ReDeEm_RTL: A Software Tool for Customizing Soft Cells for Embedded Applications

Surendra G and S K Nandy
Supercomputer Education and Research Center
Indian Institute of Science
Bangalore 560 012 INDIA
[surendra@rishi. , nandy@]serc.iisc.ernet.in

Paul Sathya
Central R & D
STMicroelectronics
Noida 201 301 INDIA
paul.sathya@st.com

Abstract

Significant reduction in design time for System on Chip (SoC) applications can be achieved through IP reuse. Such a design methodology encourages designers to develop IP blocks that add to a library of soft cells in anticipation of the market trends and meet stringent time to market constraints. However for application specific ICs all the hardware in each IP block may not be used. This gives rise to an opportunity to reduce the number of hardware components in such blocks depending on the application and customizing the IP or soft cell for the application context. In this paper we present a method to automatically detect and remove logic in RTL blocks (soft cells) that are not used by the embedded application thereby reducing area and power. A prototype tool ReDeEm_RTL (Re-Design in Embedded RTL, read as Redeem RTL) has been implemented to remove all redundant logic and its performance although dependent on the type of application, shows that sizable reduction in logic can be obtained.

1. Introduction

Miniaturization of pervasive consumer electronic items has led to the emergence of System on Chip (SoC) solutions. Furthermore the convergence of computers, telephony, internet appliances and wireless products justify the development of SoC based devices. Products based on SoC usually face unique design constraints that may include low power, portability or temperature requirements. Intellectual Property (IP) reuse allows all designers to create new system blocks that can be quickly integrated with others. Using existing IP is a practical solution to managing the complexity of current day electronic designs constrained by time to market considerations. The idea is to create a stable framework into which prequalified IP suited to a particular application can be placed. Often IP which is best suited for an application set will not be available. In such cases an IP which is possibly closest can be used. Statistics have shown that over 25% of the IP used in a complete design or SoC belongs to the same company undertaking that design. The Virtual Socket Interface Alliance (VSI)[1] was institutionalized to support the needs of the industry for design reuse. Virtual Component (VC) is a block that meets the VSI specifications and is used as a component in virtual socket design environment.

VCs can be of three types: Soft, Firm and Hard. Soft VCs are delivered in the form of synthesizable HDL, and have the advantage of being flexible but the disadvantage of not being as predictable in terms of performance (ie. timing, area, power)[1]. This is because Soft VCs are not optimized in structure and topology for performance and area through floorplanning or routing.

Area and power are two of the key factors to be considered during the design of an embedded system. It has been found that multiplexer networks account for more than 40% of power consumption in control-flow intensive circuits[2] with clocks and registers also accounting for significant power[3]. Consequently a lot of research has gone into reducing the power in control-flow intensive circuits. Concepts such as resource sharing[4], clock gating[5], glitch reduction[2] and more recently control generated clocking[6] have been proposed. The concepts of area and power reduction in the IP domain is relatively new and we confine our attention to this domain in the paper. The article in [7] gives a good insight to IP based design.

The reuse of previously created blocks to design Application Specific Integrated Circuits (ASIC) or Application Specific Instruction Processors (ASIP) presents the opportunity for a designer to reduce some logic (within the block) that would not be used by the application. While reprogrammability is desirable, its cost in terms of power consumption and area can be excessive. If the application set is known reasonably well apriori it can be used to create new power and area efficient designs. In this paper we present a method to automatically remove logic that is not utilized by the application in embedded systems made up of RTL components. We assume that a new component block is created by making use of previously designed ones (IP reuse) with the RTL description of each block being available. We have developed a software tool - ReDeEm_RTL that is used to tailor conditional constructs in the RTL blocks to suit the embedded application.

Our method is based on language constructs and can be applied to any logic description. We will refer to the terms "RTL reuse" and "RTL_VC" which are anologous to IP reuse and VC blocks respectively for the rest of the paper. The only difference between an IP block and a RTL_VC is that some Soft VCs may be encrypted while a "RTL_VC" is not. The tool ReDeEm_RTL can be used for logic optimization, for example by a design team which is involved in creating a new component C, making use of already existing components say A and B (in the form of RTL) which were designed by another team. The customization of RTL_VCs is done for a set of applications (hereafter called application set) which belong to a particular class (such as for example DSP). We consider logic optimization carried out for "logic cells" and not "non logic" cells. We assume that the application context is fully characterized in terms of execution traces (test sequences) for the application in a manner that captures both the dynamic control flow and dynamic data ranges. The rest of the paper is organized as follows. In Section 2 we describe our methodology for hardware reduction. Section 3 describes ReDeEm_RTL. Section 4 presents results of applying our methodology to ST7 (a processor developed by ST Microelectronics) and in section 5 we summarize the contributions of this work and set goals for further work.

2. Methodology

In this section we propose a methodology that exploits RTL reuse for logic reduction in application specific embedded architectures. A high level flow diagram of the methodology when applied to the processor core is shown in Figure 1. Processor RTL refers to the RTL description of various components in the processor core while peripheral RTL refers to other components.

The procedure begins with compiling a fully characterized application to obtain a program ROM file. The RTL description of the RTL_VCs is subjected to simulation along with the stimulus to get an initial set of results denoted in the figure by Results1. The same RTL serves as input to ReDeEm_RTL along with appropriate stimuli. The RTL_VCs are modified by ReDeEm_RTL into an equivalent representation depending on the logical constructs and semantics in a manner described in section 2.1. An execution trace of the application is obtained when the modified RTL is subjected to simulation. ReDeEm_RTL reduces the RTL logic based on the trace information obtained. Simulation is again carried out on the reduced RTL and a comparison is made between the results obtained through reduced RTL (Results2) and those obtained by the original RTL (Results1) to verify correctness. Therefore ReDeEm_RTL works in two phases. In the first phase the RTL_VCs are modified to enable collection of statistics during simulation runs and in the second the modified RTL is reduced.

Given a set of RTL components and an algorithm for the embedded application, an optimal match between algorithm and architecture is obtained by following the procedure below:

1. Introduction of stubs to determine code coverage in RTL_VCs.
2. Determining unused RTL statements post simulation.
3. Automatic removal of unused logic in each RTL_VC.

Our algorithm parses the RTL description of the RTL_VCs and determines all those conditional statements which are not executed for a particular application. ReDeEm_RTL incorporates the above mentioned algorithm and automatically reduces the logic in each RTL_VC and also determines the number of times a conditional block statement is utilized. This data can be used to tailor the hardware to suit the application and also incorporate clock gating in sparsely used logic functional units if this has not been implemented in the original RTL_VC. Clock gating is a method in which the clock to a functional unit is turned off when the unit is idle. This reduces the switching activity thereby saving power[5].

2.1. Introduction of stubs for code coverage

The detection of conditional statements involves the parsing of the RTL description of each RTL_VC while looking for conditional constructs. The following represent conditional behaviour in common Hardware Description Languages (HDLs).

- if-then else statements
- conditional signal assignments (concurrent)
- case statements

2.1.1. If-then else statements

For each `if-then else` statement a stub function is added so that it is possible to determine which of the

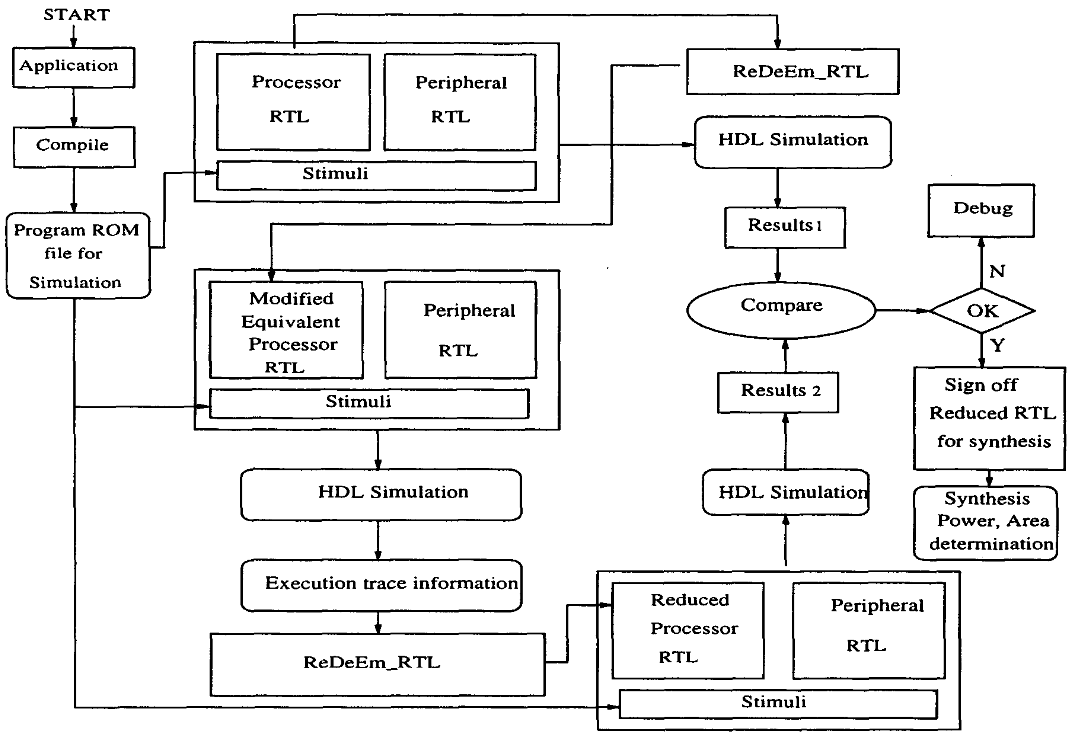

Figure 1. **Flow Diagram**

statements were utilized (executed) by the application. Consider for example the following `if-else` conditional block.

```
if{condition}do_X else do_Y ;
```

This is modified into

```
if{condition}
stub_function();
do_X;
else
stub_function();
do_Y;
```

The stub function when executed writes an unique token into a trace file called **Redeem_trace**.

2.1.2. Concurrent statements

Concurrent signal assignment statements are converted into `if-then else` statements. The stub functions are then introduced following the procedure described above. For example consider a general concurrent statement which assigns to `X` the value of signal `Y` when condition `Z` is true, else assigns `P` if the condition is false. This concurrent statement is executed whenever the value of `Z` changes.

The above statement is written in terms of `if-then else` statements as given below.

```
if {Z is true}
X <= Y
else X <= P
```

2.1.3. Case statements

Stub functions can be inserted directly into the `case` statements. For example

```
case (condition)
condition_value1:
condition_value2:
end case;
```

is converted to

```
case (condition)
stub_function()
condition_value1:
stub_function()
condition_value2:
end case;
```

All other types of conditional and concurrent statements can be realized in terms of `if-else` or `case` statements.

2.2. Determining unused logic

The modified RTL description of the RTL_VCs obtained using the method described in the previous section is subjected to a simulation with the target application set as input. Each conditional statement when executed also executes its associated stub_function(). The stub_function() is a function written in the same High level language as the RTL_VCs. Each stub_function() writes an unique token into **Redeem_trace** when it is executed. At the end of simulation **Redeem_trace** contains information regarding the usage of RTL logic statements in the form of tokens. Tokens not present in **Redeem_trace** indicate the unused RTL statements. The default case condition is not considered as redundant and never removed during the logic reduction process.

2.3. Automatic removal of unused logic

The first step described in section 2.1 modifies the RTL description of the RTL_VCs and fills certain data structures. In the second step simulation is done with the modified RTL and the application set as inputs to ReDeEm_RTL. **Redeem_trace** contains information which is the union of all conditional statements utilized by the application set. Making use of the data collected from the above two steps unused statements are determined. These unused statements are removed from the RTL description of the RTL_VCs thereby reducing logic. The automatic logic reduction process determines the type of conditional statements that were executed and accordingly modifies the RTL.

Some of the pros and cons of the methodology are mentioned below. The optimization of the design for low power and area is done based on a target application set. This naturally takes into account characteristics of programs belonging to that application set. The stimuli plays an important role in determining unused logic. Components in embedded systems are often prone to harsh external conditions. Some statements are exclusively meant to handle such situations. An example would be the return from an invalid state in a FSM to a valid one. Code meant to enable such a return will not be executed in normal conditions. Another example would be statements concerning interrupts which may or may not be executed. Logic other than those which handle abnormal behaviour are optimized to suit the application set. Depending on the application it is possible to limit excepting conditions to the bare minimum relevant to the application.

3. The tool : ReDeEm_RTL

In this section we briefly describe the details of ReDeEm_RTL and how the method described in the previous section is implemented. The first step for the user of the tool is to specify all the RTL_VCs for which logic optimization is required. For example the user may want to apply the customization process only to the core components of the embedded processor and not the peripherals. ReDeEm_RTL parses the RTL files looking for conditional constructs. Concurrent signal assignment statements are converted to if-else statements. Whenever an if statement is encountered during parsing it is pushed onto a stack along with information regarding its location in the RTL file. The contents from the top of the stack are popped and stored in a list when an end if statement is encountered or when the if conditional block terminates. This gives information about how many times nesting occurs in a conditional block as well as locations of nested statements which is used by ReDeEm_RTL during the reduction phase (phase 2). The conditional statements are then modified by adding stubs. The location and type of each conditional statement along with other relevant information are stored in a list.

This modified RTL is subjected to simulation runs with the embedded application set as its input. At the end of the simulation **Redeem_trace** contains the tokens of all executed conditional statements. The list is searched with the token as the key to determine the node and type of statement executed. The number of conditional statements in each conditional block is determined. A conditional block is a set that contains all conditional statements beginning with an if statement and terminated by an end if statement.

A similar logic is applied to case statements. The concurrent signal assignment statements which were converted to if-else statements are restored back to their original form from the data stored in the list.

4. Experimental setup and results - A VHDL case study

We now describe how the tool works for RTL_VCs written in VHDL. The first step described in section 2.1.1 was the introduction of a stub_function(). This is done using the write and writeline functions of the TEXTIO package as described below. Consider the following if-elsif conditional block:

```
if condition1 then
 .
elsif condition2 then
 .
end if;
```

The above RTL description is modified by adding stubs which would indicate whether the conditional statements get executed during simulation runs. The modified RTL description of the above conditional block is shown below.

```
if condition1 then
--stub
write(outline,number1);
-- outline is of type line
writeline(outfile,outline);
--outfile is the trace file.
.
elsif condition2 then
write(outline,number2);
writeline(outfile,outline);
.
end if;
```

Concurrent signal assignment statements are converted into `if - else` statements within a `process` statement with appropriate sensitivity list and the above method is applied. For example:

```
Z <= A when ( X > 4 ) else B;
  is converted to:
process(X)
-- declare variables
if X>4 then
write(outline,number3);
writeline(outfile,outline);
Z<=A;
else
write(outline,number4);
writeline(outfile,outline);
Z<=B;
end if;
end process;
```

The modified RTL is subjected to simulation with the application as input and the trace information is stored in **Redeem_trace**. For above code the `if` statement writes the token `number1` to **Redeem_trace** when it is executed. The `elsif` statement when executed writes token `number2`. The simulation is carried out for a number of application instances belonging to the application set. Therefore at the end of simulation, if **Redeem_trace** contains only the token `number2` it means that only the `elsif` statement was utilized by the application and the `if` statement may be removed.

The third step involves automatic removal of unused RTL statements. The automatic logic reduction process also determines the type of conditional statements executed and accordingly modifies the RTL.

In order to obtain a setup similar to that found in a typical IP reuse environment, we used the core components of the ST7 processor developed by STMicroelectronics and the following associated components: clock generator, bus interface unit, DMA interface, PWM generator, serial communication interface and other standard peripherals. Five different applications were used to test the ST7 processor along with the VHDL files describing the core of the processor. We would like to emphasize here that the applications were typical of the environment where the ST7 processor is used. The ST7 processor is used in motor control, mouse control etc. The test applications used belong to some of the categories mentioned above. The modified RTL obtained during the first phase of processing by ReDeEm_RTL was subjected to simulation with each of the applications mentioned in Table 1 along with appropriate stimuli as input using the Synopsys VHDL simulator. Significant amounts of reduction in RTL statements were obtained by ReDeEm_RTL as indicated by the results in Table 1.

Table 1: Results

Application	Reduction in `if-else` statements	Reduction in `case` statements
Test Application1	41%	42%
Test Application2	10%	15%
Test Application3	49%	52%
Test Application4	51%	57%
Test Application5	50%	56%
Overall	8%	12%

The percentage reduction in `if-else` statements for example is calculated as the ratio of the difference between the number of conditional statements in the original RTL (`if, elsif, else`) and the reduced RTL and the number of conditional RTL statements in the original. The overall reduction is obtained by computing the aggregate of all the above test applications which was 8% and 12% for `if-else` and `case` statements respectively. The functional verification strategy was adopted in order to verify the correctness of the reduced RTL. It must be noted that the results obtained for the test applications should not be taken individually as a measure of reduction. Each of the five test applications used, represent different functionalities which are found typically in the environment in which we used the processor. The above results were obtained when the reduction algorithm was applied only to the core of the processor. Since the algorithm works based on the constructs of VHDL, the logic of other components may be similarly reduced by simply subjecting their RTL descriptions to the first phase of ReDeEm_RTL. The results shown in the table clearly indicate that the amount of logic reduction that can be obtained depends on the application. Preliminary results have shown that a single unused `if` statement in an `if-else` block contributes to atleast 14% area savings in that block. The area reduced depends upon the complexity of the logic which has been determined to be redundant by ReDeEm_RTL. The method adopted although similar in some respects to code coverage analysis differs in the intent. The main intention of code coverage is to find areas of a program not exercised by a set of test cases. Consequently a variety of coverage measures such as statement coverage, decision coverage, condition coverage etc are used. A Soft VC is a predesigned , preverified block which guarentees correct functionality (this is ensured by the designer) as it would have gone through several phases of testing including coverage analysis before being distributed. As a result ReDeEm_RTL is not required to implement other forms of coverage except decision coverage to determine redundant logic.

5. Conclusions and future work

We have described ReDeEm_RTL - a tool for automatic RTL reduction in systems made up of several predesigned components created by different design teams. We would like to emphasize that this tool can be used to optimize any system made up of several RTL_VCs, as it works based on the language constructs with which components are described. We have assumed an optimistic approach to determine unused logic for a target application set as it is based on execution traces. The results indicate that RTL reduction depends upon the embedded application for which the optimization is being carried out while power and area savings depend on the complexity of the redundant logic.

The future work planned, is to validate the functionality at gate level by synthesizing the reduced RTL obtained by ReDeEm_RTL. Cost (Area, Power) and performance analysis of the reduced RTL against the original RTL is also planned.

References

[1] VSI Alliance architecture document, www.vsi.org

[2] Anand Raghunathan,S.Dey, and N.K.Jha, "Register transfer level power optimization with emphasis on glitch analysis and reduction", IEEE Trans on Computer-Aided Design , Vol 18 No.8, August 1999, pp. 1114-1131.

[3] M Srikanth Rao, and S K Nandy, "Controller redesign based register and clock power minimization", Proceedings of IEEE International Symposium on Circuits And Systems, Geneva, May 2000.

[4] T. Kim, N. Yonezawa, J. W.S.Liu, and C.L.Lin, "A Scheduling algorithm for conditional resource sharing - A Hierarchial reduction approach", IEEE Trans on Computer Aided Design, April 1994, Vol 13 No.4, pp. 425-437.

[5] L.Benini, P.Siegel, and G.De Micheli, "Saving power by synthesizing gated clocks for sequential circuits", IEEE Design and Test of Computers, Dec 1994, pp. 32-40.

[6] M Srikanth Rao, and S K Nandy, "Power minimization using control generated clocks", Proceedings of 37th ACM, Design Automation Conference, Los Angeles, June 2000.

[7] Yervant Zorian, and Rajesh K Gupta, "Design and test of core-based systems on chips", IEEE Design and Test of Computers, Oct-Dec 1997, pp. 14-25.