



Steve Heath

Embedded Systems Design

Second Edition

EDN

SERIES FOR DESIGN ENGINEERS



Embedded Systems Design

By the same author

VMEbus: a practical companion

Newnes UNIX™ Pocket Book

Microprocessor architectures: RISC, CISC and DSP

Effective PC networking

PowerPC: a practical companion

The PowerPC Programming Pocket Book

The PC and MAC handbook

The Newnes Windows NT Pocket Book

Multimedia Communications

Essential Linux

Migrating to Windows NT

All books published by Butterworth-Heinemann

About the author:

Through his work with Motorola Semiconductors, the author has been involved in the design and development of microprocessor-based systems since 1982. These designs have included VMEbus systems, microcontrollers, IBM PCs, Apple Macintoshes, and both CISC- and RISC-based multiprocessor systems, while using operating systems as varied as MS-DOS, UNIX, Macintosh OS and real-time kernels.

An avid user of computer systems, he has had over 60 articles and papers published in the electronics press, as well as several books.

Embedded Systems Design

Second edition

Steve Heath



Newnes

OXFORD AMSTERDAM BOSTON LONDON NEW YORK
PARIS SAN DIEGO SAN FRANCISCO SINGAPORE SYDNEY TOKYO

Newnes
An imprint of Elsevier Science
Linacre House, Jordan Hill, Oxford OX2 8DP
200 Wheeler Road, Burlington MA 01803

First published 1997
Reprinted 2000, 2001
Second edition 2003

Copyright © 2003, Steve Heath. All rights reserved

The right of Steve Heath to be identified as the author of this work
has been asserted in accordance with the Copyright, Designs and
Patents Act 1988

No part of this publication may be reproduced in any material form (including photocopying or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication) without the written permission of the copyright holder except in accordance with the provisions of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London, England W1T 4LP. Applications for the copyright holder's written permission to reproduce any part of this publication should be addressed to the publisher

TRADEMARKS/REGISTERED TRADEMARKS

Computer hardware and software brand names mentioned in this book are protected by their respective trademarks and are acknowledged

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication Data

A catalogue record for this book is available from the Library of Congress

ISBN 0 7506 5546 1

Typeset by *Steve Heath*

Contents

Preface	xvii
Acknowledgements	xix
1 What is an embedded system?	1
Replacement for discrete logic-based circuits	2
Provide functional upgrades	3
Provide easy maintenance upgrades	3
Improves mechanical performance	3
Protection of intellectual property	4
Replacement for analogue circuits	4
Inside the embedded system	8
Processor	8
Memory	8
Peripherals	9
Software	10
Algorithms	10
Microcontroller	11
Expanded microcontroller	13
Microprocessor based	14
Board based	14
2 Embedded processors	15
8 bit accumulator processors	16
Register models	16
8 bit data restrictions	17
Addressing memory	18
System integrity	19
Example 8 bit architectures	19
Z80	19
Z80 programming model	21
MC6800	22
Microcontrollers	23
MC68HC05	23
MC68HC11	23
Architecture	25
Data processors	25
Complex instructions, microcode and nanocode	25
INTEL 80286	28
Architecture	28
Interrupt facilities	29
Instruction set	30
80287 floating point support	30
Feature comparison	30

INTEL 80386DX	30
Architecture	30
Interrupt facilities	32
Instruction set	32
80387 floating point coprocessor	33
Feature comparison	33
INTEL 80486	34
Instruction set	35
Intel 486SX and overdrive processors	35
Intel Pentium	36
Multiple branch prediction	38
Data flow analysis	38
Speculative execution	38
The MMX instructions	39
The Pentium II	40
Motorola MC68000	40
The MC68000 hardware	41
Address bus	41
Data bus	41
Function codes	42
Interrupts	43
Error recovery and control signals	44
Motorola MC68020	44
The programmer's model	46
Bus interfaces	49
Motorola MC68030	50
The MC68040	51
The programming model	53
Integrated processors	54
RISC processors	57
The 80/20 rule	57
The initial RISC research	58
The Berkeley RISC model	59
Sun SPARC RISC processor	60
Architecture	60
Interrupts	60
Instruction set	61
The Stanford RISC model	62
The MPC603 block diagram	63
The ARM register set	65
Exceptions	66
The Thumb instructions	67
Digital signal processors	68
DSP basic architecture	69
Choosing a processor	72

3	Memory systems	73
	Memory technologies	74
	DRAM technology	76
	Video RAM	77
	SRAM	77
	Pseudo-static RAM	78
	Battery backed-up SRAM	78
	EPROM and OTP	78
	Flash	79
	EPROM	79
	Memory organisation	79
	By 1 organisation	80
	By 4 organisation	81
	By 8 and by 9 organisations	81
	By 16 and greater organisations	81
	Parity	81
	Parity initialisation	82
	Error detecting and correcting memory	82
	Access times	83
	Packages	83
	Dual in line package	84
	Zig-zag package	84
	SIMM and DIMM	84
	SIP	85
	DRAM interfaces	85
	The basic DRAM interface	85
	Page mode operation	86
	Page interleaving	86
	Burst mode operation	87
	EDO memory	87
	DRAM refresh techniques	88
	Distributed versus burst refresh	88
	Software refresh	89
	RAS only refresh	89
	CAS before RAS (CBR) refresh	89
	Hidden refresh	89
	Memory management	90
	Disadvantages of memory management	92
	Segmentation and paging	93
	Memory protection units	97
	Cache memory	99
	Cache size and organisation	100
	Optimising line length and cache size	104
	Logical versus physical caches	105
	Unified versus Harvard caches	106
	Cache coherency	106

Case 1: write through	108
Case 2: write back	109
Case 3: no caching of write cycles	110
Case 4: write buffer	110
Bus snooping	111
The MESI protocol	116
The MEI protocol	117
Burst interfaces	118
Meeting the interface needs	119
Big and little endian	121
Dual port and shared memory	122
Bank switching	123
Memory overlays	124
Shadowing	124
Example interfaces	125
MC68000 asynchronous bus	125
M6800 synchronous bus	127
The MC68040 burst interface	128
4 Basic peripherals	131
Parallel ports	131
Multi-function I/O ports	132
Pull-up resistors	133
Timer/counters	133
Types	134
8253 timer modes	134
Interrupt on terminal count	134
Programmable one-shot	134
Rate generator	136
Square wave rate generator	136
Software triggered strobe	136
Hardware triggered strobe	137
Generating interrupts	137
MC68230 modes	137
Timer processors	138
Real-time clocks	139
Simulating a real-time clock in software	140
Serial ports	140
Serial peripheral interface	142
I ² C bus	143
Read and write access	145
Addressing peripherals	146
Sending an address index	147
Timing	148

Multi-master support	149
M-Bus (Motorola)	150
What is an RS232 serial port?	151
Asynchronous flow control	154
Modem cables	155
Null modem cables	155
XON-XOFF flow control	158
UART implementations	158
8250/16450/16550	158
The interface signals	159
The Motorola MC68681	162
DMA controllers	163
A generic DMA controller	164
Operation	164
DMA controller models	166
Single address model	166
Dual address model	167
1D model	168
2D model	168
3D model	169
Channels and control blocks	169
Sharing bus bandwidth	171
DMA implementations	173
Intel 8237	173
Motorola MC68300 series	173
Using another CPU with firmware	174
5 Interfacing to the analogue world	175
Analogue to digital conversion techniques	175
Quantisation errors	176
Sample rates and size	176
Irregular sampling errors	177
Nyquist's theorem	179
Codecs	179
Linear	179
A-law and μ -law	179
PCM	180
DPCM	180
ADPCM	181
Power control	181
Matching the drive	181
Using H bridges	183
Driving LEDs	184
Interfacing to relays	184
Interfacing to DC motors	185
Software only	186
Using a single timer	187
Using multiple timers	188

6	Interrupts and exceptions	189
	What is an interrupt?	189
	The spaghetti method	190
	Using interrupts	191
	Interrupt sources	192
	Internal interrupts	192
	External interrupts	192
	Exceptions	192
	Software interrupts	193
	Non-maskable interrupts	193
	Recognising an interrupt	194
	Edge triggered	194
	Level triggered	194
	Maintaining the interrupt	194
	Internal queuing	194
	The interrupt mechanism	195
	Stack-based processors	195
	MC68000 interrupts	196
	RISC exceptions	198
	Synchronous precise	199
	Synchronous imprecise	199
	Asynchronous precise	199
	Asynchronous imprecise	200
	Recognising RISC exceptions	200
	Enabling RISC exceptions	202
	Returning from RISC exceptions	202
	The vector table	202
	Identifying the cause	203
	Fast interrupts	203
	Interrupt controllers	205
	Instruction restart and continuation	205
	Interrupt Latency	206
	Do's and Don'ts	209
	Always expect the unexpected interrupt	209
	Don't expect too much from an interrupt	209
	Use handshaking	210
	Control resource sharing	210
	Beware false interrupts	211
	Controlling interrupt levels	211
	Controlling stacks	211
7	Real-time operating systems	212
	What are operating systems?	212
	Operating system internals	214
	Multitasking operating systems	215
	Context switching, task tables, and kernels	215
	Time slice	223

Pre-emption	224
Co-operative multitasking	224
Scheduler algorithms	225
Rate monotonic	225
Deadline monotonic scheduling	227
Priority guidelines	227
Priority inversion	227
Disabling interrupts	227
Message queues	228
Waiting for a resource	229
VMEbus interrupt messages	229
Fairness systems	231
Tasks, threads and processes	231
Exceptions	232
Memory model	233
Memory allocation	233
Memory characteristics	234
Example memory maps	235
Memory management address translation	239
Bank switching	242
Segmentation	243
Virtual memory	243
Chossoing an operating system	244
Assembler versus high level language	245
ROMable code	245
Scheduling algorithms	245
Pre-emptive scheduling	246
Modular approach	246
Re-entrant code	247
Cross-development platforms	247
Integrated networking	247
Multiprocessor support	247
Commercial operating systems	248
pSOS+	248
pSOS+ kernel	248
pSOS+m multiprocessor kernel	249
pREPC+ runtime support	249
pHILE+ file system	250
pNA+ network manager	250
pROBE+ system level debugger	250
XRAY+ source level debugger	250
OS-9	250
VXWorks	251
VRTX-32	251
IFX	252
TNX	252
RTL	252
RTscope	252
MPV	252
LynxOS-Posix conformance	252
Windows NT	254

Windows NT characteristics	255
Process priorities	256
Interrupt priorities	257
Resource protection	258
Protecting memory	258
Protecting hardware	258
Coping with crashes	259
Multi-threaded software	259
Addressing space	260
Virtual memory	261
The internal architecture	261
Virtual memory manager	262
User and kernel modes	262
Local procedure call (LPC)	263
The kernel	263
File system	263
Network support	264
I/O support	264
HAL approach	264
Linux	265
Origins and beginnings	265
Inside Linux	268
The Linux file system	269
The physical file system	270
Building the file system	271
The file system	272
Disk partitioning	274
The /proc file system	277
Data Caching	277
Multi-tasking systems	278
Multi-user systems	278
Linux software structure	279
Processes and standard I/O	280
Executing commands	281
Physical I/O	282
Memory management	283
Linux limitations	283
eLinux	284
8 Writing software for embedded systems	288
The compilation process	288
Compiling code	289
The pre-processor	290
Compilation	293
as assembler	295
Linking and loading	296
Symbols, references and relocation	296
ld linker/loader	297
Native versus cross-compilers	298
Run-time libraries	298
Processor dependent	298
I/O dependent	299

System calls	299
Exit routines	299
Writing a library	300
Creating a library	300
Device drivers	306
Debugger supplied I/O routines	306
Run-time libraries	307
Using alternative libraries	307
Linking additional libraries	307
Linking replacement libraries	307
Using a standard library	307
Porting kernels	308
Board support	308
Rebuilding kernels for new configurations	309
configAll.h	310
config.h	310
usrConfig.c	310
pSOSystem+	312
C extensions for embedded systems	313
#pragma interrupt func2	313
#pragma pure_function func2	314
#pragma no_side_effects func2	314
#pragma no_return func2	314
#pragma mem_port int2	314
asm and __asm	314
Downloading	316
Serial lines	316
EPROM and FLASH	317
Parallel ports	317
From disk	317
Ethernet	318
Across a common bus	318
9 Emulation and debugging techniques	321
Debugging techniques	321
High level language simulation	321
Low level simulation	322
Onboard debugger	323
Task level debugging	325
Symbolic debug	325
Emulation	327
Optimisation problems	328
Xray	332
The role of the development system	335
Floating point and memory management functions	335
Emulation techniques	336
JTAG	337
OnCE	337
BDM	338

10	Buffering and other data structures	339
	What is a buffer?	339
	Latency	341
	Timing tolerance	341
	Memory size	342
	Code complexity	342
	Linear buffers	342
	Directional buffers	344
	Single buffer implementation	344
	Double buffering	346
	Buffer exchange	348
	Linked lists	349
	FIFOs	350
	Circular buffers	351
	Buffer underrun and overrun	352
	Allocating buffer memory	353
	malloc()	353
	Memory leakage	354
	Stack frame errors	354
	Failure to return memory to the memory pool	355
	Housekeeping errors	355
	Wrong memory specification	356
11	Memory and performance trade-offs	357
	The effect of memory wait states	357
	Scenario 1 — Single cycle processor with large external memory	358
	Scenario 2 — Reducing the cost of memory access	360
	Using registers	360
	Using caches	361
	Preloading caches	362
	Using on-chip memory	363
	Using DMA	363
	Making the right decisions	363
12	Software examples	365
	Benchmark example	365
	Creating software state machines	368
	Priority levels	372
	Explicit locks	373
	Interrupt service routines	373
	Setting priorities	375

Task A highest priority	375
Task C highest priority	376
Using explicit locks	376
Round-robin	376
Using an ISR routine	377
13 Design examples	379
Burglar alarm system	379
Design goals	379
Development strategy	380
Software development	380
Cross-compilation and code generation	383
Porting to the final target system	385
Generation of test modules	385
Target hardware testing	385
Future techniques	385
Relevance to more complex designs	386
The need for emulation	386
Digital echo unit	387
Creating echo and reverb	387
Design requirements	390
Designing the codecs	391
Designing the memory structures	391
The software design	392
Multiple delays	394
Digital or analogue adding	395
Microprocessor selection	396
The overall system design	396
14 Real-time without a RTOS	398
Choosing the software environment	398
Deriving real time performance from a non-real time system	400
Choosing the hardware	401
Scheduling the data sampling	402
Sampling the data	405
Controlling from an external switch	406
Driving an external LED display	408
Testing	408
Problems	410
Saving to hard disk	410
Data size restrictions and the use of a RAM disk	410
Timer calculations and the compiler	411
Data corruption and the need for buffer flushing.	411
Program listing	413
Index	422

This Page Intentionally Left Blank

Preface

The term embedded systems design covers a very wide range of microprocessor designs and does not simply start and end with a simple microcontroller. It can be a PC running software other than Windows and word processing software. It can be a sophisticated multiprocessor design using the fastest processors on the market today.

The common thread to embedded systems design is an understanding of the interaction that the various components within the system have with each other. It is important to understand how the hardware works and the restraints that using a certain peripheral may have on the rest of the system. It is essential to know how to develop the software for such systems and the effect that different hardware designs can have on the software and vice versa. It is this system design knowledge that has been captured in this book as a series of tutorials on the various aspects of embedded systems design.

Chapter 1 defines what is meant by the term and in essence defines the scope of the rest of the book. The second chapter provides a set of tutorials on processor architectures explaining the different philosophies that were used in their design and creation. It covers many of the common processor architectures ranging from 8 bit microcontrollers through CISC and RISC processors and finally ending with digital signal processors and includes information on the ARM processor family.

The third chapter discusses different memory types and their uses. This has been expanded in this edition to cover caches in more detail and the challenges associated with them for embedded design. The next chapter goes through basic peripherals such as parallel and serial ports along with timers and DMA controllers. This theme is continued in the following chapter which covers analogue to digital conversion and basic power control.

Interrupts are covered in great detail in the sixth chapter because they are so essential to any embedded design. The different types that are available and their associated software routines are described with several examples of how to use them and, perhaps more importantly, how not to use them.

The theme of software is continued in the next two chapters which cover real-time operating systems and software development. Again, these have a tremendous effect on embedded designs but whose design implications are often not well understood or explained. Chapter 9 discusses debugging and emulation techniques.

The remaining five chapters are dedicated to design examples covering buffer and data structures, memory and processor performance trade-offs and techniques, software design examples including using a real-time operating system to create state machines and finally a couple of design examples. In this edition, an example real-time system design is described that uses a non-real-time system to create an embedded system. The C source code is provided so that it can be run and experimented with on a PC running MS-DOS.

Steve Heath

Acknowledgements

By the nature of this book, many hardware and software products are identified by their tradenames. In these cases, these designations are claimed as legally protected trademarks by the companies that make these products. It is not the author's nor the publisher's intention to use these names generically, and the reader is cautioned to investigate a trademark before using it as a generic term, rather than a reference to a specific product to which it is attached.

Many of the techniques within this book can destroy data and such techniques must be used with extreme caution. Again, neither author nor publisher assume any responsibility or liability for their use or any results.

While the information contained in this book has been carefully checked for accuracy, the author assumes no responsibility or liability for its use, or any infringement of patents or other rights of third parties which would result.

As technical characteristics are subject to rapid change, the data contained are presented for guidance and education only. For exact detail, consult the relevant standard or manufacturers' data and specification.

This Page Intentionally Left Blank

1

What is an embedded system?

Whenever the word microprocessor is mentioned, it conjures up a picture of a desktop or laptop PC running an application such as a word processor or a spreadsheet. While this is a popular application for microprocessors, it is not the only one and the fact is most people use them indirectly in common objects and appliances without realising it. Without the microprocessor, these products would not be as sophisticated or cheap as they are today.

The embedding of microprocessors into equipment and consumer appliances started before the appearance of the PC and consumes the majority of microprocessors that are made today. In this way, embedded microprocessors are more deeply ingrained into everyday life than any other electronic circuit that is made. A large car may have over 50 microprocessors controlling functions such as the engine through engine management systems, brakes with electronic anti-lock brakes, transmission with traction control and electronically controlled gearboxes, safety with airbag systems, electric windows, air-conditioning and so on. With a well-equipped car, nearly every aspect has some form of electronic control associated with it and thus a need for a microprocessor within an embedded system.

A washing machine may have a microcontroller that contains the different washing programs, provides the power control for the various motors and pumps and even controls the display that tells you how the wash cycles are proceeding.

Mobile phones contain more processing power than a desktop processor of a few years ago. Many toys contain microprocessors and there are even kitchen appliances such as bread machines that use microprocessor-based control systems. The word control is very apt for embedded systems because in virtually every embedded system application, the goal is to control an aspect of a physical system such as temperature, motion, and so on using a variety of inputs. With the recent advent of the digital age replacing many of the analogue technologies in the consumer world, the dominance of the embedded system is ever greater. Each digital consumer device such as a digital camera, DVD or MP3 player all depend on an embedded system to realise the system. As a result, the skills behind embedded systems design are as diverse as the systems that have been built although they share a common heritage.

What is an embedded system?

There are many definitions for this but the best way to define it is to describe it in terms of what it is not and with examples of how it is used.

An embedded system is a microprocessor-based system that is built to control a function or range of functions and is not designed to be programmed by the end user in the same way that a PC is. Yes, a user can make choices concerning functionality but cannot change the functionality of the system by adding/replacing software. With a PC, this is exactly what a user can do: one minute the PC is a word processor and the next it's a games machine simply by changing the software. An embedded system is designed to perform one particular task albeit with choices and different options. The last point is important because it differentiates itself from the world of the PC where the end user does reprogram it whenever a different software package is bought and run. However, PCs have provided an easily accessible source of hardware and software for embedded systems and it should be no surprise that they form the basis of many embedded systems. To reflect this, a very detailed design example is included at the end of this book that uses a PC in this way to build a sophisticated data logging system for a race car.

If this need to control the physical world is so great, what is so special about embedded systems that has led to the widespread use of microprocessors? There are several major reasons and these have increased over the years as the technology has progressed and developed.

Replacement for discrete logic-based circuits

The microprocessor came about almost by accident as a programmable replacement for calculator chips in the 1970s. Up to this point, most control systems using digital logic were implemented using individual logic integrated circuits to create the design and as more functionality became available, the number of chips was reduced.

This was the original reason for a replacement for digital systems constructed from logic circuits. The microprocessor was originally developed to replace a mass of logic that was used to create the first electronic calculators in the early 1970s. For example, the early calculators were made from discrete logic chips and many hundreds were needed just to create a simple four function calculator. As the integrated circuit developed, the individual logic functions were integrated to create higher level functions. Instead of creating an adder from individual logic gates, a complete adder could be bought in one package. It was not long before complete calculators were integrated onto a single chip. This enabled them to be built at a very low cost compared to the original machines but any changes or improvements required that a new