AR#00002 - QSPI Programming issues

Table of Content

Issue 1: Programming with Vivado /SDK tools failed in case the boot mode is QSPI

Affected Series:

- all Zynq Modules, especially 7 Series Zynq

Possible cases and solutions:

Vivado 2017.2 and older

- Case1: flash is empty
  - no issue
- Case 2: flash contains bootable design:
  - Possible Problem 1: running OS on zynq can prevent vivado to get access to QSPI
    - Solution 1: Stop booting for example on uboot console and try again
    - Solution 2: change boot mode some other a mode which does not boot any design

Vivado 2017.3 up to 2018.3

Programming procedure has changed (AR#70146), user must add additional FSBL now which initialise PS before Xilinx micro Uboot starts

- Case 1: flash is empty
  - Problem 1: Default FSBL stops working with error stage,because it didn't find bootable image on flash.
    - Solution 1: used special FSBL where boot mode is set fix to JTAG, see Xilinx AR#70548
- Case 2: flash contains bootable design
  - mostly no issue, default FSBL or special FSBL can be used to program flash
  - Possible Problem 1: running OS zynq can prevent vivado to get access to QSPI
    - Solution 1: Stop booting for example on uboot console and try again
    - Solution 2: change boot mode some other mode which

Vivado 2019.x up to 2020.2

Same programming procedure like 2017.3 up to 2019.3, but Vivado access to Zynqs seems to be changed.

- Case 1: flash is empty
  - Problem 1: Default FSBL stops working with error stage,because it didn't find bootable image on flash.
    - Solution: used special FSBL where boot mode is set fix to JTAG, see Xilinx AR#70548
- Case 2: flash contains bootable design
  - Problem 1: Vivado starts design from QSPI Flash before selected FSBL will be programmed. This seems to prevent Vivado from starting the selected FSBL correctly, which leads to inconsistencies and terminates the programming
    - Solution 1: use the same FSBL which was used on the design which was programmed

        - Solution 2: change boot mode some other a mode which does not boot any design and use special FSBL
    - Possible Problem 2: running OS zynq can prevent vivado to get access to QSPI
        - Solution 1: Stop booting for example on uboot console and try again
        - Solution 2: change boot mode some other a mode which does not boot any design

Vivado 2021.x up and newer

It's on evaluation.

- 7 Series Zynq: It seems to be not longer possible (we are checking possible solutions at the moment)
    - Workaround: Use older Vivado Version for programming or change Boot Mode to JTAG only if possible
- U+ Zynq: It seems to work with default FSBL now. Modified FSBL for Flash programming with Boot Mode set to JTAG on FSBL only does not longer work

Issue 2: Programming with third party tools or own software (baremetal, uboot) failed, in case PS is configured for x4 mode

Affected Series:

- all Zynq Modules, especially 7 Series Zynq

Possible cases and solutions:

- Problem 1: QE Bit of the QSPI Flash is not set and X4 access failed.
    - Solution: Write QSPI Flash one time with Xilinx Tools(Vivado or SDK), this will set QE Bit correctly. This only needs to be done once in case of a problem

Issue 3: Using more than 16MB flash on 7 Series Zynq

Affected Series:

- all 7 Series Zynq with more than 16MByte QSPI Flash

Possible cases and solutions:

- Problem: Drivers will change extended address register, in case more than 16MB is used. The extended address register is sticky, which means that only external reset events (like power cycle of the flash or external reset of the flash) can clear it. After a power cycle or an external reset of the QSPI flash, a recovery time needs to expire before accessing the device (see QSPI flash datasheet for more details). See also Xilinx AR#57744
    - Solution:

#	Solution	Good	Bad

0	Use single 16MByte flash	There are no issues. Can be implemented as assembly option, no PCB change needed, only BOM change.	SPI Flash is limited to 16MByte.
1	Use stacked 16MByte flashes	32MB can be safely accessed at full speed, only 1 extra pin needed from MIO,	Hardware (PCB) change, more space needed on PCB.
2	Use 16MB flash, parallel configuration	32MB can be safely accessed at full speed, very fast XiP.	Hardware (PCB) change, more space needed on PCB, almost all Bank500 pins are used for Flash.
3	Limit the access to lower 16Mbytes	No change of code or hardware.	Only 16MBytes can be accessed safely, may have to take special actions to actually limit the access to lower 16Mbyte
4	Preload everything above 16Mbytes in FSBL, limit access to lower 16Mbytes after FSBL handout	Only FSBL changes needed, use 24 bit bit addressing SPIx4 commands and not EAR register.	Access above 16MByte should not be performed from SSBL or application code, may have to take special actions to actually limit the access to lower 16Mbyte. All code above 16MByte has to be read in FSBL as one chunk, as last SPI Read command has to use address in lower 16Myte.
5	Preload everything above 16Mbytes in FSBL, limit access to lower 16Mbytes after FSBL handout	Only FSBL changes needed, use of 32 bit addressing SPIx1 commands and not EAR register.	Access above 16MByte should not be performed from SSBL or application code, may have to take special actions to actually limit the access to lower 16Mbyte. Flash reads above 16MByte in the FSBL are slower as the use x1 mode.
6	Rewrite FSBL, SSBL and OS/RTOS Drivers to avoid using EAR register and "legacy mode"	Truly safe solution, no hardware changes no restriction on SPI Flash Partitioning. Very good solution for bare metal applications.	A lot of Code and drivers to modify, the patches have to be applied again after each software release. Access to SPI Flash above 16Mbyte must be done using SPIx1 mode command set, when using good speed optimized code the performance penalty is not that bad.
7	Place "reboot. bin" at 16MByte boundary	No change of code of hardware.	256KByte sector at 16Mbyte offset in SPI Flash is "reserved" it must contain the "reboot.bin" image, special tool and/or scripts are needed to assemble the SPI Flash images to satisfy this requirement. If reset occurs while EAR =! 0 then Zynq PS is doing double reset sequence, first the reboot.bin executes, then it clears EAR and forces Zynq ARM core to reset followed by normal boot from Flash Address 0. However as reboot. bin does not perform any peripheral or memory or PLL initialization it executes very fast so the extra delay in startup is small. Boot history registers are also affected as there is sometimes extra reset involved during the boot. The likelihood of the double-reset to happen can be reduced if SSBL

			and application software do always include a dummy read from lower 16MByte after accesses to addresses above 16MB.
8	Duplicate FSBL at 16MByte boundary	Small change of FSBL, same FSBL at offset 0 and 16MByte. Code change affects only EAR register, all SPI Reads are still done using x4 commands. Same FSBL executes always no matter from what offset it was loaded, there is no significant change in startup time. There is no extra reset involved.	256KByte sector at 16Mbyte offset in SPI Flash is "reserved" it must contain the "boot.bin" image (with the same FSBL as at offset 0), special tool and/or scripts are needed to assemble the SPI Flash images to satisfy this requirement.
9	Duplicate EAR modified FSBL at 16MByte boundary	Small change of FSBL, mofidied FSBL at offset 16MByte. Code change affects only EAR register, all SPI Reads are still done using x4 commands. Functionally same FSBL executes always no matter from what offset it was loaded, there is no significant change in startup time, FSBL at normal start offset 0 is not modified at all. There is no extra reset involved.	256KByte sector at 16Mbyte offset in SPI Flash is "reserved" it must contain the "boot.bin" image (with the EAR patched FSBL as at offset 0), special tool and/or scripts are needed to assemble the SPI Flash images to satisfy this requirement. Two versions of the same FSBL have to be compiled each time when FSBL is changed or generated.
10	System Controller in external CPLD forcing SPI Flash reset using Flash reset pin. See also Xilinx AR#57744	Small changes of software (need to pull one MIO Pin to fixed level).	One extra MIO pin is wasted. CPLD has to detect reliable all types of resets, this is only possible with software assistance. This detection may fail during debug sessions, so extra operation mode may have to be implemented to disable the CPLD reboot resets temporary. Have to use special SPI Flash IC with dedicated Reset input.
11	System Controller in external CPLD forcing SPI Flash reset by controlling the power rail of the Flash. See also Xilinx AR#57744	Small changes of software (need to pull one MIO Pin to fixed level). Can use Flash IC with no dedicated Reset pin.	One extra MIO pin is wasted. CPLD has to detect reliable all types of resets, this is only possible with software assistance. This detection may fail during debug sessions, so extra operation mode may have to be implemented to disable the CPLD reboot resets temporary. CPLD has to be able to control the power rail of the SPI Flash using FET switch, or then be able to control the power supply that delivers the power to the Flash. There is extra FET and CPLD control pin needed, or if Flash shares power with other components then the complete power rail has to be turned off to implement Flash Reset.
12	System Controller	Small changes of software (need to	One extra MIO pin is wasted. CPLD has to detect reliable all types of resets, this is only possible

	in external CPLD forcing SPI Flash reset by using JTAG Boundary scan commands.	pull one MIO Pin to fixed level). Can use Flash IC with no dedicated Reset pin. No need to switch off power from SPI Flash.	with software assistance. This detection may fail during debug sessions, so extra operation mode may have to be implemented to disable the CPLD reboot resets temporary. CPLD has to be implement a JTAG functionality and play back a sequence that shifts in Reset command into SPI Flash. System Controller CPLD has to have access to Zynq JTAG and be large enough to implement the JTAG sequence playback.
13	System Controller in external CPLD implementing watchdog and fall-back from SPI mode to SD Card boot mode.	No software changes.	SD card must be available as boot media all the times.
14	External Watchdog forcing full power off cycle.	No software changes. No hardware changes to the module/SoM.	Hardware changes to the system or base board.
15	Limit SPI Flash access to Lower 16MB, use eMMC for main storage.	No changes if eMMC is supported and available in the target hardware. Large nonvolatile storage in eMMC.	