An Exploration of the MPEG Algorithm Using Latency Insensitive Design

EE249 Presentation (12/04/1999)
Trevor Meyerowitz
Mentored by: Luca Carloni
Outline

- Motivation
- Explanation of Latency Insensitive Design
- Why MPEG is interesting?
- Current Implementation
- Conclusions and Future work
Motivation: Doomsday for IC design?

- Die Size, Design complexity, and Clock speed are all increasing.
- Clock propagation is an increasing problem.
  - Multiple clock cycles to cross a chip
  - Verification becomes harder
  - Buffering the clock isn’t enough
The Current Situation

- Traditional design flow uses direct connections between individual blocks
- The wires might be too long.
- Blocks are highly coupled.
  - Difficult to connect independent IP’s
Solution: Latency Insensitive Design

• Remove wires and put shells around blocks
• Connect shells with an arbitrary number of relay stations
• Data coherency is preserved
Latency Insensitive Design: The Relay Station

- **Data**: the signal
- **Void**: whether the signal is legitimate data
- **Stop**: tells previous relay station whether it’s ready or not to receive data

*Source: A Methodology for Correct by Construction Latency Insensitive Design (carloni et al)*
Current Status:

- Proof of concept (Carloni, McMillan, Sangiovanni-Vincentelli 1999)
- Functional Example
- Case Study: The PDLX micro-processor
- Different examples are required
  - Demonstrate SOC feasibility
  - Especially Multimedia
Sample SOC idea

HDTV
*(picture from www.sony.com)*
High MHz
High Power
and Bandwidth

Cell Phone
*(picture from www.nokia.com)*
Low MHz
Low Power
and Bandwidth
Why MPEG Decode is Good

- Complicated & Highly Data Dependant
  - Different resolutions, encoding techniques
- Not as arbitrary or complicated as Encode
- Real Time Constraints
- Great Target for HW/SW Partitioning/Co-Design
- Relevant for SOC today
Description of MPEG Decode

- Aside from headers can be classified as 4 types of blocks
  - **D-Block** - (control information)
  - **I-Block** - Full Picture
  - **P-Block** - Picture Dependant on prior picture
  - **B-Block** - Picture Dependant on future picture

- Each non-D-Block is split up into 8x8 chunks
Description of MPEG Decode

Input Stream

software

hardware

Header Decode

VLC/RLC Decode

Inverse DCT

Inverse ZigZag

Inverse Quantize

Decoded Picture

Add Picture

Forward Predictor

Backward Predictor

Memory Bus

Picture Store (I)

Picture Store (P)

Decoded Picture

Adding Picture

Forward Predictor

Backward Predictor

Memory Bus

Picture Store (I)

Picture Store (P)

Source: MPEG-2 John Watkinson
Current Implementation

- Have translated hardware portions from MPEG-2 Decoder C file to Verilog
- Hooked up them up using relay stations
- Verified Functionality
Results:

- Relay Stations do add a startup delay similar to pipelining
  - Basic Simulation confirms 4.5* cycle delay
- Software-ish portions are a nightmare to translate to Verilog
  - Memory access and pointers
  - Software is sequential, HW isn’t usually
Conclusion

■ What Remains to be Done:
   – More thorough verification
   – Further exploration of results
   – Perhaps add VLC/RLC to hardware
   – Possibly interface with PLI

■ Possible Future Work:
   – Exploration of interfacing with software
   – Further Performance Analysis
   – Automatic Shell Synthesis
   – Instantiation of Multiple Execution Units