Proceedings of The Institute of Acoustics

THE FERRANTI SINGLE CARD DIGITAL SIGNAL PROCESSOR

D.Rawson-Harris

Ferranti Computer Systems Limited
Bird Hall Lane, Cheadle Heath, Stockport

ARCHITECTURE OF CURRENT NAVAL SONARS

The Signal Processing of contemporary naval sonars can be accomplished by an "open loop" architecture in which data passes from the array, through the analogue front end, the digital beamforming, band shifting and filtering, spatial, temporal, or frequency domain processing, normalisation and associated processing, and out to data and display processing equipment.

Each of these signal processing functions is largely independent of the others. The data for each function arrives in a well defined order, with well defined block sizes, rates, significances, and so on. There is a certain amount of variation of algorithms within each function, but the control can be achieved by simple mechanisms such as tag words or select lines which are set in step with the flow of data. Controls can be generated in each function for transmission further down stream as necessary.

The algorithms are dominated by the multiply-accumulate operation, with data and coefficients being multiplied in some cases, or data and data in others. Some algorithms involve running averages, either over data points at constant time or over data points at different times, or division of data by data, and so on, but they are in the minority.

The signal processing algorithms for a given sonar system can be provided by a small number of different cards. Some have multiplier accumulators and local stores, to perform the signal processing functions. Store cards which can buffer data for intervals from milliseconds to seconds are needed at certain nodes. Interface cards, sequencers or controllers, monitors, and so on are needed to support the processors and allow the construction of complete sonar systems.

Data flows within the system from one card, or group of cards, performing one function, to the next, by point to point links, during a transfer cycle, and is processed during intervening processing cycles.

A typical example would be the 2 octave narrowband sonar, shown in the above terms in Figure 1.

FERRANTI D S P

ALGORITHMS OF CURRENT NAVAL SONARS

From the point of view of designing cards to perform the signal processing functions, the functions fall into three fairly well defined classes:

A) High data rates, heavy processing loads, almost trivial algorithms and controls;

B) Moderate data rates and loads, more elaborate algorithms and controls;

C) Low data rates, moderate to low loads, complicated algorithms and controls.

Good examples of A) are time-domain beamforming, complex band shifting, and FIR filtering. Special purpose cards for such functions are easy to design and commission, and offer great processing power per card.

B) is typified by the discrete Fourier transform or Curtis's Vernier Frequency interpolation algorithm. It is possible to design fixed logic cards for these functions, but they will not offer significantly more processing power per card than a more general purpose card can offer, and can be difficult to commission, or modify for changing requirements.

C) applies for example to passive cross-correlation and associated processing, to normalisation and trend removal, amplitude and fine-bearing extraction, and other "back-end" algorithms. It is very hard to design fixed logic cards capable of all these functions, and the rapidity with which the specifications and algorithms are changed makes it uneconomical even when it is possible.

FERRANTI DIGITAL SIGNAL PROCESSOR ARCHITECTURE

The Ferranti single card DSP is designed to meet the requirements of algorithms of group C), within the environment described in general terms above.

Briefly, it comprises a 16-bit multiplier-accumulator and a 16-bit arithmetic-logic unit. There is a data bus for each input of the MAc, and a third bus for the ALU. Each MAc bus has an I/O port, a data RAM store with its own address generator, and a link to the ALU bus. The ALU bus has its own I/O port for either data or control, a separate control port, a loop counter, a PROM store for preset data, and registers for use by the ALU in controlling the address generators and the programme sequencer.

The programmer is provided with a model of the DSP, Figure 2. It shows the RAMs and the preset and program PROMs, all the registers available to the programmer, and the programmable "devices": XBUS, YBUS, ABUS, MAC, ALU, XAG (X RAM address generator), YAG, PSEQ (program sequencer), S (synchroniser for data transfers). The arrows show the allowed directions of data flow, but bi-directional arrows can only be used in one direction at each instruction. Actually, there are registers within the ALU, XAG, YAG, and PSEQ which are also available to the programmer, and subsidiary models to explain the actions of these devices.

There are no devices specifically for data I/O, such as DMA or FIFO buffers. Data transfer is controlled by program during the transfer cycle.

The DSP is programmed at microcode level, and the card is so designed that each process functions in a single beat of the clock. Thus, each instruction refers to one operation in each process, and for each process it specifies a source register, an operation, and one or more destination registers. Within the ALU there can be two source registers in some operations. Each programmable device must be controlled at each beat, including the PSEQ, which handles conditional and similar tests. Thus, the programmer can set up pipelined structures which are close to the ideal for any given algorithm, subject to the restriction that only one MAc and one ALU are available.

PROGRAMMING SUPPORT

The DSP is programmed at microcode level because anything else is impractical within a single card. Thus, it is essential that adequate support be provided, and that good programming conventions be adopted, to eliminate most of the trouble which usually arises in microcode programming.

The programming is done on a proforma which supports the processing pipelines that are natural with the DSP. Each programmable device has a column down the page, and all columns are divided into rows across, with each row specifying the operations on the card at a given beat. Thus, a given data word can be seen to move from register to register, through the various processes, across and down the page, just as it can be on the model. The mnemonics also have been chosen to represent the flow of data from source, through process, to destination, in a more natural way than is usual. Thus, they refer to the register names explicitly, rather than as a subset of a combination of symbols. For example, in the ALU, the DSP operation R31 - ACC → R31 is represented by the manufacturer's mnemonics TORAA SUBS 31. In this particular case, for example, it is difficult in practice to remember whether it ought to be SUBR instead of SUBS.

Thus the programmer need learn only the model and a limited amount of extra material, and finds that the mnemonics represent the data flow and processing in natural ways.

FERRANTI D S P

EXAMPLE OF A DSP ROUTINE

The example shows the accumulations of the complex cross correlation coefficient and of the beam power for 32 successive beams, including reading the complex half beam samples. The accumulations are both held as double length (MS product and LS product) numbers, MS in X RAM and LS in Y RAM. Real parts are held in SEGØ, and imaginary parts in SEG1, at common addresses.

The bases of the data areas are extracted as immediate operands (I) and used to initialise the address generators in instructions Ø to 4, and the Loop Counter is also set up. The address generator stacks are set up conveniently in 5, which also reads the first data word (Left half beam real sample, LR). Since this is not a loop operation (due to the address generator operations), loop return is to 6 and the sequencer stack is set up in 5. The instructions 6 to 9 read the beam samples, store them in the ALU, and set up the MAC to do

Real Cross Corr Coeff:= Real Cross Corr Coeff + (LR*RR + LI*RI).

The current value of the coefficient is preloaded in 2, and the new value is stored in 10, at the same address. Then, 11 to 15 do the corresponding imaginary component in the same way. The address is incremented now in preparation for reading the next beam's worth of data.

Instructions 16 to 20 preload the current beam power value, add the new contribution (LR+RR)**2 + (LI+RI)**2 and store the new value at the same address. The real component is formed in 8, and the imaginary in 17.

The loop counter is decremented (D) in 6, and tested in 17. If it is not zero, LNZ gives a 1, which selects 23 after 21, whereas if it is zero, LNZ gives a zero, which selects 22 after 21.

At 21, the preload for the next beam is done, as was done in the loop header for the first beam, and at 23 the first input for the next beam is done. The programme sequencer selects TOS in 23, which returns to 6 or 7. (The "source" in the PSEQ defines bits 1 to 10 of the next instruction's address, and "condition" determines bit Ø.) Since condition in 23 is zero, 6 is the return address.

FERRANTI D S P

PERFORMANCE IN TYPICAL APPLICATIONS

There are various aspects to performance in the context of sonar sets and similar equipments.

In terms of computing power, the DSP does a 1K complex DFT in about 12 msec, and with 2 msec for transfer, it can do 64 such DFTs per second. It can use the 1K routine for 2K and 4K complex DFTs also, and a subset for 256 point DFTs, and so on.

In terms of efficiency compared with a card of dedicated logic, in an FIR application for example, the DSP in FMS 12 does 128 point filters on 64 channels every 1.95 msec, including input-output. At 200 nsec per cycle, the MAc is used for 84% of the time, on average.

Clearly, a dedicated card could provide two MAcs for this purpose, which leads to a further consideration of performance. The FIR code for the processing cycle is four instructions for the main loop, and five instructions including loop exits for the header. Thus, there is scope for other routines in the program store of the filter DSP. In fact, in the FMS12 two octave narrowband sonar, there are eight DSP's: 2 HF and 1 LF beamformer, 1 HF and 1 LF filter, 1 DFT, 1 Vernier, 1 "back-end", and there is enough room in PROM for the code of all of these to fit into every DSP. Thus, in principle, one part number and a single spare would do for all eight functions. (Other considerations lead to three stock numbers as the best compromise, in this case.)

Again comparing with dedicated cards, there is the time required to program a DSP implementation compared with designing a hardware implementation. The FMS12 filter code was written as a training exercise by the programmer who later wrote the "back end" code. The FMS 12 beamforming code, including an initialisation pass which calculates the offsets and coefficients of interpolation from the array parameters, speed of sound, and beam axis directions, was written in six weeks by a degree mathematician, as his first job after leaving University. The 1K DFT was written by another degree mathematician, also on her first job, from the algebraic treatment of the particular method which is best suited to the DSP, and from specimens of code of the inner loops of the various passes. Including finding the mistakes in the algebra and the specimen code, developing the other sizes of DFT, and writing VAX test routines, it took about four months.

FERRANTI D S P

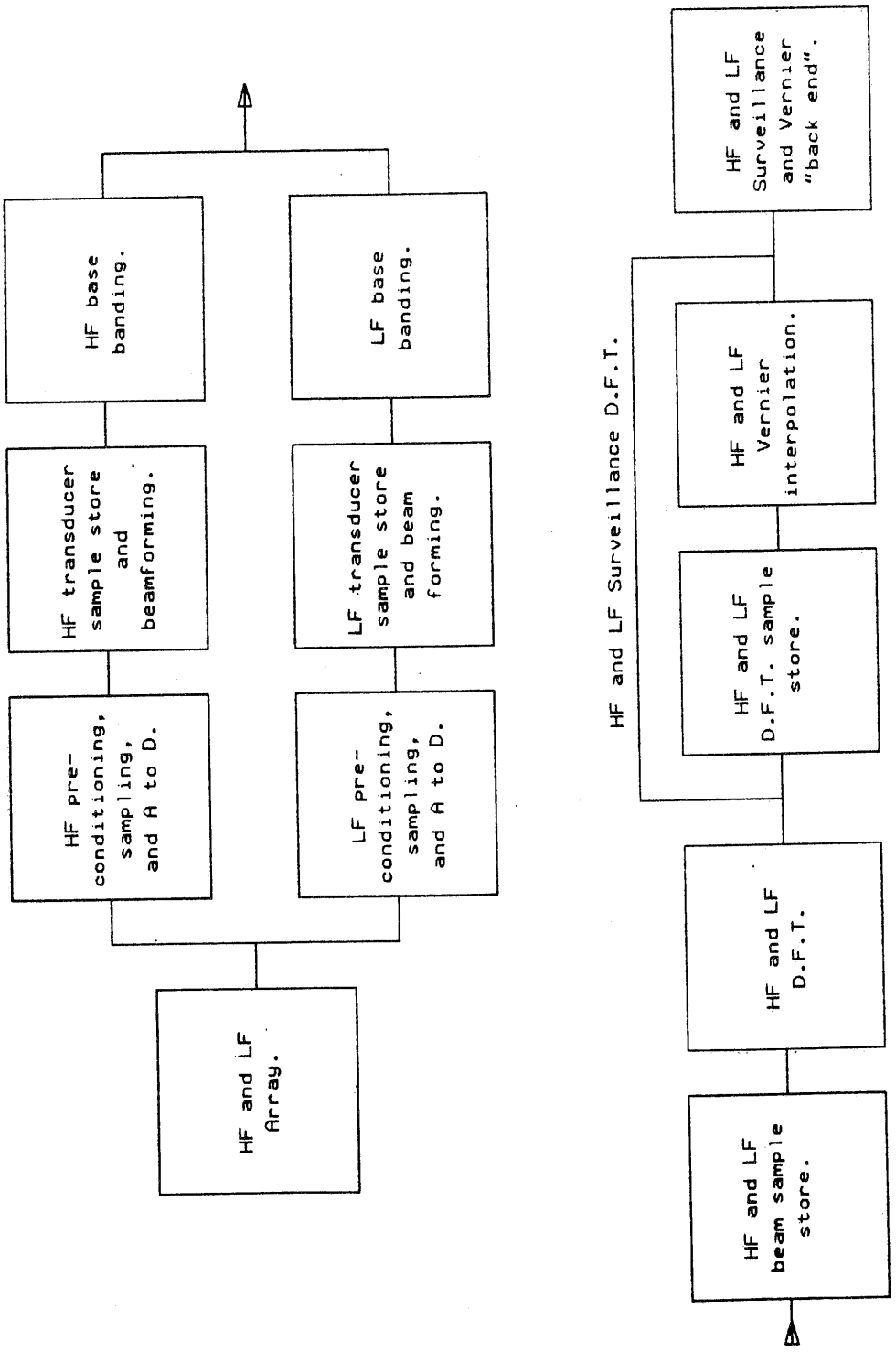


FIGURE 1

FERRANTI D S P

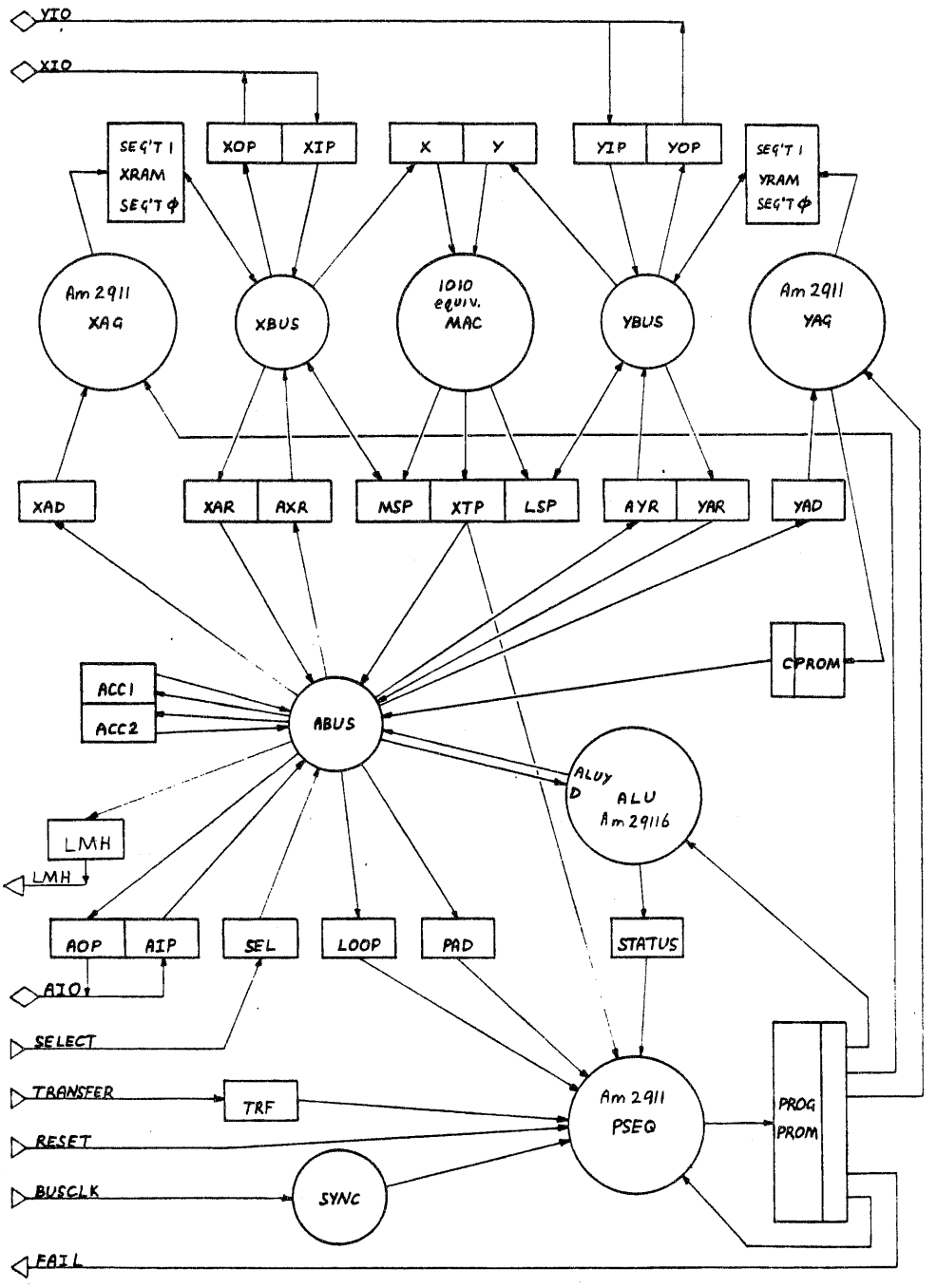
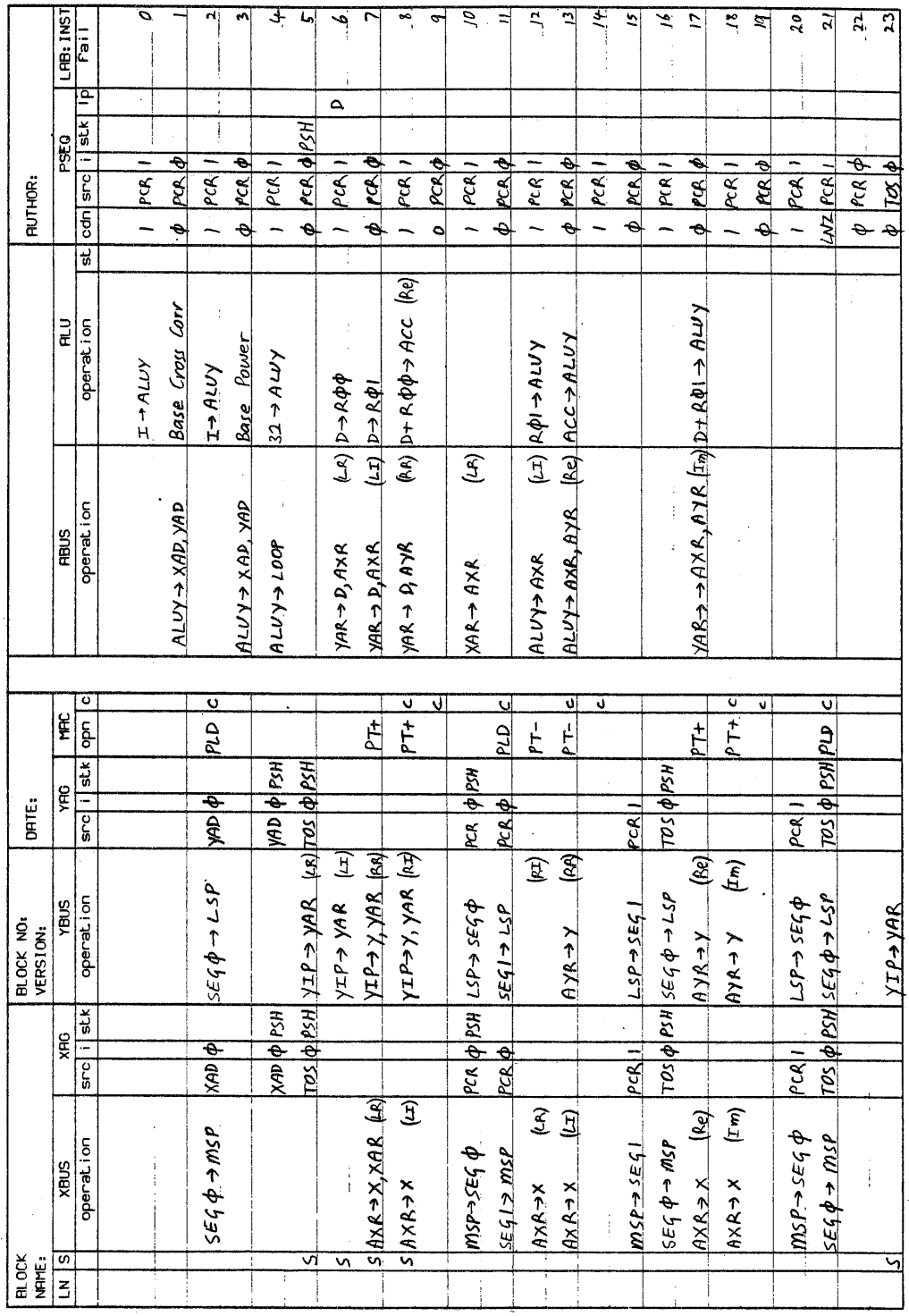


FIGURE 2

FERRANTI D S P



EXAMPLE OF DSP CODE