Electronic Reverse Engineering
• Introduction and Motivation
• HDL and VeriLog
• Conversion Methodology
• AVGM System Architecture
  – Image Analysis System
  – Circuit graph
  – Graph translator to links (report generation)
  – Translation to Verilog HDL model
• Model example
• Industrial Practice
Introduction and Motivation

legacy systems main problems are:

• The piece parts, modules, IC’s, etc. Are out of stock or have been superseded. Therefore it is very hard to acquire them.

• Incomplete technical information or the technical information is not available.

• In particular cases the system technical information is not always updated after some modifications have been performed. The real system does not conform to the posted technical information.

Goals:

• Migration into new Technologies
• Update Systems
Purpose of HDL Languages

- Simplify design so you can concentrate on the overall picture
- HDL allows description in language rather than schematic, speeding up development time
- Allows reuse of components (standard cells ie libraries)
VHDL/Verilog Differences

• Verilog modelled after C, VHDL is modelled after Ada
  – VHDL is more strongly typed (ie it checks the typing more rigorously)
  – Verilog is case sensitive while VHDL is not

• This means that VHDL has a higher learning curve than Verilog, but with proficiency, offers more flexibility.

• Verilog used extensively in the US while VHDL is used internationally
Verilog Types

• wire, wire[msb:lsb] (single/multiple bit wire)
• reg, reg[msb:lsb] (single/multiple bit register)
• integer (integer type, 32 bits)
• time (unsigned 64 bit)
• real (floating point double)
• string
Operators

Arithmetic:
- Binary +, -, *, /, % (mod)
- Unary +, - (sign)

Relational
- Binary <, >, <=, >=
Equivalence Operators
• === (equivalence including x and z), ==, !== (again including x and z), !=

Bit-wise Operators
• ~ (not), & (and), | (or), ^ (xor), ~^, ^~ (xnor)
Operators (con’t)

• Shift Operators $<<$, $>>$
• Cocatenation $\{a, b, c\}$
• Replication $\{n\{m\}\}$ (m n times)
• Conditional
  
  \[
  \text{cond}\_\exp \ ? \ \text{True}\_\exp : \text{false}\_\exp
  \]
• Can be used without having to build
• and, nand, or, nor, xor, xnor (n-input gates), used as:
  and (out, in1, in2, in3, …)
• buf,bufif1,bufif0,not,notif1,notif0
  (transmission gates/tristates)
Entity Instantiation

• No equivalent of architecture keyword in Verilog (all is treated as one architecture)

VHDL:  entity myentity is
          port ( a, b : in std_logic;
              c : out std_logic);
       end myentity;

       Architecture implementation of myentity
       --do stuff
       end implementation;

Verilog: module myentity (a, b, c);
         input a, b;
         output c;
         wire a, b, c;
         //do stuff
         endmodule
• Verilog doesn’t require explicit instantiation

VHDL: component DFF

    port( d : in std_logic; q : out std_logic);
  end component;
MyDff : DFF
  Port map (foo, bar); --implicit
  Port map (d => foo, q => bar);--explicit
Verilog:    dff u0 (foo, bar); //implicit
            dff u0 (.foo(d), .bar(q)); //explicit
• Note that gate level primitives in Verilog can be a function or bitwise symbol

**VHDL:**

\[
a \leftarrow b;
\]
\[
a \leftarrow b \text{ AND } c;
\]
\[
a \leftarrow b + c;
\]

**Verilog:**

\[
a \leftarrow b; \quad \text{//parallel assignment}
\]
\[
\text{and}(a, b, c); \quad \text{//a is output}
\]
\[
a \leftarrow b \& c; \quad \text{//same as line above}
\]
\[
\{\text{carry}, \text{sum}\} \leftarrow b + c; \quad \text{//carry is}
\]
\[
\text{//optional}
\]
Flow Logic (I): Case

- Differentiation in VHDL for select statements and conditional assignment; none in Verilog

VHDL: With std_logic_vector'(A) select

\[
Q <= "1" \text{ when } "0"
\]

\[
<= "0" \text{ when } "1"
\]

\[
<= "0" \text{ when others;}
\]

VHDL can also use case statements similar to Verilog

Verilog: case (A)

\[
0 : Q <= 1;
\]

\[
1 : Q <= 0;
\]

\[
\text{default : } Q <= 0;
\]

endcase
• **Equivalence operators similar to C in Verilog**

**VHDL:**
```
Q <= '1' when A = '1' else
    '0' when B = '1' else
    '0';
```

**Verilog:**
```
if (A == 1) begin
    Q <= 1;
end else if (B == 1) begin
    Q <= 0;
end else begin
    Q <= 0;
end
```
VHDL:  process (a)
    begin
        b <= a;
    end process;

Verilog:  always @ (a)
    begin
        b <= a;
    end
• Example: D Flip Flop: (note edge triggering is in always statement in Verilog)

VHDL: process(clk)
    begin
        if rising_edge(clk)
            q <= d;
        end if;
    End process;

Verilog: always @ (posedge clk)
    begin
        q <= d;
    end
• Useful for simulation and verification
• Keyword “initial begin .. end” starts a testbench
  – Remember “always begin .. end”
• #delay is used to delay by delay ‘time units’
• $display(“Random text”) displays Random text
• `timescale sets the ‘time unit’ unit
### Migration methodology

<table>
<thead>
<tr>
<th>Features and Requirements</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>* No technical Information is available</td>
<td>* Complete Technical Information</td>
</tr>
<tr>
<td>* Incomplete Information</td>
<td>* Require Schematic</td>
</tr>
<tr>
<td>* Modified Systems</td>
<td>* Netlist or ABEL code</td>
</tr>
<tr>
<td>* Repaired Systems</td>
<td>* Conversion from TTL or PLD</td>
</tr>
<tr>
<td>* No Schematic available</td>
<td></td>
</tr>
<tr>
<td>* Conversion from TTL</td>
<td></td>
</tr>
</tbody>
</table>

#### Migration Methodology

- **Legacy Systems**
  - Reverse Engineering
  - Re-Engineering
  - New Technology

- **Standard Systems Technologies**

- **Legacy Systems**
  - Conversion Model
  - New Technology

- **Standard Systems Technologies**
AVMG: Automatic Verilog HDL Model Generator
AVMG System Architecture

- Image Acquisition
  - Low Level Image processing
    - High Level Image Processing
      - Information Classifier
        - Graph Generator
          - Translator to Links information code
            - Verilog HDL Code Generator

- Lighting System
- Optical Character Recognition
- Integrated Circuit Database

Simulate or Verify
Output representation of each AVMG steps

PCB Image

Circuit Graph

Link \langle A1,B3 \rangle
Link \langle A3,B6 \rangle
Link \langle B5,C2 \rangle
Link \langle A7,C8 \rangle
.
.
Link \langle 00,A6 \rangle

Elements decomposition
Translation

Code generation
HDL model
VERILOG XL
Image Analysis System

Real PCB image → Enhancing IC’s for identification → IC’s identification and board location

Link identified between B and C IC’s → Image with IC’s identified
Circuit graph

- Inputs
- Outputs
- Conexión point
Graph translator to links (report generation)

**Circuit Graph**

**Elements decomposition**

- Link (A1, B3)
- Link (A3, B6)
- Link (B5, C2)
- Link (A7, C8)
- ...
- Link (00, A6)

**Primitive Graph to Links**

Link (A1, B3) means A1 → B3
Pin No. 1 of device A, goes to Pin No 3 of B
Translation to Verilog HDL Model

Code Generation

Module flop(data, clock, q, qb)
    input data, clock, clear;
    output q, qb;

    nand #10 nd1 (a, data, clock),
    nd2(b, ndata, clock),
    nd4(d, c, b, clear),

    Module rcount_func(out, clk)
        input clk, clkb;
        output [2:0] q;
        flop f1(d[0], clk, q[0]);
    endmodule
Decoder 2/4 decoder

Link between Symbol
IC - IC *
Input - IC !
IC - Output #

Links
- link!(0002,B012)
- link!(0001,B013)
- link!(0002,B010)
- link%(B011,0004)
- link%(B008,0003)
- link%(B006,0005)
- link%(B003,0006)
- link!(0001,A001)
- link!(0002,A003)
- link*(A004,B001)
- link*(A002,B002)
- link*(A002,B005)
- link!(0002,B004)
- link*(A004,B009)
module decdos;
reg O/I[1:0],O/I[2:0],sal O/I[6:5],O/I[3:4],W[1:2],wire W[1:1],W[2:1],
`define LIB_LS7404 chip1 (O/I[1:0],O/I[2:0],W[1:1],W[2:1],O/I[6:5],O/I[3:4],O/I[6:5],O/I[3:4],)
// START TEST
always #100 begin
#200->final;
$finish;
end
endmodule

module decdos(m0,m1,m2,m3);
reg [1:0] ent;
output m0, m1, m2, m3;
wire w0,w1,w2,w3;
`define stim #100 ent = 2'b
event final;
LIB_LS00 chip2
(w1,w0,ent[0],w0,w1,ent[1],ent[0],ent[1],m0,m1,m2,m3);
LIB_LS04 chip1 (ent[1],ent[0], , , ,w0,w1, , , ,);
// START TEST
always #100 begin
$strobe ("time%d, ent[0]=%b, ent[0]=%b, %b, %b,%b",
$time, ent[1], ent[0],m0, m1, m2, m3);
end
begin repeat(2)
begin
ent = 2'b00;
`stim 01;
`stim 10;
`stim 11;
#200 ->final ;
end
$finish;
end
endmodule

Verify or Simulate  FPGA
Industrial Practice

• proprietary method that is considered company confidential

• Technologies used:
  – standard technologies, i.e., x-ray, milling and sanding
  – hand continuity tracing

• The primary output is usually:
  – block diagram, schematic diagram, circuit board layout, flowchart, or an operations manual
  – Gerber files for the production of the PCB

• Provided by:
  http://www.reverseengineerit.com/
  http://www.armisteadtechnologies.com/ and others
There are at least four basic steps in reverse engineering a printed circuit board.

1. Two samples of the board. Measure all chip capacitors in place then remove all components and log them.
2. Identify all of the electrical connections between components on the board (node list, sometimes called a net list) and build a schematic.
3. Capture the schematic in software including building component images for all components that are not in a parts library. Check the schematic connections to the board.
4. Generate Gerber files for the board.
   - x-ray technology in combination with deconstructing the board to reveal trace position (length, width, thickness, layer position) will help the engineer understand where high frequency traces need to be routed.

The previous steps will cover about 95% of all printed circuit board reverse engineering requirements.

The Patent/Copyright issue:

- If a board has a copyright mark, then you should not reverse engineer the traces verbatim. Rerouting the board using any one of the auto router software packages will generally render a board that is unlike the existing board although still electrically and schematically identical.
- The majority of the patented boards are patented for onboard firmware or highly specialized digital circuits. Unless you have very deep pockets, this is one PCB reverse engineering area that it is advisable to stay away from altogether. Of course, if your company owns the patent then there is no problem.
Reverse engineering PCB's is a necessary process to obtain lost manufacturing files (Gerbers).

Sometimes, reverse engineering combined with re-engineering can revitalize old circuits to save time and money.

Through RE Schematic diagrams can be provided.

Migration into a new Technology is possible.

Industrial RE is Vendor confidential.