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

2. ハードウェア記述言語：組合せ回路 (1)
Hardware Description Language: Combinational Circuit (1)

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. 2023-04-13a

Verilog HDL

- ハードウェア記述言語 (Hardware Description Language)
  - IEEE 1364 として標準化
- C言語に近い（C言語を参考にしている）

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/lec2/ 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/lec2/* .
```

- code001.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 code001.v
$ ./a.out
```

code001.v モジュールの定義と文字列の表示

```
$ iverilog code001.v
$ ./a.out
```

- code001.v をシミュレーションして，その表示を確認すること．
- モジュールの定義はキーワードmoduleからキーワードendmoduleまで．
- moduleの後にモジュール名を書く．この例ではmainがモジュール名．
- モジュール名の後の括弧内に入出力の端子名を列挙する．ここでは端子は何も定義していない．
- セミコロン(;)で，モジュール名と端子の列挙を終える．
- キーワードinitialにより，シミュレーション開始時（時刻0）から処理を始めることを指定する．
- $display または $write はシステムタスクの1つで，メッセージを出力する． $write では改行されない．書式はC言語のprintfと同様．

Verilog HDL code (code001.v)

```
module main ();
  initial $display("hello, world");
endmodule
```

Simulation output

```
hello, world
```

Verilog HDLのコードは青色で，シミュレーションの出力は黄色で示す．

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．

code002.v ブロックの指定とコンパイルエラーの対処

- code002.v をシミュレーションして，その表示を確認すること．
- 2つのシステムタスク$displayを用いた出力の例．2つのシステムタスクをブロックとしてまとめている．
- ブロックはキーワードbeginで始まり，キーワードendで終わる．C言語の { } に対応．
- code002_ng1.vは2番目の$displayがinitialブロックに含まれないので文法エラーとなる．

code002.v

```
module main ();
  initial begin
    $display("hello, world");
    $display("in Verilog HDL");
  end
endmodule
```

```
hello, world
in Verilog HDL
```

code002_ng.v

```
module main ();
  initial $display("hello, world");
  $display("in Verilog HDL");
endmodule
```

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．

code003.v 複数のinitialの利用

- code003.v をシミュレーションして，その表示を確認すること．
- モジュール内で複数の initial を用いても良い．
  code002.v と code003.v の出力は同じ．

code002.v

```
module main ();
  initial begin
    $display("hello, world");
    $display("in Verilog HDL");
  end
endmodule
```

```
hello, world
in Verilog HDL
```

code003.v

```
module main ();
  initial $display("hello, world");
  initial $display("in Verilog HDL");
endmodule
```

```
hello, world
in Verilog HDL
```

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．

code005.v 指定した時間が経過するまで待たせる命令#

- code005.v をシミュレーションして，その表示を確認すること．
- 指定した時間が経過するまで待たせる命令# を用いた例．
- #200 により，ここではシミュレーション開始時（時刻0）から200の時間が経過した時刻200に hello, world を表示する．
- #100 により，ここではシミュレーション開始時（時刻0）から100の時間が経過した時刻100に in Verilog HDL を表示する．
- 1行目はコメント，Verilog HDLのコメントはC, C++と同様．
- 時間の単位は nsec とする．#300 は 300nsec の時間経過を表す．

code005.v

```
/* sample Verilog code */
module main ();
  initial #200 $display("hello, world");
  initial #100 $display("in Verilog HDL");
endmodule
```

```
in Verilog HDL
hello, world
```

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．

code006.v 複数のinitialを用いたコードの時間制御の例

- code006.v をシミュレーションして，その表示を確認すること．
- $displayによる出力の順番はどうなるか？

code006.v

```
module main ();
  initial #200 $display("hello, world");
  initial begin
    #100 $display("in Verilog HDL");
    #150 $display("When am I displayed?");
  end
endmodule
```

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．

code007.v シミュレーションはいつ終わるのか

- code007.v をシミュレーションして，その表示を確認すること．
- 出力はどうなるか？

- Vivadoを用いてシミュレーションする場合，デフォルトの設定では1000nsしかシミュレーションしないので Verilog is easy? は出力されない．

code007.v

```
module main ();
  initial #200 $display("hello, world");
  initial begin
    #100 $display("in Verilog HDL");
    #150 $display("When am I displayed?");
    #1000 $display("Verilog is easy?");
  end
endmodule
```

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．

code008.v システムタスク$time

- code008.v をシミュレーションして，その表示を確認すること．
- システムタスク$timeは，64ビットのシミュレーション時刻を返す．
- このコードでは，それぞれの $display が表示する時刻を表示する．
- 複雑な回路のシミュレーションでは，どの出力がどの時刻に出力されたのかわかりにくい場合がある．その場合，この例のように時刻を出力すると良い．

code008.v

```
module main ();
  initial #200 $display("%3d hello, world", $time);
  initial begin
    #100 $display("%3d in Verilog HDL", $time);
    #150 $display("%3d When am I displayed?", $time);
  end
endmodule
```

```
100 in Verilog HDL
200 hello, world
250 When am I displayed?
```

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．

code009.v システムタスク$finish

- code009.v をシミュレーションして，その表示を確認すること．
- システムタスク$finishは，シミュレーションを終了させる．
- このコードでは時刻210でシミュレーションが終了する．
- Vivadoのデフォルトの設定では1000nsシミュレーションするが，それより短い時間のシミュレーションや，ある条件でシミュレーションを終了させたい場合に用いると良い．

code009.v

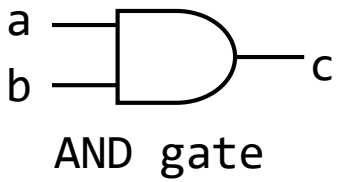
```
module main ();
  initial #200 $display("%3d hello, world", $time);
  initial begin
    #100 $display("%3d in Verilog HDL", $time);
    #150 $display("%3d When am I displayed?", $time);
  end
  initial #210 $finish;
endmodule
```

```
100 in Verilog HDL
200 hello, world
```

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．

code011.v ANDゲートと重要な記述の幾つか

- code011.v をシミュレーションして，その表示を確認すること．
- reg型の信号a, bを宣言する．reg型はC言語の変数に相当する．
- wire型の信号cを宣言する．wire型はハードウェア記述言語に固有のもの．
- 継続的代入assign は，wire型の信号cをa & bに接続する．initialブロックやalways@ブロックの外に記述する．
- & はANDの論理演算子．
- always@ブロックは，@以降に書かれた事象が発生するたびに繰り返し実行される．always@(*) では，何らかの入力が変化した事象となる．
- initialブロックの中の <= はノンブロッキング代入と呼ばれ，reg型の信号への代入を表す． a <= 0; はreg型の信号aに値0を代入する．wire型にノンブロッキング代入は使えない．



AND gate

code011.v

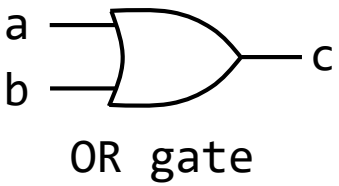
```
module main ();
  reg a, b;
  wire c;
  assign c = a & b;

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

Simulation output

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

code012.v ORゲート，論理演算子，算術演算子

- code012.v をシミュレーションして，その表示を確認すること．
- | はORの論理演算子．
- 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．
- 算術演算子には，+ (加算), - (減算), * (乗算), / (除算), % (剰余)がある．
- これらの論理演算子，算術演算子はC言語と同じ．



OR gate

code012.v

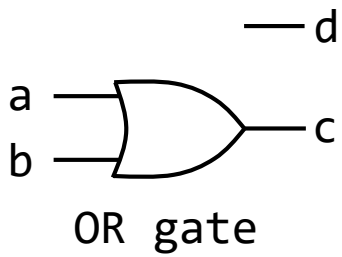
```
module main ();
  reg a, b;
  wire c;
  assign c = a | b;

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

Simulation output

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

code013.v ORゲート，不定値とハイインピーダンス

- code013.v をシミュレーションして，その表示を確認すること．
- wire dはどこにも接続されていない．シミュレーション結果は？
- 信号線の取り得る値には，0, 1, x, z がある．xは不定値を，zはハイインピーダンスを表す．
- どこにも接続されていないwire，あるいは明示的にハイインピーダンスに設定したwireはzとなる．
- 初期化されていないreg型の信号や，不定値を用いた演算結果などはxとなる．
- 意図的にx, zとしていないのにx, zが出力される場合，コードに記述ミスがあることが多いので，コードの記述を見直すと良い．



code013.v

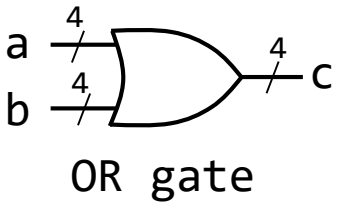
```
module main ();
  reg a, b;
  wire c, d;
  assign c = a | b;

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

Simulation output

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

code014.v 複数本の信号線(バス), 数値の表現

- code014.v をシミュレーションして, その表示を確認すること.
- Verilog HDLでは, 2本以上の信号線の束をバス(bus)と呼ぶ.
- reg型, wire型の信号線をバスとして宣言するには, reg, wire の後に [3:0] の様に本数を指定する. 例えば reg [3:0] a は, a[3], a[2], a[1], a[0]の4本から成るバスを宣言する.
- code014.v では, 4ビット幅のバスとしてreg型a, bを, 4ビット幅のバスとしてwire型cを宣言する.
- 数値を表現するためには, '(シングルクォーテーション)より前の数字がビット幅を表し, 'の後のbが2進法であることを表す(その他, 16進法h, 10進法d, 8進法oがある). 例えば, 4'b1010 は2進法で示された4ビットの1010となる. 数値の表現では大文字, 小文字は区別されない. 4'b1010 と 4'B1010 と 4'hAは同じ値となる.
  ビット幅を省略すると32ビットとなる. 基数を指定しないと10進法となる.
- システムタスク$displayでは, 2進法で表示するための %b を利用できる.



code014.v

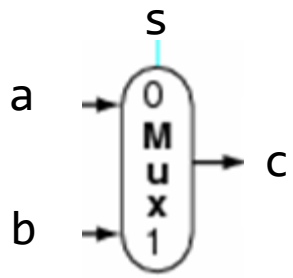
```
module main ();
  reg  [3:0] a, b;
  wire [3:0] c;
  assign c = a | b;

  initial begin
    #10 a <= 4'b1010; b <= 4'b1100;
  end
  always@(*) #1 $display("%2d: %b %b -> %b", $time, a, b, c);
endmodule
```

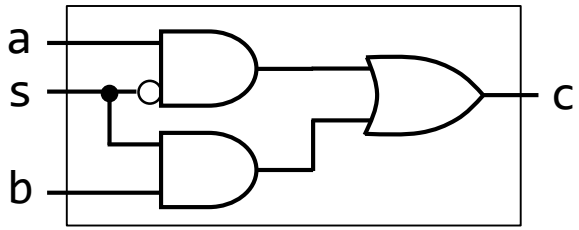
Simulation output

```
11: 1010 1100 -> 1110
```

code015.v マルチプレクサ

- code015.v をシミュレーションして，その表示を確認すること．
- Multiplexer（マルチプレクサ）のVerilog HDL記述を考える．
  - s が 0 であれば a を出力として，s が1であれば b を出力とする回路．





code015.v

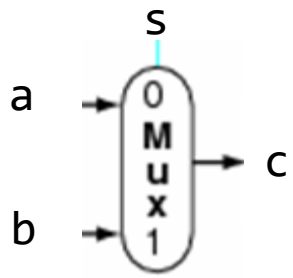
```
module main ();
  reg  a, b, s;
  wire c;
  assign c = (a & ~s) | (s & b);
  initial begin
    #10 s <= 0; a <= 0; b <= 0;
    #10 s <= 0; a <= 0; b <= 1;
    #10 s <= 0; a <= 1; b <= 0;
    #10 s <= 0; a <= 1; b <= 1;
    #10 s <= 1; a <= 0; b <= 0;
    #10 s <= 1; a <= 0; b <= 1;
    #10 s <= 1; a <= 1; b <= 0;
    #10 s <= 1; a <= 1; b <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %b", $time, s, a, b, c);
endmodule
```

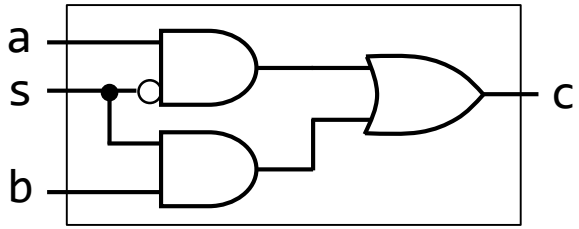
Simulation output

```
      s, a, b    c
11:  0  0  0 -> 0
21:  0  0  1 -> 0
31:  0  1  0 -> 1
41:  0  1  1 -> 1
51:  1  0  0 -> 0
61:  1  0  1 -> 1
71:  1  1  0 -> 0
81:  1  1  1 -> 1
```

code016.v 3項演算子とマルチプレクサ

- code016.v をシミュレーションして，その表示を確認すること．
- マルチプレクサのVerilog HDL記述を考える．
- 3項演算子の条件演算子 (? :) を使っても同じ結果になる．この記述の方が簡潔でわかりやすい．





code016.v

```
module main ();
  reg  a, b, s;
  wire c;
  assign c = s ? b : a;
  initial begin
    #10 s <= 0; a <= 0; b <= 0;
    #10 s <= 0; a <= 0; b <= 1;
    #10 s <= 0; a <= 1; b <= 0;
    #10 s <= 0; a <= 1; b <= 1;
    #10 s <= 1; a <= 0; b <= 0;
    #10 s <= 1; a <= 0; b <= 1;
    #10 s <= 1; a <= 1; b <= 0;
    #10 s <= 1; a <= 1; b <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %b", $time, s, a, b, c);
endmodule
```

Simulation output

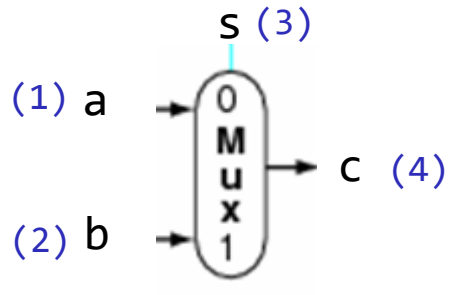
```
      s, a, b    c
11: 0 0 0 -> 0
21: 0 0 1 -> 0
31: 0 1 0 -> 1
41: 0 1 1 -> 1
51: 1 0 0 -> 0
61: 1 0 1 -> 1
71: 1 1 0 -> 0
81: 1 1 1 -> 1
```

code017.v モジュールのインスタンス化

- code017.v をシミュレーションして，その表示を確認すること．
- モジュール名とインスタンス名を記述して，入出力端子名を列挙する．列挙した順序で配線が接続される．
- C言語の関数呼び出しに似ている．この例では m_mux というモジュール名のインスタンス m_mux0 を生成し，m_topモジュールの a, b, s, c をインスタンス m_mux0 の a, b, s, c に接続している．

code017.v

```
module m_top ();
  reg  a, b, s;
  wire c;
  initial begin
    #10 s <= 0; a <= 0; b <= 0;
    #10 s <= 0; a <= 0; b <= 1;
    #10 s <= 0; a <= 1; b <= 0;
    #10 s <= 0; a <= 1; b <= 1;
    #10 s <= 1; a <= 0; b <= 0;
    #10 s <= 1; a <= 0; b <= 1;
    #10 s <= 1; a <= 1; b <= 0;
    #10 s <= 1; a <= 1; b <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %b", $time, s, a, b, c);
  m_mux m_mux0 (a, b, s, c);
endmodule
            (1) (2) (3) (4)
module m_mux (a, b, s, c);
  input  wire a, b, s;
  output wire c;
  assign c = s ? b : a;
endmodule
```



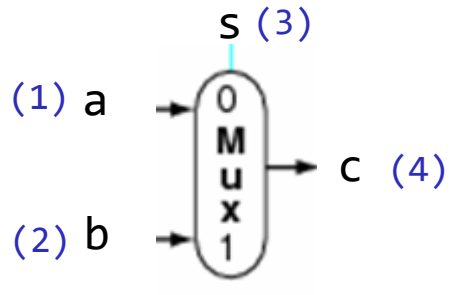
Simulation output

```
     s, a, b    c
11:  0  0  0 -> 0
21:  0  0  1 -> 0
31:  0  1  0 -> 1
41:  0  1  1 -> 1
51:  1  0  0 -> 0
61:  1  0  1 -> 1
71:  1  1  0 -> 0
81:  1  1  1 -> 1
```

モジュールのインスタンス化の補足

- モジュール名とインスタンス名を記述して，入出力端子名を列挙する．列挙した順序で配線が接続される．
- この例では m_mux というモジュール名のインスタンス m_mux0 を生成し，m_topモジュールの a, b, s, c をインスタンス m_mux0 の w_a, w_b, w_s, w_c に順番に接続している．
- インスタンス化するモジュールの入出力端子名とそれを利用するモジュールの配線の名前は一致しなくても良い．

```
module m_top ();
  reg  a, b, s;
  wire c;
  initial begin
    #10 s <= 0; a <= 0; b <= 0;
    #10 s <= 0; a <= 0; b <= 1;
    #10 s <= 0; a <= 1; b <= 0;
    #10 s <= 0; a <= 1; b <= 1;
    #10 s <= 1; a <= 0; b <= 0;
    #10 s <= 1; a <= 0; b <= 1;
    #10 s <= 1; a <= 1; b <= 0;
    #10 s <= 1; a <= 1; b <= 1;
  end
  always@(*) #1 $display("%2d: %d %d %d -> %b", $time, s, a, b, c);
  m_mux m_mux0 (a, b, s, c);
endmodule
                (1)  (2)   (3)   (4)
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
```



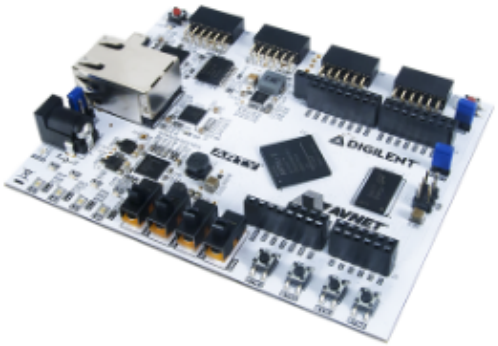
Simulation output

```
     s, a, b    c
11:  0  0  0 -> 0
21:  0  0  1 -> 0
31:  0  1  0 -> 1
41:  0  1  1 -> 1
51:  1  0  0 -> 0
61:  1  0  1 -> 1
71:  1  1  0 -> 0
81:  1  1  1 -> 1
```

ACRiルームで利用するFPGAボード Digilent Arty A7-35T

Arty A7

The Arty A7, formerly known as the Arty, is a ready-to-use development platform designed around the Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. It was designed specifically for use as a MicroBlaze Soft Processing System. When used in this context, the Arty A7 becomes the most flexible processing platform you could hope to add to your collection, capable of adapting to whatever your project requires. Unlike other Single Board Computers, the Arty A7 isn't bound to a single set of processing peripherals: One moment it's a communication powerhouse chock-full of UARTs, SPIs, IICs, and an Ethernet MAC, and the next it's a meticulous timekeeper with a dozen 32-bit timers.



Store

Reference Manual | Technical Support

Arty A7

Artix-7 FPGA Development Board

Features

- Programmable over JTAG and Quad-SPI Flash
- On-chip analog-to-digital converter

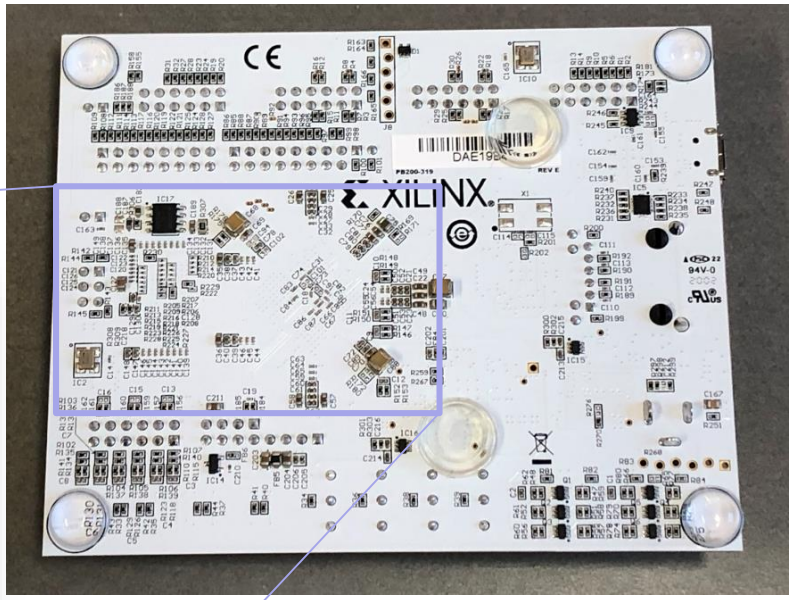
Key Specifications

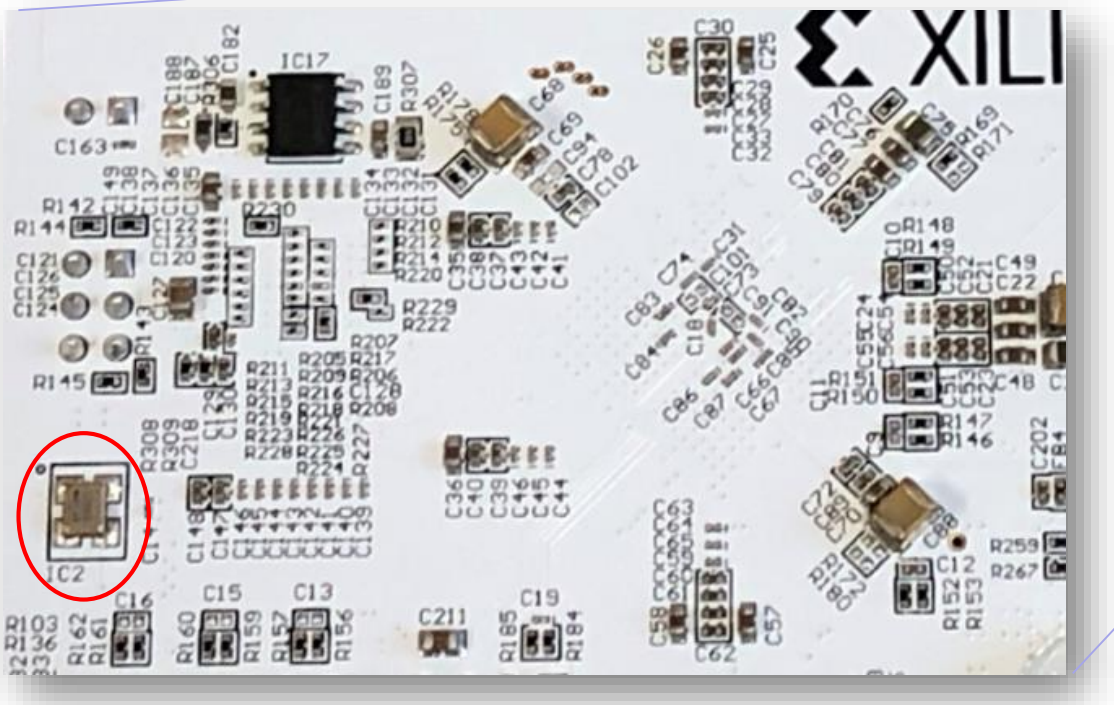
FPGA Part #	XC7A35TICSG324-1L (XC7A100TCSG324-1*)
Logic Slices	5,200 (15,850*)
Block RAM	1,800 Kbits (4,860* Kbits)
DSP Slices	90 (240*)
DDR3	256 MB @ 667 MHz
Internal clock	450 MHz+
Quad-SPI Flash	16 MB
Ethernet	10/100 Mbps

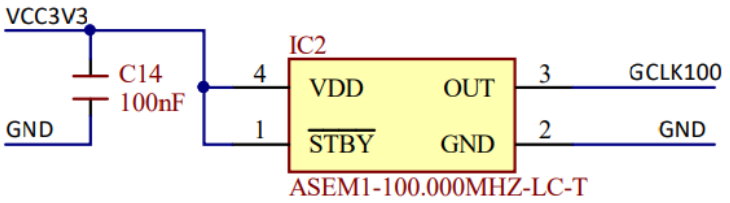
https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start

ASEM1-100.000MHZ-LC-T 100MHz 発振器

- FPGA ボードに実装されている発振器が 100MHz のクロック信号を生成し，それが FPGA の E3 という名前のピンに接続される．（FPGA の入力信号となる．）







FPGA constraint file, XDC (Xilinx Design Constraints)

- main11.xdc の内容を Ubuntu のターミナルあるいは Vivado で確認すること.
- 拡張子が xdc のファイルは，制約 (constraint) を与えるために利用する.
- 制約ファイル main11.xdc の1行目では，w_clk という信号を E3 というピン（100MHzのクロック信号）に割り当てる制約を追加する.
  - w_clk は論理合成のためのトップモジュール m_main としてVerilog HDL記述で列挙した信号名
- 信号をピンを割り当てる制約が無い場合，その信号はVivadoによって自動的に適切なピンに割り当てられる.
- 2行目で、入力ピン w_clk が，10.00ns (100MHz)のクロックであることを指定する.
- このピンを LVCMOS33 (low voltage CMOS 3.3V) とする制約を追加している. この制約について，本演習では詳細を理解する必要はない.

main11.xdcの最初の2行

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

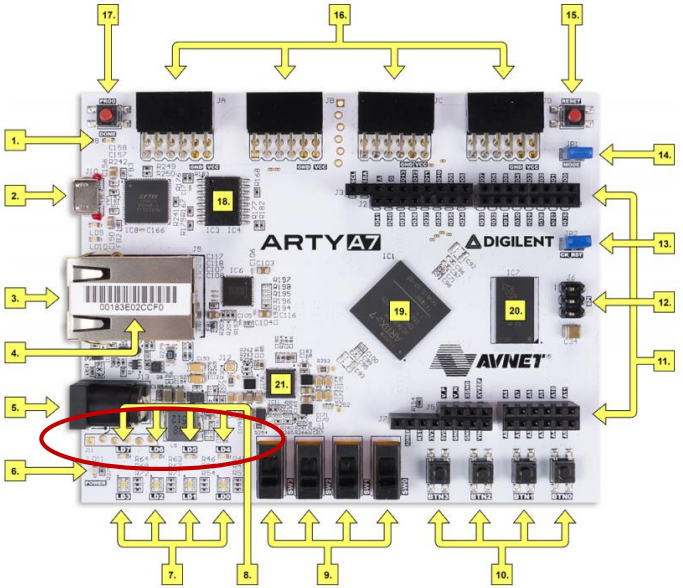
```
module m_main (w_clk);
  input  wire w_clk;
  .
  .
  .
```

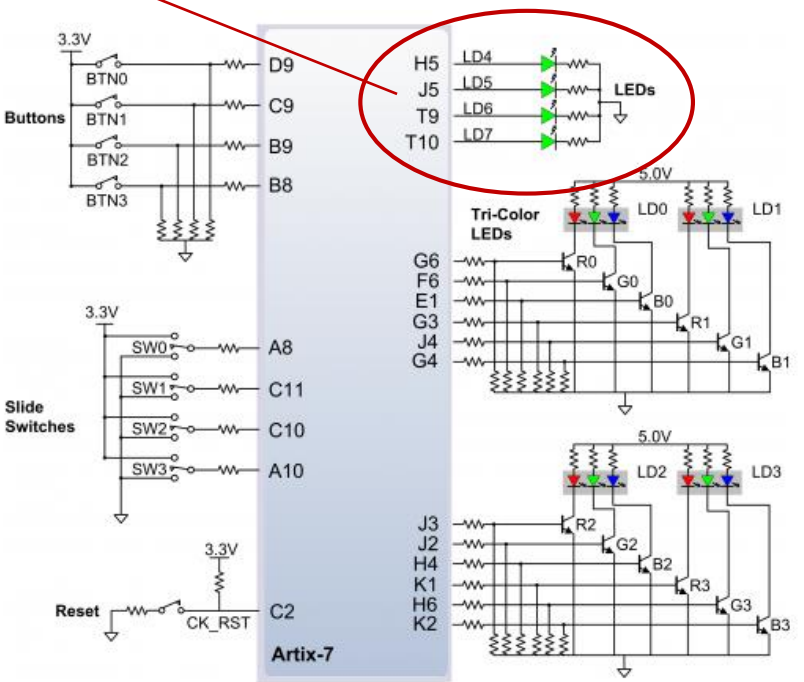
FPGA constraint file, XDC (Xilinx Design Constraints)

- main11.xdc の2行目以降では，w_led[0] の信号を H5 のピンに割り当てる制約を追加する．
  同様に，w_led[1], w_led[2], w_led[3] に，J5, T9, T10 のピンを割り当てる．

main11.xdc

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

set_property -dict { PACKAGE_PIN H5  IOSTANDARD LVCMOS33} [get_ports { w_led[0] }];
set_property -dict { PACKAGE_PIN J5  IOSTANDARD LVCMOS33} [get_ports { w_led[1] }];
set_property -dict { PACKAGE_PIN T9  IOSTANDARD LVCMOS33} [get_ports { w_led[2] }];
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports { w_led[3] }];
```





	Description	Callout	Description	Callout	Description
	FPGA programming DONE LED	8	User RGB LEDs	15	chipKIT processor reset

ACRi ルームで使う2種類のパスワード

- ACRi ルームに関するよくある質問(FAQ)
  - https://gw.acri.c.titech.ac.jp/wp/manual/faq
- 現在のACRiルームでは，次の2種類のアカウントを用いて運用

  [1] ACRiルームのWeb上の予約システムのアカウント

  [2] ACRiルームのLinuxサーバにログインするためのアカウント

  混乱を避けるために，[1] と [2] のパスワードを同じものに設定すると良い．

> **パスワードを忘れてしまいました。**
>
> 予約システムのパスワードを忘れた場合には、ログイン画面の「パスワードをお忘れですか？」のリンクから再発行の手続きを行ってください。
>
> サーバーのパスワードを忘れた場合には、acri-room at acri dot c.titech.ac.jp (at, dot は適切に置き換え) まで問い合わせてください。

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