# Modern Methods and Tools for Signal Processing System Design


*Prof. Brian L. Evans*

**Embedded Signal Processing Laboratory**
**Dept. Electrical & Computer Engineering**
**The University of Texas at Austin**
**Austin, TX 78712-1084**

***http://signal.ece.utexas.edu/***

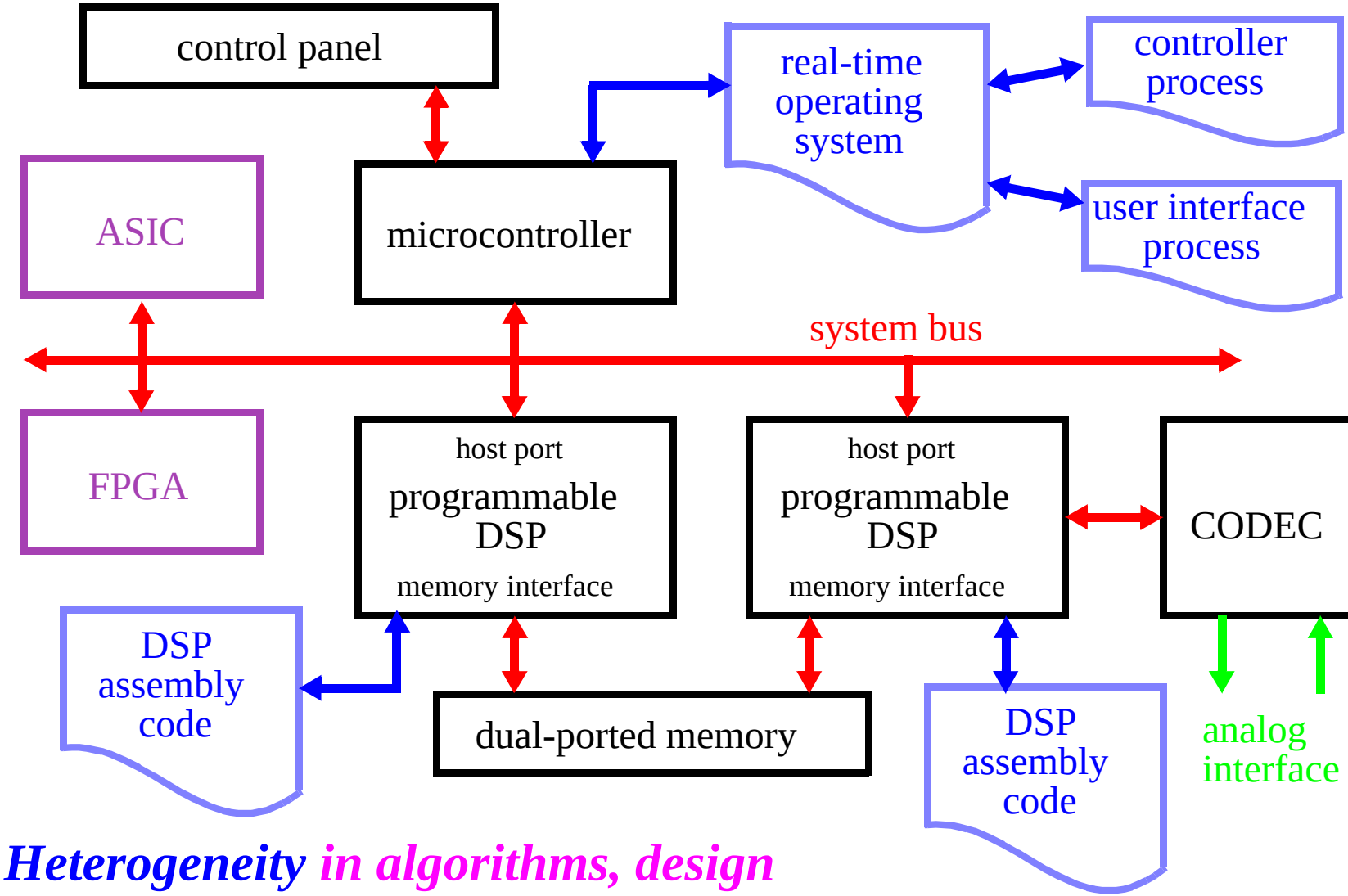
# Outline

- Introduction to system-level design
  - Heterogeneity in algorithms, design methods, implementation technologies
- Algorithm development environments
  - Quick prototyping of sequential algorithms
- Block diagrams
  - Formal modeling of computation and communication in algorithms
- Synthesis of software for digital signal processors
  - Optimal scheduling of multirate signal processing subsystems
- System-level design environments
  - Design assistance for mapping block diagrams into implementations
- Conclusion

# Introduction to System-Level Design

- New standard for speech/audio compression, image/video compression and mobile communications each year
- Heterogeneity in implementation technologies
  - Dedicated and configurable hardware
  - Dozens of high-volume programmable processors introduced each year (general-purpose processors, digital signal processors, microcontrollers)
- Heterogeneity in specification languages
  - Two dominant hardware description languages (VHDL and Verilog)
  - Revolution in high-level languages every 10 years (Fortran, C, C++, Java)
- Increasing complexity in implementation technologies
  - *Moore's Law:* Number of transistors on a chip doubles every 18 months
  - Networked, distributed, and multiprocessor systems

# Embedded Signal Processing Systems



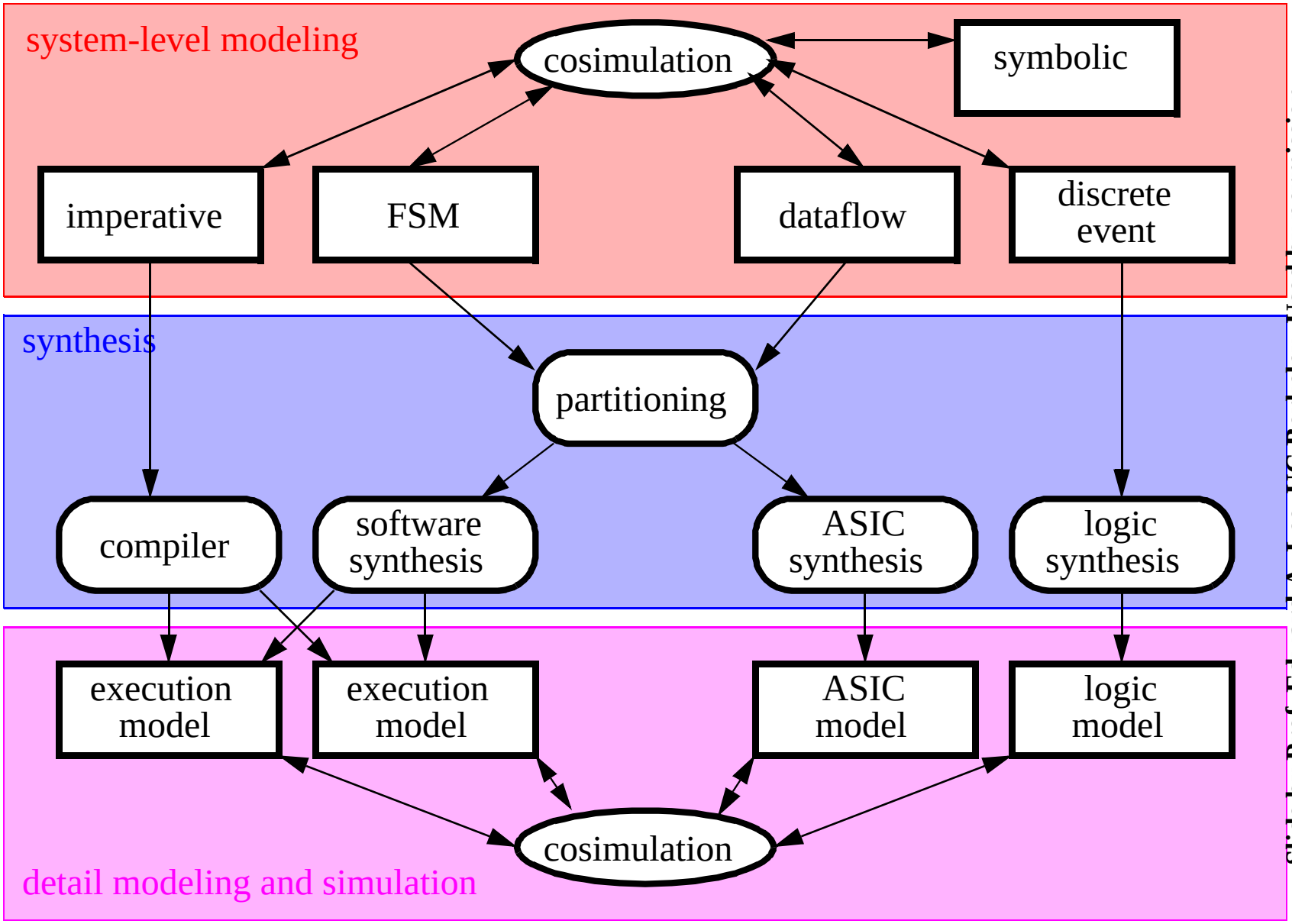

***Heterogeneity** in algorithms, design methods, and implementation technologies*

Slide by Prof. Edward A. Lee, UC Berkeley. Used by permission

# Handling Heterogeneity in System-Level Design

- **Raise the level of abstraction**
  - **More possibilities to map functionality and behavior to lower levels**
  - **Better reuse of existing designs**
- **Software subsystems**
  - ***Abstraction*: assembly, high-level languages, language-independent spec.**
  - ***Lessons*: object-oriented design, compilers lag behind processor architecture**
- **Hardware subsystems**
  - ***Abstraction*: transistor, cell library, hardware description languages**
  - ***Lessons*: timed and untimed simulation important, synthesis maturing**
- **System specification should be an abstraction of hardware and software specification to avoid implementation bias**

# Heterogeneity in a System-Level Design Flow




Slide by Prof. Edward A. Lee, UC Berkeley. Used by permission

# Algorithm Development Environments

- **Reduces development time**
  - **Abstracts compile-and-debug loop (portability across multiple platforms)**
  - **Visualization of results**
  - **Domain-specific extensions**
- **Imperative languages (Matlab, Interactive Data Language)**
  - **Vector computation exposes parallelism in a single computation**
  - **Imperative nature hides parallelism in overall computation**
  - **Imperative nature hides dependencies between functions (communication)**
- **Graphical languages (Labview, Khoros, Simulink)**
  - **Exposes parallelism and dependencies between functions exposed**
  - **Efficiency depends on models of computation and communication used**

# Algorithm Development Environments

| | *Matlab* | *Interactive Data Language* |
|---|---|---|
| **Commands** | interpreted (compiler available) | compiled |
| **Visualization** | signal/image/video plots | tailored for large data sets |
| **Applications** | signal and image processing, optimization, and controls | biomedical, remote sensing, seismic, and video imaging |
| **Visual interface** | Simulink and Stateflow | Envi |

| | *Labview* | *Khoros* |
|---|---|---|
| **Commands** | interpreted (compiler available) | interpreted |
| **Visualization** | limited | signal/image/video plots and large data sets |
| **Applications** | test/measurement, automation | image and video processing |
| **Block diagram** | dataflow language (G) | dataflow and control flow |

- **All support 8/16/32 bit integer data, single/double precision floating-point numbers, and data structures defined by user**
- **Image processing example: $\log_{10}(1 + |\text{FFT}(\text{image})|)$**

# Imperative Algorithm Development Environments

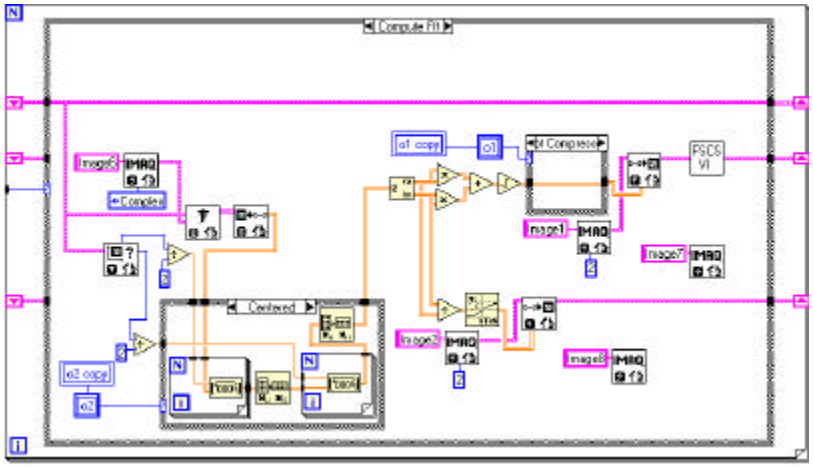
```
file_id = fopen('mandrill', 'r');
fsize = [512, 512];
[I1, count] =
    fread(file_id, fsize, 'unsigned char');
fclose(file_id);
I1=I1';
figure, image(I1);
axis off, axis square, colormap(gray(256))
map = 0:1/255:1;
map = [map', map', map'];
imwrite(I1, map, 'mandrill.tiff', 'tiff')
I2 = fft2(I1);
I2 = abs(I2);
I2 = log10(I2 + 1);
range = max(max(I2)) - min(min(I2));
I2 = (255/range) * (I2 - min(min(I2)));
I2 = fftshift(I2);
figure, image(I2);
axis off, axis square, colormap(gray(256))
imwrite(I2, map, 'mandrillFFT.tiff', 'tiff')
```

**Matlab Code**

```
file1 = 'mandrill'; lun = 1;
openr, lun, file1
pic = bytarr(512, 512)
readu, lun, pic
close, lun
picr = rotate(pic, 7)
tiff_write, 'mandrill', pic
window, 0, xsize=512, ysize=512, title='image'
tvscl, picr, 0, 0
picrf = fft(picr, -1)
picrfd = abs(picrf)
picrfd = alog10(picrfd + 1.0)
range = max(picrfd) - min(picrfd)
picrfd = ((255 / range) * (picrfd - min(picrfd)))
picrfd = shift(picrfd, 256, 256)
tiff_write, 'mandrillFFT.tiff', picrfd
window, 1, xsize=512, ysize=512, title='fft'
tv, picrfd, 0, 0
return
```

**IDL Code**

# Graphical Algorithm Development Environments



## Labview

## Khoros

## Web sites

**Mathworks**
**www.mathworks.com**

**IDL**
**www.rsinc.com**

**Labview**
**www.natinst.com**

**Khoros**
**www.khoral.com**

# Properties of Block Diagrams

- **Modular**
  - Large designs are composed of smaller designs
  - Modules encapsulate specialized expertise
- **Hierarchical**
  - Composite designs themselves become modules
  - Modules may be very complicated
- **Concurrent**
  - Modules logically operate simultaneously
  - Implementations may be sequential or parallel or distributed
- **Abstract**
  - Interaction of modules occurs within a “model of computation”
  - Many interesting and useful models of computation have emerged
- **Domain Specific**
  - Expertise encapsulated in models of computation and libraries of modules

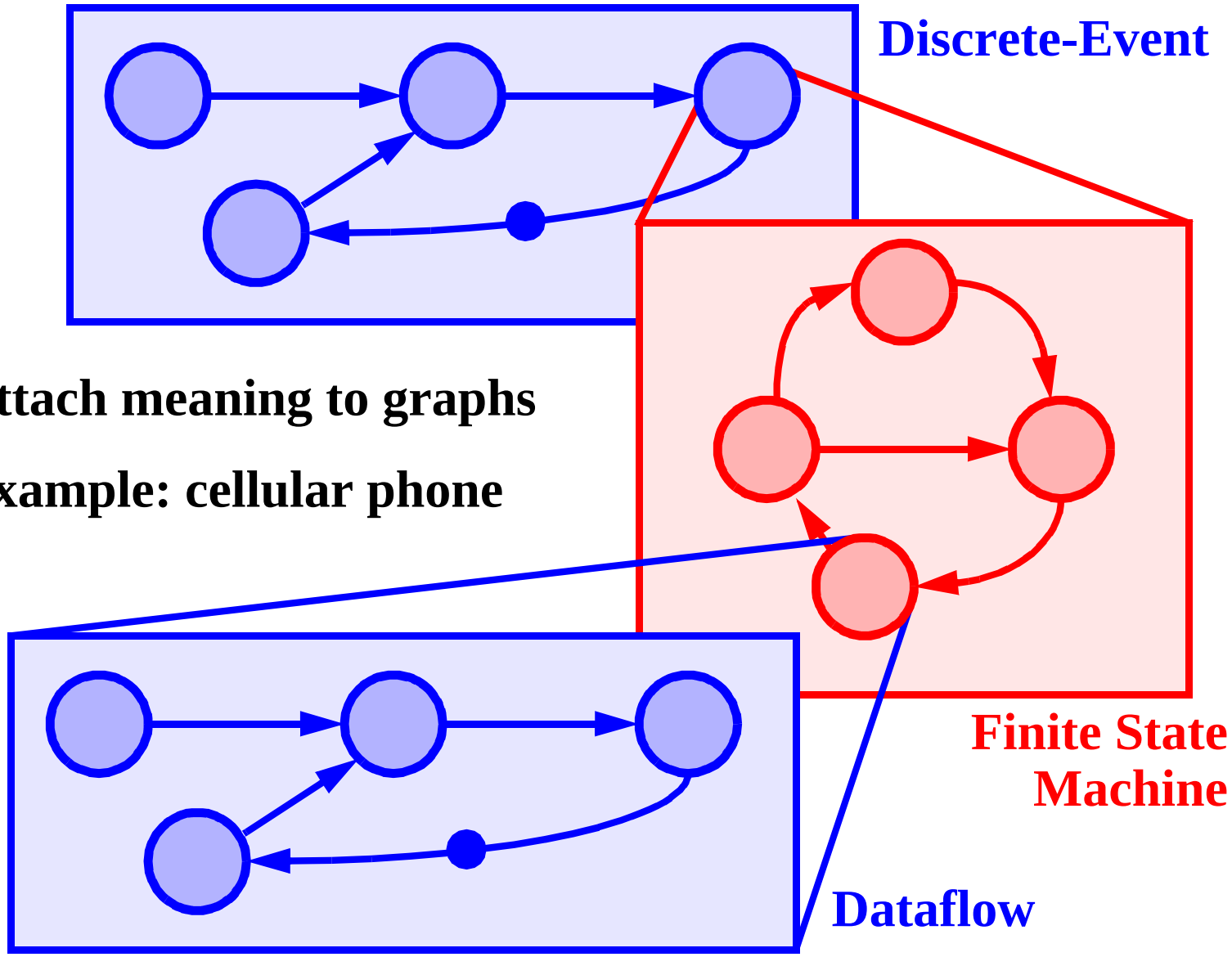
# Formal Modeling of Computation and Communication

- **Models of computation**
  - **Coordinate computation of and communication between subsystems**
  - **Ideally unbiased towards implementation in hardware and software**
  - **Map be mapped onto a variety of implementation technologies**

| *Subsystem* | *Model of Computation* |
|---|---|
| speech/audio processing | dataflow (1-D) |
| image processing | dataflow (1-D/2-D) |
| image/video resampling | dataflow (m-D multirate) |
| user interface | synchronous/reactive |
| communication protocols | finite state machine |
| digital control | dataflow |
| scalable descriptions | process networks |

- **Hierarchical combination forms heterogeneous systems**

# Specification Using Hierarchical Block Diagrams




- **Attach meaning to graphs**
- **Example: cellular phone**

# System Simulation and Synthesis

- **Two sides of the same coin**
  - ***Simulation*: scheduling then execution on desktop computer(s)**
  - ***Synthesis*: scheduling then code generation in C++, C, assembly, VHDL, etc.**
- **Models of computation enable**
  - **Global optimization of computation and communication**
  - **Scheduling and communication that is *correct by construction***

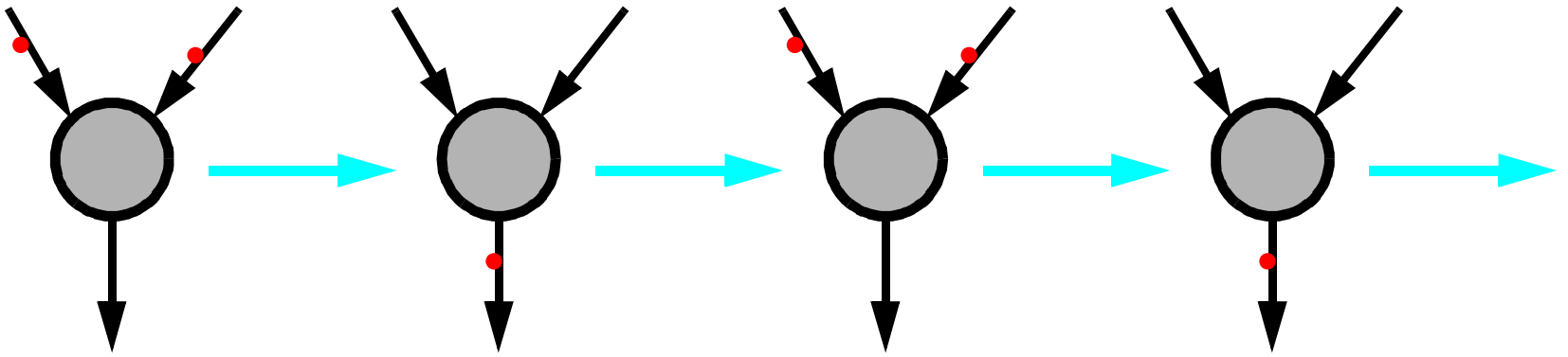
| *Model of Computation* | *Global State* | *Type of Comm.* | *Type of Scheduling* | *Optimal Scheduling* | *Simulation Speed* |
|---|---|---|---|---|---|
| synchronous dataflow | finite | asynch | static | $n^3$ | fast |
| Boolean dataflow | infinite | asynch | quasi-static | infinite | medium |
| process networks | infinite | asynch | dynamic | infinite | medium |
| finite state machine | finite | either | static | not poly. | fast |
| synchronous/reactive | finite | synch | static | $n^2$ | fast |
| discrete event | infinite | synch | dynamic | infinite | very slow |

# System Simulation and Synthesis

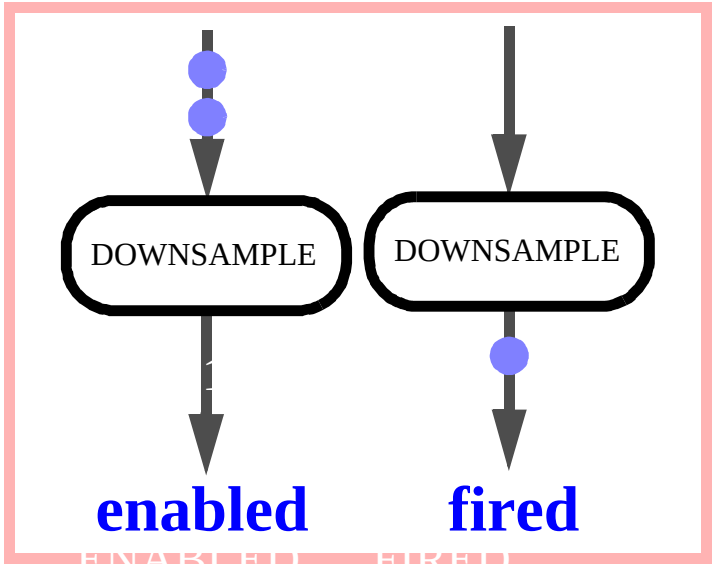
- **Design space (global state)**
  - ***Finite*, e.g. finite state machine, synchronous dataflow, synchronous/reactive**
  - ***Infinite*, e.g. imperative programming, Boolean dataflow, process networks**
- **Worst-case optimal scheduling time**
  - **Finite time if design space is finite, and infinite time if design space is infinite**
  - **Infinite-time off-line scheduling is impractical: use heuristics (e.g. compilers)**
  - **Process networks on-line scheduling takes infinite time but bounded memory**
- **Use models of computation with finite state when possible**
  - **Enables formal analysis (consistency, deadlock, boundedness, verification)**
  - **Suitable for fixed topologies (VLSI and embedded software implementations)**
- **Validation by simulation important throughout design flow**

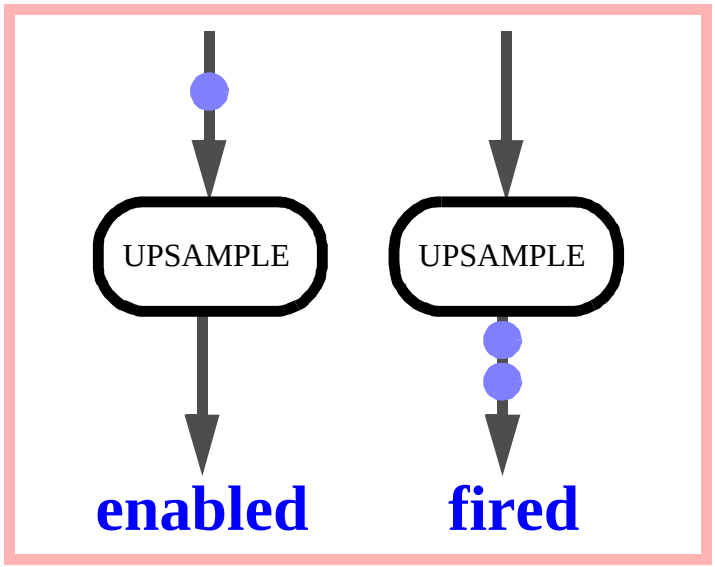
# Dataflow Models of Computation

- Matched to data-intensive processing, local communication
- A *signal* is a sequence of *tokens* (samples)
- An *actor* maps input tokens onto output tokens
- A set of *firing rules* specify when an actor can fire
- A firing *consumes* input tokens from and *produces* output tokens on *FIFO queues* with one writer and many readers



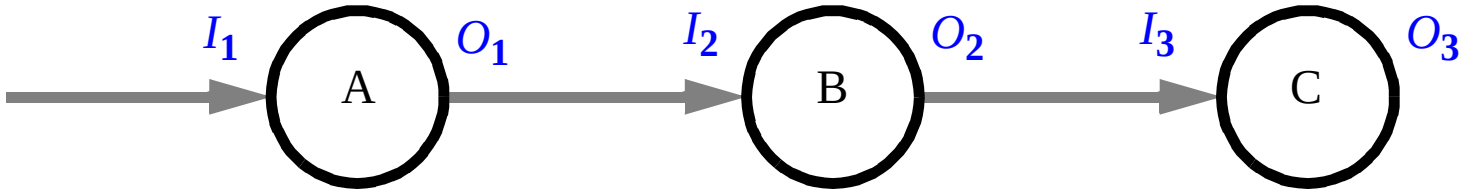
# Synchronous Dataflow




- **Firing rules** [Lee&Messerschmitt, 1986]
  - An actor is enabled when enough tokens are available at all of the inputs
  - When an actor executes, it produces and consumes same fixed amount of tokens
  - Flow of data through graph may not depend on values of data
- **Scheduling**
  - Control flow predictable at compile time
  - Performed once, repeatedly executed
- **Multirate signal processing**
  - Infinite streams of data
  - FIR filtering, FFTs, and resampling

# Scheduling Synchronous Dataflow Graphs

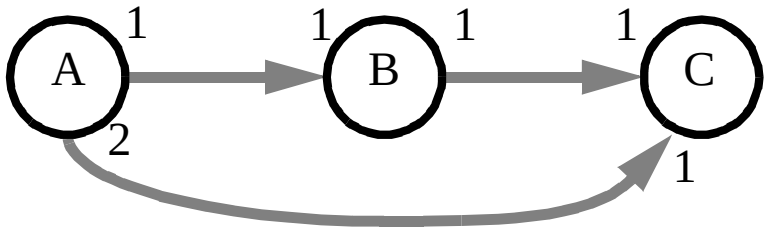



- **Load balance production and consumption of tokens by finding smallest integers $r_i$ in balance equations**

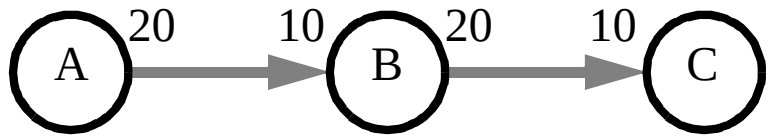
$$r_A O_1 = r_B I_2$$

$$r_B O_2 = r_C I_3$$

- **Schedule actor firings according to data dependencies until repetitions $r_i$ have been met for all actors**
- **Balance equations have no solution if graph is *inconsistent***



# Many Possible Synchronous Dataflow Schedules




**A produces 20 tokens.**
**B consumes 10 tokens and produces 20 tokens.**
**C consumes 10 tokens.**

- **Load balancing**
  - **Linear-time linear-space algorithm in size of synchronous dataflow graph**
  - **Periodic schedule: fire A 1 time, B 2 times, and C 4 times**
- **Scheduling data dependencies on a uniprocessor**
  - **List scheduling:**
    **quadratic-time algorithm**
  - **Looped scheduler:**
    **cubic-time algorithm**

| Scheduler | Schedule | Buffer Size |
|---|---|---|
| List | ABCBCCC | 50 |
| Looped | A (2 B(2 C)) | 40 |
| Looped | A (2 B)(4 C) | 60 |
| Looped | A (2 BC)(2 C) | 50 |

- **Second schedule has smallest code size and data size**

# Software Synthesis of Synchronous Dataflow Graphs

- **Features of conventional digital signal processors (DSPs)**
  - Limited, separate, on-chip data and program memory (often equal amount)
  - No-overhead downcounting looping (one pipeline flush to set up)
  - Function calls should be avoided when possible (high overhead)
- **Scheduling optimizations**
  - Minimize sum of program and data memory usage subject to constraints on actor repetitions and data dependencies
  - Non-polynomial optimization problem: polynomial-time heuristics are used
- **Synthesized uniprocessor code for the second schedule on the previous slide (minimum size)**

```
code block for A
for (i = 2; i > 0; i--) {
   code block for B
   for (j = 2; j > 0; j--) {
      code block for C
   }
}
```

# Software Synthesis from Synchronous Dataflow Graphs

- **Generate DSP assembly language (bypass compiler)**

| IIR Demonstration | Hand coded assembly | Generated assembly | Generated C code | Hand coded C |
|---|---|---|---|---|
| Program memory | 86 | 88 | 143 | 152 |
| X data memory | 20 | 20 | 24 | 21 |
| Y data memory | 30 | 33 | 31 | 42 |
| Run time (cycles) | 41,946 | 42,031 | 67,132 | 69,246 |

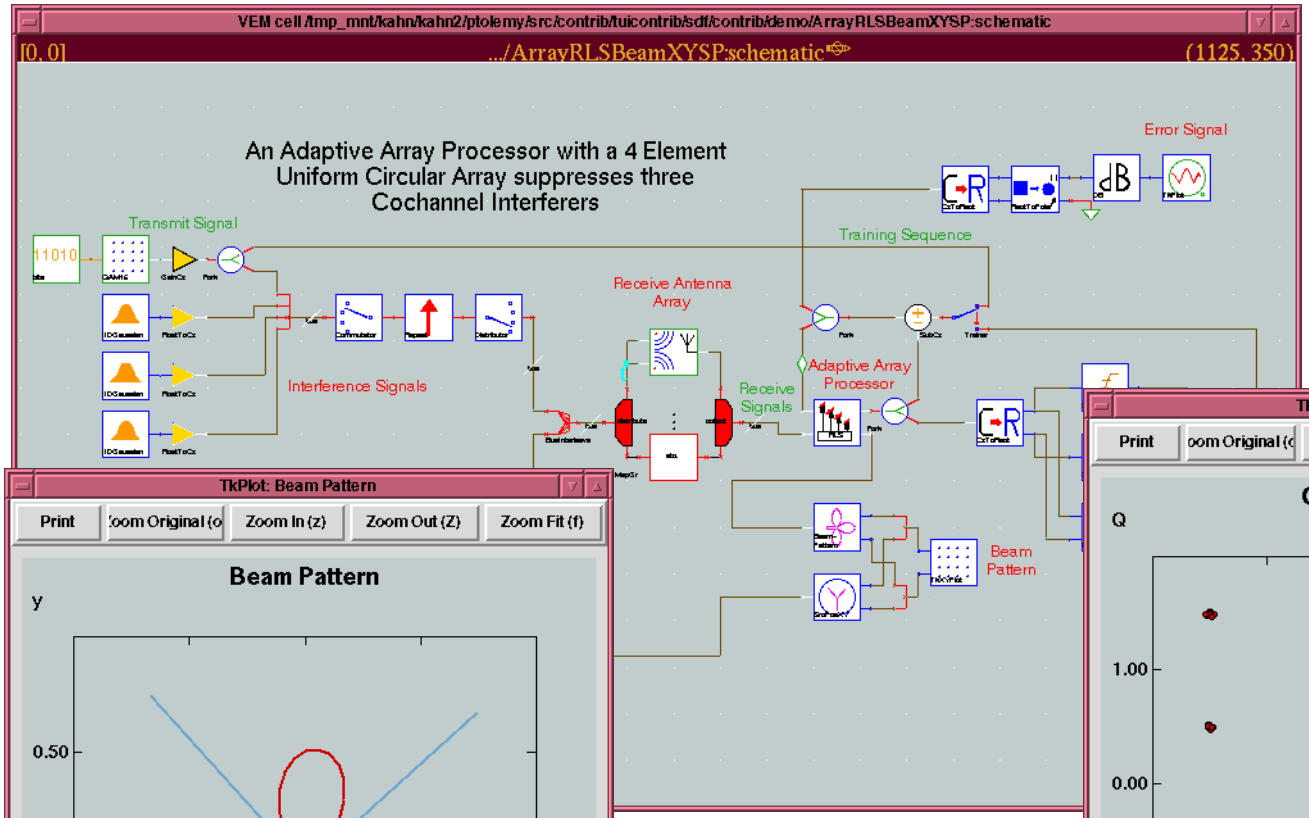
| CD to DAT converter | Generated assembly | Generated C code | Hand coded C (Julius Smith) |
|---|---|---|---|
| Program memory | 413 | 586 | 687 |
| X data memory | 456 | 468 | 398 |
| Y data memory | 283 | 283 | 324 |
| Run time (cycles) | 295,069 | 381,076 | 463,004 |

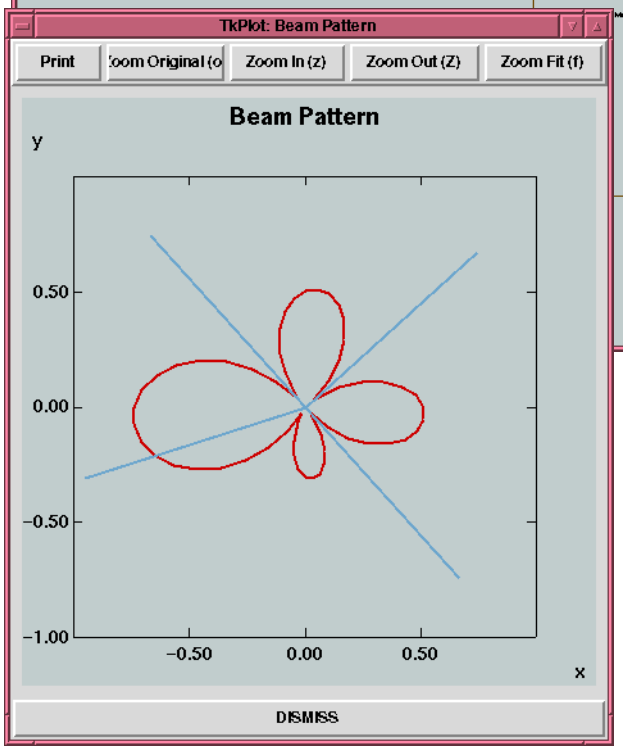
- **Benchmarks: Motorola 56000 DSP, KCCA56 C compiler, Generated code from SDF graphs in Ptolemy environment**

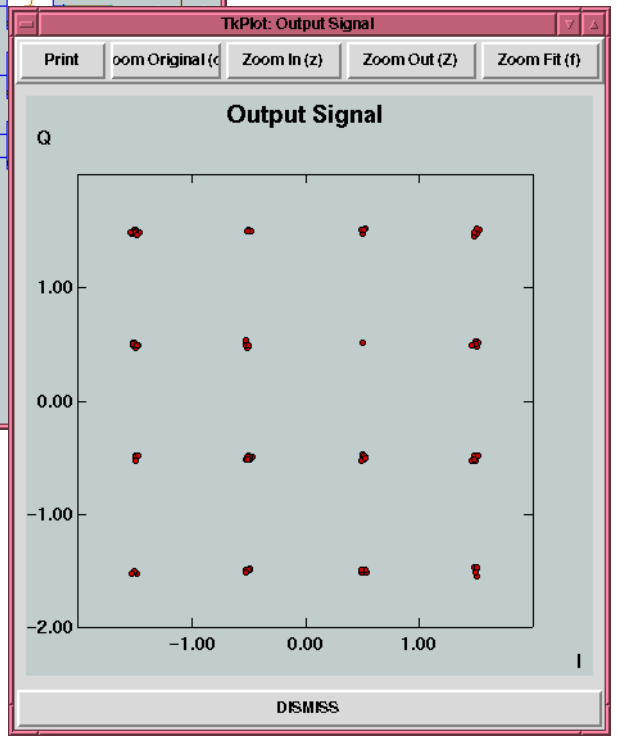
# System-Level Design Environments

- **Free (UC Berkeley)** http://www-cad.eecs.berkeley.edu
  - Ptolemy software environment for cosimulation and synthesis of dataflow graphs (biased toward dataflow modeling) http://ptolemy.eecs.berkeley.edu
  - Polis hardware/software codesign framework for embedded systems (biased toward finite state machines) http://www-cad.eecs.berkeley.edu/~polis
- **Commercial**
  - May cost $1 million for complete toolset (specification to manufacture)
  - All synthesize board-level designs and software for digital signal processors
  - Cadence (includes SPW dataflow modeling) http://www.cadence.com/
  - Mentor Graphics (includes Dataflow Language) http://www.mentor.com/
  - Synopsys (includes COSSAP dataflow modeling) http://www.synopsys.com/
  - HP EEsof (includes HP Ptolemy) http://www.tmo.hp.com/tmo/hpeesof

# Ptolemy Software Environment



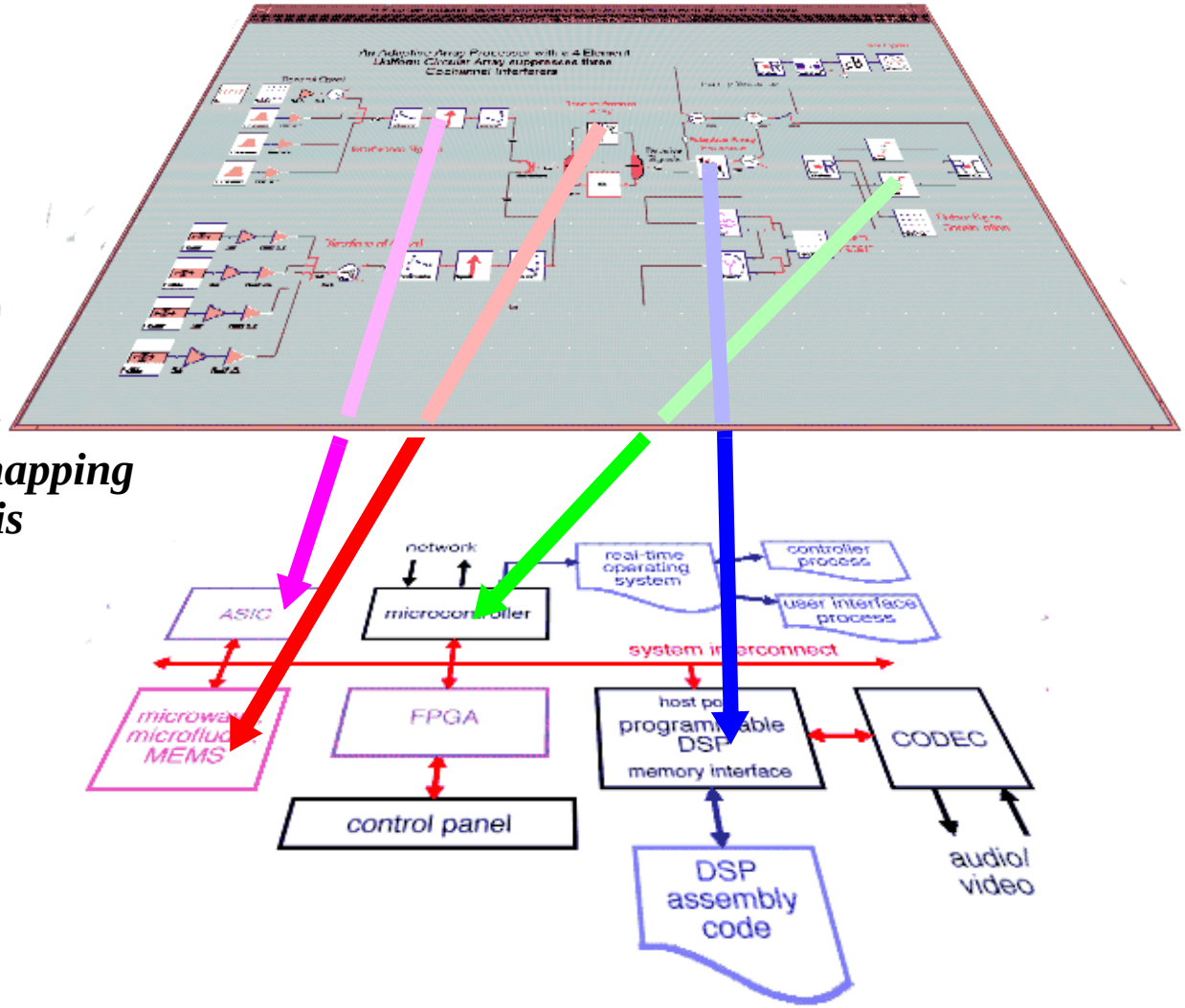




**Synchronous Dataflow Simulation**

**Author: Uwe Trautwein, Technical University of Ilmenau, Germany**

# Ptolemy Software Environment

**Executable specification (heterogeneous models of computation)**




***modeling, mapping and synthesis (design assistance)***

**Implementation (heterogeneous implementation technologies)**

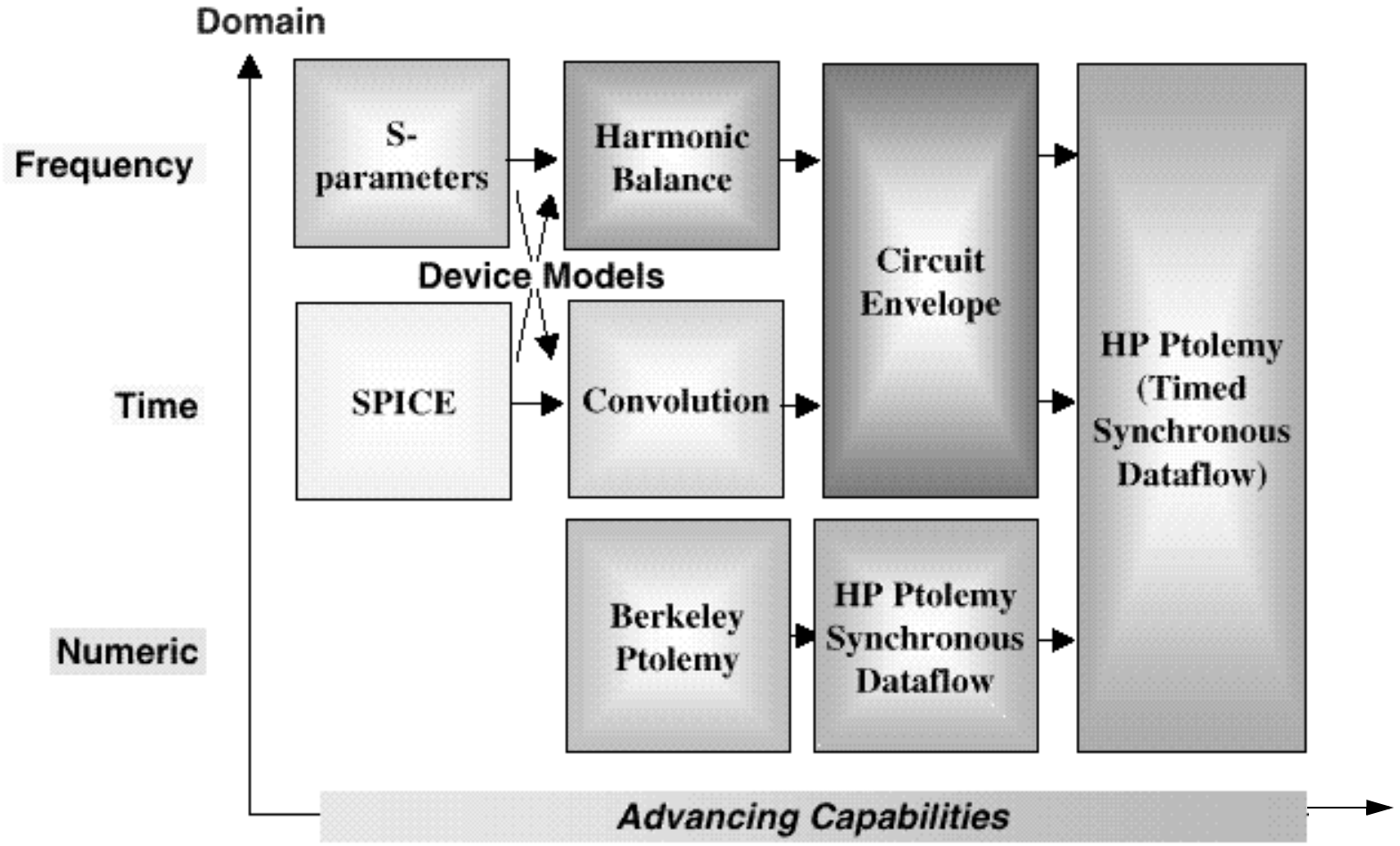
**Slide by Prof. Edward A. Lee, UC Berkeley. Used by permission**

# HP EEsof Advanced Design System

- **Specification, simulation, and synthesis of mixed analog, RF, and digital designs**
- **End-to-end wireless system simulation**
  - **Digital signal processing: dataflow modeling**
  - **Analog circuits: Spice**
  - **RF circuits: harmonic balance (frequency domain)**
  - **RF/analog signals: circuit envelope (time domain)**
  - **Antenna and propagation models**
  - **Applications: IS-95, GSM, wideband CDMA, digital broadcast TV**
- **End-to-end wireline communication system simulation**
  - **Mixed analog/digital designs (subset of wireless communication systems)**
  - **Analog channel models for modems, cable modems and DSL modems**
- **Synthesis**
  - **Board-level descriptions for Cadence and Mentor tools for fabrication**
  - **Board-level designs for base stations, handsets, and wireline modems**

# Advanced Design System

- **Mixed-domain simulation technologies**




- **Uses 3 generations of university (UC Berkeley) CAD tools**

Slide by Jose Luis Pino and Kal Kalbasi, HP EEsof. Used by permission

# Conclusion

- **Traditional signal processing development environment**
  - **One style of algorithm (e.g. speech/audio or image/video)**
  - **One implementation technology (e.g. Matlab, C, or digital signal processor)**
- **Decoupling system specification from its implementation**
  - **Implementation-unbiased models of computation**
  - **Compose models to specify complex heterogeneous systems**
  - **Simulate and synthesize heterogeneous systems**
- **Example system-level electronic design automation tools**
  - ***UC Berkeley Ptolemy*: dataflow, FSM, discrete event, synchronous/reactive models plus synthesis in C, assembly, and VHDL**
  - ***HP EEsof Advanced Design System*: mixed analog, RF, and digital design for communication systems using Spice, harmonic balance, dataflow modeling**

# Resources

- **Selected relevant papers**
  - J. Davis II, M. Goel, *et al.*, "Overview of the Ptolemy Project," ERL Tech. Report UCB/ERL No. M99/37, Dept. of EECS, UC Berkeley, July 1999. http://ptolemy.eecs.berkeley.edu/publications/papers/99/overview/
  - J. L. Pino & K. Kalbasi, "Cosimulating Dataflow with Analog RF Circuits," *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, Nov. 1998. http://www.ece.utexas.edu/~bevans/professional/asilomar98/pino.pdf
- **Related on-line course material**
  - B. L. Evans, "Embedded Software Systems," an intro to system-level design, graduate course, http://www.ece.utexas.edu/~bevans/courses/ee382c/
  - B. L. Evans, "Real-Time Digital Signal Processing Lab," an intro to digital signal processors, http://www.ece.utexas.edu/~bevans/courses/realtime/
  - E. A. Lee, "Specification and Modeling of Reactive Real-Time Systems," formal mathematics underlying models of computation, graduate course, http://www.eecs.berkeley.edu/~eal/ee290n/index.html
- **Other resources**
  - comp.dsp news group: FAQ http://www.bdti.com/faq/dsp_faq.hml
  - Embedded processors and systems: http://www.eg3.com