DESIGN OF A PROGRAMMABLE CONTROL SYSTEM

Andrei COZMA, Dan PITICA
Applied Electronics Department, Technical University of Cluj Napoca, Romania
E-mail: andrei.cozma@ael.utcluj.ro, dan.pitica@ael.utcluj.ro

Abstract: **This paper presents the design of a system intended to be used for automatic control applications. The system consists of a signal acquisition module, a programmable high speed Digital Signal Processing (DSP) core, a Microcontroller Unit (MCU), an output module and a communication module. Both analog and digital signals can be given as input to the system. The analog signals are digitized by a high speed analog to digital converter, while the digital signals are read through an I2C bus. The programmable DSP core can perform basic arithmetic operations, finite and infinite impulse response filtering and fuzzy logic operations at high speed and with a high degree of parallelism on the data received from the signal acquisition module. The MCU is a soft processor core with a RISC instruction set and it is used for controlling the operation of all the other modules and for implementing more complicated control algorithms that cannot be performed by the DSP core. The output module contains multiple Pulse Width Modulated (PWM) signal generators and high speed Digital to Analog Converters (DAC) for analog control signals generation from the digital control signals received from the MCU or DSP core. The communication module transfers data to/from a PC through a USB connection. By integrating all the above mentioned modules into one single chip a complete real time control core is provided.**

Keywords: *automatic control, microcontroller, digital signal processor*

I. INTRODUCTION

Nowadays in the cases where accurate control is needed Microcontroller Units (MCU) or Digital Signal Processors (DSP) are used to implement the control algorithm. Control dominated software functions are better suited to MCUs while DSPs are preferred for computation intensive signal processing tasks. This kind of control systems also provide great flexibility since they are programmable and can perform a variety of functions without modifying the hardware itself. Compared to analog systems, performing signal manipulation with digital systems has numerous advantages: systems provide predictable accuracy, they are not affected by component aging and operating environment, and they permit advanced operations which may be impractical or even impossible to realize with analog components. For example, complex adaptive filtering and error correction algorithms can only be implemented using digital signal processing techniques [1].

Modern control systems are implemented as Systems on Chip (SoC) and combine the advantages provided by the MCUs and DSPs. This kind of chip is a high-performance multiprocessor system which incorporates various types of hardware cores: programmable processors, Application Specific Integrated Circuit (ASIC) blocks, on-chip memories, peripherals, analog components, and various interface circuits [2]. Having the complete controller on a single chip allows the hardware design to be simple and very inexpensive [3].

The advances in CMOS technologies have enabled the development of complex systems on a chip by exploiting reusable programmable processor cores which are now characterized by low power consumption and a diminishing die area when compared to the size of the on-chip memories. These programmable processor cores help shorten the time to market for new system designs because they are already designed and verified. Typically embedded processor cores are delivered either in a soft or hard form. Soft cores are processor cores delivered as synthesizable Hardware Description Language (HDL) code and optimized synthesis scripts and thus they can quickly be retargeted to a new semiconductor technology. Hard cores, in turn, are designed for a certain semiconductor technology and delivered as transistor-level layouts, typically in the Graphic Data System II (GDSII) format. As opposed to soft cores, hard cores generally perform better in terms of die area and power consumption, however, when core portability is of primary concern, a soft core should be preferred. [2]

This paper introduces a programmable control system that can be used for a wide range of automation applications. The system combines the speed and signal processing power of a DSP with the flexibility given by a MCU thus providing an excellent environment for implementing control algorithms that can range from very simple ones like the classical Proportional Integral Derivative (PID) control, to fuzzy or hybrid control algorithms. Also more complicated control structures can

Manuscript received January 10, 2010; revised March 16, 2010

be implemented like cascade controllers or supervisory control. Both the DSP and MCU are programmable thus making the system highly flexible. The paper is organized as follows: Section II presents the system's functional requirements, Section III gives a general description of the system's architecture and presents the role of each of the system's building blocks, Section IV presents in detail the design and the instruction set of the programmable DSP core, Section V shows a comparison between the proposed system and existing solutions and Section VI presents conclusions.

II. FUNCTIONAL REQUIREMENTS

The programmable control system presented in this paper was designed taking into consideration the following functional requirements:

- Accept multiple input signals coming from various types of sensors. Both analog and digital sensors can be connected to the system. The digital sensors are connected trough an I2C bus. The signals coming from the analog sensors are digitized by an Analog to Digital Converter (ADC).
- Perform programmable signal processing algorithms on the input signals at high speed and with a high degree of parallelism.
- Execute PID, fuzzy or hybrid control algorithms. Implement more complicated control structures like cascade controllers and supervisory control.
- Transform the digital control signals into analog signals either by employing Pulse Width Modulation (PWM) or Digital to Analog Converters (DAC).
- The system can be configured, controlled and monitored from a PC through an USB communication link.

III. SYSTEM ARCHITECTURE

Fig. 1 presents the block diagram of the programmable control system with the connections and the data flow between the system's components.

A. Input Module

The input signals of the control system can be either analog or digital, depending on the type of used sensors. An *I2C Bus Controller* block reads data from multiple sensors connected to a common I2C bus. The I2C addresses of the sensors are configured by the MCU through the system bus, and the *I2C Bus Controller* reads the configured sensors in a serial fashion and stores the data read from each sensor in a separate 4x16 bit data buffer. A 16 bit ADC is used for digitizing the input analog signals. The analog signals are multiplexed at the ADC's input and the results of the analog to digital conversion are stored in separate data buffers by the means of a programmable demultiplexer. Both the multiplexer and demultiplexer can have two operation modes: automatic mode, when the input/output selection is done automatically based on a configuration programmed by the MCU; manual mode, when the input/output selection is controlled by the MCU.

B. Programmable DSP core

The programmable DSP core is able to perform a number of specialized data processing functions at high speed and with a high degree of parallelism on the data received from the *Input module*. Also control algorithms like PID, fuzzy or hybrid algorithms can be implemented. All the computations done by the DSP core are on 16 bit fixed point numbers. The data coming from the input module is fetched by the *Input Controller* block, from which the DSP core will extract the data when appropriate. The *Output Controller* block takes the data that was processed by the DSP core and passes it to the *Output module* or/and to the *Communication module*.

C. Programmable MCU

The programmable MCU is based on the Xilinx MicroBlaze soft processor core. It is used for controlling the operation of all the other modules and for implementing more complex control algorithms that

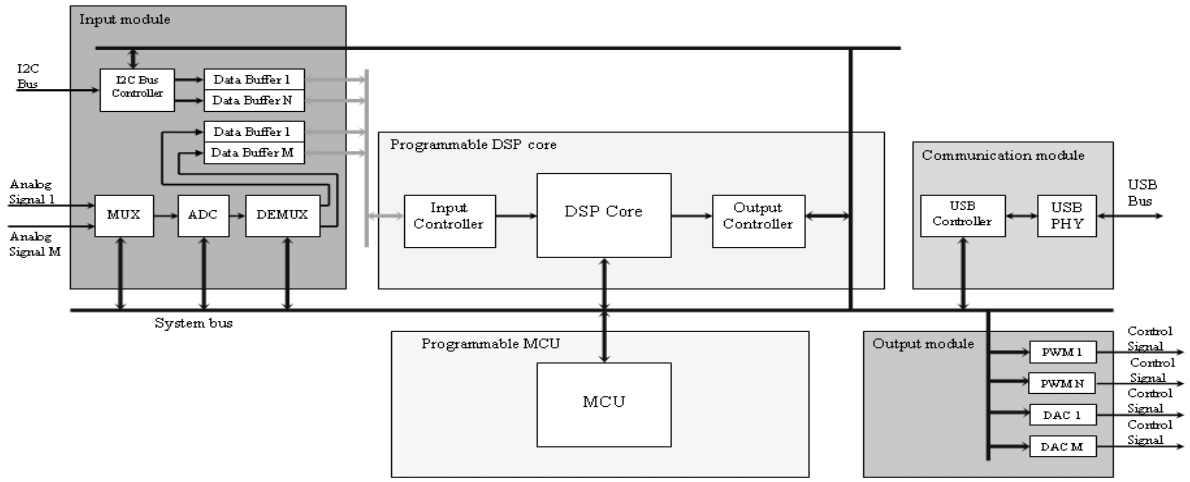

Figure. 1. Programmable control system block diagram

cannot be performed by the DSP core. It can be used for implementing supervisory control algorithms that perform corrections on the control parameters and can even reprogram the DSP core while it is running in order to achieve better control results.

The MicroBlaze embedded processor soft core is a reduced instruction set computer (RISC) optimized for implementation in Xilinx Field Programmable Gate Arrays (FPGAs) [5]. In terms of its instruction-set architecture, MicroBlaze is very similar to the RISC-based DLX architecture described in [4]. With few exceptions, the MicroBlaze can issue a new instruction every cycle, maintaining single-cycle throughput under most circumstances. MicroBlaze's primary I/O bus is the CoreConnect bus. User-defined coprocessors are supported through a dedicated FIFO-style connection called FSL (Fast Simplex Link). The coprocessor(s) interface can accelerate computationally intensive algorithms by offloading parts or the entirety of the computation to a user-designed hardware module [6]. The MicroBlaze soft core processor is highly configurable, allowing the selection a specific set of features required by a design. The fixed feature set of the processor includes: thirty-two 32-bit general purpose registers, 32-bit instruction word with three operands and two addressing modes, 32-bit address bus, single issue pipeline [5]. In addition to these fixed features many aspects of the MicroBlaze can be user configured: cache size, pipeline depth (3-stage or 5-stage), embedded peripherals, memory management unit, bus-interfaces, floating point operations. The performance optimized version expands the execution-pipeline to 5-stages, allowing a top operating frequency of 210 MHz [6]. Without the MMU, MicroBlaze is limited to running operating systems with a simplified protection and virtual memory-model: *e.g.*, μClinux and FreeRTOS. With the MMU, MicroBlaze is capable of hosting operating systems which require hardware-based paging and protection, such as the Linux kernel [6]. The advantage of using an open source soft core like the Xilinx MicroBlaze comes from the fact that it is provided as part of a embedded development kit that includes compilers and other libraries [7].

D. System bus

The system bus is implemented using MicroBlaze's primary I/O bus, the CoreConnect bus. This bus is an IBM-developed on-chip communications link that enables chip cores from multiple sources to be interconnected to create entire new chips. CoreConnect technology eases the integration and reuse of processor, system and peripheral cores within standard product platform designs to achieve overall greater system performance. The CoreConnect bus architecture includes the Processor Local Bus (PLB), the On-chip Peripheral Bus (OPB), a bus bridge, two arbiters, and a Device Control Register (DCR) bus [8]. High-performance peripherals connect to the high-bandwidth, low-latency PLB. Slower peripheral cores connect to the OPB, which reduces traffic on the PLB. There are bridging capabilities to the competing AMBA bus architecture allowing reuse of existing SoC-components.

E. Output module

This module converts the digital control signals received from either the MCU or the DSP core into analog control signals. It contains multiple PWM generators and a high speed DACs. The PWM generators have a 16 bit resolution and the frequencies of the output waves are programmable and can range between 1KHz to 10MHz.

F. Communication module

This module is able to send/receive data to/from a PC through an USB connection. The system sends to the outside world data related to the system's operation and receives configuration parameters and programs for the DSP core.

IV. PROGRAMMABLE DSP CORE DESIGN

The programmable DSP core can be divided into several functional units: input controller, data processing unit, data buffers controller and output controller.

In Fig. 2 is presented the block diagram of the programmable DSP core.

The digital input signals are stored in separate input buffers from where they are dispatched to the DSP blocks by the *Input controller*. A DSP block can access data from any input buffer, and multiple DSP blocks can use data from the buffer. A DSP block has two output buffers:

- a final output buffer from where the *Output controller* takes the data and sends if either to the *Communication module* or to the *Output module*
- an intermediate output buffer from where any other DSP block can take the data for further processing. The access to the intermediate output buffers is controlled by the *Data buffers controller* block.

The number of inputs and also the number of DSP blocks contained by the programmable DSP can be configured at synthesis time. The programs ran by the DSP blocks are loaded by the MCU into the internal memories of the DSP blocks either from a ROM memory or from a PC connected through USB to the system. The *Output Controller* receives the data output by the programmable DSP and to transfers it through the system bus to the *Output module* and to the *Communication module*. The operation of this block is programmed by the MCU.

A. Input controller

Data from each sensor connected to the system is stored in a 4x16 bit input buffer. The sampling frequency of each input buffer can be configured independently and can range from 10 KHz to 10 MHz. An input buffer acts as a FIFO. Each time data is read from the buffer the read sample is deleted from the buffer making room for a

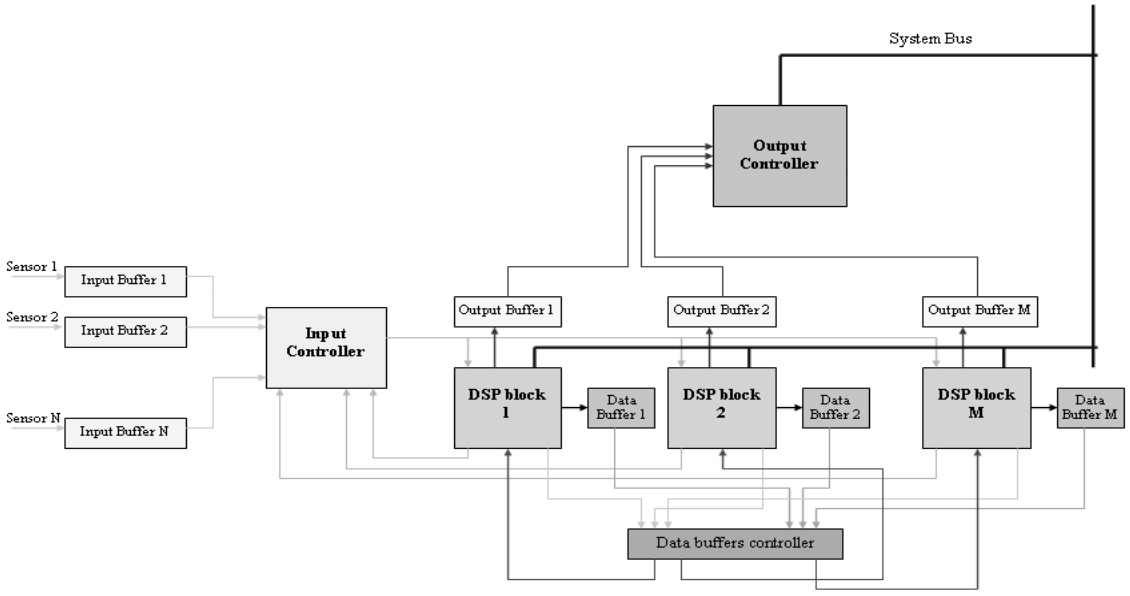

Figure 2. DSP core block diagram

new input sample. If the buffer isn't read fast enough by the data processing units the oldest samples are automatically discarded and new samples are pushed into the FIFO. The *Input controller* manages the data transfer between the input buffers and the data processing units. Each data processing unit has a dedicated control bus and data bus with the input controller. This allows the controller to service all the DSP blocks in one clock cycle. Each DSP block sends to the *Input controller* the number of the input buffer from which to get the data. If multiple DSP blocks require data from the same input it is possible to specify if they should receive the same sample, or different samples should be provided to each of them. The *Input controller* implements a priority scheme to help decide which DSP block should receive the input data in case of concurrent access. The priority scheme is based on the following rules:

- the DSP block with the lowest index has the highest priority in case multiple simultaneous requests are received
- the DSP block with the oldest un-serviced request has the highest priority

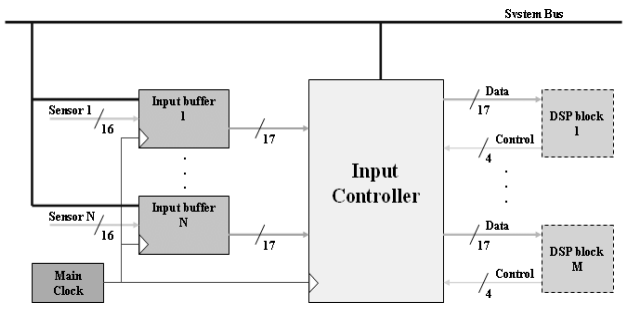

Figure. 3 Input unit

The samples received from the sensors are maximum 16 bits wide. The *Input controller* adds an extra bit to the sample bits to signal to the connected blocks if new data is available and the data retrieve request was successfully completed. Each DSP block sends a 4 bit control request to the Input controller with the following format:

- Bit 3 - 0 = get the same input samples as other blocks, 1 = get new input sample
- Bits 2:0 – the number of the input from which the sample should be retrieved

Having 3 bits to specify the input number limits the number of possible inputs to 8. Fig. 3 presents a detailed diagram with the interconnections between the input controller, the input buffers and DSP blocks.

B. Data processing unit

This unit is composed of multiple programmable DSP blocks. Each DSP block runs its own program and can take as input data from the connected sensors and data from other DSP blocks. The DSP blocks operate on 16 bit fixed point samples having 15 bits for the fractional part and one sign bit. Negative numbers are represented in 2's complement format. Thus the data range can be considered to be in the interval [-1.0, 1.0) with a resolution of $1/2^{15}$. A DSP block is able to perform the following types of operations:

- data transfers from input ports to internal memories and from internal memories to output ports
- data transfers between internal memories
- arithmetic operations as addition, subtraction, multiplication and division
- signal processing functions as finite impulse response (FIR) filtering and infinite impulse response (IIR) filtering
- fuzzy operations

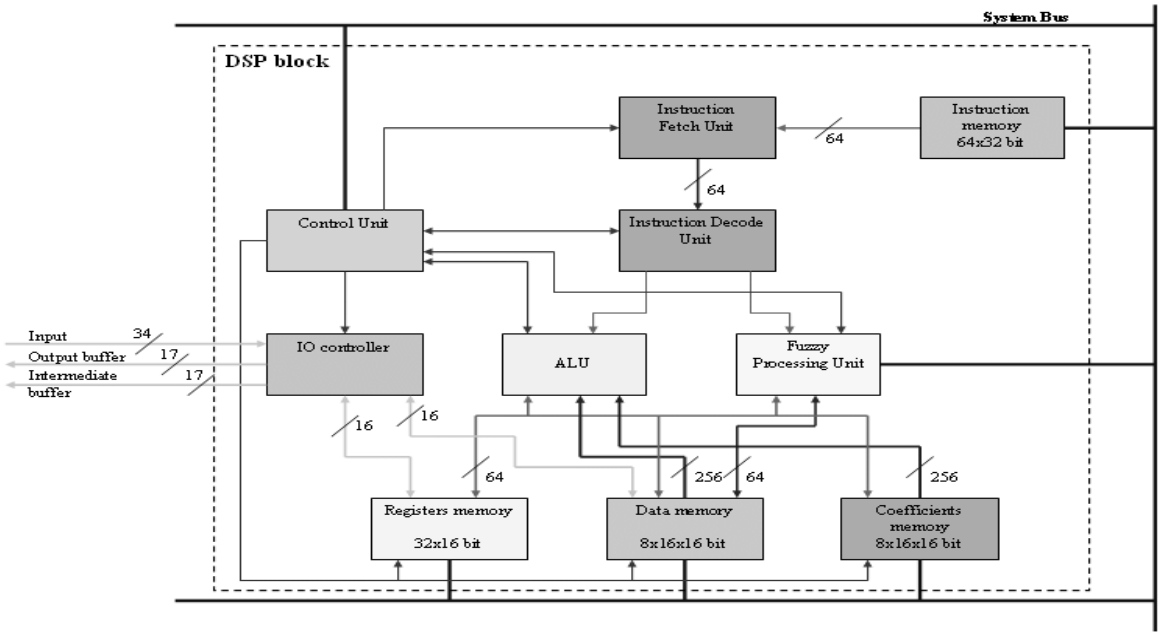

Figure. 4 DSP block

- minimum and maximum of two numbers
- conditional and jump instructions

Fig. 4 presents the structure of a DSP block and the data flow between the block's components.

The architecture of the DSP core is completely scalable and the number of DSP blocks that are incorporated in the DSP core can be selected at synthesis time. The program executed by a DSP block is loaded in the instruction, data and coefficients memories by the MCU through the system bus.

The instruction memory can store 64 instructions. Each instruction is 32 bits wide. A 4 stages pipeline is implemented for instruction execution. Up to 2 arithmetic instructions can be executed in parallel. The *Instruction Fetch Unit* fetches 2 instructions at time. If the 2 instructions are arithmetic instructions they will be scheduled for execution in parallel, and thus 2 instructions will be executed in one step.

Following there is a description of the instructions supported by a DSP block:

- NOP : no operation. It used only for program alignment so that arithmetic instructions that can be scheduled to be executed in parallel are always 64 bit aligned.

- **MOV[L]** *src, [rowNo], dst* : moves data from *src* to *dst. Src* and *dst* can be registers, data or coefficients memory locations. If the *L* flag is set than the *dst* must be a memory location and the *rowNo* operand must be specified. In this case the following *rowNo* rows following the row in which the data is moved to are shifted to the right by one and on the first position of each row is copied the last sample of the previous row. This is useful when computing filters longer than one memory row.

- **INP[L,S]** *portNo, [rowNo], dst* : moves data from an input port specified by *portNo* to *dst. Dst* can be a register or a data memory location. The *L* flag and the *rowNo* parameter have the same meaning as for the **MOV** instruction. The *S* flag specifies if the instruction should retrieve the same data as other concurrent **INP** instructions, or a new sample should be read. Port numbers between 0x00 and 0x0F are allocated for sensor inputs, and port numbers between 0x10 and 0x1F are allocated for intermediate buffers inputs.

- OUTP *src, portNo* : moves data from *src* to an output port specified by *portNo*. *Src* can be a register or a data memory location. Port number 0x00 specifies the default output, and port number 0x10 specifies the intermediate buffer outputs.

- MUL[U,S,M] *src1, src2, dst*: multiplies *src1* by *src2* and stores the result in *dst*. S*rc1*, *src2* and *dst* can be registers or data memory locations. If the *U* flag is set the operands are considered to be unsigned numbers. If the *S* flag is set the instruction must be followed by a **SCL** (scale) instruction that specifies the scale factor of the result. The *M* flag instructs the *Instruction Fetch Unit* that the next instruction is also an arithmetic instruction and should be scheduled for execution in the same step as the current instruction. This flag is set by the compiler during the code optimization stage.

- **DIV[U,S,M]** *src1, src2, dst*: divides *src1* by *src2* and stores the result in *dst*. *Src1*, *src2* and *dst* can be registers or data memory locations. The *U*, *S* and *M* flags have the same meaning as for the **MUL** instruction.

- **ADD[U,S,M]** *src1, src2, dst*: adds *src1* to *src2* and stores the result in *dst*. S*rc1*, *src2* and *dst* can be registers or data memory locations. The *U*, *S* and *M* flags have the same meaning as for the **MUL** instruction.

- **SUB[U,S,M]** *src1, src2, dst*: subtracts *src1* from *src2* and stores the result in *dst*. S*rc1*, *src2* and *dst* can be registers or data memory locations. The *U*, *S* and *M* flags have the same meaning as for the **MUL** instruction.

- **ABS[S,M]** *src, dst* : computes the absolute value of *src* and stores the result into *dst*. S*rc* and *dst* can be registers or data memory locations. The *S* and *M* flags have the same meaning as for the **MUL** instruction.

- **MIN[U,M]** *src1, src2, dst* : computes the minimum between *src1* and *src2* and stores the result in *dst*. S*rc1*, *src2* and *dst* can be registers or data memory locations. If the *U* flag is set the operands are considered to be unsigned numbers. The *M* flag has the same meaning as for the **MUL** instruction.

- **MAX[U,M]** *src1, src2, dst* : computes the maximum between *src1* and *src2* and stores the result in *dst*. S*rc1*, *src2* and *dst* can be registers or data memory locations. If the *U* flag is set the operands are considered to be unsigned numbers. The *M* flag has the same meaning as for the **MUL** instruction.

- **FIR[U,S]** *src, dst* : computes the result of a finite impulse response filter using the data and the coefficients located in the data and coefficients memories at row *src* and stores the result in a register specified by *dst*. A row of memory contains 16 samples. If filters with a length smaller than 16 are to be computed than the unused coefficients positions must be filled with 0. If filters with more than 16 taps are to be compute than the filtering operation can be performed in sequential steps using consecutive memory rows, and the intermediate results added in the end to obtain the final result. The *S* and *U* flags have the same meaning as for the **MUL** instruction. The following equation is used to describe the instruction's operation:

$$y(kT) = \sum_{i=0}^{N-1} u((k-i)T) * c(i) \qquad (1)$$

where *T* is the sampling period, *y(kT)* is the output of the filter and moment *t = kT*, *u(kT)* is the input of the filter at the moment *t = kT* and *c* is the filter's taps array with length *N*.

- **IIR[U,S]** *src, dst* : computes the result of an infinite impulse response filter using the data and the coefficients located in the data and coefficients memories at rows *src* and *src* + 1 and stores the result in a register specified by *dst*. A row of memory contains 16 samples. If filters with a length smaller than 16 are to be computed than the unused coefficients positions must be filled with 0. Filters with more than 16 taps cannot be computed. The current output value is automatically shifted in the corresponding location from the data memory to be used in the next filtering operation. The *S* and *U* flags have the same meaning as for the **MUL** instruction. The following equation is used to describe the instruction's operation:

$$y(kT) = \sum_{i=0}^{N-1} u((k-i)T) * cu(i) - \sum_{j=0}^{M-1} y((k-j-1)T) * cy(i) \qquad (2)$$

where *T* is the sampling period, *y(kT)* is the output of the filter and moment *t = kT*, *u(kT)* is the input of the filter at the moment *t = kT* , *cu* is the filter's forward taps array having the length *N* and *cy* is the filter's feedback taps array having the length *M*.

- **IIR1[U,S]** *src,coeff,dst* : computes a single pole infinite response filter using the current input specified by *src*, the previous output located at *src* + 1 and the coefficient given by *coeff*. *Src1*, *coeff* and *dst* can be registers. The *S* and *U* flags have the same meaning as for the **MUL** instruction. The following equation is used to describe the instruction's operation:

$$y(kT) = \alpha * u(kT) + (1-\alpha) * y((k-1)T) \qquad (3)$$

where *T* is the sampling period, *y(kT)* is the output of the filter and moment *t = kT*, *u(kT)* is the input of the filter at the moment *t = kT* and α is the filter's coefficient.

- **FUZZY** *src1, src2, dst* : computes the output of a fuzzy controller with 2 input variables specified by *src1* and *src2* and stores the result in *dst*. The input variables are fuzzyfied by the *Fuzzy Processing Unit* according to equation (4), and afterwards all the min and max combinations between the input variables are computed.

$$mu(i) = 1 - \frac{c(i) - val}{d(i)} \qquad (4)$$

Where *mu(i)* is the membership function of the input variable having the value *val* to the the ith fuzzy set, *c(i)* is the center value of the fuzzy set i and *d(i)* is the width of the fuzzy set i. Based on a pre-programmed set of rules the *Fuzzy processing unit* loads the useful min and max computations into data memory, and the output is computed by the *ALU* using two **FIR** operations and a division according to the centroid defuzzyfication method described by equation (5).

$$y = \frac{\sum_{i=1}^{N} mu(i) * out(i)}{\sum_{j=1}^{N} mu(j)} \qquad (5)$$

where *y* is the output of the fuzzy controller, *mu(i)* is the truth value of rule *i*, *out(i)* is the output of rule *i* and *N* represents the number of rules.

- **CMPJNE** *src1, src2, addr* : tests the equality between *src1* and *src2*, and if they aren't equal sets the program counter to *addr*. s*rc1* and *src2* are registers and *addr* is an immediate value.

- **CMPJLE / CMPUJLE** *src1, src2, addr* : tests if *src1* is less than *src2*. If the condition is true sets the program counter to *addr*. s*rc1* and *src2* are registers and *addr* is an immediate value. If the *U* flag is set the operands are considered to be unsigned numbers.

- **CMPJGE / CMPUJGE** *src1, src2, addr* : tests if *src1* is greater than *src2*. If the condition is true sets the

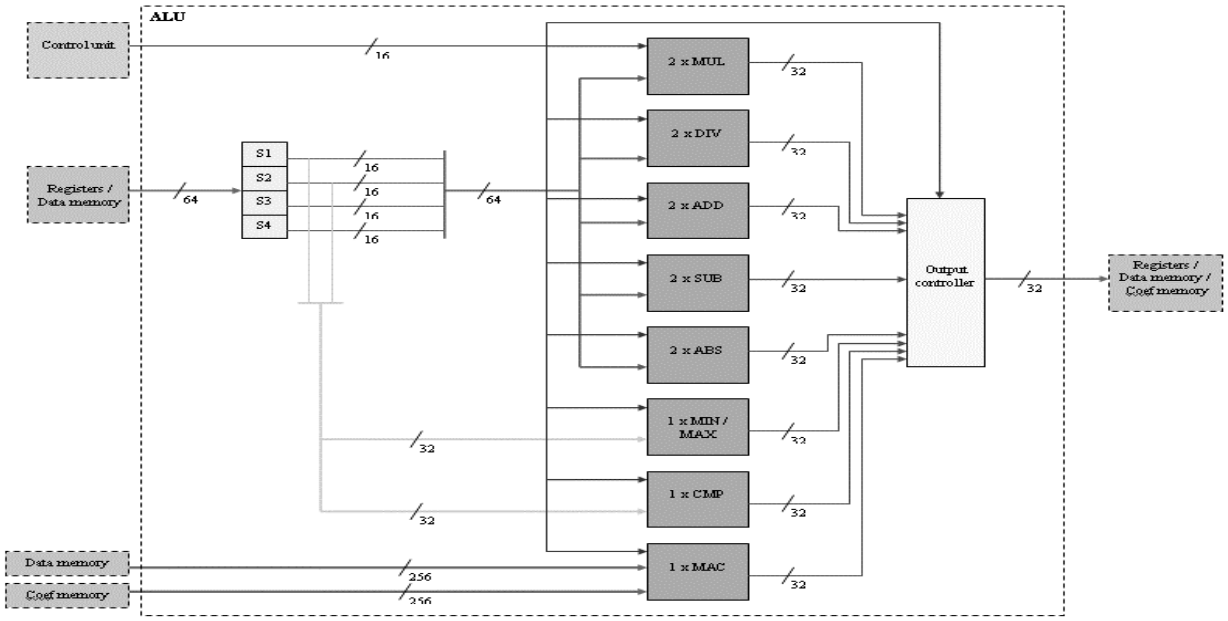

Figure. 5 ALU block diagram

program counter to *addr*. S*rc1* and *src2* are registers and *addr* is an immediate value. If the *U* flag is set the operands are considered to be unsigned numbers.

- **SCL** *val* : Scales the result of the previous instruction with a number equal to *val*, where *val* is a power of 2.

The *Registers memory* is interfaced with the *ALU* and the *Fuzzy Processing Unit* by a 64 bit bus, allowing for four 16 bit words to be transferred in one clock cycle.

The filtering instructions operate directly with the data and coefficients memories. These memories have 256 bits data busses, which allow the transfer towards the *ALU* of sixteen 16 bit words in one clock cycle. The *ALU* is capable of performing 2 arithmetic operations in parallel. It contains the following processing blocks: 2 multiplication units, 2 division units, 2 addition units, 2 subtraction units, 2 absolute value units, 1 minimum / maximum computation unit, 1 comparator unit, 1 MAC unit with 32 inputs for filtering operations. Fig. 5 presents the block diagram of the ALU.

It is estimated that each instruction will take 4 cycles to execute, but it has to be seen during detailed system design and the implementation stages if optimizations can be made to reduce the cycle count for some instructions.

An assembler written in the Tool Command Language (TCL) language will be used for generating the machine code of the programs targeted for the DSP core.

C. Data buffers controller

Each DSP block communicates with an intermediate buffer to store the results that are to be used by other DSP blocks. These buffers are 4x16bit memory blocks and their operation is similar to that of the input buffers. The *Data buffers controller* manages the access of the DSP blocks to the intermediate buffers. Each DSP block has dedicated data and control lines with the controller in order for it to be able to service all the DSP blocks in one clock cycle.

D. Output controller

This module transfers data from the programmable DSP to the *Communication module* and the *Output module*. It samples the output buffers on each clock cycle to check if new data is available. For each input from the programmable DSP the *Output controller* implements an 8 bit register with the following structure:

- Bits 7:6 – 00 = input not connected, 01 = send data to *Communication module*, 10 – send data to *Output module*

- Bits 5:0 – number of the output port from the *Output module* to which data must be sent / header to be added to the sample in case it must be sent to the *Communication module*.

The configuration registers are accessible to the MCU through the system bus and can be configured every tine the DSP core is programmed.

V. COMPARISON WITH EXISTING SOLUTIONS

The presented system has many similarities with the existing Digital Signal Controllers (DSC) in terms of operation and targeted applications, but it also brings a set of new features that are useful for implementing complex control algorithms.

A DSC can be thought of as a hybrid of microcontrollers and digital signal processors. Like microcontrollers, DSCs have fast interrupt responses, offer control-oriented peripherals like PWMs and watchdog timers, and are usually programmed using the C programming language, although can be programmed using the device's native assembly language. On the DSP side, they incorporate features found on most DSPs such as single-cycle multiply-accumulate (MAC) units, barrel

shifters, and large accumulators. DSCs are used in a wide range of applications, but the majorities go into motor control, power conversion, and sensor processing applications. [9] [10]

Similar to a DSC the presented system combines the features of a DSP and a microcontroller, but on the DSP side it incorporates extra capabilities like finite and infinite response filters, a fuzzy processing unit and a logical unit for executing conditional instructions. These features allow the DSP not only to process the input signals but also to execute complex real time control algorithms at higher sampling rates. The microcontroller incorporated into the design can be programmed using the C programming language, while the programs for the DSP core are written using the assembly instructions presented in Chapter IV.

All DSPs are based on a Harvard architecture with separate busses for data and instructions. This allows both data and instructions to be fetched simultaneously and greatly increases throughput. The DSP core described in this paper takes this concept further and uses separate busses for instructions, IO data and internal data memories. The data is stored in three separate memories which can be accessed simultaneously, thus reducing the execution time for filtering operations.

Like all DSPs the presented DSP core has an instruction pipeline that allows more than one instruction to be executed at one time. As an improvement the arithmetic unit is able to execute two arithmetic instructions in parallel, and also the instruction fetch unit and instruction decode unit are optimized to fetch / decode two instructions at a time. If the decoded instructions are arithmetic instructions they will be executed in parallel.

VI. CONCLUSIONS

In this paper was presented the design of a programmable digital control system. The purpose of the design is to provide a complete real time control solution that can be used to control the operation of various types of systems, starting from simple ones to complex multivariable systems. By combining the advantages of a programmable DSP with those of a MCU high performance control algorithms can be implemented.

A programmable DSP core was introduced, which is to be used for real time processing of the input signals and also for real time execution of control algorithms. The architecture of the DSP core is completely scalable and the number of DSP blocks that are incorporated in the DSP core can be selected at synthesis time. The instruction set of the DSP contains arithmetic, signal processing and conditional instructions. A fuzzy unit was incorporated in the DSP core to allow the execution of fuzzy operations on the input signals.

The design of the system is oriented towards flexibility and scalability. Processing blocks as well as inputs and outputs can be easily added or removed from the system to suit the particular needs of various applications.

As a future improvement an image processing block can be added to the system in order to give the ability to perform real time image processing operations. This feature will provide the possibility to use the proposed system in control applications for autonomous robots, or even vehicles, which employ control algorithms based on data from different types of sensors and also on data from video cameras.

REFERENCES

[1] E. C. Ifeachor and B. W. Jervis, “Digital Signal Processing: A Practical Approach”, Addison Wesley Longman, Inc., Menlo Park, CA, U.S.A., 1993
[2] Mika Kuulsa, “DSP Processor Core-Based Wireless System Design”, PhD Thesis, 18th of August 2000
[3] “Modern Control Technologies: Components and systems, Second edition”, Thomson Delmar Learning, 2001
[4]David A. Patterson, John L. Hennessy, “Computer organization and design: the hardware/software interface”, Morgan Kaufmann Publishers, 2005
[5] “Microblaze Processor Reference Guide”, Xilinx, 2008
[6] http://en.wikipedia.org/wiki/MicroBlaze, 2009
[7] Peter Wilson, “Design Recipes for FPGAs”, Newnes, 2007
[8] www.xilinx.com/products/ipcenter/dr_pcentral_coreconnect.htm, 2009
[9] http://en.wikipedia.org/wiki/Digital_signal_controller, 2009
[10] Ross Bannatyne, “The evolution of the digital signal controller”, Motorola Semiconductor Products, 2009