Embedded System Design
Modeling, Synthesis, Verification

Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner

Chapter 3: Modeling

7/8/2009

Modeling

- **Abstract view of a design**
  - Representation of reality in each design step
    - Apply analysis, synthesis and verification techniques
- **Core of automated design flow**
  - Varying levels of abstraction
    - Level & organization of detail determines speed & accuracy of results
  - Well-defined and unambiguous semantics
    - Objects, composition rules and transformations

- ➢ **System behavior**
  - ➢ Concurrent hierarchical processes
  - ➢ Variables and channels
- ➢ **System platform**
  - ➢ Processing, storage and communication elements (PEs and CEs)
  - ➢ Networks of busses

Outline

✓ **Introduction**
- **Models of Computation**
  - Programming models
  - Process-based models
  - State-based models
- **System Design**
- **Processor Modeling**
- **Communication Modeling**
- **System Models**
- **Summary and Conclusions**

Models of Computation (MoCs)

- **Conceptual, abstract description of system behavior**
  - Classification based on underlying characteristics
    - Computation and communication
  - Well-defined, formal definition and semantics
    - Functionality (data) and order (time)
    - ➤ Formal analysis and reasoning
    - ➤ Various degrees of complexity and expressiveness
  - Decomposition into pieces and their relationship
    - Objects and composition rules
    - Data and control flow

- ➤ **Analyzability and expressiveness of behavioral models**

Programming Models

- **Imperative programming models**
  - Ordered sequence of statements that manipulate program state
  - ➢ Sequential programming languages [C, C++, …]
- **Declarative programming models**
  - Dataflow based on explicit dependencies (causality)
  - ➢ Functional or logical programming languages [Haskell or Prolog]

- **Synchronous programming models**
  - Reactive vs. transformative: explicit ordering (time)
  - Lock-step operation of concurrent statement blocks
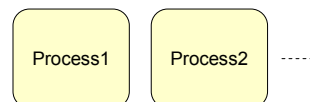 - ➢ Synchronous languages [Esterel (imperative), Lustre (declarative)]

Process-Based Models

➢ **Concurrency and causality (data flow)**

- **Set of processes**
  - Processes execute in parallel
    - Concurrent composition
  - Each process is internally sequential
    - Imperative program

Process1   Process2   ------

- **Inter-process communication**
  - Shared variables [Java]
    - Synchronization: critical section/mutex, monitor, …
  - Message passing [MPI]
    - Synchronous, rendezvous (blocking send)
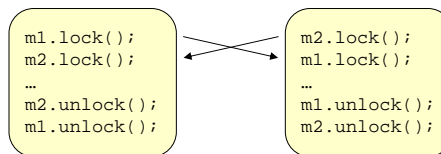   - Asynchronous, queues (non-blocking send)

Producer

Consumer
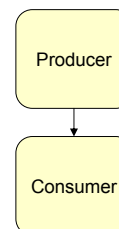
➢ **Implementation: OS processes or threads**
  - ➢ Single or multiple processors/cores

Deadlocks

- **Circular chain of 2 or more processes which each hold a shared resource that the next one is waiting for**
 - Circular dependency through shared resources

```
m1.lock();          m2.lock();
m2.lock();          m1.lock();
…                   …
m2.unlock();        m1.unlock();
m1.unlock();        m2.unlock();
```

  - ➤ Prevent chain by using the same precedence
  - ➤ Use timeouts (and retry), but: livelock

- ➤ **Dependency can be created when resources are shared**
  - ➤ Side effects, e.g. when blocking on filled queues/buffers

---

Determinism

- **Deterministic: same inputs always produce same results**
- **Random: probability of certain behavior**
- **Non-deterministic: undefined behavior (for some inputs)**
  - Undefined execution order
    - Statement evaluation in imperative languages: `f(a++, a++)`
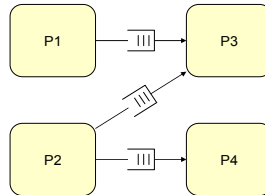   - Concurrent process race conditions:

```
x = a;          a = 1;
y = b;          b = 2;
```
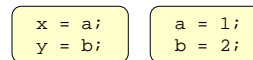
    x = ?, y = ?

- ➤ **Can be desired or undesired**
  - ➤ How to ensure correctness?
    - ➤ Simulator must typically pick one behavior
  - ➤ But: over-specification?
    - ➤ Leave freedom of implementation choice

Kahn Process Network (KPN) [Kahn74]

- **C-like processes communicating via FIFO channels**
  - Unbounded, uni-directional, point-to-point queues


- ➤ **Deterministic**
  - Behavior does not depend on scheduling strategy
  - Focus on causality, not order (implementation independent)
- ➤ **Difficult to implement [Parks95]**
  - ➤ Size of infinite FIFOs in limited physical memory?
  - ➤ Dynamic memory allocation, dependent on schedule
  - ➤ Boundedness vs. completeness vs. non-termination (deadlocks)

Dataflow

- **Breaking processes down into network of *actors***
  - Atomic blocks of computation, executed when *firing*
  - Fire when required number of input *tokens* are available
    - Consume required number of tokens on input(s)
    - Produce number of tokens on output(s)
  - ➤ Separate computation & communication/synchronization
    - ➤ Actors (indivisible units of computation) may fire simultaneously, any order
    - ➤ Tokens (units of communication) can carry arbitrary pieces of data
- **Unbounded FIFOs on arcs between actors**
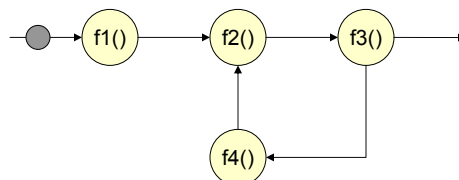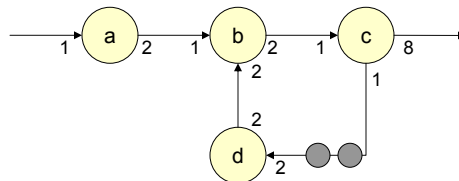

- ➤ **Signal-processing applications**

Synchronous Dataflow (SDF) [Lee86]

- **Fixed number of tokens per firing**
  - Consume fixed number of inputs
  - Produce fixed number of outputs

- ➢ **Can be scheduled statically**
  - ➢ Solve system of linear equations for relative rates
  - ➢ Periodically schedule actors in proportion to their rates
- ➢ **Find a sequence of firings in each period**
  - ➢ Trade-off code size and buffer sizes
    - – Single-appearance vs. memory-minimal schedule

Process Calculi

- **Rendezvous-style, synchronous communication**
  - Communicating Sequential Processes (CSP) [Hoare78]
  - Calculus of Communicating Systems (CCS) [Milner80]
  - ➢ Restricted interactions

- ➢ **Formal, mathematical framework: process algebra**
  - Algebra = <objects, operations, axioms>
    - – Objects: processes {$P, Q, \dots$}, channels {$a, b, \dots$}
    - – Composition operators: parallel ($P \| Q$), prefix/sequential ($a \rightarrow P$), choice ($P + Q$)
    - – Axioms: indemnity ($\emptyset \| P = P$), commutativity ($P + Q = Q + P$, $P \| Q = Q \| P$)
  - ➢ Manipulate processes by manipulating expressions
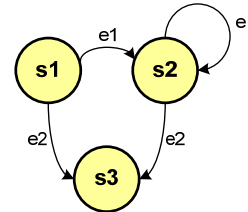
- ➢ **Parallel programming languages**
  - ➢ CSP-based [Occam/Transputer, Handle-C]

State-Based Models

> **Sequence and reactivity (control flow)**

- **Explicit enumeration of computational states**
  - State represents captured history
- **Explicit flow of control**
  - Transitions in reaction to events

> **Stepwise operation of a machine**
  > Cycle-by-cycle hardware behavior
  > Finite number of states
> **Formal analysis**
  > Reachability, equivalence, …

Finite State Machines

- **Finite State Machine (FSM)**
  - Basic model for describing control and automata
    - Sequential circuits
  - States *S,* inputs/outputs *I/O,* and state transitions
    - FSM: <*S, I, O, f, h*>
    - Next state function *f*: $S \times I \rightarrow S$
  - Output function *h*
    - Mealy-type (input-based), *h*: $S \times I \rightarrow O$
    - Moore-type (state-based), *h*: $S \rightarrow O$
- **Finite State Machine with Data (FSMD)**
  - Computation as control and expressions
    - Controller and datapath of RTL processors
  - FSM plus variables *V*
    - FSMD: <*S, I, O, V, f, h*>
    - Next state function *f*: $S \times V \times I \rightarrow S \times V$
    - Output function *h*: $S \times V \times I \rightarrow O$
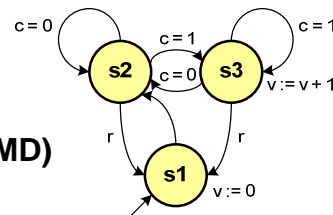
Hierarchical & Concurrent State Machines

- **Superstate FSM with Data (SFSMD)**
  - Hierarchy to organize and reduce complexity
    - Superstates that contain complete state machines each
    - Enter into one and exit from any substate
- **Hierarchical Concurrent FSM (HCFSM)**
  - Hierarchical (OR) and parallel (AND) state composition
  - Communication through variables, signals and events
  - ➤ Graphical notation [StateCharts]
    - ➤ Lock-step concurrent execution
    - ➤ Event propagation in case of combinatorial cycles? [StateMate]
  - ➤ Fully synchronous interpretation
    - ➤ Instantaneous and zero-delay events [Argos]
    - ➤ Fixed-point [SyncCharts/Esterel]
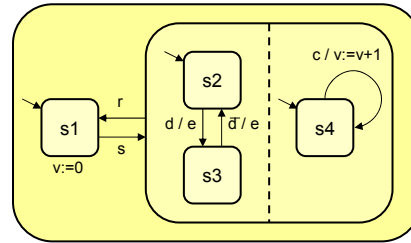
Process State Machines

- **Sound combination of process and state based models**
  - Asynchronous concurrent HCFSM execution [UML StateDiagram]
    - Explicit event queues, deadlock analysis [PetriNet]
  - Globally asynchronous, locally synchronous (GALS) composition
    - Co-design Finite State Machines (CFSM) [Polis]
  - Leaf states are imperative processes
    - Program State Machine (PSM) [SpecSyn]
  - ➤ Processes and abstract channels
    - Computation & communication
    - ➤ Process State Machine (PSM) [SpecC]


```
…
c1.receive(d,e);
a = 42;
while (a<100)
{ b = b + a;
  if (b > 50)
    c = c + d;
  else
    c = c + e;
  a = c;
}
c2.send(a);
…
```
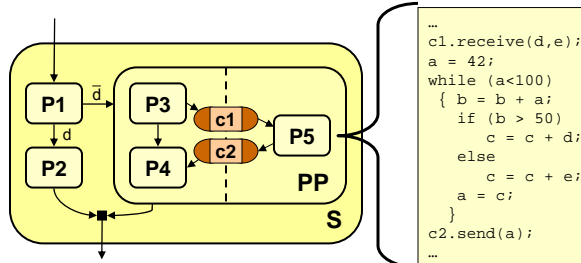
Outline

- ✓ **Introduction**
- ✓ **Models of Computation**
- • **System Design**
  - • System design languages
  - • System modeling
- • **Processor Modeling**
- • **Communication Modeling**
- • **System Models**
- • **Summary and Conclusions**

Languages

➤ **Represent a model in machine-readable form**
  ➤ Apply algorithms and tools

- • **Syntax defines grammar**
  - • Possible strings over an alphabet
  - • Textual or graphical
- • **Semantics defines meaning**
  - • Mapping of strings to an abstract state machine model
    - – Operational semantics
  - • Mapping of strings into a mathematical domain (e.g. functions)
    - – Denotational semantics

➤ **Semantic model vs. MoC vs. design model instance**
  ➤ Basic semantic models can represent many MoCs (e.g. FSMs in C)
  ➤ MoCs can be represented in different languages

Design Languages

- **Netlists**
  - Structure only: components and connectivity
  - Gate-level [EDIF], system-level [SPIRIT/XML]

- **Hardware description languages (HDLs)**
  - Event-driven behavior: signals/wires, clocks
  - Register-transfer level (RTL): boolean logic
  - Discrete event [VHDL, Verilog]

- **System-level design languages (SLDLs)**
  - Software behavior: sequential functionality/programs
 - C-based, event-driven [SpecC, SystemC, SystemVerilog]

System Modeling

- **Basis of any design flow and design automation**
  - Inputs and outputs of design steps
    - Capability to capture complex systems
    - Precise, complete and unambiguous
  - Models at varying levels of abstraction
    - Level and granularity of implementation detail
    - Speed vs. accuracy

- **Design models as an abstraction of a design instance**
  - Representation of some aspect of reality
    - Virtual prototyping for analysis and validation
  - Specification for further implementation/synthesis
    - Describe desired functionality
  - Documentation & specification
    - Abstraction to hide details that are not relevant or not yet known
    - Different parts of the model or different use cases for the same model

Design Process

- **From specification to implementation**
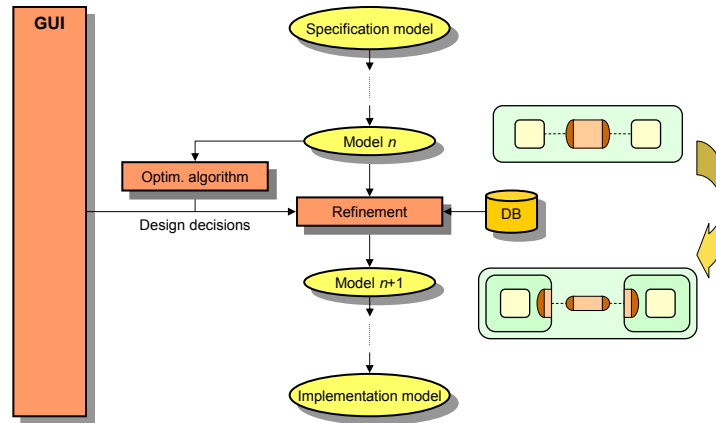
- ➢ **Successive, stepwise model refinement**
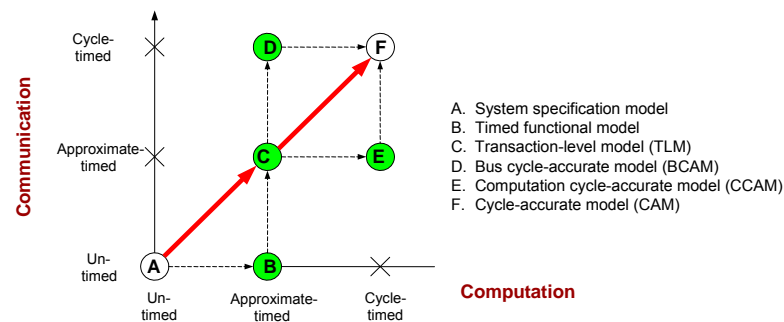- ➢ **Layers of implementation detail**

Abstraction Levels

- **Abstraction based on level of detail & granularity**
  - Computation and communication
- ➢ **System design flow**
  - ➢ Path from model A to model F

A. System specification model
B. Timed functional model
C. Transaction-level model (TLM)
D. Bus cycle-accurate model (BCAM)
E. Computation cycle-accurate model (CCAM)
F. Cycle-accurate model (CAM)

- ➢ **Design methodology and modeling flow**
  - ➢ Set of models and transformations between models

Outline

Processors

- **Basic system component is a *processor***
  - Programmable, general-purpose software processor (CPU)
  - Programmable special-purpose processor (e.g. DSPs)
  - Application-specific instruction set processor (ASIP)
  - Custom hardware processor
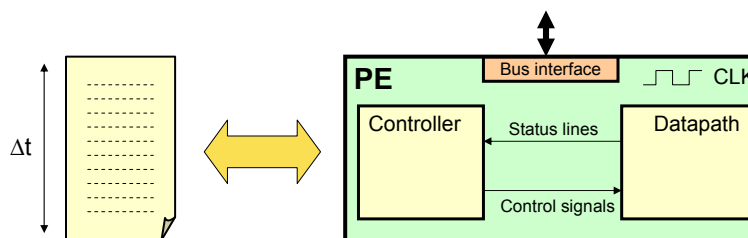
➢ **Functionality *and* timing**

Processor Modeling
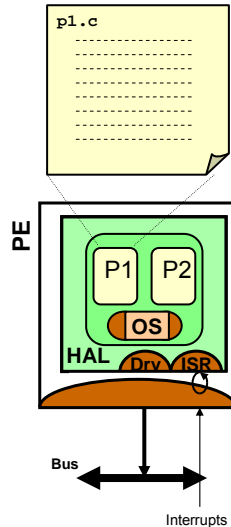
- **Application modeling**
  - Native process execution (C code)
  - Back-annotated execution timing

- **Processor modeling**
  - Operating system (OS)
    - Real-time multi-tasking (RTOS)
    - Bus drivers (C code)
  - Hardware abstraction layer (HAL)
    - Interrupt handlers
    - Media accesses
  - Processor hardware
    - Bus interfaces (I/O state machines)
    - Interrupt suspension and timing

Application Layer

- **High-level, abstract programming model**
  - Hierarchical process graph
    - ANSI C leaf processes
    - Parallel-serial composition
  - Abstract, typed inter-process communication
    - Channels
    - Shared variables
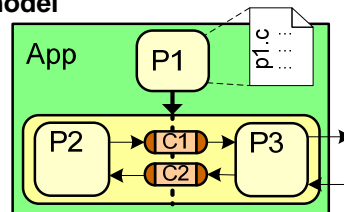
- ➤ **Timed simulation of application functionality (SLDL)**
  - Back-annotate timing
    - Estimation or measurement (trace, ISS)
    - Function or basic block level granularity
  - Execute natively on simulation host
    - Discrete event simulator
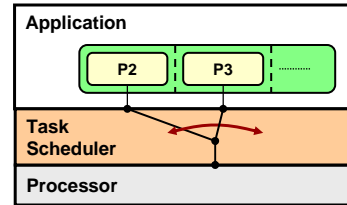   - Fast, native compiled simulation

```
process p1()
{
  ...
  waitfor(5);
  ...
}
```

Logical time
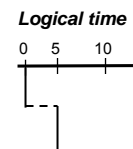
Operating System Layer

- **Scheduling**
  - Group processes into tasks
    - Static scheduling
  - Schedule tasks
    - Dynamic scheduling, multitasking
    - Preemption, interrupt handling
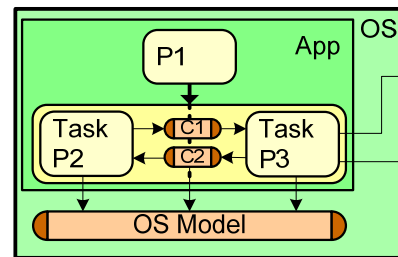   - Task communication (IPC)

- ➤ **Scheduling refinement**
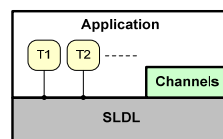 - Flatten hierarchy
  - Reorder behaviors
- ➤ **OS refinement**
  - Insert OS model
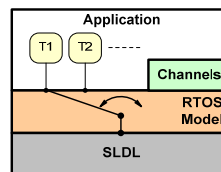 - Task refinement
  - IPC refinement
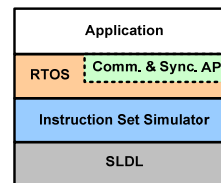

OS Modeling

- **High-level RTOS abstraction**

| Specification	TLM	Implementation

- Specification is fast but inaccurate
  - Native execution, concurrency model
- Traditional ISS-based validation infeasible
  - Accurate but slow (esp. in multi-processor context), requires full binary

- ➤ Model of operating system
  - ➤ High accuracy but small overhead at early stages
  - ➤ Focus on key effects, abstract unnecessary implementation details
 - ➤ Model all concepts: Multi-tasking, scheduling, preemption, interrupts, IPC

Simulated Dynamic Behavior

Application

Logical time

OS

Inaccuracy due to timing granularity
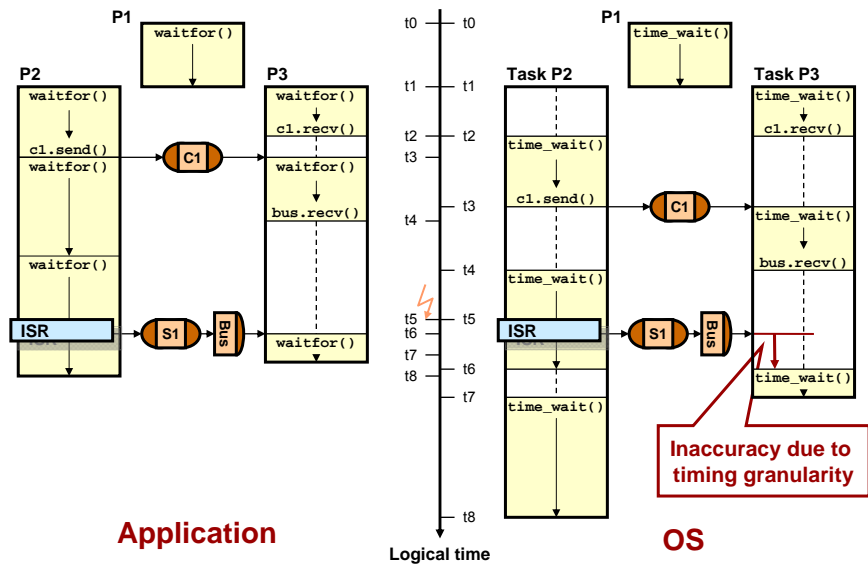
Operating System Layer
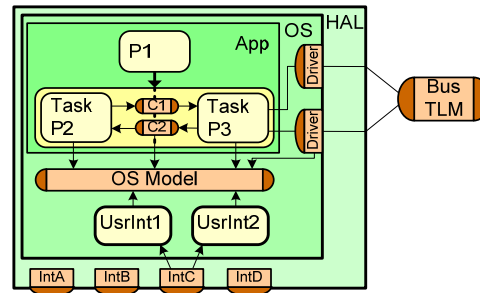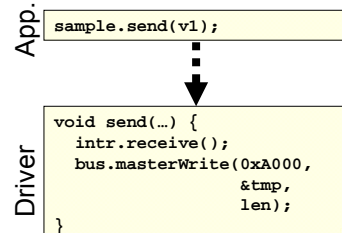
> **OS model**
- On top of standard SLDL
- Wrap around SLDL primitives, replace event handling
  - Block all but active task
  - Select and dispatch tasks
- Target-independent, canonical API
  - Task management
  - Channel communication
  - Timing and all events
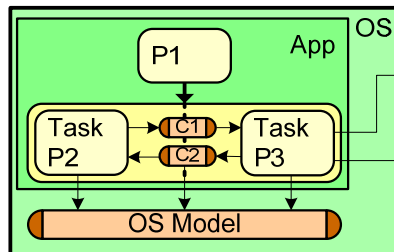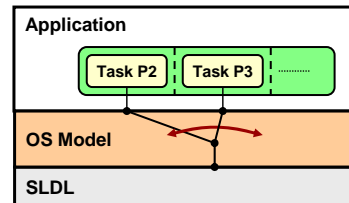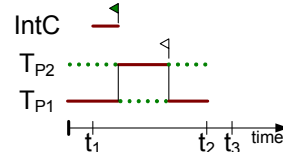
Hardware Abstraction Layer

- **External communication**
  - Software Drivers
    - Presentation, session, network communication layers
    - Synchronization (interrupts)
  - Hardware/software boundary
    - Low-level HW access
    - Bus drivers and interrupt handlers
    - Canonical HW/SW interface
  - External interface
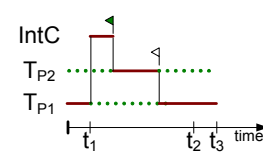   - Bus transactions (TLM)
    - Interrupt trigger

App.
```
sample.send(v1);
```

Driver
```
void send(…) {
    intr.receive();
    bus.masterWrite(0xA000,
                    &tmp,
                    len);
}
```
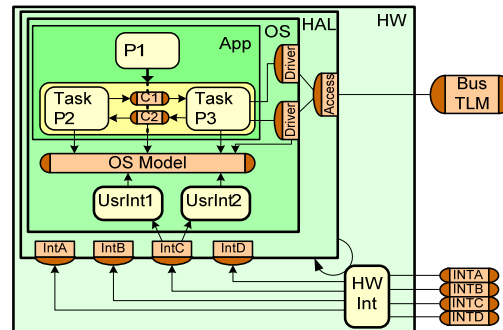

Hardware Layer (1)

HAL:

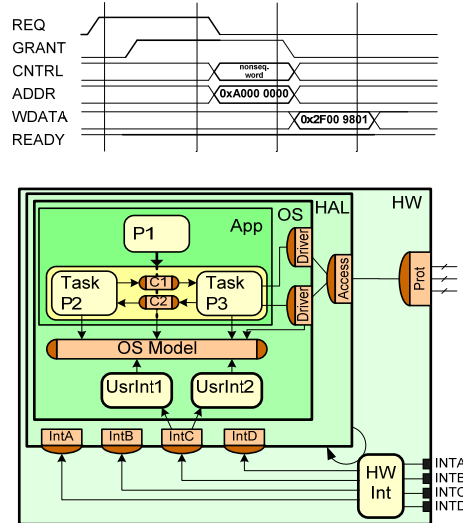Hardware:

- **Processor TLM**
  - HW interrupt handling
    - Interrupt logic
      » Suspend user code
    - Interrupt scheduling
      » Priority, nesting
  - Peripherals
    - Interrupt controller
    - Timers
  - TLM bus model
    - Bus transactions

Hardware Layer (2)

- **Bus-functional model (BFM)**
  - Pin-accurate processor model
    - Timing-accurate bus and interrupt protocols
  - Bus model
    - Pin- and cycle-accurate
    - Driving and sampling of bus wires

REQ
GRANT
CNTRL
ADDR
WDATA
READY

nonseq. word
0xA000 0000
0x2F00 9801

Processor Model

- **Processor layers**
  - Application
    - Native C
    - Back-annotated timing
  - Operating system
    - OS model
  - Hardware abstraction
    - Middleware, Firmware
  - Processor hardware
    - Bus I/F
    - Interrupts, suspension
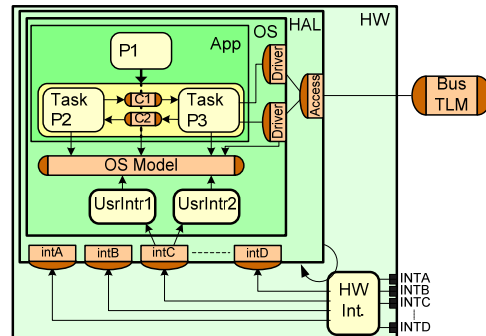
Features						
Target approx. computation timing	Appl.					
Task mapping, dynamic scheduling		OS				
Task communication, synchronization						
Interrupt handlers, low level SW drivers			HAL			
HW interrupt handling, int. scheduling				HW-TLM		
Cycle accurate communication					HW-BFM	
Cycle accurate computation						BFM - ISS

Outline

Busses

- **For each transaction between two communication partners**
  - 1 sender, 1 receiver
  - 1 master (initiator), 1 slave (listener)

PE1   PE2

Sender ————————t_n————————→ *Receiver*

Receiver ←————————t_m———————— *Sender*

Master and/or slave?   *Master and/or slave?*
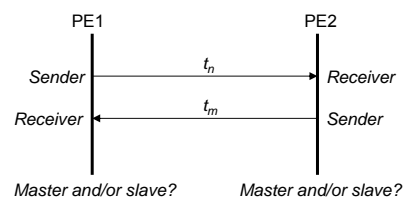
➢ **Any combination of master/slave, sender/receiver**
  ➢ Master/Slave bus
    ➢ Statically fixed master/slave assignments for each PE pair
    ➢ PEs can be masters, slaves or both (dual-port)
  ➢ Node-based bus (e.g. Ethernet, CAN):
    ➢ Sender is master, receiver is slave

➢ **Reliable (loss-less, error-free)?**

Communication Modeling

- **ISO/OSI 7-layer model**

Layer	Semantics	Functionality	Implementation	OSI
Application	Channels, variables	Computation	Application	7
Presentation	End-to-end typed messages	Data formatting	OS	6
Session	End-to-end untyped messages	Synchronization, Multiplexing	OS	5
Transport	End-to-end data streams	Packeting, Flow control	OS	4
Network	End-to-end packets	Subnet bridging, Routing	OS	3
Link	Point-to-point logical links	Station typing, Synchronization	Driver	2b
Stream	Point-to-point control/data streams	Multiplexing, Addressing	Driver	2b
Media Access	Shared medium byte streams	Data slicing, Arbitration	HAL	2a
Protocol	Media (word/frame) transactions	Protocol timing	Hardware	2a
Physical	Pins, wires	Driving, sampling	Interconnect	1

➢ **A *model*, not an implementation !**

Communication Primitives

- **Events, transitions**
  - Pure control flow, no data
- **Shared variables**
  - No control flow, no synchronization
- **Synchronous message passing**
  - No buffering, two-way control flow
- **Asynchronous message passing**
  - Only control flow from sender to receiver guaranteed
  - May or may not use buffers (implementation dependent)
- **Queues**
  - Fixed, defined queue length (buffering)
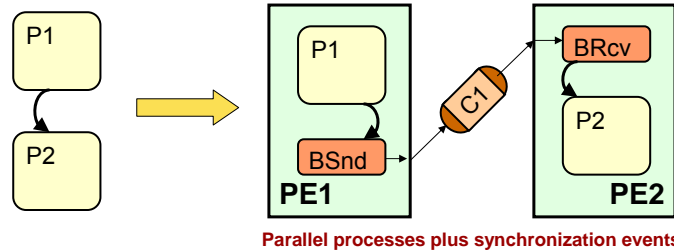- **Complex channels**
  - Semaphores, mutexes

➢ **Reliable communication primitives (lossless, error-free)**

Application Layer (1)

- **Synchronization**
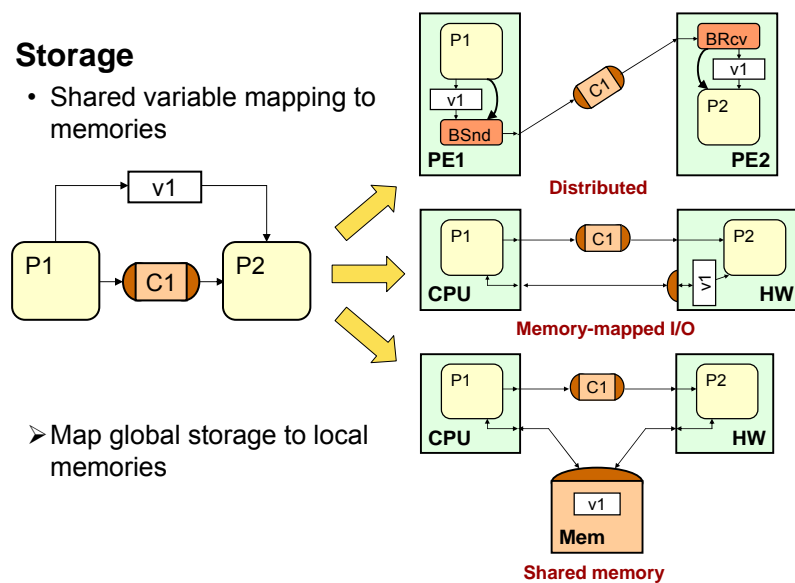 - Synthesize control flow

Parallel processes plus synchronization events

  ➢ Implement sequential transitions across parallel components

---

Application Layer (2)
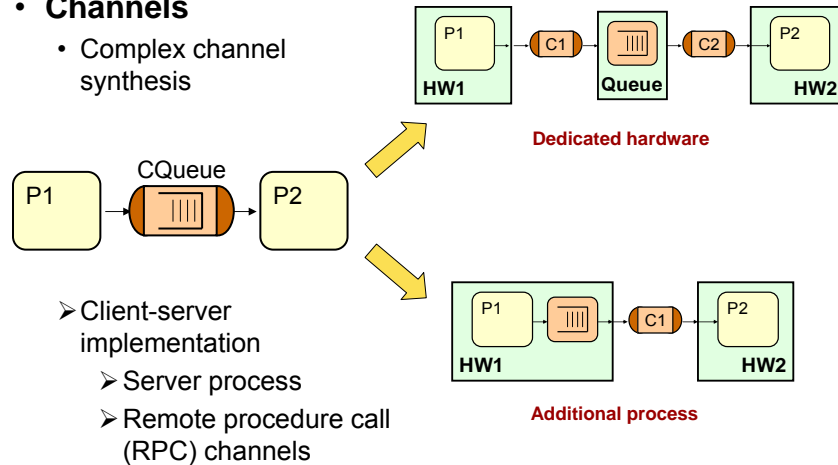
- **Storage**
  - Shared variable mapping to memories

Distributed

Memory-mapped I/O

  ➢ Map global storage to local memories

Shared memory

Application Layer (3)

- **Channels**
  - Complex channel synthesis

- Client-server implementation
  - Server process
  - Remote procedure call (RPC) channels

Dedicated hardware

Additional process

Presentation Layer

- **Data formatting**
  - Translate abstract data types into canonical network byte layout
    1. Global network data layout
    2. Shared, optimized layout for each pair of communicating PEs
  - Convert typed messages into untyped, ordered byte streams
 - Convert variables into memory byte layout
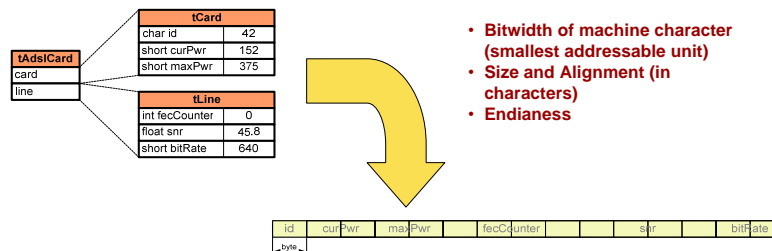
- **Bitwidth of machine character (smallest addressable unit)**
- **Size and Alignment (in characters)**
- **Endianess**

Session Layer

- **Channel merging**
    - Merge application channels into a set of untyped end-to-end message streams
        1. Unconditionally merge sequential channels
        2. Merge concurrent channels with additional session ID (message header)
    - ➤ Channel selection over end-to-end transports
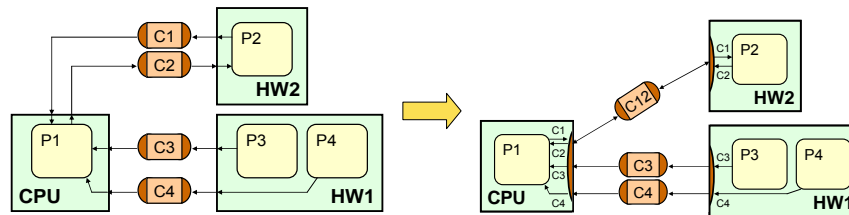
Network Layer

- **Split network into subnets**
    - Routing of end-to-end paths over point-to-point links
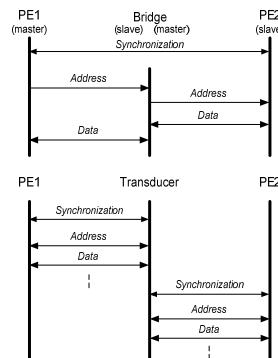   - Insert communication elements (CEs) to connect busses
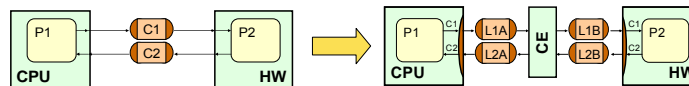
    - Bridges
        - Transparently connect slave & master sides at protocol level
        - Bridges maintain synchronicity, no buffering
    - Transducers
        - Store-and-forwarding of data packets between incompatible busses
        - Intermediate buffering, results in asynchronous communication

Transport Layer

- **Packeting and routing**
  - ➢ Packetization to reduce buffer sizes
    1. Fixed packet sizes (plus padding)
    2. Variable packet size (plus length header)

  - ➢ Protocol exchanges (ack) to restore synchronicity
    - ➢ Iff synchronous message passing and transducer in the path

  - ➢ Packet switching and identification (logical routing)
    1. Dedicated logical links (defer identification to lower layers)
    2. Network endpoint addressing (plus packet address headers)

  - ➢ Physical routing in case of multiple paths between PEs
    1. Static, predetermined routing (based on connectivity/headers)
    2. Dynamic (runtime) routing

Link Layer (1)

- **Synchronization (1)**
  - Ensure slave is ready before master initiates transaction
    1. Always ready slaves (memories and memory-mapped I/O)
    2. Defer to fully synchronized bus protocol (e.g. RS232)
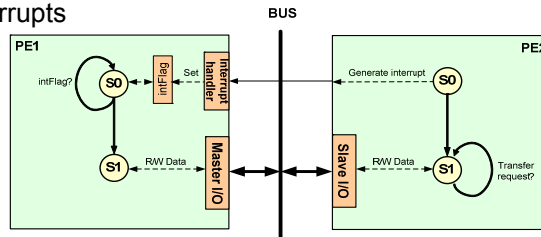   3. Separate synchronization mechanism
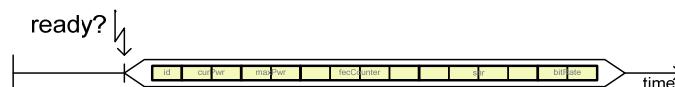
  - ➢ Events from slave to master for master/slave busses
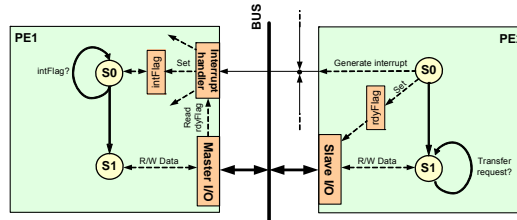 - ➢ Synchronization packets for node-based busses

Link Layer (2)

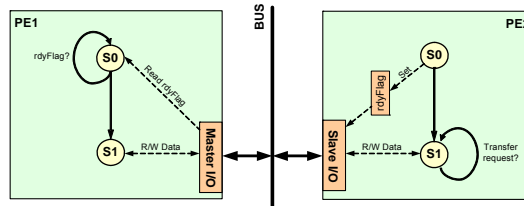- **Synchronization (2)**
  - ➢ Dedicated interrupts
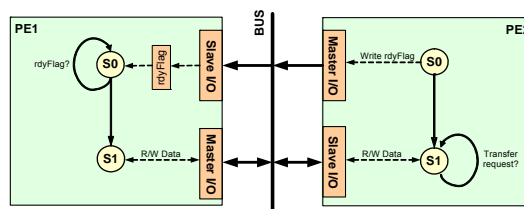

  - ➢ Shared interrupts

Link Layer (3)

- **Synchronization (3)**
  - ➢ Slave polling

  - ➢ Flag in master

Stream Layer

- **Addressing**
  - Multiplexing of links over shared medium
  - Separation in space through addressing
  - ➤ Assign physical bus addresses to links
    1. Dedicated physical addresses per link
    2. Shared physical addresses plus packet ID/address in packet header

---

# Media Access (MAC) Layer

- **Data slicing**
  - Split data packets into multiple bus word/frame transactions

  - ➤ Optimized data slicing utilizing supported bus modes (e.g. burst)

- **Arbitration**
  - Separate individual bus transactions in time
    1. Centralized using arbiters
    2. Distributed
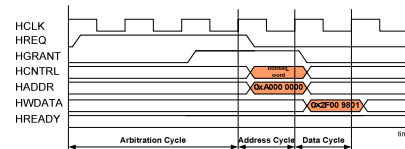 - ➤ Insert arbiter components

Protocol, Physical Layers

- **Bus interface**
  - Generate state machines implementing bus protocols
  - Timing-accurate based on timing diagrams and timing constraints
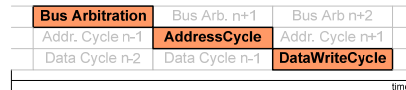
  ➢ Bus protocol database

- **Port mapping and bus wiring**
  - Connectivity of component ports to bus, interrupt wires/lines
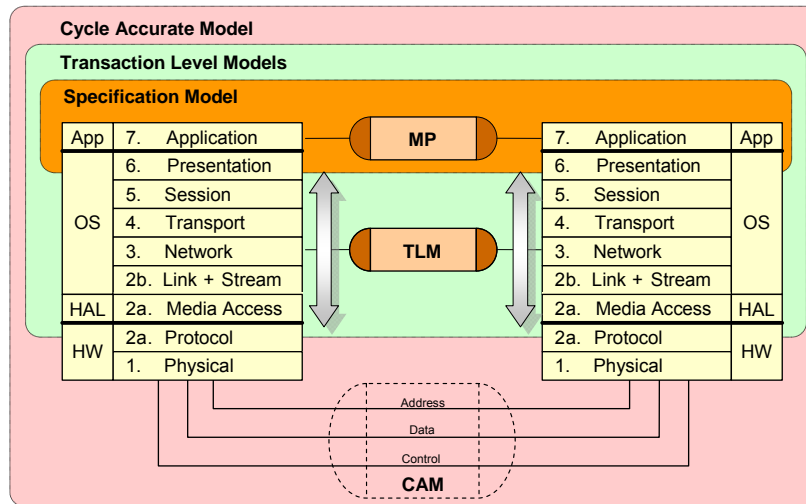 - ➢ Generate top-level system netlist

---

Outline

✓ **Introduction**

✓ **Models of Computation**

✓ **System Design**

✓ **Processor Modeling**

✓ **Communication Modeling**

- **System Models**
  - Specification model
  - Transaction-level models
  - Bus-cycle accurate model
  - Cycle-accurate model
- **Summary and Conclusions**

System Models

- **From layers to system models…**

Cycle Accurate Model

Transaction Level Models

Specification Model

App	7.	Application				7.	Application	App
	6.	Presentation				6.	Presentation	
	5.	Session				5.	Session	
OS	4.	Transport				4.	Transport	OS
	3.	Network				3.	Network	
	2b.	Link + Stream				2b.	Link + Stream	
HAL	2a.	Media Access				2a.	Media Access	HAL
HW	2a.	Protocol				2a.	Protocol	HW
	1.	Physical				1.	Physical	

MP

TLM

Address

Data

Control

CAM

# Specification Model

- **Abstract, high-level system functionality**
  - Computation
    - Processes
    - Variables
  - Communication
    - Sync./async. message-passing
    - Memory interfaces
    - Events

P1

P2

Abstract
MP Channel

Abstract
MP Channel
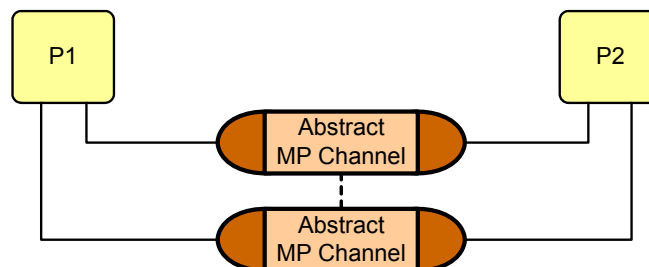
Network TLM
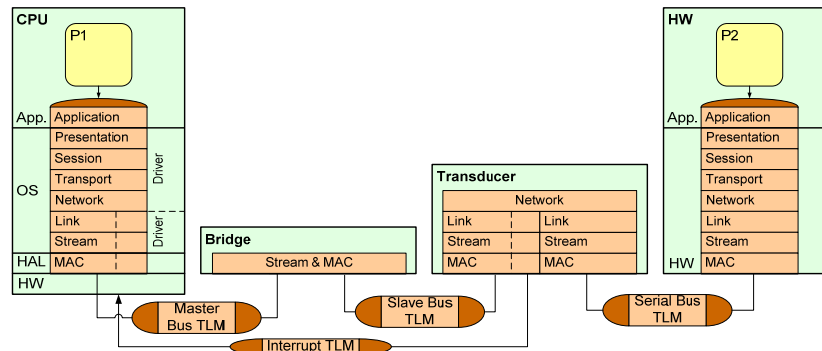
- **Topology of communication architecture.**
  - PEs + Memories + CEs
  - Upper protocol layers inserted into PEs/CEs
  - Communication via point-to-point links
    - Synchronous packet transfers (data transfers)
    - Memory accesses (shared memory, memory-mapped I/O)
    - Events (control flow)
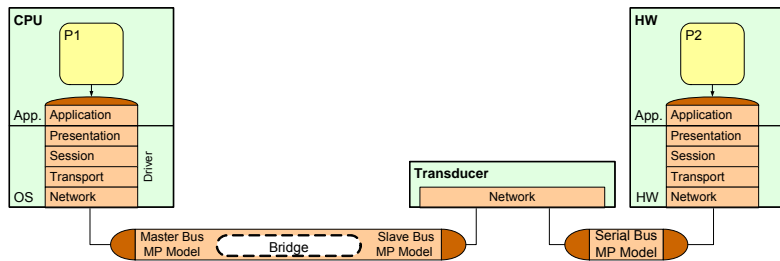
Protocol TLM

- **Abstract component & bus structure/architecture**
  - PEs + Memories + CEs + Busses
  - Communication layers down to protocol transactions
  - Communication via transaction-level channels
    - Bus protocol transactions (data transfers)
    - Synchronization events (interrupts)

Bus Cycle-Accurate Model (BCAM)

- **Component & bus structure/architecture**
  - PEs + Memories + CEs + Busses
  - Pin-accurate bus-functional components
  - Pin- and cycle-accurate communication
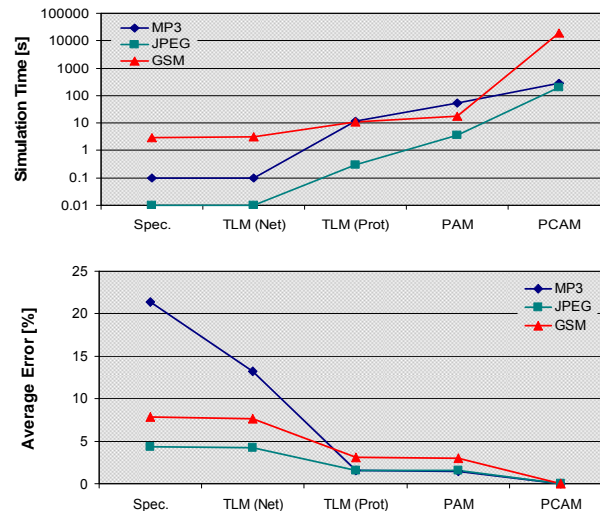   - Bus and interrupt protocols
    - Pins and wires
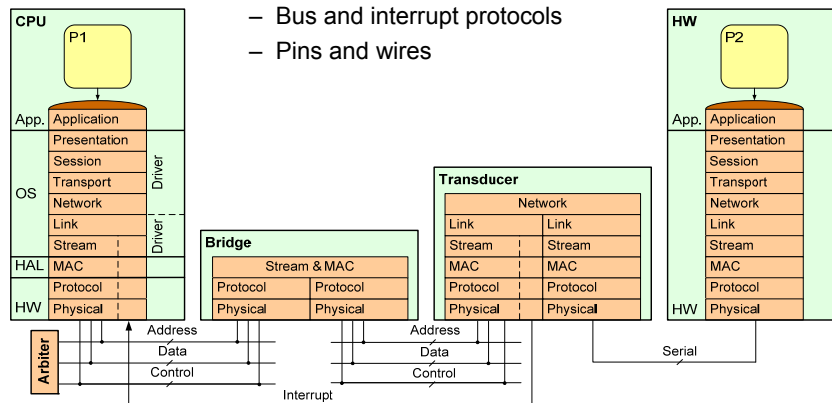
Cycle-Accurate Model (CAM)

- **Component & bus implemenation**
  - PEs + Memories + CEs + Busses
  - Cycle-accurate components
    - Instruction-set simulators (ISS) runing final target binaries
    - RTL hardware models
    - Bus protocol state machines
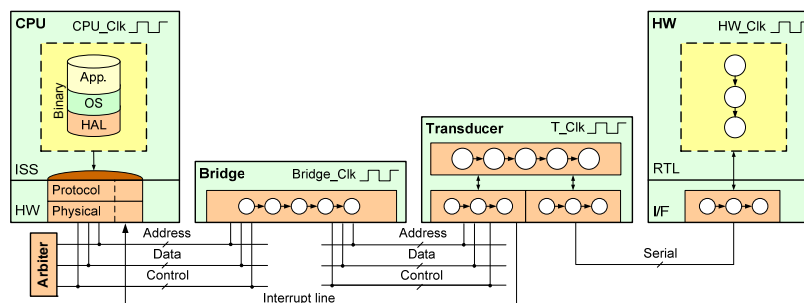
Modeling Results

Summary and Conclusions

- **Modeling of system computation and communication**
  - From specification
    – System behavior, Models of Computation (MoCs)
  - To implementation
    – Layers of implementation detail
  - ➢ Flow of well-defined models as basis for automated design process

- **Various level of abstraction, accuracy and speed**
  - Functional specification
    – Native speeds but inaccurate
  - Traditional cycle-accurate model (CAM)
    – 100% accurate but slow
  - ➢ Transaction-level models (TLMs)
    - ➢ Fast and accurate virtual prototyping