Part II — Design Tools

Chapter 5 System Design I: Functional Decomposition

> *At Sony, we assume all products of our competitors will have basically the same technology, price, performance, and features. Design is the one thing that differentiates one product from another in the marketplace.*—Norio Ohgo, Chairman and CEO, Sony

After the technical concept is selected, it is translated into a solution that satisfies the system requirements. The designer must put on paper, or the computer screen, a representation that is meaningful and clear; in other words, a useful abstraction of the system. Engineering designs are often complex, consisting of many systems and subsystems, thus this representation should facilitate the design process and effectively describe the system. In addition, it serves an important function in communicating the design to all members of the team. Imagine a scenario where each team member is responsible for designing part of a large system. Each person develops a part in isolation and several months later the team gets back together to integrate the pieces. Of course, the system won't work unless the team has collectively defined and communicated the functionality and interfaces for all subsystems in the design.

This chapter presents a well-known design technique—known as ***functional decomposition***—that is intuitive, flexible, and straightforward to apply. It is probably the most pervasive design technique used for engineering systems and is applicable to a wide variety of problems that extend well beyond electrical and computer engineering. In functional decomposition, systems are designed by determining the overall functionality and then iteratively decomposing it into component subsystems, each with its own functionality.

The objective of this chapter is to present both basic design concepts and the functional decomposition design technique. A process for functional decomposition is provided and it is applied to examples in analog electronics, digital electronics, and software systems.

Learning Objectives

By the end of this chapter, the reader should:

- Understand the differences between bottom-up and top-down design.
- Know what functional decomposition is and how to apply it.
- Be able to apply functional decomposition to different problem domains.
- Understand the concepts of coupling and cohesion, and how they impact designs.

5.1 Bottom-Up and Top-Down Design

Two general approaches to synthesizing engineering designs are known as ***bottom-up*** and ***top-down***. In the case of bottom-up, the designer starts with basic components and synthesizes them to create the overall system. To use an analogy, consider the case of creating an automobile. In the bottom-up approach, you have pieces of the automobile, such as the tires, motor, axle, transmission, alternator, and they are brought together to create a car. The implication is that the final system depends upon the parts at hand. In other words, in the bottom-up approach, the parts and subsystems are given, and from them an artifact is created.

The top-down approach is analogous to the concept of divide and conquer. In top-down the designer has an overall vision of what the final system must do, and the problem is partitioned into components, or subsystems that work together to achieve the overall goal. Then each subsystem is successively refined and partitioned as necessary. In the case of the automobile, the overall objective is determined; the major subsystems are defined, such as electrical, power drive train, and the suspension; and then each subsystem is further refined into its component parts until the complete system is designed.

A debate that continues in the design community revolves around which is the better approach. It might appear that top-down is better, since it starts with the overall goal (requirements) and from that a solution is developed. Top-down is particularly valuable on large projects with many subsystems, where it is unlikely that bringing together pieces in an ad-hoc fashion will successfully solve the problem. A disadvantage of top-down design is that it tends to limit the solution space and innovation. Top-down design is inclined to follow a vertical thought process (Chapter 4) where the designer starts with a problem and successively refines the subsystems until a blueprint for solving the problem is defined. Furthermore, the designer cannot create a top-down design in a vacuum without bottom-up knowledge of existing technology and how the system can be realized.

Bottom-up has the advantage of lending itself to creativity. It allows the designer to take different technologies and from them create something new, allowing more "what if?" questions to be asked. Bottom-up design is applicable when there are constraints on the components that can be used. This is a realistic scenario. Consider the case of variant design, where the goal is to improve the performance of an existing, or legacy, system. For example, automobile manufacturers might have to redesign their models to meet new emissions, mileage, or safety standards. If you are not starting with a new design and must utilize existing systems, it requires bottom-up thinking. In reality, most problems require a combination of bottom-up and top-down thinking, and the designer must alternate between them.

In summary, it is most effective to work between bottom-up and top-down. A completely top-down approach is not feasible because the designer must have an understanding of the bottom level technology for the components of the design hierarchy to be realistic. Likewise, completely bottom-up by itself is generally not feasible, particularly as the system complexity grows.

5.2 Functional Decomposition

Functional decomposition is a recursive process that iteratively describes the functionality of all system components. It is analogous to the mathematical concept of a function, for example, $y = f(x)$. In this function there is an input x, an output y, and a transformation between the input and output $f()$. This is easily extended to the case of multiple inputs and outputs where the inputs and outputs are vectors, $\vec{y} = f(\vec{x})$. In functional decomposition, the same items are defined as in the mathematical analogy—the inputs, the outputs, and the transformation between the inputs and outputs (the functionality). Those three items constitute what is known as the ***functional specification*** or ***functional requirement***—the requirement that a functional module should meet. A ***module*** is a block, or subsystem, that performs a function. Functional decomposition has a strong top-down flavor, due to the fact that the highest level functionality is defined and then further refined into subfunctions, each with its own inputs, outputs, and functionality. The process is repeated until some base level functionality is reached where the modules can be actualized with physical components.

A process for applying functional decomposition is illustrated in Figure 5.1. It starts with a definition of the highest level (Level 0) of system functionality (the functional requirement for the system). This is followed by definition of the next level of the hierarchy that is needed to achieve the design objective. The Level 1 design is typically referred to as the main ***design architecture*** of the system. In this context, architecture means the organization and interconnections between modules. Care must be taken at each design level to ensure that it satisfies the requirements of the higher level. The process is repeated for successive levels of the design and stops when the ***detailed design*** level is reached. Detailed design is where the problem can be decomposed no further and the identification of elements such as circuit components, logic gates, or software code takes place. The number of levels in the design hierarchy depends upon the complexity of the problem.

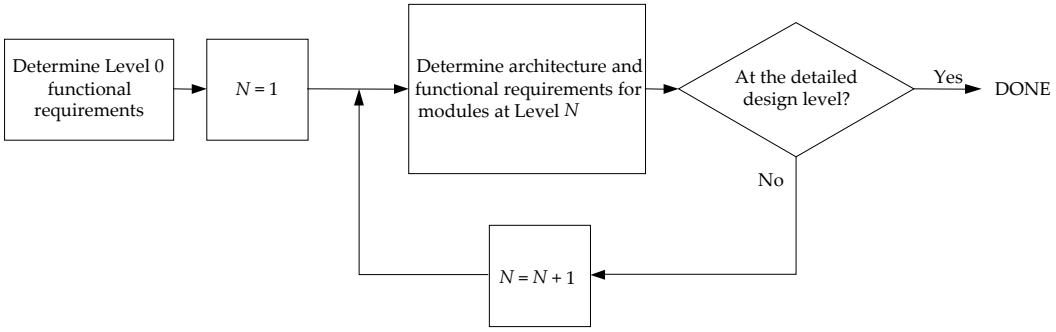


Figure 5.1 A process for developing designs using functional decomposition.

5.3 Guidance

The following guidance is provided before examining applications of the functional decomposition technique:

- *It is an iterative process*. During the first pass, it is not possible to know all of the detailed interfaces between components and the exact functionality of each block. In act, some details are not known until the implementation level is reached, so the designer needs to iterate, work between top-down and bottom-up, and adjust the design as necessary.
- *Set aside sufficient time to develop the design*. This is a corollary to the previous point. The iterative nature means that it takes time to examine different solutions and to refine the etails into a working solution.
- *Pair together items of similar complexity.* Modules at each level should have similar complexity and granularity.
- *A good design will have the interfaces and functionality of modules well-defined.* It is fairly easy to piece together some blocks into an apparently reasonable design. However, the functional requirements should be clearly defined and the technical feasibility understood. If not, the design will fall apart when it comes to the implementation stage. Consider the following advice of a well-known architectural designer:

  The details are not the details. They make the design.—Charles Eames

- *Look for innovations.* Top-down designs tend to follow a vertical thinking process, where the designer proceeds linearly from problem to solution. Try to incorporate lateral thinking strategies from Chapter 4 and examine alternative architectures and technologies for the solution.
- *Don't take functional requirements to the absurd level.* Common elements, such as analog multipliers or digital logic gates, do not require explicit functional specifications. Doing so may become cumbersome and add little to the design. However, it depends upon the level at which you are working. If the goal is to design an analog multiplier chip, it is entirely appropriate to develop the functional requirement for the multiplier.
- *Combine functional decomposition with other methods of describing system behavior.* There is no single method or unifying theory for developing designs. Functional decomposition alone cannot describe all system behaviors. It may be supplemented by other tools such as flowcharts (logical behavior), state diagrams (stimulus-response), or data flow diagrams. In the digital stopwatch example presented later in this chapter, the behavior is defined by state diagrams. Other methods for describing system behavior are addressed in Chapter 6.

- *Find similar design architectures.* Determine if there exist similar designs and how they operate. Realize that this creates a bias toward existing solutions.
- *Use existing technology*. Many designers take the attitude that they are going to develop the entire design themselves, the sentiment being to ignore technology that they did not develop. Furthermore, engineering education predisposes us to design at a fundamental level. Both factors lead to time spent reinventing the wheel. If existing technology is available that meets both the engineering and cost requirements, then use it.
- *Keep it simple*. Do not add complexity that is not needed.

> *A designer knows that he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away.*—Antoine de St-Exupery

- *Communicate the results*. It is important to describe the theory of operation (the *why*) as well as the implementation (the *what*). The *what* in the completed design is usually quite clear from the implementation, but documenting the description of operation and design decisions helps later when the system must upgraded. Designs can also become very complex, so consider how much information can be effectively communicated on a single page. If the information is too complex to show reasonably on a page or two, then it probably is too detailed and another level in the hierarchy should be added.

5.4 Application: Electronics Design

We now examine the application of functional decomposition in different problem domains. In the domain of analog electronics, the inputs and outputs of modules are voltage and current signals. Typical transformations applied to the inputs are alterations in amplitude, power, phase, frequency, and spectral characteristics. Consider the design of an audio power amplifier that has the following engineering requirements.

The system must

- Accept an audio input signal source with a maximum input voltage of 0.5 V peak.
- Have adjustable volume control between zero and the maximum volume level.
- Deliver a maximum of 50 W to an 8 Ω speaker.
- Be powered by a standard 120 V, 60 Hz, AC outlet.

Level 0

The Level 0 functionality for the amplifier is shown in Figure 5.2, which is fairly simple—the inputs are an audio signal, volume control, and wall outlet power, and the output is an amplified audio signal.

Figure 5.2 Level 0 audio power amplifier functionality.

The system should be described in as much detail as possible for each level via the functional requirement. The Level 0 functional requirement for this design is as follows.

Module	Audio Power Amplifier
Inputs	- Audio input signal: 0.5 V peak. - Power: 120 V AC rms, 60 Hz. - User volume control: variable control.
Outputs	- Audio output signal: ? V peak value.
Functionality	Amplify the input signal to produce a 50-W maximum output signal. The amplification should have variable user control. The output volume should be variable between no volume and a maximum volume level.

Not all values can be known on the first pass through the design, as was indicated in the guidelines. Underlined items represent values that need to be determined or refined as the design proceeds. In this case, the peak value of the audio output voltage is determined from the system requirements on power gain. Knowing that the maximum power is given by $P_{\max} = V_{\text{peak}}^2 / R$ allows the maximum output voltage to be computed as $V_{\text{peak}} = \sqrt{8\ \Omega * 50\ \text{W}} = 20\ \text{V}$.

Level 1

The Level 1 diagram, or system architecture, is shown in Figure 5.3. This architecture is common in amplifier design and is but one possible solution. It contains three cascaded amplifier stages and a DC supply that powers the three stages. The first amplifier stage, the *buffer amplifier*, provides a high-resistance buffer that minimizes loading effects with the source. Buffer amplifiers have extremely high input resistance and a unity signal gain. The *high-gain amplifier* increases the amplitude of the signal, but provides little in terms of the output current necessary to drive the speakers. The last stage in the cascade is the *power output stage*, which provides the current needed to drive the speakers, but has no voltage amplification.

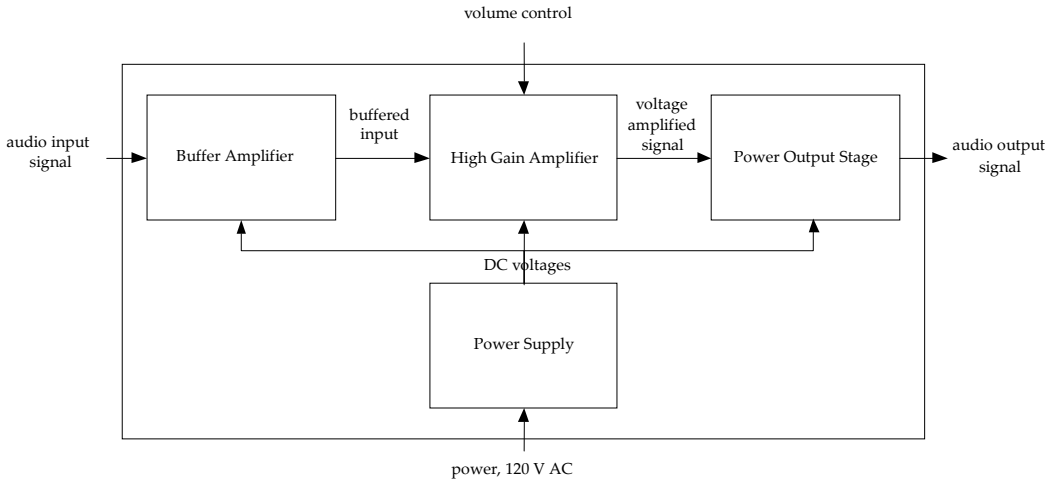


Figure 5.3 Level 1 audio amplifier design.

The functional requirements for the Level 1 subsystems are now detailed, starting with the buffer amplifier.

Module	Buffer amplifier
Inputs	- Audio input signal: 0.5 V peak. - Power: ± 25 V DC.
Outputs	- Audio signal: 0.5 V peak.
Functionality	Buffer the input signal and provide unity voltage gain. It should have an input resistance > 1 MΩ and an output resistance < 100 Ω.

Where did the ±25 V DC value for the DC input power come from? The system must produce a ±20 V AC output signal to satisfy the Level 0 requirement, so supply values that exceed that are required to power the electronics. How about the values for the input and output resistance? They are educated guesses, based on knowledge of what is achievable with the technology (bottom-up knowledge). The exact resistance requirements are refined later on the basis of the overall design, taking into account the input and output resistances for all stages.

Now consider the functional requirements for the high-gain amplifier.

Module	High-gain amplifier
Inputs	- Audio input signal: 0.5 V peak. - User volume control: variable control. - Power: ± 25 V DC
Outputs	- Audio signal: 20 V peak.
Functionality	Provide an adjustable voltage gain, between 1 and 40. It should have an input resistance > 100 kΩ and an output resistance < 100 Ω.

The gain of 40 is determined from the overall system power and gain requirements (the maximum input voltage of 0.5 V must be able to be amplified to 20 V), while the resistances are again educated guesses.

Now consider the power output stage.

Module	Power Output Stage
Inputs	- Audio input signal: 20 V peak. - Power: ± 25 V DC.
Outputs	- Audio signal: 20 V peak at up to 2.5 A.
Functionality	Provide unity voltage gain with output current as required by a resistive load of up to 2.5 A. It should have an input resistance > 1 MΩ and output resistance < 1 Ω.

For the power output stage, it is clear that 20 V peak needs to be delivered, but how was the requirement on current determined? The current needed to drive the speaker is determined from Ohm's law as $I = V/R = 20/8 = 2.5$ A.

The last module to examine at this level is the power supply.

Module	Power Supply
Inputs	- 120 V AC rms.
Outputs	- Power: ± 25 V DC with up to 3.0 A of current with a regulation of < 1%.
Functionality	Convert AC wall outlet voltage to positive and negative DC output voltages, and provide enough current to drive all amplifiers.

It is clear that the power supply needs to deliver ± 25 V DC, while the 3.0 A current capability was selected to supply the 2.5 A needed for the peak output power requirement plus the current needed to power the other amplifier stages.

Finally, it is necessary to determine if the values of the input and output resistances selected for the stages are realistic. For cascaded amplifier stages, the overall voltage gain is given by the product of gains multiplied by the voltage divider losses between stages [Sed04]. In this case the overall gain is

$$\begin{aligned}\text{Voltage gain} &= \text{gain}_1 \times \text{gain}_2 \times \text{gain}_3 \left(\frac{R_{\text{in2}}}{R_{\text{in2}} + R_{\text{out1}}}\right)\left(\frac{R_{\text{in3}}}{R_{\text{in3}} + R_{\text{out2}}}\right) \\ &= 1 \times 40 \times 1 \left(\frac{100\text{ k}}{100\text{ k} + 100}\right)\left(\frac{1\text{M}}{1\text{M} + 100}\right) \\ &\approx 40\end{aligned} \quad . \qquad (1)$$

So the resistance values selected in the functional requirements satisfy the overall system requirements. If not, it would be necessary to go back and refine them.

Level 2

At this point, the three amplifier stages are ready for detailed component level design, while the power supply needs another level of refinement, as shown in Figure 5.4. The functional requirement for each of the elements in the power supply would be developed similarly. Functional decomposition stops at this point—all levels of the hierarchy are defined and the next step is the detailed design, where the actual circuit components are determined.

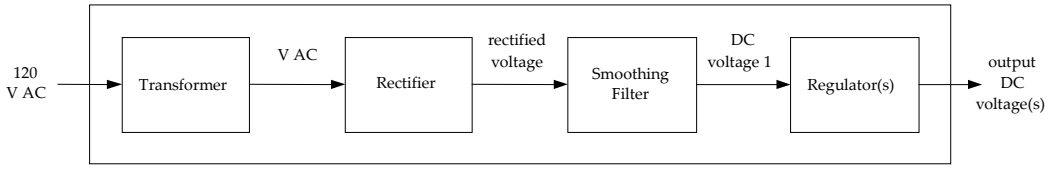


Figure 5.4 Level 2 design of the power supply.

5.5 Application: Digital Design

Functional decomposition is widely applied to the design of digital systems, where it is known as *entity-architecture* design. The inputs and outputs refer to the entity, and the architecture describes the functionality. The application of functional decomposition to digital systems is demonstrated in the following example. Consider the design of a simple digital stopwatch that keeps track of seconds and has the following engineering requirements.

The system must

- Have no more than two control buttons.
- Implement run, stop, and reset functions.
- Output a 16-bit binary number that represents seconds elapsed.

Level 0

The Level 0 diagram and functional requirements are shown in Figure 5.5.

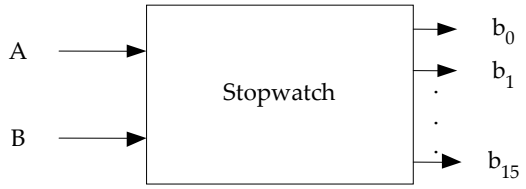


Figure 5.5 Level 0 digital stopwatch functionality.

Module	Stopwatch
Inputs	- A: Reset button signal. When the button is pushed it resets the counter to zero. - B: Run/stop toggle signal. When the button is pushed it toggles between run and stop modes.
Outputs	- b_{15}–b_0: 16-bit binary number that represents the number of seconds elapsed.
Functionality	The stopwatch counts the number of seconds after B is pushed when the system is in the reset or stop mode. When in run mode and B is pushed, the stopwatch stops counting. A reset button push (A) will reset the output value of the counter to zero only when the stopwatch is in stop mode.

Level 1

The Level 1 architecture in Figure 5.6 contains three modules: a seconds counter, a clock divider, and a finite state machine (FSM). The stopwatch counts seconds, thus the seconds counter module counts the seconds and outputs a 16-bit number representing the number of seconds elapsed. The clock divider generates a 1 Hz signal that triggers the seconds counter. The FSM responds to the button press stimuli and produces the appropriate control signals for the seconds counter. The system clock is included to clock both the FSM and the clock divider.

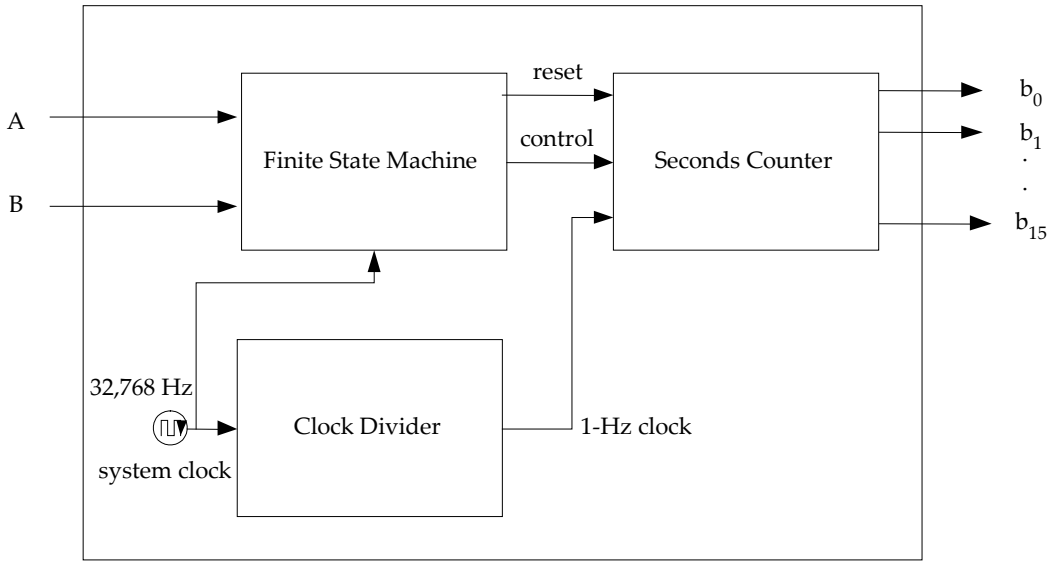
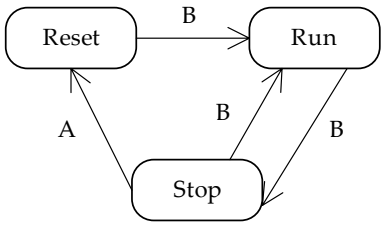


Figure 5.6 Level 1 design for the digital stopwatch.

The functionality of the Level 1 modules is described as follows, starting with the finite state machine.

Module	Finite State Machine
Inputs	- A: Signal to reset the counter. - B: Signal to toggle the stopwatch between run and stop modes. - Clock: 1 Hz clock signal.
Outputs	- Reset: Signal to reset the counter to zero. - Control: Signal that enables or disables the counter.
Functionality	

The functionality of the finite state machine is described with a tool that is probably familiar to the reader, the state diagram. State diagrams are covered in more detail in Chapter 6. The state diagram describes stimulus-response behavior, and shows how the system transitions between states according to logic signals from the button presses.

Next, consider the clock divider.

Module	Clock Divider
Inputs	- System clock: 32,768 Hz.
Outputs	- Internal clock: 1 Hz clock for seconds elapsed.
Functionality	Divide the system clock by 32,768 to produce a 1 Hz clock.

The value of 32,768 Hz was selected for the system clock for several reasons. It is a power of 2 that is easily divisible by digital circuitry to produce a 1 Hz output signal. It is also well above the clock rate needed for detecting button presses, and there is a wide selection of crystals that can meet this requirement.

Finally, consider the seconds counter.

Module	Seconds Counter
Inputs	- Reset: Reset the counter to zero. - Control: Enable/disable the counter. - Clock: Increment the counter.
Outputs	- b_{15}–b_0: 16-bit binary representation of number of seconds elapsed.
Functionality	Count the seconds when enabled and resets to zero when reset signal enabled.

The system decomposition would end here, assuming that the design is to be implemented with off-the-shelf chips. The next step would be to determine components at the detailed design level. However, if it were an integrated circuit design, the description would continue until the transistor level is reached.

5.6 Application: Software Design

Software also lends itself to functional decomposition, since virtually all computing languages provide the capability to call functions, subroutines, or modules. Functional software design simplifies program development by eliminating the need to create redundant code via the use of functions that are called repeatedly.

Structure charts are specialized block diagrams for visualizing functional software designs. The modules used in a structure chart are shown in Figure 5.7. The larger arrows indicate connections to other modules, while the smaller arrows represent data and control information passed between modules. Five basic modules are utilized:

1) *Input modules*. Receive information.
2) *Output modules.* Return information.
3) *Transform modules.* Receive information, change it, and return the changed information.
4) *Coordination modules.* Coordinate or synchronize activities between modules.
5) *Composite modules.* Any possible combination of the other four.

This approach to software design, also known as *structured design*, was formalized in the 1970s by IBM researchers [Ste99].

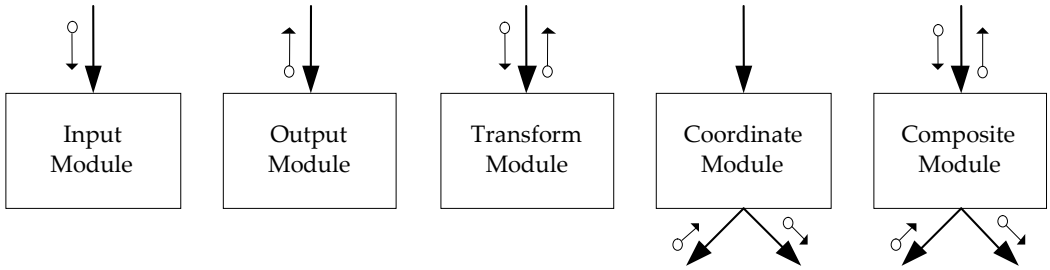


Figure 5.7 Module types for functional software design. The larger arrows indicate connections between modules and the smaller arrows represent data and control.

The following example demonstrates the application of functional decomposition to a software design with the following requirements.

The system must

- Accept an ASCII file of integer numbers as input.
- Sort the numbers into ascending order and save the sorted numbers to disk.
- Compute the mean of the numbers.
- Display the mean on the screen.

This is a fairly simple task that could easily be done in a single function, but doing so would not allow components of the design to be easily reused, tested, or troubleshot. The engineering requirements themselves provide some guidance in terms of how to arrange the functionality of the modules (*form follows function*). The architecture in Figure 5.8 contains a `main` module that calls three submodules. In this design `main` is a coordinating module that controls the processing and calling of the other modules, a common scenario. It was also decided that all user interaction would take place within `main`. The order of the processing is not described by structure charts. In our program, `main` calls `ReadArray`, `SortArray`, and `ComputeMean` in sequential order. `main` passes the filename (`fname`) to `ReadArray`, which reads in the array and the number of elements in it, and returns this information to `main`. The choice of passing in the filename was deliberate; the user could have been prompted for the filename in `ReadArray`, but doing so might limit future reuse of the function since you may not always want to do so when reading an array of data. `SortArray` is then called, which accepts the array of numbers and the number of elements in the array, and returns the sorted values in the same array. Finally, `ComputeMean` is executed, which accepts the sorted array and the number of elements, computes the mean value, and returns it to `main`.

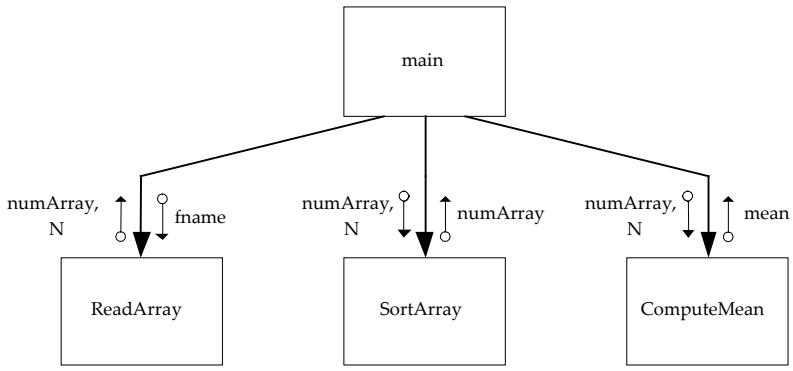


Figure 5.8 Structure chart design of sorting and mean computation program.

The functional requirements for each module in the structure chart are detailed in Table 5.1. The structure chart provides a visual relationship between modules in the design, but also has some disadvantages. It is difficult to visualize designs as the complexity of the software increases. This can be addressed by expanding sublevels in the design as necessary in different diagrams. Structure charts also lack a temporal aspect that indicates the calling order. Most software systems have many layers in the hierarchy and highly complex calling patterns. In this example, `main` calls three modules in a well-defined order, but if there were another level in the hierarchy, there is no reason why it could not be called by a module at any other level. That leads to some of the unique problems associated with software design. Functional design works well for small to moderately complex software, but tends to fall short when applied to large-scale software systems. As such, it has given way to the object-oriented design approach.

5.7 Application: Thermometer Design

The final example includes both analog and digital modules and the objective is to design a thermometer that meets the following engineering requirements.

The system must

- Measure temperature between 0 and 200°C.
- Have an accuracy of 0.4% of full scale.
- Display the temperature digitally, including one digit beyond the decimal point.
- Be powered by a standard 120 V, 60 Hz AC outlet.
- Use an RTD (resistance temperature detector) that has an accuracy of 0.55°C over the range. The resistance of the RTD varies linearly with temperature from 100 Ω at 0°C to 178 Ω at 200°C. (Note: this requirement does not meet the abstractness property identified in Chapter 3, since it identifies part of the solution. This requirement is given to provide guidance in this example.)

Table 5.1 Functional design requirements for the number sort program.

Module name	main()
Module type	Coordination
Input arguments	None.
Output arguments	None.
Description	The main function calls ReadArray() to read the input file from disk, SortArray() to sort the array, and ComputeMean() to determine the mean value of elements in the array. User interaction requires the user to enter the filename, and the mean value is displayed on the screen.
Modules invoked	ReadArray, SortArray, and ComputeMean.

Module name	ReadArray()
Module type	Input and output
Input arguments	- fname[]: character array with filename to read from.
Output Arguments	- numArray[]: integer array with elements read from file. - N: number of elements in numArray[].
Description	Read data from input data file and store elements in array numArray[]. The number of elements read is placed in N.
Modules invoked	None.

Module name	SortArray()
Module type	Transformation
Input arguments	- numArray[]: integer array of numbers. - N: number of elements in numArray[].
Output Arguments	- numArray[]: sorted array of integer numbers.
Description	Sort elements in array using a shell sort algorithm. Saves the sorted array to disk.
Modules invoked	None.

Module name	ComputeMean()
Module type	Input and output
Input arguments	- numArray[]: integer array of numbers. - N: number of elements in numArray[].
Output arguments	- mean: mean value of the elements in the array.
Description	Computes the mean value of the integer elements in the array.
Modules invoked	None.

Level 0

The overall goal is to convert a sensed temperature to a digital temperature reading. The Level 0 description is shown in Figure 5.9.

Figure 5.9 Level 0 digital thermometer functionality.

Module	Digital Thermometer
Inputs	- Ambient temperature: 0–200°C. - Power: 120 V AC power.
Outputs	- Digital temperature display: A four digit display, including one digit beyond the decimal point.
Functionality	Displays temperature on digital readout with an accuracy of 0.4% of full scale.

Level 1

The Level 1 architecture selected is shown in Figure 5.10. The temperature conversion unit converts the temperature to an analog voltage, using the RTD that is sampled by the analog-to-digital converter. The *N*-bit binary output from the converter is translated into binary-coded decimal (BCD). BCD is a 4-bit representation of the digits between 0 and 9. Since there are four display digits, there are four separate binary encoded outputs from the BCD conversion unit. Common seven-segment LEDs are used for the display. However, they do not directly accept BCD and instead have seven input lines, each of which is individually switched to control the display segments. The requirements did not specifically address cost or size constraints, nor clearly define the environment, so there are many possible solutions. For example, an analog-to-digital current converter could be used, integrated circuit temperature-sensing packages could be considered, and microcontroller-based solutions are feasible as well.

From a system design perspective, an error budget is needed to identify the maximum error that each subsystem may introduce, while still achieving the overall accuracy. In this case, error is introduced in the temperature conversion unit and A/D converter, but not in the remaining digital components. The overall accuracy that the system must achieve is 0.4%, and that translates into 0.8°C of allowable error for the 200°C range. Let's now examine the modules.

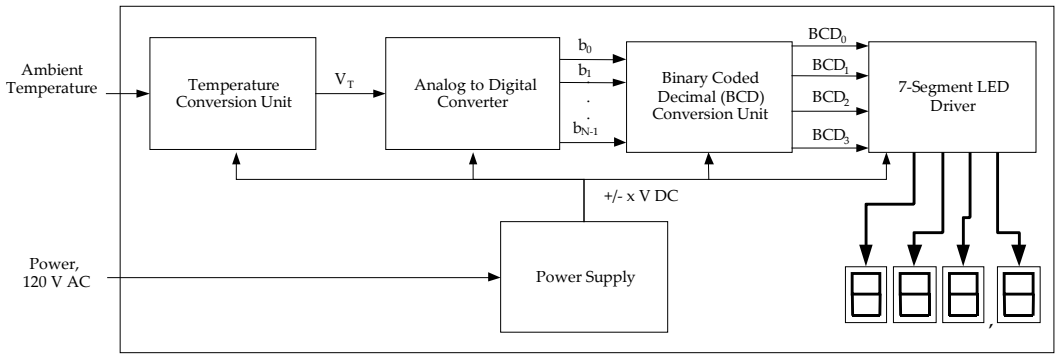


Figure 5.10 Level 1 design of the digital thermometer.

The functionality of the Level 1 modules is described as follows, starting with the temperature conversion unit.

Module	Temperature Conversion Unit
Inputs	- Ambient temperature: 0–200°C. - Power: ? V DC (to power the electronics).
Outputs	- V_T: temperature proportional voltage. $V_T = \alpha T$, and ranges from ? to ? V.
Functionality	Produces an output voltage that is linearly proportional to temperature. It must achieve an accuracy of ?%.

There are several unknowns at this point. The voltage necessary to power the electronics is not known, but a reasonable assumption could be made. The output voltage range and the accuracy are unknown. It is known that the RTD will introduce up to 0.55°C of error and that the electronics themselves will introduce additional error (the exact amount is unknown at this point). An educated guess is made that the maximum error allowed for the temperature unit is 0.6°C. This means that the electronics themselves would be required to introduce no more than 0.05°C of error as a result of the 0.55°C of error introduced by the RTD.

Now consider the analog to digital converter.

Module	A/D Converter
Inputs	- V_T: voltage proportional to temperature that ranges from ? to ? V. - Power: ? V DC.
Outputs	- b_{N-1}–b_0: ?-bit binary representation of V_T.
Functionality	Converts analog input to binary digital output.

It is not likely that it will be necessary to design the A/D converter because low-cost, off-the-shelf solutions are available. The requirements drive the converter selection. There are two

unknowns—the number of bits and the range of the input voltage. The number of bits affects the accuracy, since the greater the number of bits, the better the accuracy. The number of bits needed for the converter is calculated from the maximum allowable error that the A/D can introduce (0.2°C), the number of discrete intervals, and the temperate range as

$$\text{Max error} = \frac{\text{range}}{\text{number of intervals}} = \frac{200°C}{2^N} \le 0.2°\text{C} \quad \Rightarrow N \ge 9.97 \text{ bits.} \tag{2}$$

So the A/D converter needs to have at least 10 bits. How is the voltage range selected? It is typically fixed for a particular integrated circuit solution, but the temperature conversion subsystem output should be matched to the voltage range so that all bits are effectively utilized; otherwise, error is introduced.

Now, consider the BCD conversion unit.

Module	BCD Conversion Unit
Inputs	- 10-bit binary number (b_9–b_0): Represents the range 0.0–200.0°C. - Power: ? V DC.
Outputs	- BCD_0: 4-bit BCD representation of tenths digit (after decimal). - BCD_1: 4-bit BCD representation of ones digit. - BCD_2: 4-bit BCD representation of tens digit. - BCD_3: 4-bit BCD representation of hundreds digit.
Functionality	Converts the 10-bit binary number to BCD representation of temperature. Must refresh the displays twice a second.

The objective of the BCD conversion unit is fairly simple, although the component level design of the circuitry to accomplish the conversion is not.

This leads to the last module, the seven-segment LED driver, whose functionality is described as follows.

Module	Seven-Segment LED Driver
Inputs	- BCD_0: 4-bit BCD representation of tenths digit (after decimal). - BCD_1: 4-bit BCD representation of ones digit. - BCD_2: 4-bit BCD representation of tens digit. - BCD_3: 4-bit BCD representation of hundreds digit. - Power: ? V DC.
Outputs	- Four 7-segment driver lines.
Functionality	Converts the BCD for each digit into outputs that turn on LEDs in seven-segment package to display the temperature.

For completeness, the functional requirements of the power supply are supplied. They are similar to the power supply requirements utilized in the audio amplifier design in Section 5.4.

Module	Power supply
Inputs	- 120 V AC rms.
Outputs	- ± ? V DC with up to ? mA of current. - Regulation of ?%.
Functionality	Convert AC wall outlet voltage to positive and negative DC output voltages, with enough current to drive all circuit subsystems.

At this point, the requirements for the major subsystems are completed and ready for design at the component level. Illustration of the complete design would require a fair amount of detail, and while it is not presented here, some of the issues involved are discussed. First, there are a variety of electronic circuits (inverting op amps, single BJT configurations, and current mirrors, etc.—see Section 4.3.3) that could be utilized as a current source to drive the RTD in the temperature conversion subsystem. A midrange resolution A/D converter is needed, and its particular input voltage range drives the output voltage requirements for the temperature conversion module. The BCD conversion circuitry could be implemented using combinational digital logic (tedious due to the number of discrete gates), or a more efficient, but slower, sequential logic design. Finally, the seven-segment display converters could be designed using combinational logic that maps the BCD inputs into outputs to activate the appropriate display segments.

5.8 Coupling and Cohesion

The concepts of coupling and cohesion are examined before concluding this chapter. They originated to describe software designs [Ste99], but are applicable to electrical and computer systems. To understand their importance, consider the relationship between the number of modules in a system and the number of connections between them. For our purposes, a connection between two modules may consist of any number of signals without regard to their direction. Thus, a system consisting of two modules has, at most, one connection. If the number of modules is increased to three, the number of possible connections increases to three, a system with four modules has six possible connections, and five modules increases the number of possible connections to ten. The point is that the maximum number of potential connections increases rapidly with the number of modules in the system. The relationship between the maximum possible connections and number of modules (n) is given by

$$\text{Connections}_{\max} = \frac{n(n-1)}{2}. \tag{3}$$

Modules are coupled if they depend upon each other in some way to operate properly. ***Coupling*** is the extent to which modules or subsystems are connected [Jal97]. Although there is no agreed upon mathematical definition of coupling, it seems obvious that increasing the exchange of control and data between two modules leads to a higher degree of coupling. When systems are highly coupled, it is difficult to change one module without affecting the other. Consider the extreme case where all modules in a system are connected to each other—an error in one module has the potential to affect every other module in the system. Errors in a module are propagated to others to a degree that is related to the amount of coupling. From this point of view, it is good to minimize coupling. Yet coupling cannot be eliminated, since the point of functional decomposition is to break a design into components that work together to produce a higher level behavior.

There are two ways to reduce coupling—minimize the number of connections between modules and maximize cohesion within modules. ***Cohesion*** refers to how focused a module is—highly cohesive systems do one or a few things very well. Stevens et al. [Ste99] defined six types of cohesion from the weakest to strongest as: coincidental, logical, temporal, communicational, sequential, and functional. More information on this can be found in the original work, but the conclusion is that modules with high functional cohesion are the most desirable. So it is best to design modules with a single well-defined functional objective consistent with the philosophy of functional decomposition. This leads to the important design principle that it is desirable to maximize cohesion, while minimizing coupling.

Coupling and cohesion impact the later stages of testing and system integration. If a particular module is highly cohesive, then it should be possible to test it independently of the other modules to verify its operability. This does not mean that it will necessarily operate properly when integrated into the overall system, but the probability that it will is higher if provided with proper inputs from connected modules. Contrast this to the case of a low-cohesion system. In that case, it will likely be difficult to test the individual modules without first integrating them.

To develop a better understanding, consider the amplifier design in Figure 5.3 (Section 5.4) with three cascaded amplifier stages. Each stage is highly cohesive, performing a singular function of signal amplification. Each of these stages could easily operate as a stand-alone module independent of the complete system. How about coupling? In terms of the number of connections, it is fairly low as each amplifier stage has an input and output voltage signal. The most coupled module in the system is the power supply, and not surprisingly, its failure leads to a complete system failure. Coupling in this case can also be viewed in terms of the resistance matching between input and output of the cascaded stages, producing the voltage divider effect in equation (1). For voltage amplifiers, the goal is to have high input resistance and low output resistance, which minimize both voltage losses and coupling. The stages are not completely uncoupled, because the input resistances, although large, are not infinite, and the output resistances are not zero. The modules in the power supply unit in Figure 5.4 (rectifier, smoothing filter, and regulator) have a much higher degree of coupling. In fact, it is

difficult to develop a clear functional decomposition of the power supply module because the elements in the smoothing filter also serve as part of the rectifier circuit (refer to a basic electronics textbook [Sed04] for more information).

As another example, consider a software design where two options are under consideration: one large function with 1000 lines of code, versus 15 cohesive functions, each with an average of 100 lines of code. Both perform the same function, but which runs faster? Most likely the first, as it would be highly integrated and would not suffer from overhead needed with multiple functions. Which is easier to upgrade and debug a year from now? That is clearly the second case. Although loosely coupled and highly cohesive designs may facilitate better design and testing, they may not be best in terms of performance.

5.9 Project Application: The Functional Design

The following is a format for documenting and presenting functional designs.

Design Level 0

- Present a single module block diagram with inputs and outputs identified.
- Present the functional requirements: inputs, outputs, and functionality.

Design Level 1

- Present the Level 1 diagram (system architecture) with all modules and interconnections shown.
- Describe the theory of operation. This should explain how the modules work together to achieve the functional objectives.
- Present the functional requirements for each module at this level.

Design Level *N* (for *N* > 1)

- Repeat the process from Design Level 1 for as many levels as necessary.

Design Alternatives

- Describe the different alternatives that were considered, the tradeoffs, and the rationale for the choices made. This should be based upon concept evaluation methods communicated in Chapter 4.

5.10 Summary and Further Reading

This chapter presented the functional decomposition design technique, where every level of the design is decomposed into submodules, each of which is the domain of the next lower level. The inputs, outputs, and functionality must be determined for a given module. Appling the process in Figure 5.1 and following the guidelines in Section 5.3 should aid in the

application of functional decomposition. Functional decomposition is applicable to a wide variety of systems, and in this chapter designs of analog electronics, digital electronics, and software were examined.

Nigel Cross presents a good overview of the functional decomposition method with application to mechanical systems [Cro00], but with less focus on the description of the functional requirements than presented here. The work by Stevens et al. [Ste99] is interesting reading that gives an understanding of the evolution of structured design. It delves into the concepts of coupling and cohesion. Coupling and cohesion are also addressed well in the book by Jalote [Jal97]. An in-depth treatment of structured systems design is found in The Practical Guide to Structured Systems Design [Pag88]. This guide also integrates data flow diagrams with functional techniques. Finally, the thermometer design example was inspired by Stadtmiller's book [Sta01] on electronics design.

5.11 Problems

5.1 Describe the differences between *bottom-up* and *top-down* design.

5.2 Develop a functional design for an audio graphic equalizer. A graphic equalizer decomposes an audio signal into component frequency bands, allows the user to apply amplification to each individual band, and recombines the component signals. The design can employ either analog or digital processing. Be sure to clearly identify the design levels, functional requirements, and theory of operation for the different levels in the architecture.

The system must

- Accept an audio input signal source, with a source resistance of 1000 Ω and a maximum input voltage of 1 V peak-to-peak.
- Have an adjustable volume control.
- Deliver a maximum of 40 W to an 8 Ω speaker.
- Have four frequency bands into which the audio is decomposed (you select the frequency ranges).
- Operate from standard wall outlet power, 120 V rms.

5.3 Develop a functional design for a system that measures and displays the speed of a bicycle. Be sure to clearly identify the design levels, functional requirements, and theory of operation for each level.

The system must

- Measure instantaneous velocities between zero and 75 miles per hour with an accuracy of 1% of full scale.

- Display the velocity digitally and include one digit beyond the decimal point.
- Operate with bicycle tires that have 19-, 24-, 26-, and 27-inch diameters.

5.4 Draw a structure chart for the following C++ program:

```
void IncBy5(int &a, int &b);
int  Multiply(int a, int b);
void Print(int a, int b);

main(){
   int x=y=z=0;
   IncBy5(x,y);
   z=Mult(x,y);
   Print(x,z);
}
void IncBy5(int &a, int &b) {
   a+=5;
   b+=5;
   Print(a,b);
}

int Multiply(int a, int b){
 return (a*b);
}

void Print(int a, int b) {
   cout << a << ", " << b;
}
```

5.5 Develop a functional design for software that meets the following requirements.

The system must

- Read an array of floating point numbers from an ASCII file on disk.
- Compute the average, median, and standard deviation of the numbers.
- Store the average, median, and standard deviation values on disk.

The design should have multiple modules and include the following elements: (a) a structure chart, and (b) a functional description of each module.

5.6 Describe in your own words what is meant by coupling in design. Describe the advantages of both loosely and tightly coupled designs.

5.7 **Project Application.** Develop a functional design for your project. Follow the presentation guidelines in Section 5.9 for communicating the results of the design.