Optimization of Behavioral IPs in Multi-Processor System-on-Chips

Yidi Liu and Benjamin Carrion Schafer#
Department of Electronic and Information Engineering
b.carrionschafer@polyu.edu.hk#
Outline

• High-Level Synthesis 101
• C-based SoC
• Target Architecture
• Motivational Example
• Behavioral IP (BIP) optimization flow
  • Pre-Step : HLS DSE
  • Step 1: SoC generation
  • Step 2: System files generation
  • Step 3: HLS and Cycle-accurate simulation
  • Step 4: BIPs Optimizations
• Experimental Results
• Summary and Conclusions
int A, B, C, D;
int E, F;
main()
{
    int x;
    x = A + B;
    E = x * D;
    F = (B + C) * x
}

Const add32s : 1
mul32s : 1

Clock step1
Clock step2
Clock step3
1/freq

D A B C
Add #1 Mult #1

Allocation
Freq
Delay Area

Scheduling

Binding

Add #1 Mult #1

1/freq

Benefits of HLS: Automatic Alternative Architecture Generation

Behavioral Description in C

```c
char A,B,C,D;
char E,F;
main()
{
    char X;
    X = A + B;
    E = X * D;
    F = (B + C) * X;
}
```

**RTL**

1 cycle
Delay:2T

FU constraints

```
S0 -> S1
S1 -> S2
```

3 cycles
Delay:1T

```
+ : 2
* : 2
```

```
+ : 1
* : 1
```
Micro-Architectural Design Space Exploration

- Pareto-optimal (dominating) designs
- 3 main exploration knobs:
  - Synthesis attributes (pragmas inserted in source code)
  - Global synthesis options
  - FU number

Area

Latency

Non optimal designs

Optimal designs

Efficient frontier
C-Based SoC

- Commercial Tools provide bus generators (AHB/AXI)
  - Inputs:
    1. Masters
    2. Slaves
    3. Arbiter type (e.g. fixed, round robin)
    4. Memory map
  - Outputs
    1. Synthesizable C code for bus and bus interface

- After HLS of the entire system → cycle-accurate model is generated

```c
#define AMBA_AHB {
    width address = 32;
    width data = 32;
    module master = {MA, MB};
    module slave = {SA};
    mode arbiter_rule = RoundRobin;
} bus1;

module AMBA_AHB_MASTER {
    mode burst = Enable;
    mode data transfer = Direct;
    mode clock = Enable;
    mode reset = Enable;
} MA, MB;

module AMBA_AHB_SLAVE {
    mode burst = Enable;
    map address = 0x1000ff00-0x1000ffff & 0xffffffff;
} SA;

int i;
for (i = 0; i < DSIZE; i++) {
    abc[i] = i;
}
CBM_burst_write(0x1000ff00, abc, DSIZE);
```
Target Architecture

- Heterogeneous MPSoC
- Memory mapped shared bus
- BIPs instantiated loosely coupled HWAccs slaves
  - Each BIP Optimized for performance separately BUT:
    - Wait for master to start communication
    - Need to wait for arbiter to pass control of bus to return data
Motivational Example

• Observation 1: Different Task mappings for the same system lead have the same area, but different performance.

• Observation 2: There is a design $D_M$ (fit), with same performance, but smaller area than using fastest micro-architecture designs

![Diagram showing area and performance for different mappings.](image)

**Objective:** Find the smallest micro-architecture of each BIP mapped as a HWacc slave for the fastest SoC configuration ($D_M$ (fit))
Proposed Optimization Flow

- **Pre-Step**: HLS DSE - for each BIP in the system.

- **Step 1**: SoC Generation. Generate systems with 1-N masters and different tasks’ mappings using fastest BIP micro-architecture.

- **Step 2**: System Generation. Reads bus definition file and creates synthesizable SystemC files of entire system.

- **Step 3**: Cycle-accurate Simulation. HLS on each process, generates cycle-accurate model, compile (g++) and execute.

- **Step 4**: BIP Optimization. Read cycle-accurate timing report of each slaves’ idle time and select smallest micro-architecture based on slack.
Pre-Step : HLS DSE

• 3 main *knobs*
  • Synthesis attributes (pragmas inserted in source code)
  • Global synthesis options
  • FU number ➔ Used in this work

1. Synthesize each BIP without FU constraint file (FCNT):
   • HLS tool allocates as many FUs to fully parallelize description
   • Generates FCNT indicating the type and number of FUs

2. Reduce FCNT file by X % until a single FU of teach time is reached
Step 1 : SoC Generation

- Given $S$ Slaves $\rightarrow$ Generate SoCs for 1 to $S$ masters
- For each configuration $m$ all possible task mappings
  - Tasks periodically repeating
  - Execution order is not considered $\rightarrow$ Number of mappings follow Stirling number of second kind $S(s,m)$, with $s$=Slaves and $m$=[1,M]masters
- E.g. $M=3$ (masters), $S=4$ (slaves)

$$S(s, m) = \frac{1}{m!} \sum_{i=0}^{m} (-1)^{m-i} \binom{S}{i} i^s$$

<table>
<thead>
<tr>
<th>Mappings</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinations</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
</tr>
<tr>
<td></td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
</tr>
<tr>
<td></td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
</tr>
<tr>
<td></td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
<td>$(1, 2, 3, 4)$</td>
</tr>
</tbody>
</table>
Step 1: SoC Generation con’t

- **Inputs:**
  - BIPs trade-off curves (fastest design used)
  - Bus parameters (AHB/AXI, bus bitwidth, arbiter)

- **Outputs**
  - Bus definition file for bus generator
  - Synthesizable C code for masters and slaves using synthesizable bus read/write APIs
  - Tasks mappings following Sterling number of second kind for each system with unique masters.
Step 2: System Generation

- Commercial HLS tool bus generator called with:
  - bus definition file generated in step
  - Masters and slaves
- Generates synthesizable files for:
  - Top level module
  - Bus
  - Bus interfaces (masters and slaves)
Step 3: HLS and Cycle-accurate Simulation

1. HLS is a single process synthesis method → synthesize each synthesizable process
2. Call cycle-accurate model generator
   • Input: the scheduling result of each process
   • Output: cycle-accurate SystemC model of the entire system
3. Update slaves’ cycle-accurate model to report time when reading, computing or writing data
4. Compile (g++) and execute SoC model
5. Read timing report of each BIP
   • Read, Write, computation and idle time of each BIP
Step 4: Slave (BIP) Optimizations

- Computation latency $L_i = L_{\text{read}} + L_{\text{comp}} + L_{\text{write}}$
- Extract for each BIP smallest idle (waiting) time $W_{\text{min}}$
- New adjusted Latency $L_{\text{adj}} = \text{floor}(L_{\text{comp}} + W_{\text{min}})$
Step 4: Slave (BIP) Optimizations con’t

• Choose micro-architecture with closes smallest latency to new latency

• Re-synthesize and re-simulate the new system with each new micro-architecture
Experimental Setup

- Complex systems based on computationally intensive tasks were formed by grouping individual benchmarks as HWacc
- S2CBench benchmark suite ([www.s2cbench.org](http://www.s2cbench.org))
- Experiments run on Intel dual 2.4 GHz Xeon with 16GBytes of RAM running Linux Fedora release 19
- HLS tool NEC’s CyberWorkBench v. 5.5
- Target technology Nangate’s 45nm’s Opencell
- Target synthesis frequency 100MHz

```
<table>
<thead>
<tr>
<th>Bench</th>
<th>DSE</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5C</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Kasumi</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Interp</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FIR</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Adpcm</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bsort</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Tasks</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Designs</td>
<td>16</td>
<td>19</td>
<td>11</td>
<td>18</td>
<td>26</td>
<td>27</td>
<td>26</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>
```
Experimental Results: Area

- OPT_IP vs. exhaustive search (BF)
- BF tries all possible micro-architectures of HLS DSE result
- Find the micro-architecture of each BIP for the fastest system
- In all cases same throughput within 1% can be achieved
- On average the area is reduced:
  - BF = 17.43%
  - OPT_IP = 13.21% (~5% larger than BF)

<table>
<thead>
<tr>
<th>Masters</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>83.4</td>
<td>84.2</td>
<td>95.8</td>
<td>68.0</td>
<td>69.6</td>
</tr>
<tr>
<td>OPT_IP</td>
<td>83.4</td>
<td>95.1</td>
<td>95.8</td>
<td>68.0</td>
<td>69.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Masters</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>79.5</td>
<td>80.1</td>
<td>80.1</td>
<td>82.1</td>
</tr>
<tr>
<td>OPT_IP</td>
<td>79.5</td>
<td>86.4</td>
<td>87.0</td>
<td>91.3</td>
</tr>
</tbody>
</table>

Note: The table shows area comparison between exhaustive search (BF) and proposed method (OPT_IP) in percentage.
Experimental Results: Running Time

- OPT_IP is on average ~16x faster than BF

<table>
<thead>
<tr>
<th>Bench</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>192</td>
<td>622</td>
<td>297</td>
<td>10,518</td>
<td>1,543</td>
<td>2,752</td>
<td>1,666</td>
<td>5,045</td>
<td>2,779</td>
</tr>
<tr>
<td>OPT_IP</td>
<td>27</td>
<td>15</td>
<td>20</td>
<td>56</td>
<td>45</td>
<td>170</td>
<td>193</td>
<td>828</td>
<td>169</td>
</tr>
</tbody>
</table>
• Presented a method to optimize the micro-architecture of BIPs mapped onto heterogeneous MPSoCs as loosely coupled HWAcc.

• Two main advantages of C-Based VLSI design leveraged in this work:
  1. HLS DSE to achieve micro-architectures of different characteristics.
  2. State of the art HLS tools allow the generate and simulation (cycle-accurate) of entire SoCs.

• Results show that our proposed method leads to good results while being much faster than an exhaustive search.
www.eie.polyu.edu.hk/~schaferb/darclab