Course number: CSC.T341

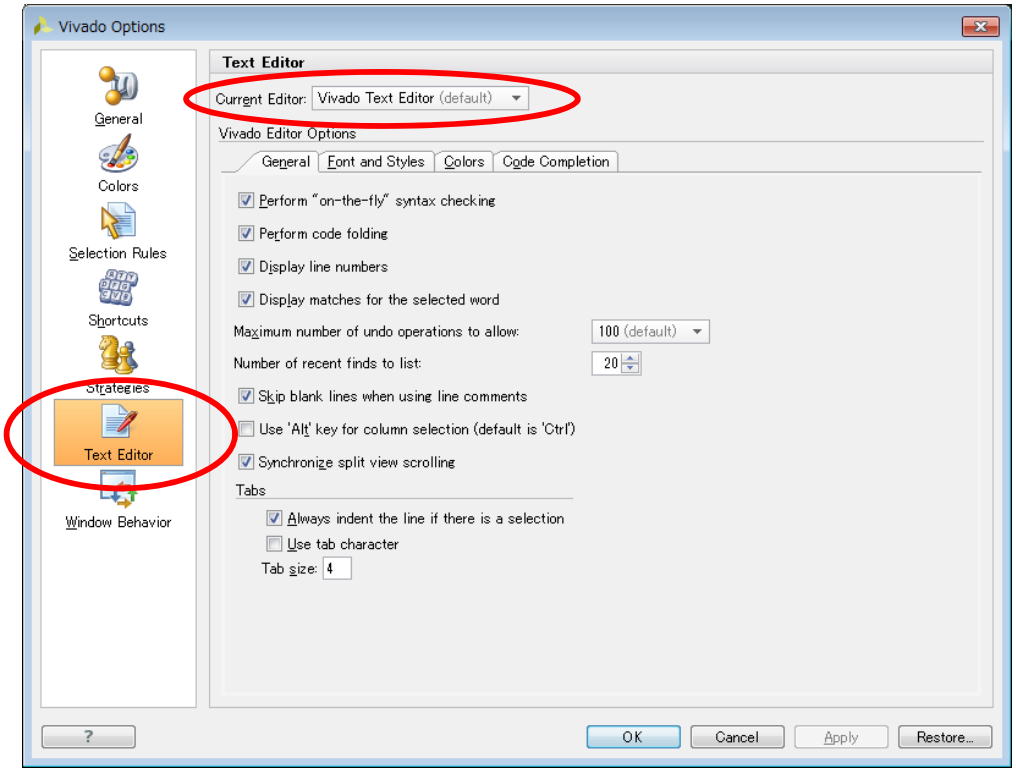
コンピュータ論理設計
Computer Logic Design

4. ハードウェア記述言語：よく使われる回路
Hardware Description Language: Typical Circuits

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/
W621 講義室　月 10:45-12:15, 木 9:00-12:15

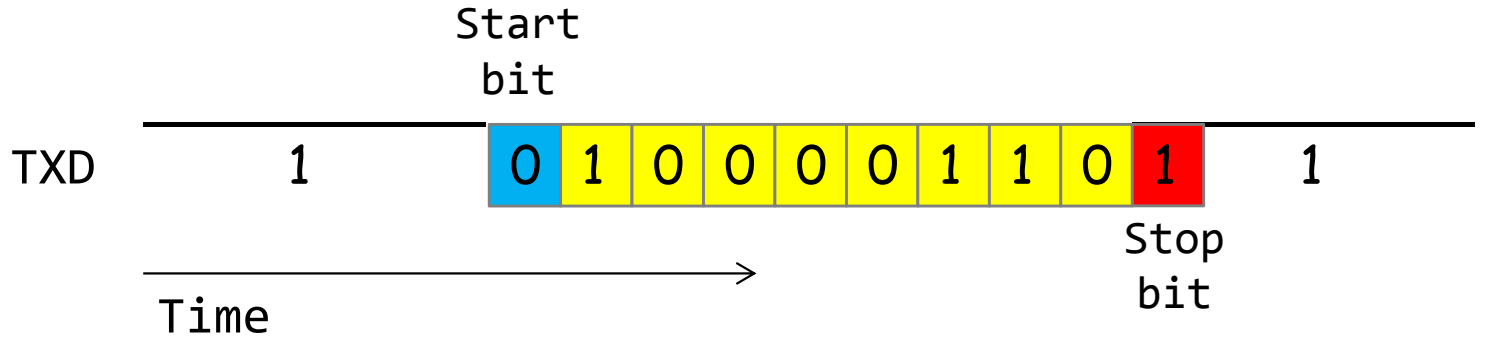
Vivado Options: Text Editor

- You can use your favorite text editor instead of Vivado text editor (default).
  - Select Tools, Options menu.
  - Select Text Editor, then click Current Editor box.
  - Select your favorite editor.



UART (Universal Asynchronous Receiver/Transmitter)

- 調歩同期方式によるシリアル信号をパラレル信号に変換したり，その逆方向の変換をおこなう集積回路をUARTと呼ぶ．8ビット（1バイト）単位でデータを送信・受信する．
- UARTを用いることで，FPGAとコンピュータの間でのお手軽なデータ通信が可能．
- 例えば，'a' という文字を送信する場合，'a' は 8'h61, 8'b01100001（次スライドのASCII Tableを参照）なので，下図のタイミングで送信線TXDを制御する．
  - データが送信されるまで送信線TXD を1とする．
  - まず，青色で示した0（これをスタートビットと呼ぶ）を送信することで，データ送信の開始を明示．
  - 次に，黄色で示した様に送信したいデータ 8'b01100001 の最下位ビットから順番に送信する．
  - 最後に，赤色で示した1（これをストップビットと呼ぶ）を送信する．
- 1ビットを送受信するための時間間隔は送信側と受信側で同じレートを用いる．これをボー・レート (baud) と呼ぶ．
  - 9,600, 14,400 baud 等が一般に用いられる．この演習では主に 1,000,000 baudを用いる．

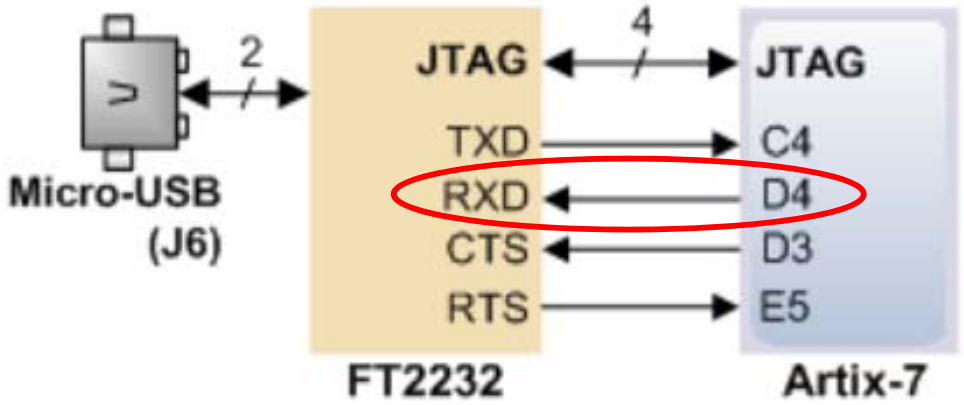


Inside main05.xdc

- main.xdcをmain05.xdcの内容となるように入力する.
- ピン w_txd は, UARTのFPGAからのデータ送信のために用いる.
  - コンピュータからはUARTのデータ受信 (RXD) となる.

main05.xdc

```
set_property -dict { PACKAGE_PIN E3   IOSTANDARD LVCMOS33} [get_ports { w_clk }];
create_clock -add -name sys_clk -period 10.00 -waveform {0 5} [get_ports {w_clk}];

set_property -dict { PACKAGE_PIN D4   IOSTANDARD LVCMOS33} [get_ports { w_txd }];
```



Inside code062.v

- シリアル通信によるFPGAからの送信回路の例を示す.
- シリアル通信のボー・レートを 25Mbaudとする(速すぎてあまり実用的ではない).
  - すなわち, クロックが100MHzなので, 4サイクルで1ビットを送信する.
- main.vをcode062.vの内容となるように入力し, 3000nsec までシミュレーションする.

code062.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  reg r_we=0;
  wire w_txd;
  initial begin #130 r_we = 1; #100 r_we = 0; end
  m_UartTx m_UartTx0(r_clk, 8'h61, r_we, w_txd);
  always@(posedge r_clk) $write("%4d %b %b %b¥n", $time, r_we, w_txd, m_UartTx0.r_data);
endmodule

module m_UartTx (w_clk, w_data, w_we, r_txd);
  input wire       w_clk, w_we;
  input wire [7:0] w_data;;
  output reg       r_txd;
  initial r_txd = 1;
  reg [8:0] r_data =~0;
  reg [7:0] r_wait = 0;
  always@(posedge w_clk) begin
    if (w_we) begin
      r_data <= {w_data, 1'b0}; // add start bit
      r_wait <= 0;
    end else if (r_wait >= (4-1)) begin
      r_txd  <= r_data[0];
      r_data <= {1'b1, r_data[8:1]};
      r_wait <= 0;
    end else begin
      r_wait <= r_wait + 1;
    end
  end
endmodule
```

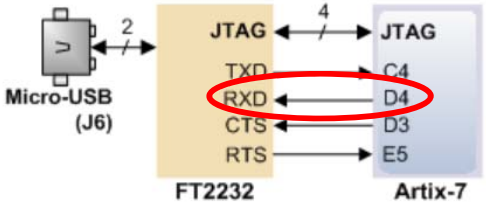
```
  50 0 1 111111111
 150 1 1 111111111
 250 0 1 011000010
 350 0 1 011000010
 450 0 1 011000010
 550 0 1 011000010
 650 0 0 101100001
 750 0 0 101100001
 850 0 0 101100001
 950 0 0 101100001
1050 0 1 110110000
1150 0 1 110110000
1250 0 1 110110000
1350 0 1 110110000
1450 0 0 111011000
1550 0 0 111011000
1650 0 0 111011000
1750 0 0 111011000
1850 0 0 111101100
1950 0 0 111101100
2050 0 0 111101100
2150 0 0 111101100
2250 0 0 111110110
2350 0 0 111110110
2450 0 0 111110110
2550 0 0 111110110
2650 0 0 111111011
2750 0 0 111111011
2850 0 0 111111011
2950 0 0 111111011
```

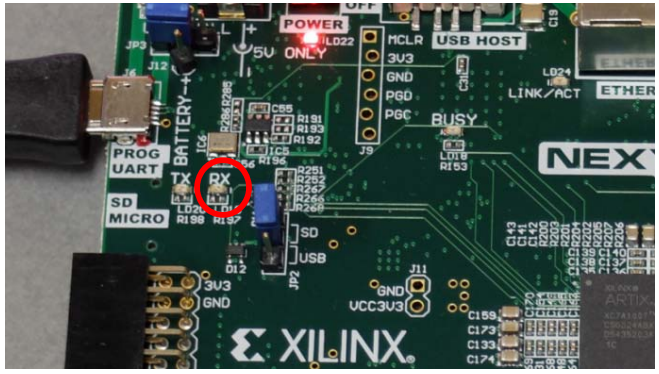
Inside code063.v

- シリアル通信によるFPGAからの送信回路の例を示す.
- シリアル通信のボー・レートを 1,000,000 baud (1Mbaud) とする.
  - すなわち，クロックが100MHzなので，100サイクルで1ビットを送信する.
- main.vをcode063.vの内容となるように入力し，合成，コンフィギュレーションする.
  - FPGAボードのRXと書かれたLEDが一定間隔で点滅する.

code063.v

```
module m_main(w_clk, w_txd);
  input  wire w_clk;
  output wire w_txd;
  reg r_we;
  reg [31:0] r_cnt = 1;
  always@(posedge w_clk) r_cnt <= (r_cnt>50000000) ? 0 : r_cnt + 1;
  always@(posedge w_clk) r_we <= (r_cnt==0);
  m_UartTx m_UartTx0(w_clk, 8'h61, r_we, w_txd);
endmodule
```



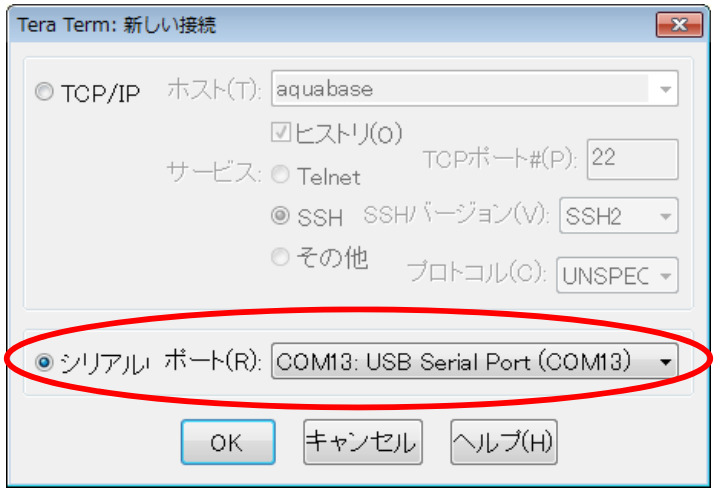


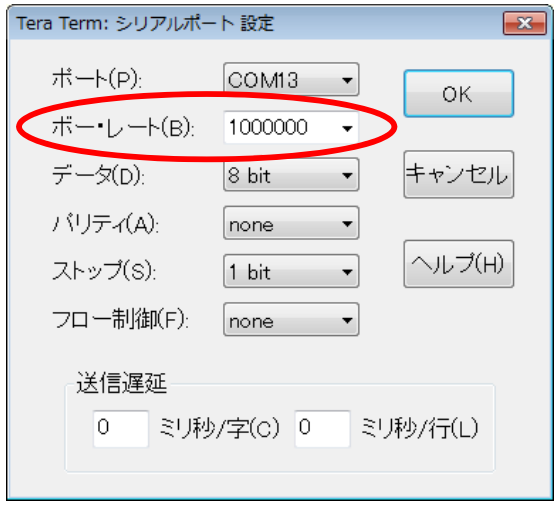
code063.v

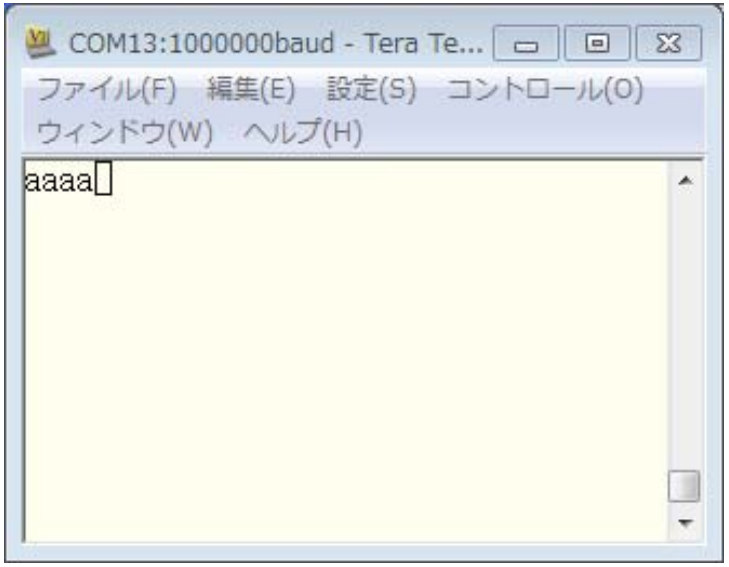
```
module m_UartTx (w_clk, w_data, w_we, r_txd);
  input wire       w_clk, w_we;
  input wire [7:0] w_data;;
  output reg       r_txd;
  initial r_txd = 1;
  reg [8:0] r_data =~0;
  reg [7:0] r_wait = 0;
  always@(posedge w_clk) begin
    if (w_we) begin
      r_data <= {w_data, 1'b0}; // add start bit
      r_wait <= 0;
    end else if (r_wait >= (100-1)) begin
      r_txd  <= r_data[0];
      r_data <= {1'b1, r_data[8:1]};
      r_wait <= 0;
    end else begin
      r_wait <= r_wait + 1;
    end
  end
endmodule
```

Inside code063.v

- main.vをcode063.vの内容となるように入力し，合成，コンフィギュレーションする．
- Tera Term を起動する．
  - シリアルポートを選択．複数のポートが選択できる場合には最も大きい番号を選択．
  - 設定，シリアルポートを選択し，ボー・レートに 1000000 を入力
  - Tera Term に一定の間隔で a の文字が表示される．





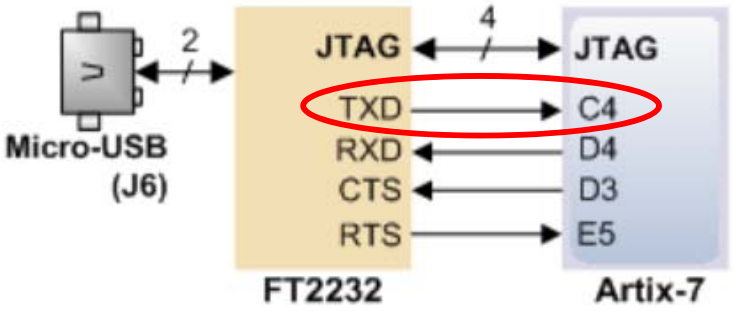


Inside main06.xdc

- main.xdcをmain06.xdcの内容となるように入力する.
- ピン w_rxd は, FPGAのデータ受信. コンピュータからはデータ送信 (TXD) となる.

main06.xdc

```
set_property -dict { PACKAGE_PIN E3  IOSTANDARD LVCMOS33} [get_ports { w_clk }];
create_clock -add -name sys_clk -period 10.00 -waveform {0 5} [get_ports {w_clk}];

set_property -dict { PACKAGE_PIN C4  IOSTANDARD LVCMOS33} [get_ports { w_rxd }];

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports { w_led[0] }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports { w_led[1] }];
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports { w_led[2] }];
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports { w_led[3] }];
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports { w_led[4] }];
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports { w_led[5] }];
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports { w_led[6] }];
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports { w_led[7] }];
```



Inside code064.v

- シリアル通信を用いたFPGA側の受信回路の例を示す.
- シリアル通信のボーレートを25M baudとする. クロックが100MHzなので 4サイクルで1ビットを受信.
- main.vをcode064.vの内容となるように入力し, 5000nsec をシミュレーションする.
  - code063.v の m_UartTx も必要.

code064.v

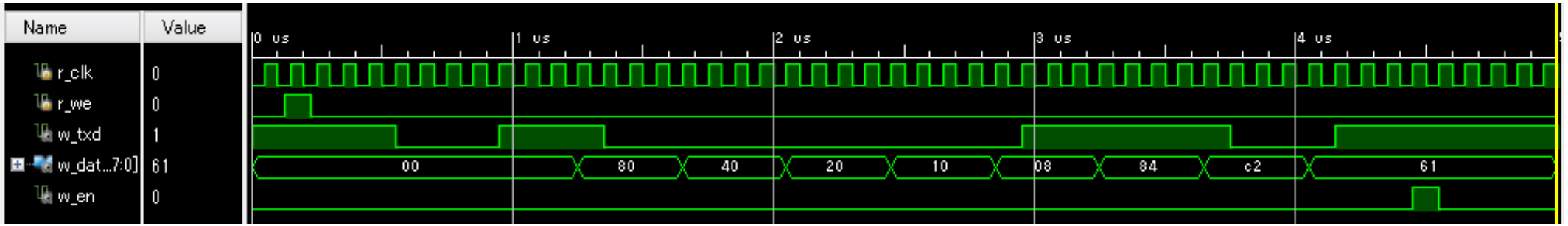
```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  reg r_we=0;
  wire w_txd;
  initial begin #130 r_we = 1; #100 r_we = 0; end
  m_UartTx m_UartTx0(r_clk, 8'h61, r_we, w_txd);
  wire [7:0] w_data;
  wire       w_en;
  m_UartRx m_UartRx0(r_clk, w_txd, w_data, w_en);

  always@(posedge r_clk) begin
    $write("%4d %b %b", $time, r_we, w_txd);
    $write(" %b -> ", m_UartTx0.r_data);
    $write("%b %b¥n", w_data, w_en);
  end
endmodule
```

code064.v

```
module m_UartRx(w_clk, w_rxd, r_data, r_en);
  input  wire      w_clk, w_rxd;
  output reg [7:0] r_data;
  output reg       r_en;
  initial begin r_data = 0; r_en = 0; end
  reg [5:0] r_detect = 0;
  always @(posedge w_clk) r_detect <= (w_rxd) ? 0 : r_detect + 1;
  reg       r_recv = 0;
  reg [6:0] r_wait = 0;
  reg [3:0] r_cnt  = 0;
  always@(posedge w_clk) begin
    if (r_recv==0) begin
        if(r_detect>1) begin r_recv <= 1; r_data<= 0; end
        r_cnt  <= 8;
        r_wait <= 0;
        r_en   <= 0;
    end else begin
      if (r_wait >= (4-1)) begin
        r_cnt  <= r_cnt - 1;
        r_data <= (r_cnt==0) ? r_data : {w_rxd, r_data[7:1]};
        r_wait <= 0;
        if (r_cnt == 0) begin r_en <= 1; r_recv <= 0; end
      end else begin
        r_wait <= r_wait + 1;
      end
    end
  end
endmodule
```

Inside code064.v

Waveform window




```
  50 0 1 111111111 -> 00000000 0
 150 1 1 111111111 -> 00000000 0
 250 0 1 011000010 -> 00000000 0
 350 0 1 011000010 -> 00000000 0
 450 0 1 011000010 -> 00000000 0
 550 0 1 011000010 -> 00000000 0
 650 0 0 101100001 -> 00000000 0
 750 0 0 101100001 -> 00000000 0
 850 0 0 101100001 -> 00000000 0
 950 0 0 101100001 -> 00000000 0
1050 0 1 110110000 -> 00000000 0
1150 0 1 110110000 -> 00000000 0
1250 0 1 110110000 -> 00000000 0
1350 0 1 110110000 -> 10000000 0
1450 0 0 111011000 -> 10000000 0
1550 0 0 111011000 -> 10000000 0
1650 0 0 111011000 -> 10000000 0
1750 0 0 111011000 -> 01000000 0
1850 0 0 111101100 -> 01000000 0
1950 0 0 111101100 -> 01000000 0
2050 0 0 111101100 -> 01000000 0
2150 0 0 111101100 -> 00100000 0
2250 0 0 111110110 -> 00100000 0
2350 0 0 111110110 -> 00100000 0
2450 0 0 111110110 -> 00100000 0
```

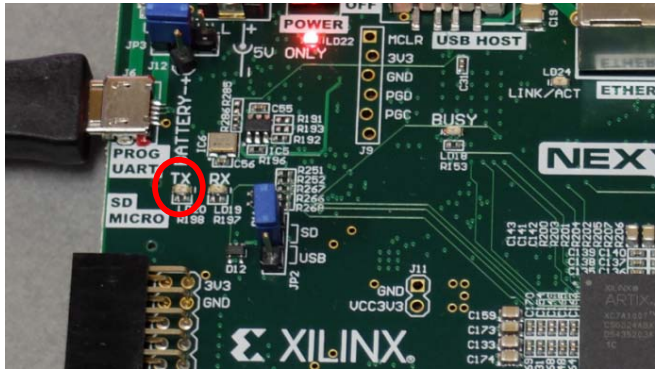
```
2550 0 0 111110110 -> 00010000 0
2650 0 0 111111011 -> 00010000 0
2750 0 0 111111011 -> 00010000 0
2850 0 0 111111011 -> 00010000 0
2950 0 0 111111011 -> 00001000 0
3050 0 1 111111101 -> 00001000 0
3150 0 1 111111101 -> 00001000 0
3250 0 1 111111101 -> 00001000 0
3350 0 1 111111101 -> 10000100 0
3450 0 1 111111110 -> 10000100 0
3550 0 1 111111110 -> 10000100 0
3650 0 1 111111110 -> 10000100 0
3750 0 1 111111110 -> 11000010 0
3850 0 0 111111111 -> 11000010 0
3950 0 0 111111111 -> 11000010 0
4050 0 0 111111111 -> 11000010 0
4150 0 0 111111111 -> 01100001 0
4250 0 1 111111111 -> 01100001 0
4350 0 1 111111111 -> 01100001 0
4450 0 1 111111111 -> 01100001 0
4550 0 1 111111111 -> 01100001 1
4650 0 1 111111111 -> 01100001 0
4750 0 1 111111111 -> 01100001 0
4850 0 1 111111111 -> 01100001 0
4950 0 1 111111111 -> 01100001 0
```

Inside code065.v

- シリアル通信を用いたFPGA側の受信回路の例を示す.
- シリアル通信のボーレートを 1,000,000 baud (1Mbaud) とする.
  - すなわち, クロックが100MHzなので, 100サイクルで1ビットを受信する.
- main.vをcode065.vの内容となるように入力し, 合成, コンフィギュレーションする.
- Tera Term からの入力で, LEDの点滅を確認.

code065.v

```
module m_main(w_clk, w_rxd, w_led);
  input  wire w_clk;
  input  wire w_rxd;
  output wire [7:0] w_led;

  wire w_en;
  m_UartRx m_UartRx0 (w_clk, w_rxd, w_led, w_en);
endmodule
```



code065.v

```
module m_UartRx(w_clk, w_rxd, r_data, r_en);
  input  wire      w_clk, w_rxd;
  output reg [7:0] r_data;
  output reg       r_en;
  initial begin r_data = 0; r_en = 0; end
  reg [5:0] r_detect = 0;
  always @(posedge w_clk) r_detect <= (w_rxd) ? 0 : r_detect + 1;
  reg       r_recv = 0;
  reg [6:0] r_wait = 0;
  reg [3:0] r_cnt  = 0;
  always@(posedge w_clk) begin
    if (r_recv==0) begin
        if(r_detect>10) begin r_recv <= 1; r_data<= 0; end
        r_cnt  <= 8;
        r_wait <= 0;
        r_en   <= 0;
    end else begin
      if (r_wait >= (100-1)) begin
        r_cnt  <= r_cnt - 1;
        r_data <= (r_cnt==0) ? r_data : {w_rxd, r_data[7:1]};
        r_wait <= 0;
        if (r_cnt == 0) begin r_en <= 1; r_recv <= 0; end
      end else begin
        r_wait <= r_wait + 1;
      end
    end
  end
endmodule
```

Inside code071.v

- ある回路の記述例を示す.
  - 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を使うとエラーと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．全てのソースコードの先頭に `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("%d %d -> %b %b¥n", 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
```

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

code072.v

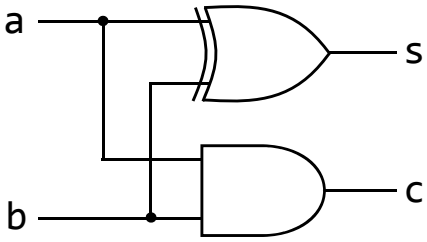
```
`default_nettype none

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("%d %d -> %b %b¥n", 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
```

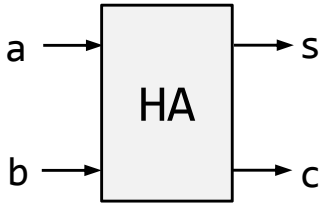
Inside code073.v : Half Adder

- 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, 出力 c (carry out), s (sum) とするtruth table(真理値表)を table073 に示す.

table073

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





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
```

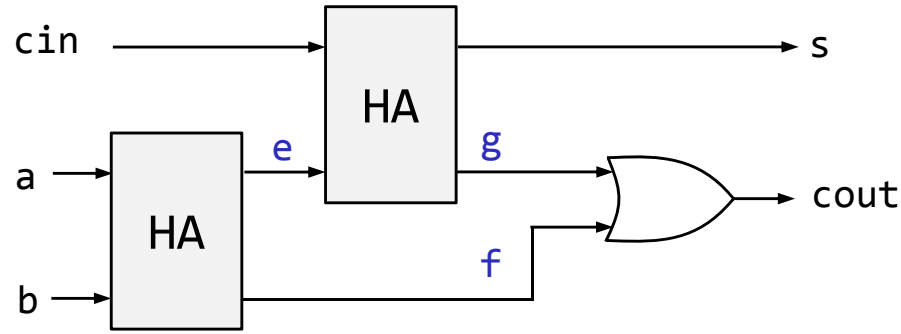
```
11: 0 0 -> 0 0
21: 0 1 -> 0 1
31: 1 0 -> 0 1
41: 1 1 -> 1 0
```

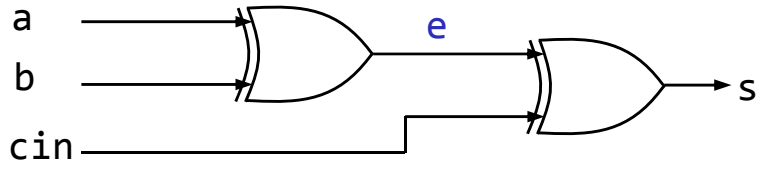
Inside code074.v : Full Adder

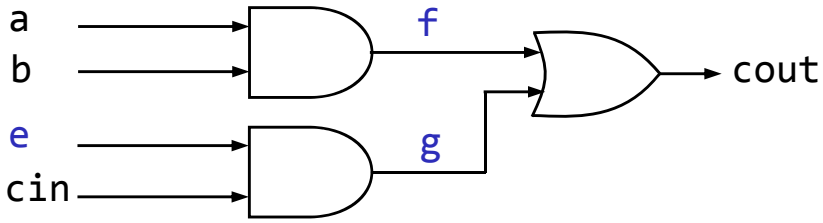
- Full Adder, FA (全加算器)の回路とその記述の例を示す.
  - 1ビットの入力 a, b, cin の加算をおこなう回路.
  - 入力 a, b, cin (carry in), 出力 cout (carry out), s (sum) とするtruth tableを table074 に示す.

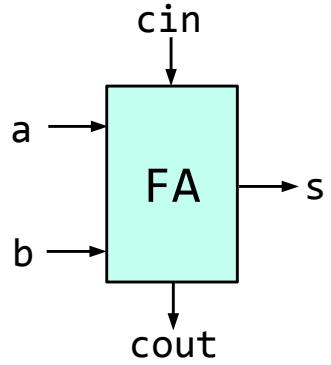
table074

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







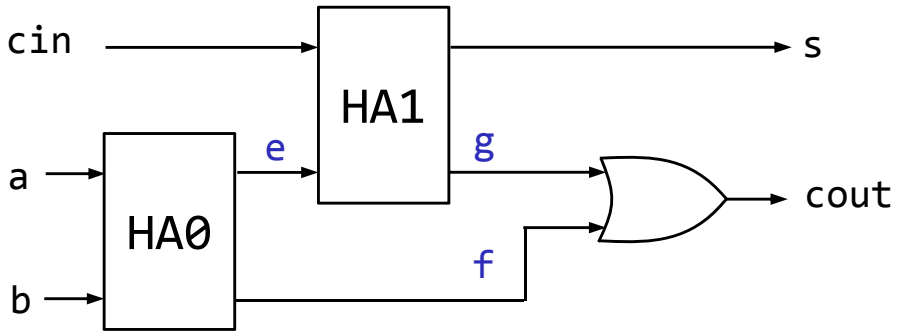


Inside code074.v : Full Adder

- Full Adder, FA (全加算器)の回路とその記述の例を示す.
- main.vをcode074.vの内容となるように入力して, シミュレーションする.

table074

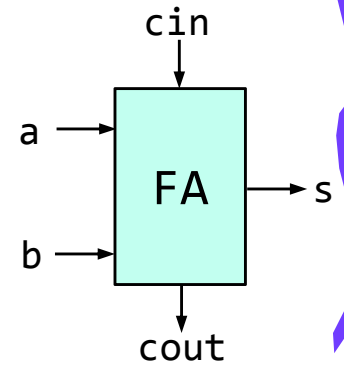
a	b	cin	Cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



code074.v

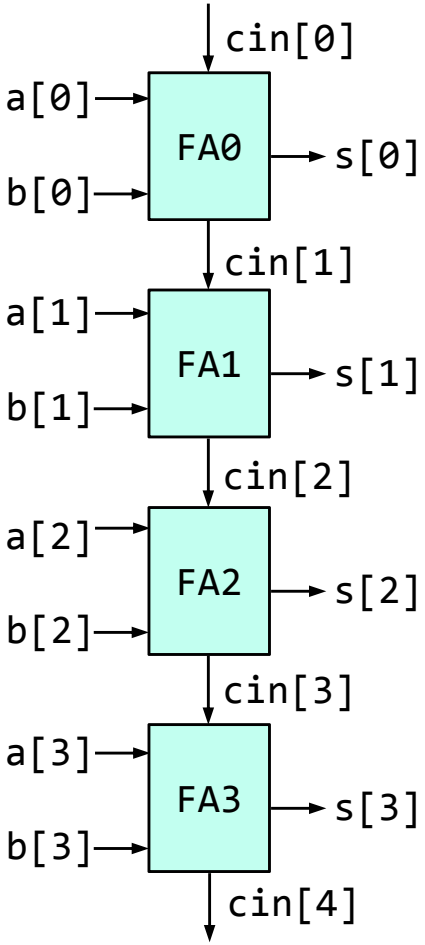
```
module m_top ();
  reg  r_a, r_b, r_cin;
  wire w_s, w_cout;
  initial begin
    #10 r_a <= 0; r_b <= 0;  r_cin <= 0;
    #10 r_a <= 0; r_b <= 0;  r_cin <= 1;
    #10 r_a <= 0; r_b <= 1;  r_cin <= 0;
    #10 r_a <= 0; r_b <= 1;  r_cin <= 1;
    #10 r_a <= 1; r_b <= 0;  r_cin <= 0;
    #10 r_a <= 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);
  input  wire w_a, w_b, w_cin;
  output wire w_s, w_cout;
  wire w_e, w_f, w_g;
  m_HA m_HA0 (w_a, w_b, w_e, w_f);
  m_HA m_HA1 (w_cin, w_e, w_s, w_g);
  assign w_cout = w_g | w_f;
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
```



Inside code075.v : 4-bit Adder

- 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.
- main.vをcode075.vの内容となるように入力して, シミュレーションする. m_FA, m_HA の記述も必要.



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 m_FA0 (w_a[0], w_b[0], w_cin[0], w_s[0], w_cin[1]);
  m_FA m_FA1 (w_a[1], w_b[1], w_cin[1], w_s[1], w_cin[2]);
  m_FA m_FA2 (w_a[2], w_b[2], w_cin[2], w_s[2], w_cin[3]);
  m_FA m_FA3 (w_a[3], w_b[3], w_cin[3], w_s[3], w_cin[4]);
endmodule
```

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

Inside code076.v : 32-bit Adder

- 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].mFA0, Gen[1].mFA0, Gen[2].mFA0, … となる.
- main.vをcode076.vの内容となるように入力して，シミュレーションする. m_FA, m_HA の記述も必要.

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
```

Inside code077.v : n-bit Adder

- 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.
  - main.vをcode077.vの内容となるように編集して, シミュレーションする. m_FA, m_HA の記述も必要.

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
```

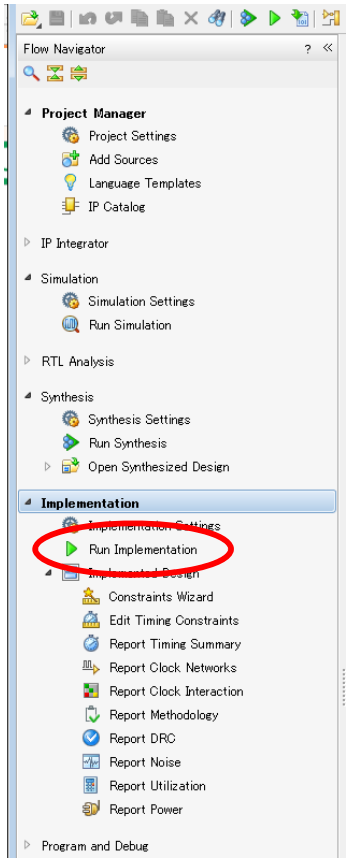
```
   1   28 ->   29
   0    0 ->    0
```

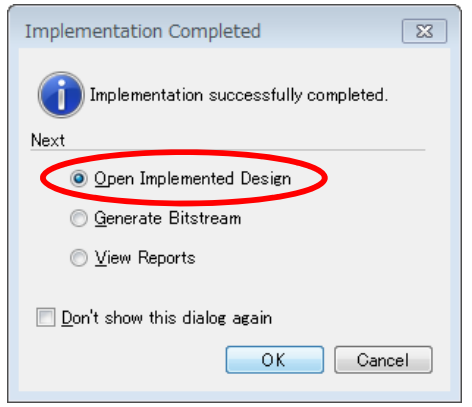
Inside code078.v : n-bit Adder

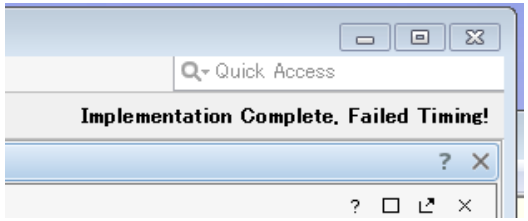
- 100MHzの動作周波数で制約を満たす n-bit Adder の最大の n を求める.
  - ヒント: D_N 48, 56 としてそれぞれの合成結果を確認してみる.
  - main.vがcode078.vとなるように入力し, 合成する(Run Implementation) . Bitstreamは生成しない.
  - D_Nの値を変化させて合成. Failed Timing! と出力された時は制約を満たしていない. 次スライド参考.
  - D_Nの値を小さくして合成. Implementation Complete のみが出力された時は制約を満たしている.

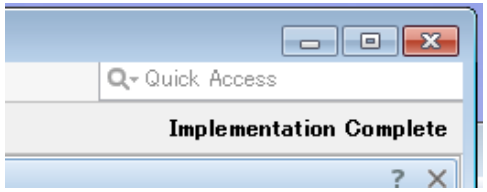
code078.v

```
`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
```









Worst Negative Slack (WNS) & Critical Path

- From Vivado menu, select Run Implementation, 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．
  - 左図の例では，クロック周波数が 100MHz で 10 ns の制約に対して WNG は 0.982 ns となっており 0.982 ns 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は 10 – 0.982 = 9.018 ns となる．
  - 右図の例では，WNG は -2.262 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は 10 + 2.262 = 12.262 ns となる．

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	0.982 ns	Worst Hold Slack (WHS):	0.123 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 とした時の加算器の合成結果

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	-2.262 ns	Worst Hold Sl	
Total Negative Slack (TNS):	-20.196 ns	Total Hold Sla	
Number of Failing Endpoints:	18	Number of Fa	
Total Number of Endpoints:	190	Total Number	

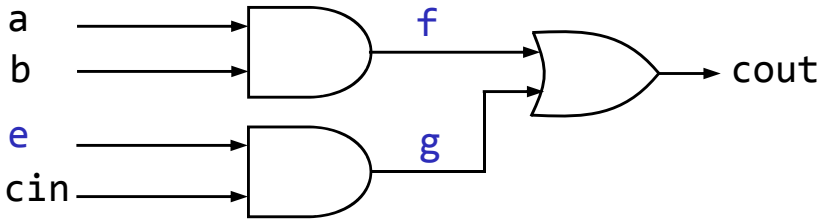
Timing constraints are not met.

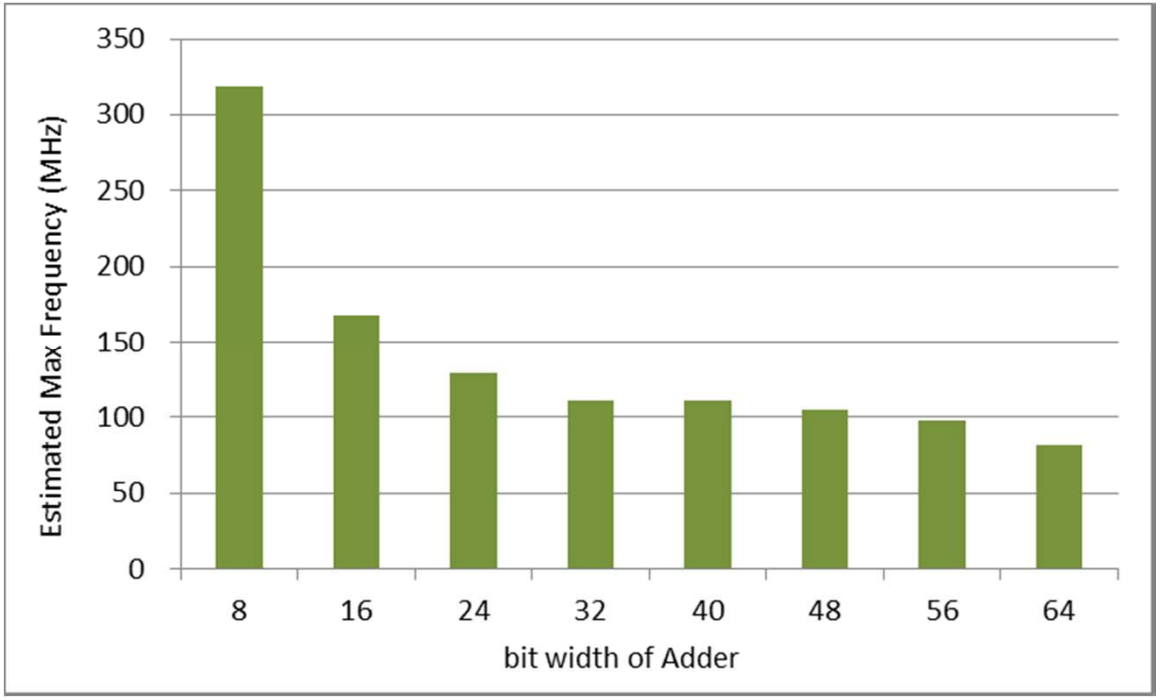
D_N 64 とした時の加算器の合成結果

Inside code078.v : n-bit Ripple Carry Adder

- 100MHzの動作周波数で制約を満たす n-bit Adder の最大の n を求める.
  - ヒント: D_N 48, 56 としてそれぞれの合成結果を確認してみる.
  - nを変化させてWNSを求めた結果を示す. Critical Path, Max Freq. は見積もり.
  - The carry out signal takes two gate delays per bit.

n	WNS	Critical Path (ns)	Max Freq. (MHz)
8	6.858	3.142	318.27
16	4.056	5.944	168.24
24	2.266	7.734	129.30
32	0.982	9.018	110.89
40	0.997	9.003	111.07
48	0.479	9.521	105.03
56	−0.165	10.165	98.38
64	−2.262	12.262	81.55





Inside code079.v : n-bit Adder

- パラメータを用いた n-bit Adder の記述例を示す.
  - code077.v と code079.v は同じ回路を記述している. code079.vでは, モジュール名と信号との間に parameter によりパラメータと値を定義する. この例では, パラメータ P_N を値8として定義している.
  - モジュールをインスタンス化する時に, 定義したパラメータの値を上書きできる. code079.v では, #(5) により, モジュール m_ADDER のパラメータ P_N の値を5で上書きしている.

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
```

code079.v

```
module m_top ();
  reg  [4:0] r_a, r_b;
  wire [4:0] 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 #(5) m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER #(parameter P_N=8) (w_a, w_b, w_s);
  input  wire [P_N-1:0] w_a, w_b;
  output wire [P_N-1:0] w_s;
  wire [P_N:0] w_cin;
  assign w_cin[0] = 0;
  generate genvar g;
    for (g = 0; g < P_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
```

Inside code080.v : n-bit Adder

- パラメータを用いた n-bit Adder の別の記述例を示す.
  - code079.v と code080.v は同じ回路を記述している.
  - code079.v で記述したRipple Carry Adder（順次桁上げ加算器）よりも，code080.v の記述の方がとても高速な回路を得ることができる.
  - 桁上げ先見加算器 (Carry Lookahead Adder) について調べてみる.

code079.v

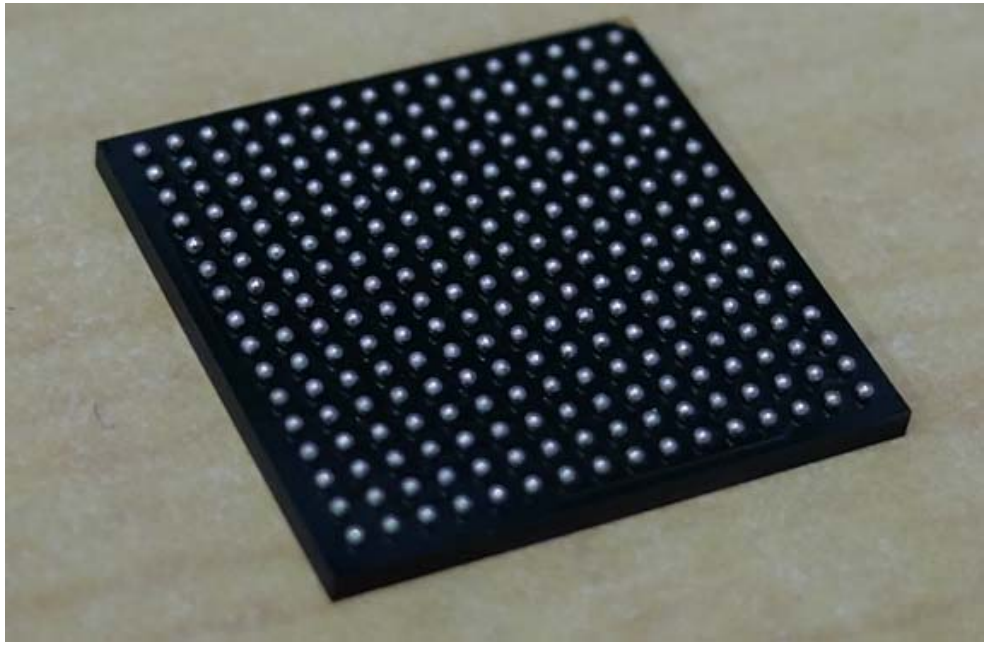
```
module m_top ();
  reg  [4:0] r_a, r_b;
  wire [4:0] w_s;
  initial begin
    #10 r_a <=  321; r_b <= 4444;
    #10 r_a <= 1024; r_b <= 2048;
  end
  always@(*) #1 $write("%4d %4d -> %4d¥n", r_a, r_b, w_s);
  m_ADDER #(5) m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER #(parameter P_N=8) (w_a, w_b, w_s);
  input  wire [P_N-1:0] w_a, w_b;
  output wire [P_N-1:0] w_s;
  wire [P_N:0] w_cin;
  assign w_cin[0] = 0;
  generate genvar g;
    for (g = 0; g < P_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
```

code080.v

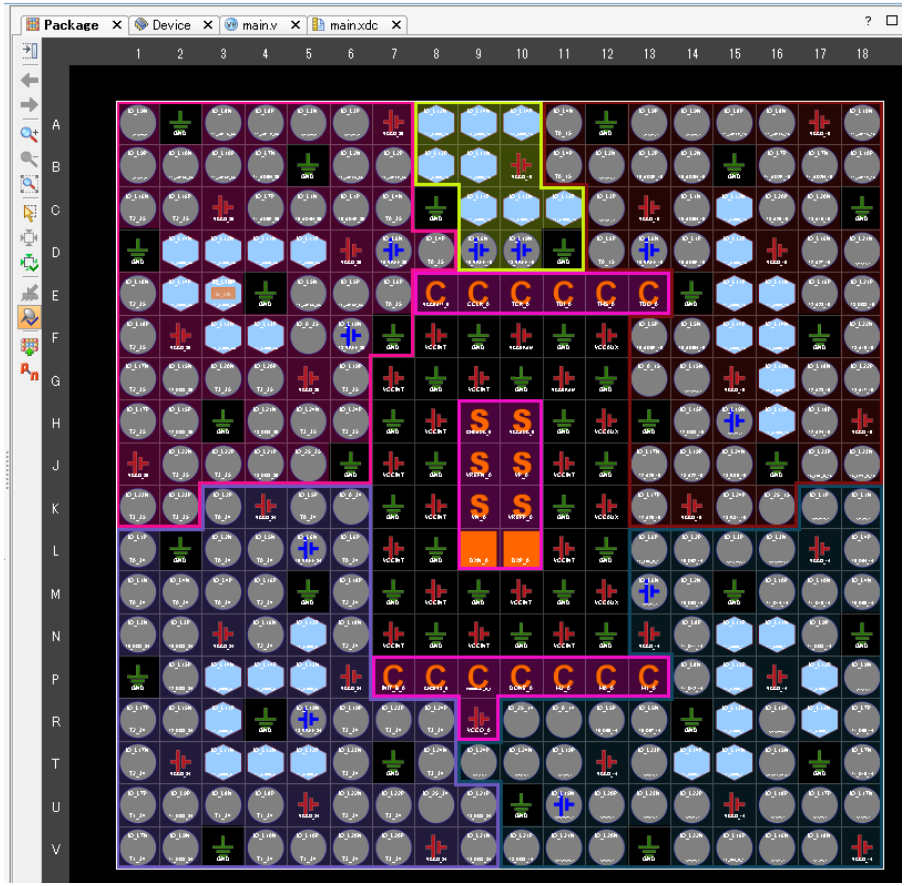
```
module m_top ();
  reg  [4:0] r_a, r_b;
  wire [4:0] w_s;
  initial begin
    #10 r_a <=  321; r_b <= 4444;
    #10 r_a <= 1024; r_b <= 2048;
  end
  always@(*) #1 $write("%4d %4d -> %4d¥n", r_a, r_b, w_s);
  m_ADDER #(5) m_ADDER0 (r_a, r_b, w_s);
endmodule

module m_ADDER #(parameter P_N=8) (w_a, w_b, w_s);
  input  wire [P_N-1:0] w_a, w_b;
  output wire [P_N-1:0] w_s;
  assign w_s = w_a + w_b;
endmodule
```

FPGA (Field Programmable Gate Array)

- Artix-7 FPGA (xc7a100tcsg324-1)
- Vivado, select Layout Menu, then I/O planning
  - 16 x 16 = 324 pins, max user pins of 300



FPGA chip photo




```
set_property -dict { PACKAGE_PIN E3   IOSTANDARD LVCMOS33} [get_ports { w_clk }];
set_property -dict { PACKAGE_PIN D4   IOSTANDARD LVCMOS33} [get_ports { w_txd }];
```

FPGA (Field Programmable Gate Array)

- Artix-7 FPGA (xc7a100tcsg324-1)
  - CLB(Configurable Logic Block), Slice, LUT(lookup Table), FF

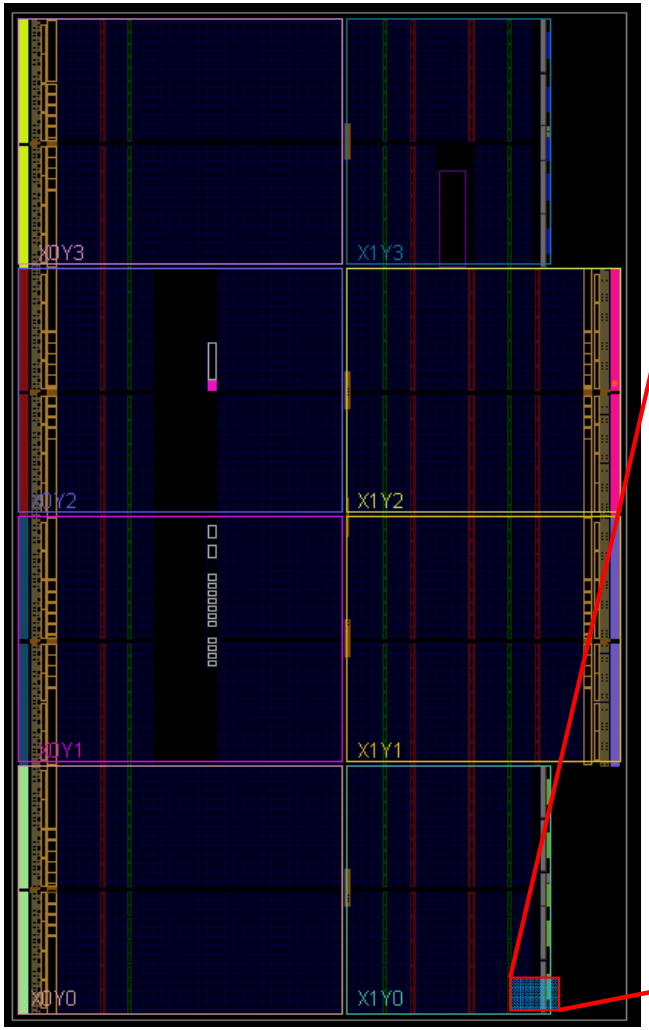
表 4: Artix-7 FPGA の機能一覧

デバイス	ロジックセル	コンフィギャラブルロジックブロック (CLB)		DSP48E1 スライス (2)	ブロック RAM ブロック (3)			CMT (4)	PCIe (5)	GTP	XADC ブロック	総 I/O バンク (6)	最大ユーザー I/O (7)
		スライス数 (1)	最大分散 RAM (Kb)		18Kb	36Kb	最大 (Kb)						
XC7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XC7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XC7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	250
XC7A75T	75,520	11,800	892	180	210	105	3,780	6	1	8	1	6	300
XC7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	8	1	6	300
XC7A200T	215,360	33,650	2,888	740	730	365	13,140	10	1	16	1	10	500

注記:
1. 7 シリーズ FPGA の各スライスには、4 つの LUT と 8 つのフリップフロップが含まれ、一部のスライスでのみ LUT を分散 RAM または SRL として使用できます。
2. 各 DSP スライスには 25 × 18 乗算器、加算器、アキュムレータが 1 つずつ含まれます。
3. ブロック RAM は基本的に 36Kb ですが、2 つの独立した 18Kb ブロックとしても使用できます。
4. 各 CMT には MMCM と PLL が 1 つずつ含まれます。
5. Artix-7 FPGA の PCI Express 用インターフェイス ブロックは最高 ×4 Gen 2 をサポートします。
6. コンフィギュレーション バンク 0 は含まれません。
7. 記載の数値に GTP トランシーバーは含まれません。

FPGA (Field Programmable Gate Array)

- Artix-7 FPGA (xc7a100tcsg324-1)



xc7a100tcsg324-1 FPGA die

CLB(Configurable Logic Block),

References

- Computer Logic Design support page
  - http://www.arch.cs.titech.ac.jp/lecture/CLD/
- 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 Nexys 4 DDR Atrix-7 FPGA
  - https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start
  - https://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog
- Frix (Feasible and Reconfigurable IBM PC Compatible SoC)
  - http://www.arch.cs.titech.ac.jp/a/Frix/
- 7シリーズFPGAデータシート：概要
  - https://japan.xilinx.com/support/documentation/data_sheets/j_ds180_7Series_Overview.pdf
- 7シリーズFPGA コンフィギャラブルロジックブロック ユーザーガイド
  - https://japan.xilinx.com/support/documentation/user_guides/j_ug474_7Series_CLB.pdf