# Software engineering: System process

## v process:

* Requirements Analysis
* System Design
* Program Design
* Coding
* Unit & Integration Testing
* System Testing
* Acceptance Testing
* Operation Maintenance

## static and dynamic architecture:

Static architecture is the system interactions; while dynamic is the system interaction and what happens when they are executed.

testing categories

* Unit/Component testing
  + Testing of individual program components
  + Usually the responsibility of the component developer (except sometimes for critical systems)
  + Tests are derived from the developer’s experience
* Integration/System testing
  + Testing of groups of components integrated to create a system or sub-system
  + The responsibility of an independent testing team
  + Tests are based on a system specification
* Interface Testing
  + Takes place when modules or sub-systems are integrated to create larger systems
  + Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
* OO Testing
  + The components to be tested are object classes that are instantiated as objects
  + Larger grain than individual functions so approaches to white-box testing have to be extended
* Smoke Testing

Refers to the first test made after assembly or repairs to a system

* + Software components that have been translated into code are integrated into a “build.”
  + A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
  + A series of tests is designed to expose errors that will keep the build from properly performing its function.
  + The intent should be to uncover “show stopper” errors that have the highest likelihood of throwing the software project behind schedule.
  + The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.
  + The integration approach may be top down or bottom up.
* Regression Testing

Is the re-running of all tests after each error is corrected.

* Stress Testing
  + Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light
  + Executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
  + Stressing the system test failure behaviour.. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data
  + Particularly relevant to distributed systems which can exhibit severe degradation as a network becomes overloaded

Configuration Management:

* New versions of software systems are created as they change
  + For different machines/OS
  + Offering different functionality
  + Tailored for particular user requirements
* Configuration management is concerned with managing evolving software systems
  + System change is a team activity
  + CM aims to control the costs and effort involved in making changes to a system

# Embedded c

* Switch case (one of the cases without break)

When a switch statement is used without breaks, the  
code continues executing even after a matching case is found.

* Post increment and pre increment
  + loop = ++count; /\* same as count = count + 1; loop = count;

loop = count++; /\* same as loop = count; count = count + 1;

* #define?? #include?? #pragma?? #asm?? #ifdef...#endif?
  + #define define a macro
  + #include include a source file
  + #pragma is for compiler directives that are machine-specific or operating-system-specific, i.e. it tells the compiler to do something, set some option, take some action, override some default, etc. that may or may not apply to all machines and operating systems.
  + #asm to include the assembly directive
  + #ifdef compile these lines if NAME is not defined
  + #endif to delimit the scope of these directives

The term directive is applied in a variety of ways that are similar to the term command. It is also used to describe some [programming language](http://en.wikipedia.org/wiki/Programming_language) constructs (e.g. those specifying how a [compiler](http://en.wikipedia.org/wiki/Compiler) or [assembler](http://en.wikipedia.org/wiki/Assembly_language) should process its input).

* const: means that something is not modifiable
* extern: Keyword extern indicates that the actual storage and initial value of a variable, or body of a function, is defined elsewhere, usually in a separate source code module.
* volatile used in embedded systems programming specifically, C's volatile keyword is a qualifier that is applied to a variable when it is declared. It tells the compiler that the value of the variable may change at any time--without any action being taken by the code the compiler finds nearby.

volatile int foo;

A variable should be declared volatile whenever its value could change unexpectedly. In practice, only three types of variables could change:

1. Memory-mapped peripheral registers

2. Global variables modified by an interrupt service routine

3. Global variables accessed by multiple tasks within a multi-threaded application

* static

|  |  |
| --- | --- |
|  | A static variable inside a function keeps its value between invocations.  A static global variable or a function is "seen" only in the file it's declared in  When specified on a function declaration, it makes the function local to the file. |

* register: The register type modifier tells the compiler to store the variable being declared in a CPU register (if possible), to optimize access.

Local Variables are stored in Stack. Register variables are stored in Register. Global & static variables are stored in data segment. The memory created dynamically are stored in Heap And the C program instructions get stored in code segment and the extern variables also stored in data segment.

* Data Types:
  + short int 2 16 -32,768 -> +32,767 (16kb)
  + unsigned short int 2 16 0 -> +65,535 (32Kb)
  + unsigned int 4 32 0 -> +4,294,967,295 ( 4Gb)
  + int 4 32 -2,147,483,648 -> +2,147,483,647 ( 2Gb)
  + long int 4 32 -2,147,483,648 -> +2,147,483,647 ( 2Gb)
  + signed char 1 8 -128 -> +127
  + unsigned char 1 8 0 -> +255
  + float 4 32
  + double 8 64
  + long double 12 96

Difference between signed and unsigned variables:

An unsigned number means that the variable holds only positive values. Your typical unsigned int has a range from 0 to 65,535.

## C memory layout

* Program code
  + Static size
  + Loaded at one end of memory
* Stack and Heap grow into free memory
  + Stack grows and shrinks in ordered manner from one end
  + Heap blocks allocated near other end, including holes (free space in middle of heap)
* Stack contains:
  + Function local variables and arguments
  + Function return address
  + Space allocated at top of stack on function entry
  + Space freed on function return

**Dynamic allocation** is the automatic allocation of memory in C/C++, Unlike declarations, which load data onto the programs data segment, dynamic allocation creates new usable space on the programs STACK (an area of RAM specifically allocated to that program).

**Macro:** a *macro* is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro. There are two kinds of macros. They differ mostly in what they look like when they are used. *Object-like* macros resemble data objects when used, *function-like* macros resemble function calls.

**Function:** The function follows a simple algorithm

**MISRA Rules:**  is a software development standard for the [C](http://en.wikipedia.org/wiki/C_(programming_language)) [programming language](http://en.wikipedia.org/wiki/Programming_language) developed by [MISRA](http://en.wikipedia.org/wiki/MISRA) (Motor Industry Software Reliability Association). Its aims are to facilitate code [safety](http://en.wikipedia.org/wiki/Safety), [portability](http://en.wikipedia.org/wiki/Portability) and reliability in the context of [embedded systems](http://en.wikipedia.org/wiki/Embedded_systems), specifically those systems programmed in [ISO C](http://en.wikipedia.org/wiki/ISO_C).

A **namespace** (sometimes also called a name scope) is an abstract container or environment created to hold a logical grouping of unique [identifiers](http://en.wikipedia.org/wiki/Identifier) or [symbols](http://en.wikipedia.org/wiki/Symbol) (i.e., names). An identifier defined in a namespace is associated only with that namespace. The same identifier can be independently defined in multiple namespaces.

**typedef**is a [keyword](http://en.wikipedia.org/wiki/Keyword_(computer_programming)) in the [C](http://en.wikipedia.org/wiki/C_(programming_language)) and [C++](http://en.wikipedia.org/wiki/C%2B%2B) [programming languages](http://en.wikipedia.org/wiki/Programming_language). The purpose of typedef is to assign alternative names to existing types, most often those whose standard declaration is cumbersome, potentially confusing, or likely to vary from one implementation to another.

# Specifications of Embedded SW:

a) Functionality. What is the software supposed to do?

b) External interfaces. How does the software interact with people, the system’s hardware, other hardware, and other software?

c) Performance. What is the speed, availability, response time, recovery time of various software functions, etc.?

d) Attributes. What are the portability, correctness, maintainability, security, etc. considerations?

e) Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

# Embedded Systems:

**Real time system:** A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless!

**Input capture:** is a method of dealing with input signals in an [embedded system](http://en.wikipedia.org/wiki/Embedded_system).

Embedded systems using input capture will record a [timestamp](http://en.wikipedia.org/wiki/Timestamp) in memory when an input signal is received. It will also set a flag indicating that an input has been captured. This allows the system to continue executing without interruption while an input is being received while still having the capability to trigger events based on the exact time when the input was received.

The corresponding capability to trigger an [output](http://en.wikipedia.org/wiki/Output) at a specified time, based on a timestamp in memory, is called [output compare](http://en.wikipedia.org/wiki/Output_compare).

*The input capture module has the task of capturing the current value of the timer counter upon an input event. This module is mainly used for the frequency or time period measurements and pulse measurements (e.g. mean count rate measurement).*

**Interrupts:**

Types:

A **level-triggered interrupt** is a class of interrupts where the presence of an unserviced interrupt is indicated by a particular state, high level or low level, of the [interrupt request](http://en.wikipedia.org/wiki/Interrupt_request) line. A device wishing to signal an interrupt drives line to its active level, and then holds it at that level until serviced. It ceases asserting the line when the CPU commands it to or otherwise handles the condition that caused it to signal the interrupt.

An **edge-triggered** interrupt is a class of interrupts that are signalled by a level transition on the interrupt line, either a[falling edge](http://en.wikipedia.org/wiki/Falling_edge) (high to low) or a [rising edge](http://en.wikipedia.org/wiki/Rising_edge) (low to high). A device wishing to signal an interrupt drives a pulse onto the line and then releases the line to its inactive state. If the pulse is too short to be detected by [polled I/O](http://en.wikipedia.org/wiki/Polled_I/O) then special hardware may be required to detect the edge.

*How interrupts work:*

It is good practice to save (PUSH) to the stack any registers used during the ISR routine and to restore (POP) these registers at the end of the ISR routine, thus preserving the registers’ contents, just like a register is preserved within a subroutine program. The last instruction in the ISR routine is a RETI (RETurn from Interrupt) instruction and this instruction causes the 8051 to restore the IE register values, enable the INT0 flag, and restore the Program Counter contents from the stack.

**Synchronous Vs Asynchronous:**

Asynchronous communication utilizes a transmitter, a receiver and a wire without coordination about the timing of individual [bits](http://www.inetdaemon.com/tutorials/basic_concepts/number_systems/binary/bits_n_bytes.shtml).

When the receiver of a [signal](http://www.inetdaemon.com/tutorials/basic_concepts/communication/signals/index.shtml) carrying information has to derive how that [signal](http://www.inetdaemon.com/tutorials/basic_concepts/communication/signals/index.shtml) is organized without consulting the transmitting device, it is called asynchronous communication. In short, the two ends do not always negotiate or work out the connection parameters before communicating. Asynchronous communication is more efficient when there is low loss and low error rates over the [transmission medium](http://www.inetdaemon.com/tutorials/basic_concepts/communication/transmission/transmission_media.shtml) because data is not retransmitted and no time is spent setting negotiating the connection parameters at the beginning of transmission. Asynchronous systems just transmit and let the far end station figure it out. Asynchronous is sometimes called "best effort" transmission because one side simply transmits, and the other does its best to receive.

Synchronous systems negotiate the communication parameters at the [data link](http://www.inetdaemon.com/tutorials/basic_concepts/network_models/osi_model/data_link.shtml) layer before communication begins. Basic synchronous systems will synchronize both clocks before transmission begins, and reset their numeric counters for errors etc. More advanced systems may negotiate things like [error correction](http://www.inetdaemon.com/tutorials/basic_concepts/communication/error_correction/index.shtml) and compression.

**PWM:** Pulse width modulation (PWM) is a powerful technique for controlling analog circuits with a processor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement and communications to power control and conversion.

The micro-controller used in this project contains a built-in A/D converter with

a multiplexer, but it lacks of a D/A converter. To manage this problem we will

use the internal PWM generator instead. Pulse-width modulation (PWM) uses

a square wave whose pulse width is modulated resulting in the variation of the

average value of the waveform.

The term [duty cycle](http://en.wikipedia.org/wiki/Duty_cycle) describes the proportion of 'on' time to the regular interval or 'period' of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on.

By controlling analog circuits digitally, system costs and power consumption can be drastically reduced. What's more, many microcontrollers and DSPs already include on-chip PWM controllers, making implementation easy.

Figure 2 shows a simple circuit that could be driven using PWM. In the figure, a 9 V battery powers an incandescent lightbulb. If we closed the switch connecting the battery and lamp for 50 ms, the bulb would receive 9 V during that interval. If we then opened the switch for the next 50 ms, the bulb would receive 0 V. If we repeat this cycle 10 times a second, the bulb will be lit as though it were connected to a 4.5 V battery (50% of 9 V). We say that the duty cycle is 50% and the modulating frequency is 10 Hz.

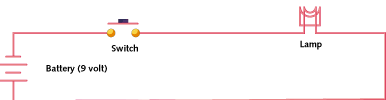


Figure 2. A simple PWM circuit

Most loads, inductive and capacitative alike, require a much higher modulating frequency than 10 Hz. Imagine that our lamp was switched on for five seconds, then off for five seconds, then on again. The duty cycle would still be 50%, but the bulb would appear brightly lit for the first five seconds and off for the next. In order for the bulb to see a voltage of 4.5 volts, the cycle period must be short relative to the load's response time to a change in the switch state. To achieve the desired effect of a dimmer (but always lit) lamp, it is necessary to increase the modulating frequency. The same is true in other applications of PWM. Common modulating frequencies range from 1 kHz to 200 kHz.

-Modulating Frequency

-Duty Cycle

In a nutshell, PWM is a way of digitally encoding analog signal levels. Through the use of high-resolution [counters](http://www.netrino.com/Embedded-Systems/How-To/Counters.php), the duty cycle of a square wave is modulated to encode a specific analog signal level. The PWM signal is still digital because, at any given instant of time, the full DC supply is either fully on or fully off. The voltage or current source is supplied to the analog load by means of a repeating series of on and off pulses. The on-time is the time during which the DC supply is applied to the load, and the off-time is the period during which that supply is switched off. Given a sufficient bandwidth, any analog value can be encoded with PWM.

PWM finds application in a variety of systems. As a concrete example, consider a PWM-controlled brake. To put it simply, a brake is a device that clamps down hard on something. In many brakes, the amount of clamping pressure (or stopping power) is controlled with an analog input signal. The more voltage or current that's applied to the brake, the more pressure the brake will exert.

The output of a PWM controller could be connected to a switch between the supply and the brake. To produce more stopping power, the software need only increase the duty cycle of the PWM output. If a specific amount of braking pressure is desired, measurements would need to be taken to determine the mathematical relationship between duty cycle and pressure. (And the resulting formulae or lookup tables would be tweaked for operating temperature, surface wear, and so on.)

PWM is economical, space saving, and noise immune. And it's now in your bag of tricks. So use it.

## **microprocessor vs microcontroller**

Usually, microprocessor is only the core. It is a CPU, sometime a memory controller, and that's it. You have to use it with other components. You need to add memory, IO, devices, ...  
  
Microcontrolers are targetted toward embedded applications. The point is to try to use the less possible external components. So, generally, microcontrolers have memory (program memory and some RAM), IO ports, counters, UART, ... all integrated into a single chip.

. uControllers are self-contained with RAM, ROM, EEPROM,  
. sometimes with A/D, D/A converters, Serial Ports, etc.  
. Processors are more general purpose, with Memory, converters  
. communication harware external to the uprocessor, and added  
. to the design as needed.

## **can vs lin**

LIN [Local Interconnect Network] is used as an in-vehicle [Automotive] communication and networking serial bus between intelligent sensors and actuators operating at 12 volts. Other auto body electronics include air conditioning systems, doors, seats, column, climate control, switch panel, intelligent wipers, and sunroof actuators. The LIN specification covers the transmission protocol [Physical Layer and the Data Link Layer of LIN], and the transmission medium. The maximum communication speed on a LIN bus is 19200 baud [a 1K termination pull-up resistor is required]. The LIN Bus is a class A protocol operating at a maximum bus speed of 19200 baud over a maximum cable length of 40 meters. LIN nodes can send/receive a full 8 byte command every 10ms (or shorter commands every 5ms). The LIN specification will also handle 2400, and 9600 baud rates, and may be used as a sub-bus [sub-network] for a [CAN bus](http://www.interfacebus.com/Design_Connector_CAN.html) interface. The LIN bus uses a Master/Slave approach, having one Master and one or more Slaves. The LIN bus does not need to resolve bus collisions because only one message is allowed on the bus at a time.

The LIN bus is a small and slow network system that is used as a cheap sub-network of a [CAN](http://en.wikipedia.org/wiki/Controller_Area_Network) bus to integrate intelligent sensor devices or actuators in today’s cars.

Controller–area network (CAN or CAN-bus) is a [vehicle bus](http://en.wikipedia.org/wiki/Vehicle_bus) standard designed to allow [microcontrollers](http://en.wikipedia.org/wiki/Microcontroller) and devices to communicate with each other within a vehicle without a [host computer](http://en.wikipedia.org/wiki/Host_computer).

CAN is a message-based protocol, designed specifically for automotive applications but now also used in other areas such as industrial automation and medical equipment.

A **deadlock** occurs when a series of synchronization objects are held in a preemptive MT system in such a way that no process can move forward.

Simulation is the imitation of another environment.

Emulation is the imitation of an object as another object.

CPU load: CPU usage indicates how hard the CPU is running. The percentage refers to how much of the individual CPU’s capacity is in use.

**Stack:**  Modern [CPUs](http://cplus.about.com/od/termsc/g/cpudefinition.htm) use stack based architecture for handling [function](http://cplus.about.com/od/introductiontoprogramming/g/functiondefn.htm) calls and [parameter](http://cplus.about.com/od/glossar1/g/parametersdefn.htm) handling. When a function is called, the [address](http://cplus.about.com/od/glossar1/g/addressdefn.htm) of the next instruction is pushed onto the stack. When the function exits, the address is popped off the stack and execution continues at that address.

**Mutexes** are typically used to serialise access to a section of re-entrant code that cannot be executed concurrently by more than one thread. A mutex object only allows one thread into a controlled section, forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section.

A **semaphore** restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore), and can signal that they have finished using the resource (incrementing the semaphore)

A **watchdog** timer (or computer operating properly (COP) timer) is a [computer](http://en.wikipedia.org/wiki/Computer) hardware or software [timer](http://en.wikipedia.org/wiki/Timer) that triggers a system [reset](http://en.wikipedia.org/wiki/Reset_(computing)) or other corrective action if the main [program](http://en.wikipedia.org/wiki/Computer_program), due to some fault condition, such as a [hang](http://en.wikipedia.org/wiki/Hang_(computing)), neglects to regularly service the watchdog (writing a "service pulse" to it, also referred to as "kicking the dog", “petting the dog”, "feeding the watchdog"[[1]](http://en.wikipedia.org/wiki/Watchdog_timer#cite_note-0) or "waking the watchdog"). The intention is to bring the system back from the unresponsive state into normal operation.

This page lists C operators in order of *precedence* (highest to lowest). Their *associativity* indicates in what order operators of equal precedence in an expression are applied.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Associativity** |
| () [] . -> ++  -- | Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2) | left-to-right |
| ++  -- +  - !  ~ (*type*) \* & sizeof | Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (change *type*) Dereference Address Determine size in bytes | right-to-left |
| \*  /  % | Multiplication/division/modulus | left-to-right |
| +  - | Addition/subtraction | left-to-right |
| <<  >> | Bitwise shift left, Bitwise shift right | left-to-right |
| <  <= >  >= | Relational less than/less than or equal to Relational greater than/greater than or equal to | left-to-right |
| ==  != | Relational is equal to/is not equal to | left-to-right |
| & | Bitwise AND | left-to-right |
| ^ | Bitwise exclusive OR | left-to-right |
| | | Bitwise inclusive OR | left-to-right |
| && | Logical AND | left-to-right |
| || | Logical OR | left-to-right |
| ?: | Ternary conditional | right-to-left |
| = +=  -= \*=  /= %=  &= ^=  |= <<=  >>= | Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment | right-to-left |
| , | Comma (separate expressions) | left-to-right |
| **Note 1:**  Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer.  **Note 2:**  Postfix increment/decrement have high precedence, but the actual increment or decrement of the operand is delayed (to be accomplished sometime before the statement completes execution). So in the statement **y = x \* z++;**the current value of **z** is used to evaluate the expression (*i.e.,***z++**evaluates to **z**) and **z** only incremented after all else is done. See[**postinc.c**](http://www.difranco.net/cop2220/examples/k&r_examples/postinc.c) for another example. | | |

Serial to Peripheral Interface (SPI) is a hardware/firmware communications protocol developed by Motorola and later adopted by others in the industry. Microwire of National Semiconductor is same as SPI. Sometimes SPI is also called a "four wire" serial bus.

The **Serial Peripheral Interface** or SPI-bus is a simple 4-wire serial communications interface used by many microprocessor/microcontroller peripheral chips that enables the controllers and peripheral devices to communicate each other. Even though it is developed primarily for the communication between host processor and peripherals, a connection of two processors via SPI is just as well possible.

**Measurement** is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize them according to clearly defined rules.

Typical CAN speed: 1 Mbps

**Logical Analyzer:**

It is a versatile tool that can help you with digital hardware debug, design verification and embedded software debug. The logic analyzer is an indispensable tool for engineers who design digital circuits.

Logic analyzers are used for digital measurements involving numerous signals or challenging trigger requirements. In this document, you will learn about logic analyzers and how they work.

A **logic analyzer** is an electronic instrument which displays signals in a digital circuit. A logic analyzer may convert the captured data into timing diagrams, protocol decodes, state machine traces, assembly language, or correlate assembly with source-level software.

**Peripheral Timer:** These chips are often used in computers so that programs can keep track of the elapsed time (e.g. to keep track of the time of day, to generate interrupts and to implement multi-tasking).

**Pragmas** are machine- or operating system-specific by definition, and are usually different for every compiler.

**Software Process Improvement** Capability Determination (SPICE), is a "framework for the assessment of software processes". This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMMI. It models processes to manage, control, guide and monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team.