ACTS Projects SUNBEAM (Smart Universal Beamforming) Deliverable
This work is partially sponsored by the European Commission

Title:	Reduced Complexity Processing Elements for SMART antenna systems
Document Number:	AC347/UoB/A82/DS/P/013/b1
Document Type:	Public Report
Workpackage:	A8.2
Organisations:	UoB
Authors:	Nigel Kingswood
Circulation List:	All partners
Date:	8th November 1999

Abstract

This report outlines ways in which the complexity of the computation in the beamformer could be reduced. The first simplification necessary is the avoidance of floating point numbers. These complicate the calculation process and make the optimisations described in this report difficult or impossible to apply.

The beamformer can be implemented as a DSP based solution or a full hardware implementation. The trade-offs between these two approaches are discussed, in particular the flexibility of the DSP approach.

Possible DSP optimisation techniques are outlined, in particular the use of instruction pipelining. This permits the overlapping of different iterations of a loop to provide a considerable increase in performance. This is particularly useful for tight inner loops that are at the core of many beamforming algorithms.

Hardware optimisations that may be applicable are then described. These take various approaches to reducing complexity.

The first, used in the 3CM method, is to reduce the number of multipliers. This is an advantage for hardware implementation, but whether this is suitable for DSP implementation depends on what extra savings can be made: these are architecture/algorithm dependent.

One of the operands to the multiplication can be simplified using a Non-Uniform number space. A non-uniform number space does not contain all possible values so the operand is approximated to the nearest available value. This considerable simplifies the layout of a hardware multiplier.

Bit-parallel multipliers have traditionally been used for high performance and bit-serial ones for low-speed applications. Between these are digit-serial architectures, which can be created systematically from bit-serial ones. With the addition of bit-level pipelining they offer performance close to bit-parallel architectures but with reduced hardware.

The flexibility of the DSP approach, using with the latest products that use instruction pipelining, makes it the most desirable solution. If this does not provide the required throughput the digit-serial solution should be the easiest hardware implementation since the required architecture can be built up systematically. The other approaches could give good results however more detailed work will be required since the improvements are algorithm/architecture dependant.

Feedback and comments may be directed to:
Dr Duncan Brooks, SUNBEAM Project Co-ordinator
ERA Technology Ltd, Cleeve Road, Leatherhead, Surrey, KT22 7SA
Tel: +44 (0) 1372 367055
Fax: +44 (0) 1372 367087
Email: duncan.brooks@era.co.uk

CONTENTS

1. INTRODUCTION

Previous reports [Anton98], [Mestre99] have dealt with selection of suitable algorithms for the digital beamformer and the high computational requirements of these algorithms. This report looks at how the computational complexity could be reduced to increase execution speed.

The algorithms are broadly similar in that they are designed to combine the baseband signals from the antennas to produce a desired signal while minimising the effects of interference. The differences in the algorithms lie in their ability, or otherwise, to cope with such aspects as, the type and magnitude of interference, the strength of the desired signal, number of interferers, the type signal being received and the type of protocol being used.

Whatever algorithm is chosen the speed of execution is determined by the speed of the unlaying implementation. This report looks at ways the DSP or hardware implementation can be optimized to improve execution speeds. There are a number of ways of doing this, most of which involve reducing the complexity of the implementation.

1.1 Floating point representation

The most obvious way of reducing the complexity of the implementation is in the representation of the data. For a human the most accurate and comprehensible approach is to use floating point numbers. However from an efficiency point of view this will slow computation. Each floating point number consists of an exponent and mantissa so during multiplication of two numbers each must be split in two and the mantissas multiplied together and exponents added together. The result may then need to be rescaled requiring adjustment to the mantissa and exponent before they are combined again into a single entity. This is an obviously complex process whether using a DSP or special hardware. Integer representations, either twos complement of signed integer, are much easier to manipulate and are therefore essential for rapid execution. As much of the implementation as possible should using integer representations to maximize the speed of execution. Although it may be necessary to include extra circuitry/code to convert to and from the integer representation this overhead will be more than compensated for by the increased execution speed. In the rest of the report it is assumed that an appropriate integer representation is used.

1.2 Optimization techniques covered

It is possible to implement the required algorithm using a DSP or a custom hardware implementation. The one chosen will depend on a number of criteria including flexibility and speed. Whichever implementation is chosen any optimization technique will need to use the inherent characteristics of the underlying technology. For this reason a particular technique that is appropriate for a hardware implementation may be of no use in a DSP implementation and vice versa. The constraints of the underlying technology should be bourne in mind since they can reverse the desired improvement. These points are discussed further in section 2.

Modern DSP have tools that use a variety of techniques to improve performance the chief of these being instruction pipelining. This utilises the fact that modern DSP chips contain multiple execution units to overlap execution of different iterations of loops. Pipelining together with the reduced cycle times of the latest products can produce a significant increase in execution speed. Pipelining is described in more detail in section 3.

Algorithms vary and will utilise different aspects of the overall problem to be more efficient; however these tend to be algorithm specific. To improve performance for as many algorithms as possible it is necessary to concentrate on the low-level components that all approaches share. This report therefore concentrates on optimizing multiplication, which is very prevalent in all algorithms. This aim is made more difficult because the signals involved need to be considered as complex variables since they include in-phase and quadrature components. This leads to a significant increase in complexity since a direct implementation of a complex multiplier requires four real multipliers and two real adders. Ways of reducing the complexity of complex multipliers are described in the remaining sections of the report.

One method of reducing complexity is that used in the 3CM approach which only uses 3 real multipliers at the expense of supplementary adders. In DSPs multiply and add instructions both take a comparable number of cycles and therefore there is little to be gained by using this technique. In some circumstances there may be ways to ameliorate this. This is described in section 4.

Another approach is to reduce the complexity of the data being multiplied by using Non-uniform number spaces. This basically approximates one of the operands to the multiplication to the nearest sum of powers of two. For example a 8-bit number could be approximated by the sum of say 3 powers of two. This permits the multiplication to be considerably simplified. This is described in section 5.

For maximum performance bit parallel multipliers have traditionally been used while for minimal hardware, low-performance applications a bit-serial approach has been found more appropriate. A useful compromise between the two is a digit-serial approach. Here the word is split into a small number of digits. The bits that make up each digit are processed in parallel while the digits themselves are dealt with serially. These digit-serial architectures can often be pipelined at the bit level to increase performance close to the bit-parallel approach with less hardware. This is described in section 6.

1.3 Techniques not covered

A number of other techniques were also considered but it was decided that they were not appropriate for use with the beamformer. The two most significant methods in this category were MSB First and MSB Truncation.

The MSB First technique [Harris-93] uses an array structure so is unsuitable for DSP implementation. Although well suited to hardware implementation it is clearly aimed at the design of filters with coefficients of greater than 8 bits. It may be worth consideration in other parts of the system rather than the beamformer.

The MSB truncation approach [Moshnya99] is again based on an array of processing elements and so requires a hardware implementation. Its main aim is reducing the number of bit transitions that occur during computation to reduce power rather than to increase speed so is not appropriate for the current application.

2. FPGA/ASIC VERSUS DSP IMPLEMENTATION

The key difference between DSP based and hardware implementation is the fact that DSP solutions are inherently more flexible; their functionality being changeable by a simple re-compilation. However in situations in which high performance is required they may not be able to produce the required throughput.

When deciding how to optimize the performance of a particular solution, whether DSP or hardware based, it is important to utilize the inherent characteristics of that particular implementation, i.e. there is no common set of improvements that are applicable to all solutions. As an example of this consider the use of different architectures for implementing a complex multiplier. In DSP the only way to change the implementation would be to change the equation used in the calculation. In a hardware implementation there is considerable scope for altering the way the multiplication is carried out both in terms of minising the number of the multipliers required and in simplification of the multipliers used. This trade-off is possible in hardware implementations since the area occupied by an adder is considerably smaller than for a multiplier. It is therefore possible to trade off the number of multipliers at the expense of the number of adders and still obtain a faster overall solution. If this approach was applied to a DSP solution, since both a multiply and an add take a comparable number of cycles, the overall number of operations would increase and so performance would fall rather than rise.

2.1 Underlying technology effects

The underlying technology used in a hardware solution can have an impact on the optimisations that are possible. As an example consider implementation by FPGA. Once the circuit diagram is entered the process of splitting this across the available Combinational Logic Blocks (CLBs) in the target device is done automatically. The implementation can be altered relatively easily by changing the diagram and enacting the automatic layout. In this respect FPGA implementation represents a halfway house between DSP and full hardware implementations. However a small increase in the logic used in a design, with the intent of increasing performance, can have the opposite effect [Valls99]. The reason for this is the way the FPGA is fabricated. It is created from a fixed number of CLBs which are chosen to be as flexible as possible. Elements in the entered design, such as gates, adders, even multipliers are mapped to an optimal layout of CLBs together with the required interconnect. If an element in the design originally fitted into a single CLB it can be seen that increasing the element's functionality could require it to use 2 CLBs instead of one. This increases the CLBs used but also requires extra interconnect between the CLBs which will increase propagation delays. The amount of interconnect for the rest of the design is also decreased and the routing required will be forced to take a more circuitous path around the larger 2 CLB unit. This significantly increases the propagation delay. Any changes to the overall architecture must therefore be made with a view to the way they will effect the number of CLBs used, rather than just considering the circuit diagram.

3. OPTIMIZATION OF DSP IMPLEMENTATION USING 'PIPELINING'

The most recent DSPs have multiple execution units and techniques to make the maximum possible use of these resources. One common technique is instruction pipelining. Consider a simple program that loads two pieces of data, multiplies them together and adds the results to a running total. This is repeated in a loop for a whole series of data pairs. If these instructions are executed sequentially then the time to produce a partial result, i.e. one iteration of the loop, is the sum of the execution times of individual instructions. This is illustrated in Table 3.1, where the rows represent consecutive cycles of the DSP and the columns are the execution units of the DSP, e.g. .M2 is a multiplier, .S1 is an ALU.

CYCLE	.L1	.L2	.M1	.M2	.S1	.S2	.D1	.D2
1	LDH	LDH						
2								
3								
4								
5								
6			MPY					
7								
8					ADD			
9	LDH	LDH						
10								
11								
12								
13								
14			MPY					
15								
16					ADD			
17	LDH	LDH						
18								
19								
20								
21								
22			MPY					
23								
24					ADD			

Table 3.1 DSP execution without pipelining

It is obvious that there are many cycles in which the DSP is not active and the multiple execution units are not being used effectively. Since the different hardware units and the buses can act independently (on an instruction by instruction basis) it is possible to arrange for the various units to be executing instructions from different iterations of the loop. This is illustrated in Table 3.2.

It can be seen that once the DSP is primed with instructions, cycles 1-7, a whole loop of instructions can be executed in a single cycle and even though the individual intructions may have a latency of several cycles the throughput is one per cycle. At the end of the loop it takes several cycles to 'unwind' the instructions. This technique can obviously provide considerable speed improvements provided; (a) the loop is small and the instructions that make it up can be executed independently (b) there are no branches in the loop which would disrupt the flow and (c) the number of iterations of the loop is significantly greater than the time to prime and unwind the loop.

CYCLE	.L1	.L2	.M1	.M2	.S1	.S2	.D1	.D2
1	LDH	LDH						
2	LDH	LDH						
3	LDH	LDH						
4	LDH	LDH						
5	LDH	LDH						
6	LDH	LDH	MPY					
7	LDH	LDH	MPY					
8	LDH	LDH	MPY		ADD			
9	LDH	LDH	MPY		ADD			
10	LDH	LDH	MPY		ADD			
11			MPY		ADD			
12			MPY		ADD			
13			MPY		ADD			
14			MPY		ADD			
15			MPY		ADD			
16					ADD			
17					ADD			

Table 3.2 DSP execution using pipelining

This approach is particularly useful with the Texas Instruments TMS320C62X series of DSPs. This is described in detail in [Dahnoun00].

Another technique for faster execution is loop unrolling. This is often applied by the C compiler without the developers knowledge. Consider the execution of instructions in Table 3.2, since this is a loop it would be possible to write this using a counter and a branch instruction. This, although valid, introduces extra instructions that will disrupt the pipeling of the code. Therefore instead of a loop of N iterations the code in the loop is explicitly written out N times. This obviously increases the length of the program code but permits rapid execution.

4. 3CM IMPLEMENTATIONS

In DSP work multiply and add both take 1 instruction therefore there is little to be gained by using techniques to reduce the number of multipliers since this is usually accomplished at the expense of supplementary adders. In some circumstances there may be ways to ameliorate this [Perry97].

The signals processed by the beamformer will be complex, i.e. have in-phase and quadrature components. This increases the circuit complexity since direct implementation creates a complex multiplier from four real multipliers and two real adders by straightforward expansion of equation 4.1

$$(a+bj)(c+dj)=(ac-bd)+(ad+bc)j=v+wj \tag{4.1}$$

An obvious means of reducing this complexity is to reduce the number of multipliers used. This is the approach taken by the three real multiplier based complex multiplier, (3CM) method. Equations 4.1 can be reformatted so that it only requires three real multiplications, however it now has five add/subtract operations [Perry99]. One of the possible twelve re-arrangements is shown in equation 4.2.

$$(a+bj)(c+dj)=((a+b)c+(c+d)(-b))+((a-b)d+(c+d)b)j=v+wj \tag{4.2}$$

The 3CM multiplier structure corresponding to the particular arrangement in Equation 4.2 is shown in Figure 4.1

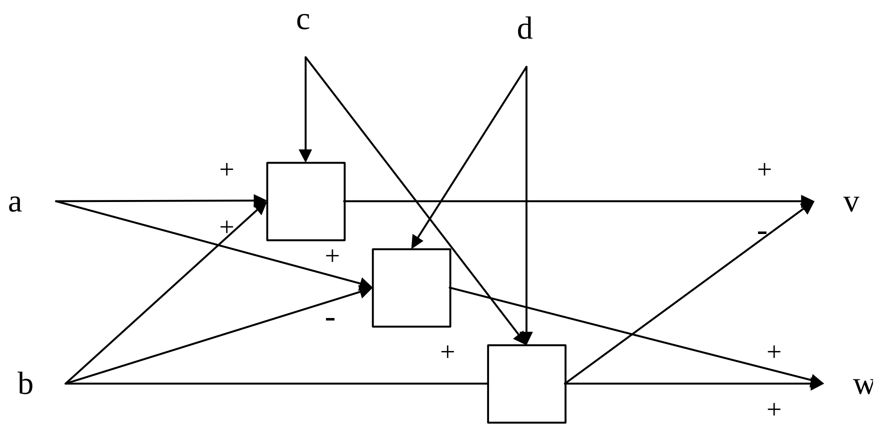

Figure 4.1 A possible structure for a 3CM multiplier

In an ASIC since adders are considerably smaller than multipliers the gain in area due to three additional adders is more than offset by the loss of a single multiplier leading to an overall reduction in silicon 'real-estate'. Smaller circuits tend to have smaller propagation delays and hence faster execution speeds.

In DSP implementations addition and multiplication operations both take a comparable number of cycles so this approach is not so fruitful. The reason for this can be illustrated by considering a DSP which executes an addition in time T_A and a multiply in time pT_A. Implementing equation 4.1 then takes time T_D.

$$T_D = 4pT_A + 2T_A \tag{4.3}$$

Equation 4.2 will require time T_{3CM}.

$$T_{3CM} = 3pT_A + 5T_A \tag{4.4}$$

The breakeven point where equation 4.2 becomes more efficient is when p>3. In most modern DSPs the value of p is usually 1 or 2 so it is obvious that the technique, in isolation, would not be suitable for DSP implementation. However in some situations, dependant of the algorithm being implemented, it is possible to overcome this overhead. An example of this is given in [Perry97]. Consider a generalized complex adaptive filter. Neglecting the calculation of the error term the filter consists of a complex multiplication of input data by coefficients and the multiplication of input data by an error term to calculate the coefficients. A 3CM-based implementation is shown in Figure 4.2.

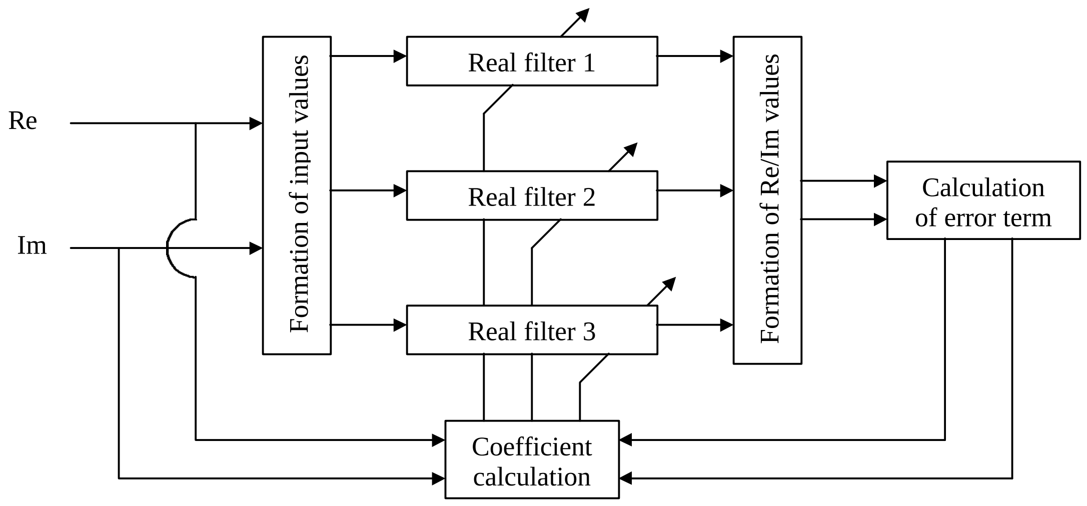
Re
Im
Formation of input values
Real filter 1
Real filter 2
Real filter 3
Formation of Re/Im values
Calculation of error term
Coefficient calculation

Figure 4.2 Generalised 3CM based adaptive filter

A conventional implementation would require 8 multipliers and 4 adders per coefficient update. Implementing this using 3CM would at first sight require 6 multipliers and 10 adders. However improvements are possible by (a) sharing the formatting of the input data for the filter and coefficient calculation and (b) framing the coefficient calculation in terms of 3CM inputs. This reduces the implementation to 6 multipliers and 6 additions per coefficient update and is illustrated in Figure 4.3.

This design now uses the same number of operations as the conventional approach so a DSP implementation would be worth investigation

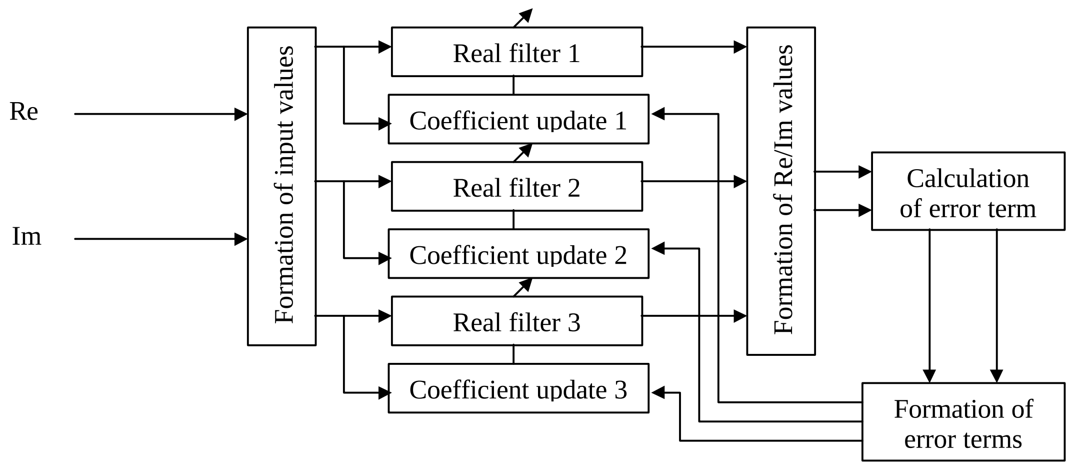

Figure 4.3 Modified 3CM-based filter

Another interesting aspect of this example is that because there are now 3 signal paths between the coefficient calculation and the filter it creates the possibility of the three filters being adapted independently of one another. Thus the application of the 3CM approach has lead to the possibility of a completely new algorithm. How well this can be applied to the beamformer depends on the algorithm chosen since the changes described in the example above are algorithm/architecture dependent.

5. NON-UNIFORM QUANTISATION OF DATA

In the previous section the reduction in complexity was tackled by reducing the number of multipliers. In this section a technique is described to reduce the complexity of the data rather than reduce the number of multipliers. This is accomplished by using a non-uniform number (NUN) approximation of one of the multiplicands. Obviously in a situation where one of the multiplicands is fixed this is easily implemented and can be optimized for minimal performance degradation. In adaptive beamforming both multiplicands will be changing, so it is not possible to do off-line optimization, and one must be chosen to be represented by non-uniform quantization. This requires a quick and efficient means of converting one multiplicand from twos complement representation to the NUN form.

A simple example of a NUN is the signed power of two (SPT) representation. A two's complement number *x* which is B bits long can be expressed as a sum of signed powers of two as in equation 5.1

$$x_N = \sum_{i=1}^{N} s(i)2^{g(i)} \qquad \mathrm{s(r)} = -1,0,1 \tag{5.1}$$

where *g(i)* is the power of the *ith* term and N is the number of power-of-two terms used to represent *x*. If *x* is an integer and $N = \lceil B/2 \rceil$ all integers in the range $-2^{B-1} \ldots 2^{B-1} - 1$ can be represented exactly. If $N < \lceil B/2 \rceil$ not all values of x can be represented exactly by x_N and so the SPT representation is an approximation to the two's complement number. Several other NUN schemes exist such as Constrained Signed Power of two (CSPT) and Constrained Power of two (CPOT).

Non-uniform multipliers consists of several shifters, one for each of the POT terms, and the twos complement multiplicand is shifted in each of these and the results summed together. If the sign of the POT term is negative then the multiplicand is shifted, inverted and one added. This basic layout can be simplified even further by using similarity to Booth encoded multipliers.

By choosing this approximation one of the multiplicands is reduced to a (small) number of N shift and add operations rather than having to do this for all B bits in the original multiplicand. This saves both silicon area and execution time. However it must be offset against the need to represent the power-of-two terms used, converting the multiplicand to a NUN and the error created by not using the exact multiplicand.

Provided the largest negative value representable, 2^{B-1}, is approximated by $2^{B-1} - 1$ it can be shown [Perry97] that the 3 term SPT number can be represented in a wordlength similar to the two's complement version for B >= 12.

Work has been done by several people e.g. [Chen96] on making the conversion from two's complement form to NUN form as efficient as possible. The algorithm to calculate the optimal choice of SPT terms, if it doesn't have some form of look-ahead functionality, uses a ripple carry that might ripple the carry across the entire word. This limits the conversion speed. A commonly used adaptation algorithm is least mean square, in which the adaptation coefficients are calculated using the general form,

$$W_{k+1} = W_k + me_k X_k \qquad (5.2)$$

where W_k is the coefficient, e_k the error value and X_k the input data. This is often simplified for VLSI implementations to sign-error LMS by replacing e_k by the sign of e_k and choosing μ to be a power of two. In these circumstances it is possible to use a faster but less precise conversion method [Chen96].

6. DIGIT-SERIAL TECHNIQUES

For real-time implementations hardware architectures need to be able to process the input data as quickly as they are received. Bit-serial systems only process one bit per clock cycle but are area-efficient and suited to low-speed applications. At the opposite end of the performance spectrum bit-parallel systems process one word of input data in one clock cycle. Between these two extremes lies the digit-serial approach [Parhi91]. This processes several bits of the input word, or digit, per clock cycle, e.g. for a input word size of 16 bits a digit-serial system with digit size of 4 will process the input word 4 bits at a time and take 4 cycles to process the whole input word. Obviously for a digit size = 1 the digit-serial approach reduces to a bit-serial system and for digit size = word size the digit-serial system is equivalent to a bit-parallel architecture.

Digit-serial architectures have found favour in situations where some of the performance of the full bit-parallel approach is required but while retaining the simplicity of the bit-serial approach. This is particularly true in VLSI implementations.

Previously digit-serial architectures were developed on an ad-hoc basis but a systematic method of creating a digit-serial architecture from the corresponding bit-serial one has been presented [Parhi91]. A systematic unfolding algorithm is applied to the data flow graph of the bit-serial architecture and this automatically creates the digit-serial version complete with control circuitry. In the above paper the examples assume the use of LSB first fractional two's complement numbers but the technique is quite general and could be applied to other number systems e.g. signed digit redundant system. It can also be applied when the digit size J is not an exact divisor of the wordlength W. However in the envisaged application this would not be appropriate.

The speed of a digit-serial system lies between that of bit-serial and bit-parallel versions. For a bit-serial adder the sample rate is given by; -

$$f_{bs} = \frac{1}{W[t_d + t_s]} \qquad (6.1)$$

where t_s is the sum of the clock rise and fall times and the delay through the latch, and t_s is the delay in computation of the sum output of the adder. For a digit-serial circuit with digit size J the sample rate is; -

$$f_{ds} = \frac{1}{\frac{W}{J}(t_d + JT_s)} \qquad (6.2)$$

so for small J the speed increases linearly with J but for larger J the increase is not so rapid. A similar trend will be true of multipliers.

Recently the unfolding algorithm has been developed so the architecture of the resulting digit-serial design is slightly changed [Chang98]. This permits the digit-serial design to be pipelined at at bit-level. This produces processing speeds comparable to a bit-parallel system with less area. Alternatively the sampling speed can be fixed and the power supply voltage dropped to allow lower power consumption. Although the simplest designs are for unsigned multiplication it can be extended for other number systems. An example given in [Chang98] is for a singly redundant multiplier, which has one multiplicand in two’s complement form but the other one and the output in redundant form belonging to the set {-1,0,+1}. The advantage of using redundant arithmetic is that it avoids carry propagation in parallel arithmetic cells allowing serial operations to proceed in the MSB mode.

The technique can also be used for two’s complement multiplication with the addition of some extra control circuitry. The example given in [Chang98] is a serial-parallel multiplier with the input data entering digit-serially and the coefficient being bit-parallel. For correct operation a guard digit must be inserted between the each input word, so a new word can only be input every $\frac{W}{J}+1$ clock cycles. Because of the bit-level pipelining the critical paths in different types of multiplier implemented by this algorithm are all $T_{FA}+T_L$ where T_{FA} and T_L are respectively the propagation delays of a full adder and a latch. The latency in the design will depend in the type of the bit-serial multiplier used as the basis for the design but this can vary between W and W/J clock cycles.

It is also possible to improve these digit-serial designs by recoding of the binary numbers, most notably by using the Modified Booth Algorithm. This reduces the number of partial products and hence the number of full adders required.

7. CONCLUSIONS

A number of techniques for reducing complexity in the beamformer have been outlined in this report. All these techniques avoid the use of floating point numbers since these complicate the calculation process. Several techniques were considered and rejected as not being suitable notably MSB First and MSB Truncation. However it should be noted MSB First may be suitable for filters in other sections of the project.

Of the two broad methods of implementation, DSP and hardware based, the former is the most desirable choice. The DSP approach is obviously more flexible and with the latest products using instruction pipelining rapid execution speeds are possible.

If this does not provide the required throughput the digit-serial solution should be the easiest hardware solution to implement since the required architecture can be built up systematically. The other approaches, non-uniform number spaces and 3CM multipliers, could give good results, however more detailed work would be required since the improvements are algorithm/architecture dependent.

8. REFERNCES

[Anton98] C. Anton, X. Mestre, J.R. Fonollosa, "Algorithms for Flexible Multi-Standard Array Processing (Part 2)", AC347/UPC/A71/DS/P/005/b1, ACTS SUNBEAM project

[Chang98] Y-N Chang, J. H. Satyanarayana, K. K. Parhi, "Systematic design of high-speed and low-power digit-serial multipliers", IEEE Trans. Circuits Syst., vol 45, pp 1585-1596

[Dahnoun00] N. Dahnoun, "DSP Implementation using the TMS320C62X processors", Pearson Education, 2000

[Harris-93] D. K. Harris-Dowsett, S. Summerfield, "Low Latency Architectures for Wave

[Mestra99] X. Mestra, C. Anton, J.R. Fonollosa, "Algorithms for Flexible Multi-Standard Array Processing (Part 3)", AC347/UPC/A72/PI/I/007/a1, ACTS SUNBEAM project

[Moshnya99] V. G. Moshnyaya, "A MSB Truncation scheme for low-power video processors", IEEE ISCAS 99

[Pahri91] K. K. Parhi, "A systematic approach for design and digit-serial signal processing architectures", IEEE Trans. Circuits Syst., vol 38, pp358-375, Apr. 1991

[Perry97] R. Perry, "Low complexity adaptive equalisation for wireless applications PhD. Thesis, Sept. 1997

[Perry99] R. Perry, A. R. Nix, D. R. Bull, "Adaptive Equalisation Methods for Next Generation Radio Systems", Insights into Mobile Multimedia Communication, Academic Press, 1999

[Smith88] S.G. Smith, P.B. Denyer, "Serial Data Computation", Boston MA, Kluwer Academic, 1988

[Valls99] J. Valls, T. Sansaloni, M. M. Peiro, E. Boemo, "Fast FPGA-based pipelined digit-serial/parallel multipliers", ISCAS 99