Description of the VHDL Library

1 The ROM Component

1.1 Component Description

The ROM component is a combinatorial read-only memory. It returns a data, as a signed number with a specified length, specified by an integer address.

1.2 Generation Process

Use `genVhdl[ "ROM" ]` function in package `Vhdl2.m`.

1.3 Files

`ROM.vhdx`: the architecture description of the ROM (see G.3).

`ComponentOfROM.vhdx`: the component description of the ROM (see G.2).

1.4 Parameters

`$size$`: the size of the read-only memory, in words.

`$comment$`: a comment.

`$values$`: the values associated to the ROM. This parameter is not mandatory, and if not supplied, 0 values are assumed.

`$name$`: the name assigned to the component.

`$wordLengthMun$`: the word length (minus one) of the values in the ROM.

1.5 Example

```
--
-- Component for a combinatoric ROM
-- ROM of size 32 and of word length 2
--

COMPONENT ROM3 IS
  PORT
  (
    address : IN INTEGER RANGE 0 TO 31;
    data : OUT SIGNED (1 DOWNTO 0)
  );
END COMPONENT;
```

2 The moduloAddress Component

2.1 Component Description

The `moduloAddress` component is a component that generates...

2.2 Generation Process

Use `genVhdl[ "moduloaddress" ]` function in package `Vhdl2.m`.

2.3 Files

`ModuloAddress.vhdx:` the architecture description of the modulo address generator (see G.3).

`ComponentOfModuloAddress.vhdx:` the component description of the modulo address generator (see G.2).

2.4 Parameters

`$period$:` the period of the modulo generator.

`$comment$:` a comment.

`$periodMun$:` perion minus one. This is generated automatically by the VHDL generator.

`$name$:` the name assigned to the component.

A The ROM

A.1 Architecture

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;
--
--  This is a generic read-only memory.
--  Parameters are the name, the size, the word length, the values.
--

-- $comment$

-- Entity definition
ENTITY $name$ IS
PORT
  (
    address : IN INTEGER RANGE 0 TO $sizeMun$;
    data : OUT SIGNED ($wordLengthMun$ DOWNTO 0)
  );
END $name$;

-- Architecture
ARCHITECTURE archiOf$name$ OF $name$ IS

  TYPE values IS ARRAY (0 TO $sizeMun$)
    OF SIGNED ($wordLengthMun$ DOWNTO 0);

  CONSTANT rom : values :=
    ($values$);

BEGIN

-- Get the value
  data <= rom(address);

END archiOf$name$;
```

A.2 Component

```
--
-- Component for a combinatoric ROM
-- $comment$
--

COMPONENT $name$ IS
  PORT
  (
    address : IN INTEGER RANGE 0 TO $sizeMun$;
    data : OUT SIGNED ($wordLengthMun$ DOWNTO 0)
  );
END COMPONENT;
```

B The Modulo Address Generator

B.1 Architecture

```
--
-- This VHDL Pattern allows a modulo address to be generated
-- The parameters are the name, the period, the period minus one,
-- the size of the address (it is calculated automatically by the
-- generator)
--
-- $comment$
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

ENTITY $name$ IS
  PORT(
    clk : IN STD_LOGIC;         -- global clock
    ceGen : IN STD_LOGIC;       -- general clock enable signal
    rst : IN STD_LOGIC;         -- reset signal
    address : OUT INTEGER RANGE 0 TO $periodMun$
  );
END $name$;

ARCHITECTURE archiOf$name$ OF $name$ IS
  SIGNAL addr: INTEGER RANGE 0 TO $periodMun$;
BEGIN

  PROCESS( clk )
  BEGIN
    IF rising_edge(clk) THEN
      IF (ceGen='1') THEN   -- when ceGen is 1 modify address
        IF (rst='0') OR (addr = $periodMun$) THEN
          addr <= 0;
        ELSE
          addr <= addr + 1;
        END IF;
```

```
      END IF;
    END IF;
  END PROCESS;

  address <= addr;

END archiOf$name$;
```

B.2 Component

```
--
-- Component for a modulo address generator
-- $comment$
--

COMPONENT $name$ IS
  PORT(
    clk : IN STD_LOGIC;
    ceGen : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    address : OUT INTEGER RANGE 0 TO $periodMun$
  );
END COMPONENT;
```

C The Registers

C.1 Architecture

```
--
-- This VHDL Pattern allows a sequence of registers to be generated
-- The parameters are the name, the size of the elements, and
-- the number of registers
--
-- $comment$
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

ENTITY $name$ IS
  PORT(
    clk : IN STD_LOGIC;         -- global clock
    enable : IN STD_LOGIC;      -- clock enable
    inReg : IN STD_LOGIC;          -- input
    outReg : OUT STD_LOGIC         -- output
  );
END $name$;

ARCHITECTURE archiOf$name$ OF $name$ IS
  SIGNAL regfile: $type$;
  BEGIN

  PROCESS(clk, enable)
  BEGIN
    IF rising_edge(clk) THEN
      IF enable = '1' THEN
        $def$;
      END IF;
    END IF;
  END PROCESS;

  outReg <= $output$;
```

```
END archiOf$name$;
```

C.2 Component

```
--
-- Component for a register file
-- $comment$
--

COMPONENT $name$ IS
  PORT(
    clk : IN STD_LOGIC;
    enable : IN STD_LOGIC;
    inReg : IN STD_LOGIC;
    outReg : OUT STD_LOGIC
  );
END COMPONENT;
```

D The FSM component

D.1 Architecture

```
--
-- Finite state machine
-- $comment$
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

ENTITY $name$ IS
  PORT(
    clk : IN STD_LOGIC;
    CE_gen : IN STD_LOGIC;
    rst : IN STD_LOGIC;
$outputs$
  );
END $name$;

ARCHITECTURE archiOf$name$ OF $name$ IS

-- Declaration of the states
TYPE state_type IS ($states$);

ATTRIBUTE ENUM_ENCODING: STRING;
ATTRIBUTE ENUM_ENCODING OF state_type : TYPE IS $encoding$;

SIGNAL curstate, nextstate : STATE_TYPE;
SIGNAL count : INTEGER;

BEGIN

---- Synchronous reset process
PROCESS(rst,clk)
BEGIN
  IF clk = '1' AND clk'event THEN
```

```
    IF CE_gen = '1' THEN
      IF rst='0' THEN
        curstate <= state0;
        count <= 0;
      ELSE
        curstate <= nextstate;
        count <= count+1;
      END IF;
    END IF;
  END IF;
END PROCESS;

-- Transition function --
PROCESS( count, curstate )
BEGIN
  CASE curstate IS
$transitions$
  END CASE;
END PROCESS;

-- Output function
$actions$

END archiOf$name$;
```

D.2 Component

```
--
-- $comment$
--
COMPONENT $name$ IS
  PORT(
    clk : IN STD_LOGIC;
    CE_gen : IN STD_LOGIC;
    rst : IN STD_LOGIC;
$outputs$
  );
END COMPONENT;
```

D.3 Call

```
G$number$ : Fsm PORT MAP (clk, CE, rst, $outputs$);
```

E The Call reg component

E.1 Architecture

```
-- $comment$
PROCESS(clk, $enable$)
BEGIN
  IF rising_edge(clk) THEN
    IF $enable$ = '1' THEN $lhs$ <= $rhs$; END IF;
  END IF;
END PROCESS
```

F The Call registers component

F.1 Architecture

```
G$number$ : $name$ PORT MAP (clk, CE, $enableIn$, $enableOut$);
```

G The PeriodicEnable component

G.1 Architecture

```
--
-- This VHDL Pattern allows a periodic enable to be generated.
-- The parameters are the name, the period, the period minus one,
-- the size of the counter (it is calculated automatically by the
-- generator)
--
-- $comment$
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

ENTITY $name$ IS
  PORT(
    clk : IN STD_LOGIC;          -- global clock
    ceGen : IN STD_LOGIC;       -- general clock enable signal
    rst : IN STD_LOGIC;          -- reset signal
    periodicGe : OUT STD_LOGIC     -- periodic clock enable
  );
END $name$;

ARCHITECTURE archiOf$name$ OF $name$ IS
  SIGNAL counter : STD_LOGIC_VECTOR ($size$ DOWNTO 0);
BEGIN

  PROCESS( clk )
  BEGIN
    IF rising_edge(clk) THEN
      -- do something only when ceGen is non zero
      IF ceGen='1' THEN
        IF rst='0' THEN        -- enable is 0 when rst
          counter <= $zero$;
        ELSIF counter /= $zero$ THEN
          counter <= counter - 1;
```

```
        ELSE
          -- or set counter to period - 1 when period reached
          counter <= $periodMun$;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- Output true when counter is zero and clock enable general is true
  periodicGe <= '1' WHEN (counter = $zero$ AND ceGen = '1') ELSE '0';

END archiOf$name$;
```

G.2 Component

```
--
-- Component for a periodic enable of period $period$
-- $comment$
--

COMPONENT $name$ IS
  PORT(
    clk : IN STD_LOGIC;
    ceGen : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    periodicGe : OUT STD_LOGIC
  );
END COMPONENT;
```

G.3 Call

```
G$number$ : $name$ PORT MAP (clk, CE, reset0, $enable$);
```