Time Shared Architecture

Lecture 9

Dr. Shoab A. Khan

Time-Shared Designs

- Dedicated Fully Parallel
  - If sampling rate fs = circuit clock fclk
  - Dedicated Operator for each operation in the algorithm
  - One operator (hardware unit, e.g. adder, multiplier, register) for each operation (e.g addition, multiplication, delay)

- Most designs: time multiplexing
  - clock frequency ! = sample frequency

$$\text{clock frequency} = \frac{\text{sample frequency}}{\text{number of clock cycles available for the job}}$$

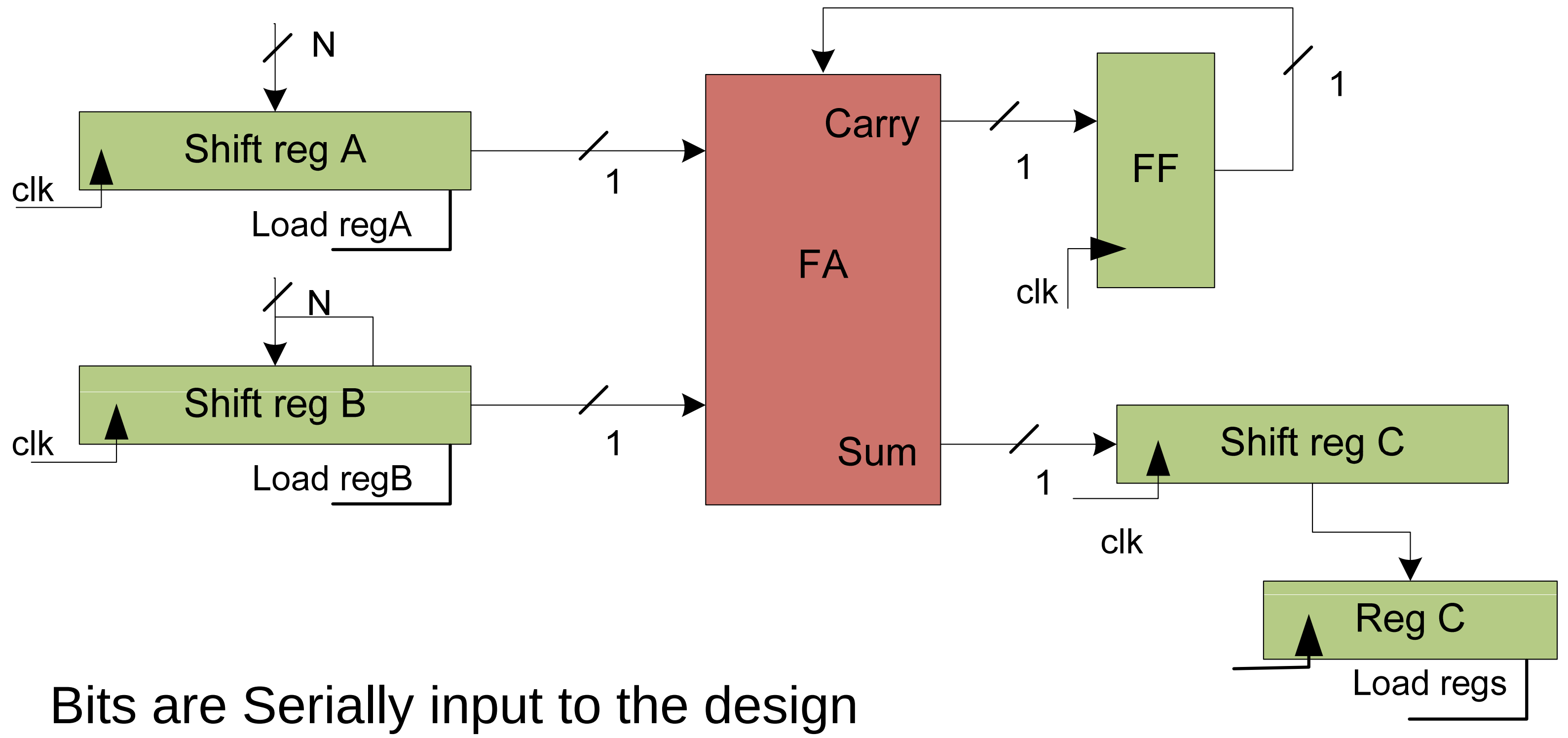
Time Shared Architecture

- Reverse of dedicated architecture
- Less hardware
- Reuse a smaller block to perform complex operations
- Reuse blocks to execute algorithms
- Require controller to schedule operations on the shared HW

Time Shared Architectures

- Bit Serial
- Word Serial
- Sequential
- Sequential Unfolding
- Systolic
- Folded DFG
- Micro-programmed

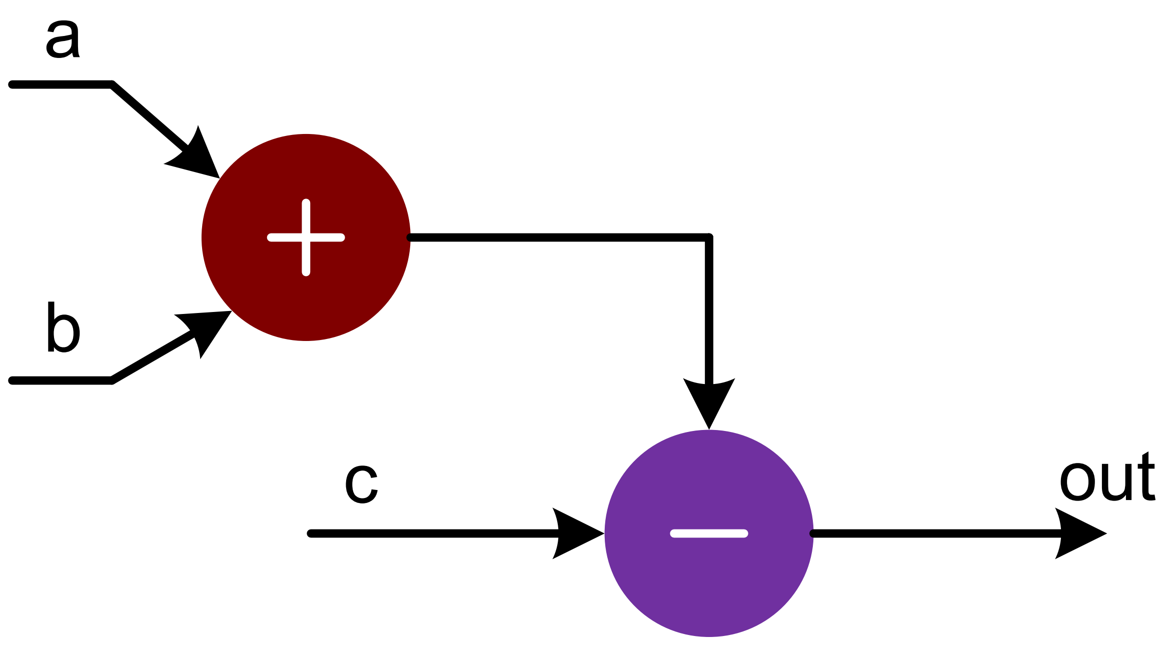
Example Bit Serial: N-bit adder



- Bits are Serially input to the design
- The architecture processes input bit by bit basis
- Bits are out serially

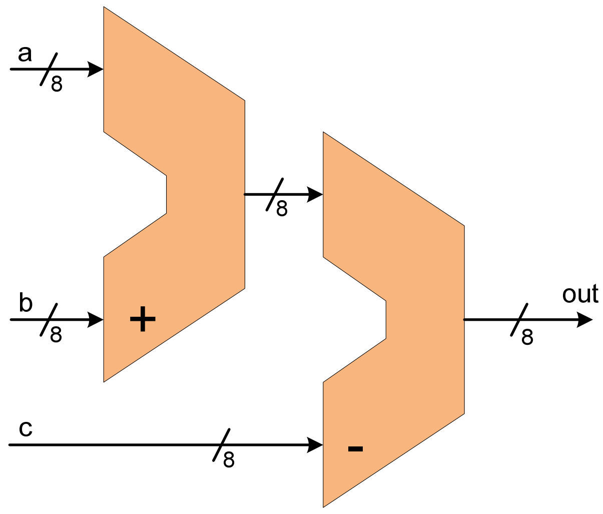
Example: Mapping from Dedicated to Time Shared Architecture

- a, b, c are 8-bit numbers
- The example maps the DFG to dedicated and time shared architecture



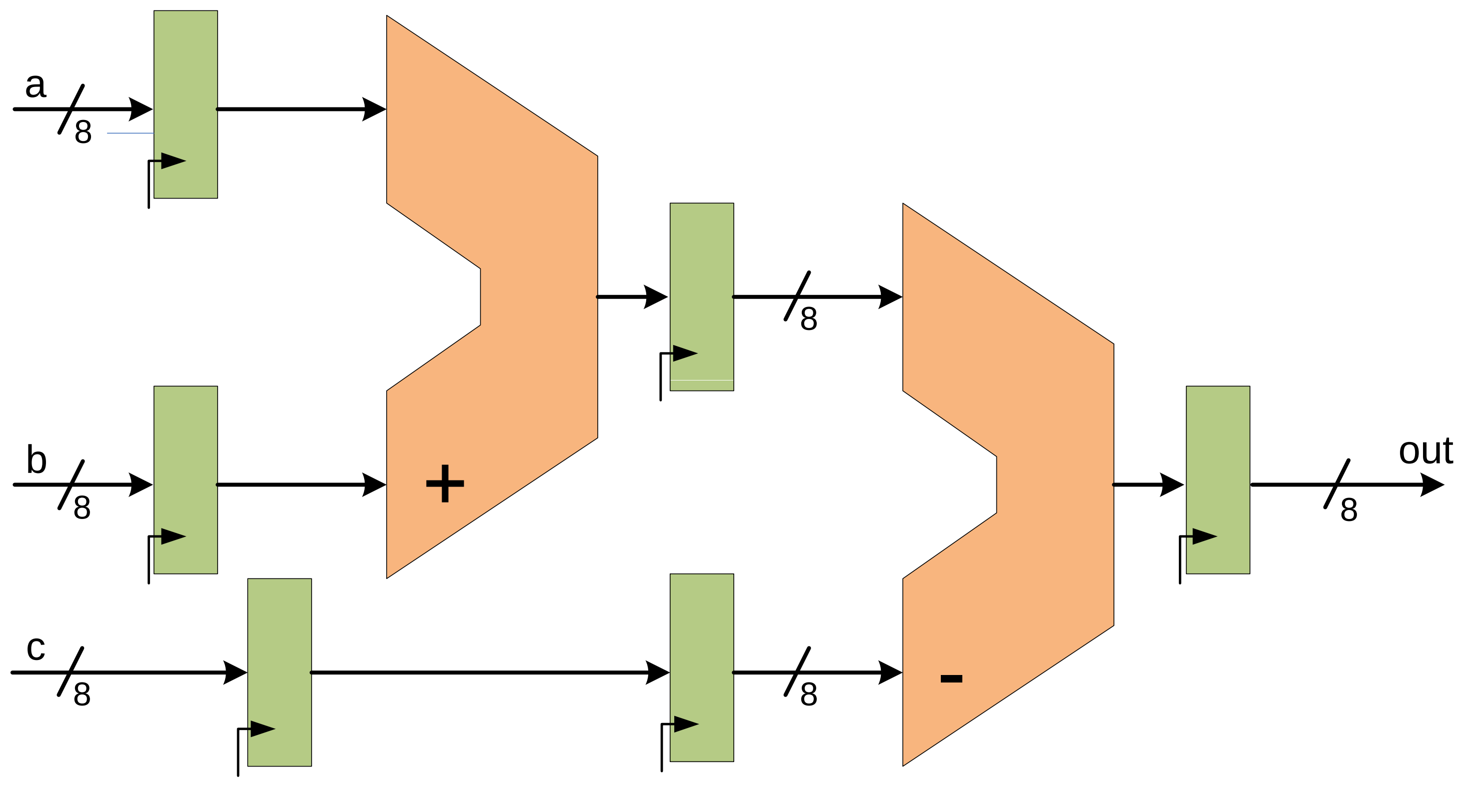
Dedicated Architecture

- Two add bit adder/subtractor for addition and subtraction
- No pipelining
- Need to add pipelining for better performance



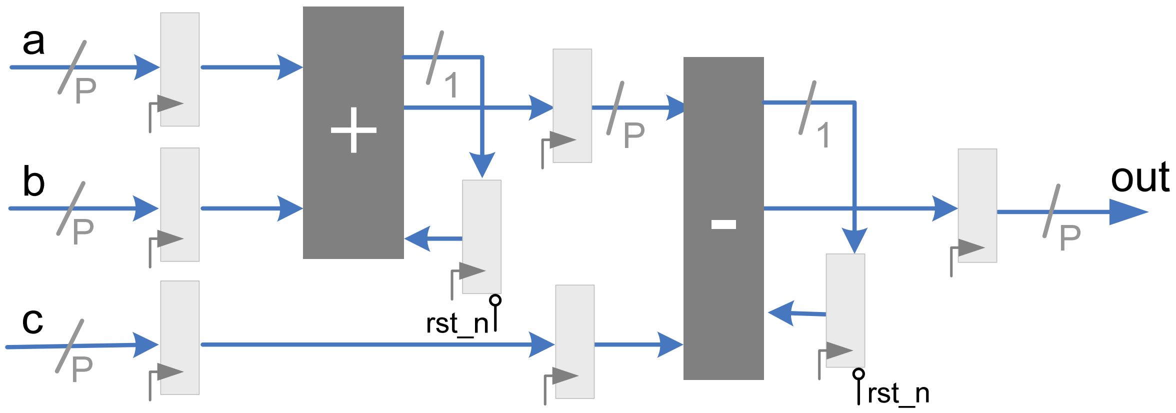
Pipeline Architecture

- Remember data coherency is important



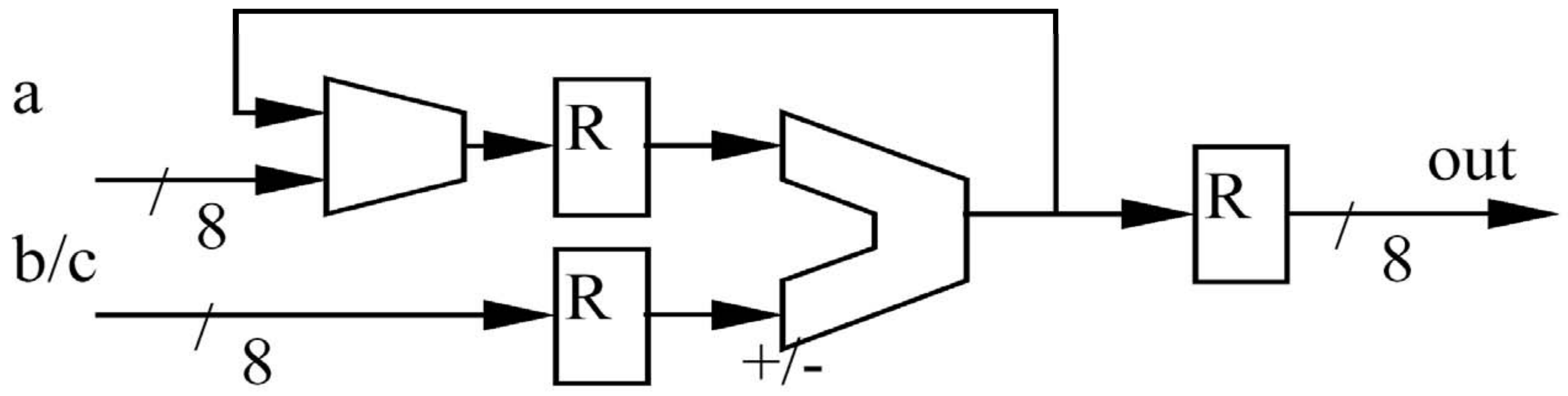
If f_c is N times f_s-> bit-serial

- If N=8 cycles are available: Bit Serial
  - Bit-serial data in
  - Two FA for 1-bit addition and subtraction
  - Flip flops for pipelining
- Word Serial
  - If N=8, 4, 2 the number of bits P=1, 2, 4



If f_c is twice of f_s -> Folded Design

- If N=8 cycles are available: Bit Serial
- Time shared architecture
- One adder/subtractor is used for addition and subtraction to compute a+b-c
- Two cycles to execute the DFG

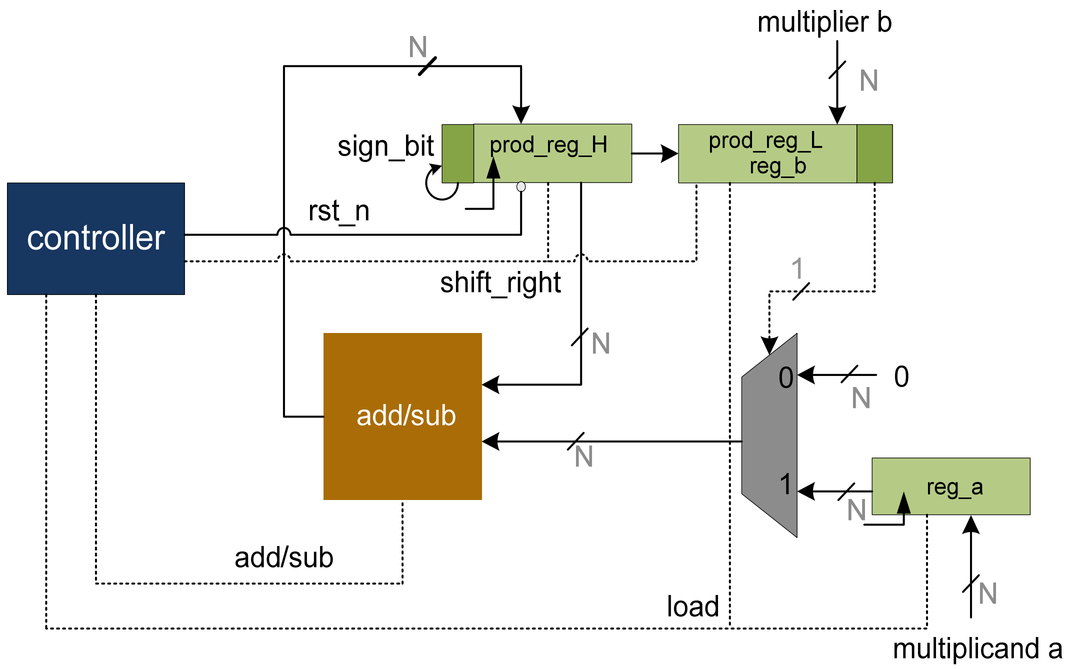


Sequential Design: Shift and Add Multiplier

```
$sum = 0;$
$for(i = 0; < N-1; i++)$
$sum += a \times b[i] \times 2^{i};$
$sum += a \times b[N-1] \times 2^{N-1};$
$prod = sum;$
```

- Sums each partial product, one at a time.
- Each partial product is shifted versions of A or 0.

Sequential Design: Shift and Add Multiplier

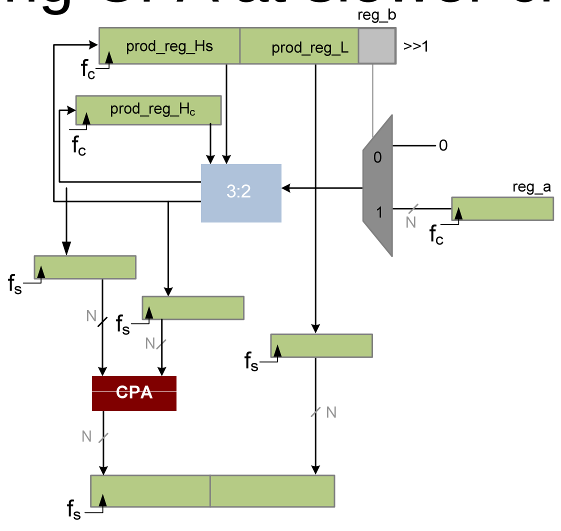


Control Algorithm

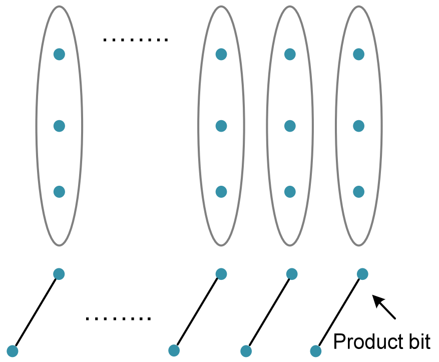
1. Prod_reg_L (P) ← 0,
   reg_a ← multiplicand,
   Prod_reg_H (regB) ← multiplier
2. If LSB of reg_b==1 then
   add reg_a to Prod_reg_L
   else add 0
3. Shift Prod right 1
4. Repeat steps 2 and 3 **n-1** times
5. Prod has product

Optimized Design

- For sequential multiplier, use compression tree
- Latch in registers at slower clock
- Add using CPA at slower clock



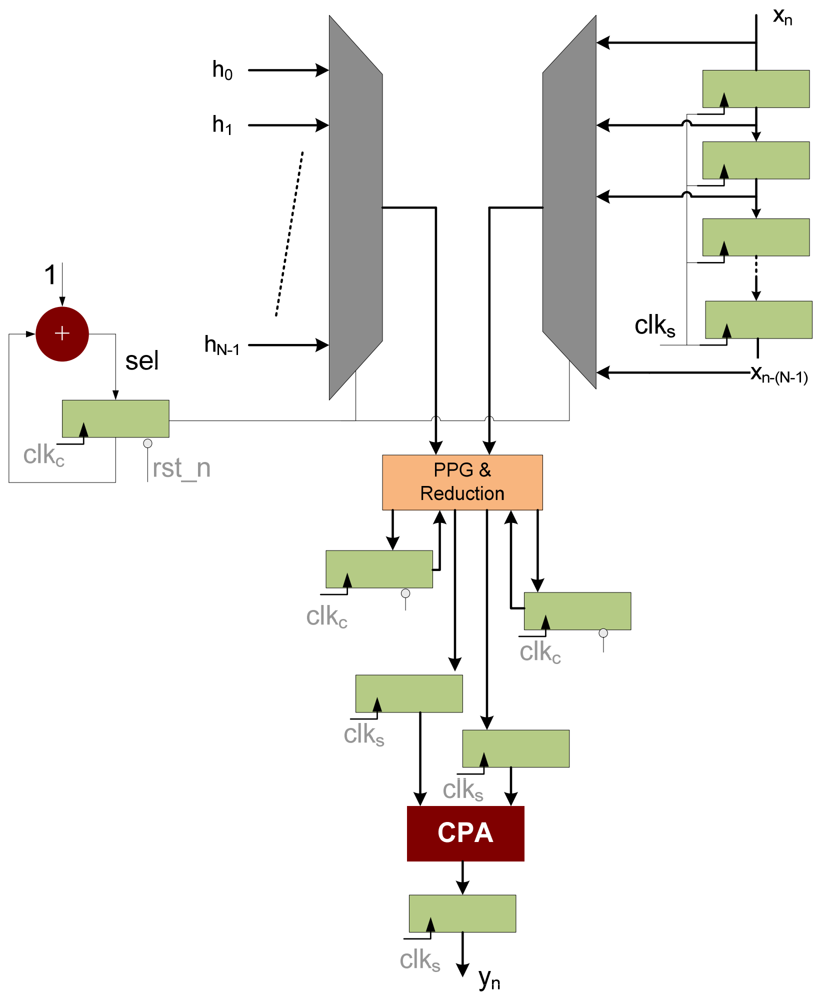
(a)



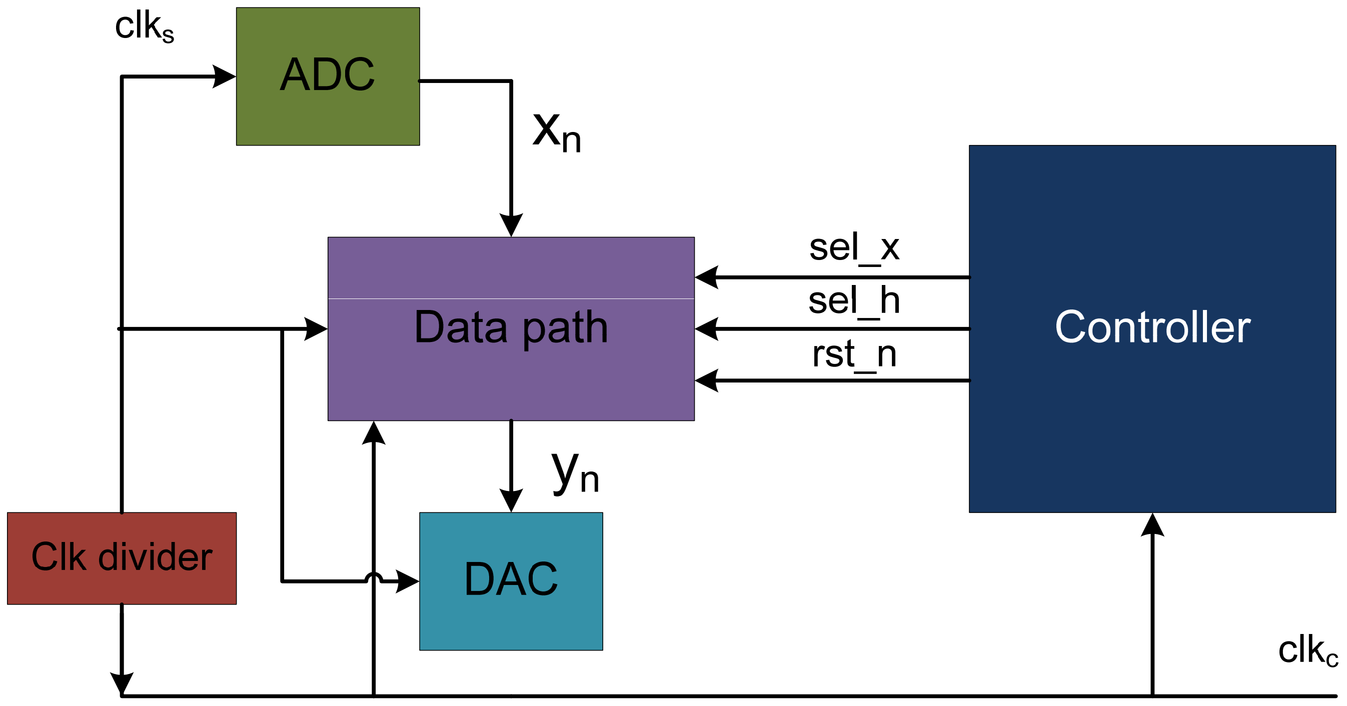
(b)

Time shared FIR filter Complete Datapath Design

- Tap delay line at sampling clock
- Filter Coefficients in a ROM
- PP generation and compression tree at fast clock
- Final addition using CPA at sampling clock

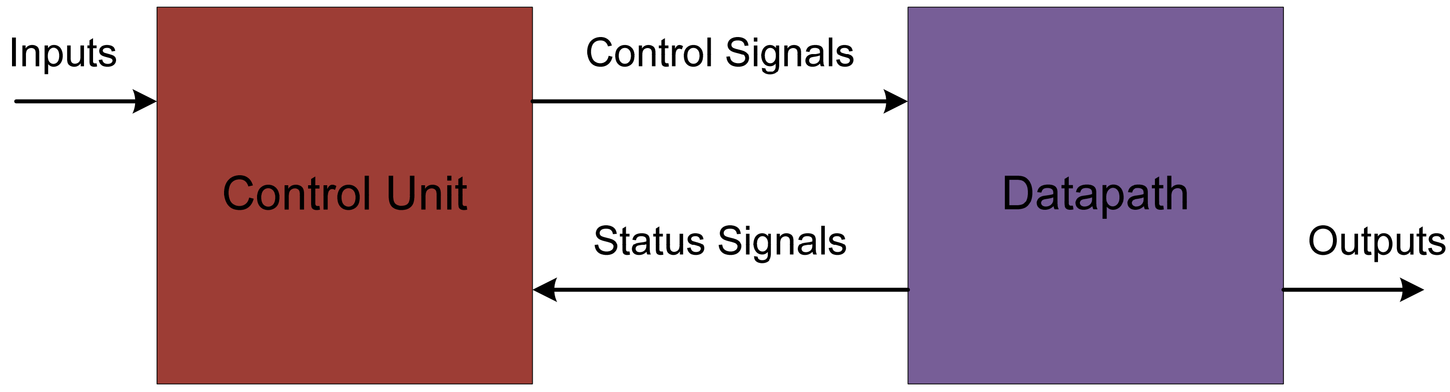


Complete Design



Sequencing and Control

- Digital Systems can be partitioned into two portions
- Datapath and Control Unit



Puppeteer Controller who pulls the strings

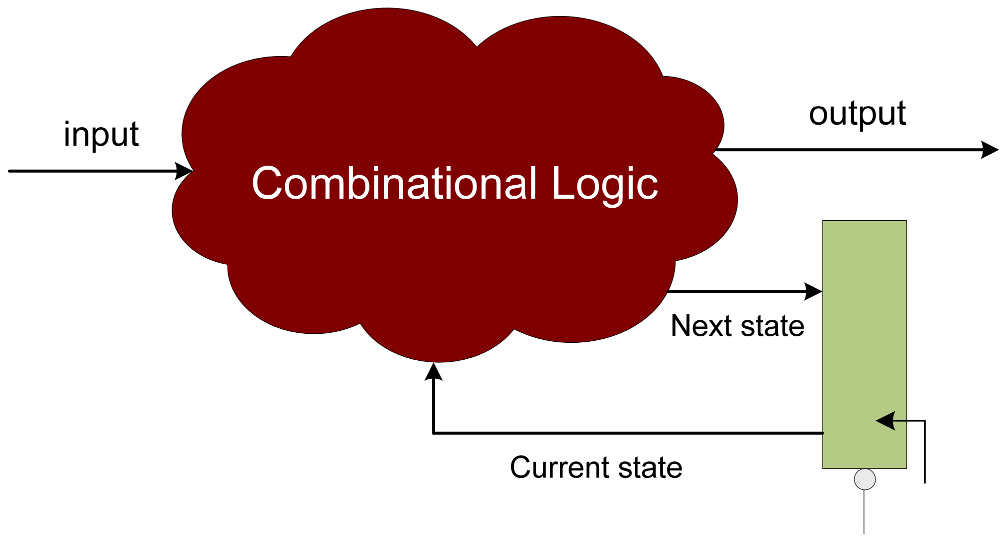
- Selects the operation
- Determines the sequence (based on status and input signals)

Datapath Puppet

- Registers, MUXes, ALU, Multipliers, Shifters, Comb, Circuits and buses
- Implements operations under control of the control signals

State machine

- A machine activity is usually consists of a synchronous sequence of operations on the registers of its datapaths, under the direction of a controlling state machine



ASM Implementation

- Two general implementation methods
  - Hardwired
  - Micro-programmed
- Two representation
  - State Diagram
  - Algorithmic State Machine
- Two Types of State Machines
  - Mealy State Machine
  - Moor State Machine

A Finite State Machine (FSM) Controller

- Controller are of two types:
  - Hardwired Finite State Machine based controller
  - Microprogramm Architecture based controller
- A FSM is a sequential system with N flip-Flops and has 2^N possible states, so the number of possible states is FINITE
- FSM can be described using a Bubble Diagrams (State Diagram) or Algorithmic State Machine Charts (ASM)

A Finite State Machine (FSM) Controller

- Controller are of two types:
  - Hardwired Finite State Machine based controller
  - Microprogramm Architecture based controller
- A FSM is a sequential system with N flip-Flops and has 2^N possible states, so the number of possible states is FINITE
- FSM can be described using a Bubble Diagrams (State Diagram) or Algorithmic State Machine Charts (ASM)

FSM Comparison

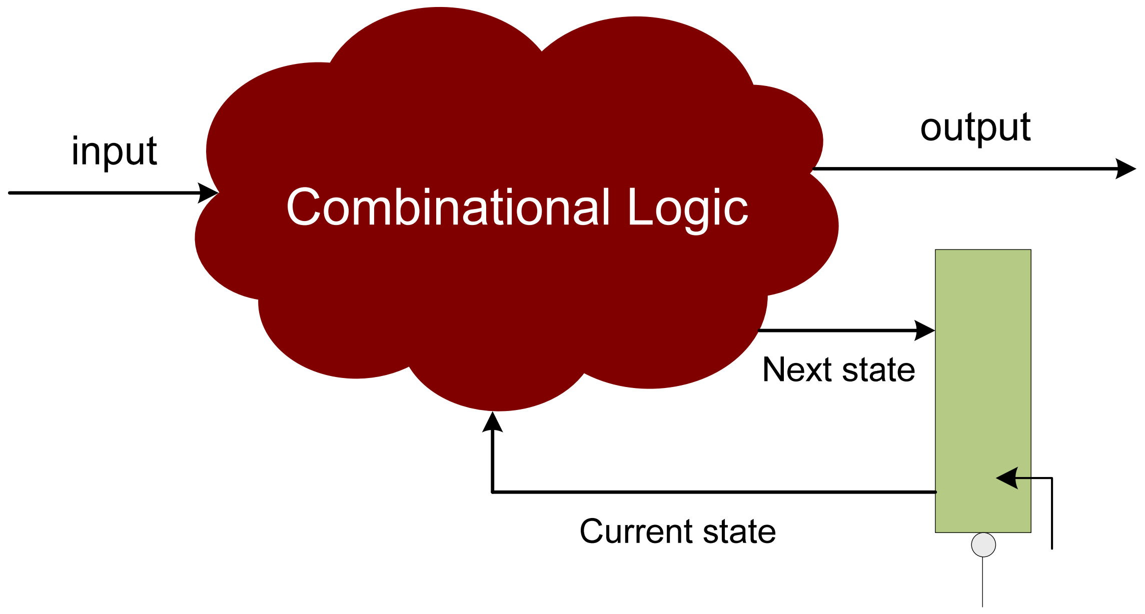
Moore Machine

- output function only of present state
- maybe more state
- synchronous outputs
  - no glitching
  - one cycle "delay"
  - full cycle of stable output

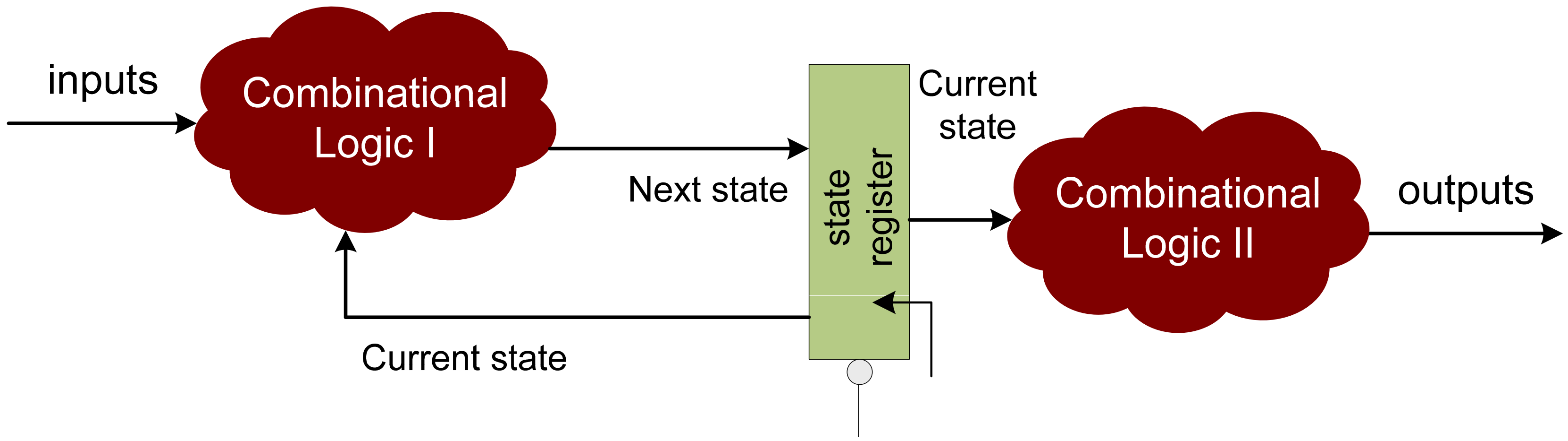
Mealy Machine

- output function of both present states & input
- maybe fewer states
- asynchronous outputs
  - if input glitches, so does output
  - output immediately available
  - output may not be stable long enough to be useful

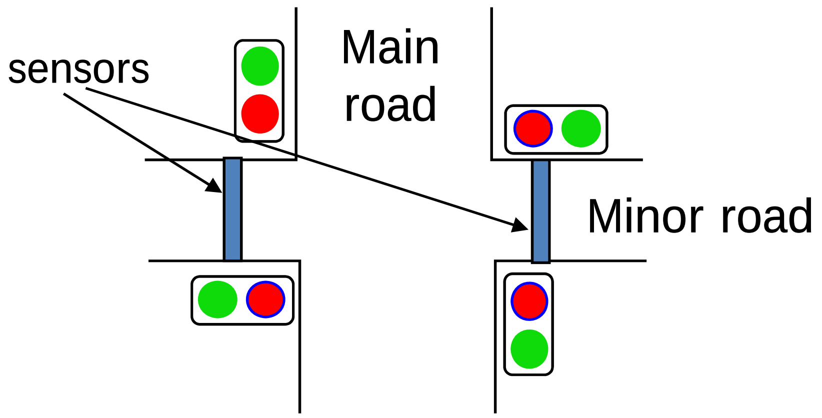
The composition of Mealy machine implementation of an FSM



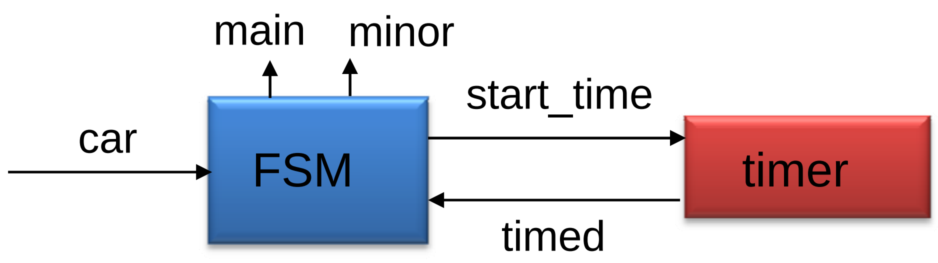
The components in Moore machine implementation of a state machine based design



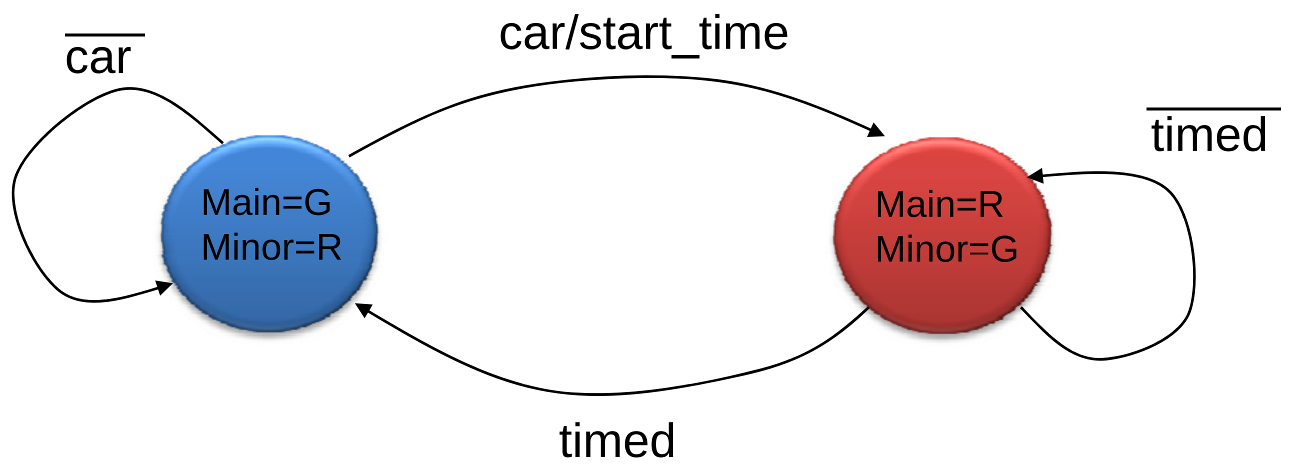
Example: Simple traffic signal controller



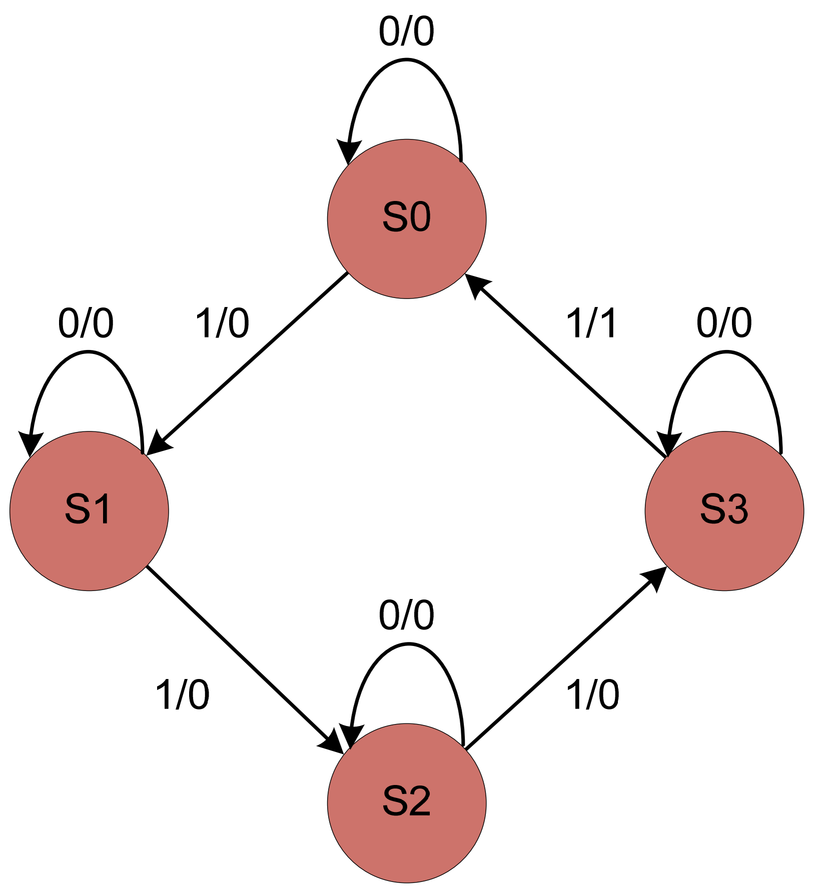
- Main road has normally a green light, and minor road a red light
- If a car is detected on minor road (sensor), semaphores change values, a timer is started which asserts a signal ‘TIMED’ at the end of counting
- When TIMED is asserted, the semaphores go back to default values



State diagram



State diagram



State Encoding: One-Hot versus Binary Coded Finite-State Assignment

parameter [1:0] S0 = 2'd00, S1 = 2'd01, S2 = 2'd10, S3 = 2'd11;	parameter [3:0] S0 = 4'b0001, S1 = 4'b0010, S2 = 4'b0100, S3 = 4'b1000;	parameter [2:0] S0 = 4'b000, S1 = 4'b001, S2 = 4'b010, S3 = 4'b100;	parameter [2:0] S0 = 4'b000, S1 = 4'b001, S2 = 4'b011, S3 = 4'b010;
(a)	(b)	(c)	(d)

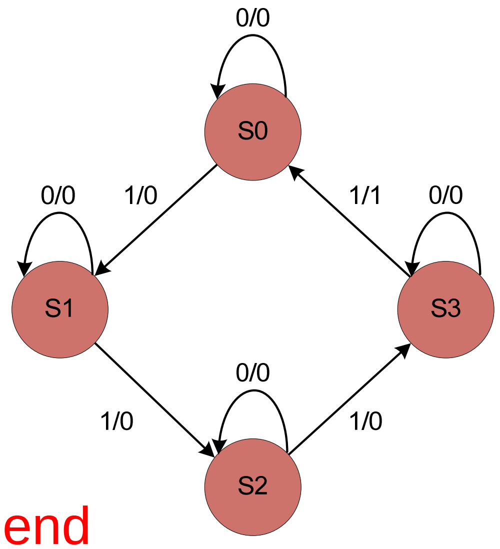
State Encoding Techniques (a) Binary (b) One-Hot (c) Almost One-Hot (d) Gray

Guideline for Coding State Machines

- Separate the state machine HDL description into two processes, one for the combinational logic and one for the sequential logic
- Use `define/parameter statements to define a state vector.
- Keep FSM logic and non-FSM logic in separate modules.

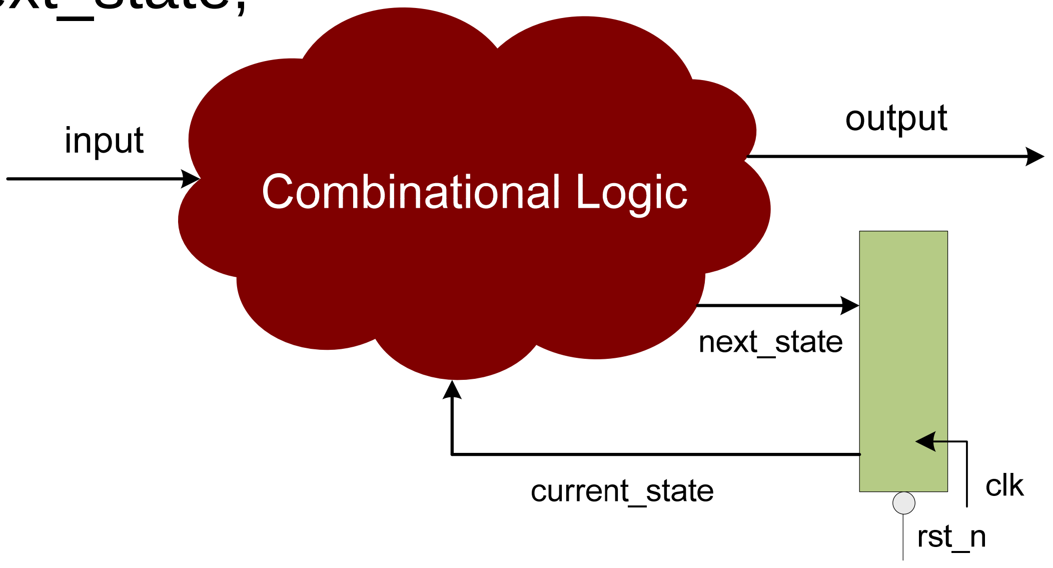
Translating FSMs into Verilog HDL: Combinational Part

```
always @(current_state or in)
begin
case (current_state)
    S0:
    begin
        if (in)
                begin next_state = S1; out = 1'b0; end
        else
                begin next_state = S0; out = 1'b0; end
    end
    S1:
    begin…
    end
endcase
```



Sequential Part

```
always @(posedge clk or negedge res_n)
begin
if (!res_n)
    current_state <= S0;
else
    curret_state <= next_state;
end
```



Why state diagrams are not enough

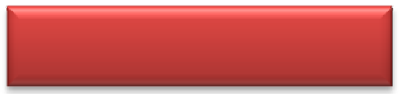
- Not flexible enough for describing very complex finite state machines
- Not suitable for gradual refinement of finite state machine
- Do not obviously describe an algorithm: that is, well specified
- Gradual shift towards program-like representations:
  - Algorithmic State Machine (ASM) Notation
  - Hardware Description Languages (e.g., Verilog, VHDL)

Algorithmic State Machine

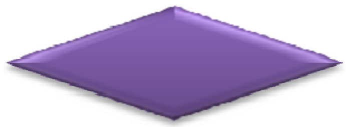
- A flowchart-like graphical notation that describes the cycle-by-cycle operations of an algorithm
- Each step takes one clock cycle
- Composed of rectangles, diamonds, ovals, and arrows interconnecting them
- Moore machines do not have ovals
- Mealy machines contain ovals
- Describes behavior rather than structure
- Provides a mechanism for performing systematic step-by-step design
- Can be directly translated to Verilog code
- Used to design synchronous sequential circuits

Algorithmic State Chart (ASM)

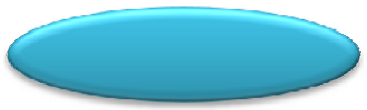
- An ASM chart is used to describe FSM behavior
  - Only three action signals can appear within an ASM chart:



- **State box.** Each box represents a state. Outputs within a state box is an UNCONDITIONAL output (always asserted in this state)

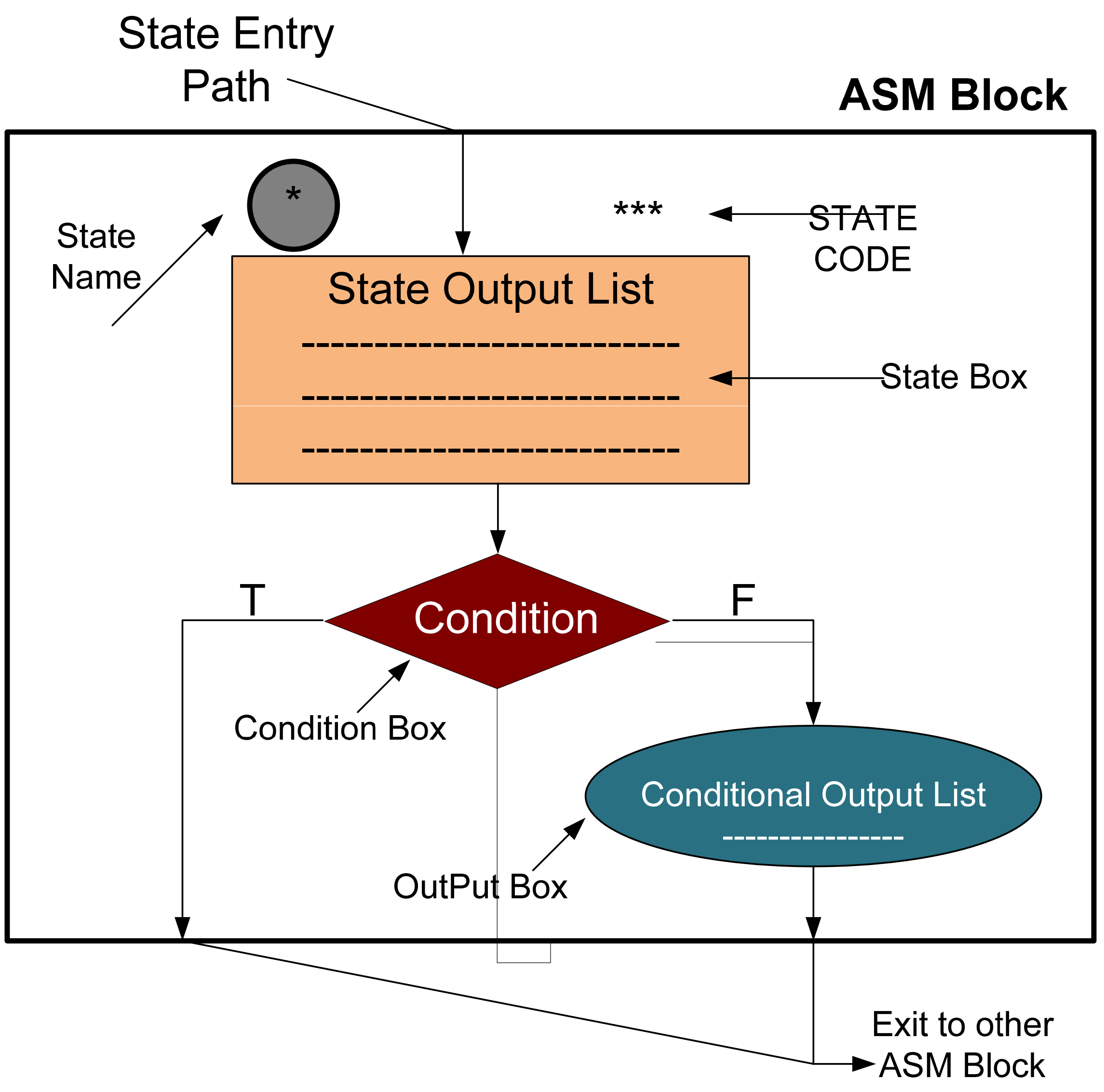


- **Decision box.** A condition in this box will decide next state condition



- **Conditional output box.** If present, will always follow a decision box; output within it is conditional.

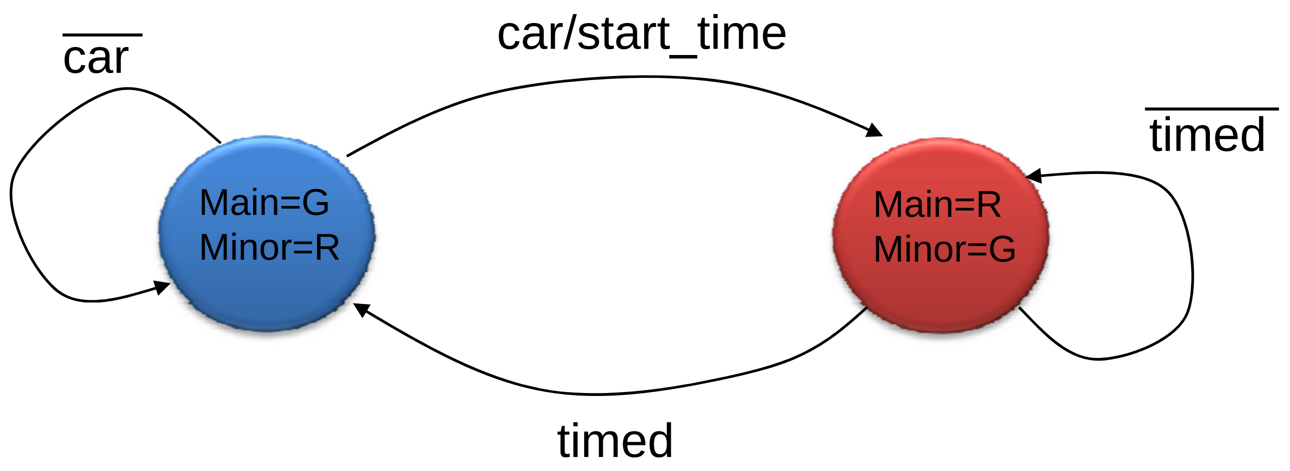
ASM Block

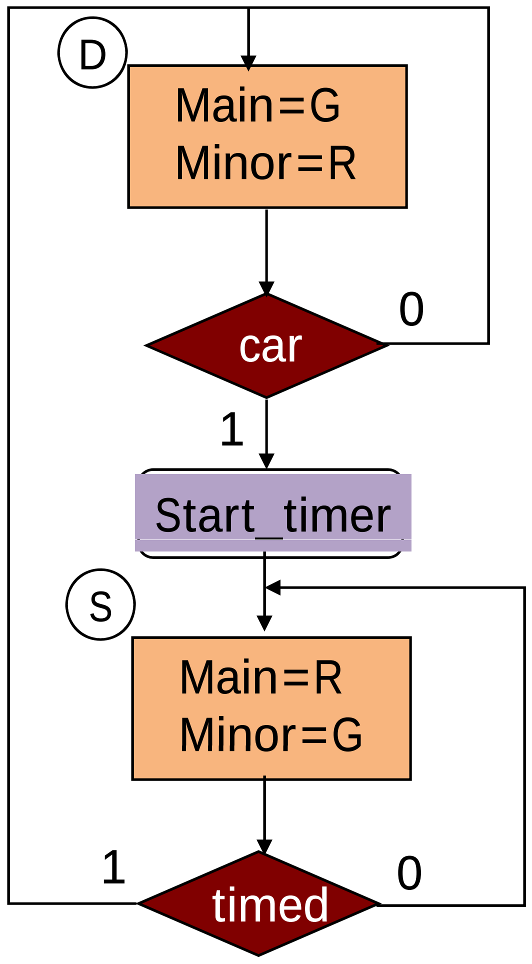

State Entry Path
ASM Block
*
***
~~STATE~~ CODE
State Name
State Output List

State Box
T
Condition
F
Condition Box
Conditional Output List

OutPut Box
Exit to other ASM Block

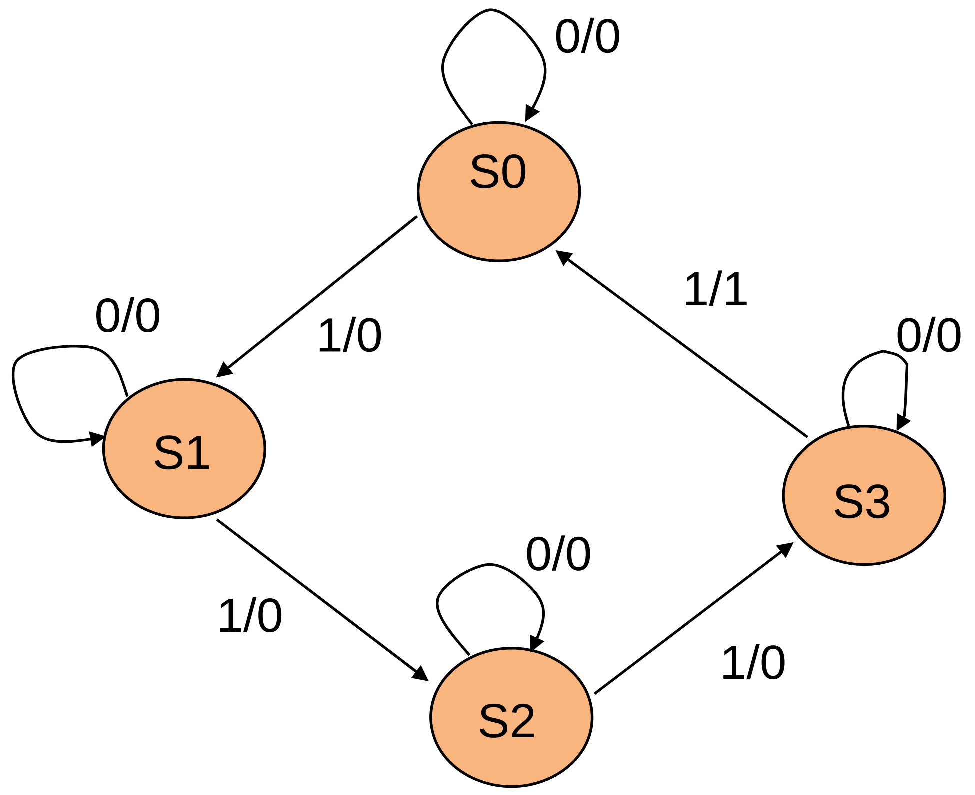
Algorithmic state machine (ASM)



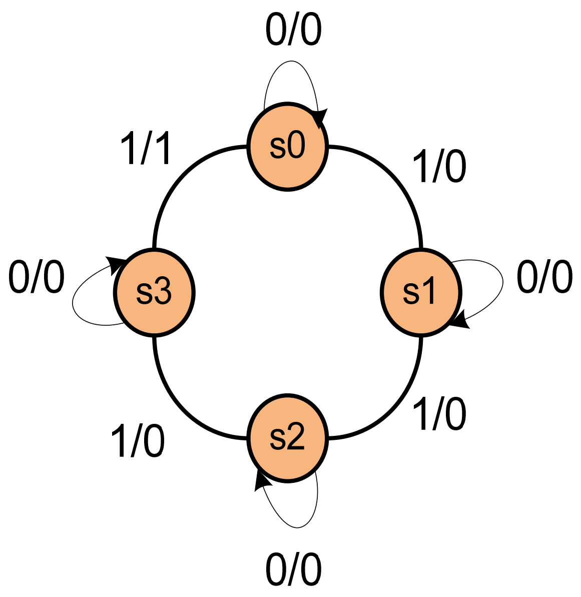


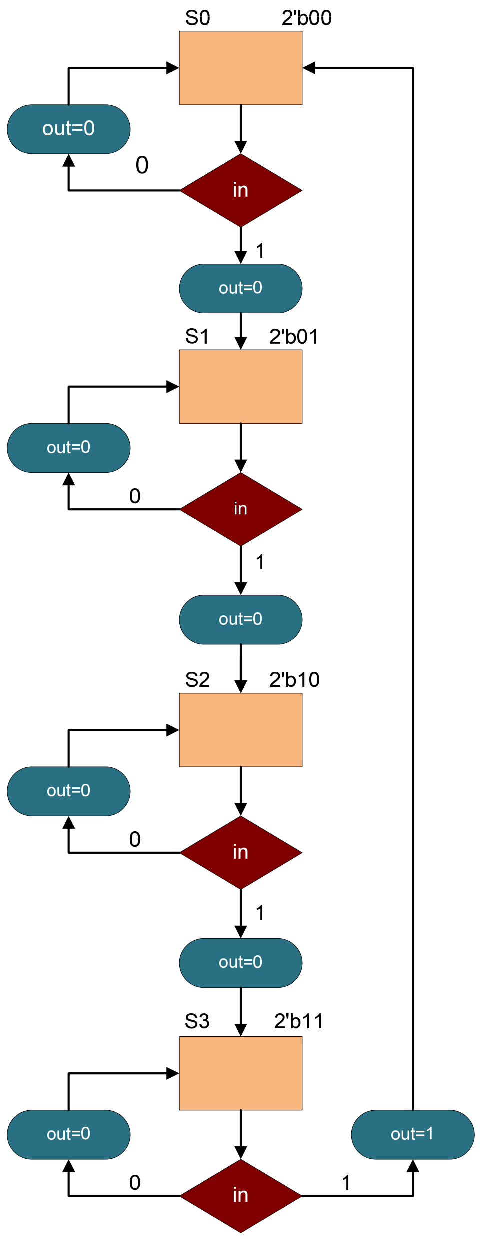
State Diagram

- Generating 1 at the output after counting four number of 1’s on a serial interface



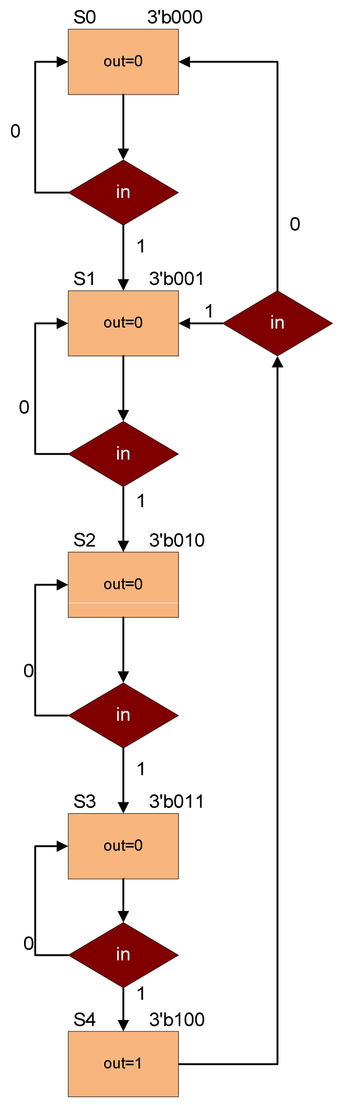
Example: Mealy Machine- State Diagram, ASM Representation





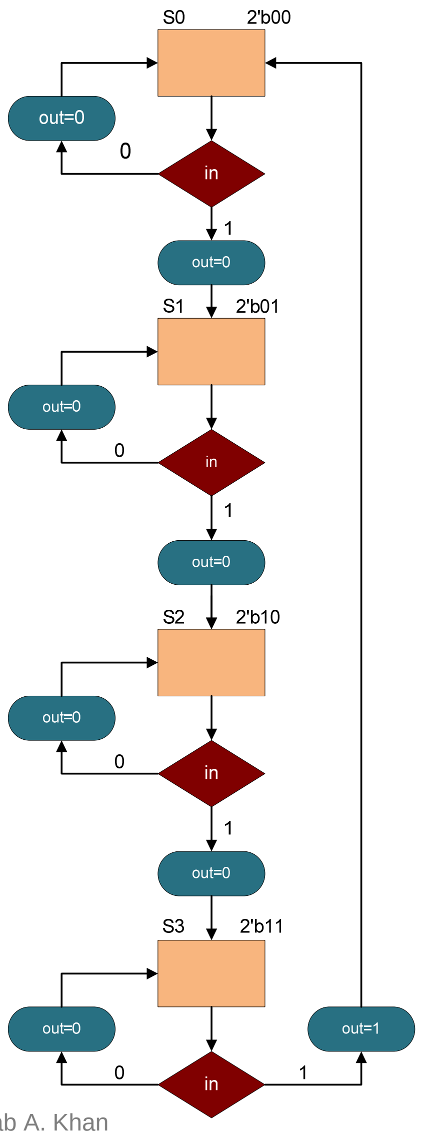
Example: Mealy Machine- State Diagram, ASM Representation

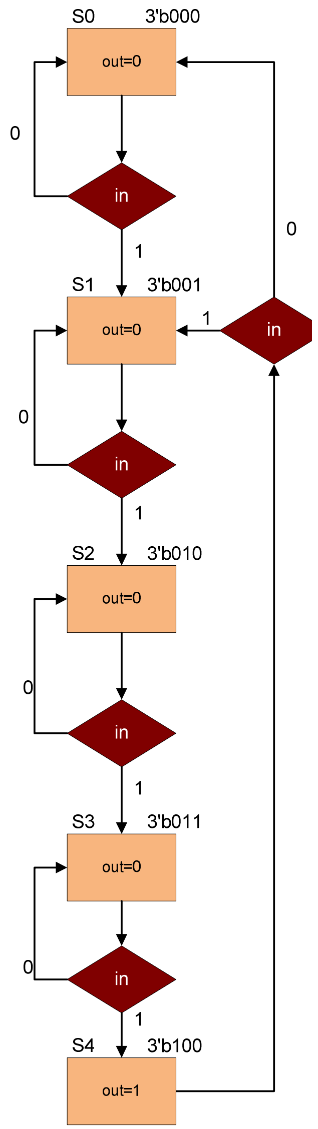
- No oval, means no conditional output
- More states
- Stable output for one complete clock cycle



Example: Mealy Machine- State Diagram, ASM Representation

- Mealy-Moore side by side
- Design independent
- Simple tradeoff



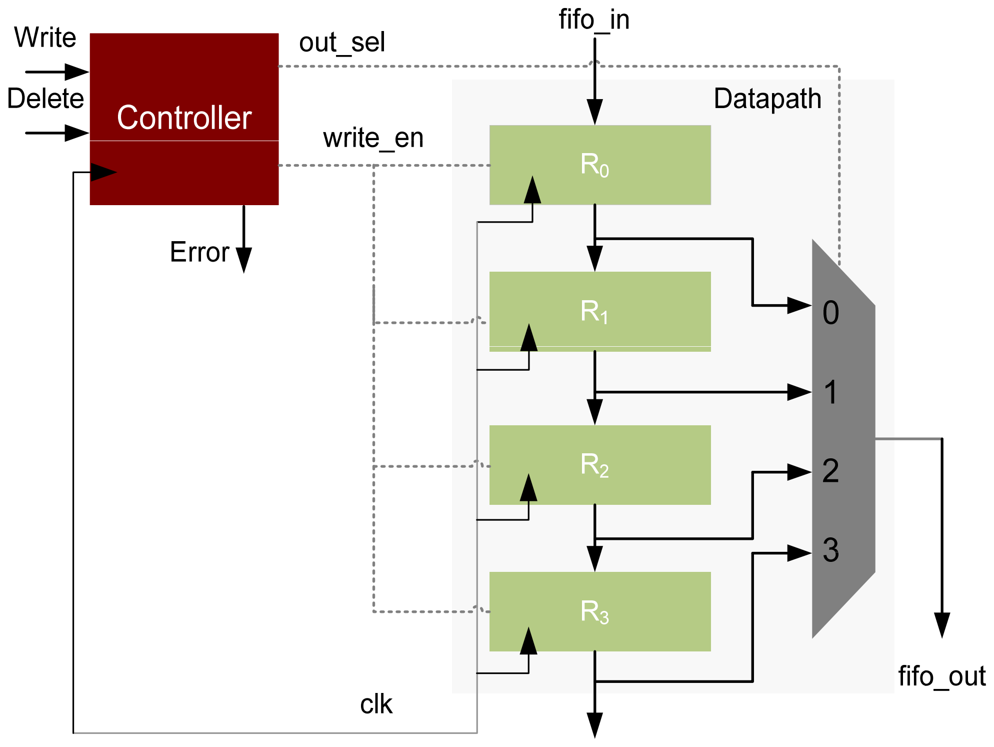


Design Example: 4 Entry FIFO

1. Design a first-in, first-out (FIFO) queue that consists of four registers R0, R1, R2, and R3
2. Write and Delete are the two operations on the queue
3. Write moves data from the fifo_in to R0 that is the tail of the queue
4. Delete deletes the first entry at the head of the queue
5. The head of the queue is available on the fifo_out
6. Writing into a full queue or deletion from an empty queue causes an ERROR condition
7. Assertion of Write and Delete at the same time also causes an ERROR condition

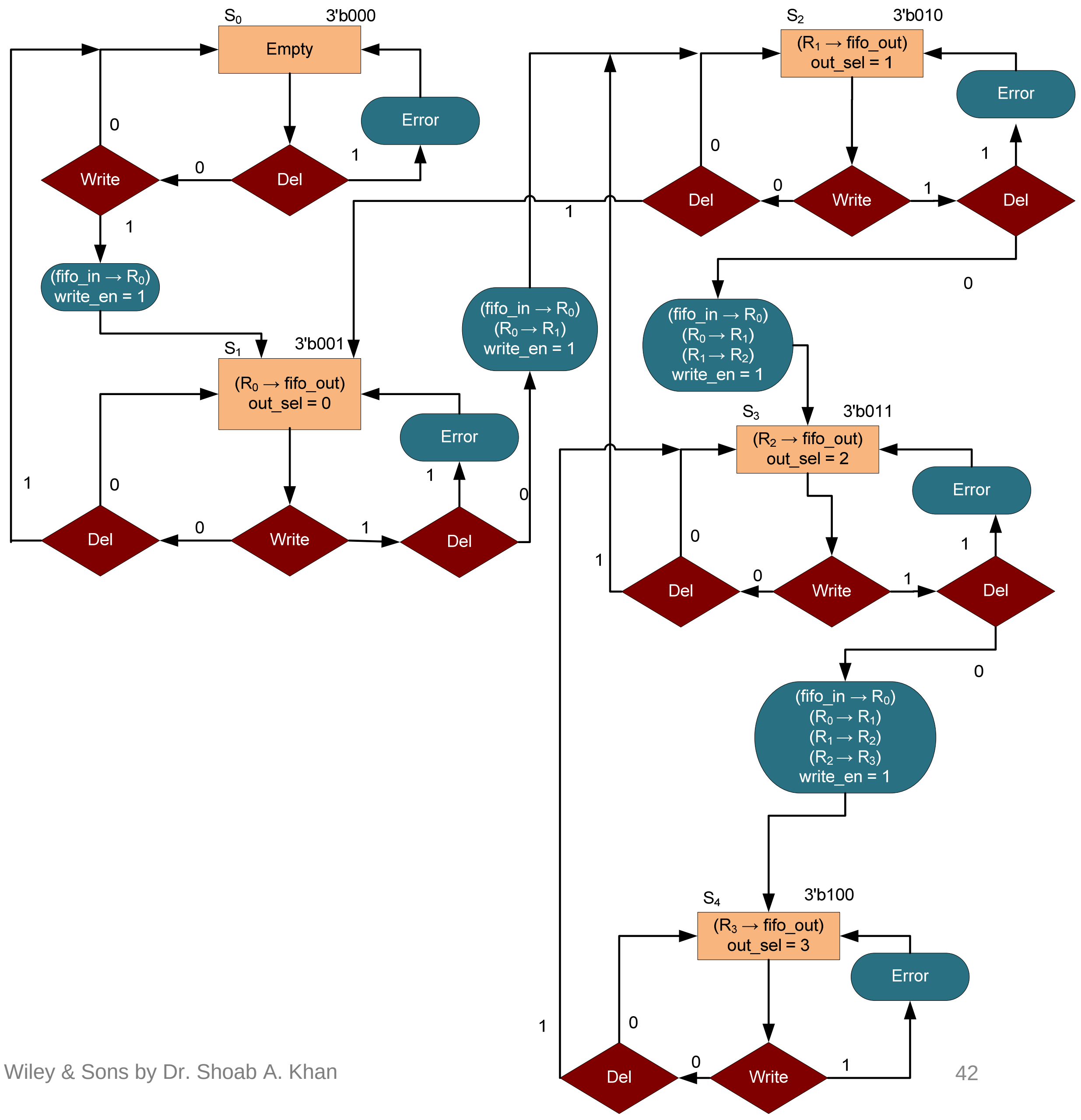
FIFO Design

- Design consists of datapath and controller
- Datapath
  - Four registers
    - Shift registers
  - One 4:1 MUX
- Controller
  - Input signals
    - Write
    - Delete
  - Output signals
    - out_sel
    - write_en
    - Error



Controller

- FSM based design
  - Mealy machine
- Five states
  - Idle
  - One entry
  - Two entries
  - Three entries
  - Full



S0 3'b000
Empty
Error
Write
Del
0
1
(fifo_in → R0) write_en = 1
S1 3'b001
(R0 → fifo_out) out_sel = 0
Error
(fifo_in → R0) (R0 → R1) write_en = 1
S2 3'b010
(R1 → fifo_out) out_sel = 1
Error
(fifo_in → R0) (R0 → R1) (R1 → R2) write_en = 1
S3 3'b011
(R2 → fifo_out) out_sel = 2
Error
(fifo_in → R0) (R0 → R1) (R1 → R2) (R2 → R3) write_en = 1
S4 3'b100
(R3 → fifo_out) out_sel = 3
Error

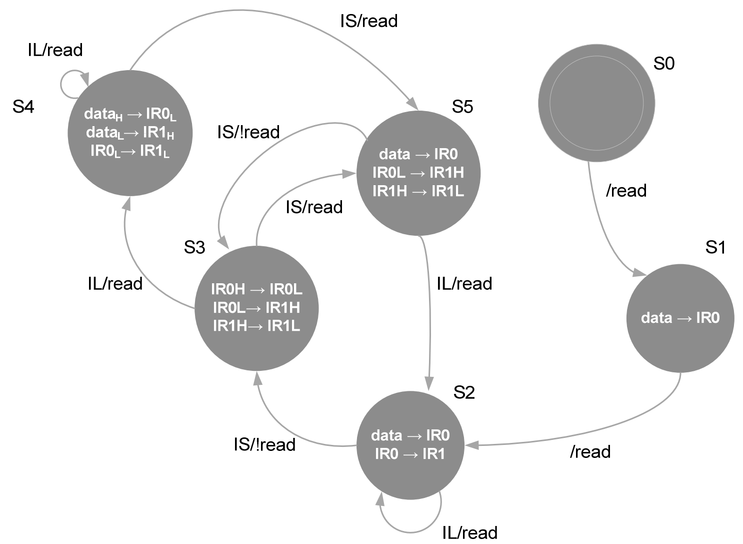
```
// Combinational part only for S0 and default state is given
always @(*)
begin
     next_state=0;
     case(current_state)

     S0:
     begin
          if(!Del&& Write)
          begin
               next_state = S1;
               write_en = 1’b1;
               Error= 1’b0;
               out_sel = 0;
          end
          else if(Del)
          begin
               next_state=S0;
               write_en =1’b0;
               Error = 1’b1;
               out_sel=0;
          end
          else
          begin
               next_state=S0;
               write_en=1’b0;
               out_sel = 1’b0;
          end
     // Similarly, rest of the states are coded //
          default:
          begin
               next_state=S0;
               write en = 1’b0;
               Error = 1’b0;
               out_sel =0;
          end
     endcase
end

// Sequential part
always @(posedge clk or negedge rst_n)
if(!rst_n)
     current_sate <= #1 S0;
else
     current_state <= #1 next_sate;
```

Design: Variable Length Instruction Dispatcher

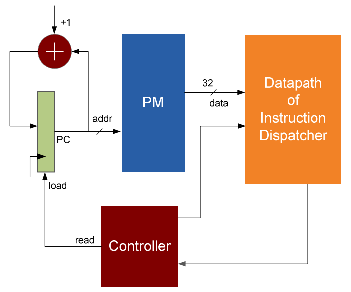
- Instruction dispatcher read 32-bit words from PM
- The instructions words are written
- into two 32-bit registers, IR0 and IR1
- The processor supports short and long instructions of lengths 16 and 32-bit
  - The LSB of the instruction is coded to specify the instruction type.
  - A 0 in the LSB indicates a 16-bit instruction and a 1 depicts the instruction is 32-bit wide

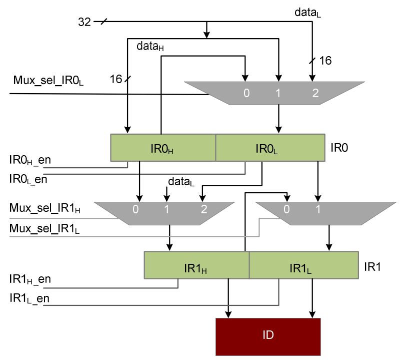
Design

- Datapath and controller
- Controller is FSM based
- Datapath is controller by signals from FSM



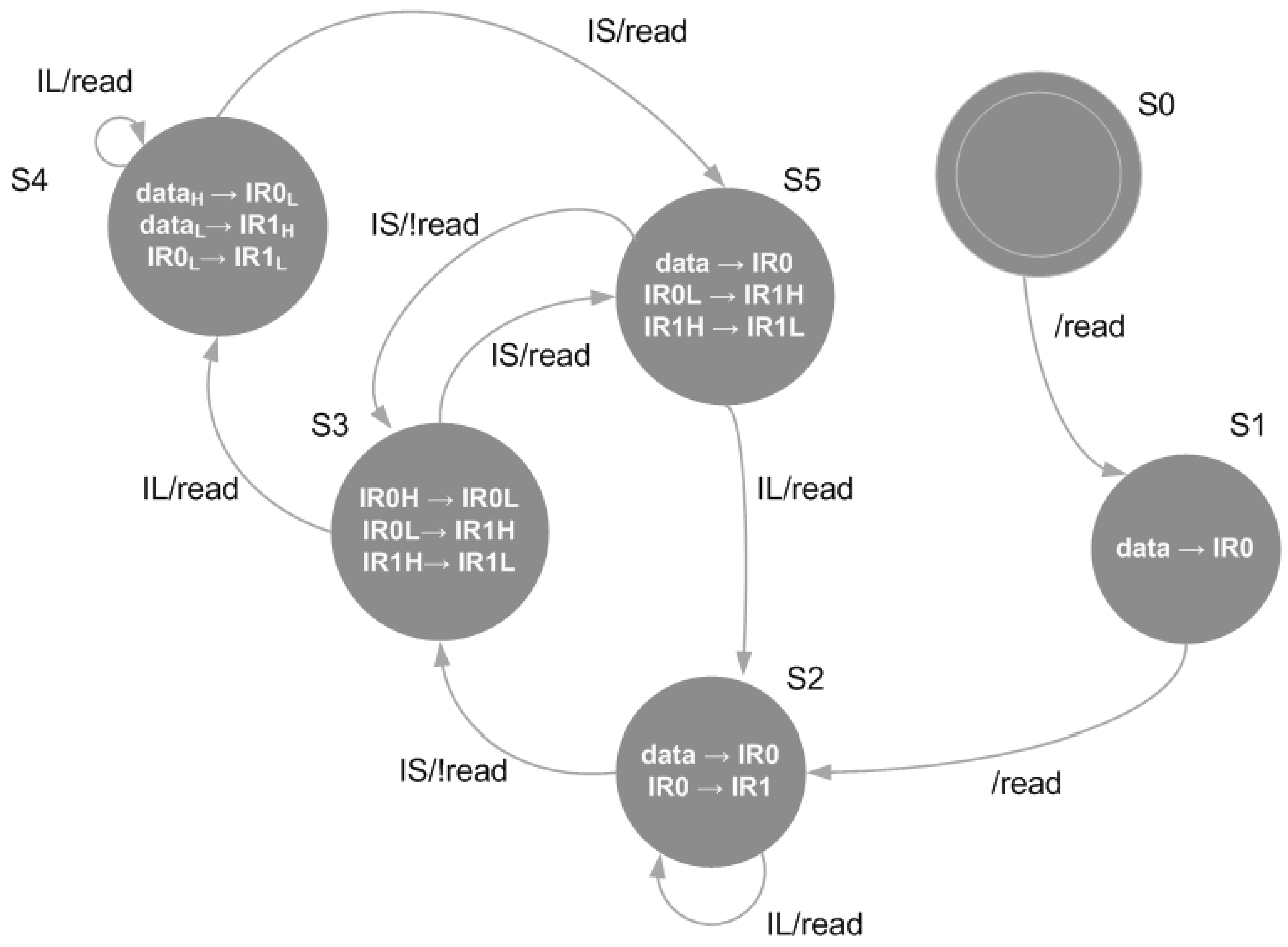
(a)



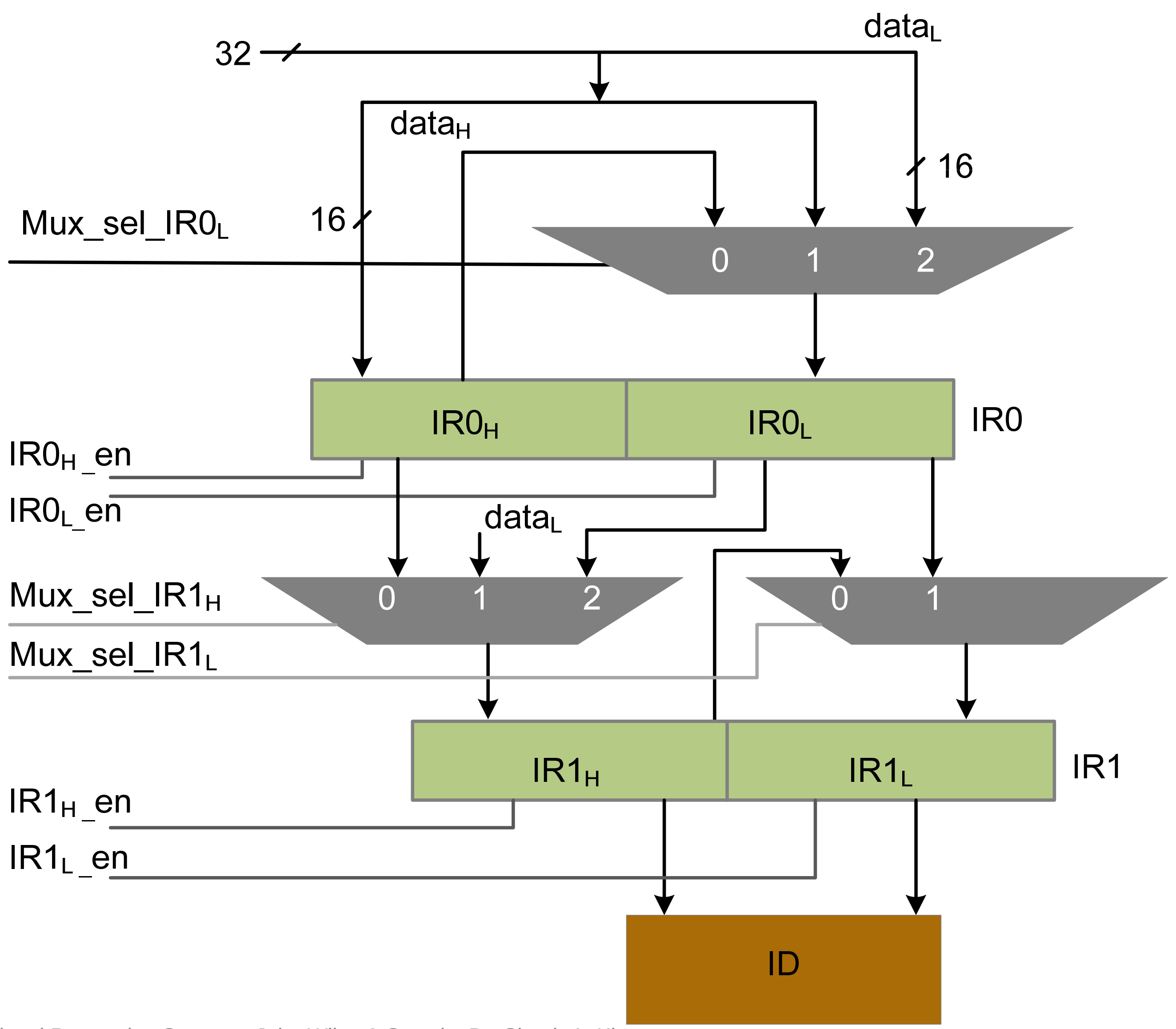


(b)

(c)


IS/read
IL/read
S0
S4
$data_H \rightarrow IR0_L$
$data_L \rightarrow IR1_H$
$IR0_L \rightarrow IR1_L$
S5
IS/!read
data → IR0
IR0L → IR1H
IR1H → IR1L
/read
IS/read
S3
S1
IL/read
IR0H → IR0L
IR0L → IR1H
IR1H → IR1L
IL/read
data → IR0
S2
IS/!read
data → IR0
IR0 → IR1
/read
IL/read

Datapath

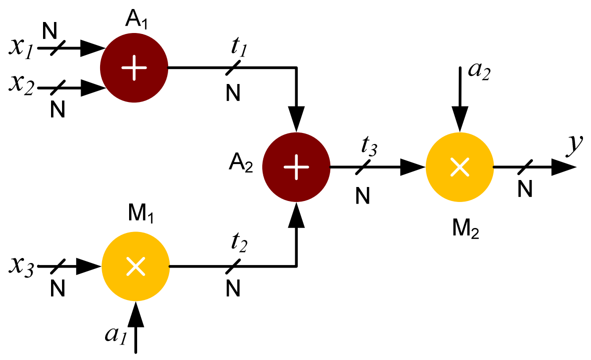

32
data$_L$
data$_H$
16
16
Mux_sel_IR0$_L$
0
1
2
IR0$_H$
IR0$_L$
IR0
IR0$_H$_en
IR0$_L$_en
data$_L$
Mux_sel_IR1$_H$
Mux_sel_IR1$_L$
0
1
2
0
1
IR1$_H$
IR1$_L$
IR1
IR1$_H$_en
IR1$_L$_en
ID

Adv dsd

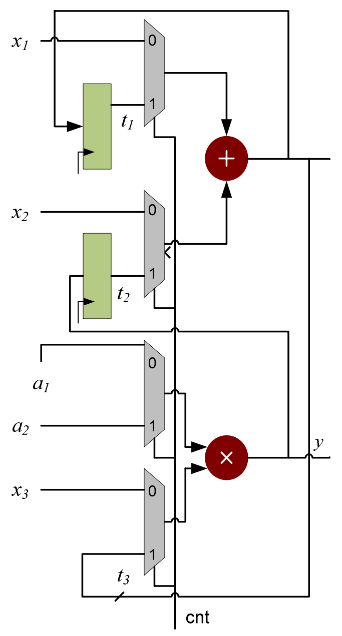
Folding Transformation

Folding Regular Structured DFGs

- Folding by a factor of 2
- Use only one adder and one multiplier
- Mathematical transformation can also be used

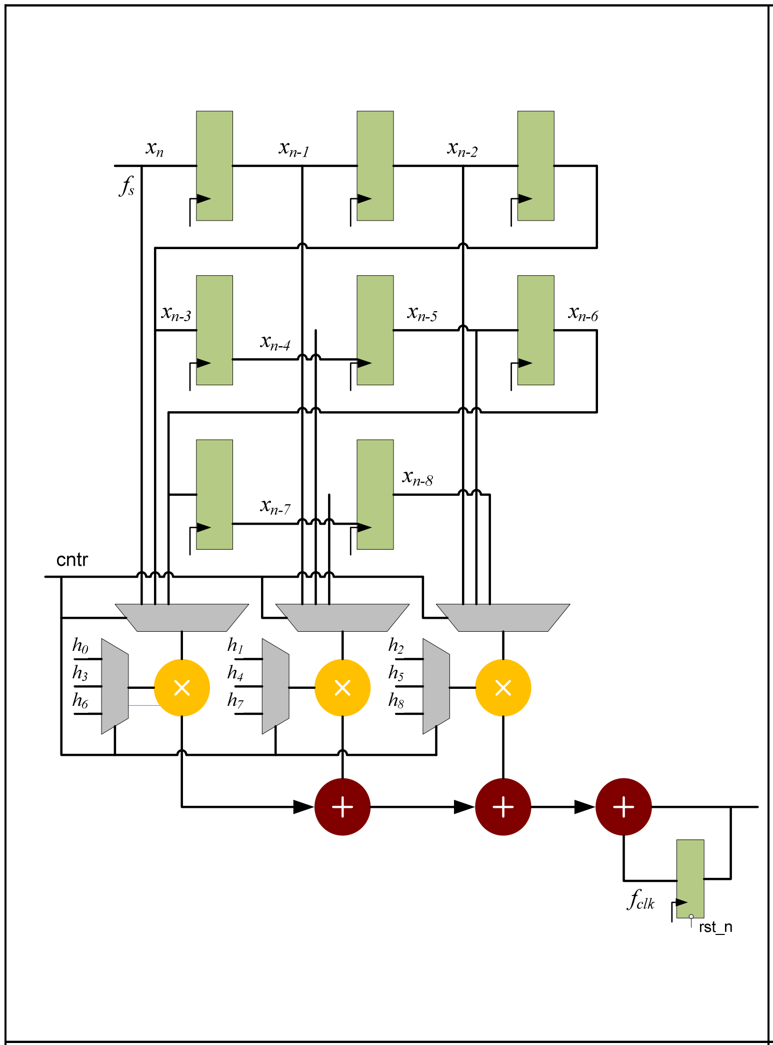


(a)

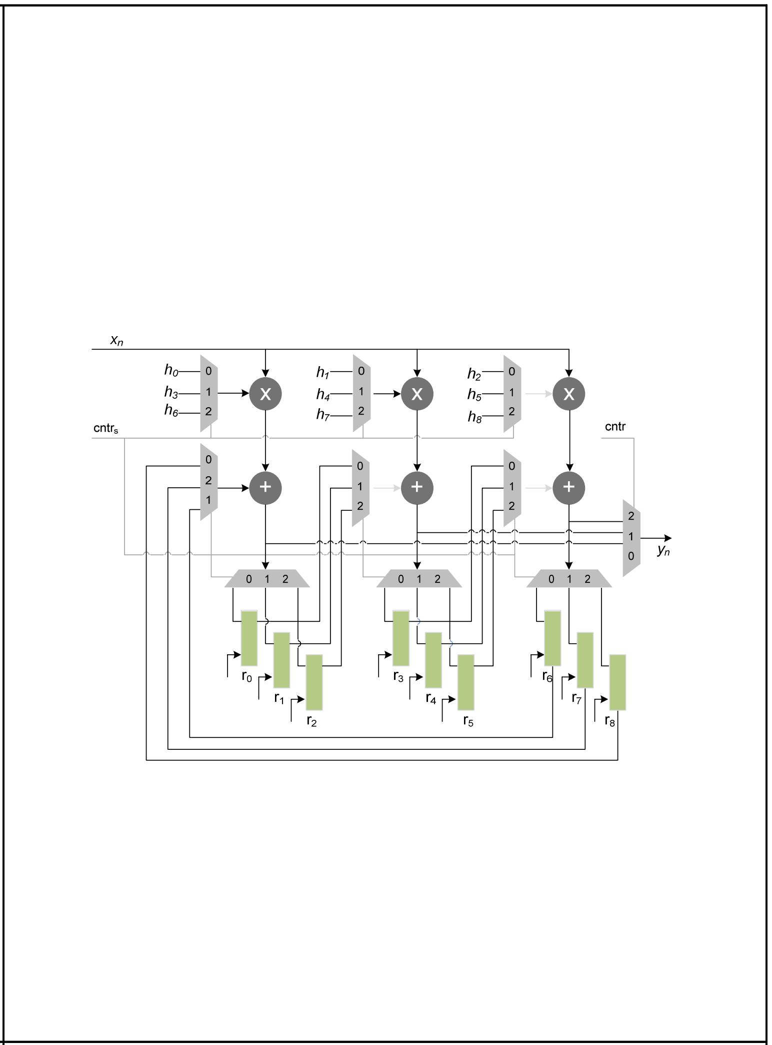


(b)

- Folded-by-3 architecture for a 9-coefficient FIR filter. (a) Folded DF architecture. (b) Folded TDF architecture



(a)



(b)

Reference Slides

REFERENCE SLIDES

A SYSTOLIC FFT ARCHITECTURE FOR REAL TIME FPGA SYSTEMS

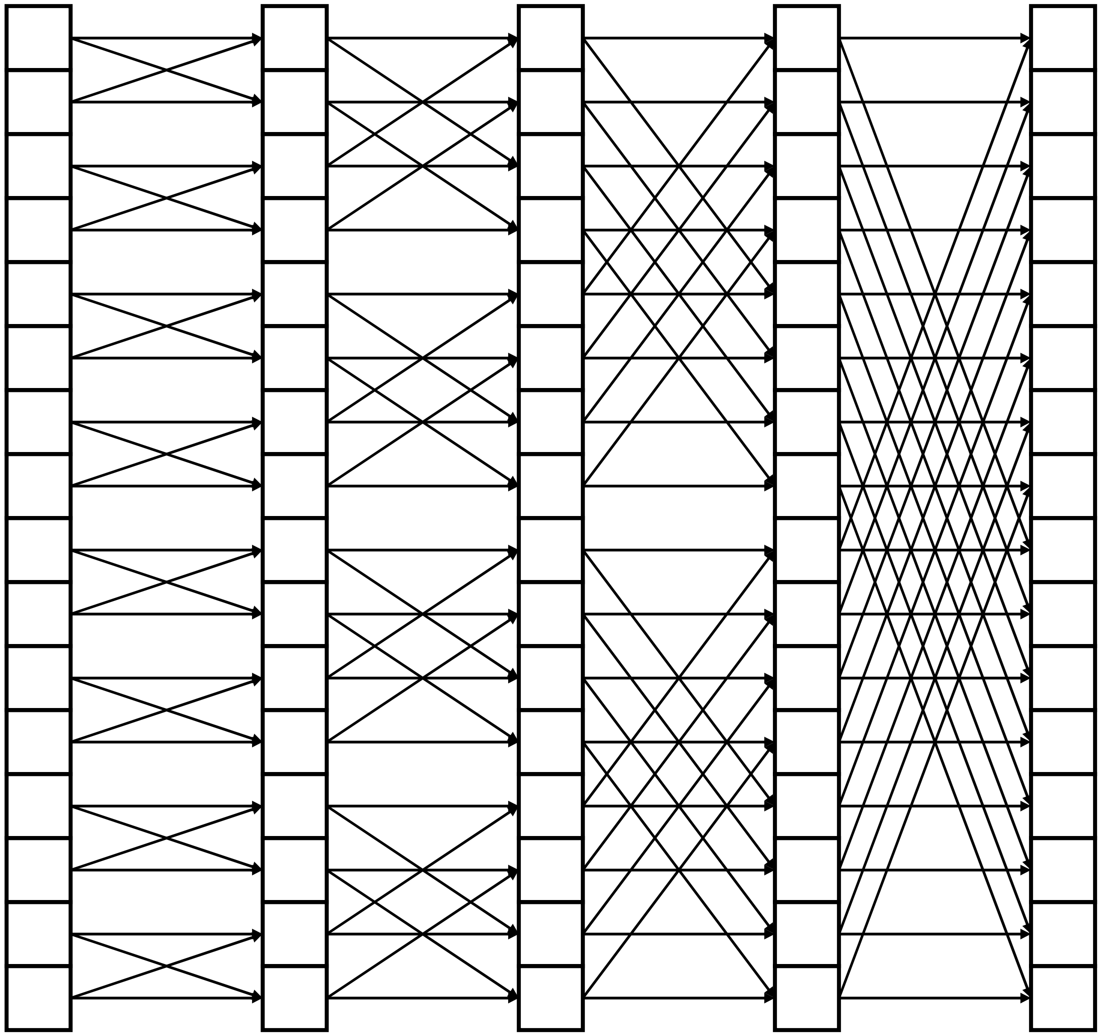
This work was sponsored by DARPA ATO under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the Department of Defense.

Preston Jackson, Cy Chan, Charles Rader, Jonathan Scalera, and Michael Vai

HPEC 2004

29 September 2004

Fully Dedicated Parallel Architecture



Size	16	8192	Δ
Pins	448	229K	
Fly	32	53K	
Mult			
Add			
Shift	0	0	

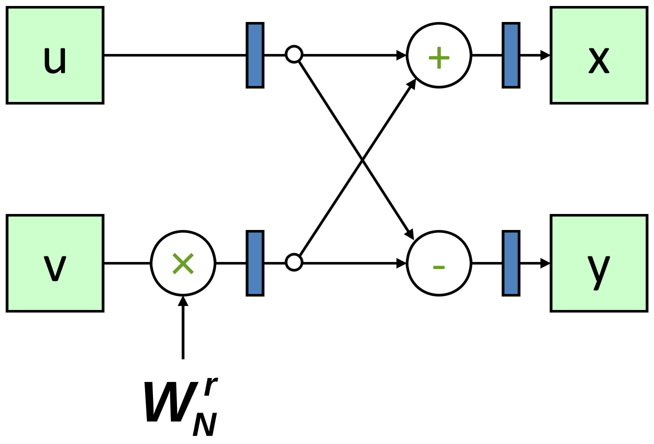
Parallel FFT

- Butterfly structure
- Removes redundant calculation

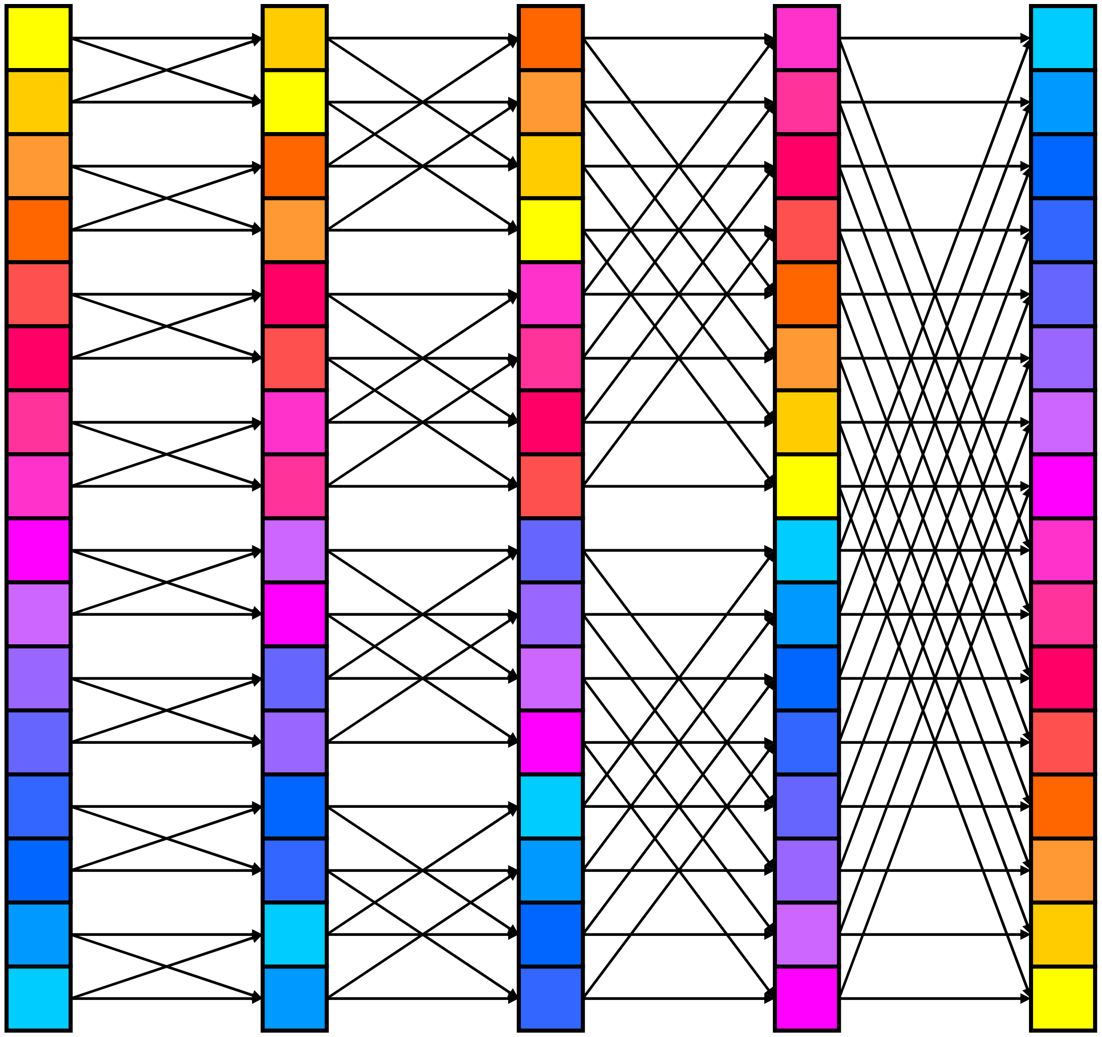
Complex Butterfly

- Butterfly contains
  - 1 complex addition
  - 1 complex subtraction
  - 1 complex, constant multiply

Size	16	8192	Δ
Pins	448	229K	
Fly	32	53K	
Mult			
Add			
Shift	0	0	



Parallel-Pipelined Architecture

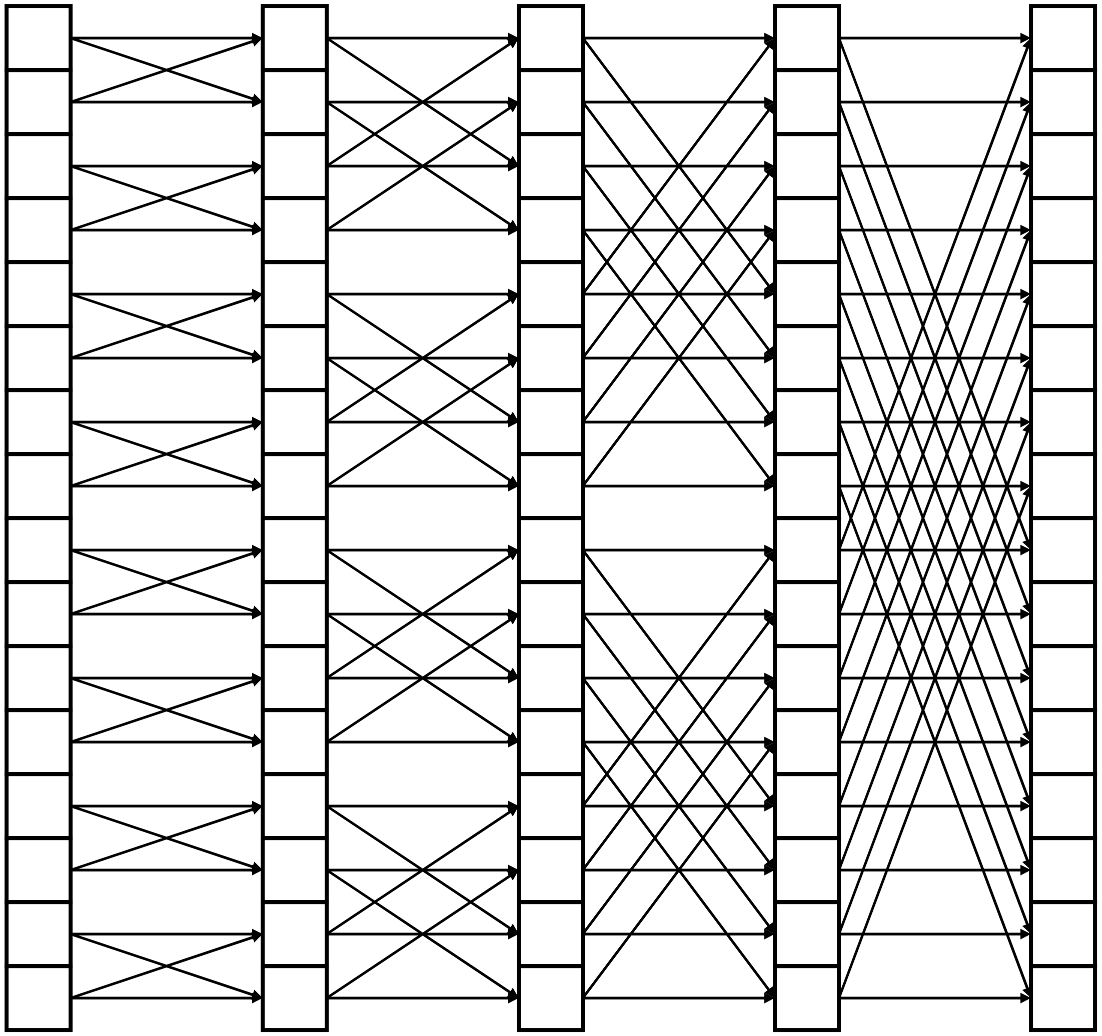


Size	16	8192	Δ
Pins	448	229K	
Fly	32	53K	
Mult	96	159K	
Add	288	480K	
Shift	0	0	

A pipelined version

- IO Bound
- 100% Efficient

Self Timed with Serial Input



Size	16	8192	Δ
Pins	28	28	.01%
Fly	32	53K	
Mult	96	159K	
Add	288	480K	
Shift	0	0	

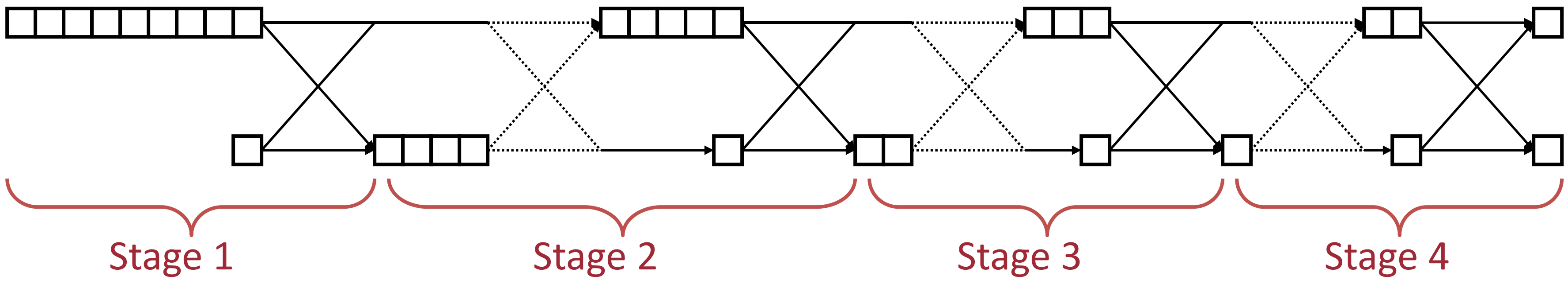
A serial version

- IO-rate matches A/D
- 6.25% Efficient

Serial Architecture

- The parallel architecture can be collapsed
  - One butterfly per stage
  - Consumes 1 sample per cycle
  - Same latency and throughput
  - More efficient design

Size	16	8192	Δ
Pins	28	28	
Fly	4	13	.03%
Mult	12	39	.03%
Add	36	117	.03%
Shift	22	12K	

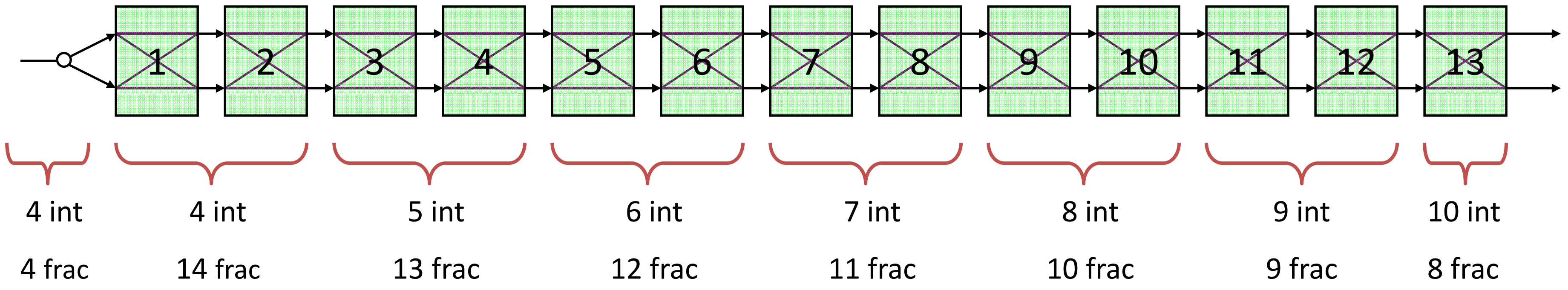


50% Efficiency

8192-Point Architecture

- Requires 13 stages
- Fixed point arithmetic
- Varies the dynamic range to increase accuracy
- Overflow replaced with saturated value

Size	16	8192	Δ
Pins	28	28	
Fly	4	13	
Mult	12	39	
Add	36	117	
Shift	22	12K	



- Multipliers limit design to 18-bits and 150 MHz
- Achieves 70 dB of accuracy

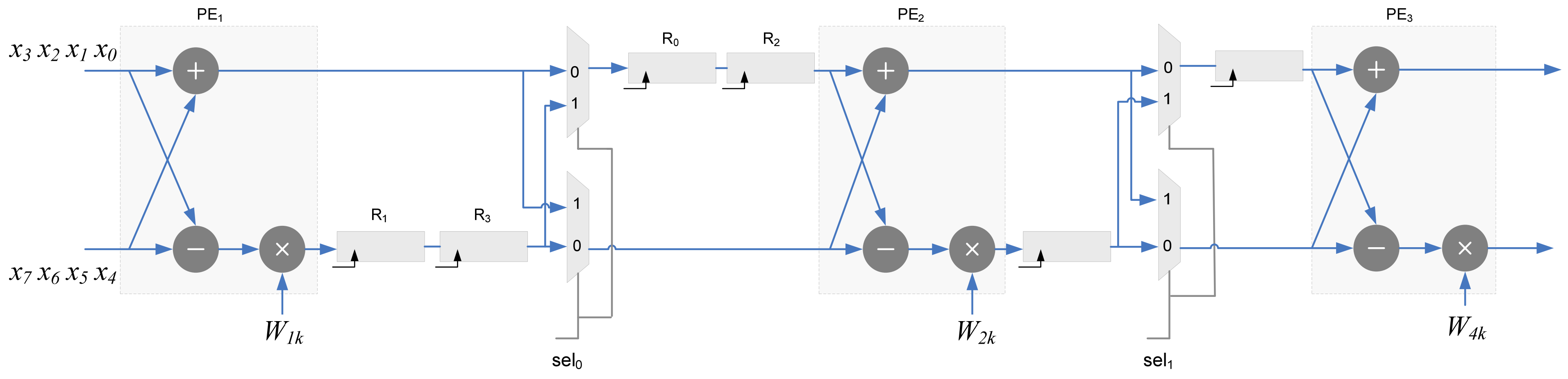
$$\underbrace{0110}.\underbrace{0101}$$

$$6 + \frac{5}{16}$$

FFT Systolic design

From the Book

Systolic Folded Architecture



- 8-point FFT
- Dual port memory with two memory read
- Folded by an order of 4
  - Horizontal folding
- Systolic architecture
  - Data flows across the architecture in systolic fashion
- 100% utilization

	PE$_1$	**R0**	**R2**	**PE$_2$**	**R4**	**PE$_3$**
		R1	**R3**		**R5**	
$\square_3\ \square_2\ \square_1\ \square_0$	$y_{10} = x_0 + x_4$					
$\square_7\ \square_6\ \square_5\ \square_4$	$y_{14} = (x_0 - x_4)\ W_8^0$					
$\square_0\ \square_3\ \square_2\ \square_1$	$y_{11} = x_1 + x_5$	y_{10}				
$\square_4\ \square_7\ \square_6\ \square_5$	$y_{15} = (x_1 - x_5)\ W_8^1$	y_{14}				
$\square_1\ \square_0\ \square_3\ \square_2$	$y_{12} = x_2 + x_6$	y_{11}	y_{10}	$y_{20} = y_{10} + y_{12}$		
$\square_5\ \square_4\ \square_7\ \square_6$	$y_{16} = (x_2 - x_6)\ W_8^2$	y_{15}	y_{14}	$y_{22} = (y_{10} - y_{12})\ W_8^0$		
$\square_2\ \square_1\ \square_0\ \square_3$	$y_{13} = x_3 + x_7$	y_{14}	y_{11}	$y_{21} = y_{11} + y_{13}$	y_{20}	$x(0) = y_{20} + y_{21}$
$\square_7\ \square_6\ \square_5\ \square_7$	$y_{17} = (x_3 - x_7)\ W_8^3$	y_{16}	y_{15}	$y_{23} = (y_{11} - y_{13})\ W_8^2$	y_{22}	$x(4) = (y_{20} - y_{21})\ W_8^0$
		y_{15}	y_{14}	$y_{24} = y_{14} + y_{16}$	y_{22}	$x(2) = y_{22} + y_{23}$
		y_{17}	y_{16}	$y_{26} = (y_{14} - y_{16})\ W_8^0$	y_{23}	$x(6) = (y_{22} - y_{23})\ W_8^0$
			y_{15}	$y_{25} = y_{15} + y_{17}$	y_{24}	$x(1) = y_{24} + y_{25}$
			y_{17}	$y_{27} = (y_{15} - y_{17})\ W_8^2$	y_{26}	$x(5) = (y_{24} - y_{25})\ W_8^0$
					y_{26}	$x(3) = y_{26} + y_{27}$
					y_{27}	$x(7) = (y_{26} - y_{27})\ W_8^0$

Questions/Feedback