1. Introduction to Embedded Microcomputer Systems

Chapter 1 objectives are to:

• Introduce embedded microcomputer systems,
• Introduce software development,
• Introduce C language development,
• Outline the basic steps in developing microcomputer systems.

An embedded computer system includes a microcomputer mechanical, chemical and electrical devices specific dedicated purpose, and packaged up as a complete system.

• communications,
• automotive,
• military,
• medical,
• consumer,
• machine control.

Each embedded microcomputer system

Jonathan W. Valvano
accepts inputs,  
performs calculations, and  
generates outputs  
runs in “real time.”

In a real time system, upper bound on the time required to  
perform the input/calculation/output  
respond to external events
Because of the real time nature of these systems, we will study the rich set of features built into these microcontrollers to handle all aspects of time.

1.2. Attitude
clients  
coworkers

Test it now.  
Plan for testing.  
Get help.

There is just no simple way to get to the moon.  
Use our creativity to break a complex problem into simple components, rather than developing complex solutions to simple problems.

1.3. Components of an embedded system
embedded "hidden inside so one can’t see it."  
computer  
processor  
nonvolatile
port
device driver

Figure 1.1. An embedded system includes a microcontroller interfaced to external devices.

microcomputer
microcontroller
interface
parallel - data is available simultaneously on groups of lines
serial - binary data is available one bit at a time on a single line
analog - data is encoded as a variable voltage
time - data is encoded as period, frequency, pulse width or phase shift

1.4. Flowcharts
structured programming.

sequence, conditional while-loop.

Figure 1.3. Flowchart showing the basic building blocks of structured programming.

1.5. Product development cycle
new requirements

analyze the problem

high level design

engineering design

implementation

testing

new constraints

• specifications
• constraints

• block diagrams
• data flow graphs

• call graphs
• data structures
• I/O interfaces

not done

done

Figure 1.4. Software development cycle.

data flow graph (dependency graph)

temperature sensor → analog amplifier → ADC

temperature calculation

fixed-point temperature

table lookup

switch

F or C

temperature resistance voltage
digital sample

time

ADC routines

LCD routines

LCD display

Figure 1.5. A data flow graph showing how the temperature signal passes through a simple thermometer.

Call-graphs

Jonathan W. Valvano
Testing
validate basic functions
static efficiency (memory requirements),
dynamic efficiency (execution speed),
accuracy (difference between truth and measured), and
stability (consistent operation.)

Maintenance
correcting mistakes,
adding new features,
optimizing for execution speed or program size,
porting to new computers or operating systems, and
reconfiguring the system to solve a similar problem.

Golden Rule of Software Development
Write software for others as you wish they would write for you.

1.6.2. Qualitative Performance Measurements

Can we prove our software works?
Is our software easy to understand?
Is our software easy to change?

self-documented code,
abstraction,
modularity, and
layered software.

*You can tell if you are a good programmer if*

1) *you can understand your own code 12 months later,*
2) *others can make changes to your code.*