8 TOP DOWN DESIGN WITH UML

8.1 Introduction

Microprocessors can be programmed to perform a wide range of tasks. In fact, they must be programmed before they can do anything at all! As they are fabricated using Large Scale Integration (LSI) technology, the unit cost of each processor is very low. However, the development of systems using microprocessor tends to be high. Proper design techniques would help reduce this developmental cost. That is the aim of this chapter.

8.1.1 Why microprocessor-based?

The microprocessor offers opportunities for products and product features which are not achievable by other means. It is difficult to achieve the range of features found in today's electronic devices. For example it would be almost impossible and not economical to design a Compact Disk player using logic gates alone.

8.1.2 Implications of Incorporating a Microprocessor

Testing the microprocessor based system is a huge hurdle one needs to overcome. Firstly, software needs to be extensively and thoroughly tested. If the underlying fault arises because of the operating system, changing it is often impossible. This is one of the problems that could arise from software testing.

Secondly, the hardware of the system is almost as difficult to test as the software. This is because hardware interacts with software and it might be difficult to isolate whether an error arises from hardware instead of software.

As microprocessors only deal with digital data, an interface must exist between the microprocessor and its peripheral devices to convert data, should the format be different.

8.2 Product Development

Trends in electronics industry

Even though products tend to have a shorter product life, every new version is expected to have a higher performance over price ratio due to the intense competition in the market.

As a result, manufacturers need to

1. shorten product design cycle using improved design methodologies and

2. invest on tools & resources to improve and increase product features.

8.3 Product Design and Development Activities

Product design is a human process. Design is a process which involves communication, creativity, negotiation and agreement.

System designers do not work in isolation. There is a strong temptation to sit down and produce the system at the earliest possible moment. This is the cause of much of the problems in the software engineering industry. The hardware designers and the software designers need to understand each other so that the final product meets requirements of the client. Very often in the design and production process, the communication, negotiation and agreement aspects of a design process are left out.

A notation which is clear, consistent, and which can be used to communicate within a system development team, and with the clients and other third-parties which the team needs to deal with, will certainly come in useful. The following diagram gives the approach we will take in this chapter.

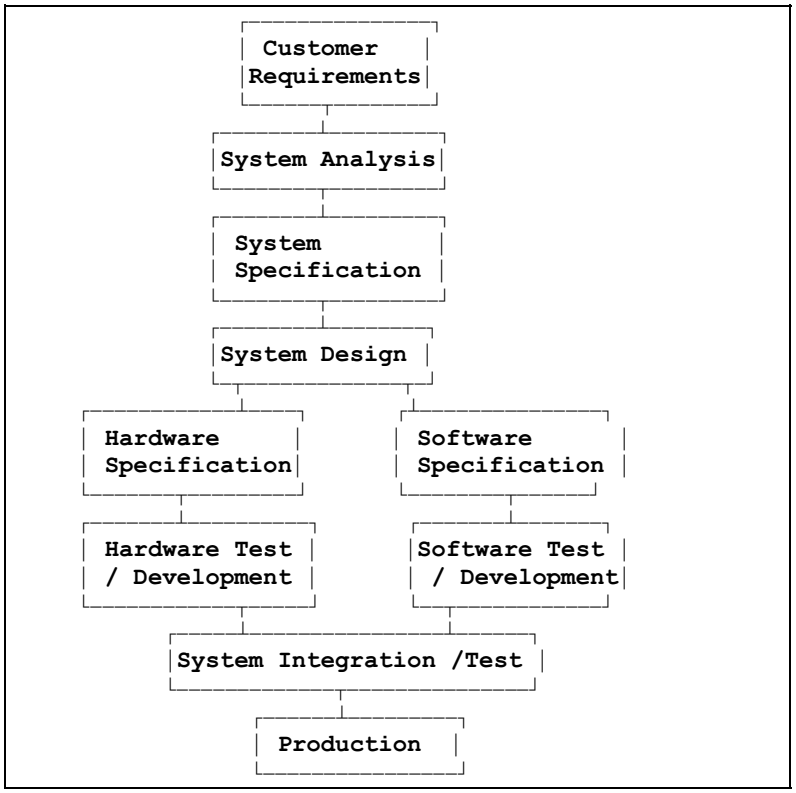

Fig 8.1 Product Design & Development Activities

Systems were constructed using a “waterfall” approach traditionally. As water flows down from the top, this approach starts off by having clients formally agree on a requirements document. The designers would then come up with a design which would be further

agreed by the clients. The system would finally be implemented, and what follows would generally be an endless process of maintenance.

Modern ideas move away from this. Many system developments use instead the iterative method. The method consists of developmental cycles with each cycle making up of analysis, design and implementation. Each subsequent cycle build on earlier cycles. One such method is the Unified Modeling Language (UML). It is a language for specifying, visualizing, constructing, and documenting the features of systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML represents an important part of system development process. UML uses mostly graphical notations to express the design of system projects. Using UML helps project teams communicate, explore potential designs, and validate the architectural design of systems. In our product design, we will use UML for System Analysis and Design.

8.4 Product Requirement

The first stage in the both the iterative method and the “waterfall” method is requirements gathering. The difference between the two is that the iterative method is only interested in capturing the most important features in the beginning and allowing subsequent cycles to handle the less important ones; the waterfall method however starts off with the intention of obtaining all the features of the system. In any case, this initial description of the product's intended functions is obtained from the customer. One is only interested in how the system will interact with its intended user and its environment at this stage.

8.4.1 Need for Product Requirement

There are two reasons why one needs the requirement gathering phase:

Avoid 'creeping featurism'

Because of the choices available to the designer when developing microprocessor based products, there is always the desire to interfere continually with product specification during development. This can significantly alter the cost and time required for product development

Avoid 'missing features'

The worst that could happen are that some required features were missed out, and when noticed, it is either too late or a major redesign is required.

8.4.2 Goals and Constraints

These are additional information from the customer apart from product requirement description.

Goals
These are aspects of customer requirements which guide the system designer where freedom of choice exists. For example, Hardware cost of the motor interface be minimized

Constraints
Aspects of customer requirement which limit the freedom of choice of the system designer. For example, The product should include a particular type of component.

Constraints narrow the scope of the project. Goals are customer preferences to be considered so that the product can function withing the given constraints.

- Goals and constraints may affect the same design feature
- There should not be any constraints that restrict the freedom of choice of the designer unnecessarily
- Avoid contradiction in the list of goals

Types of Constraints

AREA OF CONSTRAINTS	EXAMPLES
Microprocessor	Particular type specified by customer
Other hardware	Some or all of additional hardware specified
Field-repairable	PCBs must be plug-in
Power Supply	Must run from batteries
Environment	Must be insensitive to radio interference
Size	Must fit into particular box
Unit Component Cost	µP and interfaces must cost less than a particular limit
Development Cost	Amount available for supporting equipment, consultants, etc
Development Time	Prototype available for demo at an exhibition

So in the above example, a constraint would be the use of a waterproof chassis.

8.5 System Analysis

System analysis consists of activities that include examination by the system designer of the initial customer requirements. Some considerations when performing system analysis are:

Subsystem Identification
From the requirements, we try to identify hardware components or sub-systems in the product.

Dynamic modeling
Dynamic modeling tries to capture how the parts of the system behave and how they interact between each other.

Feasibility studies and/or simulation
If system designer simply does not know whether the product can be designed to specifications, the use of feasibility studies and/or simulation can remove much of the uncertainty without the expense of designing to the prototype stage.

8.5.1 Sub-system Identification

In this stage, we need to identify the components that would make up the whole system. These are components that perform a function and does not have to be extensively programmed by the designer. The advantage of subsystems are that they are complete products whose behaviour and cost are known. Typical examples of sub-systems in embedded systems are the microprocessor, LCD, keypads and stepper motors.

8.5.2 Dynamic Modeling

A task is a part of the overall product function which is to be provided by a suitably programmed microprocessor system. We need to obtain from the customer requirement a statement of what the system has to do.

Then we use dynamic models to analyse the behavior of systems. Dynamic modeling is similar to a form of story telling. The approach we are going to adopt is to look at the functions of the system we are analyzing, thinking through them in terms of scenarios and then see how these scenarios affect each of the individual sub-systems. The output of this phase is the production of interaction and use case diagrams. Detailed examples are given later.

8.5.3 Feasibility Studies/Simulation

At the analysis stage, it is important to have an idea if a particular algorithm has a chance of working. This should be done without too much expense or hardware construction.

8.5.3.2 Feasibility studies

These are special-purpose activities, aimed to answer a particular question or group of questions about an aspect of the customer requirement.

Some questions to ask are: can it meet project constraints, is the performance of the hardware system sufficient, or does the algorithm work correctly.

There may be some hardware or software design, but should be easily done, as compared to building a full prototype of the proposed product. The prototype is designed to answer most of the remaining questions about the product.

8.5.3.4 Simulation

These are means of exploring, rapidly and easily, various alternative answers to more general questions. Very often they are done in software.

- Simulation is used to build system which in some ways, behaves like the actual product.
- Various versions of the proposed system can tried out without extensive redesign.
- Principal use of simulation in µp-based product development is for evaluating competing algorithms. If these are complex, we may not be able to evaluate them based on logic or on manual calculations.

8.6 System Specification

This is produced as a result of system analysis, by the designer. It is the overall controlling document in a project. The UML diagrams that are produced far, would form part of this documentation. In this documentation, you would find the following information:

- Specification of the product function.
- A complete description of what the system should do
- The performance requirements it must meet
- Specific details of the operator/system interaction
- Specification of the system's interface with the external environment
- Procedures for error handling and diagnostics
- Constraints on the design and on the development project
- Goals to aim for in the design

The specification of the required behavior should include:

- Identification of the microprocessor tasks
- Description of how the product as a whole is to interact with its environment, particularly for products with a substantial human interaction

A useful way of providing the specification on how the product is to interact with its environment is to produce a USER'S MANUAL for the yet-to-be designed product.

The start of the system design phase of a project marks an important transition for design engineers as they must make decisions without a outside help.

8.6.1 Decision making in system design

This involves making important decisions on:

- Build or buy ready made parts
- choice of microprocessor
- Hardware or software

8.6.2 Build or Buy

To build or design from the scratch (i.e. build) may

- not work first time
- need modification

Commercially available controller/microprocessor boards or cards having been proven would not suffer from those problems above.

Another advantage is that cost and performance is known with a high degree of confidence at the beginning of the project as compared to overestimating our own abilities to design and build the required part. Therefore, it is good practice for a beginner to use such commercially available systems as much as possible in early design.

8.6.3 Choice of Microprocessor

Getting the right processor for a new project can give a strong market advantage, but using the wrong chip can weaken an architecture, prolong a design cycle or cripple a product. There are considerations for first timers and those already using a processor. The choice is always important and there are many alternatives. Cost is the overriding factor. But there are one time costs like development and recurring production costs. Some factors to consider are:

- Software availability. This refers to the code used in your products. Is there a large source for code, or old code be reused? Software comprises up to 70% of one time engineering costs.
- Readily available in quantity including reliable second sources (other companies make the part). For long term production, this can be important.
- Experience of others can be used. Training for a new processor is a long and costly affair.

8.6.3.2 Development Tools

Development time which includes programming, is expensive. In order to reduce this, microprocessor vendors have provided information very quickly. This includes data sheets and programmer's guides. Related to this is the architecture of the processor. We also need to answer questions like:

Does it have the features to do the task?
Do we have adequate development time to overcome deficiencies in the part?

Sometimes, the vendors also provide some development support in the form of ready-made boards (including processor, memory, I/O ports). Similar to these are evaluation kits which have more hardware and facilities for software development, like including a monitor program to download and execute user programs through a serial port, and being able to modify portions of memory. These kits also allow for additional hardware to be added on easily by providing extra connectors and sometimes a prototyping area.

Software development tools should be adequate. Assemblers, compilers with simulators and other debugging tools should make the task of programming easier. This is especially important if the microprocessor's architecture is inadequate in many areas.

Emulator support is of the highest priority. An emulator makes hardware debugging seem like software debugging.

8.6.3.4 Processor Capability

The processor should be fast enough to support the application at hand. This is especially true for real time applications. The addressing space should be enough for program and data.

8.6.4 Software vs Hardware

Processors can be programmed to perform a wide variety of tasks. Engineers are finding that more and more, their task is to utilize available processor hardware by writing the necessary software. However, there is still a need for some electronic hardware design in the interface.

8.6.4.2 Software/Hardware Trade-Off

In general, there would be a number of tasks which can be achieved by means of hardware or software, especially those that interact with external environment and devices. For example, a serial port may be written by manipulating some port pins, or it may be purchased as a hardware component. Not surprisingly, the deciding factor is cost. This can be seen as:

Unit Cost = Component Cost + Production Cost + Development Cost / Total No. of Units

Hardware Implementation
Has an associated component cost but there is

- less development effort
- higher processing speed as the processor does not have to work on this

Software Implementation
No component cost

- may involve large amount of development effort
- In most cases, operation is much slower

8.7 System Design

The method that was introduced in the section on system analysis is also a method for system design. UML is a modern technique of system design and it has been specified as the documentation to be used for all systems proposed in project tenders submitted to the British government.

8.7.1 The Unified Modeling Language

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the features of systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. We mentioned it earlier in System Analysis.

UML represents an important part of system development process. UML uses mostly graphical notations to express the design of system projects. Using the UML helps

project teams communicate, explore potential designs, and validate the architectural design of systems.

The primary goals in the design of the UML were:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
6. Integrate best practices.

8.7.1.2 The types of UML diagrams

UML diagrams are designed to let developers and customers view a system from a different perspective and in varying degrees of abstraction. There are altogether six diagrams to be drawn for a UML conformant documentation. In this module, we only need you to draw two - the interaction and the *use case* diagrams.

The UML *use case* diagrams display the relationship among actors (users) and *use cases*. This terminology comes from computer languages where the *case* statement described an action to be taken by a program that depends on the value of a variable, as in the C language.

To draw this, you would first have to imagine you are a user of the yet-to-be built system and you would then come up with the ways offered to you by the system that allows you to interact with the system.

Interaction Diagrams consist of:

a. Sequence Diagram: these diagrams display the time sequence of the objects participating in the interaction. These consist of the vertical dimension (time) and horizontal dimension (different objects).
b. Collaboration Diagram: these diagrams display an interaction organized around the objects and their links to one another.

In this course, our primary focus would be use case and *sequential* interaction diagrams. After the diagrams are drawn, we implement the task to be executed using a combination of hardware and software that the computer has to do. An example is given at the end of the chapter.

8.7.1.4 Hardware Testing

Should be tested on each sub-system basis, using simple programs designed to fully test out the hardware. The test programs should be kept as simple as possible to eliminate the possibility of software errors. This assumes the basic hardware is working correctly.

Modern instruments have been devised to cope with the short-comings of the simple test techniques mentioned above. These are Logic Analyzers and In-Circuit Emulators.

8.7.1.6 Software Testing

After the software code for each subsystem has been written and converted into machine code, it must be tested to verify that it functions correctly. This requires a 'protected' environment to test the software. This allows us to control and monitor the inputs and outputs of the overall system precisely.

For example the environment provided by the monitor or debugger utility program in a microprocessor development system where the performance of the system being tested can be examined in isolation.

All possible conditions going through the control structure is tested. For example, "IF" type of instructions should be tested for both conditions. This is so that all instructions will be checked and do not cause errors later in the project. The aim should be to minimize the number of untested software codes which have to be checked during system integration.

Software that depends heavily on special features provided by the actual hardware should be tested right at the very beginning

8.8 System Integration

System Integration is the process of bringing together individually designed and tested subsystems to form the full system.

- This normally begins by building the whole system from the bottom upwards and implementing the tasks as illustrated in the interaction diagrams.

- Extensively testing the tasks that needs to be performed by the system to verify that no problems have been introduced by linking sub-systems together

Ideally, if top-down design has been rigorously applied then system integration should be straight forward. However even though we have been careful with our initial documentation, there may still be some inconsistencies and these will not appear until

the system integration phase. These are often caused by focussing only on one aspect of the design and neglecting interfacing issues.

If integration is done properly, the complete system can be simply implemented as a series of subroutine calls within a main loop.

8.8.1 Problem resolution

As we are testing the system, we need to check if the system carries out every task as required and it is during this phase that we find inconsistencies. These bugs must be resolved if the system is going to work. It is therefore essential to have access to the initial documentation, and to update it with the lists of bugs found in a structured manner.

If we try to patch up inconsistencies by an ad hoc method, we are likely to get into more and more trouble, particularly if we do it in an undocumented manner.

Although it may appear to be slower, it is more cost effective in the long run to go right back to a task (the one with the bugs) to find out where we went wrong, and then make modifications which are consistent throughout the system.

When the lists have been more or less resolved, we have created a prototype of required system.

Prototype
A prototype is the initial version of the system. There are two types and are used to:

- Demonstrate the validity of the system design
- Show what the system will ultimately look like. We should be very careful about satisfying other needs, such as outward appearance or physical construction. These should only be done if it helps to confirm that the original design concept works.

System Prototype Without Hardware
The development system used as hardware, where input is a keyboard and output on a display.

Once the design has been proved, hardware can then be bought or constructed to produce a second prototype with different goals.

Hardware Prototype
In general, the prototype system is separated from the development system. Software is transferred to the prototype hardware by:

- EPROM
- In-Circuit Emulator (ICE)

8.9 Production

This marks the transition from a prototype to a manufactured product. It may involve changes in original design for the following reasons:

- When product is manufactured in quantity, assembly cost becomes a major factor in the price
- Need to ensure that the system can be assembled and tested by relatively unskilled staff
- Require documentation such as test specifications, service manuals, and user's manuals

8.9.1 Testing of Products

Different from testing during development stage

- Need to perform fault-finding relatively quickly by the use of special test jigs and/or sophisticated test instruments
- Test procedures must be devised

8.10 SAMPLE DESIGN

The following is an example of how we apply UML to a design. Note that there is not just one correct answer. As in all design activity, the designer will justify the choices made in meeting the customer requirements.

> You are required to design a microprocessor based egg timer. This device will show the egg type and boil it for a specified period of time depending on the egg type. It will sound an alarm to signal when it finishes and show a message as well. It must be waterproof.
>
> Perform System Analysis and System Design using UML. Draw a case and interaction diagram of the proposed system. Develop two of the sequential diagrams that you will use.

First, identify the subsystems in the proposed system. Note that we do not need a temperature sensor because the boiling temperature of water is constant. A convenient breakdown of the systems into subsystems would be as follows:

- microprocessor
- alarm
- timer

- LCD
- heating coil
- keypad

Then we draw a UML *use case* diagram. A UML *use case* diagram is another way of expressing user requirements. The diagram helps in uncovering user requirements that may not have been stated earlier on.

8.10.1 Use case diagram

Looking at our example from the user's perspective, the user uses the egg timer for cooking the egg. So a *use case* diagram that a designer would come up with is:

The action marked "cook egg" is a function offered by the system. You might wonder what is the purpose of such a diagram. The process of drawing such a diagram helps you, the designer to identify functions that the system must propose.

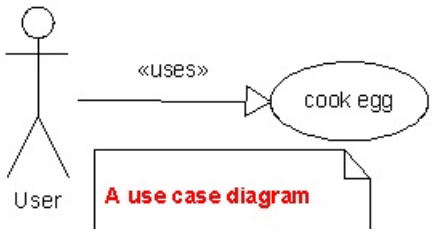

Fig 8.2 Use Case Diagram

Case diagrams are helpful in revealing requirements and planning the project. During the initial stage of a project, most *use cases* should be defined, but as the project continues more functions might become visible as you explore how the system interacts with the user. That is, you will able be to uncover functions that the system should have, but was not explicitly written. In our example, to cook an egg needs to include steps to "choose egg type" before we start to "cook egg".

A general method that one uses to come up with a use diagram is to list a sequence of steps a user might take in order to complete an action. For example:

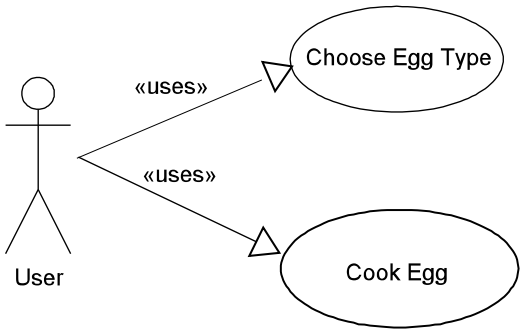

Fig 8.3 Developing the use case diagram

From this simple diagram the requirements of the egg cooker system can easily be

derived. The system will need to be able to perform actions for all of the use cases listed. As the project progresses other use cases might appear and this diagram can easily be expanded until a complete description of the egg cooking system is derived capturing all of the requirements that the system will need to perform.

8.10.2 Interaction diagram

These diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task. The two kinds of interaction diagrams are sequence and collaboration diagrams.

Interaction diagrams are used when you want to model the behavior of several sub-systems in a *use case*. They demonstrate how the sub-system collaborates with others. However, interaction diagrams do not give an in depth representation of the behavior. If you want to see what a specific object is doing for several *use cases* we should use a state diagram - which is another component of UML. To see a particular behavior over many *use cases* or threads use an activity diagram - which is yet another component of UML.

To start drawing an interaction diagram, first imagine how you would interact with the system. In our example, you - the user, would select the egg type using the key type and you would represent the interaction as in the left figure below and note how the subsystem key pad is represented as:

Developing Use case - "Select Egg Type"

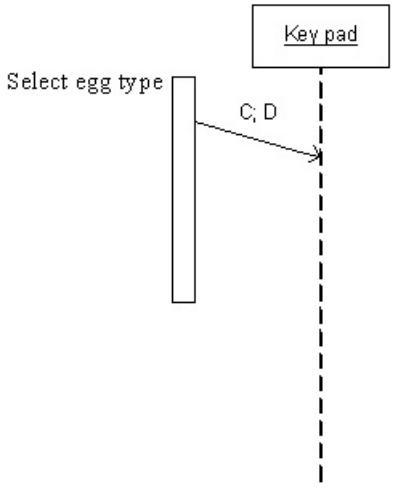

Fig 8.4 Starting Interaction Diagram

Fig 8.5 Part of Interaction Diagram

In like fashion, the other sub-systems will have similar diagrams. The diagram shown below is often used to represent the first interaction with the system in question. In this case, the first contact between the system and you, the user is the selection of the egg type for cooking by pressing either C(hicken) or D(uck):

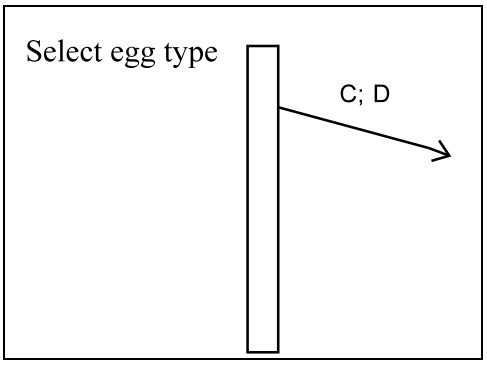

Fig 8.6 Interaction Diagram for "Select Egg Type": Step 1

To develop the interaction diagram further, after the keypad is contacted, the microprocessor would read the key entered, displays the egg type and we have the following diagram:

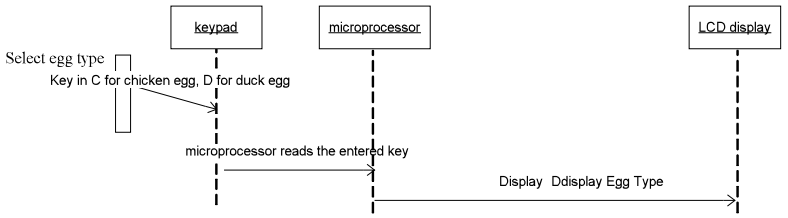

Fig 8.7 Interaction Diagram for "Select Egg Type": Step 2

Developing Use case - "Cook Egg"

In developing this *use case*, we see that once the microprocessor reads the key entered and processes it, it activates the LCD display to show the egg type selected, the timer to start running for a certain period of time depending on the egg type selected and lastly to activate the heater coil.

After the time *t* is reached, the timer would then send a signal to the microprocessor, telling it that the time is up; the microprocessor would then deactivate the heater coil and send a signal to the LCD display telling it to display the message "egg cooked".

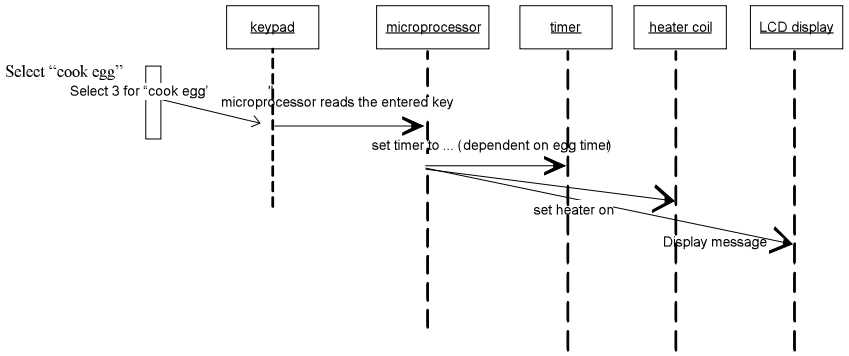

Fig 8.8 Interaction Diagram for "cook egg" step 1

After the egg is considered cooked, we need to activate the alarm. The microprocessor would then reactivate the timer for a much shorter period of time. Then the microprocessor activates the alarm. When the timer's time runs out, the timer interrupts the microprocessor and the processor then deactivates the alarm. Finally, the microprocessor would activate the LCD to redisplay the initial menu.

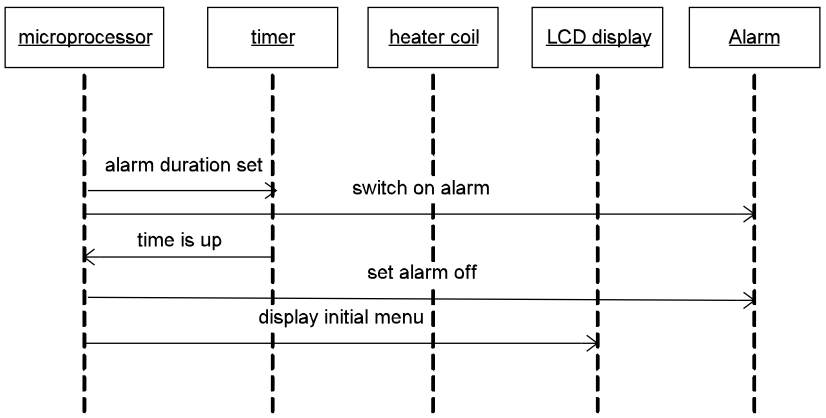

Fig 8.9 Interaction Diagram for "cook egg" : Step 2

The complete Interaction Diagram for "cook egg" would look like the following:

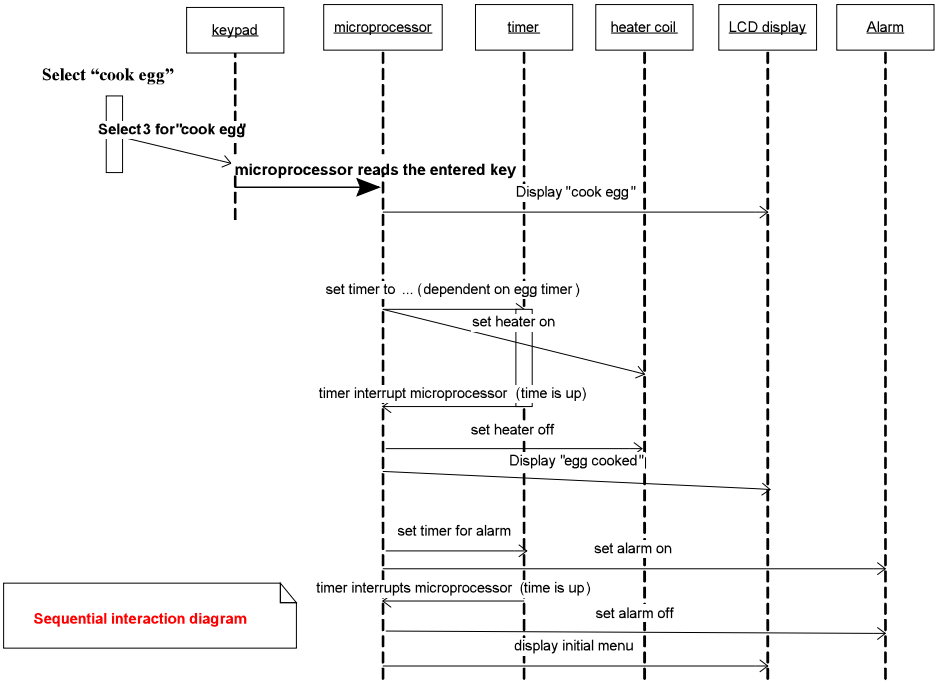

Fig 8.10 Final Sequential Interaction Diagram for “cook egg”

As you would can see, such a diagram would have helped in implementation and would prove very useful in explaining the implementation of your system to your clients.

Should there be another way in which a user interacts with your system, you would need to repeat the method and produce a similar diagram depicting how the system carry out the task.