USB 3.0-based Super Speed Data Acquisition Systems

Project Guides:

Dr. Navin Kumar
HoD, Dept of ECE, ASEB

Ms. R Srividhya
Head, DSAS, DSD, PDMG

Mr. Srinidhi M
Engr SF, DSAS, DSD, PDMG

Mr. Om Sai Ayyappa Reddy
Sci/Engr-SD, DSAS, DSD, PDMG

Team:

Sathyasri S
BL.EN.U4ECE18138

Shariq Akhtar P.P.
BL.EN.U4ECE18140

Talluri Sayanth Kishore
BL.EN.U4ECE18157

Yash Rajesh Umale
BL.EN.U4ECE18182

Abstract

- Space exploration (unmanned/ manual) involve logging data using **magnetic tape recorders.**
  - Limited data gathering and storage capabilities
  - Mechanical systems prone to failure

- **Solid-state technology** was found as an alternative for these problems.
  - Information stored in **binary** can only be altered by **voltage changes** in transistors
  - Highly reliable, durable and compact.

- **Ease of integration** with solid-state devices facilitates addition of **advanced data management** systems with minimal impact to overall system architecture.

  Availability of multiple interfaces to perform **telecommand**, **telemetry**, **record** and **playback** data make SSRs highly versatile components.

Abstract

- Solid state recorders (**SSRs**) are developed to
  - Store data during imaging or continuous **recording**, and then retrieve during ground station visibility (**playback**)
  - Monitor during remote sensing and experimental missions
  - Store data and health parameters from multiple flight payloads

- Evaluating the flight model of an SSR involves simulation of **data I/O interfaces**, **control** and **status monitoring interfaces**.

- The project focusses on developing a robust **end – to – end system** to **evaluate** the flight model of SSRs, while ensuring **SuperSpeed data recording and playback** with implementation of compression algorithms among other features using a custom board simulator with an FPGA.

Tools and Software

- The **Cypress FX3 SuperSpeed Explorer Kit** offers a one-package solution for the integration of all aforementioned interfaces.

  Supports **SuperSpeed** data transfers over **USB 3.0** to other peripherals (**GPIF**, **SPI**, **UART**) within the kit.

- The project focusses on developing a **firmware for the FX3** to facilitate data transfer between peripherals.

  The adjoint FPGA will be programmed to **run tests** on the SSR **over multiple interfaces** to gauge its reliability.

- Phase II focusses on implementing a variety of other applications, including but not limited to:
  - Implementing a **1553 bus controller**
  - Performing image and video compression over FPGA
  - Command transfer over **SPI** and **GPIF interfaces**
  - Telecommand, telemetry, record and playback data from the SSR

Objective (Phase – I)

Main Objective:

To build and deploy a **system image/ firmware** which implements super-speed data transfers between **USB**, **GPIF** and **SPI interfaces** using the **2 – bit Slave FIFO mechanism**, and to verify **bulk – IN** and **loopback transfers** across an external master (FPGA or ASIC).

Sub Objectives:

- Configuration of the **GPIF state machine** and I/O matrix.
- Provide **DMA transfer** of data for record and playback within the FX3.
- Testing the GPIF state machine with timing diagrams, streamer module and USB Control Utility.
- Testing **Bulk-IN data transfers** via the FX3 to the FPGA or USB host.
- Verifying the **loopback transfer** of data.

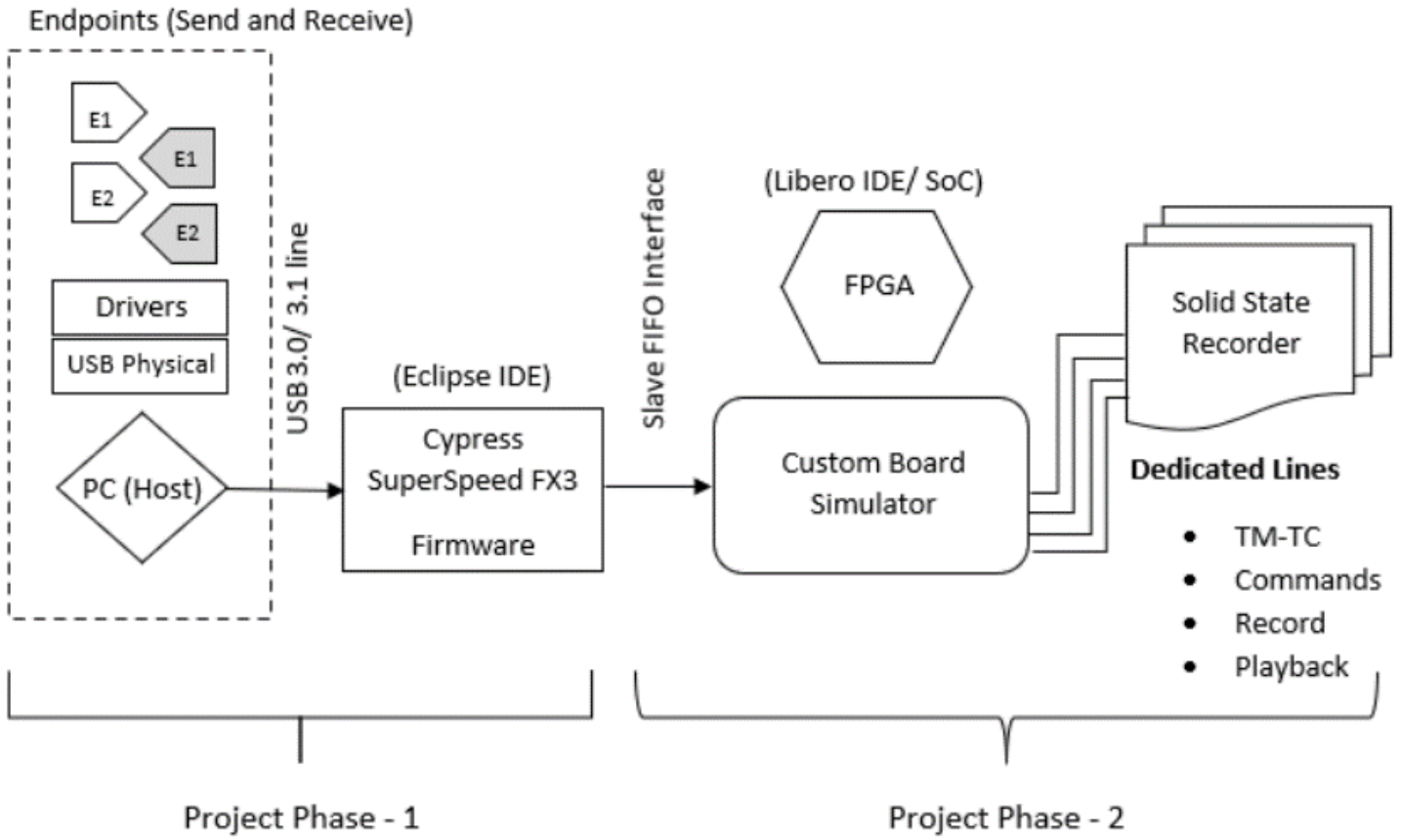
Methodology

Cypress FX3 SuperSpeed Explorer Kit uses the **AHB architecture** for internal DMA and system data transfers, along with the GPIF and SPI interfaces for our application.

The overall proposed architecture for the project is as follows:

- The USB host (2.0 or 3.0, usually a PC) is connected to the FX3 via its USB port.
- The options are propagated via lanes from the **origin USB endpoint** to the **destination USB endpoint** within the FX3.
- USB host is connected to the FX3 on the USB front, the FX3 is further connected to an external processor or the FPGA **via the GPIF pins**.
- The FPGA is further connected to the SSR, which happens to be the DUT (device under test).

Methodology



General proposed architecture for the project

Literature Survey

[1] Data Rate based performance analysis and optimization of bulk OUT transactions in USB 3.0 SuperSpeed Protocol

~ Adithya Rangan; Aravinda Holla; Vikas Kulkarni; Anurag Kumar; Akshay Patil

2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)

- USB 3.0 SuperSpeed is one of the most widely used serial communication protocols**, providing signalling rates as high as 5 Gbps.**
  Hence, data rate based performance is used to verify if the design and implementation conforms to the protocol standards and system requirements.
- This paper presents an elaborate description of various facets through which data rate based performance analysis is achieved for USB 3.0 SuperSpeed **Bulk OUT** transactions.
- Deals with the characterization and statistical variation of causative factors for different scenarios to obtain **minimum**, **maximum** and **ideal data rate** thus leading to optimum performance.

Literature Survey

[2] High-Speed Data Acquisition System for GNSS Applications

~ Jorge Leañez; José Barbería; Santiago Rodríguez; Juan Díaz; Ramón Valle; Javier Garcóa; Carlos Muravchik

2020 Argentine Conference on Electronics (CAE)

- Presents the design and implementation of a versatile high speed data acquisition system for **GNSS signals** capable of transferring large amounts of data to a PC.
- **Global Navigation Satellite System** (GNSS) refers to a constellation of satellites providing signals from space that transmit positioning and timing data to GNSS receivers.
- Implemented on a **Xilinx Virtex UltraScale+ FPGA**, the system involves a high-throughput data path through the on-board DDR4 SDRAMs, based on which a ring buffer is designed to provide the 2 GB buffering capacity.
- In addition, a simple **differencing algorithm** with threshold detection is integrated into the back end for dynamic frame rate operation.

Literature Survey

[3] Design and implementation of USB 3.0 Data Transmission System based on FPGA

~ Fenxian Tian; Juan Li; Xinxin Sun; Jun Yang

3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019)

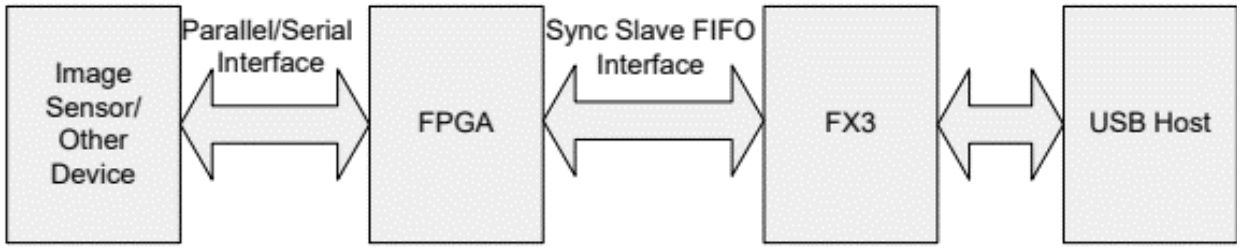
- Emphasis on a system using an FPGA as the core control chip, due to which the FX3 realizes high speed data transmission of the USB 3.0 **synchronous Slave FIFO** mode through the **GPIF** pins.
- The FX3 SDK is built and shipped along with the microcontroller to ease the design and development of the firmware.
- The GPIF II Designer, Eclipse IDE, **ThreadX RTOS**, USB Control Manager and other design tools are third-party software wrapped in Cypress APIs to enable native integration with the hardware.
- A USB device can implement one or more functions, and the capabilities of the device are reported to the host through a set of data blocks called **descriptors**.

Design and Analysis

- **Configuring the GPIF – II I/O Matrix**

  The **GPIF II Designer** is used for designing the **IO matrix**, which is responsible for assigning the **mapping of pins** as per the **slave FIFO** interface requirements.

- A **state machine** is designed using the tool in order to define the sequence of commands and execution, while ensuring conditional **state transitions based on signals** from an external processor.

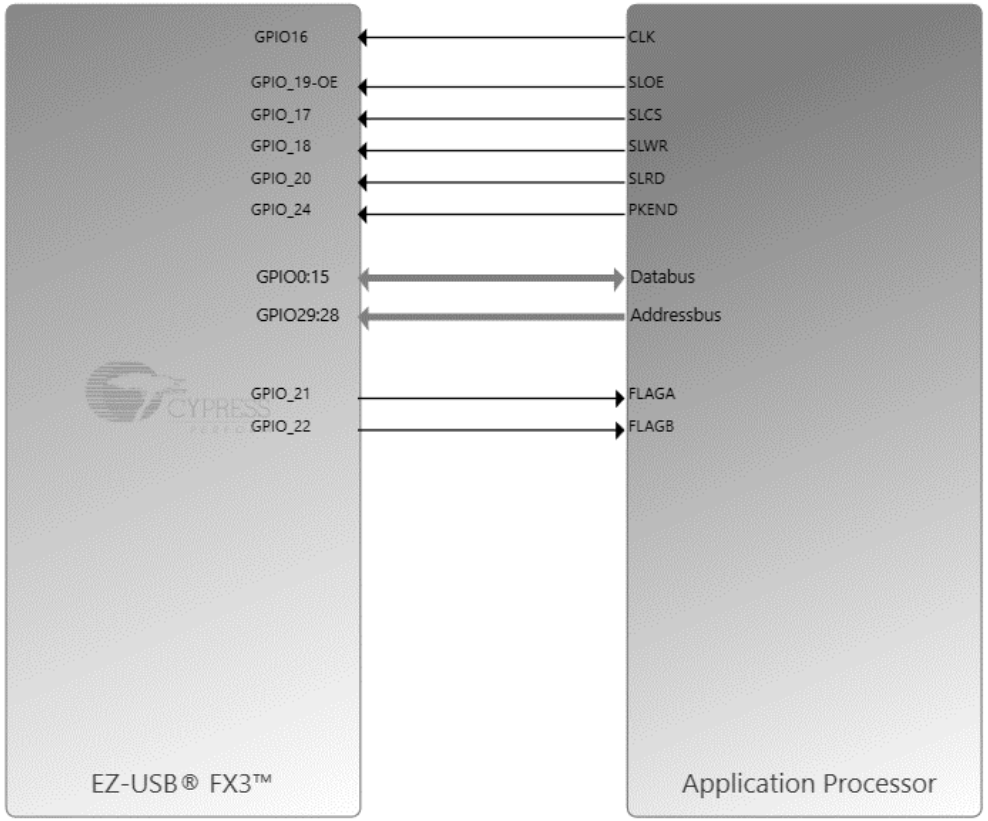


Slave FIFO for transactions between the FX3 and an external master

Configuring the I/O Matrix

- The interface definition window and I/O matrix helps define the electrical interface in terms of:
  - **data** and **address buses**
  - the number and **direction of control signals**
  - the interface clocking parameters

- The I/O matrix helps in configuring
  - FX3 peripherals such as I2C, UART, SPI
  - Data **bus width**
  - Endianness
  - Flags to indicate the status of the thread in terms of current buffer conditions
  - Pin mappings as per the desired interface

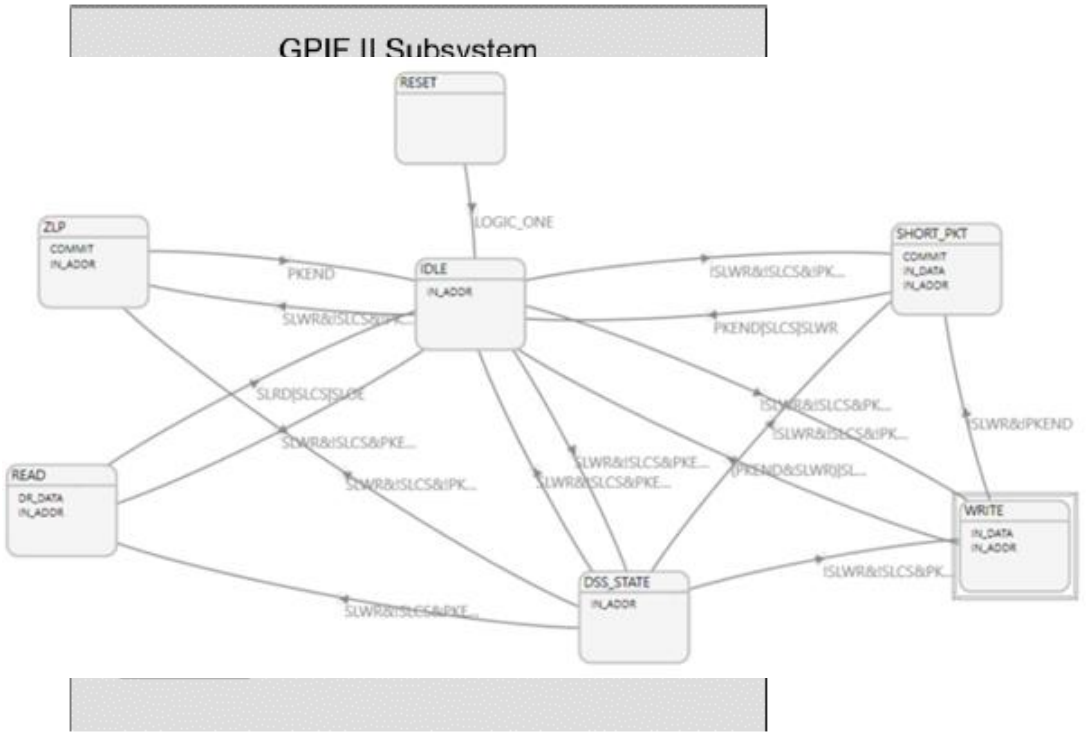


Programming the GPIF II State Machine

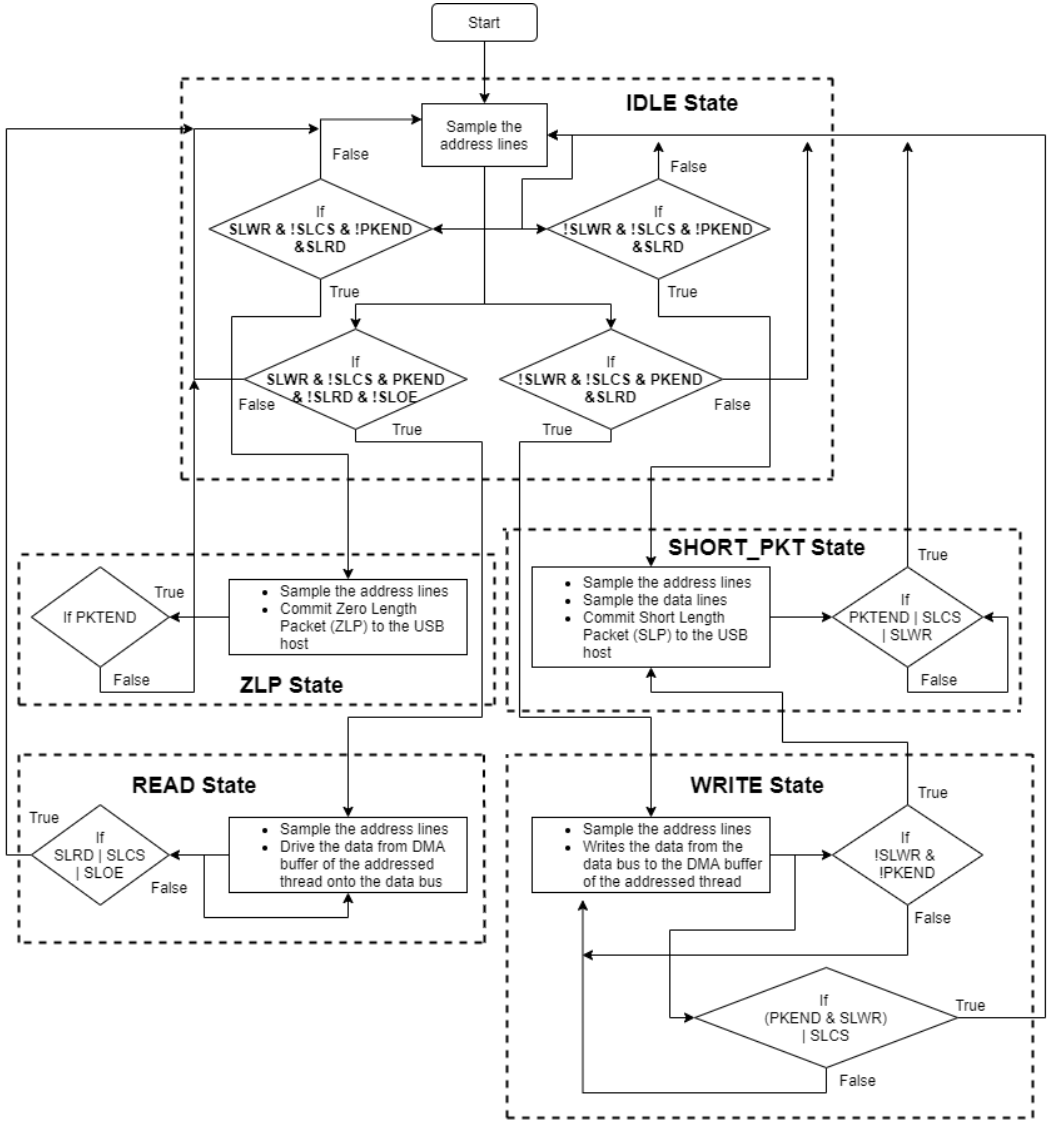
- The GPIF II port provides a parallel interface with maximum of **32 bidirectional data lines**.
  They can be **time multiplexed** into address lines.

- The GPIF II Designer allows the user to
  - Create an **I/O matrix** for mapping pins to signals
  - Defining state machines for specific interfaces
  - **Generate a C header** file encoding all of the GPIF definitions

- The header file can be used to integrate the GPIF functionality into the slave FIFO application firmware.



Logic Diagram for Synchronous Slave FIFO:



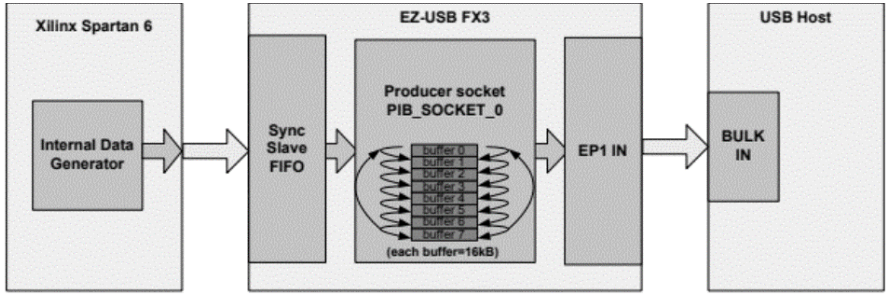
In order to configure the GPIF II interface, the following initial states need to be defined:

- **Interface type:** Slave
- **Communication type:** Synchronous
- **Endianness:** Little Endian
- **Data bus width:** 8 bit, 16 bit or 32 bit
- Address or data bus multiplexed
- **2 address pins** for selecting among 4 available threads
- **Active low signals** for **Read, Write, Chip Select** and **Output Enable.**

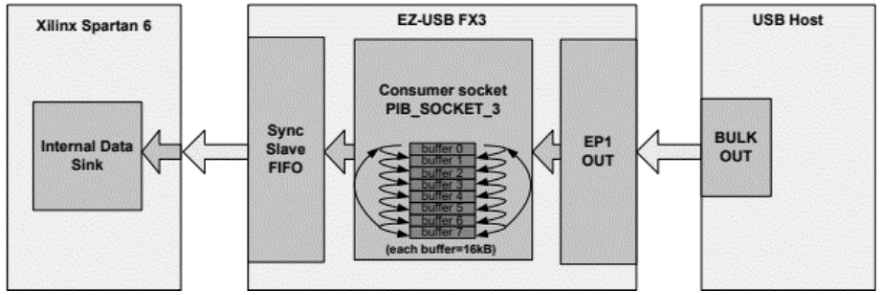
Design and Implementation Logic

- The project consists of an **FX3 Development Kit** interconnected with an **evaluation kit (FPGA)** using a **Samtec to FMC interconnect board**. The interconnect board mates with the Samtec connector on the FX3 and the FMC connector on the FPGA.
- For execution and testing of the slave FIFO interface, the following components are included in the firmware:

Streaming (IN) data transfer
The FPGA does one-directional transfers: continuously writes data to FX3 over synchronous slave FIFO.



Streaming (OUT) data transfer
The FPGA does one-directional transfers: continuously reads data from FX3 over synchronous slave FIFO.



FX3 Firmware Application Structure

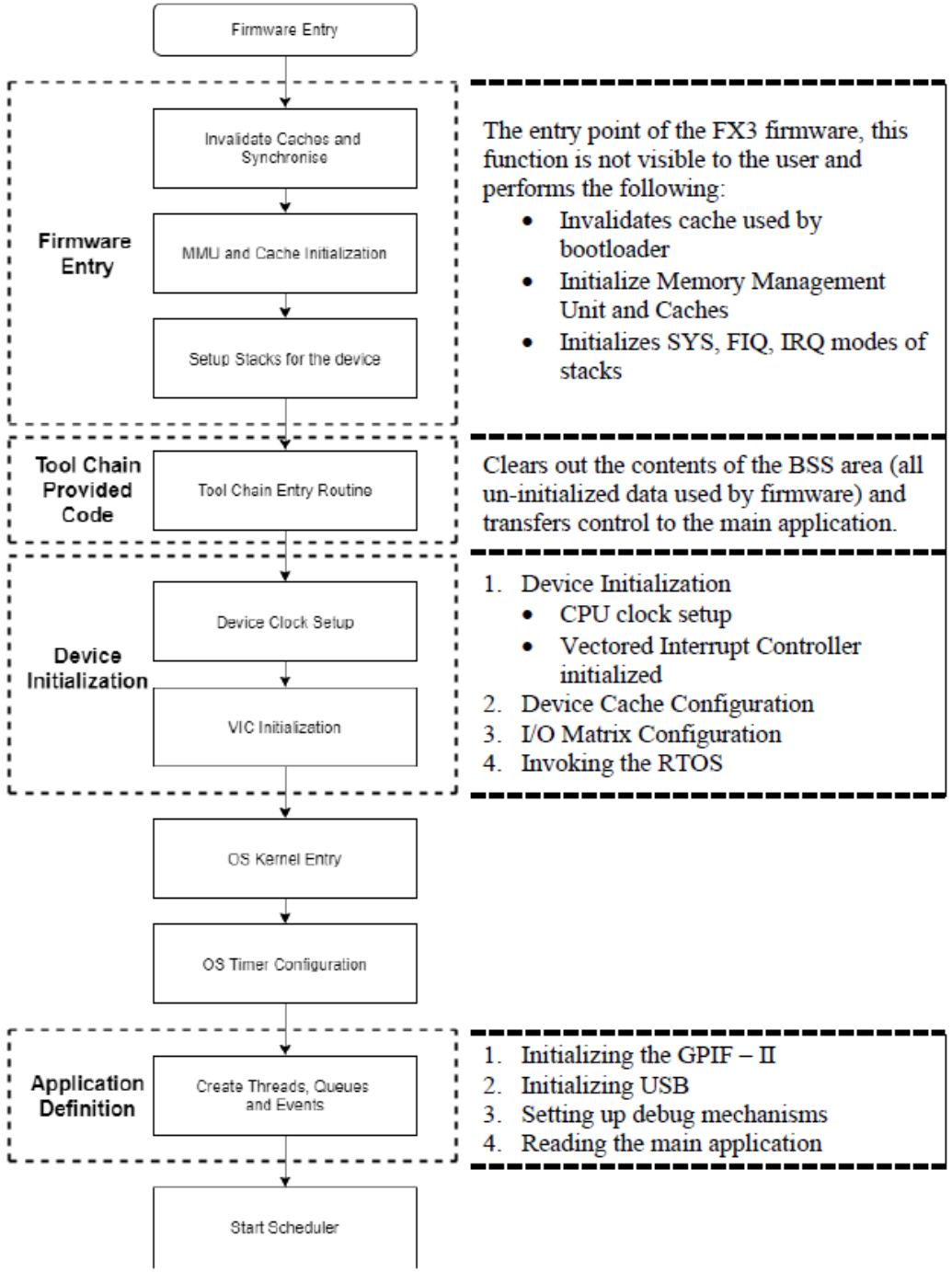
- All FX3 firmware applications consist of two parts:
  - **Initialization code** – Usually common for all applications
  - **Application code** – Code specific to the Slave FIFO application

- The typical file structure for the firmware application would be as follows:
  - A **header file** containing the **GPIF II descriptors** for 16-bit and 32-bit slave FIFO interface
  - A C program containing **USB descriptors**
  - A **header file** containing defines used in the main application. Several important constants essential for setting the initial conditions for execution are defined in this file.
  - A C program containing the **main application logic for synchronous Slave FIFO**.

Initialization Code

As part of the device initialization:

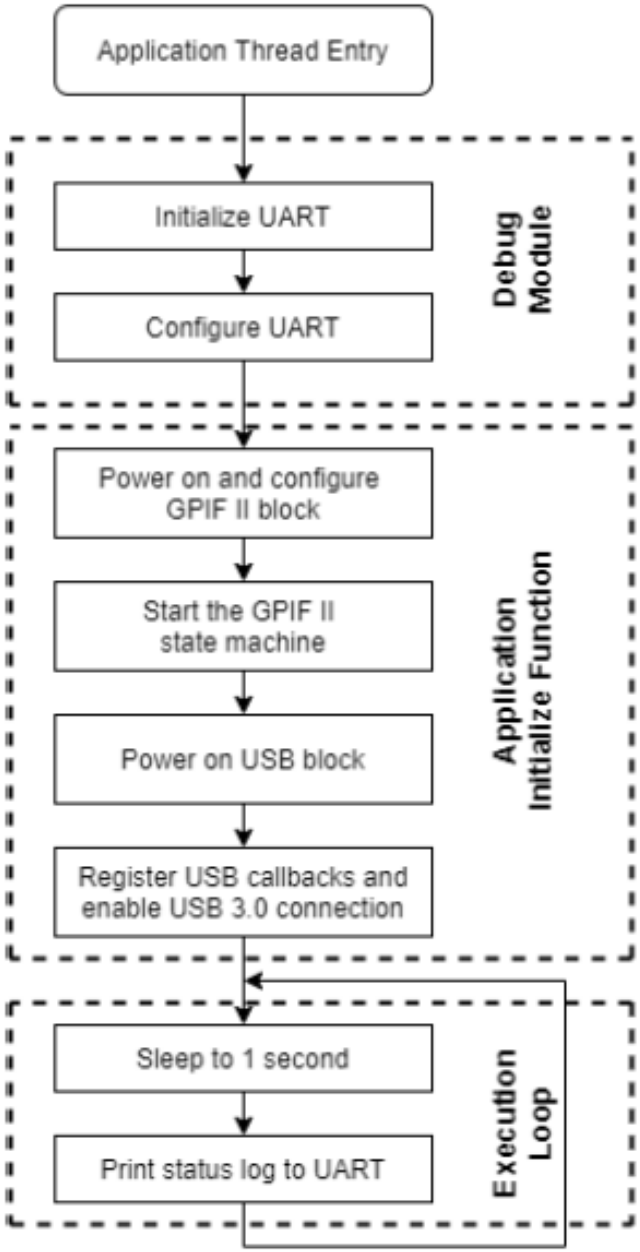
- The CPU clock is setup.
- **Vectored Interrupt Controller** (VIC) is initialized.
- Device caches are enabled (**8 kB data** cache and **8 kB instruction cache**)
- I/O's are configured based on the header files included.
- The inherent **ThreadX RTOS** is invoked, and the OS timer is initialized before transferring the control over to the RTOS scheduler.

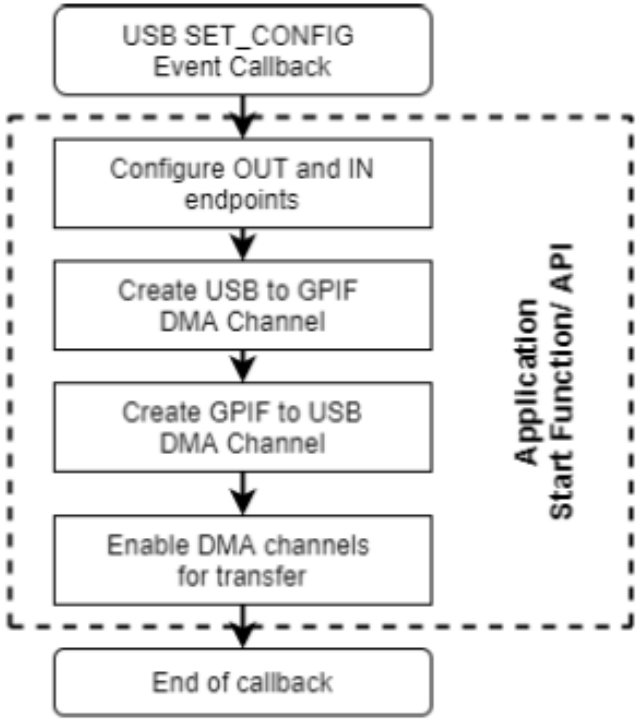


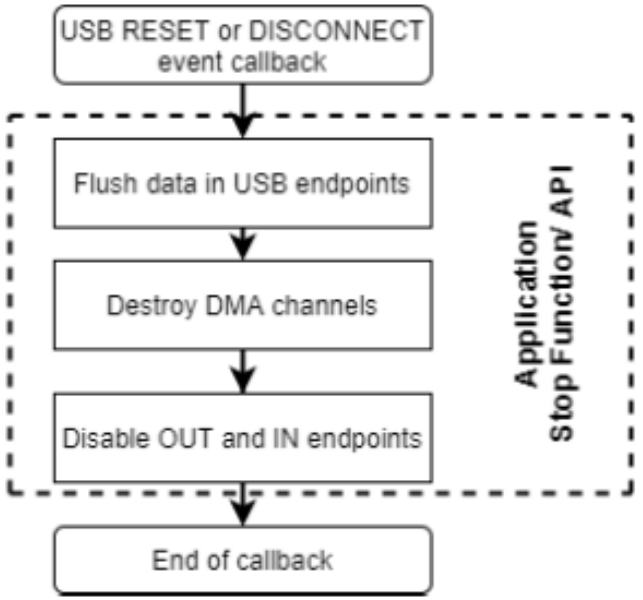
Application Code

The Slave FIFO application code comprises of three parts:

- The **application thread** function to perform application – specific initialization operations.
- The **application start** function to set up endpoints and DMA channels required for USB – GPIF data transfers.
- The **application stop** function to free up DMA channels and disable USB endpoints when a USB reset or disconnect is detected.







Application Thread

The application entry point for the slave FIFO interface execution is the application thread. The main actions performed in this thread are :

- Initializing the debug mechanism
- Initializing the main slave FIFO application

Initializing the debug mechanism

The debug module uses the UART to output debug messages. The UART

- needs to be configured before the debug mechanism is initialized.
- needs to be initialized with the requisite parameters.
- has the transfer size set to infinite

The debug module is hence initialized, with two main parameters: the destination for the debug prints (UART socket) and the verbosity of the bug.

Initializing the main slave FIFO application

The application initialization consists of the following steps:

- **GPIF II Initialization**

- **USB Initialization**
  1. Initializing the USB stack
  2. Registering call-backs
  3. Defining USB descriptors

Completed Work

- Implemented **slave FIFO interface**
- Configured the **I/O matrix**
- Designed **state machine** for Slave FIFO
- Observed Read-Write operations
- **Firmware application** designed to handle
    - USB/GPIF callbacks and events,
    - DMA configuration,
    - Threading, and
    - Descriptors

  Among other components essential for the slave FIFO data transfer mechanism.

Results and Discussions

- USB was chosen as the high-speed data link between the FPGA and PC to leverage the simplicity of a VCP (virtual communication port).
- The **FX3 SuperSpeed Explorer Kit** integrates all of the different operations that interface with the SSR (telemetry, telecommand, etc.) onto a single unit, coupling the advantage of USB 3.0's high speed data transfer mechanism.
- As a precursor to testing solid state recorders, we first implement the Slave FIFO interface to the GPIF pins of the FX3 and configure the microcontroller as explained previously.

To summarize the work and results for phase 1 of the project:

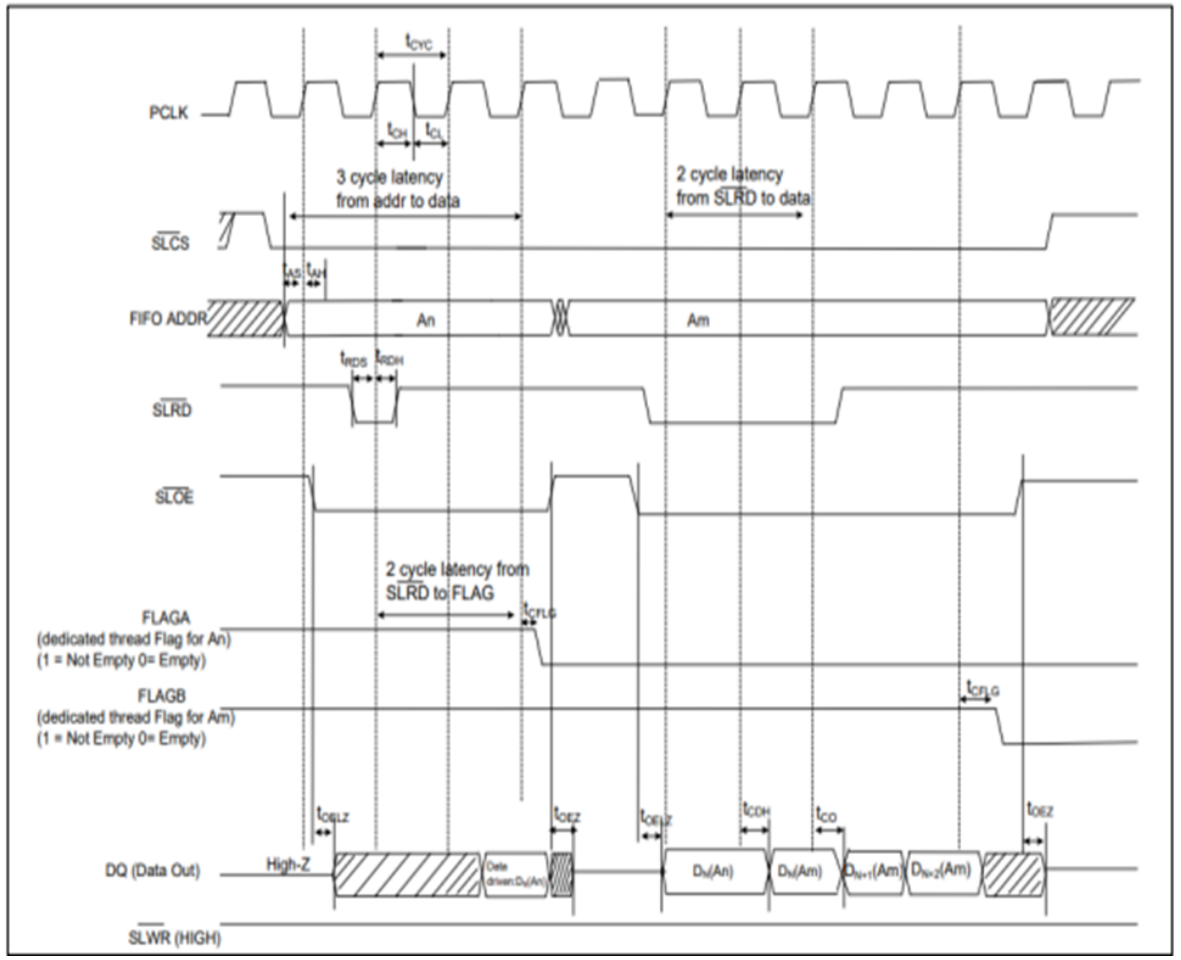
- The GPIF II Designer is used to configure the I/O matrix
- The state machine for slave FIFO interface is designed and commissioned for integration with the FX3 firmware
- Read-write operations are observed with the configured GPIF interface as functions of time and input control signals
- A comprehensive firmware application is designed to handle USB/ GPIF callbacks and events, along with DMA configurations, descriptor definitions, thread and process handling along with setting up debug mechanisms.

Slave FIFO - Read Sequence

The external master drives the two-bit address on the ADDR lines and asserts the read or write strobes.

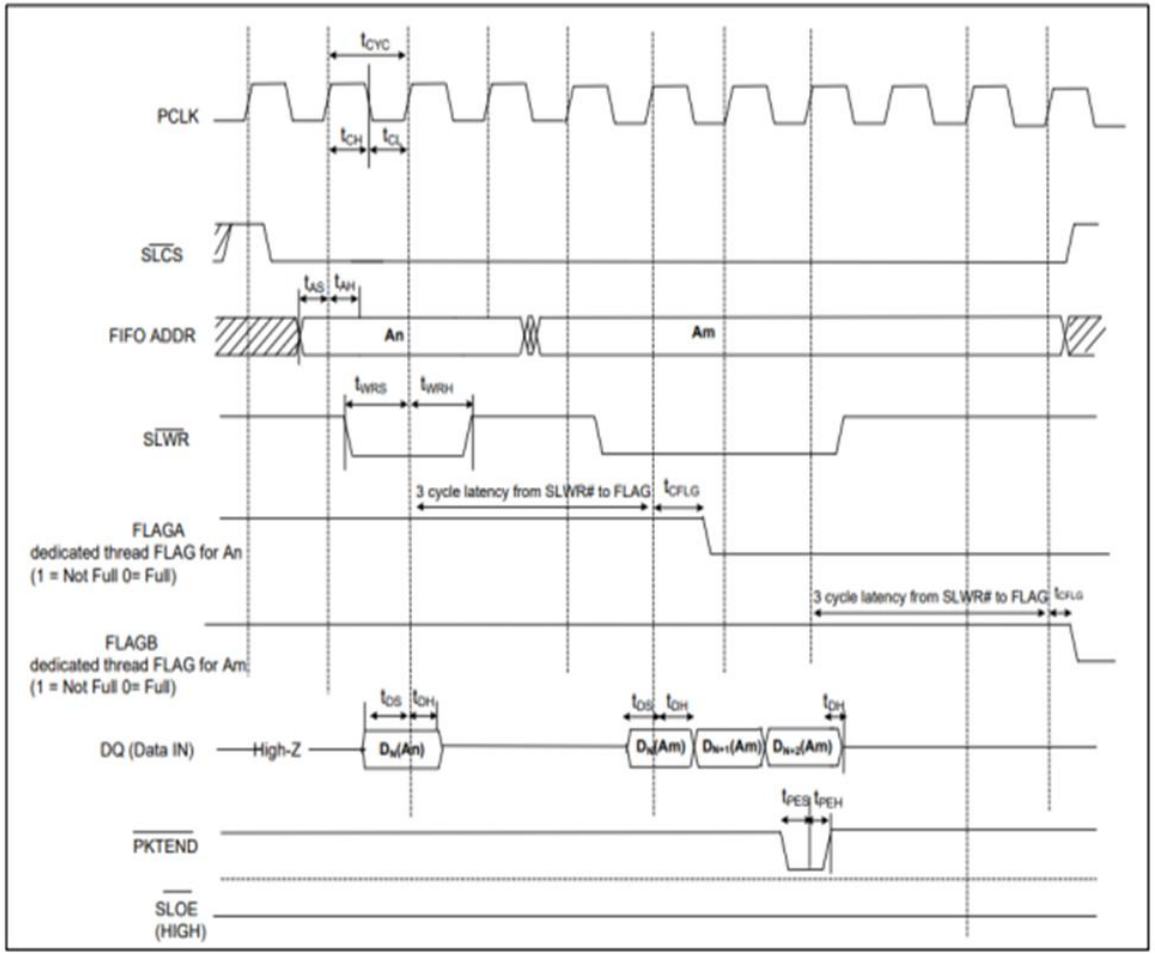
As observed:

- The FIFO address is stable and **SLCS** (chip select) is asserted
- **SLOE** (output enable) is asserted, and it signals the FX3 to drive the data bus.
- **SLRD** (read) is asserted.



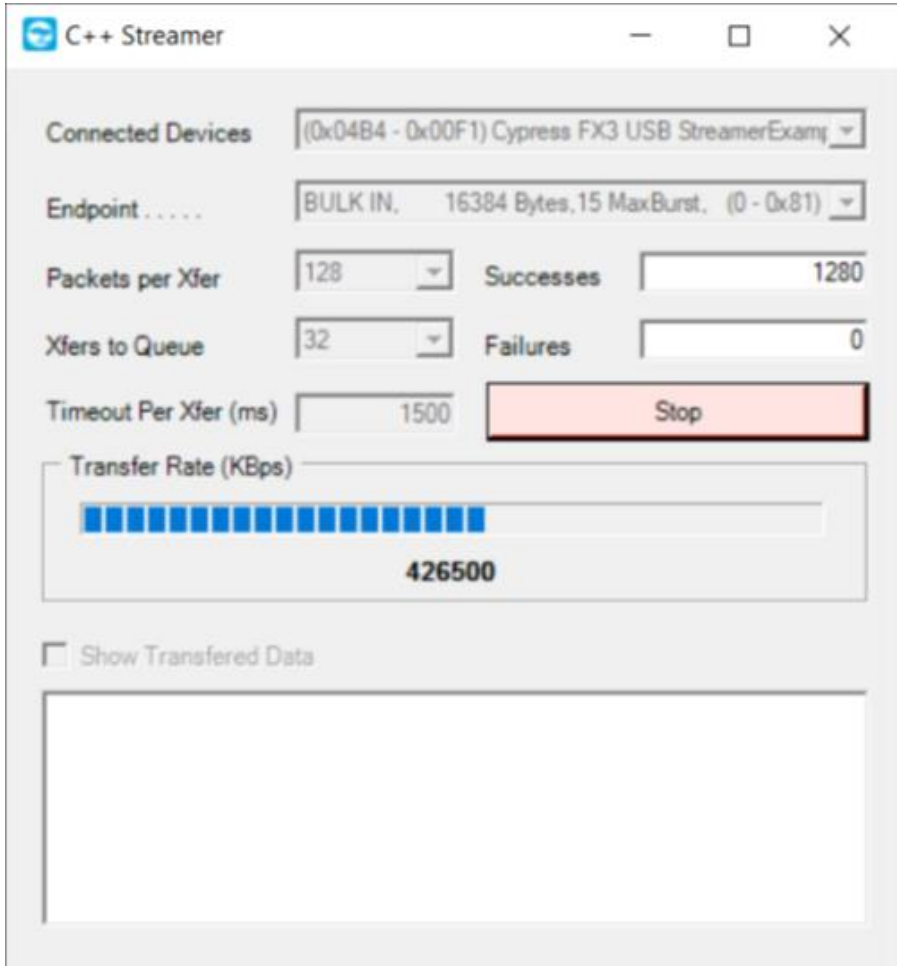
Slave FIFO - Write Sequence

- FIFO address is stable and the signal **SLCS** (chip select) is asserted.
- External master/ peripheral outputs the data onto the data bus.
- **SLWR** (write) is asserted.
- While **SLWR** is asserted, data is written to the FIFO. On the rising edge of the clock, the FIFO pointer is incremented.
- The FIFO flag is updated after a slight delay from the rising edge of the clock.



Streaming Transfers onto the FX3

- The SDK provides a couple of useful tools to track and gauge the extent of data transfer within the microcontroller.
- The FX3 firmware application is developed to facilitate data transfer between peripherals.
- The **USB Control Centre Utility**, along with the **C++ Streamer tool** can be used to quantitatively measure the average throughput observed at the USB and GPIF endpoints.



Observations

- * To be added *

Conclusion and Future Enhancements

- The current phase of the project focusses on configuration of the FX3 SuperSpeed Explorer Kit for integration with another external system for performing high speed data transfers and testing **solid-state recorders** for inclusion in space missions.
- It is possible to consider qualified alternatives to the FX3 microcontroller which consist of technological specifications and capabilities far exceeding those of the prescribed SuperSpeed Explorer Kit.
- An **Nvidia Jetson Nano** or a **Raspberry Pi 4** may seem like powerful alternatives given their superior CPU and RAM capacity, but they're not included in the system since they are standalone devices which do not contain extensive SDK support or pin configuration access for implementing interfaces such as **slave FIFO** on GPIO pins.
- The next phase of the project deals with programming an external processor (FPGA) in Verilog. Moreover, the firmware will be combined and shipped within a single package as an image file, subsequently abstracted by a GUI-driven user-friendly application built using Visual Studio and UI/UX tools such as CSS and Bootstrap.

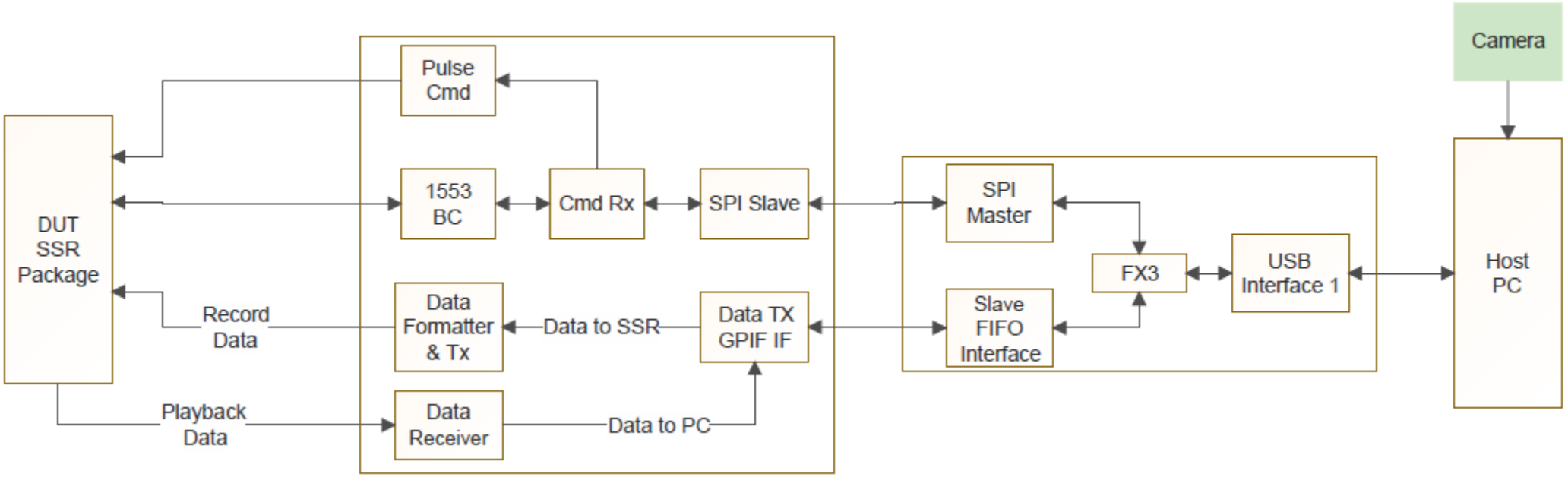
Project Extension – Phase II

Task	Tentative Completion
Implementing a 1553 Bus Controller and building software to target the same	January – March 2022
Generating and receiving command packets for 1553 BC, pulse command and simulator FPGA to forward over an SPI interface	February – March 2022
Configuring the FPGA using VHDL to accept commands from the USB host via FX3 and run tests on the SSR	March 2022
Implementing compression and decompression algorithms to handle image and video data from SSR on the fly via FPGA	

Project Extension – Phase II

Task	Tentative Completion
Implementing the command receiver to receive and interpret commands over SPI to the FPGA	April 2022
Defining and executing appropriate record/ playback protocols for data retrieval from SSR	
(Optional) – Use FPGA compute resources to perform video analysis and image processing	
Redesign FX3 firmware and FPGA configurations to facilitate real-time video streaming to FPGA/ from SSR	April – May 2022
Designing an application using Visual Studio to issue telecommand, telemetry and toggle compression for incoming or outgoing data	

Project Extension – Phase II



A rough schematic to depict the project extension for phase II.

References

[1] C. K. A. Rangan, K. A. Holla, V. Kulkarni, A. Kumar and A. Patil, "Data rate based performance analysis and optimization of bulk OUT transactions in USB 3.0 SuperSpeed protocol," 2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), 2017, pp. 114-119, doi: 10.1109/ICATCCT.2017.8389117.

[2] J. G. V. Leañez et al., "High-Speed Data Acquisition System for GNSS Applications," 2020 Argentine Conference on Electronics (CAE), 2020, pp. 14-19, doi: 10.1109/CAE48787.2020.9046375.

[3] Fenxian Tian, Juan Li, Xinxin Sun and Jun Yang. Design and implementation of USB3.0 Data Transmission System based on FPGA. 3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019), Advances in Computer Science Research, 2019.

Thank You