THE FFT BUTTERFLY OPERATION IN 4 PROCESSOR CYCLES ON A 24 BIT FIXED-POINT DSP WITH A PIPELINED MULTIPLIER

Martin Grajcar, Bernhard Sick

University of Passau, Faculty for Mathematics and Computer Science (Prof. Dr. W. Grass)
Innstr. 33, D – 94032 Passau, Germany
email: (grajcar|sick)@fmi.uni-passau.de

ABSTRACT

Most of the existing Digital Signal Processors (DSPs) are optimized for a fast and efficient computation of the Fast Fourier Transform (FFT). However, there are only two floating-point DSPs available, which perform the butterfly operation of a FFT in 4 processor cycles, but no fixed-point DSP is designed that way. The new 24 bit fixed-point DSP DAISY, which is able to execute the butterfly in 4 cycles even using a two-stage pipelined multiplier, is described in this paper. With this pipelined multiplication it is possible to reduce the processor cycle time significantly.

1. INTRODUCTION

The FFT is an important tool used in many signal processing applications, e.g. for the analysis of the frequency or the cepstral domain of digital signals. Therefore the developers of DSPs include features in their designs that support an efficient real-time computation of this algorithm. An often used FFT algorithm, the radix-2 decimation-in-time (DIT) in-place FFT, will be considered in this paper (see e.g. [1]).

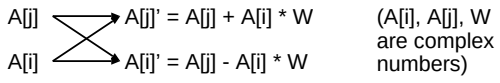

Figure 1: Radix-2 butterfly operation

A 2^x-point FFT mainly consists of $x \cdot 2^{x-1}$ executions of the butterfly operation (see fig. 1). This butterfly can be written by means of real numbers in the following way:

$$A[i]'_{re} = A[j]_{re} - (A[i]_{re} \cdot W_{re} - A[i]_{im} \cdot W_{im})$$
$$A[j]'_{re} = A[j]_{re} + (A[i]_{re} \cdot W_{re} - A[i]_{im} \cdot W_{im})$$
$$A[i]'_{im} = A[j]_{im} - (A[i]_{re} \cdot W_{im} + A[i]_{im} \cdot W_{re})$$
$$A[j]'_{im} = A[j]_{im} + (A[i]_{re} \cdot W_{im} + A[i]_{im} \cdot W_{re})$$

The indices re and im refer to the real and the imaginary part of a complex number. W_{re} and W_{im} (the twiddle factors) are cosine and sine values. The computation can be done 'in place', i.e. the values of $A[i]$ and $A[j]$ are overwritten after the completion of one butterfly (indicated with $'$). Fig. 2 shows the flow graph of an 8-point FFT.

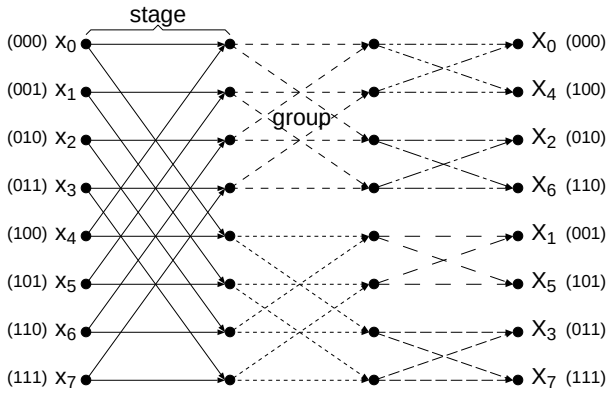

Figure 2: Flow graph of an 8-point FFT

In each stage, butterflies with identical twiddle factors are grouped together (recognizable by the same type of connection). A 2^x-point FFT consists of x stages. The number of groups is doubled in each stage, while the number of butterflies per group is halved. So, the number of butterflies per stage is a constant, 2^{x-1}. The output array of the FFT is in a 'bit-reversed' order.

2. HARDWARE REQUIREMENTS

A direct implementation of the butterfly requires 4 multiplications, 6 additions (or subtractions), 12 memory accesses on array elements ($A[i]$ and $A[j]$ are read/written twice and the twiddle factors could be held in registers) and (preparing the next butterfly) two modifications of array indices. Therefore, with the restriction to use only one hardware multiplier, a butterfly could be executed in 4 processor cycles. But as a consequence of data dependencies, several butterflies have to be computed overlapping each other (thus, 4 cycles is an average). Tab. 1 shows some benchmarks for a FFT ('cycle' means the execution time for a simple assembler instruction, 'minimal code length

of a butterfly' refers to nontrivial butterflies in which multiplications cannot be eliminated (e.g. in the first stage no multiplication is required due to $W_{re} = 1$ and $W_{im} = 0$). The DSP56000 family is the only commercial 24 bit fixed-point DSP family; solely the floating-point families DSP96000 and ADSP-21000 are able to execute the butterfly in 4 cycles.

optimization, [algorithm reference]	**execution time** (in 1000 cycles)	**minimal code length of a butterfly / FFT** (in words)
Texas Instruments TMS320C50 (fixed-point 16 bit)		
n.a., [2]	82.8	18 / n.a.
Motorola DSP56001 (fixed-point 24 bit)		
code length, [3]	63.8	6 / 40
time, [3]	33.6	6 / 105
DAISY (fixed-point 24 bit)		
code length, [4]	**29.8**	**4 / 37**
Texas Instruments TMS320C30 (floating-point 32 bit)		
code length, [5]	64.6	9 / 60
time, [5]	39.4	7.275 / 229
Motorola DSP96002 (floating-point 32 bit)		
code length, [3]	31.2	4 / 30
time, [3]	21.0	4 / 137
Analog Devices ADSP-21020 (floating-point 32 bit)		
time, [6]	21.3	4 / 158

Table 1: Benchmarks of a 1024-point FFT

Since a complex multiplication can be calculated with 3 real multiplications (by $(a + bi) \cdot (c + di) = (A - B) + (C - A - B)i$ with $A = a \cdot c$, $B = b \cdot d$ and $C = (a+b) \cdot (c+d)$), 3 cycles is the theoretical optimum for a butterfly provided that only one hardware multiplier is used. But this alternative solution requires 3 additions executed parallel to each multiplication. The corresponding hardware implementation and first of all the coding of the instruction (even in 32 bit words) would be very difficult.

The objective to execute a butterfly in 4 processor cycles leads to the following hardware requirements for the given butterfly computation scheme:

- a **hardware multiplier** which offers the possibility for MAC-operations (with multiply-and-accumulate, e.g. $A \cdot B + C \rightarrow C$, two additions of the butterfly can be integrated in MACs),
- an **ALU** (arithmetic logic unit) which provides automatic scaling and saturation arithmetic (for the 4 remaining additions),
- two **address arithmetic units** implementing **postmodify** in combination with **bit-reversed** and **circular addressing**,
- **data registers** to hold temporary results and **address registers** for indirect addressing,
- **zero-overhead loops** for the execution of a constant number of butterflies in succession without any control overhead during a loop execution (e.g. testing if a counter equals zero),
- on-chip **sine tables** in ROM for the twiddle factors (not absolutely necessary).

These features not only benefit the FFT implementation, but are also common requirements of digital signal processing algorithms [4].

3. THE DSP DAISY

DAISY is a programmable general-purpose DSP, which means that it is optimized for all common DSP algorithms (e.g. filtering, matrix operations) like other DSPs. The processor has been implemented in VHDL and simulated. Tab. 2 and fig. 3 give a short overview of its features and structure. A word length of 24 bit is useful e.g. for embedded systems with 16 or 18 bit ADCs and DACs.

number representation	24 bit fixed-point two's complement / unsigned, extended precision ($x \times 24$ Bit) supported
architecture	modified Harvard architecture; 4 stage processor pipeline; CPU with $24 \times 24 \rightarrow 48$ bit multiplier (can be used for MAC-operations, shifting and 48 bit additions/subtractions) and 24 bit ALU (with possibility for automatic scaling and saturation); two address arithmetic units; 24 bit data and 16 bit address busses; registers: 16 data, 16 address and 10 special
addressing	immediate, direct, indirect (with postmodify, bit-reversed, circular)
memory	address space: 64 kwords; RAM: 2 kwords in 4 dual-ported RAM blocks, program cache: 8×8 words; ROM: 256×4 words sine table
annotations	on-chip periphery and processor interface not yet designed

Table 2: Overview of DAISY's features

Instructions are executed in a processor pipeline with 4 stages (processor cycles):

- **cycle 0:** fetch an instruction, increment the instruction pointer (program counter);
- **cycle 1:** send operand addresses to memory, send contents of address and offset registers to the address arithmetic units;
- **cycle 2:** send operands from memory or registers to the data processing unit, write updated addresses back to the address registers;
- **cycle 3:** send result address to memory, write result(s) in memory or register(s).

Each cycle is divided into two phases. In the first phase, values are moved from registers or memory to functional units. Results are written back in the second phase of the following cycle. This solution requires an internal forwarding mechanism.

The hardware multiplier computes the result of a multiplication or MAC-operation in two stages. After the

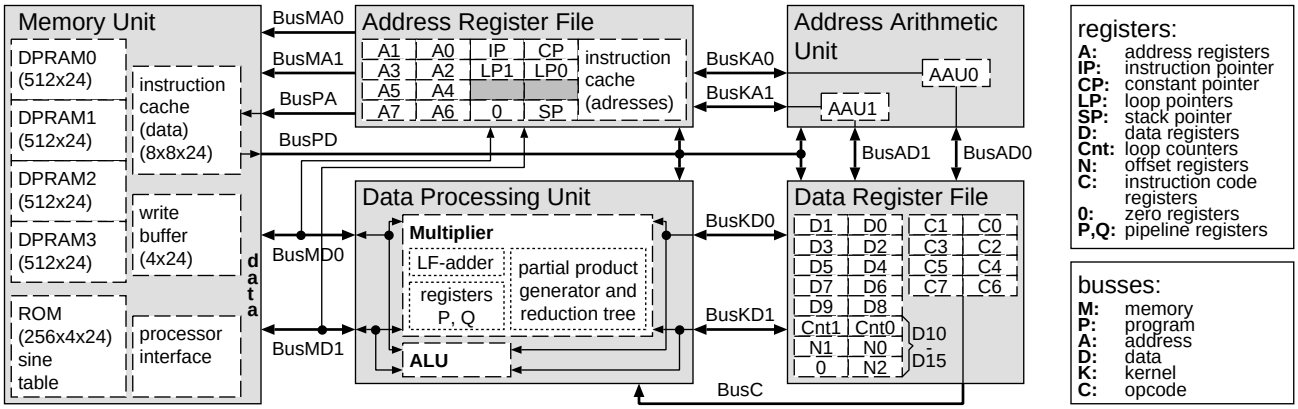

Figure 3: Overview of DAISY's structure

first processor cycle the result of partial product generation and reduction is latched in a two-component redundant binary representation in one of the two registers P or Q. The second stage executes the final addition of the two components. An operation of the form Mul D0 D1 → P → D2 means: store the redundant binary representation of D0 · D1 to P and store the sum of the two components of the previous contents of P to D2. The two stages have approximately the same execution time. The partitioning prevents the 'critical path' of the processor (which determines the optimal processor cycle time) from leading through the multiplier. The hardware implementation of a MAC-operation in fixed-point processors is quite simple: A small extension of the LF-adder causes a minimal prolongation of the second stage.

The coding of a parallel instruction in a 24 bit word is not easy. For example

Add @A2 D0 → @A2#N0 || Mul @A1 D3 → P → D1

(@ is indirect addressing, #N0 is postmodify with offsetregister N0, || is parallel execution) matches the scheme (P is fixed in this instruction)

$\mathsf{Op1}\ \mathsf{M}_a^L\ \mathsf{W}_a^R \rightarrow \mathsf{M}_a^D \parallel \mathsf{Op2}\ \mathsf{W}_m^L\ \mathsf{M}_m^R \rightarrow \mathsf{P} \rightarrow \mathsf{W}_m^D$

(with a: ALU, m: multiplier, L: left operand, R: right operand, D: destination), where Op1 (one bit) can be an addition or subtraction, Op2 (two bits) can be a MAC-operation (with + or −) or a multiplication. Memory operands (M, indirect addressed) are coded in 5 bits, register operands (W) are coded in one (W_a^R, W_m^D) or two (W_m^L) bits. That is, the number of available data registers is restricted in parallel instructions, e.g. $\mathsf{W}_a^R \in \{\mathsf{D0},\mathsf{D1}\}$ [4]. The remaining two bits are needed for the coding of the instruction type (here: 'parallel instruction').

4. FFT IMPLEMENTATION

The FFT implementation for DAISY is similar to the implementation for the 56000 given in [3]. The procedure consists of three nested loops: the outer loop corresponds to a stage (see fig. 2), the middle loop to a group and the inner loop to a butterfly. Tab. 3 shows the assembler code for the middle and the inner loop of the implementation on DAISY (the butterfly is emphasized); the allocation of variables to data, address and offset registers is given in tab. 4 (; precedes comments, [ ] are loop brackets, the register behind the instruction 'LOOP' contains the initial value of the loop counter, & is a parallel data transfer). The execution of a loop

```
... ; outer loop
[ LOOP D7 ; middle loop (D7: number of repetitions)
; load two twiddle factors (from @A7 and @A7+1) in parallel:
Move @A7#N2 → D3&2
; initialize the inner loop:
Mul @A1 D3 → P → D15
Mac @A0 * D2 – P → P
Mul @A0#N0 D3 → P → D0
Sub @A2 D0 → @A4#N0 || Mac @A1#N0 * D2 + P → P
[ LOOP D6 ; begin butterfly (D6: number of repetitions)
   Add @A2 D0 → @A2#N0 || Mul @A1 D3 → P → D1
   Sub @A3 D1 → @A5#N0 || Mac @A0 * D2 – P → P
   Add @A3 D1 → @A3#N0 || Mul @A0#N0 D3 → P → D0
   Sub @A2 D0 → @A4#N0 || Mac @A1#N0 * D2 + P → P
] ; end of inner loop (butterfly)
; execute concluding operations
; (multiplication only for the data transfer P → D1):
Add @A2 D0 → @A2#N0 || Mul @A1 D3 → P → D1
Sub @A3 D1 → @A5#N0
Add @A3 D1 → @A3#N0
; modify A0 – A5 for the next group of butterflies:
Add @A0#N1 @A1#N1 → D15
Add @A2#N1 @A3#N1 → D15
Add @A4#N1 @A5#N1 → D15
] ; end of middle loop
... ; outer loop
```

Table 3: Assembler code for a stage

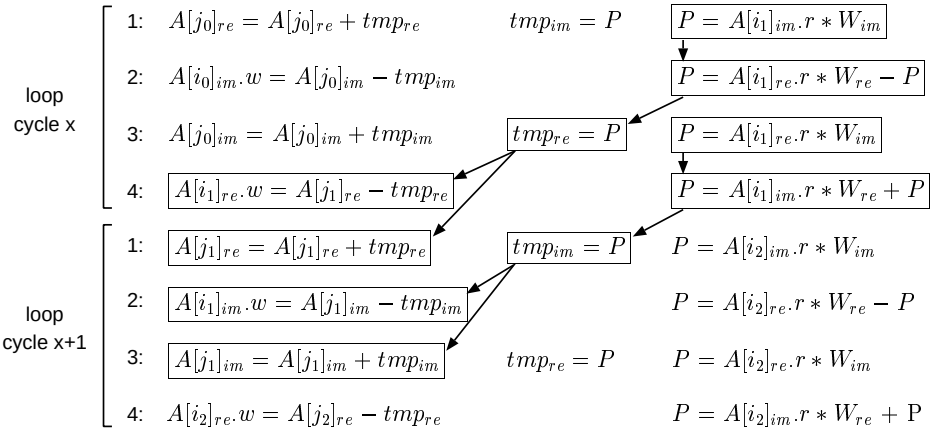

Figure 4: Overlapping butterfly computations

with any number of iterations requires two additional processor cycles outside the loop.

type	No.	contents
data	D0	tmp_{re} (temporary result)
	D1	tmp_{im} (temporary result)
	D2	W_{re} (twiddle factor)
	D3	W_{im} (twiddle factor)
	D6	number of butterflies in the actual group − 1 (repetitions of the inner loop)
	D7	number of groups in the actual stage (repetitions of the middle loop)
	D15	zero register (writing has no effect)
address	A0	pointer to $A[i_x]_{re}.r$ (read)
	A1	pointer to $A[i_x]_{im}.r$ (read)
	A2	pointer to $A[j_x]_{re}$ (read / write)
	A3	pointer to $A[j_x]_{im}$ (read / write)
	A4	pointer to $A[i_x]_{re}.w$ (write)
	A5	pointer to $A[i_x]_{im}.w$ (write)
	A7	pointer to sine table (twiddle factors)
offset	N0	1 (constant for postmodification)
	N1	number of butterflies in the actual group
	N2	number of points of the FFT / 4 (incremental value for the sine table)

Table 4: Register contents

loops:	rest	outer	middle	inner
code length (in words)	8	13	12	4
execution time (in cycles)	7	14	13	4
repetitions for a 2^x-point FFT (in total)	1	x	$2^x - 1$	$\frac{x-2}{2} \cdot 2^x + 1$
cycles for $x = 10$	7	140	13299	16388

Table 5: Implementation analysis

The program executes a butterfly in 4 cycles. Since two subsequent butterflies are overlapping each other, some initializing and concluding operations have to be done and the number of repetitions of the inner loop in each group is reduced by one. This overlapping is described in fig. 4; the actions corresponding to one butterfly are emphasized with boxes. Tab. 5 gives a short analysis of this implementation.

5. CONCLUSION

The whole FFT algorithm, optimized regarding the code length, has a length of 37 words and fits completely into the instruction cache (see fig. 3). A 1024-point FFT can be executed in 29834 processor cycles (see tab. 1 and 5). The time performance of this algorithm is about 11.3% better than the performance of the time-optimized FFT (with code length 105 words) on the comparable DSP56001. A time-optimized version of the FFT on DAISY will achieve additional performance improvements of more than 10% for a 1024-point FFT.

6. REFERENCES

[1] E. O. Brigham; *The Fast Fourier Transform;* Prentice-Hall, Inc., 1974

[2] N.N.; *TMS320 Family Development Support Reference Guide;* Texas Instruments, Inc., 1992

[3] G. R. L. Sohie; *Implementation of Fast Fourier Transforms on Motorola's DSP56000/56001 and DSP96002 Digital Signal Processors;* Motorola, Inc.; 1991

[4] M. Grajcar; *Spezifikation und Implementierung eines Digitalen Signalprozessors für Zahlen in 24-Bit Festkommadarstellung unter Verwendung von VHDL;* diploma thesis; University of Passau, 1996

[5] P. Papamichalis (ed.); *Digital Signal Processing Applications with the TMS320 Family;* Vol. 3; Texas Instruments, Inc., 1990

[6] N.N.; *ADSP-21000 Family Applications Handbook, Volume 1;* Analog Devices, Inc., 1995