EURASIP Journal on Embedded Systems
a SpringerOpen Journal

RESEARCH Open Access

Implementation of a reconfigurable ASIP for high throughput low power DFT/DCT/FIR engine

Hanan M Hassan*, Karim Mohammed and Ahmed F Shalash

Abstract

In this article we present an ASIP design for a discrete fourier transform (DFT)/discrete cosine transform (DCT)/finite impulse response filters (FIR) engine. The engine is intended for use in an accelerator-chain implementation of wireless communication systems. The engine offers a very high degree of flexibility, accepting and accelerating performance approaches that of any-number DFT and inverse discrete fourier transform, one and two dimension DCT, and even general implementations of FIR equations. Performance approaches that of dedicated implementations of such algorithms. A customized yet flexible redundant memory map allows processor-like access while maintaining the pipeline full in a dedicated architecture-like manner. The engine is supported by a proprietary software tool that automatically sets the rounding pattern for the accelerator rounder to maintain a required signal to quantization noise or output RMS for any given algorithm. Programming of the processor is done through a mid-level language that combines register-specific instructions with DFT/DCT/FIR specific-instructions. Overall the engine allows users to program a very wide range of applications with software-like ease, while delivering performance very close to hardware. This puts the engine in an excellent spot in the current wireless communications environment with its profusion of multi-mode and emerging standards.

Keywords: DFT, DCT, FIR, ASIP, reconfigurable hardware

1 Introduction

The rapid increase in the performance demand of wireless communication systems combined with the proliferation of standards both finalized and unfinalized has increased the need for a paradigm shift in the design of communication system blocks. Recent trends favor Software Defined Radio (SDR) systems due to their scalability and the ability to support multiple standards on the same platform. However, keeping performance within acceptable levels while doing this is a challenging research question.

Different approaches have been taken to address this question. Authors of [1-3] used Digital Signal Processors (DSPs) owing to their high configurability and adaptive capabilities. Although DSP performance is improving, it is still impractical due to its high power consumption and low throughput. On the other hand [4,5] used configurable HW systems due to the high performance afforded by such platforms. However, these designs fail to catch up with the rapid growth in communication standards; they only support a limited class of algorithms for which they are specifically designed. Application specific instruction processors (ASIPs) offer an interesting position between the two approaches, allowing programming-like flexibility for a certain class of applications under speed and power constraints.

Different approaches to ASIPs offer different levels of flexibility. For example: [6-8] proposed an ASIP design which has the reconfigurability to support all/some functions of the physical layer Orthogonal Frequency Division (OFDM) receiver chain including OFDM Modulation/Demodulation, channel estimation, turbo decoder, etc. This reconfigurability between non-similar functions has a severe effect on performance, lowering throughput, raising power, or both. Realizing that these blocks operate simultaneously in a pipeline in an OFDM receiver, a different approach to partitioning the problem can be taken.

The work presented provides a limited class of MICRO-CODED programmable solutions to support a large class of OFDM wireless applications. The receiver chain is divided to four main ASIP processors seen in Figure 1.

* Correspondence: sep_cameo@yahoo.com
Center for Wireless Studies, Faculty of Engineering, Cairo University, Giza, Egypt

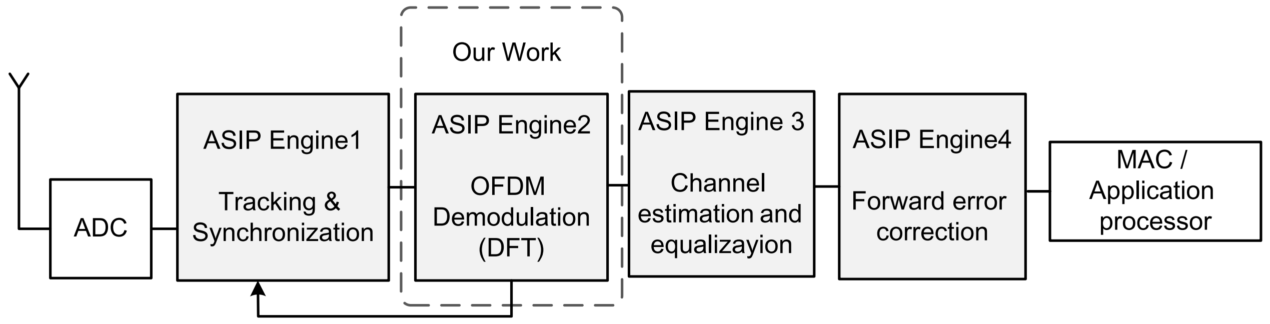


Figure 1 Physical layer OFDM receiver model.

Each block has enough flexibility to support an extensive set of applications and configurations within its class while at the same time preserving hardwired-like performance.

This chapter proposes the OFDM Modulation/Demodulation block which is basically based on Discrete Fourier Transform (DFT) and extended to support similar transformations like Discrete Cosine Transform (DCT) and finite impulse response filters (FIR). DFTs, DCTs, and FIRs are used in innumerable communication and signal processing applications. For example: the DFT is commonly used in high data rate Orthogonal Frequency Division Multiplexing (OFDM) systems such as Long Term Evolution (LTE), WiMax, WiLAN, DVB-T, etc; one of the main reasons is to increase robustness against frequency selective fading and narrow-band interference. One and two dimensional DCT are often used in audio and image processing systems such as interactive multimedia, digital TV-NTSC, low bit rate video conferencing, etc; owing to its compaction of energy into the lower frequencies. Finally FIR, is commonly used in digital signal processing applications that have a frequency spectrum with a wide range of frequency to filter frequency components by isolation, rejection or attenuation depending on system implementation.

1.1 Paper overview

We build on previous studies in [9,10] where we presented a memory based architecture controlled by an instruction set processor. In this study we combine all elements of the design: performing further optimization on the processing elements (PE) to increase their flexibility and performance; as well as presenting a complete implementation including the full memory map and the programming front-end.

The supported mathematical algorithms are discussed in Section 2, This is followed by the system architecture and embedded processor in Section 3. The hardware (HW) accelerators in Section 4, and engine programing with coding example in Section 5. Section 6 details ASIC results and comparison among previously published designs. Section 7 concludes the article.

2 Supported algorithms

The engine can support multiple algorithms some of these algorithms are listed below.

2.1 DFT

N-point Discrete Fourier Transform is defined as:

$$\mathrm{DFT}(x_n) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{1}$$

where: $\begin{cases} k = 0, \ldots N-1 \\ W_N = e^{-2\pi i/N} \end{cases}$

The direct implementation of Equation (1) is $O(N^2)$ which makes it difficult to meet typical throughput requirements. Common DFT symbol length in different communication and signal processing standard is in form 2^x except LTE down link which supports length $1536 = 2^9 \times 3$. Thus optimizing the throughput of a $2^x \times 3^y$-point DFT is our main concern.

Cooley-Tukey [11] proposed radix-r algorithms, which reduce the N-point DFT computational complexity to $O(N \log_r N)$. The main principle of these algorithms is decomposing the computation of the discrete fourier transform of a sequence of length N into smaller discrete fourier transforms see Figure 2.

For lower computation cycle counts, Higher radix algorithm should be used. In practice, the radix-2 algorithm throughput requires four times the number of cycles than the radix-4 algorithm and radix-4 algorithm requires four times the number of cycles of the radix-8 algorithm. On the other hand, higher radix implementations have big butterflies thus they consume higher power and need more complex address generators to handle data flow.

From this trade of between the radix-r algorithm throughput and used butterfly size. We defined the

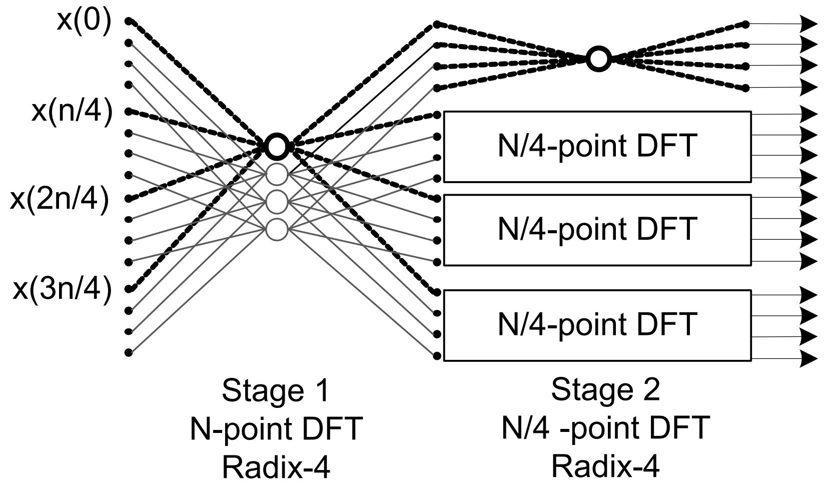


Figure 2 Flow graph of the decimation-in-frequency decomposition of an *N*-point DFT computation into four (N/4)-point DFT computations (*N* = 16).

parameter power efficiency which introduces how much power is taken to have certain throughput. Table 1 shows a comparison between the three radix butterflies. For fair comparison we toke the following assumptions:

- Fix the address generators complexity, by assuming the data are read from memory 4 samples by 4 samples.
- Normalize butterfly power by number of complex multipliers on it, which is the the dominant power consumer in the butterfly.

$$\text{Power efficiency} = \frac{\text{Power}}{\text{Throughput}} \approx \frac{\text{No of multipliers}}{1/\text{No of cycles to end}} \quad (2)$$

From Table 1 The Radix-4 algorithm have a lowest power consumption in addition to its regularity, it more interested specially in memory based architectures. Radix-4 algorithm supports only 4^z-point DFTs, So radix-2 and radix-3 algorithms are required to support all symbol lengths in the form of $2^x \times 3^y$. Radix-4, 2 and 3 butterflies are shown in Figures 3 and 4.

2.2 Inverse DFT

Swapping the real and imaginary parts of input and output data of DFT, we can get the *N*-point Inverse Discrete Fourier Transform (IDFT) (Equation 3) of a sequence $X(K)$ scaled by N (Equation 4).

$$\text{IDFT}(X_k) = \frac{1}{N}\sum_{n=0}^{N-1} X(k)W_N^{-kn}, \quad k = 0, \ldots, N-1 \quad (3)$$

$$\begin{aligned} \text{IDFT}(X_k) * N &= \sum_{n=0}^{N-1} X(k)W_N^{-kn} = \left(\sum_{k=0}^{N-1} X^{*T}(k)W_N^{kn}\right)^{*T} \\ \text{IDFT}(X_k)^{*T} \underbrace{N}_{\text{scale factor}} &= \underbrace{\sum_{k=0}^{N-1} X^{*T}(k)W_N^{kn}}_{\text{DFT of } x*T} \end{aligned} \quad (4)$$

Table 1 Energy consumed for *N*-point FFT vs.

Algorithm	Radix-2	Radix-4	Radix-8
Number of butterflies	2	1	1
Number of stages	$\log_2(N)$	$\log_4(N)$	$\log_8(N)$
Number of butterflies operations/stage	$\frac{N}{2}$	$\frac{N}{4}$	$\frac{N}{8}$
Number of clock cycles/butterflies	1	1	2
Total number of clock cycles for *N* -point FFT	$\frac{N}{4}\log_2(N)\left(\frac{N}{4}\right)(x)$	$\frac{N}{4}\log_4(N)\left(\frac{N}{4}\right)\left(\frac{x}{2}\right)$	$\frac{N}{4}\log_8(N) \times 2\left(\frac{N}{4}\right)\left(\frac{x}{3}\right)$
Normalized power	2	3	7
Power efficiency	$0.5 \times N \times x$	$0.375 \times N \times x$	$0.43 \times N \times x$
As $N = 2^x$			

Radix-*r* algorithms

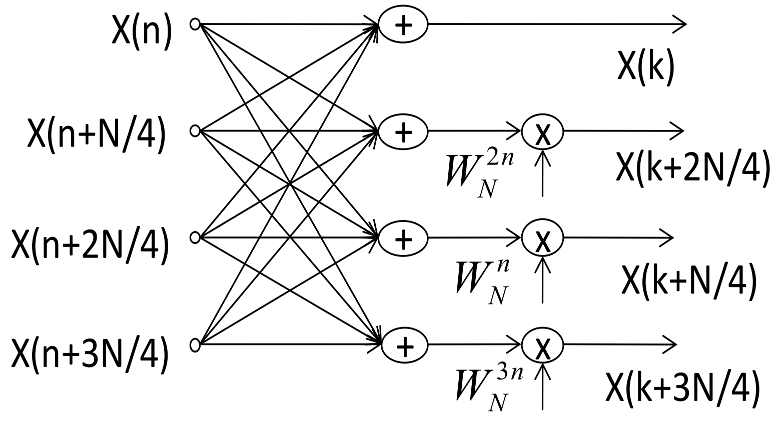


Figure 3 Radix-4 butterfly.

2.3 DCT

Several types of the DCT of a sequence $x(n)$ are defined in [12]. The most popular being type II which is defined as:

$$\mathrm{DCT}(x_n) = \omega(k)\sum_{n=0}^{N-1} x(n)\cos\left(\frac{(2n+1)\pi k}{2N}\right), \quad \omega(k) = \begin{cases}\sqrt{1/N} & k \neq 0\\ \sqrt{2/N} & k = 0\end{cases} \tag{5}$$

Braganza and Leeser [13] proposed an implemention to get a real DCT from the DFT by constructing a sequence $v(n)$ from real input data $x(n)$ as follows:

$$v(n) = \begin{cases} x(n) & n = 0 \ \dots \ N-1 \\ x(2N-n-1) & n = N \ \dots 2N-1 \end{cases} \tag{6}$$

Then the output of $\mathrm{DFT}(v_n)$ is multiplied by $2\omega(k)e^{\frac{-i2\pi k}{2N}}$.

2.4 Inverse DCT

The inverse DCT of type II is type III which is defined as:

$$\mathrm{IDCT}(x_k) = \sum_{k=0}^{N-1} \omega(k) X_k \cos\left(\frac{(2n+1)\pi k}{2N}\right), \quad \omega(k) = \sqrt{2/N} \tag{7}$$

For the IDCT, we reverse the above steps. First, X (k) is rearranged to form a complex hermitian symmetric sequence $V(k)$:

$$V(k) = \frac{1}{2}e^{\frac{j\pi k}{2N}}[x(k) - jx(N-k)], \quad k = 0, 1, 2 \dots N-1 \tag{8}$$

Then construct $v(n)$ by getting the IDFT of V (k), finally rearrange $v(n)$ to get $x(n)$.

2-Dimension modes

For 2D modes, the 1D mode is performed two times: one time in all rows of input frame then another time on the columns of the result Figure 5.

2.5 FIR

The FIR filter Equation (9) is handled using multiply accumulate (MAC) operations and accelerated by using Multiple computing units.

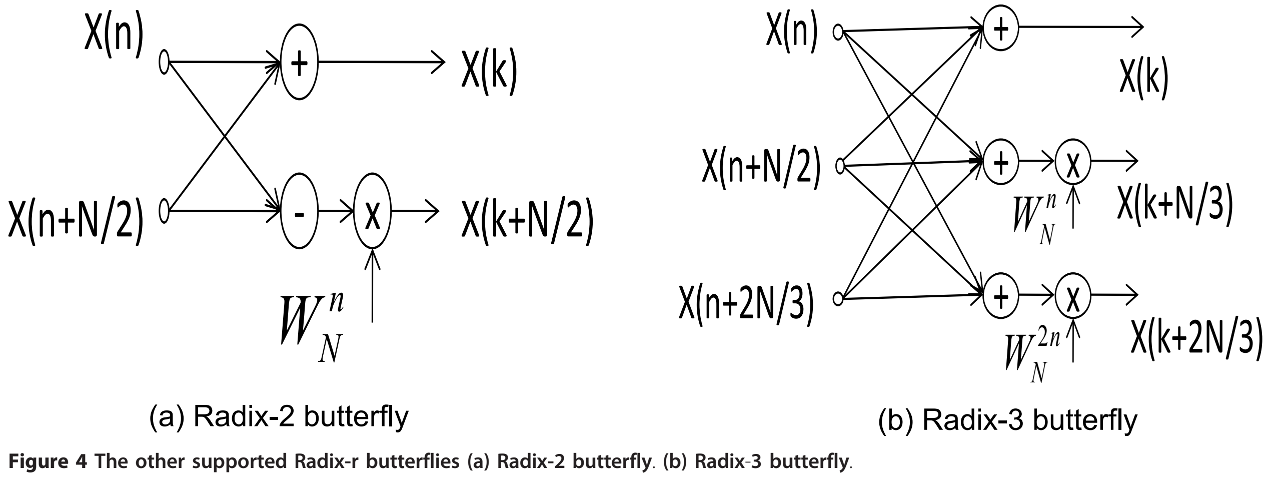


Figure 4 The other supported Radix-r butterflies (a) Radix-2 butterfly. (b) Radix-3 butterfly.

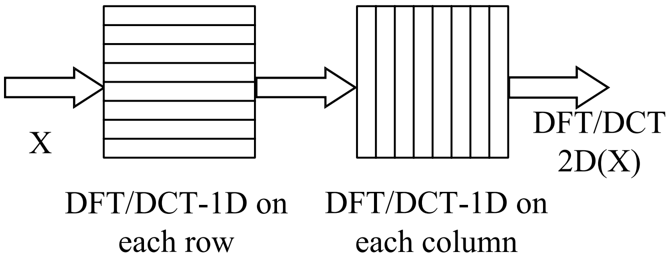


Figure 5 2D DFT/DCT from 1D DFT/DCT.

$$y_t = \sum_{k=0}^{N-1} x(k) a_{t-k} \tag{9}$$

where a's are the filter coefficients.

2.6 Other transformations

Other transformations like any-point DFT can also be handled using basic operations like MAC, accumulator and vector operations.

3 ASIP processor

Embedded architectures are divided to pipelined [14] and memory based architectures (iterative designs) [5,15]. The pipelined architectures are constructed from long chain from butterflies connected to individual memories. For example to support 4K-DFT by pipelined architecture like Radix-4 Singlepath Delay Feedback (R4SDF) [16] (seen in Figure 6). It needs six pipeline radix-4 butterflies (three complex multipliers) connected to six dual port memories. The memories have a read and write operation in each clock cycle. While The memory based architectures usually consist of one butterfly with only two dual port memories. The memories in the based architectures have also a read and write operation in each clock cycle which is approximately similar to the memory transactions in the pipelined architectures. From this discussion we prefer to use memory-based architecture and we prove our selection in Section 6 by comparing our results versus anther publish pipeline architecture.

The first step in the design of a flexible and efficient ASIP is to identify the common set of operations in the class of operations which must be supported. The computationally intensive operations are defined as coefficient-generation, address-generation, and PE. These operations are supported by HW acceleration.

To meet the high throughput demand, data operations are handled through vector instructions. Synchronization in the processing pipeline is handled through handshakes between the system blocks. This greatly reduces the load on decoders, allowing continuous flow in the pipeline and providing dedicated design-like throughput.

The critical path in the PE is relatively short. This simplicity combined with the high throughput of the pipeline allows the user to greatly under clock the circuit, thus allowing significant power scaling with application.

When a valid configuration radix-r stage is received, the HW accelerators are configured to operate on a user selected DFT/IDFT size. The read address generator is responsible for generating data addresses with their memory enables and giving its state to the coefficient generator to maintain synchronization between data and coefficients. The data and coefficients are handed to the PE which is configured to apply radix-*r* calculations. Upon finishing, the PE enables the write address generator and finally the processed data is saved in the 2nd memory Figure 7.

To allow instantaneous reading and writing and to keep the pipeline full, two *N*-word memories are used

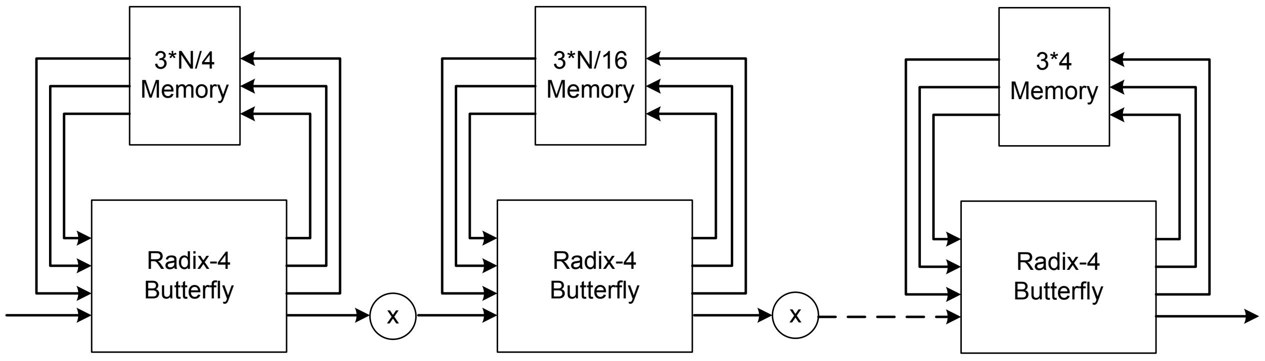


Figure 6 R4SDF pipeline architecture.

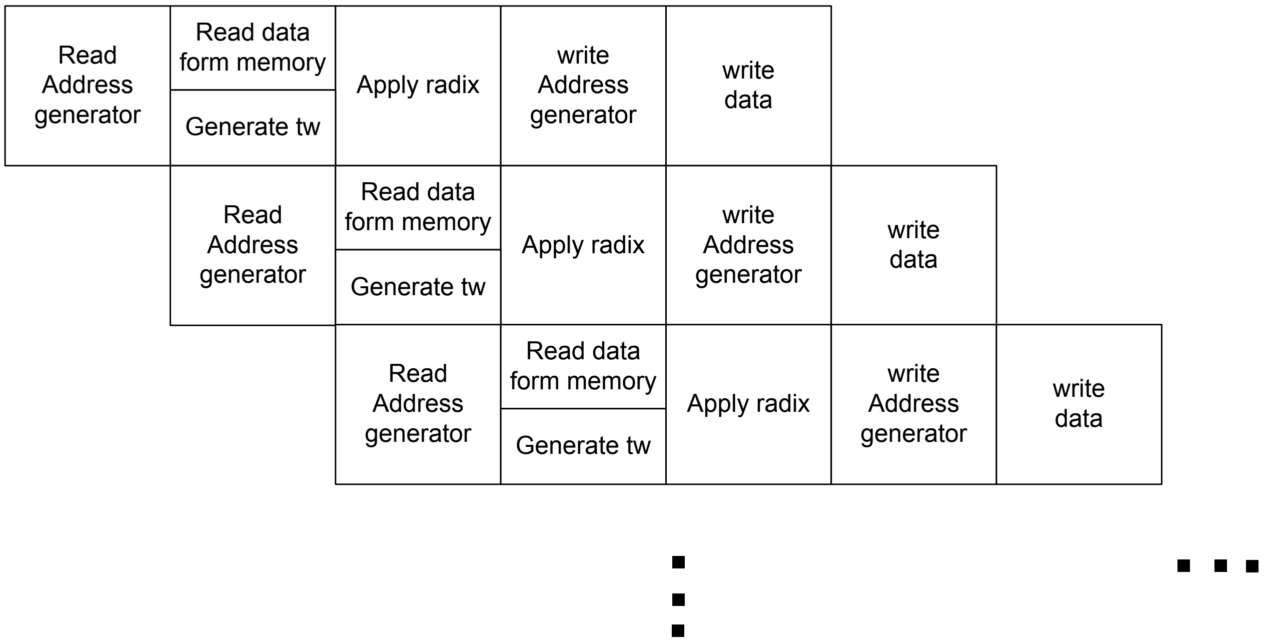


Figure 7 Pipeline process.

one for reading data and another for writing results. The source and destination memories are exchanged each stage. Each memory contains four dual port banks and has four input and output complex data buses to match the configurable memory requirements. The memory bus controller is responsible for applying the input and output data to the corresponding memory banks depending on its bank number and the memory state (read or write). Memory architecture is shown in Figure 8.

In the embedded processor architecture seen in Figure 9, input/output signals handle the interface between the decoder and the external environment. Depending on the external environment state, the decoder enables data transmission, importing, exporting or both. The I/O data bus contains four complex word buses, two for importing data and the other for exporting.

The boot-loading memory consists of a non-volatile bank responsible for initializing the processor RAMs with the required micro-code. The engine is controlled by a non-pipelined decoder with 16 registers in the register file and a 26-bit instruction set with 66 instructions.

(1) The register file is divided into even and odd sets, the real parts of complex words are saved in the even registers and the imaginary parts in following odd registers. Complex words are called by their real register number while a real word may be saved in any register and called by its index.

(2) The instruction set is divided into five classes:

- Radix instructions like: Radix-2/3/4, Inverse Radix-2/3/4 used for DFT.
- MAC instructions for FIR: multiply two data vectors and accumulate, multiply data vector by coefficient and accumulate.
- Vector Multiplications instructions For DCT/IDCT: multiply by coefficient,
- Vector instructions like: accumulate, power, energy, addition, subtraction, multiplication, multiply by coefficient used to perform general vector arithmetic.
- Word instructions like: shift, set, load, store, complex or word addition, subtraction, multiplication used mostly for control.
- Data transmission instructions like: data arrangement, data importing and exporting.
- Control instructions like: compare, conditional/unconditional branches, disable/enable dealing with imaginary part.

All vector instructions are applicable on complex words and have the ability to define the order in which data is read or written. MAC instructions are used for general implementations of FIR equations. MAC allows multiplication of data by data or data by stored or generated coefficients. MAC and Vector multiplication instructions allow multiplication by coefficients or their inverse for general transformations purpose.

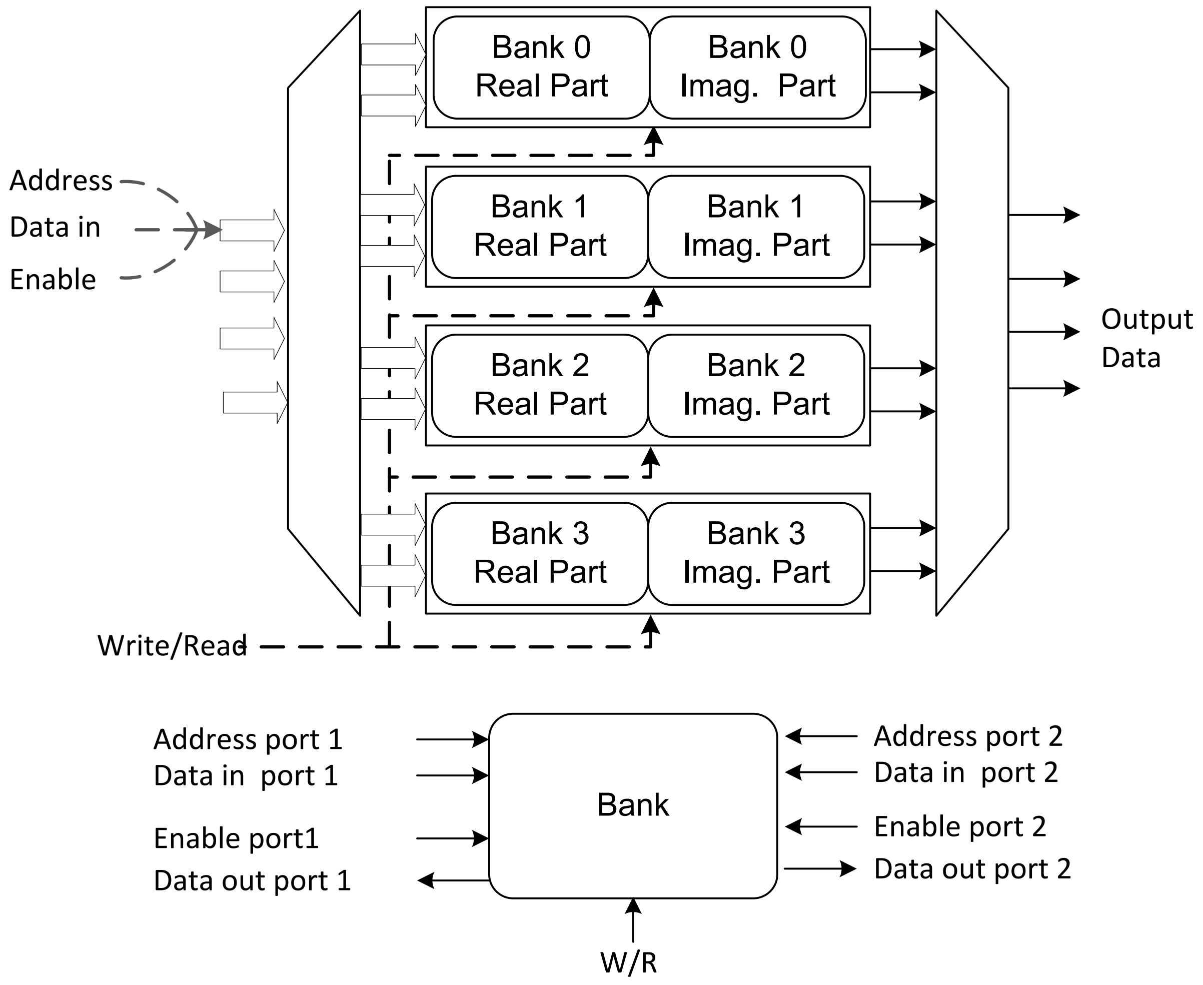


Figure 8 Memory architecture.

4 Hardware accelerators

4.1 Processing element

The PE is the primary computational unit of the engine see Figure 10. The PE can be set to perform two radix-2 butterflies, one radix-3/4 butterfly For DFT implementations, multiply For DCT/IDCT multiplication stage, multiply accumulate for general FIR implementations in addition to other operations like accumulate, addition and subtraction. It is divided into four units: Constant multiplier unit, Addition unit, Multiplication unit, and finally Rounder unit. To increase utilization we time-share the complex multiplier to perform constant multiplication functions, that is to say constant multiplier CM and multiplier 1 $M1$ in Figure 10 use the same multipliers. Data width naturally grows with processing, this is a major question in fixed-point ASIP applications. A rounder unit is placed at the final stage to re-fit data in a constant number of bits (word length). Stage scale factors can be set by the programmer and a proprietary software tool automatically generates the necessary scale factors for a given application. Complex multipliers are configured to multiply input 1 by input 2 or input 2 conjugate. Adder 3 is responsible for accumulate operations, so it is provided by a scalable truncator to prevent overflow. Multiplexers at the input and output data pins are used to swap their real and imaginary parts for inverse operations. The additional multiplexers configure the butterfly and bypass some stages like the multiplication stage.

4.2 Coefficient generator

The coefficient generator generates needed coefficients in two modes.

Mode one: Generates twiddle factors needed for Radix-r and DCT/IDCT Multiplication stage calculations. The first N/4 coefficients are stored in RAM and the remaining coefficients are generated by using the even and odd symmetry properties in the phase and amplitude of twiddle factor (Equations 10 and 11).

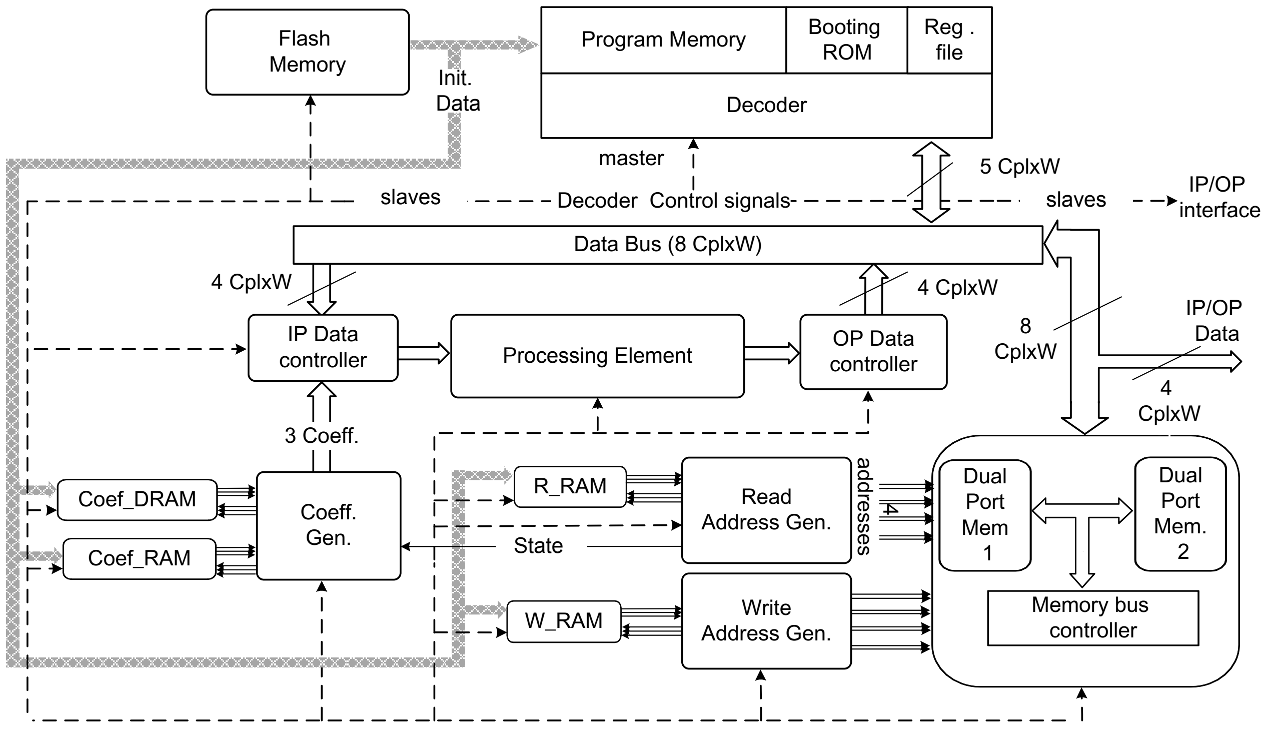


Figure 9 DFT/DCT/FIR processor.

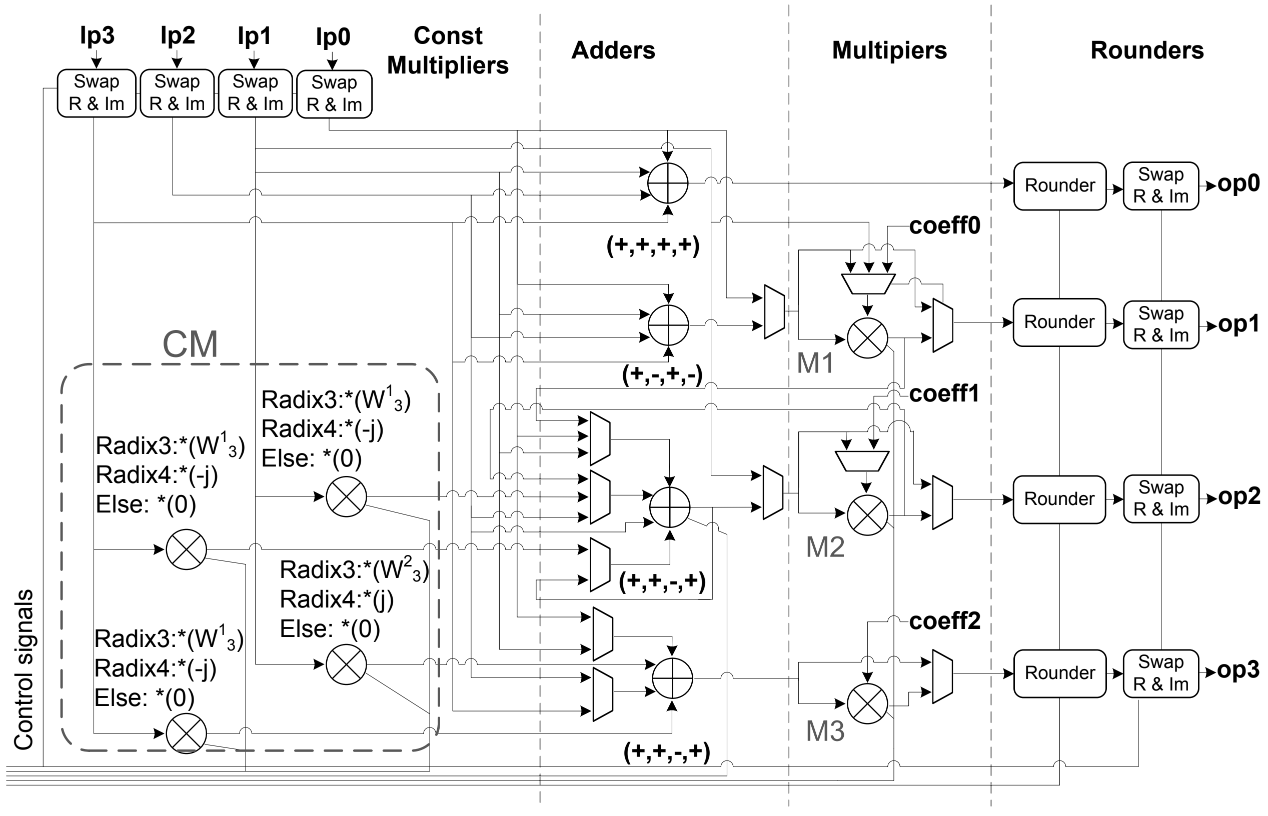


Figure 10 Processing element.

$$e^{j2\pi\frac{n}{N}} = e^{j2\pi\frac{n'}{N}} e^{j2\pi\frac{x\times N/4}{N}} = e^{j2\pi\frac{n'}{N}} \times E(x), \qquad n' = 0,\ldots,N/4 \tag{10}$$

$$E(x) = e^{j2\pi\frac{x}{4}}, \quad x = 0,1,2,3 = 1, j, -1, -j \tag{11}$$

For $e^{-j2\pi\frac{n}{N}}$ we invert the imaginary part's sign. For radix-4 computations we need to generate three twiddle factors at a time, so we use two memories, the first memory is a dual-port RAM and is used to generate $e^{-j2\pi\frac{n}{N}}$ and $e^{-j2\pi\frac{3n}{N}}$. The second memory is a single-port RAM which is used to generate $e^{-j2\pi\frac{2n}{N}}$. For frame lengths with x >2, the 2nd memory addresses are always even. So we remove all odd entries. This reduction adds a negligible noise in $x = 2$ case. For frame lengths with $x = 1$ we replace N by $N'(N' = 4 \times 3^y)$. Consequently we save the first $N'/4 = 3^y$ coefficients in RAM. To save power the 2nd memory is enabled only in radix-4 stage.

This method reduces coefficient memory size to 18% of a direct LUT implementation.

Mode two: Read stored coefficients from the first RAM starting from selected address and going in ascending or descending order depending on selected mode. This is more suitable for FIR transformation and direct implementations of general filters.

4.3 Read and write address generators

Generate continuous write and read addresses depending on their modes. The address bus is divided into four partitions: real part enable, imaginary part enable, bank number and bank index see Figure 11.

Each generator is connected to a single port RAM to get off-line generated addresses. Read and write address memories hold two addresses in each entry. To enable reading four sequential addresses in one clock cycle, write address memory is divided into two single port RAMs, one for odd entries and another for even entries.

The address generation modes are defined as:

Mode one: Generate addresses for different radix-r stages. Radix-4, 2 and 3 need to read 4, 4 (tworadix-2 handled in parallel), 3 data samples respectively for their computations.

This can be handled in several ways: Read data from memory 2 samples by 2 samples with 2 clock latency for each radix operation, double memory clock frequency and read 2 samples by 2 samples with 1 clock latency for each radix operation at the expense of double memory power, or use 4-port memories. Each of the above techniques have drawbacks to different degrees like lower throughput, power or both. In [10] we proposed an address scheme to solve the above problem with conflict-free memory access. The scheme is contingent on partitioning the memory to 4 dual-port memory banks as well as the specific way data is distributed between the banks. This guarantees that at any stage we have at most two accesses to the same memory bank.

Initially data is saved and distributed between memory banks to be ready for the first radix stage (radix-4 or 3). As $N = 2^x \times 3^y$, $(x \neq 1)$, if x is even (integer stages from radix-4) the butterfly performs radix-4 computations till the end then switches to perform radix-3 stages. Else (if x odd) the butterfly performs $\left(\frac{x-1}{2}\right)$ radix-4 stages followed by radix-2 then switches to perform radix-3 stages. Switching to radix-3 stages consumes a one-time additional stage to rearrange data in memory banks. At last radix-r stage, radix output is saved in the same locations of radix inputs.

Samples at any stage are saved in memory depending on the current radix stage (r), current DFT frame length (N), DFT frame number (f), and sample index inside the DFT frame (n) see Figure 12. The bank number results from accessing the bank Look Up Table (LUT) (Table 1) by signal bank$_t$, and the data index in the bank (Equation 12).

$$\mathrm{bank}_t = \mathrm{floor}\left(\frac{\mathrm{n}}{\mathrm{N/r}}\right) \tag{12}$$

$$\text{Bank index} = n \times \mathrm{mod}_{N/r} + f \times \frac{N}{r}$$

Mode two: Generate addresses for DCT/IDCT modes to arrange data in v_n and V_k order.

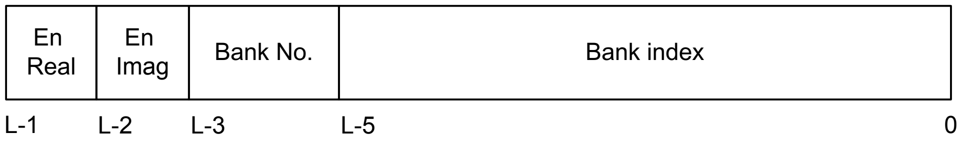


Figure 11 Address structure.

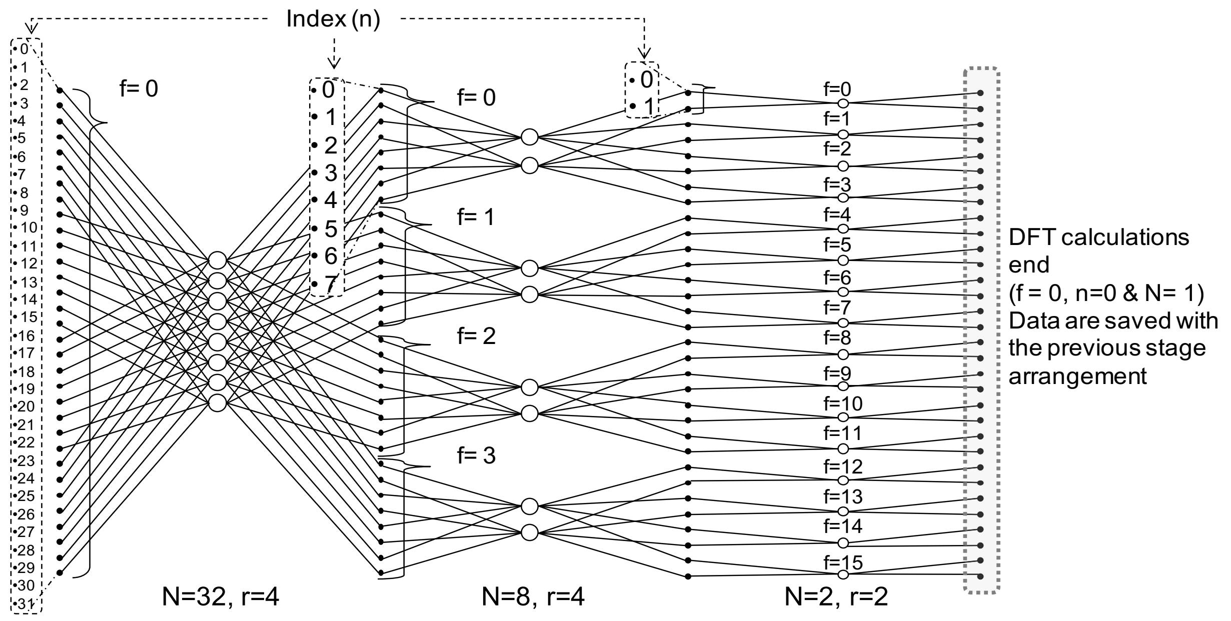


Figure 12 Signal flow graph of 32-point DFT.

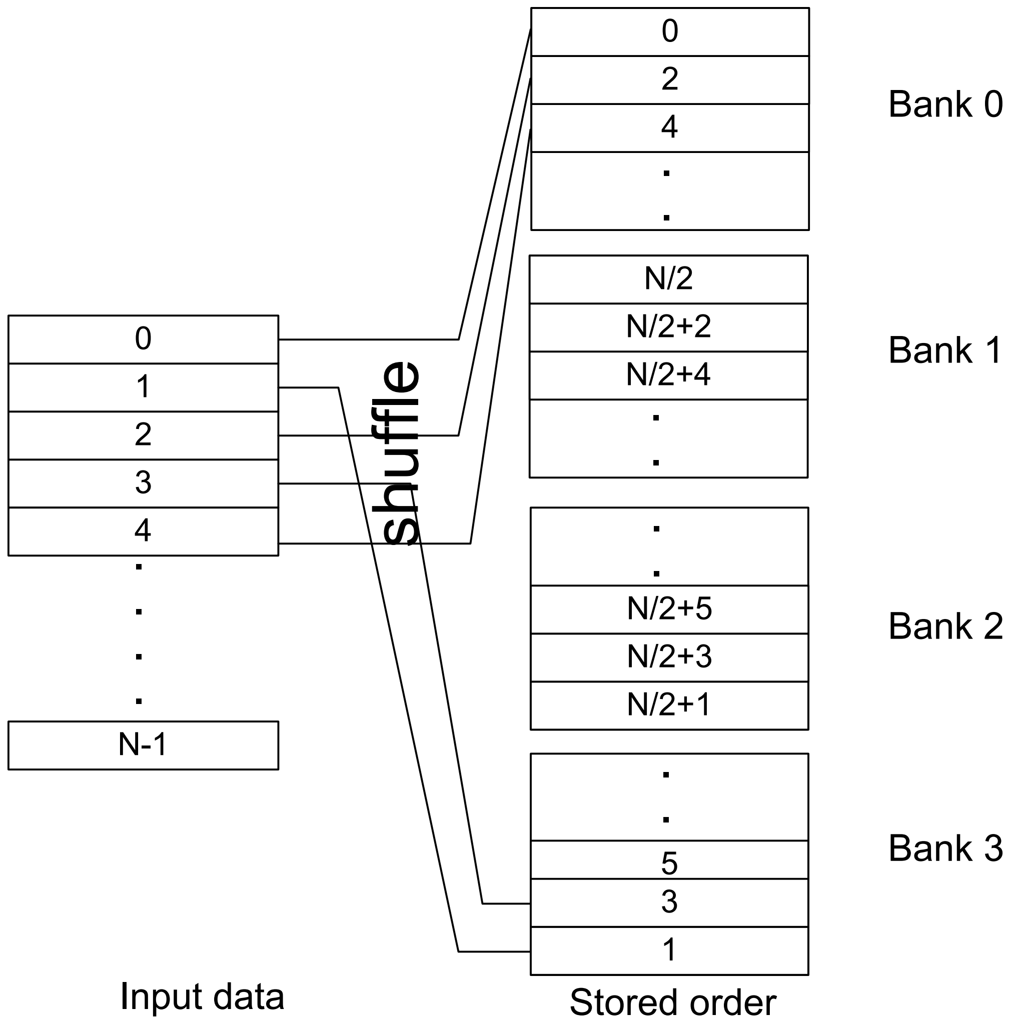


Figure 13 Re-ordering pattern for v_n sequence.

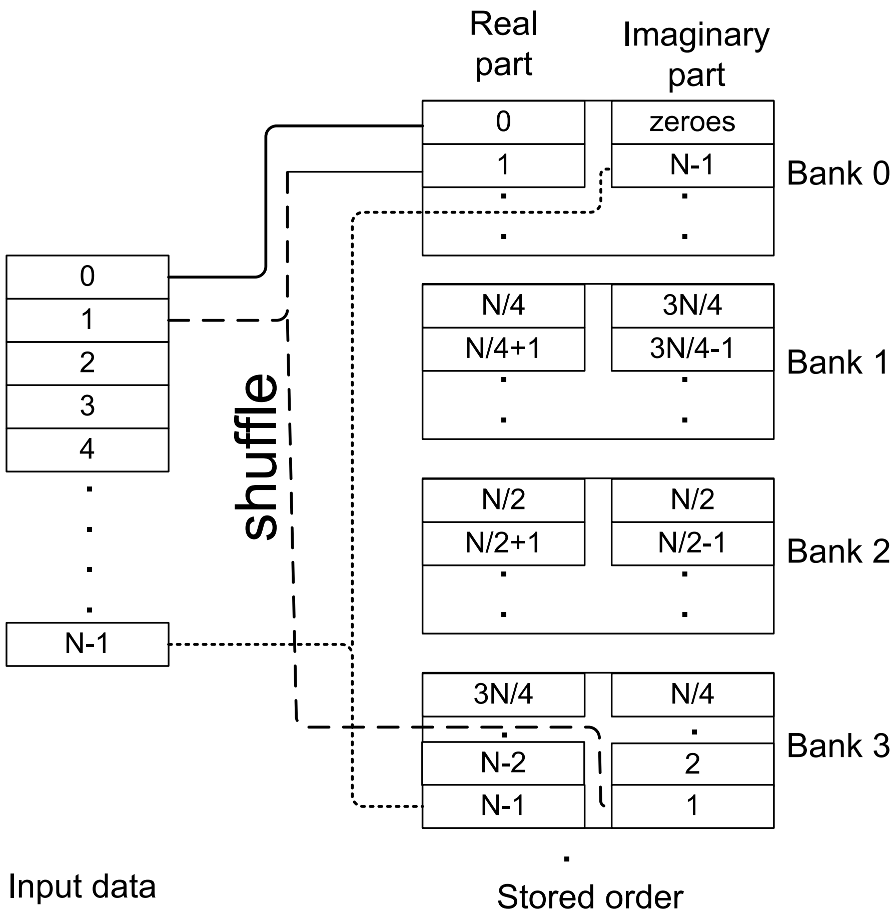


Figure 14 Re-ordering pattern for V_k sequence.

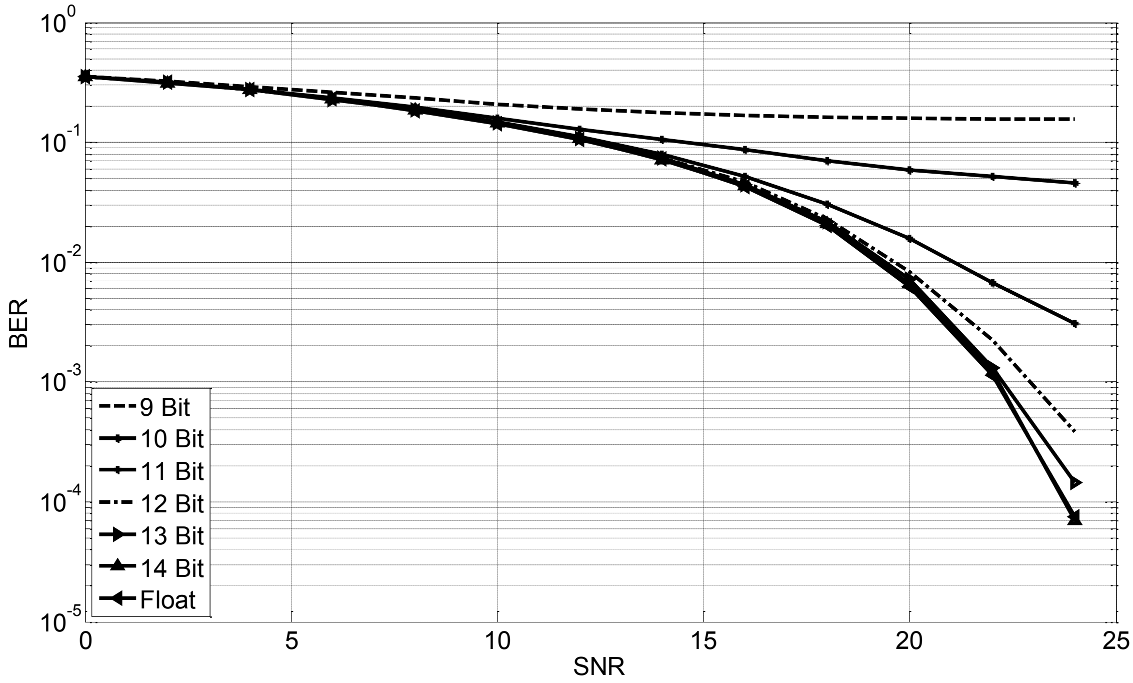


Figure 15 BER curve WIMAX system with 64QAM modulation, fading rate = 1/2, number of sub carriers = 240. Using quantized input to quantized DFT module.

For DCT inputs are saved in the shuffled order shown in Figure 13. Data is distributed in the memory banks to allow direct starting for the next radix stage.

In IDCT computation sequence data is ordered in V_k order then multiplied by coefficients in the next stage. In order to save data arrangement time, data is saved in shuffle order shown in Figure 14 then at multiplication stage the multiplier is configured to multiply the coefficients by data conjugate to construct true V_k sequence.

Each input sample is saved in two locations in memory and the data is imported 2 samples by 2 samples in order to reduce data transmission time. So, data is distributed in memory banks to prevent memory conflict.

Mode three: Generate addresses for other vector instructions like MAC. It generates two addresses for two data vectors or one data vector (two samples at time) in-order sequence or get them form memory. With start addresses and data length as an input parameters.

5 Engine programming

The embedded processor programming passes through three phases: Simulation, testing, and verification. We will discuss 1024-point DFT, 8 × 8 DCT and 64 tap FIR as case studies.

5.1 Simulation

The goal of these simulations is to find best values of our design which are: the scale factor which we divide on each radix-*r* stage, word length and coefficient factors length.

5.1.1 Scale factor

Due to the nature of DFT operation the output data range is growth with the radix stages. So, the data must be scaled after each radix-r stage to refit in fixed number of bits. If this scale is large the data will be lost, on the other hand if it is small many overflows will occur, so stage scale must be well chosen. We considerate this point and designed an optimum scales generator tool to select the best scale on each stage with two modes:

(1) Select the Highest SQNR.

(2) Guaranteed output RMS, to keep signal peaks which are needed in same applications.

The tool are designed by Matlab software, it generate all possible scale factors with corresponding signal to quantization noise (SQNR) and the RMS of the output then select the best scale vector Depending on the input mode. Using the first mode for our example reveals scale factors of (4 4 2 2 2) for 1024 point five stages, giving the highest SQNR with a Gaussian input.

```
% reset all registers
   set  0,r0
   set  0,r1
   set  0,r2
   set  0,r3
   set  0,r4
   set  0,r5
   set  0,r6
   set  0,r7
   set  0,r8
   set  0,r9
   set  0,r10
   set  0,r11
   set  0,r12
   set  0,r13
   set  0,r14
   set  0,r15
```

```
 % Data length = 1024
   set  1024,r0
% load instructions to program memory
%start address of  Instructions  in flash = 0
   bootload   pm,0
% Load instructions to read address RAM
   bootload  r_mem,1024
   bootload  w_mem,2048
   bootload  ws_mem1,3072
   bootload  ws_mem2,4096
   set  2 ,r0
   bootload  s_reg,5120
   set  0,r0
% jumb to program memory
   jumb  pm
```

Figure 16 Boot loading initialization instructions.

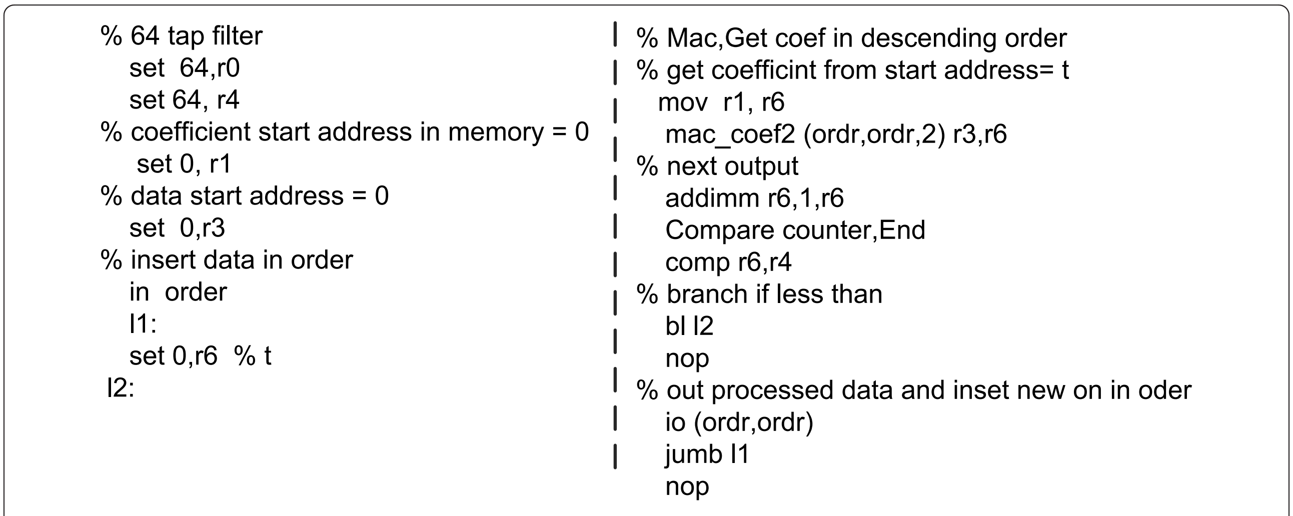
```
% Set parameters
% Input Data length = N/2 = 512
    set  512,r0    % r0 = 512
    set  256,r2
    set  256,r3
 % Import new symbol in radix 4 order
    in  (ordr_rdx4)
 % Swap 2 memories
    swap
    l1:
% Radix parameters
    set  256,r2    % N/4
    set  256,r3    % n/4
    set  64,r4     % n/16
    set  1024,r5   % n
    set  192,r0    % 3n/16
    set  4,r1      % N'/N

    set  0,r6       % Counter = 0
    set  4,r7       % 4 iterations
l2:
% Radix 4, scale = 2^2
   n_radx4 (2)
   swap
   shiftr    r3,2,r3   % r3 = r3 >> 2
   shiftr    r4,2,r4
   shiftr    r5,2,r5
   shiftr    r0,2,r0
   shiftl    r1,2,r1    % r1 = r1 << 2
   addimm  r6,1,r6 % counter += 1
% Compare counter,End
   comp   r6,r7
% Branch if less than
   bl        l2
   nop       % no operation after loop
% Last stage radix-4, scale = 2^2
   l_radx4 (Scape_multi,2)
   swap
   set   256,r3
   set  1024,r4
   set   0,r5
   set   512,r0
   set   0,r1
% I/O Data
% Get addresses from memory for exporting data
% Import new symbol in radix 4 order
   io   (mem0, ordr_rdx4)
   swap
% Process new symbol
   jumb l1
```

Figure 17 Implementation code example for 1024-point DFT.

```
   set  32,r2
% insert data and distributed it in vn order
   in  vn
   l1
   set   0,r6
   set   1,r7
   l2
   swap
   set  648,r1
   set  16,r2
   set  2,r3
   set  1,r4
   set  8,r5
% radix 4 in all rows
   n_radx4 (nlst,2)        Radix 4 not last
   swap
   shiftr   r3,2,r3      n
   shiftr   r4,2,r4      n
   shiftr   r5,2,r5      n
   shiftl   r1,2,r1      n
% radix 2 in all rows
   radx2 (lst,2)        Radix 4 not last
   swap
% disable writing imaginary part
% write real part only
  dis_img write
   set  162,r1
   set  16,r2
   set  2,r3
   set  64,r4
   set  8,r5
% multiply data by coefficients
% read data in order and write result using addresses in memory
   vmulti_coef0 (order, memory)
% enale writing imaginary part
   en_img write
   addimm r6,1,r6 % counter += 1
   Compare counter,End
   comp r6,r7
   % Branch if less than
   bl l2
   nop % no operation after loop
   set  6,r0
   set  0,r1
% out processed data and enter new one
   io  (mem, vn)
% repeat again
   jumb l1
   nop
```

Figure 18 Implementation code example for 8 × 8 DCT.



```
% 64 tap filter
   set  64,r0
   set 64, r4
% coefficient start address in memory = 0
    set 0, r1
% data start address = 0
   set  0,r3
% insert data in order
   in  order
   l1:
   set 0,r6  % t
 l2:
```

```
% Mac,Get coef in descending order
% get coefficint from start address= t
  mov  r1, r6
   mac_coef2 (ordr,ordr,2) r3,r6
% next output
   addimm r6,1,r6
   Compare counter,End
   comp r6,r4
% branch if less than
   bl l2
   nop
% out processed data and inset new on in oder
   io (ordr,ordr)
   jumb l1
   nop
```

Figure 19 Implementation code example for 64 tap FIR.

5.1.2 Word and coefficient lengths

Then, Fixed-point simulations of a 1024 point DFT in WiMAX see Figure 15 reveal that a 26 bit (13 real and 13 imaginary) complex word length and 20 bit complex twiddle factors are sufficient to keep quantization noise power under system noise by 15 dB at 10^{-3} Bit Error Rate (BER).

5.2 Testing

Code for the application is written using custom mnemonics that combine HW-specific instructions with application-specific instructions. This then passes through a assembly compiler (designed by Matlab software) which generates the boot-loading and program object files.

When processing begins, the decoder accesses address zero in the boot loading ROM and reads initialization instructions. These instructions are mainly used for loading data and instructions from flash memory to the corresponding RAM memory in the system. Upon finishing, the decoder jumps to program memory and starts processing.

Table 2 Synthesis results (with memories)

Up to 8K point-DFT 1D symbol 26 complex word length	
Technology	IBM 130 nm CMOS technology (6 layers)
Volt	1.08 V
Libraries	Gates libraries: Typical (55°c)
	Fast library(125°c) used for worst case conditions
	Memories library: (125°c)
Number of Cells	57,906 cell
Area	0.612 × 0.6 (0.36) mm^2
Power	56 mw at 100 MHz
Max frequency	700 MHz

Figure 16 shows boot loading ROM initialization instructions. The initialization process may include preloading some or all of: program memory instructions (pm), coefficients memory (ws_mem1, ws_mem2), coefficients memory length (s_reg), read address memory (r_mem) and write addresses ram (w_mem). Boot-loading is necessary if the engine is to switch modes or standards on-the-fly. Otherwise program RAMs can be replaced by ROMs carrying the required instructions.

5.2.1 1024-point DFT

Figure 17 shows code example for 1024-point DFT. In/Out operations read two words at a time, therefore for N words it takes only N/2 clock cycles. To save on processing overhead special control signals like r2 = N/radix (used by address generator) are inserted directly to reduce computational load (by adding this instruction we save the power and area of a full divider). After each stage these parameters are modified, and loop for the

Table 3 Number of clock cycles and SQNR for 1D-DFT including data transfer times between the embedded engine and the host

N -point DFT	Cycles per	Latency @ 100 MHz	SQNR Scale factor	Scale factor						
DFT	DFT	(dB)	(*μs*)	*s*1	*s*2	*s*3	*s*4	*s*5	*s*6	*s*7
64	146	1.46	83.67	4	2	2				
128	278	2.78	86.16	4	2	2	2			
256	470	4.7	96.839	4	4	2	2			
512	1002	10.02	96.37	4	4	2	2	2		
1024	1898	18.98	99.1	4	4	2	2	2		
2048	4222	42.22	98.97	4	4	2	2	2	2	
4096	8318	83.18	97.84	4	4	4	2	2	2	
8192	18578	185.78	95.25	4	4	4	2	2	2	2

Table 4 Number of clock cycles for 1D-DCT including data transfer times between the embedded engine and the host

N -point DCT	Cyles per DCT	Latency @ 100 MHz (*μ*s)
64	177	1.77
256	592	5.92
512	1247	12.47
1024	2399	23.99

next radix stage. The twiddle factors in the last stage in DFT calculations are ones so we add choice (Scape multi, multi) to disable the twiddle factors generator and bypass multiplication stage. Thus the last radix instruction is separated from the loop. Then apply *io* instruction to export the processed symbol and import a new one. finally, jump to the first radix stage and so on.

8 × 8 DCT

Figure 18 shows code example for 8 × 8 DCT. Data is read, row by row, saving each row in v_n order discussed in Section 2. Then radix stages are applied until DFT calculations on all rows are completed. The data is multiplied by the twiddle factors, by getting addresses from read address memory (to arrange data after DFT operation and exchange row by column). Writing the result is in v_n order (construct v_n for new DCT-1D operation). The imaginary parts of result are set to zero by disabling writing of imaginary results. Then, the radix and multiplication stages are applied once more. Finally, the result is output in order and the new data is simultaneously loaded.

5.2.2 FIR filter

Figure 19 shows code example for a 64 tap FIR. Data is read in order. Multiply accumulate operation are applied on the data to generate first output $y(0)$. increment output index and apply MAC for next output and so on. Till the last output (N - 1) is generated. Finally, the result is output in order and the new data is simultaneously loaded.

5.3 Verification

Verification of these and other examples is through bit-matching the results of random input patterns with fixed-point results from fixed point golden files. The golden files are verified and tested against a floating point model to make sure they perform the needed tasks. The golden files are used to verify the RTL design by generating test cases, both directed and random.

Table 5 Number of clock cycles for 2D-DCT including data transfer times between the embedded engine and the host

N × *N*-point DCT	Cycles per DCT	Latency @ 100 MHz (*μ*s)
8 × 8	186	1.86
16 × 16	390	3.9
32 × 32	1724	17.24
64 × 64	6380	63.8

6 Implementation results and performance evaluation

6.1 Implementation

The engine is fully designed by the authors, using Verilog Hardware Description language and tested by applying various programming codes. Synthesis has been carried out using Cadence first encounter using IBM 130 nm CMOS technology. The post layout synthesis results report of the entire design with 26 bit complex word length, 20 bit complex twiddle factors and support for up to 8K-point DFT include system memories has been summarized in Table 2. The table also maintain all synthesis constraints. The engine parameters like the number of bits, memories size and types are parametrized to meet different requirements.

6.2 Performance evaluation

Tables 3, 4, and 5 show a summary of features of our proposed embedded processor.

Table 6 has a list of power consumption values for previously published articles. To eliminate the process factor to make the comparisons as fair as possible, the power consumption of each design has been normalized to 130 nm technology, 1.08 V and engine throughput by Equation (13) [17]. We define the parameter power efficiency which introduces how much power is taken to have certain throughput to make fair comparisons between the engines power in the case of they have same throughput. This shows, at the very least, that the proposed engine has a significant advantage in power consumption.

$$\begin{aligned} \text{Normalized power} &= \text{Power} \times \left(\frac{130}{\text{Technology}}\right) \times \left(\frac{1.08}{\text{Volt}}\right)^2 \\ \text{Power efficiency} &= \frac{\text{Normalized power}}{\text{Throughput}} = \frac{\text{Normalized power}}{1/\text{Time to end}} \end{aligned} \quad (13)$$

6.3 Discussion

Weidong and Wanhammar [14] proposed an pipeline ASIC for pipeline FFT processor. Here we prove our discussion in Section 3, the pipeline architecture have a higher throughput but loss on power efficiency.

The authors of [5,18,19] proposed memory based Application-Specific Integrated Circuit (ASIC) for scalable DFT engine. The proposed engine in [5] enables runtime configuration of the DFT length, where the supported lengths vary only from 16-points to 4096. while the proposed engine in [18] enables reconfigurable FFT Processor, the FFT lengths vary only from 128-points to 8192. and [19] can perform 64 2048-point FFT. This engines have high throughput rates. But, they only

Table 6 Number of clock cycles and SQNR for 1D-DFT including data transfer times between the embedded engine and the host

			Parameters								
Reference	**Implementation**	**Technology (nm)**	**volt (V)**	**Frequency (MHz)**	**Max-point DFT**	**Time to end (μs)**	**Power (mW)**	**Normalized power**	**Power efficiency**	**SQNR (dB)**
[14]	Pipeline HW	350	1.5	25	1K	40.96	200	35.6	1.4	N/A
[25]	Configurable HW	180	1.8	86	8K	805	75.51	19.6	15.8	N/A
[18]	Configurable HW	180	1.8	200	8K	395	117	84.2	33	N/A
[5]	Configurable HW	65	1.3	866	4K	7.1	35	48.3	0.3	71.90
[26]	Configurable HW	180	1.8	150	8K	138	350	91	12.5	N/A
[19]	Configurable HW	180	1.8	70	2K	224	140	36.4	8.15	N/A
[2]	DSP	-	-	100	1K	403.3	N/A	N/A	N/A	N/A
[20]	ASIP	250	2.5	100	4K	52.80	275	26.6	1.4	61.23
[21]	ASIP	180	1.8	300	1K	13.8	N/A	N/A	N/A	N/A
					1K	18.98	19	19	0.3	99.1
Proposed	ASIP	130	1.08	100	4K	42.2	25	25	1.05	97.84
					8K	185.7	56	56	10.3	95.25

[21]: Present the power consumption of functional unit and data address generator only

support certain kinds of algorithms for which they are designed.

In contrast, [2] used digital signal processors owing to their high reconfigurability and adaptive capabilities. Although DSP performance is improving, it is still unsuitable due to its high power consumption and low throughput. Hsu and Lin [2] proposed an approach for DFT implementation on DSP with low-memory reference and high flexibility, however it is optimized for 2^x-point DFT, It needs 40,338 cycles to complete one 1024-point DFT.

The third solution, [20,21] is the ASIP which compromises between the above solutions. Zhong et al. [20] proposed an DFT/IDFT processor based on multi-processor rings. This engine presents four processor rings (8, 16-Point FFT) and supports DFT lengths from 16-points to 4096. Guan et al. [21] proposed an ASIP scalable

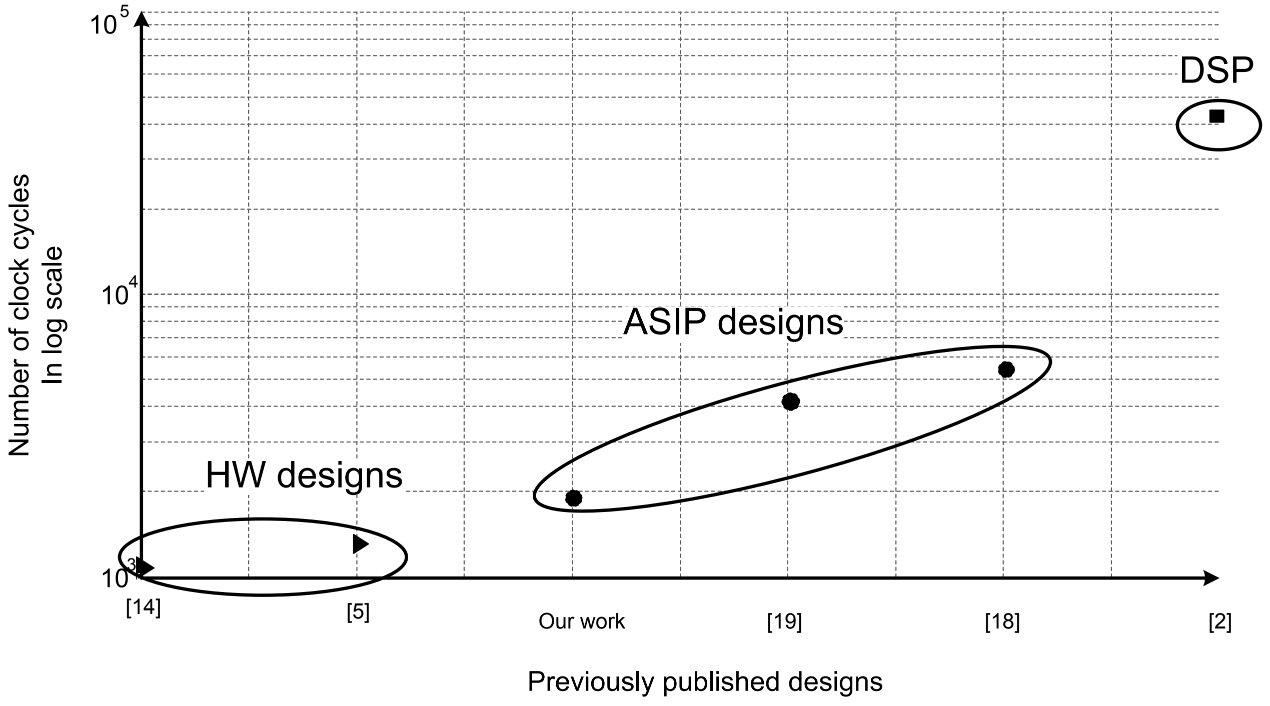


Figure 20 Number of clock cycles per one 1024-point DFT vs. implementation techniques.

Table 7 Applications that can be supported and the corresponding estimated clock frequency

DFT-1D applications	90 MHz	LTE, WI-MAX, WLAN, DVB-T, DVB-T, DVB-H, DAB, ADSLs and VDSL
DCT-2D applications	60 MHz	Low bit rate video conferencing, basic video telephony, interactive multimedia and digital TV-NTSC

architecture of any-point DFT at the expense of a large PE (contains an 8-point butterfly). the authors present only the power consumption of functional unit and data address generator so we did not include it in the table.

From our investigation, Figure 20 shows comparison between implementation techniques throughput.

Shah et al. [22] presents a pipelined scalable any-point DFT 1D/2D engine which requires 256 clock cycles for (16 × 16)-DFT 2D, while [23] and this design require 512 cycles. Nevertheless, Sohil Shah's proposal has higher area.

For DCT-1D, we use the mathematical algorithm in [12] which implements ASIC DCT-1D bulting blocks common with DFT. The engine has a throughput of one 512-point DCT per 1,771 cycles, and one 1024-point DFT per 3435 cycles.

For DCT-2D existence designs, the engine in [24] has been tailored to a particular application needing 80 cycle for (8 × 8)-DCT 2D, and programmable DSP [1] supports scalable ($N \times N$)-DCT 2D as N = 4-64. needs 2,538 cycles for (16 × 16)-DCT 2D,

The proposed engines are more power efficient than most of other proposed architectures in the literature. Engine features:

- More power efficient than most of other proposed architectures in the literature.
- Could be support many OFDM Systems with relatively low power.
- High reconfigurability which allows users to program a very wide range of applications with softwarelike ease.
- Support peripheral operations beside the main processes like CP remover which was need in the proposed WiMAX demo.
- Simple interfaces (FIFO interface) which handle data transfer between the engine and asynchronous blocks with different clock domains.
- The engine parameters like the number of bits, memories size and types are parameterized to meet different requirements and higher symbol lengths

The features that helped to get a high throughput which helped to get good power efficiency are:

- A new address generation scheme allows reading and writing the butterfly data in one clock cycle which allow performing 1 butterfly operation each clock. This reduce processing time by 50% without doubling the clock frequency no loss on power.
- The selection of radix-4 algorithm which have best power efficiency.
- Using HW accelerators accelerate the processing and reduce the complicity of the decoder.
- Using pipeline processing of the vector instructions is also accelerate the processing.
- Using simultaneously input and output data transformations with four data buses which reduce data transformations time by 75%.
- Reduce time to market by supporting a compiler tool for the engine with a simple instruction set
- The use of classified engines allows high degree of optimization.

7 Conclusion

In this article, we propose an ASIP design for low-power configurable embedded processor capable supporting DFT, DCT, FIR among other things. The defining feature of our processor is its reconfigurability supporting multiple transformations for many communication and signal processing standards with simple SW instructions, high SQNR, and relatively high throughput. The engine overall performance allows users to program a very wide range of applications with software-like ease, while delivering performance very close to HW. This puts the engine in an excellent spot in the current wireless communications environment with its profusion of multimode and emerging standards. The proposed embedded processor is synthesized in IBM 130 nm CMOS technology. The 8k-point DFT can 56 mW with a 1.08 V supply voltage to end in 13 μs with SQNR of 95.25 dB. Table 7 shows some applications which can be supported.

Acknowledgements
This study was part of a project supported by a grant from STDF, Egypt (Science and Technology Development Fund).

Competing interests
The authors declare that they have no competing interests.

Received: 19 June 2011 Accepted: 5 April 2012 Published: 5 April 2012

References
1. Liu X, Wang Y: Memory Access Reduction Method for efficient implementation of Vector-Radix 2D fast cosine transform pruning on DSP. *Proceedings of the IEEE SoutheastCon* 2010, 68-72.
2. Hsu YP, Lin SY: Implementation of Low-Memory Reference FFT on Digital Signal Processor. *Journal of Computer Science* 2008, **7**:545-549.
3. Frigo M, Johnson SG: The Design and Implementation of FFTW3. *Proceedings of the IEEE* 2005, **93**:216-231.

4. Jo BG, Sunwoo MH: **New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy.** *IEEE Transactions on Circuits and Systems* 2005, **52(5)**:911-919.
5. Jacobson AT, Truong DN, Baas BM: **The Design of a Reconfigurable Continuous-Flow Mixed-Radix FFT Processor.** *IEEE International Symposium on Circuits and Systems ISCAS* 2009, 1133-1136.
6. Hangpei T, Deyuan G, Yian Z: **Gaining Flexibility and Performance of Computing Using Application-Specific Instructions and Reconfigurable Architecture.** *International Journal of Hybrid Information Technology* 2009, **2**:324-329.
7. Poon ASY: **An Energy-Efficient Reconfigurable Baseband Processor for Wireless Communications.** *(IEEE) Trans VLSI* 2007, **15(3)**:319-327.
8. Iacono DL, Zory J, Messina E, Piazzese N, Saia G, Bettinelli A: **ASIP Architecture for Multi-Standard Wireless Terminals.** *Design, Automation and Test in Europe (DATE '06)* 2006, **2**:1-6.
9. Hassan HM, Shalash AF, Hamed HM: **Design architecture of generic DFT/DCT 1D and 2D engine controlled by SW instructions.** *Asia Pacific Conference on Circuits and Systems APCCAS 2010* 2010, 84-87.
10. Hassan HM, Shalash AF, Mohamed K: **FPGA Implementation of an ASIP for high throughput DFT/DCT 1D/2D engine.** *IEEE International Symposium on Circuits and Systems (ISCAS) 2011* 2011, 1255-1258.
11. Cooley JW, Tukey JW: **An Algorithm for Machine Computation of Complex Fourier Series.** *Mathematics of Computation* 1965, **19**:297-301.
12. Nguyen T, Koilpillai RD: **The theory and Design of Aribitrary-length cosine-modulated filter Banks and wavelets, satisfying perfect reconstruction.** *IEEE Transaction on signal processing* 1996, **44(3)**:473-483.
13. Braganza S, Leeser M: **The 1D Discrete Cosine Transform for Large Point Sizes Implemented on Reconfigurable Hardware.** *IEEE International Conference on Application-specific Systems, Architectures and Processors ASAP* 2007, 101-106.
14. Weidong Li, Wanhammar L: **A PIPELINE FFT PROCESSOR.** *IEEE Workshop on Signal Processing Systems, 1999. SiPS 99* 1999, **19**:654-662.
15. Chidambaram R, Leuken RV, Quax M, Held I, Huisken J: **A multistandard FFT processor for wireless system-on-chip implementations.** *Proc International Symposium on Circuits and Systems* 2006, 47.
16. He S, Torkelson M: **Design and Implementation of a 1024-point Pipeline FFT Processor.** *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference* 1998, 131-134.
17. Lin JM, Yu HY, Wu YJ, Ma HP: **A Power Efficient Baseband Engine for Multiuser Mobile MIMOOFDMA Communications.** *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMSI* 2010, **57**:1779-1792.
18. Sung TY, Hsin HC, Ko LT: **Reconfigurable VLSI Architecture for FFT Processor.** *WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS* 2009, **8.**
19. Lee YH, Yu TH, Huang KK, Wu AY: **Rapid IP Design of Variable-length Cached-FFT Processor for OFDM-based Communication Systems.** *IEEE Workshop on Signal Processing Systems Design and Implementation, 2006. SIPS '06* 2006, 62-65.
20. Zhong G, Xu F, Willson AN Jr: **A power-scalable reconfigurable FFT/IFFT IC based on a multi-processorring.** *IEEE Journal of Solid-State Circuits (JSSC)* 2006, **41**:483-495.
21. Guan X, Lin H, Fei Y: **Design of an Application-specific Instruction Set Processor for High-throughput and Scalable FFT.** *IEEE International Symposium on Circuits and Systems ISCAS* 2009, 2513-2516.
22. Shah S, Venkatesan P, Sundar D, Kannan M: **Low Latency, High Throughput, and Less Complex VLSI Architecture for 2D-DFT.** *International Conference on Signal Processing, Communications and Networking ICSCN* 2008, 349-353.
23. Shah S, Venkatesan P, Sundar D, Kannan M: **A Fingerprint Recognition Algorithm Using Phase-BasedImage Matching for Low-Quality Fingerprints.** *IEEE International Conference on the Image Processing* 2005, 33-36.
24. Tumeo A, Monchiero M, Palermo G, Ferrandi F, Sciuto D: **A Pipelined Fast 2D-DCT Accelerator for FPGA-based SoCs.** *IEEE Computer Society Annual Symposium on VLSI* 2007, 331-336.
25. Cho YJ, Yu CL, Yu TH, Zhan CZ, Wu AYA: **Efficient Fast Fourier Transform Processor Design for DVB-H System.** *proc VLSI/CAD symposium* 2007.
26. sung TY: **Memory-efficient and high-speed split-radix FFT/IFFT processor based on pipeline CORDIC rotations.** *IEEE proceedings, Image Signal Process* 2006, **153**:405-410.

doi:10.1186/1687-3963-2012-3
Cite this article as: Hassan *et al.*: Implementation of a reconfigurable ASIP for high throughput low power DFT/DCT/FIR engine. *EURASIP Journal on Embedded Systems* 2012 **2012**:3.