fidus
innovate•design•deliver

Multiboot with Fallback on AMD® Zynq™ UltraScale+™ MPSoC

AMD Adaptive Computing Partner PREMIER

A FIDUS SYSTEMS **WHITE PAPER**

Contents

1. INTRODUCTION

Remote and secure updates of system firmware are critical for operational integrity of high reliability electronic products. This paper outlines the implementation of a secure update mechanism with a focus on multiboot strategies implemented on ZYNQ MPSOC devices. The focus is on the implementation of A/B multiboot mechanism with fallback as supported by ZYNQ MPSOC/ RFSOC.

This paper is structured as follows. First, a high-level overview of bootloader architectures that support firmware updates is explored. Then we outline the architecture of the multiboot bootloader with A/B image switching and fallback. Next, we outline the Over the Air (OTA) update mechanism integrated on the RTOS side. Finally, we cover security considerations for this proposed architecture.

2. HARDWARE PLATFORM

The implementation of this design was done on an Avnet Ultrazed EV board. This development board uses two QSPI devices configured in dual parallel mode. The two Micron 512 Mb (64 MB) devices allow for a total of 128 MB of storage in the 0x00000000 – 0x08000000 address range. In dual parallel mode the data bytes are bit-interleaved across the two QSPI devices. This mode requires configuring the driver in 32-bit mode. The boot mode is set for QSPI32 via boot mode pins [3:0] = 0010.

3. SYSTEM COMPONENTS

The implementation of multiboot on ZYNQ MPSOC requires a syncretic approach that leverages a modified FSBL running from OCM, boot images, status metadata and embedded webservers running from QSPI flash. Specifically, the QSPI flash layout contains the following components:

- Boot Images — one or more boot images, each of which may be a basic boot image such as a bootloader (FSBL only) or full featured boot image containing an FSBL, bitstream, Free-RTOS application.
- Status Metadata are pages with custom data fields used to track the state/status of the boot process.
- HTML/JS/CSS files for embedded webserver

Due to the size limit of the QSPI-NOR flash device the maximum size of images should be smaller than 30MB to allow space for the recovery bootloader and metadata registers.

A boot image is defined as any self-containing binary that can be a single first stage bootloader (FSBL) or a complete full featured image containing the FSBL, bitstream and RTOS application.

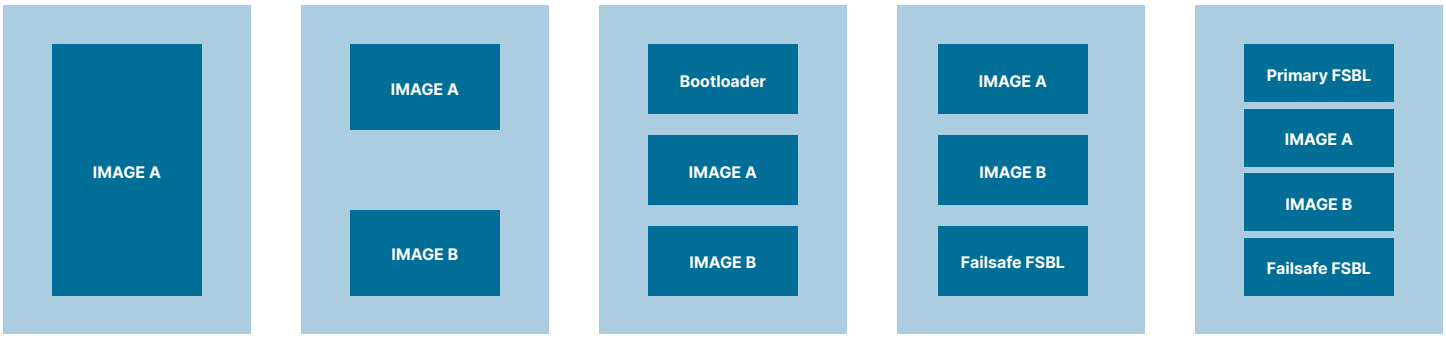
4. BOOTLOADER ARCHITECTURES

There is a large variety of permutations of bootloader designs, boot images, data blocks, on how they may be arranged in QSPI to support system update and recovery. This section provides an overview of some possible methods and outlines some options in greater detail.

The figure below shows some basic flash memory configurations for boot images. For simplicity, any additional non-image data blocks are not shown.

Option A shows the single slot bootloader implementation used to hold a single boot image. When an update is applied, it overwrites the previous image. The new working image is copied to and executed from RAM (OCM/DDR). The main disadvantage of this approach is that if the update image is corrupted or the update fails midway due to a communication channel, recovery would require another mechanism.

Option B is the simplest configuration that supports the A/B image switching with two slots for boot images. The implementation stores a golden image in Slot A and a working image in Slot B. The golden image serves as a fallback image in the event of a failed update of Image B.



Option C is similar to option B with the addition of a bootloader. The bootloader is executed after every reset and determines which image to boot based on stored state information. The wolfBoot Secure Bootloader from wolfSSL [8] implements this option by storing state information at the start of each partition (slot) and additionally supports chain-loading the image in Slot A while the system is still executing. This capability would be most useful in a system in which the boot images are being executed in place (XIP) directly from flash. In the wolfBoot implementation, the boot image is in Slot A (primary) and a new update image is in Slot B (secondary). When an update is applied, the images in the two slots are swapped sector by sector using a reserved swap partition. If an update fails, the bootloader can swap the images back and boot from the previous image.

Option D is a combination of options B and C that is specific to how the Xilinx ZynqMP implements its MultiBoot and Fallback features. When booting, the ZynqMP (specifically the CSU BootROM) will look for a valid boot image starting at flash address 0x00000000. Bootloader or application code can use the MultiBoot feature to adjust the image search offset. If a valid image isn't found at a given address, the Fallback feature will increment the search offset by 32 KB and continue the search. The search will only continue until the end of the flash memory is reached. The job of the Failsafe Bootloader is to simply reset the search address back to 0x00000000 to continue the search. The use of Slots A & B is the same as in option B. Note: an image search will continue until the end of the entire region mapped for QSPI — e.g., 512 MB in this case. Without a failsafe bootloader, the image search will technically wrap around to the starting point after the first 128 MB of physical memory is searched but a failsafe bootloader makes this explicit and should work even if the physical memory size is increased to the full 512 MB allowed.

Option E is a variation of option D with the addition of a primary bootloader. The sole task of the primary bootloader is to use boot status information to determine which of the following images should be used, then set the MultiBoot offset to "point" to the desired image and then trigger a soft reset to "jump" to that image.

5. SYSTEM COMPONENTS

5.1 Boot Status Metadata / Backup

The Boot Status Metadata (BSM) structure is defined in BootStatus.h as follows:

```
typedef struct
{
    uint32_t  tag;                    /**< 0x42534442 (BDDB)                                                          */
    uint16_t  version;                /**< Version number of this data structure                                      */
    uint16_t  length;                 /**< Number of bytes taken by this version of data structure (after length field!) */
    uint8_t   lastImage;              /**< Currently/last working image (normally same as staged image)               */
    uint8_t   requestedImage;         /**< Staged image (image to be loaded)                                          */
    uint8_t   swRollbackStatus;       /**< Rollback status (inactive/attempting/failed)                               */
    uint8_t   imageABootable;         /**< Status of image A (1 - bootable, 0 - not bootable). Marked by FSBL         */
    uint8_t   imageBBootable;         /**< Status of image B (1 - bootable, 0 - not bootable). Marked by FSBL         */
    uint8_t   reservedFutureUseImageA; /**< Reserved for future use (for image A)                                     */
    uint8_t   reservedFutureUseImageB; /**< Reserved for future use (for image B)                                     */
    uint8_t   softwareUpdateStatus;   /**< Current state of software update process (inactive by default)             */
    uint32_t  imageAOffset;           /**< Offset of image A                                                          */
    uint32_t  imageBOffset;           /**< Offset of image B                                                          */
    uint32_t  recoveryImgOffset;      /**< Offset of recovery image                                                   */
    uint32_t  CRC32;                  /**< Use standard IEEE 802.3 CRC32 calculation                                  */
} BootStatusData;
```

Field	Values	FSBL	OS	Usage
tag	0x42444442 ('BDDB')	R	R	Indicates the data structure type.
version	<uint16_t>	R	R	Data structure version — allows for future extension of the structure.
length	<uint16_t>	R	R	Number of bytes remaining in the structure after the length field.
lastImage	0x01 ImageSlotA 0x02 ImageSlotB 0x03 ImageSlotRecovery 0xFF ImageSlotUnknown	R/W	R/W	The last (working) image successfully booted. At the FreeRTOS level, this value indicates which QSPI image it was successfully booted from.
requestedImage	0x01 ImageSlotA 0x02 ImageSlotB 0x03 ImageSlotRecovery 0xFF ImageSlotUnknown	R	R/W	The image to be loaded as requested by the higher-level software (FreeRTOS). For example, during a software update, the FreeRTOS update application will set requestedImage to the newly updated image (A or B). This is also referred to as the "staged" image.
swRollbackStatus	0x01 – Rollback_Attempting 0x02 – Rollback_Failed 0xFF – Rollback_Inactive	R	R/W	Used by FreeRTOS to track the status of a software rollback — i.e., reverting a SW update to the previous version.
imageABootable	0x00 – not bootable 0x01 – bootable	R	R/W	Indicates that the image has successfully booted completely thru the FreeRTOS level.
imageBBootable	0x00 – not bootable 0x01 – bootable	R	R/W	Indicates that the image has successfully booted completely thru the FreeRTOS level.
reservedFutureUseImageA	<uint8_t>	–	–	0xFF is default after memory erase.
reservedFutureUseImageB	<uint8_t>	–	–	0xFF is default after memory erase.
softwareUpdateStatus	0x01 – SoftwareUpdate_Attempting 0x02 – SoftwareUpdate_Executed 0x03 – SoftwareUpdate_Failed 0xFF – SoftwareUpdate_Inactive	R	R/W	Used by FreeRTOS to track the status of a software update. Default: inactive
imageAOffset	<uint32_t>	R	R	Offset per memory map above.
imageBOffset	<uint32_t>	R	R	Offset per memory map above.
recoveryImgOffset	<uint32_t>	R	R	Offset per memory map above.
CRC32	<uint32_t>	R/W	R/W	Calculated over the entire data structure with the exception of this field. Used to ensure the data integrity of the BSDB.

Table 5-4 Boot Status Data Block

The CRC-32 shall be implemented according to the IEEE 802.3 standard. Specifically, the following polynomial is used:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The normal representation for this polynomial is 0x0x04C11DB7 but we are using the reversed representation which is 0xEDB88320.

The default settings for both blocks will then be as follows:

Field	Value
tag	0x42444442 ('BDDB')
version	1
length	32
lastImage	0x01 (Image A)
requestedImage	0x01 (Image A)
swRollbackStatus	0xFF (inactive)
imageABootable	0x01 (bootable)
imageBBootable	0x01 (bootable)
reservedFutureUseImageA	0xFF
reservedFutureUseImageB	0xFF
softwareUpdateStatus	0xFF (inactive)
imageAOffset	Offset per memory map above.
imageBOffset	Offset per memory map above.
recoveryImgOffset	Offset per memory map above.
CRC32	Computed CRC-32 value.

Table 5-5 Default Boot Status Data Block Configuration

As can be seen in the memory map above, there are two copies of the Boot Status Data Block. The Boot Status data structure includes a CRC-32 field that is used to verify the basic data integrity of the structure. If the primary BSDB has been corrupted, the backup BSDB can be used to recover. The primary BSDB and backup BSDB are processed according to the following rules which are applicable to both FSBL and FreeRTOS applications — for the FSBL applications, all block verification rules would be applied in the "Read/Verify A/B persistent registers" action shown in Table 5-5 above.

5.2 Primary FSBL

The image selector algorithm that reads the primary BSDB or backup BSDB is responsible for verify the CRC before processing the rest of the data.

Due to lack of space on the On Chip Memory (OCM), a function Is used to calculate the CRC32 on the fly as opposed to a lookup table. The CRC algorithm shall be the CRC-32 as specified by the IEEE 802.3 standard using the reversed polynomial representation as noted above.

The algorithm that reads or writes Boot Status data first attempts to use the primary metadata registers. In case those are corrupted, the procedure is repeated by reading the backup register. In case the integrity of the backup register is verified, the primary register is written with a cloned copy of the backup register results. If the data integrity cannot be verified for both the primary BSDB and the backup BSDB, the code shall switch to the recovery application.

Integrity checks are implemented to validate that the length value is consistent with the version and is less than or equal to the sector size and the lastImage, requestedImage, swRollbackStatus, imageABootable, imageBBootable, and softwareUpdateStatus values are consistent with the defined types.

5.2.1 A/B image switching algorithm

The A/B boot concept is based on two images in QSPI that operate in ping-pong fashion. The idea is to alternate between the two "A" and "B" User Partitions in QSPI where one is designated as the "current" image and the other is the "backup" image.

The FSBL is responsible for implementing the logic that decides on which image to boot by reading a reserved area of the QSPI memory that holds metadata that indicate which is the last booted image, which is the requested image that will be booted on the next power on reset event and if the images are actually bootable.

A **bootable** image in the A/B multiboot with fallback mechanism is defined as an image in QSPI that has successfully booted until the RTOS application stage. A **non-bootable** image is one that fails a data integrity check or cannot boot to the RTOS application layer stage. An image that contains the RTOS application that has just been updated in QSPI will be marked as non-bootable until it has been used to boot into the RTOS application. It is the responsibility of the RTOS application to update the bootable flags of the actual image.

A **requested image** refers to the image which is intended to become the current working image. For example, if a new system update image has been written to Image B, then Image B will be marked as both non-bootable and requested.

The **last booted image** refers to the last image which successfully booted. In the example above, Image A would be the last booted image.

A designated number of pages from the QSPI are set aside to maintain a set of persistent boot state registers to track and maintain the state of the requested image, the bootable/non-bootable state for each image, image offsets, as well as other information that may be required as part of the boot process.

When the system is rebooted, either via a power-on reset (POR) or via a system reset, an algorithm that implements the image selector portion of the application uses the information in the persistent boot state registers to determine which of Image A or Image B should be booted. The image selector application source for an example application can be found at [4].

The image selection algorithm was revised and augmented with the capability to check the integrity and update the metadata boot status registers. The algorithm operates as follows. On startup, the primary metadata register is read. If the CRC32 integrity check fails, the backup metadata register is read. In the unlikely scenario that both metadata pages are corrupted the primary first stage bootloader jumps to the recovery bootloader which resets the CSU register and spawns an embedded webserver accessed with a static IP address. This allows the user to connect the unit to a host via an Ethernet cable and reset the metadata to the default values followed by uploading a working binary image to the QSPI.

5.3.1 Recovery FSBL

The Recovery FSBL is a modified ZYNQ MPSOC FSBL which resets the multiboot register to 0 pointing to the start of the QSPI memory map and issues a soft reset.

6. OS IMAGE UPDATE

The RTOS update mechanism is comprised of the following components:

a. Websocket middleware with SSL/TLS support
b. QSPI driver
c. CRC32 library

The FreeRTOS application is responsible for updating the QSPI with a new image. An image update task starts with the requested Image A booting up to FreeRTOS application.

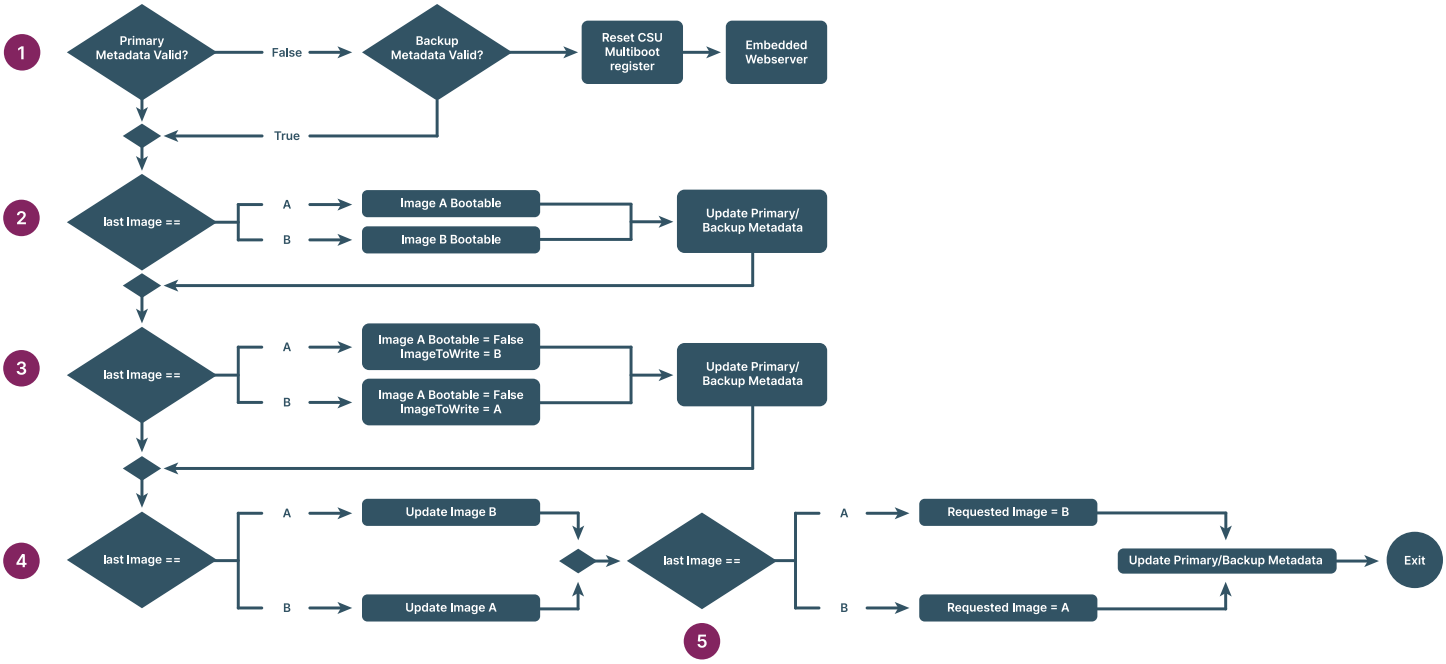
In general, the firmware update data is received via a communication channel such as serial, CAN, Ethernet or others. The choice of communication channel does not affect the multiboot A/B implementation as it's decoupled from the image switcher algorithm and is part of the RTOS task implementation.

FreeRTOS applications use the Xilinx driver for QSPI access. The driver uses the physical sector size, 64 KB, for erase commands. With the dual-parallel flash configuration, this means that the erase command will erase 128 KB so all images and data blocks in QSPI should be aligned to 128 KB (0x20000) addresses.

The QSPI update task can be partitioned into two tasks. Task A is responsible for authenticating the external webserver by loading the certificates, establishing the TCP connection, performing an SSL/TLS handshake, and finally sending the WebSocket handshake request to upgrade the protocol. After the Websocket handshake response is received, the RTOS client tasks can start receiving the firmware binary via secure WebSocket protocol in chunks.

Task B communicates with Task A via a Queue receiving the pages, performing a CRC check and writing the pages to QSPI by leveraging the Xilinx QSPI driver. Before the image page update is performed, this task reads the metadata pages, updates the bootable flags for the image that booted, marks the Image B slot as bootable and only then starts writing the pages for Image B. For each page update the CRC is calculated and checked. Once the QSPI update process is finalized the bootable flag data are updated to switch to the new image.

In the case where Image B is not bootable due to a corruption of the boot.bin binary image the primary FSBL will check the bootable flags and switch back to the previous Image A.



6.1 Image Update algorithm RTOS

This algorithm only addresses the QSPI image update and the bootable image fields. The bootable fields indicate that the image was booted all the way successfully from the FSBL to the FreeRTOS application. The flags are set by the FreeRTOS app and the FSBL is only responsible for reading the flags and deciding which image to boot based on their status but not modifying them.

The first step of the FreeRTOS QSPI update task is to read the primary metadata register. If this register is corrupted the backup metadata register is read. If the backup metadata register is also corrupted the multiboot register jumps to the failsafe recovery bootloader address whose purpose is to reset the CSU multiboot register and launch a single page web application that uses LWIP as a network stack. To deal with the constrained QSPI memory space the low-level raw API is used to send GET requests.

The second step is to update the bootable flags for the image that actually booted based on the bootable and requested flags values. The third step is to mark the updated slot as not bootable. During the next step the image update occurs. The last step requires the task to set the requested image to the updated image.

In the rare event where the communication during an update is interrupted and cannot continue the WDT would kick in and boot the previous working image. If the primary or backup metadata have been altered in such a way that this is not possible then the primary bootloader will jump to the recovery bootloader which resets the CSU multiboot register and launches an embedded webserver with a single page app that allows for a manual upload of the golden QSPI image.

7. SECURE UPDATES

ZYNQ MPSOC/ RFSOC devices support device encryption using a PUF key. After the bootloader components have been compiled and validated on hardware, they are encrypted with a public key. The bif file is updated to support both authentication and encryption. The encrypted images can only run on the device that is associated with the keys used during encryption. All boot images must be both authenticated (signed) and encrypted. The device encryption key cannot change since it is wrapped by the PUF KEK and burned in eFUSE and in addition there are only two authentication “keys” (PPK0, PPK1) supported.

8. CONCLUSION

This application note proposed a secure and robust bootloader architecture that supports A/B image switching updates and multiboot with fallback as supported on ZYNQ devices. The main aim was to implement a secure and safe firmware update mechanism that allows the product design team to remotely update system firmware.

9. REFERENCES

[1] Zynq UltraScale+ MPSoC Embedded Design Methodology Guide (UG1228)

[2] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/2662138473/Image+Selector+ImgSel+Utility

[3] Bootgen User Guide (UG1283)

[4] https://github.com/Xilinx/embeddedsw/tree/master/lib/sw_apps/imgsel

[8] https://www.wolfssl.com/products/wolfboot/

20+
years experience

Collaborating with smart teams is what fuels us every day.

3,000+
successful projects

Your unique challenges are our obsession.

400+
customers

Extending your team with our expertise brings designs to market faster.

95%
repeat customers

Customers love to work with us, again and again.

ABOUT FIDUS

Fidus Systems, founded in 2001, specializes in leading-edge electronic product development with offices in Ottawa and Waterloo, Ontario, and San Jose, California. Our hardware, software, FPGA, verification, wireless, mechanical and signal integrity teams work to innovate, design and deliver next-generation products for customers in emerging technology markets. Fueled by 20+ years' experience and creativity, along with our collaborative and process driven approach, we turn complex challenges into well-designed solutions. And with over 400 customers and 3000+ completed projects, we have the expertise to be a seamless extension of your team, providing a clear focus and commitment to getting designs and prototypes to market faster. Once you start working with us, you'll trust us like one of your own. Our hallmark is transparency. Our guiding principle is first time right.

fidus
innovate•design•deliver

fidus.com