IMPERIAL COLLEGE LONDON Master Copy

E3.06
AC5
ISE3.5

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2009

MSc and EEE/ISE PART III/IV: MEng, BEng and ACGI

VHDL AND LOGIC SYNTHESIS

Thursday, 7 May 10:00 am

Time allowed: 3:00 hours

There are FOUR questions on this paper.

Question 1 is COMPULSORY
Answer question 1 and any TWO of questions 2-4
Question 1 carries 40% of the marks, questions 2-4 each carry 30% of the marks.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : T.J.W. Clarke, T.J.W. Clarke
Second Marker(s) : C. Bouganis, C. Bouganis

Special Information for Invigilators: none.

Information for Candidates

VHDL language reference can be found in the booklet VHDL Exam Notes.

Unless otherwise specified assume VHDL 1993 compiler.

All packages that must be explicitly referenced with USE, e.g. IEEE.numeric_std, must be explicitly noted in each answer: semantically correct LIBRARY and USE statements may be omitted where this is done.

The Questions

Question 1 is COMPULSORY, and carries 40% of the total marks

1.

a) For each of the 4 VHDL processes P1, P2, P3,P4 in *Figure 1.1* indicate whether the process synthesises without error. State briefly the reasons for any errors during synthesis. If synthesis is possible state what is the driven signal list and whether the pre- and post-synthesis VHDL has similar function.

[4]

b) Write a synthesisable VHDL architecture for entity *count* in *Figure 1.2* which implements a positive edge clocked register such that the output q is as defined in Figure 1.2, where input x is interpreted as a signed integer.

[4]

c) You are given a VHDL package *data* in library MYMATH which defines constant array *values* as in *Figure 1.3*. Write a VHDL entity and architecture that uses *values* to implement a 1024 word X 10 bit ROM. All elements of *values* are non-negative and less than 2^{10}.

[4]

d) Draw the waveform diagrams of the outputs of P4 in *Figure 1.4*, annotating all transitions in the first 21 ns of simulation with time and simulation delta, assuming that *clk* waveform is as shown in *Figure 1.5*. For example the annotation 4ns+2Δ means the third simulation delta at physical time 4ns.

[8]

```
P1:PROCESS(x,y)
BEGIN
  IF z = '1' THEN
    a <= x;
  ELSE
    a <= y;
  END IF;
END PROCESS P1;
```

```
P2:PROCESS(x,y,z,w)
BEGIN
    x <= y AND w;
    y <= z;
    z <= x;
END PROCESS P2;
```

```
P3:PROCESS(x,y,z)
BEGIN
WAIT UNTIL x'EVENT AND x='0';
   clk <= y + z;
END PROCESS P3;

P4:PROCESS(x)
BEGIN
  x(0 TO 3) <= not x(1 TO 4);
END PROCESS P4;
```

Figure 1.1

```
ENTITY count IS
PORT(
 q  : OUT unsigned(9 DOWNTO 0);
 x: IN signed(2 DOWNTO 0);
 clk,reset  : std_logic
);
END count;
```

$reset_n$	q_n	q_{n+1}
0	*don't care*	*0*
1	q	$q + x$

x_n, q_n denote the value of x, q in the nth cycle of *clk*.

Figure 1.2

```
PACKAGE data IS
   TYPE romtype IS ARRAY (0 TO 1023) OF INTEGER;
   CONSTANT values: romtype := (
      0, 11, 23, …. , 987, 1
   );
END PACKAGE data;
```

Figure 1.3

```
P4: PROCESS(clk,a,b,c)
BEGIN
   a <= clk;
   b <= a;
   c <= TRANSPORT a AFTER 10 ns;
   d <= (a xor b) or c;
END PROCESS P4;
```

Figure 1.4

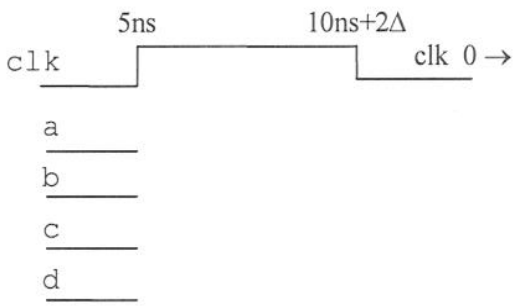


Figure 1.5

Students must answer TWO out of Questions 2-4. Each question carries 30% of total marks.

2.

a) Write a VHDL architecture for entity *control* in *Figure 2.1* which implements the FSM state transitions illustrated in *Figure 2.2*.

[12]

b) Implement additional hardware in your architecture which ensures that, in state *a*, *x* is identical to *y*, in state *b*, *x* is identical to *z*, and in any other state, *x* holds its value from the *previous* state, as shown in *Figure 2.3*.

[8]

```
USE IEEE.numeric_std.ALL
ENTITY control IS
PORT(
   x: OUT unsigned(31 DOWNTO 0);
   y, z: IN unsigned(31 DOWNTO 0);
   clk, p,q,r: std_logic
);
END control;
```

Figure 2.1

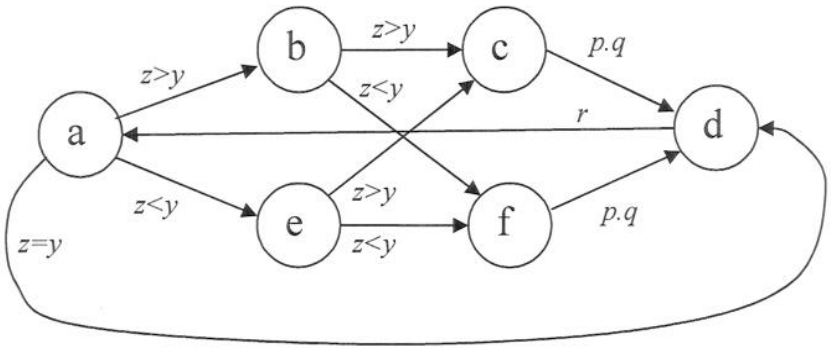


Figure 2.2

$state_n$	y_n	z_n	x_{n-1}	x_n
a	*y*	*z*	*x*	*y*
b	*y*	*z*	*x*	*z*
c,e,f,d	*y*	*z*	*x*	*x*

Figure 2.3

3.

a) *Figure 3.1* shows the entity *prng* for a pseudo-random number generator implemented using a 16 bit shift register as shown in *Figure 3.2*. On the positive edge of *clk* the shift register bits move one position to the right, thus the new value of $q(1)$ will be the old value of $q(0)$. The block XOR implements an exclusive or function of all its inputs. The shift register must be initialised to equal 8 bit *signed* value *seed* when input *init* is '1'. Write a synthesisable VHDL architecture for *prng*.

[10]

b) Write a synthesisable VHDL architecture for entity *array_prng* in *Figure 3.3* which contains 10 *prng* blocks initialised with seeds 1 to 10. The output *y* is equal to the exclusive or of the $q(15)$ outputs of all the *prng* blocks.

[10]

```
ENTITY prng IS
PORT( q    : OUT std_logic_vector(15 DOWNTO 0);
      seed: IN  std_logic_vector(7 DOWNTO 0);
      clk, init: IN std_logic );
END prng;
```

Figure 3.1

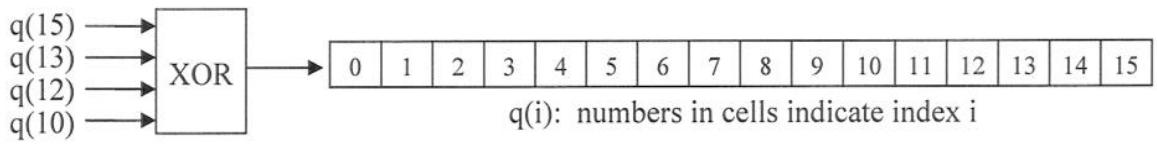


Figure 3.2

```
ENTITY array_prng IS
PORT( y: OUT std_logic;
      clk, init: IN std_logic );
END array_prng;
```

Figure 3.3

4.

a) *Figure 4.1* depicts a three level binary tree of combinational multipliers connected to multiply an array of eight 3 bit signed numbers $x(0)$, ..., $x(7)$. By considering the possible values of signed numbers at each stage in the multiplication, determine the width of each signal path and hence define VHDL types for x, y, z and w.

[5]

b) Using the types defined in part a, write an entity and synthesisable VHDL architecture implementing the multiplier tree in *Figure 4.1* with 3 bit signed inputs. Credit will be given for appropriate use of FOR LOOP constructs.

[12]

c) If the binary tree has n levels, with 3 bit signed numbers input to the first level, calculate the required length of the outputs from the m th level ($m < n$). State, giving reasons for your answer, whether it would be possible to implement this design in VHDL as an architecture to an entity with generic input n, using an outer loop that iterates over the n multiplier levels.

[3]

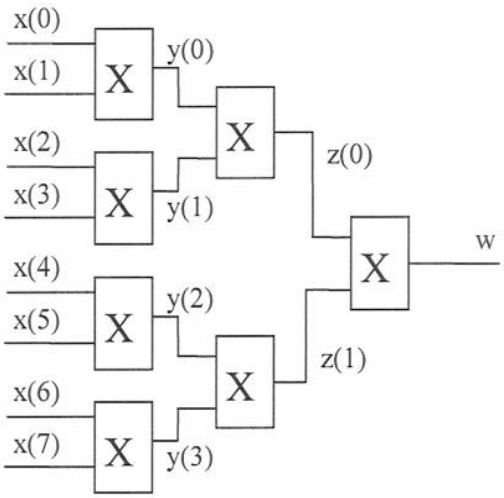


Figure 4.1

[END OF QUESTIONS]

VHDL & LOGIC SYNTHESIS

EXAM NOTES

SUMMER 2009

VHDL Sequential (Behavioural) Statements - inside PROCESS body

Meta-language	
[]	*optional part*
*	*0 or more repetitions*
{}	*grouping brackets*

- See also the dataflow statements which can also occur inside a PROCESS - previous slide
- Sequential statements (except WAIT) take zero simulation time to execute

```
variable := value ; -- variable assignment
NULL; -- empty statement
WAIT [ ON signal ]  [ UNTIL condition ] ;
WAIT FOR time ;

IF condition THEN statements
[ ELSIF condition THEN statements ]
[ ELSE statements ]
END IF ;

FOR var IN range LOOP
   for-statements
END LOOP;

CASE  var IS
     WHEN case1 => statements
     [ WHEN case2 => statements ]
     [ WHEN OTHERS => statements ]
END CASE ;
```


VHDL Dataflow statements

(Sequential also)

signal <= [TRANSPORT] *value* [AFTER *time*];

subprogram(para1 [, para2 [, ...]]);

[ASSERT *condition*] REPORT *message-if-false*
[SEVERITY *level*] ;

level ::= note | warning | error | failure

(Dataflow only)

{} meta-language grouping brackets - not VHDL

```
signal <=   value WHEN condition
          [ ELSE value WHEN condition ]
            ELSE value ;

WITH sel SELECT
   signal <=    value WHEN sel-case
              [ , value WHEN sel-case ]
                [ , value WHEN OTHERS ] ;

label :FOR var IN range GENERATE
     dataflow-statements
END GENERATE;

IF condition GENERATE
     dataflow-statements
END GENERATE;
```

```
label :  {ENTITY entity-name} | component-name
     GENERIC MAP( gen-map [ , gen-map ] )
     PORT MAP( port-map  [, port map] ) ;

[ label : ] PROCESS [ ( sensitivity-list ) ]
          process-declarations
     BEGIN sequential statements
     END PROCESS [ label ];

[ label : ] BLOCK local-declarations
     BEGIN dataflow-statements
     END [ label ] ;
```


Signal Attributes & Design Units

Signal Attributes

SIGNAL x has type T

Expression	Type	Description
x'EVENT	Boolean	TRUE if event on x
x'DELAYED(*del*)	T	x delayed by time *del*
x'LAST_VALUE	T	value of x before last or current event
x'LAST_EVENT	TIME	elapsed time from last event
x'STABLE(*tim*)	Boolean	FALSE if event on x within time *tim*

Design Units

```
ENTITY myentity IS
GENERIC( signal-list );
PORT( port-signal-list );
END myentity;

ARCHITECTURE archname OF entityname IS
declaration-statements
BEGIN
dataflow-statements
END archname;

PACKAGE mypackage IS
declarations
END [mypackage];

PACKAGE BODY mypackage IS
function and procedure body definitions
END PACKAGE BODY mypackage;
```


VHDL array syntax

unconstrained_array_type **::= STD_LOGIC_VECTOR | SIGNED | UNSIGNED | etc**
range **::= *low* TO *high* | *high* DOWNTO *low* | array_*signal* 'RANGE**

TYPE *my_type* IS ARRAY *range* OF *base_type*;
SUBTYPE *my_subtype* IS *unconstrained_array_type*(*range*) ;

SIGNAL | VARIABLE | CONSTANT name : unconstrained_array_type(range);

my_array(*range*) -- array slice on LHS or RHS
my_array(*index*) -- array element on LHS or RHS
(*val1*, *value2*, *value3*) -- array value using element values specified via position on RHS
(*index1*=>*val1*, *index2*=>*val2*, ..., OTHERS=>*valn*) -- array value on RHS

array'LEFT array'LOW
array'RIGHT array'HIGH
array'LENGTH
array'RANGE (see definitions of range above)

VHDL Declarations

Declarations

SIGNAL *sname* : *stype* [:= *init_val*];
CONSTANT *cname*: *ctype* := *init_val*;
VARIABLE vname: vtype [:= init_val];
SHARED VARIABLE vname: vtype [:= init_val];
FILE *fname*: *ftype* [OPEN *file_open_kind* IS *file_name_string*];
TYPE *tname* : *tspec*;

Types

INTEGER 1, 123, -3456
NATURAL <non-negative integer>
REAL 1.21, -0.033
CHARACTER '*', '0',
STRING "my string"
BOOLEAN FALSE, TRUE
TIME 10.1 ns, 11 fs, 10 min (units fs, ps, ns, us, ms, s, min, hr) -- physical time constant
INTEGER RANGE *low* TO *high* -- fixed range integer type
TYPE *enumeration_type* IS (*value_name-1, value_name-2, ..., value_name-n*); --enumeration type

VHDL Operators

Logical (**not** is unary)	**and**, **or**, **nand**, **nor**, **xor**, **xnor**, **not** S X S -> B, B X B -> B (unary S->S, B->B)
Relational	=,/= (any type) <,<=,>,<= (scalar or discrete types)
Shift	**sll**,**srl**,**sla**,**sra**,**rol**,**ror** V X I -> V Shift Left/Right Logical/Arithmetic Rotate Left/Right
Addition	+, - N->N, N X N -> N (all same type) Also in *std_logic_arith* for V X V -> V etc
Multiply	*,/ N X N->N (all same type) **mod**, **rem** Integer
Misc ******: exponentiation **abs**: absolute value	** N X N->N (all same type) a ** b = a^b **abs** N->N (both same type)
Concatenation	& V X V -> V, V X S -> V, S X V -> V

Key to Types

V: std_logic_vector
N: integer or real
I: integer
S: std_logic
B: Boolean

Logical, Relational, Shift, Additive, Concatenation ops are synthesisable on *vectors* and *fixed range integers*

VHDL Text File I/O

TYPE *file_type* IS FILE OF *element-type*;

FILE *file_object*: *file_type* OPEN *file_open_kind* IS *file_name_string;*

--"Automatic" file open & close in object declaration

--TEXT file access uses Package STD.TEXTIO

--Use statement required (but no library statement, since STD is always available)

--Defines TEXT file type, LINE linebuffer type

TYPE SIDE IS (right, left);

TYPE FILE_OPEN_KIND IS (read_mode, write_mode, append_mode);

TYPE FILE_OPEN_STATUS IS (open_ok, status_error, name_error, mode_error);

file_open(VARIABLE status: OUT FILE_OPEN_STATUS;
FILE *f*: TEXT: name: IN STRING;
mode: IN FILE_OPEN_KIND := read_mode);

file_close(FILE f: TEXT);

endfile(FILE f: TEXT) RETURN boolean;

readline(FILE *f*: TEXT; *line*: OUT LINE);
-- read the next line from f into line

writeline(FILE *f*: TEXT; *line*: INOUT: LINE);
-- write a line from line to f, clearing line

read(*line*: INOUT LINE; *value*: OUT *<any-type>*);

write(*line*: INOUT LINE;
value: IN *<any-type>*;
justified: IN SIDE := right;
field: IN WIDTH := 0);

<any-type> ::= BIT, BIT_VECTOR, CHARACTER, INTEGER, REAL, TIME, STRING

Functions & Procedures

Meta-language
[] *optional part*
* *0 or more repetitions*
{} *grouping brackets*

```
PACKAGE mypack IS
   <function header>;
END PACKAGE mypack;
```

<function header> ::=
[IMPURE] FUNCTION *myfunc*[(<par> { ; <par> }*)]
RETURN *rtype*;
<par> ::= [<pspec>] *pname* : [<fmode>] *ptype* ;
<pspec> ::= FILE | SIGNAL | VARIABLE | CONSTANT
-- omit <pspec> for value IN parameter
-- may omit <pspec> for VARIABLE OUT mode
<fmode ::= IN | OUT | INOUT -- default mode is IN
-- functions can only have IN parameters

```
PACKAGE BODY mypack IS
   <function_header> IS
      <function-declarations>
   BEGIN
      <function-statements>
   END FUNCTION;
END PACKAGE BODY mypack;
```

<function-statement>::= <sequential_statement> |
RETURN *expression*;

- PROCEDURE as for FUNCTION but
  - No RETURN *rtype* in header
  - WAIT allowed in body
  - OUT, INOUT parameters are allowed
- FUNCTIONs & PROCEDUREs can be defined in package body without duplicating header in package - in this case their scope is restricted to package body.
- Functions with zero parameters have no brackets in header or function call.

Built-in functions

now -- returns current simulation time
'POS -- CHARACTER -> ASCII code
'VAL -- ASCII code -> CHARACTER

<scalar_type_name>'IMAGE(<value>)
-- <value> -> printable string

```
TYPE brec IS RECORD
   finished, first: BOOLEAN;
   count: INTEGER;
END RECORD;

TEST: PROCESS
 VARIABLE bb: brec;
BEGIN
 IF bb.finished OR bb.first THEN
   bb.count := bb.count+1;
 END IF;
 ....
END PROCESS TEST;
```

```
TYPE <record-type-name> IS
RECORD
  <field-name-list> : <field-type>;
  {<field-name-list> : <field_type> ;}*
END RECORD;
```

Record & Access types

- **Record types** can contain any other type
- Signals, variables, constants, files, array elements can have record type
- **Access types** implement pointers, & allow variable length arrays or strings

```
-- typical uses, access string, access std_logic_vector,
-- but could be any type
type string_v is access string; -- variable-length strings

process
  variable sv: string_v:= new string'("0123"); -- newly created string "0123"
  variable sv1; --default value null;
begin
  report "sv is initialised to: " & sv.all; -- dereference to print
  sv.all := "abcd"; -- must be same length, overwrites string but same pointer
  report "4th element of sv is:" & sv(4); -- dereference one element
  sv := new string("n is:" & integer'image(n)); -- new pointer, can be any length
  report "sv is now: " & sv.all;
end;
```

```
TYPE <access-type-name> IS
ACCESS <access_type_base>;
 NEW <init-value>; -- create new pointer
(to be assigned to an access type object)

   If ac is an access type object:

ac.ALL -- dereference all of ac
ac(3)    -- dereference 1 element of
         -- array access type ac
NULL     -- null pointer
```

S.9

E3.06 | Int 3.5 | ACS

Master

VHDL 2009 SOLUTIONS
A=Analysis, D = Design, B = Bookwork

Question 1 is COMPULSORY, and constitutes 40% of marks, 72 minutes time, 15 minutes per part.

Solution to Question 1.

1.

a)

P1: synthesises OK. Driven list a. Incomplete sensitivity list means that pre-synthesis code behaves **differently** from post-synthesis.

P2: cyclic assignment prevents synthesis.

P3: Cannot have sensitivity list and WAIT.

P4: synthesises OK, driven list x(0 to 3), pre & post synth code same.

[4A]

b)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164;
USE IEEE.numeric_std;

ARCHITECTURE synth OF count IS
      SIGNAL q_int: signed(7 DOWNTO 0);
BEGIN

      WAIT UNTIL clk'EVENT and clk='1';
      IF reset='0' THEN
            q_int <= (OTHERS=>'0');
      ELSE
            q_int <= q_int + x;
      END IF;
      q <= unsigned(q_int);
END synth;
```

[4D]

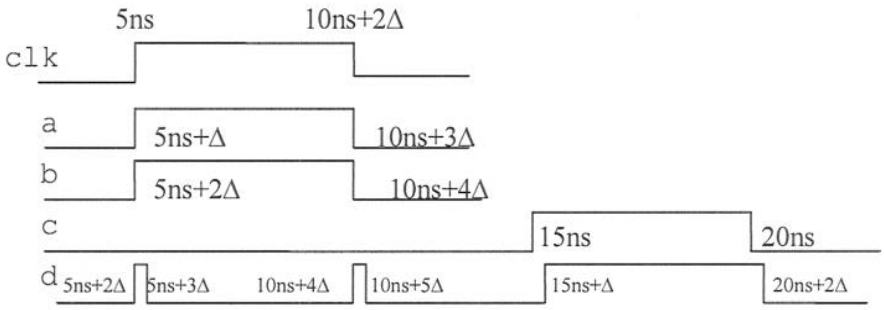
c)

```
LIBRARY IEEE;
LIBRARY MYMATH;
USE IEEE.std_logic_1164;
USE IEEE.numeric_std;
USE MYMATH.data.ALL;

ENTITY rom IS
PORT( addr: IN std_logic_vector(9 DOWNTO 0);
      data: OUT std_logic_vector(9 DOWNTO 0));
END rom;
-- NB could simplify code using unsigned addr & data but less standard
ARCHITECTURE synth OF rom IS
BEGIN
     data <=
     std_logic_vector(to_unsigned(values(to_integer(unsigned(addr),10));
END synth;
```

[4D]

d)



[8A]

VHDL 2009 SOLUTIONS
A=Analysis, D = Design, B = Bookwork

Students must answer two questions from questions 2-4, each question caries 30% of marks and takes 54 minutes.

Solution to Question 2

a)

```
ARCHITECTURE synth OF control IS
BEGIN
   TYPE fsmstate IS (a,b,c,d,e,f);
   SIGNAL ss, nss: fsmstate;
P1: PROCESS(ss,z,y,p,q,r)
BEGIN
      ns <= ss; --default
      CASE ss IS
      WHEN a => IF z=y THEN nss <= d;
                ELSIF z > y THEN
                   nss <= b;
                ELSE
                   nss <= e;
                END IF;
      WHEN b | e => IF z > y THEN
                       nss <= c;
                    ELSIF x < y THEN
                       nss <= f;
                    END IF;
      WHEN c | f => IF p and q = '1' THEN
                       nss <= d;
                    END IF;
      WHEN d => IF r = '1' THEN
                  nss <= a;
                END IF;
      END CASE;
END synth;
```

[12B/D]

b)

```
P1: PROCESS(ss,xx,y,z)
BEGIN
    CASE ss IS
       WHEN a => x<=y;
       WHEN b => x <= z;
       WHEN OTHERS => x <= xx;
    END CASE;
END
P2: PROCESS
BEGIN
    WAIT UNTIL clk'EVENT and clk='1';
    CASE ss IS
       WHEN a => xx <= y;
       WHEN b => xx <= z;
       WHEN OTHERS => NULL;
    END CASE;
END PROCESS P2;
```

[8D]

Solution to Question 3

a)

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ARCHITECTURE synth OF prng IS
  SIGNAL q_int: STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
      P1: PROCESS
      BEGIN
            WAIT UNTIL clk'EVENT and clk='1';
            IF init='1' THEN
                q_int <=std_logic_vector(resize(signed(seed),16))
            ELSE
                q_int(15 DOWNTO 1) <= q_int(14 DOWNTO 0);
                q_int(0) <= q_int(15) xor q_int(13) xor q_int(12) xor q_int(10);
            END IF;
      END PROCESS P1;

      q <= q_int;

END ARCHITECTURE synth;
```

[10D]

b)

```
ARCHITECTURE synth of array_prng IS
   TYPE q_array IS ARRAY (0 TO 9) OF std_logic_vector(15 DOWNTO 0);
   TYPE seed_array IS ARRAY (0 TO 9) OF std_logic_vector(7 DOWNTO 0);
   SIGNAL qa: q_array;
   SIGNAL sa: seed_array;
BEGIN

FOR i IN 0 TO 9 GENERATE
   I1: ENTITY poly PORTMAP(q=>qa(i), seed=>sa(i), clk=>clk, init=>init);
   sa(i)<= conv_std_logic_vector(i+1, 16);
END GENERATE;

P1: PROCESS(q)
   VARIABLE tmp: std_logic;
BEGIN
   tmp := '0';
   FOR i IN 0 TO 9 LOOP
      tmp := tmp xor qa(i)(15);
   END LOOP;
   y <= temp;
END PROCESS;

END ACHITECTURE synth;
```

[10D]

Solution to Question 4

a)

Each input *x* has range [-4,3]. the maximum result range is therefore [-12,+16]. Thus *y* requires 6 bits for signed representation. The range of the next level, z, is dominated by +256, and therefore requires 10 bits. Similarly w is dominated by 2^{16} and so requires 18 bits.

```
TYPE xa IS ARRAY (0 TO 7) OF std_logic_vector(2 DOWNTO 0);
TYPE ya IS ARRAY (0 TO 3) OF signed(5 DOWNTO 0);
TYPE za IS ARRAY (0 TO 1) OF signed(9 DOWNTO 0);
SUBTYPE wa IS std_logic_vector(17 DOWNTO 0)
```

[5B/A]

b)

```
-- assume above types defined in package used here
ENTITY mult_tree IS
PORT(
   x: IN xa;
   w: OUT wa;
END mul_tree;
```

[2D]

```
ARCHITECTURE synth Of mult_tree IS
BEGIN

   SIGNAL y: ya;
   SIGNAL z: za;

P1: PROCESS(x,y,z)
VARIABLE tmpz: signed(11 DOWNTO 0); -- used to match sizes
VARIABLE tmpw: signed(19 DOWNTO 0);
BEGIN
   FOR i in 0 TO 3 LOOP
   y(i) <= signed(x(i*2))*signed(x(i*2+1));
   END LOOP;
   FOR i in 0 TO 1 LOOP
   tmpz := (x(i*2)*x(i*2+1));
   z <= tmpz(9 DOWNTO 0);
   END LOOP;
   tmpw<= (z(0)*z(1))(17 DOWNTO 0);
   w <= std_logic_vector(tmpw(17 DOWNTO 0));

END PROCESS P1;
END ARCHITECTURE synth;
```

[10D]

c)

In general case, after m levels we have width of $2^{m+1} + 2$ bits needed by the output. This design can't be implemented like this in a single loop because the signals needed are different widths - unless one maximum width of signal is used everywhere and it is assumed that synthesis will optimise widths. This would work.

[3A]