ACK - A Framework for High Level Synthesis of Asynchronous Circuits

Prabhakar Kudva, Hans Jacobson
Ganesh Gopalakrishnan, Al Davis, Erik Brunvand
Robert Thacker, Johan Nyblom

University of Utah
Computer Science Department
Asynchronous Synthesis

- Macromodular Approach:

  Martin, Brunvand, Akella, van Berkel
  - Syntax directed translation to modules
  - Restricted low level module libraries
  - Does not exploit progress in logic synthesis
  - Differ from synchronous paradigm

- Direct Synthesis Approach:

  Davis et al., Nowick, Yun, Siegel, Stevens
  - Synthesis of customized AFSMs
  - Similar to synchronous methods
Motivation for ACK

- Synthesis flow similar to synchronous allow use of standard commercial and public domain tools - a key to acceptability of asynchronous design
- Ability to specify overall system structure rather than individual controller structure
- Take advantage of high level as well as low level synthesis techniques
- Protocol independent description allows targeting of both two and four phase implementations
The ACK Interface

• Requirements:
  - Easy to use
  - All tools in one framework

• Our solution:
  - Graphical user interface
  - Embedded in synchronous framework
Standard HDL Interface

- Requirements for asynchronous language:
  - Channels
  - Process description
  - Direct concurrency support

- Requirement from designers:
  - Standardized language

- Our solution:
  - Verilog-+
    * Synthesizable subset of Verilog
    * Extended with channels
  - Allows behavioral simulation
The Design Structure

- Structural level design:
  - Modules communicating via
    * Channels
    * Handshakes
    * Shared variables
- Module:
  - Communication interface
  - Local resources
  - Partitions
Delay Models and Environment Assumptions

• Control:
  – Huffman style
  – Delay insensitive

• Datapath:
  – Self-timed blocks
  – Bundled data between blocks
Communication Protocols

- Two phase:
  - Advantages:
    * Well suited for protocol based designs
    * All transitions do useful work
  - Disadvantages:
    * Slow or large datapath

- Four phase:
  - Advantages:
    * Well suited for datapath intensive designs
    * Fast and small datapath
  - Disadvantages:
    * Not all transitions do useful work
ACK - System Flow

Verilog +

Petri Net Language
Allocation Refinement
Partition
Petri nets to burst mode

Choice of 2/4 Phase

Viewlogic synthesis for datapath

3D
Tech map to standard gates

Lager layout tools

Magic

Verilog
Veriwell simulator

Complex-gate Realization

Cadence layout synthesis
Allocation and Refinement

- **Allocation**

  - For the structural description:
    * Allocate and connect physical resources

  - For each module:
    * Allocate physical resources
      - local variable - register
      - computation - data block
      - data dependent choice - predicate block

![Diagram of Data Block (DB) and Predicate Block (PB) models](image-url)
• Refinement
  - For each module:
    • Generate control graph acting on allocated resources
Partitioning

- Why Partition?
  - Behavioral description often large and centralized
  - Asynchronous synthesis of large designs is slow
  - Exploit spatial and temporal locality
  - Reduce controller area and propagation delay

- Why assisted partitioning?
  - Partitioning by hand is error prone
  - Complex signal sharing arrangements sometimes necessary
Problem:
- Partitioning of incompletely specified machines

Requirement:
- Correctly handle control flow and signal sharings between partitions
- Composite behavior of distributed controllers same as for centralized

Solution:
- Handshakes for sequential control flow between partitions
- Input and output state machines to handle distribution of shared signals
• Partitioning approach:
  
  – Splits sequential controller flow
  
  – Sequential flow is ensured by handshaking between partitions under fundamental mode assumption
Illustration of Partitioning
- **Example of Partitioning - CD Player Error Detector**
  - Divided into three partitions to
    * Minimize overhead for loops
    * Make design synthesizeable
  - Resulted in 5 Input and Output State Machines for shared variables

![Diagram of three partitions]

Partition 1

- \( t \neq 0 \)
- \( n = 27 \)
- \( n = 31 \)
- \( n \geq 0 \)
- \( e = \text{syn0} \)
- \( \text{Case } = \text{syn0} \)

Partition 2

- \( t \neq 0 \)
- \( n = n - 1 \)
- \( \text{syn0} = \text{gfadd}(s, \text{syn0}) \)
- \( \text{syn1} = \text{gfadd}(s, \text{alpha}(\text{syn1})) \)
- \( \text{syn2} = \text{gfadd}(s, \alpha(\alpha(\text{syn2}))) \)
- \( \text{syn3} = \text{gfadd}(s, \alpha(\alpha(\alpha(\text{syn3})))) \)

Partition 3

- \( n \geq 0 \)
- \( (\text{syn0} \neq \text{syn1} \text{ or } \text{syn1} \neq \text{syn2} \text{ or } \text{syn2} \neq \text{syn3}) \)
- \( n = n - 1 \)
- \( \text{syn0} = \text{alpha}(\alpha(\alpha(\text{syn0}))) \)
- \( \text{syn1} = \text{alpha}(\alpha(\alpha(\text{syn1}))) \)
- \( \text{syn2} = \text{alpha}(\alpha(\alpha(\alpha(\text{syn2}))) \)
- \( \text{syn3} = \text{alpha}(\alpha(\alpha(\alpha(\alpha(\text{syn3})))) \)

\( t \equiv 0 \)
# Results for Partitioning

<table>
<thead>
<tr>
<th>Controller</th>
<th>Num of BM transitions</th>
<th>Size of IO set</th>
<th>Synthesis time (secs)</th>
<th>Num of literals</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CD Error Detector</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centralized</td>
<td>1824</td>
<td>68</td>
<td>did not finish</td>
<td>-</td>
</tr>
<tr>
<td>Partition 1</td>
<td>110</td>
<td>18</td>
<td>800</td>
<td>94</td>
</tr>
<tr>
<td>Partition 2</td>
<td>32</td>
<td>29</td>
<td>220</td>
<td>96</td>
</tr>
<tr>
<td>Partition 3</td>
<td>28</td>
<td>25</td>
<td>140</td>
<td>122</td>
</tr>
<tr>
<td>ISM 1,2,3</td>
<td>81</td>
<td>13</td>
<td>230</td>
<td>63</td>
</tr>
<tr>
<td>ISM 4</td>
<td>16</td>
<td>6</td>
<td>90</td>
<td>14</td>
</tr>
<tr>
<td>ISM 5</td>
<td>7</td>
<td>14</td>
<td>80</td>
<td>92</td>
</tr>
<tr>
<td><strong>Barcode Reader</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centralized</td>
<td>960</td>
<td>49</td>
<td>did not finish</td>
<td>-</td>
</tr>
<tr>
<td>Partition 1</td>
<td>26</td>
<td>14</td>
<td>34</td>
<td>14</td>
</tr>
<tr>
<td>Partition 2</td>
<td>40</td>
<td>8</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>Partition 3</td>
<td>72</td>
<td>19</td>
<td>500</td>
<td>13</td>
</tr>
<tr>
<td>Partition 4</td>
<td>26</td>
<td>23</td>
<td>53</td>
<td>43</td>
</tr>
<tr>
<td>ISM 1,2</td>
<td>56</td>
<td>8</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>ISM 3</td>
<td>64</td>
<td>9</td>
<td>28</td>
<td>53</td>
</tr>
<tr>
<td><strong>GCD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centralized</td>
<td>126</td>
<td>25</td>
<td>33420</td>
<td>207</td>
</tr>
<tr>
<td>Partition 1</td>
<td>22</td>
<td>15</td>
<td>30</td>
<td>33</td>
</tr>
<tr>
<td>Partition 2</td>
<td>72</td>
<td>18</td>
<td>90</td>
<td>12</td>
</tr>
<tr>
<td>ISM 1,2</td>
<td>48</td>
<td>7</td>
<td>25</td>
<td>14</td>
</tr>
<tr>
<td><strong>Factorial</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centralized</td>
<td>44</td>
<td>20</td>
<td>620</td>
<td>88</td>
</tr>
<tr>
<td>Partition 1</td>
<td>12</td>
<td>13</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>Partition 2</td>
<td>28</td>
<td>15</td>
<td>36</td>
<td>9</td>
</tr>
<tr>
<td>ISM 1,2</td>
<td>16</td>
<td>5</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td><strong>Loop Example</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centralized</td>
<td>32</td>
<td>5</td>
<td>38</td>
<td>199</td>
</tr>
<tr>
<td>Partition 1</td>
<td>14</td>
<td>5</td>
<td>20</td>
<td>58</td>
</tr>
<tr>
<td>Partition 2</td>
<td>12</td>
<td>6</td>
<td>16</td>
<td>3</td>
</tr>
</tbody>
</table>
Burst Mode Generation

- Burst mode specification:
  - MIC
  - Huffman style
  - Maximal set property
  - Each state has unique entry point

- Burst mode generation:
  - Two phase
    * Refined Petri Net -> State Graph
    * State Graph -> Burst Mode Graph
  - Four phase
    * Refined Petri Net -> State Graph
    * State Graph -> Burst Mode Graph
    * Reshuffle Burst Mode Graph
- Two phase case
Refined Petri Net ⇒ State Graph ⇒ Burst Mode Graph

State Graph

Burst Mode Graph
• Four phase case

Petri Net description

Refined Petri Net controller
Refined Petri Net ➔ Burst Mode Graph

Refined Petri Net controller

Burst Mode Graph
Burst Mode Graph \Rightarrow Reshuffled Burst Mode Graph

Example of shuffling procedure

start.req+ / start.ack+
start.req- / start.ack- ain.req+ bin.req+
ain.ack+ bin.ack+ / a.load+ b.load+
a.loaddone+ / a.load- b.load-
b.loaddone+ / ain.req- bin.req-
ain.ack- bin.ack- / FAB.req+
FAB.ack+ / FAB.req-
FAB.ack- / r.load+
r.loaddone+ / r.load-
r.loaddone- / res.req+
res.ack+ / res.req-
res.ack+ / ain.req+ bin.req+

Burst Mode Graph

Reshuffled Burst Mode Graph
Technology Mapping

- Two level AND-OR gates
- Standard CMOS complex gates
- Customized CMOS complex gates
  - VLSI size decrease - wire delay significant
  - Reduce constraints in hazard free synthesis
  - Purely combinational solution to a larger class of problems
  - Method of logical effort can be applied at transistor level
• SOP/SOP Implementation

![Diagram showing SOP/SOP Implementation with cubes labeled as ON-set and OFF-set, and a logic circuit diagram with inputs and outputs marked as VDD and VSS.]
• SOP/SOP Single Gate Example

Burst Mode Graph

Karnaugh Map for output X

Two level gate - no static cover

Customized single complex gate

3D implementation
Results for Single Complex Gate

- Reduced synthesis constraints reduce need for adding state variables

- Customized complex gate gives less number of transistors than two level gates

- Delay is comparable to or less than two level gate implementation depending on static hazard occurrence and transistor stack height.

<table>
<thead>
<tr>
<th>Name</th>
<th>S.Gate #statevar</th>
<th>C.Gate area</th>
<th>S.Gate area</th>
<th>C.Gate delay</th>
<th>S.Gate delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp_tr</td>
<td>1</td>
<td>136</td>
<td>513</td>
<td>0.35</td>
<td>0.64</td>
</tr>
<tr>
<td>tri_st</td>
<td>1</td>
<td>148</td>
<td>636</td>
<td>0.19</td>
<td>0.81</td>
</tr>
<tr>
<td>store_st</td>
<td>1</td>
<td>223</td>
<td>609</td>
<td>0.43</td>
<td>0.72</td>
</tr>
<tr>
<td>dual_si</td>
<td>0</td>
<td>96</td>
<td>152</td>
<td>0.20</td>
<td>0.30</td>
</tr>
<tr>
<td>for_lp</td>
<td>1</td>
<td>213</td>
<td>332</td>
<td>0.38</td>
<td>0.53</td>
</tr>
<tr>
<td>sim_mod</td>
<td>1</td>
<td>102</td>
<td>278</td>
<td>0.24</td>
<td>0.58</td>
</tr>
</tbody>
</table>
- Single complex gate - table 2

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Output</th>
<th>C. Gate</th>
<th>Std. Gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>chu-ad-opt</td>
<td>dr</td>
<td>1.1 ns</td>
<td>1.5 ns</td>
</tr>
<tr>
<td></td>
<td>lr</td>
<td>1.75</td>
<td>1.4</td>
</tr>
<tr>
<td>van-bek-ad-opt</td>
<td>dr</td>
<td>0.9</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>zr</td>
<td>0.92</td>
<td>1.34</td>
</tr>
<tr>
<td></td>
<td>lr</td>
<td>1.2</td>
<td>0.93</td>
</tr>
<tr>
<td>sendr-done</td>
<td>DoneS</td>
<td>1.3</td>
<td>1.36</td>
</tr>
<tr>
<td>sbuf-read-ctl</td>
<td>Ack</td>
<td>1.3</td>
<td>1.36</td>
</tr>
<tr>
<td></td>
<td>RamRdSbuf</td>
<td>0.97</td>
<td>1.53</td>
</tr>
<tr>
<td>q42</td>
<td>a4</td>
<td>0.94</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>r2</td>
<td>1.32</td>
<td>1.61</td>
</tr>
</tbody>
</table>
- **SOP/SOP Multilevel Complex Gate**
  - Why Multilevel?
    * Give solution to a larger class of problems than single complex gate
    * Have solution to problems with illegally intersecting cubes of dynamic MIC transitions
    * Make purely combinational controllers
    * Reduce fundamental mode delay
**SOP/SOP Multilevel Gate Example**

(a) Initial Burst Mode Specification

(b) Karnaugh Map for output X

(c) Reduced problem to derive sum of products

(d) Gate level implementation - no static cover

(e) Complex Gate Implementation

(f) Circuit for X derived by 3D
Results for Multilevel Complex Gate

- Reduced synthesis constraints reduce need for adding state variables
- Reduce fundamental mode delay since feedbacks often can be avoided
- Multilevel customized complex gates give less number of transistors than two level gates
- Delay is comparable to or less than two level gate implementation

<table>
<thead>
<tr>
<th>Name</th>
<th>S.Gate #statevar</th>
<th>C.Gate area</th>
<th>S.Gate area</th>
<th>C.Gate delay</th>
<th>S.Gate delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus_tr</td>
<td>2</td>
<td>271</td>
<td>1107</td>
<td>0.30</td>
<td>0.55</td>
</tr>
<tr>
<td>run_dp</td>
<td>1</td>
<td>360</td>
<td>1218</td>
<td>0.38</td>
<td>1.35</td>
</tr>
<tr>
<td>si_stack</td>
<td>1</td>
<td>177</td>
<td>312</td>
<td>0.20</td>
<td>0.32</td>
</tr>
<tr>
<td>sm_stat</td>
<td>2</td>
<td>187</td>
<td>1181</td>
<td>0.31</td>
<td>0.74</td>
</tr>
<tr>
<td>sm_dyn</td>
<td>2</td>
<td>222</td>
<td>1109</td>
<td>0.23</td>
<td>0.59</td>
</tr>
<tr>
<td>l_cov</td>
<td>2</td>
<td>173</td>
<td>824</td>
<td>0.21</td>
<td>0.58</td>
</tr>
<tr>
<td>comp_dp</td>
<td>2</td>
<td>395</td>
<td>1162</td>
<td>0.39</td>
<td>0.70</td>
</tr>
</tbody>
</table>
Example: Barcode Reader (HLS95)
(a) Refined Petri Net controller

(b) State Graph

(c) Burst Mode controller

TWO PHASE
Two level gate implementation

# transistors:
T112JOIN = 22
Width.SetMux1 = 32

SOP/SOP complex gate for T112JOIN

# transistors:
T112JOIN = 14
Width.SetMux1 = 18

SOP/SOP complex gate for width.SetMux1 and prevbit.SetMux1
Two Level Gate Implementation

SOP/SOP complex gate for data.Load

SOP/SOP complex gate for T112JOIN

# transistors:
data.Load = 20
T112Join = 32
width.Load = 18

# transistors:
data.Load = 12
T112Join = 25
width.Load = 12
## Results for two and four phase

<table>
<thead>
<tr>
<th>Controller</th>
<th>2/4 phase protocol</th>
<th>Num of BM transitions</th>
<th>Synth Time</th>
<th>IO size</th>
<th>Literals</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Factorial</strong></td>
<td>2</td>
<td>40</td>
<td>150</td>
<td>19</td>
<td>101</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>13</td>
<td>35</td>
<td>19</td>
<td>69</td>
</tr>
<tr>
<td><strong>Factorial pipe</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition 1</td>
<td>2</td>
<td>32</td>
<td>38</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>11</td>
<td>30</td>
<td>15</td>
<td>42</td>
</tr>
<tr>
<td>Partition 2</td>
<td>2</td>
<td>32</td>
<td>63</td>
<td>14</td>
<td>47</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>9</td>
<td>26</td>
<td>14</td>
<td>33</td>
</tr>
<tr>
<td><strong>Reg_int</strong></td>
<td>2</td>
<td>28</td>
<td>84</td>
<td>26</td>
<td>36</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>15</td>
<td>54</td>
<td>26</td>
<td>65</td>
</tr>
<tr>
<td><strong>Reg_sh</strong></td>
<td>2</td>
<td>26</td>
<td>160</td>
<td>26</td>
<td>74</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>15</td>
<td>67</td>
<td>26</td>
<td>78</td>
</tr>
<tr>
<td><strong>Mixed_rc</strong></td>
<td>2</td>
<td>88</td>
<td>3240</td>
<td>19</td>
<td>212</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>17</td>
<td>87</td>
<td>22</td>
<td>63</td>
</tr>
<tr>
<td><strong>Diffeq (HLS92)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition 1</td>
<td>2</td>
<td>26</td>
<td>127</td>
<td>29</td>
<td>109</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>13</td>
<td>74</td>
<td>27</td>
<td>51</td>
</tr>
<tr>
<td>Partition 2</td>
<td>2</td>
<td>52</td>
<td>110</td>
<td>26</td>
<td>62</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>17</td>
<td>88</td>
<td>29</td>
<td>52</td>
</tr>
<tr>
<td>Partition 3</td>
<td>2</td>
<td>22</td>
<td>60</td>
<td>25</td>
<td>47</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>13</td>
<td>64</td>
<td>27</td>
<td>43</td>
</tr>
<tr>
<td>ISM 1,2,3</td>
<td>2</td>
<td>16</td>
<td>18</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>noshuffle</td>
<td>4</td>
<td>8</td>
<td>18</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td><strong>Barcode Reader</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition 1</td>
<td>2</td>
<td>26</td>
<td>34</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>9</td>
<td>25</td>
<td>17</td>
<td>25</td>
</tr>
<tr>
<td>Partition 2</td>
<td>2</td>
<td>40</td>
<td>25</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>9</td>
<td>18</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Partition 3</td>
<td>2</td>
<td>72</td>
<td>501</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>14</td>
<td>33</td>
<td>21</td>
<td>33</td>
</tr>
<tr>
<td>Partition 4</td>
<td>2</td>
<td>26</td>
<td>53</td>
<td>23</td>
<td>43</td>
</tr>
<tr>
<td>shuffle</td>
<td>4</td>
<td>9</td>
<td>38</td>
<td>25</td>
<td>81</td>
</tr>
<tr>
<td>ISM 1,2</td>
<td>2</td>
<td>56</td>
<td>22</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>noshuffle</td>
<td>4</td>
<td>14</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ISM 3</td>
<td>2</td>
<td>64</td>
<td>28</td>
<td>9</td>
<td>53</td>
</tr>
<tr>
<td>noshuffle</td>
<td>4</td>
<td>16</td>
<td>19</td>
<td>9</td>
<td>17</td>
</tr>
</tbody>
</table>
Contributions

- Behavioral description in standard language
- High level synthesis targeting state machines
- Two and four phase implementation
- Partitioning of incompletely specified machines
- Hazardfree technology mapping to complex gates
- Complete asynchronous design framework
- Many realistic examples