# JavaRockではじめる
# 高位合成言語による気楽なFPGA開発

三好 健文

株式会社イーツリーズ・ジャパン

# (JavaRockではじめる)
# 高位合成言語による気楽なFPGA開発

三好 健文

株式会社イーツリーズ・ジャパン

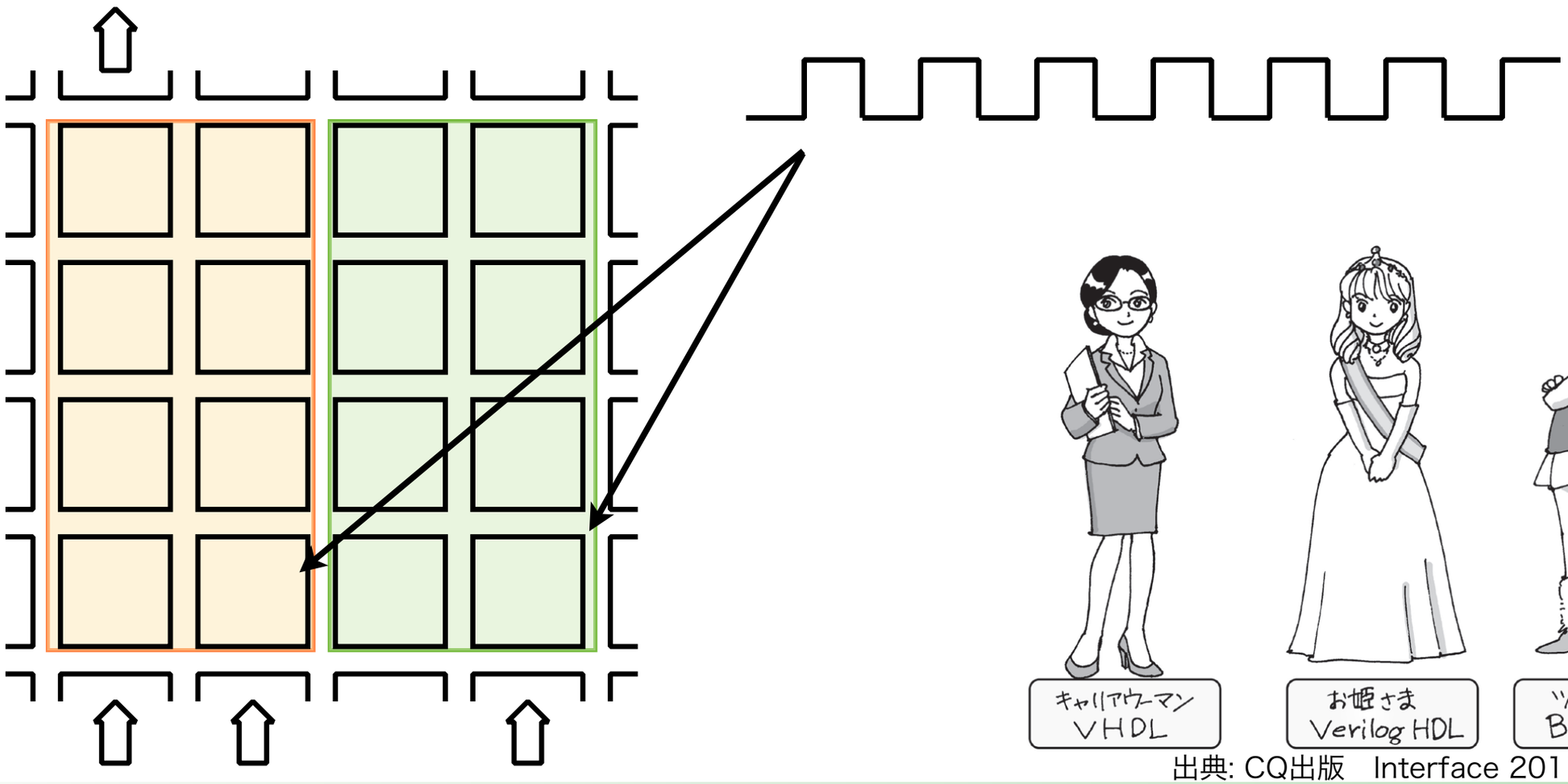
# 今日の流れ

- FPGAと高位合成処理系について
  - FPGAとは
  - 高位合成の例
- JavaRockの実装と設計
- デモ!!

# FPGAとは

## Field Programmable Gate Array

- 論理回路・データパスを自由に作り込める
- クロックレベルの同期と並列性の活用

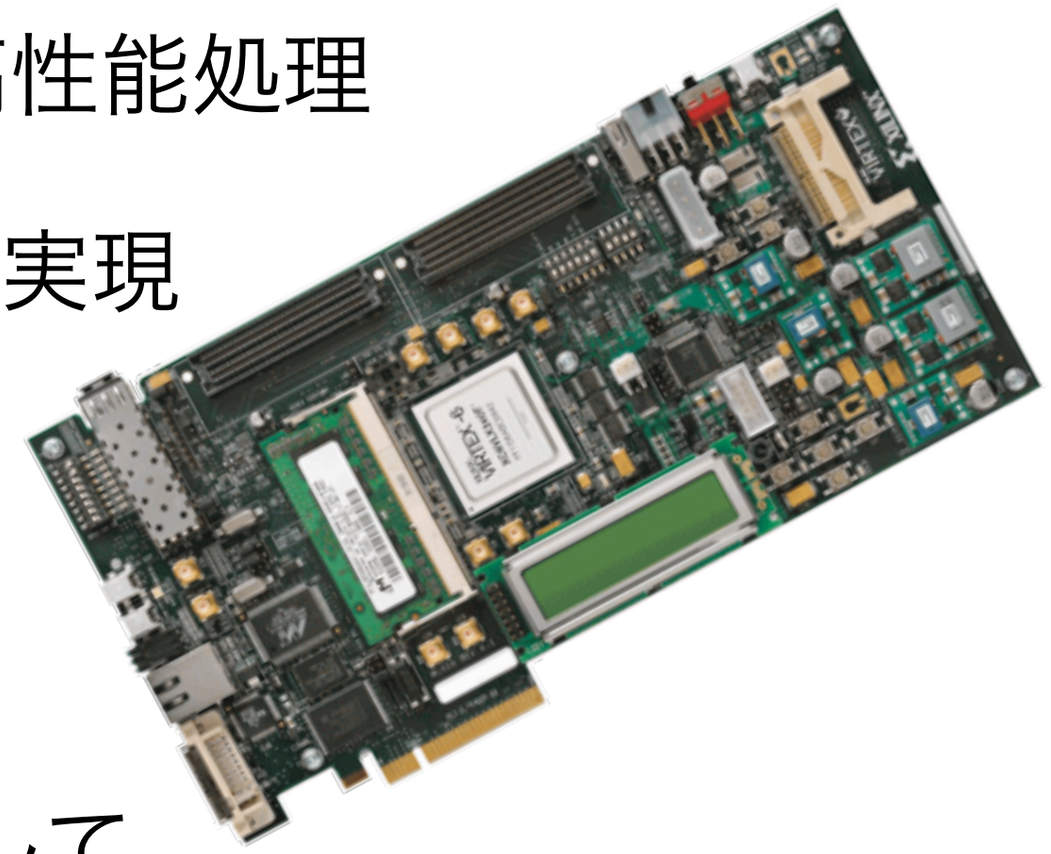





出典: CQ出版　Interface 2011年2月号より

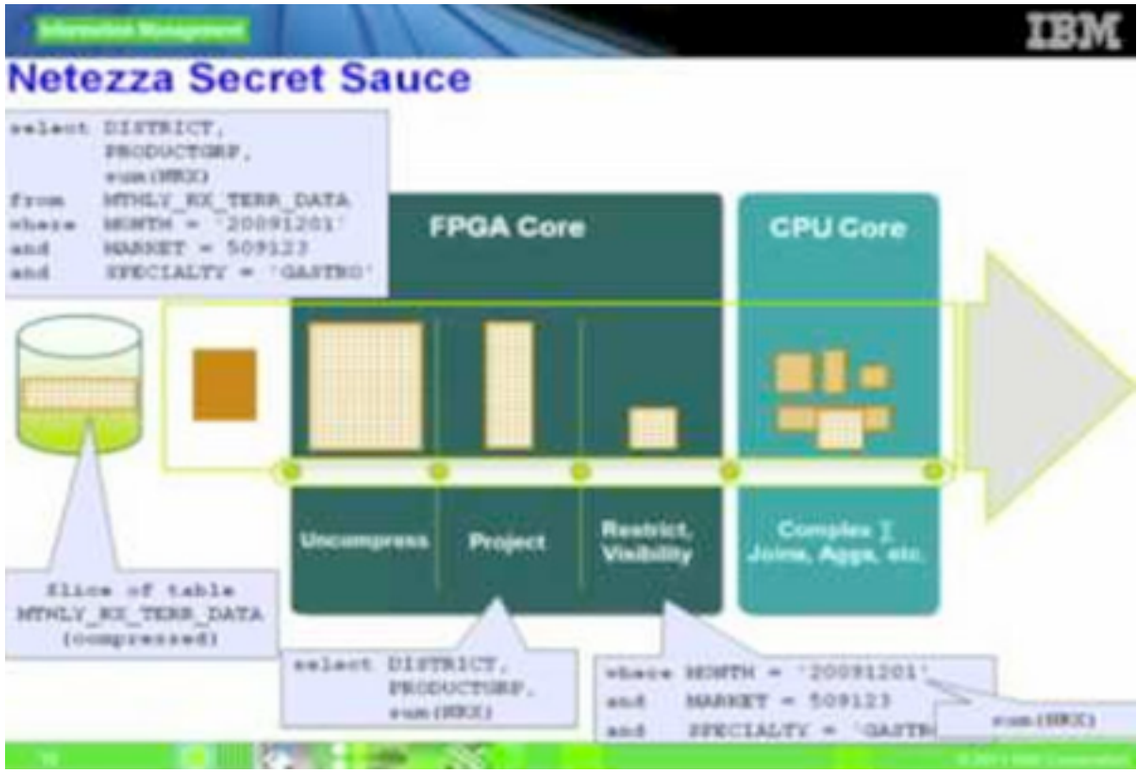
# FPGAの活用シーン

- 独自の回路を実現できるハードウェア
- 特定の処理を低消費電力で高性能処理
- デバイスに近い処理を簡単に実現
  - 自由なI/Oポートの定義



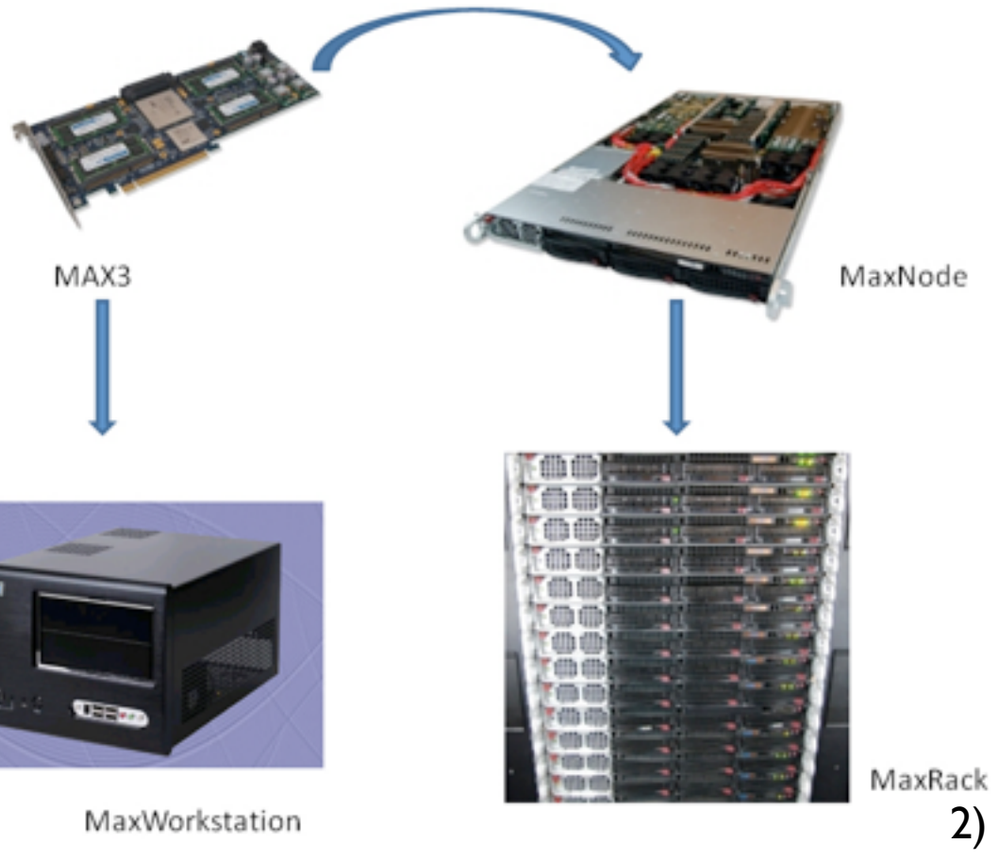
- ASIC開発のプロトタイプとして
- 特定用途向け少数生産の製品として

# FPGAの応用事例




1)

DB, 金融, 油田探索




2)

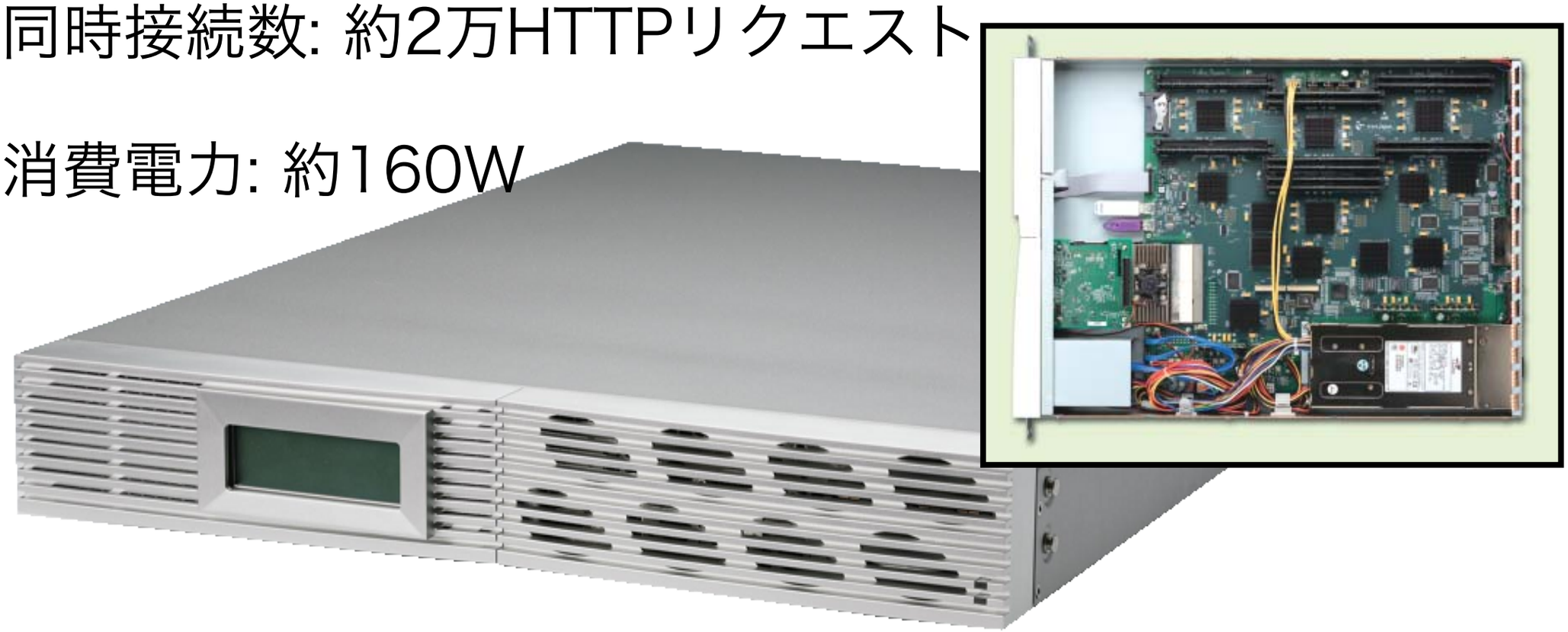
1) http://www.thenewstribe.com/2012/11/01/infotech-becomes-pakistans-first-netezza-certified-ibm-partner/

2) http://techon.nikkeibp.co.jp/article/NEWS/20110502/191552/

# 例) freeocean

## ハードウェアWebキャッシュサーバ: freeocean

- 2006年に販売開始
- 最大スループット: 1Gbps
- 最大同時処理コネクション数: 50万
- 秒間同時接続数: 約2万HTTPリクエスト
- 最大消費電力: 約160W



# 演算リソースとしてのFPGAの魅力

## 数値計算性能

内蔵DSPを活用して...

http://www.altera.com/literature/wp/wp-01142-teraflops.pdf

**< 1.25 TFLOPS, 10～12GFLOPS/W**

## CPUと比べて高い処理能力

Smith-Waterman 法による多重配列アライメン ト処理とモンテカルロ法ベースの金融シミュレーショ ンを FPGA で実行した場合,それぞれ CPU に比べ て 228 倍と 545 倍高速に処理できた

[ 1 ] Reconfigurable Computing in the Multi-Core Era. In Internal Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies, 2010.

# FPGAアプリケーションの研究事例

**@FPGA2013, FPL2013, FCCM2013**

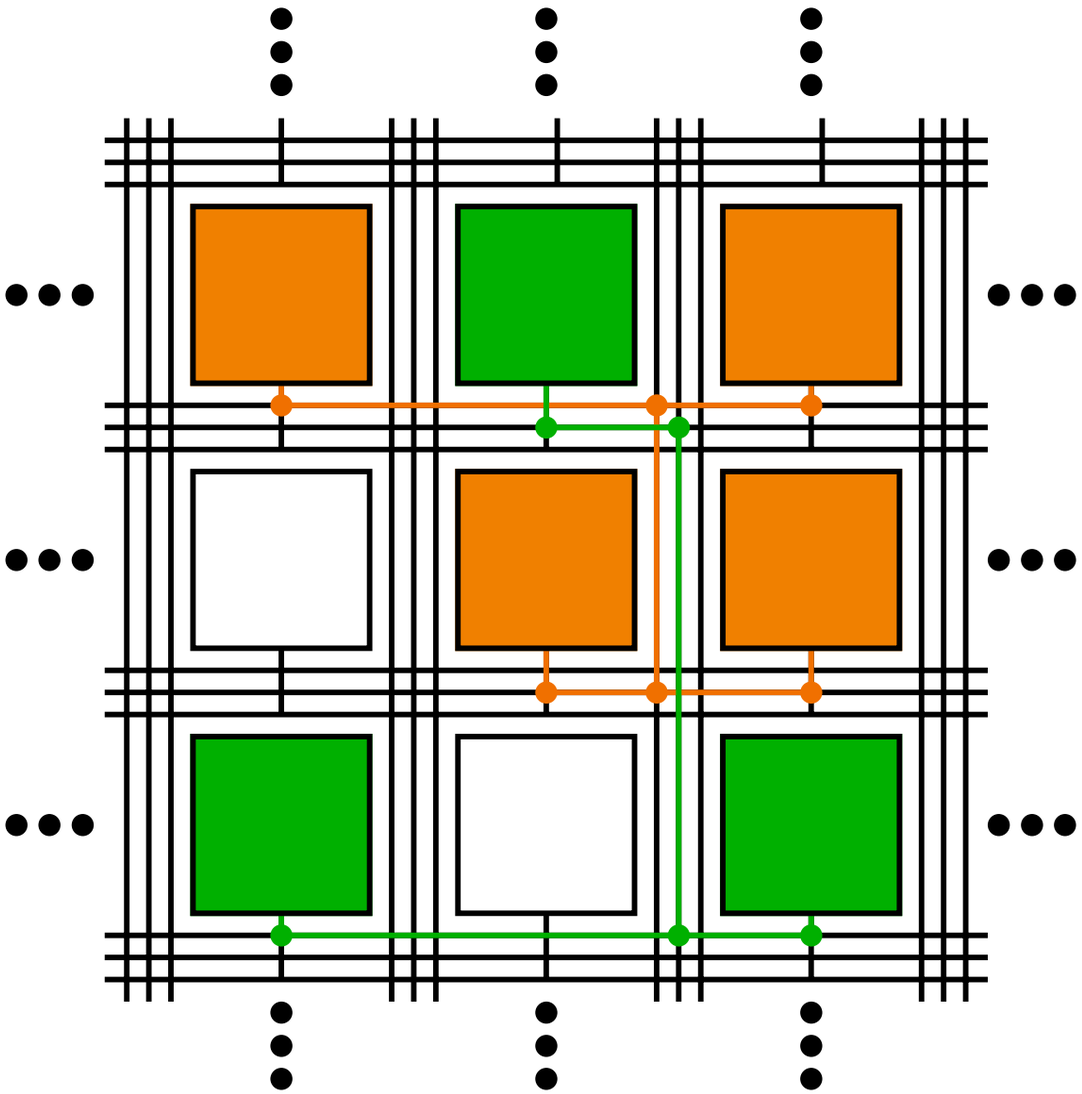
- A Fully Pipelined FPGA Architecture for Stochastic Simulation of Chemical Systems
- A Hardware Accelerated Approach for Imaging Flow Cytometry
- Accelerating Solvers for Global Atmospheric Equations though Mixed-Precision Data Flow Engine
- An FPGA Based Parallel Architecture For Music Melody Matching
- An FPGA Memcached Appliance
- High Throughput and Programmable Online Traffic Classifier on FPGA
- Join Operation for Relational Databases
- A Packet Classifier using LUT cascades Based on EVMDDs(k)
- Memory Efficient IP Lookup in 100Gbps Networks
- A Flexible Hash Table Design for 10GBps Key-Value Stores in FPGAs
- A Secure Copreocessor for Database Applications
- Fast, FPGA-based Rainbow Table Creation for Attacking Encrypted Mobile Communications

**30+**

**.....**

# FPGA上のプログラミングとは

- 論理回路構成要素の演算内容を決める
- 論理回路構成要素同士をどう接続するかを決める



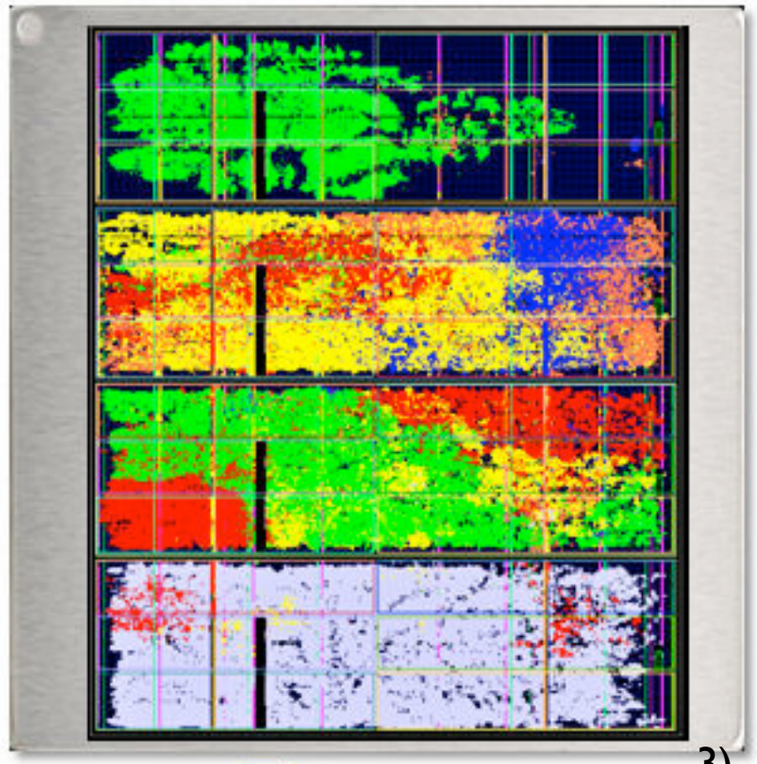
cf. http://commons.wikimedia.org/wiki/File:Two_women_operating_ENIAC_(full_resolution).jpg

# FPGA上のプログラミングとは

| Device[1] | Logic Cells | Configurable Logic Blocks (CLBs) | | DSP Slices[3] | Block RAM Blocks[4] | | | CMTs[5] | PCIe[6] | GTX | GTH | GTZ | XADC Blocks | Total I/O Banks[7] | Max User I/O[8] | SLRs[9] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Slices[2] | Max Distributed RAM (Kb) | | 18 Kb | 36 Kb | Max (Kb) | | | | | | | | | |
| XC7V585T | 582,720 | 91,050 | 6,938 | 1,260 | 1,590 | 795 | 28,620 | 18 | 3 | 36 | 0 | | | | | |
| XC7V2000T | 1,954,560 | 305,400 | 21,550 | 2,160 | 2,584 | 1,292 | 46,512 | 24 | 4 | 36 | 0 | | | | | |
| XC7VX330T | 326,400 | 51,000 | 4,388 | 1,120 | 1,500 | 750 | 27,000 | 14 | 2 | 0 | 28 | | | | | |
| XC7VX415T | 412,160 | 64,400 | 6,525 | 2,160 | 1,760 | 880 | 31,680 | 12 | 2 | 0 | 48 | | | | | |
| XC7VX485T | 485,760 | 75,900 | 8,175 | 2,800 | 2,060 | 1,030 | 37,080 | 14 | 4 | 56 | 0 | | | | | |
| XC7VX550T | 554,240 | 86,600 | 8,725 | 2,880 | 2,360 | 1,180 | 42,480 | 20 | 2 | 0 | 80 | | | | | |
| XC7VX690T | 693,120 | 108,300 | 10,888 | 3,600 | 2,940 | 1,470 | 52,920 | 20 | 3 | 0 | 80 | | | | | |
| XC7VX980T | 979,200 | 153,000 | 13,838 | 3,600 | 3,000 | 1,500 | 54,000 | 18 | 3 | 0 | 72 | | | | | |
| XC7VX1140T | 1,139,200 | 178,000 | 17,700 | 3,360 | 3,760 | 1,880 | 67,680 | 24 | 4 | 0 | 96 | | | | | |
| XC7VH580T | 580,480 | 90,700 | 8,850 | 1,680 | 1,8 | | | | | | | | | | | |
| XC7VH870T | 876,160 | 136,900 | 13,275 | 2,520 | 2,8 | | | | | | | | | | | |

1)


2)




3)

1) http://japan.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

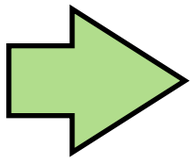
2) http://hitechglobal.com/boards/virtex7_v2000t.htm

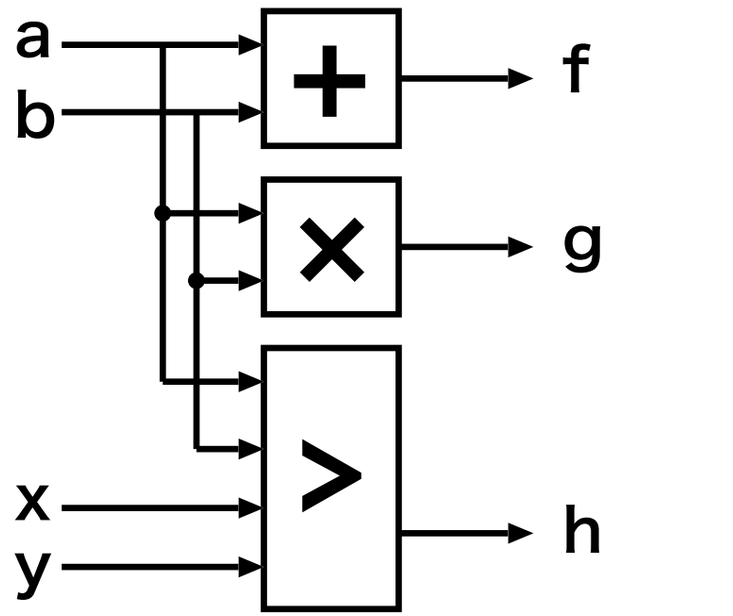
3) http://http://low-powerdesign.com/sleibson/2011/10/25/generation-jumping-2-5d-xilinx-virtex-7-2000t-fpga-delivers-1954560-logic-cells-consumes-only-20w/

# FPGAの開発手法の主役

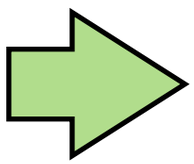
## HDL(Hardware Description Language)によるRTL(Register Transfer Level)設計

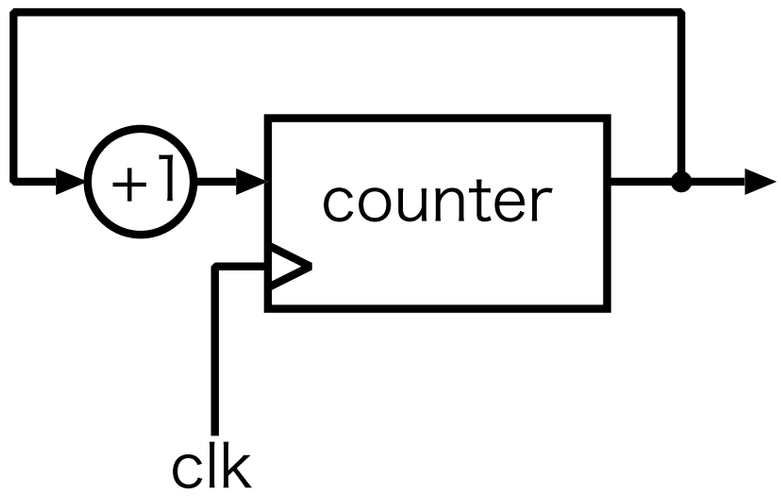
```
process (a, b, x, y)
begin
  f <= a + b;
  g <= a * b;
  if a > b then
    h <= x;
  else
    h <= y;
  end if;
end process;
```






```
process (clk)
begin
  if clk'event and clk = '1' then
    if reset = '1' then
      counter <= (others => '0');
    else
      counter <= counter + 1;
    end if;
  end if;
end process;
```





# HDLによる設計のメリット/デメリット

## メリット

- ロジックを抽象化した式/構文で設計できる
- クロックレベルのデータ制御
- 細粒度の並列性の活用

## デメリット

- “状態”を自分で管理しなければいけない
- デバッグ/動作検証が難しい
- アルゴリズムを設計するには記述が煩雑

# HDLによる設計のデメリットを克服する方法

## FPGAを使うのをやめる

- FPGAとは別にプロセッサを持ってくる
- FPGAの中にプロセッサを作る

## 高位合成言語/処理系の活用

- より抽象度の高い設計をする

# HDLによる設計のデメリットを克服する方法

## FPGAを使うのをやめる

- FPGAとは別にプロセッサを持ってくる
- FPGAの中にプロセッサを作る

環境/ソース保守コストの増大

## 高位合成言語/処理系の活用

- より抽象度の高い設計をする

# HDLによる設計のデメリットを克服する方法

## FPGAを使うのをやめる

- FPGAとは別にプロセッサを持ってくる
- FPGAの中にプロセッサを作る

## 高位合成言語/処理系の活用

- より抽象度の高い設計をする

# 高位合成言語/処理系に何を求めるか

- 記述コストの軽減
  - 高い抽象度の表現方法を利用したい
  - 言語習得のコストは低く抑えたい
- 動作検証/デバッグコストの軽減
  - 短時間で動作を確認したい
  - 見通しよく手軽なデバッグをしたい
- FPGAのパフォーマンスの活用
  - 粗粒度，細粒度の並列性を活用したい
  - IPコア，FPGA内蔵機能を活用したい

# 沢山の高位合成言語/処理系

| ベース | 言語名 |
|---|---|
| C | ImpulseC, SpecC, GorillaC, AutoESL, CWBなど沢山 |
| C++ | SystemC, OCAPI, HP-Machine |
| Java | JHDL, Lime, MaxCompiler, Sea Cucumber, JavaRock |
| C# | Kiwi |
| Python | PHDL, MyHDL |
| Ruby | RHDL |
| ML | CPAH |
| Fortran | DeepC Compiler, ROCCC, SRC-6 |
| Haksell | Lava, Bluespec System Verilog |
| Matlab | MATCH, DEFACTO Compiler |

# 高位合成処理系市場の推移

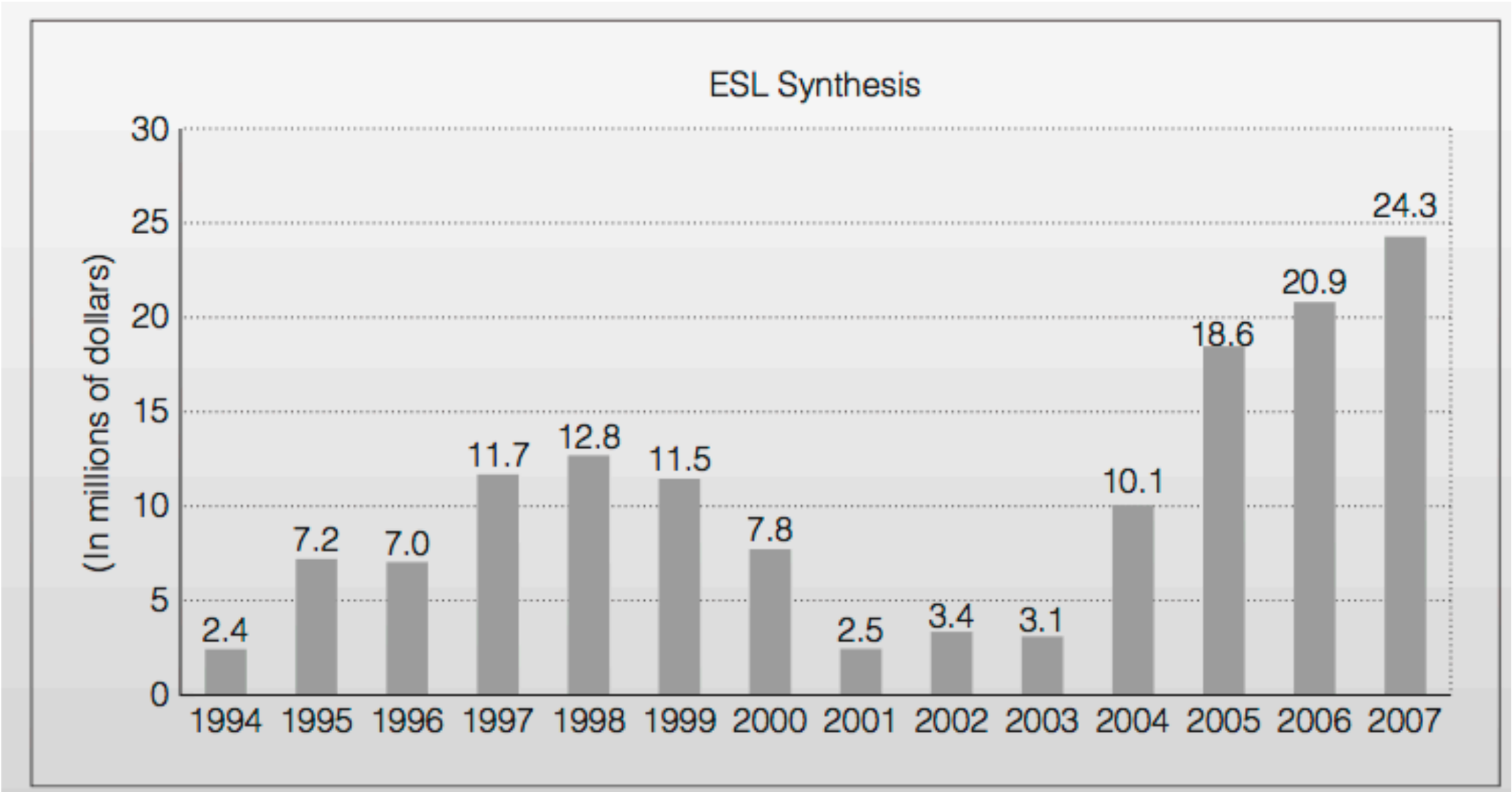



**Figure A. Sales of electronic system-level synthesis tools. (Source: Gary Smith EDA statistics.)**

AutoESL, OpenCL, ImupluseC, CWB...などなどなど

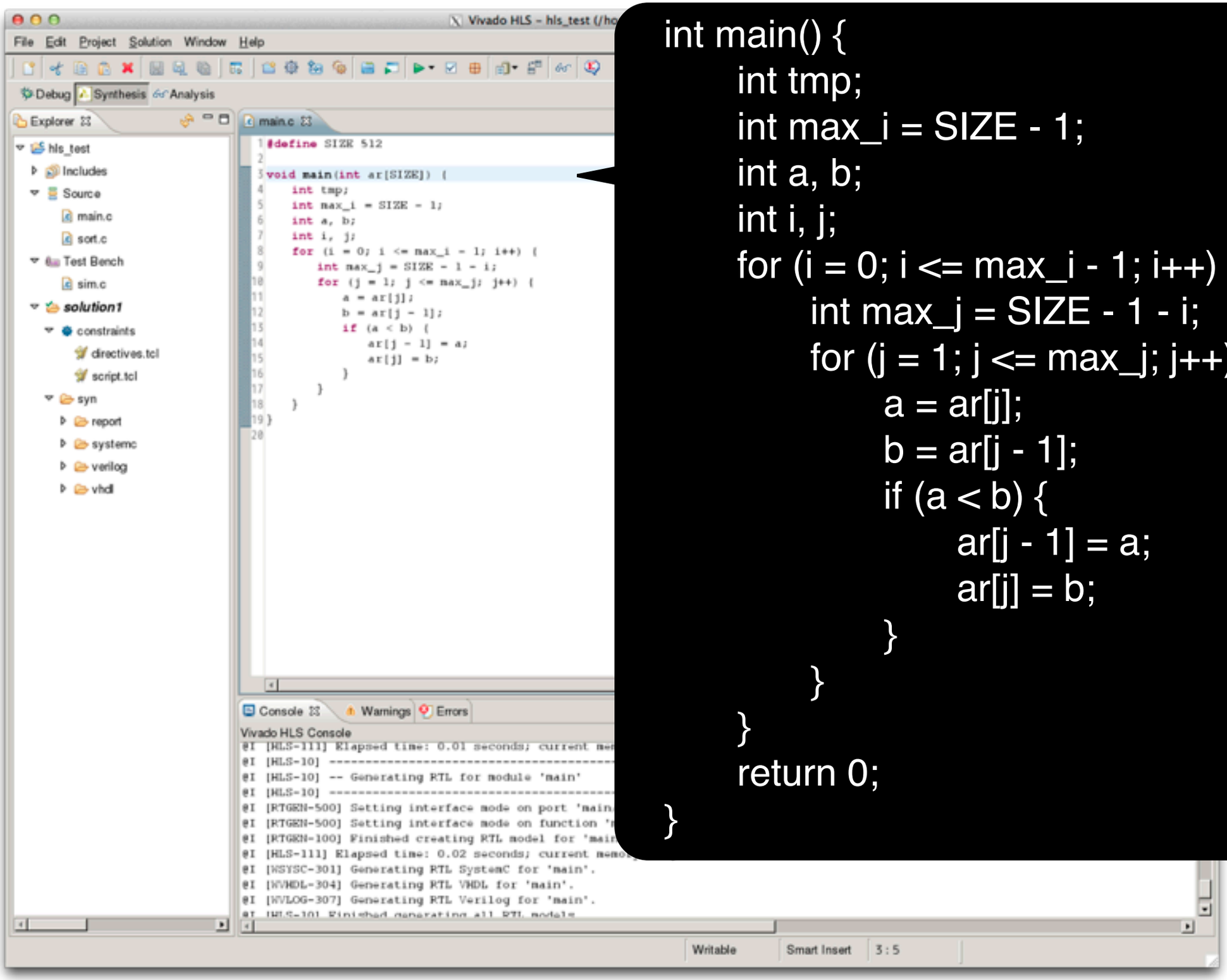
# 高位合成言語の例

- Vivado ESL (AutoESL)
- ImpulseC
- Bluespec System Verilog

# Vivado ESL (AutoESL)

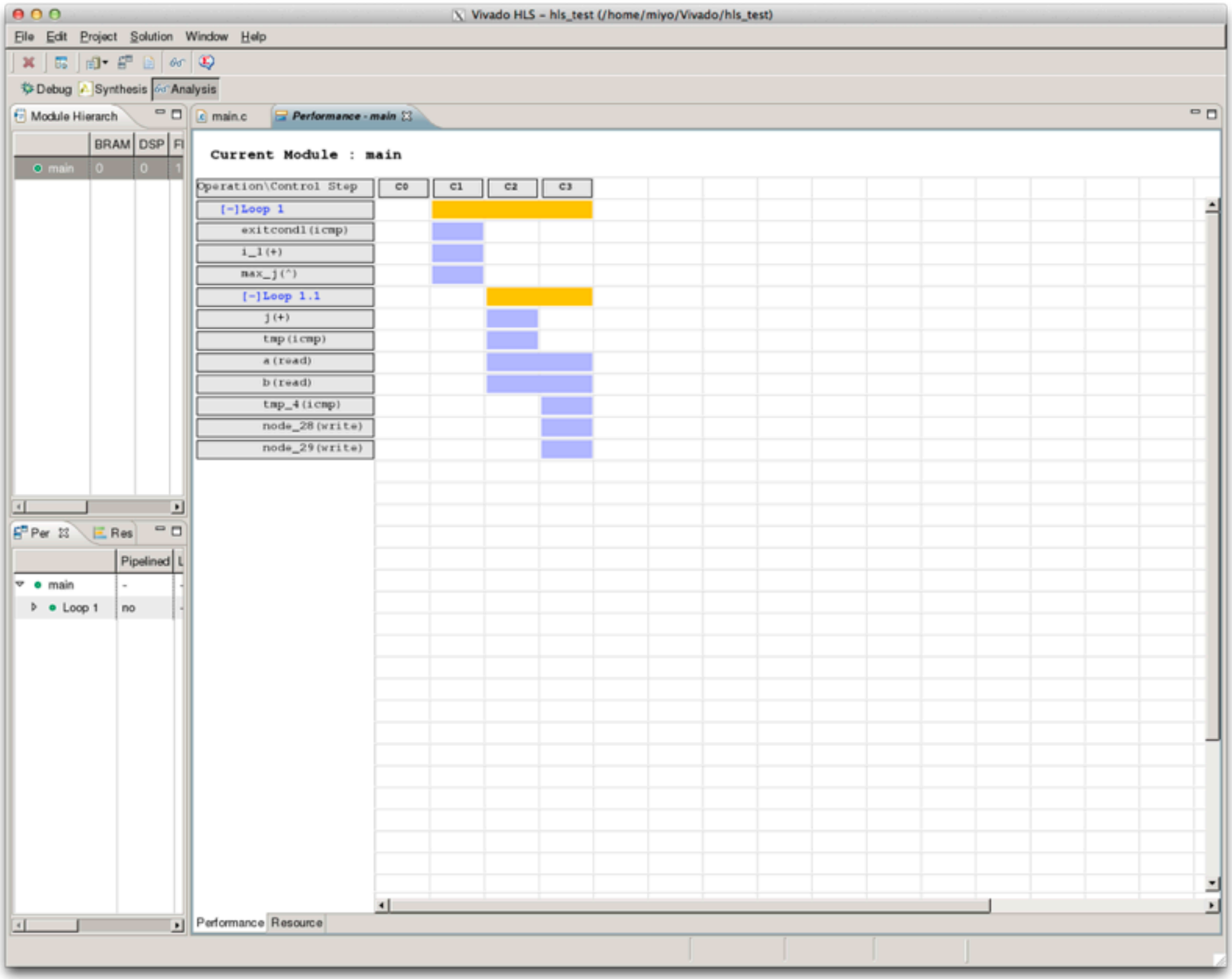
- C，C++，SystemCをサポート
- アルゴリズム合成
- インターフェイス合成(グローバル変数，関数の引数，戻り値)
- 最適化
  - 制御およびデータパスの抽出
  - スケジューリングおよびバインディング
  - 任意精度データ型
  - 並列化

# Vivado HLSでソート




```
int main() {
    int tmp;
    int max_i = SIZE - 1;
    int a, b;
    int i, j;
    for (i = 0; i <= max_i - 1; i++) {
        int max_j = SIZE - 1 - i;
        for (j = 1; j <= max_j; j++) {
            a = ar[j];
            b = ar[j - 1];
            if (a < b) {
                ar[j - 1] = a;
                ar[j] = b;
            }
        }
    }
    return 0;
}
```

# Vivado HLSでソート



# Vivado HLSでソート

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity main is
port (
    ap_clk : IN STD_LOGIC;
    ap_rst : IN STD_LOGIC;
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    ar_address0 : OUT STD_LOGIC_VECTOR (8 downto 0);
    ar_ce0 : OUT STD_LOGIC;
    ar_we0 : OUT STD_LOGIC;
    ar_d0 : OUT STD_LOGIC_VECTOR (31 downto 0);
    ar_q0 : IN STD_LOGIC_VECTOR (31 downto 0);
    ar_address1 : OUT STD_LOGIC_VECTOR (8 downto 0);
    ar_ce1 : OUT STD_LOGIC;
    ar_we1 : OUT STD_LOGIC;
    ar_d1 : OUT STD_LOGIC_VECTOR (31 downto 0);
    ar_q1 : IN STD_LOGIC_VECTOR (31 downto 0) );
end;
```

ポインタ渡しは
デュアルポートRAMになるみたい

```
architecture behav of main is

-- snip --

begin

-- snip --

    -- ap_reg assign process. --
    ap_reg_proc : process(ap_clk)
    begin
        if (ap_clk'event and ap_clk = '1') then
            if (((ap_ST_st3_fsm_2 = ap_CS_fsm) and (ap_const_lv1_0 = tmp_fu_96_p2))) then
                ar_addr_1_reg_145 <= tmp_3_fu_106_p1(9 - 1 downto 0);
            end if;
            if (((ap_ST_st3_fsm_2 = ap_CS_fsm) and (ap_const_lv1_0 = tmp_fu_96_p2))) then
                ar_addr_reg_139 <= tmp_2_fu_101_p1(9 - 1 downto 0);
            end if;
            if ((ap_ST_st2_fsm_1 = ap_CS_fsm)) then
                i_1_reg_120 <= i_1_fu_74_p2;
            end if;
            -- snip --
        end if;
    end process;
    max_j_cast2_reg_125(31 downto 9) <= "00000000000000000000000";
```

クロックに同期して
値をレジスタに保存

# Vivado HLSでソート

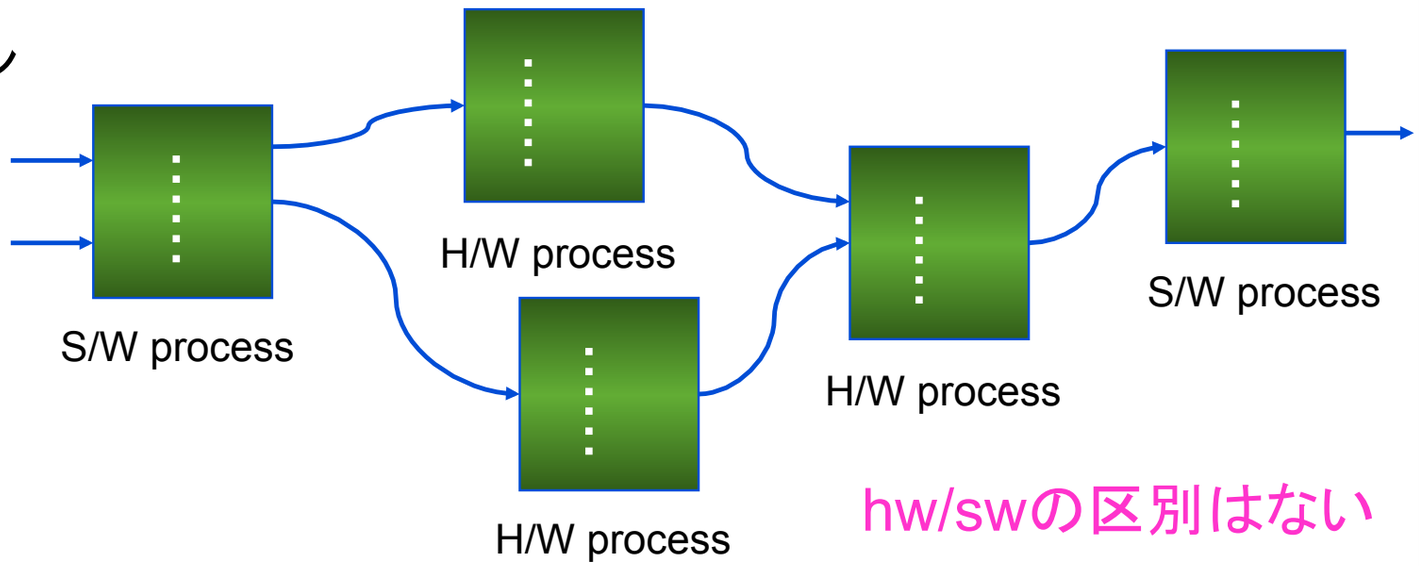
```
    -- the next state (ap_NS_fsm) of the state machine. --
    ap_NS_fsm_assign_proc : process (ap_start , ap_CS_fsm , exitcond1_fu_68_p2 , tmp_fu_96_p2)
    begin
        case ap_CS_fsm is
            when ap_ST_st1_fsm_0 =>
                if (not((ap_start = ap_const_logic_0))) then
                    ap_NS_fsm <= ap_ST_st2_fsm_1;
                else
                    ap_NS_fsm <= ap_ST_st1_fsm_0;
                end if;
            when ap_ST_st2_fsm_1 =>
                if (not((exitcond1_fu_68_p2 = ap_const_lv1_0))) then
                    ap_NS_fsm <= ap_ST_st1_fsm_0;
                else
                    ap_NS_fsm <= ap_ST_st3_fsm_2;
                end if;
    -- snip --
            end case;
        end process;


    -- snip --
        exitcond1_fu_68_p2 <= "1" when (i_reg_43 = ap_const_lv9_1FF) else "0";
        i_1_fu_74_p2 <= std_logic_vector(unsigned(i_reg_43) + unsigned(ap_const_lv9_1));
        j_fu_90_p2 <= std_logic_vector(unsigned(indvar_reg_54) + unsigned(ap_const_lv32_1));
        max_j_cast2_fu_86_p1 <= std_logic_vector(resize(unsigned(max_j_fu_80_p2),32));
        max_j_fu_80_p2 <= (i_reg_43 xor ap_const_lv9_1FF);
        tmp_2_fu_101_p1 <= std_logic_vector(resize(signed(j_fu_90_p2),64));
        tmp_3_fu_106_p1 <= std_logic_vector(resize(signed(indvar_reg_54),64));
        tmp_4_fu_111_p2 <= "1" when (signed(ar_q0) < signed(ar_q1)) else "0";
        tmp_fu_96_p2 <= "1" when (signed(j_fu_90_p2) > signed(max_j_cast2_reg_125)) else "0";
    -- snip --
```

ステートマシン

計算は組み合わせ回路で

# ImpulseC

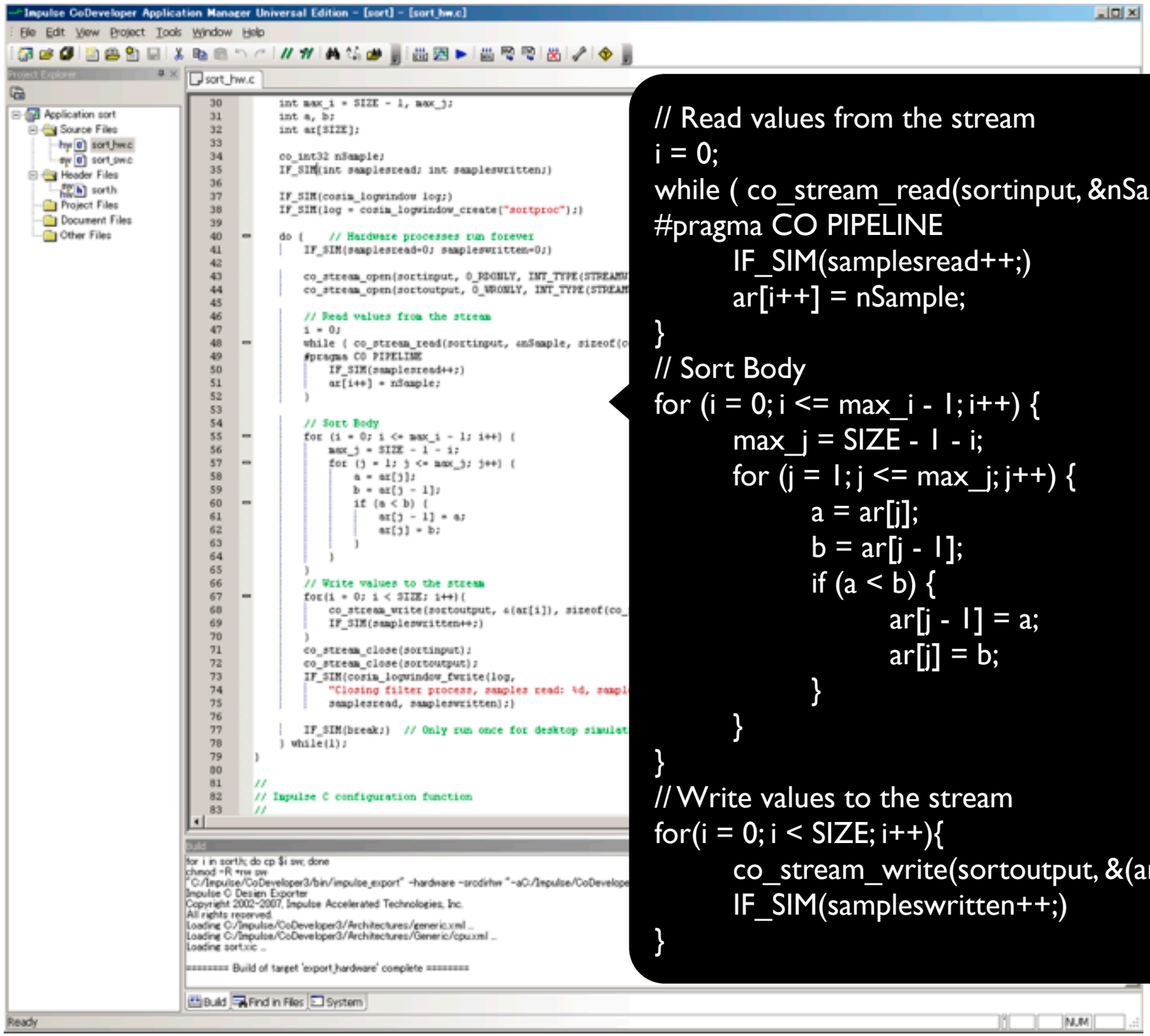
- ANSI-Cが入力
- Stream-Cの商用化
- CSPベースの並列動作モデル
  - データストリームはFIFO
  - プロセスレベル並列化
  - 各プロセスはANSI-設計




- pragmaによる合成制約(Pipeline, Unroll, StageDelayなど)

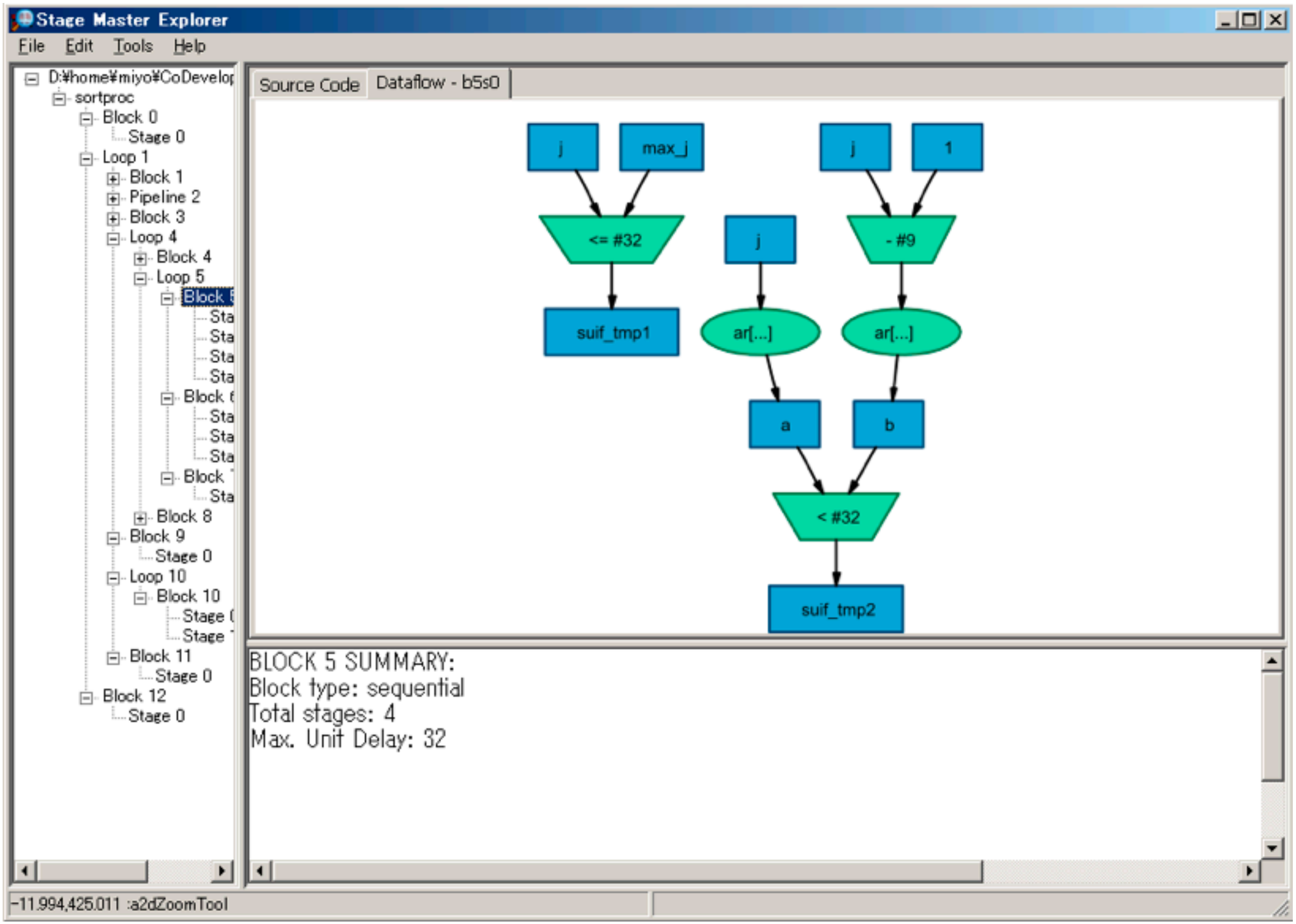
c.f. iLink アイリンク合同会社,“ANSI-C記述でhwモジュール生成 + CPUコアとのhw/sw協調設計 CoDeveloper doc_Ver3.6”

# ImpulseCでソート




```
// Read values from the stream
i = 0;
while ( co_stream_read(sortinput, &nSample, sizeof(co_int32)) == co_err_none ) {
#pragma CO PIPELINE
        IF_SIM(samplesread++;)
        ar[i++] = nSample;
}
// Sort Body
for (i = 0; i <= max_i - 1; i++) {
        max_j = SIZE - 1 - i;
        for (j = 1; j <= max_j; j++) {
                a = ar[j];
                b = ar[j - 1];
                if (a < b) {
                        ar[j - 1] = a;
                        ar[j] = b;
                }
        }
}
// Write values to the stream
for(i = 0; i < SIZE; i++){
        co_stream_write(sortoutput, &(ar[i]), sizeof(co_int32));
        IF_SIM(sampleswritten++;)
}
```

# ImpulseCでソート



# ImpulseCでソート

```
library impulse;
use impulse.components.all;

entity sortproc is
  port (signal reset : in std_ulogic;
    signal sclk : in std_ulogic;
    signal clk : in std_ulogic;
    signal p_sortinput_rdy : in std_ulogic;
    signal p_sortinput_en : inout std_ulogic;
    signal p_sortinput_eos : in std_ulogic;
    signal p_sortinput_data : in std_ulogic_vector (31 downto 0);
    signal p_sortoutput_rdy : in std_ulogic;
    signal p_sortoutput_en : inout std_ulogic;
    signal p_sortoutput_eos : out std_ulogic;
    signal p_sortoutput_data : out std_ulogic_vector (31 downto 0));
end sortproc;
```

入出力はストリーム(FIFO)

```
process (r_suif_tmp3,ni207_suif_tmp0,ni213_suif_tmp2,ni210_suif_tmp1,thisState)
begin
  case thisState is
  when init =>
    nextState <= b0s0;
  when b0s0 =>
    nextState <= b1s0;
  when b1s0 =>
    nextState <= b2s0;
  when b2s0 =>
    nextState <= b3s0;
  when b3s0 =>
    nextState <= b4s0;
  when b4s0 =>
    nextState <= b5s0;
  when b5s0 =>
    if ((not ni210_suif_tmp1(0)) = '1') then
      nextState <= b8s0;
    else
      nextState <= b5s1;
    end if;
  when b5s1 =>
    nextState <= b5s2;
  when b5s2 =>
    nextState <= b5s3;
  when b5s3 =>
    if (ni213_suif_tmp2(0) = '1') then
      nextState <= b6s0;
    elsif ((not ni213_suif_tmp2(0)) = '1') then
      nextState <= b7s0;
    else
      nextState <= b5s3;
    end if;
```

状態遷移

# Vivado HLSでソート

```
s_b2_stall <=  '0' when (s_b2_vstall and s_b2_stage) = "000" else '1';
s_b2_vwrite(0) <= s_b2_stage(0) when s_b2_stall = '0' and s_b2_final = '0' else '0';
s_b2_vwrite(1) <= s_b2_stage(1) when s_b2_stall = '0' and s_b2_final = '0' else '0';
s_b2_vwrite(2) <= s_b2_stage(2) when s_b2_stall = '0' else '0';
s_b2_vcont(0) <= (s_b2_stage(0) and not s_b2_vbreak(0)) and not s_b2_final;
s_b2_vcont(1) <= s_b2_stage(1) and not s_b2_final;
s_b2_vcont(2) <= '0';
s_b2_final <= '0';
s_b2_flushing <= '1' when s_b2_state = flush else s_b2_final;
s_b2_flushed <=
  '1' when s_b2_stall = '0' and (s_b2_vflush or s_b2_stage) = "000" else
  '1' when s_b2_vflush(2) = '1' else
  '0';
with s_b2_state select
  s_b2_done <=
    '0' when init,
    '0' when run,
    s_b2_flushed when flush,
    '0' when others;
-- b3s0
  ni206_i <= X"00000000";

-- b4s0
  ni208_max_j <= sub(X"000001ff", r_i);
  ni209_j <= X"00000001";

-- b5s0
  ni210_suif_tmp1 <= "00000000000000000000000000000000" & cmp_less_equal_s(r_j, r_max_j);
```

いろいろな計算式

```
process (clk)
begin
  if (clk'event and clk='1') then
    case thisState is
    when b1s0 =>
      r_i <= ni201_i;
    when b2s0 =>
      if (s_b2_vwrite(1) = '1') then
        r_i <= ni205_i;
      end if;
    when b3s0 =>
      r_i <= ni206_i;
    when b8s0 =>
      r_i <= ni215_i;
    when b9s0 =>
      r_i <= ni216_i;
    when b10s0 =>
      r_i <= ni219_i;
    when others =>
    end case;
  end if;
end process;

process (clk)
begin
  if (clk'event and clk='1') then
    case thisState is
    when b4s0 =>
      r_max_j <= ni208_max_j;
    when others =>
    end case;
  end if;
end process;
```

クロックに合わせて
状態に合わせて
レジスタにセット

# Bluespec System Verilog

- Verilogに似た構文/中身はHaskellぽい強い型付け言語
- RTL合成も検証も一貫した記述

- 型によるコンパイル時チェック
- 副作用の明示的な記述

- 多数のライブラリ
  - 有限ステートマシン
  - サーバレスポンス/リクエスト

# BSVのコード例

```
module mkTest(Test_ifc);
  Reg#(Bit#(26)) counter <- mkReg(0);
  rule r0;
    counter <= counter + 1;
  endrule
  method Bit#(26) result();
    return readReg(counter);
  endmethod
endmodule
```

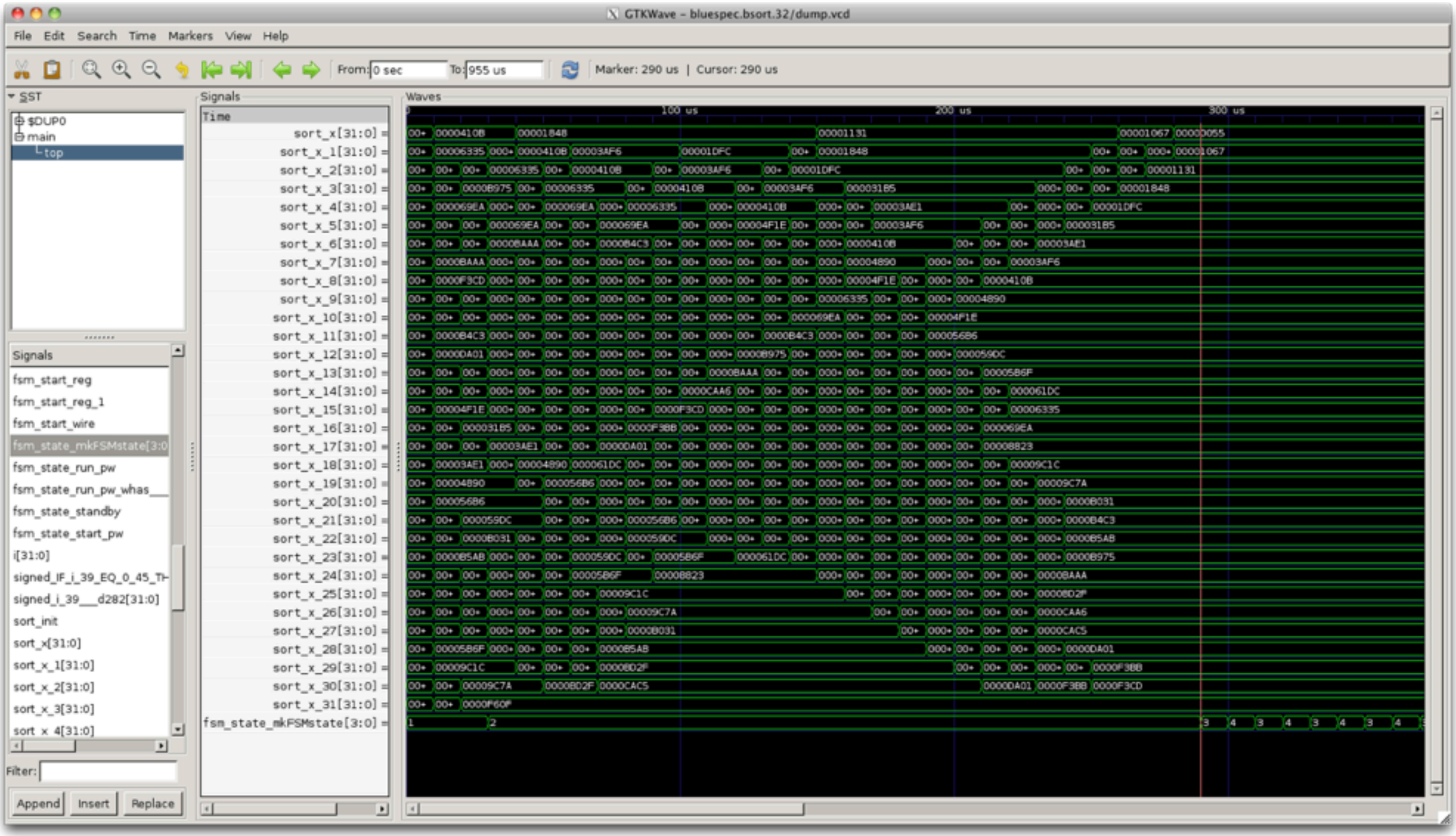
```
module mkTest(CLK, RST_N, result, RDY_result);
  input  CLK;
  input  RST_N;
/* snip */
  // register counter
  assign counter$D_IN = counter + 26'd1 ;
  assign counter$EN = 1'd1 ;
/* snip */
  always@(posedge CLK)
  begin
    if (!RST_N)
      begin
        counter <= `BSV_ASSIGNMENT_DELAY 26'd0;
      end
    else
      begin
        if (counter$EN) counter <= `BSV_ASSIGNMENT_DELAY counter$D_IN;
      end
  end
/* snip */
endmodule  // mkTest
```

# BSVでソート

```
import Vector::*;
import DReg::*;

interface BubSort_ifc#(type size_t);
   method Action start(Vector #(size_t, int) a);
   method Vector#(size_t, int) result();
endinterface

module mkBubSort(BubSort_ifc#(size_t));

   Vector#(size_t, Reg#(int)) x <- replicateM(mkReg(0));

   Reg#(Bool) init <- mkReg(True);
   rule print_version(init);
      action
         $display("Bubble Sort");
         init <= False;
      endaction
   endrule
```

```
   function Bool sorted();
      Bool flag = True;
      for(Integer i = 0; i < valueOf(size_t) - 1; i = i + 1) begin
         if(!(x[i] <= x[i+1])) flag = flag && False;
      end
      return flag;
   endfunction

   for(Integer i = 0; i < valueof(size_t) - 1; i = i + 1) begin
      rule swap((x[i] > x[i+1]));
         x[i] <= x[i + 1];
         x[i+1] <= x[i];
      endrule
   end

   method Action start(Vector#(size_t, int) a);
      writeVReg(x, a);
   endmethod

   method Vector#(size_t, int) result() if(sorted());
      return readVReg(x);
   endmethod

endmodule
```

# BSVでソート


GTKWave – bluespec.bsort.32/dump.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 955 us
Marker: 290 us | Cursor: 290 us
SST
$DUP0
main
top
Signals
fsm_start_reg
fsm_start_reg_1
fsm_start_wire
fsm_state_mkFSMstate[3:0]
fsm_state_run_pw
fsm_state_run_pw_whas___
fsm_state_standby
fsm_state_start_pw
i[31:0]
signed_IF_i_39_EQ_0_45_Th
signed_i_39___d282[31:0]
sort_init
sort_x[31:0]
sort_x_1[31:0]
sort_x_2[31:0]
sort_x_3[31:0]
sort_x_4[31:0]
Filter:
Append
Insert
Replace
Signals
Time
sort_x[31:0]
sort_x_1[31:0]
sort_x_2[31:0]
sort_x_3[31:0]
sort_x_4[31:0]
sort_x_5[31:0]
sort_x_6[31:0]
sort_x_7[31:0]
sort_x_8[31:0]
sort_x_9[31:0]
sort_x_10[31:0]
sort_x_11[31:0]
sort_x_12[31:0]
sort_x_13[31:0]
sort_x_14[31:0]
sort_x_15[31:0]
sort_x_16[31:0]
sort_x_17[31:0]
sort_x_18[31:0]
sort_x_19[31:0]
sort_x_20[31:0]
sort_x_21[31:0]
sort_x_22[31:0]
sort_x_23[31:0]
sort_x_24[31:0]
sort_x_25[31:0]
sort_x_26[31:0]
sort_x_27[31:0]
sort_x_28[31:0]
sort_x_29[31:0]
sort_x_30[31:0]
sort_x_31[31:0]
fsm_state_mkFSMstate[3:0]
Waves
100 us
200 us
300 us

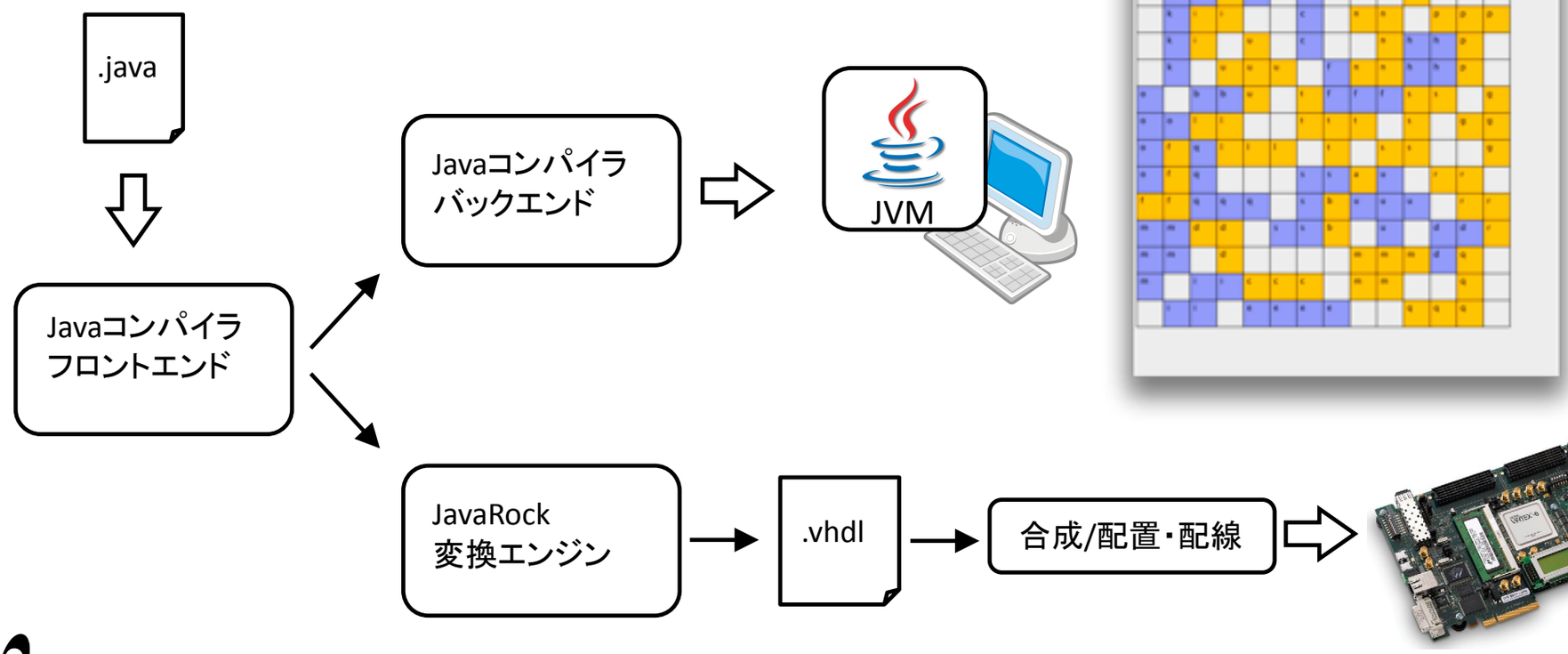

http://javarock.sourceforge.net/

# JavaRockとは

http://javarock.sourceforge.net/

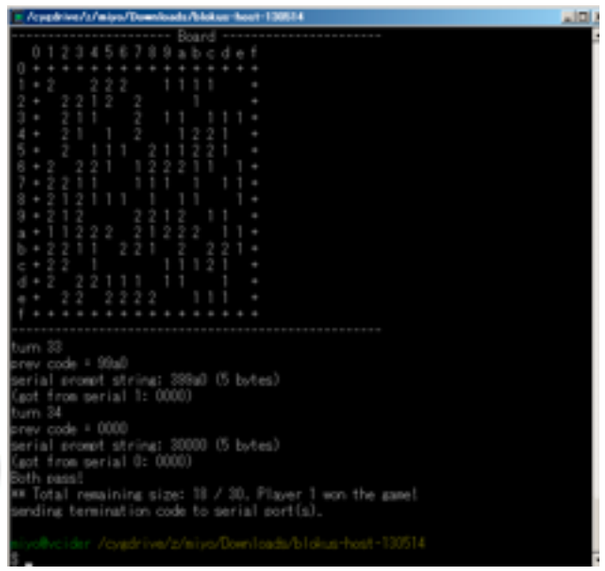
## JavaRockの目指すところ


- Javaプログラムをそのまま HDLに変換→FPGA上のHWにする
  - 追加構文，データ型は導入しない
  - 記述に制限は加える
- “HDLで書けることをJavaで書けるようにする”ではない
- “Javaで書けることを全部HDLにする”ではない







# 高位合成処理系 入力言語として見たJava

- クラスによるオブジェクト指向設計
  ←HWのモジュール設計との親和性が高そう
- Threadやwait-notify，synchronizedの仕組み
  ←言語仕様内で並列性の記述ができそう
- 明示的なポインタを扱う必要がない
  ←言語の想定するメモリ構造から自由になれそう

⇨ コンパイラでがんばれるか？

- 動的な振る舞いがたくさんある
  ←HW化するのは厄介そう

# JavaRockの設計方針

- Javaプログラムをそのまま HW化する
  - 追加構文，データ型は導入しない
  - 記述に制限は加える
- プログラムカウンタをステートマシンに置換
  - まずは，１文１ステート
  - forやwhileに合わせたステートマシン生成
- メソッド呼び出し相当のHDLコード生成

# JavaRockの設計方針

- “HDLで書けることをJavaで書けるようにする”ではない
- “Javaで書けることを全部HDLにする”ではない

Java(のサブセット）を実行可能なHDLコード規約
あるいは
フェッチのないJavaプロセッサの生成

# コードの例

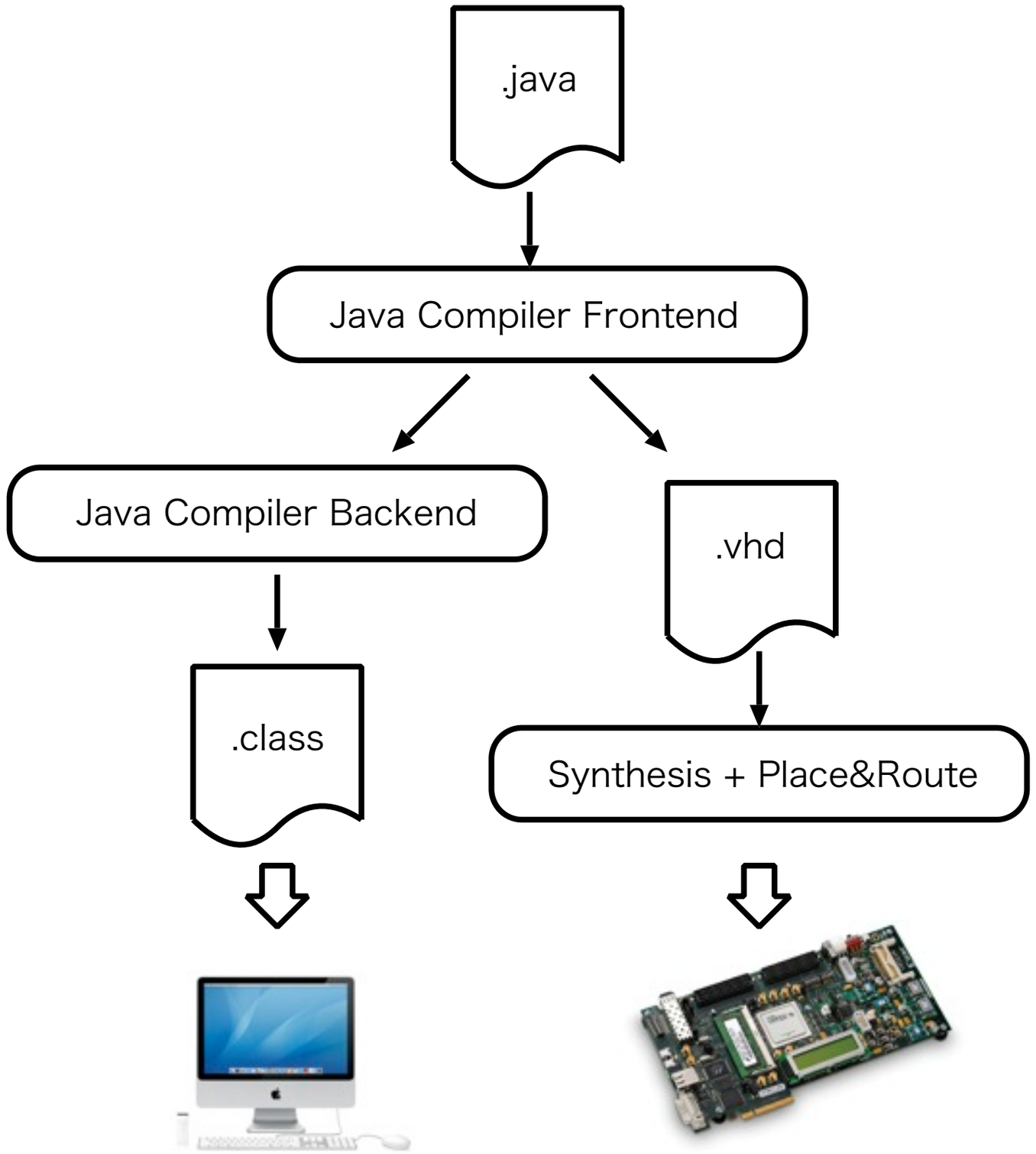
Hoge.java

<-> [X] 0+ Hoge.java

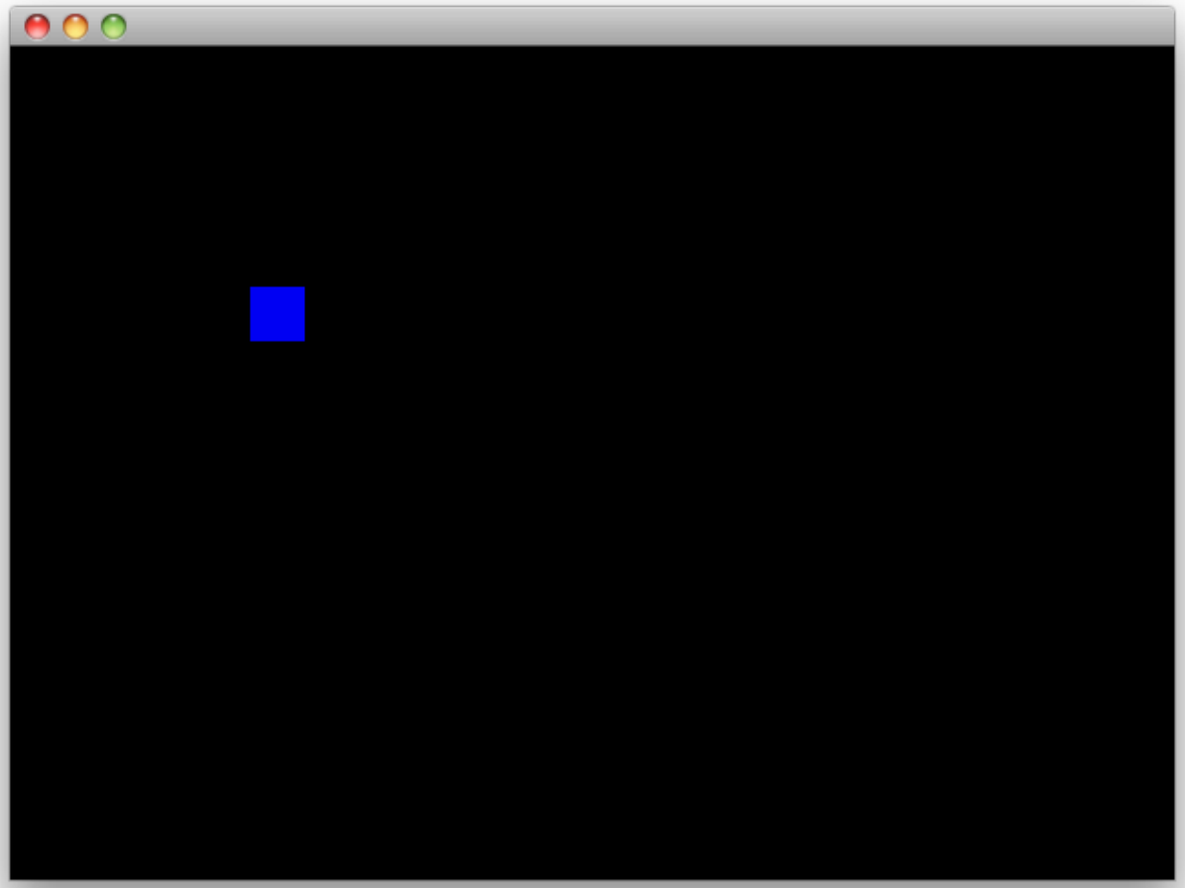
```
public class Hoge extends Thread{

  private int counter;

  public boolean isFlag(){
    if(counter > 1000000){
      return true;
    }else{
      return false;
    }
  }

  public void run(){
    while(true){
      counter++;
    }
  }

}
```

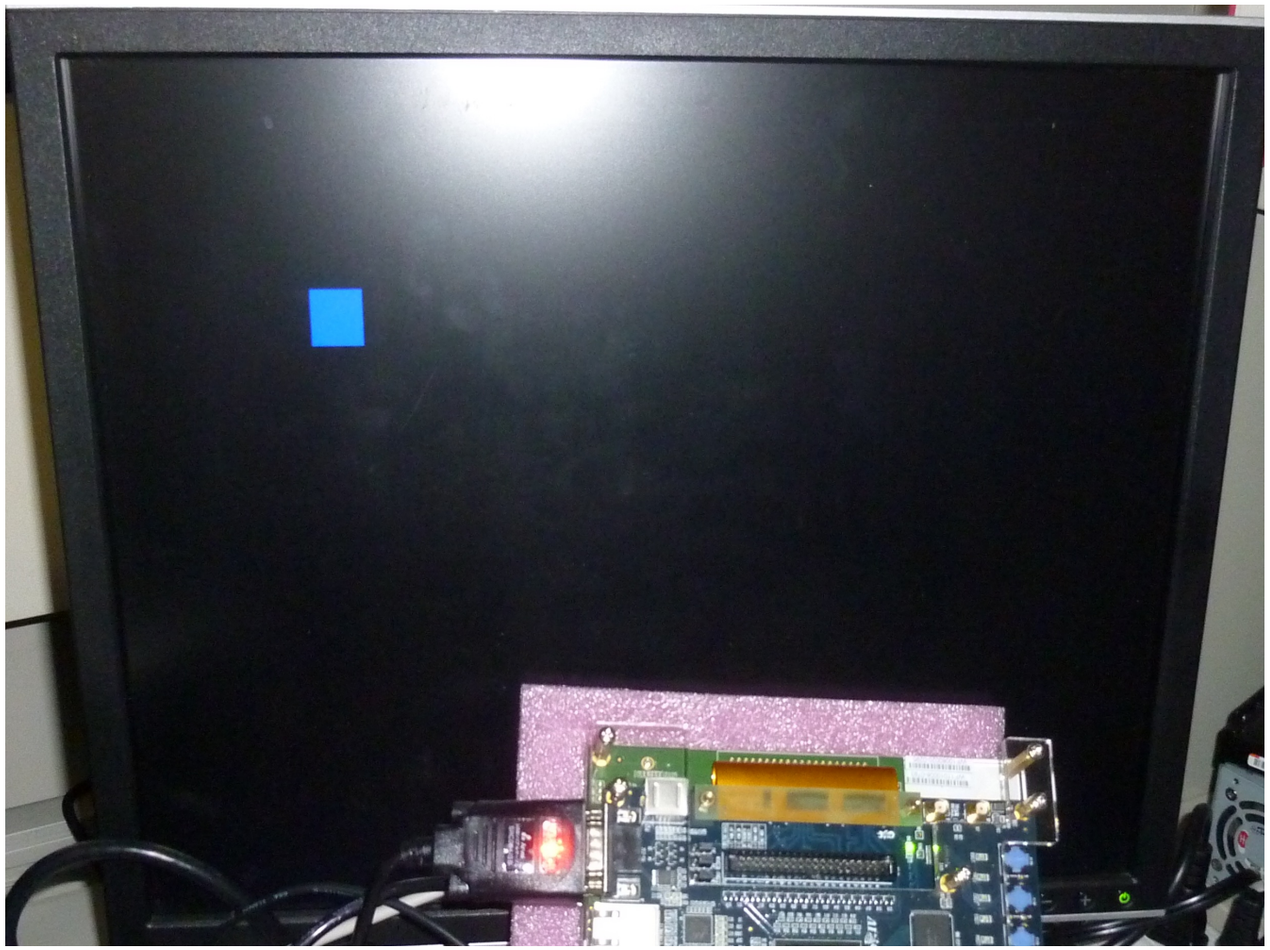
-U:---  Hoge.java  All (1,0)  [0]  (Java/l Abbrev)--2:55PM 0.39----------

# コンパイルフロー


.java
Java Compiler Frontend
Java Compiler Backend
.vhd
.class
Synthesis + Place&Route

# ソフトウェアとハードウェアで同じ動作

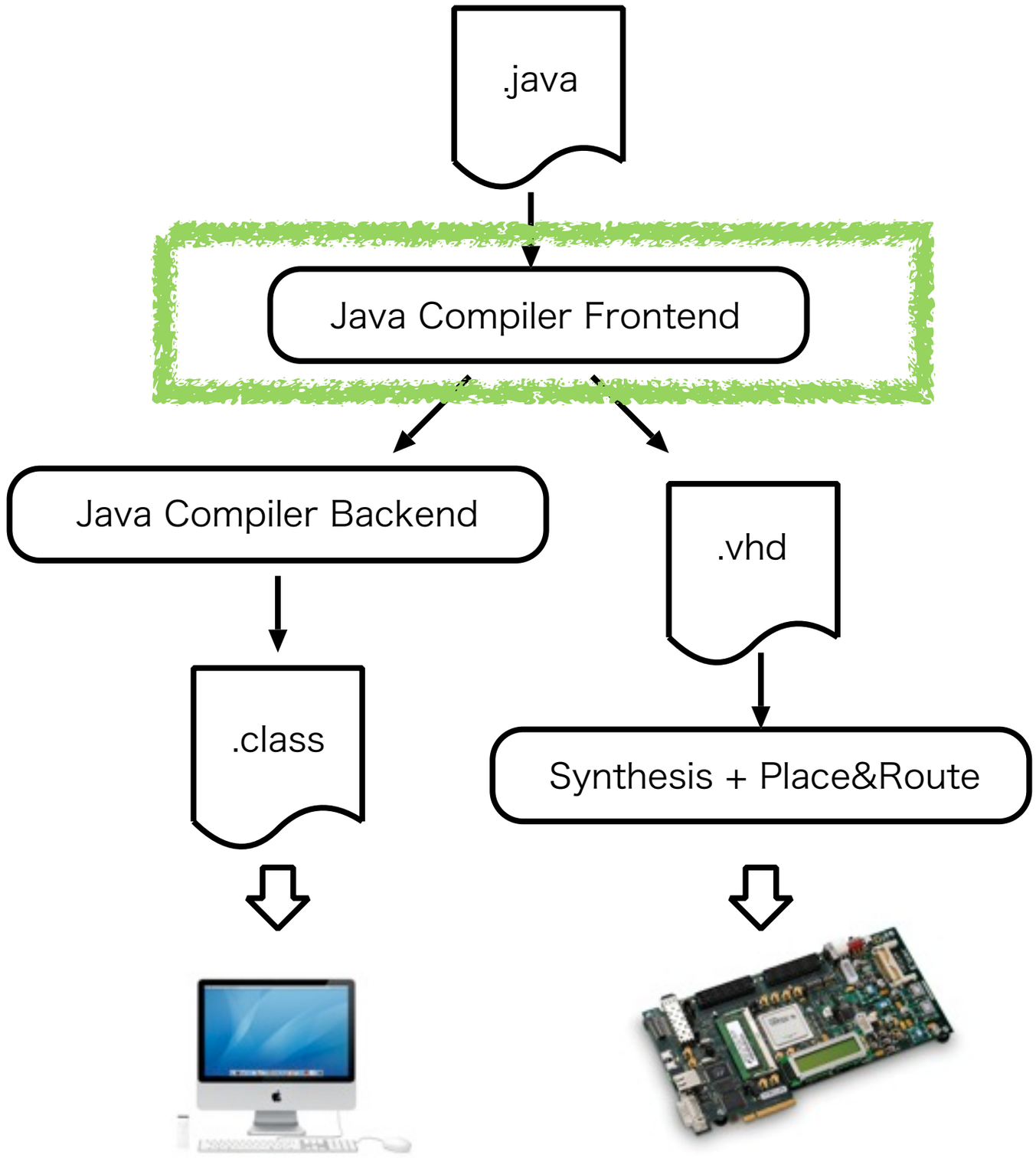




どうやって

JavaをHDLにするか

# ソースコードレベルでHDLに変換



# Javaのソースコード解析

OpenJDK(http://openjdk.java.net)のjavacを活用

コンパイラ本体: com.sun.tools.javac.main.JavaCompiler.java

⇐ コンパイル処理の途中にフックを追加

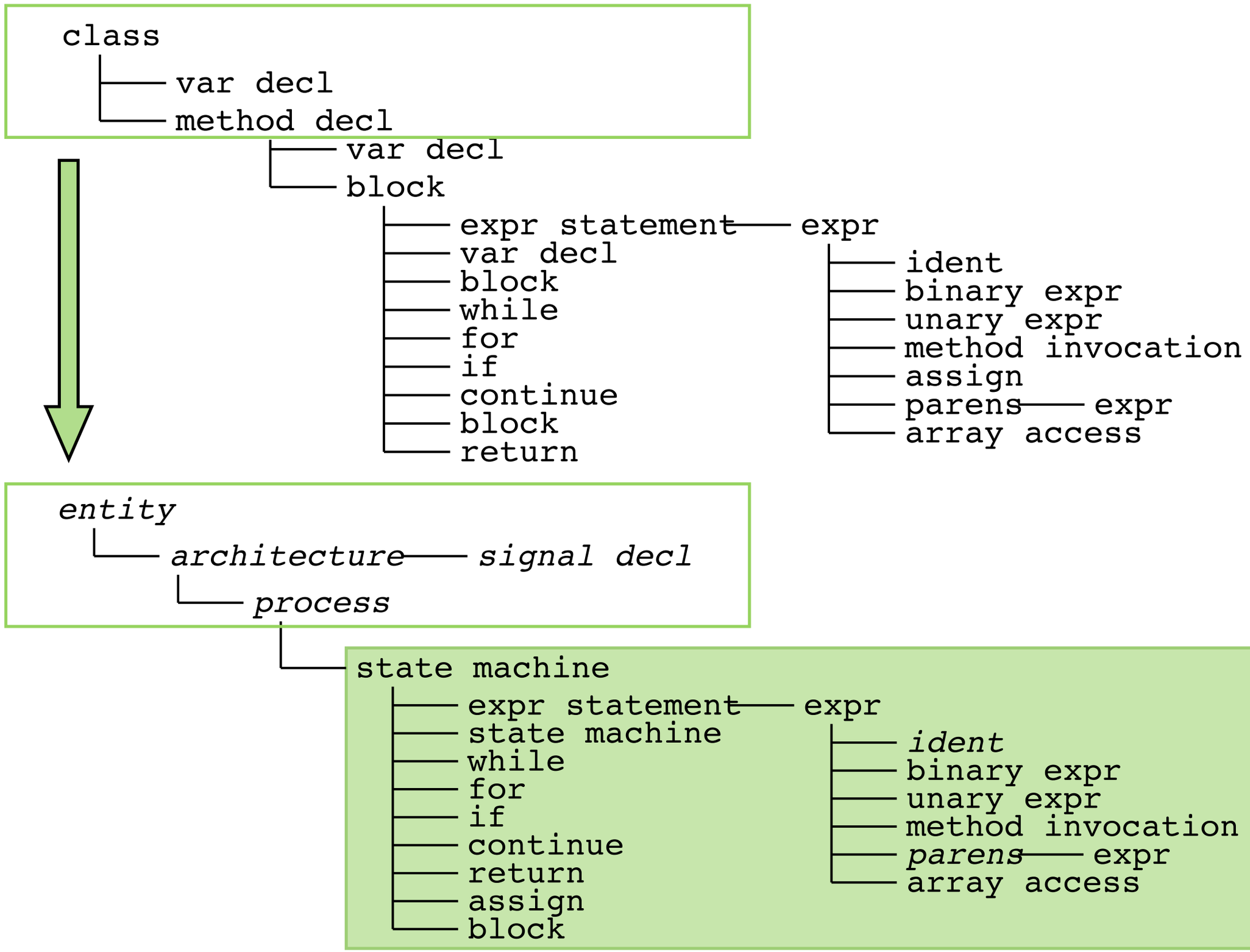
```
genCode(Env<AttrContext> env, JCClassDecl cdef) ～
```

- 与えられたソースコードをJavaRockツリーに変換

```
compile(List<JavaFileObject> sourceFileObjects,
        List<String> classnames,
        Iterable<? extends Processor> processors)～
```

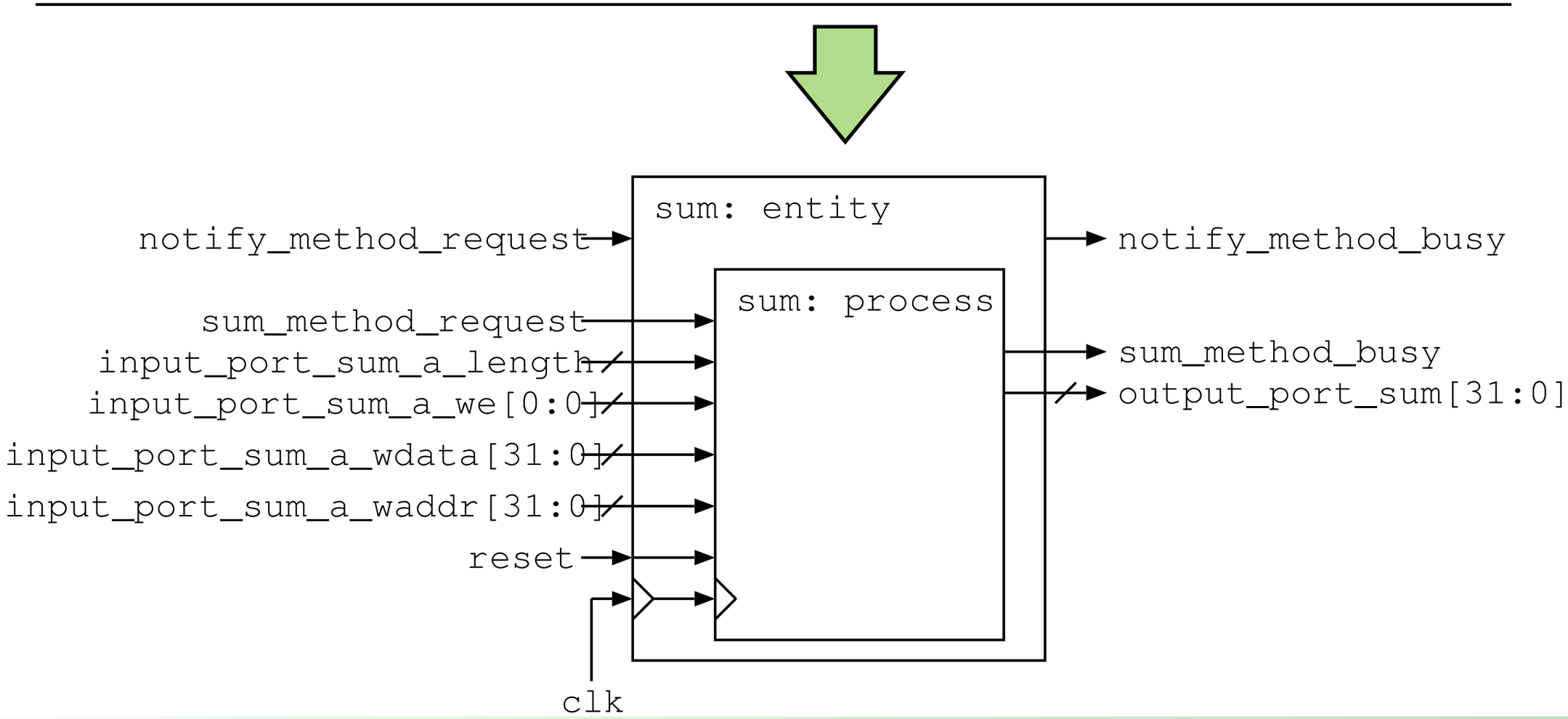
- クラス間の関係の解析，モジュールの結合を決定
- 最適化
- コード生成

# Java構文木をHDL構文木に変換



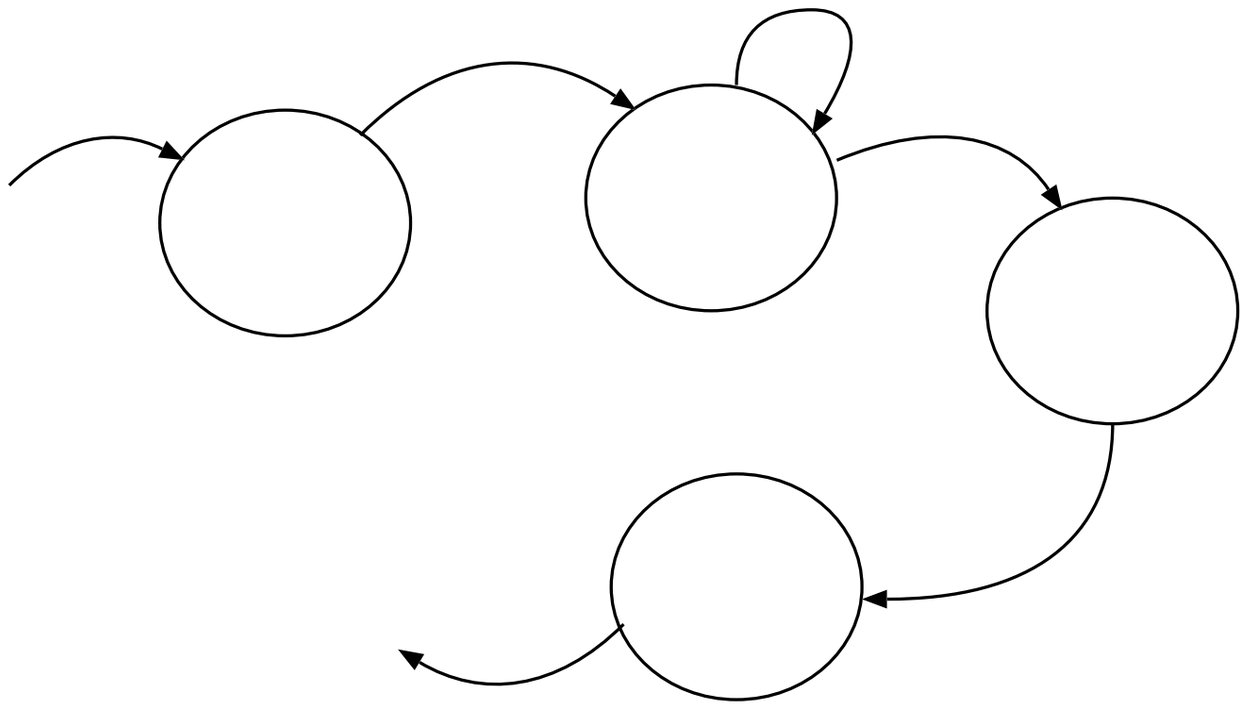
# 生成する基本モジュール

```
1: public class sum{
2:  public int sum(int[] a){
3:    int sum = 0;
4:    for(int i = 0; i < a.length; i++){
5:      sum += a[i];
6:    }
7:    return sum;
8:  }
9: }
```

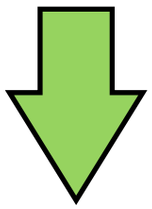


# コード生成規約: 基本的なストラテジ

- Javaの1文を1ステートとするステートマシン
- ブロック文はさらにステートマシンを分割
  - if, while, for, {}
- 配列アクセスはBRAMアクセスに分解



```
1: public class sum{
2:  public int sum(int[] a){
3:    int sum = 0;
4:    for(int i = 0; i < a.length; i++){
5:      sum += a[i];
6:    }
7:    return sum;
8:  }
9: }
```

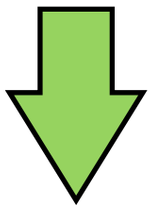


ステート変数

```
1: case conv_integer(sum_method_state) is
2:   ...
3:  when 2 =>
4:    sum_0 <= conv_std_logic_vector(0, 31-0+1);
5:    sum_method_state <= sum_method_state + 1;
6:  when 3 =>
7:   ...
```

上記のJavaコードに相当するステート

```
1: public class sum{
2:  public int sum(int[] a){
3:    int sum = 0;
4:    for(int i = 0; i < a.length; i++){
5:      sum += a[i];
6:    }
7:    return sum;
8:  }
9: }
```



長くなるので次のスライドへ

# for(int i = 0; i < a.length; i++){ }

```
 1: case conv_integer(sum_method_state) is
 2:   ...
 3:    when 3 =>
 4:      case conv_integer(state_counter_sum_1) is
 5:        when 0 =>
 6:          i_1 <= conv_std_logic_vector(0, 31-0+1);
 7:          state_counter_sum_1 <= state_counter_sum_1 + 1;
 8:        when 1 =>
 9:          if (conv_integer(i_1) < input_port_sum_a_length) then
10:            state_counter_sum_1 <= state_counter_sum_1 + 1;
11:          else
12:            sum_method_state <= sum_method_state + 1;
13:            state_counter_sum_1 <= (others => '0');
14:          end if;
15:        when 2 =>
16:             ...
17:        when 3 =>
18:            i_1 <= conv_std_logic_vector(conv_integer(i_1 + 1), 31-0+1);
19:            state_counter_sum_1 <= conv_std_logic_vector(1, 32);
20:        when others => state_counter_sum_1 <= (others => '0');
21:      end case;
22:    when 4 =>
23:   ...
```

ステート変数

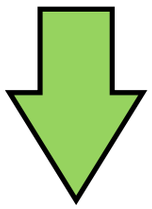
このforループのステート変数

cond式

init式

update式

```
1: public class sum{
2:  public int sum(int[] a){
3:    int sum = 0;
4:    for(int i = 0; i < a.length; i++){
5:      sum += a[i];
6:    }
7:    return sum;
8:  }
9: }
```



これも，長くなるので次のスライドへ

## sum += a[i];

```
 1: when 2 =>
 2:   case conv_integer(state_counter_sum_2) is
 3:     when 0 =>
 4:       case conv_integer(array_index_operation_state_counter_3) is
 5:         when 0 =>
 6:           param_input_port_sum_a_raddr <=
                      conv_std_logic_vector(conv_integer(i_1), 11-1-0+1);
 7:           array_index_operation_state_counter_3 <=
                            array_index_operation_state_counter_3 + 1;
 8:         when 1 =>
 9:           array_index_operation_state_counter_3 <= (others => '0');
10:           state_counter_sum_2 <= state_counter_sum_2 + 1;
11:         when others => array_index_operation_state_counter_3 <=
                                                      (others => '0');
12:       end case;
13:     when 1 =>
14:       sum_0 <= conv_std_logic_vector(
             conv_integer(sum_0 + param_input_port_sum_a_rdata), 31-0+1);
15:       state_counter_sum_1 <= state_counter_sum_1 + 1;
16:       state_counter_sum_2 <= (others => '0');
17:     when others => state_counter_sum_2 <= (others => '0');
18:   end case;
19: when 3 =>
20: ...
```

メモリアクセスのステート変数

読むアドレスを指定

データを読んでsumに加算

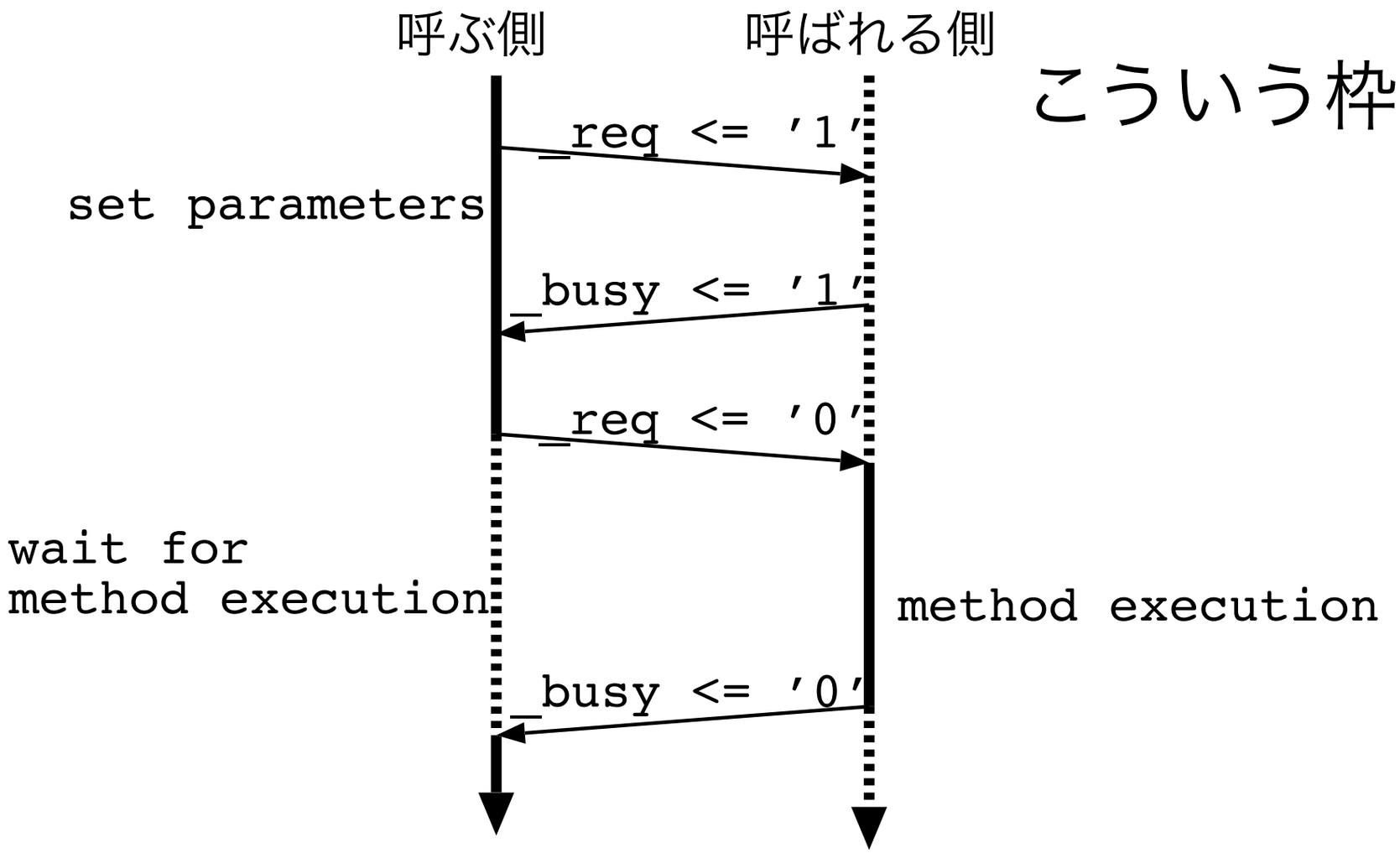
1サイクル待たないといけない

# コード生成規約: メソッド呼び出し

Java

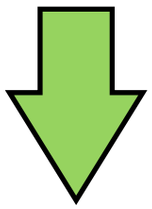
```
public class Hoge{
    // メソッド定義
    public int zero(){ return 0; }
```

```
    Hoge hoge = new Hoge();
...
        hoge.zero(); // メソッド呼出し
```




こういう枠にはめる.

```
1: public class sum{
2:   public int sum(int[] a){
3:     int sum = 0;
4:     for(int i = 0; i < a.length; i++){
5:       sum += a[i];
6:     }
7:     return sum;
8:   }
9: }
```



これも，長くなるので次のスライドへ

# int sum(int[] a){ }

```
 1: case conv_integer(sum_method_state) is
 2:   when 0 =>
 3:     if(sum_method_request = '1') then
 4:       sum_method_busy <= '1';
 5:       sum_method_state <= sum_method_state + 1;
 6:     else
 7:       sum_method_busy <= '0';
 8:     end if;
 9:   when 1 =>
10:      if(sum_method_request = '0') then
11:        sum_method_state <= sum_method_state + 1;
12:      end if;
13:   when 2 =>
14:     ...
15:   when 4 =>
16:    output_port_sum <= conv_std_logic_vector(conv_integer(sum_0), 31-0+1);
17:    sum_method_state <= (others => '0');
15:   when 5 =>
16:     sum_method_busy <= '0';
17:     sum_method_state <= (others => '0');
18:   when others => sum_method_state <= (others => '0');
19: end case;
```

メソッドが呼び出されるのを待つ

この間にパラメタを引き渡す

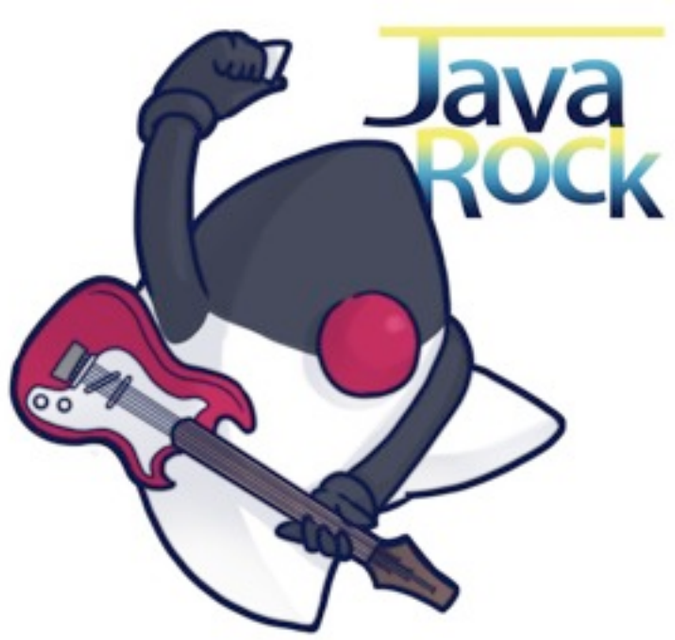
終わったら，返戻値をセットしておしまい

# JavaRockでソート

```
public class BubbleSort{

  private final int[] br = new int[512];

  public void test(int[] ar){
    for(int i = 0; i < ar.length; i++){ br[i] = ar[i]; }
    for(int i = 0; i <= (ar.length-1) - 1; i++){
      for(int j = 1; j <= ar.length - 1 - i; j++){
        int a = br[j];
        int b = br[j-1];
        if(a < b){
          br[j-1] = a;
          br[j] = b;
        }
      }
    }
  }

  public int get(int i){ return br[i]; }

}
```

# JavaRockでソート

# FPGAを活用するために



# FPGAを“活用”するために

- 並列化
  - 細粒度自動並列化
  - ユーザによるThreadレベル並列化
- HDLモジュールの利用
  - JavaRock Hardware Interface(JRHI)
  - JavaRock HDL

# 並列化手法

- 演算レベルの並列性

→基本ブロック内の自動並列化

- タスク・データレベルの並列性

→JavaのスレッドをHWにマッピング

- パイプライン並列性

→wait-notifyで記述

→JavaRock HDLでの記述

# JavaのスレッドをHWにマッピング

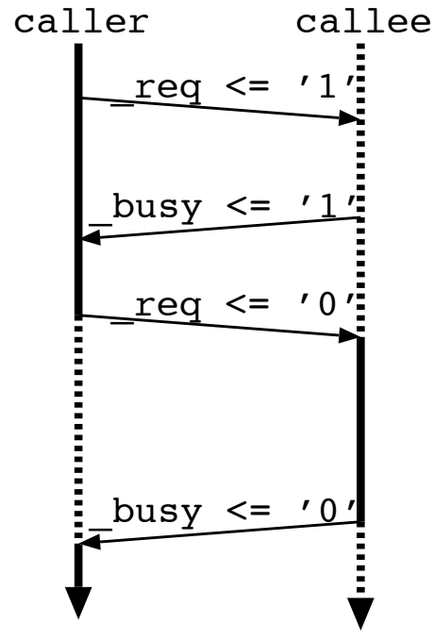
```
led obj0 = new led();
echo obj1 = new echo();
sc1602_test obj2 = new sc1602_test();

  obj0.start();
  obj1.start();
  obj2.start();
```

スレッドの生成
&
実行

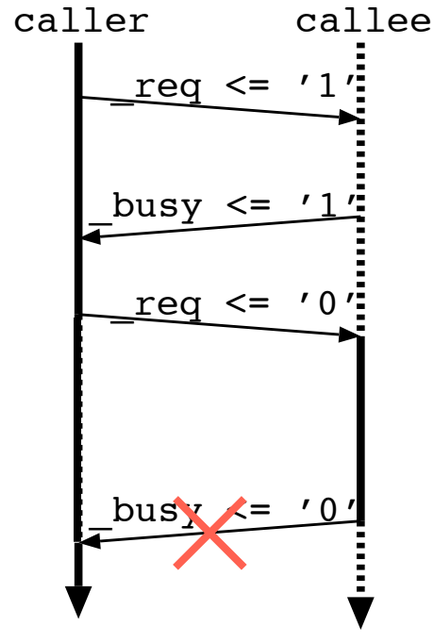
```
public class led extends Thread{
  counter c = new counter();
  boolean flag;

  public void run(){
    while(true){
      c.up();
      int v = c.read();
      if(v == 1000000){
        c.clear();
        flag = !flag;
      }
    }
  }
}
```

```
public class echo extends Thread{
  rs232c obj = new rs232c();
  byte[] data = new byte[128];

  public void run(){
    while(true){
      obj.write((byte)'>');
      int i = 0;
      byte c = 0;
      boolean flag = true;
      while(true){
        c = obj.read();
        if(c == (byte)'\n' || c == (byte)'\r'){
          break;
        }else{
          data[i] = c;
          i++;
        }
      }
      for(int j = 0; j < i; j++){
        c = data[j]
```

## 通常の関数呼び出し

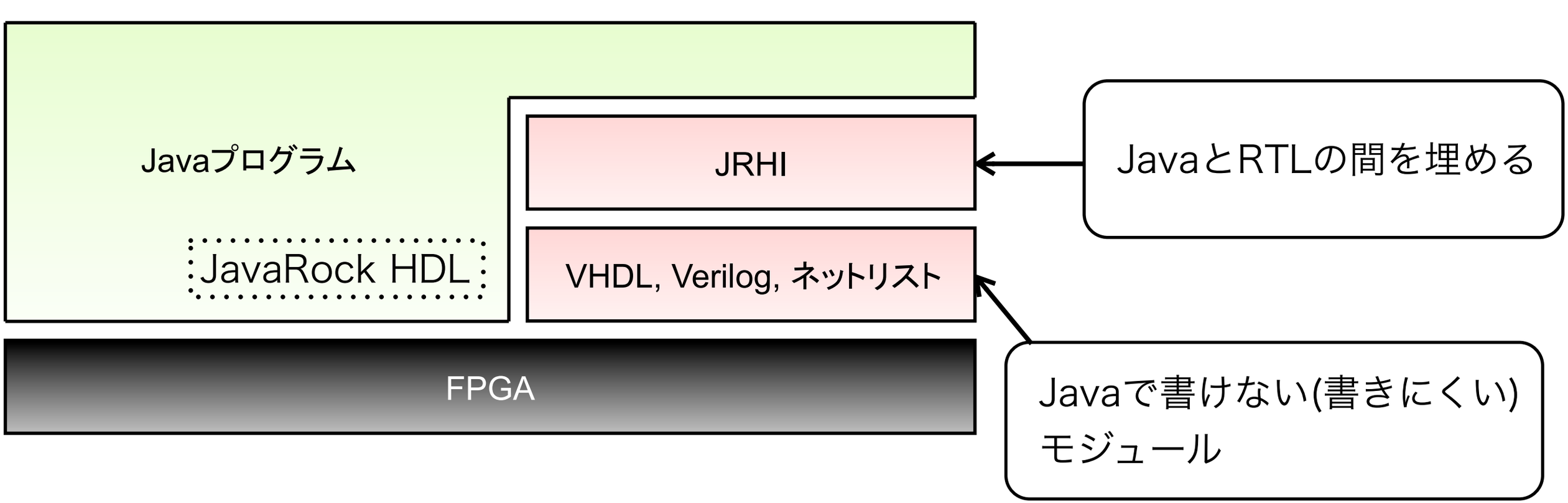



## スレッドの場合



# HDLモジュールの活用

- IPコア，手書きHDLモジュールの活用
- ハードウェアを意識したJavaコード記述支援



# JavaRock Hardware Interface

- VHDL/Verilogで記述したモジュールを使いたい
- Javaからみて自然な形でアクセスしたい
  - Javaプログラム中に変な記述をいれたくない

- VHDL/Verilogモジュールの入出力をJavaの変数
- HDL中で定義したブロックRAMを配列にみせる

# JRHIの使いどころ

- メモリ
- グラフィクスコントローラ

→制御タイミングが重要

- FPGA内部の専用HWリソース

→BlockRAM(内蔵メモリ)

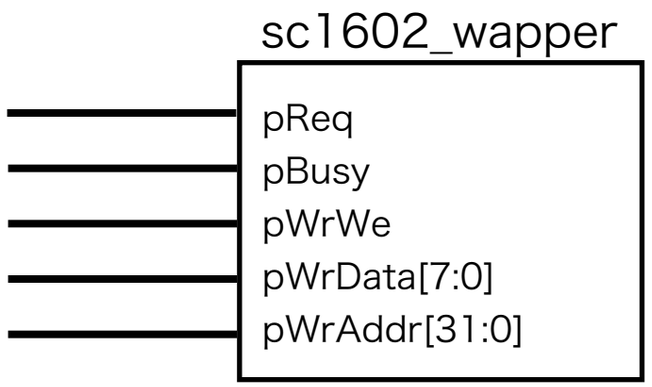
→DSP

- 既存のHDLコードの活用

# JavaRock Hardware Interface

```
entity sc1602_wrapper is
  port (
    clk : in std_logic;
    pLCD_RS : out std_logic;
    pLCD_E : out std_logic;
    pLCD_DB : out std_logic_vector(3 downto 0);
    pLCD_RW : out std_logic;

    pReq : in std_logic;
    pBusy : out std_logic;
    pWrData : in std_logic_vector(7 downto 0);
    pWrAddr : in std_logic_vector(31 downto 0);
    pWrWe : in std_logic;

    reset : in std_logic;
  );
end sc1602_wrapper;
```

使いたいHDLモジュール




SC1602Wrapper
pReq: boolean
pBusy: boolean
pWrWe: boolean
pWrData: byte
pWrAddr: int

```
public class SC1602Wrapper extends VHDLSimpleLibrary implements JavaRockComponentIface{

  public boolean pReq;
  public boolean pBusy;
  public boolean pWrWe;
  public byte pWrData;
  public int pWrAddr;

  public SC1602Wrapper(String... args){
    super("sc1602_wrapper", args);
```

JRHIクラス

# JRHIの利用

```
public class SC1602Wrapper extends VHDLSimpleLibrary implements JavaRockComponentIface{

  public boolean pReq;
  public boolean pBusy;
  public boolean pWrWe;
  public byte pWrData;
  public int pWrAddr;

  public SC1602Wrapper(String... args){
    super(“sc1602_wrapper”, args);
```

用意したJRHIクラス

```
public class SC1602_USER{
    private final SC1602Wrapper obj = new SC1602Wrapper();

    public void put(){
        obj.pWrData = (byte)’a’;
        obj.pWrAddr = 0;
        obj.pWrWr = true;
        obj.pWrWr = false;
        obj.pReq = true;
        obj.pReq = false;
    }
```

- Javaプログラムとしてコンパイルが可能
- JavaRockでHDLなsc1602_wrapperと
- 組み合わせる

使う側

# もう一歩すすんだJRHIの活用

```
public class MemoryDevice extends ～{
    public byte[] byte_data;
    public int[] int_data;

    public MemoryDevice(String... args){
        super(“memory”, args);
```

JRHIクラス

```
public class MemoryDevice_USER{
    private final MemoryDevice obj = new MemoryDevice();

public void put(){
    obj.byte_data[0] = (byte)100;
    obj.int_data[0] = 1000;
}
```

使う側

ヒープメモリの一部をHDLモジュール内の
メモリに見せるというような使い方が可能

# JavaRock HDL

- アノテーションによるJavaのHDL化

test.java

<-> [X] 0+ test.java

```
import net.wasamon.javarock.rt.*;

@javarockhdl
public class test{
  led obj0 = new led();
  echo obj1 = new echo();
  sc1602_test obj2 = new sc1602_test();
  //VGAJavaTest vga = new VGAJavaTest();

  @combination
  public boolean led(){
    return obj0.led();
  }

  @auto
  public void main(){
    obj0.start();
    obj1.start();
    obj2.start();
    //vga.start();
  }
}
```

--:---  test.java  All (7,0)  [0] SVN-128  [(Java/l Abbrev)]--8:09AM 0.49----------

# アノテーション

| アノテーション | 概要 | 対象 |
|---|---|---|
| @javarockhdl | このクラスがJavaRockHDLであることを示す | クラス |
| @raw | メソッド内のifやwhileの条件判定式をHDLに直接変換 | メソッド |
| @auto | メソッドが呼び出されることなく動作するようにする | メソッド |
| @width | 変数のbit幅を定義する | 変数 |
| @no_wait | メソッド終了を待たずメソッド呼出し元の処理を進める | メソッド |

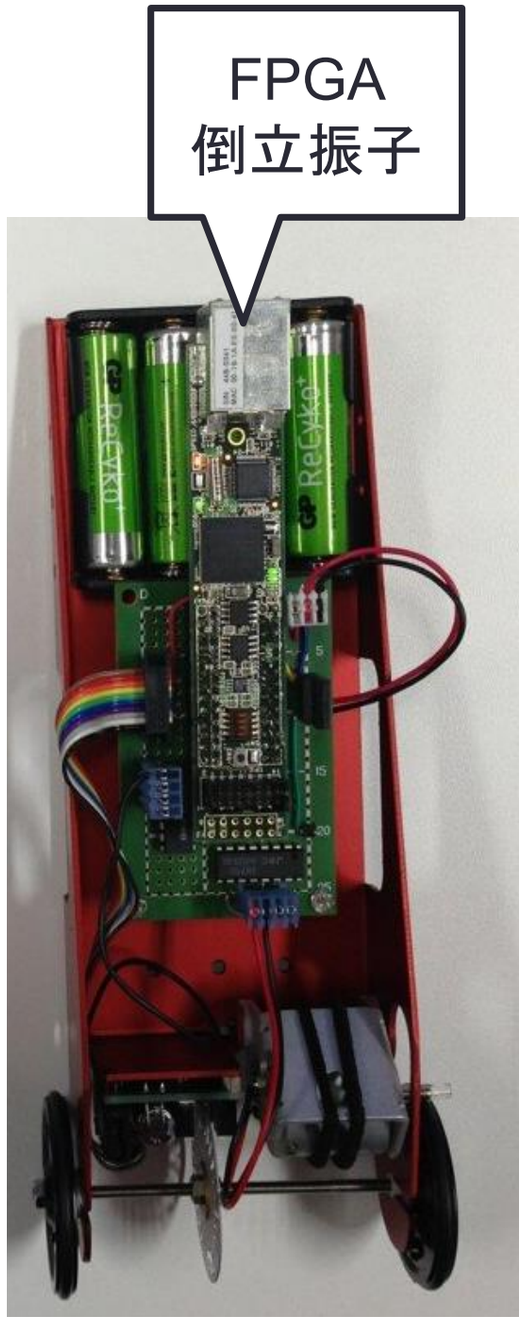
# JavaRockの制限

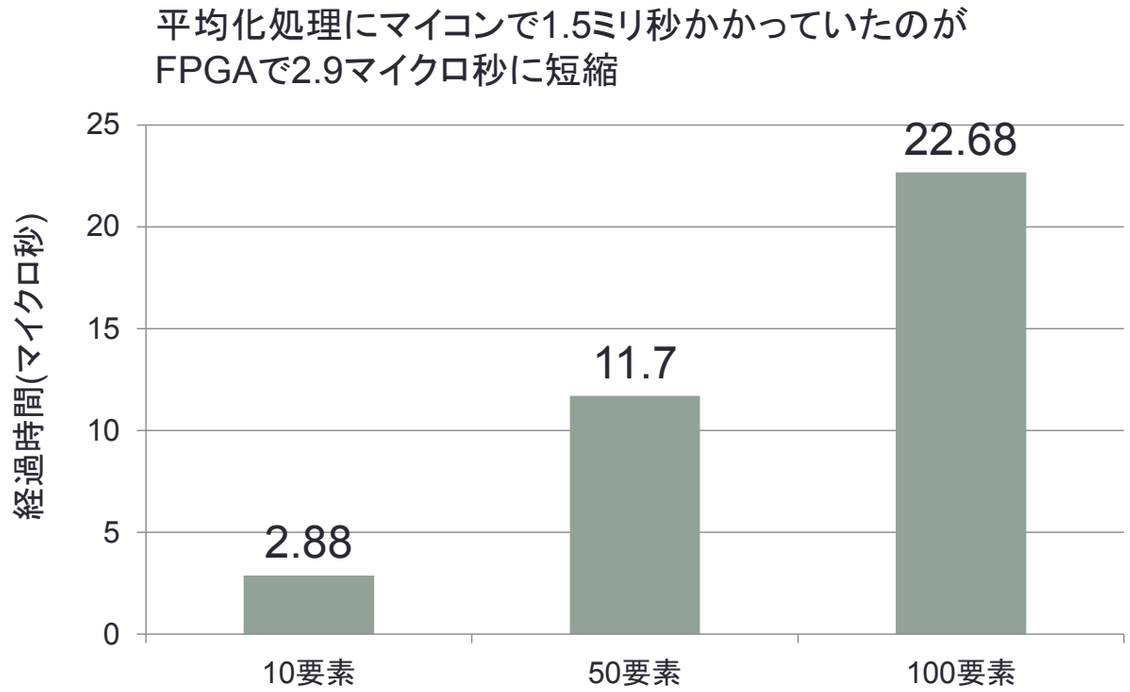
- インスタンスはファイナルでの生成のみ
  - スタティックなHWモジュールとして合成
- インスタンスの共有は不可
  - 管理用の信号のアービトレーション未実装
- 浮動小数点数演算は不可
- 配列はプリミティブ型(除float/double)のみ
- /とか%とか未サポート

# 適用事例 1/4

## 倒立振子 (by 宇都宮大学 大川先生)

- 制御分野にFPGAを導入したい
- 電気自動車の制御 < 50μ秒
- 3.8m秒@H8 → 6.3μ秒@FPGA




植竹 他 "倒立振子ロボットのFPGAを用いた超高速制御"
情報処理学会第75回全国大会

# 適用事例 2/4

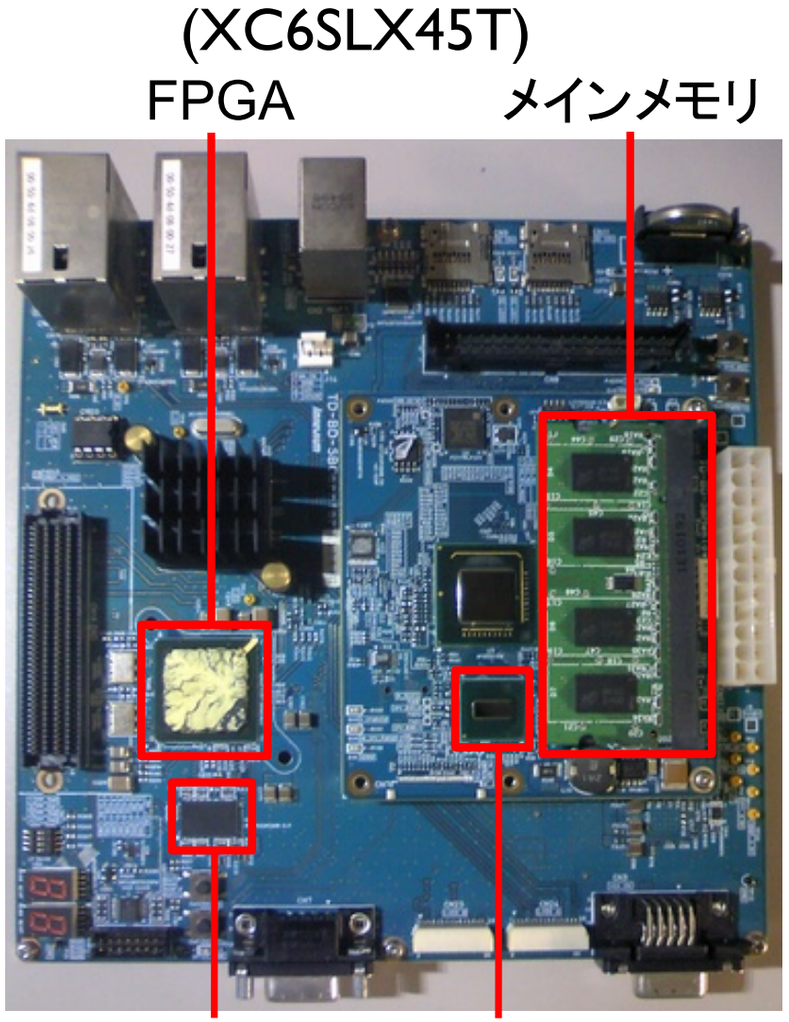
## Reconfigurable Android (by 農工大 中條研)

- SHA-1

| | 実行時間 [ms] |
|---|---|
| Java による記述のみ | 3360 |
| Andoird NDK を使用 | 51.7 |
| FPGA 使用（手書き） | 41.3 |
| FPGA 使用（JavaRock） | 209 |

- エッジ検出(3x3ラプラシアンフィルタ)

| | 実行時間 [ms] |
|---|---|
| Java による記述のみ | 392 |
| Andoird NDK を使用 | 8.85 |
| FPGA 使用（手書き） | 8.83 |
| FPGA 使用（JavaRock） | 105 |



# 適用事例 3/4

## 文字列処理

- ハードウェアシステムのコンソールなど

```
private boolean parseIP(int addr, int offset){
  int s, e, v;
  s = offset;
  byte sep = '.';
  for(int i = 0; i < 4; i++){
    if(i == 3){ sep = (byte)0; }
    e = str.match(s, sep, true);
    v = str.atoi(s, e, 10);
    if(v == -1){ return false; }
    value[i] = v;
    s = e + 1;
  }
  return true;
}
```

“192.168.10.1” → 0xc0, 0xa8, 0x0a, 0x01

```
public int match(int offset, byte ch, boolean flag){
  for(int i = offset; i < arg.length; i++){
    if((flag == true && arg[i] == ch) || (flag == false && arg[i] != ch)){
      return i;
    }
  }
  return -1;
}
```

# 適用事例 4/4

## ORB Engine with JavaRock

- Networked FPGA system based on ORB

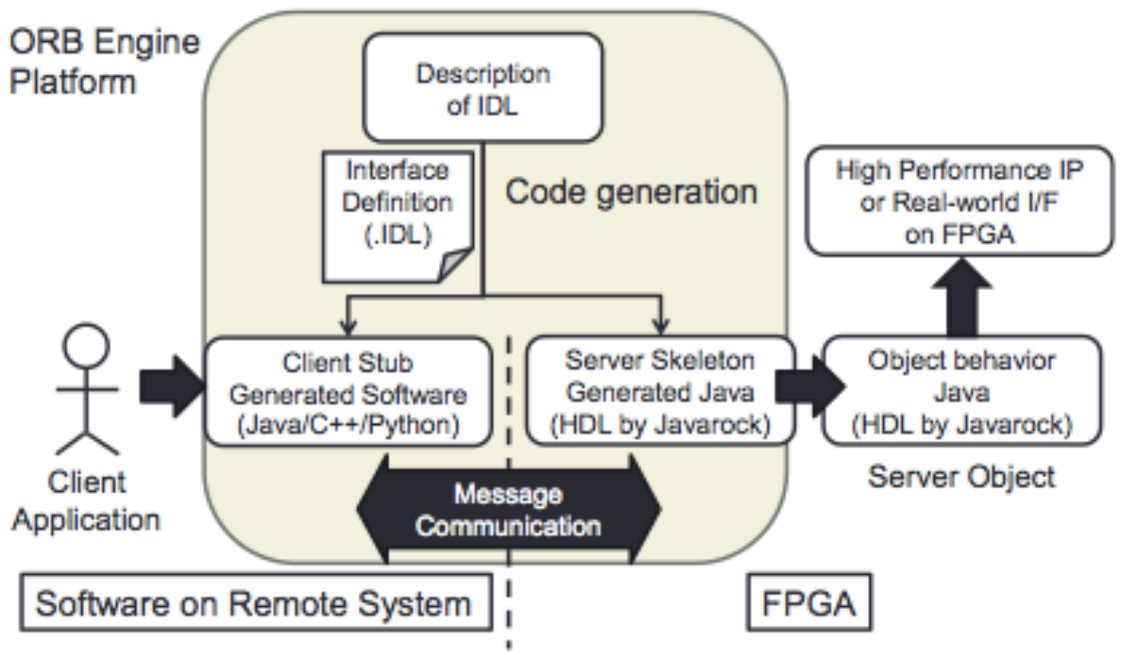



**Figure 3 ORB Engine design flow including the communication code generation from IDL file**

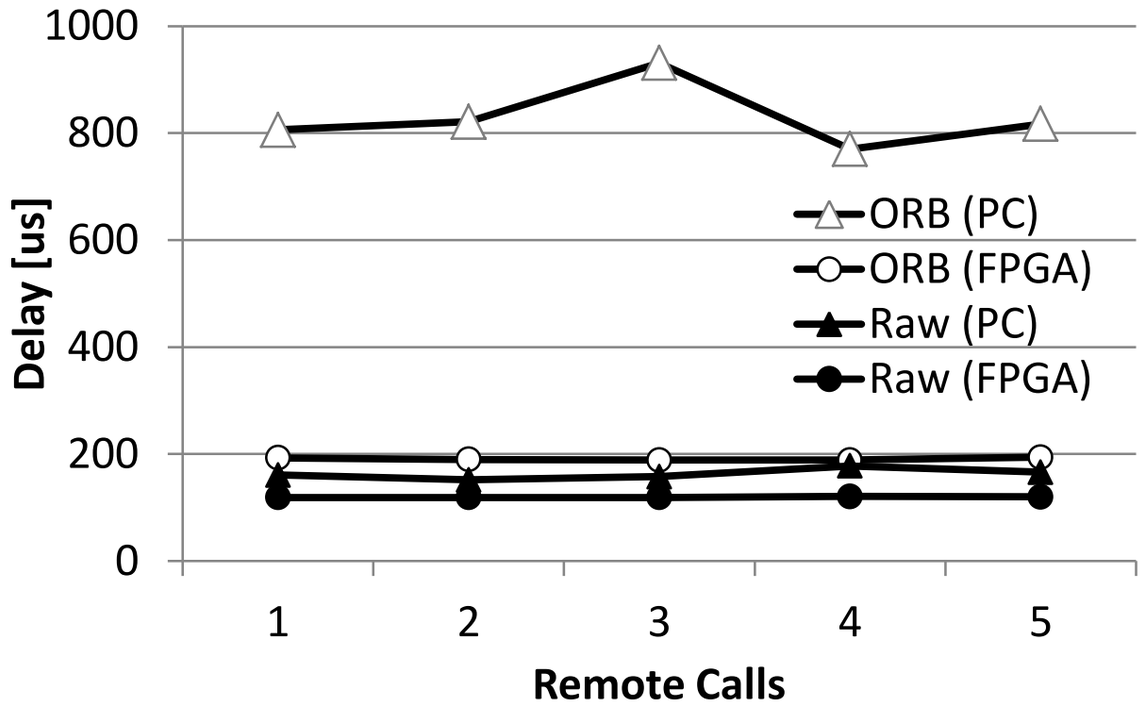
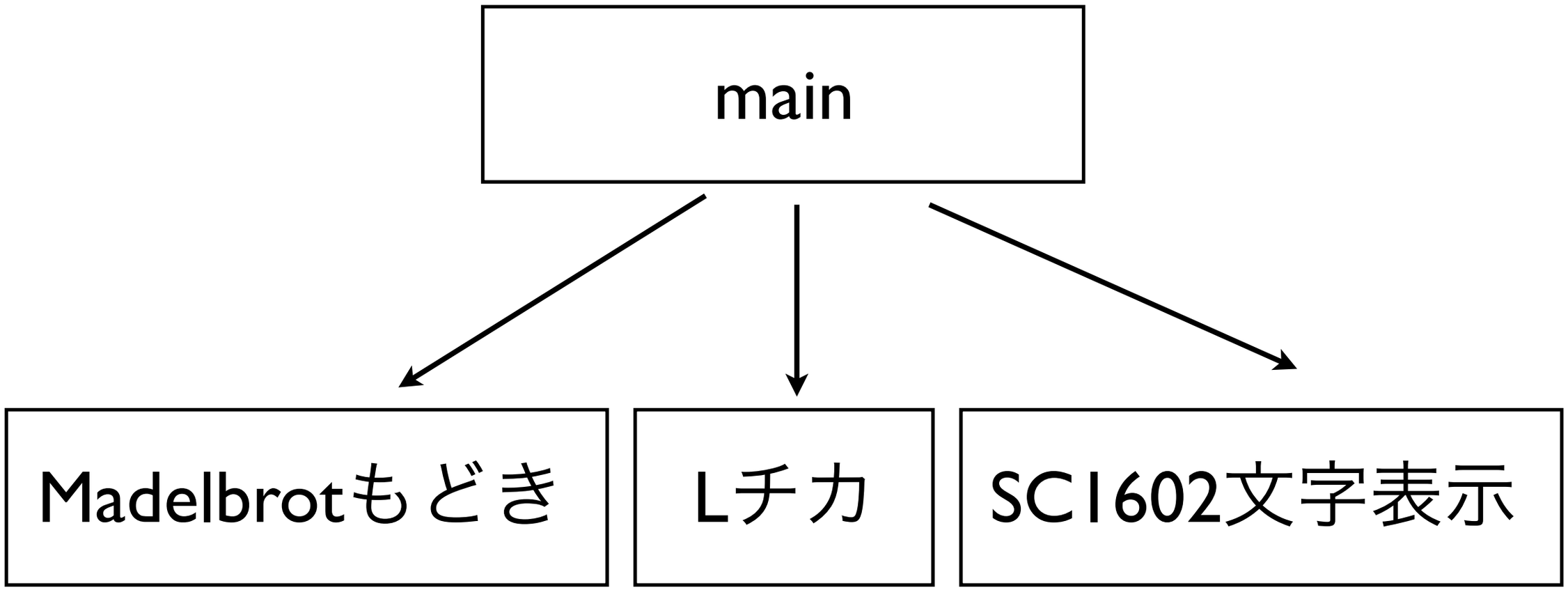



**Figure 11 Measurement performance of the synthesized ORB Protocol Engine compared to RAW protocol system**

T. Ohkawa et al., "Reconfigurable and Hardwired ORB Engine on FPGA by Java-to-HDL Synthesizer for Realtime Application," Proc. 4th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART 2013), June 2013.

# デモ


main
Madelbrotもどき
Lチカ
SC1602文字表示

# まとめ

- 高位合成言語/処理系を紹介しました
- JavaRockのコード生成を紹介しました
- JavaRockでの事例を紹介しました

# 少し先のJavaRockについて

- もっとJavaらしいプログラムのHDL化
  - オブジェクトの動的生成
  - インスタンスチェインの取り扱い
- JavaRock-Thrash成果の取り込み
  - 浮動小数点数演算
  - 使用する演算リソースを設定できるように
  - レジスタ共有，ループアンローリング
  - 演算のチェインニング