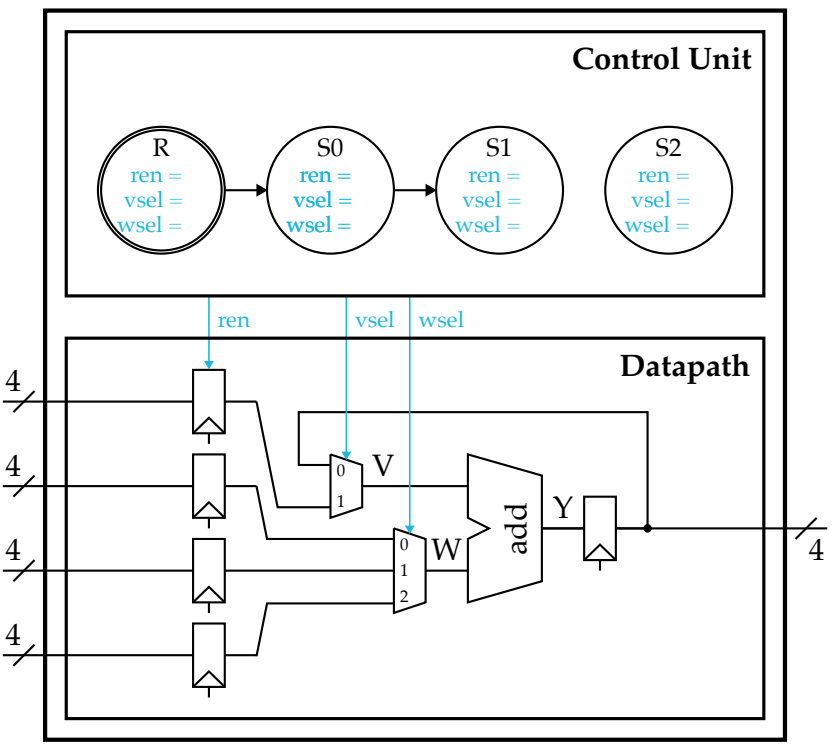
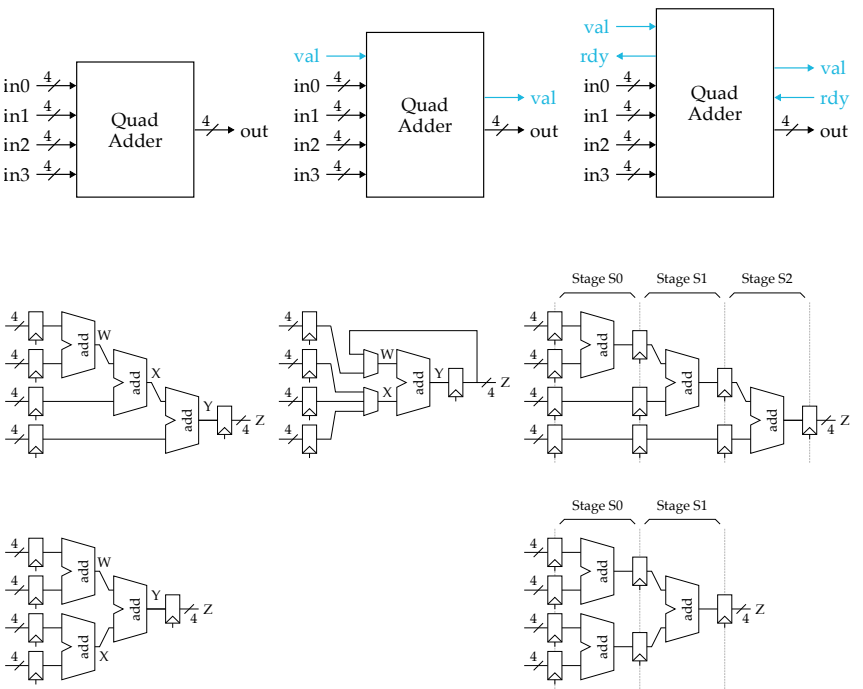
5.5. Control/Datapath Decomposition

- Improve modularity by separating control logic from datapath logic
  - Control unit determines *when* operations happen
  - Datapath determines *how* operations happen
- Multi-cycle adder uses FSM for register enables and mux selects
  - Control unit will contain the FSM
  - Datapath will contain the registers, muxes, and adder

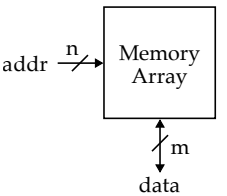


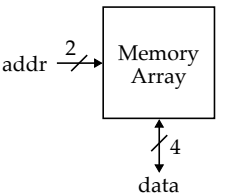
5.6. Latency-Insensitive Interfaces

- Improve modularity by using a latency-insensitive interface
  - Valid signal indicates if the producer has valid data to send that cycle
  - Ready signal indicates if the consumer is ready to accept data that cycle
  - At end of cycle, both producer and consumer AND valid with ready to determine if data was actually sent
- Single-cycle, multi-cycle, pipelined, and tree quad adders all have different latencies and occupancies

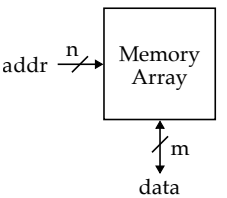


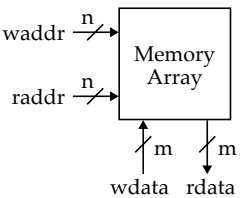
6. Memory Arrays

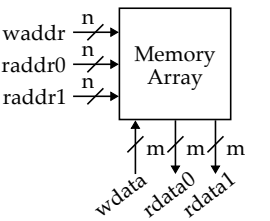


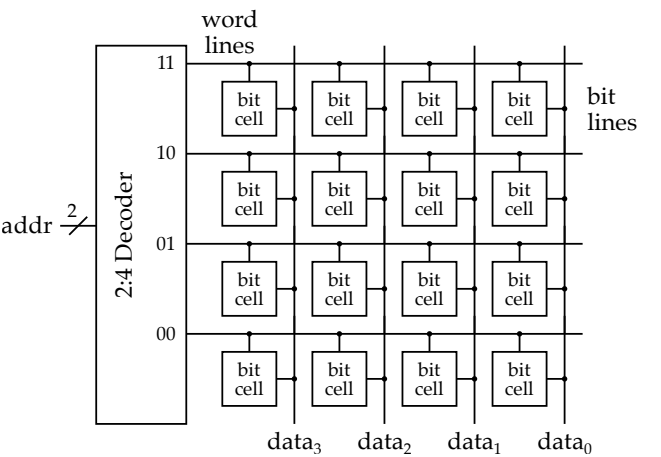


addr	data			
11	0	1	1	0
10	0	0	1	1
01	1	1	0	0
00	1	0	0	0

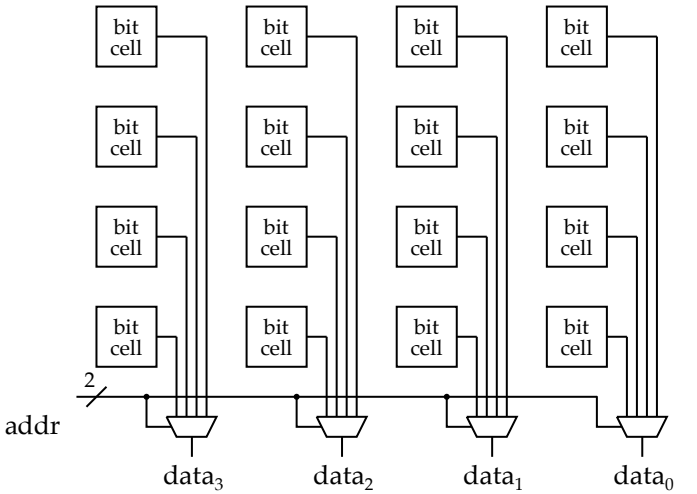




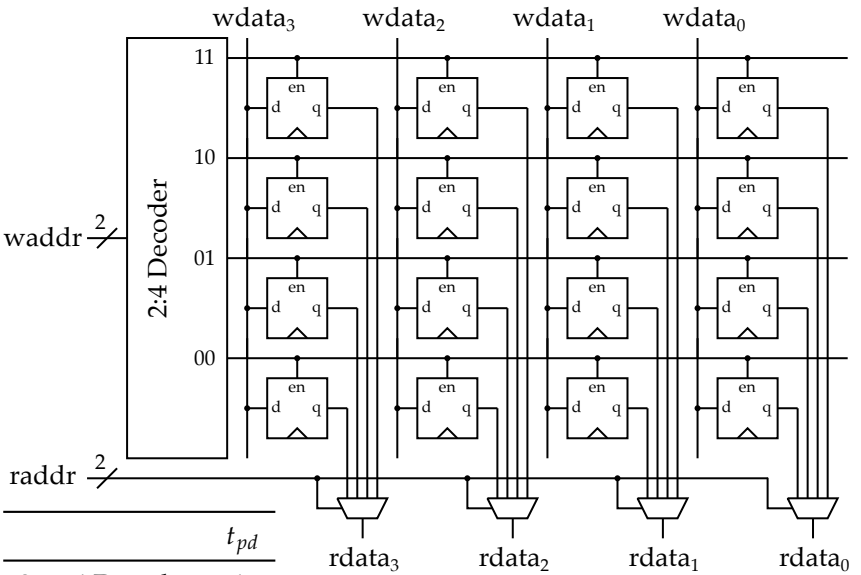




6.1. Read-Only Memories



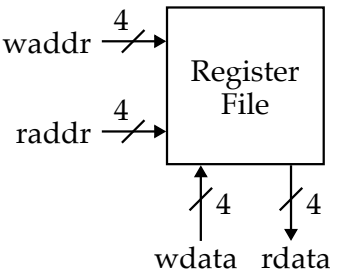
6.2. Register Files



	t_{pd}
2-to-4 Decoder	4τ
1-bit 4-to-1 Mux	10τ
FF (t_{cq})	9τ
FF ($t_{d,setup}$)	10τ
FF ($t_{en,setup}$)	8τ

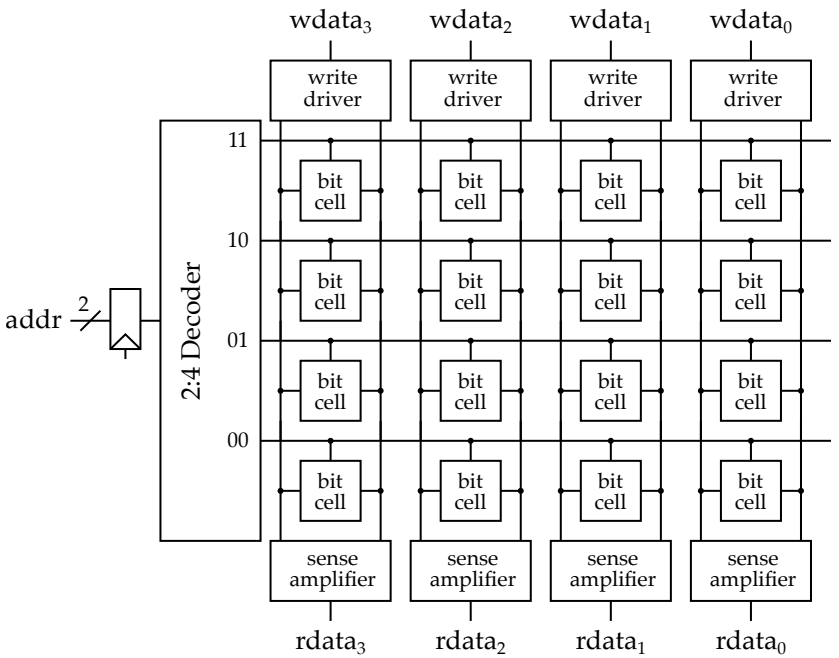
Path	**Propagation Delay**

- We can build a delay model of a register file and then use this model when analyzing more complex hardware designs

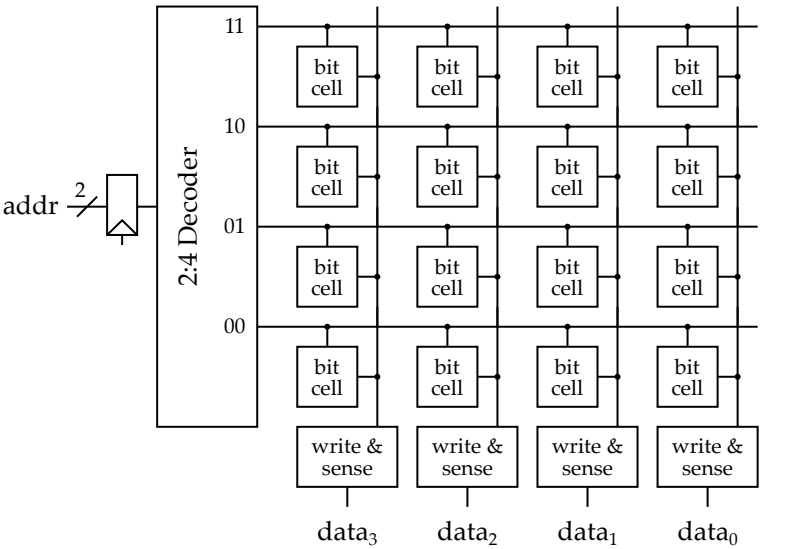


4×4b Reg File ($t_{pd,cq}$)
4×4b Reg File ($t_{pd,read}$)
4×4b Reg File (t_{setup})

6.3. Static Random Access Memories (SRAM)



6.4. Dynamic Random Access Memories (DRAM)



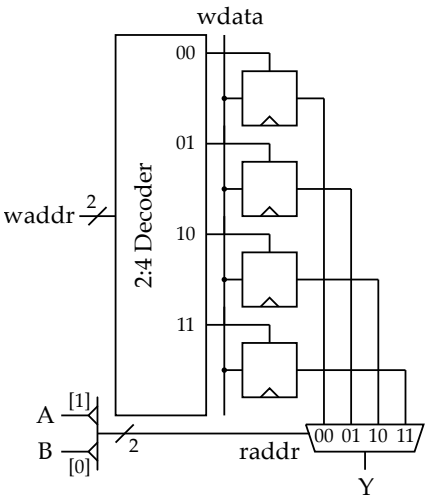
6.5. Comparing Memory Arrays

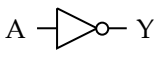
Memory Type	Transistors per Bit Cell	Density	Latency
Register File			
SRAM			
DRAM			

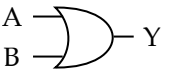
7. Programmable Logic

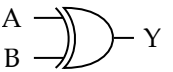
- Programmable logic uses sequential logic-gates to enable reconfiguring ("programming") the behavior of hardware after a chip has been fabricated

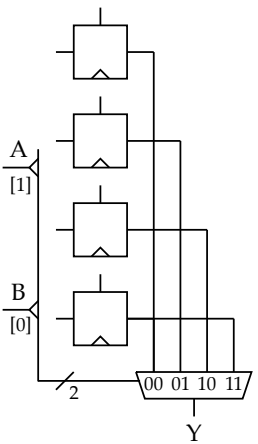
7.1. Programmable Truth Tables

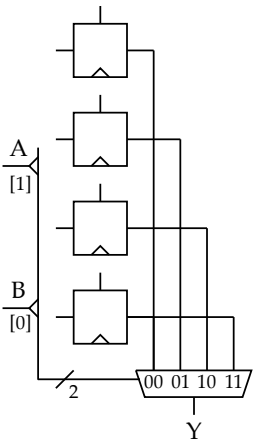


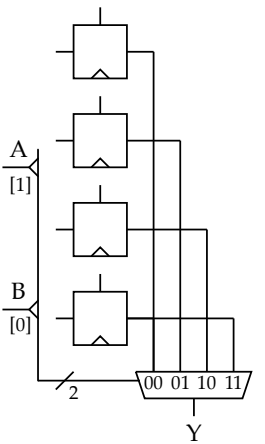

A
Y

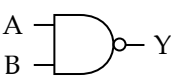

A
B
Y

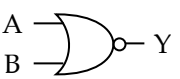

A
B
Y

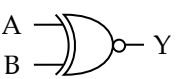

A
[1]
B
[0]
2
00 01 10 11
Y

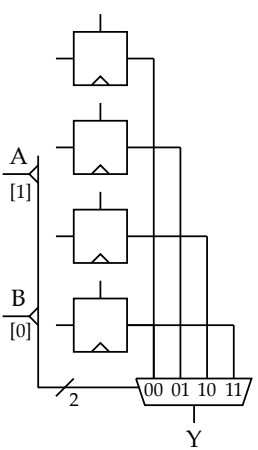

A
[1]
B
[0]
2
00 01 10 11
Y

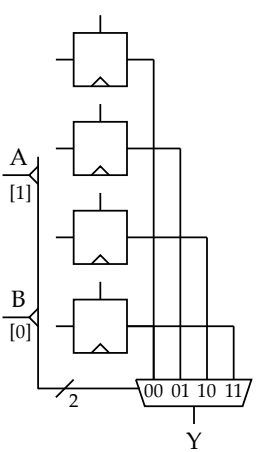

A
[1]
B
[0]
2
00 01 10 11
Y

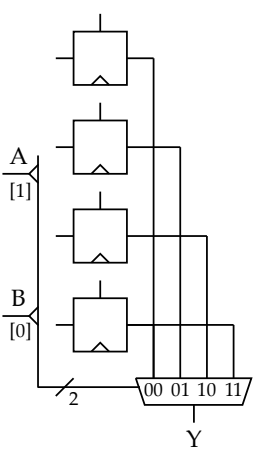

A
B
Y


A
B
Y

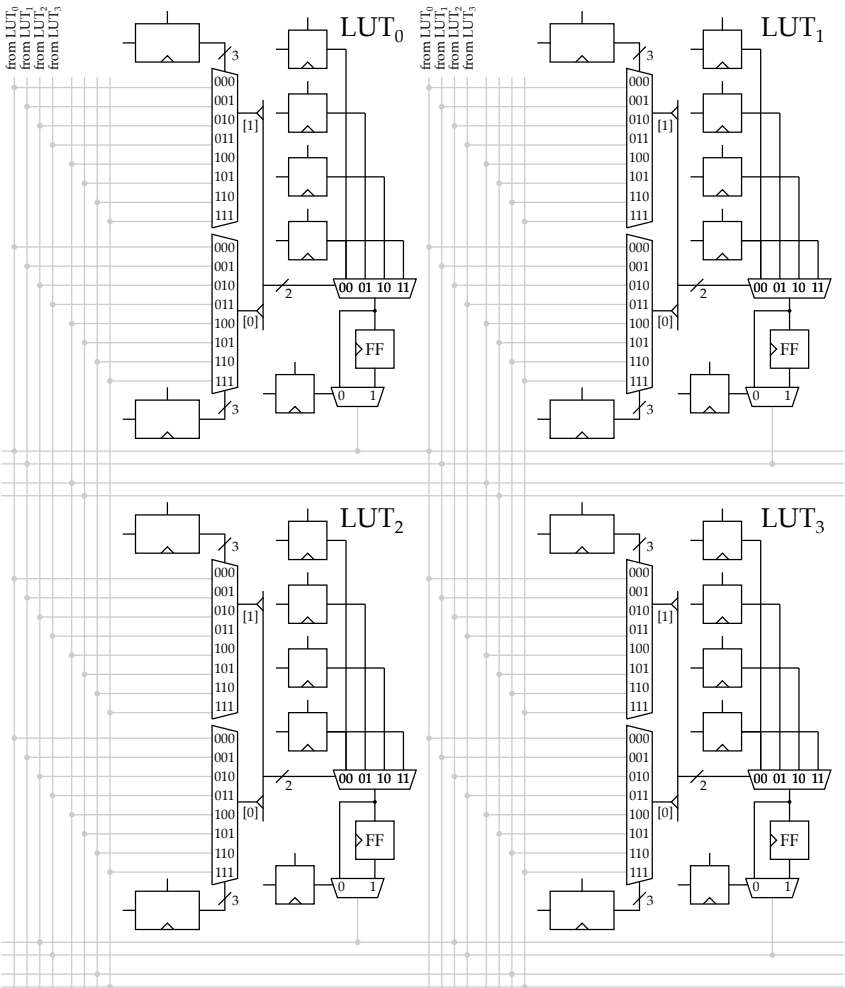

A
B
Y

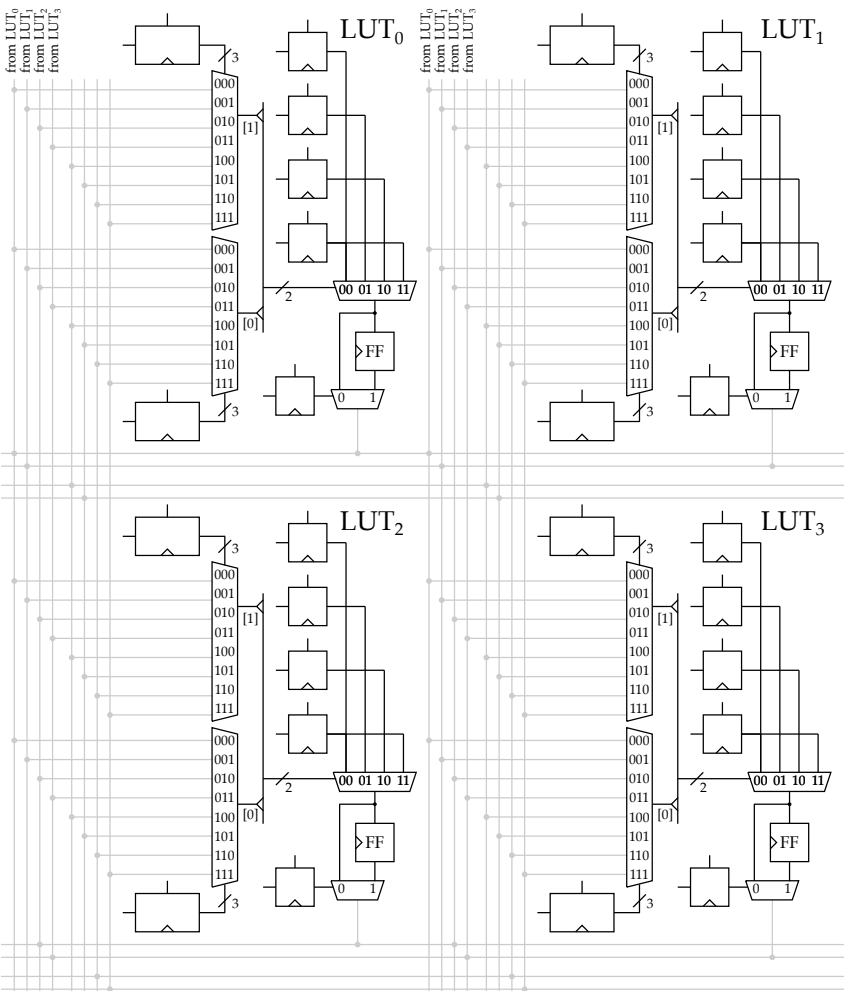

A
[1]
B
[0]
2
00 01 10 11
Y

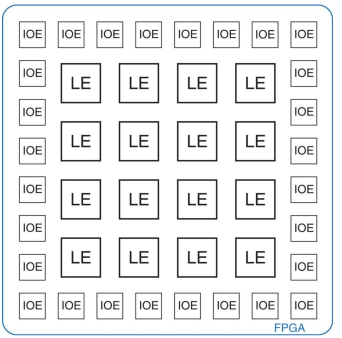

A
[1]
B
[0]
2
00 01 10 11
Y

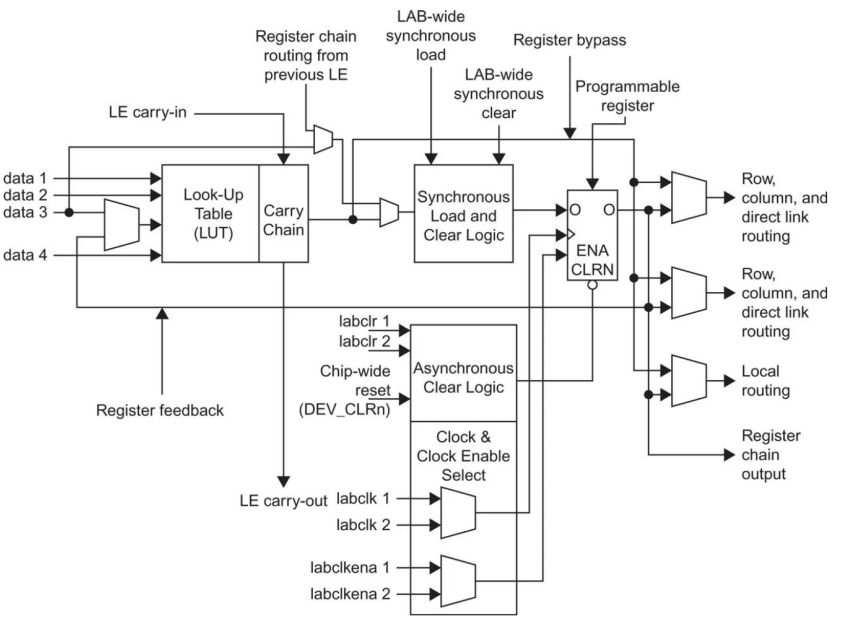

A
[1]
B
[0]
2
00 01 10 11
Y

7.2. Field Programmable Gate Arrays



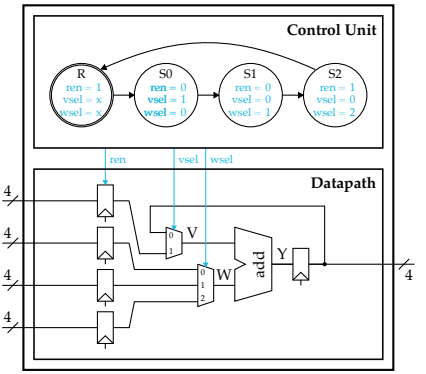

from LUT0
from LUT1
from LUT2
from LUT3
LUT0
LUT1
LUT2
LUT3
3
000
001
010
011
100
101
110
111
[1]
[0]
2
00 01 10 11
FF
0 1


IOE
LE
FPGA


Register chain routing from previous LE
LAB-wide synchronous load
Register bypass
LAB-wide synchronous clear
Programmable register
LE carry-in
data 1
data 2
data 3
data 4
Look-Up Table (LUT)
Carry Chain
Synchronous Load and Clear Logic
ENA
CLRN
Row, column, and direct link routing
Row, column, and direct link routing
Local routing
Register chain output
labclr 1
labclr 2
Chip-wide reset (DEV_CLRn)
Asynchronous Clear Logic
Clock & Clock Enable Select
Register feedback
LE carry-out
labclk 1
labclk 2
labclkena 1
labclkena 2

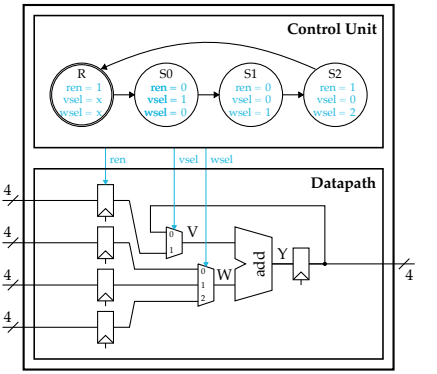
8. Verilog Modeling

```
1  module QuadAddMcycleDpath_RTL
2  (
3    input  logic        clk,
4    input  logic        rst,
5    input  logic [3:0] in0,
6    input  logic [3:0] in1,
7    input  logic [3:0] in2,
8    input  logic [3:0] in3,
9    output logic [3:0] out
10
11   // Control Signals
12   input  logic        ren,
13   input  logic        vsel,
14   input  logic        wsel
15 );
16
17   logic [3:0] in0reg_out;
18   Register_4b_RTL in0_reg
19   (
20     .clk (clk),
21     .rst (rst),
22     .en  (ren),
23     .d   (in0),
24     .q   (in0reg_out)
25   );
26
27   // Other input registers
28   // not shown ...
29
30   logic [3:0] yreg_out;
31   logic [3:0] vmux_out;
32   Mux2_4b_RTL vmux
33   (
34     .in0 (yreg_out),
35     .in1 (in0_reg_out),
36     .out (vmux_out)
37   );
```




```
38   logic [3:0] wmux_out;
39   Mux3_4b_RTL wmux
40   (
41     .in0 (in1reg_out),
42     .in1 (in2reg_out),
43     .in2 (in3reg_out),
44     .out (wmux_out)
45   );
46
47   logic [3:0] add_out;
48   Adder_4b_RTL add
49   (
50     .in0 (vmux_out),
51     .in1 (wmux_out),
52     .out (add_out)
53   );
54
55   Register_4b_RTL yreg
56   (
57     .clk (clk),
58     .rst (rst),
59     .en  (1'b1),
60     .d   (add_out),
61     .q   (out)
62   );
63
64 endmodule
```

```
1  module QuadAddCtrl_RTL
2  (
3    input  logic       clk,
4
5    // Control Signals
6    output logic       ren,
7    output logic       vsel,
8    output logic       wsel
9  );
10
11   localparam R  = 2'b00;
12   localparam S0 = 2'b01;
13   localparam S1 = 2'b10;
14   localparam S2 = 2'b11;
15
16   logic [1:0] state, state_next;
17   Register_2b_RTL state_reg
18   (
19     .clk (clk),
20     .rst (rst),
21     .d   (state_next),
22     .q   (state)
23   );
24
25   always_comb begin
26     case ( state )
27       R  : state_next = S0;
28       S0 : state_next = S1;
29       S1 : state_next = S2;
30       S2 : state_next = S0;
31       default : state_next = 'x;
32     endcase
33   end
```



```
module Regfile1r1w_4x4b_flat_RTL
(
  input  logic       clk,

  input  logic       wen,
  input  logic [1:0] waddr,
  input  logic [3:0] wdata,

  input  logic [1:0] raddr,
  output logic [3:0] rdata
);

  // State

  logic [3:0] mem [4];

  // Sequential write port

  always_ff @( posedge clk ) begin

    if ( wen )
      mem[waddr] <= wdata;
    else if ( !wen )
      mem[waddr] <= mem[waddr];
    else
      mem[waddr] <= 'x;

  end

  // Combinational read port

  assign rdata = mem[raddr];

endmodule
```

