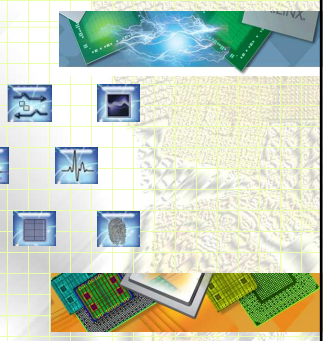
Embedded Systems

Register Transfer-level Design: Languages and Design Tools

George Kornaros

Embedded Technology

- Embedded Processing
- Connectivity
- DSP
- Memory Interfacing
- Power Efficiency
- High Speed Serial
- Analog Mixed Signal
- Design Security
- Stacked Silicon Interconnect



Web Resources

- P.J. Ashenden, The VHDL CookBook
  http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf
- http://digitaldesign.ashenden.com.au/vhdl/vhdl-tutorial.html

- Xilinx ISE Software/WebPack Installation is free
- GtkWave - timing viewer http://gtkwave.sourceforge.net/
- GHDL – simulator: http://ghdl.free.fr/
  GHDL is an open-source simulator for the VHDL language.

All times researchers' desire: C to HDL converters

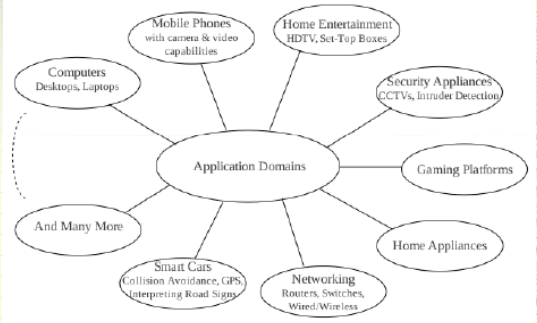
- http://en.wikipedia.org/wiki/C_to_HDL (25 entries)

Introduction – Embedded Systems

- An **embedded** system is a computer system designed to do one or a few dedicated and/or specific functions often with real-time computing constraints. It is **embedded** as part of a complete device often including hardware and mechanical parts.
- Embedded systems are application specific
  - ✓ Optimize to reduce size
  - ✓ Optimize to reduce cost
  - ✓ Optimize to increase performance
  - ✓ Optimize to increase reliability

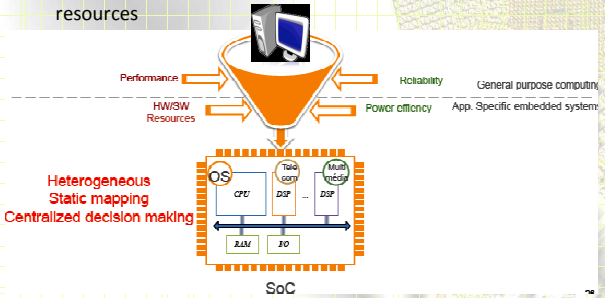


Embedded Systems



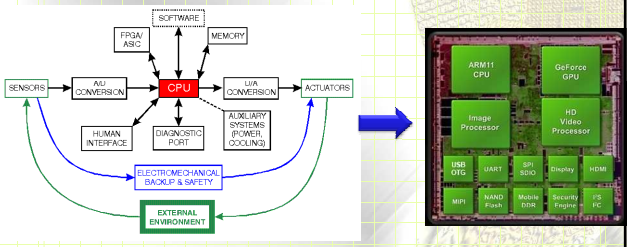
Embedded Systems

- Typical MPSoC : a platform with CPUs, GPUs, DSP, …
- Programmers need to make the best use of all the available resources



Complex Embedded Systems

- Systems on Chip



System-on-Chip (SoC)

- SoC refers to integrating all components of an electronic system into a single chip (what motivates such integration?):
  - ✓ Microprocessors, microcontrollers, DSP's
  - ✓ ASIC's, FPGA's
  - ✓ Memories, IO's, A/D and D/C converters
  - ✓ Analog, mixed-signal, RF blocks, voltage regulators
- Communication infrastructure: Bus based, Network-on-Chip (NoC).
- System-level design (automation) and simultaneous optimization of numerous design metrics is key challenge.
- A SoC is a complex embedded system.

Embedded Systems Characteristics

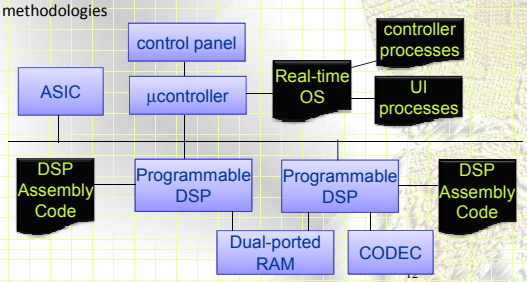
- Computational
- Interact (sense, manipulate, communicate) with the external world: sensors, actuators
- Reactive: at the speed of the environment
- Heterogeneity: hardware/software blocks, mixed architectures
- Networked: shared, adaptive, sensor networks (buildings, environmental monitoring), smart products, wearable computing
- Flexibility: can run/implement multiple applications sequentially or concurrently - concurrency
- Reprogrammability/reconfigurability: flexibility in upgrading, bug fixing, product differentiation, product customization
- Part of a larger system (system within system)
- Performance and constraints:
  - Timing (frequency, latency, throughput)
  - Cost, power, area, weight, temperature, reliability

Key Trends

- Increasing computation demands, increasing complexity
  - e.g. multimedia processing in set-top boxes, HDTV
- Increasingly networked
- Increasing need for flexibility
  - time-to-market under ever changing standards
- HW-SW co-design
- Higher integration: more blocks on the same chip
- IP reuse, platform based design, NoC vs. Bus
- Systems are designed and built as "systems of systems".
- Diversity in design methodologies, platform dependent, lack of standards → quality risks, customer confusion

Complexity and Heterogeneity

- Complexity - handled by working at higher levels of abstraction, hierarchy
- Heterogeneity - handled by sophisticated HW-SW co-design methodologies



Design Layers

- High-level Organization : Hardware Architectures
- System Building Blocks : Arithmetic units, controllers
- Circuit Elements : Memories, logic blocks
- Transistor-level circuit implementations
- Circuit primitives : Transistors, wires

FPGA Technology

- Field Programmable Gate Arrays (FPGAs) are programmable semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects.
- As opposed to Application Specific Integrated Circuits (ASICs) where the device is custom built for the particular design, FPGAs can be programmed to the desired application or functionality requirements.



FPGA Technology

- Modern FPGA devices
  - ✓Performance
  - ✓Area
  - ✓Application domain specific devices
  - ✓Reduction of development costs
  - ✓Time-to-market







Evolution

- As of 2009:



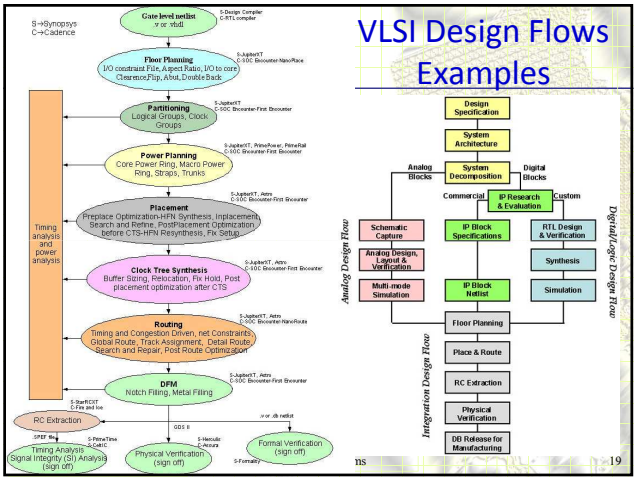
- State-of-the-art LX110T FPGA: ~1M logic gates.
  - Interfaces: Audio in/out, digital video, ethernet, on-board DRAM, PCIe, USB, ...

FPGA Design

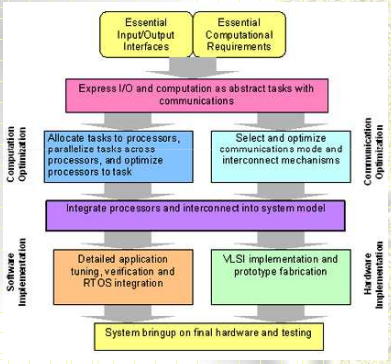
- Primary physical characteristics of a digital design
  - ✓Speed
  - ✓Area
  - ✓Power
- A set of design constraints must be met
- The design must be optimized

Methodologies – Design Flows

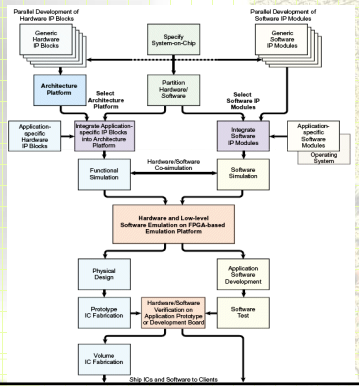
VLSI Design Flows Examples



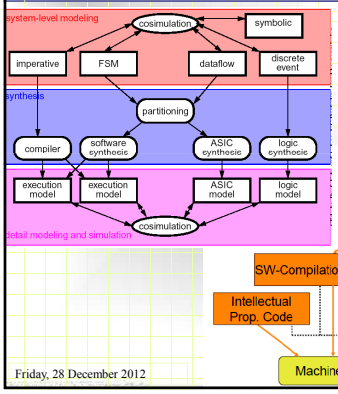
SoC Design Flow (Tensilica)



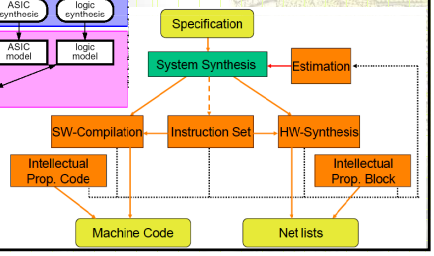
SoC Design Flow (ARM)



Tackling Heterogeneity



- Fixed Processor Architecture
- Application Specific HW Block
- Application-Specific Instruction Set Processor

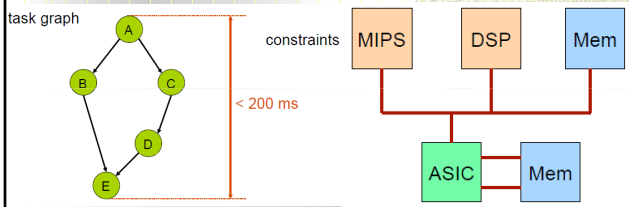


SoC Design Flow

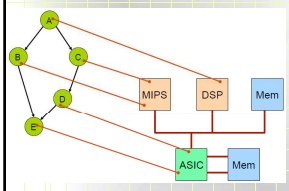
- Various design methodologies
  - ✓ Computation centric: Bus based – past and present
  - ✓ Communication centric: **Network-on-Chip based** – more recent
- Key design steps: Intertwined subtasks
  - ✓ Modeling/specification
  - ✓ Refining (or "functional/structural partitioning") : the function to be implemented into smaller, interacting pieces → hierarchy
  - ✓ HW/SW partitioning: elements in the refined model to either (1) HW units, or (2) SW running on custom hardware or a suitable programmable processor
  - ✓ Allocation (platform design): select components/blocks/IPs
  - ✓ Binding (mapping = implementation): assign functions to HW/SW components
  - ✓ Scheduling: compute the times at which the functions are executed. This is important when several modules in the partition share a single hardware unit.
  - ✓ Verification & debugging
- Crucial is the co-design and joint optimization of hardware and software
- For NoC based approach: network synthesis step is crucial

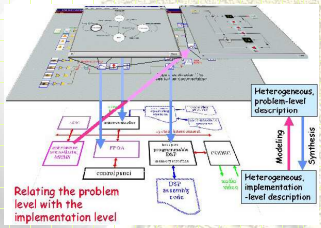
System Specification (Platform Design)

- Specification, Allocation, Binding, Scheduling



System Platform Design





System-level Design

- System-level design is a complex synthesis tasks
  - ✓ software synthesis and code generation
  - ✓ hardware synthesis
  - ✓ interface and communication synthesis
  - ✓ hardware/software partitioning and component selection
  - ✓ hardware/software scheduling
- Major Components:
  - ✓ Application specification
  - ✓ Design space exploration and system optimization
  - ✓ Estimation: Synthesis based on abstraction only makes sense if there are powerful estimation methods available:
    - o Estimate properties of the next layer(s) of abstraction.
    - o Design decisions are based on these estimated properties.

Hardware vs. Software Modules

- A significant part of the problem is deciding which parts should be in software on programmable processors, and which in specialized hardware.
- **Hardware** = functionality implemented via a custom architecture (datapath + FSM)
- **Software** = functionality implemented in software on a programmable processor
- Key differences:
  - Multiplexing
    - software modules multiplexed with others on a processor
    - hardware modules are typically mapped individually on dedicated hardware
  - Concurrency
    - processors usually have one "thread of control"
    - dedicated hardware often has concurrent datapaths

Challenges

- Increasing application complexity
  - ✓ large systems with legacy functions
  - ✓ mixture of event driven and data flow tasks
  - ✓ examples: multimedia, automotive, mobile communication
- Increasing target system complexity: heterogeneity → HW/SW co-design
  - ✓ mixture of different technologies, processor types, and design styles
  - ✓ large SoCs combining components from different sources, distributed system implementations
- Design chain integration:
  - ✓ Design Roles →Methodology → Tools
- Parallel architectures
- Embedded software and control
- Numerous constraints and design objectives
  - ✓ examples: cost, power consumption, timing constraints, dependability

Present & Future Design Methodologies

- HW-SW co-design
- Design automation (CAD) tools

Basic VHDL concepts

- **Very High Speed Integrated Circuit Hardware Description Language**
- It is designed to describe hardware at all levels of hierarchy
- VHDL is used like a programming language (write code, check for errors, simulate etc.)

Beware!

- The structure of Hardware is largely parallel
- Use of signals → Buses and cables on hardware, impose a latency

VHDL Components

- Components are the building blocks of VHDL
- Each component is made up of two main parts:
  - ✓ Entity: Port declaration for inputs a
  - ✓ Architecture: Structural or behavio

```
entity VHDL_example is
port(A : in std_logic;
B : out std_logic);
end entity VHDL_example;

architecture beh of VHDL_ex
begin
        B <= not A; ...
end beh;
```

Std_logic: signal type, size of a bit
- 'U': uninitialized. This signal hasn't been set yet.
- 'X': unknown. Impossible to determine this value/result.
- '0': logic 0
- '1': logic 1
- 'Z': High Impedance
- 'W': Weak signal, can't tell if it should be 0 or 1.
- 'L': Weak signal that should probably go to 0
- 'H': Weak signal that should probably go to 1
- '-': Don't care.

Concurrent VHDL

- The signal assignment happens "concurrently" (in parallel)

```
entity VHDL_example_2 is
port(
A, B : in std_logic_vector (2 downto 0);
 F, G : out std_logic_vector (2 downto 0)
);
end entity VHDL_example_2;

architecture beh of VHDL_example_2 is
begin
        F <= A AND B;
        G <= A XOR B;
end beh;
```

Std_logic_vector: a vector of std_logic bits
The most significant bit is bit (2) of the vector

Sequential VHDL

- The commands/assignments are executed in sequence BUT no time elapses in the sequential part of VHDL

```
architecture beh of VHDL_example_3 is
begin

process (a, b, ... )
        if a = "0110" then
        c <= a;
        else
        c <= (others => '0');
         end if;
 end process;

end beh;
```

Process: a block of sequential VHDL code

Sensitivity list: signals which cause process initiation every time their value changes

A formulation to set logic '0' to every bit of a vector irrespective of its size

Sequential VHDL

- Clocked processes are fundamental in VHDL design
  - ✓ flip-flops, registers, FSM etc.

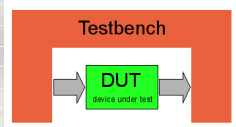
```
entity dff is
port( data_in : in std_logic;
clock : in std_logic;
data_out: out std_logic );
end dff;

architecture behv of dff is
begin
process(data_in, clock)
begin
   if (clock='1' and clock'event) then
         data_out <= data_in;
   end if;
end process;
end behv;
```

Data propagation is executed on the rising edge of the clock

Verification

- **Verification** is a process used to demonstrate the functional correctness of a design
- A **testbench** is code that is used to create a pre-determined input sequence to a design
  - ✓ Usually to observe the response
- It is a closed system for the Design Under Test (DUT)



Advanced techniques using VHDL

Improving Speed

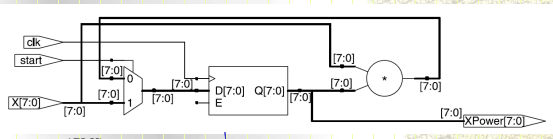
- How can speed be defined in a digital circuit?
- **Throughput :** The amount of data that is processed per clock cycle
- **Latency:** The time (number of clock cycles) between data input and processed data output
  - ✓Timing: Logic delays between sequential elements (the delay of the critical path)

Increase Throughput

- Increase the data rate by implementing a **pipelined design**
- Pipeline
  - ✓In principle it is a digital assembly line
  - ✓Data can begin processing before the prior data has finished



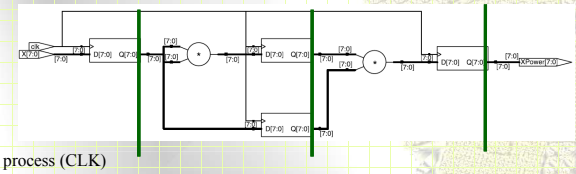
Example: *Calculate the third power of X*




```
process (CLK)
begin
if (CLK'event and CLK='1') then
  if (start = '1') then
        XPower <= X;
        ncount <= 2;
  elsif (ncount > 0) then
        ncount <= ncount - 1;
        XPower <= XPower * X;
  end if;
end if;
end process;
```

Throughput = 8/3, or 2.7 bits/clock
Latency = 3 clocks
Timing Delay= One multiplier delay in the critical path

Example: *Calculate the third power of X*



```
process (CLK)
begin
if (CLK'event and CLK='1') then
        X1 <= X;
        XPower1 <= X;
        X2 <= X1;
        XPower2 <= XPower1 * X1;
        XPower <= XPower2 * X2;
end if;
end process;
```

Throughput = 8/1, or 8 bits/clock
Latency = 3 clocks
Timing Delay= One multiplier delay in the critical path

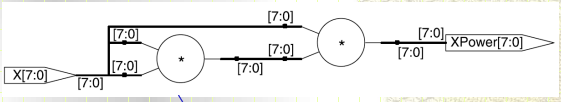
Increase Throughput

- Unrolling an iterative loop increases throughput
- The penalty for unrolling an iterative loop is **a proportional increase in area**

Decrease Latency

- A low-latency design passes the data from the input to the output as quickly as possible by minimizing the intermediate processing delays.
- Can be achieved by
  - ✓Parallelisms
  - ✓Removal of pipelining
  - ✓Logical shortcuts

Decrease Latency




```
process (X) begin
  X1 <= X;
  XPower1 <= X;
  X2 <= X1;
  XPower2 <= XPower1 * X1;
  XPower <= XPower2 * X2;
end process;
```

Throughput = 8 bits/clock (assuming one new input per clock)
Latency = Between one and two multiplier delays, 0 clocks
Timing = Two multiplier delays in the critical path

Decrease Latency

- Latency can be reduced by removing pipeline registers
- The penalty by removing pipeline registers is an **increase in combinatorial delay between registers**