Embedded Systems Project Management

1. EMBEDDED SYSTEM PROJECT MANAGEMENT

There are two approaches for the embedded system design: (1) *The software life cycle ends and the life cycle for the process of integrating the software into the hardware begins at the time when a system is designed.* (2) *Both cycles concurrently proceed when co-designing a time critical sophisticated system.* The final design, when implemented, gives the targeted embedded system and thus the final product. Therefore, an understanding of the (a) software and hardware designs and integrating both into a system and (b) hardware–software co-designing are important aspects of designing embedded systems.

Refer to an interview of Jean-Louis Brelet in an article *"Exploring Hardware/ Software Co-design with Vertex-II Pro FPGAs" [Xcell Journal, pp. 24-29, Summer issue, 2002]*. A Brelet reply, quoted verbatim, when asked about the expertise required for successful implementation is as follows: *"Software people must understand the nature of hardware design and type of problems encountered by hardware team. They also must understand the possibilities and capabilities of hardware. Likewise hardware team must have a good understanding of software and how the applications operate. Both teams must have a good understanding of each other's language and a willingness to adapt*."

Wayne Wolf, Burak Ozer and Tiehan Lu, Embedded Systems Group, Princeton University recently reported research findings in a paper *"Smart Cameras as Embedded System" by Micro - IEEE Computer, pp48-53, Sept. 2002.* The paper clearly demonstrates that *the selection of the right hardware during hardware design and an understanding of the possibilities and capabilities of hardware during software design is critical* especially for a sophisticated embedded system development.

Recall Section 8 and Table 9 of web material 'Software Engineering Approach in Embedded System Development Process' (SWE). It explained software project management using

Pressman's famous four Ps: people, product, process and project. The embedded systems project management is similar to this. People involved here are *a team of software development, hardware development and system integration engineers*. Table 9 listed these four Ps. Embedded system's project management also means *organizing these four Ps*. Table 1 shows how these four can be organised for the development process of an embedded systems project.

Table 1

Embedded System Project Management's Four Components

Component		Roles	What is not advised
People	Senior Manager	Same as defined in Table 9 of web material on SWE	Same as defined in Table 9
	Project Technical manager or Team leader	(i) Selects software and hardware languages, tools and software development process life-cycle models for the software and hardware development process (ii) Tunes and reorganizes available software and hardware specifications, designs and components and existing processes and (iii) to (v) described in Table 9	Lack of appreciation to implementers perceptions and uncoordinated development
	Implementers	Implements software and hardware development process and integration process by using modeling, source code engineering, testing, simulating, debugging and product verification tools.	Not following the agreed and accepted design and lack of coordination among fellow implementers
	Customer of an embedded system	Specifies the product and its quality requirements and negotiates cost with senior manager (s)	Interference in the development process, changing the product specifications after agreeing to these

End-users	Uses the product within the suggested boundaries	Not using product as per guidelines. For example, driving with ACC control system on an icy road [Section 12.3.]
Product (embedded system)	Same as in Table 9	Lack of correct product specifications
Process	Using layering model (Table 1 of web article on SWE), partition the system into layers between the application and hardware and adapt the process similar to the one given in Table 9	Incorrect partitioning and adaptation of incorrect model
Project	Embedded system project management goal is to create a successful product based on the criteria listed in Table 9.	Improper planning, incorrect effort, estimates, and lack of successful goal-oriented focus, keeping people busy in non-project activities

2. EMBEDDED SYSTEM DESIGN AND CO-DESIGN ISSUES IN SYSTEM DEVELOPMENT PROCESS

2.1 Embedded System Development Process Goal

The goal to be achieved in the last phase of an embedded system development process is to produce a thoroughly *tested and verified system*.

2 Action Plan

Recall the software development process life cycle model. Refer to Section 2 and Table 1 of web material on SWE. It gives us the action plan for an embedded system development process also. Even a simple small-scale embedded system needs a detailed plan.

Defining an action plan is the first step in any system design. Figure 1 shows an action plan for designing a system in its development phase.

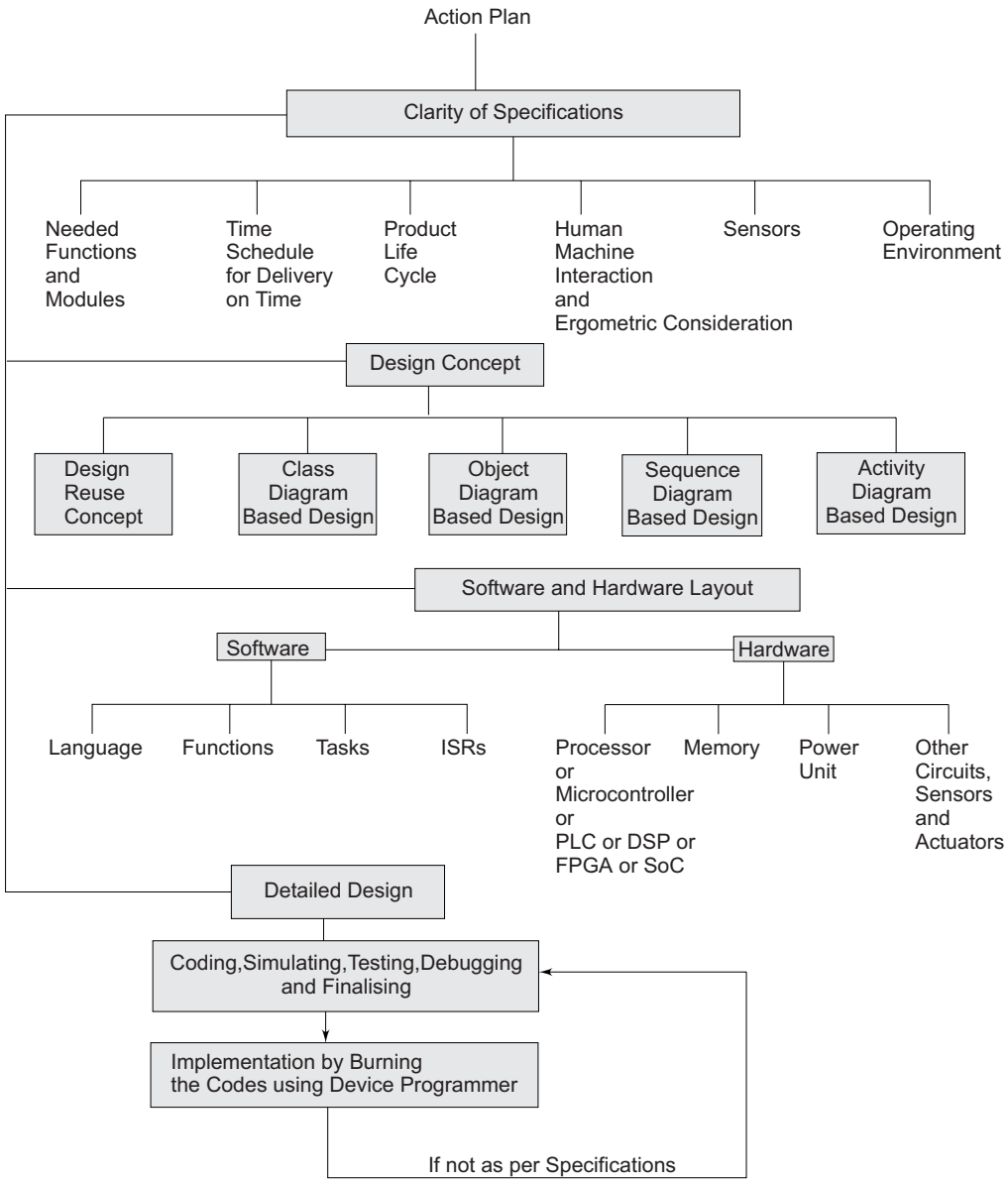

Figure 1: An action plan for designing a system in its development phase

Consider a plan for the development of a small-scale embedded system for an automatic washing machine. [A '*Clothes In Clothes Out'* embedded system!] We consider this example with a view to get an easier understanding of the action plan needed in any embedded system development. For the user, it is a simple clothes-in and clothes-out system. For an embedded system designer, the things are not that soft! For the system designer, it is '*Bytes In Bytes Out*' system in place of the Clothes In-Clothes Out machine. The designer has to design according to a full-fledged action plan. The following are the steps needed for the system development process.

2.3 Complete Specifications and System Requirements

The first step is to have ***complete clarity about the specifications for the needed system.***. Specifications given in the first column of Table 2 have to be completely to the development team.

Table 2

Specifications of the System

System Specifications	Explanation
Product functions and tasks	Understanding the functions and user tasks needed for the system are essential. Consider an embedded system design for a robot. What are the functions expected from it? What are the degrees of freedom required to move of all its parts (waist, shoulder, elbows, hand and fingers)? What are the tasks it should accomplish? Is it smart?
Delivery Time Schedule	Time schedules for delivery are important. Tight delivery schedule will force high-speed rapid development model after suitably adapting linear sequential development process or force the use of a combination of the object orientation, 4^{th} generation tools and readily available hardware design, employing general-purpose processor.
Product Life-Cycle	Life cycle of the product. When product cycle is short, it will need frequent design changes by the developer team.
Load on System	System load is important specification. System can fail on over-loading. For example, consider automatic washing machine embedded system.

	Load can be small, medium and full tank load of cloths. The answer will have bearing on the motor capacity, time and level of incoming water during washing and rinsing cycles. Another example is that hardware and processor for processing an image, a video clip and real time video will have different loads.
Human-Machine Interaction	Specifications are needed that answer the questions given below. What will be the human - machine interactions? This has a bearing on the plan for the keypad inputs and display outputs. What and how are the displays to be specified? The answer has a bearing on the interface circuit and the program for the displays. For example, refer to the keypad for remote-controller for a TV.
Operating Environment	Operating temperatures, humidity and environment parameters specifications are essential. A system may fail on the mountains or may fail in high temperature in the vicinity.
Sensors	Sensor specifications for sensitivity, precession, resolution and accuracy are essential for designing as per the requirement. Video-conferencing image sensor and video images sensor will generate system input in different pixel resolutions, formats and rates.
Power Requirement and Environment	A system having greater load will need greater power requirement. Battery-dependent system will need power management solutions by clever power saving software design and hardware design. . A system operating under continuous power availability condition and interrupted power availability will have different specifications. The design is simpler in case of the former as there is no memory needed to frequently save the system status.
System Cost	Maximum bearable cost must be specified to decide whether a project is acceptable to a development team and the amount of efforts to be made by the team.

System specifications for the following must be prepared before starting the design process. (i) Product functions and tasks (ii) Delivery Time Schedule (iii) Product Life-Cycle (iv) Load on System (v) Human-Machine Interaction (vi) Operating Environment (vii) Sensors (viii) Power Requirement and Environment (viii) System Cost.

2.4 Conceptual Design

The second step is developing a ***conceptual design*** of the system. The question to be addressed is as follows: What will be the model of the system development process? Let us recall Section 6.5. A conceptual design model can be developed, using UML approach. A conceptual design can use UML 'User Diagram', 'Object Diagram', 'Sequence Diagram', 'State Diagram', 'Class Diagram' and 'Activity Diagram'. A conceptual design helps in developing the application software and hardware structure and layout.

UML Class diagram, 'User Diagram', 'Object Diagram', 'Sequence Diagram', 'State Diagram' help in developing a conceptual design and later on get the structure and layout of the application software and hardware.

2.5 Software and Hardware Layout Design

The third step is the development of a ***software and hardware structure and layout*** of the system. There can be two approaches.

(1) ***Independent Design Approach Followed by Integration***: Software life cycle ends and life cycle for the process of integrating into the hardware starts at the time when the system is designed.

(2) ***Concurrent Co-design approach***: Both cycles concurrently proceed when co-designing a time critical sophisticated system.

When developing a software layout, we answer the following question. What will be the software modules needed? The exemplary three modules in an automatic washing machine system may be as follows:

(i) A *software module*, which takes the user inputs and provides the human–machine interaction with the machine using the LED outputs. Examples of the functions *for human-machine interactions* are as follows: (a) Current default user settings on LEDs are shown at the start. Human–machine interaction can be made smart. [The default settings can be according to the user's previous preferences. Woolen

preference in winter and cotton preference in summer!] (b) A key pressing can cyclically set the clothes load (if that is not automatic) among the three possibilities. An LED shows the selection at an instant. The user stops further pressing when the desired load is set. The processor, if it does not find any further pressing at the corresponding port, loads an input in EEPROM memory. (c) Another key pressing can cyclically set the clothes type. It can be one among the three possibilities: wool, polyester and cotton. The module also stores this user directive.

(ii) When there is a user directive to start, which signals after a key is pressed or after a remote switch signal or even an Internet message (in Internet compliant systems), there has to be another *software module, which initiates the cycles and schedules each cycle-time. It* initiates the cycles (for wash, rinse and dry). (a) This has to be done after ensuring that the clothes insertion and picking door are closed. (b) The power supply output has to be according to need. (c) The power sources to the motor and solenoid valves have to be available according to need. (d) There has to be a reset of the machine status in EEPROM, if the need is to initiate the cycles from the beginning. (e) If the machine status indicates left-over cycles and functions due to some interruption during the previously run cycles in the machine, then it decides the alternative actions to be taken.

(iii) Another *module* is to *start a wash cycle*. Exemplary functions will be as follows: (a) Let the water inlet be on. (b) The water inlet must be off after the level sensors arrays output shows that the water is there up to the needed level for the given clothes load. The water inlet should also be off after a preset time to have a watchdog action for the level sensor inputs. (c) Start the motor in slow or medium spinning mode.

Hardware and Software Implementation Tools Specifications

The foremost question before developing a layout and a detailed design is as follows: What are the elements (hardware and software requirements) of the development process? The answer can

be given by selecting the elements needed to fulfill the specifications mentioned in Table 12.1. For example, for an automatic washing machine the elements are as follows: (a) Motor Tank. (b) Water-level sensor-array. (c) Power sources for the motor and electronic circuit. (d) General-purpose or embedded processor or microcontroller. For example, an eight-bit *microcontroller* with adequate internal ROM, EEPROM and RAM, timers, interrupt handlers, peripheral serial or ports controllern. (e) An *ergonomically designed* key array and *LEDs.* Input keys are for user directives. LEDs indicate machine status and the cycles completed or remaining. (g) Interface circuits. (h) *Power source* 220V and solenoid valves for water inlet and outlet.

We may refer to different exemplary systems hardware units and refer to Sections 1.2 and 1.3 that described hardware elements. These guide us to take a decision on the **hardware requirements specifications**.

There are two design approaches. The first is an independent design, which follows system integration. This approach is that software life cycle ends and life cycle for the process of integrating into the hardware starts when a system is designed. Another approach needed for a sophisticated embedded system is a concurrent co-design approach. Both cycles concurrently proceed when co-designing a time critical sophisticated system. There are a number of software and hardware tools to implement the designed system easily with simple efforts.

2.6 Detailed Design

The fourth step is the ***detailed design*** of the codes and the target system by first selecting the processor and memory. Then decide about the functions that are to be implemented in the hardware and in the software. Software and hardware layout later helps in the detailed design for the implementation of detailed software codes and the circuit for obtaining the target system.

2.7 Implementation Tools

We may refer to Table 3, which lists hardware tools during system development process.

Table 3

Hardware tools for the detailed design

Hardware Tools	Application
Emulator	A circuit for emulating the target system that remains independent of a particular targeted system and processor, usable during the development phase for most of the target systems that will incorporate a particular microcontroller chip. It provides great flexibility and ease for developing various applications on a single system in place of multiple targeted systems. It works independently as well as by connecting to the PC through a serial link.
In-Circuit Emulator	An emulator circuit that also emulates the target processor circuit and that must connect to the PC through a serial link and to the target system processor or microcontroller using a ribbon cable. Emulates various versions of a microcontroller family during development phase.
Logic Analysers	A power tool to collect through its multiple input lines (say, 24 or 48) from the buses, ports, etc. many bus transactions (about 128 or more). It displays these on the monitor (screen) to debug real-time triggering conditions. It helps in sequentially finding the signals as the instructions execute.
Device Programmer @	A programming system for a device, which may be a PROM or EPROM chip or a unit in a microcontroller or PLA, GAL or PLC. A device inserted into a socket (at the device programmer circuit) programs on transferring the bytes for each address using a software tool at the computer and interconnecting the computer with this circuit. Device program needs in the output the locator output records. These outputs

must reflect the final design or a boot program plus the compressed record, which the processor decompresses before the embedded system processor starts execution.

[@] Refer detail later in Section 13.4.1

Hardware tools for hardware design and system integration are emulator and In-Circuit Emulator. Software tools are simulators, editors, compilers, assemblers, source code engineering tool, profiler (for viewing time spent at each function or set of instructions), memory scope, stethoscope-like view of code execution, memory and code coverage scope.

2.8 Testing

We divide the problem into small parts so that testing is easy at the initial stages. Define inputs and outputs from each stage clearly and identify and make the data flow graphs (DFGs) (Section 6.1).

Recall Sections 7 and 8 of SWE. A test technique is testing by *calling the interrupt service routines*. The use of an *assert* macro is another important test technique. For example, consider a command, "assert (*pPointer* != NULL);". When the *pPointer* becomes NULL, the program will halt. We insert the codes in the program that check whether a condition or a parameter actually turns true or false. If it turns false, the program stops. We can use the assert macro at different critical places in the application program.

Testing and debugging have to be there at each stage as well as at the final stage when the modules are put together. Religiously follow the rule, *wrong until confirmed right*, for testing and debugging. Documentation in detail for each stage is also a necessity.

There are software tools for assembly language programming, high level language programming, RTOS, debugging and system integration tools for decision on s***oftware requirements specifications*.**