CprE 588
Embedded Computer Systems

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #1 – Introduction and Overview

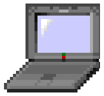
Digital System v. Embedded System

- **Digital System**: may provide service
  - as a self-contained unit (e.g., desktop PC)
  - as part of a larger system (e.g., digital control system for manufacturing plant)
- **Embedded System**:
  - part of a larger unit
  - provides dedicated service to that unit

G. De Micheli and R. Gupta, "Hardware/Software Co-Design," *Proceedings of the IEEE*, 85(3), March 1997, pp. 349-365

Embedded Systems Overview

- Computing systems are everywhere
- Most of us think of “desktop” computers
  - PC’s
  - Laptops
  - Mainframes
  - Servers

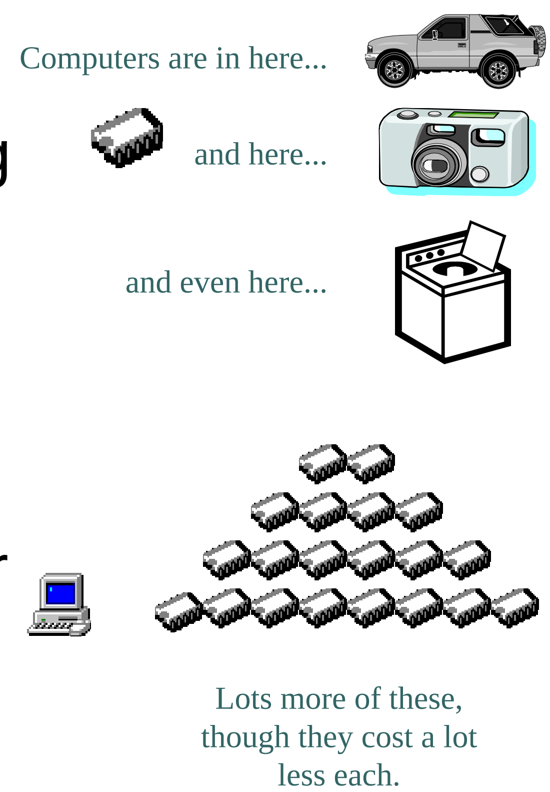


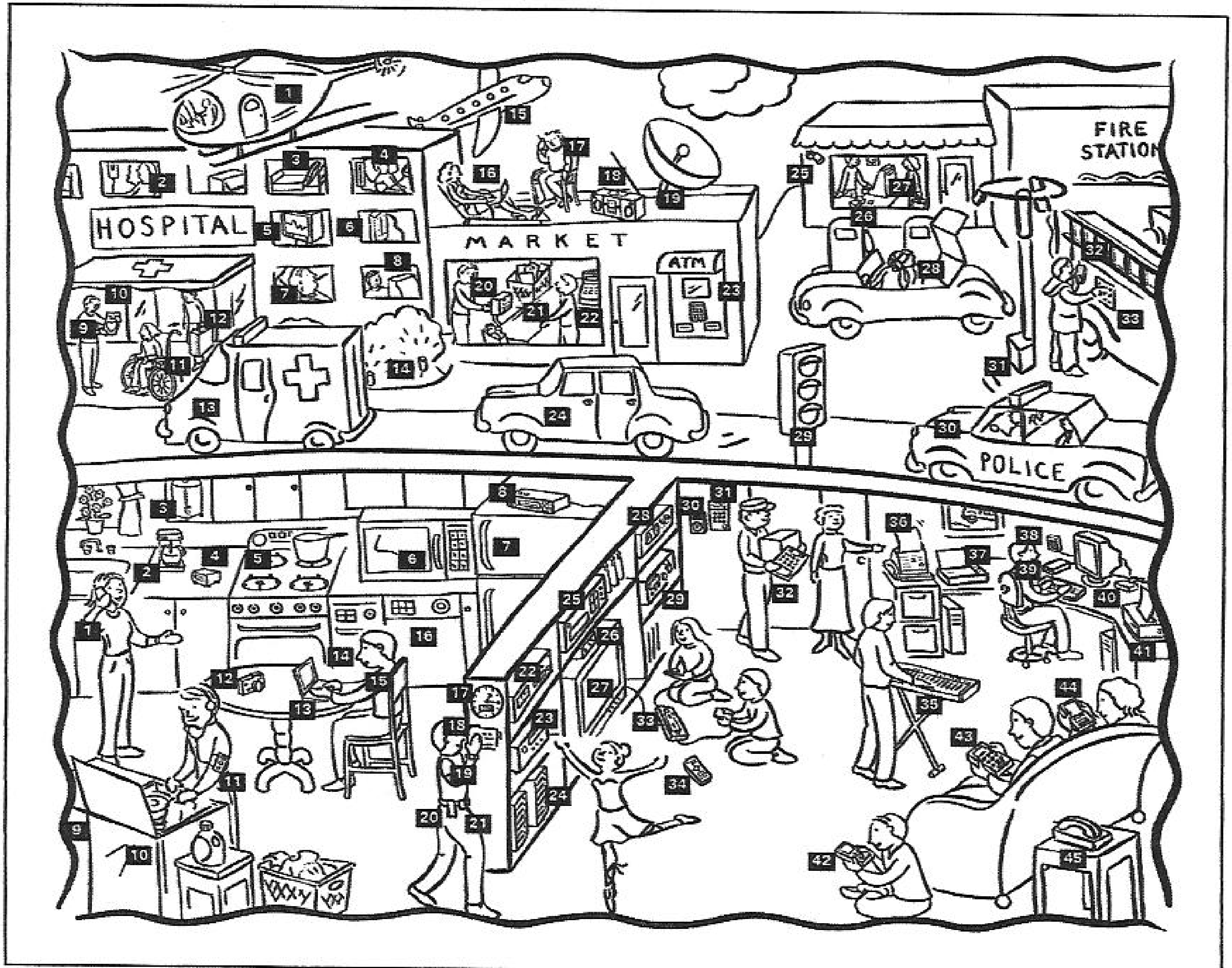
- But there’s another type of computing system
  - Far more common...

F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley & Sons, 2002.

Embedded Systems Overview (cont.)

- Embedded computing systems
  - Computing systems embedded within electronic devices
  - Hard to define. Nearly any computing system other than a desktop computer
  - Billions of units produced yearly, versus millions of desktop units
  - Perhaps 100s per household and per automobile

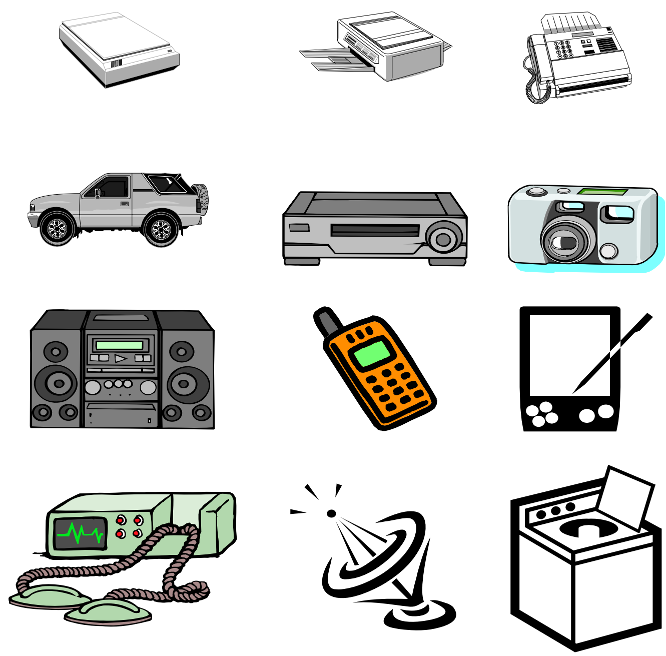



HOSPITAL
MARKET
ATM
FIRE STATION
POLICE

A “Short List” of Embedded Systems

Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Automatic transmission
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Curbside check-in systems
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems

Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR’s, DVD players
Video game consoles
Video phones
Washers and dryers



And the list goes on and on

Examples of Embedded Systems

- PC having dedicated software programs and appropriate interfaces to a manufacturing assembly line
- Microprocessor dedicated to a control function in a computer, e.g., keyboard/mouse input control

Outline

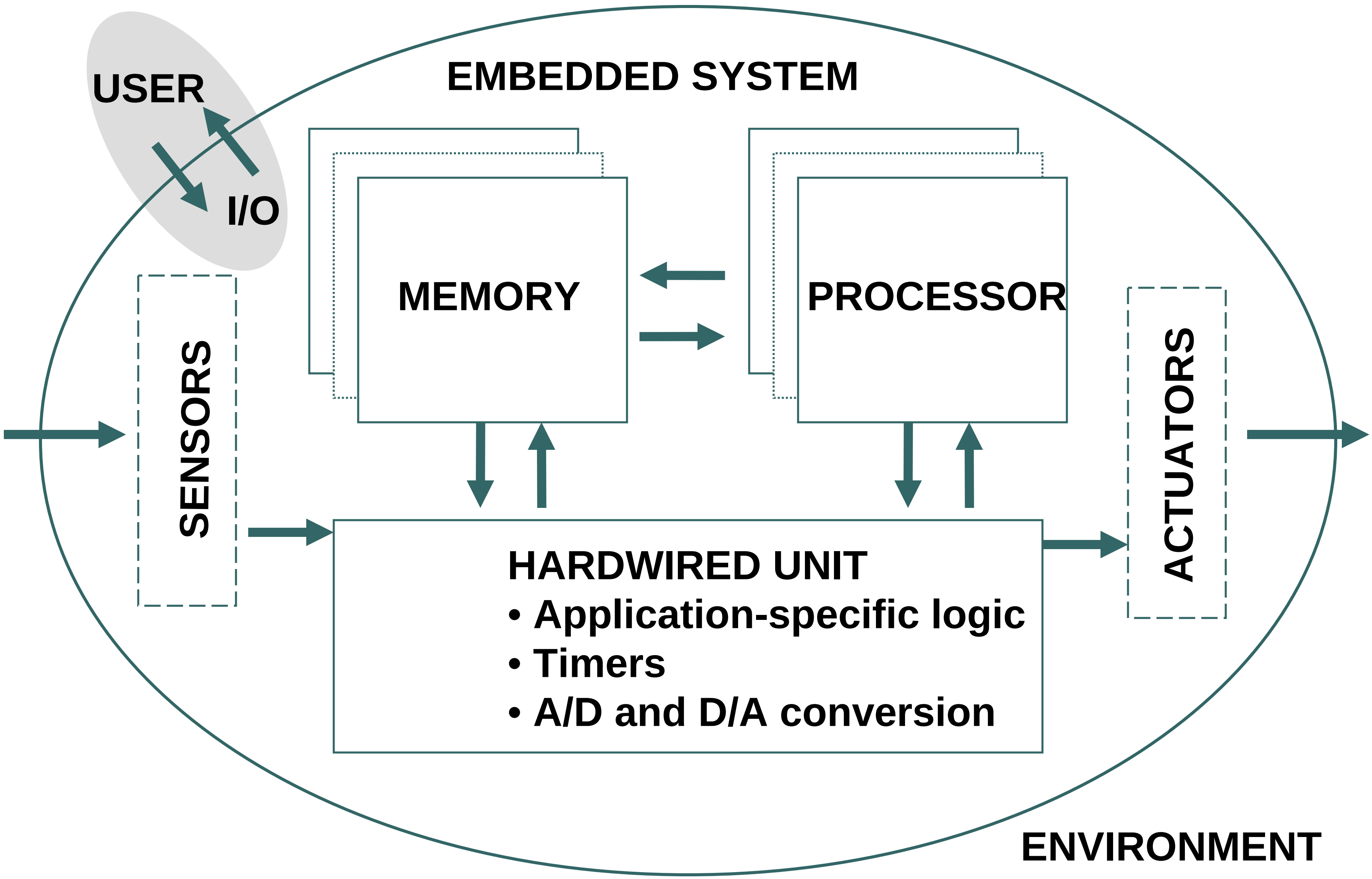
- Embedded systems overview
- Design challenge – optimizing design metrics
- Technologies
  - Processor technologies
  - Design technologies
- Generic codesign methodology

Some Application Domains

- CONSUMER PRODUCTS
  - Appliances, Games, A/V, Intelligent home devices
- TRANSPORTATION
  - Autos, Trains, Ships, Aircrafts
- PLANT CONTROL
  - Manufacturing, Chemical, Power Generation
- NETWORKS
  - Telecommunication, Defense

- Local
  - e.g., appliance
- Locally distributed
  - e.g., aircraft control over a LAN
- Geographically distributed
  - e.g., telephone network

Parts of an Embedded System



Parts of an Embedded System (cont.)

- Actuators - mechanical components (e.g., valve)
- Sensors - input data (e.g., accelerometer for airbag control)
- Data conversion, storage, processing
- Decision-making

- Range of implementation options
- Single-chip implementation: system on a chip

Functions and Design Criteria

- Monitoring and control functions for the overall system (e.g., vehicle control)
- Information-processing functions (e.g., telecommunication system -- data compression, routing, etc.)

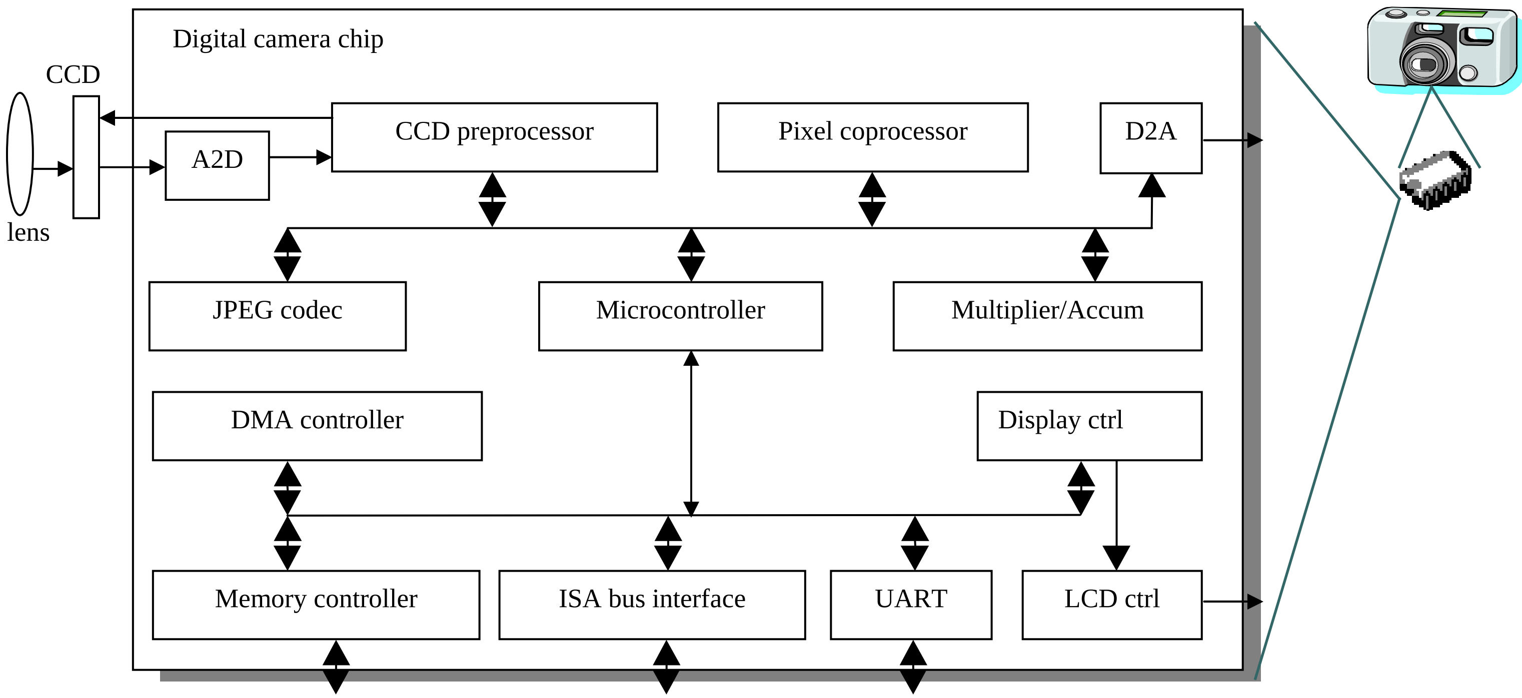
- Criteria: performance, reliability, availability, safety, usability, etc.

Some Common Characteristics

- Single-functioned
  - Executes a single program, repeatedly
- Tightly-constrained
  - Low cost, low power, small, fast, etc.
- Reactive and real-time
  - Continually reacts to changes in the system's environment
  - Must compute certain results in real-time without delay

An Embedded System Example

- Digital Camera



- Single-functioned -- always a digital camera
- Tightly-constrained -- Low cost, low power, small, fast
- Reactive and real-time -- only to a small extent

Design Challenge – Optimization

- Obvious design goal:
  - Construct an implementation with desired functionality
- Key design challenge:
  - Simultaneously optimize numerous design metrics
- Design metric
  - A measurable feature of a system's implementation
  - Optimizing design metrics is a key challenge

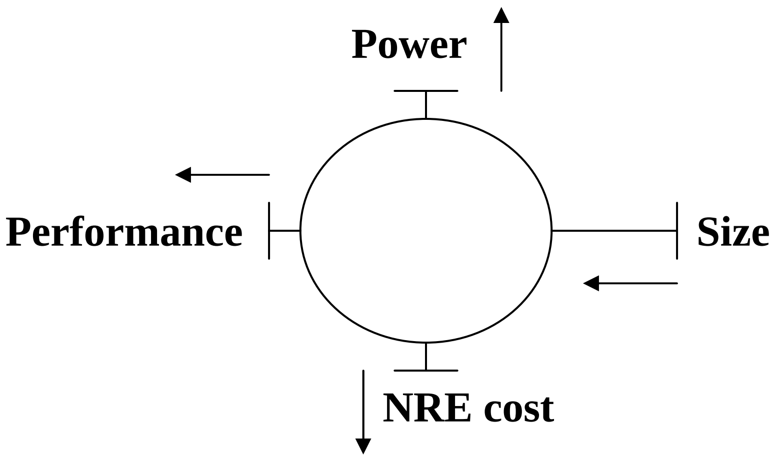
Design Challenge – Optimization (cont.)

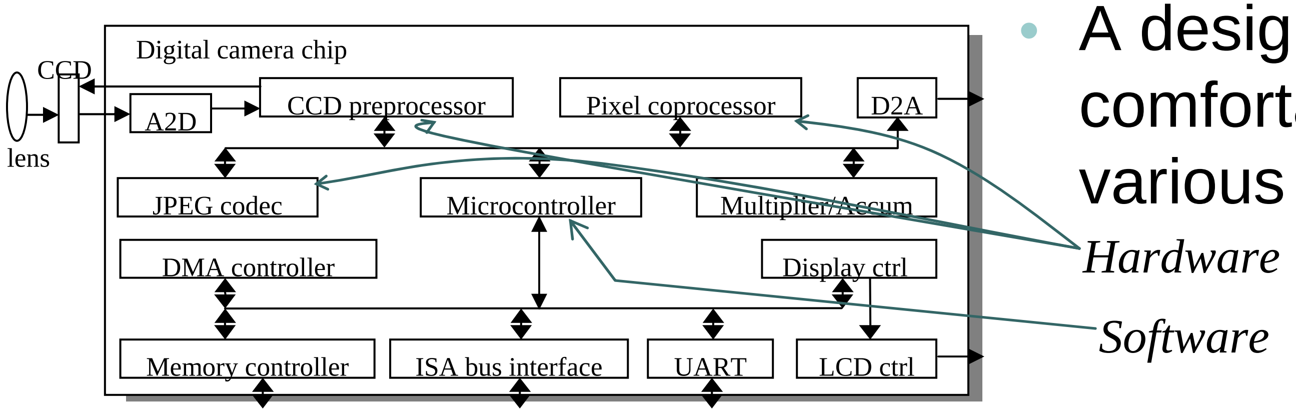
- Common metrics
  - **Unit cost:** the monetary cost of manufacturing each copy of the system, excluding NRE cost
  - **NRE cost (Non-Recurring Engineering cost):** The one-time monetary cost of designing the system
  - **Size:** the physical space required by the system
  - **Performance:** the execution time or throughput of the system
  - **Power:** the amount of power consumed by the system
  - **Flexibility:** the ability to change the functionality of the system without incurring heavy NRE cost

Design Challenge – Optimization (cont.)

- Common metrics (continued)
  - Time-to-prototype: the time needed to build a working version of the system
  - Time-to-market: the time required to develop a system to the point that it can be released and sold to customers
  - Maintainability: the ability to modify the system after its initial release
  - Correctness, safety, many more

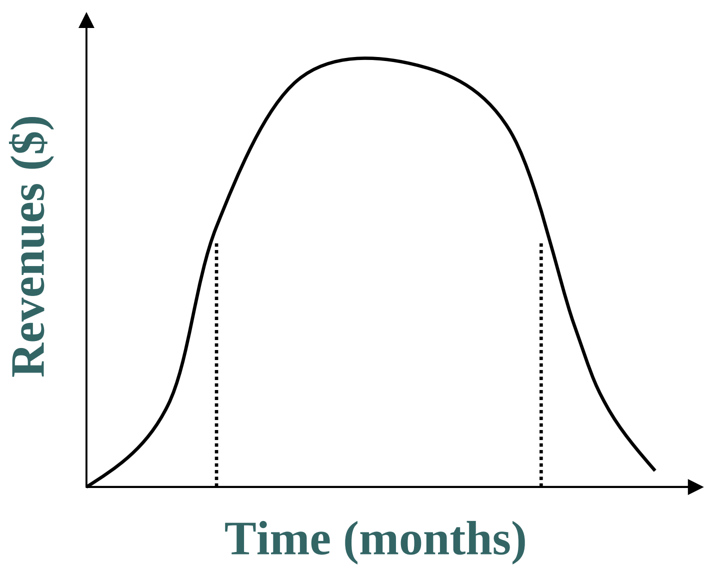
Design Metric Competition





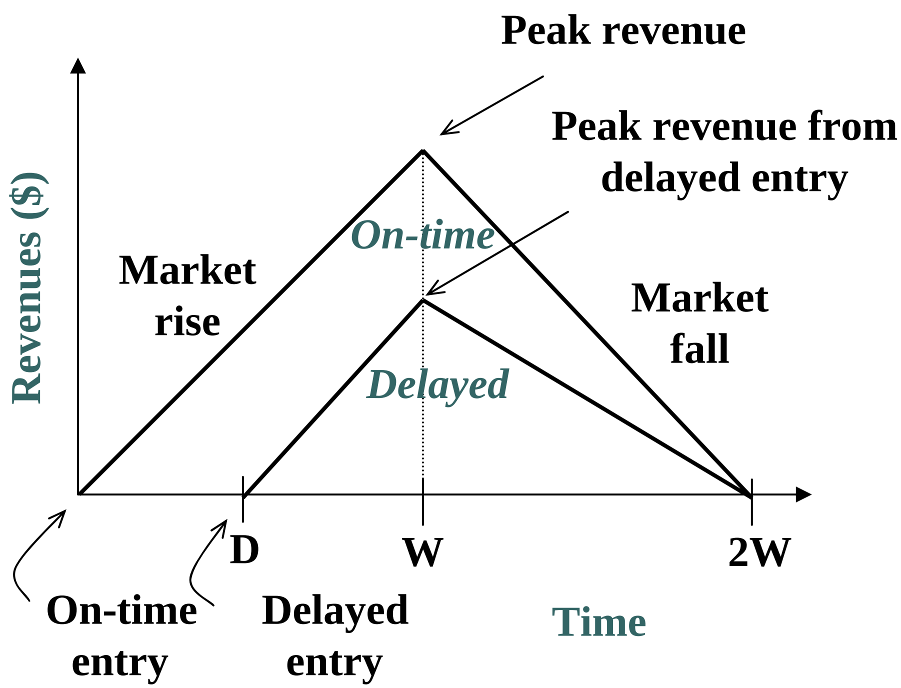
- Expertise with both **software** and **hardware** is needed to optimize design metrics
  - Not just a hardware or software expert, as is common
  - A designer must be comfortable with various technologies

Time-to Market



- Time required to develop a product to the point it can be sold to customers
- Market window
  - Period during which the product would have highest sales
- Average time-to-market constraint is about 8 months
- Delays can be costly

Delayed Market Entry



- Simplified revenue model
  - Product life = 2W, peak at W
  - Time of market entry defines a triangle, representing market penetration
  - Triangle area equals revenue
- Loss
  - The difference between the on-time and delayed triangle areas

NRE and Unit Cost Metrics

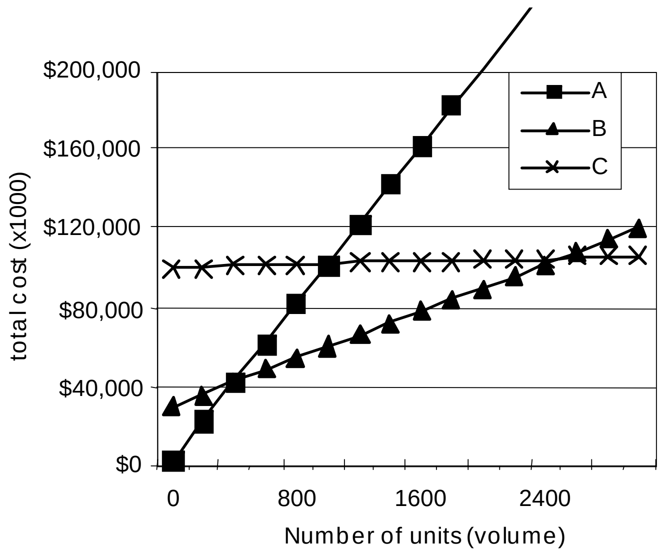
- Costs:
  - Unit cost: the monetary cost of manufacturing each copy of the system, excluding NRE cost
  - NRE cost (Non-Recurring Engineering cost): the one-time monetary cost of designing the system
  - *total cost = NRE cost + unit cost * # of units*
  - *per-product cost = total cost / # of units*
    = (NRE cost / # of units) + unit cost

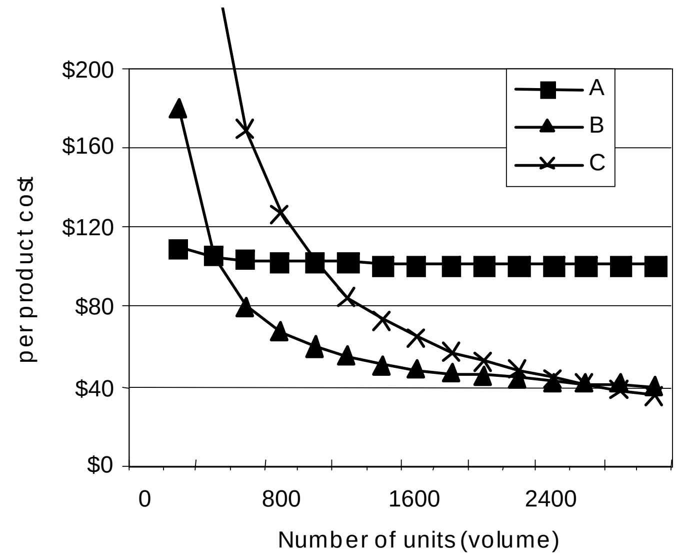
- Example
  - NRE=$2000, unit=$100
  - For 10 units
    - total cost = $2000 + 10*$100 = $3000
    - per-product cost = $2000/10 + $100 = $300

Amortizing NRE cost over the units results in an additional $200 per unit

NRE and unit cost metrics

- Compare technologies by costs -- best depends on quantity
  - Technology A: NRE=$2,000, unit=$100
  - Technology B: NRE=$30,000, unit=$30
  - Technology C: NRE=$100,000, unit=$2





- But, must also consider time-to-market

The Performance Design Metric

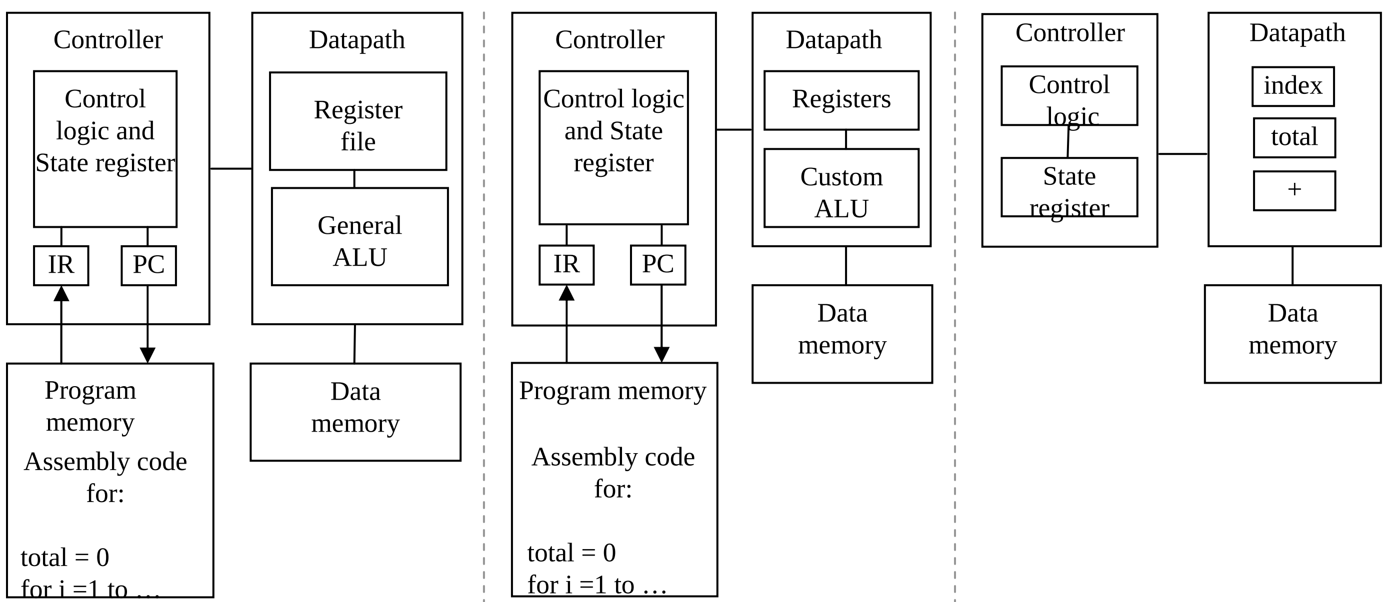
- Widely-used measure of system, widely-abused
  - Clock frequency, instructions per second – not good measures
  - Digital camera example – a user cares about how fast it processes images, not clock speed or instructions per second
- Latency (response time)
  - Time between task start and end
  - e.g., Camera's A and B process images in 0.25 seconds
- Throughput
  - Tasks per second, e.g. Camera A processes 4 images per second
  - Throughput can be more than latency seems to imply due to concurrency, e.g. Camera B may process 8 images per second (by capturing a new image while previous image is being stored).
- *Speedup* of B over S = B's performance / A's performance
  - Throughput speedup = 8/4 = 2

Three Key Technologies

- Technology
  - A manner of accomplishing a task, especially using technical processes, methods, or knowledge
- Three key technologies for embedded systems
  - Processor technology (CprE 581, 583, 681)
  - IC technology (EE 501, 507, 511)
  - Design technology (CprE 588)

Processor Technology

- The architecture of the computation engine used to implement a system’s desired functionality
- Processor does not have to be programmable
  - “Processor” *not* equal to general-purpose processor



General-purpose (“software”) **Application-specific** **Single-purpose** (“hardware”)

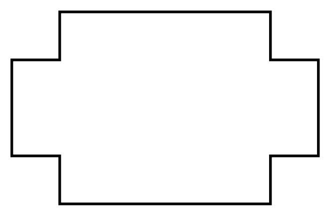
Processor Technology (cont.)

- Processors vary in their customization for the problem at hand

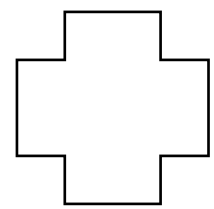
Desired functionality

```
total = 0
for i = 1 to N  loop
   total += M[i]
end loop
```

General-purpose processor



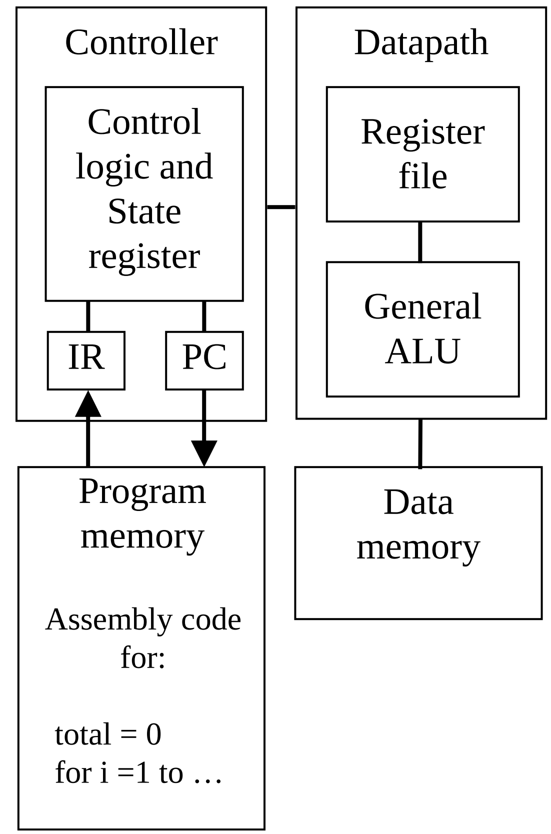
Application-specific processor



Single-purpose processor

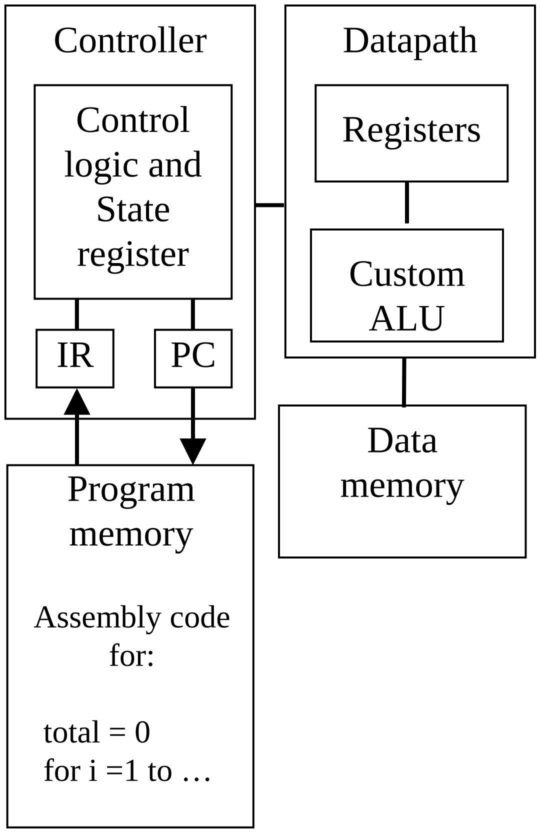
General-Purpose Processors

- Programmable device used in a variety of applications
  - Also known as "microprocessor"
- Features
  - Program memory
  - General datapath with large register file and general ALU
- User benefits
  - Low time-to-market and NRE costs
  - High flexibility
- "Intel/AMD" the most well-known, but there are hundreds of others



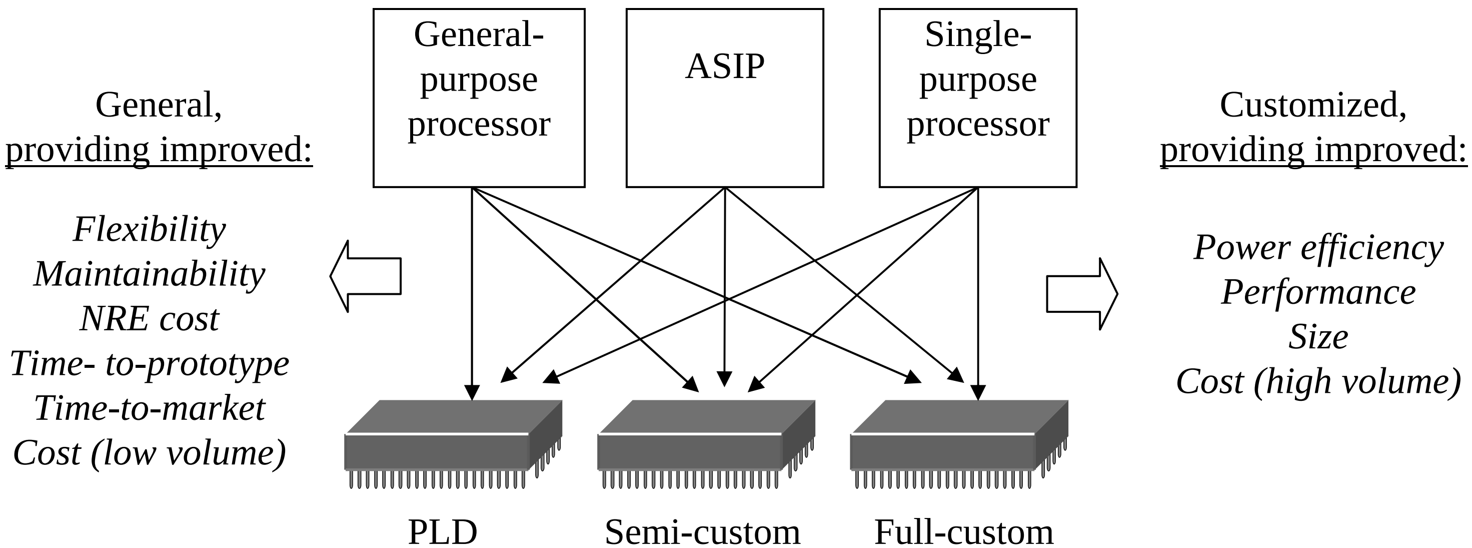
Application-Specific Processors

- Programmable processor optimized for a particular class of applications having common characteristics
  - Compromise between general-purpose and single-purpose processors
- Features
  - Program memory
  - Optimized datapath
  - Special functional units
- Benefits
  - Some flexibility, good performance, size and power



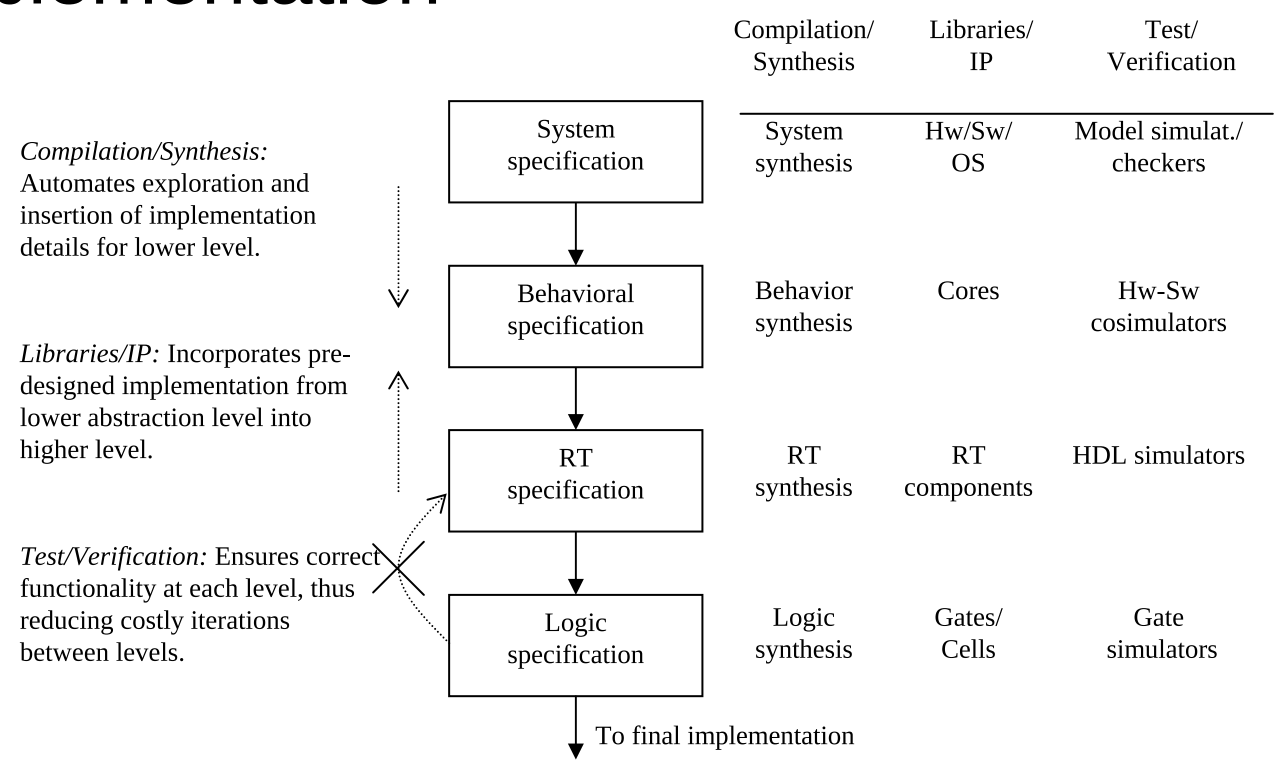
Independence of Processor Technologies

- Basic tradeoff
  - General vs. custom
  - With respect to processor technology or IC technology
  - The two technologies are independent

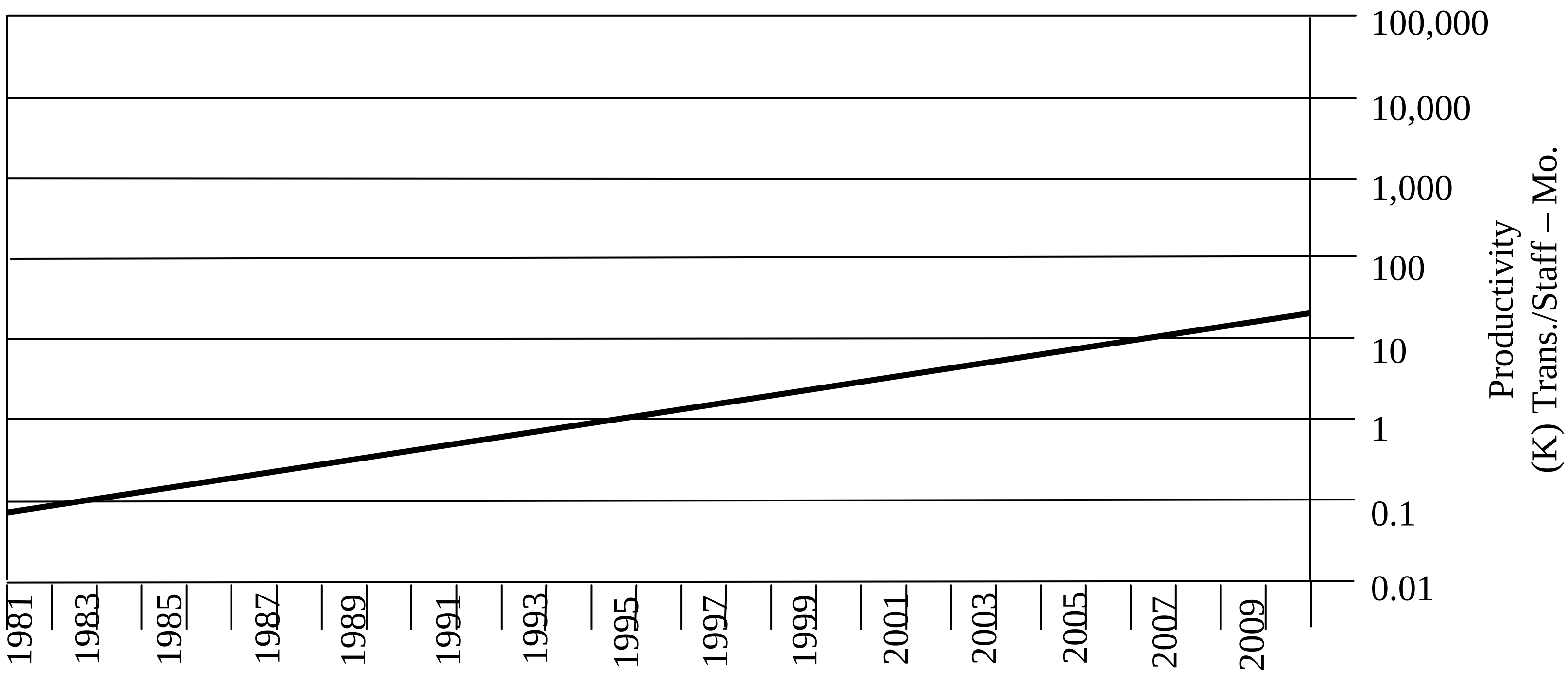


Design Technology

- The manner in which we convert our concept of desired system functionality into an implementation



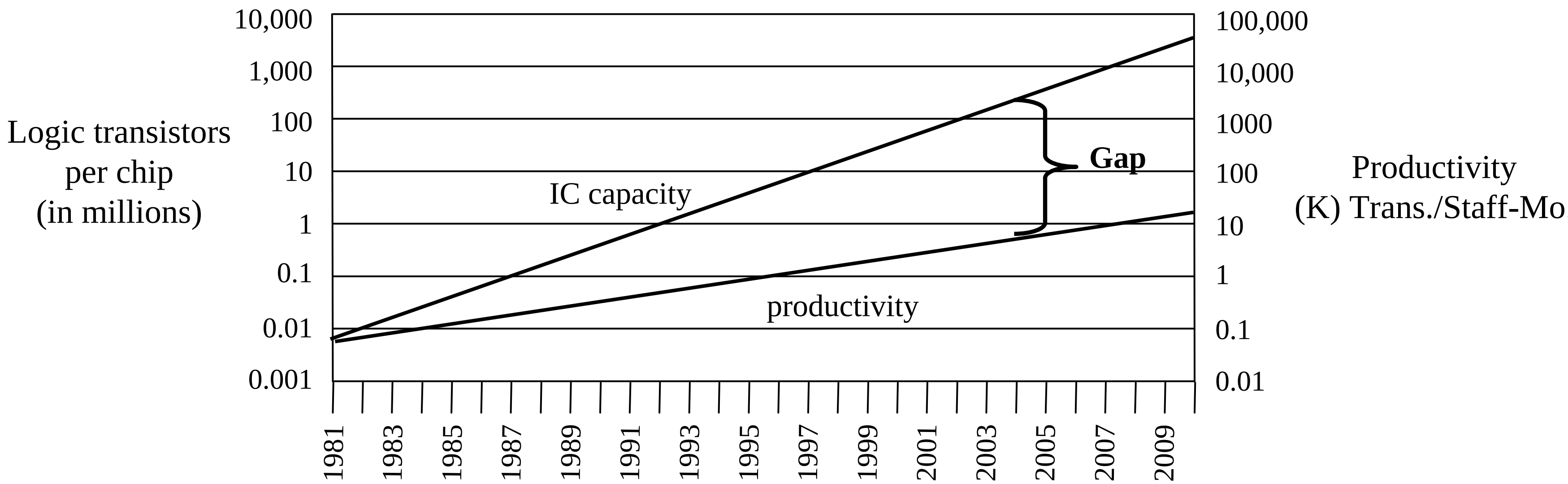
Design Productivity Exponential Increase



- Exponential increase over the past few decades

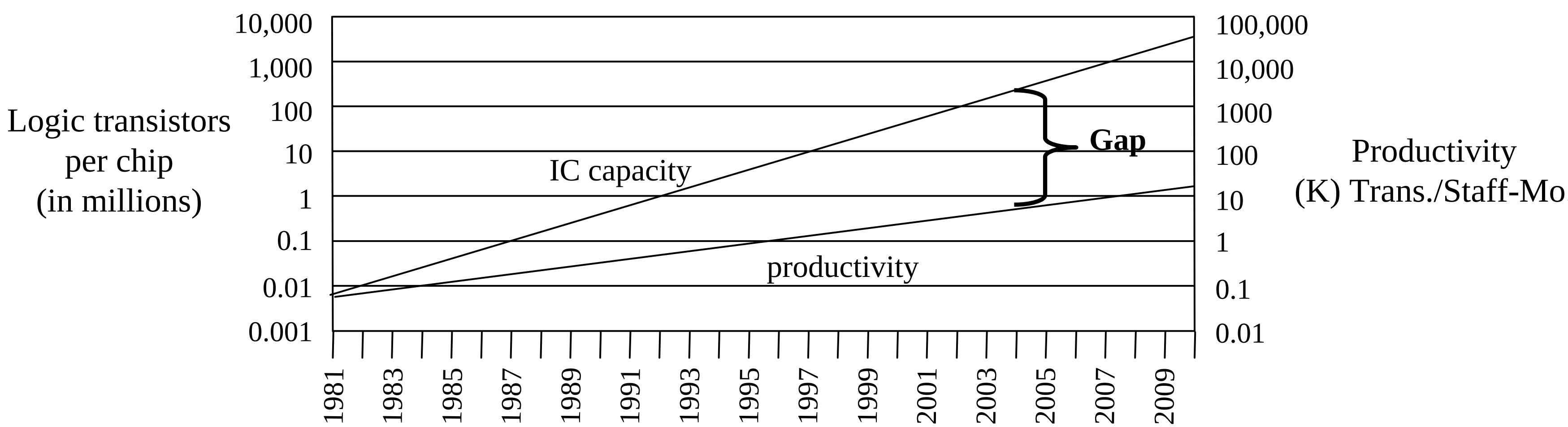
Design Productivity Gap

- While designer productivity has grown at an impressive rate over the past decades, the rate of improvement has not kept pace with chip capacity



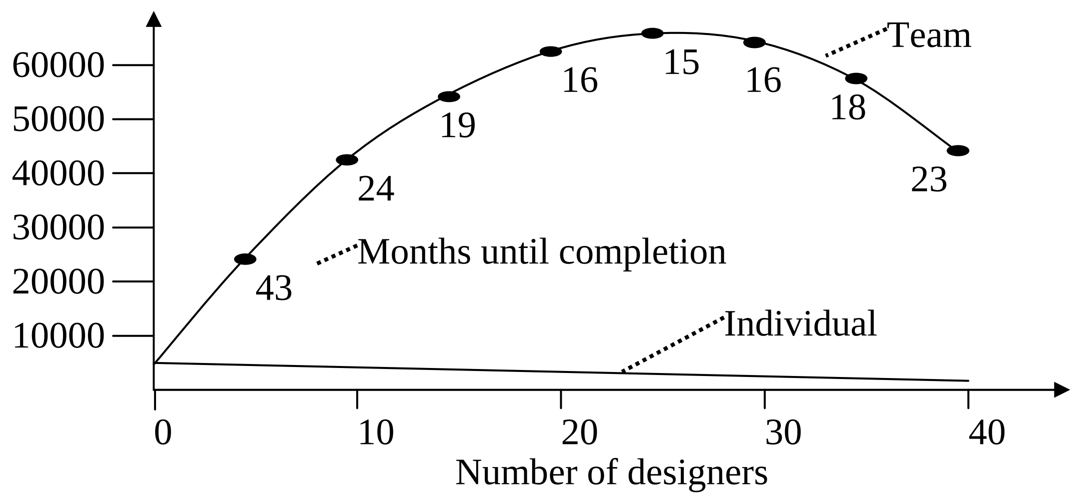
Design Productivity Gap (cont.)

- 1981 leading edge chip required 100 designer months
  - 10,000 transistors / 100 transistors/month
- 2002 leading edge chip requires 30,000 designer months
  - 150,000,000 / 5000 transistors/month
- Designer cost increase from $1M to $300M



The Mythical Man-Month

- The situation is even worse than the productivity gap indicates
- In theory, adding designers to team reduces project completion time
- In reality, productivity per designer decreases due to complexities of team management and communication
- In the software community, known as “the mythical man-month” (Brooks 1975)
- At some point, can actually lengthen project completion time! (“Too many cooks”)
  - 1M transistors, 1 designer=5000 trans/month
  - Each additional designer reduces for 100 trans/month
  - So 2 designers produce 4900 trans/month each



Co-Design Methodology

- **Co-design**
  - Design of systems involving both hardware and software components
  - Starts with formal, abstract specification; series of refinements maps to target architecture: allocation, partitioning, scheduling, communication synthesis
  - Means to manage large-scale, complex systems

R. Domer, D. Gajski, J. Zhu, “Specification and Design of Embedded Systems,” *it+ti magazine*, Oldenbourg Verlag (Germany), No. 3, June 1998.

Complex Systems

- SOC (System-On-a-Chip)
  - Millions of gates on a chip
    - Decreasing processing technologies (deep sub-micron, 0.25 μm and below): decreasing geometry size, increasing chip density
  - Problems
    - Electronic design automation (EDA) tools
    - Time-to-market

Complex Systems (cont.)

- **Abstraction**
  - Reduce the number of objects managed by a design task, e.g., by grouping objects using hierarchy
  - Computer-aided design (CAD) example
    - Logic level: transistors grouped into gates
    - Register transfer level (RTL): gates grouped into registers, ALUs, and other RTL components

Complex Systems (cont.)

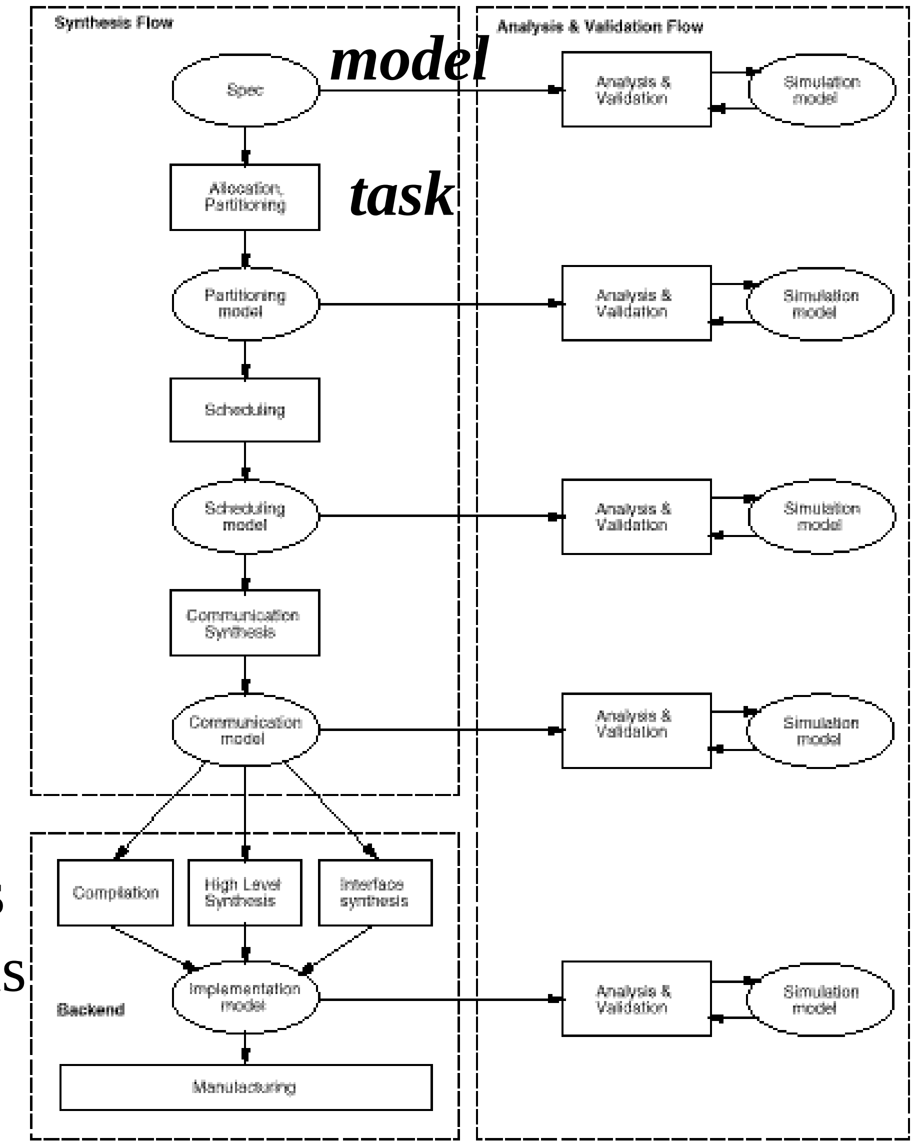
- **Abstraction**
  - Co-design example
    - System level: processors (off-the-shelf or application-specific), memories, application-specific integrated circuits (ASICs), I/O interfaces, etc.
    - Integration of intellectual property (IP) - representations of products of the mind
    - Reuse of formerly designed circuits as core cells

Generic Co-Design Methodology

<u>Synthesis</u>
- Specification
- Allocation
- Partitioning
- Scheduling

- Communication synthesis

<u>Implementation</u>
- Software synthesis
- Hardware synthesis
- Interface synthesis



<u>Analysis & Validation</u>

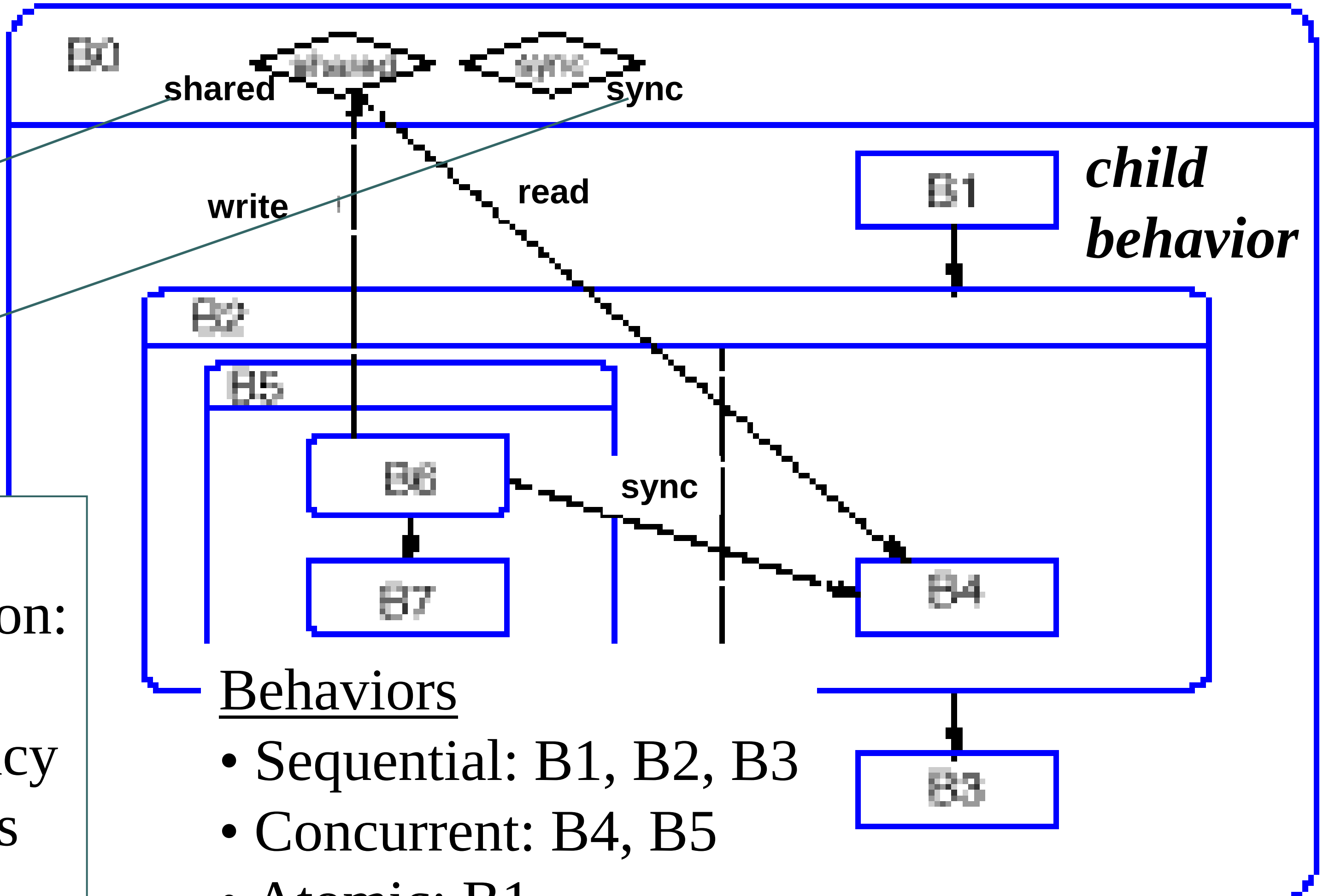
Note: design models may be captured in the same language

System Specification

- Describes the functionality of the system without specifying the implementation
- Describes non-functional properties such as performance, power, cost, and other quality metrics or design constraints
- May be *executable* to allow dynamic verification

System Specification Example

B0: ***top behavior***
- integer variable
- boolean variable



child behavior

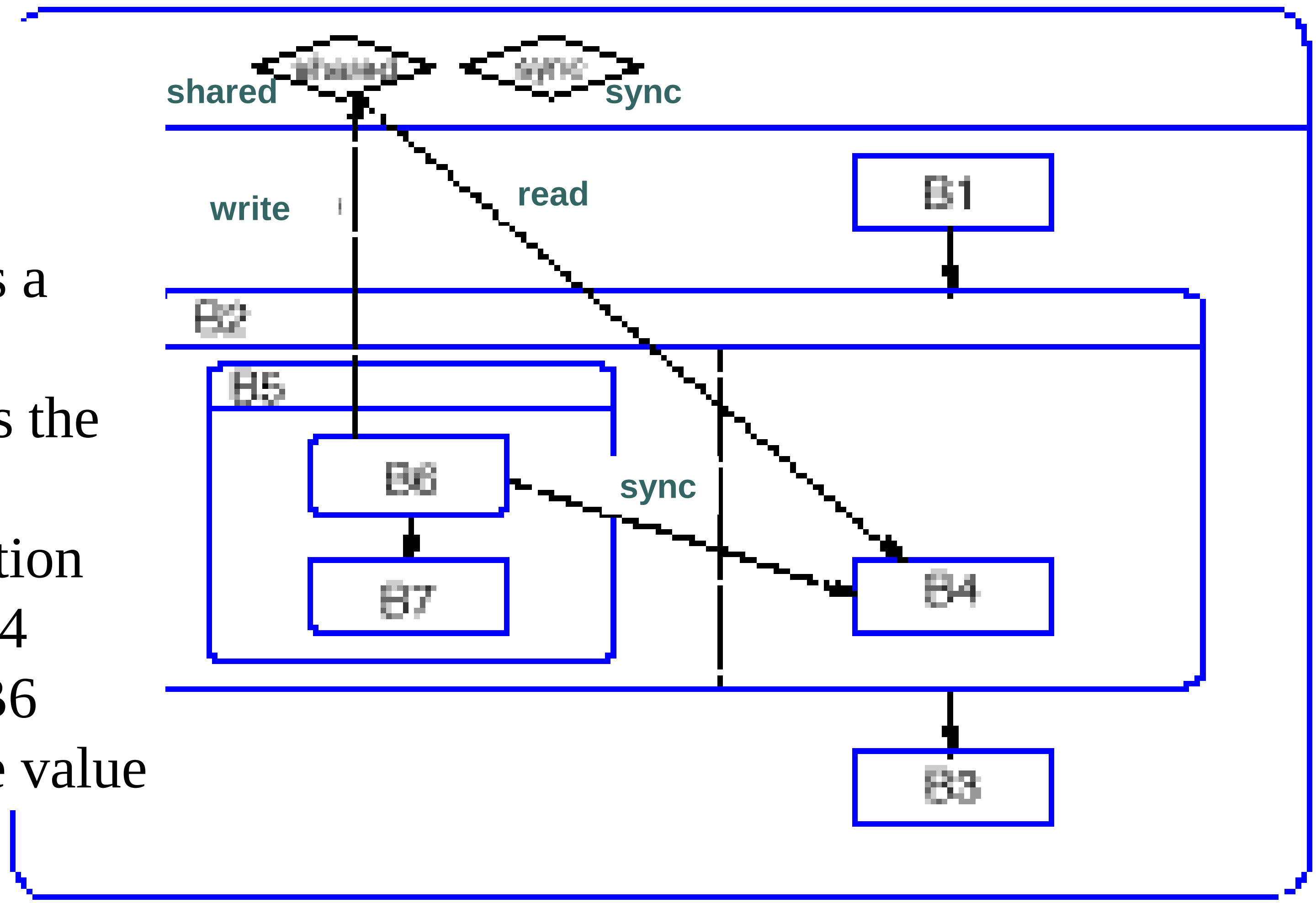
Graphical representation:
- Hierarchy
- Concurrency
- Transitions between behaviors

Behaviors
- Sequential: B1, B2, B3
- Concurrent: B4, B5
- Atomic: B1
- Composite: B2

System Specification Example (cont.)

<u>Producer-consumer functionality</u>

- B6 computes a value
- B4 consumes the value
- Synchronization is needed: B4 waits until B6 produces the value



System Specification Example

- Atomic behaviors

```
B1( )
{
   stmt;
   ...
}
```

```
B3( )
{
   stmt;
   ...
}
```

```
B7( )
{
   stmt;
   ...
}
```

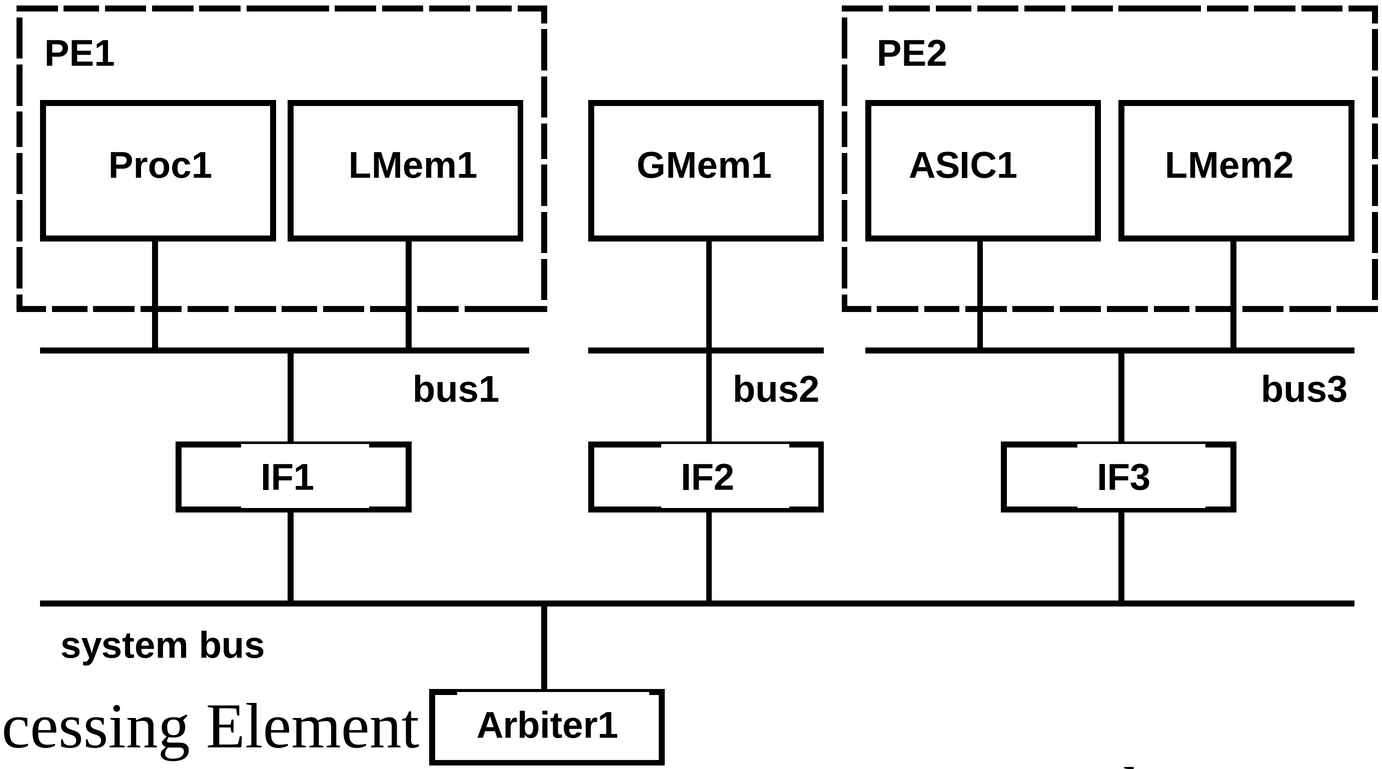
```
B6( )
{
   int local;
   …
   shared = local + 1;
   signal(sync);
}
```

```
B4( )
{
   int local;
   wait(sync);
   local = shared - 1;
   ...
}
```

Allocation

- Selects the type and number of components from a library and determines their interconnection
- Implements functionality so as to
  - Satisfy constraints
  - Minimize objective cost function
- Result may be customization of a generic target architecture

Allocation Example



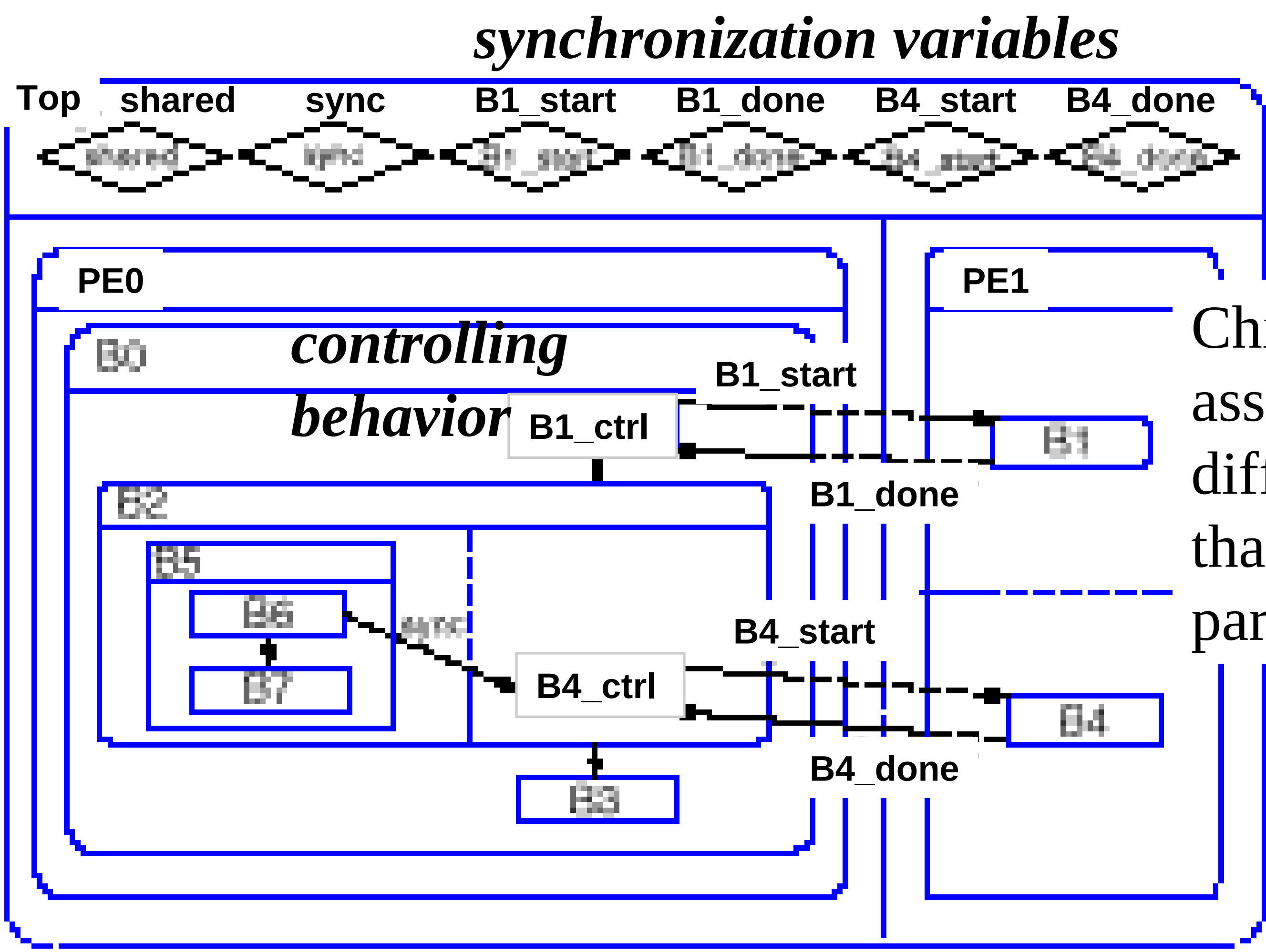
PE: Processing Element
LMem: Local Memory
GMem: Global Memory
IF: Interface

Target Architecture Model

Partitioning

- Defines the mapping between the set of behaviors in the specification and the set of allocated components in the architecture
  - Satisfy constraints
  - Minimize costs
- Not yet near implementation
  - Multiple behaviors in a single PE (scheduling)
  - Interactions between PEs (communication)
- Design model
  - additional level of hierarchy
  - functional equivalence with specification

Partitioning Example



Child (B1) assigned to different PE than parent (B0)

System model after partitioning

Partitioning Example (cont.)

- Atomic behaviors

```
B1( )
{
   wait(B1_start);
   …
   signal(B1_done);
}
```

```
B1_ctrl( )
{
   signal(B1_start);
   wait(B1_done);
}
```

```
B3( )
{
   stmt;
   ...
}
```

```
B7( )
{
   stmt;
   ...
}
```

```
B4( )
{
   int local;
   wait(B4_start);
   wait(sync);
   local = shared - 1;
   …
   signal(B4_done);
}
```

```
B4_ctrl( )
{
   signal(B4_start);
   wait(B4_done);
}
```

```
B6( )
{
   int local;
   …
   shared = local +
     1;
   signal(sync);
}
```

Scheduling

- Given a set of behaviors and optionally a set of performance constraints, determines a total order in time for invoking behaviors running on the same PE
- Maintains the partial order imposed by dependencies in the functionality
- Minimizes synchronization overhead between PEs and context-switching overhead within each PE

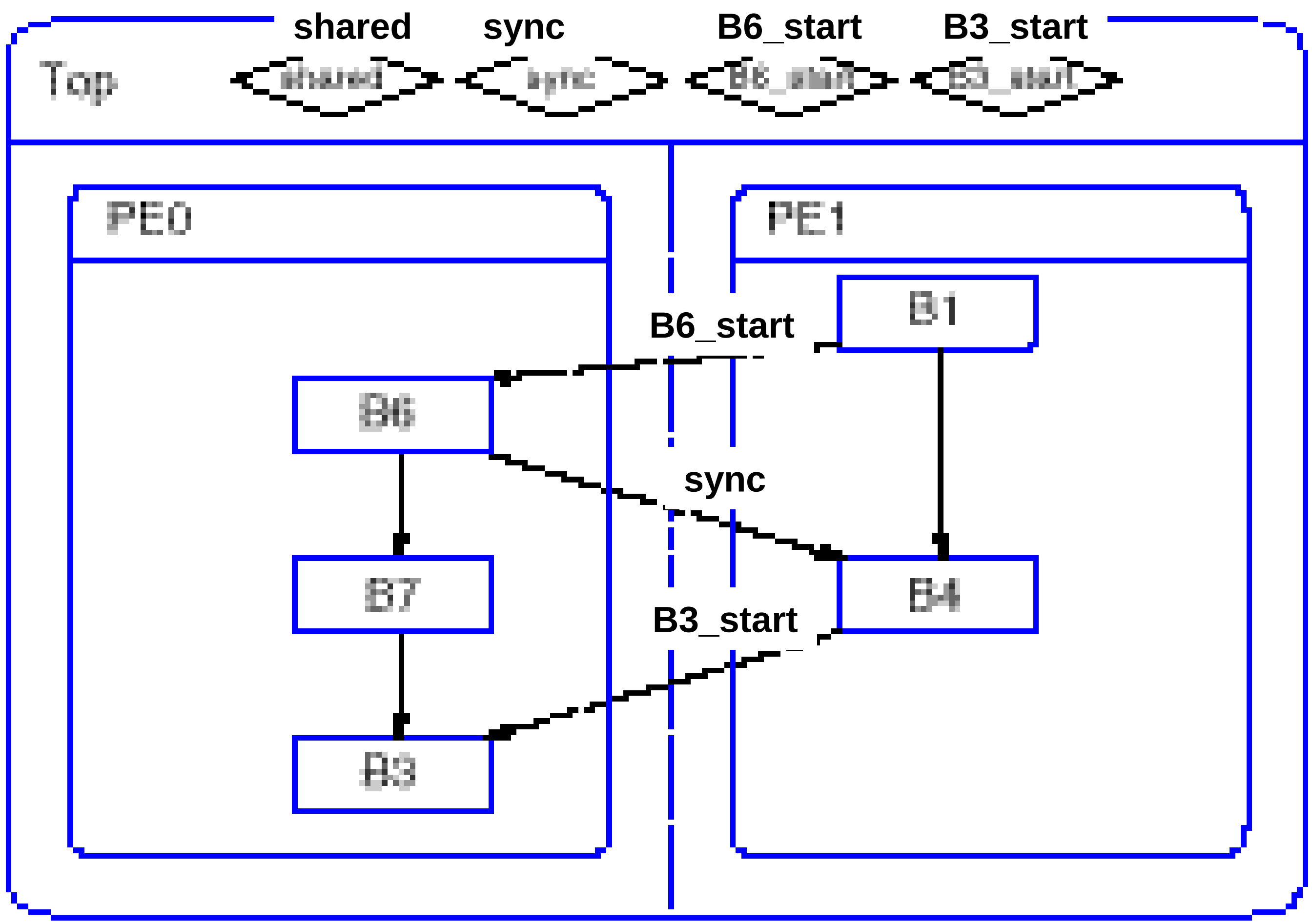
Scheduling

- Ordering information
  - Known at compile time
    - Static scheduling
    - Higher <u>inter-PE synchronization overhead</u> if inaccurate performance estimation, i.e., longer wait times and lower CPU utilization
  - Unknown until runtime (e.g., data-, event-dependent)
    - Dynamic scheduling
    - Higher <u>context-switching overhead</u> (running task blocked, new task scheduled)

Scheduling Example

Scheduling decision:

- Sequential ordering of behaviors on PE0, PE1
- Synchronization to maintain partial order across Pes
- Optimization - no control behaviors



System model after static scheduling

Scheduling Example (cont.)

- Atomic behaviors

```
B1( )
{
   …
   signal(B6_start);
}
```

```
B3( )
{
   wait(B3_start);
   ...
}
```

```
B7( )
{
   stmt;
   ...
}
```

```
B6( )
{
   int local;
   wait(B6_start);
   …
   shared = local + 1;
   signal(sync);
}
```

```
B4( )
{
   int local;
   wait(sync);
   local = shared - 1;
   …
   signal(B3_start);
}
```

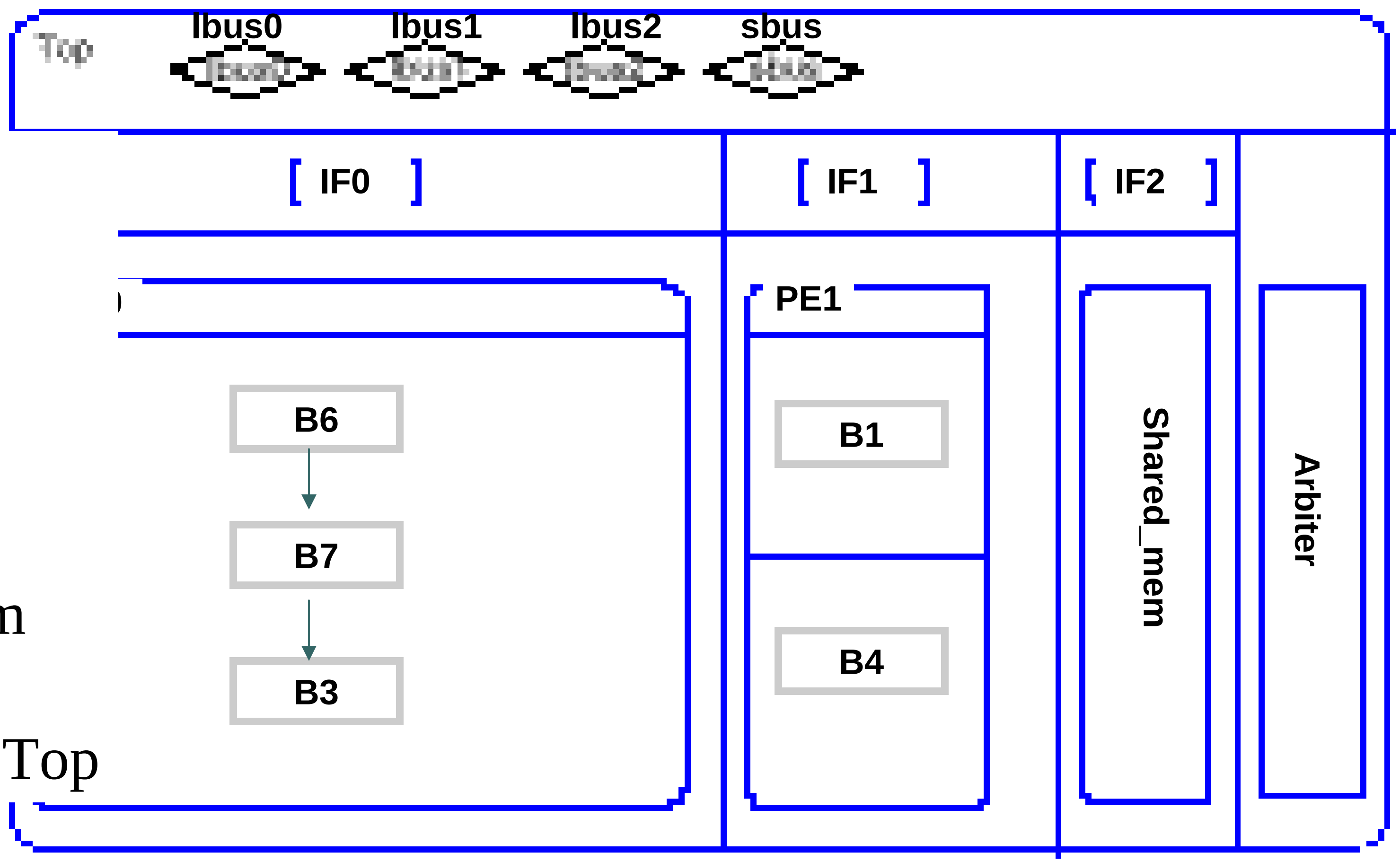
Communication Synthesis

- Implements the shared-variable accesses between concurrent behaviors using an inter-PE communication scheme
  - Shared memory: read or write to a shared-memory address
  - Local PE memory: send or receive message-passing calls
- Inserts interfaces to communication channels (local or system buses)

Communication example

Synthesis decision:

- Put all global variables into Shared_mem
- New global variables in Top



System model after communication synthesis

Communication Example (cont.)

- Atomic behaviors

```
B1( )
{
  …
  signal
    (*B6_start_addr);
}
```

```
B3( )
{
  wait(*B3_start_addr);
  ...
}
```

```
B7( )
{
  stmt;
  ...
}
```

```
B6( )
{
  int local;
  wait (*B6_start_addr);
  …
  *shared_addr = local + 1;
  signal(*sync_addr);
}
```

```
B4( )
{
  int local;
  wait (*sync_addr);
  local = *shared_addr - 1;
  …
  signal (*B3_start_addr);
}
```

Communication Example (cont.)

- Atomic behaviors

```
IF0( )
{
   stmt;
   ...
}
```

```
IF1( )
{
   stmt;
   ...
}
```

```
IF2( )
{
   stmt;
   ...
}
```

```
Arbiter( )
{
   stmt;
   ...
}
```

```
Shared_mem( )
{
   int shared;
   bool sync;
   bool B3_start;
   bool B6_start;
}
```

Analysis and Validation

- Functional validation of design models at each step using simulation or formal verification
- Analysis to estimate quality metrics and make design decisions
- Tools
  - Static analyzer - program, ASIC metrics
  - Simulator - functional, cycle-based, discrete-event
  - Debugger - access to state of behaviors
  - Profiler - dynamic execution information
  - Visualizer - graphical displays of state, data

Backend

- Implementations
  - Processor: compiler translates model into machine code
  - ASIC: high-level synthesis tool translates model into netlist of RTL components
  - Interface
    - Special type of ASIC that links a PE with other components
    - Implements the behavior of a communication channel

Summary

- Embedded systems are everywhere
- Key challenge: optimization of design metrics
  - Design metrics compete with one another
- A unified view of hardware and software is necessary to improve productivity
- Key technologies
  - Processor: general-purpose, application-specific, single-purpose
  - Design: compilation/synthesis, libraries/IP, test/verification