ARL-TN-0987 ● Jan 2020

DEVCOM ARMY RESEARCH LABORATORY

Wired Signal Time-Stamping with a Software-Defined Radio Telemetry Receiver

by Michael L Don, Thomas G Brown, and Edward F Bukowski

ARL-TN-0987 ● JAN 2020

Wired Signal Time-Stamping with a Software-Defined Radio Telemetry Receiver

Michael L Don, Thomas G Brown, and Edward F Bukowski
Weapons and Materials Research Directorate, CCDC Army Research Laboratory

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* January 2020	**2. REPORT TYPE** Technical Note	**3. DATES COVERED (From - To)** May–July 2019

4. TITLE AND SUBTITLE Wired Signal Time-Stamping with a Software-Defined Radio Telemetry Receiver	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Michael L Don, Thomas G Brown, and Edward F Bukowski	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
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-TN-0987
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): Michael Don, 0000-0002-8021-9066

14. ABSTRACT
Over the last several years the Army Research Laboratory (ARL) has developed a software-defined radio (SDR) telemetry receiver. Research has been published on the receiver design, real-time decryption, and layered protocol. This technical note presents an additional capability, wired signal time-stamping. During field experiments, there are multiple components or external events that must be related to the telemetry data. By time-stamping triggers and indicators, all of the available data sources can be synchronized. After detailing the time-stamping design, results of a successful field experiment are presented. With the addition of wired signal time-stamping, ARL's SDR telemetry receiver has gained another valuable capability that will allow convenient and inexpensive testing independent of military range instrumentation.

15. SUBJECT TERMS
software-defined radio, telemetry, time-stamping, range instrumentation, flight testing

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Michael L Don
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	83	**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. Software-Defined Radio (SDR) Telemetry Receiver Overview **1**

3. Software Modifications **2**

3.1 Wired Input Ports 2

3.2 Original Accumulator Design 3

3.3 Wired Input Time-Stamping 5

3.4 Continuous Telemetry Output 6

3.5 Design Simulation 9

3.6 High-Resolution Time-Stamping 11

3.7 Sub-Cycle Resolution Time-Stamping 15

3.8 Telemetry GUI Setup for Wired Inputs 15

4. Hardware Modifications **16**

4.1 Electrical 17

4.2 Mechanical 18

5. Experimental Results **20**

6. Conclusion **21**

7. References **22**

Appendix A. Software-Defined Radio Field-Programmable Gate Array Verilog Code **23**

Appendix B. MATLAB Data Analysis Scripts **58**

Appendix C. Mechanical and Electrical Design **65**

List of Symbols, Abbreviations, and Acronyms **73**

Distribution List **75**

List of Figures

Fig. 1 Telemetry receiver B200 block diagram 2
Fig. 2 Original `decom_acc` state diagram 5
Fig. 3 `decom_acc1` state diagram, modified to accommodate wired inputs 6
Fig. 4 `decom_acc2` block diagram 7
Fig. 5 `decom_acc2` state diagram, modified for continuous output 8
Fig. 6 Simulation of a dummy frame in `decom_acc2` 10
Fig. 7 Simulation of a dummy frame in `decom_acc2`, close-up of extra words 11
Fig. 8 Simulation of a real frame in `decom_acc2` 11
Fig. 9 `decom_acc3` block diagram 13
Fig. 10 Simulation of `decom_acc3` 13
Fig. 11 Example test data showing the first wired input state using both low-resolution time-stamps at the frame rate and high-resolution time-stamps at the master clock rate. The time axis is normalized to zero 14
Fig. 12 A close-up of Fig. 11. In this example, the high-resolution data captured bouncing that the low-resolution data completely missed 14
Fig. 13 Tapped delay line 15
Fig. 14 Signal setup in telemetry GUI 16
Fig. 15 Example scaled wired inputs 16
Fig. 16 The Susumu PAT series attenuator circuit (left) and the pi attenuation circuit (right) 17
Fig. 17 Pi attenuation circuit 18
Fig. 18 USRP B200—unmodified 19
Fig. 19 USRP B200—modified 19

List of Tables

Table 1 Digital inputs enabled on the B200 2
Table 2 Extra telemetry words 3
Table 3 Dummy frames 9
Table 4 Simulation parameters 9
Table 5 Telemetry rate parameters 12
Table 6 Telemetry parameters used to record the signals shown in Fig. 11 14
Table 7 Example time stamped data recorded in a field exercise 20

1. Introduction

Over the last several years the Army Research Laboratory (ARL) has developed a software-defined radio (SDR) telemetry receiver. Research has been published on the receiver design,[1] real-time decryption, and layered protocol.[2] This technical note presents an additional capability, wired signal time-stamping. During field experiments, there are multiple components or external events that must be related to the telemetry data. By time-stamping triggers and indicators, all of the available data sources can be synchronized.

This technical note first reviews the basic telemetry receiver design. Next, the software and hardware modifications to support wired signal time-stamping are described. Finally, experimental results from an example flight experiment are presented. With the addition of wired signal time-stamping, ARL's SDR telemetry receiver has gained another valuable capability that will enable convenient and inexpensive testing independent of military range instrumentation.

2. Software-Defined Radio (SDR) Telemetry Receiver Overview

ARL's SDR telemetry receiver is based on Ettus Research's Universal Software Radio Peripheral (USRP) B200 SDR. This is a single board SDR, using Analog Devices' RF IC that combines an RF front end, in-phase/quadrature (I/Q) demodulator, and analog-to-digital converters (ADCs) into a single IC that covers a range of 70 MHz to 6 GHz.[3] There is an optional GPS disciplined oscillator (GPSDO) that can be installed on the B200 to enable global timing alignment to within 50 ns.[4] Figure 1 shows the receiver architecture. Demodulation, bit synchronization, and frame synchronization modules were developed in Verilog and added to field-programmable gate array (FPGA) firmware. The decimating filters, which are normally required to reduce the data rate to speeds slow enough for the host computer to process, were able to be replaced by non-decimating filters due to the enhanced processing capabilities of the FPGA. A LabVIEW telemetry display program was designed for the host computer. 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.

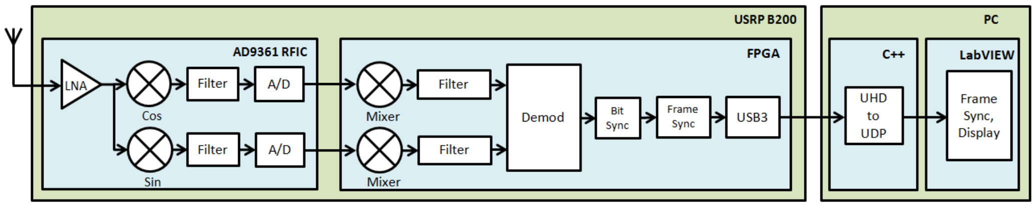


Fig. 1 Telemetry receiver B200 block diagram

3. Software Modifications

3.1 Wired Input Ports

In order to support wired input time-stamping, additional FPGA inputs were enabled on the B200. The USRP B2x0 series SDRs come in two varieties, the B210 and the B200. The relevant difference between these versions is that the B210 has a larger Xilinx Spartan 6 XC6SLX150 FPGA, while the B200 has a smaller XC6SLX75 FPGA. This has two repercussions. First, Xilinx's Integrated Synthesis Environment (ISE) free software edition does not support the larger XC6SLX150 FPGA. This, together with the difficulty of ordering software at ARL, was one of the reasons that the B200 was chosen as the platform for the telemetry receiver. Second, the B210 supports more input/output (IO) ports than the B200. Thus, even though the B200 and B210 have the same printed circuit board (PCB) layout, the 38-pin debug connector of the B210 is not officially supported on the B200. Some of the pins, however, do connect to valid IO pins on the B200's FPGA. Table 1 specifies eight IOs that were identified on the debug connector that could be enabled on the B200.[5]

Table 1 Digital inputs enabled on the B200

MICTOR pin	FPGA pin	B210 net name	B200 net name
5	A12	debug_clk(0)	din(0)
6	C12	debug_clk(1)	din(1)
23	F15	debug(23)	din(2)
24	D7	debug(7)	din(3)
31	C17	debug(19)	din(4)
32	B8	debug(3)	din(5)
37	A18	debug(16)	din(6)
38	D10	debug(0)	din(7)

These ports were added to the user constraint file (UCF) as 3.3-V inputs with pull-up resistors. The pull-up resistors support open-drain signals that can only pull the port voltage low. When the signal is active, the port is pulled low, when the signal is inactive, the pull-up resistor acts to pull the port voltage high. In most circumstances these resistors should not interfere with full push–pull type inputs, but they can be disabled or switched to pull-down resistors, by adjusting the UCF file and recompiling the FPGA image.

3.2 Original Accumulator Design

As mentioned previously, the FPGA attaches extra words to each frame before they are sent to the PC. Table 2 shows the original seven extra 16-bit telemetry words added to the end of every frame.

Table 2 Extra telemetry words

Index	Name	Shorthand	Description
1	RSSI [31:16]	RSSI2	RSSI word 1
2	RSSI [15:0]	RSSI1	RSSI word 0
3	TIME [63:48]	TIME4	Timestamp word 3
4	TIME [47:32]	TIME3	Timestamp word 2
5	TIME [31:16]	TIME2	Timestamp word 1
6	TIME [15:0]	TIME1	Timestamp word 0
7	AVE	AVE	Average value of demodulated data

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 extra words to the end of each frame. 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 , along with Din 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_acc` 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. In order to handle this problem, the state machine keeps track of the proper section of the output register to load, either D1 or D2.

Starting in state RST, the state machine automatically transitions to the LD1 state. When the input strobe `ld_in` is asserted, D2 is loaded with Din, and the state machine transitions to the WAIT1 state. A counter delays the state machine in WAIT1 for `clk_div`+1 clock cycles before transitioning to LD2, which is a sufficient period of time for `ld_in` to be deasserted. `clk_div` is set to the number of clock cycles per PCM bit. When `ld_in` is asserted again, D1 is set to Din 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 TIME2, `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 TIME22, 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.

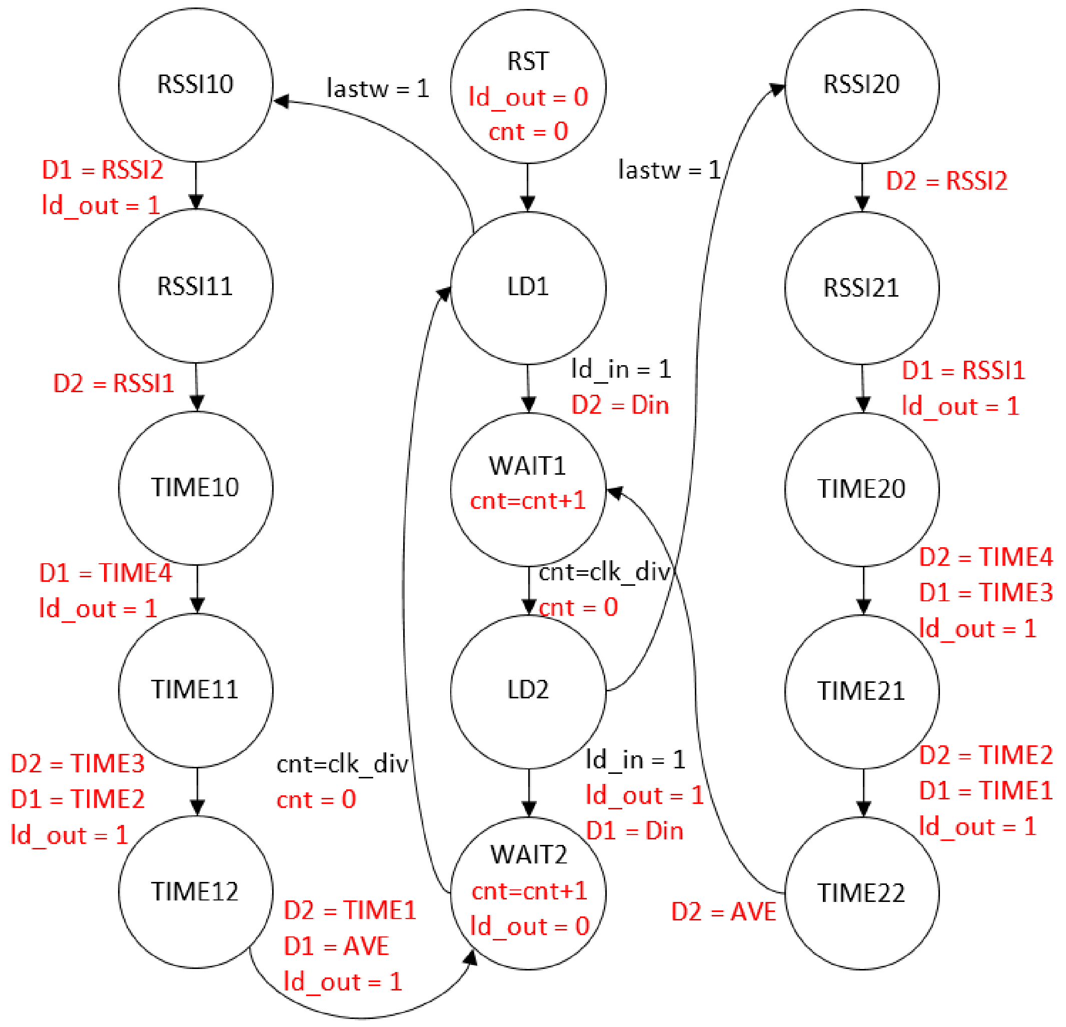


Fig. 2 **Original `decom_acc` state diagram**

3.3 Wired Input Time-Stamping

In order to support wired inputs, an eighth extra 16-bit word, DIN, was added at the end of every frame. The eight wired inputs are represented by the lower byte, while the upper byte is unused. The DIN signal was routed from the top level of the design to the accumulator module. The original `decom_acc` code was modified and renamed `decom_acc1` (see Appendix A). Figure 3 shows the modified state diagram. Now that there is an additional extra word, state TIME22 can fill the whole data_out register with D2 = AVE and D1 = DIN. TIME22 transitions to WAIT2, which will lead to LD1 where the beginning of the next frame will be loaded into D2. An additional state, TIME13 is added after TIME12, where D2 is loaded with DIN. TIME13 transitions to WAIT1, where the next word will be loaded into D1.

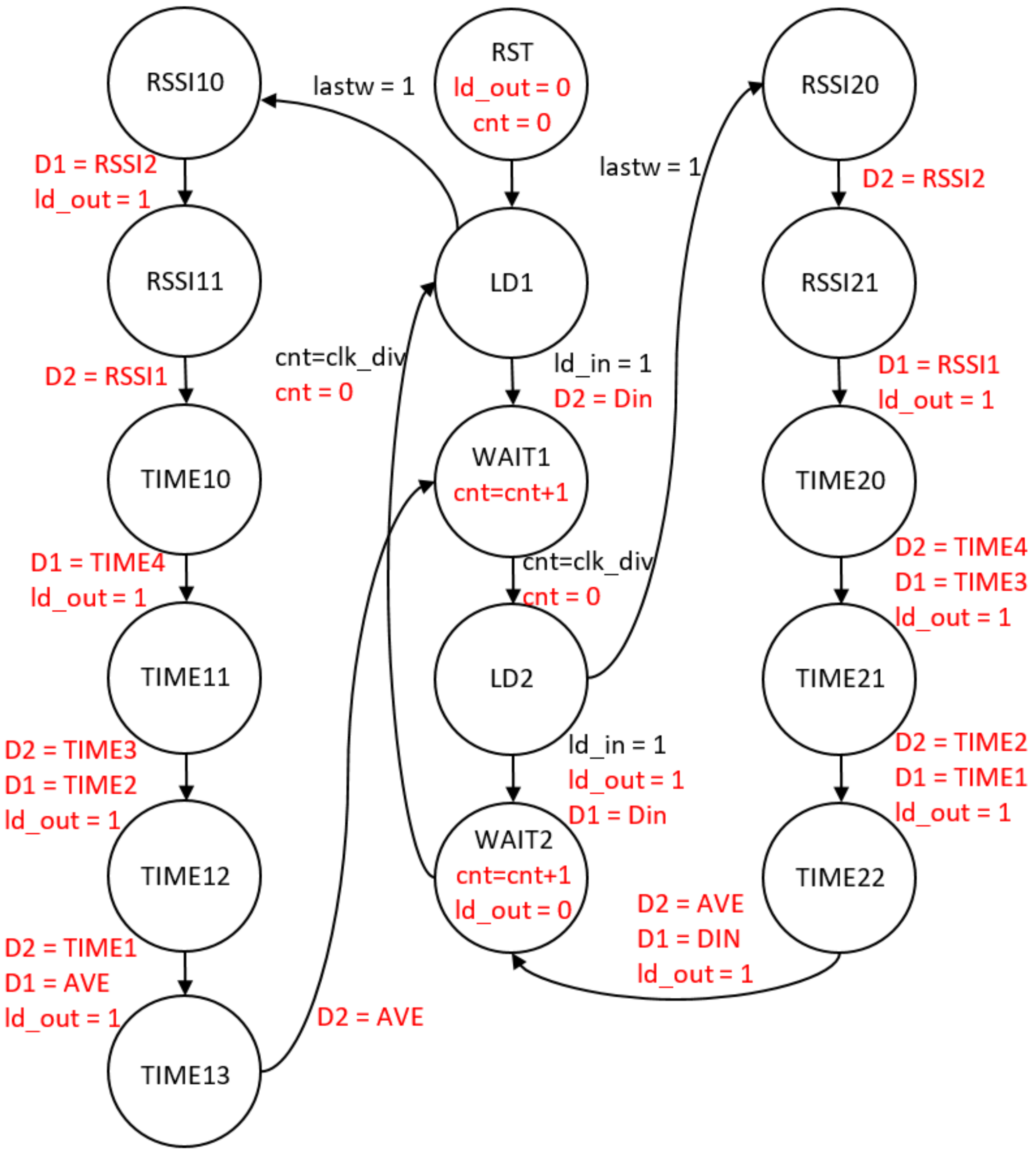


Fig. 3 `decom_acc1` **state diagram, modified to accommodate wired inputs**

3.4 Continuous Telemetry Output

In its normal operating mode, the receiver is designed to output telemetry frames as they are received. When frames are not being detected, no data is output. This makes time-stamping wired signals unreliable while receiving telemetry data. Any dropped telemetry frames will also result in a loss of DIN data. In order to be used reliably, the receiver must be used in a simulator mode, where the SDR continually outputs simulated telemetry frames, irrespective of the received RF signal. It would be desirable, however, to receive a real RF telemetry stream while also time-stamping wired input signals. In order to accomplish this, the accumulator was further modified to output data even when telemetry frames were not detected in its normal operation mode. Figure 4 shows a block diagram of this version of the 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 telemetry words. When there is a full frame of words available in the FIFO, they are unloaded and sent to the PC. 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.

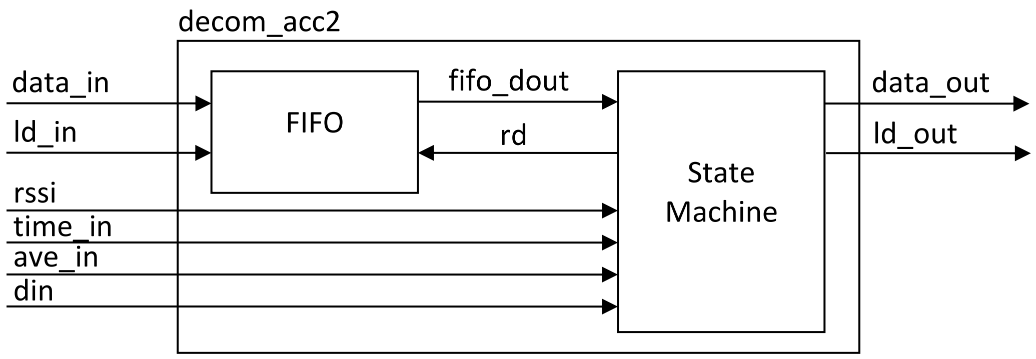


Fig. 4 `decom_acc2` block diagram

Figure 5 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 respectively. `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 telemetry 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, otherwise `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 telemetry data rate. This ensures that the FIFO does not overflow when the transmitter data rate is slightly higher than the receiver's expected data rate due to a mismatch between transmitter and receiver clocks. Thus, even when telemetry data is consistently received, a dummy frame will be output occasionally. 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 3. 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.

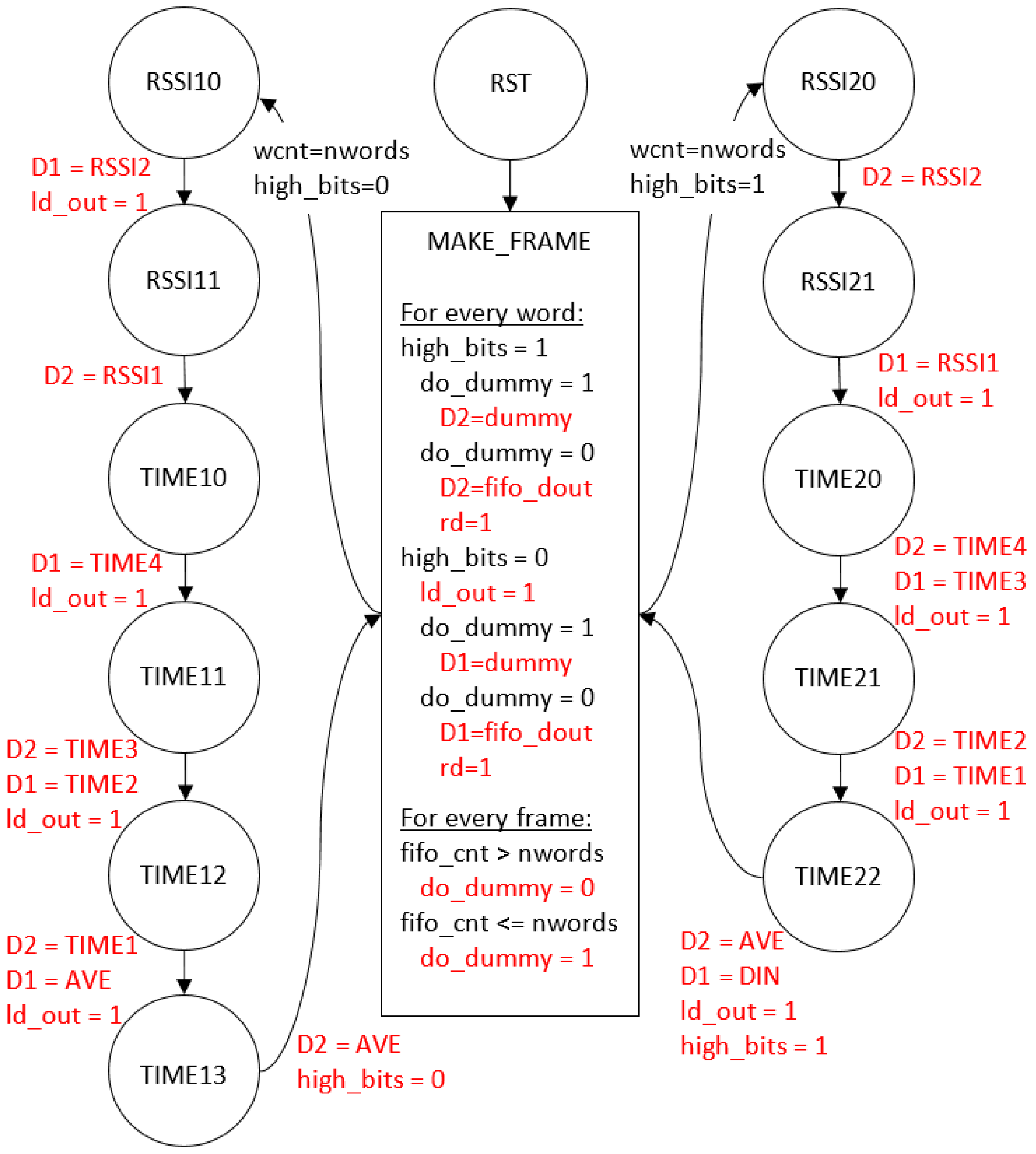


Fig. 5 `decom_acc2` state diagram, modified for continuous output

Table 3 Dummy frames

Word index	Words
0	SYNC [31:16]
1	SYNC [15:0]
2	{Dummy_SFID , x00}
3 . . . NWORDS-3	Word index
NWORDS-2	FCNT
NWORDS-1	1

3.5 Design Simulation

Due to the long compile times of FPGA images, simulation is a key part of FPGA design. A test bench for SDR receiver modifications, `dcc_chain_tb_din`, is included in Appendix A. The test bench simulates at the digital down converter (DDC) level, which contains the accumulator. A full explanation of the DDC is outside of the scope of this report, although some aspect of the higher-level design will be explained briefly. Parameters of the B200 are stored in setting registers in the FPGA and are set using the USRP Hardware Driver (UHD). Typically, adding additional setting registers would require modification and recompilation of UHD. In order 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 4, and are loaded into custom setting registers at the beginning of the test bench using the method just described. By setting `sim_pcm_en`, the receiver generates simulated telemetry frames for transmission to the PC. For our purposes, these simulated frames can be used to take the place of frames received from the demodulator. Although similar, these simulated frames should not be confused with the dummy frames generated in the accumulator.

Table 4 Simulation parameters

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

Figure 6 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 an output strobe. 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 `data_out` are received first, and the lower bits are received last. Thus, a data_out value of x6BFE4028 is received at the PC as xFE6B2840.

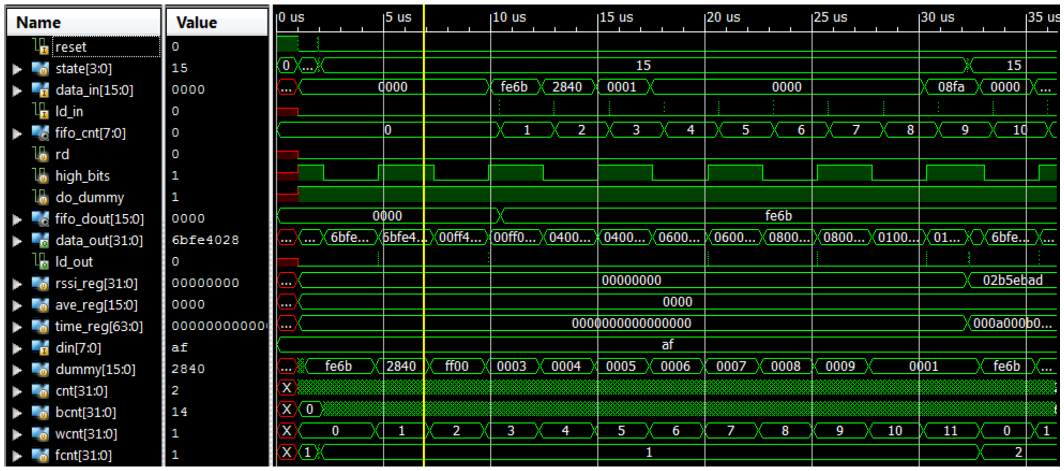


Fig. 6 Simulation of a dummy frame in `decom_acc2`

Figure 7 shows a close-up of the extra words at the end of the frame. Transitioning from MAKE_FRAME with high_bits = 1, the state machine enters RSSI20 (state = 7). The RSSI, TIME, AVE, and DIN registers are inserted into `data_out`, and strobed out with `ld_out`. The state machine returns to the MAKE_FRAME state with `high_bits` active, ready to load the next word into the higher bits of `data_out`.

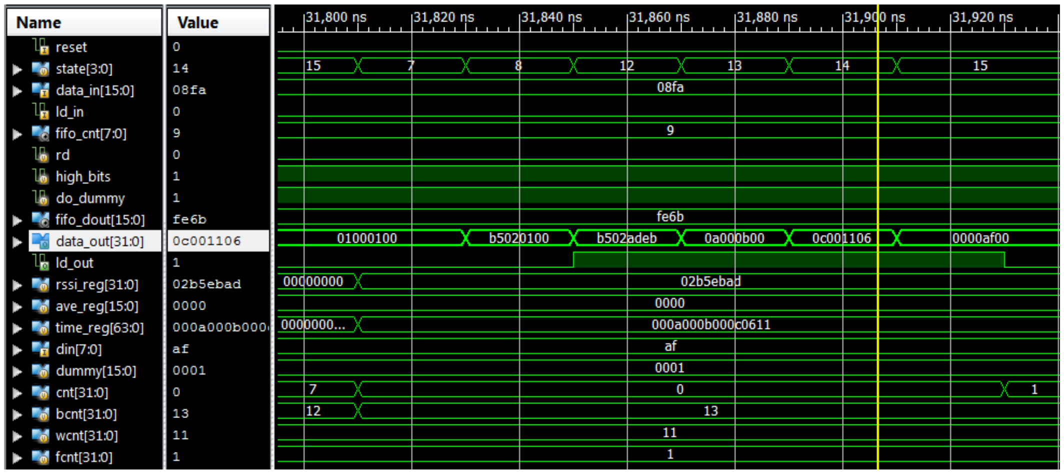


Fig. 7 **Simulation of a dummy frame in `decom_acc2`, close-up of extra words**

Figure 8 shows the simulation of a real frame in `decom_acc2`. `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`.

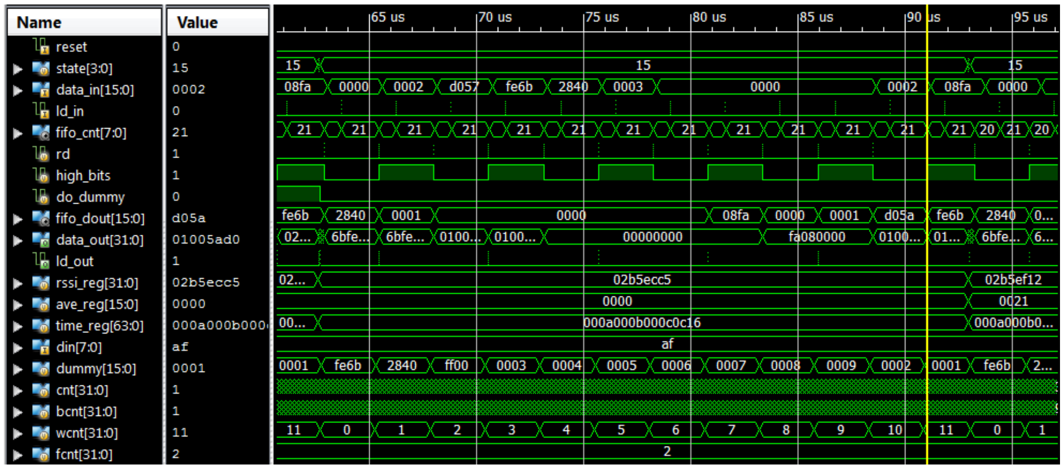


Fig. 8 **Simulation of a real frame in `decom_acc2`**

3.6 High-Resolution Time-Stamping

By attaching an extra DIN word to every frame, the maximum DIN sampling rate is the frame rate. When the SDR is used for both telemetry and to record wired inputs, the frame rate is determined by the telemetry requirements. For most applications, it is sufficient for the time resolution of the wire inputs to match the telemetry frame rate. In cases where a higher resolution is desired, the SDR can be used for the wired inputs alone, running in simulation mode at higher rates than the telemetry rate. For example, Table 5 shows typical and accelerated telemetry parameters that determine the frame rate. By adjusting the typical parameters, the frame rate, and therefore the DIN sampling rate, was increased by roughly a factor of 3. This accelerated frame rate is about as fast as the PC telemetry software can run on a PC with a 2.4 GHz i7-2760QM CPU and 8 GB of RAM. If higher time resolution is needed, another time-stamping method must be employed.

Table 5 Telemetry rate parameters

Parameter	Typical value	Accelerated value
NWORDS	48	24
Bits per word	16	16
Samples per bit	8	8
Sampling clock	32 MHz	50 MHz
Bit rate	4 Mbs	6.25 Mbs
Frame rate	5.2 kHz	16.3 kHz

One method to increase time-stamping resolution is to record the time at every DIN transition, save it to a FIFO, and empty the FIFO into extra telemetry words. This increases the time resolution to that of the system clock, but it would require the addition of multiple extra words to handle a fast transitioning signal. In order to retain the ability to record transitions at every frame, the previous time-stamping method in `decom_acc2` was left in place, and this new method was added utilizing the dummy frames for time-stamp output instead of extra words at the end of the telemetry frames. Figure 9 shows a block diagram of the new implementation, `decom_acc3`. A DIN state machine saves a time-stamp into a new DIN FIFO every time the DIN signals change value. The DIN FIFO input data width is 128 bits. The first three 16-bit words are synchronization words: x0123, x4567, and x89AB. The next 16-bit word combines the previous and new values of the DIN byte. The final four words contain the 64-bit time-stamp. The FIFO's output width is 16 bits. Whenever the DIN FIFO is not empty, data words in the dummy frame are replaced by words read out of the DIN FIFO. All words in the dummy frame are considered data words except for the first two synchronization words, the third SFID word, the second-to-last word used as a frame counter, and the last word used as a checksum placeholder.

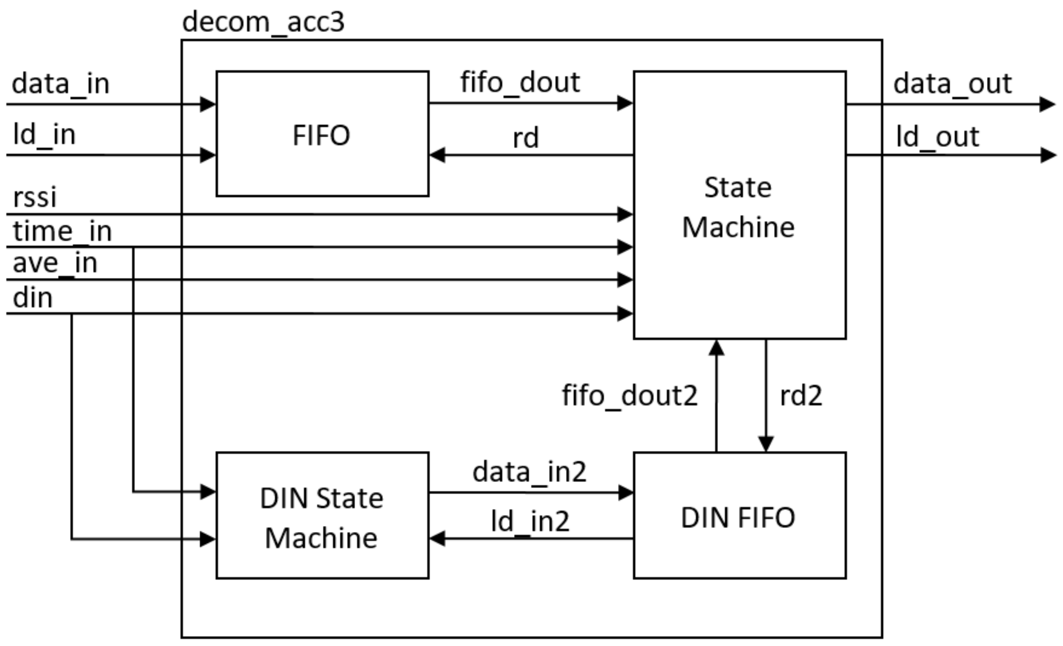
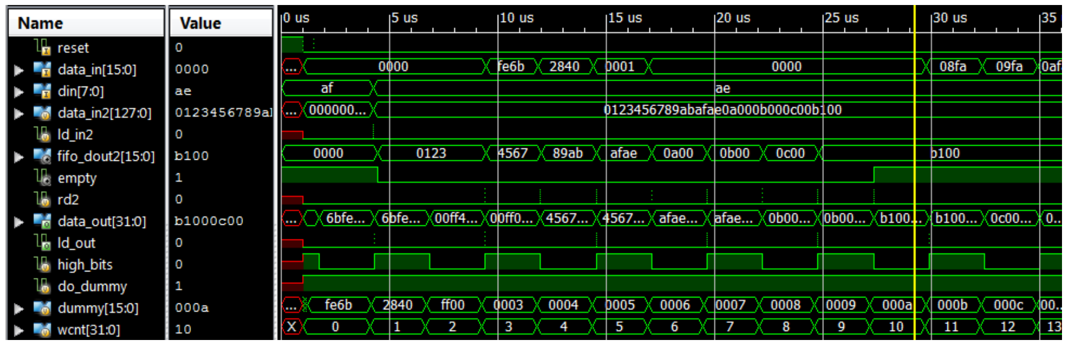


Fig. 9 **`decom_acc3` block diagram**

Figure 10 shows a simulation of `decom_acc3`. When the value of DIN changes, the time-stamp information is saved in the DIN FIFO, which deasserts the empty flag. The time-stamp information is then read, 16 bits at a time, during a dummy frame and strobed out on `data_out`. Once all of the words are read from the DIN FIFO, the empty flag is reasserted.



Name	Value
reset	0
data_in[15:0]	0000
din[7:0]	ae
data_in2[127:0]	0123456789al
ld_in2	0
fifo_dout2[15:0]	b100
empty	1
rd2	0
data_out[31:0]	b1000c00
ld_out	0
high_bits	0
do_dummy	1
dummy[15:0]	000a
wcnt[31:0]	10

Fig. 10 **Simulation of `decom_acc3`**

Example data was recorded comparing this new time-stamping method to the previous method of sampling DIN once per frame. The parameters used for this test are shown in Table 6. Note that the effective sampling rate of this new method is the rate of the sampling clock, 10,000 times faster than the frame rate. Figure 11 shows a comparison between the low-resolution framed-based time-stamping and the high-resolution master clock-based time-stamping for the first wired input. The signals were extracted from a saved telemetry file using the MATLAB scripts included in Appendix B. The signals align well, showing that the high-resolution data was decoded correctly. Figure 12 shows a close-up of the first transition in Fig. . The high-resolution signal captures signal bouncing that the low-resolution signal completely misses.

Table 6 Telemetry parameters used to record the signals shown in Fig.

Parameter	Typical value
NWORDS	48
Bits per word	16
Samples per bit	13
Sampling clock	52 MHz
Bit rate	4 Mbs
Frame rate	5.2 kHz

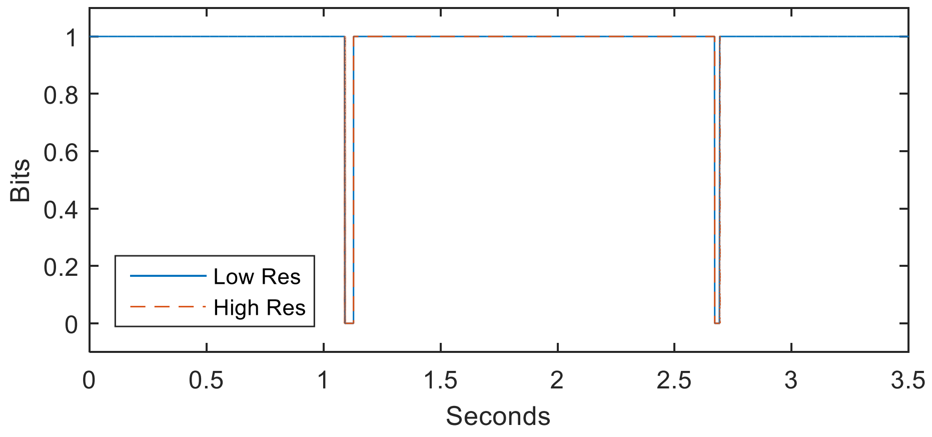


Fig. 11 Example test data showing the first wired input state using both low-resolution time-stamps at the frame rate and high-resolution time-stamps at the master clock rate. The time axis is normalized to zero.

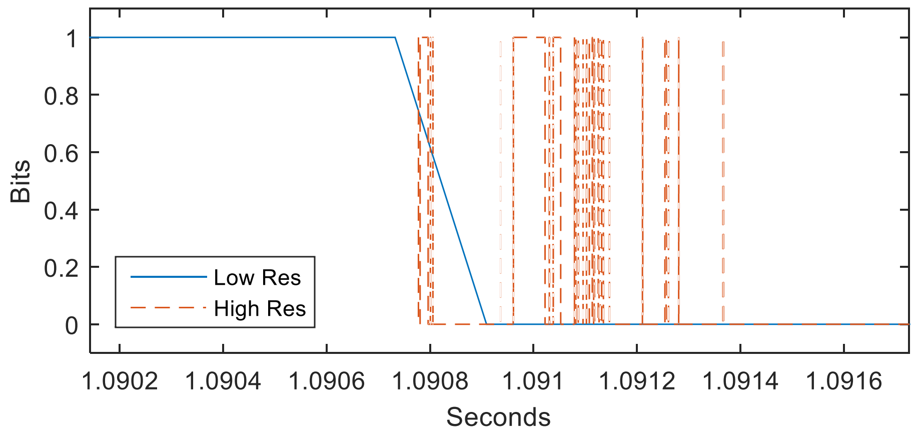


Fig. 12 A close-up of Fig. 11. In this example, the high-resolution data captured bouncing that the low-resolution data completely missed.

3.7 Sub-Cycle Resolution Time-Stamping

A method for achieving time resolution equal to the master clock rate was presented in the previous section. This method relies on the 64-bit TIME signal, which is implemented as a simple counter. This type of synchronous design is the standard digital logic design recommended for FPGA coding, simulation, and implementation tools. Asynchronous designs that rely on wire and logic delays frequently cause problems with these design tools, and often the FPGA designs themselves. Nevertheless, asynchronous structures can be used to obtain sub-cycle time interval measurements. One asynchronous method is the tapped delay line shown in Fig. 13. In this implementation, the input signal goes through a series of delays τ, which are tapped as clock inputs into a series of flip-flops. The system clock is connected to the data inputs, allowing the clock to be sampled at the rising edge of the input signal at a rate of $1/\tau$. A comparison of the clock phase of the various delayed inputs can then be used to determine a fine sub-cycle time interval measurement. Normal counters can still be used to create a coarse time measurement. Although sub-cycle time-stamping has not been implemented on the SDR telemetry receiver, ARL has demonstrated a proof of concept design that could be incorporated into the SDR in the event that higher time resolution is required.[6]

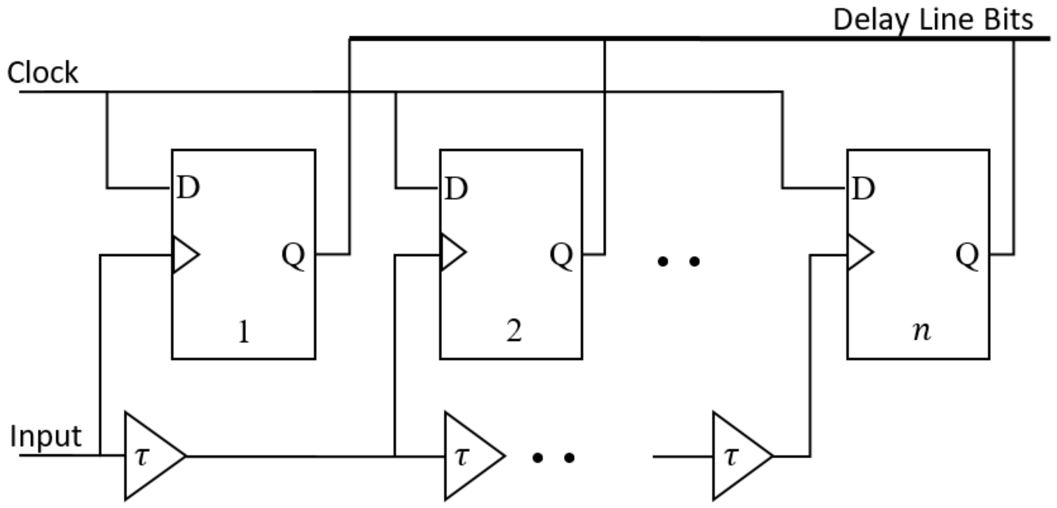
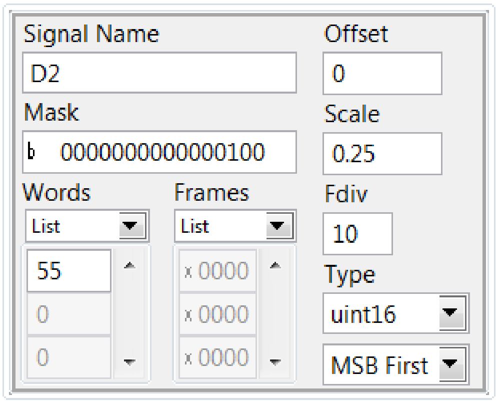


Fig. 13 Tapped delay line

3.8 Telemetry GUI Setup for Wired Inputs

It is often desirable to monitor wired input in real-time during experiments. The original signal setup parameters in the LabVIEW GUI shown in the block diagram in Fig. 1 is capable of displaying the wired inputs without any software modifications. Individual input channels can be monitored using the Mask and

Scale properties of the signal. For channel n, bit n of the Mask is set to 1, and the Scale is set to $1/2^n$. Figure 14 shows an example signal setting for a wired input D2, with $n = 2$ referenced from zero. The Mask extracts each individual channel, while the Scale normalizes the value of each channel to 1. The word number is also referenced from zero, giving the eighth extra word an index of 55 for a 48-word frame. Figure 15 shows two of the GUI's charts, where the top signals are properly normalized, while the bottom signals have not been normalized.



Signal Name: D2
Offset: 0
Mask: b 0000000000000100
Scale: 0.25
Words: List — 55, 0, 0
Frames: List — x 0000, x 0000, x 0000
Fdiv: 10
Type: uint16
MSB First

Fig. 14 Signal setup in telemetry GUI

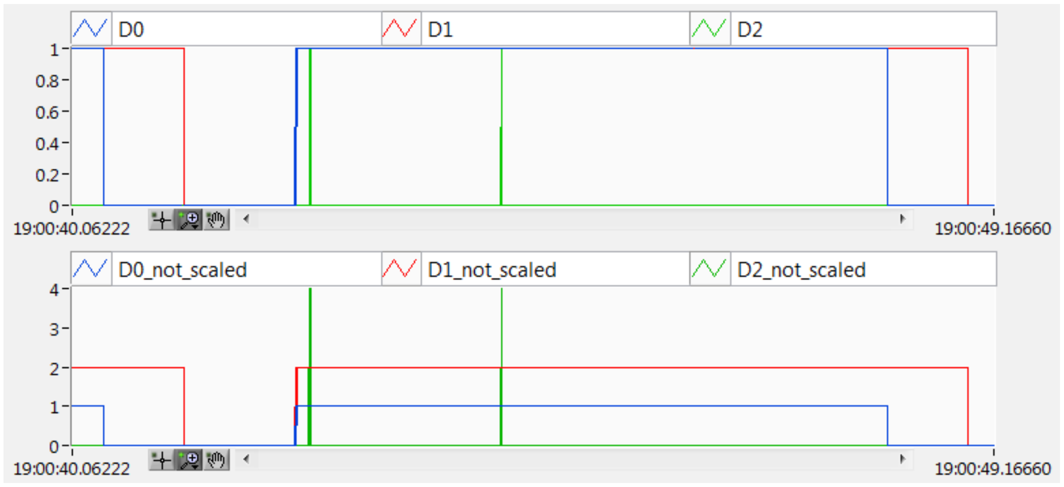


Fig. 15 Example scaled wired inputs

4. Hardware Modifications

ARL's SDR telemetry receiver utilizes Ettus Research's USRP B200 electronic and mechanical hardware. To accommodate the time synchronization of multiple signals (up to eight channels) with respect to GPS time, several modifications to the mechanical hardware and additional electrical peripheries were required.

4.1 Electrical

A custom printed circuit breakout-board (BoB) was designed to allow up to eight external signals to be input into the SDR. The SDR itself is a single, enclosed unit with a B200 USRP circuit board securely mounted inside. The BoB was required to fit inside the SDR enclosure and accommodate a broad range of DC voltage level. Ruggedness, ease-of-use, and versatility were also required.

The BoB was designed to interface with the B200 using a Tyco Electronics MICTOR high-speed, fine pitch vertical connector that mated with the stock debug connector on the B200 circuit board. External signals were input to the BoB using standard BNC connectors with 50Ω impedance. BNC connectors were chosen because they are the most commonly used connectors at experimental facilities and would provide the most practical means to connect signal cables to the SDR. The enclosure was modified so that the BNC connectors on the BoB could be mounted on the sides of the SDR securing the BoB to both the enclosure and the B200 circuit board. The ruggedness of the unit was also increased by using edge-mounted BNC connectors and a board thickness of 0.1 inches.

The B200 debug connector provides access to multiple IO pins on the Xilinx Spartan-6 FPGA inside the SDR. The maximum allowable DC voltage into these pins is 3 V. In order to limit the DC voltage of the external signals into the FPGA, an adjustable attenuation circuit was included on each signal line. The BoB was designed so that either a 0805 case size Susumu PAT series attenuator chip or a pi resistor attenuation circuit could be used to decrease the DC voltage level of any input signal to a maximum of 3.3 V, see Fig. 16.

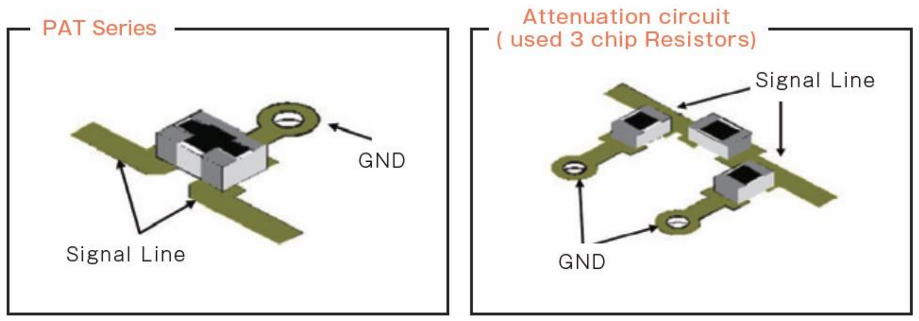


Fig. 16 The Susumu PAT series attenuator circuit (left) and the pi attenuation circuit (right)

The required gain in dB is given by

$$G = 20\log\left(\frac{V_{in}}{V_{out}}\right). \tag{1}$$

For example, in order to decrease 5 V to 3 V, a 4.43 dB attenuator is required.

High precision chip attenuators can typically be purchased in 1 dB steps. In this application, a Susumu PAT1220 high precision chip attenuator with 50Ω impedance and 4 dB attenuation was used to decrease the input voltage level from 5 VDC to approximately 3.15 VDC. Another option is to use a three-resistor pi attenuator circuit to obtain the required attenuation. Figure 17 shows the pi attenuator consisting of one series resistor and two parallel shunt resistors to ground at the input and the output.

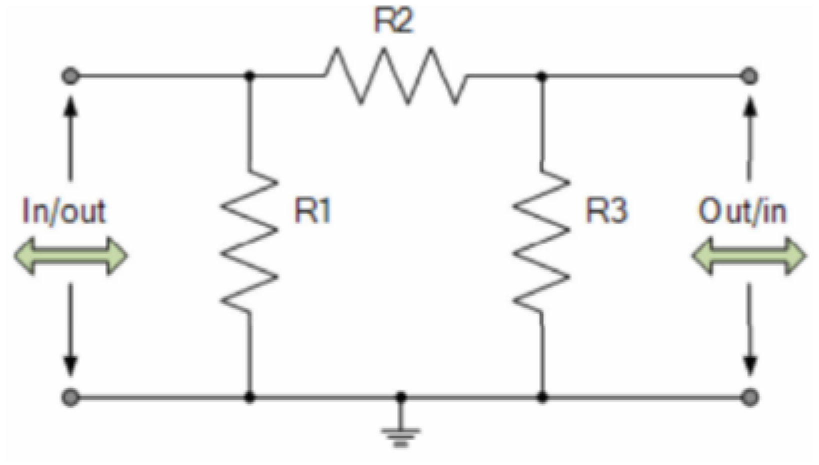


Fig. 17 Pi attenuation circuit

The resistor values are

$$R1 = R3 = \mathrm{Z}\left(\frac{K+1}{K-1}\right), \tag{2}$$

$$R2 = \mathrm{Z}\left(\frac{K^2-1}{2K}\right), \tag{3}$$

where $K = 10^{(G/20)}$ and Z is the impedance. For example, a 4-dB attenuator with 50Ω impedance would result in $R1 = R3 \approx 221\Omega$ and $R2 \approx 24\Omega$.

4.2 Mechanical

The mechanical design process began with solid modeling of the mechanical and electrical components needed to meet the rigorous time accuracy specifications. A depiction of the original USRP B200 box and SDR mechanical model is seen in Fig. 18. Most mechanical component models were created from scratch using measurements of existing hardware since no commercial models were uncovered.

This became a good starting point for required modification and the design of new electrical and mechanical interfaces.

Since up to eight channels could be evaluated using the existing USRP B200 design architecture, it was required to design a mechanical and electrical interface capable of routing proper signals to the SDR. BNC connectors are used as the input interface for the eight timing pulses. Two additional BNC connections act as PCM input and output, connected to the UART (J400) debug port. An additional PCB was designed to route electrical connections from the input BNC connectors to the SDR. This modified package including peripheral board and connectors is shown in Fig. 19. Details of the mechanical and electrical layout is included in Appendix C.

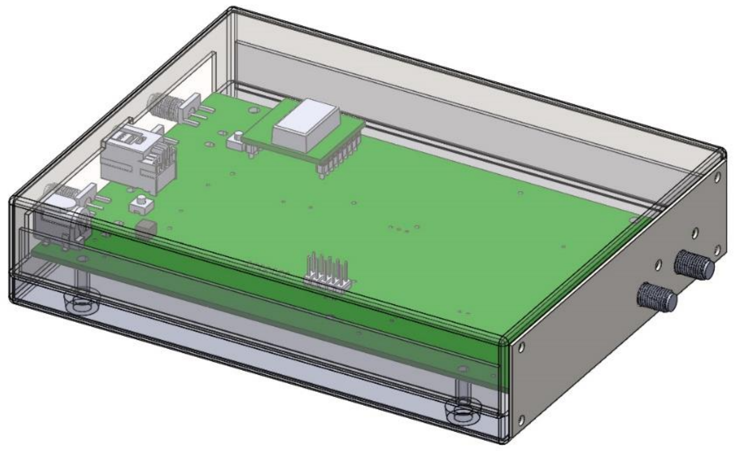


Fig. 18 USRP B200—unmodified

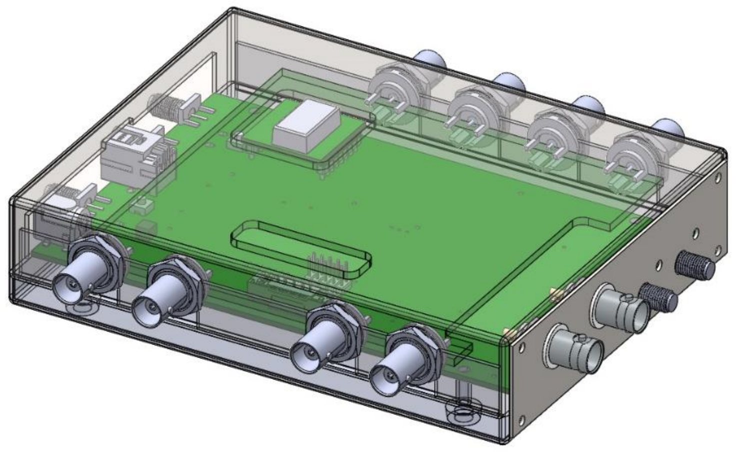


Fig. 19 USRP B200—modified

5. Experimental Results

Multiple events were recorded for a particular experiment conducted in July 2019 that required absolute time for sequencing and comparison. Although eight channels could be recorded, three channels were recorded and used for measurement and sequencing during the experiment using the SDR receiver. This experiment used the `decom_acc1` accumulator design, with the SDR configured to output simulated telemetry frames with the typical parameter values in Table 5. The data shown in Table 7 summarizes the experiment results and represents valuable information for later analysis of multiple sequenced events. The three IRIG times represent sequential times after a truck was released down a ramp. The first measurement indicates when the truck passed a position down the ramp by "breaking" continuity set to a known distance (10 ft) from the release height. The second measurement, Firing Pulse, represents a preprogrammed time that a relay switched a signal to send a firing pulse to the gun. The last measurement, Muzzle Exit, is the time that an IR threshold (flash) exceeded a set value to indicate the time the projectile exited the muzzle. Times are given in IRIG format (seconds from midnight, GMT). The local time was recorded and updated using the IRIG time at Truck Break.

Table 7 Example time stamped data recorded in a field exercise

			Time			
Rnd #	GTB #	Date	local	IRIG	IRIG	IRIG
				Truck Break	Firing Pulse	Muzzle Exit
1	BS3	7/19/2019	11:32:04	37924.47855	37925.47849	37925.47791
2	HMA1	7/19/2019	15:29:00	52140.43407	52141.43362	52141.45263
3	BS1	7/22/2019	12:12:46	40365.63909	40366.63884	40366.6613
4	HMA2	7/22/2019	14:51:35	49894.67532	49895.67487	49720.00063
5	BS2	7/24/2019	10:47:29	35249.48749	35250.48781	35196.05274
6	HMA5	7/24/2019	14:39:54	49193.88762	49194.88717	49194.91674
7	HMA4	7/24/2019	16:02:27	54147.24282	54148.24276	54148.27156
8	HMA3	7/24/2019	16:59:58	57598.24621	57599.24596	57599.27975

The average of the time from the Firing Pulse to the Muzzle Exit was approximately 0.027 s, as would be expected for a typical initiation and interior ballistic event. However, three measurements of Muzzle Exit indicated a premature exit condition and therefore a false trigger. No premature exit conditions existed according to other diagnostics including high-speed video. These three values of exit were not used in calculating the average exit time.

6. Conclusion

Leveraging recent developments in SDR telemetry, ARL has added the capability to time-stamp wired signals. A commercial SDR was modified to measure up to eight channels of time events. Its rugged design and compact size make it suitable for both laboratory and field applications. It has already been successfully used in a field experiment, and will provide convenient and inexpensive time synchronization for future experiments independent of military range instrumentation.

7. References

1. Don ML. A low-cost software-defined telemetry receiver. International Foundation for Telemetering; 2015.
2. Don M, Ilg M. Advances in a low-cost software-defined telemetry system. International Foundation for Telemetering; 2017.
3. Software defined radio-solutions from ADI [accessed 5 September 2019]. https://www.analog.com/media/en/news-marketing-collateral/solutions-bulletins-brochures/Software-Defined-Radio-Solutions-From-ADI.pdf.
4. B200-B210 specification sheet [accessed 5 September 2019]. https://www.ettus.com/wp-content/uploads/2019/01/b200-b210_spec_sheet.pdf.
5. Spartan-6 FPGA packaging and pinouts - product specification (UG385) [accessed 5 September 2019]. https://www.xilinx.com/support/documentation/user_guides/ug385.pdf.
6. Don M. Multichannel time-interval measurement with a field programmable gate array (FPGA) device. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2018. Report No.: ARL-TR-8602.

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

This appendix includes the following Verilog files:

1) `decom_acc`: the original accumulator code
2) `decom_acc1`: the extra DIN word added
3) `decom_acc2`: continuous output added
4) `dcc_chain_tb_din`: the test bench
5) `decom_acc3`: high-resolution time-stamping added

```
//take 16 bit words, and load into 32 samples to output to PC
//add on extra words at end of each frame
module decom_acc(
clk,
reset,
data_in,
ld_in,
data_out,
ld_out,
clk_div,
rssi,
lastw,
time_in,
ave_in
);

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;
```

```
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;
```

```
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
          //byte order switched so that correct order is received on PC
          data_out[31:16] <= {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
      cnt <= 0;
      state <= LD2;
   end else begin
      cnt <= cnt+1;
   end
end
LD2 : begin
   rssi_reg<=rssi;  //load next word, assert ld to load 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  //just did load, wait to load 2cd slot
   ld_out <= 1'b0;
   if(cnt == clk_div) begin
      cnt <= 0;
      state <= LD1;
   end else begin
      cnt <= cnt+1;
   end
end

DO_RSSI10: begin  //loaded 1st 16 bit value, do next 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 //1st value, 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 //2cd slot, 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 bits, go back to words in 1st slot
  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 <= WAIT2;
end

DO_TIME20: begin  //do next 2cd slot 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 bits and ld, go to TM words, 1st slot
        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  //This is 1st value, go to TM words in 2cd slot
        data_out[31:16] <= {ave_reg[7:0],ave_reg[15:8]};
        ld_out <= 1'b0;
        state <= WAIT1;
      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
module decom_acc1(
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[31:16] <= {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
    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,
  DO_TIME10 = 9,
  DO_TIME11 = 10,
  DO_TIME12 = 11,
  DO_TIME20 = 12,
  DO_TIME21 = 13,
  DO_TIME22 = 14,
  MAKE_FRAME = 15,
  DO_TIME13 = 16;



  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;
        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_sfid,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 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
        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
 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  //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;
    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
    end
  end

endmodule



//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); // Scale factor

   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_iii5p7 #(.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(),
     .config_reg(config_reg));

   always @(posedge clk) begin
     if(reset) begin
       time_reg<=64'h000A000B000C000D;
     end else begin
       time_reg<=time_reg+1;
     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
//output continuously
//add high-resolution din data to dummy frames
module decom_acc3(
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 [63:0] time_reg2;
reg from_LD1;
reg high_bits;
reg do_dummy;

reg [15:0] dummy;
wire [9:0] nwords2;
assign nwords2 = {2'd0,nwords};

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,
  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,
  MAKE_FRAME_ST = 15,
  DO_TIME13 = 16,
  RST2=0,
  WAIT2=1;


  wire [9:0] fifo_cnt;
  wire [15:0] fifo_dout;
  reg [127:0] data_in2;
  wire [15:0] fifo_dout2;
  reg rd,rd2,ld_in2;
  wire full,empty;
  reg [7:0] din2;


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));

din_fifo ddc_output_fifo2
   (.rst(reset), .wr_clk(clk), .rd_clk(clk), .din(data_in2),
.wr_en(ld_in2),
   .rd_en(rd2), .dout(fifo_dout2), .full(full), .empty(empty));

reg [3:0] state;
reg [3:0] state2;

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


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

     //run bit counter
    if (cnt  == clk_div-1)
      if (bcnt == nbits-1) //nbits = 16 for word size of 16 bits
         bcnt <= 0;
  else
      bcnt <= bcnt+1;

//run word counter
if (bcnt == nbits-1 && cnt  == clk_div-1)
     if (wcnt == nwords)  //nwords = 47 for frame size of 48
         wcnt <= 0;
     else
         wcnt <= wcnt+1;

case(wcnt)
    0 : dummy<=sync[31:16];
    1 : dummy<=sync[15:0];
    2 : dummy<={dummy_sfid,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
            if (empty == 0 && wcnt > 2 && wcnt < (nwords-1)) begin
            //if din fifo has words:
               data_out[31:16]<=fifo_dout2;
               rd2<=1;
            end else
```

```
                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 else begin
         if (do_dummy == 1) begin
             if (empty == 0 && wcnt > 2 && wcnt < (nwords-1)) begin
             //if din fifo has words:
                data_out[15:0]<=fifo_dout2;
                rd2<=1;
             end else
                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
     high_bits<=~high_bits;
 end else begin
         ld_out <= 1'b0;
         rd<=  1'b0;
         rd2<=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 that will output a dummy frame every once in a while
 //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
     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 // 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 // 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  //This is 1st value, do back to words in 2cd slot
    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_ST;
    high_bits<=0;
 if (fifo_cnt > nwords2) begin
     do_dummy<=0;
     cnt <= 0;
     bcnt<=0;
     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 // go back to words, start with 1st slot
        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  //go back to words 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_ST;
        if (fifo_cnt > nwords2) begin
            do_dummy<=0;
            cnt <= 0;
            bcnt<=0;
            wcnt<=0;
        end else
            do_dummy<=1;
      end

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

 //create delayed version of din
 always @(posedge clk) begin
   din2<=din;
 end

//DIN state machine:
 always @(posedge clk) begin
    if((reset == 1'b 1)) begin
```

```
      state2 <= RST2;

    end

    else begin

      case(state2)


      RST2 : begin

        ld_in2<=0;

        data_in2 <= 0;

        state2 <= WAIT2;

      end


      WAIT2 : begin

        //load din fifo when change in din

        if (din2 != din) begin


   data_in2<={16'h0123,16'h4567,16'h89AB,din2,din,time_in[55:48],time_in[
63:56],time_in[39:32],time_in[47:40],time_in[23:16],time_in[31:24],time_i
n[7:0],time_in[15:8]};

          ld_in2<=1;

        end else

          ld_in2<=0;

      end


     default : begin

        state2 <= RST;

      end

      endcase

    end

  end

endmodule
```

Appendix B. MATLAB Data Analysis Scripts

This appendix includes the following MATLAB files:

1) `decode_telemetry_din.m`: A script that sets example parameters, runs the `decode_bin_file_function` function, and plots results.
2) `decode_bin_file_function.m`: A function that takes a binary SDR receiver file, and extracts telemetry frames, including frame time-stamps, and high-resolution time-stamps included in dummy frames.

```
%  decode_telemetry_din.m, an example telemetry decode script

clear

close all

tic

filename='test_din3.bin';

seconds_into_file=0;

duration=inf;

do_plot=1;


data_rate=4e6;

Fs=52e6; %SDR sampling clock freq

time_zone=-5; %offset from UTC

sync=[hex2dec('FE') hex2dec('6B') hex2dec('28') hex2dec('40')]';
% SYNC in bytes

dummy_SFID=hex2dec('FF');

do_crc=0;

WordsPerFrame=48;

BitsPerWord=16;

extra_words=8; %number of extra words SDR tacks onto end of
frames

[words,bytes,irig,synci,bytesf,din_irig, din_HR] =
decode_bin_file_function(filename,WordsPerFrame,Fs,time_zone,extr
a_words,sync,seconds_into_file,duration,data_rate,dummy_SFID);

[~,nframes]=size(words);


din=false(8,length(irig));

times=zeros(8,1);
```

```
for i=1:8
  din(i,:)=bitand(uint16(2^(i-
1)),words(WordsPerFrame+extra_words,:))>0;
  index=find(diff(din(i,:)),1);
  if ~isempty(index)
    times(i)=irig(index);
  else
    times(i)=nan;
  end
end

if do_plot
  figure(1)
  subplot(3,1,1)
  plot(irig,words(WordsPerFrame-1,:))
  xlim([irig(1) irig(end)])
  title('Frame Count');

  subplot(3,1,2)
  plot(irig,words(WordsPerFrame+extra_words,:))
  xlim([irig(1) irig(end)])
  title('Digital byte')

  subplot(3,1,3)
  plot(irig,irig)
  xlim([irig(1) irig(end)])
  title('Irig Time')

  figure(2)
  for i=1:8
    plot(irig, din(i,:))
    hold on
  end
  xlim([irig(1) irig(end)])
```

```
    for i=1:8
      legendtext{i}=sprintf('DIN%d',i-1);
    end
    title('DIN');
    legend(legendtext,'Location','NorthEastOutside')

    figure(3)
    din_bit1=bitget(din_HR,1);
    plot(irig-irig(1), din(1,:))
    ylim([-0.1 1.1])
    %xlim([-0.0 3.5])
    hold on
    plot(din_irig-irig(1), din_bit1,'--')
    xlabel('Seconds')
    ylabel('Bits')
    legend('Low Res', 'High Res', 'Location', 'SouthWest')
end

times
times_from_din0=times-times(1)
tstart=irig(1)
duration= irig(end)-irig(1)
toc




%% decode_bin_file_function.m, decodes SDR telemetry file
%duration and seconds_into_file assumes no missing frames
function [words,bytes,irig,synci,bytesf,din_irig, din_HR] =
decode_bin_file_function(filename,WordsPerFrame,Fs,time_zone,extr
a_words,sync,seconds_into_file,duration,data_rate,dummy_SFID)


%Read file
fstart=seconds_into_file*(data_rate/16/WordsPerFrame);   %frames
per sec * seconds = frame to start at
```

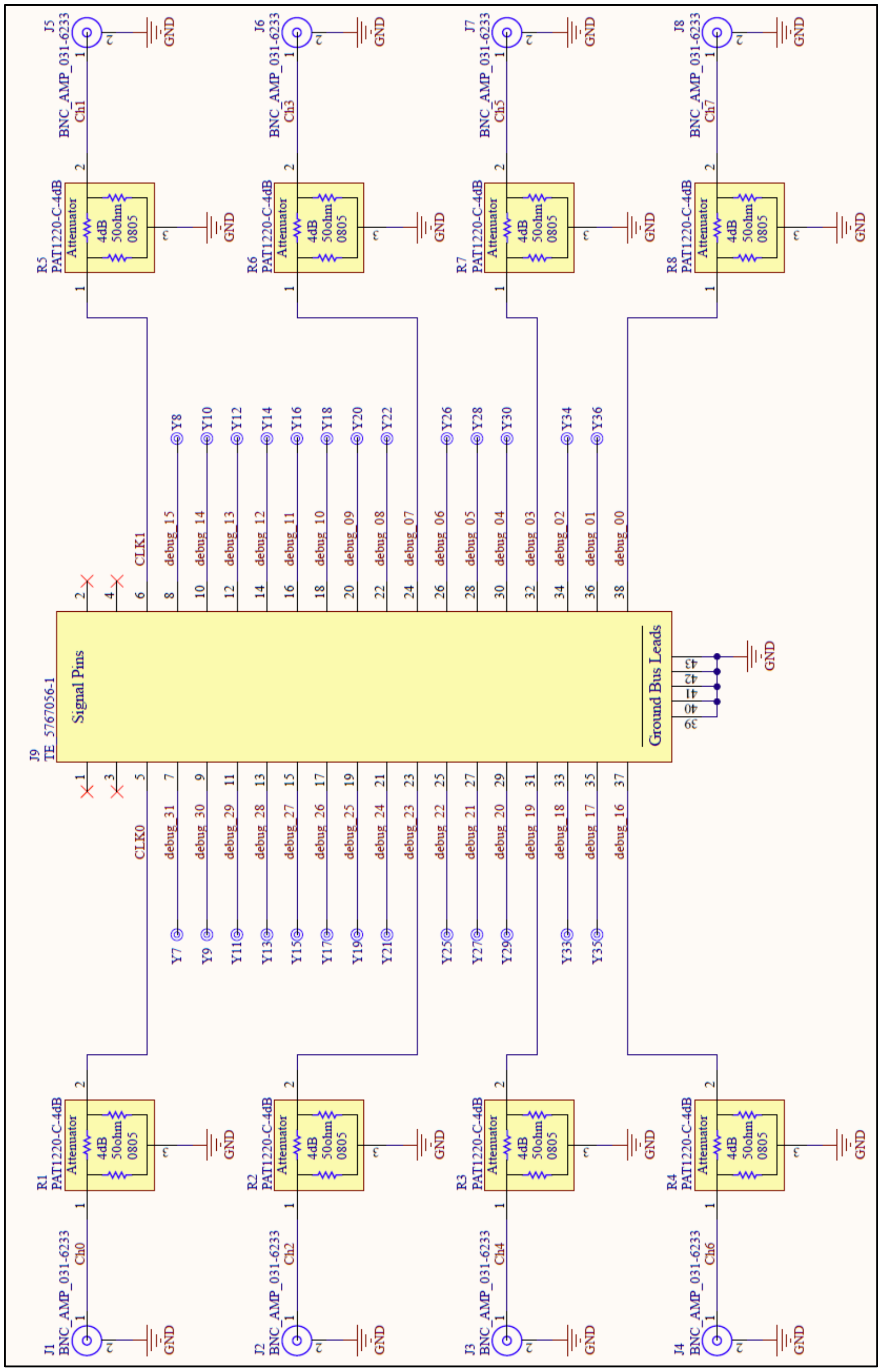
```
bstart= fstart*(WordsPerFrame+extra_words)*2; %byte to start at

nframes=duration*(data_rate/16/WordsPerFrame);

nbytes=nframes*(WordsPerFrame+extra_words)*2;

fid = fopen(filename, 'r');

fseek(fid,round(bstart),'bof');

bytes = fread(fid,nbytes,'*uint8');

fclose(fid);


%Find sync indexes

synci=find((bytes(1:end-3)==sync(1)) & (bytes(2:end-2)==sync(2))
..

   & (bytes(3:end-1)==sync(3)) & (bytes(4:end)==sync(4)));

synci=synci(1:end-1);


% dec2hex(bytes(synci(6):synci(6)+10))   8 is real


%Fill in words matrix

nframes=length(synci);

words=zeros((WordsPerFrame+extra_words)*2,nframes,'uint8');

if (synci(nframes)+(WordsPerFrame+extra_words)*2-1) >
length(bytes)

   synci=synci(1:end-1);

   nframes=nframes-1;

end

for i=1:nframes

   words(:,i)=bytes(synci(i):synci(i)+(WordsPerFrame+extra_words)*
2-1);

end


%Reshape as 16 bit words

bytesf=words;

words=typecast(words(:),'uint16')';

words=swapbytes(words);

words=reshape(words,WordsPerFrame+extra_words,[]);

bytesf=reshape(bytesf,(WordsPerFrame+extra_words)*2,[]);
```

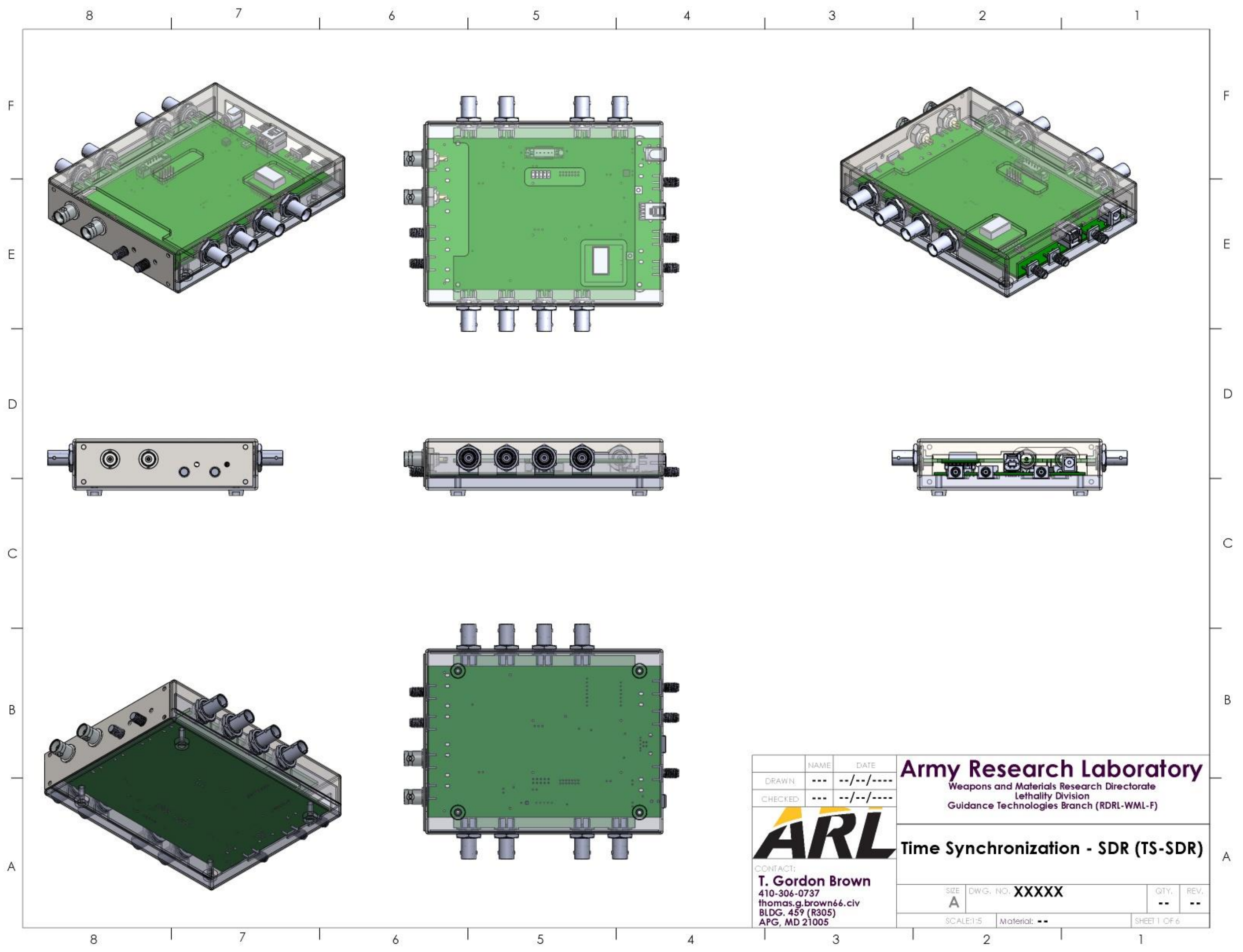
```
%bytesf=bytesf(1:WordsPerFrame*2,:);


%Create time signal

time64 = (words((6+WordsPerFrame):-1:(3+WordsPerFrame),:));

time64 = time64(:);

time64 = typecast(time64,'uint64');

time = double(time64)/Fs;


% display real start time, then start time vector at 0

% 0 is 1/1/70 00:00:00 from usrp

sec_fract=sprintf('%0.12g',time(1)-floor(time(1)));

start_time=datestr(time_zone/24+time(1)/(24*60*60)+datenum('1-
Jan-1970'),'mmmm dd, yyyy HH:MM:SS');

start_time=[start_time '.' sec_fract(3:end)];

irig=rem(time+time_zone*3600,24*3600)';  %seconds from start of
day, local time

%words=words(1:WordsPerFrame,:); %get rid of extra words


%find high res din time-stamps

dummy=bytesf(:,bytesf(5,:)==dummy_SFID);

dummy2=dummy(7:WordsPerFrame*2-4,:);

dummy2=dummy2(:);

%Find sync indexes

sync=[hex2dec('23') hex2dec('01') hex2dec('67') hex2dec('45')]';
% SYNC in bytes

synci_din=find((dummy2(1:end-3)==sync(1)) & (dummy2(2:end-
2)==sync(2)) & (dummy2(3:end-1)==sync(3)) &
(dummy2(4:end)==sync(4)));

if isempty(synci_din)

   din_irig=[];

   din_HR=[];

else

   synci_din=synci_din(1:end-1);

   din_words=zeros(16,length(synci_din),'uint8');

   for i=1:length(synci_din)

      din_words(:,i)=dummy2(synci_din(i):synci_din(i)+15);
```

```
    end
    time64 = (din_words(16:-1:9,:));
    time64 = time64(:);
    time64 = typecast(time64,'uint64');
    din_time = double(time64)/Fs;
    sec_fract=sprintf('%0.12g',din_time(1)-floor(din_time(1)));
    din_irig=rem(din_time+time_zone*3600,24*3600)';  %seconds from
start of day, local time
    din_irig = [din_irig-3/Fs; din_irig-2/Fs];
    din_irig=din_irig(:);

    din_HR = din_words(8:-1:7,:);
    din_HR=din_HR(:)';
end
end
```

Appendix C. Mechanical and Electrical Design

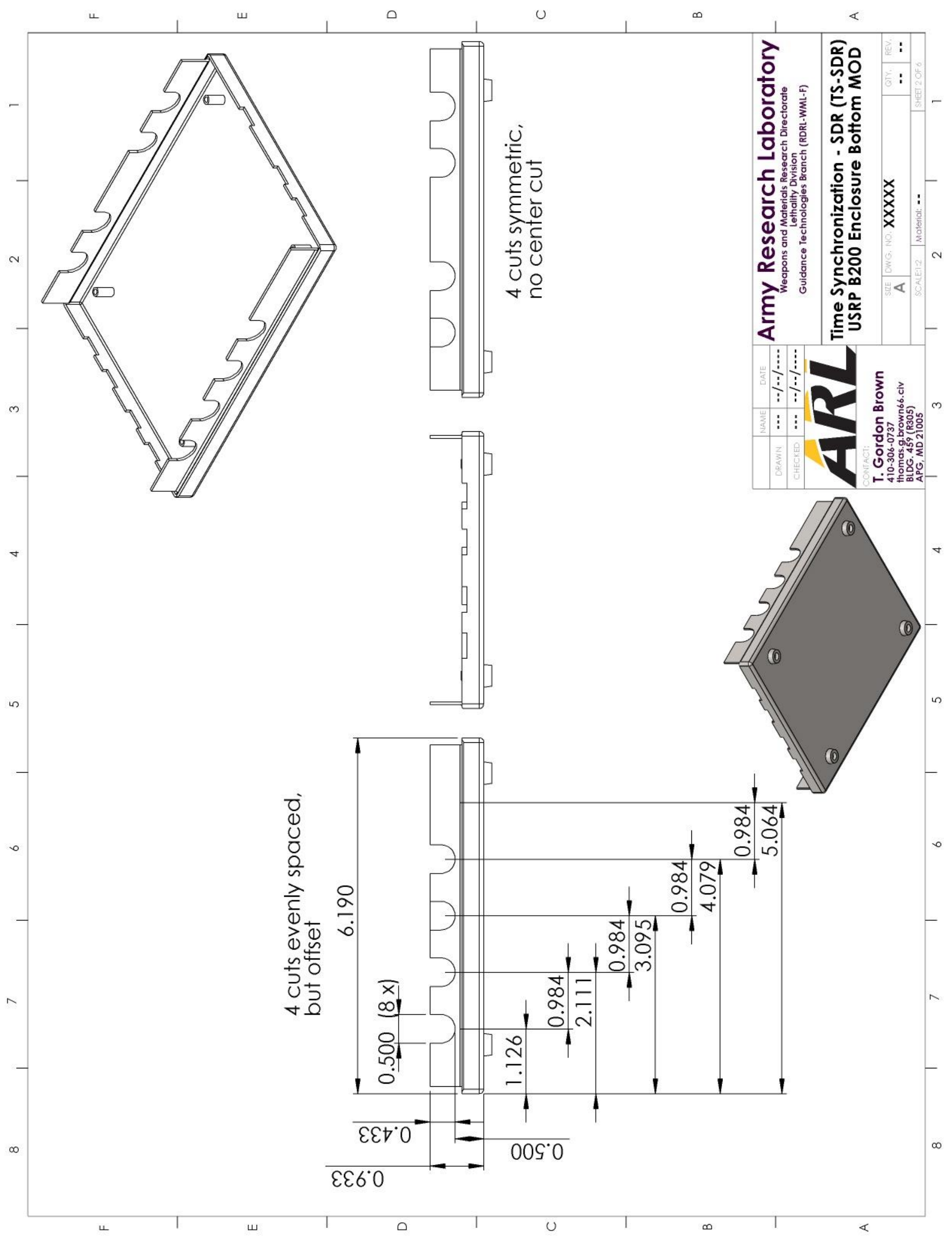
This appendix includes electrical and mechanical design drawings and modifications needed to reproduce the modified software-defined radio (SDR).


J1 BNC_AMP_031-6233 Ch0
J2 BNC_AMP_031-6233 Ch2
J3 BNC_AMP_031-6233 Ch4
J4 BNC_AMP_031-6233 Ch6
J5 BNC_AMP_031-6233 Ch1
J6 BNC_AMP_031-6233 Ch3
J7 BNC_AMP_031-6233 Ch5
J8 BNC_AMP_031-6233 Ch7
R1 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R2 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R3 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R4 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R5 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R6 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R7 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
R8 PAT1220-C-4dB Attenuator 4dB 50ohm 0805
GND
J9 TE 5767056-1
Signal Pins
Ground Bus Leads
CLK0
CLK1
debug_00 … debug_31
Y7 Y9 Y11 Y13 Y15 Y17 Y19 Y21 Y25 Y27 Y29 Y33 Y35
Y8 Y10 Y12 Y14 Y16 Y18 Y20 Y22 Y26 Y28 Y30 Y34 Y36

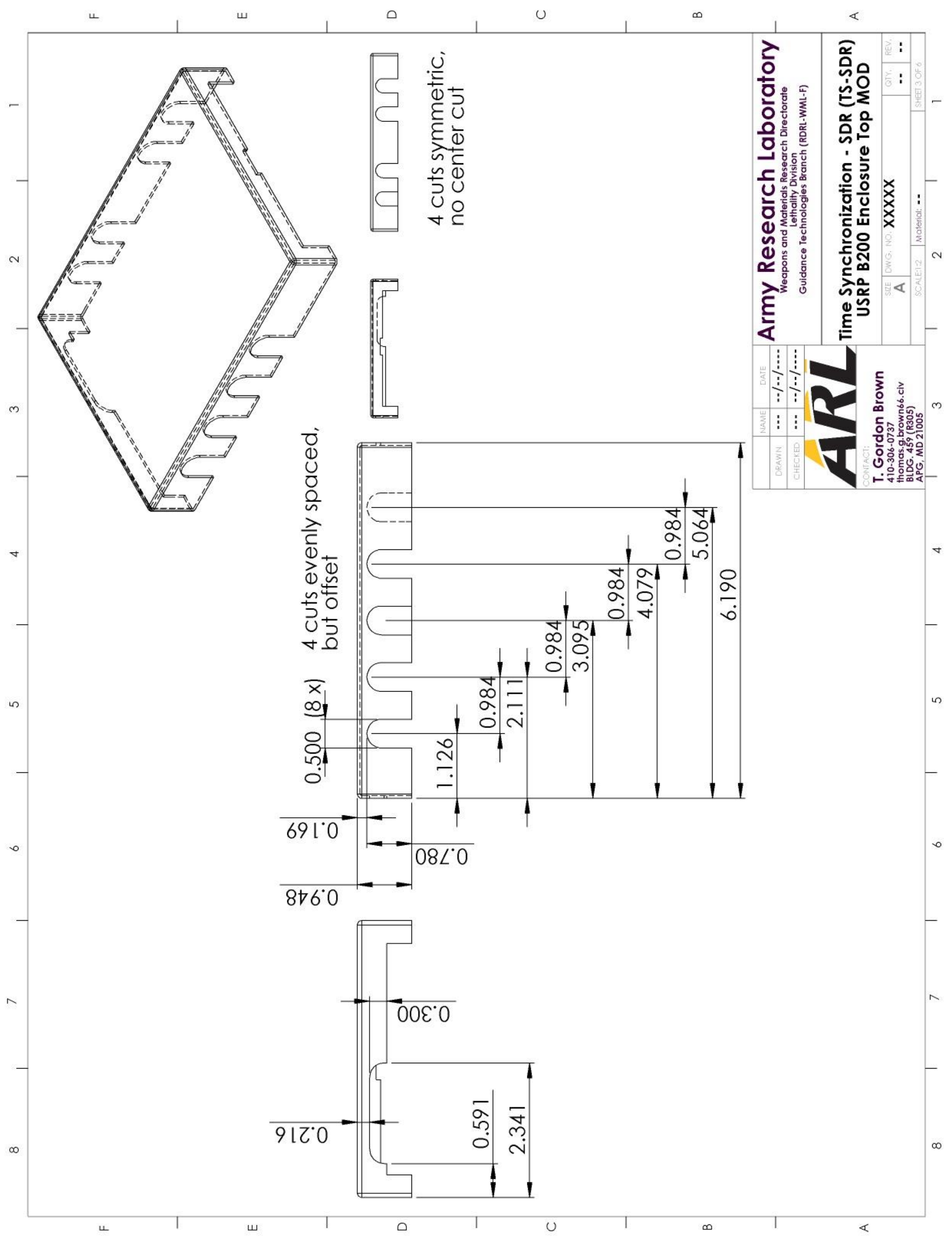

8
7
6
5
4
3
2
1
F
E
D
C
B
A
NAME
DATE
DRAWN

--/--/----
CHECKED

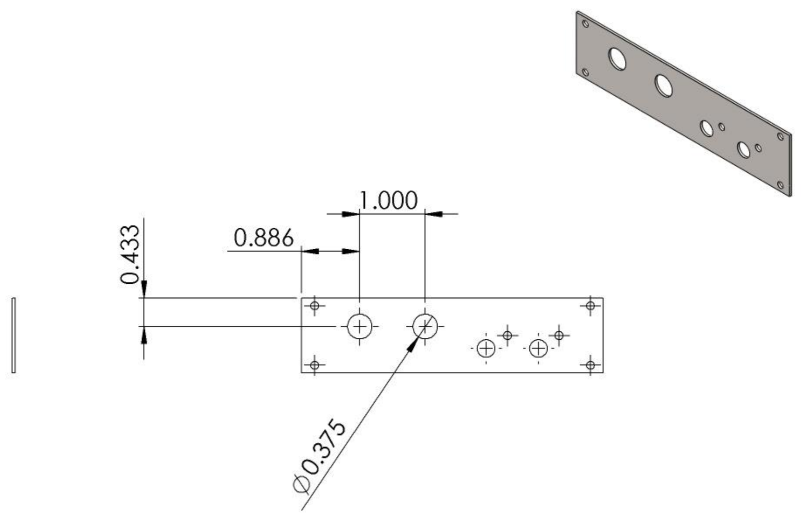
--/--/----
Army Research Laboratory
Weapons and Materials Research Directorate
Lethality Division
Guidance Technologies Branch (RDRL-WML-F)
ARL
Time Synchronization - SDR (TS-SDR)
CONTACT:
T. Gordon Brown
410-306-0737
thomas.g.brown66.civ
BLDG. 459 (R305)
APG, MD 21005
SIZE
A
DWG. NO. XXXXX
QTY.
--
REV.
--
SCALE:1:5
Material: --
SHEET 1 OF 6



4 cuts symmetric, no center cut
4 cuts evenly spaced, but offset
6.190
0.500 (8 x)
0.984
0.984
0.984
0.984
1.126
2.111
3.095
4.079
5.064
0.433
0.500
0.933
Army Research Laboratory
Weapons and Materials Research Directorate
Lethality Division
Guidance Technologies Branch (RDRL-WML-F)
Time Synchronization - SDR (TS-SDR)
USRP B200 Enclosure Bottom MOD
DWG. NO. XXXXX
SIZE A
SCALE:1:2
Material: --
QTY. --
REV. --
SHEET 2 OF 6
DRAWN
CHECKED
NAME
DATE
CONTACT:
T. Gordon Brown
410-306-0737
thomas.g.brown66.civ
BLDG. 459 (R305)
APG, MD 21005

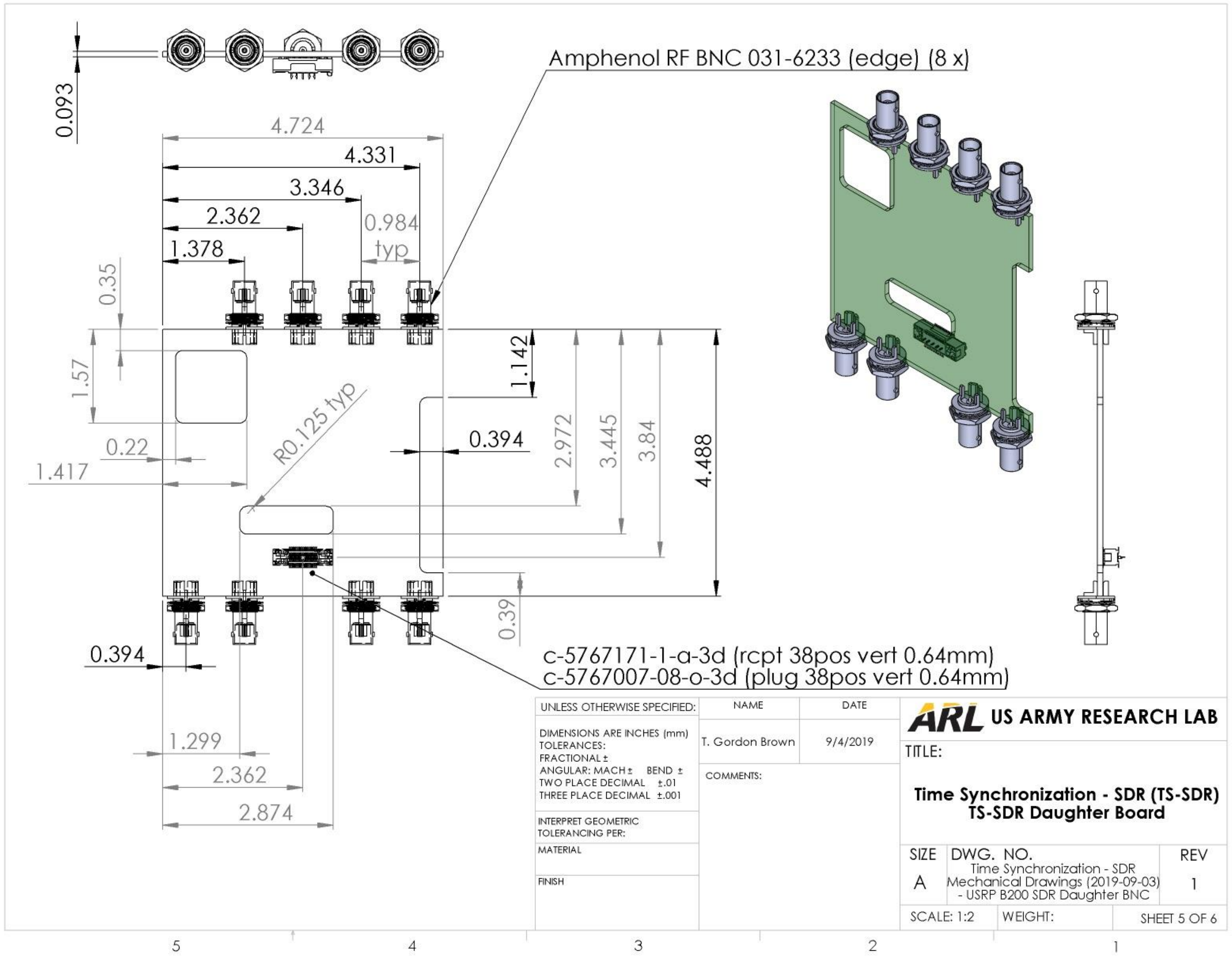


4 cuts symmetric, no center cut
4 cuts evenly spaced, but offset
0.500 (8 x)
1.126
0.984
2.111
0.984
3.095
0.984
4.079
0.984
5.064
6.190
0.169
0.780
0.948
0.300
0.216
0.591
2.341
Army Research Laboratory
Weapons and Materials Research Directorate
Lethality Division
Guidance Technologies Branch (RDRL-WML-F)
Time Synchronization - SDR (TS-SDR)
USRP B200 Enclosure Top MOD
NAME
DATE
DRAWN
CHECKED
CONTACT:
T. Gordon Brown
410-306-0737
thomas.g.brown66.civ
BLDG. 459 (R305)
APG, MD 21005
SIZE A
DWG. NO. XXXXX
QTY. --
REV. --
SCALE:1:2
Material: --
SHEET 3 OF 6



UNLESS OTHERWISE SPECIFIED:	NAME	DATE	ARL US ARMY RESEARCH LAB
DIMENSIONS ARE INCHES (MM) TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.001	T. Gordon Brown	9/4/2019	TITLE: **Time Synchronization - SDR (TS-SDR) USRP B200 Enclosure Side MOD**
INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:		
MATERIAL			SIZE A
FINISH			DWG. NO. Time Synchronization - SDR Mechanical Drawings (2019-09-03) - USRP B200 SDR Daughter BNC
			REV 1
			SCALE: 1:2
			WEIGHT:
			SHEET 4 OF 6

5 4 3 2 1



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE INCHES (mm)
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.001

INTERPRET GEOMETRIC TOLERANCING PER:

MATERIAL

FINISH

NAME	DATE
T. Gordon Brown	9/4/2019

COMMENTS:

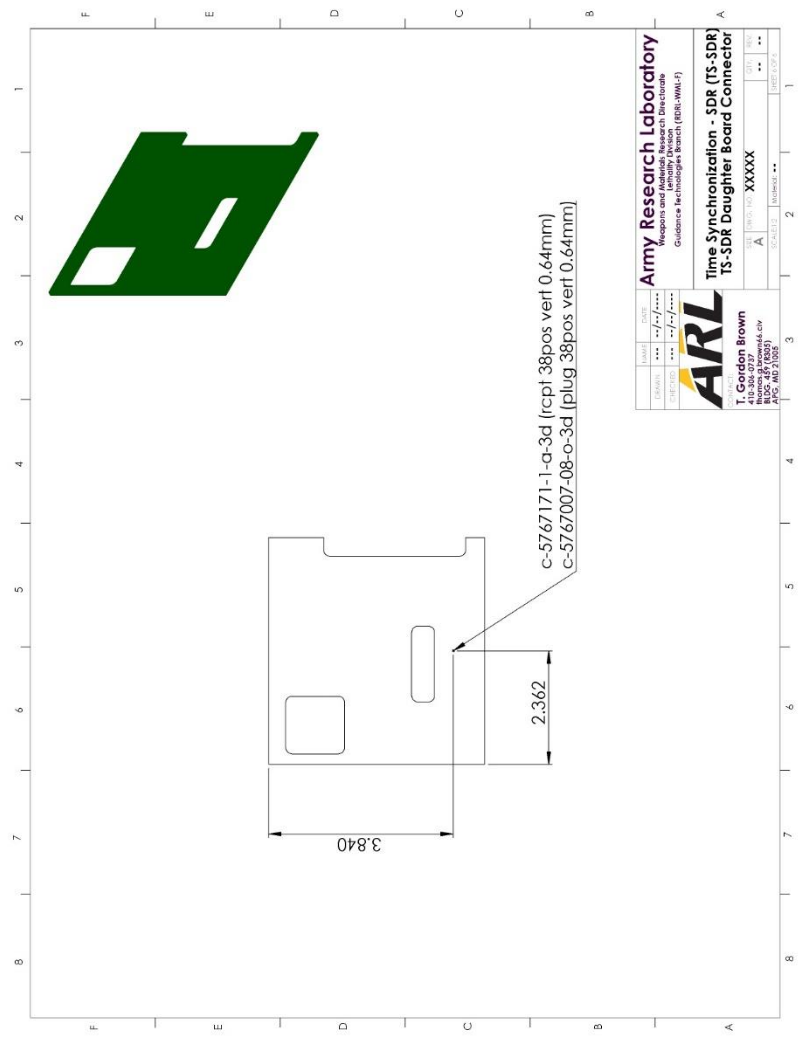
US ARMY RESEARCH LAB

TITLE:

Time Synchronization - SDR (TS-SDR)
TS-SDR Daughter Board

SIZE	DWG. NO.	REV
A	Time Synchronization - SDR Mechanical Drawings (2019-09-03) - USRP B200 SDR Daughter BNC	1

SCALE: 1:2 WEIGHT: SHEET 5 OF 6


c-5767171-1-a-3d (rcpt 38pos vert 0.64mm)
c-5767007-08-o-3d (plug 38pos vert 0.64mm)
2.362
3.840
Army Research Laboratory
Weapons and Materials Research Directorate
Lethality Division
Guidance Technologies Branch (RDRL-WML-F)
Time Synchronization - SDR (TS-SDR)
TS-SDR Daughter Board Connector
DRAWN
CHECKED
NAME
DATE
CONTACT:
T. Gordon Brown
410-306-0737
thomas.g.brown66.civ
BLDG. 459 (R305)
APG, MD 21005
SIZE A
DWG. NO. XXXXX
QTY. --
REV. --
SCALE:1:2
Material: --
SHEET 6 OF 6

List of Symbols, Abbreviations, and Acronyms

ADC	analog-to-digital converter
AES	advanced encryption standard
ARL	Army Research Laboratory
BoB	breakout-board
CPU	central processing unit
DC	direct current
DDC	digital down converter
FIFO	first in, first out buffer
FM	frequency modulation
FPGA	field-programmable gate array
GMT	Greenwich mean time
GPS	global positioning system
GPSDO	global positioning system disciplined oscillator
GUI	graphical user interface
IC	integrated circuit
IO	input/output
I/Q	in-phase/quadrature
IR	infrared
ISE	Integrated Synthesis Environment
PC	personal computer
PCB	printed circuit board
PCM	pulse code modulated
RAM	random access memory
RF	radio frequency
RSSI	received signal strength indicator

SDR	software-defined radio
SFID	sub-frame identifier
UCF	user constraint file
UDP	user datagram protocol
UHD	Universal Software Radio Peripheral hardware driver
USRP	Universal Software Radio Peripheral

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	CCDC ARL FCDD RLD CL TECH LIB
23 (PDF)	CCDC ARL FCDD RLW LF B ALLIK B J ACKER T G BROWN S BUGGS E BUKOWSKI J COLLINS J CONDON B DAVIS M DON D EVERSON R HALL J HALLAMEYER M HAMAOUI T HARKINS M ILG B KLINE J MALEY C MILLER B NELSON D PETRICK K PUGH N SCHOMER B TOPPER