CZT Readout System Documentation

Suchet Gopal and Ashwajit Singh

Updated: December 2024

1 Introduction

Our goal was to create a data readout system for a Cubesat with 2 CZT (Cadmium Zinc Telluride) detectors interfaced with the CMOD A7 FPGA board through SPI. We chose this board because it met our size requirements[1], and had a PMOD header allowing us to connect 2 SPI devices (in this case, detectors) easily. The second requirement was not strictly necessary, as we can configure GPIO (general purpose IO) pins for SPI as well, it was purely for convenience.

The FPGA board had to perform two main functions:

1. Act as a middleman, interfacing with both the onboard computer, and the two detectors, and passing data between them
2. Decoding and encoding data sent between these two devices, since the computer sends and receives 32/64 bit packets, while the detector sends and receives 8/16 bit packets

There was legacy code to achieve a similar goal (although written specifically for the Pynq instead of fully in HDL so it could be used with any FPGA), but we didn't have access to this because it hadn't been backed up to GitHub. We did however have access to the bit file and the ipynb file that interfaced with it, which was useful to compare results once we had written the code. This code is available on the STC branch on the Daksha repo (we specifically used test-2-data-iitb). Please use GitHub.

[1] smaller than 100 mm $\times$ 100 mm

2 Block Diagram

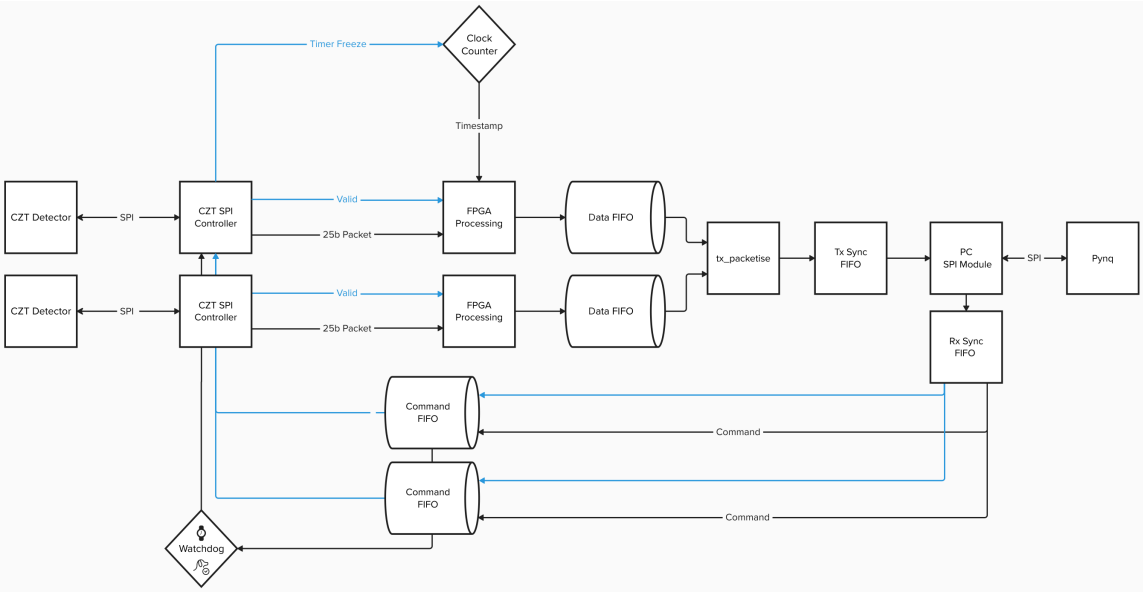

Figure 1: Block diagram

Each detector has an SPI controller that acts as the master, and decides when to poll for events

3 Initial Testing Approach

Since we were initially having issues configuring the Pynq board, we attempted to test the code on the CMOD directly using a USB-UART converter to send and receive packets. This code is available on the UART-Test branch. However, do not do this, altering the code to make it compatible with UART (which is much slower and more error-prone) required a lot of work, and even after this testing it was extremely hard, and did not lead to any clear results. The advantage of the Pynq board is that having the PS interface allowed us to debug and read data much more easily, since you can write simple Python code to encode and decode messages. We thus shifted to first testing completely on the Pynq before shifting the PL (programmable logic) part to the CMOD board.

4 Part 1: Testing on Pynq

Our code consists of two parts: the bulk of it is in VHDL, which is on the PL part of the board, and an ipynb file which encodes and decodes the sent and received packets. For the Python code that interfaces with the onboard AXI SPI, we used this GitHub repo, which is an excellent resource.

We faced multiple bugs that we incrementally solved. They are listed here:

1. The watchdog code (that ensures the code doesn't get stuck) seemed to be stuck at a value, and since it was non-essential while testing, we removed it for the time being

2. We updated the FIFOs to make assigning data_out asynchronous instead of synchronous, as it was delaying the output by a cycle, causing issues

3. We had timing issues with the onboard Pynq clock, which is an error several other users have faced as well. This was solved by using the internal 125 MHz clock (Pin H16) instead of the PL fabric clocks

5 Part 2: Testing 2 Detectors on CMOD

We obtained inconsistent results when we ported the code to the CMOD board. We determined that this was primarily due to two causes, that were absent when we were testing solely on the Pynq board:

1. The jumper wires connecting the CMOD to the Pynq board were loose, and occasionally lost connection. This manifested as the system turning off entirely for short bursts and also in occasional bit errors

2. Since the Pynq and CMOD boards had independent clocks (although at the same frequency), they are not perfectly synchronised

We solved the first problem using a FIFO that is written to and read from in different clock domains - i.e., the SPI Slave is clocked using the SPI Master's clock, and the FIFO is used for CDC synchronisation.

6 How to interface the Detector

Going through the datasheet for the OMS40G256 detector, it is clear that it lacks several important low-level details on how it is meant to be interfaced. We will enumerate some key points we have found from testing that may be helpful:

1. Internal FIFO: The detector has an internal FIFO for 256 events. It fills this FIFO passively when not in Event Read Mode as well.

2. Event Read Mode (EVRM): The detector powers-on into a state in which EVRM is initially OFF. So on startup we can immediately send commands without any issues. Once EVRM is on, to exit we simply send a BREAK or EVRM OFF command. Since the Busy/Ready bit now indicates the existence of events, simply ignore this bit and send the command regardless. Note that using commands involving data reads/writes will require the internal FIFO of the detector to be empty.

3. Parity Calculation: The number of 1s during any given cycle must be made even by means of the parity bit. In a COMM cycle, this effectively means setting the parity to be equal to xor_reduce(cmd_id), where cmd_id is the 8 bit code given in the datasheet. For data cycles, the first bit is necessarily 1 (refer datasheet), so we set the parity to be: $\sim$xor_reduce(cmd_data).

4. SPI Protocol and SCLK: Though not clear in the datasheet, the detector follows something akin to SPI Modes 1 and 2: We sample on falling edges and shift on rising edges. However, there are a few key differences:

   (a) SCLK clocks the detector. It can't be turned off when not in use. This is why it is simply denoted as CLK in the datasheet.

   (b) This is not a standard SPI cycle! The first bit received after SS goes low indicates whether the cycle should be continued or not. We have found that particularly during Data Read/Write cycles, it takes a few cycles of SCLK for the detector to become ready for reading or writing. Therefore using a standard SPI device for readout is infeasible.

7 Authors

Suchet Gopal (gopal.suchet@gmail.com, 22b1814@iitb.ac.in)
Ashwajit Singh (singh.ashwajit@gmail.com, 22b1227@iitb.ac.in)