Demystifying Data-Driven and Pausible Clocking Schemes

Robert Mullins

Tutorial presented at 18th UK Asynchronous Forum
Newcastle, September 2006
Synchronous to Delay-Insensitive Approaches to System Timing

Timing Assumptions

- Synchronous
  - Sub-System
  - Local
  - Local Relative
  - Wire Delay
  - Isochronic Forks
- Delay Insensitive
  - Global
  - Local
  - Local Relative
  - Wire Delay
  - Isochronic Forks

Local Clocks/Interaction with data (becoming aperiodic)

Data-Driven and Pausable Clocks

Less Detection
Introduction

- Clock Stretching Idea
- History and Uses
  - Crossing clock domains
  - Data-driven clocking
- Building local clock generators
  - Data-Driven and Pausible Clocks
  - Clock-Tree Insertion Delays
- Beyond synchronization
  - Novel Applications
Clock Stretching

• Local clock generator is constructed using a tunable delay-line and inverter (ring-oscillator)
• We are able to interrupt the ring oscillator and delay the generation of next clock edge until some event has completed
• Length of each clock cycle may now vary

Minimum clock period is fixed

Clock period may be stretched
• Metastability always an issue when attempting to synchronize an external input to a local clock
• Typical solution is to allocate a fixed period of time for metastability to resolve
  – e.g. two-flip flop synchronizer (1 cycle delay)
  – Accept a small chance of failure
• Pěchouček (1976)
  – Observed it was “fundamentally impossible to exclude arbitrarily long responses in any flip-flop”
  – Described circuits to allow the clock period to be stretched until metastability has been resolved
Data-Driven Clocking

• Metastable states may be avoided if asynchronous input starts local clock
• Pěchouček also outlined such a system
  – No longer a synchronization issue if clock is quiescent when data arrives
• Seitz (Ch. 7, Mead and Conway book)
  – Seitz creates a self-timed interface or wrapper around synchronous blocks
  – Used in proprietary designs since 1968
  – “Assures performance and correct operation will not be compromised by wire delay in signals or in clock distribution”
Data-Driven Clocking

• Example Berkeley MAIA chip (2000)
  – GALS architecture, data-flow driven processing elements (“satellites”)
Data-Driven Clocking

• On-Chip Network Routers
  – Difficult to set global network clock frequency
  – Let local data streams determine rate of clocking
  – Need to sample other inputs on receiving an input request – who is to be admitted and who must wait (arbitration)
    • Similar logic to static priority arbiter implementations
Building local clock generators

- Lots of published schemes, but similarities often hidden
  - Different circuit styles
  - Different specification & synthesis techniques
- Our approach
  - Introduce two basic types of local clock generator
    - *Data-driven* and *Pausible*
    - In reality both are very similar, distinction is aimed at understanding circuits and related work
  - For each case we will look at a number of different input port behaviours we may need
    - *Arbitrated*, *Sampled* and *Synchronised*
  - Demonstrate how a complete wrapper may be “assembled”
Data-Driven and Pausible Clocks

(a) Ring Oscillator

(b) Data-Driven Clock

(c) Ring Oscillator (II)

(d) Pausible Clock
Data-Driven Clocks

(a) Ring Oscillator

(b) Data-Driven Clock

(c) Ring Oscillator (II)

(d) Pausible Clock
Data-Driven Clocks

- Basic input port behaviours:
  - Arbitrated Inputs
    - Can only progress with one input at a time
  - Synchronised Inputs
    - Need full set of inputs to progress
  - Sampled Inputs
    - Can progress with variable number of data inputs (e.g. on-chip network router)
Data-Driven Clock: Sampled Inputs

- Sample inputs when at least one input is ready (and clock is low)
- Assert Lock
- Either admitted or locked out
- Data-Driven Clock Template
Data-Driven Clocking: Pipelines and Flushing

- May require multiple clocks to process a single data item, e.g. when logic is pipelined
- Flushing Alternatives:
  - Eager Flushing
    - Initialise counter to make N successive requests for a new clock to be generated
  - Time-Out Flush
    - May wish to allow pipeline to be data-driven, time-out as last resort and flush using counter
  - Uninterrupted Flush
  - Pull-Driven Flush (driven by output)
- Flush logic is easy to add, simply appears as additional input
Clock Stretching Approach

• Common to see GALS wrappers built around a 'stretchable' clock circuit
• Idea is very similar to a data-driven clock
• Interface is reduced to a single ‘stretch’ input
  – Stretch signal is asserted synchronously
  – Removed asynchronously to permit next rising clock edge
Pausible Clocks

(a) Ring Oscillator

(b) Data-Driven Clock

(c) Ring Oscillator (II)

(d) Pausible Clock
Pausible Clock

• In contrast to data-driven approach, the clock is normally free-running
• Mutual-exclusion element permits clock to be interrupted or paused
• Often used to ensure value-safe communication in a GALS system
Pausible Clock – Input Ports

Arbitrated Inputs

Tree Arbiter – with Eager Output Request

Sampled Inputs

Synchronized Inputs
Consumer Side Interface Example

- Data is latched in first register while input holds grant from MUTEX
- MUTEX is released, new clock edge is generated, data is admitted
Arbitrated-call based pausable clock interface

- Similar consumer side interface using a single input register
- In this case, if the input is granted it also goes on to request the next clock edge

- Potential for creating a wider window for accepting data
Asynchronous Synchronisers and Q-elements

- Amulet3 interrupt synchroniser (1997)
- Similarities to pausible clock circuit
- Also interesting to look at Q-Modules again [Rosenberger/Molnar et al, 1988]
Output Ports

• Output port types:
  – **Scheduled**
    • Output operation must be completed on a particular clock cycle
  – **Registered**
    • Cannot overwrite output register until it has been read
  – **Polled**
    • Port polls output to determine when data may be sent (synchronisation issue which may require clock period to be extended)

• Implementations are based on input port circuits already discussed
A GALS Wrapper Example

• Outline specification:
  – Free running clock
  – Asynchronous input
    • we know nothing about when data will arrive
    • For simplicity, let's assume we can always accept new data
  – Registered output feeding asynchronous FIFO
A GALS Wrapper Example: Step 1.

Local clock generator with H/S interface
A GALS Wrapper Example: Step 2.
A GALS Wrapper Example: Step 3.

Provide registered output port support (stretchable clock template)
A GALS Wrapper Example: Step 4.
Clock Tree Insertion Delays

• Delay from root to leaf of clock tree can be considerable (certainly non-zero!)
• If every clock cycle is the same, this *clock insertion delay* is not normally an issue
• If we stretch the clock the insertion delay must be considered in our timing analysis (also true for clock gating in synchronous world)
• Not difficult to handle, but can increase time required to admit new data
  – Very large synchronous islands make little sense if we pursue a GALS approach anyway
Clock Tree Insertion Delays

• Must ensure input data is latched before we remove data and request!

• (a) Need to be careful not to extend clock cycle

• (b) Illustrates a simple approach if insertion delay is less than one cycle
Clock Tree Insertion Delays

- How do we handle multi-cycle insertion delays?
- Need to ensure we admit data on the correct clock cycle
- Cannot cheat and promote data! (requires arbitration, clock edge already in tree)

We simply remember on which clock cycle data has been scheduled to be admitted
Summary

• Simple view of elastic clocking ideas
• Practical wrappers can be assembled from a few simple circuit templates
• Lots I haven’t talked about
  – approaching specification and synthesis more formally (next talk!)
  – formal verification
    • I use Veraci (Paul Cunningham’s PhD work)
    • Verification aided by modular view of wrappers
  – performance and optimisation of interfaces...
Novel Applications: Timing Speculation

- Circuit-Level Timing Speculation – Razor [ARM/Michigan]
- Exploit elastic clocking ideas to simplify error recovery mechanism and timing analysis
Novel Applications: Fine-Grain Clock Gating

• Power gating techniques increasingly important as static power consumption increases

• Multiple levels of sleep
  – In some cases will require restoration of state

• Exploit elastic clocking
  – Self-timed wake-up process
  – Simplify timing analysis and implementation
  – Power gate much smaller blocks
  – Speculative sleep and wake-up control
Synchronous vs. Asynchronous Control for GALS

• Synchronous alternatives
  – 2 Flip-flop synchronizers
  – Clock Gating
  – Pipeline flushing mechanisms
  – Timing analysis and implementation can get complex
  – Not necessarily transparent to IP core architecture

• Elastic clocking techniques
  – Fully transparent, robust, simple to compose blocks and techniques
  – Optimise common-case and make exceptional cases work
Conclusions

• Scaling trends suggest we will soon see hundreds of networked IP cores on a single chip

• Timing and integration issues alone suggest a GALS approach based on elastic clocking should play a role
  – Competition from well established synchronisation techniques

• The need to optimise each IP block by exploiting timing speculation, clock, signal and power gating, DVFS etc. may see significant additional advantages
  – Research opportunity