EEL 4783: HDL in Digital System Design

Lecture 5: *

Prof. Mingjie Lin
Why Asynchronous Logic?

Low Power
  Do nothing when there is nothing to be done

Modularity
  Added design freedom and component reusability

Electromagnetic Interference (EMI)
  Clocks concentrate noise energy at particular frequencies

Security?
  Surprise the hackers!
Contents

Some asynchronous microprocessors
Problems in processor design
Problems and opportunities in asynchronous design
Example solutions to a selection of problems
Memories and peripherals
Design styles
GALS and asynchronous interconnection
Conclusions
Why Microprocessors?

Well defined problem
  Easy to demonstrate correct function

Self-contained
  Make stand-alone devices

Not obviously suited to asynchronous techniques
  Forced to examine ‘real’ problems

Interesting problems
  New techniques to devise
Microprocessors as Design Examples

AMULET1 (1994)
ARM6 compatible processor (almost)
1.0um 60 000 transistors
Hand designed
Bundled data, two-phase control
Feasibility study

Experiences
Two-phase logic hard to work with and interface to
Microprocessors as Design Examples

AMULET2e (1996)
ARM7 compatible processor
Asynchronous cache
0.5um 450 000 transistors
Hand designed
Bundled data, four-phase control

Experiences:
Easy to use, self-contained system
Microprocessors as Design Examples

AMULET3i (2000)
ARM9 compatible processor
SoC: Memory, DMA controller, bus, …
0.35um 800 000 transistors
Mostly hand designed
Bundled data, two-phase control
Commercial application (DRACO)
Experiences:
Universities don’t have the resources for such projects!
Other Asynchronous Microprocessors

“Caltech Asynchronous Microprocessor” (1989)
First asynchronous microprocessor

University of Tokyo’s “TITAC-2” (1997)
Mostly hand designed

Caltech “MiniMIPS” (R3000) (1997)
Hand designed
Other Asynchronous Microprocessors

IMAG (Grenoble) “ASPRO-216” (1998)
16-bit signal processor

Philips Research Laboratories 80C51 (1998)
Synthesized using “Tangram” tool

Theseus Star-8 (2000)
Uses Null Convention Logic (NCL)
AMULET3 Processor Architecture

Highly pipelined structure
Similar to synchronous architecture

Features:
- Branch prediction
- Halting
- Forwarding
- Out-of-order completion
- Precise exceptions
Synchronous vs. Asynchronous Architectures

An AMULET looks very like a synchronous ARM

  Functional blocks divided by pipeline latches

But:

Some ideas can be ‘copied', some need reinvention

Some synchronous ‘tricks’ don’t work in an asynchronous environment

  Non-local interactions, dependency resolution, ...

Pipelining is too easy

  Temptation to inefficient design
Data-dependent Timing

ARM instructions allow a shift before an ALU operation
These are rarely exploited
The shifter is often bypassed
The execution timing may be adjusted appropriately
Process-level Parallelism

Example: decode and execute stages
Various threads – many invoked conditionally
Skewed pipeline latches (lower power EMI)
Variable stage delay (e.g. ‘stretch’ for series shift)
Differing pipeline depths (extra buffer for LDM/STM)
Conditional invocation of functions
In a synchronous design non-local values can be read.

The ARM uses the PC as an operand (e.g., relative branch).

The actual value is two instructions ‘out’ due to the pipeline depth of the original implementation.
In an asynchronous design only ‘adjacent’ stages can communicate

AMULET supplies the PC value with every instruction

This can be adjusted as required in an implementation independent manner
Handshaking allows pipeline occupancy to be changed without global control

Thumb instructions normally fetched in pairs but executed individually

Same scheme applied to (e.g.) ‘Load Multiple’

Similar scheme applied to removing ‘surplus’ packets
Halting

Any unit can impose an arbitrary delay

A pipeline stage could choose to delay indefinitely

This will halt the whole pipeline shortly thereafter

   Downstream stages ‘drain’
   Upstream stages ‘back up’

Halted CMOS circuits use ‘no’ power

   No clock power either

Restart is instantaneous

   Both AMULET2 and AMULET3 exploit this

   for easy power management
Coloring

- Pipeline occupancy may be non-deterministic

Branch changes colour

Not all stages need be full

Colours matched here

Branches change the local colour and request a new stream
Prefetched operations discarded until a new stream arrives
Deadlock

Question: if pipeline occupancy is variable, what happens if a token is inserted into a ‘full’ pipeline?

Answer: Deadlock!

A danger with the large number of states available can be avoided with careful design.

Two cases in AMULET3:

Branches when the prefetch pipeline is full

Memory conflict between instruction and data fetches

Both were known early and prevented at a higher level.
Data Aborts

Wait for MMU to abort or not
Stretch cycle if memory access
Data Aborts

Speculate on memory not aborting
Register results returned out of order
More parallelism => higher throughput
Reorder Buffer

- Allows instructions to complete in any order
- Resolves register dependencies
- Allows register forwarding
- Permits low-overhead memory management
- Supports exact page fault exceptions
Data can arrive along any path at any time, providing their targets are mutually exclusive.

Read out waits for each register to be filled in turn, then copies out the result (or not, if unwanted).

Copy out frees the register but does not delete the data.
AMULET3i Memory System

The RAM is ‘dual-port’ (at this level)
The instruction bus is simpler
so it has a higher bandwidth
Memory Structure

The local RAM is divided into 1 Kbyte blocks

Unified RAM model

Close to dual-port efficiency

About 50% instruction fetches are from the ‘Ibuffers’
AMULET2e Cache

Pipelined
Data-dependent timing
Asynchronous line fetch

Newer design includes:
Copy-back
Write buffer
Victim cache
Synthesis vs. Hand Design

Most of the AMULET3i system was designed at schematic level

Part (the DMA controller) was a test of Balsa

A new asynchronous synthesis system

Synthesised blocks are more efficient to design but less efficient in operation

Suitable for (e.g.) peripherals that are rarely invoked

No timing closure problems!
DMAC

About 70,000 transistors
Regular structures (register banks) in full custom
Control synthesised from Balsa description
Cheats slightly by letting a clock into one corner!
Asynchronous on-chip Interconnection

MARBLE

Centrally arbitrated, multi-channel, asynchronous on-chip bus
Supports: 8-, 16- and 32-bit transfers, bus locking, sequential bursts, ...
Separate, decoupled, asynchronous transfer phases for address and data
32-bit bundled data pathways
Used on AMULET3i
Standard ‘master’ and ‘slave’ interfaces
Standard interface to on-chip synchronous bus too
Asynchronous on-chip Interconnection

CHAIN

Delay insensitive coding for ‘distance’ transmission
Requires more wires per bit
Exploit lack of clock to send serial symbols fast
4 wires, 2-bit (1-of-4) symbols
Point-to-point unidirectional wiring
Standard ‘master’ and ‘slave’ interfaces
Could easily provide standard synchronous interfaces
GALS

Globally Asynchronous, Locally Synchronous interconnection

Use ‘conventional’ synchronous design blocks for SoC

Use asynchronous interconnection to avoid timing closure problems

May be the first big application of asynchronous logic

No reason why the ‘local’ blocks need to be synchronous

…
AMULET3i

AMULET3 microprocessor (ARMv4T)
8 Kbytes RAM
16 Kbytes ROM
Flexible, multi-channel DMAC
Programmable memory interface
On-chip asynchronous bus (MARBLE)
Bridge to on-chip synchronous bus
Configuration registers
Software debug support
Test interface
Experience of Large Asynchronous Designs

Hard, but feasible

Competitive

Advantages?

- Power management
- EMI
- Composability (GALS)
- Security?

Commercial

- Philips, Theseus, ADD, Intel?, …
Conclusions

Asynchronous logic:
Can be competitive with ‘conventional’ designs
Has advantages with low-power and low EMI
  think portable systems
May be the only solution for some tasks
  block interconnections on large chips

but

Designing big systems is a lot of work
It’s hard to catch up with the big companies
Lecture schedule

See Webpage:

www.eecs.ucf.edu/~mingjie/EEL4783
Final issues

• Come by my office hours (right after class)

• Any questions or concerns?