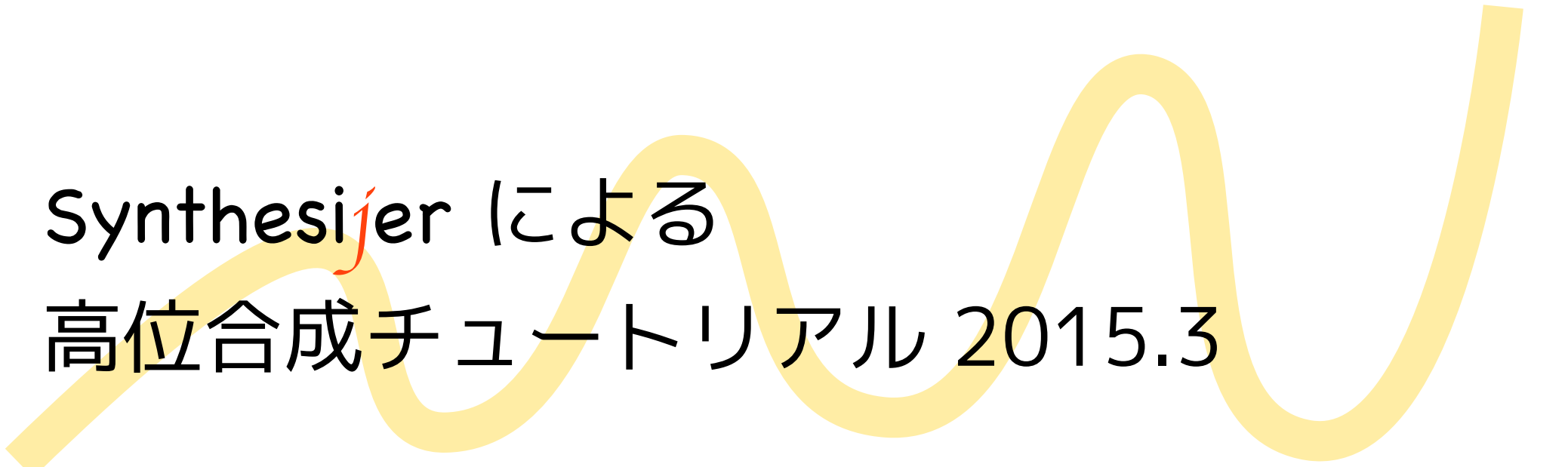




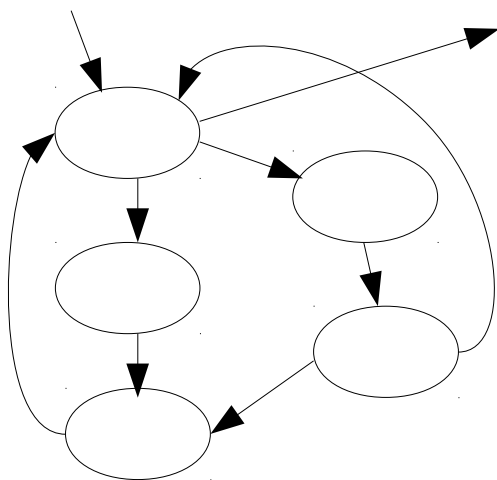
Synthesi^jer による 高位合成チュートリアル 2015.3

わさらぼ合同会社 三好 健文

2015.3.5

Synthesijer とは

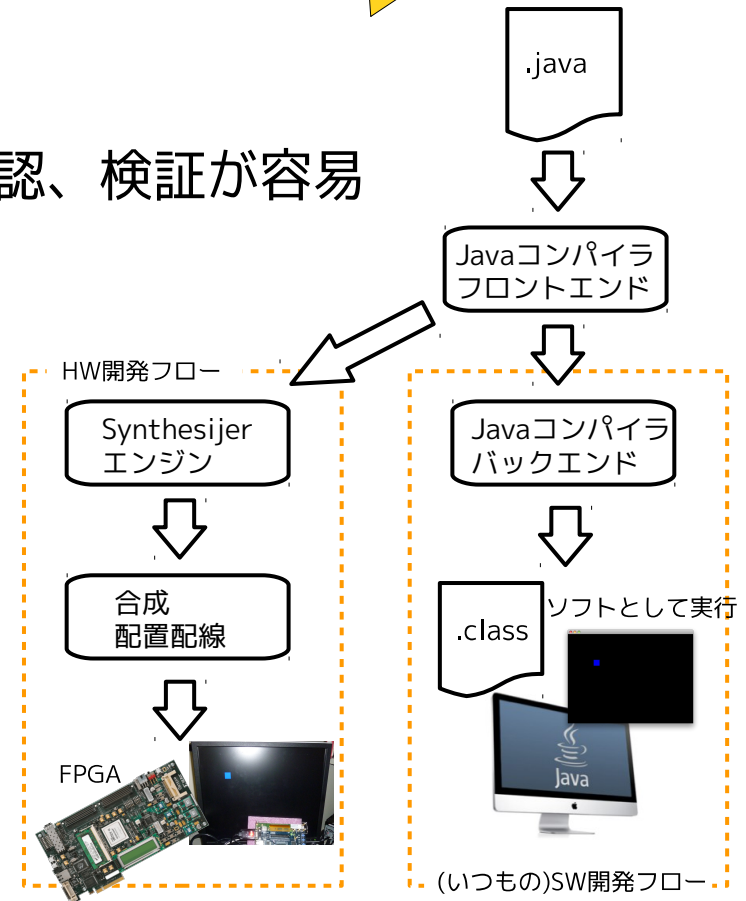
- ✓ JavaプログラムをFPGA上のハードウェアに変換
 - ✓ 複雑なアルゴリズムのハードウェア実装を楽に
 - ✓ オブジェクト指向設計による再利用性の向上
- ✓ 特殊な記法, 追加構文はない
 - ✓ ソフトウェアとして実行可能. 動作の確認、検証が容易
 - ✓ 書けるプログラムに制限は加える
(動的なnew, 再帰は不可など)



```
while(){  
  if(...){  
    ...  
  }else{  
    ...  
  }  
  ...  
}
```

複雑な状態遷移も, Javaの制御構文を使って楽に設計できる

Open-source



同じJavaプログラムをソフトウェアとしても
FPGA上のハードウェアとしても実行可能

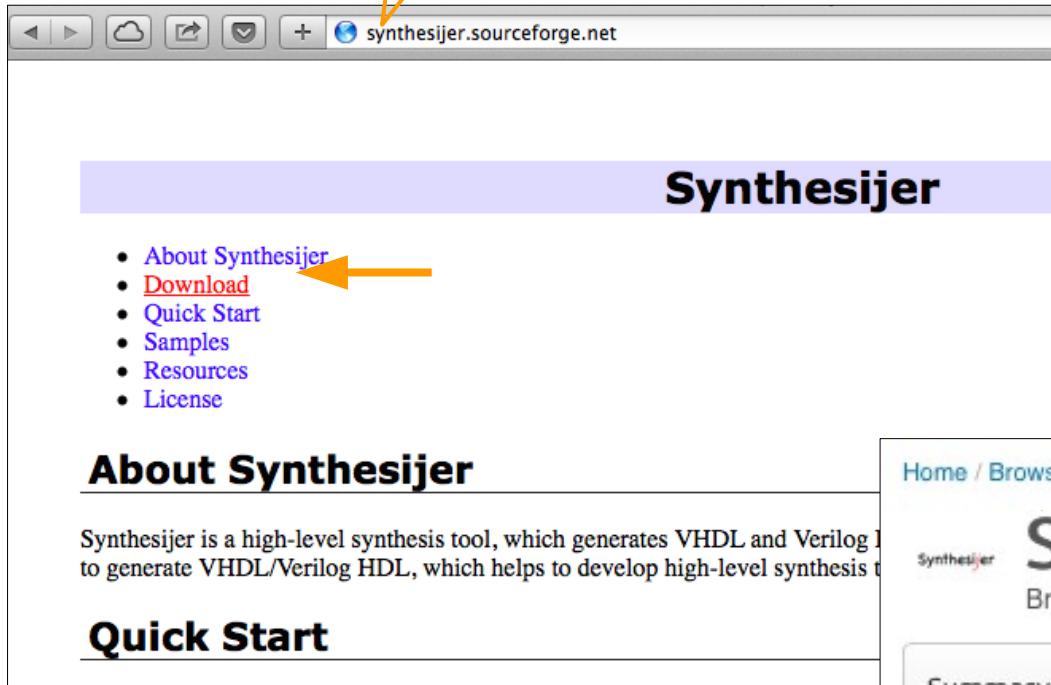
Synthesi^jer クイックスタート

クイックスタート 1/8

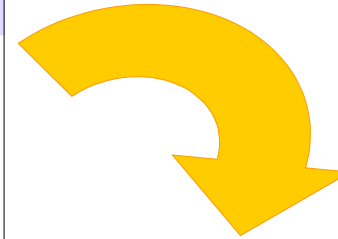
(1) バイナリパッケージをダウンロードします

<http://synthesijer.sourceforge.net>

- ✓ JDK7が必要です.
- ✓ Eclipseを使って開発してみます.

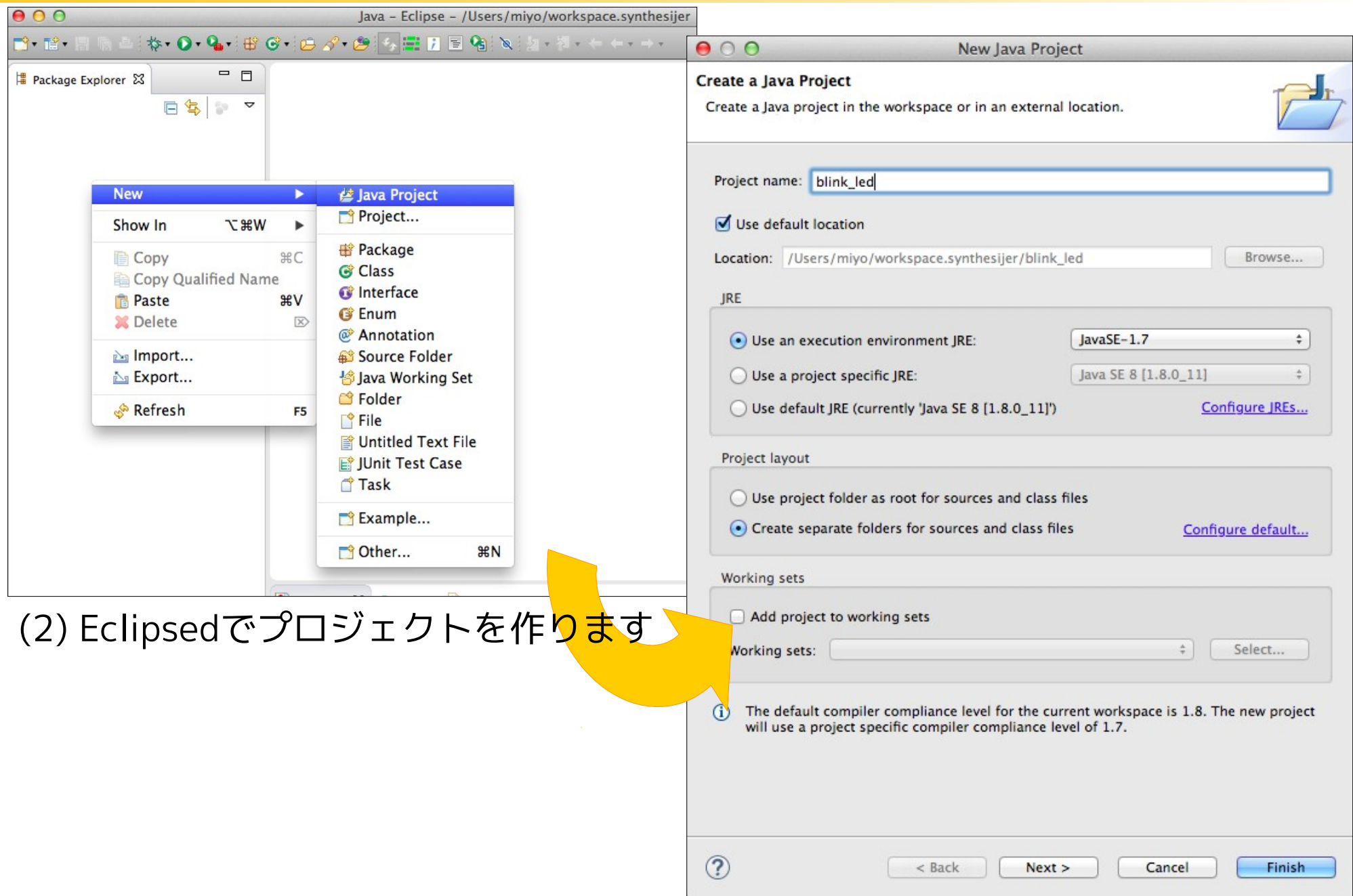


ダウンロードページに遷移します



なるべく日付の新しいものをDLしてください

クイックスタート 2/8

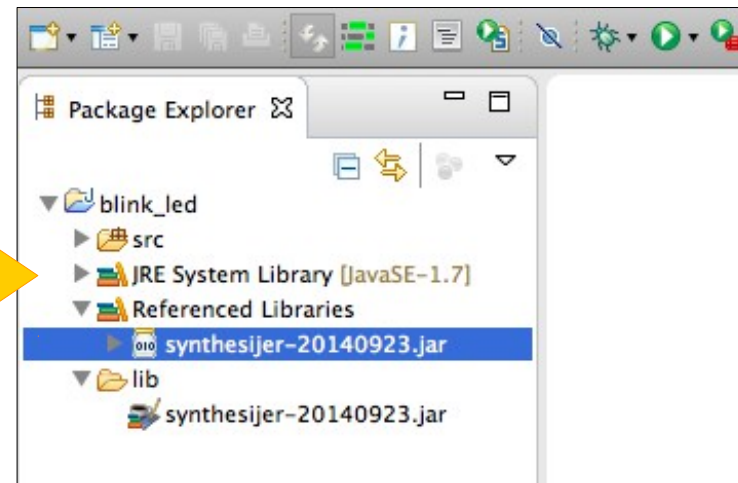
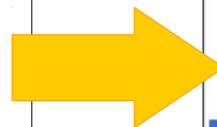
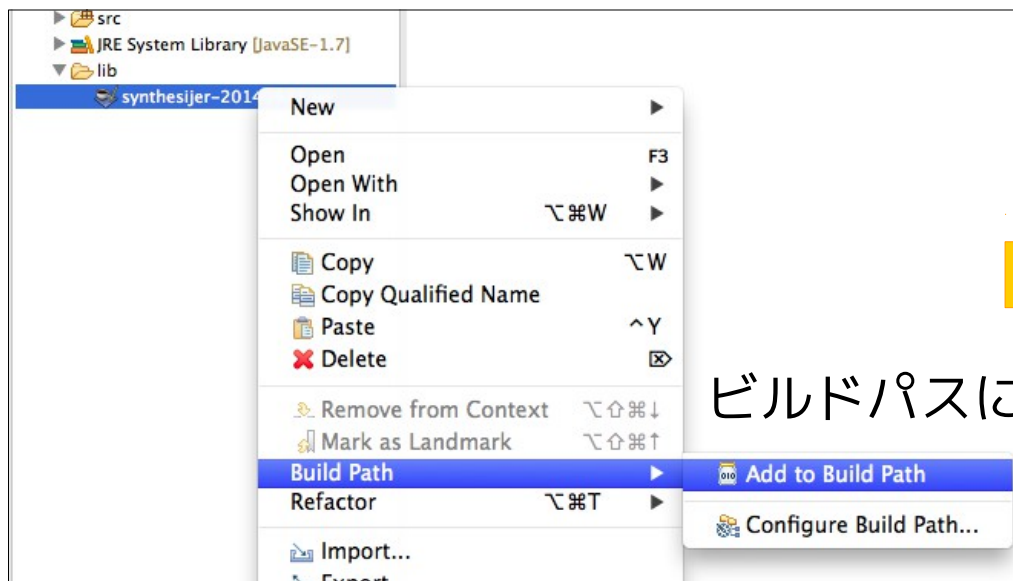
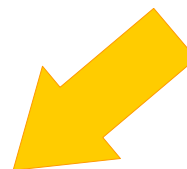
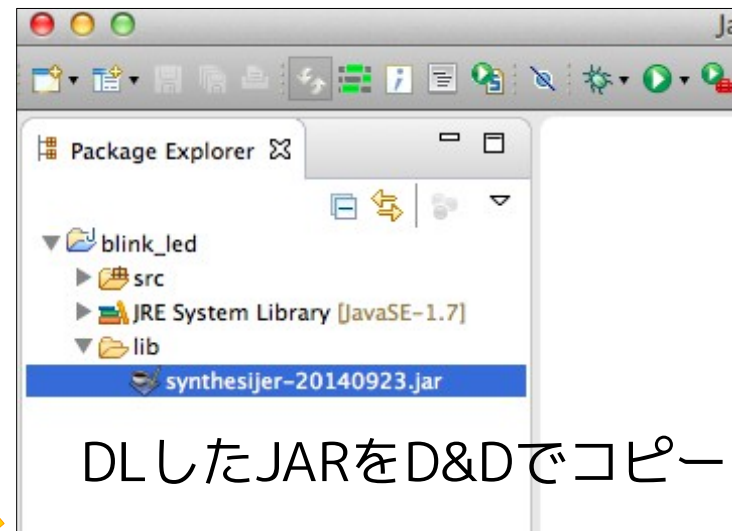
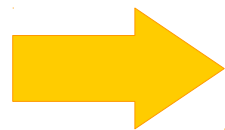
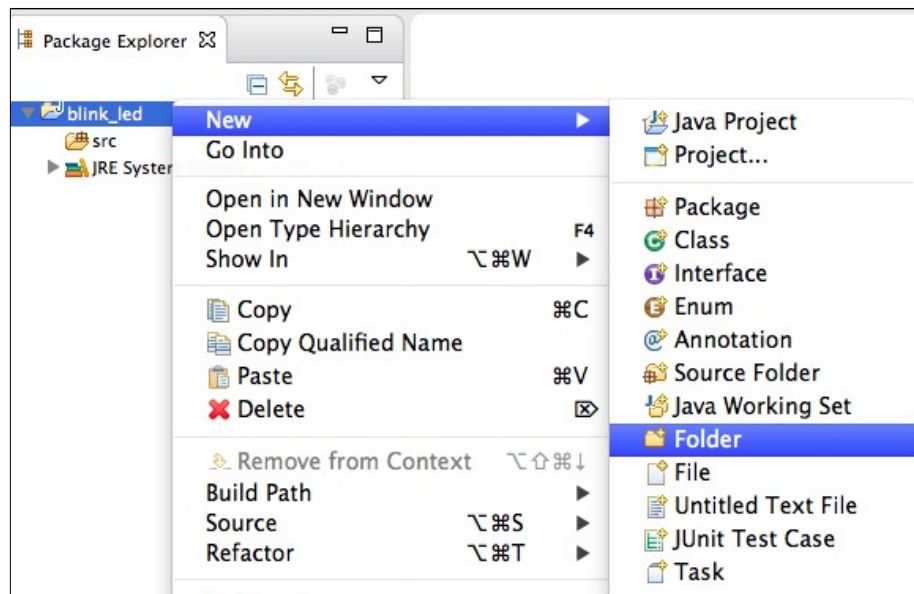


The image shows two overlapping screenshots from the Eclipse IDE. The left screenshot displays the 'New' menu, which is open, showing options like 'Java Project', 'Project...', 'Package', 'Class', 'Interface', 'Enum', 'Annotation', 'Source Folder', 'Java Working Set', 'Folder', 'File', 'Untitled Text File', 'JUnit Test Case', 'Task', 'Example...', and 'Other...'. The right screenshot shows the 'New Java Project' dialog box. The 'Project name' field is filled with 'blink_led'. The 'Use default location' checkbox is checked. The 'Location' field shows the path '/Users/miyo/workspace.synthesijer/blink_led'. Under the 'JRE' section, 'Use an execution environment JRE:' is selected, with 'JavaSE-1.7' chosen from the dropdown. Under 'Project layout', 'Create separate folders for sources and class files' is selected. Under 'Working sets', 'Add project to working sets' is unchecked. At the bottom, there is a message: 'The default compiler compliance level for the current workspace is 1.8. The new project will use a project specific compiler compliance level of 1.7.' The 'Finish' button is highlighted in blue.

(2) Eclipsedでプロジェクトを作ります

クイックスタート 3/8

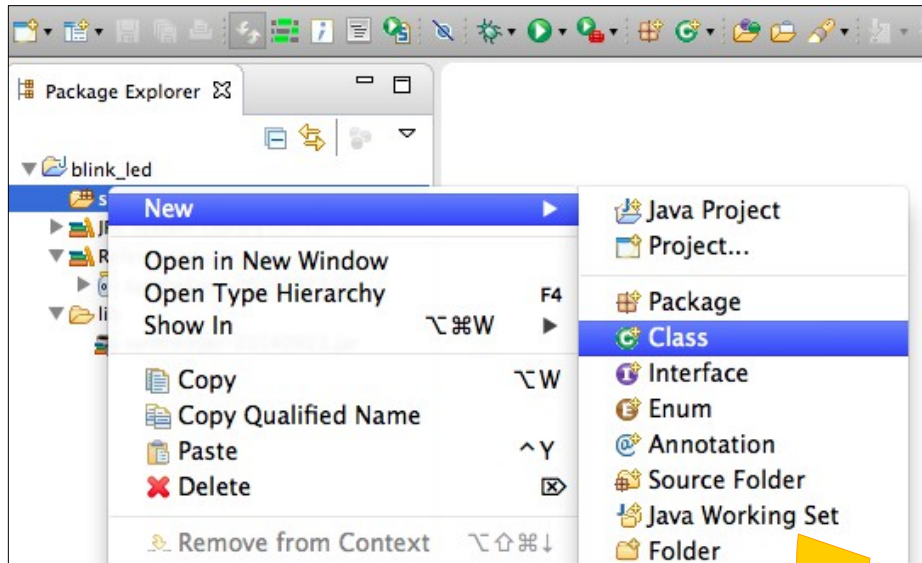
(3) libフォルダを作ってDLしたJARをコピー，ビルドパスに追加する



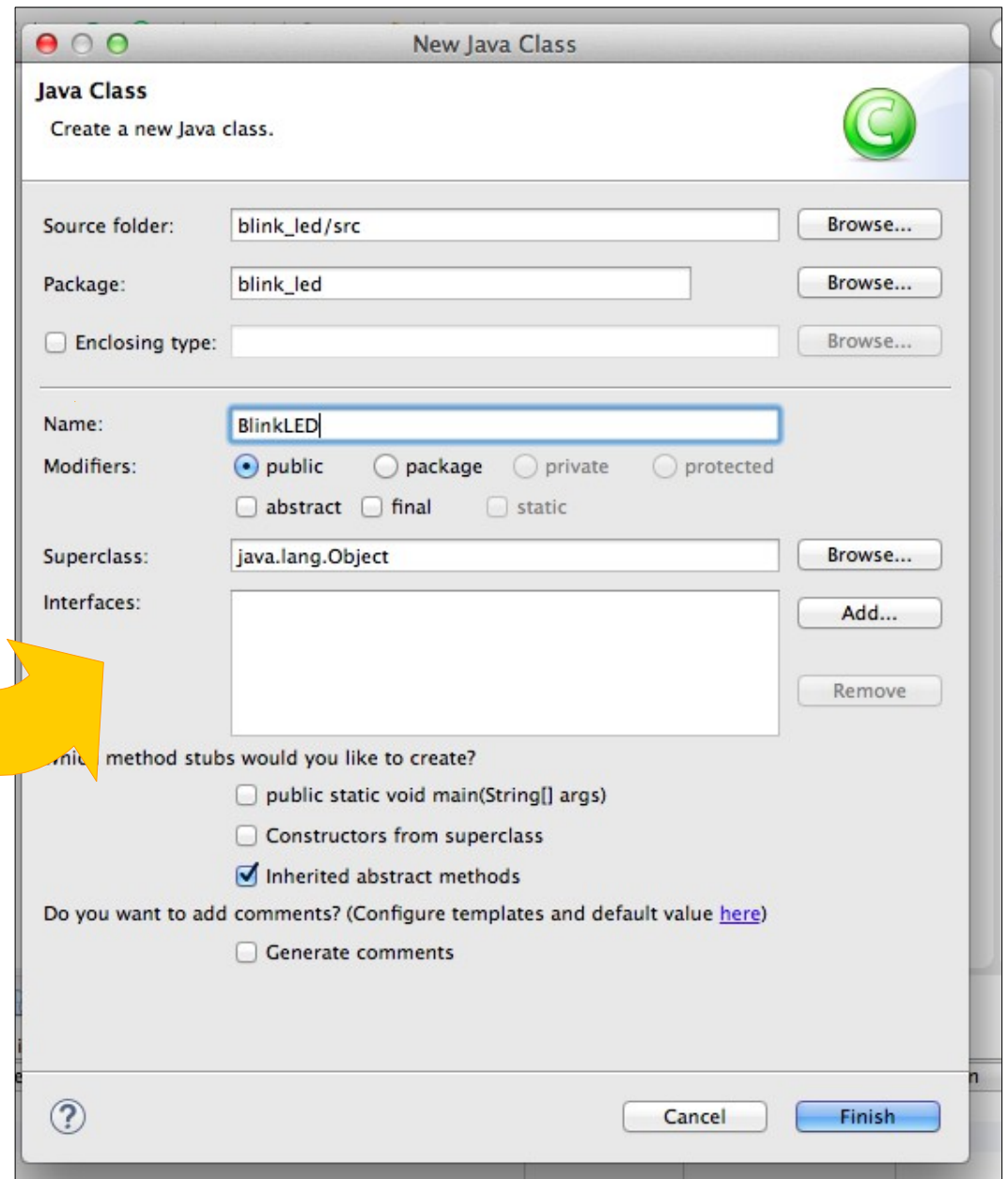
準備完了!!

クイックスタート 4/8

(4) Javaのクラスを作る

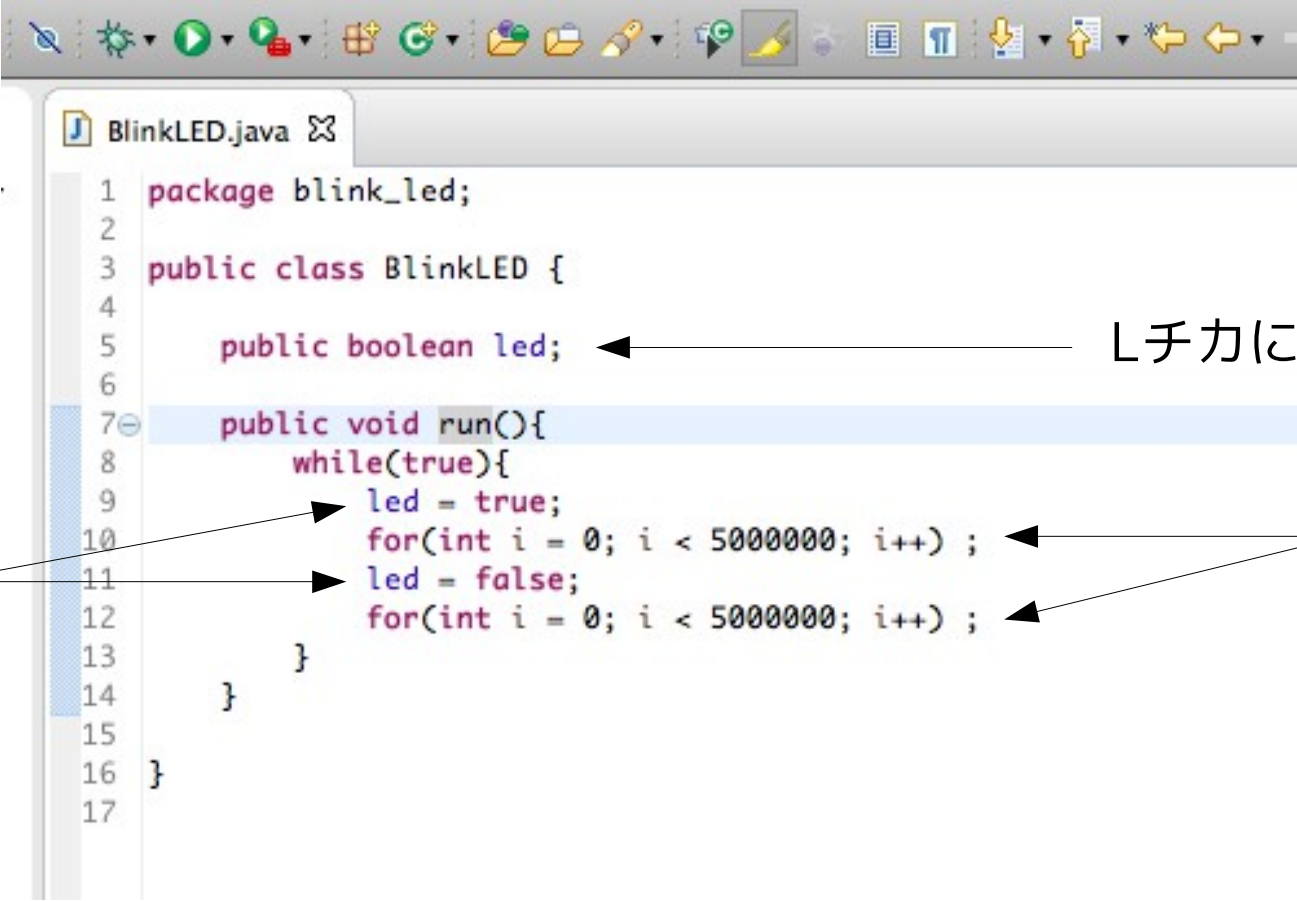


クラス生成ダイアログ



クイックスタート 5/8

(5) 間隔をおいて変数ledをtrue/falseするプログラムを書く



The screenshot shows a code editor window titled "BlinkLED.java". The code is as follows:

```
1 package blink_led;
2
3 public class BlinkLED {
4
5     public boolean led;
6
7     public void run(){
8         while(true){
9             led = true;
10            for(int i = 0; i < 5000000; i++) ;
11            led = false;
12            for(int i = 0; i < 5000000; i++) ;
13        }
14    }
15
16 }
17
```

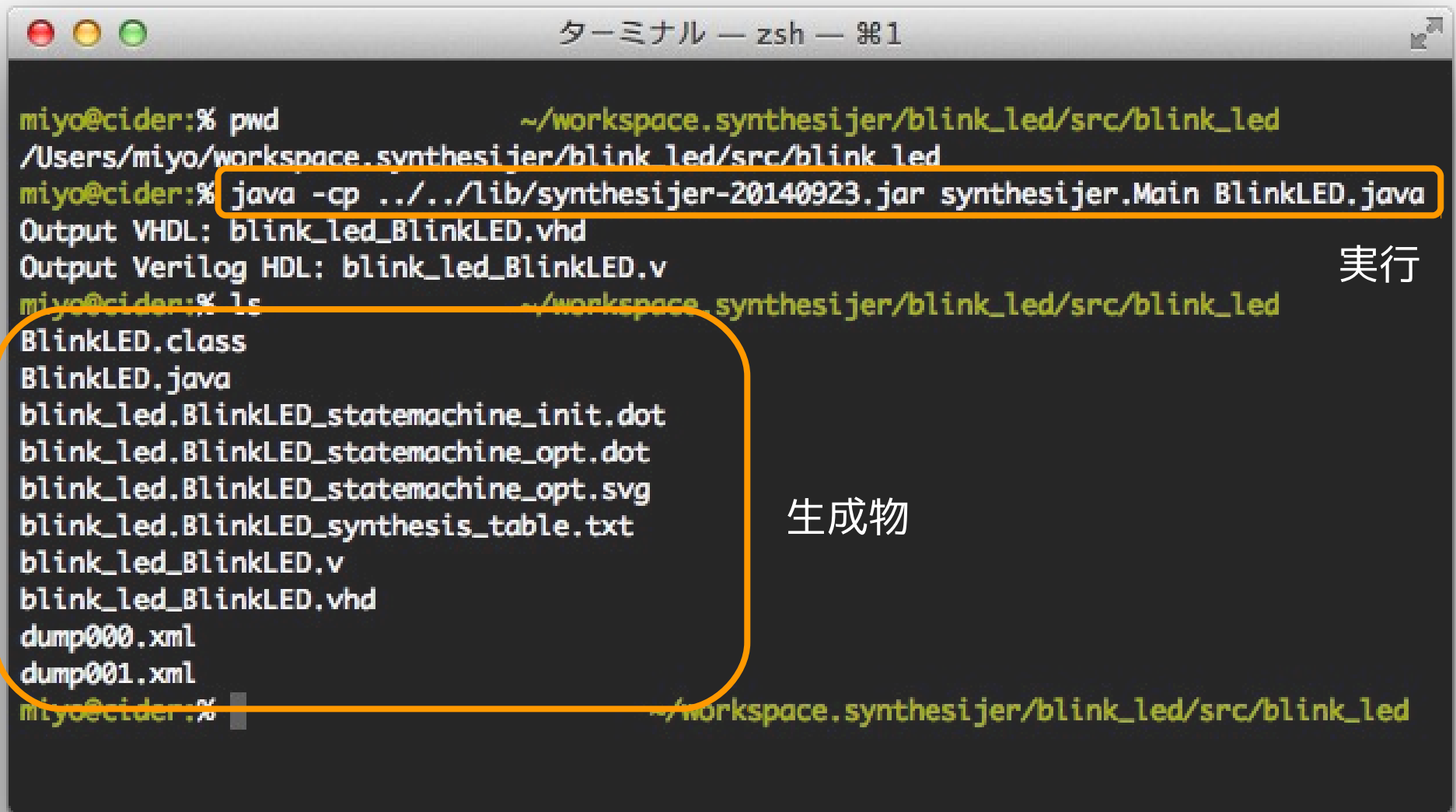
Annotations with arrows point to specific parts of the code:

- An arrow points from the text "Lチカに相当する変数" to the line `public boolean led;`.
- An arrow points from the text "点滅" to the line `led = true;`.
- An arrow points from the text "点滅" to the line `led = false;`.
- An arrow points from the text "適当なウェイト" to the first `for` loop.
- An arrow points from the text "適当なウェイト" to the second `for` loop.

自動コンパイルが裏で動くので、Javaコードとしての正しさは即座にチェックされる

クイックスタート 6/8

(6) Synthesijerを使ってJavaコードをHDLに変換



```
ターミナル — zsh — 881

miyo@cider:% pwd
~/workspace.synthesijer/blink_led/src/blink_led
/Users/miyo/workspace.synthesijer/blink_led/src/blink_led
miyo@cider:% java -cp ../../lib/synthesijer-20140923.jar synthesijer.Main BlinkLED.java
Output VHDL: blink_led_BlinkLED.vhd
Output Verilog HDL: blink_led_BlinkLED.v
miyo@cider:% ls
BlinkLED.class
BlinkLED.java
blink_led.BlinkLED_statemachine_init.dot
blink_led.BlinkLED_statemachine_opt.dot
blink_led.BlinkLED_statemachine_opt.svg
blink_led.BlinkLED_synthesis_table.txt
blink_led_BlinkLED.v
blink_led_BlinkLED.vhd
dump000.xml
dump001.xml
miyo@cider:%
```

実行

生成物

クイックスタート 7/8

(7) ISEで合成, 配置配線

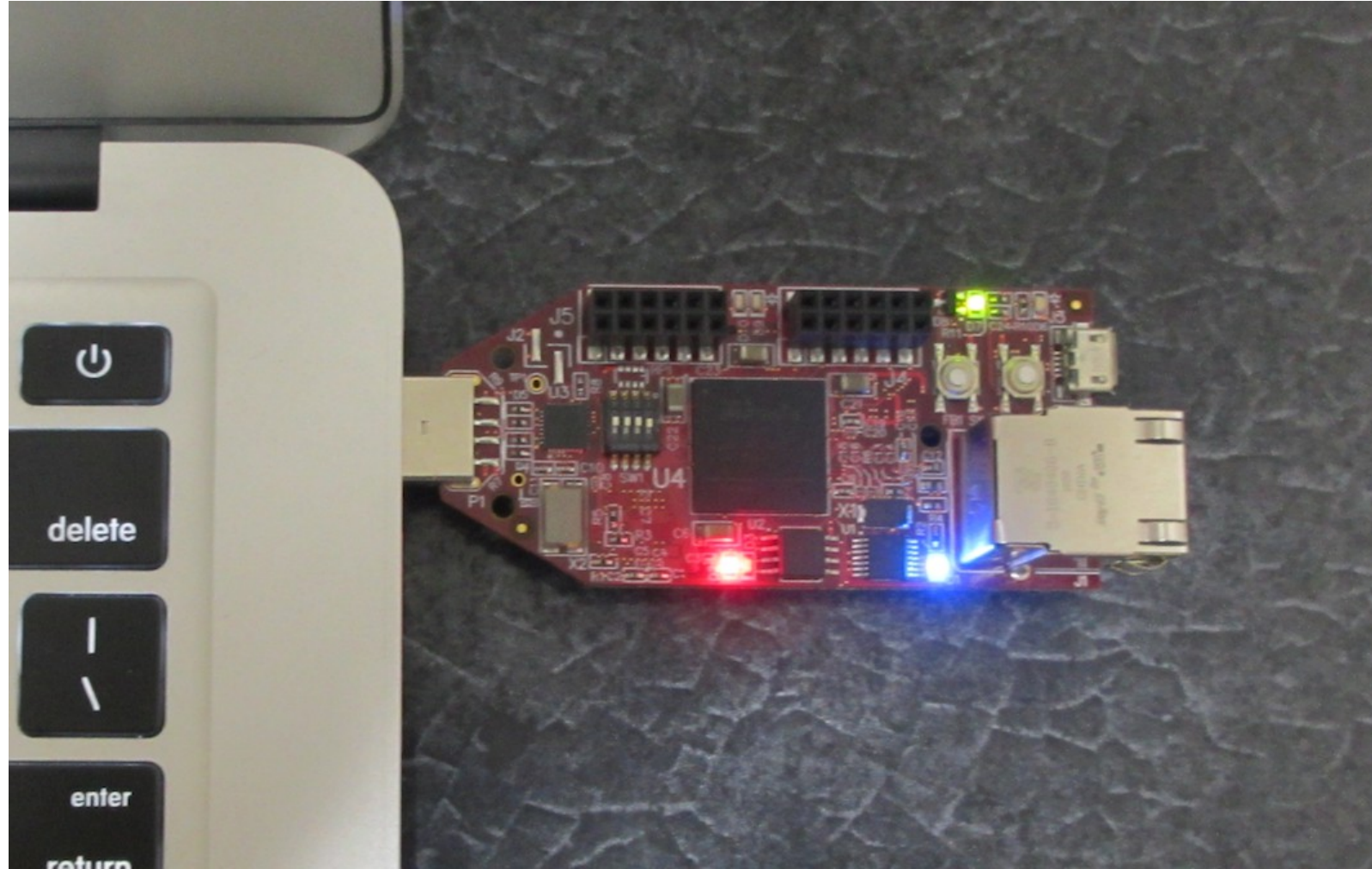
トップのHDLとUCFは別途用意

生成したモジュールのインスタンスを生成

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity blink_led_BlinkLED_top is
5   port (
6     clk, reset : in std_logic;
7     led : out std_logic );
8 end blink_led_BlinkLED_top;
9
10 architecture RTL of blink_led_BlinkLED_top is
11   component blink_led_BlinkLED
12     port (
13       clk, reset : in std_logic;
14       field_led_output : out std_logic;
15       field_led_input : in std_logic;
16       field_led_input_we : in std_logic;
17       run_req : in std_logic;
18       run_busy : out std_logic );
19   end component blink_led_BlinkLED;
20 begin
21   U: blink_led_BlinkLED port map(
22     clk => clk, reset => reset,
23     field_led_output => led,
24     field_led_input => '0', field_led_input_we => '0',
25     run_req => '1',
26     run_busy => open );
27 end RTL;
```

クイックスタート 8/8

(8) FPGAをコンフィギュレーションして動作を確認



Synthesijer でのHDL生成: 例題

```
package blink_led;

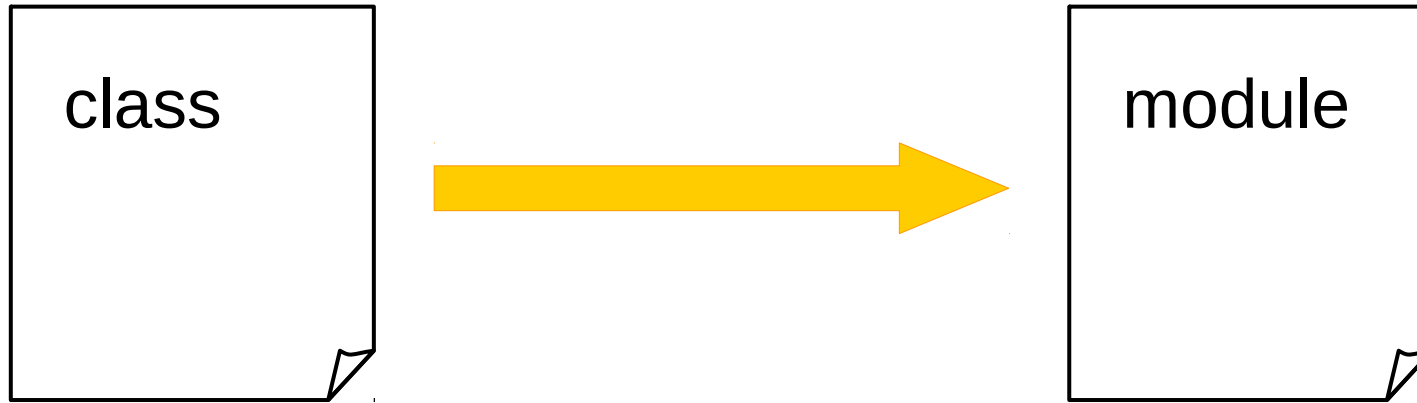
public class BlinkLED {

    public boolean led;

    public void run(){
        while(true){
            led = true;
            for(int i = 0; i < 5000000; i++) ;
            led = false;
            for(int i = 0; i < 5000000; i++) ;
        }
    }
}
```

Synthesizer でのHDL生成: 大枠

クラス毎にHWモジュールを生成



```
class BlinkLED {  
    public boolean led;  
    public void run(){  
        while(true){  
            ...  
        }  
    }  
}
```

```
entity blink_led_BlinkLED is  
    port (  
        clk : in std_logic;  
        reset : in std_logic;  
        field_led_output : out std_logic;  
        field_led_input : in std_logic;  
        field_led_input_we : in std_logic;  
        run_req : in std_logic;  
        run_busy : out std_logic  
    );  
end blink_led_BlinkLED;
```

補足

- ✓ 配列(= ブロックRAM)やVHDL/Verilogライブラリ:

`synthesijer_lib_yyyymmdd.zip`

- ✓ サンプル:

`synthesijer_samples_yyyymmdd.zip`

補足(2)

```
miyo@cider:% export SYNTHESIJER=/SDCARD/Downloads/synthesijer-20150303.jar
miyo@cider:% java -cp $SYNTHESIJER synthesijer.Main
```

Synthesijer: 2.0.1

Usage: java [-cp classpath] synthesijer.Main [options] sources

Options:

- h, --help: print this help

- vhdl: output VHDL

- verilog: output Verilog HDL

(If you specify neither --vhdl nor --verilog, --vhdl is selected.)

- config=file: the specified file is used for compiler settings

- no-optimize: do not apply any optimizations

- chaining: do operation chain in greedy manner

- operation-strength-reduction: do operation strength reduction

- ip-exact=TOP: generates a IP package template for "TOP" module

- vendor=name: to specify vendor id for generating a IP package

- libname=name: to specify library name for generating a IP package

Please access to <http://synthesijer.sorceforge.net/> for more information.

```
miyo@cider:%
```

今日の内容

- ✓ サンプル
- ✓ Synthesijer活用の3パターン
- ✓ Synthesijer.Scala

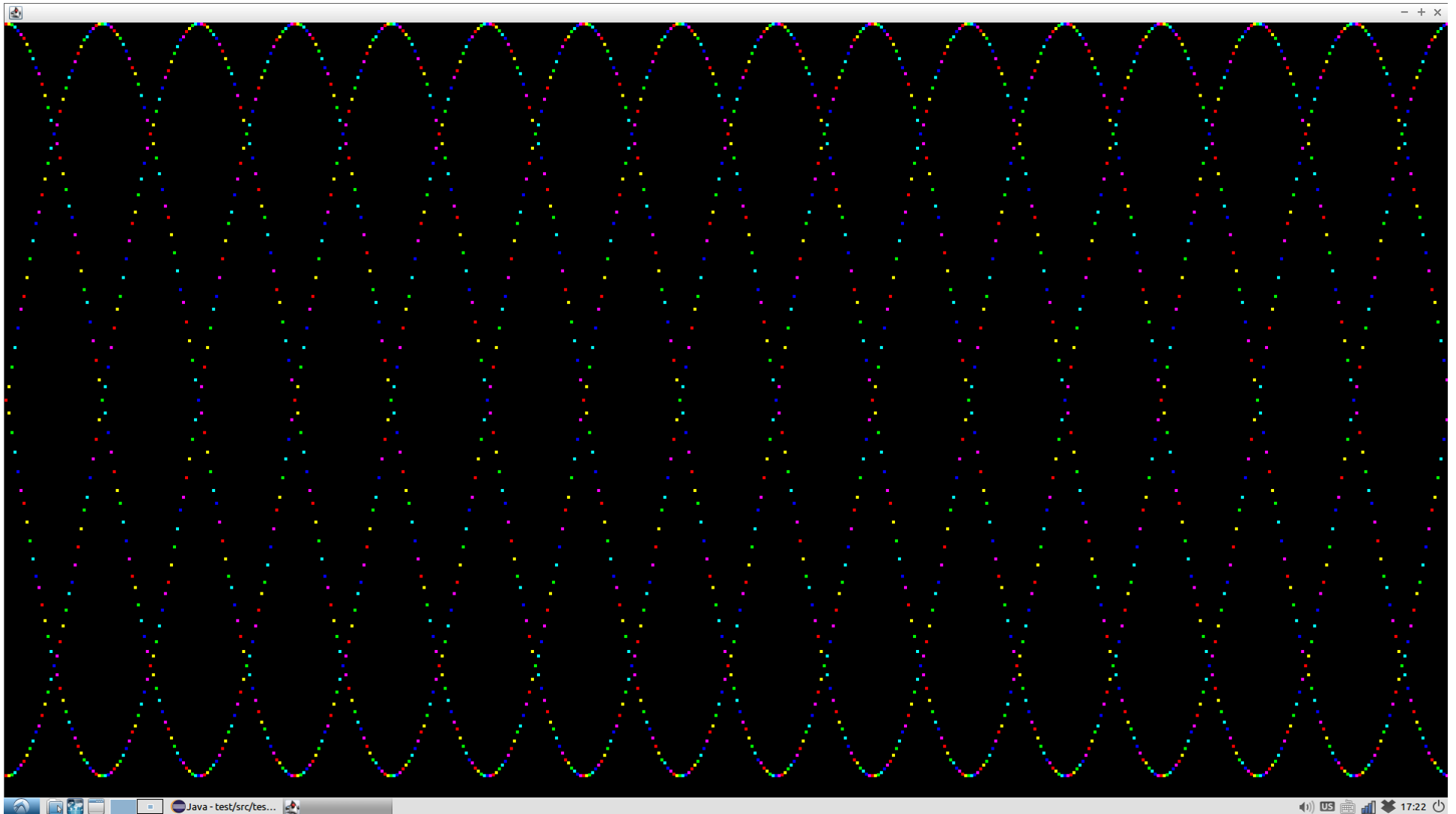
Synthesijer サンプル

サンプル

- ✓ グラフィクス表示
- ✓ BrainF**k インタプリタ

サンプル1: グラフィクス表示

- ✓ 90度位相ずれしたサインカーブをSWで描画



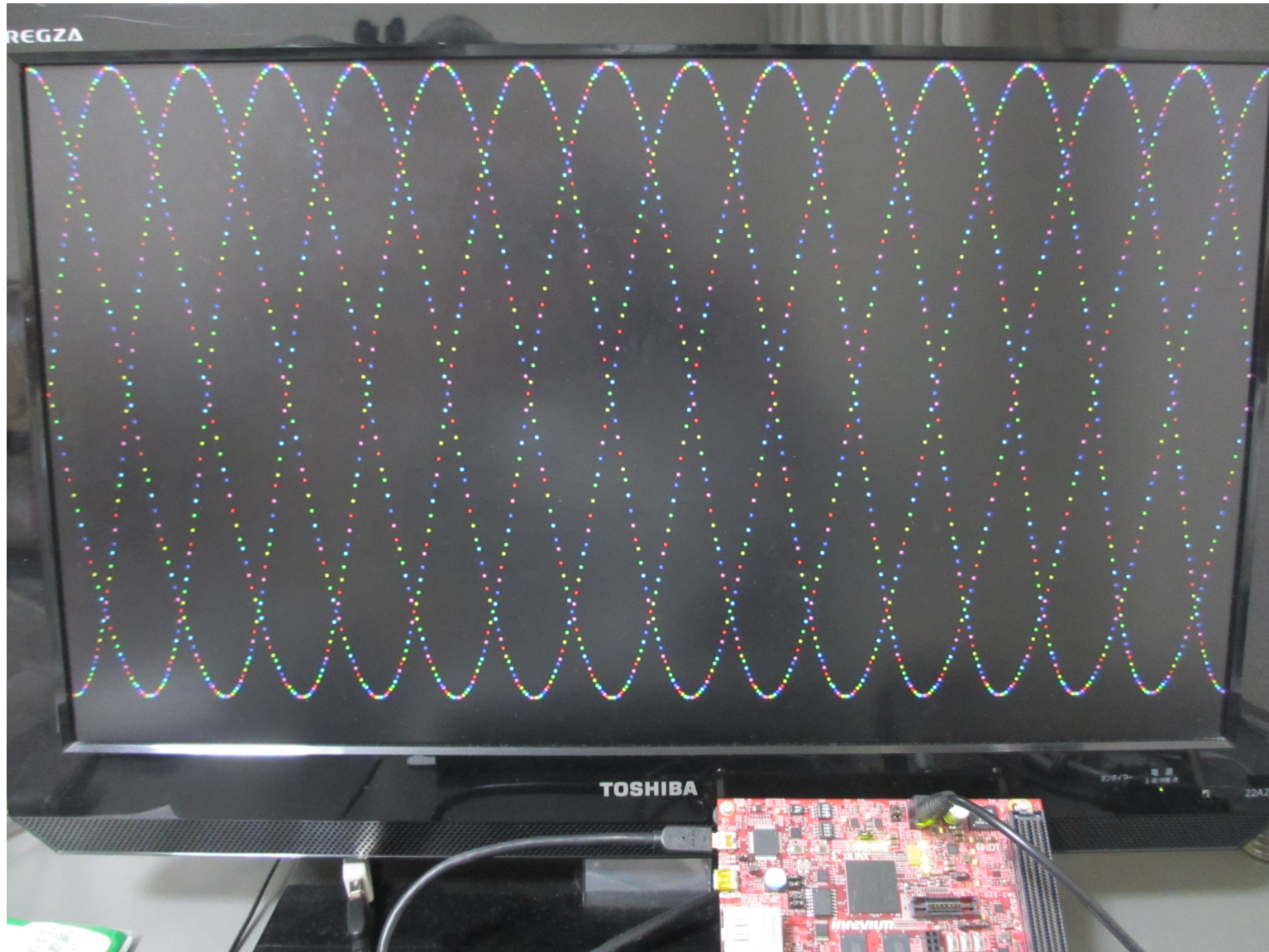
サインカーブ描画のJavaコード(抜粋)

```
public class RGBTest{
    ...
    private final TestFrame obj = new TestFrame();
    private final SinTableRom sin = new SinTableRom();
    private final int colortbl[] = new int[6];

    public void paint_sincurve(int offset){
        int c_id = 0;
        for(int i = 0; i < (1920 >> 2); i++){
            int x = i << 2;
            int y = sin.sintable[(i+offset)&0x7F];
            int c = colortbl[c_id];
            obj.fill_rect(x, y, 4, 4, c);
            c_id = c_id + 1;
            if(c_id == 6) c_id = 0;
        }
    }
    public void run(){
        init_colortbl();
        while(true){
            paint_sincurve(0);
            paint_sincurve(32);
            paint_sincurve(64);
            paint_sincurve(96);
            obj.flush();
        }
    }
}
```

FPGAでの実行例

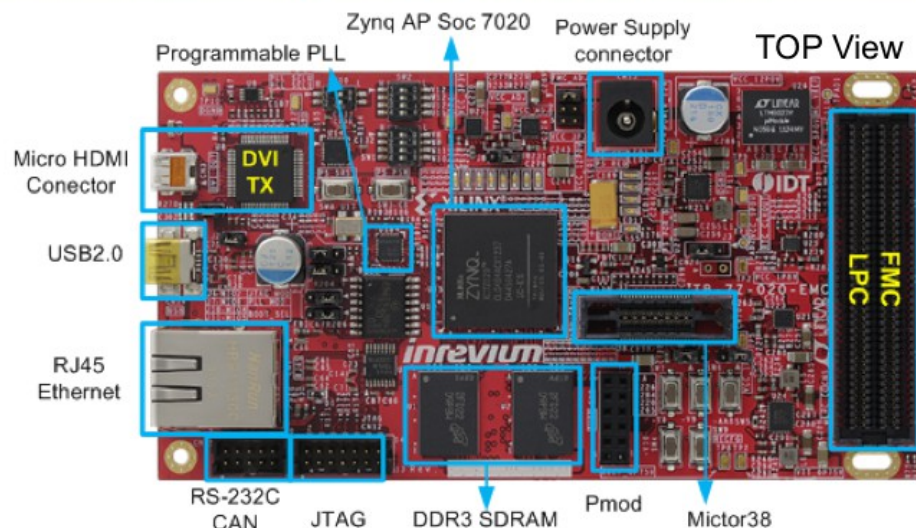
- ✓ 90度位相ずれしたサインカーブを実機で確認



使用したボード

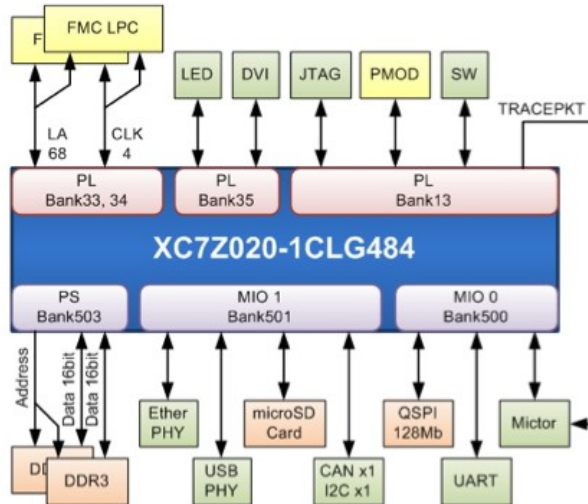
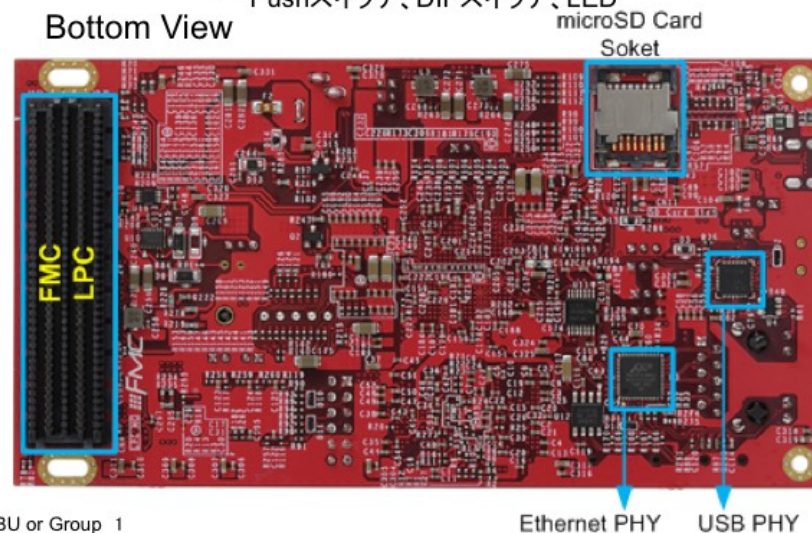
TB-7Z-020-EMC

inrevium



- ・搭載デバイス: XC7Z020-CLG484
- ・搭載メモリ:
 - DDR3 SDRAM x 2(16bit data bus x 2)
 - MicroSDカードソケットとメディア x 1
 - 128Mbit QSPI フラッシュメモリ x 1
- ・拡張インターフェース
 - FMC(LPC) x 2
 - PMOD(Digilent module interface) x 1
- ・汎用インターフェース
 - Gigabit Ethernet x 1
 - USB2.0 x 1(Host Device)
 - UART(RS-232C) x 1
 - CAN x 1
 - DVI TX x 1
- ・デバックインターフェース
 - MICTOR-38 for ARM
 - 10 pin header for Xilinx Cable
 - Pushスイッチ、DIPスイッチ、LED

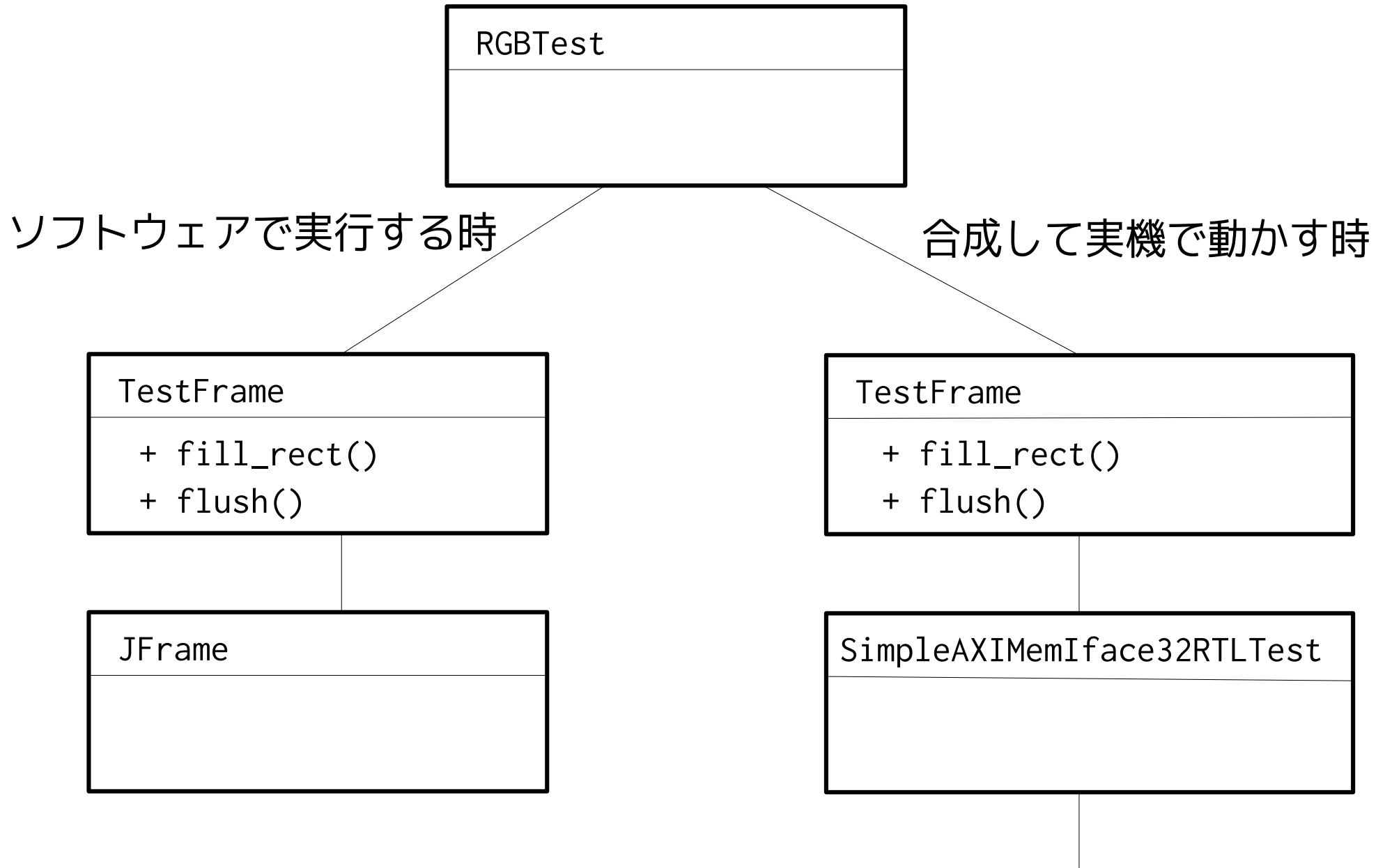
Bottom View



TOKYO ELECTRON DEVICE LIMITED

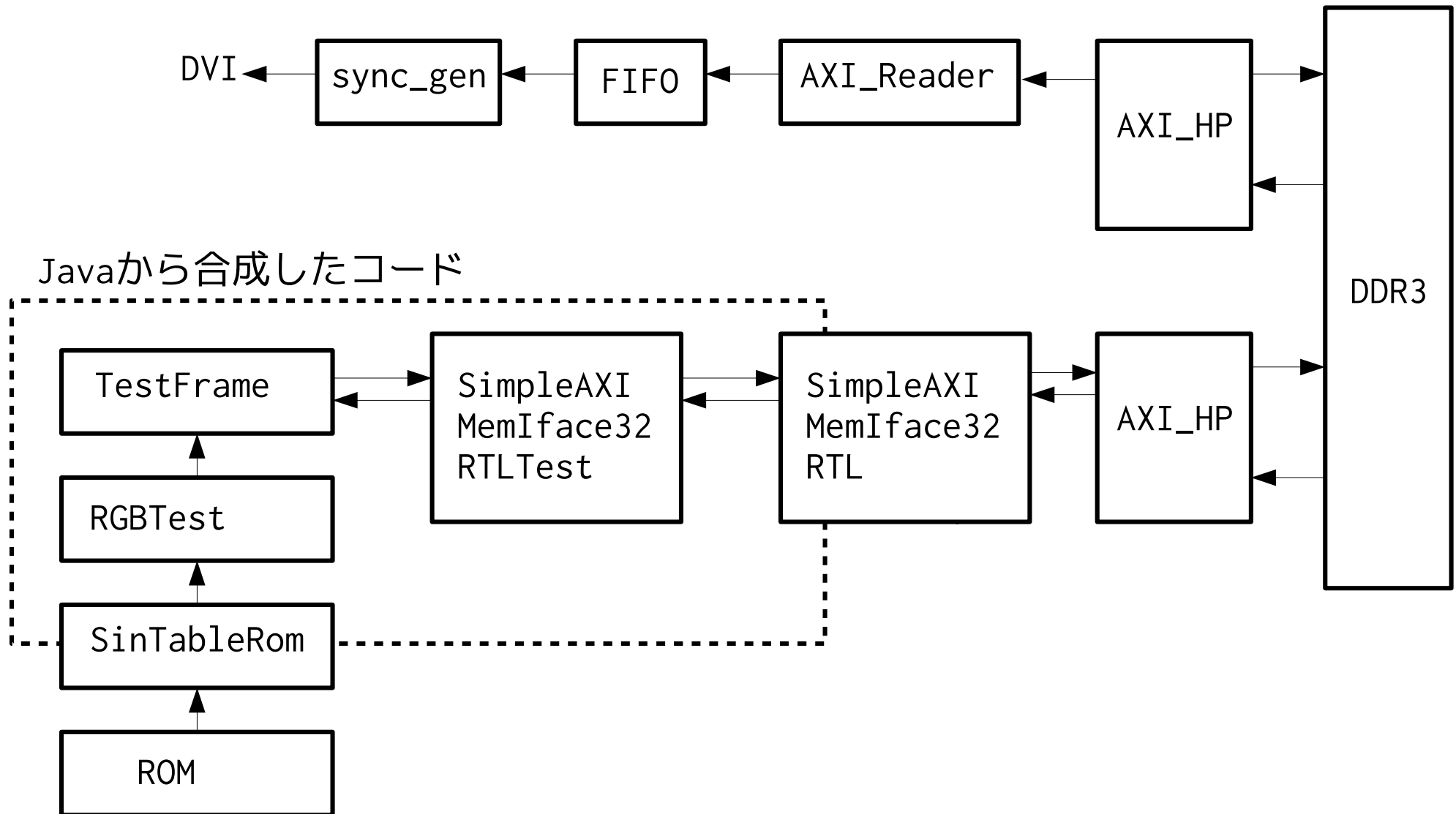
Author / BU or Group 1

SWとHWで同じコードを利用



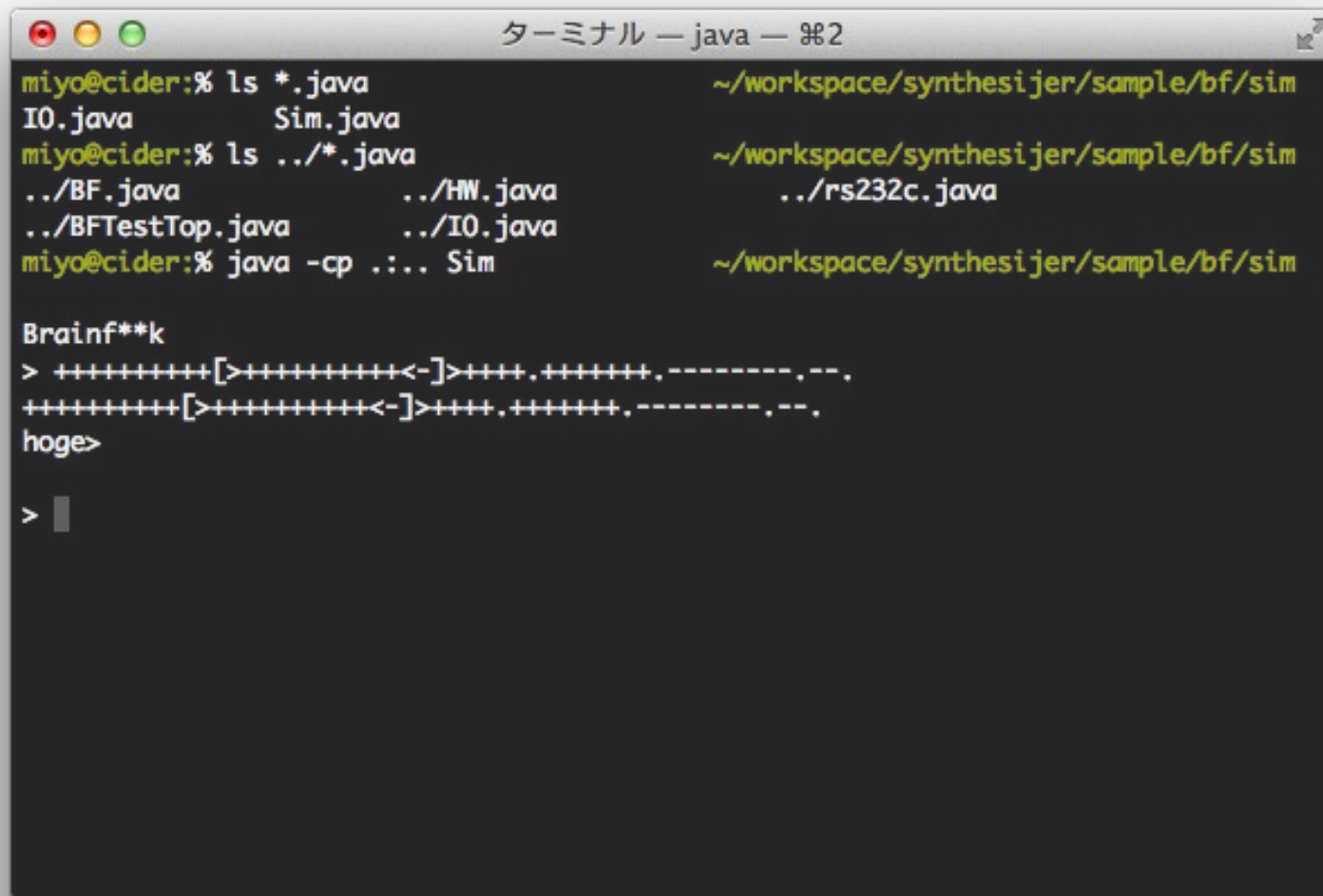
システム全体のブロックダイアグラム

- ✓ 水面下はちょっと複雑



サンプル2: BrainF**k

- ✓ 標準入出力越しでBrainF**kインタプリタを実行
 - ✓ +, -, >, <, [,], ,, の記号からなるインタプリタ



```
ターミナル — java — ㊿2
miyo@cider:% ls *.java                               ~/workspace/synthesijer/sample/bf/sim
IO.java          Sim.java
miyo@cider:% ls ../*.java                             ~/workspace/synthesijer/sample/bf/sim
../BF.java       ../HW.java                          ../rs232c.java
../BFTestTop.java ../IO.java
miyo@cider:% java -cp :... Sim                       ~/workspace/synthesijer/sample/bf/sim

Brainf**k
> ++++++[>++++++<-]>++++.+++++.-----.--.
++++++[>++++++<-]>++++.+++++.-----.--.
hoge>

> █
```

サンプル2: BrainF**k

```
private IO io = new IO();
private byte[] prog = new byte[CODESIZE];
private byte[] data = new byte[ARRAYSIZE];
private int ptr, pc;

public boolean step() {
    byte cmd = prog[pc];
    byte tmp;
    int nlvl = 0;
    switch (cmd) {
        case 0: return false;
        case '>': ptr++; break;
        case '<': ptr--; break;
        case '+': data[ptr] = (byte) (data[ptr] + 1); break;
        case '-': data[ptr] = (byte) (data[ptr] - 1); break;
        case '.': io.putchar(data[ptr]); break;
        case ',': data[ptr] = io.getchar(); break;
        case '[':
            if (data[ptr] == (byte) 0) {
                while (true) {
                    pc++;
                    if (prog[pc] == '[' && nlvl == 0) break;
                    if (prog[pc] == '[') nlvl++;
                    if (prog[pc] == ']') nlvl--;
                }
            }
            break;
        case ']':
            ...
    }
}
```

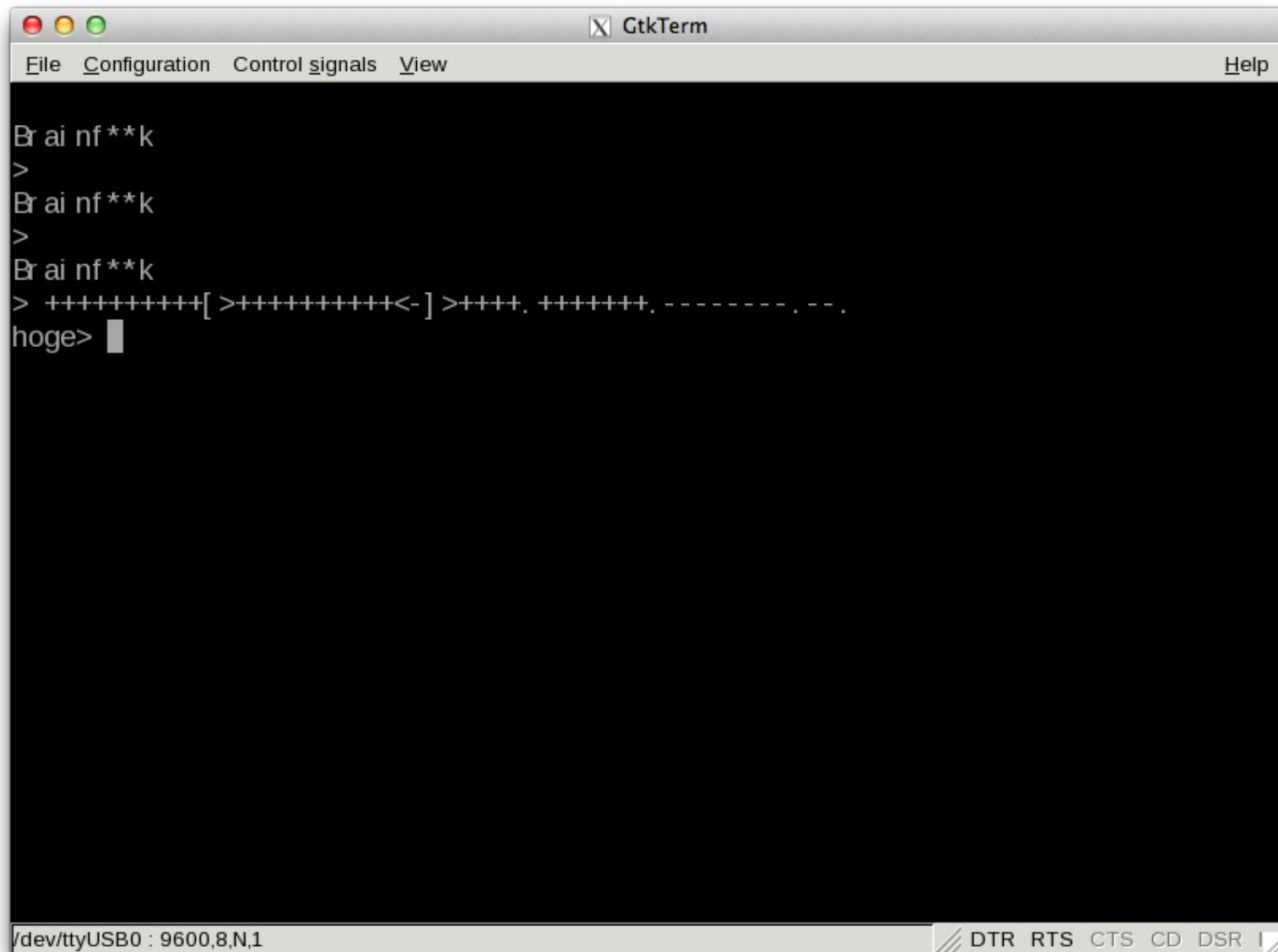
サンプル2: BrainF**k

```
public void read() {  
    prompt();  
    for (int i = 0; i < CODESIZE; i++) {  
        byte b;  
        b = io.getchar();  
        //io.putchar(b);  
        if (b == '\n' || b == '\r') {  
            prog[i] = (byte) 0;  
            break;  
        } else {  
            prog[i] = b;  
        }  
    }  
}
```

```
public void print() {  
    boolean flag = true;  
    for (int i = 0; i < CODESIZE; i++) {  
        byte b = prog[i];  
        if (b == 0) {  
            break;  
        }  
        io.putchar(b);  
    }  
    io.putchar((byte) '\n');  
}
```

FPGAで実行

- ✓ 標準入出力の代わりにシリアル通信越しで実行

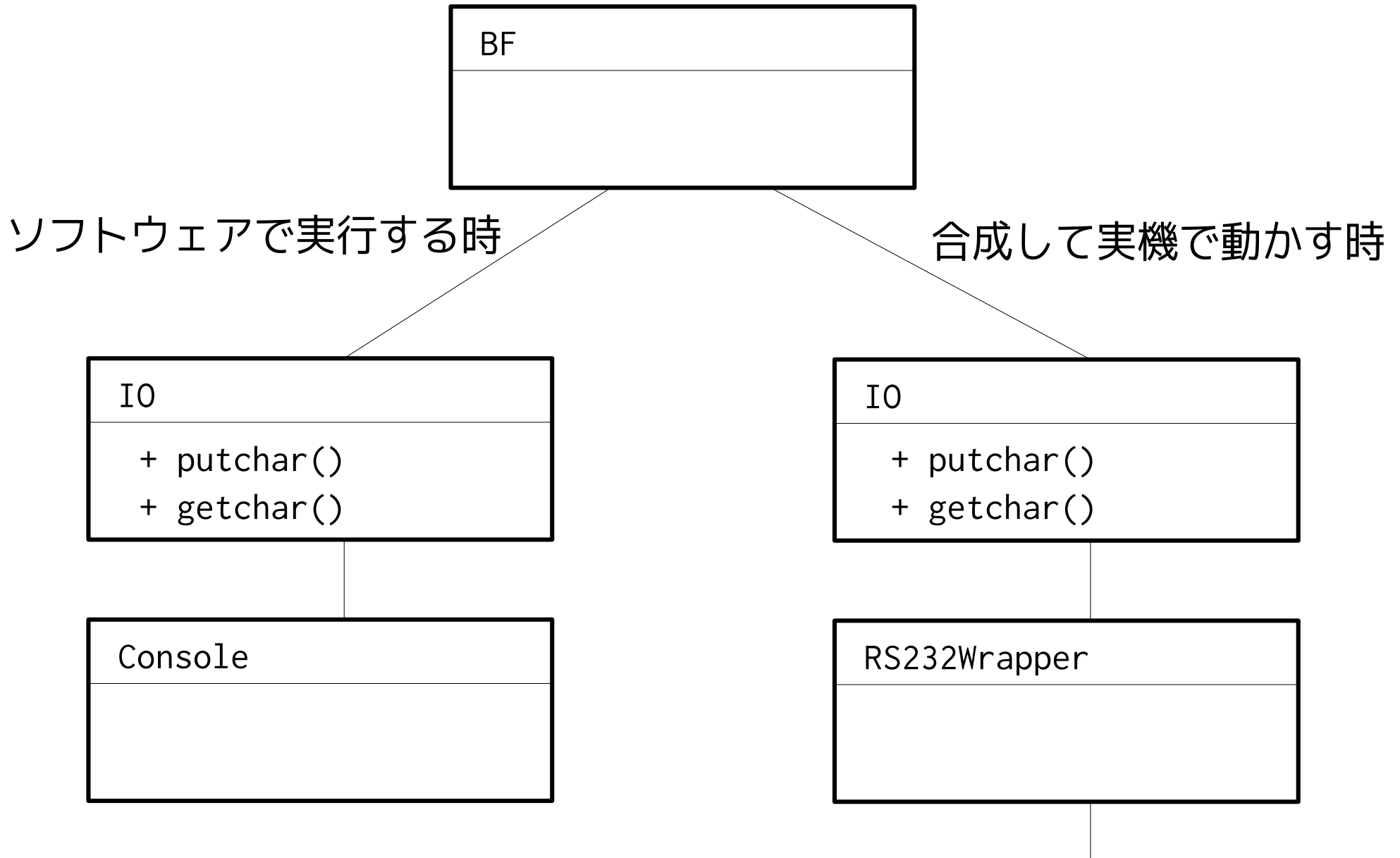


The image shows a screenshot of a GtkTerm window, which is a terminal emulator. The window has a title bar with the text "GtkTerm" and standard window control buttons (red, yellow, green). Below the title bar is a menu bar with the following items: "File", "Configuration", "Control signals", "View", and "Help". The main area of the window is black and contains white text. The text shows a series of commands and responses in a serial terminal format. The first three lines are "Br ai nf **k", ">", and "Br ai nf **k", ">". The fourth line is "Br ai nf **k", ">". The fifth line is "> ++++++[>+++++<-] >++++. +++++. -----. ---.". The sixth line is "hoge>". At the bottom of the window, there is a status bar with the text "/dev/ttyUSB0 : 9600,8,N,1" on the left and "DTR RTS CTS CD DSR I" on the right.

```
Br ai nf **k
>
Br ai nf **k
>
Br ai nf **k
> ++++++[ >+++++<- ] >++++. +++++. -----. ---.
hoge>
```

/dev/ttyUSB0 : 9600,8,N,1 DTR RTS CTS CD DSR I

SWとHWで同じコードを利用

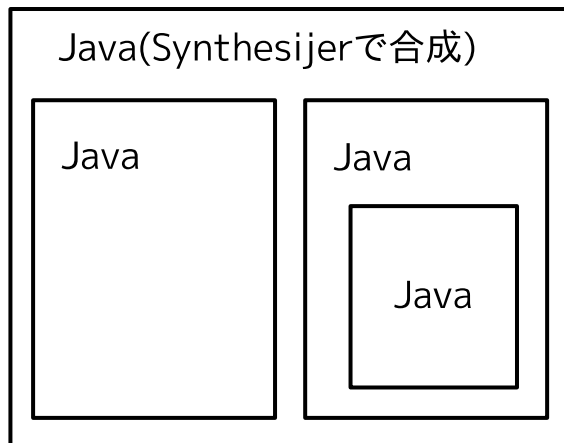


Synthesi^jer 活用方法 - 3つのパターン

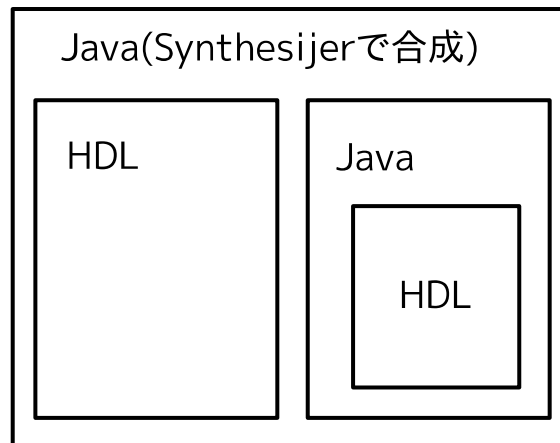
3つの開発パターン

- (1) Javaによるモジュールだけでシステムを構成
- (2) HDLによるモジュールを部品として利用．全体管理はJavaで記述
＝ 既存IPコア，FPGAの性能を活用
- (3) Javaによるモジュールを部品として利用．全体管理はHDLで記述
＝ 複雑なアルゴリズムをSWプログラマに書いてもらう．SW資産の活用

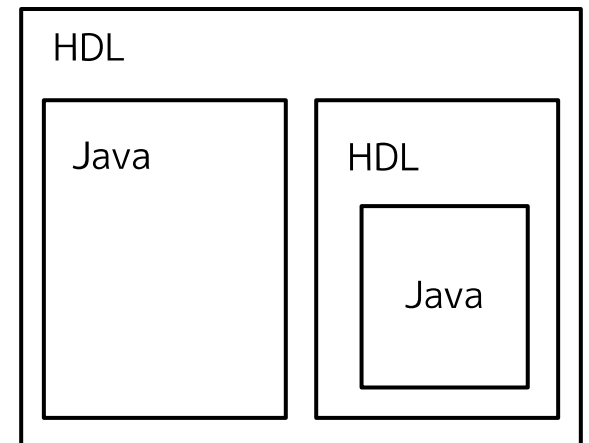
(1)Javaでシステムを構成



(2)Java + HDL記述の部品



(3)HDL + Java記述の部品

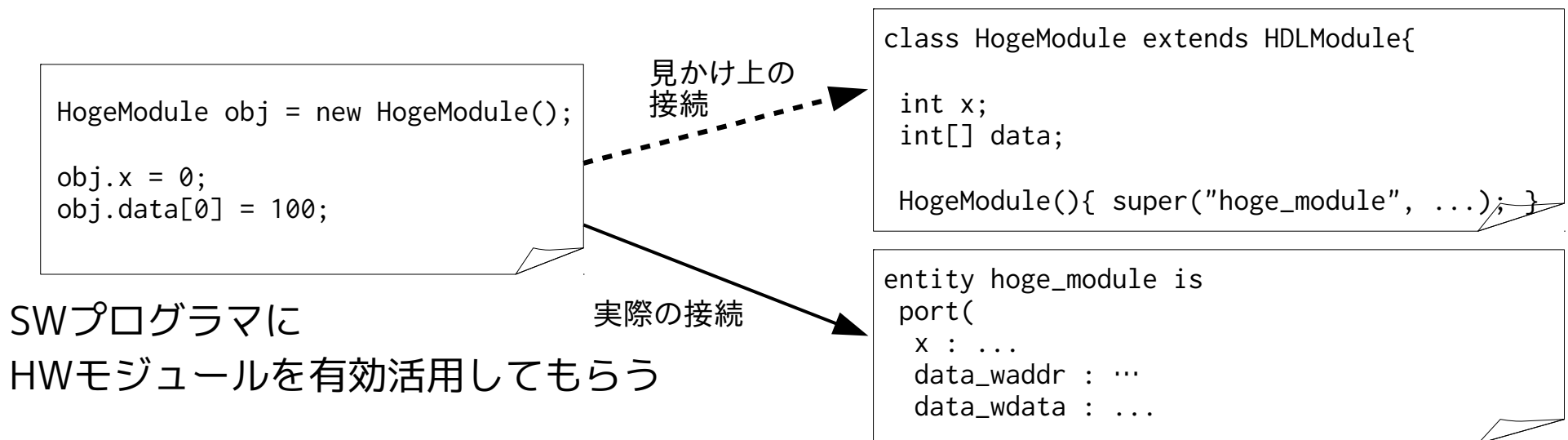


(1) Javaでシステムを構成

- ✓ すべてをJavaで記述
- ✓ トップモジュール, UCF/XDCなんかは必要
- ✓ I/Oなどは用意されたライブラリを利用
 - ✓ シリアルI/O, SC1602, 純粋なInput/Output, ...
 - ✓ AXI(32bit逐次, シンプルなキャッシュ), UPL, ...

(2) Java+HDL記述の部品

- ✓ JavaプログラムにHDL記述の部品を埋め込む
- ✓ Javaで記述したオブジェクトのようにHDL記述部品をインスタンス化して使う
- ✓ HDLモジュールを簡単に使い回せる
- ✓ Synthesizer内部でどうオブジェクトが扱われるかを知る必要がある
 - ✓ Javaとのブリッジのために HDLModule を継承したクラスを実装
 - ✓ 合成時にはHDL記述のコードと組み合わせる
- ✓ HWプログラマがサポートしつつSWプログラマにFPGAを活用してもらう!!

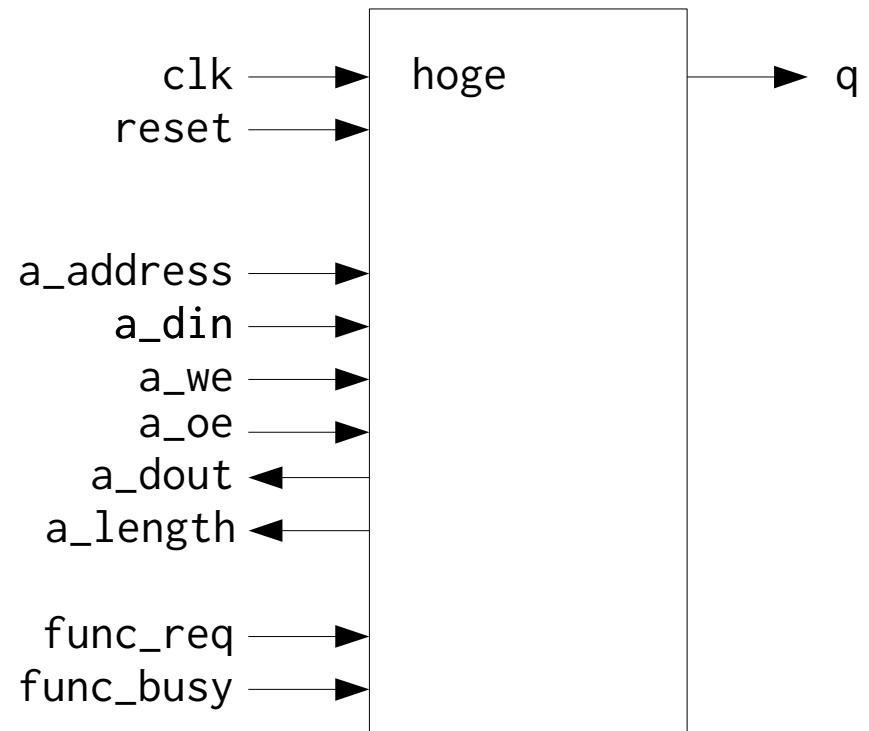


インスタンス生成・拡張モジュール

VHDL/Verilog HDLで書いたモジュールのインスタンス化

ex. Verilog HDLで書いたhogeをJavaから呼び出す

```
module hoge(  
    input wire clk,  
    input wire reset,  
  
    input signed [31:0] a_address,  
    input signed [31:0] a_din,  
    input wire we,  
    input wire oe,  
    output signed [31:0] a_dout,  
    output signed [31:0] a_length,  
  
    input wire func_req,  
    output wire func_busy,  
  
    output wire [31:0] q  
);  
  
...  
  
endmodule
```



インスタンス生成・拡張モジュール

VHDL/Verilog HDLで書いたモジュールのインスタンス化
↓のようなJava用のダミーコードを書いて、使う

```
class HogeIface extends HDLModule{
```

```
    int[] a;  
    void func(){}
```

```
    public HogeIface(String... args){
```

```
        newPort("a_address", HDLPort.DIR.IN,  genSignedType(32));  
        newPort("a_din",      HDLPort.DIR.IN,  genSignedType(32));  
        newPort("a_we",       HDLPort.DIR.IN,  genBitType());  
        newPort("a_oe",       HDLPort.DIR.IN,  genBitType());  
        newPort("a_dout",     HDLPort.DIR.OUT, genSignedType(32));  
        newPort("a_length",   HDLPort.DIR.OUT, genSignedType(32));  
        newPort("func_req",   HDLPort.DIR.IN,  genBitType());  
        newPort("func_busy",  HDLPort.DIR.OUT, genBitType());
```

```
        newPort("q", HDLPort.DIR.OUT, genVectorType(32), Enum.of(OPTION.EXPORT));
```

```
    }
```

```
    ...
```

```
}
```

```
...
```

```
    private final HogeIface obj = new  
    HogeIface();
```

```
...
```

```
    private void func(){  
        obj.a[10] = 100;  
        obj.func();
```

```
...
```

} a[]

} func()

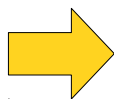
これはJavaでは使わない

(3) HDL+Java記述の部品

- ✓ HDLで設計したシステムにJava記述の部品を埋め込む
- ✓ 複雑なアルゴリズムの処理でもJavaなら(比較的)手軽に実装
- ✓ HDLで自然に実装できるストリーム処理, 並列処理を活用
- ✓ Synthesijerで合成した回路がどういう構成か知る必要がある
- ✓ SWプログラマ/資産をFPGA設計に活用!!

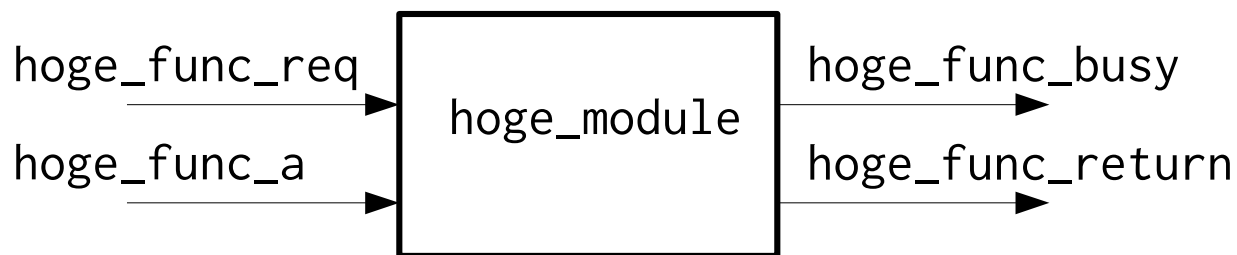


```
class HogeModule{  
    int hoge_func(int a){  
        ...  
    }  
}
```

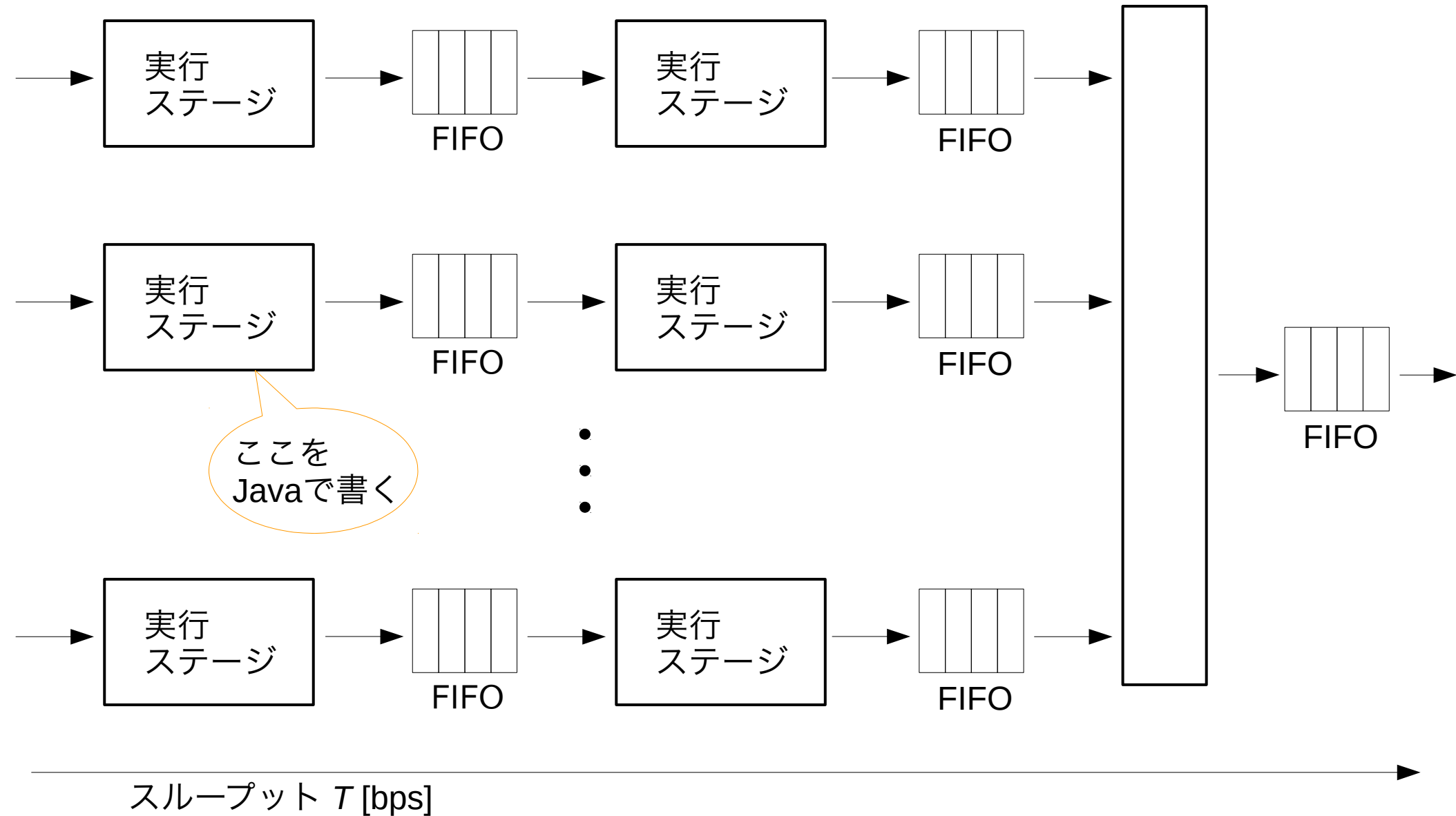


Synthesijer

```
entity hoge_module is  
    port(  
        hoge_func_a : in ...  
        hoge_func_return : out ...  
        hoge_func_req : in ...  
        hoge_func_busy : out ...  
    )  
end entity
```

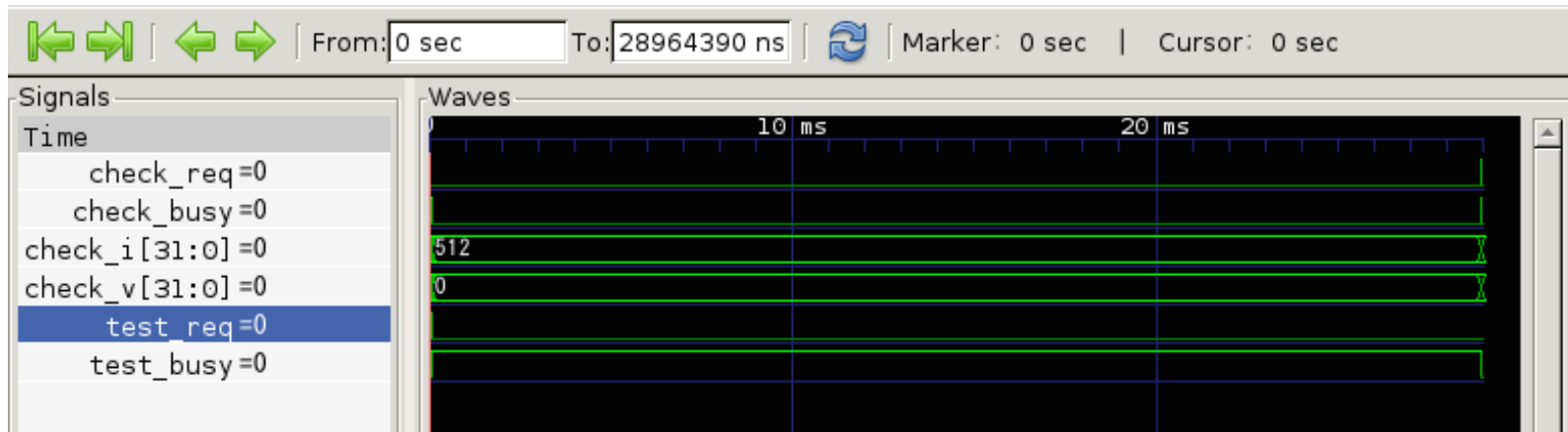


最近よく実装しているパターン

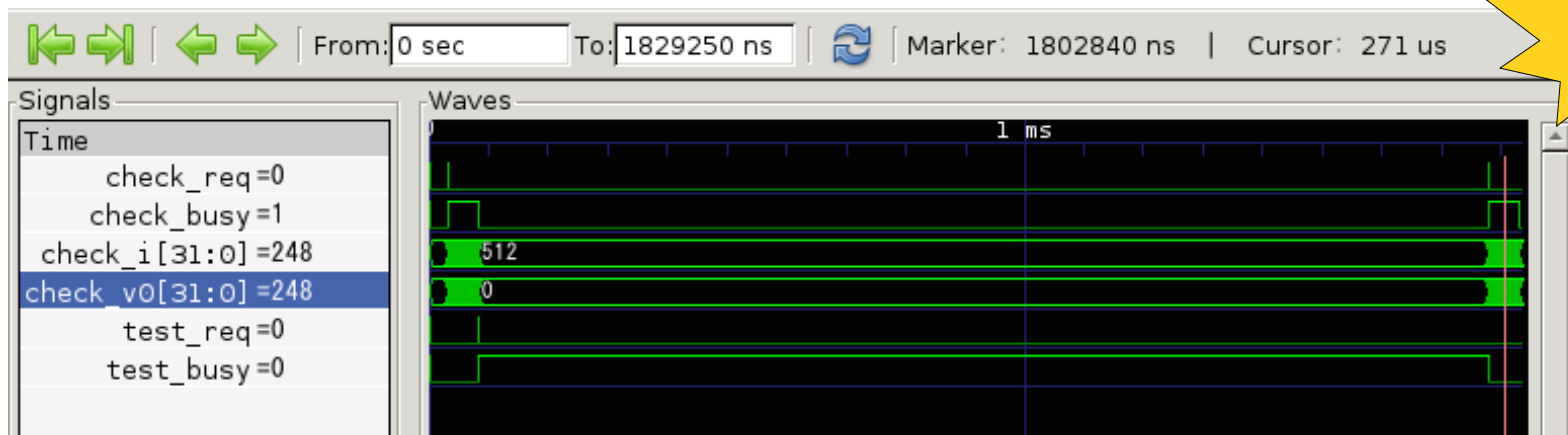


複雑なアルゴリズム処理はJavaで実装

- ✓ 例: バブルソートじゃなくてマージソートの採用, など
- ✓ バブルソート(512個の降順→昇順)



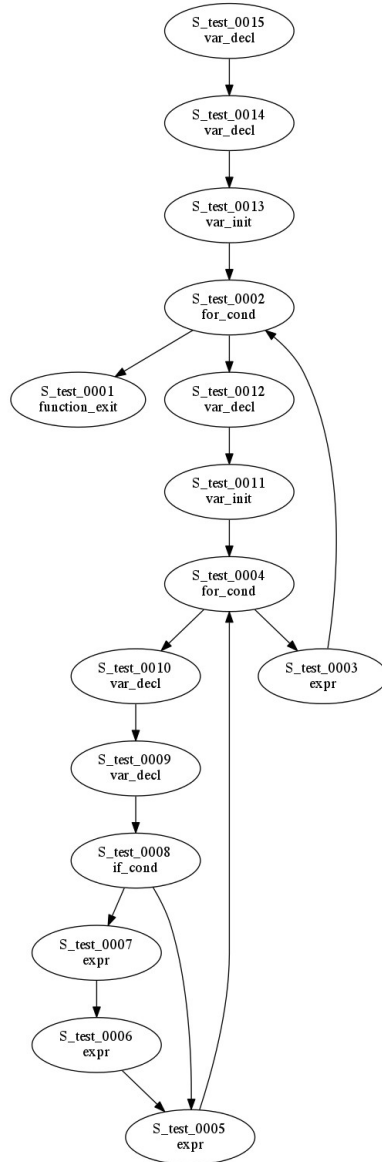
- ✓ マージソート(512個の降順→昇順)



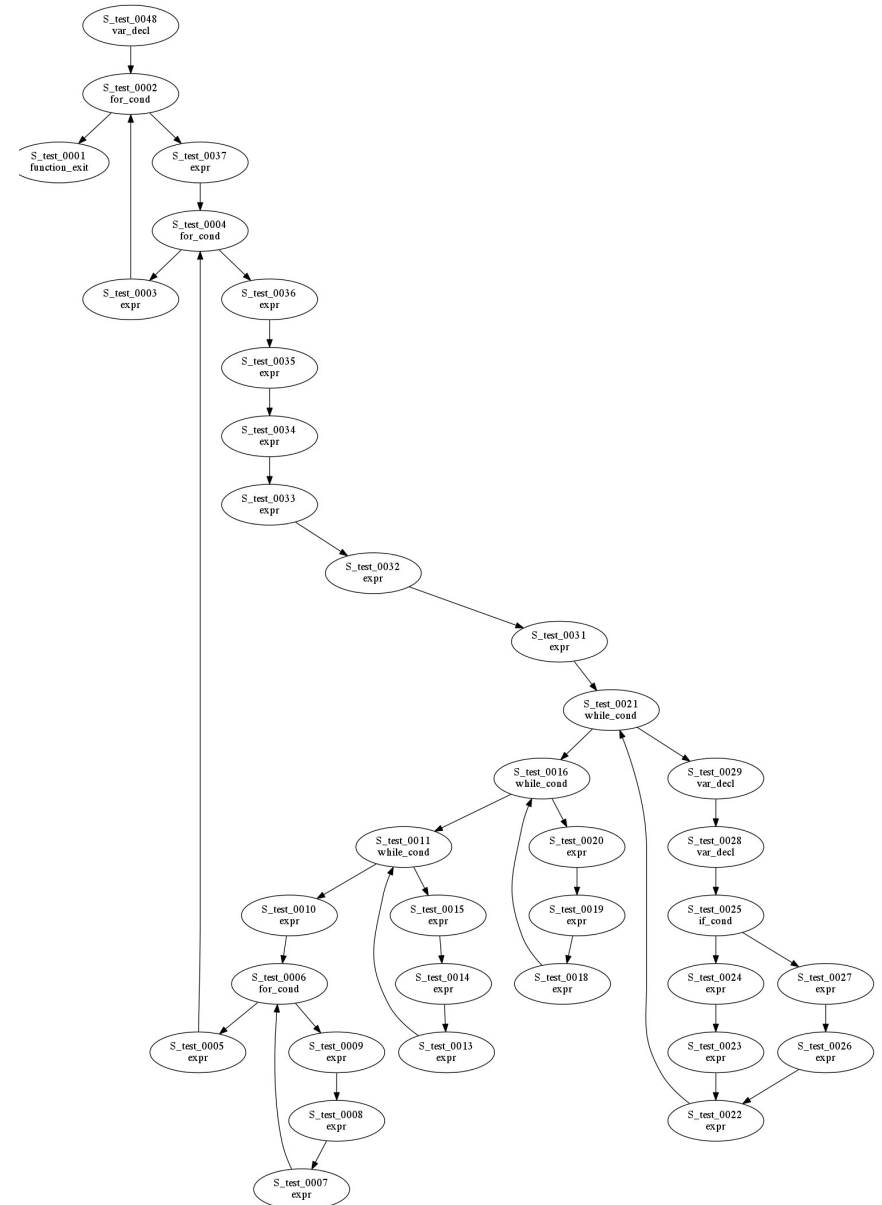
x15

複雑なアルゴリズム処理はJavaで実装

✓ バブルソートの状態遷移



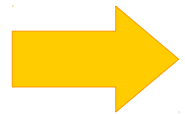
✓ マージソートの状態遷移



Synthesi^jer.scala

Synthesi^jer.scala

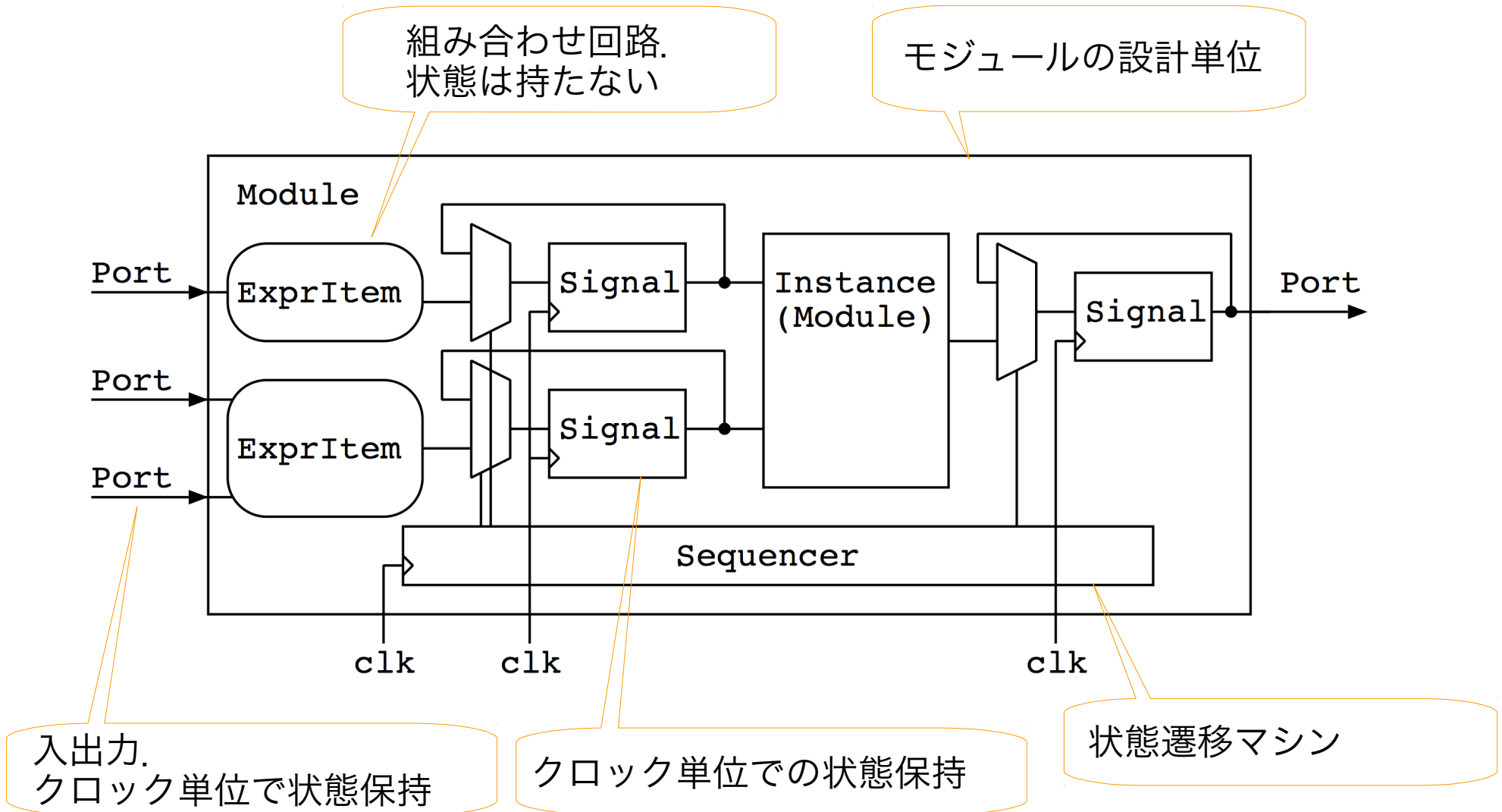
- ✓ Javaで全部書けない...とはいえ
- ✓ もう普通のHDL(= VHDL/Verilog HDL)は書きたくない
 - ✓ たくさんの似たような名前の変数の定義
 - ✓ リセット, パルス信号をもとに戻すの忘れる
 - ✓ などなど
- ✓ 制御構造とデータ処理が同じ, な設計モデルはいやだ
- ✓ 記述の再利用をしたい
 - ✓ データ処理とデータ代入をわけたい



もっと高級なHDLがあればいい

Synthesijer.scala とは

- ✓ Synthesijerのバックエンドを使ってScalaでHDLを書く



Module: 設計単位

- ✓ ハードウェアモジュールの設計単位
- ✓ 構成要素の生成メソッドを提供
- ✓ 自身に相当するVHDL/Verilogを生成するメソッドを提供

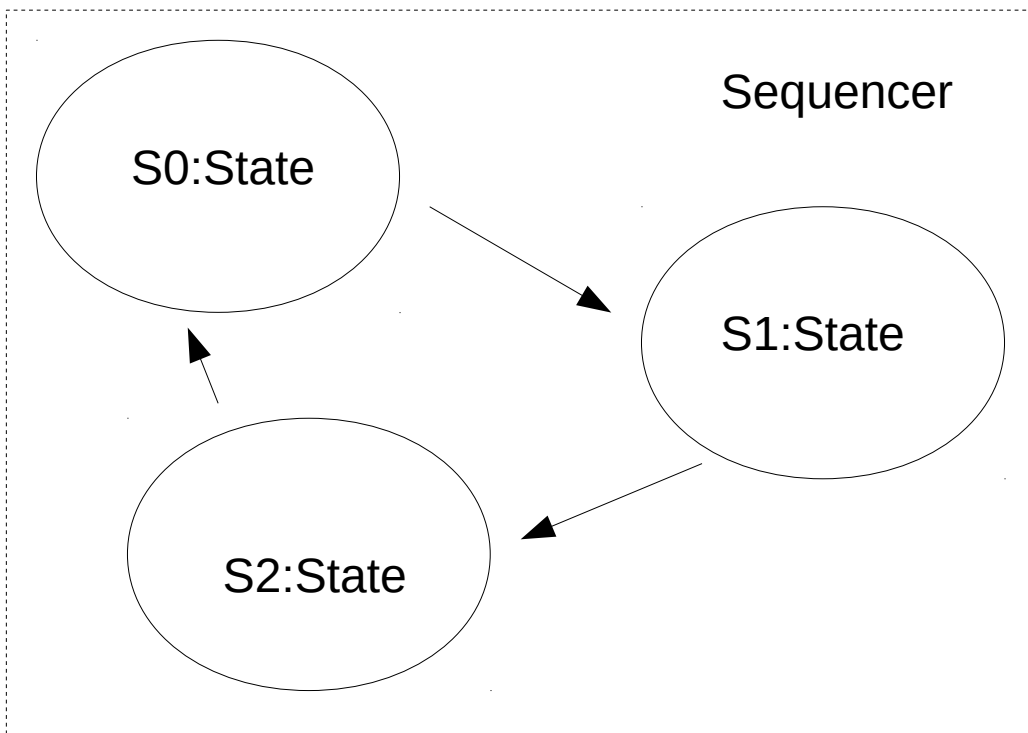
```
def main(args:Array[String]) = {  
  val m = new Module("hoge")  
  m.genVHDL()  
  m.genVerilog()  
  m.genComponentXML() // IPパッケージ  
  m.visualize_statemachine() // dot  
}
```



```
entity hoge is  
  port (  
    clk : in std_logic;  
    reset : in std_logic  
  );  
end hoge;  
architecture RTL of hoge is  
  signal clk_sig : std_logic;  
  signal reset_sig : std_logic;  
begin  
  clk_sig <= clk;  
  reset_sig <= reset;  
  -- expressions  
  -- sequencers  
end RTL;
```

Sequencer: 状態遷移機械

- ✓ 状態遷移機械(Sequencer)と状態(State)
- ✓ $\rightarrow$ 演算子で状態遷移. 状態毎に遷移する.
- ✓ $\rightarrow$ 演算子の引数にタプルを渡した場合,
タプルの1番目の引数が真の場合にのみ遷移.



```
val m = new Module("hoge")
```

```
val seq0 = m.sequencer("S")  
val s0, s1, s2 = seq0.add()  
s0  $\rightarrow$  s1  $\rightarrow$  s2  $\rightarrow$  s0
```

```
val seq1 = m.sequencer("T")  
val t0, t1, t2 = seq1.add()  
t0 * flag  $\rightarrow$  t1  
t0 * (flag!)  $\rightarrow$  t2
```

Signal/Port: 状態に応じた値の保持

- ✓ SignalはFPGA上のレジスタに対応
- ✓ Portはモジュール外部とのデータ入出力ポート
- ✓ `:=`演算子で値を代入できる
- ✓ `<=`演算子で、 特定の状態で値を代入できる

```
val m = new Module("hoge")
```

```
val p = m.inP("data", 32)  
val seq = m.sequencer("S")  
val s0, s1, s2 = seq.add()
```

```
val s = m.signal(32)  
val t = m.signal(32)
```

```
s := p  
t <= s0 * u
```

ExprItem: 組み合わせ回路

✓ 演算構成要素

✓ Signal/Port

✓ Value (即値)

✓ ExprItem同士の演算結果

```
val m = new Module("hoge")
```

```
val s = m.signal(32)
```

```
val t = m.signal(32)
```

```
val e0 = s + s
```

```
t := e0
```

```
s <= s0 * (s + 1)
```

表 A.1 ExprItem 同士で定義される二項演算

記号	演算内容
+, -, *	和, 差, 積
and, or, xor	論理積, 論理和, 排他的論理和
==, /=	等しい, 等しくない
<, >, leq, geq	小さい, 大きい, 以下, 以上
&, concat	結合

表 A.2 ExprItem と Int で定義される二項演算

記号	演算内容
+, -	和, 差
==, /=	等しい, 等しくない
<, >, leq, geq	小さい, 大きい, 以下, 以上
>>, >>>	算術右シフト, 論理右シフト
<<	左シフト

表 A.3 ExprItem に対して定義されるその他の演算

記号 (用例)	演算内容
e!	否定.
?(e0, e1)	選択. 自身が'1' のとき e0, '0' のとき e1 を返す
padding(n)	n-bit の算術 bit 伸長
padding0(n)	n-bit の論理 bit 伸長
drop(n)	先頭から n-bit を落とす
range(n0, n1)	n0-bit ~ n1-bit を切り出す
ref(n)	n-bit 目を取り出す

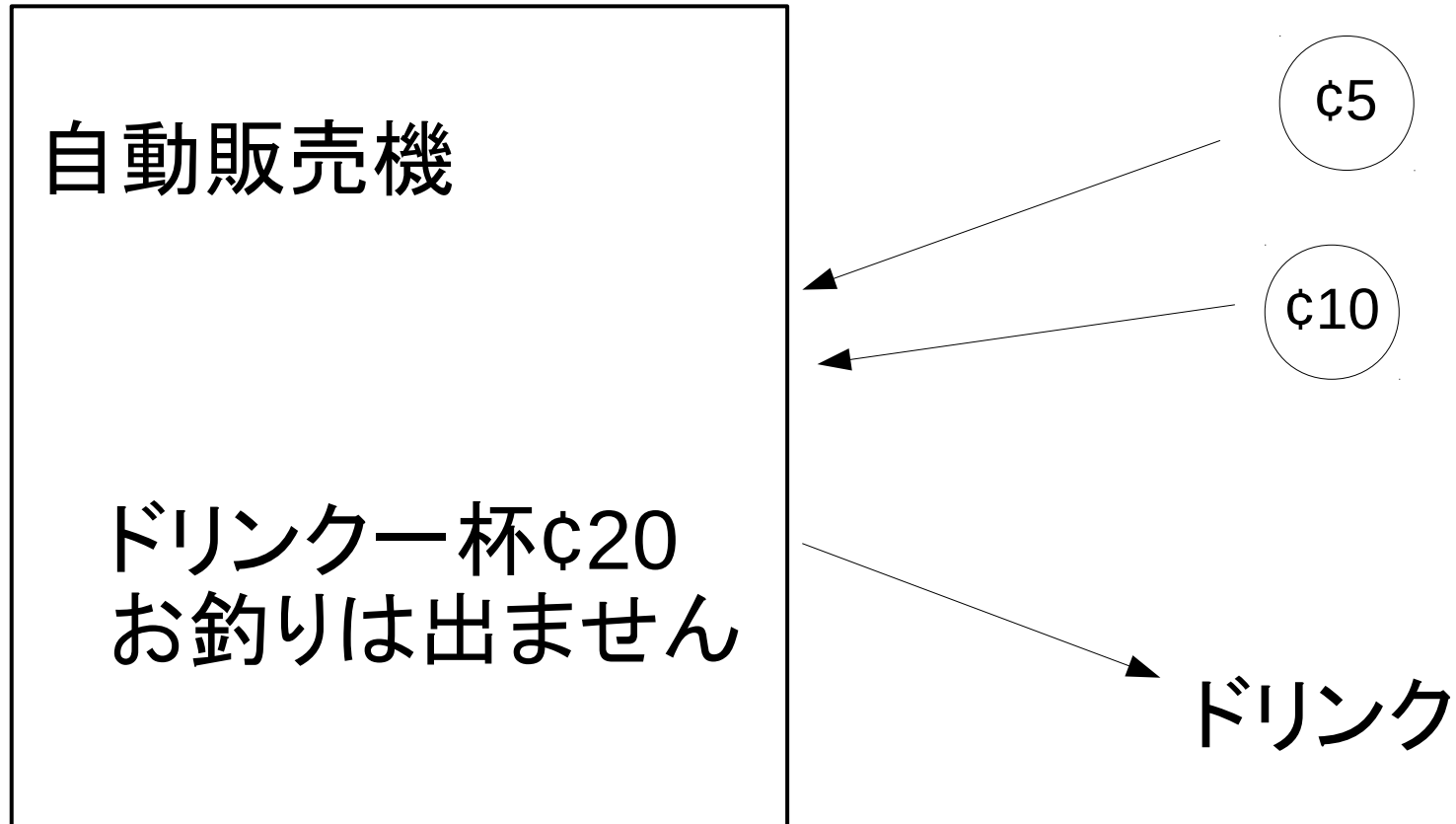
はじめてのSynthesijer.scala

✓ Lチカの例

```
def generate_led() : Module = {  
  val m = new Module("led")  
  val q = m.outP("q")  
  val counter = m.signal(32)  
  q <= counter.ref(5)  
  
  val seq = m.sequencer("main")  
  counter <= seq.idle * VECTOR_ZERO  
  val s0 = seq.idle -> seq.add()  
  counter is (counter + 1 in s0) // counter <= s0 * (counter+1)  
  return m  
}
```

```
def generate_sim(target:Module, name:String) : SimModule = {  
  val sim = new SimModule(name)  
  val inst = sim.instance(target, "U")  
  val (clk, reset, counter) = sim.system(10)  
  inst.sysClk <= clk  
  inst.sysReset <= reset  
  return sim  
}
```

ベンディングマシンの例



RTLでの設計の場合

```
...
type stateType is (NONE, CHARGE_5, CHARGE_10, CHARGE_15, OK)
signal state, next_state: stateType := NONE;
...
case (state) is
  when NONE =>
    if nickel then
      next_state <= CHARGE_5;
    elsif dime then
      next_state <= CHARGE_10;
    else
      next_state <= NONE;
    end if;
  when CHARGE_5 =>
    if nickel then
      next_state <= CHARGE_10;
    elsif dime then
      next_state <= CHARGE_15;
    else
      next_state <= NONE;
    end if;
...

```

ベンディングマシンの例

```
class VendingMachine(n:String,c:String,r:String) extends Module(n,c,r){
```

```
  val nickel = inP("nickel")  
  val dime = inP("dime")  
  val rdy = outP("rdy")  
  val seq = sequencer("main")
```

```
  val s5,s10,s15,s_ok = seq.add()  
  rdy <= seq.idle * LOW
```

```
  rdy is (s_ok in HIGH) // rdy <= s_ok * HIGHと等価  
  seq.idle * nickel -> s5  
  seq.idle * dime -> s10
```

```
  (s5 | nickel -> s10  
   | dime -> s15) // ガードっぽくもかける
```

```
  s10 * nickel -> s15  
  s10 * dime -> s_ok
```

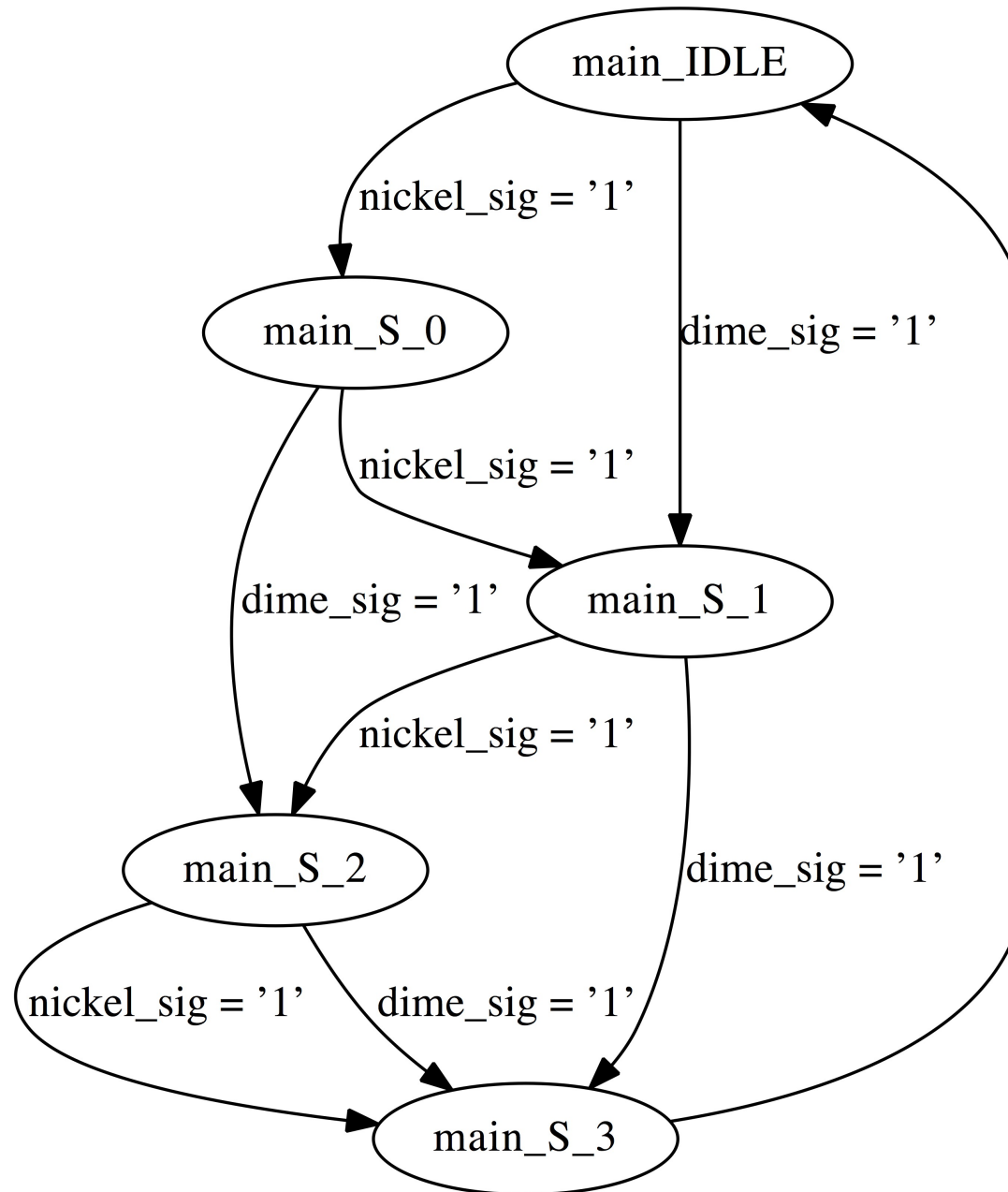
```
  s15 * nickel -> s_ok  
  s15 * dime -> s_ok
```

```
  s_ok -> seq.idle
```

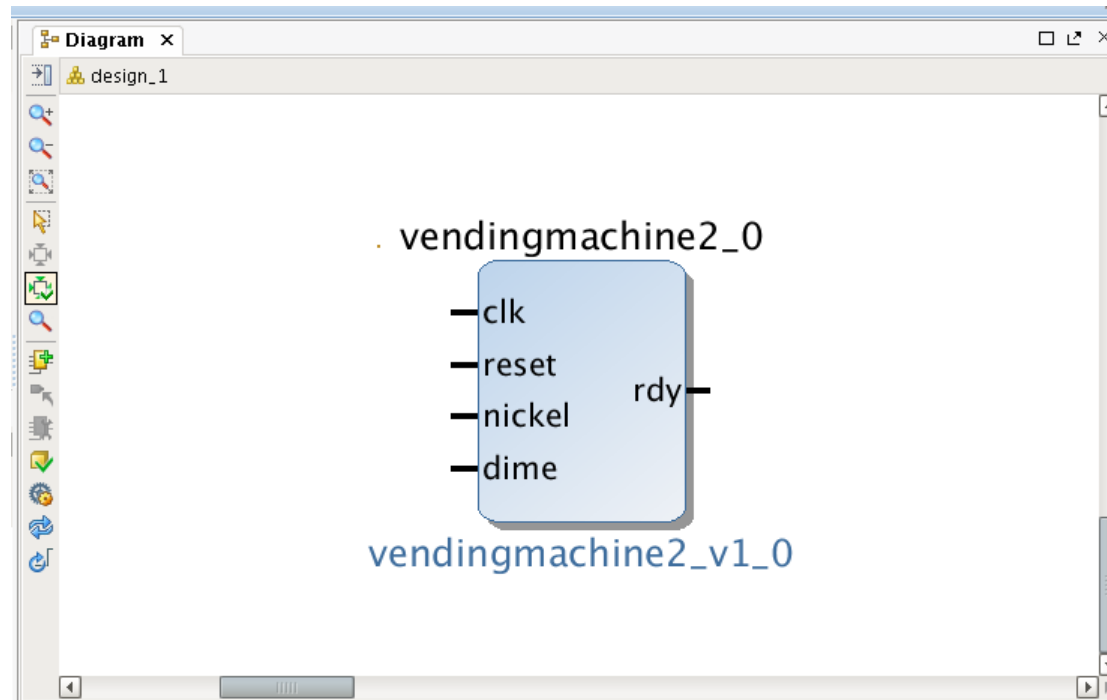
```
}
```

```
val m = new VendingMachine("machine", "clk", "reset")  
m.genVHDL()  
m.visuallize_statemachine()
```

ベンディングマシンの状態マシン



ベンディングマシンのIPコア

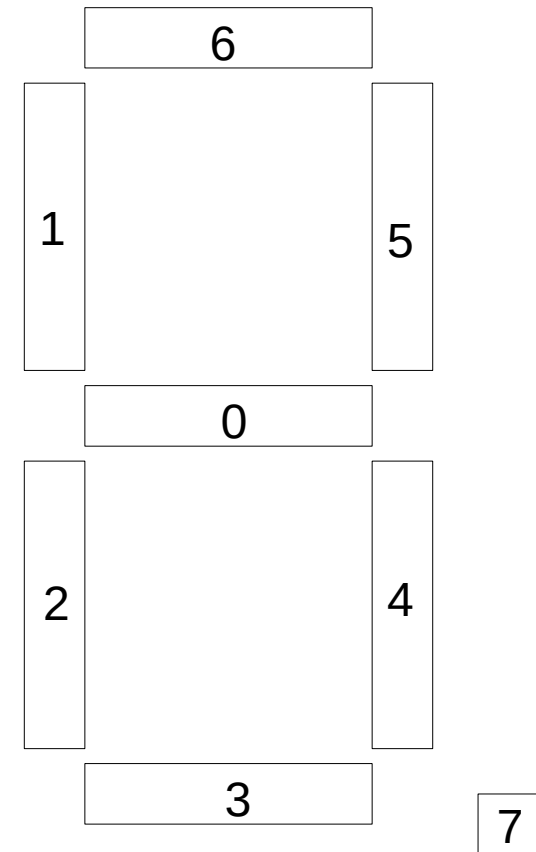
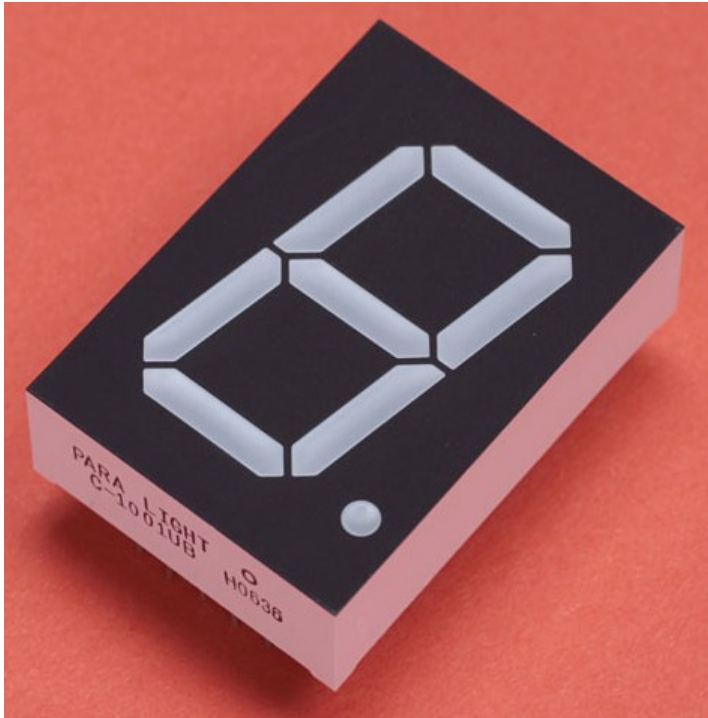


Scalaの機能を使った設計効率の向上

- ✓ 設計資産をメソッドやクラスとして保存, 再利用
- ✓ 類似オブジェクトや類似オブジェクトの生成
- ✓ ループとパターンマッチによる直列化/復元コードの生成

設計資産の再利用

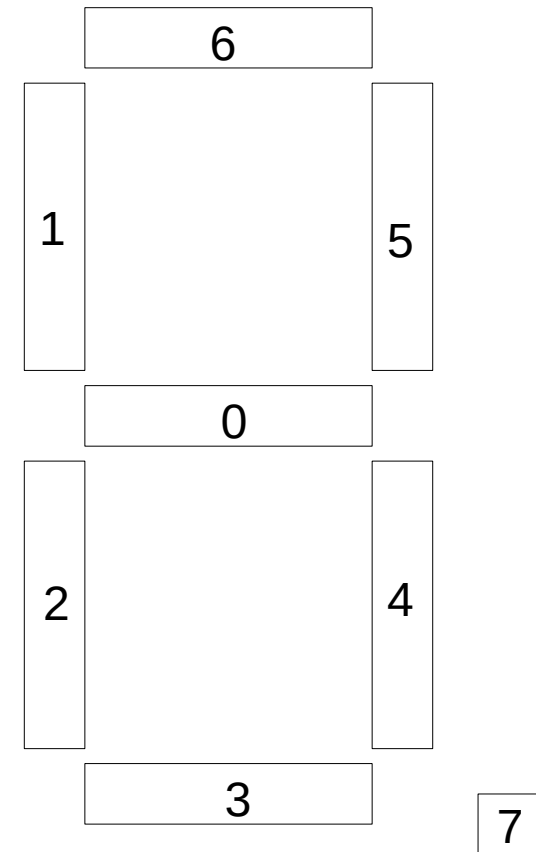
例: デコーダ (7セグメントLEDの点灯)



設計資産の再利用

従来のHDLによるRTL設計の場合

```
process (data)
begin
  case (data) is
    when 0 => segment <= X"7e";
    when 1 => segment <= X"30";
    ...
    when 9 => segment <= X"79";
    when others => null;
  end case;
end process;
```



毎回このパターンを書かないといけない。

設計資産の再利用

Synthesi^jer.scalaを使う場合

パターンを作るメソッド. 回路のレシピ.

```
def decoder(s:ExprItem, l:List[(Int,Int)], w:Int) = {  
  l.foldRight(value(0,w)){  
    (a, z) => ?(s == a._1, value(a._2, w), z)  
  }  
}
```



実データに合わせた回路の生成

```
val tbl = List((0, 0x7e), (1, 0x30), ..., (9, 0x79))  
segment := decoder(data, tbl, segment.width())
```

類似オブジェクトの生成

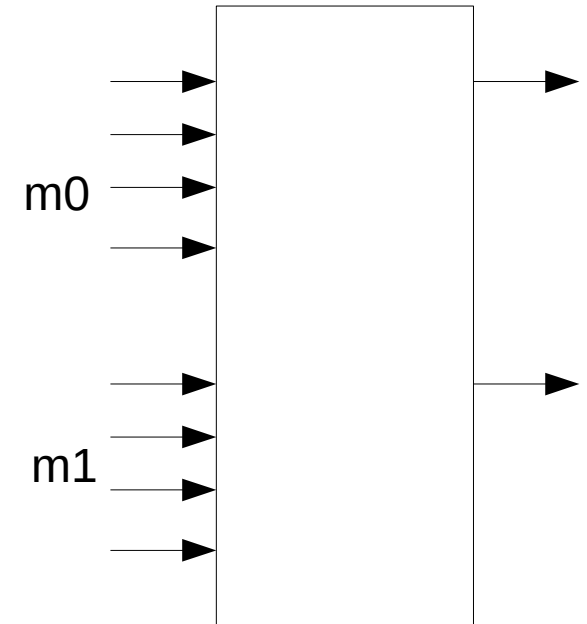
ユニットをクラスとしてまとめる

```
class MemPort(m:Module, p:String, d:Int){  
  val we = inP(p + "_we")  
  val oe = inP(p + "_oe")  
  val addr = inP(p + "_addr")  
  val din  = inP(p + "_din")  
  val dout = outP(p + "_dout")  
}
```



生成

```
val m0 = new MemPort(module, "m0", 32)  
val m1 = new MemPort(module, "m1", 8)
```



直列化/復元コードのテンプレート化

```
val data = inP("data", 32)
```

```
val seq = seq("main")
```

```
def for_first(s:State, d:Port) = {...}
```

```
def for_second(s:State, d:Port) = {...}
```

```
def for_rest(s:State, d:Port) = {...}
```

```
var s = seq.add()
```

```
for(i <- 0 to 1){
```

```
  i match {
```

```
    case 0 => for_first(s, data)
```

```
    case 1 => for_second(s, data)
```

```
  }
```

```
  s = s -> seq.add()
```

```
}
```

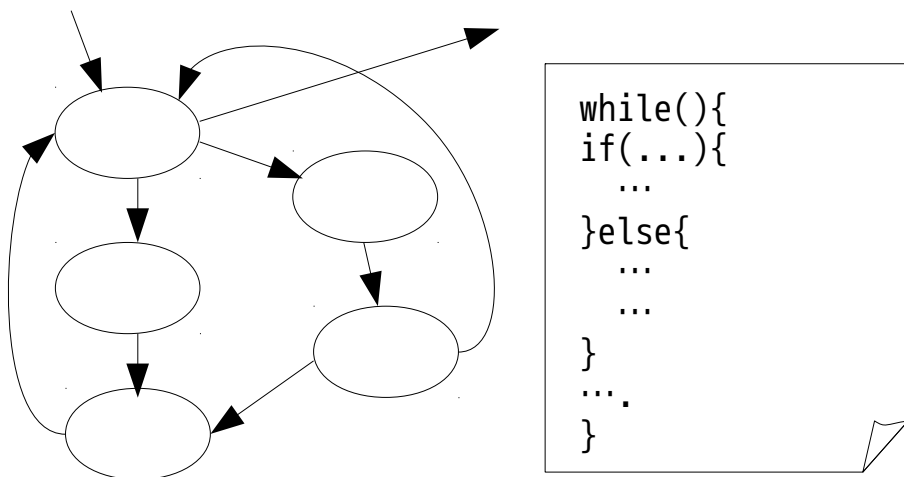
```
for_rest(s, data)
```

```
s -> (done, done_state)
```

まとめと今後のロードマップ

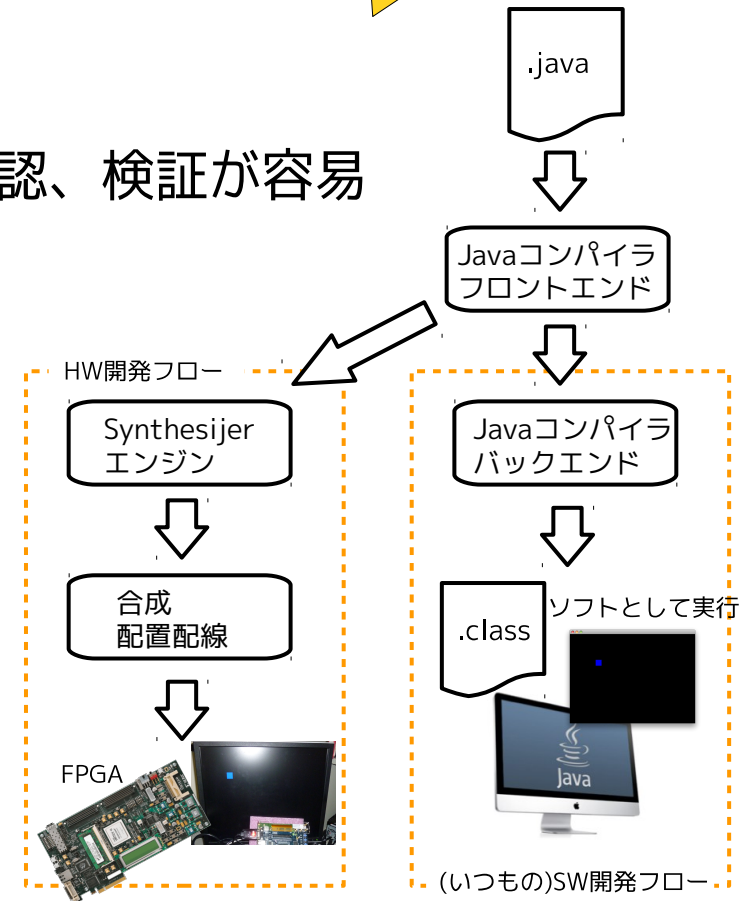
Synthesijer - まとめ -

- ✓ JavaプログラムをFPGA上のハードウェアに変換
 - ✓ 複雑なアルゴリズムのハードウェア実装を楽に
 - ✓ オブジェクト指向設計による再利用性の向上
- ✓ 特殊な記法, 追加構文はない
 - ✓ ソフトウェアとして実行可能. 動作の確認、検証が容易
 - ✓ 書けるプログラムに制限は加える
(動的なnew, 再帰は不可など)



複雑な状態遷移も, Javaの制御構文を使って楽に設計できる

Open-source



同じJavaプログラムをソフトウェアとしても
FPGA上のハードウェアとしても実行可能

ロードマップ



2014年7月



2014年12月



2015年3月



2015年6月

手続き型言語的
基本演算, 操作

整数プリミ
ティブ型変
数の利用

算術, 論理,
シフト演算
(除算・剰
余算以外)

制御構文
(分岐, ルー
プ)

メソッド呼び
出し

除算・剰余
算

浮動小数点
数演算

オブジェクト指向
的インスタンス
協調

finalでのイ
ンスタンス
の生成

インスタン
スメソッドの
呼び出し

インスタン
ス変数への
リードアクセ
ス

インスタン
ス変数への
ライトアクセ
ス

インスタン
ス間での配
列読み書き
のサポート

インスタン
スの配列,
配列の配列
のサポート

コンストラク
タのサポー
ト

インスタン
スのチェイ
ンアクセス

並列化
パフォーマンス

基本ブロッ
ク内自動並
列化

Threadに
よる明示的
な処理の並
列化をサ
ポート

ループ内パ
イプライニ
ング

ライブラリ

整数プリミ
ティブ変数
の配列

AXI接続用
のコンポー
ネントライ
ブラリの提供

Stringクラ
スのサポー
ト

ユーザビリティ

コマンドライ
ンでのコン
パイル

HDLモ
ジュールと
のバイン
ディング機
構

FPGA合成
ツールとの
連携, 統一
的な開発フ
ローの提供

スケジュー
リング可視
化ツール

HDL生成ラ
イブラリを
活用した
DSLの提供

参考

- ✓ <http://synthesijer.sourceforge.net>
 - ✓ リソース一式, クイックスタートガイドなど
- ✓ <http://labs.beatcraft.com/ja/index.php?Synthesijer>
 - ✓ Altera DE0-nanoでのサンプルの動作手順など
(ビートクラフト様)
- ✓ <http://wasa-labo.com/wp/>
 - ✓ わさらぼ ブログ – 開発状況, Tipsなど
- ✓ <http://javarock.sourceforge.net>
 - ✓ Synthesi^jer の前身であるJavaRockプロジェクト

サポートについて

- ✓ バグ, 機能追加要望はSourceforgeのBugTrackにお願いします
- ✓ Synthesi~~j~~erについての有償サポートを希望される場合は,
株式会社アックス様までお問い合わせください.



参考

- ✓ <http://synthesijer.sourceforge.net>
 - ✓ リソース一式, クイックスタートガイドなど
- ✓ <http://labs.beatcraft.com/ja/index.php?Synthesijer>
 - ✓ Altera DE0-nanoでのサンプルの動作手順など
(ビートクラフト様)
- ✓ <http://wasa-labo.com/wp/>
 - ✓ わさらぼ ブログ – 開発状況, Tipsなど
- ✓ <http://javarock.sourceforge.net>
 - ✓ Synthesijer の前身であるJavaRockプロジェクト