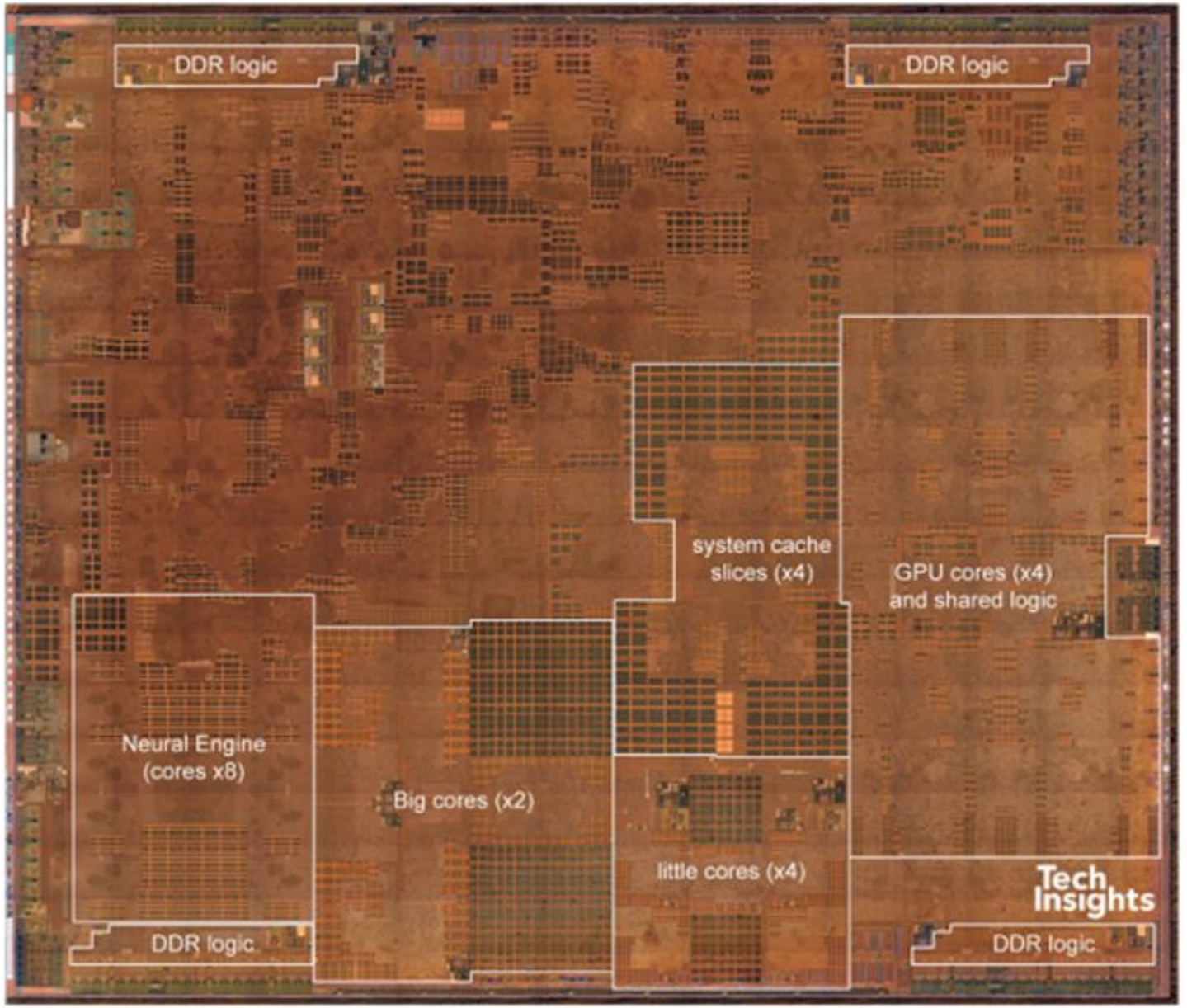
Digital Circuits and Systems
Lecture 4 Modeling Combinational Logic

Tian Sheuan Chang

SYSTEM VIEW

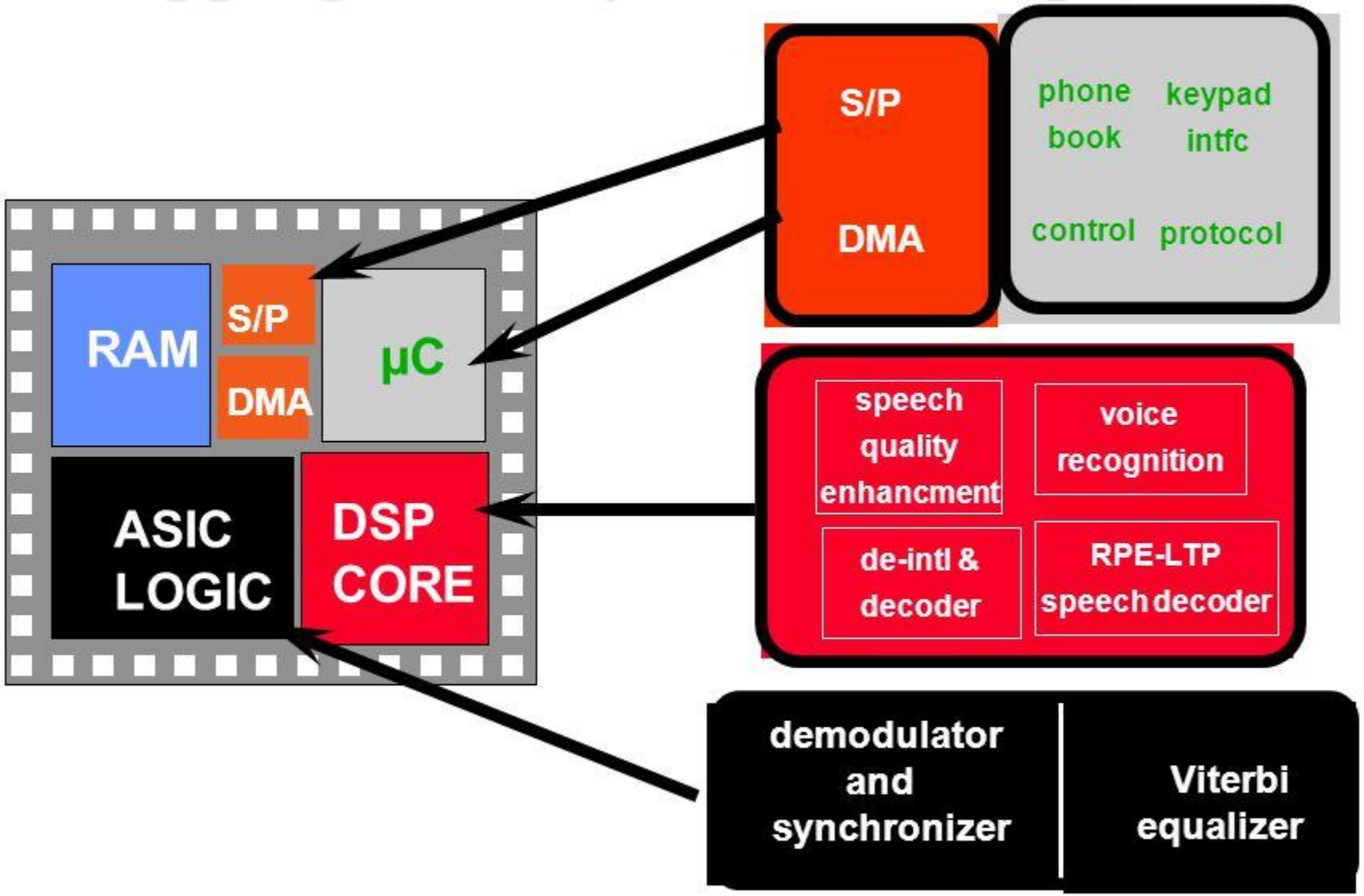
A12



https://www.phonearena.com/news/Apples-A12-chip-packed-with-70-percent-more-transistors_id109291

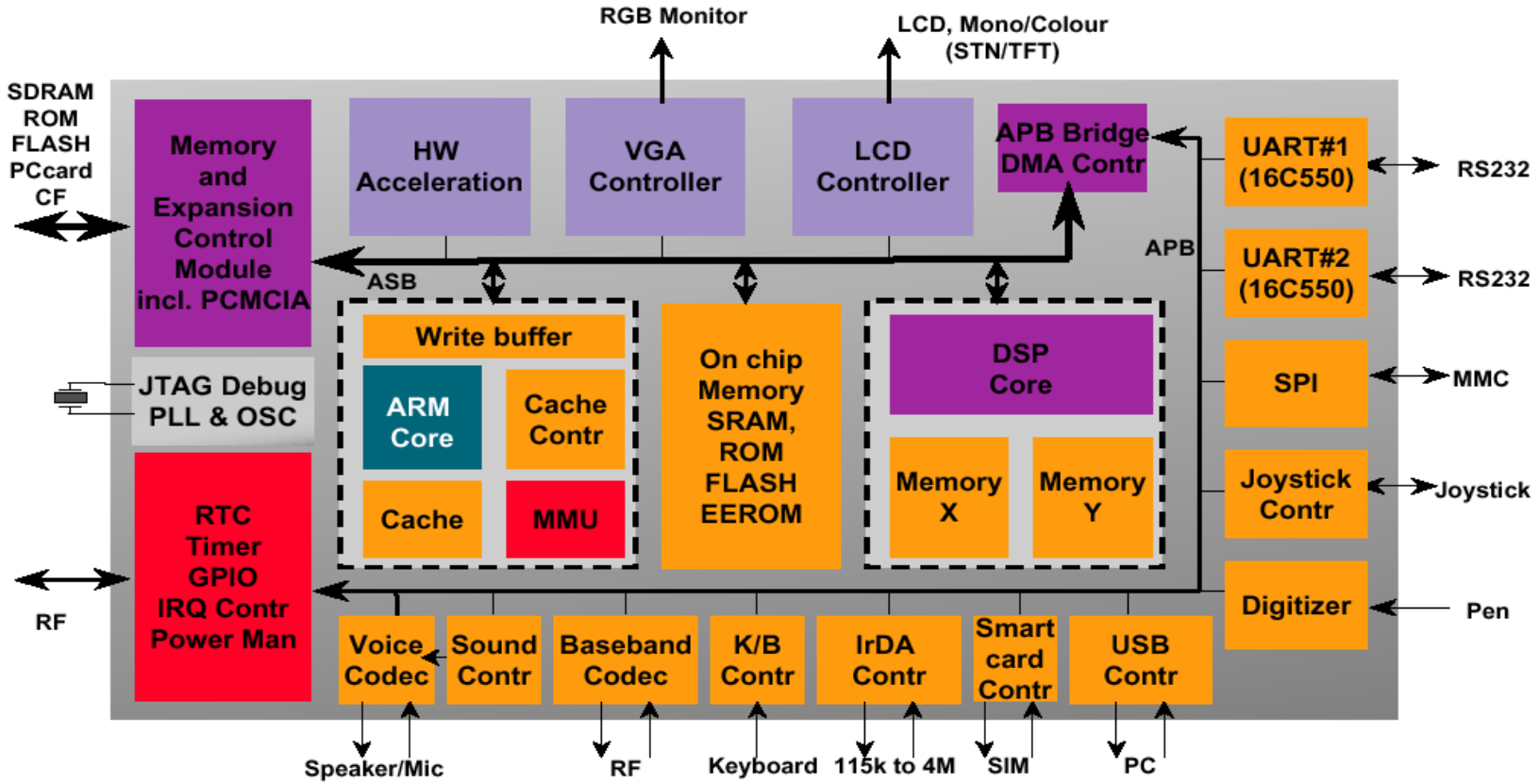
https://www.loveios.net/2018/08/tsmc-virus-attack-may-affect-a12-production.html

Mapping Onto System-on-Chip (SoC)

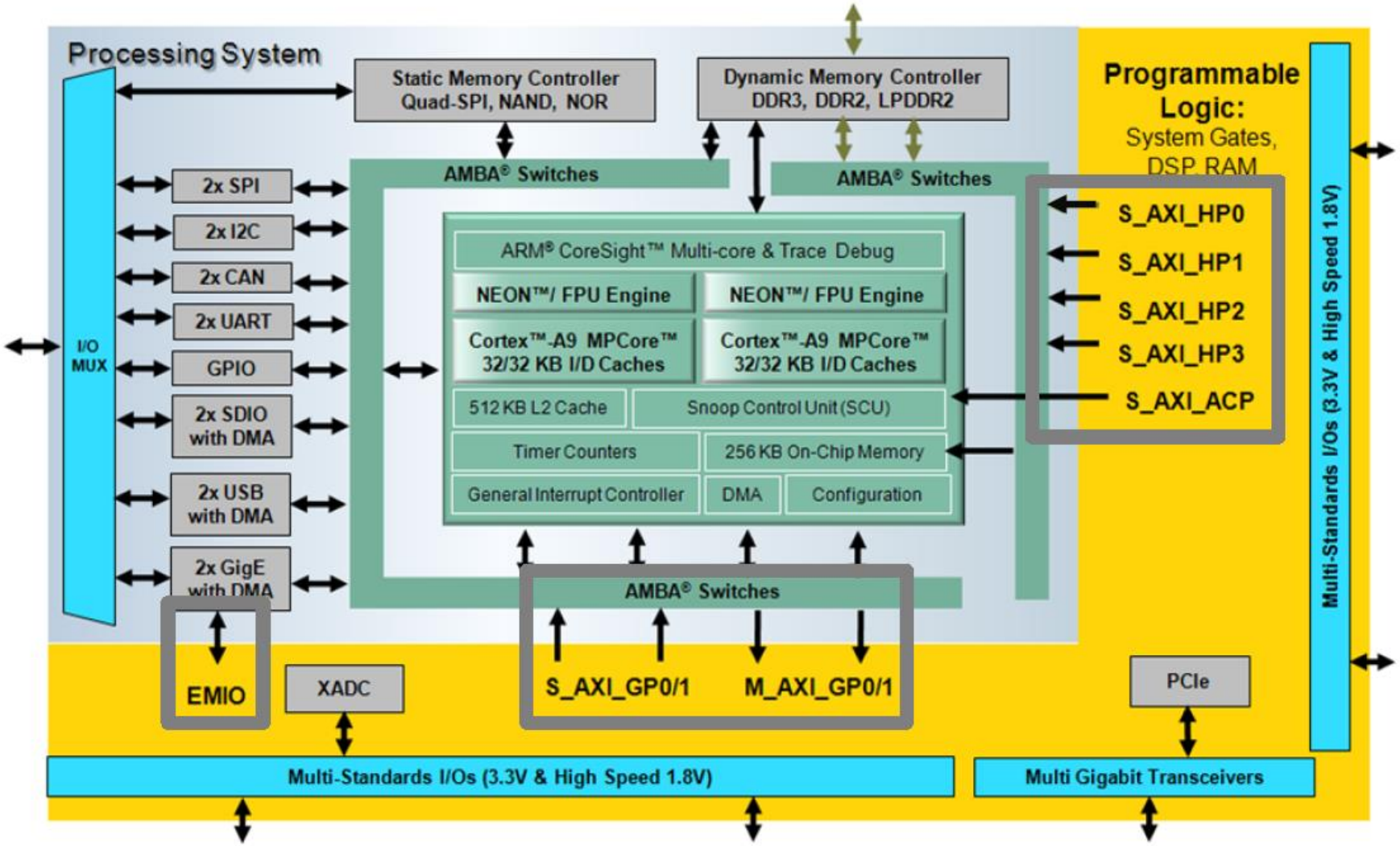


System on a Chip (SOC)

Source: ARM



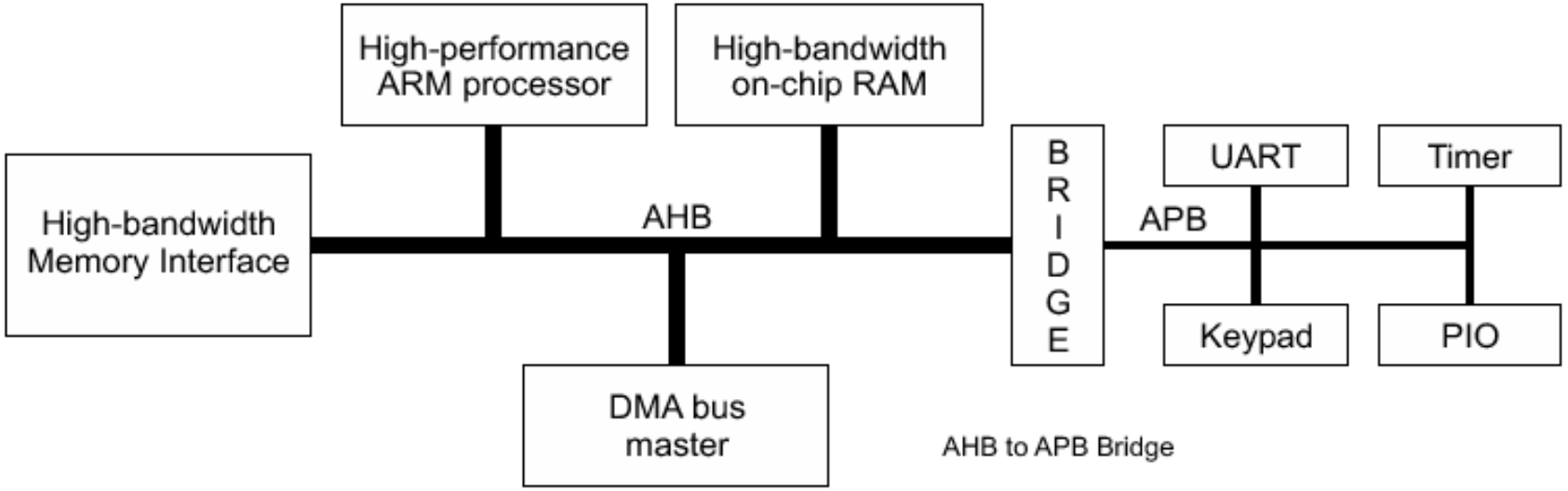
PYNQ


Processing System
Static Memory Controller
Quad-SPI, NAND, NOR
Dynamic Memory Controller
DDR3, DDR2, LPDDR2
Programmable Logic:
System Gates, DSP, RAM
AMBA® Switches
AMBA® Switches
I/O MUX
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
ARM® CoreSight™ Multi-core & Trace Debug
NEON™/ FPU Engine
NEON™/ FPU Engine
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
Cortex™-A9 MPCore™ 32/32 KB I/D Caches
512 KB L2 Cache
Snoop Control Unit (SCU)
Timer Counters
256 KB On-Chip Memory
General Interrupt Controller
DMA
Configuration
S_AXI_HP0
S_AXI_HP1
S_AXI_HP2
S_AXI_HP3
S_AXI_ACP
AMBA® Switches
Multi-Standards I/Os (3.3V & High Speed 1.8V)
EMIO
XADC
S_AXI_GP0/1
M_AXI_GP0/1
PCIe
Multi-Standards I/Os (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers

Breakdown of Modern SOC Design

- Core
  - CPU/DSPs/GPUs
  - Hardware accelerators (AI engine/video codec/MP3 codec)
  - Complex data path + FSMs (pipeline/parallel)
- Bus
  - Connect core and peripherals
  - MUX/arbiters
- Peripheral
  - Interface from/to the external
  - speech/audio, camera/display, I2S/I2C, GPIO
  - Serial to parallel, parallel to serial + FSM

An Example AMBA System



AHB to APB Bridge

AMBA Advanced High-performance Bus (AHB)
* High performance
* Pipelined operation
* Burst transfers
* Multiple bus masters
* Split transactions

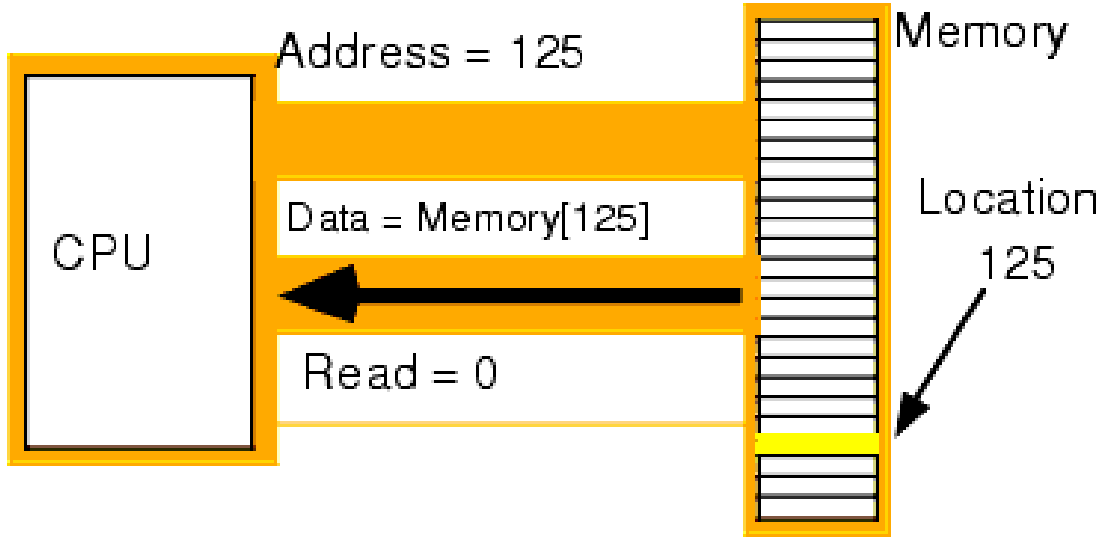
AMBA Advanced Peripheral Bus (APB)
* Low power
* Latched address and control
* Simple interface
* Suitable for many peripherals

From C Array to Memory Access

Memory [125] = 0

Memory Write Operation

CPU = Memory [125]

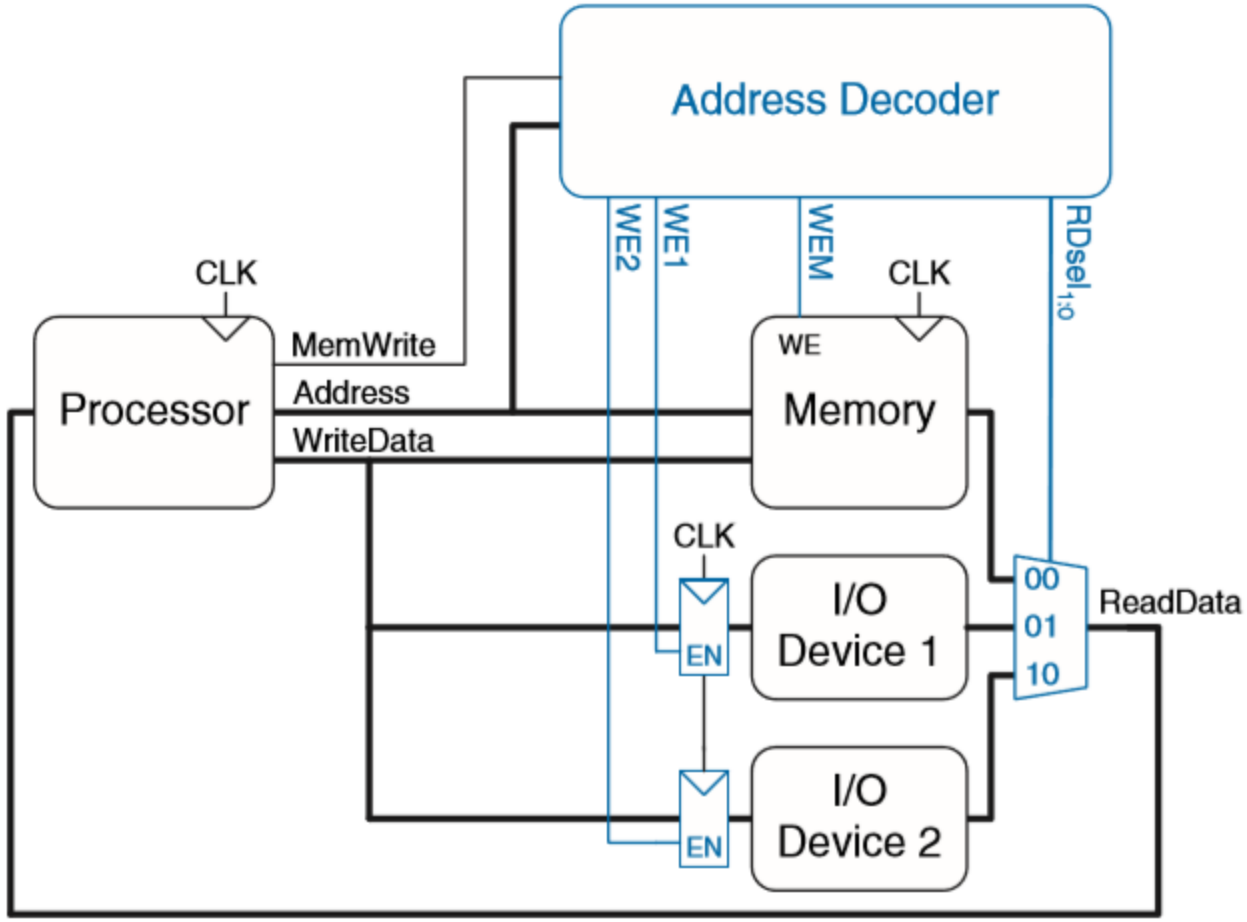


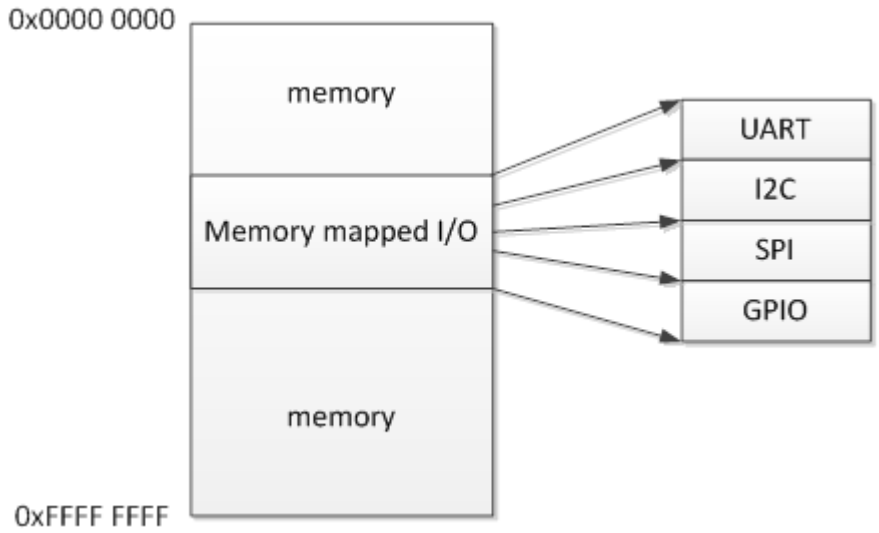
Memory Read Operation

http://www.phatcode.net/res/260/files/html/SystemOrganizationa2.html

How CPU access these I/O interface?

- Memory mapped I/O (treat other components as memory)



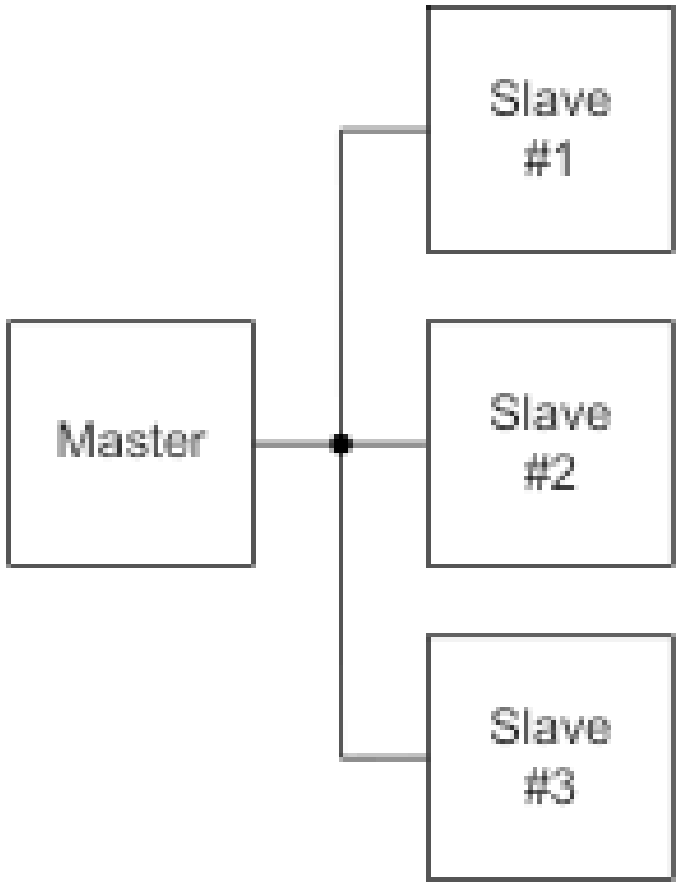


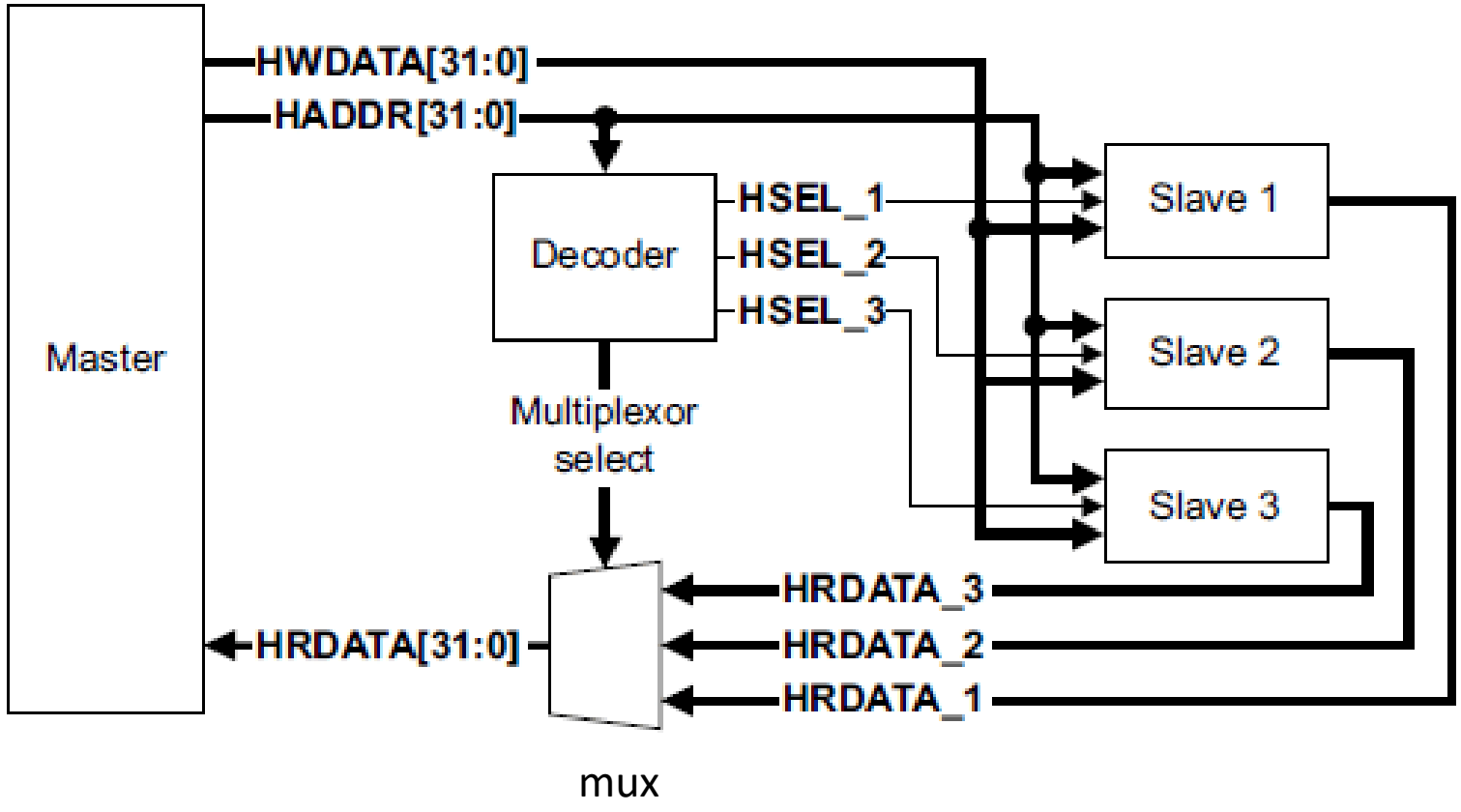
Example: USB sub system memory map in Beagle board

- I/O與memory共用記憶體空間，不用特別的指令來存取I/O，以記憶體讀寫的方式來進行I/O port的存取，在本文中使用的是ARM的核心，TI AM335X，便提供這種方式來進行I/O port的存取。

Block Name	Start_address (hex)	End_address (hex)	Size	Description
USBSS	0x4740_0000	0x4740_0FFF	20KB	USB Subsystem Registers
USB0	0x4740_1000	0x4740_12FF		USB0 Controller Registers
USB0_PHY	0x4740_1300	0x4740_13FF		USB0 PHY Registers
USB0 Core	0x4740_1400	0x4740_17FF		USB0 Core Registers
USB1	0x4740_1800	0x4740_1AFF		USB1 Controller Registers
USB1_PHY	0x4740_1B00	0x4740_1BFF		USB1 PHY Registers
USB1 Core	0x4740_1C00	0x4740_1FFF		USB1 Core Registers
USB CPPI DMA Controller	0x4740_2000	0x4740_2FFF		USB CPPI DMA Controller Registers
USB CPPI DMA Scheduler	0x4740_3000	0x4740_3FFF		USB CPPI DMA Scheduler Registers
USB Queue Manager	0x4740_4000	0x4740_4FFF		USB Queue Manager Registers

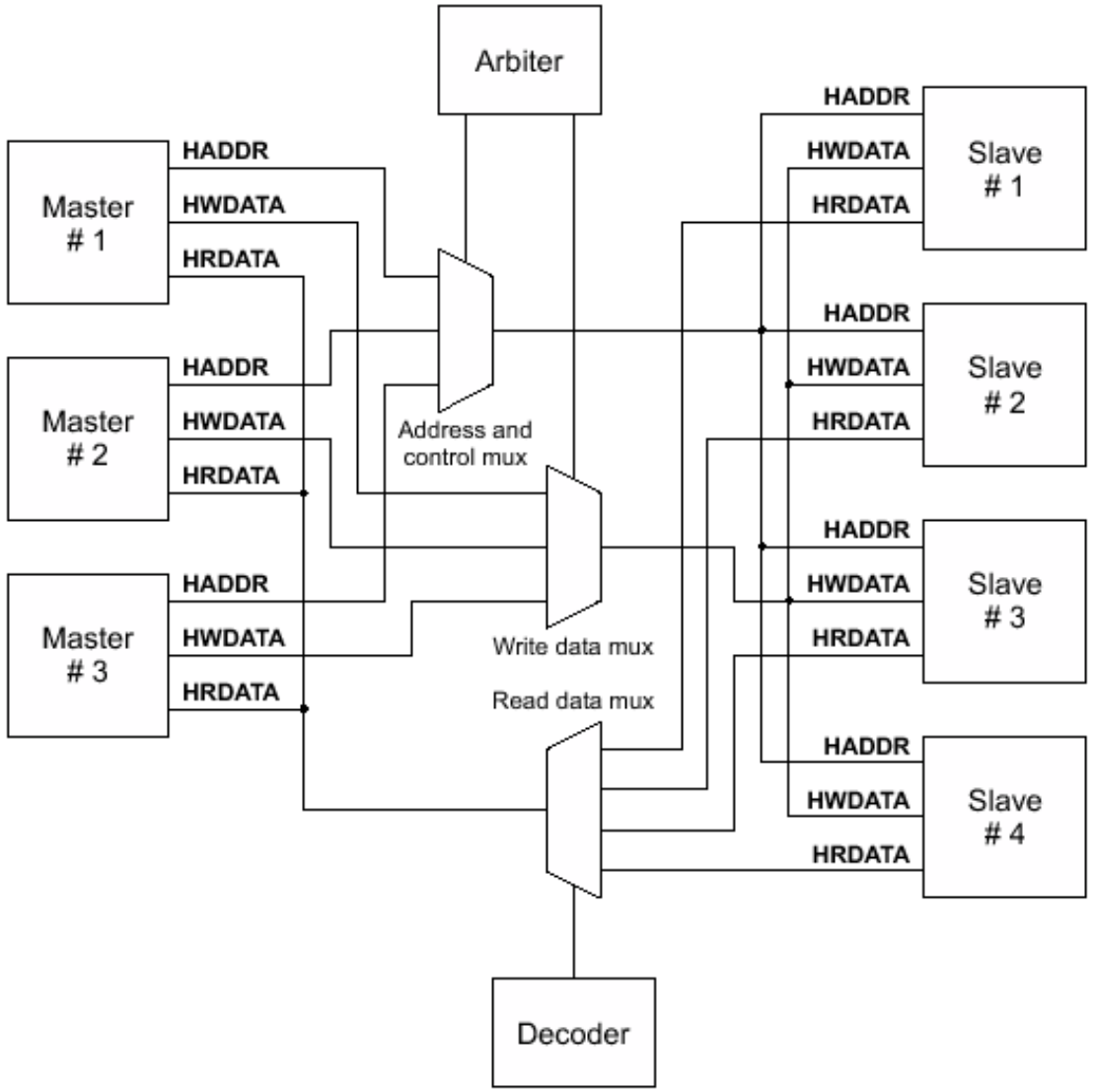
AHB-Lite


Master
Slave #1
Slave #2
Slave #3


Master
HWDATA[31:0]
HADDR[31:0]
Decoder
HSEL_1
HSEL_2
HSEL_3
Multiplexor select
Slave 1
Slave 2
Slave 3
HRDATA_3
HRDATA_2
HRDATA_1
HRDATA[31:0]

mux

AHB Interconnect



Basic components:

- Master
- Slave
- Arbiter
- Decoder
- Mux

Outline

- From K-Map to Verilog [Dally 7.1]
- Combinational building blocks – the idioms of digital design [Dally 8]
  - Decoder (binary to one-hot)
  - Encoder (one-hot to binary)
  - Muliplexer (select one of N)
  - Arbiter (pick first of N, 找第一個1) and priority encoder
  - Comparators
  - Read-only memories (ROM) / random access memory (RAM)

學習重點

- 知道如何對應基本硬體與Verilog code
  - 寫法很多種，哪一個比較好?
  - Verilog語法的差別
- 從2-bit 到 16bit 怎麼設計?怎麼寫?
  - Iterative circuit
  - Behavior level Verilog

FROM K-MAP TO VERILOG
從手動化簡到自動化電路
CASE/CASEZ/CASEX

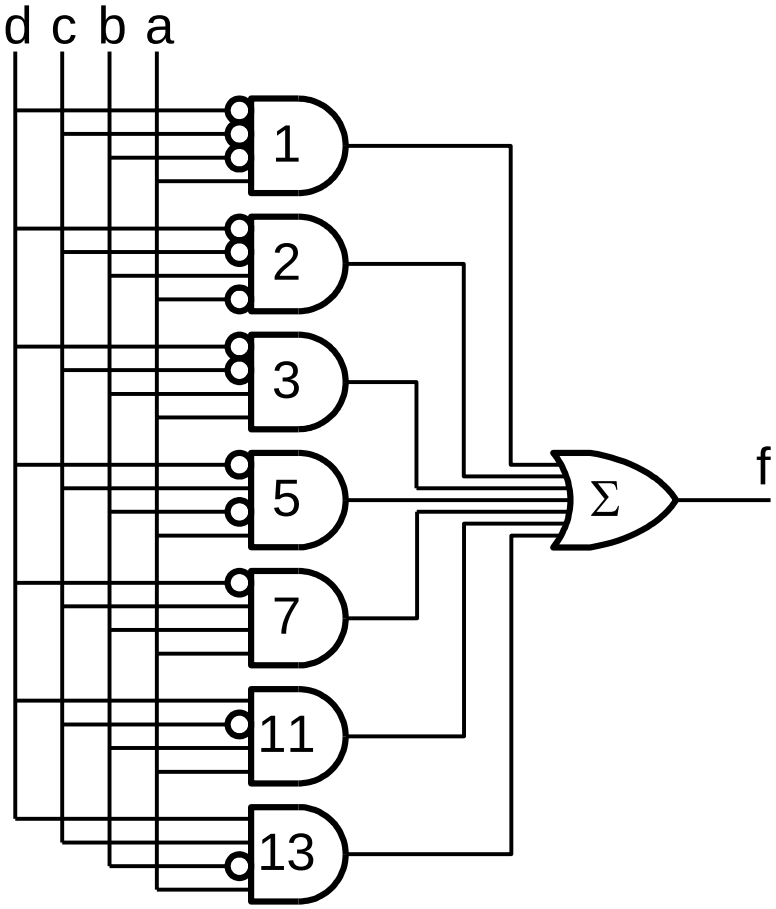
Prime Number Detection

- F(d,c,b,a) is true if input d,c,b,a is prime

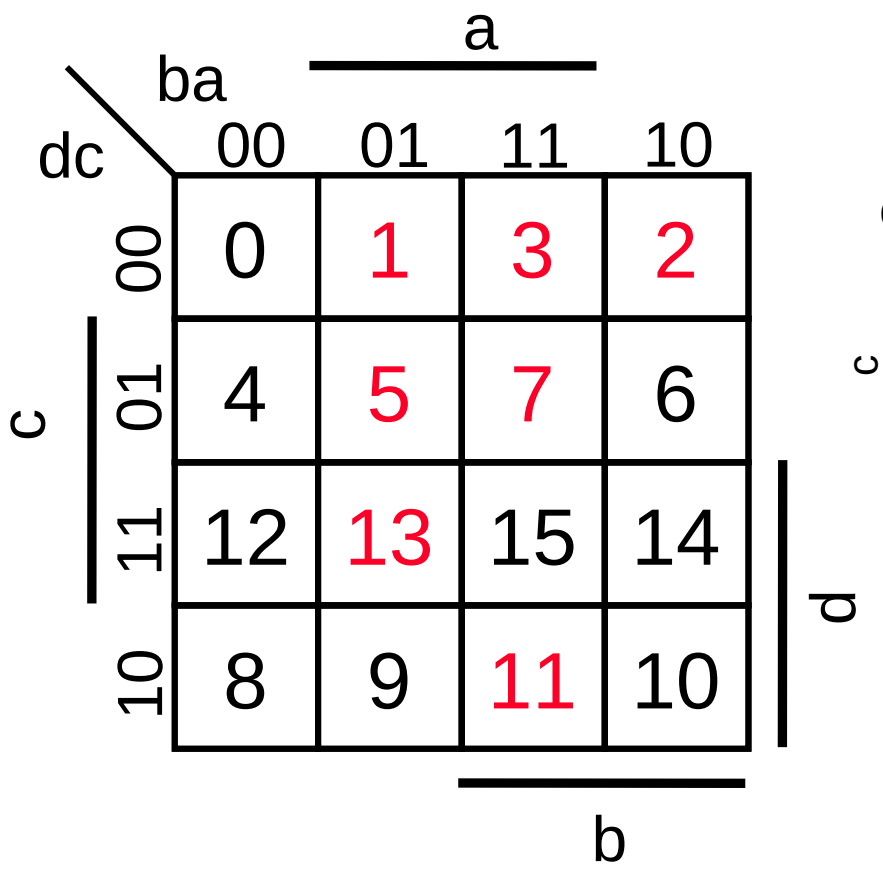
$$f = \sum_{dcba} m(1,2,3,5,7,11,13)$$

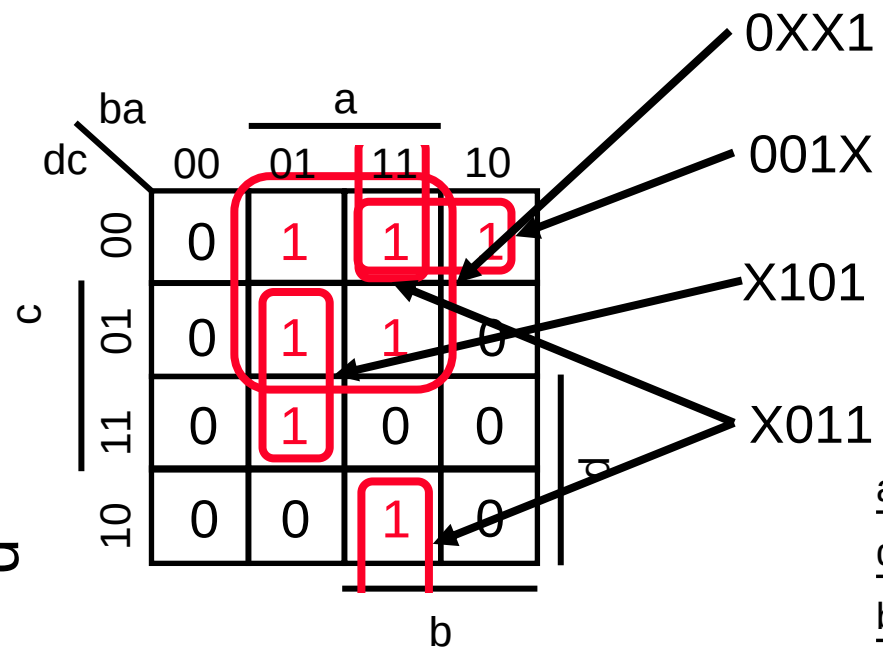
No	dcba	q
0	0000	0
1	0001	1
2	0010	1
3	0011	1
4	0100	0
5	0101	1
6	0110	0
7	0111	1
8	1000	0
9	1001	0
10	1010	0
11	1011	1
12	1100	0
13	1101	1
14	1110	0
15	1111	0

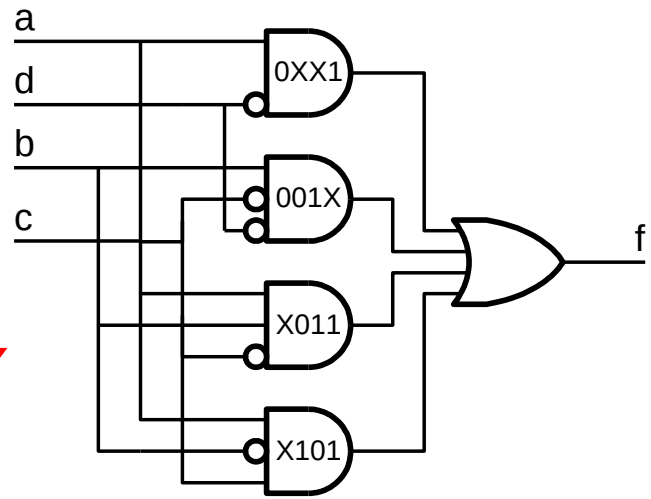
Schematic Logic Diagram:



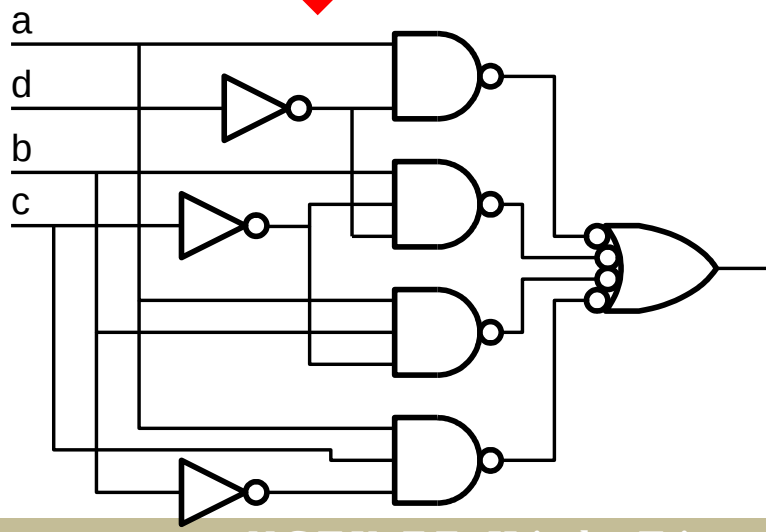
Karnaugh Map of 4-bit Prime Number







NAND for CMOS

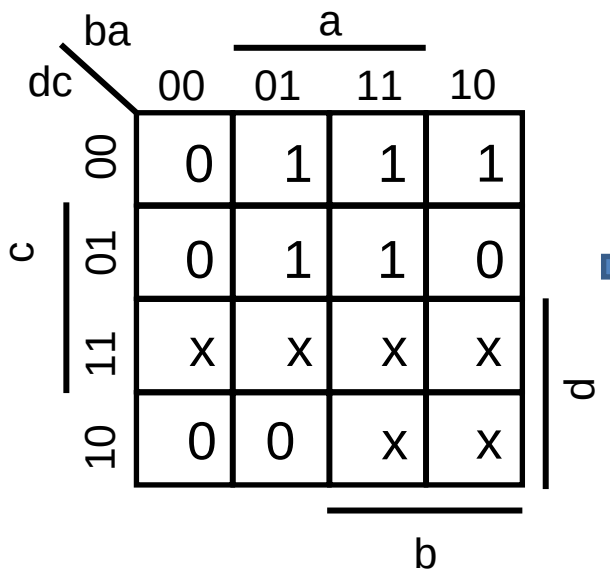


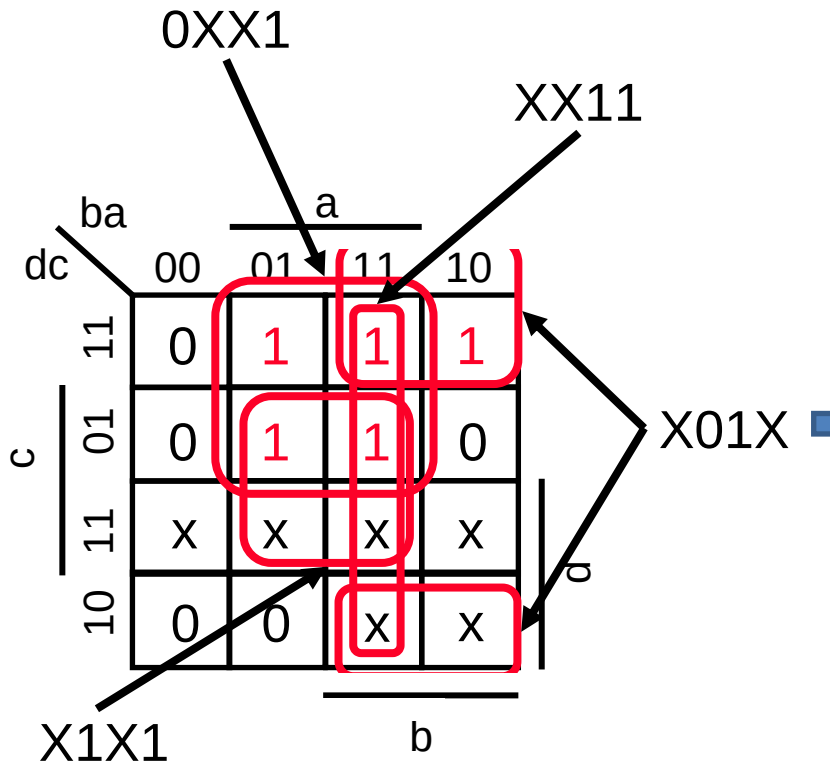
Position of minterms on a 4-bit Karnaugh map

更進一步化簡 Decimal Prime: includes don't cares

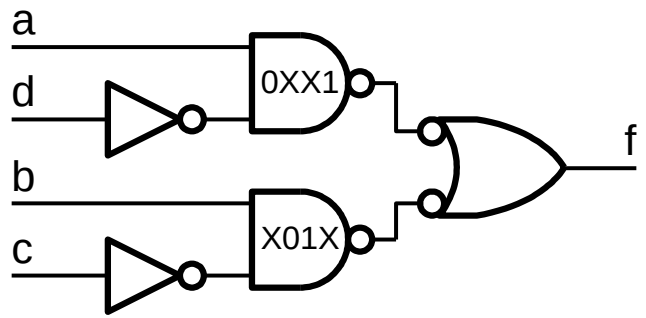
$$f = \sum_{dcba} m(1,2,3,5,7) + D(10,11,12,13,14,15)$$

Cover:
0XX1
X01X





$$f = (a \wedge \bar{d}) \vee (b \wedge \bar{c})$$



4-bit Prime Number Function in Verilog Code – Using case

```
module prime(in, isprime) ;
  input [3:0] in ;          // 4-bit input
  output      isprime ;     // true if input is prime
  logic       isprime ;

  always_comb begin
    case(in)
       1,2,3,5,7,11,13: isprime = 1'b1 ;
      default:          isprime = 1'b0 ;
    endcase
  end
endmodule
```

$$f = \sum_{dcba} m(1,2,3,5,7,11,13)$$

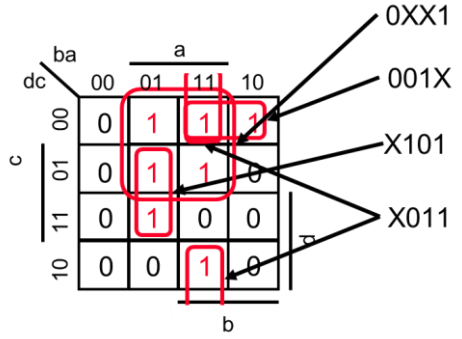
You can have a simplified Verilog coding if with don't care input

這很好，可是若有DON'T CARE INPUT，寫法可以更化簡

4-bit Prime Number Function in Verilog Code – Using casez

```
module prime1(in, isprime) ;
  input [3:0] in ;          // 4-bit input
  output      isprime ;     // true if input is prime
  logic       isprime ;

  always_comb begin
    casez(in)
      4'b0??1: isprime = 1 ;
      4'b001?: isprime = 1 ;
      4'b?011: isprime = 1 ;
      4'b?101: isprime = 1 ;
      default: isprime = 0 ;
    endcase
  end
endmodule
```

dcba
0xx1
001x
x101
x011



	case	casez	casex
比對的值	0, 1, Z. X	0, 1, X	0, 1
當作do't care		Z	X, Z
電路合成	0, 1	0, 1	0, 1

X: unknown, Z: high impedence, ?: Z

當輸入有X，以下電路會全部X

使用casez 利用到DON'T CARE INPUT，
但避免X propagation，造成
Simulation-synthesis mismatch

http://www.sunburst-design.com/papers/CummingsSNUG1999SJ_SynthMismatch.pdf
http://www.cnblogs.com/poiu-elab/archive/2012/11/02/2751323.html

Priority encoder

4-bit Prime Number Function in Verilog Code – Using assign

```
module prime(in, isprime) ;
  input [3:0] in ;          // 4-bit input
  output      isprime ;     // true if input is prime

  wire isprime = (in[0] & ~in[3]) |
                 (in[1] & ~in[2] & ~in[3]) |
                 (in[0] & ~in[1] & in[2]) |
                 (in[0] & in[1] & ~in[2]) ;
endmodule
```

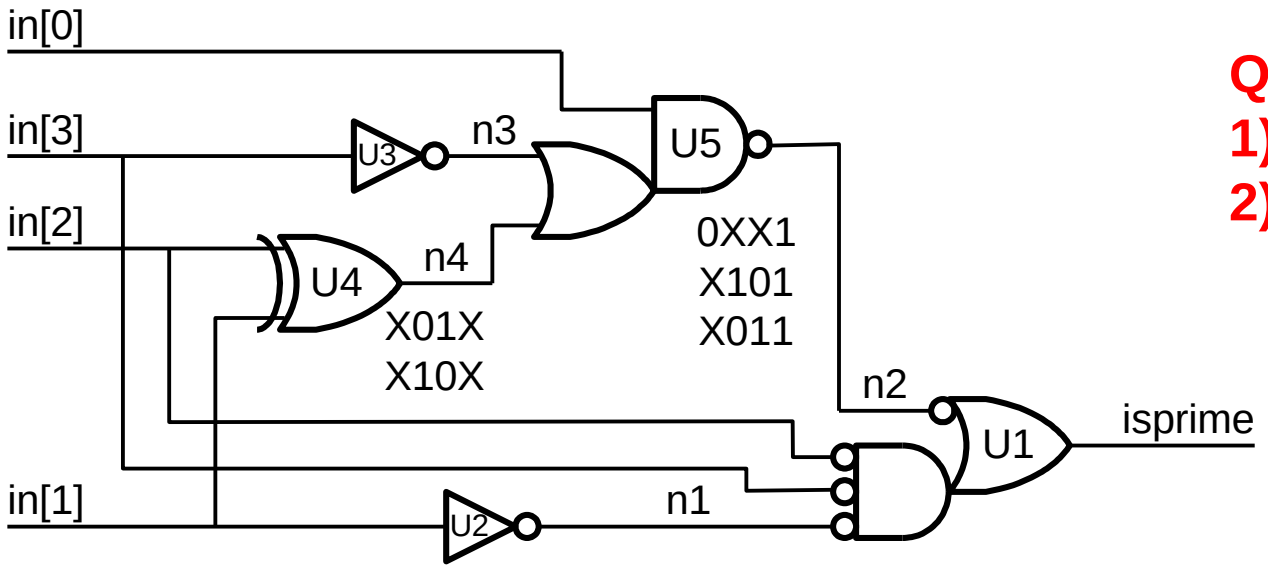
*簡單的化簡就留給**EDA tools** 做*

複雜的架構才值得花時間

You don't need to simplify logic by yourself, Left this to EDA tools

4-bit Prime Number Function in Verilog Code – Result of synthesizing description using case

```
module prime ( in, isprime );
input  [3:0] in;
output isprime;
    wire n1, n2, n3, n4;
    OAI13 U1 ( .A1(n2), .B1(n1), .B2(in[2]), .B3(in[3]), .Y(isprime) );
    INV   U2 ( .A(in[1]), .Y(n1) );
    INV   U3 ( .A(in[3]), .Y(n3) );
    XOR2  U4 ( .A(in[2]), .B(in[1]), .Y(n4) );
    OAI12 U5 ( .A1(in[0]), .B1(n3), .B2(n4), .Y(n2) );
endmodule
```



Quiz:
1)Where are these gates?
2)identify the critical path?

Synthesis Reports

```
****************************************
Report : area
Design : prime
Version: 2003.06
Date   : Sat Oct  4 11:38:08 2003
****************************************

Library(s) Used:

    XXXXX

Number of ports:                5
Number of nets:                 9
Number of cells:                5
Number of references:           4

Combinational area:          7.000000
Noncombinational area:       0.000000
Net Interconnect area:      undefined  (Wire load has zero
     net area)

Total cell area:             7.000000
Total area:                 undefined
```

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : prime
Version: 2003.06
Date   : Sat Oct  4 11:38:08 2003
****************************************

Operating Conditions:
Wire Load Model Mode: enclosed

  Startpoint: in[2] (input port)
  Endpoint: isprime (output port)
  Path Group: (none)
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  prime              2K_5LM                XXXXX

  Point                                    Incr       Path
  ------------------------------------------------------------
    -
  input external delay                     0.000      0.000 r
  in[2] (in)                               0.000      0.000 r
  U4/Y (EX210)                             0.191      0.191 f
  U5/Y (BF051)                             0.116      0.307 r
  U1/Y (BF052)                             0.168      0.475 f
  isprime (out)                            0.000      0.475 f
  data arrival time                                   0.475
  ------------------------------------------------------------
    -
  (Path is unconstrained)
```

Left part: area report
Right part: timing report

Constraint File

```
//設定clock, I/O 限制
create_clock "clk" -name clk -period 2 -waveform {0 1.7}
set_clock_uncertainty 0.2 clk
set_fix_hold all_clocks()
set_input_delay 0.5 -clock clk {in}
set_output_delay -max 0.8 -clock clk {isprime}
//設定 loading
set_load -pin_load 5 {isprime}
```

//Note: these commands are for practical applications to include
// clock jitter, input/output loading capacitances

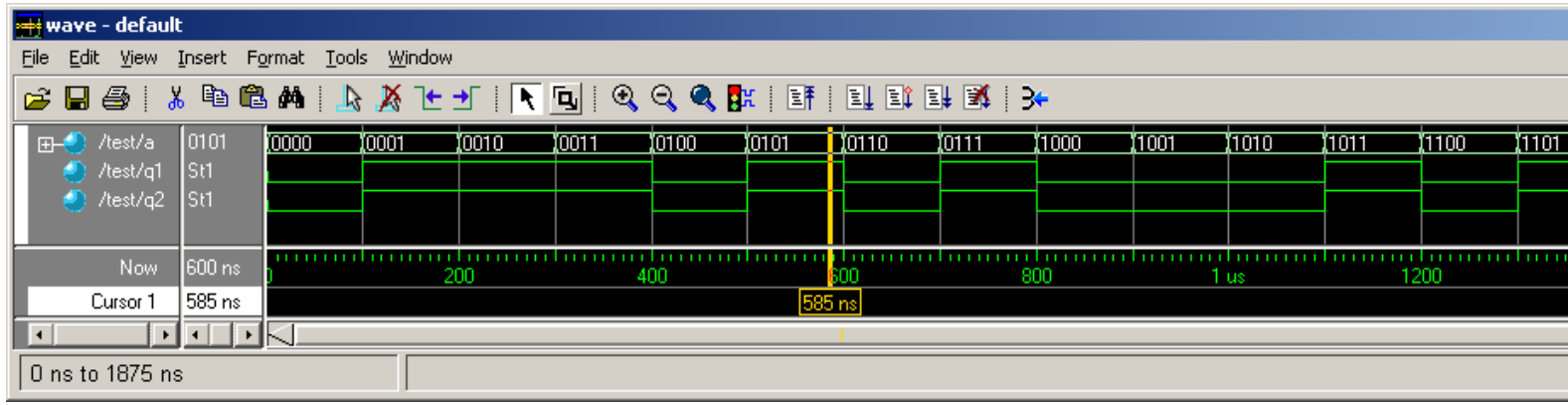
Test bench

```
module test_prime ;
  reg [3:0] in ;
  wire isprime ;

  // instantiate module to test
  prime  p0(in, isprime) ;

  initial begin
    in = 0 ;
    repeat (16) begin
      #100
      $display("in = %2d isprime = %1b",in,isprime) ;
      in = in+1 ;
    end
  end
endmodule
```

//this test module is included to
//simulate the behavior of your
//Verilog-HDL description
//Note: very often input data
//will be limited

Quiz: Why “divide-and-conquer”
Is often exploited in DCS design?

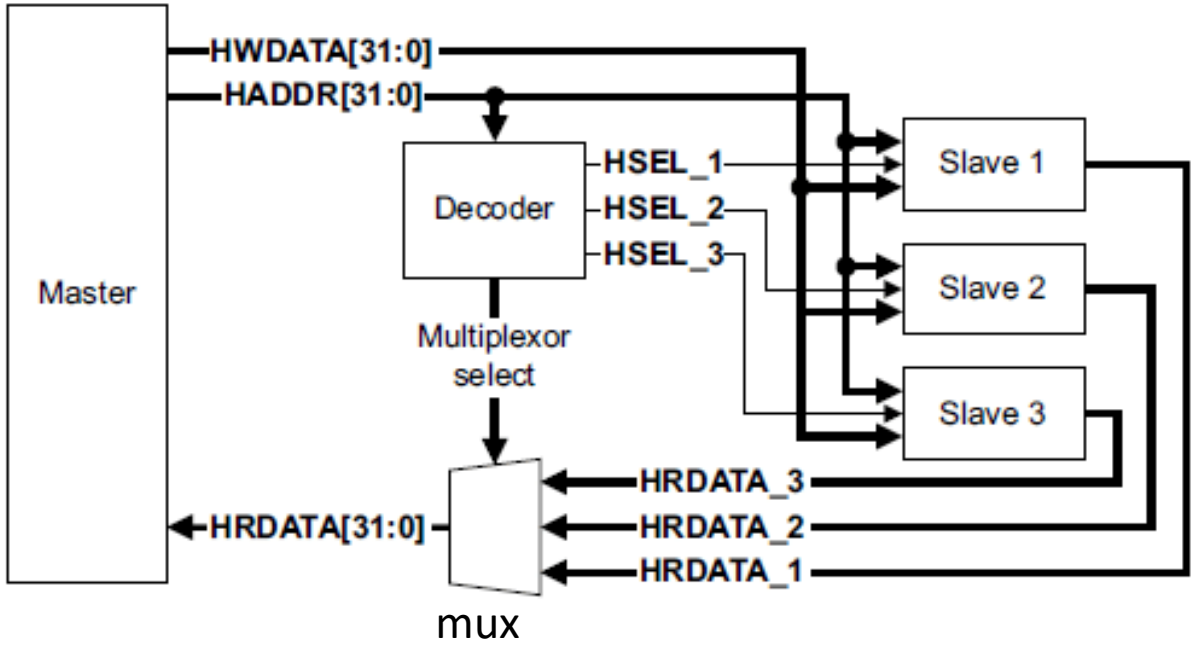
```
# in =  0 isprime = 0
# in =  1 isprime = 1
# in =  2 isprime = 1
# in =  3 isprime = 1
# in =  4 isprime = 0
# in =  5 isprime = 1
# in =  6 isprime = 0
# in =  7 isprime = 1
# in =  8 isprime = 0
# in =  9 isprime = 0
# in = 10 isprime = 0
# in = 11 isprime = 1
# in = 12 isprime = 0
# in = 13 isprime = 1
# in = 14 isprime = 0
# in = 15 isprime = 0
```



0 1 2 3 4 5 6 7 8 9 10 11 12 1

Summary

- K-Map helps you understand how logic synthesizer works
  - But no need to do it by yourself
  - Synthesis tool will do the optimization
- Use case
- Or casez if you have don't care input



DECODER AND ENCODER (BINARY <-> ONE HOT)

Binary to One-Hot Representation (Decoder)

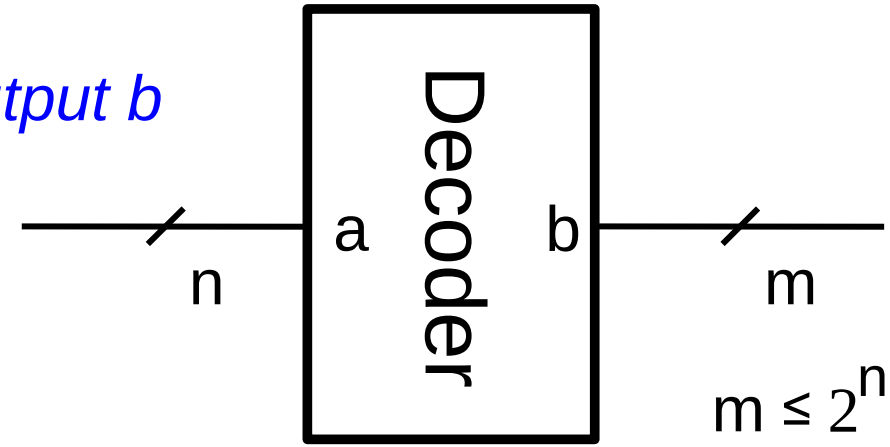
Binary	One-hot
000	00000001
001	00000010
010	00000100
…	…
110	01000000
111	10000000

常用於記憶體的address decoder
Used in address decoder for memory

Binary input a to one-hot output b

b[i] = 1 if a = i

b = 1<<a

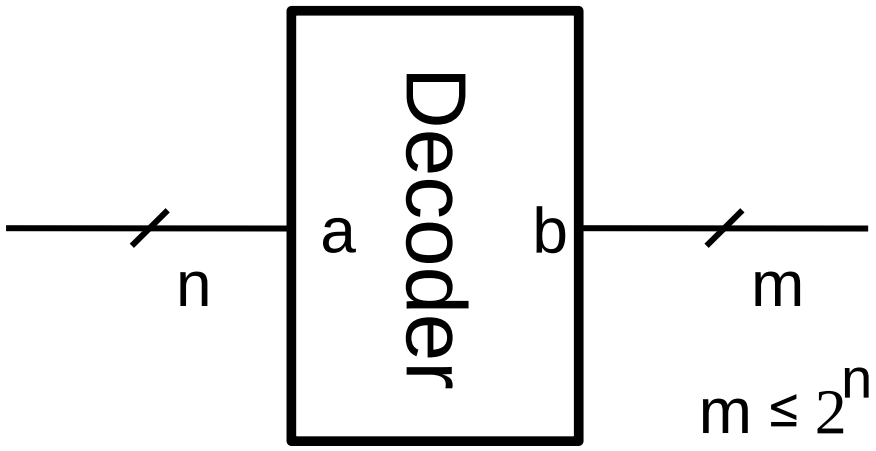


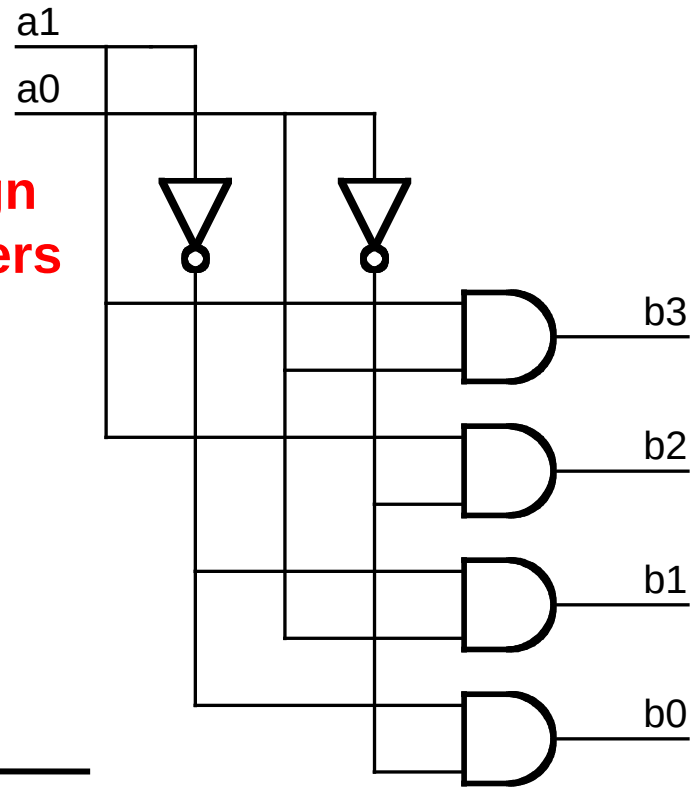
***can be found in source and communication systems**

Verilog implementation of a decoder

```
// a - binary input   (n bits wide)
// b - one hot output (m bits wide)
module dec(a, b) ;
  parameter n=2 ;
  parameter m=4 ;

  input  [n-1:0] a ;
  output [m-1:0] b ;

  wire [m-1:0] b;

  assign b = 1<<a ;
endmodule
```

```
//this is a parameterized design
//you need to assign parameters
//when exploited.
```

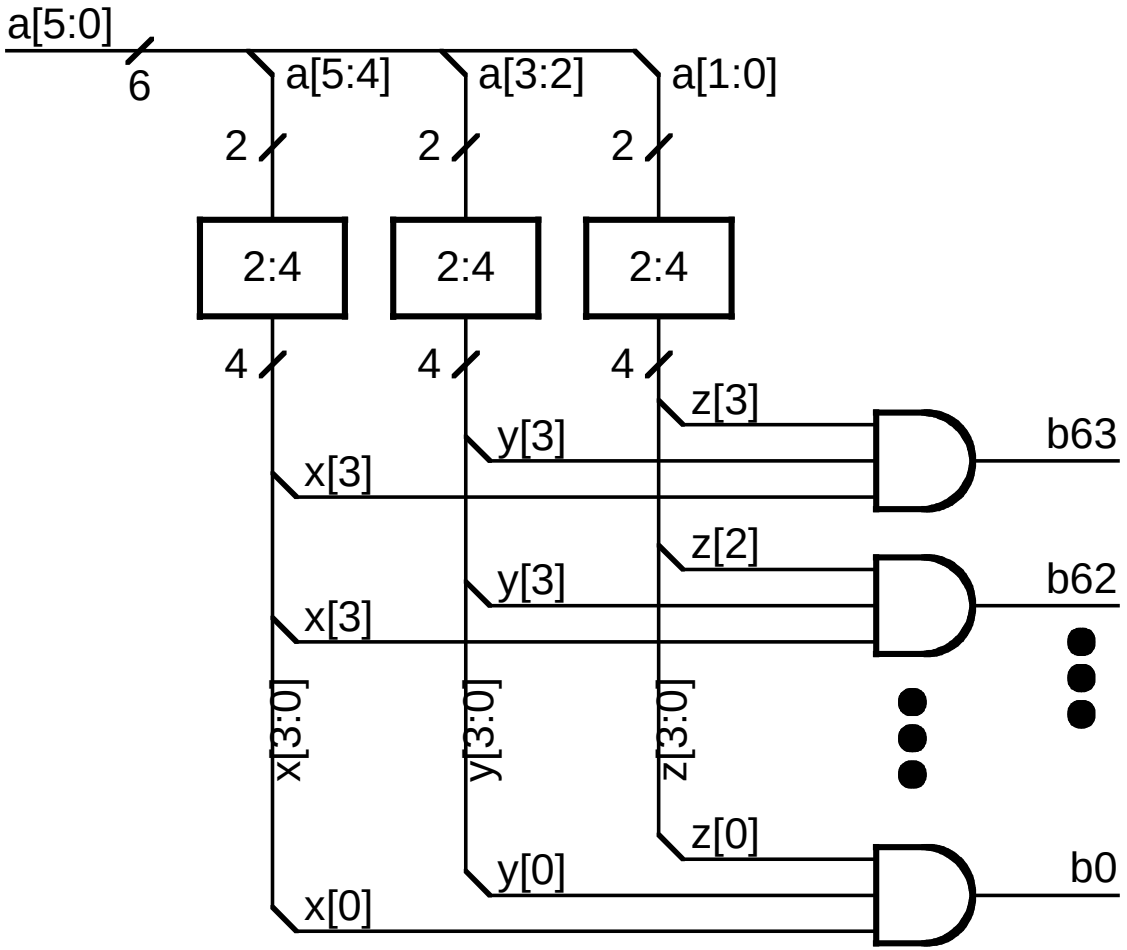




From 2->4 decoder to 6->64 Decoder
大的Decoder 如何又快又省

- Need hierarchical design 階層式設計
- 6->64 decoder requires:
  - 64 6-input AND gates (384 inputs)
- 6->64 decoder using 2->4 decoders requires:
  - 12 2-input AND gates (24 inputs)
  - 64 3-input AND gates (192 inputs)
- **Faster, smaller, lower power performances can be achieved.**

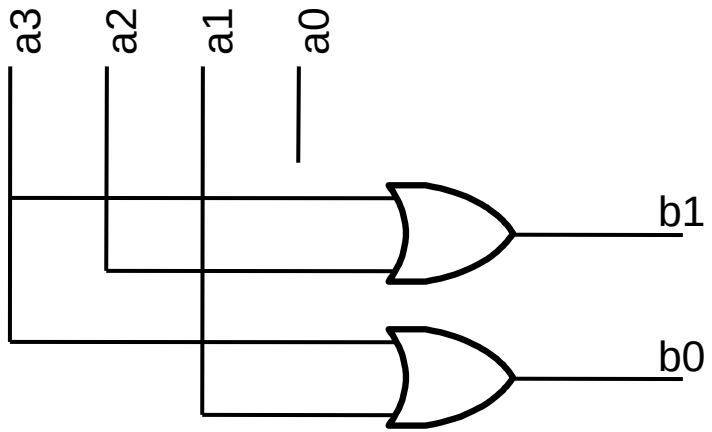
2-Stage decoders – the picture



Encoder

a 4 -> 2 encoder

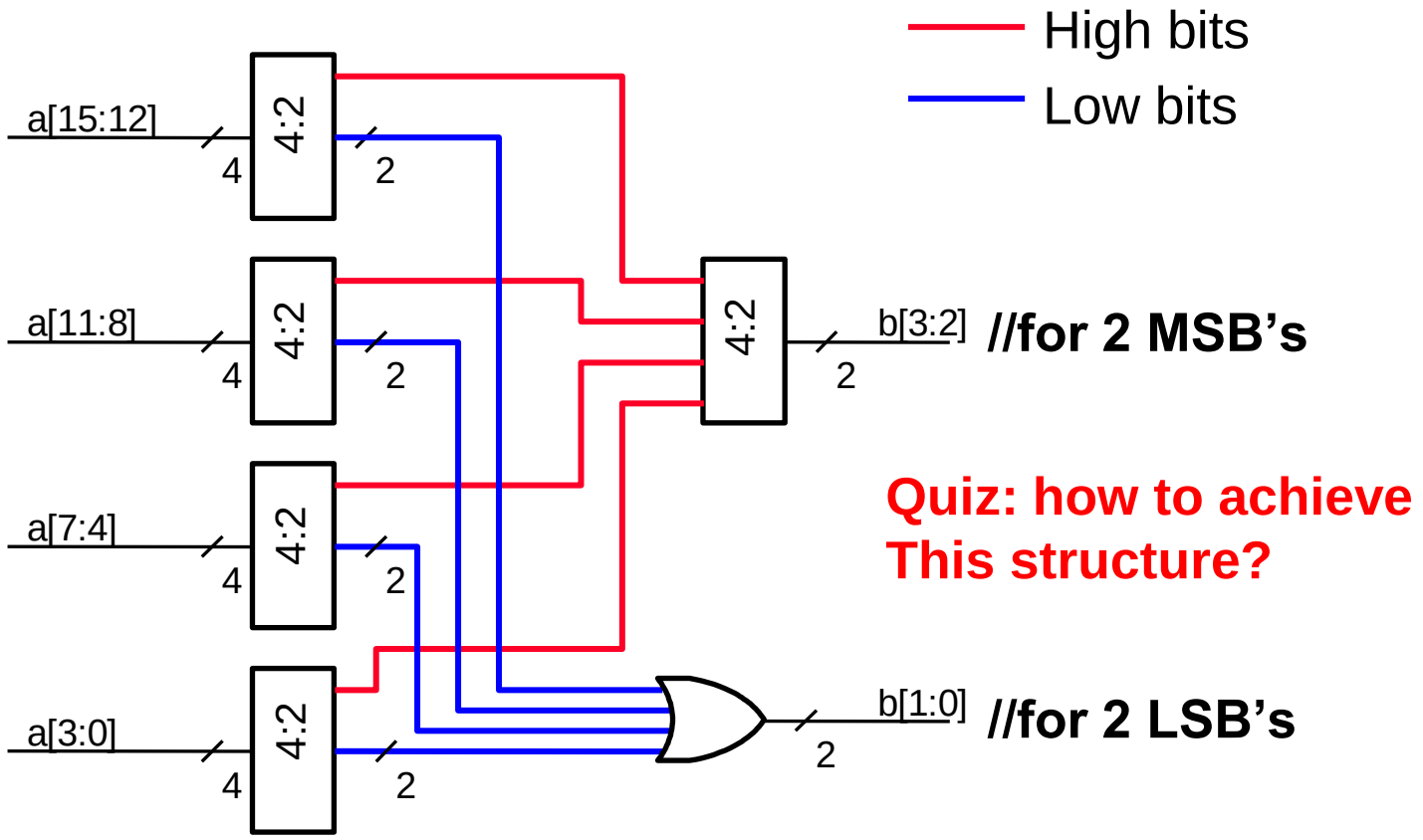
a3	a2	a1	a0	b1	b0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



那大的Encoder呢? E.g. 16-> 4

16->4 Encoder

One hot



Two writing styles 兩種寫法

```
// encoder - fixed width
module Enc42a(a, b) ;
  input  [3:0] a ;
  output [1:0] b ;
  wire   [1:0] b ;

  assign b[1] = a[3] | a[2] ;
  assign b[0] = a[3] | a[1] ;
endmodule
```

More like schematic

```
// encoder - fixed width
module Enc42a(a, b) ;
  input  [3:0] a ;
  output [1:0] b ;
  wire   [1:0] b ;

  always_comb begin
       case (a)
         4'b0001: b = 2'd0;
         4'b0010: b = 2'd1;
         4'b0100: b = 2'd2;
         4'b1000: b = 2'd3;
         4'b0000: b = 2'd0; // to faciltate large encoder
         default: b = 2'dxx;
       endcase
  end
endmodule
```

Behavior level

To avoid simulation and synthesis mismatch,
Set default value to one of the options. E.g. b = 2'd0;

Vote

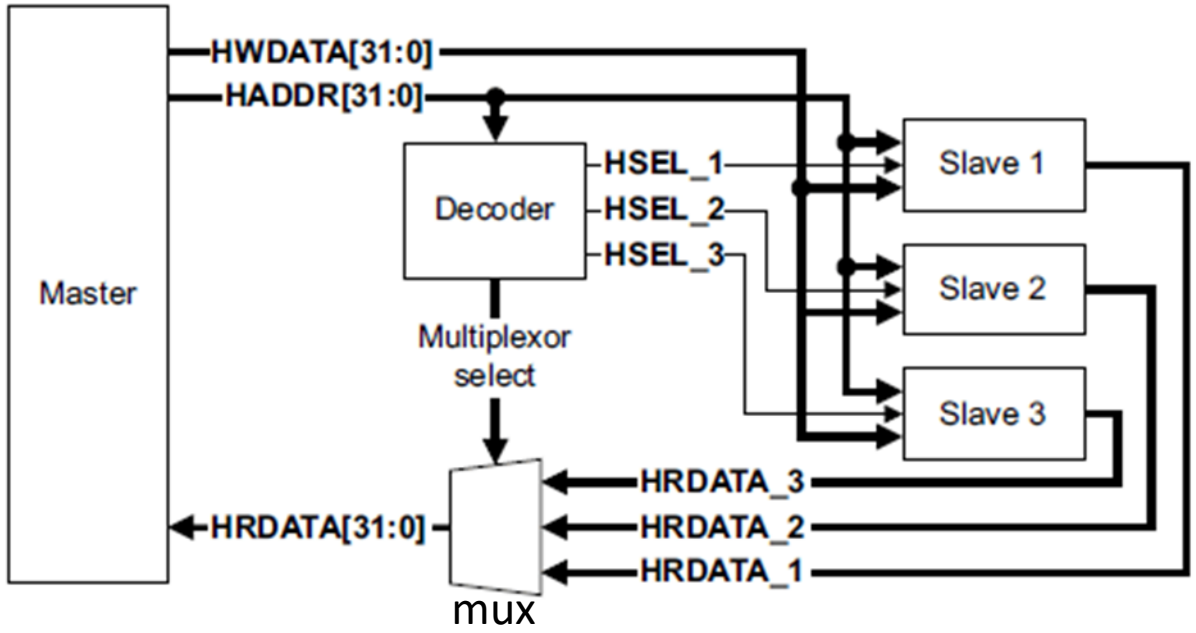
Binary Encoder

- Input is 1-hot

```
// Simple binary encoder (input is 1-hot)
module encode (A, Y);
input  [7:0] A;           // 8-bit input vector
output [2:0] Y;           // 3-bit encoded output
reg    [2:0] Y;           // target of assignment

  always @(A)
    case (A)
      8'b00000001: Y = 0;
      8'b00000010: Y = 1;
      8'b00000100: Y = 2;
      8'b00001000: Y = 3;
      8'b00010000: Y = 4;
      8'b00100000: Y = 5;
      8'b01000000: Y = 6;
      8'b10000000: Y = 7;
      default:     Y = 3'bXXX;    // Don't care when input is not 1-hot
    endcase
endmodule
```

- Cases are executed sequentially

```
// Priority encoder
module encode (A, Y);
input  [7:0] A;           // 8-bit input vector
output [2:0] Y;           // 3-bit encoded output
reg    [2:0] Y;           // target of assignment

  always @(A)
    case (1'b1)
      A[0]:    Y = 0;
      A[1]:    Y = 1;
      A[2]:    Y = 2;
      A[3]:    Y = 3;
      A[4]:    Y = 4;
      A[5]:    Y = 5;
      A[6]:    Y = 6;
      A[7]:    Y = 7;
      default: Y = 3'bXXX;// Don't care when input is all 0's
    endcase
endmodule
```



MULTIPLEXER

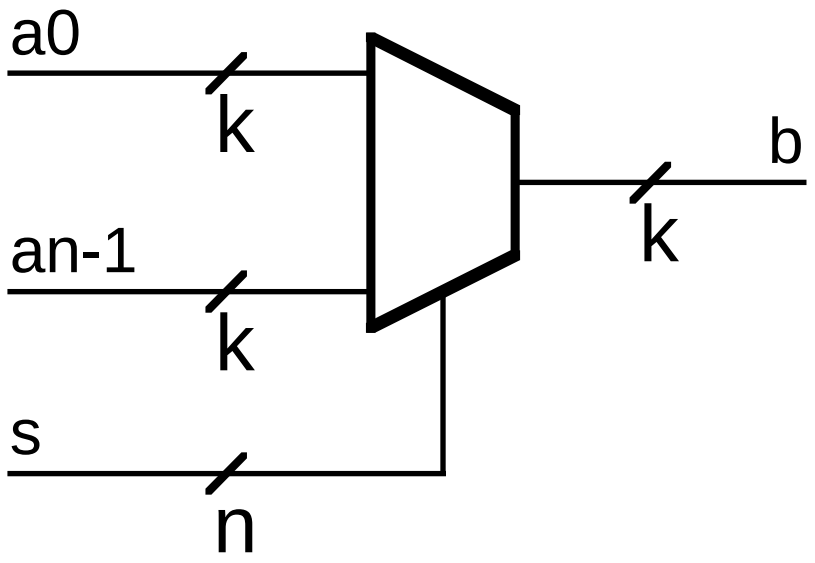
Multiplexer (one-hot selection signals)

- Multiplexer:
  - n *k-bit* inputs
  - n-bit one-hot select signal s
  - Multiplexers are commonly used as *data selectors*

Selects one of n k-bit inputs

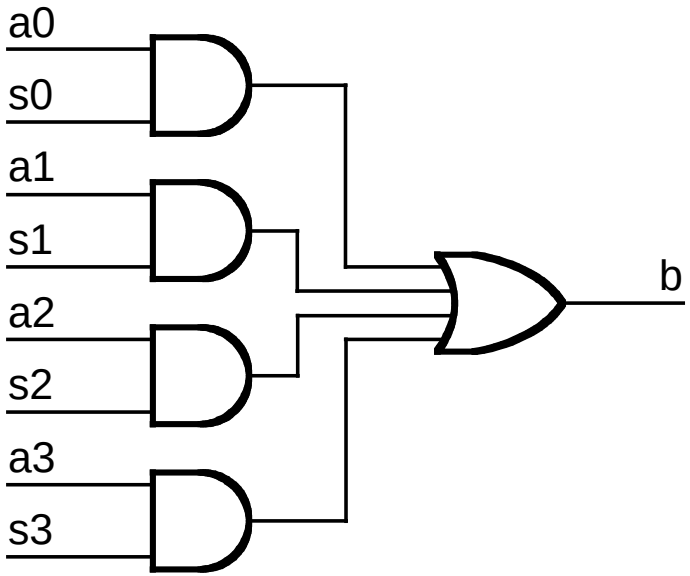
s must be one-hot

b=a[i] if s [i] = 1

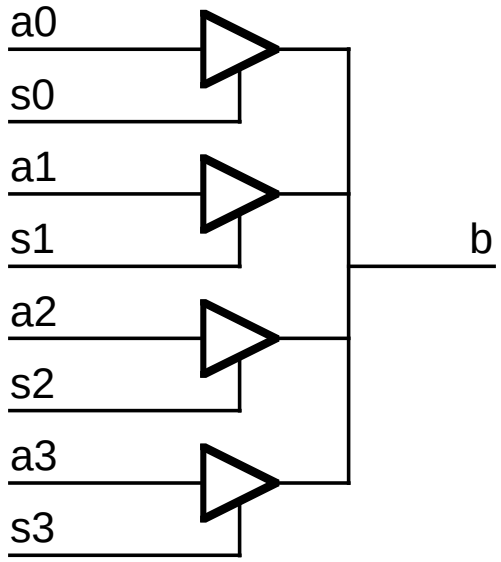


兩種架構，哪個好?

AND-OR structure



Tri-state structure



Quiz: What's the difference between these two implementations?

MUX4 v.s. MUX3

```
Mux4(a3, a2, a1, a0, s, b) ;
  parameter k = 1 ;
  input [k-1:0] a0, a1, a2, a3 ;
  input [3:0]   s ; // one-hot select
  output[k-1:0] b ;
  wire [k-1:0] b = ({k{s[0]}} & a0) |
                   ({k{s[1]}} & a1) |
                   ({k{s[2]}} & a2) |
                   ({k{s[3]}} & a3) ;
endmodule
```

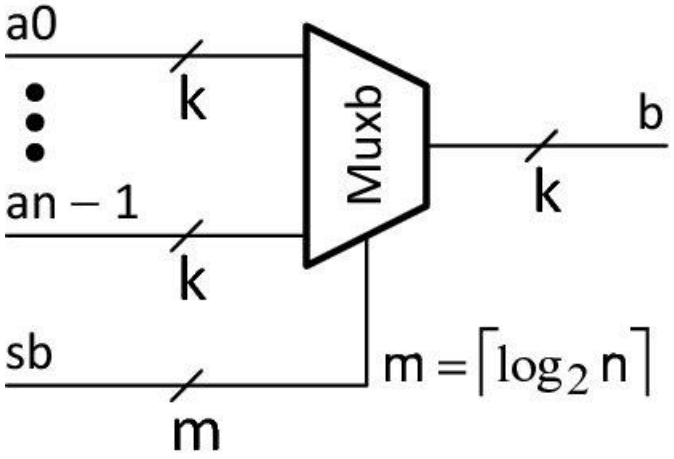
More like schematic

```
module mux3a(a2, a1, a0, s, b) ;
  parameter k = 1 ;
  input [k-1:0] a0, a1, a2 ;  // inputs
  input [2:0]   s ; // one-hot select
  output[k-1:0] b ;
  logic [k-1:0] b ;

  always_comb begin
    case(s)
      3'b001: b = a0 ;
      3'b010: b = a1 ;
      3'b100: b = a2 ;
      default: b =  {k{1'bx}} ;//don't care "X"
    endcase
  end
endmodule
```

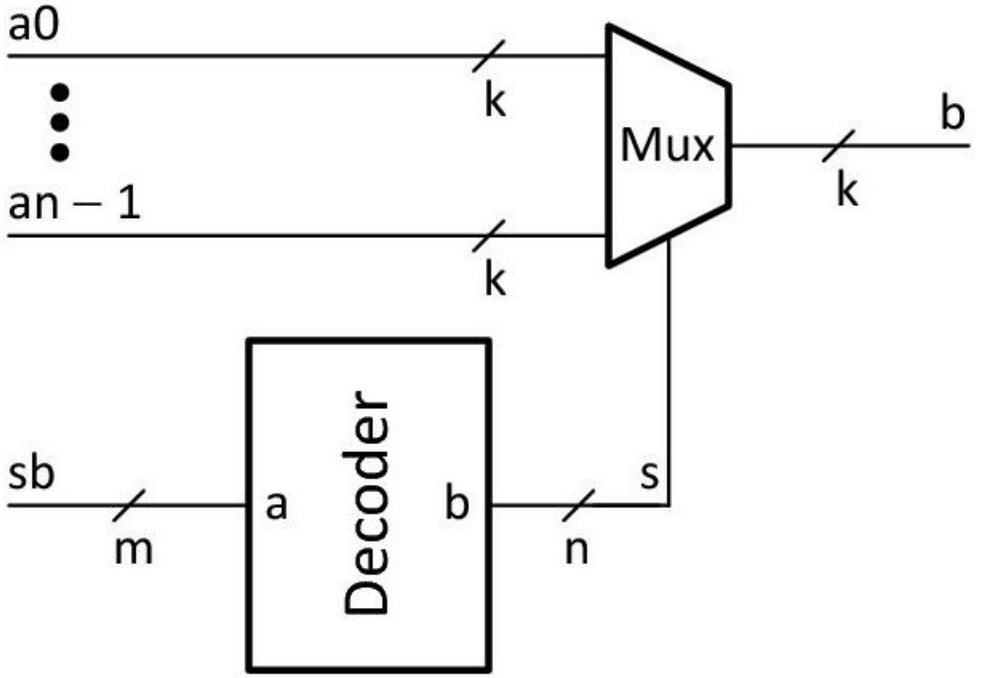
Behavior level

Vote

Binary Select MUX



(a)



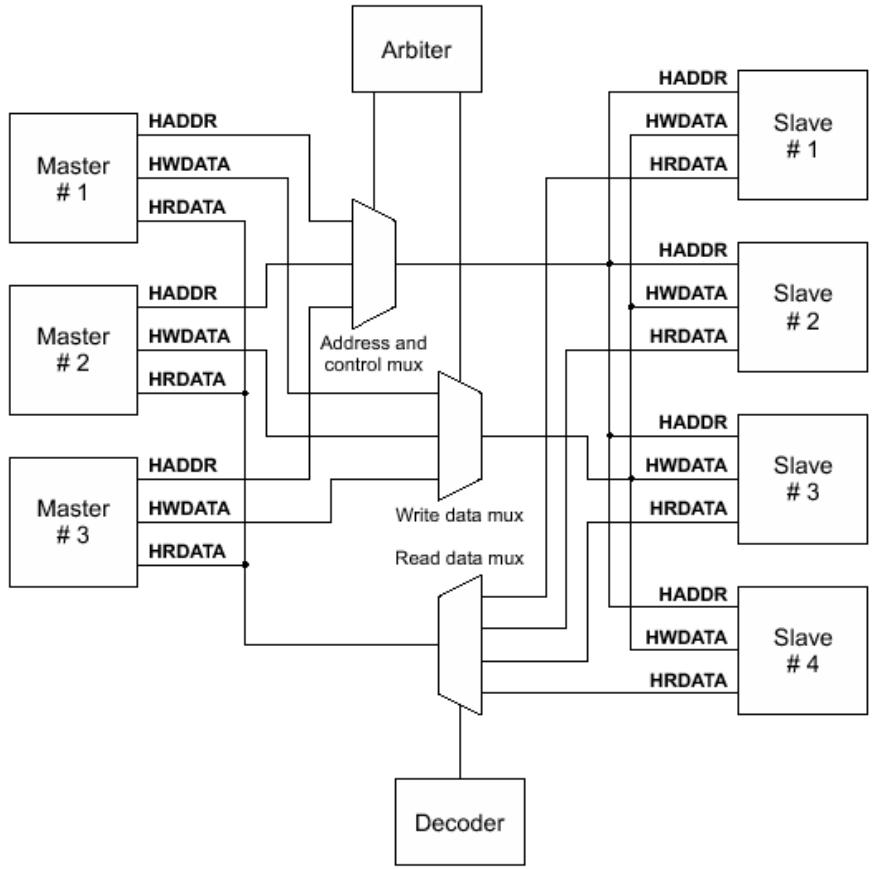
(b)

兩種寫法

```
// 3:1 multiplexer with binary select (arbitrary width)
module Muxb3(a2, a1, a0, sb, b) ;
  parameter k = 1 ;
  input [k-1:0] a0, a1, a2 ;  // inputs
  input [1:0]   sb ;          // binary select
  output[k-1:0] b ;
  wire  [2:0]   s ;

  dec #(2,3) d(sb,s) ; // Decoder converts binary to one-hot
  mux3 #(k)  m(a2, a1, a0, s, b) ; // multiplexer selects input
endmodule
```

```
always_comb begin
        case(sb)
        0: b = a0;
        1: b = a1;
        2: b = a2;
        default: b = {k{1'bx}};
end
```

那大的MUX怎麼辦?



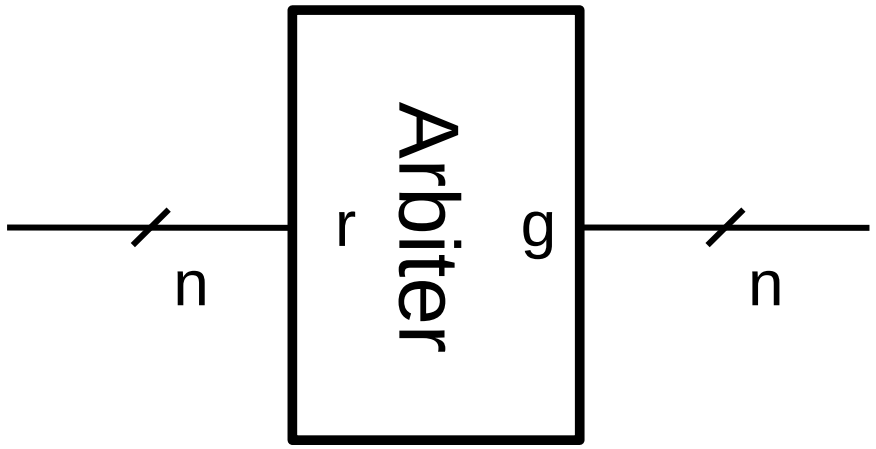
ARBITER AND PRIORITY ENCODER

找第一個1

000010111

Arbiter

- Arbiter handles requests from multiple devices to use a single resource
  - For bus arbitration
  - Normalization in floating point operations



e.g. LSB has the higher priority

input 0101_1100

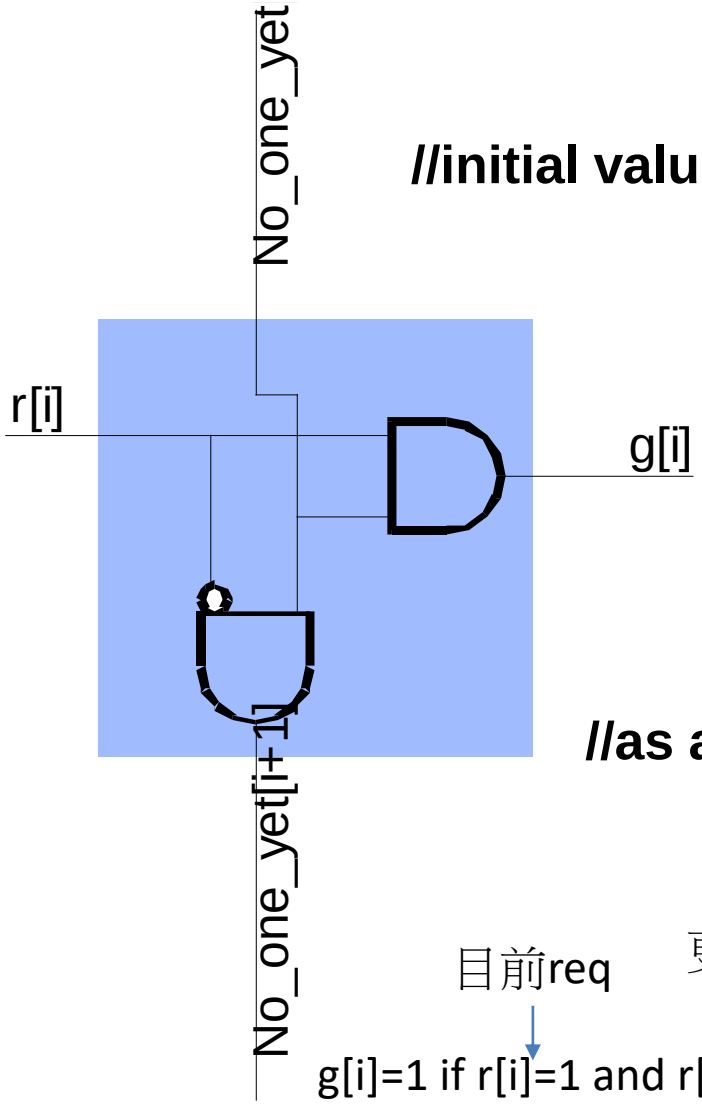
Output 0000_0100

Finds first "1" bit in r

g[i]=1 if r[i]=1 and r[j]=0 for j≥i

Leading one detector

1-Bit and 4-bit Arbiters



//initial value=1

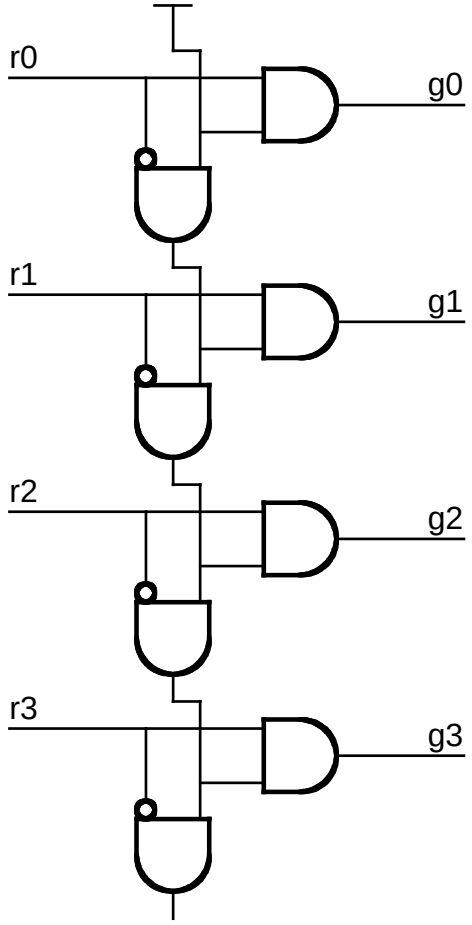
//as a leaf cell

目前req

更高權限的req

g[i]=1 if r[i]=1 and r[j]=0 for j>i

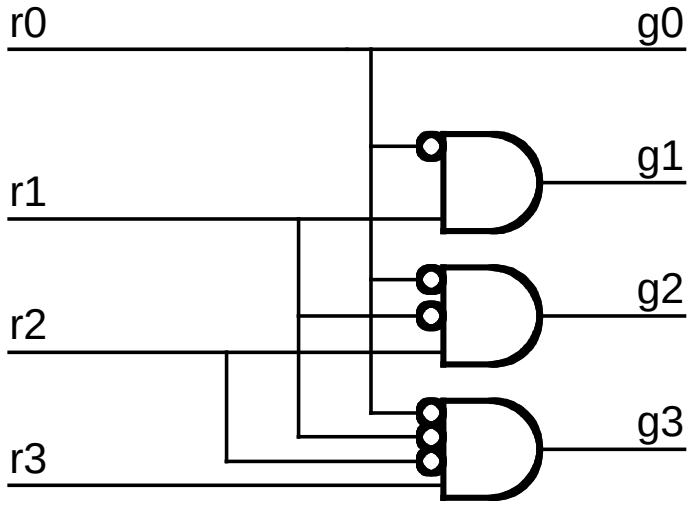
Using Bit-Cell



Iterative circuit

G0=r0,
G1=none & r1 = r0' & r1;

Using Look-Ahead



Quiz: What's the difference between These two?

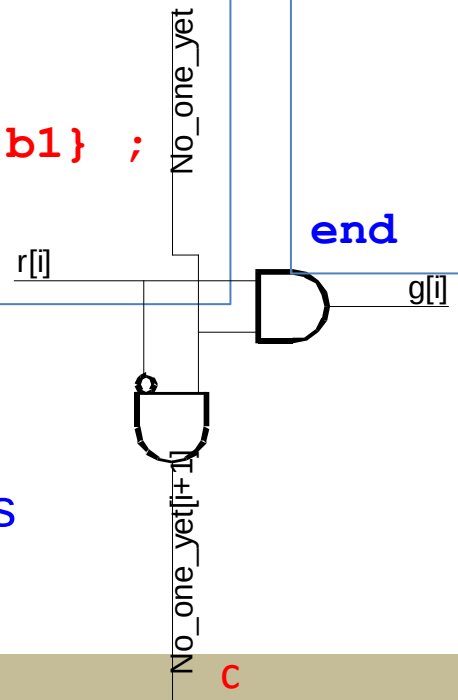
Implementing Arbitrary Width Arbiter Using Verilog

```
// arbiter (arbitrary width),
// LSB is the highest priority
module Arb(r, g) ;
  parameter n=8 ;
  input  [n-1:0] r ;
  output [n-1:0] g ;
  wire   [n-1:0] c ;
  wire   [n-1:0] g ;

  assign c = {(~r[n-2:0] & c[n-2:0]),1'b1} ;
  assign g = r & c ;
endmodule
```

```
always_comb begin
        casez(r)
          4'b0000: g = 4'b0000;
          4'b???1: g = 4'b0001;
          4'b??10: g = 4'b0010;
          4'b?100: g = 4'b0100;
          4'b1000: g = 4'b1000;
          default: g = 4'hx;
        endcase
end
```

Bit-slice coding style using
concatenation {a, b}
index ranges c[n-2:0]
c is 1s up to first 1 in r, then 0s

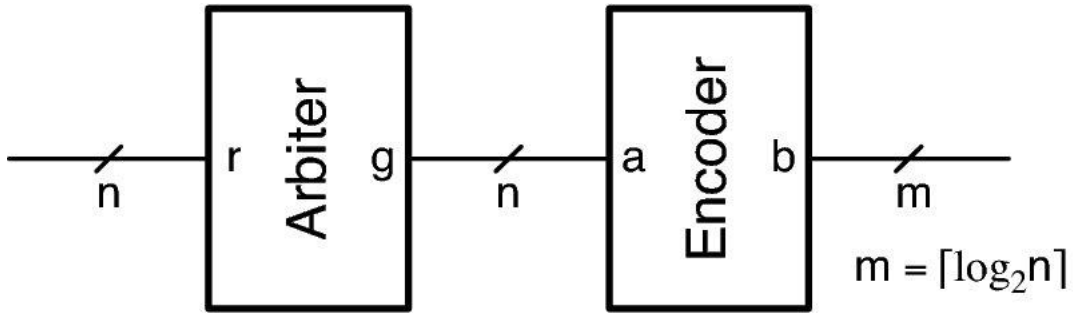
g[i]=1 if r[i]=1 and r[j]=0 for j>i



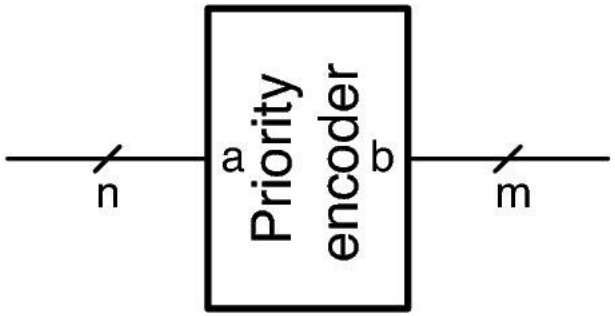
Priority Encoder

- Priority Encoder:
  - n-bit input signal a
  - m-bit output signal b
    - b indicates the position of the first 1 bit in a

```
always_comb begin
        casez(r)
          4'b???1: g = 2'd0;
          4'b??10: g = 2'd1;
          4'b?100: g = 2'd2;
          4'b1000: g = 2'd3;
          default: g = 2'dx;
        endcase
end
```



(a)

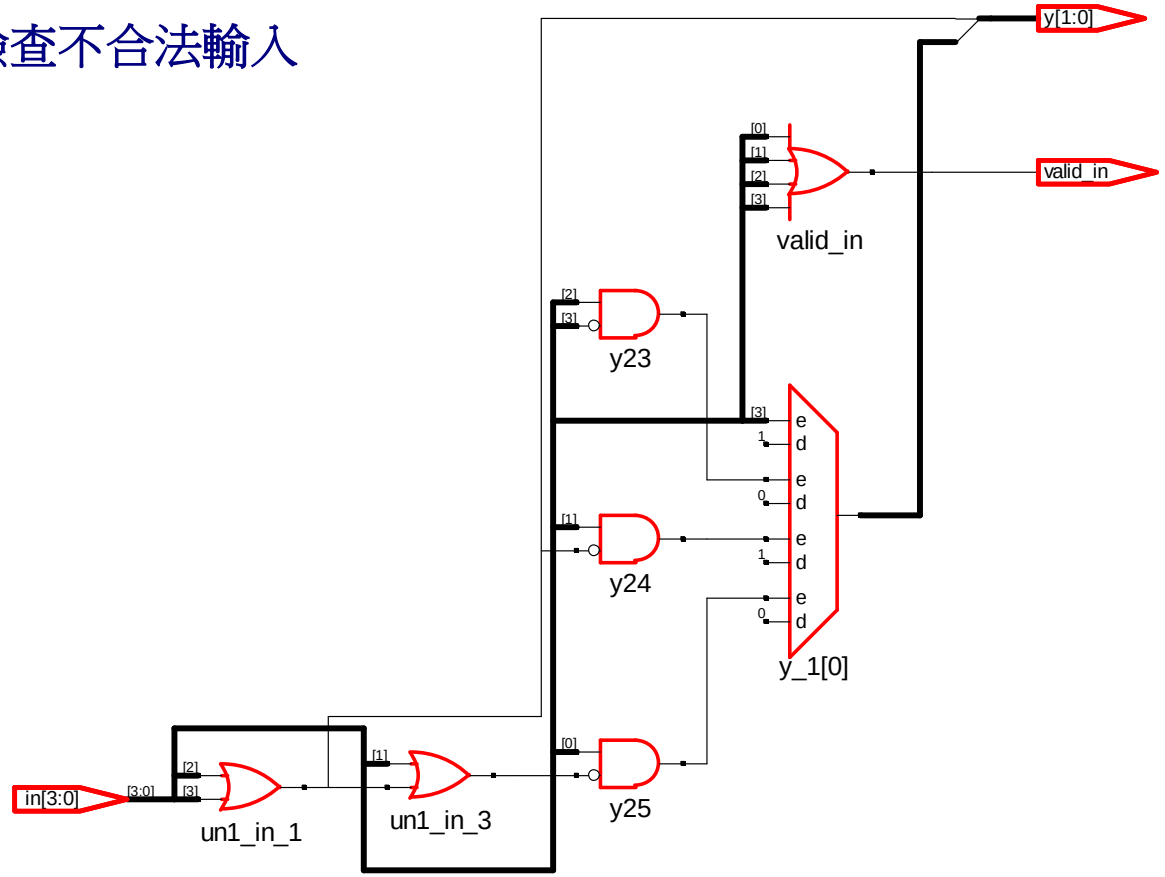


(b)

Check example 8.3 for programmable priority encoder

Priority Encoder with if-else

```
assign valid_in = |in; //檢查不合法輸入
always @(in) begin
    if (in[3]) y = 3; else
    if (in[2]) y = 2; else
    if (in[1]) y = 1; else
    if (in[0]) y = 0; else
    y = 2'bx;
end
```



Priority Encoder with Priority if-else

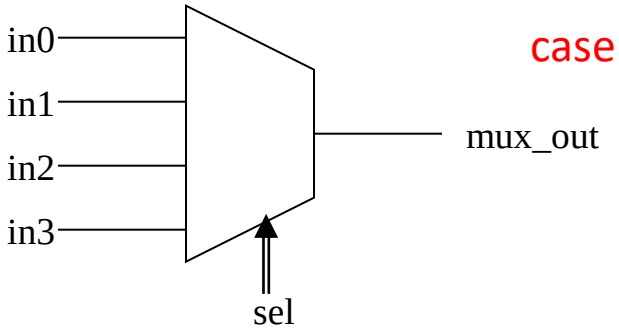
- Priority (SystemVerilog syntax)
  - 告訴simulator/synthesizer 就照這個順序執行，沒列在上面的就當作don't care，可以被化簡

```
module encoder (
input in0 , in1 , in2 , in3 ,
output logic [3:0] encoded_output ;
always_comb
  priority if ( in0 ) encoded_output = 4'b0001 ;
       else if ( in1 ) encoded_output = 4'b0010 ;
       else if ( in2 ) encoded_output = 4'b0100 ;
       else if ( in3 ) encoded_output = 4'b1000 ;
endmodule
```


http://www.sutherland-hdl.com/papers/2005-SNUG-paper_SystemVerilog_unique_and_priority.pdf

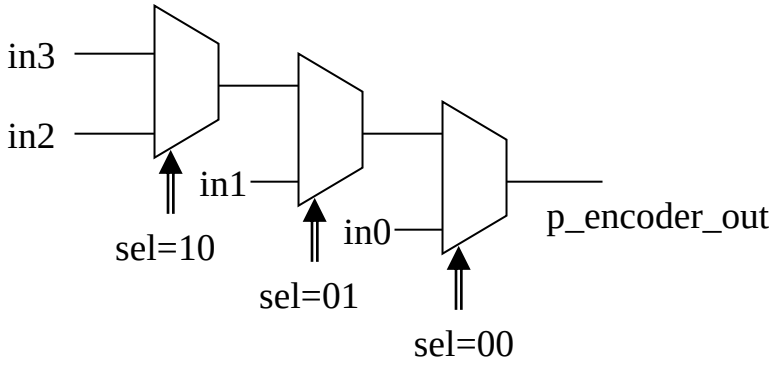
Priority Encoder if-else (?:) v.s. case

```
module mux (in0, in1, in2, in3, sel, mux_out);
          input     in0, in1, in2, in3;
          input     [1:0] sel;
          output    mux_out;
          reg       mux_out;
          always @(in0 or in1 or in2 or in3 or sel)   begin
                    case (sel)
                      2'b00:  mux_out = in0;
                      2'b01:  mux_out = in1;
                      2'b10:  mux_out = in2;
                      default:mux_out = in3;
                    endcase
          end
endmodule

module p_encoder (in0, in1, in2, in3, sel, p_encoder_out);
          input     in0, in1, in2, in3;
          input     [1:0] sel;
          output    p_encoder_out;
          reg       p_encoder_out;
          always @(in0 or in1 or in2 or in3 or sel)  begin
                    if (sel == 2'b00)
                      p_encoder_out = in0;
                    else if (sel == 2'b01)
                      p_encoder_out = in1;
                    else if (sel == 2'b10)
                       p_encoder_out = in2;
                    else p_encoder_out = in3;
          end
endmodule
```



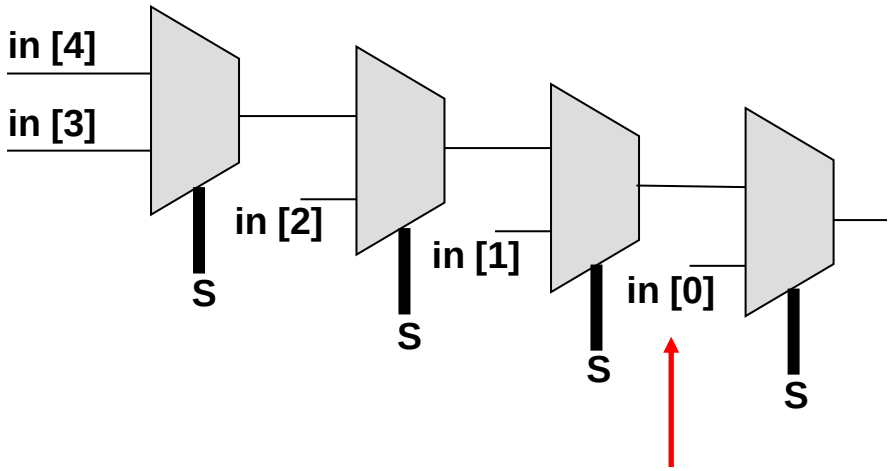
case 合成的電路比較快

Generally, If-Else is **slower** unless you intend to build a priority encoder!



Priority Encoder “if-then-else” When to use?

```
always_comb
begin
 if (sel == 3'h0)
       out = in[0];
   else if (sel == 3'h1)
       out = in[1];
   else if (sel == 3'h2)
       out = in[2];
   else if (sel == 3'h3)
       out = in[3];
   else if (sel == 3'h4)
       out = in[4];
   else
       out = in[5];
end
```

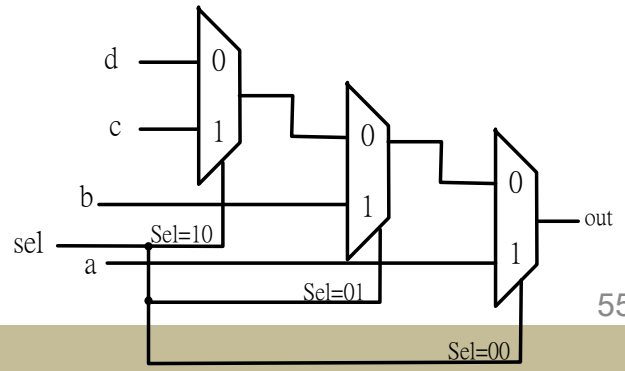


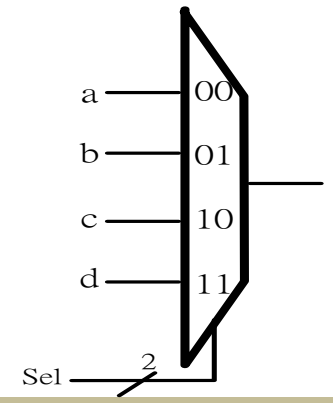
最慢到的輸入放在最靠近輸出

- Assign highest priority to a late arriving critical signal
- Nested “if-then-else” can increase area and delay
- Use “case” statement if possible to describe the same function

TLDR: if-then-else vs. case

- *If-then-else* often infers a cascaded encoder
  - inputs signals with different arrival time
- *case* infers a single-level mux
  - *case* is better if priority encoding is not required
  - *case* is generally simulated faster than *if-then-else*
- *conditional assignment* (? :)
  - infers a mux with slower simulation performance
  - better avoided

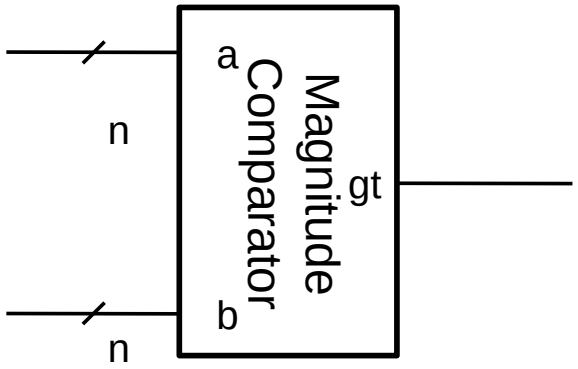


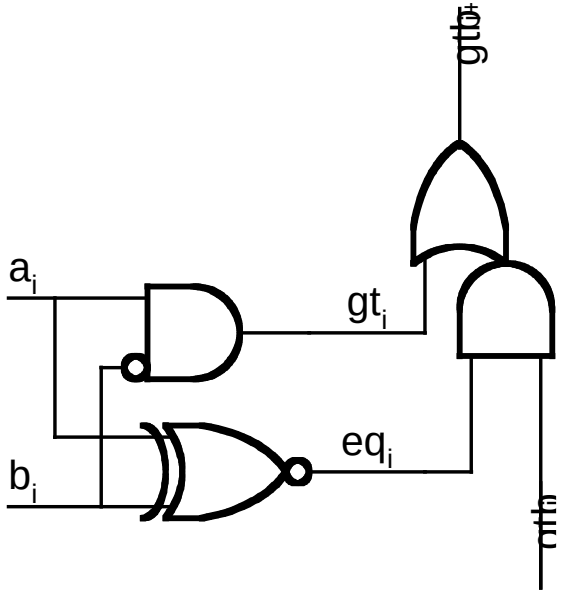


COMPARATORS

Magnitude Comparator

//bit-slice design with leaf cell (from LSB)
//note that initial value to be considered





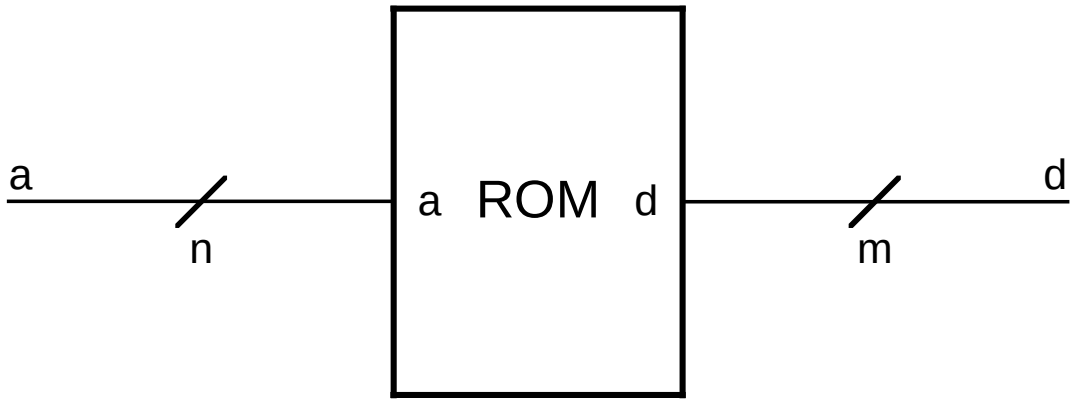

```
// magnitude comparator
module MagComp(a, b, gt) ;
  parameter k=8 ;
  input  [k-1:0] a, b ;
  output gt ;
  wire   [k-1:0] eqi = a ~^ b ;
  wire   [k-1:0] gti = a & ~b ;
  wire   [k:0]   gtb {((eqi[k-1:0] & gtb[k-1:0]) | gti[k-1:0]), 1'b0} ;
  wire   gt = gtb[k] ;//MSB is selected as output gt;
  endmodule
```

```
assign gt = (a > b) ;
```

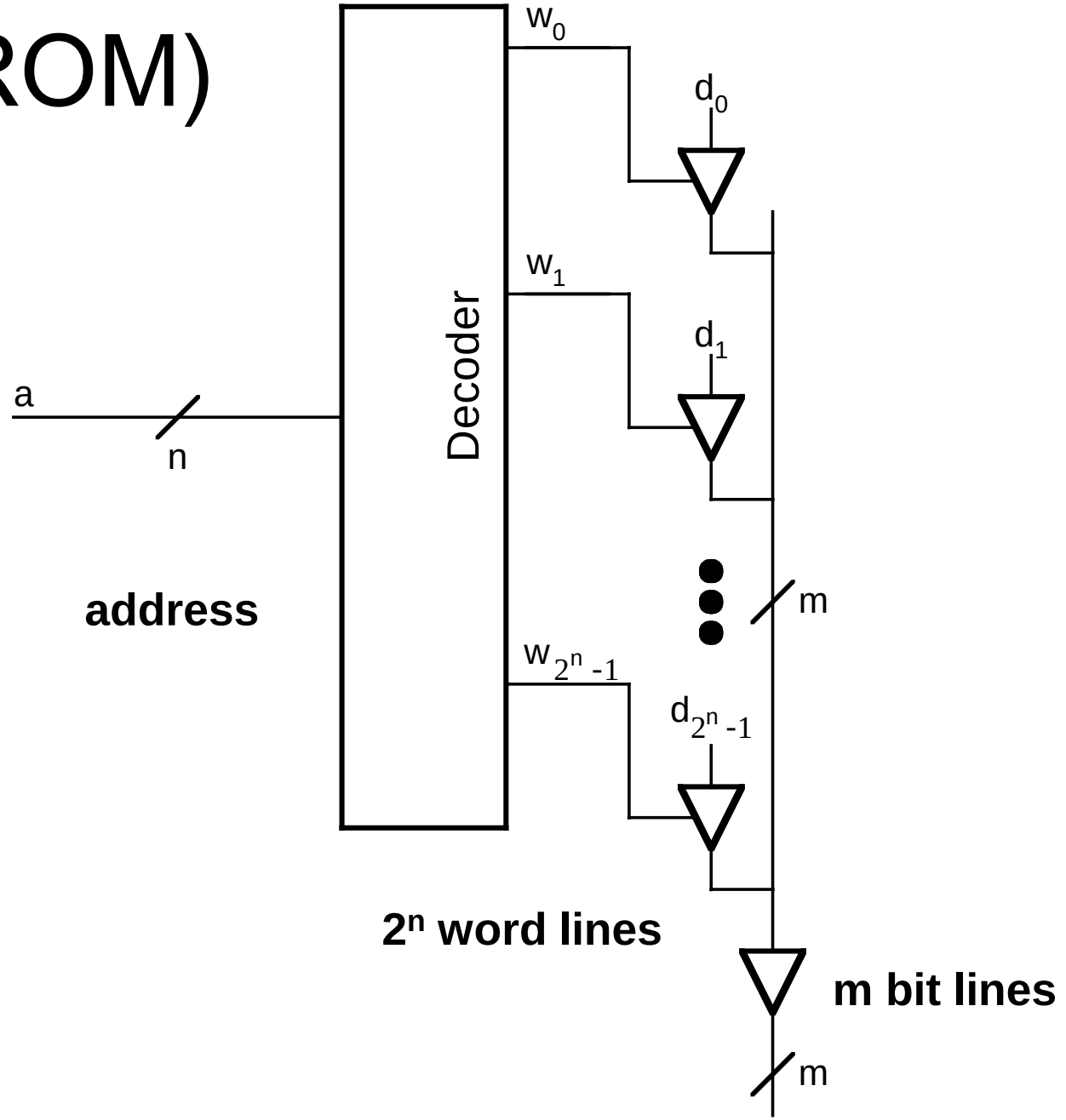
READ ONLY MEMORIES (ROM)
RANDOM ACCESS MEMORY (RAM)

Read-only memory (ROM)

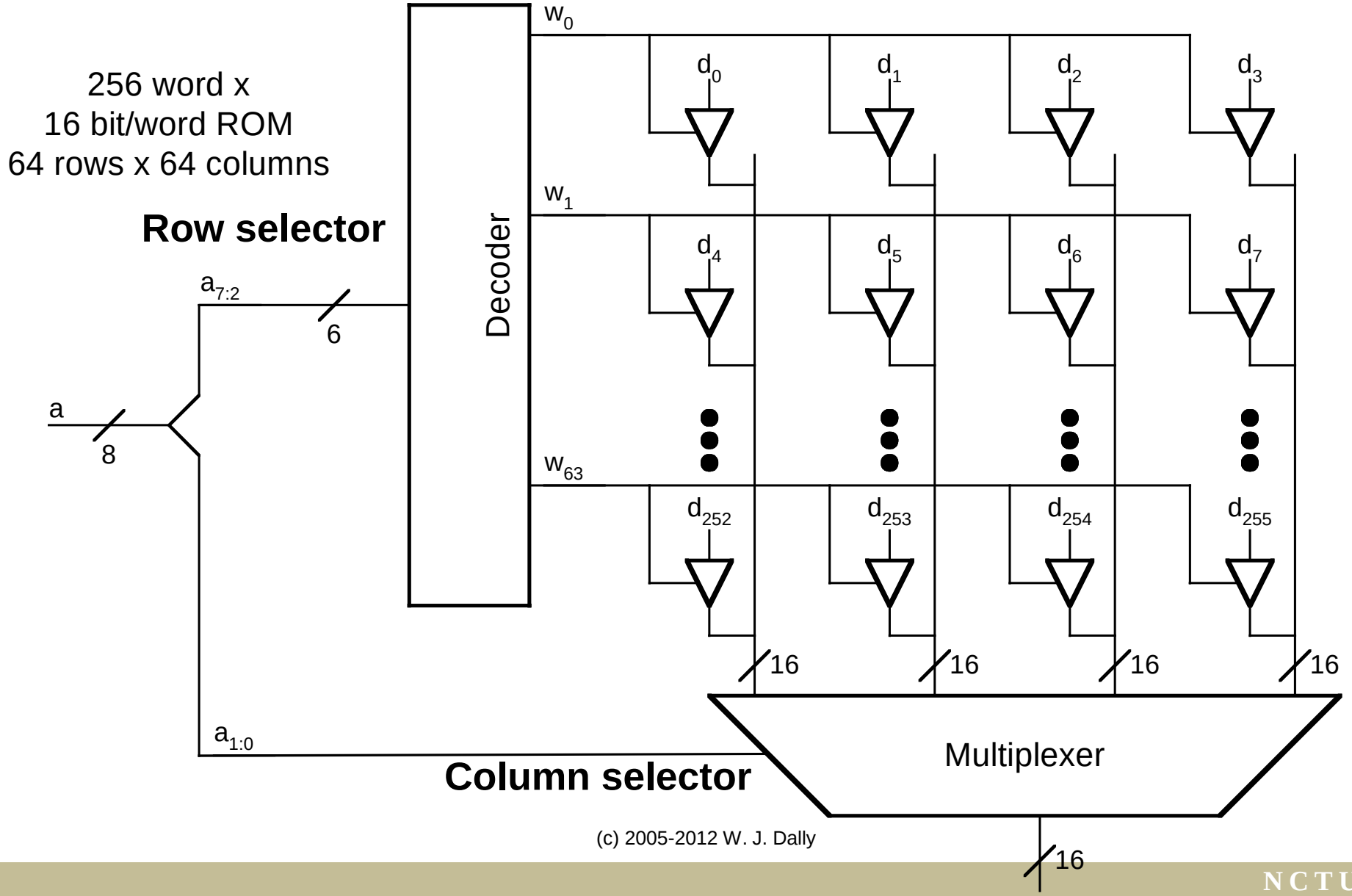
- 存常數值，
- 可實現任意函數
- 需要address decoder，所以大小夠大才划算



d[m] = ROM[a[n]];



2-D array implementation



ROM Model: loading data from file

```
module rom_using_file (
input  wire [7:0] address , // Address input
output wire [7:0] data    , // Data output
input  wire       read_en , // Read Enable
input  wire       ce        // Chip Enable
);

reg [7:0] mem [0:255] ;

assign data = (ce && read_en) ? mem[address] : 8'b0;

initial begin
  $readmemb("memory.list", mem); // memory_list is memory file
end

endmodule
```

memory.list

```
11001
11010

@5 11001000 // 将11001存入mem[5]
@2 11010000 //将11010存入mem[2]

@7 //直至7开始
11001000
00000000
111111111
```

適合大的ROM size

http://www.asic-world.com/code/sv_examples/rom_using_file.sv

ROM with case

```
module rom_using_case (
input  wire [7:0] address , // Address input
output reg  [7:0] data    , // Data output
input  wire       read_en , // Read Enable
input  wire       ce        // Chip Enable
);

always_comb
begin
  case (address)
    0 : data = 10;
    1 : data = 55;
    2 : data = 244;
    3 : data = 0;
    4 : data = 1;
    ……
    13 : data = 8'h90;
    14 : data = 8'h70;
    15 : data = 8'h90;
  endcase
end
endmodule
```

適合很小的ROM size

http://www.asic-world.com/code/sv_examples/rom_using_case.sv

RAM Model

```
module ram_sp_sr_sw #(parameter DATA_WIDTH = 8,
                      parameter ADDR_WIDTH = 8,
                      parameter RAM_DEPTH = (1 << ADDR_WIDTH))(
input  wire                      clk      , // Clock Input
input  wire [ADDR_WIDTH-1:0] address  , // Address Input
inout  wire [DATA_WIDTH-1:0] data     , // Data bi-directional
input  wire                      cs       , // Chip Select
input  wire                      we       , // Write Enable/Read Enable
input  wire                      oe         // Output Enable
);
//--------------Internal variables----------------
reg [DATA_WIDTH-1:0]   data_out ;
// Use Associative array to save memory footprint
typedef reg [ADDR_WIDTH-1:0] mem_addr;
reg [DATA_WIDTH-1:0] mem [mem_addr];
```


http://www.asic-world.com/code/sv_examples/ram_sp_sr_sw.sv

```
//--------------Code Starts Here--------------------
// Tri-State Buffer control
// output : When we = 0, oe = 1, cs = 1
assign data = (cs && oe && !we) ? data_out : 8'bz;

// Memory Write Block
// Write Operation : When we = 1, cs = 1
always @ (posedge clk)
begin : MEM_WRITE
   if ( cs && we ) begin
       mem[address] = data;
   end
end

// Memory Read Block
// Read Operation : When we = 0, oe = 1, cs = 1
always @ (posedge clk)
begin : MEM_READ
    if (cs && !we && oe) begin
         data_out = mem[address];
    end
end

endmodule // End of Module ram_sp_sr_sw
```

Reference

- [Dally] Ch. 7. 8. 9
- [Roth] Ch. 4
- http://www.asic-world.com/index.html