A Networked Software-Defined Radio Telemetry Receiver

by Mitchell J Grabner, Michael L Don, J Michael Zajicek, Mark D Ilg, Rex Hall, and Jonathan M Hallameyer

Approved for public release; distribution is unlimited.
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
A Networked Software-Defined Radio Telemetry Receiver

Mitchell J Grabner
ORAU, Oak Ridge, TN

Jonathan M Hallameyer
SURVICE Engineering, Belcamp, MD

Michael L Don, J Michael Zajicek, Mark D Ilg, and Rex Hall
Weapons and Materials Research Directorate, CCDC Army Research Laboratory

Approved for public release; distribution is unlimited.
### REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. **REPORT DATE** (DD-MM-YYYY)  
   July 2020

2. **REPORT TYPE**  
   Technical Report

3. **DATES COVERED (From - To)**  
   15–26 July 2019

4. **TITLE AND SUBTITLE**  
   A Networked Software-Defined Radio Telemetry Receiver

5a. **CONTRACT NUMBER**

5b. **GRANT NUMBER**

5c. **PROGRAM ELEMENT NUMBER**

6. **AUTHOR(S)**  
   Mitchell J Grabner, Michael L Don, J Michael Zajicek, Mark D Ilg, Rex Hall, and Jonathan M Hallameyer

7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**  
   CCDC Army Research Laboratory  
   ATTN: FCDD-RLW-LF  
   Aberdeen Proving Ground, MD 21005

8. **PERFORMING ORGANIZATION REPORT NUMBER**  
   ARL-TR-8987

9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

10. **SPONSOR/MONITOR'S ACRONYM(S)**

11. **SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. **DISTRIBUTION/AVAILABILITY STATEMENT**  
   Approved for public release; distribution is unlimited.

13. **SUPPLEMENTARY NOTES**  
   ORCID ID(s): Mitchell J Grabner, 0000-0003-2550-2907; Michael L Don, 0000-0002-8021-9066

14. **ABSTRACT**  
   This report discusses the implementation of a software-defined radio receiver for frequency-shift keying telemetry signals. The field-programmable gate array modifications and host software used to enable reliable telemetry frame decoding in a multireceiver distributed network framework are also presented in detail. The hardware infrastructure used to construct the network backbone for the US Army Combat Capabilities Development Command Army Research Laboratory flight test of its highly maneuverable airframe is also presented. The resulting aggregate frame error rate of three networked receivers is very low at 2.15%.

15. **SUBJECT TERMS**  
   telemetry, software-defined radio, pulse-code modulated/frequency modulation, PCM/FM, field-programmable gate array, automatic gain control

16. **SECURITY CLASSIFICATION OF:**  
   a. **REPORT** Unclassified  
   b. **ABSTRACT** Unclassified  
   c. **THIS PAGE** Unclassified

17. **LIMITATION OF ABSTRACT**  
   UU

18. **NUMBER OF PAGES**  
   72

19a. **NAME OF RESPONSIBLE PERSON**  
   Mitchell J Grabner

19b. **TELEPHONE NUMBER (Include area code)**  
   (410) 306-0775

---

Standard Form 298 (Rev. 8/98)  
Prescribed by ANSI Std. Z39.18
## Contents

List of Figures v

List of Tables vi

1. Introduction 1

2. SDR TM Receiver Overview 1

3. Software Modifications 2
   3.1 Original Accumulator Design 2
   3.2 Continuous TM Output 5
   3.3 Accumulator Design Simulation 8

4. Networked SDR Architecture 10
   4.1 Porting the FPGA Firmware 11
   4.2 ZeroMQ Frame Distribution in GNU Radio 13
   4.3 Software AGC for Reliable Frame Detection 13
   4.4 Remote TM Monitoring and Logging Front End 15

5. Highly Maneuverable Airframe Flight Experiment 15
   5.1 Ruggedized Field Network Switch 15
   5.2 Ruggedized Field TM Receiver 19
   5.3 Flight Test TM Deployment 21

6. Experimental Results 22

7. Conclusion 26

8. References 28

Appendix A. Software-Defined Radio Field-Programmable Gate Array Verilog Code 29
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Python GNU Radio E312 Initialization</td>
<td>49</td>
</tr>
<tr>
<td>C</td>
<td>Ruggedized Field Network Switch Parts List</td>
<td>57</td>
</tr>
<tr>
<td>D</td>
<td>Ruggedized Field Telemetry Receiver Parts List</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>List of Symbols, Abbreviations, and Acronyms</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>Distribution List</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Figures

Fig. 1  TM receiver B200 block diagram ................................................................. 2
Fig. 2  decom_acc1 state diagram, modified to accommodate wired inputs ......... 4
Fig. 3  decom_acc2 block diagram ......................................................................... 5
Fig. 4  decom_acc2 state diagram, modified for continuous output .................. 7
Fig. 5  Simulation of a dummy frame in decom_acc2 ........................................... 9
Fig. 6  Simulation of a dummy frame in decom_acc2; close-up view of extra words ................................................................................................................. 9
Fig. 7  Simulation of a real frame in decom_acc2 .................................................. 10
Fig. 8  Functional block diagram of the USRP E312 embedded SDR................. 11
Fig. 9  FPGA LPF magnitude response vs. baseband frequency ......................... 12
Fig. 10  FPGA LPF phase delay vs. baseband frequency ....................................... 12
Fig. 11  FPGA LPF real taps vs. baseband frequency ........................................... 13
Fig. 12  ZeroMQ TM frame distribution flow graph using GNU Radio ............... 13
Fig. 13  Software AGC control flow graph using GNU Radio message passing ................................................................. 14
Fig. 14  Visual basic/MATLAB remote monitoring front-end example ............... 15
Fig. 15  CAD model rendering of top-side internals of field network switch ... 17
Fig. 16  CAD model rendering of bottom-side internals of field network switch ................................................................. 17
Fig. 17  Picture of the top aluminum plate of the assembled field network switch ................................................................................................................. 18
Fig. 18  Picture of the exterior connectivity of the assembled field network switch ................................................................................................................. 18
Fig. 19  Field TM box block diagram .................................................................. 19
Fig. 20  Pelican ruggedized field TM receiver enclosure exterior ....................... 20
Fig. 21  Ruggedized field TM receiver aluminum mounting plate ...................... 20
Fig. 22  Ruggedized field TM receiver internal component layout .................... 21
Fig. 23  Satellite view of the HMA flight experiment installation at the trench warfare test site .................................................................................................... 22
Fig. 24  Graph of RSSI vs. time for HMA 3 gun location TM receiver .............. 23
Fig. 25  Graph of RSSI vs. time for HMA 3 mid-range TM receiver ................. 24
Fig. 26  Graph of RSSI vs. time for HMA 3 impact location TM receiver ......... 24
Fig. 27  Graph of frame counter delta vs. time for HMA 3 gun location TM receiver ................................................................................................................. 25
Fig. 28  Graph of frame counter delta vs. time for HMA 3 mid-range location TM receiver.................................................................................................................. 25

Fig. 29  Graph of frame counter delta vs. time for HMA 3 impact location TM receiver........................................................................................................... 26

List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Extra TM words</td>
<td>3</td>
</tr>
<tr>
<td>Table 2</td>
<td>Dummy frames</td>
<td>7</td>
</tr>
<tr>
<td>Table 3</td>
<td>Simulation parameters</td>
<td>8</td>
</tr>
<tr>
<td>Table 4</td>
<td>HMA flight experiment TM link budget</td>
<td>22</td>
</tr>
</tbody>
</table>
1. Introduction

The US Army Combat Capabilities Development Command (CCDC) Army Research Laboratory (ARL) has decades of experience using telemetry (TM) systems to transmit sensor data during flight tests for postprocessing and performance analysis. Unfortunately, much of CCDC Army Research Laboratory’s TM equipment is now antiquated and cumbersome to use in an age of ubiquitous network connectivity and low-cost embedded high-performance computing systems. This problem led ARL to develop new software-defined radio (SDR) solutions to old communication tasks that traditionally required expensive, standalone, application-specific hardware. An SDR TM receiver suitable for frequency-shift keying (FSK) pulse-code modulated (PCM) S-band transmitters was developed for laboratory use, including support for Advanced Encryption Standard (AES) encryption and layered protocol.\textsuperscript{1,2} This report extends previous TM receiver research to support a scalable, multireceiver networked and remotely monitored SDR framework for field deployment.

In the first section of the report, we review the preexisting receiver architecture; the remainder of the report documents the field-programmable gate array (FPGA) modifications and host software used to enable reliable TM frame decoding in a multireceiver distributed network framework, the remote monitoring and data-logging front-end software development, the hardware infrastructure used to construct the network backbone for the flight test, and the results of the TM data collection.

2. SDR TM Receiver Overview

ARL’s SDR TM receiver is based on the Ettus Research’s Universal Software Radio Peripheral (USRP) B200.\textsuperscript{3} This is a single-board SDR, using the Analog Devices (Cambridge, Massachusetts) RF integrated circuit (IC) that combines an RF front-end, in-phase/quadrature demodulator and analog-to-digital converters into a single IC that covers a range of center frequencies from 70 MHz to 6 GHz. There is an optional GPS-disciplined oscillator that can be installed on the B200 to enable global timing alignment to within 50 ns. Figure 1 shows the B200 SDR receiver architecture. Demodulation, bit synchronization, and frame synchronization modules were developed in Verilog and added to the FPGA firmware. The decimating half-band filters, which are normally required to reduce the data rate to speeds slow enough for the host computer to process, were replaced by nondecimating low-pass filters (LPFs) due to the enhanced processing capabilities of the FPGA. This allowed an increased baseband sampling resolution.
equal to the 32 MHz master clock rate of the FPGA. A LabVIEW TM display program was designed for the host computer to visualize the processed frames. A separate C++ program was written using the USRP hardware driver (UHD) to configure the USRP and route data to a user datagram protocol (UDP) port. The LabVIEW program reads the UDP port to access data from the USRP, performs frame synchronization, extracts the frame data, and displays the results. Frame synchronization is performed on the FPGA as well so that extra data, such as time and received signal strength indicator (RSSI) data, can be added to the end of each frame.

![Fig. 1 TM receiver B200 block diagram](image)

### 3. Software Modifications

#### 3.1 Original Accumulator Design

Referring back to Fig. 1, after the signal is demodulated and the bits are identified through bit synchronization, the frames are identified through a frame synchronization module. This frame synchronization module outputs 16-bit words and a strobe signal to an accumulator module, which converts the 16-bit words into a 32-bit format for transmission to the PC. Additionally, the accumulator module adds eight extra words to the end of each frame as specified in Table 1. The original Verilog code for the accumulator module is included as decom_acc in Appendix A, which operates according to the state diagram in Fig. 2. The state diagram uses the shorthand names for the RSSI and TIME signals specified in Table 1, along with D_in for data_in, D2 for data_out[31:16], and D1 for data_out[15:0]. The states are represented as circles, black text indicates the condition for state transition, and red text indicates a value change in a state, or during a state transition. The main caveat in the operation of decom_accl is that since there can be a total odd number of words per frame, and since the 16-bit input words are loaded into a 32-bit output register, a given input word will not always line up with the same 16 bits of the output register. To handle this problem, the state machine keeps track of the proper section of the output register to load, either D1 or D2.
Table 1  Extra TM words

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Shorthand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RSSI [31:16]</td>
<td>RSSI2</td>
<td>RSSI word 1</td>
</tr>
<tr>
<td>2</td>
<td>RSSI [15:0]</td>
<td>RSSI1</td>
<td>RSSI word 0</td>
</tr>
<tr>
<td>3</td>
<td>TIME [63:48]</td>
<td>TIME4</td>
<td>Timestamp word 3</td>
</tr>
<tr>
<td>4</td>
<td>TIME [47:32]</td>
<td>TIME3</td>
<td>Timestamp word 2</td>
</tr>
<tr>
<td>5</td>
<td>TIME [31:16]</td>
<td>TIME2</td>
<td>Timestamp word 1</td>
</tr>
<tr>
<td>6</td>
<td>TIME [15:0]</td>
<td>TIME1</td>
<td>Timestamp word 0</td>
</tr>
<tr>
<td>7</td>
<td>AVE</td>
<td>AVE</td>
<td>Average value of demodulated data</td>
</tr>
<tr>
<td>8</td>
<td>DIN</td>
<td>DIN</td>
<td>Digital inputs (lower byte)</td>
</tr>
</tbody>
</table>

Starting in state reset (RST), the state machine automatically transitions to the LD1 state. When the input strobe ld_in is asserted, D2 is loaded with D_in, and the state machine transitions to the WAIT1 state. A counter delays the state machine in WAIT1 for \( \text{clk\_div} + 1 \) clock cycles before transitioning to LD2, which is a sufficient period of time for ld_in to be deasserted. \( \text{clk\_div} \) is set to the number of clock cycles per PCM bit. When ld_in is asserted again, D1 is set to D_in and the strobe out signal, ld_out, is asserted, sending the full 32 bit data_out signal to the PC. ld_in also triggers a state transition to WAIT2, which serves a similar function to WAIT1. The state machine returns to LD1 from WAIT2 where the process is repeated. This process continues until a full frame of words has been processed. The assertion of lastw indicates that the current input word is the last word of the frame. If lastw is asserted in the LD1 state, the state machine transitions to RSSI10. If it is asserted in LD2, the state machine transitions to RSSI20. In both of these branches of the state machine, extra words are loaded into the output register for transmission to the PC. The branch starting with RSSI10 loads D2, since D1 was just loaded; whereas, the branch starting with RSSI20 loads D1, since D2 was just loaded. Each branch then continues, alternating between loading D1 and D2 before returning to the initial branch of the state machine. In state TIME22, data_out is fully loaded; therefore, the state machine returns to WAIT2, which will transition to LD1 and begin by loading D2 once again. In state TIME13, D2 has been loaded but not D1; therefore, the state machine returns to WAIT1 where it will transition to LD2 for D1 to be loaded.
Fig. 2  decom_acc1 state diagram, modified to accommodate wired inputs
3.2 Continuous TM Output

In its normal operating mode, the receiver is designed to output TM frames as they are received. When frames are not being detected, no data are output. This results in two main problems. First, no RSSI information is sent to the computer, making it impossible to perform any kind of automatic gain control (AGC) to improve detection accuracy. Second, the timestamping of wired signals is unreliable. Any dropped TM frames will also result in a loss of DIN data. For the computer to receive these extra words without frame detection, the receiver must be used in a simulator mode, where the SDR continually outputs simulated TM frames irrespective of the received RF signal. This is clearly impractical for implementing an AGC and also undesirable for timestamping wired signals. To fix these problems, the accumulator was modified to output data even when TM frames are not detected in its normal operation mode. Figure 3 shows a block diagram of the new accumulator, decom_acc2. The ld_in signal was modified so that it is only asserted one clock cycle, and is used to load a first-in, first-out (FIFO) buffer with the incoming TM words. When there is a full frame of words available in the FIFO, they are unloaded and sent to the computer. When there are not enough words available, a dummy frame is output. In either case, the extra words, including DIN, are output with the frames.

![decom_acc2 block diagram](image)

Figure 4 shows a state diagram of decom_acc2. The main words of the frame are handled in the MAKE_FRAME state, whose operation is briefly outlined in the diagram. Count registers cnt, bcnt, and wcnt are used to count cycles per bit, bits per word, and words per frame. high_bits is used to determine if the current word is loaded into the lower or higher bits of the 32-bit output signal and is inverted after each word. do_dummy determines if the current frame source is generated dummy frame data or real TM data from the FIFO. The FIFO is of the “first-word fall-through” variety, allowing the FIFO word to be available
immediately. The `rd` signal is asserted every time the FIFO output is used, allowing the next FIFO word to be available when needed. `fifo_cnt` is the number of words in the FIFO. At the end of each frame, `fifo_cnt` is used to determine the value of `do_dummy`. If there are sufficient words in the FIFO, `do_dummy` is deasserted. If not, `do_dummy` is asserted. The internal count registers are used now to determine transition to the extra word states instead of the external `lastw` signal. This transition occurs slightly before the end of a full frame, so that the total data rate is slightly higher than the TM data rate. This was done to ensure that the FIFO does not overflow in the case where the transmitter data rate might be slightly higher than the receiver’s expected data rate due to a mismatch between transmitter and receiver clocks. Thus, even when TM data are consistently received, a dummy frame will be occasionally output. The states for the extra words have remained generally the same, only now `high_bits` determines if the extra words begin in the higher bits of `data_out` (RSSI20) or the lower bits of `data_out` (RSSI10). `high_bits` must also be set correctly when transitioning back to the `MAKE_FRAME` state. The dummy frame format is specified in Table 2. The third word the dummy frame is set as is the subframe ID (SFID), with the upper byte specified by a configurable dummy_SFID parameter and the lower byte set to zero. The second-to-last word is a 16-bit frame counter while the last word is a checksum placeholder set to 1.
**Table 2** Dummy frames

<table>
<thead>
<tr>
<th>Word index</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SYNC [31:16]</td>
</tr>
<tr>
<td>1</td>
<td>SYNC [15:0]</td>
</tr>
<tr>
<td>2</td>
<td>{Dummy_SFID, x00}</td>
</tr>
<tr>
<td>3 ... NWORDS-3</td>
<td>Word index</td>
</tr>
<tr>
<td>NWORDS-2</td>
<td>FCNT</td>
</tr>
<tr>
<td>NWORDS-1</td>
<td>1</td>
</tr>
</tbody>
</table>
3.3 Accumulator Design Simulation

Due to the long compile times of FPGA images, simulation is a key part of FPGA design. *dcc_chain_tb_din*, a test bench for SDR receiver modifications, is also included in Appendix A. The test bench simulates at the digital down converter (DDC) level, which contains the accumulator described previously. A full explanation of the DDC is outside the scope of this report, although some aspects of the higher-level design are explained briefly. Parameters of the B200 are stored in setting registers in the FPGA and are set using the UHD. Adding additional setting registers would typically require modification and recompilation of the UHD. To avoid this, the timekeeper module was modified to allow for additional setting registers. When the 32-bit timekeeper register is loaded with x01234560, the next 32-bit load to the timekeeper, set_data[31:0], will be interpreted as a custom register load. set_data[31:28] and set_data[3:0] are ignored, set_data[27:22] is interpreted as a custom register index, and set_data[21:4] is the custom register data. The important parameters for the DDC simulation are listed in Table 3, and are loaded into custom setting registers at the beginning of the test bench using the method described. By setting sim_pcm_en, the receiver generates simulated TM frames for transmission to the PC. For our purposes, these simulated frames can take the place of frames received from the demodulator and should not be confused with the dummy frames generated in the accumulator in Fig. 4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync1</td>
<td>xFE6B</td>
<td>First synchronization word</td>
</tr>
<tr>
<td>sync0</td>
<td>x2840</td>
<td>Second synchronization word</td>
</tr>
<tr>
<td>clk_div</td>
<td>8</td>
<td>Clock cycles per bit</td>
</tr>
<tr>
<td>nbits</td>
<td>16</td>
<td>Bits per word</td>
</tr>
<tr>
<td>nwords</td>
<td>11</td>
<td>Number of words per frame - 1</td>
</tr>
<tr>
<td>sim_pcm_en</td>
<td>1</td>
<td>Output simulated frames</td>
</tr>
<tr>
<td>dummy_sfid</td>
<td>xFF</td>
<td>High byte of the third word of a dummy frame</td>
</tr>
</tbody>
</table>

Figure 5 shows a simulation of a dummy frame in *decom_acc2*. The state machine stays in the MAKE_FRAME state most of the time, which has a value of 15. Words are loaded into the FIFO using *ld_in*. Observe *fifo_cnt* increasing as the FIFO fills. Since this is a dummy frame, no words are read from the FIFO. Instead, generated dummy words are loaded into *data_out*. *high_bits* alternates as words are loaded into the high or low bits of *data_out*, and *ld_out*
is asserted as a strobe output. Note that for the PC to receive the data in big-endian format, each word is transmitted as little-endian. Also, the higher bits of \text{data\_out} are received first, and the lower bits are received last. Thus, a \text{data\_out} value of x6BFE4028 is received at the PC as xFE6B2840.

Figure 5  Simulation of a dummy frame in \text{decom\_acc2}

Figure 6 shows a close-up view of the extra words at the end of the frame. Transitioning from \text{MAKE\_FRAME} with \text{high\_bits} = 1, the state machine enters RSSI20 (state = 7). The RSSI, TIME, AVE, and DIN registers are inserted into \text{data\_out}, and strobed out with \text{ld\_out}. The state machine returns to the \text{MAKE\_FRAME} state with \text{high\_bits} active, ready to load the next word into the higher bits of \text{data\_out}.

Figure 7 shows the simulation of a real frame in \text{decom\_acc2}. \text{fifo\_cnt} shows that there is more than a full frame of words in the FIFO, which causes
do_dummy to become inactive. The FIFO is unloaded using the rd signal, and data_out is loaded with the FIFO output, fifo_dout.

4. Networked SDR Architecture

Since the USRP B200 device used in the original TM receiver communicates to the host device using USB3, a new SDR is needed to enable networked TM monitoring over Ethernet for proper field deployment. The device chosen for this task is the USRP E312 embedded series SDR. The E312 contains a Xilinx ZYNQ 7020 SoC, which integrates an FPGA and a dual-core ARM CPU onto a single package. The functional block diagram of the USRP E312 can be seen in Fig. 8. This allows any additional processing that may have been done on a connected host device such as additional frame synchronization or AGC to be self-contained on the SDR hardware. Additionally, the decoded TM frames can be broadcasted over the E312’s Ethernet port for remote monitoring and logging.
4.1 Porting the FPGA Firmware

Because the TM receiver is implemented in the FPGA of the B200 for the increased processing power, implementing the baseband processing on the E312 is more difficult than simply recompiling a C++ code repository and running it on the embedded ARM processor. Since the FPGA image for each USRP is device specific, the firmware source code for the E312 FPGA needed to be downloaded and compiled separately from the B200 code used previously. Once basic functionality of the source-build E312 FPGA image was verified as identical to the manufacturer-supplied image, the modified accumulator code with continuous output was then integrated into the DDC functionality of the E312. The same simulation test bench seen in Figs. 5–7 was used to verify accumulator behavior before implementation on the physical device. Since the FPGA source on the E312 used a newer Xilinx Vivado license, the included LPF implementation used in place of the half-band decimating filters had been deprecated and had to be redesigned. The new filter implemented in the FPGA is a fixed point direct form II, flat phase, finite impulse response (FIR) LPF. The filter sample rate is 32 MHz, which matches the master clock rate of the FPGA. However, since the FPGA master clock rate is programmable, new filter coefficients would need to be chosen if this value changes. The end of the pass band is 2.2 MHz and the start of the stop band is 3.6 MHz with 20 dB stop band attenuation using 21 real taps and an overall filter gain of 2. This puts the –3 dB point of the magnitude response at 2.7 MHz. This filter response was chosen to maintain the baseband signal bandwidth of 4 MHz while removing as much out-of-band noise energy as possible. The input data resolution of the input baseband signal is 24 bits. The magnitude response, phase delay, and filter taps for the filter can be seen in Figs. 9–11, respectively.
Fig. 9   FPGA LPF magnitude response vs. baseband frequency

Fig. 10   FPGA LPF phase delay vs. baseband frequency
4.2 ZeroMQ Frame Distribution in GNU Radio

Because the E312 TM receiver is meant to be field deployed and remotely monitored, a new host program needs to be used to distribute the TM frame data. This host program is run on the embedded ARM processor and implements a distributed messaging protocol called ZeroMQ. A GNU Radio flow graph was developed to take the 16-bit words from the radio module and stream them over Ethernet using ZeroMQ. The flow graph in Fig. 12 first initializes the radio using a UHD USRP source block and then sends two words (32 bits) at a time over transmission control protocol (TCP) port 9999 using the ZMQ PUSH sink block. The Python source code for the E312 initialization program can be seen in Appendix B.

4.3 Software AGC for Reliable Frame Detection

For the TM receiver to properly decode frames, the RSSI in the radio needs to be kept within a 25-dB window. Additionally, the radio needs to quickly respond to
the large spike in signal strength after projectile launch and the logarithmic decay of the signal power as the projectile travels downrange. A proportional-integral-derivative (PID) AGC system is developed in software to meet these requirements. The continuous time PID equation\(^7\) can be written as

\[
u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt},
\]

where the control variable is \(u(t)\), the proportional gain is \(K_p\), the integral gain is \(K_i\), and the derivative gain is \(K_d\). The error value \(e(t)\) is calculated by \(e(t) = r(t) - y(t)\) where the set point is \(r(t)\) and the process variable is \(y(t)\). To implement the PID control in software, Eq. 1 needs to be discretized into the form

\[
u[n] = K_p e[n] + K_i \sum_{k=0}^{N} e[k] \Delta t + K_d \frac{e[n] - e[n - 1]}{\Delta t},
\]

where the sampling time between error measurements in seconds is \(\Delta t\) and the current measurement index is \(n\). For the AGC system, the RSSI error in decibels is \(e[n]\), which makes \(r[n]\) the target RSSI and \(y[n]\) is the current RSSI value. The radio gain is changed by \(u[n]\) after each PID iteration to complete the closed-loop first-order control. The PID AGC block added to the flow graph in Fig. 12 can be seen in Fig. 13.

![Fig. 13 Software AGC control flow graph using GNU Radio message passing](image)

The block first serializes the incoming words and then checks for the sync word of the TM frame. Once a frame is found, the RSSI information is extracted and averaged over a specified number of frames. Changing the number of averaged frames will change the PID sampling time \(\Delta t\). The gain values for the PID system are \(K_p = 0.6\) and \(K_i = 0.0\) and \(K_d = 0.0067\), which are manually tuned to minimize rise time and overshoot\(^7\) to begin receiving valid frames quickly and to drop as few
frames as possible after launch, respectively. The integral gain is ignored entirely to avoid large error buildup between round loading and gun launch. Simply clipping the integral component would also work but reduce setting time at the expense of rise time. The sampling interval $\Delta t$ is around 20 ms for 100 averaged frames to give a fast control rate without losing stability.

### 4.4 Remote TM Monitoring and Logging Front End

The remote monitoring and logging of the TM frames is handled using the ZeroMQ distributed messaging protocol in a visual basic GUI front end. The front end is capable of saving large amounts of frame data from any number of networked SDR receivers as well as plotting RSSI and frame counter information in real time. The frame data are saved in .mat files for easy postprocessing in MATLAB. An example of the monitoring front-end functionality can be seen in Fig. 14.

![Visual basic/MATLAB remote monitoring front-end example](image)

**Fig. 14** Visual basic/MATLAB remote monitoring front-end example

### 5. Highly Maneuverable Airframe Flight Experiment

The highly maneuverable airframe (HMA) flight experiments were conducted by ARL from July 15–26, 2019, at the trench warfare test facility located at Aberdeen Proving Ground (APG). The experiment consisted of eight flight tests using eight airframes developed by the Guidance Technology Branch (GTB). The TM field deployment equipment used in these tests consists of the field network switch and field TM receiver.

#### 5.1 Ruggedized Field Network Switch

In support of the multiple prescribed flight experiments, an urgent requirement for field-expedient network connectivity was identified. The switch design needed to
be versatile enough to support Wi-Fi access points, TM radios, and assured power at experimental facilities that lack existing communication infrastructure. Field network switches are capable of transmitting and receiving power and data over long ranges from different locations where smart projectiles are tested. With the addition of this capability, the GTB would be able to acquire all data necessary in support of flight experiments in a uniform and timely fashion. All items used in this configuration were bought commercially except for the mounting brackets that were rapid prototyped using a 3-D printer. The full parts list of the field network switch is included in Appendix C.

A durable Pelican 1430 Protector Top Loader Case (Pelican Products, Inc., Torrance, California) was chosen for its polypropylene exterior, O-ring seal, and double-throw latches. This particular case proved useful in keeping the internal network switch and other electrical components safe from any particulates or unexpected weather hazards encountered during the flight experiments. For the internals, an Ubiquiti Networks (New York, New York) EdgeSwitch 8, 8-port gigabit switch was chosen for its power over Ethernet capability and support for high-speed fiber connectivity. The field network switch can be powered one of three different ways: via an external generator, a 65-W 5-A solar-charge controller, or internal batteries. The battery interface consists of three lithium 12.8-V, 6.6-Ah battery packs routed through a 48V 100W DC/DC converter. A 3.0-A, 12.8-V Smart Charger was incorporated to keep the batteries fully charged in case of generator or solar power failure.

The computer-aided design (CAD) drawings pictured in Figs. 15 and 16 show the internal electronic components mounted to two aluminum plates that fit comfortably inside the external enclosure. The top aluminum plate seen in Fig. 17 is fitted with two weather-tight Ethernet ports and a digital voltmeter controlled with a panel-mounted push-button switch to verify the charge state of the batteries. The external Ethernet, fiber optics, and power ports seen in Fig. 18 all use the RJ45 waterproof connection for enhanced durability.
Fig. 15  CAD model rendering of top-side internals of field network switch

Fig. 16  CAD model rendering of bottom-side internals of field network switch
The field fiber box was designed to have three distinct operating modes controlled by a three-position rotary knob: on, off/charge, and storage. Once the field network switches and corresponding cables were fabricated, functionality was verified via a series of electrical tests. All single and multimode fiber-optic cables and ports were
tested with a fiber scope and connectivity was verified using the network switch and an SDR. Ethernet cables and ports were tested with a cable tester and connectivity was again verified with the network switch. The battery charging circuitry was tested by plugging the switch into a 110-VAC outlet while monitoring the battery voltage and charging current with a multimeter. The internal batteries were further tested by performing several charge/discharge cycles using the internal electronics.

### 5.2 Ruggedized Field TM Receiver

The field TM box was used to house the networked SDR receiver and supporting RF components in conjunction with the field fiber box that supplied connectivity. Figure 19 shows a block diagram of the full configuration.

![Field TM box block diagram](image)

The box was built using a Pelican 1400 series weatherproof case, which can be seen in Fig. 20.
The Pelican case internal components were fixed to a piece of 1/8-inch aluminum plate mounted to a panel frame inside the box (Fig. 21).

An RG405 0.086-inch RF coaxial semi-rigid cable was used to connect the internal components. This cable was chosen because it had an impedance of 50 ohms, a max frequency of 40 GHz, a solid center conductor, and a maximum operating...
temperature of 125 °F. The surface-mounted RF input connector we used was a Belden (St Louis, Missouri) TBCF81 RF/coaxial connector with an impedance of 75 ohms and maximum frequency of 3 GHz. The other internal RF connectors used were SMA male to SMA male right-angle connectors with an impedance of 50 ohms. The active RF components consisted of a PE15A63007 low-noise amplifier (LNA) and ZFSWA2-63DR+ SPST RF switch connected to a BW-S15W2 15-dB 50-ohm attenuator. The 12V out surface-mounted HD-BNC to HD-BNC RG58 connector was chosen for this application. Lastly, we went with a modular plug connector 8p8c (RJ45, Ethernet) position-shielded Cat5e insulation displacement connector. This TM box was powered over Ethernet by the field network box feeding a PT-POE-171S power-over-Ethernet switch. These components, including the USRP SDR, are mounted on the back of the aluminum plate and can be seen in Fig. 22. The full parts list of the field TM receiver is included in Appendix D.

Fig. 22 Ruggedized field TM receiver internal component layout

5.3 Flight Test TM Deployment

A total of three networked TM receivers were deployed during the HMA flight experiment at the trench warfare test site at APG. The installation locations of the three receivers relative to the flight experiment can be seen in Fig. 23. The deployed receivers are referenced as gun, impact, and mid-range (right side) throughout the report. The planned mid-range left-side receiver was not used for this flight experiment.
6. Experimental Results

The estimated RF link budget for the HMA flight experiment TM link can be seen in Table 4. The good RX sensitivity of $-80 \text{ dBm}$ and a calculated minimum received power of $-18.8 \text{ dB}$ means the receiver should have sufficient signal-to-noise ratio (SNR) throughout the flight.

**Table 4  HMA flight experiment TM link budget**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>2255.5 MHz</td>
<td>TM RF center frequency</td>
</tr>
<tr>
<td>TX power</td>
<td>27 dBm</td>
<td>Measured power output of the HMA TX radio</td>
</tr>
<tr>
<td>TX antenna gain</td>
<td>$-10 \text{ dB}$</td>
<td>Average far-field gain of the HMA TX antenna</td>
</tr>
<tr>
<td>FSPL (min)</td>
<td>53 dB</td>
<td>Minimum free space path loss for flight experiments</td>
</tr>
<tr>
<td>FSPL (max)</td>
<td>98.8 dB</td>
<td>Maximum free space path loss for flight experiments</td>
</tr>
<tr>
<td>LNA gain (max)</td>
<td>30 dB</td>
<td>Gain of the PE15A630007 LNA</td>
</tr>
<tr>
<td>Cable loss</td>
<td>25 dB</td>
<td>Loss from wired connections</td>
</tr>
<tr>
<td>RX gain range</td>
<td>0–55 dB</td>
<td>Possible gain value of the AD9361 RFIC (clipped)</td>
</tr>
<tr>
<td>RX antenna gain</td>
<td>+3–13 dB</td>
<td>Far-field gain range of the 6H2223 RX antenna</td>
</tr>
<tr>
<td>RX sensitivity</td>
<td>$-80 \text{ dBm}$</td>
<td>Lowest receivable signal power</td>
</tr>
</tbody>
</table>

Note: TM = telemetry; RF = radio frequency; TX = transmit; HMA = highly maneuverable airframe; FSPL = free space path loss; LNA = low-noise amplifier; RX = receive; RFIC = radio-frequency integrated circuit.
The RSSI values from the HMA 3 launch test can be seen in Figs. 24–26 for TM receivers at the gun, mid-range (right side), and at impact, respectively. We can see that the PID RSSI control at the gun had the best performance locking to the target 77-dB RSSI in under 250 ms and only losing lock briefly at 4 s, and at terminal when the gain value was clipped due to the high path loss and poor line of sight to the projectile. The mid-range receiver exhibited oscillation behavior because the receiver was facing the side of the round for the duration of the experiment. The oscillation rate closely matches the 10-Hz rotation rate of the projectile, which was launched without deployable canards. The PID will need to be tuned further to deal with this rotation if another HMA launch is conducted without deployable canards. The impact receiver likely experienced problems for the first half of the launch thanks to the overly conservative maximum gain value set in the PID feedback. This conservative max gain of 55 was chosen to prevent damaging the radio during gun launch, which quickly spikes the signal power by 30 dB. A higher max gain in the feedback will need to be chosen to solve this problem. If this value is also made tunable in software, we can set different values per receiver at runtime, which will avoid damaging the receiver at the gun, which is most likely to experience high-input power at the time of launch. The frame drop graphs for the three receivers can be seen in Figs. 24–29. Frame counter deltas greater than 1 indicate $N - 1$ dropped frames.

![Graph of RSSI vs. time for HMA 3 gun location TM receiver](image)

**Fig. 24** Graph of RSSI vs. time for HMA 3 gun location TM receiver
Fig. 25 Graph of RSSI vs. time for HMA 3 mid-range TM receiver

Fig. 26 Graph of RSSI vs. time for HMA 3 impact location TM receiver
Fig. 27  Graph of frame counter delta vs. time for HMA 3 gun location TM receiver

Fig. 28  Graph of frame counter delta vs. time for HMA 3 mid-range location TM receiver
By aggregating all of the received frames from the three receivers and finding any missing frame numbers we can find the frame error rate (FER) in percent by computing \( \frac{N_e}{N_F} \times 100 \) where the number of missed frames is \( N_e \) and the total number of frames is \( N_F \). Using this method, the FER is found to be around 2% with most of the errors occurring at the very end of flight.

7. Conclusion

In summary, this report has discussed the successful development, integration, and deployment of a networked SDR receiver for FSK S-band TM signals. The field experiment of the receivers, conducted during the GTB flight test of the HMA, was very successful with an overall FER of 2.15% from three receivers.

Further research and development to be conducted to improve the performance of the TM receiver design includes the following:

1) Increasing the FPGA digital baseband dynamic range to take full advantage of the RF front-end sensitivity, which will also relax the AGC rise-time requirements and leading to better steady-state error performance.
2) Implementing a more robust AGC algorithm, specifically a clipped integral error component, will allow for improved steady-state response.

3) Implementing a digital frequency-locked loop will remove any frequency offset due to clock mismatch and drift between the transmitter and receiver.

4) Implementing forward error correction coding and exploiting the multiple-input multiple-output (MIMO) capabilities of the E312 to provide diversity gain at the receiver, which will lead to a reduced error rate at low SNR and provide mitigation from channel fading effects.
8. References


Appendix A. Software-Defined Radio Field-Programmable Gate Array Verilog Code

This appendix appears in its original form, without editorial change.
This appendix includes the following Verilog files:

1) decom_acc1: the extra digital inputs (DIN) word added
2) decom_acc2: continuous output added
3) dcc_chain_tb_din: the test bench

//take 16 bit words, and load into 32 samples to output to PC
//add on extra words at end of each frame, including din

module decom_accl(
  clk,
  reset,
  data_in,
  ld_in,
  data_out,
  ld_out,
  clk_div,
  rssi,
  lastw,
  time_in,
  ave_in,
  din
);

input clk;
input reset;
input [15:0] data_in;
input ld_in;
output [31:0] data_out;
output ld_out;
input [5:0] clk_div;
input [31:0] rssi;
input lastw;
input [63:0] time_in;
input [15:0] ave_in;
input [7:0] din;
wire clk;
wire [15:0] data_in;
wire ld_in;
wire reset;
reg [31:0] data_out;
reg ld_out;
wire [5:0] clk_div;
wire [31:0] rssi;
reg [31:0] rssi_reg;
wire lastw;
wire [63:0] time_in;
wire [15:0] ave_in;

reg [15:0] ave_reg;
reg [63:0] time_reg;
reg from_LD1;

integer cnt;

parameter [3:0]
  RST = 0,
  LD1 = 1,
  LD2 = 2,
  WAIT1 = 3,
  WAIT2 = 4,
  DO_RSSI10 = 5,
  DO_RSSI11 = 6,
  DO_RSSI20 = 7,
  DO_RSSI21 = 8,
  DO_TIME10 = 9,
  DO_TIME11 = 10,
  DO_TIME12 = 11,
  DO_TIME20 = 12,
  DO_TIME21 = 13,
  DO_TIME22 = 14,
DO_TIME13 = 15;

reg [3:0] state;

always @(posedge clk) begin : P1

if((reset == 1'b 1)) begin
    state <= RST;
end
else begin

    case(state)
    RST : begin
        cnt <= 0;
        data_out <= 0;
        state <= LD1;
        time_reg <= 64'd0;
        ave_reg <= 16'd0;
        from_LD1 <= 0;
    end
    LD1 : begin  //load one 16 bit word
        rssi_reg<=rssi;
        ld_out <= 1'b0;
        if(ld_in == 1'b1) begin
            data_out[15:0] <= {data_in[7:0],data_in[15:8]};
            if (lastw == 1'b0)
                state <= WAIT1;
            else begin
                time_reg<=time_in;
                ave_reg<=ave_in;
                state <= DO_RSSI10;
            end
        end
    end
    WAIT1 : begin  //wait for load signal to go low
        if(cnt == clk_div) begin
            // rest of the code...
        end
        // rest of the code...
    end

end
cnt <= 0;
state <= LD2;

end else begin
  cnt <= cnt+1;
end

end

LD2 : begin

rssi_reg<=rssi; // assert load out to load out 32 bit value

if(ld_in == 1'b1) begin
  data_out[15:0] <= {data_in[7:0],data_in[15:8]};
  ld_out <= 1'b1;
  if (lastw == 1'b0)
    state <= WAIT2;
  else begin
    time_reg<=time_in;
    state <= DO_RSSI20;
  end
end

end

WAIT2: begin

ld_out <= 1'b0;

if(cnt == clk_div) begin
  cnt <= 0;
  state <= LD1;
end else begin
  cnt <= cnt+1;
end

end

DO_RSSI10: begin // now do next 2cd with ld out

  data_out[15:0] <= {rssi_reg[23:16],rssi_reg[31:24]};
  ld_out <= 1'b1;
  state <= DO_RSSI11;
end

DO_RSSI11: begin // now do time, start with 2cd slot

  data_out[31:16] <= {rssi_reg[7:0],rssi_reg[15:8]};
ld_out <= 1'b0;
state <= DO_TIME10;
end

DO_RSSI20: begin  //This is 1st slot
  data_out[31:16] <= {rssi_reg[23:16],rssi_reg[31:24]};
  ld_out <= 1'b0;
  state <= DO_RSSI21;
end

DO_RSSI21: begin  // now do time, start with 1st slot
  data_out[15:0] <= {rssi_reg[7:0],rssi_reg[15:8]};
  ld_out <= 1'b1;
  state <= DO_TIME20;
end

DO_TIME10: begin  // now do next 2cd with ld out
  data_out[15:0] <= {time_reg[55:48],time_reg[63:56]};
  ld_out <= 1'b1;
  state <= DO_TIME11;
end

DO_TIME11: begin  //Do whole 32 bit value and ld out
  data_out[31:16] <= {time_reg[39:32],time_reg[47:40]};
  data_out[15:0] <= {time_reg[23:16],time_reg[31:24]};
  ld_out <= 1'b1;
  state <= DO_TIME12;
end

DO_TIME12: begin  //32 bit value
  data_out[31:16] <= {time_reg[7:0],time_reg[15:8]};
  data_out[15:0] <= {ave_reg[7:0],ave_reg[15:8]};
  ld_out <= 1'b1;
  state <= DO_TIME13;
end

DO_TIME13: begin  //This is 1st value, do back to words in 2cd slot
  data_out[31:16] <= {din[7:0],8'b00000000};
  ld_out <= 1'b0;
  state <= WAIT1;
end
DO_TIME20: begin  // now do next 2cd with ld out
    data_out[31:16] <= {time_reg[55:48],time_reg[63:56]};
    data_out[15:0] <= {time_reg[39:32],time_reg[47:40]};
    ld_out <= 1'b1;
    state <= DO_TIME21;
end

DO_TIME21: begin  // Do 32 bit value
    data_out[31:16] <= {time_reg[23:16],time_reg[31:24]};
    data_out[15:0] <= {time_reg[7:0],time_reg[15:8]};
    ld_out <= 1'b1;
    state <= DO_TIME22;
end

DO_TIME22: begin  // 32 bits, go back to words in 1st slot
    data_out[31:16] <= {ave_reg[7:0],ave_reg[15:8]};
    data_out[15:0] <= {din[7:0],8'b00000000};
    ld_out <= 1'b1;
    state <= WAIT2;
end

default: begin
    state <= RST;
end
endcase
end
end
endmodule

// take 16 bit words, and load into 32 samples to output to PC
// add on extra words at end of each frame, including din
// continuous data output
module decom_acc2(
    clk,
    reset,
    data_in,
ld_in,
data_out,
ld_out,
clk_div,
rssi,
lastw,
time_in,
ave_in,
din,
nbits,
nwords,
sync,
dummy_sfid
);

input clk;
input reset;
input [15:0] data_in;
input ld_in;
output [31:0] data_out;
output ld_out;
input [5:0] clk_div; // clock cycles per bit
input [31:0] rssi;
input lastw;
input [63:0] time_in;
input [15:0] ave_in;
input [7:0] din;
input [4:0] nbits; // bits per word
input [8:0] nwords; // words per frame - 1
input [31:0] sync;
input [7:0] dummy_sfid;

wire clk;
wire [15:0] data_in;
wire ld_in;
wire reset;
reg [31:0] data_out;
reg ld_out;
wire [5:0] clk_div;
wire [31:0] rssi;
reg [31:0] rssi_reg;
wire lastw;
wire [63:0] time_in;
wire [15:0] ave_in;
wire [4:0] nbits;
wire [8:0] nwords;
wire [31:0] sync;
wire [7:0] dummy_sfid;

reg [15:0] ave_reg;
reg [63:0] time_reg;
reg from_LD1;
reg high_bits;
reg do_dummy;

reg [15:0] dummy;

integer cnt; //cnt clk cycles for 1 bit
integer bcnt; //cnt bits in a word
integer wcnt; //cnt words in frame
integer fcnt; //cnt frames

parameter [4:0]
  RST = 0,
  LD1 = 1,
  LD2 = 2,
  WAIT1 = 3,
  WAIT2 = 4,
  DO_RSSI10 = 5,
  DO_RSSI11 = 6,
  DO_RSSI20 = 7,
  DO_RSSI21 = 8,
wire [7:0] fifo_cnt;
wire [15:0] fifo_dout;
reg rd;

//incoming frame fifo
ddc_output_fifo ddc_output_fifo1 (.clk(clk), .rst(reset), .din(data_in), .wr_en(ld_in), .rd_en(rd),
.dout(fifo_dout), .full(), .empty(), .data_count(fifo_cnt));

reg [3:0] state;

always @(posedge clk) begin : P1

if((reset == 1'b 1)) begin
  state <= RST;
end
else begin

  case(state)
  RST : begin
    fcnt<=0;
    do_dummy<=1;
    cnt <= 0;
    bcnt<=0;
    wcnt<=0;
    data_out <= 0;
  endcase

end
state <= MAKE_FRAME;
time_reg <= 64'd0;
ave_reg <= 16'd0;
from_LD1 <= 0;
rssi_reg <= 32'd0;
high_bits <= 1; // first data load will be high bits
end

// cnt - counts clks; bcnt - counts bits; wcnt - counts words
// each increments when one below reaches max value
// dummy word set with wct - will change 1 cycle after wcnt changes
MAKE_FRAME : begin
    // run clock counter
    if (cnt == clk_div - 1) // clk_div is cycles per bit
        cnt <= 0;
    else
        cnt <= cnt + 1;

    // run bit counter
    if (cnt == clk_div - 1)
        if (bcnt == nbits - 1) // nbits is bits per word
            bcnt <= 0;
        else
            bcnt <= bcnt + 1;

    // run word counter
    if (bcnt == nbits - 1 && cnt == clk_div - 1)
        if (wcnt == nwords) // nwords is words per frame - 1
            wcnt <= 0;
        else
            wcnt <= wcnt + 1;

    case(wcnt)
        0 : dummy <= sync[31:16];
        1 : dummy <= sync[15:0];
2 : dummy<=\{dummy_sf_id,8'd0\};
(nwords-1) : dummy<=fcnt;
nwords : dummy<=1;
default : dummy<=wcnt;
endcase

//make frame counter
if (cnt == 0 & & bcnt == 0 & & wcnt == 0)
  if (fcnt == 65535)
    fcnt<=0;
  else
    if (do_dummy == 1)
      fcnt<=fcnt+1;

//output data
if (cnt == 0 & & bcnt == 1) begin
  if (high_bits == 1) begin
    if (do_dummy == 1) begin
      data_out[31:16]<=\{dummy[7:0],dummy[15:8]\};
      ld_out <= 1'b0;
    end
    else begin
      data_out[31:16]<=\{fifo_dout[7:0],fifo_dout[15:8]\};
      ld_out <= 1'b0;
      rd<=1;
    end
  end
  end else begin
    if (do_dummy == 1) begin
      data_out[15:0]<=\{dummy[7:0],dummy[15:8]\};
      ld_out <= 1'b1;
    end
    else begin
      data_out[15:0]<=\{fifo_dout[7:0],fifo_dout[15:8]\};
      ld_out <= 1'b1;
      rd<=1;
    end
  end
end
high_bits<=~high_bits;
end else begin
    ld_out <= 1'b0;
    rd <= 1'b0;
end

//state transition
//there will be at most 5 cycles to do extra words
//want full frame period to be slight less than full period
//so have some dummy frames even when getting data
//make sure the FIFO is kept empty
//state change at nbits-4 will slowly empty fifo
if ((cnt == clk_div-1) && (bcnt == nbits-4) && (wcnt == nwords))
begin
    time_reg<=time_in;
rssi_reg<=rssi;
ave_reg<=ave_in;
    if (high_bits==1) //this means that just did low
        state <= DO_RSSI20;
    else
        state <= DO_RSSI10;
end

DO_RSSI10: begin // now do next 2cd with ld out
    data_out[15:0] <= {rssi_reg[23:16],rssi_reg[31:24]};
    ld_out <= 1'b1;
    state <= DO_RSSI11;
end

DO_RSSI11: begin // now do time, start with 2cd slot
    data_out[31:16] <= {rssi_reg[7:0],rssi_reg[15:8]};
    ld_out <= 1'b0;
    state <= DO_TIME10;
end

DO_RSSI20: begin //This is 1st slot
    data_out[31:16] <= {rssi_reg[23:16],rssi_reg[31:24]};
    ld_out <= 1'b0;
state <= DO_RSSI21;
end

DO_RSSI21: begin //now do time, start with 1st slot
    data_out[15:0] <= {rssi_reg[7:0],rssi_reg[15:8]};
    ld_out <= 1'b1;
    state <= DO_TIME10;
end

DO_TIME10: begin // now do next 2cd with ld out
    data_out[15:0] <= {time_reg[55:48],time_reg[63:56]};
    ld_out <= 1'b1;
    state <= DO_TIME11;
end

DO_TIME11: begin //Do whole 32 bit value and ld out
    data_out[31:16] <= {time_reg[39:32],time_reg[47:40]};
    data_out[15:0] <= {time_reg[23:16],time_reg[31:24]};
    ld_out <= 1'b1;
    state <= DO_TIME12;
end

DO_TIME12: begin //This is 1st value
    data_out[31:16] <= {time_reg[7:0],time_reg[15:8]};
    data_out[15:0] <= {ave_reg[7:0],ave_reg[15:8]};
    ld_out <= 1'b1;
    state <= DO_TIME13;
end

DO_TIME13: begin //This is 1st value, do back to words in 2cd slot
    data_out[31:16] <= {din[7:0],8'b00000000};
    ld_out <= 1'b0;
    state <= MAKE_FRAME;
    high_bits<=0;
if (fifo_cnt > nwords) begin
    do_dummy<=0;
    cnt <= 0;
    bcnt<=0;
end
wcnt<=0;
end else
do_dummy<=1;
end

DO_TIME20: begin // now do next 2cd with ld out
data_out[31:16] <= {time_reg[55:48],time_reg[63:56]};
data_out[15:0] <= {time_reg[39:32],time_reg[47:40]};
ld_out <= 1'b1;
state <= DO_TIME21;
end

DO_TIME21: begin // Do 32 bit value and ld out
data_out[31:16] <= {time_reg[23:16],time_reg[31:24]};
data_out[15:0] <= {time_reg[7:0],time_reg[15:8]};
ld_out <= 1'b1;
state <= DO_TIME22;
end

DO_TIME22: begin //32 bit value, go back to MAKE_FRAME in 2cd slot
data_out[31:16] <= {ave_reg[7:0],ave_reg[15:8]};
data_out[15:0] <= {din[7:0],8'b00000000};
ld_out <= 1'b1;
high_bits<=1;
state <= MAKE_FRAME;
if (fifo_cnt > nwords) begin
  do_dummy<=0;
cnt <= 0;
bcnt<=0;
wcnt<=0;
end else
  do_dummy<=1;
end

default : begin
  state <= RST;
end
endcase
// Testbench to test the DIN function and cont. output of the dcc

`timescale 1ns / 1ps

module dcc_chain_tb_din;

localparam SR_RX_DSP = 8'd144;
localparam SR_TIME = 8'd100;

reg clk = 0;
reg reset = 1;
reg run = 0;
wire strobe;
reg [23:0] rx_fe_i, rx_fe_q, debug_reg;
integer i, i2;
reg [1:0] pcm_in = 2'b00;
wire [2:0] scale_rx, scale_rx2;
wire [3:0] half_clk_div;
wire [8:0] nwords;
wire external_pcm_en, sim_pcm_en, randomized, use_filt_10;
wire sync_select, swap_bytes, en_crc;
wire [1:0] sync_size;
wire [4:0] nbits;
wire [7:0] dummy_sfid;
wire [15:0] sync0, sync1;

// Telemetry parameters:
assign sync0 = 16'hfe6b;
assign sync1 = 16'h2840;
assign half_clk_div = 4'd4;
assign nwords = 9'd11; // nwords is really nwords-1, nwords=47 gives 48 words
assign external_pcm_en = 1'b0;
assign sim_pcm_en = 1'b1;
assign randomized = 1'b0;
assign use_filt_10 = 1'b0;
assign sync_select = 1'b0;
assign scale_rx = 3'd1;
assign swap_bytes = 1'b0;
assign scale_rx2 = 3'd1;
assign en_crc = 1'b0;
assign decrypt = 1'b0;
assign sync_size = 2'd3; //3 = 32, 2=24
assign nbits = 5'd16;
assign dummy_sfid = 8'hFF;

always #10 clk =~ clk;

initial
begin
  rx_fe_i <= 24'b001000000000000000000000;
  rx_fe_q <= 24'b001000000000000000000000;
  #1000 reset = 0;
  @(posedge clk);
  set_addr <= 8'd144; set_data <= 32'd8434349; set_stb <= 1;
  @(posedge clk); // CORDIC
  set_addr <= 8'd145; set_data <= 18'd19800; set_stb <= 1;
  @(posedge clk);
  set_addr <= 8'd146; set_data <= {1'b1, 1'b1, 1'b1, 1'b0, 6'd47};
  set_stb <= 1;
  @(posedge clk); // {enable_hb1_real, enable_hb2_real,
  cic_decim_rate_real}
  set_addr <= 8'd147; set_data <= 0; set_stb <= 1;
  @(posedge clk); // Swap iq
  set_addr <= 8'd148; set_data <= 0; set_stb <= 1;
  @(posedge clk); // filter taps
  set_addr <= 8'd186; set_data <= {1'b1, 1'b1, 4'd0, 4'd4};
  set_stb <= 1; @(posedge clk); // {enable_hb1, enable_hb2,
  interp_rate_duc}
set_addr <= 8'd128; set_data <= 32'hF001F002; set_stb <= 1;
@(posedge clk);

// Set config regs using timekeeper:
// 4 upper blank, next 6 address, next 18 data, next 4 blank

// sync0
set_addr <= 8'd101; set_data <= 32'h01234560;
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);
set_addr <= 8'd101; set_data <= {4'd0, 6'd0, 2'b0, sync0, 4'd0};
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);

// sync1
set_addr <= 8'd101; set_data <= 32'h01234560;
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);
set_addr <= 8'd101; set_data <= {4'd0, 6'd1, 2'b0, sync1, 4'd0};
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);

// config2
set_addr <= 8'd101; set_data <= 32'h01234560;
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);
set_addr <= 8'd101; set_data <= {4'd0, 6'd2, sync_select, use_filt_10, randomized, sim_pcm_en, external_pcm_en, nwords, half_clk_div, 4'd0};
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);

// config3
set_addr <= 8'd101; set_data <= 32'h01234560;
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);
set_addr <= 8'd101; set_data <= {4'd0, 6'd3, 7'd0, decrypt, en_crc, scale_rx2, swap_bytes, sync_size, scale_rx, 4'd0};
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);

// set config4 last - triggers reset
set_addr <= 8'd101; set_data <= 32'h01234560;
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);
set_addr <= 8'd101; set_data <=
{4'd0,6'd26,5'd0,dummy_sfid,nbits,4'd0};
set_stb <= 1; @(posedge clk); set_stb <= 0; @(posedge clk);

repeat(10) @(posedge clk);
run <= 1'b1;
#
4000000;
$finish;
end

reg [7:0] set_addr;
reg [31:0] set_data;
reg set_stb = 1'b0;
wire [7:0] ddc_debug;
wire [15:0] i_out, q_out;
wire fm_out;
wire [437:0] config_reg;
wire [31:0] debug;
reg [63:0] time_reg;
reg [7:0] din = 8'b10101111;

ddc_chain_i3i5p7 #( .BASE(SR_RX_DSP), .DSPNO(0), .WIDTH(24)) ddc_chain
(.clk(clk), .rst(reset), .clr(1'b0),
 .set_stb(set_stb), .set_addr(set_addr), .set_data(set_data),
 .rx_fe_i(rx_fe_i), .rx_fe_q(rx_fe_q),
 .sample({i_out,q_out}), .run(run), .strobe(strobe),
 .ddc_debug(ddc_debug),
 .debug(debug), .pcm_in(pcm_in), .config_reg(config_reg),
 .time_in(time_reg), .din(din) );

wire [63:0] vita_time;
timekeeper_with_subregs #( .BASE(SR_TIME) ) timekeeper
(.clk(clk), .reset(reset), .pps(1'b0),
 .set_stb(set_stb), .set_addr(set_addr), .set_data(set_data),
 .vita_time(vita_time), .vita_time_lastpps() ,

47
always @(posedge clk) begin
  if(reset) begin
    time_reg <= 64'h000A000B000C000D;
  end else begin
    time_reg <= time_reg + 1;
  end
end

endmodule
Appendix B. Python GNU Radio E312 Initialization

This appendix appears in its original form, without editorial change.
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

#################################################
# GNU Radio Python Flow Graph
# Title: E312 Tm Rx
# Generated: Mon May  6 09:55:55 2019
#################################################

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import uhd
from gnuradio import zeromq
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from optparse import OptionParser
import telemetry
import time

class e312_tm_rx(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self, "E312 Tm Rx FPGA")

# Variables

self.server_address = server_address = "192.168.10.77"

# TM VARIABLES
self.samp_rate = samp_rate = 1e6 # RX rate
self.freq = freq = 20000e5 # center freq
self.freq = freq = 22555e5 # center freq
self.crate = crate = 32e6 # master clk rate
self.gain = gain = 50 # gain

self.samp_per_bit = samp_per_bit = 8
self.nwords = nwords = 48
self.nbits = nbits = 16
self.dummy_sfid = dummy_sfid = 255 # upper byte of 3rd word
self.external_pcm_en = external_pcm_en = 0
self.sim_pcm_en = sim_pcm_en = 0
self.no_filt = no_filt = 0 # assign
use_filt_10 = config_reg2 [16];

self.randomized = randomized = 1
self.sync0 = sync0 = 65131
self.sync1 = sync1 = 10304 # 10304
self.scale_rx = scale_rx = 1
self.scale_rx2 = scale_rx2 = 1
self.sync_size = sync_size = 3
self.swap_bytes = swap_bytes = 0
self.gps_en = gps_en = 0
self.crc = crc = 1
self.decrypt = decrypt = 0

self.key = key =
"000100020003000400050006000700080009000A000B000C000D000E000F0010"
self.nounce = nounce = "001100120013001400150016"

###########################################################################
# Blocks
###########################################################################
self.zeromq_push_sink_0_0_0 =
zeromq.push_sink(gr.sizeof_short, 2, 'tcp://*:9999', 50, False, -1)

self.uhd_usrp_source_0 = uhd.usrp_source(
  ",".join(('fpga=/home/root/custom_fpga_images/e300_dum
my_reg.bit', "")),
    uhd.stream_args(
        cpu_format="sc16",
        otw_format='sc16',
        channels=range(1),
    ),
)

#telemetry soft agc
#self.telemetry_frame_sync_0 =
telemetry.frame_sync(65131, 25, 0.5, 0.009, 0.0009, gain, 70)

#self.telemetry_frame_sync_0 =
telemetry.frame_sync(65131, 50, 0.6, 0.018, 0.0036, gain, 70)

#self.telemetry_frame_sync_0 =
telemetry.frame_sync(65131, 50, 0.6, 0.018, 0, gain, 75)

self.telemetry_frame_sync_0 =
telemetry.frame_sync(65131, 100, 0.6, 0.0, 0.0072, gain, 77)

self.telemetry_frame_sync_0 =
telemetry.frame_sync(65131, 100, 0.6, 0.036, 0.0072, gain, 77)

self.blocks_vector_to_stream_0 =
blocks.vector_to_stream(gr.sizeof_short*1, 2)

#self.uhd_usrp_source_0.set_subdev_spec('A:B', 0)
#self.uhd_usrp_source_0.set_antenna('TX/RX', 0)
self.uhd_usrp_source_0.set_clock_rate(crate, 0)
self.uhd_usrp_source_0.set_samp_rate(samp_rate)
```python
#self.uhd_usrp_source_0.set_rx_agc(True,'/mboards/0/dboards
/A/rx_frontends/A/gain/agc/mode/value','slow',0);
    self.uhd_usrp_source_0.set_gain(gain, 0)
self.uhd_usrp_source_0.set_center_freq(freq, 0)
#self.uhd_usrp_source_0.set_bandwidth(4e6, 0)
#print (self.uhd_usrp_source_0.get_bandwidth(0))

#self.uhd_usrp_source_0.set_subdev_spec('A:A', 0)
#set the AGC to fast

#self.uhd_usrp_source_0.set_rx_agc(True,'/mboards/0/dboards
/A/rx_frontends/A/gain/agc/mode/value','slow',0);
#self.uhd_usrp_source_0.set_antenna('TX/RX', 0)

# CUSTOM REGISTER HACK
print('Setting custom registers...')
reg_hack = 19088736

self.hex_key = key_hex = [0] * 22
print('Filling AES key...')
for x in range(16):
    tmp = key[x*4:((x+1)*4)]
    key_hex[x] = int(tmp,16)

for x in range(0,6):
    tmp = nounce[x*4:((x+1)*4)]
    key_hex[x+16] = int(tmp,16)
print key_hex

# load AES-256
for x in range(22):
```
self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(reg_hack/crate))

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(((x+4)*pow(2.0,22.0)+key_hex[x]*pow(2.0,4.0))/crate))

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(reg_hack/crate))

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t((0*pow(2.0,22.0)+sync0*pow(2.0,4.0))/crate)) # sync0

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t((1*pow(2.0,22.0)+sync1*pow(2.0,4.0))/crate)) # sync1

config_reg = (samp_per_bit/2)+(nwords-1)*pow(2.0,4.0)+external_pcm_en*pow(2.0,13.0)+sim_pcm_en*pow(2.0,14.0)+randomized*pow(2.0,15.0)+no_filt*pow(2.0,16.0);

print("config2 = %d" % (config_reg))

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t((2*pow(2.0,22.0) + config_reg*pow(2.0,4.0))/crate))

config_reg = scale_rx + sync_size*pow(2.0,3.0) + swap_bytes*pow(2.0,5.0) + scale_rx2*pow(2.0,6.0) + crc*pow(2.0,9.0) + decrypt*pow(2.0,10.0);

print("config3 = %d" % (config_reg))
self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t((3*\(2.0, 22.0\) + config_reg*\(2.0, 4.0\))/crate))

# set nbits last so that FPGA is reset

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(reg_hack/crate))
config_reg = nbits + dummy_sfid*\(2.0, 5.0\);
print("config4 = %d " % (config_reg))

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t((26*\(2.0, 22.0\) + config_reg*\(2.0, 4.0\))/crate))

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(19088752/crate))  # not sure what this is, but it's in original.

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(0/crate))
print('Done!')

# actually set time now

self.uhd_usrp_source_0.set_time_now(uhd.time_spec_t(0))
time_now = self.uhd_usrp_source_0.get_time_now()
print "Actual FPGA time is: ", time_now.get_real_secs()," secs"

###########################
# Connections
###########################

self.msg_connect((self.telemetry_frame_sync_0, 'command'), (self.uhd_usrp_source_0, 'command'))
self.connect((self.blocks_vector_to_stream_0, 0), (self.telemetry_frame_sync_0, 0))
self.connect((self.uhd_usrp_source_0, 0),
  (self.blocks_vector_to_stream_0, 0))

self.connect((self.uhd_usrp_source_0, 0),
  (self.zeromq_push_sink_0_0_0, 0))

def main(top_block_cls=e312_tm_rx, options=None):

    tb = top_block_cls()
    tb.start()
    tb.wait()

    if __name__ == '__main__':
        main()
1) Pelican 1430 Protector Top Loader Case with Foam - Yellow- www.markertek.com/ P/N BPL1430Y (8/5/2019)
3) Ubiquiti Networks EdgeSwitch 8-Port 150-Watt Managed PoE+ Gigabit Switch with SFP www.bhphotovideo.com/ P/N ES-8-150W (8/5/2019)
6) Knob ROTARY BLACK 21MM HIGH - www.digikey.com/ P/N360-2366-ND (8/5/2019)
8) Bezel Rectangular 46.38mm x 32.51mm Outside Dim- www.digikey.com/ P/N 811-1128-ND (8/5/2019)
10) LiFePO4 26650 Battery: 12.8V 6.6Ah (84Wh, 16A rate) - UN38.3 Passed -www.batteryspace.com/ P/N LFP-4S2P-14A-V1 (8/5/2019)
11) Smart Charger (3.0 A) for 12.8V (4 cells) LiFePO4 Battery Pack, 110-240VAC - CE / FCC -* www.batteryspace.com/ P/N CU-JAS213 (8/5/2019)
12) 65W 5A Solar Charge Controller with MPPT for Lithium Batteries www.batteryspace.com/ P/N GV-5-LI142 (8/5/2019)
14) Fiber Optic Plug Connector LC Duplex 125µm Beige - www.digikey.com/ P/N A122063-ND (8/5/2019)

17) Connector Protective Cap For LC ODVA Compliant Connectors- www.digikey.com/ P/N 1918177-1-ND (8/5/2019)

18) Connector Cap (Cover) For RJ45 Plug, Circular Bayonet Coupling- www.digikey.com/ P/N A31780-ND (8/5/2019)

19) Connector Neutrik power CON TRUE1 Chassis - www.markertek.com/ P/N NAC3MPX (8/5/2019)

20) Plug Modular Connector 8p8c (RJ45, Ethernet) Position Shielded Cat5e IDC-- www.digikey.com/ P/N A107361-ND (8/5/2019)


22) Instrumentation Handle Thermoplastic Screw Holes, Front- www.digikey.com/ P/N 1722-1234-ND (8/5/2019)

23) Sealing Cap Assemble with metal bead chain-Newark- P/N 208800-1 (8/5/2019)

24) Circular Connector, CPC Series 1, Panel Mount Receptacle, 4 Contacts, Nylon (Polyamide) Body-Newark- P/N 23C9744 (8/5/2019)


1) **RG 405** .086” RF coaxial semi rigid coax /www.fairviewmicrowave.com/PN# FM-SR086ALTN-STR

2) **Surface mounted RF coaxial connector**/ www.newark.com/PN# TBCF81

3) **SMA male to SMA male right angle connector**/ www.pasternack.com/PN# PE3822

4) **HD BNC to HD BNC RG58 connector** /www.mouser.com/PN# 034-1110-12G

5) **Plug modular connector 8p8c (RJ45, Ethernet) position shielded Cat5e IDC** / www.digikey.com/ P/N A107361-ND

6) **PE15A63007 LNA** / www.everythingrf.com/products/microwave-rf-amplifiers/pasternack-enterprises-inc/567-20-pe15a63007

7) **ZFSWA2-63DR+ SWITCH & DRIVER** / www.minicircuits.com/WebStore/dashboard.html?model=ZFSWA2-63DR%2B


9) **USRP E312 (Battery Operated, 2X2 MIMO, 70 MHZ - 6 GHZ) SDR** / www.ettus.com/all-products/usrp-e312/
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-D</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>AGC</td>
<td>automatic gain control</td>
</tr>
<tr>
<td>APG</td>
<td>Aberdeen Proving Ground</td>
</tr>
<tr>
<td>CAD</td>
<td>computer-aided design</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>DC</td>
<td>direct current</td>
</tr>
<tr>
<td>DDC</td>
<td>digital down converter</td>
</tr>
<tr>
<td>DIN</td>
<td>digital inputs</td>
</tr>
<tr>
<td>FCNT</td>
<td>frame counter</td>
</tr>
<tr>
<td>FEC</td>
<td>forward error correcting codes</td>
</tr>
<tr>
<td>FER</td>
<td>frame error rate</td>
</tr>
<tr>
<td>FIFO</td>
<td>first in, first out</td>
</tr>
<tr>
<td>FIR</td>
<td>finite impulse response</td>
</tr>
<tr>
<td>FM</td>
<td>frequency modulation</td>
</tr>
<tr>
<td>FPGA</td>
<td>field-programmable gate array</td>
</tr>
<tr>
<td>FSK</td>
<td>frequency-shift keying</td>
</tr>
<tr>
<td>GPS</td>
<td>global positioning system</td>
</tr>
<tr>
<td>GTB</td>
<td>Guidance Technology Branch</td>
</tr>
<tr>
<td>GUI</td>
<td>graphical user interface</td>
</tr>
<tr>
<td>HMA</td>
<td>highly maneuverable airframe</td>
</tr>
<tr>
<td>IC</td>
<td>integrated circuit</td>
</tr>
<tr>
<td>ID</td>
<td>identification</td>
</tr>
<tr>
<td>IDC</td>
<td>Insulation Displacement Connector</td>
</tr>
<tr>
<td>LNA</td>
<td>low-noise amplifier</td>
</tr>
<tr>
<td>LPF</td>
<td>low-pass filter</td>
</tr>
</tbody>
</table>
MIMO multiple-input multiple-output
PC personal computer
PCM pulse-code modulated
PID proportional-integral-derivative
RF radio frequency
RSSI received signal strength indicator
RST reset
RX receive
SDR software-defined radio
SFID subframe ID
SMA SubMiniature version A
SNR signal-to-noise ratio
SoC system-on-chip
TCP transmission control protocol
TM telemetry
TX transmit
UDP user datagram protocol
UHD USRP hardware driver
USRP Universal Software Radio Peripheral