JavaRockを用いたHW/SW協調設計の検討

三好 健文　船田 悟史

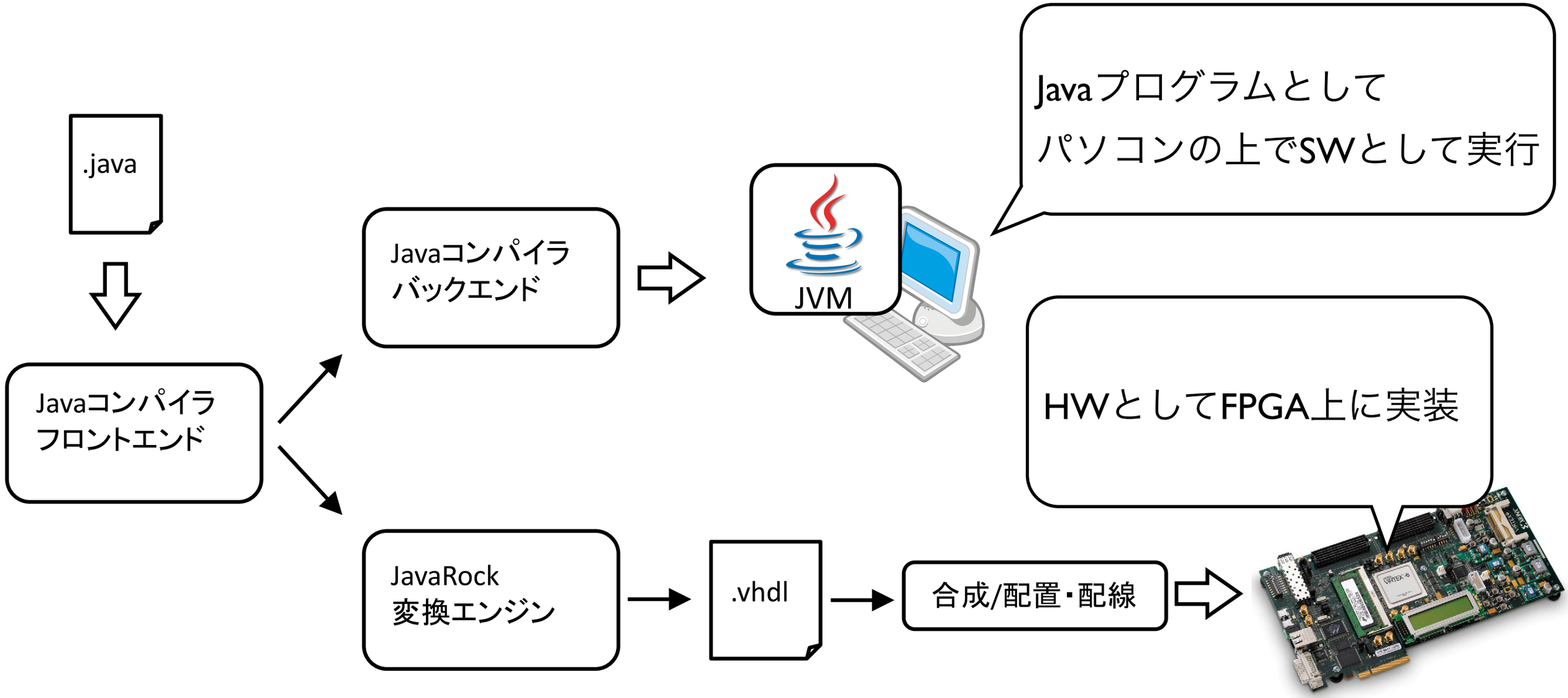
株式会社イーツリーズ・ジャパン

JavaRockとは?

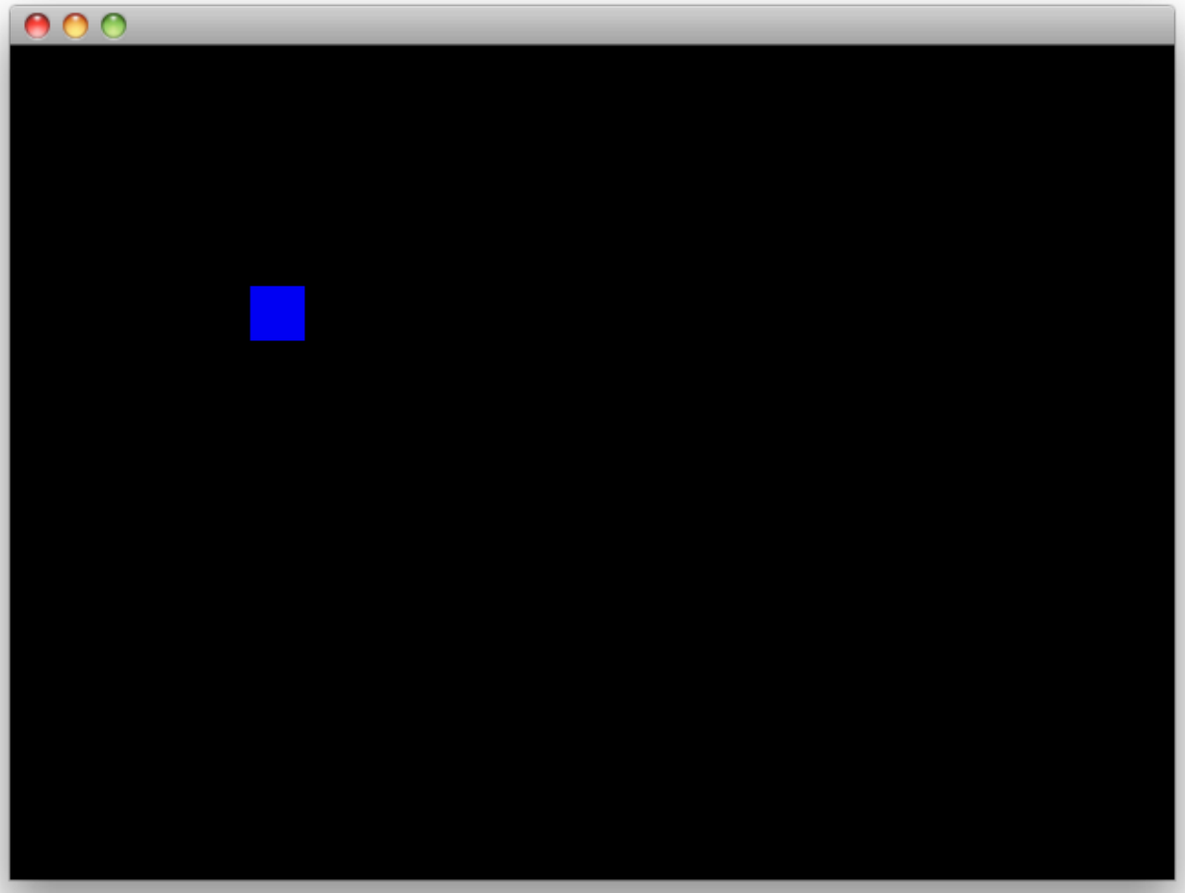
- Javaをベースとした高位合成処理系
  - 特別な文法，型は導入しない素のJavaプログラムをVHDLに変換
  - 書いたコードはSWとしてもHWとしても実行可能
- Threadやwait-notifyを並列性にマッピング
  - 粗粒度の並列性をHWでもそのまま活用
  - 細粒度の並列性はコンパイラで頑張る

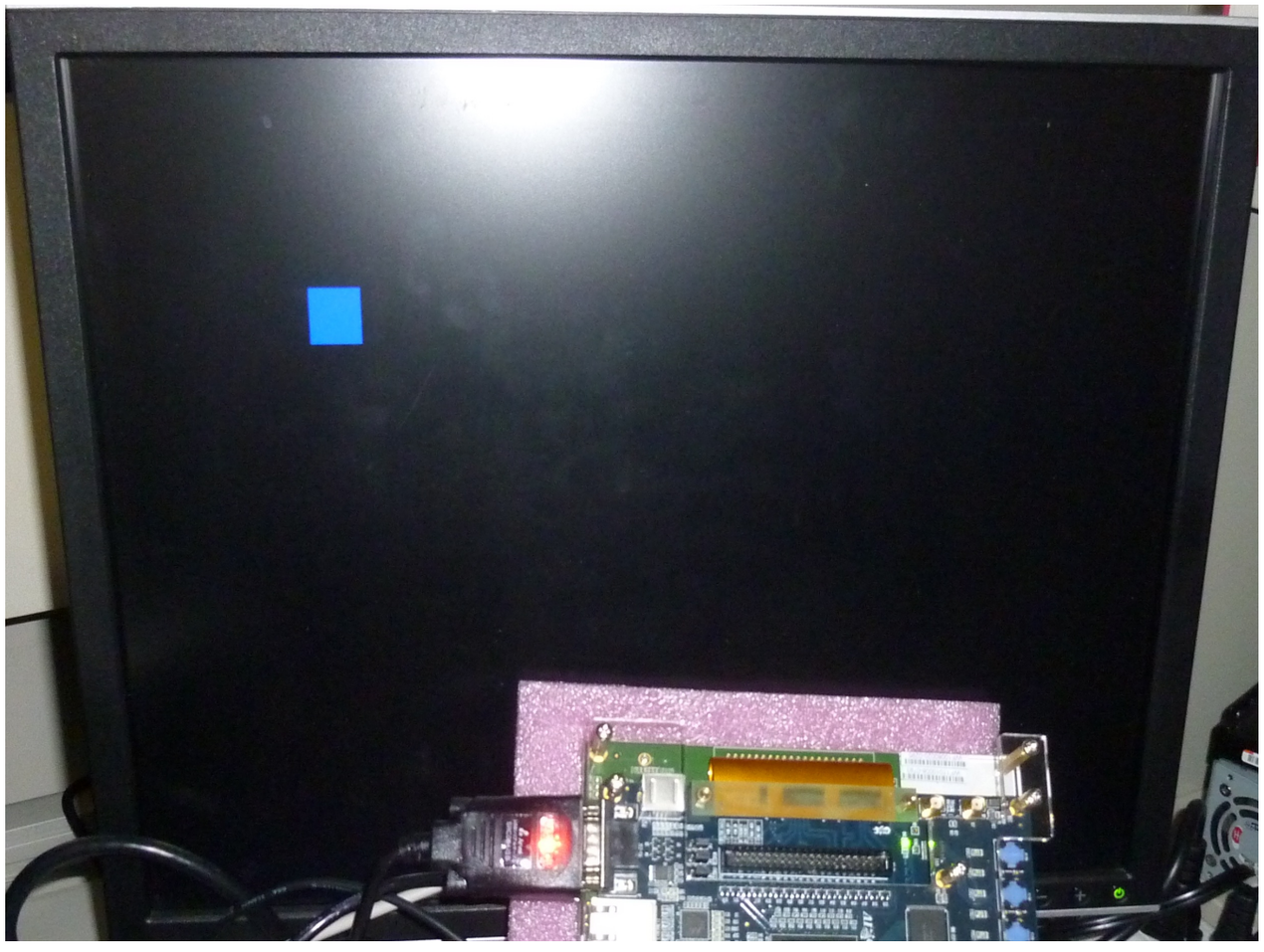
JavaRockのコンパイルフロー

同一ソースコード(Java)でSWとHWを開発可能

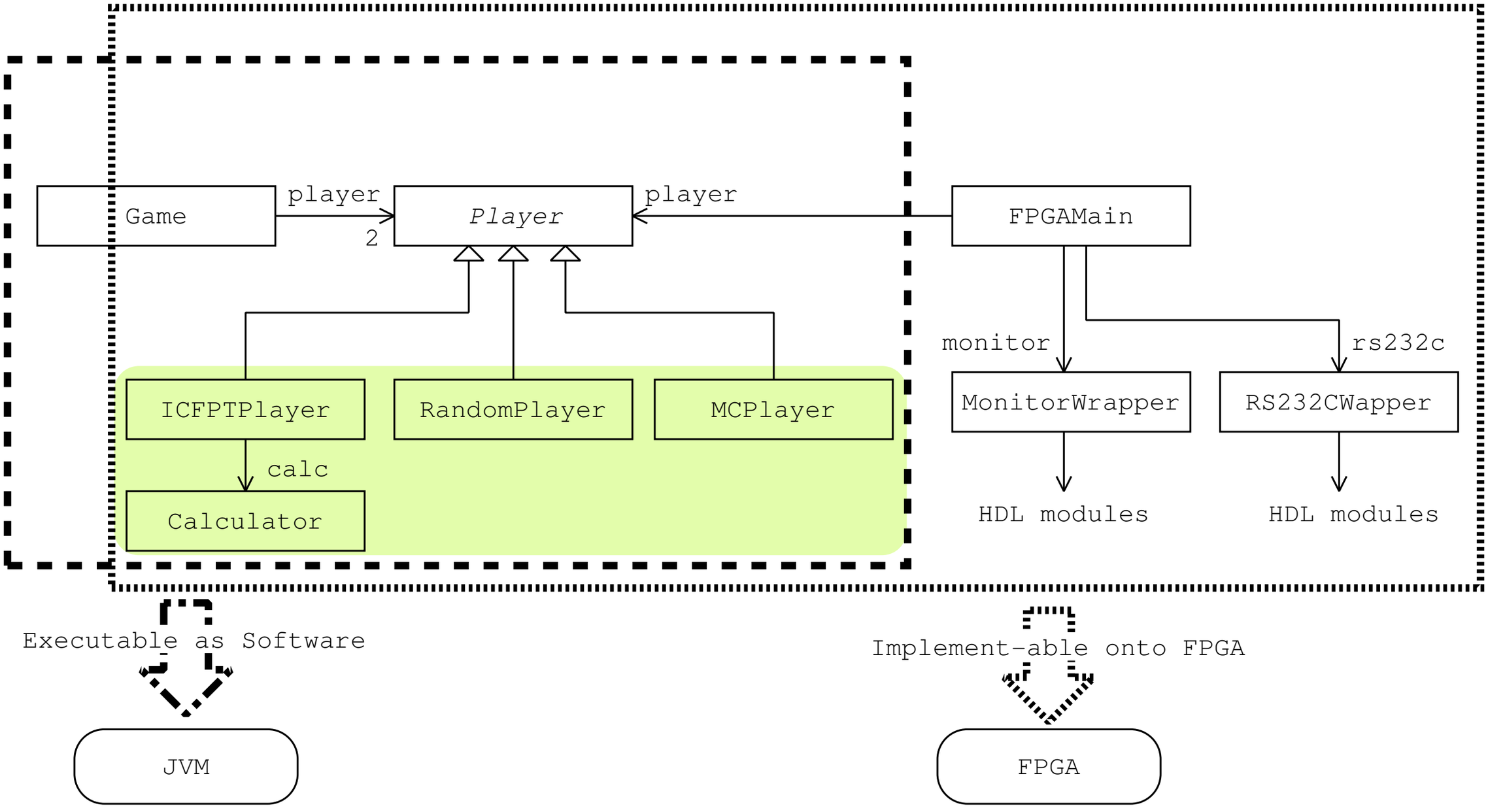


ex. SWとHWで機能レベルで同じ動作





ex. SWとHWでソースコードを共有

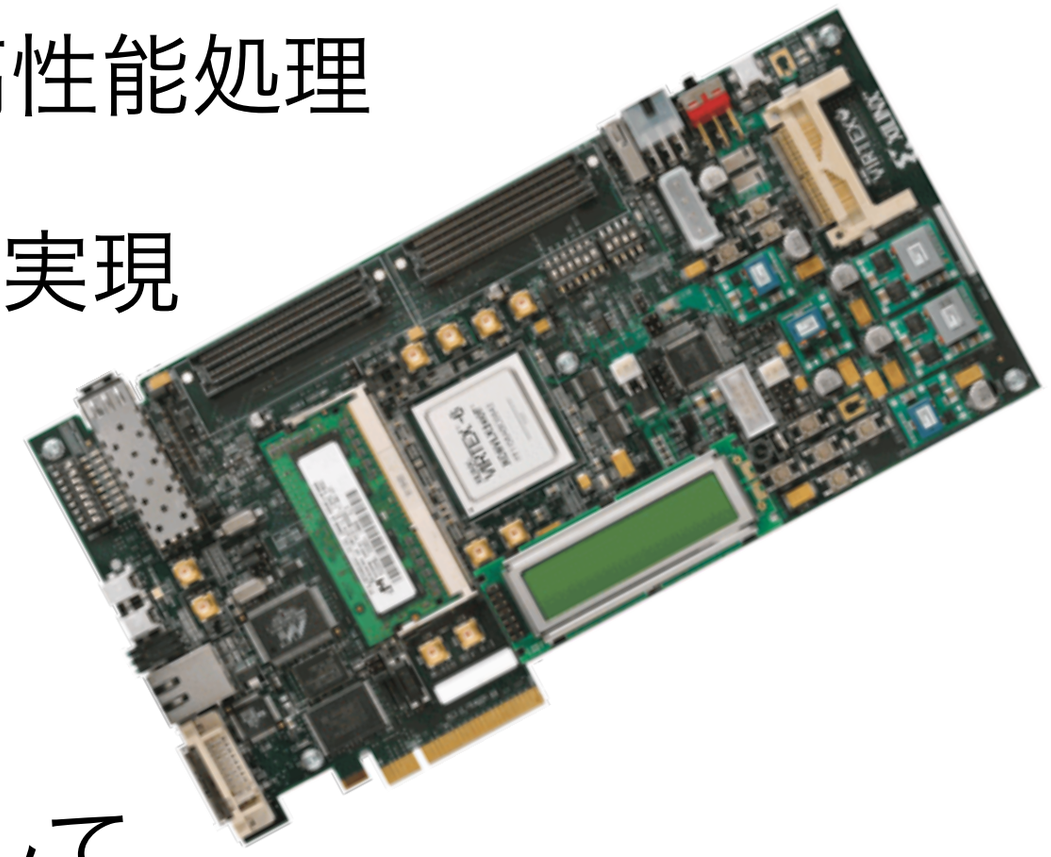


今回の発表の流れ

- FPGA・CPU混在環境で効率良いシステムを開発したい（= HW/SW協調設計したい）
  - 効率良い = 小さく，低消費電力で高性能．くわえて機能検証，バグフィクス，機能追加がしやすい．
- JavaRockを使えば「書く」のは困らなそう
- うまくHWとSWの連携処理が実現できるか？
  - HW部とSW部をがうまく通信できるかが鍵

FPGAとは？

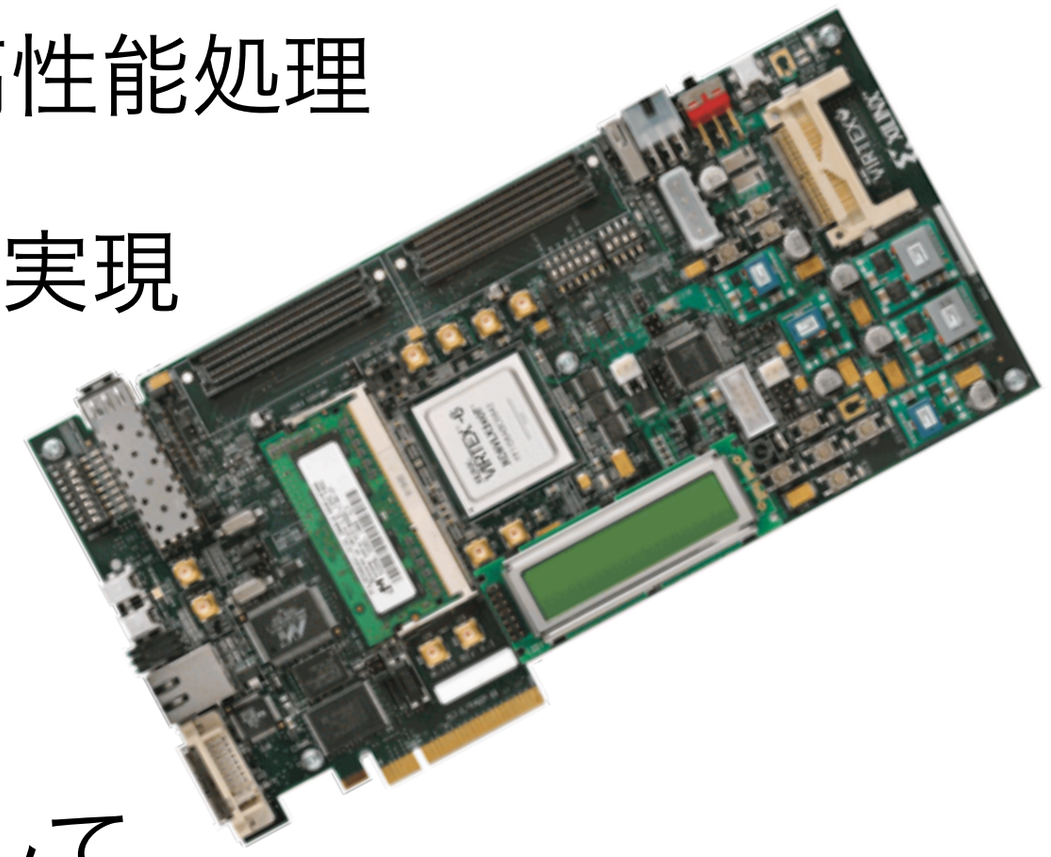
- 独自の回路を実現できるハードウェア
- 特定の処理を低消費電力で高性能処理
- デバイスに近い処理を簡単に実現
  - 自由なI/Oポートの定義



- 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とは？

- 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

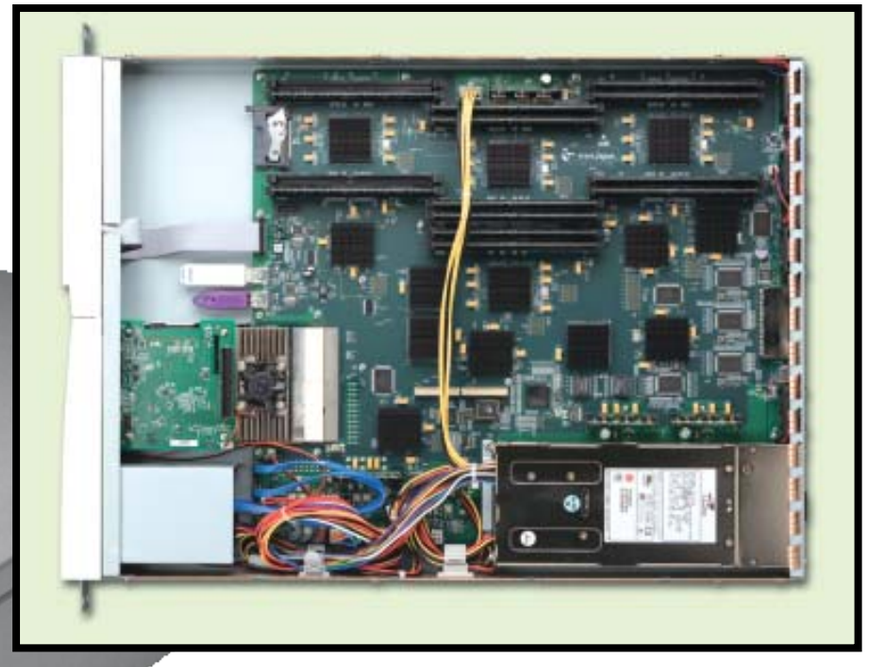


- ASIC開発のプロトタイプとして
- 特定用途向け少数生産の製品として

freeocean

ハードウェアWebキャッシュサーバ

- 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: 300W以下



freeocean

ハードウェアWebキャッシュサーバ

- 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
- 最大消費電力 W以下

HW/SW協調システム

HW/SW協調設計

省スペース/低消費電力で高性能の機器を
安く開発する

- SW中の高負荷な部分をHWで高速化
- HW中のHWで不得手な部分をCPUに実行させる
- 設計コスト削減のためHWとSWを使い分け

HW/SW協調設計

省スペース/低消費電力で高性能の機器を
安く開発する

- SW中の高負荷な部分をHWで高速化
- HW中のHWで不得手な部分をCPUに実行させる
- 設計コスト削減のためHWとSWを使い分け

高位合成言語・高位合成器で
開発コストを削減できそうだ

高位合成言語でHW/SW協調設計

期待できるメリット

- ソースコードの管理が楽になる
- 開発・デバッグ環境が統一できてうれしい
- SW/HWの機能分割が表現しやすい
  （関数，オブジェクト）

待ち受ける課題

- 実際にどう分割するか
- どうやってデータを共有するか

高位合成言語でHW/SW協調設計

期待できるメリット

- ソースコードの管理が楽になる
- 開発・デバッグ環境が統一できてうれしい
- SW/HWの機能分割が表現しやすい
  （関数，オブジェクト）

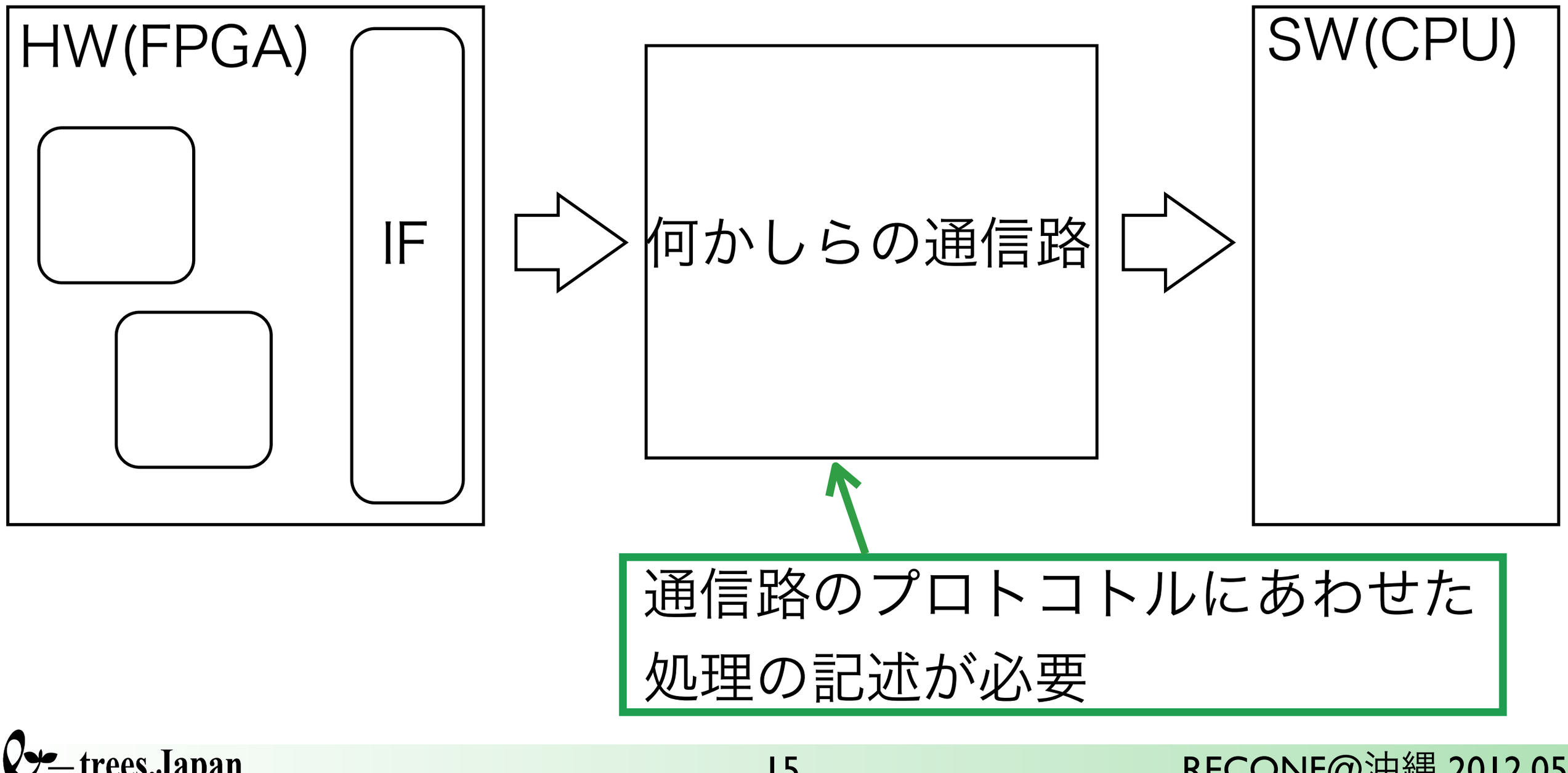
待ち受ける課題

- 実際にどう分割するか
- どうやってデータを共有するか

JavaRockで

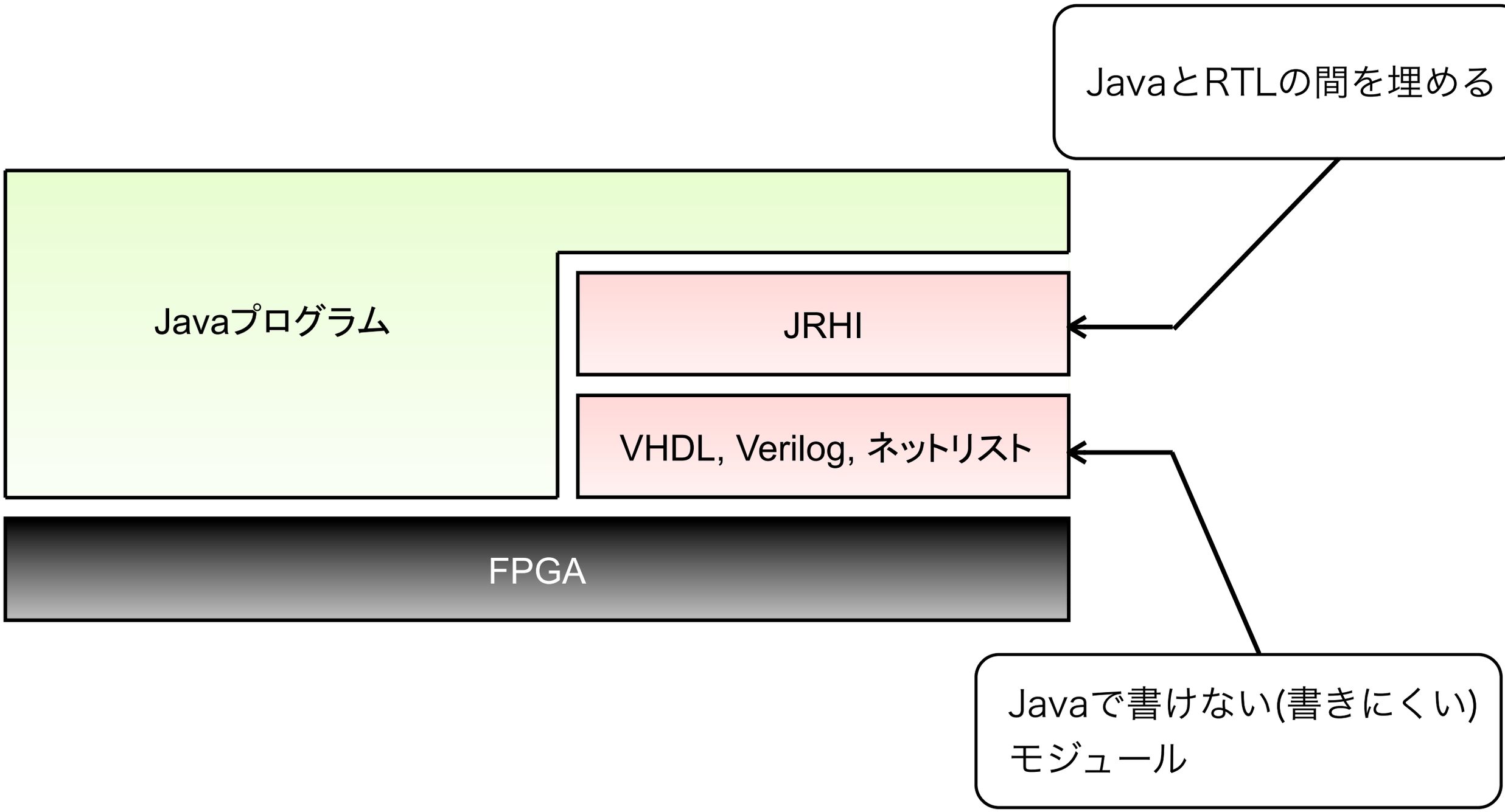
HWとSWのデータ共有

FPGA外のIFとのデータ授受が必要



JavaRock Hardware Interface

JavaRockのソフトウェア階層モデル

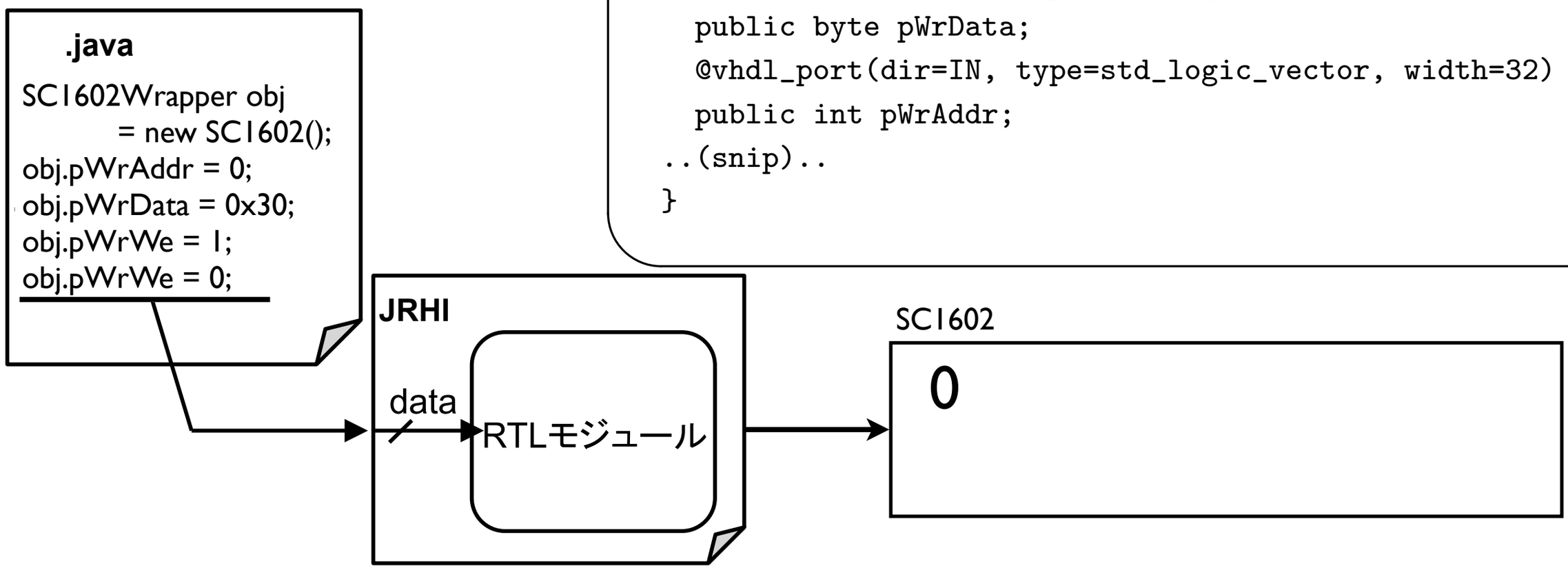


JRHIの実装上の悩み

- Javaプログラムとしてコンパイルしたい
  - 特殊な文法は導入しない
  - Javaの記述上の意味論を崩したくない
- Javaの意味論でRTLモジュールにアクセスしたい

JRHIを介したRTLモジュールへのアクセス

```
import net.wasamon.javarock.rt.*;
import net.wasamon.javarock.libraries.*;
public class SC1602Wrapper extends VHDLSimpleLibrary
                                   implements VHDLComponentIface{
  @vhdl_port(dir=IN, type=std_logic)
  public boolean pReq;
  @vhdl_port(dir=OUT, type=std_logic)
  public boolean pBusy;
  @vhdl_port(dir=IN, type=std_logic)
  public boolean pWrWe;
  @vhdl_port(dir=IN, type=std_logic_vector, width=8)
  public byte pWrData;
  @vhdl_port(dir=IN, type=std_logic_vector, width=32)
  public int pWrAddr;
 ..(snip)..
}
```



JRHIの拡張

- “Java”を崩さずに信号の集合を扱いたい

```
obj.pWrAddr = 0;  obj.pWrData = 0x30;  obj.pWrWe = 1;  obj.pWrWe = 0;
```

- プリミティブとしてJavaプログラムとRTLモジュールでデータが共有できればいい

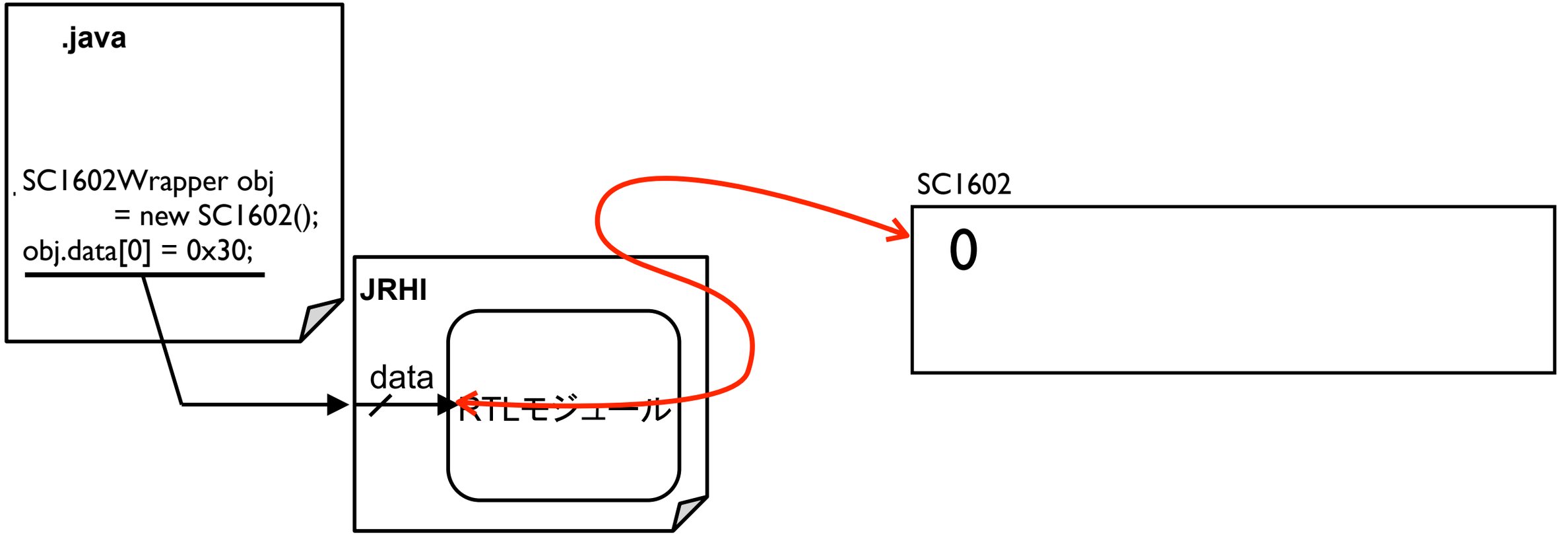
⇔ 通信路特有の制御信号は明示的に記述する

- Javaプログラムの配列をRTLモジュール内のメモリアクセスに透過的に見せる

ヒープメモリの一部をメモリに見せるというような意味付けに相当

JRHIの拡張

```
@vhdl_port(dir=OUT, type=blockram, width=8, depth=32)
public byte[] data;
```



ケーススタディによる評価

- 記述できるかどうか
- JRHIを介したアクセスの速度
- 合成した回路のリソース使用量

ケーススタディ

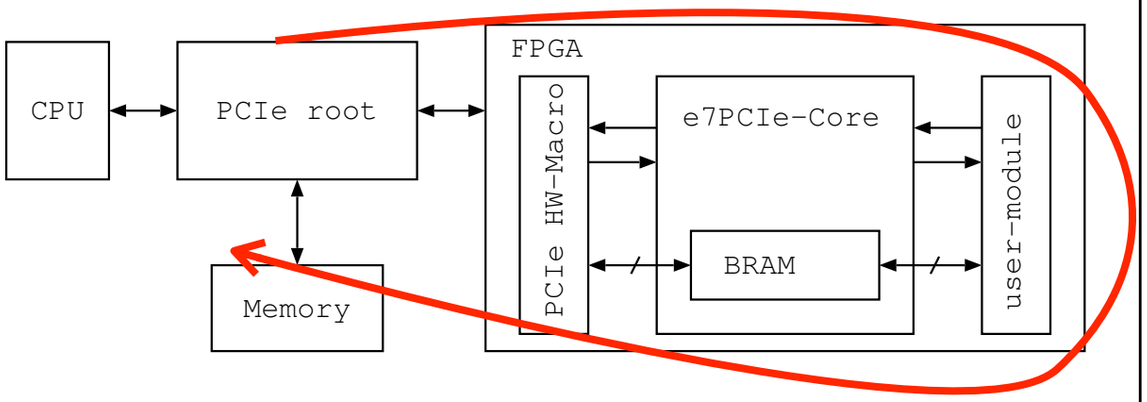
PCIeで接続する場合

- 広く普及しているx86環境
- e7 PCIe IPコア(x1, Gen1)

ネットワークで接続する場合

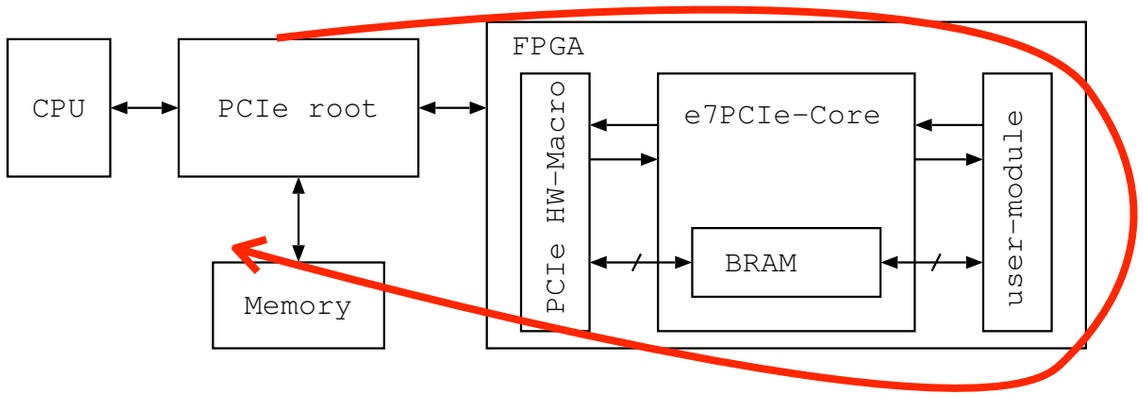
- イーサネットでの接続
- クラスタ計算システムのようなものを想定
- e7 UDP/IP IPコア(GbE)

ケーススタディ1: PCIe接続の場合




```
public class PCIeEcho{
 private final PCIe pcie = new PCIe();
 private long i = 0;
 public void run(){
   while(true){
     if (i == 0x100000) i = 0;
     while(pcie.busy == true) ;
     pcie.address = i;
     pcie.command = 0x01;
     pcie.data_length = 0x80;
     pcie.start = true;
     pcie.start = false;
     while(pcie.valid == false) ;
     pcie.read_flag = true;
     pcie.read_flag = false;
     pcie.address = 0x100000 + i;
     pcie.command = 0x04;
     pcie.data_length = 0x80;
     pcie.start = true;
     pcie.start = false;
     i = i + 1024;
   }
 }
}
```

ケーススタディ1: PCIe接続の場合



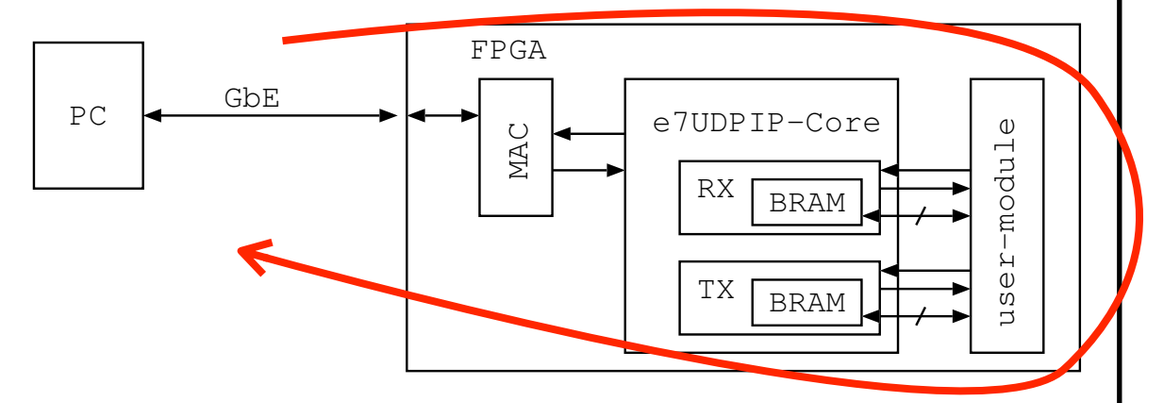
処理性能

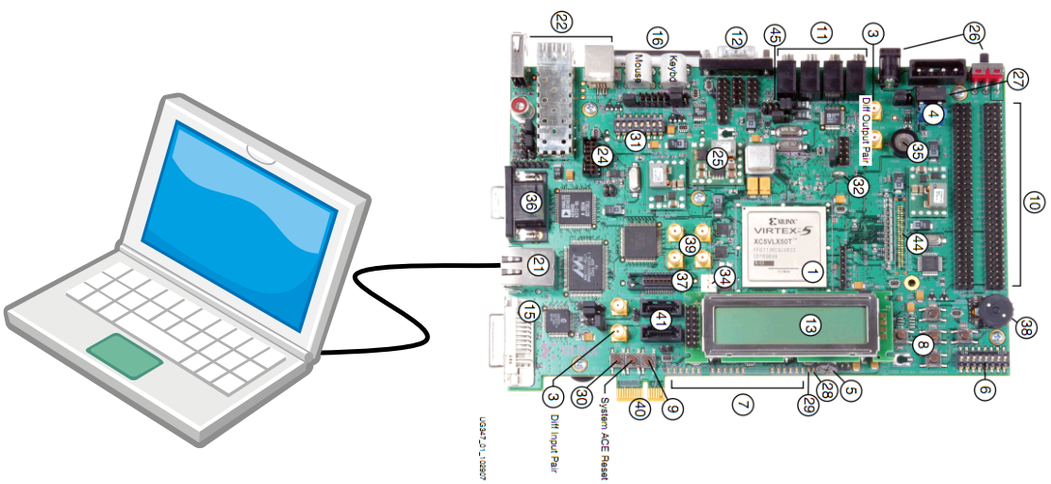
	スループット(MBps)
Native	56.40
JavaRock	56.32

リソース使用量

	レジスタ	LUT	占有スライス	BRAM36E1
Native	1620	2212	779	12
JavaRock	1472	2084	785	12

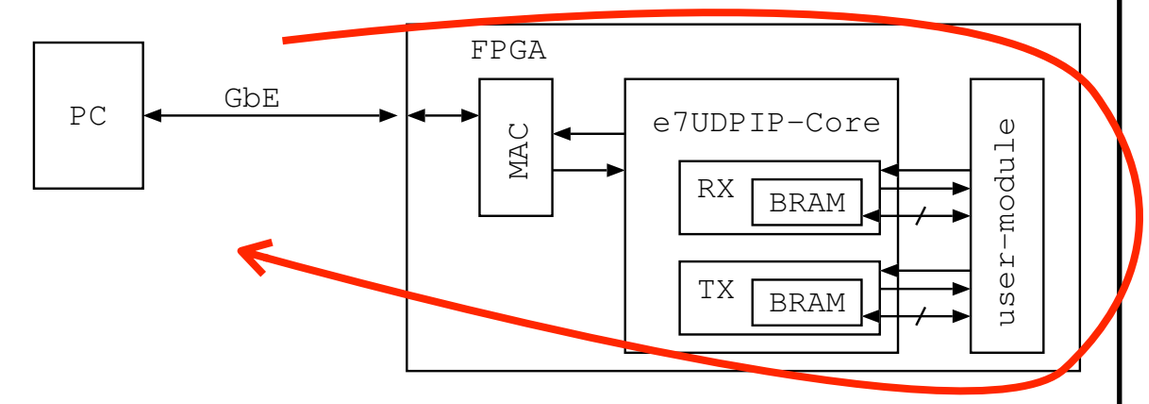
ケーススタディ2: ネットワーク接続の場合

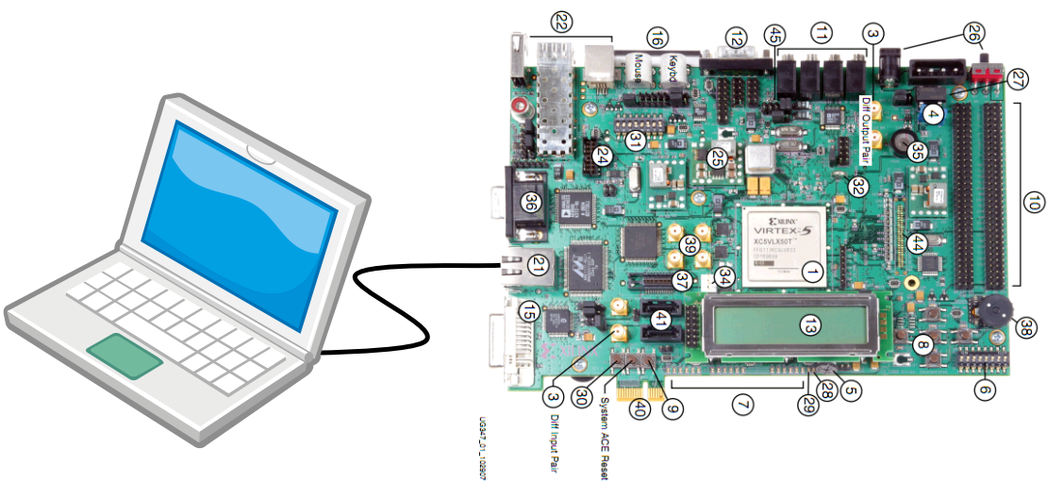




```
public class UDPEcho{
  private final UDP_TX udp_tx = new UDP_TX();
  private final UDP_RX udp_rx = new UDP_RX();
  public void run(){
    while(true){
      while(udp_rx.valid == false) ;
      udp_rx.read_flag = true;
      for(int i = 0; i < udp_rx.data_length; i++){
        int d;
        d = udp_rx.data[i];
        udp_tx.data[i] = d;
      }
      udp_tx.dest_ip     = udp_rx.src_ip;
      udp_tx.dest_port   = udp_rx.src_port;
      udp_tx.src_ip      = udp_rx.dest_ip;
      udp_tx.src_port    = udp_rx.dest_port;
      udp_tx.data_length = udp_rx.data_length;
      udp_rx.read_flag = false;
      while(udp_tx.busy == true) ;
      udp_tx.start = true;
      udp_tx.start = false;
    }
  }
}
```

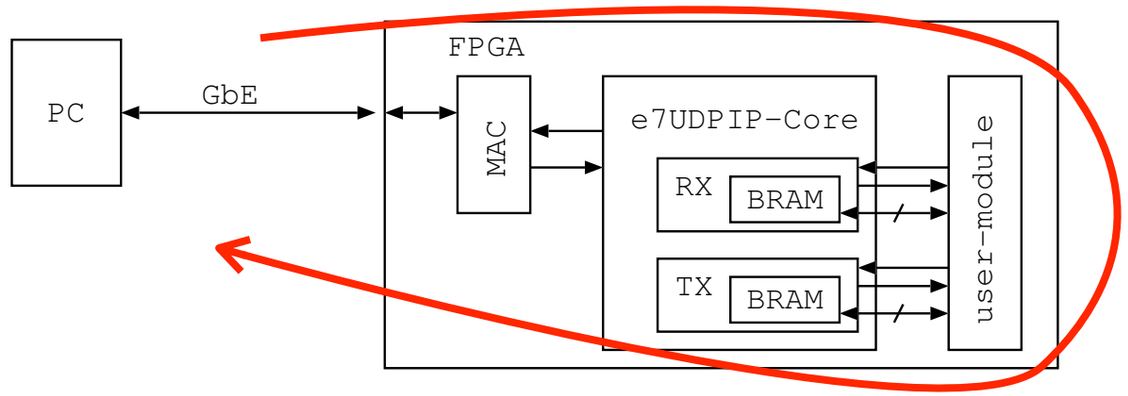
ケーススタディ2: ネットワーク接続の場合





```
public class UDPEcho{
  private final UDP_TX udp_tx = new UDP_TX();
  private final UDP_RX udp_rx = new UDP_RX();
  public void run(){
    while(true){
      while(udp_rx.valid == false) ;
      udp_rx.read_flag = true;
      for(int i = 0; i < udp_rx.data_length; i++){
        int d;
        d = udp_rx.data[i];
        udp_tx.data[i] = d;
      }
      udp_tx.dest_ip     = udp_rx.src_ip;
      udp_tx.dest_port   = udp_rx.src_port;
      udp_tx.src_ip      = udp_rx.dest_ip;
      udp_tx.src_port    = udp_rx.dest_port;
      udp_tx.data_length = udp_rx.data_length;
      udp_rx.read_flag = false;
      while(udp_tx.busy == true) ;
      udp_tx.start = true;
      udp_tx.start = false;
    }
  }
}
```

ケーススタディ2: ネットワーク接続の場合



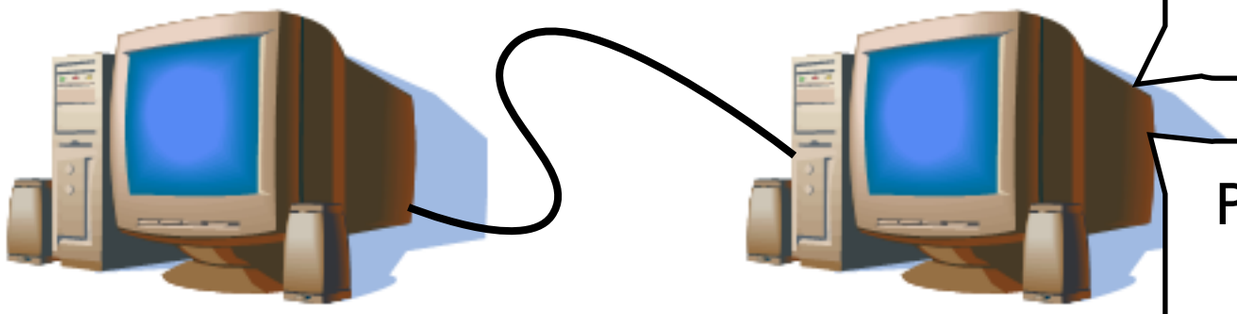
処理性能(MBps)

	16Byte	256Byte	1024Byte
Native	0.262	3.31	8.47
JavaRock	0.262	3.20	8.32
JavaRock(従来版)	0.259	3.03	6.82

リソース使用量

	レジスタ	LUT	占有スライス	BRAM36E1/18E1
Native	2226	3184	1075	14/4
JavaRock	2450	3236	1302	14/4

ケーススタディ2: ネットワーク接続の場合



PC(1)
Pentium4 2.8GHz
512MB
CentOS

PC(2)
Core i7-3930K 3.2GHz
16GB
Windows7 Pro.

処理性能(MBps)

	16Byte	256Byte	1024Byte
JavaRock	0.262	3.20	8.32
PC(1) Java	0.079	1.28	4.61
PC(1) C	0.107	1.71	5.21
PC(2) Java	0.080	1.28	5.05

まとめ

- JavaRockでHW/SW協調設計するためにJRHIを拡張
  - メモリアクセスをRTLと透過的にみせる
  - ヒープの一部を外部にマッピングすることに相当
- PCIe, ネットワーク接続のケーススタディ
  - 記述できるかどうか
  - 性能/リソース使用量はNativeと大きく変わらない

今後の課題

- 通信路の上に，一般的なプロトコルをのせる
  - RPC，CORBA，...
  - SCI-ME
- IFをJavaの標準ストリームにマッピングする

http://javarock.sourceforge.net