An Algorithm For Direct Digital Implementation of Continuous-Time Systems

by Rodney A. Morejon
Naval Coastal Systems Center
Panama City, Florida

Abstract

Dynamic systems, which occur in numerous engineering applications from navigation systems to communication and control systems, are often represented in the form of continuous-time differential equation models. Frequently these continuous-time models have time-varying coefficients or nonlinear effects. These conditions have traditionally posed problems for most of the existing digital implementation and simulation procedures. However, by directly implementing the analog model in its natural modular form, these problems can often be handled with much less effort. In this paper some promising techniques are investigated for the direct digital implementation of analog prototype models which may contain nonlinear or time varying parameters.

1: Introduction

The direct digital implementation of dynamic systems presents both advantages and disadvantages. By directly implementing dynamic systems in their natural form, intricate discretizing procedures are avoided. Since the integrity of the analog states is retained, time varying coefficients pose no difficulty beyond the updating of variables. Thus the resulting models are continuously programmable. Nonlinear effects are also handled more readily. However, the processing requirements per sample period are often much higher for directly implemented models. In addition to this problem, higher sample rates are often necessary to satisfy the desired time and frequency response specifications.

In this investigation the formulation of a practical procedure for the direct implementation of dynamic systems is established. Procedures are structured to minimize the disadvantages associated with direct implementation schemes versus existing classical discretizing methods. Several discretizing architectures are used to formulate a variety of direct form algorithms. The most promising algorithm is applied to several analog prototype examples.

2: Background

Digital Signal Processors are high-speed special-purpose processors which are used in a broad range of applications from telecommunications and digital audio to robotics and engine control. The growing field of applications for DSP circuitry has put pressure on industry to make the processors faster, more accurate, and more versatile. Advancements in semiconductor VLSI technology have paved the way for the continuous evolution of these processors into more sophisticated and powerful circuits. The Texas Instruments TMS320C30, for example, has a 60ns cycle time and incorporates 700,000 transistors which allow it to execute more than 33 million floating point operations per second (MFLOPs) [1]. Many other companies, including Motorola, AT&T, and Fujitsu, are also involved in meeting the new demands on DSP technology. The advancements in processing speed are accompanied by similar gains in software flexibility. This flexibility allows the performance of many different tasks with no hardware changes; only a change in the program is required. DSP solutions are also desirable because of their reliability, compactness, and long term stability [1]. These features make DSP chips an excellent choice for implementing dynamic systems in digital hardware.

Countless engineering applications require the implementation of a dynamic system of one form or another. When the needed system is discrete in nature, DSP chips are ideally suited for the situation. However, when the dynamic system is described by a continuous-time differential equation, there are two choices for implementation of the analog model. One technique is to use active and passive electronic components to produce an analog circuit which represents the dynamic system. This technique suffers not only from the environmental and temporal fluctuations associated with most analog circuitry, but also from the lack of repeatability in producing components with identical characteristics. These difficulties, and the advantages of DSP circuitry, often motivate the use of more reliable digital techniques to implement a system which closely approximates the original continuous-time model.

Many popular techniques exist for implementing analog systems with digital hardware. These techniques are based on numerical integration methods, transform methods, and mapping methods. The bilinear transform and backward difference methods are based on numerical integration. They result in mapping equations from the s-domain to the z-domain. The impulse invariant and step invariant methods are transform methods. They require the use of z-transformations on s-domain transfer functions.
The matched pole-zero mapping method involves the mapping of individual poles and zeros from the s-plane to the z-plane [2]. Although these techniques often provide excellent results, complications may arise when the continuous-time model has time-varying or nonlinear properties.

When the analog model to be implemented contains time varying coefficients, the corresponding discrete model must be updated each time the coefficients change. Each of the above techniques involves an intricate discretization process. The numerical integration techniques require the calculation of all the coefficients of the resulting discrete difference equation as functions of all the new coefficients in the differential equation. The transform methods present computational difficulties as well. Taking the z-transform of a higher order transfer function necessitates factoring and a partial fraction expansion of the transfer function [2]. The matched pole-zero mapping method is also unsuited for conveniently updating coefficients since the transfer function must be factored each time.

If a nonlinear analog system is implemented using these discretization processes, then further computational considerations must be included. All three implementation types begin with a transfer function in the s-domain. This transfer function results from taking the Laplace transform of the defining differential equation. Since the Laplace transform produces an algebraic transfer function for linear differential equations, the nonlinear analog system must be linearized about a particular operating point before the corresponding s-domain transfer function is found. If the operating point changes, the system must be linearized again. Each time there is another linearization, the result s-domain transfer function must be discretized as well.

The primary reason for these difficulties originates in the discretization process itself. All of the methods described above involve the conversion of a differential equation to a difference equation in the z-domain. The resulting difference equation loses the integrity of all the continuous-time states except the output state. All the derivative variables are lost. When the coefficients of the states change, a new difference equation is required since the existing model does not retain the direct form of the analog prototype. In summary, these methods are appropriate for implementing a system described by a linear time invariant differential equation. However, for systems with more complicated dynamics, the difficulties with using these methods make it worthwhile to investigate techniques which implement the analog model in its more direct and natural modular form.

The direct digital implementation of dynamic systems presents both advantages and disadvantages. By directly implementing dynamic systems in their natural form, intricate discretizing procedures are avoided. Since the integrity of the analog states are retained, time varying coefficients pose no difficulty beyond the updating of variables. Thus the resulting models are continuously programmable. Nonlinear effects are also handled more readily. However, the processing requirements per sample period are often much higher for directly implemented models. In addition to this problem, higher sample rates are often necessary to satisfy the desired time and frequency response specifications.

3: Architectural Algorithm Development

The formulation of a direct digital implementation architectural algorithm for analog dynamic systems will be the focus in the following development. To assist in the development, a specific example of a continuous-time differential equation is used as a guide throughout the progression. This approach allows a qualitative comparison of architectures as they are formulated and is naturally extended to other dynamic systems. First, the prototype model is expressed in its analog computer diagram form. The model is discretized in its natural form using several digital implementation schemes. The resulting architectures are compared to determine the most efficient design. Additional topics of interest are also addressed.

3.1: Computer Diagrams

An analog computer diagram is a block diagram which represents the solution of a continuous-time differential equation in a form which is suited for analog implementation. The elements include inverters, summing amplifiers (adders), potentiometers (multipliers), and integrators [3]. For this development, a third-order linear differential equation is used to illustrate the procedure. The process is logically extendable to more general models.

The model to be implemented is the general third-order linear differential equation

\[ \ddot{y}(t) + b\dot{y}(t) + cy(t) + dy(t) = ax(t) \]  

(1)

The well known "bootstrap" method [3] for implementing the equation requires that (1) be solved explicitly for its highest order derivative,

\[ \ddot{y}(t) = ax(t) - b\dot{y}(t) - cy(t) - dy(t) \]  

(2)

The highest order derivative is integrated repeatedly to produce all the other derivatives and the output variable. These variables are then multiplied by their corresponding values and fed back to produce the highest order derivative as described by (2). The resulting analog computer diagram with its op-amp inversions is shown in Figure 1.

![Figure 1. Analog Computer Diagram for Third Order System](image-url)
All the elements in Figure 1 are directly realizable with analog circuitry. However, the reliability and versatility of digital circuits motivates a digital implementation instead. The primary difficulty in using the simulation diagram with digital components stems from the serial operation of digital devices [4]. In analog circuitry, variables are updated continuously and concurrently. With digital hardware, the output of one block must be computed before it can be used as an input to any other block. This phenomenon introduces delays in the computer diagram.

Two inherent delays in the digital implementation of the continuous-time system of Figure 1 are apparent. The first is the processing delay of the output. Since the output cannot be computed instantaneously, the current input does not affect the output until the next sample. The second delay occurs in the computation of the highest order derivative. Since the highest derivative is the sum of a feedback loop, there is an automatic delay in its calculation. These delays are the primary sources of difficulty in using digital techniques for continuous-time models.

### 3.2: Digital Integration Models

Many practical procedures have been developed for the application of digital integration. These procedures include the following: Euler’s Method (rectangular rule), Heun’s Formula (trapezoidal rule), Simpson’s Rules, Taylor Series, Runga-Kutta methods, and numerous multistep methods. The selection of the optimum routine involves a careful judgement of the advantages and disadvantages of each procedure. Some important considerations include computing time, accuracy, stability, ease of programming, internal precision, and memory requirements. There are many trade-offs associated with the selection of an integration scheme. A comprehensive discussion of this topic is a peripheral issue that would detract from the focus of this investigation. For this analysis, only the Euler and Heun integration schemes are considered, but the extension for the use of more complicated integration techniques is natural.

Euler’s Method of integration uses the rectangular rule to accumulate the area under the input. The next output is simply the previous output added to a rectangle of area $T_s \cdot x(k)$, where $T_s$ is the sample period and $x(k)$ is the current input. The following difference equation describes the function of an Euler integrator.

$$y(k) = y(k-1) + T_s \cdot x(k)$$

The Euler Method is the numerical integration principle used for the backward difference method of discretizing a continuous-time system.

Heun’s Formula for integration uses the trapezoidal rule to calculate the area under the input. The next output adds an area which is the product of the sample period and the average of the current and previous inputs to the previous output. The following difference equation describes the function of a Heun integrator.

$$y(k) = y(k-1) + \frac{T_s}{2} [x(k) + x(k-1)]$$

Heun’s Formula is the numerical integration scheme used for the bilinear transform method of discretizing a continuous-time system.

### 3.3: Direct Digital Implementation

In this section the third order equation (1) is implemented to illustrate the direct form algorithm. The coefficients of (1) are chosen to represent a third-order low pass Chebyshev filter with a ripple of 2 dB. Although any other prototype would suffice, this prototype was selected for two reasons. First, since this is a third order model, direct implementation of a higher order system without partitioning is established. The second reason involves system dynamics. A third order Chebyshev filter with a 2 dB ripple possesses significant resonating dynamics in both the transient and frequency responses.

From design tables the coefficients of (1) are found and result in the following normalized equation (5):

$$\ddot{y}(t) + 0.7146\dot{y} + 0.9583y + 0.2969y = 0.2969\dot{x}$$

Equation (5) represents a third order lowpass Chebyshev filter with 2 dB ripple, DC gain of one, and a cutoff frequency of 1 radian per second. Using the diagram of Figure 1 and the Euler and Heun discrete integration schemes, (5) is implemented digitally to obtain a comparison with the actual response of the system. The sampling ratio for a maximum percent error in the passband of the filter was chosen as the basis for evaluating implementations. The sampling ratio is defined as the ratio of the sampling frequency to the model’s cutoff frequency. Without using predictors to correct for the aforementioned feedback and output delays, the sampling ratios (i.e. processing requirements) were 1500 for 1% errors - much too high! With predictors, direct implementations are vastly improved.

### 3.4: Predictors

The direct implementation of Figure 1 without predictor correctors resulted in the requirement of very high sampling rate to cutoff frequency ratios. These surprisingly high sampling ratios are attributed to the feedback and output variable delays. Two predictors are chosen to lessen the adverse effects of these delays. Since the feedback delay involves more terms, it is examined first. Then, the output processing delay is addressed.

A predictor is a digital processing element which is used to estimate the next value of a sequence based on existing values and existing derivatives. Since the direct implementation scheme preserves the integrity of the derivatives, it is well suited for using predictor models. Like digital integration schemes, there are many useful predictor models to choose from. For this investigation, a simple point-slope (Euler’s Method) predictor is used to determine if the required sampling ratios can be reduced.
When favorable results were obtained, other predictors were tried. Taylor Series, Adams-Bashforth, and midpoint method predictors were compared, but none performed as well as the Euler predictor. The Euler predictor estimates the next value by adding the sample period multiplied by the current slope to the current value. Equations (6) describe the predictors used.

\[ \hat{y}(k+1) = y(k) + T_s \cdot \dot{y}(k) \]  

(6a)

\[ \hat{y}(k+1) = \hat{y}(k) + T_s \cdot \dot{y}(k) \]  

(6b)

\[ \hat{y}(k+1) = \bar{y}(k) + T_s \cdot \bar{y}(k) \]  

(6c)

If equations (6) approximate the next values accurately, the feedback delay in Figure 1 is effectively eliminated.

A similar procedure is used to account for the output processing delay in Figure 1. For this predictor, Euler's method is modified to include a constant prediction factor, \( \alpha \). Thus the equation to predict the output is given by

\[ \hat{y}_o(k+1) = y(k) + \alpha \frac{T_s}{2} \cdot \dot{y}(k) \]  

(7)

This factor is included to allow the regulation of the output estimate. The range of \( \alpha \) should be between 0 and 2. A value of zero implies that there is no forward approximation or prediction. When \( \alpha = 2 \), the value is estimated one full sample ahead based on the slope at the current point. This selection provides good frequency response correlation for highly dynamic models, but can result in large settling times for the step response error. A more conservative selection of \( \alpha = 1 \) results in a favorable compromise between transient and frequency response characteristics. Table I presents the required sampling ratios for various values of \( \alpha \) using Heun integrators and Euler feedback predictors.

<table>
<thead>
<tr>
<th>Prediction Factor ( \alpha )</th>
<th>1.0%</th>
<th>2.0%</th>
<th>5.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 1.0 )</td>
<td>56:1</td>
<td>39:1</td>
<td>24:1</td>
</tr>
<tr>
<td>( \alpha = 1.5 )</td>
<td>51:1</td>
<td>36:1</td>
<td>22:1</td>
</tr>
<tr>
<td>( \alpha = 2.0 )</td>
<td>44:1</td>
<td>30:1</td>
<td>18:1</td>
</tr>
</tbody>
</table>

The results in Table I illustrate that reasonable sampling rates can be used with these direct implementation architectures. The direct implementation algorithm for the results of Table I was based on the following filter model equations:

\[ \ddot{y}(t) = a\ddot{y}(t) - b\dot{y}(t) + c\ddot{y}(t) - dy(t) \]  

(8)

\[ \ddot{y}(k) = \ddot{y}(k-1) + \frac{T_s}{2}[\dddot{y}(k) + \dddot{y}(k-1)] \]  

(9a)

\[ \dddot{y}(k) = \dddot{y}(k-1) + \frac{T_s}{2}[\dddot{y}(k) + \dddot{y}(k-1)] \]  

(9b)

\[ \dddot{y}(k) = \dddot{y}(k-1) + \frac{T_s}{2}[\dddot{y}(k) + \dddot{y}(k-1)] \]  

(9c)

\[ \dddot{y}(k+1) = \dddot{y}(k) + T_s \cdot \dddot{y}(k) \]  

(10a)

\[ \dddot{y}(k+1) = \dddot{y}(k) + T_s \cdot \dddot{y}(k) \]  

(10b)

\[ \dddot{y}(k+1) = \dddot{y}(k) + T_s \cdot \dddot{y}(k) \]  

(10c)

The architectural implementation of these equations are depicted in Figure 2.

Figure 2. Direct Digital Implementation of a Third Order System

The architecture of Figure 2 has been developed to directly implement dynamic systems with digital algorithms without requiring unfeasibly high sampling rates. In addition to the topics that were discussed in the development of this example, there are various associated topics that need to be discussed. Computational requirements, processor characteristics, and scaling are among these issues. These issues will be deferred to a planned full-length journal paper where space will permit adequate treatment of these peripheral, but very important topics.

4: Infinite Impulse Filter Examples

The vast amount of literature on the design of analog infinite impulse response (IIR) filters makes them a fitting candidate for direct digital implementation. By directly implementing these systems, the extensive database of analog design coefficients can be used without transformation. The Butterworth and Chebyshev filter families are two well documented IIR filter families [5]. The Butterworth filter is characterized by its maximally flat gain in the passband. Chebyshev filters are characterized by an equal ripple gain in the passband. Both filter types are described by an Nth order differential equation of the same form, which is

\[ y^{(N)} + a_{N-1}y^{(N-1)} + a_{N-2}y^{(N-2)} + \cdots + a_1\dot{y} + a_0y = bx \]  

(12)

Solving (12) for its highest derivative and using the method described in the previous section results in the digital implementation diagram form of Figure 2.

The system model of Figure 2 was simulated for 1st to 5th order Butterworth and Chebyshev filters. The results are tabulated in Tables II.
TABLE II

<table>
<thead>
<tr>
<th>Order</th>
<th>Maximum Percent Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1.0% 2.0% 5.0%</td>
</tr>
<tr>
<td>1</td>
<td>23:1 15:1 8:1</td>
</tr>
<tr>
<td>2</td>
<td>41:1 29:1 18:1</td>
</tr>
<tr>
<td>3</td>
<td>54:1 38:1 24:1</td>
</tr>
<tr>
<td>4</td>
<td>60:1 42:1 26:1</td>
</tr>
<tr>
<td>5</td>
<td>58:1 41:1 26:1</td>
</tr>
<tr>
<td>Butterworth Filters</td>
<td>Chebyshev Filters</td>
</tr>
</tbody>
</table>

The information in Tables II suggests that higher order systems may require larger sampling ratios in order to achieve similar accuracy. A comparison of the Butterworth results with the Chebyshev results shows that the required increase in sampling ratio is not a simple function of the filter order. It was anticipated that the increase in the required sampling ratio is a direct function of the dynamics of the system. This speculation was verified with further studies.

5: Nonlinear and Time-varying System

The direct implementation architecture or algorithm is ideally suited for nonlinear and time-varying system models. Typically, these models are implemented by simply varying the analog coefficients as a function of time and system variables, respectively. Filters with time-varying coefficients and a Vander Pol nonlinear model were implemented to illustrate these concepts. Again, space does not permit elaboration at this time (these implementations will be covered in the planned future journal paper). In any event, the implementation of the direct form architecture, though not difficult, is complicated by time-varying and nonlinear effects. For example, sampling ratios can be larger than expected, nonlinear functions can be difficult to evaluate, etc.

6: Conclusions

From the results of the investigation described in this paper, it can be concluded that a proven direct digital implementation procedure for analog systems has been established. Numerous advantages are available as listed:

- Continuous-time differential equation models can be implemented directly in digital hardware/software without the requirement for intricate discretization procedures.
- The vast database of analog filters and other dynamic models is available for digital implementation in a direct form.
- Higher order models can be implemented without partitioning the models.
- Nonlinear systems can be simulated without the requirement for linearization.
- Time-varying filter and controller system components can be continuously updated with a simple bus communication path to the CPU.

The direct implementation of time-varying models and certain classes of nonlinear models is a testimony to the power and flexibility of the procedure described in this paper.

Finally, the preliminary results of this study implies that there are a number of additional topics deserving further study, some of which are summarized as follows:

- An extensive review of the available DSP hardware to determine what is suitable for implementing the developed procedure and other similar extensions.
- The formulation of design rules which could be used as a guide to determine the hardware requirements when given the system specifications.
- The development of processor-dependent software algorithms for the realization of the direct implementation procedure.
- Actual hardware/software implementation of the procedure with appropriate system analysis results (using a Digital Signal Analyzer).
- The investigation of other possible direct implementation procedures using different digital integration and prediction routines.

References


Computer Simulation Notes

All the computer simulation results for this investigation were obtained with an IBM XT clone using Lotus 1-2-3 version 2.01. This software package was originally selected for its versatility and the ease with which a graphical output could be produced. These two characteristics were crucial during the developmental stages of the project on which this paper was based. However, the speed of recalculation of the spreadsheets was often slow and tiresome. That will not be the case for the newer and faster inexpensive machines so readily available now. Furthermore, carefully designed software simulations, based on FORTRAN or C, would provide more powerful and efficient design tools once the initial development work is completed.