Design and Verification of a Parameterizable Interleaver and Deinterleaver for Visible Light Communication

Mateus G. Silva, Gabriel R. T. Araújo, Elisa S. Bacelar, Ricardo O. Duarte
Universidade Federal de Minas Gerais
{mateusgs,grt.araujo,elisabacelar,ricardoduarte}@ufmg.br

ABSTRACT

This paper proposes an open IP block of a parameterizable interleaver and deinterleaver for Visible Light Communication (VLC). Architectural details, verification methodology, and FPGA synthesis are explored for the implemented digital design. Achieved results in device utilization and performance are comparable to available commercial IPs and met throughput requirements of the target application.

KEYWORDS

Interleaver, VLC, Reusable IP

1 INTRODUCTION

The process of data exchange in any communication channel is subjected to interference from the environment that may cause errors in the transmitted data. These errors can be individual or bursts of bit flips [1, 2]. Burst errors are harmful because they may easily violate the maximum capacity of Error Correction Codes (ECC) employed by communication systems. Interleavers are designed in order to mitigate such disturbance caused by burst errors. They are present in many communication standards such as HSPA+, 3GPP, WiMAX, WLAN, and DVB-T/H. This work proposes an implementation of a generic digital system (de)interleaver compliant with IEEE 802.15.7 [3], which is an emerging standard for VLC.

The fundamental principle of a (de)interleaver is to reorder a set of data through a certain permutation pattern [2]. By doing so, eventual burst errors are spread throughout the data stream, which leverages the effectiveness of ECCs in correcting them [1]. Permutation procedures applied in interleavers are classified into two main categories: block and convolutional. Block interleavers organize data into a matrix structure, in which data is written column-wise and read row-wise. Convolutional interleavers distribute input data into branches of shift registers with increasing length, and commutators select one of those branches for input and output data using a rotating scheme. IEEE 802.15.7 requires a block interleaver.

The paper is organized as follows: Section 2 analyses some related works with interleaver proposals. In section 3, the architecture of our IP block is described in detail. Section 4 covers the verification process used to check the implemented RTL design and also analyzes synthesis results obtained for the target FPGA device. Conclusions and future works are presented in section 5.

2 RELATED WORKS

In order to contextualize the reader about some types of existing interleavers used in communication systems, this section reviews academic papers of digital designs of interleavers for WiMAX and DVB. These standards have been selected due to their high number of citations in technical papers. In addition, commercial IPs that provide general purpose interleavers will also be reviewed. Even though their implementation is proprietary, their manuals describe interleavers that function similarly to our proposed IP.

WiMAX (IEEE 802.16e) is a broadband wireless communication system with long range and high transmission rate. This standard requires an interleaving method based on a block interleaver similar to what is proposed in IEEE 802.15.7. The main difference is that WiMAX interleaver also performs inter-row and inter-column permutations, whose parameters depend on the modulation schemes and the data rates required for a transmission. Papers [4] and [5] propose optimized designs of address generation circuits for (de) interleaver that support all operation modes prescribed in WiMAX.

Digital Video Broadcasting (DVB) is a transmission scheme focused on the broadcasting of audio and video services over satellite, in cable networks and via terrestrial transmitters. As shown in [6] and [7], this standard requires convolutional, bit and symbol interleavers. Bit interleaving consists of multiple parallel bit interleavers with different permutation sequences each, whereas the symbol interleaver operates by applying pseudo-random permutations to even-indexed-symbols during writing and to odd-indexed-symbols during reading. Those blocks have no direct connection to our work since they perform types of permutation very different from [3].

The commercial IP blocks cover convolutional and block interleaver features. Their block interleaver operates writing input symbols row-wise and reading the block column-wise. Xilinx [8] and Lattice [9] IPs also support inter-row and inter-column permutations if required. Row and column permutations are costumizable and predefined during their configuration. Both IPs support variable-size messages and incomplete rows. MegaCore IP [10] has a simpler design, which disallows neither permutations nor incomplete rows and supports only constant row and column widths.

As regards the works presented, only the (de)interleavers developed by the commercial IPs have a solution applicable to IEEE 802.15.7 standard. Therefore, our work can make a contribution by developing an open implementation of this block.

3 ARCHITECTURE DESIGN DESCRIPTION

Equation (1) defines the permutation rule of the block interleaver specified by IEEE 802.15.7. The number of rows (n) has a fixed value, whereas the interleaver depth (D) represents a variable number of columns filled after the writing process, which is bounded by the internal memory size. The dimension of the block (S_{block}) is given by $n * D$, and p is the number of missing elements to complete the last column of the block. $l(i)$ provides the interleaved indexes, while $z(t)$ obtains the location of the indexes to be punctured. Our implementation provides a feature that encompasses the behaviour

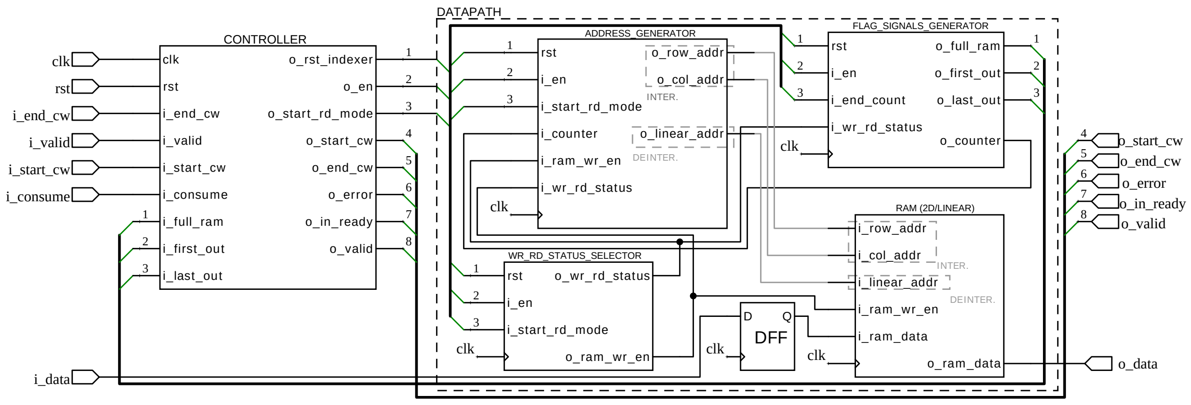

Figure 1: Top-level architecture of both interleaver and deinterleaver blocks.

of $l(i)$ and $z(t)$ at the same time. Thus, post-processing for puncturing is not needed. In order to obtain a generic solution of this block interleaver, n, the internal memory size, and input symbol width (w) are parameterizable in our IP.

$$l(i) = (i \, mod \, D) * n + \left\lfloor \frac{i}{D} \right\rfloor; \; i = 0, 1, ..., (S_{block} - 1)$$
$$z(t) = (n - p + 1) * D + t * D - 1; \; t = 0, 1, ..., (p - 1) \quad (1)$$

3.1 Top-level architecture

Fig. 1 illustrates the top-level architecture for both interleaver and deinterleaver. They have the same port interface (Table 1) and architectural organization. The (de)interleaver datapath consists of three internal blocks, along with a RAM memory unit. The *flag_signals_ generator* keeps track of the number of elements stored in RAM and provides relevant control flags: RAM's maximum capacity (*o_full_ram*), the transmission of the first (*o_first_out*) and the last (*o_last_out*) output data. The *wr_rd_status_selector* determines whether the datapath is in reading or in writing mode and enables writing in memory. The *address_generator* updates the RAM address indexes whenever a new symbol is received or sent.

Table 1: Ports interface of (de)interleaver

Name	Description
clk	System clock pin
rst	System reset pin
i_consume	Readiness to consume o_data
i_end_cw (o_end_cw)	End of data reception (transmission)
i_valid (o_valid)	Validity of input (output) symbols
i_start_cw (o_start_cw)	Start of data reception (transmission)
i_data (o_data)	Input (output) data
o_in_ready	Readiness to accept new input symbols
o_error	Error status

The RAM addressing mode is the only difference between interleaver and deinterleaver blocks, as seen in Figure 1. The interleaver uses a RAM with two dimensions (2D), addressed by rows and columns, because it prevents computing complex arithmetic operations, such as multiplication and mod, for calculating linear memory addresses. On the other hand, the deinterleaver uses a linear RAM, as the received codeword is manipulated using a single index. The 2D addressing mode is not used in this case because the number of columns D is known in advance and, for that reason, it is not possible to write row-wise in memory.

3.2 Address Generator - Interleaver

Since a 2D RAM memory is used for the interleaver, the address generator (Fig. 2(a)) needs to increment row and column indexes in column-wise and row-wise fashion for writing and reading processes to transmit symbols in the correct order. *2D_index_counter* is responsible for such task, and it basically incorporates two counters which are incremented according to the required mode (*i_ mode*). The *2D_index_counter* is set to column-wise during the writing process, then the row counter bounded by "$n - 1$" is incremented whenever a new valid symbol is received, and the column counter is incremented whenever the row counter returns to zero. Once the complete codeword is received, *2D_index_counter* is reset and its current indexes ($D - 1$ and I_{last_row}) are saved into registers. Row-wise mode is set in *2D_index_counter* during the reading process, and the roles of its two counters are switched. The only difference is that the column counter is bounded by *i_col_bound*, which is known after the reading process. Since the last column might be incomplete, a comparator detects the rows with one element less and *i_col_bound* is decremented accordingly. Hence, puncturing procedure is already done.

3.3 Address Generator - Deinterleaver

The deinterleaver is supposed to apply the inverse operation of the interleaver. Then, its address generator must write the input data row-wise and read it column-wise. However, the deinterleaver uses a linear RAM because the number of columns (D) is variable due to the dynamic depth supported by our IP. Memory indexes during the writing process are determined by a counter inside *flag_signals_generator*, which monitors the number of received symbols. Hence, row and column indexes of the block deinterleaver must be mapped into a single memory address during the reading process. For this reason, a simplified version of *2D_index_counter* with only column-wise mode is still required because the number of columns (D) and the number of elements of the last column (I_{last_row}) are used to convert 2D (I_{row} and I_{col}) to linear (I_{lin}) addresses according to equation (2).

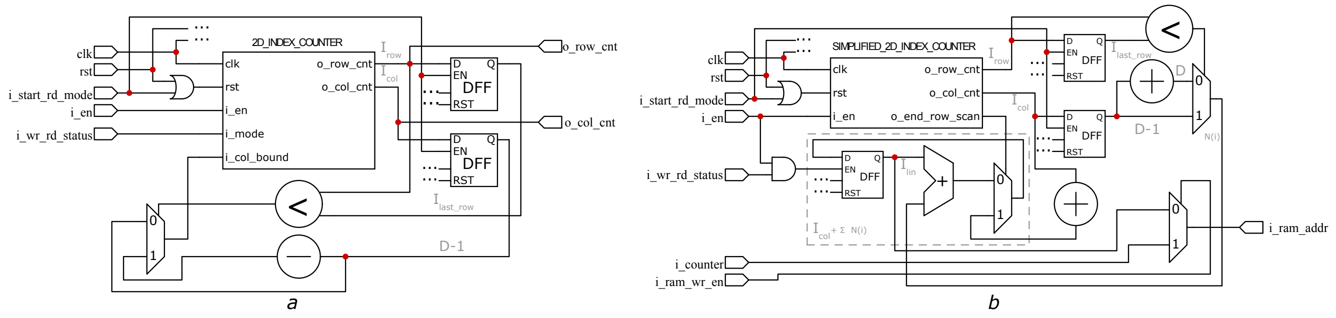

Figure 2: Interleaver (a) and Deinterleaver (b) address generators

$$I_{lin}(I_{row}, I_{col}) = I_{col} + \sum_{i=0}^{I_{row}-1} N(i) \tag{2}$$
$$N(i) = \begin{cases} D, i \leq I_{last\_row} \\ D-1, i > I_{last\_row} \end{cases}$$

At the end of that writing process, $D-1$ and I_{last_row} are stored into the registers, and *simplified_2D_index_counter* is reset. Fig. 2(b) highlights the variables of equation (2). The signal *o_end_row_scan* is a delimiter that indicates that the row counter reached its bound. Then, the register which holds I_{lin} is set to $I_{col}+1$ (I_{col} is just about to be updated at this point) to start the next column-wise scan.

3.4 Controller

The control unit implements the same FSM (Fig. 3) for the interleaver and the deinterleaver. During reset, the controller is set to *WAIT_SYMBOL (WS)*. Once the interleaving process is initiated, it goes through two sequential phases: codeword reception and transmission. *RECEIVE (RC)* and *STALL* belong to the first phase. The former is responsible for storing valid data, and the latter freezes the datapath when input data is invalid. *LAST_INPUT (LI)* does the transition between these two phases and stores the last codeword symbol. The second phase is composed of *CONSUMING (CS)* and *LAST_ OUTPUT (LO)*. Those states are responsible for handling the transmission of stored data in the reordered way according to external requests for data. *LO* is just a specialized transmission state for signaling end of the output codeword. At the end of the transmission phase, the control unit moves back to the WS. The ERROR state exists only to interrupt the (de)interleaver if an unexpected sequence of control input signals is detected. The controller only leaves that state if a reset signal is received.

4 VERIFICATION, RESULTS AND DISCUSSIONS

The proposed IP was verified considering the parameters values obtained from IEEE 802.15.7. Then, w and n are equal to 4 and 15, and the memory has size of 4385 symbols. Formal Verification (FV) was the approach selected to verify the (de)interleaver due to its simplicity to set up the verification environment and exhaustive nature for non-complex problems. This method uses tools that mathematically analyze the state space of the Device Under Test

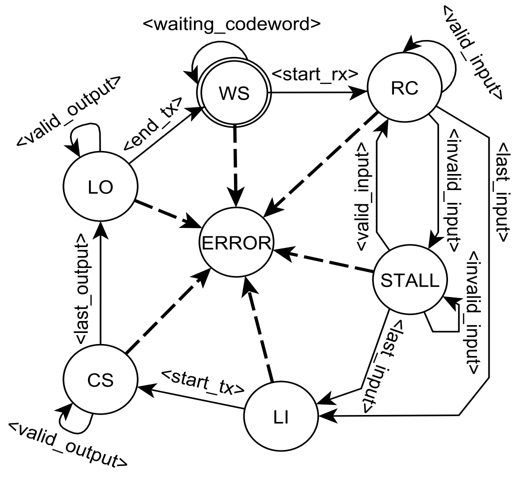

Figure 3: FSM implemented by the CONTROLLER.

(DUT) instead of performing a simulation based on input vectors [11]. The system requirements to be verified are expressed in SystemVerilog Assertion (SVA) properties, and the FV tool informs whether a property is proven or has a counter example to debug.

The FV tool adopted in this work is Cadence® JasperGold® 2019.12v, and the machine model used to run the experiments is a Intel® Core® i7-7700 3.6 GHz with a RAM memory of 16 GB. The system requirements of (de)interleaver are split up into Reset, Interface Control and Functional categories. For each assertion, a cover is created to attest the reachability of its related expression. Reset and Interface Control are mapped into 16 assertions, proven in less than 10 minutes. For Functional, the internal operation of the interleaver is certified using 11 assertions, proven for a progressive range of memory sizes in 12 hours. Then, the interleaver is used as a golden model to verify the deinterleaver, and they are connected together as showed in Fig. 4. This merging enables writing a simple check to attest the functionality of the deinterleaver:

```
INT_FUNC_REQ:
assert property (o_valid && o_cnt == 1 |=> o_saved_data == o_data)
```

Assumptions were created to model a contiguous input codeword of 1 up to 4385 symbols. These constraints exclude cases which have

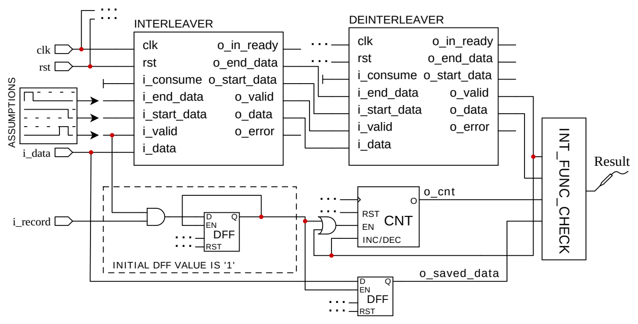

Figure 4: Verfication model for *INT_FUNC_REQ*.

invalid symbols in the input sequence. This was intentionally done to lighten proof convergence of *INT_FUNC_REQ*; however, stall mechanism was checked separately by attesting that state elements in the datapath (e.g address counters and RAM memory) remain unchanged when *i_valid* is '0'. *INT_FUNC_REQ* uses signals from a helper logic that enables FV solvers to arbitrarily store a single valid input symbol and keep track of the position of it within the received codeword. It prevents using large temporal delay operators - which increases property complexity dramatically - in *INT_FUNC_REQ* to compare deinterleaver data output with interleaver data input. Given the available computational resources and the large state space determined by the input sequence of symbols, the memory size was reduced to 438 symbols. Then, *INT_FUNC_REQ* could be proven in 14 hours.

In order to observe the critical paths and calculate the Throughput (*Th*) of the (de)interleaver, FPGA synthesis was carried out for the implemented IP. This experiment was executed in Quartus ®Prime Lite Edition 18.1.0 for 5CGXFC5C6F27C7N device from the family Cyclone V GX using "Slow °C Timing Model". Table 2 displays combinatorial logic and register utilization (CLU and RU), maximum clock frequency (F_{max}) and the Th for the interleaver and deinterleaver block. RAM memory has specific columns (CLU_{men} and RU_{mem}) because it dominates the FPGA utilization, whereas CLU_{others} and RU_{others} are related to remaining components.

Table 2: Synthesis results

IP	LCU_{mem}	LCU_{others}	RU_{mem}	RU_{others}	F_{max}	Th
Interleaver	10367	98	17580	69	84.1 Mhz	168.2 Mbps
Deinterleaver	10329	128	17540	82	134.3 Mhz	268.6 Mbps

Interleaver and deinterleaver have very similar resource utilization; however, maximum frequency is considerably higher in deinterleaver. The critical path in the interleaver is inside its 2D RAM memory. It turns out that a linear memory is still synthesized for it, and there is an additional arithmetic logic responsible for translating row and column addresses into a single linear address. On the other hand, deinterleaver does not have such overhead since it already uses a linear RAM memory, and its critical path comprehends the combinational logic that goes from *wr_rd_status_selector* to the address port of the linear RAM memory.

$$Th = F_{max}\frac{w}{2} \tag{3}$$

Equation 3 calculates the Th of the proposed IP. Since the system requires reading the complete codeword prior to its transmission, Th is divided by 2. Given that IEEE 802.15.7 established that the PHY operating modes that use interleaver modulate a single bit using an optical clock of 200 Khz, the resulting Th with Mbps scale meets by far the standard requirements. Furthermore, the F_{max} obtained for the proposed IP is similar to [10] (120 Mhz); however, it is still behind [8, 9] (>300 Mhz). These comparisons indicate that there is room for upgrades in our IP, but they are inaccurate since each IP uses different setups and devices to extract results. It has also been observed that all manuals present some latency between receiving and transmitting codewords in their waveform examples - probably due to some needed arithmetic calculation. On the other hand, the proposed IP does not require such latency since it does not need complex arithmetic blocks to perform any operation.

5 CONCLUSIONS

This paper presented a flexible IP for a block interleaver and deinterleaver targeted for IEEE 802.15.7. However, it may be employed in any other application that requires a generic block interleaver to prevent burst errors. All requirements of the implemented IP could be fully checked using FV methodology. Also, FPGA synthesis results on the target device fulfilled throughput requirements of IEEE 802.15.7 and demonstrate the feasibility of the proposed IP. In summary, the main features presented by our IP are its reconfigurable capacity, incorporated puncturing mechanism, and simple architecture through avoidance of complex operations. For future works, it is planned to make the block even more customizable by including features related to convolutional interleaving and inter-row and inter-column permutations. Also, further investigations regarding the synthesis of the 2D RAM will be carried out since the FPGA device used as prototype for our IP delivers an inefficient synthesis for such memory block.

ACKNOWLEDGMENTS

The authors thank CNPq, PRPq/UFMG, CAPES and FAPEMIG for the support and Cadence Design Systems for providing licenses.

REFERENCES

[1] Yun Q Shi, Xi Min Zhang, Zhi-Cheng Ni, and Nirwan Ansari. Interleaving for combating bursts of errors. *IEEE circuits and systems magazine*, 4(1):29–42, 2004.
[2] Rizwan Asghar. *Flexible Interleaving Sub–systems for FEC in Baseband Processors.* PhD thesis, Linköping University Electronic Press, 2010.
[3] IEEE. Standard for local and metropolitan area networks–part 15.7: Short-range optical wireless communications," in ieee std 802.15.7-2018. Revision of IEEE Std 802.15.7-2011, 2019.
[4] N Pradeep and Smt H Umadevi. Design & implementation of address generation circuit of ieee 802.16 e deinterleaver using fpga.
[5] Nithish Kumar Venkatachalam, Lakshminarayanan Gopalakrishnan, and Mathini Sellathurai. Low complexity and area efficient reconfigurable multimode interleaver address generator for multistandard radios. *IET Computers & Digital Techniques*, 10(2):59–68, 2016.
[6] Hossein Afshari and Mahmoud Kamarei. A novel symbol interleaver address generation architecture for dvb-t modulator. In *2006 International Symposium on Communications and Information Technologies*, pages 989–993. IEEE, 2006.
[7] Yun-Nan Chang. A multibank memory-based vlsi architecture of dvb symbol deinterleaver. *IEEE transactions on very large scale integration (VLSI) systems*, 18(5):840–843, 2009.
[8] Xilinx. Interleaver/de-interleaver v8.0 - logicore ip product guide. IP Manual, 2015.
[9] Lattice. Interleaver/de-interleaver ip core user's guide. IP Manual, 2010.
[10] Altera. Symbol interleaver/deinterleaver - megacore function user guide. IP Manual, 2002.
[11] Erik Seligman, Tom Schubert, and MV Achutha Kiran Kumar. *Formal verification: an essential toolkit for modern VLSI design.* Morgan Kaufmann, 2015.