Advances in Designing
Clockless Digital Systems

Prof. Steven M. Nowick

nowick@cs.columbia.edu

Department of Computer Science (and Elect. Eng.)
Columbia University
New York, NY, USA
Synchronous vs. Asynchronous Systems?

- **Synchronous Systems**: use a *global clock*
  - entire system operates *at fixed-rate*
  - uses "centralized control"
Synchronous vs. Asynchronous Systems? (cont.)

- **Asynchronous Systems**: *no global clock*
  - components can operate at *varying rates*
  - *communicate locally via “handshaking”*
  - uses “*distributed control*”

“handshaking interfaces” (channels)
Trends and Challenges

Trends in Chip Design: next decade

- “Semiconductor Industry Association (SIA) Roadmap”

Unprecedented Challenges:

- complexity and scale (= size of systems)
- clock speeds
- power management
- reusability & scalability
- “time-to-market”

Design becoming unmanageable using a centralized single clock (synchronous) approach...
1. **Clock Rate:**

- **1980:** several MegaHertz
- **2001:** ~750 MegaHertz - 1+ GigaHertz
- **2006:** several GigaHertz

**Design Challenge:**

- "*clock skew*": clock must be **near-simultaneous** across entire chip
Trends and Challenges (cont.)

2. Chip Size and Density:

Total #Transistors per Chip: 60-80% increase/year

- ~1970: 4 thousand (Intel 4004 microprocessor)
- today: 50-200+ million
- 2006 and beyond: towards 1 billion+

Design Challenges:

- system complexity, design time, clock distribution
- clock will require 10-20 cycles to reach across chip
3. Power Consumption

- Low power: ever-increasing demand
  - consumer electronics: battery-powered
  - high-end processors: avoid expensive fans, packaging

Design Challenge:

- *clock inherently consumes power continuously*
- “power-down” techniques: complex, only partly effective
4. Time-to-Market, Design Re-Use, Scalability

Increasing pressure for faster “time-to-market”. Need:

- **reusable components**: “plug-and-play” design
- **flexible interfacing**: under varied conditions, voltage scaling
- **scalable design**: easy system upgrades

Design Challenge: mismatch w/ central fixed-rate clock
Trends and Challenges (cont.)

5. Future Trends: “Mixed Timing” Domains

Chips themselves becoming *distributed systems*....

* contain many sub-regions, *operating at different speeds*:

Design Challenge: breakdown of single centralized clock control
Asynchronous Design: Potential Advantages

Several Potential Advantages:

- *Lower Power*
  - *no clock ➔* components use power only “on demand”

- *Robustness, Scalability*
  - *no global timing ➔* “mix-and-match” variable-speed components
  - *composable/modular design style ➔* “object-oriented”

- *Higher Performance*
  - *systems not limited to “worst-case” clock rate*
Asynchronous Design: Recent Industrial Developments

1. Philips Semiconductors:
   - Wide commercial use: 100 million async chips for consumer electronics: pagers, cell phones, smart cards, digital passports, automotive
   - Benefits (vs. sync):
     - 3-4x lower power (and lower energy consumption/ops)
     - much lower “electromagnetic interference” (EMI)
     - instant startup from stand-by mode (no PLL’s)
   - Complete CAD tool flows:
     - “Tangram”: Philips (mid-90’s to early 2000’s)
     - “Haste”: Handshake Solutions (incubated spinoff) (early 2000’s to present)
   - Synthesis strategy: syntax-directed compilation
     - starting point: concurrent HDL (“Tangram”, “Haste”)
     - 2-step synthesis:
       - front-end: HDL spec => intermediate netlist of concurrent components
       - back-end: each component => standard cell (... then physical design)
     - +: fast, ‘transparent’, easy-to-use
     - -: few optimizations, low/moderate-performance only
Asynchronous Design: Recent Industrial Developments

2. Intel:
- experimental Pentium instruction-length decoder = “RAPPID” (1990’s)
- *3-4x faster* than synchronous subsystem
- *~2x lower power*

3. Sun Labs:
- commercial use: high-speed FIFO’s in recent “Ultra’s” (memory access)

4. IBM Research:
- experimental: high-speed pipelines, FIR filters, mixed-timing systems

5. Recent Async Startups:
- Fulcrum Microsystems (California): *Ethernet routing chips*
- Theseus Logic (Orlando): *very low-power/robust designs*
- Handshake Solutions (Netherlands): *incubated by Philips, tools + design*
- Silistrix (UK): *interconnect for heterogenous/mixed-timing systems*
Asynchronous CAD Tools: Recent Developments

DARPA’s “CLASS” Program: (2003-07)
- Major clockless initiative ($14M): to make async commercially viable

Goals:
- CAD tool: produce viable commercial-grade async tool flow
- Demonstration: a complex Boeing ASIC chip

Participants:
- Lead (PI): Boeing
- Industrial participants:
  - Philips (via async incubated startup, “Handshake Solutions”)
  - Theseus Logic, Codetronix
- Academic participants:
  - Columbia, UNC, UW, Yale, OSU

Target: cover wide “design space” – very robust to high-speed circuits
DARPA’s “CLASS” Program: Clockless Initiative (cont.)

Synthesis Flow: targets 3 async circuit styles

(a) Low-speed -- very robust, low-power: NCL circuits (Theseus Logic)
(b) Moderate-speed -- consumer electronics: Haste circuits
   (Philips/Handshake Solutions)
(c) High-speed -- higher-end applications: “Mousetrap” pipelines
   (Nowick, Columbia U.)

Columbia’s role:
* automated optimizations for (a) + (b)
* high-speed pipelines for (c) -- integrated into Philips/Haste CAD tool flow

Goal (potentially): commercial async CAD tool flow
Asynchronous Design: Challenges

- Critical Design Issues:
  * components must *communicate cleanly*: 'hazard-free' design
  * *highly-concurrent designs*: much harder to verify!

- Lack of Automated “Computer-Aided Design” Tools:
  * most commercial “CAD” tools targeted to synchronous
What Are CAD Tools?

Software programs to aid digital designers = "*computer-aided design*" tools

* automatically *synthesize* and *optimize* digital circuits

**Input:**
- desired circuit specification

**Output:**
- optimized circuit implementation
Lack of Existing Asynchronous Design Tools:

- Most commercial “CAD” tools targeted to synchronous

- Synchronous CAD tools:
  - major drivers of growth in microelectronics industry

- Asynchronous “chicken-and-egg” problem:
  - few CAD tools ↔ less commercial use of async design
  - especially lacking: tools for designing/optmzng. large systems
Overview: My Research Areas

- **CAD Tools/Algorithms for Asynchronous Controllers (FSM’s)**
  - “MINIMALIST” Package: for synthesis + optimization

- **CAD Tools/Algorithms for Large-Scale Async Systems**

- **Mixed-Timing Interface Circuits:**
  - for interfacing sync/sync and sync/async systems

- **High-Speed Asynchronous Pipelines:**
  - for static or dynamic logic
CAD Tools for Async Controllers

MINIMALIST: developed at Columbia University [1994-]
- extensible CAD package for synthesis of asynchronous controllers
- integrates synthesis, optimization and verification tools
- used in 80+ sites/17+ countries (was taught in IIT Bombay)
- URL: http://www.cs.columbia.edu/async
- ... new release: expected early 2007 (or contact me)

Includes several optimization tools:
- State Minimization
- CHASM: optimal state encoding
- 2-Level Hazard-Free Logic Minimization
- Verilog back-end
- Verification: spec vs. implementation

Recent application: space measurement chip
- joint funded project: NASA/Columbia (2006-2007)
- fabricated experimental chip: taped out (Oct. 06)

Key goal: facilitate design-space exploration
Example: “PE-SEND-IFC” (HP Labs)

Inputs:
- req-send
- treq
- rd-iq
- adbld-out
- ack-pkt

Outputs:
- tack
- peack
- adbld

From HP Labs
“Mayfly” Project:
B. Coates, A. Davis, K. Stevens,
"The Post Office
Experience: Designing a
Large Asynchronous Chip",
INTEGRATION: the
VLSI Journal, vol. 15:3,
EXAMPLE (cont.):

Design-Space Exploration using MINIMALIST: optimizing for area vs. speed
Overview: My Research Areas

- **CAD Tools/Algorithms for Asynchronous Controllers (FSM’s)**
  - “MINIMALIST” Package: for synthesis + optimization

- **CAD Tools/Algorithms for Large-Scale Async Systems**

- **Mixed-Timing Interface Circuits:**
  - for interfacing sync/sync and sync/async systems

- **High-Speed Asynchronous Pipelines:**
  - for static or dynamic logic
Existing Asynchronous CAD Frameworks

For large async systems:

- **Tangram**: Philips Semiconductors (since mid-1980’s)
  -- developed in research labs (van Berkel, et al.)
  -- commercial use in product divisions (several countries)

- **Haste**: Handshake Solutions (incubated Philips spinoff)
  -- commercial use

Starting point: high-level behavioral system specification

- use concurrent program language (based on CSP)
- features: block-structured, algorithmic, models concurrency

End point: VLSI circuit implementation (layout)
Asynchronous CAD Frameworks

Commercial applications:

- **Tangram**: microcontroller chips, error correctors, ...
  - in several commercial Philips products:
    - ==> smartcards, pagers, cell phones, automotive, digital passports

- **Haste**: entire ARM processors, ... (likely to be offered by ARM Ltd.)

Many sophisticated tool features:

- profilers, early estimation tools (power, delay), testing
- Benefits: rapid development, ease-of-design

**History**: based on “Macromodules Project” (Clark/Molnar, Wash. U., 1960’s)
2 main synthesis steps

- **Syntax-directed translation:**
  - start with concurrent “program” = system specification
  - translate to intermediate network of handshake components

- **Template-based mapping:**
  - map each handshake component directly into library modules

- **Advantages:**
  - Can synthesize large systems
  - Good runtime $\Rightarrow$ syntax-directed compilation
  - “Transparency”: final circuit is predictable, matches spec!

- **Disadvantages:**
  - Few optimizations!: circuits often have poor performance
  - Some peephole optimizations...: only localized, limited
Basic Automated Compiler Flow: Tangram

- Basic Automated Compiler Flow:
  - TANGRAM/HASTE PROGRAM
  - "HANDSHAKE CIRCUIT"
  - MAPPED IMPLEMENTATION

Concurrent specification
Intermediate representation
Final VLSI circuit

- syntax-directed translation (unoptimized)
- template-based mapping
**Automatic Synthesis: a Small Example**

2-Place “Ripple Register” (= FIFO) [van Berkel]

---

**Tangram Program**

```plaintext
proc (a?T & b!T)
  begin
    x0, x1: var T
    | forever do
      b! x1;
      x1 := x0;
    a? x0
  od
end
```

---

**Intermediate “Handshake Circuit”**

---

**Syntax-directed translation** (with no optzns.)
A Larger Example

Intermediate “Handshake Circuit”

Research Goal:
To develop a powerful optimizer for these “intermediate circuits”
New Synthesis Flow: with Back-End Optimizer

TANGRAM/HASTE PROGRAM

INIT “HANDSHAKE CIRCUIT”

BACK-END OPTIMIZER

FINAL “HANDSHAKE CIRCUIT”

Intermediate Representation (unoptimized)

Intermediate Representation (optimized)

syntax-directed translation

new transforms

MAPPED IMPLEMENTATION

RESYNTHESIZED IMPLEMENTATION

#29
Contributions

Provide powerful optimizing back-end for Tangram compiler

**Goal** = facilitate design-space exploration

- **New transformations:** overcome lack of global, aggressive optimizations in syntax-directed compilation
  - **Resynthesis:**
    - cluster handshake components
    - change “interleavings” on handshake interfaces
  - **Peephole:**
    - Template-based replacement
Background: Handshake Components

#1. Active Port: *initiates communication*

#2. Passive Port: *responds to communication*
Components communicate using “4-phase handshaking”

- **O1**: *initiates* communication
- **O2**: *completes* communication

Channel impltn. => use 2 wires:
- req => start operation
- ack => operation done

(... can be extended to handle data)
2-Way Sequencer: activated on channel P; then activates 2 processes in sequence on channels A1 and A2

Goal: activate two sequential processes (i.e. operations)
**Basic Handshake Components: PAR Component**

**PAR Component:** *activated* on channel $P$; then *activates 2 processes in parallel* on channels $A1$ and $A2$

**Goal:** activate two parallel processes

Diagram:
- PAR component with lines connecting to channels $A1$ and $A2$.
- Operation: $X1 \parallel X2$
Basic Handshake Components: MIXER (multiplexer)

2-Way “MIXER”: *activated* on *either* channel A1 *or* A2; then *activates process* on channel B

Goal: facilitate resource sharing between 2 mutually-exclusive processes
WHILE Module: activated on channel A;
repeat { while loop variable TRUE on channel B, activate loop body on channel C}

Goal: control "while loop" operation
Basic Automated Compiler Flow

TANGRAM/BALSA PROGRAM

'HANDSHAKE CIRCUIT'

syntax-directed translation (unoptimized)

template-based mapping

Concurrent specification

Intermediate representation

VLSI circuit

MAPPED IMPLEMENTATION
Synthesizing a System: a Small Example

2-Place “Ripple Register” (= FIFO)

Tangram Program

\[
\text{proc (a?T & b!T)}
\begin{align*}
& \text{begin} \\
& \text{\hspace{1em} x0, x1: var T} \\
& \text{\hspace{1em} | forever do} \\
& \text{\hspace{2em} b! x1;} \\
& \text{\hspace{2em} x1 := x0;} \\
& \text{\hspace{2em} a? x0} \\
& \text{\hspace{1em} od} \\
& \text{end}
\end{align*}
\]

Intermediate “Handshake Circuit”

syntax-directed translation (unoptimized)
procedure Buf1 (  
    input i: byte;
    output o: byte) is
local variable x : byte
begin
    loop begin
        i -> x ;
        o <- x
    end
end

Tangram Spec Syntax-Directed Translation unoptimized
Handshake Circuit
procedure Buf1 (input i: byte;
               output o: byte) is
begin
  local variable x : byte
  loop begin
    i -> x ;
    o <- x
  end
end

“Forever” component: unending loop

Start
procedure Buf1 (input i: byte;
    output o: byte) is
    local variable x : byte
begin
    loop begin
        i -> x ;
        o <- x
    end
end

“Sequencer”: sequences the two assignments

Handshake Circuits: Intermediate Representation

Tangram Spec Syntax-Directed Translation

Handshake Circuit unoptimized
procedure Buf1 ( 
    input i: byte;
    output o: byte) is
local variable x : byte
begin
    loop begin
        i -> x ;
        o <- x
    end
end

"Transferer": transfers data from source to destination

Tangram Spec

Syntax-Directed Translation

Handshake Circuit

unoptimized

Handshake Circuits: Intermediate Representation
procedure Buf1 (  
    input i: byte;  
    output o: byte) is  
local variable x : byte  
begin  
    loop begin  
      i -> x ;  
      o <- x  
    end  
end  

Tangram Spec  

Handshake Circuit  

unoptimized  

Syntax-Directed Translation  

Destination: internal variable X  
Source: interface port I  

Start  

O  

I  

X
procedure Buf1 (  
    input i: byte;  
    output o: byte) is  
local variable x : byte  
begin  
  loop begin  
    i -> x ;  
    o <- x  
  end  
end  

“Transferer”: transfers data from source to destination

Tangram Spec  

Syntax-Directed Translation

Handshake Circuit

unoptimized
procedure Buf1 (  
    input i: byte;
    output o: byte) is
local variable x : byte
begin
  loop begin
    i -> x ;
    o <- x
  end
end

Handshake Circuits: Intermediate Representation

Source: internal variable X
Destination: output port o

Tangram Spec → Syntax-Directed Translation
Handshake Circuit
unoptimized
New Synthesis Flow: with Back-End Optimizer

- **TANGRAM/HASTE PROGRAM**
- **INIT “HANDSHAKE CIRCUIT”**
- **BACK-END OPTIMIZER**
- **FINAL “HANDSHAKE CIRCUIT”**
- **MAPPED IMPLEMENTATION**
- **RESYNTHESIZED IMPLEMENTATION**

- syntax-directed translation *(unoptimized)*
- new transforms

**Intermediate Representation** *(unoptimized)*

**Intermediate Representation** *(optimized)*
New Optimizations: an Overview (part 1)

Resynthesis Transform: cluster groups of components
New Optimizations: an Overview [part 2]

Before optimizations

After optimizations

Datapath components

Control components

Peephole Transform: replace components in “window”
Overview: My Research Areas

- **CAD Tools/Algorithms for Asynchronous Controllers (FSM’s)**
  - “MINIMALIST” Package: for synthesis + optimization

- **CAD Tools/Algorithms for Large-Scale Async Systems**

- **Mixed-Timing Interface Circuits:**
  - for interfacing sync/sync and sync/async systems

- **High-Speed Asynchronous Pipelines:**
  - for static or dynamic logic
Mixed-Timing Interfaces: Challenge

**Goal:** provide low-latency communication between "timing domains"

**Challenge:** avoid synchronization errors
**Mixed-Timing Interfaces: Solution**

Solution: insert mixed-timing FIFO’s ⇒ provide safe data transfer

... developed complete family of mixed-timing interface circuits

Overview: My Research Areas

- **CAD Tools/Algorithms for Asynchronous Controllers (FSM’s)**
  - “MINIMALIST” Package: for synthesis + optimization

- **CAD Tools/Algorithms for Large-Scale Async Systems**

- **Mixed-Timing Interface Circuits:**
  - for interfacing sync/sync and sync/async systems

- **High-Speed Asynchronous Pipelines:**
  - for static or dynamic logic
High-Speed Asynchronous Pipelines

NON-PIPELINED COMPUTATION:

"datapath component" = adder, multiplier, etc.

SYNCHRONOUS
High-Speed Asynchronous Pipelines

“PIPELINED COMPUTATION”: like an assembly line

SYNCHRONOUS

no global clock

ASYNCHRONOUS
High-Speed Asynchronous Pipelines

Goal: fast async datapath components

∗ speed: comparable to fastest existing synchronous designs
∗ additional benefits:
  ∗ dynamically adapt to variable-speed interfaces: handles dynamic voltage scaling
  ∗ “elastic” processing of data in pipeline
  ∗ no requirement of equal-delay stages
  ∗ no clock distribution

Contributions: 3 New Async Pipeline Styles [SINGH/NOWICK]

(i) MOUSETRAP: static logic [ICCD-01, TVLSI-07 (to appear)]
(ii) Lookahead (LP): dynamic logic [Async-02, TVLSI-07 (to appear)]
(iii) High-Capacity (HC): dynamic logic [Async-02, ISSCC-02, TVLSI-07 (to appear)]

Obtain multi-GigaHertz speeds:

- (iii) used by IBM: for experimental fabricated FIR filter chip for disk drives [ISSCC-02]
- (i): currently incorporated into Philips experimental async CAD tool flow
MOUSETRAP: A Basic FIFO (no computation)

Stages communicate using transition-signaling:

[Singh/Nowick, IEEE Int. Conf. on Computer Design (2001)]
“MOUSETRAP” Pipeline: w/computation

Function Blocks: use “synchronous” single-rail circuits (not hazard-free!)

“Bundled Data” Requirement:
* each “req” must arrive after data inputs valid and stable
Recent Research
Recent Research: Asynchronous CAD Tools/Algorithms

CAD Tools/Optimizations for Very Robust Async Circuits
- Cheoljoo Jeong
  - Collaboration with Orlando-based startup: Theseus Logic
  - Low-power applications
  - CAD tools: multi-level logic optimization, technology mapping
  - Circuit improvements: >40% speed, >20% area reduction
  - Technology transfer: ongoing

CAD Tools for Async Controller Decomposition
- Melinda Agyekum
  - Goal = improved runtime during synthesis
  - CAD tools: partitioning large/complex controllers
  - Over 1000x runtime improvement
Goal: fast analytical techniques + tools
- to handle large/complex asynchronous + mixed-timing systems
  * using stochastic delay models (Markovian): [McGee/Nowick, CODES-05]
  * using bounded delay models (min/max): work in progress

Applications: system-level analysis + optimization

- Large Async Systems:
  * Evaluate latency, throughput, critical vs. slack paths, average-case rating
  * Drive optimization: pipeline granularity, module selection

- Large Heterogeneous (mixed-clock) or “GALS” Systems:
  * Evaluate critical vs. slack paths, buffer requirements
  * Drive optimization: dynamic voltage scaling, load balancing of threads

- Peggy McGee
Recent Research: Collaborations

NASA (Goddard Space Center): space measurement equipment
  * Uses our CAD tools/circuit styles for async controllers
  * Joint chip design: Nowick + NASA manager
  * Prototype chip #1: back from fab
  * Prototype chip #2: Summer 07
  * Funding: from NASA (2007)

High-Throughput Async Interconnect: for “supercomputer-on-chip”
  * Collaboration with parallel architectures/algorithms group: U. of Maryland
  * Goal: very flexible, low-power interconnect = CPU’s <--> caches
  * Funding: from Columbia Exec. VP of Research (2006-2008)
    * Matt Carlberg (undergrad)
**CAD Tools for Large-Scale Asynchronous Systems**

**Input Specification:**
= "Control Data-flow Graph"

**Target Architecture:**
control unit

- Controller 1
- Controller 2
- Controller 3

**Target:**
- Synthesize distributed control
- 1 controller per functional unit

[Theobald/Nowick, IEEE Design Automation Conf. (2001)]
Other Handshake Components: Data Operations

**REGISTER:**
Goal: data storage (with read/write ports)

**TRANSFERER:**
Goal: transfer data

**ADDER, MULTIPLIER:**
Goal: do addition, mult
Part IV

A Back-End Optimizer
MOUSETRAP: A Basic FIFO

Stages communicate using \textit{transition-signaling}:

1 \textit{transition per data item!}

One Data Item
Components communicate on a communication channel

- **Component O1**: *initiates* communication = “active”
- **Component O2**: *completes* communication = “passive”