Watermarking Technique for HDL-based IP Module Protection

Min-Chuan Lin, Guo-Ruey Tsai, Chun-Rong Wu, Ching-Hui Lin
Department of Electronic Engineering, Kun Shan University
mclin123@mail.ksu.edu.tw

Abstract

Reuse-based intellectual property (IP) design is one of the most promising techniques to take the SoC design quickly into market. To facilitate better IP reuse, it is desirable to have IP exchanged in the software form such as hardware description language (HDL) source codes. However, soft IP has higher protection requirements than hard IP, and most existing hard IP protection techniques are not applicable to soft IP. Here, we propose two practical schemes for HDL code protection by inherent characteristic of the FPGA, including look-up table (LUT) units and distributed SRAM, which can be properly documented and synthesizable for reuse. For combinational logic system, the LUT components are very suitable for hiding watermarking by assigning some author’s signature into unused logic states. For sequential logic system, we use RAM-based Finite State Machine (FSM) or Programmable Finite State Machine (PSM) to embed the personal watermark. Without changing the original algorithm in the reused device and increasing extra HDL modules, the proposed watermarking technique is suitable for HDL-based reused IP protection.

1. Introduction

Due to the portability, reuse and scaling-ability, the HDL based IP is preferred for most complex SoC (system-on-a-chip) system. The packaging of HDL based IP is so simple that the IP content is easy to be read, modified and reshaped into new invention, which is hard to justify who is the original creator. How to protect the HDL based IP is the key factor for prosperous marketing of soft IP.

SoC design adopts reusable IP module to shorten development period and cut cost. But the IP market always suffers from pirate danger which is harmful to the IP sharing and circulation. We need to embed authentication message into the IP design to claim private property. Many IP watermarking techniques have been proposed to claim privacy and to achieve anti-clone [1]. The watermarking techniques include two topics: watermark and authentication message embedding. Figure 1 shows the common watermarking technique. Both software and hardware watermarking are used to statically or dynamically prevent from attack.

Figure 1. Watermarking technique for IP protection

Figure 2 illustrates current watermarking authentication by specific patterns (hardware layout, Hash function), special data structure, or preset execution trace to claim the IP privacy. The watermarking should be equipped with credibility, low data-rate, perceptual invisibility, part protection, and resilience.

Figure 2. Watermarking authentication

There are four categories of watermarking techniques: static constraint-based watermarking [2], dynamic finite state machine (FSM) based watermarking [3, 4], digital signal processing watermarking [5], and direct access test scheme (DATS) watermarking [6]. Constraint-based
watermarking has low technique threshold, but weak detectable verification. FSM-based watermarking has low technique threshold and better detectable verification, but hard authentication. Digital signal processing watermarking does not have precise authentication process and case by case. DATS-based algorithm is a kind of tracing and extraction process based on test circuit of SoC, and test module will be combined with the original IP module to constitute a new IP module.

Both watermark tagging and tracking technique and watermark embedding technique can reduce the clone probability, but they are not suitable for text-typed HDL-based IP. Some module duplication and module splitting design [7] have been proposed, but can not resist attack from HDL experts.

Here we will propose a composite algorithm combining LUT-hiding and RAM-based FSM watermarking to protect HDL-based IP from attack. Section 2 will describe combinatory LUT-hiding watermark in detail. Section 3 will introduce sequential RAM-based FSM watermark hiding. Section 4 illustrates the Programmable Finite State Machine (PSM) to embed the personal watermark. Section 5 discusses the possible attack and corresponding protection strategies. Section 6 gives the summary.

2. Combinational LUT watermarking

The unused states in truth table can be used to store the author’s signature. Even not, we can expand the logic input to embed authentication message, shown as Figure 3. Both if/then/else and case/when syntax are easy to crack, so we build the LUT table to store the watermark which is encrypted by checksum, parity or CRC algorithm.

![Figure 3. Embedding watermarking authentication message in LUT](image)

In the code conversion table between BCD and Gray codes, there are six unused states, and watermark can be implemented here as shown in Table 1. In the code conversion table between binary and Gray code, there no unused states. We can add an extra 4-bit LUT, and then we have 16 unused states to store watermark message.

<table>
<thead>
<tr>
<th>LUT4_b0 : LUT4</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic map(INIT =&gt; X&quot;CA66&quot;) ---- watermarking</td>
</tr>
<tr>
<td>port map(O=&gt;gray_lutbcd(0), -- LUT general output</td>
</tr>
<tr>
<td>I0 =&gt; data_in(0), -- LUT input</td>
</tr>
<tr>
<td>I1 =&gt; data_in(1), -- LUT input</td>
</tr>
<tr>
<td>I2 =&gt; data_in(2), -- LUT input</td>
</tr>
<tr>
<td>I3 =&gt; data_in(3));  -- LUT input</td>
</tr>
<tr>
<td>-- End of LUT4_b0 instantiation</td>
</tr>
</tbody>
</table>

3. Sequential RAM-based FSM Watermarking

Adopting unused states of the FSM to embed the watermark will still suffer from the attack from HDL experts. Here, we adopt RAM-based FSM design to replace counters, shift resisters, and FSM HDL coding, shown as Figure 4.

All the FSM state transitions and their corresponding output can be stored into the RAM in form of data storage in advance. With the data storage form and without unused states, the watermarking message can be implemented into the FSM RAM and Y RAM.

![Figure 4. Basic framework of RAM-based FSM](image)

For example, we could have a system of which data_out will only last 1-clock long, and the data_in may last n-clock as shown in Figure 5.

![Figure 5. Example for RAM-based FSM Design](image)
Figure 6 illustrates all possible state transitions of the example system in Figure 5. A 4-state FSM needs minimum 2-bit state variable. Adding a data out output, the ROM/RAM will have 3-bit output. If the 2-bit state variable is fed as the address input, adding the 1-bit data_in, we only need a 8×3 ROM/RAM to build up a RAM-based FSM. The SRAM in FPGA chip are distributed, its basic storage capacity is 16×1. Therefore, an 8×3 SRAM and an 16×3 SRAM consume the same FPGA resource. We will adopt a 16×3 ROM/RAM to design the RAM-based FSM which has extra memory space for watermark message. The RAM-based FSM must specify every possible state transition, shown as Figure 6. Every state (st0, ..., st3) is coded by state codes which can be regarded as the embedded watermark. Using the state transition and state codes in Figure 6, and combining with the 1-bit data_in, we have the state transition table, shown as Table 2. Table 3 lists the corresponding HDL code of the RAM-based FSM.

**Table 2. State transition table in Figure 6**

<table>
<thead>
<tr>
<th>Current state</th>
<th>Data_out</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>0001</td>
<td>x</td>
<td>xx</td>
</tr>
<tr>
<td>0010</td>
<td>x</td>
<td>xx</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>0100</td>
<td>x</td>
<td>xx</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>0111</td>
<td>x</td>
<td>xx</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1010</td>
<td>x</td>
<td>xx</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>x</td>
<td>xx</td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1111</td>
<td>x</td>
<td>xx</td>
</tr>
</tbody>
</table>

**Table 3. HDL codes for RAM-based FSM**

```vhdl
data_outram <= s2; -- FSM output

ROM16X1_d0: ROM16X1
  generic map (INIT => X"4900") ---- watermarking
  port map (O => s0, -- ROM output
            A0 => A0, -- ROM address[0]
            A1 => A1, -- ROM address[1]
            A2 => A2, -- ROM address[2]
            A3 => A3 ); -- ROM address[3]

ROM16X1_d1 : ROM16X1
  generic map (INIT => X"6820") ---- watermarking
  port map (O => s1, -- ROM output
            A0 => A0, -- ROM address[0]
            A1 => A1, -- ROM address[1]
            A2 => A2, -- ROM address[2]
            A3 => A3 ); -- ROM address[3]

ROM16X1_d2 : ROM16X1
  generic map (INIT => X"2120") ---- watermarking
  port map (O => s2, -- ROM output
            A0 => A0, -- ROM address[0]
            A1 => A1, -- ROM address[1]
            A2 => A2, -- ROM address[2]
            A3 => A3 ); -- ROM address[3]

------ End of ROM16X1_inst instantiation
```

Using a LUT-based decoder to replace the Y-RAM output in Figure 4, Figure 7 illustrates a HDL-based IP module with multiple watermarking protections, which is equipped with both RAM-based FSM and LUT-based data path simultaneously.
4. PSM (programmable finite state machine) Watermarking

Both controller and data path modules in common digital system can be designed by a PSM and packaged together. Combining stored program and programmable processor, we propose a universal PSM to copy with various applications. With program memory, we can find spared space to store the watermarking message.

In the instruction set of PSM, we can use the Branch command for input decision, state transition, and data output. Its format is shown as the following:

```
Branch outdata, current_state
```

We use the VHDL code

```
pc := conv_integer (data_in & p_data(1 downto 0))
```

to let PSM transfer to the next program counter which is decided by data_in and current_state. When we adopt 4-bit program code to design the PSM, the Branch command can handle 2-bit output, 2-bit state variable, and 2-bit input data.

5. Attack Analysis

The proposed HDL-based IP protection strategies including RAM-based FSM and LUT-based data path design, can cope with two kinds of attacks: removal attacks and embedding attacks.

5-1 Masking and Removal Attacks

Removal attacks are divided into combinational redundancy removal and sequential redundancy removal. The LUT-based data path is built in form of data bits; each data bit is possible to be the watermark. To remove a LUT-based data path, one needs to completely understand the effective truth table of the combinatorial module. For a HDL program with several LUT-based data paths, it is really not economical to remove all watermarks. It is same for sequential case. For double protection including both RAM-based FSM and LUT-based data path, it is not real to try to crack and clone.

To attack PSM-typed watermarking, one need to resolve the PSM HDL programming code, reverse the instruction set and its function, and then study PSM execution program code from program ROM. The chance for success is very low.

5-2 Embedding Attacks (Forging)

To change LUT table, one must take care of too many detail. Thief action including both state re-encoding and state reduction, need to re-build RAM table and LUT table. It will spend more effort than re-design the IP. Strategies including cutting into small HDL design, packing by component syntax, and combining with other watermark modules, all can confuse the reversion attack.

6. Conclusions

Here we propose a simple watermark encryption algorithm to protect HDL-based IP. The main strategy is to utilize the intrinsic characteristics within FPGA: LUT units and distributed SRAM. For combinatory logic circuit, we use the LUT to replace the common VHDL syntax and embed the watermark message into the unused states. For sequential logic circuit, we use RAM-based FSM design, and implement the watermark message into the FSM state memory. For larger controller and data path module, the watermark message is stored within PSM program code. The proposed protection strategies do not change the original algorithm of the reuse device, and do not increase extra HDL program module either.

7. References