Mapping-Aware Constrained Scheduling for LUT-Based FPGAs

Mingxing Tan, Steve Dai, Udit Gupta, Zhiru Zhang

School of Electrical and Computer Engineering
Cornell University
High-Level Synthesis (HLS) for FPGAs

- HLS has become increasingly important to achieve higher design productivity & quality

```
out = ((i1 & i2) ^ i3) ^ (i4 & i5)
```
A Typical HLS Flow

High-level Programming Languages (C/C++, OpenCL, SystemC Java, Matlab, Python ...)

Compilation

Transformations

Scheduling

Resource sharing

RTL generation

if (condition) {
    ...
} else {
    \( t_1 = a + b \);
    \( t_2 = c \times d \);
    \( t_3 = e + f \);
    \( t_4 = t_1 \times t_2 \);
    z = t_4 - t_3;
}
SDC-Based Scheduling

- Scheduling based on system of difference constraints (SDC) formulation [Cong and Zhang, DAC’06]

Let $s_i$ be the schedule variables

- Dependency constraints
  \[
  \langle o_1, o_2 \rangle : s_1 - s_2 \leq 0 \\
  \langle o_2, o_4 \rangle : s_2 - s_4 \leq 0 \\
  \langle o_3, o_4 \rangle : s_3 - s_4 \leq 0
  \]

- Cycle time constraint
  \[
  o_1 \sim o_4 : s_1 - s_4 \leq -1
  \]

- Latency constraints
- Resource constraints
- ...

- Target clock period: 5ns
- Delay estimate: 2ns

Total delay = 6ns > 5ns
SDC-Based Scheduling

- Representing SDC constraints with constraint graph

**SDC constraints**

\[ s_1 - s_2 \leq 0 \]
\[ s_2 - s_4 \leq 0 \]
\[ s_3 - s_4 \leq 0 \]
\[ s_1 - s_4 \leq -1 \]

**Constraint graph**

Price of abstraction? Pre-characterized delay estimation for individual operation is often too **pessimistic** for logic operations
A k-input LUT (k-LUT) can be configured to implement any k-input 1-output combinational logic

- Delay is a constant for all K-LUTs

Cones and cuts are K-feasible when # of inputs ≤ K
Mapping enables more aggressive chaining by packing more operations into each cycle
  - **LUT level** $l(v)$: arrive time in depth of LUTs
Considering Mapping in Scheduling

Target clock period is 5ns, each operation or LUT takes 2ns

Conventional scheduling

Cycle 0

\[ i1 \rightarrow o1 \rightarrow o2 \rightarrow o4 \rightarrow o3 \]

Cycle 1

\[ i2 \rightarrow i4 \rightarrow o3 \]

Estimated delay = 6ns > 5ns, 2 cycles

Considering LUT mapping

Cycle 0

\[ i1 \rightarrow i2 \rightarrow i3 \rightarrow i4 \rightarrow i5 \]

Cycle 1

\[ o1 \rightarrow o2 \rightarrow o3 \rightarrow o4 \]

Estimated delay = 4ns < 5ns, 1 cycle

A better schedule needs to consider mapping!
Scheduling and Mapping Interdependence

HLS

Determine register boundaries
⇒ Prefer mapping information for more accurate delay estimation

Logic Synthesis

Determine LUT mapping
⇒ Typically occurs between register boundaries
MAPS: Mapping-Aware Constrained Scheduling

**Idea**
- Considering mapping in scheduling to enable more aggressive chaining

**Contributions**
- An algorithm that finds the minimum-latency schedule under SDC constraints considering LUT mapping

**Results**
- Significant latency reduction over a range of logic-intensive applications
We introduce *L-values* to represent the integrated scheduling and mapping information.

\[ L_v = (s_v, l_v) \]

**Time step**  
Inter-cycle scheduling information

**LUT level**  
Intra-cycle mapping Information

Goal: find a legal schedule with minimum L-value for each operation
We keep refining the lower-bound of L-values by relaxation
  – A generalization of Bellman-Ford shortest-path algorithm

Step1: **Initialize** the L-value as (0, 0), an obvious lower-bound without considering any constraint

Step2: **Iteratively** improve the L-values as follows until convergency

For each node on constraint graph
  (1) Mapping constraints: choose the best cut with minimal L-value
  \[ f_v = \min_{\forall C \in CUT_v} \max_{\forall u \in C} \{L_u + (0, Delay_v)\} \]
  (2) Scheduling constraints: decide new L-values according to input edges
  \[ g_v = \max_{\forall u \rightarrow v \in E} \{L_u + (Lat_{u \rightarrow v}, 0) + (0, Delay_v)\} \]
  (3) Update the L-value based on mapping and scheduling constraints
  \[ L_v = \max\{f_v, g_v\} \]

Return a legal schedule when the algorithm converges
We use constraint graph (CG) to represent all SDC constraints
[source: Zhang and Liu, ICCAD’13]
Relaxation-Based Labeling

Assuming 3-LUTs and LUT level $\leq 3$

**Step 1:** Initialize the L-value as (0,0) for each node

**Step 2:** Iteratively update L-values by relaxation

**Iteration 1**

**Node D**

Propagate L-values for mapping constraints

$L_B: (0, 0)$

$L_D: (0, 1)$

$L_E: (0, 2)$

Choose the best cut with minimal L-value
Relaxation-Based Labeling
Assuming 3-LUTs and LUT level $\leq 3$

Step 1: Initialize the L-value as (0,0) for each node

Step 2: Iteratively update L-values by relaxation

Iteration 1

Node I

Propagate L-values for mapping constraints

Black-box operation H has only trivial cut

Maximum LUT level is restricted by cycle time

<table>
<thead>
<tr>
<th>Time step</th>
<th>LUT level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_H$: (0, 3)</td>
<td></td>
</tr>
<tr>
<td>$L_I$: (0, 4) (1, 1)</td>
<td></td>
</tr>
</tbody>
</table>

LUT level $\leq 3$
Step 1: Initialize the L-value as (0,0) for each node

Step 2: Iteratively update L-values by relaxation

Iteration 1  ⇝  Iteration 2

Node I

Scheduling constraints:

\[ g_v = \max_{\forall u \rightarrow v \in E} \{ L_u + (Lat_{u\rightarrow v}, 0) + (0, Delay_v) \} \]

Back edge from I to F: maximum latency constraint
F and I must be in the same cycle

\[ L_I: (1, 1) \]
\[ L_F: (1, 1) \]
\[ L_G: (1, 1) \]
\[ L_H: (1, 2) \]
\[ L_I: (1, 3) \]
Relaxation-Based Labeling
Assuming 3-LUTs and LUT level ≤ 3

Step 1: Initialize the L-value as (0,0) for each node

Step 2: Iteratively update L-values by relaxation

Iteration 1  ⇔  Iteration 2  ⇔  Iteration 3

Node F

Mapping constraints: satisfied
Scheduling constraints: F and I are in the same cycle

Every node reaches its minimum legal L-value!

The algorithm converges
Optimality of MAPS Labeling

Proof by induction that MAPS labeling algorithm always maintains the **lower bound** of L-values

**Base Case:** All L-values are initialized as (0,0), which are the lower bound without considering any constraints;

**Induction:** assume iteration \( k \) maintain the lower bound of L-values, => L-values in iteration \( (k+1) \) are also lower bound, because our algorithm only monotonically increases minimal L-values that satisfy a part of the given constraints

Upon convergence, MAPS returns a legal schedule with a minimum L-value for each node
Conventional vs. MAPS schedule

**Conventional schedule**
- 3 cycles; 2 cycle/iteration

**MAPS schedule**
- Total 2 cycles; 1 cycle/iteration
Resource constraints for black-box operations
- e.g. memory port limits, hardened multipliers

Incremental scheduling heuristic
- Legalize the initial solution from the labeling step
- Gradually serialize resource-constrained operations
Experimental Results

▶ Setup
  – A state-of-the-art commercial HLS
    • MAPS is implemented an LLVM pass
    • We leverage the commercial HLS as the back end for RTL generation
    • We use the same commercial HLS tool as baseline
  – Target device: Virtex-7 FPGA with 6-LUTs
    • 5ns target clock period

▶ Benchmarks
  – 3 kernels: XORR, GFMUL, CLZ
  – 8 logic-intensive applications from MiBench and CHStone
    • Communication: CRC, Reed-Solomon decoder (RS)
    • Cryptography: MD5, AES, SHA
    • Scientific Computing: DFADD, Mersenne twister (MT)
    • Machine Learning: Digit recognition (DR)
Latency Reduction

Generate combinational circuits
For XORR and GFMUL kernels

Reduce by up to 60% for CRC
Average reduction: 29%

Latency Reduction

3 kernels

XORR GFMUL CLZ CRC MD5 AES SHA D3FADD MT RS DR

8 real-life applications

5ns target clock period is met for all designs

MAPS significantly reduces latency by enabling more aggressive chaining
Resource Usage Comparison

- **3 kernels**
- **8 real-life applications**

### Average Reduction

- **#LUT:** 9%
- **Average reduction up to 28%**

- **#FF:** 25%
- **Average reduction up to 59%**

Slight resource increase for some apps.
Case Study for Digit Recognition (DR)

Random Sampling of MNIST

| 2 | 9 | 6 | 1 | 3 |
| 3 | 9 | 4 | 0 | 3 |
| 6 | 9 | 4 | 1 | 9 |
| 9 | 5 | 0 | 8 | 5 |
| 8 | 8 | 3 | 5 | 1 |

(a) Binary string in 2D array
(b) Binary image

```
void count_set_bit(bit49 input, bit6 &ones)
{
    for (int i=0; i<49; i++)
        ones += input[i];
}
```

49 input bits

6 output bits

Target clock period = 5ns

Baseline:
7 levels of operations
2 cycles

MAPS:
3 levels of 6-LUTs
1 cycles

23% latency reduction for the entire DR app
Conclusions

- Cross-layer optimizations that integrate different steps of the FPGA flow can enable next leap in QoR improvement for HLS

- MAPS: a mapping-aware constrained scheduling algorithm
  - Elegantly integrate LUT mapping information into scheduling
  - Achieve latency-optimal schedule under SDC constraints
  - Significantly improve performance and reduce hardware resource
THANKS!

QUESTIONS?
Complexity of MAPS Algorithm

▸ Each iteration will traverse each node and edge once
  – Complexity for a single iteration: $O(|V|^K + |E|)$

▸ MAPS labeling converges within at most $D*|V|$ iterations
  – Each iteration will monotonically increase the L-value by at least 1
  – The upper bound of each L-value is $D*|V|$, where $D$ denotes the maximum delay for any edge; $D$ is usually a small constant.

▸ Total complexity of MAPS Labeling is $O(D*|V|*(|V|^K + |E|))$, which is polynomial when $K$ and $D$ are small constants.

MAPS labeling guarantees to obtain a legal schedule with optimal L-value for each node in pseudo-polynomial time.
## Runtime Evaluation for MAPS

### Synthesis time (seconds)

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>MAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>23.0</td>
<td>23.0</td>
</tr>
<tr>
<td>XORR</td>
<td>56.0</td>
<td>64.7</td>
</tr>
<tr>
<td>GFMUL</td>
<td>4.3</td>
<td>11.1</td>
</tr>
<tr>
<td>CLZ</td>
<td>24.0</td>
<td>29.7</td>
</tr>
<tr>
<td>CRC</td>
<td>3.9</td>
<td>11.8</td>
</tr>
<tr>
<td>MD5</td>
<td>15.6</td>
<td>28.8</td>
</tr>
<tr>
<td>AES</td>
<td>20.5</td>
<td>61.9</td>
</tr>
<tr>
<td>SHA</td>
<td>8.9</td>
<td>19.6</td>
</tr>
<tr>
<td>DFADD</td>
<td>9.3</td>
<td>11.1</td>
</tr>
<tr>
<td>MT</td>
<td>36.5</td>
<td>193.5</td>
</tr>
<tr>
<td>RS</td>
<td>23.0</td>
<td>24.6</td>
</tr>
<tr>
<td>DR</td>
<td>44.5</td>
<td>50.5</td>
</tr>
<tr>
<td>Design</td>
<td>Approach</td>
<td>CP(ns)</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>XORR</td>
<td>baseline</td>
<td>2.88</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>2.28</td>
</tr>
<tr>
<td>GFMUL</td>
<td>baseline</td>
<td>2.93</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>1.68</td>
</tr>
<tr>
<td>CLZ</td>
<td>baseline</td>
<td>2.93</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>2.93</td>
</tr>
<tr>
<td>CRC</td>
<td>baseline</td>
<td>2.93</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>2.93</td>
</tr>
<tr>
<td>MD5</td>
<td>baseline</td>
<td>4.39</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>4.24</td>
</tr>
<tr>
<td>AES</td>
<td>baseline</td>
<td>4.78</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>4.44</td>
</tr>
<tr>
<td>SHA</td>
<td>baseline</td>
<td>4.21</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>3.87</td>
</tr>
<tr>
<td>DFADD</td>
<td>baseline</td>
<td>4.81</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>4.80</td>
</tr>
<tr>
<td>MT</td>
<td>baseline</td>
<td>3.96</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>4.03</td>
</tr>
<tr>
<td>RS</td>
<td>baseline</td>
<td>4.23</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>4.30</td>
</tr>
<tr>
<td>DR</td>
<td>baseline</td>
<td>3.70</td>
</tr>
<tr>
<td></td>
<td>MAPS</td>
<td>3.80</td>
</tr>
<tr>
<td>AVERAGE</td>
<td></td>
<td>-29%</td>
</tr>
</tbody>
</table>
Kernel: Xor Reduction for Bit Vector

Target clock period is 5ns, each one-bit addition has 2ns latency

Original Schedule
2 cycles, 4 LUTs

MAPS Schedule
1 cycles, 3 LUTs
Kernel Example: Galois Field Multiplication (GFMUL)

Target clock period = 5ns

Original Schedule
4 cycles, 3 LUTs

MAPS Schedule
1 cycle, 1 LUT
Separate Mapping and Scheduling

- How about performing mapping before scheduling?

**Mapping + Scheduling**

2 cycles per iteration

Total Latency = 2*N for N iterations

**Optimal schedule**

1 cycle per iteration

Total Latency = N for N iterations
Loop-Prioritized Mapping and Scheduling

- How about prioritizing mapping for loops

Mapping + Scheduling
3 cycles, 3 LUTs

Optimal schedule
2 cycles, 2 LUTs
Retiming Based Mapping and Scheduling

- Can we address the problem using retiming?

Mapping + Scheduling
2 cycles per iteration
Total Latency = 2*N for N iterations

Mapping + Scheduling + Retiming
Still 2 cycles per iteration
Word-Level Tracking

- Bit-Level Dependence Tracking

\[ \text{inputs}(C) = \{A, B\} \]

\[ \text{inputs}(C) = \{A_0, A_1, \ldots, B_0, B_1, \ldots\} \]