TangerineSDR

TangerineSDR

Data Engine Firmware / Software

Architecture Document and Test Plan

Version Number: 0.2 *PRELIMINARY*

Version Date: April 25, 2023

Document Number: TBD

VERSION HISTORY

Version Number	Implemented By	Revision Date	Approved By	Approval Date	Description of Change
0.1	T. McDermott	September 19 - October 6, 2022			Original Issue. Much missing content.
0.2	T. McDermott	April 25, 2023			Extensively restructure. Add detailed test plan for two configurations

1. Introduction

1.1. Scope

The Data Engine (DE) Version 1 provides the connection of TangerineSDR modules (Receiver, Clock, Magnetometer, VLF receiver) with an FPGA. The network interface connects the DE to the Local Host (LH) using 1 Gigabit Ethernet. The control interfaces to the hardware peripheral modules are via I2C and SPI. The clock module also uses USB connections but these are not provided by the DE. There are other module-specific data interfaces to the various receiver modules. The DE Version 1 hardware contains a MAX10 FPGA to control the system, and to communicate with the Local Host computer via 1 GbE.

The MAX10 Development Kit contains a subset of the interfaces provided by the DE. An adaptor board is used to connect the DevKit to the Receiver, Clock, and other peripherals. The DevKit will be used for initial prototype development until the actual DE is available and tested.

This document outlines the architecture of the FPGA firmware and software to accomplish the TangerineSDR initial objectives. It also discusses some alternative approaches for controlling the DMA engines.

Version 0.2 adds detailed tests for the two FPGA receiver configurations of Test-1B and Test-1C. These tests are primarily drawn from the 2022 version of the ARRL Test Procedures Manual, April 2022, modified as appropriate for TangerineSDR receiver.

1.2. FPGA Overview

The FPGA firmware is created using Altera/Intel Quartus II version 20.1.written in Verilog, and synthesized to a binary loadable image for the FPGA. There are three main sections of FPGA firmware:

1. The digital signal processing, buffer management and signal I/O.
    a. This code will be hand written in Verilog.
2. The NIOS II processor, peripherals, DDR3RAM, QSPI flash memory, internal RAM, DMA controllers, timers, and the Ethernet interface.
    a. This code is designed using the Platform Designer (Qsys) graphical GUI tool. It is then synthesized to Verilog by the Quartus Qsys tool.
    b. About 50% of the FPGA capacity is consumed by the processor and peripherals synthesized by the Qsys tools.

3. C-code running on the NIOS processor. This software initializes the system, controls the peripherals, and processes the Local-Host-to-Data-Engine protocol packets.
    a. This code is written using the Quartus Eclipse-based design environment which provides the proper software libraries and linkage.
    b. The code relies on the Altera-supplied RTOS (uCOSII) and TCP/IP stack software (Nichestack). Nichestack is deprecated in later versions of the Quartus toolset.
    c. The NIOS executable code may be bundled with the Verilog from 1) and 2) into a single unified binary for download to the FPGA. The Quartus toolset also allows separating the software binary from 3) into a separate download should only the software portion need to be updated.

The Verilog generated for 1) and 2) above needs to co-exist in the Verilog module namespace. Traditionally this would be done by creating a top_level Verilog module that instantiates both sections (NIOS and Tangerine). Currently however Quartus requires the Qsys Verilog to be defined as the top module. To solve this problem (perhaps temporarily perhaps not) the DSP and other Verilog from section 1) in included via a file include directive inside the Qsys generated top_level so as to appear as a top level module along with the Qsys module. The Qsys top level module is called m10_rgmii.v while the module from 1) is called Tangerine.V. In this way, all the code in the Tangerine module is fully isolated from the Qsys modules and appears as a separate top_level module. If Qsys were to over-write the synthesized top_level module, only a single `include directive needs to be retyped into m10_rgmii.v

1.3. Phased Development Approach

The initial FPGA firmware and software will be targeted at debugging the receiver module using the MAX10 DevKit.

1.3.1. Phase 1A

Phase 1A will test the I2C and SPI registers on the receiver. This will demonstrate proper connection of the MAX10 development kit, adaptor board, and receiver module as well as proper communication with the FPGA using gigabit Ethernet, processing the LH-DE protocol discovery of the DevBoard, and processing of the Module Read (MR) and Module Write (MW) commands. The test should be able to:

- Acquire a DHCP address (static IP not yet supported, MAC address hard-coded).
- Respond to OpenHPSDR discovery broadcast.
- Turn each of the 6 relays on and off and turn the two LEDs on and off.
- Read the Ident PROM unique serial number.
- Read and write the ADC via the SPI interface.

1.3.2. Phase 1B

Phase 1B will be to pack roughly 1500 bytes of data from a single receiver channel into a sequence of Ethernet frames on the data socket. These frames will consist of raw ADC samples. Initially the ADC can be programmed to send various test patterns on the receiver DDR interface. This test will verify that the DDR interface is operating properly and the patterns can be properly received. Once that is achieved, the actual receiver data will be sent from one channel via the DDR interface. This will consist of a sequence of data frames that are contiguous in time for 16384 samples, then idle. These will be sent to gnuradio via a UDP socket block. The delimiting 0th sample will have a special mark in the header that allows a custom gnuradio OOT module to properly group this sequence into a single gnuradio vector. The sequence will be repeated at a slow rate of a few vectors per second. This is essentially a wideband spectrum from the receiver. A gnuradio flowgraph will be constructed to analyze these vectors to look for the presence of sampling and other spurs amid the base noise sequence.

1.3.3. Phase 1C

Phase 1C will use a NCO / Sin-Cos generator and a sequence of CIC and FIR filters to source a single downconverted RF channel to the Ethernet stream. This stream will be sent to gnuradio using a standard UDP socket where the samples can be analyzed. Gnuradio flowgraphs will be constructed to analyze the Noise Figure of the receiver at a couple of discrete RF channel frequencies. Additionally (depending on test equipment availability) the 3rd order dynamic range of the receiver will be estimated.

It is anticipated that the tests through Phase 1 will be able to use processor-directed DMA engines and FIFOs to pass a limited amount of data to the attached gnuradio platform, which is anticipated to be a Linux based host, likely a general purpose desktop computer.

1.4. Phase 2

Continued development during Phase 2 will include additional LH-DE protocol processing, multiple channels, clock module testing, etc. The needed data throughput is much higher. The details are TBD at this point.

2. DMA of Data I/O

One key architectural issue is that the dual ADC converter receiver module can produce samples at a very high rate of speed. That sample rate needs to be reduced before the frames can be sent over the 1 GbE UDP socket. The concern is that the NIOS processor may not be fast enough to handle the rate of packets that could be emitted.

Some back-of-the-envelope numbers help illustrate the issue. The two receivers are synchronously clocked at 122.88 MHz and produce 14-bit receiver samples plus over-range. For simplicity this will be transferred as a 16-bit word for each channel per clock cycle. The FPGA hardware clocks samples into the FPGA using a 16-bit wide differential DDR (Double Data Rate interface) operating at a 245.76 MHz clock rate. The total data sample rate is:

> 122.88 Ms/s * 4 bytes * 8 bits/byte = 3.932 16 Gb/sec.
> 1 / 122.88 Ms/s = 8.14 nanoseconds per 4 bytes.

The 1 GbE interface cannot stream at 4 Gb/s obviously, so the data streams need to be decimated.

Each packet needs to have a UDP socket header added (presumably from a table constructed in RAM by the NIOS processor).

The protocol spec lists 1024 bytes of data per frame as desirable. For Ethernet, a 1500-byte frame is largest standard-sized frame allowed. Jumbo frames can allow up to about 9k bytes for frame. At a speed of 1 Gb/s, the frame time is:

> 1024 bytes + 22 bytes overhead = 8.514 microseconds per Ethernet frame
> 1500 bytes + 22 bytes overhead = 12.176 microseconds
> 9000 bytes + 22 bytes overhead = 72.176 microseconds.

The NIOS processor may not be able to service an interrupt and handshake the frame each 8, or 12, or 72 microseconds. It needs to find the packet in memory, append the appropriate UDP header, then link the start address and length of the packet into the DMA controller descriptor table. After transmission (interrupt) it needs to de-link and recycle the sent buffer. With jumbo frames this problem becomes easier.

The NIOS processor has a maximum asynchronous clock speed of 160 MHz, but the actual processor core needs to run on the system clock (so that it can synchronize memory and peripheral reads and writes).

The processor is estimated to be able to process an assembly instruction roughly once each 40 nsec. period (25 MHz.) thus providing roughly:

- 212 instructions for 1024 byte data frames.
- 304 instruction cycles for normal frames.
- 1800 instruction cycles for Jumbo frames.

The actual interrupt handler must be significantly shorter in order to allow time for non-interrupt processing to occur.

2.1. Phased DMA implementation

The first prototype of the implementation will try processor-based interrupt handling of the DMA controllers to test how practical that approach is, and to quickly get to a minimal level of receiver performance in order to characterize the receiver.

The receiver characterization needs two different types of data streams:

1. Downconverted, decimated receiver data. This is used to calculate the Noise Figure (NF) of the receiver. Different frequency bands will be hand-selected to provide NF estimates across the 100 KHz to 60 MHz range of the receiver. It also provides a way to characterize the dynamic range of the receiver.
2. Wideband interrupted data. This consists of a run of 16384 contiguous time samples with a long dead time in between each run. This allows measuring the presence of spurious signals across the entire receiver frequency range.

2.2. Longer term DMA implementation

If shorter packets are needed (for example 1024 bytes) then the NIOS performance required increases. After Phase 1 it can be decided how the final DMA architecture should be done.

A phase 2 architectural approach might then be to have the NIOS processor setup all the receiver conditions (number of channels to be downconverted, frequencies, decimation rate, number of wideband channels and how they are time-gapped) and to set up buffer descriptors for all the above. Then Scatter-Gather DMA controllers would be responsible for transferring the data blocks to the Ethernet Verilog module. The Verilog would need to handle packing buffers and queuing them to the SGDMA engines, and de-queuing and recycling spent buffers.

The NIOS processor would still talk directly to the Ethernet frames via a socket interface, but it would only process command, control, and provisioning packets, not actual data packets.

3. Detailed Tests

3.1. Test Phase 1B

The test configuration 1B consists of infrequently-sent consecutive vectors of 16384 time-contiguous samples. These vectors are infrequently produced. There are three tests for this configuration:

3.1.1. ADC DC Offset Voltage

With the input to the receiver channel terminated in a 50 ohm resistor, find the mean of the samples (the average). This represents the DC offset voltage of the receiver in terms of a digital word compared to the receiver 14-bit range. This digital value can then be converted to an equivalent DC voltage as a percent of full scale (next test).

3.1.2. ADC Full Scale Signal

Connect a sinewave signal generator to the receiver under test. Set the attenuators to zero. Terminate the UDP stream in a gnuradio instance. Slowly increase the signal generator level until visual clipping of the received signal in the ADC bit stream occurs. Reduce the value until clipping stops. The voltage produces by the signal generator presents the Full-scale signal of the receiver ADC.

3.1.3. Spurious Wideband Signal

Terminate the receiver input with a 50 ohm well-shielded load. Send the samples to gnuradio and perform an FFT on those samples. Since these are real samples, the output of the FFT is mirror-symmetric. Discard the duplicate half of the sample sets resulting in a 8192-length output vector of the baseband signals. Integrate the FFT values with a bank of low-alpha IIR (approximately alpha=**TBD** (probably about 0.000001) filters until the results stabilize. Identify any large sample peaks (value **TBD** below FS).

3.2. Test Phase 1C

This test configuration consists of a continuous stream of samples that have been downconverted and decimated to 192,000 samples per second. These shall be in I/Q 32-bit floating point format. At this point the receive tests behave very much like a traditional SSB receiver. The detailed tests here come for the ARRL Test Procedure Manual 2022, and are referenced by that document’s test number and description.

3.2.1. 5.1 NOISE FLOOR TEST

This test presumes a 500 Hz audio filter. Such a filter is not present in the receiver and would need to be implemented in gnuradio software.

3.2.2. 5.2 AM RECEIVE SENSITIVITY TEST 35

The AM detector needs to be implemented in gnuradio

3.2.3. 5.3 FM SINAD AND QUIETING TEST 36

The FM detector needs to be implemented in gnuradio.

3.2.4. 5.4 RECEIVE FREQUENCY RANGE TEST 39

3.2.5. 5.4A RECEIVER PROCESSING DELAY TIME TEST 41

This test is dependent on the receiver demodulator implementation. As such the test may not provide the information needed in the application. An alternative approach is to likely needed. Of key concern for the TangerineSDR application is injecting a GPS timing mark, and identify in some manner where the equivalent sample(s) appear in the output stream at various different offset frequencies (-96ksps to 0 to +96 Ksps)

3.2.6. 5.5 FIRST IF AND IMAGE REJECTION TEST 43

3.2.7. 5.6 ANTENNA PORT ISOLATION TEST 45

This test requires that the two receivers be functional, and that is not available in the Test-Phase-1C configuration. A later configuration of the FGPA firmware will be needed to run this test.

3.2.8. 5.7 BLOCKING GAIN COMPRESSION 47

3.2.9. 5.7A RECIPROCAL MIXING 50

3.2.10. 5.8 TWO-TONE 2ND AND 3RD ORDER DYNAMIC RANGE TEST 52

3.2.11. 5.9 FM ADJACENT-CHANNEL SELECTIVITY TEST 57

3.2.12. 5.10 FM TWO-TONE 3RD ORDER DYNAMIC RANGE TEST 60

3.2.13. 5.11 AUDIO POWER OUTPUT TEST 62

This test is not appropriate as the receiver does not have a physical output stage, and cannot be conducted meaningfully

3.2.14. 5.12 AUDIO AND IF FREQUENCY RESPONSE TEST 64

3.2.15. 5.13 SQUELCH SENSITIVITY TEST 67

This test depends on the implementation of a FM detector and squelch in gnuradio and is likely not very meaningful.

3.2.16. 5.14 S METER TEST 68

The TangerineSDR does not have an S-meter. Any implementation of an S-meter function would be scaled from the ADC output levels, and thus the test is likely not meaningful

3.2.17. 5.15 IN BAND IMD TEST 70

3.2.18. 5.16 NOTCH FILTER TEST 73

This test requires the implementation of a notch filter in gnuradio, and is thus likely not meaningful.

3.2.19. 5.17 DSP NOISE REDUCTION TEST 77

This test requires the implementation of a DSP-noise-reduction filter in gnuradio, and is thus likely not meaningful.

3.2.20. 5.18 EQUIVALENT RECTANGULAR BANDWIDTH TEST 80

This test is not described in the ARRL Test Procedure Manual, it is listed as an automated test with not details or specifics. Thus by documentation it is not implementable

3.2.21. 5.19 NOISE FIGURE CALCULATION

The description of the test in the manual is not very helpful. This test should be run with a wideband calibrated noise source using the Y-factor method and using gnuradio to identify the noise-on versus the noise-off levels.

4. References

ARRL Test Procedures Manual, Revision O, April 2022