# A High-Performance Processor for Embedded Real-Time Control

René Cumplido, Simon Jones, Roger M. Goodall, and Stephen Bateman

*Abstract*—**This brief reports on an algorithm and corresponding processor architecture for the construction of high-performance processors targeted at linear time invariant (LTI) control. The overall approach involves reformulating the controller into a particular discrete state–space representation, which is optimized for numerical efficiency using the $\delta$ operator, then programming this into a specially-designed control system processor (CSP) implemented using a "programmable ASIC" device. This architecture presents large cost and performance benefits for control applications over traditional architectures, particularly for large multiple-input–multiple-output (MIMO) controllers. Results of implementing control of the vertical modes of a Maglev vehicle are presented and compared with implementations using commercial processors.**

*Index Terms*—**Algorithms, computer architecture, control systems, linear systems, multiple-input–multiple-output (MIMO) systems.**

## I. Introduction

THERE ARE now a variety of control design methods by which appropriate control laws can be created for complex multivariable systems, but the actual implementation of control laws is a part of the design process which most control engineers want to achieve as straightforwardly and transparently as possible. One approach is to program a fixed-point microprocessor device in a high-level language, using floating-point variables so that numerical problems are not a concern. Unfortunately this represents a costly solution and also creates a slow controller due to the large number of instructions needed to perform the calculations using short data word length. Indeed, our view is that this approach is both inelegant and inefficient and arises from the lack of a clear understanding of the numerical requirements of controllers and a corresponding absence of hardware to provide tailored support for such computational operations. A second approach is to use floating-point digital signal processors (DSPs) which provide a large numeric range that can handle signals of widely varying magnitudes. However, the signal range requirements are usually modest in well-designed digital control algorithms. Rather it is the accuracy of the calculations that is important.

The difficulty is that there are specific numerical requirements in control system processing for which standard processor devices are not well suited, in particular arising from the high sample rates needed to avoid adverse effects of sample delays upon stability margins and controller performance. These could be satisfied in either microprocessor or DSP devices using 'hand-crafted' numerical routines, probably written in assembler language, but, as mentioned previously, control engineers generally have neither the will nor the skill to do this. There is therefore a clear need to understand the numerical requirements properly, to identify optimized forms for implementing control laws, and to translate these into efficient processor architectures.

The contribution of this brief is an approach that both identifies an efficient, numerically robust algorithm for high-speed linear time invariant (LTI) control, and develops a corresponding processing architecture. Bringing together control engineering and electronic system design skills has produced a highly efficient method. The end result is demonstrated using an Actel's ProASIC, which is a fine-grained low-power reprogrammable application specific integrated circuit (ASIC) device [in many respects similar to a field programmable gate array (FPGA)], although any form of programmable, semi- or full-custom silicon device could be used.

This approach needs to be contrasted with toolsets that are available to transfer controller designs from Simulink into FPGAs—these provide an easy conversion into the programmable device, but have not addressed the algorithmic aspects that are a key part of an optimized implementation. Furthermore, such algorithmic compilation techniques require a re-execution of circuit partitioning, placing and routing activities, as well as revalidation using application-specific test benches. This is in contrast to a processor where simulation of the new program is the only required step. The approach also needs to be compared with the range of work on direct FPGA implementation of infinite impulse response (IIR) filters [1]. The work described in this brief differs in two ways: first there are special requirements to ensure robust numerical operation with the high-complexity high sample rate filters commonly required for control, and second the objective is a highly efficient processor core which can be programmed for a wide variety of control applications.

There are many common aspects between processing for control and digital filter implementations for signal processing in telecommunications, however there are some important

Manuscript received December 21, 2001; revised October 1, 2003. Manuscript received in final form July 23, 2004. Recommended by Associate Editor D. Gorinevsky. This work was supported in part by the National Council for Science and Technology (CONACYT) of Mexico under Grant 71530 and in part by the Actel Corporation.

R. Cumplido was with the Department of Electronic and Electrical Engineering, Loughborough University, Loughborough LE11 3TU, U.K. He is now with the National Institute for Astrophysics, Optics, and Electronics, 72840 Puebla, Mexico (e-mail: rcumplido@inaoep.mx).

S. Jones was with the Department of Electronic and Electrical Engineering, Loughborough University, Loughborough LE11 3TU, U.K. He is now with the Media Laboratory Europe, Dublin 8, Ireland (e-mail: simon.jones@medialabeurope.org).

R. M. Goodall is with the Department of Electronic and Electrical Engineering, Loughborough University, Loughborough LE11 3TU, U.K. (e-mail: r.m.goodall@lboro.ac.uk).

S. Bateman was with the Actel Corporation, Mountain View, CA 94043 USA. He is now with Chip Express, Santa Clara, CA 95054 USA (e-mail: sbateman@chipx.com).

Digital Object Identifier 10.1109/TCST.2004.839579

differences that make the approach described in this brief not suitable for applications in telecommunications. First, finite impulse response (FIR) filter formulations which are frequently used in telecommunications are never appropriate for control because of the larger phase lags introduced compared with IIR filters, although sometimes this is also a problem in telecommunications. Second, the high sample rates that are necessary for control are very rarely, if ever, required in telecommunications. Third, the multiple-input–multiple-output (MIMO) formulation, often combined with high-dynamic complexity, is almost entirely a special requirement for control processing that leads to an emphasis upon state–space rather than transfer function formulations.

The remainder of this brief is structured as follows. Section II describe some related previous work and motivation of this work. Section III describes the controller formulation used to implement the control system processor (CSP) and the associated numerical issues. Section IV describes the CSP architecture and its multiply-accumulator unit. Section V details the CSP instruction set, program structure and software suite. Section VI presents evaluation results of implementing a complex controller using the CSP. Section VII describes future work and finally Section VIII concludes.

## II. Previous Work and Motivation

In the past there has been much work on architectures for control applications, they can be classified into three groups according to the approach used in their design. First, general-purpose solutions in which the main goal is to provide a general solution for control systems. Microcontrollers and some DSPs targeted for control evolved from this same idea and have so far achieved great commercial success. Based also on the general-purpose approach, Furber *et al.* describe an asynchronous controller for small embedded systems (AMULET2e) which is based on a 32-b asynchronous processor core with pipelined cache, memory interface, and assorted programmable control functions [2]. When using a general-purpose processor, the control algorithm has to be artificially partitioned and constrained to meet the physical bus widths and mapped on the instruction set, any parallelism inherent in the algorithm will be lost when it is translated into the serial code performed by the processor. Also, in most cases the flexibility provided by the processor is not needed to implement many digital control applications, for any given clock cycle only a small portion of the logic elements on the device may be doing something associated with the control process. Furthermore, the execution time for control system software can be difficult to predict beforehand due to software complexity and resources like cache memory, pipelining, interruptions, etc.

Second, a number of dedicated solutions that are optimized to solve a specific task have been proposed. Garbergs considers the use of an ASIC for a stand-alone controller that estimates process-states and controls the dynamic process [3]. Costa proposes an architecture dedicated mainly to medium-range applications that demand computational power combined with low cost for the resulting hardware system [4]. Liu *et al.* proposes an integrated solution to compute real-time robot control using a special-purpose very large scale integration (VLSI) array [5]. Agrawal *et al.* presents a system design of an industrial controller that can be customized for specific tasks where the customized functionality is achieved in software and then ported into the controller [6]. These designs provided good performance results at the expense of flexibility. However, the cost of producing custom silicon proved prohibitive for initial exploitation and restricted experimentation with different architectural constructs.

Finally, some configurable solutions that are based mainly in parallel computation have also been proposed. Spray and Jones propose a programmable adaptive computing engine (PACE), which is a medium-grained cellular automation-based architecture that supports regularly and irregularly structured functions within a regularly structured array [7]. Tsunekawa and Miura describe a direct approach to the solution of state–space equations. It is based on multiple processing elements (PEs), with each PE being a single chip, that perform the calculations for one output or next state variable [8]. A similar design was proposed by Fujioka *et al.*, they describe a single chip PE, consisting of two multipliers, two adders, local memory, and a reconfigurable switch circuit, that are combined to perform several multiply-accumulate operations per cycle [9].

It is the author's opinion that no real attempt has been made to adopt a system approach involving control and electronics design requirements. During the design of the CSP requirements of both areas were taken into account. The resulting CSP architecture does not fall directly into the approaches described previously; as it is based on a simplified datapath similar to those found on general purpose processor, however, the datapath has been optimized to perform the operations needed to implement LTI control when using a formulation based on the $\delta$ operator. Our view is that by considering the numerical and computational requirements of the algorithm and through architectural exploration of hardware structures to support these numerical requirements, we can develop a design method and processor which satisfies high-speed applications at a modest cost. Such systems are likely to be smaller, cheaper, faster, and lower power than conventional signal processors. This approach is feasible due mainly for the availability of high-level design tools such as logic synthesis CAD suites allied to large low-cost FPGA. Additionally, for high-volume products, special-purpose processors may also be less expensive as only those functions needed by the application are implemented. Implementations using special-purpose processors (the CSP for example) and general-purpose processors are not mutually exclusive. In fact, they may be integrated to provide a more complex solution in which the CSP performs the computational demanding operations and the general-purpose processor performs the additional functions needed to implement real-time digital control.

## III. Controller Formulation

### A. Discrete Form and $\delta$ Operator

The discrete state–space form shown in (1) is a natural way of expressing the implementation equations for complex controllers, and for this reason it is used as a basis in this brief. Transforming other expressions, e.g., discrete transfer functions

or continuous expressions, into the following form is relatively straightforward [10]:

$$\begin{aligned}\mathbf{X}_{k+1} &= \mathbf{A}\mathbf{X}_k + \mathbf{B}\mathbf{U}_k \\ \mathbf{Y}_k &= \mathbf{C}\mathbf{X}_k + \mathbf{D}\mathbf{U}_k\end{aligned} \tag{1}$$

where $\mathbf{X}_k$ is the internal state vector of the controller, $\mathbf{Y}_k$ is the output vector of the controller, and $\mathbf{U}_k$ is the controller input vector (i.e., the measurements) at the $k$th sample instant. $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and $\mathbf{D}$ are matrices of coefficients that describe the controller. This general description can implement any formulation, whether based upon the $z$ or the $\delta$ operator, with the particular formulation resulting in an identifiable structure in the $\mathbf{A}$ matrix.

The $z$-operator is the most commonly used in the literature and is the traditional choice for many engineers. The $z$-operator is defined as

$$zx(k) \triangleq x(k+1) \tag{2}$$

whereas the delta operator is either defined as

$$\delta x(k) \triangleq \frac{x(k+1) - x(k)}{T} \tag{3}$$

or as

$$\delta x(k) \triangleq x(k+1) - x(k) \tag{4}$$

where $T$ is the sample period [11], [12].

From (2) and (3), it can be seen the relation between both operators

$$\delta = \frac{z-1}{T} \quad \text{or} \quad z = \delta T + 1. \tag{5}$$

Similarly, if the definition in (2) and (4) are used, the relation between both operators is

$$\delta = z - 1 \quad \text{or} \quad z = \delta + 1. \tag{6}$$

Thus, any system expressed in terms of $z$ can be converted to a model in $\delta$ and vice versa [11]. In this brief, we use the simpler definition of the $\delta$ operator given in (4), which is relevant when the focus is upon implementation [11]. Both definitions of the $\delta$ operator give normal sensitivity, i.e., the percentage accuracy required for the coefficients is the same as that required for the overall characteristics of the control law, typically 5% [13]. This is in contrast to the $z$ operator formulations, for which the coefficients often need to be hundreds of times as accurate [10], [11].

### B. Controller Structure and Implementation

The generalized controller structure based upon the $\delta$ operator is illustrated diagrammatically in Fig. 1. For real-time processing the operation that must be implemented is $\delta^{-1}$, just as

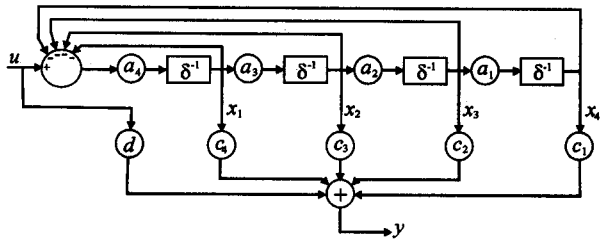


Fig. 1. Formulation used to implement the CSP.

$z^{-1}$ is implemented in normal formulations. The operation involves a shift and add, instead of just the shift that is required for the $z$ form - this can be seen in the following:

$$\begin{aligned}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_{k+1} &= \begin{bmatrix} 1-a_1 & -a_1 & -a_1 & -a_1 \\ a_2 & 1 & 0 & 0 \\ 0 & a_3 & 1 & 0 \\ 0 & 0 & a_4 & 1 \end{bmatrix} \\ &\quad \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_k + \begin{bmatrix} a_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} u_k \\ y_k &= \begin{bmatrix} c_1 & c_2 & c_3 & c_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}_k \\ &\quad + [d_1] u_k.\end{aligned} \tag{7}$$

This formulation yields straightforward real-time equations that can be implemented easily on a processor based system. It provides a structured $\mathbf{A}$ matrix such that a full matrix calculation is unnecessary and the use of forward path multipliers associated with the $\delta^{-1}$ operations creates well-scaled controller variables, i.e., they all have similar maximum values. Not only is this a key factor in facilitating the use of a fixed-point architecture, but also it avoids the need to convert the controller into cascaded first/second-order sections, something that is almost essential for high-order controllers using formulations based upon the $z$ operator. This structure is completely general and can be used for any LTI controller. It is only necessary that the controller is converted into the form given in (7), and this can readily be achieved using MATLAB and similar tools, including controllers specified in continuous rather than discrete form. An outline of the procedure is given elsewhere [10].

### C. Numerical Requirements

The overall bit resolution required to represent the internal state variables is determined by the number of bits used to sample input data and the number of bits required to handle internal underflow and overflow. The number of bits per sample is determined by the resolution of the analog–digital conversion (ADC) and digital–analog conversion (DAC) hardware used, which for control applications is usually around 12 b. The number of underflow bits can be derived from the structure in terms of coefficients and the required fractional output accuracy for small inputs. These bits ensure that small input values will not be truncated to zero when multiplied by a

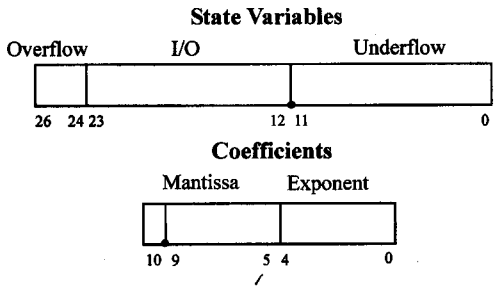


Fig. 2. Numerical specification.

small coefficient value and their value will propagate through the structure. The number of overflow bits required can be determined from the maximum overall gain that is likely to be seen in practice.

Using $\delta$-based controller formulation, the requirements for coefficients and controller state variables can be standardized across a wide range of applications. Fig. 2 shows a general format for the variables and coefficients. The variables are 27-b fixed-point, with the input values brought in as signed integer form, a small 3-b allowance for overflow (although this is a nominal requirement because of the good scaling properties mentioned previously), and 12 fractional bits for underflow. The position of the binary point is chosen so the input–output values map directly into the state variable format. The coefficients are held in a simple low-precision floating-point form, with 6 b for the mantissa and 5 b for the exponent. In this case, the position of the binary point is predetermined, and the 6 b mantissa provide a percentage accuracy typically around 1%, which is more than enough for most control applications.

This numerical specification will successfully implement the vast majority of LTI controller examples [13], and allows for the sample frequency being at least three orders of magnitude higher than the lowest pole in the controller, although normally we use a sampling frequency around 100 times the control loop bandwidth, which keeps the sampling/computation phase lag to below 5° [11]. Of course, if there are exceptional requirements it is always possible to reprogram the CSP hardware, maintaining the essential principles but extending the hardware precision as required. An example of implementing extremely high-sample frequency digital filters using the $\delta$ form approach can be found in [14].

## IV. CSP Hardware

### A. CSP Architecture

The CSP architecture has also been designed to allow both for single matrix implementation and for block-structured controllers. In the latter, a number of transfer-function blocks, each implemented by the formulation described previously, can be arbitrarily interconnected from inputs to outputs. This approach permits inter alia cascade-type controllers, and can also be used to reduce controller complexity if it is possible to identify appropriate substructures. More importantly complex classically-designed controllers can readily be implemented, because such controllers are almost invariably built up through intuitive understanding and experience using an application-specific set of loops, compensators and filters, i.e., the block-structured form.

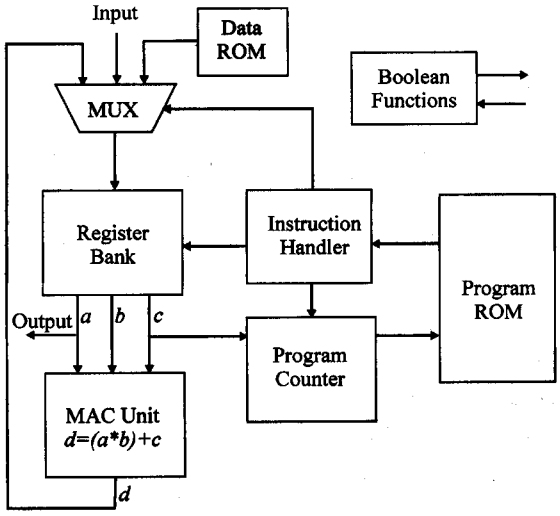


Fig. 3. Processor architecture.

Fig. 3 shows a block diagram for the CSP system. The core of the CSP comprises a simplified datapath with storage and computation capabilities. The register bank stores all the constants, coefficients, state variables, input data, output data, and partial products needed to perform the calculations. This architecture employs separate program and data buses to access separate data and program memories, an arrangement which increases speed since instruction and data can move in parallel and execute simultaneously rather than sequentially. The most significant feature of the CSP architecture is a 27-b fixed-point by 11-b floating-point multiplication executed in a single clock cycle. The operations are specified by the instructions fetched from an internal program read-only memory (ROM) and decoded by the instruction handler. The instruction format contains the source and destination addresses of the operands used in the MAC operation. The program counter addresses the next instruction in program memory to be executed. The internal data ROM contains the coefficients and the initial values for the state variables and program counter registers. A group of ADC and DAC converters provide the interface to the physical system being controlled. External data buses are used to connect the ADCs and DACs to the CSP input and output ports. The processor will be embedded within the complete control system.

The core of the CSP includes a special-purpose multiplier-and-accumulator unit (MAC) and a four-port register bank (three read, one write). The MAC unit executes the operations required to perform the control algorithm, i.e., $d = (a * b) + c$. The $a$ input is in coefficient format and the $b$ and $c$ values are in variable format as shown in Fig. 2. By using simultaneous program and data buses plus one pipeline stage, overall execution times are accelerated. To perform the MAC operation, the coefficient is split into its mantissa and exponent sections. The multiplier block multiplies a state variable by the mantissa using the Baugh–Wooley technique to multiply two numbers in two's complement format [15]. The product is then shifted by a number of bits determined by the coefficient exponent to complete the multiplication. Finally, the result is added to another specified state variable to produce the output.

TABLE I
CSP COMPLEXITY

| Block | Tiles (ProAsic) | Equivalent gates |
|---|---|---|
| Instruction Handler | 101 | 808 |
| MAC Unit | 1105 | 8840 |
| Program counter | 175 | 1400 |
| I/O Block | 60 | 480 |
| Pipeline registers | 120 | 960 |
| Program ROM | 900 | 7200 |
| Data ROM | 80 | 640 |
| **Total** | 2541 | 20328 |

TABLE II
CSP INSTRUCTIONS

| Instruction | Function | Description |
|---|---|---|
| MAC d, s1, s2, s3 | (s1*s2) + s3 -> d | Multiply accumulation operation |
| WRITEPC sel, s | s -> pc[sel] | Write to program counter registers (PC,jump1,jump2) |
| READ d, in_pt, s | If in_pt = 0 DataRom[s] -> d else nput[in_pt] -> d | Read from data ROM or input ADC |
| WRITE out_pt, s | s -> output[out_pt] | Read from register bank and write to output DAC |

### B. Complexity and Clock Speed

Table I shows the CSP complexity in terms of Actel's ProAsic device tiles and equivalent gates [16]. Everything except the program and data ROM are fixed in size for all controllers; these are hardwired, and their size and speed depends upon the control algorithm being implemented. The figures shown are for the controller specified in Section III. The synthesis of the CSP results in an overall gate count of fewer than 21 000 gates and 20 ns delay, which allows a clock frequency of 50 MHz. The register bank is implemented using nine embedded RAM blocks provided by ProAsic devices. Each block contains a 256-word deep by 9-b wide memory, with two ports (one read, one write).

## V. CSP SOFTWARE

### A. Instruction Set

The CSP instruction set is very simple and specialized, and it is targeted toward high-speed computation. The processor only has four instructions (Table II). The MAC instruction executes a multiply-accumulate operation on the operands, indicated by the source addresses, and stores the result in the destination address. This instruction performs the matrix multiplication according to the state–space representation of the control system. Because the constants 0, 1, and $-1$ are stored in the register bank, other calculations can also be mapped into this format. The MAC instruction can be used to add two values, increment a value by one, invert the sign of a value, simulate a no operation instruction or copy the value contained in one register. The READ instruction allows loading initial values from the data ROM into the register bank during the initialization process. Also, when the algorithm loop has begun, this instruction is used to read sampled input data from the input bus and to indicate the conversion time for the ADCs. Finally, the WRITE instruction is used to transfer the output values to the output bus. The instruction format of the CSP processor requires 38 b divided in five fields: 2-b operation code, 9-b destination address, and three 9-b source addresses.

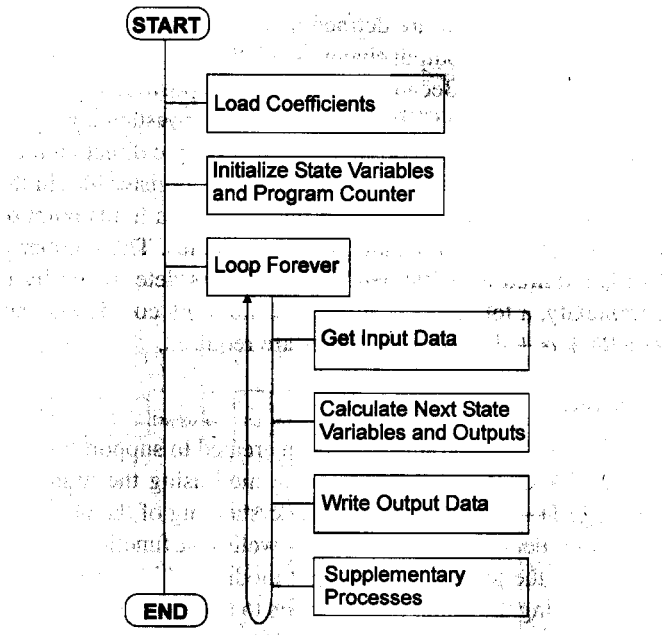


Fig. 4. Software scheme.

### B. Program Scheme

To generate the CSP program it is necessary to consider the order in which operations must be done, the number of inputs, the number of outputs, and the order of the control system to be implemented. Although the CSP program is modified according to the system to be controlled, the program scheme shown in Fig. 4 remains the same. The program is divided into two main parts: initialization and algorithm loop. In the initialization part, all the coefficients and state variable initial values are transferred from the external data ROM to the register bank. Also, the program counter registers, that specify the initial and final instruction for the algorithm loop, are updated. Finally, the program enters an infinite loop where the input samples are used to calculate the output values to the control system. The subtasks contained within the CSP program are listed in the following. Additionally, the numbers of CSP instructions required to perform each task are provided.

| Task | Number of CSP instructions |
|---|---|
| Load Coefficients | $n + n\alpha + \beta n + \alpha\beta + 3$ |
| Initialize State Variables and PC | $n + 2\beta + \alpha + 6$ |
| Start convert Signal | 1 |
| : Algorithm cycle start | |
| Get Input Data | $\alpha + 1$ |
| Calculate $\mathbf{DU}_k$ | $\alpha\beta$ |
| Calculate $\mathbf{Y}_k$ | $\beta$ |
| Write Output Data | $\beta + 1$ |
| Calculate $\mathbf{X}_{k+1}$ | $(2 + \alpha)n$ |
| Calculate $\mathbf{AX}_k$ | |
| Calculate $\mathbf{BU}_k$ | |
| Calculate $\mathbf{CX}_{k+1}$ | $n\beta$ |
| Start convert Signal | 1 |
| : End Algorithm cycle | |

Where $\alpha, \beta$, and $n$ are defined as the number of input channels, the number of output channels and the number of state variables, respectively. Because the number and position of the variables involved in the CSP algorithm remain constant during the program execution, it is possible to use a simple direct memory addressing mode to access the entries in the register file. In this mode, the addresses specified in the instruction fields point directly to the physical location of the variables. The number of words required to implement a controller is determined by its complexity, a total of $3 + n + n\alpha + nb + ab$ coefficients and $n + 2b + \alpha + 4$ internal variables are required.

### C. Software Suite

A number of programs have been created to support the CSP concept. A CSP model was programmed using the high-level language J++ to provide a clear understanding of the algorithm and its numerical requirements, as well as a functional specification of the processor. The CSP model architecture is modular, allowing the replacement of PEs to undertake performance comparisons and to explore new architectures. Input data to the model is provided from MATLAB simulations or from the CSP signal generator and the program to be simulated is generated by the CSP program generator.

A CSP signal generator provides input test data to the CSP model. The types of signal, number of samples, magnitude, sampling frequency, among others are indicated by parameters sent to the program in the command line. The format used to store data in the files is compatible with MATLAB double formats, so these test files can be used as inputs in any part of the design process.

The CSP program is created using the program generator. Each calculation part is generated using these parameters to modify the source and destination addresses for each instruction. Also, the program generator rearranges the order in which the instructions are executed to avoid the data dependency problems associated with pipelined designs. To program the CSP, the code is downloaded into the instruction memory. The chip is then placed in the system board and operation commences on an asserted start signal.

## VI. System Evaluation

### A. Basic Performance

A comprehensive set of tests has been undertaken to prove the CSPs operation and validate the effectiveness of its numerical formulation for a variety of filter types over a range of input conditions. The test scheme is shown in Fig. 5. A MATLAB program is used to implement and simulate the control algorithm using full precision of the variables to represent the coefficients and state variables. Next, a hardware implementation of the CSP is tested using the same input signals to compare the results against those obtained with the MATLAB program. The input values are all specified as integers within the 12-b input–output variable range and the output values produced by the CSP have been rounded to the nearest integer. The key comparison is between the CSPs output and that of the exact responses obtained from MATLAB, any differences being the consequence of quantization of both the coefficients and the variables.

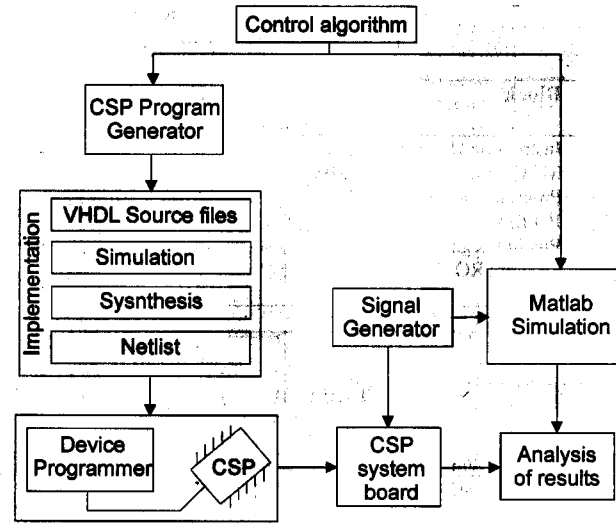


Fig. 5. Overview of process used to evaluate the CSP.

Several examples, drawn from real applications, have been used to establish the performance of the CSP. Here the focus is on the most complex system tested, which is a 46th-order twelve-input four-output controller. This is a classically-designed controller that provides the control of the vertical modes of a Maglev vehicle, a diagram of which is shown in Fig. 6. This was selected because it is a real example, originally implemented in analog form, from one of the author's previous research activities. It is a dynamically-complex example, and having a MIMO formulation provides a very demanding example for evaluating the CSP. Further details are not given because, although one time response is given, the main purpose is to obtain benchmarking results that do not rely upon the controllers' characteristics.

Fig. 7 is a typical result, in this case a step response from one input to one output. Despite the complexity of the example the comparison of the CSPs output with the exact response remains excellent; the amplitude quantization effects which can be seen are the result of greater than unity gains in the paths through the controller, i.e., amplification of the quantized levels of the input variables, rather than a consequence of internal quantization within the control algorithm. Table III provides further assessment of this controller's performance, giving the output errors for different sinusoidal input conditions. It can clearly be seen that the errors remain small under nearly all conditions, giving further confirmation of the effectiveness of the numerical format chosen for the CSP. The only occasion where the error becomes at all significant is with the low-frequency sinusoidal signal into one of the inputs where the error rises to 20%, but this is a particular feature of the quantization and scaling for this example, i.e., not a consequence of the CSP.

### B. Benchmarking Results

Also, benchmark comparisons have been made with other processors: Texas Instruments Incorporated TMS320C31-60 32-b floating-point and TMS320VC5416 16-b fixed-point DSPs, Infineon's C167 16-b fixed-point microcontroller, Intel's StrongArm-110 32-b processor, and Intel's 32-b Pentium III processor. The real-time code for these processors has been

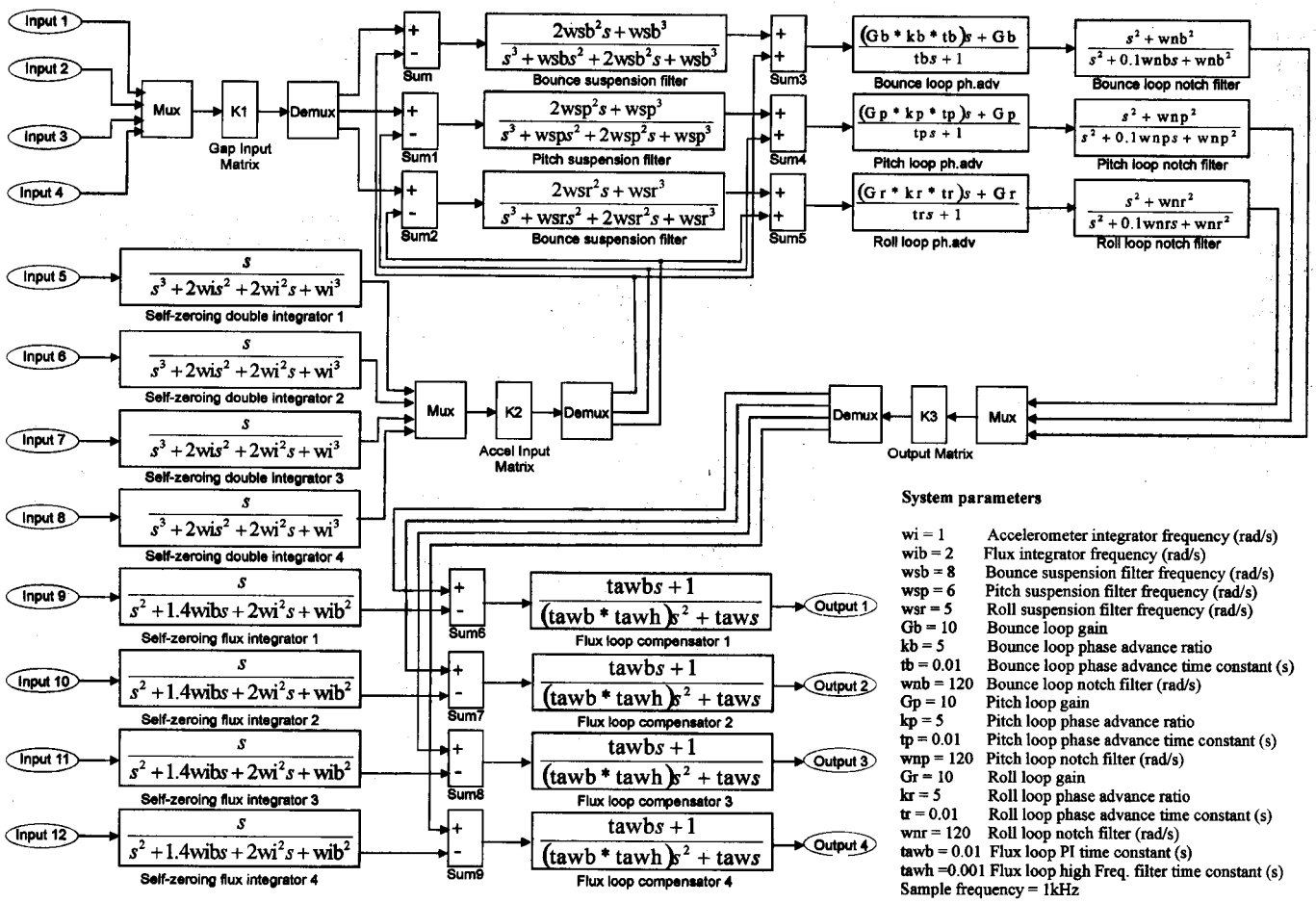


Fig. 6. Fourty-sixth-order 12-input four-output Maglev vehicle controller.

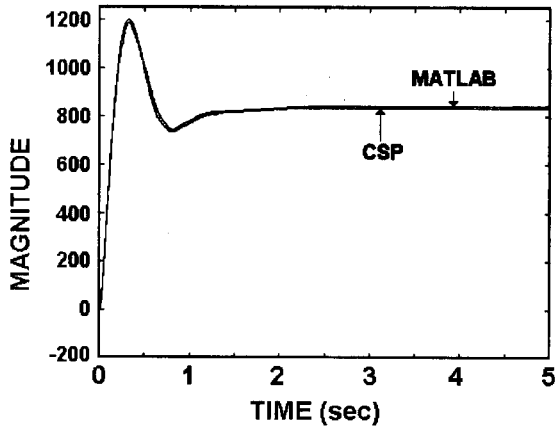


Fig. 7. Typical response of a 46th-order controller.

TABLE III
RMS ERROR AND PERCENTAGE ERROR OF THE INPUT–OUTPUT VARIABLES

| Signal applied to input number: | Sine 0.1 Hz | | Sine 1Hz | | Sine 10Hz | |
|---|---|---|---|---|---|---|
| | error | % error | error | % error | error | % error |
| 1 | 6 | 0.56 | 7 | 2.12 | 23 | 6.05 |
| 9 | 1 | 20 | 1 | 2.22 | 1 | 1.8 |

carefully assessed to ensure that a fair comparison is presented. To perform the calculations, compiled code with 32-b IEEE standard floating-point variables has been used in the 32-b floating-point devices and with 32-b signed integers for the 16-b fixed-point devices.

TABLE IV
COMPARISON OF CSP COMPUTATION TIME WITH OTHER PROCESSORS FOR THE MAGLEV CONTROLLER

| Processor | Frequency (MHz) | Computation time normalized to the CSP running at 50MHz |
|---|---|---|
| CSP | 50 | 1 |
| TMS320C31 | 60 | 3.83 |
| TMS320C54 | 160 | 5.97 |
| C167 | 25 | 61.7 |
| StrongArm | 233 | 0.75 |
| Pentium III | 500 | 0.29 |

The time required by the CSP to complete a complete iteration for the Maglev controller is 5860 ns. This means that a remarkably high 170-kHz sampling frequency is possible. Table IV shows how this result compares with the other processors. The computation times have been normalized to that of the CSP running at 50 MHz. The closest DSP in performance is the 60-MHz TMS320C31 device, which still takes 3.83 times as long to compute. The fastest fixed-point device is the 160-MHz TMS320C54 with $32 \times 16$-b multiplication which takes 5.97 times as long as the CSP. Only the StrongArm and Pentium III processors were faster than the CSP, they took 0.75 and 0.29 times, respectively. This is because the large number of instructions required by these processors is compensated by

the high clock frequencies at which they operate. For example the StrongArm operates some 25% faster than the CSP but this achieved at the expense of a clock speed over four times that of the CSP. The Pentium III achieves a 70% speed up but requires a tenfold increase in clock speed. This implies a substantial increase in energy consumption, an important consideration for some applications. Furthermore, the 50-MHz speed of the CSP is a consequence of the use of FPGA technology; a custom silicon implementation is likely to reach speeds in excess of 200 MHz restoring the overall benefit of the CSP approach.

## VII. Future Work

Further opportunities for its use are as an IP core to enable systems integrators to utilize its capabilities. The CSPs dedicated architecture and careful numerical formulation ensure that it will perform deterministically in a real-time embedded control environment, although it is recognized that other functions are necessary in such applications for which the CSP is not well suited. Research is continuing to address ways in which the variety of functions required for high-performance real-time control can be most effectively achieved and to develop a more complex system solution. The goal is to develop a whole control system solution on a programmable chip integrating the CSP with a general purpose processor, an AMBA compliant bus and standard communication interface on a single device.

## VIII. Conclusion

The brief describes the CSP, which is a compact, high-speed special-purpose processor, which enables a low-cost solution to a LTI control problems. The CSP architecture is optimized to implement a controller formulation base on the $\delta$ operator. Results of using the CSP to implement a very complex 46th-order system to control the vertical modes of a Maglev vehicle have been shown. It is important to appreciate that although the CSP outperforms, even in its FPGA implementation, a number of high-performance processors by a significant margin, it is much simpler. Indeed, its modest gate count confers a number of advantages, namely reduced cost due to small die size and simpler packaging, and low power permitting considerable periods of operation using battery power.

## References

[1] R. Landry, V. Calmettes, and E. Robin, "High speed IIR filter for Xilinx FPGA," in *Proc. IEEE Midwest Symp. Systems and Circuits*, South Bend, IN, Aug. 1998, pp. 46–49.

[2] S. B. Furber, J. D. Garside, P. Riocreux, S. Temple, P. day, J. Liu, and N. Paver, "AMULET2e, an asynchronous embedded controller," *Proc. IEEE*, vol. 87, no. 2, pp. 243–256, Feb. 1999.

[3] B. Garbergs and B. Sohlberg, "Specialised hardware for state space control of a dynamic process," in *Proc. IEEE TENCON—Digital Signal Processing Applications*, vol. 2, 1996, pp. 895–899.

[4] A. Costa, A. De Gloria, F. Giudici, and M. Olivieri, "Fuzzy logic microcontroller," *IEEE Micro*, pt. 1, vol. 17, pp. 66–74, Jan.–Feb. 1997.

[5] J. Liu, Z. Q. Mao, G. Z. Lu, and W. H. Han, "A new VLSI architecture for real-time control of robot manipulators," in *Proc. 1991 IEEE Int. Conf. Robotics Automation*, vol. 2, Apr. 1991, pp. 1828–1835.

[6] J. P. Agrawal, E. Bouktache, O. Farook, and C. R. Sekhar, "Hardware software system design of a generic embedded controller for industrial applications," in *Proc. Conf. Rec. IEEE Industry Applications Conf.*, vol. 3, 1995, pp. 1887–1892.

[7] A. Spray and S. Jones, "PACE: A regular array for implementing regularly and irregularly structured algorithms," *Proc. Inst. Elect. Eng.-pt. G*, vol. 138, pp. 613–619, 1991.

[8] Y. Tsunekawa and M. Miura, "High performance VLSI architecture suitable for control systems for state-space digital filters using distributed arithmetic," *Electron. Commun. Japan*, vol. 5, pp. 12–21, 1995.

[9] Y. Fujioka, M. Kameyama, and N. Tomabechi, "Reconfigurable parallel VLSI processor for dynamic control of intelligent robots," *Proc. Inst. Elect. Eng.–Comput. Dig. Techniques*, vol. 143, pp. 23–29, 1996.

[10] R. M. Goodall and D. S. Brown, "High speed digital controllers using an 8-bit microprocessor," *Softw. Microsyst.*, vol. 4, pp. 109–116, 1985.

[11] R. H. Middleton and G. C. Goodwin, *Digital Control and Estimation—A Unified Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[12] R. M. Goodall, "Perspectives on processing for real-time control," in *Proc. IFAC Workshop AARTC*, Palma de Mallorca, Spain, May 2000, pp. 1–9.

[13] W. Forsythe and R. M. Goodall, *Digital Control: Fundamentals, Theory, and Practice*. New York: McGraw-Hill, 1991.

[14] R. M. Goodall and B. Donaghue, "Very high sample rate digital filters using the operator," *Proc. Inst. Elect. Eng.*, pt. G, vol. 140, pp. 199–206, 1993.

[15] P. Pirsch, *Architectures for Digital Signal Processing*. New York: Wiley, 1998.

[16] *Actel Corporation*, Oct. 2000. ProASIC A500 K family.