

Contents

1	Introduction	2
2	Tool Flow	2
3	Using the tools	2
4	Class Hierarchy	2
5	Program Flow	4
6	Test Example	5
A	Usage	6
B	Files and their relations	8
C	Result Samples	8
D	C++ Source Code for compiling	12

1 Introduction

This application translates C++ source language which has classes from Scenic and ICSP class library. Polymorphism concept in Programming Language is applied to generate RTL VHDL. In this report, application architecture and implementation method is described.

2 Tool Flow

The flow of the entire process of translation is shown in figure 1.

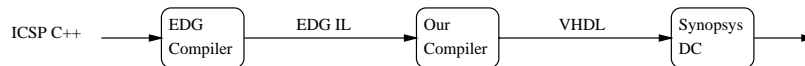


Figure 1: *Tool Flow*

A C++ file is converted into the `il` format using the `edg` tool. The EDG tool performs the lexical analysis and parsing of the program to generate the intermediate format which is termed as the *il* format. The tool provides methods to access this internal format which is read by our program to generate the vhd code. This `il` format is read by our program termed as 'our compiler' in the figure. It converts this `il` format to VHDL. Once we have converted it into VHDL we can use any of the available tools for synthesis purposes.

3 Using the tools

This translation is done using two application. The first one is to generate Intermediate Language of parse tree using Edison Design Group Front End. And our application translate this IL into RTL VHDL. ICSP classes can guide to generate Register Transfer Level description.

- `edgcpfe` (EDG tool)
This is the front end which generate “.il” from C++ sources.
- `pm_pass` (our Compiler)
This is our application to translater VHDL files.

4 Class Hierarchy

We have extended EDG with a new driver by adding the two classes,

- `rtl_translater_c`
- `rtl_body_dump`

These two classes perform the entire translation from the il_format.

These classes read the il-format generated by EDG and process it to translate it to the corresponding VHDL equivalent. The class hierarchy is shown in fig 2. The two classes have a very flat structure with the functions in rtl_translater_c relating to translating the high level structure of a C++ file into the equivalent VHDL. It uses the methods in rtl_body_dump to dump VHDL equivalents for the constructs of statements and expressions viz. if-statements, while-loops, function calls, declaration-statements, conditions etc. The class hierarchy shows the member functions of the classes.



Figure 2: *Class Hierarchy*

5 Program Flow

The `rtl_translater_c` has a member function `transform_class_to_entity` which transforms a class from a scenic C++ description to a VHDL entity. This class is the main class which does the translation work. The `transform_class_to_entity` function is called for every file to translate it into VHDL. First EDG maps the files into an `il_header`. This `il_header` format is read into the memory and checks the `il_header` to identify the various types of classes. The routine checks to see if the header type is either one of the following types :

- `icsp_signal_format_c`
- `icsp_polymorphic_format_c`
- `icsp_aggregate_c`

These are the base classes in the scenic library and the user defined classes are derived from them. The flow is shown in figure 3

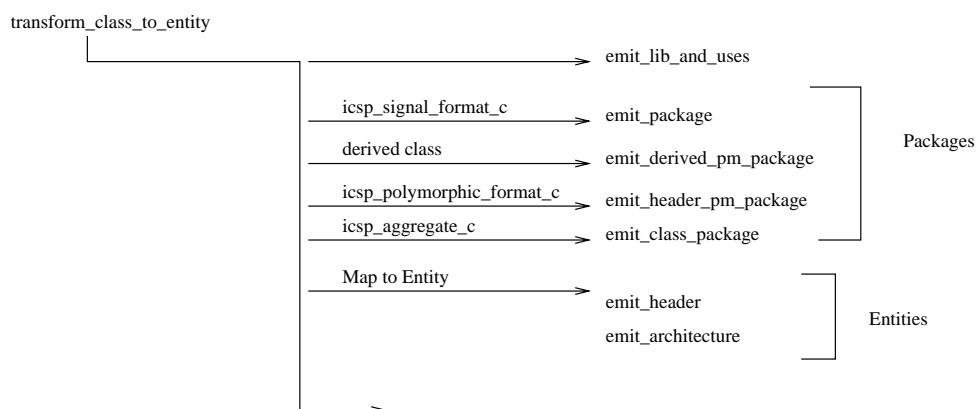


Figure 3: *Program Flow*

The libraries required by VHDL, e.g. `IEEE.std_logic_1164.all`. are added to each VHDL file that is generated. This is done by the `emit_lib_and_uses()` function. We focus on the class hierarchy of the program and for every base class we generate packages with the appropriate functions. The `emit_package()` function dumps the packages in VHDL format. For derived classes the function `emit_derived_pm_package()` is called. When the base class is of type `"icsp_polymorphic_format_c"` the `emit_header_pm_package()` is used for the polymorphic classes. For the type `icsp_aggregate_c` the `emit_class_package()` function is called to write the corresponding package.

In a package function the appropriate records are added and all the functions are translated. The classes for which we need to generate the entity and the corresponding architecture, they are generated by the `emit_header()` and `emit_architecture()` functions respectively.

6 Test Example

We have used a dlx pipeline as a test example for the entire process. We have implemented the entire pipeline in System C, using polymorphism. The design of this pipeline is shown in figure 4.

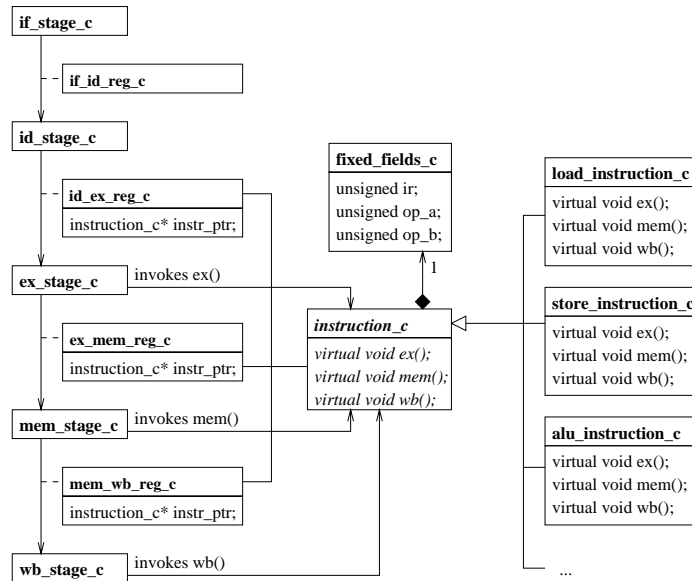


Figure 4: *DLX Pipeline*

We have also implemented the pipeline in VHDL by hand to compare the results. The VHDL design is completely implemented and synthesized.

In the appendix we have described the compilation and translation process for the scenic files along with a result sample file.

A Usage

To generate VHDL, we use DLX pipeline architecture which is written in Scenic and ICSP Class libraies. Here is the example of usage:

```
%
% cat compile_cc
#!/bin/csh

foreach file ( if_stage_c \
               id_stage_c \
               ex_stage_c \
               mem_stage_c \
               wb_stage_c \
               pc_setting_logic_c \
               hazard_detection_unit_c \
               if_id_format_c \
               id_ex_format_c \
               ex_mem_format_c \
               mem_wb_format_c \
               dlx_pipeline_c \
               fixed_fields_c \
               instruction_c \
               alu_instruction_c \
               branch_instruction_c \
               halt_instruction_c \
               load_instruction_c \
               nop_instruction_c \
               store_instruction_c \
               )
    echo Compiling $file...
    ../../rtlgen/FrontEnd/edgcpfe --no_bool -I. -DEDG_SYNTHESIS -o $file.cil $file.c
end

echo done.
%
% compile_cc
Compiling if_stage_c...
Compiling id_stage_c...
Compiling ex_stage_c...
```

```

Compiling mem_stage_c...
Compiling wb_stage_c...
Compiling pc_setting_logic_c...
Compiling hazard_detection_unit_c...
Compiling if_id_format_c...
Compiling id_ex_format_c...
Compiling ex_mem_format_c...
Compiling mem_wb_format_c...
Compiling dlx_pipeline_c...
Compiling fixed_fields_c...
Compiling instruction_c...
Compiling alu_instruction_c...
Compiling branch_instruction_c...
Compiling halt_instruction_c...
Compiling load_instruction_c...
Compiling nop_instruction_c...
Compiling store_instruction_c...
done.
%
% cat to_vhdl
#!/bin/csh

foreach file ( if_stage_c \
               id_stage_c \
               ex_stage_c \
               mem_stage_c \
               wb_stage_c \
               pc_setting_logic_c \
               hazard_detection_unit_c \
               if_id_format_c \
               id_ex_format_c \
               ex_mem_format_c \
               mem_wb_format_c \
               dlx_pipeline_c \
               fixed_fields_c \
             )
    echo Translating $file to vhd...
    ../controlled_src/pm_pass $file.cil > vhd/$file.vhd
end

```

```

echo done.
%
% to_vhdl
Translating if\_stage\_c to vhdl...
Translating id\_stage\_c to vhdl...
Translating ex\_stage\_c to vhdl...
Translating mem\_stage\_c to vhdl...
Translating wb\_stage\_c to vhdl...
Translating pc\_setting\_logic\_c to vhdl...
Translating hazard\_detection\_unit\_c to vhdl...
Translating if\_id\_format\_c to vhdl...
Translating id\_ex\_format\_c to vhdl...
Translating ex\_mem\_format\_c to vhdl...
Translating mem\_wb\_format\_c to vhdl...
Translating dlx\_pipeline\_c to vhdl...
Translating fixed\_fields\_c to vhdl...
done.
%
% exit

```

B Files and their relations

- rtl_tralstrate.c.cc
- rtl_tralstrate.c.hh
- rtl_body_dump.cc
- rtl_body_dump.hh

C Result Samples

This is a Scenic sample if_stage.c.cc

```

////////////////////////////////////

```

```

#pragma implementation "if_stage_c.hh"

#include "if_stage_c.hh"
//#include <iostream.h>

////////////////////////////////////
if_stage_c::if_stage_c(const char* NAME,
    sc_clock_edge& TICK,
    const sc_signal<bool>& RESET,
    const sc_signal<bool>& STALL,
    sc_signal<if_id_format_c>& IF_ID_REG,
    sc_signal<unsigned>& NPC,
    const sc_signal<unsigned>& PC,
    memory_unit_base_c<unsigned>& PROGRAM_MEMORY
    ) : sc_sync(NAME, TICK),
    reset(RESET),
    stall(STALL),
    npc(NPC),
    pc(PC),
    program_memory(PROGRAM_MEMORY),
    if_id_reg(IF_ID_REG){
    watching(reset.delayed()== true);
}

////////////////////////////////////
void if_stage_c::entry(void) {

    unsigned ir, pc_value;
    if_id_format_c if_id_value;

    // The reset should retrigger this method...
    // There should be a watch on the reset signal...
    // (check for that later...)
    npc.write(pc.read());

    while (true) {

        wait();
    }
}

```

```

W_BEGIN; // Local watch for stall
{
    // Note that watched events are sampled only at the active
    // edge of the process, i.e. on the execution of the wait().
    watching(stall.delayed()== true);
}

W_DO;
{
    npc.write(pc.read());
    pc_value= pc.read();
    ir=      program_memory.read(pc_value);

    pc_value++;
    if_id_value.ir=  ir;
    if_id_value.npc= pc_value;

    npc.write(pc_value);
    if_id_reg.write(if_id_value);
}

W_ESCAPE;
{// do nothing. the read on the pc will just be delayed a cycle.
    switch(ir) {
        case 0:
            ir--;
            break;
        case 1:
            ir++;
            break;
        default:;
    }
}

W_END;

//cout << "| (1) IF: IR= " << hex << ir << " [";
//cout << sc_clock::time_stamp() << "]" << endl;
}

```

```
}
```

The translated result:

```
-----  
-- This file was generated automatically by...  
-----
```

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;
```

```
library work;  
use work.all;
```

```
ENTITY if_stage_c is  
port (  
clk: in std_logic;  
reset: in std_logic;  
stall: in std_logic;  
if_id_reg: out if_id_format_c;  
npc: out integer;  
pc: in integer  
);  
end if_stage_c;
```

```
ARCHITECTURE my_compiler of if_stage_c is
```

```
BEGIN
```

```
entry: process  
variable ir: integer;  
variable pc_value: integer;  
variable if_id_value: if_id_format_c;  
begin  
if clk'event & clk = '1' then  
if stall = '1'  
ir
```

```
else

npc <= pc <= ;
pc_value <= pc;
ir := program_memory <= pc_value;
pc_value := pc_value + 1;
if_id_value.ir := ir;
if_id_value.npc := pc_value;
npc <= pc_value;
if_id_reg <= if_id_value;

end if;
end if;
end process entry;

END my_compiler;
```

D C++ Source Code for compiling

- rtl_tralslate_c.cc
- rtl_tralslate_c.hh
- rtl_body_dump.cc
- rtl_body_dump.hh