a binary manipulation language

*blooRTLs

Apurv Gaurav (ag3596)
Peter H Burrows (phb2114)
Pinhong He (ph2482)
Zhibo Wan (zw2327)

- Motivation, Overview, and Tutorials
- Introduction of the blooRTLs
  - Language Features
- Project Architecture and keywords
  - Scanner, Parser, AST, VHDL
- Test Suites
- Summary and Lessons Learned
- Demo!

- An RTL description language geared towards catalyzing the development, simulation, and synthesis of RTL specs
- “Object-Oriented” - but NOT in the traditional sense
- “Reasonably” fast clock frequency assumed (>MHz)
- Compiles down to Sequential VHDL

* Overview: Behavioral Language for Object-Oriented RTL Specs

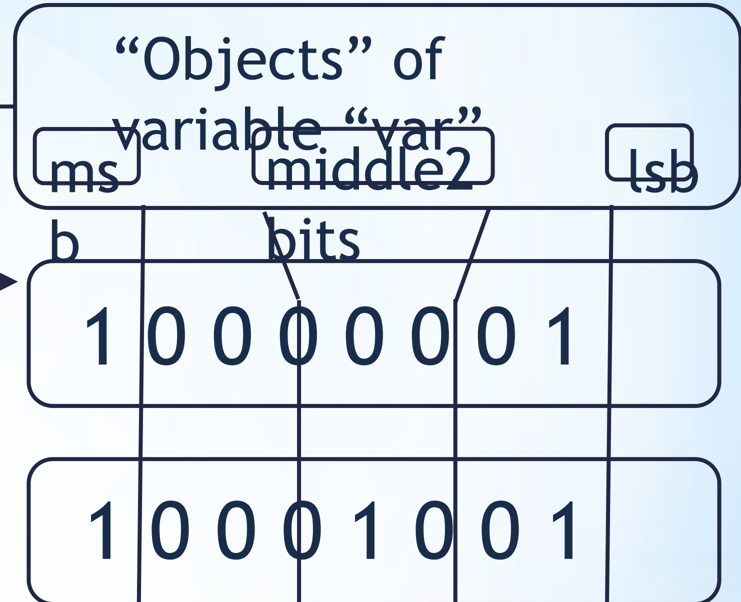
blooRTLs Tutorial

```
BINMAP var {
        msb  :=  [7];
        lsb  :=  [0];
        middle2bits:=  [4][3];
}

var := 10000001;

var.middle2bits := var.msb * var.lsb;

PRINT var middle2bits;
PRINT var;
```




```
peter@MacDonald:~/Desktop/blooRTLs_Precompiler$ ./blooRTLs<in.txt

======================= blooRTLs ==========================>

var.middle2bits =
        Integer Representation:         1
        Binary Indices:                 [4;3]
        Binary Representation:          [0;1]



var. =
        Integer Representation:         137
        Binary Indices:                 [7;6;5;4;3;2;1;0]
        Binary Representation:          [1;0;0;0;1;0;0;1]
```

Compiler Considerations for VHDL: The VHDL Libraries

NOT STANDARDIZED

```
library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
```

IEEE STANDARDIZED

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

- *Early 1990s* → **Synopsys** developed the arithmetic library with a user-friendly VHDL arithmetic syntax and packaged it into the IEEE library

- *Late 1990s* → **IEEE** developed and standardized the numeric library due to unexpected behavior across various toolkits that used the arithmetic library

Compiler Considerations for VHDL: The VHDL Libraries

NOT STANDARDIZED

```
library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
```

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

- Tradeoff: The NUMERIC library is **MORE RELIABLE** for simulation and synthesis, however it is much **QUIRKIER !**
  - It does **NOT** raise an error for overflow/underflow
  - It does **NOT** permit arithmetic for vectors of varying lengths; however, there is a clever work-around

Compiler Considerations for VHDL: The Sequential Framework

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

entity moore is
  port (
    Clock       : in STD_LOGIC;
    Resetn      : in STD_LOGIC;

    -- declare all inputs/outputs & vector length
    -- Arbitrary examples to illustrate syntax:
    input0      : in std_logic_vector (7 downto 0);
    input1      : in std_logic;          -- 1 bit
    output0 : out std_logic_vector(2 downto 0));

end moore;


architecture rtl of moore is
  type State_type is (A,B,C); --Declare RTL states
  signal state: State_type;

  -- declare all signals & vector lengths
  -- Arbitrary examples to illustrate syntax:

  signal signal0 : std_logic_vector (5 downto 0);
  signal signal1 : std_logic_vector (2 downto 0) := "000";
  signal signal2: std_logic := '1';
```

The Main Logic:

```
begin
  process (Resetn, Clock)
  begin
    if Resetn = '0' then
      state <= A;
    elseif (Clock'EVENT AND Clock = '1') then
      case state is
        when A =>
          if ( ... ) then
            -- some statements/code...
            state <= A;
          else
            state <= B;
          end if;

        when B =>
            -- some statements/code..

        when C =>
            -- some statements/code..
      end case;
    end if;
  end process;

-- push the desired logic to the output port. Arbitrary example
output0 <= "001" when state = C else "000";

end rtl;
```

blooRTLs Compiled

```
BINMAP var {
        lsb := [0];
        nibble1 := [7][6][5][4];
        nibble0 := [3][2][1][0];
}

var := 00100001;

var.nibble0 := var.nibble1 * var.lsb;

PRINT var;
PRINT var nibble0;
PRINT var nibble1;
PRINT var lsb;
```

```
architecture rtl of example is
  -- signals declared here
  signal var          : std_logic_vector (7 downto 0);
  signal nibble0_tmp  : std_logic_vector (7 downto 0);
  signal one3         : std_logic_vector (3 downto 0) := "0001";

begin
  process (Resetn, Clock)
  begin
    if Resetn = '0' then
      state <= A;
    elseif (Clock'EVENT AND Clock = '1') then
      case state is

        when A =>
          nibble0_tmp <=
            std_logic_vector(
                 one3_tmp                                          --0001
               * ( var(7) & var(6) & var(5) & var(4) )            --nibble1
               * ("0" & "0" & "0" & var(0)) );                     --lsb
            state <= B;

         when B =>
            var(3) <= nibble0_tmp(3);
            var(2) <= nibble0_tmp(2);
            var(1) <= nibble0_tmp(1);
            var(0) <= nibble0_tmp(0);
            state <= A;

         when C =>
            -- some statements/code..
        end case;
     end if;
  end process;

-- push the desired logic to the output port.
output0 <= var;

end rtl;
```

2 clock cycles

Concat operator

The Bottom Line : the blooRTLs compiler performs VHDL *"length inferencing"* for you

Precompiler

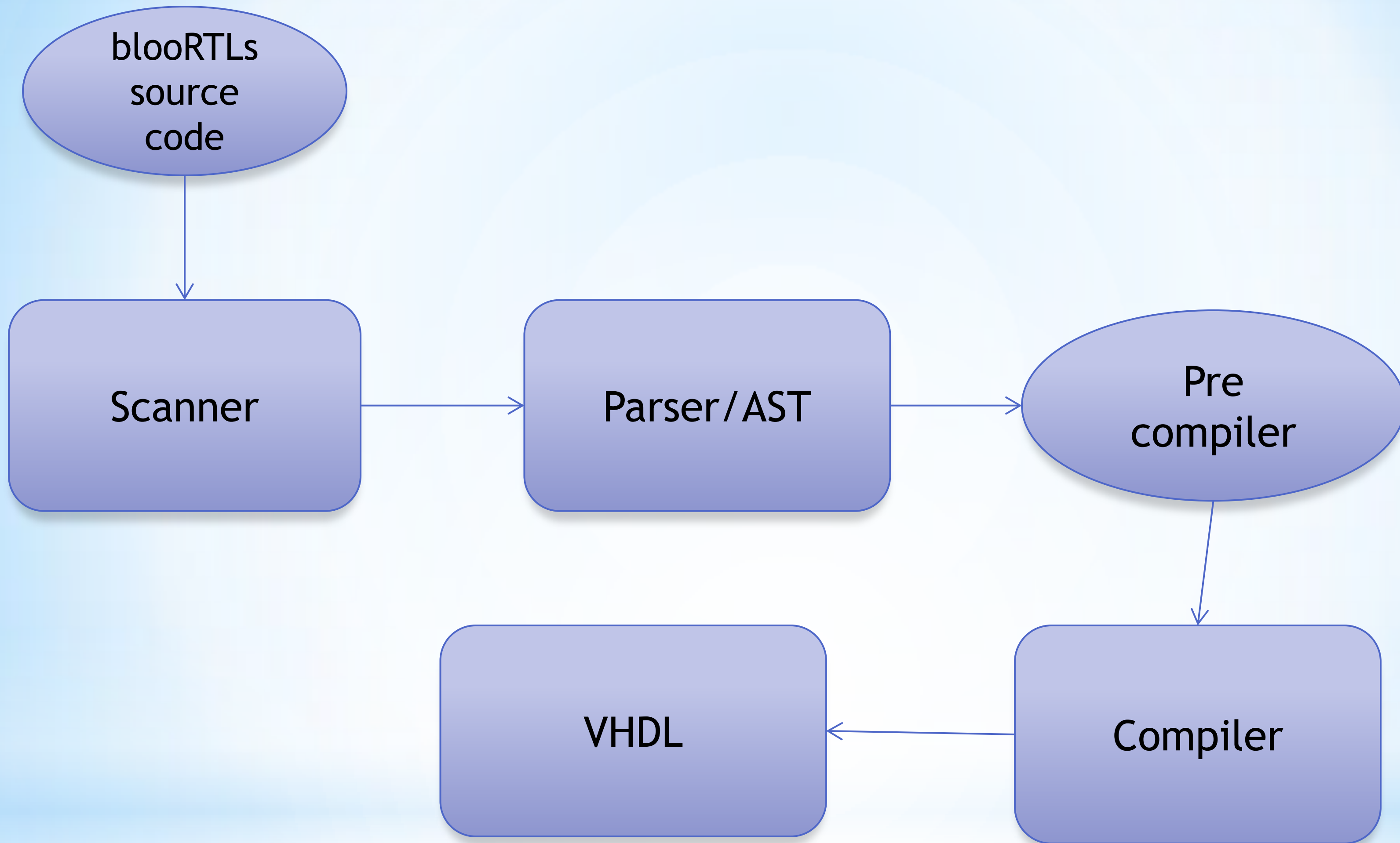
- Before compiling, the blooRTLs source code **MUST** be precompiled in order to:
  - **Cache the bit vector indices given by the BINMAP**
  - **Check for arithmetic over/underflows**
- In *Ocaml*, a map module was implemented to cache/log the values and indices of variables and objects...

Precompiler: Ocaml environment

It's a Map of Maps!

Keys (Variables)	Values (Maps)	
	Keys (Objects)	**Values (Tuples of int*int list * int list))**
"var" →	"" →	(137, [7;6;5;4;3;2;1;0], [1;0;0;0;1;0;0;1],0)
	"msb" →	(1, [7], [1], 0)
	"lsb" →	(1, [0], [1], 0)
	"middle2bits" →	(1, [4;3], [0;1], 0)

```
# let binmap = mapUpdate "var" "" ( 137, [7;6;5;4;3;2;1;0], [1;0;0;0;1;0;0;1],0 ) binmap;;
val binmap : (int * int list * int list * int) NameMap.t NameMap.t = <abstr>
# NameMap.find "" (NameMap.find "var" binmap);;
- : int * int list * int list * int =
(137, [7; 6; 5; 4; 3; 2; 1; 0], [1; 0; 0; 0; 1; 0; 0; 1], 0)
#
```



*Project Architecture

Program Keywords

- Variable Declaration var1
- Assign value for variables :=
- Basic operations: + - * =
- Binary shifting << >>
- BINMAP
- IF-THEN-ELSE, REPEAT-UNTIL
- PRINT

```
rule token = parse
   [' ' '\t' '\r' '\n'] { token lexbuf }
|  '+'  { PLUS }
|  '-'  { MINUS }
|  '*'  { TIMES }
|  ">>"  { SHIFTR }
|  "<<"  { SHIFTL }
|  ":="  { ASSIGN }
|  "="  { EQU }
|  ';'  { SEMI }
|  '('  { LP }
|  ')'  { RP }
|  '{'  { LCB }
|  '}'  { RCB }
|  '.'  { PERIOD }
|  ['0' - '9']+'d' as decimal { DEC (decimal) }
|  ['0' - '1']+ as bits { BITS (bits) }
|  ['a'-'z']['a'-'z' '0'-'9']+ as lxm { ID(lxm) }
|  "PRINT" { PRINT }
|  "INPUT" { INPUT }
|  "REPEAT" { REPEAT }
|  "UNTIL"  { UNTIL }
|  "IF"           { IF }
|  "THEN"         { THEN }
|  "ELSE"         { ELSE }
|  "BINMAP" { BINMAP }
|  '['['0'-'9']']' as ind
          { IND(int_of_char ind.[1] - 48) }
|  '['['0'-'9']['0'-'9']']' as ind
          { IND((10*(int_of_char ind.[1] - 48))
                   +((int_of_char ind.[2] - 48)) ) }
|  '['['0'-'9']['0'-'9']['0'-'9']']' as ind
          { IND((100*(int_of_char ind.[1] - 48))
                 +(10*(int_of_char ind.[2] - 48))
                      +((int_of_char ind.[3] - 48)) ) }
|  '['['0'-'9']['0'-'9']['0'-'9']['0'-'9']']' as ind
          { IND((1000*(int_of_char ind.[1] - 48))
                 +(100*(int_of_char ind.[2] - 48))
                  +(10*(int_of_char ind.[3] - 48))
                       +((int_of_char ind.[4] - 48)) ) }

|  eof { EOF }
```

*Scanner

```
statement:
  BINMAP ID LCB objdecl RCB                    { Binmap ($2,$4) }
| expr                                                { Expr ($1) }
| statement statement                          { Stmtseq ($1, $2) }
| PRINT ID ID SEMI                                 { Print ($2,$3) }
| PRINT ID SEMI                                    { Printvar ($2) }
| IF LP expr EQU expr RP THEN
        LP statement RP ELSE
        LP statement RP                            { Ifthen($3,$5,$9,$13) }
| REPEAT LP statement RP
  UNTIL LP expr EQU expr RP                    { Repeat($3,$7,$9) }


objdecl:
  ID ASSIGN objdecl SEMI                       { Objmap($1,$3)         }
| ID ASSIGN objdecl SEMI objdecl       { Objdeclseq($1,$3,$5) }
| IND                                                 { Indices($1)           }
| IND objdecl                                      { Indseq($1,$2)         }

expr:
|  BITS                                               { Bits($1) }
| DEC                                                 { Lit($1) }
| ID ASSIGN expr SEMI                          { AsnRoot($1,$3) }
| ID PERIOD ID ASSIGN expr SEMI          { AsnObj($1,$3,$5) }
| ID                                                  { IdenRoot($1) }
| ID PERIOD ID                                     { IdenObj($1,$3) }
| expr PLUS   expr                                 { Binop($1, Add, $3) }
| expr MINUS  expr                                 { Binop($1, Sub, $3) }
| expr TIMES  expr                                 { Binop($1, Mul, $3) }
| expr SHIFTR  expr                            { Binop($1, Shiftr, $3) }
| expr SHIFTL  expr                                { Binop($1, Shiftl, $3) }
| expr expr                                           { Exprseq ($1,$2) }
```

*Parser

```
type operator = Add | Sub | Mul | Shiftr | Shiftl

type expr =
    Lit of string
  | Bits of string (* binary string *)
  | AsnRoot of string * expr
  | AsnObj of string * string * expr
  | IdenRoot of string
  | IdenObj of string * string
  | Binop of expr * operator * expr
  | Exprseq of expr * expr


type objdecl =
        Objdeclseq of string * objdecl * objdecl
  | Objmap of string * objdecl
  | Indices of int
  |     Indseq of int * objdecl


type statement  =
    Binmap of string * objdecl
  | Expr of expr
  | Stmtseq of statement * statement
  | Print of string * string
  | Printvar of string
  | Ifthen of expr * expr * statement * statement
  | Repeat of statement * expr * expr
```

*Our Featured Test Case

* Using blooRTLs bit-mapping feature on sequential data, we can encode important sequential data, such as DNA, and be able to track genes
* In addition, DNA encoding can be optimized to use less space and digits
* Original Encoding:
  * A = 00, C = 01, T = 10, G = 11
* Huffman Encoding:
  * A = 0, T = 10, C = 101, G = 110

*Nucleotide Frequency

- We will take a DNA sequence that has been encoded into binary numbers and count how many of each nucleotide there are.
- Features Tested:
  - BINMAP, If-Then-Else, Repeat-Until, PRINT, bit manipulation, (Switch)

Output:

14

6

8

7

```
BINMAP var1 {
 nucleotide := [1][0];
}
var1 :=
10001100000000010001001101111010111001101001010000001110110001011000000;
adenosine := 0d; cytosine := 0d; thymine := 0d; guanine := 0d;
REPEAT (
IF (var1.nucleotide = 00)
THEN ( adenosine := adenosine + 1d;
    var1 >> 2d; )
IF (var1.nucleotide = 01)
THEN (     cytosine := cytosine + 1d;
    var1 >> 2d;)
IF (var1.nucleotide = 10)
THEN (     thymine := thymine + 1d;
    var1 >> 2d;)
ELSE
    ( guanine := guanine + 1d;
    var1 >> 2d;)
)
UNTIL (var1 = 0d)
PRINT adenosine; PRINT cytosine; PRINT thymine; PRINT guanine;
```

Huffman Algorithm

* Based on the nucleotide frequencies, we will re-encode the DNA code using the more efficient Huffman Algorithm
* Allots less bits to high freq info, more bits for low freq info
* Features Tested:
  * BINMAP, If-Then-Else, Repeat-Until, PRINT, bit manipultion, (Switch)

```
BINMAP oldseq {
 nucleotide := [1][0];
}
oldseq :=
1001110000000010011001101111010111001101111010000001110101111011000000;
BINMAP newseq {
abits := [70]; tbits := [71][70];  cbits := [72][71][70];
gbits := [72][71][70];   }
newseq := 0d; seqlength := 0d;
REPEAT (
IF (oldseq.nucleotide = 00)
THEN ( newseq.abits := 0;
   oldseq >> 2d;
   newseq >> 1d;
   seqlength := seqlength + 1d;)
IF (oldseq.nucleotide = 01)
THEN ( newseq.tbits := 10;
   oldseq >> 2d;
   newseq >> 2d;
   seqlength := seqlength + 2d; )
IF (var1.nucleotide = 10)
THEN ( newseq.cbits := 110;
   oldseq >> 2d;
   newseq >> 3d;
   seqlength := seqlength + 3d; )
ELSE
   ( newseq.gbits := 111;
   oldseq >> 2d;
   newseq >> 3d;
   seqlength := seqlength + 3d; )
UNTIL (var1 = 0d)
PRINT newseq; PRINT seqlength;
```

Summary and Lessons Learned

* Athough summer term is SHORT
* Better sense of how does Ocaml work and creating a compiler
* We learned to appreciate the complexity behind routine operations like loops and if-then statements that we take for granted in existing languages
* The levels of abstraction that exist between the programming language and machine code
* Computer Science is more than just coding

Thank you

```
*BINMAP var1 {
*   nibble:=[3][2][1][0];
*}
*var1 := 10001110;
*var1.nibble := 0000;
*PRINT var1;
```