Field-programmable gate array implementation of a single photon-counting receive modem

William P. Simon*, Jennifer N. Downey, Nicholas C. Lantz, Thomas P. Bizon, Michael A. Marsden, Brian E. Vyhnalek, Daniel J. Zeleznikar
NASA Glenn Research Center, 21000 Brookpark Road, Cleveland, OH, USA 44135-3127

ABSTRACT

We present a field-programmable gate array (FPGA) implementation of a single photon-counting receive modem for a pulse position modulated signal. The modem is compliant with the Consultative Committee for Space Data Systems (CCSDS) High Photon Efficiency (HPE) Optical Communications Coding and Synchronization standard and is capable of a maximum data rate of 267 Mbps. The system is designed on a commercial off-the-shelf FPGA platform and utilizes superconducting nanowire single photon counting detectors, analog to digital converters (ADCs) to sample the detectors, and two FPGAs. Symbol timing recovery, photon counting, convolutional deinterleaving, and codeword synchronization are performed in the first FPGA. The second FPGA performs iterative decoding on each codeword of the serially concatenated pulse position modulated (SCPPM) signal. A digital filter is included to compensate for timing jitter of the detector, and the decoder throughput can be adjusted through reconfigurable parallelization. The decoder also implements a resource-efficient, algorithmic polynomial interleaver and deinterleaver. Both FPGAs can be reconfigured to switch between pulse position modulation (PPM)-16 and PPM-32 with code rates 1/3, 1/2, and 2/3. In this paper, we describe the receiver architecture and FPGA implementation of the timing recovery loop and SCPPM decoder, FPGA utilization for the different modes, and receive modem characterization test results.

Keywords: Field-programmable gate array, optical receiver, timing recovery, iterative decoder

1. INTRODUCTION

The National Aeronautics and Space Administration (NASA) is planning to incorporate photon-counting optical communication systems for deep space missions including the Optical Artemis-II Orion (O2O)[1] demonstration aboard the Artemis-II mission and the Psyche[2] mission into deep space. Current development for these optical communication demonstrations uses the Consultative Committee for Space Data Systems (CCSDS) High Photon Efficiency (HPE) Standard.[3] The CCSDS HPE standard uses serially concatenated pulse position modulation (SCPPM) with PPM orders 4, 8, 16, 32, 64, 128, and 256 with code rates 1/3, 1/2, and 2/3. The PPM pulse widths range from 512 ns to 125 ps, so the maximum data rate supported is ~2Gbps.

The NASA Glenn Research Center has developed a field-programmable gate array (FPGA) implementation of a photon-counting receive modem using commercial off the shelf (COTS) components that complies with a subset of modes from the CCSDS HPE standard. The FPGA-based receive modem samples pulses from the superconducting nanowire single-photon detectors. A fiber interconnect couples light from the telescope backend optics to the superconducting nanowire single-photon detectors (SNSPDs). The system currently supports a minimum 500 ps slot width and data rates up to 267 Mbps but is scalable up to 533 Mbps.

*Send correspondence to william.p.simon@nasa.gov

Presented in this paper are the receive modem FPGA implementation and integration details, resource utilization statistics, and performance results. A receive system overview is presented in Section 2. Symbol timing recovery implementation details are described in Section 3, and iterative decoder implementation details are described in Section 4. Performance results and resource utilization numbers are presented in Section 5.

2. RECEIVE SYSTEM OVERVIEW

The receive system delivers light over a fiber interconnect to a set of SNSPDs that output an electrical pulse when a photon is detected. The output of each SNSPD is amplified by a low-noise amplifier (LNA). The detectors have less than 100 ps full width half maximum (FWHM) of jitter and a 1/*e* reset time on the order of 15 ns. The output of each LNA is sampled with an analog-to-digital converter (ADC) in the FPGA-based receive modem. The modem performs symbol timing recovery and iterative decoding of the received codewords on two separate FPGAs as shown in Figure 1. The symbol timing recovery FPGA is responsible for sampling the SNSPD pulses using 16 analog-to-digital converters (ADCs) and feeding the samples to the programmable logic for photon counting, timing recovery and Doppler offset mitigation, convolutional deinterleaving, and SCPPM codeword synchronization. The decoder FPGA is responsible for iterative decoding, data frame synchronization, data link deframing, and the Gigabit (Gb) Ethernet interface. The symbol timing recovery and iterative decoder FPGAs are connected by an 8.25 Gbps high-speed serial link. The link is used to send codeword blocks from the symbol timing recovery FPGA to the decoder FPGA.

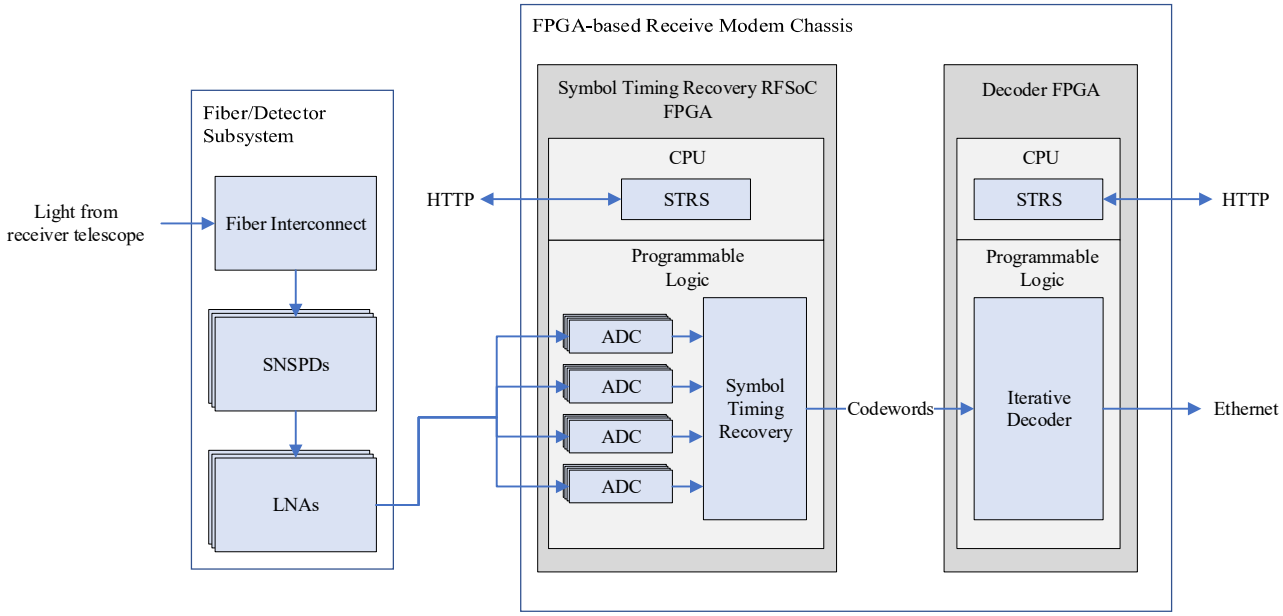


Figure 1. Receiver system block diagram showing both the fiber/detector subsystem and the FPGA-based receiver subsystem.

2.1 Receive Modem Configuration and Control

The receive modem FPGA cards are housed in a Micro Telecommunications Computing Architecture (MicroTCA) chassis. The timing recovery FPGA card uses a Xilinx Radio Frequency System on a Chip (RFSoC), and the decoderFPGA card uses a Virtex Ultrascale FPGA. The receive modem architecture is scalable and can be expanded to support higher data rates by adding additional FPGA cards to the chassis. Command, control, and status of the timing recovery FPGA card is handled by the embedded SoC central processing unit (CPU) over the Advanced eXtensible Interface (AXI). Command and status of the decoder FPGA is handled by a physically separate CPU, which has access to the FPGA fabric over AXI as well. The Space Telecommunications Radio System (STRS)[4] is used to query telemetry from both FPGA cards as well as provide the command interface to the modem shown in Figure 1. The telemetry adheres to a client-server networked architecture through the hypertext transfer protocol (HTTP) and utilizes standard data formats, such as plaintext or JavaScript Object Notation. The control software interacts with the underlying processor to send commands to and receive telemetry from the FPGA through memory-mapped registers.

3. SYMBOL TIMING RECOVERY

The symbol timing recovery, codeword alignment, and deinterleaving is implemented on a RFSoC FPGA. The FPGA has 16 ADCs which sample the output of the SNSPDs. The symbol timing recovery loop tracks differences in transmitter and receiver local oscillators as well as spacecraft Doppler. This receiver is designed to track and compensate for slot clock differences of up to 66 parts per million. Channel statistics are calculated and sent to the decoder FPGA along with the codeword slot counts. A block diagram of the timing recovery FPGA is shown in Figure 2.

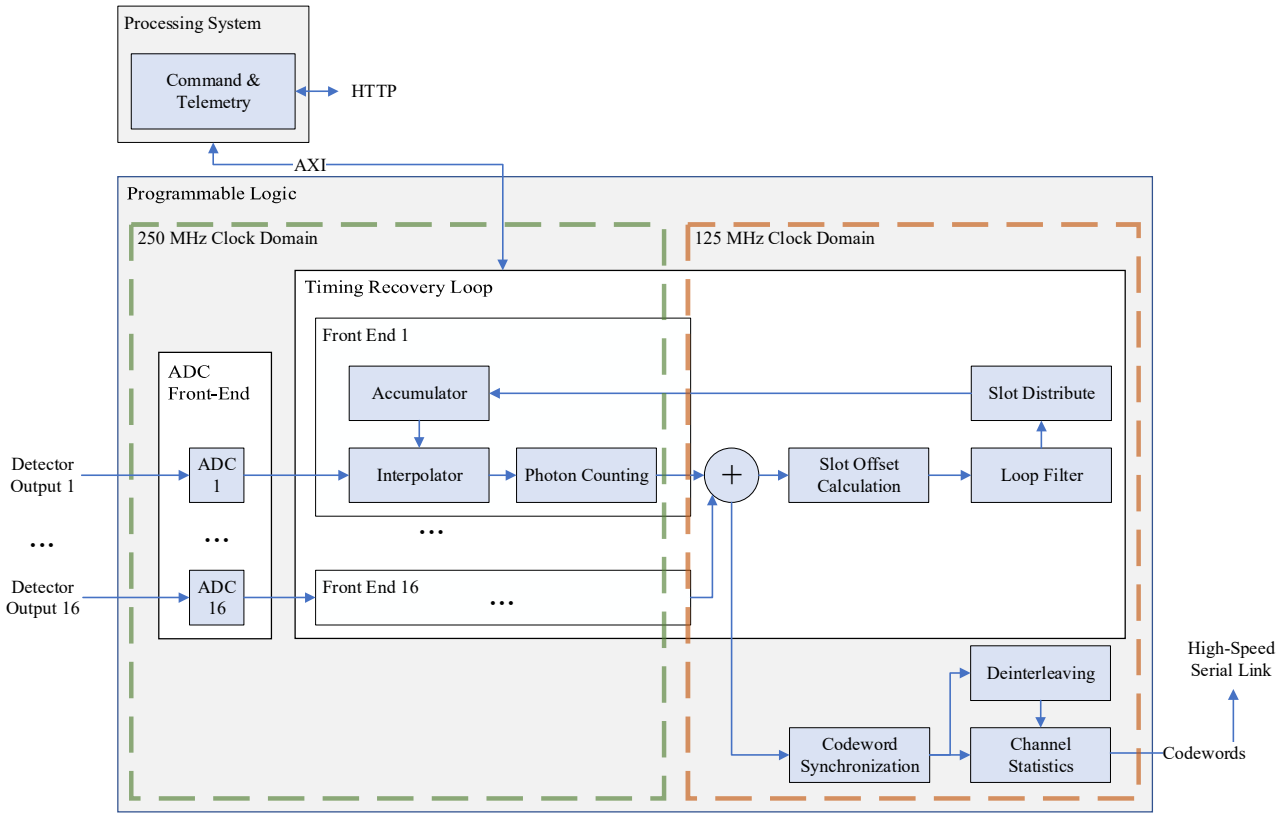


Figure 2. Timing recovery FPGA architecture with 16 ADC channels processed across a 250 MHz to a 125 MHz clock domain.

3.1 Analog to Digital Conversion

Each detector output is sampled by an ADC at a sample rate of 2 GSps. The programmable logic has 4 tiles of 4 ADCs each. The ADCs within a tile are time aligned to an external 2 GHz sample clock. The Xilinx Multi-Tile Synchronization application programmer interface is used to time align the tiles. Time alignment has been previously demonstrated to an accuracy of +/- 5 ps.[5] 10-bit samples are sent out of the ADCs in parallel groups of 8 at 250 MHz.

3.2 Sample Interpolation

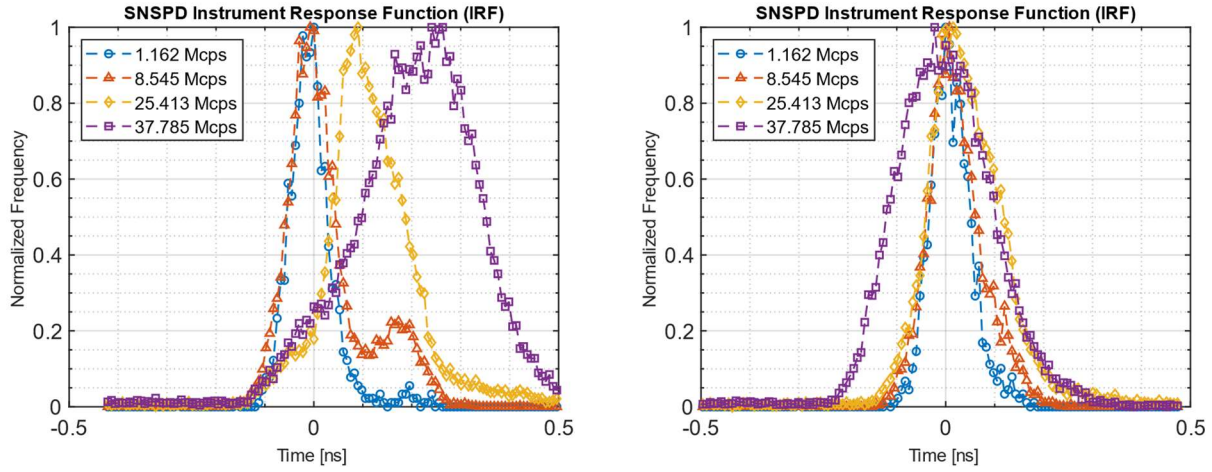
The detector ADC samples are linearly interpolated at the slot boundary. The interpolation point is calculated by the timing recovery loop. Each adjustment is applied to a group of 8 samples in parallel by first finding the slope of each subgroup of 2 contiguous samples and then resampling along the slope according to the adjustment. If the slot adjustment to the samples is greater than +0.5 or less than -0.5 of a sample, a sample will be removed or added to track the received signal. After the interpolator, a shift register is used to convert the data stream to 10 samples in parallel.

3.3 Photon Counting

At count rates up to approximately 10-20 Mcps the detectors introduce less than 100 ps FWHM of jitter between the incident photon arrival time and detection time. However, as the input photon flux increases, the width of the timing distribution increases which leads to receiver implementation loss. The amount of loss depends on the PPM order, slot

width, and the flux rate of the signal. In addition to SNSPD geometrical effects and other intrinsic effects, a portion of this jitter is caused by differences in electrical pulse amplitudes at the output of the detectors. Following a previous detection, if a new detection occurs before the full reset of the detector, the detector bias current will not have returned to its full operating level, and so the resulting pulse amplitude will vary. For a fixed threshold, this amplitude variation causes the SNSPD output pulses to be detected at different relative times along the rising edge of the pulse. The amplitude-induced detection time delay, referred to as pulse walk, introduces some timing uncertainty, and the photon may be counted in an incorrect slot. Figure 3 (a) shows the measured SNSPD instrument response functions (IRF) for a fixed detection threshold. The measured IRFs are the detection time distributions relative to an input 50 MHz repetition rate, 70 fs mode-locked fiber laser, attenuated from ~ 0.01 photon per pulse at the lowest rate to more than 1 photon per pulse at the highest input rate. In this test, the output of the SNSPD was sampled at a rate of 20 GSps and the samples were interpolated to find the time the pulse crossed the detection threshold. The figure shows the detection delay increases by ~250 ps at the highest count rate (37.785 Mcps). This is a significant drift for a 500 ps slot width.

A constant fraction discriminator[6] can be used to compensate for the timing uncertainty of pulse walk. This can be implemented with a detection threshold that is varied based on the amplitude of each pulse. The variable detection threshold for pulse detection is calculated as a percentage of the maximum and minimum peak of each pulse and is used to detect the rising edge of the pulse in the same relative position between pulses. Figure 3 (b) shows the SNSPD IRFs when using a variable detection threshold. This figure shows that the IRF for the high photon flux rate (37.785 Mcps) is pulled back to align with the IRF for the low photon flux rate (1.162 Mcps).



(a) (b)

Figure 3. Measured SNSPD instrument response functions (IRF) using (a) fixed detection thresholds and (b) variable detection thresholds for different average count rates.

The FPGA uses a fixed threshold to find the approximate location of the maximum peak in the pulse, and 3 samples are used before and after the fixed threshold to locate the absolute maximum and local minimum of each peak. The pulse amplitude is calculated from the minimum and maximum peaks, and the variable detection threshold for photon counting is set to 50% of the pulse amplitude. A reset threshold is also used to prevent misdetections of pulses during the detector reset period.

Photon counting is performed at a rate of up to 200 Mcps per ADC channel, or at a total rate of 3.2 Gcps for all 16 ADCs on the RFSoC FPGA. The photons are counted and accumulated in slots. Each slot is saturated at 7 photon counts. The FPGA processing clock is shifted to 125 MHz for the rest of the design.

3.4 Slot Offset Calculation

The slot offset calculation accumulates the photon counts in respective slots over 512 symbols nominally. This vector is used to calculate the slot offset error.[7] This error is driven to values close to zero (-0.5 to +0.5 slots) when the loop is tracking. However, the calculation outputs a number in the range $[0\ ,\ 1.25M)$, where $M \in \{16, 32\}$ is the PPM order. Results close to $1.25M$ are unwrapped back to negative numbers close to zero.

3.5 Loop Filter

The loop filter consists of a proportional integral (PI) controller.[8] This filter is designed to track a slot clock frequency offset of up to +/- 66 parts per million and is sufficient to track the expected levels of Doppler rate of change (+/- 1,034 Hz/s for the 0.5 ns slot clock). The loop filter takes in the slot offset error signal of the slot offset calculation and outputs a filtered slot offset error signal to the distribute module.

3.6 Slot to Sample Distribution

The slot offset error calculation requires the use of many symbols and during the calculation time the error can accumulate into large step corrections. In cases with high Doppler, this slot offset error can be greater than a full slot. Correcting the slot offset directly with these large values will cause discontinuities in the data stream. The approach taken in this receiver is to break down the single large slot adjustment into multiple smaller adjustments and apply them to the ADC sample stream while the subsequent slot offset is being calculated. This is accomplished by dividing the total slot offset by powers of 2 until the smaller offsets are below an acceptable level of 3% of the sampling period. These smaller offsets are then applied to the stream in linearly spaced time increments over the slot offset calculation period. This module also converts from slots back to ADC samples for oversampled modes (>500 ps slot widths).

3.7 Sample Offset Integration and Wrapping

The small sample offset steps output from the distribute module are integrated to track the sample offset over time. The result of this integration is the sample offset at which to interpolate. As the timing recovery loop tracks the received signal, the timing offset is always changing, and this integration register will go to positive or negative infinity without wrapping. Wrapping takes place when the integration register is greater than +0.5 samples or less than -0.5 samples. If the sample offset is outside of this range, a sample will be added or subtracted from the integration register and the interpolator will output either one additional or one less sample as discussed in Section 3.2.

A separate sample offset integration register is used for each ADC. Each integration register is initialized to a different offset between +/- 1 sample (+/- 500 ps) to digitally compensate for differences in physical path lengths of the RF cables and optical fibers outside of the FPGA. This digital phase shifter eliminates the need for external RF phase shifters to align the detector channels prior to input into the FPGA.

3.8 Codeword Synchronization

The codeword synchronization module uses an approximation to a maximum likelihood correlator[9]. The hardware searches the entire codeword symbol-by-symbol to locate the most likely position of the codeword sync marker. Once located, the module continues to search subsequent codewords for the most likely position of the codeword sync marker. Lock is declared when 6 codeword sync markers in a row are detected in the same position. Once locked, the module removes the codeword sync marker and passes the codeword-aligned symbols to subsequent modules. The module also continues to search all codewords to locate the most likely position of the codeword sync marker. Unlock is declared if the module misses 6 codeword sync markers in a row.

3.9 Convolutional Deinterleaving

The convolutional deinterleaver[10] utilizes external double data rate 4th generation (DDR4) random access memory (RAM) to store the data. With a marginal increase in design complexity, the use of DDR4 results in significant savings of FPGA resources due to the large amount of data that needs to be buffered and reordered. Data is pre- and post-processed in the FPGA to both parallelize and serialize the data to efficiently use the full width of the DDR4 memory bus (512 bits). A parallelization factor of 9 was chosen for this implementation. For jitter compensation, the guard band slots adjacent to the symbol slots are retained.

4. ITERATIVE DECODING

The iterative decoder is implemented on a Virtex Ultrascale FPGA and performs SCPPM decoding[11] for the receive modem. Currently, the PPM-16, code rate 1/3, and PPM-32, code rates 1/3, 1/2, and 2/3 CCSDS HPE modes have been implemented. In the decoder front-end, a buffer stores the received channel statistics and a log-likelihood ratio (LLR) coefficient calculated from the mean signal photons per signal slot, K_s, and the mean background photons per slot, K_b. The LLRs for each symbol slot are calculated and queued into a decoder slice. Iterative Bahl, Cocke, Jelinek, and Raviv (BCJR) decoding[12] is performed with multiple decoder slices and the output codewords are reordered, derandomized, and desliced,

Transfer frame synchronization and data link deframing is performed and the decoded and decapsulated data is transmitted out of the receiver over Gb Ethernet, shown in Figure 4.

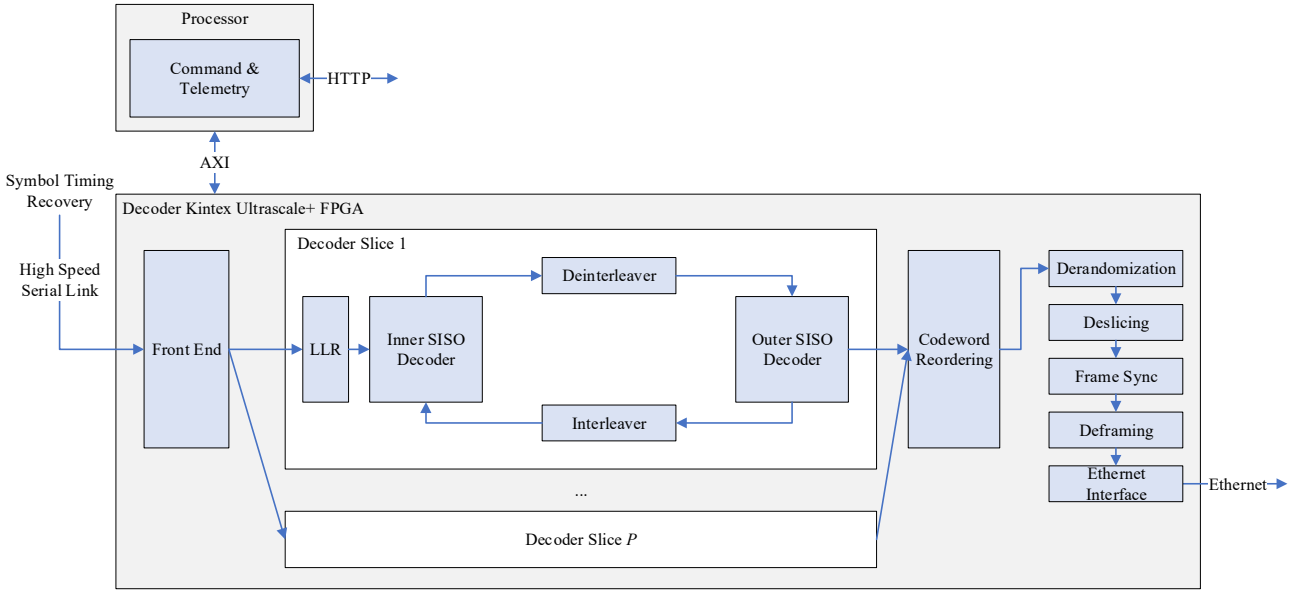


Figure 4. Decoder FPGA architecture with *P* iterative decoder slices in parallel.

The remainder of the PPM orders in the HPE standard can be implemented with modifications to the decoder logic for the needed resource optimizations.

4.1 Multiple Decoder Slices

Each decoder slice is comprised of an inner PPM soft-input soft-output (SISO) decoder, a block deinterleaver, an outer convolutional SISO decoder, and a feedback path through a block interleaver to provide a-priori LLR information to each subsequent iteration of the decoding process.[13] A cyclic redundancy check (CRC) is calculated for each decoded codeword at the output of the outer decoder to determine if the decoding was successful. If not, the codeword is fed back to the inner decoder with a-priori LLR information from the previous iteration.

A codeword may take up to 32 iterations to decode, so multiple decoder slices are used to match the data throughput of the received codewords. The number of decoder slices per HPE mode is determined by ensuring a simulation of the front end codeword buffer does not overflow for a 10^{-4} CWER. The front-end of the decoder assigns an 11-bit cumulative sequence number to each codeword. Once a codeword successfully decodes or the maximum iterations are reached, the front-end assigns a new codeword to that decoder slice. Codewords are assigned to decoders in an ascending sequence, with the first decoder given the highest priority for accepting new codewords.

If the CRC is invalid and 32 iterations are reached, the codeword is dropped and a codeword error counter is incremented. If the CRC is valid, the decoded codeword is stored in a specific RAM in the FPGA determined by the sequence number of the codeword. Once decoded codewords have been stored in RAMs, the RAMs are read in ascending order. If a codeword decodes out of order, the RAM will hold that codeword until the iteration cycles of the previous codeword are completed, and the previous codeword is stored in a RAM. If all the RAMs are full and the CRC for a codeword in the decoder is valid, the decoder will hold the codeword until the RAM is available. Backpressure from higher data rates and high iteration counts will cause codewords to overflow the decoder front-end buffer, and those dropped codewords are counted as codeword errors.

4.2 Decoder Reconfigurability

The decoder FPGA can be reconfigured to support each of the HPE modes; however, the decoder must be resynthesized for different PPM orders and code rates. The code rate and PPM order are set in a hardware description language (HDL)

package file along with additional reconfigurable parameters such as the number of RAMs used for codeword reordering and the number of decoder slices used for each HPE mode.

4.3 Pulse Timing Uncertainty Compensation

The pulse walk compensation in the timing recovery FPGA eliminates some of the pulse timing uncertainty; however, the detectors still introduce additional timing uncertainty due to the variance of the delay between the incident photons and the output voltage pulse as shown in Figure 3 (b). This pulse timing uncertainty, referred to as detector jitter, is represented as an offset in the arrival time of the photons. Detector jitter causes the arrival of signal photon counts in adjacent slots to the signal slot.

To reduce this effect of jitter, Srinivasan, Rogalin, Lay, Shaw, and Tkacenko demonstrate that the LLR of a signal slot k can be found and maximized from the channel statistics by calculating

$$LLR(k) = \sum_{i=k-D}^{k+D} u_i \ln\left(1 + f(i,k,\Delta_k,\sigma_i)\frac{\widehat{K}_s}{\widehat{K}_b}\right) + \mathrm{B}_1 \quad (1)$$

where B_1 is a constant cancelled out in decoder bit estimates, D is the number of adjacent slots on either side of slot index k to incorporate in the filter, u_i is the slot counts for symbol slot i, and $f(i,k,\Delta_k,\sigma_i)$ is a scaling constant that depends on the distribution of detector jitter, slot offset, and the signal slot.[14] The FPGA applies this detector jitter filter to the LLR calculation to maximize the LLR for each signal slot and reduce the performance degradation due to detector jitter.

4.4 Inner Decoder

The inner PPM decoder takes 4 inputs, the channel statistics as PPM symbols with 3-bit slot counts for each slot in the symbol and 3 log-likelihood factor terms defined as $\ln\left[1 + f(i,k,\Delta_k,\sigma_i)\frac{K_s}{K_b}\right]$ from (1). The inner decoder calculates the LLR using (1) from the channel statistics. The resulting LLRs are combined with a-priori LLR symbol estimates calculated from the a-posteriori bit estimates of the outer decoder and fed to the BCJR code trellis in forward and backward order.[15] The inner decoder processes the forward and backward paths in parallel. At the intersection of the forward and backward paths along the trellis, a new set of a-posteriori LLR symbol estimates are found and fed to the deinterleaver in parallel forward and backward order starting from the middle of the codeword.

4.5 Generalized Algorithmic Polynomial Interleaver

The CCSDS HPE standard uses a polynomial block interleaver to interleave the symbols of a 15120-bit codeword according to

$$I_j = h_{\pi(j)} \quad (2)$$

where j is the index of the symbol in the codeword, h is the codeword, I is the codeword with symbol bit LLRs interleaved, and $\pi(j) = (11j + 210j^2) \bmod 15120$.[3] The inner PPM decoder outputs symbol estimates starting from the middle of the codeword in the forward and backward interleaved order defined by (2). Deinterleaving is performed by writing the bytes of I to a deinterleaver RAM sequentially followed by reading the bytes of h into the outer convolutional decoder by calculating the RAM addresses using the inverse polynomial $\pi^{-1}(j) = (7331j + 7770j^2) \bmod 15120$.

To simplify the implementation at the output of the outer decoder, the bit estimates are re-interleaved by writing the bytes of h into RAM with each position defined by $\pi^{-1}(j)$ and reading the bytes of I out of the RAM sequentially.

The decoder calculates the inverse interleaved order in real time using the generalized polynomial $f(j) = aj + bj^2$ where $a = 7331$ and $b = 7770$. Cheng, Moision, Hamkins, and Nakashima show that the next position in the sequence can be calculated by expanding to the $f(j+1)$ case and recursively computing the next subsequent permutation from the previous with only addition.[16]

Each LLR in a symbol is processed in parallel throughout the decoder to take advantage of parallel trellis edges, so multiple addresses for the deinterleaver must be simultaneously generated. The $f(j+1)$ case is only sufficient to generate one address at each clock cycle, so a more general form $f(j+n)$ is found, where the value of $n \in \mathbb{Z}$ determines both the number of addresses and direction of the polynomial. For the backwards direction, the reverse order of the permutations can be found with $n < 0$. Recalling from the general form $f(j) = aj + bj^2$, we find that

$$f(j+n) = (aj + bj^2 + an + 2bjn + bn^2) \bmod N$$
$$= \big(f(j) + g_n(j)\big) \bmod N \quad (3)$$

where N is the number of codeword bits and

$$g_n(j) = (an + bn^2 + 2bjn) \bmod N. \quad (4)$$

The $g_n(j)$ term repeats after a set number of iterations on j for some values of n. An example of this behavior is shown in Table 1 for $n = \{1,2,3,4,5,6\}$ along with the corresponding address permutations $f(j)$.

Table 1. Address permutations $f(j)$ with $g_n(j)$ term for $n = \{1,2,3,4,5,6\}$ over 15120 iterations

j	$f(j)$	$g_1(j)$	$g_2(j)$	$g_3(j)$	$g_4(j)$	$g_5(j)$	$g_6(j)$
0	0	15101	382	1203	2444	4105	6186
1	15101	401	1222	2463	4124	6205	8706
2	382	821	2062	3723	5804	8305	11226
3	1203	1241	2902	4983	7484	10405	13746
4	2444	1661	3742	6243	9164	12505	1146
5	4105	2081	4582	7503	10844	14605	3666
6	6186	2501	5422	8763	12524	1585	6186
7	8687	2921	6262	10023	14204	3685	8706
…	…	…	…	…	…	…	…
15117	2577	13841	12982	12543	12524	12925	13746
15118	1298	14261	13822	13803	14204	15025	1146
15119	439	14681	14662	15063	764	2005	3666

For example, when $n = 6$, $g_n(j)$ repeats every 6 iterations of j. This is convenient for an FPGA implementation because the values can be precomputed and stored in memory.

A more general form for generating n $g_n(j)$ values in parallel is found by expanding (4) to the $g_n(j+n)$ case, where

$$g_n(j+n) = (an + bn^2 + 2bjn + 2bn^2) \bmod N$$
$$= (g_n(j) + 2bn^2) \bmod N. \quad (5)$$

Per clock cycle, 3 addresses are needed in parallel for a 1/3 and 2/3 code rate and 2 addresses are needed in parallel for a 1/2 code rate. Ideally, n would be set to the number of addresses used for each code bit estimate; however, these code rates share a common denominator of 6, so this implementation selects a value of $n = 6$ and generates 6 addresses in parallel. In addition, when $n = 6$, $2bn^2 \bmod N = 0$ which simplifies the arithmetic.

Parallel BRAMs are used for interleaver and deinterleaver memory. Both a forward and backward symbol estimate consisting of $m = \log_2 M$ LLRs each arrive simultaneously to the deinterleaver from the inner decoder. There are m parallel dual port BRAMs with $\frac{N}{m}$ entries each, with each entry consisting of a single LLR. Index pairs are mapped to the address permutations and are used to read the deinterleaver RAM in sequential order, as shown in Figure 5, and write to the interleaver RAM in interleaved order.

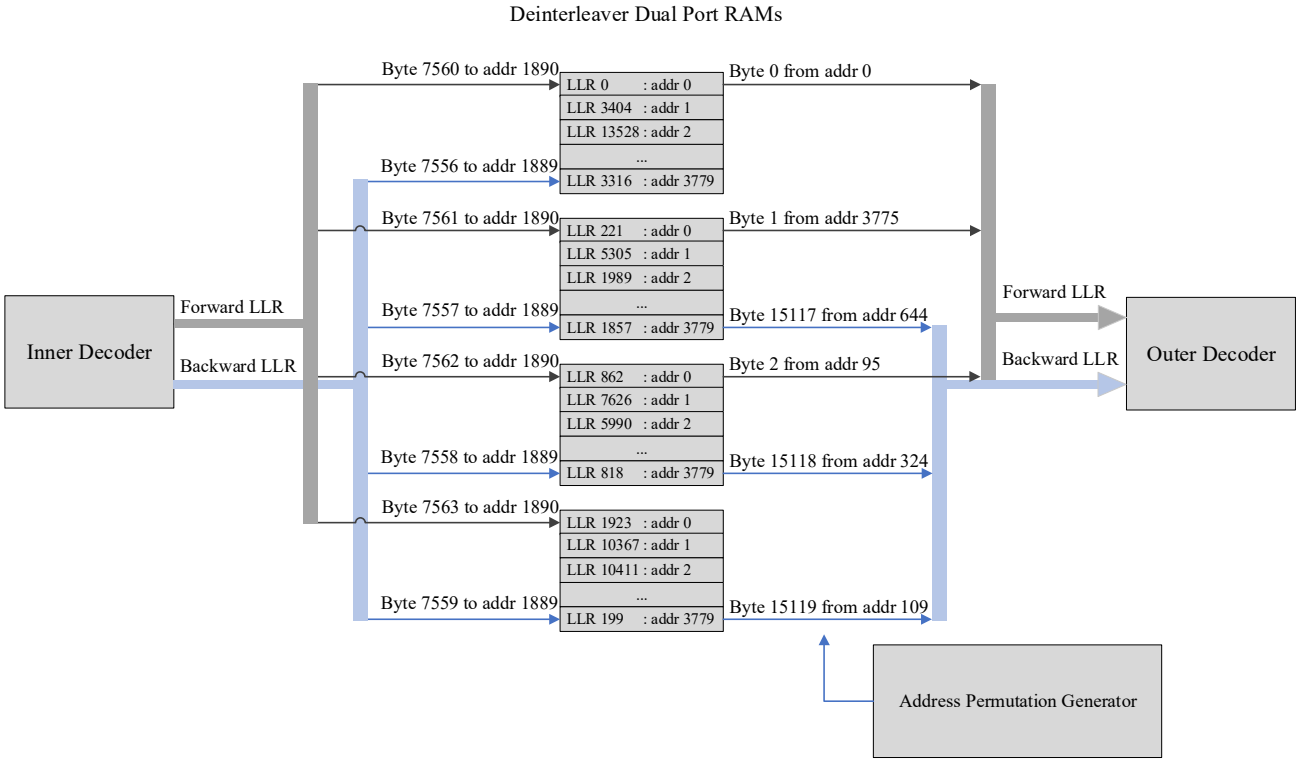


Figure 5. PPM-16 with a 1/3 Code Rate deinterleaver RAM. The symbols comprised of 4 LLR bytes each arrive to the deinterleaver in forward and backward order from the outer decoder. The symbols are written starting at the middle of the codeword to the dual port RAMs. The address permutations are used to read the bytes into the inner decoder 3 at a time, with each address determined by the inverse interleaver polynomial index pair.

Cheng, Moision, Hamkins, and Nakashima show that each interleaved position $f(j)$ is mapped to an index pair $(r_f(j), q_f(j))$, where $r_f(j) \triangleq f(j) \bmod m$ selects the BRAM partition and $q_f(j) \triangleq \lfloor \frac{f(j)}{m} \rfloor$ selects the address entry into the BRAM partition.[16] To find n BRAM partitions in parallel, initialize the first n values of $r_{g,n}(j) = g_n(j) \bmod \mathrm{m}$ and $r_f(j) = f(j) \bmod m$ and calculate

$$r_f(j+n) = [r_f(j) + r_{g,n}(j)] \bmod m. \tag{6}$$

To find the next n values of r_g, calculate

$$r_{g,n}(j+n) = [r_{g,n}(j) + r_{2bn^2}] \bmod m \tag{7}$$

where r_{2bn^2} is a pre-defined constant. To find n BRAM addresses in parallel, initialize the first n values of $q_{g,n}(j) = \left\lfloor \frac{g_n(j)}{m} \right\rfloor$ and $q_f(j) = \left\lfloor \frac{f(j)}{m} \right\rfloor$ and let $g_n(j) \triangleq q_{g,n}(j)m + r_{g,n}(j)$ and $f(j) \triangleq q_f(j)m + r_f(j)$. Calculate

$$\begin{aligned} q_f(j+n) &= \left\lfloor \frac{f(j+n)}{m} \right\rfloor \\ &= q_f(j) + q_{g,n}(j) + \left\lfloor \frac{r_j(j) + r_{g,n}(j)}{m} \right\rfloor - Ind * \frac{N}{m} \end{aligned} \tag{8}$$

where $Ind = 1$ when $q_f(j) + q_{g,n}(j) + \left\lfloor \frac{r_j(j)+r_{g,n}(j)}{m} \right\rfloor \geq \frac{N}{m}$, otherwise $Ind = 0$. To find the next n values of $q_{g,n}$, calculate

$$q_{g,n}(j+n) = q_{g,n} + q_{2bn^2} + \left\lfloor \frac{r_{g,n} + r_{2bn^2}}{m} \right\rfloor - Ind * \frac{N}{m} \tag{9}$$

where $2bn^2 \bmod N = q_{2bn^2}m + r_{2bn^2}$ and $Ind = 1$ when $q_{g,n} + q_{2bn^2} + \left\lfloor \frac{r_{g,n}+r_{2bn^2}}{m} \right\rfloor \geq \frac{N}{m}$, otherwise $Ind = 0$. For the $n = 6$ case, $2bn^2 \bmod N = 0$. A breakdown of the partition and address selections for each PPM-16 symbol is shown in Table 2.

Table 2. PPM-16, 1/3 Code Rate RAM partitions and addresses for forward interleaved order with $n = 6$

Cycle	j	$f(j)$	$g_6(j)$	$r_f(j)$	$q_f(j)$	$r_{g,6}(j)$	$q_{g,6}(j)$
0	0	0	6186	0	0	2	1546
0	1	15101	8706	1	3775	2	2176
0	2	382	11226	2	95	2	2806
0	3	1203	13746	3	300	2	3436
0	4	2444	1146	0	611	2	286
0	5	4105	3666	1	1026	2	916
2	6	6186	6186	2	1546	2	1546
2	7	8687	8706	3	2171	2	2176
…	…	…	…	…	…	…	…
5038	15117	2577	13746	1	644	2	3436
5038	15118	1298	1146	2	324	2	286
5038	15119	439	3666	3	109	2	916

With $n = 6$ in the FPGA implementation, the interleaver and deinterleaver positions are found with only addition elements and the BRAMs needed to store the codewords.

4.6 Outer Decoder

The outer decoder works in a similar manner to the inner decoder in that it accepts a-priori LLRs and traverses a convolutional code trellis in forward and backward order according to the BCJR algorithm to produce a-posteriori LLR bit estimates. The LLRs are converted back to uncoded a-priori symbol estimates by calculating symbols from the interleaved order of the bit estimates.[15] The bit estimates are used to generate the decoded bit stream on the output of the outer decoder.

4.7 Derandomizer and Deslicer

The decoded bits are derandomized according to the pseudo-randomizer sequence defined in the CCSDS HPE standard. The codeword padding is removed from the derandomized sequence and then desliced according to the CCSDS HPE standard.

4.8 Transfer Frame Synchronization

The transfer frame synchronization marker (TFSM) 0x1ACFFC1D is used to identify the boundaries of the transfer frames as defined in the CCSDS HPE standard. The length of each transfer frame is fixed with a TFSM in between each frame. The number of required correct TFSMs in a row before transfer frame synchronization is achieved is reconfigurable in the FPGA. The lock detector may similarly miss a reconfigurable amount of TFSMs before losing transfer frame synchronization.

5. IMPLEMENTATION RESULTS

5.1 Timing Recovery FPGA Utilization

The resource utilization for all supported timing recovery FPGA modes is given in Table 3.

Table 3. Zynq Ultrascale+ XCZU29DR timing recovery resource utilization

Resource	Utilization	Available	Utilization %
BRAM	584	1080	54.07
CLB LUT	158848	425280	37.35
CLB Register	169892	850560	19.97
DSP	288	4272	6.74

5.2 Iterative Decoder FPGA Utilization

The resource utilization for all supported decoder FPGA modes with a 500 ps slot width is given in Table 4. In general, the higher the data rate, the more resources are required to implement all the decoder slices in parallel. However, since the decoder processes codewords a symbol-at-a-time, the logic required for processing increases with larger symbols.

Table 4. Virtex Ultrascale XCVU440 decoder resource utilization per HPE mode

Mode	Decoder Slices	Resource	Utilization	Available	Utilization %
PPM-32, Code 1/3	13	BRAM	679	2520	26.94
		CLB LUT	694848	2532960	27.43
		CLB Register	668198	5065920	13.19
PPM-16, Code 1/3	16	BRAM	664.5	2520	26.37
		CLB LUT	470546	2532960	18.58
		CLB Register	464449	5065920	9.17
PPM-16, Code 1/2	18	BRAM	765.5	2520	30.38
		CLB LUT	528374	2532960	20.86
		CLB Register	516666	5065920	10.20
PPM-16, Code 2/3	20	BRAM	914.5	2520	36.29
		CLB LUT	603938	2532960	23.84
		CLB Register	580165	5065920	11.45

5.3 Jitter Compensation Performance

The relative performance improvements of both a variable detection threshold and detector jitter filtering for PPM-16 with a 1/3 code rate are shown in Figure 6. The tests were performed with a test optical transmitter[17] to emulate the received signal and compared against a software simulated baseline curve. The curve for no jitter compensation and a fixed detection threshold approaches a 10^{-4} CWER just below -15 dB photons per slot. Using a fixed threshold with the detector jitter filter improves the signal by ~0.75 dB photons per slot. A variable detection threshold without the detector jitter filter improves the signal by ~1.0 dB signal photons per slot. Together, a variable detection threshold and a detector jitter filter compensation improve the performance by ~1.25 dB photons per slot for this mode.

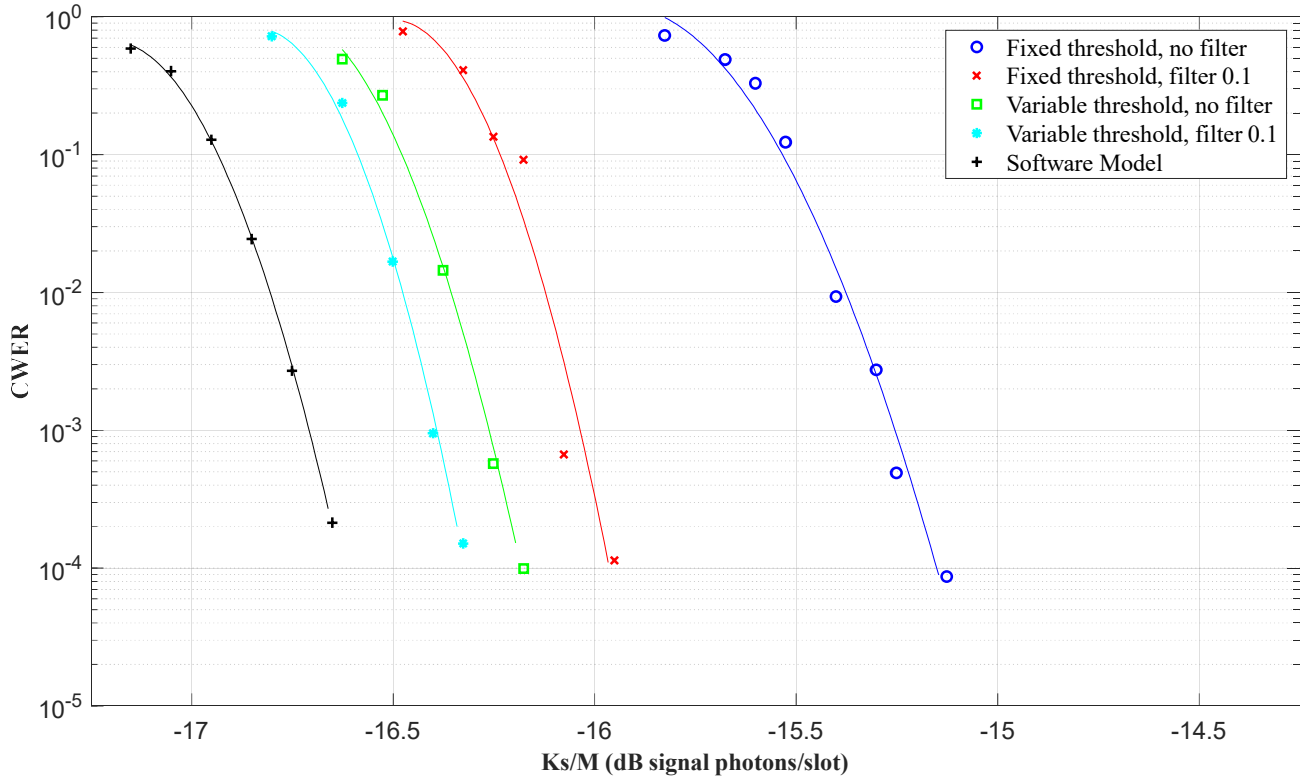


Figure 6. Receiver performance with detector jitter compensation for PPM-16 at a 1/3 code rate.

The largest improvements in performance are found using both a variable detection threshold in the timing recovery FPGA and the detector jitter filter in the decoder FPGA; however, each method independently shows a large improvement in jitter mitigation and should be considered separately. The detector jitter filter is a more complex implementation due to its dependency on channel statistics and detector characteristics. The variable detection threshold only affects the photon counting, so in the presence of pulse-walk this is easier to implement for similar performance improvements. Smaller PPM orders (16 and below) and narrower slot widths should see more improvements due to the detector jitter filter and using a variable detection threshold because of the ratio of the slot width to the detector jitter and the ratio of the PPM order to the detector reset time.

6. CONCLUSION

A single photon-counting receive modem compatible with PPM orders 16 and 32 and code rates 1/3, 1/2, and 2/3 from the CCSDS HPE standard was designed and built with COTS FPGA components. The modem is separated into two primary FPGAs, namely the timing recovery FPGA and the iterative decoder FPGA, which are connected via a high-speed serial interconnection. The modem includes detector jitter filtering, parallel processing, and resource optimizations to meet the maximum data rate of 267 Mbps while remaining below a 10^{-4} CWER. The modularity of the design combined with the robust performance on a COTS platform allows for rapid infusion for both non-commercial and commercial optical ground receivers. In the future, NASA GRC plans to implement additional HPE modes.

ACKNOWLEDGEMENTS

The authors of this paper would like to acknowledge Yousef Chahine and Sarah Tedder for their contributions to the receiver development. This work was funded by the NASA Space Communications and Navigation program.

REFERENCES

[1] Seas A., Robinson, B., Shih, T., Khatri, F., and Brumfield, M. "Optical communications systems for NASA's human space flight missions", Proc. SPIE International Conference on Space Optics 11180 (2019).

[2] Biswas, A., Srinivasan, M., Piazzolla, S., and Hoppe, D. "Deep Space Optical Communications," Proc. SPIE Free-Space Laser Communication and Atmospheric Propagation XXX 10524 (2018).

[3] "Optical communications coding and synchronization recommended standard," CCSDS 142.0-B-1 Blue Book (2019).

[4] "Space Telecommunications Radio Systems (STRS) Architecture Standard," NASA-STD-4009 (2018).

[5] "Synchronization of Signal Processing in Multiple RF Data Converter Subsystems" Xilinx, Inc. (XAPP1349 (v1.0)), (17 February 2022). [Online]. Available: https://docs.xilinx.com/r/en-US/xapp1349-rfdc-subsystems

[6] D. A. Gedcke and W. J. McDonald, "A constant fraction pulse height trigger for optimum time resolution," Nucl. Instrum. Methods, vol. 55, pp. 377–380, 1967.

[7] Rogalin, R. and Srinivasan, M. "Synchronization for optical PPM with inter-symbol guard times," The Interplanetary Network Progress Report 42-209 (2017).

[8] F. Gardner. "Phaselock Techniques," 2005, pp 70-71.

[9] Hamkins, J. "High Photon Efficiency (HPE) Recommendation Status," Consultative Committee for Space Data Systems Fall Meeting Presentation (2016).

[10] Downey, J., Shalkhauser, M., Bizon, T. "Parallelized convolutional interleaver implementation for efficient DDR memory access," SPIE Free-Space Laser Communications XXXIV, 11993 (2022).

[11] Moision, Bruce E. and Jon Hamkins. "Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation." (2005).

[12] Bahl, L., Cocke, J., Jelinek, F., Raviv, J. "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," IEEE Transactions on Information Theory, IT-20 (1974).

[13] Jon Hamkins, "How to decode SCPPM," Jet Propulsion Laboratory, Caltech (8 May 2016).

[14] Meera Srinivasan, Ryan Rogalin, Norman Lay, Matthew Shaw, Andre Tkacenko, "Downlink receiver algorithms for deep space optical communications," Proc. SPIE 10096, Free-Space Laser Communication and Atmospheric Propagation XXIX, 100960A (24 February 2017).

[15] Moision, B., Hamkins, J., Barsoum, M., Cheng, M., & Nakashima, M. (2009). "Hardware Implementation of Serially Concatenated PPM Decoder" (No. NPO-42246).

[16] M. K. Cheng, B. E. Moision, J. Hamkins and M. A. Nakashima, "An interleaver implementation for the serially concatenated pulse-position modulation decoder," 2006 IEEE International Symposium on Circuits and Systems, Kos, Greece, 2006, pp. 4 pp.-, doi: 10.1109/ISCAS.2006.1693566.

[17] Jennifer N. Downey, Sarah A. Tedder, Brian E. Vyhnalek, Nicholas C. Lantz, Michael A. Marsden, William P. Simon, Thomas P. Bizon, Daniel J. Zeleznikar, "A real-time optical ground receiver for photon starved environments," Proc. SPIE 12413, Free-Space Laser Communications XXXV, 124130P (15 March 2023).

APPENDIX A: POLYNOMIAL INTERLEAVER DERIVATIONS

This appendix includes a series of derivations for the modulus and quotient calculations of $f(j)$. The recursive relation for the remainder of $f(j)$ with a step size n for $m = \log_2(M)$ is given as

$$\begin{aligned} r_f(j+n) &= f(j+n) \bmod m \\ &= \left[\left(f(j) + g(j)\right) \bmod N\right] \bmod m \\ &= [f(j) + g(j,n)] \bmod m \\ &= f(j) \bmod m + g(j,n) \bmod m \\ &= [r_f(j) + r_g(j,n)] \bmod m \end{aligned} \tag{10}$$

where $r_{g,n}(j) = g_n(j) \bmod m$ and $r_f(j) = f(j) \bmod m$. The derivation of the recursive relation for the remainder of $g_n(j)$ is given as

$$\begin{aligned} r_{g,n}(j+n) &= g_n(j+n) \bmod m \\ &= [g_n(j) + 2bn^2] \bmod m \\ &= [g_n(j) \bmod m + 2bn^2 \bmod m] \bmod m \\ &= [r_{g,n}(j) + r_{2bn^2}] \bmod m \end{aligned} \tag{11}$$

where r_{2bn^2} is a pre-defined constant. Let $g_n(j) \triangleq q_{g,n}(j)m + r_{g,n}(j)$. The derivation of the recursive relation for the quotient of $g_n(j)$ is given as

$$\begin{aligned} q_{g,n}(j+n) &= \left\lfloor \frac{g_n(j+n)}{m} \right\rfloor \\ &= \left\lfloor \frac{(g_n(j) + 2bn^2) \bmod N}{m} \right\rfloor \\ &= \left\lfloor \frac{g_n(j) \bmod N + 2\mathrm{bn}^2 \bmod N}{\mathrm{m}} \right\rfloor \\ &= \left\lfloor \frac{\left(\left(q_{g,n}(j) + q_{2bn^2}\right)m + \left(r_{g,n} + r_{2bn^2}\right)\right) \bmod N}{m} \right\rfloor \\ &= q_{g,n} + q_{2bn^2} + \left\lfloor \frac{r_{g,n} + r_{2bn^2}}{m} \right\rfloor - Ind * n \end{aligned} \tag{12}$$

where $2bn^2 \bmod N = q_{2bn^2}m + r_{2bn^2}$ and $Ind = 1$ when $q_{g,n} + q_{2bn^2} + \left\lfloor \frac{r_{g,n}+r_{2bn^2}}{m} \right\rfloor \geq \frac{N}{m}$, otherwise $Ind = 0$.