SOFTWARE DEFINED RADIO IMPLEMENTATION OF A DVB-S TRANSCEIVER

Ashwin Amanna, James Bohl, Zachary Goldsmith (ANDRO Computational Solutions, LLC, Rome, NY, USA; aamanna@androcs.com, jbohl@androcs.com, zgoldsmith@androcs.com); Michael Gudaitis, Benjamin Kraines, Robert DiMeo, William Lipe, Richard Butler II (Air Force Research Lab RIT, Rome, NY, USA; michael.gudaitis@us.af.mil, benjamin.kraines.1@us.af.mil, robert.dimeo@us.af.mil, william.lipe@us.af.mil, richard.butler.10@us.af.mil)

ABSTRACT

Most waveforms operating on Software Defined Radios (SDR) are actually 'firmware' defined implementations with significant elements operating in the FPGA on closed platforms. This diminishes the full potential of SDR in terms of rapid development time cycles, accessibility to waveform code, and adaptable rapid reconfiguration. We address the challenges of rapid waveform development on SDR using the Digital Video Broadcast Satellite (DVB-S) standard as a case study with a true software defined implementation where all I/Q processing is performed on an Intel processor in conjunction with low-cost Ettus B205mini and Nuand bladeRF SDRs. We distill the waveform to core functional components and sequence the implementation into "sprints" while maintaining end-to-end functionality at each iteration. Innovations include strategic use of machine code for computational intensive operations and efficient multi-thread management. Our approach yielded a full transceiver implementation within 2 man-weeks using only the published specification for reference. The transmitter was tested for interoperability with a consumer satellite receiver. CPU usage on an i7 processor was approximately 22%, with a memory usage of 16MB out of the 8GB available. The mean latency of the system is approximately 50 milliseconds. A similar test showed that the system latency is affected by symbol rate and decreases as the rate is increased. When the symbol rate is set to 15 Msymbols/sec, the latency drops to 17 milliseconds.

1. INTRODUCTION

Early promises of software defined radios (SDR) included accelerated development timelines, greater accessibility to waveforms, and ease of modification. However, the general-purpose processors of the 1990's were not powerful enough to support complex waveforms that have demanding digital signal processing (DSP) algorithms. This led to implementations with firmware emphasis where most complex processing was performed in a FPGA. A drawback of this FPGA trend was lack of portability across different FPGA products, which results in closed systems, longer development times, and inability to adapt waveform code on-the-fly. Recent improvements in general purpose processors (GPP) to accommodate graphics processing allow GPPs to handle more DSP functions and thus enabling the original potential of SDR. As a case study for this GPP-based software approach, we present a 100% software implementation of a Digital Video Broadcast Satellite (DVB-S) standard transceiver on low-cost hardware with all I/Q processing performed in a general-purpose processor [1].

A practical challenge for affordability was to limit the total hardware costs to less than $3000 for a laboratory demonstration. This affordability constraint encouraged developers to seek innovative technical optimizations while working within the constraints of the low-cost hardware and processing-intensive operations of forward error correction (FEC) with the GPP. Government waveform efforts typically follow a traditional requirements-based acquisition cycle. We overcame this slower approach by adopting an agile-based development approach.

Our methodology to waveform development starts by distilling the waveform into its core functional components. We then simplify or eliminate blocks to implement an initial minimal functional prototype. From here, we incrementally add components and adapt existing elements to match the configurations defined in the waveform specification.

To achieve efficient operations on a GPP platform, we implement targeted functions in assembly code. Similarly, we divided the waveform code into multiple threads to equalize multi-core utilization. The Viterbi decoder, filtering and carrier recovery proved to be the most difficult to optimize. Interoperability was demonstrated using our transmitter with an off-the-shelf Coolsat DVB-S satellite receiver. We quantified bit error rate performance, CPU usage of transmitter and receiver signal flows, and measured latency.

We have shown that the GPP platform can achieve performance previously reserved for firmware implementations. Our agile-based development process yields waveforms significantly faster than a traditional requirements-based approach and proven through interoperability with off-the-shelf devices. The structure of this paper is as follows: we summarize existing software implementations of the DVB family and present our DVB-S implementation. Interoperability validation is described followed by benchmarking performance tests.

2. BACKGROUND

A literature search indicates several SDR implementations of DVB-related waveforms [2-4]. Most of the papers focus on DVB-Terrestrial (DVB-T). DVB-T is like DVB-S with similar data framing and error correction. Modulation is more complex with orthogonal frequency division multiplexing (OFDM) in DVB-T compared to QPSK in DVB-S.

Our implementation shares several similarities with [2] including 100% software implementation, leveraging multiple threads to optimize performance, and use of SIMD code to parallelize some functions. A key difference is in the multi-threading model. We are breaking up the processing into separate threads that pass data between each thread. They are using a thread pool with a main thread directing the individual threads to awaken when there is input available to process. In our approach, there is no main thread. Instead, the inter-thread buffers manage blocking/unblocking thread execution when an input/output buffer is available. Our approach should simplify waveform development as there is no need to set up the main thread and implement the logic for determining when data is ready for the threads. This logic is effectively implemented locally by the inter-thread buffers. The thread-management code is contained in a library that never needs to be modified by the waveform developer.

Their use of an older processor in [2] most likely limits SIMD to SSE while our more modern processor enables Advanced Vector Extensions (AVX). They are using multi-threading functions. We have only employed this technique for more complicated waveforms and found it unnecessary for DVB-S. Finally, we implemented a complete real-time receiver, while they created an offline receiver.

3. IMPLEMENTATION

The transceiver is based on DVB-S standard *EN 300 421 V1.1.2 (1997-08).* The transmitter block diagram is shown in Figure 1. A video file is encoded by VLC producing MPEG2 transport stream frames. Null frames are inserted in this stream as needed to adapt the bit rate of the video stream to the bit rate of the DVB-S transmitter. The randomizer operates on 8 MPEG2 frames at a time. The first byte in an MPEG2 frame is a synchronization marker. These synchronization markers are used directly by DVB-S for its own synchronization purposes. The synchronization byte in the first MPEG2 frame of each randomizer frame is inverted to indicate the start of the randomizer frame.

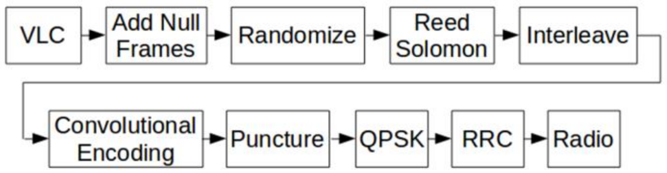

Figure 1. DVB-S Transmitter Block Diagram

Each 188-byte MPEG2 frame is Reed-Solomon encoded producing 204-byte frames. Frames are interleaved using a convolutional interleaver. The byte stream output from the interleaver is converted to a bit stream, most significant bit (MSB) first, and convolutionally encoded. The output of the convolutional encoder is punctured by the selected rate, either 1/2, 2/3, 4/5, 5/6, or 7/8. The encoded bit streams are QPSK modulated, RRC filtered, and transmitted by the radio.

The DVB-S receiver is shown in Figure 2. The system first removes DC offset introduced by the transmitter and receiver. This is performed twice, before the carrier synchronization to remove the receiver induced offset, and again after carrier synchronization to remove the transmitter induced component. The process is performed on blocks of 4096 consecutive samples. The average is calculated and then subtracted from each sample.

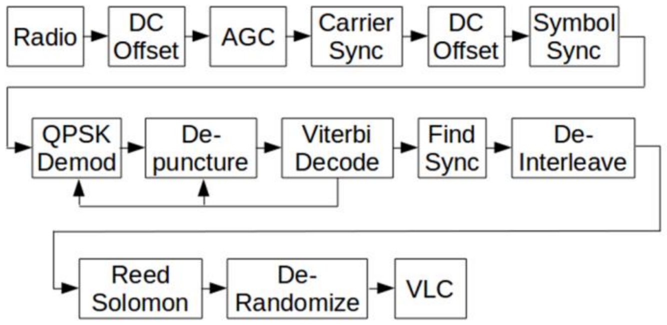

Figure 2. DVB-S Receiver Block Diagram

Automatic gain control (AGC) is similarly performed on blocks of 4096 samples to normalize the amplitude of the received signal. This is required to prevent a variation in carrier synchronization and symbol synchronization response time with the received signal level. Each sample is divided by the root mean squared (RMS) amplitude of the entire 4096 sample block.

Next, a phase-locked loop (PLL) removes the phase/frequency offset from the signal. The phase/frequency error is calculated by taking the 4th power of each input sample. This produces a strong impulse at 4x the frequency offset. This step initially showed high computation cost. Originally, this PLL was calculating the phase error and updating the feedback loop on every sample coming in. This resulted in an update rate of 25 million samples/second. Since the frequency offset is minor compared to the sample rate, it is not necessary to update the PLL feedback loop for every sample. The PLL was redesigned to calculate the phase error on a block of samples and then update the feedback loop for each block. The block size is currently set to 8 samples.

The same technique was applied to the symbol timing PLL. In this case, the maximum sample rate offset is smaller than the maximum frequency offset. Therefore, the block size can be made larger resulting in a greater reduction in computations. The block size is currently set to 1024 samples.

A similar approach is used for symbol synchronization. By squaring each sample, a strong frequency component is identified at the symbol rate. A PLL locks onto this frequency and optimal symbol timing is determined based on the phase of the PLL numerically controlled oscillator (NCO). A Root-Raised-Cosine (RRC) filter is again used to interpolate at the optimal time. Here, the RRC filter dot product operation is implemented in assembly code.

At this point, the symbols are QPSK demodulated producing two bit streams. These bits have the values +1/-1. The two bit streams are de-punctured which inserts 0 wherever a bit was punctured. The de-punctured bit streams are Viterbi decoded producing a single output bit stream. Since a QPSK constellation is symmetric over a 90-degree rotation, there may be a 90-degree phase ambiguity in the received symbols. To correct for this, the constellation is rotated periodically until synchronization is detected by the Viterbi decoder. Also, the de-puncturing must be correctly aligned with the incoming bit streams. To obtain the proper alignment, the de-puncturing is periodically shifted with respect to the incoming bit stream until the Viterbi decoder detects synchronization. The output bit stream from the Viterbi decoder is searched to locate the MPEG2 sync bytes. Once found, the bit stream is converted to bytes and passed into the de-interleaver. De-interleaved code-words are decoded by the Reed-Solomon decoder and de-randomized. The de-randomized code-words are sent to VLC to display the video.

4. LIMITATIONS

This DVB-S implementation has several limitations related to synchronization stability. Synchronization loss occurs intermittently with data transfers exceeding approximately 900,000 MPEG2 frames. When synchronization loss occurs, frames are dropped and bit errors are incurred in some received frames as synchronization stabilizes. Furthermore, these low-cost SDR platforms have limited hardware automatic gain control (AGC) functionality. To compensate for the limited AGC, calibration of transmitter and receiver gain is consequently sensitive and requires frequent tuning.

5. TEST PLATFORM

We have tested the DVB-S transceiver on software defined radios in cabled RF and over-the-air (OTA) wireless configurations. To demonstrate interoperability, we transmitted a video file to a commercial off the shelf (COTS) Coolsat receiver connected to a television. Table 1 lists configuration parameters necessary to recreate the system model.

Note that the bladeRF [5] is operating at 1GHz which is the intermediate frequency (IF) of the Satellite TV receiver. A label on the Coolsat receiver states that 950-2150MHz is the IF range. In a real system, there would be an upconverter to the KU band (to 11.7 to 12.7GHZ) at the transmitter. At the receiver (Coolsat), there would be a Ku downconverter. In this case we are only operating at 1GHz which is directly received by the Coolsat.

The low noise block (LNB) downconverter input also has 18 volts DC component. This DC voltage powers the downconverter when it is used. It is important to disable this DC voltage or use a DC blocking capacitor when directly connect this to the bladeRF SDR. At one point, we directly connected an attenuator to the LNB input and then connected it to the SDR which led to the attenuator getting hot to the touch. There is an option to disable the 18V DC which would allow direct connection.

The RF out of the satellite receiver connects directly to an old television. The LNB Input port of the receiver is connected to a coaxial cable. Soldering was required to connect the coaxial cable's center wire and ground to the antenna connector. We have found that the receiver sensitivity is good enough that the system receives video without the antenna on the Coolsat receiver.

Table 1. **Components Used in Demonstration System**

Item	Description
DVBS receiver	Coolsat 5000 Platinum
DVBS specification	EN 300 421 V1.1.2 (1997-08)
SDR platforms	Tested with BladeRF [5] and USRP B205MINI [6]
Linux distribution	Ubuntu 14.04.3
Linux kernel version	3.13.0
Software dependencies	• VLC 2.1.6-0-gea01d28 • libbladerf • libuhd
Antenna	OmniLOG 70600 Antenna
Receiver gain	Between 20dB and 40dB
Transmitter gain	Between 20dB and 35dB
Samples/symbol	2.25M
Sample rate	33.75M
Frame size	188 Bytes

6. RESULTS

Benchmarking tests included CPU usage, bit error rate, and latency. The Linux utility, *htop*, was used to monitor CPU usage of the transmitter software alone, the receiver alone, and both the transmitter and receiver running simultaneously. CPU usage was measured on an Intel Core i7 and i3, as indicated in Table 2.

Table 2. *htop* DVBS CPU Usage Results

Laptop	Features	HTop Reported CPU Usage out of 100%		
		Tx/Rx	Tx	Rx
Dell Precision M2800	Intel® Core™ i7-4610M CPU@3GHz 8GB RAM, 4CPUs	22%	8%	16%
Lenovo L530	Intel® Core™ i3-2348M CPU@2.3GHz 4GB RAM, 4CPUs	50%	24%	35%

Note that for quad-core processors, *htop* typically reports results out of a maximum of 400% based on the utilization of four cores. Here, results are normalized to 100% maximum. On average, the overall CPU usage for the entire system is relatively low. On a Dell Precision M2800 with an Intel i7 processor with 4 CPUs, the lowest usage percentage we achieved was 22% for the transceiver, 8% for the transmitter only, and 14% for the receiver only. Additionally, the RAM usage on the same platform remained constant at 0.2%, which is a total usage of 16MB out of the remaining 8GB.

Processor usage increases with symbol rate as shown in Figure 3. Normalized CPU usage increases as the symbol rate is increased from 5.5 to 15.5 symbols. Note that the transmitter usage is shown in blue and the receiver usage is shown in orange.

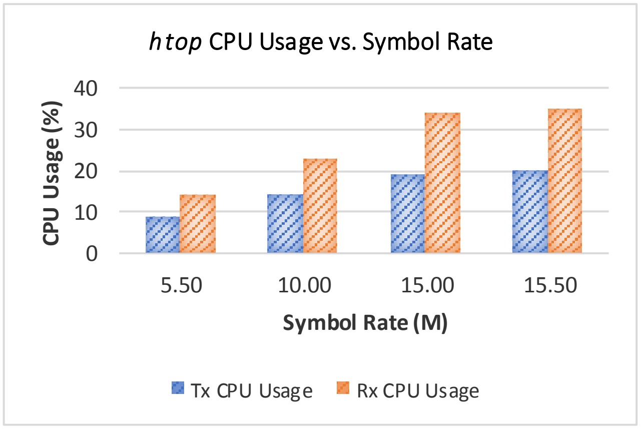

Figure 3. DVB-S *Htop* CPU Usage vs Symbol Rate

BER tests were conducted by measuring bits in error from fully synchronized MPEG2 frames at the receiver. Mock MPEG frames 188 bytes in length were transmitted in 2ms bursts with 10 frames/burst over a wired and wireless link. Under stable synchronization, 13 repetitions of 90,000 bursts (900,000 total frames) were received with no measured BER. In tests where synchronization was unstable, measured BER was on the order of 4.5E-5. Commercial DVB-S strives for BER on the order of 1E-10. To statistically measure levels that low, approximately 1E12 bits needs to be transmitted continuously. We were unable to transmit that much data at one time and maintain synchronization.

System latency is defined as the time between generating a MPEG2 frame and decoding a received MPEG2 frame, as shown in Figure 4. Table 3 shows the mean latency in the DVB-S transceiver in a wired environment (coaxial cable connecting the Tx and Rx RF ports) with increasing test burst sizes and a constant period.

Table 3. DVBS Transceiver Latency (Wired)

Bursts	Period	Mean Latency (µs)	Standard Deviation (µs)
100	100	49172	284
1000		49117	196
10000		49242	188

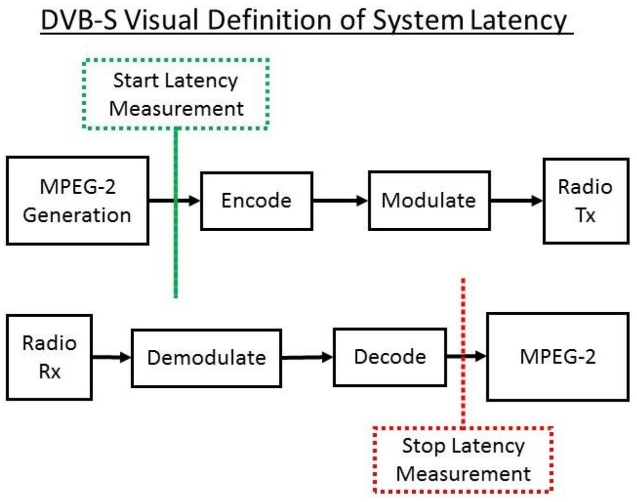

Figure 4. Latency Measurement Endpoints

The mean latency of our system hovers around 50 milliseconds despite the increase in the number of bursts that are sent. Although increasing burst size has no effect on the mean latency, it does reduce standard deviation of reported latency as more are sent. We further compare the mean latency of the DVB-S system to the set symbol rate shown in Figure 5 below.

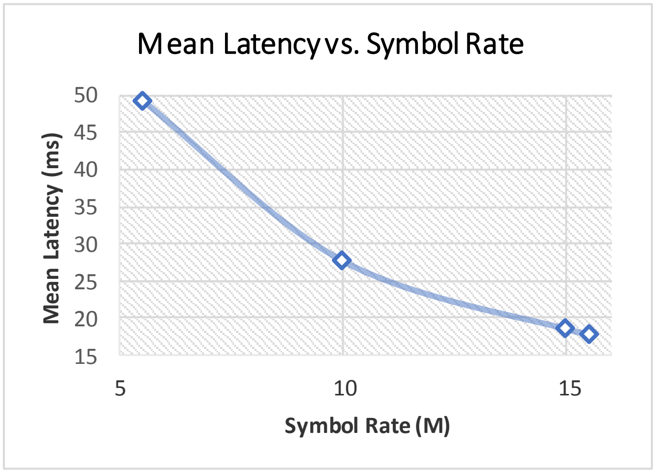

Figure 5. DVB-S Latency vs. Symbol Rate

The set symbol rate directly effects the mean latency of the system. As symbol rate is increased from 5.5 to 15.5 Msymbols, the mean latency decreases from 50ms to approximately 17ms.

7. CONCLUSION

We presented software implementation of a DVB-S transmitter and receiver where all I/Q processing is performed in GPP. Key elements include taking advantage of multiple processors through efficient threading and optimal usage of AVX instructions. Our agile approach to waveform development increases productivity and lends itself to faster timelines and real-time troubleshooting due to the flexibility of an all software implementation. Our approach has been successfully applied across multiple commercial and military waveforms.

We successfully showed that real-time operation of both transmitter and receiver I/Q digital processing is possible on an i3 or better processor. Benchmark results indicate peak normalized CPU usage at 50% for second generation i3 mobile processor, and 22% for a fourth generation i7 mobile processor. Bit error performance was comparable to the DVB-S waveform specification requirements when stable synchronization is achieved, with system latencies measured between 50ms and 17ms depending on the symbol rate.

8. ACKNOWLEDGEMENT

Approved for Public Release; Distribution Unlimited: 88ABW-2017-4648, 22 Sep 2017.This material is based upon work supported by Air Force Research Laboratory (AFRL/RITE) under Contract No. FA8750-15-C-0250. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author (s) and do not necessarily reflect the views of AFRL.

9. REFERENCES

[1] DVB-S Specification, EN 300 421 V1.1.2 (1997-08), European Telecommunications Standards Institute, www.etsi.org/deliver/etsi_en/300400_300499/300421/01.01.02_60/en_300421v010102p.pdf

[2] G. Baruffa, L. Rugini and P. Banelli, "Design and Validation of a Software Defined Radio Testbed for DVB-T Transmissions," Radioengineering, vol. 23, pp. 387-398, 2014.

[3] Y. Jiang, W. Xu and C. Grassman, "Implementing a DVB-T/H Receiver on a Software-Defined Radio Platform," International Journal of Digital Multimedia Broadcasting, vol. 009, p. 7, 2009.

[4] C. Fantozzi, L. Vangelista, D. Vorig and O. Campana, "SDR Implementation of a DVB-T2 transmitter: The core building blocks," IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, 2011.

[5] bladeRF Software Defined Radio (SDR), www.nuand.com

[6] USRP B205mini, Ettus Research, A National Instruments Company, www.ettus.com/product/details/USRP-B205mini-i