USB3DevIP Data Recorder by FAT32 Design

Rev1.1 13-Mar-15

1 Introduction

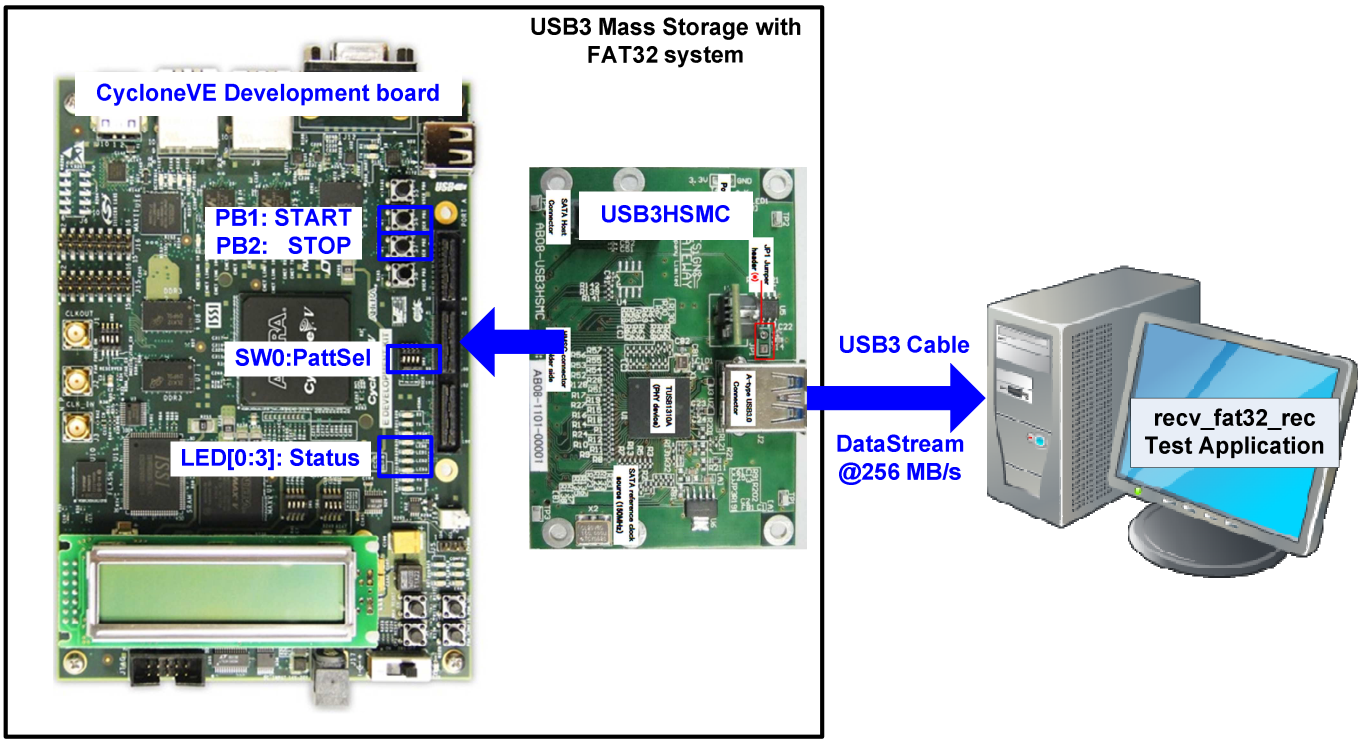
USB3 Mass Storage with FAT32 system
CycloneVE Development board
PB1: START
PB2: STOP
SW0:PattSel
LED[0:3]: Status
USB3HSMC
USB3 Cable
DataStream @256 MB/s
recv_fat32_rec Test Application

Figure 1 FAT32 Data Recorder Hardware on CycloneVE board

The demo system implements USB3 Device IP to be USB3 Mass storage device by using DDR3 RAM on the board to be data storage. The design is specially designed for 24hrs-run data recording system through USB3 interface. To compatible with standard system, record data is stored in FAT32 file system, so PC can design test application by using standard open/read file to dump the record data. In the demo, "recv_fat32_rec.exe" test application is designed to dump record data with/without data verification.

The hardware system is designed by using CycloneVE/ArriaV GX development board which has 512/256 MB DDR3 RAM to be storage. USB3 interface is implemented by using USB3HSMC adapter board to connect FPGA board to PC in device mode.

The demo uses 256/512 MB DDR3 RAM available on FPGA board to be record data storage. Special firmware is implemented on CPU within FPGA to emulate DDR3 RAM to be 32 GB storage size with FAT32 file system. When plugged-in the system to PC, PC will detect 32 GB storage with 947 available files in FAT32 system. 947 files consists of 945 record data files (F0000.BIN – F0944.BIN), and two control files (REC_STS.BIN, READ_STS.BIN). The feature of each file is follows.

- The record data file size is fixed to be 32 MB. Each file valid status is monitored through REC_STS.BIN.
- REC_STS.BIN is 4-byte size to show the latest file number which record data is valid. Maximum valid file depends on DDR3 size (7 files for 256 MB and 15 files for 512 MB). Valid value of the latest file number is 0-944 and default value after system boot-up or stop recording is 0xFFFF_FFFF. For example, if value is 200, it means that F0186.BIN/F0194.BIN – F0200.BIN are valid. This file is updated by CPU on FPGA board, and PC can read only.
- READ_STS.BIN is 4-byte size to show the latest file number which PC has been read. Similar to REC_STS.BIN, valid value is 0-944 and default value after system boot-up or stop recording is 0xFFFF_FFFF. This file is updated by PC and FPGA board will read this file to flush the record data from DDR3. For example, if value is 200, it means that F0186.BIN/F0194.BIN – F0200.BIN have been already read. CPU on FPGA board can flush these data from DDR3 and replace new data to DDR3.

To read control files which are 4-byte size, test application on PC (“rec_fat32_rec.exe”) is implemented with special setting to avoid read cache. It needs to use direct access to see the updated value from the hardware in REC_STS.BIN. For record data, total file size (32 MBx945) is bigger than cache size, so the test application can read the updated record data without special setting. Test application must read data in a sequence starting from F0000.BIN to F0945.BIN, and then restart to read F0000.BIN to dump the data continuously.

2 Hardware Structure

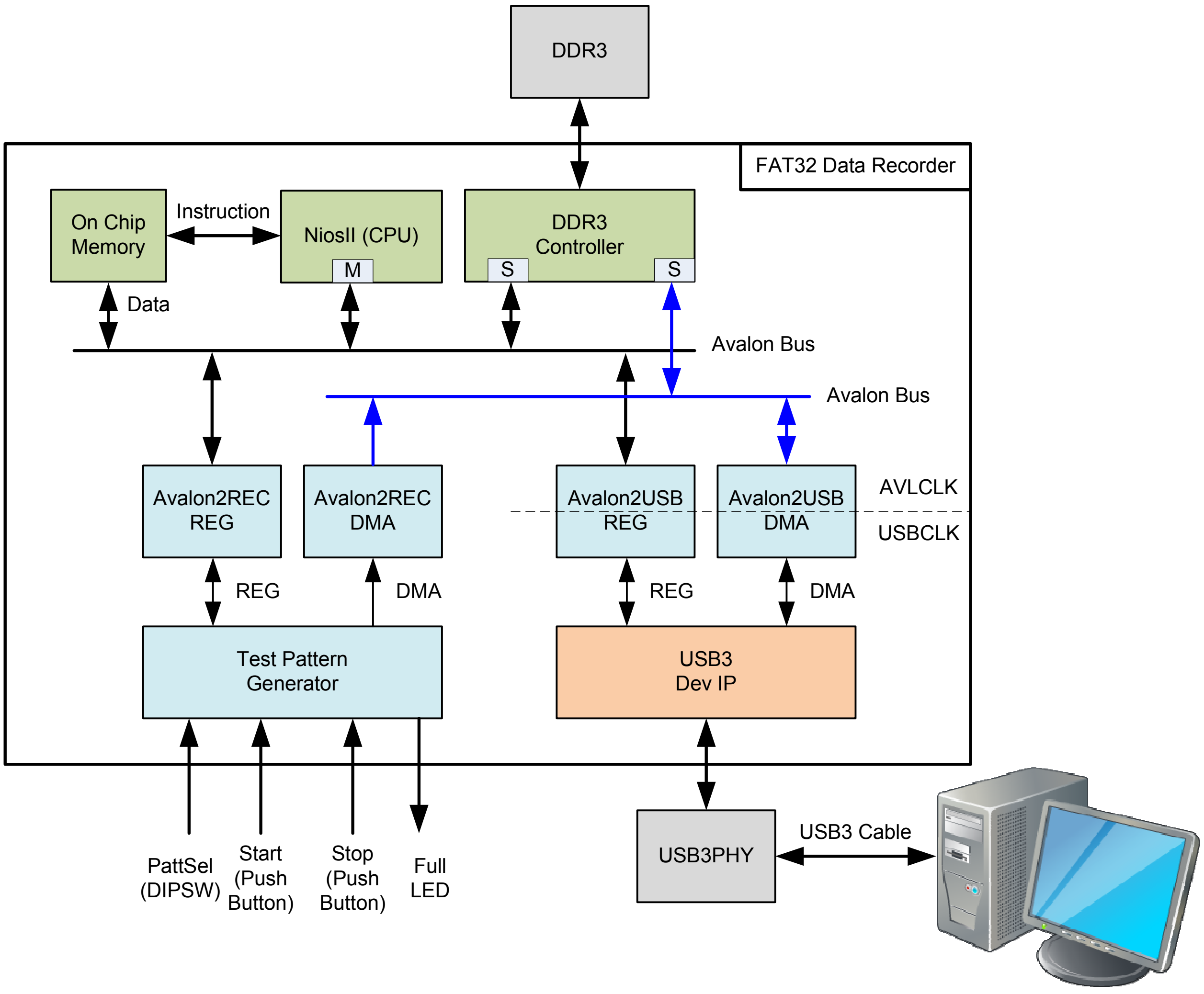

Figure 2 Block Diagram of FAT32 Data Recorder system

Record data stream at 256 MB/sec is generated by Test Pattern Generator. Two patterns can be selected through PattSel SW, i.e. 32-bit increment data, and 32-bit decrement data. Two push buttons are controlled by user to start and stop data stream generating. If PC cannot read and flush data in time, buffer will be overflow and Full LED will be ON to show error status. Test pattern generator will stop the operation immediately after overflow.

Avalon2REC_REG is interface with CPU to check control and status of record system such as Start DDR3 address, the available/space area in DDR3, and pattern selection/busy status of test pattern generator module.

Avalon2REC_DMA is DMA engine module to upload data stream from FIFO within test pattern generator to DDR3 by fixed burst size at 512-byte transfer. Available space in DDR3 and available numbers of data in FIFO of test pattern generator must be monitored before uploading.

To build USB3 mass storage, USB3 Device IP is designed in Link layer and protocol layer. Avalon2USB_REG and Avalon2USB_DMA are additional blocks to convert register interface and data interface of USB3 Device IP to be Avalon bus interface for connecting to NiosII system. The upper layer is implemented by NiosII firmware.

To implement USB3 mass storage, NiosII firmware includes the function to decode CBW (Command Block Wrapper) from the host, and then create CSW (Command Status Wrapper) and parameter data to the host. The additional feature from standard demo to support FAT32 data recording is implemented by adding the function during initialization phase to include FAT32 header in DDR3. FAT32 header will emulate DDR3 to be 32 GB storage with 947 files. Also, during recording there is one function running to update value in REC_STS.BIN and monitor value in READ_STS.BIN. More details about each block are described as follows.

2.1 USB3 Block

a) USB Device IP
The details of IP has described in the USB3 Device IP datasheet. It shows the description of each interface, memory map for register interface, timing diagram of DMA engine, and the basic sequence of USB3 device. Data interface in the demo for USB3-IP is 32/64-bit@125 MHz which is enough for USB3 bandwidth (USB3 bandwidth = 5.0 Gbps). For USB3 mass storage function, three endpoints are used, i.e. EP0 (bi-directional) for control pipe, EPI#1for device to host transfer, and EPO#2 for host to device transfer.

b) Avalon2USB_REG
Avalon2USB_REG decodes Avalon address from CPU and the convert into enable signal in one-hot format for interfacing with USB3 Device IP. Asynchronous circuit is also included to transfer the signal between AVLCLK (Avalon bus) and USBCLK (USB3-IP) domain.

c) Avaon2USB_DMA
Avalon2USB_DMA converts 32/64-bit DMA interface of USB3-IP which runs at USBCLK into 64/128-bit Avalon master interface which runs at USBCLK. So, asynchronous FIFO must be used to support data burst transfer with clock-crossing feature, and bus size conversion in case of different bus size. State machine is designed to control DMA sequence of both interfaces. There are two DMA burst sizes in the module, i.e. 256-byte and 1024-byte. 1024-byte is used only for M2U (MemorytoUSB) direction with big size transfer while other cases use 256-byte transfer.

2.2 Test Pattern Block

a) Test Pattern Generator
After Start button is pressed, Test pattern generator creates 32-bit increment/decrement data stream (depending on PattSel input) at 256 MB/s. 32 kB FIFO is included to be temporary buffer before uploading to DDR3. By using start/stop button, user can control record time. Test pattern starts operation after start button is pressed immediately, but operation is stopped after stop button is pressed and current 32 MB data file is completed.

This block runs at 75 MHz clock, so 256 MB/s data stream is generated by 32-beat of 128-bit burst data in every 150 clock period (128-bit x 32 x 75M/150M = 256 MB/s). Burst size is fixed to align 512-byte similar to Avalon2REG_DMA burst size. So, all test pattern data in FIFO will be uploaded to DDR3 by DMA engine in Avalon2REG_DMA. FIFO status must be monitored before generating the next 512-byte data. If FIFO is full, test pattern generator will stop the operation and FullLED will be ON.

b) Avalon2REC_REG
Avalon2REC_REG is interface with CPU to write/read control/status signal of Test pattern generator operation. Register map of this module is shown in Table 1.

Only 256/512 MB is available to be record data buffer but total file size is 32 GB, so data in each file will share the same buffer area. At most 7/15 data file can be stored at the same time, so file number 0-944 can be converted to slot number 0-14/6 by modulo 15/7.

Table 1 Avalon2REC_REG Register Map

Offset (byte) Rd/Wr	Register Name	Description (Bit order is little endian)
0x0000_0000 [14:0] Wr/Rd	Record Slot Status (RECSLOTSTS_REG)	Record data file (FXXXX.BIN) status. Bit[0]-[6]/[14] are referred to file slot number. For example, in case of 15 slots Bit[0]: F0000.BIN, F0015.BIN, F0030.BIN, …, F0930.BIN status Bit[1]: F0001.BIN, F0016.BIN, F0031.BIN, …, F0931.BIN status Bit[13]: F0013.BIN, F0028.BIN, F0043.BIN, …, F0943.BIN status Bit[14]: F0014.BIN, F0029.BIN, F0044.BIN, …, F0944.BIN status Rd: ‘1’- File in slot is ready to read, ‘0’- File in slot is empty Wr: ‘1’-Clear File status from ready to empty status, ‘0’-No operation
0x0000_0004 [31:0] Wr/Rd	Start DMA Address (RECDMASTT_REG)	Wr: Start DMA address for F0000.BIN. Must align in sector unit. Bit[8:0] value will be ignored. Rd: Current DMA address in transferring (align in sector unit). Bit[8:0] are all 0s.
0x0000_0008 [31:0] Rd	Test Pattern Status (TSTPATSTS_REG)	Rd: Status flag of Test pattern generator module [0]-PattSel input value (‘0’-Increment, ‘1’-Decrement) [8]-Recording in process (‘1’-Recording, ‘0’-No operation) [31]-Overflow status (‘1’-Buffer overflow, ‘0’-Normal operation). This bit is auto-cleared when start new recording.

c) Avalon2REG_DMA

During recording, Avalon2REC_DMA monitors FIFO status and Record Slot Status register. At least 512-byte (one burst size) data must be available in FIFO to start each burst transaction. Also, before starting each burst, RECSLOTSTS_REG must be read to check that current slot and next slot are empty status. Next slot is also monitored to confirm that at least one slot will be in empty status. Empty status is necessary for CPU firmware because firmware must have stop position to break the process to update REC_STS.BIN. More details are described in NiosII topic.

If FIFO and slot are ready, DMA engine will send write request to Avalon bus and transfer all 512-byte data to DDR3. The flag in the slot will be updated to '1' when completing all 32 MB transfer of each file. Next, the slot number will be increment and then start data transfer for the next file. The operation runs in the loop of 32 MB transfer. The signal will be re-initialized to be default value when start after stop is detected. So, after stop and restart data recording, record data will be stored starting from F0000.BIN.

3 NiosII system

USB3 Mass storage protocol is implemented by NiosII firmware. In the main function, CPU checks interrupt flag status of the IP for both link and protocol layer, and call the function to run the service of each interrupt.

To support FAT32 file system, two additional functions are designed from standard firmware, i.e. FAT32 initialization which will run only one time after system boot-up and FAT32 update feature which will always run during data recording.

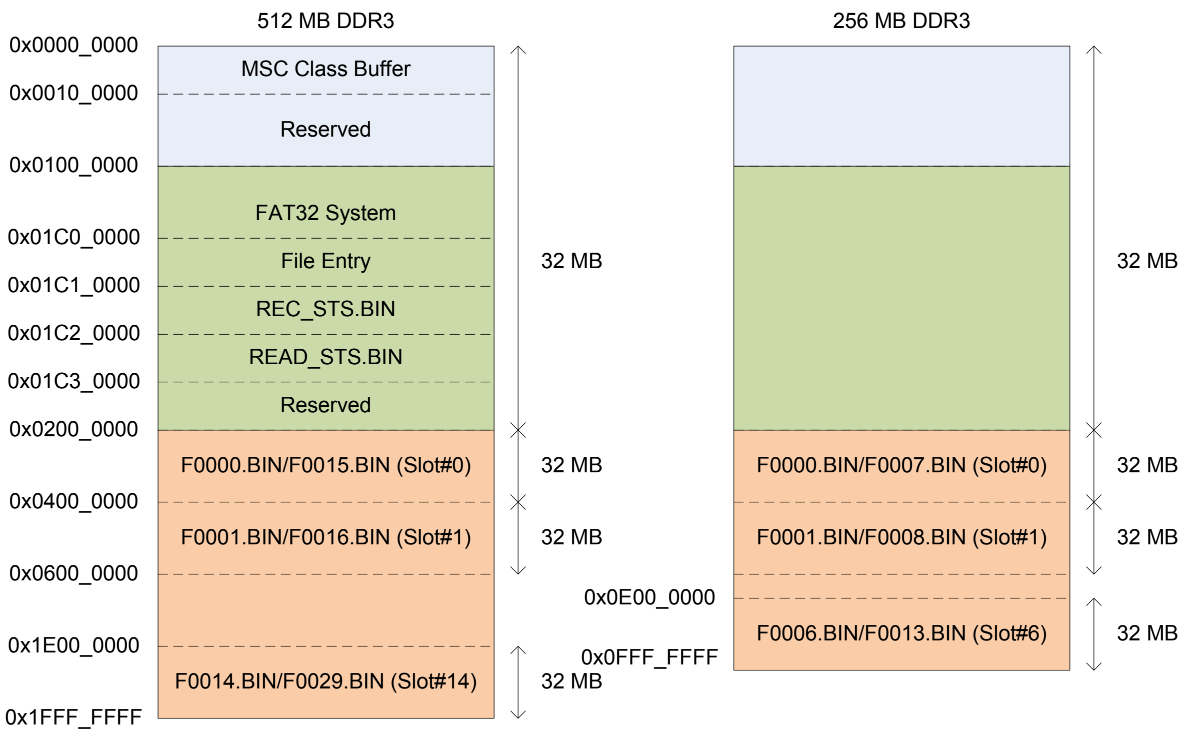

Figure 3 DDR3 Memory Map

FAT32 initialization function will run to create FAT32 system by setting value to DDR3 address at 0x0100_0000 – 0x1C3_0000 following Figure 3.

- FAT32 system area: Master boot record to show partition table in the storage, File system information to show drive name and size, and FAT table to show file allocation in the storage.
- File entry: Entry of root directory, two control files, and 946 record files to define file name, file attribute, modified date, start address of data file, and file size.
- Initial value of REC_STS.BIN and READ_STS.BIN to be 0xFFFF_FFFF

Since the system implements 32 GB storage by using only 256/512 MB DDR3 memory, file entry is special setting. Start address of data file will be set to the same value in every 7/15 data files. So, data in F0000.BIN, F0007/F0015.BIN, F0014/F0030.BIN will share the same data space.

Another additional function is FAT32 update. This function always runs during data recording. Three processes are designed in this function, i.e. updating record status, updating read status, and stop management.

3.1 Updating Record Status

This process monitors RECSLOTSTS_REG to check file available from hardware. If new file is available, the value in REC_STS.BIN will be updated. During operation, firmware will remember the latest record file number (RecFileNo), and convert file number to be next slot value for scanning by mod 7/15. Firmware sequence is follows.

1) Check RECSLOTSTS_REG [RecFileNo mod 7/15] value. Go to next step if read value = ‘1’, or end process if read value = ‘0’.
2) Update RecFileNo to REC_STS.BIN and then increase RecFileNo by 1. If current value is 944, next value must be 0.
3) Go back to step 1) for checking next slot status.

3.2 Updating Read Status

This process monitors the latest read file number (ReadFileNo) from READ_STS.BIN. If value is updated, RECSLOTSTS_REG will be set to flush data within DDR3. During operation, firmware will remember ReadFileNo which can convert to slot value by mod 7/15. Firmware sequence is follows.

1) Read READ_STS.BIN value and compare with the latest read file number. Go to next step if read value is not equal, or end process if equal.
2) Increase ReadFileNo by 1. If current value is 944, next value must be 0. Set RECSLOTSTS_REG [ReadFileNo mode 7/15] to ‘1’ to flush data in DDR3.
3) Go back to step 1) for checking next file status.

3.3 Stop management

This process monitors TSTPATSTS_REG[8] status to check that system has been stopped. READ_STS.BIN is also monitored to check that PC has already read all record files. Then, the value within REC_STS.BIN and READ_STS.BIN will be set to default value. Firmware sequence is follows.

1) Check TSTPATSTS_REG[8]=’0’ and the latest record file number (RecFileNo) and read file number (ReadFileNo) are same value which is not default value (0xFFFF_FFFF).
2) Reset RecFileNo, ReadFileNo, and the value within REC_STS.BIN and READ_STS.BIN to be default value (0xFFFF_FFFF).

4 Test Application

"recv_fat32_rec.exe" is designed to run on PC to dump record data from the hardware with/without data verification. Similar to CPU firmware, test application must remember the latest read file number to compare to the value in "REC_STS.BIN". If new file is detected, data in the new file will be dumped to the buffer which has 16x32 MB size.

Three parameters are inputs in test application, i.e. drive name of device, data verification ON/OFF, and pattern format (32-bit increment/decrement). The sequence of test application on PC is follows.

1) Read value from REC_STS.BIN and compare with the current read file number. If the value is not equal, go to next step. But if REC_STS.BIN value is equal to 0xFFFF_FFFF and the current read file number is not reset value, test application will be closed (detect stop condition on hardware).
2) Read FXXXX.BIN (xxxx is the file number) and verify it with expect pattern. Test application will stop operation with error message if data error is detected.
3) Increase file number from READ_STS.BIN by 1. If current value is 944 or 0xFFFF_FFFF, next value must be 0.
4) Update the value to READ_STS.BIN, and print file number to the console every second. So, file number will be increased about 8 in every second (256 MB = 8 x 32 MB file).

The test application is designed to run on Windows7 OS. To avoid read cache during read value from REC_STS.BIN, the file must be opened by CreateFile function with enable FILE_FLAG_NO_BUFFERING.

5 Revision History

Revision	Date	Description
1.0	27-Feb-15	Initial version release
1.1	13-Mar-15	Update demo details