ARCHITECTURE AND PROGRAMMING OF A MULTIRATE DIGITAL FILTER

Nicholas Roethe

Institut für Übertragungstechnik
Technische Hochschule Darmstadt
Merckstraße 25
6100 Darmstadt, West Germany

Abstract

This paper describes a signalprocessor that is specialized on the realization of single- and multi-rate digital filters. It is designed around a TRW multiplier /accumulator and contains a coefficient- and data-address -computer especially suited to the filter-algorithms. It is programmed in a high level filter-description language with a syntax resembling PASCAL. A crosscompiler translates these programs directly into the machines microcode. Since the language is limited to constructs, that the machines architecture supports efficiently, the automatically generated microcode is only marginaly less efficient than optimal handwritten code.

Introduction

The signalprocessor SL16 described in this paper was designed as a tool for the realtime realization and evaluation of multirate digital filters, namely

- interpolators, consisting of a sampling rate increase and a succeeding lowpass or bandpass filter, and
- decimators, consisting of a filter and a succeeding sampling rate decrease /3/.

Our main application are single channel single-sideband modulators and demodulators, containing several multirate filters /7/. Our previous experience with various small and a singlechip (NEC 7720 /6/) signalprocessors showed, that

- the capabilty of their arithmetic units (several million FIR-taps per second) would be sufficient for our application, but
- that the arithmetic unit can be kept running for only 20 to 40% of the time, whereas the rest of the processors cycles is needed for loading the adressgenerator, manipulating addresses, transfering data, synchronization and so on.

In addition the coding of our nontrivial algoritims in one of the microcode languages used by most small signalprocessors is very tedious and unacceptable for an occasional user of the machine. We therefore had the following design-objectives for our signal processing system:

- specialization on digital filtering
- full utilization of the arithmetic units capabilities
- an easy-to-program architecture, allowing efficient automatic codegeneration
- programming in a high level language and crosscompilation to the machine language level.

Machine Architecture

Figure 1 shows the architecture of the signalprocessor. The arithmetic unit is a TRW 16·16 Bit multiplier/accumulator. Two 256·16 RAMs store the data and coefficients respectively. They are connected to the arithmetic unit through separate buses. The analog in- and outputs, two each, are connected to one of the buses through 64 word FIFO memories. The FIFOs considerably simplify the realization of multirate systems and save processor time otherwise needed for loading the address -generator, synchronization etc. The machine is controlled by a sequencer AMD 2910, permitting subroutines, repetitive loops and conditional branching. An additional 16·1 memory stores 16 boolean flags, that can be tested and modified by the sequencer. The microprogram memory is a 256·40 RAM. Compressing the microprogram wordlength to 40 bits required some overlaying of fields, resulting in three different microinstruction types:

- 'Conditional Jump'
- 'Load Address-computer'
- 'Arithmetic Instruction'

The address-computer is controlled by the microinstructions and supplies the coefficient- and data-addresses required for the arithmetic operations. The following two addressing modes are required:

- 'Absolute Addressing' for temporary data in a scratchpad part of the data memory and for constants in the coefficient memory. In this addressing mode the address is taken directly from the microinstruction.
- 'Vector Addressing' is needed for the realization of the digital filters. The coefficients and the state variables of the filters are stored as vectors in areas of the coefficient and data memories respectively, and their elements must be accessed sequentially.

The tapped delayline for the state variables can be realized in two ways:

- The data are shifted physically from memorycell to memorycell; this requires N additional read/write cycles per output-datapoint for an Nth degree filter. However addressing is very simple in this configuration, an absolute or an autoincrement mode is sufficient.
- The data are shifted virtually through the delayline by means of address transformations. This is the most efficient implementation in

CH1841-6/83/0000-0427 $1.00 © 1983 IEEE

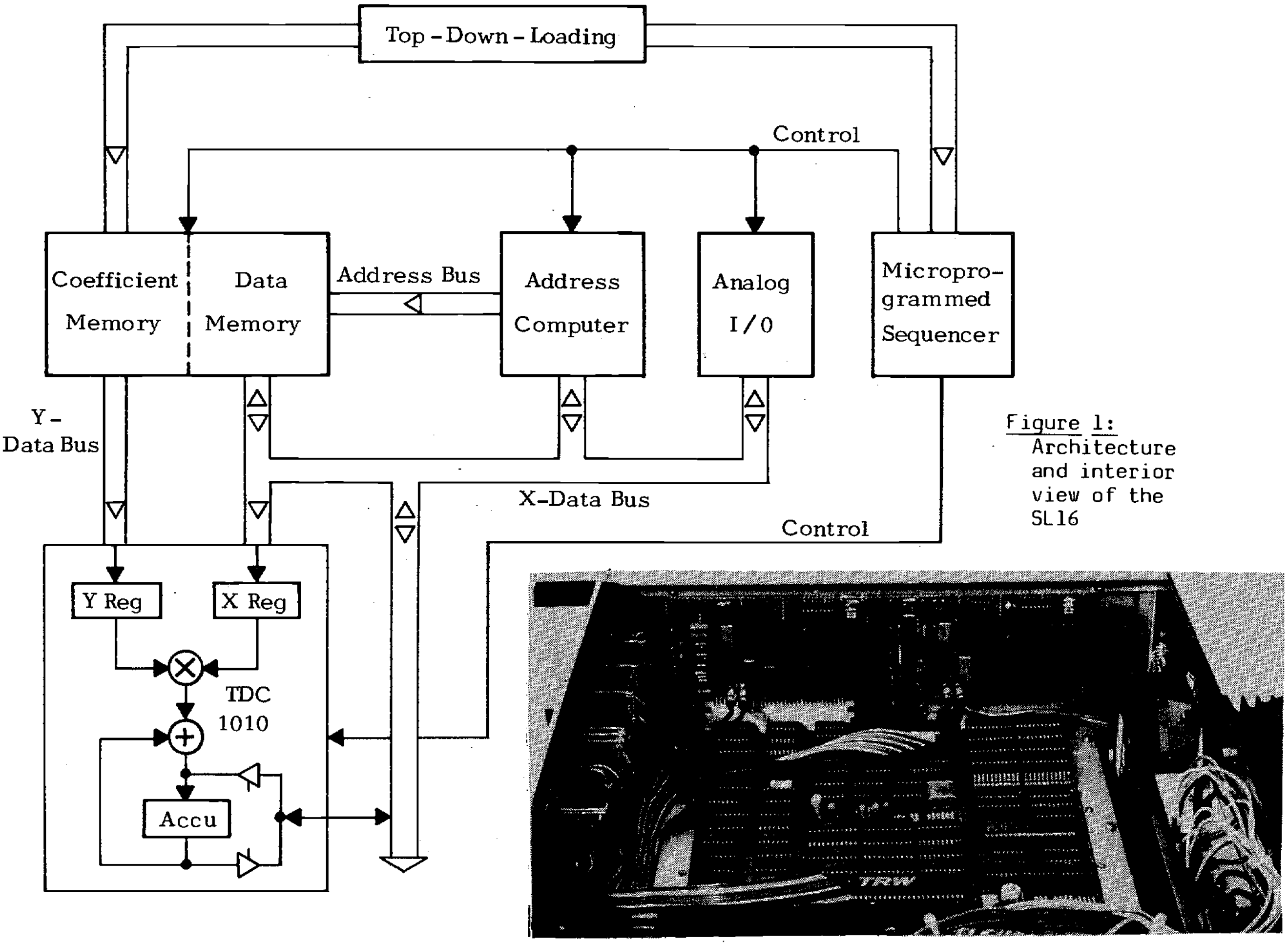


Figure 1: Architecture and interior view of the SL16

terms of the utilization of the arithmetic unit. It requires, that the delaylines are closed to endless loops with cyclic addressing, a cumbersome undertaking with most signalprocessor architectures.

The address-computer of the SL16 handles this problem with a vector-addressingmode based on the following algorithm:

```
Coefficient_Address := C_Base + Counter
Data_Address := D_Base +
                    (Pointer + Counter) modulo Size
```

'C_Base' and 'D_Base' are the startaddresses of the data and coefficient areas, and 'Size' is the number of elements in the area. All three are loaded with the 'Load Address-computer' microinstruction. 'Pointer' is the distance between the data areas startaddress and the momentary address of the delaylines first element. It can be decremented for moving the data in the delayline, and it can be stored in the scratchpad area of the data memory. The 'Counter' is cleared, when the address-computer is loaded and incremented after each computational cycle until the end of the vectors is reached. The hardware realization of this algorithm is simplified by the following two restrictions:

- 'Size', the length of the delayline, is rounded to the next highest power of two. This means, that more data memory cells are used than required by the algorithm.
- The 'Base'-addresses must be multiples of 'Size'. This can always be achieved by sorting and storing the data- and coefficient areas in the order of decreasing size.

With these restrictions two additions and the modulo operation in the addressing algorithms can be substituted with bit-by-bit logical operations:

```
Coefficient_Address := C_Base or Counter
Data_Address := D_Base or
                    (Pointer + Counter) and Size
```

A slightly different addressing mode is provided for accessing specific elements in a delayline, for instance the center element in a hilbert transformer. In this mode the autoincrement 'Counter' is substituted by an absolute 'Offset' from the microprogramm.

The performance of the processor is characterized by the following examples:

Cycletime	200 ns
N-tap FIR-filter	N+2 cycles
Nth order IIR-filter	2N+7 cycles
64 tap FIR-filter	13.2 µs
2nd order IIR-filter	2.2 µs

10.2

The processor consists of about 120 integrated circuits, mostly Low Power Schottky TTL, and consumes 3 Amperes at 5 Volts. The instruction and the coefficient memory can be loaded either top-down from a host or locally from an EPROM.

Programming Language

In order to make the programming of the machine simple, we have defined a language for the description and concatenation of single- and multirate digital filters. The syntax of the language closely resembles PASCAL, however its scope is much more restricted. The following datatypes are provided:

```
CONST  Constants, e.g. for scaling
VAR    Variables, e.g. for intermediate results
COEFFICIENT Coefficient vectors for filters
```

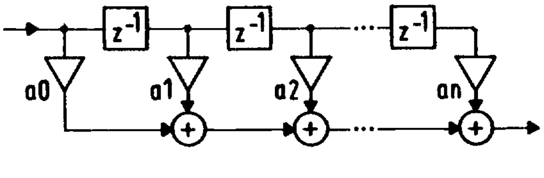
The elements of these three types are 16 bit fixed-point numbers between +1 and -1.

```
FLAG   Boolean variable for controlling the
          programmflow.
```

The analog inputs and outputs are predefined VAR-variables with the names INPUTA, OUTPUTA, INPUTB and OUTPUTB.
The following controllstructures are provided by the language:

```
BEGIN .. END   delimits a block
BEGINLOOP ..ENDLOOP  delimits a block, that is
      repeated forever. The processor synchroni-
      zes to an external event (sampleclockpulse)
      at BEGINLOOP
GOTO label
IF (NOT) flag THEN .. ELSE ..
FOR index := .. TO/DOWNTO .. DO ..
```

Due to the limitations of the arithmetic unit only three operators are allowed: ´+´, ´-´ and ´*´. The language does not provide procedures, since the hardware would be inappropriate for the required parameter passing. FIR- and IIR-filters are predefined functions.
The following examples shall demonstrate some of the features of the language. Note that the memory area for the delaylines, called SILOs, is reserved implicitly by the filterdefinition in the declaration part. The first two programms describe an FIR- and an IIR-filter:

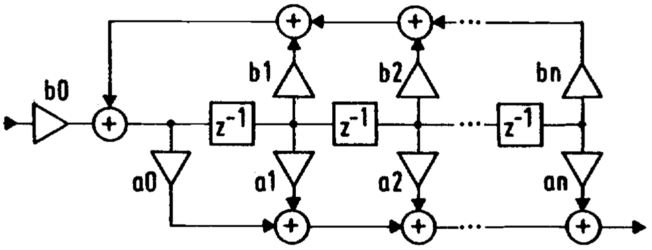


```
PROGRAM firtest;

COEFFICIENT (* defines a coefficient vector *)
   bandpasscoeff
      0 : 0.31562E-2,  1 : -0.17334E-2,
      (* and so on *)  28 : 0.31562E-2;

FIRFILTER bandpass (* defines a filter *)
      COEFF = bandpasscoeff;
      SILOLENGTH = 29;
   END;

BEGINLOOP (* Program body: filtering *)
   OUTPUTA := bandpass (INPUTA);
ENDLOOP.
```



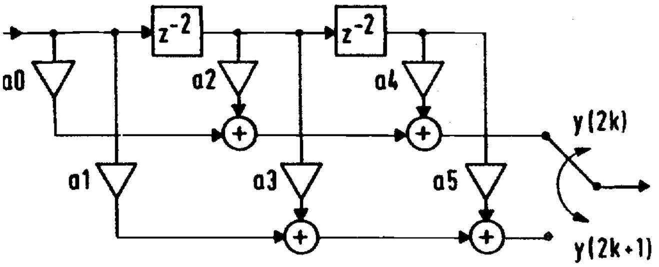
```
PROGRAM iirtest;

COEFFICIENT
   lowpassnumerator
      0 : 0.54946, 1 : 1, 2 : 0.54946;
   lowpassdenominator
      0 : 0.19576, 1 : 0.96285, 2 : -0.43906;

IIRFILTER iirlowpass
   NUMERCOEFF = lowpassnumerator;
   DENOMCOEFF □ lowpassdenominator;
   SILOLENGTH = 2;
  END;

BEGINLOOP
   OUTPUTA :□ iirlowpass (INPUTA);
ENDLOOP.
```

For the realization of an interpolator with an interpolation-factor of n the coefficient-vector and the SILO must be split into n sections; this arrangement is called a polyphase filter. The splitting is done automatically by the compiler, when an interpolationfactor is included in the filterdefinition:



```
PROGRAM interpolatortest;

COEFFICIENT interpolatorcoeff;
   0 : 0.05603, 1 : 0.25708, 2 : 0.39990,
   3 : 0.25708, 4 : 0.05603;

FIRFILTER interpolator;
   COEFF = interpolatorcoeff;
   SILOLENGTH = 4;
   INTERPOLFACT □ 2;
  END;
```

```
BEGINLOOP
   interpolator := INPUTA;
   FOR i := 0 to 1 do
      OUTPUTA := interpolator (i);
ENDLOOP.
```

In the case of decimators several inputsamples are shifted into the SILO before an outputsample is computed. This can be expressed as follows:

```
PROGRAM decimatortest;

CONST decimationfactor = 2;

COEFFICIENT decimatorcoeff;
   0 : 0.05603, 1 : 0.25708, 2 : 0.39990,
   3 : 0.25708, 4 : 0.05603;

FIRFILTER decimator;
   COEFF = decimatorcoeff;
   SILOLENGTH = 4;
  END;

BEGINLOOP
   FOR i := 1 to decimationfactor do
      decimator := INPUTA;
   OUTPUTA := decimator;
ENDLOOP.
```

By concatenating several of these basic filters multi-stage multirate filters can be described easily. For instance the program for a Hartley-modulator including 2 IIR-decimators and 6 FIR-interpolators consists of 120 lines of declarations (mainly of the coefficient-values and the filters), and a 20 line program-body defining how the filters are linked.

Compiler

The compiler has two tasks:

- syntactic checking of the sourceprogramm for formal correctness, and
- translation of the sourceprogramm down to the microcode level.

The compiler is written in PASCAL and runs on a PDP-11/04. The frontend, scanner and parser, were taken with some small modifications from /5/. Extensive error-diagnostics with usefull messages were included in order to help the unskilled user. The first phase of the compiler generates an intermediate language form of the program and a memory map for the data and coefficient memories. The second phase generates the microcode in two passes. Its output is in a symbolic form and includes automatically generated comments. Finally a microassembler, that can be used for handwritten code as well, translates the symbolic code down to the bit-pattern level. The compiler is quite simple and straightforward. It was defined, written, debugged and documented by a graduate student within three months. Through a strict limitation of the machines architecture and the elements of the language very good performance of the system could be achieved: The code generated by the compiler is only marginally less efficient than handwritten code. This is mainly due to the fact, that the compiler produces optimal code for the filters which consume most of the processor time. The code is non-optimal for the controllstructures and the passing of variables between filters. In the forementioned example of a Hartley-modulator the runtime per inputsample of the high level language program was still only about 10% higher than that of an optimized handwritten microprogramm. The time for writing and debugging the program on the other hand is at least an order of magnitude lower for the high level language.

Acknowledgements

Most of the work presented in this paper was done by two of our graduate students as their diploma-thesises: Wolfgang Stahl designed and built the processor /1/ and Lothar Baier wrote and debugged the compiler /2/.

References

/1/ Stahl: Entwurf eines schnellen Signalprozessors für "Multirate-Filtering"; Diploma-Thesis D1896ÜT, TH Darmstadt, 1982.

/2/ Baier: Definition einer Hochsprache und Implementierung eines Crosscompilers für den Signalprozessor SL16; Diploma-Thesis D1959ÜT, TH Darmstadt, 1982.

/3/ Crochiere, Rabiner: Interpolation and Decimation of Digital Signals - A Tutorial Review; Proc. IEEE, March 1981, pp. 300-331.

/5/ Wirth: Compilerbau; Teubner, Stuttgart, 1981.

/6/ Nishitani et al.: LSI Signal Processor Development for Communications Equipment; Proceedings ICASSP 1980, pp. 386-389.

/7/ Darlington: On Digital Single-Sideband Modulators; IEEE CT, August 1970, pp. 409-414.