ADVANCED ACCELERATOR CONTROL AND INSTRUMENTATION MODULES BASED ON FPGA*

P. Messmer#, V. Ranjbar, D. Wade-Stein, Tech-X Corporation, Boulder, CO 80303, U.S.A
P. Schoessow, Euclid TechLabs, LLC, Rockville, MD 20850, U.S.A , J. G. Power, ANL, Argonne, IL 60439, U.S.A.

Abstract

Field Programmable Gate Arrays (FPGAs) offer a powerful alternative to Application Specific Integrated Circuits (ASIC) or general purpose processors in accelerator control applications. Software development for these devices can be awkward and time consuming, however, when using low level hardware design languages. To facilitate the use of FPGAs in control systems we are developing a library of software tools based on ImpulseC, a high level subset of the C language specifically designed for FPGA programming. Development and testing of the software will be performed on a Xilinx Virtex-4 FPGA demo board. Here we present initial results of algorithms of relevance to controls applications implemented in Impulse C.

INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are integrated circuits consisting of programmable logic elements and programmable interconnects between these elements. The high gate density available with present technology provides significant computational power to these units. FPGAs have become increasingly important components in distributed control systems for particle accelerators. This is particularly true for applications providing local intelligence and control functions such as implementing PID (Proportional/Integral/Derivative) control algorithms for control of an accelerator subsystem [1], or providing an interface from a PC to a chain of crates in a legacy CAMAC system [2]. FPGAs are also rapidly replacing Application Specific Integrated Circuits (ASICs) because of the reduced development cost of FPGA applications and the ability to revise and upgrade the programming of the system "on the fly". The downside of the improved development cycle and versatility in hardware is the difficulty in software development. Application development in hardware description languages (HDLs) like VHDL or Verilog require the user to become proficient in new and difficult languages. GUI-based approaches in which the programmer specifies the program in terms of a functional block diagram work well for relatively small applications but become unwieldy to use for complex programs. Here we present some results of investigations using high-level languages for FPGA algorithm development.

*Work supported by U.S. DOE Office of Science, Office of High Energy Physics, under grant DE-FG02-06ER84486.
#messmer@txcorp.com

IMPULSE C

While there is a number of tools currently available for high-level FPGA development, we investigate the Impulse C [1,2] language and the CoDeveloper programming environment. Among the reasons for choosing Impulse C is the large coverage of the ANSI C standard, as well as the broad support for various hardware platforms.

Impulse C allows a software developer with knowledge of C programming to describe mixed software/hardware systems. In this environment, hardware/software partitioning is defined using a small set of C-compliant programming extensions (in the form of a set of library functions) for expressing parallelism and process-to-process communications. An Impulse C application programming interface (API) provides communication methods including data streaming, shared memory and message passing models of communication. Impulse C supports different FPGA targets by automatically generating PC-to-host communications and allowing software processes running on the host computer to communicate directly and efficiently with the compiler-generated FPGA hardware. The software developer, using standard C profilers and interactive optimization tools, can decide if a process should be located on the host CPU or on the FPGA and partition the application accordingly.

The Impulse C compiler translates the software processes into native code for the host CPU, while the hardware processes are compiled to produce FPGA-ready hardware definitions. Using a sophisticated platform support package mechanism, code for a broad variety of platforms can be generated, ranging from FPGA accelerated supercomputers like the Cray XD1, to PCMCIA FPGA cards like the Pico Computing E-14 [5].

For completeness, we mention a common value-added feature of FPGAs, the concept of IP (intellectual property) cores. This refers to third party software available on the FPGA to implement a particular functionality. An IP core is referred to as soft (implemented as a configuration bit stream loaded at initialization) or hard (actually burned into memory on the FPGA). Some of the capabilities provided by IP cores are floating point arithmetic, ethernet, fast Fourier transform, and even general purpose microprocessors. Impulse C provides full access to features provided by IP cores.

FIRST EXPERIMENT: BOX-CAR FILTER

Listing 1 shows the source code for the core of the hardware process for the box car filter. This process reads

new values coming on the input stream, adds the value to the running total and subtracts the last value in the window from the running total. It then adds the new value at the position of the last value in the window and advances the position pointer in the ring buffer. Finally, the actual average is computed by dividing the running sum by the window length. This value is then shipped back to the host CPU.

```
while(co_stream_read(input_stream,&c,
      sizeof(int32))==co_err_none)
{
     avg_new = avg + c - buf[pos];
     buf[pos] = c;
     pos_new = pos + 1;

     if(pos_new == BUFLEN) pos_new = 0;
     avg_frac = avg_new / BUFLEN;
     co_stream_write(output_stream,
            &avg_frac,sizeof(int32));
     avg = avg_new;
     pos = pos_new;
}
```

Listing 1: Core of the hardware process for the boxcar filter. This code is translated into VHDL and can then be synthesized into an FPGA configuration file (bitstream).

In order to simulate a data source for this filter, we wrote a producer process which generates data sets on the host CPU and sends them to the FPGA.

TIMING TESTS

Once the VHDL code is generated, Impulse C provides tools that allow determining the performance and timing of a given logic block. It turns out that the conditional for resetting the circular buffer (see listing 1) causes the main loop to be split into three logic parts. As a result, the generated logic requires for the main box-car loop 5 stages (which will correspond to 5 clock cycles on the FPGA). The overall maximum time delay for this logic is estimated to be 96 time units. This time delay will determine the maximum clock frequency at which the FPGA can be run.

By changing the source code for the ring buffer to:

pos_new = (pos + 1) % BUFLEN;

the generated logic only requires 3 stages at the same maximum gate delay. This means that while the maximum obtainable clock frequency will still be the same as with the previous implementation, it will produce a result every 3 clock cycles, rather than every 5 cycles.

One of the great advantages of an FPGA implementation is that one is not limited to sequential execution, but can exploit temporal parallelism (pipelining). In Impulse C, this is accomplished by inserting a compiler directive,

#pragma CO pipeline

at the beginning of a loop. While it takes the pipelined loop still three clock cycles to generate one result, it can start working on a new result while it is still processing the previous result, resulting in one result every two clock cycles.

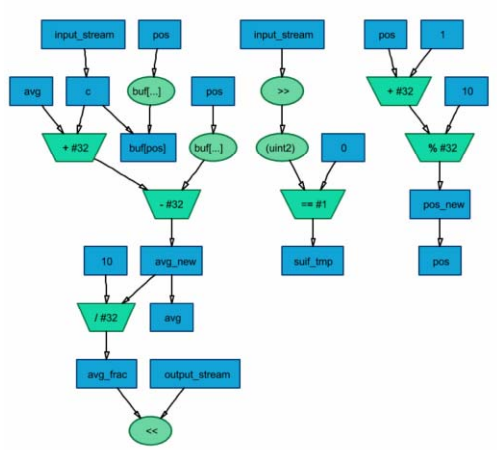

Figure 1: Data flow graph for the box-car filter. These graphs are one of the tools offered by the ImpulseC suite to investigate and optimize the hardware processes generated from C code.

ImpulseC offers a variety of tools to analyze the theoretical performance of a hardware process prior to the place and route step. E.g. data flow graphs help to determine if a sufficient level of instruction level parallelism was extracted by the compiler.

MATRIX-VECTOR PRODUCT

A kernel often encountered in controls applications is the evaluation of a matrix-vector product on integer quantities. Listing 2 shows the source code of a matrix vector product in Impulse C for a vector length N = 100.

While this is a straight forward implementation of a matrix-vector product, one has to note the directive in the inner-most loop. Loop unrolling results in the concurrent execution of all iterations of the inner most loop by replicating its logic. In this case, this results in 100 multiply/add units that are executed concurrently. The overall cost of the entire matrix-vector product is therefore 100 clock cycles.

```
   k = 0;
   for(i = 0; i < N; i++){
     for(j=0; j < N; j++){
#pragma CO unroll
       res[i] += matrix[k]*vec[j];
     }
   }
```

Listing 2: Matrix-Vector product in Impulse C.

After compilation of the source and generation of the hardware configuration, it turns out that the generated hardware can only be run at a frequency of 58 MHz, primarily due to a slow bus interface of our FPGA board. Using an optimized bus interface would allow to run the FPGA at about 240 MHz. At this rate, the overall evaluation of the matrix-vector product with N = 100 takes less than 1.8 µs. For comparison, on a 1.8 GHz AMD Opteron processor, the same computation takes 23 µs.

The significant speedup observed was only made possible by loop unrolling, which exploits the spatial parallelism available on FPGAs. As long as enough logic blocks are available, this results in significant speedup.

The generated hardware for the fully unrolled matrix-vector product takes up about 50 % of all the logic gates on a Virtex 4 FX60 FPGA. While manual optimization of the placement could reduce the logic use, it shows that for more complicated algorithms including multiple matrix-vector products full loop unrolling could not be afforded. However FPGA logic density is expected to increase, thus enabling larger designs to be fully unrolled.

CONCLUSION

Particle accelerator controls applications could greatly benefit from the processing power offered by modern FPGAs. However, the programming complexity of these devices makes is hard for scientists to time-efficiently develop algorithms for these devices. High-level tools, such as Impulse C, mask this programming complexity of FPGAs by allowing the use of standard C for development. Application kernels, like box-car filter or matrix-vector products, can therefore be implemented relatively quickly. The effect of different optimization techniques, e.g. loop unrolling or pipelining, can be investigated prior to the time-consuming place and route steps.

REFERENCES

[1] T. Czarski, K. Pozniak, R. Romaniuk, S. Simrock, ``TESLA Cavity Modeling and Digital Implementation with FPGA Technology Solution for Control System Development", TESLA Report 2003-28.

[2] M. Browne, A. Gromme, E. Siskind, ``Front End CAMAC Controller for SLAC Control System", Proc. ICALEPCS 2001, San Jose CA

[3] Practical FPGA Programming in C, D. Pellerin, S. Thibault, Prentice Hall 2005.

[4] http://www.impulsec.com

[5] http://www.picocomputing.com

