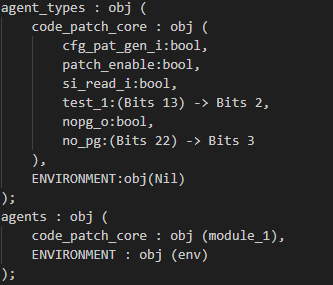
# **Module**



*Figure 1 Module in SV*

**

*Figure 2 Module in Aplan*

Figure 1 shows a module in System verilog. During translation to Aplan, it is transformed into Agent type and Agent, which can be seen in Figure 2, which shows this same module.   
  
The **code\_patch\_core** module is transformed into **Agent Type** **code\_patch\_core**, on the basis of which an **Agent** named module\_1 is created. The number of the module in the name is issued sequentially depending on the number of modules in the processed program

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/assigns/assign_1/assign_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/assigns/assign_1/aplan/project.env_descript)

# **Declarations**

Any of the declarations located in the middle of the module, be it Input, Output port or wire, reg, genvar, int, and others become attributes of the generated Agent type from this module, as can be seen in Figure 2

input si\_read\_1

output si\_data\_o\_1

wire adr\_act

reg no\_pg

genvar j

int i

*Listing 1.1 Input,Output ports, wire, reg, int, genvar declaration without size*

If the declarations were without specifying the size (as shown in listing 1.1), then we transform it into bool type, exceptions are genvar and int , which are represented with type int (see listing 1.2).

si\_read\_1: bool,

si\_data\_o\_1: bool,

adr\_act: bool,

no\_pg: bool,

j:int,

i:int

*Listing 1.2 Presentation of declarations without size in Aplan*

If the size was specified during the announcement, we present it as bits of the specified size (see listing 2.2)

input [31:0] si\_read\_2

*Listing 2.1 Declaration with size*

si\_read\_2:Bits 32

*Listing 2.2 Presentation of declaration with size in Aplan*

If the size and dimension were specified(see listing 3.1), then we will present the size as a function (see listing 3.2)

input [31:0] si\_read\_3[2]

*Listing 3.1 Declaration with dimension size*

si\_read\_3:(Bits 32) -> Bits 2

*Listing 3.2 Presentation of declaration with dimension size*Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/code_patch/code_patch_core.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/code_patch/aplan/project.env_descript)

# **Assigns**

Any assignment is translated into action.

Example :

patch\_enable = si\_read\_i & cfg\_pat\_gen\_i;

*Listing 4.1 Assign example*

assign\_1 = (

(1)->

("code\_patch\_core#module\_1:action 'assign (patch\_enable = si\_read\_i & cfg\_pat\_gen\_i)';")

(module\_1.patch\_enable = module\_1.si\_read\_i & module\_1.cfg\_pat\_gen\_i)),

*Listing 4.2 Assign action example*

As you can see, the assignment shown in Listing 4.1 is translated into the action shown in Listing 4.2. The names of the variables are changed to attribute calls of the agent, which is an interpretation of the module from SV

After casting to the action, you also need to place the action's call in the behavior. The placement of the call to action in the behavior depends on where the training was. Its placement will be considered in the following paragraphs (Assignment during declaration, Assignment inside blocks, Out of block assignments)

## **Assignment during declaration**

If the assignment took place during the declaration (see listing 4.3), then it will be called in the INIT\_MODULE\_N protocol (where N is the number of the module to which the assignment refers) (see listing 4.5)

module code\_patch\_core

(

input si\_read\_i = 1'b0,

);

*Listing 4.3 Assignment during declaration*

assign\_1 = (

(1)->

("code\_patch\_core#module\_1:action 'assign (0)';")

(0)),

*Listing 4.4 Assignment during declaration action*

B\_MODULE\_1 = INIT\_MODULE\_1,

INIT\_MODULE\_1 = assign\_1

*Listing 4.5 Assignment during declaration action call in behavior*

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/assigns/assign_1/assign_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/assigns/assign_1/aplan)

## **Assignment inside blocks**

If the assignment took place inside blocks (see listing 4.6), then it will be called in the blocks protocol(in example it is always) (see listing 4.8)

module code\_patch\_core

(

input si\_read\_i,

);

always\_comb

begin

si\_read\_i = 1'b0

end

*Listing 4.6 Assignment inside blocks declaration*

assign\_1 = (

(1)->

("code\_patch\_core#module\_1:action 'assign (0)';")

(0)),

*Listing 4.7 Assignment inside blocks action*

B\_MODULE\_1 = Sensetive(ALWAYS\_1),

ALWAYS\_1 = B\_1,

B\_1 = assign\_1

*Listing 4.8 Assignment inside blocks action call in behavior*

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/if_statemens/if_statement_1/if_statement_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/if_statemens/if_statement_1/aplan)

## **Out of block assignments**

If the assignment took place out of block (see listing 4.9), then it will be called in the MAIN\_MODULE\_Nprotocol (where N is the number of the module to which the assignment refers) (see listing 5.1)

module code\_patch\_core

(

input si\_read\_i,

);

assign si\_read\_i = 1'b0

*Listing 4.9 Assignment Out of block*

assign\_1 = (

(1)->

("code\_patch\_core#module\_1:action 'assign (0)';")

(0)),

*Listing 5.0 Assignment Out of block action*

B\_MODULE\_1 = MAIN\_MODULE\_1,

MAIN\_MODULE\_1 = assign\_1

*Listing 5.1 Assignment Out of block action call in behavior*

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/assigns/assign_1/assign_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/assigns/assign_1/aplan)

# **Operators**

Binary and arithmetic operators are not further transformed in any way, in some cases they are replaced by equivalent ones so that AVM (Algebraic Virtual Machine) understands them correctly.

## **Parallel assignment**

Parallel assignments (see Listing 5.2) are converted to regular assignments, but their call is wrapped in Sensetive() (see Listing 5.3, 5.4)

Data\_out <= Data\_in & 16'hFFFE;

*Listing 5.2* Parallel assignment

assign\_3 = (

(1)->

("register\_write\_once\_example#module\_1:action 'assign (Data\_out <= Data\_in & 65534)';")

(module\_1.Data\_out = module\_1.Data\_in & 65534)),

*Listing 5.3* Parallel assignment action

Sensetive(assign\_3)

*Listing 5.4* Parallel assignment action call

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/sv_example_1/sv_example_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/sv_example_1/aplan)

# **Values**

Hex and binary values ​​are translated to decimal

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/sv_example_1/sv_example_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/sv_example_1/aplan)

# **If statement**

The condition from the If statement (see listing 5.5) is translated into the action If\_N (where N is the number of the if statement in the processed program) (see listing 5.6), after which the action is called in the behavior (see listing 5.7)

if (~ip\_resetn)

begin

Data\_out[0] <= Write\_once\_status;

end

*Listing 5.5 If statement*

if\_1 = (

(~(module\_1.ip\_resetn))->

("register\_write\_once\_example#module\_1:action 'if (~ip\_resetn)';")

(1)),

*Listing 5.6 If statement action*

B\_1 = if\_1.IF\_BODY\_1 + !if\_1,  
IF\_BODY\_1 = Sensetive(assign\_1),

*Listing 5.7 If statement action call*

# **Loops**

Loops (see listing 6.1) are transformed into protocols consisting of serial and parallel calls of such protocols as:

LOOP\_COND\_N , LOOP\_BODY\_N, LOOP\_INC\_N, LOOP\_MAIN\_N (where N is the number of the loop protocols in processed program)   
  
The LOOP\_N protocol call is executed like other protocol calls that are in the body of any of the structures (see listing 6.2)

The LOOP\_COND N protocol includes a call to the cond\_N action (where N is the loop number in the processed program) (see listing 6.3)

always\_comb begin

for (int h = 0; h < 8; h++) begin

assign patch\_match[h] = (si\_addr\_i == ctl\_pat\_addr\_i[h]) & ctl\_pat\_pen\_i[h];

end

end

*Listing 6.1 Loop in always block*

B\_MODULE\_1 = Sensetive(ALWAYS\_COMB\_1),

ALWAYS\_COMB\_1 = LOOP\_1,

LOOP\_1 = LOOP\_INIT\_1.LOOP\_MAIN\_1,

LOOP\_MAIN\_1 = LOOP\_COND\_1.(LOOP\_BODY\_1;LOOP\_INC\_1;LOOP\_MAIN\_1) + !LOOP\_COND\_1,

LOOP\_INIT\_1 = assign\_1,

LOOP\_INC\_1 = assign\_2,

LOOP\_COND\_1 = cond\_1,

LOOP\_BODY\_1 = assign\_3

*Listing 6.2 Loop call*

cond\_1 = (

(module\_1.h < 8)->

("code\_patch\_core#module\_1:action 'cond (h < 8)';")

(1))

*Listing 6.3 Condition action*

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/loops/loop_1/loop_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/loops/loop_1/aplan)

# **Generate**

The Generate block (see listing 6.1) is transformed into the GENERATE\_N protocol (where N is the number of the generate protocols in processed program) , which is called in the MAIN\_MODULE\_Nprotocol (where N is the number of the module to which the generate refers) (see listing 6.2)

generate

for (j = 0; j < 8; j++) begin

assign patch\_match[j] = (si\_addr\_i == ctl\_pat\_addr\_i[j]) & ctl\_pat\_pen\_i[j];

end

endgenerate

*Listing 7.1 Generate block*

B\_MODULE\_1 = (MAIN\_MODULE\_1),

MAIN\_MODULE\_1 = GENERATE\_1,

GENERATE\_1 = LOOP\_1

*Listing 7.2 Generate protocol call*

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/loops/loop_1/loop_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/loops/loop_1/aplan)

# **Always**

The always structure (see Listing 8.1) is transformed into the ALWAYS\_N protocol (where N is the always number in the processed program). Protocols are ALWAYS called in parallel (see Listing 8.2)

always @(posedge Clk or negedge ip\_resetn)

if (~ip\_resetn)

begin

Data\_out <= 16'h0000;

Write\_once\_status <= 1'b0;

end

*Listing 8.1 Always structure*

B\_MODULE\_1 = Sensetive(ALWAYS\_1, module\_1.Clk || !module\_1.ip\_resetn)

*Listing 8.2 Always call*

Examples:

* [system verilog example](https://github.com/vladyslav-dubina/SV2Aplan/blob/SV2Aplan_stable/examples/sv_example_1/sv_example_1.sv)
* [aplan example](https://github.com/vladyslav-dubina/SV2Aplan/tree/SV2Aplan_stable/examples/sv_example_1/aplan)