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

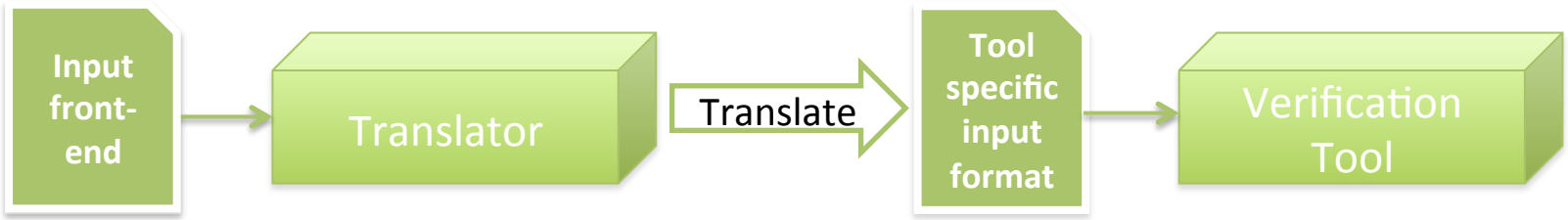
- Safety critical systems are using FBD(Function Block Diagram) to design software
  - It used PLC(Programmable Logic Controller) programming language in plant automation industry
  - These systems requires rigorous quality demonstration
    - Formal verification techniques
      - E.g.) Model Checking,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

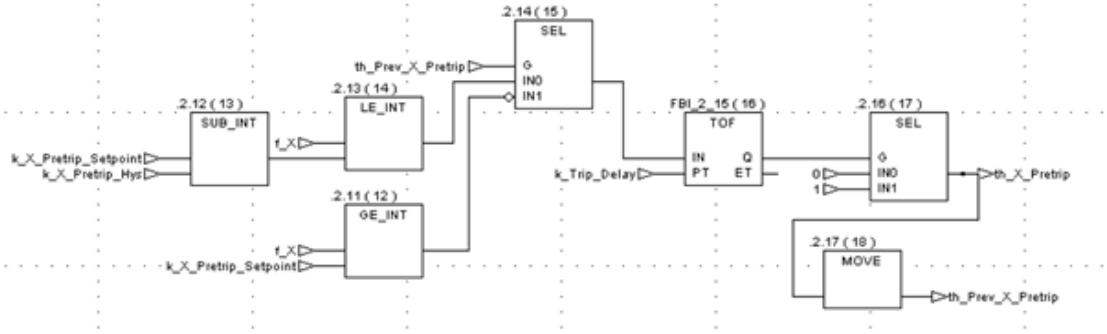
FBDtoVerilog: A Vendor-Independent Translation from FBDs into Verilog Programs

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

```
1   module GATE_Example(clk, X0, X1, X2, Y0, Y1, Y2, Y3);
2       /* variable declaration */
3       input clk;
4       input X0, X1, X2;       // input variables
5       output Y0, Y1, Y2, Y3;  // output variables
6
7       /* assign statements */
8       assign Y0 = ~(X0 & X1 & X2);
9       assign Y1 = ~(X0 | X1 | X2);
10      assign Y2 = X0 ^ X1 ^ X2);
11      assign Y3 = ~(X0 ^ X1 ^ X2);
12
13      always @(posedge clk) begin
```

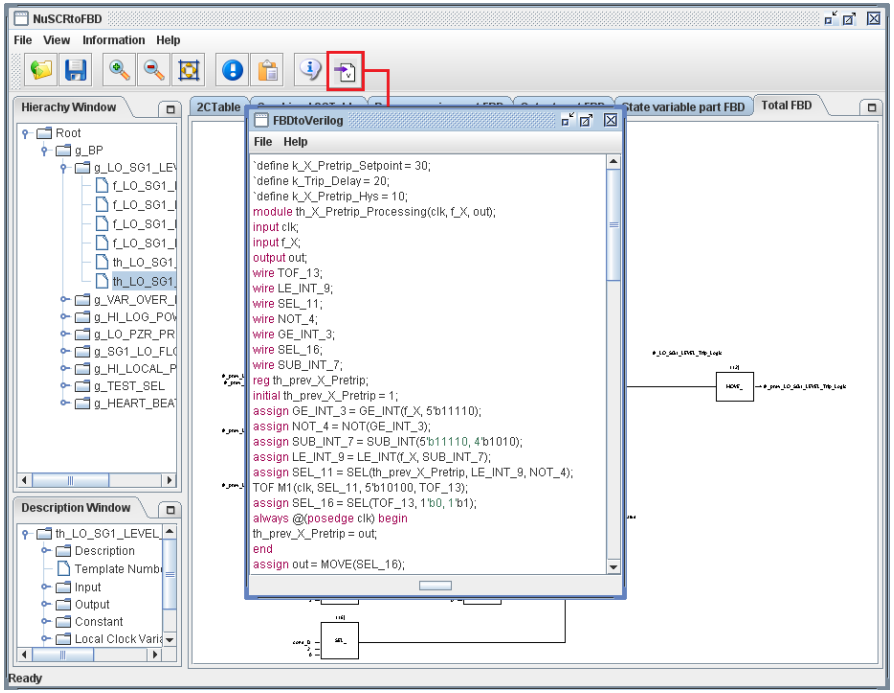
A simple example of Verilog program

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

FBDTOVERILOG

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">
                                        ......
    - <addData>
      - <data name="th_X_Pretrip" handleUnknown="discard">
          <BOOL />
        </data>
      </addData>
    </block>
```

addData element

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

Category	Rule Number	Constructs of FBD	Constructs of Verilog
Function block	1	FB = <*Name, IP, OP, BD*>	*function* [type of *OP*] *name_of_function;* *…* *endfunction*
	2	$Ip_1 \in IP$	*input* [type of *IP*] ip_1
……			
Component_FBD	5	Component_FBD = <FBs, T, I, O>	*module name_of_Component_FBD(list of I, list of O);* *…* *endmodule*
	6	$v_1 \in I$ - Provided $v_1 \in V_{comp\_FBD\text{-}O}$	*input* [*type_of_*v_1] v_1;
……			

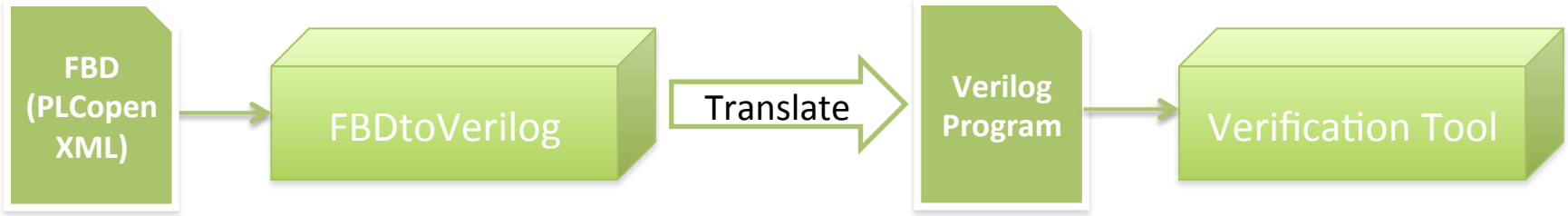
A part of translation rules

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

CASE STUDY

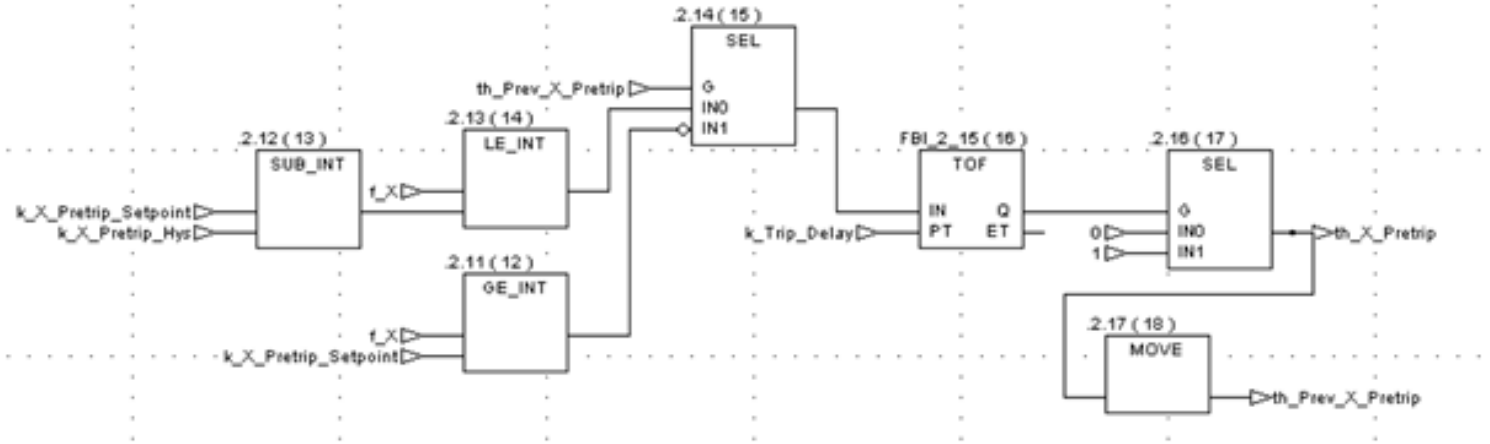
Case Study

- 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)

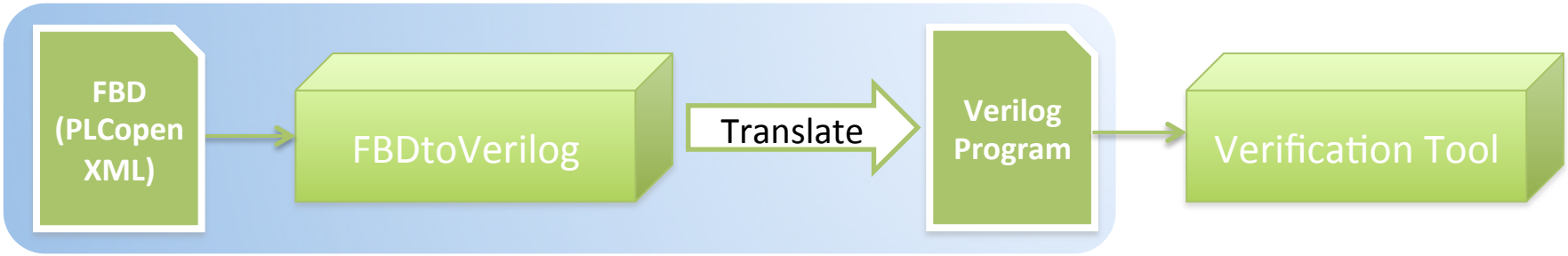
- *'th_X_Pretrip'*
  - Logic in KNICS project
    - Consists of 8 function blocks
    - One input, one output and one internal output variables
      - f_X, th_X_Pretrip, th_Prev_X_Pretrip
    - 3 constant variables
      - k_X_Pretrip_Setpoint, k_X_Pretrip_Hys, k_Trip_Delay



'th_X_Pretrip' logic

Case Study (Cont'd)

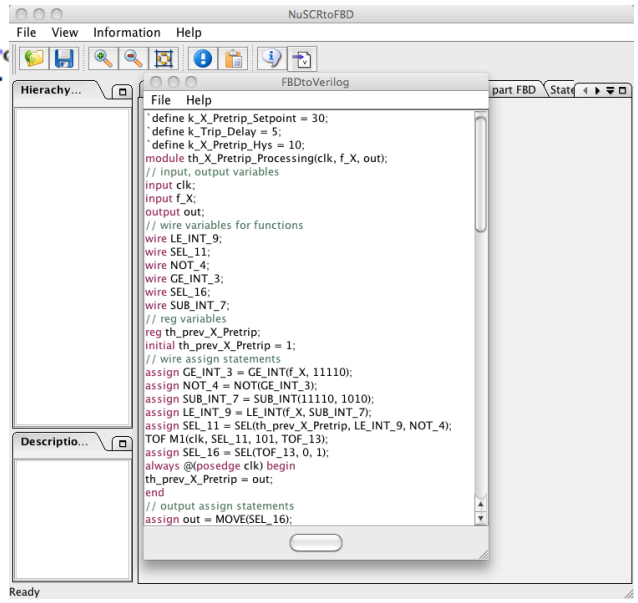
- Translate into Verilog program by *FBDtoVerilog*



Case Study (Cont'd)

- Translated into Verilog program by *FBDtoVerilog*
  - 98 lines Verilog program
  - It covers 12 translation rules of the 13 rules

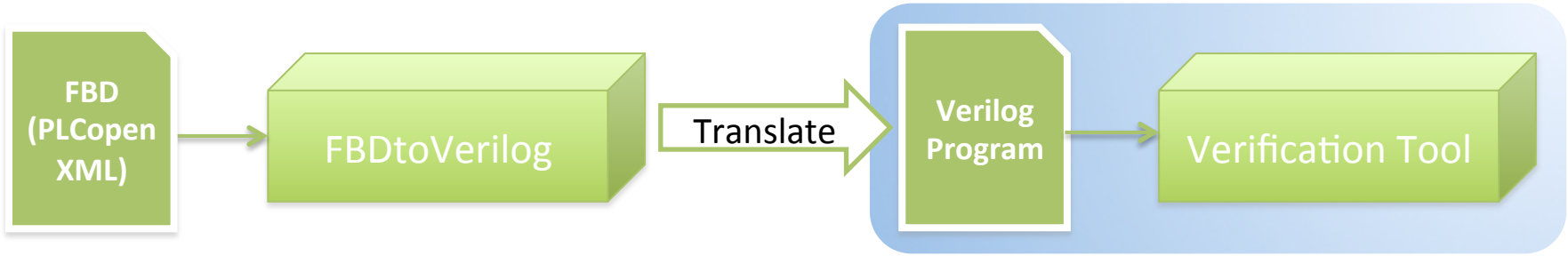
```
- <FBD>
  + <inVariable localId="1" height="16" width="32">
  - <inVariable localId="2" height="16" width="32">
      <position x="0" y="0" />
    - <connectionPointOut>
        <relPosition x="32" y="8" />
      </connectionPointOut>
      <expression>k_X_Pretrip_Setpoint</expression>
    </inVariable>
  - <block instanceName="GE_INT" localId="3" typeName="
      .....
```




```
1  `define k_X_Pretrip_Setpoint = 30;
2  `define k_Trip_Delay = 5;
3  `define k_X_Pretrip_Hys = 10;
4  module th_X_Pretrip_Processing(clk, f_X, out);
5  // input, output variables
6  input clk;
7  input f_X;
8  output out;
9  // wire variables for functions
10 wire LE_INT_9;
11 wire SEL_11;
12 wire NOT_4;
13 wire GE_INT_3;
14 wire SEL_16;
15 wire SUB_INT_7;
16 // reg variables
17 reg th_prev_X_Pretrip;
18 initial th_prev_X_Pretrip = 1;
19 // wire assign statements
20 assign GE_INT_3 = GE_INT(f_X, 11110);
21 assign NOT_4 = NOT(GE_INT_3);
22 assign SUB_INT_7 = SUB_INT(11110, 1010);
23 assign LE_INT_9 = LE_INT(f_X, SUB_INT_7);
24 assign SEL_11 = SEL(th_prev_X_Pretrip, LE_INT_9,
   NOT_4);
25 TOF M1(clk, SEL_11, 101, TOF_13);
26 assign SEL_16 = SEL(TOF_13, 0, 1);
27 always @(posedge clk) begin
28 th_prev_X_Pretrip = out;
29 end
```

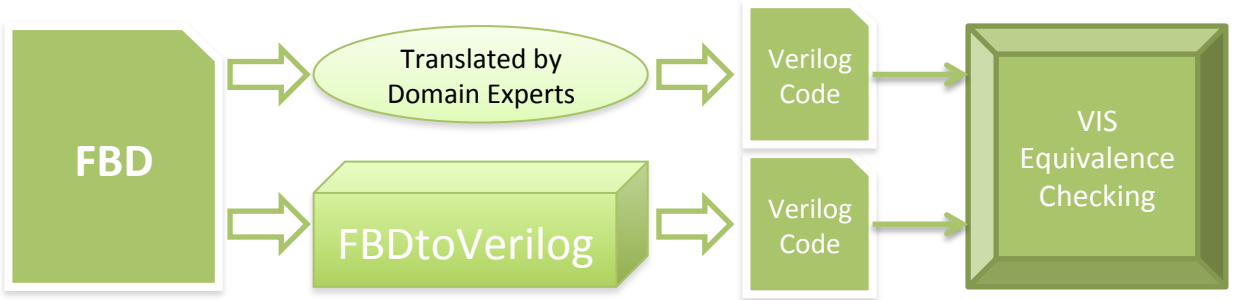
Case Study (Cont'd)

- 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



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

Translated Code	Post-processed Code

input a;
input b;
 	
var = ADD(a, b);
 	
function ADD_INT;
input in1;
input in2;
begin
ADD_INT = (in1 + in2);
end
endfunction
	input [0:6] a;
	input [0:6] b;
	
	var = ADD(a, b)
	
	function [0:6] ADD_INT;
	input [0:6] in1;
	input [0:6] in2;
	begin
	ADD_INT = (in1 + in2);
	end
	endfunction

Optimization

- Code can be optimized
  - Code for Function Blocks

Func	Translated Code	Optimized Code

ADD	input a;
	input b;
	
	var = ADD(a, b);
	
	function ADD_INT;
	input in1;
	input in2;
	begin
	ADD_INT = (in1 + in2);
	end
	endfunction
		input a;
		input b;
		
		var = a + b;

Translation Rules(1)

- Rules for Function block

Function block	*1*	$FB = < Name, IP, OP, BD >$	***function*** [*type_of_OP*] *name_of_function*; … ***endfunction***
	2	$ip_i \in IP$	***input*** [*type_of_OP*] ip_i;
	3	*BD* (logical or arithmetic)	***begin*** *name_of_function* = $ip_0 \oplus ip_1 \oplus \cdots \oplus ip_n$; ***end***
	4	*BD* (selection) - *provided IP* = { $k, ip_1, ip_2, \cdots, ip_n$ }	***begin*** *name_of_function* = ($k == 0$) ? ip_0 : ($k == 1$) ? ip_1 : … ($k ==$ n) ? ip_n : *default*; ***end***

Translation Rules(2)

- Rules for Component FBD

Component_FBD	5	*Component_FBD = < FBs, T, I, O >*	***module*** *name_of_Component_FBD (list_of_I , list_of_O);* … ***endmodule***
	6	$v_i \in I$ *- provided* $v_i \in V_{comp\_FBD\text{-}I}$	***input*** *[type_of_*v_i*]* v_i*;*
	7	$v_o \in O$ *- provided* $v_o \in V_{comp\_FBD\text{-}O}$	***output*** *[type_of_*v_o*]* v_o*;* ***wire*** *[type_of_*$v_{intermediate}$*]* $v_{intermediate}$*;* … ***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	***reg*** *[type_of_*v_o*]* v_o*;* ***initial*** v_o *=initial_value_of_*v_o*;* … ***always @(posedge clk) begin*** *assignment_for_*v_o*;* ***end***
	9	*function block* $\in$ *FBs*	***function*** *...* *definition_of_function_block* ***endfunction***

Translation Rules(3)

- Rules for System FBD

System_FBD	*10*	*System_FBD = < FBDs, T, I, O >*	***module*** *name_of_System_FBD (list_of_I , list_of_O);* … ***endmodule***
	11	$v_i \in I$ *- provided* $v_i \in V_{sys\_FBD\text{-}I}$	***input*** *[type_of_*v_i*]* v_i*;*
	12	$v_o \in O$ *- provided* $v_o \in V_{sys\_FBD\text{-}O}$	***output*** *[type_of_*v_o*]* v_o*;* ***wire*** *[type_of_*$v_{intermediate}$*]* $v_{intermediate}$*;* … ***assign*** v_o *= assignment_for_*v_o*_using_Verilog_moudle_instantiations;*
	13	$v_o \in O$ *- provided* $\exists v_i \in I$ *&* $i == o$ *-* v_i *precedes* v_o	***reg*** *[type_of_*v_o*]* v_o*;* ***initial*** v_o *=initial_value_of_*v_o*;* … ***always @(posedge clk) begin*** *assignment_for_*v_o*;* ***end***