COLUMBIA UNIVERSITY
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

EHDL

Easy Hardware Description Language

COMS 4115 Programming Languages and Translators

Final Report

Paolo Mantovani (pm2613)
Mashooq Muhaimen (mm3858)
Neil Deshpande (nad2135)
Kaushik Kaul (kk2746)

December, 2011

Contents

Chapter 1 Introduction 4
1.1 Motivation 4
1.2 Overview 4
1.2.1 Easy 4
1.2.2 Imperative 4
1.2.3 Concise 4
1.2.4 Adaptive 5
1.2.5 Powerful 5
Chapter 2 Tutorial 6
2.1 Getting Started 6
2.2 Selection Statements 6
2.3 Function Calls, Global Constants and Locals 7
2.4 Types 7
2.5 POS 8
2.6 Parallelism in EHDL 9
2.7 While Loops 10
2.8 Arrays 10
2.9 State Machines 11
2.10 Compilation 12
Chapter 3Reference Manual 13
3.1 Introduction 13
3.2 Syntax Notation 13
3.3 Program 13
3.4 Lexical Conventions 13
3.4.1 Tokens 13
3.4.2 Whitespace 13
3.4.3 Comments 14
3.4.4 Identifiers 14
3.4.5 Reserved Identifiers 14
3.4.6 Literals 14
3.4.7 Operators 15
3.4.8 Separators 15
, : ; ( ) { } 15
3.5 Meaning of Identifiers 15
3.5.1 Types 15

3.5.2 Functions ... 15
3.6 Expressions ... 16
3.6.1 Numbers ... 16
3.6.2 Variables ... 16
3.6.3 Operator Expressions ... 17
3.7 Declarations ... 17
3.7.1 const Declaration ... 17
3.7.2 ASYNC Keyword ... 18
3.7.3 int Declaration ... 18
3.7.4 Array Declaration ... 18
3.7.5 Function Declaration ... 18
3.8 Statements ... 19
3.8.1 Selection statement ... 19
3.8.2 POS statement ... 19
3.8.3 While Statement ... 20
3.8.4 Call Statement ... 20
3.9 Scope ... 21
3.10 EHDL and Parallelism ... 21
Chapter 4Project Plan ... 22
4.1 Team Responsibilities ... 22
4.2 Project Timeline ... 23
4.3 Software Development Environment ... 23
Chapter 5Architectural Design ... 24
5.1 Overview ... 24
5.1.1 The Scanner ... 24
5.1.2 The Parser ... 24
5.1.3 The Abstract Syntax Tree ... 25
5.1.4 The Semantically Checked Abstract Syntaxt Tree ... 26
5.1.5 The Translator ... 27
Chapter 6 Test Plan ... 30
6.1 Error Cases ... 30
6.2 Working Cases ... 31
6.3 Bisect ... 34
6.4 Regression Tests ... 34
Chapter 7 Lessons Learned ... 36
7.1 Team-oriented Development: ... 36
7.2 Interface-oriented Design: ... 36
7.3 Version Control: ... 36
7.4 Test Suite ... 36
7.5 Bisect : Code Coverage ... 36
References ... 37
Appendix AComplete Listings ... 38

Chapter 1

Introduction

1.1 Motivation

Fueled by advances in integrated circuits technology, digital systems have grown increasingly complex over the past several decades. The rise in complexity makes the hardware designer's job difficult. Our work aims to alleviate this difficulty by introducing a Hardware Description Language(HDL) that is easy to learn and straightforward to code in. We feel that traditional HDLs, such as VHDL, are unnecessarily verbose and more low-level than necessary. By developing a language that is succinct and at the same time defers more implementation details to the compiler, we hope to increase the productivity of hardware designers.

1.2 Overview

EHDL is an easy, imperative, concise, adaptive and powerful language for simulating and synthesizing digital systems.

1.2.1 Easy

Our goal was to design a hardware description language that could be learned quickly and be mastered easily. We partly accomplish this by making our language syntactically similar to well known programming languages like C. Our hope is that programmers with exposure to traditional programming languages will be able to master our language with relative ease.

1.2.2 Imperative

A casual look at world's most popular programming languages will show us the immense popularity enjoyed by languages with imperative features. Even O'Caml, the primarily functional language that the EHDL compiler is written in, has imperative features. We believe this is so because an imperative style makes it easier for the programmer to think across abstraction levels. We kept this observation in mind while designing EHDL.

1.2.3 Concise

A typical VHDL program is verbose. It has lots of boiler-plate code that the programmer should not have to worry about. In contrast, EHDL aims to be concise.

1.2.4 Adaptive

EHDL is adaptive to changing technology. A powerful feature of EHDL is the feasibility it allows in re-timing circuits. All we have to do to adapt our design to a faster clock is by taking the existing code and adding one line POS statements. We will discuss this in more detail shortly.

1.2.5 Powerful

EHDL is powerful because it retains most of the capabilities of VHDL.

Chapter 2

Tutorial

2.1 Getting Started

Our "hello world" program is a 32 bit adder. Here is how we write it:

```
int(32) c main (int(32) a, int(32) b ) {
        c = a + b;
}
```

This program is defining a function "main". All EHDL programs consist of functions and optional global constants. The "main" function is a special one. It is the top level entity that glues together the entire system. Inputs and outputs of main are inputs and outputs of the entire system.

For our simple example, the output of main is c, a 32 bit bus. Similarly, a and b are 32 bit input buses. This program is purely combinational, i.e. the output signal depends purely on the input signals (later, we will talk about sequential logic). The statement "c= a + b;" specifies the output signal to be the sum of the input signals.

Assuming we save our program to adder.ehdl, to compile, we use *ehdl,* the EHDL compiler:

```
./ehdl -o adder.vhd adder.ehdl
```

This outputs VHDL code into adder.vhd. We can subsequently use adder.vhd to synthesize or simulate an adder circuit.

2.2 Selection Statements

If/elseif/else and switch-case statements are used to express decisions. They can be nested. See figure 2 and figure 3 for examples.

```
int(1) b main(int(1) a) {
    if (a == 1) {
        b = 0;
    } else {
        b = 1;
    }
}
```

Figure 2. Example of if/else dumb inverter

```
int(8) z main
(int(8) a, int(8) b, int(8) c, int(8)
d, int(2) sel ) {
       switch ( sel ) {
              case 0: z = a;
              case 1: z = b;
              case 2: z = c;
              default:z = d;
       }
}
```

Figure 3. Example of switch/case : Four to One Multiplexer

2.3 *Function Calls, Global Constants and Locals*

The program in figure 4 takes in two 4 bit numbers a = b3-b2-b1-b0 and b = c3-c2-c1-c0 and sets the output d to b3-b2-c1-c0 + C1 + 1, and output e to c + C2. C1 and C2 are global constants.

Although seemingly meaningless, this program does touch upon several important features of EHDL. We will start at main. Main calls the function func3(), the output of func3 is wired to the output of main and the two inputs of func3 are wired to the main inputs a and b. func3() in turn calls func1() and func2(). Observe that the definition of the callees precede the definition of the callers in the program text. This ordering is not cosmetic, it is a requirement for a compilable program.

Let us now delve into the body of func3(). First note the declarations of local variables m1,m2 and m3. Every EHDL variable must have a reset value. If a variable is not initialized(e.g. m1 and m2), the reset value is implicitly set to 0. For m3, we set it explicitly to 1.It should be obvious from the example that once defined, local variables can be used in other statements in the body of the function. They can be written to and read from. Second, note the declaration of the global constants. Global constants can be defined anywhere outside a function definition. They are visible to all functions that are defined in the file.

```
int(4) b func1 ( int(2) a ) {
         b(3:2) = a;
}

int(4) b func2 ( int(4) a ) {
         b(1:0) = a(1:0);
}

int(4) c func3( int(4) a , int(4) b ) {

        int(4) m1;
        int(4) m2;
        int(4) m3 = 1;
        (m1) = func1(a(3:2));
        (m2) = func2(b);
        m3 = C1;
        c = m1+ m2 + m3;
}

const int(4) C1 = 1;
const int(4) C2 = 2;


(int(4) d, int(4) e) main ( int(4) a,
int(4) b, int(2) c ) {
         (d) = func3(a,b);
         e = c + C2;
}
```

Figure 4. An EHDL program with function calls, locals variables and global constants

2.4 *Types*

EHDL is a strongly typed language. Types of all local variables, global constants and input/output port must be known at compile time. We can only use variables of the same type in an operator expression. The program in figure 1 will not compile if we make C2 in the expression *e = c + C2* to be an int(8) rather than an int(4) constant. For all assignment statements except the ones containing multiplication, the type of the variable on the left hand side must equal the types of all the variables on the right hand side. For multiplication, two 4 bit numbers, when multiplied together, give an 8 bit type. Therefore, if a and b are 4 bit types in the statement *c = a * b;* , c must be an int(8) type.

EHDL does not have any concept of casting, implicit or explicit.

2.5 POS

Consider a 4 bit ripple carry adder. This circuit can be built using combinational logic. To speed up the design however, we would like to introduce registers between the 1 bit adders and then use pipelining. POS is the special keyword that gives the EHDL programmer the ability to place positive edge triggered registers that break the combinational path. Re-timing requires only moving the POS statement within the code, without having to rewrite any of the functional blocks.

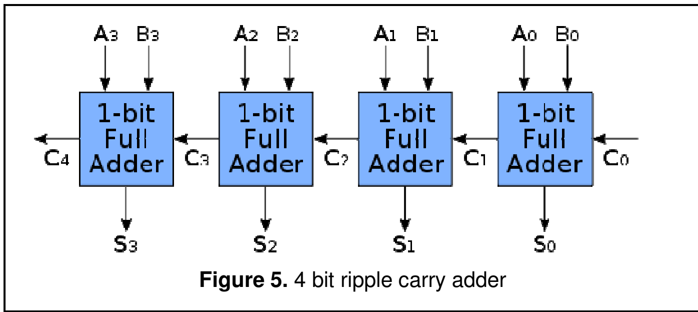


Figure 5. 4 bit ripple carry adder

```
(int(1) sum, int(1) carry)
fulladder(int(1) a, int(1) b, int(1)
carryin){

        sum = a ^ b ^ carryin;
        carry =(a&&b)^(carryin && (a^b));

}


(int(4) s, int(1) overflow) main(int(4)
a, int(4) b, int(1) carryin) {

int(1) sum[4];
int(1) carry[4];

(sum[0], carry[0]) =
fulladder(a(0),b(0),carryin);
POS(1);
(sum[1], carry[1]) =
fulladder(a(1),b(1),carry[0]);
POS(1);
(sum[2], carry[2]) =
fulladder(a(2),b(2),carry[1]);
POS(1);
(sum[3], carry[3]) =
fulladder(a(3),b(3),carry[2]);
POS(1);

s(3) = sum[3];
s(2) = sum[2];
s(1) = sum[1];
s(0) = sum[0];
overflow = carry[3];
}
```

Figure 6. Pipelined 4 bit Ripple Carry Adder

Figure 6 shows a pipelined implementation of the ripple carry adder. After the first POS statement, the identifiers sum and carry actually refer to the output of the register file between the first and second adder, the original local variables sum and carry that are input to the register file are no longer available. The register outputs are only read at the positive edge of a clock.

As is evident in the example, POS gives us the ability to read a variable that was written to in one of the earlier lines. This imperative style is fundamental to many programming languages, but not typically seen in HDLs.

We will mention two important points with regards to POS: first, in the statement POS(*condition*); *condition* can be any Boolean expression. In the ripple carry adder example, the condition was set to always true, so the reads from the register file were happening unconditionally at every positive edge. If instead we wrote POS(a==1), the reading of new values would be stalled until the condition a == 1 was met.

Second, the actual outputs of the function (s and overflow) are only written to at the last pipeline stage. Writes to these variables at earlier pipeline stages are actually writes to intermediate register files. Of course, if we write something to an output port at an earlier pipeline stage, they propagate through the pipeline registers and eventually reach the last pipeline stage, but there is a latency.

2.6 Parallelism in EHDL

Any language that aims to describe digital systems must deal with parallelism. Consider the EHDL program in Figure 4.

```
int(8) a func1 (int(8) b ) {
        int(8) c ;
        /* S4*/ c = b; /*S4*/
        /* S5*/ POS(1) ; /*S5*/
        /*S6*/a = b + c; /*S6*/
}

(int(8) out1, int(8) out2) main ( int(8) in1,
int(8) in2 ) {
        /*S1*/ out1 = in1 + 5; /*S1*/

        int(8) m;
        /*S2*/ (m) = func1(in1); /*S2*/

        /*S3*/  if ( int1 == 1 ) {
                if ( in2 == 2 ) {
                    out2 = in2 - m;
                } else {
                    out2 = in2 + m;
                }
        } /*S3*/
}
```

Figure 4. Statements S1, S2, S3 ,S4, S5 and S6 are all executed in parallel.

In a traditional programming language, the statements S1 to S6 would be executed in sequence. EHDL executes them in parallel.

Formally, let *outermost-statements* be the statements included in a function body that are not nested (even though they may contain nested statements). For instance, assuming the *selection-statement if (expr) stmt* is an *outermost-statement*, its body, *stmt,* represents a *nested-statement*. POS and while statements can only be outer statements.

All EHDL *outermost-statements* except for POS and While are executed in parallel. E*xpressions, selection-statements* and *call-statements* are asynchronous and the assignments they contain take effect as soon as a variable changes. In contrast, POS and while statements are synchronous and the assignments they contain take effect once per clock cycle.

This implicit parallelism forces us to introduce additional constraints on how variables are assigned. Two parallel statements cannot assign the same variable. e.g. the following non-program will not compile:

```
int(8) c main ( int(8) a, int(8) b ) {
     c = a;
     c = b;
 }
```

2.7 While Loops

While loops are another powerful construct in EHDL. They are used in combination with POS statements to implement a fundamental and widely applicable concept in computer science: tail recursion.

```
int(8) c main(int(8) a, int(8) b){

while (a != b) {
        if (a > b) {
                a = a - b;
        }else{
                b = b - a;
        }
       POS(1);
}

POS (a==b);
 c = a;
```

Figure 7. Euclid's gcd algorithm.

```
int(8) c main(int(8) n) {
       int(16) fact = 1;
       while (n > 1) {
           fact = fact(7:0) * n;
           n = n - 1;
       }

       POS(n==1);
       c = fact(7:0);
}
```

Figure 8. Factorial of n. There is an implicit POS(1) at the end of the while loop body.

```
int(8) f main(int(8) n) {

  int(8) a = 0;
  int(8) b = 1;
  int(8) fib;
  int(8) cntr = 1;
  while ( cntr < n ) {
       cntr = cntr + 1;
       b = a + b;
       a = b;
  }
  POS(cntr == n);
  f = a;
}
```

Figure 9: nth Fibonacci number

Figure 7 shows the EHDL implementation of Euclid's algorithm for computing the greatest common divisors between two numbers.

The loop in figure 7 senses the variables a and b when the condition (a!=b) does not hold and loads them onto temporary registers. Variables a and b now refer to the values in the temporary registers rather than the original input wires. The loop then starts to execute: one iteration per clock cycle until the while condition (a != b) evaluates again to false. At this point the while loop re-reads the a and b inputs and repeats. In the meantime, the statement POS(a==b) stalls until the loop condition becomes false. The output is then set to a, the variable that contains the result.

Figure 8 and 9 show two more examples of while loop, the first one computes the factorial of n while the second one outputs the n'th Fibonacci number. Note that a POS is inferred at the end of the while loop body if there isn't one. The program in Figure 10 shows an example where the output variable is changed from within the loop.

2.8 Arrays

We have already seen examples of arrays in the ripple carry adder. Figure 10 shows an implementation of Sieve of Eratostheses that indexes into arrays using changing loop variables.

Arrays in EHDL have certain restrictions. There are no "array objects" that can be passed in as function parameters, if a is an array, `(b) = func(a)` is not permitted. We can however pass in individual array elements in function calls (i.e `(b) = func(a[4])` is allowed). Similarly, output variables of array types are not permitted.

```
/*Emits all prime numbers less than m.
m must be less than 200 as there is
a bounded buffer of size 200 that is
being used */
(int(32) primes=2) main (int(32) m) {

        int(1) a[200];
        int(1) sig;
        int(32) n = 2;
        int(32) k = 2;

        while (n <= m) {
            if ((a[n] == 0) && (k <= m))  {
                 if (k == n) {
                      primes = n;
                 } else {
                // mark as non-prime
                   a[k] = 1;
                 }
                 k = k + n;
            }else {
                 n = n + 1;
                 k = n + 1;
            }
     }
                      }
```

Figure 10. Sieve of Eratostheses: example of array indexing in loops.

2.9 State Machines

State machines are easy to write in EHDL. Throw in a while (1) loop and a few switch-case statements, then grab some popcorn and enjoy the show. Figure 10 shows the implementation of a traffic light.

```
const int(2) HG = 0; // Highway Green
const int(2) HY = 1; // Highway Yellow
const int(2) FG = 2; // Farm Green
const int(2) FY = 3; // Farm Yellow
const int(8) YDuration = 10; // Duration of yellow light
const int(8) FDuration = 100; // timeout for farm green light

(int(1) hwGreen, int(1) hwYellow, int(1) farmGreen, int(1) farmYellow)
main ( int(1) car ) {

     int(2) state;
     int(8) yCntr;
     int(8) fCntr;

     state = HG;
     while (1) {
           switch ( state ) {

                 case HG:
                      hwGreen = 1; hwYellow = 0; farmGreen = 0; farmYellow = 0;
                      if ( car == 1 ) {
                         state = HY;
                         yCntr = 1;

                      }
                 case HY:
                      hwGreen = 0; hwYellow = 1;
                      farmGreen = 0;farmYellow = 0;

                      yCntr = yCntr + 1;
                      if ( yCntr == YDuration ) {
                         state = FG;
                         fCntr = 1;
                      }

                 case FG:
                      hwGreen = 0;hwYellow = 0;
                      farmGreen = 1;farmYellow = 0;

                      fCntr = fCntr + 1;
                      if ((car == 0) || ( fCntr == FDuration )) {
                         state = FY;
                         yCntr = 1;
                      }

                 case FY:
                      hwGreen = 0; hwYellow = 0;
                      farmGreen = 0; farmYellow = 1;

                      yCntr = yCntr + 1;
                      if ( yCntr == YDuration ) {
                         state = HG;
                      }

           }
     }

}
```

Figure 10. State Machine for Traffic Lights

2.10 Compilation

A program can be compiled like so:

```
./ehdl myprogram.ehdl
```

If the program compiles, this will spit out a main.vhd file. If we want, we can specify the name of the output file like so:

```
./ehdl –o myvhd.vhd myprogram.ehdl
```

It is often desirable to define modules once, save the definition and use them in different programs. Say we have an adder module saved in adder.ehdl, and we want to use the module form adder_user.ehdl. We can accomplish this by linking the two files like so:

```
./ehdl –o myvhd.vhd adder.vhd adder_user.vhd
```

The functional units still has to come in order. "`./ehdl –o myvhd.vhd adder_user.ehdl adder.ehdl`" will give a compilation error as the definition of the adder function does not precede the definition of main in adder_user.

Chapter 3

Reference Manual

3.1 Introduction

This manual describes the EHDL language. EHDL is a programming language that allows the programmer to use an imperative style to formally describe and design digital systems. The output of the translator is a fully synthesizable RTL design coded into VHDL.

3.2 Syntax Notation

In the syntax notation used in this manual, we make frequent use of regular expression notation to specify grammar patterns. r* means the pattern r may appear zero or more time, r+ means the r may appear one or more times, r? means r may appear zero or once. r1 | r2 denotes an option between two patterns, r1 r2 denotes r1 followed by r2.

3.3 Program

An EHDL program is a list of global constants and a list of functions. A function is a list of output buses, input buses and a body that describes the functionality of a portion of the hardware design that that function represents.

3.4 Lexical Conventions

3.4.1 Tokens

There are 7 types of tokens: white space, comments, identifiers, keywords, literals, operators, and other separators. If the input string stream has been separated into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

3.4.2 Whitespace

Blanks, tabs, and newlines, collectively referred to as “white space” are ignored except to separate tokens.

3.4.3 Comments

There are two types of comments: single line and multiline. The characters // introduce a single line comment. The characters /* introduce a multiline comment, which terminate with the characters */.

// has no special meaning inside a /* ... */ block, and /* and */ lose their meaning if they come after // in a line.

3.4.4 Identifiers

An identifier consists of a letter followed by other letters and digits. The letters are the ascii characters a-z and A-Z. Digits are ascii characters 0-9. Upper and lower case characters are different (EhDl and ehdl are separate identifiers). There is no limit on the length of an identifier.

letter → ['a'-''z' 'A'-'Z']

digit → ['0'-''9']

identifier → letter(letter | digit)+

3.4.5 Reserved Identifiers

The following identifiers are reserved as EHDL keywords and may not be used otherwise:

If	Switch	Int	POS
Else	Case	While	ASYNC
Default	Const	Uint	

The following identifiers are not EHDL keywords but their usage is forbidden:

```
"abs"|"access"|"after"|"alias"|"all"|"and"|"architecture"|"array"|"assert"|"attribute"|"be
gin"|"block"|"body"|"buffer"|"bus"|"component"|"configuration"|"constant"|"disconnect"|"do
wnto"|"elsif"|"end"|"entity"|"exit"|"file"|"for"|"function"|"generate"|"generic"|"group"|"
guarded"|"impure"|"in"|"inertial"|"inout"|"is"|"label"|"library"|"linkage"|"literal"|"loop
"|"map"|"mod"|"nand"|"new"|"next"|"nor"|"not"|"null"|"of"|"on"|"open"|"or"|"others"|"out"|
"package"|"port"|"postponed"|"procedure"|"process"|"pure"|"range"|"record"|"register"|"rej
ect"|"return"|"rol"|"ror"|"select"|"severity"|"signal"|"shared"|"sla"|"sli"|"sra"|"srl"|"s
ubtype"|"then"|"to"|"transport"|"type"|"unaffected"|"units"|"until"|"use"|"variable"|"wait
"|"when"|"with"|"xnor"|"xor"
```

3.4.6 Literals

A literal is a sequence of digits optionally preceded by the character '-' to indicate negativity. Some examples of literals are: 123, -123 , 0 etc.

literal → -? digit+

3.4.7 Operators

EHDL has the following operators:

+   -   *   !   =

<   >   <=   >=   !=

== || &&   ^   <<

>>   ^   [   ]

The precedence and associativity of the operators are described in section 5.3.

3.4.8 Separators

EHDL has the following separators and delimiters:

,   :   ;   (   )   {   }

3.5 Meaning of Identifiers

Identifiers refer to a variety of things: functions, constants, and variables. A variable is defined solely by its type.

3.5.1 Types

There is one fundamental type: int(k) type. There is also a derived type: the array type.

3.5.1.1 int(k) Type

type_specifier → int(*k*)

int(*k*) is used to indicate a *k* bit input or output bus, where k is a sequence of digits and is greater than 0. The value an int(k) bus takes is interpreted to be a signed integer, with the exception of array references, where the index is always interpreted to be an unsigned integer. Examples of int(*k*) types are: int(5), int(32) etc.

3.5.1.2 Array Type

Arrays are vectors containing a particular type. e.g. int(32) imem[512] is a 512 length vector of int(32) types.

3.5.2 Functions

An EHDL function represents a portion of hardware design that has well defined inputs and outputs and that performs a well-defined function. [3]

3.6 Expressions

Expressions are constants, variables and operator expressions.

expr → numbers

| variables

| ops

3.6.1 Numbers

Numbers are a sequence of digits.

3.6.2 Variables

A variable has the following form:

variable → identifier

| array-reference

| subbus

3.6.2.1 Array References

array-reference → identifier[*expr*]

The first identifier must be an array type. The expr must be a literal or evaluate to an int(k) type, while expressions of constants or literals only are not allowed. If the expression is evaluated to be an out of bounds index at runtime, the behavior of the array reference is defined by the language EHDL is translated into (for VHDL, the array reference will return 0).

3.6.2.2 Subbus

subbus → identifier(number:number)

subbus → identifier(number)

Subbuses can be used to refer to a subset of bits that form a named bus. E.g. if m is an int(32) input bus, m(5:0) denotes the first 6 bits of m. The default ordering for *subbus* arguments is (*high*:*low*), but any order is acceptable. *Subbus* is always translated according to the default ordering; i.e. m(5:0) is equivalent to m(0:5).

When referring to a single bit the shorthand form can be used. E.g. m(5) denotes the 6th bit of m.

3.6.3 Operator Expressions

Table 1 lists the operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence are applied.

Operator	Description	Associativity
[ ]	Array indexing	/
-	unary minus	right to left
!	logical/bitwise negation	right to left
<< >>	bitwise shift left/ bitwise shift right	left to right
&& || ^	logical/bitwise and/or/xor	left to right
*	Mult	left to right
+ -	plus/minus	left to right
< <= > >=	less than/less than equal to/greater than/greater than equal to	left to right
== !=	is equal to/is not equal to	left to right
=	Assignment	right to left

Table 1. Operator precedence and associativity

Multiple assignment expressions cannot be used in the same line. e.g. a = b = c+1 is not permitted.

Expressions containing operators applied to number and/or constants only are not permitted.

An assignment is to be interpreted as the connection of a signal to a driver. Therefore, multiple assignments to the same variable are not permitted, with the exception of assignments inside *while-statements* (see Section 3.8.2). Assignments to the same variable appearing in different branches of *selection-statements* (see Section 3.8.1) are not considered as multiple assignments.

3.7 Declarations

An EHDL declaration is a *const-declaration*, *int-declaration*, *array-declaration* or a *function-declaration*.

declaration → const-declaration

| ASYNC? int-declaration

| array-declaration

| function-declaration

3.7.1 const Declaration

A const declaration has the following form :

const-declaration → const type-specifier identifier = number;

example: const int(6) rtype = 0;

3.7.2 ASYNC Keyword

If a variable must be asynchronously connected to different logic blocks, separated by registers, it must be declared as an asynchronous variable through use of the keyword ASYNC. Asynchronous variables are never assigned by *pos-statements* (see Section 8.4) and they can be written only once (otherwise: conflict because we will end up with multiple drivers for the same signal). For the same reason asynchronous variables cannot be assigned within the body of a *while-statement* (see Section 3.8.4).

3.7.3 int Declaration

An *int-declaration* has the following form:

int-declaration → type-specifier identifier;

| type-specifier identifier = number;

The second option enables the programmer to specify the initial value of the variable just declared. If the value is not initialized, it defaults to 0. For the non-asynchronous variables, the initial value is interpreted also as reset value for *pos-statements* (see Section 3.8.4).

Notice that variable initialization is not supported by logic synthesis tools, which ignore it. However the reset value is taken into account and it must be constant.

3.7.4 Array Declaration

An array declaration has the following form:

array-declaration → type-specifier identifier[digit+];

| type-specifier identifier[digit+] = number;

The second option enables the programmer to specify the initial value of all the elements in the array just declared. If the value is not initialized, it defaults to 0.

3.7.5 Function Declaration

function-declaration → (outputlist) identifier (inputlist) { stmtlist }

Both *inputlist* and *outputlist* are comma separated lists of *int*-declarations or *array-declarations*. *stmtlist* is a list of semicolon separated statements.

3.8 Statements

A statement has the following form:

stmt → { stmtlist }

| expr;

| selection statement

| while statement

| call statement

| POS (expr);

3.8.1 Selection statement

selection-statement → if (expr) stmt;

| if (expr) stmt else stmt;

| switch (expr) case-statement;

case-statement → case-statement-list

| case (const-expr) : stmtlist;

| default : stmtlist

case-statement-list is a semicolon separated list of case-statements. C*onst-expr* is either a *number* or an identifier referring to a global constant declaration.

3.8.2 POS statement

POS is the keyword that allows EHDL to instantiate registers. The statement has the following syntax:

pos-statement → POS(*expr*) ;

The expression in parenthesis is the enable of the register. EHDL will generate a register for each variable which has been assigned in the scope of the *pos-statement*, including the variables involved in a previous *pos-statement* or, if it is the first POS of a function, the function arguments. When reset signal is active (low), all synchronous variables are assigned to their reset value. When reset is not active, a clock signal, shared among all the components of the system, triggers the assignment of all variables assigned in the scope of the *pos-statement*. This introduces a latency of one clock cycle between statements placed before POS and statements placed after POS.

After a *pos-statement* the old identifiers of the variables refer to the output of the register, and the reference to the input is no longer available.

3.8.2.1 POS and Selection Statements

A POS statement is not allowed to exist in the body of a selection statement. POS has the effect of synthesizing registers and if it was allowed to exist for example in the *if block*, but not in the *else block*, this would have no physical meaning. We can not dynamically create a register based on a value that we figure out at “runtime”.

3.8.2.2 POS and While Statements

The body of a while loop must contain at least one *pos-statement* at the end of the body. If it does not, a *pos-statement* is inferred as the last statement of the while body.

3.8.3 While Statement

while-statement → while (expr) stmt

While statements are used to describe logic blocks that implement iterative algorithms (e.g. multiply and accumulate unit).

While statements are translated into VHDL sequential processes where a clock signal, shared among all the components of the system, triggers the next step of the loop. The same variable is updated at every cycle, because a register, represented by an implicit or explicit *pos-statement* (see section 3.8.2.2), breaks the combinational loop. When the expression within the parenthesis evaluates to zero, the VHDL process senses the inputs (i.e. the variables which are read inside the loop) and the loop starts over at the next rising edge of the clock. A meaningful EHDL program therefore requires that all variables read inside a *while-statement* are either function inputs or internal signals correctly assigned to an expression.

A while statement may not contain another while statement in its body. This is a limitation of the current translator.

3.8.4 Call Statement

call statement → (output-list) = identifier(arglist?)

output-list and *arglist* are comma separated lists of variables or constants. The variables must only be of type int(*k*) or a subbus. Array references are also permitted, because their type is int(*k*), while array types are forbidden. If b is an array type, gcd(b[2]) is valid, gcd(b) is not.

3.9 Scope

The lexical scope of a constant (which can only be defined outside functions) is the bodies of all the functions defined in the same file. A function can be defined before the constant is defined, as long as the constant is defined somewhere in the file, it will be visible to the function.

The lexical scope of a parameter of a function begins at the beginning of the block defining the function, and persists through the end of the function body. The lexical scope of a local variable identifier begins at the end of its declaration and persists through the end of the function body.

The lexical scope of a function begins at the end of its definition and ends either at the end of the file (i.e. it is visible to all functions in the file that follow its definition) or until the beginning of a function definition that has the same signature.

3.10 EHDL and Parallelism

EHDL aims to describe the behavior of a digital system; hence it must deal with parallelism.

Let *outermost-statements* be the list of statements included in a function body (i.e. they are not nested, even though they can contain *nested-statements*.). For instance, assuming the *selection-statement if (expr) stmt* is an *outermost-statement*, its body, *stmt,* represents a *nested-statemet*.

Outermost-statements of type *expression* and *selection-statement are* translated into VHDL combinational processes; *call-statements* are translated into component instantiations; *pos-statements* and *while-statements* are translated into VHDL sequential processes. *While-statements* and *pos-statements* must always be *outermost-statements*, except for *pos-statements* appearing in the body of *while-statements* (see Section 3.8.2).

All EHDL *outermost-statements* “execute” in parallel; *expressions, selection-statements* and *call-statements* are asynchronous and the assignments (see Section 6.3 for details on assignments) they contain take effect as soon as a variable changes. *Pos-statements* and *while-statements* are synchronous and the assignments they contain take effect only once per clock cycle (see Sections 3.8.2 and 3.8.4).).

Pos-statements enable the EHDL programmer to delay the effects of other statements, a common technique in hardware design for reducing clock period.

While-statements combined with at least one *pos-statement* break combinational loops, introducing a memory element which stores the updated value of each variable once per clock cycle. The latency introduced by a *while-statement* depends on the number of *pos-statements* included in the body. However the programmer must distinguish between the latency added to assignments, which is determined by POS, and the actual number of clock cycles required to have the correct result of the loop. This number depends on the body of the loop and in general on the previously assigned variables.

Chapter 4

Project Plan

4.1 Team Responsibilities

As designing this language required knowledge of VHDL, the project roles were divided on the basis of the VHDL knowledge of the team members. All team members were responsible for designing, coding, testing and integrating their respective components and everyone held their end of the bargain.

Team Members	Project area handled
Paolo, Mashooq, Neil and Kaushik	Deciding the features to be included Overall Architectural Design Code repository initialization Grammar and Scanner Final Report
Paolo and Mashooq	LRM and Parser
Neil and Kaushik	Abstract Syntax Tree (AST)
Neil and Kaushik	Semantic Type Check (SAST)
Paolo and Mashooq	Translator and Test suite

4.2 *Project Timeline*

The following deadlines were set for key project development goals:

September 28, 2011	Project Proposal , Core language features defined
October 15, 2011	Development repository initialization, code convention defined, project roles and work areas defined
October 25, 2011	Language reference manual, grammar, scanner complete
November 15, 2011	Parser complete
December 10, 2011	Semantic Check complete, Translator complete
December 18, 2011	Test Suite complete, Dev Freeze

4.3 *Software Development Environment*

The project was developed primarily on Linux using Ocaml, using the following major components:

OCaml 3.12.0	:	Primary Development Language
OCamllex	:	OCaml variant of lex utility for lexical analysis
OCamlyacc	:	OCaml variant of yacc utility for parsing
SVN	:	Version control for source code
Mentor Graphics Modelsim SE 6.5b	:	VHDL simulation (golden output)
Synopsys DC 2008/B09	:	VHDL pre-synthesis elaboration (golden output)
Makefile and Regression test scripts	:	Make and bash scripts

Chapter 5

Architectural Design

5.1 Overview

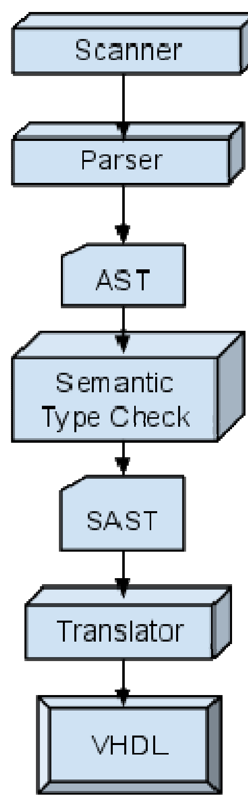


figure 11: High Level Architecture Diagram of the EHDL compiler

The basic components of our translator were : scanner, parser, semantic type checking and translation in that order.

5.1.1 The Scanner

The scanner is used to tokenize the input file into lexemes. At this stage, we strip the comments and the whitespace off the code. Furthermore, any illegal characters or VHDL reserved keywords are caught and failure is reported. The scanner is built using the ocamllex utility.

5.1.2 The Parser

The parser generates the abstract syntax tree (AST) from the stream of tokens that it receives from the scanner. See Appendix A for a listing of the grammar that the parser accepts. The parser is built using the ocamlyacc utility.

5.1.3 The Abstract Syntax Tree

The AST is a recursive structure whose building blocks are shown in the figure below. The AST defines the interface between the scanner / parser and the semantic type checking module.

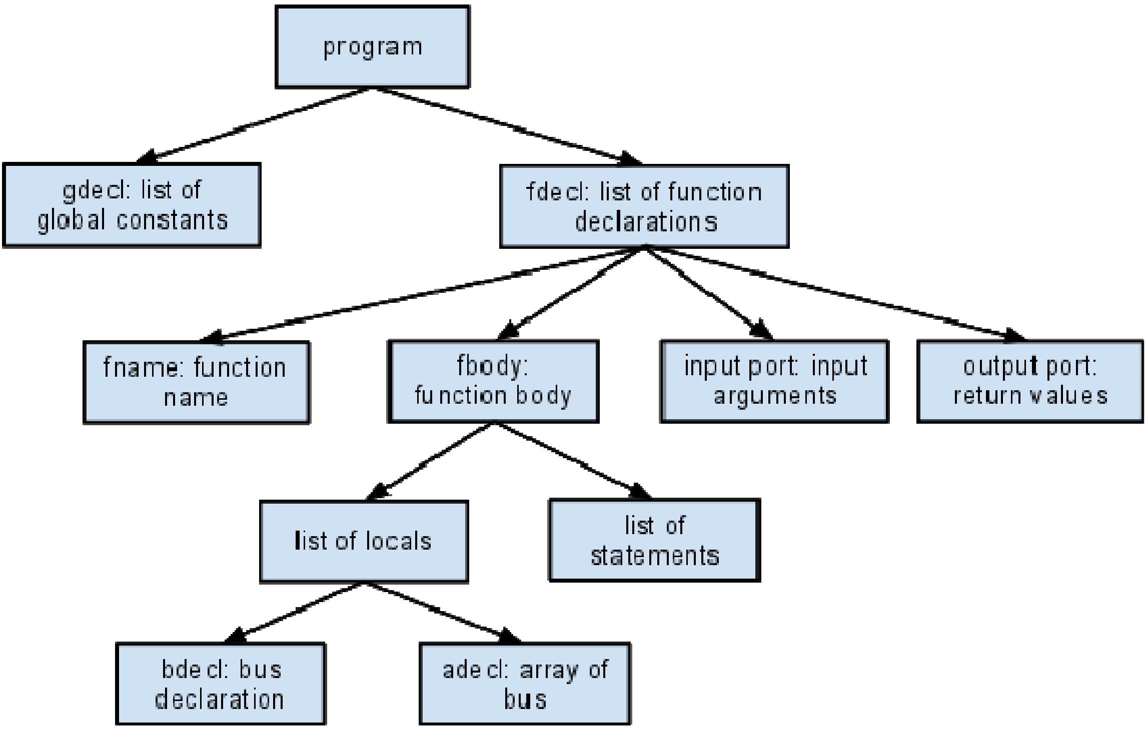


figure 12. Structure of the EHDL AST

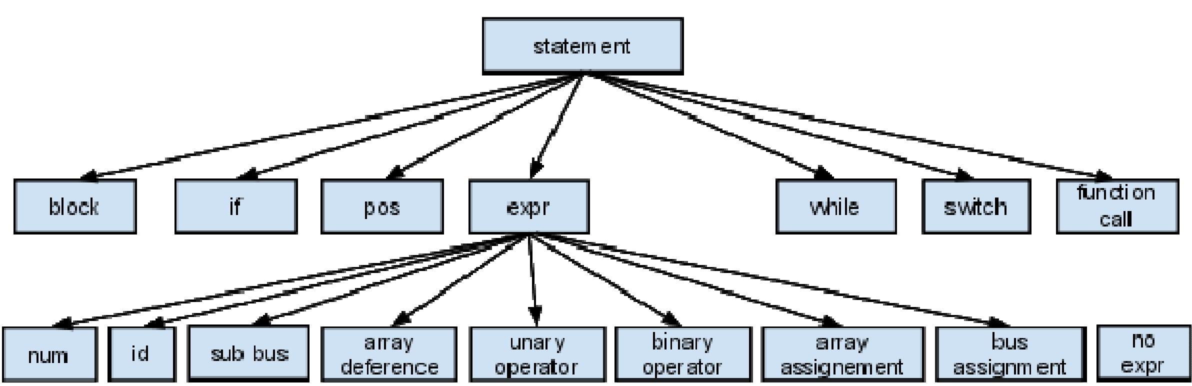


figure 13. Structure of an EHDL statement

5.1.4 The Semantically Checked Abstract Syntaxt Tree

The Semantically checked Abstract Syntax Tree or SAST is built during the semantic type checking phase. It defines the interface between the type check module and the translator.

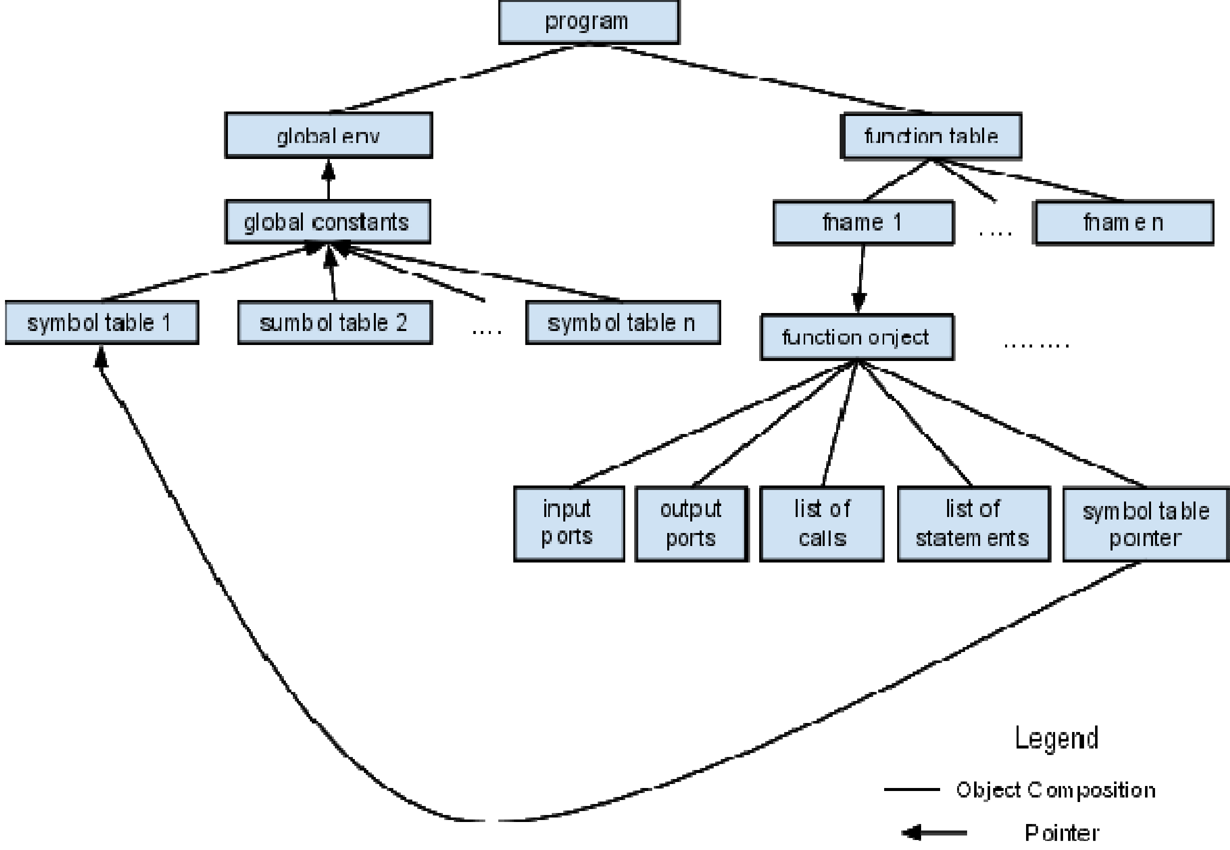


figure 14. Structure of the EHDL SAST

The basic paradigm used during SAST construction is as follows:
- Recursive walk over the AST.
- Analysis of a node returns its type or signals an error.
- The environment maintains information about what symbols are currently in scope

The kinds of checks performed during semantic type checking are:
- Used identifiers must be defined
- Function calls must refer to functions which have been defined earlier.
- Identifier references must be to variables which have been previously declared
- The types of operands for unary and binary operators must be consistent in terms of bus width
- The predicate of an if and while must be a Boolean.
- It must be possible to assign the type on the right side of an assignment to the lvalue on the left in terms of bus width. Moreover, the expression on the left must be a valid lvalue which in the case of ehdl is only an identifier, since the language lacks references.
- Multiple assignments to variables are prohibited except inside conditionals (if / else, switch / case)

and the while loop. This is because multiple assignment in a sequential flow corresponds to shorting different components.
- The types and number of arguments passed to a function call must match the function definition

5.1.5 The Translator

Figure 15 shows the structure of the translator. The translator builds maps to keep track of the structure of the program.

The first task performed by the translator is linking together the files passed as arguments to the command line. This is required to check whether a function call is bound to a function declaration. Then it calls the SAST function that returns a semantically checked Abstract Syntax Tree, from which the translator derives the VHDL code.

The translation of components is divided into three main blocks: printing standard VHDL libraries, printing the so-called VHDL entity that represents the interface of a component and printing the body or architecture of the component.

EHDL function body consists of statements and local variable declarations. Local variables are translated into VHDL signals, while statements are translated into processes. However, to allow the EHDL user to refer to the same variable across registers (see POS statements), the translator must keep track of all assignments through a map and it must distinguish between synchronous (default) variables and asynchronous ones (explicitly declared through ASYNC; see ASYNC). Furthermore the translator keeps track of POS statements and while loops and it combines the information from the assignment map in order to connect properly the VHDL signals.

The strategy is to create a copy of all variables (locals, inputs and outputs) per POS statement in the body of an EHDL function. Every time a POS is detected, a sequential process is printed into the output file. The sensitivity list contains only clock and reset, which are automatically inferred as common signals among all the functions. The process itself constitutes the register that samples all non-asynchronous assigned variables (inputs included). Also a variable keeping track of the current clock cycle is incremented, thus the new references to an assigned variable will be printed using the copy related to that cycle. This is why the input of the register is no longer available after a POS statement. If it was necessary to both sample a signal and read it after the POS, it must be assigned to an asynchronous variable before POS.

The while loop is a more complex structure which requires knowledge of which variables are assigned only before the while loop (list1), which variables are assigned before and inside the loop (list2) and which variables are assigned inside the loop only (list3). Variables in list1 need a simple VHDL assignment of the old variables to their copies referring to the current clock cycle after the while loop. List2 represents a list of potential inputs and outputs of the while loop. When the condition of the loop doesn't hold, the value assigned before the loop is evaluated, whereas when the loop is running, the value refers to the copy for the current clock cycle. This translates into an if statement nested inside the sequential process. Finally, variables in list3 are added to the assignment map, because they're set inside the while statement.

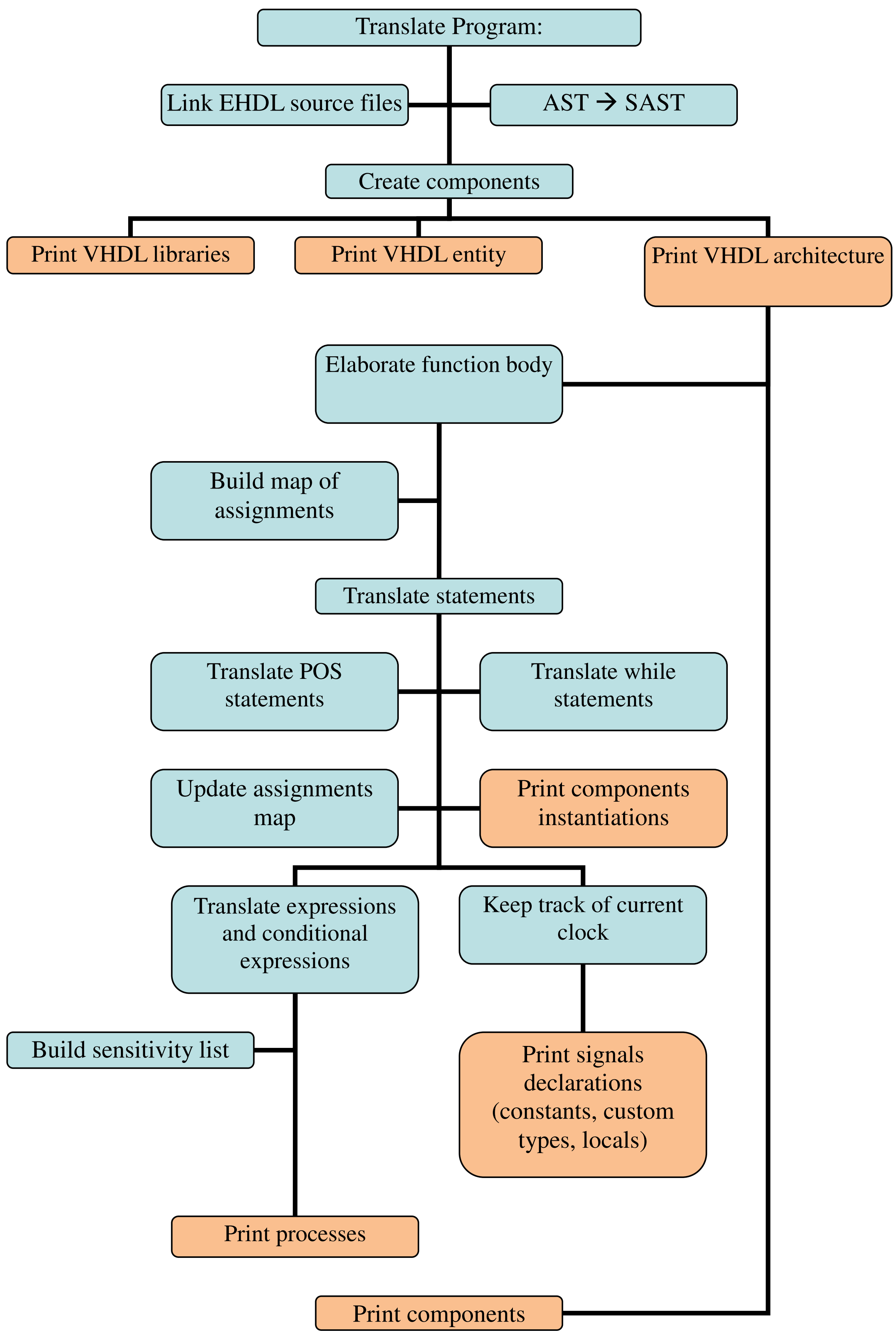


Figure 15. EHDL Translator

Among the other statements, function calls require a special treatment: these are not translated into processes. Each call, instead, makes the translator print a component interface within the preamble of the VHDL architecture and a component instantiation in its body. Multiple function calls to the same object produces a single component interface, but multiple instantiations. The translator guarantees the uniqueness of instance names by generating instance labels.

All other outermost-statements (they can be nested; see the LRM) are printed as combinational processes. The translator analyzes all the expressions and all nested statements to generate the sensitivity list for the process.

The last operation is to concatenate all the portions of the VHDL program together with a few common key words.

Chapter 6

Test Plan

6.1 Error Cases

Table 1 gives the test cases that were used to verify that our compiler was rejecting semantically incorrect programs. To come up with these test cases, we systematically went over the parser, picked statements one by one and tried to come up with cases that the SAST/translator should reject.

func_test2.ehdl	Initial value is bigger than what bus size can hold.
func_test4.ehdl	Calling undefined function.
func_test5.ehdl	Argument mismatch in function call.
func_test6.ehdl	Expressions are not permitted to be used as function arguments.
func_test7.ehdl	Can't use arrays as function arguments
func_test8.ehdl	Passing in a constant whose bus size is diff. from function sig.
if_Test2.ehdl	POS inside if
switchcase_test2.ehdl	case expression not constant
switchcase_test3.ehdl	POS inside switch
typecheck_1.ehdl	assigning input int(2) to int(3)
typecheck_2.ehdl	assigning local variable int(2) to int(3)
typecheck_4.ehdl	type mismatch with mults
typecheck_6.ehdl	another mult typecheck fail
typecheck_7.ehdl	Array can't just be assigned to an int(k) type
typecheck_9.ehdl	Array index out of bounds
typecheck_10.ehdl	Wrong array assignment
typecheck_13.ehdl	assigning larger bus to smaller array element
typecheck_15.ehdl	Wrong bus reference
typecheck_16.ehdl	Const expression not supported
typecheck_19.ehdl	Assigning narrower subbus to wider bus
typecheck_20.ehdl	Const expr in array reference
typecheck_21.ehdl	another array index out of bounds
typecheck_22.ehdl	Multiple drivers for the same variable
typecheck_23.ehdl	assigning another input to input port
typecheck_24.ehdl	assigning local var to input port

Table 1. Error Cases

That Sast rejects these cases was verified by observing the translator error messages.

For example, here's a rejection of a program that calls a non-existent function:

```
@ubuntu:~/PLT/project/ehdl/src$ more ../regsuite/func_test4.ehdl
//Function sub undefined
int(2) c adder ( int(2) a , int(2) b ) {
       c = a + b;
}

int(2) d main ( int(2) a, int(2) b ) {
       (d) = sub(a,d);
}

@ubuntu:~/PLT/project/ehdl/src$ ./ehdl ../regsuite/func_test4.ehdl
…..
Fatal error: exception Failure("undefined function sub")
```

And here's an example of a program that gets rejected because POS is inside an if:

```
@ubuntu:~/PLT/project/ehdl/src$ more ../regsuite/if_test2.ehdl
// Should get Rejected: POS is inside if statement
int(4) res main(int(4) a, int(4) b ) {

      if ( a == 2 ) {
             res = 3;
       POS(1);
       }else {
             res = 4;
       }

}

@ubuntu:~/PLT/project/ehdl/src$ ./ehdl ../regsuite/if_test2.ehdl
….
Fatal error: exception Sast.Error("Pos inside if statement")
```

6.2 Working Cases

Table 2 gives the test cases that were expected to compile fine. Some of these cases were just taking the error cases and making sure the anti-error cases work fine. We tried to knock off several good cases in one program, something we could not do for the error cases, hence the number of good cases is smaller than the number of error cases. There is also some redundancy in these cases as far as code coverage goes. Several programs use the same set of language constructs (e.g. gcd and factorial) but we included them nevertheless because they have already been tested and in general, redundancy in testing does not hurt.

The VHDL code that these programs got translated to were verified to be correct by compiling and simulating the code on ModelSim. What follows is the VHDL code generated by the gcd program in figure 7.

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

```
entity main  is

port (
      clk : in std_logic;
      rst : in std_logic;
      a : in  std_logic_vector(7 downto 0);
      b : in  std_logic_vector(7 downto 0);
      c : out std_logic_vector(7 downto 0));

end main;

architecture e_main of  main is



signal     c_r0,     c_r2,     c_r1     :     std_logic_vector(7     downto     0)     :=
ieee.std_logic_arith.conv_std_logic_vector(0,8);
signal     b_r0,     b_r2,     b_r1     :     std_logic_vector(7     downto     0)     :=
ieee.std_logic_arith.conv_std_logic_vector(0,8);
signal     a_r0,     a_r2,     a_r1     :     std_logic_vector(7     downto     0)     :=
ieee.std_logic_arith.conv_std_logic_vector(0,8);


begin
a_r0 <= a;
b_r0 <= b;
c <= c_r2;

--Pos--
process(clk,rst)
begin
if rst = '0' then
b_r1 <= ieee.std_logic_arith.conv_std_logic_vector(0,8);
a_r1 <= ieee.std_logic_arith.conv_std_logic_vector(0,8);
elsif clk'event and clk = '1' then
if ((a_r1) /= (b_r1))then
if 1 /= 0  then
             if (((a_r1) > (b_r1))) then
             a_r1 <= ((a_r1) - (b_r1));

             else
             b_r1 <= ((b_r1) - (a_r1));
             end if;
end if;
else
if 1 /= 0  then
b_r1 <= b_r0;
a_r1 <= a_r0;
end if;
end if;
end if;
end process;

--Pos--
process(clk,rst)
begin
```

```
if rst = '0' then
b_r2 <= ieee.std_logic_arith.conv_std_logic_vector(0,8);
a_r2 <= ieee.std_logic_arith.conv_std_logic_vector(0,8);
elsif clk'event and clk = '1' then
if ((a_r1) = (b_r1)) then
b_r2 <= b_r1;
a_r2 <= a_r1;
end if;
end if;
end process;
      process (a_r2)
      begin
            c_r2 <= a_r2;
      end process;
end e_main;
```

Figure 11 shows the simulation run on ModelSim using the VHDL code for gcd.

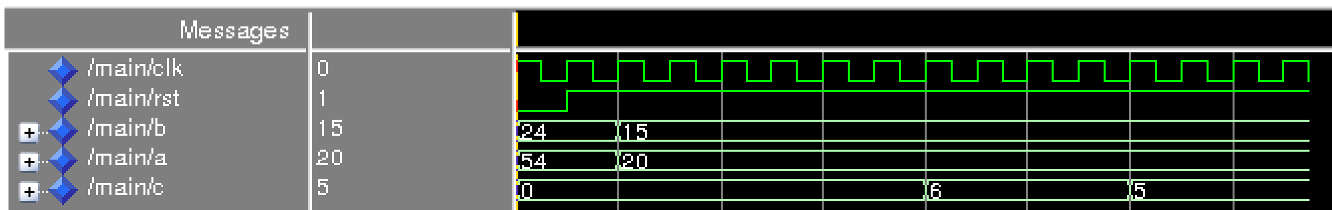


Figure 11. GCD Simulation

adder.vhd	basic combinational logic unit. Just adds two numbers.
choose.vhd	basic if/else
dumb_inv.vhd	basic if/else
factorial.vhd	while loop with POS
fibonacci.vhd	while loop with POS
four_one_mux.vhd	switch case
func_test9.vhd	main calls a function that calls another function
gcd.vhd	canonical example of while loop
if_test1.vhd	nested if/else, >=, <= , > , <
if_test3.vhd	if (a), just using an input variable as an if condition
invoke_function.vhd	Function calling, POS
ops_test1.vhd	An ALU that tests almost all the binary operators
priority_encoder.vhd	switch case
ripple_carry.vhd	POS pipelining
sieve.vhd	array indexing in loop, setting output inside the while loop
subbus.vhd	subbus assignments
switchcase_test1.vhd	switch expr has multiple variables in it
trafficlight.vhd	State machine
two_bit_alu.vhd	basic combinational logic
uminus.vhd	Unary Minus
adder_std.ehdl	Linking multiple files
adder_std_tester.ehdl	Linking multiple files

8-9 additional typecheck cases	These mainly check that the SAST is not rejecting correct programs. One example of such a test case is making sure c = a * b, where c is 16 bits and a,b are 8 bits does not get rejected.

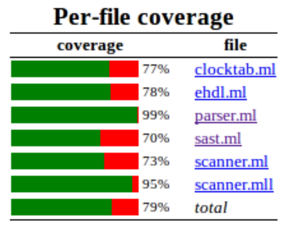
Table 2. Working Cases

6.3 Bisect

Bisect is a code coverage tool for O'Caml. Bisect helped us to expand our test cases to get more code coverage. Below is the report generated by Bisect.

Overall statistics

kind	coverage	kind	coverage
binding	535 / 576 (92%)	class expression	0 / 0 (-%)
sequence	22 / 24 (91%)	class initializer	0 / 0 (-%)
for	13 / 15 (86%)	class method	0 / 0 (-%)
if/then	101 / 185 (54%)	class value	0 / 0 (-%)
try	22 / 24 (91%)	toplevel expression	1 / 1 (100%)
while	0 / 0 (-%)	lazy operator	34 / 47 (72%)
match/function	451 / 615 (73%)		



Per-file coverage

coverage	file
77%	clocktab.ml
78%	ehdl.ml
99%	parser.ml
70%	sast.ml
73%	scanner.ml
95%	scanner.mll
79%	*total*

Generated by Bisect 1.1 on 2011-12-22 18:12:39

6.4 Regression Tests

The VHDL programs that were compiled, simulated and verified in ModelSim were put into a golden folder and the corresponding EHDL files were put into the regression test suit. A script *regression.sh* was used to automatically compile and diff the VHDL output of each test program with the golden version.

Here's an output of the test script:

```
@:~/Dropbox/coms4115/ehdl/regsuite$ ./regression.sh all
PASSED regression test for ./adder.ehdl ...
PASSED regression test for ./choose.ehdl ...
PASSED regression test for ./dumb_inv.ehdl ...
PASSED regression test for ./func_test3.ehdl ...
PASSED regression test for ./func_test9.ehdl ...
PASSED regression test for ./gcd.ehdl ...
PASSED regression test for ./if_test1.ehdl ...
PASSED regression test for ./if_test3.ehdl ...
PASSED regression test for ./invoke_function.ehdl ...
PASSED regression test for ./subbus.ehdl ...
PASSED regression test for ./switchcase_test1.ehdl ...
```

PASSED regression test for ./two_bit_alu.ehdl ...
PASSED regression test for ./switchcase_test1.ehdl ...
PASSED regression test for ./ops_test1.ehdl ...
PASSED regression test for ./factorial.ehdl ...
PASSED regression test for ./four_one_mux.ehdl ...
PASSED regression test for ./priority_encoder.ehdl ...
PASSED regression test for ./trafficlight.ehdl ...
PASSED regression test for ./fibonacci.ehdl ...
PASSED regression test for ./uminus.ehdl ...
PASSED regression test for ./ripple_carry.ehdl ...
PASSED regression test for ./sieve.ehdl ...
PASSED regression test for ./while1.ehdl ...
PASSED regression test for ./while2.ehdl ...
PASSED regression test for ./while3.ehdl ...
PASSED regression test for ./while4.ehdl ...

```
#!/bin/bash
alltests="adder choose dumb_inv func_test3 func_test9 gcd \
if_test1 if_test3 invoke_function subbus switchcase_test1 two_bit_alu \
switchcase_test1 ops_test1 factorial four_one_mux priority_encoder \
trafficlight fibonacci uminus ripple_carry sieve while1 while2 while3 \
while4"
usage="\nregression.sh all \nOR \nregression.sh if_test1 if_test2 ...\n"
if [ $# -eq 0 ]
then
    echo -e $usage
    Exit
fi
comppath="../src/ehdl "
testpath="./"
mainpath="../src"
if [ "$1" = "all" ]
then
    tests=$alltests
Else
    tests=$@
fi
for var in $tests
do

    $comppath $testpath$var.ehdl > temp.txt
    diff main.vhd ../golden/$var.vhd > temp.txt
    difflines=`wc -l temp.txt | cut -f1 -d" "`
    if [ $difflines -eq 0 ]
    Then
        echo "PASSED regression test for $testpath$var.ehdl ..."
    else
        echo "FAILED regression test for $testpath$var.ehdl ..."
        cat temp.txt
    fi

done
```

Chapter 7

Lessons Learned

7.1 Team-oriented Development:

Complementary strengths made this project possible. Paolo and Mashooq had a good background in hardware which was essential for the back end. Neil and Kaushik had a software background and could program the front end.

7.2 Interface-oriented Design:

The AST was frozen quite early in the game, and helped us get up to speed really quickly. We did stub out parts of the SAST, which based off the AST. However, there were some instances where other team members had to wait on the SAST which could have been avoided with stubs.

7.3 Version Control:

SVN was a good productivity tool but we could have used more branches to cut the wait times. There were instances when bugs due to ongoing development on one module stalled or adversely affected development in an unrelated module. These could have been avoided through generous use of stubs and branches.

7.4 Test Suite

Helped uncover a ton of bugs. Moreover, for Neil and Kaushik who had no prior experience with hardware development, this helped to improve understanding of the semantics.

7.5 Bisect : Code Coverage

Again, helped catch bugs by forcing us to devise new test cases.

References

[1] Ritchie, Dennis. C Reference Manual. Retrieved on October 28, 2011 from http://cm.bell-labs.com/cm/cs/who/dmr/cman.pdf

[2] Edwards, Stephen. Micro C Compiler

http://www.cs.columbia.edu/~sedwards/classes/2011/w4115-fall/microc.pdf

[3]VHDL reference manual. Retrieved on October 28, 2011 from http://www.usna.edu/EE/ee462/manuals/vhdl_ref.pdf

[4] Ripple Carry Adder Figure: http://en.wikipedia.org/wiki/Adder_(electronics)

Appendix A

Complete Listings

Start at next page.

Ast.mli

```
1   type operator = Add | Sub | Mul | Lt | Gt | Lte | Gte | Eq | Neq | Or | And | Xor | Shl | Shr | Not
    | Umin
2
3   type bus = {name : string; size : int; init : int; async : bool; isAssigned : bool array}
4
5   type gdecl =
6     Const of bus * int (* bus * constant value *)
7
8   type locals =
9     Bdecl of bus
10  | Adecl of bus * int (* bus * array length *)
11
12  type expr =
13    Num of int
14  | Id of string                          (* Bus name *)
15  | Barray of string * expr               (* Array reference *)
16  | Subbus of string * int * int          (* Subbus reference *)
17  | Unop of operator * expr               (* Unary operations *)
18  | Binop of expr * operator * expr       (* Binary operations *)
19  | Basn of string * expr                 (* bus name * value *)
20  | Aasn of string * expr * expr          (* Array name * array index * value *)
21  | Subasn of string * int * int * expr   (* Bus name * bit range * value *)
22  | Noexpr
23
24
25  type stmt =
26    Block of stmt list
27  | Expr of expr
28  | Pos of expr           (*Insert a rule that avoids having Pos inside if else!*)
29  | If of expr * stmt * stmt
30  | While of expr * stmt
31  | Switch of expr *  (expr * stmt) list  (* switch (expr) {...} *)
32  | Call of string * (expr list) * (expr list)
33
34  type fbody =  locals list * stmt list
35
36  type fdecl = {
37    portout : bus list;
38    fname   : string;
39    portin  : bus list;
40    body    : fbody;
41  }
42
43  type program = gdecl list * fdecl list
```

Parser.mly

```
1   %{open Ast%}
2
3   %token PLUS MINUS TIMES LT GT LTE GTE EQ NEQ
4   %token OR AND XOR SHL SHR NOT
5   %token IF ELSE WHILE FOR
6   %token ASN SEMI LPAREN RPAREN LBRACKET RBRACKET LBRACE RBRACE COMMA CONST
7   %token SWITCH CASE DEFAULT COLON POS ASYNC EOF
8   %token <int> NUM INT
9   %token <string> ID
10
11  /*Need to check precedence in C!*/
12  %nonassoc NOELSE
13  %nonassoc ELSE
14  %right ASN
15  %left EQ NEQ
16  %left LT GT LTE GTE
17  %left PLUS MINUS
18  %left TIMES DIVIDE MODULO
19  %left OR XOR
20  %left AND
21  %left SHL SHR
22  %right NOT
23  %nonassoc UMINUS /* Highest precedence for unary minus! */
24
25  %start program
26  %type <Ast.program> program
27
28  %%
29
30  /* a "program" is a list of "globals" and a list of "function declarators" */
31  program :
32                          { [],[] }
33  |  program gdecl        { ($2 :: fst $1), snd $1}
34  |  program fdecl        { fst $1, ($2 :: snd $1) }
35
36  /* Constant arrays are not useful */
37  gdecl :
38    CONST bdecl SEMI              { Const ($2, $2.init) }
39
40  bdecl :
41    async_opt spec ID init_opt      { { name = $3;
42                                        size = $2;
43                                        init = $4;
44                                        async = $1;
45                        isAssigned = Array.make $2 false} }
46  async_opt :
47                  { false }
48  | ASYNC         { true }
49
50  spec :
51    INT           { $1 }
52
53  init_opt :
54                  { 0 }
55  | ASN NUM       { $2 }
56
57  /* a "function declarator" is a "list of output bus", a "list of input bus" and a "body" */
58  fdecl :
59    out_port_list ID LPAREN port_list RPAREN LBRACE fbody RBRACE
60                          { { portout = $1;
61                              fname   = $2;
62                              portin  = $4;
63                              body    = $7 } }
64  /* Be careful while translating, to check that the user does not override
65    the ports with other local variables!!!! Raise an error! */
66
67  /* no need for parens if just one output bus */
68  out_port_list:
69                                  { raise (Failure("Empty output port list"))   }
70   | bdecl                        { [$1] }
```

```
71    | LPAREN port_list RPAREN {$2}
72
73   port_list :
74                          { [] }
75    | port_rlist           { List.rev($1) }
76
77   /* VHDL ports cannot be custom type */
78   port_rlist :
79     bdecl                 { [$1] }
80   | port_list COMMA bdecl { $3 :: $1 }
81
82
83   /* the "function body" is the list of "local variables" and a list of "statements"  */
84   fbody :
85                          { [], [] }
86   | fbody local           { ($2 :: fst $1), snd $1 }
87   | fbody stmt            { fst $1, ($2 :: snd $1) }
88
89   local :
90     vdecl SEMI            { $1 }
91
92   vdecl :
93     bdecl                 { Bdecl($1) }
94   | adecl                 { Adecl(fst $1, snd $1) }
95
96   adecl :
97     async_opt spec ID LBRACKET NUM RBRACKET init_opt { ( { name = $3;
98                                                           size = $2;
99                                                           init = $7;
100                                                          async = $1;
101                                      isAssigned = Array.make ($5) false}, $5 ) } /* setting the
     bitfield to the size of the array and not the size of the bus*/
102
103  stmt :
104    LBRACE stmt_list RBRACE                        { Block(List.rev $2) }
105  | asn_expr SEMI                                  { Expr($1) }
106  | POS LPAREN other_expr RPAREN SEMI                      { Pos($3) }
107  | IF LPAREN other_expr RPAREN stmt %prec NOELSE         { If($3, $5, Block([]))}
108  | IF LPAREN other_expr RPAREN stmt ELSE stmt            { If($3,$5,$7) }
109  | WHILE LPAREN other_expr RPAREN stmt                   { While($3, $5) }
110  | SWITCH LPAREN other_expr RPAREN LBRACE case_stmt case_list RBRACE{ Switch($3,$6::(List.rev $7)) }
111  /* Enforcing at least one case_stmt in the parser so no need to do it later */
112  | LPAREN actuals_list RPAREN ASN ID LPAREN actuals_list RPAREN SEMI     { Call($5, (List.rev $2),
     (List.rev $7)) }
113
114
115  stmt_list :
116  { [] }
117  | stmt_list stmt        { $2 :: $1 }
118
119
120  other_expr :
121    NUM                                            { Num($1) }
122  | ID                                             { Id($1) }
123  | ID LBRACKET other_expr RBRACKET                { Barray($1, $3) }
124  | ID LPAREN NUM COLON NUM RPAREN                 { Subbus($1, $3, $5) }
125  | ID LPAREN NUM RPAREN                           { Subbus($1, $3, $3) }
126  | MINUS other_expr %prec UMINUS                  { Unop(Umin, $2) }
127  | NOT other_expr %prec NOT                       { Unop(Not, $2) }
128  | other_expr PLUS other_expr                     { Binop($1, Add, $3) }
129  | other_expr MINUS other_expr                    { Binop($1, Sub, $3) }
130  | other_expr TIMES other_expr                    { Binop($1, Mul, $3) }
131  | other_expr LT other_expr                       { Binop($1, Lt, $3) }
132  | other_expr GT other_expr                       { Binop($1, Gt, $3) }
133  | other_expr LTE other_expr                      { Binop($1, Lte, $3) }
134  | other_expr GTE other_expr                      { Binop($1, Gte, $3) }
135  | other_expr EQ other_expr                       { Binop($1, Eq, $3) }
136  | other_expr NEQ other_expr                      { Binop($1, Neq, $3) }
137  | other_expr OR other_expr                       { Binop($1, Or, $3) }
138  | other_expr AND other_expr                      { Binop($1, And, $3) }
```

```
139   | other_expr XOR other_expr                        { Binop($1, Xor, $3) }
140   | other_expr SHL other_expr                        { Binop($1, Shl, $3) }
141   | other_expr SHR other_expr                        { Binop($1, Shr, $3) }
142   | LPAREN other_expr RPAREN                         { $2 }
143
144   /*No multiple assignments within the same line! a = b = c + 1 is not permitted*/
145   asn_expr :
146   | ID ASN other_expr                      { Basn($1, $3) }
147   | ID LBRACKET other_expr RBRACKET ASN other_expr { Aasn($1, $3, $6) }
148   | ID LPAREN NUM COLON NUM RPAREN ASN other_expr {Subasn($1, $3, $5, $8)}
149   | ID LPAREN NUM RPAREN ASN other_expr           {Subasn($1, $3, $3, $6)}
150
151   case_list :
152    { [] : (expr * stmt) list}
153   | case_list case_stmt  { $2 :: $1 }
154
155   case_stmt :
156     CASE other_expr COLON stmt_list {($2,Block($4)) }
157   | DEFAULT COLON stmt_list {(Noexpr,Block($3))}
158
159
160   /* ehdl.ml checks whether expressions are used*/
161   actuals_list :
162     actuals_rlist                 { List.rev $1 }
163
164   actuals_rlist :
165     other_expr                            { [$1] }
166   | actuals_list COMMA other_expr         { $3 :: $1 }
```

Scanner.mll

```
1   { open Parser }
2
3   rule token = parse
4     [' ' '\t' '\r' '\n']                  { token lexbuf }
5
6   (* Comment *)
7   | "/*"                                  { comment lexbuf }
8   | "//"                                  { sl_comment lexbuf }
9   (* Binary Operators *)
10  | '+'                                   { PLUS }
11  | '-'                                   { MINUS }
12  | '*'                                   { TIMES }
13  | '<'                                   { LT }
14  | '>'                                   { GT }
15  | "<="                                  { LTE }
16  | ">="                                  { GTE }
17  | "=="                                  { EQ }
18  | "!="                                  { NEQ }
19  | "||"                                  { OR }
20  | "&&"                                  { AND }
21  | "^"                                   { XOR }
22  | "<<"                                  { SHL }
23  | ">>"                                  { SHR }
24  (* Unary Operators *)
25  | "!"                                   { NOT }
26  (* Need to hande unary minus as well *)
27
28  (* types keywords *)
29  | "const"                               { CONST }
30
31  | "int("['0'-'9']+')' as lit            { INT(int_of_string (String.sub lit 4 (String.length lit -
    5))) }
32
33
34  | '='                                   { ASN }
35  | ','                                   { COMMA }
36  | ';'                                   { SEMI }
37  | ':'                                   { COLON }
38
39  | '('                                   { LPAREN }
40  | ')'                                   { RPAREN }
41  | '['                                   { LBRACKET }
42  | ']'                                   { RBRACKET }
43  | '{'                                   { LBRACE }
44  | '}'                                   { RBRACE }
45  (* keywords *)
46  | "if"                                  { IF }
47  | "else"                                { ELSE }
48  | "while"                               { WHILE }
49  | "switch"                              { SWITCH }
50  | "case"                                { CASE }
51  | "default"                             { DEFAULT }
52  | "POS"                                 { POS }
53  | "ASYNC"                               { ASYNC }
54  |
    "_"|"abs"|"access"|"after"|"alias"|"all"|"and"|"architecture"|"array"|"assert"|"attribute"|"begin"|"block
    as reserved { raise (Failure("Reserved Keyword " ^  reserved)) }
55  | ['a'-'z''A'-'Z'] ['a'-'z' 'A'-'Z' '0'-'9']* as lit  { ID(lit) }
56  | ['0'-'9']+ as lit                                     { NUM(int_of_string lit) }
57
58  | eof                                   { EOF }
59  | _ as char                             { raise (Failure("illegal character " ^ Char.escaped char)) }
60
61  and comment = parse
62    "*/"                                  { token lexbuf }
63  | _                                     { comment lexbuf }
64
65  and sl_comment = parse
66    '\n'                                  { token lexbuf }
67  | _                                     { sl_comment lexbuf }
```

Sast.ml

```
1   open Ast
2
3   module StringMap = Map.Make(String);;
4
5   (*Auxiliary functions*)
6   (*USE THIS FUNCTION FOR TYPE CHECKING WHEN NEEDED!*)
7   let bit_required x =
8   let s = if x < 0 then 1 else 0
9   in let x = abs x
10  in let log2 y = int_of_float ( ((log (float_of_int y)) /. (log 2.)) )
11  in let res = ((log2 x) + 1 + s)
12  in print_endline (string_of_int res); res
13
14  exception Error of string
15
16  type local_t =
17      In_port
18    | Out_port
19    | Int_signal
20
21  type types =
22      Bus
23    | Array
24    | Const
25    | Function
26    | Void
27
28  (* Covers both buses and array, out of bounds exceptions should done at run time *)
29  type symbol_table = {
30    parent : symbol_table option;
31    variables : (Ast.bus * int * types * local_t * bool) list;
32    isIf : bool array;
33    isWhile : bool array
34                        }
35
36  type translation_environment = {
37          scope : symbol_table; (* symbol table for vars *)
38  }
39
40  type function_decl = {
41           pout : Ast.bus list;
42           fid  : string;
43           pin  : Ast.bus list;
44           floc : translation_environment;
45       fcalls : function_decl list; (* list of other functions that are called by this function *)
46           fbod : s_stmt list;
47  }
48
49  and expr_detail =
50    Num of int
51  | Id of string
52  | Barray of Ast.bus * int * expr_detail
53  | Subbus of Ast.bus * int * int
54  | Unop of Ast.operator * expr_detail
55  | Binop of expr_detail * Ast.operator * expr_detail
56  | Basn of Ast.bus * expr_detail
57  | Aasn of Ast.bus * int * expr_detail * expr_detail
58  | Subasn of Ast.bus * int * int * expr_detail
59  | Noexpr
60
61  (* expression * type retuned * size *)
62  (* To return the size of expr is redundant, but helpful for type checking!!*)
63  (* The field returns the size of the bus, even with arr ays, because
64     the size of the array is already stored in the symbol_table *)
65  and expression = expr_detail * types * int
66
67  and s_stmt =
68      Block of s_stmt list
69    | Expr of expression
70    | If of expr_detail * s_stmt * s_stmt
```

```
71     | While of expr_detail * s_stmt
72     | Pos of expr_detail
73     | Switch of expr_detail * ((expr_detail * s_stmt)     list)
74     | Call of function_decl * (expr_detail list) * (expr_detail list)
75 
76 
77   let string_of_sast_type (t : types) =
78     match t with
79         Bus -> "Bus"
80       | Array -> "Array"
81       | Const -> "Const"
82       | Function -> "Function"
83       | Void -> "Void"
84 
85 
86   (* Find variable in scope *)
87   let rec find_variable (scope : symbol_table) name =
88           try
89                   List.find (
90                                   fun ( v , _, _, _, _ ) -> v.name = name
91                                   ) scope.variables
92           with Not_found ->
93                   match scope.parent with
94                             Some(parent) -> find_variable parent name
95                           | _ -> raise (Error("Variable " ^ name ^ " is undeclared"))
96 
97 
98   (* Add local to Symbol Table *)
99   let check_and_add_local (vbus, x, t, lt, dr) (env : translation_environment) =
100    let _ = print_endline ("Adding local " ^ vbus.name ^ " " ^ string_of_int x ^ " " ^ string_of_bool
     dr) in
101    if bit_required vbus.init > vbus.size then raise (Error("Initial value does not fit bus size:
     "^vbus.name)) else
102    if dr then ( if (x = 0)
103                                   then for i = 0 to vbus.size-1 do vbus.isAssigned.(i) <- true done
104                      else for i = 0 to x-1 do vbus.isAssigned.(i) <- true done)
105    else if (x = 0)
106                   then (for i = 0 to vbus.size-1 do vbus.isAssigned.(i) <- false;  done)
107     else for i = 0 to x-1 do vbus.isAssigned.(i) <- false done;
108 
109    let var = (vbus, x, t, lt, dr) in
110  (* Un-comment to print the list of locals name *)
111    (*let _ = print_endline vbus.name in*)
112    if List.exists (fun (varbus, _, _, _, _) -> varbus.name = vbus.name) env.scope.variables
113    then raise (Error("Multiple declarations for " ^ vbus.name))
114    else let new_scope = { parent = env.scope.parent;
115                          variables = var :: env.scope.variables; isIf = env.scope.isIf; isWhile =
     env.scope.isWhile}
116    in let new_env = { scope = new_scope;}
117         in  new_env
118 
119 
120  let check_operand_type type_1  =
121    match type_1 with
122      Bus | Array| Const -> true
123                  | _ -> raise(Error("Operand types should be bus or array or const"))
124 
125 
126  let check_types e1 op e2 =
127    let (detail_1, type_1, size_1) = e1
128    in let (detail_2, type_2, size_2) = e2
129       in let _ = check_operand_type type_1
130          in let _ = check_operand_type type_2
131            in
132                          match op with
133                          Or | And | Xor -> if size_1 != size_2 then raise(Error("Operand size
     mismatch in logical operation "))
134                                                             else size_1
135                  | Mul                              -> size_1 + size_2
136                  | Lt | Gt | Lte | Gte | Eq | Neq    -> 1
```

```
137                  | Shl | Shr                          -> if type_1 = Const then raise(Error("Bit
    Shift operators cant be used on Constants"))
138                                                             else size_1
139                  | Add|Sub                          -> Pervasives.max size_1 size_2
140                  | _                                -> raise(Error("Unary operators passed to
    binop"))
141
142 let check_switchable e1 t1 =
143   check_operand_type t1
144
145
146 let check_array_dereference varray size e1 t1 s1 =
147   match e1 with
148   | Noexpr -> raise(Error("Array index undefined: "^varray.name))
149   | Num(v) -> if v > (size - 1) then raise(Error("Array index out of bound "^varray.name)) else()
150   | _ -> if s1 > bit_required(size)
151                 then print_endline ("Warning: possible array index out of bound: "^varray.name) else
    ()
152
153 let check_basn env vbus e1 =
154  let _ = print_endline ("Checking variable "^vbus.name)
155   in let (detail, t, size) = e1
156    in match t with
157      Bus -> if size = vbus.size
158             then for i = 0 to vbus.size-1 do                  (* print_endline ("Checking bit " ^
    string_of_int i); *)
159                                                                if (vbus.isAssigned.(i) && not
    (env.scope.isWhile.(0)))
160                                                                then raise (Error("Variable
    "^vbus.name^" has more than one driver"))
161                                                                else vbus.isAssigned.(i) <- true
    done
162
163            else raise (Error("Bus size mismatch for "^vbus.name))
164     | Const -> for i = 0 to vbus.size-1 do if (vbus.isAssigned.(i) && not(env.scope.isWhile.(0)))
165                                          then raise (Error("Variable "^vbus.name^" has more than
    one driver"))
166                                          else vbus.isAssigned.(i) <- true done;
167
168                                (match detail with
169                                          Num(v) -> if (bit_required v) > vbus.size then raise(Error
    ("size mismatch "^vbus.name))
170                                                    else()
171                                        | Id(s) -> let b,_,_,_,_ = find_variable env.scope s in
172                                                     if b.size != vbus.size then raise(Error("size
    mismatch "^vbus.name))
173                                                     else()
174                                        | Subbus(b,strt,stop) -> if (abs (strt - stop)) !=
    ((vbus.size)-1)
175                                                                 then raise (Error("Size mismatch
    "^vbus.name))
176                                                                 else ()
177                                        | _ -> raise (Error("Illegal bus assignment:
    "^vbus.name))   )
178     | Array -> for i = 0 to vbus.size-1 do if (vbus.isAssigned.(i) && not(env.scope.isWhile.(0)))
179                                          then raise (Error("Variable "^vbus.name^" has more than
    one driver"))
180                                          else vbus.isAssigned.(i) <- true done;
181                                (match detail with
182                                        Barray(b,_,_) -> if b.size != vbus.size then raise(Error
    ("size mismatch "^vbus.name))
183                                                         else()
184                                        | _ ->  raise (Error("Expected variable of type bus or
    const "^vbus.name)) )
185     | _ -> raise (Error("Expected variable of type bus or const "^vbus.name))
186
187
188
189 let check_aasn env vbus size e1 e2 =
190   let(detail_e1,t_e1, size_e1) = e1
```

```
191      in match t_e1 with
192          Const -> let (ed2,t_e2, size_e2) = e2
193                            in (match t_e2 with
194                                      Const -> (match ed2 with
195                                                      Num(v) -> if size_e2 > vbus.size
196                                                            then raise(Error("Bus size mismatch for
    "^vbus.name)) else ()
197                                                    | Unop(uop,ued) -> (match uop with
198                                                                              Umin -> (match ued with
199                                                                                              Num(v) ->
    if size_e2 > vbus.size
200                                                                                                    then
    raise(Error("Bus size mismatch for "^vbus.name)) else ()
201                                                                                            | _ -> if
    size_e2 != vbus.size
202                                                                                                    then
    raise(Error("Bus size mismatch for "^vbus.name)) else ()        )
203                                                                            | _ ->  if size_e2 !=
    vbus.size
204                                                                                    then raise(Error
    ("Bus size mismatch for "^vbus.name)) else ()   )
205                                                    | _ -> if size_e2 != vbus.size
206                                                            then raise(Error("Bus size mismatch for
    "^vbus.name)) else ()   )
207                                    | _ ->  if size_e2 != vbus.size
208                                            then raise(Error("Bus size mismatch for "^vbus.name)) else
    ()      );
209                            (match detail_e1 with
210
211                                     Num(v) -> if v > (size-1) then raise(Error("Array index out of
    bound "^vbus.name))
212                                                     else() ;
213                                                       if (vbus.isAssigned.(v) && not(env.scope.isWhile.
    (0)))
214                                                     then raise (Error("Array index has more than one
    driver "^vbus.name))
215                                                       else vbus.isAssigned.(v) <- true
216                                   | Id(s) -> let b,_,_,_,_ = find_variable env.scope s in
217                                                  if b.init > (size-1) then raise(Error("Array index out
    of bound "^vbus.name))
218                                                     else if (vbus.isAssigned.(b.init) && not
    (env.scope.isWhile.(0)))
219                                                     then raise (Error("Array index has more than one
    driver "^vbus.name))
220                                                       else vbus.isAssigned.(b.init) <- true
221                                   | Subbus(b,x,y) -> let rec addone p = (function
222                                                                 0 -> p + 1
223                                                               | n -> let newp = p + (int_of_float ( 2. **
    (float_of_int n) ))
224                                                                       in addone newp (n-1)     )
225                                                        in let subn number strt stop = let r = abs (strt-
    stop)
226                                                               in let ones = addone 0 r in (number land
    ones)
227                                                        in let init = subn b.init x y
228                                                        in if init > (size-1) then raise(Error("Array
    index out of bound "^vbus.name))
229                                                               else if (vbus.isAssigned.(init) && not
    (env.scope.isWhile.(0)))
230                                                               then raise (Error("Array index has more
    than one driver "^vbus.name))
231                                                               else vbus.isAssigned.(init) <- true
232                                   | _ -> raise (Error("Array index is an expression of constants:
    "^vbus.name))  )
233          | Bus -> if(size_e1 > bit_required size)
234                    then print_endline ("Warning: possible array index out of bound: "^vbus.name) else
    ();
235                   for i = 0 to size-1 do if (vbus.isAssigned.(i) && not(env.scope.isWhile.(0)))
236                                                                                    then raise (Error
    ("Array index has more than one driver "^vbus.name))
```

```
237                                                    else vbus.isAssigned.(i) <- true done ;
238                         let (ed2,t_e2,size_e2) = e2
239                         in (match t_e2 with
240                                   Const -> (match ed2 with
241                                                   Num(v) -> if size_e2 > vbus.size
242                                                         then raise(Error("Bus size mismatch for
    "^vbus.name)) else ()
243                                                 | _ -> if size_e2 != vbus.size
244                                                         then raise(Error("Bus size mismatch for
    "^vbus.name)) else ()   )
245                                 | _ ->  if size_e2 != vbus.size
246                                         then raise(Error("Bus size mismatch for "^vbus.name)) else
    ()      )
247       | _ -> raise (Error("Array index should be const or bus "^vbus.name))
248
249
250
251 let check_subbus vbus x y =
252   let (x,y) = if x < y then (x,y) else (y,x)
253   in if x >= 0 && y <= vbus.size && x <= y then ()
254   else raise (Error("Incorrect subbus dereference for "^vbus.name))
255
256 let check_subasn env vbus x y e1 =
257   let (x,y) = if x < y then (x,y) else (y,x)
258   in let (detail, t, size) = e1
259   in match t with
260     Bus -> let _ = check_subbus vbus x y
261            in if size = y-x+1
262            then for i = x to y do if (vbus.isAssigned.(i) && not(env.scope.isWhile.(0)))
263                                                    then raise (Error("Variable "^vbus.name^"
    has more than one driver"))
264                                              else vbus.isAssigned.(i) <- true done
265            else raise (Error("Size of expression is different from subbus width for "^vbus.name))
266      | Const -> (let _ = check_subbus vbus x y
267                  in let _ = match detail with
268                                      Num(v) -> if(bit_required v) > y-x+1
269                                                then raise (Error("Size of expression is bigger
    than subbus width for "^vbus.name))
270                                                else ()
271                                    | Id(s) ->  let b,_,_,_,_ = find_variable env.scope s in
272                                                    if b.size != y-x+1
273                                                    then raise (Error("Size of expression is
    different from subbus width for "^vbus.name))
274                                                    else ()
275                                    | Subbus(_,strt,stop) -> if (abs (strt - stop)) != y-x
276                                                           then raise (Error("Size of
    expression is different from subbus width for "^vbus.name))
277                                                           else ()
278                                    | _ -> raise (Error("Const expressions are not supported"))
279                       in for i = x to y do if (vbus.isAssigned.(i) && not(env.scope.isWhile.(0)))
280                                              then raise (Error("Variable "^vbus.name^" has more than
    one driver"))
281                                              else vbus.isAssigned.(i) <- true done)
282       | Array -> (let _ = check_subbus vbus x y
283                  in if size = y-x+1
284                  then for i = x to y do if (vbus.isAssigned.(i) && not(env.scope.isWhile.(0)))
285                                                    then raise (Error("Variable "^vbus.name^"
    has more than one driver"))
286                                              else vbus.isAssigned.(i) <- true done
287                  else raise (Error("Size of expression is different from subbus width for
    "^vbus.name))   );
288                  (match detail with
289                         Barray(b,_,_) -> ()
290                         | _ ->  raise (Error("Expected variable of type bus or const "^vbus.name)) )
291       | _ -> raise (Error("Expected variable of type bus "^vbus.name))
292
293
294 let pred (b,_,_,_,_) (b',_,_,_,_) =
295                  let _ =
296                  for i=0 to ((Array.length b.isAssigned) - 1) do
```

```
297                         b.isAssigned.(i) <- b'.isAssigned.(i) || b.isAssigned.(i)
298                 done
299         in true
300
301 let check_function_outvars env e vbus2 =
302   let (ed, t,sz) = e
303   in match t with
304     Bus -> (
305            match ed with
306            Id(vname) -> (
307                     let vbus1, _, vtype, _, _ =
308                                             try
309                                                     find_variable env.scope vname (* locate a
    variable by name *)
310                                             with Not_found ->
311                                                     raise (Error("undeclared identifier " ^
    vname))
312                     in
313                                     if vbus1.size = vbus2.size then
314                                                         let _ = (for i = 0 to vbus1.size-1
315                                                                 do if (vbus1.isAssigned.(i)
    && not(env.scope.isWhile.(0)))
316                                                                         then raise(Error
    ("Bus "^vbus1.name^" has more than one driver"))
317                                                             else vbus1.isAssigned.(i) <-
    true done) in true
318                                             else raise(Error("Function output variable width
    mismatch "^vbus1.name^" "^vbus2.name))
319                     )
320                 | Subbus(vbus,x,y) -> (
321                               let (x,y) = if x < y then (x,y) else (y,x)
322                               in if(vbus2.size = y-x+1)
323                               then (let _ = (for i = x to y
324                                             do if (vbus.isAssigned.(i) && not(env.scope.isWhile.
    (0)))
325                                             then raise (Error("Variable "^vbus.name^" has more than
    one driver"))
326                                             else (vbus.isAssigned.(i) <- true) done) in true)
327                               else raise (Error("Size mismatch in function output assignment
    "^vbus.name))
328                              )
329                 |_ -> (raise(Error("Expected bus or subbus")))    )
330          | Array -> ( match ed with
331                                 Barray(vbus, sz, exd) ->
332                                 if vbus.size != vbus2.size
333                                         then raise (Error("Size mismatch in function output
    assignment "^vbus.name))
334                                   else let _ =
335                                         ( match exd with
336                                                   Num(idx) -> (if (vbus.isAssigned.(idx) && not
    (env.scope.isWhile.(0)))
337                                                                 then raise (Error("variable
    "^vbus.name^" has more than one driver"))
338                                                                 else vbus.isAssigned.(idx) <- true )
339                                             | Id(s) -> let gl_env = (match env.scope.parent with
340                                                                           Some(g) -> g
341                                                                         | _ -> raise (Error("No
    Global Environment")) )
342                                                               in
343                                                              (try let b,_,_,_,_ = find_variable
    gl_env s
344                                                                 in (if (vbus.isAssigned.(b.init) &&
    not(env.scope.isWhile.(0)))
345                                                                 then raise (Error("variable
    "^vbus.name^" has more than one driver"))
346                                                                 else vbus.isAssigned.(b.init) <-
    true )
347                                                              with Error(_) ->  raise (Error
    ("Function input and output ports must be static: " ^vbus.name)) )
348                                             | _ -> raise (Error("Function input and output
```

```
ports must be static: " ^vbus.name))  )
349                                             in true
350                              |_ -> raise(Error("Expected type of variable Barray "))        )
351     | _ -> raise (Error("function assignment must be to a bus or an array"))
352
353
354   let check_function_invars env e vbus2 =
355     let (ed, t, sz) = e
356      in match t with
357        Bus -> (
358              match ed with
359              Id(vname) -> (
360                       let vbus1, _, vtype, _, _ =
361                                             try find_variable env.scope vname (* locate a
variable by name *)
362                                             with Not_found -> raise (Error("undeclared
identifier " ^ vname))
363                       in if vbus1.size != vbus2.size then raise(Error("Function input variable width
mismatch "^vbus1.name^" "^vbus2.name))
364                           else true        )
365                  | Subbus(vbus,x,y) -> (
366                             let (x,y) = if x < y then (x,y) else (y,x)
367                             in if(vbus2.size != y-x+1)
368                             then raise (Error("Size mismatch in function output assignment
"^vbus.name))
369                             else true         )
370                  |_ -> (raise(Error("Expected bus or subbus: ")))   )
371     | Array -> ( match ed with
372                                Barray(vbus, sz, exd) ->
373                                if vbus.size != vbus2.size
374                                        then raise (Error("Size mismatch in function output
assignment "^vbus.name))
375                                  else let _ =
376                                        ( match exd with
377                                                  Num(idx) -> ()
378                                                | Id(s) -> let gl_env = (match env.scope.parent with
379                                                                         Some(g) -> g
380                                                                       | _ -> raise (Error("No
Global Environment")) )
381                                                              in
382                                                             (try let _ =find_variable gl_env s
383                                                                in ()
384                                                             with Error(_) ->  raise (Error
("Function input and output ports must be static: " ^vbus.name)) )
385                                                | _ -> raise (Error("Function input and output
ports must be static: " ^vbus.name))  )
386                                             in true
387                              |_ -> raise(Error("Expected type of variable Barray "))        )
388     | Const -> (match ed with
389                                        Num(v) -> if(bit_required v) > vbus2.size
390                                                  then raise (Error("Size of expression is bigger
than subbus width for "^vbus2.name))
391                                                  else true
392                                      | Id(s) ->  let b,_,_,_,_ = find_variable env.scope s in
393                                                        if b.size != vbus2.size
394                                                        then raise (Error("Size of expression is
different from subbus width: "^s))
395                                                        else true
396                                      | Subbus(vbus,strt,stop) -> if ((abs (strt - stop))+1) !=
vbus2.size
397                                                                 then raise (Error("Size of
expression is different from subbus width: "^vbus.name))
398                                                                 else true
399                                      | _ -> raise (Error("Const expressions in function
call")) )
400     | _ -> raise (Error("function input must be to a bus a const or an array reference"))
401
402
403   let check_call env out_actuals in_actuals fd =
404     let _ = print_endline ("Checking function call: "^fd.fid)
```

```
405        in
406      let _ =( try List.for_all2 (check_function_outvars env) out_actuals fd.pout
407            with (Invalid_argument(_)) -> raise (Error("Port mismatch for function call: "^fd.fid)) )
408      in let _ = ( try List.for_all2 (check_function_invars env) in_actuals fd.pin
409                    with (Invalid_argument(_)) -> raise (Error("Port mismatch for function call:
     "^fd.fid))      )
410          in ()
411
412
413    (* Check expressions *)
414    (* This returns expr_detail * types * int *)
415    let rec chk_expr function_table env = function
416    (* An integer constant: convert and return Int type *)
417            Ast.Num(v) ->
418            let min_size = bit_required v in
419            Num(v), Const, min_size (*If assigned to the bus vbus, check that vbus.size >= min_size!!*)
420    (* An identifier: verify it is in scope and return its type *)
421      | Ast.Id(vname) ->
422            let vbus, _, vtype, _, _ =
423                    try
424                            find_variable env.scope vname (* locate a variable by name *)
425                    with Not_found ->
426                            raise (Error("undeclared identifier " ^ vname))
427            in Id(vbus.name), vtype, vbus.size (*Be careful!!! An Id could be a constant, a bus or an
     array!*)
428      | Ast.Binop(e1, op, e2) ->
429            let e1 = chk_expr function_table env e1 (* Check left and right children *)
430                    and e2 = chk_expr function_table env e2
431            in let output_size = check_types e1 op e2 in
432            let (e1,t1,_) = e1 and (e2,t2,_) = e2
433            in let t = if ((t1 == Const) && (t2 == Const)) then Const else Bus
434            in Binop(e1, op, e2), t, output_size (* Success: result is bus *)
435      | Ast.Basn(vname, e1) ->
436            let _ = print_endline ("Checking bus assignment for "^vname) in
437                    let e1 = chk_expr function_table env e1
438      and vbus, _, _, _, _ = find_variable env.scope vname
439      in let _ = check_basn env vbus e1
440            in let (e1, _, _) = e1
441            in Basn(vbus, e1), Bus, vbus.size
442      | Ast.Subasn(vname, x, y, e1) ->
443            let e1 = chk_expr function_table env e1
444      and vbus, _, _, _, _ = find_variable env.scope vname
445      in let _ = check_subasn env vbus x y e1
446            in let (e1, _, _) = e1
447            in Subasn(vbus, x, y, e1), Bus, (abs(x-y) +1);
448      | Ast.Aasn(vname, e1, e2) ->
449                    let e1 = chk_expr function_table env e1
450                    and e2 = chk_expr function_table env e2
451                    and vbus, size, _, _, _ = find_variable env.scope vname
452        in let _ = check_aasn env vbus size e1 e2
453            in let (e1, _, _) = e1 and (e2, _, _) = e2
454            in Aasn(vbus, size, e1, e2), Bus, vbus.size
455      (* NEED TO CHECK OUTPUT PORTS MATCH WITH LOCALS ASSIGNMENT!!!*)
456      | Ast.Unop(op, e1) ->
457            let (e1, t1, s1)= chk_expr function_table env e1
458          in Unop(op, e1), t1, s1
459      | Ast.Subbus(vname, x, y) ->
460            let _ = print_endline ("Check Sub bus asn for: "^vname^" with params: "
461                                            ^ (string_of_int x) ^ " " ^ (string_of_int y)) in
462            let vbus, _, _, _, _ = find_variable env.scope vname
463            in check_subbus vbus x y;
464            Subbus(vbus, x, y), Bus, (abs(x-y) +1)
465      | Ast.Barray(vname, e1) ->
466            let (e1, t1, s1) = chk_expr function_table env e1
467          and varray, size, vtype, _, _ = find_variable env.scope vname
468          in check_array_dereference varray size e1 t1 s1;
469        Barray(varray, size, e1), vtype, varray.size
470      | Ast.Noexpr -> Noexpr, Void, 0
471
472    (*Check Statements*)
```

```
473  let rec chk_stmt function_table env = function
474      Ast.Expr(e) -> Expr(chk_expr function_table env e)
475    | Ast.If(e1, s1, s2) ->
476          let e1, t1, _ = chk_expr function_table env e1
477       in
478      let temp = { scope =
479                   { env.scope with
480                          variables = List.map (
481                                  fun (b, s, t, l, f) ->
482                                          ( { b with isAssigned =
483                                                 Array.copy b.isAssigned },
484                                           s, t, l, f )
485                                  ) env.scope.variables
486                  }
487          }
488      in let stmt_1 = chk_stmt function_table temp s1
489         in
490      let stmt_2 = chk_stmt function_table env s2
491      in let _ = List.for_all2 pred (env.scope.variables) (temp.scope.variables)
492        in If(e1,  stmt_1, stmt_2)
493    | Ast.While(e1, s1) ->
494          let _ = env.scope.isWhile.(0) <- true in
495          let e1, t1, _= chk_expr function_table env e1
496       in (* check_conditional e1 t1;*)
497          let statement = chk_stmt function_table env s1
498       in let _ = env.scope.isWhile.(0) <- false in
499          While(e1, statement)
500    | Ast.Pos(e1) ->
501          let e1, t1, _= chk_expr function_table env e1
502       in (* check_pos_expr e1;*)
503          Pos(e1)
504    | Ast.Block(slist) ->
505          let run_chk_stmt (env : translation_environment) (actual : Ast.stmt) =
506            let s1 = chk_stmt function_table env actual
507              in s1
508          in let new_stmt_list =
509          let rec stmt_helper l = function
510              [] -> List.rev l
511            | hd::tl -> let new_l = ( run_chk_stmt env hd )::l
512                  in stmt_helper new_l tl
513          in stmt_helper [] slist
514  (* Un-comment to check if Blocks are parsed *)
515          (*in let _ = print_endline "parsed a Block"*)
516          in Block(new_stmt_list)
517    | Ast.Switch(e, caselist) ->
518          let e, t1, _ = chk_expr function_table env e
519          in let _ = check_switchable e t1
520           in let chk_case_list (env : translation_environment) ( (e1, s1) : (Ast.expr * Ast.stmt) ) =
521              let e1, t1, _ = chk_expr function_table env e1 in
522                          let _ = if not(t1 = Const || t1 = Void) (* Void represents the default case
     *)
523               then raise(Error("Case constants must be CONSTANTS. Received case expression of type:
     " ^ string_of_sast_type t1))
524                                  else ()
525                  in let s1 = chk_stmt function_table env s1
526                  in (e1, s1)
527          in let rec clist_helper l = function
528                                 [] -> List.rev l
529                               | hd::tl ->
530                                          let temp = { scope =
531                                                          { env.scope with
532                                                          variables = List.map (
533                                                                  fun (b, s, t, l, f) ->
534                                                                  ( { b with isAssigned =
535                                                                        Array.copy b.isAssigned },
536                                                                  s, t, l, f )
537                                                                  ) env.scope.variables
538                                                          }
539                                  }
540                                                          in  let new_l = ((chk_case_list temp
```

```
hd),temp) :: l
541                                                     in clist_helper new_l tl
542     in let nlist = clist_helper [] caselist
543        in let clist = List.map (fun ((x,y),z) -> let _ = (List.for_all2 pred env.scope.variables
z.scope.variables) in (x,y)) nlist
544 (* Un-comment to check if Switch is parsed *)
545         (*in let _ = print_endline "parsed a Switch"*)
546         in Switch(e, clist)
547
548   | Ast.Call(fname, out_list, in_list ) ->
549         let _ = print_endline ("Checking function call: "^fname) in
550         let _ = List.iter (fun x -> match x with
551                                 Ast.Id(v) -> print_endline ("Assigning to: "^v)
552                         | _ -> print_endline "This better be an array deref or a subbus") out_list
in
553         let _ = List.iter (fun x -> match x with
554                                 Ast.Id(v) -> print_endline ("Call param: "^v)
555                         | _ -> print_endline "Check expr passed as call param:") in_list in
556         let func_decl =
557                 try StringMap.find fname function_table
558          with Not_found -> raise (Failure ("undefined function " ^ fname))
559     in let inlist = List.fold_left
560     ( fun l x -> let e1 =  chk_expr function_table env x in e1::l ) [] in_list
561     in let outlist = List.fold_left
562     ( fun l x -> let e1 =  chk_expr function_table env x in e1::l ) [] out_list
563        in let _ = print_endline "Just before checking function call"
564        in let _ = check_call env outlist inlist func_decl
565           in let outlist = List.fold_left (fun l x -> let (e1, _, _) = x in e1::l) [] outlist
566              in let inlist = List.fold_left (fun l x -> let (e1, _, _) = x in e1::l) [] inlist
567               (* Uncomment to check if Function Call is parsed *)
568                 (* in let _ = print_endline "Function Call parsed" *)
569                         in Call(func_decl, outlist, inlist)
570
571
572 (* Function translation Ast -> Sast. Build Symbol table; parse statements*)
573 let check_func (env : translation_environment) (portin : (Ast.bus list)) (portout : (Ast.bus list))
(body : Ast.fbody) function_table =
574   let _ = print_endline "Checking fucntion... " in
575   let pin_env = List.fold_left (
576     fun (pin_env : translation_environment) (actual : Ast.bus) ->
577       check_and_add_local (actual, 0, Bus, In_port, true) pin_env
578                                 ) env portin
579   in
580   let pout_env = List.fold_left (
581     fun (pout_env : translation_environment) (actual : Ast.bus) ->
582       check_and_add_local (actual, 0, Bus, Out_port, false) pout_env
583                                 ) pin_env portout
584   in
585   let (locals_list, stmts) = body
586         in let full_env = List.fold_left (
587                                                 fun (env : translation_environment) (actual :
Ast.locals) ->
588                             match actual with
589                                   Bdecl(vbus) -> check_and_add_local (vbus, 0, Bus, Int_signal,
false) env
590                               | Adecl(vbus, size) -> check_and_add_local (vbus, size, Array,
Int_signal, false) env
591                                                   ) pout_env locals_list
592         in let run_chk_stmt (env : translation_environment) (stmt_lst, call_lst) (actual : Ast.stmt)
=
593                                 let s1 = chk_stmt function_table env actual
594                                 in let call_lst =
595                                         (match actual with
596                                                         Ast.Call(fname, _, _ ) ->
597                                                         let f_decl =
598                                                                 try StringMap.find fname
function_table
599                                                         with Not_found -> raise (Failure
("undefined function " ^ fname))
600                                                 in f_decl::call_lst
```

```
601                                                         | x -> call_lst (* do nothing *) )
602                         in (s1::stmt_lst, call_lst)
603 
604         in let (new_stmt_list,call_lst) =
605         List.fold_left (run_chk_stmt full_env) ([],[]) (List.rev stmts)
606         in (full_env, List.rev call_lst, List.rev new_stmt_list)
607 
608 
609 (* Function table *)
610 let func (env : translation_environment) (astfn : Ast.fdecl) tmp_ftable =
611   let func_scope = { parent = Some(env.scope); variables = []; isIf = Array.make 2 false; isWhile =
Array.make 1 false }
612   in let func_env = {scope = func_scope }
613     in let (chk_floc, chk_calls, chk_fbod) = check_func func_env astfn.portin astfn.portout
astfn.body tmp_ftable
614      in let fobj = {    pout = astfn.portout;
615                         fid  = astfn.fname;
616                             pin  = astfn.portin;
617                             floc  = chk_floc;
618                             fcalls = chk_calls;
619                             fbod = chk_fbod; }
620         in let new_ftable = StringMap.add astfn.fname fobj tmp_ftable
621         in let _ = print_endline ("Added function "^astfn.fname)
622           in new_ftable
623 
624 
625 (* Program transaltion Ast -> Sast *)
626 let prog ((constlist : Ast.gdecl list), (funclist : Ast.fdecl list)) =
627   let _ = print_endline "Starting prog..." in
628   let clist = List.map (
629     fun (gdecl : Ast.gdecl)->
630       let Ast.Const(vbus, value) = gdecl
631       in
632       let dummy_env = {scope = { parent = None; variables = []; isIf = Array.make 2 false; isWhile
= Array.make 1 false}} (*Workaround for while/if multiple assignment *)
633         in
634       let _ = check_basn dummy_env vbus (Num(value), Const, bit_required value)
635       in (vbus, value, Const, Int_signal, true)
636                            ) (List.rev constlist)
637   in let global_scope = { parent = None; variables = List.rev clist; isIf = Array.make 2 false;
isWhile = Array.make 1 false}
638         in let global_env = { scope = global_scope }
639 
640            in let rec create_map mymap = function
641                                           [] -> mymap
642                                         | hd::tl -> let new_mymap = func global_env (hd : Ast.fdecl)
mymap
643                                                         in create_map new_mymap tl
644            in let ftable = create_map StringMap.empty (List.rev funclist)
645                 in global_env, ftable
```

Clocktab.ml

```
1   open Ast
2   open Sast
3
4   module Im = Map.Make(struct
5     type t = Sast.expr_detail
6     let compare x y = Pervasives.compare x y
7   end)
8
9   module Sm = Map.Make(struct
10    type t = string
11    let compare x y = Pervasives.compare x y
12  end)
13
14  (*Call this function when an assignment occurs*)
15  (*takes clock cycle #, assignment, current assignment map and updates it*)
16  let update_asn (asn_expr : Sast.expr_detail) asn_map =
17          let vname = match asn_expr with
18            Basn(x,_) -> x.name
19          | Aasn(x,_,_,_) -> x.name
20          | Subasn(x,_,_,_) -> x.name
21          | x -> raise (Error("not an assignment"))
22          in Im.add asn_expr vname asn_map
23
24  (*Debug: check assignments - this is a kind of int. representation*)
25  let asn_to_string k _ s =
26          let s = s^ ""
27          in let _ = match k with
28          Basn(x,_) -> print_endline (x.name^" -> assigned")
29          | Aasn(x,i,_,_) -> print_endline ((x.name)^" "^(string_of_int i)^" -> assigned")
30          | Subasn(x,a,b,_) -> print_endline ((x.name)^" range "^(string_of_int a)^":"^(string_of_int
    b)^" -> assigned")
31          | x -> raise (Error("not an assignment"))
32          in s
33
34  let print_asn_map asn_map = Im.fold asn_to_string asn_map ""
35
36  (*CHECK ASYNC!*)
37  let asn_map_to_list k _ (sync,async) = match k with
38            Basn(x,e1) -> if x.async then (sync,(Basn(x,e1))::async) else ((Basn(x,e1))::sync,async)
39          | Aasn(x,sz,e1,e2) -> if x.async then (sync,(Aasn(x,sz,e1,e2))::async) else ((Aasn
    (x,sz,e1,e2))::sync,async)
40          | Subasn(x,a,b,e1) -> if x.async then (sync,(Subasn(x,a,b,e1))::async) else ((Subasn
    (x,a,b,e1))::sync,async)
41          | x -> raise (Error("not an assignment"))
42  let get_asn asn_map = Im.fold asn_map_to_list asn_map ([],[])
43
44
45  (*Auxiliary functions*)
46  (* insert non-duplicates *)
47  let rec insert_uniq l name =
48          try let _ = List.find ( fun ( s ) -> s = name ) l in l
49          with Not_found-> name::l
50  (* returns a list whose vals are unique *)
51  let uniq lst =
52      List.fold_left (fun l s -> insert_uniq l s) [] lst
53
54  (* adds range to a list *)
55  let range_list bl (a,b) = if a >= b then (a,b)::bl else (b,a)::bl
56
57  (* returns non common range list *)
58  let others_range_list (a0,b0) l =
59          let (a0,b0) = if a0 >= b0 then (a0, b0) else (b0,a0)
60          in let l = List.rev ( List.sort (fun (c1,_) (c2,_) -> Pervasives.compare c1 c2) l )
61          in let add_range r (c1,c2) = if( (a0 > c1) && (b0 > c1) )then (a0,b0)::r else
62                                       if( (a0 > c1) && (b0 >= c2) )then (a0,(c1+1))::r else
63                                       if( (a0 > c1) && (b0 < c2) )then (a0,(c1+1))::(((c2-1),b0)::r)
    else
64                                       if( (a0 >= c2) && (b0 < c2) )then  ((c2-1),b0)::r else
65                                         (a0,b0)::r
66          in let tmp = List.fold_left add_range ([]) l
```

```
67         in let res = uniq tmp in res
68
69  (* returns non common assignments to an array *)
70  let ncaasn x sz l =
71  let add_index_list il = function
72            Aasn(x,i,e1,_) -> if e1 = Id("constant") then i::il else raise(Error("Multiple
    assignments to "^x.name^" within the while statement!"))
73          | _ -> raise (Error("Illegal assignment to variable "^x.name))
74  in let il = List.fold_left add_index_list [] l
75  in let il = sz::il
76  in let il = List.sort (fun c1 c2 -> Pervasives.compare c1 c2) il
77  in let rec irange (ci,l) maxi = if ((ci < maxi) && (ci < sz)) then irange ((ci+1),(ci::l)) maxi else
    (ci+1), (List.rev l)
78  in let _,il = List.fold_left irange (0,[]) il
79  in let index_to_aasn i = Aasn(x,i,Id("constant"),Id("constant"))
80  in List.map index_to_aasn il
81
82  (*Get all assignments to vname*)
83  let get_asn_to_vname asn v (vname,l) = vname, (if v = vname then asn::l else l)
84
85  (*Get assignments from map1 not reported in map2*)
86  let get_nc_asn map1 map2 =
87          let list_nc_asn asn vname l =
88             if Im.mem asn map2 then l else
89             let _, tmp_asn_l = Im.fold get_asn_to_vname map2 (vname,[])
90             in let nca ncal = function
91                    [] -> asn::ncal
92                  | hd::tl -> (match hd with
93                                  Basn(x,e1) -> ncal (*if bus -> single assignment!*)
94                                | Aasn(x,sz,e1,e2) -> (match e1 with
95                                (*If not Id("constant") the array is entirely assigned by the while
    loop: index not known at compile time*)
96                                                  Id("constant") -> (match asn with
97                                                          Aasn(x0,sz0,e01,_) -> ( match e01 with
98                                                                        Id("constant") -> asn::ncal
99                                                                        (*If not Id("constant")
    then return all Aasn not in hd::tl*)
100                                                                       | _ ->  let nl = ncaasn x0
    sz0 (hd::tl)
101                                                                               in List.fold_left
    (fun l e -> e::l) ncal nl       )
102                                                       | _ -> raise (Error("Illegal assignment to
    variable "^x.name)) )
103                                                | _ -> if tl = [] then ncal else raise(Error
    ("Multiple assignments to "^x.name^" within the while statement!"))
104                                                       (*The assignment is in common for sure!*)  )
105                               | Subasn(x,a,b,e1) -> (match asn with
106                                                       Subasn(_,a0,b0,_) ->
107                                                       let bitl = [] (*range_list [] (a,b)*)
108                                                       in let rec fill_bitl bl = (function
109                                                                  [] -> bl
110                                                                | hd1::tl1 -> (match hd1 with
111                                                                               Subasn(_,a1,b1,_) -
    > fill_bitl (range_list bl (a1,b1)) tl1
112                                                                             | _ -> raise (Error
    ("Wrong list in get_nc_asn"))   )       )
113                                                         in let bitl = fill_bitl bitl (hd::tl)
114                                                       in let subasn_list =
115                                                        let subasn_from_range l (a2,b2) = (Subasn
    (x,a2,b2,e1))::l
116                                                        in let orl = others_range_list (a0,b0) bitl
117                                                        in List.fold_left subasn_from_range ncal
    orl
118                                                       in subasn_list
119                                                       | Basn(_,_) -> let bitl = [] (*range_list
    [] (a,b)*)
120                                                       in let rec fill_bitl bl = (function
121                                                                  [] -> bl
122                                                                | hd1::tl1 -> (match hd1 with
123                                                                               Subasn(_,a1,b1,_) -
```

```
    > fill_bitl (range_list bl (a1,b1)) tl1
124                                                                                  | _ -> raise (Error
    ("Wrong list in get_nc_asn"))    )       )
125                                                           in let bitl = fill_bitl bitl (hd::tl)
126                                                         in let subasn_list =
127                                                          let subasn_from_range l (a2,b2) = (Subasn
    (x,a2,b2,e1))::l
128                                                          in let orl = others_range_list
    ((x.size-1),0) bitl
129                                                          in List.fold_left subasn_from_range ncal
    orl
130                                                         in subasn_list
131                                                         | _ -> raise (Error("Illegal assignment to
    variable "^x.name)) )
132                                 | _ -> raise (Error("not an assignment"))       )
133                 in nca l tmp_asn_l
134         in let nc_list = Im.fold list_nc_asn map1 []
135         in nc_list
```

Ehdl.ml

```
1   open Ast
2   open Sast
3   open Clocktab
4
5   module CompMap = Map.Make(struct
6     type t = string
7     let compare x y = Pervasives.compare x y
8   end)
9
10  exception ErrorS of string
11
12  type s_env = {
13      sens_list : string list;
14             } (* can add more stuff to this, list of locals for example, so POS can update it *)

15
16
17  (* insert non-duplicate fcalls, should really do operator overloading and use a generic uniq
    function *)
18  let rec insert_call l c =
19          try let _ = List.find ( fun ( s ) -> s.fid = c.fid ) l in l
20          with Not_found-> c::l
21  (* returns a list whose vals are unique *)
22  let uniq_calls clist =
23      List.fold_left (fun l c -> insert_call l c) [] clist
24
25  (* Utility function for producing a delimeter separated string from a list*)
26  let rec delim_sprt delim p = match List.length p with
27         0 -> "" |
28         x -> let head = (List.hd p)
29                (* don't want to put in a delim after the last element, hence the if check *)
30               in if ( x = 1 ) then head else  (head ^ delim ^ (delim_sprt delim (List.tl p)) )
31
32  let rec port_descr_list p inOrOut (ports: Ast.bus list) = match ports with
33         [] -> p
34         | hd::tl -> let typedescr =
35                       (match hd.size with
36                          0 -> raise (Failure ("bus size cannot be zero " ))
37                          (* not doing std_logic because then we have to convert 1 to '1' *)
38                    | x -> " std_logic_vector(" ^ string_of_int(hd.size-1) ^ " downto 0)" )
39                     in  let s = "\t" ^ hd.name ^ " : "  ^ inOrOut ^ typedescr
40                         in port_descr_list  (s::p) inOrOut tl
41
42
43  let port_gen cname cobj =
44      let inportlist = port_descr_list [] "in " cobj.pin
45          in let portList =  port_descr_list inportlist "out" cobj.pout
46            in (delim_sprt ";\n" (List.rev portList))
47
48
49  (*Auxiliary function: adds conv_std_logic_vector *)
50  let num_to_slv v size =
51     try let _ = int_of_string v
52          in "ieee.std_logic_arith.conv_std_logic_vector("^v^","^(string_of_int size)^")"
53     with Failure(s) -> v
54
55  (*Auxiliary function: adds conv_integer *)
56  let to_int v =
57     try let _ = int_of_string v
58          in v
59     with Failure(s) -> "conv_integer("^v^")"
60
61
62  let translate (genv, ftable) =
63  let create_component cname cobj components =
64  (*cloc is the local symbol table, required while translating statements and expressions*)
65  let (cloc, cname) = (cobj.floc,cobj.fid)
66    in let libraries = "\nlibrary ieee;\n" ^
67      "use ieee.std_logic_1164.all;\n" ^
68      "use ieee.std_logic_signed.all;\n\n\n"
```

```
69
70     in let entity cname cobj = (* entity *)
71              let s = port_gen cname cobj
72               in "entity " ^ cname ^ "  is \n\nport (\n" ^
73                  "\tclk : in std_logic;\n\trst : in std_logic;\n" ^ s ^ ");\n\nend " ^ cname ^ ";\n
\n"
74
75
76
77  (**************While loop processing*****************)
78
79  in let translate_while (wcond : Sast.expr_detail) (wblock : Sast.s_stmt list) curr_asn_map
(curr_cc : int) (curr_fc : int)=
80
81  (* Evaluate expressions *)
82  let rec weval e env asn_map cc = match e with
83      Num(i) -> string_of_int i,string_of_int i, env, asn_map
84      | Id(i) -> (i ^ "_r" ^ (string_of_int (cc+1))),(i ^ "_r" ^ (string_of_int (cc))), env, asn_map
85      | Barray(bs, _, e1) -> let v1,v2 = match e1 with
86                            Num(i) -> (string_of_int i), (string_of_int i)
87                          | x -> let i1,i2, _, _ = weval x env asn_map cc
88                                  in ("ieee.std_logic_unsigned.conv_integer(" ^ i1 ^ ")"),
("ieee.std_logic_unsigned.conv_integer(" ^ i2 ^ ")")
89                  in (bs.name^"_r"^(string_of_int (cc+1))) ^ "(" ^ v1 ^ ")", (bs.name^"_r"^
(string_of_int (cc))) ^ "(" ^ v2 ^ ")", env, asn_map
90                  (* Using "a" rather than "a(i)" in the sensitivity list, which is fine, because the
list must be static *)
91      | Subbus(bs, strt, stop) -> let range =
92                      if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
strt) ^ ")" else
93                                          "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int
stop) ^ ")"
94                  in (bs.name ^ "_r" ^ (string_of_int (cc+1))) ^ range, (bs.name ^ "_r" ^
(string_of_int (cc))) ^ range, env, asn_map
95      | Unop(op,e1) -> let v11,v12, _, _ = weval e1 env asn_map cc in
96      ( match op with
97        Umin -> "(- " ^ v11 ^ ")","(- " ^ v12 ^ ")", env, asn_map
98      | Not -> "(not " ^ v11 ^ ")","(not " ^ v12 ^ ")", env, asn_map
99      | x -> raise (Failure ("ERROR: Invalid Unary Operator ")) )
100     | Binop(e1,op,e2) ->
101      let v11,v12,_, _ = weval e1 env asn_map cc in let v21, v22, _, _ = weval e2 env asn_map cc
102         in let shift_v21 = to_int v21 in let shift_v22 = to_int v22
103      in (match op with
104          Add  -> "(("^v11^")" ^ " + " ^ "("^v21^"))","(("^v12^")" ^ " + " ^ "("^v22^"))", env,
asn_map
105        | Sub  -> "(("^v11^")" ^ " - " ^ "("^v21^"))","(("^v12^")" ^ " - " ^ "("^v22^"))" , env,
asn_map
106        | Mul  -> "(("^v11^")" ^ " * " ^ "("^v21^"))","(("^v12^")" ^ " * " ^ "("^v22^"))"  , env,
asn_map
107        | Lt   -> "(("^v11^")" ^ " < " ^ "("^v21^"))","(("^v12^")" ^ " < " ^ "("^v22^"))", env,
asn_map
108        | Gt   -> "(("^v11^")" ^ " > " ^ "("^v21^"))","(("^v12^")" ^ " > " ^ "("^v22^"))", env,
asn_map
109        | Lte  -> "(("^v11^")" ^ " <= " ^ "("^v21^"))","(("^v12^")" ^ " <= " ^ "("^v22^"))", env,
asn_map
110        | Gte  -> "(("^v11^")" ^ " >= " ^ "("^v21^"))","(("^v12^")" ^ " >= " ^ "("^v22^"))", env,
asn_map
111        | Eq   -> "(("^v11^")" ^ " = " ^ "("^v21^"))","(("^v12^")" ^ " = " ^ "("^v22^"))", env,
asn_map
112        | Neq  -> "(("^v11^")" ^ " /= " ^ "("^v21^"))","(("^v12^")" ^ " /= " ^ "("^v22^"))", env,
asn_map
113        | Or   -> "(("^v11^")" ^ " or " ^ "("^v21^"))","(("^v12^")" ^ " or " ^ "("^v22^"))", env,
asn_map
114        | And  -> "(("^v11^")" ^ " and " ^ "("^v21^"))","(("^v12^")" ^ " and " ^ "("^v22^"))", env,
asn_map
115        | Xor  -> "(("^v11^")" ^ " xor  " ^ "("^v21^"))","(("^v12^")" ^ " xor  " ^ "("^v22^"))",
env, asn_map
116        | Shl  -> "(to_stdlogicvector( to_bitvector("^v11^")" ^ " sll " ^ "("^shift_v21^")
))","(to_stdlogicvector( to_bitvector("^v12^")" ^ " sll " ^ "("^shift_v22^") ))", env, asn_map
117        | Shr  -> "(to_stdlogicvector( to_bitvector("^v11^")" ^ " srl " ^ "("^shift_v21^")
```

```
    ))","(to_stdlogicvector( to_bitvector("^v12^")" ^ " srl " ^ "("^shift_v22^") ))", env, asn_map
118        | x    -> raise (Failure ("ERROR: Invalid Binary Operator ")) )
119    | Basn(i, e1) -> let asn_map = update_asn (Basn(i,Id(i.name))) asn_map
120                 in let v1, v2, _, _ = weval e1 env asn_map cc
121                 in  let slv_v1 = num_to_slv v1 i.size
122                 in  let slv_v2 = num_to_slv v2 i.size
123                   in ("\t\t" ^ i.name ^ "_r" ^ (string_of_int (cc+1)) ^ " <= " ^ slv_v1 ^ ";\n" ),
124                      ("\t\t" ^ i.name ^ "_r" ^ (string_of_int (cc+1)) ^ " <= " ^ slv_v2 ^ ";\n" ),
env, asn_map
125    | Subasn(i, strt, stop, e1) -> let asn_map = update_asn (Subasn(i, strt, stop, Id(i.name)))
asn_map
126                 in let v1, v2, _, _ = weval e1 env asn_map cc
127                 in let slv_v1 = num_to_slv v1 ((abs (strt - stop)+1))
128                 in let slv_v2 = num_to_slv v2 ((abs (strt - stop)+1))
129                   in let range =
130                    if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
strt) ^ ")" else
131                         "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int stop) ^ ")"
132                 in ("\t\t" ^ i.name ^ "_r" ^ (string_of_int (cc+1)) ^ range ^ " <= " ^ slv_v1 ^ ";
\n" ),
133                    ("\t\t" ^ i.name ^ "_r" ^ (string_of_int (cc+1)) ^ range ^ " <= " ^ slv_v2 ^ ";
\n" ), env, asn_map
134    | Aasn(bs,sz,e1,e2) -> let v11,v12, _, asn_map = match e1 with
135                          Num(i) -> let am = update_asn (Aasn(bs,i,Id("constant"),Id("constant")))
asn_map
136                                        in (string_of_int i),(string_of_int i), env, am
137                        | Id(i) -> let bus_from_var var = let (bus, _,_,_,_) = var in bus
138                                   in (try let bs_i = bus_from_var (find_variable genv.scope i)
139                                        in let am = update_asn (Aasn(bs, bs_i.init, Id("constant"),
Id("constant"))) asn_map
140                                        in ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ "_r" ^
string_of_int (cc+1) ^ ")"),
141                                           ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ "_r" ^
string_of_int cc ^ ")"), env, am
142                                    with Error(_) ->  let am = update_asn (Aasn(bs,sz,e1,Id
(bs.name))) asn_map
143                                                    in let i1,i2, _, _ = weval e1 env am cc
144                                                    in ("ieee.std_logic_unsigned.conv_integer("
^ i1 ^ ")"),
145                                                       ("ieee.std_logic_unsigned.conv_integer("
^ i2 ^ ")"), env, am )
146                        | x -> let am = update_asn (Aasn(bs,sz,x,Id(bs.name))) asn_map
147                                in let i1,i2, _, _ = weval x env am cc
148                                in ("ieee.std_logic_unsigned.conv_integer(" ^ i1 ^ ")"),
149                                   ("ieee.std_logic_unsigned.conv_integer(" ^ i2 ^ ")"), env, am
150                     in let v21,v22, _, _ = weval e2 env asn_map cc
151                      in  let slv_v21 = num_to_slv v21 bs.size
152                      in  let slv_v22 = num_to_slv v22 bs.size
153                      in ("\t\t" ^ bs.name ^ "_r" ^ (string_of_int (cc+1)) ^ "(" ^ v11 ^ ") " ^ " <=
" ^ slv_v21 ^ ";\n" ),
154                         ("\t\t" ^ bs.name ^ "_r" ^ (string_of_int (cc+1)) ^ "(" ^ v12 ^ ") " ^ " <=
" ^ slv_v22 ^ ";\n" ), env, asn_map
155    | x ->  raise (Failure ("Illegal expression in the body of the While statement" ))
156
157 (*Evaluate conditional expressions*)
158 in let rec wcondeval e env asn_map cc= match e with
159     Num(i) -> (string_of_int i)^" /= 0 ",(string_of_int i)^" /= 0 ", env, asn_map
160     | Id(i) -> (i ^ "_r" ^ (string_of_int (cc+1)) ^ " /= 0 "),(i ^ "_r" ^ (string_of_int cc) ^ " /=
0 "), {sens_list = (i^"_r"^string_of_int cc)::env.sens_list;}, asn_map
161     | Barray(bs, _, e1) -> let v1,v2 = match e1 with
162                          Num(i) -> (string_of_int i),(string_of_int i)
163                        | x -> let i1,i2, _, _ = weval x env asn_map cc
164                                in ("ieee.std_logic_unsigned.conv_integer(" ^ i1 ^ ")"),
("ieee.std_logic_unsigned.conv_integer(" ^ i2 ^ ")")
165                 in (bs.name^"_r"^(string_of_int (cc+1))) ^ "(" ^ v1 ^ ")" ^ " /= 0 ", (bs.name^"_r"^
(string_of_int cc)) ^ "(" ^ v1 ^ ")" ^ " /= 0 ", env, asn_map
166     | Subbus(bs, strt, stop) -> let range =
167                    if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
strt) ^ ")" else
```

```
168                           "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int stop) ^ ")"
169                           in (bs.name ^ "_r" ^ (string_of_int (cc+1))) ^ range ^ " /= 0 ", (bs.name ^
     "_r" ^ (string_of_int cc)) ^ range ^ " /= 0 ", env, asn_map
170       | Unop(op,e1) ->
171       ( match op with
172         Umin -> let v1,v2,_, _ = weval e1 env asn_map cc in "(- " ^ v1 ^ ")"  ^ " /= 0 ", "(- " ^ v2
     ^ ")"  ^ " /= 0 ", env, asn_map
173       | Not -> let v1,v2,_ , _ = wcondeval e1 env asn_map cc in "(not " ^ v1 ^ ")", "(not " ^ v2 ^
     ")", env, asn_map
174       | x -> raise (Failure ("ERROR: Invalid Unary Operator ")) )
175       | Binop(e1,op,e2) ->
176        let v11,v12, _, _ = weval e1 env asn_map cc in let v21,v22,_ , _ = weval e2 env asn_map cc
177           in let shift_v21 = to_int v21 in let shift_v22 = to_int v22
178        in (match op with
179            Add  -> "(("^v11^")" ^ " + " ^ "("^v21^"))" ^ " /= 0 ","(("^v12^")" ^ " + " ^ "("^v22^"))"
     ^ " /= 0 ", env, asn_map
180          | Sub  -> "(("^v11^")" ^ " - " ^ "("^v21^"))" ^ " /= 0 ","(("^v12^")" ^ " - " ^ "("^v22^"))"
     ^ " /= 0 ", env, asn_map
181          | Mul  -> "(("^v11^")" ^ " * " ^ "("^v21^"))" ^ " /= 0 ","(("^v12^")" ^ " * " ^ "("^v22^"))"
     ^ " /= 0 ", env, asn_map
182          | Lt   -> "(("^v11^")" ^ " < " ^ "("^v21^"))","(("^v12^")" ^ " < " ^ "("^v22^"))", env,
     asn_map
183          | Gt   -> "(("^v11^")" ^ " > " ^ "("^v21^"))","(("^v12^")" ^ " > " ^ "("^v22^"))", env,
     asn_map
184          | Lte  -> "(("^v11^")" ^ " <= " ^ "("^v21^"))","(("^v12^")" ^ " <= " ^ "("^v22^"))", env,
     asn_map
185          | Gte  -> "(("^v11^")" ^ " >= " ^ "("^v21^"))","(("^v12^")" ^ " >= " ^ "("^v22^"))", env,
     asn_map
186          | Eq   -> "(("^v11^")" ^ " = " ^ "("^v21^"))","(("^v12^")" ^ " = " ^ "("^v22^"))", env,
     asn_map
187          | Neq  -> "(("^v11^")" ^ " /= " ^ "("^v21^"))","(("^v12^")" ^ " /= " ^ "("^v22^"))", env,
     asn_map
188          | Xor  -> "(("^v11^")" ^ " xor  " ^ "("^v21^"))" ^ " /= 0 ","(("^v12^")" ^ " xor  " ^
     "("^v22^"))" ^ " /= 0 ", env, asn_map
189          | Shl  -> "(to_stdlogicvector( to_bitvector("^v11^")" ^ " sll " ^ "("^shift_v21^") ))" ^ " /
     = 0 ","(to_stdlogicvector( to_bitvector("^v12^")" ^ " sll " ^ "("^shift_v22^") ))" ^ " /= 0 ", env,
     asn_map
190          | Shr  -> "(to_stdlogicvector( to_bitvector("^v11^")" ^ " srl " ^ "("^shift_v21^") ))" ^ " /
     = 0 ","(to_stdlogicvector( to_bitvector("^v12^")" ^ " srl " ^ "("^shift_v22^") ))" ^ " /= 0 ", env,
     asn_map
191          | Or   -> let v11,v12, _, _ = wcondeval e1 env asn_map cc in let v21,v22, _, _ = wcondeval
     e2 env asn_map cc
192                           in "(("^v11^")" ^ " or " ^ "("^v21^"))","(("^v12^")" ^ " or " ^
     "("^v22^"))", env, asn_map
193          | And  -> let v11,v12, _, _ = wcondeval e1 env asn_map cc in let v21,v22, _, _ = wcondeval
     e2 env asn_map cc
194                           in "(("^v11^")" ^ " and " ^ "("^v21^"))","(("^v12^")" ^ " and " ^
     "("^v22^"))", env, asn_map
195          | x    -> raise (Failure ("ERROR: Invalid Binary Operator ")) )
196      | x ->  raise (Failure ("Illegal conditional expression" ))
197
198
199    (* translate_wstmt *)
200    in let rec translate_wstmt (env,str1,str2,asn_map,cc) stmt =
201       (  match stmt with
202             Block(stmts)  -> List.fold_left translate_wstmt (env,str1,str2,asn_map,cc) (List.rev
     stmts)
203           | Expr(ex) -> let (e, ex_t, ex_s) = ex
204               in let s1,s2,_,asn_map = weval e env asn_map cc in (env, (str1 ^ s1), (str2 ^ s2),
     asn_map, cc)
205           | If(e,if_stmt,else_stmt) ->
206                           (*Check there is no POS*)
207                           let _ = (match if_stmt with
208                                       Block(sl) -> let chk_if_pos = (function
209                                                    Pos(_) -> raise (Error("Pos inside if
     statement"))
210                                                   | _ -> "" )
211                                              in let _ = (List.map chk_if_pos sl) in ""
212                                      | Pos(_) -> raise (Error("Pos inside if statement"))
213                                      | _ -> "" )
```

```
214                         in let _ = (match else_stmt with
215                                         Block(sl) -> let chk_if_pos = (function
216                                                         Pos(_) -> raise (Error("Pos inside if
    statement"))
217                                                       | _ -> "" )
218                                                 in let _ = (List.map chk_if_pos sl) in ""
219                                         | Pos(_) -> raise (Error("Pos inside if statement"))
220                                         | _ -> "" )
221                 in let s1,s2,_,_ = wcondeval e env asn_map cc
222             in let _,if_block1,if_block2,asn_map,_ = translate_wstmt (env,"","",asn_map,cc) if_stmt
223             in let _,else_block1,else_block2,asn_map,_ = translate_wstmt (env,"","",asn_map,cc)
    else_stmt
224             in (env, (str1^"\t\tif (" ^ s1 ^ ") then \n" ^ if_block1 ^ "\n\t\telse\n" ^ else_block1
    ^ "\t\tend if;\n"),
225                      (str2^"\t\tif (" ^ s2 ^ ") then \n" ^ if_block2 ^ "\n\t\telse\n" ^ else_block2
    ^ "\t\tend if;\n"), asn_map,cc)
226         | Switch ( e, c_list ) ->
227             ( match c_list with
228               [] -> env,"","",asn_map,cc
229              |hd::tl ->
230               let (e1, stmt) = hd
231                         (*Check there is no POS*)
232                         in let _ = (match stmt with
233                                         Block(sl) -> let chk_if_pos = (function
234                                                         Pos(_) -> raise (Error("Pos inside switch
    statement"))
235                                                       | _ -> "" )
236                                                 in let _ = (List.map chk_if_pos sl) in ""
237                                         | Pos(_) -> raise (Error("Pos inside switch statement"))
238                                         | _ -> "" )
239            in let s11,s12,_,_ = weval e env asn_map cc in let s21,s22,_,_ = weval e1 env asn_map
    cc
240            in let s31 = "\t\tif (" ^ s11 ^ " = " ^ s21 ^ ") then \n"
241            in let s32 = "\t\tif (" ^ s12 ^ " = " ^ s22 ^ ") then \n"
242            in let _,if_block1,if_block2,asn_map,_ = translate_wstmt (env,"","",asn_map,cc) stmt
243                    in let _,s51,s52,asn_map,_ = List.fold_left (translate_case (s11,s12))
    (env,"","",asn_map,cc) tl
244                    in (env, (str1^s31 ^ if_block1 ^ s51 ^ "\t\tend if;\n"),(str2^s32 ^ if_block2 ^
    s52 ^ "\t\tend if;\n"),asn_map,cc ) )
245         | x ->  raise (Failure ("Illegal statement within the body of the While statement" )) )
246     and translate_case (left1,left2) (env,s1,s2,asn_map,cc) (e,stmt) =
247                         (*Check there is no POS*)
248                         let _ = (match stmt with
249                                         Block(sl) -> let chk_if_pos = (function
250                                                         Pos(_) -> raise (Error("Pos inside switch
    statement"))
251                                                       | _ -> "" )
252                                                 in let _ = (List.map chk_if_pos sl) in ""
253                                         | Pos(_) -> raise (Error("Pos inside switch statement"))
254                                         | _ -> "" )
255       in ( match e with
256      (* SAST needs to check there is atmost one dafault and no duplicate
257      case expressions *)
258           Noexpr->   translate_wstmt (env,s1 ^ "\t\telse \n", s2 ^ "\t\telse \n",asn_map,cc) stmt
259         | x     ->    let right1,right2,_,asn_map = weval e env asn_map cc
260          in translate_wstmt (env,s1 ^ "\t\telsif (" ^ left1 ^ " = " ^ right1 ^ ") then \n",
261                                 s2 ^ "\t\telsif (" ^ left2 ^ " = " ^ right2 ^ ") then
    \n",asn_map,cc ) stmt
262          )
263 (* end of translate_wstmt *)
264
265
266 (*Translating While loop*)
267 in let rec build_while wstr str1 str2 asn_map prev_asn_map cc = function
268           [] -> wstr, str1, str2, asn_map, prev_asn_map, cc
269         | (Pos(en))::tl -> (let sen1, sen2, _, _ = wcondeval en {sens_list=[]} asn_map cc
270                         (*While condition: always check the output of the loop*)
271                        in let wcs1, wcs2,_,_ = wcondeval wcond {sens_list=[]} Im.empty cc
272                        in let (_,async) = get_asn curr_asn_map
273                        in let sync = get_nc_asn curr_asn_map asn_map
```

```
274                             in let (psync,_) = get_asn prev_asn_map
275                             in let (wsync,wasync) = get_asn asn_map
276                              (*No Asynchronous Variables can be assigned within a While loop or multiple
     assignments will occure!*)
277                              in let _ = match wasync with
278                                      [] -> ""
279                                     | _ -> raise (Error("Asynchronous variables assigned in the body of
     a While statement"))
280                              (*Even inside the While statement Variables can be assigned only once!*)
281                              in let chk_asn (asn : Sast.expr_detail) = try let _  = (List.find (fun a ->
     a = asn) psync) in
282                                                                                  raise (Error("Multiple
     variable assignment within a While statement"))
283                                                                                  with Not_found -> ""
284                              in let _ = List.map chk_asn wsync
285                              (*Find common sync assignments between curr_asn_map and while asn_map*)
286                              in let print_ccp1 (ap) = (function
287                              | Basn(x,_) -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ " <= " ^
     x.name ^ "_r" ^ (string_of_int cc) ^ ";\n"
288                              | Aasn(x,i,e1,_) -> (match e1 with (*Either e1 is a constant or the whole
     array is assigned!*)
289                                                        Id("constant") -> ap ^ x.name ^ "_r" ^
     (string_of_int (cc+1)) ^ "("
290                                                                                  ^ string_of_int i ^ ")" ^ " <= "
     ^ x.name ^ "_r"
291                                                                                  ^ (string_of_int cc) ^ "(" ^
     string_of_int i ^ ")" ^ ";\n"
292                                                        | e -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1))
     ^ " <= " ^ x.name ^ "_r"
293                                                                          ^ (string_of_int cc) ^ ";\n"          )
294                              | Subasn(x,strt,stop,_) -> let range =
295                                                  if strt < stop then "(" ^ (string_of_int stop) ^ " downto
     " ^ (string_of_int strt) ^ ")" else
296                                                                      "(" ^ (string_of_int strt) ^ " downto
     " ^ (string_of_int stop) ^ ")"
297                                      in ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ range ^ " <= " ^
     x.name ^ "_r" ^ (string_of_int cc) ^ range ^ ";\n"
298                              | x -> raise (Error("not an assignment"))       )
299                              in let print_reset (ap) = (function
300                              | Basn(x,_) -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ " <=
     ieee.std_logic_arith.conv_std_logic_vector("
301                                                  ^ string_of_int (x.init) ^ "," ^ string_of_int (x.size) ^
     ");\n"
302                              | Aasn(x,i,e1,_) -> (match e1 with (*Either e1 is a constant or the whole
     array is assigned!*)
303                                                        Id("constant") -> ap ^ x.name ^ "_r" ^
     (string_of_int (cc+1)) ^ "("
304                                                                                  ^ string_of_int i ^ ")" ^ " <=
     ieee.std_logic_arith.conv_std_logic_vector("
305                                                                                  ^ string_of_int (x.init) ^ "," ^
     string_of_int (x.size) ^ ");\n"
306                                                        | e -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1))
     ^ " <= (others => ieee.std_logic_arith.conv_std_logic_vector("
307                                                                                  ^ string_of_int (x.init) ^ "," ^
     string_of_int (x.size) ^ "));\n"                )
308                              | Subasn(x,strt,stop,_) -> let range =
309                                                  if strt < stop then "(" ^ (string_of_int stop) ^ " downto
     " ^ (string_of_int strt) ^ ")" else
310                                                                      "(" ^ (string_of_int strt) ^ " downto
     " ^ (string_of_int stop) ^ ")"
311                                      in ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ range ^ " <=
     ieee.std_logic_arith.conv_std_logic_vector("
312                                                  ^ string_of_int (x.init) ^ "," ^ string_of_int ((abs
     (strt-stop))+1) ^ ");\n"
313                              | x -> raise (Error("not an assignment"))       )
314
315                              in let nw = (List.fold_left print_ccp1 ("--Pos--\n") async) ^
     (List.fold_left print_ccp1 ("") sync)
316                              in let ywreset = (List.fold_left print_reset ("") wsync) ^ (List.fold_left
     print_reset ("") psync)
```

```
317                         in let ywpr = List.fold_left print_ccp1 ("") psync
318                         in let ywyr1 = str1
319                         in let ywyr2 = (List.fold_left print_ccp1 ("") wsync) (*use str2 for the
    other version*)
320
321                         in let seqp = "process(clk,rst)\nbegin\nif rst = '0' then\n"
322                         in let posedge = "elsif clk'event and clk = '1' then\n"
323                         in let swc_if = "if " ^ wcs1 ^ "then\n"
324                         in let sen1 = "if " ^ sen1 ^ " then\n"
325                         in let swc_else = "end if;\nelse\n" (*use "end if;\nelsif " ^ wcs2 ^ " then
    \n" for the other version*)
326                         in let sen2 = "if " ^ sen2 ^ " then\n"
327                         in let endp = "end if;\nend if;\nend if;\nend process;\n\n"
328                         in let wstr = wstr^
    (nw^seqp^ywreset^posedge^swc_if^sen1^ywpr^ywyr1^swc_else^sen2^ywpr^ywyr2^endp)
329                         in let str1 = "" in let str2 = ""
330                         in let prev_asn_map =
331                                 let up pam asn = update_asn asn pam
332                                 in List.fold_left up prev_asn_map wsync
333                         in let asn_map = Im.empty
334                         in let cc = cc+1
335                         in build_while wstr str1 str2 asn_map prev_asn_map cc tl )
336         | hd::tl -> let _,str1,str2,asn_map,cc = translate_wstmt ({sens_list=[]},str1,str2,
    asn_map ,cc) hd
337                 in build_while wstr str1 str2 asn_map prev_asn_map cc tl
338
339 in let wstr, str1, str2, asn_map, prev_asn_map, curr_cc = build_while "" "" "" Im.empty Im.empty
    curr_cc wblock
340
341 in let (psync,_) = get_asn prev_asn_map
342 in let (sync,_) = get_asn asn_map
343         in let tmp_asn_map =
344                 let up pam asn = update_asn asn pam
345                 in List.fold_left up curr_asn_map psync
346         in let curr_asn_map =
347                 let up pam asn = update_asn asn pam
348                 in List.fold_left up tmp_asn_map sync
349 in {sens_list=[]},wstr,curr_asn_map,curr_cc
350
351 (**************End of while loop processing*****************)
352
353
354 (* Evaluate expressions *)
355  in let rec eval e env asn_map cc= match e with
356     Num(i) -> string_of_int i, env, asn_map
357     | Id(i) -> (try let _ = (find_variable genv.scope i) (*no constants in the sensitivity list!*)
358                         in (i ^ "_r" ^ (string_of_int cc)), env, asn_map
359                 with (Error(_)) -> (i ^ "_r" ^ (string_of_int cc)), {sens_list =
    (i^"_r"^string_of_int cc)::env.sens_list;}, asn_map )
360     | Barray(bs, _, e1) -> let v1, env = match e1 with
361                           Num(i) -> (string_of_int i), env
362                         | x -> let i, env, _ = eval x env asn_map cc
363                                 in ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ ")"), env
364                 in (bs.name^"_r"^(string_of_int cc)) ^ "(" ^ v1 ^ ")", {sens_list =
    (bs.name^"_r"^string_of_int cc)::env.sens_list;}, asn_map
365                 (* Using "a" rather than "a(i)" in the sensitivity list, which is fine, because the
    list must be static *)
366     | Subbus(bs, strt, stop) -> let range =
367                    if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
    strt) ^ ")" else
368                         "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int stop) ^ ")"
369                 in (try let _ = (find_variable genv.scope bs.name)
370                         in (bs.name ^ "_r" ^ (string_of_int cc)) ^ range, env, asn_map
371                     with (Error(_)) -> (bs.name ^ "_r" ^ (string_of_int cc)) ^ range, {sens_list =
    (bs.name^"_r"^string_of_int cc)::env.sens_list;}, asn_map )
372     | Unop(op,e1) -> let v1, env, _ = eval e1 env asn_map cc in
373     ( match op with
374       Umin -> "(- " ^ v1 ^ ")"
375     | Not -> "(not " ^ v1 ^ ")"
376     | x -> raise (Failure ("ERROR: Invalid Unary Operator ")) ), env, asn_map
```

```
377      | Binop(e1,op,e2) ->
378       let v1, env, _ = eval e1 env asn_map cc in let v2, env, _ = eval e2 env asn_map cc
379          in let shift_v2 = to_int v2
380       in (match op with
381            Add  -> "(("^v1^")" ^ " + " ^ "("^v2^"))"
382          | Sub  -> "(("^v1^")" ^ " - " ^ "("^v2^"))"
383          | Mul  -> "(("^v1^")" ^ " * " ^ "("^v2^"))"
384          | Lt   -> "(("^v1^")" ^ " < " ^ "("^v2^"))"
385          | Gt   -> "(("^v1^")" ^ " > " ^ "("^v2^"))"
386          | Lte  -> "(("^v1^")" ^ " <= " ^ "("^v2^"))"
387          | Gte  -> "(("^v1^")" ^ " >= " ^ "("^v2^"))"
388          | Eq   -> "(("^v1^")" ^ " = " ^ "("^v2^"))"
389          | Neq  -> "(("^v1^")" ^ " /= " ^ "("^v2^"))"
390          | Or   -> "(("^v1^")" ^ " or " ^ "("^v2^"))"
391          | And  -> "(("^v1^")" ^ " and " ^ "("^v2^"))"
392          | Xor  -> "(("^v1^")" ^ " xor  " ^ "("^v2^"))"
393          | Shl  -> "(to_stdlogicvector( to_bitvector("^v1^")" ^ " sll " ^ "("^shift_v2^") ))"
394          | Shr  -> "(to_stdlogicvector( to_bitvector("^v1^")" ^ " srl " ^ "("^shift_v2^") ))"
395          | x    -> raise (Failure ("ERROR: Invalid Binary Operator ")) ), env, asn_map
396     | Basn(i, e1) -> let asn_map = update_asn (Basn(i,Id(i.name))) asn_map
397                 in let v1, env, _ = eval e1 env asn_map cc
398                 in  let slv_v1 = num_to_slv v1 i.size
399                   in ("\t\t" ^ i.name ^ "_r" ^ (string_of_int cc) ^ " <= " ^ slv_v1 ^ ";\n" ) ,
env, asn_map
400     | Subasn(i, strt, stop, e1) -> let asn_map = update_asn (Subasn(i, strt, stop, Id(i.name)))
asn_map
401                 in let v1, env, _ = eval e1 env asn_map cc
402                 in let slv_v1 = num_to_slv v1 ((abs (strt - stop)+1))
403                   in let range =
404                    if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
strt) ^ ")" else
405                         "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int stop) ^ ")"
406                 in ("\t\t" ^ i.name ^ "_r" ^ (string_of_int cc) ^ range ^ " <= " ^ slv_v1 ^ ";\n"
) , env, asn_map
407     | Aasn(bs,sz,e1,e2) -> let v1, env, asn_map = match e1 with
408                           Num(i) -> let am = update_asn (Aasn(bs,i,Id("constant"),Id("constant")))
asn_map
409                                         in (string_of_int i), env, am
410                         | Id(i) -> let bus_from_var var = let (bus, _,_,_,_) = var in bus
411                                    in (try let bs_i = bus_from_var (find_variable genv.scope i)
412                                         in let am = update_asn (Aasn(bs, bs_i.init, Id("constant"),
Id("constant"))) asn_map
413                                         in ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ "_r" ^
string_of_int cc ^ ")"), env, am
414                                     with Error(_) ->  let am = update_asn (Aasn(bs,sz,e1,Id
(bs.name))) asn_map
415                                                         in let i, env, _ = eval e1 env am cc
416                                                         in ("ieee.std_logic_unsigned.conv_integer("
^ i ^ ")"), env, am )
417                         | x -> let am = update_asn (Aasn(bs,sz,x,Id(bs.name))) asn_map
418                                 in let i, env, _ = eval x env am cc
419                                 in ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ ")"), env, am
420              in let v2, env, _ = eval e2 env asn_map cc
421                      in  let slv_v2 = num_to_slv v2 bs.size
422                      in ("\t\t" ^ bs.name ^ "_r" ^ (string_of_int cc) ^ "(" ^ v1 ^ ") " ^ " <= " ^
slv_v2 ^ ";\n" ), env, asn_map
423     | x ->  raise (Failure ("Expression not supported yet " ))
424
425
426 (*Evaluate conditional expressions*)
427 in let rec condeval e env asn_map cc= match e with
428     Num(i) -> (string_of_int i)^" /= 0 ", env, asn_map
429     | Id(i) -> (try let _ = (find_variable genv.scope i) (*no constants in the sensitivity list!*)
430                       in (i ^ "_r" ^ (string_of_int cc) ^ " /= 0 "), env, asn_map
431                 with (Error(_)) -> (i ^ "_r" ^ (string_of_int cc) ^ " /= 0 "), {sens_list =
(i^"_r"^string_of_int cc)::env.sens_list;}, asn_map )
432     | Barray(bs, _, e1) -> let v1, env = match e1 with
433                           Num(i) -> (string_of_int i), env
434                         | x -> let i, env, _ = eval x env asn_map cc
435                                 in ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ ")"), env
```

```
436                 in (bs.name^"_r"^(string_of_int cc)) ^ "(" ^ v1 ^ ")" ^ " /= 0 ", {sens_list =
(bs.name^"_r"^string_of_int cc)::env.sens_list;}, asn_map
437                 (* Using "a" rather than "a(i)" in the sensitivity list, which is fine, because the
list must be static *)
438     | Subbus(bs, strt, stop) -> let range =
439                    if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
strt) ^ ")" else
440                         "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int stop) ^ ")"
441                 in (try let _ = (find_variable genv.scope bs.name)
442                         in (bs.name ^ "_r" ^ (string_of_int cc)) ^ range ^ " /= 0 ", env, asn_map
443                   with (Error(_)) -> (bs.name ^ "_r" ^ (string_of_int cc)) ^ range ^ " /= 0 ",
{sens_list =  (bs.name^"_r"^string_of_int cc)::env.sens_list;}, asn_map )
444     | Unop(op,e1) ->
445     ( match op with
446       Umin -> let v1, env, _ = eval e1 env asn_map cc in "(- " ^ v1 ^ ")"  ^ " /= 0 "
447     | Not -> let v1, env, _ = condeval e1 env asn_map cc in "(not " ^ v1 ^ ")"
448     | x -> raise (Failure ("ERROR: Invalid Unary Operator ")) ), env, asn_map
449     | Binop(e1,op,e2) ->
450      let v1, env, _ = eval e1 env asn_map cc in let v2, env, _ = eval e2 env asn_map cc
451         in let shift_v2 = to_int v2
452      in (match op with
453           Add  -> "(("^v1^")" ^ " + " ^ "("^v2^"))" ^ " /= 0 "
454         | Sub  -> "(("^v1^")" ^ " - " ^ "("^v2^"))" ^ " /= 0 "
455         | Mul  -> "(("^v1^")" ^ " * " ^ "("^v2^"))" ^ " /= 0 "
456         | Lt   -> "(("^v1^")" ^ " < " ^ "("^v2^"))"
457         | Gt   -> "(("^v1^")" ^ " > " ^ "("^v2^"))"
458         | Lte  -> "(("^v1^")" ^ " <= " ^ "("^v2^"))"
459         | Gte  -> "(("^v1^")" ^ " >= " ^ "("^v2^"))"
460         | Eq   -> "(("^v1^")" ^ " = " ^ "("^v2^"))"
461         | Neq  -> "(("^v1^")" ^ " /= " ^ "("^v2^"))"
462         | Xor  -> "(("^v1^")" ^ " xor  " ^ "("^v2^"))" ^ " /= 0 "
463         | Shl  -> "(to_stdlogicvector( to_bitvector("^v1^")" ^ " sll " ^ "("^shift_v2^") ))" ^ " /=
0 "
464         | Shr  -> "(to_stdlogicvector( to_bitvector("^v1^")" ^ " srl " ^ "("^shift_v2^") ))" ^ " /=
0 "
465         | Or   -> let v1, env, _ = condeval e1 env asn_map cc in let v2, env, _ = condeval e2 env
asn_map cc
466                         in "(("^v1^")" ^ " or " ^ "("^v2^"))"
467         | And  -> let v1, env, _ = condeval e1 env asn_map cc in let v2, env, _ = condeval e2 env
asn_map cc
468                         in "(("^v1^")" ^ " and " ^ "("^v2^"))"
469         | x    -> raise (Failure ("ERROR: Invalid Binary Operator ")) ), env, asn_map
470    | x ->  raise (Failure ("Illegal conditional expression" ))
471
472
473  (* translate_Stmt *)
474  in let rec translate_stmt (env,str,asn_map,cc) stmt =
475     (  match stmt with
476          Block(stmts)  -> List.fold_left translate_stmt (env,str,asn_map,cc) (List.rev stmts)
477        | Expr(ex) -> let (e, ex_t, ex_s) = ex
478           in let s,env,asn_map = eval e env asn_map cc in (env, (str ^ s), asn_map, cc)
479        | If(e,if_stmt,else_stmt) ->
480                         (*Check there is no POS*)
481                         let _ = (match if_stmt with
482                                         Block(sl) -> let chk_if_pos = (function
483                                                          Pos(_) -> raise (Error("Pos inside if
statement"))
484                                                        | _ -> "" )
485                                                in let _ = (List.map chk_if_pos sl) in ""
486                                         | Pos(_) -> raise (Error("Pos inside if statement"))
487                                         | _ -> "" )
488                         in let _ = (match else_stmt with
489                                         Block(sl) -> let chk_if_pos = (function
490                                                          Pos(_) -> raise (Error("Pos inside if
statement"))
491                                                        | _ -> "" )
492                                                in let _ = (List.map chk_if_pos sl) in ""
493                                         | Pos(_) -> raise (Error("Pos inside if statement"))
494                                         | _ -> "" )
495                 in let s,env,_ = condeval e env asn_map cc
```

```
496             in let env,if_block,asn_map,_ = translate_stmt (env,"",asn_map,cc) if_stmt
497             in let env,else_block,asn_map,_ = translate_stmt (env,"",asn_map,cc) else_stmt
498             in (env, (str^"\t\tif (" ^ s ^ ") then \n" ^ if_block
499             (* the tabbing needs to be done programmatically, not manually.
500             I am assuming SAST will tell us the nesting depth *)
501             ^ "\n\t\telse\n" ^ else_block ^ "\t\tend if;\n"), asn_map,cc)
502         | Switch ( e, c_list ) ->
503             ( match c_list with
504               [] -> env,"",asn_map,cc
505              |hd::tl ->
506              (*let s,env,asn_map = eval e env asn_map
507            in *)let (e1, stmt) = hd
508                         (*Check there is no POS*)
509                         in let _ = (match stmt with
510                                         Block(sl) -> let chk_if_pos = (function
511                                                          Pos(_) -> raise (Error("Pos inside switch
    statement"))
512                                                        | _ -> "" )
513                                                 in let _ = (List.map chk_if_pos sl) in ""
514                                       | Pos(_) -> raise (Error("Pos inside switch statement"))
515                                       | _ -> "" )
516            in let s1,env,_ = eval e env asn_map cc in let s2,env,_ = eval e1 env asn_map cc
517            in let s3 = "\t\tif (" ^ s1 ^ " = " ^ s2 ^ ") then \n"
518            in let env,if_block,asn_map,_ = translate_stmt (env,"",asn_map,cc) stmt
519                    in let env,s5,asn_map,_ = List.fold_left (translate_case s1) (env,"",asn_map,cc)
    tl
520                    in (env, (str^s3 ^ if_block ^ s5 ^ "\t\tend if;\n"),asn_map,cc ) )
521         | While(e1,s1) -> let curr_fc,sl = (match s1 with
522                                         Block(sl) -> let inc_fc curr_fc = (function
523                                                         Pos(_) -> curr_fc + 1
524                                                        | _ -> curr_fc )
525                                                 in (List.fold_left inc_fc cc sl), sl
526                                       | _ -> raise (Error("While statement requires a block
    containing at least one POS and another statement")))
527                         in let rsl = List.rev sl
528                         in let rsl, curr_fc = match List.hd rsl with
529                                         Pos(en) -> rsl, curr_fc
530                                       | _ -> (print_endline "Warning: inferred Pos(1) at the end
    of the while loop"); (Pos(Num(1)))::rsl, curr_fc+1
531                         in let sl = List.rev rsl
532                         in translate_while e1 sl asn_map cc (curr_fc-1)
533         | Pos(en) -> let sen,_,_ = condeval en env asn_map cc
534                        in let (sync,async) = get_asn asn_map
535                         in let print_ccp1 (ap) = (function
536                           Basn(x,_) -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ " <= " ^
    x.name ^ "_r" ^ (string_of_int cc) ^ ";\n"
537                         | Aasn(x,i,e1,_) -> (match e1 with (*Either e1 is a constant or the whole
    array is assigned!*)
538                                                  Id("constant") -> ap ^ x.name ^ "_r" ^
    (string_of_int (cc+1)) ^ "("
539                                                                   ^ string_of_int i ^ ")" ^ " <= "
    ^ x.name ^ "_r"
540                                                                   ^ (string_of_int cc) ^ "(" ^
    string_of_int i ^ ")" ^ ";\n"
541                                                 | e -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1))
    ^ " <= " ^ x.name ^ "_r"
542                                                          ^ (string_of_int cc) ^ ";\n"          )
543                         | Subasn(x,strt,stop,_) -> let range =
544                                          if strt < stop then "(" ^ (string_of_int stop) ^ " downto
    " ^ (string_of_int strt) ^ ")" else
545                                                                "(" ^ (string_of_int strt) ^ " downto
    " ^ (string_of_int stop) ^ ")"
546                                 in ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ range ^ " <= " ^
    x.name ^ "_r" ^ (string_of_int cc) ^ range ^ ";\n"
547                         | x -> raise (Error("not an assignment"))       )
548                         in let print_reset (ap) = (function
549                           Basn(x,_) -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ " <=
    ieee.std_logic_arith.conv_std_logic_vector("
550                                          ^ string_of_int (x.init) ^ "," ^ string_of_int (x.size) ^
    ");\n"
```

```
551                            | Aasn(x,i,e1,_) -> (match e1 with (*Either e1 is a constant or the whole
    array is assigned!*)
552                                                  Id("constant") -> ap ^ x.name ^ "_r" ^
    (string_of_int (cc+1)) ^ "("
553                                                                    ^ string_of_int i ^ ")" ^ " <=
    ieee.std_logic_arith.conv_std_logic_vector("
554                                                                    ^ string_of_int (x.init) ^ "," ^
    string_of_int (x.size) ^ ");\n"
555                                                  | e -> ap ^ x.name ^ "_r" ^ (string_of_int (cc+1))
    ^ " <= (others => ieee.std_logic_arith.conv_std_logic_vector("
556                                                                    ^ string_of_int (x.init) ^ "," ^
    string_of_int (x.size) ^ "));\n"                  )
557                            | Subasn(x,strt,stop,_) -> let range =
558                                           if strt < stop then "(" ^ (string_of_int stop) ^ " downto
    " ^ (string_of_int strt) ^ ")" else
559                                                               "(" ^ (string_of_int strt) ^ " downto
    " ^ (string_of_int stop) ^ ")"
560                                    in ap ^ x.name ^ "_r" ^ (string_of_int (cc+1)) ^ range ^ " <=
    ieee.std_logic_arith.conv_std_logic_vector("
561                                             ^ string_of_int (x.init) ^ "," ^ string_of_int ((abs
    (strt-stop))+1) ^ ");\n"
562                            | x -> raise (Error("not an assignment"))       )
563
564                            in let nr = List.fold_left print_ccp1 ("--Pos--\n") async
565                            in let reset = List.fold_left print_reset ("") sync
566                            in let yr = List.fold_left print_ccp1 ("") sync
567                            in let seqp = "process(clk,rst)\nbegin\nif rst = '0' then\n"
568                            in let posedge = "elsif clk'event and clk = '1' then\n"
569                            in let sen = "if " ^ sen ^ " then\n"
570                            in let endp = "end if;\nend if;\nend process;\n\n"
571                            in env,(nr^seqp^reset^posedge^sen^yr^endp),asn_map,(cc+1)
572
573            | Call(fdecl, out_list, in_list ) ->
574               (* start of f *)
575               let f (s,l,am) b =
576                    let bus_from_var var = let (bus, _,_,_,_) = var in bus
577                    (*using the field "size" in Barray(_,size,_) to identify which bus in the vector is
    assigned*)
578                    in let actual_barray am bs =  function
579                             Num(i) -> let am = update_asn (Aasn(bs, i, Id("constant"), Id
    ("constant"))) am
580                                             in (string_of_int i), am
581                            | Id(i) -> (try let bs_i = bus_from_var (find_variable genv.scope i)
582                                             in let am = update_asn (Aasn(bs, bs_i.init, Id("constant"),
    Id("constant"))) am
583                                             in ("ieee.std_logic_unsigned.conv_integer(" ^ i ^ "_r" ^
    (string_of_int cc) ^ ")"), am
584                                       with Error(_) -> raise (Failure("Function Call to " ^ fdecl.fid
    ^ ": actual "^ bs.name ^ " is not static"))  )
585                            | x -> raise (Failure("Function Call to " ^ fdecl.fid ^ ": illegal actual
    assignment"))
586                in
587                let s1, asn_map = (match (List.hd l) with
588                 Num(v) -> let s = num_to_slv (string_of_int v) b.size in s,am
589               |  Id(i) -> let bs_i = bus_from_var (find_variable cloc.scope i)
590                            in let am = update_asn (Basn(bs_i, Id("port map"))) am (*I don't care about
    the expr_detail in the assignment*)
591                            in i ^ "_r" ^ (string_of_int cc), am
592               | Barray(bs, _, e1) -> let v1,am = actual_barray am bs e1
593                                    in bs.name ^ "_r" ^ (string_of_int cc) ^ "(" ^ v1 ^ ")", am
594               | Subbus(bs, strt, stop) -> let range =
595                         if strt < stop then "(" ^ (string_of_int stop) ^ " downto " ^ (string_of_int
    strt) ^ ")" else
596                            "(" ^ (string_of_int strt) ^ " downto " ^ (string_of_int stop) ^ ")"
597                            in let am = update_asn (Subasn(bs, strt, stop, Id("port map"))) am
598                            in bs.name ^ "_r" ^ (string_of_int cc) ^ range, am
599               | x ->  raise (Failure ("Function Call to " ^ fdecl.fid ^ ": In/Output port mapping must
    use pre-existing variables " ))   )
600               in  s^",\n\t\t"^b.name^" => " ^ s1 , (List.tl l) , asn_map   (* end of f *)
601
```

```
602             (* When a function uses the same component multiple times, it needs to use unique labels
    to describe the
603             separate instantiations. One way to do this is to append a string that is a function of
    the head of the
604             output_list. The output_list is guranteed to be non-empty, SAST should also gurantee
    that the same output
605             variable does not get used in two different calls as outputs. *)
606             in let array_label = function
607                  Num(i) -> string_of_int i
608                | Id(i)  -> i
609                | x -> raise (Failure("Function Call to " ^ fdecl.fid ^ ": illegal actual
    assignment"))
610             in let label = match (List.hd out_list) with
611                Id(i) -> i
612              | Barray(bs, _, e1) ->  (bs.name) ^ (array_label e1)
613              | Subbus(bs, strt, stop) -> bs.name ^ "_" ^ (string_of_int strt) ^ "_" ^ (string_of_int
    stop)
614              | x->  raise (Failure ("In/Output port mapping must use pre-existing variables " ))
615             in  let s = str ^ fdecl.fid ^ "_" ^ label ^ " : " ^ fdecl.fid ^ " port map (\n\t\tclk =>
    clk,\n \t\trst => rst"
616
617              in let s,_,_ = List.fold_left f (s,(List.rev in_list),asn_map) fdecl.pin
618              in let s,_,asn_map = List.fold_left f (s,(List.rev out_list),asn_map) fdecl.pout
619              in ({sens_list=env.sens_list;}, s ^ ");\n\n",asn_map,cc) )
620     and translate_case left (env,s,asn_map,cc) (e,stmt) =
621                         (*Check there is no POS*)
622                         let _ = (match stmt with
623                                         Block(sl) -> let chk_if_pos = (function
624                                                          Pos(_) -> raise (Error("Pos inside switch
    statement"))
625                                                        | _ -> "" )
626                                                 in let _ = (List.map chk_if_pos sl) in ""
627                                       | Pos(_) -> raise (Error("Pos inside switch statement"))
628                                       | _ -> "" )
629       in ( match e with
630      (* SAST needs to check there is atmost one dafault and no duplicate
631      case expressions *)
632           Noexpr->   translate_stmt (env,s ^ "\t\telse \n",asn_map,cc) stmt
633         | x     ->    let right,env,asn_map = eval e env asn_map cc
634          in translate_stmt (env,s ^ "\t\telsif (" ^ left ^ " = " ^ right ^ ") then \n",asn_map,cc )
    stmt
635          )
636 (* end of translate_stmt *)
637
638
639     in let print_process prev (env,s) =
640      let l = uniq env.sens_list in
641      ( match l with   [] -> prev ^ s (* Don't make this a process if nothing in the sensitivity
    list, affects Call() and consts *)
642                    | x  -> let ss = delim_sprt ", " l
643                        in prev ^ "\n\tprocess (" ^ ss ^ ")\n\tbegin\n" ^ s ^ "\n\tend process;\n
    \n" )
644
645     in let body cobj asn_map=
646     let empty_env = {sens_list=[];}
647                 in let rec hsa l asn_map cc= function
648                    [] -> (List.rev l), asn_map, cc
649                  | hd::tl -> let (ts_env, ts_str,new_asn_map, new_cc) = (translate_stmt
    (empty_env,"",asn_map,cc) hd)
650                               in let new_l = (ts_env,ts_str)::l
651                                 in hsa new_l new_asn_map new_cc tl
652                 in let (stmt_attr, full_asn_map, fc) = hsa [] asn_map 0 cobj.fbod
653 (*un-comment the three lines below to print out the list of assigned variables*)
654         (*in let _ = print_endline ("function "^cname^":")
655         in let _ = print_endline ("final clock :"^(string_of_int fc))
656         in let _ = print_asn_map full_asn_map*)
657         in let s = List.fold_left print_process "" stmt_attr
658         in s, fc
659
660    in let arch cname cobj = (*arch *)
```

```
661      (* Add input ports to assignment map *)
662        let pin_asn = List.map (fun b -> Basn(b,Id(b.name))) cobj.pin
663          in let asn_map =
664            let rec ha0 asn0 = function
665               [] -> asn0;
666             | hd::tl -> let new_asn0 = update_asn hd asn0
667                          in ha0 new_asn0 tl
668             in ha0 Im.empty pin_asn
669
670  (* body takes Function objetc, assignment map at clock 0 and returns the body string and the final
clock*)
671     in let behavior, fc = body cobj asn_map
672  (* need to print out the locals before begin *)
673
674      (* print out component list *)
675      in let comp_decl s fdecl =
676              let s1 = port_gen fdecl.fid fdecl
677               in s ^ "component " ^ fdecl.fid ^  "\nport (\n" ^
678                  "\tclk : in std_logic;\n\trst : in std_logic;\n" ^ s1 ^ ");\nend component;\n\n"
679          (* if same component is used twice, we just print them once, hence the call to uniq_calls
*)
680      in let cl_s = List.fold_left comp_decl "" (uniq_calls cobj.fcalls)
681
682
683          in let rec print_bus bs ss = function
684             0 -> "signal " ^ bs.name ^ "_r0" ^ ss ^ " : std_logic_vector(" ^ (string_of_int
(bs.size-1))
685                  ^ " downto 0) := ieee.std_logic_arith.conv_std_logic_vector(" ^ string_of_int
(bs.init)
686                  ^ "," ^ string_of_int (bs.size) ^ ");\n"
687           | x -> print_bus bs (ss ^ ", " ^ bs.name ^ "_r" ^ string_of_int x) (x-1)
688          in let rec print_const bs ss = function
689             0 -> "constant " ^ bs.name ^ "_r0" ^ ss ^ " : std_logic_vector(" ^ (string_of_int
(bs.size-1))
690                  ^ " downto 0) := ieee.std_logic_arith.conv_std_logic_vector(" ^ string_of_int
(bs.init)
691                  ^ "," ^ string_of_int (bs.size) ^ ");\n"
692           | x -> print_const bs (ss ^ ", " ^ bs.name ^ "_r" ^ string_of_int x) (x-1)
693          in let rec print_array bs ss = function
694             0 -> "signal " ^ bs.name ^ "_r0" ^ ss ^ " : " ^ bs.name
695                  ^ "_type := (others => ieee.std_logic_arith.conv_std_logic_vector("
696                  ^ string_of_int (bs.init) ^ "," ^ string_of_int (bs.size) ^ "));\n"
697           | x -> print_array bs (ss ^ ", " ^ bs.name ^ "_r" ^ string_of_int x) (x-1)
698
699          in let print_signals ss var =
700            let (bs, sz, tp, _, _) = var
701            in let ps = (match tp with
702               Bus -> ss ^ (print_bus bs "" fc)
703             | Const -> ss ^ (print_const bs "" fc)
704                  (*!!PREVENT THE USER TO CALL AN ARRAY <ID>_TYPE. Maybe we want to remove '_' from
ID regular expression*)
705             | Array -> let s_type = "type " ^ bs.name ^ "_type is array (0 to " ^ string_of_int
(sz-1) ^ ") of std_logic_vector("
706                           ^ string_of_int (bs.size-1) ^ " downto 0);\n"
707                          in ss ^ s_type ^ (print_array bs "" fc)
708             | x -> raise (Failure("There's something wrong with the type symbol table!"))
709                  )
710                  in ps
711
712          in let const_list = (genv.scope).variables
713          in let bus_list = ((cobj.floc).scope).variables
714           in let c_sgnls = List.fold_left print_signals "" const_list
715           in let b_sgnls = List.fold_left print_signals "" bus_list
716           in let sgnls = c_sgnls^b_sgnls
717
718          in let print_inasn ss ibus = ss ^ ibus.name ^ "_r0 <= " ^ ibus.name ^ ";\n"
719          in let print_outasn ss obus = ss ^ obus.name ^ " <= " ^ obus.name ^ "_r" ^ (string_of_int
fc) ^ ";\n"
720          in let inasn = List.fold_left print_inasn "" (cobj.pin)
721          in let outasn = List.fold_left print_outasn "" (cobj.pout)
```

```
722
723       in "architecture e_" ^ cname ^ " of  " ^ cname ^ " is \n\n" ^ cl_s ^"\n\n"^ sgnls ^"\n\nbegin\n"
724         ^ inasn ^ outasn ^ "\n" ^ behavior
725         ^ "\n\nend e_" ^ cname ^ ";\n\n"
726
727     in let s = libraries ^ (entity cname cobj) ^ (arch cname cobj)
728     in CompMap.add cname s components
729     in let components = StringMap.fold create_component ftable CompMap.empty
730     in components
731
732   let print_programs outfile components =
733     let s = CompMap.fold ( fun k d s -> s ^ d ) components ""
734       in let out_channel = open_out outfile
735        in output_string out_channel s
736
737
738   let _ =
739   let usage = "Usage : ehdl [-o outfile] infile1 infile2 ...\n" in
740   let _ = if Array.length Sys.argv < 2 then raise (ErrorS ("input files not provided!" ^  usage) ) in
741   let j = if Sys.argv.(1) = "-o" then 3 else 1 in
742   let outfile =
743       (if Sys.argv.(1) = "-o" then
744           if ( Array.length Sys.argv < 4 ) then raise (ErrorS ("-o should follow an outfile and one
      or more infiles! \n" ^  usage ^ "\n")  )
745           else Sys.argv.(2)
746        else "main.vhd") in
747   let temp_file = "temp.txt" in
748   let temp_out = open_out temp_file in
749   let _ =
750   for i = j to Array.length Sys.argv - 1 do
751       let in_channel =
752       try
753            open_in Sys.argv.(i)
754       with e -> raise(ErrorS ("Failed to open " ^ Sys.argv.(i)))
755       in
756       let rec readfile infile =
757           try let line = input_line infile in line ^ "\n" ^ (readfile infile)
758           with End_of_file -> "" in
759           output_string temp_out ((readfile in_channel) ^ "\n\n")
760   done in
761   let _ = close_out temp_out in
762   let in_channel = open_in temp_file in
763   let lexbuf = Lexing.from_channel in_channel in
764   let program = Parser.program Scanner.token lexbuf in
765   let _  = print_programs outfile ( translate (prog (fst program , snd program) ) ) in
766   Printf.printf "Find the output in %s ...\n" outfile
```