EECS 222C: System-on-Chip Software Synthesis Lecture 1

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 1: Overview

- Course Overview
  - Administration and communication
  - Context and contents, objectives and outcomes
  - Literature
- Introduction to Embedded Systems
  - Design complexity challenge
  - Hardware/software co-design flow
- The Concept of a Model
  - Modeling and abstraction
  - System-on-Chip model features
  - Separation of concerns: The SpecC model
- Application Case Study
  - MP3 audio decoder
  - Assignment 1

Course Administration

- Course web pages at **`http://eee.uci.edu/13s/18416/`**
  - Instructor information
  - Course description
  - Course syllabus
  - Course objectives and outcomes
  - Course resources
  - Assignments
- Course communication
  - Message board
  - Email

Course Context

- EECS 222 A-C: Set of 3 courses on SoC Design

A. System-on-Chip Description and Modeling (EECS 222)
B. System-on-Chip Design and Exploration (EECS 225)
C. System-on-Chip Software Synthesis (EECS 226)
~~**D. System-on-Chip Hardware Synthesis**~~

- ~~**Course A is prerequisite for B, C, and D, or consent of instructor**~~

(effective 2013-14)

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
    Computational models for System-on-Chip (SoC). System-level specification and description languages and execution semantics. Concepts, requirements, examples. SoC modeling at different levels of abstraction (untimed, approximate time, cycle-accurate). Modeling of IP (IP wrappers), design constraints, test benches. Simulation semantics and algorithms. Co-simulation methodology.
  - **B. System-on-Chip Design and Exploration**
  - **C. System-on-Chip Software Synthesis**
  - **D. System-on-Chip Hardware Synthesis**

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
  - **B. System-on-Chip Design and Exploration**
    System-on-Chip design flow and methodology. Design space exploration. Co-design of hardware and software, hardware/software partitioning. System-on-Chip architecture exploration and synthesis. On-chip network and communication design and synthesis. On-chip software/hardware interface generation.
  - **C. System-on-Chip Software Synthesis**
  - **D. System-on-Chip Hardware Synthesis**

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
  - **B. System-on-Chip Design and Exploration**
  - **C. System-on-Chip Software Synthesis**
    System-on-Chip software concepts, requirements, examples, for engineering applications such as automotive and communication. Software synthesis methodology. Algorithmic specification, design constraints. Applications using embedded operating systems. Static, dynamic scheduling. Input/output, interrupt handling. Code generation, retargetable compilation. Instruction set simulation. Debugging and prototyping.
  - **D. System-on-Chip Hardware Synthesis**

Course Context

- EECS 222: Set of 4 courses on SoC Design
  - **A. System-on-Chip Description and Modeling**
  - **B. System-on-Chip Design and Exploration**
  - **C. System-on-Chip Software Synthesis**
  - **D. System-on-Chip Hardware Synthesis**
    Hardware IP specification. Real-time constraints. Cycle-accurate languages and modeling. Target architectures, data path and control unit. Design tasks and design methodology. Behavioral synthesis. Resource allocation, operation scheduling, binding of operations and variables to functional units, storage units and busses. Communication protocol and interface synthesis. Arbiter, bridge, Transducer, Controller design and synthesis. Net list generation.

Course Contents

- EECS 222C: SoC Software Synthesis
  - System-on-Chip software
    - concepts, requirements, and examples,
    - for engineering applications such as automotive and communication.
  - Software synthesis methodology.
  - Algorithmic specification and design constraints.
  - Applications using embedded operating systems.
  - Static, dynamic, real-time scheduling.
  - Input/output, interrupt handling.
  - Code generation, retargetable compilation.
  - Instruction set simulation.
  - Debugging and prototyping.

Course Goals

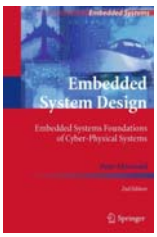
- Objectives
  - To learn embedded software concepts in System-on-Chip designs
  - To be able to design, develop and debug software in SoC designs
  - To understand software code generation for SoC
- Outcomes
  - Students understand
    - the special requirements of software for SoC.
    - the process of code generation and integration for SoC.
  - Students are able to
    - develop application SW, middleware, and/or drivers for SoC.
    - implement, test and debug a software application for a SoC.

Course Topics

1 – Embedded software concepts, requirements
2 – SoC software specification, modeling
3 – Embedded software design flow
4 – Real-Time Operating Systems (RTOS)
5 – Real-time requirements, real-time scheduling
6 – Software synthesis, code generation
7 – Hardware-dependent Software (HdS)
8 – Target processors
9 – (Cross-) compilation, execution, debugging
10 – Instruction-set simulation

Course Literature

- Primary Textbooks
  - P. Marwedel:
    "Embedded System Design",
    Embedded Systems Foundations
    of Cyber-Physical Systems,
    2nd edition, Springer, 2011.
    eBook: ISBN 978-94-007-0257-8
    Softcover: ISBN-13 978-94-007-0256-1



  - A. Gerstlauer, R. Doemer, J. Peng, D. Gajski:
    "System Design: A Practical Guide with SpecC",
    Kluwer Academic Publishers, Boston, June 2001.
    ISBN 0-7923-7387-1
    Hardcover: ISBN 978-0-7923-7387-2
    Softcover: ISBN 978-1-4613-5575-5

Embedded Computer Systems

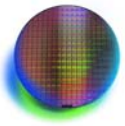
- Computers are ubiquitous, omnipresent…







- *System-on-Chip (SoC) Design*: Design of complex embedded systems on a single chip

Embedded Systems

- System embedded into another system
  - Constraints from external input (often real-time)
  - Application specific (not general purpose)
- Omnipresent in our environment
  - In many application domains
  - In 2005 [Source Netrino]
    - Only 2% of all processors in workstations
    - Remaining 8.8 billion in embedded systems
  - Pervasive


Source: P. Chou, UCI


Source: Edumicator


Source: Miele


Source: Philips


Source: www.trouper.com


Source: www.medicacorp.com/

Embedded System Design

- Design challenges
  - Often mobile
    - Battery powered (low power)
  - Often highly reliable
    - Extreme environment (e.g. temperature)
  - High performance constraints
    - Often real-time requirements
  - High complexity
    - E.g. Mercedes Benz E-class
      - 55 electronic control units
      - 5 communication busses
  - Tightly coupled
    - Software
    - Hardware
  - Rapid development for low price…

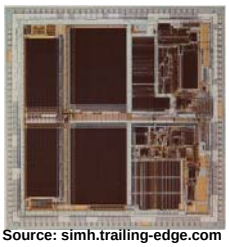
Source: Motorola Inc

Source: Daimler

EECS222C: SoC Software Synthesis, Lecture 1 (c) 2013 R. Doemer 15

Embedded System Design

- Design Advantages
  - *Application known at design time*
  - *Environment known at design time*
  - Allows for customized / optimized solution
    - Improved performance
    - More functionality
    - At lower power



Source: simh.trailing-edge.com

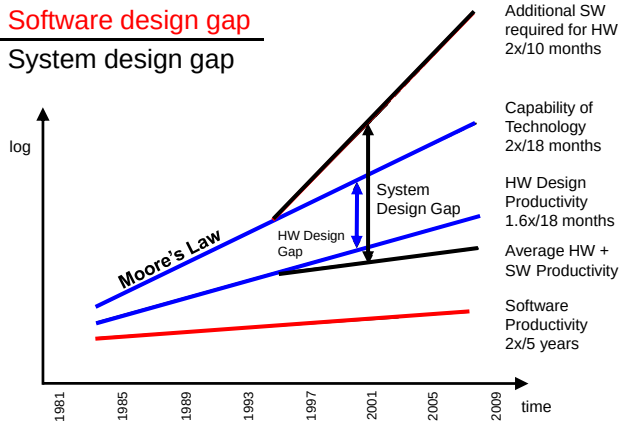
- Custom Platform, SW and HW components
  - Multi-Processor System-on-Chip (MPSoC),
    - Complete embedded system integrated on a chip
  - General-purpose and application-specific processors
  - Application Specific Integrated Circuit (ASIC)
  - Field Programmable Gate Array (FPGA)
  - Circuit board with off-the-shelf-components

Source: Xilinx

EECS222C: SoC Software Synthesis, Lecture 1 (c) 2013 R. Doemer 16

Design Complexity Challenge

- Productivity Gap
  - Hardware design gap
  - \+ Software design gap
  - = System design gap



(source: *"Hardware-dependent Software"*, Ecker et al., 2009)

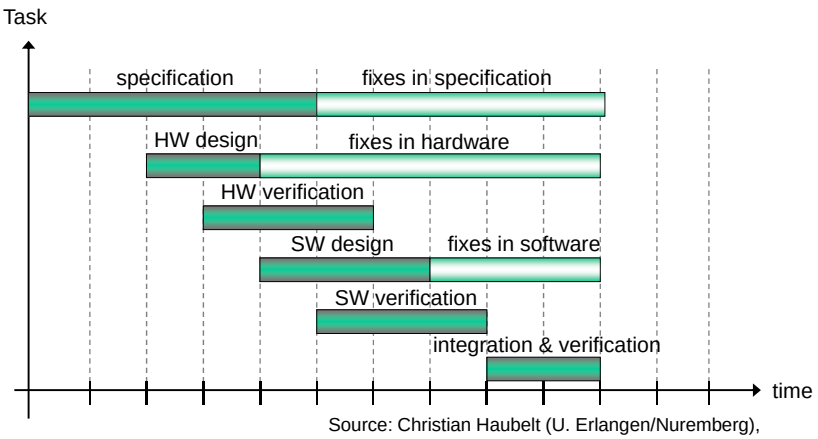
Design Complexity Challenge

- Productivity Gaps
  - Hardware productivity gap
    - Capacities in chip size outpace capabilities in chip design
    - Moore's law: chip capacity doubles every 18 months
    - HW design productivity estimated at 1.6x over 18 months
  - Software productivity gap
    - Growth of SW productivity estimated at 2x every 5 years
    - Needs in embedded SW estimated at 2x over 10 months
  - System productivity gap
    - HW gap + SW gap

Hardware/Software Codesign

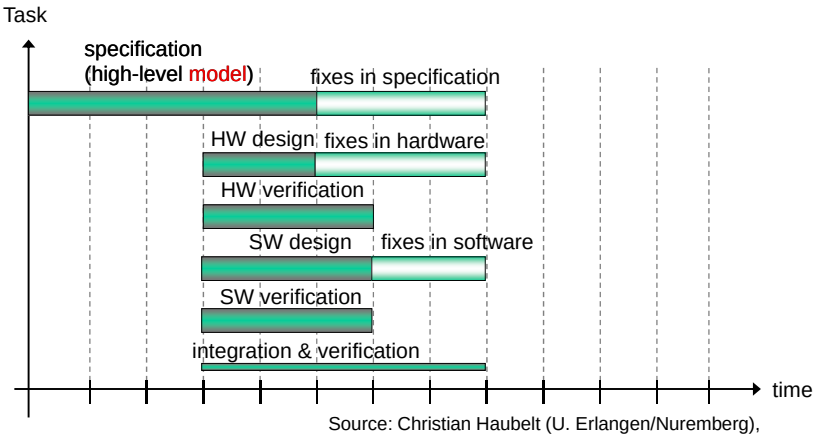
- Traditionally, software development follows hardware
- New: Unified, *concurrent* Design of
  - Hardware and
  - Software
- ➢ Improving Time to Market
  - Faster delivery of new products
  - Higher probability of on time delivery
- ➢ Using a single specification model (System Model)
  - New specification model
  - New specification language
  - ➢ Tight integration of
    - software development
    - hardware development

Traditional Design Flow



Source: Christian Haubelt (U. Erlangen/Nuremberg), Andreas Gerstlauer (UT Austin)

Co-Design Flow (ESL Design)



Source: Christian Haubelt (U. Erlangen/Nuremberg), Andreas Gerstlauer (UT Austin)

➢ Aka. *Model-based Design*!

The Concept of a Model

- What is a Model?
  - Definition: A Model is an Abstraction of Reality.
- Examples of Models
  - Toy car
    - Abstract model of a real car
      - Smaller scale
      - Simpler, many details left out (no motor, no lights, …)
      - Less expensive, less dangerous
      - Less, but sufficiently functional
  - Architectural blueprint of a house
    - 2-dimensional model of a real building (3-dimensional)
      - Smaller, but to scale (floor plan, room sizes, window placement..)
      - Simpler, many details left out (no bricks, just paper)
      - Some features over-emphasized (e.g. layout of pipes, cables)
      - Less expensive, easy to estimate and modify

The Concept of a Model

- What is a Model?
  - Definition: A Model is an Abstraction of Reality.
- What is Abstraction?
  - Part of model building
  - Simplification or omission of details
    - Some aspects of reality are simplified or omitted
  - Approach to reduce and factor out details
    - so that one can focus on a few features at a time
- What is Modeling?
  - Model building
  - To make or construct a model
- What is Specifying?
  - Creating the initial model in a design flow

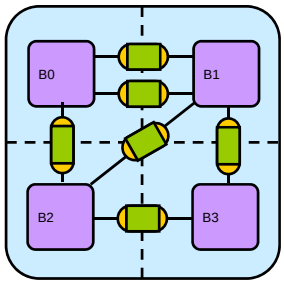
Embedded System Models

- Modeling an Embedded System
  - Decide what feature/property/characteristic
    - is needed (and to what degree)
    - is not needed (can be abstracted away)
- Typical Features in System-on-Chip Models
  - Functionality: important, most often needed (to a varying degree of accuracy)
  - Executability: important, often needed
  - Structure: increasingly needed in later design phases
  - Communication: needed to a varying degree of accuracy
  - Timing: needed to a varying degree of accuracy
  - Power consumption: sometimes needed, sometimes not
  - Temperature: usually not needed

Co-Design Models: Hardware and Software

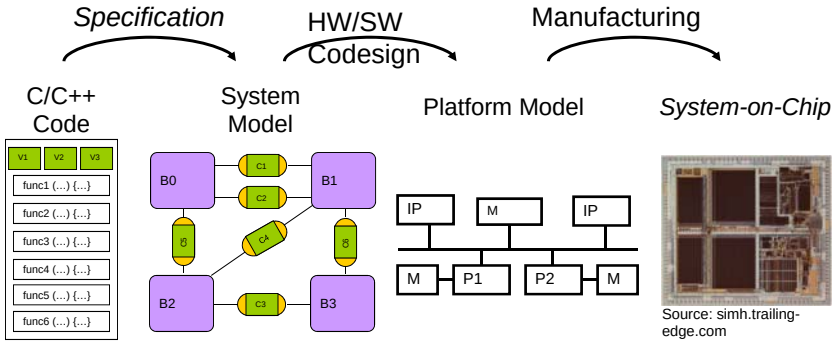
- System Level Modeling
  - Abstract description of a complete system
  - Software + Hardware
- Key Concepts in System Modeling
  - Explicit Structure
    - Block diagram structure
    - Connectivity through ports
  - Explicit Hierarchy
    - System composed of components
  - Explicit Concurrency
    - Potential for parallel execution
    - Potential for pipelined execution
  - Explicit Communication and Computation
    - Channels and Interfaces
    - Behaviors / Modules

System Model



System-on-Chip Co-Design Flow

- Specification and Modeling
- Codesign: concurrent HW design and SW development
- Ongoing Research: Automatic Model Transformations



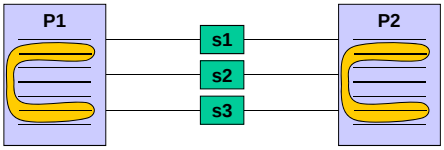
Source: simh.trailing-edge.com

Separation of Concerns

- Fundamental Principle in Modeling of Systems
- Clear *separation of concerns*
  - address separate issues independently
- System-Level Description Language (SLDL)
  - orthogonal concepts
  - orthogonal constructs
- System-level Modeling
  - Computation
    - encapsulated in modules / behaviors
  - Communication
    - encapsulated in channels

System-Level Model

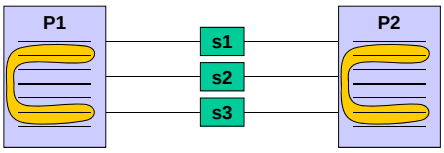
- Traditional model



  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

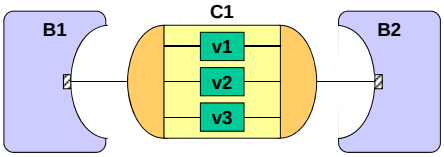
System-Level Model

- Traditional model



  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

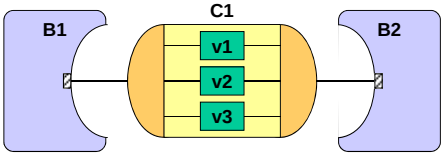
- SpecC model



  - Behaviors and channels
  - Separation of computation and communication
  - Plug-and-play!

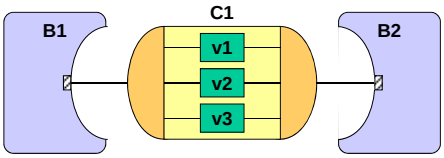
System-Level Model

- SpecC Model
  - Behaviors
    - Computation
  - Channels
    - Communication
  - ➢ System Modeling!

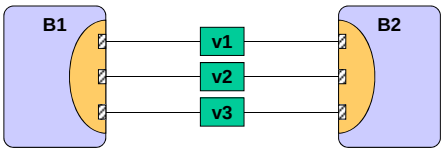


System-Level Model

- SpecC Model
  - Behaviors
    - Computation
  - Channels
    - Communication
  - ➢ System Modeling!



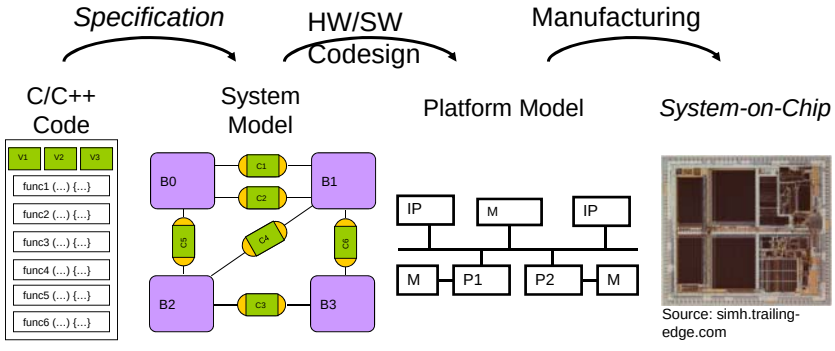
- Implementation through *Protocol Inlining*
  - Channel disappears
  - Communication is inlined into behaviors
  - Signals are exposed



➢ Model is converted to traditional model for implementation!

Application Case Study

- Taking an example application through the SoC Codesign flow
  - With the focus on software aspects
    - EECS222C: *SoC Software Synthesis*



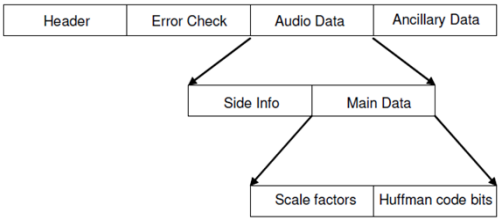
Application Case Study

- Project Application: MP3 Audio Decoder
  - Digital compression of audio data reduces
    - Communication bandwidth and
    - Storage requirements
  - MPEG 1 Layer 3 (aka. MP3) compression algorithm
    - most commonly used
    - uses a variety of clever tricks to compress digital music
      - by 90% or more!
    - performs lossy compression
    - applies perceptual science of psycho acoustic models
      - exact input signal does not need to be retained
      - human ear can only distinguish a certain amount of detail
      - sufficient if output signal sounds identical to the human ears

[Source: CECS-TR-05-04.pdf]

Application Case Study

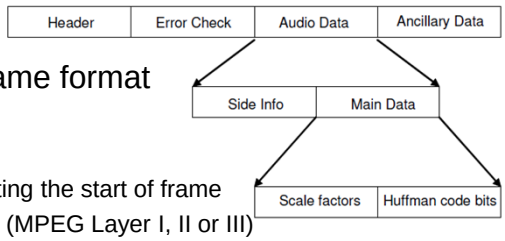
- Project Application: MP3 Audio Decoder
  - MP3 audio bit stream
    - organized in frames of bits
    - each frame contains 1152 encoded PCM samples
    - frame length depends on the bit rate (quality)
    - bit rate may vary in variable rate encoded streams
    - frame header contains information for the frame detection
  - MPEG 1 Layer 3 frame format



[Source: CECS-TR-05-04.pdf]

Application Case Study

- Project Application: MP3 Audio Decoder

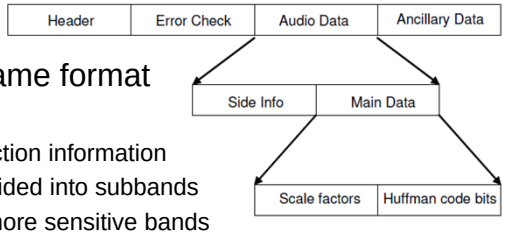


[Source: CECS-TR-05-04.pdf]

  - MPEG 1 Layer 3 frame format
    - Header
      - 4 bytes
      - Sync word indicating the start of frame
      - Layer information (MPEG Layer I, II or III)
      - Bitrate information
      - Sampling frequency
      - Mode information (mono or stereo)
    - Error Check
      - 16 bit parity check for optional error detection

Application Case Study

- Project Application: MP3 Audio Decoder

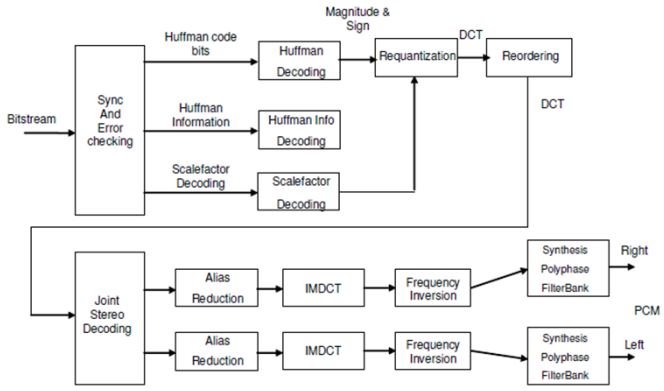


[Source: CECS-TR-05-04.pdf]

  - MPEG 1 Layer 3 frame format
    - Side Information
      - Scale factor selection information
        - » spectrum divided into subbands
        - » samples in more sensitive bands are scaled more than others
      - Global gain (to be applied to all samples)
      - Number of bits used to encode scalefactors
      - Huffman table selection (1 out of 32 Huffman tables)
    - Main data
      - Scale factors
      - Quantized values encoded using Huffman codes

Application Case Study

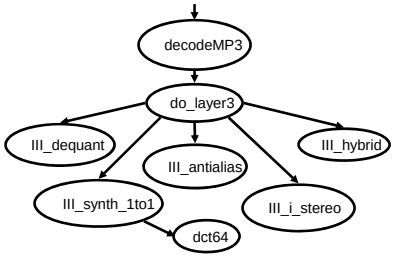
- Project Application: MP3 Audio Decoder
  - MP3 decoder block diagram



[Source: CECS-TR-05-04.pdf]

Application Case Study

- Project Application: MP3 Audio Decoder
  - MP3 decoder C reference code
    - Underbit Technologies Inc.
    - MAD: MPEG Audio Decoder
    - **`http://www.underbit.com/products/mad`**



Partial function hierarchy in MP3 reference code

[Source: P. Chandraiah]

Assignment 1

- Administration
  - Linux Servers
    - **`gamma.eecs.uci.edu`** (NSF client)
    - **`omicron.eecs.uci.edu`** (NSF client)
    - Intel Pentium based workstations
    - RedHat Linux (Fedora Core 12)
    - Access via secure shell protocol (**`ssh`**)
  - Accounts
    - User ID same as your UCI net ID
    - Password as discussed in class
  - SpecC Software (© by CECS, UCI)
    - SpecC Compiler and Simulator
    - System-on-Chip Environment (SCE)

Assignment 1

- Login on Server via SSH
  - Account infos will be emailed
- Install MP3 Decoder example
  - **`mkdir eecs222c`**
  - **`cd eecs222c`**
  - **`gtar xvzf /home/doemer/EECS222C/mad_C.tar.gz`**
  - **`cd mad_C`**
  - **`make clean`**
  - **`make`**
  - **`make test`**
- Become familiar with the application and its structure
  - Browse and read the source files
  - Draw a block diagram of the major functions

Assignment 1

- Analyze the given MP3 Decoder application
- Questions to study:
  - Example MP3 streams
    - Do they play?
    - Length in seconds?
    - Number of samples?
  - Application source code
    - How many source files?
    - How many lines of code?
    - How many functions?
  - What are the major functions?
    - How do they relate?
    - Function call graph?
  - What are the most critical functions?
    - Where is the most time spent?
  - What type of operations are performed?
    - Floating point?
    - Others?
  - Where is any potential for parallel execution?