Xilinx Zynq-7000 Avnet MiniZed Tutorial

D. W. Hawkins (dwh@caltech.edu)

Version 0.1

July 1, 2019

Contents

1 Introduction **3**

2 MiniZed Getting Started (Out-of-the-Box) **5**

3 Zynq Power-on-Reset and Boot **6**

4 Blinky LED Designs **13**
4.1 PS bicolor LED control via U-Boot . . . 13
4.2 PS bicolor LED control via GPIO MIO . . . 15
4.3 PL bicolor and blue LED control via GPIO EMIO . . . 24
4.4 PL bicolor and blue LED control via AXI GPIO . . . 25
4.5 Scripting the Blinky LED Designs . . . 25

5 Programmable Logic (PL) Only Designs **26**
5.1 Using the FPGA internal oscillator . . . 26
5.2 Using the PS-to-PL clocks . . . 26

6 Peripheral Designs **27**
6.1 QSPI Flash . . . 27
6.2 I2C to PMIC and Sensors . . . 28
6.3 SPI to PMod ADC and DAC . . . 29

A Factory Restore **30**
A.1 Viewing Tool and Software Versions . . . 30
A.2 Restore Image Details . . . 34
A.2.1 QSPI Image Details . . . 36
A.2.2 eMMC Image Details . . . 37
A.2.3 U-Boot Image Versions and Checksums . . . 38
A.3 QSPI Flash Programming . . . 40
A.3.1 XSCT `program_flash` . . . 40
A.3.2 Vivado Flash Programming . . . 42
A.4 Restore Procedure . . . 43

B Resources **47**
B.1 Avnet . . . 47
B.2 Xilinx . . . 47

1 Introduction

The MiniZed is a low-cost (US$89) development board from Avnet containing a Xilinx Zynq-7000 programmable system-on-chip. Figure 1 shows a block diagram of the MiniZed board. The MiniZed features [2, 4]:

- Xilinx Zynq-7000
  - Zynq 7Z007S single-core ARM Cortex-A9 processor (XC7Z007S-1CLG225C)
  - 28nm Artix-7 series programmable logic
    - 3,600 logic slices containing [10]
      - 14,400 6-input LUTs
      - 28,800 registers
    - 50 Block RAMs (BRAMs)
    - 66 DSP Blocks (DSP48E1 with pre-adder, 25×18 multiplier, 48-bit accumulator)
- Memory
  - Micron 512MB DDR3L with 16-bit data bus (MT41K256M16TW-107:P)
  - Micron 128Mb (16MB) QSPI Flash (MT25QL128ABA8E12-1SIT)
  - Micron 8GB eMMC Flash (MTFC8GAKAJCN-4M IT)
- Interface I/O
  - 33.3333MHz PS reference clock (Microchip DSC1001DI1-033.3333T)
  - 1 × PS bi-color LED
  - 1 × PL bi-color LED
  - Reset push button
  - User push button
  - User switch
- Communications
  - Murata "Type 1DX" (LBEE5KL1DX-883) wireless module with
    - WiFi 802.11b/g/n
    - Bluetooth 4.1 plus EDR (Enhanced Data Rate) and BLE (Bluetooth Low Energy)
  - USB 2.0 Host Controller (Microchip USB3320C ULPI PHY)
  - 2 × PMod Connectors (16 GPIO)
  - Arduino-compatible shield interface (22 GPIO)
  - Digilent USB JTAG plus USB COM port interface (based on the FTDI FT2232H)
- Sensors
  - ST Micro LIS2DS12 Motion and Temperature sensor
  - ST Micro MP34DT05 MEMS Microphone
- Power
  - Dialog DA9062 Power Management IC
  - Auxiliary microUSB connector for USB host-mode power

This tutorial is written for FPGA designers new to the Zynq processor. The tutorial provides cross-references to Avnet and Xilinx documentation, walks through several designs to familiarize the user with the Xilinx development tools, Zynq processor configuration, and programmable logic synthesis.

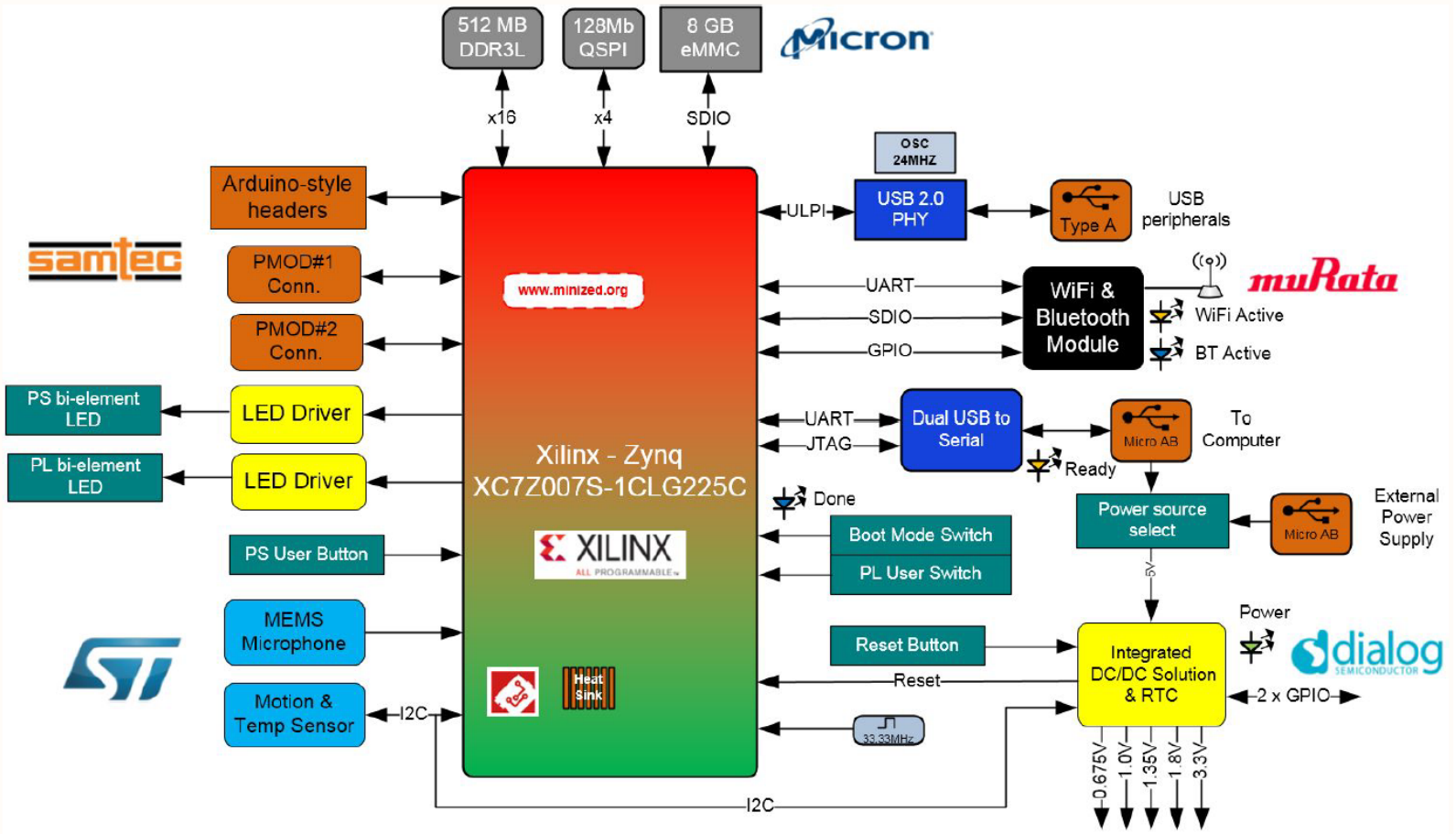


Figure 1: MiniZed block diagram (from the schematic [4]).

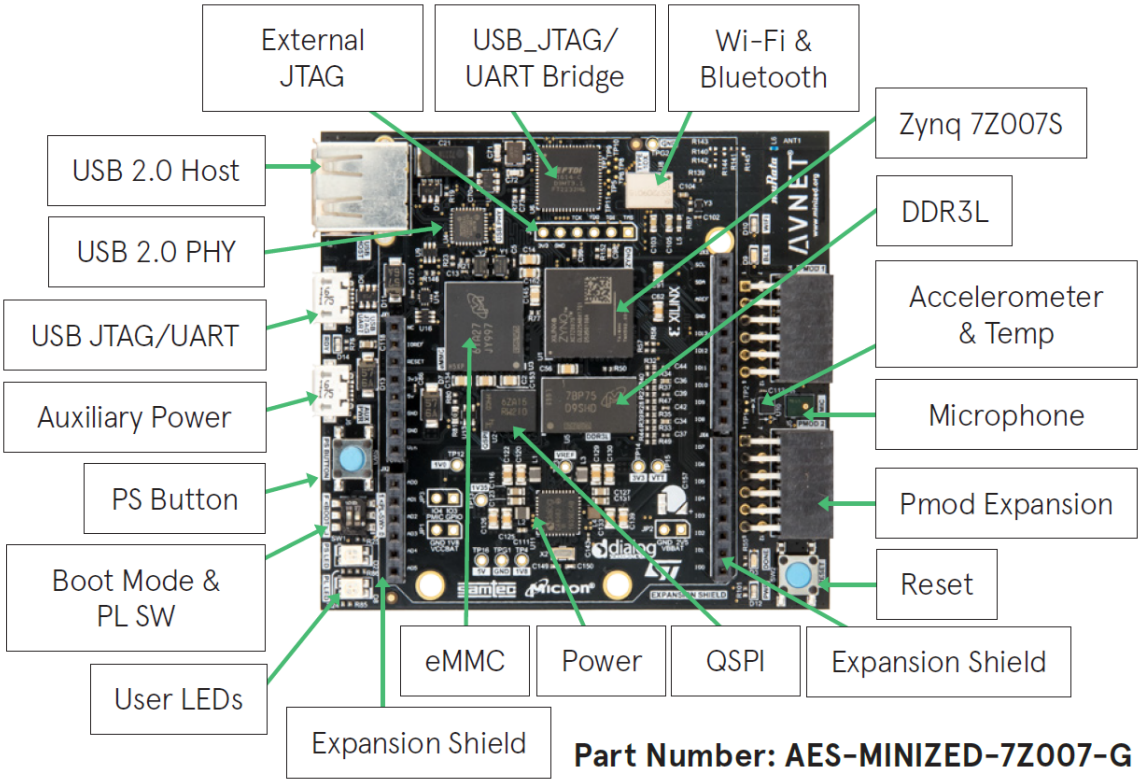


Figure 2: MiniZed peripherals diagram (from the Quick Start Card [3]).

2 MiniZed Getting Started (Out-of-the-Box)

Figure 2 shows a photo of the MiniZed board from the Avnet *Quick Start Card* [3]. The Avnet MiniZed *Quick Start Card* and *Getting Started Guide* [5] contain instructions on how to start using the MiniZed. This section assumes that you have gone through both of these documents and are ready to learn more about how the MiniZed works.

The MiniZed is powered via the USB JTAG/UART shown in Figure 2 (on the left side). The USB JTAG/UART interface is a Digilent design[1]. The USB JTAG/UART circuit uses the FTDI FT2232H dual USB-to-UART/FIFO controller: with the JTAG interface implemented on the first channel, and the UART implemented on the second channel. The MiniZed schematic shows the circuit on page 6 [4]. The circuit shows that the FTDI interface can also be used to toggle the MiniZed (power manager) Power-on-Reset (POR) and (Zynq) System Reset (SRST)[2].

The MiniZed boots from QSPI flash, loads the Xilinx-generated First Stage Boot Loader (FSBL), which configures the FPGA programmable logic (causing the blue LED to turn on) and loads U-Boot (the Second State Boot Loader), and U-Boot then loads Linux from the eMMC card. The designs in this tutorial use both the QSPI and eMMC flash. Appendix A shows how to view the MiniZed U-Boot and Linux versions, and how to restore the board to factory condition. Restoring the board to factory condition allows the next user to begin using the MiniZed from the out-of-the-box starting point.

[1]Digilent provides development boards to the Xilinx University Program, and is part of National Instruments.
[2]The MiniZed hardware user guide contains a block diagram of the reset sources in Figure 7 on page 22 [2].

3 Zynq Power-on-Reset and Boot

The Zynq processor power-on-reset and boot procedure is documented in the *Technical Reference Manual* (TRM) and *Software Developer User Guide* [10, 11]. The Zynq boot and configuration design hub contains additional cross-references on the power-on-reset and boot flow.

The MiniZed board supports two boot modes; JTAG and QSPI Flash. The MiniZed Zynq processor determines the boot mode based on the *boot mode pin settings* that are read after power is valid, when power-on-reset (`PS_POR_B`) deasserts[3]. Figure 3 shows how seven MIO pins control the boot mode. Figure 4 shows the MiniZed connections for the boot mode pins. Table 1 summarizes the MiniZed boot mode pin connections.

Table 1: MiniZed boot mode MIO pin settings.

MIO Pin(s)	Logic	Description
8:7	0b	MIO Bank 1+0 voltages are both 3.3V (pp3-4 [4])
6	0b	Execute BootROM after the PS clock PLLs lock
5 5:3	0b or 1b 000b 100b	JTAG/Flash boot mode switch • JTAG boot mode • QSPI Flash boot mode
2	0b	JTAG chain is cascaded (ARM processor and FPGA logic)

The Zynq processor JTAG boot mode is useful during software development. The SDK can be used to configure the programmable logic, configure the processor (using `ps7_init.tcl`), download the application image, and then run or debug the application. The programmable logic is configuring by the SDK using the *bitstream* included in the hardware definition file exported from Vivado.

The MiniZed QSPI Flash boot mode flow is:

1. Power is valid and power-on-reset deasserts.
2. The hardware samples the boot strap pins (QSPI Flash mode) and enables the PS clock PLLs.
3. When the PS clock PLLs lock, the PS starts executing the on-chip BootROM code.
4. The BootROM code reads the BootROM Header (Boot Image Header) from QSPI Flash.
5. The BootROM code loads the First Stage Boot Loader (FSBL) into On-chip memory (OCM).
6. The BootROM code starts executing the FSBL (ending the BootROM execution).
7. (Optional) The FSBL code configures the programmable logic.
8. The FSBL code determines what happens next, eg., on the MiniZed it typically loads and executes the second stage boot loader (SSBL), U-Boot.
9. The U-Boot code loads and executes Linux.

[3]The Zynq TRM shows the required timing relationship between the power-on-reset and system reset signals in Figure 6-4: *Power and Reset Sequencing Waveforms* on p164 [10].

Pin-signal / Mode	MIO[8]	MIO[7]	MIO[6]	MIO[5]	MIO[4]	MIO[3]	MIO[2]
	VMODE[1]	VMODE[0]	BOOT_MODE[4]	BOOT_MODE[0]	BOOT_MODE[2]	BOOT_MODE[1]	BOOT_MODE[3]
Boot Devices							
JTAG Boot Mode; cascaded is most common(1)				0	0	0	**JTAG Chain Routing**(2) 0: Cascade mode 1: Independent mode
NOR Boot(3)				0	0	1	
NAND				0	1	0	
Quad-SPI(3)				1	0	0	
SD Card				1	1	0	
Mode for all 3 PLLs							
PLL Enabled			0	Hardware waits for PLL to lock, then executes BootROM.			
PLL Bypassed			1	Allows for a wide PS_CLK frequency range.			
MIO Bank Voltage(4)							
	Bank 1	**Bank 0**	Voltage Bank 0 includes MIO pins 0 thru 15. Voltage Bank 1 includes MIO pins 16 thru 53.				
2.5 V, 3.3 V	0	0					
1.8 V	1	1					

Notes:

1. JTAG cascaded mode is most common and is the assumed mode in all the references to JTAG mode except where noted.
2. For secure mode, JTAG is not enabled and MIO[2] is ignored.
3. The Quad-SPI and NOR boot modes support execute-in-place (this support is always non-secure)
4. Voltage Banks 0 and 1 must be set to the same value when an interface spans across these voltage banks. Examples include NOR, 16-bit NAND, and a wide TPIU test port. Other interface configuration may also span the two banks.

Figure 3: Zynq boot mode MIO pins (from Table 6-4, p166 [10]).

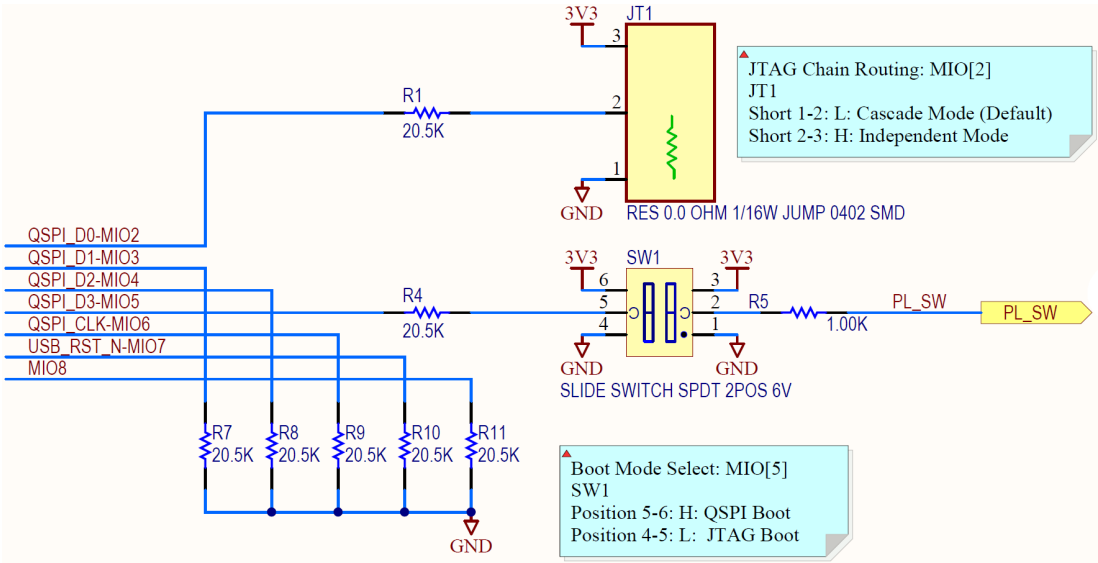


Figure 4: MiniZed Zynq boot mode MIO pins (from schematic p3 [4]).

The Zynq processor memory map during BootROM execution, and after handoff to the FSBL, is given in Figure 6-11 [10]: the main change in the memory map is the presence of the 64kB BootROM (at address 0x00030000) after the 192kB OCM (at address 0) during BootROM execution.

Figure 5 shows the format of the BootROM Header (Boot Image Header) in the QSPI flash boot image. The BootROM header is defined in the Zynq *Technical Reference Manual* (TRM) [10], while boot image creation using the `bootgen` tool is described in the *Software Developers Guide* (pp48-51, and Appendix A [11]). The format of the MiniZed QSPI Flash boot image is typically:

- BootROM Header (Boot Image Header)
- First Stage Boot Loader (FSBL)
- Programmable Logic bitstream
- U-Boot (the second stage bootloader)
- Additional partitions, eg., Linux and the device tree.

Insight into the BootROM Header can be gained by decoding an example. The BootROM header for the MiniZed `flash_fallback_7007S.bin` image can be viewed using U-Boot via:

```
Zynq> sf probe
SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> sf read 10000000 0 1000000
device 0 whole chip
SF: 16777216 bytes @ 0x0 Read: OK
Zynq> crc32 10000000 fc0be4
crc32 for 10000000 ... 10fc0be3 ==> 005b7b3c
Zynq> md 10000000 30
10000000: eafffffe eafffffe eafffffe eafffffe    ................
10000010: eafffffe eafffffe eafffffe eafffffe    ................
10000020: aa995566 584c4e58 00000000 01010000    fU..XNLX........
10000030: 00001700 00018008 00000000 00000000    ................
10000040: 00018008 00000001 fc164530 00000000    ........0E......
10000050: 00000000 00000000 00000000 00000000    ................
10000060: 00000000 00000000 00000000 00000000    ................
10000070: 00000000 00000000 00000000 00000000    ................
10000080: 00000000 00000000 00000000 00000000    ................
10000090: 00000000 00000000 000008c0 00000c80    ................
100000a0: ffffffff 00000000 ffffffff 00000000    ................
100000b0: ffffffff 00000000 ffffffff 00000000    ................
Zynq>
```

The U-Boot commands copy the QSPI flash to DDR memory, calculates the image checksum (to confirm it matches the value in Table 3), and displays the first 48×32-bit words (192-bytes) of the Boot Image Header. The image details are:

1. `0x000`: **Interrupt Table for Execution-in-Place**

   The interrupt table is the first 8×32-bit words of the header. The 32-bit value 0xEAFFFFFE corresponds to an ARMv7 branch-to-self instruction [1], so each interupt handler will *lock-up* the processor.

2. `0x020`: **Width Detection**

   0xAA995566 is used for width detection (p172 [10]).

Header Address	32-bit Word	Parameter	Boot Device		
			Secure Usage	Non-Secure Usages	
			OCM	OCM	Execute In Place (6)
0x000 – 0x01F	0 - 7	Interrupt Table for Execution-in-Place	no	no	yes
0x020	8	Width Detection	Quad-SPI	Quad-SPI	Quad-SPI
0x024	9	Image Identification	yes	yes	yes
0x028	10	Encryption Status	yes	yes	yes
0x02C	11	FSBL/User Defined (3)	~	~	~
0x030	12	Source Offset	yes	yes	~
0x034	13	Length of Image	yes	yes	set = 0
0x038	14	FSBL Load Address	~	~	set = 0
0x03C	15	Start of Execution	yes	yes	yes
0x040	16	Total Image Length	note (1)	note (2)	set = 0
0x044	17	QSPI Config Word ,set to 0x01	~	~	~
0x048	18	Header Checksum	yes	yes	yes
0x04C – 0x097	19 - 39	FSBL/User Defined (76-Byte)(3)	~	~	~
0x0A0 – 0x89F	40 - 551	Register Initialization (2048-Byte) (4)	yes	yes	yes
0x8A0 – 0x8BF	552 - 559	Image Header(3)	yes	yes	yes
0x8C0 – 0x8FF	560 - 576	Partition Header	yes	yes	yes
0x900	577 and up	FSBL Image or User Code	192 KB	192 KB	see (5)

Notes:

1. In secure mode, the Total Image Length parameter is greater than Length of Image parameter because of encryption.
2. In non-secure OCM mode, the Total Image Length parameter must be set equal to the Length of Image parameter.
3. The FSBL/User Defined parameter and FSBL Image and User Code fields are not used by the BootROM code and do not affect the hardware. See UG821, *Zynq-7000 SoC Software Developers Guide* for more information.
4. The addresses that can be accessed by Register Initialization is restricted, see Table 6-7. The secure boot mode has more address restrictions than a non-secure boot.
5. The size of the FSBL image (or User code) for Execute-in-place depends on the allowed capacity of the boot device less the 0x8C0 (the size of the BootROM Header). The maximum Quad-SPI size is described in section 6.3.4 Quad-SPI Boot. For NOR, refer to section 6.3.6 NOR Boot.
6. To select the execute-in-place feature, set the Length of Image and Total Image Length parameters to 0 and load the PS interconnect address into the Source Offset.

Figure 5: Zynq BootROM Image Header (from Table 6-5, pp171-172 [10]).

3. `0x024:` **Image Identification**

   0x584C4E58 corresponds to 8-bit ASCII codes for X, L, N, and X in little-endian format, which is why the ASCII value displayed by U-Boot is XNLX (p172 [10]).

4. `0x028:` **Encryption Status**

   0x00000000 indicates the image is not encrypted.

5. `0x02C:` **FSBL/User Defined**

   0x01010000 indicates the BootROM header version (p173 [10]).

6. `0x030:` **Source Offset**

   0x00001700 offset in bytes to the FSBL image. The flash contents near this offset are:

```
Zynq> md 100016e0 10
100016e0: ffffffff ffffffff ffffffff ffffffff    ................
100016f0: ffffffff ffffffff ffffffff ffffffff    ................
10001700: ea000049 ea000025 ea00002b ea00003b    I...%...+...;...
10001710: ea000032 e320f000 ea000000 ea00000f    2..... .........
```

   The locations prior to offset 0x1700 are blank, and the data starting at offset 0x1700 look like ARMv7 instructions.

7. `0x034:` **Length of Image**

   0x00018008 the image length is 96kbytes plus 8-bytes.

8. `0x038:` **FSBL Load Address**

   0x00000000 load to the start of OCM.

9. `0x03C:` **Start of Execution**

   0x00000000 execute from the start of OCM.

10. `0x040:` **Total Image Length**

    0x00018008 is the same as `0x034:` **Length of Image**.

    For non-secure boot mode, 0x040 and 0x034 must match (p174 [10]).

11. `0x044:` **QSPI Configuration Word**

    0x00000001 matches expected value (p174 [10]).

12. `0x048:` **Header Checksum**

    0xFC164530 matches the inverted sum of 32-bit words 0x020 to 0x044, i.e., ~((0xAA995566 + 0x584C4E58 + 0x01010000 + 0x00001700 + 0x00018008 + 0x00018008 + 0x00000001) & 0xFFFFFFFF).

13. `0x04C:` **FSBL/User Defined** (76-bytes)

    Bytes 0x04C to 0x097 are all zero (unused). These bytes can be defined using `bootgen` and the `udf_field` in the BIF file (see Appendix A [11]).

14. `0x098:` **Image Header Table Offset**

    0x000008C0 Image Header Table offset.

    Note: this field is missing from Table 6-5, but is listed in the text, with the wrong name! (see p174 [10]).

15. `0x09C`: **Partition Header Table Offset**

    0x00000C80 Partition Header Table offset.

    Note: this field is missing from Table 6-5, but is listed in the text, with the wrong description! (see p175 [10]).

16. `0x0A0`: **Register Initialization** (2048-bytes)

    The repeated pattern of 0xFFFFFFFF 0x00000000 indicates that the register initialization block is not used (the register address 0xFFFFFFFF terminates register initialization (p175 [10]).

17. `0x8A0`: **Image Header**

    Table 6-5 refers to this as the Image Header area, however, the text on p176 indicates this is FSBL/User Defined. The Image Header Table offset points to 0x8C0, not 0x8A0. Viewing the copy of the QSPI flash image in DDR using U-Boot gives:

```
Zynq> md 100008a0 58
100008a0: ffffffff ffffffff ffffffff ffffffff    ................
100008b0: ffffffff ffffffff ffffffff ffffffff    ................
100008c0: 01020000 00000004 00000320 00000240    ........ ...@...
100008d0: 00000000 ffffffff ffffffff ffffffff    ................
100008e0: ffffffff ffffffff ffffffff ffffffff    ................
100008f0: ffffffff ffffffff ffffffff ffffffff    ................
10000900: 00000250 00000320 00000000 00000001    P... ............
10000910: 7a796e71 5f667362 6c2e656c 66000000    qnyzbsf_le.l...f
10000920: 00000000 ffffffff ffffffff ffffffff    ................
10000930: ffffffff ffffffff ffffffff ffffffff    ................
10000940: 00000260 00000330 00000000 00000001    `...0...........
10000950: 64657369 676e5f31 5f777261 70706572    ised1_ngarw_repp
10000960: 2e626974 00000000 00000000 ffffffff    tib.............
10000970: ffffffff ffffffff ffffffff ffffffff    ................
10000980: 00000270 00000340 00000000 00000001    p...@...........
10000990: 752d626f 6f742e65 6c660000 00000000    ob-ue.to..fl....
100009a0: ffffffff ffffffff ffffffff ffffffff    ................
100009b0: ffffffff ffffffff ffffffff ffffffff    ................
100009c0: 00000000 00000350 00000000 00000001    ....P...........
100009d0: 696d6167 652e7562 00000000 00000000    gamibu.e........
100009e0: ffffffff ffffffff ffffffff ffffffff    ................
100009f0: ffffffff ffffffff ffffffff ffffffff    ................
```

    This shows that the bytes at offset 0x8A0 are unused, and the bytes at 0x8C0 are used. The **Image Header Table Offset** 0x098 indicates there is where the image header table is located. The format of the image header table is defined in Table A-4, p63 [11]. The first word matches the expected value of 0x01020000. The second word indicates there are 4 image headers. The third word indicates the partition header is at word offset 0x320, which is byte offset 0xC80, and that value matches the value for the **Partition Header Table Offset** 0x09C. The forth word indicates the first image header is at word offset 0x240, which is byte offset 0x900. The fifth word 0x00000000 is unused, and the values 0xFFFFFFFF pad the header table to the next 64-byte (0x40) boundary.

18. `0x8C0`: **Partition Header**

    Table 6-5 refers to this as the Partion Header area, however, the decoding of the header indicates there is where the Image Header Table is located. The image header was decoded in the previous step.

19. `0x900`: **FSBL Image or User Code**

    The Image Header Table indicates that this is the location of the first of the array of image headers. The Image Header format is defined in in Table A-7, p66 [11]. Decoding the strings for the four image names gives; `zynq_fsbl.elf`, `design_1_wrapper.bit`, `u-boot.elf`, and `image.ub`. This indicates that the QSPI flash fallback image contains the FSBL, a bitfile, U-boot, and a Linux image, i.e., exactly what we would expect to find in the fallback image.

The manual decoding of the MiniZed QSPI flash fallback image has exposed some minor inconsistencies in the Xilinx documentation, however, the documentation provides sufficient information to interpret a boot image.

4 Blinky LED Designs

The MiniZed has three LEDs;

- A bicolor LED connected to the Processor System (PS).
- A bicolor LED connected to the Programmable Logic (PL).
- A blue LED connected to the programmable logic configuration controller DONE output.

The design examples in this section show how to blink the LEDs using software running on the PS. The hardware designs differ in how the LEDs are controlled by the PS;

- PS bicolor LED control via PS GPIO MIO (no PL use).
- PL bicolor and blue LED control via the PS GPIO EMIO.
- PL bicolor and blue LED control using an AXI GPIO.

These simple designs demonstrate how the PS and PL are configured using Vivado, and how bare-metal applications are created using the Software Development Kit (SDK).

4.1 PS bicolor LED control via U-Boot

The MiniZed schematic (p4 [4]) shows that the PS bicolor LED is controlled by the Zynq GPIO signals MIO52 (`PS_LED_R`) and MIO53 (`PS_LED_G`). The Zynq TRM describes the GPIO multiplexed I/O (MIO) and extended multiplexed I/O (EMIO) in Chapter 14 [10]. Figure 6 shows the GPIO block diagram; banks 0 and 1 connect to the MIO, while banks 2 and 3 connect to the EMIO. Figure 6 shows the GPIO control registers. The GPIO control register definitions and location in the Zynq address map is defined in Appendix B.19 [10]. The GPIO registers are located at address 0xE000A000.

The PS bicolor LED is controlled by MIO bits 52 and 53 which are located in GPIO bank 1. The GPIO bank 1 control registers are; `DATA` (0xE000A00C), `DIRM` (0xE000A244), and `OEN` (0xE000A248) registers. The PS bicolor LED can be controlled from U-Boot as follows:

```
# Set the direction
Zynq> mw e000a244 00300000

# Enable the output (the bicolor LED turns off)
Zynq> mw e000a248 00300000

# Write to the registers using the mask
Zynq> mw e000a00c ffcf0000  # off
Zynq> mw e000a00c ffcf0010  # red
Zynq> mw e000a00c ffcf0020  # green
Zynq> mw e000a00c ffcf0030  # red+green
```

The PL LEDs can also be controlled using U-Boot commands, however, the programmable logic first needs to be configured with a design that either routes the EMIO to the LEDs, or implements the AXI GPIO that controls the LEDs. Using U-Boot for read and write access to Zynq PS registers and PL designs is a simple, but powerful, debugging tool.

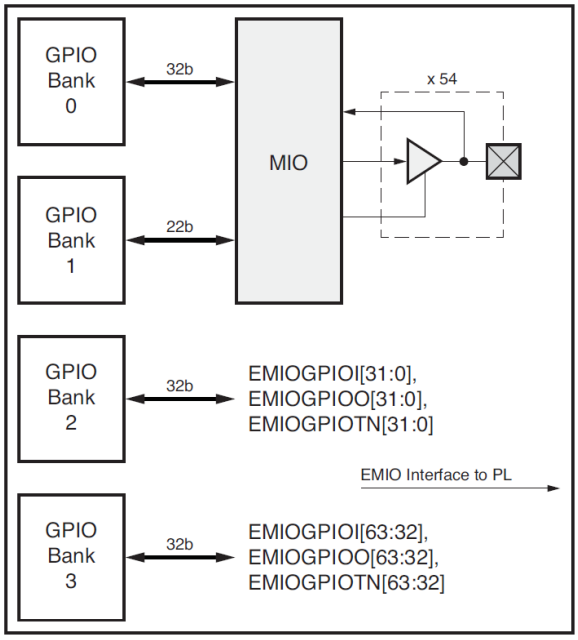


Figure 6: Zynq GPIO block diagram (from Figure 14-1 [10]).

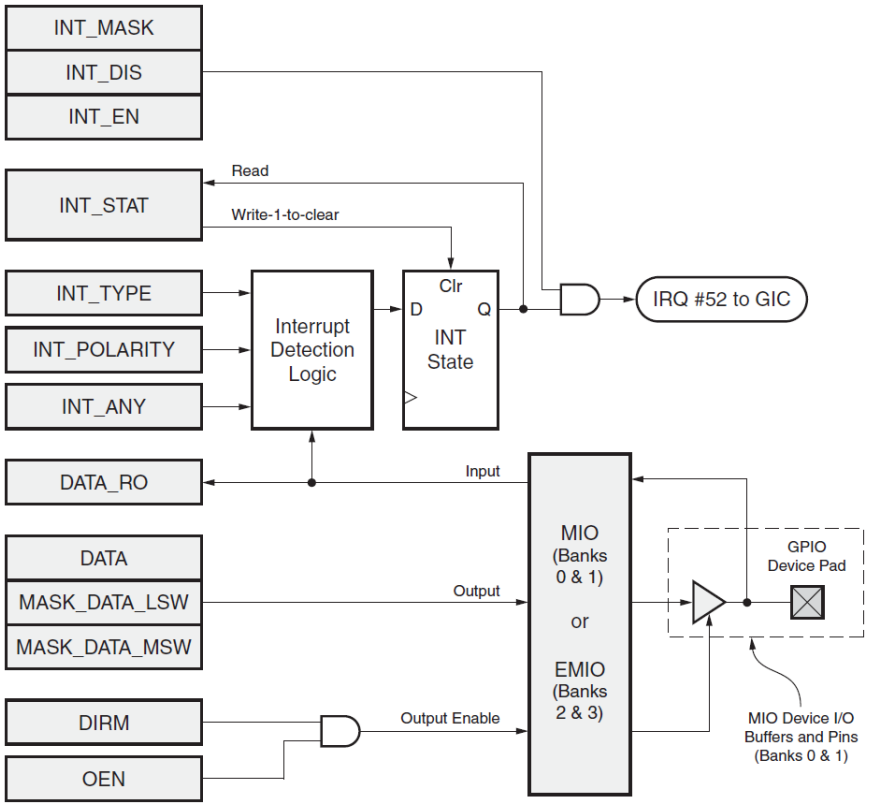


Figure 7: Zynq GPIO control registers (from Figure 14-2 [10]).

4.2 PS bicolor LED control via GPIO MIO

The PS bicolor LED control via the GPIO MIO design uses only the Processor System (PS), it does not use the Programmable Logic (PL). The Xilinx Vivado tool is used to configured the Zynq Processor, and the Xilinx Software Development Kit (SDK) is used to develop a bare-metal application to blink the bicolor LED. The PS bicolor LED design duplicates many of the steps found in Avnet MiniZed tutorials 01, 02, and 04 [7–9]. The Avnet tutorials should be reviewed in conjunction with this design.

1. **Vivado Project Creation**

   Create the MiniZed board `blinky_gpio_mio` project:

   (a) Start Vivado.

   (b) Under *Quick Start*, click on *Create Project*.

   (c) **Create a New Vivado Project**

   - Click *Next*

   (d) **Project Name**

   - *Project name*: `blinky_gpio_mio`
   - *Project location*: `c:/temp/minized/blinky_gpio_mio/vwork`
   - Uncheck *Create project subdirectory*
   - Click *Next*

   The project location specifies the Vivado work directory, `vwork`, where Vivado generates the project files. After the GPIO MIO project has been completed, a script is developed that automates the project generation.

   (e) **Project Dialog**

   - Select the *RTL Project* bullet
   - Check *Do not specify sources at this time*
   - Click *Next*

   (f) **Default Part**

   - Click on the *Boards* tab
   - On the *Vendor* pull-down, select *em.avnet.com*
   - Select the *MiniZed* in the board list
   - Click *Next*

   (g) **New Project Summary**

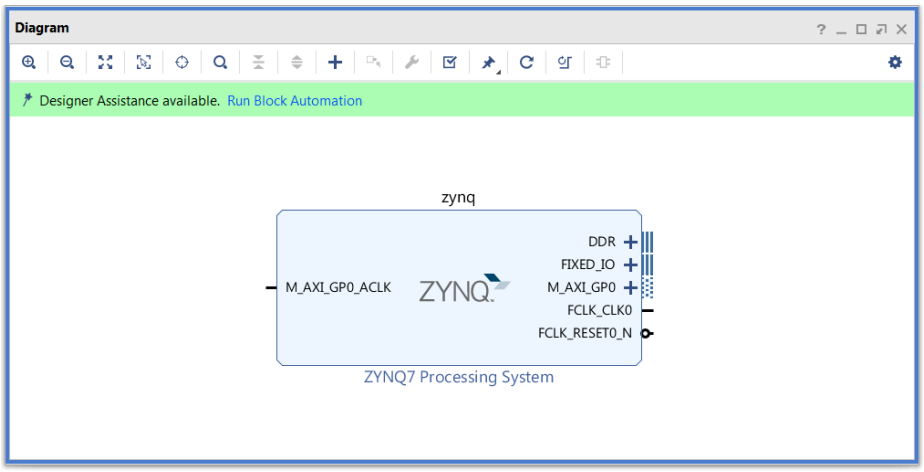
   - Click *Finish*

2. **Vivado Block Diagram Creation**

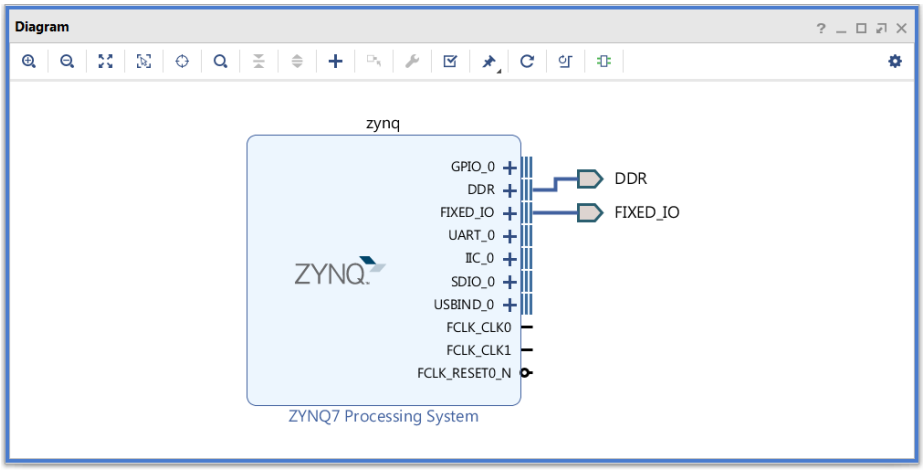
   (a) In the *Project Manager*, under *IP Integrator*, select *Create Block Design*.

   (b) **Create Block Design**

   - *Design name*: `system`
   - *Directory*: `<Local to Project>`
   - *Specify Source set*: `Design Sources`
   - Click *OK*



(a) Zynq processor



(c) Zynq processor after block automation

Figure 8: PS bicolor LED control via GPIO MIO design Vivado diagram window.

(c) **Add the Zynq Processor**

- Click the '+' in the middle of the design area to open the IP list
- In the *Search* bar enter `Zynq`
- Double-click on `ZYNQ7 Processing System` to add the Zynq to the design
- In the **Block Properties** dialog, change the name from `processing_system7_0` to `zynq`.

(d) **Run Block Automation**

- Click on the blue hyperlink Run Block Automation
- Accept the block automation defaults by clicking *OK*

Figure 8(a) shows the diagram window after adding the Zynq to the `system` design. Figure 8(b) shows the diagram window after running block automation. Block automation configures the Zynq processor using the board presets, `preset.xml`, which is part of the MiniZed board definition files (BDF) located in the directory:

`C:\software\Xilinx\Vivado\2019.1\data\boards\board_files\minized\1.2\`

This example uses the MiniZed Zynq processor defaults. Later design examples customize the Zynq processor.

3. **Vivado Synthesis**

   The need for *Synthesis* in a design that does not use the programmable logic sounds redundant. However, the Zynq design flow uses Vivado to configure the (required) Zynq processor system and the (optional) programmable logic. Vivado generates the hardware definition file (HDF) used by the SDK. Vivado was used to configure the Zynq with the MiniZed presets, so that the SDK can be used to create the blinky LED application. The hardware support files needed by the SDK are generated as follows.

   (a) **Create HDL Wrapper**

   - In the Vivado **Sources** window, highlight `system (system.bd)`
   - Right-mouse-click and select *Create HDL Wrapper*
   - Accept the default option *Let Vivado manage wrapper and auto-update*
   - Click *OK*
   - This generates the top-level design file `system_wrapper.v`

   (b) **Constraints**

   This design uses only Zynq processor system I/Os, no programmable logic I/Os, so pin constraints are not required. The Zynq pins are configured by the FSBL, which is generated by Vivado as part of the synthesis step.

   (c) **Run Synthesis**

   - In the Vivado **Project Manager** click *Run Synthesis*
   - Accept the defaults in the **Launch Runs** dialog
   - Click *OK*
   - Synthesis generates output in the directory `vwork/blinky_gpio_mio.runs`
   - When the **Synthesis Completed** dialog appears, close it by clicking *Cancel*

   (d) **Export Hardware**

   - Select *File→Export→Export Hardware*
   - Leave the *Include bitstream* checkbox unchecked
   - Accept the default *Export to:* `<Local to Project>`
   - Click *OK*
   - This step generates the HDF file used by the SDK `vwork/blinky_gpio_mio.sdk/system_wrapper.hdf`

(e) **HDF Analysis** (optional step)

- Create a copy of the HDF file
- Change the extension from `.hdf` to `.zip`
- Unzip the file and view the contents of `system_wrapper`:
  - `ps7_init.tcl`
    Used by the SDK to initialize the processor during JTAG download
  - `ps7_init*.h/c`
    First stage bootloader (FSBL) source.
  - `system_bd.tcl`
    Block design Tcl script.
- Delete the zip file and the `system_wrapper` folder

4. **SDK Project Creation**

(a) **Method 1**: Start the SDK tool from Vivado

- Select the Vivado menu option *File→Launch SDK*
- Accept the **Launch SDK** dialog defaults and click *OK*
- This method sets up the SDK to use the `.sdk` folder within the Vivado work folder, i.e.,
  `blinky_gpio_mio\vwork\blinky_gpio_mio.sdk`
- The SDK unzips the `.hdf` archive into the folder
  `blinky_gpio_mio\vwork\blinky_gpio_mio.sdk\system_wrapper_hw_platform_0`
- The SDK opens and displays the **Address Map for processor ps7_cortexa9_0**, (which is generated from the settings in `system.hdf`). Table 2 shows a selection of addresses relevant to the GPIO MIO design.
- Double-mouse-click on `ps7_init.html` to see a detailed view of the address map, and the register settings. The registers are configured when the SDK runs `ps7_init.tcl` during JTAG download, or when the processor boots and runs the FSBL.
- Scroll down to **MIO 52** and **MIO 53** to see that the multiplexed I/O (MIO) used to control the red and green PS LEDs are configured as GPIO.
- Scroll further down and see that the PS-to-PL clocks **FPGA0, 1, 2, 3** are configured for frequencies of 50MHz, 100MHz, 10MHz, and 10MHz. The PS-to-PL clocks are not used in the GPIO MIO example.

(b) **Method 2**: Start the SDK tool from the Windows Start Menu (or from a Linux console)
The Avnet MiniZed Tutorial 02 [8] shows how to start the SDK and import a Vivado hardware platform. The Avnet tutorial shows the SDK steps that are automated by starting the SDK from within Vivado. The manual method of creating the SDK is useful if you want to change the SDK project names from the defaults used by Vivado.

Table 2: PS bicolor LED GPIO MIO Example Address Map (selected addresses).

Zynq Component	Cell	Base Address
Memory		
On-Chip Memory#0	`ps7_ram_0`	`0x00000000`
DDR Memory	`ps7_ddr_0`	`0x00100000`
QSPI Linear (Read-Only) Addresses	`ps7_qspi_linear_0`	`0xFC000000`
On-Chip Memory#1	`ps7_ram_1`	`0xFFFF0000`
Peripheral		
UART#0 Registers	`ps7_uart_0`	`0xE0000000`
UART#1 Registers	`ps7_uart_1`	`0xE0001000`
GPIO MIO Registers	`ps7_gpio_0`	`0xE000A000`
QSPI Registers	`ps7_qspi_0`	`0xE000D000`
DDR Registers	`ps7_ddrc_0`	`0xF8006000`

5. **SDK Board Support Package (BSP) Creation**

   (a) Select *File→New→Board Support Package*

   (b) Accept the default settings for the project `standalone_bsp_0` and click *Finish*.

   (c) **Board Support Package Settings**
   The MiniZed connects the Zynq UART#0 to the bluetooth device and UART#1 to the FTDI USB UART interface. The board support package needs to be configured to use UART#1 for standard input/output as follows:

   - Select *Overview*, *standalone*
   - Change the pull-downs for `stdin` and `stdout` to `ps7_uart_1`
   - Click *OK*

   After the BSP has been generated, under *Project Explorer*, expand `standalone_bsp_0`, and double-mouse-click on `system.mss` to see the documentation and example designs for the components in the system.

6. **SDK Baremetal Application Creation**

   The PS bicolor LED control application is created by starting with *Hello World* example application. The *Hello World* application configures the SDK project to use the `platform.h/c` files, which define the functions for initialization and cleanup. The platform initialization configures the UART interface to be used for C library standard input and output.

   (a) Select *File→New→Application Project*

   (b) **Application Project**
   Configure the project name, and change the project to the BSP just generated

   - *Project name:* `blinky_gpio_mio`
   - *OS Platform:* `standalone`

- *Hardware Platform:* `system_wrapper_hw_platform_0`
- *Language:* `C`
- *Board Support Package: Use Existing* `standalone_bsp_0`

(c) Click *Next*

(d) Select *Hello World*

(e) Click *Finish*

(f) Under the *Project Explorer*, expand the `blinky_gpio_mio` project

(g) Right-mouse-click on `src`, select `helloworld.c`, right-mouse-click, select *Rename* and rename the file `blinky_gpio_mio.c`

(h) Replace `blinky_gpio_mio.c` with the source code shown in Figure 9.

The SDK will automatically build the GPIO MIO bare-metal application.

7. **SDK Download and Run the Application**

(a) Set the MiniZed Flash/JTAG mode switch to JTAG mode

(b) Connect the MiniZed to your development machine using the USB JTAG/UART interface

(c) Select the *SDK Terminal*, click the green + symbol, and setup the UART communications to the MiniZed at 115200 baud (the timeout can be set to 1 second)

(d) Under the *Project Explorer*, select the `blinky_gpio_mio` project, right-mouse-click, and select *Run As→Launch on Hardware (System Debugger)* to download and run the application

(e) The PS LED will cycle through off, green, red, and amber (red+green)

(f) Select the *SDK Terminal* and you will see a message for each LED change

Congratulations! You have run your first custom application on the MiniZed board.

8. **SDK Create the First Stage Boot Loader (FSBL)**

The procedure to create the FSBL is similar to the procedure used to create a new application, however, no user customization is required, as the FSBL uses information (register settings) defined in the hardware definition file exported from Vivado.

(a) Select *File→New→Application Project*

(b) **Application Project**

Configure the project name, and change the project to the BSP just generated

- *Project name:* `zynq_fsbl`
- *OS Platform:* `standalone`
- *Hardware Platform:* `system_wrapper_hw_platform_0`
- *Language:* `C`
- *Board Support Package: Create New* `zynq_fsbl_bsp`
  A new BSP is needed as the FSBL requires `xilffs` library support.

(c) Click *Next*

(d) Select *Zynq FSBL*

(e) Click *Finish*

(f) Modify the BSP to use `ps7_uart_1` for stdin and stdout.

```
#include <sleep.h>
#include "platform.h"
#include "xil_printf.h"

int main()
{
    int i = 0;

    init_platform();

    xil_printf("MiniZed Blinky GPIO MIO Example\r\n");
    xil_printf("-------------------------------\r\n");

    // Clear the data (DATA) bits
    *(volatile unsigned int *)0xE000A00C = 0xFFCF0000;

    // Set the direction (DIRM) to output
    *(volatile unsigned int *)0xE000A244 = 0x00300000;

    // Enable the outputs (OEN)
    *(volatile unsigned int *)0xE000A248 = 0x00300000;

    // Blink the bicolor LED
    while (1) {
        switch (i%4) {
            case 0:
                xil_printf("%d: Off\r\n", i);
                *(volatile unsigned int *)0xE000A00C = 0xFFCF0000;
                break;

            case 1:
                xil_printf("%d: Green\r\n", i);
                *(volatile unsigned int *)0xE000A00C = 0xFFCF0020;
                break;

            case 2:
                xil_printf("%d: Red\r\n", i);
                *(volatile unsigned int *)0xE000A00C = 0xFFCF0010;
                break;

            default:
                xil_printf("%d: Amber\r\n", i);
                *(volatile unsigned int *)0xE000A00C = 0xFFCF0030;
                break;
        }
        sleep(1);
        i++;
    }

    cleanup_platform();
    return 0;
}
```

Figure 9: PS bicolor LED GPIO MIO bare-metal application.

The SDK will automatically build the FSBL BSP and FSBL. The SDK Console output should contain the message `Finished building target: zynq_fsbl.elf`.

9. **SDK Create the QSPI Flash Boot Image**

   (a) In the SDK, highlight the `blinky_gpio_mio` application.

   (b) From the SDK main menu, select *Xilinx→Create Boot Image*.
   Figure 10 shows the Create Boot Image dialog. The defaults will generate the file `BOOT.bin`. The Avnet MiniZed Tutorial 04 indicates that the boot image file type should be changed to .MCS [9], however, this is not necessary.

   (c) Click *Create Image*

10. **Program the QPSI Flash Boot Image**

    (a) If Vivado is not running, start Vivado, and select *Open Hardware Manager* from within the main GUI.
    If Vivado has the `blinky_gpio_mio` project open, then in the *Flow Navigator*, under *PROGRAM AND DEBUG*, select *Open Hardware Manager*

    (b) Use the hardware manager to connect to the target (the MiniZed board).

    (c) From the Vivado main menu, select *Tools→Add Configuration Memory Device*.
    Note that this option becomes disabled (the menu option becomes greyed out) after you have added a configuration memory device.

    (d) In the *Add Configuration Memory* dialog:
    - Enter the **Filter** settings:
      - *Manufacturer:* `Micron`
      - *Density (Mb):* `128`
      - *Type:* `qspi`
      - *Width:* `x1-single`
    - From the **Select Configuration Memory Part** list, select the first device: `my25ql128-qspi-x1-single`
    - Click *Ok*

    (e) In the *Program Configuration Memory Device* window:
    - For **Configuration file:** navigate to the `BOOT.bin` file in the `blinky_gpio_mio` application folder.
    - For **Zynq FSBL:** navigate to the `zynq_fsbl.elf` file in the `zynq_fsbl` application folder.
      Figure 11 shows the configuration window after the files have been selected.
    - Click *Ok* to program the QSPI Flash

    (f) Unplug the MiniZed, change the boot mode switch to Flash, and plug it back into the development machine.

The MiniZed will boot from flash and the bicolor LED will start cycling off, green, red, and amber. The blue LED will remain off, as the PS bicolor LED project does not use the PL, so the configuration DONE signal never asserts.

Congratulations! Your MiniZed board boots and runs your first custom application.

The next design adds the progammable logic into the mix!

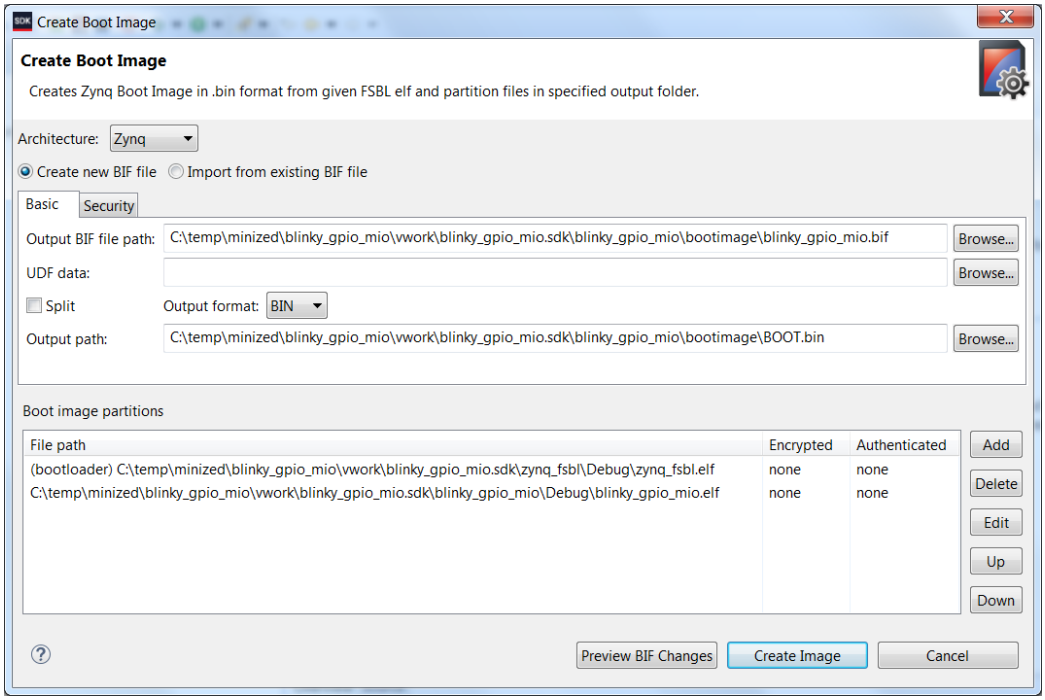


Figure 10: PS GPIO MIO design Create Boot Image window.

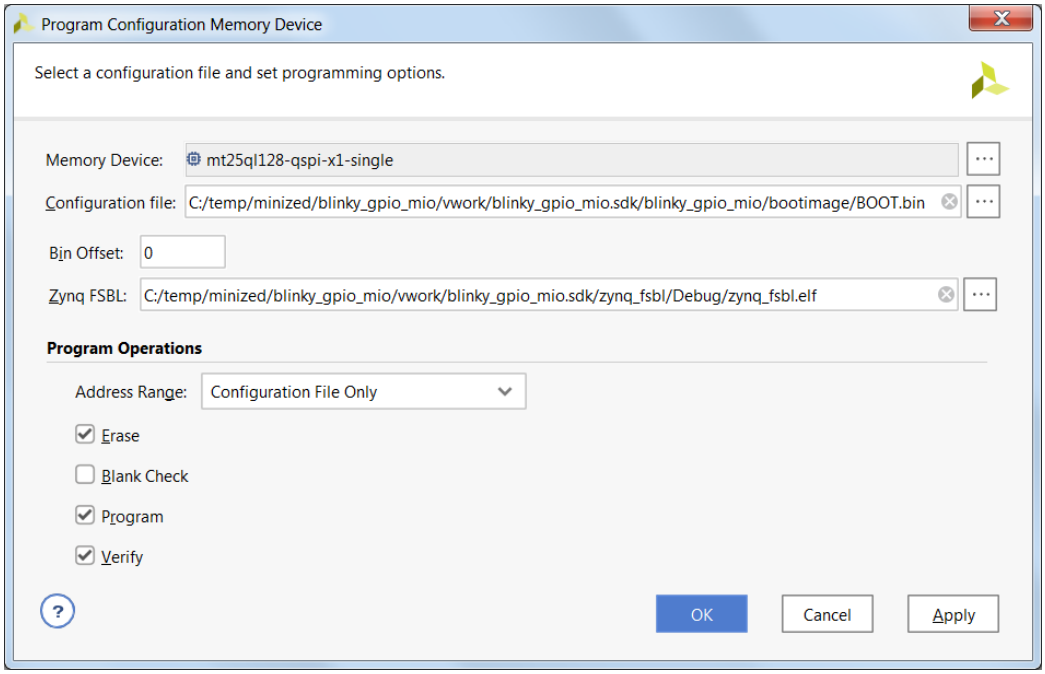


Figure 11: PS GPIO MIO design Program Coniguration Memory Device window.

4.3 PL bicolor and blue LED control via GPIO EMIO

The PL bicolor and blue LED control design extends the GPIO MIO design:

- 16-bits of PS GPIO EMIO are connected to the PL.
- The 7-series `STARTUPE2` component is instantiated in the PL to control the blue LED.
- The PL is used to connect GPIO EMIO outputs to the bicolor and blue LED.
- The PL is used to connect GPIO EMIO inputs to the user push button and switch.
- The Vivado design uses a top-level file that is *not* managed by Vivado.
- The Vivado block design is instantiated in the top-level design.
- An unmanaged constraints Tcl file is used to implement pin constraints.

TODO

- I have already extended the GPIO MIO design and confirmed that EMIO can control the PL bicolor LED.
- The post-block-automation Zynq processor has 16-bits of EMIO support. All I needed to do was to select the GPIO_0 port, right-mouse-click and select *Make External* and a port named `GPIO_0_0` is created. I renamed it `GPIO`.
- Extend the design to support the blue LED, the user switch, and push button.
- EMIO registers: - DATA_2 at E000A048 Output Data, GPIO Bank 2, EMIO - DATA_2_RO at E000A068 Input Data, GPIO Bank 2, EMIO - DIRM_2 at E000A284 Direction Mode, GPIO Bank 2, EMIO - OEN_2 at E000A288 Output Enable, GPIO Bank 2, EMIO
- Describe how to use U-Boot to control the EMIO LED?

Tutorial walk-through notes

Follow through the first tutorial and just make notes about the changes.

- Vivado: Create a new project `blinky_gpio_emio`
- Zynq PS: After running block automation, select GPIO_0, *Make External*, and rename to `GPIO`
- At this point we have a top-level design that we do not want to generate a wrapper for, rather we want a top-level file in which we can drop the `system` design as a component.
- Figure XXX shows a top-level HDL component created based on the port definitions on `vwork\blinky_gpio_emio.srcs\sources_1\bd\system\synth\system.v`
- Add `minized.sv` to the project
- Add the pin constraints Tcl file to the project and set it to be used for *Implementation* only.
- Run Synthesis. No issues, so Run Implementation. No issues, so Generate Bitstream.
- Check the arduino pins. A3 should be pin E13 and A4 should be pin E12.

```
| E11        | arduino_a[5]      | High Range | IO_L2P_T0_AD8P_35       | BIDIR      | LVCMOS3
| E12        | arduino_a[4]      | High Range | IO_L2N_T0_AD8N_35       | BIDIR      | LVCMOS3
| E13        | arduino_a[3]      | High Range | IO_L1N_T0_AD0N_35       | BIDIR      | LVCMOS3
| F12        | arduino_a[2]      | High Range | IO_L1P_T0_AD0P_35       | BIDIR      | LVCMOS3
| F13        | arduino_a[1]      | High Range | IO_L3P_T0_DQS_AD1P_35   | BIDIR      | LVCMOS3
| F14        | arduino_a[0]      | High Range | IO_L3N_T0_DQS_AD1N_35   | BIDIR      | LVCMOS3
```

- Export hardware and include the bitstream.
- Launch SDK, setup the BSP, and the application.
- Since this design has a bitstream, the FPGA needs to be programmed first, and then the application downloaded.
- Changing the LEDs using the EMIO should just need DIRM = 1 at offset 284. Yes. That and the data output were all that were needed. Ok, now I have enough to write up this tutorial step.

PL bi-color LED on pin E12 PL_LED_R, E13 and PL_LED_G. Not sure about the names, since they're also on Arduino pins.

4.4 PL bicolor and blue LED control via AXI GPIO

Create a Zynq processor system with;

- PS bicolor LED control
- PL bicolor LED control from AXI GPIO
- PL blue DONE lED control from AXI GPIO
- PL switch readback via AXI GPIO
- PL push button readback via AXI GPIO
- Do not use the Vivado block design wrapper file.
- Describe the use of unmanaged constraints.

4.5 Scripting the Blinky LED Designs

5 Programmable Logic (PL) Only Designs

Programmable Logic Only.

Blink an LED from the internal oscillator, the external oscillator (none), and an PS oscillator.

Select which oscillator using the select switch?

Or just blink the blue LED using the internal oscillator clock and the bi-color LED using the PS clock.

If the QSPI flash is erased, or the PS design does not enable the FCLK, then the counter clocked by the PS does not increment.

Describe the use of unmanaged constraints.

5.1 Using the FPGA internal oscillator

5.2 Using the PS-to-PL clocks

6 Peripheral Designs

6.1 QSPI Flash

Create an application that uses the QSPI PS library to access the flash. The application should support all the flash features via the UART or WiFi interface. Write a PC-side application that allows you to read and write the contents of flash. Use this to read the contents of a MiniZed that has never been programmed, and one restored to factory defaults.

Use the Xilinx QSPI code to read the flash and calculate the CRC32 using the same code I used under Cygwin. This would demonstrate a pretty basic use of the library.

Extend this application to access the eMMC too?

6.2 I2C to PMIC and Sensors

6.3 SPI to PMod ADC and DAC

A Factory Restore

The Avnet guide *Restoring MiniZed to the Factory State* [6] describes how to restore the MiniZed to factory state. This section shows how to view the version information of U-Boot and Linux on a MiniZed, how to calculate the checksums of the images in on-board non-volatile memory, and how to compare those checksums to the recovery or user images.

A.1 Viewing Tool and Software Versions

To view the MiniZed software versions, start by setting the boot mode switch to Flash, connecting the board to your development machine using the USB JTAG/UART interface (see Figure 2), and configure a serial terminal for communication at 115200 baud 8N1. Hit enter and the MiniZed console output (for a device purchased in March 2019) was

```
PetaLinux 2017.4 MiniZed /dev/ttyPS0

MiniZed login:
```

Press the blue reset button (by the PMod connectors on the bottom right of Figure 2), press the enter key to stop U-Boot from loading Linux, and enter the `version` command to get details on the U-Boot version. The console output was

```
U-Boot 2017.01 (Mar 22 2018 - 23:33:56 -0700)

Board: Xilinx Zynq
DRAM:  ECC disabled 512 MiB
MMC:   Card did not respond to voltage select!
sdhci@e0100000 - probe failed: -95
sdhci_transfer_data: Error detected in status(0x208000)!
Card did not respond to voltage select!

SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
U-BOOT for MiniZed

Hit any key to stop autoboot:  0
Zynq>  version

U-Boot 2017.01 (Mar 22 2018 - 23:33:56 -0700)
arm-xilinx-linux-gnueabi-gcc (Linaro GCC 6.2-2016.11) 6.2.1 20161016
GNU ld (GNU Binutils) 2.27.0.20160806
Zynq>
```

Note that the PetaLinux prompt and U-Boot messages have different version strings. The PetaLinux prompt implies that Xilinx Vivado and PetaLinux tools version 2017.4 were used to build PetaLinux, while the U-Boot version string implies 2017.1 tools were used to build U-Boot.

Type `?` and then press the enter key to see the U-Boot help menu. U-Boot can be used to read and write memory locations, read and program flash, boot standalone programs, real-time operating systems, and Linux. The example designs in this tutorial demonstrate some of these U-Boot features.

Boot Linux using the `boot` command. The inital lines of the console output were

```
Zynq> boot
boot Petalinux
reading image.ub
43267036 bytes read in 14650 ms (2.8 MiB/s)
## Loading kernel from FIT Image at 10000000 ...
   Using 'conf@1' configuration
   Trying 'kernel@0' kernel subimage
     Description:  Linux Kernel
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x100000d4
     Data Size:    3925272 Bytes = 3.7 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x00008000
     Entry Point:  0x00008000
     Hash algo:    sha1
     Hash value:   f2ac69bcdcf755b54af38b5fd870f843fb0e9bc7
   Verifying Hash Integrity ... sha1+ OK
## Loading ramdisk from FIT Image at 10000000 ...
   Using 'conf@1' configuration
   Trying 'ramdisk@0' ramdisk subimage
     Description:  ramdisk
     Type:         RAMDisk Image
     Compression:  uncompressed
     Data Start:   0x103c28d8
     Data Size:    39323409 Bytes = 37.5 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: unavailable
     Entry Point:  unavailable
     Hash algo:    sha1
     Hash value:   d57b450521fd302afdc95bfe2e7c521b08745d10
   Verifying Hash Integrity ... sha1+ OK
## Loading fdt from FIT Image at 10000000 ...
   Using 'conf@1' configuration
   Trying 'fdt@0' fdt subimage
     Description:  Flattened Device Tree blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x103be6e0
     Data Size:    16712 Bytes = 16.3 KiB
     Architecture: ARM
     Hash algo:    sha1
     Hash value:   bd55217bdcb35db08dcfd467cfbbf0dd5e4cb754
```

The hash values for the kernel, RAM disk, and device tree can be compared to the versions in the restored image to confirm they are the same. Save a copy of the Linux boot messages, as they maybe useful later when building PetaLinux from scratch.

Login at the Linux prompt as user `root` with password `root` and type a few standard Linux commands to determine some detail about the Linux and hardware configuration;

```
root@MiniZed:~# uname -a
Linux MiniZed 4.9.0-xilinx-v2017.4 #1 SMP PREEMPT Thu Mar 22 22:22:16 MST 2018
armv7l GNU/Linux

root@MiniZed:~# mount
rootfs on / type rootfs (rw,size=227296k,nr_inodes=56824)
proc on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=227296k,nr_inodes=56824,mode=755)
tmpfs on /run type tmpfs (rw,nosuid,nodev,mode=755)
tmpfs on /var/volatile type tmpfs (rw,relatime)
/dev/mmcblk1p1 on /run/media/mmcblk1p1 type vfat (rw,relatime,gid=6,fmask=0007,
dmask=0007,allow_utime=0020,codepage=437,iocharset=iso8859-1,shortname=mixed,
errors=remount-ro)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=000)
/dev/mmcblk1p1 on /mnt/emmc type vfat (rw,relatime,gid=6,fmask=0007,dmask=0007,
allow_utime=0020,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)

root@MiniZed:~# df
Filesystem           1K-blocks      Used Available Use% Mounted on
devtmpfs                227296         4    227292   0% /dev
tmpfs                   255204        84    255120   0% /run
tmpfs                   255204        44    255160   0% /var/volatile
/dev/mmcblk1p1          123089     58385     64704  47% /run/media/mmcblk1p1
/dev/mmcblk1p1          123089     58385     64704  47% /mnt/emmc

root@MiniZed:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
BogoMIPS        : 666.66
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

Hardware        : Xilinx Zynq Platform
Revision        : 0003
Serial          : 0000000000000000

root@MiniZed:~# cat /proc/version
Linux version 4.9.0-xilinx-v2017.4 (sroussea@xterra1) (gcc version 6.2.1 20161016
(Linaro GCC 6.2-2016.11) ) #1 SMP PREEMPT Thu Mar 22 22:22:16 MST 2018

root@MiniZed:~# poweroff
```

Per the recommendation on page 30 of the *Getting Started Guide* [5], use the `poweroff` command before powering off the MiniZed to avoid corruption of the eMMC flash.

The QSPI flash also contains a fallback Linux image. This image can be used when the eMMC flash has been erased. The fallback image can be booted from U-Boot as follows:

```
Zynq> run boot_qspi
Booting backup kernel from QSPI..
SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
device 0 offset 0x270000, size 0xd80000
SF: 14155776 bytes @ 0x270000 Read: OK
## Loading kernel from FIT Image at 10000000 ...
   Using 'conf@1' configuration
   Trying 'kernel@0' kernel subimage
     Description:  Linux Kernel
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x100000d4
     Data Size:    3925320 Bytes = 3.7 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x00008000
     Entry Point:  0x00008000
     Hash algo:    sha1
     Hash value:   8d758a5b6f7da43eecf5324a02ff10a78e139113
   Verifying Hash Integrity ... sha1+ OK
## Loading ramdisk from FIT Image at 10000000 ...
   Using 'conf@1' configuration
   Trying 'ramdisk@0' ramdisk subimage
     Description:  ramdisk
     Type:         RAMDisk Image
     Compression:  uncompressed
     Data Start:   0x103c2908
     Data Size:    10018537 Bytes = 9.6 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: unavailable
     Entry Point:  unavailable
     Hash algo:    sha1
     Hash value:   2f1d8fc5a9ce9dfbdf1ea753a23dcf7f0d881d55
   Verifying Hash Integrity ... sha1+ OK
## Loading fdt from FIT Image at 10000000 ...
   Using 'conf@1' configuration
   Trying 'fdt@0' fdt subimage
     Description:  Flattened Device Tree blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x103be710
     Data Size:    16712 Bytes = 16.3 KiB
     Architecture: ARM
     Hash algo:    sha1
     Hash value:   bd55217bdcb35db08dcfd467cfbbf0dd5e4cb754
```

The hash values of the kernel, RAM disk, and device tree are different than the versions loaded from the eMMC.

A.2 Restore Image Details

The MiniZed contains three non-volatile memory devices;

- 2kbit EEPROM
- 128Mbit QSPI Flash
- 8GB eMMC Flash

The 2kbit EEPROM contains the configuration for the FTDI FT2232H device. The Avnet restore procedure does not include restoring the EEPROM contents. The EEPROM contents can be read and saved using the FT_PROG tool from FTDI. All MiniZed boards ship with the same USB JTAG/UART serial number: `1234-oj1`. Connecting multiple MiniZed boards to the same development machine at the same time requires the use of unique USB serial numbers, as under Windows, each USB serial number defines a unique COM port number. The FTDI FT_PROG tool can be used to change the MiniZed FTDI FT2232H serial number. Do not change any other FT2232H parameters, otherwise the Digilent JTAG device will not be detected by the Xilinx tools as a JTAG programmer.

The Avnet MiniZed documentation page has a link to a zip file containing the restore images [6]. The zip file contains the following files;

```
Restoring MiniZed to Factory Status_03
+-- Programming files
|   +-- Micron QSPI Flash
|       +-- flash_fallback_7007S.bin
|       +-- flash_only_boot_7007S.bin
|       +-- zynq_fsbl.elf
+-- Restoring_MiniZed_to_Factory_Status_03.pdf
+-- USB memory stick files
    +-- image.ub
    +-- smallboot.bin
    +-- wpa_supplicant.conf
```

The MiniZed factory QSPI image is `flash_fallback_7007S.bin`, while the eMMC filesystem contains the files in `USB memory stick files`.

The images programmed in the MiniZed board QSPI flash and eMMC could be compared to the factory images using byte-by-byte comparisons, however, there is a simpler (and faster) solution: calculate and compare checksums. The MiniZed U-Boot image supports the `crc32` checksum (the U-Boot source provides support for additional checksum algorithms), and the MiniZed Linux image supports `sha1sum`, `sha256sum`, and `sha512sum`. Table 3 contains the `crc32` and `sha1sum` checksums for the restore images.

Table 3: MiniZed Factory Restore Image Checksums.

Image	Image Byte Length	Checksum	
		crc32	sha1
U-Boot			
flash_fallback_7007S.bin	0x00FC0BE4	0x005B7B3C	0x71511941C14604A88BDD57B0C7762444D1538B5B
flash_only_boot_7007S.bin	0x00FDA5F4	0x86771B07	0x191BE50A37CF5E7D453A67AE510264C1F2CDF89B
smallboot.bin	0x00FC0BE4	0x005B7B3C	0x71511941C14604A88BDD57B0C7762444D1538B5B
Linux			
image.ub	0x00FC0BE4	0x005B7B3C	0xD6182171821848 62C69AB1559EF806665B7D0E83

A.2.1 QSPI Image Details

The image in MiniZed QSPI flash can be viewed by copying the contents of the 16MB flash to DDR memory, and then reading the image from DDR memory. The U-Boot commands to detect the QSPI flash and copy 16MB (0x1000000) to DDR address 0x10000000 are

```
Zynq> sf probe
SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> sf read 10000000 0 1000000
device 0 whole chip
SF: 16777216 bytes @ 0x0 Read: OK
```

The QSPI image header can be displayed by reading the DDR memory

```
Zynq> md 10000000 30
10000000: eafffffe eafffffe eafffffe eafffffe    ................
10000010: eafffffe eafffffe eafffffe eafffffe    ................
10000020: aa995566 584c4e58 00000000 01010000    fU..XNLX........
10000030: 00001700 00018008 00000000 00000000    ................
10000040: 00018008 00000001 fc164530 00000000    ........0E......
10000050: 00000000 00000000 00000000 00000000    ................
10000060: 00000000 00000000 00000000 00000000    ................
10000070: 00000000 00000000 00000000 00000000    ................
10000080: 00000000 00000000 00000000 00000000    ................
10000090: 00000000 00000000 000008c0 00000c80    ................
100000a0: ffffffff 00000000 ffffffff 00000000    ................
100000b0: ffffffff 00000000 ffffffff 00000000    ................
```

The BootROM Image Header format is defined in Table 6-5 of the Zynq TRM (p171 [10]).

The U-Boot crc32 checksum for the QSPI image copied to DDR memory is

```
Zynq> crc32 10000000 fc0be4
crc32 for 10000000 ... 10fc0be3 ==> 005b7b3c
```

The checksum matches that for `flash_fallback_7007S.bin` in Table 3.

A.2.2 eMMC Image Details

The eMMC images can be viewed from the MiniZed Linux console using:

```
root@MiniZed:~# ls -al /mnt/emmc/
total 58385
drwxrwx---    2 root     disk           512 Jan  1  1970 .
drwxr-xr-x    4 root     root            80 Mar 23 06:07 ..
-rwxrwx---    1 root     disk      43267036 Jul 28  2017 image.ub
-rwxrwx---    1 root     disk      16518116 Jul 28  2017 smallboot.bin
-rwxrwx---    1 root     disk           177 Jul 28  2017 wpa_supplicant.conf

root@MiniZed:~# od -Ax -tx4 -v -N 128 /mnt/emmc/image.ub
000000 edfe0dd0 dc339402 38000000 f4319402
000010 28000000 11000000 10000000 00000000
000020 74000000 bc319402 00000000 00000000
000030 00000000 00000000 01000000 00000000
000040 03000000 04000000 64000000 3c9ab45a
000050 03000000 24000000 00000000 6f422d55
000060 6620746f 6d497469 20656761 20726f66
000070 786e6c70 6d72615f 72656b20 006c656e

root@MiniZed:~# od -Ax -tx4 -v -N 192 /mnt/emmc/smallboot.bin
000000 eaffffe eaffffe eaffffe eaffffe
000010 eaffffe eaffffe eaffffe eaffffe
000020 aa995566 584c4e58 00000000 01010000
000030 00001700 00018008 00000000 00000000
000040 00018008 00000001 fc164530 00000000
000050 00000000 00000000 00000000 00000000
000060 00000000 00000000 00000000 00000000
000070 00000000 00000000 00000000 00000000
000080 00000000 00000000 00000000 00000000
000090 00000000 00000000 000008c0 00000c80
0000a0 ffffffff 00000000 ffffffff 00000000
0000b0 ffffffff 00000000 ffffffff 00000000
```

The image checksums can be calculated via

```
root@MiniZed:~# sha1sum /mnt/emmc/image.ub
d618217182184862c69ab1559ef806665b7d0e83  /mnt/emmc/image.ub

root@MiniZed:~# sha1sum /mnt/emmc/smallboot.bin
71511941c14604a88bdd57b0c7762444d1538b5b  /mnt/emmc/smallboot.bin
```

The checksums match those in Table 3. The `wpa_supplicant.conf` file on the MiniZed was different than the restore image, however, the user has to edit that file to match their setup, so that difference was expected.

A.2.3 U-Boot Image Versions and Checksums

This section contains the console output for U-Boot power-on-reset, `version`, and `crc32` commands for the three Avnet restore images.

Image: flash_fallback_7007S.bin

```
U-Boot 2017.01 (Mar 22 2018 - 23:33:56 -0700)

Board: Xilinx Zynq
DRAM:  ECC disabled 512 MiB
MMC:   Card did not respond to voltage select!
sdhci@e0100000 - probe failed: -95
sdhci_transfer_data: Error detected in status(0x208000)!
Card did not respond to voltage select!

SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
U-BOOT for MiniZed

Hit any key to stop autoboot:  0
Zynq> version

U-Boot 2017.01 (Mar 22 2018 - 23:33:56 -0700)
arm-xilinx-linux-gnueabi-gcc (Linaro GCC 6.2-2016.11) 6.2.1 20161016
GNU ld (GNU Binutils) 2.27.0.20160806

Zynq> sf probe
SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> sf read 10000000 0 1000000
device 0 whole chip
SF: 16777216 bytes @ 0x0 Read: OK
Zynq> crc32 10000000 fc0be4
crc32 for 10000000 ... 10fc0be3 ==> 005b7b3c
```

Image: flash_only_boot_7007S.bin

```
U-Boot 2016.07 (Jul 27 2017 - 21:56:59 -0700)

DRAM:  ECC disabled 512 MiB
MMC:   sdhci@e0100000: 0, sdhci@e0101000: 1
SF: Detected N25Q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
U-BOOT for
```

```
Hit any key to stop autoboot:  0
Zynq> version

U-Boot 2016.07 (Jul 27 2017 - 21:56:59 -0700)
arm-linux-gnueabihf-gcc (Linaro GCC 5.2-2015.11-2) 5.2.1 20151005
GNU ld (Linaro_Binutils-) 2.25.0 Linaro 2016_02
Zynq> sf probe
SF: Detected N25Q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> sf read 10000000 0 1000000
device 0 whole chip
SF: 16777216 bytes @ 0x0 Read: OK
Zynq> crc32 10000000 fda5f4
crc32 for 10000000 ... 10fda5f3 ==> 86771b07
```

Image: smallboot.bin

```
U-Boot 2017.01 (Mar 22 2018 - 23:33:56 -0700)

Board: Xilinx Zynq
DRAM:  ECC disabled 512 MiB
MMC:   Card did not respond to voltage select!
sdhci@e0100000 - probe failed: -95
sdhci_transfer_data: Error detected in status(0x208000)!
Card did not respond to voltage select!

SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
U-BOOT for MiniZed

Hit any key to stop autoboot:  0
Zynq> version

U-Boot 2017.01 (Mar 22 2018 - 23:33:56 -0700)
arm-xilinx-linux-gnueabi-gcc (Linaro GCC 6.2-2016.11) 6.2.1 20161016
GNU ld (GNU Binutils) 2.27.0.20160806

Zynq> sf probe
SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> sf read 10000000 0 1000000
device 0 whole chip
SF: 16777216 bytes @ 0x0 Read: OK
Zynq> crc32 10000000 fc0be4
crc32 for 10000000 ... 10fc0be3 ==> 005b7b3c
```

The console messages from smallboot.bin are identical to flash_fallback_7007S.bin. The easiest way to determine which image is programmed, is to calculate the `crc32` checksum using the image length and expected values in Table 3.

A.3 QSPI Flash Programming

The Avnet MiniZed factory restore document [6] shows how to program the QSPI flash using the Xilinx Software Command-line Tool (XSCT) and the Vivado GUI. The restore images were copied to the location `c:\temp\minized` prior to the operations in the next section.

A.3.1 XSCT `program_flash`

XSCT can be started under Windows by selecting the tool from *Xilinx Design Tools→SDK 2019.1→Xilinx Software Command Line Tool*. Once the tool starts, change directory to the folder containing the restore images and issue the `program_flash` command using the Tcl exec command, eg.,

```
xsct% cd {c:\temp\minized}
xsct% exec program_flash -f flash_only_boot_7007S.bin -fsbl zynq_fsbl.elf
  -flash_type qspi_single
```

Running the `program_flash` command using the Tcl `exec` command supresses the command console output until the flash programming completes. More insight into the flash programming is gained if the tool is run directly, so that the console output is seen while the flash is programmed.

The location of the `program_flash` tool can be determined using

```
xsct% exec which program_flash
C:\software\Xilinx\SDK\2019.1\bin\program_flash
```

The MiniZed flash was erased using the U-Boot command

```
Zynq> sf probe
SF: Detected N25Q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> sf erase 0 1000000
SF: 16777216 bytes @ 0x0 Erased: OK
```

and the Flash/JTAG switch changed to the JTAG position. The flash programming tool was then run from a Cygwin console using the full path to the application via

```
$ cd c:/temp/minized
$ C:/software/Xilinx/SDK/2019.1/bin/program_flash -f flash_only_boot_7007S.bin
  -fsbl zynq_fsbl.elf -flash_type qspi_single

****** Xilinx Program Flash
****** Program Flash v2019.1 (64-bit)
  **** SW Build 2552052 on Fri May 24 14:49:42 MDT 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

Connected to hw_server @ TCP:localhost:3121
Available targets and devices:
Target 0 : jsn-MiniZed V1-1234-oj1A
        Device 0: jsn-MiniZed V1-1234-oj1A-4ba00477-0

Retrieving Flash info...

Initialization done, programming the memory
===== mrd->addr=0xF800025C, data=0x00000000 =====
BOOT_MODE REG = 0x00000000
Downloading FSBL...
```

```
Running FSBL...
Finished running FSBL.
===== mrd->addr=0xF8000110, data=0x000FA220 =====
READ: ARM_PLL_CFG (0xF8000110) = 0x000FA220
===== mrd->addr=0xF8000100, data=0x00028008 =====
READ: ARM_PLL_CTRL (0xF8000100) = 0x00028008
===== mrd->addr=0xF8000120, data=0x1F000200 =====
READ: ARM_CLK_CTRL (0xF8000120) = 0x1F000200
===== mrd->addr=0xF8000118, data=0x00113220 =====
READ: IO_PLL_CFG (0xF8000118) = 0x00113220
===== mrd->addr=0xF8000108, data=0x00024008 =====
READ: IO_PLL_CTRL (0xF8000108) = 0x00024008
Info:  Remapping 256KB of on-chip-memory RAM memory to 0xFFFC0000.
===== mrd->addr=0xF8000008, data=0x00000000 =====
===== mwr->addr=0xF8000008, data=0x0000DF0D =====
MASKWRITE: addr=0xF8000008, mask=0x0000FFFF, newData=0x0000DF0D
===== mwr->addr=0xF8000910, data=0x000001FF =====
===== mrd->addr=0xF8000004, data=0x00000000 =====
===== mwr->addr=0xF8000004, data=0x0000767B =====
MASKWRITE: addr=0xF8000004, mask=0x0000FFFF, newData=0x0000767B

U-Boot 2019.01-07026-gae88108-dirty (Mar 22 2019 - 04:38:02 -0600)

Model: Zynq CSE QSPI Board
DRAM:  256 KiB
WARNING: Caches not enabled
In:    dcc
Out:   dcc
Err:   dcc
Zynq> sf probe 0 0 0
Warning: SPI speed fallback to 100 kHz
SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
Zynq> Sector size = 65536.
f probe 0 0 0
Performing Erase Operation...
sf erase 0 FE0000
SF: 16646144 bytes @ 0x0 Erased: OK
Zynq> Erase Operation successful.
INFO: [Xicom 50-44] Elapsed time = 1 sec.
Performing Program Operation...
0%...sf write FFFC0000 0 20000
device 0 offset 0x0, size 0x20000
SF: 131072 bytes @ 0x0 Written: OK
Zynq> sf write FFFC0000 20000 20000
device 0 offset 0x20000, size 0x20000
SF: 131072 bytes @ 0x20000 Written: OK

... [more sector programming messages] ...

Zynq> sf write FFFC0000 FA0000 20000
device 0 offset 0xfa0000, size 0x20000
```

```
SF: 131072 bytes @ 0xfa0000 Written: OK
Zynq> 100%
sf write FFFC0000 FC0000 1A5F4
device 0 offset 0xfc0000, size 0x1a5f4
SF: 108020 bytes @ 0xfc0000 Written: OK
Zynq> Program Operation successful.
INFO: [Xicom 50-44] Elapsed time = 67 sec.

Flash Operation Successful
```

The output from `program_flash` shows that it downloads the FSBL, a U-Boot image, and then uses U-Boot commands to program the flash. Interesting! The U-Boot flash programming occurs much faster than when using the UART, as the U-Boot image used by `program_flash` uses the ARM JTAG Debug Communications Channel (DCC) to transfer sectors to U-Boot for programming to flash: the initial U-Boot console message shows the use of `dcc` for stdin, stdout and stderr.

A.3.2 Vivado Flash Programming

The Avnet MiniZed factory restore document [6] shows how to use the *Vivado Hardware Manager* to program the QSPI flash. When I first followed the procedure using the flash images located in the folder generated by unzipping the restore images zip file, Vivado would pop-up an error dialog with the message `[Labtools 27-3161] Flash Programming Unsuccessful`. The Vivado console did not provide any insight into what might be wrong. Flash programming works if the restore files are first copied to `c:/temp/minized`.

The Vivado Tcl console output for programming of the image `flash_fallback_7007S.bin` was

```
program_hw_cfgmem -hw_cfgmem [ get_property PROGRAM.HW_CFGMEM
  [lindex [get_hw_devices xc7z007s_1] 0]]

Performing Erase Operation...
Erase Operation successful.
INFO: [Xicom 50-44] Elapsed time = 28 sec.

Performing Program Operation...
Program Operation successful.
INFO: [Xicom 50-44] Elapsed time = 68 sec.

Performing Verify Operation...
INFO: [Xicom 50-44] Elapsed time = 91 sec.
Verify Operation successful.

INFO: [Labtoolstcl 44-377] Flash programming completed successfully
```

After programming, the Flash/JTAG boot mode switch was switched to Flash mode, U-Boot booted, and the procedure in Section A.2.3 used to confirm the U-Boot version and checksum.

A.4 Restore Procedure

The Avnet MiniZed factory restore procedure is [6]:

1. Power-on the MiniZed and program the QSPI flash with the flash-only image.
2. Power-on the MiniZed with the additional USB power source connected and a USB stick containing the files to program to the eMMC, boot from the flash-only image, and run the scripts that program the eMMC.
3. Program the QSPI flash with the fallback image.

To ensure the restore procedure works consistently, the MiniZed eMMC and QSPI flash can first be erased. The following steps were performed on a MiniZed board programmed with the QSPI U-Boot image `flash_fallback_7007S.bin`.

1. **Erase eMMC Flash**

   The fallback Linux image was booted from U-Boot using the command `run boot_qspi`. The eMMC partitions are then erased using the `fdisk` utility via

```
root@MiniZed:~# fdisk /dev/mmcblk1

The number of cylinders for this disk is set to 232448.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): p

Disk /dev/mmcblk1: 7616 MB, 7616856064 bytes
4 heads, 16 sectors/track, 232448 cylinders
Units = cylinders of 64 * 512 = 32768 bytes

        Device Boot      Start         End      Blocks  Id System
/dev/mmcblk1p1               1        3907      125016  83 Linux

Command (m for help): d
Selected partition 1

Command (m for help): p

Disk /dev/mmcblk1: 7616 MB, 7616856064 bytes
4 heads, 16 sectors/track, 232448 cylinders
Units = cylinders of 64 * 512 = 32768 bytes

        Device Boot      Start         End      Blocks  Id System

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table
fdisk: WARNING: rereading partition table failed, kernel still uses old table:
```

```
Device or resource busy
root@MiniZed:~# reboot
```

In this example console output, the `p` command was used to print the partitions (there was only one), and the `d` command was used to delete the partition.

After reboot, halt the U-Boot boot sequence, start Linux using `run boot_qspi`, and use `fdisk` to confirm there are no partitions on the eMMC. U-Boot can also be used to view the empty partition table via

```
Zynq> mmc dev 1
sdhci_transfer_data: Error detected in status(0x208000)!
switch to partitions #0, OK
mmc1(part 0) is current device
Zynq> mmc part

Partition Map for MMC device 1  --   Partition Type: DOS

Part    Start Sector    Num Sectors     UUID            Type
Zynq>
```

2. **Erase QSPI Flash**

   The U-Boot commands to erase QSPI flash are

   ```
   Zynq> sf probe
   SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
   Zynq> sf erase 0 1000000
   SF: 16777216 bytes @ 0x0 Erased: OK
   ```

   Power-down the board (remove the USB connections), and change the Flash/JTAG boot mode switch to JTAG mode.

3. **Program QSPI Flash with `flash_only_boot_7007S.bin`**

   Use one of the flash programming procedures outlined in Section A.3 to program the image. After programming, power-down the board, and change the Flash/JTAG boot mode switch to Flash mode.

4. **Prepare a USB flash drive**

   Format a USB flash drive in FAT or FAT32 format, and copy the restore image files from within the zip file:

   ```
   +-- USB memory stick files
       +-- image.ub
       +-- smallboot.bin
       +-- wpa_supplicant.conf
   ```

   onto the USB drive. Edit `wpa_supplicant.conf` to match your local WiFi network settings.

5. **Power the MiniZed with two USB cables**

   The MiniZed USB host-mode interface is used to access the USB flash drive. The power for the USB host-mode interface is provided by the second micro-USB connector identified with the label *Auxiliary Power* in Figure 2.

Insert the USB flash drive in the MiniZed USB host-mode connector, and connect both micro-USB connectors to your development host. The flash-only boot image contains Linux. The Linux version information is

```
PetaLinux 2016.4 plnx_arm /dev/ttyPS0

plnx_arm login: root
Password:
root@plnx_arm:~# uname -a
Linux plnx_arm 4.6.0-xilinx #1 SMP PREEMPT Thu Jul 27 22:24:51 PDT 2017
armv7l GNU/Linux
```

6. **Restore MiniZed Linux**

   The Linux commands used to restore the MiniZed eMMC Linux installation to factory state are implemented in the script `onetest.sh` (p16 [6]). The location and contents of the script can be viewed via

   ```
   root@plnx_arm:~# which onetest.sh
   /usr/local/bin/onetest.sh
   root@plnx_arm:~# cat /usr/local/bin/onetest.sh
   ```

   The script creates a DOS partition on the eMMC, formats the partition to use a FAT32 filesystem, mounts the eMMC filesystem, mounts the USB flash stick, copies the files from the USB stick to eMMC, programs the QSPI flash with `smallboot.bin`, loads the WiFi driver, brings up the WiFi interface, runs the I2C and LED test application, and then shuts down the system.

   Run the script. Programming the QSPI flash via the `flashcp` command takes the most time. The I2C and LED test application will run until you press a key, and then the board will shutdown.

   Press the MiniZed reset button, interrupt the U-Boot boot process, and confirm the U-Boot checksum is that of `smallboot.bin`.

7. **Program QSPI Flash with `flash_fallback_7007S.bin`**

   The Avnet restore procedure lists this as an optional step (p21 [6]). Use one of the flash programming procedures outlined in Section A.3 to program the fallback image.

8. **MiniZed Tests**

   The Avnet restore procedure has instructions for testing the MiniZed

   - Bluetooth (p18 [6])
   - The Power Management controller (p19 [6])
   - Motion sensor and GPIO LEDs (p20 [6])

9. **Save the U-Boot Environment to QSPI Flash**

   If the flash sector containing the environment is blank, or the CRC is incorrect, the U-Boot power-on messages contain the warning

   ```
   *** Warning - bad CRC, using default environment
   ```

   This warning can be eliminated by saving the U-Boot environment.

   ```
   Zynq> saveenv
   Saving Environment to SPI Flash...
   SF: Detected n25q128 with page size 256 Bytes, erase size 64 KiB, total 16 MiB
   Erasing SPI flash...Writing to SPI flash...done
   ```

   The environment sector does not need to be erased, as U-Boot does that before writing to the SPI flash. The environment sector can be erased via

   ```
   Zynq> sf erase ff0000 10000
   SF: 65536 bytes @ 0xff0000 Erased: OK
   ```

   After erasing the environment section, press the reset button, halt the U-Boot boot sequence, and the CRC error message will be present in the U-Boot console messages.

B Resources

B.1 Avnet

- Avnet ZedBoard web site:
  - MiniZed product page
  - MiniZed documentation page
  - MiniZed reference designs page
- *MiniZed Schematic* [4]
- *MiniZed Quickstart Guide* [3]
- *MiniZed Getting Started Guide* [5]
- *MiniZed Hardware User Guide* [2]
- *Restoring MiniZed to the Factory State* [6]

B.2 Xilinx

- Xilinx Design Hubs
  - https://www.xilinx.com/support/documentation-navigation/design-hubs.html

  The *Zynq-7000 SoC Design* section links to multiple Zynq-7000 design hubs:
  - Design Overview
  - Data Movers
  - Power Management
  - Boot and Configuration
  - Security
  - Performance and Acceleration
  - Debug
- Zynq-7000 Technical Reference Manual [10]
- Zynq-7000 Software Developers Guide [11]

References

[1] ARM. DDI0406C: ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition (issue C.d), March 2018.
(DDI0406C_d_armv7ar_arm.pdf).

[2] Avnet. MiniZed Hardware User Guide (version 1.0), June 2017.
(MiniZed-HW-UG-v1-0-V1_0.pdf).

[3] Avnet. MiniZed Quick Start Card, September 2017.
(5239-QSC-XC7Z007S-1CLG225C-V7a.pdf).

[4] Avnet. MiniZed Zynq Development Board Schematic (revision 1), June 2017.
(MiniZed_Rev_1_Schematic.zip).

[5] Avnet. MiniZed Getting Started Guide (version 1.2), May 2018.
(MiniZed-GSG-v1_2.pdf).

[6] Avnet. Restoring MiniZed to the Factory State (version 3.0), April 2018.
(Restoring_MiniZed_to_Factory_Status_03.zip).

[7] Avnet. Tutorial 01: Build a Zynq Hardware Platform (version 2018.1), April 2018.
(01_MiniZed_Zynq_Intro_2018_1_01.zip).

[8] Avnet. Tutorial 02: First Application - Hello World (version 2018.1), April 2018.
(02_MiniZed_Zynq_Hello_World_2018_1_01.zip).

[9] Avnet. Tutorial 04: FSBL and Boot from QSPI (version 2018.1), April 2018.
(04_MiniZed_FSBL_Boot_2018_1_01.zip).

[10] Xilinx. UG585: Zynq-7000 SoC Technical Reference Manual (v1.12.2), July 2018.
(ug585-Zynq-7000-TRM.pdf).

[11] Xilinx. UG821: Zynq-7000 All Programmable SoC Software Developers Guide (v12.0), September 2018.
(ug821-zynq-7000-swdev.pdf).