Practical Advances and Applications of Asynchronous Design

MSSoc 2013
Sept 27, 2013

Peter A. Beerel
University of Southern California
USC Asynchronous CAD/VLSI Group (async.usc.edu)
Flip-flops (aka registers/latches)
- Memory elements that store “state” of system

Combinational Logic
- Performs logical functions on data (e.g., add, mult, etc…)

Clock
- Periodic square wave that controls update of memory elements
- Flip-flops update value on rising edge of clock
Trends: Synchronous Design Challenges

Process variability is increasing
- Transistors becoming less and less ideal
- *Margins increase in synchronous design*

Leakage currents are increasing
- Transistors leak and consume power even when off
- *New interest in switchable power supplies*

Power has become critical
- *Heightened interest in optimal clock gating*

Scalability
- *Chips becoming too big/complex to be driven by one global clock*

Cost of doing business as usual is increases ➔
Interest in alternatives increases!
The Asynchronous Alternative

Synchronization and communication between blocks implemented with asynchronous channels that send and receive tokens.
Asynchronous Circuit Design - Today

Applications

- 3D Network on chips (STMicroelectronics)
- Ethernet Switches (Intel SRD)
- Ultra high-speed FPGAs (Achronix)
- Low-power chip design (Encryption – Tiempo, …)
- Resilience / Process variation

Basic challenges: Automation

Proteus design flow (USC)

- Started at USC Async CAD/VLSI Group
- Commercialized by TimeLess (2008)
- Acquired by Fulcrum (2010)
- Intel Acquired Fulcrum (2011)
- Used in Intel Ethernet Alta FM6000 chip

Fulcrum Microsystems Ethernet switch chip (up to 72 10G ports, 40G) - 1.2 B transistors, 90% Asynchronous 13% Proteus
Intel (Fulcrum) Ethernet Switch/Router

- Microcode Programmable Ethernet Switch
- TSMC 65nm GP
- 1.2 billion transistors
- 15 MB total memory
- 72x10G / 18x40G
- 400 ns latency
- 77% Full-Custom Async Memories & Crossbars
- 13% Async P&R (Proteus)
- <100 man-years

Industrial High-Performance Asynchronous Design
SoC Interconnect – “Nexus”

- Seamlessly bridges clock domains
- Low-latency clock domain crossings
- Low-latency pipelined interconnect
- Non-blocking pipelined crossbar w/ line-rate arbitration
- High-throughput and bandwidth

Generic Nexus Example

- Synchronous IP block
- Asynchronous IP block
- Pipelined repeater
- Clock domain converter
SoC Interconnect – Synch to Async Converters

- **Protocol**
  - Synchronous side adheres to request/grant protocol
  - Does not change data until grant is received
  - Implies synchronous side is “latency-insensitive”
- **Low Latency**
Agenda

Asynchronous Review

• Potential Advantages
• Channel-Based Design
• Bundled-Data versus QDI Design Styles

Proteus Design Flow

• RTL Legacy Design
• System-Verilog Front End
• Power Optimizations
• Performance Optimizations
Async logic can remove wasteful margins and can achieve robust, fast circuits, with efficient power usage.
Breaking Flip-Flop Alignment

Flip-to-Flop Critical Path

True “Algorithmic” Cycle
The Sub/Near Threshold Advantage

Issues with Synchronous

• Cycle time bounded by worst-case PVT requirements leads to poor energy efficiency
• Process variability causes up to 50% deviation in gate delay

Asynchronous Solution

• Shorter algorithmic cycle reduces leakage energy dissipation per computation lowering the optimal $V_{DD}$ and significantly increasing energy efficiency
• Local control logic can mitigate effects of intra-die gate delay variations by delaying handshaking signals using replicated logic gates

[Chang, 2010]

[Chang, 2010]
System Design

- Collection of **Processes** linked by **Channels**
- Channels pass messages with **guaranteed delivery**
- Processes synchronize
- Processes can be **decomposed** into smaller processes

![Diagram of system design](image)
Asynchronous Channels

Bundled-Data Channel
small area and lower power

Data rails don’t switch when the same value is communicated multiple times

Dual-Rail (1-of-N) Channel
Fewer timing assumptions, higher power/area

Data rails switch even when the same value is communicated over and over
Asynchronous Blocks
Bundled-Data Design

- Static combinational logic (typically) and standard FFs or latches
- Controller CTRL drives local clock to bank of FFs (or latches)
- Delay line matches worst-case delay of combinational logic
  - PVT+ margins limits performance
  - Post-silicon tuning can help!
Asynchronous Blocks - Precharged Half Buffer

Supports 1-of-N Rail Channels
- Delay-insensitive communication
- Communication robust to variations in delay

Implemented quasi-delay-insensitivity
- Negligible timing assumptions
- Logic robust to variations in delay

Dynamic Logic Datapath
- Low latency (no margins)
- High switching activity
Asynchronous Block Design - Proteus

• Automated Synthesis, Place & Route
  • Starts with high-level CSP specification
  • Maps to a fixed async cell library (301 cells)
  • Leverages Commercial Tools

• Productivity benefits
  • At increased area, latency, power cost
  • Use custom for the big but simple units (crossbars, RAM)
  • Use Proteus to fill in the control and complex units
  • Enables quick ECO changes and ATPG

• Applications
  • Legacy RTL
  • New Async Designs
The Proteus-A Flow – Legacy RTL

Key Features

- Re-uses **synchronous** EDA tools
- Seamless **integration** into existing flows
- Multiple back-end **design style supported**
- Up to **2X** higher performance

Tool Status

- Uses proprietary QDI library
- Academic version (Proteus-A) enhanced significantly at USC
- Bundled-data version in progress

Advances

- Advanced performance analysis
- Performance-driven and power optimizations
module acc (_RESET, CLK, A, Sum);
    parameter WIDTH = 16;
    input _RESET, CLK;
    input [WIDTH-1:0] A;
    output [WIDTH-1:0] Sum;
    reg [WIDTH-1:0] Sum;
    always@(posedge CLK, negedge _RESET)
    begin
        if(!_RESET)
            Sum <= 0;
        else
            Sum <= Sum + A;
    end
endmodule

LOGIC3_69 X4 \body.add_40_17.g15833 (.A0 (\value$L[12] ), .A1
LOGIC3_69 X4 \body.add_40_17.g15834 (.A0 (\value$L[8] ), .A1
LOGIC2_2 X4 \body.add_40_17.g15835 (.A0 (\body.add_40_17.n_2 ), .A1
    (\body.add_40_17.n_50 ), .X (\body.add_40_17.n_55 ));
LOGIC5_1F1FFFF_X2 \body.add_40_17.g15836 (.A0 (\body.add_40_17.n_32
    ), .A1 (\body.add_40_17.n_47 ), .A2 (\body.add_40_17.n_37 ), .A3
    (\value$L[14] ), .A4 (\body.add_40_17.n_42 ), .X
    (\body.add_40_17.n_54 ));
LOGIC5_4F4FFFF_X2 \body.add_40_17.g15837 (.A0 (\body.add_40_17.n_44
    (\body.add_40_17.n_53 ));
LOGIC3_69 X4 \body.add_40_17.g15838 (.A0 (\value$L[10] ), .A1
LOGIC3_96 X4 \body.add_40_17.g15839 (.A0 (\value$L[5] ), .A1
LOGIC2_2 X4 \body.add_40_17.g15846 (.A0 (\body.add_40_17.n_27 ), .A1
    (\body.add_40_17.n_46 ), .X (\body.add_40_17.n_50 ));
LOGIC3_68 X4 \body.add_40_17.g15841 (.A0 (\body.add_40_17.n_43 ), .A1
    (\body.add_40_17.n_22 ), .A2 (\body.add_40_17.n_164 ), .X
    (\body.add_40_17.n_49 ));
LOGIC3_69 X4 \body.add_40_17.g15842 (.A0 (\value$L[6] ), .A1
LOGIC3_61 X4 \body.add_40_17.g15843 (.A0 (\body.add_40_17.n_102 ), .A1
    (\body.add_40_17.n_34 ), .A2 (\body.add_40_17.n_39 ), .X
    (\body.add_40_17.n_47 ));
Amber23 – Proteus-A Case Study

- Download from http://opencores.com/project,amber
- ARM-compatible 32-bit RISC processor
- 3 stages: FETCH, DECODE and EXECUTE

Zhang, USC Summer Research, 2012
Amber23 – Performance Comparison

Average-case data benefit can be significant!
Async Design - SystemVerilog CSP (SVC)

- SystemVerilog interface abstracts **channel wires** as well as **communication protocol**

- **Send/Receive**
  - Blocking tasks (Flow control)

```verilog
module Sender (interface R);
  parameter WIDTH = 8;
  logic [WIDTH-1:0] data;
  always begin
    //produce data
    R.Send(data);
  end
endmodule

module Receiver (interface L);
  parameter WIDTH = 8;
  logic [WIDTH-1:0] data;
  always begin
    //consume data
    L.Receive(data);
  end
endmodule
```
// Sender (DataGen)
always begin
    #Delay; R.Send(data);
end

// Receiver
always begin
    L.Receive(data);
    #FL;
    R.Send(data);
    #BL;
end

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Cursor</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td></td>
</tr>
<tr>
<td>bitsd[15:0]</td>
<td>'h6A9A5A6</td>
</tr>
<tr>
<td>bitse[15:0]</td>
<td>'hFFFF</td>
</tr>
<tr>
<td>lastTokValue[15:0]</td>
<td>'h7B0D</td>
</tr>
<tr>
<td>comInd</td>
<td>0</td>
</tr>
<tr>
<td>tokCount</td>
<td>'d 4</td>
</tr>
<tr>
<td>status</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Dfs</th>
<th>100,000,000fs</th>
<th>100,500,000fs</th>
<th>101,000,000fs</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>zzzzzzz</td>
<td>00000000</td>
<td>6B99556</td>
<td>00000000</td>
<td>66695598</td>
</tr>
<tr>
<td>zzzzz</td>
<td>'hFFFF</td>
<td>5E81</td>
<td>5699</td>
<td></td>
</tr>
<tr>
<td>zzzzz</td>
<td>3524</td>
<td>'hFFFF</td>
<td>5699</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>s_pend</td>
</tr>
<tr>
<td>idle</td>
<td>s_pend</td>
<td>idle</td>
<td>s_pend</td>
<td></td>
</tr>
</tbody>
</table>

---

- Receiver pending on Receive
- Sender performs Send, Communication happens
- No one is Sending or Receiving
- Sender pending on Send
- Receiver performs Receive, Communication happens
The Proteus-A Flow – SVCRTL

Key New Features

- Supports System Verilog CSP front-end
- Enables user-defined conditional communication
- Saves power at architectural level

Tool Status

- Proprietary version starting from CAST developed at Fulcrum
- System Verilog version subsequently developed at USC
- Used in current research at USC and Technion and 40+ person async class
Conditional communication reduces token flow, saving power

- Traditionally - manually introduced via user-created decomposition
- Recent research - automatically introduced via Operand Isolation

Saifhashemi, PATMOS 2012
// e1=0, e2=1, e3=0, e4=1

if (e1==1) I1.Receive(d1);
if (e2==1) I2.Receive(d2);
o1 = f1 (d1,d2);
o2 = f2 (d1,d2);
if (e3==1) O1.Send(o1);
if (e4==1) O2.Send(o2);
end
Power Optimization Overview

- Conditioning
  - Automatically add conditional communication
- Reconditioning
  - Optimize the existing conditionality
Power Saving - The Opportunity

Unnecessary calculation
Our Solution - Adding Isolation Cells

- All inputs/outputs are unconditional
- Operand Isolation
  - And-based isolation cells
  - Generated by synchronous RTL synthesizer
  - Does not prevent switching in asynchronous circuits

Isolation cells are not effective in asynchronous circuits
Our Solution - Conditioning

No Activity
Power Optimization Results

• Case study: 32-bit ALU placed and routed
  • Back annotated switching activity using a VCD file
  • Results:
    • Isolating ADD and SUB are detrimental for $r_{ADD}$ and $r_{SUB} > 0.2$
    • 53% power reduction when only isolating MUL ($r_f=0.25$)
    • Area cost of isolating MUL is about 4% and no performance penalty

Table 1: Post-layout total switching power measurements (mW).

<table>
<thead>
<tr>
<th>Activity</th>
<th>$P^o$</th>
<th>$P^i_M$</th>
<th>$P^i_{ASM}$</th>
<th>$P^m$</th>
<th>$(P^o-P^i_M)/P^o$</th>
<th>$(P^o-P^i_{ASM})/P^o$</th>
<th>$(P^m-P^i_{ASM})/P^m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{AND} = 1$</td>
<td>110</td>
<td>34</td>
<td>34</td>
<td>39</td>
<td>69%</td>
<td>69%</td>
<td>13%</td>
</tr>
<tr>
<td>$r_{ADD} = 1$</td>
<td>110</td>
<td>34</td>
<td>38</td>
<td>41</td>
<td>69%</td>
<td>66%</td>
<td>8%</td>
</tr>
<tr>
<td>$r_{SUB} = 1$</td>
<td>110</td>
<td>34</td>
<td>38</td>
<td>42</td>
<td>69%</td>
<td>65%</td>
<td>9%</td>
</tr>
<tr>
<td>$r_{MUL} = 1$</td>
<td>110</td>
<td>106</td>
<td>106</td>
<td>108</td>
<td>4%</td>
<td>3%</td>
<td>2%</td>
</tr>
<tr>
<td>$r_f = 0.25$</td>
<td>110</td>
<td>52</td>
<td>54</td>
<td>56</td>
<td>53%</td>
<td>51%</td>
<td>3%</td>
</tr>
</tbody>
</table>

Saifhashemi, Patmos 2012
i1=1; i2=1; i3=1;

fork
    //Receive e1, ..., e4
    if (e1) begin
        I1.Receive(i1);
        I2.Receive(i2);
    end
    if (e2)
        I3.Receive(i3);
join
    o1 = g1 (i1,i2,i3);
    o2 = g2 (i1,i2,i3);
if (e3)
    01.Send(o1);
if (e4)
    02.Send(o2);

Conditional communication is explicit and only at primary IO
Definition (The Reconditioning Problem): Rearrange location of RECEIVE and SEND cells to minimize Power consumption while preserving functional behavior.
Power Results

**Power Comparison: 32 bit**

- **RECON1:** Dual-mode arithmetic unit
- **RECON2:** Conditional multiplier
- **ALU-OI:** ALU after operand isolation

The Slack Matching Problem - Add minimum number of pipeline buffers to the circuit to meet a target cycle time $\tau$.

- This problem is unique to asynchronous design
- Unfortunately, often yields significant area and power

Represent 30% of area and power

Peter A. Beerel; Andrew M. Lines; et. al., “Slack matching asynchronous designs,” ASYNC’06
Conditional Slack Matching Advantage – Conditional behavior yields less stalls and thus not as many pipeline buffers needed

- Previously ignored – conservatively modeled as unconditional

Najibi, ICCAD 2013
# Conditional Slack Matching - Results

Najibii, 2013

<table>
<thead>
<tr>
<th>Bench</th>
<th>SMBuffs (count)</th>
<th>$SM_{Time}$ (sec.)</th>
<th>$\Delta_{MAX}$ (trans.)</th>
<th>$\tau_B$ (trans.)</th>
<th>$\tau_0$ (trans.)</th>
<th>$\tau_1$ (trans.)</th>
<th>$\tau_{sim}$ (ps)</th>
<th>SMBuffs (Red. %)</th>
<th>$\overline{\tau}_{sim}$ (Incr. %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>as27</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td>20.90</td>
<td>18</td>
<td>32</td>
<td>1014.06</td>
<td>35.71</td>
<td>12.67</td>
</tr>
<tr>
<td>as298</td>
<td>77</td>
<td>2</td>
<td>4</td>
<td>19.65</td>
<td>18</td>
<td>26</td>
<td>935.97</td>
<td>6.10</td>
<td>4.00</td>
</tr>
<tr>
<td>as386</td>
<td>105</td>
<td>2</td>
<td>12</td>
<td>21.33</td>
<td>18</td>
<td>34</td>
<td>989.59</td>
<td>18.60</td>
<td>-0.97</td>
</tr>
<tr>
<td>as349</td>
<td>38</td>
<td>1</td>
<td>4</td>
<td>30.06</td>
<td>30</td>
<td>30</td>
<td>1501.55</td>
<td>59.14</td>
<td>23.33</td>
</tr>
<tr>
<td>as382</td>
<td>114</td>
<td>3</td>
<td>4</td>
<td>25.63</td>
<td>24</td>
<td>26</td>
<td>1273.84</td>
<td>31.74</td>
<td>16.00</td>
</tr>
<tr>
<td>as400</td>
<td>131</td>
<td>2</td>
<td>12</td>
<td>21.56</td>
<td>18</td>
<td>35</td>
<td>1004.90</td>
<td>20.61</td>
<td>11.66</td>
</tr>
<tr>
<td>as420</td>
<td>166</td>
<td>2</td>
<td>35</td>
<td>21.90</td>
<td>18</td>
<td>36</td>
<td>1058.66</td>
<td>31.97</td>
<td>17.63</td>
</tr>
<tr>
<td>as444</td>
<td>137</td>
<td>3</td>
<td>26</td>
<td>21.82</td>
<td>18</td>
<td>36</td>
<td>1034.22</td>
<td>42.44</td>
<td>14.91</td>
</tr>
<tr>
<td>as510</td>
<td>36</td>
<td>6</td>
<td>4</td>
<td>29.26</td>
<td>28</td>
<td>34</td>
<td>1457.98</td>
<td>73.91</td>
<td>12.16</td>
</tr>
<tr>
<td>as526a</td>
<td>176</td>
<td>3</td>
<td>19</td>
<td>21.77</td>
<td>18</td>
<td>36</td>
<td>1043.59</td>
<td>39.93</td>
<td>15.95</td>
</tr>
<tr>
<td>as713</td>
<td>133</td>
<td>3</td>
<td>1</td>
<td>28.41</td>
<td>28</td>
<td>30</td>
<td>1379.17</td>
<td>27.72</td>
<td>6.29</td>
</tr>
<tr>
<td>as820</td>
<td>86</td>
<td>8</td>
<td>7</td>
<td>30.52</td>
<td>29</td>
<td>36</td>
<td>1464.75</td>
<td>42.28</td>
<td>4.57</td>
</tr>
<tr>
<td>as838</td>
<td>436</td>
<td>11</td>
<td>28</td>
<td>21.44</td>
<td>18</td>
<td>34</td>
<td>1004.55</td>
<td>22.28</td>
<td>11.62</td>
</tr>
<tr>
<td>as1196</td>
<td>811</td>
<td>51</td>
<td>40</td>
<td>21.79</td>
<td>18</td>
<td>35.33</td>
<td>1081.56</td>
<td>26.00</td>
<td>20.17</td>
</tr>
<tr>
<td>as1488</td>
<td>124</td>
<td>35</td>
<td>8</td>
<td>31.23</td>
<td>30</td>
<td>35.6</td>
<td>1516.00</td>
<td>62.87</td>
<td>1.72</td>
</tr>
<tr>
<td>as1238</td>
<td>1006</td>
<td>52</td>
<td>37</td>
<td>21.80</td>
<td>18</td>
<td>35.33</td>
<td>1061.75</td>
<td>17.94</td>
<td>17.97</td>
</tr>
<tr>
<td>as5378</td>
<td>1384</td>
<td>215</td>
<td>36</td>
<td>23.49</td>
<td>20</td>
<td>36</td>
<td>1107.04</td>
<td>12.85</td>
<td>0.64</td>
</tr>
<tr>
<td>as9234</td>
<td>316</td>
<td>99</td>
<td>18</td>
<td>42.58</td>
<td>36</td>
<td>44</td>
<td>2011.46</td>
<td>30.55</td>
<td>-4.22</td>
</tr>
</tbody>
</table>

33% less buffers on average

Najibii, 2013
Asynchronous Advantages
- Variation-aware slack matching

Can easily avoid performance bottlenecks caused by variations
Optimizing pipeline rings

• Definition (informal)
  • Pipeline buffers configured in a loop
  • Can be combined with forks & joins
• Used in implementing iterative algorithms
  • Each iteration implemented by a token traversing the loop
• Multiple tokens in loop possible
  • Each token independent of others
  • Implements multi-threading

function gcd(A, B)
  while A \neq B
    if A > B
      A := A - B
    else
      B := B - A
  return O = A

Euclid’s Algorithm for Greatest Common Divisor (GCD)
Example - A GCD Implementation

- **Implementation Notes**
  - MUXs are same as MERGEs but consume both input tokens
  - TB is a token buffer
    - Generates a token on initialization with configurable value
    - Acts as a buffer afterwards
  - FORK cells implied by branching channels (for clarity)
  - All cells use pipeline handshaking
- **Architectural Feature**
  - Contains many pipeline rings
Example - A GCD Implementation

- Operation
  - TB generates tokens to select input tokens come in on PIs A & B
  - Tested for equality which controls how they are routed
    - If != routed to SUBs & ‘<‘
    - Otherwise, A is routed to output
  - SUBs concurrently generate differences.
  - Specific difference routed back to merge controlled by ‘<‘ and MUXs
GCD Circuit – Clustering into pipeline stages

• Pipeline stages communicate with each other via handshaking
  • Local cycle time can limit throughput
• More pipeline stages
  • Increase in latency
  • Increase in area
  • Smaller local cycle time
• General goal
  • Use largest pipeline stages
  • That do not create a throughput bottleneck
Asynchronous Advantages - Variation-aware pipelining

Performance quickly becomes average latency around ring!

Throughput vs. Ring latency graph with annotations for FL = 2, BL = 8, and 10% Variation.
Asynchronous Router – EE552

NoC Component

- Fully flow controlled
- Pipelined
- Fully Arbitrated

EE-552 Class Project

- SystemVerilog Spec
- Decomposition
  - Arbiter - custom component
  - Others – Proteus-A
- 120+ students since 2010
Final Flow Considerations

Static Timing Analysis
- Verify timing constraints and performance is a must
- Trick traditional tools into working with asynchronous circuits

Analog Verification
- Domino logic used in QDI flows sensitive to charge sharing
- Asynchronous channels cannot tolerate cross-talk glitches
- Special spiced-based tools developed

Asynchronous Scan
- Asynchronous scan is a must but doable

Design for Silicon Debug
- Chip deadlock is still difficult to debug
Conclusions

The Asynchronous Design Landscape

- Synchronous design rigidity continues to hamper quality design
- Asynchronous design offers solutions but has many challenges

Design Flow Requirements

- Design flows must easily integrate into synchronous designs
- Circuit quality must compete very well to warrant switching design styles

Our approach

- Proteus provides a good design framework for automation of both legacy RTL and SystemVerilog CSP
- Variation-aware design yields near-threshold design advantages
Acknowledgements