The Embedded System Design Process – Specs, Execution and Documentation

Luis F. González P., PhD
Computer Science Department
ITESM-GDA

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions

Introduction

- Embedded computing is in many ways much more demanding than the sort of programs that we may write for PCs
- Functionality is important in both but embedded apps must meet many other constraints as well
  - User interface
  - Real time
  - Limited resources (memory)
  - Multirate
  - Manufacturing costs
  - Power consumption

Introduction

- Most embedded systems are designed by small teams on tight deadlines
- A self-fulfilling prophecy… the fact that systems can be built with microprocessors by small design teams invariably encourages management to assume that ALL microprocessor-based systems can be built by small teams…
- At present time, building a product using embedded software makes a lot of sense…

Introduction

- Hardware and software can be debugged somewhat independently
- Design revisions can be made much more quickly
- Hardware can be used to support different releases and/or families of products by modifying its functionality in terms of software

Challenges

- External constraints are one important source of difficulty in embedded system design
- Important decisions to make are
  - How much hardware do we need?
    - Too little may make the system to fail in meeting deadlines
    - Too much may make the system too expensive
  - How do we meet deadline?
    - Speeding up the hardware so that the program runs faster
    - Yes but… how about power consumption and cost
    - Moreover, increasing clock rate does not necessarily guarantee that execution time will increase accordingly

* External constraints are one important source of difficulty in embedded system design
* Important decisions to make are
  * How do we minimize power consumption?
    * Excessive power consumption can increase heat dissipation
    * One way to consume less power is to slow down the processor
    * Yes but… missed deadlines can occur
    * Goal: careful design to slow down noncritical parts of the system while still meeting performance goals

* Challenges

Challenges

* External constraints are one important source of difficulty in embedded system design
* Important decisions to make are
  * How do we design for upgradeability?
    * How can we design a system that will provide the required performance for software that hasn't been written yet?
* Does it really work?
  * Reliability
  * Waiting till a system is running to correct bugs may be too late, too expensive and too long

Additional Challenges

- Complex testing
  - An embedded system is part of a machine
  - In order for the embedded system to be tested properly, the real machine must be run
  - We cannot separate the testing of an embedded system from the machine in which it is embedded
- Limited observability and controllability
- Restricted development environments

Design Process

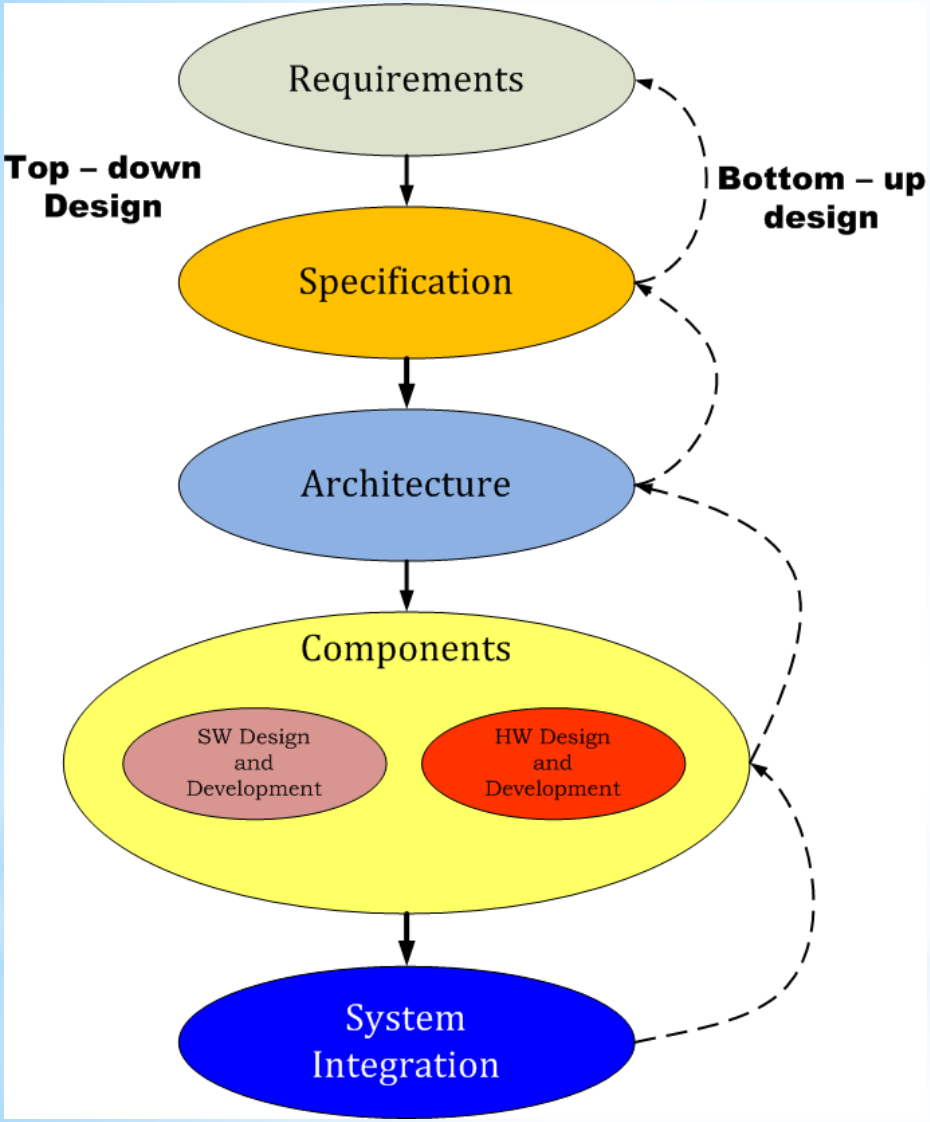
- Normally, embedded systems are designed with an ad-hoc approach that is heavily based on earlier experience with similar products
- Often, the design process requires several iterations to obtain convergence, because the system is not specified in a rigorous and unambiguous fashion
- And the levels of abstraction, details and design styles in various parts are likely to be different
- As the complexity of embedded systems scales up, the ad-hoc approach must be changed
- A design methodology must be adopted

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions

Design Methodology

- Importance of design methodology
  - It allows to keep a scorecard on the design to ensure that we have done everything we need to do
  - It facilitates the communication between members of the design team
    - Team members understand more easily what they are supposed to do
    - They know what they should receive from other team members and when
    - They know what they should hand off when they complete their tasks



Design process

* Major steps
  * Requirements: wish list
  * Specs: a more detailed description of what we want. It states how the system behaves, not how it is built
  * Architecture definition: details of the system's internals
  * Components: HW/SW codesign
  * System integration

Design process

- At every step in the design process
  - We must analyze the design to determine how we can meet the specs
  - We must then refine the design to add detail
  - We must verify the design to ensure that it still meets all system goals (cost, speed, …)

Top-down vs Bottom-up

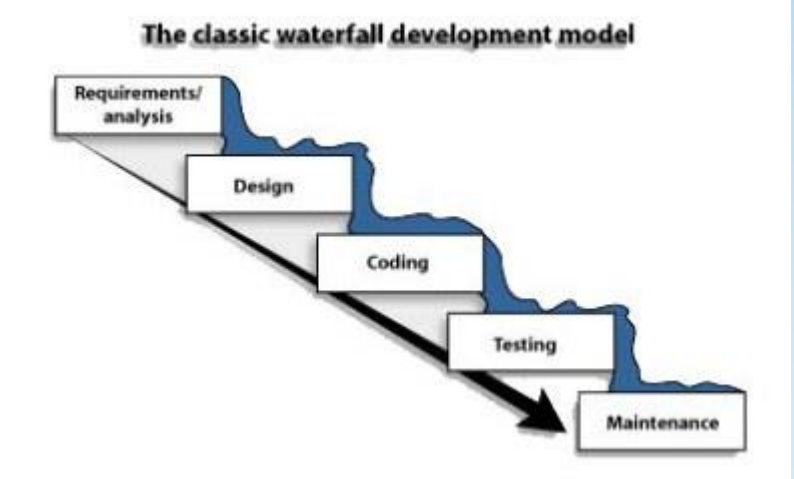
* The top-down design approach relies on beginning with the most abstract description of the system and conclude with concrete details
* In the bottom up approach, we start with components to build a system
* Bottom-up is needed when we do not have perfect insight into how later stages of the design process will turn out. Decisions at one stage of the design are based upon estimates of what will happen later
* If the original estimates are wrong, the design team has to backtrack and amend the original decisions
* The less experience a design team is the more it will have to rely on bottom-up design to refine the system

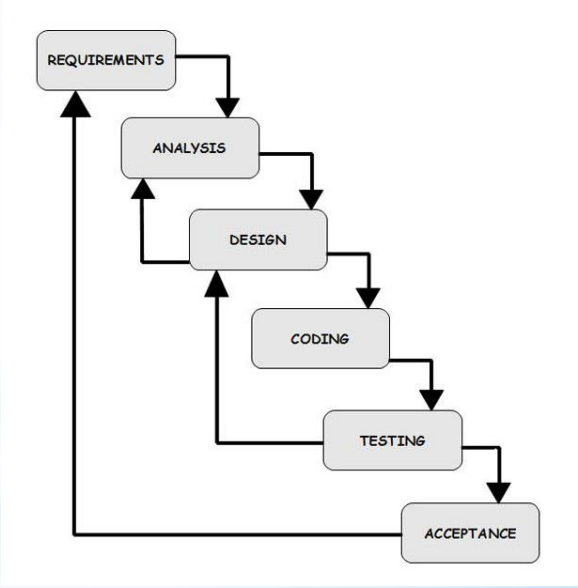
Design Flows

- A design flow is a sequence of steps to be followed during a design
- Different models of design flow exist
  - Waterfall model
  - Spiral model
  - Successive refinement
  - Hierarchical design
  - Concurrent engineering
  - Agile Methodologies

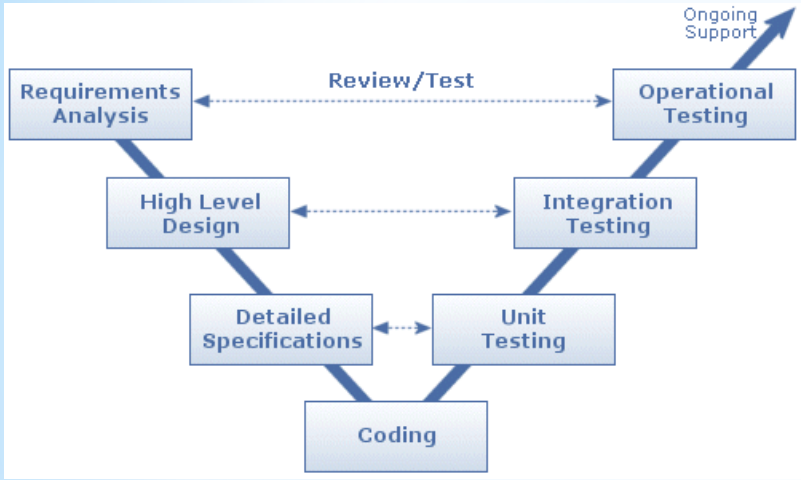
*Waterfall model

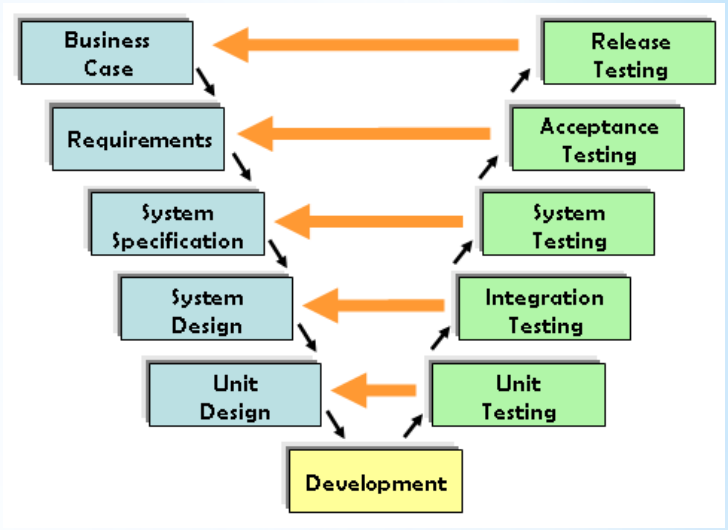
- Five major phases (levels of abstraction)
  - Requirements determines the basic features of the system
  - Analysis serves to indicate if the system is feasible
  - Architecture design decomposes the functionality into major components
  - Coding implements and integrates the components
  - Testing uncovers bugs
  - Acceptance and maintenance entails deployment in the field, bugs fixes and upgrades
- Limited amount of feedback to upper levels
- Unrealistic design process since most design processes imply experimentation and changes (bottom-up feedback)





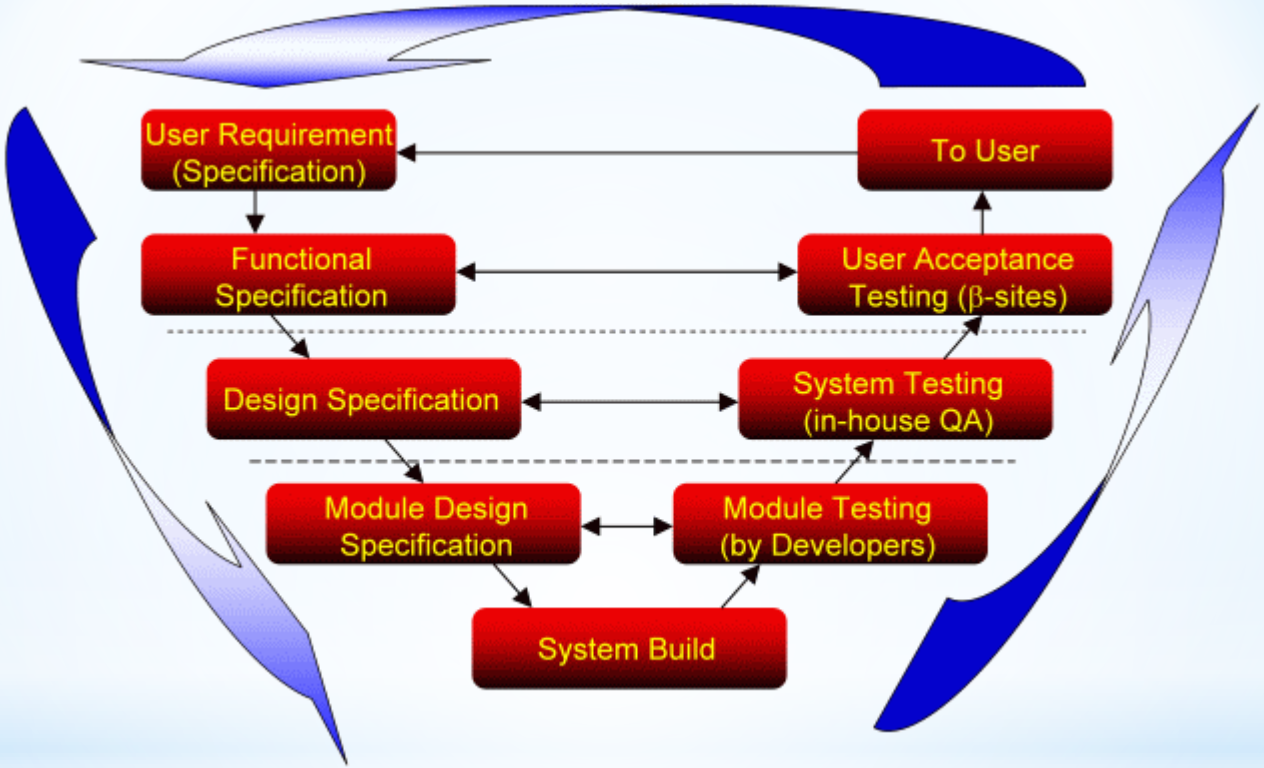
* Every implementation level must have a corresponding verification level





* More refined waterfall models

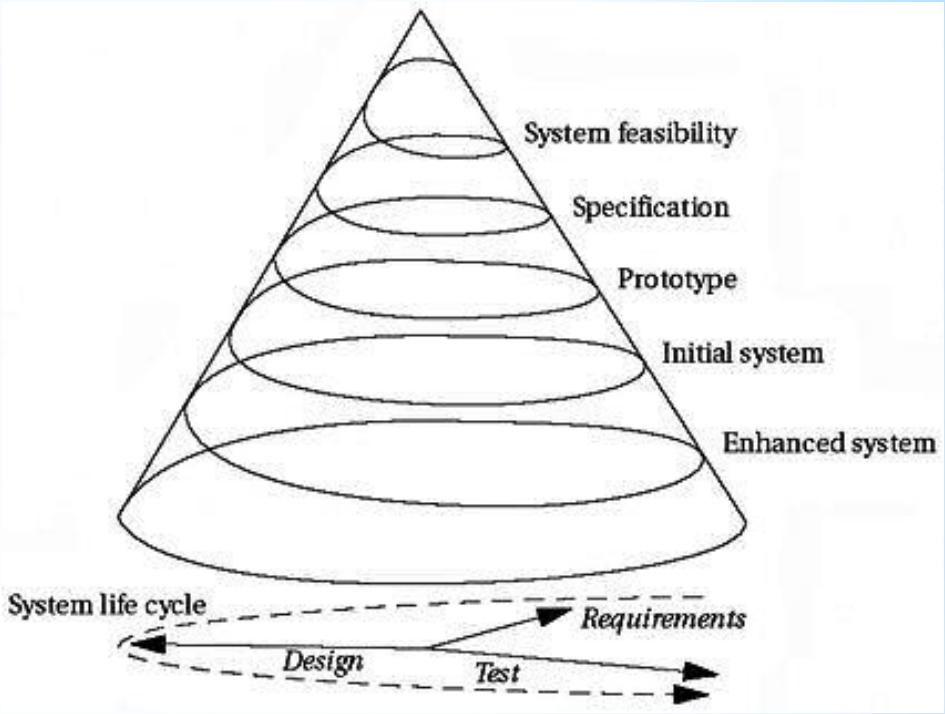
* Every implementation level must have a corresponding verification level



*More refined waterfall models

Spiral model

- This model assumes that several versions of the system will be built
- Early systems will be simple mock-ups constructed to aid the designers' intuition and to build experience with the system
- As design progresses, more complex systems are constructed
- At each level of design, designers go through requirements, construction and testing phases



*Spiral model

- At later stages when more complete versions of the system are constructed, each phase requires more work, widening the design spiral
- This successive refinement helps to understand the system through a series of design cycles
- The first cycle at the top of the spiral is simple and short
- The final cycle add detail learned from earlier cycles
- This model is more realistic
- However, this model could lead to large design times

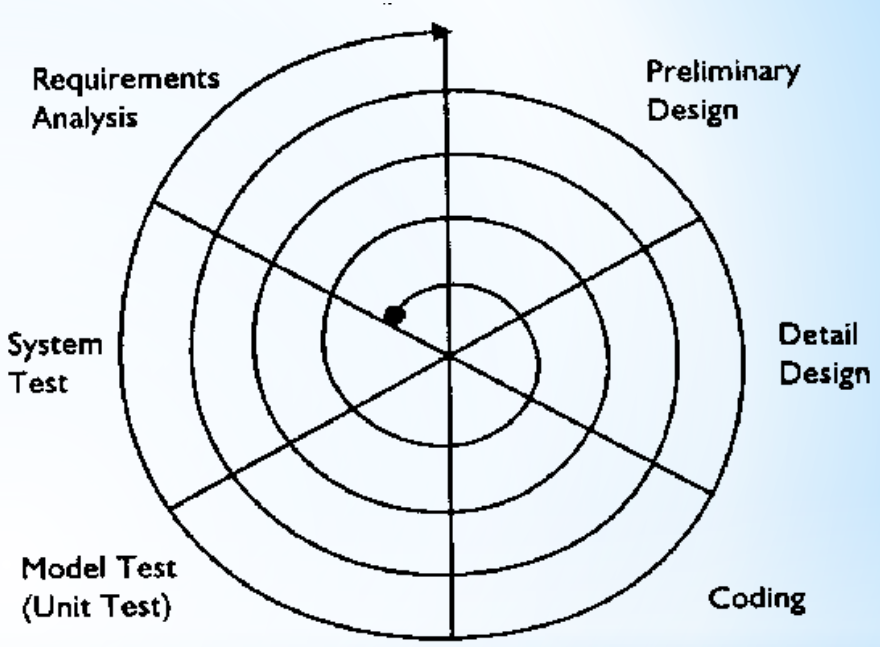
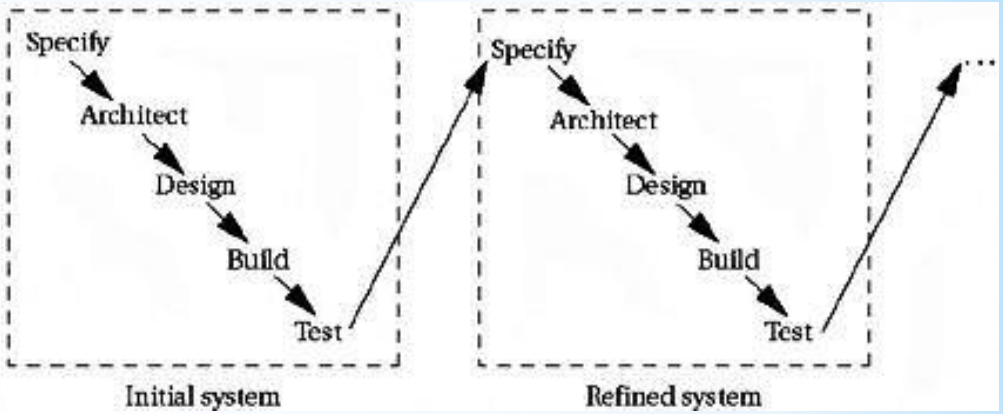


FIGURE 3: SPIRAL MODEL ADAPTED FROM [10].

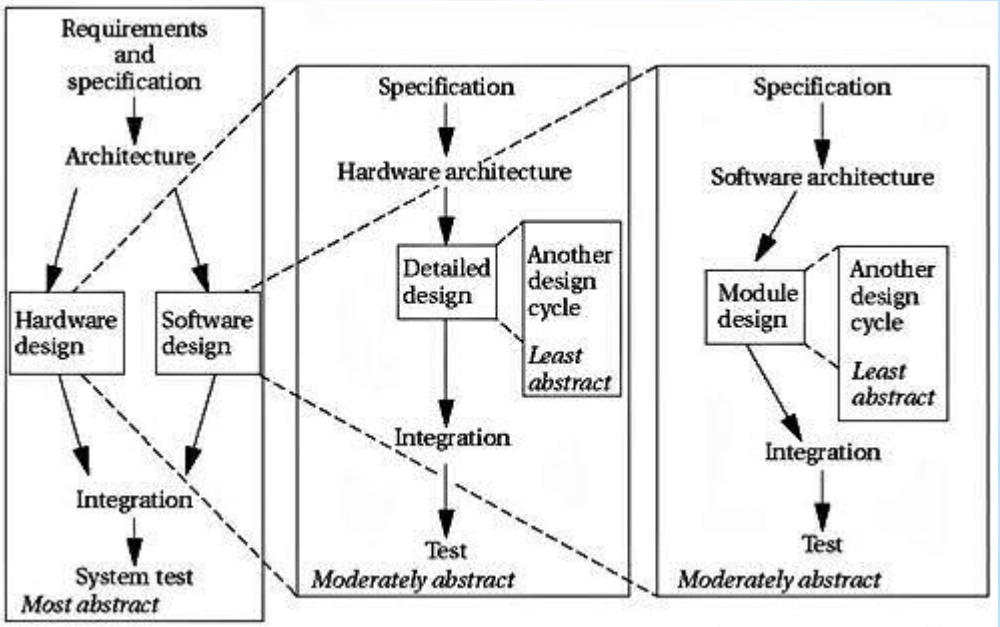
Successive refinement

- The system is built several times
- A first system is used as a rough prototype and successive models of the system are further refined
- This methodology is ideal when we are unfamiliar with the application domain for which the system is being built
- The refinement allows testing out architecture and design techniques
- The iterations may partially completed



Hierarchical design

* Embedded system design involves HW and SW design
* Complex embedded systems are built of smaller designs, either brand new of reusable
* The design flow follows the levels of abstraction in the system, from complete system design flows at the most abstract, to design flows for individual components
* The implementation phase is itself a complete design flow, each being handled by a separate teams
* Good communication is vital



Concurrent Engineering

* When designing large systems with many people, it is easy to lose track of the complete design flow
* Each designer take a narrow view of his/her role in the design flow
* Concurrent engineering attempts to take a broader approach in the total flow
* Its goal is reduced design time but it can help with any aspect of the design (reliability, performance, power consumption, etc.)
* The main goal is to eliminate “over-the wall” design steps

* Elements of concurrent engineering:
  * Cross-functional teams
  * Concurrent product realization. The heart of concurrent engineering
  * Incremental information sharing to avoid surprises
  * Integrated project management

* Problem statement:
  * SW development projects suffer from:
    * Deviation of schedule (delivery) and budget…
    * Which impacts testing and final integration (the weakest link in the chain)…
    * Which impacts quality (testing is cut short)

* Agile

* Project goals
  * Improve delivery quality by:
    * Carrying out formal project development tools, flows and methods
    * Reducing risks in test and integration phases which are prone to be cut short and, ironically, take way more time than estimated/expected
    * Focusing on developing projects on schedule and on budget

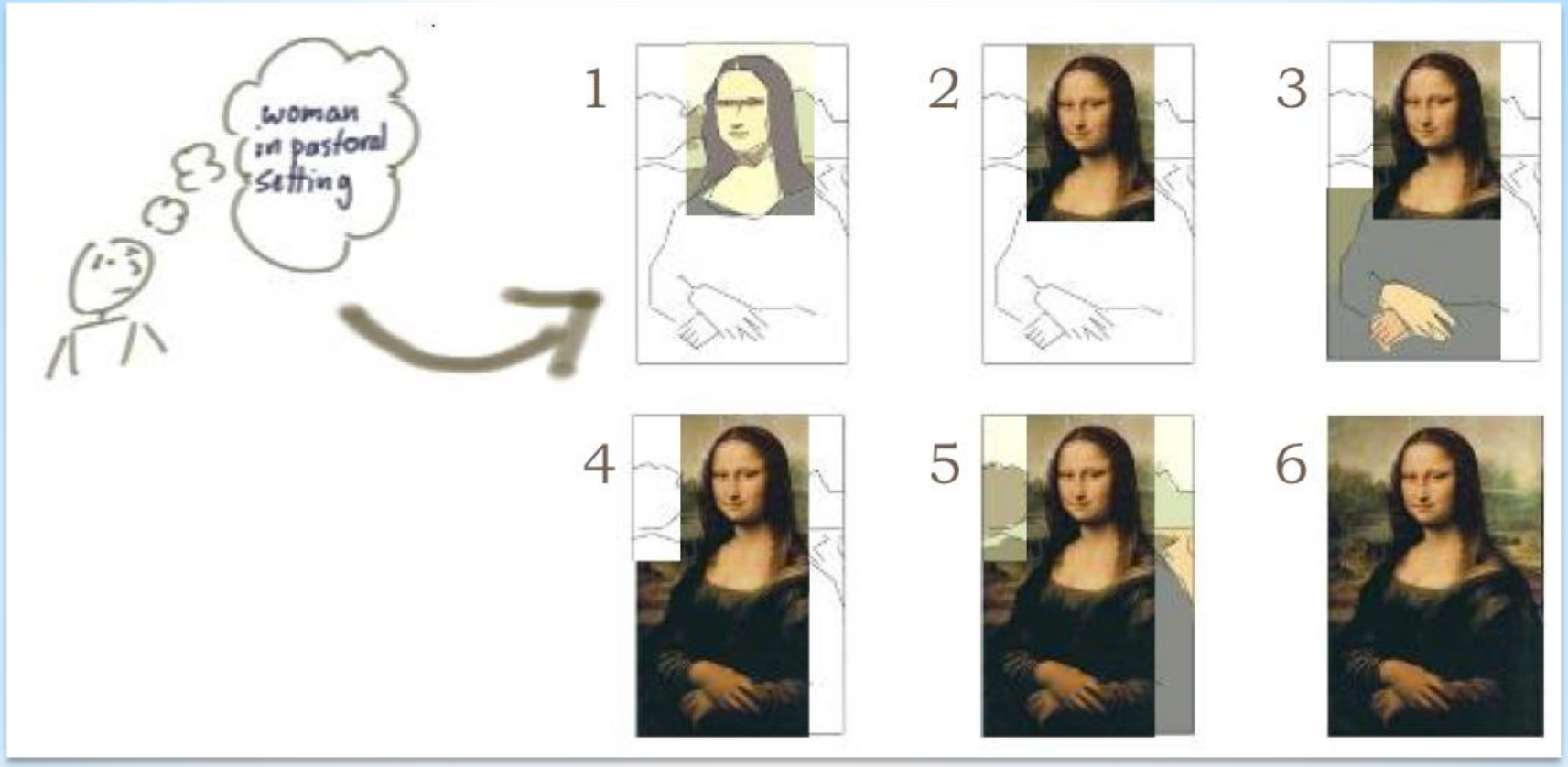
* Agile

* How
  * Implementing a well-defined Project Life Cycle, with formal:
    * Requirements gathering process
    * Project development methodology and philosophy
    * Tools for SW development and tracking
    * Metrics and reporting tools (automated generation of reports, dashboards, charts)
    * Stakeholder management

* Agile

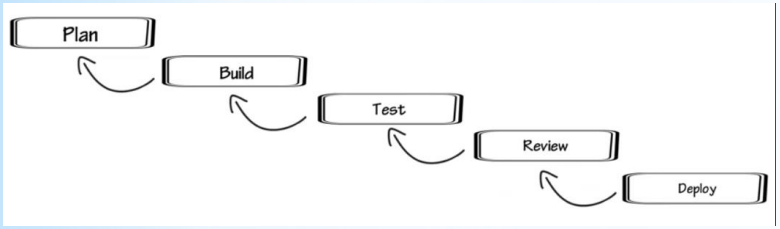
Agile Features

* Focus on (Principles)
  * Satisfying the customer through early and continuous delivery of valuable software
  * Welcoming changing requirements, even late in the development, for the sake of customer's competitive advantage
  * Delivering working software frequently, with a preference to short timescales. Working software is the primary measure of progress
  * Collaboration: Stakeholders and developers working and interacting together to deliver the greatest value
  * Time-boxed, incremental and iterative development

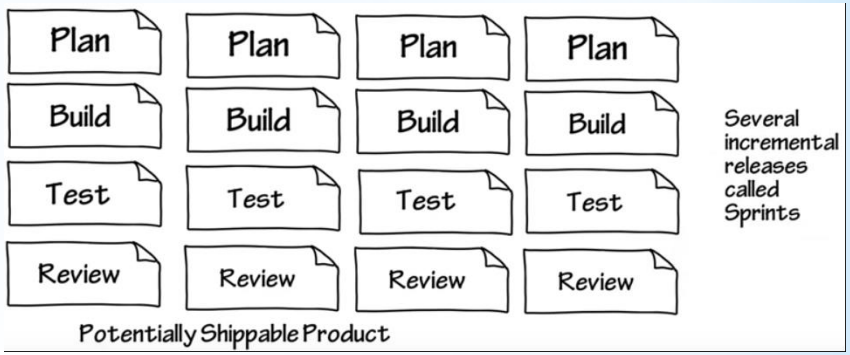


*Incremental and iterative development

* IMPORTANT: It is assumed that we are working in a time-bounded project, i.e., a project sizing has been done, presented to the customer and he/she has accepted

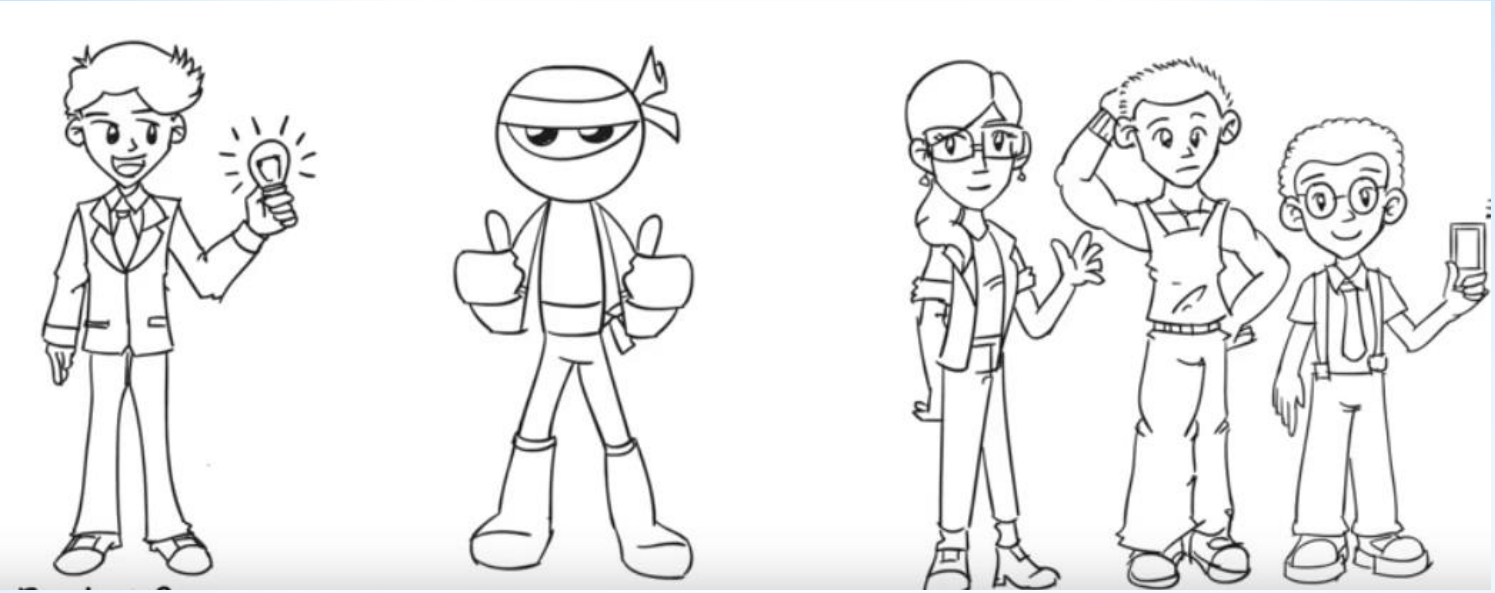


Waterfall



Scrum

* Scrum-like adoption: What is scrum

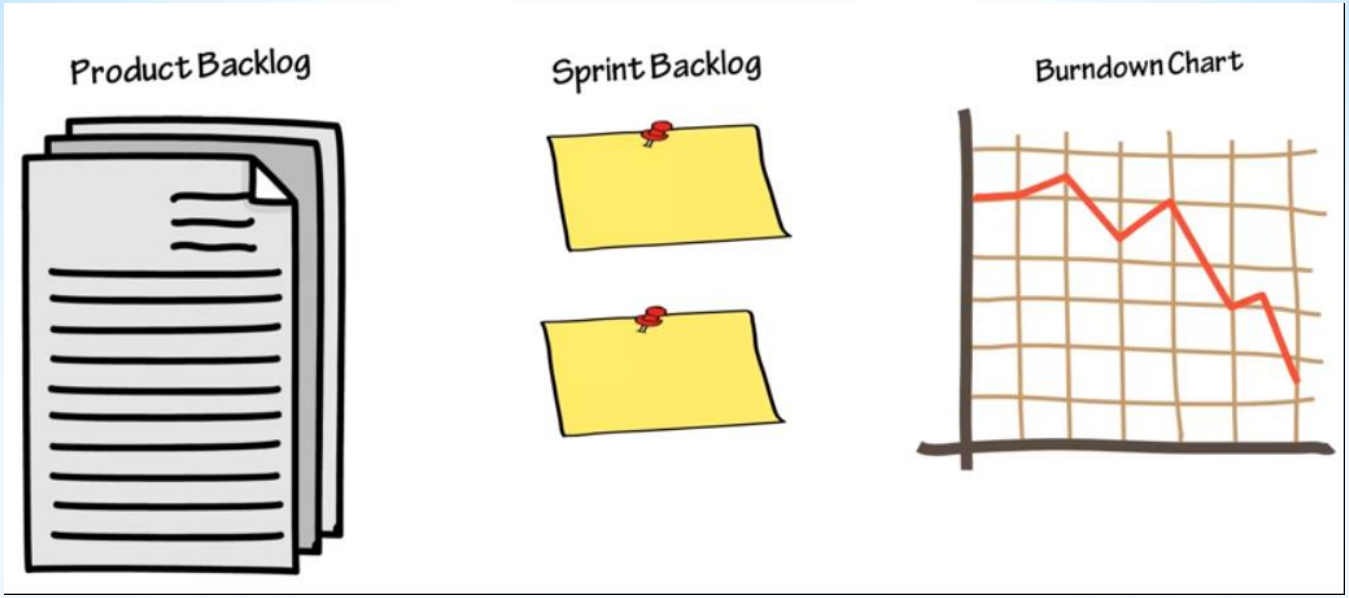


Product owner

Scrum master

Development team

*Scrum-like adoption: Roles



Created by the product owner.
A prioritized list of features (aka “user stories”) that will go into the product

User stories committed to a given sprint.
They are sized by the team & scrum master

Reporting mechanism that shows how the work is being completed

*Scrum-like adoption: Artifacts

*Scrum-like adoption: Ceremonies

Sprint Planning

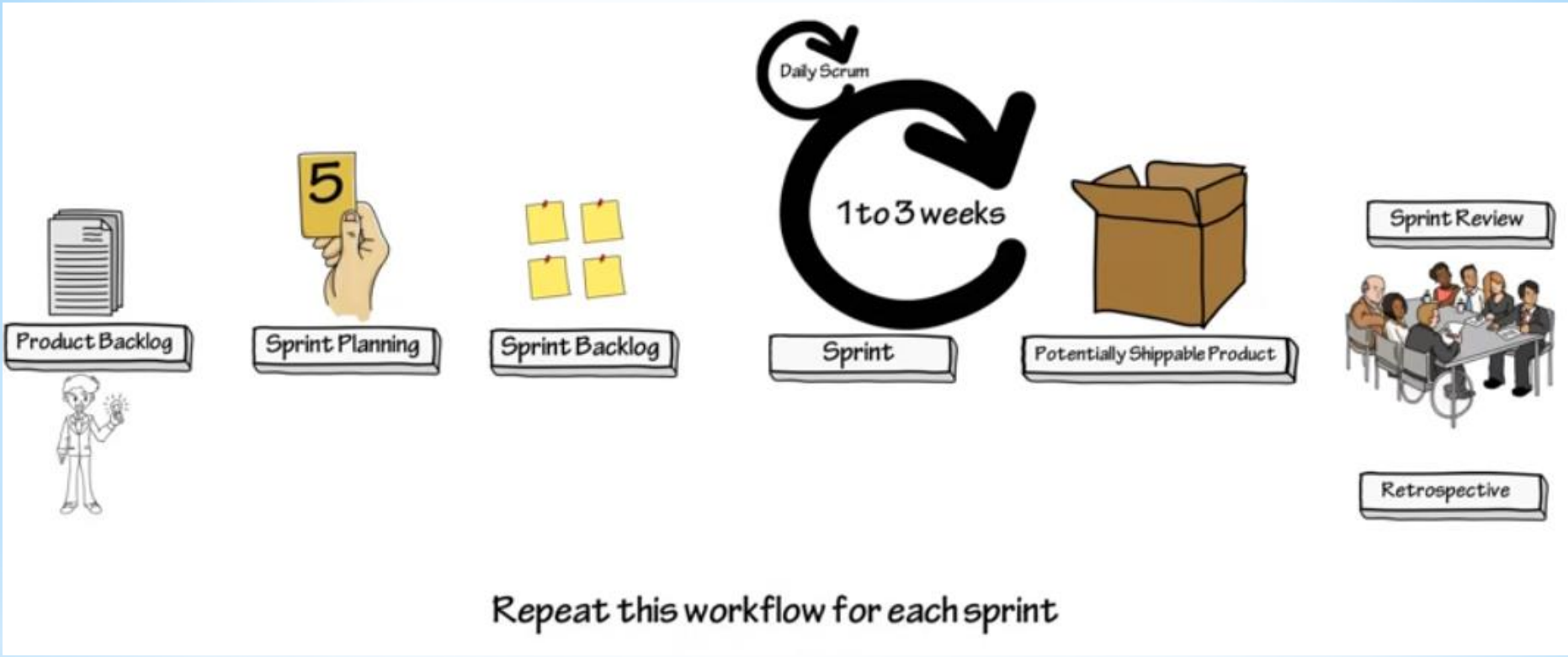
Discussion on what user stories will be done in a given sprint, size complexity. Output: sprint backlog

Daily Scrum

Brief discussion on what has been done, what will be done, risks, showstoppers, etc.

Sprint Review

Demo to the product owner and customer. Customer gives feedback and may even add/remove/adjust/change requirements



*Scrum Workflow

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions

Requirements

- Informal descriptions of what the “customer” wants
- It is directed to the outward behavior of the system
- The goal is to create a document that represents an effective communication between the customers and the designers
- With this document the designer knows what he/she is expected to design for the customer
- And the customer (final customer of marketing department) understands what he/she will get

Requirements and Specifications

* Sometimes these terms are used as synonyms
* There's a subtle difference
* Requirements are more informal
* When requirements are refined and "translated" to more quantitative/technical information, they are called specifications
* Specifications are more detailed, precise, and consistent descriptions of the system that can be used to create the architecture
* Separating requirements and specifications is important because of the gap between what the customer can describe about the system they want and what the designer need to design the system
* Consumers and customers' understanding of the system is based on how they envision users' interaction with the system
* It is very common that they have unrealistic expectations as to what can be done within their budgets and sometimes with current technology
* It is also common that they express their desires in a different language from designer's jargon

Types of Requirements

- Functional requirements: What the system must do
- Nonfunctional requirements: Any number of other attributes such as
  - Performance
  - Cost, target price, manufacturing cost, NRE cost,…
  - Physical size and weight
  - Power consumption
  - …

Good Requirements

- **Correctness:** The requirements should not mistakenly describe what the customer wants. Part of correctness is avoiding over-requiring—the requirements should not add conditions that are not really necessary
- **Unambiguousness:** The requirements document should be clear and have only one plain language interpretation
- ***Completeness:*** All requirements should be included.
- ***Verifiability:*** There should be a cost-effective way to ensure that each requirement is satisfied in the final product. For example, a requirement that the system package be "attractive" would be hard to verify without some agreed upon definition of attractiveness

Good Requirements

- ***Consistency:*** One requirement should not contradict another requirement.
- ***Modifiability:*** The requirements document should be structured so that it can be modified to meet changing requirements without losing consistency, verifiability, and so forth.
- ***Traceability:*** Each requirement should be traceable in the following ways:
  - We should be able to trace backward from the requirements to know why each requirement exists.
  - We should also be able to trace forward from documents created before the requirements (e.g., marketing memos) to understand how they relate to the final requirements.
  - We should be able to trace forward to understand how each requirement is satisfied in the implementation.
  - We should also be able to trace backward from the implementation to know which requirements they were intended to satisfy.

Simple requirements form

* A simple requirements form should at least include
  * Name
  * Purpose: One or two line description of what the system is supposed to do
  * Inputs and outputs
  * Functions: A more detailed description of the system
  * Performance
  * Manufacturing cost
  * Power
  * Physical size and weight

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions

Specs

- The specification (specs) is more precise
- The specs describe the sort of system that will satisfy the requirements
- It is the very exact deliverable of the designers to the customer
- It is written in a way that accurately reflects the customer's requirements in a way that can be clearly followed by the design team
- It is essential to create working systems with minimal design effort

Specs

- The primary goal of specs is to avoid making wrong or faulty assumptions
- Specs should be pretty understandable so that anyone can verify that system requirements and customer expectations are met
- What would happen if specs are unclear?

Specs process

- Sommerville's six step real-time design process
  - Identify the stimuli to be processed by the system and the required responses to these stimuli
  - For each stimulus and response, identify the timing constraints. These timing constraints must be quantifiable
  - Aggregate the stimulus and response processing into concurrent software processes. A process may be associated with each class of stimulus and response
  - Design algorithms to process each class of stimulus and response. These must meet the given timing requirements
  - Design a scheduling system that will ensure that processes are started in time to meet their deadlines
  - Integrate using (if needed) an RTOS

Specs process

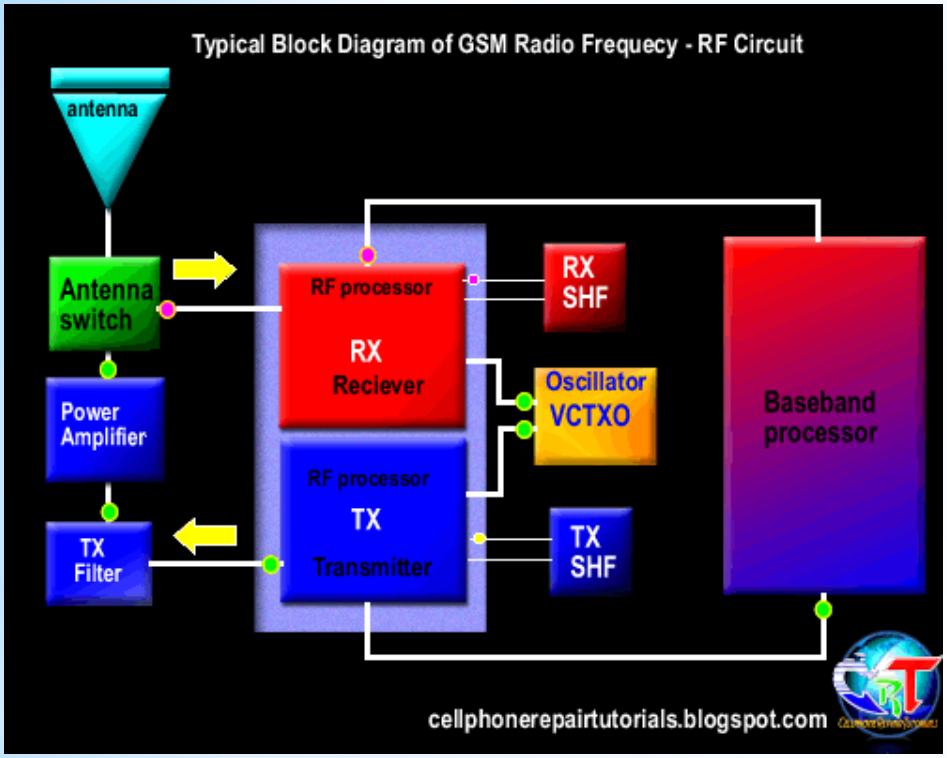
- Sommerville's six step real-time design process
  - Identify the stimuli to be processed by the system and the required responses to these stimuli
  - For each stimulus and response, identify the timing constraints. These timing constraints must be quantifiable
  - Aggregate the stimulus and response processing into concurrent software processes. A process may be associated with each class of stimulus and response
  - Design algorithms to process each class of stimulus and response. These must meet the given timing requirements
  - Design a scheduling system that will ensure that processes are started in time to meet their deadlines
  - Integrate using (if needed) an RTOS

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions

System Analysis

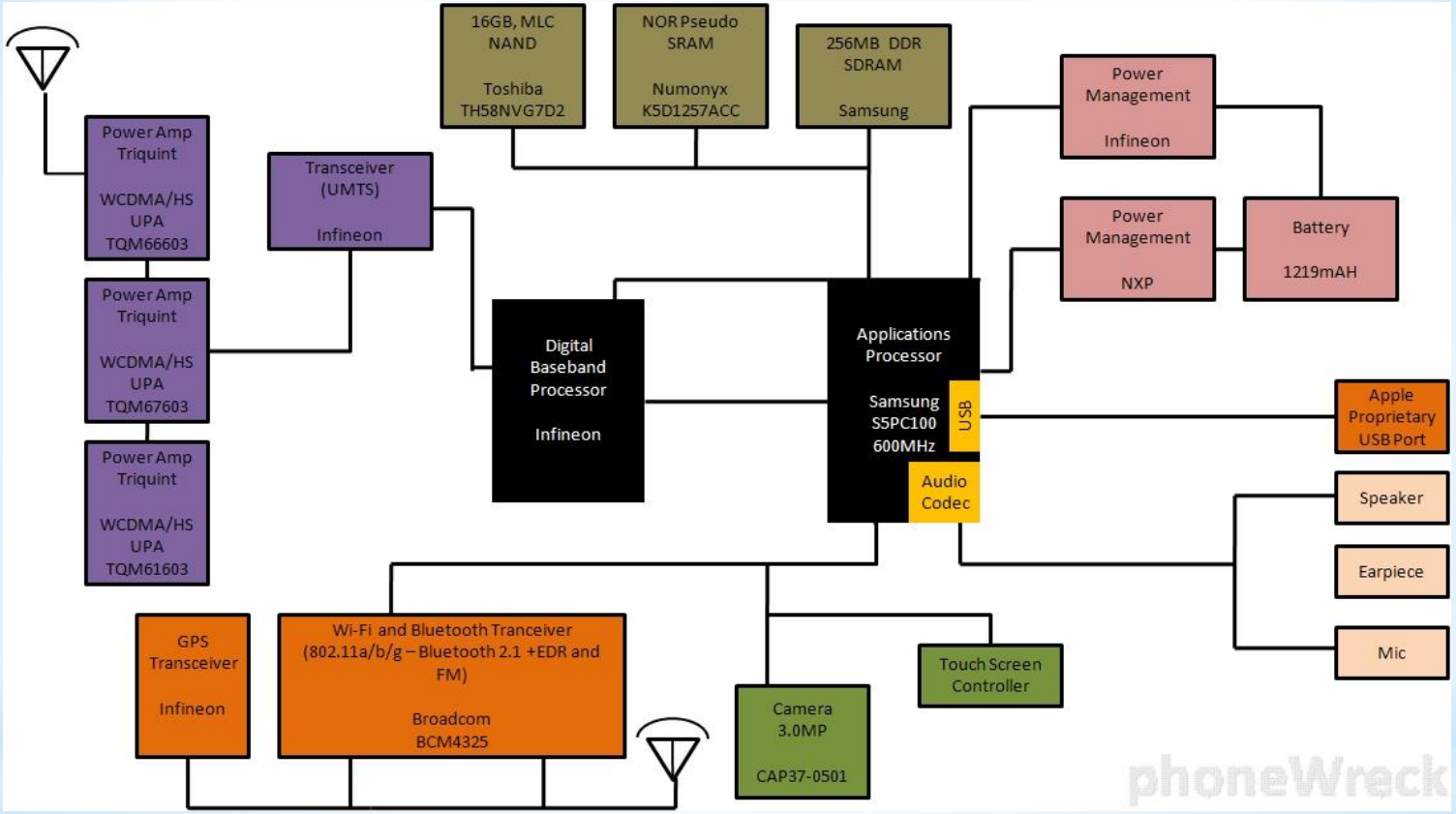
- Describing how the system implements specs is the purpose of system analysis and architecture design
- It is a plan for the overall structure of the system that will be used to design the components that make up the architecture
- The most part of designer's time in this phase is spent doing algorithm development, i.e., exploring approaches to solve the problems defined in the specs phase
- In this phase the designer is not concerned with the details of how the algorithm will be implemented
- The partitioning decisions of where the algorithms will be hosted (DSP, GPP, GPU, ASIC, FPGA, μC, etc.) are not a concern



* System analysis outcome

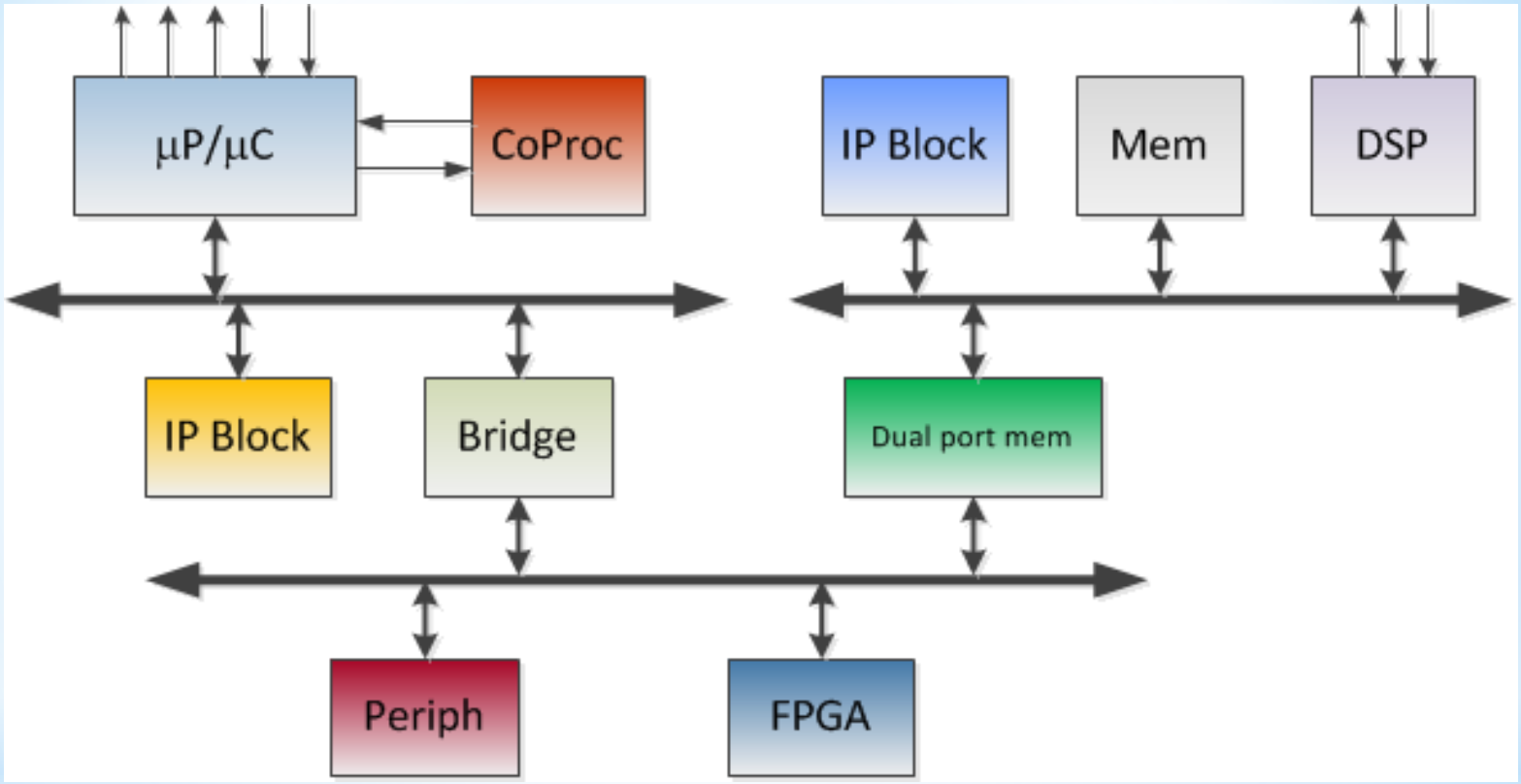
* System Analysis

* Refinement of the overall system analysis that may yield a HW architecture…



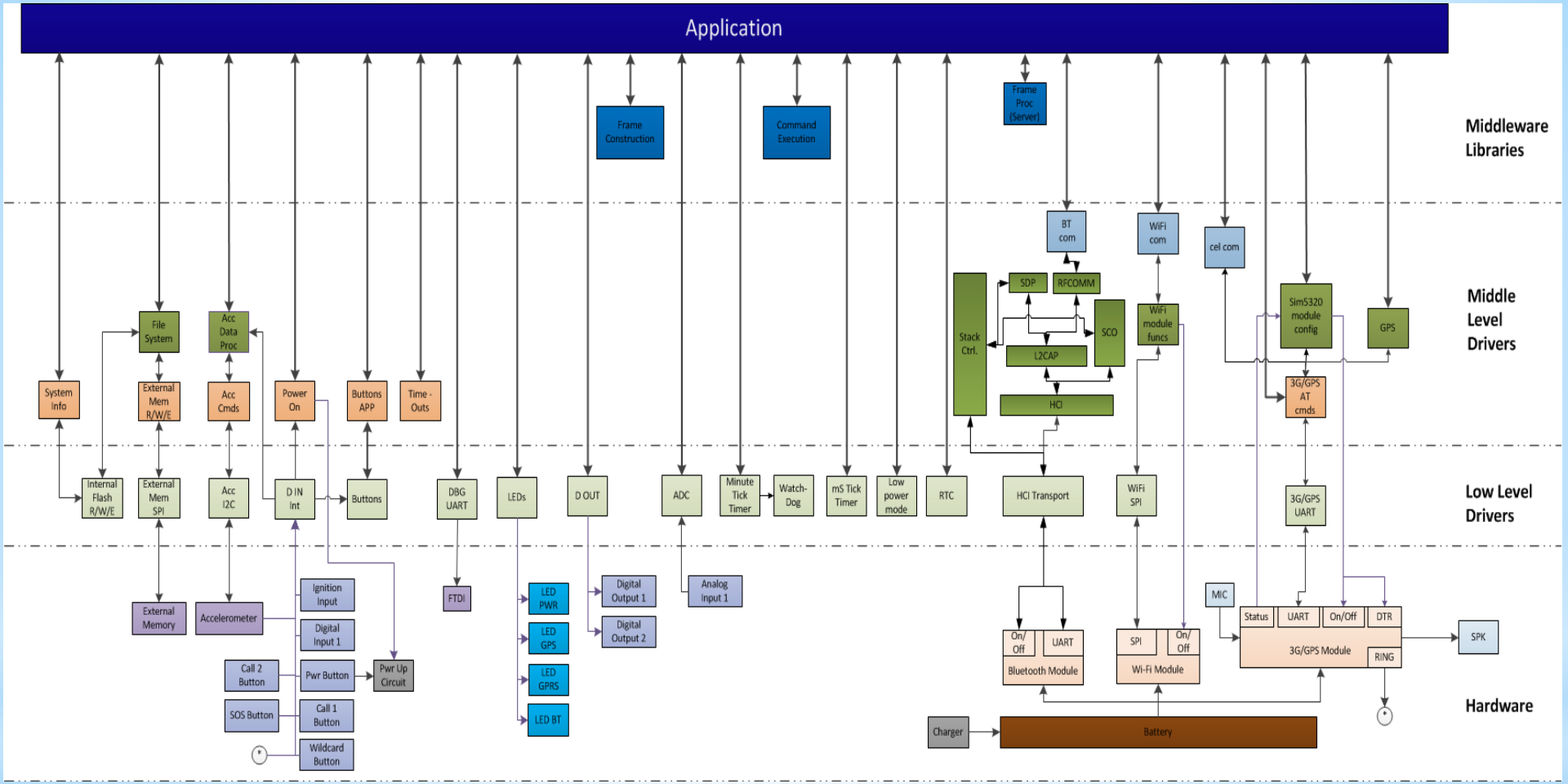
* System Analysis & Architecture Design

*System Architecture



*Architecture Design

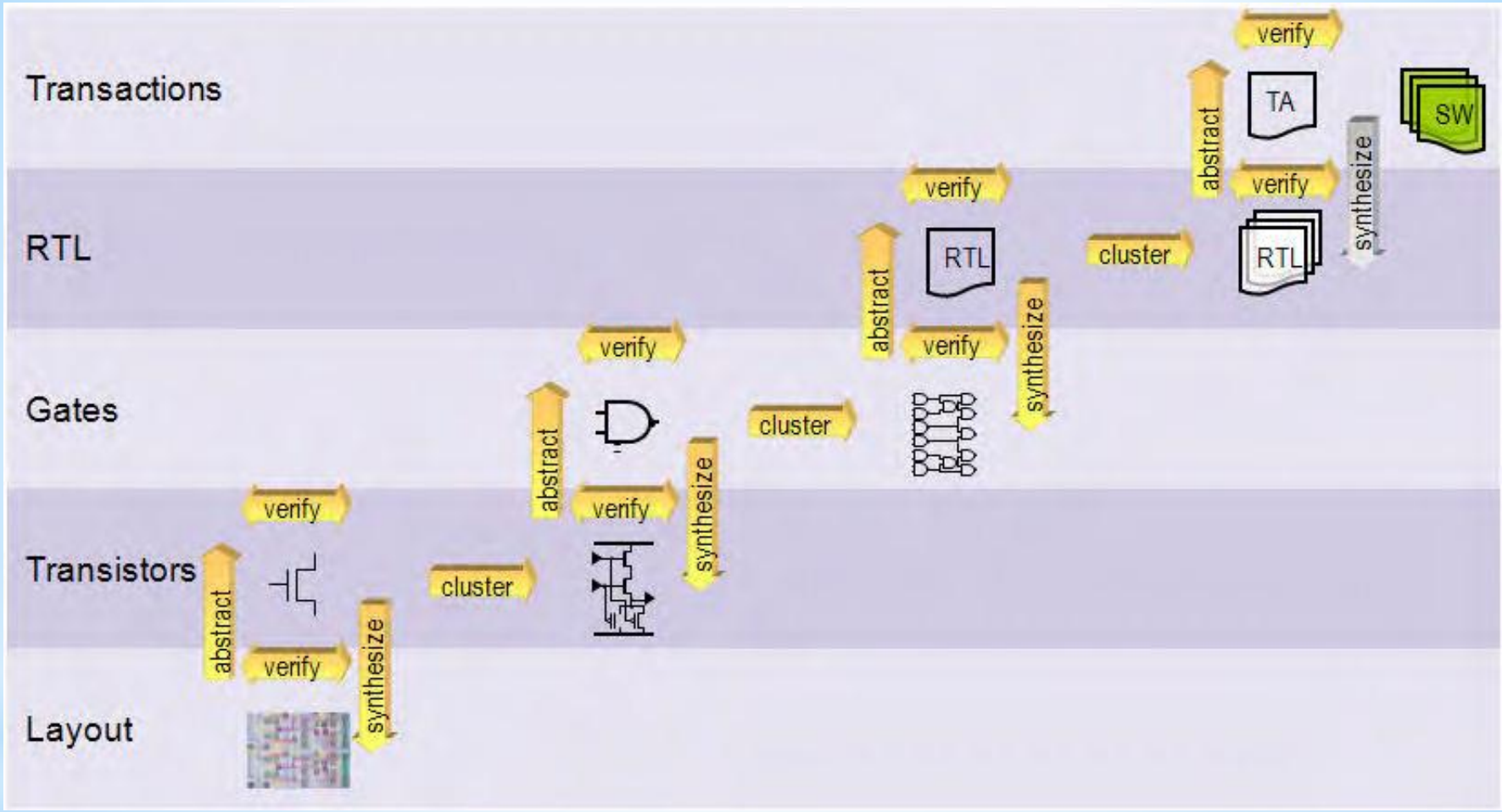
And a FW architecture



System Analysis & Architecture Design

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions



*Levels of abstraction

Hardware Implementation

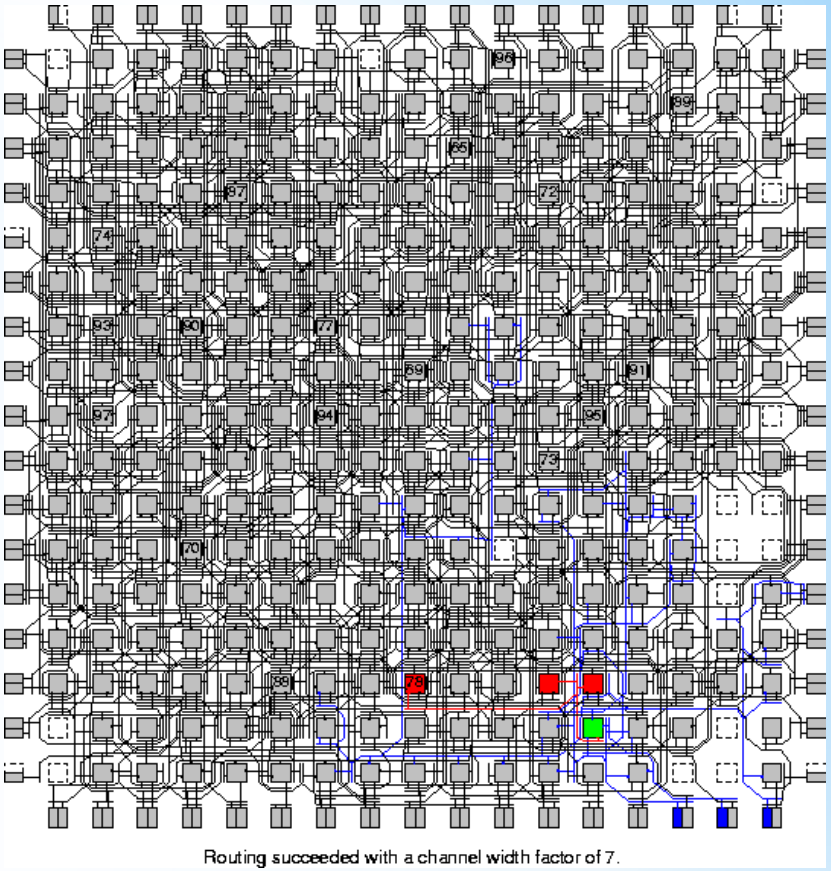
* Modern hardware implementation starts with register-transfer-level (RTL)
* RTL is a level of abstraction where functionality and timing of a circuit is modeled with the accuracy of a clock cycle, i.e., it is known in which clock cycle each operation occurs but the delay of each operation on the inputs of the registers is not known
* The number of registers and bit widths is precisely known
* Target technologies: ASIC or FPGA
* The model is written in an HDL
* It is meant to be simulated under a variety of timing models

Logic Synthesis

- Synthesis consists in turning the HDL model into a design architecture in terms of interconnection of logic gates and registers
- Two substeps:
  - RTL synthesis and module generation: Transforms high level operators into Boolean gates
  - Logic synthesis: optimizes the combinatorial logic resulting from RTL synthesis under cost and performance constraints (area, delay, power)
- Modern EDA (Electronic Design Automation) tools perform these tasks

Placement, Routing and Extraction

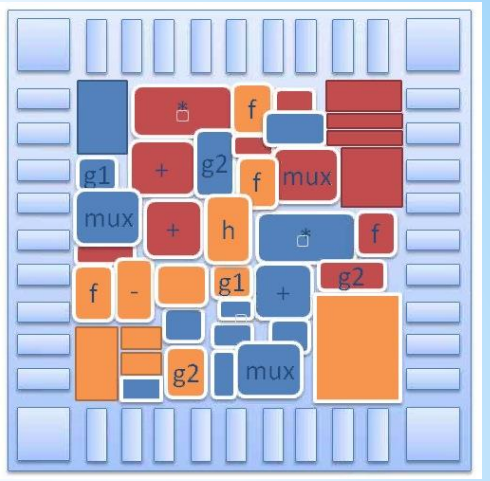
- After synthesis, gates are placed on silicon. Placement must avoid overlaps between cells and must satisfy timing constraints, avoid excessively long wires on critical paths
- An extremely difficult problem
- Routing consists in generating (or selecting) the wires that will interconnect the placed cells

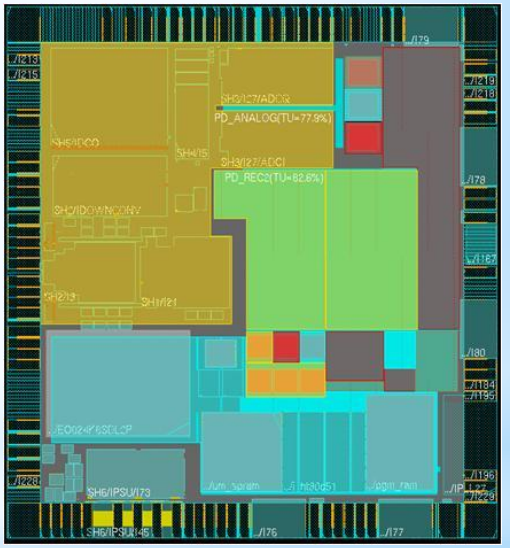


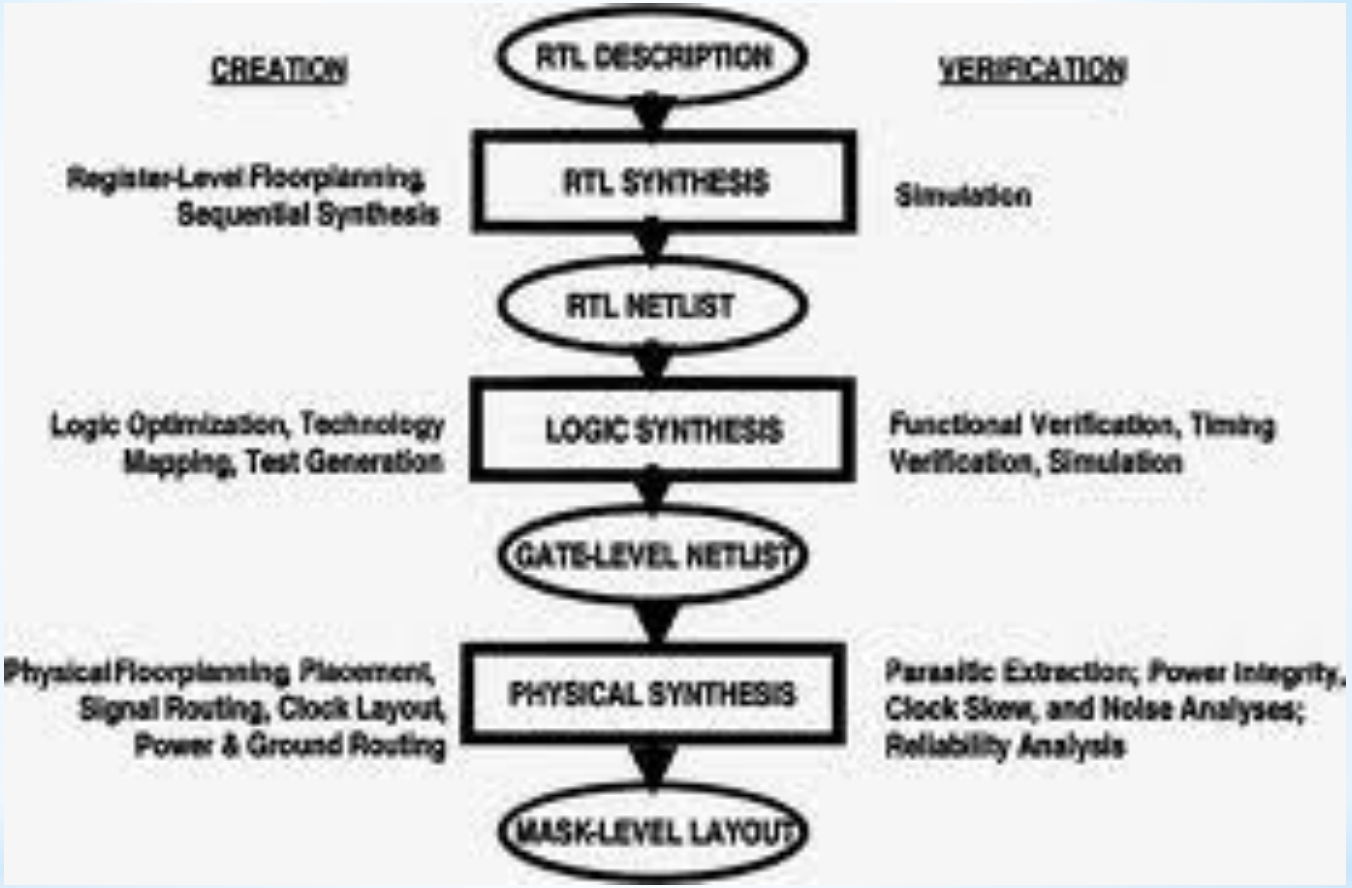
Placemente & routing in FPGA

Placement, Routing and Extraction

* Extraction is the process of verifying the correct placement and routing
  * All design rules are satisfied by the final layout
  * All the desired interconnects have been realized
* It is done by extracting electrical and logic models from layout masks and comparing these models with the input netlist







*Hardware Implementation Summary

Software Implementation

* In the embedded systems world, far more effort goes into embedded software than hardware development
* Software development tools are crucial due to the limited accessibility of the CPU
* Main tools/steps
  * Compilation
  * Assembly
  * Linking
  * Debugging
  * Profiling
* Embedded processors are useless without these development tools
* A good set of software tools can create a tight, efficient code that reduces memory costs and cuts performance requirements so that the processor can be run at lower clock rates (and lower energy consumption and power dissipation)

Software Implementation

* These tools are gathered into Integrated development environments (IDE)
* At present time, IDEs are mandatory and can have a profound effect on programmer efficiency
* Key factors affecting design team's ability to meet goals:
  * Compiler's and assembler's code-optimizing efficiency
  * Programming styles used to develop the source code
* In embedded systems, the C language is the *de facto* programming language for generating source code

Embedded Compiler

- A compiler translates HLL source code into assembly language semantics
- Modern compilers typically consist of a front end and a back end
- Front end is where syntactic and semantic processing takes place
- Back end performs general optimizations as well as code generation and further optimization for a particular target processor

Embedded Assembler

* An assembler converts machine-generated or hand-coded assembly level semantics into machine code
* Some modern assemblers do some additional tasks
  * Instruction relaxation
  * Instruction scheduling
  * Instruction alignment

Embedded Linker

* A linker serves as the intermediary between the object file generated by the compiler or the assembler and the executable code required to run the program on the target processor
* The linker places the code, data and other sections of object file into an executable file
* Object files contain symbolic references and the linker resolves these references to specific addresses based on the memory map of the target hardware

Example 2-1. C64x+ Linker Command File

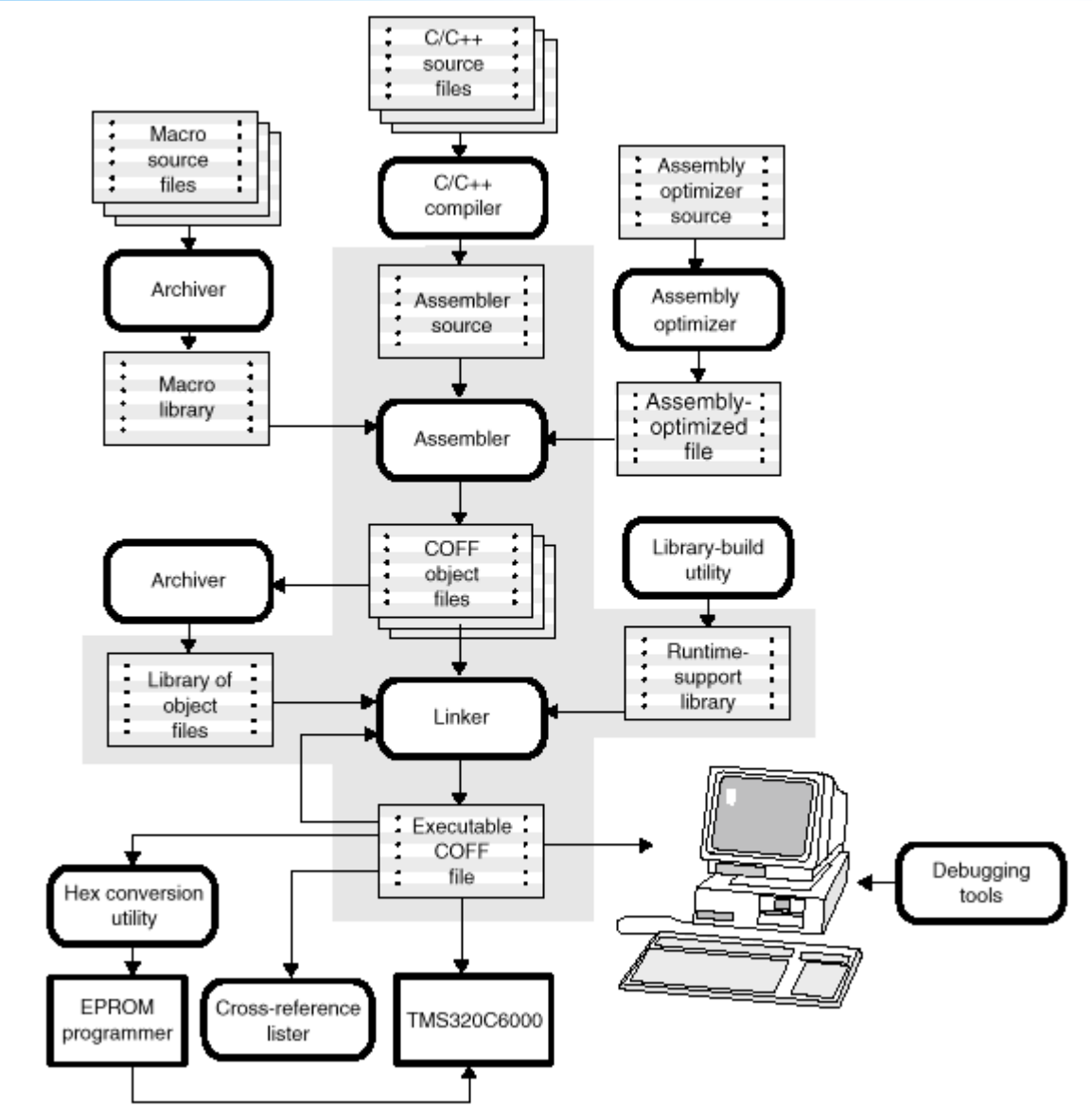
```
MEMORY
{
        L2SRAM:   origin = 00800000h   length = 001C0000h
        CE0:      origin = 80000000h   length = 01000000h

}

SECTIONS
{

        .cinit     >  L2SRAM
        .text      >  L2SRAM
        .stack     >  L2SRAM
        .bss       >  L2SRAM
        .const     >  L2SRAM
        .data      >  L2SRAM
        .far       >  L2SRAM
        .switch    >  L2SRAM
        .sysmem    >  L2SRAM
        .tables    >  L2SRAM
        .cio       >  L2SRAM
        .external  >  CE0
}
```

*Embedded Linker



*Software Implementation Summary

Outline

* Introduction
* Design Methodologies
* Requirements Analysis
* Specifications
* System Analysis and Architecture Design
* System Implementation
* Quality Assurance
* Conclusions

Quality Assurance

* The quality of a product or service can be judged by how well it satisfies its intended function
* Quality assurance is the process for delivering a satisfactory system
* Important… it is a process that takes place during the life cycle of the project
* It involves all the activities related to ensuring that the system will be accepted

Quality Assurance

- During system development
  - Design review: a meeting in which team members (and external members) discuss a design
- After system development
  - Verification
  - Validation
  - Field tests
  - Deployment

Outline

- Introduction
- Design Methodologies
- Requirements Analysis
- Specifications
- System Analysis and Architecture Design
- System Implementation
- Quality Assurance
- Conclusions

Conclusions

- Embedded system design is a very demanding process
- Different system design methodologies exist depending on the type of system, skills, budget, etc.
- Picking the right system design methodology is crucial
- Five main steps
- Highlights on HW and SW implementation were given