Department of Computer Science
Course number: CSC.T341

2023年度の講義と演習は，学術国際情報センター3階 情報工学系計算機室で実施します．

コンピュータ論理設計
Computer Logic Design

3. ハードウェア記述言語：組合せ回路 (2)
Hardware Description Language: Combinational Circuit (2)

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
講義：月曜日 10:45-12:25, 木曜日 10:45-12:25

Ver. 2023-04-17a

Sample Verilog HDL code

- ACRi Room のサーバーにリモートデスクトップでログインする.
  - /home/tu_kise/cld/lec3/ にサンプルのコードがあるので，Ubuntu のターミナルで次のコマンドを入力して，自分のディレクトリにコピーする.
  - /home/tu_kise は automount のディレクトリなので，アクセスしないとファイルが見えない．tabキーによる補完がうまく動作しないことがあるので注意する.

```
$ cd
$ mkdir cld
$ cd cld
$ cp /home/tu_kise/cld/lec3/* .
```

- code010.v をシミュレーションするためには.
  - リモートデスクトップで接続した Ubuntu のターミナルで次のコマンドを入力する.
  - コマンド iverilog でコンパイルして，生成される a.out を実行する.

```
$ iverilog code010.v
$ ./a.out
```

code010.v C言語で書いたかもしれないコード

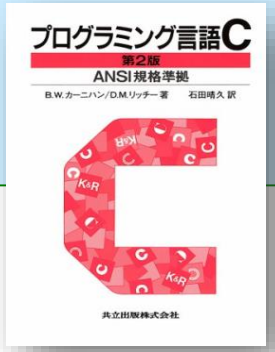
- code010.v をシミュレーションして，その表示を確認すること．
- 整数integerとforループを用いた温度変換プログラムの例．
- 整数型のfahr, celsiusを定義．
- C言語の様に演算子++は使えない．fahr++ という記述はエラーとなるので注意．

```
$ iverilog code010.v
$ ./a.out
```

code010.v

```
module main ();
  integer fahr, celsius;
  initial begin
    for (fahr = 0; fahr <= 300; fahr = fahr + 20) begin
      celsius = 5*(fahr-32) / 9;
      $display("%3d %6d", fahr, celsius);
    end
  end
endmodule
```

```
  0    -17
 20     -6
 40      4
 60     15
 80     26
100     37
120     48
140     60
160     71
180     82
200     93
220    104
240    115
260    126
280    137
300    148
```



スライドPDFからコピーすると正しく動作しないことがあるので，コードは /home/tu_kise/ からコピーしたものを使うこと．

Some rules for following lectures and exercises

- モジュールの名前には m_ から始まる名前を使う. wire型の信号の名前には w_ から始まる名前を使う. reg型の名前には r_ から始まる名前を使う.
- シミュレーションの最上位のモジュール(トップモジュール)には m_top という名前を使う.
  - $display などのシステムタスクは m_top の中でしか用いてはいけない.
- 論理合成のトップモジュールには m_main という名前を使う.
- Nameという名前のモジュールのインスタンス名には Nameに数字を付加した名前を使う.

ルールを適用したVerilog HDL記述の例

```
module m_top ();
  reg  r_a, r_b, r_s;
  wire w_c;
  initial begin
    #10 r_s <= 0; r_a <= 0; r_b <= 0;
    #10 r_s <= 0; r_a <= 0; r_b <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %b", $time, r_s, r_a, r_b, w_c);
  m_mux m_mux0 (r_a, r_b, r_s, w_c);
endmodule

module m_mux (w_a, w_b, w_s, w_c);
  input  wire w_a, w_b, w_s;
  output wire w_c;
  assign w_c = w_s ? w_b : w_a;
endmodule
```

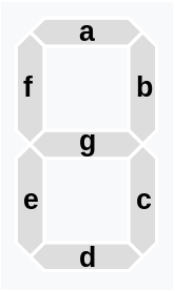
code018.v case文を用いたLEDデコーダ

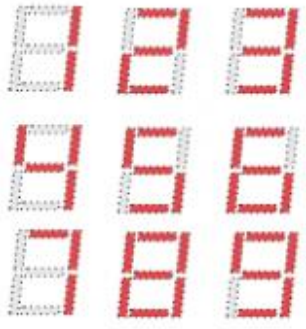
- code018.v をシミュレーションして，その表示を確認すること．
- 0～9を表示する seven-segment LED decoder の例を示す．
- 場合分けの処理を記述するための case文 がある．記述はC言語と同様．
- モジュールm_7segledでは，入力の値により，点灯させるLEDのビットを1とする．
  - r_led の MSBから，LEDのabcdefgのセグメントを割り当てる．

```
module m_7segled (w_in, r_led);
  input  wire [3:0] w_in;
  output reg  [6:0] r_led;
  always @(*) begin
    case (w_in)
      4'd0   : r_led <= 7'b1111110;
      4'd1   : r_led <= 7'b0110000;
      4'd2   : r_led <= 7'b1101101;
      4'd3   : r_led <= 7'b1111001;
      4'd4   : r_led <= 7'b0110011;
      4'd5   : r_led <= 7'b1011011;
      4'd6   : r_led <= 7'b1011111;
      4'd7   : r_led <= 7'b1110000;
      4'd8   : r_led <= 7'b1111111;
      4'd9   : r_led <= 7'b1111011;
      default: r_led <= 7'b0000000;
    endcase
  end
endmodule
```

code018.v

```
module m_top ();
  reg  [3:0] r_in;
  wire [6:0] w_led;
  integer i;
  initial
    for (i=0; i<=15; i=i+1) begin r_in <= i; #10; end
  initial       $display("      abcdefg");
  always@(*) #1 $display(" %x -> %b", r_in, w_led);

  m_7segled m_7segled0 (r_in, w_led);
endmodule
```





```
      abcdefg
 0 -> 1111110
 1 -> 0110000
 2 -> 1101101
 3 -> 1111001
 4 -> 0110011
 5 -> 1011011
 6 -> 1011111
 7 -> 1110000
 8 -> 1111111
 9 -> 1111011
 a -> 0000000
 b -> 0000000
 c -> 0000000
 d -> 0000000
 e -> 0000000
 f -> 0000000
```

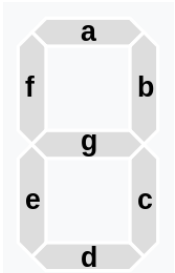
code019.v 3項演算子を用いたLEDデコーダ

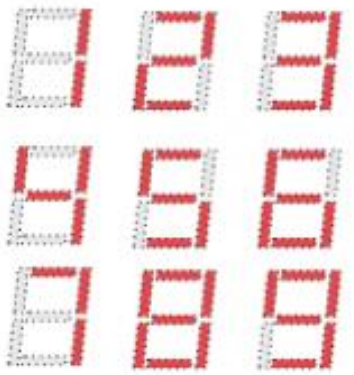
- code019.v をシミュレーションして，その表示を確認すること．
- Seven-segment LED decoder の別の例を示す．
- 関係演算子(==)は等しい時に1'b1となり，そうでなければ1'b0となる．
- code019.v ではreg型は使っていない．このため，m_7segled からは組合せ回路が合成される．
- code018.v の m_7segled から，組合せ回路が合成されるか？順序回路が合成されるか？
  - reg型の信号が常にレジスタに合成されるという訳ではない．

code019.v （m_topの記述はcode18.vと同じ）

```
module m_7segled (w_in, w_led);
  input  wire [3:0] w_in;
  output wire [6:0] w_led;

  assign w_led = (w_in==4'd0) ? 7'b1111110 :
                 (w_in==4'd1) ? 7'b0110000 :
                 (w_in==4'd2) ? 7'b1101101 :
                 (w_in==4'd3) ? 7'b1111001 :
                 (w_in==4'd4) ? 7'b0110011 :
                 (w_in==4'd5) ? 7'b1011011 :
                 (w_in==4'd6) ? 7'b1011111 :
                 (w_in==4'd7) ? 7'b1110000 :
                 (w_in==4'd8) ? 7'b1111111 :
                 (w_in==4'd9) ? 7'b1111011 :
                 7'b0000000;
endmodule
```

```
      abcdefg
0 -> 1111110
1 -> 0110000
2 -> 1101101
3 -> 1111001
4 -> 0110011
5 -> 1011011
6 -> 1011111
7 -> 1110000
8 -> 1111111
9 -> 1111011
a -> 0000000
b -> 0000000
c -> 0000000
d -> 0000000
e -> 0000000
f -> 0000000
```





code020.v case文ではすべての入力を定義する

- code020.v をシミュレーションして，その表示を確認すること．
- Seven-segment LED decoder の別の例を示す．
- code020.v の m_7segled から，組合せ回路が合成されるか？順序回路が合成されるか？
  - w_in が 4'ha の時に，どうして 7'b1111011 が出力されるのか？

code018.v

```
module m_7segled (w_in, r_led);
  input  wire [3:0] w_in;
  output reg  [6:0] r_led;
  always @(*) begin
    case (w_in)
      4'd0   : r_led <= 7'b1111110;
      4'd1   : r_led <= 7'b0110000;
      4'd2   : r_led <= 7'b1101101;
      4'd3   : r_led <= 7'b1111001;
      4'd4   : r_led <= 7'b0110011;
      4'd5   : r_led <= 7'b1011011;
      4'd6   : r_led <= 7'b1011111;
      4'd7   : r_led <= 7'b1110000;
      4'd8   : r_led <= 7'b1111111;
      4'd9   : r_led <= 7'b1111011;
      default: r_led <= 7'b0000000;
    endcase
  end
endmodule
```

```
      abcdefg
0 -> 1111110
1 -> 0110000
2 -> 1101101
3 -> 1111001
4 -> 0110011
5 -> 1011011
6 -> 1011111
7 -> 1110000
8 -> 1111111
9 -> 1111011
a -> 0000000
b -> 0000000
c -> 0000000
d -> 0000000
e -> 0000000
f -> 0000000
```

code020.v

```
module m_7segled (w_in, r_led);
  input  wire [3:0] w_in;
  output reg  [6:0] r_led;
  always @(*) begin
    case (w_in)
      4'd0   : r_led <= 7'b1111110;
      4'd1   : r_led <= 7'b0110000;
      4'd2   : r_led <= 7'b1101101;
      4'd3   : r_led <= 7'b1111001;
      4'd4   : r_led <= 7'b0110011;
      4'd5   : r_led <= 7'b1011011;
      4'd6   : r_led <= 7'b1011111;
      4'd7   : r_led <= 7'b1110000;
      4'd8   : r_led <= 7'b1111111;
      4'd9   : r_led <= 7'b1111011;
    endcase
  end
endmodule
```

```
      abcdefg
0 -> 1111110
1 -> 0110000
2 -> 1101101
3 -> 1111001
4 -> 0110011
5 -> 1011011
6 -> 1011111
7 -> 1110000
8 -> 1111111
9 -> 1111011
a -> 1111011
b -> 1111011
c -> 1111011
d -> 1111011
e -> 1111011
f -> 1111011
```

code021.v if文を用いたLEDデコーダ

- code021.v をシミュレーションして，その表示を確認すること．
- Seven-segment LED decoder の別の例を示す．
- case文ではなく，if文 (if else) を用いて記述することもできる．
  - code18.v と code21.v は同じ出力となる．

code018.v

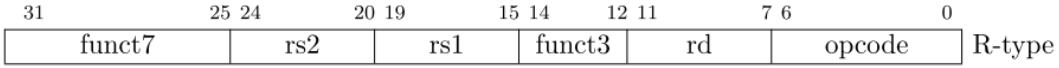
```
module m_7segled (w_in, r_led);
  input  wire [3:0] w_in;
  output reg  [6:0] r_led;
  always @(*) begin
    case (w_in)
      4'd0   : r_led <= 7'b1111110;
      4'd1   : r_led <= 7'b0110000;
      4'd2   : r_led <= 7'b1101101;
      4'd3   : r_led <= 7'b1111001;
      4'd4   : r_led <= 7'b0110011;
      4'd5   : r_led <= 7'b1011011;
      4'd6   : r_led <= 7'b1011111;
      4'd7   : r_led <= 7'b1110000;
      4'd8   : r_led <= 7'b1111111;
      4'd9   : r_led <= 7'b1111011;
      default: r_led <= 7'b0000000;
    endcase
  end
endmodule
```

code021.v

```
module m_7segled (w_in, r_led);
  input  wire [3:0] w_in;
  output reg  [6:0] r_led;
  always @(*) begin
    if      (w_in==4'd0) r_led <= 7'b1111110;
    else if (w_in==4'd1) r_led <= 7'b0110000;
    else if (w_in==4'd2) r_led <= 7'b1101101;
    else if (w_in==4'd3) r_led <= 7'b1111001;
    else if (w_in==4'd4) r_led <= 7'b0110011;
    else if (w_in==4'd5) r_led <= 7'b1011011;
    else if (w_in==4'd6) r_led <= 7'b1011111;
    else if (w_in==4'd7) r_led <= 7'b1110000;
    else if (w_in==4'd8) r_led <= 7'b1111111;
    else if (w_in==4'd9) r_led <= 7'b1111011;
    else                 r_led <= 7'b0000000;
  end
endmodule
```

code022.v ビット選択

- code022.v をシミュレーションして，その表示を確認すること．
- ビット選択の例を示す．バスは多ビットの束で表現されるので，バスから選択するビットの範囲を指定する．RISC-Vアーキテクチャの機械命令で用いられるR形式の命令から各フィールドを選択する例．




```
module m_top ();
  reg  [31:0] r_ir = 32'h12345678;
  wire [6:0]  w_fct7, w_op;
  wire [4:0]  w_rs2, w_rs1, w_rd;
  wire [2:0]  w_fct3;
  initial begin #1
    $display(" %x -> %x %x %x %x %x %x", r_ir, w_fct7, w_rs2, w_rs1, w_fct3, w_rd, w_op);
    $display(" %x -> %d %d %d %d %d %d", r_ir, w_fct7, w_rs2, w_rs1, w_fct3, w_rd, w_op);
  end
  m_decode m_d0 (r_ir, w_fct7, w_rs2, w_rs1, w_fct3, w_rd, w_op);
endmodule

module m_decode (w_ir, w_fct7, w_rs2, w_rs1, w_fct3, w_rd, w_op);
  input  wire [31:0] w_ir;
  output wire [6:0]  w_fct7, w_op;
  output wire [4:0]  w_rs2, w_rs1, w_rd;
  output wire [2:0]  w_fct3;
  assign w_fct7 = w_ir[31:25];
  assign w_rs2  = w_ir[24:20];
  assign w_rs1  = w_ir[19:15];
  assign w_fct3 = w_ir[14:12];
  assign w_rd   = w_ir[11:7];
  assign w_op   = w_ir[6:0];
endmodule
```

```
12345678 -> 09 03 08 5 0c 78
12345678 ->   9  3  8 5 12 120
```

Simulation output

code023.v ビットの連結と複製

- code023.v をシミュレーションして，その表示を確認すること．
- ビットの連結 (concatenation) の例を示す．
- 連結演算子 ({ } , 波括弧 curly brackets) は，幾つかの信号を連結してビット長の大きい1つのバスにできる．4ビットの信号 w_a, w_b を連結するには {w_a, w_b} と記述する．4ビットの信号 w_a, w_b, w_c を連結するには {w_a, w_b, w_c} と記述する．
- ある信号を複製してビット長の大きい1つのバスにできる．例えば，4ビットの信号 w_a を3回複製して連結するには {3{w_a}} と記述する．例えば，{4{w_a}} と {w_a, w_a, w_a, w_a} は同じビット列となる．
- 最後の例で示した下位ビットのMSBを複製して上位ビットを補填する操作は，2の補数で表現された符号付きの整数を符号拡張する際に用いられる．後の講義で解説する．

code023.v

```
module m_top ();
  reg [3:0] r_a = 4'b1001;
  reg [3:0] r_b = 4'b0101;
  reg [3:0] r_c = 4'b1111;
  initial #1 begin
    $display("%b", {r_a, r_b});
    $display("%b", {r_a, r_b, r_c});
    $display("%b", {2{r_a}});
    $display("%b", {3{r_a}});
    $display("%b", {4{r_a}});
    $display("%b", {{4{r_a[3]}}, r_a});
    $display("%b", {{4{r_b[3]}}, r_b});
  end
endmodule
```

Simulation output

```
10010101
100101011111
10011001
100110011001
1001100110011001
11111001
00000101
```

w_bの値を赤色で強調した．

code024.v 関係演算子

- code024.v をシミュレーションして，その表示を確認すること．
- 関係演算子 (>, <, >=, <=, ==, !=) の例を示す．
- 例えば，w_a >= w_b は，w_a の値が w_b の値以上であれば 1'b1，そうでなければ 1'b0 となる．
- C言語と同様．
- ノンブロッキング代入の演算子 <= と 関係演算子 <= は同じ記述だが，文法的に区別できる．
  この演習では (w_a >= w_b) の様に，関係演算子の比較の前後に ( ) を追加して明示的に区別する．

code024.v

```
module m_top ();
  reg [3:0] r_a = 4'd7;
  reg [3:0] r_b = 4'd8;
  initial #1 begin
    $display("%b", (r_a> r_b));
    $display("%b", (r_a< r_b));
    $display("%b", (r_a>=r_b));
    $display("%b", (r_a<=r_b));
    $display("%b", (r_a==r_b));
    $display("%b", (r_a!=r_b));
  end
endmodule
```

Simulation output

```
0
1
0
1
0
1
```

code025.v 論理シフト演算

- code025.v をシミュレーションして，その表示を確認すること．
- 論理シフト演算 (>>, <<) の例を示す．
- C言語と同様．
- 例えば，w_a << 3 は，w_a の値を左に3ビット移動させ，下位の3ビットは0となる．同様に，w_b >> 2 では，w_bの値を右に2ビット移動させ，上位の2ビットは0となる．
- 論理シフト演算では，シフトさせるビット数としてワイヤ型やレジスタ型の信号を用いてもよい．

code025.v

```
module m_top ();
  reg [7:0] r_a = 8'b11110101;
  reg [2:0] r_s = 3'd3;
  initial #1 begin
    $display("%b", (r_a>>0));
    $display("%b", (r_a>>1));
    $display("%b", (r_a<<1));
    $display("%b", (r_a>>r_s));
    $display("%b", (r_a<<r_s));
  end
endmodule
```

Simulation output

```
11110101
01111010
11101010
00011110
10101000
```

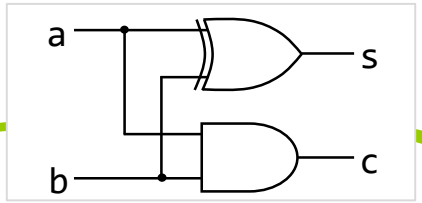
code026.v リダクション演算子

- code026.v をシミュレーションして，その表示を確認すること．
- リダクション演算子(&, |, ^) の例．
  例えば ^ はバスの全てのビットの排他的論理和となる．コメントを参照．

code026.v

```
module m_top ();
  reg  [4:0]  r_btn;
  wire [2:0]  w_led;
  initial begin
    #10 r_btn <= 5'b00000;
    #10 r_btn <= 5'b11111;
    #10 r_btn <= 5'b00010;
  end
  always@(*) #1 $display(" %b -> %b", r_btn, w_led);
  m_main m_main0 (r_btn, w_led);
endmodule

module m_main (w_btn, w_led);
  input  wire [4:0] w_btn;
  output wire [2:0] w_led;
  assign w_led[0]  = &w_btn; // same as w_btn[0] & w_btn[1] & w_btn[2] & w_btn[3] & w_btn[4]
  assign w_led[1]  = |w_btn; // same as w_btn[0] | w_btn[1] | w_btn[2] | w_btn[3] | w_btn[4]
  assign w_led[2]  = ^w_btn; // same as w_btn[0] ^ w_btn[1] ^ w_btn[2] ^ w_btn[3] ^ w_btn[4]
endmodule
```

code071.v default_nettype の追加



- code071.v と code072.v を iverilog でコンパイルすること.
- 入力ミスで，定義していない信号 M_s を用いている．定義していない信号を使うと，1ビットのwireとして扱われる．定義していない信号の使用をエラーにするにはソースコードの最初に `default_nettype none を追加すれば良い.
- code072.v ではエラーとなる．今後，すべてのソースコードに `default_nettype none を追加すること.

code071.v

```
module m_top ();
  reg  r_a, r_b;
  wire w_c, w_s;
  initial begin
    #10 r_a <= 0; r_b <= 0;
    #10 r_a <= 0; r_b <= 1;
    #10 r_a <= 1; r_b <= 0;
    #10 r_a <= 1; r_b <= 1;
  end
  always@(*) #1
    $write("%2d: %d %d -> %b %b¥n",
           $time, r_a, r_b, w_c, w_s);
  m_HA m_HA0 (r_a, r_b, w_c, w_s);
endmodule

module m_HA (w_a, w_b, w_c, w_s);
  input  wire w_a, w_b;
  output wire w_c, w_s;
  assign w_c = w_a & w_b;
  assign M_s = w_a ^ w_b;
endmodule
```

```
 1: x x -> x z
11: 0 0 -> 0 z
21: 0 1 -> 0 z
31: 1 0 -> 0 z
41: 1 1 -> 1 z
```

code072.v

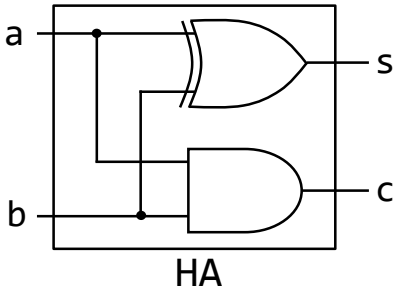
```
`default_nettype none

module m_top ();
  reg  r_a, r_b;
  wire w_c, w_s;
  initial begin
    #10 r_a <= 0; r_b <= 0;
    #10 r_a <= 0; r_b <= 1;
    #10 r_a <= 1; r_b <= 0;
    #10 r_a <= 1; r_b <= 1;
  end
  always@(*) #1
    $write("%2d: %d %d -> %b %b¥n",
           $time, r_a, r_b, w_c, w_s);
  m_HA m_HA0 (r_a, r_b, w_c, w_s);
endmodule

module m_HA (w_a, w_b, w_c, w_s);
  input  wire w_a, w_b;
  output wire w_c, w_s;
  assign w_c = w_a & w_b;
  assign M_s = w_a ^ w_b;
endmodule
```

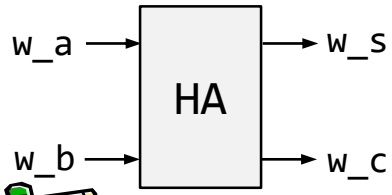
code073.v 半加算器(Half Adder)

- code073.v をシミュレーションして，その表示を確認すること.
- Half Adder, HA (半加算器)の回路とその記述の例を示す.
  - 1ビットの入力 a, b の加算をおこなう回路.
  - 入力 a, b と出力 c (carry out), s (sum) とするtruth table(真理値表)を table073 に示す.

table073

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





code073.v

```
`default_nettype none

module m_top ();
  reg  r_a, r_b;
  wire w_c, w_s;
  initial begin
    #10 r_a <= 0; r_b <= 0;
    #10 r_a <= 0; r_b <= 1;
    #10 r_a <= 1; r_b <= 0;
    #10 r_a <= 1; r_b <= 1;
  end
  always@(*) #1
    $write("%2d: %d %d -> %b %b¥n", $time, r_a, r_b, w_c, w_s);
  m_HA m_HA0 (r_a, r_b, w_s, w_c);
endmodule

module m_HA (w_a, w_b, w_s, w_c);
  input  wire w_a, w_b;
  output wire w_s, w_c;
  assign w_c = w_a & w_b;
  assign w_s = w_a ^ w_b;
endmodule
```

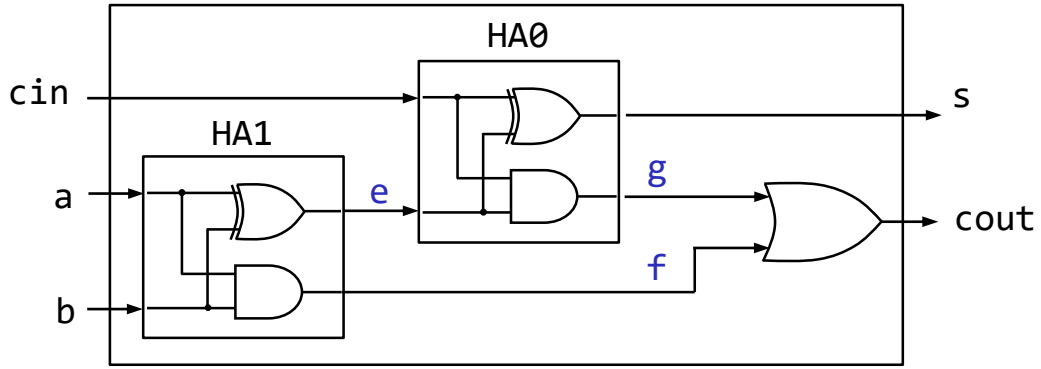
```
11: 0 0 -> 0 0
21: 0 1 -> 0 1
31: 1 0 -> 0 1
41: 1 1 -> 1 0
```

code074.v 全加算器(Full Adder)

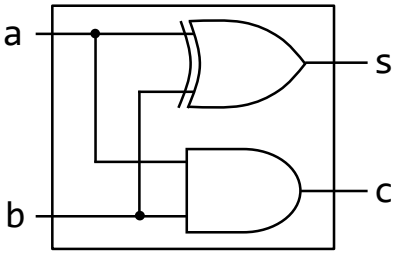
- Full Adder, FA (全加算器)の真理値表と回路を示す.
  - 1ビットの入力 a, b, cin の加算をおこなう回路.
  - 入力 a, b, cin (carry in), 出力 cout (carry out), s (sum) とする真理値表 (truth table) を示す.

table074

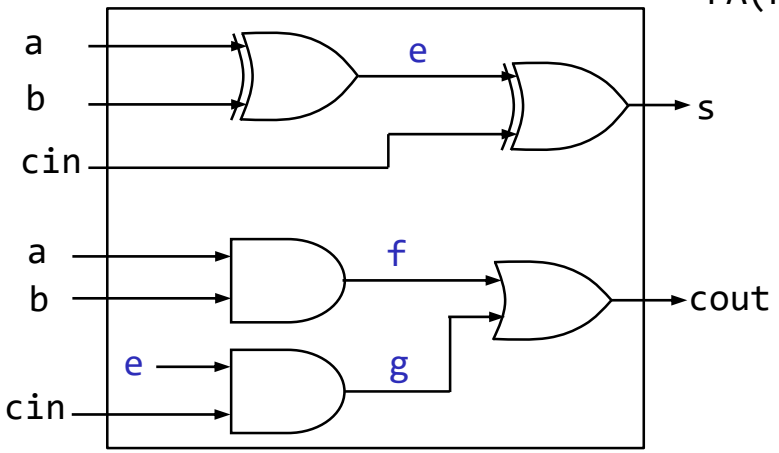
a	b	cin	cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



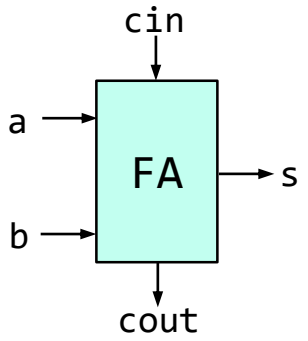
FA(Full Adder)



HA



FA(Full Adder)

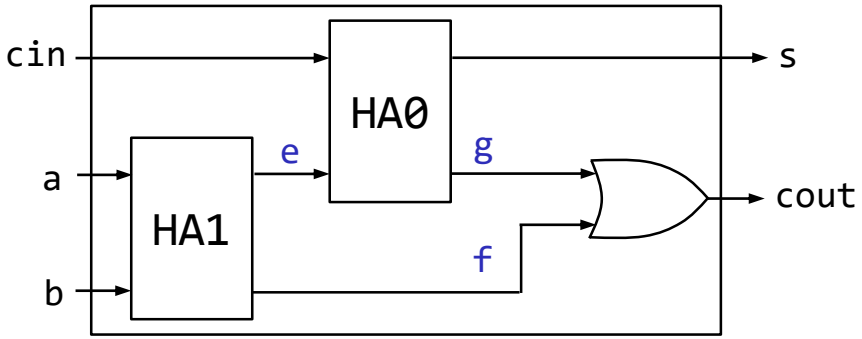


code074.v 全加算器(Full Adder)

- 全加算器として動作するように code074.v の青色の部分を変更し，シミュレーションで確認すること．
- Full Adder, FA (全加算器) の回路とその記述の一部を示す．
- 次のスライドにヒントあり．

table074

a	b	cin	cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

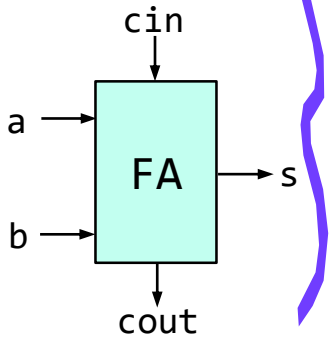


FA(Full Adder)

code074.v

```
module m_top ();
  reg  r_a, r_b, r_cin;
  wire w_s, w_cout;
  initial begin
    #10 r_a <= 0; r_b <= 0;  r_cin <= 0;
    #10 r_a <= 0; r_b <= 0;  r_cin <= 1;
    #10 r_a <= 0; r_b <= 1;  r_cin <= 0;
    #10 r_a <= 0; r_b <= 1;  r_cin <= 1;
    #10 r_a <= 1; r_b <= 0;  r_cin <= 0;
    #10 r_a <= 1; r_b <= 0;  r_cin <= 1;
    #10 r_a <= 1; r_b <= 1;  r_cin <= 0;
    #10 r_a <= 1; r_b <= 1;  r_cin <= 1;
  end
  always@(*) #1 $write("%d %d %d -> %b %b¥n",
                  r_a, r_b, r_cin, w_cout, w_s);
  m_FA m_FA0 (r_a, r_b, r_cin, w_s, w_cout);
endmodule

module m_FA (w_a, w_b, w_cin, w_s, w_cout);
  /* Please describe here by yourself */
endmodule

module m_HA (w_a, w_b, w_s, w_c);
  input  wire w_a, w_b;
  output wire w_s, w_c;
  assign w_c = w_a & w_b;
  assign w_s = w_a ^ w_b;
endmodule
```

```
0 0 0 -> 0 0
0 0 1 -> 0 1
0 1 0 -> 0 1
0 1 1 -> 1 0
1 0 0 -> 0 1
1 0 1 -> 1 0
1 1 0 -> 1 0
1 1 1 -> 1 1
```



ヒント code074.v 全加算器(Full Adder)

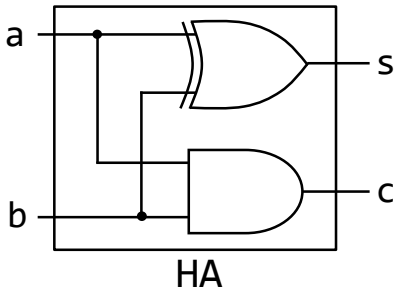
- 全加算器として動作するように code074.v の青色の部分を変更し，シミュレーションで確認すること．
- Full Adder, FA (全加算器) の回路とその記述の一部を示す．

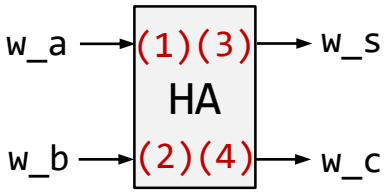
code074.v の一部

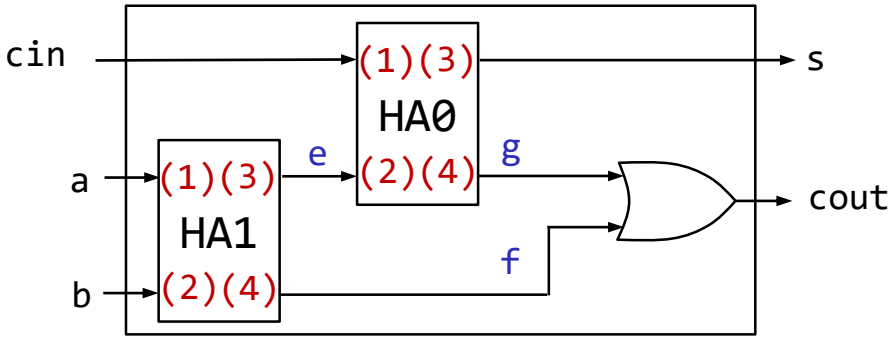
```
module m_FA (w_a, w_b, w_cin, w_s, w_cout);
  /* Please describe here by yourself */
endmodule
            (1) (2) (3) (4)
module m_HA (w_a, w_b, w_s, w_c);
  input  wire w_a, w_b;
  output wire w_s, w_c;
  assign w_c = w_a & w_b;
  assign w_s = w_a ^ w_b;
endmodule
```

ヒント: 少し記述を追加した code074.v の一部

```
module m_FA (w_a, w_b, w_cin, w_s, w_cout);
   input wire w_a, w_b, w_cin;
   output wire w_s, w_cout;
   wire w_e, w_f, w_g;
   m_HA HA0 ( /* connect wires here */ );
   m_HA HA1 ( /* connect wires here */ );
   assign w_cout = w_f | w_g;
endmodule

module m_HA (w_a, w_b, w_s, w_c);
  input  wire w_a, w_b;
  output wire w_s, w_c;
  assign w_c = w_a & w_b;
  assign w_s = w_a ^ w_b;
endmodule
```

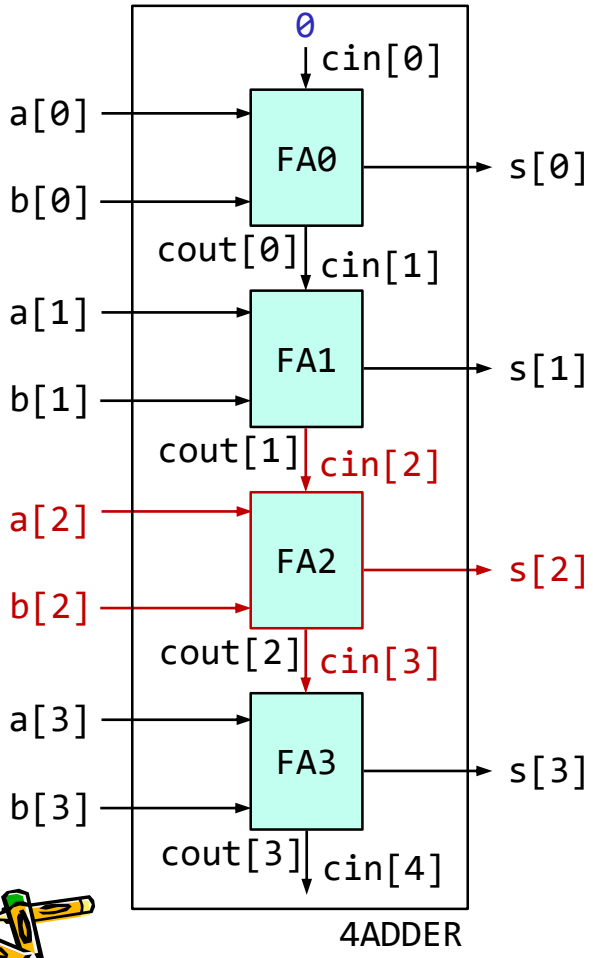






code075.v 4-bit Ripple Carry Adder

- code075.v をシミュレーションして，その表示を確認すること．
- 4-bit Adderの回路とその記述の例を示す．この構成の加算器は順次桁上げ加算器 (Ripple Carry Adder) と呼ばれる．



code075.v

```
module m_top ();
  reg  [3:0] r_a, r_b;
  wire [3:0] w_s;
  initial begin
    #10 r_a <= 3; r_b <= 4;
    #10 r_a <= 1; r_b <= 9;
    #10 r_a <= 8; r_b <= 9;
  end
  always@(*) #1 $write("%2d %2d -> %2d¥n", r_a, r_b, w_s);
  m_4ADDER m_4ADDER0 (r_a, r_b, w_s);
endmodule

module m_4ADDER (w_a, w_b, w_s);
  input  wire [3:0] w_a, w_b;
  output wire [3:0] w_s;
  wire        [4:0] w_cin;
  assign w_cin[0] = 0;
  m_FA FA0(w_a[0], w_b[0], w_cin[0], w_s[0], w_cin[1]);
  m_FA FA1(w_a[1], w_b[1], w_cin[1], w_s[1], w_cin[2]);
  m_FA FA2(w_a[2], w_b[2], w_cin[2], w_s[2], w_cin[3]);
  m_FA FA3(w_a[3], w_b[3], w_cin[3], w_s[3], w_cin[4]);
endmodule

module m_FA (w_a, w_b, w_cin, w_s, w_cout);
  input wire w_a, w_b, w_cin;
  output wire w_s, w_cout;
  wire [1:0] w_sum = w_a + w_b + w_cin;
  assign w_s = w_sum[0];
  assign w_cout = w_sum[1];
endmodule
```

```
 3  4 ->  7
 1  9 -> 10
 8  9 ->  1
```

code076.v 32-bit Ripple Carry Adder

- code076.v をシミュレーションして，その表示を確認すること．
- 32-bit Adderの記述の例を示す．
  - generate を使うことで，ループによる複数モジュールのインスタンス化や接続ができる．
  - for の最後の：Gen で，インスタンス名を Gen に指定する．
  - この例では，Gen[0].m_FA0, Gen[1].m_FA0, Gen[2].m_FA0, … となる．

code076.v

```
module m_top ();
  reg  [31:0] r_a, r_b;
  wire [31:0] w_s;
  initial begin
    #10 r_a <=  321; r_b <= 4444;
    #10 r_a <= 1024; r_b <= 2048;
  end
  always@(*) #1 $write("%4d %4d -> %4d¥n", r_a, r_b, w_s);
  m_ADDER m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER (w_a, w_b, w_s);
  input  wire [31:0] w_a, w_b;
  output wire [31:0] w_s;
  wire [32:0] w_cin;
  assign w_cin[0] = 0;
  generate genvar g;
    for (g = 0; g < 32; g = g + 1) begin : Gen
      m_FA m_FA0(w_a[g], w_b[g], w_cin[g], w_s[g], w_cin[g+1]);
    end
  endgenerate
endmodule
```

```
 321 4444 -> 4765
1024 2048 -> 3072
```

code077.v n-bit Ripple Carry Adder

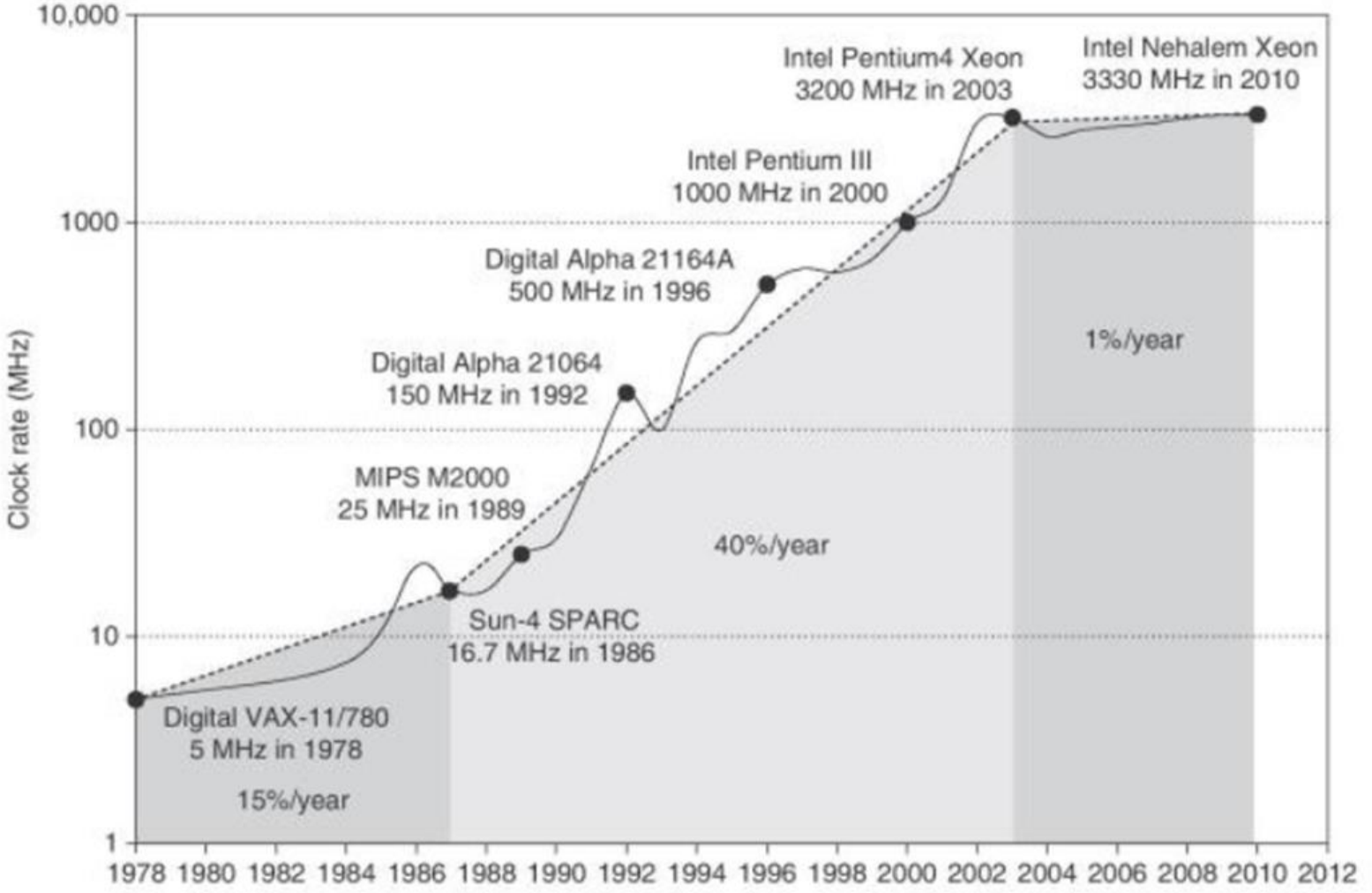
- code077.v をシミュレーションして，その表示を確認すること．
- n-bit Adderの記述の例を示す．
  - define を用いた記述の例．D_N の値を変更するだけで，加算器のビット幅を変更できる．
  - この演習では，defineにより定義される定数の名前は D_ から始まるものとする．．

code077.v

```
`define D_N 5

module m_top ();
  reg  [`D_N-1:0] r_a, r_b;
  wire [`D_N-1:0] w_s;
  initial begin
    #10 r_a <=  321; r_b <= 4444;
    #10 r_a <= 1024; r_b <= 2048;
  end
  always@(*) #1 $write("%4d %4d -> %4d¥n", r_a, r_b, w_s);
  m_ADDER m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER (w_a, w_b, w_s);
  input  wire [`D_N-1:0] w_a, w_b;
  output wire [`D_N-1:0] w_s;
  wire [`D_N:0] w_cin;
  assign w_cin[0] = 0;
  generate genvar g;
    for (g = 0; g < `D_N; g = g + 1) begin : Gen
      m_FA m_FA0(w_a[g], w_b[g], w_cin[g], w_s[g], w_cin[g+1]);
    end
  endgenerate
endmodule
```

```
   1   28 ->   29
   0    0 ->    0
```

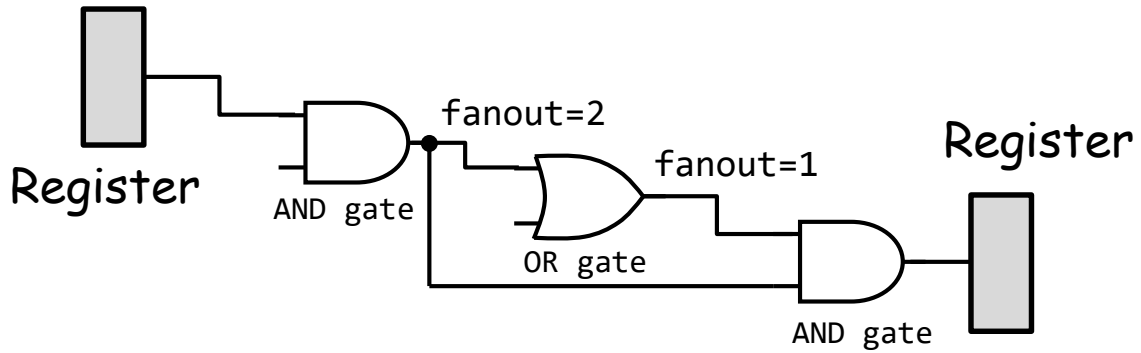
Growth in clock rate of microprocessors



From CAQA 5th edition

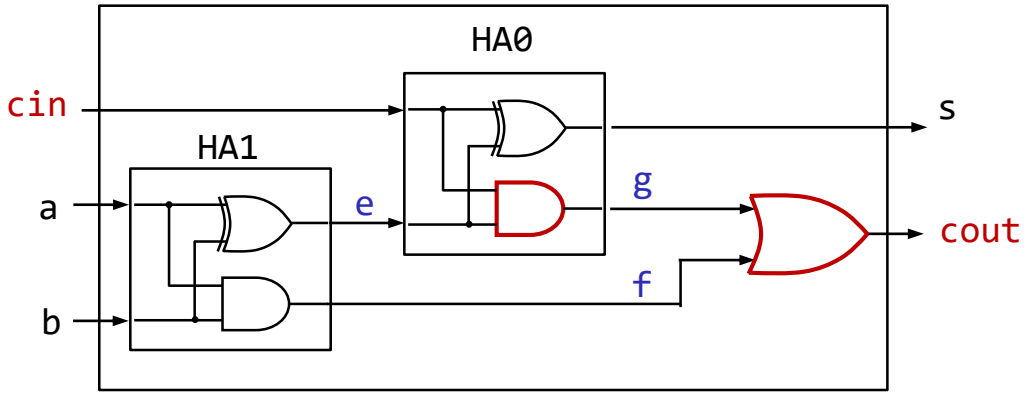
Clock rate is mainly determined by

- Switching speed of gates (transistors)
- The number of levels of gates
  - The maximum number of gates cascaded in series in any combinational logics.
  - In this example, the number of levels of gates is 3.
- Wiring delay and fanout
- The slowest of all paths is called the critical path

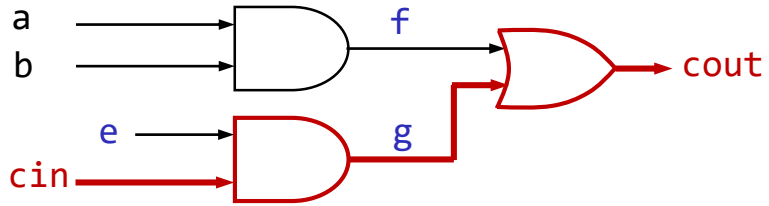


code078.v n-bit Ripple Carry Adder のクリティカルパス

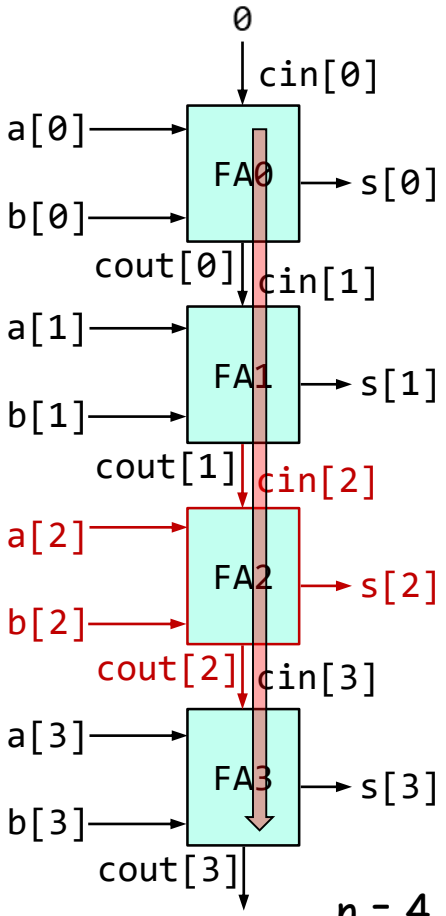
- The carry out signal (w_cout) from the carry in signal (w_cin) takes two gate delays per bit.



FA(Full Adder)

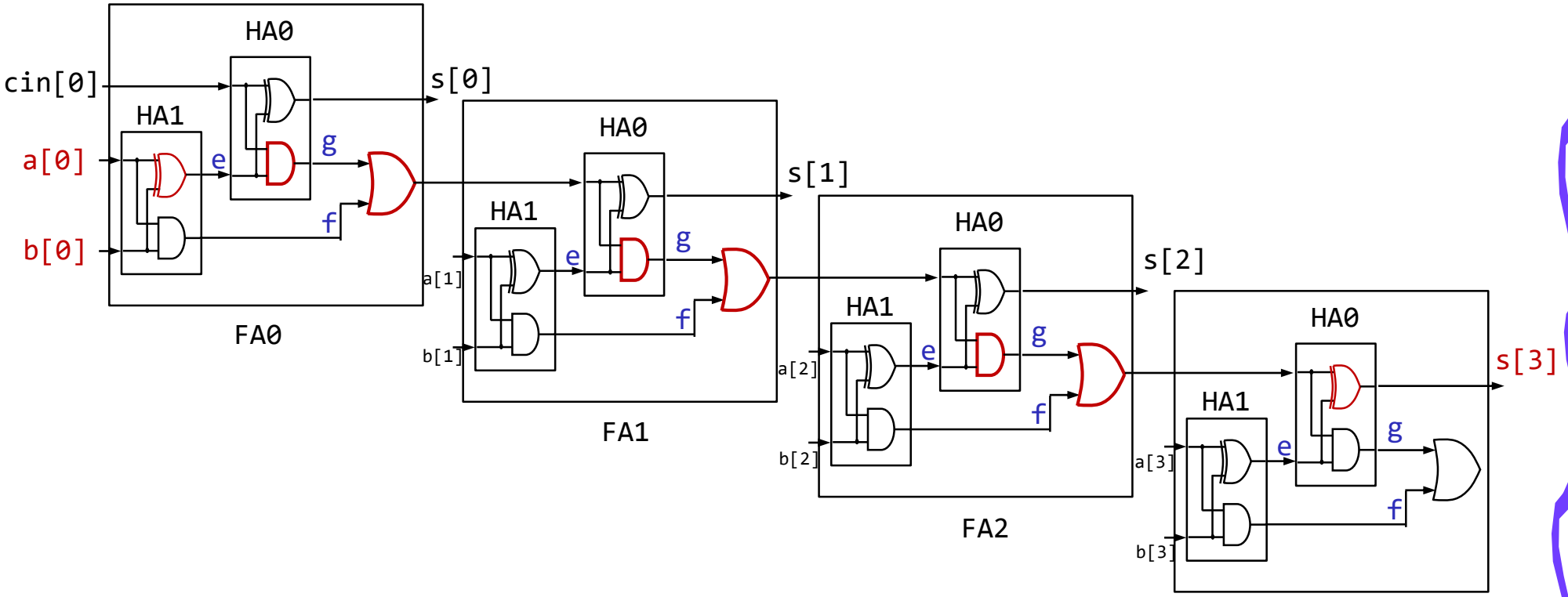


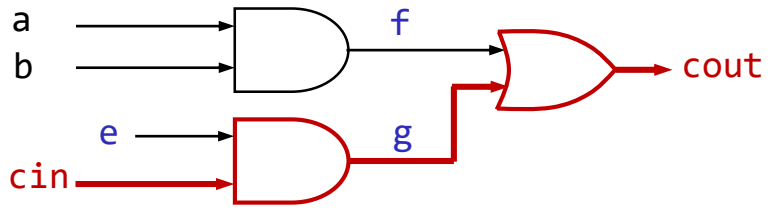
two gate delays



n = 4 の構成

code078.v 4-bit Ripple Carry Adder のクリティカルパス





two gate delays per bit

加算器のクリティカルパスの遅延を計測する

- code078.v の m_FA の青色の部分を，code074.v と同様に変更すること．

code078.v

```
`define D_N 32

module m_main (w_clk, w_a, w_b, w_dout);
  input  wire w_clk, w_a, w_b;
  output wire w_dout;
  reg [`D_N-1:0] r_a=0, r_b=0, r_s=0;
  wire [`D_N-1:0] w_s;
  assign w_dout = ^r_s;
  always@(posedge w_clk) begin
    r_a <= {w_a, r_a[`D_N-1:1]};
    r_b <= {w_b, r_b[`D_N-1:1]};
    r_s <= w_s;
  end
  m_ADDER m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER (w_a, w_b, w_s);
  input  wire [`D_N-1:0] w_a, w_b;
  output wire [`D_N-1:0] w_s;
  wire [`D_N:0] w_cin;
  assign w_cin[0] = 0;
  generate genvar g;
    for (g = 0; g < `D_N; g = g + 1) begin : Gen
      m_FA m_FA0(w_a[g], w_b[g], w_cin[g], w_s[g], w_cin[g+1]);
    end
  endgenerate
endmodule

module m_FA (w_a, w_b, w_cin, w_s, w_cout);
   /* Please describe here by yourself */
endmodule

module m_HA (w_a, w_b, w_s, w_c);
  input  wire w_a, w_b;
  output wire w_s, w_c;
  assign w_c = w_a & w_b;
  assign w_s = w_a ^ w_b;
endmodule
```

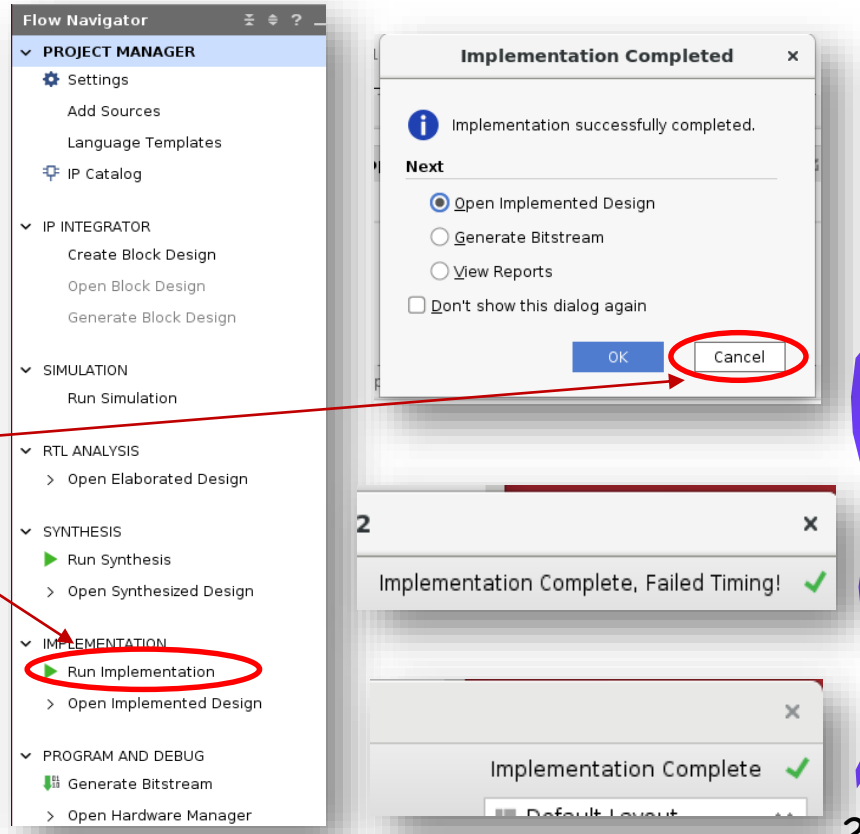
加算器のクリティカルパスの遅延を計測する

- 演習で，code078.v を修正して，100MHzの動作周波数の制約を満たす n-bit Adder の最大の n を求めること．ただし，n は5の倍数とする．
  - code078.v を用いて合成する(Run Implementation) ． Bitstreamは生成する必要はない．
  - 1行目の D_N の値を変化させて合成．Failed Timing! と出力された時は制約を満たしていない．
  - 1行目の D_N の値を小さくして合成．Implementation Complete が出力された時は満たしている．

code078.v

```
`define D_N 32

module m_main (w_clk, w_a, w_b, w_dout);
  input  wire w_clk, w_a, w_b;
  output wire w_dout;
  reg [`D_N-1:0] r_a=0, r_b=0, r_s=0;
  wire [`D_N-1:0] w_s;
  assign w_dout = ^r_s;
  always@(posedge w_clk) begin
    r_a <= {w_a, r_a[`D_N-1:1]};
    r_b <= {w_b, r_b[`D_N-1:1]};
    r_s <= w_s;
  end
  m_ADDER m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER (w_a, w_b, w_s);
  input  wire [`D_N-1:0] w_a, w_b;
  output wire [`D_N-1:0] w_s;
  wire [`D_N:0] w_cin;
  assign w_cin[0] = 0;
  generate genvar g;
    for (g = 0; g < `D_N; g = g + 1) begin : Gen
      m_FA m_FA0(w_a[g], w_b[g], w_cin[g], w_s[g], w_cin[g+1]);
    end
  endgenerate
endmodule

以降は省略
```

click this



Worst Negative Slack (WNS) & Critical Path

- From Vivado menu, select Open Implemented Design
- Design Timing Summary ウィンドウが表示される.
- WNS が正の値であれば，生成された回路は制約を満たしている．また，回路にはその値だけの余裕(slack)があることを示す.
  - 左図の D_N = 32 の例では，クロック周波数が 100MHz で 10 ns の制約に対して WNS は 1.796 ns となっており，これだけの余裕があることを示す．つまり制約を満たしている．この回路のクリティカルパスの遅延は 10 – 1.796 = 8.204 ns となる.
  - 右図の D_N = 80 例では，WNS は -3.527 であり，制約を満たしていない．この回路のクリティカルパスの遅延は 10 + 3.527 = 13.527 ns となる.

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	1.796 ns	Worst Hold Slack (WHS):	0.166 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	94	Total Number of Endpoints:	94

All user specified timing constraints are met.

D_N 32 とした時のRipple Carry Adderの合成結果

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	-3.527 ns	Worst Hold Slack (WHS):	0.136 ns
Total Negative Slack (TNS):	-60.012 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	29	Number of Failing Endpoints:	0
Total Number of Endpoints:	238	Total Number of Endpoints:	238

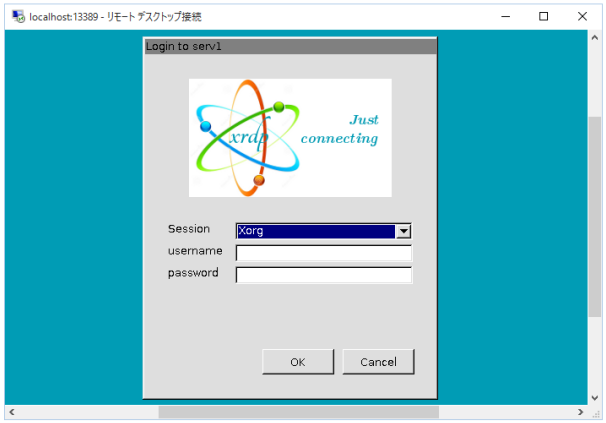
Timing constraints are not met.

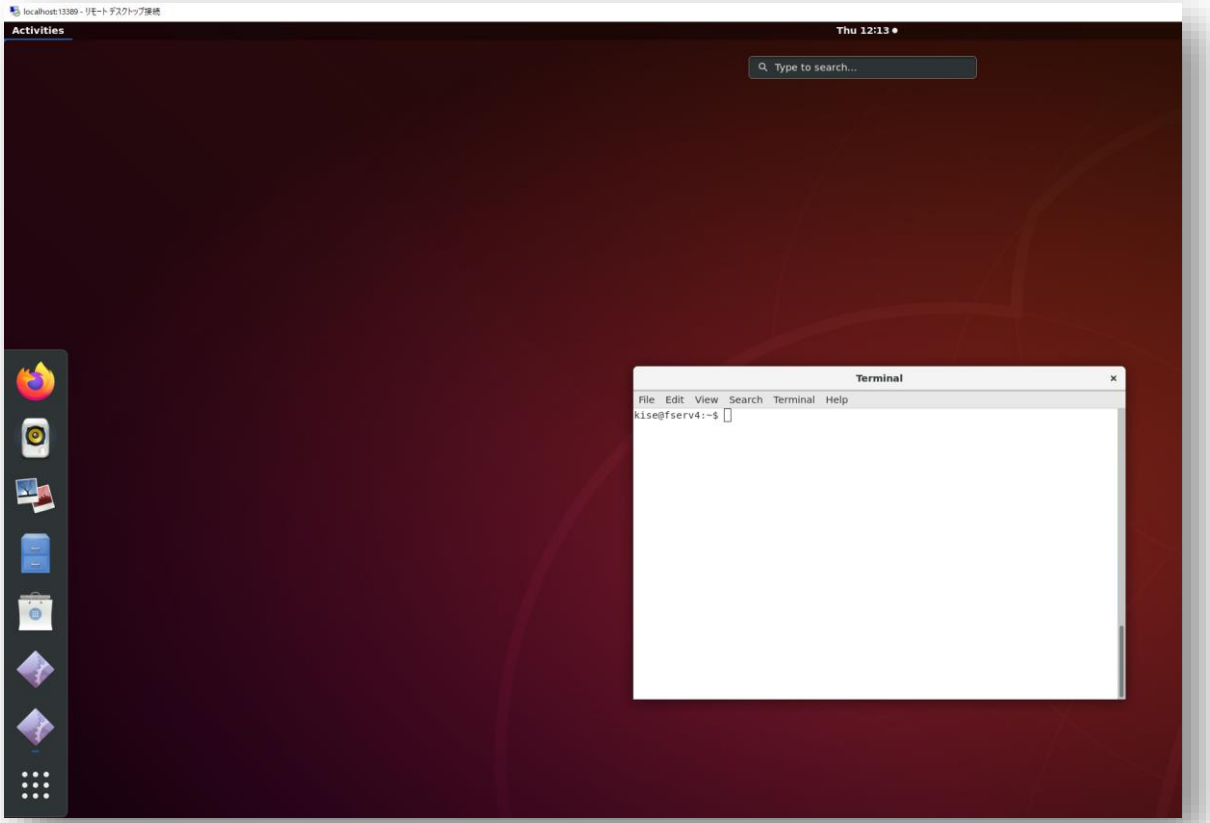
D_N 80とした時のRipple Carry Adderの合成結果

Vivado 2022.2 を利用

ACRiルームのデモンストレーション

- Vivado での VIO の使い方.
- Vivado での HW manager, オープンしたデザイン, ソースの切り替え方法.
- WNS





References

- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog