USB RAID 1 Disk Design Using EZ-USB® FX3S™

Author: Hingkwan Huen
Associated Project: Yes
Associated Part Family: EZ-USB® FX3S™
Software Version: N/A
Related Application Notes: AN75705 - Getting Started with FX3

AN89661 describes how to design and implement a USB Redundant Array of Independent Disks (RAID) level 1 disk using EZ-USB® FX3S™. It includes an example design of a USB RAID 1 disk for a server boot storage application to help you develop server boot storage solutions.

Contents

1. Introduction

The USB standard allows many peripherals to be connected using a single, standardized interface. Proven over the years as a reliable, expandable, fast, low-cost, low-power, and hot-pluggable interface, USB is now ubiquitous in a wide variety of applications. This application note addresses one such application, a RAID boot disk in servers.

Among server applications, server virtualization is the trend in IT and has become the fastest growing segment of the server market. Server virtualization is the process of using software to partition a physical server into multiple virtual servers. Server virtualization requires a fast, reliable, and compact boot disk that is separated from the main server storage.

The EZ-USB FX3S is a RAID-on-Chip controller with integrated dual Secure Digital (SD) interfaces, which is a perfect fit for a low cost RAID Boot Disk. It not only provides the benefit and function of a SuperSpeed USB mass storage device, but also it manages storage redundancy for high reliability through its RAID-on-Chip functionality.

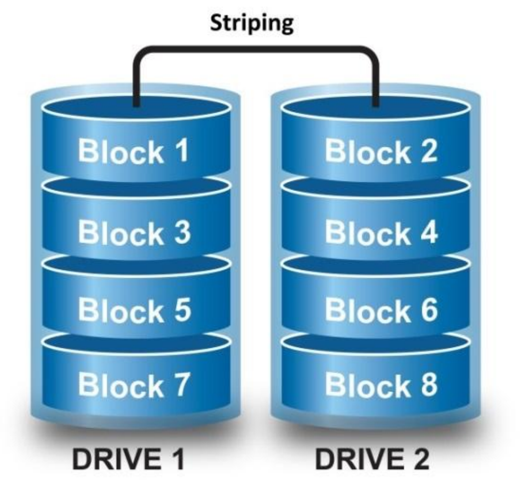
This application note presents an example design of a USB RAID 1 disk using EZ-USB FX3S for a server boot storage application. The RAID 1 functionality is managed entirely within FX3S. This application also may be used as a standalone mass storage device using two SD cards. A FX3S RAID-on-Chip dongle kit can be purchased off the shelf. The complete source code is available in the appendix of this application note.

2. Introduction to RAID

RAID is a storage technology that combines multiple storage disk components into a logical unit. Data is distributed across the disks in one of several ways, called “RAID levels,” depending on the amount of redundancy and performance required. The different schemes or architectures are named “RAID level,” or simply just “RAID,” followed by a number. FX3S supports RAID 0 and RAID 1.

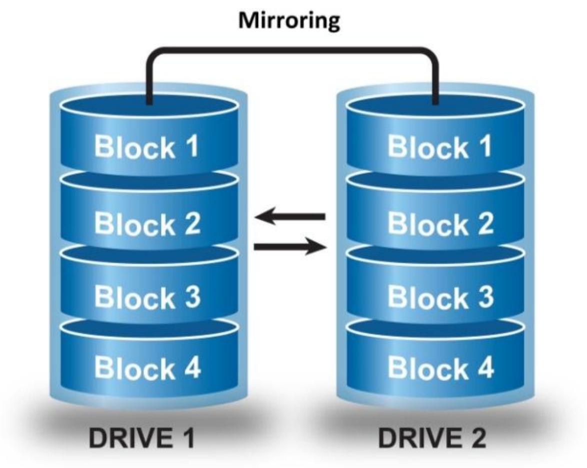
RAID 0 (block-level striping) has no redundancy. Because it interleaves data between two physical storage components, it provides improved performance but no fault tolerance. Any disk failure destroys the storage array. Typically, the RAID 0 architecture supports two disks, as shown in Figure 1.

Figure 1. RAID Level 0



For RAID 1 (mirroring), data is written identically to two disks, producing a “mirrored set,” as shown in Figure 2. Either of the two disks containing the requested data services the read request. A write request updates the stripes of both disks. At least two disks are required to create such an array.

Figure 2. RAID Level 1



3. System Overview

Figure 3 is a high-level block diagram of a server that uses a RAID boot disk. Since the advent of server virtualization, a single physical server has been used to host multiple virtual servers using virtualization software. The virtualization software typically resides on a boot disk that is separate from the main storage in a server. To increase reliability, a RAID 1 configuration of this boot disk is required. This RAID 1 boot disk, illustrated in the blue box in Figure 3, can be implemented using the EZ-USB FX3S RAID-on-Chip USB dongle, shown in Figure 4.

The FX3S USB RAID 1 disk is the initial boot source for the main server system. With the fast USB 3.0 link to the Platform Controller Hub (PCH), the server system can boot up quickly. A typical server system also uses a separate control link to the Board Management Controller (BMC) to manage the RAID operations.

The FX3S RAID-on-Chip dongle provides a separate slave MultiMediaCard (MMC) interface for the BMC connection. Because the BMC control link protocol is vendor specific, it is not covered in this application note. However, you can easily add and customize the feature in the FX3S firmware. Please create a technical support case from cypress.com/go/support to work out the implementation details of a specific protocol.

The example firmware in this application note implements a generic RAID 1 solution without the BMC control.

The EZ-USB FX3S RAID-on-Chip dongle hardware shown in Figure 4 is manufactured by Pactron and is available from pactronstore.com/products/cypress-fx3s.html.

The hardware schematic and example firmware source code are included in the appendix of this application note.

Figure 3. Server Overview

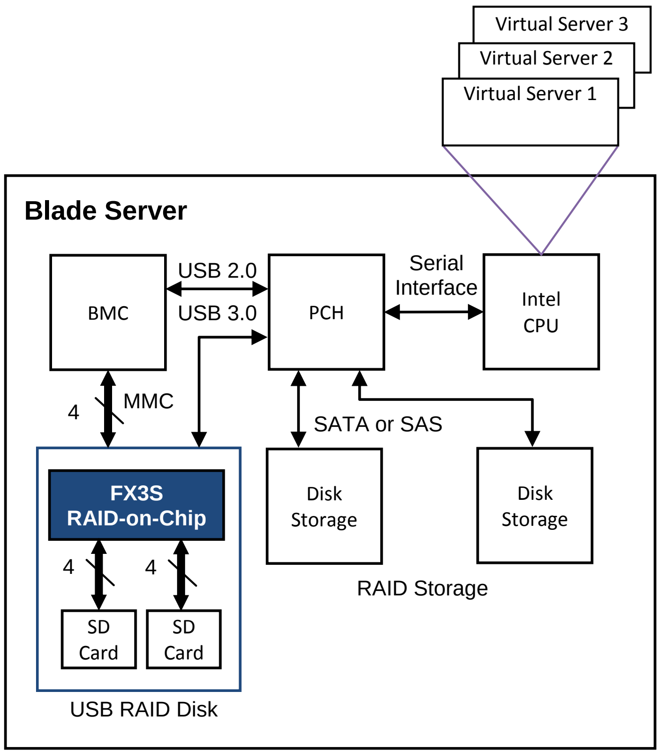
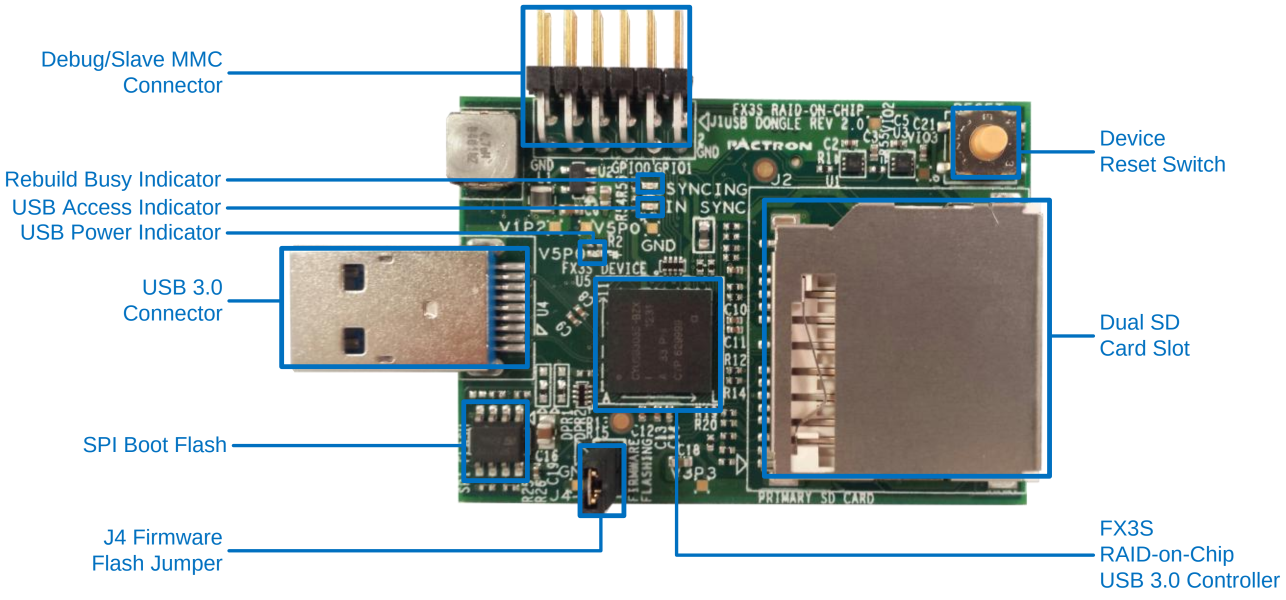


Figure 4. EZ-USB FX3S RAID-on-Chip USB Dongle



4. Functional Overview

The FX3S USB RAID 1 disk functions as a standard USB 3.0 mass storage device. The firmware example provided in this application note implements the RAID 1 function using two SD cards. The device enumerates as a single, mirrored storage volume on the USB host PC. All RAID operations are managed inside FX3S. No explicit external RAID handling is required by the user.

FX3S recognizes two SD cards: a primary card and a secondary card. For a read operation, data returned to the USB host is read from the primary card. If the primary card fails or is ejected, read operations continue from the secondary card. For a write operation, data from the USB host is simultaneously written to both the primary and secondary cards. If one card fails or is ejected, the transaction is aborted only to this card without interruptions to the other card. In both cases, FX3S maintains the seamless USB disk operations whenever an error condition occurs on one of the two cards.

The RAID 1 firmware example supports automatic volume rebuild when either the primary or secondary card is replaced. If one card is ejected while the disk is operational, the USB host sees no interruptions. If a new card is inserted, the RAID 1 firmware triggers the rebuild process by copying the entire contents of the active card to the newly inserted card.

During the rebuild process, the LED rebuild busy indicator, as shown in Figure 4, remains on. Storage access from the USB host to the SD cards is blocked until the rebuild process completes. The USB host sees a temporary read/write request denial and keeps trying until the rebuild process completes. The host operating system does this automatically, requiring no user intervention.

In summary, the FX3S USB RAID 1 disk does the following:

- Operates as a standard mass storage device at USB SuperSpeed, High Speed, and Full Speed.
- Uses two SD 3.0 (UHS-I) cards
- Reports and uses the smaller capacity if the two SD cards are of different size.
- Performs data mirroring during normal mass storage operations
- Provides uninterrupted access to one SD card if the other SD card is removed
- Performs data synchronization automatically when one card is replaced
- Handles SD card hot-plug events (removal and insertion)

5. Hardware

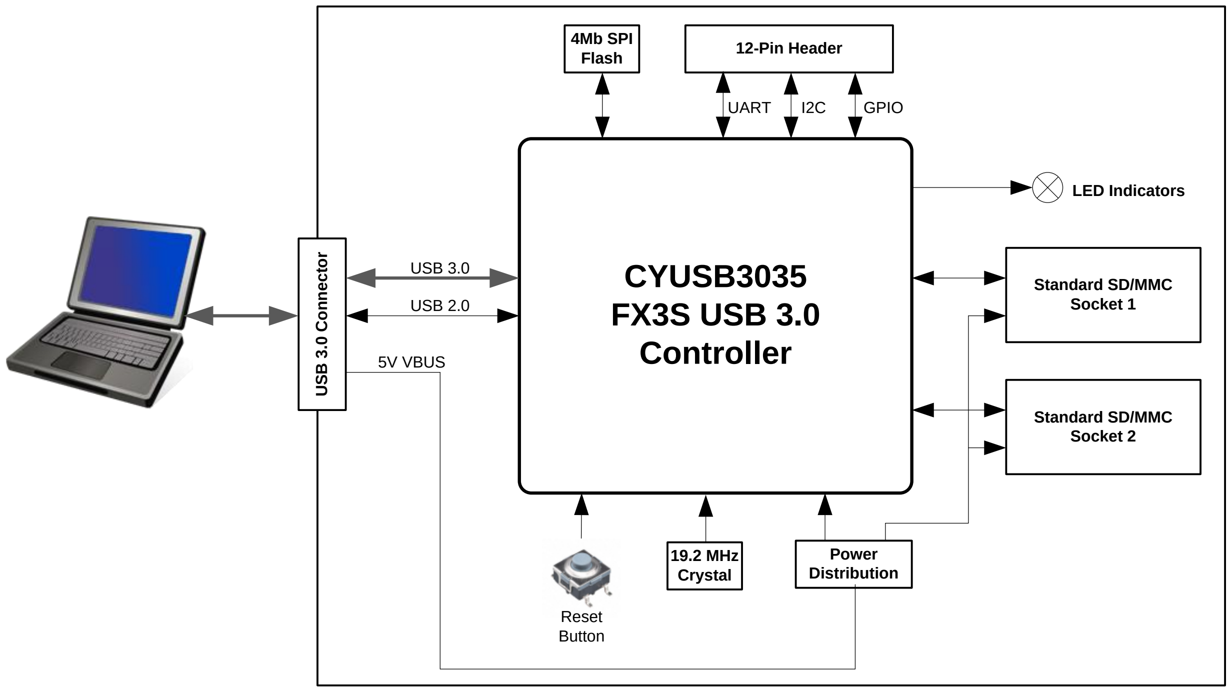
The USB RAID-on-Chip dongle hardware design shown in Figure 5, uses FX3S to implement a SuperSpeed USB mass storage device with built-in RAID management. The board features these components:

- USB 3.0 peripheral interface supporting SuperSpeed, High Speed, and Full Speed
- Dual independent storage ports, each supporting SD/SDIO 3.0 (UHS-I) and embedded MMC 4.41 devices
- 4-Mb SPI flash for FX3S firmware storage
- 19.2-MHz crystal input as the system clock source
- 12-pin header that includes UART for debug and a MMC 4.2 interface (not used in the RAID 1 example presented in this application note)
- LED indicators for power, USB access and rebuild operation
- Hardware system reset button

The RAID-on-Chip dongle is bus powered via the USB interface. All FX3S and SD card power rails are derived from the 5-V VBUS provided by the USB host.

FX3S features a 200-MHz ARM9 MCU, 512KB of system memory, and a full array of peripherals well suited to this application. Although the board is set to boot load its firmware from an onboard SPI flash device, you can also configure the board to boot load over the USB for system development.

Figure 5. Hardware Overview



6. Firmware

The firmware example project supplied with this application note is located in the FX3MSC_RAID1 folder of the source zip file. This example project compiles and generates a binary image without modifications. You can download the binary image to FX3S using the Cypress USB Control Center. Before downloading, make sure J4 of the board is open to allow FX3S to come up in bootloader mode.

Refer to AN75705 - Getting Started with EZ-USB FX3 for more details on how to work with the FX3S firmware projects and download image using the Cypress USB Control Center.

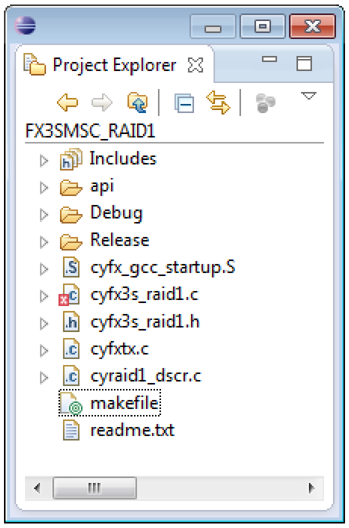
6.1 Firmware Source

The RAID 1 firmware application runs on top of a Real-Time Operating System (RTOS), which efficiently manages FX3S internal resources. The FX3S firmware application communicates with the FX3S hardware peripherals via a set of application programming interface (API) library functions that abstract low-level device details, significantly simplifying the development effort.

Figure 6 illustrates the files and directories included in the example RAID 1 firmware project:

- Includes: A directory that contains a collection of C library header and project-specific header files.
- api: A directory that contains the FX3S API library binaries.
- Debug and Release: The project can be compiled with either debug or release configurations. Debug and Release directories contain the output of the respective compilation configurations.
- *cyfx_gcc_startup.S*: The startup code for the ARM9 core on the FX3S device. This assembly source file follows the syntax of the GNU assembler. GNU, self-referentially, short for GNU’s not UNIX, is a UNIX-compatible software system developed by the Free Software Foundation (FSF).
- *cyfx3s_raid1.c*: The main implementation for the RAID 1 USB mass storage device example.
- *cyfx3s_raid1.h*: Constant definitions and function declarations for the RAID 1 mass storage application.
- *cyfxtx.c*: ThreadX RTOS wrappers and utility functions required by the FX3S API library.
- *cyraid1_descr.c*: USB descriptor definitions.
- *makefile*: A shell command file that manages and organizes code compilation.
- *readme.txt*: A simple description of the FX3MSC_RAID1 project.

Figure 6. RAID 1 Firmware Source



All RAID functions are implemented within the *cyfx3s_raid1.c* file. Table 1 summarizes these functions.

Table 1. Functions in cyfx3s_raid1.c Implementing the RAID 1 Application

Function Category	Reference Section	Function Name	Description
RTOS	6.2	main()	This function is the firmware C code entry point after the ARM core and C library initialization. It sets up caches, configures the FX3 I/Os, and starts the RTOS kernel.
	6.2	CyFxApplicationDefine()	This function defines the application thread that is executed by the RTOS.
	6.2, 6.3, 6.4	MscAppThread_Entry()	This function is the RAID 1 application thread that does the following: Calls the application-level initializations Manages the internal state transitions for the mass storage device
Mass storage application-level initialization	6.3	CyFxMscApplnInit()	This function calls the FX3S device-level initialization functions to perform the following tasks: 1. Initialize the GPIOs. 2. Initialize two SD storage ports. 3. Initialize the USB port. 4. Register event callbacks. 5. Register the USB descriptors. 6. Initialize the required DMA channels. 7. Enable the USB connection to host.
	6.3	CyFxMscApplnDebugInit()	This function initializes the FX3 UART block for printing debug messages.
	6.3	CyFxMscApplnDmaInit ()	This function allocates buffer memory and initializes the required DMA channels between USB and the storage port. In a normal condition, when two SD cards are attached, two DMA configurations are initialized: For read operation: one-to-one single channel DMA connecting the USB BULK-IN endpoint and primary SD card For write operation: one-to-two multicast channel DMA connecting the USB BULK-OUT endpoint to both SD cards
	6.3	CyFxMscApplnDmaReInit ()	When one of the two SD cards goes offline and comes back online again, this is the function to reconfigure the one-to-two multicast channel DMA.
FX3S device-level initialization	6.3	CyFxRdDmaInit()	For read operation, this function creates a one-to-one single-channel DMA connecting the USB BULK-IN endpoint and primary SD card.
	6.3	CyFxWrErrDmaInit()	When one of the two SD cards goes offline, this is the function to call and reconfigure the multicast channel DMA into single-channel DMA.
	6.3	CyFxMscApplnSibInit()	This function initializes the two-SD storage controller.
	6.3	CyFxMscApplnGpioInit ()	This function initializes the GPIO required for the application.
Mass storage application callbacks	6.2	CyFxMscApplnUSBEventCB()	This callback function handles general USB events (such as reset, connect and disconnect, and so on).
	6.2	CyFxMscApplnUSBSetupCB()	This callback function handles USB setup events during enumeration and control transfers.
	6.2	CyFxMscApplnSibCB()	This callback function handles storage port events (such as card insert and remove, data transfer complete, and so on).
	6.2	CyFxMscApplnDmaCb()	This callback function handles one-to-one single-channel DMA events.
	6.2	CyFxMscApplnMultiDmaCb()	This callback function handles one-to-two multicast-channel DMA events.
Mass storage application operations	Not referenced	CyFxMscApplnResetCtlr()	This function resets the clear the USB data path.
	6.3	CyFxMscApplnParseCbw()	This function parses the received storage command and handles it accordingly.
	Not referenced	CyFxMscApplnSendDataToHost()	This function stages the data to the BULK-OUT endpoint for the host to pick up.
	Not referenced	CyFxMscApplnSendCsw()	This function stages the storage command execution status to the BULK-OUT endpoint for the host to pick up.
	6.4	CyFxMscApplnQueryDevStatus()	This function probes the two SD storage ports and reports their statuses.
	Not referenced	CyFxAppLedOn()	This function sets the GPIO to turn on the LED.
		CyFxAppLedOff()	This function sets the GPIO to turn off the LED.

6.2 Callback Operations

Because the RTOS manages the system resources, RAID 1 firmware execution is event driven. Peripherals (USB and SD ports) and the internal DMA system generate events. These events are handled by callback functions registered during firmware initialization.

When RAID 1 firmware execution starts, it performs a series of initialization sequences for the ARM9 core, GNU tool chain library, and RTOS before entering the `main()` function in *cyfx3s_raid1.c*. The RTOS begins by calling `CyU3PKernelEntry()` from the `main()`. Before the RTOS starts the thread scheduling, at least one thread is created to perform the application task. For the RAID 1 example project, the application thread is `MscAppThread_Entry()`, which starts by configuring the peripheral interfaces and registering the event callback functions to handle USB, storage, and DMA events. These callback functions include the following:

- `CyFxMscApplnUSBEventCB:` This callback function handles these USB events:
  - *`CY_U3P_USB_EVENT_SUSPEND`*: The USB suspend event. The device goes into a low power mode.
  - *`CY_U3P_USB_EVENT_DISCONNECT`*: The USB disconnect event. The device is disconnected from the host.
  - *`CY_U3P_USB_EVENT_RESET`*: The USB reset event. A reset is received from the host.
  - *`CY_U3P_USB_EVENT_CONNECT`*: The USB connect event. The device is connected to the host.
  - *`CY_U3P_USB_EVENT_SETCONF`*: The host “configures” the device to complete the enumeration process.
- `CyFxMscApplnUSBSetupCB:` This callback function handles the following USB setup events, which occur mainly during enumeration:
  - `CY_FX_MSC_USB_STANDARD_REQ`: A standard setup request is received from the host.
  - `CY_FX_MSC_USB_CLASS_REQ`: A class-specific request is received from the host.
  - `CY_FX_MSC_USB_VENDOR_REQ`: A vendor-specific request is received from the host. This is a placeholder in case a custom vendor host driver is used.
- `CyFxMscApplnSibCB:` This callback function handles the following storage port events:
  - *`CY_U3P_SIB_EVENT_XFER_CPLT`*: A DMA transfer has completed.
  - *`CY_U3P_SIB_EVENT_INSERT`*: A card insertion is detected from one of the two SD sockets.
  - *`CY_U3P_SIB_EVENT_REMOVE`*: A card removal is detected from one of the two SD sockets.
  - *`CY_U3P_SIB_EVENT_DATA_ERROR`*: An error has occurred on one of the two cards.
  - *`CY_U3P_SIB_EVENT_ABORT`*: A read/write operation has been aborted.
- `CyFxMscApplnDmaCb:` This callback function handles events for one-to-one DMA channels. In a one-to-one channel, there is one data provider and one data consumer. These events usually occur when receiving a mass storage Command Block Wrapper (CBW) over USB, sending a mass-storage Command Status Wrapper (CSW) over the USB, or reading from one storage port.
  - *`CY_U3P_DMA_CB_SEND_CPLT:`* A DMA outgoing data transfer has completed.
  - *`CY_U3P_DMA_CB_RECV_CPLT:`* A DMA incoming data transfer has completed.
- `CyFxMscApplnMultiDmaCb:` This callback function handles the following events for a one-to-many DMA channel. In a one-to-many transfer, there is one data provider and multiple data consumers—in this case, the two SD cards. These events usually occur during a storage write command when data from the USB needs to be written into both storage cards.
  - *`CY_U3P_DMA_CB_RECV_CPLT:`* A DMA transfer with outgoing data has been completed.
  - *`CY_U3P_DMA_CB_PROD_EVENT`*: A DMA transfer with incoming data has been completed.

6.3 Mass Storage Operations

After the RAID 1 firmware completes the device-level and application-level initialization sequence, execution continues in `MscAppThread_Entry()` and stays in an infinite loop waiting for USB mass storage application events raised by the device.

The RAID 1 firmware application implements the standard USB mass-storage class (MSC) using bulk-only transport (BOT). BOT defines three basic stages of the MSC operation:

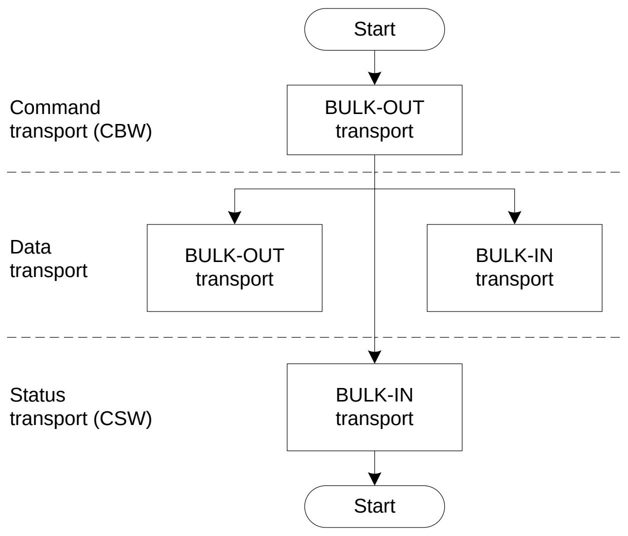
- Command transport: In command transport, the host PC sends commands to the device using BULK-OUT transfers. This command packet is defined as the CBW, and BOT must always begin with the CBW.

- Data transport: Data transport is used to transfer data between the host and the device. For example, with the read/write command, the actual data of the various storage sectors is sent using data transport, which utilizes multiple bus transactions. Data transfers carried out using data transport use either BULK-OUT or BULK-IN transport.

- Status transport: Status transport is used to send the results of command execution from the device back to the host using BULK-IN transport. The CSW defines this status packet. BOT must always end with the CSW.

Figure 7 shows the high-level MSC stage transitions.

Figure 7. USB MSC Transfer Stages



To support the standard MSC protocol, the following events are handled within the `MscAppThread_Entry()`:

- *`CY_FX_MSC_RESET_EVENT`*: The mass storage device has been reset.
- *`CY_FX_MSC_SETCONF_EVENT`*: Enumeration has completed. Need to set up the device for MSC operation.
- *`CY_FX_MSC_CBW_EVENT`*: A MSC CBW package has been received. The command (using SCSI format) is parsed and executed when this event is received.
- *`CY_FX_MSC_DATASENT_EVENT`*: At the data transport stage, data has been sent to the host, and firmware will move the device to the status transport stage.
- *`CY_FX_MSC_SIBCB_EVENT`*: This event requires the mass storage device to set up the storage port for the next data transaction.
  - If the device is at the command transport stage, the firmware sets up a receive buffer to accept a CBW package from the host.
  - If the device is at the data transport stage, the firmware sets up a transmit or receive buffer depending on the data direction of the current command.
  - If the device is at the status transport stage, firmware updates the CSW package and sends it to the host.

The RAID 1 firmware manages the mass storage device function via an internal state machine. States are defined for each of the MSC transfer stages shown in Figure 7. The following C code example shows how these states are defined in the firmware:

```
typedef enum
{
    /*    Inactive    state,    waiting    for
SET_CONFIG. */
    CY_FX_MSC_STATE_INACTIVE = 0,

    /* Waiting to queue a CBW command. */
    CY_FX_MSC_STATE_CBW,

    /* transitional state that it tells USB
host it is ready to receive a CBW command.
*/
    CY_FX_MSC_STATE_WAITING,

    /* Waiting to complete data transfer for
a command. */
    CY_FX_MSC_STATE_DATA,

    /* Waiting to send CSW for a command. */
    CY_FX_MSC_STATE_STATUS,

    /*  Waiting  for  host  to  read  out  CSW
packet. */
    CY_FX_MSC_STATE_CSW
} CyFxMscFuncState;
```

When the FX3S USB RAID disk is enumerated, the USB host sends the CBW package to the BULK-OUT endpoint. FX3S saves the CBW data in a buffer: `glMscCbwBuffer`. When the device is ready to execute a command, the current device state changes to `CY_FX_MSC_STATE_CBW` and an MSC CBW event `CY_FX_MSC_CBW_EVENT_FLAG` is raised. The device is set to the `CY_FX_MSC_STATE_CBW` state after a successful USB enumeration, or after successful completion of the last command.

After `MscAppThread_Entry()` detects the `CY_FX_MSC_CBW_EVENT_FLAG` event, it calls `CyFxMscApplnParseCbw()` to parse the command stored in `glMscCbwBuffer`. Depending on the command type, it moves the device state to `CY_FX_MSC_STATE_DATA` or `CY_FX_MSC_STATE_STATUS`.

If the command is `CY_FX_MSC_SCSI_READ_10` or `CY_FX_MSC_SCSI_WRITE_10`, a storage callback event `CY_FX_MSC_SIBCB_EVENT_FLAG` is raised. The `MscAppThread_Entry()` function handles this event by setting up the proper DMA channel between the SD card and the USB endpoint. When the USB host initiates the transfer, data flows through the DMA channel to or from the SD card until the data count is reached. When a DMA transfer completes, one of the two DMA callback functions receives a transfer completion event, raising the MSC data sent event `CY_FX_MSC_DATASENT_EVENT_FLAG`. After the

`MscAppThread_Entry()` function handles the MSC data sent event, the command status is updated. The device state is also moved to `CY_FX_MSC_STATE_STATUS`.

When the device is in `CY_FX_MSC_STATE_STATUS` state, a MSC CSW packet is constructed and sent back to the USB host. Once the CSW is sent, the device state is transitioned back to `CY_FX_MSC_STATE_CBW`, and the device is ready to execute the next command.

6.4 Automatic Rebuild

The RAID 1 firmware supports automatic volume rebuild to restore storage redundancy. `MscAppThread_Entry()` manages the volume rebuild by monitoring two status flags: stale status and card presence, one set for each storage.

The stale status flag is set whenever the card fails or is removed from the socket. It is cleared when both cards are operational and synchronized, usually after a successful rebuild operation. The stale status is also set by default after an initial power up with two SD cards inserted. The firmware always assumes the two SD cards are synchronized under the initial power-up condition. It is recommended that you format the disk after initial power up to ensure both cards are synchronized at the start.

The card presence flag is updated when a card insertion (`CY_U3P_SIB_EVENT_INSERT`) or removal (`CY_U3P_SIB_EVENT_REMOVE`) event occurs. These events are handled within the storage callback function `CyFxMscApplnSibCB()`. If the card is removed, the card presence flag is cleared. When the card is inserted, the `CY_U3P_SIB_EVENT_INSERT` event is raised and `CyFxMscApplnSibCB()` calls `CyFxMscApplnQueryDevStatus()` to detect and initialize the card. The card presence flag is set if the card is successfully initialized.

If `MscAppThread_Entry()` detects that the card presence and stale status flags are both set, it initiates the volume rebuild process by calling `CyWbRaidFullCardSync()`. The volume rebuild process continues to run until the stale status is cleared. The stale status is cleared when redundancy is restored after the full content of the SD card has been successfully transferred to the newly inserted card. During the rebuild process, disk access on the USB host may become unavailable because the primary SD card is busy servicing the rebuild process. In this case, the host continually retries disk access until the rebuild operation completes and the disk is again accessible.

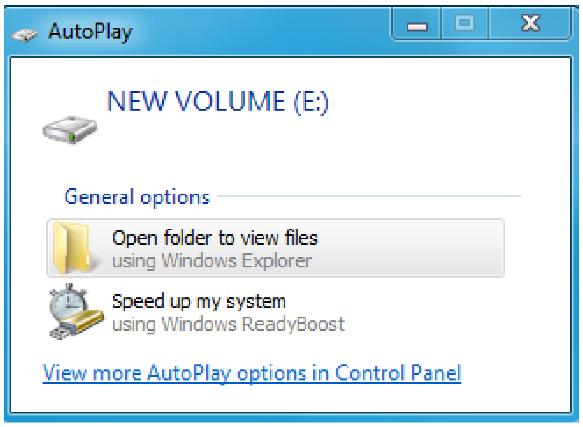
7. Operating Procedure

The following operating procedure can be used to test the supported functions of the FX3S USB RAID disk:

1. Plug the FX3S USB RAID disk into any USB slot (SuperSpeed, High Speed, or Full Speed). Notice that the LED power indicator turns on. Any ongoing USB access to SD cards also blinks the LED USB access indicator. Figure 4 shows the locations of these LED indicators.

2. After the USB enumeration, a disk drive in the Windows Explorer AutoPlay window appears as shown in Figure 8.

Figure 8. Windows Explorer AutoPlay Window



3. Format the disk using the FAT or FAT32 file system. Notice that it appears as a single disk volume, hiding the fact that it uses two mirrored SD cards.

4. Move a video file into the card, and then play the video.

5. While the video plays, remove one of the SD cards. Observe that the video continues playing without interruption. This demonstrates the safety feature of the RAID design – if one card fails or is removed the system continues operating with the single card.

6. Delete the contents of the SD card removed in step 4 and reinsert the card. This triggers a rebuild operation. The LED rebuild busy indicator shown in Figure 4 turns on and PC access to the card is inhibited. The video may play a bit longer due to internal buffering, but eventually it stops.

7. When the rebuild operation completes, the LED rebuild busy indicator turns off and the video resumes playing. This demonstrates that a rebuild operation requires no user intervention.

8. Remove the same SD card as in step 4 and check the contents. It should be a copy of the SD card that is still connected to the FX3S RAID-on-Chip USB Dongle kit.

8. Performance

With the RAID 1 firmware running on the EZ-USB FX3S RAID-on-Chip USB boot disk, performance presented in this section was measured under the following conditions:

- Measurement tool: CrystalDiskMark v3.0.2 x64
- SD card type: SanDisk Extreme Pro 8 GB
- USB host: Lenovo ThinkPad T430 with Intel Ivy integrated USB 3.0 host running Windows 7 x64

In general, it is recommended to use identical SD cards to achieve optimal performance. If the two cards are of different size and speed, the smaller size is used for the logical RAID 1 disk volume. Performance is also limited by the slower card.

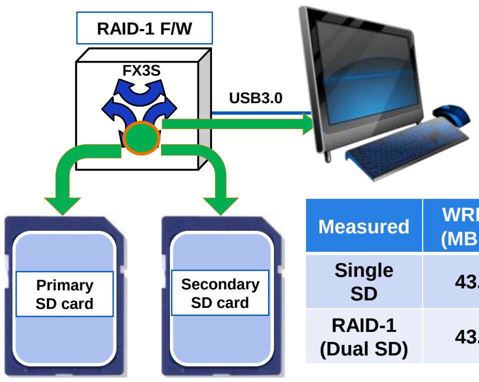
8.1 USB⇔SD Card

USB to SD card transfer performance is measured by reading and writing the drive from the USB 3.0 host using a popular disk benchmark tool, CrystalDiskMark. The tool is available for download from

crystalmark.info/software/CrystalDiskMark/index-e.html.

Figure 9 shows the expected USB⇔SD card performance.

Figure 9. USB 3.0⇔SD Performance



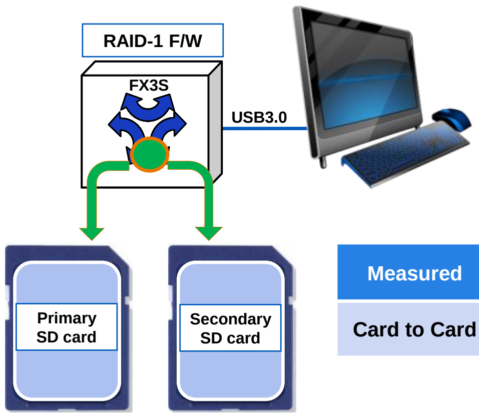
Measured	WRITE (MBps)	READ (MBps)
Single SD	43.2	43.6
RAID-1 (Dual SD)	43.7	43.7

8.2 SD Card⇔SD Card

SD card to SD card transfers are initiated by removing and then reinserting one of the SD cards. This triggers a rebuild operation in which the entire contents of the active card are transferred to the newly inserted card.

Figure 10 shows the expected SD card ⇔ SD card performance.

Figure 10. SD⇔SD Performance



Measured	WRITE (MBps)
Card to Card	43.5

9. Summary

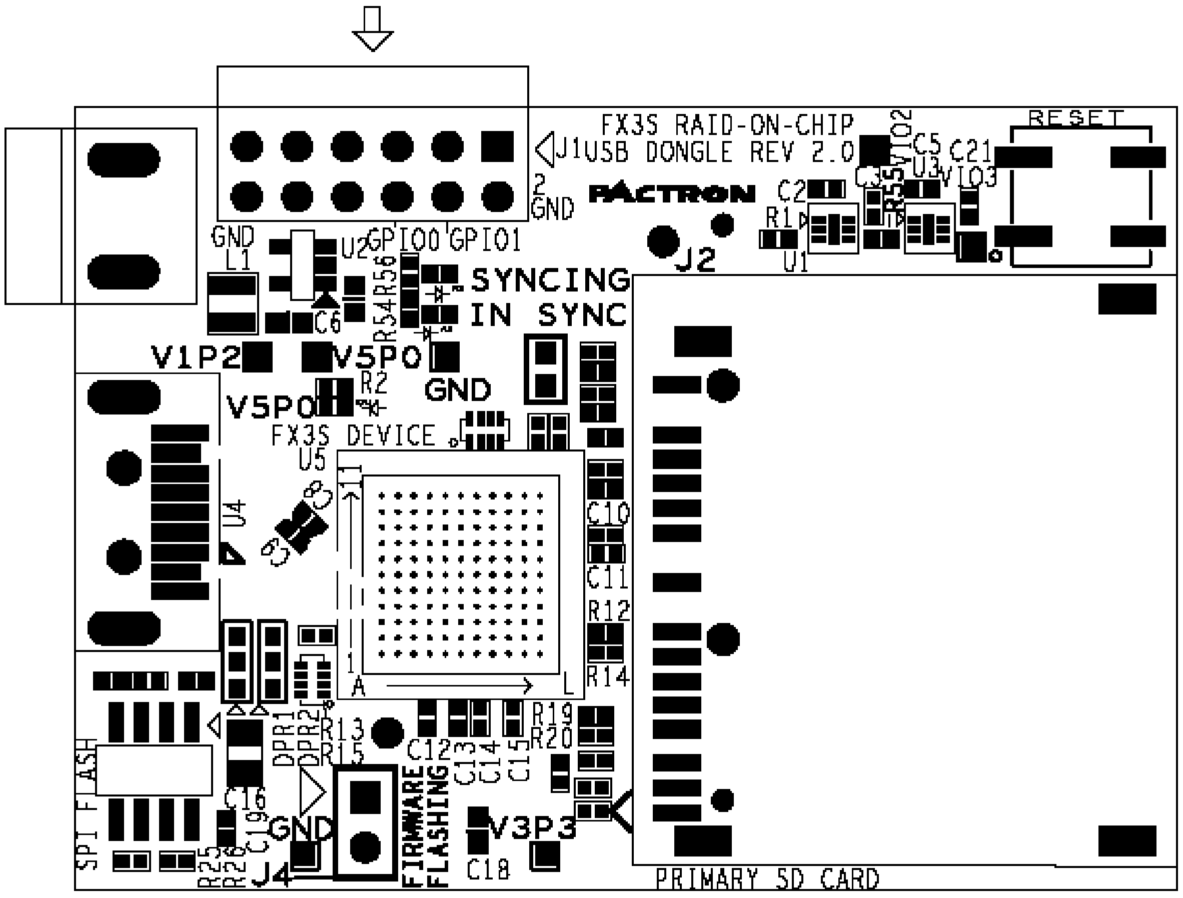
This application note demonstrated a RAID-on-Chip USB mass storage device using EZ-USB FX3S for server boot storage applications. Included in this document are associated board design schematics and example firmware source code.

10. About the Author

Name:	Hingkwan Huen
Title:	Systems Engineer Principal

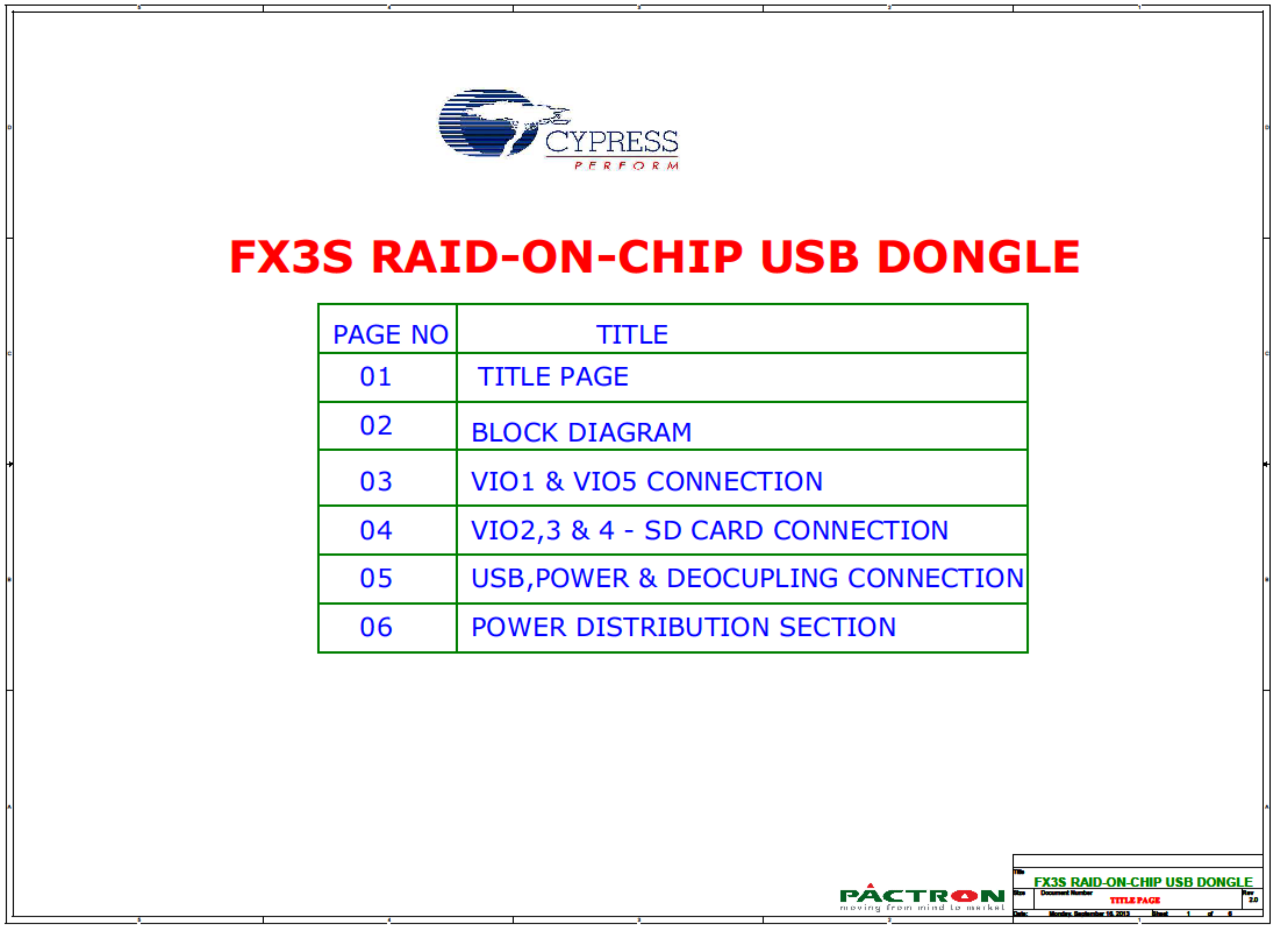
Appendix A: FX3S RAID-on-Chip USB Boot Disk Board Layout View

Double-click this image to open top and bottom views of the board layout.



Appendix B: FX3S RAID-on-Chip USB Boot Disk Schematic

Double-click this image to open the board schematic file.



FX3S RAID-ON-CHIP USB DONGLE

PAGE NO	TITLE
01	TITLE PAGE
02	BLOCK DIAGRAM
03	VIO1 & VIO5 CONNECTION
04	VIO2,3 & 4 - SD CARD CONNECTION
05	USB,POWER & DEOCUPLING CONNECTION
06	POWER DISTRIBUTION SECTION

PACTRON moving from mind to market

Title: FX3S RAID-ON-CHIP USB DONGLE
Size | Document Number: TITLE PAGE | Rev: 2.0
Date: Monday, September 16, 2013 | Sheet 1 of 6

Document History

Document Title: AN89661 - USB RAID 1 Disk Design Using EZ-USB® FX3S™

Document Number: 001-89661

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4176619	HKH	10/29/2013	New Spec.
*A	4228644	HKH	12/21/2013	No change to spec itself. Removed a typo from the example project source.
*B	5687998	AESATMP8	04/19/2017	Updated logo and Copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709