Automating the Design of an Asynchronous DLX Microprocessor

Manish Amde IIT – Bombay
Ivan Blunno Politecnico di Torino
Luciano Lavagno Politecnico di Torino
Christos Sotiriou ICS - FORTH

Outline

• The ASPIDA project
• The DLX microprocessor
• Pipefitter
• DLX architecture
• Verilog specification
• Synthesis result
• Conclusions and future work
**ASPIDA**

ASynchronous open-source Processor
IP of the DLX Architecture

Project page: www.ics.forth.gr/carv/aspida
Opencores page: www.opencores.org/projects/aspida

Some information…

- Open-source IP DLX
- Financed by the European Community
- Three years project
- Cooperation of three groups:
  - University of Manchester (Bus and Interfaces)
  - ICS-FORTH (Logic Synthesis, P&R, Testability)
  - Politecnico di Torino (Specification, HLS, Simulation)
ASPIDA

Features…

- Well suited for IP re-use (e.g., in SoC)
- Low-power and low-EMI
- Technology-portable
- WISHBONE interface
- Targeted for ASIC EDA tools
- Industrial-quality testability (internal scan)

DLX

Features…

- 32-bit 5-stage pipelined microprocessor
- Simple instruction set
- Many tools for compilation/emulation
- Good for teaching purposes
### Pipefitter

Fully automated asynchronous design flow based on a standard HDL (Verilog), standard commercial EDA tools and standard cells

### Pipefitter Story

- 1998: First prototype
- 2000: 8-bit FIR filter (ASYNC 2000)
- 2002: 8-bit non-pipelined microprocessor (ACID 2002)
- 200x: Ready for the industry…
Pipefitter Architecture

- Asynchronous Control Unit: STG or DC netlist
- Synchronous Data Path: matched delay
- Handshake protocol: four-phase

Pipefitter Design Flow

- Behavioral HDL
- Simulation
Pipefitter Design Flow

CU flow

STG

optimization (petrify)

Afsmgen

standard cell

Behavioral HDL

control/data splitting

optimizations

synthesizable HDL

logic synthesis

standard cell

delay insertion

HDL netlist implementation

DP flow

HDL netlist implementation

simulation

simulation

HDL netlist implementation
Pipefitter Design Flow

- HDL netlist + constraints
- Simulation
- P & R
- CHIP

Specification Language: Verilog

- Assignments
- Arithmetic/Logic Operations
- HANDSHAKES
- Control Statements
• Reads the new instruction
• Sets the address for next instruction
• Sets the control bits
• Reads data from the Register File

• Executes address and data arithmetic operations
• Decides on conditional branches
• Reads/writes the Data Memory

• Sets the value to be written back in the Register File
**Input-to-Output Handshakes**

Asynchronous Module

- wait(DI_req);
- DO = DI;
- fork
- begin
  - DI_ack = 1;
  - wait(!DI_req);
  - DI_ack = 0;
- end
- begin
  - DO_req = 1;
  - wait(DO_ack);
  - DO_req = 0;
  - wait(!DO_ack);
- end
- join

**IF – Next Address**

Adder

- Add = Mux + 4;
- module _IF_OU_0(ck, input_1_0, input_2_0,
  output_0);
  input ck;
  input [31:0] input_1_0;
  input [31:0] input_2_0;
  output [31:0] output_0;
  reg [31:0] output_0;
  wire check_0 = output_0[0];
  always @ (posedge ck)
    output_0 = input_1_0 + input_2_0;
endmodule
Add = Mux + 4;

module _IF_reg_outdata_Add(ck, en, input_0, output_0);
input ck;
input en;
input [31:0] input_0;
output [31:0] output_0;
reg [31:0] output_0;
always @(posedge ck)
  if(en)
    output_0 = input_0;
endmodule

IF – Next Address

Register

GPR[3] is read now

GPR[3] will be written here

Data Hazard

1) GPR[3] = 5;
Data Hazard

NEW DECODING

\[
\begin{array}{cc}
0 & 0 \\
1 & 1 \\
2 & 0 \\
30 & 0 \\
31 & 0 \\
\end{array}
\]

REGISTER LOCKING

---

Data Hazard

Verilog code

```
// WB on GPR[i] decoded
lock[i] = 1;
......
GPR[i] = WB_data;
lock[i] = 0;
```
Instruction Memory

ADD → COMB. REG. FILE → DATA

req → delay → ack

Synthesis Result

<table>
<thead>
<tr>
<th>MODULE</th>
<th>CU</th>
<th>DP</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>15%</td>
<td>79%</td>
<td>6%</td>
</tr>
<tr>
<td>ID</td>
<td>27%</td>
<td>67%</td>
<td>6%</td>
</tr>
<tr>
<td>EX</td>
<td>27%</td>
<td>67%</td>
<td>6%</td>
</tr>
<tr>
<td>MEM</td>
<td>26%</td>
<td>63%</td>
<td>11%</td>
</tr>
<tr>
<td>WB</td>
<td>32%</td>
<td>56%</td>
<td>12%</td>
</tr>
<tr>
<td>ADLX</td>
<td>25%</td>
<td>68%</td>
<td>7%</td>
</tr>
</tbody>
</table>
Conclusions

• A first version of an asynchronous DLX has been developed
• An automated design flow has been effectively used
• The design by a student showed the ease of use
Future Work

- Comparison with synchronous equivalent
- Support for interrupts
- Support for floating point operations
- Integration with interfaces