THE COMPLEX VECTOR PROCESSOR (CT-CVP/100).

1. INTRODUCTION.

The CVP is a compact, programmable digital signal processing (DSP) engine for high throughput, front end signal processing operations, such as beamforming, filtering and spectral analysis.

The processing module integrates two basic functions:

i. a peripheral components interconnect (PCI) interface, containing a high-speed PCI bus interface and some control logic

and

ii. a DSP module based on a high density FPGA and fast static random access memory.

Currently, the CVP can be used in any system that supports a commercial PCI bus interface, for example, most Intel based PCs, etc. A CompactPCI version is also available - Compact PCI is an industrial extension of the open PCI standard that uses standard interface chips on a board with VME form-factor (either 3U or 6U) to provide a low-cost, high reliability system, similar to VME but with higher bus bandwidths.

PCI busmaster device drivers have been developed that allow the CVP module to be used with several PC operating systems (e.g. DOS, Windows 95, Windows NT4, Windows 2000, etc). The processing engine has a pass based architecture, where data flow and processing operations are set up to process a large block of data. This allows very straightforward processing software to be developed from a variety of high level languages (e.g. from C, Visual C++, Pascal, Delphi, etc), for example only 4 lines of Delphi code are needed to program the CVP to calculate a complex 64k-point fast Fourier transform (FFT). Typical transform speeds in the range 10 to 60 microseconds for a 1024 point complex FFT are realised, depending on the exact CVP configuration chosen - this procesing time includes windowing the incoming time-series data and log-magnitude extraction of the processed frequency domain data block.

2. MODULE ARCHITECTURE.

A block schematic of the module is shown in figure 1: the DSP unit consists of a complex, block floating point processor implemented in a Xilinx VirtexII FPGA. It is optimised for block-orientated algorithms and array processing and uses a four port bi-directional data flow architecture that allows high speed synchronous static RAM to be used for working data and coefficient storage and high speed FIFOs for input/output data queuing.

The module has been designed to handle large transform sizes relatively easily. Up to 48 megabits of memory are used to support the DSP processing - data ports A,B and C each interface to up to 256k complex words of fast static RAM, whilst the Q port interfaces to a 256k complex word bi-directional FIFO for data I/O.

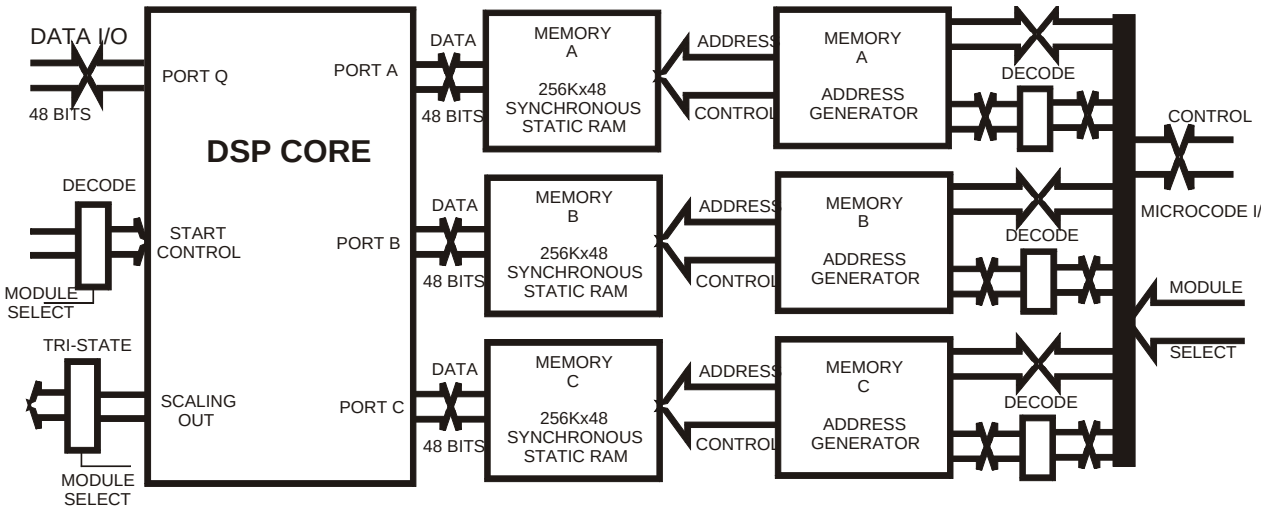

Figure 1 - Module Schematic

The flexibility of the data routing on the module allows the use of synchronous SRAM and FIFOs, rather than dual port memory. This allows much larger memory sizes and faster access times than previous systems. Processing block sizes up to 256k-point complex can be handled directly.

The DSP core is a pass-based processor where each function opcode and directional data flow opcode is valid for one complete pass of the data. Each function opcode requires a related data flow opcode that specifies the data source and destination. This data flow opcode defines how data moves internally through the execution unit - data can be routed though any of 12 available data paths.

The execution unit handles the high level processing functions such as filtering, spectral processing, correlation, modulation and de-modulation. The power of the execution unit lies in its ability to perform transforms very efficiently, particularly radix-16 based FFTs. The opcodes that the unit implements are compact and operate at the macro level. The execution unit supports DSP, complex arithmetic, vector arithmetic and a number of general purpose functions as embedded operations. The unit supports either fixed (24 bit real + 24 bit imaginary) or block floating point arithmetic (24 bit + 24 bit + 8 bit block exponent). A table of the data flow paths and of the execution unit opcodes are shown in Tables 1(a) and 1(b).

Programmable address generators, also implemented in the FPGA, drive the A. B and C memories (see Figure 2) - these provide the address sequencing for the different address patterns for processing block sizes from 2 to 256k points. Table 2 outlines the types of addressing sequencing supported.

Data is fed to and from the unit via the PCI bus. The use of a bus-master PCI interface allows data to be written and read from the unit at up to 132 Mbytes/second: 256k complex bi-directional FIFOs are used on the CVP unit so that data loading/unloading and processing can be performed concurrently.

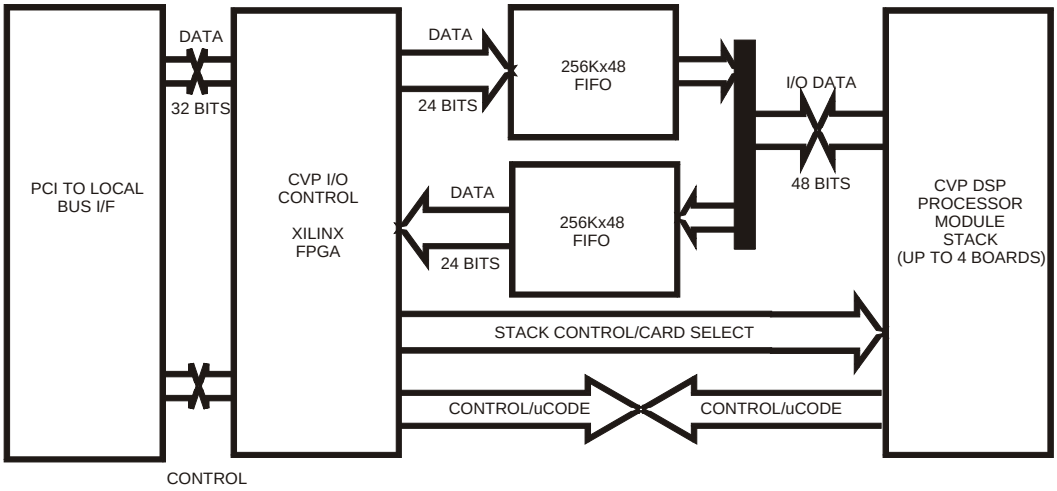

Figure 2 - Processing Card Schematic

The CVP/PCI interface has been designed to be data flow driven - when there is sufficient input data in the input FIFO and enough room for results in the output FIFO, then the process is triggered. The process flow is defined using a tag or token in the input data stream block header to point to a sequence of opcodes and direction control codes previously loaded into the control FIFO: hence the system is most efficient when processing multiple channels of data, for example for LOFAR processing, where processing parameters, for example vernier zoom, are controlled via the tag on a channel-to-channel basis.

Examples of some high level system software are given in Table 3.

MNEUMONIC	DATA FLOW
RAWB	read from port A, write to port B
RAWC	read from port A, write to port C
RAWQ	read from port A, write to port Q
RBWA	read from port B, write to port A
RBWC	read from port B, write to port C
RBWQ	read from port B, write to port Q
RCWA	read from port C, write to port A
RCWB	read from port C, write to port B
RCWQ	read from port C, write to port Q
RQWA	read from port Q, write to port A
RQWB	read from port Q, write to port B
RQWC	read from port Q, write to port C

TABLE 1(a) - DATA FLOW OPERATIONS AND MNEUMONICS

OP-CODE	MNEUMONIC	PROCESSOR OPERATION
$00	BFLY16	Radix 16 Butterfly
$01	BFLY4	Radix 4 Butterfly
$02	BFLY2	Radix 2 Butterfly
$03	{RESERVED}	
$04	{RESERVED}	
$05	{RESERVED}	
$06	{RESERVED}	
$07	BRFT	Dual real FFT separation pass
$08	{RESERVED}	
$09	{RESERVED}	
$0A	{RESERVED}	
$0B	{RESERVED}	
$0C	CMAG	Complex magnitude
$0D	CMUL	Complex multiply
$0E	{RESERVED}	
$10	CADD	Complex Add
$11	CSUB	Complex Subtract
$12	VMUL	Vector multiply
$13	VABS	Vector absolute value
$14	{RESERVED}	
$15	{RESERVED}	
$16	{RESERVED}	
$17	{RESERVED}	
$18	{RESERVED}	
$19	VPAS	NOP
$1A	{RESERVED}	
$1B	{RESERVED}	
$1C	MOVD	Move data from (A,B or Q) to (A,B,C or Q)
$1D	MOVC	Move data from C to (A,B or Q)
$1E	VMXM	Vector maximum and minimum
$1F	LOOP	Re-start control sequence

TABLE 1(b) - PROCESSOR OPERATIONS AND MNEUMONICS

The CVP can support address patterns which generate the following types of addressing sequences:

Radix 2, radix 4, radix 16 and mixed radix FFTs

Digit reverse for FFTs

Double length real data only FFT separation pass

Two at a time real data only FFT separation pass

Decimation

Interpolation

Modulo increment/decrement

Circular buffering

Table 2 - ADDRESS GENERATION OPERATIONS

{DumpCode(Operation,Dataflow,AAddress,BAddress,CAddress)}

```
DumpCode(BFLY16,RAWB,BF160   ,BF160   ,TF160   );
DumpCode(BFLY16,RBWA,BF161   ,BF161   ,TF161   );
DumpCode(BFLY16,RAWB,BF162   ,BF162   ,TF162   );
DumpCode(BFLY16,RBWA,BF163   ,BF163   ,TF163   );
```

Table 3 - DELPHI CODE TO PROGRAM BCVP FOR 64k POINT FFT

This code controls WINNT device driver and sets up processor address generators and control logic for the four passes used to perform the transform. Once the micro-code has been loaded, when sufficient data is fed to the Q input store and there is sufficient room in the Q output FIFO for the processed results, the operations defined by the code operate in sequence. So 64k points of data are read from the Q input FIFO and written to RAM store A in digit reverse order on the first data pass, operations 1 through 4 perform the four passes of the 64k point, radix 16 FFT, with data ping-ponged between RAM stores A and B, and the complex spectral data read from RAM store A and written to the Q output FIFO on the last operation.