



モデル検査向け上位設計言語Melasy+に対する 仕様埋め込み拡張とHDLコード生成

信州大学大学院工学系研究科 情報工学専攻
09TA526H 白鳥 航亮

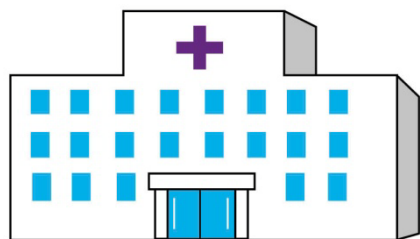
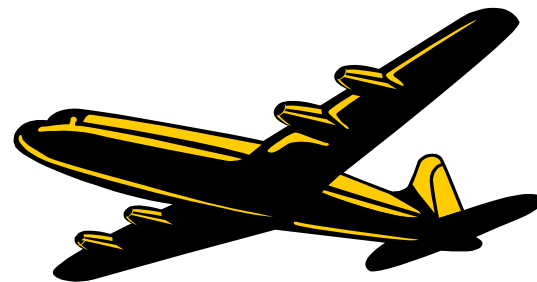
概要

- 背景と目的
- モデル検査向け上位設計言語Melasy+
- 仕様の埋め込みと展開
- HDLコード生成とシミュレート
- 高機能アービタの検証と実装
- 評価と考察
- まとめ

背景と目的

背景

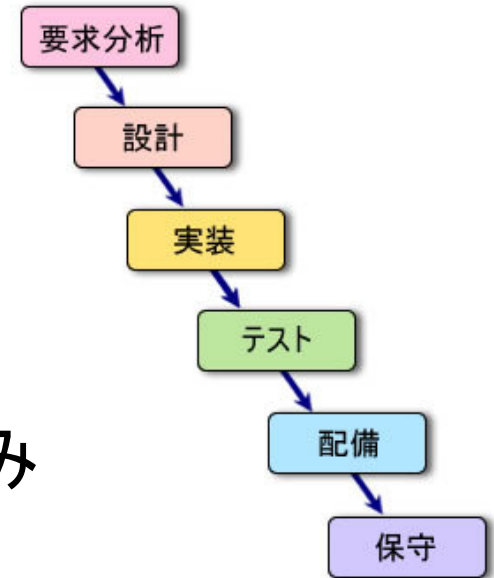
- 社会には様々なシステムが存在
- 信頼性の向上



- 欠陥を検査によってださない
 - 欠陥があっても継続して動作する
-
- 問題
 - 回路規模の増大
 - 回路の複雑化

信頼性向上へのアプローチ

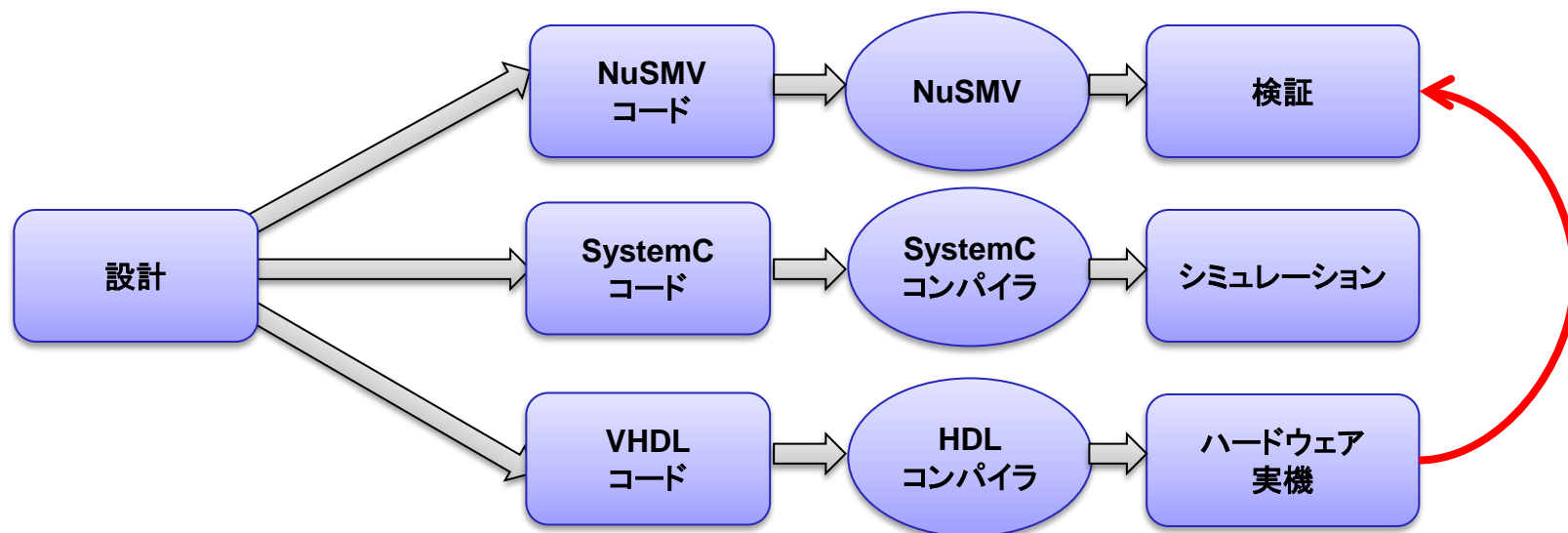
- モデル検査
 - 設計段階での検査手法
 - 設計が仕様をみたすのか？
- シミュレートでは特定の入力に対してのみ
 - 状態機械としての表現
 - すべての状態が存在(すべての入力列)



- B.Berard, M.Bidoit, A.Finkel, F.Laroussinie, A.Petit, L.Petrucci, Ph.Schnoebelen, P.McKenzie : “Systems and Software Verification Model-Checking Techniques and tools” ; Springer, 2001.

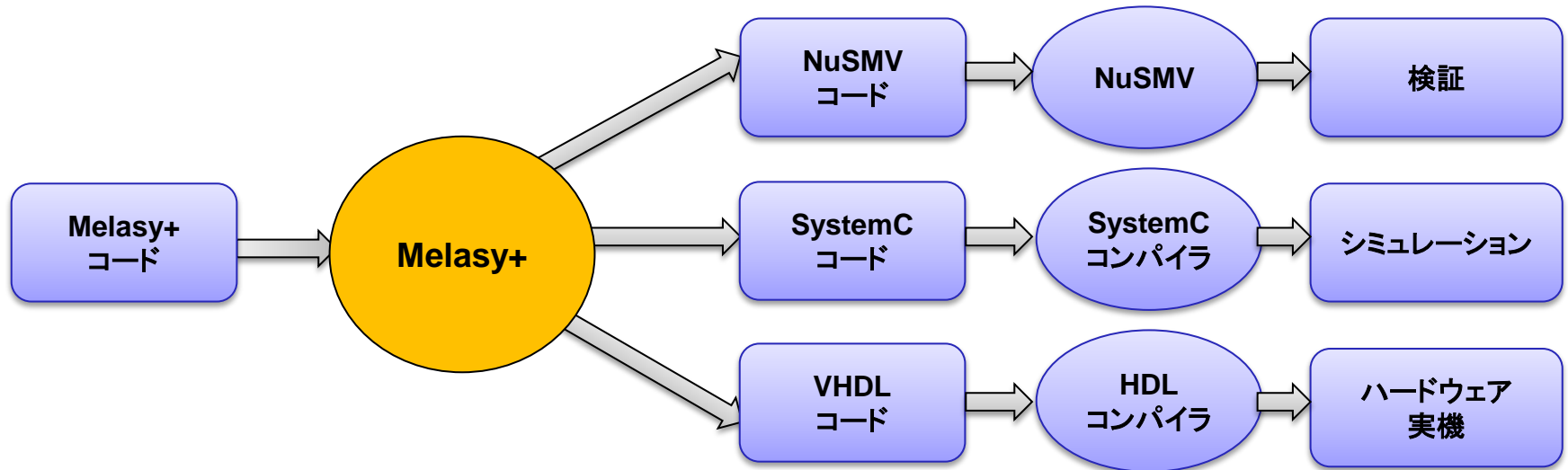
工程数の増加

- ハードウェアをソフトウェアベースで開発 (HDL)
- 複数のツールを使用 = 複数のコードを記述
- 仕様の変更に伴う、システムの再設計
 - 検証に立ち戻る必要あり



上位設計言語 Melasyの開発

- 既存の記述言語の上の層に新たな上位記述言語を置く。
- 単一のコードからの自動生成

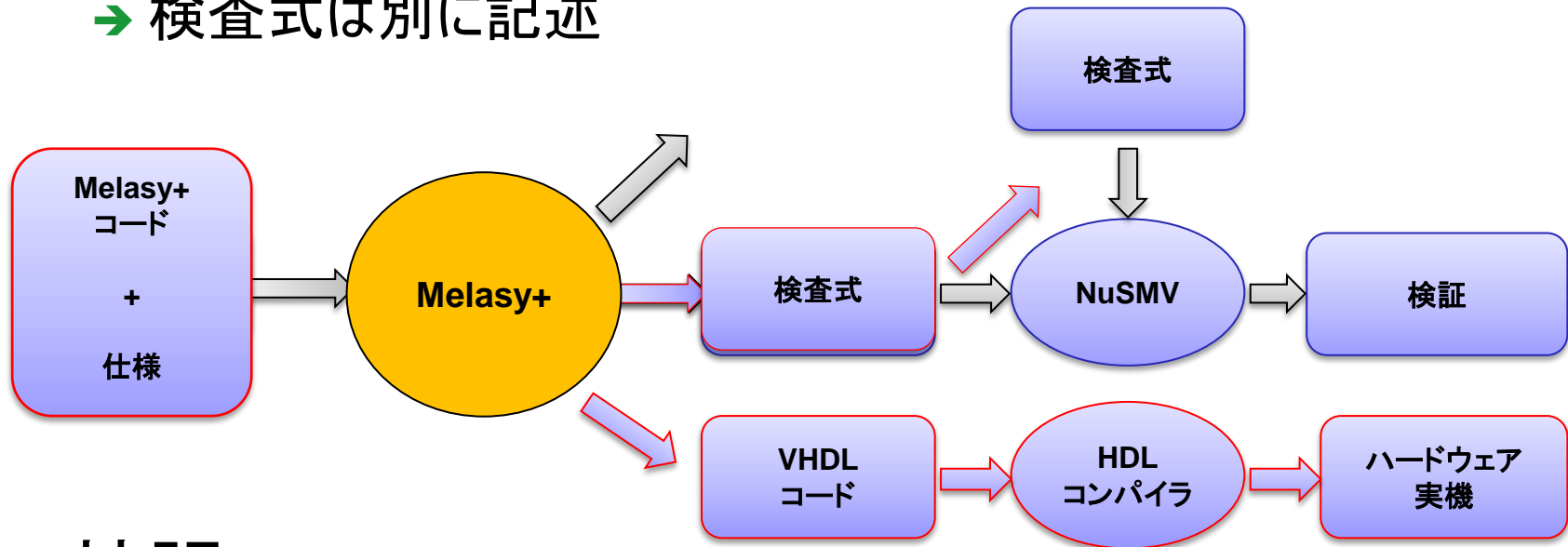


- N.Iwasaki, K.Wasaki : ``A Meta Hardware Description Language Melasy for Model Checking Systems'' ; Proc. of the 5th Int'l Conf. on Information Technology : New Generations (ITNG2008), pp.273-278, 2008.

検証から実装までを一貫して行う

■ 従来 : モデル検査向け機能のみ

→ 検査式は別に記述



■ 拡張

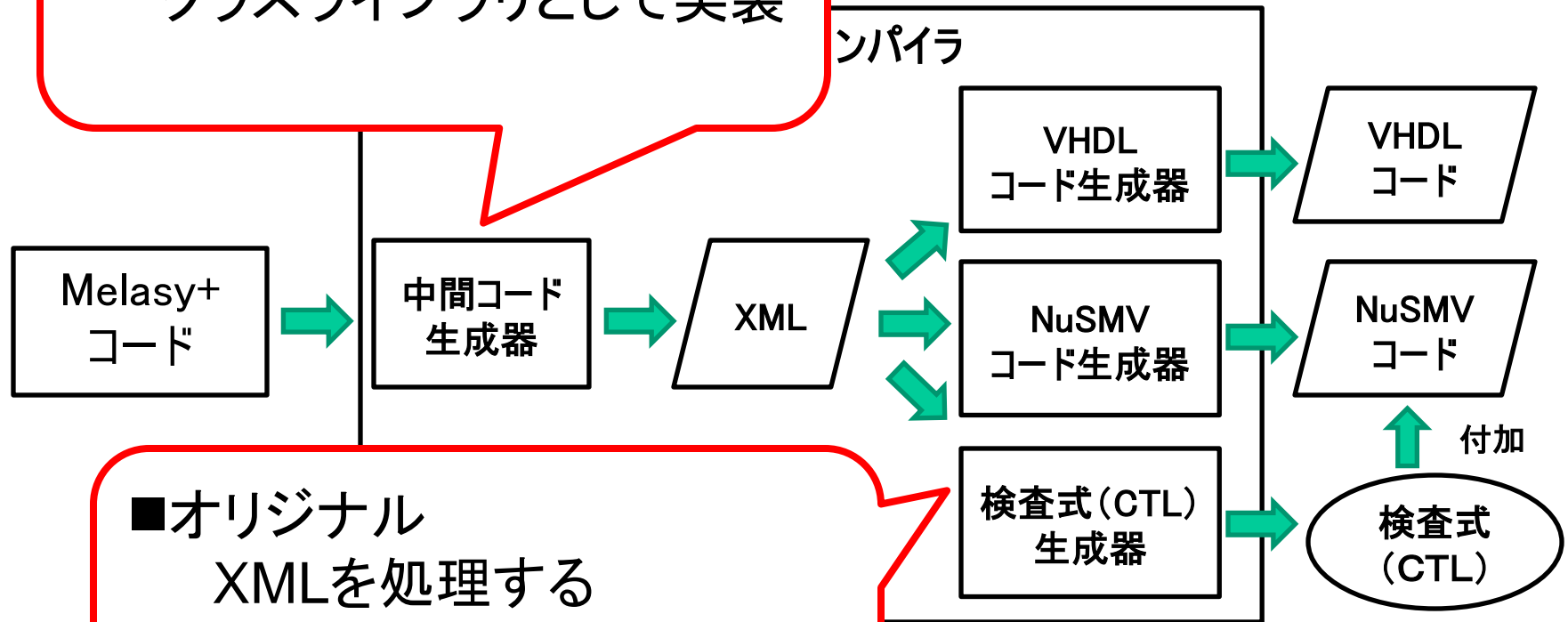
→ 仕様の埋め込み 検査式の生成

→ HDLコードの生成 実装

モデル検査向け上位設計言語 Melasy+

Melasy+の構成

■C++コンパイラを利用
必要な機能をC++の
クラスライブラリとして実装



を吐くコンパイラの開発

■オリジナル
XMLを処理する
Python,perl言語などで聞
記述

する必要がなくなる

特徴

- 構文規則: C++
- C++ のライブラリ “melasy.h”
 - 回路記述のための 型・機能 を提供
- C++ 既存の機能を利用可能
 - for, if, template など

Melasy+

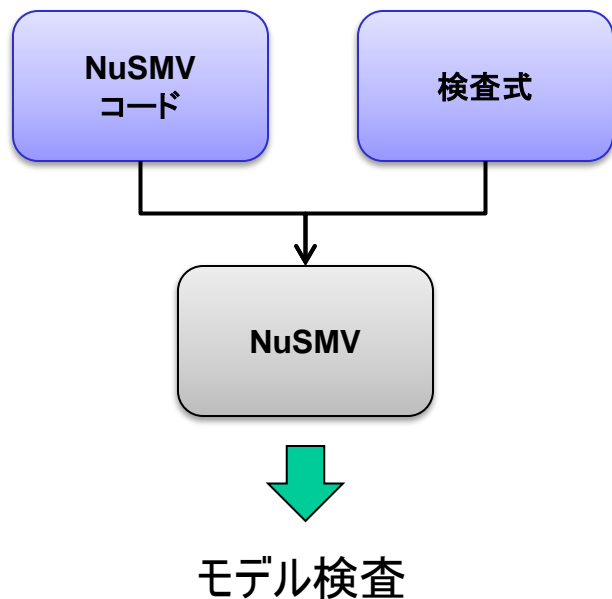
```
for(int i = 2; i < N; i++){  
    flag[ i ].in <= flag[ i-1].out,  
}
```

従来手法

```
MODULE Main()  
VAR  
    flag_2 : Flag(flag_1.out);  
    flag_3 : Flag(flag_2.out);  
    ⋮  
    flag_N-1: Flag(flag_N-2.out);
```

仕様の埋め込みと展開

モデル検査を行うために



■ モデル検査を行うためには

1. 回路モデル
2. 検査式(仕様を論理式で表現したもの)

が必要。

仕様の埋め込み

- 従来: 論理式での直接記述 (Computation Tree Logic)

$$\text{AG}(Q \rightarrow !E[!R \cup (!P \& !R \& \text{EX}(P \& E[!R \cup (!P \& !R \& \text{EX}(P \& E[!R \cup (!P \& !R \& \text{EX}(P \& !R \& \text{EF}(R))))))]))])$$

- 仕様パターンとして、仕様を埋め込む

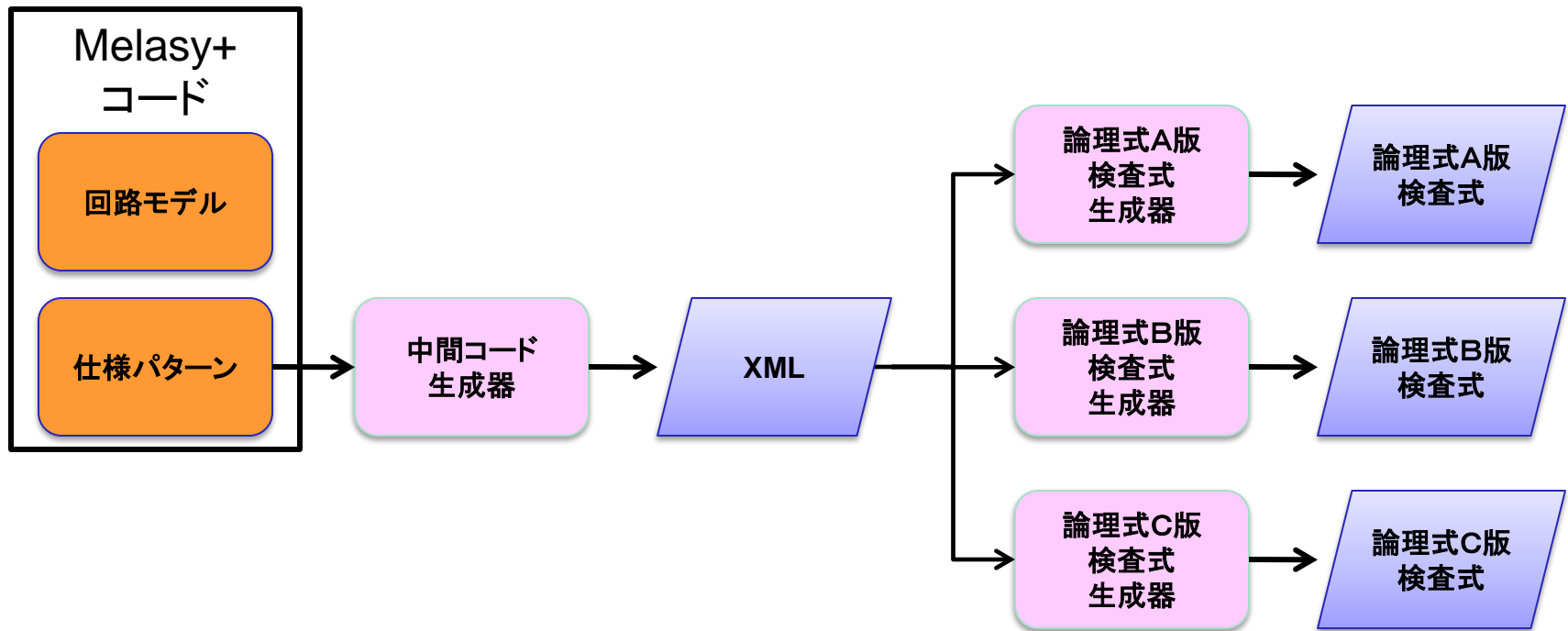
P Existence 2 times Between Q and R

- 記述・理解が容易に
- C++コードの言語機能の利用が可能

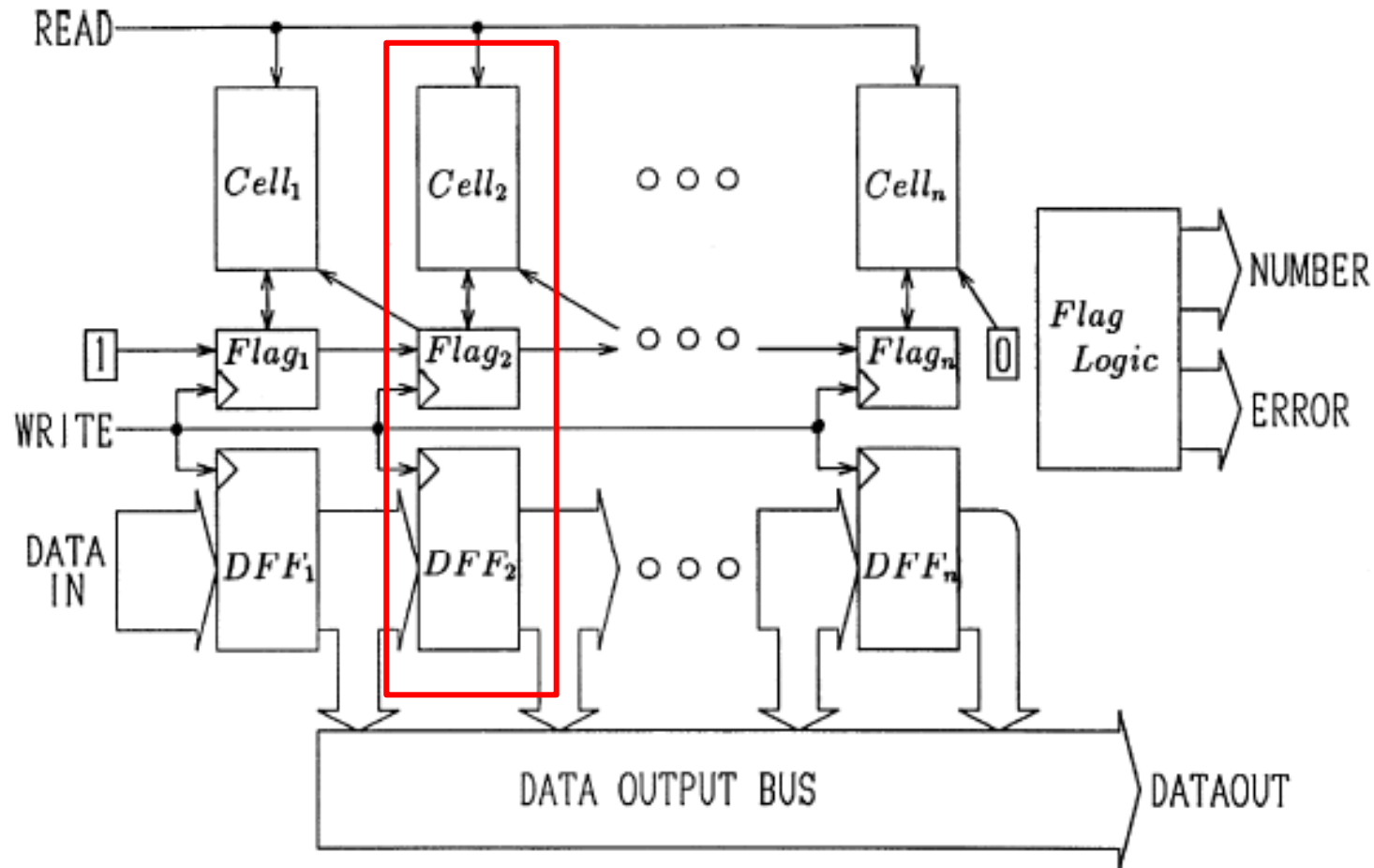
- M.B.Dwyer, G.S.Avrutin, J.C.Corbett ; “Patterns in Property Specifications for Finite-state Verification”, Proc. of the 21st Int’l Conf. on Software Eng., 1999.

仕様の展開

■ 回路モデルと同様に展開



ケーススタディ (FIFOメモリ)



■ 和崎克己, 不破 泰, 江口正義, 中村八束: “セルオートマトンの概念を用いた自己回復能力をもつ通信用バッファ”; 電子情報通信学会論文誌, Vol.J77-D-I, No.1, pp.41-52, 1994.

ケーススタディ (FIFOメモリ)

- すべてのフラグが満たすべき仕様を記述
- forループを用いた抽象化表現

```
template<int N>
class Cyg_main : public Component
{
public :
    Cyg_main(){
        //仕様
        for(int i = 0; i < N; i++){
            Response re(cpu.write == 1 , flag[i].own == 1);
            re.globally();
            CTL(re);
        }
    }
};
```

Melasy+

```
ctl generator for melasy version 1.0
AG(CPU 0 . write = 1 → AF(flag 0 . own = 1))
AG(CPU 0 . write = 1 → AF(flag 1 . own = 1))
AG(CPU 0 . write = 1 → AF(flag 2 . own = 1))
```

CTL

HDLコード生成とシミュレート

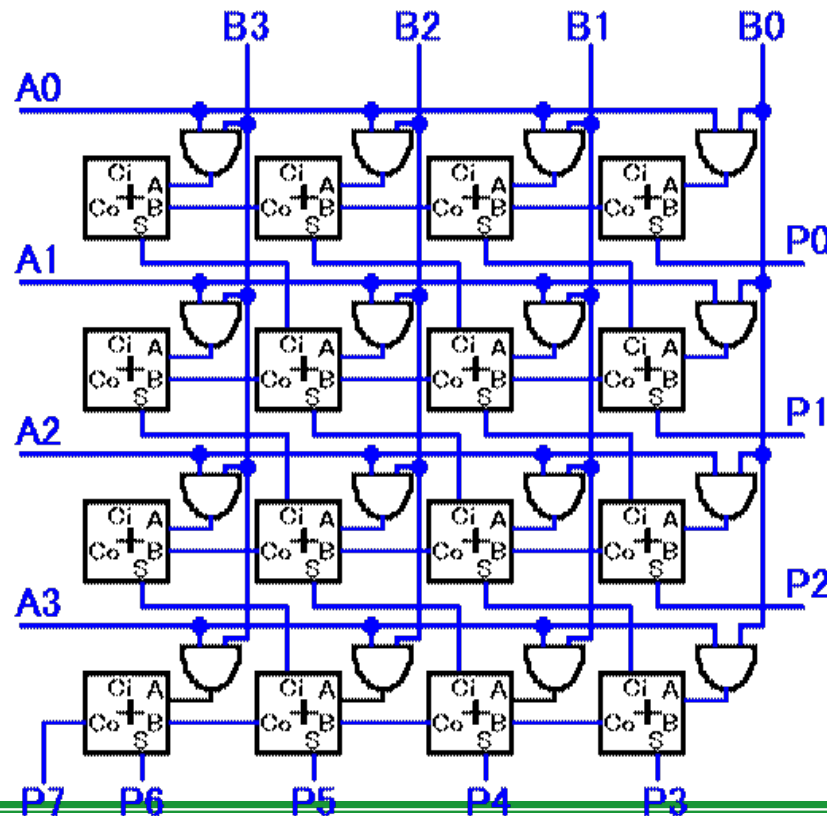
HDLコード生成

- Melasy+の目標：
検証から実装までを一貫して行う
- VHDLコード生成器
→ Python言語で記述

Melasy+	VHDL
Logic	std_Logic
Digit $\langle n \rangle$	std_logic_vector($n - 1$ downto 0)
PortMap	PortMap
in	in
out	out
sync	<=

ケーススタディ(乗算器の設計)

- 加算器 + 部分積計算用のandゲート
- 同様の構造が二次元に展開する



乗算回路の設計

Melasy+

```
#include "melasy.h"
using namespace std;
class MULTIPLER : public Component
{
    MULTIPLER(){
        in(x);
        in(y);
        out(z);

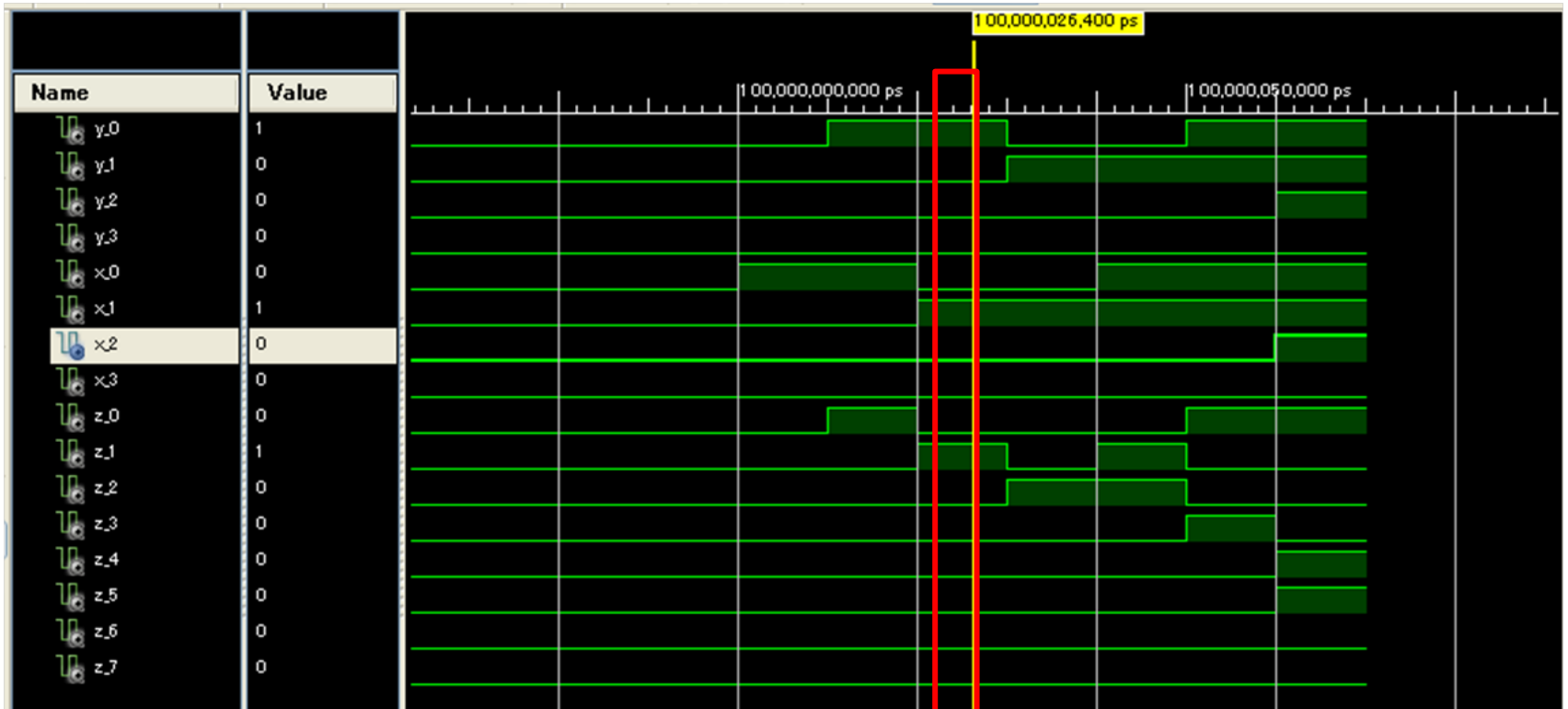
        for(int i = 0; i < N; i++){
            for(int j = 0; j < N; j++){
                if(i == 0 && j == 0){
                    PortMap pm_csa[] = {
                        csa[i][j].ci <= '0',
                        csa[i][j].b <= '0',
                        csa[i][j].y <= y[j],
                        csa[i][j].x <= x[i]
                    };
                }
            }
        }
    };
};
```

VHDL

```
MULTIPLER_ELEMENT_0 :
    MULTIPLER_ELEMENT port map (
        ci => '0',
        b => '0',
        co => MULTIPLER_ELEMENT_0_co,
        s => MULTIPLER_ELEMENT_0_s,
        y => y_0,
        x => x_0
    );
MULTIPLER_ELEMENT_1 :
    MULTIPLER_ELEMENT port map (
        ci => MULTIPLER_ELEMENT_4_s,
        b => '0',
        co => MULTIPLER_ELEMENT_1_co,
        s => MULTIPLER_ELEMENT_1_s,
        y => y_1,
        x => x_0
    );
```

シミュレート

■ XILINX ISim を用いたシミュレート



$$\boxed{y_3 \cdots y_0 : 0001} \quad * \quad \boxed{x_3 \cdots x_0 : 0010} \\ = \quad \boxed{z_7 \cdots z_0 : 00000010}$$

演算の表現方法

- C++演算子のオーバーロード：異なる型は演算不可

- LogicとDigitの二つの型

Logic * Logic, Digit<2> * Digit<2>

○

Digit<2> * Digit<4> : 二つの型は異なる

×

- 解決のためのアプローチ

1. ライブラリにして埋め込む => 乗算回路は複数存在

×

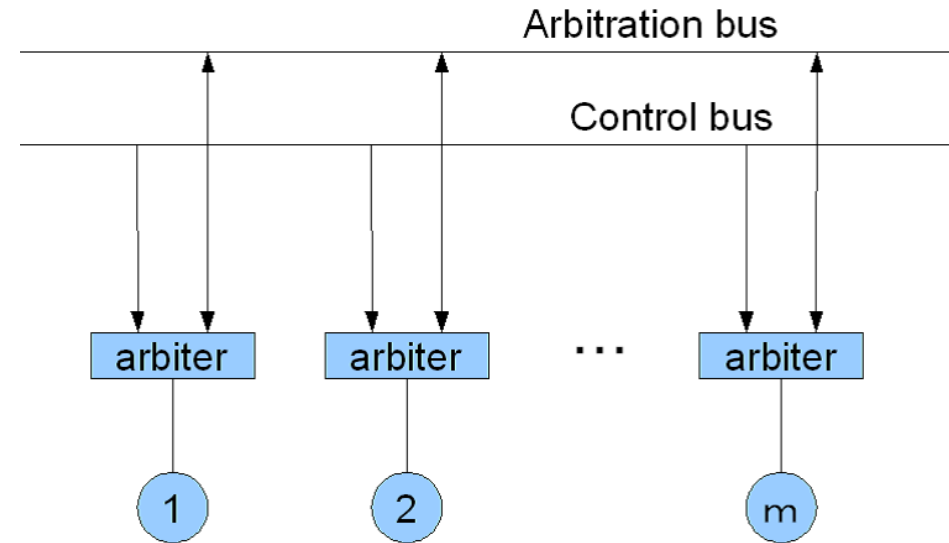
2. クラスのメンバ関数を実装(出力は演算子)

```
multi(){
    in = in.multiplier(a, b);
    sync(o, in);
}
```

高機能アービタの検証と実装

ケーススタディ(バスアービタ)

- 検証から実装までを行う
 - モデル検査: NuSMV
 - シミュレート: Xilinx ISim
 - 実装: Xilinx Spartan3E FPGA
- 高機能バスアービタ
 - 並列決定方式
 - 組み込み自己テスト機能
 - 各アービタはIDを保持
 - IDの大小で優先度が決定



時藤, 黒川, 古賀, : “セルフテストングバスアービタの一実現法について” ;
信学論D-I,75(1), 30-40, 1992

回路構成

Melasy+

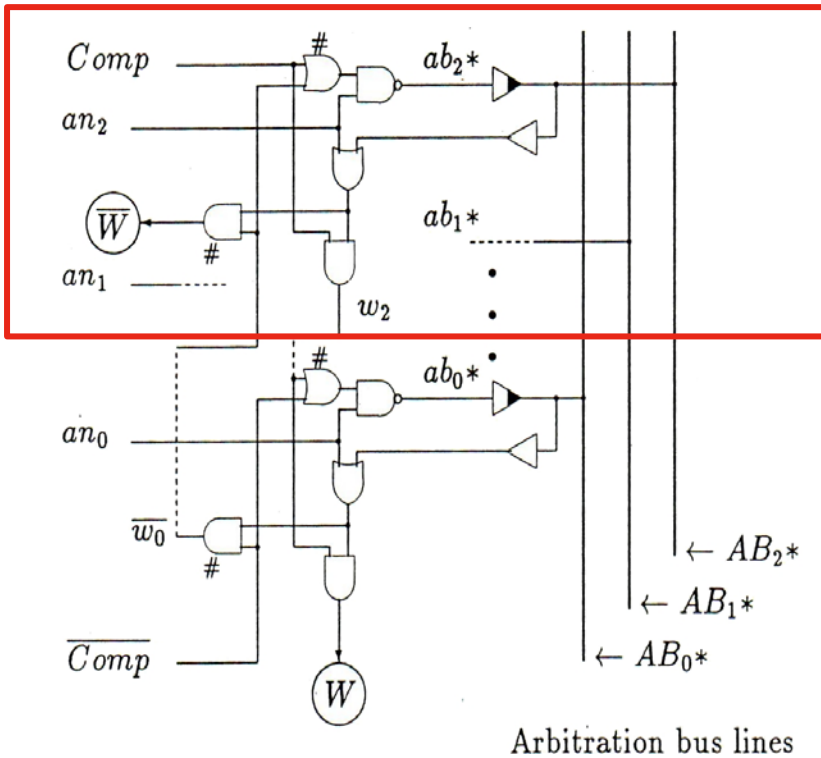


Fig. Circuit configuration for each node

```

template<int N>
class Arbiter : public Component
{
public :
    ...
    Arbiter ()
    {
        in(an);

        //PortMap For BusArbiter
        for(int i = 0; i < N; i++)
        {
            if(i == 0) {
                PortMap pmba[] = {
                    BA[i].comp <= BA[i+1].w,
                    BA[i].compB <= compB,
                    BA[i].an <= an[i],
                    BA[i].ab_wb <= ab_wb[i]
                };
            }
            ...
        }
    }
};
    
```

コード生成

```
MODULE Arbiter_3(ab_wb_0, ab_wb_1, ab_wb_2, an_0, an_1, an_2, comp, compB)
VAR
  ArbiterElement_0 : ArbiterElement(ab_wb_0, an_0, ArbiterElement_1.w, compB);
  ArbiterElement_1 : ArbiterElement(ab_wb_1, an_1, ArbiterElement_2.w, ArbiterElement_0.wB);
  ArbiterElement_2 : ArbiterElement(ab_wb_2, an_2, comp, ArbiterElement_1.wB);
  ...
```

NuSMV

```
architecture melasy_generated of Arbiter_3 is component ArbiterElement
...
begin
  ArbiterElement_0 : ArbiterElement port map (
    compB => compB,
    ab => ArbiterElement_0_ab,
    wB => ArbiterElement_0_wB,
    comp => ArbiterElement_1_w,
    ...
```

VHDL

モデル検査とシミュレート

- 生成されたコードを持ちいて、正常にモデル検査が行われることを確認
- 仕様
 - 複数のアービタが同時にバスの使用权をえることはない
 - 最大のIDを持つノードが必ずバスの仕様権を得る など

```
kousuke@LENOVO-9C758E9B ~/melasy_plus/arbiter_source
```

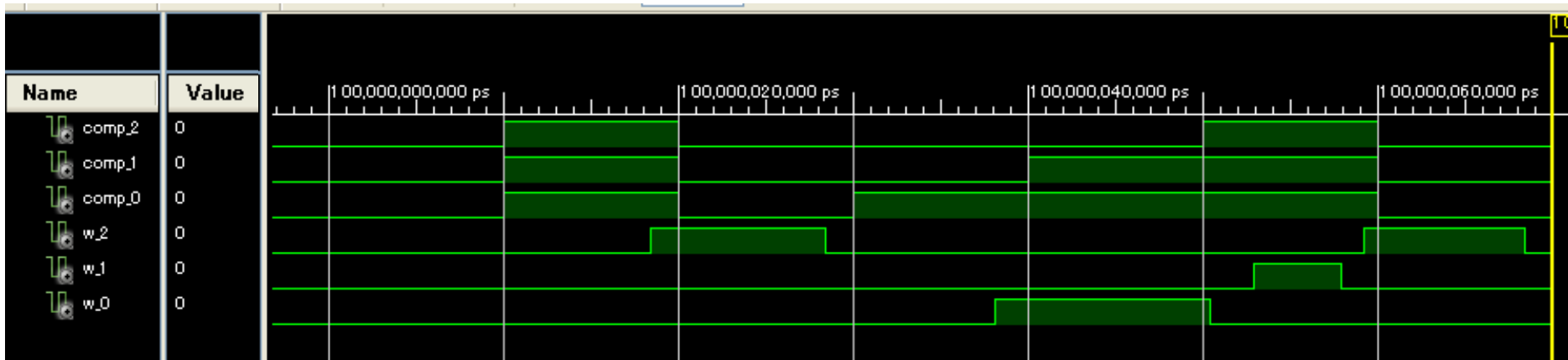
```
$ NuSMV.exe p100126¥ ¥(1¥).smv
```

```
*** This is NuSMV 2.4.3 (compiled on Mon May 18 14:07:13 UTC 2009)  
*** For more information on NuSMV see <http://nusmv.irst.itc.it>  
*** or email to <nusmv-users@irst.itc.it>.  
*** Please report bugs to <nusmv@irst.itc.it>.
```

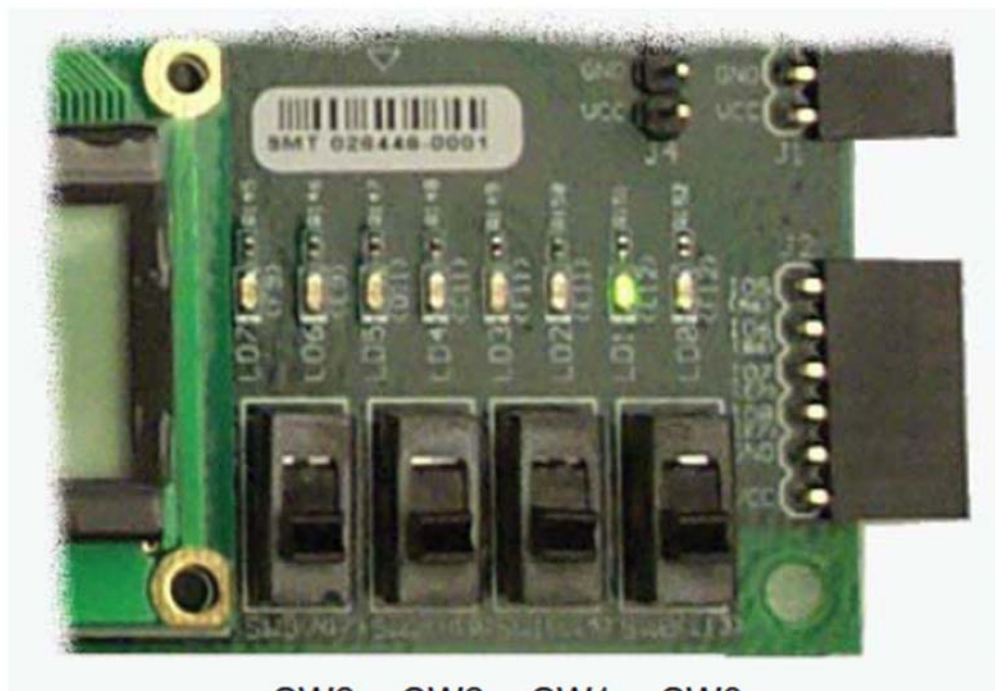
```
-- specification AG (((LocalArbiter_3_1.Controller_3_0.counter = 0d5_3 &  
LocalArbiter_3_1.Controller_3_0.state = 0b2_1) & LocalArbiter_3_2.Controller_3_0.state  
= 0b2_1) -> A [ LocalArbiter_3_2.Controller_3_0.counter != 0d5_31  
U !((LocalArbiter_3_2.Controller_3_0.counter = 0d5_31 &  
LocalArbiter_3_1.Controller_3_0.phase1_w = 1) &  
LocalArbiter_3_2.Controller_3_0.phase1_w = 1) ] ) is true
```

VHDLシミュレート

- Target : XILINX SPARTAN3E xc3s500e FPGA
- HDL compiler/Logic synthesizer : XILINX ISE
- すべてのノードがバスの使用权を要求
- 優先度 : Node2 > Node1 > Node0
- 各信号線
 - Comp_* : バス使用权の要求
 - W_* : '1' のとき調停に勝利



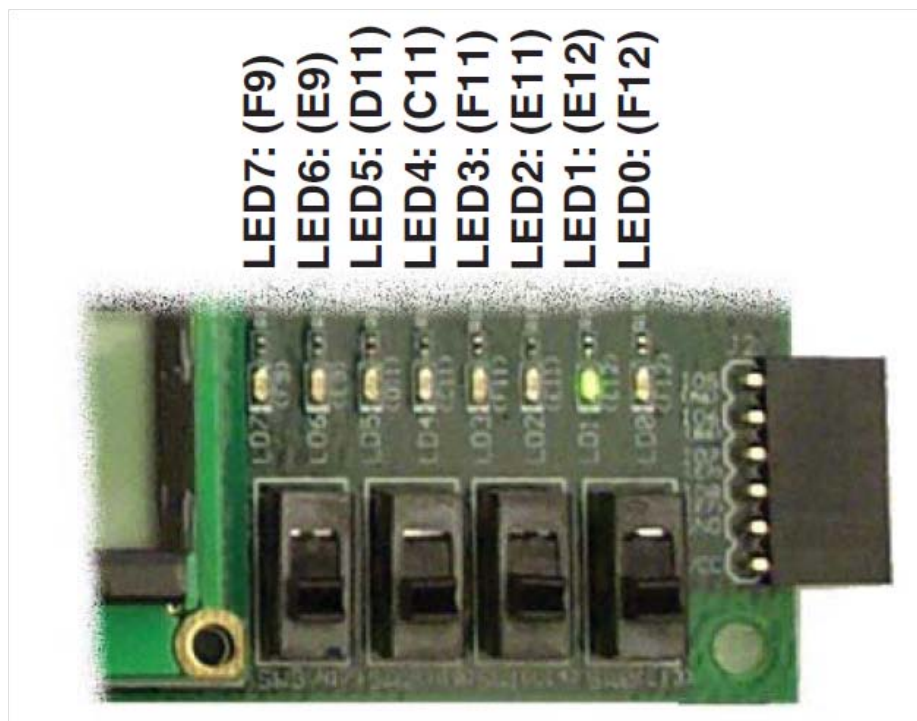
入力の割り当て



SW3 (N17) SW2 (H18) SW1 (L14) SW0 (L13)

スイッチ	信号
SW0	Comp0
SW1	Comp1
SW2	Comp2

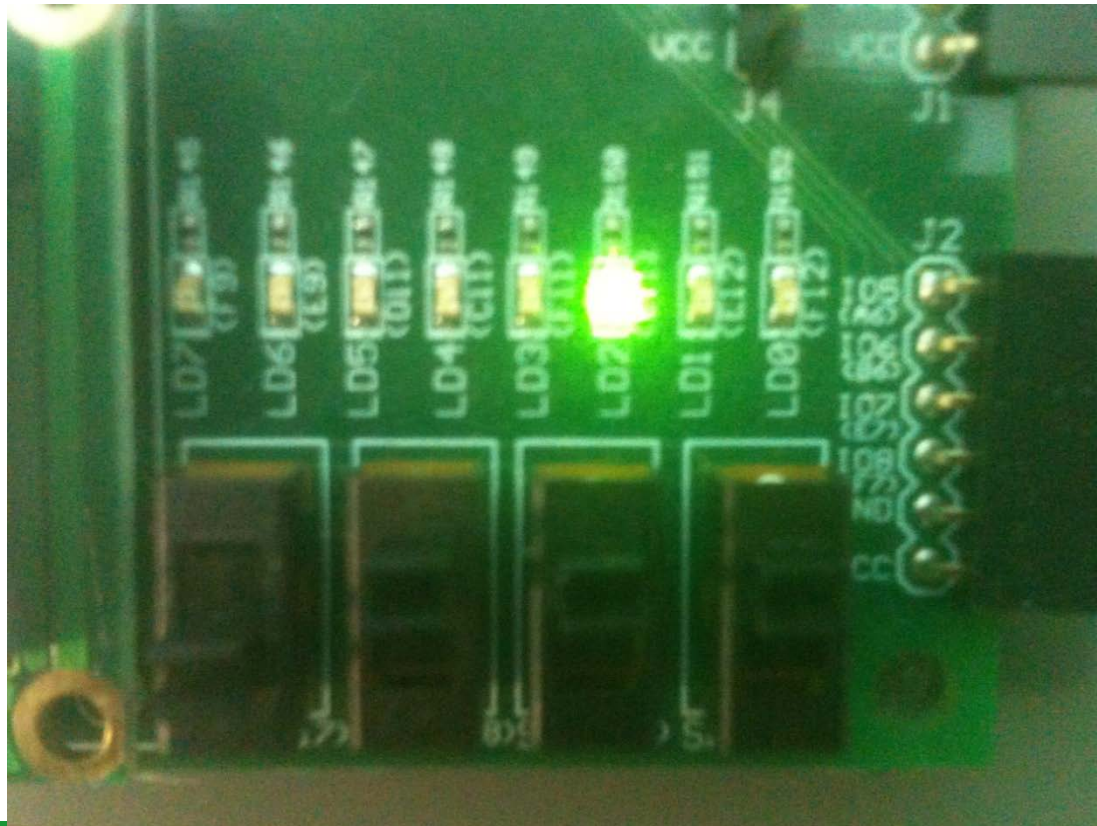
出力の割り当て



LED	信号
LED0	W0
LED1	W1
LED2	W2
LED3	
LED4	
LED5	W0_B
LED6	W1_B
LED7	W2_B

実装結果

- すべてのノードがCompをアクティブに
- 優先度 ノード2 > ノード1 > ノード0



評価と考察

検証から実装まで

- 仕様の埋め込み
 - 仕様パターンを用いた記述
 - C++の言語機能の利用
- HDLコード生成機能
 - VHDLコードを生成
 - 検証から実装までを一貫して行う

課題：内部信号線の必要性

- 途中で定義した変数は出力されない
 - 可読性の低下
 - 他のコードではゆるされない形式

```
Digit<4> inport ;  
Digit<4> outport;  
signal(  
    Digit<8> signalA = inport * "1011";  
    Digit<4> signalB = signalB.copy(signalA, 7,4);  
    sync ( out , port );  
}
```

Melasy+

```
architecture melasy_generated of signal is  
begin  
    out_port <= in_port * "11"(7 downto 4);  
end melasy_generated;
```

VHDL

ソフトウェア・ハードウェア協調設計

- システム
 - ハードウェアとソフトウェアの組み合わせ
 - 割合に応じて、コストや動作速度が増減
- C++のコードとして記述
 - C++のコードとして駆動できる可能性
- ハードウェアとソフトウェアの割合を自由に調整できる

組み込みテスト回路の生成

- 回路をコンポーネントごとに切り分けた記述
- 仕様の記述
 - 各コンポーネントの入出力ポートについて記述
 - 仕様として存在する = 重要なポイント
- テスト回路の規模をどの程度まで許容するか
- テスト回路の動作速度が全体の速度に影響する。

まとめ

- 仕様の埋め込みを行い、検査式の生成機能を実装した
- Melasy+コード唯一つの記述でモデル検査が可能
- VHDLコード生成機能の実装
- モデル検査から実装までを一貫して行える
- 内部信号線の必要性
- ソフトウェア・ハードウェア協調設計
- 組み込み自己テスト回路

参考文献

- 米田, 梶原, 土屋 : ``ディペンダブルシステムー高信頼システム実現のための耐故障・検証・テスト技術” ; 共立出版, 2005.
- The SMV System : Carnegie Mellon University, <http://www.cs.cmu.edu/~modelcheck/smv.html>
- NuSMV: a new symbolic model checker, <http://nusmv.irst.itc.it/>
- E.M.Clarke, O.Grumberg, D.Peled : ``Model Checking” ; MIT Press, 2000.
- B.Berard, M.Bidoit, A.Finkel, F.Laroussinie, A.Petit, L.Petrucci, Ph.Schnoebelen, P.McKenzie : ``Systems and Software Verification Model-Checking Techniques and tools” ; Springer, 2001.
- VHDL : VHSIC Hardware Description Language, <http://vhdl.org/>
- Verilog : IEEE Standard Verilog Hardware Description Language, IEEE 1364-2001 Revision C, IEEE Verilog Standardization Group, <http://www.verilog.com/IEEEVerilog.html>
- N.Iwasaki, K.Wasaki : ``A Meta Hardware Description Language Melasy for Model Checking Systems “ ; Proc. of the 5th Int ‘l Conf. on Information Technology : New Generations (ITNG2008), 273-278, 2008..

- 和崎克己, 不破 泰, 江口正義, 中村八束 : ``セルオートマトンの概念を用いた自己回復能力をもつ通信用バッファ“ ; 電子情報通信学会論文誌, Vol.J77-D-I, No.1, pp.41--52, 1994
- 時藤, 黒川, 古賀, : ``セルフテストイングバスアービタの一実現法について” ; 信学論 D-I, 75(1), 30-40, 1992.
- M.B.Dwyer, G.S.Avrinin, J.C.Corbett ; ``Patterns in Property Specifications for Finite-state Verification”, Proc. of the 21st Int'l Conf. on Software Eng., 1999.