An FPGA Operating System to Allow Superior Implementation of High Throughput Event-Based DSP Algorithms

Bo Marr, Dan Thompson, Bill Noble, Jeff Caldwell, Ray Hsu
Raytheon Co.
Space and Airborne Systems
El Segundo, CA

***Abstract*— A light Operating System (OS) for an FPGA is presented. The key features allowing application isolation and abstraction are described and an event repetition interval (ERI) application that tracks multiple simultaneous events and establishes tracks in real time is presented as a use case; all of these differentiating this work from other works. The authors experienced about an order of magnitude speedup in design time compared to previous FPGA efforts with no OS.**

Keywords—FPGA, Operating System, Tracker, Abstraction, Event Repitition Interval

I. Higher Level Abstractions for FPGAs

As FPGA devices continue to grow in scale, methods for greater efficiency in design time of FPGAs become a dominant concern. Features allowed by the classical definition of an operating system – modularity and isolation of applications from each other, pre-built services, and some level of abstraction from low level hardware – are all necessary to achieve this. We show a software defined radio application use case with this OS concept.

A. Trendlines Supporting FPGA OS

Historical context shows we are at a unique time in the logic function capacity for FPGAs to implement such an idea, as shown in Figure 1.

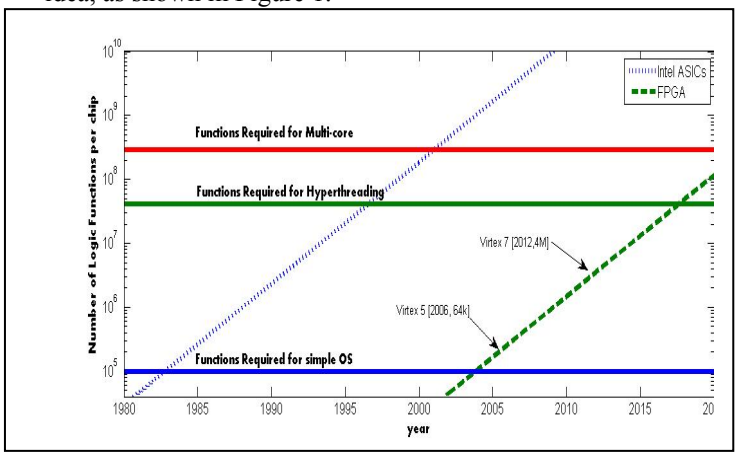

Figure 1. Trendline comparing intel processors and FPGAs in number of logic functions per year and functions needed for features like an OS.

Figure 1 shows FPGAs are about 7 years behind Intel processors in logic count. Logic functions were computed as gates on an Intel processor and LUTs on the FPGA. Interestingly FPGAs have recently crossed the logic threshold required for an operating system. Note the intel 8086 hosted approximately 29,000 logic functions in 1980 when the first widows operating system was installed [3] and the Xilinx Virtex 5 FX100T part hosts about 64,000 logic functions [4].

B. Critical Functions of an FPGA Operating System

Here we define three essential functions needed by an FPGA OS that are shared with a classical software OS.

- *Isolation of application programs from underlying hardware* – in this case by providing pre-defined access points for specific hardware I/O for example.
- *Interprocess Communications* –a general purpose data bus where any application or service can send data to any other application, and a point to point configurable path between applications .
- *Interprocess Isolation* – predefined program areas are provided into which an application program can be inserted. This isolation ensures that the actions of a program executing in one program area does not affect an independent program executing in another

Figure 2 shows the FPGA application abstraction layer as used in this work.

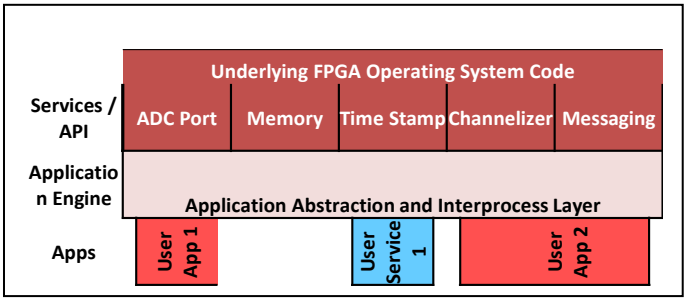

Figure 2. User application abstraction.

C. Related Works

The BORPH [1] OS allows user designs to run as normal UNIX processes in a system like a software program complete with a virtual file system. In Dataflow and DSP based applications, the associated latency and overhead is

unnecessary. Similarly in [2] the attempt is to allow software designers to more easily implement programs on FPGAs, whereas this OS concept is targeted at firmware designers. We believe it is functionality and interfaces, not firmware design, that should be abstracted. Most other works focus on aspects of dynamic reconfigurability.

II. IMPLEMENTATION

Several key components of our implementation are shown in Figure 3.

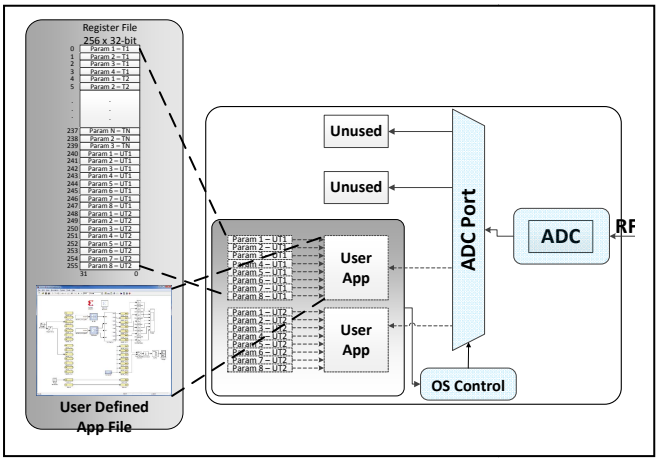

Figure 3. Example of user apps gaining ADC port access and a program space through an OS control point

The light OS concept presented here includes OS-controlled access points for I/O such as an ADC, inter-process communication through an OS control block, a pre-defined application space, and indirect memory access for applications that can be written to by other software that need not be of a pre-allocated size.

The partial reconfiguration tools were used with Xilinx ISE to create user application partitions as shown in Figure 4. This feature allows *hardware abstraction* such that the user only need simulate and close timing on the application itself and not the entire FPGA design. We find this integration step is the single most time consuming step in the FPGA design process which is mitigated with this approach. This also allows dynamic application loading during runtime.

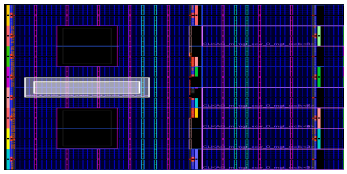

Figure 4. User app partition

A clock speed of > 200 MHz for the user application is used in this light OS design; > 200 MHz was found to be of sufficient throughput for a broad range of applications, but low enough to not cause timing errors assuming reasonable pipelining – abstracting a critical hardware trait from the designer and greatly speeding up design times.

III. USE CASE AND CONCLUSION

A novel event repetition interval (ERI) tracker application was created that simultaneously tracks multiple high rate (> 1 Giga Event per second) event sequences, computing 72 Giga Multiply Accumulates per second (GMAC/s). The problem is given a sequence of events $E_1, E_2, \ldots E_n$, that arrive at times $T_1, T_2, \ldots, T_n$ determine if any of the events form a periodic set and determine what the period is, defined as the ERI. With a *dynamic programming* substitution made, all time difference of arrivals (TDOAs), and thus any periodic set, can be calculated in real time from the current event time of arrival, T_I.. The algorithm was successfully compiled to our light OS and measured in hardware as shown in Figure 5. It concurrently ran with a waveform generation app compiled in a separate user space.

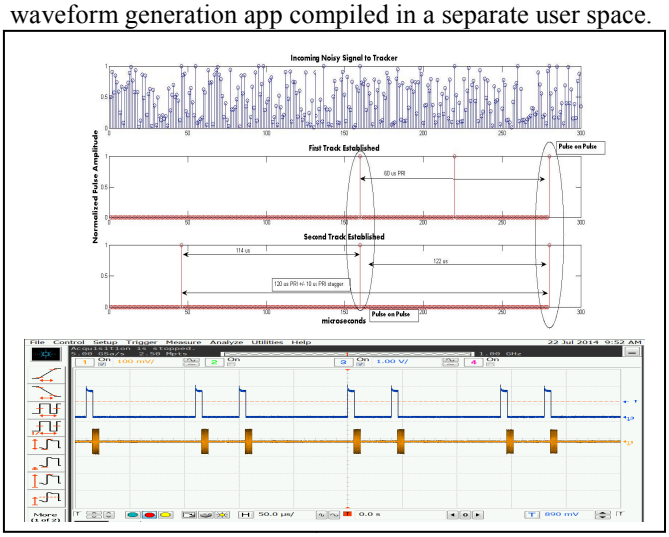

Figure 5. Simulated and hardware results of high rate event tracking app.

	V5 Resource Usage	V7 Resource Usage
Rf Synthesis and Analysis service	76%	7.6%
Rest of FPGA OS	5%	6%
Event Detector	10%	0.7%

Table 1. Utilization of OS and App

In conclusion, design time went from man-years to man-months for a moderately complex tracking algorithm compiled on top of a light OS presented here.

REFERENCES

[1] So, Hayden Kwok-Hay, and Robert Brodersen. "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH." *ACM Transactions on Embedded Computing Systems (TECS)* 7.2 (2008).

[2] Ismail, Aws, and Lesley Shannon. "FUSE: Front-end user framework for O/S abstraction of hardware accelerators." *Field-Programmable Custom Computing Machines (FCCM)*, 2011 IEEE 19th Annual International Symposium on. IEEE, 2011.

[3] Morse, Stephen P., William B. Pohlman, and Bruce W. Ravenel. "The Intel 8086 Microprocessor: a 16-bit Evolution of the 8080." Computer 11.6 (1978): 18-27.

[4] "Virtex 5 product family", http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/Virtex5_LX_Product_Table.pdf