CE 654 – Embedded Systems

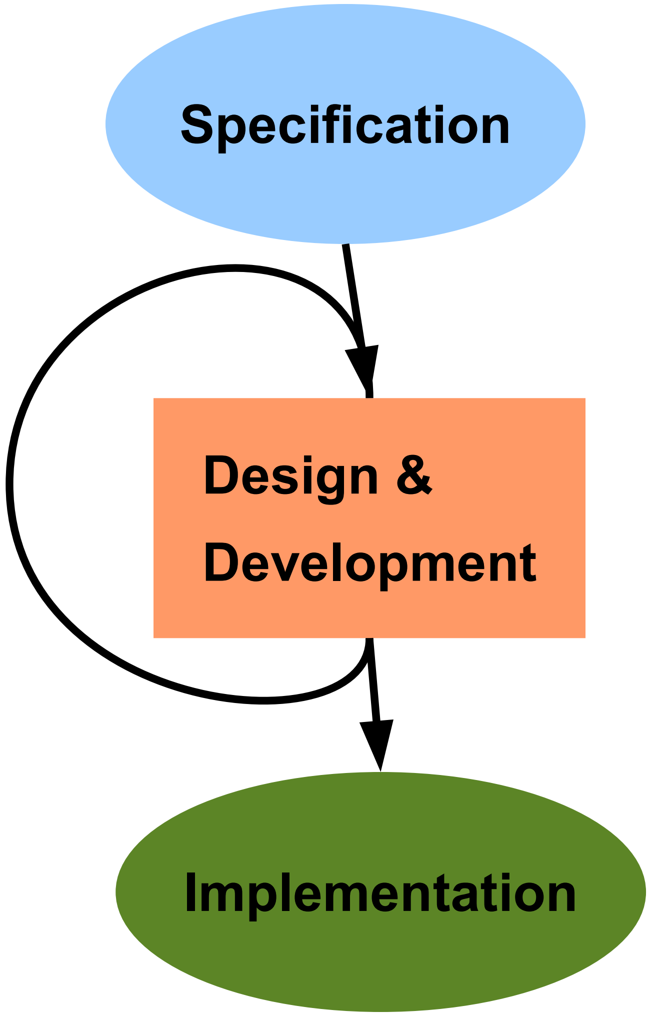
Lecture 2

Specification and Modeling of Embedded Systems

Nikos Bellas

Computer and Communications Engineering Department
University of Thessaly

Specifications in Embedded Systems



Specifications in Embedded Systems

- Requirements and Constraints: informal description of what the customer wants
- Specification: Detailed, technical description of what the team will deliver.
- Requirements and analysis phase links customer with designers

Types of specifications, requirements and constraints

- Functional: Input-Output relationships

- Non-functional:
  - Timing
  - Power consumption
  - Production cost
  - Physical size, weight
  - Time to market
  - Safety requirements
  - Environmental requirements

Specifications, requirements and constraints

- They should be
  - Correct
  - Unambiguous
  - Complete
  - Verifiable: we should be able to check that the specifications, requirements, and constraints are satisfied in the final system
  - Consistent: they do not contradict each other
  - Modifiable: they can be updated easily
  - Reasonable: the specifications, requirements and constraints should be easily understood, and designers should know why they exist

Setting requirements and constraints

- Techniques include
  - Customer interviews
  - Comparisons with competitors
  - Feedback from sales and marketing departments
  - Experience from prototypes and similar products

Setting specifications

- A complete specification captures non-functional requirements (speed, power, cost, size) and the behavior of the system by providing:
  - Relation between inputs and outputs
  - Possibly internal states
  - Algorithm for the system functionality
- The design team must have the capability to verify the correctness of the specification and to compare the specification with the implementation.
- Basic specification styles:
  - ✓ Textual
  - ✓ Graphical
  - ✓ Mixed

Setting specifications properties

- Specifications can be formulated in:
  - ✔ Natural language (informal)
  - ✔ Specification languages or models (more detailed)
- A specification language or model has to be
  - ✔ able to express the basic properties and basic aspects of the system behavior in a clear manner
  - ✔ able to check the system requirements and to ensure the synthesis of an efficient system implementation
- Depending on the particularities of the system or parts of the system, adequate languages or models have to be chosen
- The specification language or model has to contain the appropriate constructs (textual or graphical) in order to express the system's functionality and requirements.

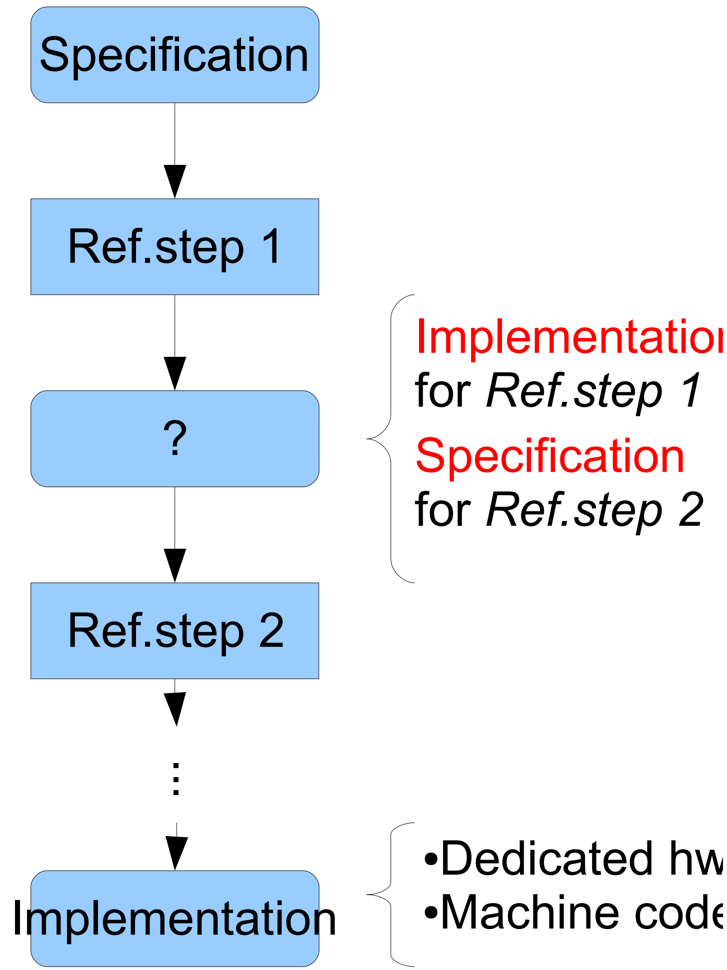
Specifications and refinement

- The design process consists of a sequence of steps:
  - each step performs a transformation from a more abstract description to a more detailed one

- A design step takes a specification (model, code, etc.) of the design at a level of abstraction and refines it to a lower one.

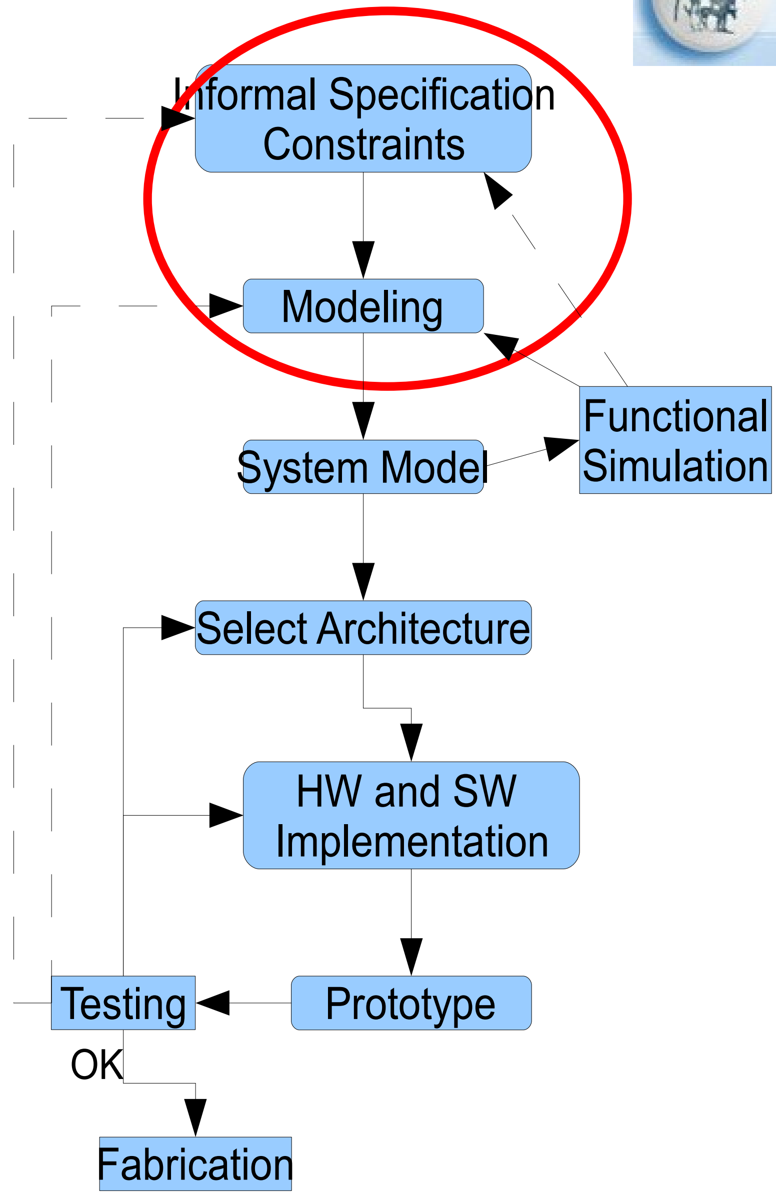
From specification to implementation

The designer gets a specification (behavior description and other properties) as an input and finally has to produce an implementation, after a sequence of refinement steps.



Simplified design flow

1. Start from some informal specification of functionality and a set of constraints (time, power, cost limits, etc.)
2. Generate a more formal specification of the functionality, based on some modeling concept (e.g. finite state machines). This model may be in Matlab, C, UML
3. Simulate the model in order to check the functionality. If needed, make adjustments.
4. Choose an architecture (µprocessor, buses, etc.) such that the cost limits are satisfied and, hopefully, time and power constraints will be fulfilled.
5. Build a prototype and implement the system
6. Verify the system: time, power constraints satisfied?
   - Go back to 4 and choose another architecture to start a new implementation
   - Or negotiate with the customer on the constraints.

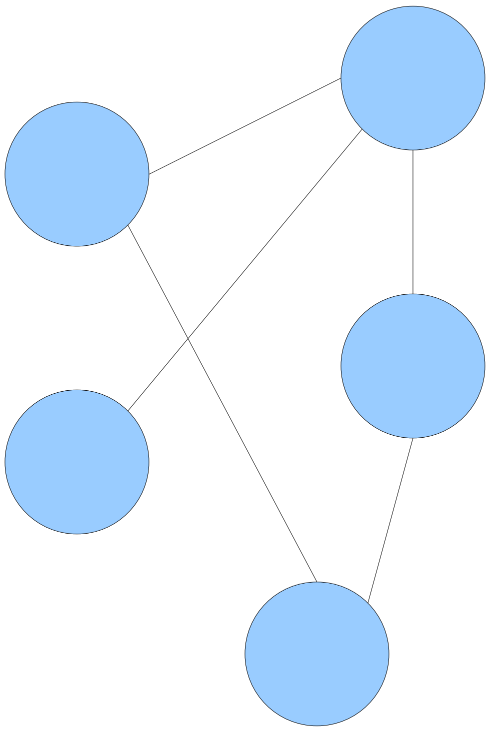


System modeling: use of computation models

- A computation model assists the designer to understand and describe the behavior of a system by providing a “vehicle” to compose the system's behavior from simpler objects.
- A computation model provides a set of objects and rules for composing those objects in order then to be able to formally represent (model) the behavior of our system
- A system is represented as a set of components, which can be considered as isolated modules (often called processes or tasks) interacting with each other and the environment
- Usually, computation models are based on some kind of graph representation
- The computation models define the behavior and interaction mechanisms of the system modules.
- The computation models help the designer to formally analyze, estimate some useful parameters, verify (at the high level) the system by using the proper tools.

System modeling: use of computation models

- Thus, computation models usually refer to:
  - ✓ How each module (process or task) performs internal computation
  - ✓ How the modules transfer information between them (communication)
  - ✓ How they are related in terms of execution order and synchronization
- Some computation models do not refer to aspects related to the internal computation of the modules but only to modules' interaction.



Order of execution

- Two different approaches for ordering the execution of tasks in computation models
  - ✓ Data-driven
  - ✓ Control-driven

Data-driven order of execution

- The system is specified as a set of processes without any *explicit* specification of the ordering of executions.

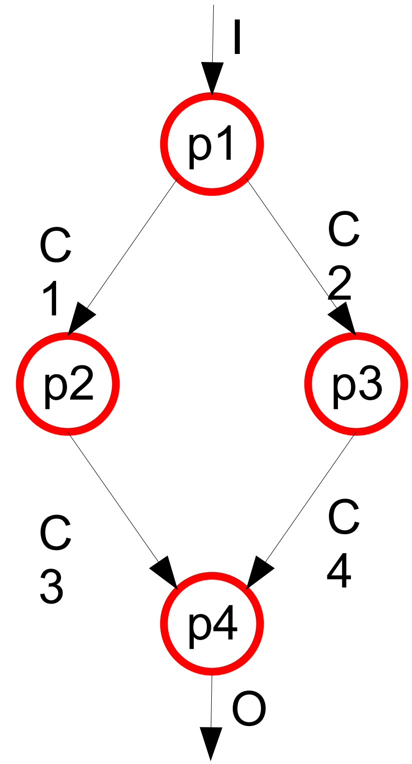
- The execution order of processes (and the possible parallelism) is determined solely by data dependencies
  - Typical for many DSP applications.

Data-driven order of execution

- Processes communicate by passing data through FIFO channels
- Each process is blocked until there is sufficient data in the channel.

A process that tries to read from a channel waits until data is available.

Process p1 (in a, out x, out y) {...};
Process p2 (in a, out x) {...};
Process p3 (in a, out x) {...};
Process p4 (in a, in b, out x) {...};
channels I, O, C1, C2, C3, C4;



p1(I, C1, C2)
p2(C1,C3);
p3(C2, C4);
p4(C3,C4,O);

It doesn't matter in which order they are expressed

Control-driven order of execution

- The execution order of processes is given explicitly in the system specification
- Explicit constructs are used to specify sequential execution and concurrency

```
module p1
....
end p1
module p2
....
end p2
module p3
....
end p3
module p4
....
end p4
```

```
run p1
run p2  || run p3
run p4
```

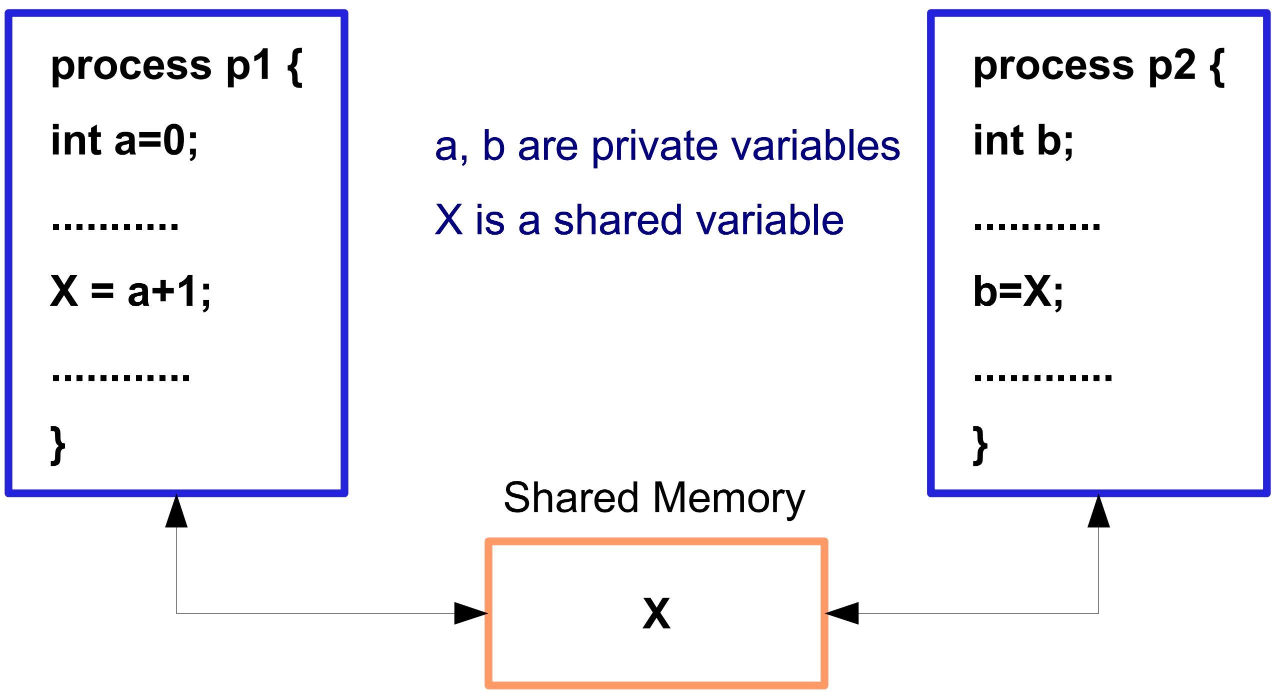
- Here, *p1* starts first, and has to finish before the beginning pf *p2* and *p3*
- *p2* and *p3* start in parallel
- Both *p2* and *p3* have to finish before *p4* starts

Communication and Synchronization

- Processes have to communicate to exchange information
- Various communication models are used:
  - Shared memory
  - Message passing
    - Blocking
    - Non-blocking
- Synchronization cannot be separated from communication. Any interaction between two processes implies a certain degree of communication and synchronization
- Synchronization: one process is suspended until another one reaches a specific point in its execution

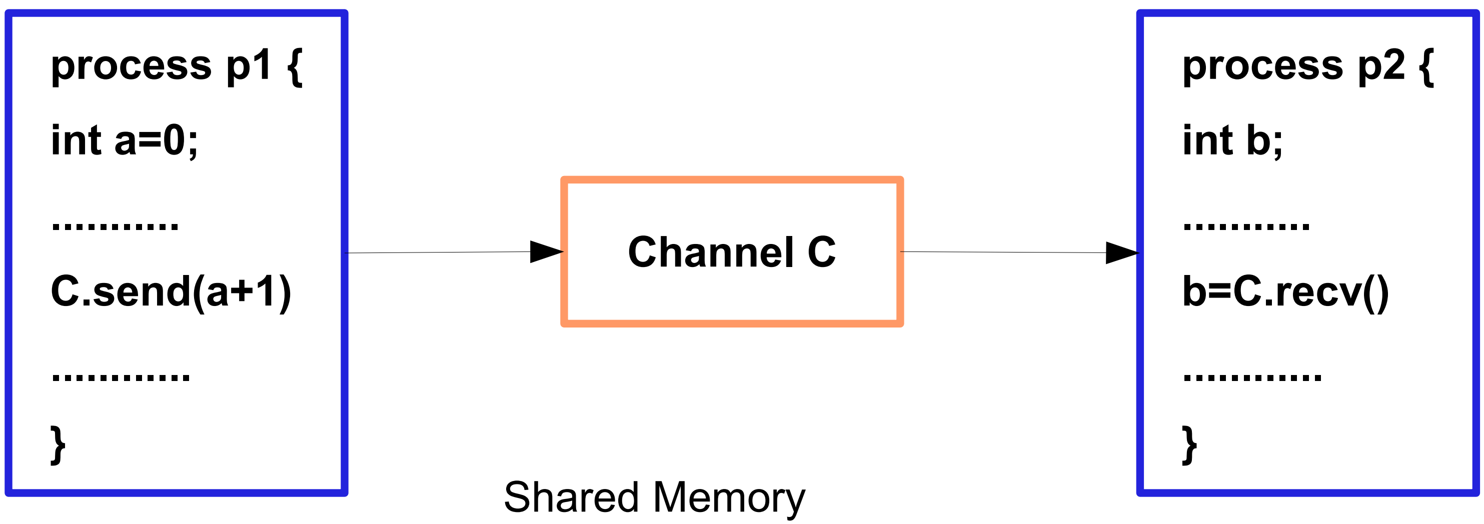
Shared memory communication

Processes communicate by reading and writing to shared variables in a global memory space



Message-passing communication

Messages that carry data pass through an abstract communication medium called channel



This communication model is adequate for describing distributed systems.

Message-passing communication

- Blocking communication
  - A communicating process blocks itself until the receiving process is ready for data transfer
  - The two processes have to synchronize before communication

Message-passing communication

- Non-blocking communication
  - The communication is asynchronous. However, buffers have to be inserted between processes to accommodate lack of synchronization
  - The sending process has to place a message to the buffer and continues execution
  - The receiving process reads the next message from the buffer when it is ready to do so

Common computation models

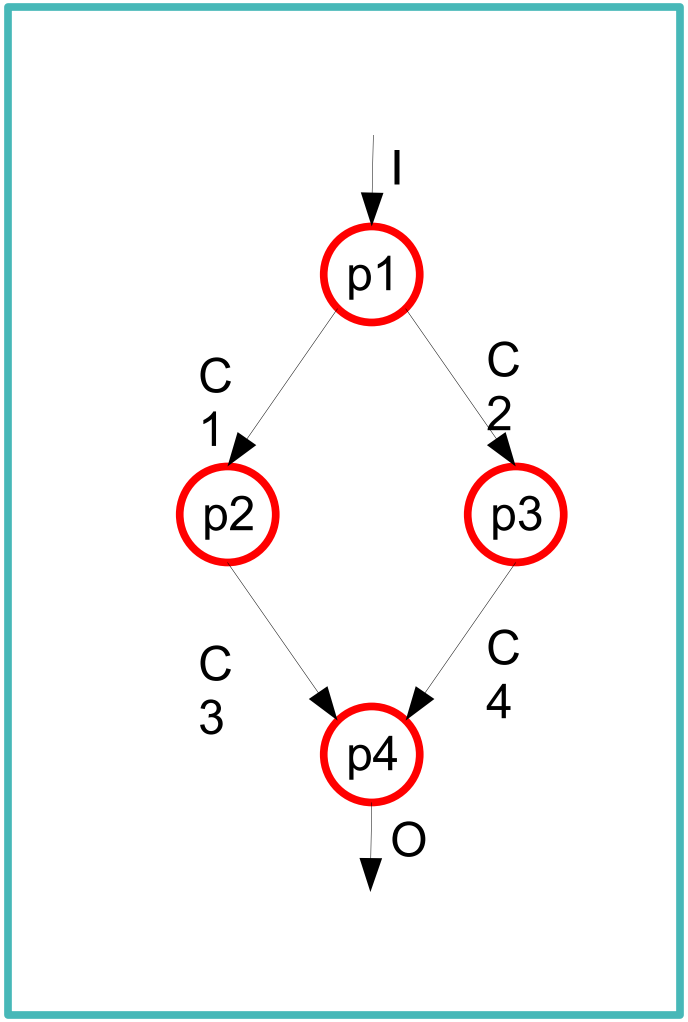
- Different computation models provide different properties
- We choose the appropriate computation model for the application domain we are working on
- The following computational models are commonly used to describe the functionality and structure of embedded systems
  - ✓ Data flow models
  - ✓ Finite state machines
  - ✓ Petri Nets

Common computation models

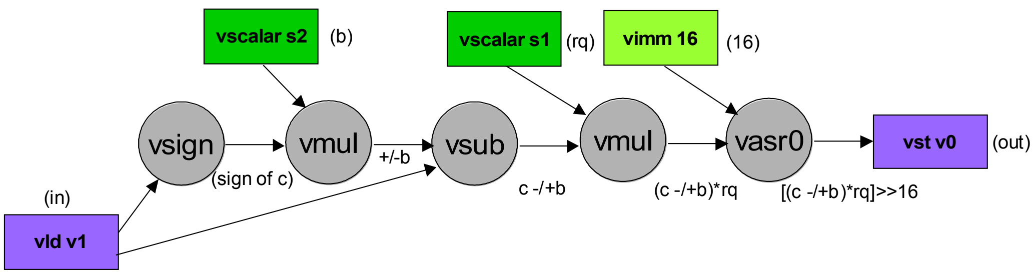
- Most applications can be classified as control-dominated or data-dominated
- A control-dominated application is dominate by monitoring inputs and reacting by setting control outputs
- A data-dominated application mainly consists of transforming streams of input data to streams of output data

Data flow models

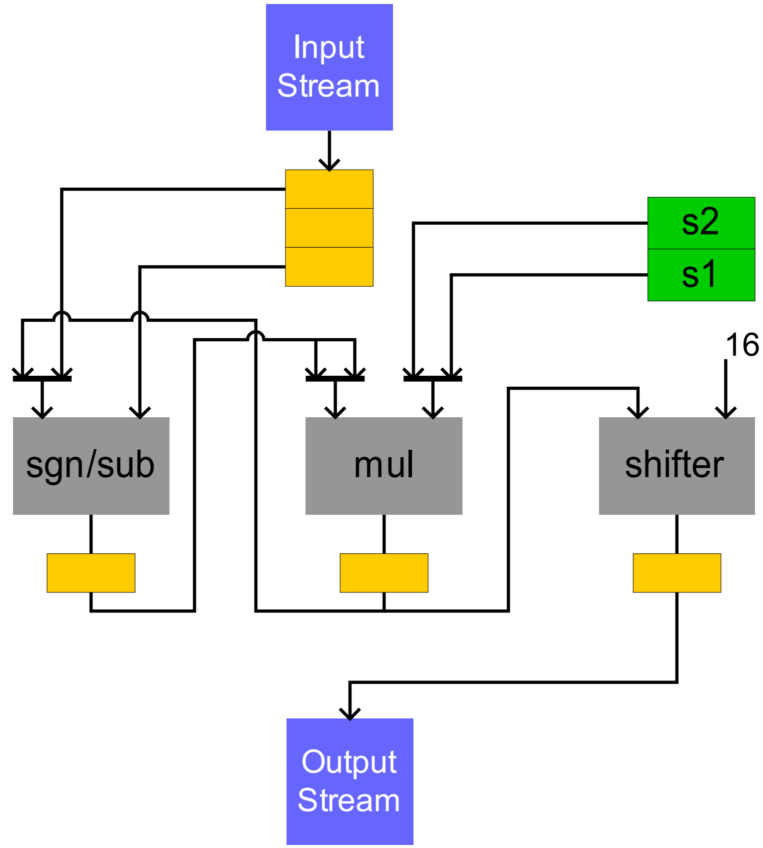
- Systems are specified as directed graphs where:
  - ✓ Nodes represent computations (processes)
  - ✓ Arcs represent sequences (streams) of data
- Suitable for signal processing algorithms that are expressed as block diagrams (filters, encoders



Data flow model example



- vsign produces -1, 0, 1 for <0, ==0, >0
- Scalar s1 is rq
- Scalar s2 is b
- Vasr0 is arithmetic shift right and truncate towards zero i.e. integer divide by power of 2



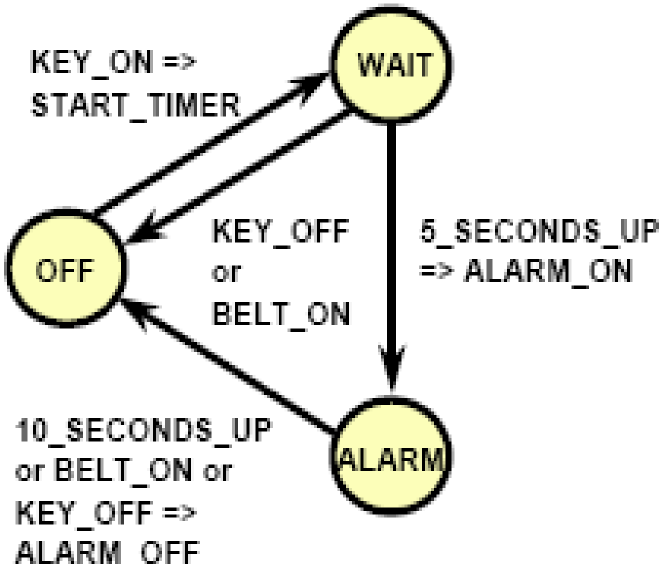
Finite State Machines

- The system is specified by representing its states and its transitions from state to state
- One particular state is specified as the initial one
- Finite number of states and transitions
- Transitions are triggered by input events
- Transition generate outputs
- FSMs are used to model control-dominated reactive systems, i.e. react on inputs with specific outputs
- Not too much computation

FSM example

Informal Specification

- If the driver
  - turns on the key, and
  - does not fasten the seat belt within 5 seconds
- then sound the alarm
  - for 5 seconds, or
  - until the driver fastens the seat belt
  - or until the driver turns off the key



No explicit condition => implicit self-loop in the current state

Finite State Machines

- Complex systems tend to have a very large number of states particularly in case of concurrency. This is called state explosion.
- Expressing such a system with a single FSM is very difficult
- There are two important tools that simplify the FSM modeling:
  - ✔ Hierarchy
  - ✔ Concurrency
- These tools only reduce the size of the graphical representation of the FSM. The inherent complexity does not change.
- The FSM model that uses these two mechanisms is called Hierarchical/Concurrent FSM (HCFSM)

Finite State Machines

- Hierarchy
  - A single state *s* can represent a whole FSM *F*
  - Being in state *s* means that the FSM *F* is active, and the system is in one of the states of *F*.
- Concurrency
  - Two or more finite state machines are viewed as being simultaneously active
  - The two FSMs operate in parallel or they may communicate

- Another option is the Program State Machine (PSM) model that extends FSMs to allow use of sequential program code in order to define a state's action.

Computation models and specification languages

- A single specification language can be used for the specification of a whole system.
- This does not mean that we have a homogeneous specification (one computational model)
- It means that the specification language can cover multiple computation models, each one describing components of the system
- For example, it is possible to specify in the same HDL language parts of the program using the FSM model, and parts of the program using the data-flow model
- Several languages are capable of describing a system
  - Specific languages for the hardware part (Verilog) and the software part (C, or Java)

Specification languages

- General purpose programming languages (Matlab, C, C++, Java) or hardware programming languages (VHDL, Verilog, SystemC). They may support multiple models of computation
- Synchronous languages (FSM-based): Esterel
  - It describes set of interacting synchronous FSMs
- Languages for description of networks of communicating processes: UML, SDL
- Streaming languages for hardware description (ImpulseC, mitrion-C, etc.)