Asynchronous Design Basics: Handshaking

Kenneth S. Stevens

Electrical and Computer Engineering
University of Utah
Asynchronous Design Basics

- Asynchronous Design is conceptually very elegant and easy
- Consists of a few basic aspects
  1. Handshake Channel
     - Control Plane
     - Data Plane
  2. Communication Protocol
  3. Timing Formalism

Now you know 80% of what you need to design async chips!
Asynchronous Design Basics

“I still don’t understand how a transistor works”
– A VLSI God & one of my Mentors
Handshaking

How does one best communicate task transfers?

- Between co-workers in an office?
- Between pipeline stages on a chip?
- Between cars entering an intersection?
- Between terrestrial and satellite hardware?
- (Between a wine shop and its supplier/customers?)

Often need to communicate initiation and completion
Handshaking

Even clocked designers are familiar with handshakes. Consider:
Handshaking

This is a typical four-phase handshake
Handshaking

1. Req
2. Ack
3. execute
4. Req

Ack

Ack

Ack

Ack

Ack

Ack
Handshaking

Asynchronous handshake protocols are effectively the same:

- Same basic activity
- But, NOT controlled by a clock
- Activated by the transitions of the control signals
Handshaking

Specification languages are highly useful and insightful:
Handshake Data Protocols

Four-phase protocols can employ various phases:

- **Narrow**:
  - Req: 1, 2, 3, 4
  - Ack: execute

- **Broad**:
  - Req: 1, 2, 3, 4
  - Ack: execute
Handshake Data Protocols

When is data valid at receiver?
Handshake Data Protocols

When is data valid at receiver?
Handshake

Two Phase Protocol

A

Req

Ack

Data

B

Req

Ack

Data

data1

data2

1

2

1

2
Two Phase Design

Petri-net specification

Diagram:

Two Transition boxes:

1. 1
2. 2

Data boxes:

data1
data2

Two boxes:

A
B

Requests:

Req
Ack

Data:

Req
Ack
Data

Response:

Req
Ack
Handshake Channel Protocol Properties

Observations:

1. **Channel** defines the wires and their specification between self timed elements.

2. **Handshake protocol** is *very* simple:
   - do forever { req followed by ack }
   - Same for two or 4 phase

3. **Control Elements** interface channels based on their specification
   (Channels can be either *push* or *pull*)
Handshake Channel Protocol Properties

*Bundled data (push) channel*

(a)

(b) 4-phase protocol

(c) 2-phase protocol
Exercise and Discussion

1. Which protocol is better, a 2 phase or 4 phase protocol?
2. Are there circuit examples that justify your argument?
What About Data Plane?

clock network
Protocol Timing

**Time** is a core aspect of system design.

- Chuck Seitz referred to it as the *third dimension* on a chip.
  - Spatial $x, y$ values (and possibly architecture) are other dimensions

- Time is fundamental
  - Time dictates the **correctness** of a design
  - Time dictates the **performance** of a design

- Directly relates protocols and hardware.

Timing should be *formally modeled*
Exercise and Discussion

1. Can an asynchronous design operate independently of delays?
2. Should an asynchronous design operate independent of delays?
Handshake Formalism

Request Acknowledge Handshaking

\( x \preceq y \): \( x \) precedes \( y \)

\( x \not\preceq y \land x \not\succeq y \): \( x \) is concurrent with \( y \): \( x \equiv y \)

Properties:

- reflexive: \( x \leq x \)
- antisymmetric: \( x \leq y \land y \leq x \Rightarrow x \equiv y \)
- transitive: \( x \leq y \land y \leq z \Rightarrow x \leq z \)
Function Formalism

Combinational Logic $CL$ is defined as:

$CL = CL \lor \bigcup_{CL} : CL$ obeys interconnect rule

Interconnect rule:

$I$ is input to interconnection

$I_{CL}$ is input to CL

$O$ is output of interconnection

$O_{CL}$ is output of CL

- $\forall I_{CLi} \in CLi \in \bigcup_{CL} : I_{CLi} = O_{CLj} \in CLj \in \bigcup_{CL} \lor I_{CLi} \in I$
- $\forall O_i O \in I \lor O_i \in O_{CLi}$
- There are not closed loops.

_basically produces a fully connected directed acyclic graph_
Timing Formalism

Equipotential Region

Two signals are considered to be equipotential iff

- $x - \Delta \leq y \leq x + \Delta$

- where $\Delta$ is usually defined as $\approx \tau$ in an integrated circuit.

(\tau is the delay of a minimum inverter driving another minimum inverter with no parasitics. Traditional FO4 gate delay is $5\tau$)

This basically means that the variation in delay between the two signals is unobservable by the logic.
Timing Formalism

Weak conditions:

1. Some input becomes defined before some output becomes defined
2. All inputs become defined before all outputs become defined
3. All outputs become defined before some input becomes undefined
4. Some input becomes undefined before some output becomes undefined
5. All inputs become undefined before all outputs become undefined
6. All outputs become undefined before some inputs become defined
Timing Formalisms

1. Delay Insensitive
   - Unbounded delays \([0 - \infty)\) on all gates and wires
   - Why do we need unbounded delay on both?

2. Speed Independent
   - Unbounded delay \([0 - \infty)\) on gates or wires
   - What is the difference?

3. Self-Timed
   - Timing is local to the block
   - Not mathematically well defined

4. Relative Timed
   - General theory of timing that applies to all systems
Timing Example

What are the conditions of failure under such a design?

- Theory of acknowledgment
  - OR gate or functionality problematic
Bundled vs Delay Insensitive Design

Following is a “Bundled Data” design, that uses Relative Timing or Speed Independent theory.

What is needed for delay insensitive?
Bundled vs Delay Insensitive Design

This style needs *encoding* of the data bits!
Delay Insensitive Codes

Dual Rail most common:

4-phase dual-rail (push) channel

<table>
<thead>
<tr>
<th></th>
<th>d.t</th>
<th>d.f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty (&quot;E&quot;)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Valid &quot;0&quot;</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Valid &quot;1&quot;</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Not used</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Data {d.t, d.f}

"0"  "E"  "1"
Delay Insensitive Codes

Handshake and dual rail codes:
Modular Asynchronous Protocols

Delay Insensitive: Every data bit and function logic must be encoded

- Dual Rail
  - two wires encode a single data bit
  - a delay insensitive protocol
    - 00: spacer or NULL token
    - 01: valid 1
    - 10: valid 0
  - data busses move from fully invalid to fully invalid
  - simple to encode and decode (C-element)
  - naturally half shielded
  - data activity factor: 100%!!!
Modular Asynchronous Protocols

Exercise: What is dual rail code for two-phase protocol?
(LEDR)
Modular Asynchronous Protocols

- n-of-m Codes
  - $n$ signals asserted for $m$ wires
  - a delay insensitive protocol
  - 1-of-4
    - 0000: spacer or NULL token
    - 0001: valid binary 0
    - 0010: valid binary 1
    - 0100: valid binary 2
    - 1000: valid binary 3
  - same number of wires as Dual Rail
  - more difficult to encode and decode
  - data activity factor: 50%
Modular Asynchronous Protocols

Bundled Data Protocols

- Data-path logic can be same as with clocked design
  - combinational logic
  - can have hazards
  - max frequency analysis
- careful design of sequential control
- various handshake protocols
  - no min-delay races due to protocols
- equipotential region for data/req race
Modular Asynchronous Protocols

Three major optimization points of bundled protocols

1. Control Protocol
   - 4-Cycle, 2-Cycle, ...
   - active or passive channels

2. Data
   - broad, narrow, etc.

3. Timing
   - fundamental mode, burst mode
   - relative timing
Modular Asynchronous Protocols

4-cycle return-to-zero protocol

- most common protocol
- two control wire transitions per data
- logic and control state is *level sensitive*
  - channel active when any signal high

4-cycle protocol, early narrow data
Modular Asynchronous Protocols

2-cycle non-return-to-zero protocol

- less common protocol
- one control wire transition per data
- logic and control state is *edge sensitive*
  - channel active when $\text{req} \oplus \text{ack}$
- tradeoff of more complex logic ($\text{xors}$) with fewer control cycles
  - best for long channel communication latency

\[ \text{req} \]
\[ \text{ack} \]
\[ \text{data} \]
Modular Asynchronous Protocols

Pulse-mode protocol

- pulses can be “captured” inside logic
- contains properties of both 2- and 4-cycle circuits
  - level sensitive logic
  - no “redundant” handshakes
- care must be taken to ensure pulses propagate
  - pulses are minimum size (3–5 FO4 delays)
  - wires are pulse filters
- cannot easily determine state by observing channel wires
Modular Asynchronous Protocols

Source Synchronous Protocol

- combination of \textit{wave pipelining} and pulse-mode circuits
  - no \textit{ack} handshake signal!
  - data is latched at receiver
  - pulse and spacing is usually bigger – “clock-like”
- timing issues reduced by slower clock
- overflow controlled at system level or at larger granularity
Data Protocols

Communication interface

- commonly called a channel
- orthogonal data parameters

1. push or pull / active or passive channel interface
   - determines if the data is forwarded or requested

2. data direction
   - data can go either or both directions in a pipeline
   - no additional need for extra control signals
Data Protocols

3. “wide” or “narrow” protocol
   - protocol is wide if data is asserted *entire* handshake cycle while both request and acknowledge are asserted.
   - narrow protocol is when it is asserted from $\frac{1}{4}$ to $\frac{3}{4}$ of the full cycle
   - previous examples use more common narrow protocol
   - determines how and when to latch data
   - does data need to be latched in broad protocol?

4. early or late protocol
   - data can be asserted on rising (early) or falling (late) edges of the handshake signals
Timing Domain - the Third Dimension

- Delay Insensitive (DI): Unbounded wire and gate delay
  - timing is *irrelevant* if physical circuit properties hold
  - no such circuits in reality...

- Quasi Delay Insensitive (QDI): DI with some equipotential wires

- Speed-Independent (SI): Unbounded gate delay, negligible wire delay
  - Wire forks are assumed to be *isochronic*: indistinguishable delay variation between devices on a wire

- Fundamental Mode: Circuit stabilizes between input transitions
  - Huffman SIC circuits and Burst-Mode circuits
Timing Domain - the Third Dimension

- Equipotential Regions
  - delay variation between wires and logic is bounded
- Speed-independent: timing kept inside circuit elements
  - unspecific assumptions necessary for circuit to operate properly
  - necessary for some protocols
Relative Timing

We’ll get there...
Differences in the Protocols

Long distance communication employing different protocol families


Differences in the Protocols

**Timing** is the key differentiator

![Average energy per transfer graph](image)

Differences in the Protocols

Evaluation Parameters matter **a lot**: 
Employing **typical use** activity factors:

![Bandwidth energy graph](attachment:image.png)
Handshake Protocol Comparison

Efficiency points out difference between technologies:
32-bit 10,000µm communication link on 65nm node

- Compares clocked and asynchronous protocols, plus async transmission line
- $x$-axis is bandwidth per unit area
- Diffusive wires have 16 repeaters
  - Replace with flopped repeaters
    - Increases performance, power
- Standard DI technology top left
  - Fulcrum, NULL convention logic, …
- Timed protocols enable power and performance efficiency
  - have $2-3 \times$ better power and area

![Energy for Area Scaled Bandwidth Graph]
Asynchronous Design Properties

- Reactive
  - Don’t wait until sample time occurs
    - Always sampling
- Love operating at different frequencies
  - Each element adjusts to frequency of the environment
  - Can easily combine elements operating at different frequencies
- Modular
  - Simply compose protocols together to build systems
    - (conceptually – why doesn’t this work?)
Circuit Timing Protocol

**Sequencing** largely determined by time

- Relative timing will become natural solution

1. Synchronous Sequencing
   - Global, metranomic
   - Asynchronous state machine is the flip-flop
   - Setup and hold times

2. Asynchronous
   - Handshaking
   - Forward and backward propagation of sequencing
     - Req: Here is work to do
     - Ack: I can/can’t do more work
Circuit Timing Protocol Matrix

- **Sequencing and timing**
  - Traditional clocked design
    - Pros: Easy to validate, CAD
    - Cons: Clock Power
    - Best use: homogeneous designs
  - Timing ref travels with data
    - Pros: low latency, power efficient
    - Cons: CAD support
    - Best use: heterogeneous designs

- **Sequencing only**
  - Not useful

- **Clocks**
  - Derived from time (clock)

- **Self-timed**
  - Derived from data validity

“In the self-timed discipline, sequence and time are connected in the interior of parts called elements.” Mead & Conway, 1980, p218
Evolutionary Convergence

- **Clocked Systems:**
  - skewing clocks to match data delays
  - local clock gating, frequencies, voltages
  - data validity bits, stalling, handshaking
  - mesochronous, GALS, etc.

- **Self-timed systems:**
  - bundled data
  - worst-case & embedded in clock cycle (e.g. SRAM)
  - balanced pipeline delays
  - optimize based on both timing and sequencing
Summary

Design Models: Continuous and Discrete

- Logic Domain: (Voltage)
  - Digital model is discrete
  - Analog model is continuous

- Timing Domain: (Frequency)
  - Clocked model is discrete
  - Asynchronous model is continuous

Sequencing:

- Global vs Local
Exercise and Discussion

1. When would one use a push channel versus a pull channel?