The 5th International Conference on Information Science and Applications ICISA2014
May 6th – 9th, 2014, Ewha Womans University, Seoul, Korea

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

Dong-Ah Lee[+], Eui-sub Kim[+], Junbeom Yoo[+], Jang-Soo Lee[*], Jong Gyun Choi[*]

[+]Konkuk University

[*]Korea Atomic Energy Research Institute, Korea

Dependable Software Laboratory

KU KONKUK UNIVERSITY

Outline

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.

PLC (CPU-based) +

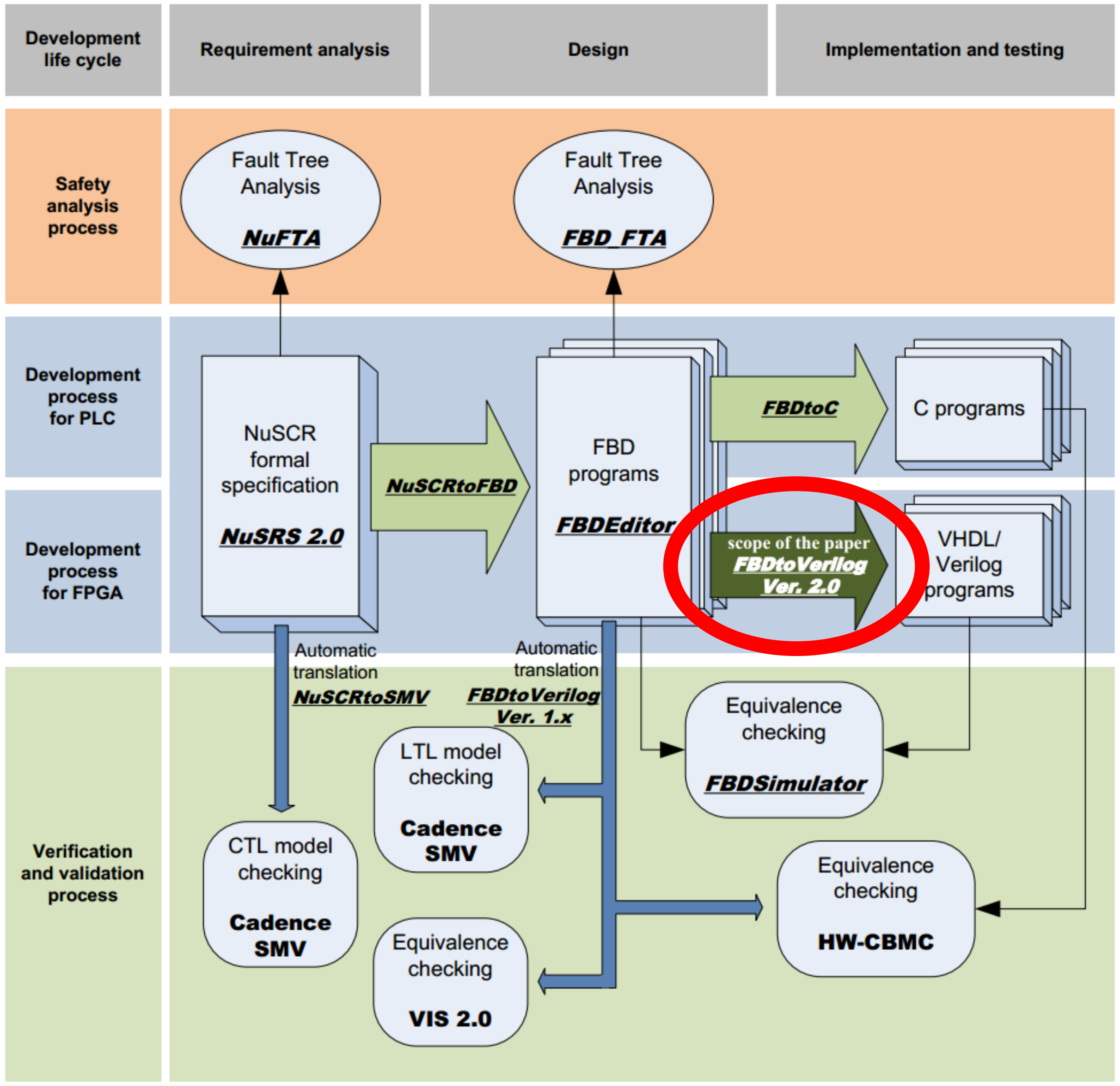
FPGA = DIVERSITY

- Big challenges
  - Software engineers in nuclear domain are not familiar with hardware implementation. Experience, knowledge and practice about developing PLC may be are 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)

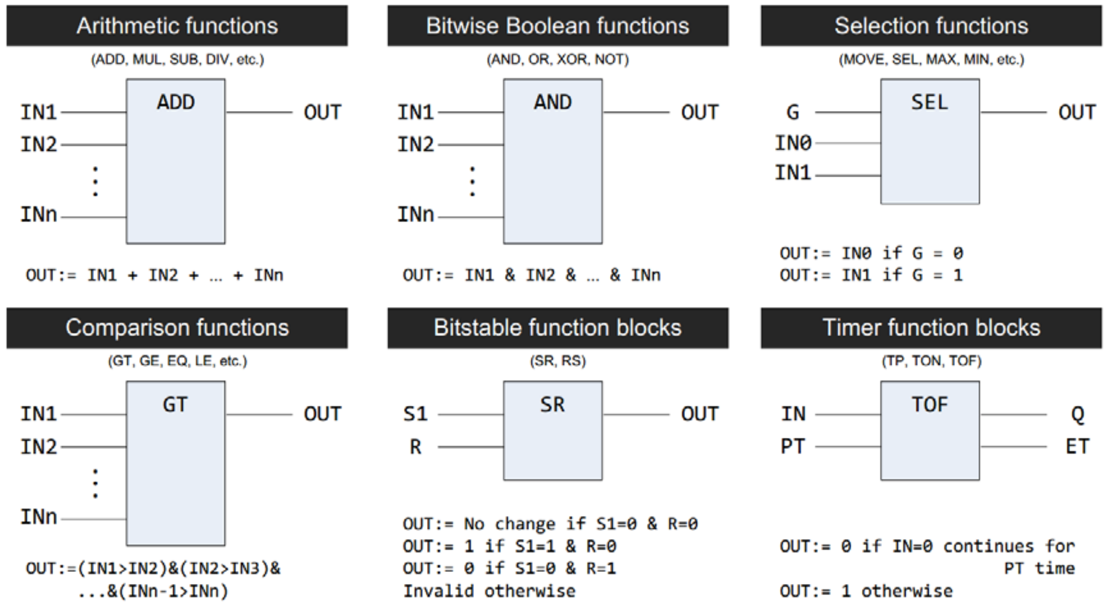
- Nuclear Development Environment
- A formal methods based process for developing safety-critical software
- We are now extending the environment from PLC-based RPS development to FPGA-based RPS development



Background (2/2)

Function Block Diagram (FBD)

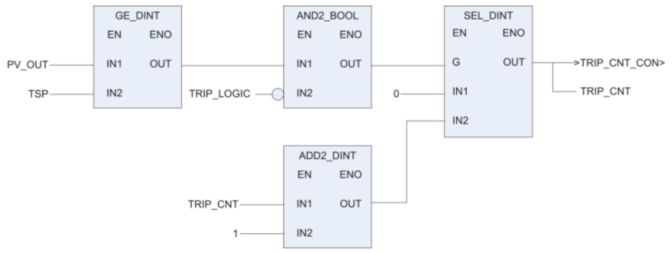
- IEC 61131-3 standard declared 5 programming languages for PLC
  - FBD, ST , LD, IL, SFC
- Sequential interconnections between functions and function blocks
  - Functions: No storable state
  - Function Blocks: Storable state

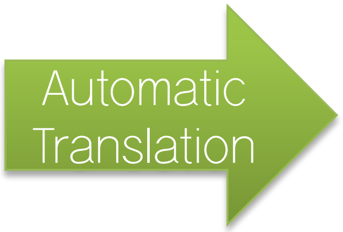


FBDtoVerilog 2.0

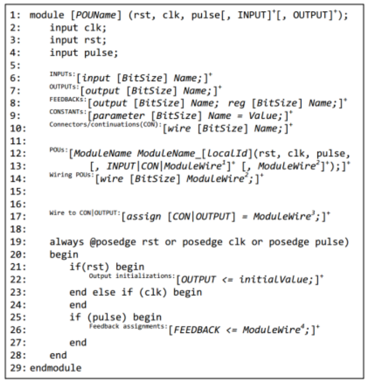
Translation rules

- 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.





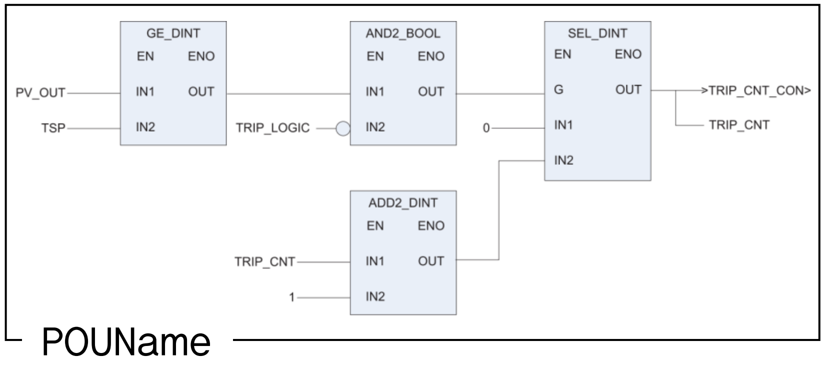
A user-defined POU



```
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;]+

    POUs:[ModuleName ModuleName_[LocalId](rst, clk, pulse,
            [, INPUT|CON|ModuleWire1]+ [, ModuleWire2]+);]+
    Wiring POUs:[wire [BitSize] ModuleWire2;]+


    Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire3;]+

    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 <= ModuleWire4;]+
        end
    end
endmodule
```

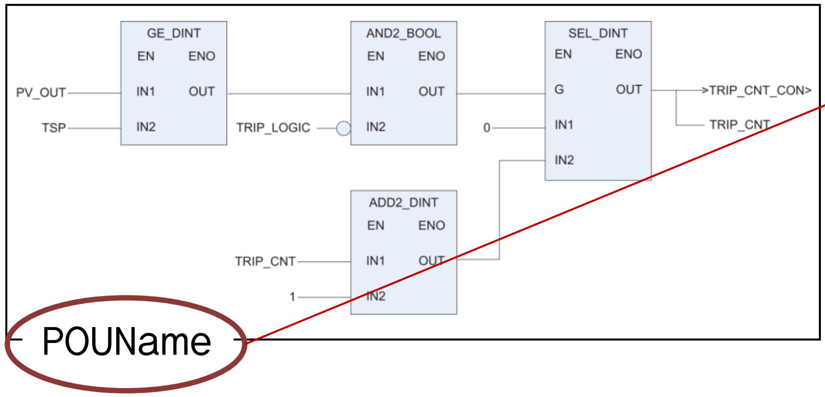
A Verilog module

FBDtoVerilog 2.0

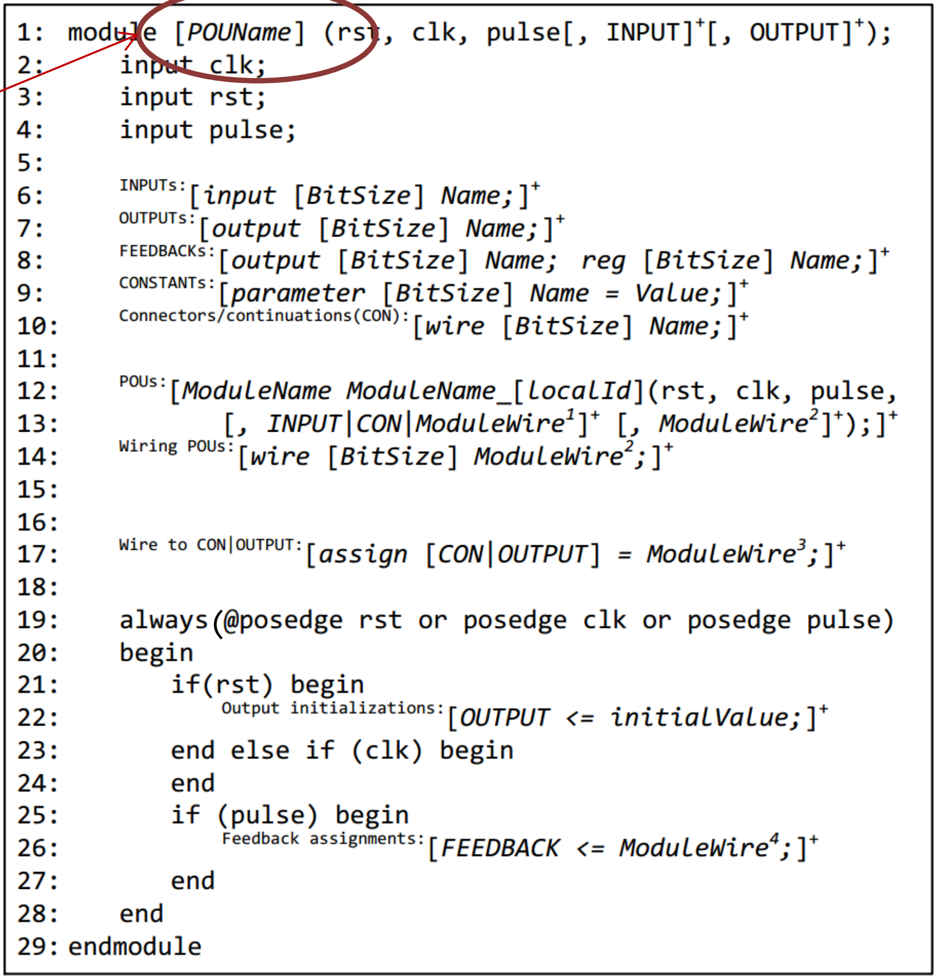


```
module [POUName] (rst, clk, pulse[, INPUT]^+[, OUTPUT]^+);
    input clk;
    input rst;
    input pulse;

    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;]^+

    POUs:[ModuleName ModuleName_[localId](rst, clk, pulse,
         [, INPUT|CON|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
    Wiring POUs:[wire [BitSize] ModuleWire^2;]^+


    Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^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 <= ModuleWire^4;]^+
        end
    end
endmodule
```

FBDtoVerilog 2.0

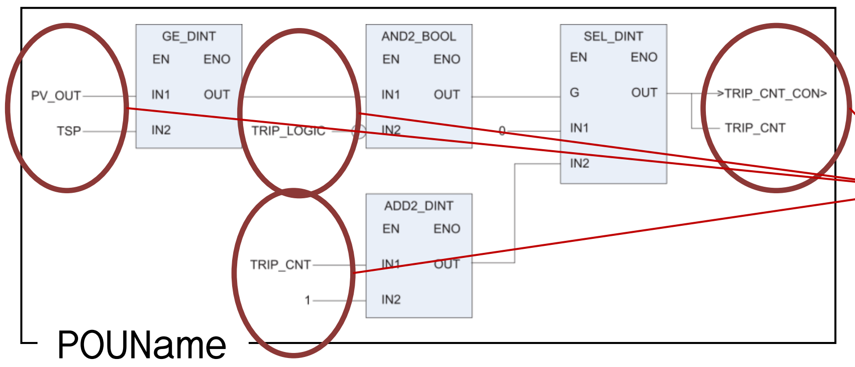


- Line1 : defining interface of a module



```
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|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
14:     Wiring POUs:[wire [BitSize] ModuleWire^2;]^+
15:
16:
17:     Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^3;]^+
18:
19:     always(@posedge rst or posedge clk or posedge pulse)
20:     begin
21:         if(rst) begin
22:             Output initializations:[OUTPUT <= initialValue;]^+
23:         end else if (clk) begin
24:         end
25:         if (pulse) begin
26:             Feedback assignments:[FEEDBACK <= ModuleWire^4;]^+
27:         end
28:     end
29: endmodule
```

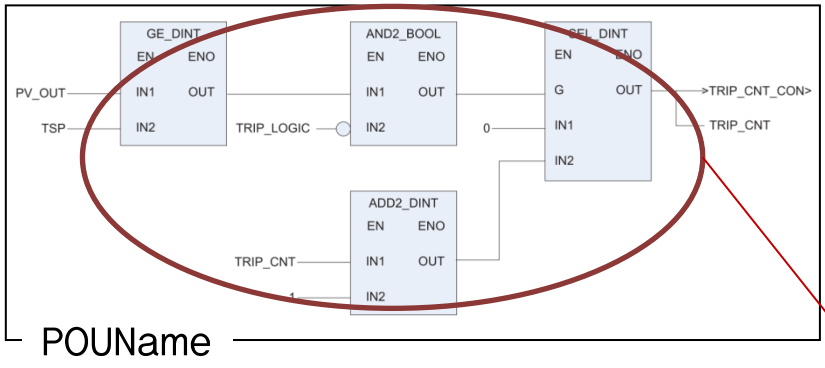
FBDtoVerilog 2.0



```
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|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
14:     Wiring POUs:[wire [BitSize] ModuleWire^2;]^+
15:
16:
17:     Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^3;]^+
18:
19:     always(@posedge rst or posedge clk or posedge pulse)
20:     begin
21:         if(rst) begin
22:             Output initializations:[OUTPUT <= initialValue;]^+
23:         end else if (clk) begin
24:         end
25:         if (pulse) begin
26:             Feedback assignments:[FEEDBACK <= ModuleWire^4;]^+
27:         end
28:     end
29: endmodule
```

- Line1-10: definition of input/output ports, feedback/constant variables, and connector/continuation pairs

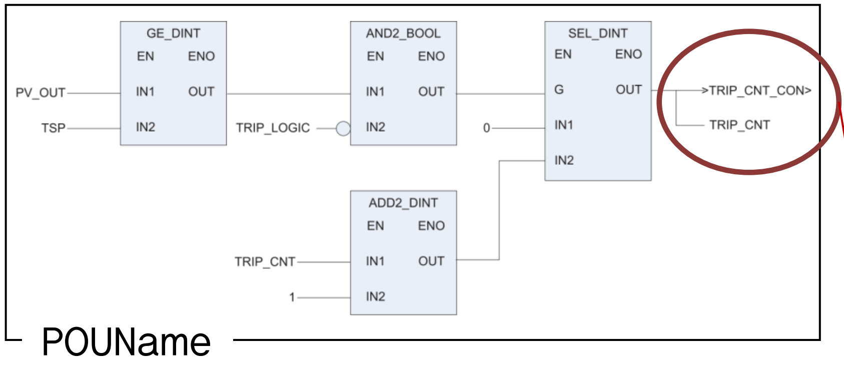
FBDtoVerilog 2.0



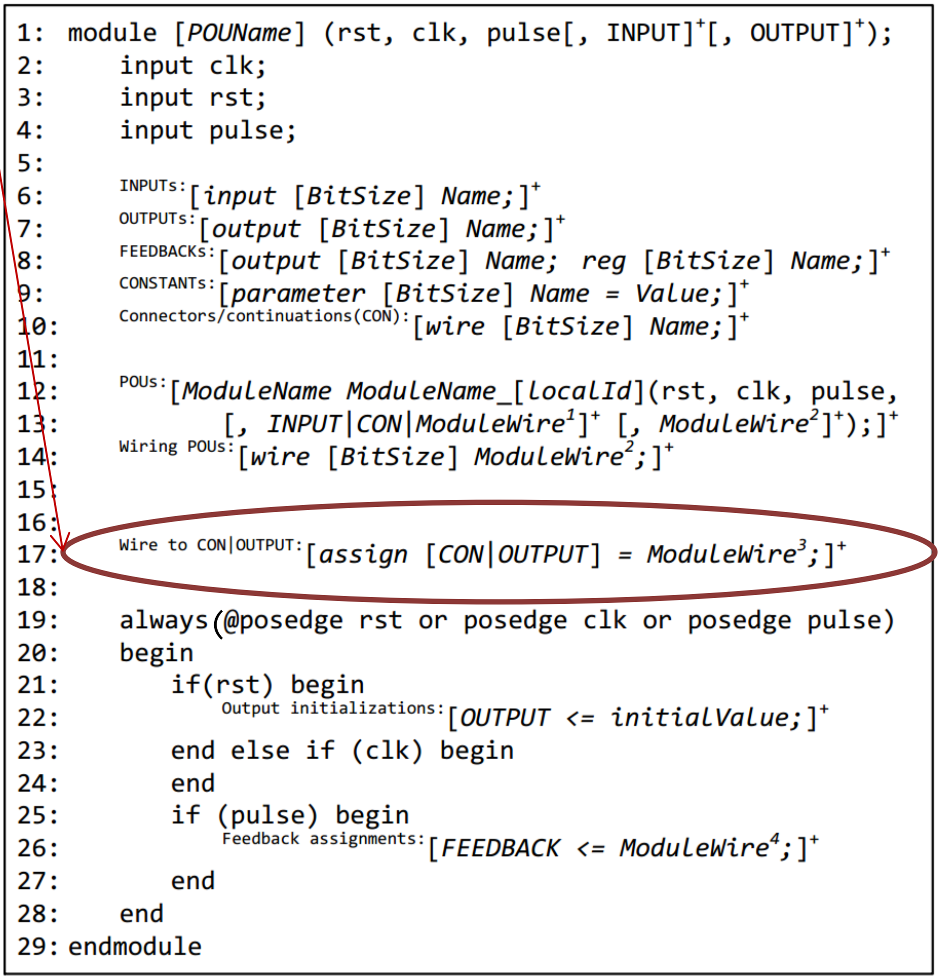
- Line12–14 : module calls to implement behaviors

```
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|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
14:     Wiring POUs:[wire [BitSize] ModuleWire^2;]^+
15:
16:
17:     Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^3;]^+
18:
19:     always(@posedge rst or posedge clk or posedge pulse)
20:     begin
21:         if(rst) begin
22:             Output initializations:[OUTPUT <= initialValue;]^+
23:         end else if (clk) begin
24:         end
25:         if (pulse) begin
26:             Feedback assignments:[FEEDBACK <= ModuleWire^4;]^+
27:         end
28:     end
29: endmodule
```

FBDtoVerilog 2.0

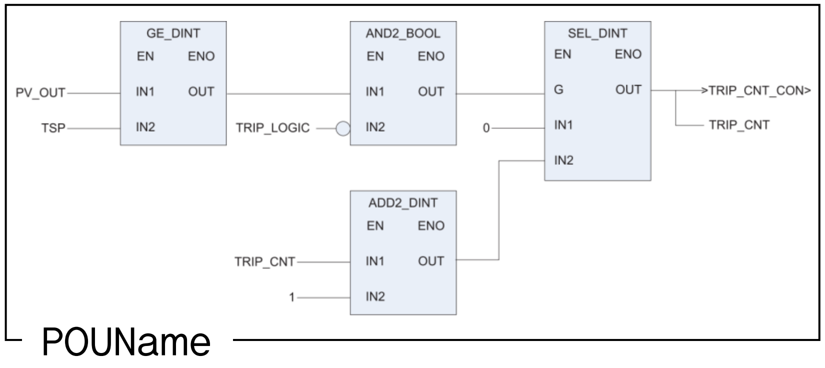


- Line17 : setting connection between module calls and ports/variables



```
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|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
14:     Wiring POUs:[wire [BitSize] ModuleWire^2;]^+
15:
16:
17:     Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^3;]^+
18:
19:     always(@posedge rst or posedge clk or posedge pulse)
20:     begin
21:         if(rst) begin
22:             Output initializations:[OUTPUT <= initialValue;]^+
23:         end else if (clk) begin
24:         end
25:         if (pulse) begin
26:             Feedback assignments:[FEEDBACK <= ModuleWire^4;]^+
27:         end
28:     end
29: endmodule
```

FBDtoVerilog 2.0



```
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|ModuleWire^1]^+ [, ModuleWire^2]^+);]^+
14:     Wiring POUs:[wire [BitSize] ModuleWire^2;]^+
15:
16:
17:     Wire to CON|OUTPUT:[assign [CON|OUTPUT] = ModuleWire^3;]^+
18:
19:     always(@posedge rst or posedge clk or posedge pulse)
20:     begin
21:         if(rst) begin
22:             Output initializations:[OUTPUT <= initialValue;]^+
23:         end else if (clk) begin
24:         end
25:         if (pulse) begin
26:             Feedback assignments:[FEEDBACK <= ModuleWire^4;]^+
27:         end
28:     end
29: endmodule
```

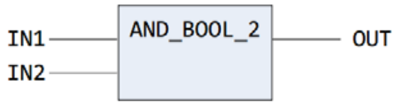
- Line21–22: initiation of variables in the module using `rst` signals
- Line25–27: modeling cycles of FBD using `pulse` signals

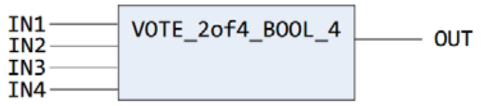
FBDtoVerilog 2.0

Verilog library

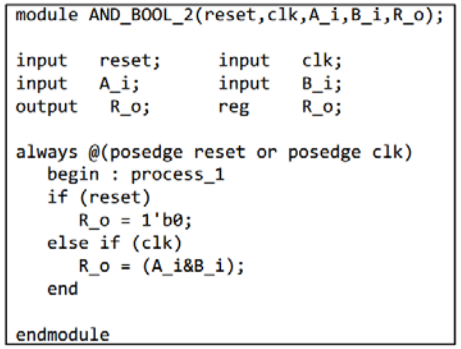
- IEC 61131-3 Std. defines standard functions and function blocks (Std. Fs/FBs)
- Experts in KAERI have developed Verilog modules for the each Std. Fs/FBs
- Not only Std. Fs/FBs but also voting functions

Fs/FBs



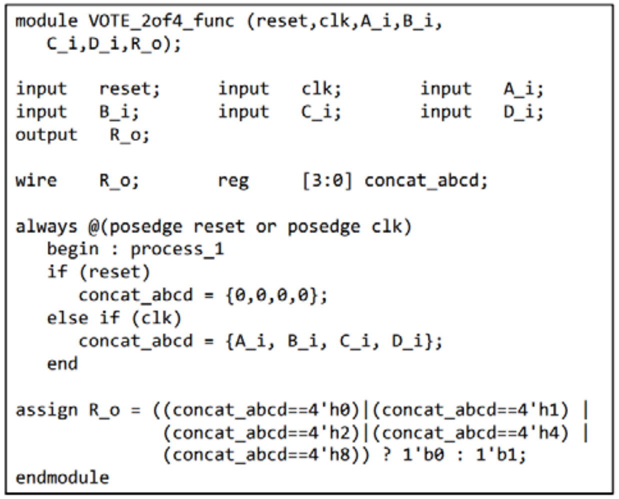


Verilog Modules in Library



```
module AND_BOOL_2(reset,clk,A_i,B_i,R_o);

input   reset;      input   clk;
input   A_i;        input   B_i;
output   R_o;       reg     R_o;

always @(posedge reset or posedge clk)
   begin : process_1
   if (reset)
      R_o = 1'b0;
   else if (clk)
      R_o = (A_i&B_i);
   end

endmodule
```

(a) The AND module with 2 inputs in Verilog



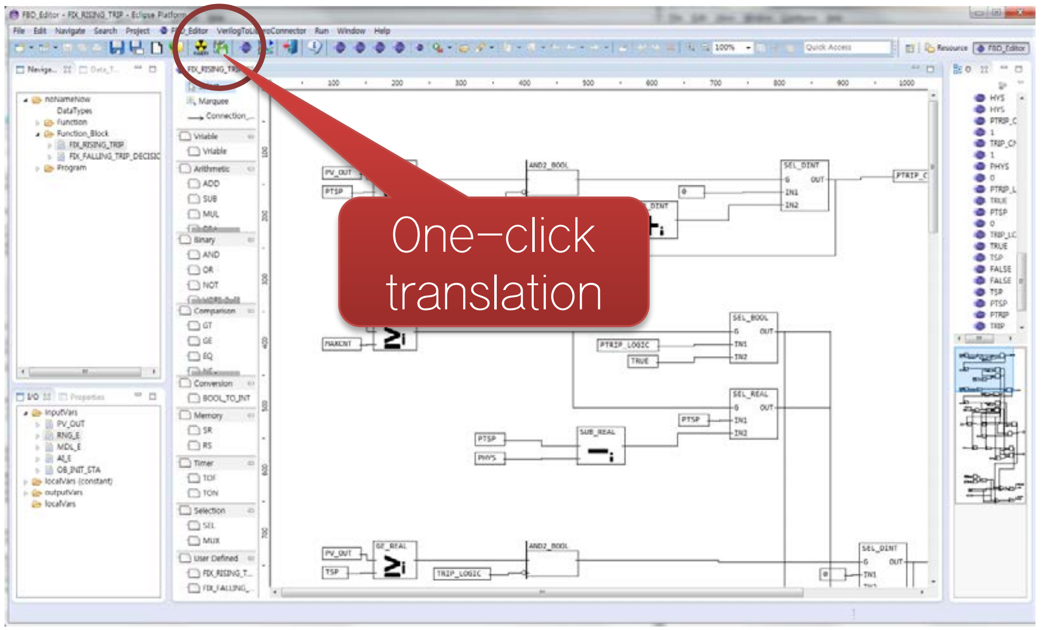
```
module VOTE_2of4_func (reset,clk,A_i,B_i,
   C_i,D_i,R_o);

input   reset;      input   clk;        input   A_i;
input   B_i;        input   C_i;        input   D_i;
output   R_o;

wire    R_o;        reg     [3:0] concat_abcd;

always @(posedge reset or posedge clk)
   begin : process_1
   if (reset)
      concat_abcd = {0,0,0,0};
   else if (clk)
      concat_abcd = {A_i, B_i, C_i, D_i};
   end

assign R_o = ((concat_abcd==4'h0)|(concat_abcd==4'h1) |
              (concat_abcd==4'h2)|(concat_abcd==4'h4) |
              (concat_abcd==4'h8)) ? 1'b0 : 1'b1;
endmodule
```

(b) The 2 out of 4 voting module with 4 inputs in Verilog

FBDtoVerilog 2.0

Implementation Issues

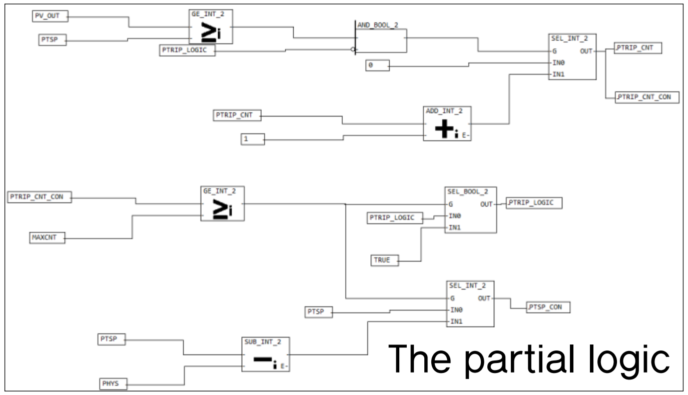
- FBDtoVerilog 2.0 is an eclipse plug-in to be integrated into NuDE which is based on eclipse plug-in environment.
- Input: PLCopen TC6 XML version 2.01 scheme (*de facto* standard)
- Independent translator, but embedded into FBDEditor for convenience



Case study

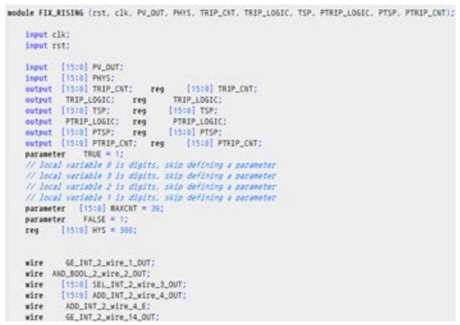
- Two bistable trip logics in a RPS: FIX RISING TRIP and FIX FALLING TRIP DECISION
- Each logic has 5 inputs, 8 outputs, and 33 functions and function blocks
- Result
  - 3 modules: BP, FIX_RISING_TRIP, FIX_FALLING_TRIP
  - about 300 lines of Verilog code

    except the pre-translated modules in the Library



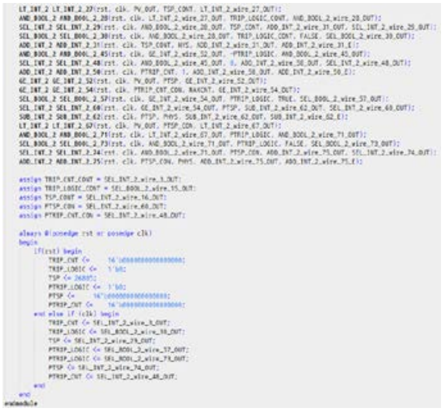
The partial logic

FBDtoVerilog 2.0



```
module FIX_RISING (rst, clk, PV_OUT, PHYS, TRIP_CNT, TRIP_LOGIC, TSP, PTRIP_LOGIC, PTSP, PTRIP_CNT);

    input clk;
    input rst;

    input   [15:0] PV_OUT;
    input   [15:0] PHYS;
    output  [15:0] TRIP_CNT;    reg     [15:0] TRIP_CNT;
    output   TRIP_LOGIC;    reg      TRIP_LOGIC;
    output  [15:0] TSP;     reg     [15:0] TSP;
    output   PTRIP_LOGIC;   reg      PTRIP_LOGIC;
    output  [15:0] PTSP;    reg     [15:0] PTSP;
    output  [15:0] PTRIP_CNT;   reg     [15:0] PTRIP_CNT;
    parameter    TRUE = 1;
    // local variable 0 is digits, skip defining a parameter
    // local variable 3 is digits, skip defining a parameter
    // local variable 2 is digits, skip defining a parameter
    // local variable 1 is digits, skip defining a parameter
    parameter   [15:0] MAXCNT = 30;
    parameter    FALSE = 1;
    reg     [15:0] HYS = 300;


    wire     GE_INT_2_wire_1_OUT;
    wire  AND_BOOL_2_wire_2_OUT;
    wire   [15:0] SEL_INT_2_wire_3_OUT;
    wire   [15:0] ADD_INT_2_wire_4_OUT;
    wire     ADD_INT_2_wire_4_E;
    wire     GE_INT_2_wire_14_OUT;
```

…

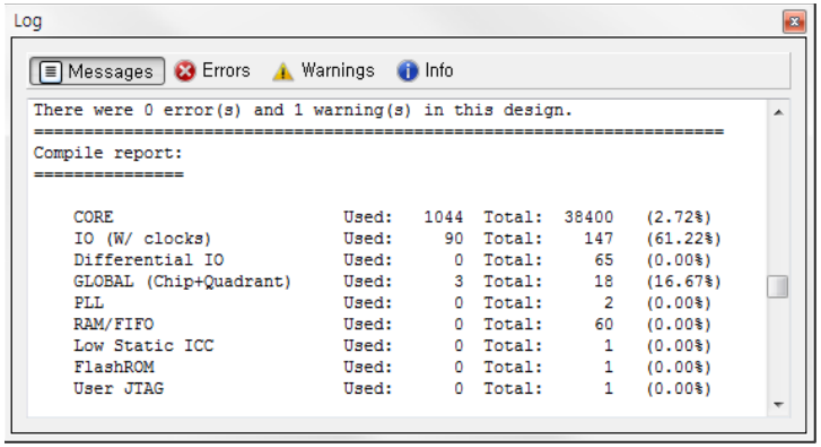


```
    LT_INT_2 LT_INT_2_27(rst, clk, PV_OUT, TSP_CONT, LT_INT_2_wire_27_OUT);
    AND_BOOL_2 AND_BOOL_2_28(rst, clk, LT_INT_2_wire_27_OUT, TRIP_LOGIC_CONT, AND_BOOL_2_wire_28_OUT);
    SEL_INT_2 SEL_INT_2_29(rst, clk, AND_BOOL_2_wire_28_OUT, TSP_CONT, ADD_INT_2_wire_31_OUT, SEL_INT_2_wire_29_OUT);
    SEL_BOOL_2 SEL_BOOL_2_30(rst, clk, AND_BOOL_2_wire_28_OUT, TRIP_LOGIC_CONT, FALSE, SEL_BOOL_2_wire_30_OUT);
    ADD_INT_2 ADD_INT_2_31(rst, clk, TSP_CONT, HYS, ADD_INT_2_wire_31_OUT, ADD_INT_2_wire_31_E);
    AND_BOOL_2 AND_BOOL_2_45(rst, clk, GE_INT_2_wire_52_OUT, ~PTRIP_LOGIC, AND_BOOL_2_wire_45_OUT);
    SEL_INT_2 SEL_INT_2_48(rst, clk, AND_BOOL_2_wire_45_OUT, 0, ADD_INT_2_wire_50_OUT, SEL_INT_2_wire_48_OUT);
    ADD_INT_2 ADD_INT_2_50(rst, clk, PTRIP_CNT, 1, ADD_INT_2_wire_50_OUT, ADD_INT_2_wire_50_E);
    GE_INT_2 GE_INT_2_52(rst, clk, PV_OUT, PTSP, GE_INT_2_wire_52_OUT);
    GE_INT_2 GE_INT_2_54(rst, clk, PTRIP_CNT_CON, MAXCNT, GE_INT_2_wire_54_OUT);
    SEL_BOOL_2 SEL_BOOL_2_57(rst, clk, GE_INT_2_wire_54_OUT, PTRIP_LOGIC, TRUE, SEL_BOOL_2_wire_57_OUT);
    SEL_INT_2 SEL_INT_2_60(rst, clk, GE_INT_2_wire_54_OUT, PTSP, SUB_INT_2_wire_62_OUT, SEL_INT_2_wire_60_OUT);
    SUB_INT_2 SUB_INT_2_62(rst, clk, PTSP, PHYS, SUB_INT_2_wire_62_OUT, SUB_INT_2_wire_62_E);
    LT_INT_2 LT_INT_2_67(rst, clk, PV_OUT, PTSP_CON, LT_INT_2_wire_67_OUT);
    AND_BOOL_2 AND_BOOL_2_71(rst, clk, LT_INT_2_wire_67_OUT, PTRIP_LOGIC, AND_BOOL_2_wire_71_OUT);
    SEL_BOOL_2 SEL_BOOL_2_73(rst, clk, AND_BOOL_2_wire_71_OUT, PTRIP_LOGIC, FALSE, SEL_BOOL_2_wire_73_OUT);
    SEL_INT_2 SEL_INT_2_74(rst, clk, AND_BOOL_2_wire_71_OUT, PTSP_CON, ADD_INT_2_wire_75_OUT, SEL_INT_2_wire_74_OUT);
    ADD_INT_2 ADD_INT_2_75(rst, clk, PTSP_CON, PHYS, ADD_INT_2_wire_75_OUT, ADD_INT_2_wire_75_E);

    assign TRIP_CNT_CONT = SEL_INT_2_wire_3_OUT;
    assign TRIP_LOGIC_CONT = SEL_BOOL_2_wire_15_OUT;
    assign TSP_CONT = SEL_INT_2_wire_16_OUT;
    assign PTSP_CON = SEL_INT_2_wire_60_OUT;
    assign PTRIP_CNT_CON = SEL_INT_2_wire_48_OUT;

    always @(posedge rst or posedge clk)
    begin
        if(rst) begin
            TRIP_CNT <=     16'b0000000000000000;
            TRIP_LOGIC <=   1'b0;
            TSP <= 26805;
            PTRIP_LOGIC <=  1'b0;
            PTSP <=     16'b0000000000000000;
            PTRIP_CNT <=    16'b0000000000000000;
        end else if (clk) begin
            TRIP_CNT <= SEL_INT_2_wire_3_OUT;
            TRIP_LOGIC <= SEL_BOOL_2_wire_30_OUT;
            TSP <= SEL_INT_2_wire_29_OUT;
            PTRIP_LOGIC <= SEL_BOOL_2_wire_57_OUT;
            PTRIP_LOGIC <= SEL_BOOL_2_wire_73_OUT;
            PTSP <= SEL_INT_2_wire_74_OUT;
            PTRIP_CNT <= SEL_INT_2_wire_48_OUT;
        end
    end
endmodule
```

+ Library

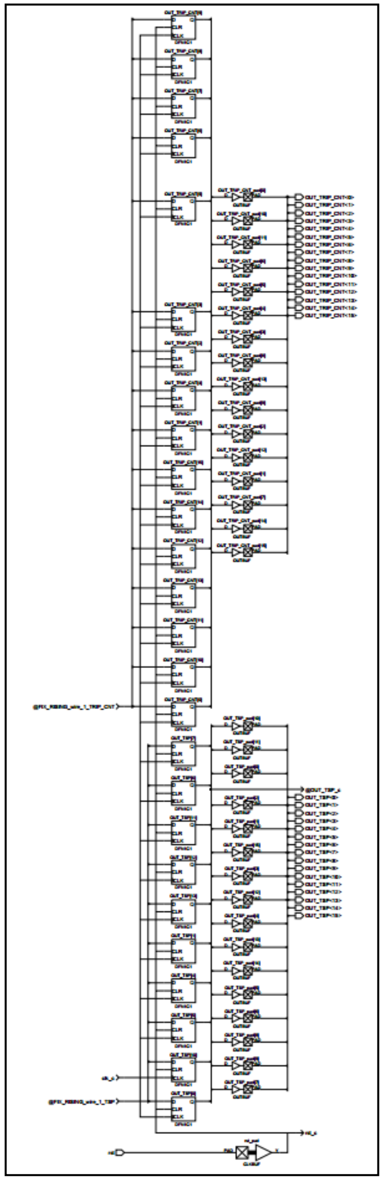
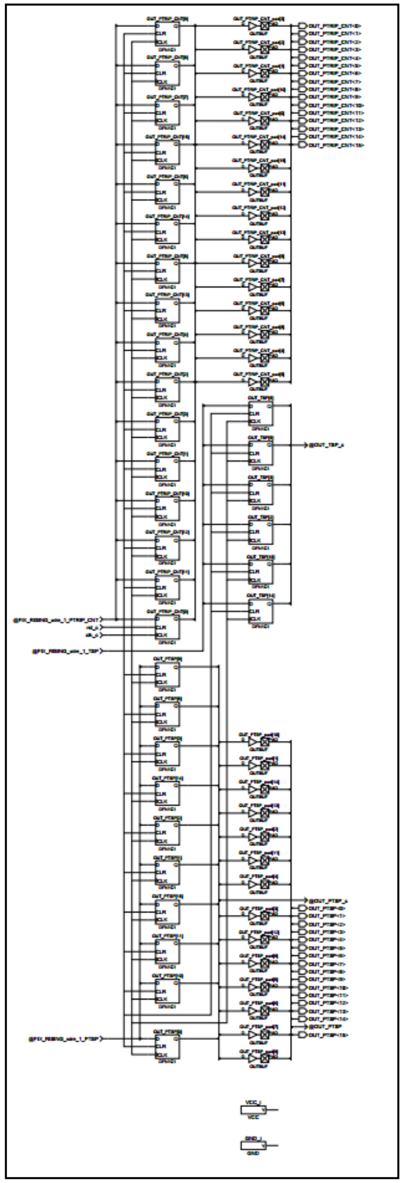
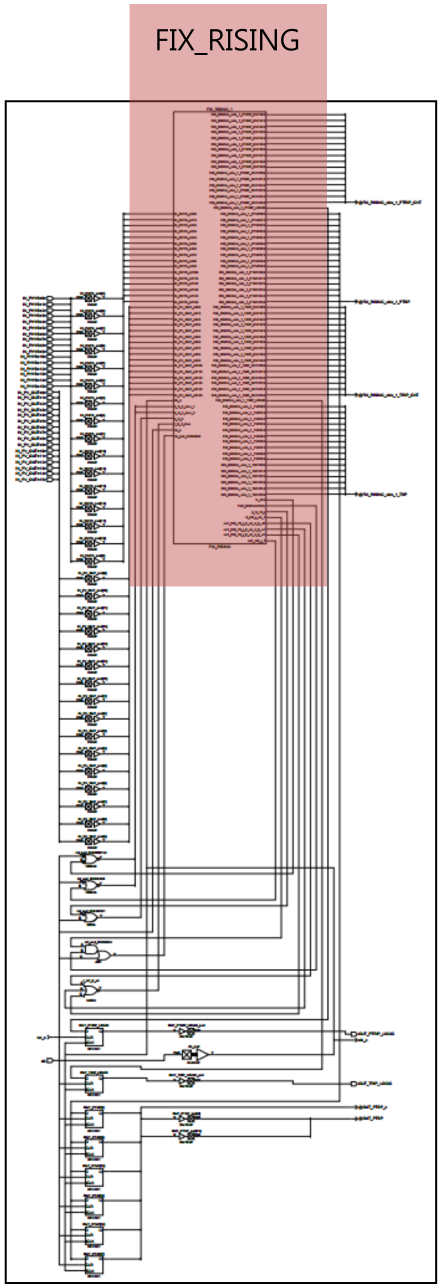
Case study

- 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

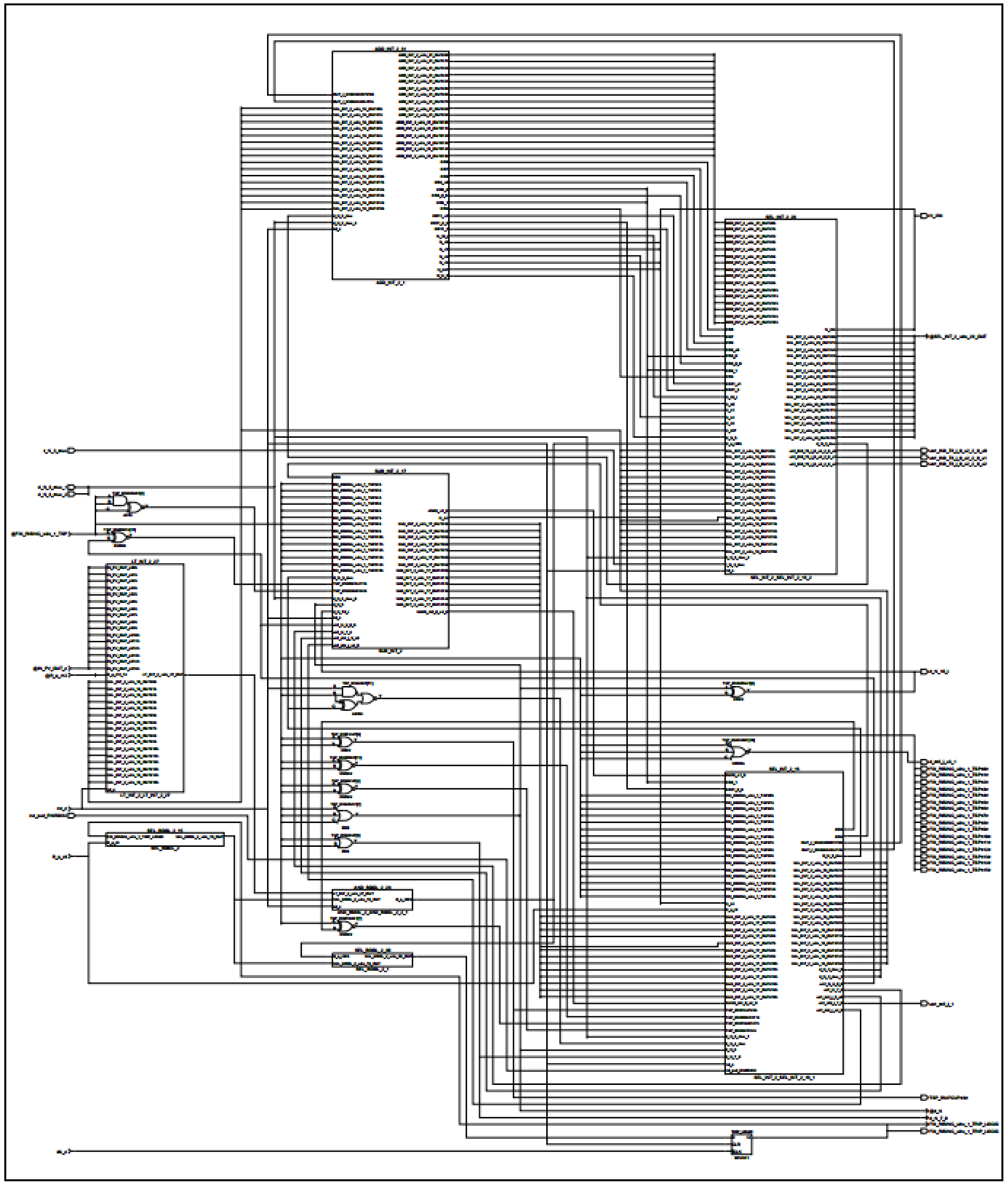


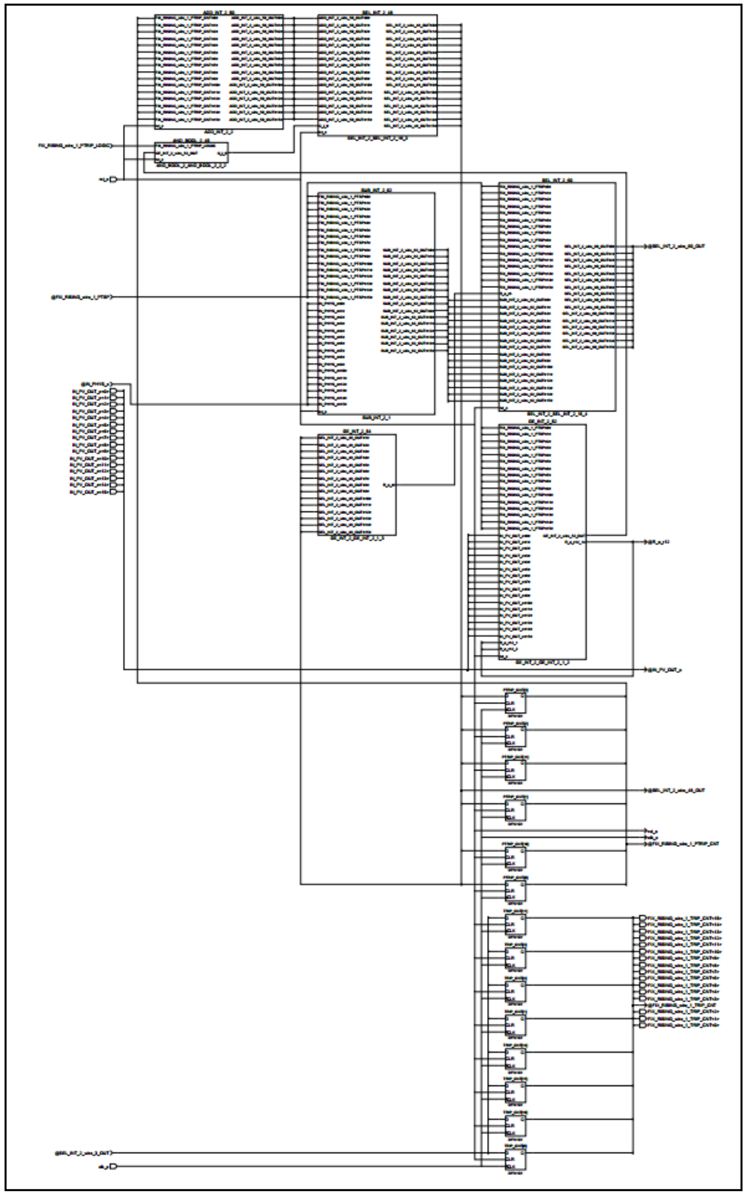
```
There were 0 error(s) and 1 warning(s) in this design.
=====================================================================
Compile report:
===============

    CORE                       Used:   1044  Total:  38400   (2.72%)
    IO (W/ clocks)             Used:     90  Total:    147   (61.22%)
    Differential IO            Used:      0  Total:     65   (0.00%)
    GLOBAL (Chip+Quadrant)     Used:      3  Total:     18   (16.67%)
    PLL                        Used:      0  Total:      2   (0.00%)
    RAM/FIFO                   Used:      0  Total:     60   (0.00%)
    Low Static ICC             Used:      0  Total:      1   (0.00%)
    FlashROM                   Used:      0  Total:      1   (0.00%)
    User JTAG                  Used:      0  Total:      1   (0.00%)
```

Netlist view： Level 0

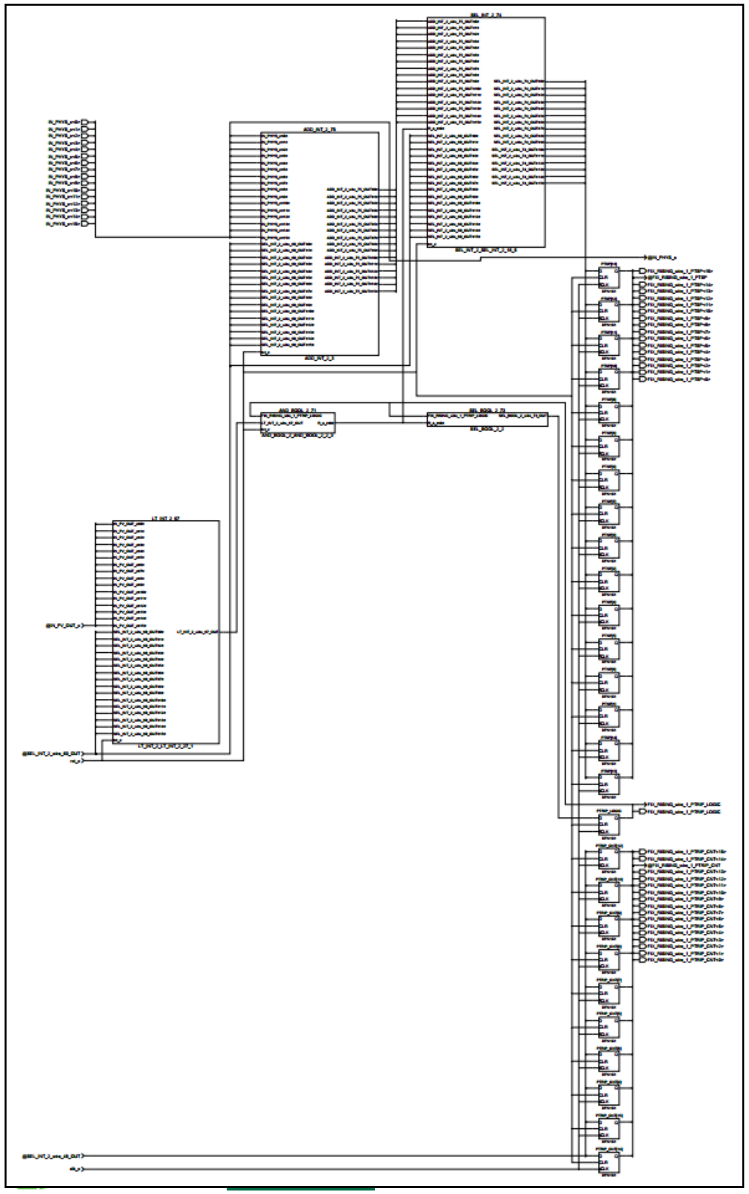
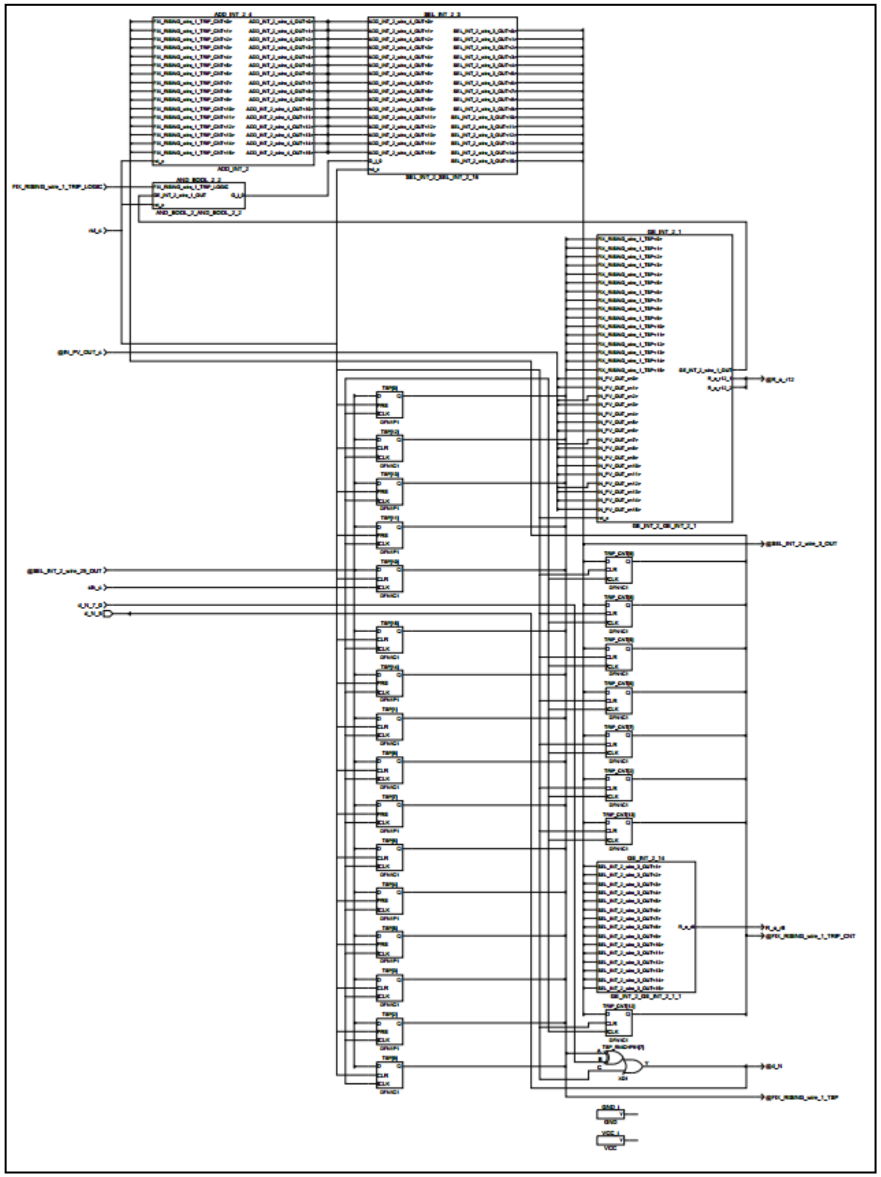




Netlist view: Level 1 (FIX_RISING) (1)





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