A CONVERGED HARDWARE SOLUTION FOR FFT, DCT AND WALSH TRANSFORM

Eric Tell, Olle Seger, Dake Liu

Department of Electrical Engineering, Linköping University, SE-581 83 LINKÖPING, Sweden
{erite, olles, dake}@isy.liu.se

ABSTRACT

We are interested in developing a programmable baseband processor for multiple radio standards, including the wireless LAN standards 802.11a and 802.11b. 802.11a is based on OFDM and uses a 64-point FFT. Demodulation of the complementary code keying (CCK) used in 802.11b includes the computation of a modified Walsh transform.

Similarities have been found between the radix-4 FFT and the fast Walsh transform (FWT) and this has enabled the design of a converged FFT and FWT processor. With small modifications this processor can also be used for calculating the discrete cosine transform (DCT).

A converged FFT/FWT/DCT processor was designed and synthesized in a 0.13 μm process. Results indicate that the hardware can run at 385 MHz, which means a 64-point FFT/DCT is calculated in 140 ns and a FWT for 802.11b 11Mb/s CCK in 47 ns. The area including memory is 0.40 mm^2.

1. INTRODUCTION

With the upcoming 4th generation wireless systems and convergence of multiple radio standards into a single terminal, there is a need for building blocks that can be configured for computing different algorithms used in different standards.

As a starting point for developing a programmable baseband processor , the IEEE wireless LAN standards 802.11 a/b/g have been studied. It was found that computation of FFT, which is used in OFDM standards such as 802.11a and g, and the fast Walsh transform, which is used in the 802.11b standard, can use much the same datapath *if* the radix-4 FFT algorithm is used.

This paper describes converged hardware for computation of 64-point FFT, 64-point discrete cosine transform (DCT) and Walsh transform. 64-Point FFT is used in several OFDM standards, including IEEE 802.11a. The Walsh transform is needed for demodulation of CCK (complementary code keying) which is used in IEEE 802.11b.

The discrete cosine transform is used in several common audio and video compression algorithms. This makes the processor useful for both baseband and application acceleration for example for DAB (Digital Audio Broadcasting), DVB (Digital Video Broadcasting) or a 4th generation wireless multimedia terminal. DCT is often computed using a FFT processor with some pre- and postprocessing. The described FFT/FWT processor has been extended to also allow efficient computation of DCT.

Although this particular implementation only computes 64-point transforms, the concept is easily extended to include eg. 256- or 1024-point transforms. Only memory size and some parts of address generation would be modified.

Section 2 of this paper explains the theory behind the radix-4 FFT and the modified fast Walsh transform. Section 3 presents the proposed datapath and section 4 describes the addressing scheme. This is followed by implementation and synthesis results and conclusions.

2. THEORY

2.1. The radix-4 FFT algorithm

The discrete Fourier transform, DFT, for $0 \leq l < 64$, is defined by

$$X(l) = \sum_{k=0}^{63} x(k) W_{64}^{kl}, \tag{1}$$

where $W_{64} = \exp(-j2\pi/64)$. We now set out to derive a radix-4 FFT of (1). This is done by factoring $64 = 4 \times 4 \times 4$. The resulting algorithm will be similar to a 3-D DFT on a $4 \times 4 \times 4$-cube. We make the following replacements of the indices

$$k = 16k_2 + 4k_1 + k_0 \qquad l = 16l_2 + 4l_1 + l_0, \tag{2}$$

where $0 \leq k_i, l_i < 4$. We will also need $\tilde{l}$,the bitreversed version of l,

$$\tilde{l} = l_2 + 4l_1 + 16l_0. \tag{3}$$

We begin by evaluating

$$\begin{aligned} W_{64}^{k\tilde{l}} &= W_{64}^{(16k_2+4k_1+k_0)(l_2+4l_1+16l_0)} \\ &= \underbrace{W_4^{k_2l_2} W_4^{k_1l_1} W_4^{k_0l_0}}_{\mathrm{DFTkernels}} \cdot \underbrace{W_{64}^{4k_0l_1+k_0l_2+4k_1l_2}}_{\mathrm{twiddlefactors}}. \end{aligned} \tag{4}$$

By inserting (2), (3) and (4) into (1), we get

$$\begin{aligned} X(\tilde{l}) &= \sum_{k_2=0}^{3}\sum_{k_1=0}^{3}\sum_{k_0=0}^{3} x(k) W_{64}^{(k_0+4k_1+16k_2)\cdot(16l_0+4l_1+l_2)} \\ &= \sum_{k_0=0}^{3} W_{64}^{4l_1k_0}\left[\sum_{k_1=0}^{3} W_{64}^{l_2(k_0+4k_1)}\right. \\ &\left.\left[\sum_{k_2=0}^{3} x(k)(-j)^{k_2l_2}\right](-j)^{k_1l_1}\right](-j)^{k_0l_0}, \end{aligned} \quad (5)$$

where we have used that $W_4 = -j$. (5) is a radix-4 FFT that produces a bit-reversed output vector.

2.2. The modified FWT algorithm

The modified Walsh transform, for $0 \le l < 64$, is defined by

$$X(l) = \sum_{k=0}^{7} x(k)(-j)^{p(k,l)}. \quad (6)$$

The kernel function $p(m,n)$ is given by

$$p(m,n) = \sum_{i=0}^{2} m_i n_i, \quad (7)$$

where $m =< m_2m_1m_0 >$ is in base 2 and $n =< n_2n_1n_0 >$ is in base 4.

We now try a deduction similar to the one carried out in section 2.1

$$\begin{aligned} l &= 16l_2 + 4l_1 + l_0 \\ k &= 4k_2 + 2k_1 + k_0, \end{aligned} \quad (8)$$

where $0 \le l_i < 4$ and $0 \le k_i < 2$. With this representation we have

$$p(k,l) = k_2l_2 + k_1l_1 + k_0l_0. \quad (9)$$

Inserting (8) into (6), we get

$$\begin{aligned} X(l) &= \sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\sum_{k_2=0}^{1} x(k)(-j)^{(k_2l_2+k_1l_1+k_0l_0)} \\ &= \sum_{k_0=0}^{1}\left[\sum_{k_1=0}^{1}\left[\sum_{k_2=0}^{1} x(k)(-j)^{k_2l_2}\right](-j)^{k_1l_1}\right](-j)^{k_0l_0}. \end{aligned} \quad (10)$$

By comparing (5) and (10) it is easy to see that they both can compute the modified Walsh transform. Suppose that the input vector in (10) is given by

$$x(k) =< a, b, c, d, e, f, g, h > . \quad (11)$$

To compute modified Walsh transform with the FFT machinery in (5) we make two modifications:

1. Short circuit the multiplication with the twiddle factors W_{64}.

2. The input vector x in (5) has 64 entries. We make the following assignments: $x(0) = a$, $x(1) = b$, $x(4) = c$, $x(5) = d$, $x(16) = e$,$x(17) = f$, $x(20) = g$, $x(21) = h$ and all other entries are zero.

With these modifications (5) and (10) are identical.

2.3. The DCT algorithm

To calculate a DCT using FFT hardware the input samples have to be reordered according to figure 3 c and the the output samples have to be multiplied with compensation factors. The theory behind this method is explained for example in [5].

The input to the DCT is real-valued and only the real part of the output is used. However by using both real and imaginary parts of the input samples, and applying some post processing, two DCTs can be computed simultaneously.

3. DATAPATH

Figure 1 depicts the suggested integrated data path for FFT, DCT and FWT. In FFT/DCT mode the datapath executes one radix-4 FFT butterfly per clock cycle and in FWT mode it executes two Walsh transform butterflies per clock cycle.The mode is controlled by a single, 1-bit control signal controlling all the multiplexers. The multipliers and coefficient ROM are not used in FWT mode. FFT and DCT uses the same datapath, but the coefficients are different in the final step. One of the multipliers is redundant for FFT since the first coefficient of every butterfly is one. Furthermore to enable DCT the multipliers have to be two bits wider (eg. 16x12 instead of 14x12) and the coefficient ROM will hold 64x4=256 coefficients instead of 48x3=144.

In FFT and DCT modes the datapath is pipelined into three stages. In the first stage two additions are executed, in the second stage one (real) multiplication is executed and the third stage has one addition (which is part of the complex multiplication) and one round or truncate operation.

In FWT mode, the critical path is just one addition and the datapath is not pipelined.

Based on 802.11a, 12-bit precision (ie 12 bits each for real and imaginary parts) has been chosen for input and output data. Precision requirement investigations has shown that to reach this precision it is enough to use 16-bit precision for intermediate results and 12-bit precision for coefficients. Each complex multiplier consists of four 12x16 bit real multipliers and two 30-bit adders. The memory word length is 32 bits.

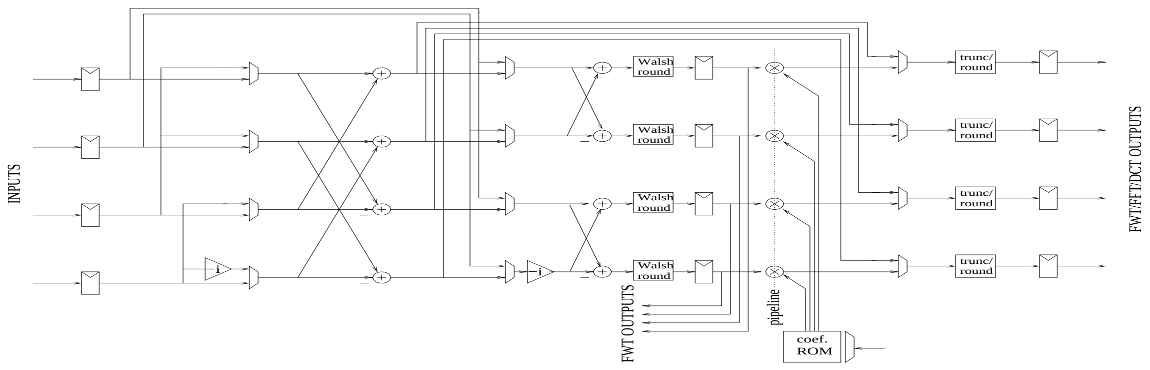


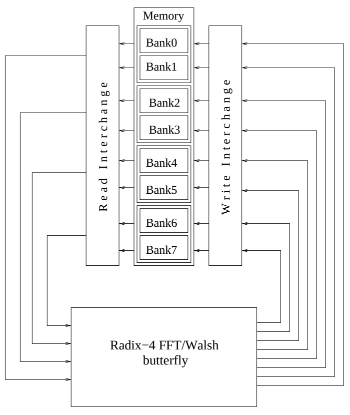
Figure 1: Data path

3.1. Memory architecture

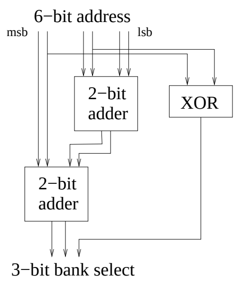
All Algorithms use in-place calculation so it is enough to use one memory.

The memories are divided into four banks of 16 words each, and each bank into two subbanks of eight words each. In each clock cycle the processor may read one word from each bank (totally four words) and write one word to each subbank (totally eight words).

Our implementation uses a register file with eight banks that have separate read and write ports.



(a) The implemented memory architecture



(b) Calculation of bank select signal

Figure 2: Memory architecture

A scheme for making sure that all four parallel reads are always in different banks and that all eight parallel writes are always in different sub banks, has been found by further developing the scheme presented in [4] as shown in figure 2 b. Hence two two-bit adders and one XOR-gate are needed to find the right memory bank, given the address.

Once the correct sub bank has been selected, the three most significant bits of the address is used for addressing within the sub bank.

4. ADDRESSING

All addressing is based on two 6-bit counters, one for generating read addresses and one for write addresses (The write counter is delayed a number of steps corresponding to the pipeline depth; In FFT- or DCT-mode the coefficient ROM is also addressed by a delayed value of the read counter). In the following description it is assumed that the output of the 6-bit read or write counter is $\{x_0, x_1, x_2, x_3, x_4, x_5\}$

4.1. FFT and DCT

When a FFT or DCT is computed the counter runs from 0 ($\{0,0,0,0,0,0\}$) to 47 ($\{1,0,1,1,1,1\}$). x_0 and x_1 indicates the current step. Each of the three steps has 16 iterations (=16 butterflies). The four addresses that are used in parallel are the following:

$$\text{if } \{x_0, x_1\} = \{0, 0\} : \quad \begin{aligned} a0 &= \{0, 0, x_2, x_3, x_4, x_5\} \\ a1 &= \{0, 1, x_2, x_3, x_4, x_5\} \\ a2 &= \{1, 0, x_2, x_3, x_4, x_5\} \\ a3 &= \{1, 1, x_2, x_3, x_4, x_5\} \end{aligned}$$

$$\text{if } \{x_0, x_1\} = \{0, 1\} : \quad \begin{aligned} a0 &= \{x_2, x_3, 0, 0, x_4, x_5\} \\ a1 &= \{x_2, x_3, 0, 1, x_4, x_5\} \\ a2 &= \{x_2, x_3, 1, 0, x_4, x_5\} \\ a3 &= \{x_2, x_3, 1, 1, x_4, x_5\} \end{aligned}$$

$$\text{if } x_0 = 1 : \quad \begin{aligned} a0 &= \{x_2, x_3, x_4, x_5, 0, 0\} \\ a1 &= \{x_2, x_3, x_4, x_5, 0, 1\} \\ a2 &= \{x_2, x_3, x_4, x_5, 1, 0\} \\ a3 &= \{x_2, x_3, x_4, x_5, 1, 1\} \end{aligned}$$

4.2. FWT

When a FWT transform is computed the counter runs from 2 ($\{0,0,0,0,1,0\}$) to 15 ($\{0,0,1,1,1,1\}$). x_2 and x_3 indicates the current step. The first step has 2 iterations (=4 butterflies), the second step has 4 iterations and the last step has 8 iterations. The eight addresses that are written in parallel are a0-a7 below. The four addresses that are read in parallel are a0-a3:

if $\{x_2, x_3\} = \{0, 0\}$:

$a0 = \{0, 0, 0, x_5, 0, 0\}$	$a4 = \{1, 0, 0, x_5, 0, 0\}$
$a1 = \{0, 1, 0, x_5, 0, 0\}$	$a5 = \{1, 1, 0, x_5, 0, 0\}$
$a2 = \{0, 0, 0, x_5, 0, 1\}$	$a6 = \{1, 0, 0, x_5, 0, 1\}$
$a3 = \{0, 1, 0, x_5, 0, 1\}$	$a7 = \{1, 1, 0, x_5, 0, 1\}$

if $\{x_2, x_3\} = \{0, 1\}$:

$a0 = \{x_4, x_5, 0, 0, 0, 0\}$	$a4 = \{x_4, x_5, 1, 0, 0, 0\}$
$a1 = \{x_4, x_5, 0, 1, 0, 0\}$	$a5 = \{x_4, x_5, 1, 1, 0, 0\}$
$a2 = \{x_4, x_5, 0, 0, 0, 1\}$	$a6 = \{x_4, x_5, 1, 0, 0, 1\}$
$a3 = \{x_4, x_5, 0, 1, 0, 1\}$	$a7 = \{x_4, x_5, 1, 1, 0, 1\}$

if $x_2 = 1$:

$a0 = \{x_3, x_4, x_5, 0, 0, 0\}$	$a4 = \{x_3, x_4, x_5, 0, 1, 0\}$
$a1 = \{x_3, x_4, x_5, 0, 0, 1\}$	$a5 = \{x_3, x_4, x_5, 0, 1, 1\}$
$a2 = \{x_3, x_4, x_5, 1, 0, 0\}$	$a6 = \{x_3, x_4, x_5, 1, 1, 0\}$
$a3 = \{x_3, x_4, x_5, 1, 0, 1\}$	$a7 = \{x_3, x_4, x_5, 1, 1, 1\}$

4.3. Input and Output

The fact that in-place calculation is used results in the input (for DCT and FWT) and output (for all transforms) not being in order. The data therefore has to be reordered before and/or after the computation.

For FFT the input is in order and the output is in bitreversed order, that is sample number $\{s_0, s_1, s_2, s_3, s_4, s_5\}$ is found at address $\{s_5, s_4, s_3, s_2, s_1, s_0\}$. For FWT the input and output is reordered as described by figure 3 a and b. For DCT the input is reordered as described by figure 3 c and the output is in bitreversed order.

In our implementation the reordering is built into the ports used for storing input data and reading output data.

5. RESULT

The FFT/FWT/DCT processor described above has been implemented in VHDL and synthesized in a 0.13 μm process using Cadence Physically Knowledgeable Synthesis. The processor consists of 24970 cells and the area is 0.40 mm^2 including memory. The processor can operate at 385 MHz. The FFT and (complex) DCT executes in 54 clock cycles or 140 ns and the FWT in 18 clock cycles or 47 ns. The performance is certainly good enough for the 802.11a and b standards where the symbol times are 4 μs and 8/11 μs respectively.

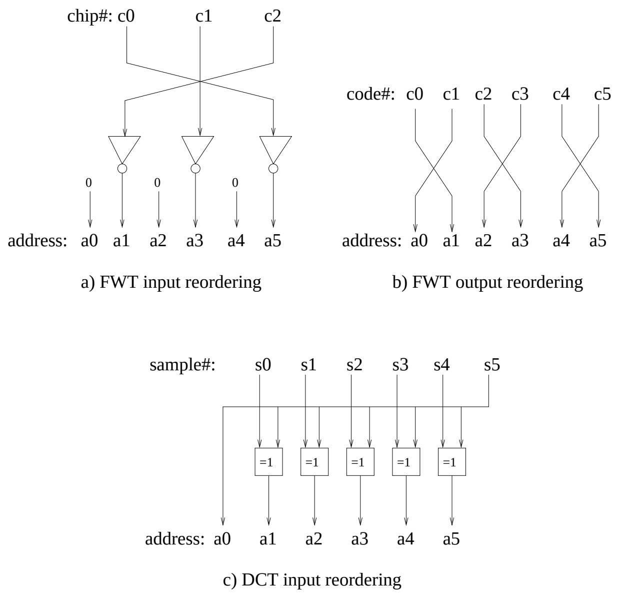


Figure 3: Reordering schemes for FWT and DCT inputs and outputs

6. CONCLUSION

This paper has presented similarities that have been found between the radix-4 FFT algorithm and the modified Walsh transform and has shown how these similarities have been exploited to design a converged processor for 64-point FFT, 64-point DCT and modified Walsh transform.

The proposed architecture is suitable for a baseband processor that needs to handle both OFDM standards like IEEE 802.11a and the IEEE 802.11b standard which uses the modified Walsh transform for CCK demodulation. The performance exceeds by far the requirements of these two standards.

7. REFERENCES

[1] IEEE Std 802.11b. 1999.

[2] IEEE Std 802.11a. 1999.

[3] B Pearson, *Complementary Code Keying made simple*, application note, Intersil Corporation, 2001.

[4] B. S. Son et. al, *A High-Speed FFT Processor for OFDM Systems*, proc. ISCAS 2002, vol 3 pp. 281-284.

[5] R. Storn, *Efficient Input Reordering for the DCT based on a Real-Valued Decimation in Time FFT*, International Computer Science Institute (www.icsi.berkeley.edu) technical report, 1995.