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

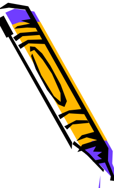
4. ハードウェア記述言語：順序回路
Hardware Description Language: Sequential Circuit

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. 2023-04-17b

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/lec4/ 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/lec4/* .
```

- code051.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 code051.v
$ ./a.out
```

code051.v 波形ビューワ GTKwave を使って波形を見る

```
$ iverilog code051.v
$ ./a.out
$ gtkwave main.vcd &
```

- code051.v をシミュレーションして，その表示を確認すること．
- シミュレーションのためのクロック信号の記述を示す．
- forever文は，続くブロックの処理を無限に繰り返す．この例では，reg型の信号r_clkを開始時に0に初期化し，#50の後にr_clkの値の反転を繰り返す．
- GTKwaveで波形を確認すると，10MHz（周期 100ns）のクロックが生成されることがわかる．

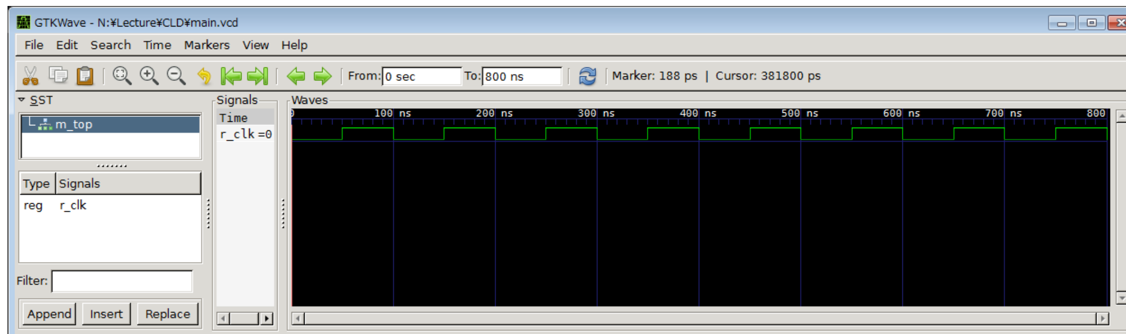
code051.v

```
module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;

  always@(*) $write("%3d %d¥n", $time, r_clk);
  initial #800 $finish;

  initial $dumpfile("main.vcd"); /* file name for GTKWave */
  initial $dumpvars(0, m_top);   /* module    for GTKWave */
endmodule
```

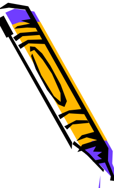
Simulation output

```
  0 0
 50 1
100 0
150 1
200 0
250 1
300 0
350 1
400 0
450 1
500 0
550 1
600 0
650 1
700 0
750 1
800 0
```

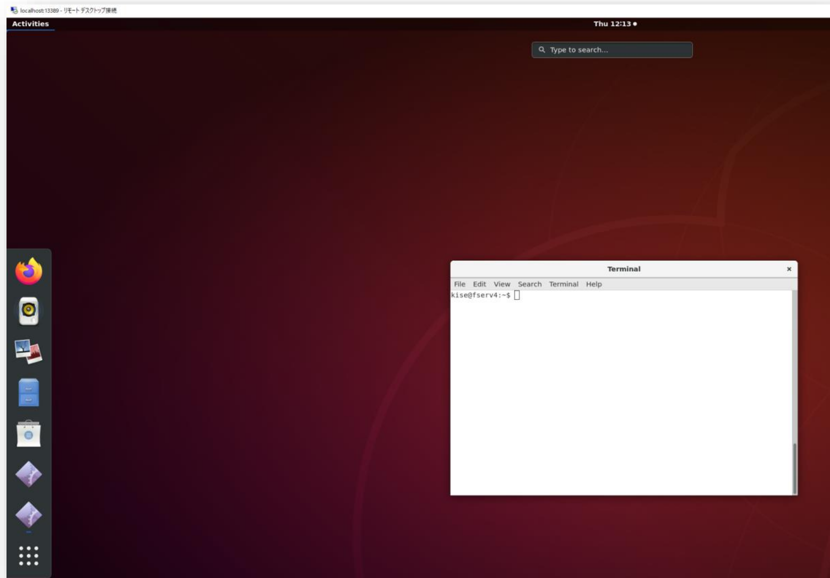
Waveform of GTKwave

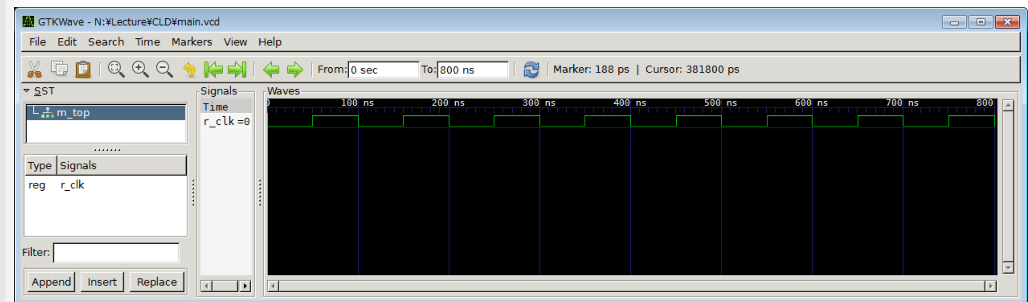


GTKwave 波形ビューワのデモンストレーション



- Vivado での GTKwave の使い方.
- ファイルの読み込み，表示する信号の追加.
- 表示範囲の調整，2進法，10進法での表示.





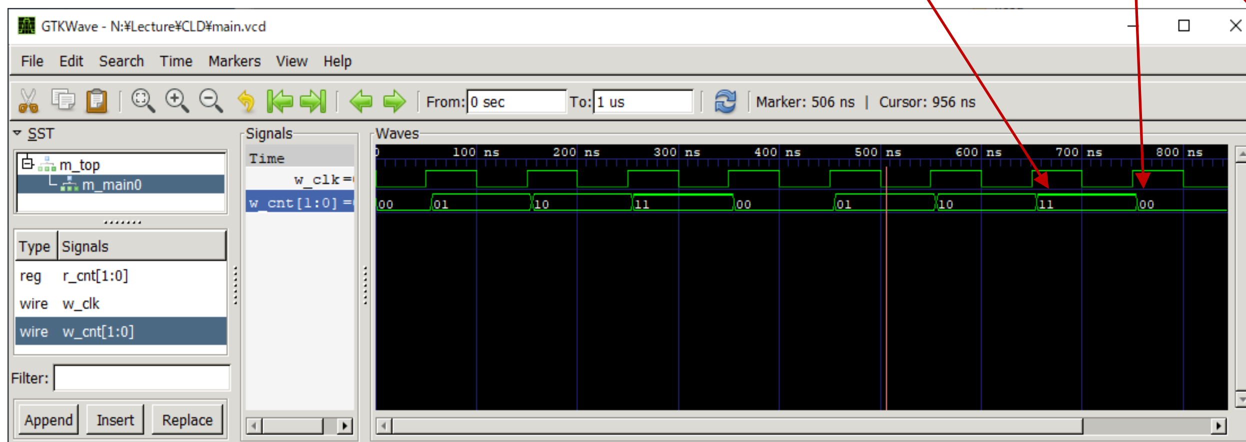
code052.v カウンタ回路を記述して波形を見る

- code052.v をシミュレーションして，その表示を確認すること.
- 単純な2ビットカウンタの記述例を示す.
- クロック信号 w_clk の立ち上がりの時 (posedge w_clk) に，1インクリメントする. ただし，r_cnt の最初の値を 0 とする。
- r_cnt の更新はクロック信号の立ち上がりから #5 だけ経過してから.
- 2ビットカウンタなので，最大値の 3 (2'b11) の次に 0 (2'b00) となる.
- iverilogでシミュレーションし，GTKWaveで波形を確認する.

```
$ iverilog code052.v
$ ./a.out
$ gtakwave a.out &
```

code052.v

```
module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;
  wire [1:0] w_cnt;
  m_main m_main0 (r_clk, w_cnt);
  initial $dumpfile("main.vcd");
  initial $dumpvars(0, m_main0);
  initial #1000 $finish;
endmodule


module m_main (w_clk, w_cnt);
  input  wire w_clk;
  output wire [1:0] w_cnt;

  reg [1:0] r_cnt = 0;
  always@(posedge w_clk) begin
      r_cnt <= #5 r_cnt + 1;
  end
  assign w_cnt = r_cnt;
endmodule
```



Waveform

code053.v regの出力はregに接続されたwire

- code053.v をシミュレーションして，その表示を確認すること.
- 単純な2ビットカウンタの別の記述例を示す.
- クロック信号 w_clk の立ち上がりの時 (posedge w_clk) に，1インクリメントする．ただし，r_cnt の最初の値を 0 とする。
- モジュールの出力ポートでは wire と reg を用いることができる.
- `timescale 1ns/100ps という記述の 1ns は #1 が実時間で 1ns であることを示し，次の 100ps でシミュレーションの精度が 100ps 単位であることを指定する.
    - timescale の行はこのまま記述しておいた方が良い.

code053.v

```
`timescale 1ns/100ps
`default_nettype none

module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;
  wire [1:0] w_cnt;
  m_main m_main0 (r_clk, w_cnt);
  initial $dumpfile("main.vcd");
  initial $dumpvars(0, m_main0);
  initial #1000 $finish;
endmodule


module m_main (w_clk, r_cnt);
  input  wire w_clk;
  output reg [1:0] r_cnt;

  initial r_cnt = 0;
  always@(posedge w_clk) r_cnt <= #5 r_cnt + 1;
endmodule
```

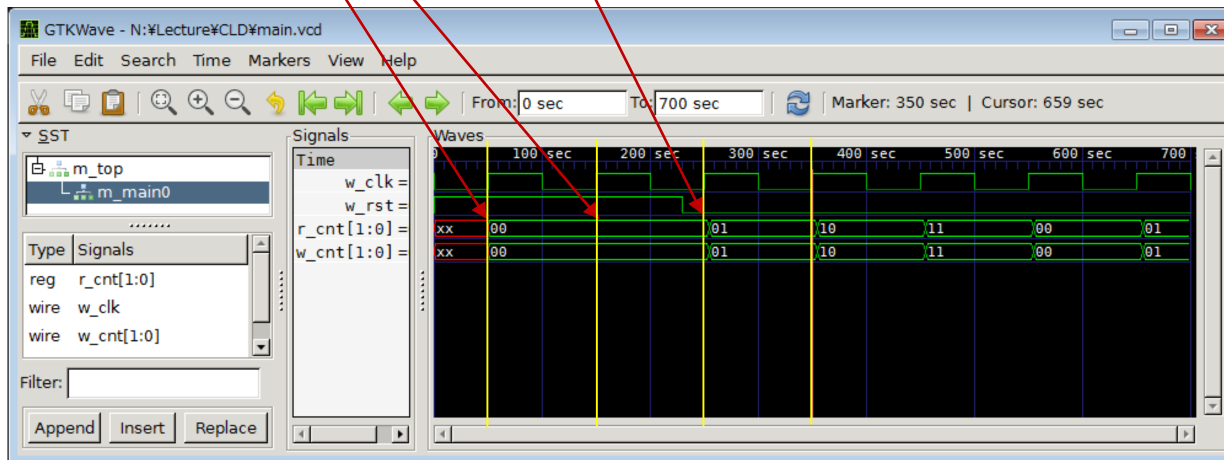
code052.v

```
module m_main (w_clk, w_cnt);
  input  wire w_clk;
  output wire [1:0] w_cnt;

  reg [1:0] r_cnt = 0;
  always@(posedge w_clk) begin
      r_cnt <= #5 r_cnt + 1;
  end
  assign w_cnt = r_cnt;
endmodule
```

code054.v 同期リセット付きのカウンタ回路

- code054.v をシミュレーションして，その表示を確認すること．
- 同期リセット付き2ビットカウンタの記述例を示す．
- クロック信号 w_clkの立ち上がりの時に，リセット信号 r_rst が1でカウンタの値はゼロで初期化され，
  そうでなければ1インクリメントされる．
- iverilogでシミュレーションし，GTKWaveで波形を確認する．

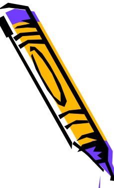
Waveform



code054.v

```
module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;
  reg r_rst=1;
  initial #230 r_rst=0;
  wire [1:0] w_cnt;
  m_main m_main0 (r_clk, r_rst, w_cnt);
  initial $dumpfile("main.vcd");
  initial $dumpvars(0, m_main0);
  initial #1000 $finish;
endmodule

module m_main (w_clk, w_rst, w_cnt);
  input  wire w_clk, w_rst;
  output wire [1:0] w_cnt;

  reg [1:0] r_cnt;
  always@(posedge w_clk) begin
    if (w_rst) r_cnt <= 0;
    else r_cnt <= #5 r_cnt + 1;
  end
  assign w_cnt = r_cnt;
endmodule
```

同期リセット，非同期リセット，リセット無し



- 同期リセットの記述（左）では，クロック信号に同期してリセットされる．
- 非同期リセットの記述（中央）では，クロック信号に同期せずに，リセット信号の立ち上がりのタイミングでリセットされる．
- 右の記述はリセット信号を使わない記述．
- 必要がなければ，リセットを使わない記述（右）が良い．
  リセットが必要であれば，同期リセット（左）を使う記述が良い．

code054.v

```
module m_main (w_clk, w_rst, w_cnt);
  input  wire w_clk, w_rst;
  output wire [1:0] w_cnt;

  reg [1:0] r_cnt;
  always@(posedge w_clk) begin
    if (w_rst) r_cnt <= 0;
    else r_cnt <= #5 r_cnt + 1;
  end
  assign w_cnt = r_cnt;
endmodule
```

同期リセットの例

```
module m_main (w_clk, w_rst, w_cnt);
  input  wire w_clk, w_rst;
  output wire [1:0] w_cnt;

  reg [1:0] r_cnt;
  always@(posedge w_clk or posedge w_rst) begin
    if (w_rst) r_cnt <= 0;
    else r_cnt <= #5 r_cnt + 1;
  end
  assign w_cnt = r_cnt;
endmodule
```

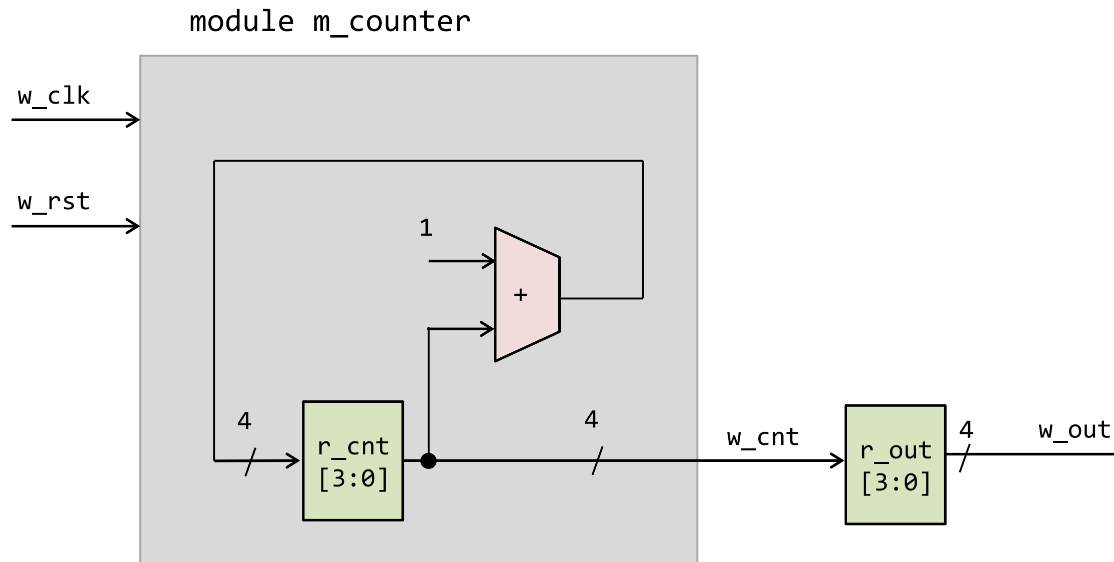
非同期リセットの例

```
module m_main (w_clk, w_cnt);
  input  wire w_clk;
  output wire [1:0] w_cnt;

  reg [1:0] r_cnt = 0;
  always@(posedge w_clk) begin
     r_cnt <= #5 r_cnt + 1;
  end
  assign w_cnt = r_cnt;
endmodule
```

リセットを使わない例

Sequential Circuit（順序回路）とレジスタの動作

- 順序回路では，クロック信号に対するレジスタの動作を正しく理解することが重要
- レジスタは，クロック信号の立ち上がりのタイミングで入力線の値を取得(sampling)して，少し遅れて取得した値を出力線に出力する.
- レジスタの更新では， initial や always でタイミングを指定すること.
- このモジュールの波形を考えよう.



code055.v, code056.v 一定の間隔で生じる動作の記述

- code055.v の 999999 を 4 に変更してシミュレーションして，その表示を確認すること.
- 1MHz のクロック信号を入力として、1秒の間隔で 0, 1, 0, 1 と変化させるハードウェアの記述例を示す. LED を点滅させる場合などに利用する。
- 補足
  - 1MB = 1024 x 1024 B
  - 1MHz = 1000 x 1000 Hz
- code055.v の2か所のalwaysブロックをまとめると, code056.v になる.

code055.v

```
`timescale 1ns/100ps
module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;
  wire w_out;
  m_main m_main0 (r_clk, w_out);
  initial $dumpfile("main.vcd");
  initial $dumpvars(0, m_main0);
  initial #1000 $finish;
endmodule


module m_main (w_clk, r_out);
  input  wire w_clk;
  output reg  r_out;

  reg [31:0] r_cnt = 0;
  always@(posedge w_clk) begin
    r_cnt <= (r_cnt==999999) ? 0 : r_cnt +1;
  end

  initial r_out = 0;
  always@(posedge w_clk) begin
    r_out <= (r_cnt==0) ? ~r_out : r_out;
  end
endmodule
```

code056.v

```
module m_main (w_clk, r_out);
  input  wire w_clk;
  output reg  r_out;

  initial r_out = 0;
  reg [31:0] r_cnt = 0;
  always@(posedge w_clk) begin
    r_cnt <= (r_cnt==999999) ? 0 : r_cnt +1;
    r_out <= (r_cnt==0) ? ~r_out : r_out;
  end
endmodule
```

code057.v 演習(1)で使ったハードウェア記述

- 100MHz のクロック信号を入力として、1秒の間隔で 0, 1, 0, 1 と変化させるハードウェアの記述例を示す. LED を点滅させる場合などに利用する.
- 今は理解できる.

code057.v

```
module m_main (w_clk, w_led);
  input  wire w_clk;
  output wire [3:0]  w_led;

  reg         r_out = 0;
  reg [31:0] r_cnt = 0;
  always@(posedge w_clk) begin
    r_cnt <= (r_cnt==99999999) ? 0 : r_cnt +1;
    r_out <= (r_cnt==0) ? ~r_out : r_out;
  end
  assign w_led = {r_out, r_out, r_out, r_out};
  // vio_0 vio_00(w_clk, w_led[3], w_led[2], w_led[1], w_led[0]);
endmodule
```

code061.v シフトレジスタ（右シフト）

- code061.v をシミュレーションして，その表示を確認すること．
- シフトレジスタは，サイクル毎に，その内容を1ビットだけ右（あるいは左）にシフトする．また，最上位ビット（あるいは最下位ビット）に入力された値を格納する．

code061.v

```
module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;
  wire [7:0] w_out;
  m_main m_main0 (r_clk, w_out);
  initial $dumpfile("main.vcd");
  initial $dumpvars(0, m_main0);
  initial #1000 $finish;
  always@(*) #1 $display("%3d: %b", $time, w_out);
endmodule


module m_main (w_clk, r_sreg);
  input  wire w_clk;
  output reg [7:0] r_sreg = 8'b10101111;

  wire w_in = 0;
  always@(posedge w_clk) begin
    r_sreg <= {w_in, r_sreg[7:1]};
  end
endmodule
```

右シフト

Simulation output

```
  1: 10101111
 51: 01010111
151: 00101011
251: 00010101
351: 00001010
451: 00000101
551: 00000010
651: 00000001
751: 00000000
```

code062.v シフトレジスタ（左シフト）

- code062.v をシミュレーションして，その表示を確認すること．
- シフトレジスタは，サイクル毎に，その内容を1ビットだけ右（あるいは左）にシフトする．また，最上位ビット（あるいは最下位ビット）に入力された値を格納する．

code062.v

```
module m_top ();
  reg r_clk=0;
  initial forever #50 r_clk = ~r_clk;
  wire [7:0] w_out;
  m_main m_main0 (r_clk, w_out);
  initial $dumpfile("main.vcd");
  initial $dumpvars(0, m_main0);
  initial #1000 $finish;
  always@(*) #1 $display("%3d: %b”, $time, w_out);
endmodule


module m_main (w_clk, r_sreg);
  input  wire w_clk;
  output reg [7:0] r_sreg = 8’b10101111;

  wire w_in = 0;
  always@(posedge w_clk) begin
    r_sreg <= {r_sreg[6:0], w_in};
  end
endmodule
```

左シフト

Simulation output

```
  1: 10101111
 51: 01011110
151: 10111100
251: 01111000
351: 11110000
451: 11100000
551: 11000000
651: 10000000
751: 00000000
```

code064.v EXOR（排他的論理和）ゲート

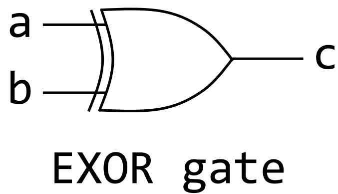
- code064.v をシミュレーションして，その表示を確認すること．
- 論理演算子には，
  単項演算子の ~ (NOT), 2項演算子として & (AND), | (OR), ^ (EXOR)がある．

code064.v

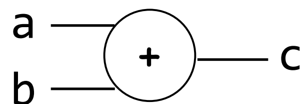
```
module m_top ();
  reg r_a, r_b;
  wire w_c;
  assign w_c = r_a ^ r_b;

  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 $display("%2d: %d %d -> %d", $time, r_a, r_b, w_c);
endmodule
```

Simulation output

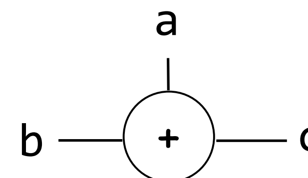
```
11: 0 0 -> 0
21: 0 1 -> 1
31: 1 0 -> 1
41: 1 1 -> 0
```



EXOR gate



1ビットの加算器



code065.v 3入力の EXOR ゲート

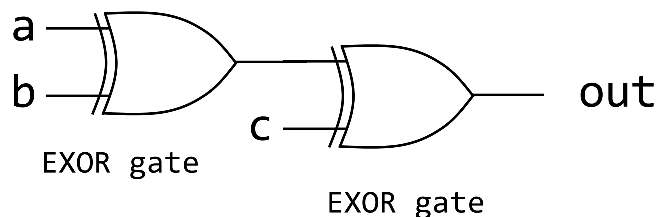
- code065.v をシミュレーションして，その表示を確認すること．
- 論理演算子には，2項演算子として & (AND), | (OR), ^ (EXOR)がある．
- 3入力以上の論理演算子を考えよう．

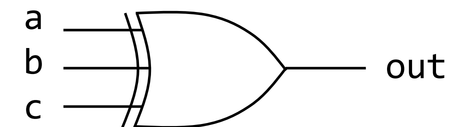
```
module m_top ();
  reg r_a, r_b, r_c;
  wire w_out;
  assign w_out = r_a ^ r_b ^ r_c;

  initial begin
    #10 r_a <= 0; r_b <= 0; r_c <= 0;
    #10 r_a <= 0; r_b <= 0; r_c <= 1;
    #10 r_a <= 0; r_b <= 1; r_c <= 0;
    #10 r_a <= 0; r_b <= 1; r_c <= 1;
    #10 r_a <= 1; r_b <= 0; r_c <= 0;
    #10 r_a <= 1; r_b <= 0; r_c <= 1;
    #10 r_a <= 1; r_b <= 1; r_c <= 0;
    #10 r_a <= 1; r_b <= 1; r_c <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %d", $time, r_a, r_b, r_c, w_out);
endmodule
```

code065.v

Simulation output

```
11: 0 0 0 -> 0
21: 0 0 1 -> 1
31: 0 1 0 -> 1
41: 0 1 1 -> 0
51: 1 0 0 -> 1
61: 1 0 1 -> 0
71: 1 1 0 -> 0
81: 1 1 1 -> 1
```





code066.v 3入力の EXOR ゲートのバスを用いた記述

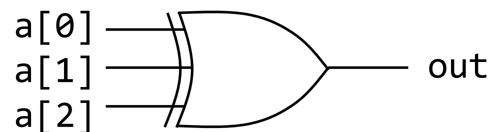
- code066.v をシミュレーションして，その表示を確認すること．

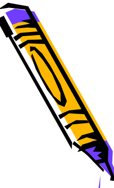
code066.v

```
module m_top ();
  reg [2:0] r_a;
  wire w_out;
  assign w_out = ^r_a;

  initial begin
    #10 r_a <= 3'b000;
    #10 r_a <= 3'b001;
    #10 r_a <= 3'b010;
    #10 r_a <= 3'b011;
    #10 r_a <= 3'b100;
    #10 r_a <= 3'b101;
    #10 r_a <= 3'b110;
    #10 r_a <= 3'b111;
  end
  always@(*) #1 $display("%2d: %b -> %d", $time, r_a, w_out);
endmodule
```

Simulation output

```
11: 000 -> 0
21: 001 -> 1
31: 010 -> 1
41: 011 -> 0
51: 100 -> 1
61: 101 -> 0
71: 110 -> 0
81: 111 -> 1
```





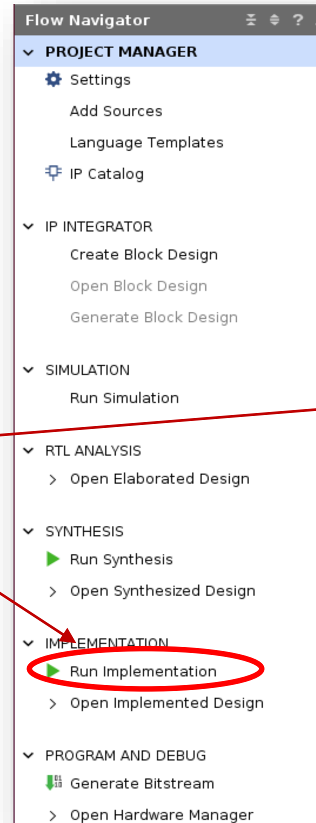
演習(2) 加算器のクリティカルパスの遅延を計測する

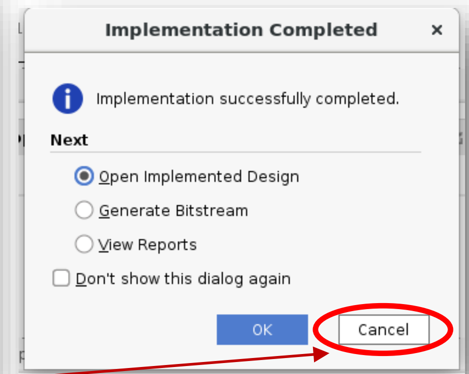
- code078.v を修正して，100MHzの動作周波数の制約を満たす **n**-bit Adder の最大の **n** を求めること．ただし，n は5の倍数とする．
  - code078.v を用いて合成する(Run Implementation)．Bitstreamは生成する必要はない．
  - 1行目の D_N の値を変化させて合成．Failed Timing! と出力された時は制約を満たしていない．
  - 1行目の D_N の値を小さくして合成．Implementation Complete が出力された時は満たしている．

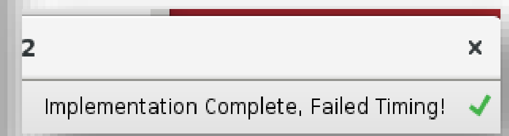
code078.v

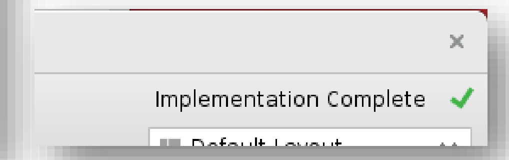
```
`define D_N 32

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

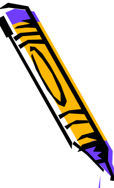




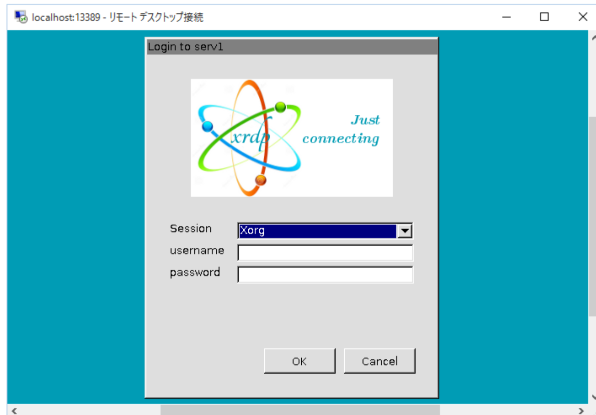


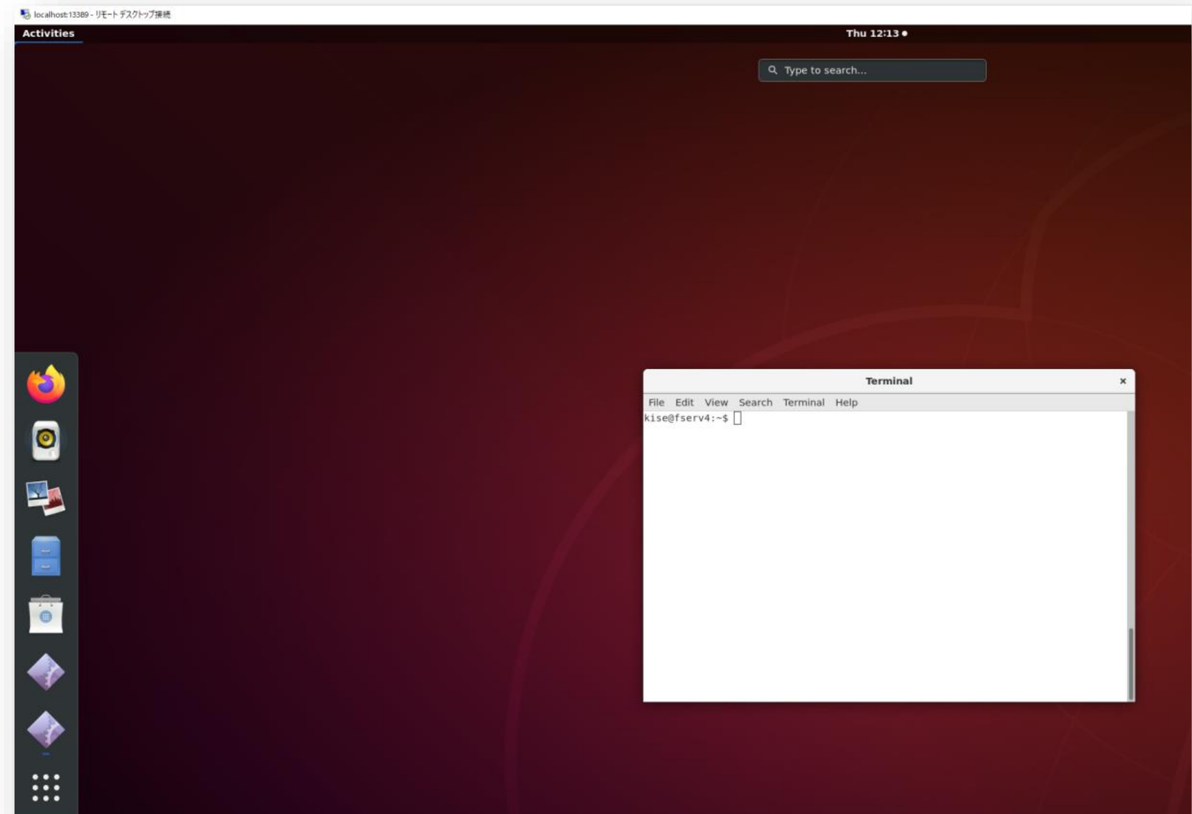


ACRiルームのデモンストレーション

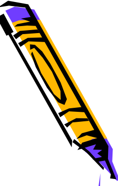


- Vivado での VIO の使い方.
- Vivado での HW manager, オープンしたデザイン, ソースの切り替え方法.
- WNS
- Vivado における回路図(schematic)の表示.





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