Open Image

Summary of Hardware, Firmware, and Software Design

WHF

February 2021

1 INTRODUCTION

The Open Image concept is to provide a completely open ultrasonic imaging solution. By removing the barriers put up by commercial equipment manufacturers, researchers and students will have unfettered access to all levels of the ultrasound processing chain, spurring innovation.

With the Open Image system researchers, engineers, and innovators will have free and unrestricted access to: all hardware schematics and circuit board layouts; to the firmware and application interface source codes; and to a framework for integrating new processing algorithms. These pieces are described in the sections below.

2 HARDWARE

The Open Image system consists of two boards: a processing board, and the ultrasound daughterboard. To reduce costs we chose an existing FPGA evaluation board as the processing board. We then designed and built a daughterboard that mounts to it.

2.1 PROCESSING BOARD

So as to maximize the potential of the design, we chose the ZCU106 evaluation board from Xilinx.

It has:

- a Xilinx Zynq UltraScale+ XCZU7EV Multi-Processor System on Chip with
  - ARM Cortex-A53 64-bit quad-core processor
  - ARM Cortex-R5F dual-core real-time processor
  - 504k programmable logic cells
  - 1728 DSP slices
- USB 2/3
- a PCIe endpoint (up to 8 GT/s)
- 8 GB RAM
- SD card interface
- Gigbit Ethernet
- DisplayPort and HDMI (for local visualizations)

The evaluation board has two 400-pin mezzanine connectors, which adhere to the VITA 57.1 FMC specification. The ultrasound daughterboard interfaces with the processing board via these connectors.

2.2 ULTRASOUND DAUGHTERBOARD

2.2.1 Ultrasonic Pulsers

To create sixteen channels of ultrasonic pulses, we placed four HV7321 pulser ICs on the board. See Figure 1. Each chip can support four channels, switching between 5 different output levels at 220 MHz. On our circuit we have connected +80V, +40V, 0V, -40V, and -80V as the output voltages. The device has an integrated Transmit/Receive switch, which provides isolation for the receive electronics during transmission of the pulse.

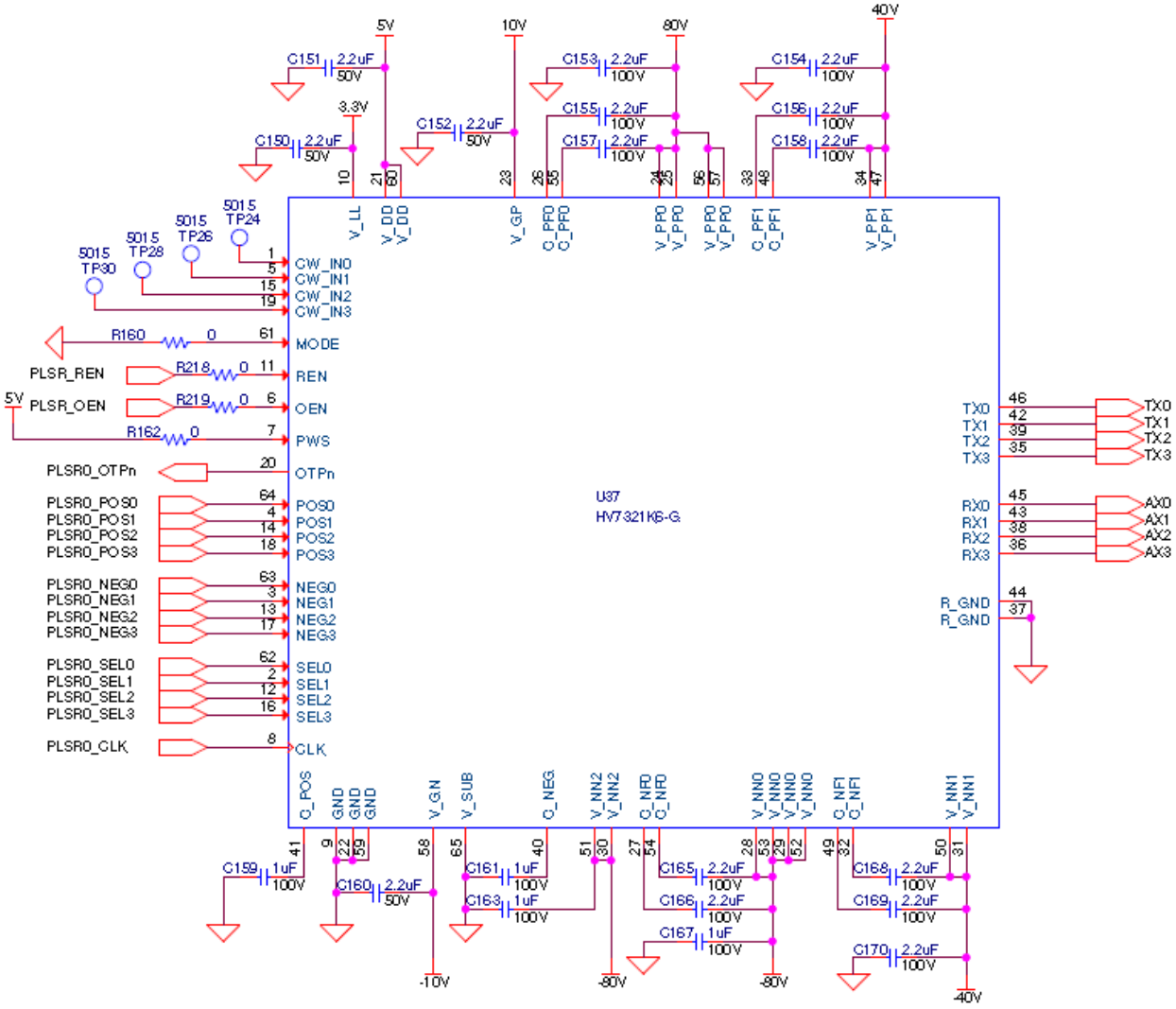

Figure 1. One instance of the pulser IC.

Each pulser is driven via a synchronous 12-bit parallel interface, that simultaneously updates each channel on the rising edge of each clock.

2.2.2 Receive Channels

In order to digitize the return signal from the sixteen channels of signal, we added two AD9670 ICs. Each chip supports eight channels, with an analog front end (AFE) and digitizer for each. The AFE features a low noise preamplifier with a selectable gain of up to 21.6 dB; a PGA with a gain of up to 30 dB; and a Variable Gain Attenuator, for time-gain compensation, offering up to 45 dB of attenuation.

The ADC has a 14-bit converter, and can sample up to 125 MHz, with a full-scale SNR of 59 dB.

We show the input channel signal conditioning in Figure 2. Each channel has 50Ω input termination, and clamping diodes to prevent voltage excursions from damaging the hardware.

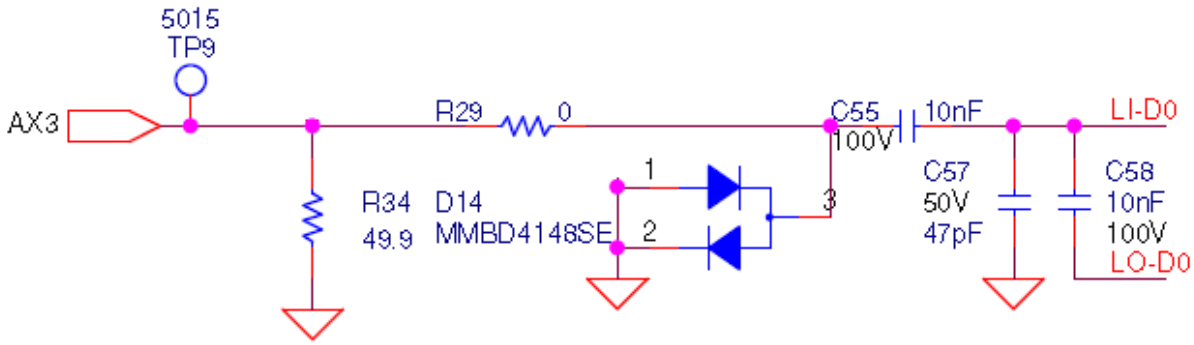

Figure 2. Receive channel instance.

We chose to sample at 80 kHz. Each channel has its own dedicated LVDS (Low Voltage Differential Signaling) signal pair which routes to the FPGA. These serial data are accompanied by two LVDS output clocks: a DDR (Double Data Rate) bit clock, DCO, and a frame clock FCO.

Each AD9670 is accompanied by a DAC (Digital to Analog Converter) to drive its TGC input. The AD5424 is an 8-bit, 10 MHz DAC with a parallel interface. The circuit in Figure 3 serves several purposes.

- It converts the current output of the DAC to a voltage.
- It scales the voltage output to match the expected input range of the GAIN input of the ADC.
- It drives the differential GAIN input.
- It provides reconstruction filtering to limit the impact of the DAC's discrete output steps.

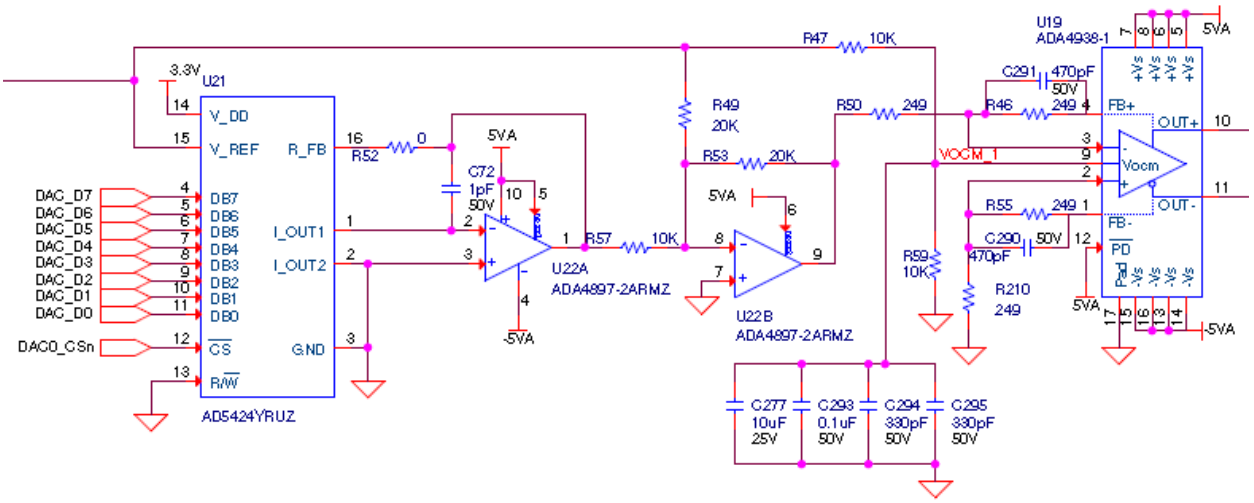

Figure 3. DAC filtering and TGC driver.

The board also contains several housekeeping voltage supplies generated from the 12V available from the processor board.

3 FIRMWARE

Each major component of the Open Image electronics has a corresponding component inside the FPGA. These IP blocks are written in VHDL by our team, and interface the ARM processor in the Zynq SOC with our hardware. The blocks are interconnected graphically inside Xilinx's FPGA synthesis tool, Vivado ®.

3.1 ULTRASONIC PULSER INTERFACE: AD7321_AXI4

Figure 4 shows the connections for our pulser IP. This IP communicates with the processor via a Slave AXI4 data bus. This bus is used to setup control registers and write the output waveforms used to drive the pulsers. When the 'start' signal goes high, the block drives the output clock and the 12 output signals.

The IP also controls the REN (Regulator Enable) and OEN (Output Enable) signals to the HV7321, monitors the OTP (Over Temperature) signal for faults, and drives the TX and RX signals used for triggering.

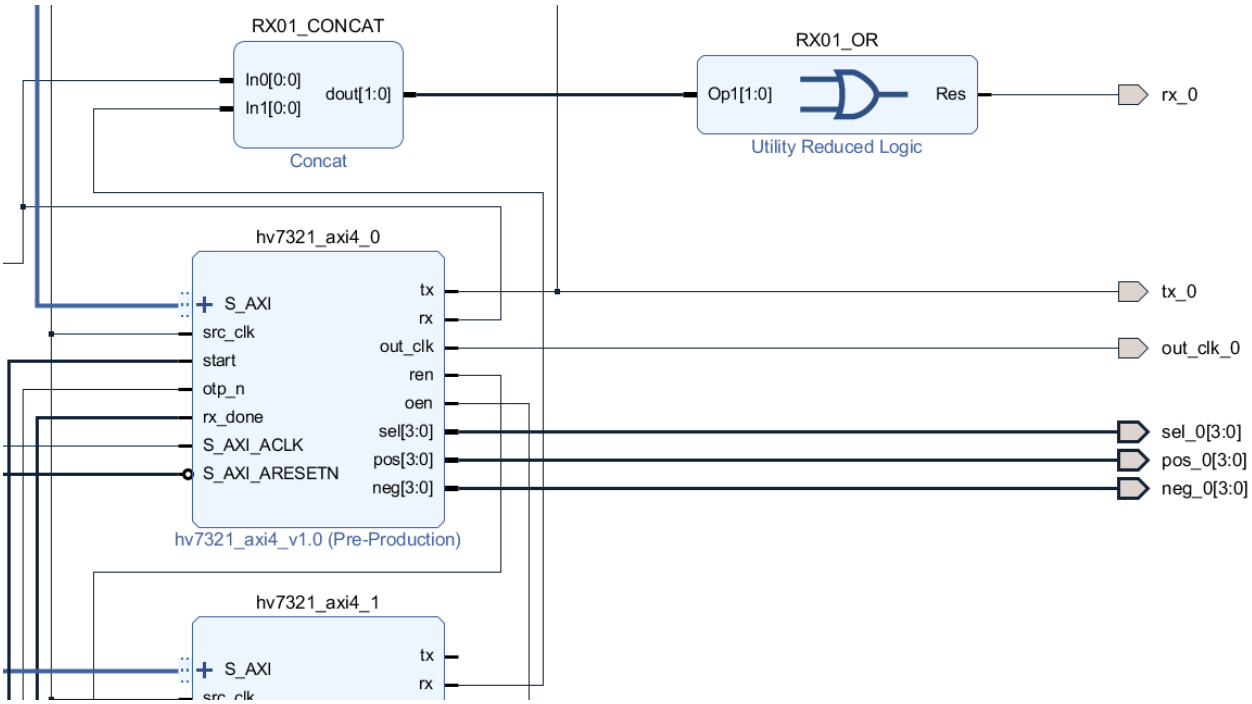

Figure 4. Connections of a pulser IP, showing the output ports.
Note that the 'RX' signal of pulser blocks 0 and 1 are OR'd to make the RX signal for the ADC.

3.2 ULTRASONIC RECEIVE INTERFACE: AD9670_AXI4

Figure 5 shows the ADC interface IP, the block representing the Zynq processor, and the DMA (Direct Memory Access) peripheral used to connect the two.

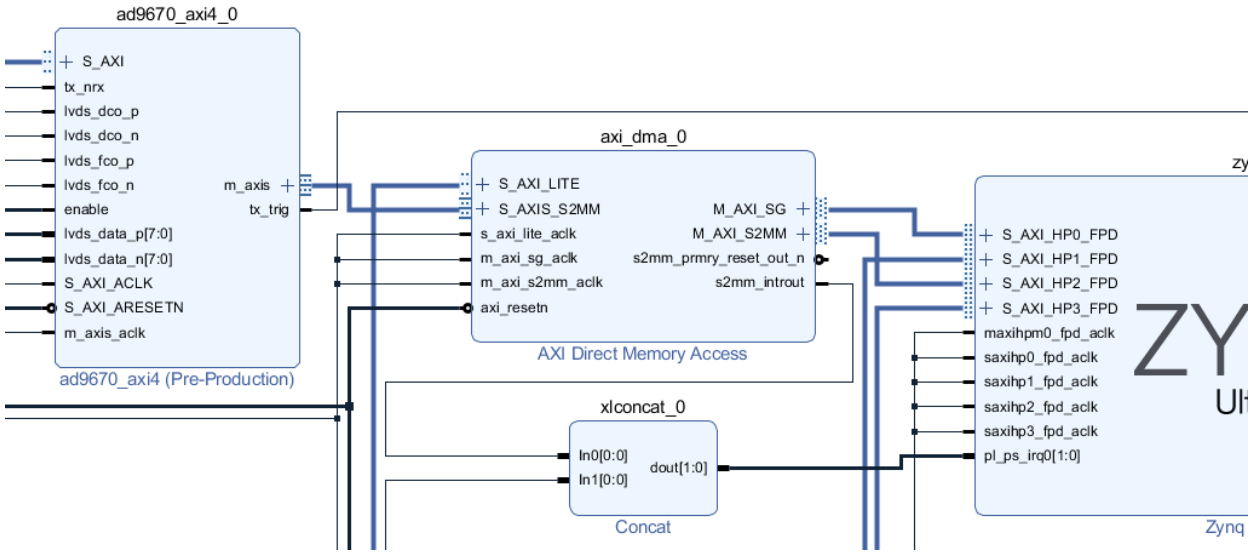

Figure 5. Interconnections between ADC IP, DMA, and processor.

Our custom IP peripheral, the AD9670_AXI4, interfaces to Xilinx's standard AXI DMA block via a streaming AXI bus. Unlike a standard AXI bus, a streaming bus has no notion of addressing. In this case the DMA block receives the streamed data samples from the ADC and places them into the RAM chips on the ZCU106 evaluation board. It does this through the Zynq SoC itself, through special high-speed ports designed for that purpose. The SoC has a dedicated hardware interface to the RAM chips, promising excellent performance and obviating the need for generating the interface in the logic fabric.

Please note that the firmware described in this section is only used for streaming the samples from the ADC to memory. Configuration of the ADCs themselves is handled by software using a synchronous serial port interface.

3.3 TGC DAC Interface: AD5424_AXI4

Lastly, we show the IP block representing the interface between the processor and the TGC DACs in Figure 6. This single block interfaces with both DACs (one for each ADC). Like the pulser IP, this block is preloaded with the TGC waveforms via the Slave AXI bus. When the 'start' signal goes high, the block outputs the waveform at 1 MHz. It drives the two DACs alternately using the chip select signal, which also acts as a clock. When driven low, the DAC accepts the data on the parallel bus.

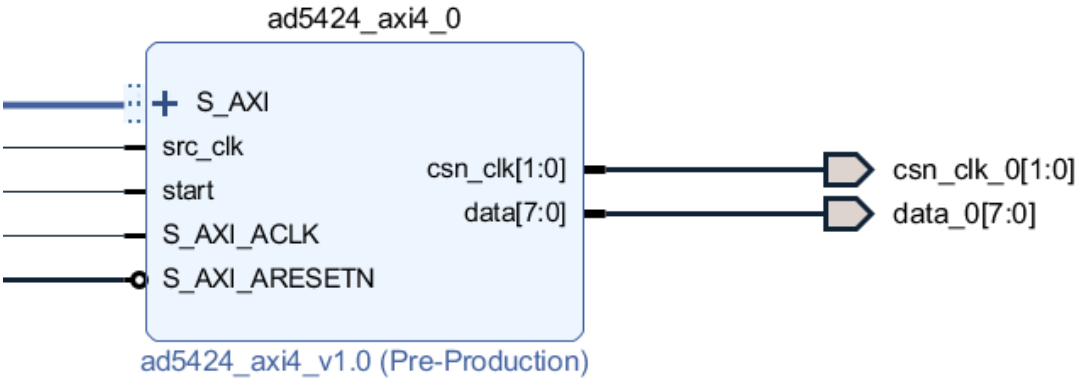

Figure 6. Connections of IP used to drive TGC DACs.

4 SOFTWARE

The software for the Open Image device runs on one of the processor cores in the Zynq. It manages the hardware via the firmware modules mentioned above, and communicates with the host machine via the Zynq's Gigabit Ethernet interface.

4.1 State Machine

The software is driven by a state machine, which organizes the behavior of the system into discrete modes of operation, separated by distinct transition criteria. We have depicted the system states in Figure 7.

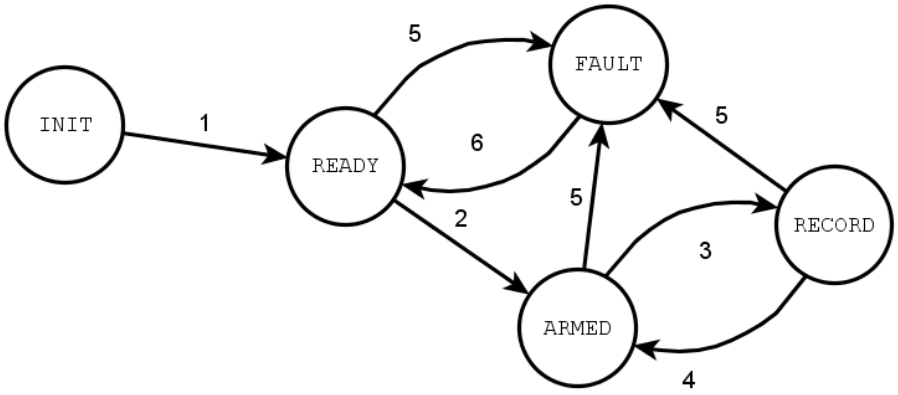

Figure 7. Open Image software state machine.

A summary of the states is in Table 1.

State	Description
INIT	Initial state at power up, prepares the system for operation
READY	Default rest state
ARMED	Prepares all subsystems for recording
RECORD	Waits for the firmware to record the desired number of samples
FAULT	Catch state when an error occurs

Table 1. States of the machine.

Table 2 describes the transitions between the states. Please refer to the indices which mark the transitions in the state machine of Figure 7.

Transition	Source State	Destination	Criteria
1	INIT	READY	Initialization operations are complete
2	READY	ARMED	The command to record is received
3	ARMED	RECORD	The hardware is ready to fire and record the next shot
4	RECORD	ARMED	Recording of the shot is complete

5	several	FAULT	An error condition occurs
6	FAULT	READY	The clear-fault command is received

Table 2. State machine transitions.

4.2 MODULES

To facilitate understanding and reduce errors, the software is broken into distinct blocks, called modules, each with its own defined purpose. We show a graphical depiction of the modules and their interrelationships in Figure 8. Each module has an "oi" prefix to denote that it is specific to the Open Image application. Note that low level dependencies, such as the C Runtime Library, the Board Support Package which contains hardware drivers, and the Lightweight IP library are excluded. All firmware is written in ISO standard C.

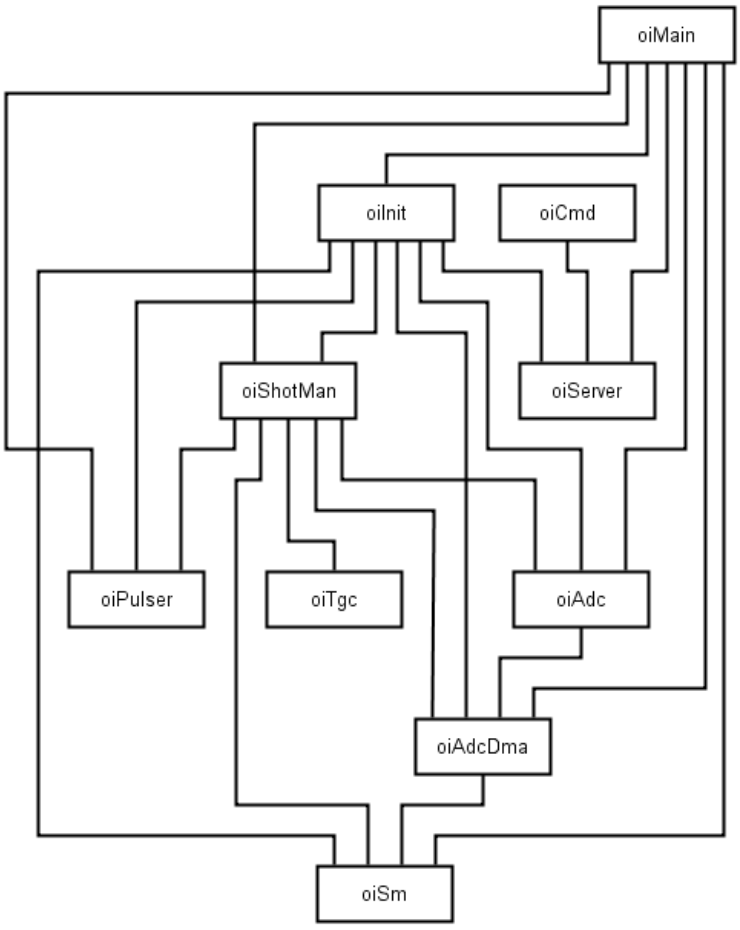

Figure 8. Modules which comprise the Open Image software.

Table 3 lists each module in alphabetical order and describes their purpose.

Module	Purpose
oiAdc	Initialization, configuration, and power management of the analog-to-digital converters
oiAdcDma	Managing of the streaming of digital samples to memory
oiCmd	Parses commands received from the host over TCP/IP
oiInit	Handles ordered initialization of all firmware, hardware, and software modules
oiMain	Contains the primary software loop
oiPulser	Manages the pulser firmware, including setup and writing pulser waveforms received from the host
oiServer	Configures the LWIP library to provide a TCP socket for the host to connect to; forwards incoming commands to the oiCmd module
oiShotMan	Manages the shots requested by the host for a frame
oiSm	Handles the state machine, receiving events and performing state transitions
oiTgc	Configures the DAC firmware to perform Time Gain Compensation as requested by the host

Table 3. Software modules and their function.

5 MATLAB INTERFACE

To allow the user to interact efficiently with the Open Image system, we have created a MATLAB interface. Figure 9 is a Unified Modeling Language representation of the class hierarchy. Example scripts are provided to demonstrate use of the classes, but briefly the procedure is:

1. Instantiate the openimage class, that represents the connection to the system.
2. Connect to the hardware using the open() method.
3. Instantiate the oi_frame class, that represents the settings desired for data acquisition.
4. Specify the number of shots desired by setting the nShots member of the frame.
5. For each shot, define the rx and tx parameters.
6. Begin the ultrasound operation via the queue_frame() method of openimage.

7. Wait a brief interval for the frame to complete.
8. Download the frame data using the get_frame() method.

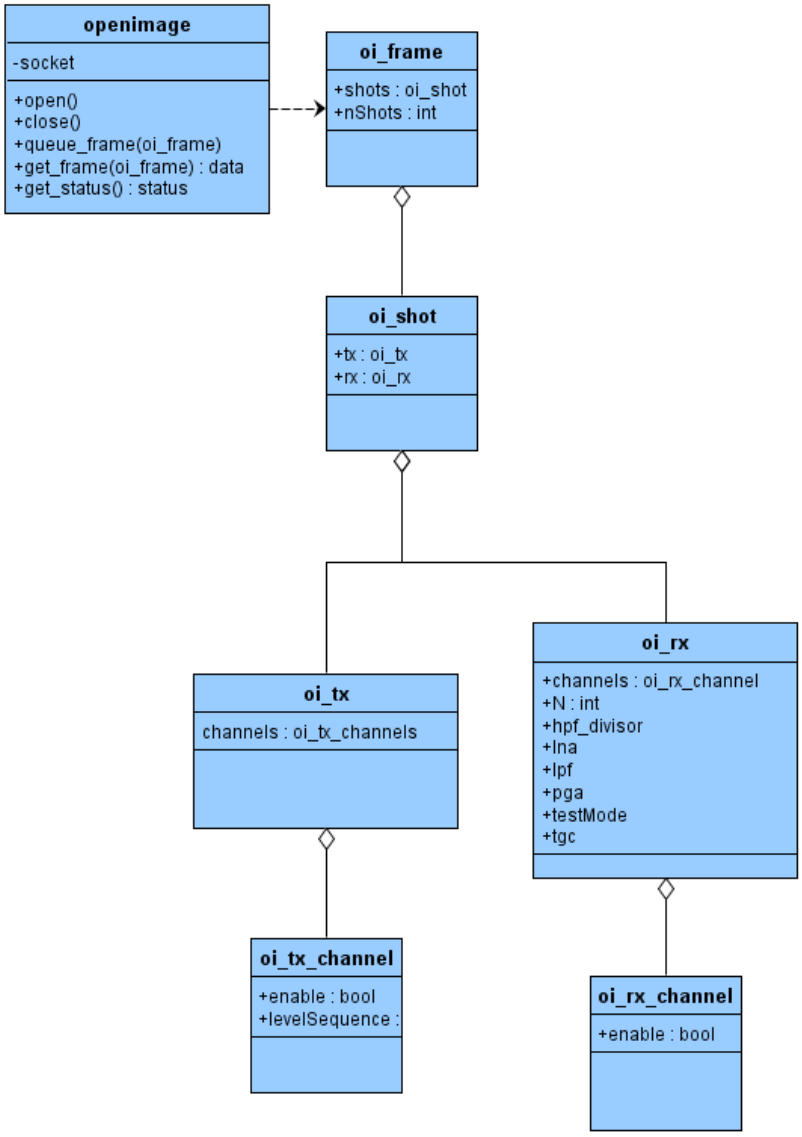

Figure 9. A UML diagram of the Open Image MATLAB class hierarchy.