HIGH-THROUGHPUT HARDWARE DECODER IMPLEMENTATION FOR OPTICAL DEEP-SPACE COMMUNICATIONS

C. Roth\textsuperscript{1}, D. Stadelmann\textsuperscript{1}, F. Arnold\textsuperscript{2}, C. Benkeser\textsuperscript{2}, and Q. Huang\textsuperscript{1}

\textsuperscript{1}Dept. of Information Technology and Electrical Engineering, ETH Zurich, 8092, Zurich, Switzerland
\textsuperscript{2}RUAG Space, RUAG Schweiz AG, Schaffhauserstrasse 580, 8052 Zurich, Switzerland
\textit{e-mail}: \{rothc, huang\}@iis.ee.ethz.ch
\textit{e-mail}: \{felix.arnold, christian.benkeser\}@ruag.com

I. INTRODUCTION

Future space missions will have to meet the demand for very high data rates. Transmission of scientific data and other high-bandwidth communications from deep space back to Earth are among the key challenges coming along with next-generation missions to the Moon and beyond. Optical communication has gained significant interest recently as a promising candidate to offer these high data rates. Compared to today’s deep-space communication systems that operate in the radio frequency (RF) spectrum, the small diffraction losses of an optical link yield significantly higher concentration of received signal energy. This improved signal level enables corresponding space equipment to operate at lower transmit power and, at the same time, to achieve higher data rates. Furthermore, optical transmit modules are much smaller in size than state-of-the-art RF transmit antennas, which entails a potential reduction in both dimension and weight of space equipment.

In order to prove the feasibility of such an optical link, the National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA) have been collaborating on the Lunar Laser Communications Demonstration (LLCD) project \cite{1} with the goal to demonstrate an optical communication link from NASA’s LADEE lunar-orbit spacecraft to Earth. This link has been specified to offer up to 16 multiplexed sub-channels, where each individual subchannel provides a data rate of 38.55Mbps, resulting in a maximum aggregate data rate of roughly 620Mbps. In order to achieve high link reliability, the proposed link is based on NASA’s capacity-approaching modulation and coding scheme, which comprises a serial concatenation of an inner accumulate pulse-position modulation (PPM) and an outer convolutional code (SCPPM) \cite{2}. Among several existing approaches to decode the specified SCPPM code, employing the turbo principle has proved to yield best results in terms of error-rate performance \cite{2}. Unfortunately, turbo decoding entails significant computational complexity due to the fact that costly soft-input soft-output (SISO) maximum a-posteriori decoders for the inner and outer code are required, exchanging soft information about the transmitted code bits in an iterative fashion. In contrast to the prominent class of binary parallel concatenated turbo codes well-established in cellular communication systems, the use of higher-order PPM further exacerbates the implementation. In fact, the high implementation costs of turbo decoding in conjunction with the high throughput requirements render the SCPPM turbo-decoder a key challenge in the receiver design for the LLCD project and future space missions.

A. Contributions and Outline

In this paper, we present a novel SCPPM turbo-decoder hardware architecture optimized for field-programmable gate-array (FPGA) devices and fully compliant with the coding scheme specified by the LLCD project. In order to respond to the challenge of achieving high throughput, we employ a highly parallel SISO-decoding scheme and combine it with subblock-parallel turbo decoding, which refers to the deployment of multiple SISO decoders that jointly decode a code frame \cite{3}. Compared to parallelization on turbo-decoder level, the advantage of a subblock-parallel architecture is increased hardware efficiency and lower decoding latency per code frame \cite{3}.

While subblock-parallel turbo decoding has become the basis for high-throughput turbo-decoder implementations for cellular systems (see, e.g., \cite{4}), it has only partly been considered in SCPPM turbo-decoder implementations. For instance, the prominent reference FPGA-implementation presented in \cite{5} deploys several outer decoders, as a means of balancing the different rates of inner and outer code, while using only one inner decoder. We propose to extend this approach to the use of multiple inner decoders in order to allow for higher data rates per encoder instance. To this end, we present an efficient memory architecture that supports the high memory-bandwidth required for subblock-parallel SCPPM turbo-decoding by exploiting two key properties of the specified quadratic-polynomial permutation (QPP) interleaver. The corresponding FPGA-implementation results demonstrate that our decoder architecture enables significant throughput gains compared to prior-art implementations.

The remainder of this paper is organized as follows. In Sec. II we briefly review the principle of SCPPM turbo decoding as well as best practice for SISO decoding. The subblock-parallel turbo-decoder architecture is introduced in Sec. III, and Sec. IV details the proposed memory architecture. The corresponding FPGA-implementation results are presented in Sec. V. We finally conclude in Sec. VI.
II. ITERATIVE SCPPM TURBO-DECODING

Consider the optical communication system shown on top in Fig. 1. The SCPPM encoder first encodes a sequence of information bits $u_k$ ($k = 0, \ldots, K - 1$) into a sequence of coded bits $x_n$ of length $N = 2^K$ using an outer 4-state rate-1/2 convolutional encoder. We refer to $K$ as the frame length of the code. The coded sequence is permuted in pseudo-random order by a quadratic-polynomial permutation (QPP) interleaver that maps coded bits $x_n$ into permuted coded bits $x_{\pi(n)}$ with

$$\pi(n) = (11n + 210n^2) \mod N.$$  

The interleaved coded bits are then passed through the inner encoder, consisting of an accumulator and an $M$-ary PPM mapper, to produce $M$-ary PPM symbols $c_t$ ($t = 0, \ldots, T - 1$) with $T = N / \log_2 M$. Note that each PPM symbol is associated with $Q = \log_2 M$ interleaved coded bits. In each PPM-symbol interval, a laser pulse is conveyed over the optical channel in one of the $M$ possible time slots depending on the corresponding information to be transmitted. At the receiver, the optical signal is sensed by a photo detector that delivers for each time slot $m$ ($m = 0, \ldots, M - 1$) of PPM symbol $c_t$ the number of detected photons $k_{t,m}$. Based on these photon counts, a soft-output detector computes the vector of so-called slot values $L_t^m = [L_t^0, L_t^1, \ldots, L_t^{M-1}]$, where $L_t^m$ represents the soft-information about the corresponding time slot in the form of a log-likelihood ratio (LLR) value, expressing the reliability of being either a signaling slot or a nonsignaling slot. Assuming the Poisson channel model proposed in [5], the slot values $L_t^m$ are computed according to

$$L_t^m = \frac{(n_s + n_b)k_{t,m} e^{-(n_s+n_b)}}{n_p m^{n_s+n_b}},$$

where $n_b$ and $n_s + n_b$ are the average photon counts per nonsignaling and signaling slot, respectively.

A. Turbo Principle

We employ the turbo principle [6] to decode the SCPPM code as proposed in [2]. The turbo principle applied to the considered system is illustrated on the upper right in Fig. 1 and essentially amounts to iteratively exchang-
ing extrinsic LLR values $L_{n}^{E_{1}}E_{0}$ about the coded bits $x_n$ between an inner and an outer SISO decoder. Each iteration comprises a run of the inner decoder followed by a pass of the outer one. During interleaved half-iterations the inner decoder decodes the inner code and generates intrinsic a-posteriori LLC values $L_{n}^{D_{1}}$, based on the received slot values $L_{n}^{r}$ and on $L_{n}^{D_{1}}$. The latter denotes the a-priori information for the coded bits $x_n$ and is initialized with 0 prior to decoding. Based on the intrinsic information, the extrinsic LLC values $L_{n}^{E_{1}} = L_{n}^{D_{1}} - L_{n}^{D_{0}}$ are computed, serving as a-priori information for the outer decoder after being de-interleaved. Correspondingly, the outer decoder decodes the outer code during non-interleaved half-iterations to produce $L_{n}^{D_{0}}$ and $L_{n}^{E_{0}}$ based on $L_{n}^{D_{0}}$. In addition to these LLC values, the outer decoder also computes a-posteriori estimates $\tilde{u}_n$ for each transmitted information bit. With every iteration, the frame error-rate (FER) $Pr(\tilde{u} \neq u)$ of the decoded code frame improves until it starts saturating after a specific number of iterations.

B. Trellis Representation of Outer and Inner Codes

It is worth mentioning that both outer and inner code can be represented by a trellis diagram consisting of states and branches as illustrated in the middle of Fig. 1. The outer binary convolutional code exhibits four states with two branches per state. The inner accumulate $M$-ary PPM code, on the other hand, has two states with $M$ branches per state. Note that the outer trellis comprises $K$ trellis steps, according to the $K$ input information bits, while the inner trellis has $T$ steps corresponding to the $T$ transmitted PPM symbols. Each state $s$ in the outer trellis is associated with so-called state metrics $A_k(s)$ and $B_k(s)$. Branches are associated with branch metrics $\Gamma_k(s',s)$, where $s'$ and $s$ correspond to connected states at trellis steps $k - 1$ and $k$, respectively. Similarly, each state in the inner trellis is associated with state metrics $A_k(s)$ and $B_k(s)$, and branch metrics $\Gamma_k^i(s',s)$ ($l = 0, \ldots, M/2 - 1$) are assigned to each of the $M/2$ parallel branches.

C. The Max-log BCJR Algorithm

Both inner and outer decoder apply the log-domain BCJR algorithm [7], [8], which amounts to traversing the respective trellises in forward and backward directions to recursively compute the state metrics $A$ and $B$. To avoid disadvantage, the ideal log-domain BCJR algorithm relies on transcendental functions (i.e., log sums of exponentials), which lead to long signal delays and require a significant amount of resources when implemented in FPGA devices as shown in [5]. Therefore, we use the hardware-friendly max-log approximation [8] and mitigate the resulting FER-performance loss by employing extrinsic-LLR scaling [9]. In what follows, we briefly discuss the basics of the max-log-BCJR algorithm. For a more thorough treatment, refer to, e.g., [8], [5].

Applying the max-log approximation, the outer decoder recursively computes the forward state-metrics $A_k(s)$ in forward trellis direction using simple add-compare-select (ACS) operations according to

$$A_k(s) = \max\{A_{k-1}(s_0) + \Gamma_k(s_0,s), A_{k-1}(s'_1) + \Gamma_k(s'_1,s)\},$$

(3)

where $s'_0$ and $s'_1$ correspond to the two predecessor states of state $s$ (see middle-left part in Fig. 1). The state metrics $B_k(s)$ are computed similarly in backward direction.

In the inner decoder, the computation of the state metrics is more involved due to the larger number of branches per state. As proposed in [5], we simplify the calculation by first identifying for each state transition $s' \rightarrow s$ in the inner trellis at step $t$ the maximum value of the corresponding parallel branch metrics, referred to as super branch-metric $\Gamma^s_t(s',s)$, and then perform the ACS operation based on these maximum values as depicted in the middle-right part in Fig. 1. In case of the forward state-metrics, the simplification results in

$$A_t(s) = \max\{A_{t-1}(s'_0) + \Gamma_t^s(s'_0,s), A_{t-1}(s'_1) + \Gamma_t^s(s'_1,s)\},$$

(4)

In order to compute the output LLR values, the two SISO decoders follow the butterfly scheme [10], which refers to the concurrent computation of forward and backward state-metrics. Specifically, both inner and outer decoder perform the following two phases as illustrated on the bottom in Fig. 1. 1) during the first phase, the forward state-metrics $A(s)$ (as in Eq. 3 and Eq. 4) and simultaneously the backward state-metrics $B(s)$ are computed and buffered, starting from the beginning and end of the trellis, respectively. As soon as the two computations meet in the middle of the trellis, the second phase starts. 2) during the second phase, a-posteriori and extrinsic LLC values are computed in forward direction (based on concurrently computed forward state-metrics and buffered backward state-metrics) and simultaneously in backward direction (based on concurrently computed backward state-metrics and buffered forward state-metrics). Compared to a simpler scheme where the two recursions are computed strictly sequentially (as implemented in, e.g., [5]), the choice of the described butterfly

---

1 We consider the accumulation and PPM modulation jointly as a single $M$-ary code. The two operations could also be separated, which would result in a simpler trellis, but also decreased error-rate performance [2].
scheme allows us to minimize the decoding time per SISO decoder at the expense of additional hardware resources required to perform the two recursions concurrently.

We emphasize that, due to the different rates of inner and outer code, the two SISO decoders generate a different number of LLR values per trellis step. In particular, the outer decoder generates in each trellis step $k$ (simultaneously in forward and backward direction) the 2 successive LLR values $L_{2k}$ and $L_{2k+1}$. The inner decoder, on the other hand, computes per trellis step $t$ the $Q$ successive LLR values $L_{2t+q}$ ($q = 0, \ldots, Q - 1$) according to the number of coded bits per PPM symbol. Hence, the inner decoder generates a-posteriori and extrinsic LLR values at $r = Q/2$ times higher rate than the outer one.

III. SUBBLOCK-PARALLEL SCPPM TURBO-DECODER ARCHITECTURE

We next present our SCPPM turbo-decoder architecture. As we strive for high throughput, we propose to employ subblock-parallel turbo decoding [3] as a means of reducing the decoding time per half-iteration. To this end, we divide the inner trellis into $P$ subblocks of equal length and instantiate $P$ inner decoders that concurrently decode the assigned subblocks during the interleaved half-iterations. For practical reasons, we must choose $P$ such that it divides the length of the inner trellis $T$ with zero remainder. Similarly, we divide the outer trellis into $rP$ subblocks and instantiate $rP$ outer SISO decoders to decode the outer code during non-interleaved half-iterations. Note that with the particular choice of $rP$ outer SISO decoders, we account for the different rate of inner and outer code and achieve that the decoding time for the two codes is identical. Compared to a conventional architecture with $P = 1$, the proposed architecture enables a $P$-fold gain in terms of throughput.

In the remainder of this paper, we assume the PPM order $M = 16$ and frame length $K = 7560$ as specified by the LLCD project. Consequently, there are $N = 15120$ extrinsic LLR values, $Q = 4$ extrinsic LLRs are associated with each PPM symbol, and $r = 2$ outer decoders are instantiated per inner decoder. We emphasize, however, that the presented architecture can easily be adapted to other PPM orders and frame lengths as well.

A. High-level Architecture

Fig. 2 shows the high-level block diagram of the proposed architecture. It consists of $P$ inner and $2P$ outer decoders based on the max-log BCJR algorithm, an input buffer for the storage of the received slot values $L_k$, an intermediate memory to store the extrinsic LLR values, and an output buffer for the computed estimates of the transmitted information bits $\hat{b}_k$. The interleaver units control the read and write accesses of the inner decoders to the intermediate memory during interleaved half-iterations (see Sec. IV for details).

Each SISO decoder processes one trellis step per clock cycle in both forward and backward direction according to the butterfly scheme discussed in Sec. II. Specifically, during interleaved half-iterations and in the first phase of the butterfly scheme, each inner decoder reads $2Q = 8$ extrinsic LLRs from the intermediate memory per clock cycle, 4 associated with the forward and 4 with the backward recursion, respectively (see Fig. 2).
Similarly, during non-interleaved half-iterations, each outer decoder requires 4 LLRs per clock cycle. Both decoder types buffer the LLRs read during the first phase internally to have them available also for the computations performed during the second phase. Recall that no output LLRs are computed in this first phase. During the second phase, each inner (outer) decoder writes \(4 \times 4\) computed extrinsic LLRs to the intermediate memory per clock cycle, 4 (2) in forward and 4 (2) in backward direction as shown in Fig. 2. Simultaneously, each outer decoder updates the output buffer with two bits per clock cycle.

B. Concurrent Decoding of Two Code Frames

We exploit the fact that the inner and outer codes are decoded during two separate half-iterations to let the two sets of inner and outer decoders work on two different code frames concurrently. This approach doubles the throughput of our turbo decoder at the sole overhead of doubled storage capacity of input buffer, output buffer as well as intermediate memory required to store data of two code frames (which is a small overhead compared to duplicating the entire turbo decoder).

IV. EFFICIENT CONTENTION-FREE MEMORY ARCHITECTURE

Enabling contention-free access of the inner and outer decoders to the intermediate memory is highly challenging due to the high memory-bandwidth coming along with subblock-parallel decoding. We next propose a generic architecture for the intermediate memory that enables contention-free memory access by exploiting two key properties of the specified QPP interleaver.

In the ensuing discussion, we only describe the memory access for the extrinsic LLRs processed in forward trellis direction, i.e., the signals corresponding to the solid lines in Fig. 2. The memory access for the LLRs processed in backward direction (dashed lines) is handled identically and simultaneously by exploiting the fact that modern FPGA devices typically feature dual-port BRAMs (short for block RAM) that provide two independent read/write ports (which we use for forward and backward signals, respectively).

A. Memory Organization

Fig. 3 illustrates the proposed architecture assuming as an example \(P = 4\) inner decoders and consequently 8 outer ones. The \(N\) extrinsic LLR values are distributed among \(Q = 4\) BRAMs according to the modulo-\(Q\) remainder of their non-interleaved indices. Furthermore, the individual BRAMs are organized in \(P\)-folded fashion [4], where \(P\) LLR values with constant index-offset \(N/P = 3780\) are stored per row (i.e., per memory address) in increasing order (see Fig. 3). Hence, each BRAM contains \(N/PQ\) rows.

B. Access of Inner Decoders

The access of the inner decoders to the intermediate memory during interleaved half-iterations is illustrated on top in Fig. 3. One key property of QPP interleavers is that 4 successive non-interleaved indices have unique modulo-4 remainders after being interleaved, i.e., \(\pi(4n + l) \mod 4 = i(l),\) with \(l = 0, \ldots, 3, i = 0, \ldots, 3\) and \(i(l) \neq i(l')\) if \(l \neq l'\) (see [11] for more details). This property ensures that the 4 successive interleaved LLRs accessed per clock cycle by each inner decoder are stored in different BRAMs. For the particular interleaver in (1), it additionally holds that the modulo-4 remainder of a non-interleaved index remains unchanged after being interleaved, i.e., \(i(l) = l\). Hence, as shown in Fig. 3 for the first inner decoder, the \(q\)th of the 4 successive LLR values is stored in BRAM-\(q\). We emphasize that for other QPP interleavers that lack the aforementioned property, we could simply reroute the inputs and outputs of the inner decoders according to the mapping \(i(l)\).

We now focus on all \(P\) inner decoders and consider for each decoder the \(q\)th of the 4 successive interleaved LLRs (i.e., the ones associated with modulo-4 remainder \(q\)). Another key property of QPP interleavers is that they are contention-free [11], which ensures that these considered \(P\) LLR values are stored in the same row in BRAM-\(q\) and, thus, can be accessed concurrently. However, a permutation of the \(P\) LLR values is required according to the specifics of the employed interleaver as depicted in Fig. 3.

It is shown in [11] that the property of contention-free interleaving is valid for all \(P\) dividing \(N\) with zero remainder. Furthermore, we can show that for the specified interleaver in (1), the uniqueness of the modulo remainder under interleaving holds for all PPM orders \(M = 2^Q\) up to \(Q = 10\).

C. Access of Outer Decoders

The memory access of the outer decoders during non-interleaved half-iterations is illustrated on the bottom in Fig. 3. The two successive LLRs accessed in forward direction by the odd-numbered \((2p - 1)\)th outer decoder \((p = 1, \ldots, P)\) are stored alternately in the \(p\)th column of BRAM-0 and BRAM-1 and the \(p\)th column of BRAM-2 and BRAM-3, respectively. Simple multiplexer are employed to switch between the two sets of BRAMs. At
the same time, the two LLRs accessed by the even-numbered $2p$th outer decoder are stored alternately in the $p$th column of BRAM-2 and BRAM-3 and the $p$th column of BRAM-0 and BRAM-1, respectively. We emphasize that, depending on the parameters $P$, $Q$, and $N$, it may be required to delay the operation of the even-numbered outer decoders by one clock cycle to make the described alternating access scheme possible.

V. FPGA IMPLEMENTATION RESULTS

We have implemented the proposed SCPPM turbo-decoder architecture for $M = 16$ and $K = 7560$ on a Xilinx Virtex-6 XC6VLX240T (speed grade -1) FPGA device. Our goal was to reach the LLCD-subchannel throughput of 38.55Mbps with a single turbo-decoder instance at 20 iterations, which is, according to our simulations, the point where the error-rate performance of the considered SCPPM code starts saturating. We achieve this goal by deploying $P = 4$ inner (and accordingly 8 outer) decoders. In the following discussion, we summarize the key characteristics of our $P = 4$ subblock-parallel SCPPM turbo-decoder and compare it to the prominent de-facto standard reference SCPPM turbo-decoder implementation reported in [5].

A. Error-rate Performance

In order to achieve near-to-optimum performance with our max-log-BCJR-based implementation, all input slot values and extrinsic LLR values are quantized to 6bit. We employ modulo-normalization inside the recursive ACS computations of forward and backward state-metrics, which allows us to limit the quantization of the state-metrics to 9bit and 10bit in the outer and inner decoders, respectively, without the overhead of costly normalization circuits [12]. Furthermore, we scale the extrinsic LLRs at the outputs of inner and outer decoders with a hardware-friendly constant of 0.75. In Fig. 4 we show the resulting FER performance of our implementation at 20 iterations and compare it to the one of the ideal decoding algorithm, i.e., the log-BCJR algorithm based on floating-point arithmetics. It can be observed that the overall implementation loss is smaller than 0.2dB. Hence, for the nominal deep-space mission scenario with an average background photon count $n_b = 0.2$, our decoder requires only $n_s = 1.9$ average detected photons per signaling slot to achieve a FER of $10^{-4}$.

B. Resource Utilization and Throughput

The computations inside the outer and inner decoders are pipelined (including the computation of the super branch-metrics in the inner decoders) to limit the longest path of the decoders to the delay of the recursive ACS computation. We further note that for our $P = 4$ subblock-parallel implementation, 16 permutation networks and, correspondingly, 16 interleaver units are required to realize the access of the inner decoders to the intermediate memory – 8 for the signals associated with the forward recursion (read and write) and 8 with the backward
recursion. Although QPP interleavers can be efficiently implemented in hardware using simple additions and modulo operations \[4\], we realize the interleaver units in form of look-up tables that are stored in BRAMs. This enables us to trade some of the vast BRAM resources available in modern FPGA devices for logic slices.

The considered target FPGA provides a total of 416 BRAM blocks and 37680 logic slices. In Tbl. I we show the resulting resource utilization after place-and-route in percentage for the outer and inner decoders as well as other miscellaneous components in our implementation, such as the input and output buffers, the intermediate memories, and the permutation networks. We find that our implementation occupies 46% of all logic slices and 21% of the available BRAM blocks. As expected, the 4 inner decoders require more logic resources than the 8 outer ones due to the additional overhead of the super branch-metrics computation. Interestingly, one can observe that most of the used BRAM resources are occupied by the miscellaneous components, i.e., the input and output buffers as well as the intermediate memories (that are required to provide enough capacity to store data for the two concurrently decoded frames).

The maximum clock frequency of the implementation after place-and-route is reported as 100MHz, which results in a decoding throughput of 39.5Mbps at 20 iterations – high enough to decode one LLCD subchannel with a single decoder. We emphasize, however, that in our current implementation, the clock frequency of the decoder is limited by the delay of the permutation networks between the inner decoders and the intermediate memory. Implementing these networks more efficiently (e.g., by pipelining) is part of on-going work and entails a potential speed-up of more than 50%.

Tbl. II provides a comparison of our design to the prominent implementation reported in \[5\], which is also based on the max-log-BCJR algorithm, but optimized for \(M = 64\) with 1 inner decoder and 3 outer ones performing the forward and backward recursions in strictly sequential fashion. It can be observed that, assuming a constant clock frequency for both implementations (as shown in the last row in Tbl. II), our design achieves roughly 10\(\times\) higher throughput. Interestingly, the throughput of our architecture scales with \(\log_2 M\) according to the number of coded bits associated with each PPM symbol. Hence, if we would adapt our architecture to \(M = 64\), we would achieve a throughput of roughly 37.4Mbps (at 63MHz), which is a 16\(\times\) advantage compared to the reference design in \[5\]. This advantage is a consequence of our highly parallel implementation that deploys \(P = 4\) inner decoders, processes two code frames concurrently, and uses the butterfly scheme to process the trellis diagrams in both directions simultaneously.

### C. Comparison to Conventional \(P = 1\) Architecture

Finally, we illustrate the increased hardware efficiency coming along with our subblock-parallel implementation compared to a conventional \(P = 1\) design, considering the limited resources provided by the target FPGA device. In particular, it can be seen from Tbl. I that the considered FPGA model allows us to instantiate two \(P = 4\) decoders (each working independently on different code frames) on a single device. On the other hand, our implementation results indicate that the available BRAM resources allow only five independent \(P = 1\) decoders to be mapped on one device, mainly due to the large overhead of \(5\times\) replicated input, output and intermediate memories. Hence, the subblock-parallel architecture achieves a \(8/5 = 1.6\times\) throughput advantage per FPGA device compared to the conventional architecture. It is worth mentioning that, according to our implementation results, the described advantage of the subblock-parallel architecture would also be present if we employed sliding-window trellis processing in the SISO decoders, which is typically done in turbo-decoder designs for cellular communications implemented in application-specific integrated circuits with the goal to reduce the memory requirements in the SISO decoders (see, e.g., \[4\]).
TABLE I
RESOURCE-UTILIZATION BREAKDOWN OF $P = 4$ SCPPM TURBO-DECODER IMPLEMENTED ON A VIRTEx-6 XC6VLX240T FPGA DEVICE

<table>
<thead>
<tr>
<th></th>
<th>Logic slices used (%)</th>
<th>BRAM blocks used (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outer decoders</td>
<td>11.5</td>
<td>3.3</td>
</tr>
<tr>
<td>Inner decoders</td>
<td>27.6</td>
<td>1.6</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>6.9</td>
<td>16.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>46</strong></td>
<td><strong>21</strong></td>
</tr>
</tbody>
</table>

TABLE II
COMPARISON TO PRIOR-ART IMPLEMENTATION

<table>
<thead>
<tr>
<th></th>
<th>[5] $M = 64, K = 7560$</th>
<th>This work $M = 16, K = 7560$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. clock frequency (MHz)</td>
<td>63</td>
<td>100</td>
</tr>
<tr>
<td>Max. throughput* (Mbps)</td>
<td>2.35</td>
<td>39.5</td>
</tr>
<tr>
<td>Throughput* (Mbps) at 63MHz</td>
<td>2.35</td>
<td>24.9</td>
</tr>
</tbody>
</table>

*Measured at 20 turbo iterations

VI. CONCLUSIONS

Future space missions require high-throughput SCPPM turbo-decoders. In this paper, we have presented a novel generic subblock-parallel SCPPM turbo-decoder architecture that has been carefully optimized for high-throughput following best practice reported in the turbo-decoder literature. To handle the high memory-bandwidth coming along with subblock-parallel decoding, an efficient memory architecture has been proposed that exploits the mathematical properties of QPP interleavers to guarantee contention-free access. The reported FPGA-implementation results demonstrate that we can achieve a decoding throughput of more than 39Mbps with a single decoder instance deploying 4 inner and 8 outer SISO decoders, which is high enough to decode one LLC channel in real-time and corresponds to a more than 10-fold throughput gain compared to prior-art implementations.

REFERENCES