Event Driven Data Processing Architecture

Ingemar Söderquist †
Saab AB, Saab Avitronics
SE-581 88 Linköping, Sweden
Email: ingemar.soderquist@saabgroup.com

Abstract

This paper describes a data processing architecture where events and time are in focus. This differs from traditional von Neumann and data flow architectures. New instruction codes are defined and special circuitry is introduced to express and execute event and time operations. This results in reconfigurable software controlled functionality together with real-time performance comparable to dedicated VLSI solutions. The architecture is demonstrated in a real-time radar jammer application. The architecture is promising also for applications as routers and network processors. A prototype system on silicon (SoC), complete with signal memory, instruction memory, four processing units in parallel and interfaces for digitized signals and host computer, is fabricated in 0.35 µm standard CMOS. Time events of signal data on two simultaneous 8-bit links can be programmed with a time resolution of one clock period. Measurements verified correct function and performance above 400 MHz clock frequency at 3.3 Volt supply. Power consumption is 3.6-Watt @320 MHz.

1. Introduction

The basic radar jammer function is to deny the radar to detect the real target and sometimes also to create non-existing targets. This is accomplished by recording incoming radar pulse at the real target, modify the pulse in time and frequency, and then retransmit the pulse back to the radar. The success to create non-existing targets depends strongly on the detection algorithm implemented in the radar, and on the pulse-to-pulse coherence attained by the jammer. The jammer digitizes incoming radar pulses, stores the pulse in memory for a short period, and retransmits time-delayed replicas. Each pulse in the chain of pulses must have exactly the same time-delay to be interpreted as coherent by the radar [5].

† The author is also with Electronic Devices, Linköping University, SE-581 83 Linköping, Sweden. Email: ingemar.soderquist@isy.liu.se

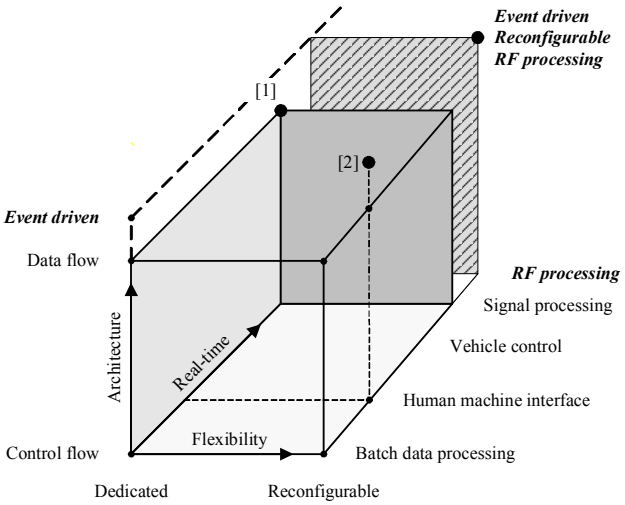

Fig 1. The three orthogonal axis encountered in high-speed system design.

The demand for systems with very high real-time performance and at the same time full programmability poses a significant challenge for software, architecture and circuit design of new computation platforms. At highest performance where missed deadlines cannot be tolerated, performance and functional verification before fabrication and life trial becomes a prohibitive design challenge. The generality of common program languages and associated architectures results in less support for precise time control of programmable functionality and hence results in performance far less than achievable with ordinary dedicated VLSI design. Dedicated full custom VLSI design will always have optimum outstanding performance. Its drawback is hard coded functionality and therefore not flexible enough for altering system specifications.

This becomes even more obvious when designing platforms for reconfigurable real-time RF applications where time constraints in the range of a single clock

978-3-9810801-2-4/DATE07 © 2007 EDAA

period are not unusual [5]. The rapid scaling of CMOS and the associated speed gain in digital computation have made digital solutions favorable compared to analogue. However regarding software performance we have not seen quite the same progress. Usually the desired system functionality is well modeled whereas desired time performance is not modeled at all. Exact the opposite holds for hardware part of the platform where low-level time performance is well predicted, but high-level system functionality is not visible at all. The consequence is well known by designers, the design process reaches a point where all high-level system functionality is implemented but system time performance is unknown. This lack of connection between high-level system functionality and low-level time performance has its origin in the early separation between the two, often introduced at system specification, and then kept all through the design process until final integration and system verification.

The graph in Fig. 1 is an attempt to classify digital systems based on architecture, flexibility and real-time performance. Different architectures here control flow, data flow and the proposed event driven, are placed on the Z-axis. Increased degree of flexibility in terms of programmability is placed on the X-axis, and increased real-time performance on the Y-axis. The main part of today's systems is found in the reconfigurable control flow area, with performance spanning different applications up close to the boundary given by signal processing. Highest performance is found for dedicated systems [1], using data flow architecture, point marked [1]. Event driven architecture is used in reconfigurable systems with lower performance [2], point marked [2]. The proposed architecture is aiming at event driven reconfigurable RF processing applications, upper right hand point in Fig. 1.

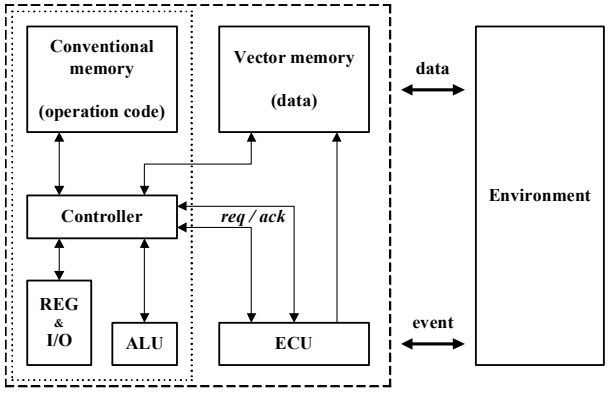

Fig 2. Von Neumann architecture (the dotted line) extended to event processing architecture suitable to interact with environment in real time RF applications.

2. Processing architecture

John von Neumann computing architecture is dominating since EDVAC historical development in the spring of 1945. The architecture is widespread and today used in standard microprocessors. After long extensive progress of hardware integration technology, cost effective implementations of new innovative computing architectures might constitute alternatives or complements to the von Neumann architecture. Progress is seen for e.g. massively parallel processor architectures, data flow architectures, associative architectures, neural net architectures, biological inspired architectures and dedicated as well as reconfigurable architectures [3].

This new architecture called event driven data processing architecture attempts to merge functionality and time performance closer during the entire design process [8]. Event-driven behavior is modeled as a chain of events with associated actions, where the word action is used to emphasis the symbiosis between the event and the achievement expected as a direct consequence of the event. Hopefully this will both reduce the total design effort and increase final design performance.

The approach is to strengthen the software means of expression to handle the event and the time, by adding a new type of machine code operation. The new operation is executed in an event-controlled unit (ECU) in a similar way as arithmetic is handled in an ALU. Fig.2 expands the traditional Von Neumann architecture with two major parts, Vector memory [4] and Event Control Unit (ECU).

At machine instruction level event and time constraints is captured in a pulse package (PPG) operation code, Fig. 3, with four operands:

1) ***Event*** that initiate execution.
2) ***Delay*** defines time interval to elapse between event and access of vector memory to capture or deliver data.
3) ***Vector*** defines location in vector memory.
4) ***Action*** defines future processing to be carried out on data on its way to/from vector memory.

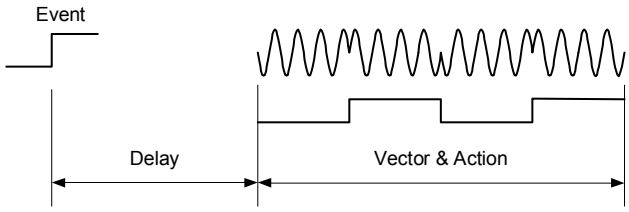

Fig 3. Function of one pulse package instruction, here action is passing parameters to phase modulation.

The ECU recognizes a new event at event input pin, or as the completion of the previous PPG instruction. Delay is expressed in number of clock periods. Vector consists of the number of bytes and the pointer to start position, where data is stored in vector memory. Action is parameters associated to data processing outside the processor (e.g. phase modulation).

At_Event_do *(event_1, delay_1, vector_1, action_1)*

Begin

Chain_Event_do *(previous_2, …)*

At_Event_do *(event_3, …)*

Variable assignments and arithmetic operations

while *(logical-expression)* **do**

Chain_Event_do *(previouse_4, …)*

Variable assignments and arithmetic operations

end_while

end
Loop event

Simplified High Language Level (HLL) code example above shows how event and arithmetical instructions can be mixed freely. Two HLL instructions, At_Event_do (external event) and Chain_Event_do (internal event) are introduced. The code execution is initially halted. When event_1 occurs the first command At_Event_do (event_1) executes instantly followed by execution of the command Chain_Event_do (previous_2), then execution halts. Later when event_3 occur the command At_Event_do (event_3) executes followed by variable assignments and arithmetic operations. Instantly followed by execution of the command Chain_Event_do (previous_4) and following variable assignments and arithmetic operations. The commands in the while-do statement are repeated in sequence. Finally the logical-expression fails and execution is halted, whereupon the whole sequence is repeated from the beginning. This is different compared to traditional software interrupt that suspend current executing code and jumps to another code portion to complete the specified request.

3. SoC implementation

Fig. 4 shows the event processor architecture part of the prototype SoC. The processor is placed in between two busses. Bus #1 for connection to the host processor and Bus #2 for passing parameters to the ECU's, to the underlying datapaths and to the external components. The modulator included in the output datapath is an example of an external component.

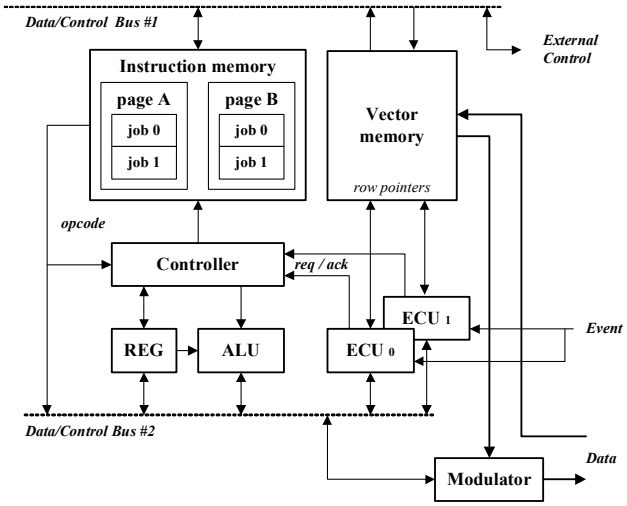

Fig 4. Block diagram of the event processor.

The controller with attached ALU and registers, manages execution of the operation codes needed for the four ECU's, one for the receiver function and one for the transmitter function and two additional ECU's for extended functions not shown in Fig.4. Operation codes for the two functions are stored in separate areas in a SRAM instruction memory, job 0 and job 1. Instruction memory has two pages, page A and page B, one page is being executed while another one is updated from Bus #1.

The vector memory is a true four-port SRAM, two ports are used for connection to the datapaths with simultaneous read and write, and the other two ports is used for signal analysis. The ECU includes a package controller (PAC), a combined controller and time counter, signals to interface to Vector memory, signals to control external devices (the modulator), and buffers to hold operands for next PPG instruction to facilitate momentary switch. PAC will request new operands from the overlaying controller using handshake signals. Execution of machine instructions in controller is halted or resumed depending on the handshake signals.

High programmable time delay resolution is obtained by splitting the implementation into two parts, shown in Fig. 5. One timer function in each ECU's package controller uses the most significant part of the local PPG instruction delay value. A second timer function in the output buffer connected to the vector memory is using the five least significant bits, passed from ECU to the high-speed controller (HSC). The HSC operates at the clock frequency and the timer function in ECU operates at the lower frequency, here clk/32. When the ECU executes an event operation, then after a time delay, corresponding to the most significant parts in the delay register elapsed, ECU initiates a vector read operation by activating the signal ECU_rd and pass the row pointer to Vector

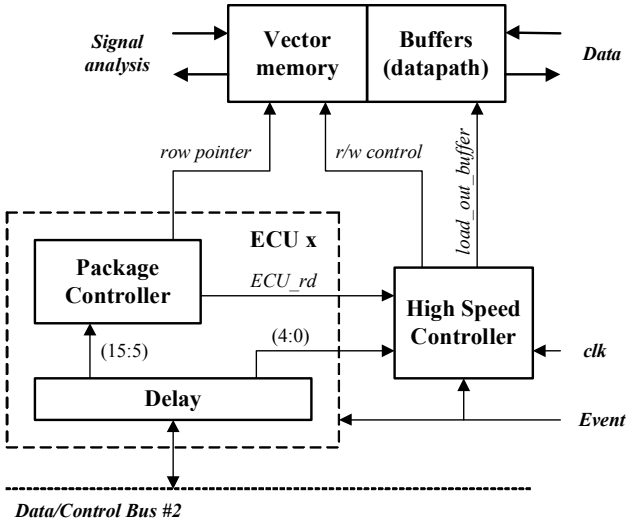

Fig 5. Programmable high-resolution time delay.

memory and the least significant part of delay register to HSC. HSC immediately creates r/w control signals and carry out the RAM access. Output data however is stored in a 32-byte buffer until the signal load_out_buffer becomes active, which is delayed a time corresponding to the five least significant bits. Write is handled in a similar way.

Fig. 6 shows a photograph of the prototype SoC. Central in the chip is input and output buffers and the four 1024x64 bit SRAM blocks for Vector memory placed. At right hand side; Instruction memory SRAM holding 512x32 bit instruction code, Modulator, Event Processor (EP) except ECU's, Measurement controller (MEC) used for RF signal analysis, Clock distribution block (CDB), Low speed controller (LSC) holding all four ECU's and HSC.

The event driven processor chip is designed for 320 MHz clock frequency using standard 3.3 volt 0.35 µm double poly CMOS process. Interface with two parallel 8-bit wide differential high-speed links for radar signals, based on low voltage pseudo ECL with internal 100-ohm termination, two control busses, and other signals gives all together 300 pads. The chip size is 88.3 mm^2, and it contains 2.5 million transistors. Finally the chip is mounted in a 256-pin BGA package.

4. SoC development and test environment

The need of specific development environment for the reconfigurable systems design was identified early. At application level a VHDL based system model is developed to capture and simulate the SoC functionality and performance in its natural environment. The model includes complete jammer functionality including A/D, D/A, host computer, and the environment. Each

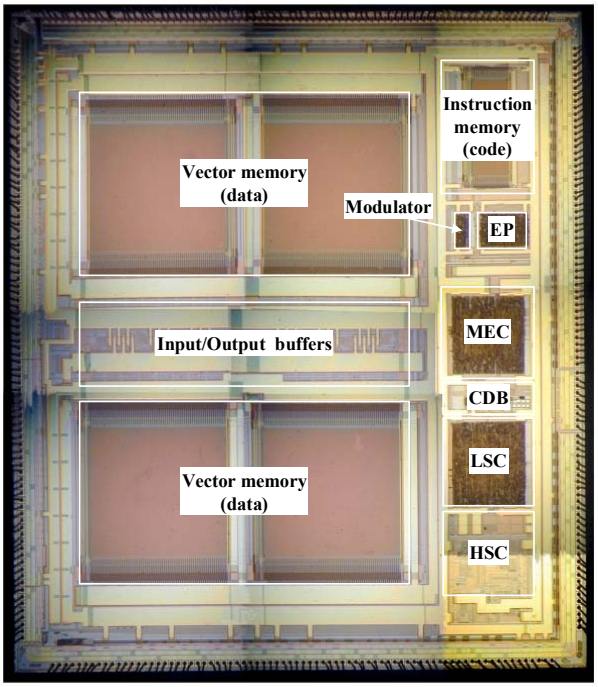

Fig 6. Photograph of prototype SoC.

simulation starts with a complete initiating sequence including the reset at power up and the download of compiled application software. A special compiler is developed to support HLL software development and debugging. The HLL is based on a subset of the C++ language.

Each block in the SoC represents a unique subsystem with well-defined functionality and its own optimal implementation method. The functional block partitioning and refinement methodology from [7] is used. Each block is translated from behavioral to RTL. Integrated and routed using the system model toplevel information for automated generation of the final netlist. Full custom dynamic TSPC logic [6] is used for high-speed I/O, input and output buffers, CDB and HSC. The modificator, EP, MEC and LSC including four ECU's is synthesized using a standard cell library. The memories were purchased as two intellectual property blocks.

Special considerations were taken with respect to verification. All the critical time constraint is verified by Spice simulation. Functional verification is performed at two levels. First each VHDL block is replaced by a netlist at transistor or cell level, and correct function is verified in the system model. Then the entire chip is replaced by a netlist and correct function is verified once again. 42 fabricated circuits were evaluated using same application software and functional tests as during verification prior to fabrication. The jammer was implemented on one single VME board and used as circuit tester, see Fig. 7.

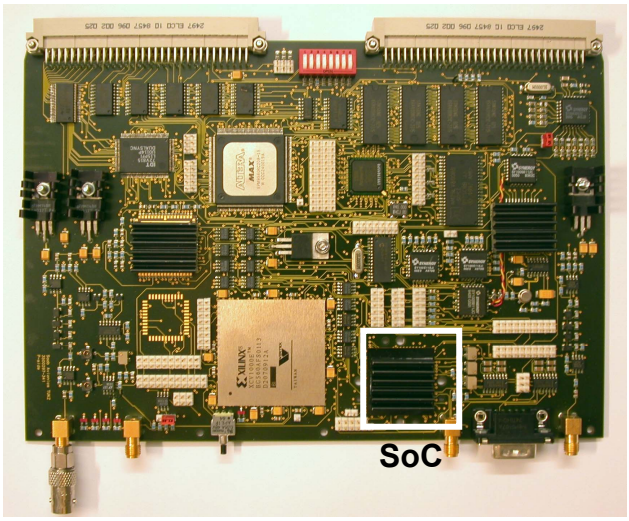

Fig 7. Photograph of jammer VME board.

A Power PC on the VME board replaces the host computer in the system model. Typical observed waveforms at VME board level are shown in Fig. 8. The programmable delay resolution down to one clock period is verified. The clock frequency for each circuit is increased until incorrect function is detected. 32 circuits passed the test, all with correct function above 400 MHz clock frequency at 3.3 Volt supply. Measured power consumption is 3.6 Watt at 320 MHz.

5. Conclusion

The success with correct function at first fabrication run is encouraging, not only for the new event driven architecture but also for the used integration and verification approach.

The reconfigurable event driven architectural handles the time constraints in a simple and elegant way and its performance can be predicted deterministically without any uncertainty caused by cache misses or interfering operative system. The prototype SoC demonstrates how programmable hard real-time actions with one clock cycle resolution can be reached. The event driven architecture can be scaled up to large-scale systems containing multiple processors without any time performance degradation by using global synchronization [7].

Acknowledgment

The author gratefully thanks Prof. Christer Svensson for his guidance. The design and test efforts of Anders Ödmark and Rolf Loh are greatly appreciated.

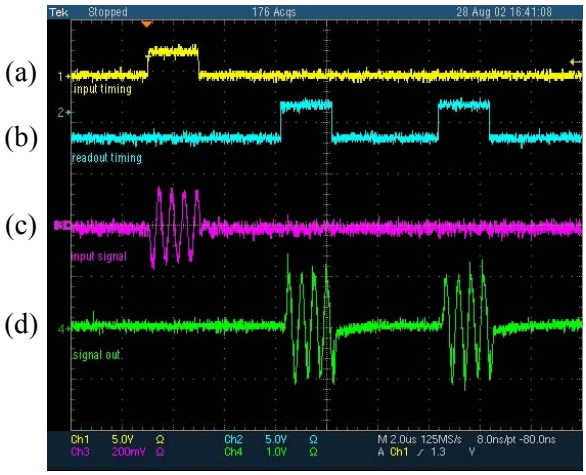

Fig 8. Functional test pattern from top the event signal (a), vector memory read signal ECU_rd (b), received RF signal (c) and transmitted RF signal (d).

References

[1] W. Rhett Davis et al., "A design environment for high-throughput low-power dedicated signal processing systems," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 420-431, March 2002.

[2] Chi-Hong Hwang and Allen C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. of 1997 IEEE/ACM International Conference on Computer-Aided Design*, San Jose, 1997, pp. 28–32.

[3] K.-E. Grosspietsch, "Unorthodox Computing Architectures," in *Proc. of 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Spain, pp 209, Jan. 2002

[4] Ingemar Söderquist, "Expandable High Throughput Vector Based Access Memory Architecture," in *Proc. of the 28th European Solid-State Circuit Conference*, Firenze, 2002, pp. 599-602.

[5] S. J. Roome, "Digital radio frequency memory", *Electronic & Communication Engineering Journal*, vol. 2, no. 4, pp. 147-53, Aug. 1990.

[6] Y. Ji-Ren, I. Karlsson, and C. Svensson, "A true single-phase-clock dynamic CMOS circuit technique," *IEEE J. Solid-State Circuits*, vol. 22, pp. 899-901, Oct. 1987.

[7] Ingemar Söderquist, "Globally updated mesochronous design style", *IEEE J. Solid-State Circuits*, vol. 38, pp. 1242-1249, July 2003.

[8] Ingemar Söderquist and Rolf Loh, "Digital signal processor", patent no. US2004128491, July 2004.