FBDtoVerilog 2.0
An automatic translation of FBD into Verilog to develop FPGA

Dong-Ah Lee\textsuperscript{\textdagger}, Eui-sub Kim\textsuperscript{\textdagger}, Junbeom Yoo\textsuperscript{\textdagger}, Jang-Soo Lee\textsuperscript{*}, Jong Gyun Choi\textsuperscript{*}

\textsuperscript{\textdagger}Konkuk University

\textsuperscript{*}Korea Atomic Energy Research Institute, Korea
Outline

1. Introduction
2. Background
   1. NuDE
   2. Function Block Diagram
3. FBDtoVerilog 2.0
   1. Translation of POU
   2. Pre-defined library
   3. Implementation issues
4. Case study
5. Conclusion and future work
Introduction (1/2)

• The nuclear industry modernizes existing analog I&C systems to digital I&C systems.
  – Software and network are parts of the systems.
  – Example: PLC (Programmable Logic Controller): Real-time controllers in nuclear RPSs (Reactor
    Protection Systems)

• Digital systems offer higher reliability, better plant performance and additional diagnostic
  capabilities.

• However, CCFs (Common Cause Failure) and security problems are rising in the field of the
  digital I&C systems in nuclear power plant. Furthermore, increasing complexity and
  maintenance cost are being brought up recently.
  – System’s diversity is one of solutions to prevent the threats.
Introduction (2/2)

- Using CPU-based controllers and FPGA-based controllers implements diversity.

- **Big challenges**
  - Software engineers in nuclear domain are not familiar with hardware implementation. Experience, knowledge and practice about developing PLC may be useless.
  - Safety certification is too costly.

- **Proposed methods**
  - This paper proposes an automatic translation of FBD, a programming language of PLC software, into behaviorally equivalent Verilog design.
Background (1/2)

- Nuclear Development Environment
- A formal methods based process for developing safety-critical software
- We are now extending the environment from PLC-based RPS development to FPGA-based RPS development
Background (2/2)

Function Block Diagram (FBD)

- IEC 61131–3 standard declared 5 programming languages for PLC
  - FBD, ST, LD, IL, SFC
- Sequential interconnections between functions and function blocks
  - Functions: No storable state
  - Function Blocks: Storable state
FBDtoVerilog 2.0

Translation rules

- FBDtoVerilog 2.0 translates user-defined FBD programs by a programmable organization unit (POU)
  - POU: function, function block, and program
  - Hierarchical organization

- Standard Fs/FBs are pre-translated POUs in the library.

A user-defined POU

A Verilog module
FBDtoVerilog 2.0

POUName

```
module [POUName] (rst, clk, pulse[, INPUT][, OUTPUT]);
input clk;
input rst;
input pulse;

INPUT:[input [BitSize] Name;]^
OUTPUT:[output [BitSize] Name;]^
FEEDBACKS:[output [BitSize] Name; reg [BitSize] Name;]^
CONSTANTS:[parameter [BitSize] Name = Value;]^
Connectors/continuations(CON):[wire [BitSize] Name;]^

POUS:[ModuleName ModuleName_[LocalId](rst, clk, pulse,
 INPUT[, INPUT]CON|ModuleNameWire^1 [, ModuleNameWire^2]);]^
Wiring POUS:[wire [BitSize] ModuleNameWire^2;]^

Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleNameWire^3;]^
always(@posedge rst or posedge clk or posedge pulse)
begin
if(rst) begin
    Output initializations:[OUTPUT <= initialValue;]^
end else if (clk) begin
end
if (pulse) begin
    Feedback assignments:[FEEDBACK <= ModuleNameWire^4;]^
end
end
endmodule
```
**POUName**

- **Line1**: defining interface of a module

```verilog
1: module [POUName] (rst, clk, pulse[, INPUT][, OUTPUT]);
2:   input clk;
3:   input rst;
4:   input pulse;
5:   
6:   INPUTS:[input [BitSize] Name;]^
7:   OUTPUTS:[output [BitSize] Name;]^
8:   FEEDBACKS:[output [BitSize] Name; reg [BitSize] Name;]^
9:   CONSTANTS:[parameter [BitSize] Name = Value;]^
10:  Connectors/continuations(CON):[wire [BitSize] Name;]^
11:  
12:  POUS:[ModuleName ModuleName_[LocalId](rst, clk, pulse,
13:      [, INPUT][CONModuleWire1] [, ModuleWire2]);]^
14:  Wiring POUS:[wire [BitSize] ModuleWire2];]
15:  
16:  Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire3;]^
17:  
18:  always(@posedge rst or posedge clk or posedge pulse)
19:    begin
20:      if(rst) begin
21:        Output initializations:[OUTPUT <= initialValue;]^
22:      end else if (clk) begin
23:        end
24:      if (pulse) begin
25:        Feedback assignments:[FEEDBACK <= ModuleWire4;]^
26:      end
27:    end
28:  end
29: endmodule
```
FBDtoVerilog 2.0

- Line 1–10: definition of input/output ports, feedback/constant variables, and connector/continuation pairs

```verilog
module [POUName] (rst, clk, pulse[, INPUT][][ OUTPUT][]);
  input clk;
  input rst;
  input pulse;
  input [BitSize] Name;
  output [BitSize] Name;
  output [BitSize] Name; reg [BitSize] Name;
  parameter [BitSize] Name = Value;
  wire [BitSize] Name;
  [ModuleName ModuleName] LocalId (rst, clk, pulse, [ INPUT]CON[ModuleWire][,] ModuleWire);]
  assign [CON|OUTPUT] = ModuleWire;
  always (@posedge rst or posedge clk or posedge pulse)
  begin
    if (rst) begin
      Output initializations: [OUTPUT <= initialValue;]
    end else if (clk) begin
    end
    if (pulse) begin
      Feedback assignments: [FEEDBACK <= ModuleWire;]
    end
  end
endmodule
```
POUName

- Line12–14: module calls to implement behaviors

```verilog
module [POUName] (rst, clk, pulse[, INPUT][, OUTPUT]);
  input clk;
  input rst;
  input pulse;

  INPUTS:[input [BitSize] Name];
  OUTPUTS:[output [BitSize] Name];
  FEEDBACKS:[output [BitSize] Name; reg [BitSize] Name];
  CONSTANTS:[parameter [BitSize] Name = Value;]
  Connectors/continuations(CON):[wire [BitSize] Name];

  POUK: [ModuleName ModuleName_[LocalId](rst, clk, pulse,
    [, INPUT][CON|ModuleWire][, ModuleWire]);]

  [wire [BitSize] ModuleWire];

  Wire to CON|OUTPUT: [assign [CON|OUTPUT] = ModuleWire;]

  always (@posedge rst or posedge clk or posedge pulse)
    begin
      if(rst) begin
        Output initializations:[OUTPUT <= initialValue;]
      end else if (clk) begin
        end
      if (pulse) begin
        Feedback assignments:[FEEDBACK <= ModuleWire;]
      end
    end
  endmodule
```
POUName

- Line 17: setting connection between module calls and ports/variables

```verilog
module [POUName] (rst, clk, pulse[, INPUT][, OUTPUT]);
  input clk;
  input rst;
  input pulse;
  input [BitSize] Name;
  output [BitSize] Name;
  wire [BitSize] Name;
  wire [BitSize] ModuleWire;
  assign [CON|OUTPUT] = ModuleWire;
  always (@posedge rst or posedge clk or posedge pulse)
    begin
      if(rst) begin
        Output initializations:
        end else if (clk) begin
        end
      if (pulse) begin
        Feedback assignments:
      end
    end
endmodule
```
POUName

- Line21–22: initiation of variables in the module using \texttt{rst} signals
- Line25–27: modeling cycles of FBD using \texttt{pulse} signals

```verilog
1: module [POUName] (rst, clk, pulse[, INPUT][, OUTPUT]);
2:     input clk;
3:     input rst;
4:     input pulse;
5:     
6:     INPUTS:[input [BitSize] Name;]^
7:     OUTPUTS:[output [BitSize] Name;]^
8:     FEEDBACKS:[output [BitSize] Name; reg [BitSize] Name;]^
9:     CONSTANTS:[parameter [BitSize] Name = Value;]^
10:    Connectors/continuations(CON):[wire [BitSize] Name;]^
11:    
12:    POUS:[ModuleName ModuleName_[LocalId](rst, clk, pulse,
13:        [, INPUT]CON|ModuleWire1][][, ModuleWire2]);^
14:    Wiring POUS:[wire [BitSize] ModuleWire2];^
15:    
16:    Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire3;]^
17:    
18:    always(@posedge rst or posedge clk or posedge pulse)
19:    begin
20:        if(rst) begin
21:            Output initializations:[OUTPUT <= initialValue;]^
22:        end else if (clk) begin
23:            end
24:        if (pulse) begin
25:            Feedback assignments:[FEEDBACK <= ModuleWire4;]^
26:        end
27:    end
28:    endmodule
```
FBDtoVerilog 2.0

Verilog library

- IEC 61131-3 Std. defines standard functions and function blocks (Std. Fs/FBs)
- Experts in KAERI have developed Verilog modules for the each Std. Fs/FBs
- Not only Std. Fs/FBs but also voting functions

![Diagram of Verilog modules](image)
FBDtoVerilog 2.0

Implementation Issues

- FBDtoVerilog 2.0 is an eclipse plug-in to be integrated into NuDE which is based on eclipse plug-in environment.
- Input: PLCopen TC6 XML version 2.01 scheme (de facto standard)
- Independent translator, but embedded into FBDEditor for convenience
Case study

- Two bistable trip logics in a RPS: FIX RISING TRIP and FIX FALLING TRIP DECISION
- Each logic has 5 inputs, 8 outputs, and 33 functions and function blocks
- Result
  - 3 modules: BP, FIX_RISING_TRIP, FIX_FALLING_TRIP
  - about 300 lines of Verilog code except the pre-translated modules in the Library
Case study

- Synthesis, compile, and P&R (Place and Route) for implementation of FPGA hardware
  - Design software: Libero Soc v11.1
  - Target hardware: ProASIC3 Start Kit
- No errors and one warnings resulted from the implementation
Netlist view: Level 0
Netlist view: Level 1 (FIX_RISING) (1)
Netlist view: Level 1 (FIX_RISING) (2)
Conclusion and future work

- FBDtoVerilog 2.0 translate FBD programs into Verilog language designs.
  - Using pre-translated Verilog library
- We identified behavioral equivalence manually.
  - Structural equivalence
  - Simulation of the designs

- Translation in development process needs more rigorous quality.
- We plan to perform V&V activities to demonstrate FBDtoVerilog 2.0’s quality in diverse methods.
  - Co-simulation between a FBD program and a Verilog design.
  - Safety/dependability case
  - Etc.
THANK YOU