Host Programming of QSPI Flash

XAPP1372 (v1.1) October 4, 2022

Summary

Versal® devices have a built-in hardware QSPI controller in the platform management controller (PMC). The QSPI controller is routed to the PMC MIO pins when connected to QSPI flash—this is commonly used as a boot device. The QSPI controller can be accessed by both the internal processing system (PS) and an external CPU. This application note provides a reference design for using an external CPU programming QSPI flash. The external CPU, later known as the host, communicates with the QSPI controller in the Versal device through the PCIe® bus, network on chip (NoC), and PMC in a sequential manner. The PS is not involved in this entire process. For customers with limited memory resources, this application does not require DDR memory. The programmable logic (PL) block RAM is sufficient.

Download the reference design files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design.

Features

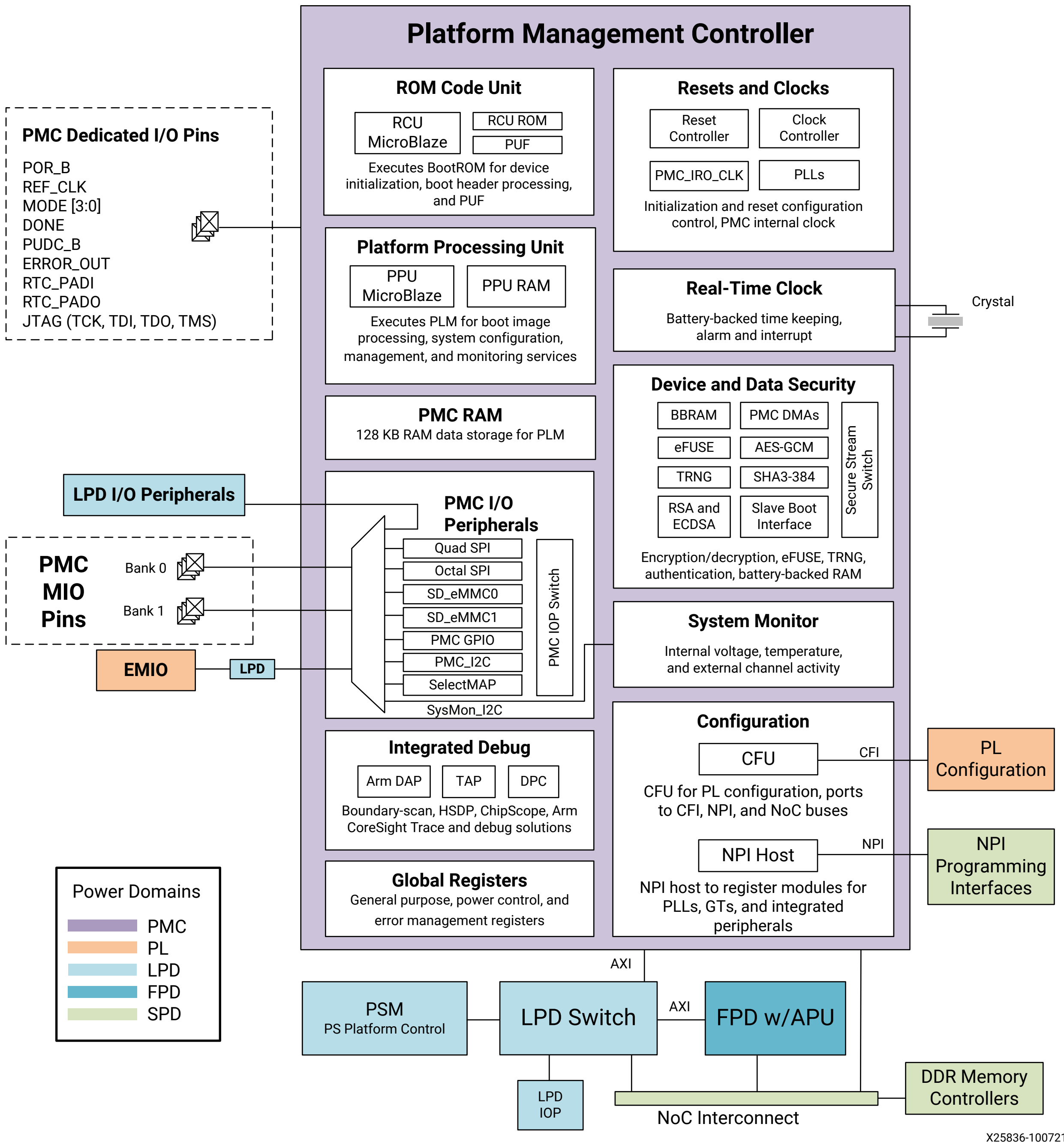
The reference design has the following features:

- Read QSPI flash ID
- Erase/write QSPI flash
- Verify QSPI flash
- All the host operations to QSPI are via PCIe. The hardware PCIe bus is configured as Gen2 x1 using the Integrated Block for PCI Express.
- No involvement of the Versal device PS
- QSPI flash DMA read support
- PL block RAMs replace DDR memory to store the verified data, thereby saving DDR space

Introduction

The QSPI controller is a hardened module inside the PMC, which is instantiated in the Vivado® Control, Interfaces, and Processing System (CIPS) IP. In the PMC functional block diagram, the QSPI controller is located at the PMC I/O peripherals, which includes the QSPI controller, OSPI controller, SD/EMMC controllers, I2C controller, and GPIO controller. The following figure shows that there is an NoC interconnect connected to the PMC through which the PL can access the QSPI controller.

Figure 1: **PMC Functional Block Diagram**

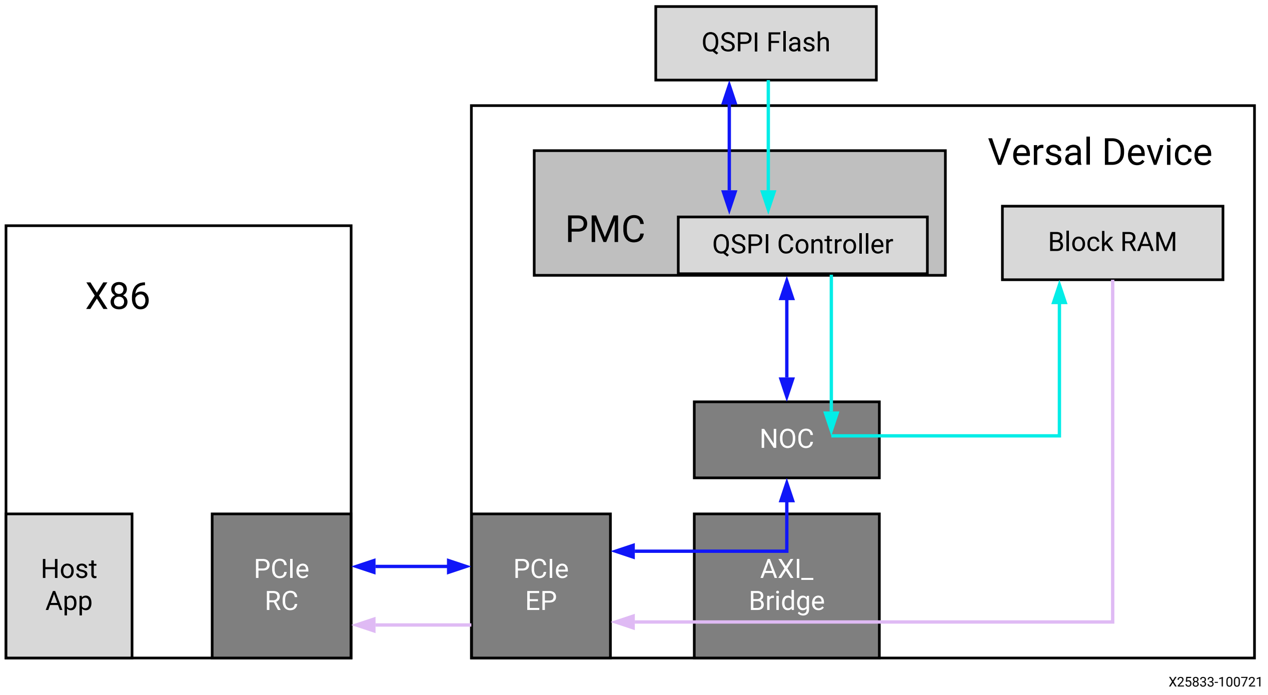

Platform Management Controller
ROM Code Unit
RCU MicroBlaze
RCU ROM
PUF
Executes BootROM for device initialization, boot header processing, and PUF
Resets and Clocks
Reset Controller
Clock Controller
PMC_IRO_CLK
PLLs
Initialization and reset configuration control, PMC internal clock
PMC Dedicated I/O Pins
POR_B
REF_CLK
MODE [3:0]
DONE
PUDC_B
ERROR_OUT
RTC_PADI
RTC_PADO
JTAG (TCK, TDI, TDO, TMS)
Platform Processing Unit
PPU MicroBlaze
PPU RAM
Executes PLM for boot image processing, system configuration, management, and monitoring services
Real-Time Clock
Battery-backed time keeping, alarm and interrupt
Crystal
PMC RAM
128 KB RAM data storage for PLM
Device and Data Security
BBRAM
PMC DMAs
eFUSE
AES-GCM
TRNG
SHA3-384
RSA and ECDSA
Slave Boot Interface
Secure Stream Switch
Encryption/decryption, eFUSE, TRNG, authentication, battery-backed RAM
LPD I/O Peripherals
PMC I/O Peripherals
Quad SPI
Octal SPI
SD_eMMC0
SD_eMMC1
PMC GPIO
PMC_I2C
SelectMAP
SysMon_I2C
PMC IOP Switch
PMC MIO Pins
Bank 0
Bank 1
EMIO
LPD
System Monitor
Internal voltage, temperature, and external channel activity
Integrated Debug
Arm DAP
TAP
DPC
Boundary-scan, HSDP, ChipScope, Arm CoreSight Trace and debug solutions
Configuration
CFU
CFI
PL Configuration
CFU for PL configuration, ports to CFI, NPI, and NoC buses
NPI Host
NPI
NPI Programming Interfaces
NPI host to register modules for PLLs, GTs, and integrated peripherals
Global Registers
General purpose, power control, and error management registers
Power Domains
PMC
PL
LPD
FPD
SPD
AXI
PSM
PS Platform Control
LPD Switch
AXI
FPD w/APU
LPD IOP
NoC Interconnect
DDR Memory Controllers
X25836-100721

Besides the Versal device internally accessing the QSPI controller path, on the other side, the host can access the Versal device NoC through the PCIe interface in the Versal device. Thus, the complete access path is as follows:

Host: PCIe RC > Versal device: PCIe EP (QDMA) > NoC > QSPI controller inside PMC > QSPI flash

The following figure shows the system architecture.

Figure 2: **System Architecture**



As shown in the figure, there are three types of datapath in the design:

- **Blue Path: Host Access QSPI Controller:** This path is used to read and write QSPI controller registers. Thus, there are two directions: one for write and another one for read.

  Both Read Flash ID and Erase/Write Flash features are completed via this datapath. In a Read Flash ID, the host issues a series of QSPI controller register write and read operations until the QSPI Flash ID is obtained.

  The Erase or Write QSPI Flash has a set of read and write operations. The main difference between Erase and Write operations is the content to write. Erase QSPI Flash writes zeroes to all QSPI flash, but the Write QSPI Flash operation writes specific content to the QSPI flash.

- **Cyan Path: QSPI Controller DMA Read:** After the host issues a DMA read sequence to the QSPI controller via the blue path, the QSPI controller sends the data to the block RAM by the cyan path until the DMA data length reaches a defined size.

- **Purple Path: Host Reads Back QSPI Flash Data in Block RAM:** All these three paths take in the Verify QSPI Flash operation. First, The QSPI command is sent through the blue path. Secondly, the QSPI controller uses DMA to send the data to block RAM via the cyan path. Finally, the host fetches the block RAM data through the purple data path, and then compares with the local memory data.

The reference design accompanying this application note includes a host software design and Versal ACAP hardware design (see Reference Design). The reference design is verified successfully in a X86 host and Xilinx VCK190 evaluation board environment. The Xilinx VCK190 evaluation board uses the Versal device XCVC1902.

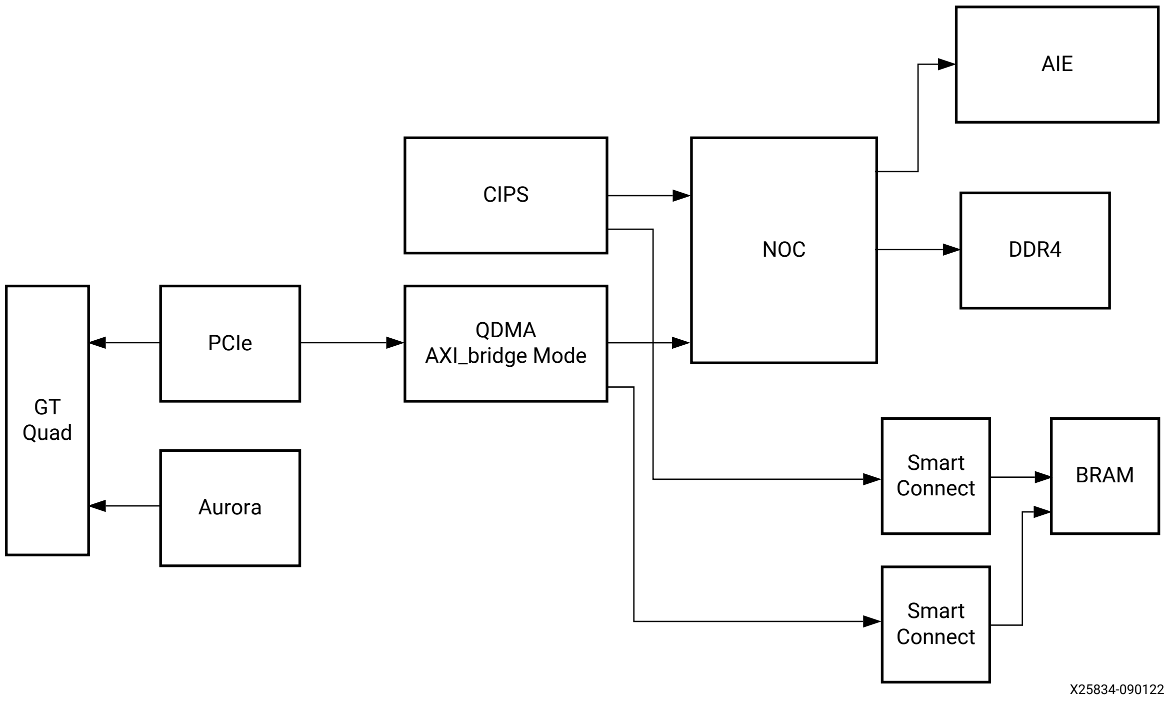
Versal Device Hardware Design

Block Design Introduction

It is assumed that customers use PCIe as a controller bus to connect the host (such as a x86 or Arm® processor) with the Versal device silicon. Normally, there are low bandwidth requirements, so GT Quad sharing with other protocols is realized, as an example.

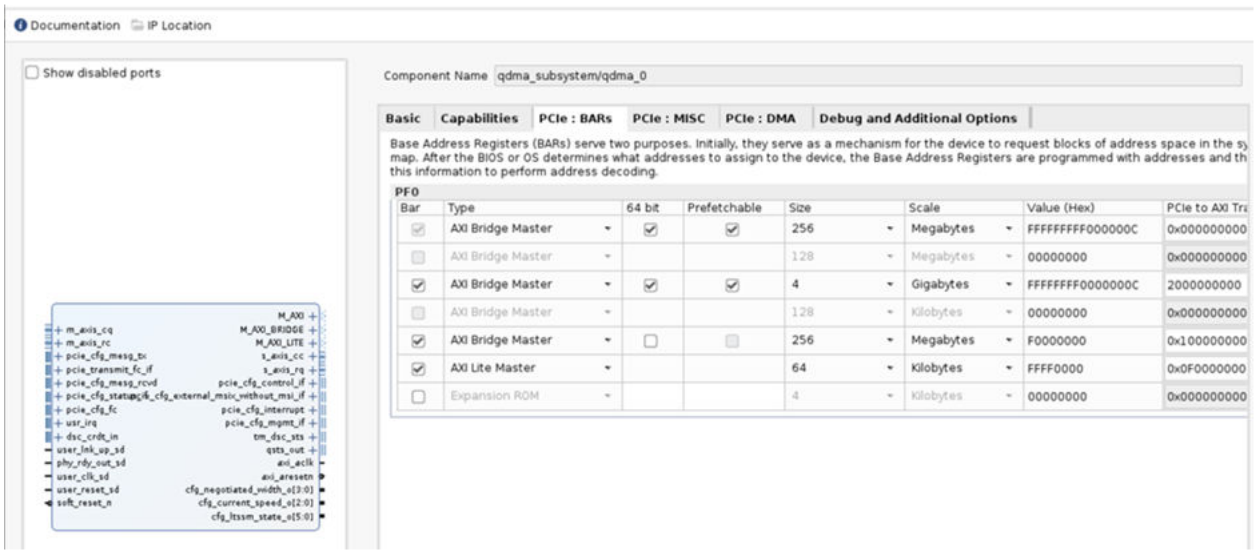
In the block design (BD), PCIe Gen2 x1 and Aurora share the same GT Quad. PCIe integrated IP in the PL and QDMA AXI_bridge mode are used. In the following figure, AI Engine, DDR4, and Aurora are not involved in the host programming of QSPI flash.

Figure 3: **Design Diagram**



Four base address register (BAR) spaces are defined, as shown in the following figure. The first is BAR1, which is an AXI bridge master that connects to data storage spaces through the NOC (DDR4) and SmartConnect (block RAM). The second BAR is an AXI bridge master that accesses AI Engine space through the NoC. The third BAR is an AXI bridge master that connects to the QSPI controller through the NOC. The fourth BAR is an AXI4-Lite master space to access block RAM on the Versal device.

Figure 4: **BAR Spaces**



Component Name qdma_subsystem/qdma_0

Base Address Registers (BARs) serve two purposes. Initially, they serve as a mechanism for the device to request blocks of address space in the sy
map. After the BIOS or OS determines what addresses to assign to the device, the Base Address Registers are programmed with addresses and th
this information to perform address decoding.

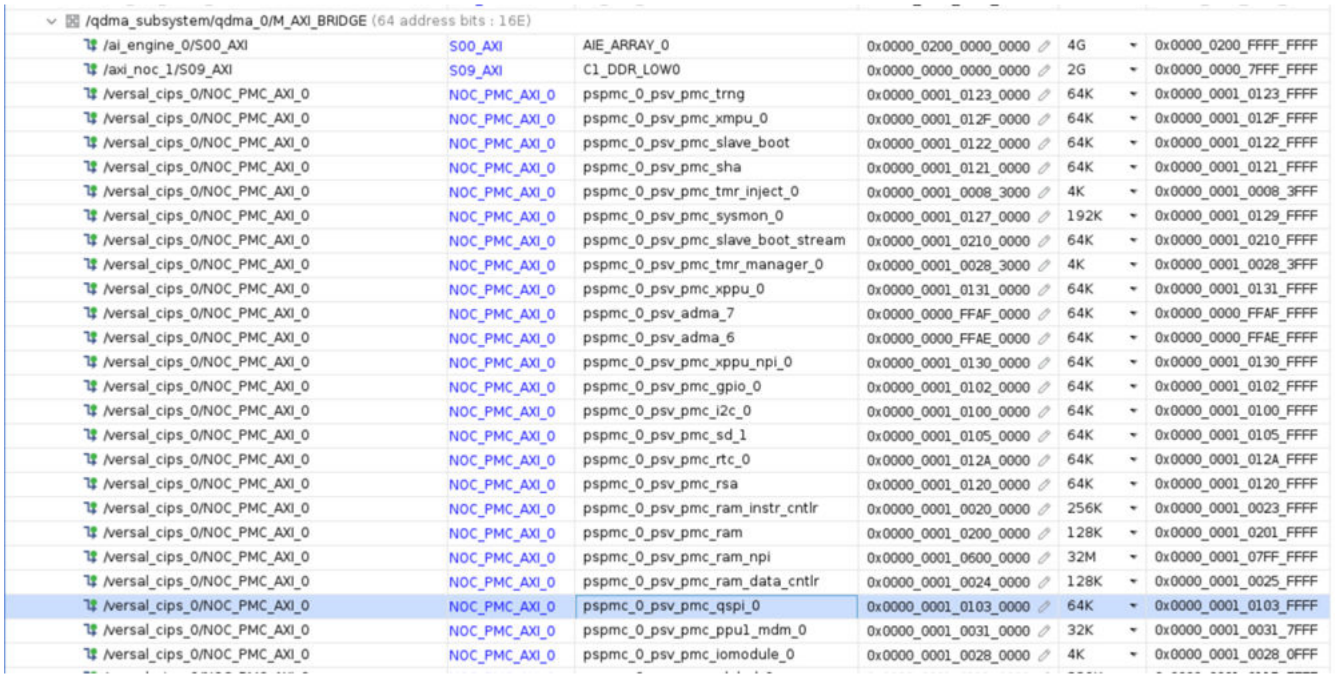
PF0

Bar	Type	64 bit	Prefetchable	Size	Scale	Value (Hex)	PCIe to AXI Tra
✓	AXI Bridge Master	✓	✓	256	Megabytes	FFFFFFFFF000000C	0x000000000
☐	AXI Bridge Master			128	Megabytes	00000000	0x000000000
✓	AXI Bridge Master	✓	✓	4	Gigabytes	FFFFFFFF0000000C	200000000
☐	AXI Bridge Master			128	Kilobytes	00000000	0x000000000
✓	AXI Bridge Master	☐	☐	256	Megabytes	F0000000	0x100000000
✓	AXI Lite Master			64	Kilobytes	FFFF0000	0x0F0000000
☐	Expansion ROM			4	Kilobytes	00000000	0x000000000

As the following address editor shows, BAR1 connects DDR, BAR2 connects AI Engine, and BAR3 connects PMC_SLAVES:

- axi_noc_1/S09_AXI: BAR1 space: The reference design does not access this space.
- ai_engine_0/S00_AXI: BAR2 space: The reference design does not access this space.
- versal_cips_0/NOC_PMC_AXI_0/pspmc_0_psv_pmc_qspi_0: This is the QSPI controller space inside the PMC space.
- versal_cips_0/NOC_PMC_AXI_0/pspmc_0_psv*: The reference design does not access these spaces.

Figure 5: **Versal Device Address Editor**

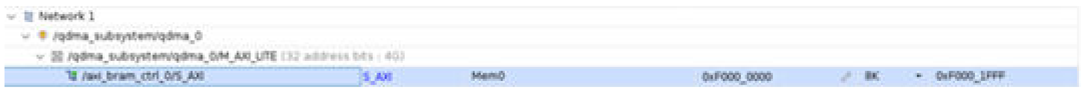


/qdma_subsystem/qdma_0/M_AXI_BRIDGE (64 address bits : 16E)					
/ai_engine_0/S00_AXI	S00_AXI	AIE_ARRAY_0	0x0000_0200_0000_0000	4G	0x0000_0200_FFFF_FFFF
/axi_noc_1/S09_AXI	S09_AXI	C1_DDR_LOW0	0x0000_0000_0000_0000	2G	0x0000_0000_7FFF_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_trng	0x0000_0001_0123_0000	64K	0x0000_0001_0123_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_xmpu_0	0x0000_0001_012F_0000	64K	0x0000_0001_012F_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_slave_boot	0x0000_0001_0122_0000	64K	0x0000_0001_0122_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_sha	0x0000_0001_0121_0000	64K	0x0000_0001_0121_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_tmr_inject_0	0x0000_0001_0008_3000	4K	0x0000_0001_0008_3FFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_sysmon_0	0x0000_0001_0127_0000	192K	0x0000_0001_0129_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_slave_boot_stream	0x0000_0001_0210_0000	64K	0x0000_0001_0210_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_tmr_manager_0	0x0000_0001_0028_3000	4K	0x0000_0001_0028_3FFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_xppu_0	0x0000_0001_0131_0000	64K	0x0000_0001_0131_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_adma_7	0x0000_0000_FFAF_0000	64K	0x0000_0000_FFAF_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_adma_6	0x0000_0000_FFAE_0000	64K	0x0000_0000_FFAE_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_xppu_npi_0	0x0000_0001_0130_0000	64K	0x0000_0001_0130_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_gpio_0	0x0000_0001_0102_0000	64K	0x0000_0001_0102_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_i2c_0	0x0000_0001_0100_0000	64K	0x0000_0001_0100_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_sd_1	0x0000_0001_0105_0000	64K	0x0000_0001_0105_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_rtc_0	0x0000_0001_012A_0000	64K	0x0000_0001_012A_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_rsa	0x0000_0001_0120_0000	64K	0x0000_0001_0120_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_ram_instr_cntlr	0x0000_0001_0020_0000	256K	0x0000_0001_0023_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_ram	0x0000_0001_0200_0000	128K	0x0000_0001_0201_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_ram_npi	0x0000_0001_0600_0000	32M	0x0000_0001_07FF_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_ram_data_cntlr	0x0000_0001_0024_0000	128K	0x0000_0001_0025_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_qspi_0	0x0000_0001_0103_0000	64K	0x0000_0001_0103_FFFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_ppu1_mdm_0	0x0000_0001_0031_0000	32K	0x0000_0001_0031_7FFF
/versal_cips_0/NOC_PMC_AXI_0	NOC_PMC_AXI_0	pspmc_0_psv_pmc_iomodule_0	0x0000_0001_0028_0000	4K	0x0000_0001_0028_0FFF

The following figure shows that BAR4 connects the block RAM controller. There is a space in the BAR4 that can be accessed by the host:

axi_bram_ctrl_0/S_AXI: This is the block RAM space that the host accesses through PCIe.

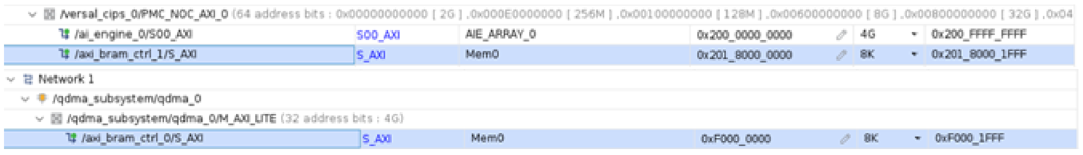
Figure 6: **BAR4 Space**



Network 1					
/qdma_subsystem/qdma_0					
/qdma_subsystem/qdma_0/M_AXI_LITE (32 address bits : 4G)					
/axi_bram_ctrl_0/S_AXI	S_AXI	Mem0	0xF000_0000	8K	0xF000_1FFF

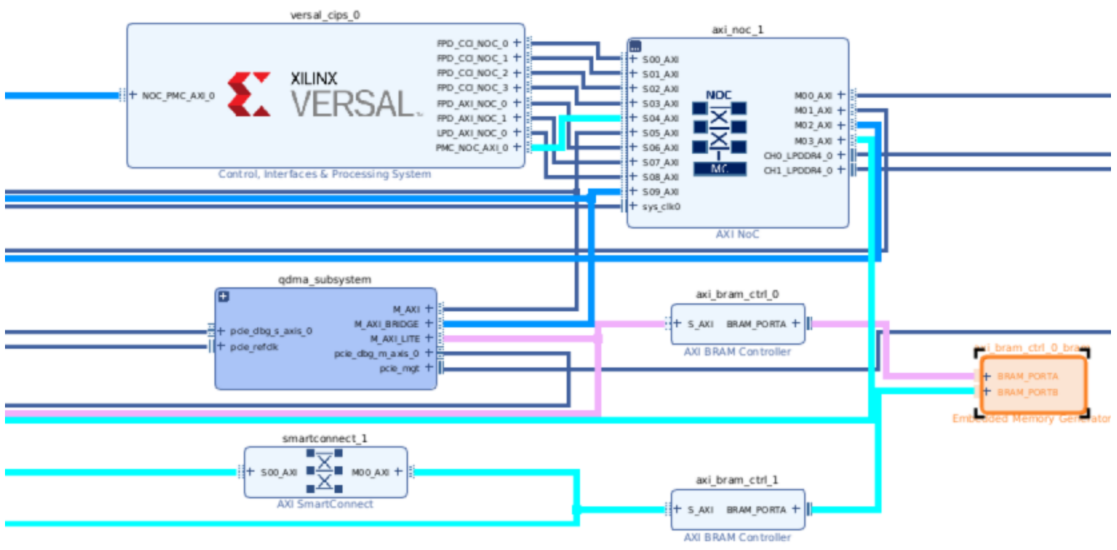
On the other hand, the block RAM controller connects with the dual-port block RAM to store QSPI flash data. The QSPI controller uses the DMA to transfer the data from QSPI flash to block RAM. There is another block RAM controller that connects the dual-port block RAM which reads out the QSPI flash data. There are two different addresses for the block RAM block. If the master is PCIe, the block RAM block's address is 0xF000_0000. If the master is the QSPI controller in PMC, the address is 0x201_8000_0000. This is more than 32 bits of address space. Configuring this address to the QSPI controller register, DMA_Dst_Addr_L = 0x8000_0000 and DMA_Dst_Addr_H = 0x0000_0201.

Figure 7: **BRAM Controller**



/versal_cips_0/PMC_NOC_AXI_0 (64 address bits : 0x00000000000 [2G] ,0x000E0000000 [256M] ,0x00100000000 [128M] ,0x00600000000 [8G] ,0x00800000000 [32G] ,0x04					
/ai_engine_0/S00_AXI	S00_AXI	AIE_ARRAY_0	0x200_0000_0000	4G	0x200_FFFF_FFFF
/axi_bram_ctrl_1/S_AXI	S_AXI	Mem0	0x201_8000_0000	8K	0x201_8000_1FFF
Network 1					
/qdma_subsystem/qdma_0					
/qdma_subsystem/qdma_0/M_AXI_LITE (32 address bits : 4G)					
/axi_bram_ctrl_0/S_AXI	S_AXI	Mem0	0xF000_0000	8K	0xF000_1FFF

Figure 8: **Datapath inside PL Design**



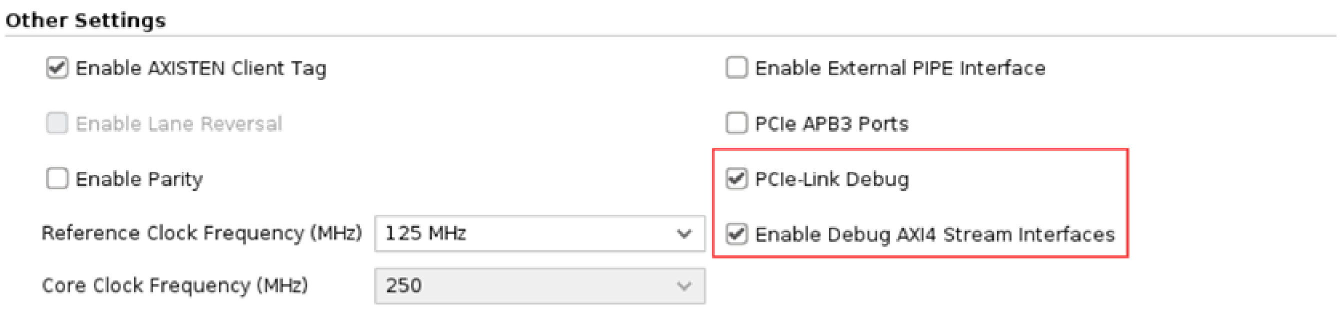
The host accesses the QSPI controller space by accessing BAR3. Data is transformed through the QDMA M_AXI_BRIDGE port, goes into the NoC slave port to the master port, and arrives at NOC_PMC_AXI_0.

The three colored datapaths are Versal device hardware datapaths in the entire system architecture. All the datapaths are identical in the whole design.

PCIe Debug Accessory

For easy debug of the PCIe link status, a PCIe specified debug hub and ILA are inserted in the BD.

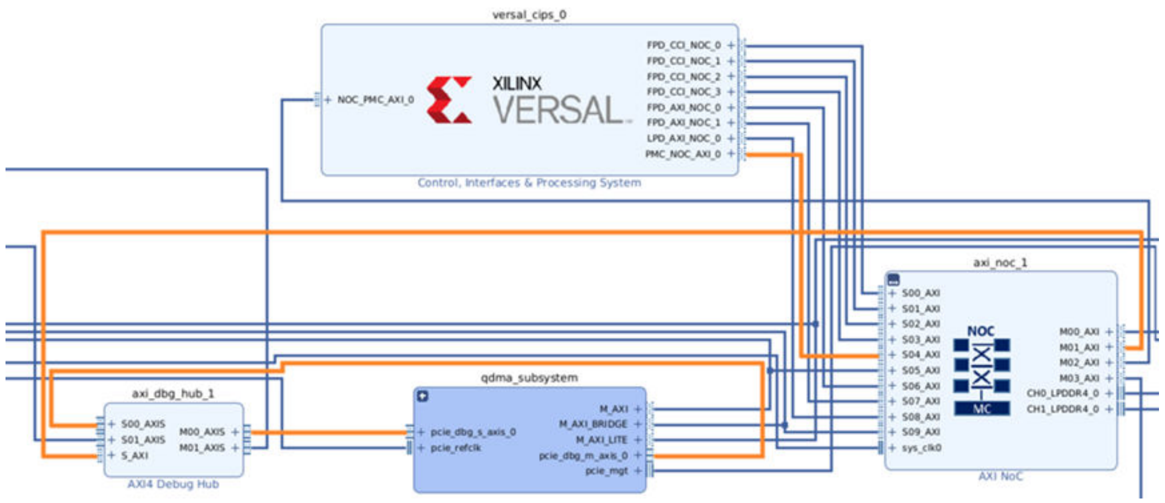
Figure 9: **PCIe Debug Hub and ILA**



Other Settings

☑ Enable AXISTEN Client Tag	☐ Enable External PIPE Interface
☐ Enable Lane Reversal	☐ PCIe APB3 Ports
☐ Enable Parity	☑ PCIe-Link Debug
Reference Clock Frequency (MHz): 125 MHz	☑ Enable Debug AXI4 Stream Interfaces
Core Clock Frequency (MHz): 250	

Figure 10: **PCIe Debug Hub and ILA Path**

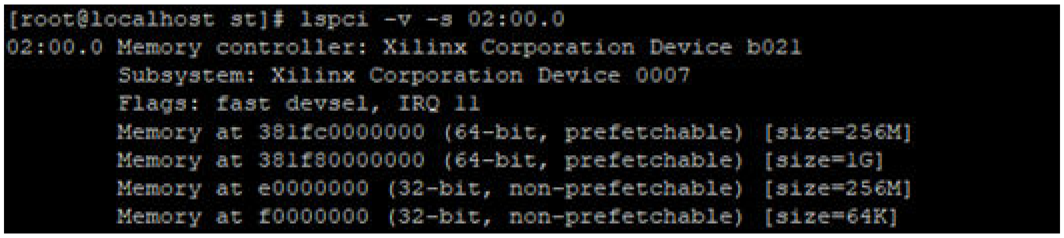


Host Software Design

Address Mapping

After setting up the host and VCK190 environment, list all the PCIe BAR spaces using the `lspci` command on the host computer. There are four PCIe BARs that the host could access, as shown in the following figure.

Figure 11: **PCIe BARs**



```
[root@localhost st]# lspci -v -s 02:00.0
02:00.0 Memory controller: Xilinx Corporation Device b021
        Subsystem: Xilinx Corporation Device 0007
        Flags: fast devsel, IRQ 11
        Memory at 381fc0000000 (64-bit, prefetchable) [size=256M]
        Memory at 381f80000000 (64-bit, prefetchable) [size=1G]
        Memory at e0000000 (32-bit, non-prefetchable) [size=256M]
        Memory at f0000000 (32-bit, non-prefetchable) [size=64K]
```

The BAR3 space has a size of 256 MB, which is from address 0xE000_0000 to 0xEFFF_FFFF. This BAR3 space is mapped to Versal device address 0x1_0000_0000 to 0x1_0FFFF_FFFF, which includes the PMC slave space. The QSPI controller is located at the PMC.

In this application note, the host also accesses the BAR4 space. There is a block RAM segment with a base address of 0xF000_0000 and size of 8 KB.

Table 1: **PCIe BARs Address Mapping**

PCIe BAR3 Space	Device Name	Versal Device Address Base
0xE103_0000 ~ 0xE103_FFFF	QSPI controller	0x1_0103_0000 ~ 0x1_0103_FFFF (64 KB)
0xF000_0000 ~ 0xF000_1FFF	Block RAM Port0 For read operation	0xF000_0000 ~ 0xF000_1FFF (8 KB)

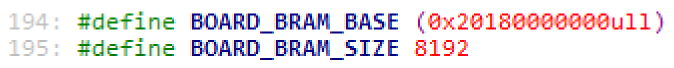
This block RAM has independent addresses for the two ports. As described previously, one address is for the host readback which maps to PCIe BAR4. Another address is written by the QSPI controller, so the software tells the QSPI controller that the BOARD_BRAM_BASE address is 0x201_8000_0000. This access is internal to the Versal device.

Table 2: **Dual Ports Block RAM Address Mapping**

PCIe BAR4	Device Name	Size (B)	Versal Device Address Base	Host Access
0xF000_0000	Block RAM Port0 For read	8K	0xF000_000	Host accesses BAR space
Nop	Block RAM Port1 For write	8K	0x201_8000_0000	Host configures this 64-bit address to QSPI controller DMA address register

In the software application, the BOARD_BRAM_BASE and BOARD_BRAM_SIZE macros are defined in the file `xqspipsu.h`.

Figure 12: **Address Mapping of Host Software**

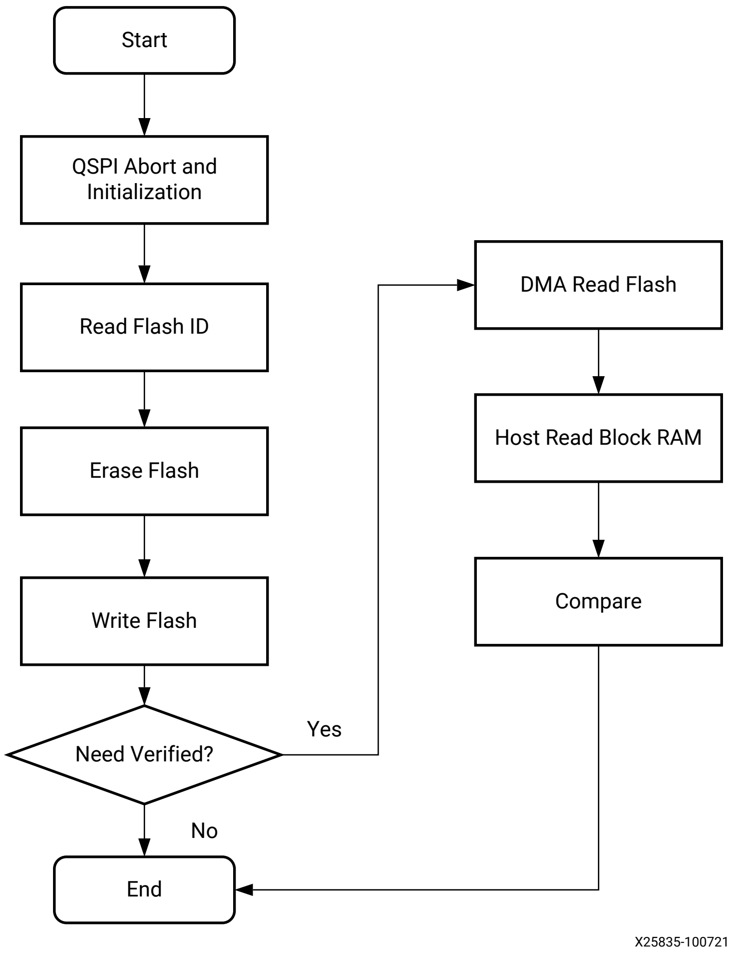


```
#define BOARD_BRAM_BASE (0x20180000000ull)
#define BOARD_BRAM_SIZE 8192
```

Host Application Data Flow

The following figure shows the overall programming guideline. Generally, the complete programming flow includes these subflows: QSPI Abort and Initialization, Read Flash ID, Erase Flash, Write Flash, and Verify Flash. In the Verify Flash subflow, there is a DMA Read Flash to the block RAM followed by a Host Read Block RAM to the Host local RAM and then a comparison on the host side.

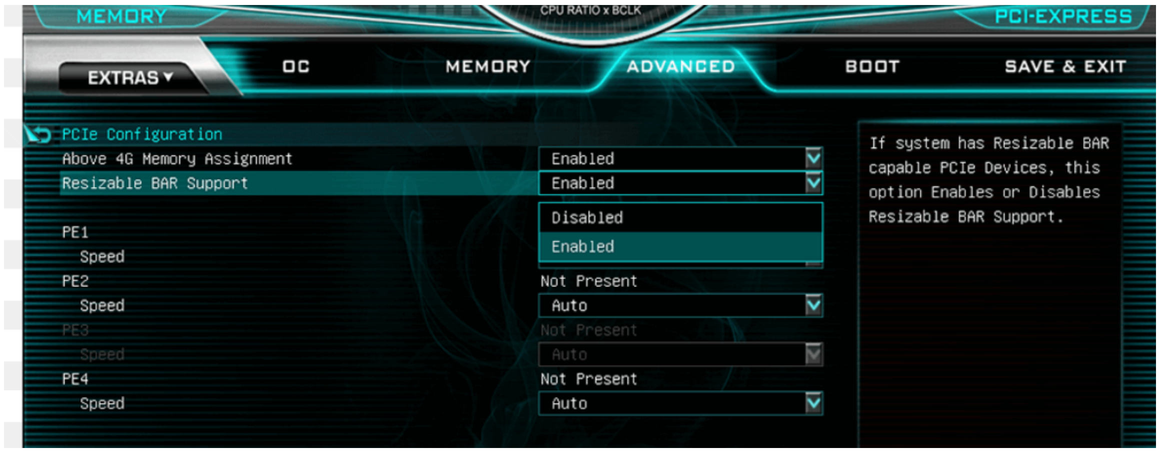
Figure 13: **Host Application Data Flow**



Debugging

Hardware Environment Setup

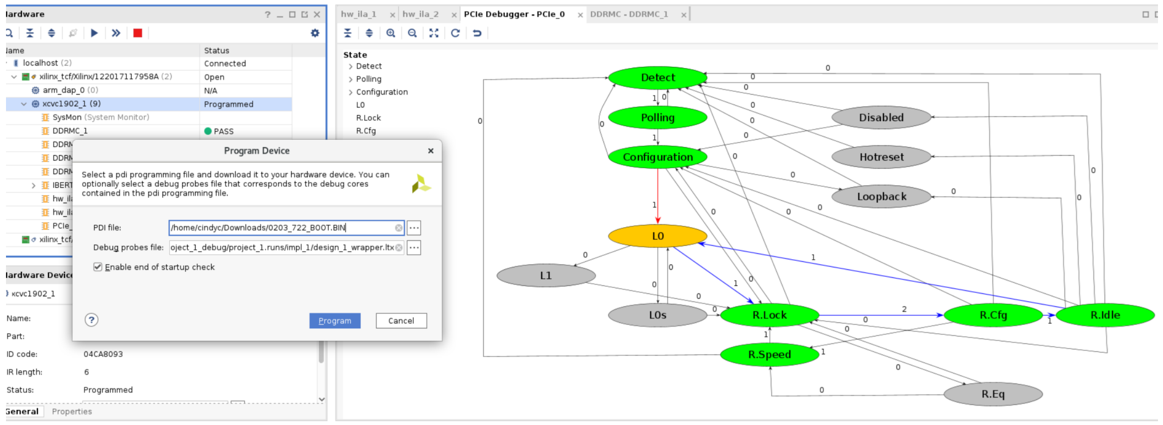
1. Some PCs might require **Above 4G Memory Assignment** and **Resizable BAR Support** to be set in the BIOS Configuration tab.



```
PCIe Configuration
Above 4G Memory Assignment        Enabled
Resizable BAR Support             Enabled
                                  Disabled
                                  Enabled
PE1
  Speed                           Not Present
PE2
  Speed                           Auto
PE3                               Not Present
  Speed                           Auto
PE4                               Not Present
  Speed                           Auto
```

If system has Resizable BAR capable PCIe Devices, this option Enables or Disables Resizable BAR Support.

2. Build the VCK190 project (refer to the `readme.txt` file in the reference design to build the VCK190 project). Download the PDI file through the Vivado hardware manager and check the PCIe link status.



Program Device

Select a pdi programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the pdi programming file.

PDI file: /home/cindyc/Downloads/0203_722_BOOT.BIN

Debug probes file: oject_1_debug/project_1.runs/impl_1/design_1_wrapper.ltx

☑ Enable end of startup check

Name	Status
localhost (2)	Connected
xilinx_tcf/Xilinx/122017117958A (2)	Open
arm_dap_0 (0)	N/A
xcvc1902_1 (9)	Programmed
SysMon (System Monitor)	
DDRMC_1	● PASS

ID code:	04CA8093
IR length:	6
Status:	Programmed

3. On the host side, use the `lspci` command to get the BAR address. Then use the `devmem` command to try to read and write the BAR spaces successfully.

Software Debugging

To compile the host application, select a directory and launch the application source code to this directory. Then compile the application by running the Makefile on an X86 host.

```
> make
```

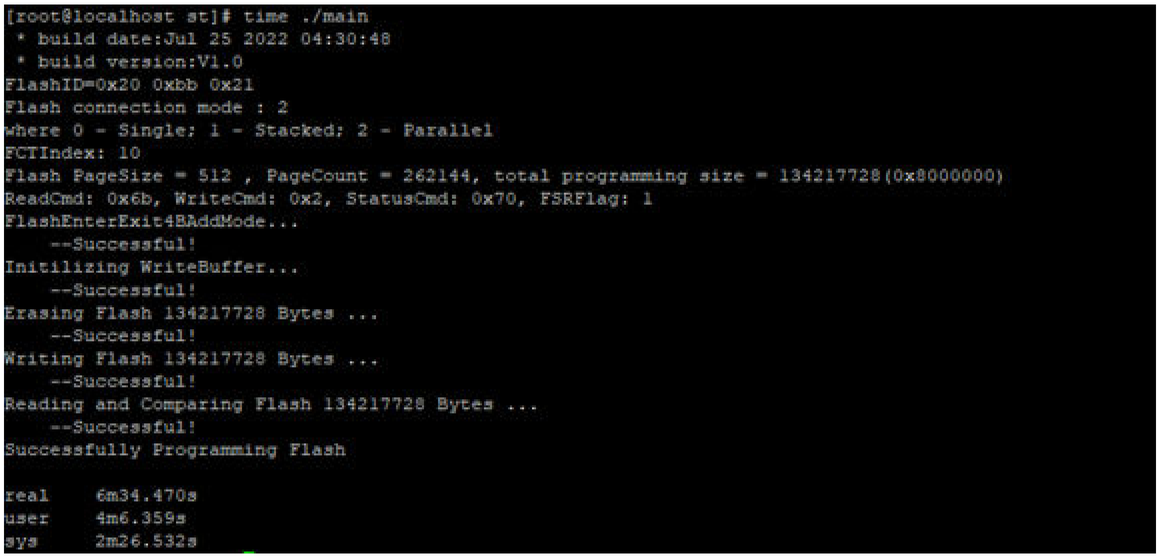
Three objects are generated, as shown in the following table.

Table 3: **Objects Generated by Running Makefile**

Object	Description
main	This executable file programs and verifies the 128 Mb flash.
program_only	This executable file programs the 128 Mb flash.
debug	This executable file programs the 128 Mb flash. At the same time, it opens the debug switch and prints all the debug logs.

Then run the executable files on the host. The following figure shows the log for the programming and verification of the 128 MB flash.

Figure 14: **Programming and Verification of 128 MB Flash**

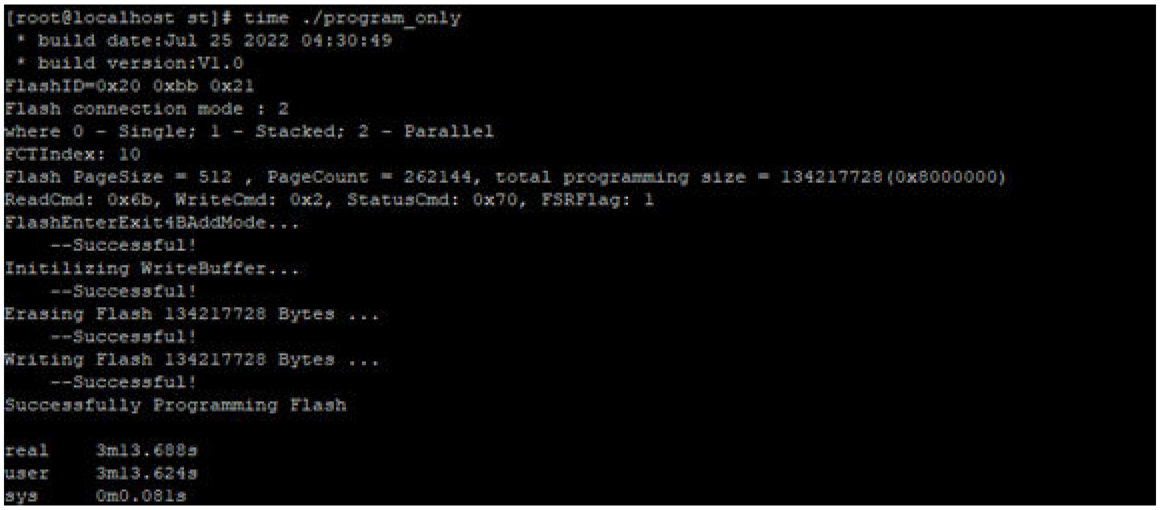


```
[root@localhost st]# time ./main
 * build date:Jul 25 2022 04:30:48
 * build version:V1.0
FlashID=0x20 0xbb 0x21
Flash connection mode : 2
where 0 - Single; 1 - Stacked; 2 - Parallel
FCTIndex: 10
Flash PageSize = 512 , PageCount = 262144, total programming size = 134217728(0x8000000)
ReadCmd: 0x6b, WriteCmd: 0x2, StatusCmd: 0x70, FSRFlag: 1
FlashEnterExit4BAddMode...
    --Successful!
Initilizing WriteBuffer...
    --Successful!
Erasing Flash 134217728 Bytes ...
    --Successful!
Writing Flash 134217728 Bytes ...
    --Successful!
Reading and Comparing Flash 134217728 Bytes ...
    --Successful!
Successfully Programming Flash

real    6m34.470s
user    4m6.359s
sys     2m26.532s
```

The following figure shows the log from programming the 128 MB flash.

Figure 15: **Programming of 128 MB Flash**



```
[root@localhost st]# time ./program_only
 * build date:Jul 25 2022 04:30:49
 * build version:V1.0
FlashID=0x20 0xbb 0x21
Flash connection mode : 2
where 0 - Single; 1 - Stacked; 2 - Parallel
FCTIndex: 10
Flash PageSize = 512 , PageCount = 262144, total programming size = 134217728(0x8000000)
ReadCmd: 0x6b, WriteCmd: 0x2, StatusCmd: 0x70, FSRFlag: 1
FlashEnterExit4BAddMode...
    --Successful!
Initilizing WriteBuffer...
    --Successful!
Erasing Flash 134217728 Bytes ...
    --Successful!
Writing Flash 134217728 Bytes ...
    --Successful!
Successfully Programming Flash

real    3m13.688s
user    3m13.624s
sys     0m0.081s
```

Performance

Programming the 128 MB image takes 193 seconds, which is 3 minutes 13 seconds. Comparing the two sets of performance results shown in the following table, it can be determined that there is a linear relation between time and programming size. For example, 48s x 4 = 192s, which is almost 193s.

Thus, 128 MB flash programming takes four times as long as 32 MB flash programming.

Programming + verify takes more time during this process. It is obvious that the verify time increases in a linear manner with the programming size.

Table 4: **Host Programming Performance**

Operation	Size (MB)	Time (s)	Description
Programming only	32	48	Read Flash ID, Erase Flash, and Write Flash
Programming only	64	97	
Programming only	128	193	
Programming + Verify	32	99	Read Flash ID, Erase Flash, and Write Flash Then Read Flash content and Compare
Programming + Verify	64	197	
Programming + Verify	128	394	

Reference Design

Download the reference design files for this application note from the Xilinx website.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 5: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target devices	Versal device XCVC1902
Source code provided?	Y
Source code format (if provided)	Vivado tools Tcl script for hardware block design/C++ for host application
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	N
Simulation	
Functional simulation performed	N
Timing simulation performed?	N
Test bench provided for functional and timing simulation?	N
Test bench format	N/A
Simulator software and version	N/A
SPICE/IBIS simulations	N
Implementation	
Synthesis software tools/versions used	Vivado synthesis
Implementation software tool(s) and version	Vivado implementation
Static timing analysis performed?	N
Hardware Verification	
Hardware verified?	Y
Platform used for verification	VCK190 evaluation board

Conclusion

QSPI flash is commonly used as a boot device. In Versal devices, there is an integrated QSPI controller in PMC which can be access by both internal A72/R5F APU (normally) and external CPU. This application provides a framework for an external CPU (host PC) programming of QSPI Flash memory. The external CPU communicates with the Versal QSPI Flash Controller through the PCIe, without involving the Versal PS. If DDR is inaccessible, this application does not even need DDR. The PL BRAM can be used instead.

References

These documents provide supplemental material useful with this guide:

1. *Versal ACAP Technical Reference Manual* (AM011)
2. *Versal ACAP Register Reference* (AM012)
3. *Versal ACAP DMA and Bridge Subsystem for PCI Express Product Guide* (PG344)
4. *Versal ACAP Transceivers Wizard LogiCORE IP Product Guide* (PG331)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
10/04/2022 Version 1.1	
General updates	Replaced XDMA with QDMA.
Versal Device Hardware Design	Added BAR4 to description of BAR spaces, and updated figures.
Host Software Design	
Debugging	• Updated with description of 128 MB flash. • Removed *Performance Optimization* section.
06/29/2022 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright