A Pseudo-Synchronous Implementation Flow for WCHB QDI Asynchronous Circuits

Yvain Thonnart, Edith Beigné, Pascal Vivet
CEA-LETI, Minatec, Grenoble, France

Async’2012, May 8th 2012
DTU, Copenhagen, Denmark
Asynchronous circuits

with synchronous CAD tools?

- A handcrafted piece of art
  - Entangled uneven loops
  - Requires minute attention to detail
  - Very valuable for specific needs
  - But very expensive design time

- A powerful heavy machinery
  - Backed-up by big EDA companies
  - Obsessed about clocks
  - Scared of loops

- Pseudo-synchronous implementation
  - “Mass-produced”
  - Much cheaper design time
  - Can run fast, nevertheless!

Trick the chain link model
Outline

- Asynchronous circuits with synchronous CAD tools?
- Pseudo-synchronous models for C-elements
- Pseudo-synchronous circuit implementation
- Benchmarking against asynchronous implementation
- Real-world implementations
- Conclusion & perspectives
**DIMS WHCB pipeline**

**combinational loops & optimization**

- **Performance is given by the loops cycle times**
  - Design optimization needs to constrain those loops
  - Synchronous CAD tools can’t handle them
  - need to **cut the loops** in the timing graph & **constrain** loop segments

- **Where to cut for a systematic approach**
  - in the WCHB C-elements: the ones gathering forward and backward data (they must be Resetted)
Asynchronous Implementation: cost & flaws

- Resulting timing constraints:
  - For each WCHB C-element in the cell library, disable timing arcs to cut the loops
    - `set_disable_timing 'C_element' from 'in' to 'out'`
  - For each path segment between two WCHB C-elements, specify a target maximum delay
    - `set_max_delay from 'C/elt/inst1/out' to 'C/elt/inst2/in' 0.5ns`

- Limitation: The WCHB C-elements themselves are not optimized
  - Minimal or no drive adaptation of cells depending on cell load
  - No consideration on signal slope on path end
  - Cells can be moved back and forth during placement
    - Synchronous CAD tools do not manage asynchronous path ends correctly

- Use pseudo-synchronous models for WCHB C-elements
  - to cut timing loops without disabling timing arcs
  - to improve tool control over path ends
Pseudo-synchronous circuit timing paths

- Loops are cut naturally at pseudo-synchronous C-elts
  - No need to disable a timing arc
  - Creates 2 kinds of paths in WCHB pipeline:
    - forward paths
    - backward paths

→ How to derive pseudo synchronous models ?
→ How to constrain resulting paths ?
Asynchronous .lib characterization

- .lib files in Liberty format to model cell timing arcs
  - As a function of input transition times and output capacitance
  - 4 values per arc: rise delay, fall delay, rise transition, fall transition

![Diagram of asynchronous .lib characterization](image)

- **rise_delay(A→Z):**
  - 30ps 120ps 200ps
  - 80ps 160ps 250ps
  - 130ps 210ps 300ps

- **A input transition**
  - 10ps 80ps 200ps

- **Z output capacitance**
  - 10fF 40fF 100fF

- **rise_tran(A→Z):**
  - 12ps 80ps 320ps
  - 20ps 85ps 320ps
  - 28ps 90ps 320ps

- **A input transition**
  - 10ps 80ps 200ps

- **Z output capacitance**
  - 10fF 40fF 100fF

When B=1 and Reset inactive
Pseudo-synchronous .lib derivation

- C-element is modeled like a synchronous flip-flop
  - Reset pin is used as a dummy clock input
  - New arc uses first row of A→Z arc, old arcs are turned to setup checks

\[
\begin{align*}
\text{Clk (was Reset)} & \quad \text{A} & \quad \text{B} & \quad \text{Z} \\
\text{A} & \quad \text{setup rise constraint} \\
\text{Clk} & \quad \text{rise_delay} \\
\text{Z} & \quad \text{rise_tran}
\end{align*}
\]

- \( \text{rise_delay(Clk→Z)}: \)
  - 0ps 10ps 20ps 30ps 40ps 50ps 60ps 70ps 80ps 90ps 100ps
  - 10fF 40fF 100fF

- \( \text{setup_rise(A→Clk)} \):
  - A input transition
  - 0ps 10ps 50ps 80ps 100ps 200ps
  - computed as diff. between 1st column of previous rise_delay(A→Z) and new rise_delay(Clk→Z)

- \( \text{rise_tran(Clk→Z)}: \)
  - 12ps 80ps 320ps
  - 20ps 85ps 320ps
  - 28ps 90ps 320ps
  - 10fF 40fF 100fF

- \( \text{setup_rise(B→Clk)} \):
  - Idem with previous B→Z
Simple pseudo-synchronous constraint

- Declaring a clock on the reset signal constrains all paths to a given “dummy” period
  - Actual asynchronous cycle time given by biggest sum of 2 fwd + 2 bwd delays on the loops (for token+bubble)
    - as bad as 4x dummy target period
    - often less (2x-3x) as no hold fixing is done

- Dummy clock period limitation:
  - Logic depth can be different on each path
  - Relaxes all paths to worst path length
  - Actual throughput not optimal when forward and backward logic are not balanced (on most critical local loop)
  - Actual forward latency can be really sub-optimal (given by sum of fwd delays)

- What about over-constraining the design?
  - Negative slack is not a big deal for implementation, circuit is QDI after all!
  - But over-constrained paths will distract the optimization kernels...
Refined pseudo-synchronous timing constraints

→ Use dummy clock declaration to identify paths, not to constrain design with a given period
  - Declare clock to break loops, with any period (e.g. 0ns)
  - Override delays on all paths with reg2reg set_max_delay constraints
    set_max_delay 0.23ns –from C/elt/inst1 –to C/elt/inst2
    (no pins given → preserve all arcs inferred by clock declaration)

→ Resulting constraints very similar to asynchronous ones, but with no timing arc disabled
  → Better control on timing paths for optimization tools
  → Leverage on all existing asynchronous STA methods to predict performance
**WHCB isochronic forks handling**

- **Green fork needs no isochronic assumption**
  - Both branches are acknowledged by protocol (C-element on point of reconvergence)

- **Red forks should be isochronic (or relaxed)**
  - Only one of the branches is acknowledged (reconvergence on a combinational gate)
  - **BUT**
    - they always occur at path ends (previous logic is shared)
    - Shortest adversary path goes through 2 C-elements and at least 1 inverting bwd logic
    - Constraining paths through the fork for shortest possible delays (with refined ‘set_max_delay’ constraints) also balances any buffer tree needed at the fork
  - \(\Rightarrow\) Adversary path isochronic hypothesis is easily met
Pseudo-synchronous implementation flow

< Your preferred asynchronous synthesis method here >

Source

Map & Opt

Place & IPO

Route & IPO

Delay Calc

Final sim.

Tape-out

PreCTS.sdc

dummy.ctsspec

postCTS.sdc

Async.lib

PSync.lib

Reset=clk

C

Reset

C

SDF

DRC, LVS…

GDS

SPEF

PSyncIP.lib

Netlist.final.v

Async.lib

PSync.lib

Source

Map & Opt

Place & IPO

Route & IPO

Delay Calc

Final sim.
Linear pipeline case study

Physically implemented & optimized with different strategies

- Implemented down to layout with Cadence SoC Encounter
- STMicro 65nm LP technology
- Very narrow floorplan 20µm*600µm to model a long NoC link

- Instantiated 4x to inject the 4 different input values on each MR4
Timing constraints strategies

**Asynchronous modeling**

*combinational loops broken at C-elements inputs.*

- **zero-delay target:**
  - ‘set_max_delay 0’ on all paths

- **zero slack:**
  - iterations on place-and-route flow adjusting per path
    ‘set_max_delay’ values until implementation reports final slack of 0ps.

- **-40ps slack:**
  - same as above, but stop iterating as soon as final negative slack is lesser than 40ps.

**Pseudo-synchronous modeling**

- **zero-delay target:**
  - ‘create_clock Reset -period 0’

- **simple:**
  - ‘create_clock Reset -period N’ with iterations until N cannot be reduced with a final slack of 0ps.

- **zero slack:**
  - ‘create_clock Reset -period 0’, plus iterations on per path
    ‘set_max_delay’ values until implementation reports a final slack of 0ps.

- **-20ps slack:**
  - same as above, with a 20ps target.
Benchmarking results @tt65_1.2V_25C

- With asynchronous modeling, disabling timing arcs to break loops at C-elements degrades performance
- Simple and 0 target synchronous are comparable in performance
  - Less iterations for 0 target, but slightly bigger area
- Ad-hoc synchronous constraints give best results
ANoC implementations

- ANoC router made of 6 kinds of WCHB processes
  - 3 per input stage, 3 per output stage
  - Generic data path size
  - Any possible combination of input stages and output stages
- 60 “generic” ‘set_max_delay’ constraints cover all possible arrangements of processes in NoC topology
  - 60 values to refine for zero-slack strategies
- Recent implementation in 3 chips with industrial partnership in 2011/2012
  - 2D-mesh based, in STMicro 65nm LP
  - Req-Resp Master-Slave based in STMicro 32nm and 28nm LP
28nm P2012_CO ANoC synthesis results

- According to dummy period:
  - Area increase up to +30%
  - Cycle time & latency reduction up to -30%
- Ad-hoc pseudo-sync. constraints allow for:
  - Reproducible best performance @ 1280Mflit/s
  - With reasonable area increase by ~20% compared to under-constrained design
MAG3D implementation results

- Technology
  - STMicroelectronics cmos 65nm low-power process

- Implementation strategy
  - Pseudo-synchronous hard-macro for routers
  - Mixed integration on top
    - Synchronous DfT
    - Pseudo-synchronous ANoC links
  - P&R Runtime ~ 17h

- ANoC Area
  - 1M Gate

- Performance
  - @tt65_1.2V_25C
  - 7 routers path
  - ~10 mm links
  - Average throughput: 850 Mflit/s
  - Average latency: 9.81ns
Conclusion

Asynchronous circuits turned synchronous (not really...)

- For the designs ➔ a bit more performance
  - DIMS WCHB circuits are not as bad as you would think, aren’t they?
- For the designers ➔ a systematic approach for loop breaking and design constraints
  - Large asynchronous designs within easy reach
- For the community ➔ a “benevolent” betrayal
  - Don’t banish me, please...
- For the industry ➔ a comfortable well-known CAD environment
  - Energy-efficient off-the-shelf soft IPs
    - OK, they are actually asynchronous, but only if they ask...

➔ But will it work for more than ANoC or DIMS WCHB?
Pseudo-synchronous timing paths in QDI (PCHB/PCFB/RSPCHB...) pipelines

- Up to 5 types of pseudo-synchronous paths instead of 2
  - (+ WCHB like paths for state variable in PCFB)
  - Not necessarily balanced in delays \(\rightarrow\) ad-hoc constraints to be considered, dummy period could be insufficient

- When no Reset input is present on the cells, create and rely on an "internal pin" for dummy clock
  - pin(dummy) \{direction : "internal"; [...]\} in .lib file
  - create_clock –name ‘dummy_clk’ \[$all_dummy_pins_in_design\] in .sdc file

- Blue paths form an isochronic fork for "bubbles"
  - Need special handling to guarantee data deactivation before EN re-activation
timing arcs diversion and timing margin

- Alternatives for relative delay constraint on isochronic fork
  - specify ‘set_data_check’
  - reduce max delay constraints separately on both paths to guarantee there is no positive slack
  - Add security margin to data arcs
    - Compatible with simple dummy clk period constraint
    - Specify margin thanks to dummy clk transition time

![Diagram showing setup rise constraint, rise delay, and rise transition with security margin.]
Many thanks to

- My co-authors for their 9-year contribution & support
- The reviewers for their inspiring feedback
- The audience for your questions?