MICROPROCESSOR – BASED PRODUCT DEVELOPMENT

Dr. David L. Danner, P.E.

IDEAMATICS, Inc.

Abstract

The six steps required to develop a microprocessor-based product – conception and evaluation, planning, design, implementation, documentation, and field test – are discussed, using the INTR-ACT System as an example. Specific techniques to aid in microprocessor-based product development are identified.

Smooth implementation of the design requires skilled hardware engineering and flexible software design. Detailed specifications and specialized test equipment allow for design modifications and continuous monitoring of progress during testing. Development tools such as cross-assemblers, emulators, CAD systems, prototype breadboards, and various electronic test equipment aid in the implementation effort.

After implementation and testing is completed, an extensive field test or "soak in" period is conducted.

Introduction

Microprocessor-based products have appeared in increasing numbers since their introduction; in 1986, we see them everywhere, doing everything from calculating exposure in 35-mm cameras to controlling digital timepieces that perform complex functions. Microprocessors are widely used as telecommunications controllers. This paper traces the development of a micro-based controller from initial concept through hardware/software design, programming and testing, prototyping, fabrication, and final product introduction. These development steps are applicable to any microprocessor-based product implementation. Six steps are requires: conception and evaluation, planning, design, implementation, documentation, and field test.

Product Conception and Evaluation

There are two ways, generally, in which a technically complex product is conceived:

1) someone sees a need or a market for a particular product and begins to investigate the feasibility of creating it, and
2) someone has developed a neat, innovative way of doing something and looks for a practical application for it.

Both of these approaches can and do produce viable products, although the second may lead the developer into the trap of becoming so enamored of innovative technology that the lack of a market for the product is never noticed or is ignored.

When a product is conceived, the development team must answer the following questions before proceeding:

1) Is there a market for the product? If so, what is the nature of the market? What options and features must the product have?
2) Is it technologically feasible to produce the product?
3) What will the cost of developing the product be? Will the finished product be reasonably priced?

4) Is the technology to product the product readily available, or will research and development be required?
5) Is the market for the product sufficiently large and long-term to defray development and production costs and to produce a reasonable profit?

If the product is deemed both feasible and profitable, the decision should be made to plan its design, development, testing and marketing. This decision to begin to plan is the first of many "go/no-go" decisions that must be made before a product can be brought to market.

Project Planning

Planning must always be the first step in product development. Without a detailed plan, including a schedule, functional specifications, and a budget, product development can fly off into never-never land, leaving the developer with an expensive mistake rather than a quality product. The elements of the initial plan should include the following:

1) Functional specifications for the product, laying out what it should do but not how it is to do it (this cannot reasonably be established until the design phase). The specified functions must correspond to the functions used to evaluate the product's feasibility – it is all too easy to allow specifications to "creep" away from the evaluated concept. This is when the marketing staff should list their recommended product requirements. In fact, development will proceed much more smoothly if product requirements are established during this phase of the development cycle and then fixed.
2) A detailed schedule for product implementation, laying out both concurrent tasks (i.e., those that can be done simultaneously by the hardware and software design teams) and those activities that must be performed sequentially. The schedule should include test dates.
3) A budget for project completion, including all design, implementation, and prototype manufacturing costs. Post-development costs such as marketing, inventory, and product approval costs are better addressed after a product has been developed and proved feasible. Planning in the product development cycle is limited to those actions necessary to obtain a workable piece of equipment. The budget should be written with the understanding, particularly if research and development is required, that false starts will be made and work will have to be modified and even thrown away at times before development has ended.

The project plan must be a written document. Specification of the functional requirements must be of sufficient detail that hardware and software design can proceed without further required discussion with the planners. The regimen of putting the "nice to haves" and "it should dos" in writing forces a considered evaluation of the complexity of the product and the scope of the development effort. It also establishes a clear baseline, so that additional requirements are readily highlighted as going beyond the original concept.

After the planning has been completed, the developers must ask themselves whether they should continue; this is the second go/no-go point in product development, and it should be taken seriously –it is the last possible stopping point before a lot of money will be spent. Developers must ask themselves if the market is still there for the product; if it will be there when the product is completed; is it is sufficient to recoup the costs laid out in the budget; if the product can be manufactured at a reasonable cost; and if, based on the specifications and budget, development of the product is both feasible and reasonable. If the decision is made to go ahead, the development team moves into the design phase.

Product Design

Designing a microprocessor-based product generally involves two separate groups working concurrently: the hardware design team and the software design team. Both teams, working from the functional specifications, begin their design tasks simultaneously, so it is very important they talk to one another…and talk…and talk. Without communications during this phase, the designs will go their separate ways, both responsive to the functional specifications but unworkable in combination.

In the remainder of this paper, I will step through the design and implementation of a microprocessor-based system, using as my example the INTR-ACT System, whose hardware has developed by Electronic Telecommunications, Inc. (ETC), of Waukesha, Wisconsin, and whose software was developed by IDEAMATICS, Inc., of Washington, D.C. The INTR-ACT System is presently being tested by the Candle Corporation of Los Angeles, California. It is a computerized, flexible and interactive voice announcement system with user-selectable functions that are invoked by the use of codes keyed in via the telephone touch pad. The INTR-ACT System is used to deliver live voice messages simultaneously to up to 3,000 callers. Message recording is an analog-to-digital conversion; no moving parts are required.

Designing and Redesigning

The design is developed from the functional specifications by both teams and is placed in a system design document that is constantly referred to throughout the project. The design document is not a Bible, however; both hardware and software teams will find themselves modifying, improving, and changing the design, and even scrapping portions of it, well into the implementation phase. The design document is a working vehicle for recording the current status of the product concept. It is never complete until turned over for volume production.

Some of the changes will be forced; for example, ETC started out using and Intergraph CAD/CAM system to design a double-layered circuit board for the line card controller, was forced to redesign the card as a primary circuit board with two daughter boards, and finally had to incorporate eight wiring layers within the primary board.

Some of the changes will be voluntary; someone simply comes up with a better, more efficient, less expensive way to do things. For example, the initial design of the INTR-ACT System did not provide for the reloading of default tables by an on-site maintenance person without a computer. The software was modified to recognize a new configuration of existing switches to force a table clear function.

The team should be considering the implications of design elements for manufacture of the product, and some changes may be made as a result. Changes made in the course of developing the design should be expected, and even encouraged, by the project manager. A good design, done early, prevents extensive (and thus expensive) patching and fixes later on in the project. (It should also be realized, however, that even with the best design some patches will have to be applied.)

Developing the Test Plan

The test plan is a vital part of the design phase of product development. Even at this early point, both development teams should be thinking about how the product is to be tested, and they should be building self-diagnostics into their designs. Development of a preliminary test plan will encourage the teams to think about testing and will also give the project manager an idea of the tent and likely cost of testing.

The test plan must address all boundary conditions as well as anticipated operating conditions. For example, the INTR-ACT system must communicate with a host computer via RS-232 protocol. The

test plan, by proper preparation, identified the need to resign the software and the hardware to allow for connection directly or through a modem.

Design Evaluation

When the design has been completed, another go/no-go point has been reached. The development team must examine the design and ask the following questions: does the product, as designed, meet the objective with which the team started out? Does the product satisfy the requirements in the functional specifications? Will it be possible to produce the product as designed? If the answer to any of these questions is no, a hard decision must be made –either to stop product development, or to cycle back to the evaluation or the design phase and begin again. If the answer to all the questions is yes, then the software and hardware teams can begin product implementation.

Another major consideration for the design evaluation is the availability of support tools. What cross-assemblers are available for software development? What emulator is needed to prototype the hardware/software interface? Do any customized, specialized analysis tools need to be developed such as a memory access display module or other data analyzer? Any one of these requirements in itself might lead to a miniature sub-development effort.

Product Implementation

When the teams have reached the point of an acceptable, approved design, they are ready to start their hands-on implementation work. Implementation can be summarized very briefly: do it, keep talking, and keep testing. There are some helpful techniques for getting through the implantation phase, as follows.

Prototypes

The beginning of product implementation may be a little tentative – both teams are feeling their way into the project and into a good working relationship. A good approach to encouraging the teams to move ahead is to have the hardware design team construct prototypes. Prototypes can be very rough; for example, the first RS-232-C interface for the INTR-ACT System was a breadboard hand-wired to fit into the back plane of an existing software development system.

What prototypes do is give both teams something to work with, to test against, and to focus on. They are extremely helpful, and they increase the speed and efficiency of implementation by identifying dead ends and suggesting workable solutions prior to the integration of the total product.

As the teams work, it is again vital that they communicate with one another. In the case of the INTR-ACT System development, in which the hardware team was based in Wisconsin, the software team in Washington, D.C., and the client in California, telephone bills were substantial, and programmers became long-distance commuters. Without such communication, missteps would have occurred that lost time and wasted work far in excess of the cost of proper communication.

Developing the Client Pleasers

As the teams develop the product, they should have in the back of their minds, and should discuss, what I call “sugar pills.” A sugar pill, of little significance to the design, is something to keep the client/user happy by showing him that his machine is actually working. Blinking status lights, messages that convey information but aren’t really necessary, and other little frills that show the machine doing something are important to the client, though they are not essential to the engineers. (A prime example

of a sugar pill is provided by a digital watch; it will do something every second to reassure its owner that it is working, such as flashing the colon.)

Ongoing Testing

Testing should be ongoing throughout implementation. Leaving large-scale testing until after the teams have finished will almost ensure massive redevelopment and reworking of both hardware and software. Ongoing testing takes a little more time, but the payoffs are large. The failure to introduce quality products into the marketplace can often be traced back to inefficient testing or the lack of a comprehensive and exhaustive test plan.

Freezing the Design

Throughout the design phase, and for a while during the implementation phase, the project manager should encourage the scrutiny and alteration of the product design. However, once implementation has commenced, design changes must be carefully analyzed before they are accepted and introduced into the product. As implementation proceeds, the criteria for inclusion become increasingly stringent.

When all elements of the product are first available for integration, the design must be frozen. No changes should occur after freezing the design. Not even brilliant ideas may be allowed to impinge on a nearly-developed product; these should not, of course, be discarded, but rather should be saved for the new, improved Version II. The manager should discuss his decision to freeze the design with the team leaders, who then convey it to their teams.

Development Techniques

A number of techniques have proved helpful in successfully developing microprocessor-based products; I discuss these briefly in the following paragraphs. They include breaking tasks up for efficiency, building flexibility into the system wherever possible, planning in advance for hardware changes, and testing based on a well-designed schedule and test plan.

Breaking Up Tasks for Efficiency: A software/hardware design effort should be broken apart, for the sake of efficiency, into series of like tasks that can be done concurrently. For example, physical layout of circuits on boards (i.e., taping artwork) is not dependent upon the software. Likewise, specification of internal software tables (residing in RAM) is essentially hardware independent. Therefore, not only can various components be implemented simultaneously (e.g., various printed circuit cards), but also elements of the hardware and software can be initialized in a non-sequential fashion.

This process can be compared with producing a film: all the location shots, regardless of their place in the plot, are shot at one time; all the interiors are shot at one time. It is the editor, later, who makes a logical sequence out of disparate pieces. Microprocessor-based product development should be done in exactly the same way.

Building in Flexibility: the more flexible the system can be made, the easier it is to deal with problems later, and the easier it is to improve and modify the system for later versions without completely redesigning it and reprogramming it. Flexibility is increased by using the following techniques.

1) Control the product with software (that can be changed) rather than hardware (that is a fixed entity that costs more to change).
2) Design table-driven software. This adds flexibility to the product (at the cost of increased program complexity). Tables are modified more easily than hard-coded assembly language programs.
3) Use distributed processing wherever possible (for example, use separate processors to perform two functions, rather than one processor to perform both). Future changes to one function will not require the redesign of both functions. In addition, the increased power of two processors may alleviate processing contention or provide future expansion through using excess capacity.
4) Plan for hardware changes by building in excess capacity; don't develop initially to the maximum of the available hardware. For example, if 32 input/output bits are available and 32 are being used, a poor design was produced; if 32 input/output bits are available, and 29 are being used, the design is good because the flexibility of additional input/output processing is available if needed, without scrapping existing inventory.

Testing: Good testing for microprocessor-based products has the following characteristics.

1) Diagnostics are built into the software and, where possible, hardware. These eliminate a lot of guesswork and make testing to isolate problems considerably easier.
2) Individual portions of software and hardware are tested initially, rather than large chunks of them later. Again, testing is supposed to isolate problems, and the smaller the unit that can be tested, the easier it is to isolate and fix errors.
3) The test starting points must be fixed, so that one thing at a time is being tested. There are three elements to a test: the hardware, the software, and the test setup (which includes test data, test emulation equipment, test procedures, and user test-support equipment). Only one element can be tested at a time. If new software is being tested, there should be no new elements introduced into the test setup; the same goes for hardware. If multiple elements are tested and something fails, what is it? Software? Hardware? Test setup? Only one new and unknown quantity should be introduced at any given time. Not testing from such a baseline is a common failing that introduces and propagates errors.

Documentation

Good documentation is vital to a product's success. Both the hardware and software must be documented, and it is essential that the documentation is clear, accurate, and readable. Documentation is the developer's conversation with the user. All too often, good systems are "shot in the foot" by careless, thrown-together documentation. To prevent this, the following things should be done:

1) Documentation should be an ongoing process rather than something done hurriedly after product implementation (incidentally, forcing early documentation actually helps during implementation, because it forces programmers and engineers to think about the product and its use).
2) Documentation should be prepared by technical writers working closely with programmers and engineers, and the technical writers should be brought into the project early enough that they understand the product and its functions. The technical writers are experts at what they do, and they should be listened to – they can often offer good design and implementation suggestions aimed at making the product easier to use and more understandable to a user.

3) Documentation should be comprehensive, covering all aspects of both the hardware and the software, and should be clearly written in good English rather than technical jargon.
4) Documentation should be completed before the product is sent out for a "soak in" field test.

Field Testing

A thorough, comprehensive, though field test is the next step in product development. This testing should be done by a user familiar with the system and competent enough to use it, but not one who has been involved in the implementation and may be wedded to the implemented version of the product.

The testing user should be provided with all relevant documentation and with assistance in setting up the product. Thereafter, he should be allowed to test without the design teams hanging over his shoulder. The elements for a successful field test are discussed below.

1) Don't thrash. Problems should not be fixed as they occur; rather, the development teams should bite their tongues and wait until the test period is ended, and then develop a coherent and comprehensive approach to fixing errors in the system.
2) Exercise strict configuration management over the test. The hardware used, the software used, the documentation provided, and the application-specific data (e.g., tables) must all be known quantities. The user may not be allowed, for example, to attempt his own "little fixes" or problems. The hardware being tested must be the hardware that will be sold; again, no "little substitutions" can be allowed.

After the Test

After the field test, and after problems have been fixed and the product has been retested, another go/no-go point has been reached. At this time the developers will have a good idea of how their product performs.

They must go back to their original product specifications and market study and decide whether what they have can be sold 1) at all, and 2) at a reasonable profit. If it is not salable, the developers can cycle back to any point in the process that is appropriate, or they can cease product development, eat their losses, and console themselves that they have learned something from the experience. However, if the product has been designed and implemented with care, the decision will probably be made to go forward.

Continuing with Product in Hand

Once a product is in hand, tested and working, the software and hardware teams have completed their job (and can begin working on Version II, incorporating all the improvements and interesting expansions of capability that had to be left out of Version I). The managers of the development effort have more to do. Briefly, the final steps before a product is brought to market include:

1) Marketing – a process that must begin well before the product is in production.
2) Obtaining necessary approvals – e.g., Federal Communications Commission registration for a telecommunications device.
3) Beginning production tooling – e.g., preparing wire lists and stock component inventories, establishing assembly lines, etc.

Conclusions

Developing microprocessor-based products requires up-front studies to determine the sale-ability of the product and the feasibility of producing it; these studies must be rigorous, the developer must take care not to become so enamored of an idea that negative studies are ignored. If evaluation is positive, though, planning, design, implementation, testing, and field testing are required to bring the product to production.

Planning must be thorough, and the results of the planning phase –functional specifications, a budget, and a schedule – must be written down in detail. The decision to proceed must be re-evaluated following the planning phase.

Throughout the design and implementation phases, communication between the hardware and software design teams and careful, thorough, ongoing testing are the most certain guarantees of producing a quality product that meets the specifications established in the beginning of the project. Care must be taken to build as much flexibility as possible into the product, and implementation must proceed efficiently, using the technique of segmenting activities so that both teams can work concurrently (rather than sequentially) on elements of the product.

The product should be documented, as well as tested, on an ongoing basis. The development teams must realize that good documentation is essential to a successful product, and must devote considerable time and effort to developing it. Documentation should be completed before field test, which should be performed by a knowledgeable and capable user.

The INTR-ACT System developed by Electronic TeleCommunications, Inc., and IDEAMATICS, Inc., is exemplary of the application of these principles to the development of a complex, sophisticated microprocessor-based product. The successful introduction of the INTR-ACT System at the end of 1985 is a monument to the proper cooperation exhibited by the entire product development team.