System level design

- System level synthesis
  - starts from concurrent algorithms and generates communicating processors
- High level synthesis
  - is an implementation in terms of existing components or specifications of subsystems
- System level specification is *abstract* and *minimal* when only the necessary requirements and constraints are defined
- System implies:
  - Complexity
  - Abstraction

System features

- Semi-isolation
- Domain specificity
- Hierarchical composition
- Heterogeneity
- Coherency
- System level synthesis tasks
  - Partitioning
  - Mapping the logical communication structure onto a physical communication structure
  - Synthesis of communication protocols

System level Partitioning

- ***Partitioning*** **is the decomposition of a system into a set of subsystems**
- ***Explorative designs*** **require innovative architectures or carefully optimized modifications of known architectures in order to meet requirements**
  - **Here, designs that can be generated automatically using high-level and subsequent synthesis tools can serve as *prototypes*.**
- **Required: *estimation* algorithms that run much faster than synthesis and produce sufficiently exact (high fidelity) predictions**
- ***Goal*: automatic partitioning and design space exploration**

Clustering

- **Objects (and existing clusters) are successively merged into new clusters such that those with largest proximity are treated first.**
- ***Multi-stage clustering*** **(an extension to hierarchical clustering) applies two or more clustering processes in series, each of them emphasizing a different aspect of the design and using a different clustering criterion.**
- **Each clustering process generates a complete cluster tree.**
  Intermediate cluster trees are cut at certain value of proximity.
  The resulting clusters form the objects for the subsequent clustering stage.

Clustering strategies

- *Control clustering*. Operators, executed in serial.

$$\text{cprox}(a,b) = \begin{cases} 1 & \text{there is a path from } a \text{ to } b \\ 0 & \text{there is no path from } a \text{ to } b \\ p & \text{there is a path from } a \text{ to } b \text{ with branching operators} \end{cases}$$

- *Data clustering*. Reducing the interconnection cost.

  dprox(a,b) = CommonData(a,b) / (TotalData(a) + TotalData(b))

- *Schedule clustering*. Based on operator incompatibility for clusters of operators $a=(a_1,...,a_n)$ and $b=(b_1,...,b_m)$. Operators that can be executed in parallel must not be in the same cluster.

  $\text{inc}(a_i,b) = \max(\text{inc}(a_i,b_1),\text{inc}(a_i,b_2),...,\text{inc}(a_i,b_m))$

Clustering strategies (cont.)

- *Operator clustering*. Based on operator compatibility, the goal is to reduce area by clustering operators which can share hardware.

$$\text{comp}(a_i,b_j) = \begin{cases} 1 & \text{if } a_i,b_j \text{ have the same type, but are not} \\ & \text{scheduled into the same control step} \\ 0 & \text{otherwise} \end{cases}$$

- *Inter-procedural clustering*. Minimizes the transfer of control between clusters contained in different blocks. The probability that an activation of cluster *b* in block *B* is due to a call of *B* from *a*:

  P(b|a) = NumberOfCalls(B,a) /
  (TotalNumberOfCalls(B) x TotalNumberOfClusters(B))

Transformations

- A large number of hardware functions may fit the desired functionality
  - This is referred to as *design space*
  - Transformations aid to explore design space
- Transformations are divided:
  - Optimizing transformations (reducing the control graph)
  - Behavioral transformations (generating parallel/pipelined processes)

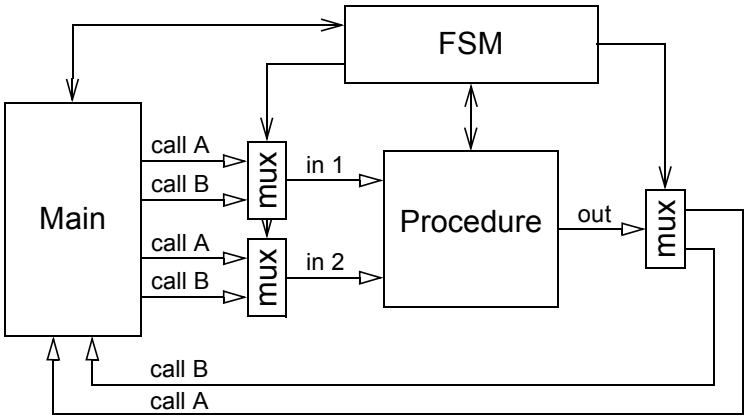
Optimizing Transformations

- Procedure In-line expansion
- Loop unrolling
- SELECT transformations
  - Combining IF- and CASE blocks
  - Motion of operators into or out of SELECT block

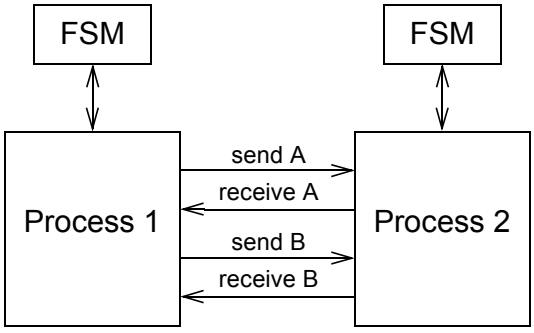
Behavioral Transformations

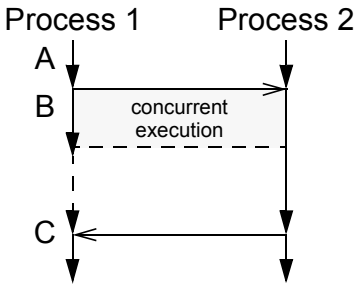
- Processes and communication
  - Parallel execution of two or more processes is called *concurrent execution*
  - Data transfer between processes is *inter-process communication*
  - *Synchronization* induces that inter-process communication has to occur at mutually agreed time slots
- Communication can be performed
  - by shared data (semaphores, monitors etc.)
  - by message passing (direct naming, mailboxes, etc.)

Procedure call implementation



Concurrent processes implementation





Pipelining

- *Functional decomposition* into stages (fetch, read, execute, write,...)
- *Divide function*, e.g., multiplier, signal processor, into more balanced stages
- Functional pipelining
  - The method to achieve functional pipelining from an algorithmic description is called *loop winding*
  - In loop winding, the data flow of a single process is partitioned into functional stages
  - The data flow edges are changed so that data that have to be sent to the following partition can only be transferred via the loop end of sender and the loop begin of the receiver

Synthesis from the system level description

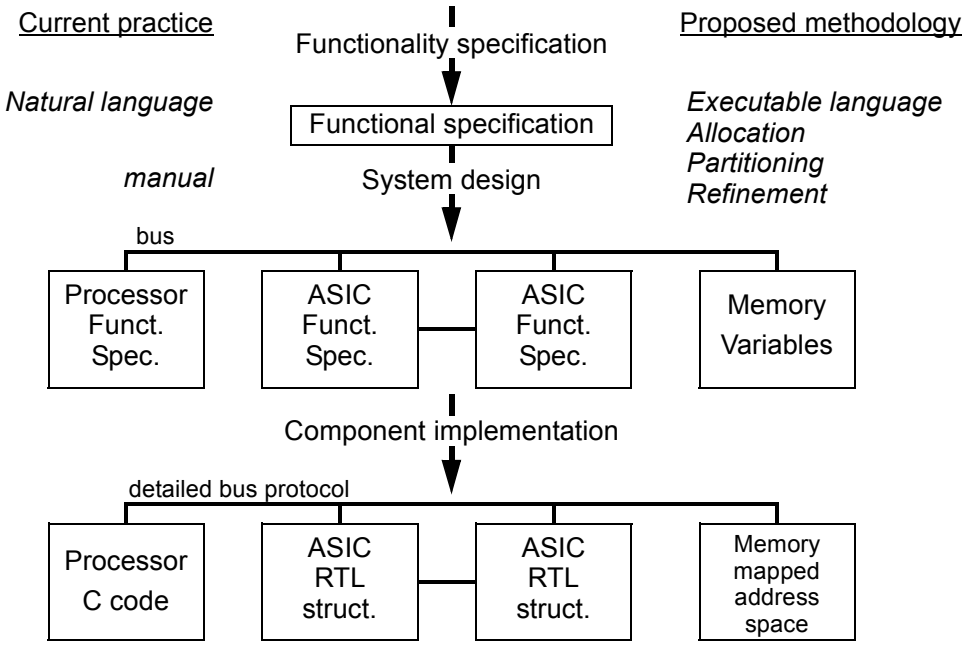
- Three different approaches to describe control and concurrency at higher level of abstraction:
  - use of fork/join constructs
  - use of extended finite state machines
  - object-oriented techniques
- The most important task for synthesis is:
  - Mapping the logical communication structure onto a physical communication structure
    - single bus / point-to-point
    - communication processor / distributed protocol
    - etc.

System design methodology by D. Gajski

- ***Design process*** **- conceptualization to manufacturing**
- ***Design methodology*** **- sequence of design tasks and associated CAD tools**

- **A design methodology must clearly specify the following:**
  - **the syntax and semantics of the input and output descriptions**
  - **the set of techniques for transforming input onto output descriptions**
  - **the set of components to be used in the design implementation**
  - **the definition and ranges of design constraints**
  - **the mechanism for selecting components and architectural styles**
  - **design exploration strategies (tasks, parameters, order)**

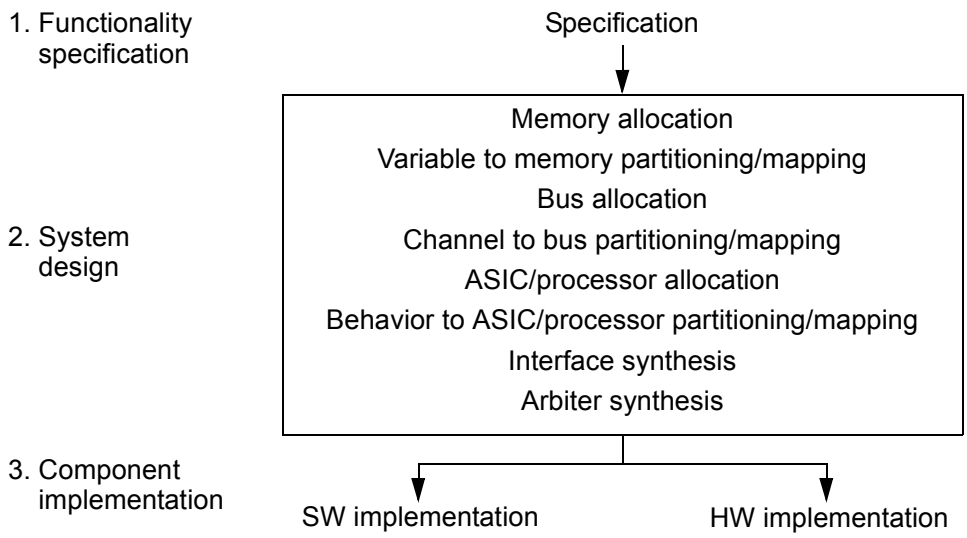
System design methodology



System design tasks

Functional objects	Allocation	Partitioning	Refinement
Variables	Memories	Variables to memories	Address assignment
Behaviors	Processors	Behaviors to processors	Interfacing
Channels	Buses	Channels to buses	Arbitration / protocols

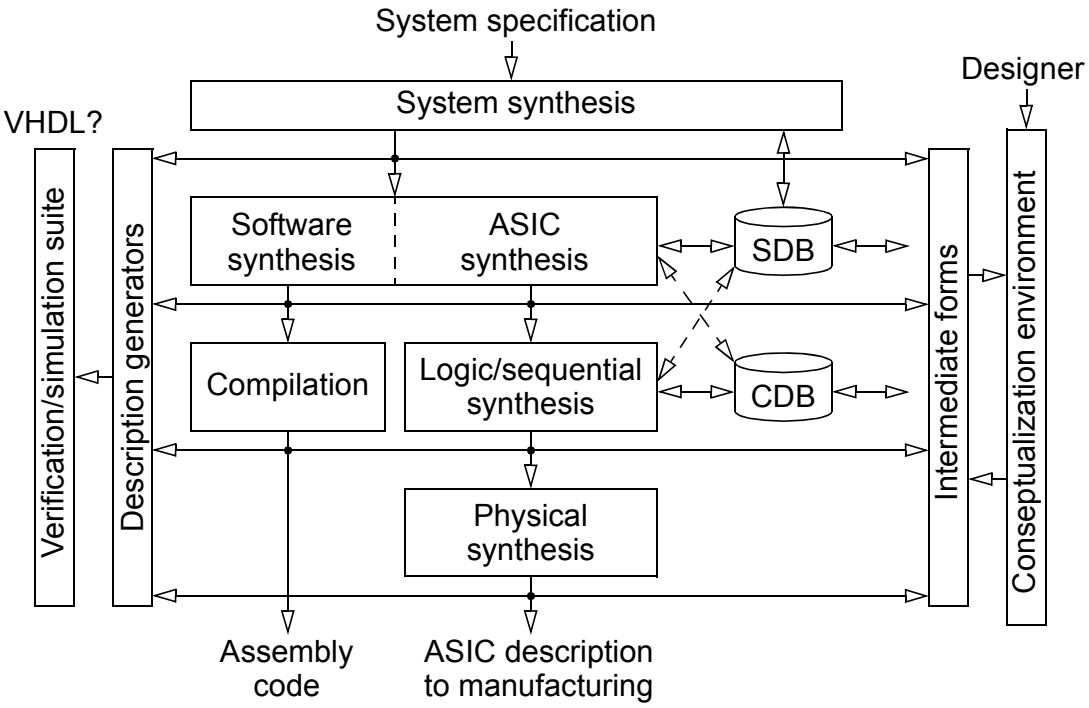
System design tasks ordering



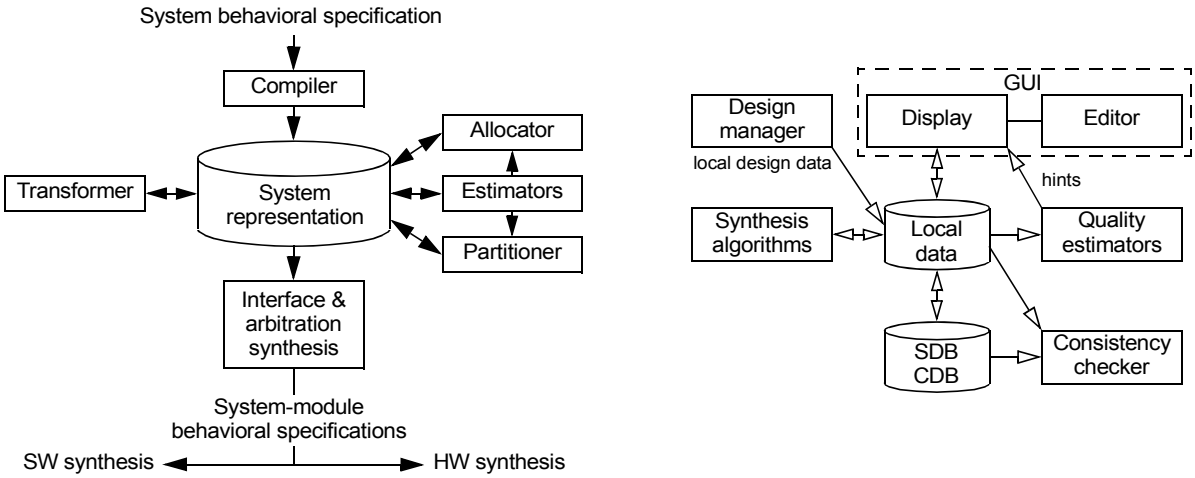
System design criteria

- **Completeness**
  - **All levels covered**
- **Extensibility**
  - **Adding new algorithms**
- **Controllability**
  - **Control tool types and order of execution**
- **Interactivity**
  - **Designer interacts with tools**
- **Upgradability**
  - **Evolutionary upgrade**

Generic synthesis system



System synthesis environments



Software synthesis

- **An executable specification possesses special features not found in traditional programming languages, like definition of concurrent tasks**
- **Multiprogramming issues:**
  - **No task "starvation"**
  - **Busy/waiting time minimization**
  - **Ensuring timing constraints of tasks**
- **Solution - tasks scheduling**

Software Development

- **System analysis**
- **RTOS/application selection**
- **Partitioning**
- **Interface definition**
- **Module development**
- **Software integration**
- **ROM/software distribution**
- **RTOS and application software**

Development Phases

- Software's architecture
  - layering, dependencies of functions, interfaces of functions
  - to be stabilized and frozen as soon as possible
- Development of new functions on the host
- Debugging (together with reused SW)
- Cross-compilation for the target processor
- Downloading to the BSP
- Integration with other software VC components

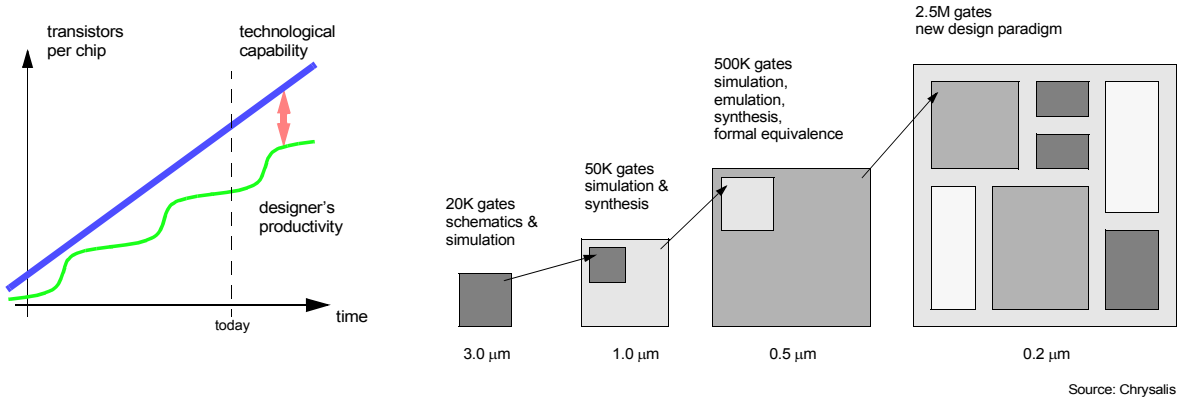
- Prototyping, debugging, and optimizing the whole system

- RTOS tuning – stack sizes, scheduling parameters, number of tasks, timeslots, etc.
- Device drivers – optimization, integration, debugging
- Test bench development – host level, on BSP, and final system integration
- Code mostly in C – critical parts in assembly & optimizing / redesigning memory layout
- Instruction set simulator (ISS) – cycle-accurate or at least cycle-approximate

Architecture of Embedded Software

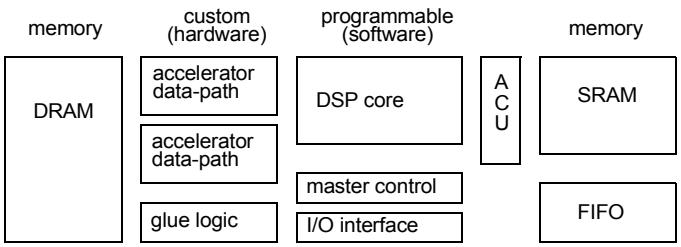
- Hardware
- Device drivers
- RTOS layer
  - kernel services, communication stack protocol, display services, file manger, alarm services, event manager, data I/O, etc.
  - Application Program Interface (API)
- Application layer
  - host applications
  - MMI/GUI, message manager, error handling, task controller, memory allocation, state machine, etc.
  - diagnostics

Bridging the Design Gap



Source: Chrysalis

Future – Platform Based Synthesis



Source: IMEC

heterogeneous system-on-a-chip

memory hierarchies

Source: IMEC