DIGITAL INTERFACE
DESIGN AND
APPLICATION
Contents

List of Figures x

List of Tables xiii

Preface xv

1 Review of Digital Electronics and Computer Architecture 1
  1.1 Embedded Systems 1
    1.1.1 Processor Architecture (Revision) 2
    1.1.2 Interface Subsystem 3
  1.2 Software Architecture 4
  1.3 Essential Basic Logic Elements 5
    1.3.1 The Basic Flip/Flop 5
    1.3.2 The Edge-Triggered D-Type Flip/Flop (Latch) 7
    1.3.3 Edge-Triggered Latch with Enable 8
    1.3.4 Multi-Bit Registers 9
  1.4 Output Configuration Options 10
    1.4.1 Open Drain Configuration 10
  1.5 The Address Decode 11
    1.5.1 Partial Address Decode 12
  1.6 ARM Architecture 14
  1.7 Interface Software Development 14
    1.7.1 Software Development for Embedded Systems 18
1.8 C Programming Revision
   1.8.1 Arrays 19
   1.8.2 Structures and typedef 21
   1.8.3 Header Files 21

1.9 Conclusion 22
References 23
Further Reading 23

2 Simple Input and Output Functions 24

2.1 Introduction 24
2.2 Computer Structure 25
2.3 Simple Interface Circuit Concepts 26
   2.3.1 An Output Interface 26
   2.3.2 Address Decode for Output 28
   2.3.3 A Simple Input Interface 29
   2.3.4 Address Decode for Input 29

2.4 Activation of I/O Circuits 30
   2.4.1 Programming an Output 30
   2.4.2 Programming an Input 31

2.5 Universal I/O Circuits 31
   2.5.1 Combined I/O Address Decode 32

2.6 Practical I/O Circuits 33
   2.6.1 STM32F4 Address Decoding 35

2.7 A Typical I/O Programme 35
   2.7.1 Example GPIO Application 37
   2.7.2 A Summary of Alternative I/O Operations 40
   2.7.3 Programming I/O in Assembler Language 41

2.8 Suggested Design Challenge 41
2.9 Conclusion 43
References 44
Further Reading 44

3 Timer Subsystems 45

3.1 Timer Subsystems 45
3.2 Basic Timer Configuration 46
3.3 The STM32F4 Timers 47
   3.3.1 The Individual Timers 50
3.4 Programming the STM32F4 Timers 51
3.5 Timer Triggering 55
3.5.1 Setting up the Time-Base 55
3.5.2 Using the Timer for an Input Measurement 56
3.6 Basic Timers 58
3.7 PWM Applications 61
3.8 Programming Challenge 63
3.9 Conclusion 64
References 65

4 Analogue Interface Subsystems 66
4.1 Analogue Interfaces 66
4.2 Digital to Analogue 67
  4.2.1 The STM32F4 DAC 69
4.3 Analogue to Digital Conversion 69
  4.3.1 Sampling 70
  4.3.2 Switched Capacitor Converter 72
  4.3.3 The Software Interface 73
  4.3.4 The STM32F4 ADC 74
4.4 Software Control of DAC 75
  4.4.1 Waveform Generation 76
  4.4.2 Waveform Timing 77
  4.4.3 DAC Using DMA 79
4.5 Software Control of ADC 83
  4.5.1 ADC Interface Using Timer and DMA 85
4.6 Programming Challenge 88
4.7 Conclusion 89
References 89
Further Reading 89

5 Serial Interface Subsystems 90
5.1 Introduction 90
5.2 RS232 Universal Asynchronous Receiver/Transmitter (UART) Communications 91
5.3 The I2C Interface 95
  5.3.1 Using the Touch Screen with an I2C Interface 96
5.4 SPI Interface 101
  5.4.1 SPI Interface to an Analogue to Digital Converter 103
5.5 HDLC Serial Communication 105
5.6 The Universal Serial Bus (USB) 107
  5.6.1 Hand-shake Packets 109
7.3 Decoding GPS Signals 153
    7.3.1 Acquiring the GPS Message 154
    7.3.2 Decoding the GPS Message 155
    7.3.3 Selecting the Message Stream 158
7.4 Conclusion 159
References 160

Appendix A: uVision IDE Notes 161
    A.1 Getting Started 161
    A.2 Help 162
    A.3 Project Development 162
    A.4 Debug Facilities 164
    A.5 Conclusion 167

Appendix B: STM Discovery Examples Library 168
    B.1 Peripheral Examples 168
    B.2 Example Application 171

Appendix C: DAC and ADC Support Software 175
    C.1 DAC Peripheral Features 175
    C.2 How to Use the DAC Driver 176
    C.3 ADC Peripheral Features 177
    C.4 How to Use the ADC driver 177
    C.5 Files for Reference 178

Appendix D: Example Keyboard Interface 179

Index 185
List of Figures

Figure 1.1  Fundamental computer architecture 2
Figure 1.2  Interface software structure 4
Figure 1.3  Cross-coupled flip/flop circuit 6
Figure 1.4  Uncoupled circuit 6
Figure 1.5  D-type edge-triggered latch 7
Figure 1.6  Timing diagram 7
Figure 1.7  Flip/flop with enable functionality 8
Figure 1.8  Four-bit register with group input enable IE and group output enable OE 9
Figure 1.9  Push/pull output driver 10
Figure 1.10 Wired-AND connection of multiple outputs 11
Figure 1.11 Full address decode example 12
Figure 1.12 Partial address decode 13
Figure 1.13 Block decode 13
Figure 1.14 Screen-shot for uVision4 IDE 15
Figure 1.15 IDE management window 20
Figure 2.1 Fundamental processor structure 25
Figure 2.2 Bus timing relationships 26
Figure 2.3 Basic latch 27
Figure 2.4 A simple output interface 27
Figure 2.5 Simple decode for output 28
Figure 2.6 A tri-state buffer 29
Figure 2.7 A simple input interface circuit 29
Figure 2.8 Combined I/O 32
Figure 2.9 GPIO pin circuit 34
Figure 2.10 Observed code output pattern 39
Figure 2.11 16-key matrix keyboard 42
Figure 2.12 Keyboard interface connections 43
Figure 2.13 Timing diagram for keyboard interface 43
Figure 3.1 A simple timer function 47
Figure 3.2 The simplified timer architecture 48
Figure 3.3 Upward count mode 49
Figure 3.4 Downward count mode 49
Figure 3.5 Centre aligned mode 49
Figure 3.6 Timing diagram for example code 54
Figure 3.7 Capture/compare input circuit 57
Figure 3.8 Simplified timers 59
Figure 3.9 A PWM signal set at 40% 61
Figure 3.10 Servo control with PWM 64
Figure 3.11 Servo controller circuit diagram 64
Figure 4.1 Binary weighted DAC network 68
Figure 4.2 DAC block diagram 68
Figure 4.3 A 3-bit DAC transfer characteristic 69
Figure 4.4 Simple ADC subsystem design 70
Figure 4.5 Sample and hold 71
Figure 4.6 Sampling a sine wave 71
Figure 4.7 Switched capacitor converter 72
Figure 4.8 ADC interface timing 73
Figure 4.9 Triangle waveform 76
Figure 4.10 Timer triggering of DAC 77
Figure 4.11 Temperature sensor circuit 88
Figure 5.1 Manchester code 91
Figure 5.2 RS232 data format 92
Figure 5.3 UART module 92
Figure 5.4 An I2C interface circuit 95
Figure 5.5 I2C read and write modes 96
Figure 5.6 SPI interface connections 102
Figure 5.7 AD7680 connections 104
Figure 5.8 HDLC frame structure 106
Figure 5.9 HDLC frame control byte 106
Figure 5.10 USB connections 107
Figure 5.11 Touch screen interface 111
Figure 6.1 Interrupt basic concepts 113
Figure 6.2  Part of the NVIC 115
Figure 6.3  Simple processor with DMA 118
Figure 6.4  DMA controller architecture 119
Figure 6.5  An upper-case A character 126
Figure 6.6  The WiFi module configuration 132
Figure 6.7  Digital camera interface (DCMI) configuration 133
Figure 7.1  The compass interface 136
Figure 7.2  MSF signal modulation format 140
Figure 7.3  MSF receiver circuit 141
Figure 7.4  GPS module connection 153
Figure A.1  The uVision screen (part) 162
Figure A.2  A typical help screen 163
Figure A.3  Context sensitive help 163
Figure A.4  Project file structure 164
Figure A.5  Project compiling and linking 164
Figure A.6  A debug run 165
Figure A.7  Debug tools 165
Figure A.8  A system viewer window 166
Figure A.9  Timer 3 registers 166
Figure A.10 Watch windows 167
List of Tables

Table 1.1 Memory map 3
Table 1.2 Truth table for the simple flip/flop 6
Table 1.3 Truth table for edge-triggered latch 7
Table 1.4 Memory map 13
Table 1.5 Code interpretation 18
Table 2.1 Output address allocation 28
Table 2.2 Input address allocation 30
Table 2.3 I/O address allocation 33
Table 2.4 STM32F4 address allocations 35
Table 2.5 Programme linked comments 37
Table 4.1 DAC output allocation 76
Table 4.2 ADC input allocations 84
Table 5.1 STMP811 system control register 2 98
Table 5.2 Temperature gauge control register 98
Table 5.3 USB PID byte functions 108
Table 6.1 An extract of the vector table; the full table has more than 81 entries! 114
Table 6.2 NVIC priority groups 116
Table 6.3 DMA allocated channels 120
Table 6.4 Interface port mappings 122
Table 6.5 Pixel assignments 125
Table 6.6 Demonstration functions 132
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 7.1</td>
<td>MAG3110 register summary</td>
<td>137</td>
</tr>
<tr>
<td>Table 7.2</td>
<td>MSF data bit allocations</td>
<td>141</td>
</tr>
<tr>
<td>Table 7.3</td>
<td>GPGGA navigation message format</td>
<td>154</td>
</tr>
</tbody>
</table>
Preface

This book aims to provide a justified and well founded cornerstone in the development of techniques required to establish reliable interface designs used when embedded computers are deployed in any demanding application. The book will focus on the ARM Microprocessor, which is now a leading technology in the electronics industry and offers a wide range of performance optimised features for particular applications. By using simple practical examples, the link between the embedded hardware and the programming task will be clearly developed so that interface design can be undertaken with confidence and the resulting systems will exhibit high reliability.

ARM Microprocessors have developed quickly over the last few years and have been very widely adopted by the electronics industry, finding pervasive applications in mobile phones, sensor networks and server systems to pick just a few examples. Applications will continue to develop and become more diverse over the years to come so a wide knowledge background in interface design will be highly valuable.

This book will bring together aspects of digital hardware, interface design and software integration (not otherwise available in a single text) in a progressive arrangement to promote thorough comprehension of the examples. In particular, the intimate linkage between low and high level languages (HLLs) will be explained so that the advantages of optimisation can be considered carefully. In many cases, the HLL approach will deliver rapid productivity but performance optimisation will require a more detailed knowledge of the overheads involved.
Readers should have a basic knowledge of digital electronics such as that established in the early years of an undergraduate degree course. The book is aimed at any student studying the requirements of interface design and although starting at an introductory level it will also provide a reference work for those involved in a wide variety of interface projects. Comprehensive details will be provided to enable access for the widest possible readership including those with a more software oriented background.

The author has over 25 years’ experience of teaching microprocessors and interfacing at York University, and previously, Sheffield Hallam University, although this was known as Sheffield Polytechnic at the time. Prior to his teaching career, the author worked for several industrial organisations mainly focussing on various projects involving microprocessors, digital electronics and computer automated electronic test systems. These teaching and development experiences have provided the author with an extensive knowledge of microcomputer architectures and interface design, which has enabled the development of this book.

The author’s earliest work with microprocessors used a very inflexible four-bit single-chip computer produced by Texas Instruments. This had no integrated peripherals and a very awkward architecture based on its optimisation for a pocket calculator role. Programming this chip involved using raw machine code and some limited assembler facilities. Fortunately, this was soon superseded by 8 and 16 bit architectures that offered full assembler programming support but still required many additional components to provide memory and peripheral resources. These types of system required considerable hardware and software development to incorporate all the elements that would be needed for a particular application. The introduction of HLL compilers and integrated development systems (IDE) provided significant advantages in productivity for system developers and higher levels of hardware integration allowed more extensive programmable interface resources to be included so removing the need for extensive hardware design. The current availability of systems based on the ARM processor and integrated peripherals, such as those found in the STM product STM32F4, together with its IDE software package, bring together all the advantages offering amazing capabilities and diverse application possibilities with rapid development.

The author has dealt with the issue of learning about many different processor developments over his career but has always found that a small program designed to flash a single LED was the best way to make a rapid start with the new technology. This approach can be recommended wholeheartedly.

Jonathan A. Dell
University of York, UK
15 October 2014
This chapter will focus on the operation of basic logic elements, which should be familiar to most readers, such as flip/flops and registers that form the basic building blocks for interface elements. It will also cover the issue of address decoding to enable these elements through a programme statement and provide an introduction to the ARM architecture and its built-in peripherals. Finally the linkage between C and low-level assembler code will be shown through a simple example.

1.1 Embedded Systems

Embedded systems take many different forms but all rely on some computing processor whether it has a physical interface with the outside world like the keyboard and screen of a typical PC, the physical connections forming a communication network or specialised sensor and actuator hardware to control an automated machine. The other essential element of any system making use of a computer is its controlling program and in fact most embedded system component vendors provide integrated development systems or environments (IDE) that run on a PC to facilitate the software development. Interface design requires careful assessment of both the hardware and software requirements
so that when the objectives are considered the most effective solution can be achieved. In summary, we can say that an embedded system is computerised and tailor-made for a particular application.

1.1.1 Processor Architecture (Revision)

It is important to have a clear understanding of the processor architecture so that the integration of interface peripherals can be appreciated. The fundamental hardware architecture of the computing element will be examined briefly to show where interface components of any variety link up. Figure 1.1 shows a simple architecture of processor and memory that is widely adopted; this is interconnected by address, data and control buses. It is assumed that the reader is familiar with the general operation of the processor in terms of the machine instructions and data held in the memory as well as the cyclic operations of instruction fetch and execution to perform the desired task. The interface circuits connect onto the same buses, so as far as the processor is concerned the interfaces look like an extension to the memory. An advantage of this arrangement is that the same processor instructions can be used to manipulate data in memory or interface values.

As a reminder, the function of the address bus is to differentiate all the different elements of the system, as no two parts can have the same address or a conflict and confusion will arise. It is usual to use a memory map in order to show how the range offered by the address bus is allocated to different parts of the system. Table 1.1 shows how memory and interface elements might be allocated in a simple system with a 16-bit address bus.

The data bus has an obvious function but it should be noted that its operation is bi-directional; except in special circumstances data is sent to the processor or delivered by it. The most important task of the control bus is to indicate the data bus flow direction and thus avoid a bus conflict. Memory and interface values.

Figure 1.1  Fundamental computer architecture
elements can then be arranged to perform the appropriate read or write function as demanded. In a complex processor like the ARM the control bus may have many other functions to accommodate such as direct memory access (DMA), where the processor operation is temporarily suspended, and interrupt, which enable hardware signals and specific software instructions to be linked. These special functions will be examined in more detail in a later chapter.

The binary machine instructions held in the memory for the processor to access are usually determined from a high-level programming language, like C, for example by a Compiler but this approach will always involve some redundancy or overhead that may form an unacceptable burden in some demanding situations where memory is limited or processing time is at a premium. So for greater efficiency a low-level assembler language, which has a close linkage with each machine instruction, is sometimes employed. Unfortunately the assembler language approach introduces considerable complexity and is associated with low programming productivity, so it should only be considered for the most demanding applications.

1.1.2 Interface Subsystem

When an interface circuit is designed it is essential to link up with the bus signals using appropriate hardware circuits. In essence, the address bus must connect with an address decode function to create latch enable signals and the data bus must connect with output from registers or special input functions. Finally, the control line, representing data bus direction, is usually combined with the address decode logic so that an output function or input function can be enabled correctly when bus access is demanded. This is quite a complex task for an interface designer but some typical circuits will be discussed in more detail in the following sections to show the principles involved. In fact, the STM32F4 has many such interface subsystems integrated within the microprocessor chip itself making their application more straightforward for the user, removing a complex task for the designer. The resulting address map

<table>
<thead>
<tr>
<th>Address range</th>
<th>System element</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xc000 to 0xffff</td>
<td>I/O interface</td>
</tr>
<tr>
<td>0xb000 to 0xbfff</td>
<td>Redundant space</td>
</tr>
<tr>
<td>0x9000 to 0xafff</td>
<td>12k Data memory</td>
</tr>
<tr>
<td>0x0000 to 0x8fff</td>
<td>36k Programme memory</td>
</tr>
</tbody>
</table>