VIDEO RECORDING IN A TACTICAL ENVIRONMENT

Alfred Rotundo, Jin Hwan Choi, Gilmer Vega
US Army – Army Futures Command
CCDC – Armament Center - Precision Munition Instrumentation Division
Picatinny Arsenal, NJ

Abstract

Imagers are being evaluated for their feasibility on munitions to record performance metrics and increase functionality. The Armaments Center is working to prove the practicality of such sensors by retrofitting an armament to evaluate its advantages. The instrumented projectile is also equipped with an independent guidance electronics unit (GEU) and optical seeker. A high-speed on board recorder is crucial in gathering both GEU navigation and guidance data and imager data. Having to record image files uncompressed and GEU guidance & navigation data simultaneously required a 40Mbps data rate, pushing USB 2.0 protocol to its max transfer rate. Using an FX3 USB Controller, custom firmware was developed to reduce typical overhead present on commercial memory devices to obtain rates suitable to record the necessary image frame rate (15 fps) to assess feasibility of a tactical seeker.

Introduction

The Precisions Munitions Instrumentation Division (PMID) – Telemetry Branch was tasked with designing an on board recorder (OBR) capable of recording multiple uncompressed video channels interlaced with other performance metrics. Unlike PMID's previous OBR designs that relied on a digital signal processor (DSP) to act as the main processor, a USB host controller was used to hit the desired theoretical speed of USB 2.0 at 40Mbps. Hitting the upper limit of USB 2.0 allowed the OBR to capture image data at a high enough frame rate for post process analysis of the seekers performance and algorithm accuracy. The following sections will go through how the PMID team tackled the design and the end product performance.

Electronics Design

1. **Selection of Chipset / USB Host Controller**

There were a multitude of factors that determined the processor that was selected. The program required a noise immune, fast data transfer rate interface due to the remote location of the OBR to each of the video processors inside the munition. Thus a USB interface was deemed to meet the performance specifications due to its noise resistant differential signal and high data transfer rates.

Many processors were taken into consideration, although ultimately, the Cypress FX3 series was chosen due to its flexibility, and previous experience. It comes with a slew of functions in a prepackaged API, has a fully configurable I/O system where it can interface to any processor, ASIC, image sensor or FPGA, and provides a host of standard interface options such as USB 2.0, USB 3.0, SPI, UART, I2S, I2C, etc. It also has intuitive software tools for development and running a critically time sensitive Real-Time Operating System (RTOS). There was also previous experience from the contractors that designed the video processors that were willing to lend a hand. These factors determined the ultimate outcome of chip selection, although there were still tasks unidentified to complete the design. Should we use USB 2.0 or USB 3.0? Should we consider power efficiency and cable length? Are the data rates fast enough for the application?

Although USB 3.0 was faster and was more power efficient, we decided to use USB 2.0 due to its lower pin count and reduced overhead. However, there were some extra hurdles that the team had to overcome - the host, video processor did not run a typical OS, and the video data speeds were close to the theoretical maximum of USB 2.0. Thus, we created proprietary code on the FX3 chip that directly communicated to the host processor with reduced overhead typically seen on the USB protocol. This allowed a more efficient and reliable transfer of data. Cypress's FX3 chip also came with prepackaged APIs, all handled internally on the RTOS, which allowed easy customization of the protocol, drastically reducing programming labor.

2. Selection of Memory Chip

Selecting the memory chip was a bit more straightforward. Our team required high capacity (min of 3GB of data) and very fast write speeds (order of 40MB/s). Choosing the Micron MT29F128G08AK NAND Flash 128 GB (16GB) memory chip allowed those specifications to be met. A NAND Flash chip had faster write speeds compared to a NOR Flash type, and its large page sizes allowed for more write periods without switching between planes, reducing the overhead that was required in changing states to switch between pages.

3. Interconnections Between Imager, FX3, and Flash Memory

With the USB Chipset and the Memory chip selected, the primary building blocks were in place to complete the overall architecture of the design. Because the Imager had two cameras running in parallel, the team decided that having independent USB host controllers, receiving each video feed via USB was the most efficient way of recording the data to memory. This is shown in Figure 01 below.

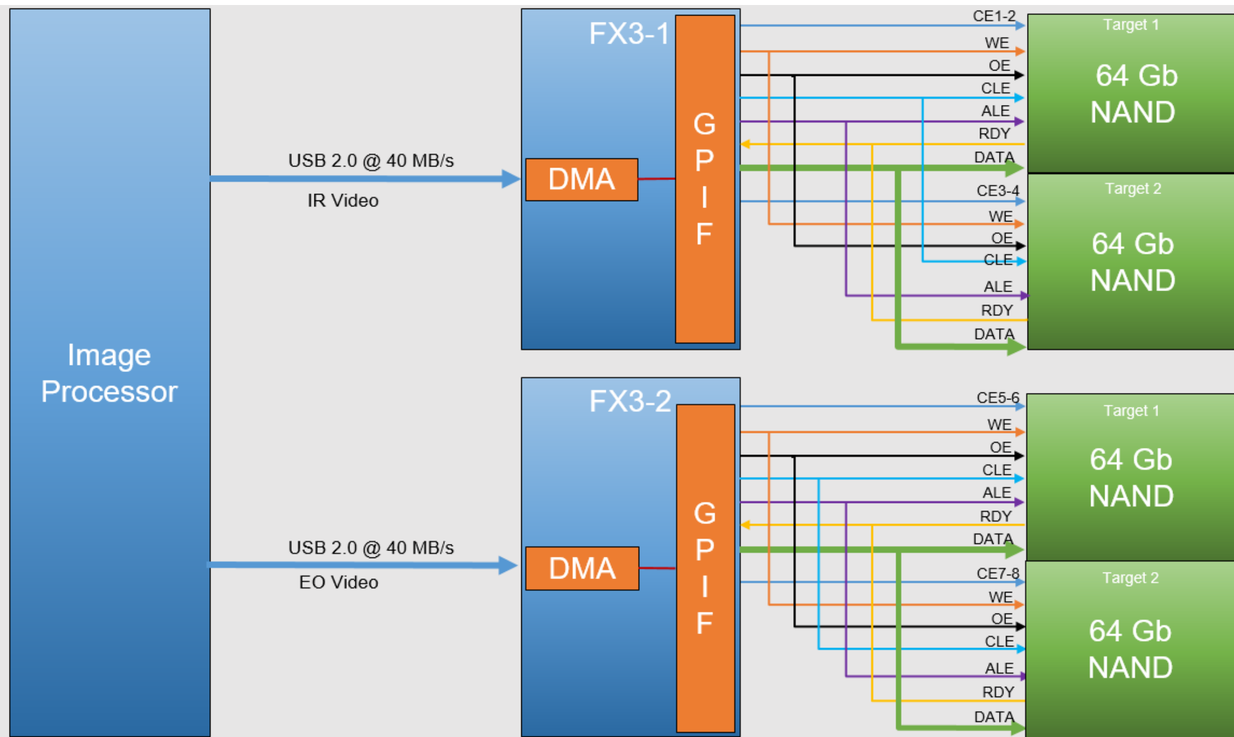

Figure 01: Block Diagram of Imager to FX3 to Flash Memory

The team required downloading of the data so using the same buses from the FX3 chip, while switch the FX3's function from slave to host allowed this to be possible. A switch was required to disconnect the interface from the Imager to a PC instead, using the same USB 2.0 interface.

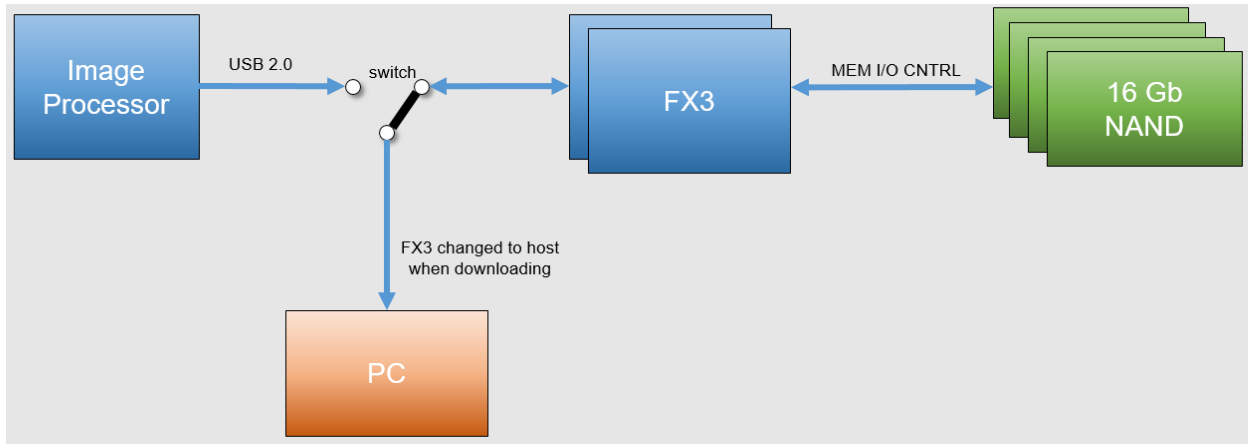

Figure 02: Block Diagram of OBR in User Download Mode

4. Board Design and Integration of System

Before the requirements were locked in, the number of high speed video channels were still evolving. At that time, it was decided to make each high-speed USB channel independent from one another and located on a separate printed circuit board (PCB). This would allow for any number of boards to be stacked, allowing for additional high speed lines if needed. This approach made the design more modular and thus more adaptable for future designs. Since each high speed record device is independent from one another, the system has an added benefit of being less susceptible to a complete system failure. In the event that one high-speed line fails the other remaining lines would not be affected or loaded with any system level cross-talk noise.

Each PCB contains its own USB host controller (Cypress, Manufacturer's Part Number: CYUSB3014), Flash NAND memory (Micron Technology, Manufacturer's Part Number: MT29F128G08AKCABH2-10ITZ: A), independent power regulation, clock, and logic circuits. Figure 03 below shows a 3D top view of one of the PCBs.

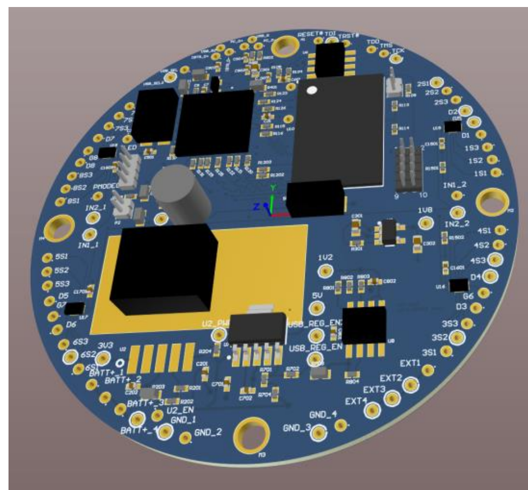

Figure 03: 3D view of PCB

The use of USB switches allowed us switch between a PC interface (to either set the mission parameters on the USB host controller and extracting the data off the recorder) and the image sensors for recording of the video data stream during the mission). The OBR was wired in such a way that the connection is defaulted to connect to the image sensors. When the USB is connected from the PC it automatically toggles the switch, allowing for seamless communication between the two interfaces.

Figure 04: USB switching logic for USB communication lines

Another part of the circuit that allows for multiple boards to be stacked while limiting the number of pins that need to be pulled out is the constellation of multiplexer integrated circuits. Using four 3:1 MUXs, it is possible to pull out one set of pins for all programming, reset, mode, and LED status pins. By using an interrogation box it is possible to switch between numerous FX3 USB host controllers to program, set the mode, reset, or check their status by toggling the MUXs using a set of two control lines. All MUX control lines are tied together so that when they are toggled all the pins are changed at once. Figure 05 shows the four sets of MUXs and how they are wired.

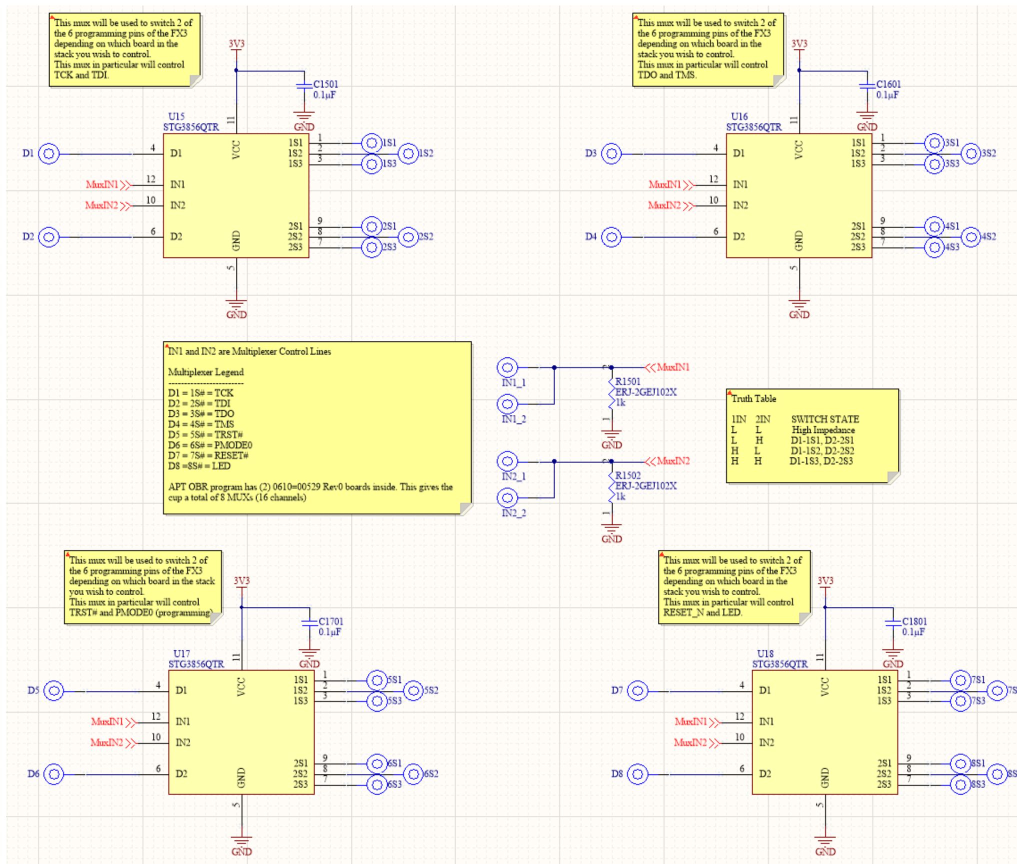

Figure 05: MUX controller to switch between multiple USB host controllers

PCB routing best practices were of paramount importance for this design due to the speed on the USB 2.0 data traces. Without the proper considerations the data being transferred over USB 2.0 can be corrupted, dropped, or arrive at different times from one another. In addition, impedance must be matched between the two differential data plus and data minus USB lines. USB 2.0 standards dictate a 90 Ohm impedance on both traces. This is achieved by being mindful of a few critical aspects of the design. These include but are not limited to the following: the width of each trace, the amount of copper on each trace (i.e. ounces of copper), the space between each differential trace, the layers above and below the trace layer with the impedance matched traces, the thickness of the dielectric between those layers, and the dielectric properties of said layer.

It is common practice for the designer, to set the thickness of the trace and the space between the differential traces. There are "toolboxes" inside of most CAD software packages to set these parameters to ensure both thickness and space between traces are consistent. Copper thickness, trace width, distance between differential traces, the layer(s) they are located on, the board layer stack up, and the desired impedance are relayed to the board house. The board house will select the FR4 material, considering its dielectric properties, and set the dielectric thickness to achieve the desired impedance.

The previous paragraph is the bare minimum required to create impedance matched differential traces. There are additional best practices that must be adhered to in order to maintain signal integrity for USB lines. The following are considerations for impedance matching and high-speed signal lines:

- Avoid any slit in the immediate planes above/below the USB lines.

- All high speed lines should be routed over a solid ground plane on an adjacent layer for a ground return path.
- When two pairs of USB traces cross each other in different layers, a ground layer should run all the way between the two USB signal lines.
- When transitioning USB lines between layers, place ground vias roughly 40mils from signal vias.
- Differential super speed pair trace lengths should be matched within 0.12mm (5mils). High speed D+ and D- signal trace lengths should be matched within 1.25mm (50mils)

Firmware Design

The Cypress code initializes the ARM CPU environment (MMU, VIC, core clocks, etc.), loads the program into RAM, Instruction Cache (IC), Data Cache (DC), including initializing all interrupt vectors, and then configures the runtime environment to call the main() function. At this point in time, the RTOS is not running so the CPU speed can be reconfigured and the Instruction and Data Caches enabled, if needed.

RTOS allocates required memory for the thread(s), which the application will create in order to manage the I/O blocks. Next, it will call CyFx_Application_Define() routine, where the thread(s) stack, priority, and preemption threshold are defined. This is shown in the flow diagram below in Figure 06.

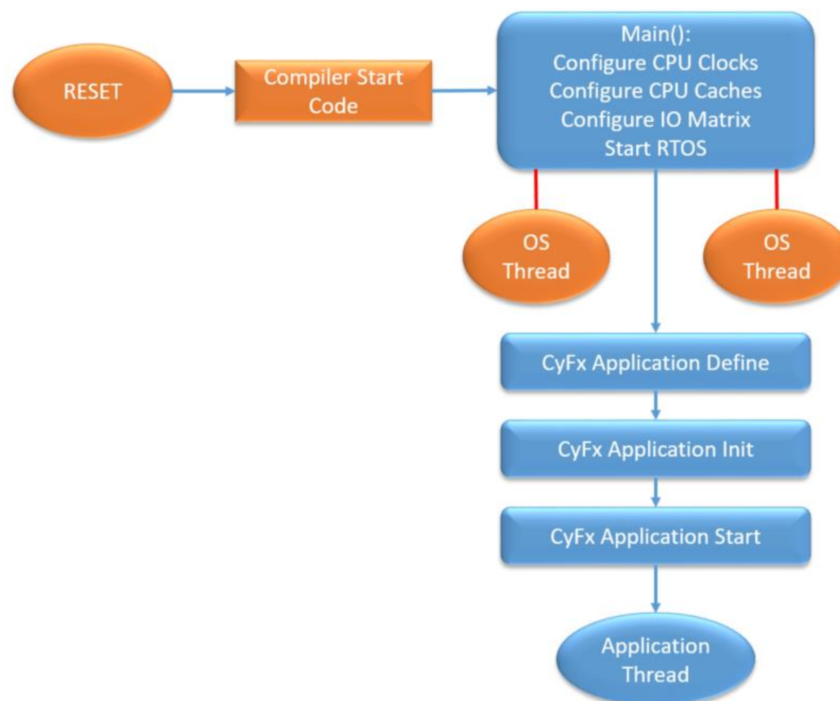

Figure 06: Firmware Flow Chart

The subsections to follow will breakdown the function and inner workings of each routine within the firmware design. Each routine is responsible for a critical function of the firmware and is critical for overall success and performance of the system.

CyFx_Application_Init():

This function initializes the USB interface, as well as, the GPIF interface, which was used to communicate with the NAND Flash Micron memory device. The firmware loads the GPIF configuration file, which includes connections on data and control lines between the FX3 Controller and the Micron NAND Flash device.

Fast enumeration is the easiest way to setup a USB connection since all enumeration phases are handled by the library. Only the class/vendor requests need to be handled by the application.

Since this application uses a USB 2.0 High Speed rate, the High Speed Device Descriptor must be initialized. Finally, the firmware needs to check if a USB connection is already active; if not, the USB bus and appropriate direct memory access (DMA) channels will need to be configured in the CyFx_Application_Start() function.

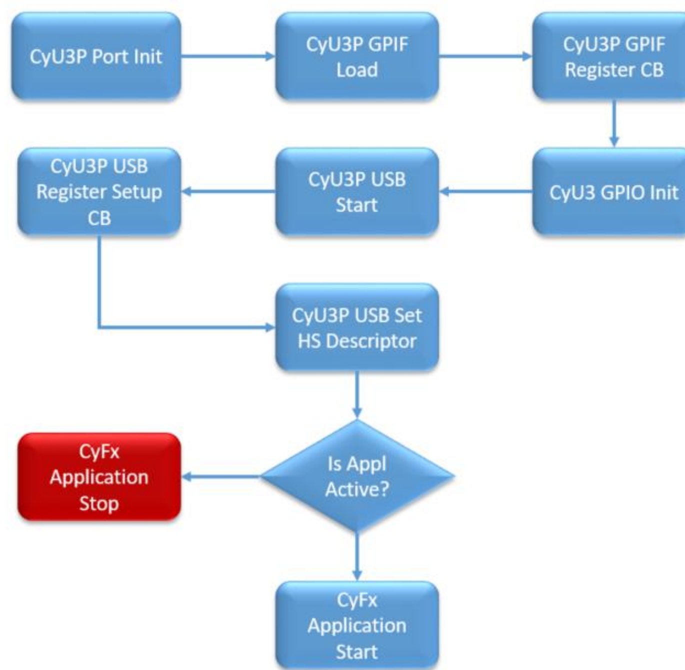

Figure 07:CyFx_Application_Init flow diagram

CyFx_Application_Start():

The firmware makes use of one of FX3's core features, a distributed DMA controller that is capable of moving data at 800 MBps. This DMA controller is attached to internal devices via sockets. A socket provides a consistent interface to the DMA controller side and is customized on the device such that all internal devices look like standard block input/output (I/O) devices.

For this application we made use of 2 DMA channels in AUTO mode, this way data is moved from producer socket(s) to consumer socket(s) while achieving its maximum throughput.

The firmware also utilizes a third DMA channel in MANUAL mode as there needs to be CPU involvement to issue commands to the GPIF interface, such as increment addresses or send trigger signals. This is represented in Figure 08.

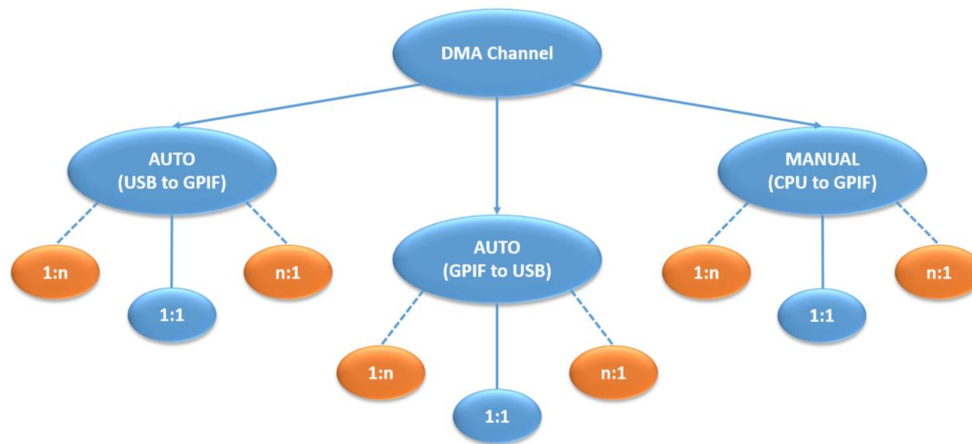

Figure 08: DMA Auto/Manual Channels

The CyFx_Application_Start() function first identifies the USB Speed. For this application, once the High Speed is identified, the USB Endpoint configuration handler defines a bulk transfer type and packet size. A producer and consumer endpoints are configured by calling the API CyU3PSetEPConfig() routine. As mentioned above, the application's DMA channels are created and set to transfer data. This was achieved by calling API DMA routines. The last block, starts the GPIF machine, which issues a reset command to both NAND Flash devices. This basically means that after a successful reset, the GPIF will send a series of bytes (0xFF) to both devices (targets) within a NAND Flash, then will send a Set Features command (0xEF) and configure the NAND Flash to support maximum write speeds when storing data.

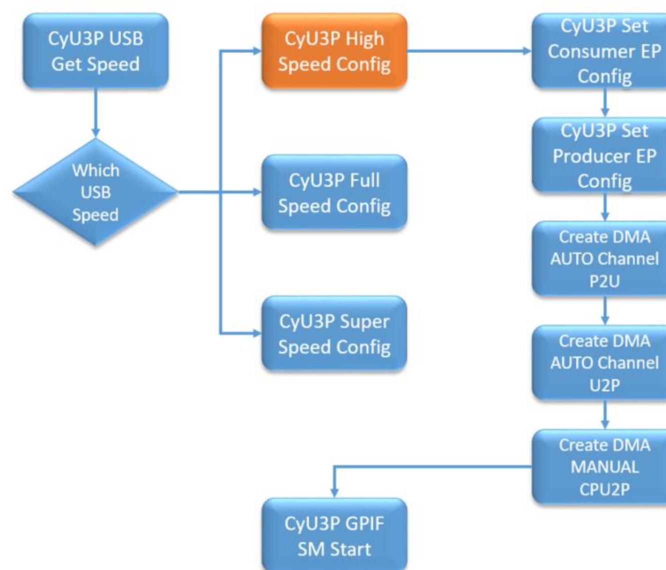

Figure 09: CyFx_Application_Start flow diagram

Application Thread:

The program established a well-defined interconnect control document (ICD), which lays out all of the commands the image processor needs to send to the FX3 in order to achieve a specific task. The table below explains the ICD in more detail

Definition	*Opcode Value*	*Data Size*	*Data Direction IN (Request), OUT (Command)*	*Setup Packet Data Field*
Scratchpad	0x0	1 to 64	IN, OUT	Write/Read 64 byte buffer in OBR FX3
Signature	0x1	11	IN	OBR FX3 returns: "APT OBR FX3"
FW Version	0x2	1	IN	Returns FW Version
OBR Status	0x3	1	IN	Returns FX3 State Machine Status 0 = Unknown, 1 = Ready, 2 = Busy, 3 = Erase 4 = Read, 5 = Write, 6 = OBR LB
Start Recording	0x4	N/A	OUT	FX3 switches to state machine, starts recording
USB Loopback	0x5	N/A	OUT	Data received at DMA CH2 is routed back to DMA CH0
Set LED Rate	0xAA	1	OUT	Set LED Duty Cycle (0 to 100)
Set Interrupt	0xBB	1	OUT	0: Disable, 1: Enable
Read First Page	0xC0	7	OUT	Read first page of each block + spare area
Read Pages	0xCC	10	OUT	Read specific number of pages
Program Pages	0xDD	5	OUT	Program specific number of pages
Erase Blocks	0xEE	7	OUT	Erase specific number of pages
Set Features	0xFF	15	OUT	Set Features for 1st and 2nd devices

Figure 10: OBR Interconnect Control Document

The function Application_USB_Callback() is invoked to handle USB setup requests. As it is shown in the below diagram (Figure 11), the APT firmware first decodes the fields from the setup request and initializes variables, such as the Type of Request and data direction, IN-to-host or OUT-of-host.

For simplicity, the three main USB Requests are the blocks highlighted in green: Read, Write, and Erase. If the USB setup request matches that of the write command, the OBR will go into OBR_Write State and both OBR FX3 devices will start recording data. The incoming USB data bulk transfers will be received on configured DMA buffers, having their byte sizes equal to that of the Micron NAND flash page.

As the DMA buffers get filled, the consumer socket will start emptying these buffers while the producer socket fills the next DMA buffers. The GPIF interface will then execute the Program State Machines and ultimately move these packets externally to NAND flash.

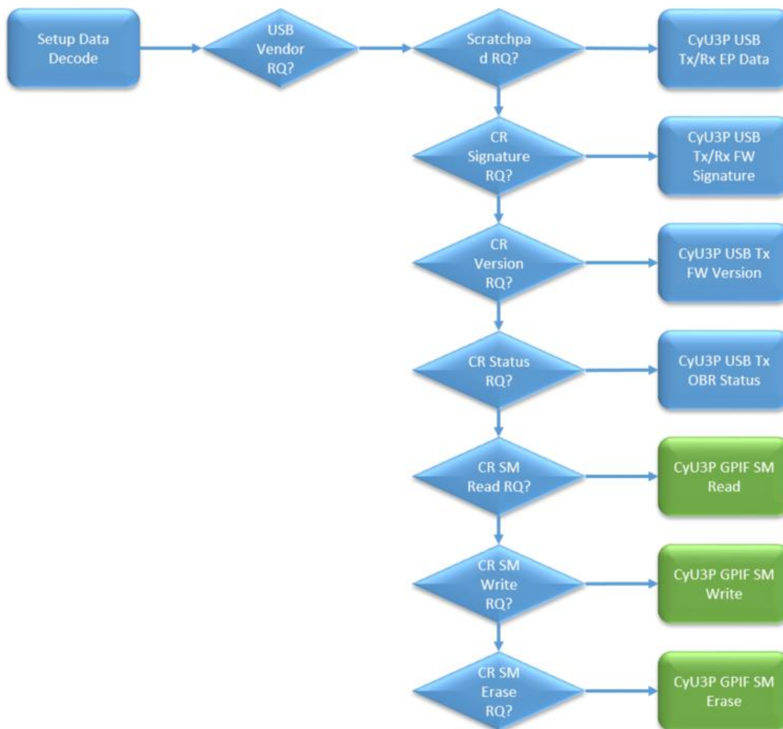

Figure 11: Application_USB_Callback flow diagram

If the USB setup request matches that of the Read command, data will be downloaded from NAND Flash onto the host via USB 2.0. This is achieved by running a Control Center executable Graphic User Interface (GUI) application on the host computer. As seen on Fig 12 below, various parameters can be configured on this GUI by the user.

The GUI is also able to display the number of bytes being read in real time, giving the user valuable status information of the current NAND block being serviced.

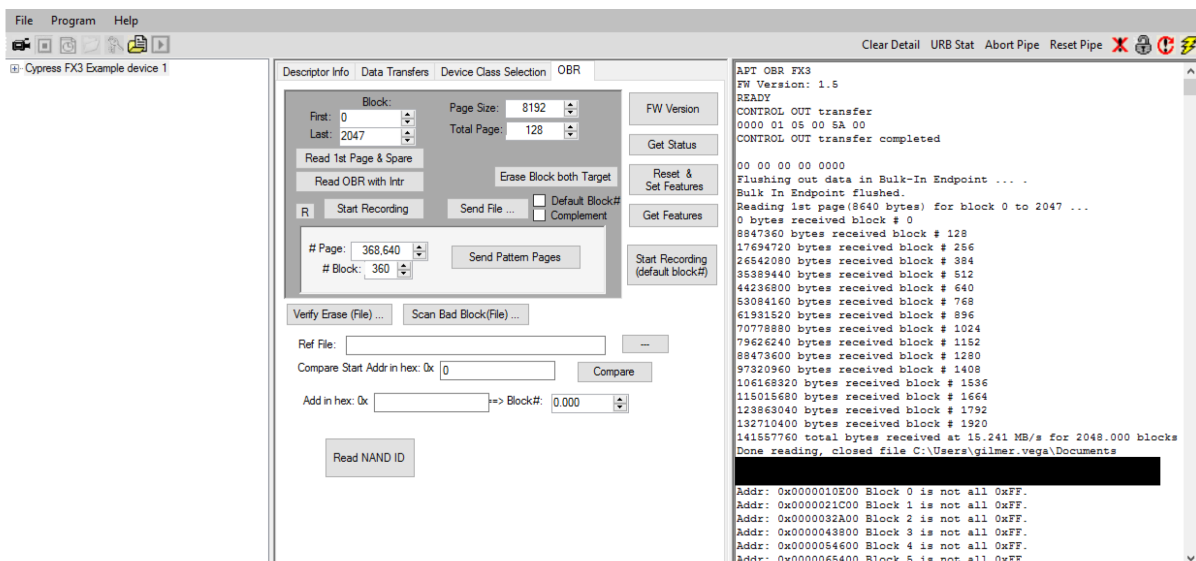

Figure 12: OBR Graphic User Interface

OBR Bad Block Management

Most NAND Flash devices, like all types of storage, include some initial bad blocks within the memory array. These blocks are typically marked as bad by the manufacturer, indicating that they should not be used in any system. If these factory-marked bad blocks are erased, their markings are also erased.

By using the APT OBR GUI, we are able to perform a read on the first page of all blocks in the NAND Flash memory, where we are able to see which bad blocks are defective and must be avoided. Also, because good blocks in a NAND Flash can degrade and wear out, it is important to track not only the initial factory-marked bad blocks but also the blocks that go bad during normal device operation.

Therefore, the APT OBR GUI implements a feature where a series a patterns are written onto the NAND Flash to then be read back and downloaded into a file. To detect additional bad blocks, the GUI compares the downloaded data to a fixed set of patterns, and if there are any mismatches, these blocks are also marked as bad.

Serial #	*Bad Blocks from Manufacturer*	*Bad Blocks on ECC 1 (0x55AA)*	*Bad Blocks on ECC 2 (0xAA55)*	***Total Bad Blocks***	*Pre-Underfill*	Metal Base - OBRs on Standoffs
104	13, 45, 65, 111, 136, 156, 271, 429, 552, 703, 719, 805, 835, 847, 849, 853, 1029, 1044, 1102, 1117, 1149, 1158, 1167, 1191, 1199, 1203, 1211, 1217, 1254, 1263, 1302, 1325, 1415, 1610, 1629, 1655, 1662, 1687, 1710, 1755, 1765, 1772, 1786, 1790, 1815, 1821, 1931, 1969, 2001, 2032		186	**57**	Pass	ECC1 (1st read): 111, 136, 156, 271, 429, 552 ECC2 (1st read): 111, 136, 156, ,186, 271, 429, 552
110					BAD NAND CHIP	
116	45, 277, 435, 667	265, 547	181	**7**	Pass	ECC1 (1st read): blocks 265, 277, 435, 547 ECC2 (1st read): 181, 277, 435
118	45	566		**2**	Pass	ECC1 (1st read): blocks 275, 367, 429, 566 ECC2 (1st read): 110, 170, 186, 364, 494
201	30, 45, 795, 905, 1396, 1828, 2041	475	100, 377	**10**	Pass	ECC1 (1st read): block 475 (18 bits) ECC2 (1st read): block 100, 377
203	45, 137, 300, 439, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 6280, 639, 821, 879, 961, 999, 1162, 1223, 1257, 1369, 1428, 1439, 1456, 1752, 1821		177	**48**	Pass	ECC1 (1st read): block 137, 300, 439 ECC2 (1st read): block 137, 177, 300, 439
205					BAD NAND CHIP	
208	45	120, 399	254, 526, 585	**6**	6	ECC1 (1st read): ECC2 (1st read): block 300, 367, 474, 482
210	45, 1198		471	**3**	Pass	ECC1 (1st read): block 466, 569 ECC2 (1st read): block 139, 229, 233, 251, 270, 284, 302, 406
211	45, 293, 600, 601, 602, 603, 648, 1263, 1299		175, 247, 444	**12**	Pass	ECC1 (1st read): block 293 (entire block) ECC2 (1st read): 175, 247, 444
216	45, 1398	100	229, 462, 512	**6**	Pass	ECC1 (1st read): block 100 ECC2 (1st read): 229, 462, 512
301	27, 45	335	129, 562	**5**	Pass	ECC1 (1st read): block 335 (entire block) ECC2 (1st read): 129, 562
310	45	350	290, 468	**4**	Pass	ECC1 (1st read): 350 ECC2 (1st read): 290, 468
311	45, 974, 1651, 1837			**4**	Pass	ECC1 (1st read): None ECC2 (1st read): None
316	45, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 1823	456, 572	306	**16**	Pass	ECC1 (1st read): 456, 572 ECC2 (1st read): 306
317	45, 78, 87, 96, 104, 110, 159, 202, 223, 230, 284, 293, 407, 459, 844, 893, 904, 906, 922, 927, 936, 964, 1234, 1335, 1339, 1345, 1363, 1435, 1451, 1502, 1571, 1609, 1695, 1711, 1713, 1730, 1874, 1931, 2019		187, 188, 329, 331, 356, 359, 565, 576, 590, 591, 592, 593, 594, 595, 596,	**53**	Not realiable system	ECC1 (1st read): 217, 258 ECC2 (1st read):

Figure 13: OBR Bad Block Management Example

Field Test / Performance

On October of 2019, CCDC-AC and partnering contractors demonstrated the functionality of the Electro-optical/Infrared (EO/IR) imager on a guided munition.

The objective was to demonstrate proof of concept of the imager hardware and software in a relevant munition environment. It was critical that the imager's performance was captured during flight, EO and IR video data were collected from the CCDC-AC PMID developed OBR, and structural integrity of the system was maintained through gun-launch and flight.

The OBRs were fired in a munition at approximately 9kgs for 5 shots. Out of the 5 rounds that were fired, only 4 were expected to record EO data, and only 3 were expected to record IR data. All OBRs successfully recorded video.

Some highlights from the tests are the following images captured by the Imager and OBR:

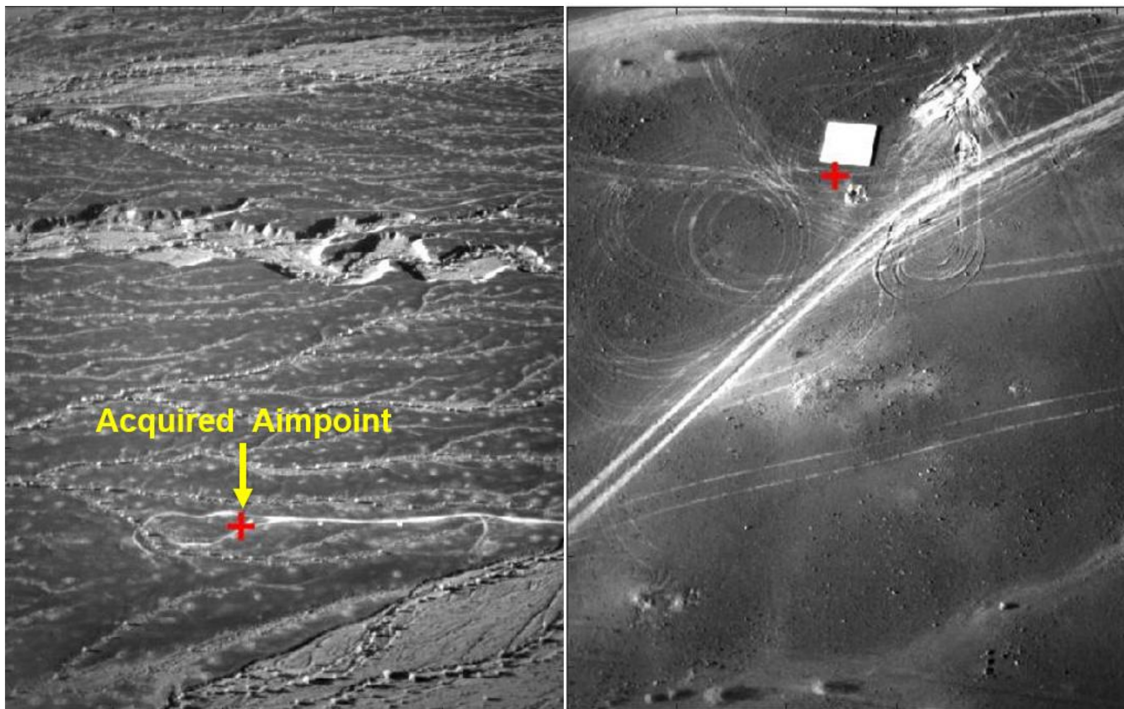

Figure 14: Resulting Captures from Imager and OBR

Future Applications

This program had a very niche, specific application where the host processor did not run a typical OS, and the data rates pushed the boundaries of the USB 2.0 protocol to its theoretical limit, with the preference for a minimum pin count. However, this program demonstrated the potential and flexibility of the FX3 chip, which allowed customization of standard protocols so that future applications that require any of these specific requirements can be met with a little bit of programming work to the processor, using its prepackaged API. Interfacing through standard and non-standard communications protocols, and its flexible GPIF interface allows this processor to be used in an extremely wide array of applications.