FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

JunBeom Yoo, Jong-Hoon Lee, Sehun Jeong, Sundeok Cha
Dependable Software Laboratory
Konkuk University, Republic of Korea
Contents

• Introduction

• Background
  – Function Block Diagram
  – PLC open
  – Verilog Programming

• FBDtoVerilog

• Case Study

• Conclusion & Future Work
FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

INTRODUCTION
Introduction

• Safety critical systems are using FBD(Function Block Diagram) to design software
  – It used PLC(Programmable Logic Controller) programming language in plant automation industry
  – These systems requires rigorous quality demonstration
    • Formal verification techniques
      – E.g.) Model Checking, Equivalence Checking

• Formal verification techniques have their own input format
  – E.g.) VIS: Verilog Program, SMV: SMV input or Verilog program
  – Requires translation from FBDs into tool specific input format
Introduction (Cont’d)

- **KNICS project** (Korea Nuclear Instrumentation & Control System)
  - Developed a new RPS (Reactor Protection System) for Korean nuclear power plants and implemented its software in FBDs
  - Our former researches: *FBD Verifier, PLC Verifier*
    - It used FBD format specific to POSCO ICT, which generated from *pSET*

- If some changes in the format of front-end,
  - It is difficult to keep consistency and correctness of the translator and verification tool
Introduction (Cont’d)

• CASE tool: *FBDtoVerilog*
  – It translates FBDs into a semantically equivalent Verilog Programs
    • Many verification techniques uses Verilog language
  – It uses standard input front-end format of FBD
    • Standard XML format of FBD (PLCopen)

• Case Study of *FBDtoVerilog*
  – Example: FBD of ‘th_X_Pretrip’ logic in KNICS project
    • Translate from FBD into Verilog program
    • Performed the formal verification techniques
FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

BACKGROUND
Background – Function Block Diagram

• FBD consists of an arbitrary number of function blocks
  – It visually expresses behavior of system
  – Blocks are wired together and execute sequentially

• Function blocks are defined in IEC 61131-3 standard
  – Defined all function blocks and 10 categories

![Function Block Diagram]
Background – PLCopen

• PLCopen is a vendor- and product-independent association

• PLCopen has defined an open interface between all different kinds of software tools
  – TC6 define the XML specification for IEC 61131-3 standard
  – Includes IL, ST, LD, FBD, SFC programming languages and other information
  – We used the XML specification of FBD programming language
Background – Verilog Programming

• Verilog program language
  – Hardware Description Languages used in Integrated Circuit design
  – Many verification and analysis techniques and tools use Verilog as an input language
  – We defined translation rules from FBD into Verilog

```verilog
module GATE_Example(clk, X0, X1, X2, Y0, Y1, Y2, Y3);

  // variable declaration
  input clk;
  input X0, X1, X2; // input variables
  output Y0, Y1, Y2, Y3; // output variables

  // assign statements
  assign Y0 = ~(X0 & X1 & X2);
  assign Y1 = ~(X0 | X1 | X2);
  assign Y2 = X0 ^ X1 ^ X2;
  assign Y3 = ~(X0 ^ X1 ^ X2);

  // always block
  always @(posedge clk) begin

  end
endmodule
```

A simple example of Verilog program
FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs
FBDtoVerilog

- CASE tool: **FBDtoVerilog**
  - Uses standard input format of FBD
    - Standard XML format of FBD (PLCopen)
  - Translate FBDs into a semantically equivalent Verilog Programs
FBDtoVerilog (Cont’d)

- FBDtoVerilog uses standard XML format of FBD proposed by PLCopen
  - We used XML specification of FBD
  - In addition, we used an *addData* element of the XML specification
    - It stores name of output that every single function block belongs to

```
  <FBD>
  + <inVariable localId="1" height="16" width="32">
  - <inVariable localId="2" height="16" width="32">
    <position x="0" y="0" />
  - <connectionPointOut>
    <relPosition x="32" y="8" />
  </connectionPointOut>
  <expression>k_X_Pretrip_Setpoint</expression>
  </inVariable>
  - <block instanceName="GE_INT" localId="3" typeName="GE_INT" executionOrderId="12" height="64" width="80">
    ......
  </block>
  <addData>
  - <data name="th_X_Pretrip" handleUnknown="discard">
    <BOOL />
  </data>
  </addData>
</FBD>
```

Part of FBD(XML) and *addData* element
**FBDtoVerilog (Cont’d)**

- **FBDtoVerilog** Translates from FBDs into Verilog programs
  - We defined translation rules from FBDs to Verilog Programs
- 13 rules for translation
  - 4 rules for Function Block
  - Others for block diagram

<table>
<thead>
<tr>
<th>Category</th>
<th>Rule Number</th>
<th>Constructs of FBD</th>
<th>Constructs of Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function block</td>
<td>1</td>
<td>FB = &lt;Name, IP, OP, BD&gt;</td>
<td>function [type of OP] name_of_function;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>... endfunction</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Ip₁ ∈ IP</td>
<td>input [type of IP] ip₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>Component_FBD</td>
<td>5</td>
<td>Component_FBD = &lt;FBs, T, I, O&gt;</td>
<td>module name_of_Component_FBD(list of I, list of O);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>... endmodule</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>v₁ ∈ I - Provided v₁ ∈ V_comp_FBD-O</td>
<td>input [type_of_v₁] v₁;</td>
</tr>
</tbody>
</table>

A part of translation rules
FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

CASE STUDY
Case Study

• Case Study for FBDtoVerilog
  – Applied to real case: ‘th_X_Pretrip’ logic
    • Logic in KNICS project

– Process of Case Study
  • Translate the FBD ‘th_X_Pretrip’ into Verilog program
  • Performed the formal verification techniques by using tools
Case Study (Cont’d)

• ‘th_X_Pretrip’
  – Logic in KNICS project
    • Consists of 8 function blocks
    • One input, one output and one internal output variables
      – f_X, th_X_Pretrip, th_Prev_X_Pretrip
    • 3 constant variables
      – k_X_Pretrip_Setpoint, k_X_Pretrip_Hys, k_Trip_Delay

‘th_X_Pretrip’ logic
Case Study (Cont’d)

- Translate into Verilog program by *FBDtoVerilog*
Case Study (Cont’d)

- Translated into Verilog program by *FBDtoVerilog*
  - 98 lines Verilog program
  - It covers 12 translation rules of the 13 rules
Case Study (Cont’d)

- Performed the formal verification techniques
Case Study (Cont’d)

- Planned to apply two verification techniques
  - SMV model checking is not performed
    - SMV model checker cannot read the Verilog code which translated by FBDtoVerilog
      - E.g.) Reuse of functions are not allowed
  - VIS equivalence checking
    - Two Verilog programs have same behavior
      - “Sequentially equivalent”
      - Between manually- and automatically-translated code
    - Validation of FBDtoVerilog
      - It shows correctness of FBDtoVerilog indirectly

SEKE 2011 Conference
FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

CONCLUSION & FUTURE WORK
Conclusion & Future Work

• A CASE Tool: FBDtoVerilog
  – It uses standard XML format of FBD as a front-end
  – It translates from FBDs into Verilog programs

  – Case Study: VIS equivalence checking
    • It validates correctness of FBDtoVerilog indirectly
  – Some issues to be improved
    • Translated code requires adaptation (Additional work for verification)
    • Translated code can be optimized

• Future Work
  – Implement the improved FBDtoVerilog
  – Perform the Case study about other logics in KNICS project
    • Bigger and more complicate cases
Thank You
Issues to be improved

Adaptation

• Requires additional works
  – Variable size set-up

Optimization

• Code can be optimized
  – Code for Function Blocks

---

<table>
<thead>
<tr>
<th>Translated Code</th>
<th>Post-processed Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>input a; input b;</td>
<td>input [0:6] a; input [0:6] b;</td>
</tr>
<tr>
<td>var = ADD(a, b);</td>
<td>var = ADD(a, b)</td>
</tr>
<tr>
<td>function ADD_INT;</td>
<td>function [0:6] ADD_INT;</td>
</tr>
<tr>
<td>input in1; input in2;</td>
<td>input [0:6] in1; input [0:6] in2;</td>
</tr>
<tr>
<td>begin</td>
<td>begin</td>
</tr>
<tr>
<td>ADD_INT = (in1 + in2);</td>
<td>ADD_INT = (in1 + in2);</td>
</tr>
<tr>
<td>end</td>
<td>end</td>
</tr>
<tr>
<td>endfunction</td>
<td>endfunction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Func</th>
<th>Translated Code</th>
<th>Optimized Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>input a; input b;</td>
<td>input a + b;</td>
</tr>
<tr>
<td></td>
<td>function ADD_INT; input in1; input in2; begin</td>
<td>function ADD_INT; input [0:6] in1; input [0:6] in2; begin</td>
</tr>
<tr>
<td></td>
<td>ADD_INT = (in1 + in2);</td>
<td>ADD_INT = (in1 + in2);</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>endfunction</td>
<td>endfunction</td>
</tr>
</tbody>
</table>
Translation Rules(1)

- Rules for Function block

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1 | $FB = \langle Name, IP, OP, BD \rangle$ | $function \ [type\_of\_OP] \ name\_of\_function; \ \cdots$  
|   |   | $\text{endfunction}$ |
| 2 | $ip_i \in IP$ | $\text{input} \ [type\_of\_OP] \ ip_i; \ \cdots$ |
| 3 | $BD$ (logical or arithmetic) | $\text{begin}$  
|   |   | $name\_of\_function = ip_0 \oplus ip_1 \oplus \cdots \oplus ip_n;$  
|   |   | $\text{end}$ |
| 4 | $BD$ (selection) \  
|   | - provided $IP = \{ k, ip_1, ip_2, \cdots, ip_n \}$ | $\text{begin}$  
|   |   | $name\_of\_function = (k == 0) \ ? \ ip_0 : \ (k == 1) \ ? \ ip_1 : \ \cdots$  
|   |   | $\quad (k == n) \ ? \ ip_n : \text{default};$  
|   |   | $\text{end}$ |
Translation Rules(2)

- **Rules for Component FBD**

<table>
<thead>
<tr>
<th></th>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
</table>
| 5 | $\text{Component_FBD} = < \text{FBs}, T, I, O >$ | $\text{module name_of_Component_FBD ( list_of_I , list_of_O );}$
|   |      | $\text{...}$
|   |      | $\text{endmodule}$ |
| 6 | $v_i \in I$
|   |      | - provided $v_i \in V_{\text{comp}_FBD-I}$ | $\text{input [type_of_v_i] v_i;}$ |
| 7 | $v_o \in O$
|   |      | - provided $v_o \in V_{\text{comp}_FBD-O}$ | $\text{output [type_of_v_o] v_o;}$
|   |      | $\text{wire [type_of_v_{\text{intermediate}}] v_{\text{intermediate}};}$
|   |      | $\text{...}$
|   |      | $\text{assign v_o = assignment_for_v_o_using_Verilog_function_calls;}$ |
| 8 | $v_o \in O$
|   |      | - provided $\exists v_i \in I \& i == o$
|   |      | - $v_i$ precedes $v_o$ | $\text{reg [type_of_v_o] v_o;}$
|   |      | $\text{initial v_o = initial_value_of_v_o;}$
|   |      | $\text{...}$
|   |      | $\text{always @(posedge clk) begin}$
|   |      | $\text{assignment_for_v_o;}$
|   |      | $\text{end}$ |
| 9 | $\text{function block } \in \text{FBs}$ | $\text{function ...}$
|   |      | $\text{definition_of_function_block}$
|   |      | $\text{endfunction}$ |
## Translation Rules (3)

- **Rules for System FBD**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System_FBD</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 10 | \( \text{System}_\text{FBD} = < \text{FBDs}, T, I, O > \) | \( \text{module} \text{name}_\text{of}_\text{System}_\text{FBD} (\text{list}_\text{of}_I, \text{list}_\text{of}_O); \)
|   |   | \( \text{endmodule} \) |
| 11 | \( v_i \in I \)
|   | - provided \( v_i \in V_{\text{sys}_\text{FBD}, I} \) | \( \text{input} [\text{type}_\text{of}_v] v_i; \) |
| 12 | \( v_o \in O \)
|   | - provided \( v_o \in V_{\text{sys}_\text{FBD}, O} \) | \( \text{output} [\text{type}_\text{of}_v] v_o; \)
|   |   | \( \text{wire} [\text{type}_\text{of}_{\text{intermediate}}] v_{\text{intermediate}}; \)
|   |   | \( \text{assign} v_o = \text{assignment}_\text{for}_v_o \text{using}_\text{Verilog}_\text{moudle}_\text{instantiations}; \) |
| 13 | \( v_o \in O \)
|   | - provided \( \exists v_i \in I \ & i == o \)
|   | - \( v_i \) precedes \( v_o \) | \( \text{reg} [\text{type}_\text{of}_v] v_o; \)
|   |   | \( \text{initial} v_o = \text{initial}_\text{value}_\text{of}_v_o; \)
|   |   | \( \text{always} @(\text{posedge} \text{clk}) \text{begin} \)
|   |   | \( \text{assignment}_\text{for}_v_o; \)
|   |   | \( \text{end} \) |