GAELS Progress

Wei Song

13/03/2013
Content

• Tool flow
• Progress
  – Signal-level DFG
  – Register Relation Graph (RRG)
  – FSM detection
• Future works
• Conclusion
Tool flow

RTL Verilog HDL

Cell Library

VCD waveform

Timing info

Pipeline usage

AVerilog synthesiser

Multiple smaller RTL Verilog HDL designs

Async interfaces

Commercial tools

blackboxes
don’t touch
Flow inside Synthesizer

- RTL Verilog
- Verilog Parser
- Elaborator
- SDFG & RRG Generation
- GALS Partition
- Simulation Waveform
- Frequency Constraints
- Area, Power constraints
- Asynchronous Component Library
- Async Pipeline Insertion
- Netlist Writer
- RTL Verilog
- Async Gate-Level Verilog
- Syn, P&R constraints
- Automatic Constraint Generation
- Asynchronous Component Library
- Area, Power constraints
- Frequency Constraints

Advanced Processor Technologies Group
The School of Computer Science
Hypotheses in Partition Detection

Boundaries are the synchronous buses with variable data rates controlled by FSMs.
Progress from Last Meeting

• Signal-level Data Flow Graph (DFG)
  – Parse Verilog to AST
  – AST to DFG conversion
  – Arc type detection

• Register Relation Graph

• Automatic FSM detection
  – Detect all FSMs, counters and flags with finite state spaces
Signal-level DFG

```vhdl
always @(posedge clk or negedge rstn)
  if(~rstn)
    state <= R;
  else
    state <= state_nxt;

always @(state or cnt) // next state
  if(cnt == 0)
    case(state)
      R: state_nxt = YR;
      YR: state_nxt = G;
      G: state_nxt = YG;
    default: state_nxt = R;
    endcase // case (state)
  else
    state_nxt = state;

always @(posedge clk or negedge rstn)
  if(~rstn)
    cnt <= 0;
  else if(cnt == 0)
    case(state)
      R: cnt <= 2;
      YR: cnt <= 49;
      G: cnt <= 4;
    default: cnt <= 49;
    endcase // case (state)
  else
    cnt <= cnt - 1;

assign red = state == R ? 1 : 0;
assign green = state == G ? 1 : 0;
assign yellow =
  (state == YR || state == YG) ? 1 : 0;
```

Advanced Processor Technologies Group
The School of Computer Science

7

13/03/2013
always @(posedge clk or negedge rstn)
    if(~rstn)
        state <= R;
    else
        state <= state_nxt;

always @(state or cnt) // next state
    if(cnt == 0)
        case(state)
            R: state_nxt = YR;
            YR: state_nxt = G;
            G: state_nxt = YG;
            default: state_nxt = R;
        endcase // case (state)
    else
        state_nxt = state;

always @(posedge clk or negedge rstn)
    if(~rstn)
        cnt <= 0;
    else if(cnt == 0)
        case(state)
            R: cnt <= 2;
            YR: cnt <= 49;
            G: cnt <= 4;
            default: cnt <= 49;
        endcase // case (state)
    else
        cnt <= cnt - 1;

assign red = state == R ? 1 : 0;
assign green = state == G ? 1 : 0;
assign yellow =
    (state == YR || state == YG) ? 1 : 0;
Signal-level DFG
Iterate all paths between two registers and reduce them to one or two arcs in RRG. (dynamic programming is used)
FSM detection

- At least one of an FSM’s output paths is a self-loop path.
- At least one of an FSM’s output paths is a control path towards another register.
- All input data of an FSM comes from self-loop paths or constant numbers.
Test Cases

<table>
<thead>
<tr>
<th>Design</th>
<th>DFG Nodes</th>
<th>Registers</th>
<th>Time</th>
<th>FSMs</th>
<th>Rate</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Reported</td>
<td>Verified</td>
<td></td>
</tr>
<tr>
<td>OR1200</td>
<td>2074</td>
<td>124</td>
<td>&lt; 1s</td>
<td>19</td>
<td>17</td>
<td>89%</td>
</tr>
<tr>
<td>Reed-Solomon</td>
<td>1063</td>
<td>325</td>
<td>2.0s</td>
<td>55</td>
<td>53</td>
<td>96%</td>
</tr>
<tr>
<td>H.264/AVC</td>
<td>7043</td>
<td>855</td>
<td>7.1s</td>
<td>55</td>
<td>49</td>
<td>89%</td>
</tr>
</tbody>
</table>

**OR1200**: micro processor, combinational loop, program counter.

**RS decoder**: ad hoc coding style, multiple signals in one always block, use range as control

**H.264**: large design with large fanouts (a global counter with 400 fanouts, 280K unfolded output paths).

**False negative error**: not found

**False positive error**: around 10%

**Causes of error**: combinational loop, range expression, sequential assigns.
Compare with Others

• Coding style
  – Synthesis tools like DC
  – Only recognise FSMs written in standard one or two always blocks

• Pattern recognition
  – No support for explicit type detection.
Future works

- Boundary detection
  - Pattern of the buses with variable data rate
  - Pattern of on-chip buses
  - Pattern of FIFOs
  - Relations between controllers
Conclusions

• Large scale Verilog designs have been parsed and converted into signal level DFGs
• FSMs and controlling counters have been automatically detected
• Need a method to detect available system boundaries