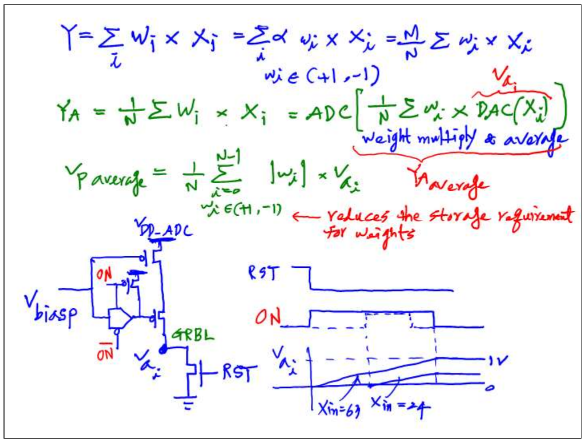
Parameters for LeNet-5	C1	C3
Input feature map size	32x32x1	14x14x6
Filter size	5x5x1x6	5x5x6x16
Output feature map size	28x28x6	10x10x16
# Local ARYs used	6	16
Rows, Cols /Local ARY	1, 25	3, 50
# Cols selected for AVG	32	64
# of MAVs/ ARY/ cycle	25x6	50x16
Cycle time (ns)	150	150
Energy/ ARY (pJ)	25.4	56.9
Energy efficiency (TOPS/W)	11.8	28.1

	Tech (nm)	ML Algo., # classes	Comp. mode	Input/ Filter	Voltage (V)	SRAM size (KB)	Throughput (GOPS)	Energy eff (TOPS/W)	FOM
This work	65	CNN, 10	Analog	7b/1b	1.2	2	10.7	28.1	150.3
ISSCC'16 [2]	65		Digital	16b	1.2	36	64	1.42	2.5
VLSI'16 [3]	40		Digital	6b/4b	0.8	144	102	1.75	1.2
Arxiv'16 [5]	65	VNN, 4	Analog	8b	1.0	16	10.2	0.98	0.6

1 MAV, MAC, SAD = 2 OPs. $FOM = \frac{Throughput \times Energy\ eff}{SRAM\ size}$

Figure 31.1.5: Measured performance of GBL_DAC and CSH_ADC in the CSRAM array. Also shown is the effect of the offset cancelation technique.

Figure 31.1.6: Energy and output distribution measured results for the first two convolutional-layers (C1, C3) of the LeNet-5 CNN for the MNIST dataset. Table showing comparison to prior work.



$$Y = \sum_i W_i \times X_i = \sum_i \alpha\, w_i \times X_i = \frac{M}{N} \sum w_i \times X_i$$

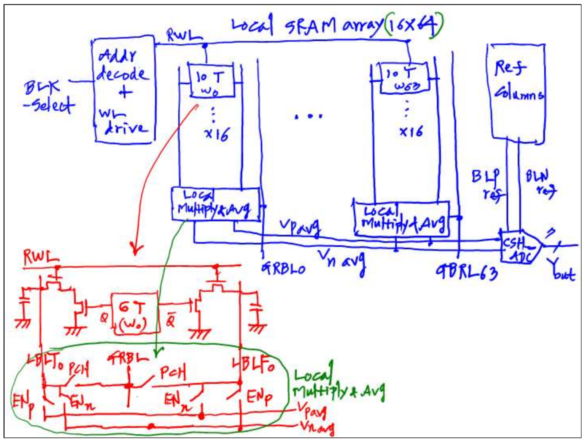
$w_i \in (+1, -1)$

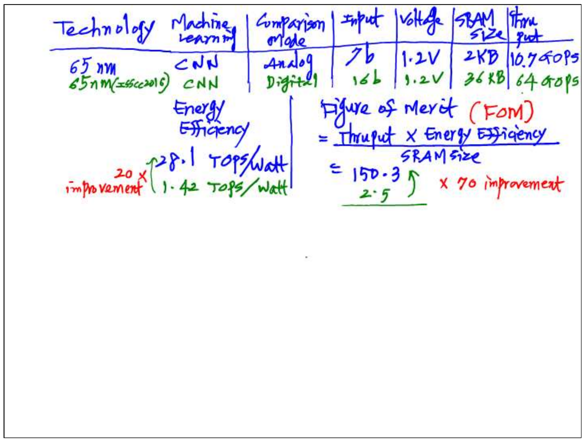
$$Y_A = \frac{1}{N} \sum W_i \times X_i = ADC\Big[\underbrace{\frac{1}{N} \sum w_i \times \overbrace{DAC(X_i)}^{V_{a_i}}}_{Y_{A\,average}}\Big]$$

weight multiply & average

$$V_{P\,average} = \frac{1}{N} \sum_{i=0}^{N-1} |w_i| \times V_{a_i}$$

$w_i \in (+1, -1)$ ← reduces the storage requirement for weights





Technology	Machine learning	Comparison mode	Input	Voltage	SRAM size	Thru put
65 nm	CNN	Analog	7b	1.2V	2KB	10.7 GOPS
65nm (ISSCC2016)	CNN	Digital	16b	1.2V	36KB	64 GOPS

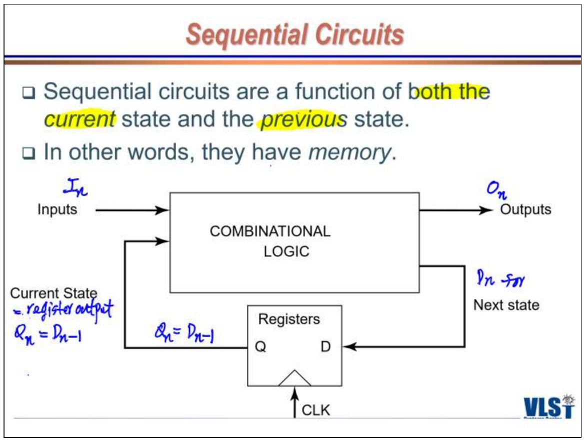
Energy Efficiency

20 x improvement $\left\{\begin{array}{l} 28.1 \text{ TOPS/Watt} \\ 1.42 \text{ TOPS/Watt} \end{array}\right.$

Figure of Merit (FOM)

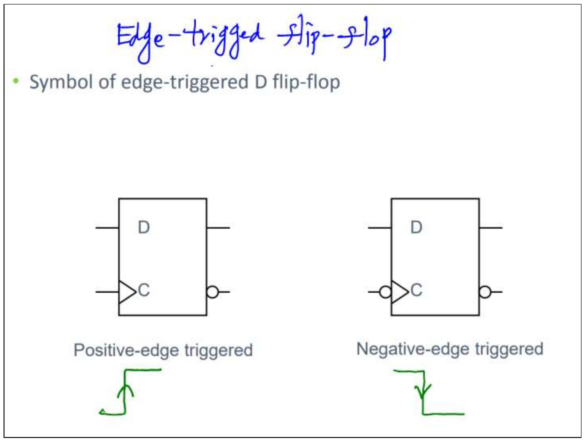
$$= \frac{\text{Thruput} \times \text{Energy Efficiency}}{\text{SRAM size}}$$

$$= \frac{150.3}{2.5}$$ x 70 improvement



Sequential Circuits

- ❑ Sequential circuits are a function of both the *current* state and the *previous* state.
- ❑ In other words, they have *memory*.

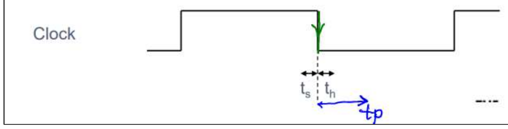


Edge-trigged flip-flop

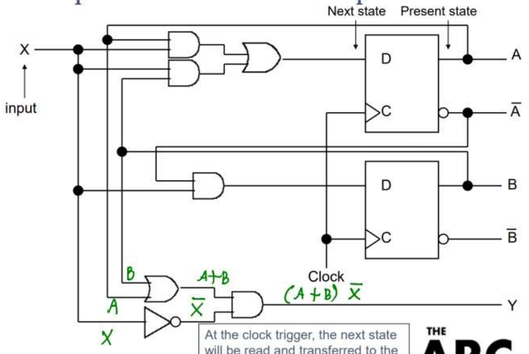
- Symbol of edge-triggered D flip-flop

Flip-Flop Timing

- Set-up time: t_s
  - Input needs to be stable before trigger
- Hold time: t_h
  - Input needs to be stable after trigger
- Propagation delay: t_p
  - Some delay from trigger to output change
- Example: Negative edge triggered flip-flip



Sequential Circuit Description



At the clock trigger, the next state will be read and transferred to the present state

Input Equations

$A_{next} = A_{present}X + B_{present}X$

$B_{next} = A'_{present}X$

Next state in terms of input and present state

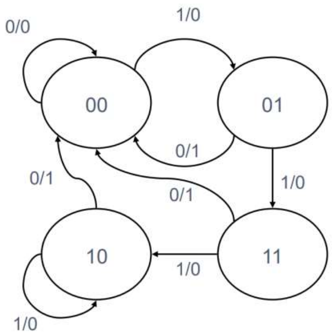
$Y = (A_{present} + B_{present})\overline{X'}$

Output in terms of input and present state

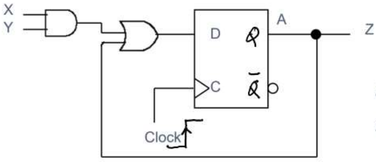
State Table

Present State		Input	Next State		Output
A	B	X	A	B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

State Diagram



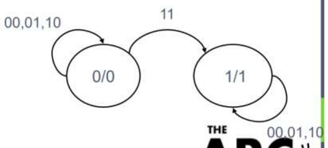
Example of Moore Model



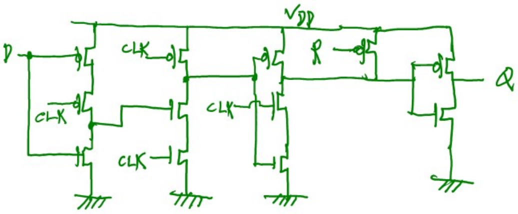
$A_{next} = A_{present} + XY$

$Z = A_{present}$

X	Y	$A_{present}$	A_{next}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



A True single-phase clock edge triggered flip-flop w/Reset (R)



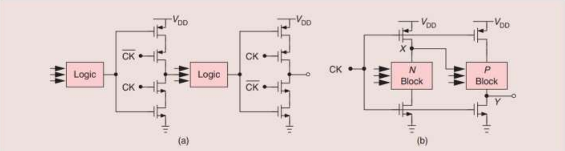


FIGURE 1: (a) C²MOS logic and (b) an example of single-phase clocking.

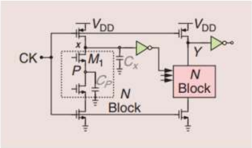


FIGURE 2: Domino logic.

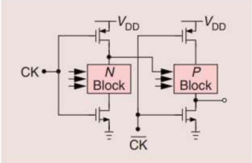


FIGURE 3: NORA logic.

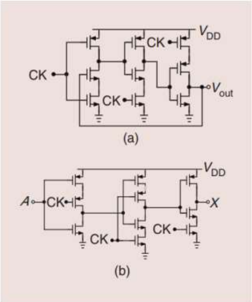


FIGURE 4: (a) A single-phase frequency divider reported by Oguey and Vittoz in 1973 and (b) a latch filed for patent by Piguet in 1974.

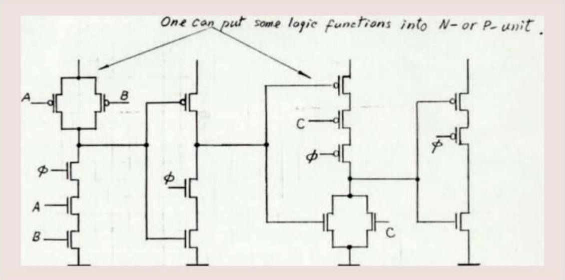


FIGURE 5: Yuan's original drawing of a TSPC circuit.

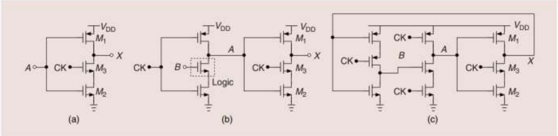


FIGURE 6: (a) A dynamic latch with a single clocked device, (b) cascaded TSPC stages, and (c) a three-stage master-slave flip-flop operating as a frequency divider.

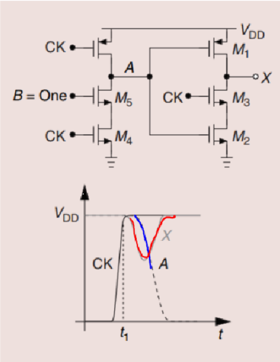
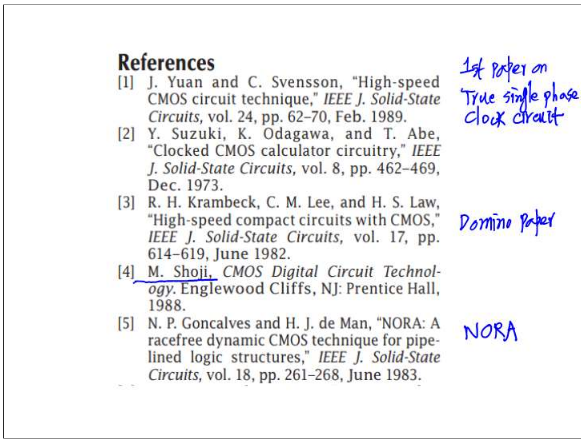


FIGURE 9: The problem of race in TSPC stages.



References

[1] J. Yuan and C. Svensson, "High-speed CMOS circuit technique," *IEEE J. Solid-State Circuits*, vol. 24, pp. 62–70, Feb. 1989.

[2] Y. Suzuki, K. Odagawa, and T. Abe, "Clocked CMOS calculator circuitry," *IEEE J. Solid-State Circuits*, vol. 8, pp. 462–469, Dec. 1973.

[3] R. H. Krambeck, C. M. Lee, and H. S. Law, "High-speed compact circuits with CMOS," *IEEE J. Solid-State Circuits*, vol. 17, pp. 614–619, June 1982.

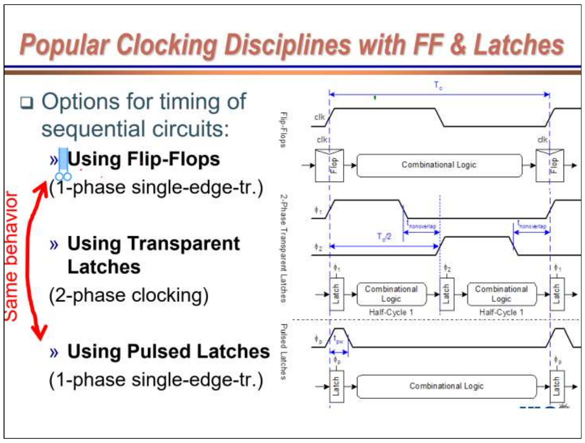
[4] M. Shoji, *CMOS Digital Circuit Technology*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[5] N. P. Goncalves and H. J. de Man, "NORA: A racefree dynamic CMOS technique for pipelined logic structures," *IEEE J. Solid-State Circuits*, vol. 18, pp. 261–268, June 1983.

1st Paper on True single phase clock circuit

Domino Paper

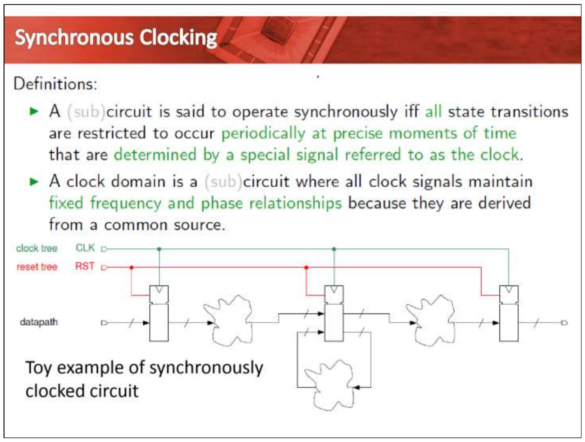
NORA



Popular Clocking Disciplines with FF & Latches

- ❑ Options for timing of sequential circuits:
  - » **Using Flip-Flops** (1-phase single-edge-tr.)
  - » **Using Transparent Latches** (2-phase clocking)
  - » **Using Pulsed Latches** (1-phase single-edge-tr.)

Same behavior

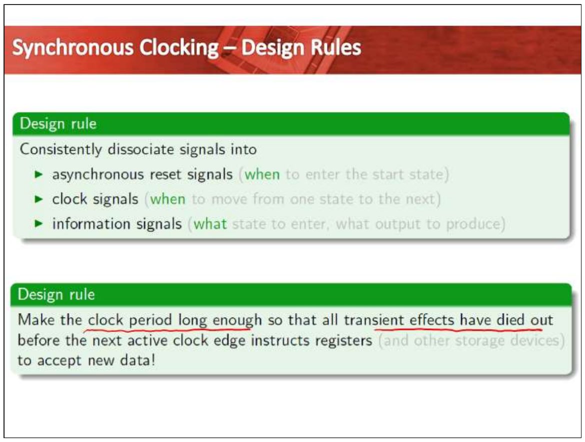


Synchronous Clocking

Definitions:

- A (sub)circuit is said to operate synchronously iff all state transitions are restricted to occur periodically at precise moments of time that are determined by a special signal referred to as the clock.
- A clock domain is a (sub)circuit where all clock signals maintain fixed frequency and phase relationships because they are derived from a common source.

Toy example of synchronously clocked circuit



Synchronous Clocking – Design Rules

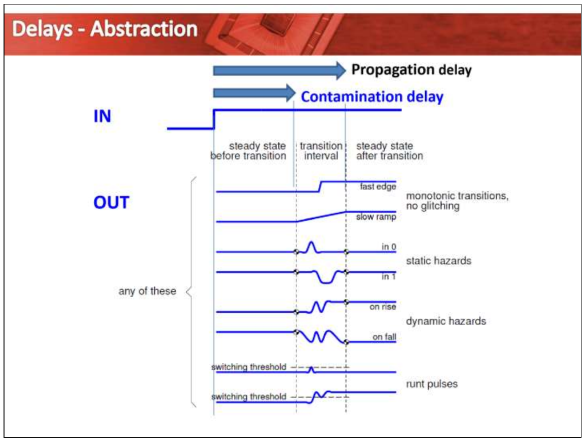
Design rule

Consistently dissociate signals into

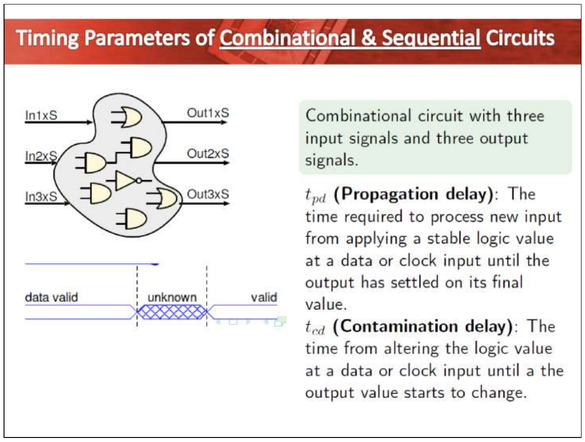
- asynchronous reset signals (when to enter the start state)
- clock signals (when to move from one state to the next)
- information signals (what state to enter, what output to produce)

Design rule

Make the clock period long enough so that all transient effects have died out before the next active clock edge instructs registers (and other storage devices) to accept new data!



Delays - Abstraction



Timing Parameters of Combinational & Sequential Circuits

Combinational circuit with three input signals and three output signals.

t_{pd} **(Propagation delay)**: The time required to process new input from applying a stable logic value at a data or clock input until the output has settled on its final value.

t_{cd} **(Contamination delay)**: The time from altering the logic value at a data or clock input until a the output value starts to change.

Timing Quantities – Comb. and Seq. Circuits

Combinational and sequential circuits

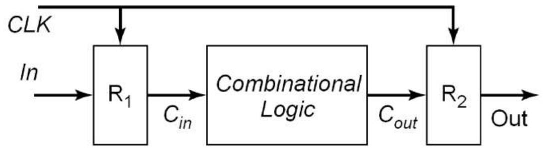
Propagation delay tpd

- New stable input (data or clock) – output settled on final value
- Example NAND: A-to-Z, and B-to-Z
- Example latch: D-to-Q, and/or CLK-to-Q

Contamination delay (or retain delay) tcd

- Altering input (data or clock) – first change of value at output
- By definition: 0≤tcd ≤tpd

Synchronous Timing



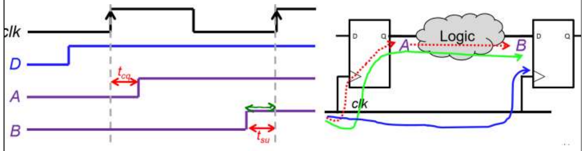
Synchronous timing:
All registers synchronized with same CLK

Timing Constraints (Setup & Hold)

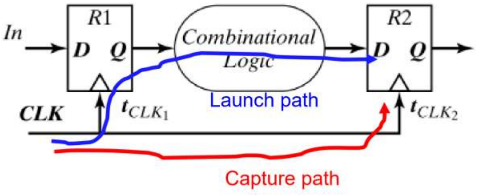
- There are two main problems that can possibly arise in synchronous logic:
  - » Max Delay violation: The data doesn't have enough time to pass from one register to the next before the next clock edge.
  - » Min Delay: The data path is so short that it causes a hold violation in capturing register.
- Max delay violations are a result of a slow (long) data paths, including the register's t_{su}, therefore they are often called "Setup violations".
- Min delay violations are a result of a fast (short) data path, causing the data to change before the t_{hold} of the reg has passed, therefore they are often called the "Hold violations".

Setup (Max) Constraint

- Let's see what makes up our clock cycle:
  - » After the clock rises, it takes t_{cq} for the data to propagate to point *A*.
  - » Then the data goes through the delay of the logic to get to point *B*.
  - » The data has to arrive at point *B* t_{su} before the next clock edge.
- In general, our timing path is a race:
  - » Between the Data Arrival, starting with the *launching* clock edge.
  - » And the Data Capture, one clock period later.



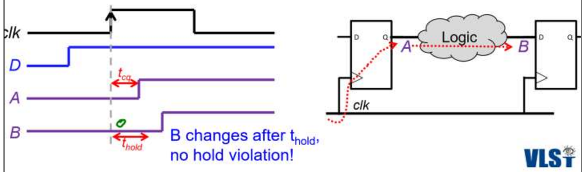
Setup (Max) Constraint



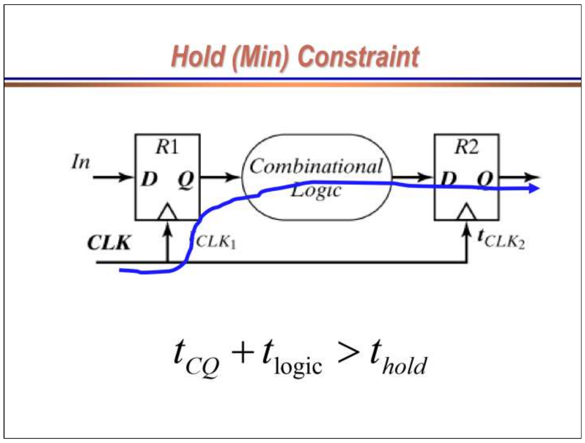
$$T > t_{CQ} + t_{\text{logic}} + t_{SU}$$

Hold (Min) Constraint

- Hold problems occur due to the logic changing before t_{hold} has passed.
- This is not a function of cycle time – it is relative to a single clock edge!
- Example of meeting the hold constraint:
  - » The clock rises and the data at A changes after t_{cq}.
  - » The data at B changes $t_{pd}(logic)$ later.
  - » Since the data at B had to stay stable for t_{hold} after the clock (for the second register), the change at B has to be at least t_{hold} **after** the clock edge.

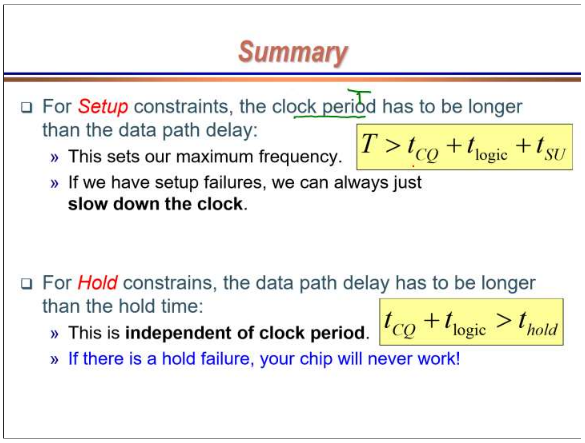


VLSI



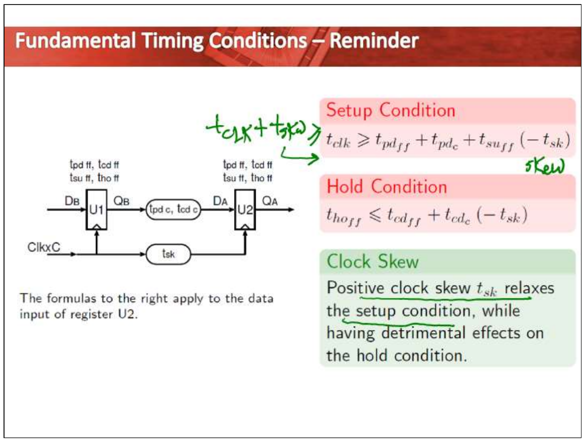
Hold (Min) Constraint

$$t_{CQ} + t_{\text{logic}} > t_{hold}$$



Summary

- For *Setup* constraints, the clock period has to be longer than the data path delay: T
  - » This sets our maximum frequency. $T > t_{CQ} + t_{\text{logic}} + t_{SU}$
  - » If we have setup failures, we can always just **slow down the clock**.
- For *Hold* constrains, the data path delay has to be longer than the hold time:
  - » This is **independent of clock period**. $t_{CQ} + t_{\text{logic}} > t_{hold}$
  - » If there is a hold failure, your chip will never work!



Fundamental Timing Conditions – Reminder

The formulas to the right apply to the data input of register U2.

Setup Condition

$t_{clk} + t_{skw} >$

$$t_{clk} \geqslant t_{pd_{ff}} + t_{pd_c} + t_{su_{ff}} \; (-t_{sk})$$

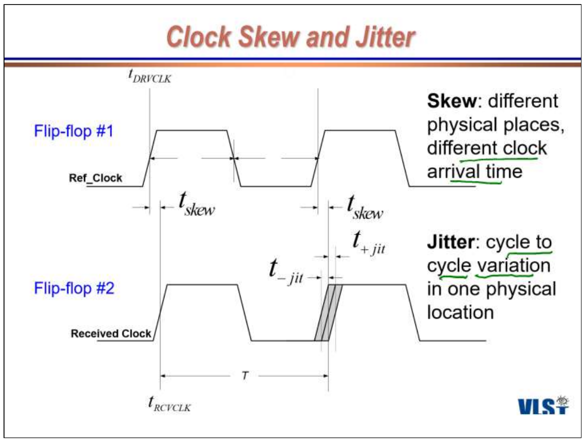
skew

Hold Condition

$$t_{ho_{ff}} \leqslant t_{cd_{ff}} + t_{cd_c} \; (-t_{sk})$$

Clock Skew

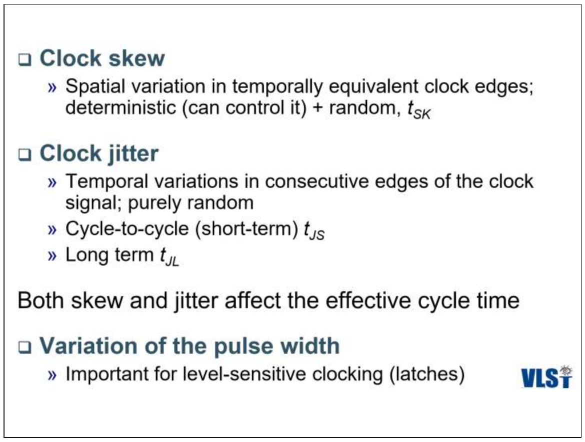
Positive clock skew t_{sk} relaxes the setup condition, while having detrimental effects on the hold condition.



Clock Skew and Jitter

Skew: different physical places, different clock arrival time

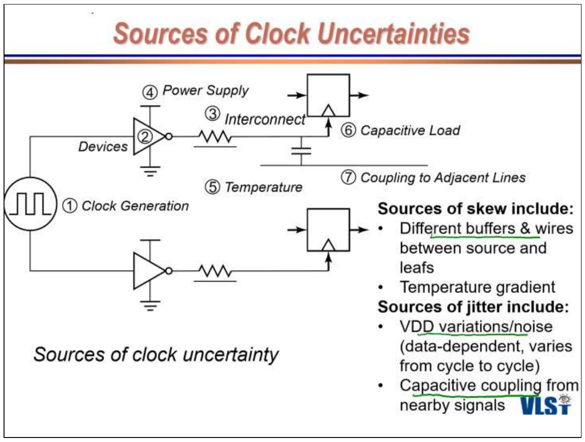
Jitter: cycle to cycle variation in one physical location



- **Clock skew**
  - » Spatial variation in temporally equivalent clock edges; deterministic (can control it) + random, t_{SK}
- **Clock jitter**
  - » Temporal variations in consecutive edges of the clock signal; purely random
  - » Cycle-to-cycle (short-term) t_{JS}
  - » Long term t_{JL}

Both skew and jitter affect the effective cycle time

- **Variation of the pulse width**
  - » Important for level-sensitive clocking (latches)

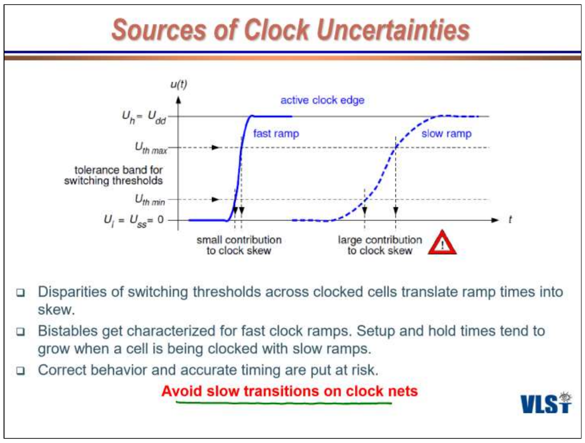


Sources of Clock Uncertainties

Sources of clock uncertainty

Sources of skew include:
- Different buffers & wires between source and leafs
- Temperature gradient

Sources of jitter include:
- VDD variations/noise (data-dependent, varies from cycle to cycle)
- Capacitive coupling from nearby signals

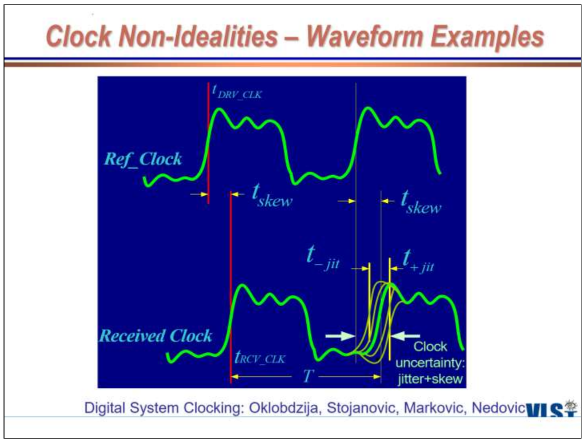


Sources of Clock Uncertainties

u(t)
active clock edge
$U_h = U_{dd}$
fast ramp
slow ramp
$U_{th\ max}$
tolerance band for switching thresholds
$U_{th\ min}$
$U_l = U_{ss} = 0$
t
small contribution to clock skew
large contribution to clock skew

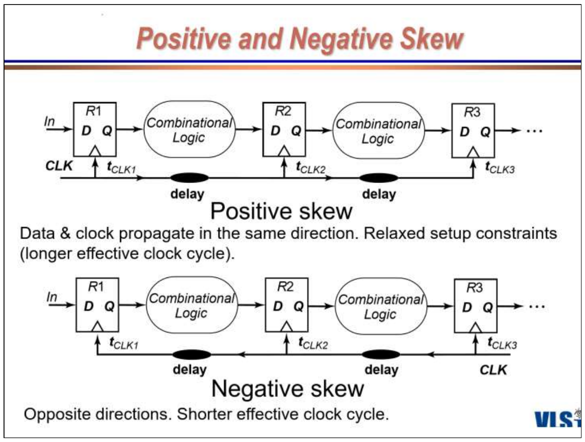
- □ Disparities of switching thresholds across clocked cells translate ramp times into skew.
- □ Bistables get characterized for fast clock ramps. Setup and hold times tend to grow when a cell is being clocked with slow ramps.
- □ Correct behavior and accurate timing are put at risk.

Avoid slow transitions on clock nets



Clock Non-Idealities – Waveform Examples

Digital System Clocking: Oklobdzija, Stojanovic, Markovic, Nedovic



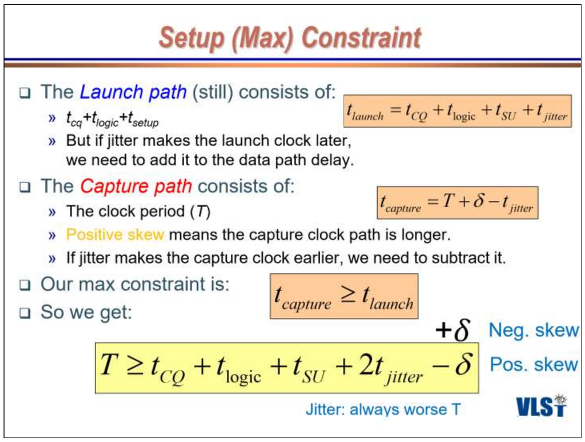
Positive and Negative Skew

Positive skew

Data & clock propagate in the same direction. Relaxed setup constraints (longer effective clock cycle).

Negative skew

Opposite directions. Shorter effective clock cycle.



Setup (Max) Constraint

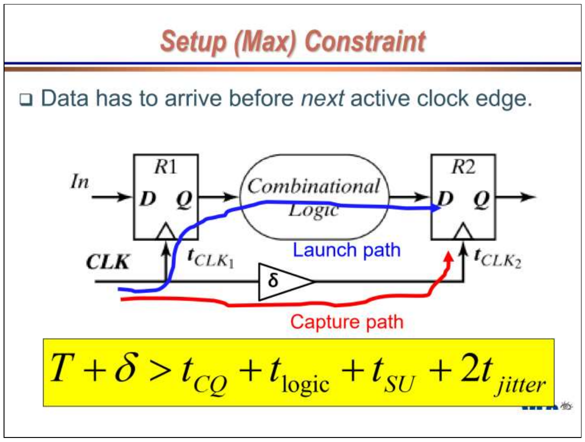
- □ The *Launch path* (still) consists of: $t_{launch} = t_{CQ} + t_{\text{logic}} + t_{SU} + t_{jitter}$
  - » t_{cq}+t_{logic}+t_{setup}
  - » But if jitter makes the launch clock later, we need to add it to the data path delay.
- □ The *Capture path* consists of: $t_{capture} = T + \delta - t_{jitter}$
  - » The clock period (*T*)
  - » Positive skew means the capture clock path is longer.
  - » If jitter makes the capture clock earlier, we need to subtract it.
- □ Our max constraint is: $t_{capture} \geq t_{launch}$
- □ So we get:

$+\delta$ Neg. skew

$$T \geq t_{CQ} + t_{\text{logic}} + t_{SU} + 2t_{jitter} - \delta$$

Pos. skew

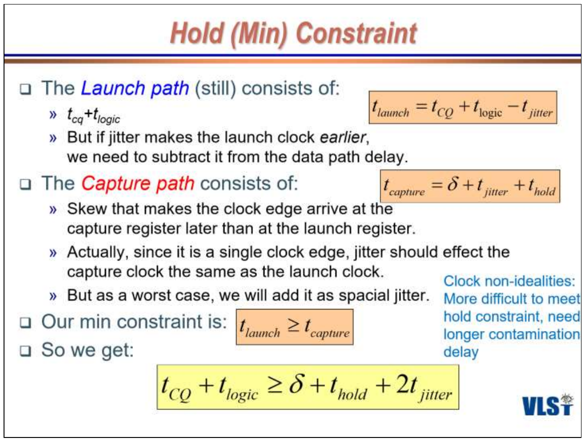
Jitter: always worse T



Setup (Max) Constraint

- □ Data has to arrive before *next* active clock edge.

$$T + \delta > t_{CQ} + t_{\text{logic}} + t_{SU} + 2t_{jitter}$$

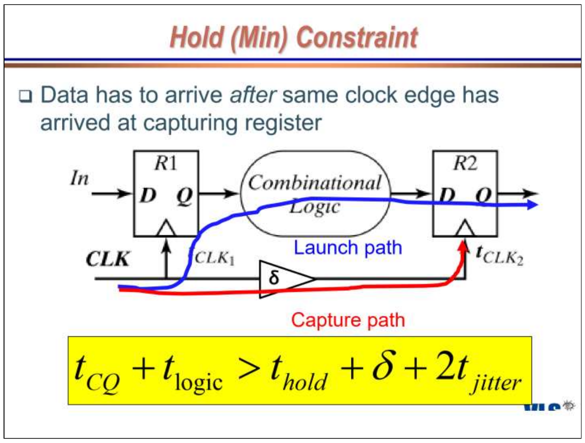


Hold (Min) Constraint

- □ The *Launch path* (still) consists of: $t_{launch} = t_{CQ} + t_{\text{logic}} - t_{jitter}$
  - » t_{cq}+t_{logic}
  - » But if jitter makes the launch clock *earlier*, we need to subtract it from the data path delay.
- □ The *Capture path* consists of: $t_{capture} = \delta + t_{jitter} + t_{hold}$
  - » Skew that makes the clock edge arrive at the capture register later than at the launch register.
  - » Actually, since it is a single clock edge, jitter should effect the capture clock the same as the launch clock.
  - » But as a worst case, we will add it as spacial jitter.
- □ Our min constraint is: $t_{launch} \geq t_{capture}$
- □ So we get:

Clock non-idealities: More difficult to meet hold constraint, need longer contamination delay

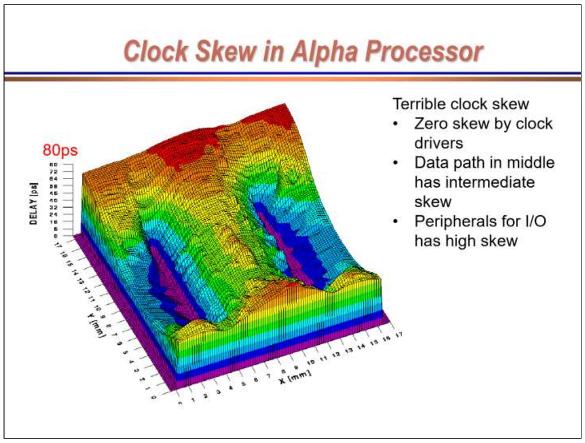
$$t_{CQ} + t_{logic} \geq \delta + t_{hold} + 2t_{jitter}$$



Hold (Min) Constraint

- □ Data has to arrive *after* same clock edge has arrived at capturing register

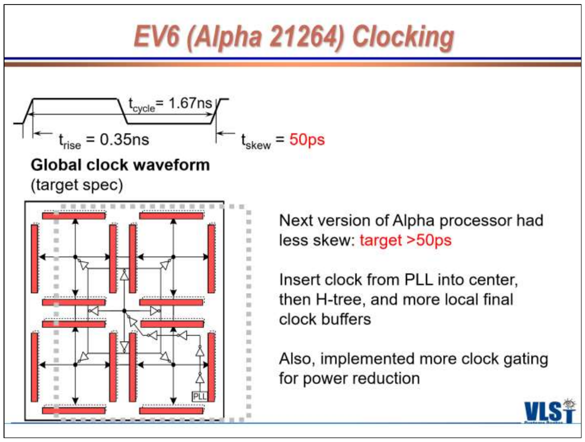
$$t_{CQ}+t_{\text{logic}} > t_{hold}+\delta+2t_{jitter}$$



Clock Skew in Alpha Processor

Terrible clock skew

- Zero skew by clock drivers
- Data path in middle has intermediate skew
- Peripherals for I/O has high skew



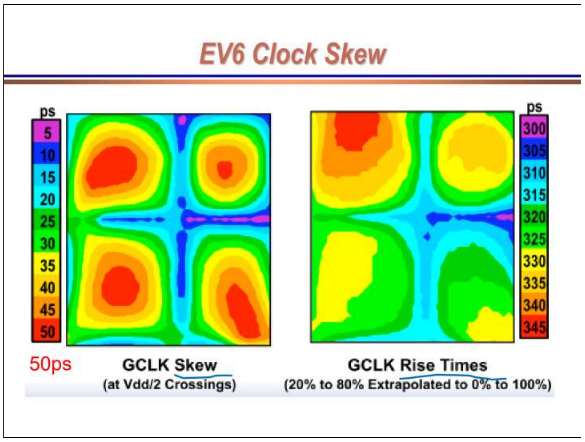
EV6 (Alpha 21264) Clocking

Global clock waveform
(target spec)

Next version of Alpha processor had less skew: target >50ps

Insert clock from PLL into center, then H-tree, and more local final clock buffers

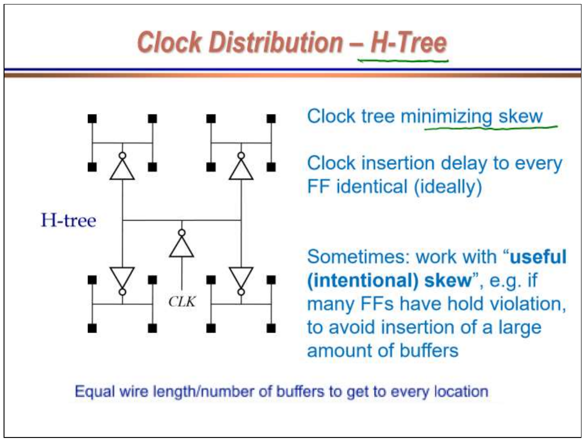
Also, implemented more clock gating for power reduction



EV6 Clock Skew

GCLK Skew
(at Vdd/2 Crossings)

GCLK Rise Times
(20% to 80% Extrapolated to 0% to 100%)



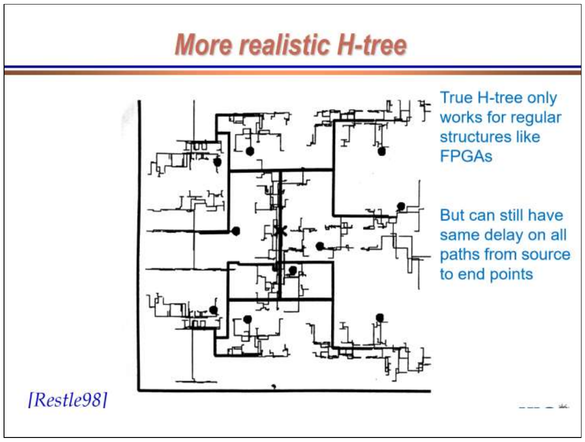
Clock Distribution – H-Tree

Clock tree minimizing skew

Clock insertion delay to every FF identical (ideally)

Sometimes: work with "**useful (intentional) skew**", e.g. if many FFs have hold violation, to avoid insertion of a large amount of buffers

Equal wire length/number of buffers to get to every location

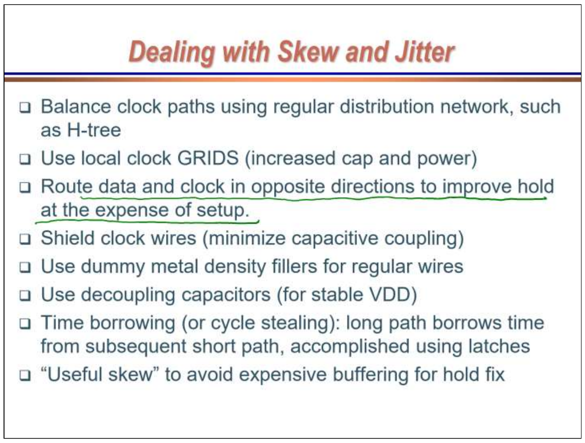


More realistic H-tree

True H-tree only works for regular structures like FPGAs

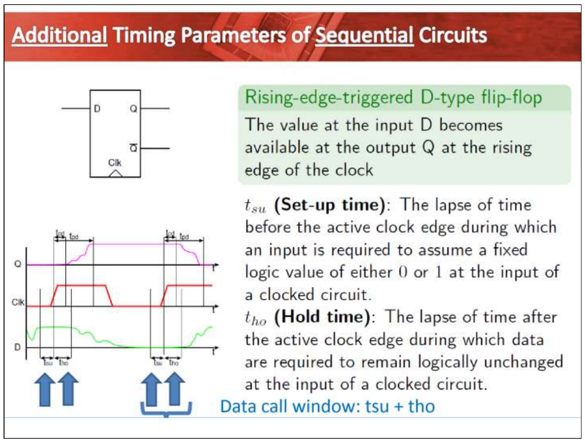
But can still have same delay on all paths from source to end points

[Restle98]



Dealing with Skew and Jitter

- ❑ Balance clock paths using regular distribution network, such as H-tree
- ❑ Use local clock GRIDS (increased cap and power)
- ❑ Route data and clock in opposite directions to improve hold at the expense of setup.
- ❑ Shield clock wires (minimize capacitive coupling)
- ❑ Use dummy metal density fillers for regular wires
- ❑ Use decoupling capacitors (for stable VDD)
- ❑ Time borrowing (or cycle stealing): long path borrows time from subsequent short path, accomplished using latches
- ❑ "Useful skew" to avoid expensive buffering for hold fix



Additional Timing Parameters of Sequential Circuits

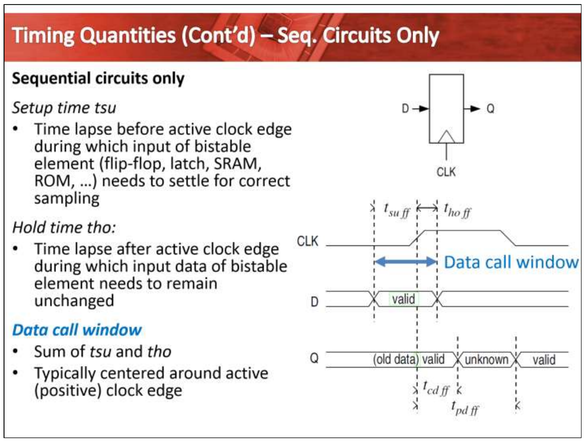
Rising-edge-triggered D-type flip-flop

The value at the input D becomes available at the output Q at the rising edge of the clock

t_{su} **(Set-up time)**: The lapse of time before the active clock edge during which an input is required to assume a fixed logic value of either 0 or 1 at the input of a clocked circuit.

t_{ho} **(Hold time)**: The lapse of time after the active clock edge during which data are required to remain logically unchanged at the input of a clocked circuit.

Data call window: tsu + tho



Timing Quantities (Cont'd) – Seq. Circuits Only

Sequential circuits only

Setup time tsu

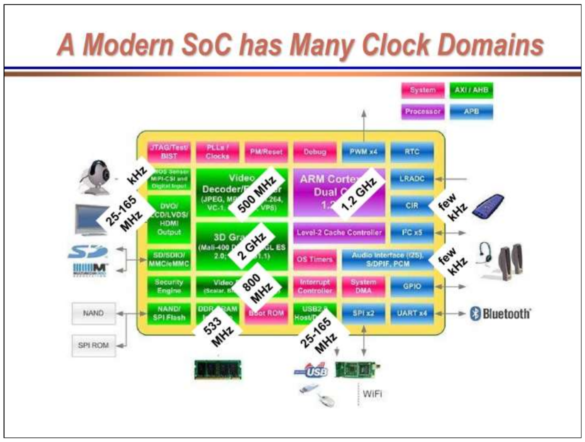
- Time lapse before active clock edge during which input of bistable element (flip-flop, latch, SRAM, ROM, …) needs to settle for correct sampling

Hold time tho:

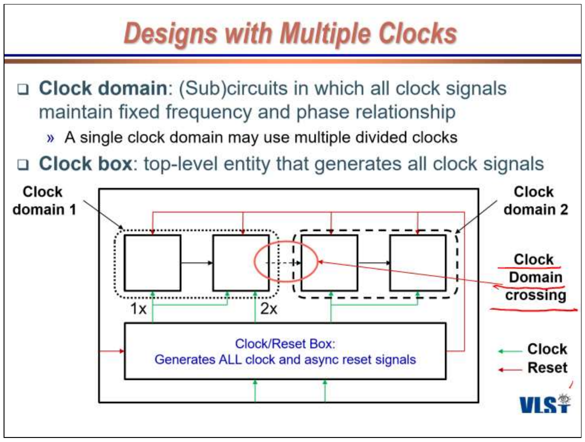
- Time lapse after active clock edge during which input data of bistable element needs to remain unchanged

Data call window

- Sum of *tsu* and *tho*
- Typically centered around active (positive) clock edge

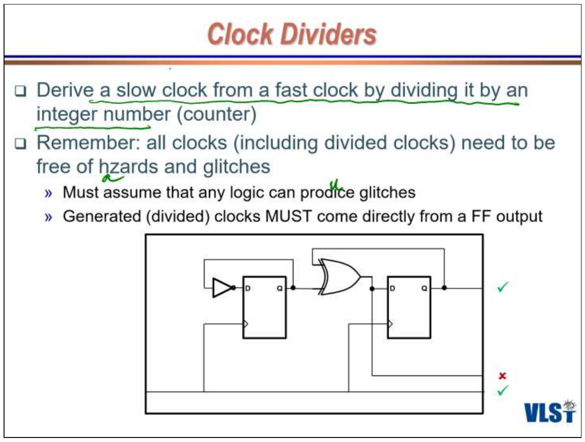


A Modern SoC has Many Clock Domains



Designs with Multiple Clocks

- ❑ **Clock domain**: (Sub)circuits in which all clock signals maintain fixed frequency and phase relationship
  - » A single clock domain may use multiple divided clocks
- ❑ **Clock box**: top-level entity that generates all clock signals

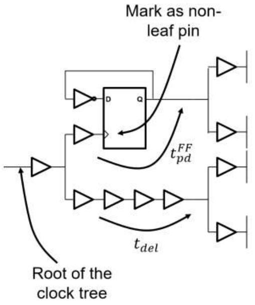


Clock Dividers

- ❑ Derive a slow clock from a fast clock by dividing it by an integer number (counter)
- ❑ Remember: all clocks (including divided clocks) need to be free of hzards and glitches
  - » Must assume that any logic can prodice glitches
  - » Generated (divided) clocks MUST come directly from a FF output

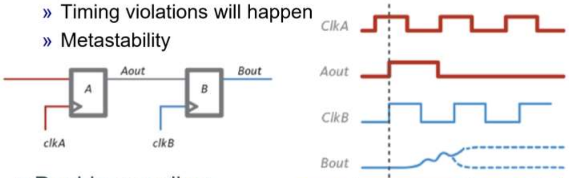
Clock Dividers in the Clock Tree

- ❑ Clock and derived clock must be phase aligned at the clock roots
- ❑ Clock divider has a delay (FF propagation delay)
  - » Need to consider clock divider during clock-tree generation
- ❑ Modern clock tree synthesis tools can trace through clock dividers

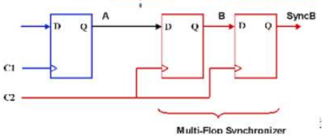


Crossing Clock Domains

- ❑ Arbitrary phase relationship
  - » Timing violations will happen
  - » Metastability

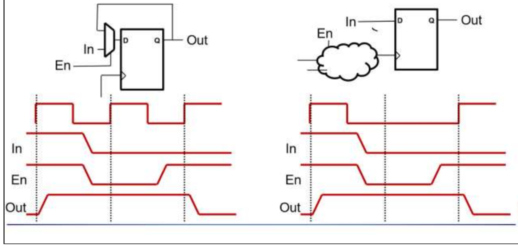


- ❑ Double sampling:
  - » Provides an entire cycle for to resolve metastability



Clock Gating

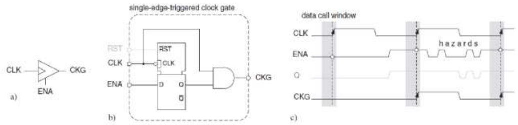
- ❑ Flip-Flops with ENABLE: prevent FlipFlop(s) from capturing new data
  - » Functional: preserve content over multiple cycles
  - » Power savings: avoid activity in a block that is inactive



Clock Gating

- ❑ Advantages:
  - » Saves a multiplexer per FlipFlop with Enable (Area, Delay, and Power advantage)
  - » Avoids activity on the clock net leading to the FlipFlop
  - » Avoids activity on the FlipFlops clock pin reducing internal power consumption
- ❑ Disadvantages:
  - » Need for additional logic to suppress the clock while FlipFlop is disabled (area and power penalty)
  - » Need to ensure that the clock signal is free of glitches

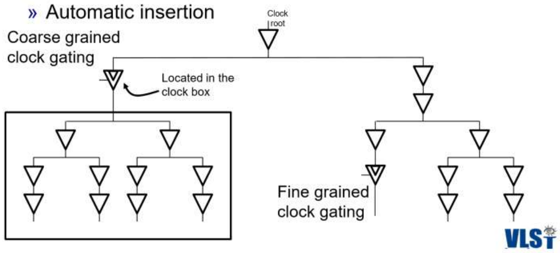
- ❑ A safe strategy to realize clock gating

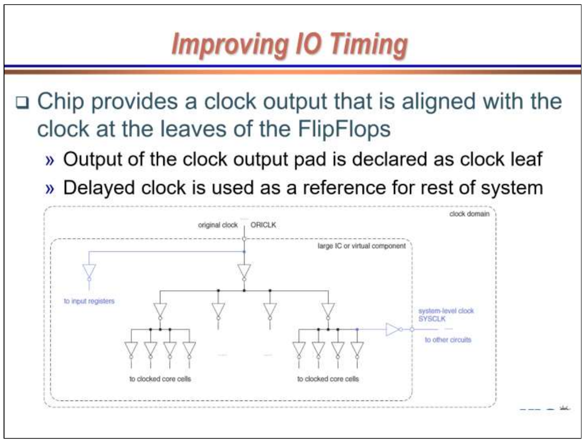


  - » Latch on the Enable signal shields glitches during the sensitive period of the clock
  - » Implemented with individual cells: some timing constraints need to be observed
  - » Often realized as single dedicated library cell

Clock Gating

- ❑ Corase grained: disable entire blocks of the design
  - » Inserted manually
- ❑ Fine grained: enable for small groups of FlipFlops
  - » Automatic insertion

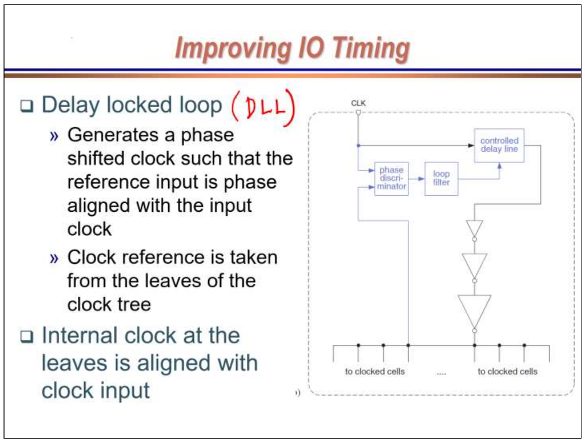




Improving IO Timing

- ❑ Chip provides a clock output that is aligned with the clock at the leaves of the FlipFlops
  - » Output of the clock output pad is declared as clock leaf
  - » Delayed clock is used as a reference for rest of system

clock domain
original clock
ORICLK
large IC or virtual component
to input registers
system-level clock
SYSCLK
to other circuits
to clocked core cells
to clocked core cells



Improving IO Timing

- ❑ Delay locked loop (DLL)
  - » Generates a phase shifted clock such that the reference input is phase aligned with the input clock
  - » Clock reference is taken from the leaves of the clock tree
- ❑ Internal clock at the leaves is aligned with clock input

CLK
controlled delay line
phase discri-minator
loop filter
to clocked cells
....
to clocked cells