Implementing Asynchronous Circuits using a Conventional EDA Tool-Flow

Christos P. Sotiriou
sotiriou@ics.forth.gr

Institute of Computer Science,
Foundation for Research and Technology – Hellas – FORTH,
Heraklion – Crete, Greece.

The Arguments for Asynchronous Design

Why implement asynchronous rather than synchronous systems?

- Implementation problems of synchronous circuits.
- Low-power consumption.
- Average-case rather than upper-bound performance.
- No explicit timing assumptions.
- Well-defined communication interfaces.
- Scalable and expandable system model.
- Well suited for realising systems-on-a-chip?
The Arguments against Asynchronous Design

But synchronous is common and straightforward…

- Asynchronous control circuit design is hard.
- How do I design asynchronous circuits?
- Industry is skeptical and accustomed to synchronous design.
- Commercial tools are targeted towards synchronous circuits.
- Asynchronous tools are research-oriented and difficult to integrate to commercial tools.

C. P. Sotiriou, ICS-FORTH 2nd ACiD-WG Workshop

Realising a fully-asynchronous EDA Flow

Aims of this Research

- “standardise” an asynchronous EDA flow.
- Use commercial EDA tools as much as possible to ensure commercial-quality.
- Deviate from synchronous EDA flows as little as possible.
- Such a flow could act as a catalyst for the adoption of asynchronous design.
- Such a flow will enable the implementation of asynchronous IP blocks and asynchronous SOCs.

C. P. Sotiriou, ICS-FORTH 2nd ACiD-WG Workshop
Typical EDA Tool Flow Features

*Essential Steps*

- **Synthesis:**
  mapping an HDL specification to generic gate netlist.
- **Mapping:**
  mapping the generic gate netlist to a technology.
- **Timing Analysis:**
  measuring and verifying a circuit’s timing.
- **Physical Design:**
  converting the technology-mapped netlist to a physical layout.

Using Conventional EDA Tools for Asynchronous Design

*Requirements for a fully-automated Asynchronous Flow*

- Asynchronous circuit synthesis from an (HDL) specification.
- Asynchronous circuit constraint derivation from an (HDL) specification.
- Use of conventional synthesis tools for asynchronous circuit description, technology mapping and optimisation.
- EDA Tool-compliant constraint specification.
- Breakdown of circuit parts into constrainable blocks (boundaries specified by user HDL code).
- Time-driven Physical block design for constraint fulfillment.
- Support for top-level constraints.
Asynchronous EDA Flow

Experiments with Conventional EDA Tools

*How suitable are they for Asynchronous Design?*

- support generic technology netlists for synthesis (*Synopsys GTECH, Ambit ALIB*).
- timing analysis for asynchronous circuits should consider *true path delays* and not combine the arrival times of intersecting paths.
- synthesis constraints must exploit hierarchy, as feedback can confuse the synthesis tools when constraining paths.
- constraints must be specified at synthesis (tool-specific) and post-synthesis (*GCF, SDF*).
Experiments with Conventional EDA Tools

How suitable are they for Asynchronous Design?

- asynchronous paths are often edge-sensitive, however timing analysis at synthesis does not support this.
- timing analysers (e.g. Pearl) support edge-sensitive paths.
- irrelevant paths must be blocked for timing analysis.
- relative path constraints are NOT as yet supported.
- GCF constraints support absolute true path delays but only on top level design pins.
- consistent signal naming throughout the flow is a problem.

Using Conventional EDA Tools for Asynchronous Design

Our Approach for Control Block Design

- use the direct-mapped approach to implement both gate-level synthesis and layout synthesis (effectively 2 flows).
- synthesis “glue” tool:
  (a) maps an asynchronous specification to a generic technology library (GTECH, ALIB).
  (b) constrains critical paths of control circuits at synthesis stage and generates constraints for physical design tools (GCF).
- constraints propagate into the physical design tools.
- time-driven placement and routing is used to guarantee constraints are fulfilled.
Types of One-hot Encoded AFSMs

Using Set-Reset Flip-Flops: Using CMOS Complex Gates:

- suitable for gate-level EDA.
- suitable for transistor-level EDA.

Advantages of one-hot circuits

- no general races between state variables; no circuit analysis is required.
- circuit independent delay constraints (implementation dependent).
- compositional and regular; easier to place and route.
- fast; small input/output delay.
- testable; can be converted to shift registers.
- allow for the expression of parallelism through parallel active state paths.
One-hot Gate-level AFSM Delay Constraints

- Critical delays for a state:
  \( \Delta(s) \), set input delay, \( \Delta(r) \), reset input delay,
  \( \Delta(sp) \), set path delay, \( \Delta(rp) \), reset path delay.

C. P. Sotiriou, ICS-FORTH

One-hot Gate-level AFSM Delay Constraints

*Conditions for correct operation:*

(I) One-hot critical race between two states does not occur.
- identical NAND gates ensure delay-insensitive operation.
- asymmetric NAND gates can cause malfunction.
- following delay condition will ensure correct operation:
  \[ \Delta n(rp) + \Delta n(sp) > 2\Delta(s). \]

(II) One-hot critical race between three or more states does not occur.
- following delay condition will ensure correct operation:
  \[ \Delta n(sp) + \Delta n(rp) > \Delta n-1(rp). \]
Exploiting Hierarchical Constraints

*State Flip-Flop with system reset in GTECH*

- Using this structure and making feedback external, the `srlatch` entity may be constrained appropriately.

Asynchronous EDA Flow Example (Manual)

*Feasibility Demonstrator*

- Evaluation circuit was described and constrained manually.
- Then, automatically placed and routed using conventional (connection-driven) and time-driven EDA algorithms.
- Evaluation circuit was a 5-state, 32-bit elastic asynchronous pipeline, a typical asynchronous circuit.
- Pipeline stages implement high-performance, 4-phase fully-decoupled latch controllers.
- Target technology was 0.18μm UMC/VST.
Asynchronous EDA Flow Example

**Evaluation Circuit:**

**Block Diagram:**

**Control AFSM:**

---

C. P. Sotiriou, ICS-FORTH

---

Asynchronous EDA Flow Example

**Part of GTECH description:**

```verbatim
... module stateackin(sreset, idle, reqin, nwaita, ackin, nackin);
input sreset, idle, reqin, nwaita;
output ackin, nackin;
wire setackin;
GTECH_NAND2 nand1 (.A(idle), .B(reqin), .Z(setackin))
; ff2in stateff (.set(setackin), .reset(nwaita), .q(ackin),
 .qbar(nackin), .sreset(sreset));
endmodule
...
```

---

C. P. Sotiriou, ICS-FORTH

2nd ACiD-WG Workshop
Asynchronous EDA Flow Example

Part of GCF Constraint File:

...  
// Set paths A(sp) for all states //  
(path_delay 0.20 9999 0 9999 ((from idle) (to ackin)))  
(path_delay 0.17 9999 0 9999 ((from ackin) (to waita)))  
...  
// Reset paths A(rp) for all states //  
(path_delay 0.23 0.37 0 9999 ((from nackin) (to nidle)))  
(path_delay 0.20 0.45 0 9999 ((from nwaita) (to nackin)))

Evaluation Circuit - Layouts:

Flat Layout:  
Hierarchical Layout:

C. P. Sotiriou, ICS-FORTH  
2nd ACiD-WG Workshop
Asynchronous EDA Flow Example

Range of Timing Violations in TDD and non-TDD mode:

However, the non-TDD design is correct and performs better!

Non-TDD Critical Delays:

<table>
<thead>
<tr>
<th>State</th>
<th>$\Delta_{cl}(\text{up}) + \Delta_{el}(\text{up})$</th>
<th>$\Delta_{cl}(\text{up})$</th>
<th>$\Delta_{el}(\text{up})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ackin</td>
<td>0.22</td>
<td>0.04</td>
<td>0.09</td>
</tr>
<tr>
<td>waita</td>
<td>0.19</td>
<td>0.04</td>
<td>0.11</td>
</tr>
<tr>
<td>wait1</td>
<td>0.25</td>
<td>0.10</td>
<td>0.09</td>
</tr>
<tr>
<td>idle</td>
<td>0.22</td>
<td>0.04</td>
<td>0.09</td>
</tr>
</tbody>
</table>

TDD Critical Delays:

<table>
<thead>
<tr>
<th>State</th>
<th>$\Delta_{cl}(\text{up}) + \Delta_{el}(\text{up})$</th>
<th>$\Delta_{cl}(\text{up})$</th>
<th>$\Delta_{el}(\text{up})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ackin</td>
<td>0.42</td>
<td>0.12</td>
<td>0.18</td>
</tr>
<tr>
<td>waita</td>
<td>0.31</td>
<td>0.08</td>
<td>0.23</td>
</tr>
<tr>
<td>wait1</td>
<td>0.56</td>
<td>0.12</td>
<td>0.18</td>
</tr>
<tr>
<td>idle</td>
<td>0.39</td>
<td>0.08</td>
<td>0.30</td>
</tr>
</tbody>
</table>

* despite the fact that absolute delays control P&R effectively, the non-TDD flow produces a faster circuit.

Why?

- Connection-driven algorithm packed circuit; synthesis constraints proved sufficient for this order of circuit size.
- Absolute timing constraints were too conservative due to the synthesis library being too conservative in estimating connection delays.
AFSMGEN – Automatic AFSM Synthesiser

C. P. Sotiriou, ICS-FORTH
2nd ACiD-WG Workshop

AUTOCONSTRAIN – Automatic Constraint-Aware AFSM Technology Mapper

C. P. Sotiriou, ICS-FORTH
2nd ACiD-WG Workshop
Using Conventional EDA Tools for Asynchronous Design

Conclusions and Future Work

• Possible to implement asynchronous circuits using conventional EDA tools.
• Need EDA vendors to incorporate relative path delay constraints as in the SDF v3.0 specification.
• Would like to collaborate with EDA vendors to standardise asynchronous flow(s).
• What about datapaths? Fixed delay, completion detection.
• What about Verilog, VHDL-> circuit mapping? Need definitions. [Blunno and Lavagno’s work on automatic synthesis of micropipelines.]
• What about system-level view and optimisation? [Shang, Xia and Yakovlev’s work on Direct Translation Synthesis.]

Implementing Asynchronous Circuits using a Conventional EDA Tool-Flow

Summary

• Use one-hot AFSM realisations to implement asynchronous control circuit synthesis.
• By implementing a synthesis glue tool, VHDL FSM specifications may be mapped to one-hot circuits.
• Glue tool integrates support for automatic constraint generation.
• Delay constraints ensure correct circuit operation.
• Delay constraints enable time-driven placement and routing.
• AFSMGEN and AUTOCONSTRAIN tools are only a first step.
• Additional tools are required: GCF generator and other tools for mapping non-FSM VHDL code, sequential statements and datapaths.