

A VHDL Implementation of an On-board ACF Application Targeting FPGAs

E.A. Bezerra¹, M.P. Gough² and A. Buckley³

^{1,2,3} Space Science Centre, School of Engineering, University of Sussex, England

¹ Faculty of Informatics, Catholic University, PUCRS, Porto Alegre, RS, Brazil

<E.A.Bezerra, M.P.Gough, A.Buckley>@sussex.ac.uk

Abstract

This work describes the implementation of the VHDL version of a microcontroller-based board. Some problems and possible solutions of using an HDL in a high level of abstraction are discussed. The gain in performance is an important incentive for using FPGAs as the main processing elements of a system, despite the programming language limitations dictated by the synthesis tools available.

I. Introduction

In December 1997, a NASA sounding rocket flew from Svalbard (Spitzbergen) carrying a scientific application computer designed at the Space Science Centre, University of Sussex. The real time science measurement performed by this embedded system is an auto-correlation function (ACF) processing of particle count pulses as a means of studying processes occurring in near Earth plasmas.

Common features of embedded systems, in general, are compactness and their application specific nature. In addition, some embedded systems have a higher demand for processing power. Good examples of this kind of system are the on-board instruments of spacecraft, which also require fault tolerance capabilities. In order to meet these requirements, microcontrollers have been traditionally used in the design of such systems [1]. However, with the advances in the configurable computing field, it becomes possible to have systems with performance rates hundreds or, in some cases, thousands

of times higher than traditional microcontroller based designs [2][3]. The first configurable computing system was proposed by Estrin in 1963 [4], but it could not be implemented at that time because of the technology limitations. Nowadays, the best way to implement a configurable computer system is by using Field Programmable Gate Arrays (FPGAs) [2]. Some advantages of using FPGAs to replace microcontrollers in embedded systems are:

- Implementation of a real application-specific design. FPGA internal resources can be configured according to the application requirements. In a microcontroller, the application has to adapt to the resources available, and in many cases, not all resources are used;
- The Printed Circuit Board (PCB) for an FPGA may be simpler than the equivalent one for a microcontroller based design. This is because of the possibility of integrating in the same device (FPGA), external hardware components like, for example, FIFOs and state machines;
- The level of performance obtained with an FPGA is higher than with a microcontroller, even when using the same clock, because of the parallel nature of hardware algorithms.

The main disadvantages of FPGAs are their high cost and the limitations imposed by the synthesis tools for developing systems in a high level of abstraction [5][6]. These problems are inherent in this early stage of development of this technology. Despite

these problems, the greatly enhanced possibilities introduced by the configurable computing technology are an invitation for on-board instrument processing implementation [7].

In [8] the implementation of a correlation detector in an FPGA is described. In that work the system is described at a low level of abstraction, by way of schematic capture diagrams. This form of design description has been used as the preferable input format for synthesis tools. The investigation of implementing a complex application using a hardware description language (HDL), in our case VHDL, in a high level of abstraction is one of the motivations for this work.

Another motivation for implementing the microcontroller based ACF application in VHDL was to investigate the feasibility of having a special purpose computer built only with FPGAs, i.e. with no microprocessors – hardware algorithm instead of software algorithm.

The successful implementation of this ACF application using only FPGAs signifies that configurable logic can be used, not only in space applications, but also to replace traditional 8051 and similar microprocessor based special-purpose systems.

The paper is organised as follows: Section II describes the original ACF system; Section III describes the implementation in VHDL, targeting FPGAs; in Section IV area usage and performance predictions are discussed; in Section V there are conclusions and future directions.

II. The Original Design (SVAL)

SVAL is the name given in this work for the sounding rocket computer module responsible for the ACF application. The ACF is a statistical method that can be used to obtain information about the behaviour of a signal, for instance revealing the presence of periodicity in a noisy signal [9]. The implemented ACF is constructed from a time series of particle count pulses $X_i(t)$, at

sampling intervals Δt , by performing a shift/multiply operation using the first half of the time series. Considering a discrete signal, the ACF amplitude R_j at lag $j = 1, \dots, N/2$ is described by equation 1.

$$R_j(\Delta t) = \sum_{i=1}^{N/2} X_i(t) \cdot X_i(t + j \cdot \Delta t) \quad (1)$$

The SVAL system consists of an implementation in hardware and software of this equation. The system is used in the data reduction of a signal in order to send to ground only relevant information about the sampled signal.

The original SVAL board, as shown in Figure 1.a, had two DS87C520 microcontrollers (8051 family) with embedded RAMs, one of them responsible for the low frequency (LF), and the other one for the high frequency (HF) ACF processing. The input for both modules is supplied by two channels of pseudo-random 250ns electron detection pulses (I/P1 and I/P2).

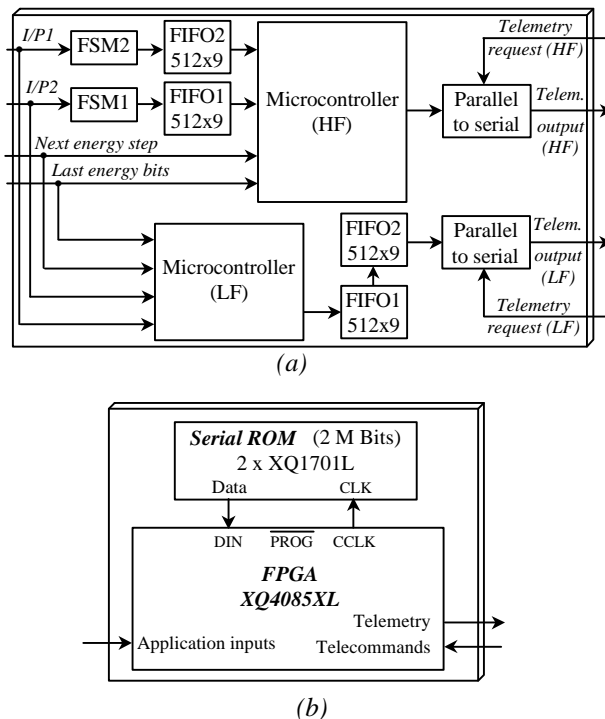


Figure 1. Block diagram of SVALBARD hardware components. (a) SVAL board (22 CIs); (b) SVAL-VHDL board (3 CIs).

The HF module has hardware correlator (state machine) input interfaces via FIFO to implement a 32 lag “buncher” 0 to 8MHz. The LF module executes a “real” ACF having software correlator input interfaces via microcontroller timers 0 and 1, 32 lag LF ACF, 0 to 10 KHz even energies and 0 to 3.3 KHz odd energies, with compression to 10 bits.

SVAl is a typical memory transfer application, with a high input sampling rate and with scarceness of processing modules. There are only two multiply-and-accumulate (MAC) modules and few instances of other arithmetic and logic operations. Basically, the information about the input signal is stored in memory, after or during the sampling, and when an output is requested, blocks of memory are processed and transferred to an output storage area. The processing activity is more demanding in the LF module because of the MAC operations required for the “real” ACF implementation. In the HF module, there are only simple arithmetic operations, as it is implemented by a “buncher” ACF. A summary of the HF and LF modules operation is described next in the VHDL version session.

III. THE VHDL VERSION (SVAl-VHDL)

SVAl-VHDL, the VHDL version of SVAl, was designed with three main goals: increased performance, increased portability, and a reduction in the number of hardware components. These goals could be met with a pure VHDL description, without pre-optimised cores instantiation, and using a generic synthesis tool. The reduction in the number of components, as shown in Figure 1.b, was achieved by using only one FPGA device configured to execute the same functionality as the SVAl system, including both the hardware and the software parts. For instance, the external FIFOs of the HF module were replaced by circular FIFOs implemented using data structures in VHDL.

The microcontrollers’ functional blocks were not described in VHDL, but instead the functionality of the application as a whole was implemented. Performance improvement performance is a result of the parallel nature of VHDL and the flexibility to implement distributed memory on FPGAs. With VHDL, the developer has to use the parallel-programming paradigm from the beginning of the design, making the system closer to real world problems. Using VHDL it is possible to obtain a certain level of portability, as the same code can be used to generate the configuration bitstream for several FPGA families, and even for ASIC generation. However, there is a portability problem related to the synthesis tool. Using different tools may result in different performance and FPGA area usage.

A. The HF Module

The HF module, as shown in Figure 2.a, receives data from the two input channels (I/P1 and I/P2). For each input there is a process, executing a state machine which sends the value of the delay between two adjacent pulses to the *Buncher Histogramme Generator* (BHG) process. It is important to notice that in the SVAl implementation there are two FIFOs between the input state machines and the microcontroller. As shown in Figure 1.a the FIFOs and the state machines were implemented using hardware components. In the SVAl-VHDL, as the processes run in parallel in a non-stop way, there is no need for the FIFOs. The BHG process keeps accumulating statistical information, when available, about the delays in an 1 Kbytes array. The position to be written in this array (histogram) is selected according to the *Last Energy* input, which is a four-bit input used to identify the energy level being sampled. The 1K bytes array is divided in 16 blocks of 64 bytes, and each block is used to store the histogram related to one of the possible 16 energy levels.

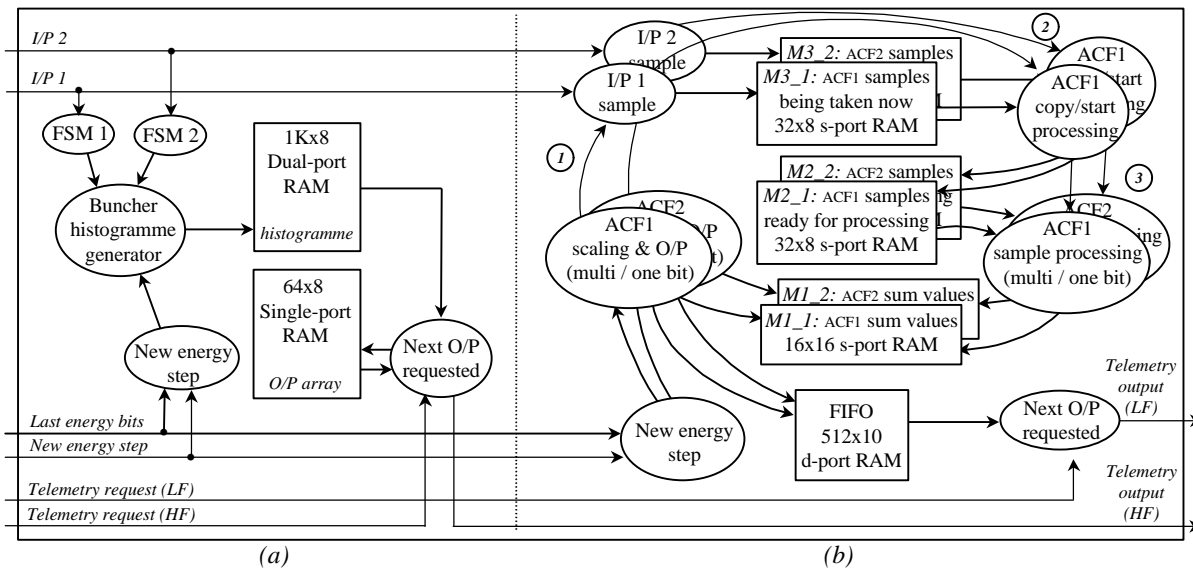


Figure 2. SVAl-VHDL: processes (circles)/memories (rectangles). (a) HF module; (b) LF module.

There are two signals that require servicing: the *New Energy Step*, used by the program to calculate the new base address for the 1 Kbytes array in the event of a new energy step; and the *Next Output Request* generated by the telemetry clock, and used to activate the output process. This process converts to serial and outputs a telemetry byte obtained from the 64 bytes output array. When the transmission of the output array finishes, another block is copied from the 1K bytes histogram array to the 64 bytes output array.

B. The LF Module

The LF module works in a similar way transferring data between arrays, sampling input signals and outputting telemetry data. An important difference from the HF module is that the LF module executes more arithmetic operations, as for example, MACs, since a time series of 32 samples is used. The inputs are the same as the HF module, but there is a separate telemetry output with its respective telemetry clock. The execution flow is slightly more complex than the HF module. At the occurrence of a *New Energy Step* event, the following activities are executed:

i. Output of the last ACF values, calculated

during sampling of the last energy step. This output is done by copying, and formatting for output, 64 bytes from the *ACF sum values* array (ASV) to the output circular FIFO;

- ii. A block of 64 bytes of data, filled by the sampling process in the *last energy step*, is copied from the *ACF samples being taken* array to the *ACF samples ready for processing* array (SRP), in order to free the input array allowing the sampling to happen concurrently to the ACF processing;
- iii. The ACF processing is executed over the data in the SRP array and the results, namely, the ACF sum values, are copied to the ASV array, being made ready for the output described in the item i.

The LF module has more data dependencies than the HF module. Because of this undesirable feature a synchronisation mechanism had to be implemented. In Figure 2.b, at the synch point 3, the ACF processing starts after the end of the ACF samples copy; at the synch point 2, the update of the SRP array is executed after the end of the sampling activity; and at the synch point 1, the sampling starts after the occurrence of a new energy step.

C. Implementation Details

In order for SVAL-VHDL to be synthesizable, the guidelines suggested in [5], [6], [10] and [11] were strictly followed. For instance, for the arrays implementation the VHDL style suggested in the Synplify User Guide [6] was used. The code for memory inferring listed in [6] tells the synthesis tool, Synplify, to use the memory blocks available in the target device. If the same code is synthesised using FPGA Express 1.5 [12], for example, then FPGA flip-flops are used instead of the RAM blocks, and the resulting configuration bitstream is not viable in terms of area. The main recommendation for synthesizable VHDL development is to keep the code as simple as possible. In [10] it is suggested to use only state machines and *if..then..else* constructs in sequential processes. For example, the state machine in Figure 3 was used to implement the nested loop required by Equation 2. The loop repetition is guaranteed by the clock signal as the *case* construct is evaluated in the event of a clock pulse.

```

case NEXT_STATE_FEW_1 is
when SX_C => NEXT_STATE_FEW_1 <= SX_C;
when S1_C => -- external loop
  WR_MEM1_1 <= '0';
  if PTR1_1 < 16 then -- sweep MEM1
    ADDR_MEM1_1 <= PTR1_1; AUX_IDX1 <= "00000";
    PTR1_2 <= '0' & HALF_SEP1;
    ADDR_MEM2_1 <= W_AUX1; AUX1 <= W_AUX1;
    NEXT_STATE_FEW_1 <= S2_C;
  else -- end of multibit ACF (end of ext. loop)
    DONE_SEP1_OUT <= '1';
    NEXT_STATE_FEW_1 <= S1_C;
  end if;
when S2_C =>
  AUXS_1 <= DATAOUT_MEM1_1;
  NEXT_STATE_FEW_1 <= S3_C;
when S3_C =>
  XDT_MEM2_1 <= DATAOUT_MEM2_1;
  ADDR_MEM2_1 <= PTR1_2; NEXT_STATE_FEW_1 <= S4_C;
when S4_C => -- internal loop
  if AUX_IDX1 < 16 then
    AUXS_1 <= AUXS_1 + DATAOUT_MEM2_1 * XDT_MEM2_1;
    PTR1_2 <= PTR1_2 + 1;
    ADDR_MEM1_2 <= AUX1 + 1; AUX1 <= AUX1 + 1;
    AUX_IDX1 <= AUX_IDX1 + 1;
    NEXT_STATE_FEW_1 <= S3_C; -- repeat inner loop
  else
    NEXT_STATE_FEW_1 <= S5_C; -- end of inner loop
  end if;
when S5_C => -- repeat external loop
  WR_MEM1_1 <= '1'; ADDR_MEM1_1 <= PTR1_1;
  DATAIN_MEM1_1 <= AUXS_1;
  W_AUX1 <= W_AUX1 + 1; PTR1_1 <= PTR1_1 + 1;
  NEXT_STATE_FEW_1 <= S1_C;
when others => NEXT_STATE_FEW_1 <= S6_C;
end case ;

```

Figure 3. Partial listing of the multibit ACF process – LF module.

$$MEM\ 1 \leftarrow \sum_{i=0}^N \sum_{j=0}^N MEM\ 2_j * MEM\ 2_{j+1} \quad (2)$$

Figure 4 shows the main process of the HF module. This process is responsible not only for the “buncher” histogramme generation, but also for the New Energy Step monitoring. In the *elsif* block (New Step) the M1_PTR signal, which is a pointer to the histogramme array, is set to a new base address, using the Last Energy input as described before. The FLAG_PWRUP signal is used in order to avoid the start of the ACF processing before the end of the PowerUp process, which runs in parallel with the other processes of the system.

```

SVALHF_PRO: process (CLK_IN, RESET_NEG_IN )
begin
  if RESET_NEG_IN = '0' then
    M1_PTR <= (others => '0');
    FLAG_WHICH_IP <= '1';
  elsif CLK_IN'event and CLK_IN = '1' then
    FLAG_NEW_ESTEP_OLD <= FLAG_NEW_ESTEP;
    if FLAG_PWR_UP_PWRUP = '1' then -- Powerup
      M1_PTR <= LAST_ENERGY_IN & "000000";
    elsif (FLAG_NEW_ESTEP = '1' ) -- New Step
      and (FLAG_NEW_ESTEP_OLD = '0') then
      M1_PTR <= LAST_ENERGY_IN & "000000 ";
      FLAG_NEW_ESTEP_OLD <= '1';
    else -- Buncher Histogramme
      if DATA_READY = '0' then
        BUNCHER_RD <= '0'; BUNCHER_WR <= '0';
        if FLAG_WHICH_IP = '1' then
          FLAG_WHICH_IP <= '0';
          ...
        else
          FLAG_WHICH_IP <= '1';
          ...
        end if;
      else
        if BUNCHER_RD = '0' then
          BUNCHER_RD <= '1';
          BUNCHER_WR <= '0';
          M1_PTR(7 downto 0) <= M1_PTR(7 downto 0)
            + ("000" & AUX_DATA);
        else
          AUX_DATA1k <= DATAOUT_RAM1k + 1;
          BUNCHER_RD <= '0';
          BUNCHER_WR <= '1';
          DATA_READY <= '0';
        end if;
      end if;
    end if;
  end process SVALHF_PRO ;

```

Figure 4. Partial listing of the “buncher” histogramme manager – HF module.

To update the histogramme, two clock cycles are necessary. In the first cycle the M1_PTR pointer is set to a new position (last bold line in Figure 4) and the data to be

written is sent to the memory process responsible for the histogramme control. In the second cycle the signal BUNCHER_WR is set to allow the histogramme array to be written (last *else* in Figure 4).

IV. Area/Performance Remarks

The main motivation for using FPGAs instead of microprocessors for on-board computer implementation, is the gain in performance with a decrease in the PCB area usage. Table 1 shows a comparison of the number of cycles (T) necessary to run some of the SVAL application processes (8051 implementation, written in assembly), and in SVAL-VHDL. All of the original VHDL electronics board, both LF and HF modules, can be implemented in SVAL-VHDL as a single chip, an XQ4085XL Xilinx FPGA [12] with two XQ1701L serial ROMs to store the bitstream (see Figure 1.b).

The RAM implementation is the SVAL-VHDL most consuming area module. A VHDL/FPGA implementation allows the developer to have in the design only the amount of memory necessary for the application. Another space consuming module is the multiplication. For instance, a 16x16 pre-optimised multiplier, generated by CoreGen [12] occupies 213 CLBs. Using the ‘*’ operator in VHDL, as shown in Figure 3 (*when S4_C* block) Synplify can generate a multiplier that consumes 215 CLBs. In both cases considering an XC4000 FPGA, and optimising for area. As there is no significant gain in area using the pre-optimised core, and in order to have all the system described in

VHDL, and at the same abstraction level, the ‘*’ VHDL operator was used.

V. Conclusion

Both systems, the original design and the VHDL version, were implemented without taking into consideration any fault tolerant strategies. The main reason for that is the short mission duration, which was about 20 minutes long. The dependability improvements discussed in [13] are necessary in case of long-life applications where maintenance is not possible, or extremely expensive.

Writing a synthesizable VHDL code, at the behavioural level, is difficult because of the restrictions imposed by synthesis tools. This is the main problem to be solved in order to effectively develop hardware algorithms in the same way as software. For instance, the code listed in Figure 3 could be implemented in a clearer way by using *for* constructors. Unfortunately, because of the present FPGA architectures, the synthesis tools available are not ready to understand and generate a desirable hardware netlist from generic algorithms.

The ACF application described was implemented using only VHDL, and can be used to generate netlists for different target devices. The application was simulated at the RTL and mapped netlist levels. Some modules were simulated also at the structural level (netlist with time information). The next step is to execute the whole application in an FPGA in order to compare the performance results with the expected ones.

Table 1. Performance comparison for the ACF application.

<i>Process</i>	<i>Microcontroller</i>	<i>FPGA</i>	<i>Rate</i>
<i>New Energy Step (HF)</i>	<i>4,518T</i>	<i>1T</i>	<i>4,518 times faster</i>
<i>“Buncher” histogram (HF)</i>	<i>8T .. 36T</i>	<i>1T</i>	<i>8 to 36 times faster</i>
<i>Next o/p requested (HF)</i>	<i>18T .. 1018T</i>	<i>1T .. 68T</i>	<i>18 to 15 times faster</i>
<i>O/P one bit ACF (LF)</i>	<i>1,240T</i>	<i>48T</i>	<i>26 times faster</i>
<i>O/P multibit ACF (LF)</i>	<i>1,334T..3,438T</i>	<i>132T..143T</i>	<i>10 to 24 times faster</i>
<i>Multibit ACF processing (LF)</i>	<i>11,116T</i>	<i>288T</i>	<i>39 times faster</i>

Another experiment to be done is the execution of the application in a reconfigurable computer. In this type of computer the same hardware can be total or partial reconfigured to execute different functions at different instants of time. The application was developed in modules that can operate in a quasi-independent way. With the correct selection of the modules to be used and the sequence of configurations, it may be possible to use a smaller FPGA device.

Acknowledgements

This research is partly supported by the Brazilian Council for the Development of Science and Technology (CNPq), and by Pontific Catholic University of Rio Grande do Sul (PUCRS), Brazil.

References

- [1] Gough, M.P. **Particle Correlator Instruments in Space: Performance Limitations Successes, and the Future.** American Geophysics Union, Santa Fe Chapman Conference, 1995.
- [2] Mangione-Smith, W. et al. **Seeking Solutions in Configurable Computing.** IEEE Computer, pp. 38-43, Nov. 1997.
- [3] Villasenor, J. and Mangione-Smith, W. **Configurable Computing.** Scientific American, pp. 66-71, Jun. 1997.
- [4] Estrin, G. et al. **Parallel Processing in a Restructurable Computer System.** IEEE Transactions on Electronic Computers, pp. 747-755, Dec. 1963.
- [5] Xilinx. **Synthesis and Simulation Design Guide.** Xilinx, 314p. 1998.
- [6] Synplicity. **Synplify Better Synthesis. User Guide 5.0.** Synplicity, 1998.
- [7] Villasenor, J. et al. **Configurable Computing Solutions for Automatic Target Recognition.** In Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, pp. 70-79, Napa, CA, Apr. 1996.
- [8] Ureña, J. et al. **Correlation detector based on an FPGA for ultrasonic sensors.** Microprocessors and Microsystems, Jun. 1999, pp. 25-33.
- [9] Beauchamp, K. and Yuen, C. **Digital Methods for Signal Analysis** George Allen & Unwin, 1979, 316pp.
- [10] IEEE **IEEE P1076.6/D1.12 Draft Standard For VHDL Register Transfer Level Synthesis.** IEEE, 1998
- [11] Figueiredo, M. and Graessle, T. **VHDL Style Guide.** Adaptive Scientific Data Processing. <http://fpga.gsfc.nasa.gov/asdp/index.htm>
- [12] Xilinx **The Programmable Logic Data Book** San Jose, Xilinx, 1999.
- [13] Bezerra, E. et al., **Improving the Dependability of Embedded Systems Using Configurable Computing.** XIV International Symposium on Computer and Information Sciences, Izmir, Turkey, Oct. 1999.