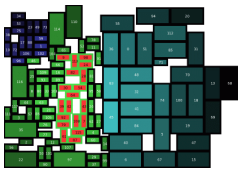
Embedded System Design and Synthesis

Robert Dick

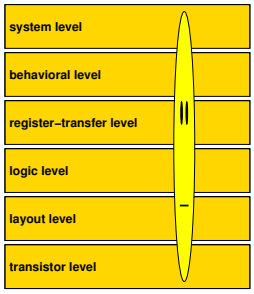
http://ziyang.eecs.northwestern.edu/~dickrp/esds-two-week
Department of Electrical Engineering and Computer Science
Northwestern University

Office at Tsinghua University: 9–310 East Main Building





Short and fat vs. tall and thin



Available resources – System-level

- General-purpose SW processors
- Digital signal processors (DSPs)
- Application-specific integrated circuits
- Dynamically reconfigurable hardware
  - E.g., field-programmable gate arrays (FPGAs)
- Busses
- Wireless communication channels
- Wires

Available resources – High-level

- Simple arithmetic/logic units
- Multiplexers
- Registers
- Wires

Specification language requirements

- Describe hardware (HW) and software (SW) requirements
- Specify constraints on design
- Indicate system-level building blocks
- To allow flexibility in synthesis, must be abstract
  - Differentiate HW from SW only when necessary
  - Concentrate on requirements, not implementation
  - Make few assumptions about platform

Software oriented design representations

- ANSI-C
- SystemC
- Other SW language-based

ANSI-C

Advantages

- Huge code base
- Many experienced programmers
- Efficient means of SW implementation
- Good compilers for many SW processors

Disadvantages

- Little implementation flexibility
  - Strongly SW oriented
  - Makes many assumptions about platform
- Poor support for fine-scale HW synchronization

SystemC

Advantages

- Support from big players
  - Synopsys, Cadence, ARM, Red Hat, Ericsson, Fujitsu, Infineon Technologies AG, Sony Corp., STMicroelectronics, and Texas Instruments
- Familiar for SW engineers

Disadvantages

- Extension of SW language
  - Not designed for HW from the start
- Compiler available for limited number of SW processors
  - New

Other SW language-based

- Numerous competitors
- Numerous languages
  - ANSI-C, C++, and Java are most popular starting points
- In the end, few can survive
- SystemC has broad support

Hardware oriented design representations

- VHDL
- Verilog
- Esterel

VHDL

Advantages

- Supports abstract data types
- System-level modeling supported
- Better support for test harness design

Disadvantages

- Requires extensions to easily operate at the gate-level
- Difficult to learn
- Slow to code

Verilog

Advantages

- Easy to learn
- Easy for small designs

Disadvantages

- Not designed to handle large designs
- Not designed for system-level

Verilog vs. VHDL

- March 1995, Synopsys Users Group meeting
- Create a gate netlist for the fastest fully synchronous loadable 9-bit increment-by-3 decrement-by-5 up/down counter that generated even parity, carry and borrow
- 5 / 9 Verilog users completed
- 0 / 5 VHDL users competed

Does this mean that Verilog is better?
Maybe, but maybe it only means that Verilog is easier to use for simple designs.

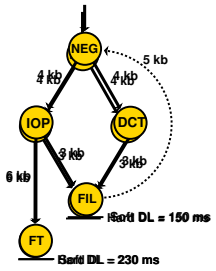
Esterel

- Easily allows synchronization among parallel tasks
- Works above RTL
  - Doesn't require explicit enumeration of all states and transitions
- Recently extended for specifying datapaths and flexible clocking schemes
- Amenable to theorem proving
- Translation to RTL or C possible
- Commercialized by Esterel Technologies

Graph based design representations

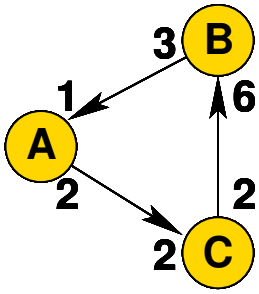
- Dataflow graph (DFG)
- Synchronous dataflow graph (SDFG)
- Control flow graph (CFG)
- Control dataflow graph (CDFG)
- Finite state machine (FSM)
- Petri net
- Periodic vs. aperiodic
- Real-time vs. best effort
- Discrete vs. continuous timing
- Example from research

Dataflow graph (DFG)

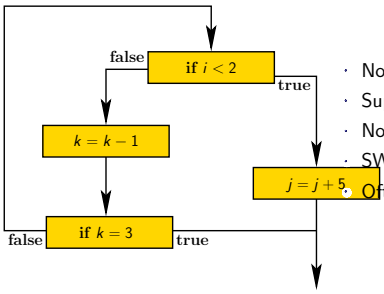


- Nodes are tasks
- Edges are data dependencies
- Edges have communication quantities
- Used for digital signal processing (DSP)
- Often acyclic when real-time
- Can be cyclic when best-effort

Synchronous dataflow graph (SDFG)

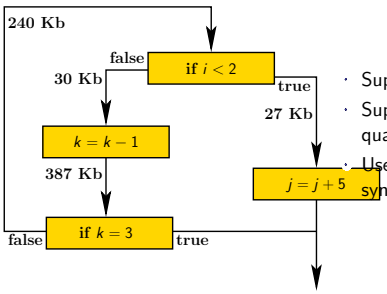


Control flow graph (CFG)



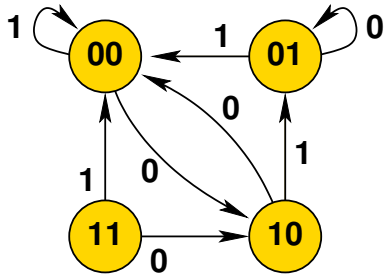
- Nodes are tasks
- Supports conditionals, loops
- No communication quantities
- SW background
- Often cyclic

Control dataflow graph (CDFG)



- Supports conditionals, loops
- Supports communication quantities
- Used by some high-level synthesis algorithms

Finite state machine (FSM)



Finite state machine (FSM)

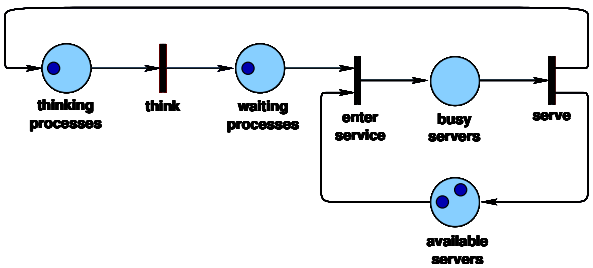
	input	
	0	1
00	10	00
01	01	00
10	00	01
11	10	00
current	next	

- Normally used at lower levels
- Difficult to represent independent behavior
  - State explosion
- No built-in representation for data flow
  - Extensions have been proposed
- Extensions represent SW, e.g., co-design finite state machines (CFSMs)

Petri net

- Graph composed of places, transitions, and arcs
- Tokens are produced and consumed
- Useful model for asynchronous and stochastic processes
- Places can have priorities
- Not well-suited for representing dataflow systems
- Timing analysis quite difficult
- Large flat graphs difficult to understand

Petri net

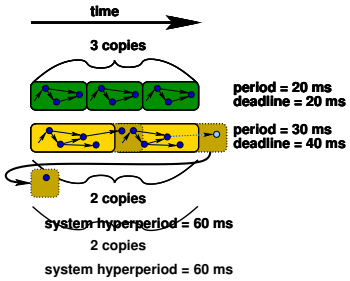


M/D/3/2: Markov arrival, deterministic service delay,
From A. Zimmermann's token game demonstration.

Periodic graphs

- Some system specifications contain periodic graphs
- Can guarantee scheduling validity by scheduling to the least common multiple of periods
- Can also meet aperiodic specifications, however, resources will sometimes be idle

Periodic graphs



Aperiodic graphs

- No precise periods imposed on task execution
- Useful for representing reactive systems
- Difficult to guarantee hard deadlines in such systems
  - Possible if minimum inter-arrival time known

Periodic vs. aperiodic

Periodic applications

- Power electronics
- Transportation applications
  - Engine controllers
  - Brake controllers
- Many multimedia applications
  - Video frame rate
  - Audio sample rate
- Many digital signal processing (DSP) applications

However, devices which react to unpredictable external stimuli have aperiodic behavior
Many applications contain periodic and aperiodic components

Aperiodic to periodic

Can design periodic specifications that meet requirements posed by aperiodic specifications

- Some resources will be wasted

Example:

- At most one aperiodic task can arrive every 50 ms
- It must complete execution within 100 ms of its arrival time

Aperiodic to periodic

- Can easily build a periodic representation with a deadline and period of 5 ms
  - Problem, requires a 50 ms execution time when 100 ms should be sufficient
- Can use overlapping graphs to allow an increase in execution time
  - Parallelism required

The main problem with representing aperiodic problems with periodic representations is that the tradeoff between deadline and period must be made at the time of synthesis

Real-time vs. best effort

- Why make decisions about system implementation statically?
  - Allows easy timing analysis, hard real-time guarantees
- If a system doesn't have hard real-time deadlines, resources can be more efficiently used by making late, dynamic decisions
- Can combine real-time and best-effort portions within the same specification
  - Reserve time slots
  - Take advantage of slack when tasks complete sooner than their worst-case finish times

Discrete vs. continuous timing

System-level: continuous

- Operations are not small integer multiples of the clock cycle

High-level: discrete

- Operations are small integer multiples of the clock cycle

Implications:

- System-level scheduling is more complicated...
- ...however, high-level also very difficult.

Processing resource description

- Often table-based
- Price, area
- For each task
  - Execution time
  - Power consumption
  - Preemption cost
  - etc.
- etc.

Similar characterization for communication resources
Wise to use process-based

Communication resource description

- Can use bus-bridge based models for distributed systems
- Wireless models
- etc.
- However, in the future, it will become increasingly important to base SOC communication model on process parameters

Example from research

Data-flow graphs

- Multirate
- Hard real-time
- Some aperiodic hard real-time tasks
- May have best-effort tasks

System-level representations summary

- No single representation has been decided upon
- Software-based representations becoming more popular
- System-level representations will become more important
- This is still an active area of research

Notes on clustering and partitioning

- Interdependence with architecture
- Heterogeneity's impact on partitioning
- Applications to grid computing
- Dynamic partitioning

Interesting future direction

Open problem

- Can specification be so simple for some embedded application domains that application experts who are not computer engineers easily do it?
- What HCI, compiler, and synthesis support is required?

Clustering and system-level pipelining references

- Steven Edwards, Luciano Lavagno, Edward A. Lee, and Alberto Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Proc. IEEE*, (3):366–390, March 1997
  - Recent technical report from embedded systems languages expert
- Wayne Wolf. Embedded computing systems and hardware/software co-design. In Wai-Kai Chen, editor, *The VLSI Handbook*. CRC Press, 2006

Assignment: Write a short paragraph describing the most important points in both of these articles.

Reading assignment

Robert P. Dick. *Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems*. PhD thesis, Dept. of Electrical Engineering, Princeton University, July 2002: Chapter 4

- Definitions and introduction to stochastic optimization

Decide topics and groups for project

Determine counts for each, add additional topics

- Models and languages
- Formal methods for designing reliable embedded systems
- Reliability optimization
- Heterogeneous multiprocessor synthesis
- Real-time systems
- Scheduling

Decide topics and groups for project

Determine counts for each, add additional topics

- Compilation techniques for embedded systems
- Embedded operating systems
- Low-power and power-aware design
- Low-power and power-aware design (continued)
- Novel fabrication techniques for compact and low-power embedded systems
- Emerging applications: sensor networks
- Hardware and software data compression for use in embedded systems

Next class

- Overview of optimization techniques
- Example solution to heterogeneous MPSoC synthesis problem
- Survey of formal methods for designing reliable embedded systems