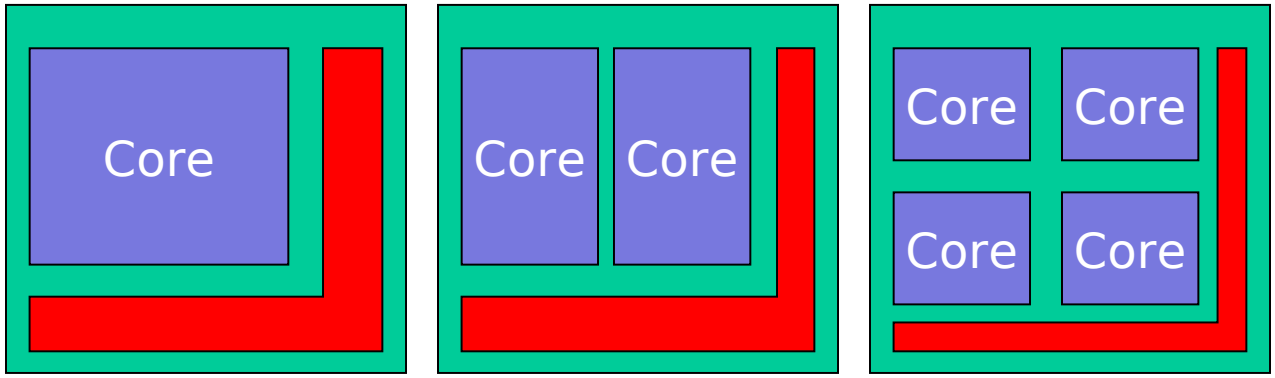


	Single Core	Dual Core	Quad Core
Core area	A	~A/2	~A/4
Core power	W	~W/2	~W/4
Core performance	P	0.9P	0.8P
Chip performance	P	1.8P	3.2P

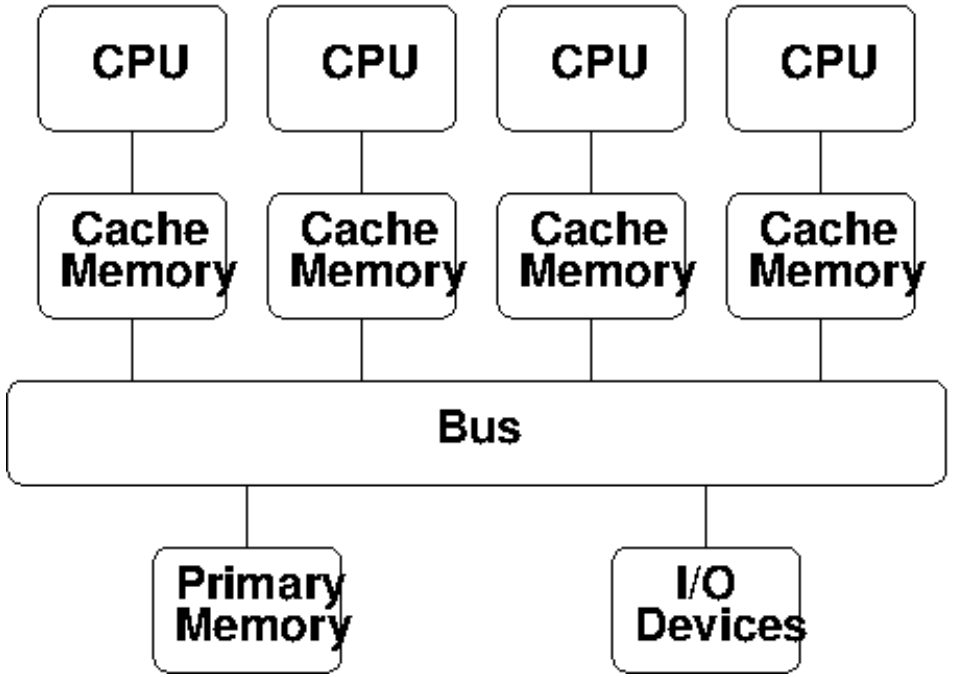
Lecture 27-28 Advanced VLSI Architecture: Parallelism

Technique for Improving Performance

- Exploiting parallelism in improving performance
- Two ways
  - **Pipeline**
    - Using pipeline latches to reduce the critical path delay
    - Can exploit to increase the clock speed of sample speed or reduce power consumption at the same speed
  - **Parallelism**
    - Multiple output are computed in parallel in a clock period with parallel hardware
    - Effective sampling speed is increased with the level of parallelism
    - Can be used for the reduction of power consumption

Parallel Processing

- Parallel processing and pipelining techniques are duals of each other
  - Both exploit concurrency available in the computation
- Parallel processing (also referred to as block processing) – computed using duplicate hardware



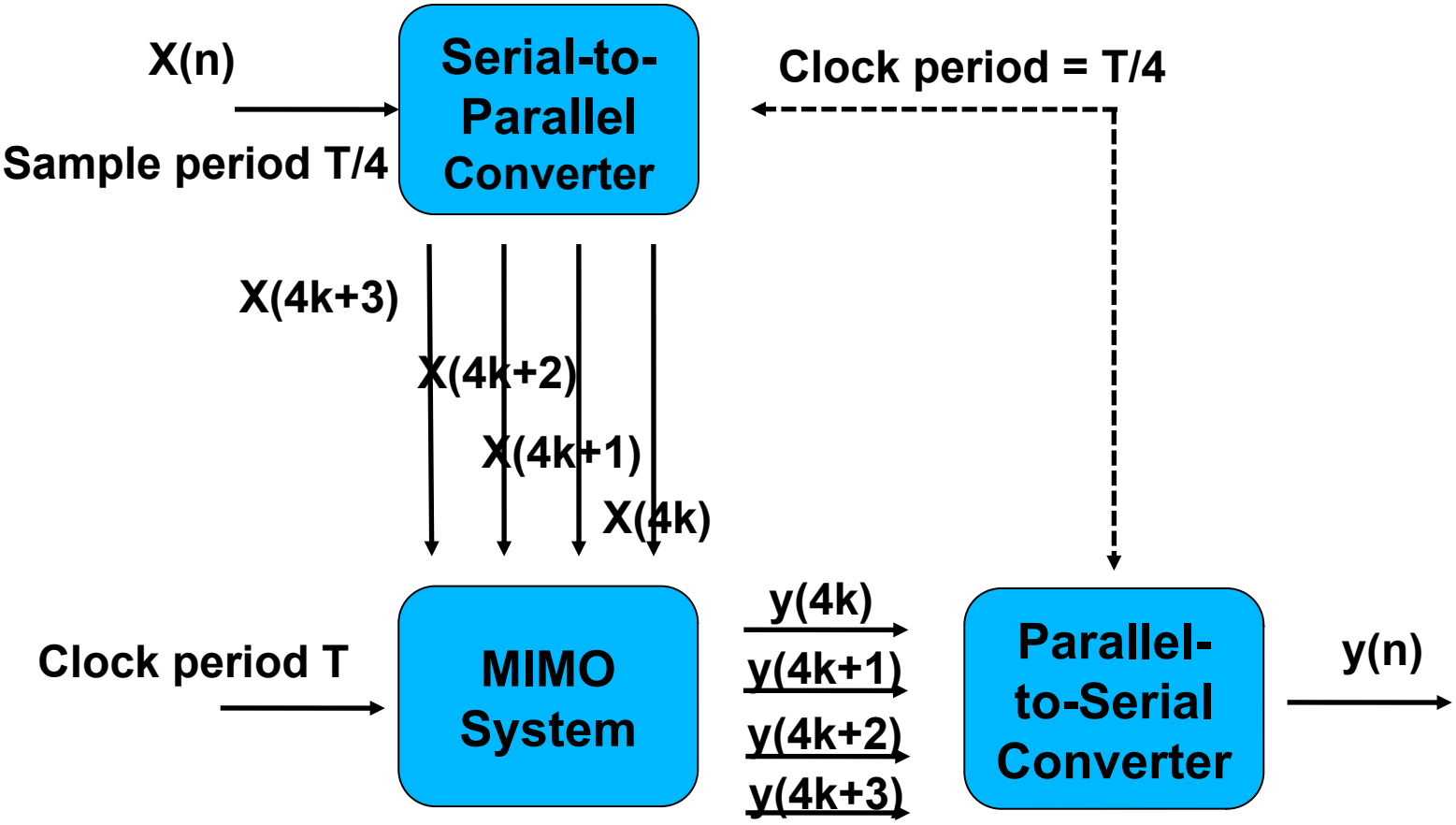
Modern Processors

- Pentium 4 has a 30-stage pipeline. If the pipeline gets too large, there is too much overhead.

- However, new processors like the CELL processor in the playstation 3 are moving to multi-core architectures.
  - The pipeline is much smaller, between 5 and 10
  - Multicore processors work best for applications that run a lot of threads applications that are easily separable.

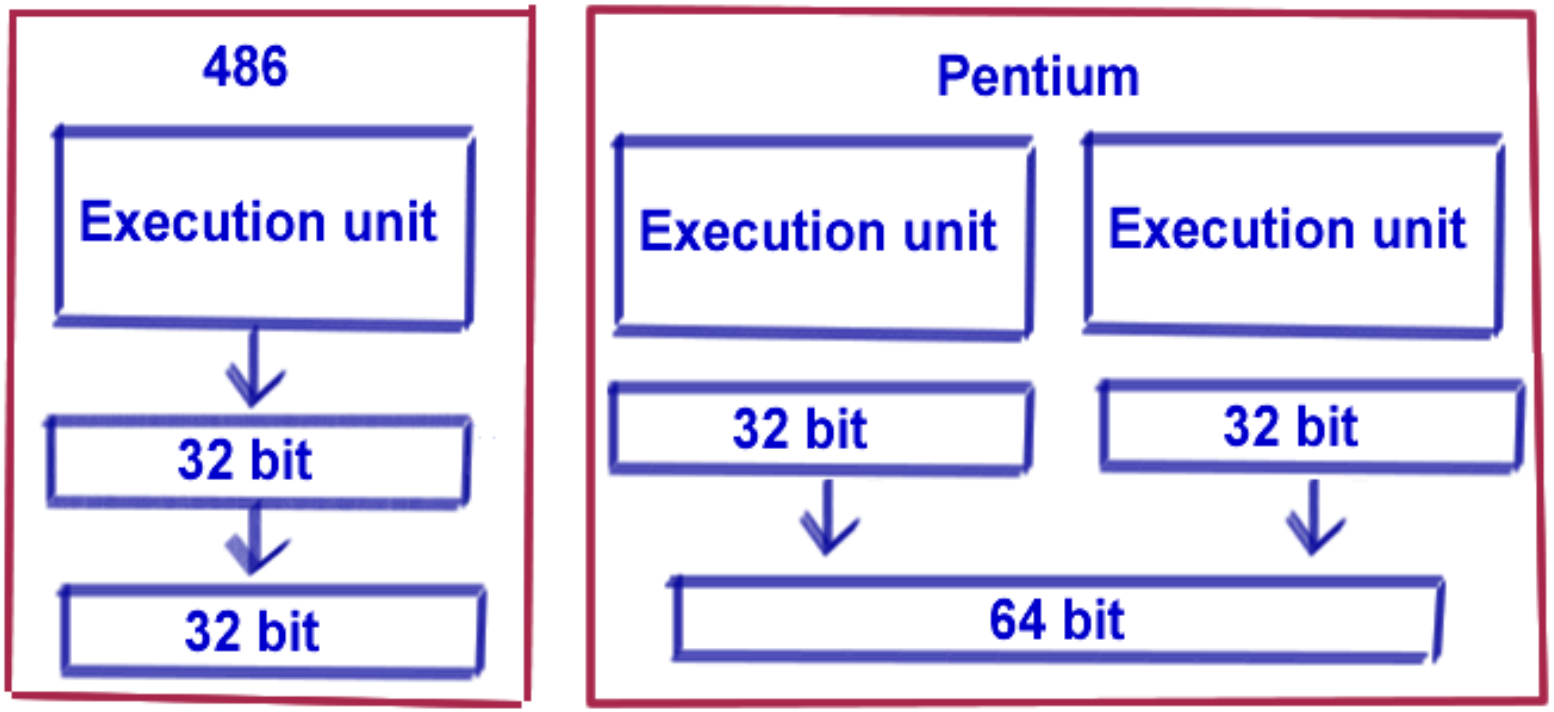
Parallel Processing

- Complete parallel processing system with block size 4



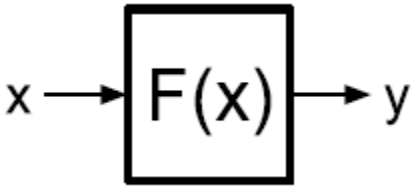
Superscalar Computer

- A superscalar computer executes more than one instruction per clock cycle. This is achieved by having more independent pipelines to proceed in parallel.



Parallel Processing Example

- Consider a single-input single-output FIR filter
  - $F(x) = y_i = a \cdot x_i \cdot x_i + b \cdot x_i + c$



- For example, to get a parallel system with 3 inputs per clock cycle (i.e., level of parallel processing =3)
  - $y_i = a \cdot x_i \cdot x_i + b \cdot x_i + c$
  - $y_m = a \cdot x_m \cdot x_m + b \cdot x_m + c$
  - $y_n = a \cdot x_n \cdot x_n + b \cdot x_n + c$

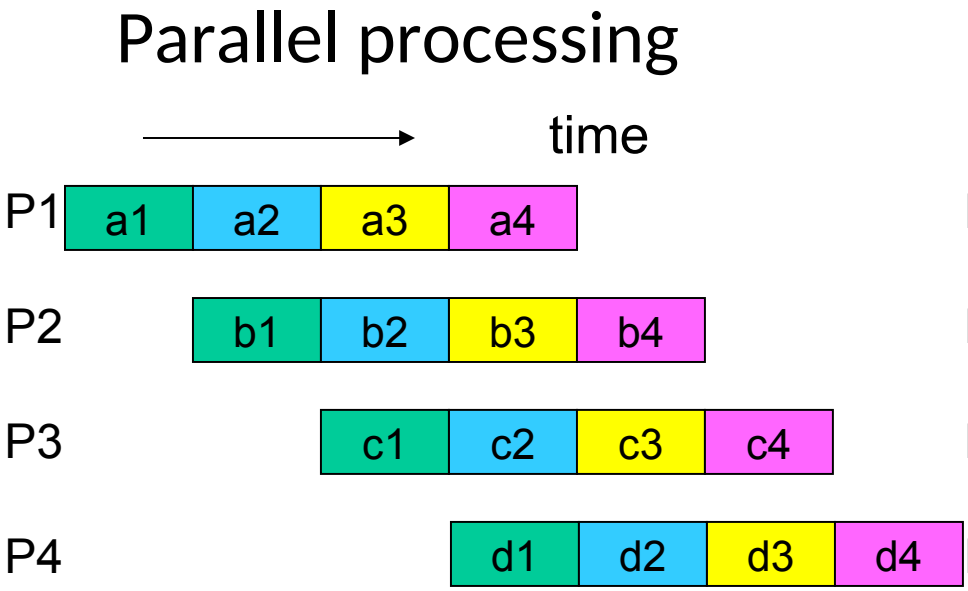
Critical path of parallel processing systems remains unchanged

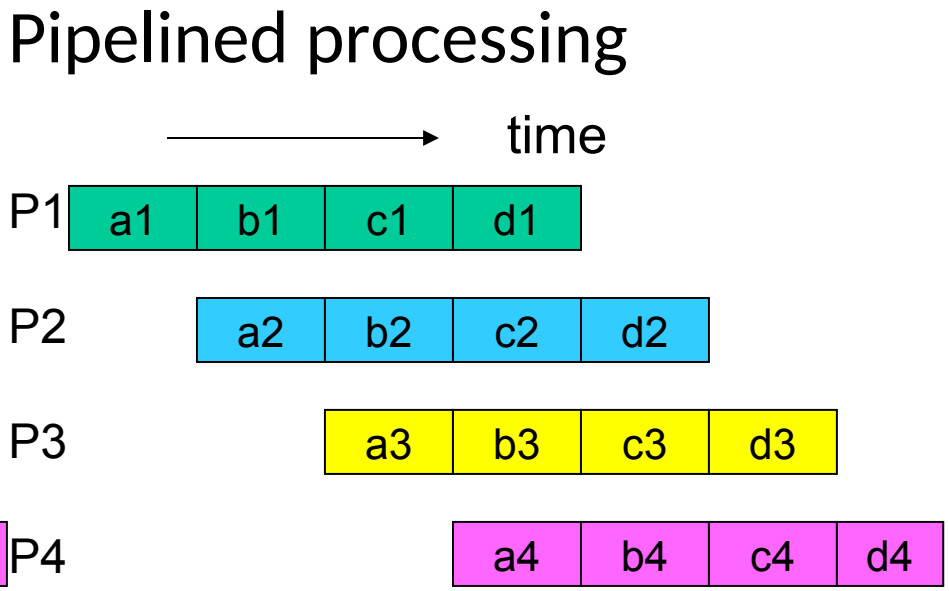
Pipeline Processing for Low Power

- Two main advantages of using pipelining and parallel processing:
  - Higher speed
  - Lower power consumption

- When sample speed does not need to be increased, these techniques can be used for lowering the power consumption

Parallelism: Basic Ideas



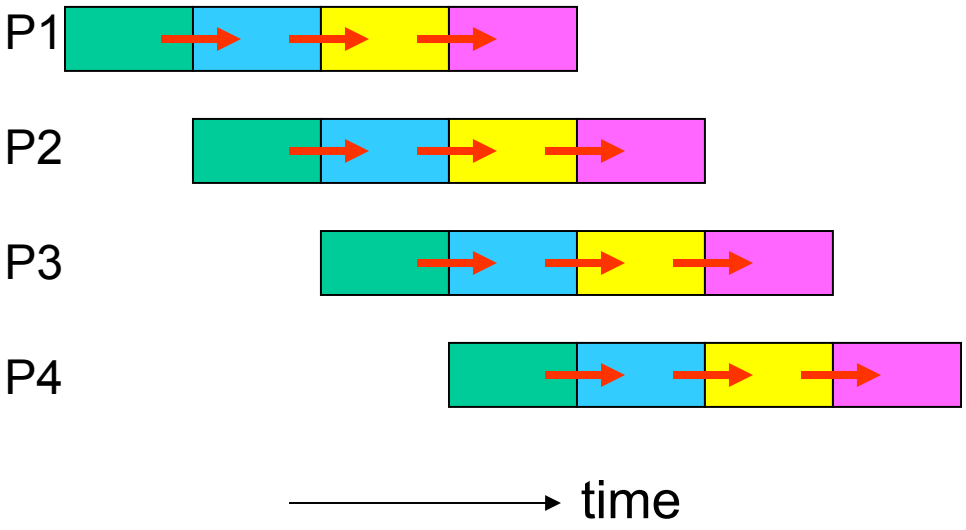


Colors: different types of operations performed
a, b, c, d: different data streams processed

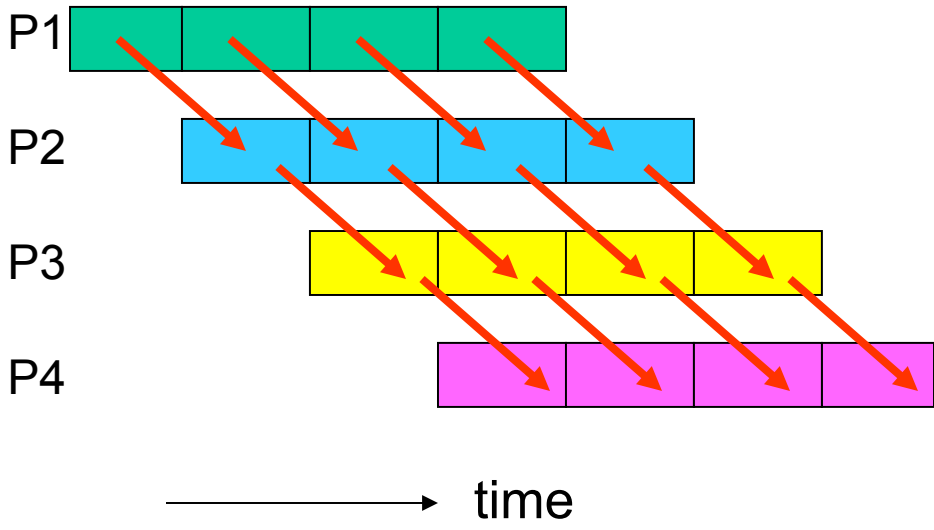
- Parallelism
  - The average throughput is greatly increased
  - Very little time is wasted
  - A lot of things are naturally parallel

Data Dependence

Parallel processing requires NO data dependence between processors



Pipelined processing will involve inter-processor communication



Parallelism

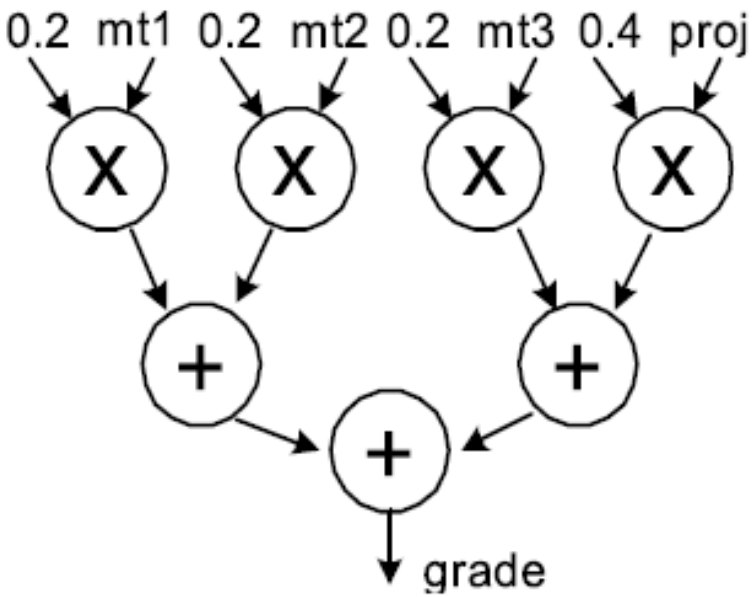
- Optimization in hardware design often involves using parallelism to trade between cost and performance.

- Example, student final grade calculation

  Read mt1, mt2, mt3, project;

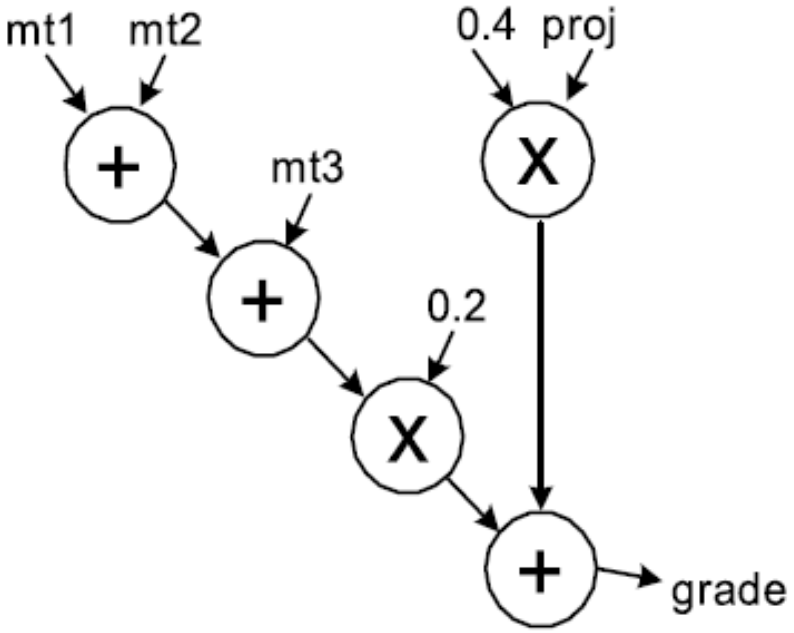
  Grade = 0.2 X mt1 + 0.2 X mt2 + 0.2 X mt3 + 0.4 X project;

  Write grade;



Parallelism

- Is there a lower cost hardware implementation?
- Can factor out multiply by 0.2?

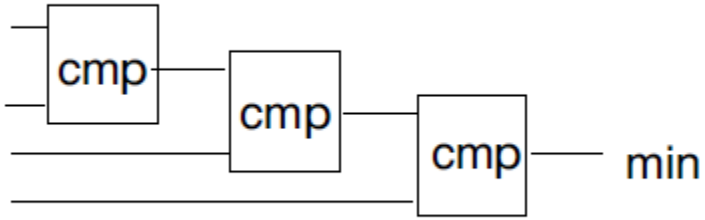


- How about sharing operators (multipliers and adders)?

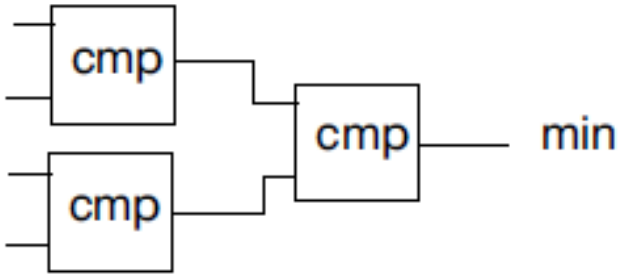
Performing Operations in Parallel

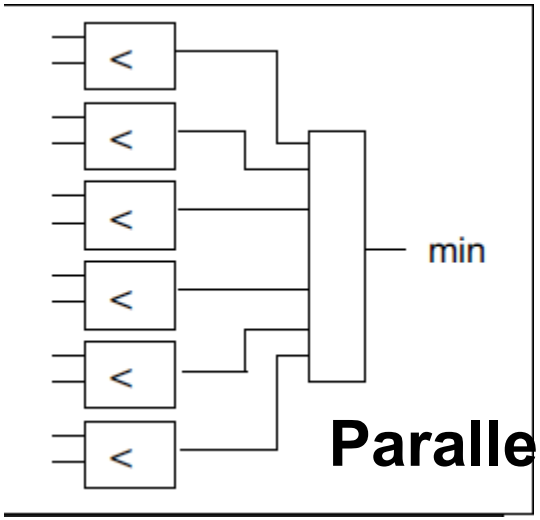
- Various search strategies are used to find the smallest element: linear search, binary search, and parallel search.

Linear search



Binary search





Parallel search

Linear Search	22.41ns	527 gates
Binary Search	14.9ns	512 gates
Parallel Search	11.4ns	612 gates

```
// Linear Search - 22.41 ns, 527 gates
module arrayCmpV1(clk, reset, inc, index, min);
input clk, reset, inc; input [1:0] index; output [1:0] min;
reg [5:0] cntr[0:4]; reg [1:0] min; // pseudo register
integer i; // compare each array element to mincount
task sel;
output [1:0] sel;
reg [5:0] mincount;
begin : _sc
mincount = cntr[0];
sel = 2'd0;
for ( i = 1; i <= 3; i=i+1 )
if ( cntr[i] < mincount ) begin
mincount = cntr[i];
sel = i;
end end
endtask
```

```
always @(cntr[0] or cntr[1] or cntr[2] or cntr[3])
sel(min);
always @(posedge clk)
if (reset)
for( i=0; i<=3; i=i+1 )
cntr[i] <= 6'd0;
else if (inc)
cntr[index] <= cntr[index] + 1'b1;
endmodule
```

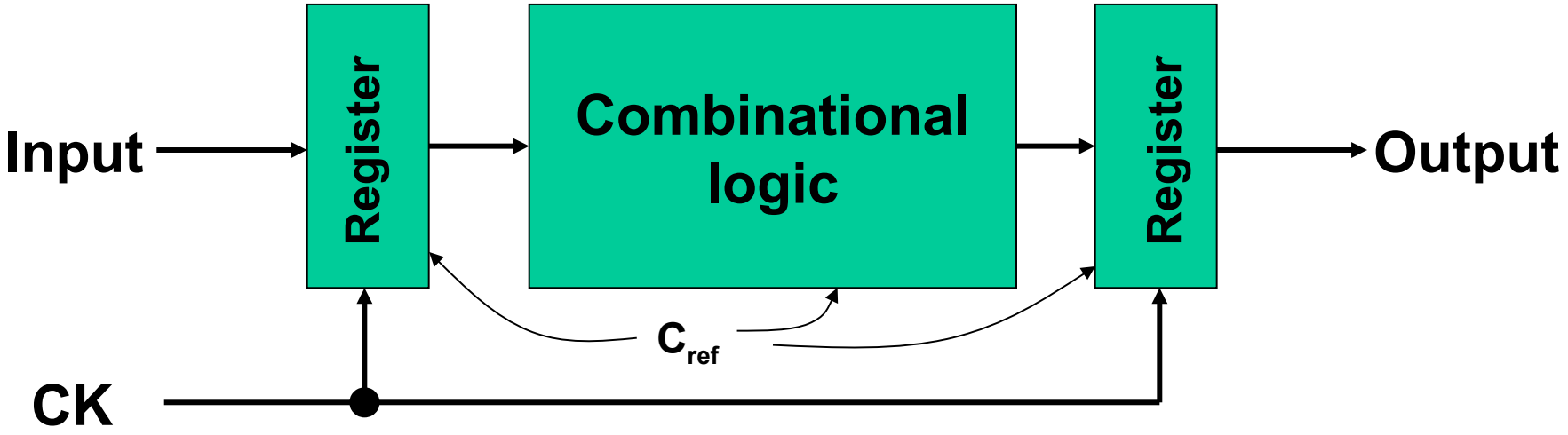
// **Binary Search - 14.9 ns, 512 gates (smallest area)**

```
module arrayCmpV2(clk, reset, inc, index, min);
input clk, reset, inc; input [1:0] index;
output [1:0] min;
reg [5:0] cntr[0:4];
integer i;
// binary tree comparison
wire c3lt2 = cntr[3] < cntr[2];
wire c1lt0 = cntr[1] < cntr[0];
wire [5:0] cntr32 = c3lt2 ? cntr[3] : cntr[2];
wire [5:0] cntr10 = c1lt0 ? cntr[1] : cntr[0];
wire c32lt10 = cntr32 < cntr10;
// select the smallest value
assign min = {c32lt10, c32lt10 ? c3lt2: c1lt0};
always @(posedge clk)
if (reset)
for( i=0; i<=3; i=i+1 )
cntr[i] <= 6'd0;
else if (inc)
cntr[index] <= cntr[index] + 1;
endmodule
```

```
// Parallel Search - 11.4 ns, 612 gates (fastest design)
module arrayCmpV3(clk, reset, inc, index, min);
input clk, reset, inc; input [1:0] index; output [1:0] min;
reg [5:0] cntr[0:4];
integer i;
// compare all counters to each other
wire l32 = cntr[3] < cntr[2];
wire l31 = cntr[3] < cntr[1];
wire l30 = cntr[3] < cntr[0];
wire l21 = cntr[2] < cntr[1];
wire l20 = cntr[2] < cntr[0];
wire l10 = cntr[1] < cntr[0];
// select the smallest value
assign min = {l31&l30 | l21&l20, l32&l30 | l10&~l21};
always @(posedge clk)
if (reset)
for( i=0; i<=3; i=i+1 )
cntr[i] <= 6'd0;
else if (inc)
cntr[index] <= cntr[index] + 1;
endmodule
```

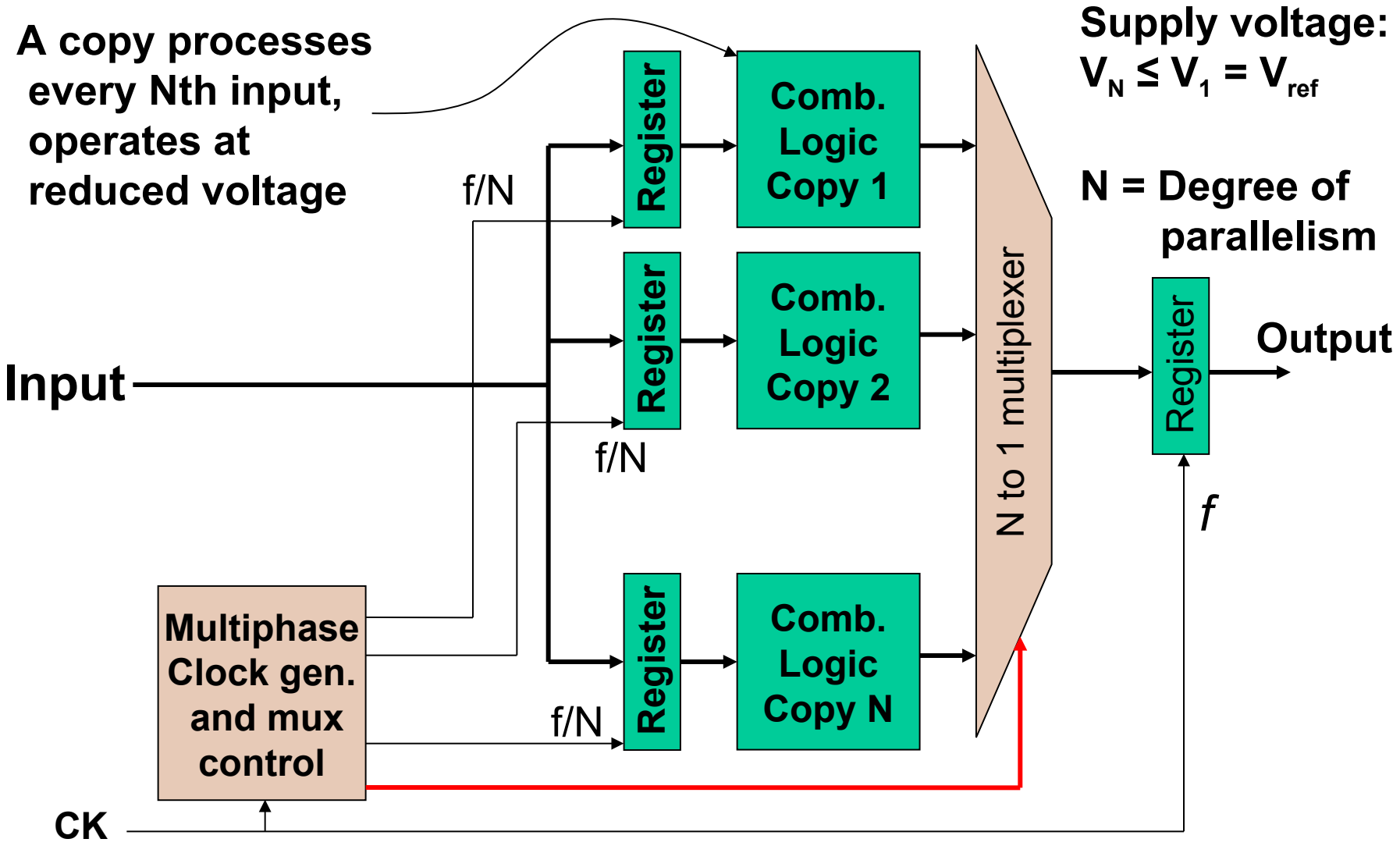
Low-Power Data Path Architecture

- Low supply voltage
  - This slows down circuit speed
  - Use parallel computing to gain the speed back



Supply voltage = V_{ref}
Total capacitance switched per cycle = C_{ref}
Clock frequency = f
Power consumption: $P_{ref} = C_{ref}V_{ref}^2f$

A Parallel Architecture



Design Tradeoffs

Number of cores N	Clock (MHz)	Core supply VDDN (Volts)	Total Power (Watts)
1	200	5.00	15.0
2	100	3.68	8.94
4	50	2.75	5.90
5	40	2.51	5.29
8	25	2.10	4.50

Voltage Scaling for Reduced Energy

- Reducing supply voltage by 0.5 improves energy per transition by 0.25
- Performance is reduced – need to use slower clock
- Can regain performance through parallel architecture

- Alternatively, can trade surplus performance for lower energy by reducing supply voltage until "just enough" performance

Parallel Architectures for Reduced Energy at Constant Throughput

[a] **8-bit adder/comparator**:

40MHz at 5V, area = 530 μm², Base power Pref

[b] **Two parallel interleaved adder/compare units:**

20MHz at 2.9V, area = 1,800 μm² (3.4×)

Power = 0.36 Pref

[c] **One pipelined adder/compare unit:**

40MHz at 2.9V, area = 690 μm² (1.3×)

Power = 0.39 Pref

[d] **Pipelined and parallel:**

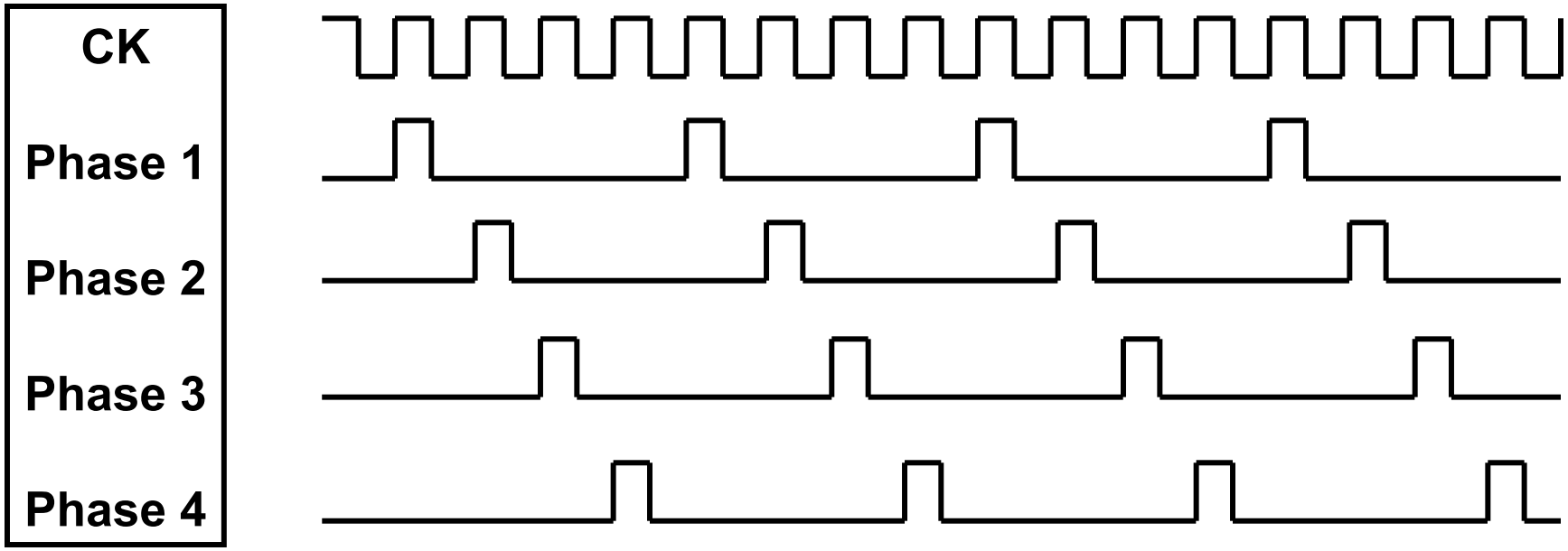
20MHz at 2.0V, area = 1,961 μm² (3.7×)

Power = 0.2 Pref

System Operating Modes

- **Fixed throughput**
  - e.g., MP3 player
  - want to minimize energy at fixed throughput (equivalent to minimizing power)
- **Maximum throughput**
  - e.g., spreadsheet update
  - want to run "*as fast as possible*"
- **How do we trade performance and energy/operation?**
  - energy-delay product gives equal weighting

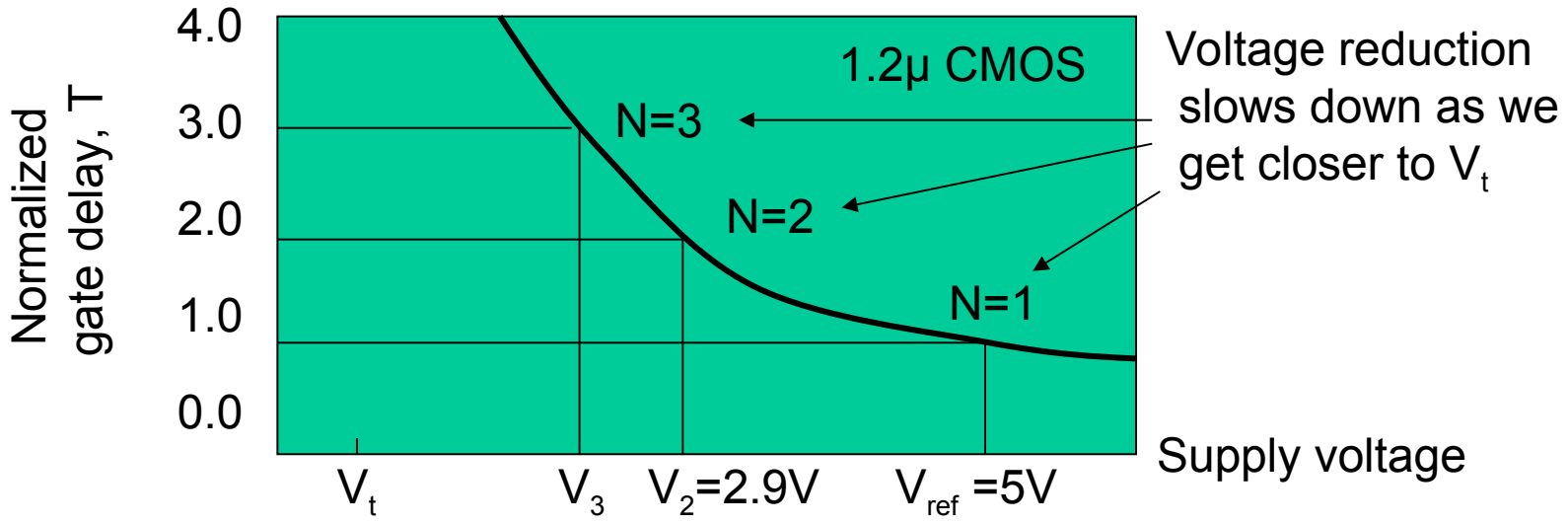
Control Signals, N =4



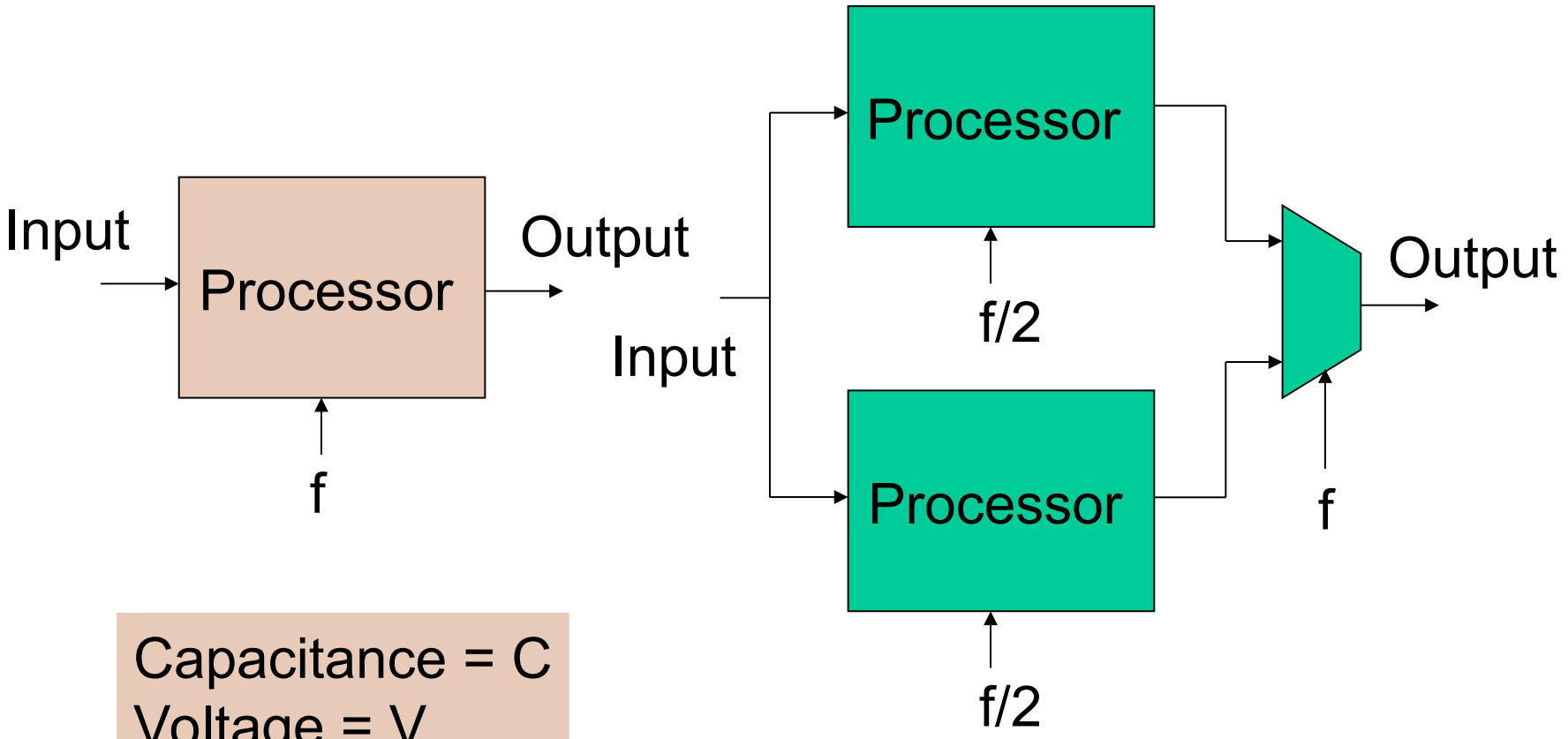
Voltage vs. Speed

$$\text{Delay of a gate, } T \approx \frac{C_L V_{ref}}{I} = \frac{C_L V_{ref}}{k(W/L)(V_{ref} - V_t)^2}$$

where I is saturation current
k is a technology parameter
W/L is width to length ratio of transistor
V_t is threshold voltage



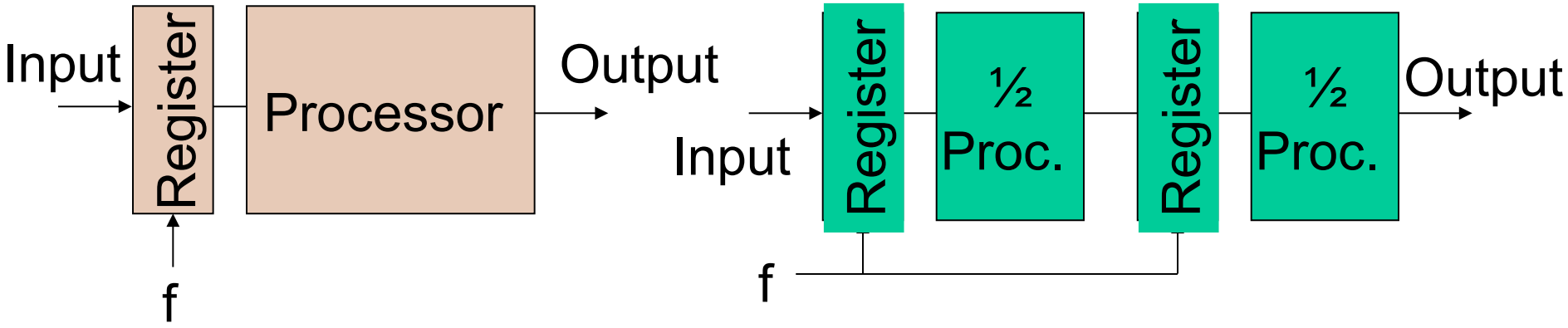
Parallel Architecture



Capacitance = C
Voltage = V
Frequency = f
Power = CV^2f

Capacitance = 2.2C
Voltage = 0.6V
Frequency = 0.5f
Power = $0.396CV^2f$

Pipeline Architecture



Capacitance = C
Voltage = V
Frequency = f
Power = CV^2f

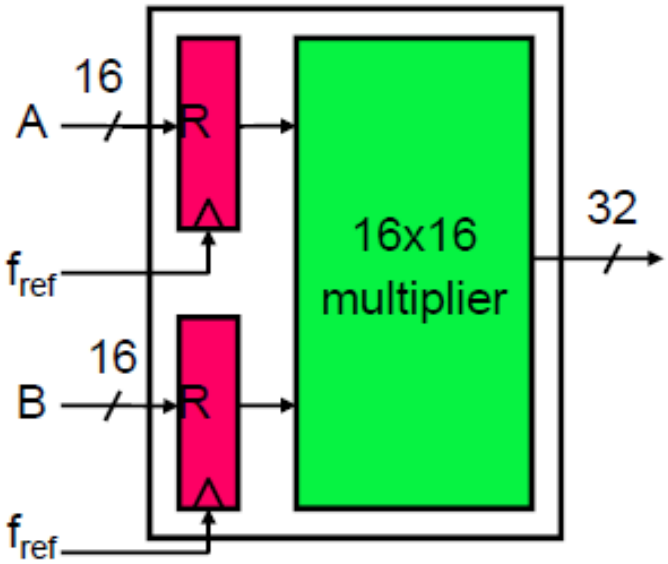
Capacitance = 1.2C
Voltage = 0.6V
Frequency = f
Power = $0.432CV^2f$

Approximate Trend

	n-parallel proc.	n-stage pipeline proc.
Capacitance	nC	C
Voltage	V/n	V/n
Frequency	f/n	f
Power	CV^2f/n^2	CV^2f/n^2
Chip area	n times	10-20% increase

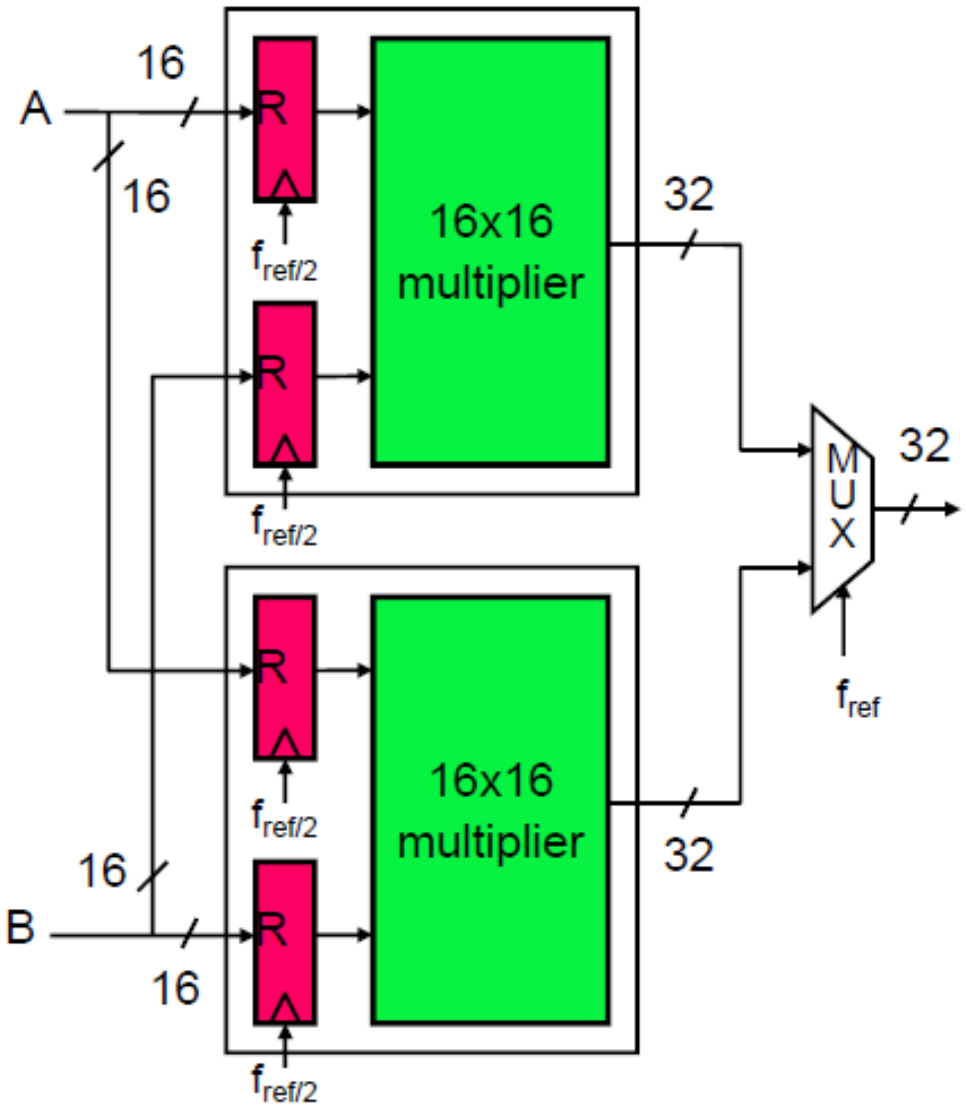
G. K. Yeap, *Practical Low Power Digital VLSI Design*, Boston: Kluwer Academic Publishers, 1998.

Parallel Architecture



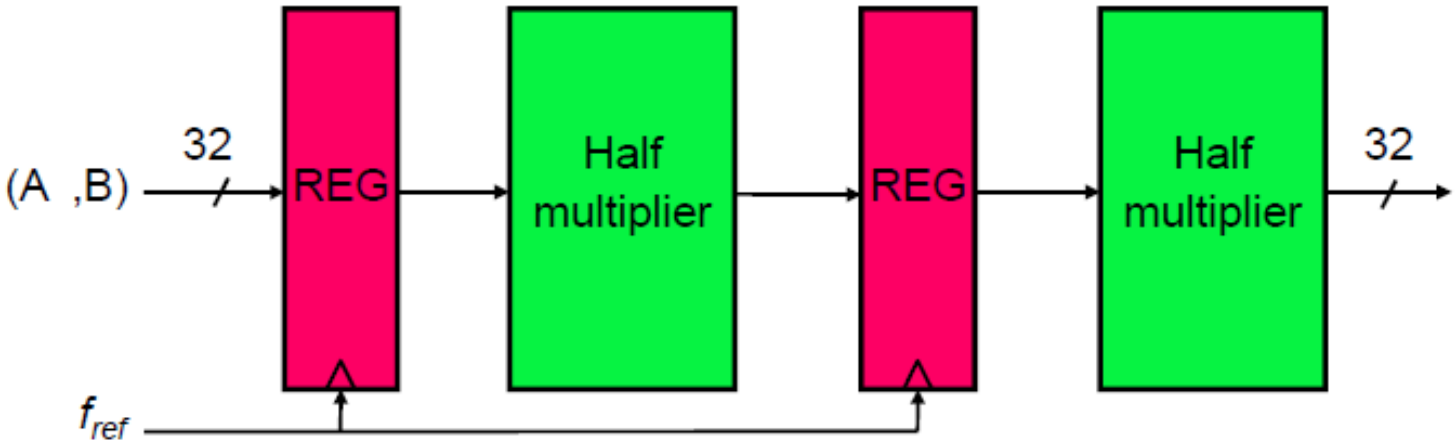
Assume that With the same 16x16 multiplier, the power supply can be reduced from V_{ref} to $V_{ref}/1.83$.

$$P_{parallel} = 2.2C_{ref}\left(\frac{V_{ref}}{1.83}\right)^2 \frac{f_{ref}}{2} = 0.33P_{ref}$$



Pipeline Architecture

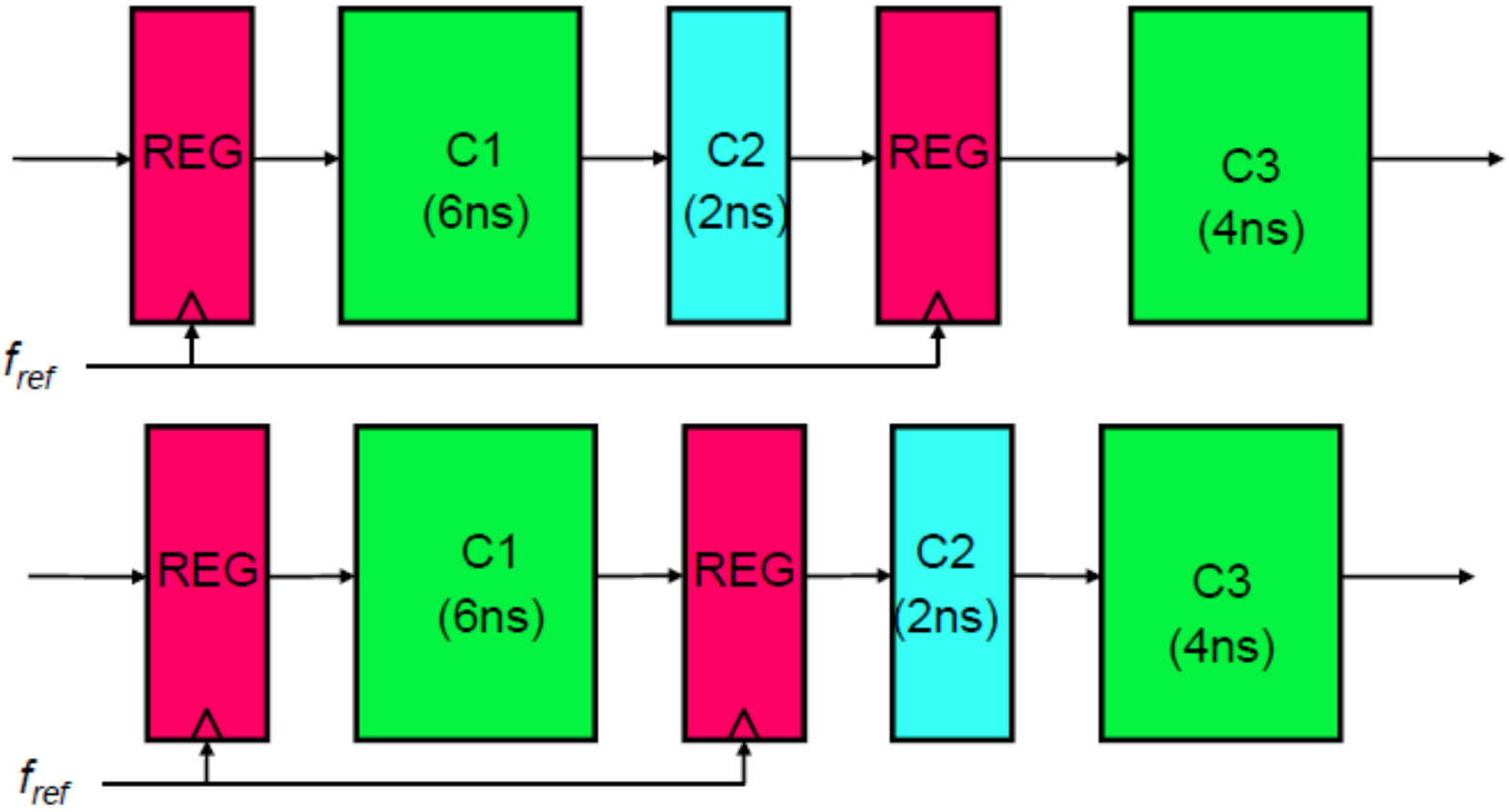
The hardware between the pipeline stages is reduced then the reference voltage V_{ref} can be reduced to V_{new} to maintain the same worst case delay. For example, let a 50MHz multiplier is broken into two equal parts as shown below. The delay between the pipeline stages can be remained at 50MHz when the voltage V_{new} is equal to $V_{ref}/1.83$



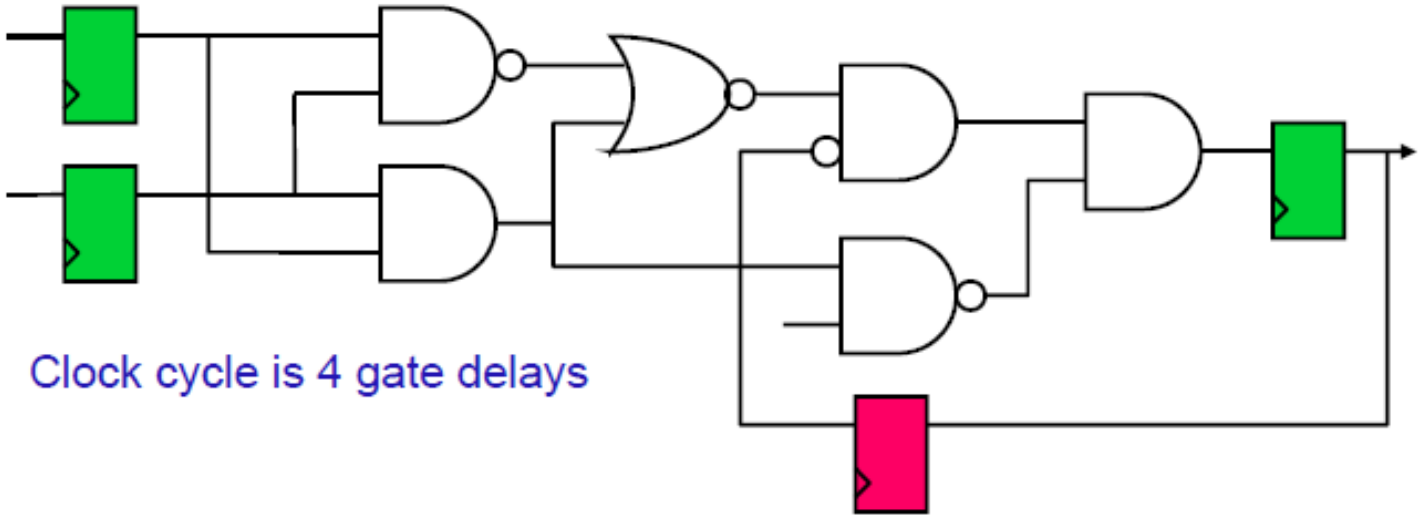
$$P_{pipeline} = 1.2C_{ref}\left(\frac{V_{ref}}{1.83}\right)^2 f_{ref} = 0.36P_{ref}$$

Pipeline Architecture with Retiming

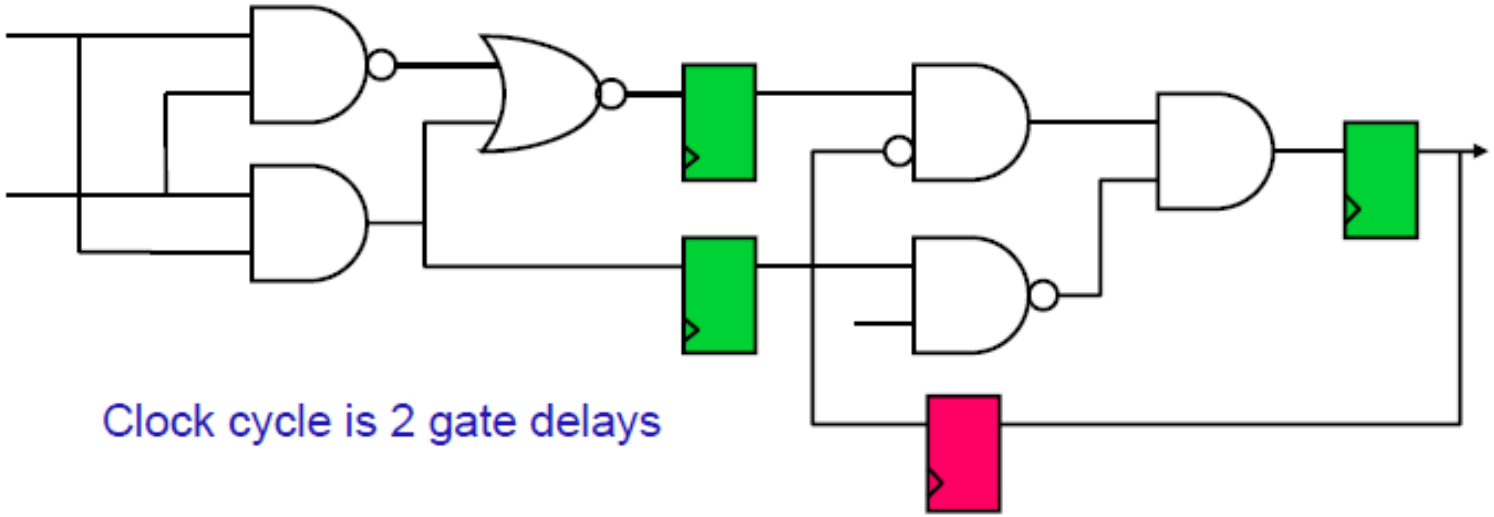
Retiming for pipeline design



Pipeline Architecture with Retiming



Clock cycle is 4 gate delays

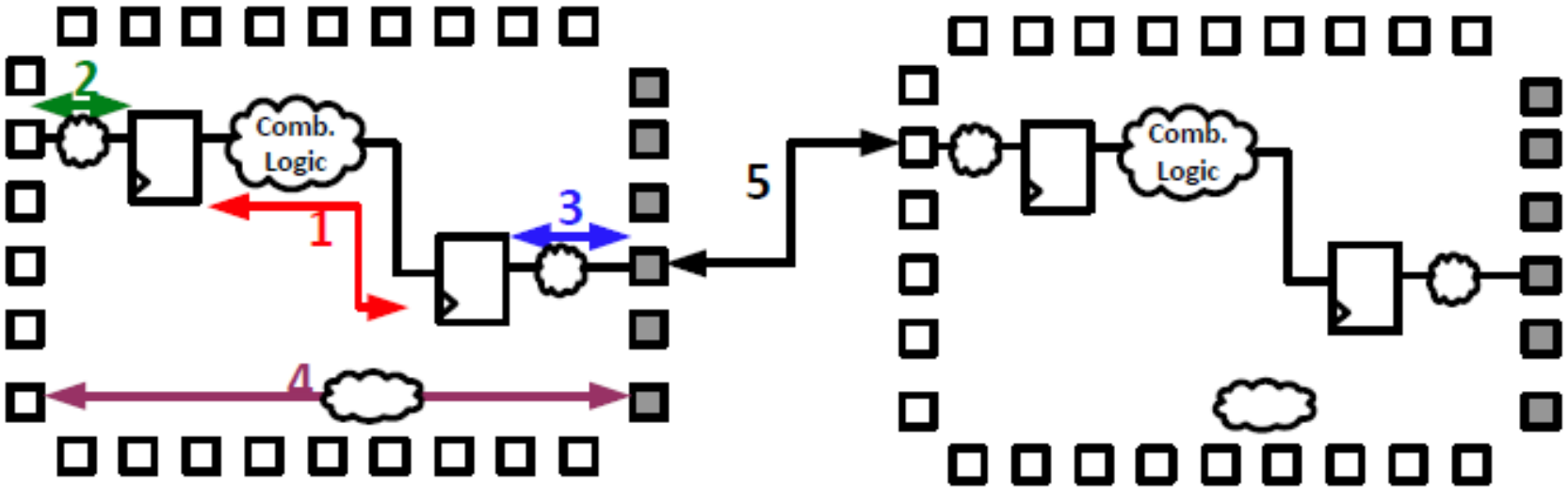


Clock cycle is 2 gate delays

When Pipelining when Parallelism?

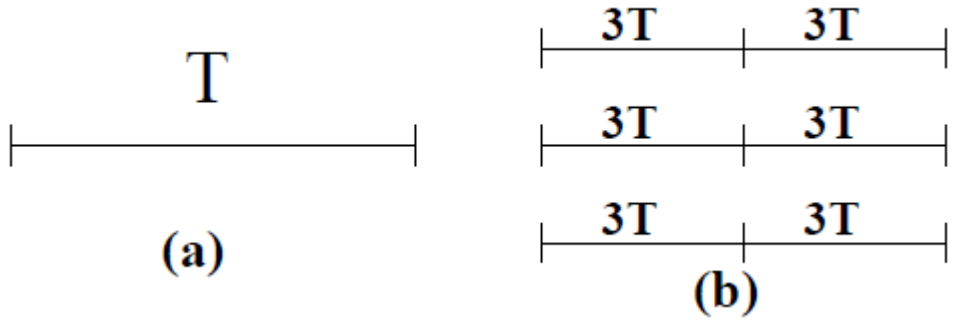
- Pipeline technique is used when the critical path is in the design (number 1, 2, 3, 4)

- Parallelism is used when the critical path is bounded by the communication or I/O bound. (number 5)
  - Pipelining does not help in this case!



Combining Pipeline and Parallel Processing for Low Power

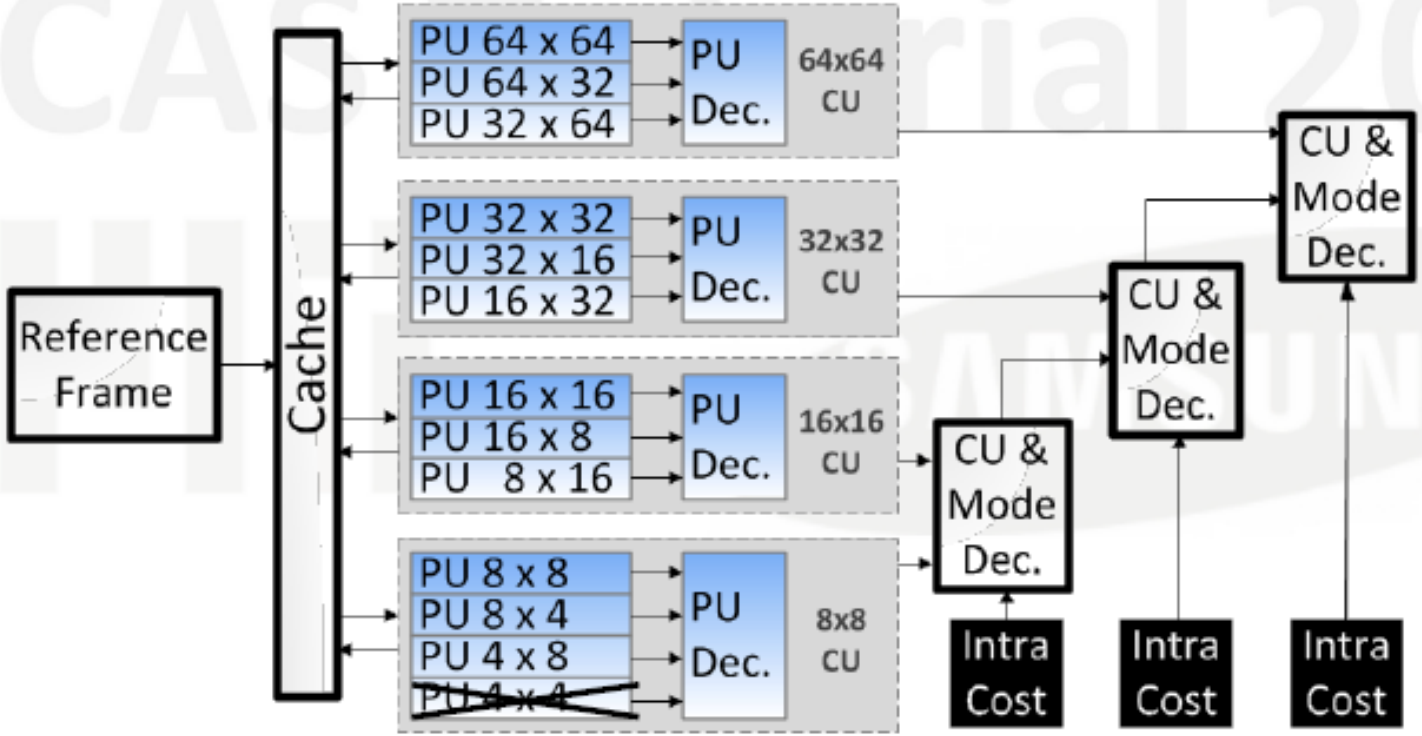
pipelining reduces the capacitance to be charged/discharged in 1 clock period, while parallel processing increases the clock period for charging/discharging the original capacitance



(a) charge/discharge of entire capacitance in clock period T
(b) Charge/discharge of capacitance in clock period 3T using two pipelines

Parallel Motion Estimation

- Perform motion estimation for each PU in inter-coded CU
- Process CUs in parallel to increase throughput
  - Share search pixels across engines to reduce memory bandwidth by 8x



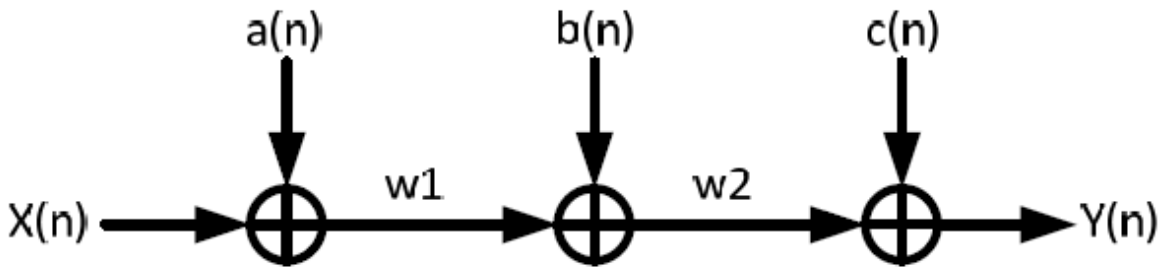
M. E. Sinangil et al., "Cost and Coding Efficient Motion Estimation Design Considerations for High Efficiency Video Coding (HEVC) Standard," *IEEE Journal of Selected Topics in Signal Processing*, 2013.

Parallelism Limitations

- Parallelism requires more overhead
  - More power, more components
  - At some point, more parallelism actually makes things slower
  - You spend too much time switching between tasks instead of doing actual work.

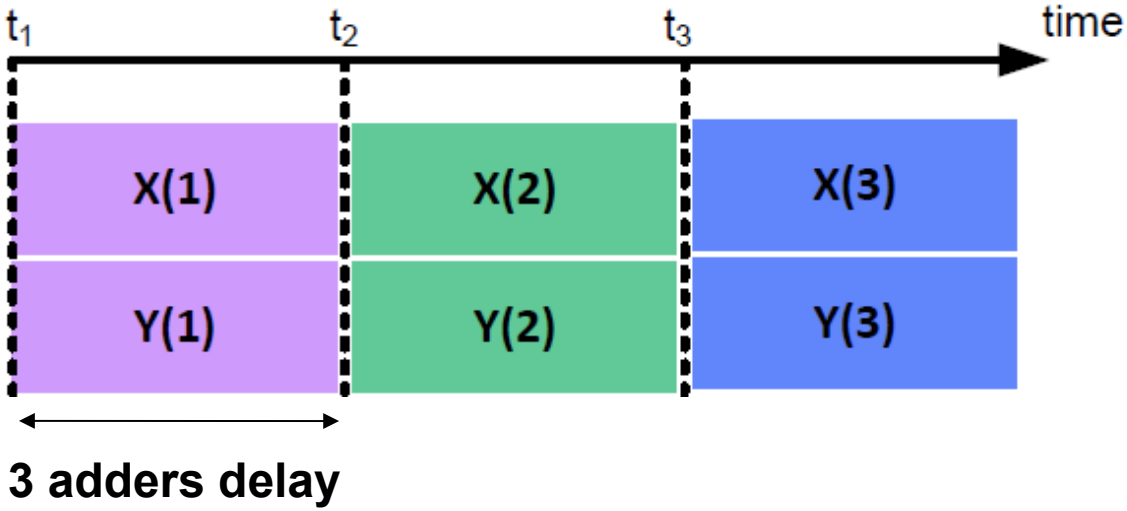
Verilog for Pipeline Architecture

- Pipeline depth: 0 (No Pipeline)
  - Critical path: 3 Adders



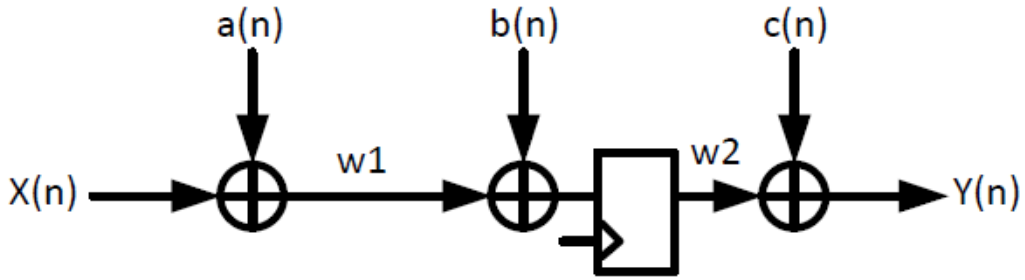

```
wire  w1, w2;
assign  w1 = X + a;
assign  w2 = w1 + b;
assign  Y = w2 + c;
```

- Latency : 0



Verilog for Pipeline Architecture

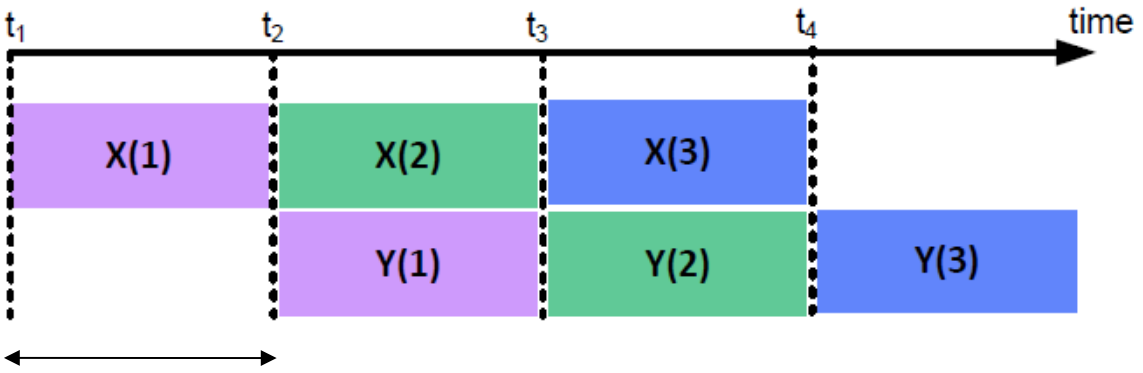
- Pipeline depth: 1 (One Pipeline register Added)
  - Critical path: 2 Adders




```
wire  w1;
reg   w2;
assign  w1 = X + a;
assign  Y = w2 + c;

always @(posedge Clk)
    w2 <= w1 + b;
```

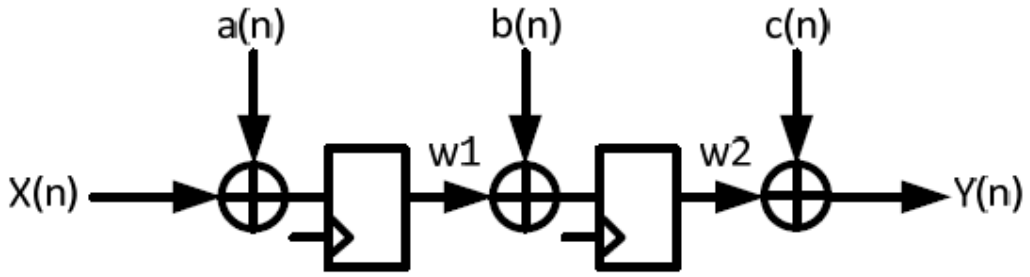
- Latency : 1



2 adders delay

Verilog for Pipeline Architecture

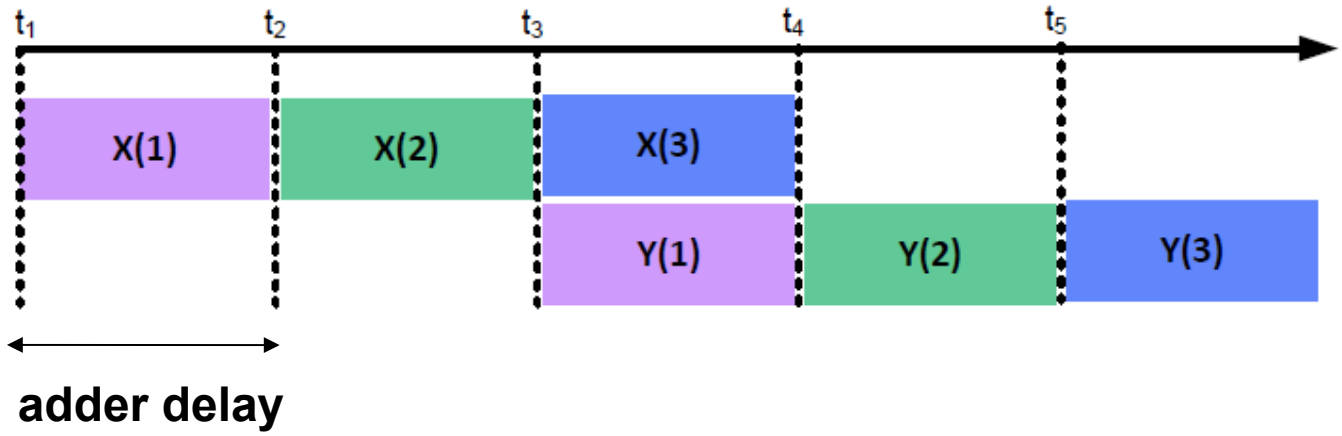
- Pipeline depth: 2 (One Pipeline register Added)
  - Critical path: 1 Adder



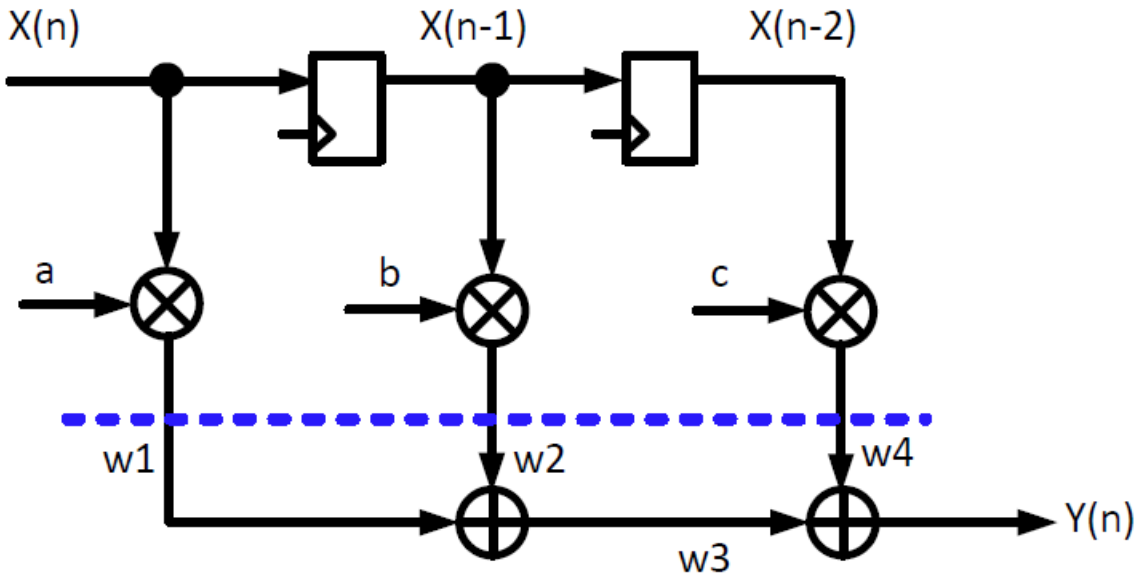

```
reg   w1, w2;
assign  Y = w2 + c;

always @(posedge Clk)
 begin
    w1 <= X + a;
    w2 <= w1 + b;
 end
```

- Latency : 2



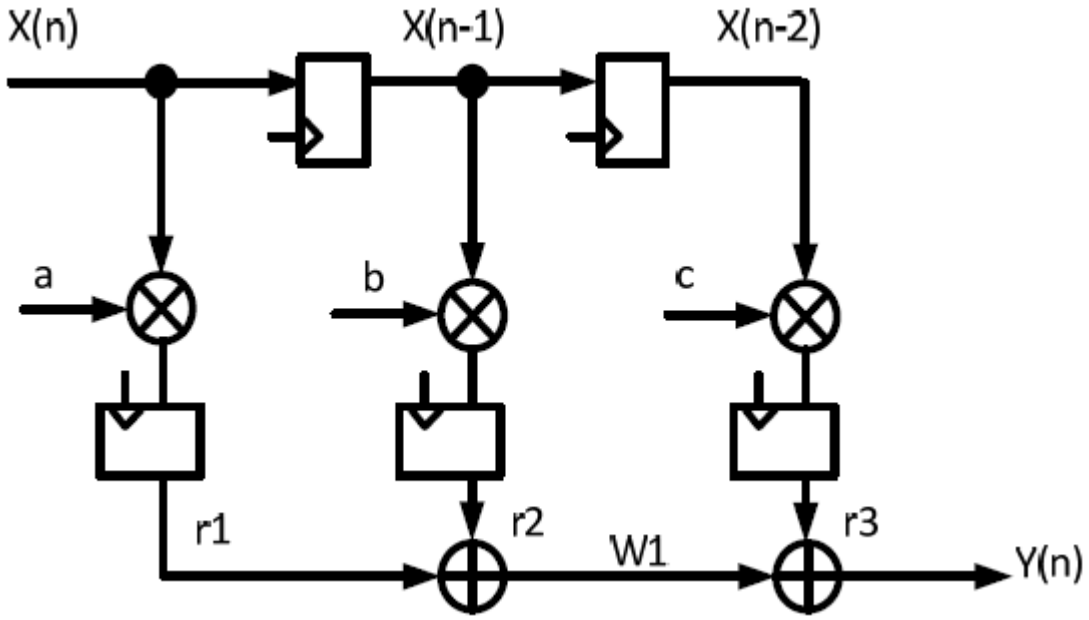
Verilog Practice




```
assign w1=a*Xn;
assign w2=b*Xn_1;
assign w3=w1+w2;
assgin w4= c*Xn_2;
assign Y=w3+w4;
always @ (posedge clk)
begin
    Xn_1 <= Xn;          Xn_2 <= Xn-1;
end
```

Critical path= 1M+2A

Verilog Practice

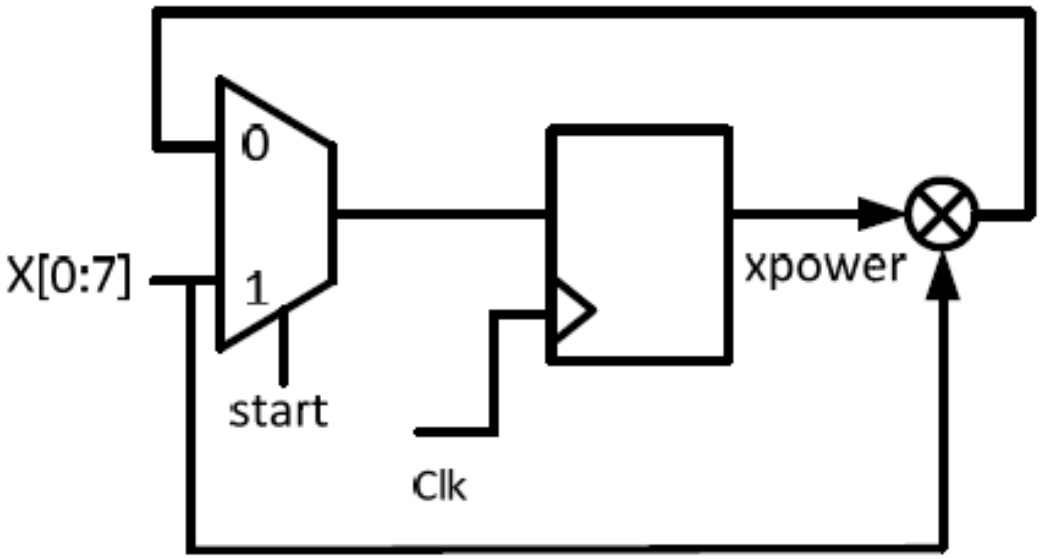



```
assign Y = r3 + w1;
assign w1 = r1 + r2;
always @ (posedge clk)
    begin
        Xn_1 <= Xn;
        Xn_2 <= Xn_1;
        r1 <= a * Xn;
        r2 <= b* Xn_1;
        r3 <= c* Xn_2;
end
```

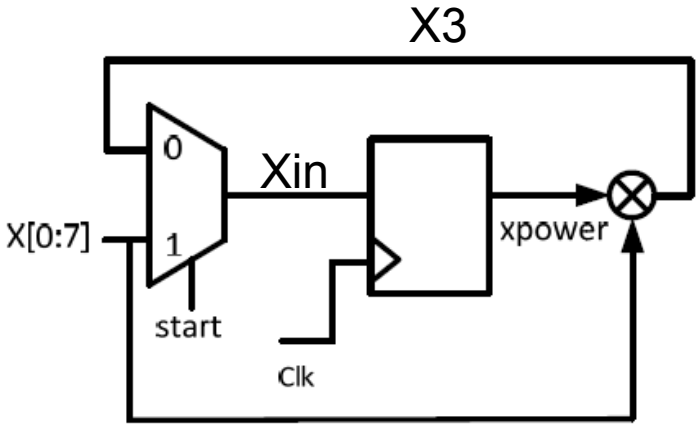
Critical path= max(1M,2A)

Verilog Practice

- Calculation of X^3
  - Latency = 3 clocks
  - Timing = one multiplier in the critical path
- Iterative implementation
- No new computations can begin until the previous computation has completed



Verilog Practice

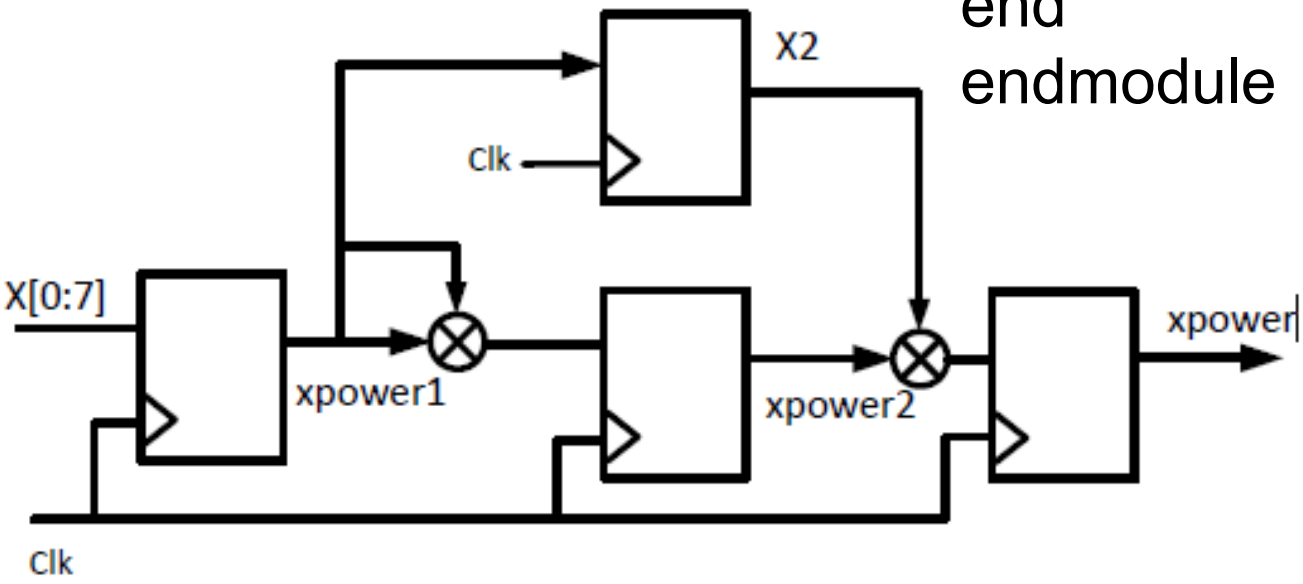



```
Module power3 (X3, finished, X, clk, start);
output reg [7:0] X3;
output finished;
input [7:0] X;
input clk, start;
reg [7:0] ncount, Xpower, Xin;
assign finished = (ncount == 0);
always@(posedge clk)
      if (start) begin
            Xpower<=X; Xin<=X;
            ncount <= 2;
            X3 <= Xpower;
      end
      else if(!finished) begin
            ncount <= ncount -1;
            Xpower <= Xpower * Xin;
end
endmodule
```

Verilog Practice

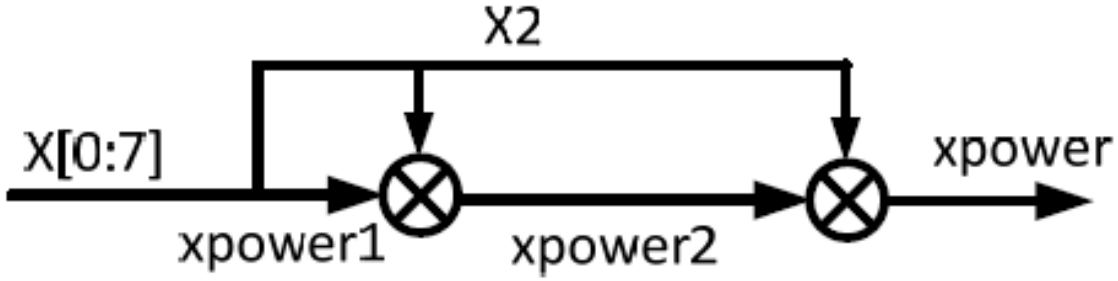
```
module power3 (Xpower, X, clk);
output reg [7:0] Xpower;
input clk;
input [7:0] X;
reg [7:0] Xpower1, Xpower2, X2;
```

```
always@ (posedge clk) begin
// pipeline stage 1
Xpower1 <= X;
// pipeline stage 2
Xpower2 <= Xpower1*Xpower1;
X2 <= Xpower1;
// pipeline stage 3
Xpower <= Xpower2 * X2;
end
endmodule
```



Verilog Practice

- Calculation of X^3
  - Latency can be reduced by removing pipeline registers



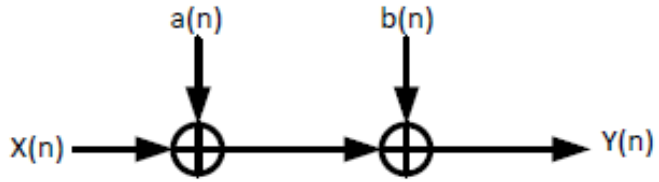

```
module power3(Xpower, X);
output [7:0] Xpower;
input [7:0] X;
reg [7:0] Xpower1, Xpower2;
reg [7:0] X1, X2;
always @ *
    Xpower1 = X;
```

```
always @ (*)
    begin
X2 = Xpower1;
Xpower2 = Xpower1 *Xpower1;
end

assign Xpower = Xpower2 * X2;
endmoudule
```

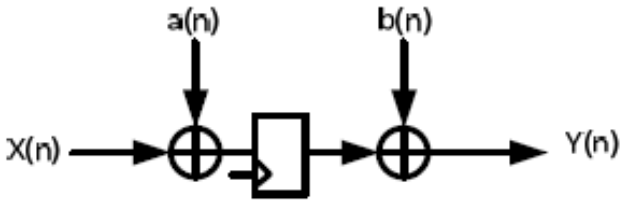
Verilog Practice

- In parallel processing the same hardware is duplicated to: (1) increases the throughput without changing the critical path, (2) increases the silicon area



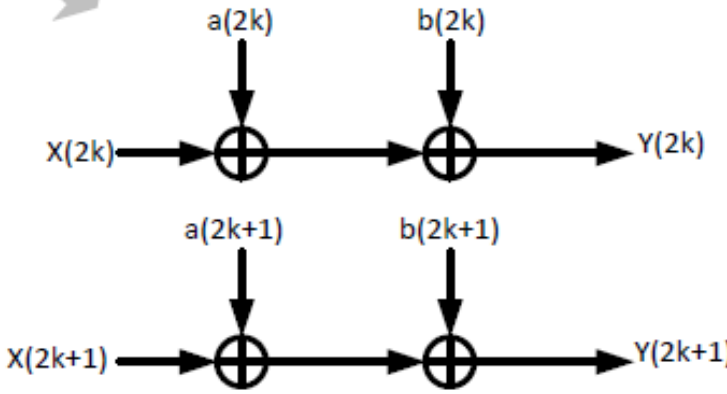
Clock Freq: f
Throughput: M samples

Pipelining



Clock Freq: 2f
Throughput: 2M samples

Parallel Processing



Clock Freq: f
Throughput: 2M samples