- Preview document showing TOC, sample images, and software project summary.
- Colour images are used in this preview, although final print version will be black and white. Colour images only available in relevant ebook formats, and at supported resolutions.
- Final TOC and page count likely to change.

TABLE OF CONTENTS

1 Zynq-7000 Adaptive SoC Fundamentals 17

1.1 Introduction 17

1.2 The Evolution of the Xilinx Zynq-7000 Adaptive SoC 21
1.2.1 The Pre-Embedded Era (1985-1999) 23
1.2.2 Platform FPGA Era (2000-2009) 26
1.2.3 The Zynq Era (2010-) 29

1.3 A Simplified Overview of the Zynq-7000 32
1.3.1 ARM Cortex-A9 MPCore 33
1.3.2 Memory Sub-System 35
1.3.3 I/O Peripherals 38
1.3.4 Interrupt Controller 38
1.3.5 Timing Options 39

1.4 Introduction to the Programmable Logic 39
1.4.1 Configurable Logic Block 41
1.4.2 Memory Options 41
1.4.3 Digital Signal Processing 41
1.4.4 SelectIO 41
1.4.5 Serial Transceivers 42
1.4.6 Clock Management 42
1.4.7 System Monitoring (XADC) 42

1.5 Zynq-7000 Development Tools 43
1.5.1 A Broad History of Xilinx Development Tools 43
1.5.2 Tools Needed for Zynq-7000 Bare-Metal Development 45

1.6 Essential Xilinx Documentation 46

1.7 A Summary of Book Topics 47

1.8 References 50

2 ARM Cortex-A9 Architecture 55

2.1 Introduction 55

2.2 A Brief History of ARM Processors 56

2.3 The Evolution of RISC 58

2.4 ARM Processor Fundamentals: ARM7TDMI 60
2.4.1 ARM7TDMI Main Features 60
2.4.2 ARM7 3-Stage Pipeline 62

2.4.3 Pipeline Hazards 64

2.5 ARM Cortex-A9 65
2.5.1 Micro-architecture Features 66
2.5.2 Instruction Sets 68
2.5.3 Processor State 68
2.5.4 Endianness 69
2.5.5 Modes and Privilege Levels 69
2.5.6 Register Set 72
2.5.7 Exception Handling 73

2.6 ARM Cortex-A9 MPCore 77
2.6.1 Snoop Control Unit 78
2.6.2 Level 2 Memory Interface 78
2.6.3 Accelerator Coherency Port 79
2.6.4 Generic Interrupt Controller 79
2.6.5 MPCore Timers 80

2.7 ARM Cortex-A9 Debug Fundamentals 80
2.7.1 Invasive Debugging 80
2.7.2 Non-Invasive Debugging 81
2.7.3 Embedded Cross Trigger (ECT) 82

2.8 References 82

3 Zynq-7000 Adaptive SoC Architecture 85

3.1 Introduction 85

3.2 Application Processing Unit 87
3.2.1 ARM Cortex-A9 MPCore 87
3.2.2 APU Memory Features 88
3.2.3 Direct Memory Access 88
3.2.4 2x Triple Timer Counters 88
3.2.5 System Watchdog Timer 89
3.2.6 System-Level Control Registers 89

3.3 Processing System 89
3.3.1 Memory Interfaces 89
3.3.2 I/O Peripherals and MIO/EMIO 91
3.3.3 Clock and Reset Generation 92
3.3.4 Processing System Interconnect 95

3.4 Programmable Logic 96
3.4.1 Fundamental XC7Z020 Structure 97
3.4.2 Configurable Logic Block 99
3.4.3 Clock Management 101
3.4.4 Block RAM 102
3.4.5 DSP 104
3.4.6 SelectIO 105
3.4.7 Multi-Gigabit Transceivers and PCIe 107
3.4.8 XADC Dual 12-Bit 1MSPS Analog-to-Digital Converter 108

3.5 PS-PL Interconnect **110**
3.5.1 Clocking and Reset 111
3.5.2 Interrupts 111
3.5.3 AXI Interconnect 111
3.5.4 DMA Access 112
3.5.5 Extended Multiplexed IO (EMIO) 112
3.5.6 Debug and Trace 113
3.5.7 Event/Miscellaneous Signals 113

3.6 Boot and Configuration **113**
3.6.1 Booting an Operating System 114
3.6.2 Booting a Bare-Metal Application (External Device) 116
3.6.3 JTAG Development Mode 117
3.6.4 Configuration and Boot Circuit 117

3.7 Debug System Overview **119**
3.7.1 The Zynq-7000 CoreSight System 120

3.8 Memory Map **128**

3.9 References **130**
3.9.1 Xilinx 7-Series User Guides 131

4 Digilent Zybo-Z7-20 Platform 133

4.1 Introduction **133**

4.2 Power Distribution Network **135**

4.3 Clock and Reset Circuits **137**
4.3.1 Clock Options 137
4.3.2 Resets 138

4.4 Processing System Interfaces **139**
4.4.1 Memory Interfaces 139
4.4.2 Board Communications - UART 142
4.4.3 Processing System GPIO 144
4.4.4 USB 146
4.4.5 Gigabit Ethernet 146
4.4.6 MIO Pin-Out Summary 147

4.5 Programmable Logic Interfaces **149**
4.5.1 Programmable Logic GPIO 150
4.5.2 XADC 152
4.5.3 Video Interfaces 153
4.5.4 Audio Interface 155

4.6 Boot and Configuration **156**

4.7 References **157**

5 Bare-Metal Development Flow 159

5.1 Introduction 159

5.2 Hardware (FPGA) Development 160
5.2.1 Fundamental Hardware Flow 160
5.2.2 Modern-day RTL Flow 162
5.2.3 Zynq-7000 HW Design Flow: IP Integrator 164

5.3 Hardware Hand-off 166

5.4 Software Development Flow 167
5.4.1 Development in SDK 167
5.4.2 Zynq-7000 Development using the Vitis software platform 171
5.4.3 Bare-Metal Device Driver Architecture 176

5.5 References 182

6 Design Entry Using Vivado 185

6.1 Introduction 185

6.2 Board-Level Requirements 186
6.2.1 Processing System GPIO 186
6.2.2 Programmable Logic GPIO 186
6.2.3 Programmable Logic Pmod ACL 188
6.2.4 Board Communications/Debug 188
6.2.5 Board Power 189
6.2.6 Boot and Configuration Options 189

6.3 System-Level Requirements 189
6.3.1 Clocking/Reset 189
6.3.2 Main Memory 189
6.3.3 Boot/Configuration 190
6.3.4 Development and Debug 190
6.3.5 Platform Communications 190
6.3.6 Processing System GPIO 190
6.3.7 Triple Timer Counters 191
6.3.8 Programmable Logic 191

6.4 PS - PL Interface 191
6.4.1 Clocking/Reset 191
6.4.2 AXI Interface 192
6.4.3 Triple-Timer Counters to EMIO 192
6.4.4 PL Interrupts 192

6.5 Programmable Logic Design 193
6.5.1 Fundamental PL Design 193
6.5.2 IP Integrator Programmable Logic Circuit 195

6.6 Zynq-7000 Processor Configuration 196
6.6.1 PS-PL Configuration Tab 197
6.6.2 Peripheral I/O Pins Tab 198
6.6.3 MIO Configuration Tab 198

6.6.4 Clock Configuration Tab 199
6.6.5 Interrupts Tab 199
6.6.6 Remaining Configuration Tabs 199
6.6.7 Board Pin-out 199

6.7 Building the Design in IP Integrator 201
6.7.1 Step 1. Create the Block Design and Add Zynq-7000 IP 202
6.7.2 Step 2. Run Block Automation for Zynq-7000 IP 203
6.7.3 Step 3. Configure Zynq-7000 with Project-Specific Settings 205
6.7.4 Step 4. Add Zynq-7000 PS Block Diagram Pins 206
6.7.5 Step 5. Add AXI GPIO IP and Configure 207
6.7.6 Step 6. Run Connection Automation for GPIO IP 209
6.7.7 Step 7. Validate the Design 210
6.7.8 Step 8. Add AXI Quad SPI IP and Configure 211
6.7.9 Step 9. Validate Design - Error Detected! 213
6.7.10 Step 10. Add Pmod ACL Interrupt Logic 214
6.7.11 Step 11. Manually Connect Concat IP 215
6.7.12 Step 12. Generate Output Products 216
6.7.13 Step 13. Create HDL Wrapper 218
6.7.14 Step 14. Create or Add Constraints File 219
6.7.15 Step 15-17. Synthesis, Implementation and Generate Bitstream 220
6.7.16 Step 18. Export Hardware 221
6.7.17 Step 19. Launch SDK 222

6.8 References 223
6.8.1 General References 223

7 Software Development in SDK 225

7.1 Introduction 225

7.2 Development Flow in SDK 225
7.2.1 Import the Hardware Platform 226
7.2.2 Create Bare-Metal BSP 227
7.2.3 Create Bare-Metal Application 228
7.2.4 Build the Application 231
7.2.5 Run the Application 232
7.2.6 Debug the Application 234

7.3 Xilinx Software Command Line Tool 245
7.3.1 Getting Started 246
7.3.2 Procedure 247

7.4 Software Projects to Come 252

7.5 References 255

8 Zynq-7000 GPIO 257

8.1 Introduction 257

8.2 Main Concepts 258
8.2.1 Processing System (PS7) GPIO 258

8.2.2 AXI GPIO 259
8.2.3 Driver Initialisation 259
8.2.4 Enabling Assertion Functionality 264
8.2.5 Main Program Structure 265

8.3 Project Details 267
8.3.1 System Overview 267
8.3.2 Zynq-7000 Block Diagram 268
8.3.3 Software Details 269

8.4 Code Discussion 270
8.4.1 Flat Single-File Structure 270
8.4.2 Header Section 270
8.4.3 Main Code Section 274
8.4.4 Function Definitions 278

8.5 Running the Program 286

8.6 References 287

9 Structured Programming in C 289

9.1 Introduction 289

9.2 Main Concepts 291
9.2.1 Code Hierarchy 291
9.2.2 Re-designing the Single-File Project 293

9.3 Code Discussion 295
9.3.1 Driver Interface Layer (AXI GPIO) 295
9.3.2 Top-Level Layer 298

9.4 Running the Program 304

10 Zynq-7000 Timers and Watchdogs 305

10.1 Introduction 305

10.2 Main Concepts 307
10.2.1 MPCore Timers 308
10.2.2 Watchdog Timers 310
10.2.3 Program Timing 311

10.3 Project Details 312
10.3.1 System Overview 313
10.3.2 Zynq-7000 Block Diagram 314
10.3.3 Software Details 314
10.3.4 Test Connections 315

10.4 Program Hierarchy Updates 316

10.5 Code Discussion 318
10.5.1 Updates to the Driver Interface Layer 318
10.5.2 Updates to Top-level Layer 323

10.6 Running the Program **329**
10.6.1 Normal Board Power-up 329
10.6.2 Test the Watchdog Timer 330
10.6.3 Test the System Reset Button Functionality 331
10.6.4 Viewing Test Signals on a Logic Analyser 332
10.7 References **334**

11 The Zynq-7000 Interrupt System 335
11.1 Introduction **335**
11.2 Main Concepts **337**
11.2.1 ARM Generic Interrupt Controller 337
11.2.2 Preemption 344
11.2.3 Zynq-7000 Interrupt System 344
11.2.4 Timed Interrupts 347
11.2.5 Foreground-Background Architecture 348
11.3 Project Details **349**
11.3.1 System Overview 349
11.3.2 Zynq-7000 Block Diagram 350
11.3.3 Software Details 351
11.3.4 Test Connections 352
11.4 Program Hierarchy Updates **354**
11.4.1 Adding Interrupt Functionality Using Xilinx Drivers 354
11.4.2 Complete Hierarchy Update Details 355
11.5 Code Discussion **355**
11.5.1 Updates to Driver Interface Layer 356
11.5.2 Updates to the Top-Level Layer 360
11.6 Running the Program **373**
11.6.1 Viewing Test Signals on a Logic Analyser 374
11.7 References **376**

12 Program Timing Using The TTC 377
12.1 Introduction **377**
12.2 Main Concepts **378**
12.2.1 Zynq-7000 Triple Timer Counters 378
12.2.2 Task Timing 382
12.3 Project Details **383**
12.3.1 System Overview 384
12.3.2 Zynq-7000 Block Diagram 385
12.3.3 Software Details 386
12.3.4 Test Connections 386
12.4 Program Hierarchy Updates **388**
12.5 Code Discussion **389**

12.5.1 Updates to the Driver Interface Layer 390
12.5.2 Updates to Top-Level Layer 395
12.6 Running the Program 399
12.6.1 Viewing Test Signals on a Logic Analyser 400
12.7 References 404

13 Button Debouncing in Software 405
13.1 Introduction 405
13.2 Main Concepts 407
13.2.1 Debounce Algorithm 408
13.3 Project Details 409
13.3.1 Project Requirements and Support Materials 409
13.3.2 System Overview 410
13.3.3 Zynq-7000 Block Diagram 411
13.3.4 Software Details 412
13.3.5 Test Connections 413
13.4 Code Discussion 414
13.4.1 Changes to Driver Interface Files 415
13.4.2 Changes to Top-Level Files 415
13.5 Running the Program 419
13.5.1 Viewing Test Signals on a Logic Analyser 420
13.6 References 423

14 UART Command Handler Design 425
14.1 Introduction 425
14.2 Main Concepts 426
14.2.1 Defining a Simple Protocol 426
14.2.2 Adding Commands 428
14.2.3 Command Handler Design 430
14.2.4 UART Communications Block 431
14.2.5 Program Flow Updates 432
14.3 Project Details 434
14.3.1 System Overview 434
14.3.2 Zynq-7000 Block Diagram 435
14.3.3 Software Details 435
14.3.4 Test Connections 437
14.4 Program Hierarchy Updates 438
14.5 Code Discussion 440
14.5.1 Updates to Driver Interface Layer 440
14.5.2 Updates to Top-Level Layer 449

14.6 Running the Program 453
14.7 References 455

15 Host Application Design 457
15.1 Introduction 457
15.2 Design Approach 458
15.3 Python Code (Jupyter) 459
15.3.1 Import Packages 459
15.3.2 Define the Python Functions 460
15.3.3 Running the Code 462
15.3.4 Defining Specific Commands 464
15.4 LabVIEW Code 466
15.4.1 Individual Sub-VIs 466
15.4.2 Creating the LabVIEW Application 471
15.4.3 Viewing Test Signals on a Logic Analyser 474
15.5 References 476

16 Programmable Logic Interrupts 477
16.1 Introduction 477
16.2 Main Concepts 479
16.2.1 Basic Design Specification 479
16.2.2 Pmod ACL Configuration 480
16.2.3 Pmod ACL SPI Communications 482
16.2.4 Host PC Application Updates 484
16.2.5 Embedded System Design 487
16.3 Project Details 488
16.3.1 System Overview 489
16.3.2 Zynq-7000 Block Diagram 489
16.3.3 Software Details 491
16.3.4 Test Connections 492
16.4 Program Hierarchy Updates 493
16.5 Code Discussion 494
16.5.1 Updates to the Driver Interface Layer 495
16.5.2 Updates to the Command Handling Code 502
16.5.3 Pmod ACL Command Example 505
16.5.4 Updates to the Top-Level Layer 506
16.6 Updating the Host Applications 510
16.6.1 Python (Jupyter) 510
16.7 Running the Program 513
16.7.1 Viewing Test Signals on a Logic Analyser 514
16.8 References 518

17 Platform Boot Options 519

17.1 Introduction 519

17.2 Zynq-7000 Bare-Metal Boot Image 520
17.2.1 Boot Header Table 522
17.2.2 Image Header Tables 523
17.2.3 Partition Tables 523
17.2.4 Images 524

17.3 Bare-Metal Boot Sequence 525
17.3.1 POR/Restart 525
17.3.2 Stage 0: BootROM 525
17.3.3 Stage 1: FSBL 527

17.4 External Boot on the Zybo-Z7-20 Platform 527
17.4.1 Preliminary Step: Enable Fat File System Library in BSP 529
17.4.2 Create FSBL Application Project 531
17.4.3 SD Card Procedure 533
17.4.4 QSPI Procedure 536

17.5 References 540
17.5.1 General References 540

18 Additional Interrupt Topics 541

18.1 Introduction 541

18.2 Nested Interrupts 542
18.2.1 Main Concepts 542
18.2.2 Program Hierarchy Updates 547
18.2.3 Code Discussion 549

18.3 Interrupts and Shared Variables 551
18.3.1 Main Concepts 551
18.3.2 Program Hierarchy Updates 552
18.3.3 Code Discussion 554

18.4 Running the Program 559
18.4.1 Nested Interrupts 560
18.4.2 Shared Variables Test 561

18.5 References 562

1. An Introduction to the Zynq-7000 APSoC

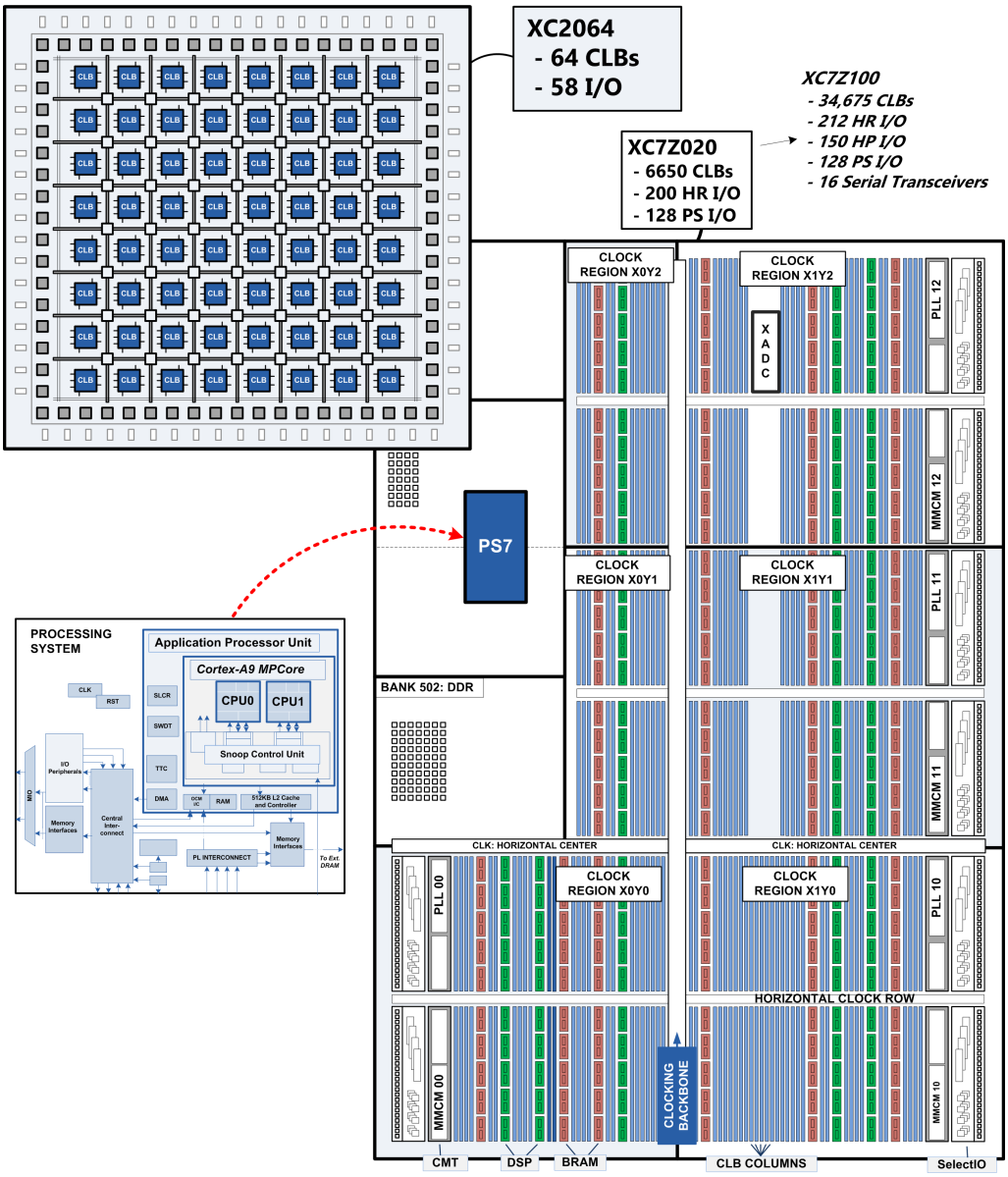

Figure 1. Layout comparison of the Xilinx XC2064, which was the first Xilinx FPGA, and the Xilinx XC7Z020, a mid-range Zynq device.

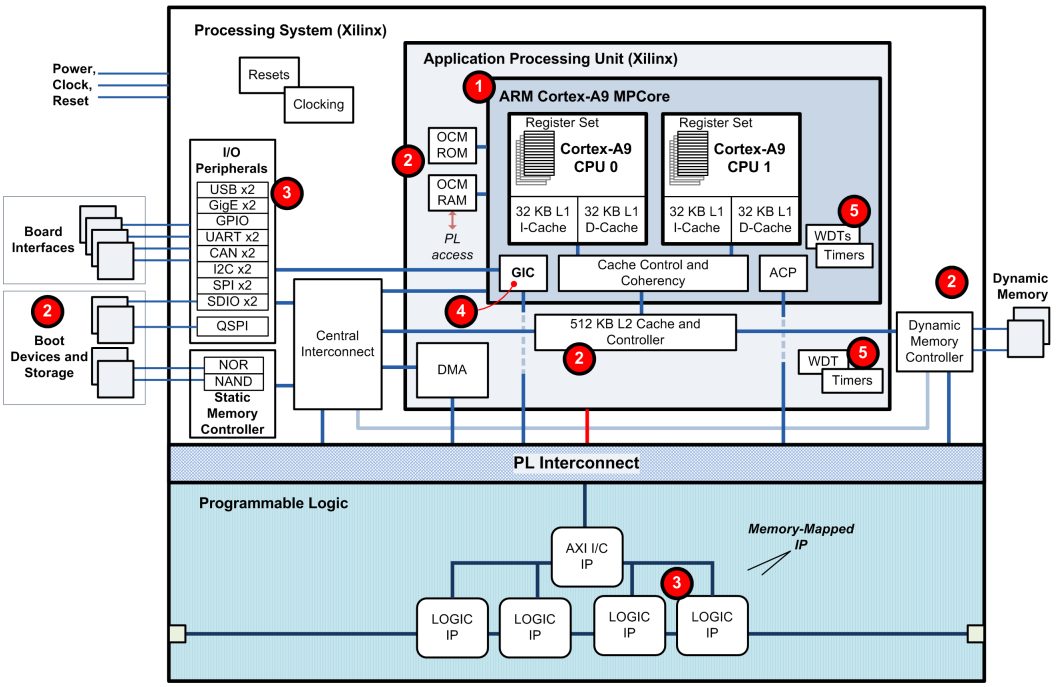

Figure 2. Simplified block diagram of the Zynq-7000, showing the main components which make up an embedded system.

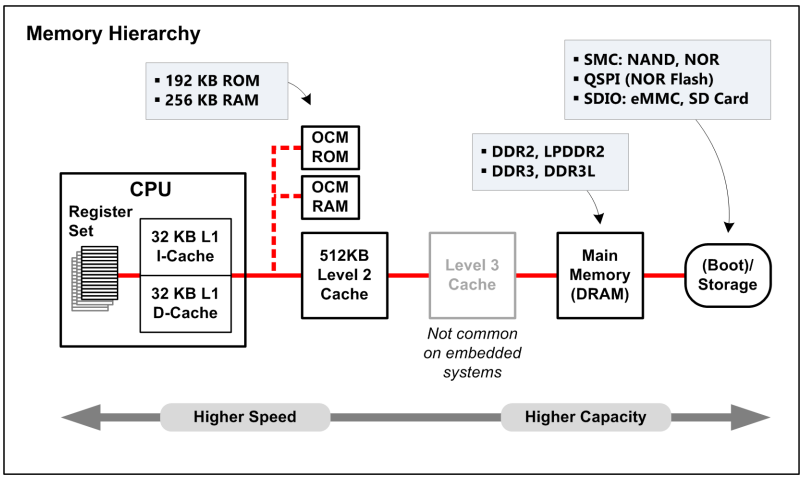

Figure 3. Typical memory hierarchy for an application-profile embedded system.

2. The ARM Cortex-A9

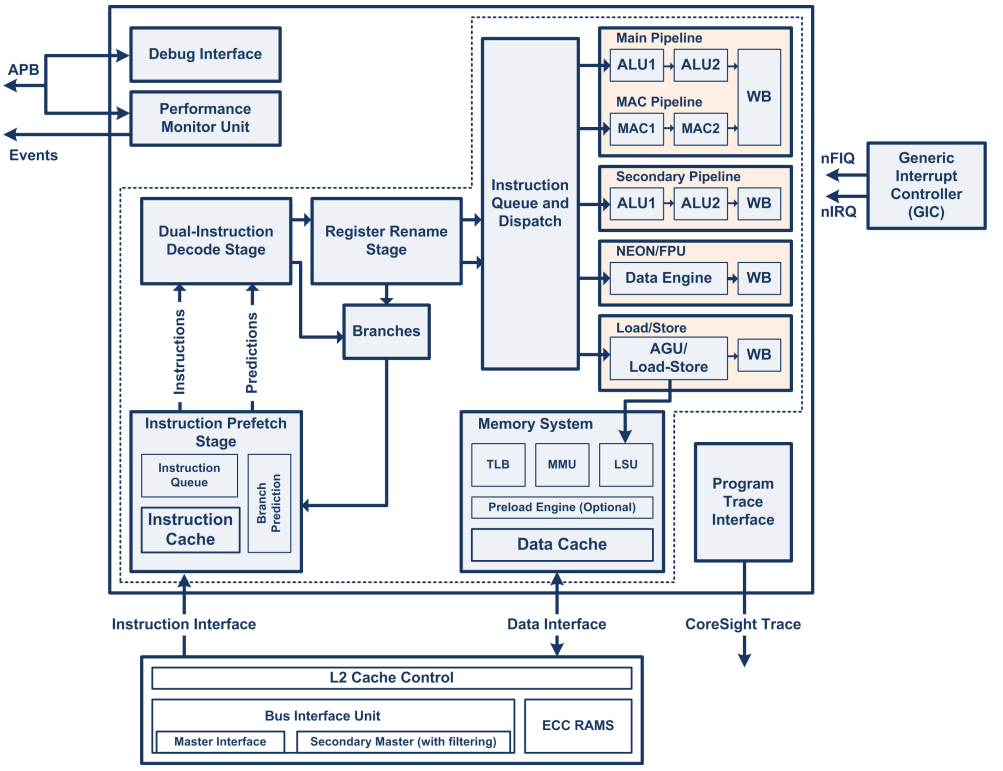

Figure 4. ARM Cortex-A9 block diagram

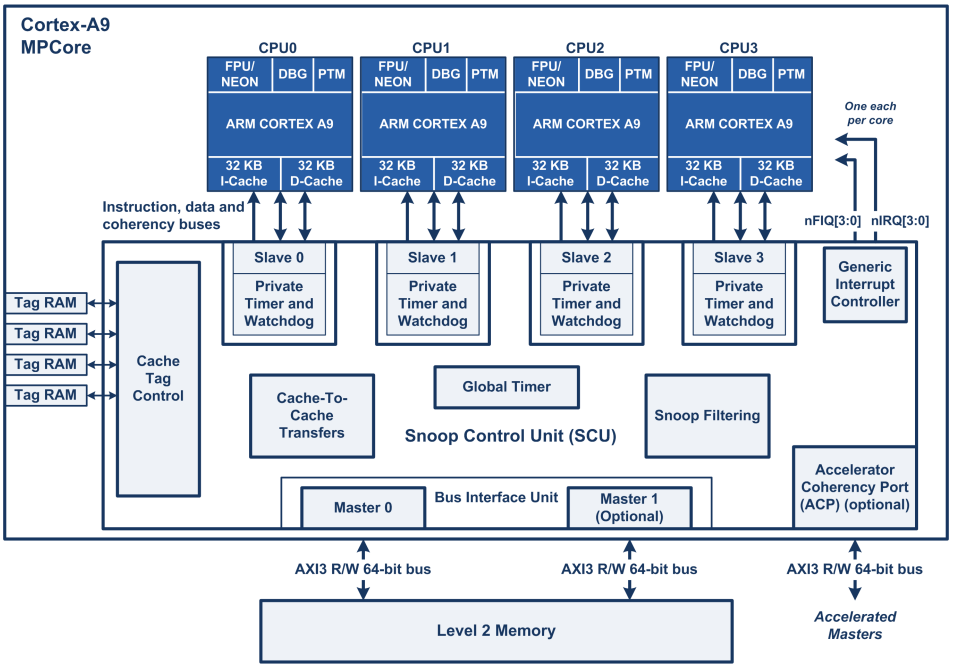

Figure 5. ARM MPCore block diagram

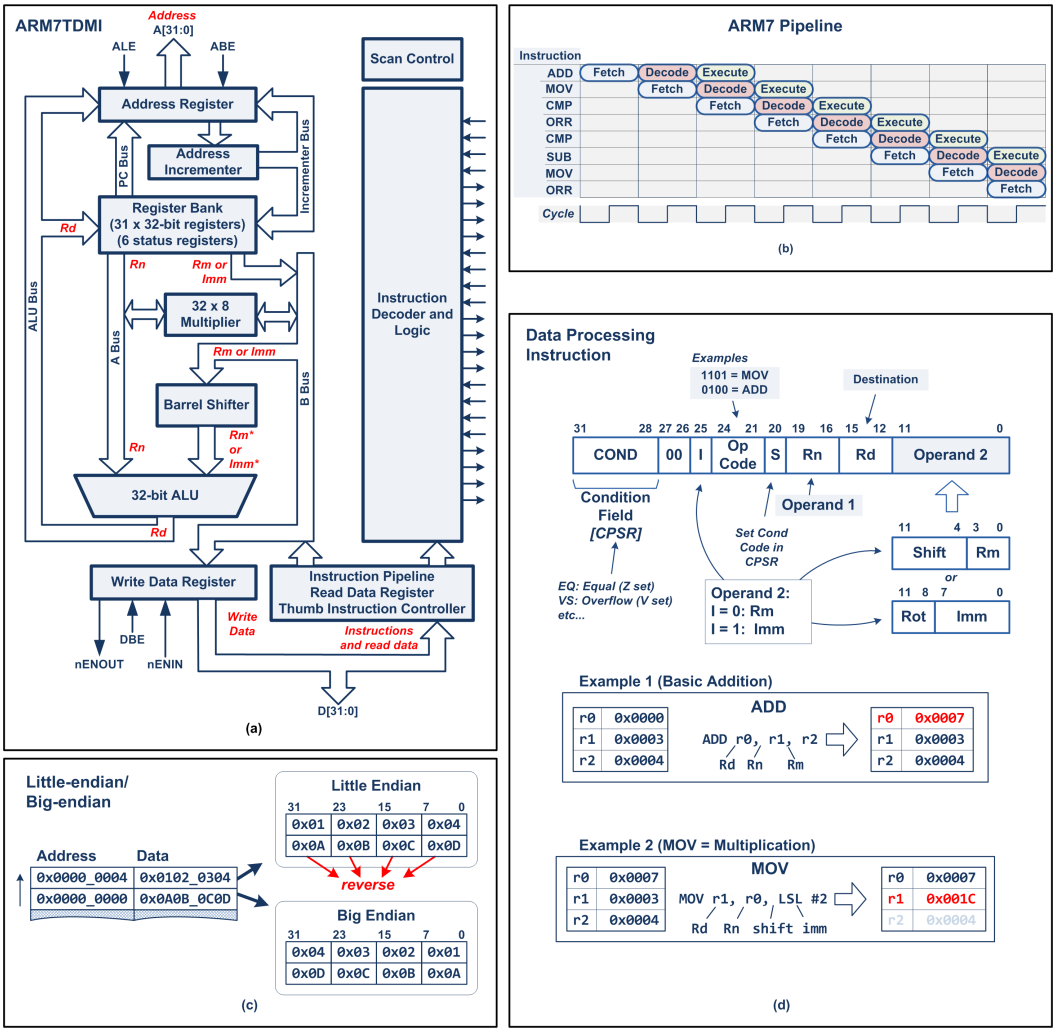

Figure 6. (a) ARM7TDMI main processor logic. (b) ARM7 Pipeline. (c) Little-endian/big-endian formats. (d) Example ARM ISA instruction: Data processing, with ADD and MOV op codes shown.

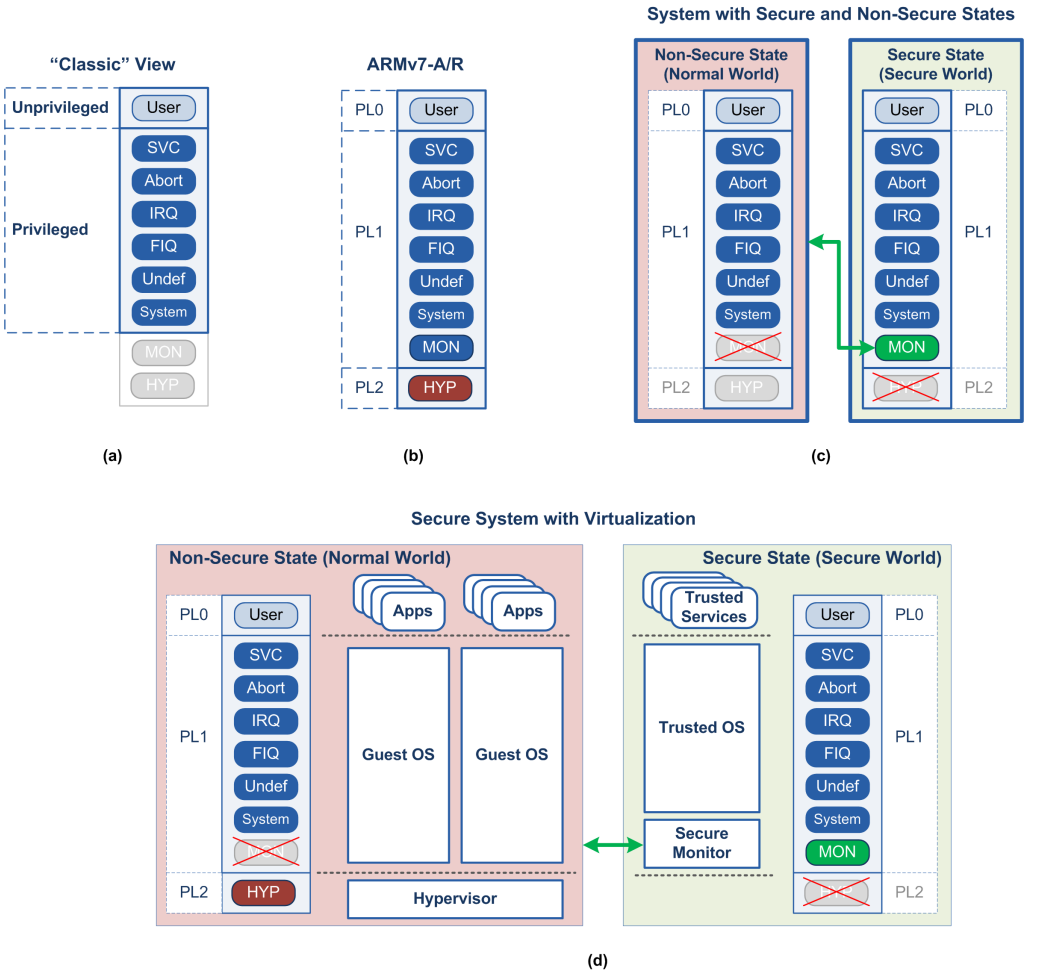

Figure 7. ARM modes and privilege levels. (a) Classic view. (b) ARMv7-A/R view. (c) Secure and non-secure states in ARMv7-A. (d) Secure system with virtualization.

3. Zynq-7000 APSoC

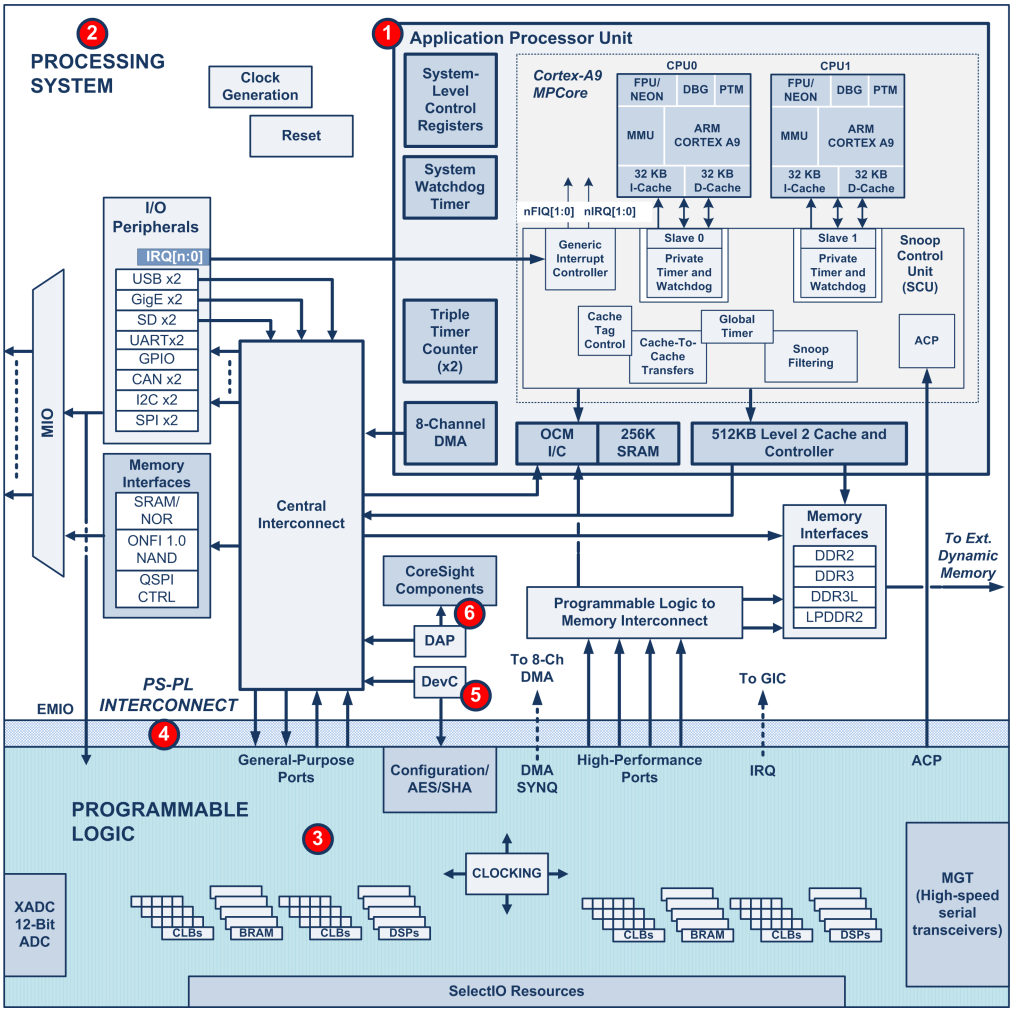

Figure 8. Simplified block diagram of the Zynq-7000 Adaptive SoC, illustrating the topics to be covered in this chapter.

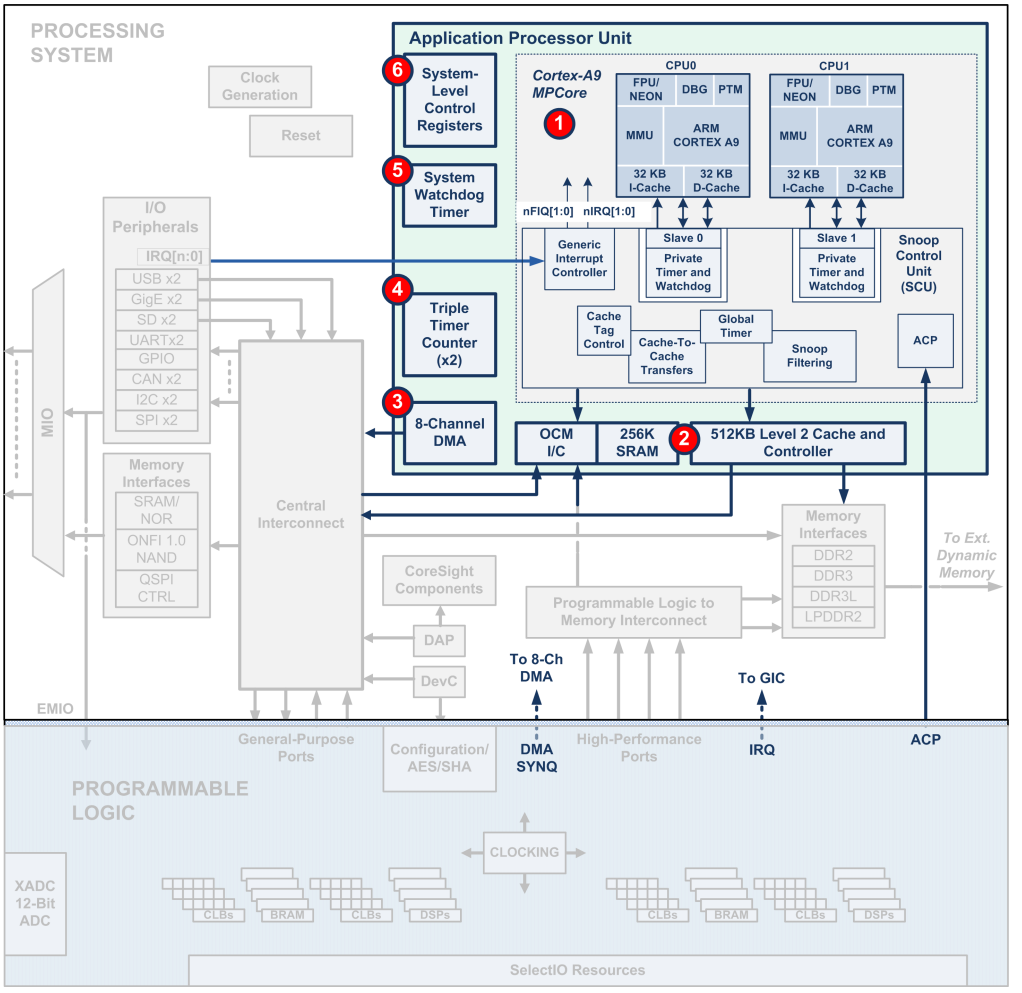

Figure 9. Zynq-7000 Application Processing Unit

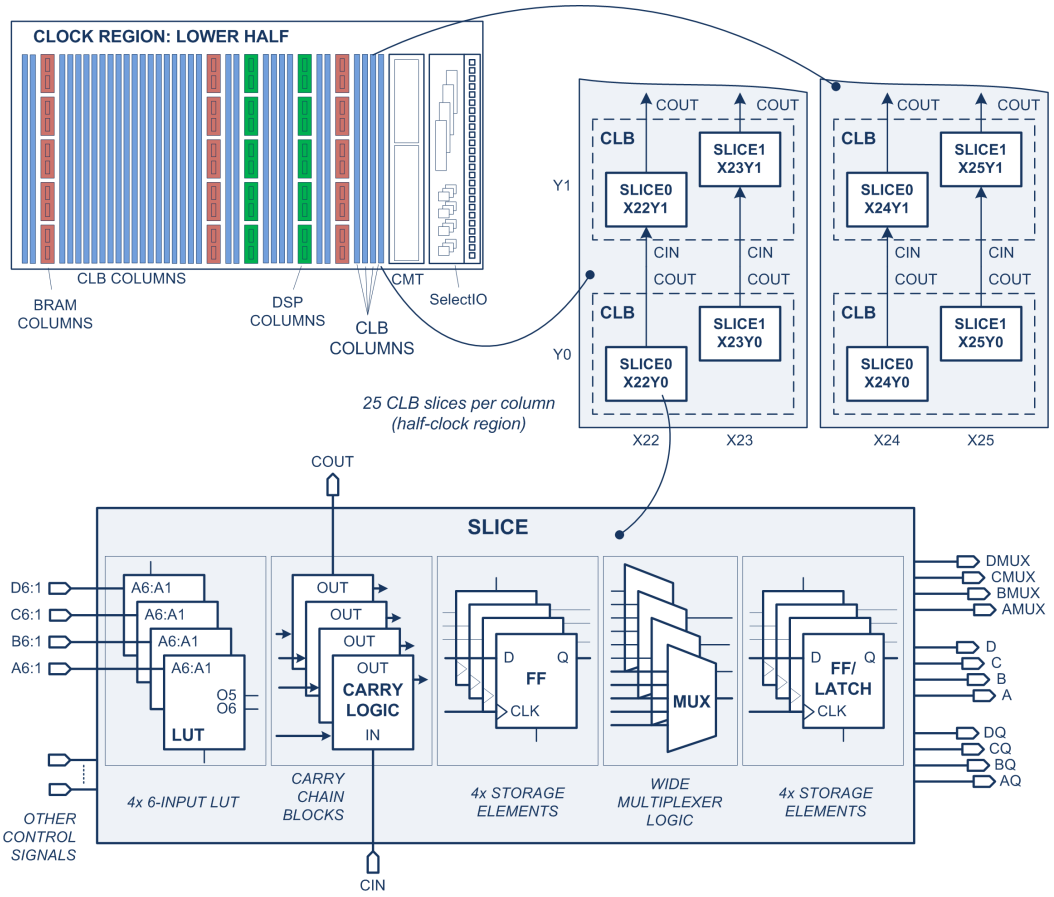

Figure 10. Simplified illustration of the 7-Series configurable logic block

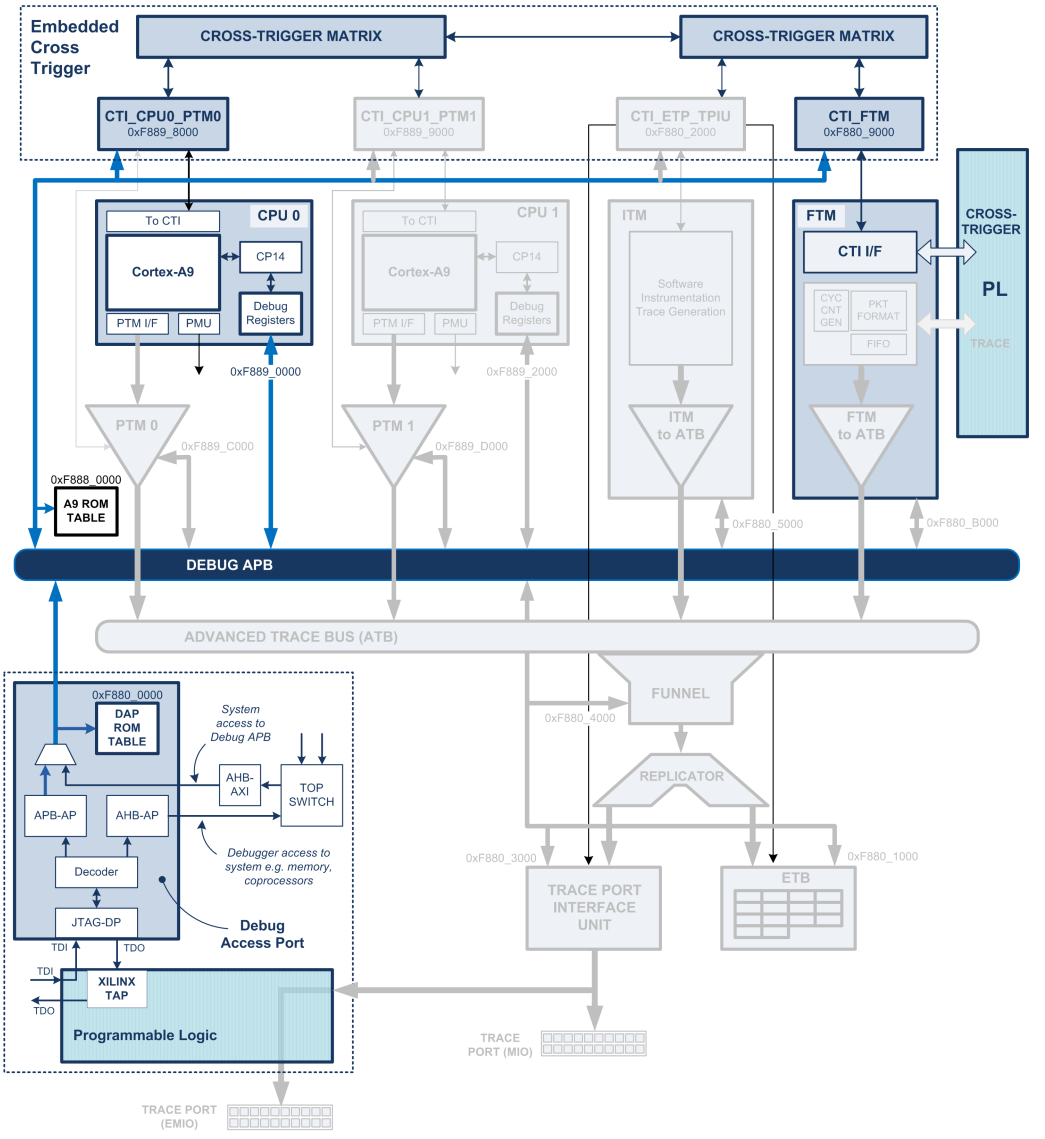

Figure 11. Halting-debug system with cross-triggering functionality added.

4. Zybo-Z7-20 Platform

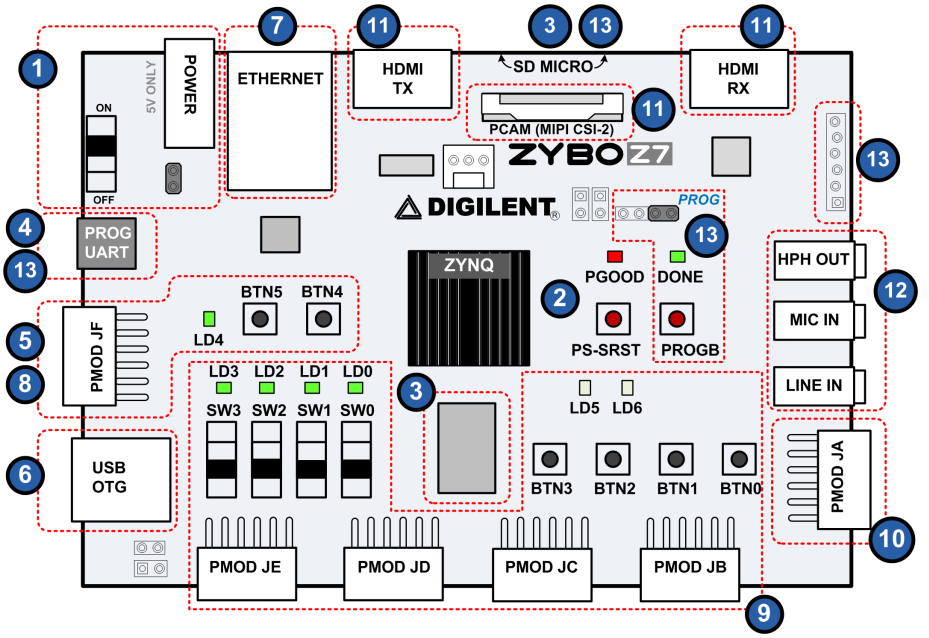

Figure 12. Zybo Z7-20 platform

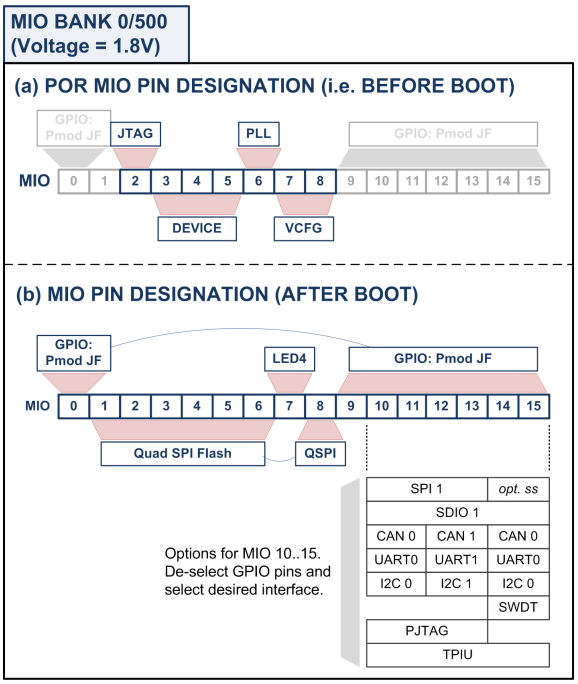

Figure 13. MIO Bank 0 pin connections for the Zybo-Z7-20 platform. (a) Pin designations during POR. (b) Pin designations in normal operation.

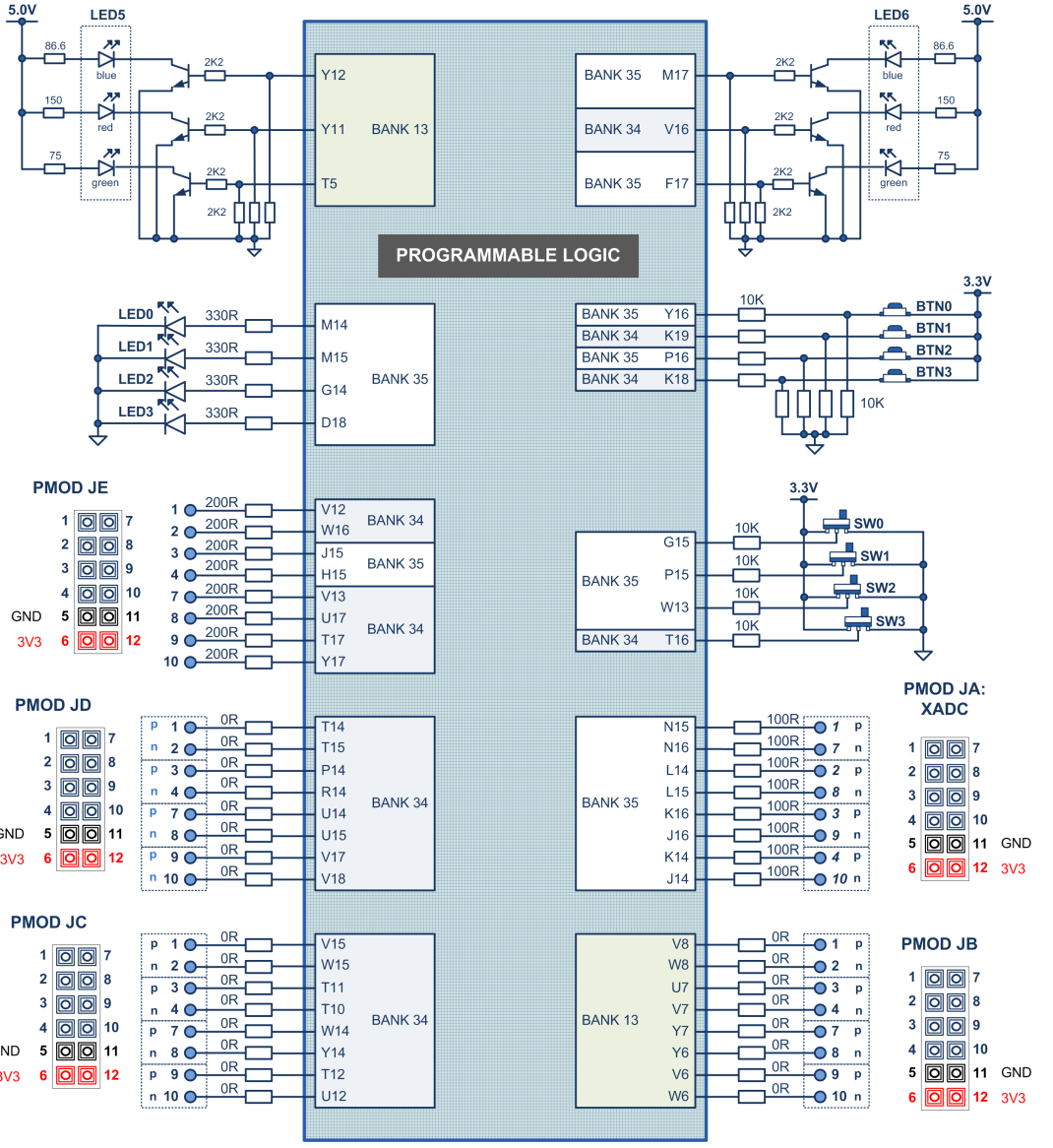

Figure 14. PL GPIO connections on the Zybo-Z7-20 platform

5. Bare-Metal Development Flow

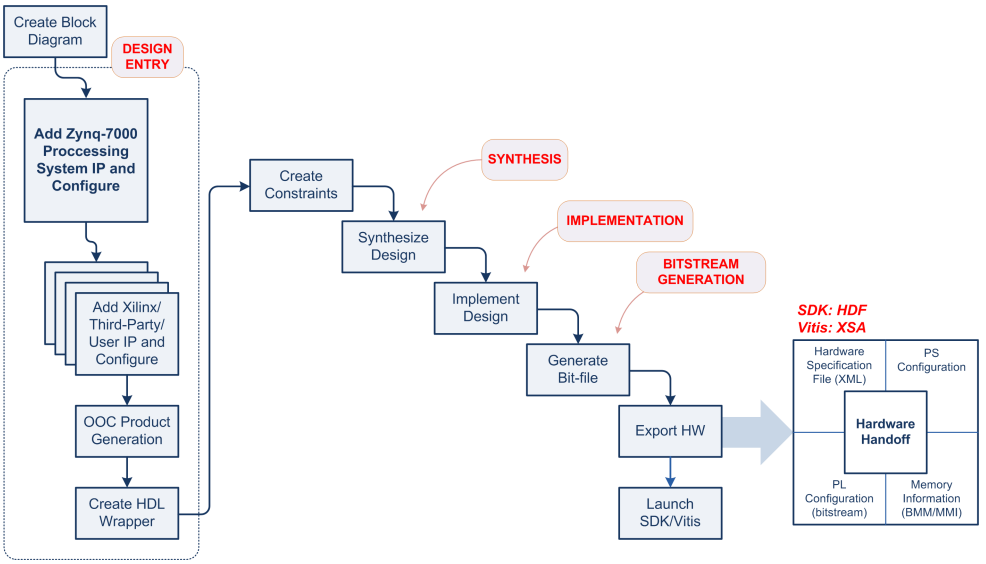

Figure 15. IP Integrator design flow used for embedded systems. This is the design flow used in this text for the Zynq-7000.

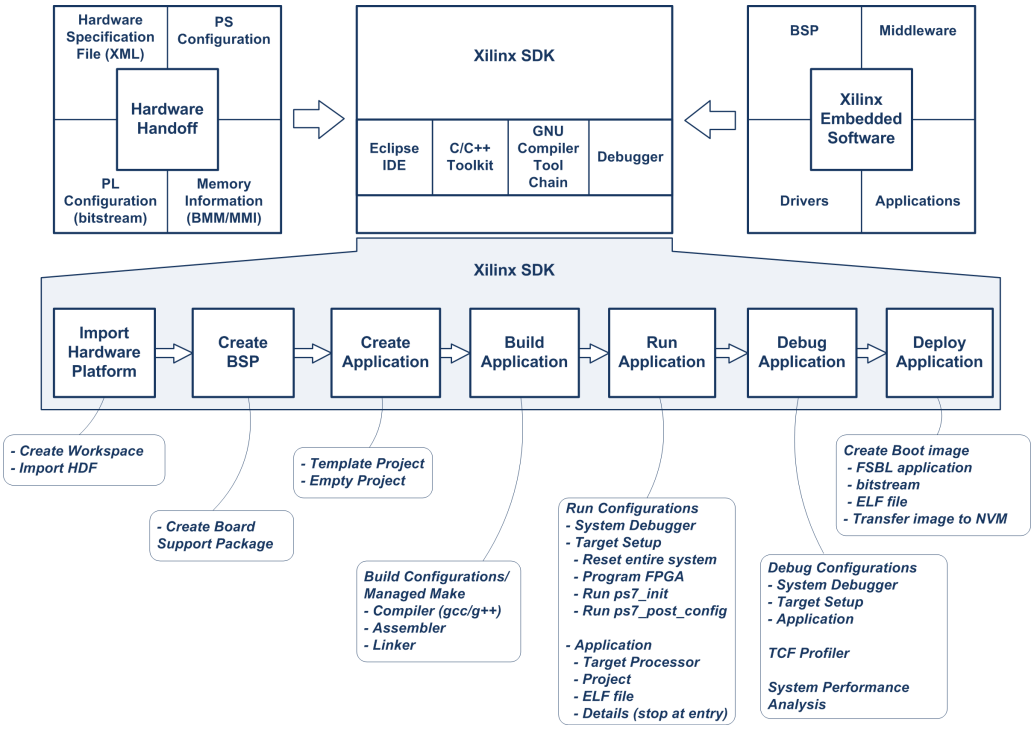

Figure 16. The Zynq-7000 software development flow in SDK.

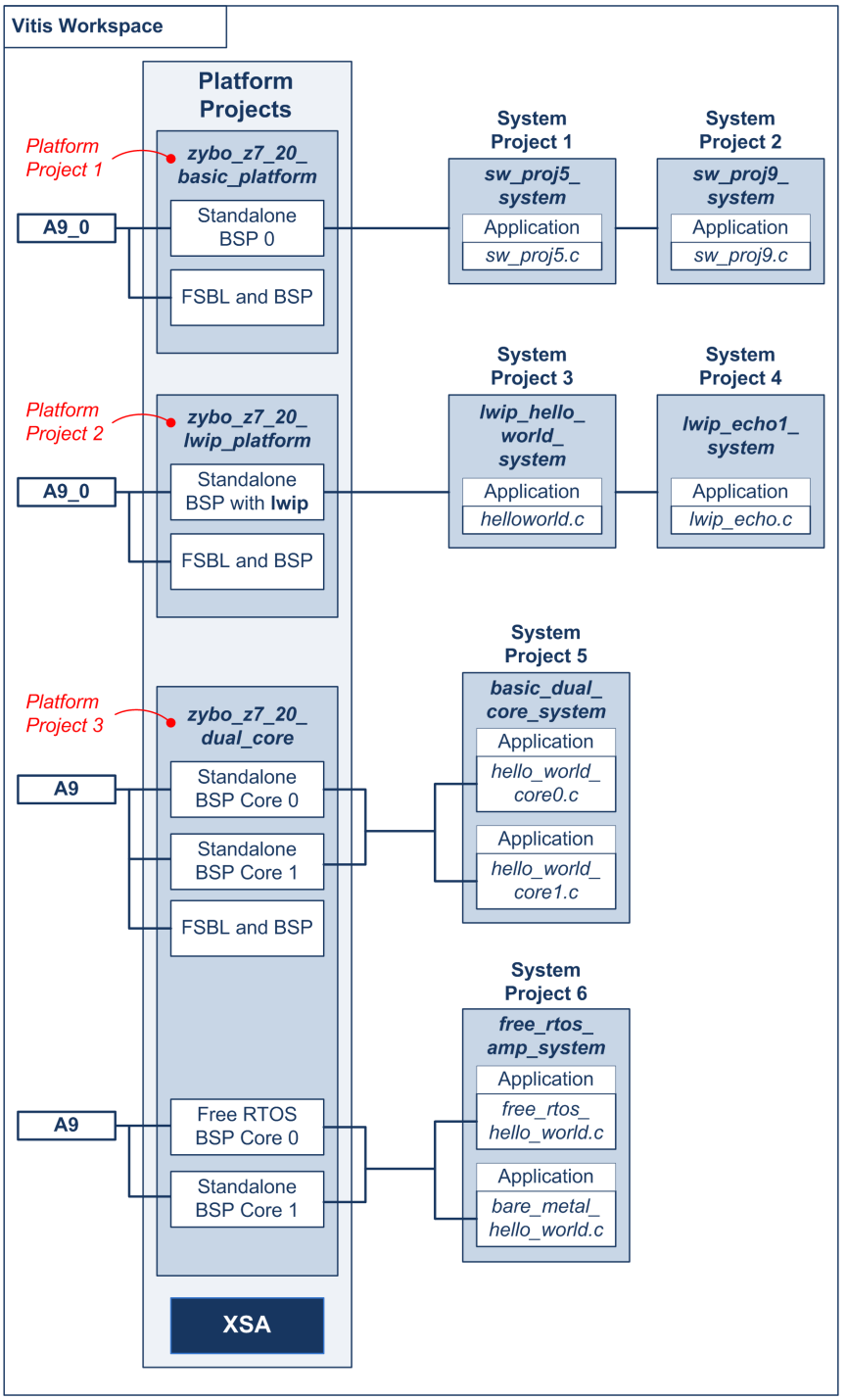

Figure 17. Vitis Workspace with three Platform Projects and six System Projects.

6. Design Entry Using Vivado

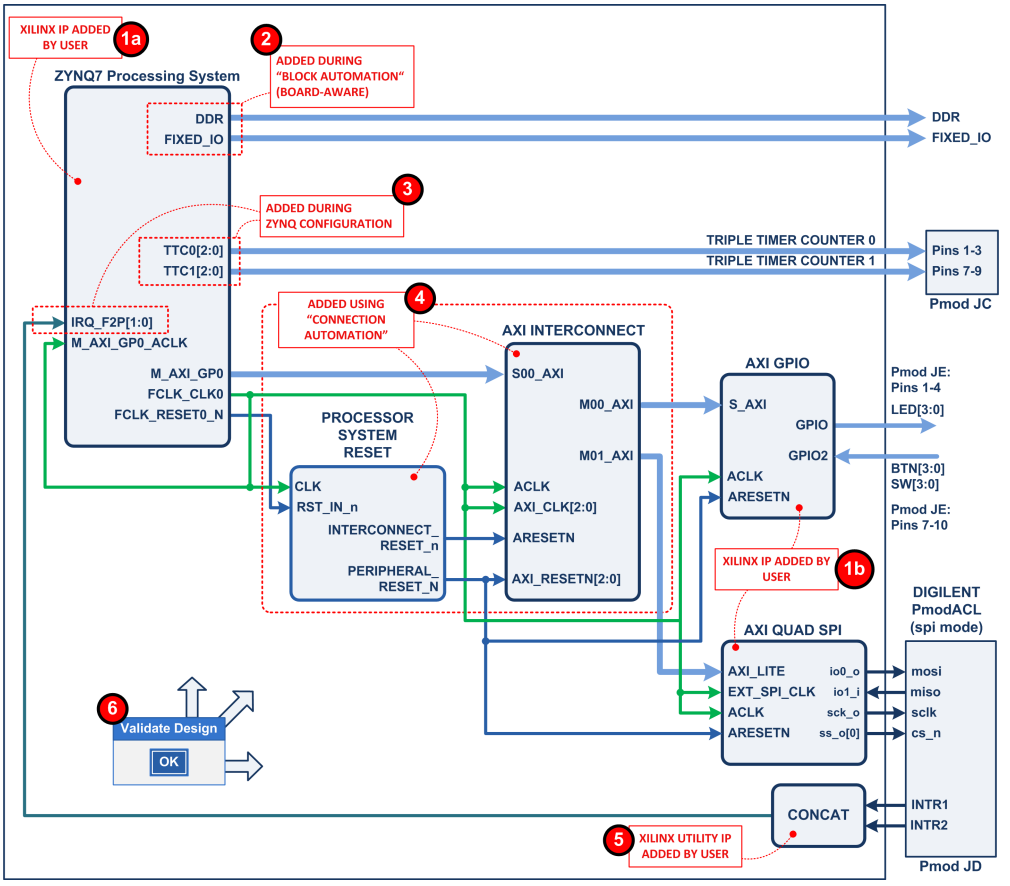

Figure 18. IP Integrator block diagram for the programmable logic circuit

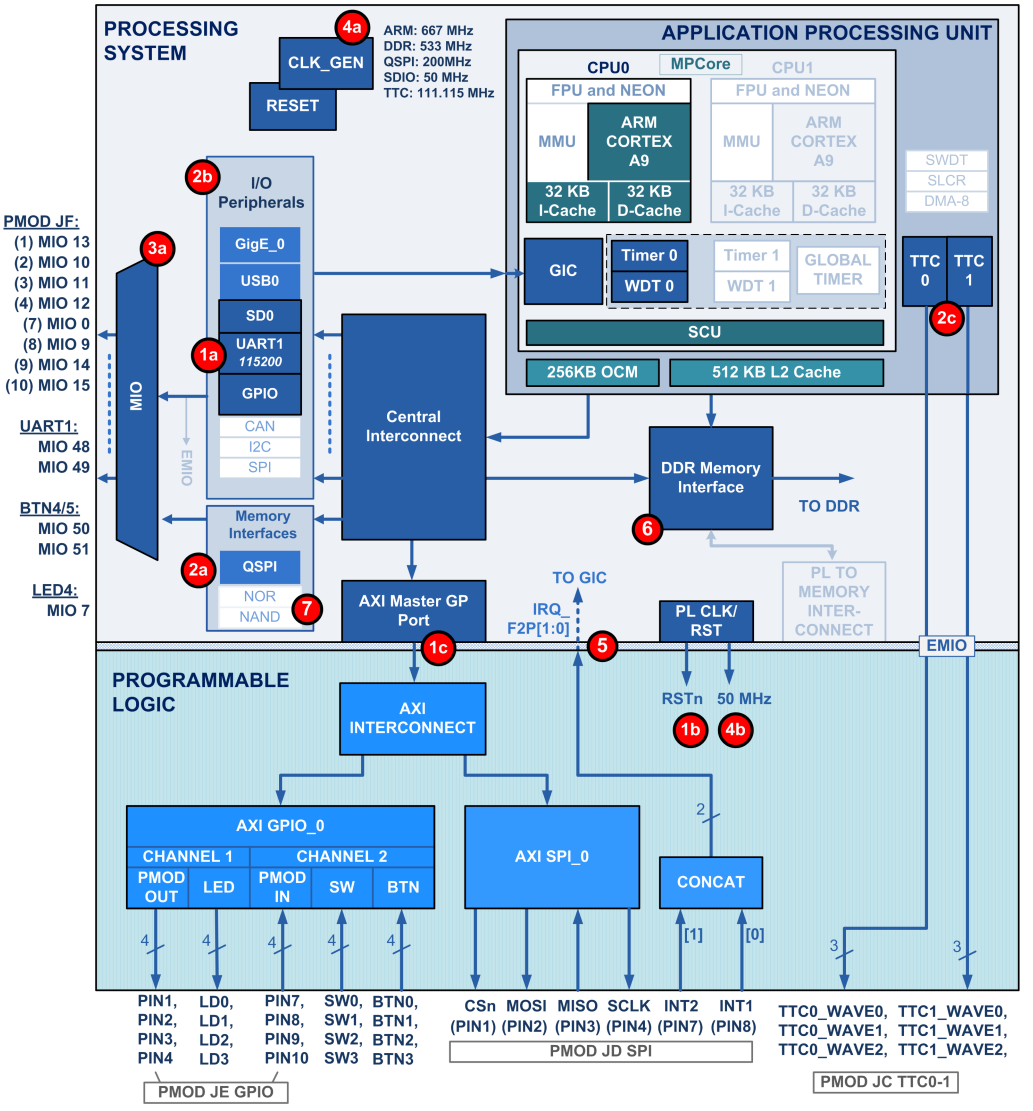

Figure 19. A guide to the Zynq-7000 processor configuration for the project in this text.

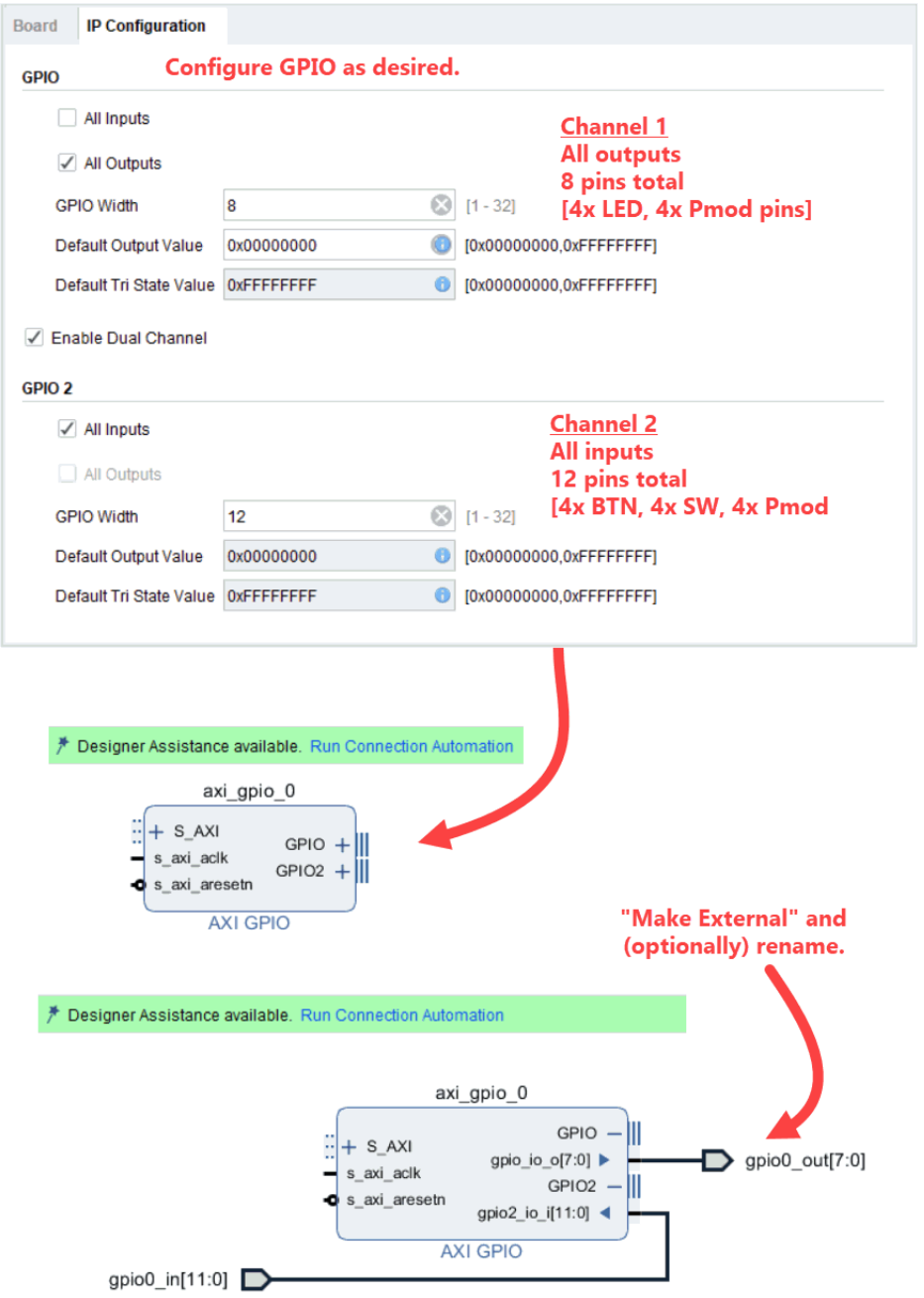

Figure 20. Set GPIO options as required in the project, and add top-level pins.

7. Software Development using SDK

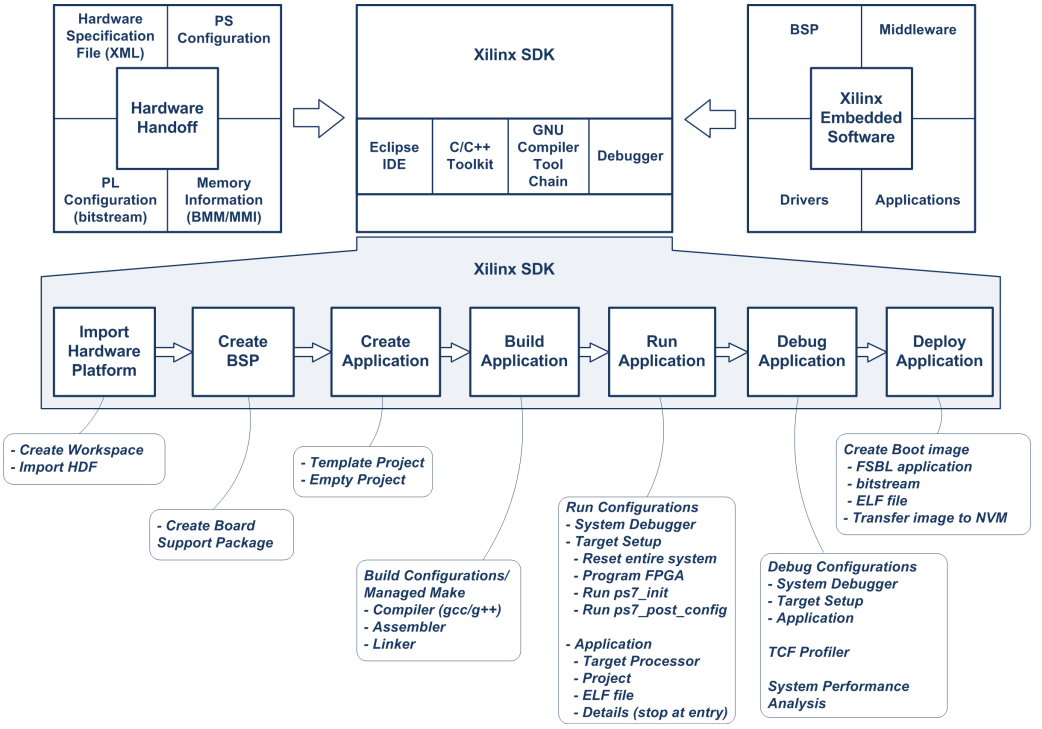

Figure 21. The Zynq-7000 software development flow in SDK.

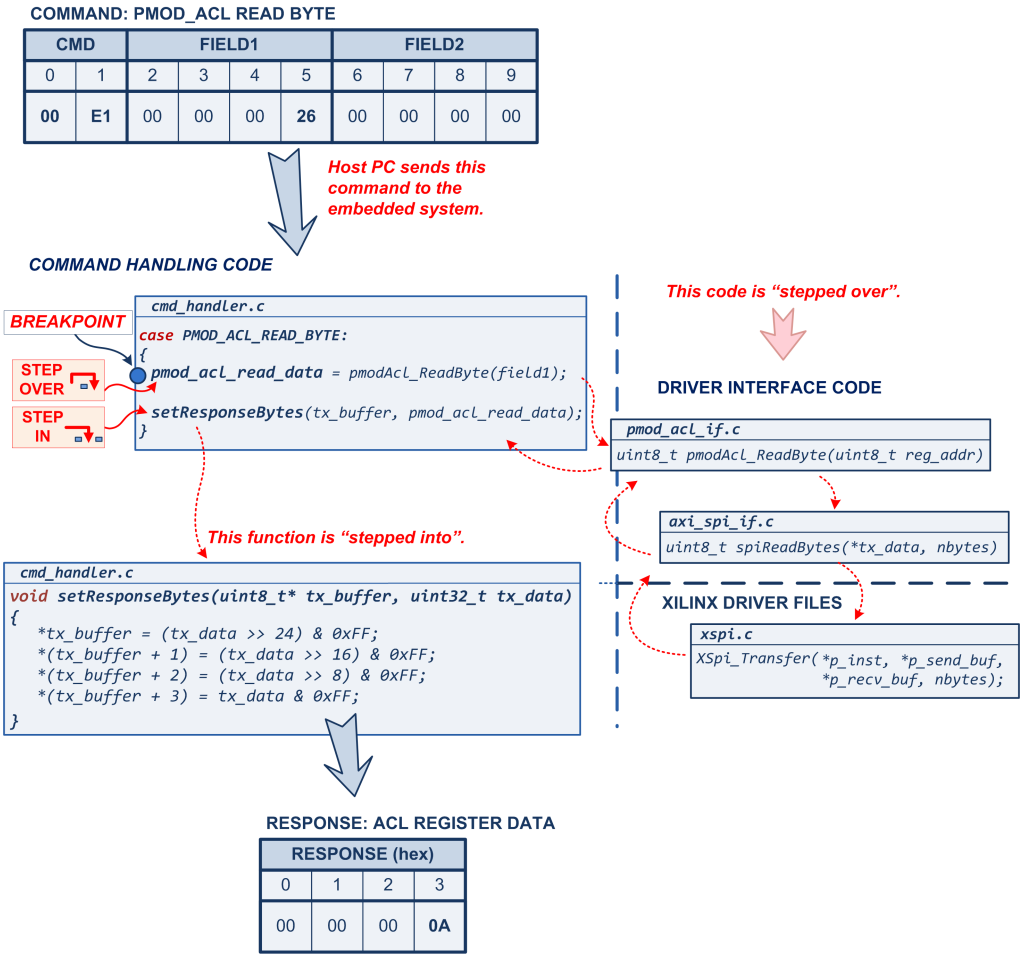

Figure 22. To demonstrate halting-debug mode, the UART command handler that is developed in Chapter 16 is used. A breakpoint is set in the command handling code, and the code is stepped through and analysed.

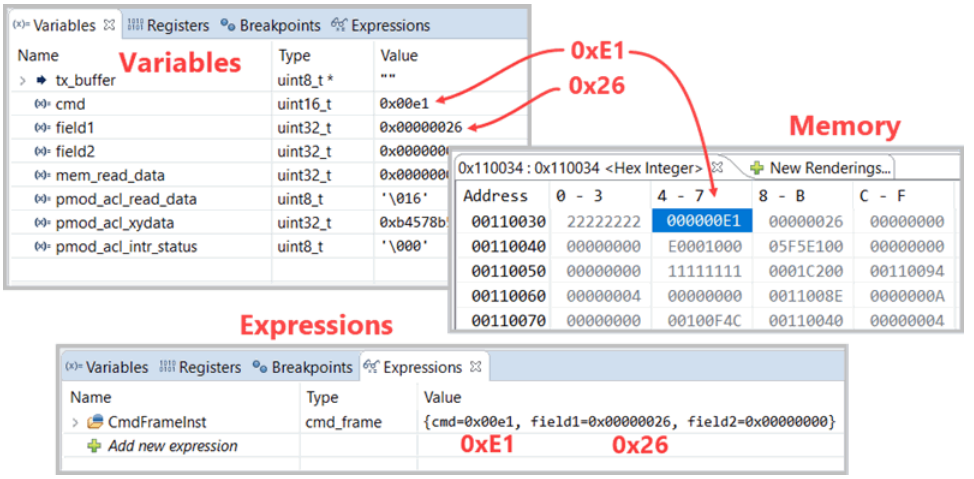

Figure 23. Use the general debug features in SDK to view variables, memory, and expressions (for example).

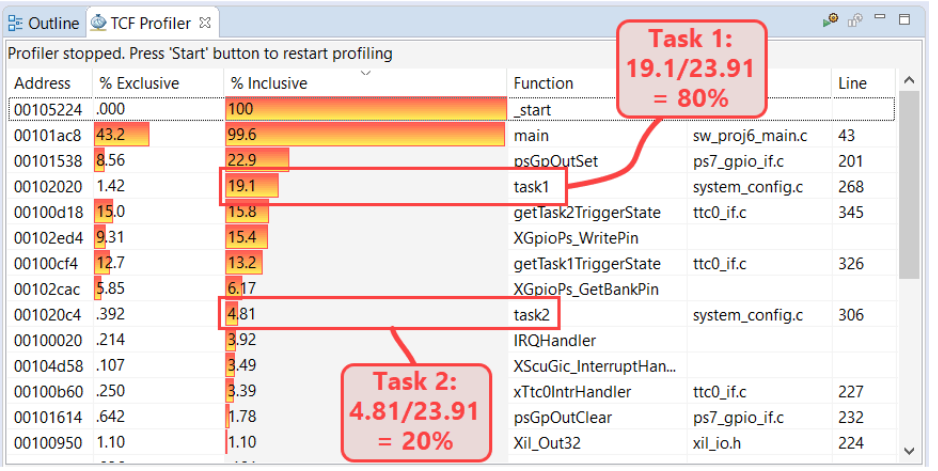

Figure 24. Task 1 and Task 2 each contain a for-loop, with Task 1 set to run four times longer than Task 2. TCF Profiler verifies that this is the case.

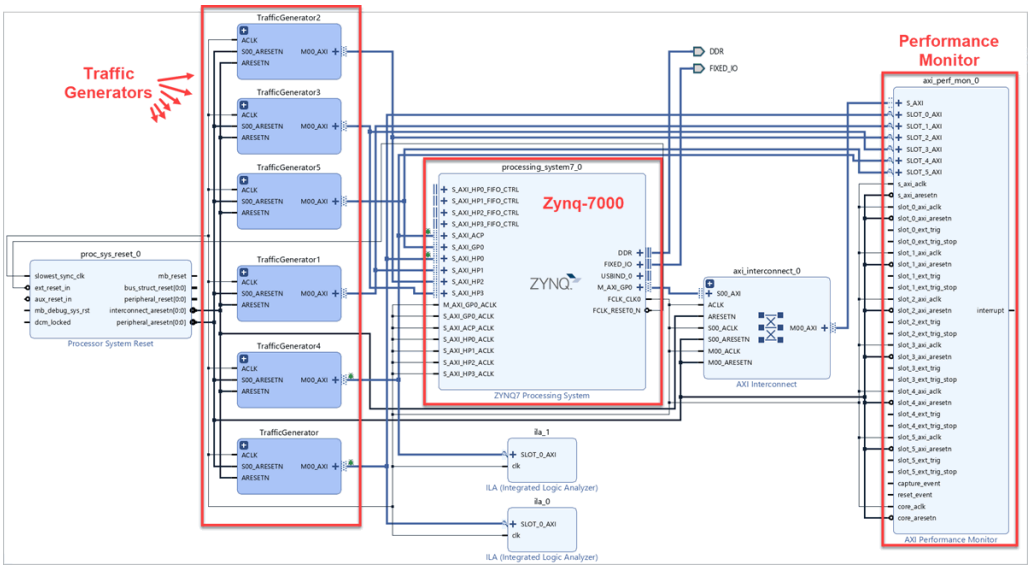

Figure 25. Traffic generators and a performance monitor are added to the Zynq-7000 block diagram for PL performance analysis.

Figure 26. A subset of programmable logic performance metrics displayed in the Performance Analysis perspective.

8. Zynq-7000 GPIO

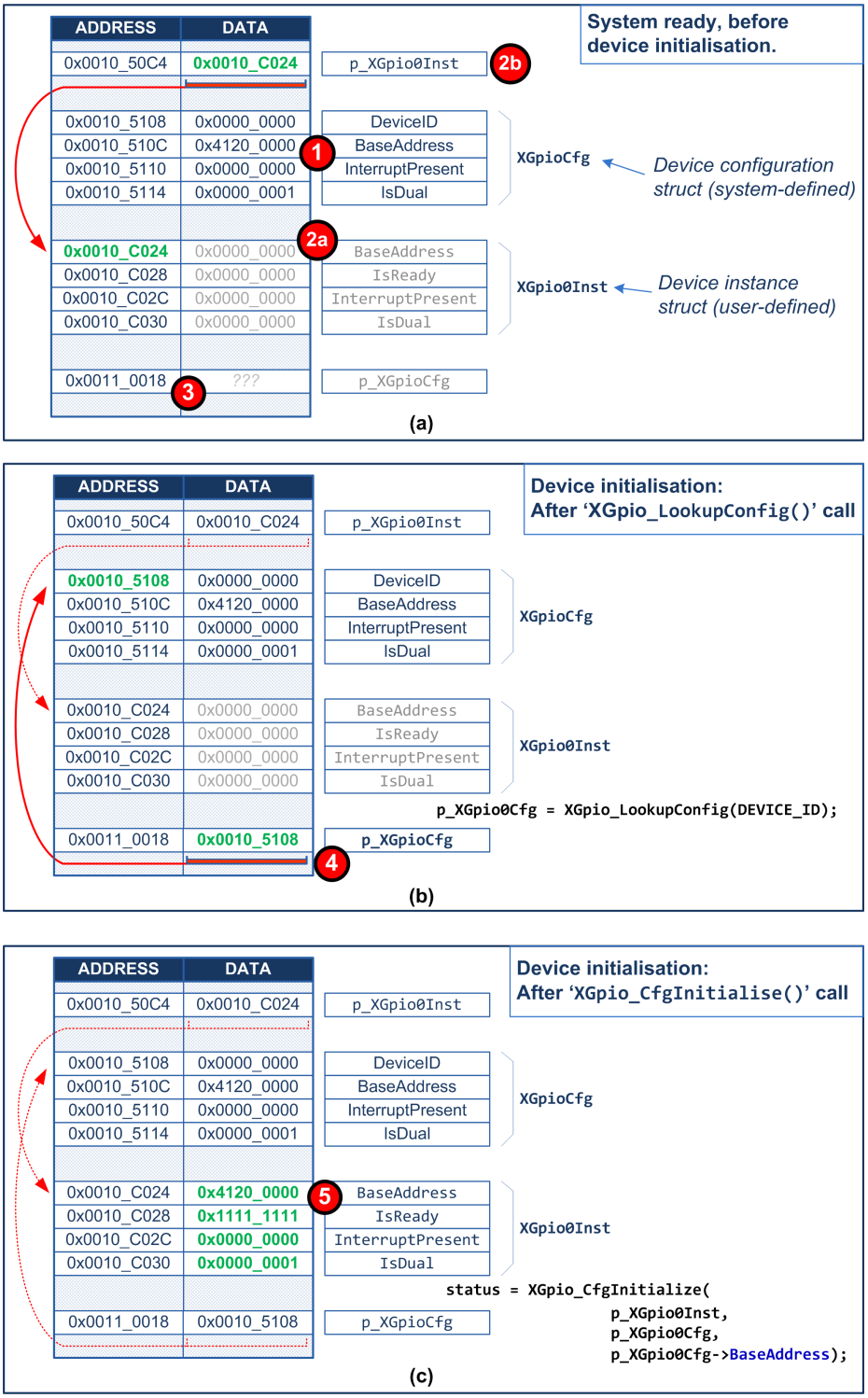

Figure 27. Memory state during driver initialisation. (a) After system initialisation, before device initialisation. (b) State after XGpio_LookupConfig(). (c) State after XGpio_CfgInitialise().

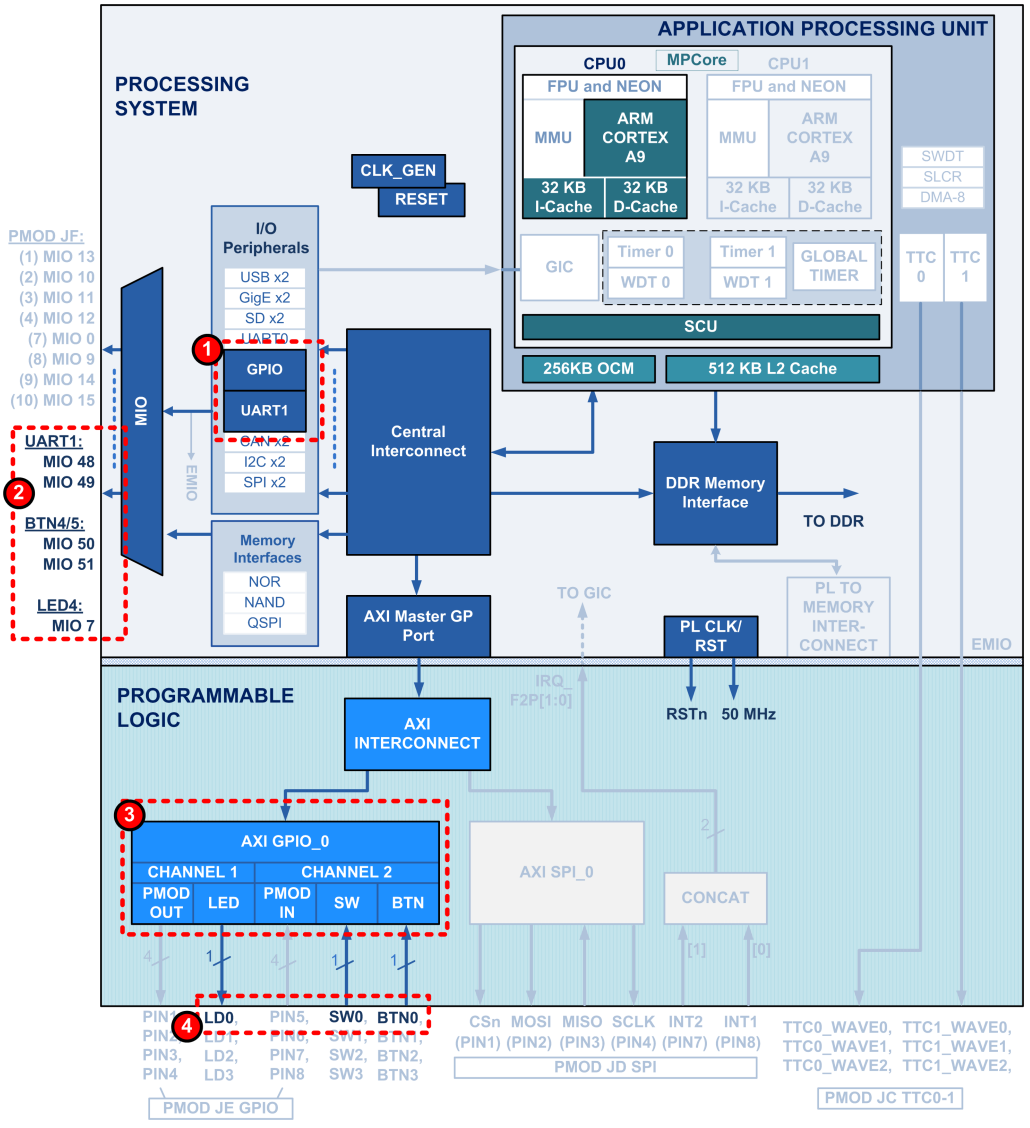

Figure 28. Simplified Zynq-7000 block diagram for software project 2.

9. Structured Progamming in C

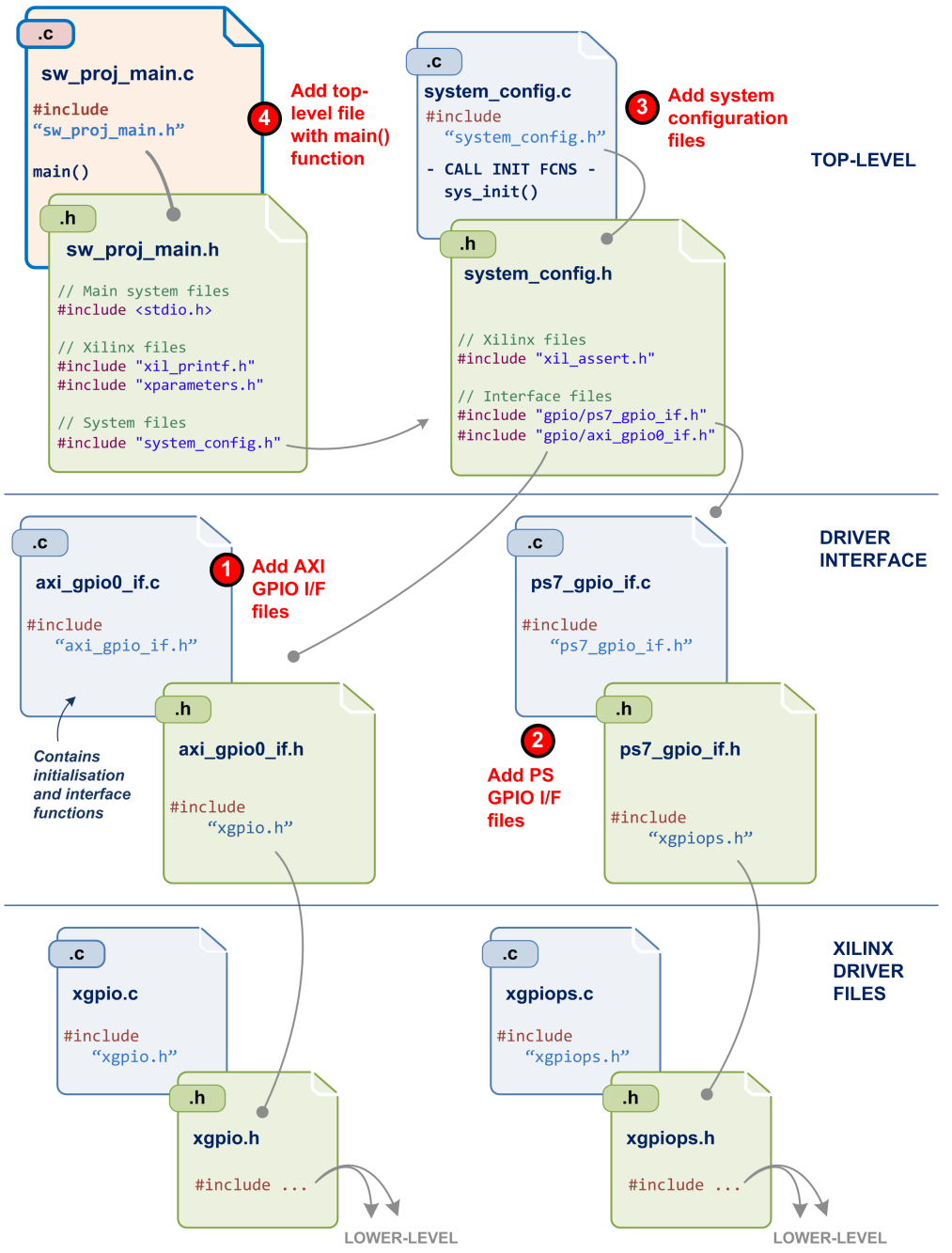

Figure 29. Flat-file program converted to a hierarchical structure

10. Zynq-7000 Timers and Watchdogs

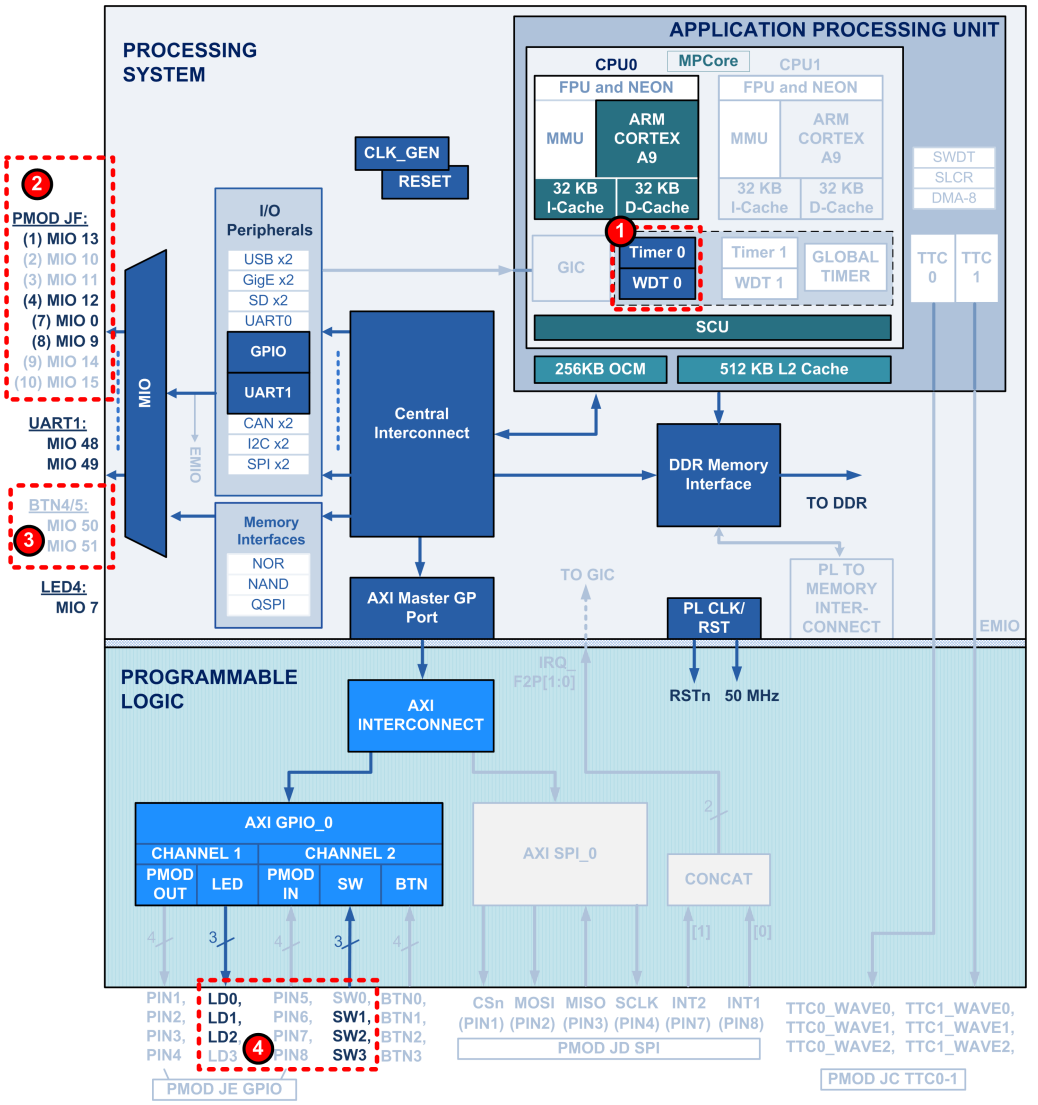

Figure 30. Simplified Zynq-7000 block diagram for software project 4

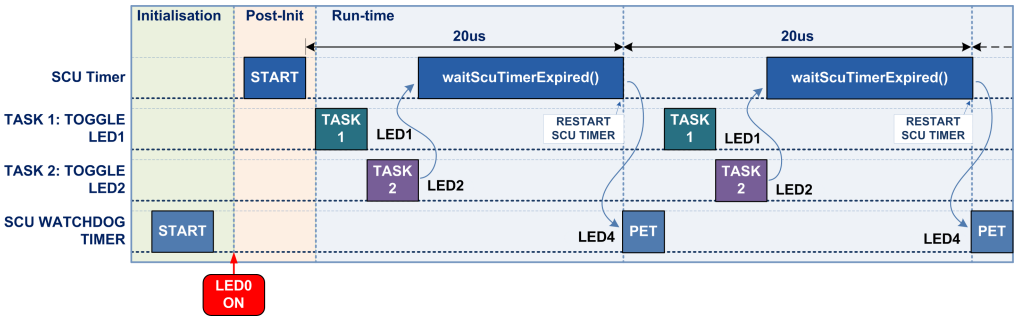

Figure 31. Program timing for software project 4, including initialisation

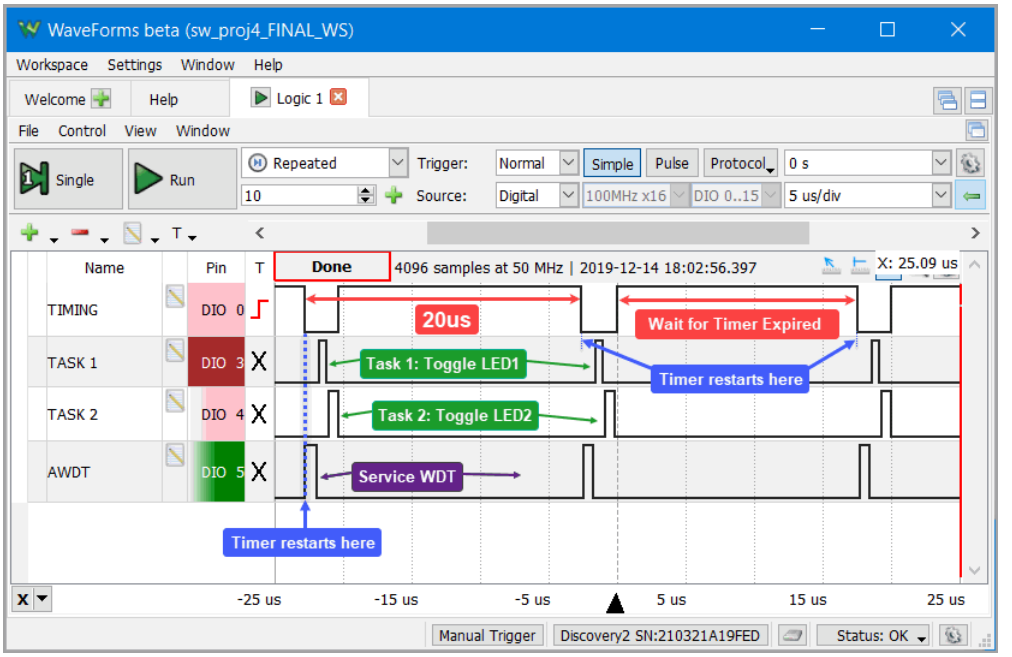

Figure 32. Test signals for software project 4.

11. An Introduction to Zynq-7000 Interrupts

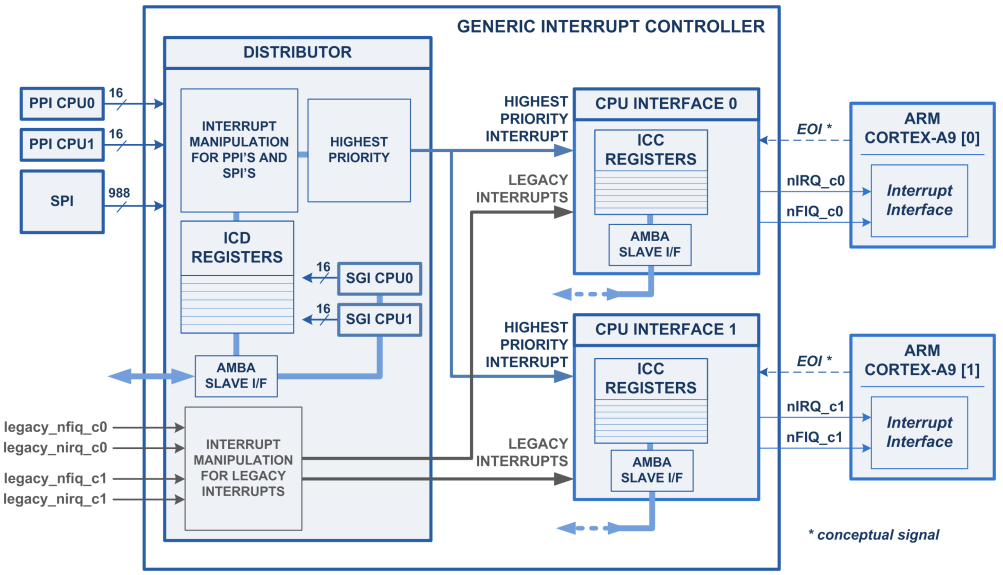

Figure 33. Simplified block diagram of the Generic Interrupt Controller

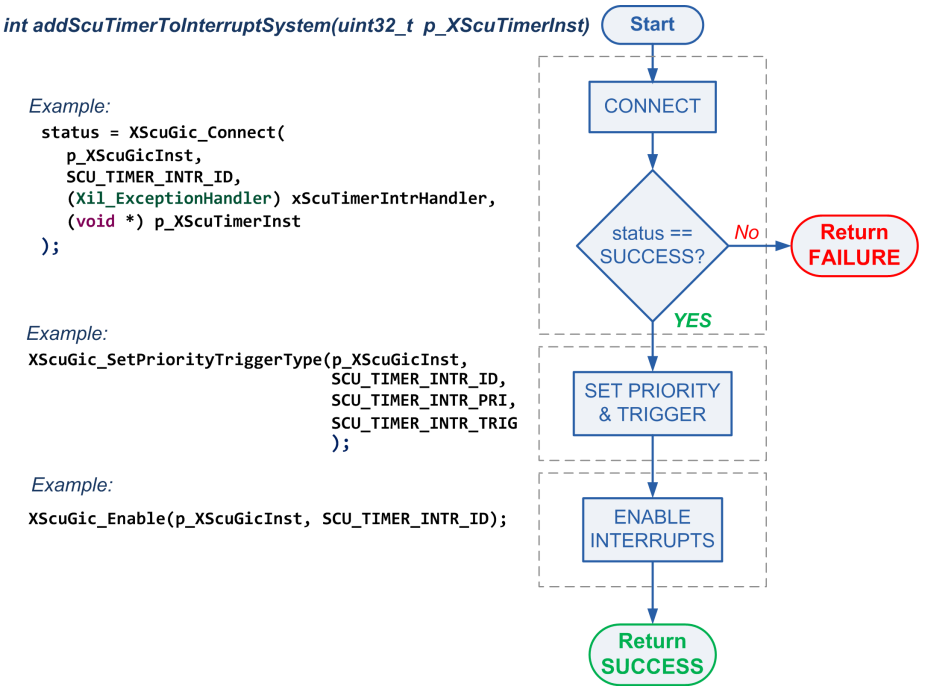

Figure 34. Program flow for adding SCU Timer to interrupt system

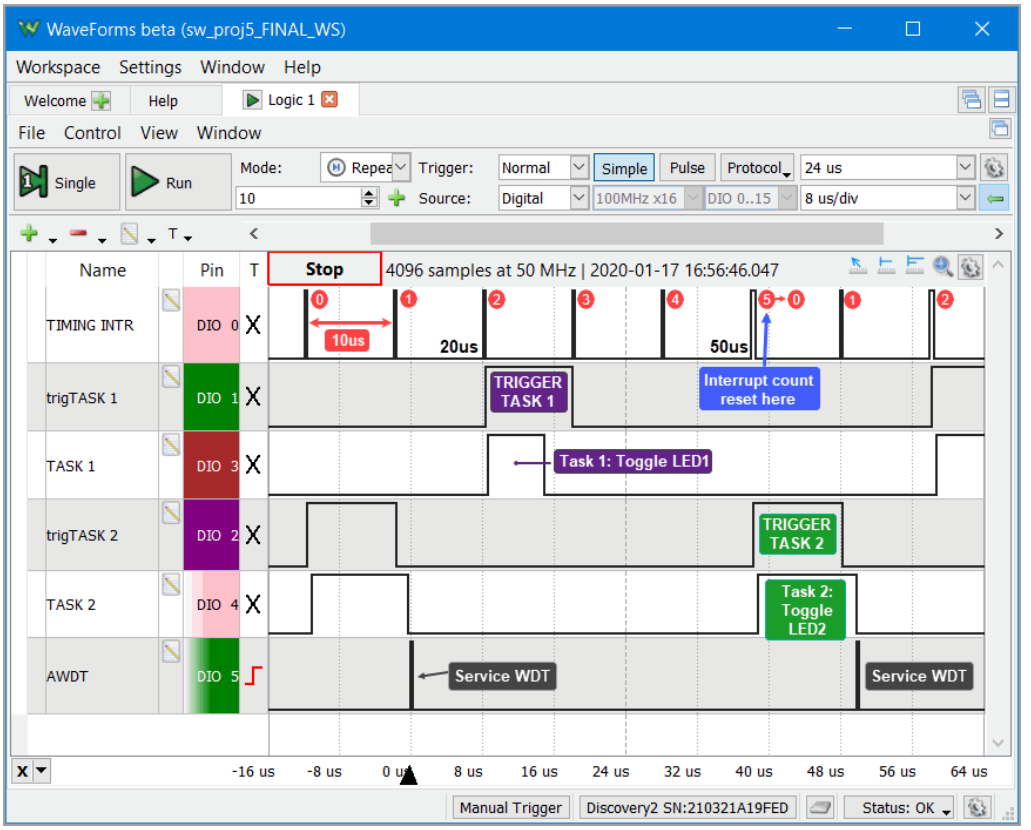

Figure 35. Logic analyser waveforms for software project 5.

12. The Triple Timer Counter

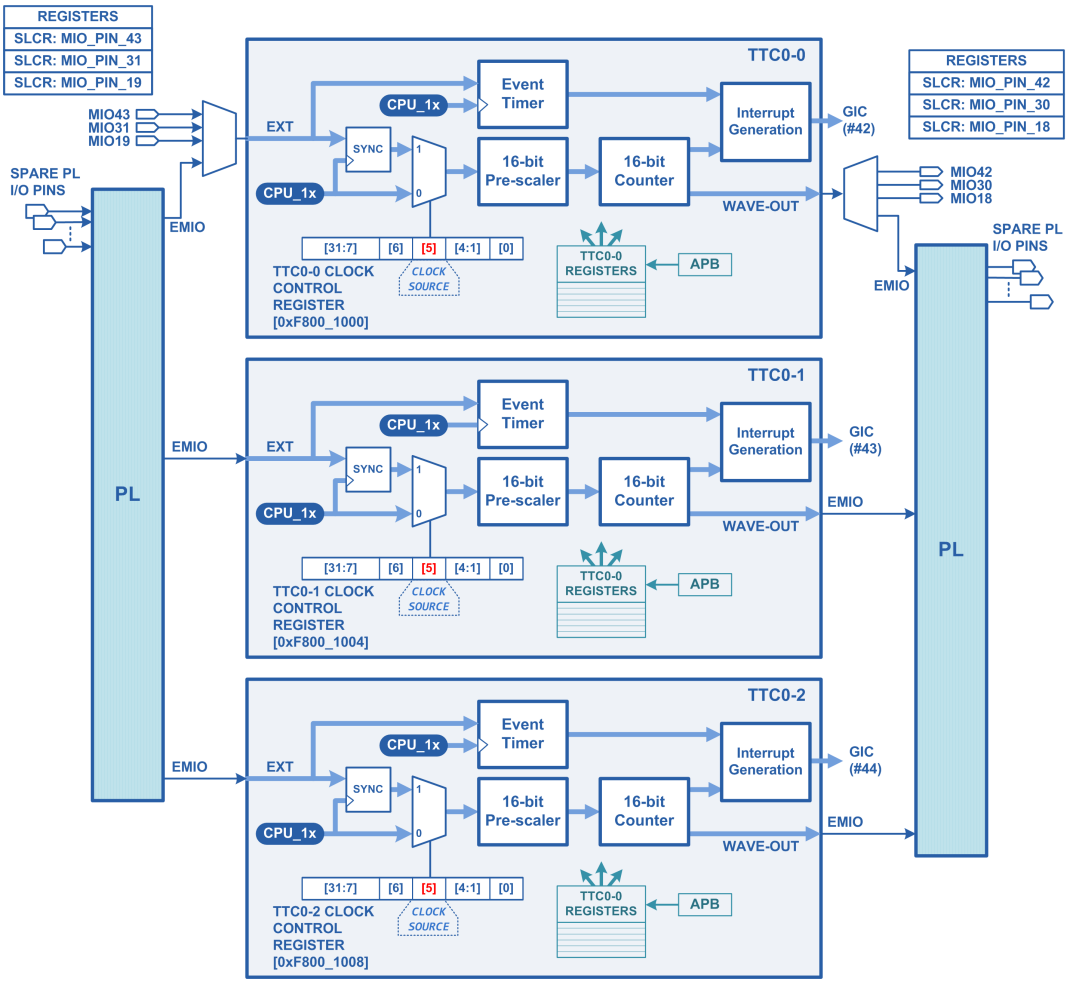

Figure 36. Conceptual block diagram for triple-timer counter 0 in the Zynq-7000. An identical second block also exists for TTC1.

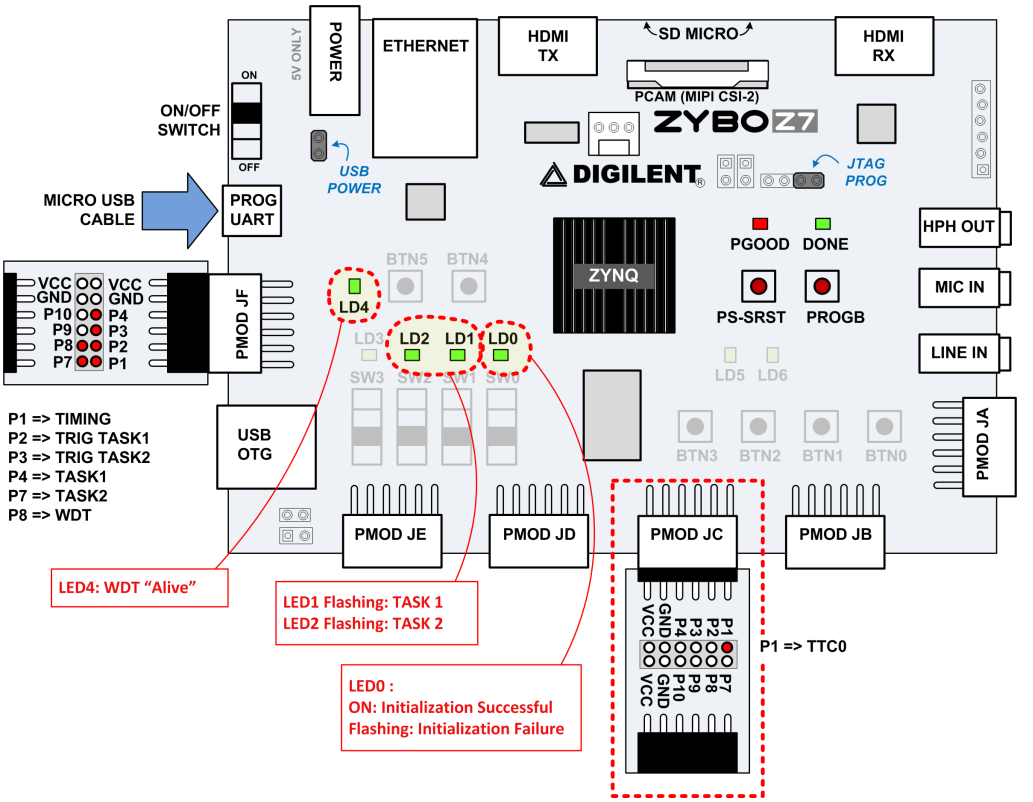

Figure 37. Board details for software project 6.

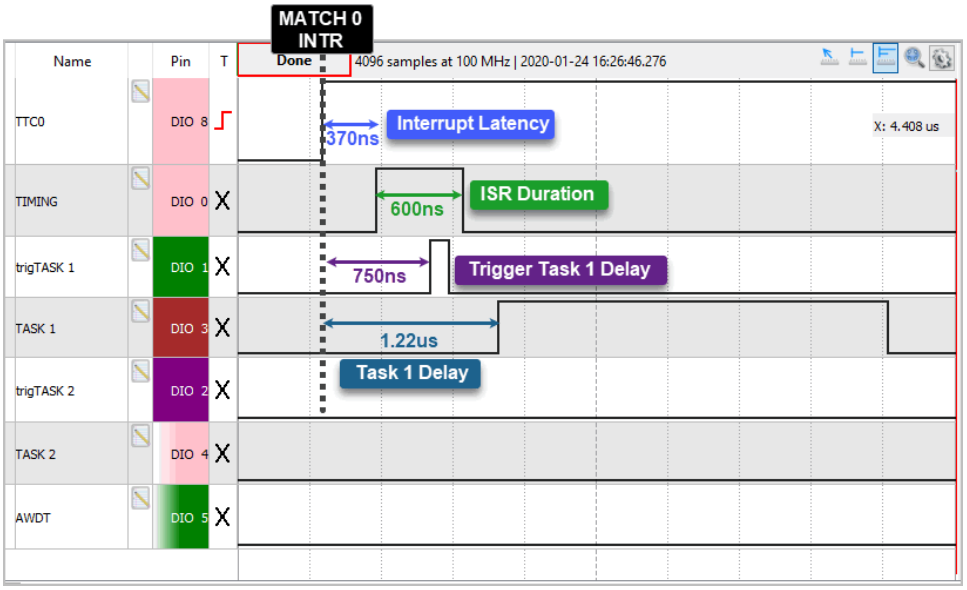

Figure 38. Timing information for the first interrupt (Match 0).

13. Button De-bouncing in Software

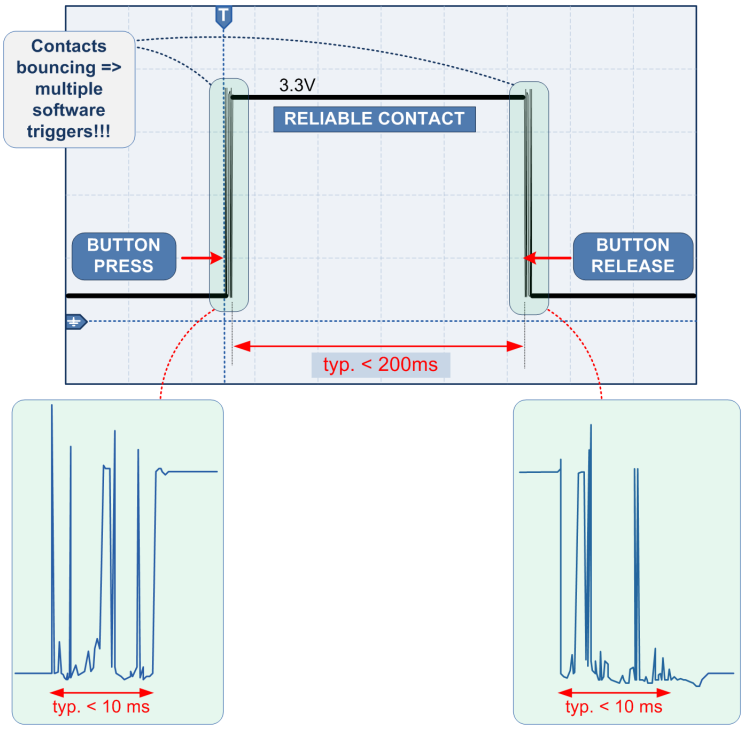

Figure 39. A simple illustration of switch bounce.

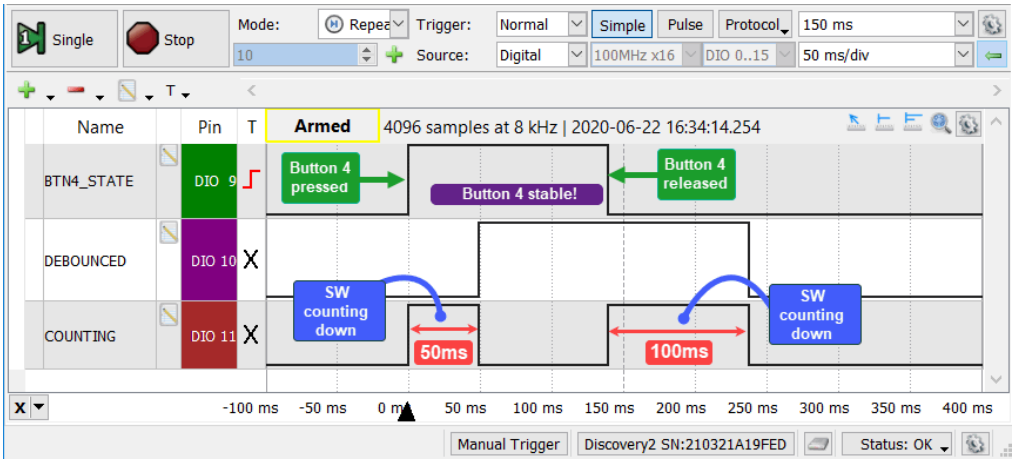

Figure 40. Viewing the button debounce signals.

14. UART Command Handler Design

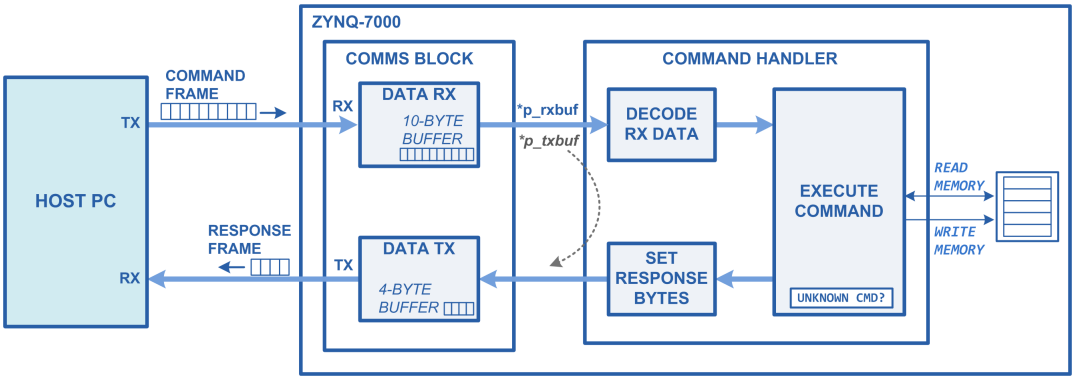

Figure 41. Block diagram showing main features of the command handling system

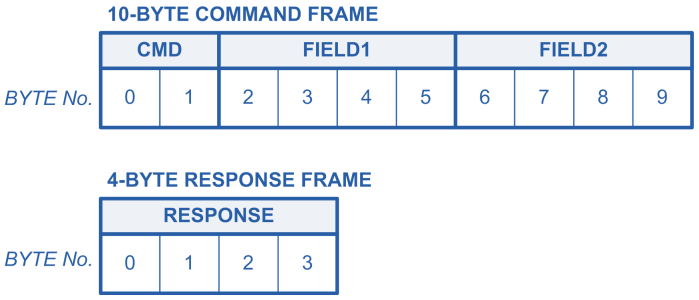

Figure 42. 10-byte Command Frame and 4-byte Response Frame

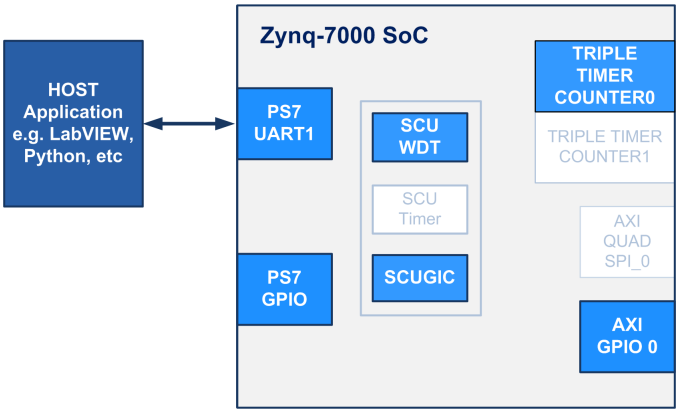

Figure 43. Project software block diagram for software project 8

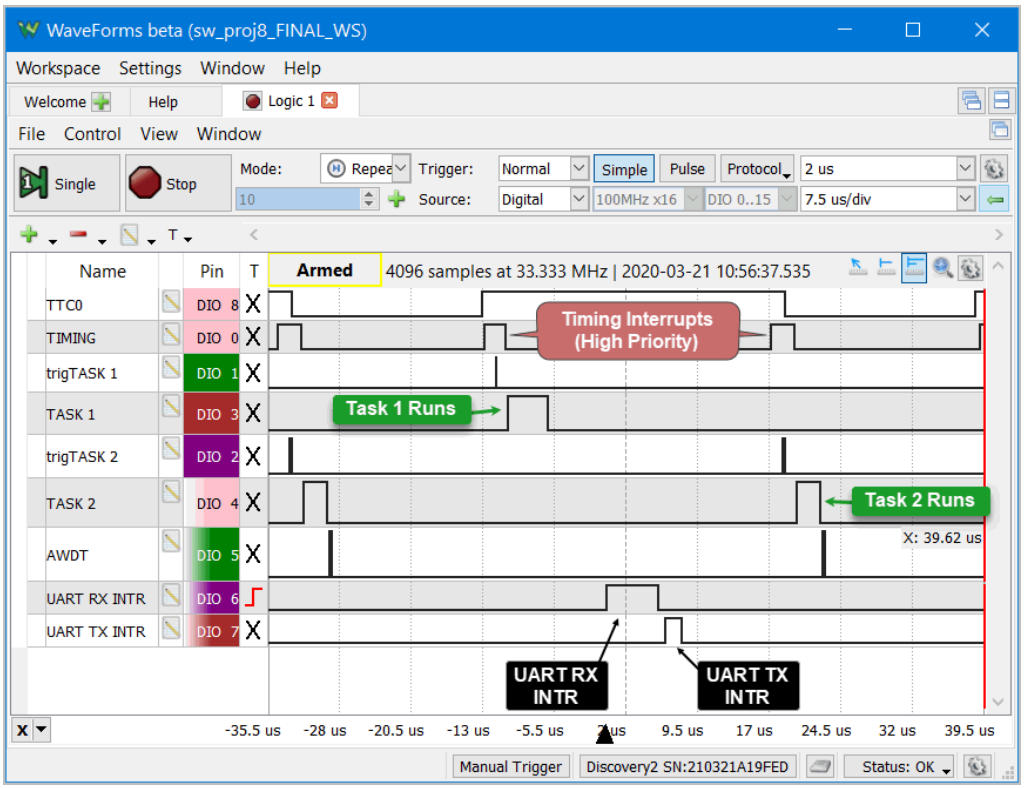

Figure 44. Logic analyser capture showing the basic relationship between timing interrupts and UART1 interrupts.

15. Host Application Design

Python:

```
def execute_mem_write(addr, data):
  cmd = 0x00D3
  field1 = addr
  field2 = data
  response = execute_cmd(cmd, field1, field2)
  return response
```

```
def execute_mem_read(addr):
  cmd = 0x00D4
  field1 = addr
  field2 = 0x00000000
  read_data = execute_cmd(cmd, field1, field2)
  return read_data
```

```
ser = open_serial_port('COM6', 115200, 2000) // Re-open if necessary!!!
response = execute_mem_write(0x04000004, 0xABCD1234)
print("Response = 0x{0:0{1}X}".format(response, 8))
```

```
Response = 0x01010101
```

```
response = execute_mem_read(0x04000004)
print("Response = 0x{0:0{1}X}".format(response, 8))
```

```
Response = 0xABCD1234
```

```
close_serial_port(ser)
```

LabVIEW:

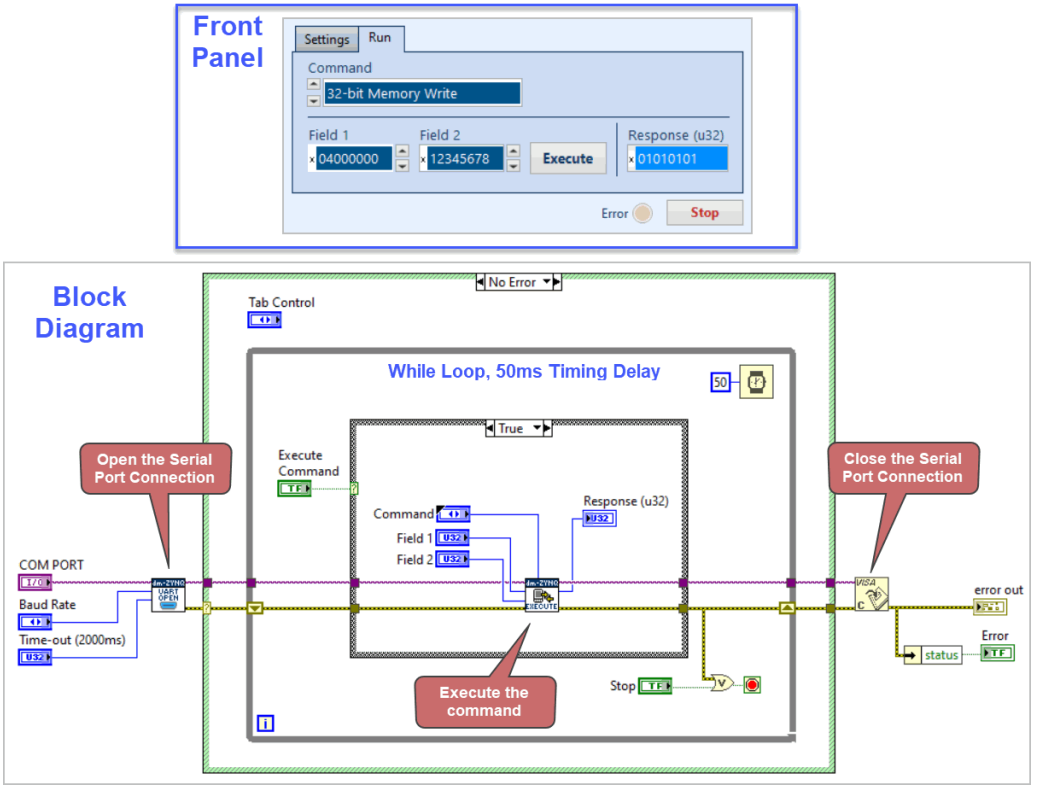

Figure 45. The main LabVIEW application.

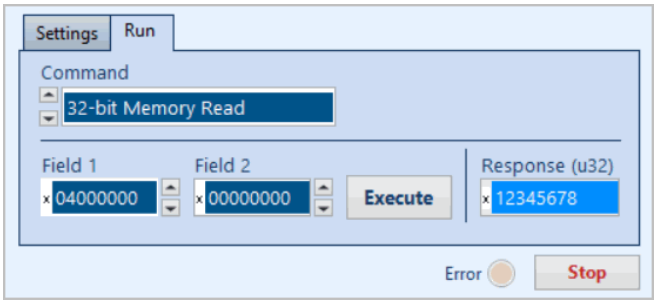

Figure 46. Test the memory read command. Field 1 is the read address (Field 2 is irrelevant). The response in this test should be 0x1234_5678.

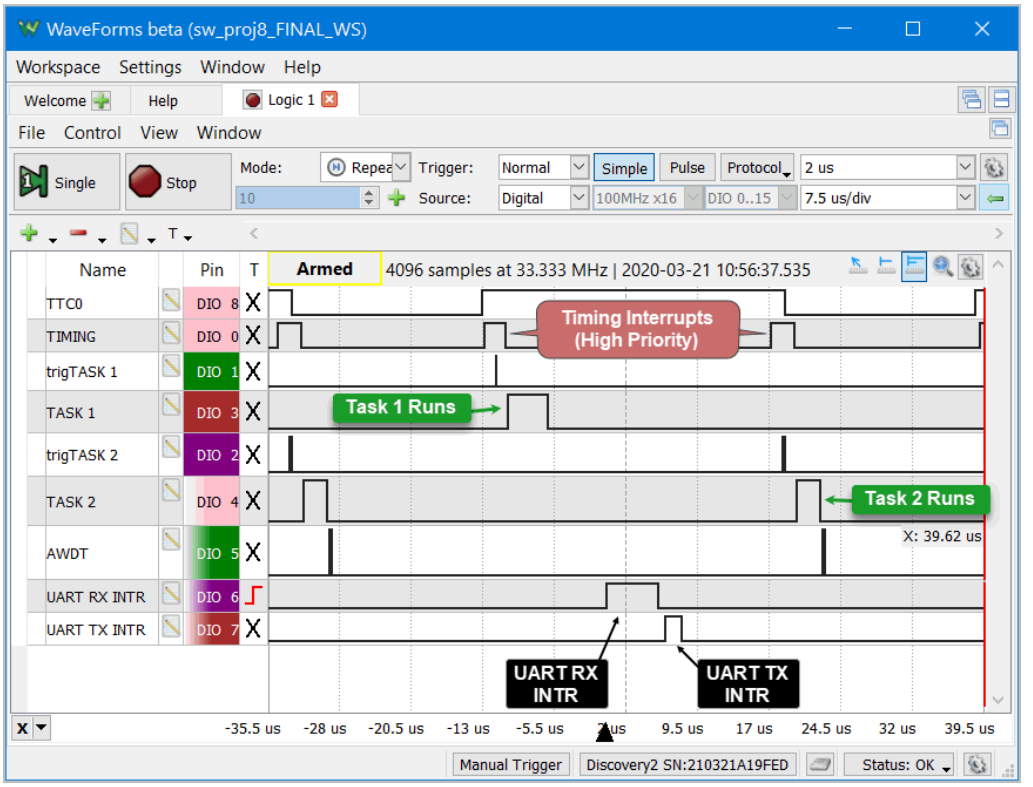

Figure 47. Logic analyser capture showing the basic relationship between timing interrupts and UART1 interrupts.

16. Programmable Logic Interrupts

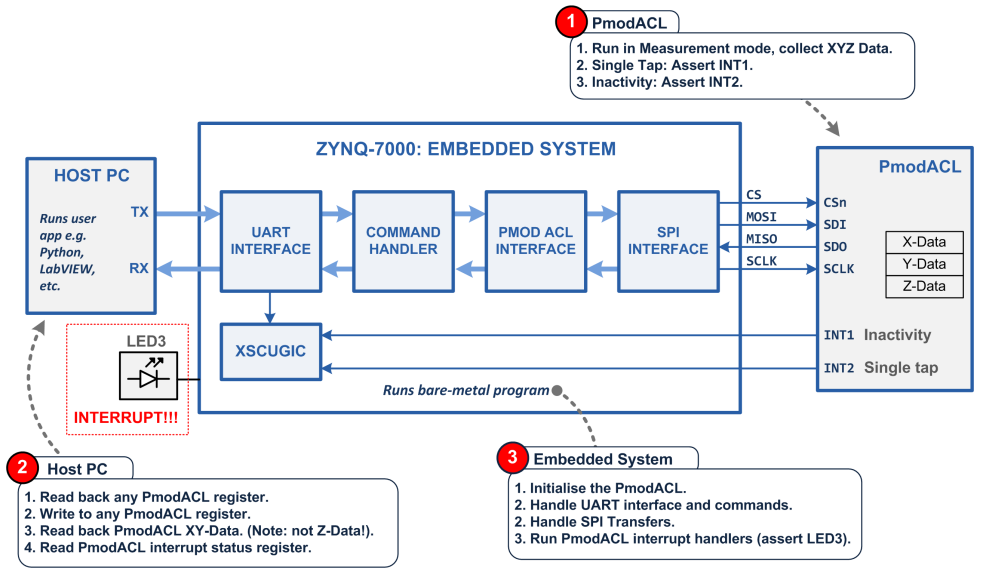

Figure 48. Proposed system which includes (1) Pmod ACL accelerometer. (2) Host PC running Python/LabVIEW; (3) Zynq-7000 embedded system.

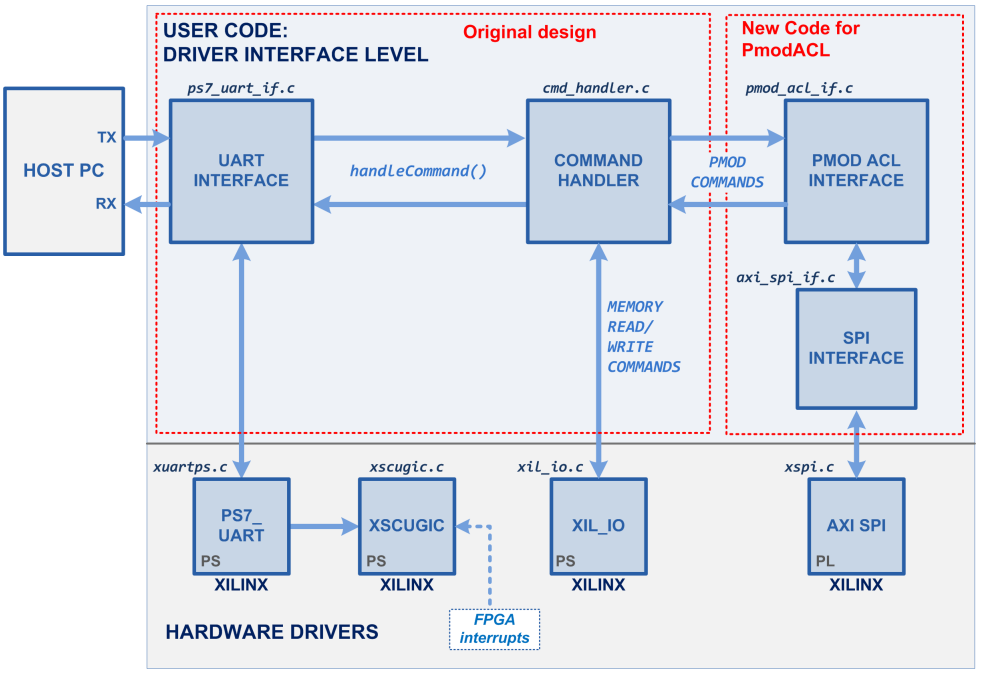

Figure 49. Software block diagram for the Pmod ACL design, showing that two new software components must be added (the Pmod ACL interface, and the AXI SPI interface).

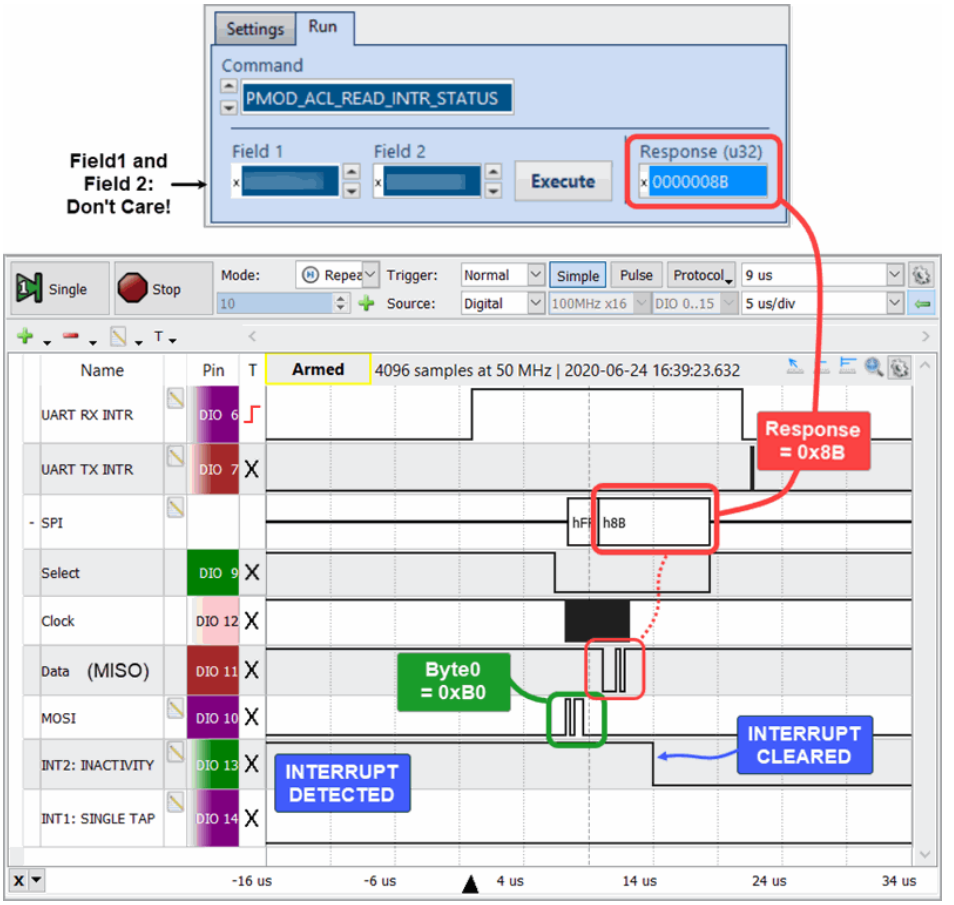

Figure 50. Testing the Pmod ACL Read Interrupt Status command, after an Inactivity Interrupt event has occurred. The response in this case is 0x8B. The interrupt signal is cleared when the status is read.

17. Platform Boot Options

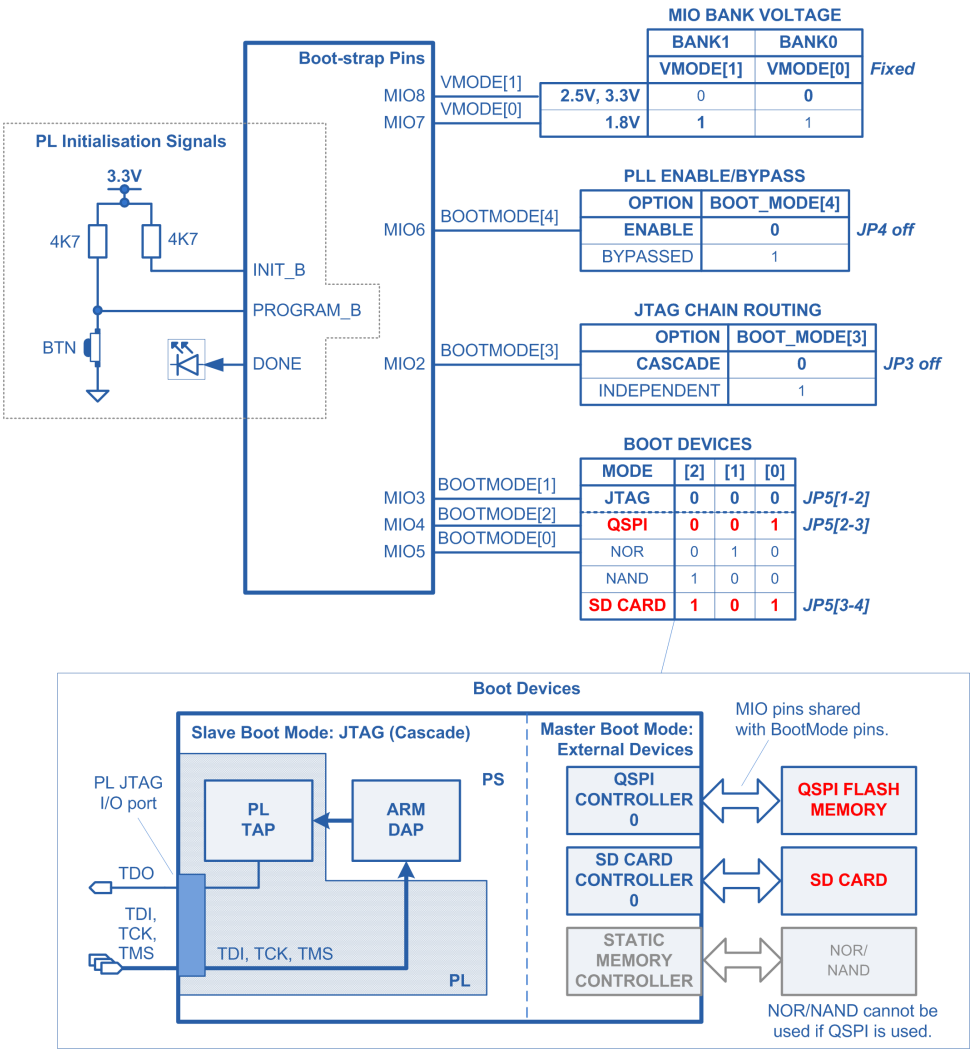

Figure 51. Zybo-Z7-20 External Boot Options

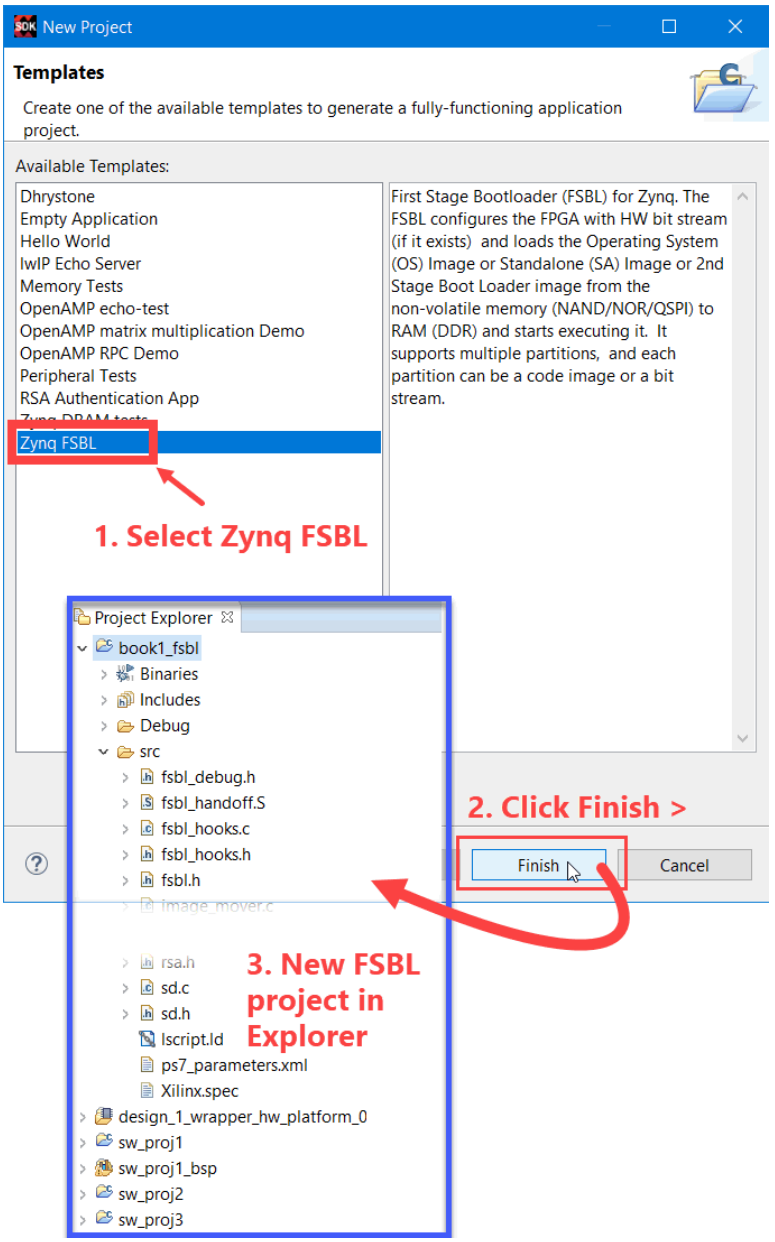

Figure 52. Select the Zynq-7000 FSBL template, and click Finish. The FSBL application is added to the Project Explorer tab.

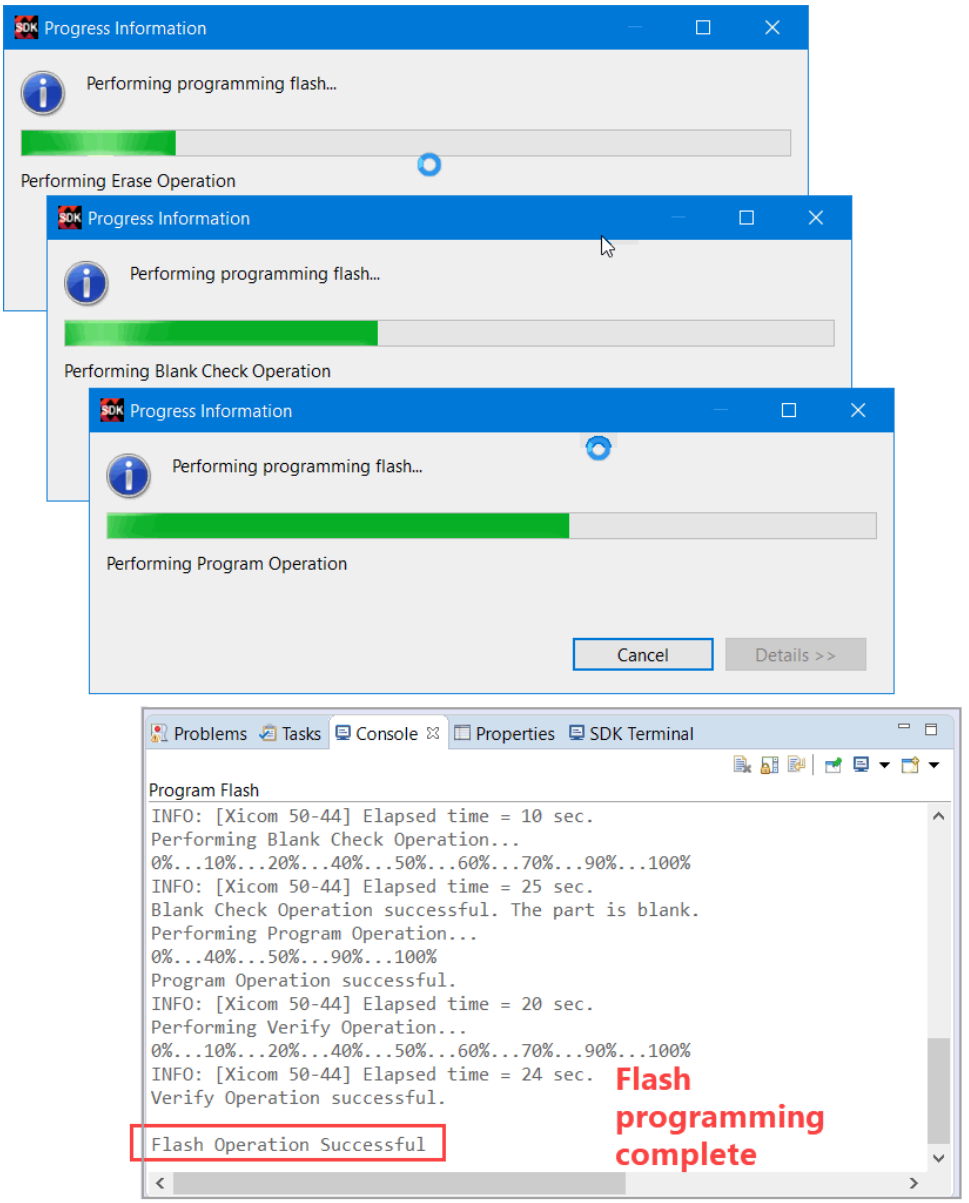

Figure 53. QSPI flash programming progression.

18. Additional Interrupt Topics

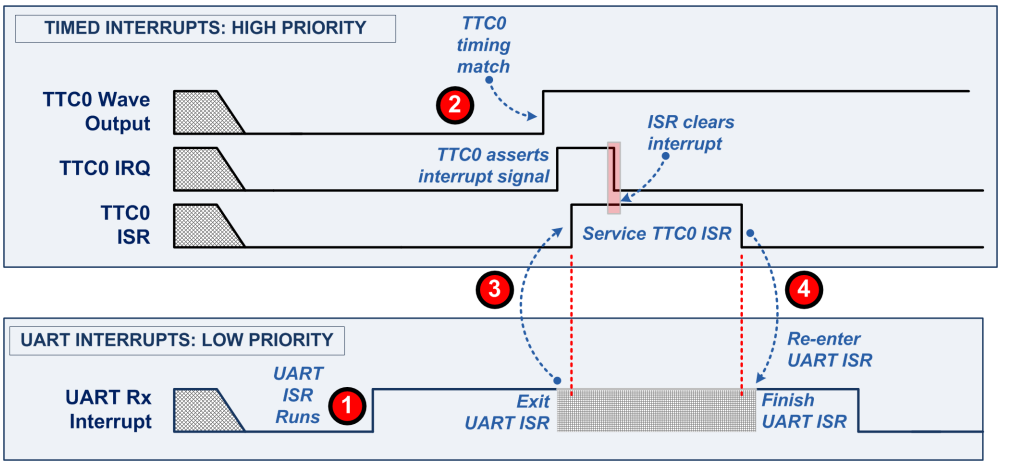

Figure 54. Waveforms for nested interrupts.

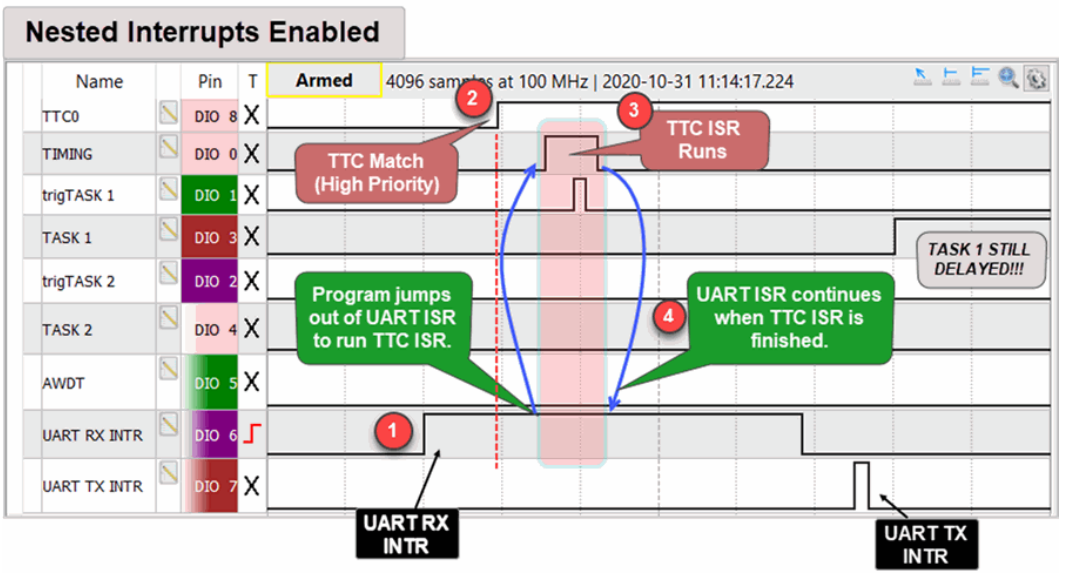

Figure 55. When nested interrupts are enabled, the high-priority timing ISR can pre-empt the low-priority UART ISR. The processor exits the UART ISR, executes the timing ISR, and then jumps back to the UART ISR.

Software Project Summary

Chapter 7 (SW Project 1): Hello World

Chapter 8 (SW Project 2): Zynq-7000 GPIO

- Introduces the Processing System GPIO.
- Introduces the Programmable logic GPIO (AXI IP).
- Describes Xilinx driver initialisation in detail.
- Shows how to add assertion functionality to the code.
- A simple program is implemented using the PS and PL GPIO interfaces. (This code uses a crude For-loop timing method.)

Chapter 9 (SW Project 3): Structured Programming in C

- In this critical chapter, a layered hierarchy is introduced to make it easier to develop and scale the remaining projects in the text.
- The flat-file structure in Software Project 2 is converted to this new layout.

Chapter 10 (SW Project 4): Zynq-7000 Timers and Watchdogs

- Introduces all timing and watchdog options in the Zynq-7000.
- Uses the CPU0 private timer to provide deterministic program timing; this replaces the crude For-loop method used in the first two projects.
- Uses the CPU0 private watchdog to cause the system to reset if an issue is encountered in the code.
- Shows how to use the PS GPIO outputs to monitor relevant code sections in the program flow.

Chapter 11 (SW Project 5): The Zynq-7000 Interrupt System

- Provides an overview of the Generic Interrupt Controller in the Zynq-7000.
- Describes the Zynq-7000 interrupt system.
- Adds the CPU0 private timer to the interrupt system, and uses a related software technique to control program timing.
- Introduces the foreground-background architecture for program flow.

Chapter 12 (SW Project 6): Program Timing Using The TTC

- Discusses the triple timer counter (TTC) in more detail.
- Uses the TTC to control program timing. This effectively replaces the software-based approach in Chapter 11 with a hardware-based approach.

Chapter 13 (SW Project 7): Button Debouncing in Software

- Describes and demonstrates an effective button debounce algorithm.

Chapter 14: (SW Project 8) UART Command Handler Design

- A command handler that can be used with any communications interface is designed. In this case, the Zynq-7000 PS UART is used as the communications block.
- A suitable command handling protocol is defined, and commands are added to read and write system memory.

Chapter 15: Host Application Design

- Shows how to design an application that runs on the host PC, and communicates with the command handler that was developed in Chapter 14.
- Python (Jupyter) and LabVIEW Community Edition are used for host application development.

Chapter 16 (SW Project 9): Programmable Logic Interrupts

- The project in this chapter uses the Digilent Pmod ACL, which is based on the Analog Device ADXL345 accelerometer.
- Basic SPI protocol concepts are explained.
- A main objective is to show how to connect programmable logic interrupts to the interrupt system.
- Embedded software is developed for the Quad-SPI AXI module in the programmable logic.
- The command handler developed in Chapter 14 is scaled up with several Pmod ACL commands, and the Python and LabVIEW host applications are also updated.

Chapter 17: Platform Boot Options

- Describes the bare-metal boot image in detail.
- Summarises the boot sequence when external non-volatile boot media is used.
- Outlines the procedure for SD card boot on the Zybo-Z7-20 platform.
- Outlines the procedure for QSPI boot on the Zybo-Z7-20 platform.

Chapter 18 (SW Project 10): Additional Interrupt Topics:

- Discusses pre-emptive (i.e. nested) interrupt concepts, and shows how to enable nested interrupts in the Zynq-7000.
- Discusses the concept of critical sections in code, and shows how to protect shared variables by disabling interrupts.