Y-chart methodology and Models of Computation and Architecture

Bart Kienhuis, Ed Deprettere, Kees Vissers,
Pieter van der Wolf, Paul Lieverse
Edward Lee.

By Bart Kienhuis

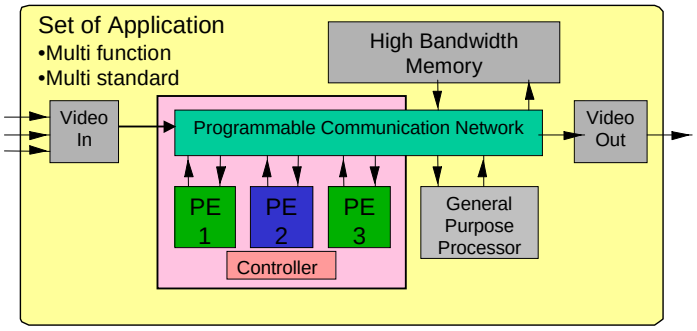
Berkeley USA, University of California,
Berkeley, Dept EECS
Cory Hall

TU Delft

GSRC

PHILIPS

Platform



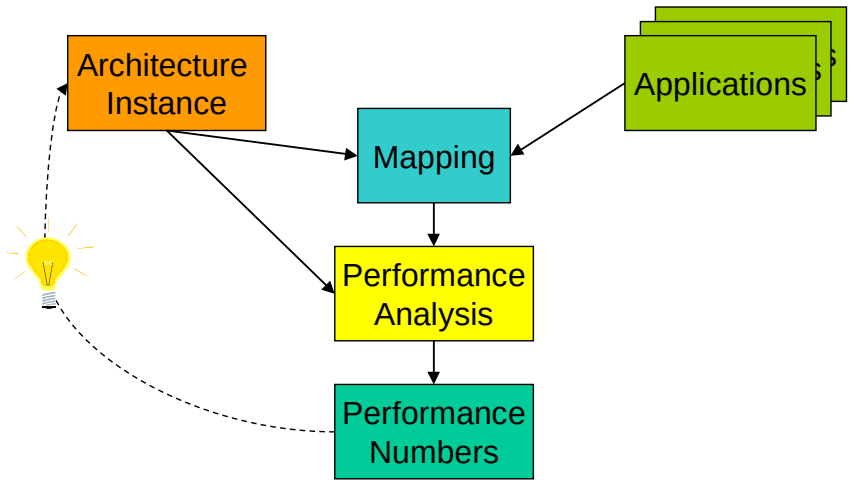
High Performance DSP Architecture

Design Choices
•Functionality PEs
•Packet Length
•Control Protocol

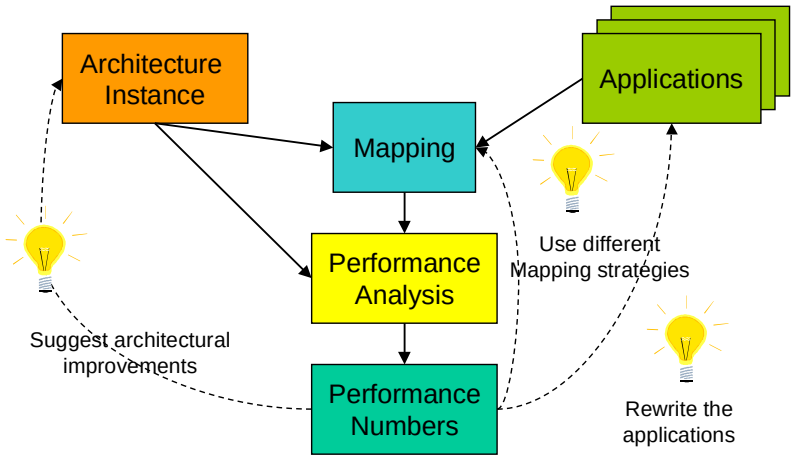
What Methodology to use to solve this problem?

Constraints
•throughput
•flexibility
•silicon cost
•power

Y-chart Approach

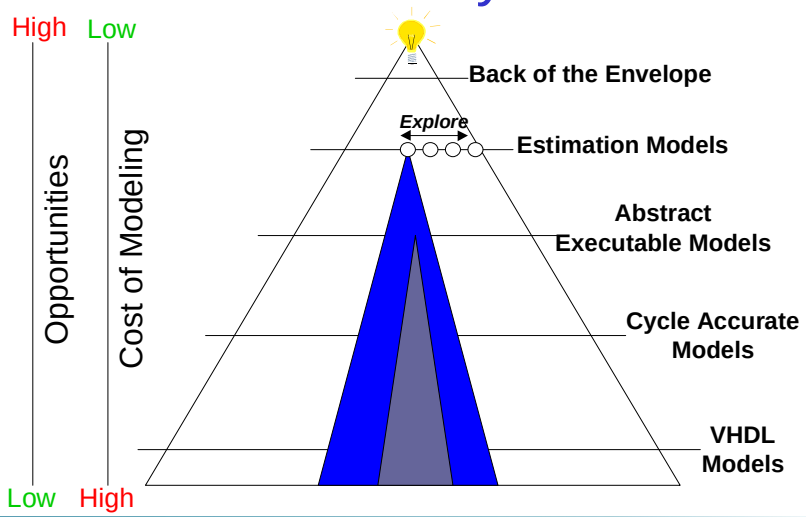


Y-chart Approach

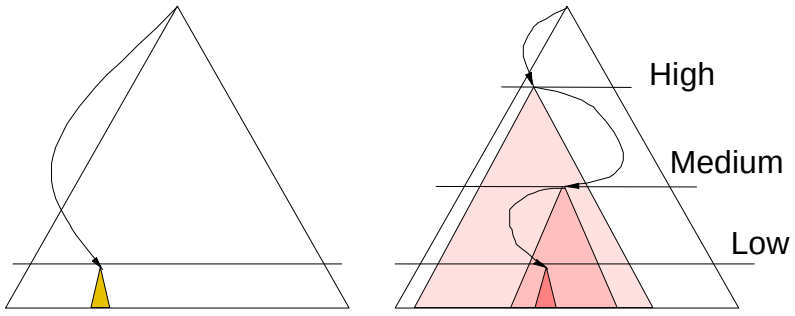


Different ways to improve a system.

Abstraction Pyramid



Stepwise Exploration of the Design Space

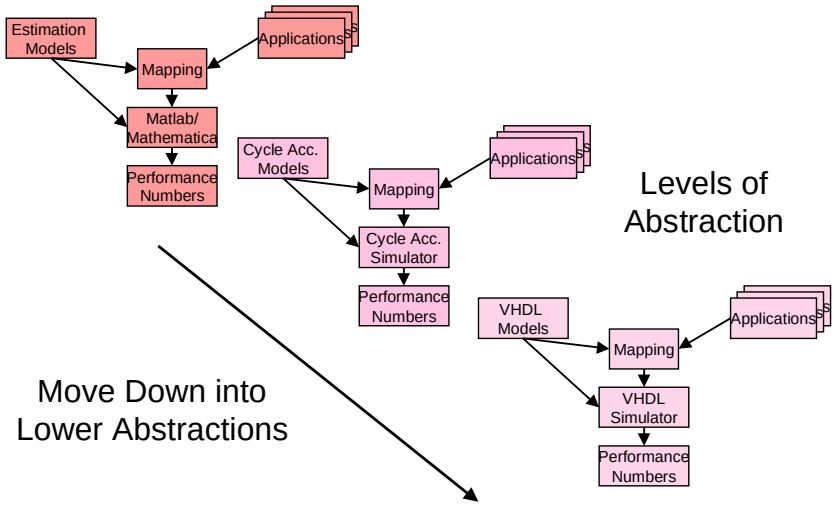


Traditional approach

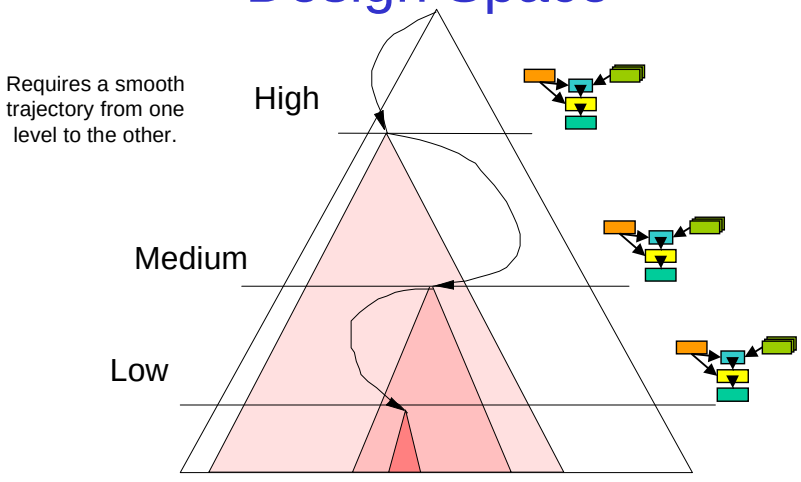
Y-chart approach

Stepwise refinement of the Design Space of an Architecture

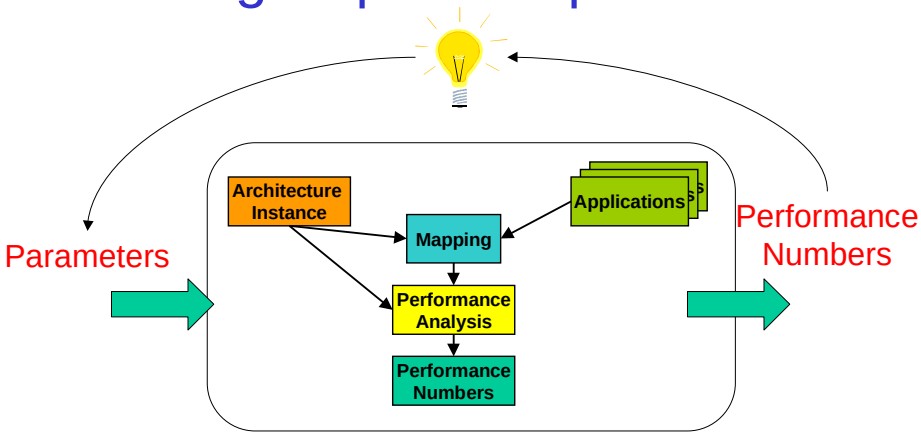
Stack of Y-chart Environments



Stepwise Exploration of the Design Space

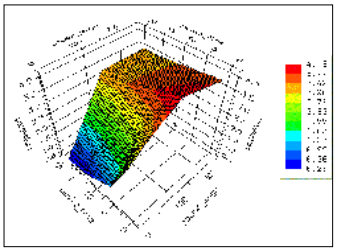


Design Space Exploration



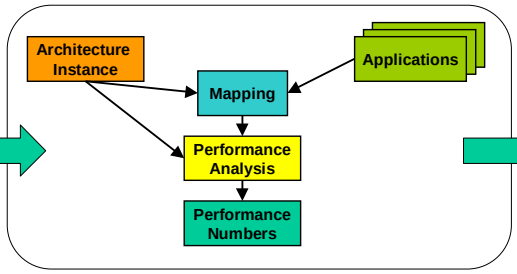
The Acquisition of Insight

Design Space Exploration



Set up a number of Experiments

- •CPU=arm
- •Packet Size=100
- •Control = Round Robing
- •PE1= {fa,fb,fc}

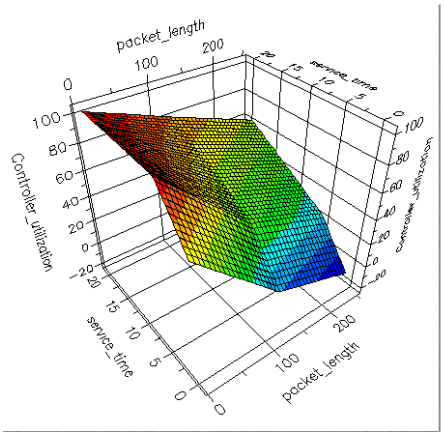


- •Throughput=10
- •utilization=45%
- •energy=0.1w

Parameters

Performance Numbers

Result of an Exploration



(negative utilization values are the result of interpolation)

- Making the proper trade-offs
  - Knee points
- Quantifying design choices
- Multi variable optimization problem

Summary of the Y-Chart approach

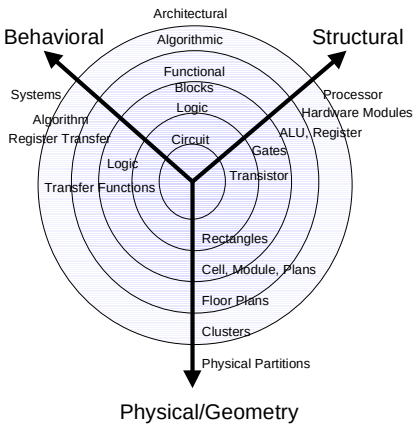
- It permits designer to quantify design choices in the architecture, the algorithms, and the mapping.
- It permits the systematic exploration of the design space of a system.
- It allows for the consideration of trade-off between various metrics for an system that obeys set-wide design objectives.
- It is invariant to a specific design level.
- It requires an explicit definition of a platform and the applications. This fosters reuse.

Historical Perspective:
Separating Architecture from Applications

- The Y-chart is a methodological representation stressing the need of separating applications from architecture at higher levels of abstraction. To couple applications and architecture, the Y-chart introduces an explicit *mapping* step.
- In computer architecture design, the separation between architecture and application has already been in use for quite some time even though the term “architecture” in that domain reflects typically the Instruction Set Architecture that is not normally viewed as an architecture in embedded system applications.
- In the design of programmable embedded systems, the importance of separation between architecture and application and its *methodological* consequences have been examined in:
  - F. Balarin, et al., Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Publishing, 1997
  - Kienhuis et al. “An Approach for Quantitative Analysis of Application-specific Dataflow Architectures”, Conf. on Application-specific Systems, Architectures and Processors (ASAP), Zurich 1997.

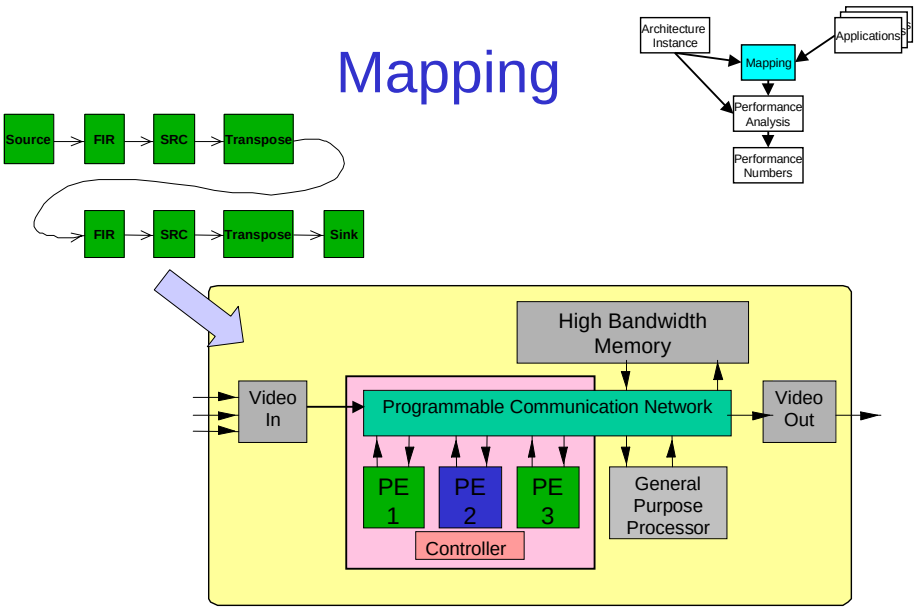
Historical Perspective:
Gajski and Kuhn’s Y-chart

- In Gajski and Kuhn's Y-chart,each axis represents a view of a model: *behavioral*, *structural*, or *physical* view.
- Moving down an axis represents moving down in level of abstraction, from the *architectural* level to the *logical* level to, finally, the *geometrical* level.
- The Gajski and Kuhn’s Y-chart expresses the manual design process of *refinement*.

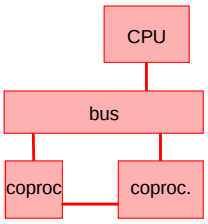


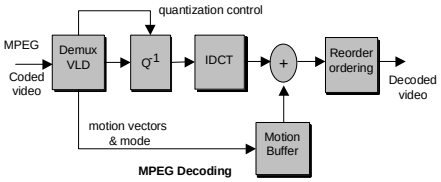
(D. Gajski, “*Silicon Compilers*”, Addison-Wesley, 1987).

Mapping



Mapping



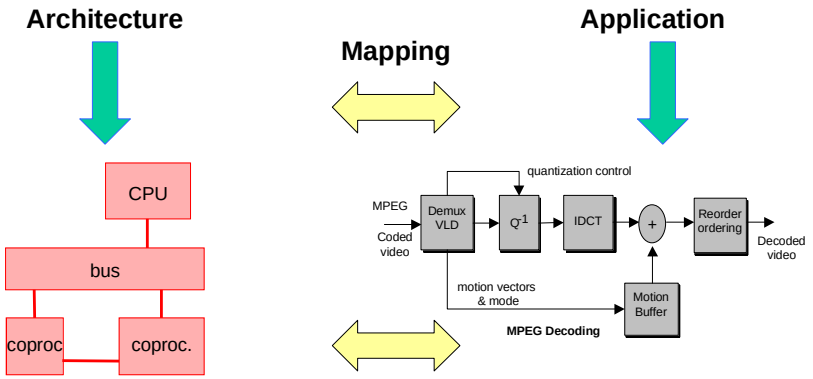


Both described a network of components that perform a particular function and that communication in a particular way

Architecture:
- Resources
  - ALUs, CORDICS, PEs
  - Registers, SRAM, DRAM
  - Busses, Switches
- Communication
  - Bits, Signals

Application:
- Computations
  - IDCT, SQRT, Quantizer
- Communication
  - Pixels, Blocks

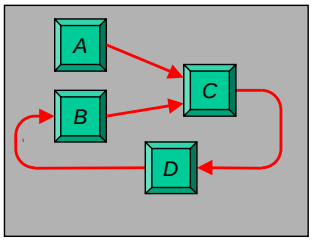
Mapping



Can we formalize the description of these networks?
"*Models of Architecture*" and "*Models of Computation*"

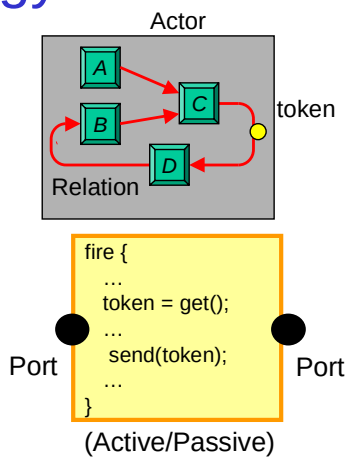
Model of Computation

A *Model of computation* is a formal representation of the operational semantics of networks of functional blocks describing the computations.

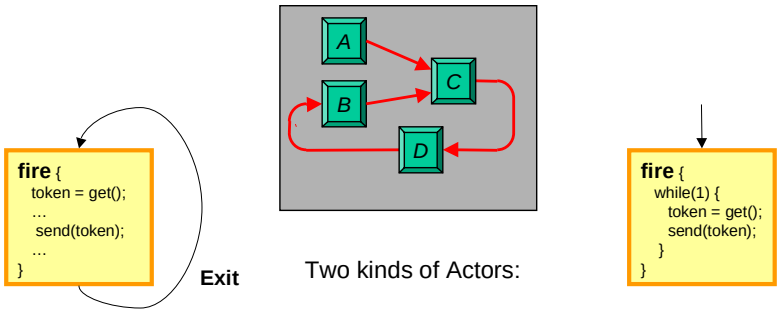


Model of Computation Terminology

- Actor
  - describes the functionality
- Relation
  - The actors are connected with each other using relations.
- Token
  - the exchange of a quantum of information.
  - It presents is a signal
- Firing
  - a computation
  - interaction with other actors



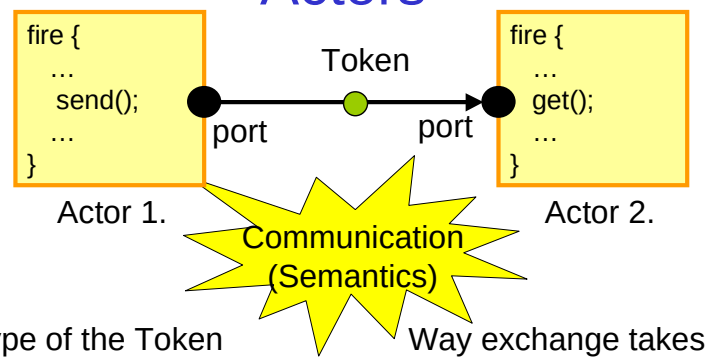
Active/Passive Actors



Passive Actor:
- Scheduler needed.
  - Schedule ABBCD
- A firing needs to terminate
- Fire-and-exit behavior

Active Actor:
- Schedules itself
- A firing typically doesn't terminate
  - Endless while loop
- Process behavior

Communication between Actors

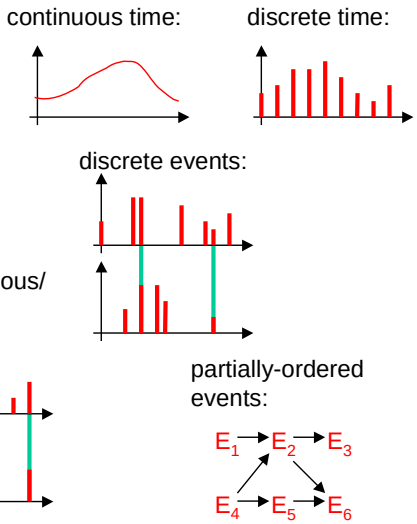


Data Type of the Token
- •Integer, Double, Complex
- •Matrix, Vector
- •Record

Way exchange takes place
- •Buffered
- •Timed
- •Synchronized

Different Semantics

- Analog computers (ODEs)
- Discrete time (difference equations)
- Discrete-event systems (DE)
- Process networks (Kahn)
- Sequential processes with rendezvous (CSP)
- Dataflow (Dennis)
- Synchronous-reactive systems (SR)
- Codesign Finite State Machines (CFSM)

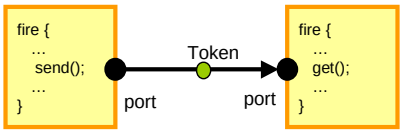


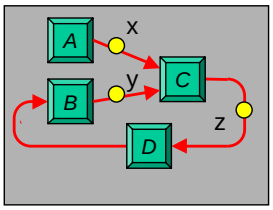
Synchronous/Reactive Models

- Network of concurrent executing actors
  - passive Actors
  - Communication is unbuffered
- Computation and Communication is instantaneous.
- A model progresses as a sequence of “ticks.”
- At a tick, the signals are defined by a fixed point equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_A(1) \\ f_b(z) \\ f_c(x,y) \end{bmatrix}$$

Fixed point equation

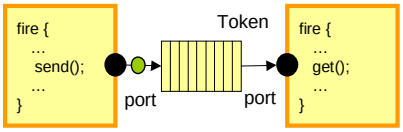


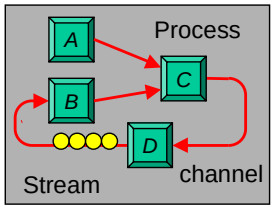


- Characteristics of SR Models
  - Tightly Synchronized
  - Control intensive systems

Process Network

- Network of concurrent executing processes
  - active Actors
  - Communicate over unbounded FIFOs
- Performing some operation, a blocking read or a non-blocking write

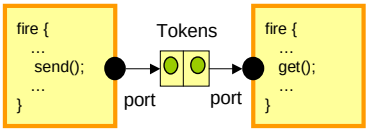


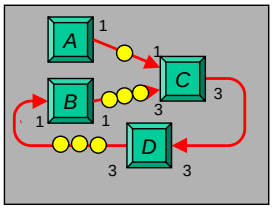


- Characteristics of Process Networks
  - Deterministic Execution
  - Doesn’t impose a particular schedule
  - (Dynamic) Dataflow

Synchronous Dataflow

- Network of concurrent executing actors
  - passive Actors
  - Communication is buffered
- A model progresses as a sequence of “iterations.”
- A “firing rule” determines the firing condition of an actor.
- At each firing, a fixed number of tokens is consumes and produces.



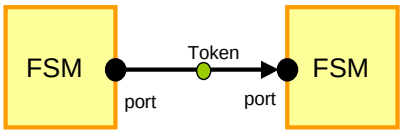


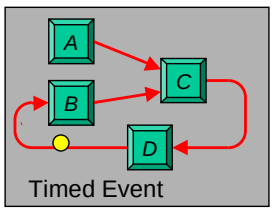
Schedule: **ABBBC**

- Characteristics of SDF
  - Compile time analyzable.
  - Memory/Schedule/Speed
  - Static Dataflow

Codesign Finite State Machine (CFSM)

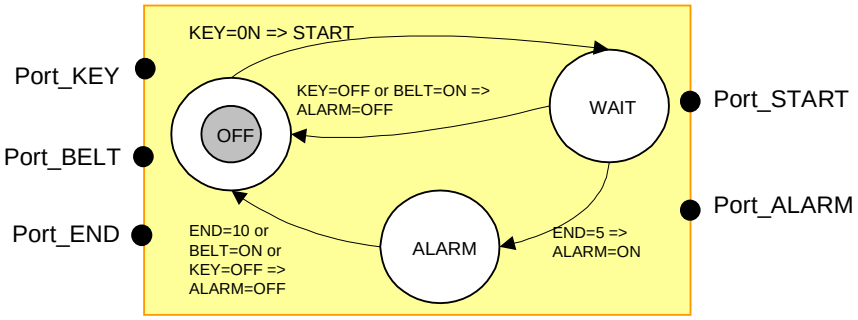
- Network of concurrent executing actors
  - Passive Actors
  - Synchronous locally
  - Asynchronous globally
- An “event” causes the evaluation (firing) of a FSM.





- Characteristics of CFSM
  - Compile time analyzable.
  - Reactive systems

Finite State Machine (FSM)



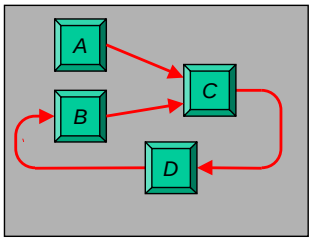
- FSM may only have one state active at the time
- FSM has only a finite number of states.

- More efficient way to describe sequential control.
- Formal semantics which allows for verifying various properties like safety, liveness, and fairness.

Model of Architecture

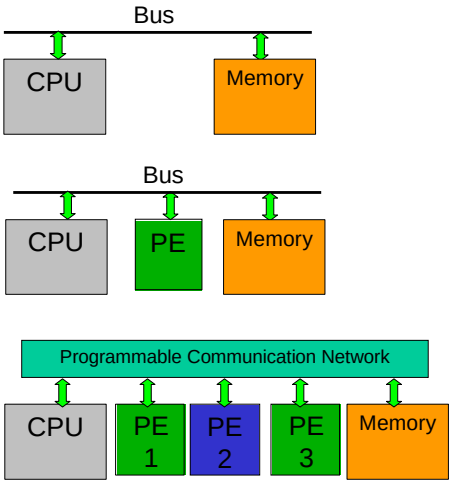
A *Model of architecture* is a formal representation of the operational semantics of networks of functional blocks describing architectures.

A,B,C and D are now hardware resources like CPUs, busses, Memory, and dedicated coprocessors.



Model of Architecture is similar to Model of Computation, but the focus is on the architecture instead of on the applications.

Examples



Control Dominated Tasks
•Sequential

Control/ Data Tasks
•Sequential
•Centralized computation

Data Dominated Tasks
•Concurrent / DMA
•Data flow
•Distributed computation

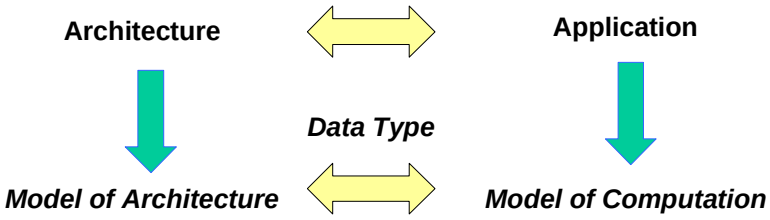
Low

Complexity

High

Less mature then MoC

Conclusion: Matching Models



When the MoC and MoA match, a simple mapping results

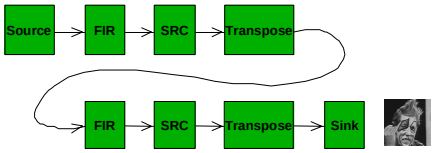
Application

Picture in Picture (PIP)

We will look at two platforms for the same application discussed here.

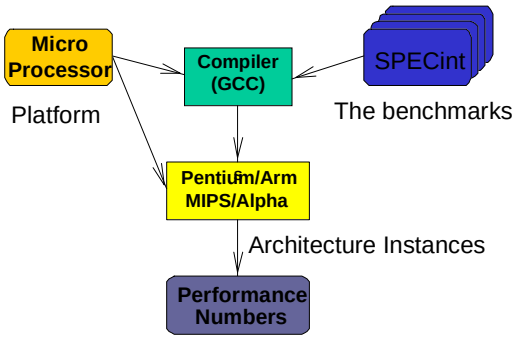
```
for i=1:1:10
  for j=1:1:10
    A(i,j)=FIR( …);
  end
end
for i=1:1:10,
  for j=1:1:10,
    A(i,j) =SRC( A(i,j) );
  end
end
for i=1:2:10,
  for j=1:1:10,
    … =Transpose( A(i,j) );
  end
end
```

The Algorithm

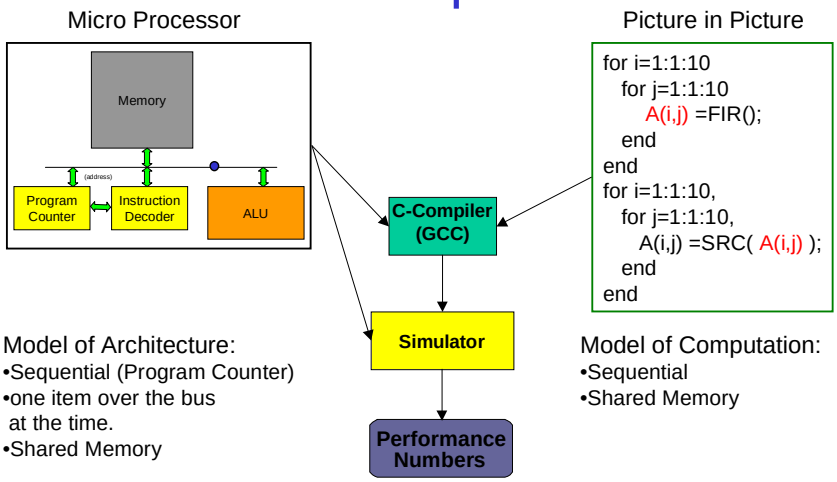


Putting it together example 1.

- Platform: Microprocessor “Von-Neumann architecture”



Putting it together example 1.



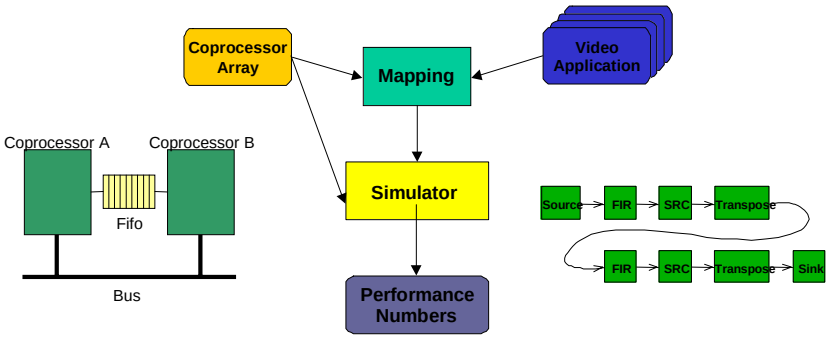
But Embedded Systems...

- But Embedded System are typically
  - Concurrent
  - Real-time
  - Heterogeneous
  - Application Specific

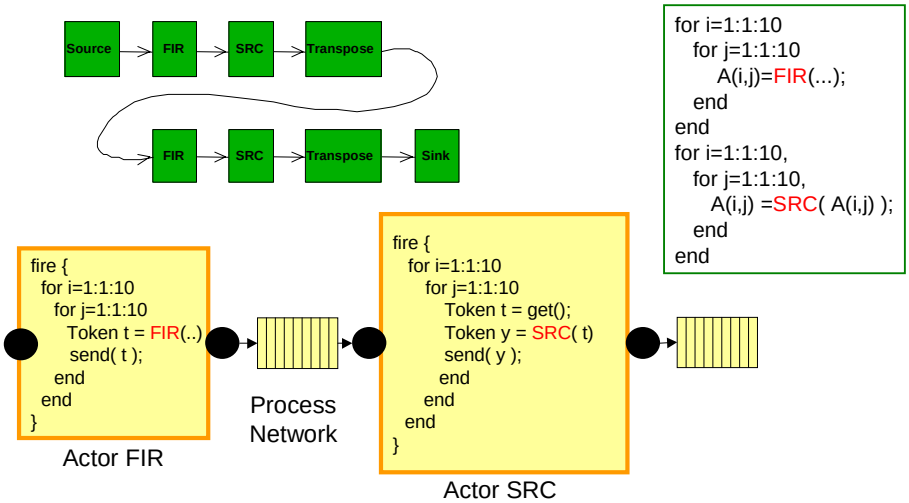
Your C/GCC compiler is not going to help you to solve the mapping problem in these embedded systems!

Putting it together example 2.

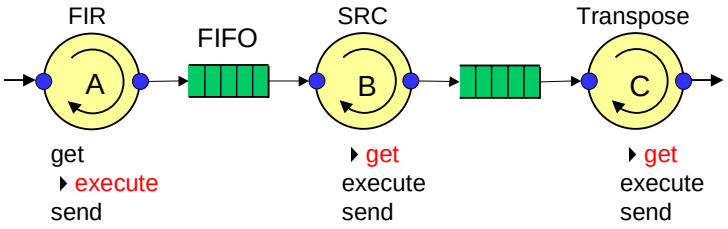
- Platform: Coprocessor Array



Application Modeling

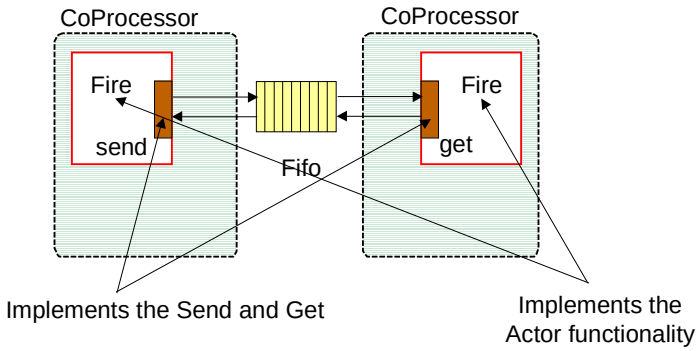


Application Modeling



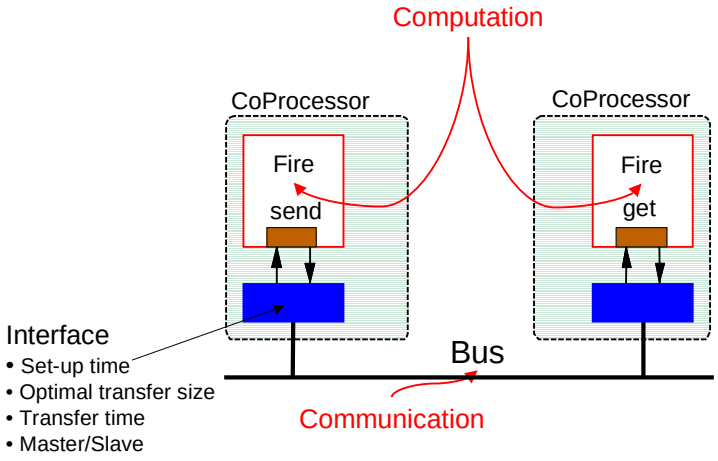
- Explicitly describes Communication and Computation
- Explicitly describes concurrency
- Doesn't impose a particular schedule

Architecture Modeling (FIFO)



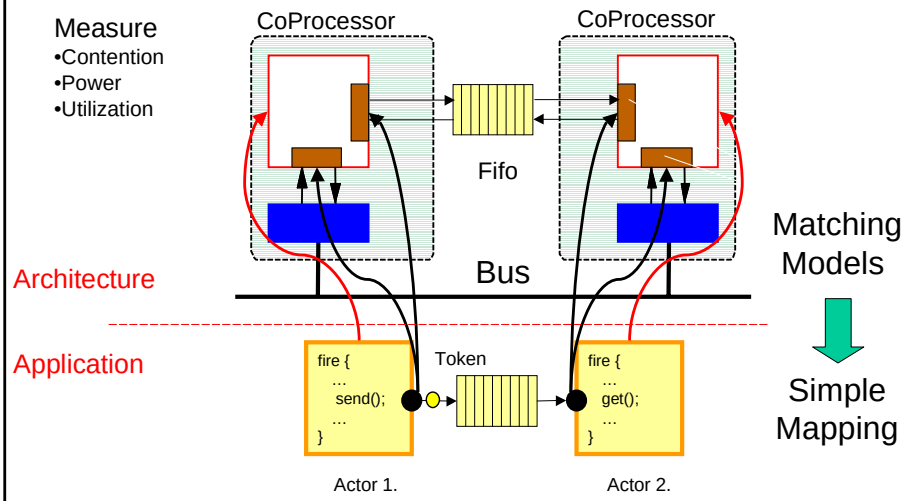
Abstract Architecture Modeling
- Cycle Accurate description

Architecture Modeling (BUS)



Exploiting the separation between *Communication* and *Computation*

Mapping



Once again: the Y-chart approach is about...

- Quantifying
  - Relentlessly quantifying design choices at each design level.
- Abstraction
  - Models of Computation / Models of Architectures
  - Exploiting Performance Trade-off
  - Stepwise exploration of design space
- Reuse
  - Reuse of applications
  - Reuse of platforms
  - Reuse of IP

References

- Y-chart approach
  - B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf, ``An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures'', In Proc. 11-th Int. Conf. on Application-specific Systems, Architectures and Processors, Zurich, Switzerland, July 14-16 1997.
  - F. Balarin, et al., “Hardware-Software Co-Design of Embedded Systems:The Polis Approach”, Kluwer Academic Publishing, 1997
  - B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf, ``The Construction of a Retargetable Simulator for an Architecture Template'', In Proc. 6-th Int. Workshop on Hardware/Software Codesign (CODES'98), Seattle, Washington, March 15 - 18 1998.
  - B. Kienhuis, ``Design Space Exploration of Stream-based Dataflow Architectures: Methods and Tools'', PhD thesis, Delft University of Technology, The Netherlands, January 1999. (Http://ptolemy.eecs.berkeley.edu/~kienhuis)
  - http://ptolemy.eecs.berkeley.edu/~kienhuis
- Model of Computation
  - Ptolemy web site (http://ptolemy.eecs.berkeley.edu)
  - W.-T. Chang, S.-H. Ha, and E. A. Lee, ``Heterogeneous Simulation -- Mixing Discrete-Event Models with Dataflow,'' invited paper, Journal on VLSI Signal Processing, Vol. 13, No. 1, January 1997.

References

- Mapping
  - Paul Lieverse, Pieter van der Wolf, Ed Deprettere, and Kees Vissers, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems" In: Proc. 1999 Workshop on Signal Processing Systems (SiPS'99), pp. 181-190, Taipei, Taiwan, Oct. 20-22 1999.
  - Ed F. Deprettere, Edwin Rijpkema, Paul Lieverse, Bart Kienhuis, “High Level Modeling for Parallel Executions of Nested Loop Algorithms”, In Proc. Application-specific Systems, Architectures and Processors ASAP2000, Boston, Massachusetts, July 2000.
  - Paul Lieverse, Pieter van der Wolf, Ed Deprettere, and Kees Vissers, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems" To appear in: Journal of VLSI Signal Processing for Signal, Image and Video Technology, special issue on the 1999 IEEE Workshop on Signal Processing Systems (SiPS'99).