Physical-Aware High Level Synthesis
Congestion resolution for the realization of high-density and low-power

Masato Tatsuoka
SoC Front-End Design Dept.
SoC Design Center
System LSI Company
Fujitsu Semiconductor Limited
Contents

- Fujitsu Semiconductor’s Custom SoC (ASIC) Solutions
- Background
- Current Challenges of High-level Design Flow (such as Routing Congestion)
- Physical-aware High-level Design Flow
- Results of Detection and Reduction of the Routing Congestion
- Summary
Fujitsu Semiconductor’s Custom SoC (ASIC) Solutions
Fujitsu Semiconductor offers "Custom SoC Solutions" for customer's SoC development.

This allows the SoC development of high performance, high quality and low power consumption in a short term.
Custom SoC Solutions: Platform-base Development

Shorter development time by customizing from the base platform

- Shortened development time and reduced re-spin risk by evaluation and verification in advance.

Full custom development:
- HW: Specification design, Logic design, Logic verification, ES manufacturing
- SW: Specification design, Design and verification for periphery of CPU, Cost reduction/period shortening

Platform-based development:
- HW: Specification, Logic design and verification, ES manufacturing
- SW: Specification, Advance evaluation and development, Software development

Application aware platform for Document system and image processing system.
Design Service Cedar™ for SoC Development

Cedar-SPEC Cedar-Specification
Service to realize better quality of LSI specification and extraction of validation items by development process using UML
- improved quality of specification document
- extraction of verification items

Cedar-ESL Cedar-Electronic System Level
LSI verification service by development process with the combination of ESL and Emulation
- Architecture design by performance evaluation model
- Shorter time for S/W development by high-speed model

Cedar-HLS Cedar-High Level Synthesis
High level synthesis service
- Optimization of high level description
- ASIC Hand-Off for high level description

Cedar-PROT Cedar-Prototyping
Proto-type board development service which enables Hardware performance verification
- Hardware performance check
- System verification including S/W

Cedar-EMU Cedar-Emulation
Emulation verification service to shorten verification time to 1/1000
- Performance verification and dissipation power measurement
- System verification including S/W

Cedar-SIM Cedar-Simulation
Verification service of simulation base that uses random technique and assertion technique
- Interface verification by inserting the assertion
- Encompassing verification by random method
- Function verification by Cedar-SPEC verification items

Verification Assets (reuse/inherit)

System design
Architecture design
Block design
Software design

Performance Analysis
Block verification
System verification
ES development
Prototype board development
System building up
Block implementation

Cedar : C-based Effective Design-Flow Apply to Real Design

Copyright 2014 FUJITSU SEMICONDUCTOR LIMITED
Motivation (Background)

- Routing congestion issue becomes remarkable with the recent process technology scaling
  - Routing congestion tends to occur in advanced process node
  - PPA (Performance / Power / Area) is degraded when routing congestion occurs
  - Establishment of the synthesis tool and design flow which takes routing congestion into account is important

- High-level synthesis (HLS) technology becomes widely used
  - Application range expands from a single module to sub-system, from several hundred thousand gates to several tens of millions gates
  - The QoR is not guaranteed, since the algorithm level description doesn't care the generated circuit
    - Routing congestion tends to happen in HLS compared with the RTL design
  - High-level synthesis technology that takes into account the routing congestion has not been established yet

- Establishment of the high-level design flow that takes into account the routing congestion is desired
  - We tried the high-level design flow that takes into account the routing congestion in cooperation with Cadence
  - We applied it into the actual design
Causes of the Routing Congestion

1. Routing congestion caused from the floor plan
   - For instance, improper placement of memory block causes routing congestion
   - This could be eliminated by adjusting the memory block placement under the discussion with the logic design team
   - However, it is difficult to identify this issue until the actual layout is executed

2. Routing congestion caused from the logical structures
   - For instance, the logic design such as big data selector of the size causes the routing congestion
   - It is possible to identify this issue even without the actual floorplan information
   - This issue should be solved during the logic design phase

Multiplexer(MUX)/Demultiplexer(DMUX)
Current Challenges of High-level Design Flow
Conventional High-level Design Flow

Since front-end and back-end design are completely separately performed, the routing congestion is not detected until the back-end process started.

Front-end

Back-end

(1) Algorithm development
- C/C++
- SystemC

(2) Code rewrite
- C/C++
- Synthesizable C'

(3) High-Level Synthesis
- HLS

(4) Logic synthesis
- Logic Synth. Tech Lib

(3) RTL from High Level Synthesis
- RTL Style Check
- RTL Rule Set

(4) Netlist from logic synthesis
- RTL-Sim/(FPGA)

Quality degradation

Congestion
Benefits of High-Level Synthesis (HLS)

The high abstraction description of SystemC/C++ enables the model similar to the algorithm.

Without changing the source code, it can select hardware implementation options by commands and constraints.

Realization of excellent PPA (Performance/Power/Area) by automatic-insertion of latency, generation of state machine, and resource/register sharing.

Compared to conventional RTL, HLS shorten the time of logic design of hardware very much, but ...
Challenges in HLS and Routing Congestion

- QoR is strongly dependent on the coding style
  - HLS tends to generate route congested circuit than RTL design
  - Especially for the beginner of HLS (even the expert of RTL design)

- Architecture selection is also a cause of the routing congestion
  - If the circuit image is unclear, the problem becomes more serious
  - Automatic architecture selection of the tool also promotes the complexity of the problem

- Aggressive resource & register sharing may also cause routing congestion
  - In order to reduce the area, the designer tends to seek maximize the resource sharing

- However, the current design flow does not address the routing congestion
Routing congestion occurs in this example from an actual design

Well, what is the problem?

SystemC code

```c
sc_in<int> in;
sc_in<int> in2[16];

int array [16];
int a, b, c;

void init() {
    a = 0;
    b = 1;
    c = in.read();
    LOOP: for (i = 0; i < 16; i++)
    {
        array[i] = in2[i].read();
    }
}
```

CtoS command:
Directed to “break the loop”.
So, the input values are substituted for the array element by taking 16 cycles.
Index Access of Array

- Large DEMUX / MUX generated for accessing the array
- HLS tends to generate route congested circuit than RTL design

```c
sc_in<int> in;
sc_in<int> in2[16];

int array[16];
int a, b, c;

void init() {
    a = 0;
    b = 1;
    c = in.read();
    LOOP: for (i = 0; i < 16; i++) {
        array[i] = in2[i].read();
    }
}
```

(1) 16x32 DEMUX generated

(2) 16x32 MUX generated
Physical-aware High-level Design Flow

Introduce the routing congestion detection and the improvement phase into the front end design, and eliminate the routing congestion caused from the logic design in early stage.
Results of Detection and Reduction of the Routing Congestion

Reduced both Routing Congestion and Area
Improvement case 1: detection of routing congestion

Parameters for synthesis

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>70%</td>
</tr>
<tr>
<td>CtoS Clock Freq.</td>
<td>200MHz</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.0 (square)</td>
</tr>
</tbody>
</table>

Execution result

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing congestion</td>
<td>H: 8.55% V: 4.28% Too Large!</td>
</tr>
<tr>
<td>Long Path violations (50 stages)</td>
<td>4,786</td>
</tr>
<tr>
<td>Routing congestion MUX number</td>
<td>8,620</td>
</tr>
<tr>
<td>Total Length ($\mu$m)</td>
<td>15,816,331</td>
</tr>
<tr>
<td>RC Area (CA)</td>
<td>42,697,674</td>
</tr>
</tbody>
</table>

Routing congestion in the vertical direction: V
Routing congestion in the horizontal direction: H

Circuit quality isn’t good
Identify the Location of Congestion

Congestion MUX Reports

4.'memwrite_STR_A_ln102'
   ...
   ...

6.'memwrite_STR_B_ln103'
   ...
   ...

8.'memwrite_STR_A_ln104'
   ...
   ...

switch(idx) {
    case 0: start = 0;
           end = 1;
           break;
    case 1: start = 1;
           end = 5;
           break;
    ...
}

...

sc_uint<5> i=start;
BREAK_LOOP:
do{
    102: STR.A[i] =0;
    103: STR.B[i] =0;
    104: STR.C[i] =0;
    ...
    i++;
}while(i<end);

...
Code Modification for Congestion Reduction

Before

```cpp
switch(idx) {
   case 0: start = 0;
           end = 1;
           break;
   case 1: start = 1;
           end = 5;
           break;
   ...
}
```

```cpp
... sc_uint<5> i=start;
... BREAK_LOOP:
   do{
       STR.A[i] =0;
       STR.B[i] =0;
       STR.C[i] =0;
       ...
   i++; }while(i<end);
...
```

After

```cpp
switch(idx) {
   case 0: start = 0;
           end = 1;
           ...
}
```

```cpp
... sc_uint<5> i=start;
... UNROLL1: for (i = start; i <= end_pu; i++) {
  STR.A[i] =0;
  STR.B[i] =0;
  STR.C[i] =0;
  ...
} break;
```

```cpp
... UNROLL1: for (i = start; i <= end; i++) {
  STR.A[i] =0;
  STR.B[i] =0;
  STR.C[i] =0;
  ...
} break;
```

Remove “Demux” for array accesses

Loop Unroll

Loop Break
## Congestion Improvement Results

**Improvement 1:** Only 3 places corrected with low effort (Quick correction)

**Improvement 2:** Several places are corrected with high effort (Deep knowledge of the circuit & correction strategy required)

<table>
<thead>
<tr>
<th>Item</th>
<th>Original</th>
<th>After improvement 1</th>
<th>After improvement 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing Congestion</td>
<td>H: 8.55% V: 4.28%</td>
<td>H: 1.79% V: 2.95%</td>
<td>H: 0.17% V: 0.81%</td>
</tr>
<tr>
<td>Long Path Violations</td>
<td>4,786</td>
<td>76 (-40% ↓)</td>
<td>472 (-42% ↓)</td>
</tr>
<tr>
<td>Number of MUXs</td>
<td>8,620</td>
<td>5,158 (-40% ↓)</td>
<td>4,982 (-42% ↓)</td>
</tr>
<tr>
<td>Total Length</td>
<td>15,816,331</td>
<td>15,415,309 (-2% ↓)</td>
<td>13,594,980 (-14% ↓)</td>
</tr>
<tr>
<td>RC Area (CA)</td>
<td>42,697,674</td>
<td>29,955,645 (-29% ↓)</td>
<td>33,149,970 (-22% ↓)</td>
</tr>
</tbody>
</table>
Routing congestion causes the quality loss of the design.

Routing congestion wasn't taken into account in the conventional High-level Design Flow.

We proposed a new High-level Design Flow which takes routing congestion into account.
- Introduced the detection and improvement steps of the routing congestion into the front end design phase.

We applied this technology into the actual SoC design.
- Routing congestions detected as expected.
- Confirmed the correlation of quality degradation in pre- and post-layout.

We'll continue the improvement of the routing congestion, and confirm the result of the post-layout.
Expectations for the tool

- Productization of Physical-aware High-level Synthesis tool as soon as possible
  - More comprehensible report for identifying the location of the routing congestion
  - Introduce the command for the improvement strategy of the routing congestion
  - Registers sharing, resources sharing, and scheduling should be performed, with taking the routing congestion into account as well
  - **Applicable without any change in the SystemC code**
shaping tomorrow with you