Identifying Approaches to the Formalization of DSPs

Kenneth L. Albin, Robert S. Boyer
Warren A. Hunt, Jr., Lawrence M. Smith
Darrell R. Word

Computational Logic Incorporated
1717 West Sixth Street, Suite 290
Austin, Texas 78703-4776

Tel: +1 512 322 9951
FAX: +1 512 322 0656

Hunt@CLI.COM

Talk Topics

- DSP Applications
- DSP Functional Characteristics
- The Motorola DSP 56100 Core Processor
- A DSP-controlled Switching Power Supply
- Conclusions

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

DSP Applications

DSPs are used in a variety of applications:

- Telecommunication
  - noise cancellation, ISDN transceivers, secure telephones
- Data Communication
  - high-speed modems, high-speed FAX, encryption/decryption
- Radio Communication
  - cellular telephone, spread-spectrum transceivers
- Computer
  - array processors, workstations, graphics accelerators
- Image Processing
  - pattern recognition, OCR, image processing, robot vision
- Graphics
  - 3-D rendering, CAE, desktop publishing, animation
- Instrumentation
  - spectral analysis, transient analysis, data acquisition

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

DSP Applications, Continued

- Speech Processing
  - speech synthesis and recognition, vocoder, authentication
- Audio Signal Processing
  - digital AM/FM radio, music processing, acoustic equalizer
- High-Speed Control
  - hard-disk servo, robotics, motor controller, position/rate sensor
- Vibration Analysis
  - electric motors, jet engines, turbines
- Medical Electronics
  - cat scanners, sonographs, X-ray analysis, NMR analysis
- Digital Video
  - digital television, high-resolution monitors
- Radar and Sonar Processing
  - navigation, automatic vehicle location, searching and tracking
- Seismic Processing
  - oil exploration, geological exploration

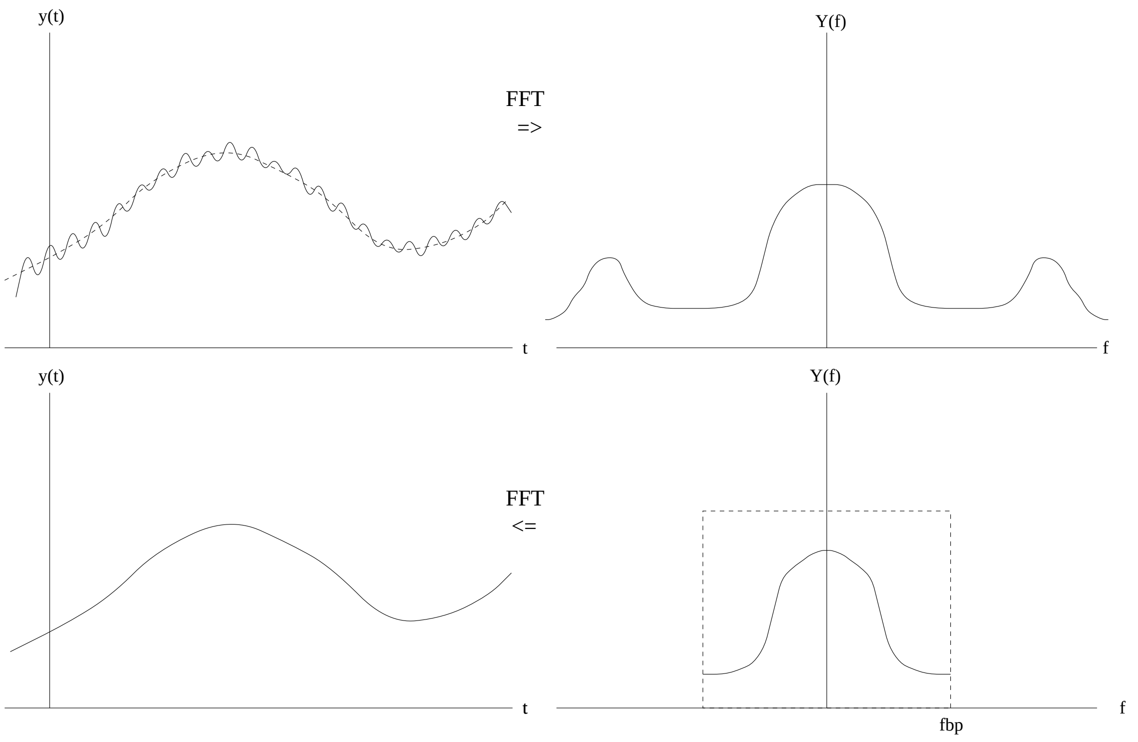
CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Fourier Transform

DSPs are often used to implement fast-fourier transforms (FFTs) and their inverse in real time.

For such an implementation, it is necessary to do a windowed, discrete FFT and its inverse.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED



Filtering in the frequency domain

DSP Functional Characteristics

There are a number of features that differentiate DSPs from standard microprocessors:

- on-board multi-ported memory (not cache);
- user-visible pipeline; fast interrupts;
- fractional arithmetic;
- programmable rounding, scaling, and saturation;
- multiply-accumulate operations;
- hardware support for DO loops;
- parallel operations and moves;
- no supervisor mode; unique addressing modes;
- high degree of integration; and highly configurable.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Hardware Verification: What Is It?

Industry-standard techniques for assuring the quality of hardware are:

- Simulation, and
- Testing.

These are crucial tools, but we hope to model hardware mathematically and prove the correctness of our hardware modules.

Our goal is to work toward the point where documentation for a hardware system is a *formula manual*, a collection of fully formal specifications that provide a precise and complete basis for using the device.

To assure that the formula manual is correct, it will be necessary to *prove* that a design meets *all* of its specification.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Simulation vs. Verification

Consider the following specification: $2 \times x$ [1].

This can be implemented as: $x + x$ [2].

One way to check that [1] can be implemented by [2] is to insert some numbers for x in both equations. This is essentially simulation.

Another way to check that [2] implements [1] is to prove the equality of [1] and [2] using algebra. This is verification.

Verification demonstrates that [2] implements [1] for all values for x.

Since hardware is finite, the number of different *states* is not infinite, but often so large that simulation can only test a small percentage of the possible behaviors of a device.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

DSP Formalization Facets

We believe that it is within the state-of-the-art to describe a DSP chip at the same top and bottom levels at which the FM9001 was described and to do a mechanical proof of the correspondence of these two levels; however, some cavets are discussed below.

As we investigated the formalization of a DSP we observed some interesting facets:

- user-visible pipeline -- an inplementation that requries a DSP user to be completely aware of the execution pipeline.
- user-visible cache -- a DSP user must be aware of how the cache works.
- block-repeat -- a user must understand the interaction with the pipelines and the interrupt mechanism.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

DSP Formalization Facets, continued

- timers -- if a timer is asynchronous with respect to a DSP’s internal clock, then we enter into a fundamentally new and deeply difficult area from a formalization perspective.
- parallelism -- some DSPs provide internal parallelism or co-processor interfaces.
- floating point -- algorithms involving floating point are difficult to formalize and verify.
- interrupts -- on Motorola’s DSP 56100 family it is possible to "loose" an interrupt.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Motorola DSP 56100 Processor

The Motorola DSP56100 core processor is the basis of the popular 56116, 56156, and 56166 DSPs. This is the simplest family in the Motorola DSP line but contains most features found in commercial DSPs. We have attempted to formalize some of its novel features to assess difficulties that will arise in creating a formal DSP specification:

- a three-stage visible pipeline,
- parallel operations within most instructions,
- complex instruction encoding,
- specialized address generation, and
- complex data transfers (sign-extension, rounding, etc.)

During our specification effort, an attempt has been made to identify methods of dealing with these added complexities while making the specification accessible to engineers.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

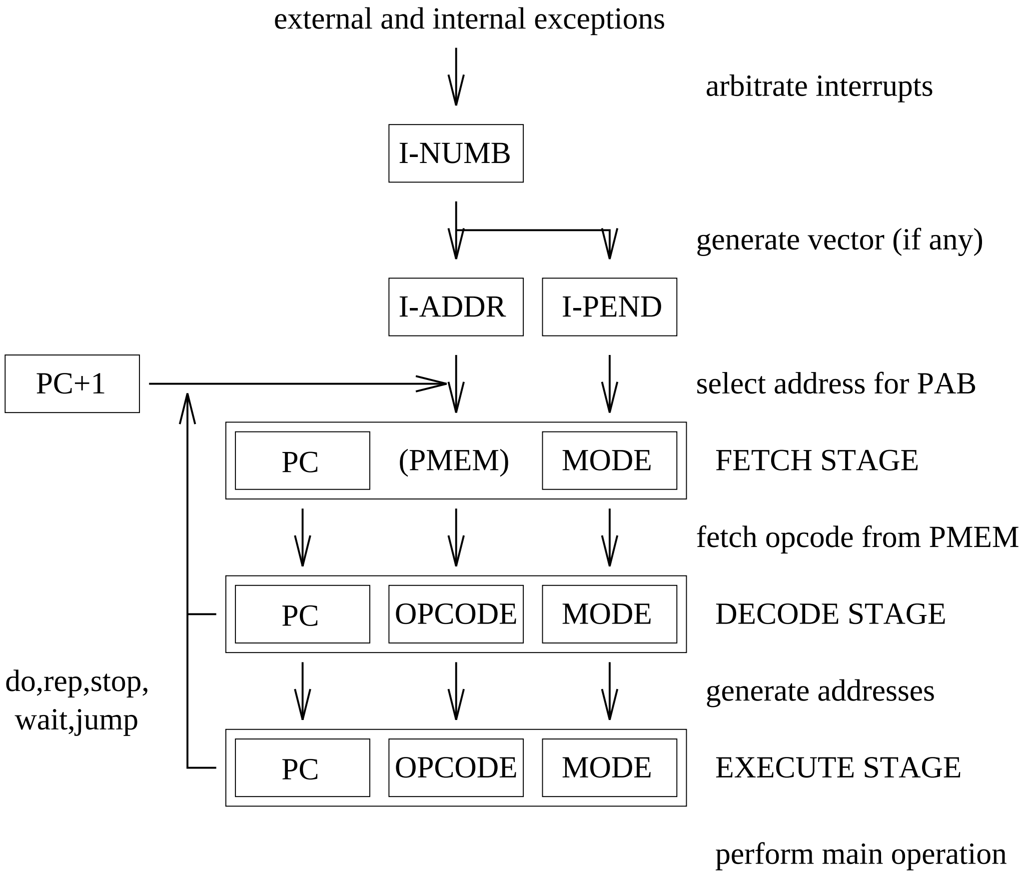
Pipeline-Oriented Step Function

Since more than one instruction executes at a time in pipelined machines, instructions sharing the pipeline may contend for resources. This interference cannot be modeled with the traditional approach of specifying instructions in isolation.

We have modeled the pipeline at a lower level with a step function which advances the pipeline one stage and all instructions in the pipeline are used to compute the next state.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Simplified View of the Pipeline



Describing the Pipeline Step

We have implemented an interpreter for an extremely simple RTL and have used RTL statements to describe the actions of DSP instructions.

Associated with each instruction specification are two lists of RTL statements describing the instruction's effect while in the decode and execution stages of the pipeline respectively. There is also an RTL specification of the pipeline updating actions.

By combining the RTL statements for each instruction in the pipeline, it is simple to write a predicate which finds statements with duplicate destinations indicating resource conflicts.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Decode Stage for ADD

The decode stage is primarily for generating addresses and updating the address registers. The Address Generation Unit also has its own status flags which are set during address register updating.

```
  ;;; use R3 for the 2nd X memory address
'(((XAB2) I<= (R 3))

  ;;; update R3 at the same time
  ((R 3) I<= (TRUNC 16 (RTL-ADD (R 3) (N 3))))

  ;;; use R0 for the 1st X memory address
  ((XAB1) I<= (R 0))

  ((AGU SR_V) I<= (IF (EQUAL (M 0) 65535)
                      (OVERFLOW-P (R 0) (N 0))
                    (IF (AND (NOT (ZEROP (M 0)))
                             (ZEROP (BIT 15 (M 0))))
                        (WRAP-P (R 0) (RTL-ADD (R 0) (N 0)) (N 0))
                      0)))

  ((AGU SR_Z) I<= (ZEROP (TRUNC 16 UPDATE*)))

  ((AGU SR_N) I<= (AND (NOT (ZEROP (M 0))) (BIT 15 UPDATE*))))
```

Decode Stage for ADD (cont.)

The following is the address register updating, which is complicated by three DSP specific modes.

```
(UPDATE* I<=
 (TRUNC 16
       (IF (ZEROP (M 0))
           (REV 16 (RTL-ADD (REV 16 (R 0))
                            (REV 16 (N 0))))
         (IF (EQUAL (M 0) 65535)
             (RTL-ADD (R 0) (N 0))
           (IF (ZEROP (BIT 15 (M 0)))
               (IF (WRAP-P (R 0)
                           (RTL-ADD (R 0) (N 0))
                           (M 0))
                   (IF (ZEROP (BIT 15 (N 0)))
                       (RTL-ADD (RTL-ADD (R 0) (N 0))
                                (BIT-NOT 16 (M 0)))
                     (RTL-ADD (RTL-ADD (R 0) (N 0))
                              (RTL-ADD (M 0) 1)))
                 (RTL-ADD (R 0) (N 0)))
             UNK)))))

((R 0) I<= UPDATE*)
```

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Execute Stage RTL for ADD

These are the actions performed by the ADD instruction while in the execute stage.

```
   ;;; main alu operation
 '((ALU-OUT* I<= (RTL-ADD (SE 16 40 (X 1)) (B)))

   ((B) I<= (TRUNC 40 ALU-OUT*))

   ;;; setting status bits
   ((CCR N) I<= (BIT 39 ALU-OUT*))
   ((CCR Z) I<= (ZEROP (TRUNC 40 ALU-OUT*)))
   ((CCR V) I<= (OVERFLOW-P (SE 16 40 (X 1)) (B) ALU-OUT*))
   ((CCR C) I<= (BIT 40 ALU-OUT*))

   ;;; parallel moves (X memory to X0 and X1)
   (XDB* I<= (XMEM (XAB1)))
   ((X 1) I<= XDB*)

   (GDB* I<= (XMEM (XAB2)))
   ((X 0) I<= GDB*))
```

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

A Pipeline Conflict

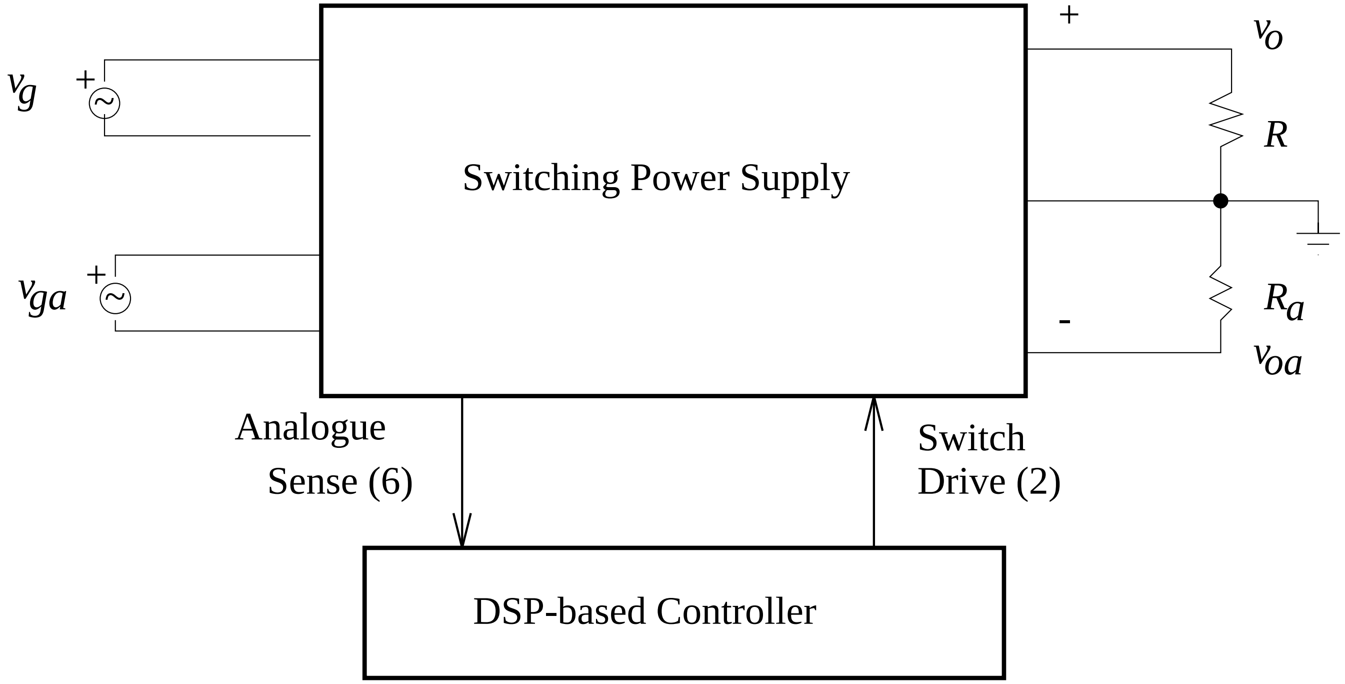
Because the actions of an opcode for each stage are represented explicitly, it is a simple matter to discover resource conflicts. For example, a MOVE instruction with R0 as the destination in the execute stage and an ADD instruction which is updating R0 during address generation have an easily spotted resource conflict.

```
;;; decode stage operations from ADD instruction
  .
  .
  ((R 0) I<= UPDATE*)
  .
  .
;;; execute stage operation from MOVE instruction
  .
  .
  ((R 0) I<= GDB*)
  .
  .
```

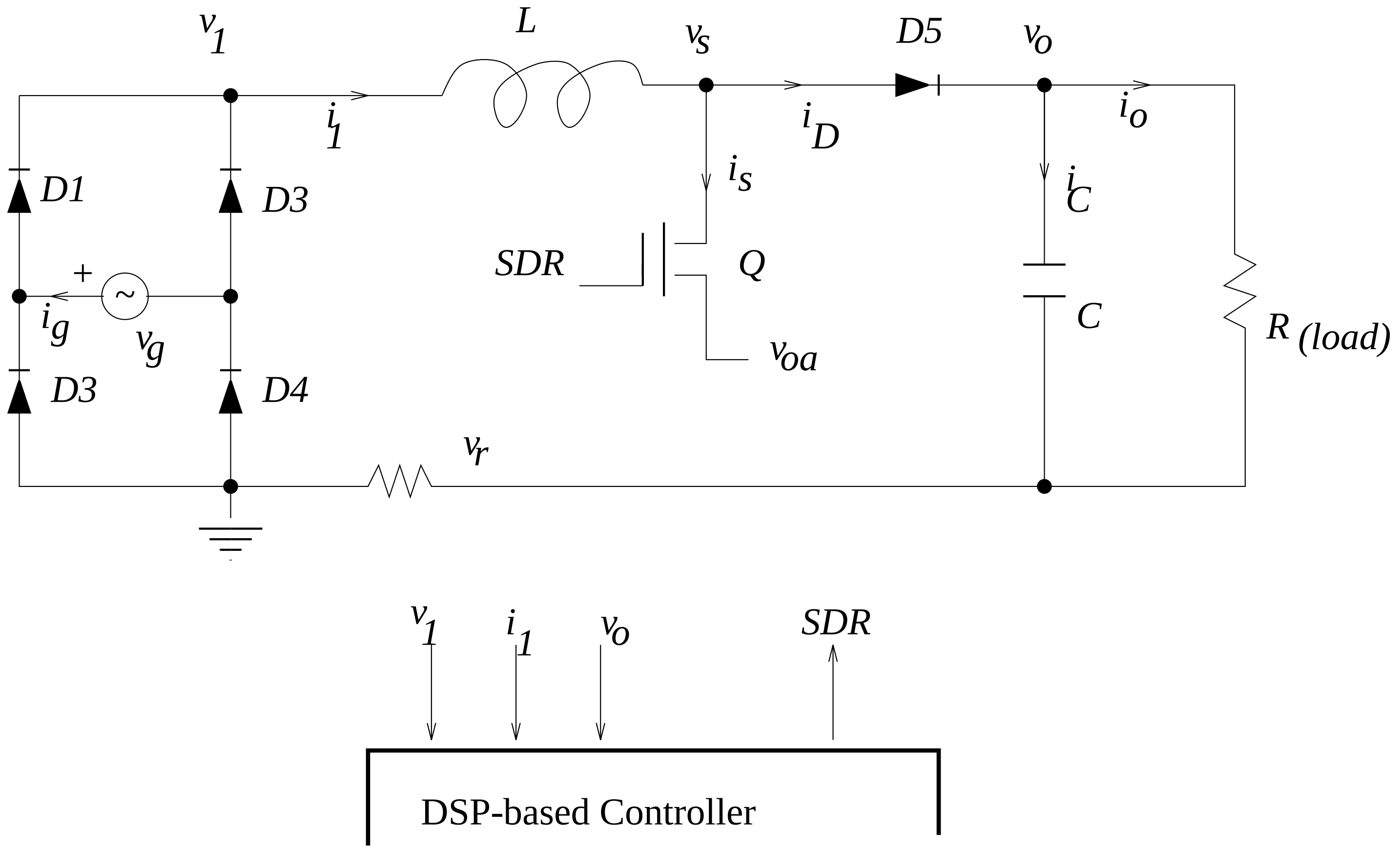
CLI
COMPUTATIONAL
LOGIC
INCORPORATED

A Power Supply

To provide a bit of realism, we have attempted to formalize some aspects of a switching power-supply design. A primary design goal of this power supply is that it have a very high power factor.



The Actual Design



CLI
COMPUTATIONAL
LOGIC
INCORPORATED

The Top-Level Specification

Our top-level specification for the power-supply is shown below.

```
(implies
 (and (various-conditions v-in i-in v-out i-out)
      (sine-p v-in))

 (let ((spec-power-output
        (times (efficiency) (power-delivered v-in i-in)))
       (output-v-i (switcher v-in i-in)))
  (let ((v-out (car output-v-i))
        (i-out (cadr output-v-i)))
   (let ((switcher-power-output (power-delivered v-out i-out)))

     (leq spec-power-output
          switcher-power-output))))))
```

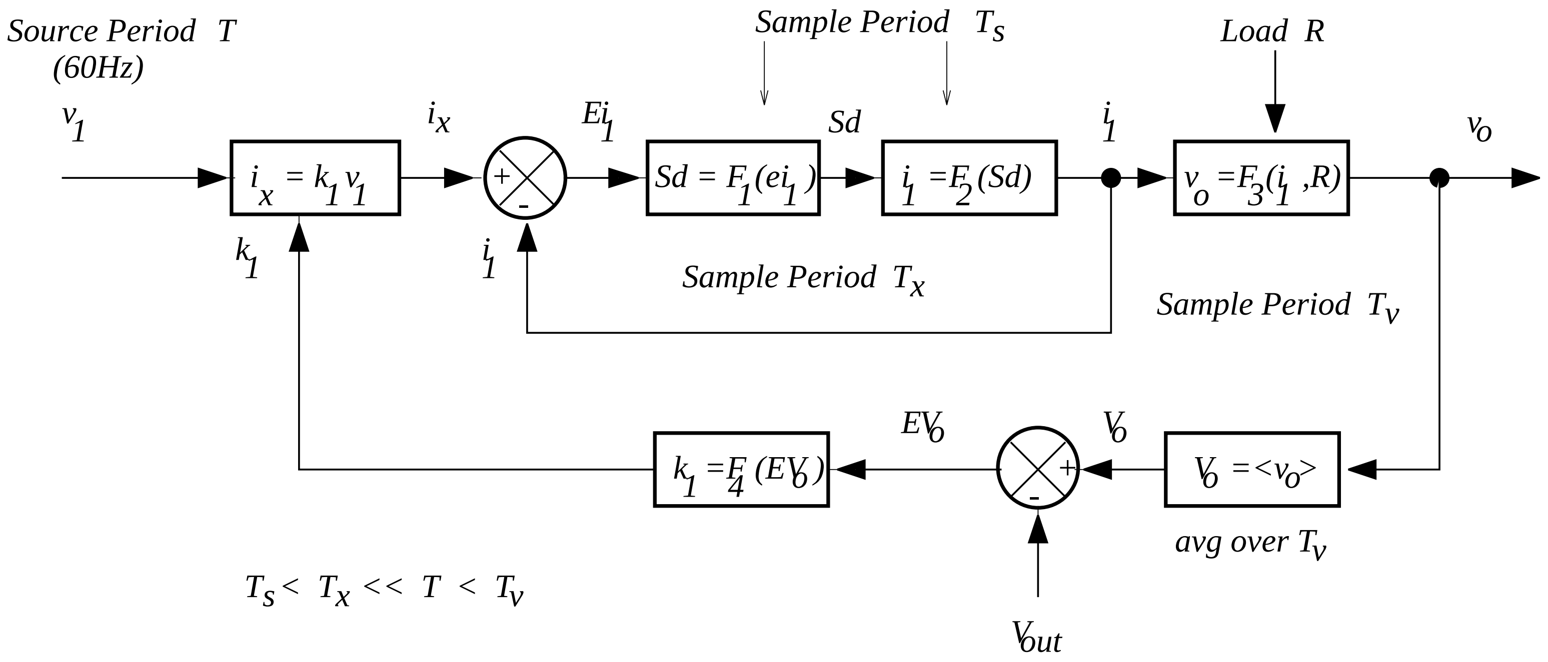
`v-in`, **`i-in`**, **`v-out`**, and **`i-out`** are lists of values containing a number of discrete voltage and currents over time.

The conclusion states that the function **`switcher`** produces at least as much power as the derated input.

CLI COMPUTATIONAL LOGIC INCORPORATED

The Power-Supply Control System

Below is a diagram of our control system.



$$T_s < T_x << T < T_v$$

Power-Supply Remarks

We have a discrete set of difference equations that provide a solution to the differential equations that govern the operation of the power supply.

We believe that it is possible to model with Nqthm these difference equations, and thus, possible to prove the correctness of a DSP-controlled power supply.

We believe that formulas can be used, not only to model a DSP, but also some of its application environment.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Power-Supply Remarks, continued

The process of mentally converting an abstract specification of a real-time control problem into a working design is complex and difficult.

By expanding use of mathematical methods to include the application environment, many important design decisions can be subjected to the rigor of a formal analysis.

This is an important new research area and we believe the formalization of such a design process to be of fundamental importance.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED

Conclusions

We believe it is within the state-of-the-art to mathematically specify a DSP, assuming that we can manage the caches, interrupts, and timers in some reasonable fashion.

We believe that it is within the state-of-the-art to prove the correspondence between DSP top and bottom levels, given that these levels are similar to the top and bottom levels used in the FM9001.

We believe that it is of fundamental importance to model a DSP's application environment as well as a DSP itself.

The degree of complexity of a commercial DSP is beyond anything that has been formally specified to date.

CLI
COMPUTATIONAL
LOGIC
INCORPORATED