A custom co-processor for the discovery of low autocorrelation binary sequences

Abstract

We present a custom processor that was designed to enhance algorithms of finding Low Autocorrelation Binary Sequences (LABS). Finding LABS is very computationally exhaustive, but no custom computing solutions have been reported in the literature so far. A computational kernel which allowed creating an effective single-purpose processor was determined and an appropriate architecture was proposed. The selected elements of the architecture were coded in High-Level Synthesis (HLS) language to speed up the design process. Afterwards, the processor was verified and tested in Xilinx’s Virtex7 FPGA. At the beginning of the paper, we briefly present the finding LABS problem and its importance. Later, we deliver the algorithm, its custom processor structure, and implementation results in terms of the processor performance, size and power.

Keywords: LABS, SDLS algorithm, custom processor, HLS, FPGA.

1. Introduction

Binary sequences with low aperiodic autocorrelation levels have many important engineering applications, such as e.g. radars. Simple pulsed radars are limited in range sensitivity by the average radiation power and in range resolution by the pulse length [1]. Therefore, they are replaced by pulse compression radars, where the pulse compression theory provides high range resolution and a good detection quality. Consequently, the study of binary sequences with low autocorrelation levels becomes a classical problem of signal design for digital communication. The LABS have also other practical applications, such as sonars, spread spectrum communications, system identification, and cryptography. It is also a notoriously difficult problem of combinatorial optimization.

Different criteria of binary sequence “goodness” have been defined for LABS. The one is the “merit factor” $F$ [2], which has the drawback of the high computational intensity when making a computer search for optimally low autocorrelation. On the other hand, the merit factor has the engineering advantage. It is tightly correlated with the signal to self-generated noise ratio of a communication or radar system in which coded pulses are transmitted and received.

Let us a binary sequence of length $L$ be defined as $s = s_1, s_2, \ldots, s_L$, with $s_i = \{-1, +1\}$ for all $i$. Its autocorrelation is given by

$$C_k(s) = \sum_{i=0}^{L-1} s_i s_{i+k}, \quad k = 0, \pm 1, \ldots, \pm (L-1). \quad (1)$$

For $k = 0$, the value of autocorrelation equals $L$ and is called the peak. The merit factor is a function defined as a ratio of the squared sequence length over twice the sum of the squares of the $L$ - 1 off-peak autocorrelations:

$$F(s) = \frac{L^2}{\sum_{k \neq 0} C_k^2(s)}. \quad (2)$$

The sum in the denominator of Equation 2 is called the sidelobe energy of the sequence $E(s)$. The best binary sequence in the sense of achieving the maximum possible merit factor has the minimal energy $E(s)$.

Belonging to the exponential size $O(2^L)$ of the configuration space, an exhaustive search for optimal binary sequences of length $L$ is unrealistic, except for a relatively small $L$. The full search algorithm with run-time characteristic $O(1.85^L)$ is discussed in [3].

Suboptimal algorithms have been introduced to extend the LABS search for lengths of up to a few hundred. Evolutionary algorithms (EA), which borrow from genetic algorithms, evolutionary strategies, and memetic algorithms (MA) have been proposed respectively in [4] and [5]. In EA, individuals in a population compete and exchange information with one another. There are three basic genetic operations, namely, crossover, mutation, and selection. The initial population is usually generated randomly while the populations of other generations are generated from some selection/reproduction procedure. MA are genetic algorithms (GA) that are enriched with an individual learning procedure capable of performing local refinements [6]. The individual learning is usually applied after the selection operation. Local search optimizations are often applied as the learning procedures within MA.

Evolutionary methods supported by local-search operators deliver optimal or near-optimal results for LABS whose optimal solutions are known today. In our research, the steepest descent local search (SDLS) procedure was a subject of the custom processor design and implementation. The processor must offer a high efficiency as for every MA generation the local search is applied to a large set of candidate sequences. An analysis of different local search strategies, used as standalone techniques and embedded within memetic algorithms is provided in [5].

2. The SDLS algorithm

A pseudocode of the algorithm for SDLS is presented in Figure 1. It searches for the best bit change that leads to improvement in the sidelobe energy of the sequence. Only one bit can be changed (flipped) at each algorithm iteration, and the best flip is selected. The algorithm outer loop is repeated as long as a single bit alteration allows the sequence to reach a local minimum of the sidelobe energy. The number of inner loop number of iterations is equal to the sequence length.

```
inputs: s_i - input sequence
        L - input sequence length
outputs: s_o - output sequence
variables: s - sequence
           s_flip - sequence after bit flip
           e_best - best sidelobe energy found
           e_flip - sidelobe energy after flip
           improv - improvement flag
           best_flip - best bit to be flipped

functions: flip_bit(s,i) - s,s\rightarrow s
           energy(s) - calculate sidelobe energy

algorithm SDLS(s_i); returns s_o
1: s := s_i; improv := true;
2: repeat
3:   e_best := energy(s);
4:   for i := 1 to L do /* search best move*/
5:     s_flip := flip_bit(s, i);
6:     e_flip := energy(s_flip);
7:     if (e_flip < e_best) then
8:       e_best := e_flip;
9:       improv := true;
10:      best_flip := i;
11:   end if
12: end for
13: if (improv == true) then
14:      s := s_flip;
15: end if
16: until not improv
17: s_o := s;
18: return s_o
```

Fig. 1. The SDLS algorithm
3. The architecture of the custom processor

To compete with the performance of general purpose CPUs, the custom processor has to introduce some method of parallelism. The best approach that mitigates a data transmission bottleneck in hybrid CPU-FPGA configuration is to implement the algorithm pipelining. However, it can be noted that the inner loop beginning at line 4 in Figure 1 features data dependency. The instruction ‘e_best := e_flip;’ cannot be executed in the next iteration as long as the instruction ‘e_flip < e_best’ is not completed in the previous iteration. The inner loop can be unrolled and pipelined as long as the ‘e_flip < e_best’ and ‘e_best := e_flip;’ instructions are performed in a single loop execution step [7, Chapter 2]. Thus, the lines 7 and 8 of the algorithm set the limit of pipeline throughput. A nuisance that prevents an easily pipelined implementation of the SDLS algorithm is the outer loop that starts at line 3. The outer loop has a dynamic range and, therefore, cannot be pipelined.

The solution for the SDLS processor architecture was inspired by the scheme GPGPUs execute computational kernels in their streaming multiprocessors (SMs). A set of the kernel threads is introduced though that was not the case in the presented implementation. Also, the side lobe energy of the optimized sequences is not returned at the moment to the host processor.

4. The FPGA-accelerated system setup

For the setup of the FPGA-enhanced computing system several elements were used; both hardware and software components.

An IP-Core for the SDSL processor was assembled in Xilinx’s Vivado 2015.4 Design Suite. Library IP-Cores are central to that platform oriented design tool. The AXI 4.0 standard is a preferable standard for a peripheral bus in Vivado. That embedded system bus fits various different computing platforms that are in use today, and it can be bridged with the PCIe bus. That provides a good cross-platform compatibility of designs.

The Vivado HLS was used to quicken a coding process of SDLS processor. The Vivado HLS allows a designer to provide a C language code instead of an HDL code, and synthesize an appropriate design netlist. A designer uses pragmas to control the process of synthesis. Although HLS provides control details of a design automatically, the data-flow of the IP-Core processor should be devised carefully by a designer before a coding process starts. As the processing model of Vivado HLS does not deliver the data feedback capability that is visible in Figure 2, only the ‘SDLS inner loop processing’ block was coded in HLS. The remaining parts of the IP-Core were prepared in VHDL. The s_i and s_o interfaces were implemented as the AXI-stream bus that is the best choice for the pipeline processing. Source codes of the SDLS processor are available at [8].

As a hardware platform, a server computer (CPU: 2×Intel Core i7 950, 3.07GHz; RAM 12 GB 1066MHz; OS: ubuntu 14.04 ) equipped with FPGA card (Xilinx’s XC7VX485T board with Virtex-7 XC7VX485T FPGA, PCIe 2.0:8) was selected for the experiment. The hardware-software interface is necessary to establish communication between the host’s OS and peripheral accelerating card. The perfect choice for our experiment came out to be the Xillybus solution [9]. Xillybus consists of an FPGA IP core and a driver for the computer. Xillybus allows a host application to read/write data to a device file, and it lets an FPGA IP-Core to read/write from FIFO. Also, Xillybus acts as a wrapper that is available for different system platforms (PCIe, AXI). That makes the Xillybus solutions easy to migrate between embedded and server systems.

The Xillybus IP-Core and drivers for the PCIe-based system were generated by the Xillybus IP-Core factory on-line tool. The Xillybus FIFOs were integrated with AXI-stream interfaces of the SDLS custom processor. The Vivado software was used to generate the bitstream necessary to configure the VC707 board.

5. Results

We wanted to verify the performance of the SDLS algorithm when executed on FPGA accelerator i.e. what is the maximum clock frequency of the SDLS processor. Also, it was necessary to check what was the maximum length of binary sequences that our custom processor architecture allowed us to process with today’s available FPGA resources. Clearly, the size of the processor has to grow with the L value as the constant requirement was set to complete the algorithm’s inner loop every single clock cycle. Simultaneously with the size, the latency of the ‘SDLS inner loop processor’ also grows with the length L.

Although the dissipated power is not the main concern in the case of LABS computations, we wanted also to know the energy requirements of our FPGA solution.

We implemented, simulated and run the SDLS processor for several values of the length L. A simple Linux application was prepared to perform experiments on the host. The obtained results in terms of the processor parameters, size, performance, and energy are given in Table 1.
We assessed the power dissipated by our custom processor using a power estimator. The power estimator is a part of the Vivado.

Tab. 1. Results of implementation of SDLS processor and Xillybus framework in Virtex-7 XC7VX485T

<table>
<thead>
<tr>
<th>Parameter</th>
<th>L=10</th>
<th>L=24</th>
<th>L=32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency, clock cycles</td>
<td>10</td>
<td>25</td>
<td>32</td>
</tr>
<tr>
<td>Clock frequency, MHz</td>
<td>250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flip-flops</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(out of 303,600)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9,041 (2.98%)</td>
<td>17,966 (5.92%)</td>
<td>29,348 (9.67%)</td>
<td></td>
</tr>
<tr>
<td>LUTs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(out of 607,200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9,539 (1.57%)</td>
<td>17,737 (2.92%)</td>
<td>27,922 (4.60%)</td>
<td></td>
</tr>
<tr>
<td>BRAM blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(out of 1,030)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.5 (1.12%)</td>
<td>12 (1.17%)</td>
<td>12 (1.17%)</td>
<td></td>
</tr>
<tr>
<td>DSP blocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(out of 2,800)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 (0%)</td>
<td>200 (7.14%)</td>
<td>528 (18.86%)</td>
<td></td>
</tr>
</tbody>
</table>

For the experiment, the optimized SDLS algorithm that was proposed in [5] was prepared in C++. Table 2 provides a comparison of execution times of the SDLS algorithm run both in software and custom processor for different sequence lengths. We measured the optimization time of one million sequences. The values of time given for the SDLS processor include the necessary data transfer time. We also measured the performance of the inner loop of the software application \((P)\). This performance is constant for the SDLS processor and counts 250 million loops per second for all \(L\).

Tab. 2. The estimated maximum power dissipation of Virtex-7 XC7VX485T for the SDLS processor

<table>
<thead>
<tr>
<th>Execution time for 1,000,000 sequences, ms</th>
<th>SDLS processor (ts)</th>
<th>Software app. (ts)</th>
<th>Sustained speedup ((t_s/t_p))</th>
<th>Inner loop performance of software ((P_s/loops/s))</th>
<th>Maximum speedup ((250,000,000/P_s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>L=10</td>
<td>~20</td>
<td>~2,900</td>
<td>~145</td>
<td>~740,000</td>
<td>~336</td>
</tr>
<tr>
<td>L=24</td>
<td>~48</td>
<td>~33,620</td>
<td>~700</td>
<td>~124,000</td>
<td>~2,015</td>
</tr>
<tr>
<td>L=32</td>
<td>~53</td>
<td>~84,450</td>
<td>~1,500</td>
<td>~70,000</td>
<td>~3,370</td>
</tr>
</tbody>
</table>

6. Conclusions

The experiments presented in this paper were performed to verify the effectiveness of FPGA-based computing in finding LABS. The results are very promising as we gained the sustained speedup of up to 1,500-times for the SDLS algorithm. Thanks to the Vivado HLS tool, the FPGA custom processor prototyping was very fast. However, the rough extrapolation of the used FPGA resources (Tab. 1) shows that the presented method works for sequences of hundreds of bits. Beyond this value, the SDLS processor will not fit available FPGAs. The achieved length is decent but sequences as long as a few thousand bits are calculated today. Also, there are other than SDLS local search algorithms in practical use. That shows that the work should be continued.

This work was performed thanks to the funds for AGH statutory activity 11.11.230.017.

7. References


[7] Russek P.: Data-intensive processing on FPGAs, AGH University of Science and Technology Press, 2015


Pawel RUSSEK, PhD

Received PhD degree in Electronics from AGH University of Science and Technology in 2003. He works as an Assistant Professor in the Department of Electronics, AGH UST and as a research fellow in the Academic Computing Center ‘Cyfronet’ AGH. His research interests focus on hardware acceleration in FPGAs. He is particularly interested in High Performance Reconfigurable Computing, Embedded Systems, and the digital systems.

e-mail: russek@agh.edu.pl

Ernest JAMRO, PhD

Received MSc degree in electronic engineering from the AGH University of Science and Technology (AGH-UST), Cracow Poland in 1996; MPhil. degree from the University of Huddersfield (U.K.) in 1997; PhD and habilitation (DSc) degree from the AGH-UST in 2001 and 2014 respectively. He is currently an assistant professor in the Department of Electronics AGH-UST. His research interest includes reconfigurable hardware (esp. FPGAs), reconfigurable computing systems, System on Chip, artificial intelligence.

e-mail: jamro@agh.edu.pl

Michał KARWATOWSKI, MSc

Received the BSc eng. and MSc degrees in electronic engineering in 2013 and 2014 respectively from the AGH University of Science and Technology, Cracow, Poland. Currently a student of PhD studies. His research interests include usage of hardware accelerators in complex computations, control, and energy efficient systems of small and big scale, mainly using Field-Programmable Gate Arrays.

e-mail: mkarwat@agh.edu.pl

Prof. Kazimierz WIAΤR, PhD

Received the MSc and PhD degrees in electrical engineering from the AGH University of Science and Technology, Cracow, Poland, in 1980 and 1987, respectively, and the DSc degree in electronics from the University of Technology of Łódź in 1999. Received the professor degree in 2002. His research interests include design and performance of dedicated hardware structures and reconfigurable processors employing FPGAs for acceleration computing. He currently is a director of Academic Computing Centre CYFRONET AGH.

e-mail: wiatr@agh.edu.pl