Contents

Preface xviii
Acknowledgements xxi
Foreword xxii
About the Companion Website xxiii

1 Introduction to DSP 1
  1.1 Introduction 1
  1.2 Multicore processors 3
    1.2.1 Can any algorithm benefit from a multicore processor? 3
    1.2.2 How many cores do I need for my application? 5
  1.3 Key applications of high-performance multicore devices 6
  1.4 FPGAs, Multicore DSPs, GPUs and Multicore CPUs 8
  1.5 Challenges faced for programming a multicore processor 9
  1.6 Texas Instruments DSP roadmap 10
  1.7 Conclusion 11
  References 12

2 The TMS320C66x architecture overview 14
  2.1 Overview 14
  2.2 The CPU 15
    2.2.1 Cross paths 16
      2.2.1.1 Data cross paths 17
      2.2.1.2 Address cross paths 18
    2.2.2 Register file A and file B 20
      2.2.2.1 Operands 20
      2.2.3 Functional units 21
        2.2.3.1 Condition registers 21
        2.2.3.2 L units 22
        2.2.3.3 M units 22
        2.2.3.4 S units 23
        2.2.3.5 D units 23
    2.3 Single instruction, multiple data (SIMD) instructions 24
      2.3.1 Control registers 24
    2.4 The KeyStone memory 24
      2.4.1 Using the internal memory 27
      2.4.2 Memory protection and extension 29
      2.4.3 Memory throughput 29
2.5 Peripherals 30
  2.5.1 Navigator 32
  2.5.2 Enhanced Direct Memory Access (EDMA) Controller 32
  2.5.3 Universal Asynchronous Receiver/Transmitter (UART) 32
  2.5.4 General purpose input–output (GPIO) 32
  2.5.5 Internal timers 32
  2.6 Conclusion 33

References 33

3 Software development tools and the TMS320C6678 EVM 35
  3.1 Introduction 35
  3.2 Software development tools 37
    3.2.1 Compiler 38
    3.2.2 Assembler 39
    3.2.3 Linker 40
      3.2.3.1 Linker command file 40
    3.2.4 Compile, assemble and link 42
    3.2.5 Using the Real-Time Software Components (RTSC) tools 42
      3.2.5.1 Platform update using the XDCtools 42
    3.2.6 KeyStone Multicore Software Development Kit 47
  3.3 Hardware development tools 47
  3.3.1 EVM features 47
  3.4 Laboratory experiments based on the C6678 EVM: introduction to Code Composer Studio (CCS) 51
    3.4.1 Software and hardware requirements 51
      3.4.1.1 Key features 52
      3.4.1.2 Download sites 53
    3.4.2 Laboratory experiments with the CCS6 53
      3.4.2.1 Introduction to CCS 55
      3.4.2.2 Implementation of a DOTP algorithm 63
    3.4.3 Profiling using the clock 65
    3.4.4 Considerations when measuring time 67
  3.5 Loading different applications to different cores 67
  3.6 Conclusion 72

References 72

4 Numerical issues 74
  4.1 Introduction 74
  4.2 Fixed- and floating-point representations 75
    4.2.1 Fixed-point arithmetic 76
      4.2.1.1 Unsigned integer 76
      4.2.1.2 Signed integer 77
      4.2.1.3 Fractional numbers 77
    4.2.2 Floating-point arithmetic 78
      4.2.2.1 Special numbers for the 32-bit and 64-bit floating-point formats 81
    4.3 Dynamic range and accuracy 82
  4.4 Laboratory exercise 83
  4.5 Conclusion 85

References 85
5 Software optimisation  86
5.1 Introduction  86
5.2 Hindrance to software scalability for a multicore processor  88
5.3 Single-core code optimisation procedure  88
5.3.1 The C compiler options  90
5.4 Interfacing C with intrinsics, linear assembly and assembly  91
5.4.1 Intrinsics  91
5.4.2 Interfacing C and assembly  92
5.5 Assembly optimisation  97
5.5.1 Parallel instructions  98
5.5.2 Removing the NOPs  99
5.5.3 Loop unrolling  99
5.5.4 Double-Word Access  100
5.5.5 Optimisation summary  100
5.6 Software pipelining  101
5.6.1 Software-pipelining procedure  105
5.6.1.1 Writing linear assembly code  105
5.6.1.2 Creating a dependency graph  105
5.6.1.3 Resource allocation  108
5.6.1.4 Scheduling table  108
5.6.1.5 Generating assembly code  109
5.7 Linear assembly  111
5.7.1 Hand optimisation of the dotp function using linear assembly  112
5.8 Avoiding memory banks  118
5.9 Optimisation using the tools  118
5.10 Laboratory experiments  123
5.11 Conclusion  126
References  126

6 The TMS320C66x interrupts  127
6.1 Introduction  127
6.1.1 Chip-level interrupt controller  129
6.2 The interrupt controller  135
6.3 Laboratory experiment  140
6.3.1 Experiment 1: Using the GIPIOs to trigger some functions  140
6.3.2 Experiment 2: Using the console to trigger an interrupt  140
6.4 Conclusion  143
References  144

7 Real-time operating system: TI-RTOS  145
7.1 Introduction  146
7.2 TI-RTOS  146
7.3 Real-time scheduling  148
7.3.1 Hardware interrupts (Hwis)  148
7.3.1.1 Setting an Hwi  149
7.3.1.2 Hwi hook functions  149
7.3.2 Software interrupts (Swis), including clock, periodic or single-shot functions  155
7.3.3 Tasks  155
7.3.3.1 Task hook functions  157
References  144
9 Inter-Processor Communication (IPC) 214

9.1 Introduction 215
9.2 Texas Instruments IPC 217
9.3 Notify module 219
9.3.1 Laboratory experiment 222
9.4 MessageQ 222
9.4.1 MessageQ protocol 224
9.4.2 Message priority 229
9.4.3 Thread synchronisation 229
9.5 ListMP module 233
9.6 GateMP module 234
9.6.1 Initialising a GateMP parameter structure 234
9.6.1.1 Types of gate protection 235
9.6.2 Creating a GateMP instance 236
9.6.3 Entering a GateMP 236
9.6.4 Leaving a gate 236
9.6.5 The list of functions that can be used by GateMP 237
9.7 Multi-processor Memory Allocation: HeapBufMP, HeapMemMP and HeapMultiBufMP 237
9.7.1 HeapBuf_Params 238
9.7.2 HeapMem_Params 239
9.7.3 HeapMultiBuf_Params 239
9.7.4 Configuration example for HeapMultiBuf 239
9.8 Transport mechanisms for the IPC 241
9.9 Laboratory experiments with KeyStone I 241
9.9.1 Laboratory 1: Using MessageQ with multiple cores 241
9.9.1.1 Overview 242
9.9.2 Laboratory 2: Using ListMP, ShareRegion and GateMP 243
9.10 Laboratory experiments with KeyStone II 249
9.10.1 Laboratory experiment 1: Transferring a block of data 249
9.10.1.1 Set the connection between the host (PC) and the KeyStone 249
9.10.1.2 Explore the ARM code 250
9.10.1.3 Explore the DSP code 259
9.10.1.4 Compile and run the program 263
9.10.2 Laboratory experiment 2: Transferring a pointer 267
9.10.2.1 Explore the ARM code 267
9.10.2.2 Explore the DSP code 271
9.10.2.3 Compile and run the program 278
9.11 Conclusion 278
References 278
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Single and multicore debugging</td>
<td>280</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td>281</td>
</tr>
<tr>
<td>10.2</td>
<td>Software and hardware debugging</td>
<td>282</td>
</tr>
<tr>
<td>10.3</td>
<td>Debug architecture</td>
<td>282</td>
</tr>
<tr>
<td>10.3.1</td>
<td>Trace</td>
<td>282</td>
</tr>
<tr>
<td>10.3.1.1</td>
<td>Standard trace</td>
<td>282</td>
</tr>
<tr>
<td>10.3.1.2</td>
<td>Event trace</td>
<td>283</td>
</tr>
<tr>
<td>10.3.1.3</td>
<td>System trace</td>
<td>285</td>
</tr>
<tr>
<td>10.4</td>
<td>Advanced Event Triggering</td>
<td>286</td>
</tr>
<tr>
<td>10.4.1</td>
<td>Advanced Event Triggering logic</td>
<td>289</td>
</tr>
<tr>
<td>10.5</td>
<td>Unified Instrumentation Architecture</td>
<td>295</td>
</tr>
<tr>
<td>10.5.1</td>
<td>Host-side tooling</td>
<td>295</td>
</tr>
<tr>
<td>10.5.2</td>
<td>Target-side tooling</td>
<td>295</td>
</tr>
<tr>
<td>10.5.2.1</td>
<td>Software instrumentation APIs</td>
<td>297</td>
</tr>
<tr>
<td>10.5.2.2</td>
<td>Predefined software events and metadata</td>
<td>297</td>
</tr>
<tr>
<td>10.5.2.3</td>
<td>Event loggers</td>
<td>297</td>
</tr>
<tr>
<td>10.5.2.4</td>
<td>Transports</td>
<td>297</td>
</tr>
<tr>
<td>10.5.2.5</td>
<td>SYS/BIOS event capture and transport</td>
<td>297</td>
</tr>
<tr>
<td>10.5.2.6</td>
<td>Multicore support</td>
<td>297</td>
</tr>
<tr>
<td>10.6</td>
<td>Debugging with the System Analyzer tools</td>
<td>298</td>
</tr>
<tr>
<td>10.6.1</td>
<td>Target-side coding with UIA APIs and the XDCtools</td>
<td>299</td>
</tr>
<tr>
<td>10.6.2</td>
<td>Logging events with Log_write() functions</td>
<td>300</td>
</tr>
<tr>
<td>10.6.3</td>
<td>Advance debugging using the diagnostic feature</td>
<td>301</td>
</tr>
<tr>
<td>10.6.4</td>
<td>LogSnapshot APIs for logging state information</td>
<td>302</td>
</tr>
<tr>
<td>10.7</td>
<td>Instrumentation with TI-RTOS and CCS</td>
<td>302</td>
</tr>
<tr>
<td>10.7.1</td>
<td>Using RTOS Object Viewer</td>
<td>302</td>
</tr>
<tr>
<td>10.7.2</td>
<td>Using the RTOS Analyzer and the System Analyzer</td>
<td>303</td>
</tr>
<tr>
<td>10.7.2.1</td>
<td>RTOS Analyzer</td>
<td>303</td>
</tr>
<tr>
<td>10.7.2.2</td>
<td>System Analyzer</td>
<td>303</td>
</tr>
<tr>
<td>10.8</td>
<td>Laboratory sessions</td>
<td>305</td>
</tr>
<tr>
<td>10.8.1</td>
<td>Laboratory experiment 1: Using the RTOS ROV</td>
<td>305</td>
</tr>
<tr>
<td>10.8.2</td>
<td>Laboratory experiment 2: Using the RTOS Analyzer</td>
<td>305</td>
</tr>
<tr>
<td>10.8.3</td>
<td>Laboratory experiment 3: Using the System Analyzer</td>
<td>312</td>
</tr>
<tr>
<td>10.8.4</td>
<td>Laboratory experiment 4: Using diagnosis features</td>
<td>314</td>
</tr>
<tr>
<td>10.8.5</td>
<td>Laboratory experiment 5: Using a diagnostic feature with filtering</td>
<td>317</td>
</tr>
<tr>
<td>10.9</td>
<td>Conclusion</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>Further reading</td>
<td>323</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Bootloader for KeyStone I and KeyStone II</td>
<td>324</td>
</tr>
<tr>
<td>11.1</td>
<td>Introduction</td>
<td>324</td>
</tr>
<tr>
<td>11.2</td>
<td>How to start the boot process</td>
<td>325</td>
</tr>
<tr>
<td>11.3</td>
<td>The boot process</td>
<td>325</td>
</tr>
<tr>
<td>11.4</td>
<td>ROM Bootloader (RBL)</td>
<td>328</td>
</tr>
<tr>
<td>11.4.1</td>
<td>The boot configuration format</td>
<td>336</td>
</tr>
<tr>
<td>11.4.1.1</td>
<td>Creating the boot parameter table</td>
<td>336</td>
</tr>
<tr>
<td>11.4.1.2</td>
<td>Creating the boot table</td>
<td>338</td>
</tr>
</tbody>
</table>
11.4.1.3 The boot configuration table 338
11.5 Boot process 340
11.5.1 Initialisation stage for the KeyStone I 340
11.5.2 Second-level bootloader 341
11.5.2.1 Intermediate bootloader 341
11.5.2.2 How to use the IBL 342
11.6 Laboratory experiment 1 345
11.6.1 Initialisation stage for the KeyStone II 350
11.6.1.1 Bootloader initialisation after power-on reset 350
11.6.1.2 Bootloader initialisation process after hard or soft reset 350
11.6.2 Second bootloader for the KeyStone II 350
11.6.2.1 U-Boot 351
11.7 Laboratory experiment 2 352
11.7.1 Printing the U-Boot environment 360
11.7.2 Using the help for U-Boot 362
11.8 TFTP boot with a host-mounted Network File System (NFS) server – NFS booting 363
11.8.1 Laboratory experiment 3 364
11.9 Conclusion 372
References 372

12 Introduction to OpenMP 374
12.1 Introduction to OpenMP 375
12.2 Directive formats 376
12.3 Forking region 377
12.3.1 omp parallel – parallel region construct 377
12.3.1.1 Clause descriptions 378
12.4 Work-sharing constructs 382
12.4.1 omp for 382
12.4.1.1 OpenMP loop scheduling 383
12.4.2 omp sections 385
12.4.3 omp single 386
12.4.4 omp master 386
12.4.5 omp task 387
12.5 Environment variables and library functions 390
12.6 Synchronisation constructs 392
12.6.1 atomic 393
12.6.1.1 Clauses 393
12.6.2 barrier 395
12.6.3 critical 396
12.7 OpenMP accelerator model 397
12.7.1 Supported OpenMP device constructs 397
12.7.1.1 #pragma omp target 397
12.7.1.2 #pragma omp target data 399
12.7.1.3 #pragma omp target update 400
12.7.1.4 #pragma omp declare target 401
12.8 Laboratory experiments 402
12.8.1 Laboratory experiment 1 402
13 Introduction to OpenCL for the KeyStone II

13.1 Introduction
13.2 Operation of OpenCL
13.3 Command queue
13.3.1 Creating a command queue
13.3.1.1 Command-queue properties
13.3.2 Enqueuing a kernel
13.4 Kernel declaration
13.5 How do the kernels access data?
13.6 OpenCL memory model for the KeyStone
13.6.1 Creating a buffer
13.6.1.1 Cl_mem_flags
13.7 Synchronisation
13.7.1 Event with a callback function
13.7.2 User event
13.7.3 Waiting for one command or all commands to finish
13.7.4 wait_group_events
13.7.5 Barrier
13.8 Basic debugging profiling
13.9 OpenMP dispatch from OpenCL
13.9.1 OpenMP for the kernel code
13.9.2 OpenMP for the ARM code
13.10 Building the OpenCL project
13.11 Laboratory experiments
13.11.1 Laboratory experiment 1: Hello World
13.11.2 Laboratory experiment 2: dotp functions
13.11.2.1 Explore the main.cpp function
13.11.2.2 Explore the kernel dotp.cl
13.11.2.3 Run the dotp program
13.11.3 Laboratory experiment 3: USE_HOST_PTR
13.11.4 Laboratory experiment 4: ALLOC_HOST_PTR
13.11.5 Laboratory experiment 5: COPY_HOST_PTR
13.11.6 Laboratory experiment 6: Synchronisation
13.11.7 Laboratory experiment 7: Local buffer
13.11.8 Laboratory experiment 8: Barrier
13.11.9 Laboratory experiment 9: Profiling
13.11.10 Laboratory experiment 10: OpenMP in kernel
13.11.11 Laboratory experiment 11: OpenMP in ARM
13.12 Conclusion
16 IIR filter implementation 542
16.1 Introduction 542
16.2 Design procedure 543
16.3 Coefficients calculation 543
16.3.1 Pole–zero placement approach 543
16.3.2 Analogue-to-digital filter design 543
16.3.3 Bilinear transform (BZT) method 544
16.3.3.1 Practical example of the bilinear transform method 547
16.3.3.2 Coefficients calculation 547
16.3.3.3 Realisation structures 548
16.3.4 Impulse invariant method 552
16.3.4.1 Practical example of the impulse invariant method 553
16.4 IIR filter implementation 556
16.5 Laboratory experiment 561
16.6 Conclusion 561
Reference 562

17 Adaptive filter implementation 563
17.1 Introduction 563
17.2 Mean square error 564
17.3 Least mean square 565
17.4 Implementation of an adaptive filter using the LMS algorithm 565
17.5 Implementation using linear assembly 567
17.6 Implementation in C language with compiler switches 572
17.7 Laboratory experiment 572
17.8 Conclusion 573
References 573

18 FFT implementation 574
18.1 Introduction 574
18.2 FFT algorithm 574
18.2.1 Fourier series 574
18.2.2 Fourier transform 575
18.2.3 Discrete Fourier transform 575
18.2.4 Fast Fourier transform 576
18.2.4.1 Splitting the DFT into two DFTs 576
18.2.4.2 Exploiting the periodicity and symmetry of the twiddle factors 577
18.3 FFT implementation 579
18.4 Laboratory experiment 582
18.4.1 Part 1: Implementation of DIF FFT 582
18.4.2 Part 2: Using ping-pong EDMA 585
18.5 Conclusion 590
References 590
<table>
<thead>
<tr>
<th>19</th>
<th>Hough transform 591</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.1</td>
<td>Introduction 591</td>
</tr>
<tr>
<td>19.2</td>
<td>Theory 591</td>
</tr>
<tr>
<td>19.3</td>
<td>Limits of $r$ and $\theta$ 593</td>
</tr>
<tr>
<td>19.4</td>
<td>Hough transform implementation 595</td>
</tr>
<tr>
<td>19.5</td>
<td>Laboratory experiment 596</td>
</tr>
<tr>
<td>19.6</td>
<td>Conclusion 603</td>
</tr>
<tr>
<td></td>
<td>References 603</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>20</th>
<th>Stereo vision implementation 604</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.1</td>
<td>Introduction 604</td>
</tr>
<tr>
<td>20.2</td>
<td>Algorithm for performing depth calculation 605</td>
</tr>
<tr>
<td>20.3</td>
<td>Cost functions 606</td>
</tr>
<tr>
<td>20.4</td>
<td>Implementation 607</td>
</tr>
<tr>
<td>20.4.1</td>
<td>Laboratory experiment 610</td>
</tr>
<tr>
<td>20.4.1.1</td>
<td>SAD implementation 610</td>
</tr>
<tr>
<td>20.4.1.2</td>
<td>NCC implementation 611</td>
</tr>
<tr>
<td>20.4.1.3</td>
<td>ZNCC implementation 611</td>
</tr>
<tr>
<td>20.5</td>
<td>Conclusion 613</td>
</tr>
<tr>
<td></td>
<td>References 616</td>
</tr>
</tbody>
</table>

| Index 617 |