A Continuous-Flow Mixed-Radix Dynamically-Configurable FFT Processor

By

ANTHONY T. JACOBSON
B.S. (University of Idaho) December, 2004

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTERS OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Chair, Dr. Bevan Baas

Member, Dr. Venkatesh Akella

Member, Dr. Zhi Ding

Committee in charge
2007

Abstract

This thesis presents the design of a dynamically configurable Fast Fourier Transform (FFT) processor built around a continuous flow, mixed-radix architecture. Complex FFTs/inverse FFTs (IFFTs) of size 16- to 4096-point can be performed, with the option to turn on/off block floating point (BFP), and to divide the input data by 2 (to prevent overflow in the butterfly). An addressing scheme is presented to accomodate performing FFTs of any size. Minimizing twiddle factor data storage is discussed with a method to improve FFT accuracy and reduce the number of multiplications performed within an FFT. The datapath for this processor varies between 16- to 34-bits. Built in a 65 nm technology, a 1024-point FFT is performed in 1.29 μs and a 4096-point FFT is performed in 6.11 μs at a clock speed of 1.01 GHz. The accuracy is 80 dB for a 64-point FFT and 73 dB for a 1024-point random-data FFT. The processor consumes 250 mW at 1.3 V, and takes up an area of 1.01 mm^2.

Acknowledgments

It is seldom that we get the chance to express ourselves as genuinely as possible without having to fill some mold or template of what is expected of us; given this very opportunity, I am almost certain to mess it up.

Tackling my gratitude from the top-level and working down, I would like to thank my family first and foremost; they have provided the foundations on which I base my entire life around. From my mother's persistence in taking the higher (and more honorable) road in all situations, to my father's class and integrity, to my siblings (Lolo, Berry, and Stevo) ability to see humor in all situations, I have been truly fortunate in upbringing, and will never take it for granted. I love you guys!

Additionally, I would be morally remiss in not including my beautiful Rachel; although our life together is only just waxing, her support and love have provided me with the smiles and memories which will accompany us through to the end of our waning moments together. Rachel, you are the best part of my every day.

To the members of VCL - Wayne, Dean, Zhiyi, Tinoosh, Paul, Zhibin and Jeremy: your continual mentoring has provided me the ability to complete my studies. Without exaggeration, I could not have achieved this milestone.

To Professor Baas: your guidance and help have bolstered my potential academically and professionally. In addition, my project was based almost entirely upon your advice and knowledge, and I thank you profusely for it.

I did not anticipate this path in life, nor recognize my potential until I received the countenance of those individuals mentioned here, and others, too many to mention. I encourage anyone to accept the help of those around you, but never forget to be thankful and gracious in accepting this help. I know that if I, with a humble, rural upbringing, and a penchant for ninjas and very fine beer, am capable of the level of success I've experienced, anyone is.

Contents

List of Figures

Chapter 1

Introduction

The Fast Fourier Transform (FFT) is a fundamental component of many multiple digital signal processing (DSP) systems. Applications such as 802.11a/g/n [1] [2] [3], 802.16 [4], digital audio/video broadcasting [5] [6], and asymmetrical digital subscriber line/very-high-speed digital subscriber line (ADSL/VDSL) [7] with high real-time data throughput needs have resulted in the development of several dedicated high-speed FFT processing engines. Recent advancement in FFT processors has resulted in increased computational efficiency and reduced area requirements. In this thesis, we present a dedicated, dynamically configurable complex FFT processor capable of performing variable-size FFTs (from 16- to 4096-point) that combines these advancements with a dual-memory architecture and addressing scheme that allows for even greater throughput. The FFT processor utilizes a mixed radix-2/4, in-place single butterfly, continuous flow architecture with a symmetrically-reduced, accuracy-increased twiddle-factor ROM.

This processor was designed with two operating environments in mind: independent operation, and integration into a separate processing architecture, such as the Asynchronous Array of Simple Processors (AsAP) [8] [9]. The I/O requirements of separate processing architectures can be compensated for by a simple restructuring of the FFT processor's I/O controllers. The FFT processor design maintains full functionality in both environments, and can be modified simply to allow for a greater range of FFT size, differing FFT configuration, and reduction of area (through memory reduction). The FFT processor includes a dual-clock FIFO that allows it to asynchronously interface with external modules upstream and downstream from itself.

One of the first FFT processors was the Honeywell chip in 1988 [10]. Since then, a variety of FFT processors have been built, ranging in technologies from 1.5 μm [11] to 0.065 μm (this work). Prior to the implementation of this work, FFT processors could perform as many as 156,000 1024-pt FFTs per second [12], and could consume as little as 600 nW [13]. FFT sizes capable of being performed range from as small as 2-point [14] to as big as 8192-point [15]. Unfortunately, the SNR of FFT processors is seldomly reported, but can be as accurate as 65 dB for a 64-point FFT [16].

The goal of this project is to develop an FFT processor that is capable of performing a wide range of configurability options, including an FFT size range of 16- to 4096-point, complex FFT/inverse FFT (IFFT) functionality, dynamic configurability, block floating point (BFP)/fixed point, and the ability to scale input data to prevent overflow/saturation. Secondary goals of being able to perform 1,000,000 1024-point FFTs per second and achieve at least 70 dB SNR of accuracy for a 1024-point FFT were also established. While the actual performance of the FFT processor falls short of the speed goal (774,000 1024-point FFTs per second), the average SNR of a 1024-point FFT is 74.13 dB, and the chip hosts the specified range of configurability options. This chip was not designed with the philosophy of minimizing power consumption or chip area foremost in consideration, though achieved reasonable measures of both, primarily due to the technology used.

The first generation of the Asynchronous Array of Simple Processors (AsAP) mentioned in this thesis was jointly developed by Zhiyi Yu, Michael Meeuwsen, Ryan Apperson, Omar Sattari, Michael Lai, Jeremy Webb, Eric Work, Tinoosh Mohsenin, Mandeep Singh, and advisor Bevan Baas. The second generation of AsAP was developed by Dean Truong, Tinoosh Mohsenin, Zhiyi Yu, Wayne Cheng, Zhibin Xiao, Paul Mejia, Christine Watnik, Gouri Landge, Eric Work, Jeremy Webb, myself (Anthony Jacobson), and advisor Bevan Baas. The design for the FFT processor was built primarily upon the ideas of Bevan Baas. Layout was performed by Dean Truong. My contribution to this project includes the design, simulation, synthesis and testing of the FFT processor, including the creation of address generation algorithms, the continuous flow architecture control, the design of the datapath pipeline, the mixed-radix butterfly module, and the optimization of the SNR of FFTs.

This thesis assumes a basic-level knowledge of the FFT algorithm as presented by Cooley and Tukey [17], and will only briefly cover relevant FFT background and previous FFT processor information in Chapter 2. An overview of the chip's architecture will then be given in Chapter 3, focusing on the FFT butterfly and dual-memory configuration. This will be followed by an indepth look at the input data-control and configurability options, the addressing scheme, and the twiddle-factor ROM architecture. After this, the chip's specifications will be given in Chapter 4, along with a comparison to other recent dedicated FFT processor's speed, power, and FFT accuracy specifications. Finally, potential improvements and future work will be covered, followed by a brief summary of the processor and concluding remarks in Chatper 5.

Chapter 2

Background

The FFT is a common method of computing the Discrete Fourier Transform (DFT) [18]. The Cooley-Tukey algorithm is a very common FFT which recursively divides the DFT into smaller DFTs with multiplication by complex unity-magnitude numbers known as twiddle factors; organizations of DFTs into arrays of 2 or 4 smaller DFTs are known as radix-2 and radix-4 FFTs, respectively. A mixed-radix FFT (MRFFT) combines multiple radices (typically radix-2 and radix-4). Due to the lower number of complex multiplications needed when performing radix-4 computations (25% fewer) [19], efficient mixed-radix FFT's perform radix-4 calculations whenever possible. In an FFT, when N (generally complex) samples are inputted, N samples will be outputted after computation has taken place.

2.1 Butterfly Architecture

The primary computational unit in the FFT is the butterfly, which performs both complex multiplication by the twiddle factors with pieces of data in the sequence and the addition/subtraction of these products. A single radix-4 butterfly performs the computational equivalent of four radix-2 butterflies; a radix-4 butterfly can also be modified to function as a radix-2 butterfly. Figure 2.1 shows a radix-2 decimated in time (DIT) butterfly structure. Figure 2.2 shows a DIT radix-4 butterfly structure. The MRFFT architecture has been utilized in previous FFT processors [12] [20] [21] [16].

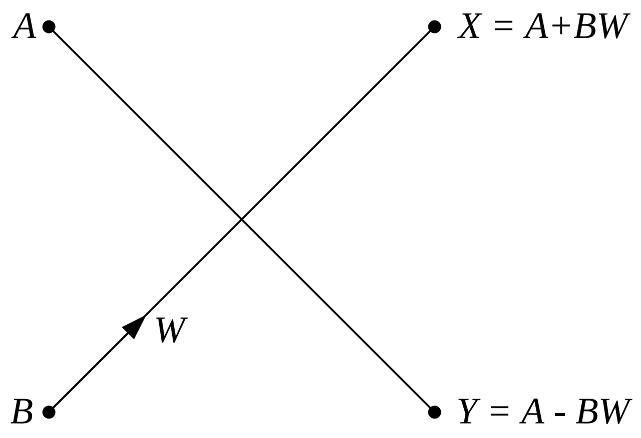

Figure 2.1: A radix-2 DIT butterfly.

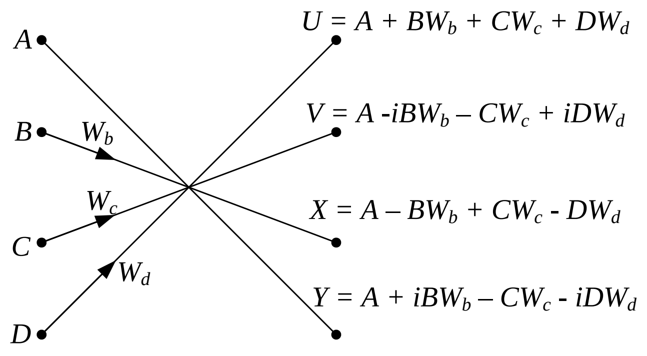
A
B
C
D
W_b
W_c
W_d
$U = A + BW_b + CW_c + DW_d$
$V = A - iBW_b – CW_c + iDW_d$
$X = A – BW_b + CW_c - DW_d$
$Y = A + iBW_b – CW_c - iDW_d$

Figure 2.2: A radix-4 DIT butterfly.

2.2 Memory-Based Architecture

The FFT algorithm consists of several stages of computation. Data storage is required between these stages, and can be handled several ways. The larger the FFT being performed, the larger the amount of data storage required. This relationship is linear.

To achieve high throughput, pipelined FFT architectures have been proposed that rely on additional processing elements in lieu of memory banks [22] [23]. These additional processing elements consume considerably more area than memory-based FFT processors [16], particularly if the "in-place" addressing scheme (where data is sent back to the same location in memory where it was retrieved) presented by Johnson [24] is implemented. As a result, the scale of the architecture expands rapidly as the size of the FFT increases, as

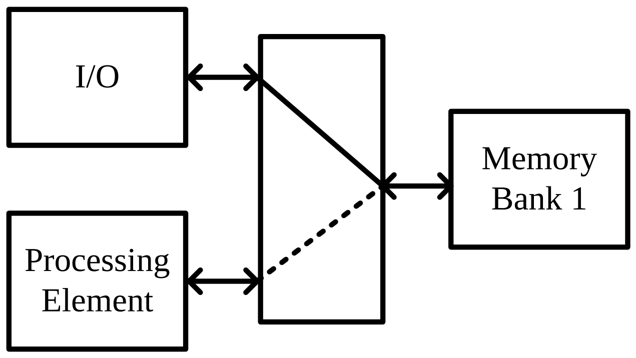

Figure 2.3: A single memory FFT processor architecture. Only the I/O or the processing element can access the memory. While the I/O is accessing memory, the processing element cannot operate, thus restricting system throughput.

the processing elements carry the burden of storing the extra scope of FFT data.

Single memory bank architectures have been proposed by Bidet [25], Baas [26] and more recently Sung [15] (see Figure 2.3). The benefit of a single memory bank architecture lies primarily in its smaller chip size. Additionally, accessing only a single memory bank requires less control than having to regulate memory accesses between multiple memory banks. The primary draw-back of this architecture is a lack of continuous operation of the processing element when the memory is busy receiving input data and sending output data via I/O. This results in a setback to throughput, as data cannot be sent to the FFT processor while the processor is computing the FFT.

A *continuous flow* architecture is one where the processing element (the butterfly) is constantly computing data, with no breaks between FFTs to accommodate I/O requirements. The idea of a continuous flow architecture is presented by Radhouane [27]; this continuous flow architecture allows for non-stop operation of the FFT processing element, though for a radix-2 butterfly. This idea is expanded upon by Baek [21], for an MR architecture, using two memory banks, each consisting of four separate memories of size $N/4$, where N is the size of the desired FFT.

In this architecture (see Figure 2.4), data is sent from I/O to memory bank 1. When all the data has been sent, the FFT processing element can begin computing the FFT on this data and the I/O can begin sending another set of data to memory bank 2.

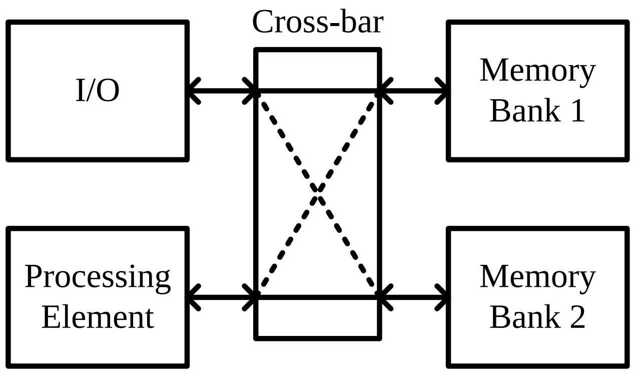

Figure 2.4: A continuous flow FFT processor architecture. While the I/O is connected to memory bank 1, the processing element is connected to memory bank 2. After the I/O and processing element have finished, they switch memory bank access; the I/O connects to memory bank 2 and the processing element connects to memory bank 1. Both the I/O and the processing element can never access the same memory simultaneously. As long as the I/O can supply enough data, the processing element can be continuously active, increasing throughput.

Once the FFT processing element has finished computing the FFT on the data in memory bank 1, it can begin to compute the FFT on the data in memory bank 2. While this new FFT is being computed, the I/O can read the data from memory bank 1, and, once this is finished, send a new set of data back to memory bank 1. The FFT processing element thus alternates between memory banks, and as long as new data is sent to the non-active memory bank, continuous operation of the processing element occurs, maximizing the throughput of the FFT processor.

2.3 Twiddle Factors

The twiddle factors needed in the computation of the FFT are represented by integer powers of the following equation:

$$W_N = e^{-i2\pi/N} \tag{2.1}$$

where N is the size of the FFT desired. This can be restated as:

$$W_N = e^{-i\theta_N} \tag{2.2}$$

where θ_N is equal to $2\pi/N$, and restated again utilizing Euler's identity:

$$W_N = \cos(\theta_N) + i\sin(\theta_N) \tag{2.3}$$

Numbers in the form of Equation 2.3 lay on the unit circle. FFT's require multiplication by various powers of twiddle factors [17]:

$$W_N^y = e^{-iy\theta_N} \tag{2.4}$$

As y in W_N^y increases linearly, the twiddle factor value rotates clockwise around the unit circle at a constant rate.

One method of attaining twiddle factors is to generate them; the CORDIC algorithm presents one such method of doing so [15]. A more common method is to store the twiddle factors in an on-chip ROM. A 4096-point FFT/IFFT requires 4096 distinct complex twiddle factor values, resulting in the need for a very-large ROM (often referred to as a W-ROM). The size of this ROM can be reduced considerably by manipulating the symmetry of the twiddle factors between 0 and 2π. A W-ROM storing just the twiddle factors from 0 to $\pi/2$ was used by Chang [28]. Symmetry was exploited further in [29], storing just the twiddle factor values from 0 to $\pi/4$, reducing the size of the ROM needed by an additional factor of 2. Thus, for a 4096-point FFT, a W-ROM consisting of 512 complex twiddle factors along with decoding logic can be used.

Chapter 3

Design

Our dynamically configurable, variable-length FFT processor features a mixed radix-2/4 continuous-flow architecture, a corresponding input data control scheme and address generating scheme, and a symmetrically-reduced twiddle-factor ROM. These components and the overall system design will be discussed below with analysis of associated performance, accuracy, and speed trade-offs. The FFT processor is capable of performing 16- to 4096-point complex FFTs/IFFTs, with or without block floating point(BFP) arithmetic, and with or without dividing incoming data by 2 (not dividing data by 2 could allow overflow in the butterfly; saturation occurs in this case).

3.1 Architecture Overview

The FFT processor is composed of an I/O half and a butterfly half, centered around two memory banks (see block diagram, Figure 3.1). The I/O half contains the modules that control the interface between the FFT processor and the user. The butterfly half contains the address generator, twiddle factor ROM (W-ROM), and the butterfly module. The memory banks are composed of simple mux logic and four 1k-word memories.

Additionally, the I/O component of the FFT processor interfaces with upstream and downstream modules asynchronously through the use of a dual-clock FIFO [30]. The upstream and downstream modules are assumed to operate on independent clock domains. These two modules are also assumed to contain their own dual-clock FIFOs, and be respon-

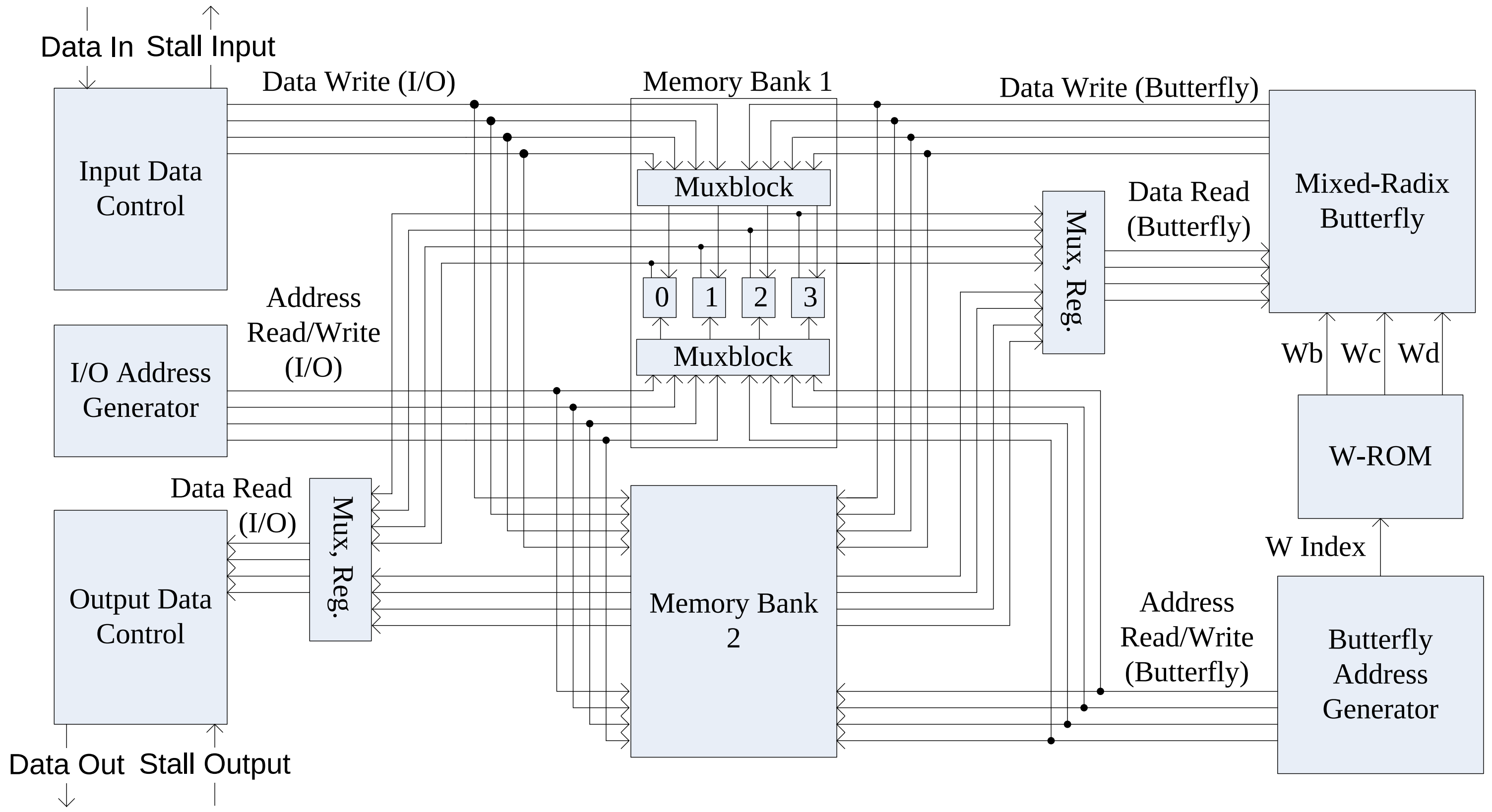

Figure 3.1: FFT Processor Block Diagram. This diagram highlights the top-level internal composition of the FFT processor, including the complete datapaths for both the I/O and butterfly portions of the processor. Memory bank 1 & 2 are identical. The component 'Mux, Reg.' consists of a multiplexor immediately followed by a register.

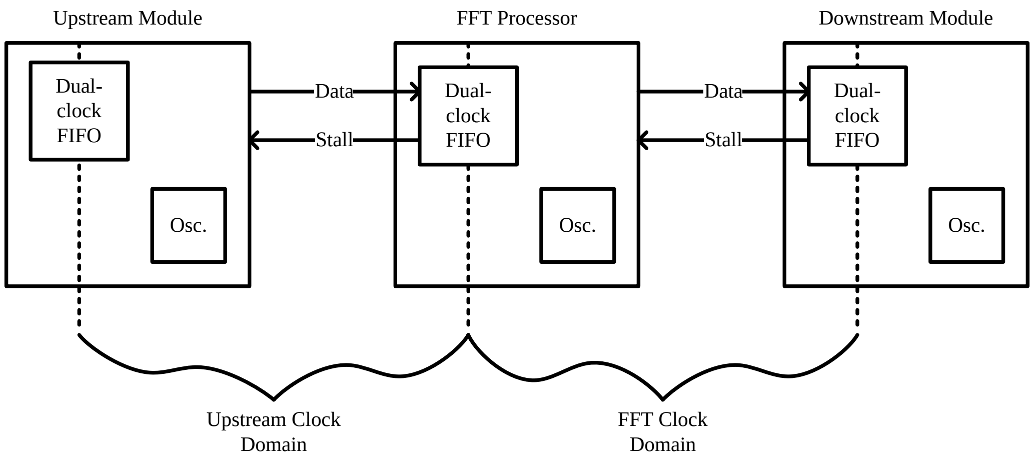

Figure 3.2: FFT processor clock domain. This figure shows the interface between the FFT processor and the upstream and downstream modules. Data is sent from the upstream module in the upstream module's clock domain, and arrives at the FFT processor's FIFO, as long as the FFT processor's stall signal is low. Likewise, the FFT processor sends data to the downstream module in its own clock domain so long as the downstream module's stall signal is low.

sive to stall signals (in the event of data congestion leading to the filling of a FIFOs buffer). This interface can be seen in Figure 3.2.

3.2 System Datapath

At the core of the FFT processor is the primary functional component, a radix-4 butterfly. The radix-4 butterfly is composed of 3 complex multipliers, 12 complex adders, and rounding logic. The butterfly input and output utilizes a 16-bit real and 16-bit imaginary word width. The processor can handle FFTs varying in size from 16-point to 4096-point. For FFTs of length 2^m with m even (16, 64, 256, 1024, 4096; integer powers of 4), the butterfly performs standard radix-4 functionality throughout the entire FFT. For m-odd sized FFTs (32, 128, 512, 2048), the butterfly functions as a radix-4 butterfly for the first $(m-1)/2$ stages, and then switches functionality to act as a radix-2 butterfly for the final needed radix-2 stage. This is easily done, since the equations for a radix-2 butterfly [19]:

$$X = A + BW \tag{3.1}$$

$$Y = A - BW \tag{3.2}$$

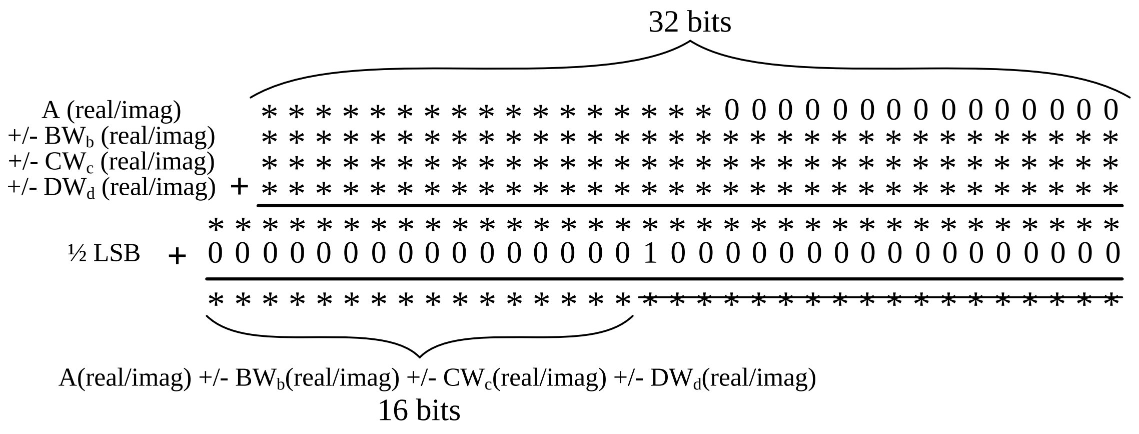

Figure 3.3: Rounding within the butterfly. After the real portions of A and the products BW_b, CW_c, and DW_d are summed, 1/2 an LSB is added to the sum, and the 16 most significant bits are kept, truncating and discarding the rest. For cases where subtraction is necessary, the values are first inverted, and then addition occurs similarly.

are functionally equivalent to the following radix-4 butterfly output equations (Figure 2.2):

$$X = A - BW_b + CW_c - DW_d \tag{3.3}$$

$$Y = A + iBW_b - CW_c - iDW_d \tag{3.4}$$

when the inputs B and D in Equations 3.3 and 3.4 are tied to zero. This occurs throughout the entire radix-2 stage.

The complex multipliers within the butterfly produce the products BW_b, CW_c, and DW_d. To achieve the highest level of accuracy, the maximum number of bits are preserved throughout the butterfly stage. After going through the multipliers, the real and imaginary parts of the products are each 32 bits long. Adding the real components of A, BW_b, CW_c, and DW_d together yields a sum that is 34 bits long. Rounding this data back to the 16 most significant bits involves adding 1/2 of a least significant bit of the future 16 bit sum and truncating the rest of the bits. Figure 3.3 shows the rounding performed after the addition of the real portions of the complex products are added together. Rounding is performed identically after the addition of the imaginary portions of the complex products are summed.

The butterfly is fed with data read from one of two banks of memory on chip. These two memory banks are each composed of four 1K x 32-bit memories, and are the

Data bit	Configurability function
15	Configuration enable
14	1 = IFFT, 0 = FFT
13	1 = divide input data by 2, 0 = do not divide input by 2
12	1 = fixed point arithmetic, 0 = block floating point arithmetic
11-8	size of FFT (see Table 3.2)
7-0	unused

Table 3.1: Composition of configuration word by bit. Bit 15 must be high in order for the FFT processor to recognize the configuration word.

foundation for the continuous-flow architecture. So long as the chip is fed with enough data, the continuous-flow architecture guarantees that the butterfly will be 100% active. An address generator controls accessing data from memory by producing memory read enable signals and sending addresses to memory to read the appropriate data. The address generator also sends the appropriate index value to the twiddle-factor ROM and controls the butterfly computation with an enable signal. The addresses for reading memory and the twiddle factor ROM depend on the type and size of the FFT being computed, which is determined by the configuration of the processor.

The configuration information is contained in the first valid piece of data sent to the FFT processor by the upstream module in a given data set. The breakdown of configuration bits is seen in Table 3.2. The configuration information, once received, is sent to the address generator by the FFT processor's I/O control logic. The I/O is also informed by the address generator when an FFT has been completed.

Each piece of data stored in memory is composed of a 16-bit real word concatenated with a corresponding 16-bit imaginary word. Four such 32-bit pieces of data are read from the memory bank simultaneously, one from each 1K memory, in order to keep the butterfly fed with data. This is accomplished by writing data to memory in such a way that regardless of the stage or butterfly the current FFT is on, each of the four butterfly inputs is fed with a piece of data from a separate memory in the memory bank. This data storage scheme is discussed below. The four words of data from memory are fed into four multiplexors, and the address generator creates select line values to send each piece of data to the appropriate butterfly input. Likewise, the four butterfly outputs are fed into four multiplexors, and the

Configuration word$_{11:8}$	Size of FFT
4'b0000	16-pt
4'b0001	32-pt
4'b0010	64-pt
4'b0011	128-pt
4'b0100	256-pt
4'b0101	512-pt
4'b0110	1024-pt
4'b0111	2048-pt
4'b1000	4096-pt

Table 3.2: Configuration size encoding. Bits 11:8 of the configuration word sent to the FFT processor before computing an FFT are encoded this way.

same select line values and memory addresses (both pipe-delayed) are used to send each piece of data to the correct memory and address (possible because of the nature of the in-place algorithm).

The continuous-flow architecture allows for one memory bank to be read from and written to by system I/O, while the other memory bank is read from and written to by the processing element. When an FFT on a set of data in one memory bank is complete, the processing element begins computing the FFT on the data in the other memory bank, if the data set has been completely written. This switching is handled by a a series of multiplexors controlled by a select signal from the FFT address generator. The signal is sent to multiplexors on the I/O side of the memory banks, and inverted to control the multiplexors on the processing side of the memory banks, guaranteeing that each memory bank will only be accessed by either the user or the processor (never both simultaneously).

The system I/O is kept to a minimum, consisting of input and output data buses, data valid signals, and stall signals. In order to accommodate the FFT processor's integration into AsAP, the data buses are limited to 16-bits wide. Each 32-bit data word is sent over two clock cycles, the 16-bit real portion, followed by the 16-bit imaginary portion. Each data word input must be accompanied by a valid signal in order to be recognized by the FFT processor. Likewise, each piece of data output by the processor will be accompanied by a valid signal. The output stall signal indicates to the module upstream that the FFT processor cannot presently accept additional data. The input stall signal from the downstream module is used to tell the FFT processor to halt outputting computed data.

The first word in a given data set sent to the FFT processor contains the configuration information for the FFT to be computed.

Within the butterfly, the output values are shifted by 2 bits in order to accommodate signal growth by a factor of 4. However, in a radix-4 architecture, there is a chance that data coming out of the butterfly can grow by more than a power of 4. This can occur because although the twiddle factors lie on the unit circle, the data on the butterfly inputs is not guaranteed to; as a result, the products in the equations seen in Figure 2.2 are not guaranteed to lie within the unit circle, and thus may have a magnitude of greater than 1. After thousands of simulations with random data, we have not had this occur, but the possibility led us to include the option to divide input data by 2, guaranteeing that multiplication by twiddle factors would result in products that lie within the unit circle, and that data coming out of the butterfly will be no more than 4 times greater than data going in. This allows us to keep the maximum number of bits in each word of data every FFT stage. If the user chooses not to divide the data by 2 initially, in the event of overflow within the butterfly, the data will be saturated to maintain the maximum amount of accuracy.

In order to further optimize accuracy within the FFT processor, block floating point (BFP) is included, with the configuration option to disable it if desired. BFP is performed by checking each piece of data coming out of the butterfly in any given stage and forwarding the minimum number of same-bits at the beginning of each word per stage ahead to the butterfly inputs of the next stage. The data entering the butterfly in the next stage is shifted by the minimum same-bit value of the previous stage in order to maximize the magnitude of the data set (preserving the maximum number bits during truncation within the butterfly). This is done once per stage, and immediately before the FFT is computed. One advantage of doing so is the ability to receive a set of data with a very small maximum magnitude and shift the data prior to computation so that the greatest number of significant bits is maintained throughout processing. This is exemplified by the case of a sinusoidal pulse input with magnitude 1/4. Without BFP, the SNR of a 1024-point FFT with this input is 25.2 dB, while with BFP the SNR is 76.3 dB. The FFT processor tracks the total number of bits shifted during BFP and concatenates it to the configuration information for output after the FFT is complete.

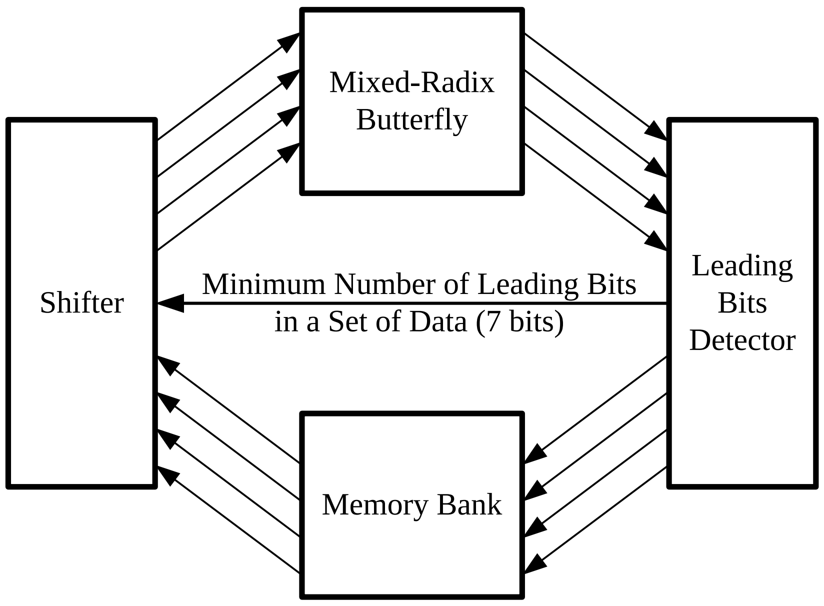

Figure 3.4: Block floating point diagram. After computation of each piece of data in a given data set, the number of identical leading bits in each of the butterfly output words is recorded. After the data set has been computed, the minimum number of leading bits in the entire data set is forwarded to a shifter, and the data is shifted to the left by this number (prior to entering the butterfly).

After the FFT has been computed, the first piece of data the processor outputs is the configuration information for the data set which was just processed. The structure of the word is the same as the input configuration word, except the last eight bits of the bus represent the number of bits the data was shifted due to BFP. After this, the computed data is read from memory and sent out to the downstream module (as long as the downstream module's stall signal is low).

3.3 Input Data Control

For an FFT, bit reversal must take place within the FFT, or at the input or output in order to correctly compute the FFT [31]. In this DIT FFT processor, bit reversal is done initially, which to simplifies outputting the computed data (data will be read out serially). For FFTs of size 2^n where n is even, bit reversal is done as follows (for a data word m):

$$m_{n-1}m_{n-2}...m_3m_2m_1m_0 => m_1m_0m_3m_2...m_{n-1}m_{n-2} \tag{3.5}$$

For FFTs of size 2^n with n odd, where a radix-2 stage of butterflies is needed at the end of the FFT, bit reversal is done as follows:

$$m_{n-1}m_{n-2}...m_4m_3m_2m_1m_0 => m_0m_2m_1m_4m_3...m_{n-1}m_{n-2} \tag{3.6}$$

The index used to identify each piece of data for bit reversal is implemented with an incrementer. The size of the incrementer is dependent on the size of the FFT being performed. In addition, the incrementer counts up for an FFT, and down for an IFFT; indexing data this way allows an IFFT to be performed with the exact same data writing and address generation scheme presented here for an FFT. It is this index that is bit reversed to determine the address and memory to which the associated piece of data will be written.

The four 1k memories within each memory bank have one read port and one write port. This restricts us to reading and writing one word of data per clock cycle; since each butterfly requires four words of input data and outputs four words of data per clock cycle, achieving maximum throughput is accomplished when each memory is written to and read from every clock cycle within an FFT stage. In order to satisfy this requirement, a scheme was developed independently from but similar to [16] for writing input data to memory. The pattern of this scheme is found in Table 3.3. Such a pattern ensures that regardless of the current stage within the FFT, each of the four pieces of data needed for a particular butterfly will come from different memories. This is seen in part by examining the sample butterfly inputs presented in Table 3.4.

This scheme is easily realized in hardware. The resulting indices are as follows:

$$data_{index} = m_{n-1}m_{n-2}\ ...\ m_3m_2\ m_1m_0 \tag{3.7}$$

The address each piece of data will be written to is equal to the first $n-2$ bits of the index:

$$address = m_{n-1}m_{n-2}...m_3m_2 \tag{3.8}$$

Determining which memory the data will be written to is more difficult. Adding together each pair of grouped bits in the index, and then employing mod-4 arithmetic to the sum yields the desired memory:

$$memory = (m_{n-1}m_{n-2} + ... + m_3m_2 + m_1m_0)\text{mod}4 \tag{3.9}$$

Address in Memory	Memory 0	Memory 1	Memory 2	Memory 3
0	**0**	**1**	**2**	**3**
1	7	**4**	5	6
2	10	11	**8**	9
3	13	14	15	**12**
4	19	**16**	17	18
5	22	23	20	21
6	25	26	27	24
7	28	29	30	31
8	34	35	**32**	33
9	37	38	39	36
10	40	41	42	43
11	47	44	45	46
12	49	50	51	**48**
13	52	53	54	55
14	59	56	57	58
15	62	63	60	61
16	67	**64**	65	66
17	70	71	68	69
18	73	74	75	72
19	76	77	78	79
20	82	83	80	81
21	85	86	87	84
22	88	89	90	91
23	95	92	93	94
24	97	98	99	96
25	100	101	102	103
26	107	104	105	106
27	110	111	108	109
28	112	113	114	115
29	119	116	117	118
30	122	123	120	121
31	125	126	127	124
32	130	131	**128**	129
⋮	⋮	⋮	⋮	

Table 3.3: FFT input data index addressing scheme, post bit reversal. This addressing scheme shows to which memory each piece of input data is written, and which address in the memory that the data is written to. This scheme is the same regardless of the length of FFT desired. The bold indices represent the first four pieces of data sent to the butterfly at the beginning of each FFT stage.

Stage	Butterfly #	Inputs
0	0	**0, 1, 2, 3**
0	1	4, 5, 6, 7
0	2	8, 9, 10, 11
0	3	12, 13, 14, 15
...	...	...
1	0	**0, 4, 8, 12**
1	1	1, 5, 9, 13
1	2	2, 6, 10, 14
1	3	2, 7, 11, 15
...	...	...
2	0	**0, 16, 32, 48**
2	1	1, 17, 33, 49
2	2	2, 18, 34, 50
2	3	3, 19, 35, 51
...	...	...
3	0	**0, 64, 128, 196**
3	1	1, 65, 129, 197
3	2	2, 66, 130, 198
3	3	3, 67, 131, 199
...	...	...

Table 3.4: The first four butterfly input indices for the first four stages of an FFT. The highlighted indices in the first butterfly of each stage correlate with the highlighted indices in Table 3.3.

Implementing Equation 3.8 is trivial, and implementing Equation 3.9 requires no more than six 2-bit adders (the modular arithmetic is performed by simply discarding the overflow and keeping the two least-significant bits).

Data is read out similarly. After the configuration information of the computed FFT is sent out to the user, the computed data is sent out. Since bit reversal has already taken place, the memory and address generation is identical to Equations 3.8 and 3.9.

3.4 Address Generation

The address generator is the primary controller behind the actual FFT computation itself. When a valid configuration word is received, the address generator initializes itself to begin an FFT of the requested size and specifications, once the entire data set has been written to memory. The address generator is responsible for producing the addresses

Stage	Address Incrementer Composition
0	$g_r g_{r-1} \dots g_1 g_0$
1	$g_{r-2} \dots g_1 g_0 b_1 b_0$
2	$g_{r-4} \dots g_0 b_3 \dots b_0$
$\vdots$	$\vdots$
x-2	$g_1 g_0 b_{s-2} \dots b_1 b_0$
x-1	$b_s b_{s-1} \dots b_1 b_0$

Table 3.5: Composition of the address incrementer. The address incrementer signal is separated into a group (g) and butterfly (b) counter for an x-stage FFT.

needed to read data from memory (the same addresses are used to write computed data back to memory due to the in-place structure of the FFT algorithm, and are pipe-delayed accordingly), the memory write enable signals, the index needed to access the appropriate twiddle factor ROM values, and the select values for the multiplexors that distribute data between the memories and the butterfly inputs and outputs. Additionally, the address generator indicates to the FFT processor's I/O control when an FFT has been completed.

Each stage of the FFT (regardless of the size of the FFT being computed) has a unique group counter and butterfly counter. The values of these counters are derived from the value of a primary address incrementer. After every address generation, the incrementer advances. The first stage of an N-point FFT is comprised of $N/4$ radix-4 butterflies within a single group. The second stage consists of $N/16$ radix-4 butterflies in each of 4 groups, and so forth. If needed, a last radix-2 stage will be broken down into one group of radix-2 butterflies. The composition of the incrementer signal is broken down as seen in Table 3.5.

As mentioned previously, a scheme has been implemented to write data to memory so that regardless of stage and butterfly, all four butterfly inputs will come from different memories. With each progressive stage, the data needed for each butterfly comes from progressively distant memory locations. Additionally, outside of the first stage (where for each butterfly, memory read addresses are constant across all 4 memories), addresses rotate in groups of 4 butterflies. For instance, if the addresses a, b, c, and d are generated for the first butterfly in a group of 4:

$$(addr0, addr1, addr2, addr3) = a, b, c, d \tag{3.10}$$

then the next three sets of butterfly addresses will look like:

$$(addr0, addr1, addr2, addr3) = d, a, b, c \tag{3.11}$$

$$(addr0, addr1, addr2, addr3) = c, d, a, b \tag{3.12}$$

$$(addr0, addr1, addr2, addr3) = b, c, d, a \tag{3.13}$$

respectively.

The addresses within each group of butterflies have a common prefix. To simplify address generation, this common prefix is assigned the value of the group number. The remainder of the address is assigned a base value for the first butterfly input with an offset added to the other butterfly inputs outside of the first stage. The address thus takes the form:

$$address = \{group\ number, (base + offset)\} \tag{3.14}$$

The base value grows in size by 2 bits with each progressive FFT stage. The first two bits of the correct base value for each butterfly is determined by subtracting each group of 2 bits of the group and butterfly counters from an initial base value of 0 (where g represents the value of the group counter, and b represents the value of the butterfly counter):

$$base_{(n-1):(n-2)} = 00 - g_r g_{r-1} - ... - g_1 g_0 - b_s b_{s-1} - ... - b_1 b_0 \tag{3.15}$$

The additional bits of the base signal (only needed for stages 3-6) are assigned the value of all but the last two bits of the butterfly signal:

$$base_{(n-3):0} = b_s b_{s-1} ... b_3 b_2 \tag{3.16}$$

The offset added to each base value is determined by which memory is being read from; the length of the offset is dependent on the current stage in the FFT and is equal to the length of the base signal. The first two bits of the offset are:

$$(memory\ 0\ offset)_{(n-1):(n-2)} = 00 \tag{3.17}$$

$$(memory\ 1\ offset)_{(n-1):(n-2)} = 01 \tag{3.18}$$

$$(memory\ 2\ offset)_{(n-1):(n-2)} = 10 \tag{3.19}$$

Butterfly #	1^{st} Input Index	2^{nd} Input Index
1	1	$(N/2)+1$
2	2	$(N/2)+2$
...	...	...
$N/2$	$N/2$	N

Table 3.6: Radix-2 stage butterfly inputs indices. In an N-point FFT, if a radix-2 stage is needed, the inputs indices for that stage are generated this way.

$$\textit{(memory 3 offset)}_{(n-1):(n-2)} = 11 \tag{3.20}$$

The remaining bits of the offset (if needed) are assigned to 0.

This addressing scheme operates independently of the length of the FFT being performed. The address generator keeps track of the FFT's current stage; as the FFT progresses from one stage to the next, the appropriate adjustments to the length of the base values, offsets, and butterfly and group counters are made. When reading and writing occurs, the address generator produces the necessary memory enable signals. After the final stage of the FFT, the address generator indicates to the FFT I/O that the FFT has finished.

The index required for the twiddle factor ROM is dependent on the butterfly count signal, and as a result is dependent on the current stage of the FFT. The index takes the form:

$$\textit{twiddle factor index} = \{b_s b_{s-1} ... b_1 b_0, 000...\} \tag{3.21}$$

The index is 10 bits long; $(10-s)$ 0's are concatenated at the end to make up the difference in length between the index signal length and the butterfly count signal length.

For instances where a radix-2 stage is needed (at the end of odd power of 2 sized-FFTs), addresses are generated slightly differently. As seen in Table 3.6, the data index for the second butterfly input is always $(N/2)$ above the first input. The addressing scheme used to write the data to memory initially accounts for this, and guarantees that both inputs to the butterfly will come from different memories. Determining which memory to read the first input's data from is identical to the method in Equation 3.9. The second input's data is read from the next sequential memory, as seen in Table 3.7.

1^{st} Input Memory	2^{nd} Input Memory
Memory 0	Memory 1
Memory 1	Memory 2
Memory 2	Memory 3
Memory 3	Memory 0

Table 3.7: Radix-2 stage input data memory locations. This table shows which memory the second butterfly input data is stored, given the memory of the first.

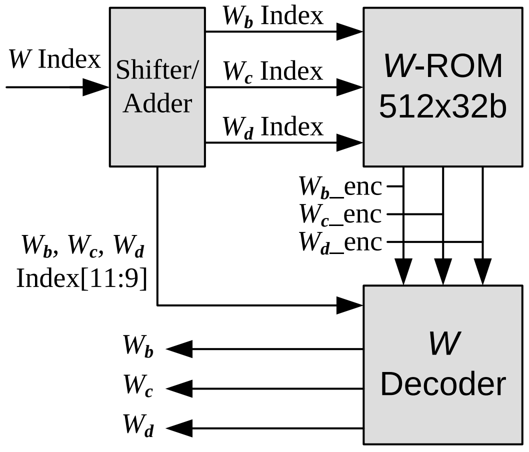

Figure 3.5: Twiddle factor ROM architecture

The address for the first input of an FFT of size N is:

$$\textit{radix-2 butterfly input 1 address} = \textit{butterfly count} \tag{3.22}$$

and the address for the second input, found by adding an offset of N/2 is:

$$\textit{radix-2 butterfly input 2 address} = \textit{butterfly count} + N/2 \tag{3.23}$$

The twiddle factor index is determined similarly to Equation 3.21.

3.5 Twiddle Factor ROM

In a radix-4 FFT, the twiddle factors W_b, W_c, and W_d, are all of the form of Equation 2.4. W_c and W_d can be calculated if W_b is known by using the following relationships [32]:

$$W_c = W_b^2 \tag{3.24}$$

$$W_d = W_b^3 \tag{3.25}$$

Writing W_b in the form:

$$W_b = e^{-i\theta_b} \tag{3.26}$$

yields:

$$W_c = (e^{-i\theta_b})^2 = e^{-i2\theta_b} = e^{-i\theta_c} \tag{3.27}$$

and:

$$W_d = (e^{-i\theta_b})^3 = e^{-i3\theta_b} = e^{-i\theta_d} \tag{3.28}$$

which makes it is easy to see:

$$\theta_c = 2\theta_b \tag{3.29}$$

$$\theta_d = 3\theta_b \tag{3.30}$$

Thus, squaring and cubing W_b is performed by rotating around the unit circle two and three times the angle of W_b. Since this angular rotation increases linearly, θ_b, θ_c, and θ_d are used to index the desired twiddle factor in the twiddle factor ROM, and are calculated from the index in Equation 3.21. The twiddle factor ROM (seen in Figure 3.5) first takes the twiddle factor index from the address generator and produces these three indices based upon it. These indices (Equations 3.31, 3.32 and 3.33) are easily calculated in binary. The index for W_b is equivalent to the index produced by the address generator:

$$W_b\_index = \{00, w\_index\} \tag{3.31}$$

Squaring W_b doubles the rotation of W_b around the unit circle, and as a result, the value of W_c is found in the twiddle factor ROM at the index that is twice that of W_b. Thus, the index for W_c is easily determined by shifting the index of W_b by one bit:

$$W_c\_index = 2(W_b\_index) = \{1'b0, w\_index, 1'b0\} \tag{3.32}$$

W_d is calculated in a similar fashion, by tripling the index of W_b. This is done by adding the indices for W_b and W_c:

$$W_d\_index = W_b\_index + W_c\_index = \{1'b0, w\_index, 1'b0\} + \{2'b00, w\_index\} \tag{3.33}$$

All of these equations are implemented easily in hardware with a shifter and an adder.

w_index (11:9)	Location on unit circle (θ)	Decoded (real)	Decoded (imag)
3'b000	$0 < \theta \leq -\pi/4$	α[31:16]	-α[15:0]
3'b001	$-\pi/4 < \theta \leq -\pi/2$	α[15:0]	-α[31:16]
3'b010	$-\pi/2 < \theta \leq -3\pi/4$	-α[15:0]	-α[31:16]
3'b011	$-3\pi/4 < \theta \leq -\pi$	-α[31:16]	-α[15:0]
3'b100	$\pi < \theta \leq 3\pi/4$	-α[31:16]	α[15:0]
3'b101	$3\pi/4 < \theta \leq \pi/2$	-α[15:0]	α[31:16]
3'b110	$\pi/2 < \theta \leq \pi/4$	α[15:0]	α[31:16]
3'b111	$\pi/4 < \theta \leq 0$	α[31:16]	α[15:0]

Table 3.8: Twiddle factor decoding at angle θ and ROM value α. When the twiddle factor index is sent to the W-ROM, the first three bits are used to decode the value read from the ROM (in this case α). This table shows the decoding for each possible octant of the unit circle.

The first three bits of each of these indices are sent ahead to a decoder; these bits represent the unit circle octant that the twiddle factor lays in. The trailing nine bits of these indices are sent to the symmetrically-reduced encoded twiddle factor ROM. The ROM itself only stores the positive magnitude of the twiddle factor values from 0 to $-\pi/4$, so these nine-bit pieces of the W-indices reference a value stored within this range. Every twiddle factor not found in this range correlates with a value of equivalent magnitude located within this range. This value is read from the W-ROM and sent to the decoder along with the first three bits of the indices (the octant indicators). The decoder then analyzes the octant indicators and decodes the ROM values to the appropriate format. This decoding is shown in Table 3.8, where the encoded value read from the ROM is "α". Storing the data in this way reduces the ROM to 1/8 the size it needs to be otherwise.

To represent all possible twiddle factor values from -1 to +1, a 2.14 number format must be used. To improve accuracy, a 1.15 number format was used to store twiddle factor values; this allows for greater accuracy in computation at the cost of not being able to store the value "+1" in the ROM. Detecting when a twiddle factor of "+1" is needed and bypassing multiplication altogether solves this problem and saves a large number of power-consuming multiplications within the butterfly. When multiplying by "+1", or "1 + 0i", in a complex multiplier, the product is equal to the multiplicand. The multiplier can be bypassed then by forwarding the multiplicand to the output of the multiplier. The amount of multiplications this saves varies based on FFT size. The total number of multiplications

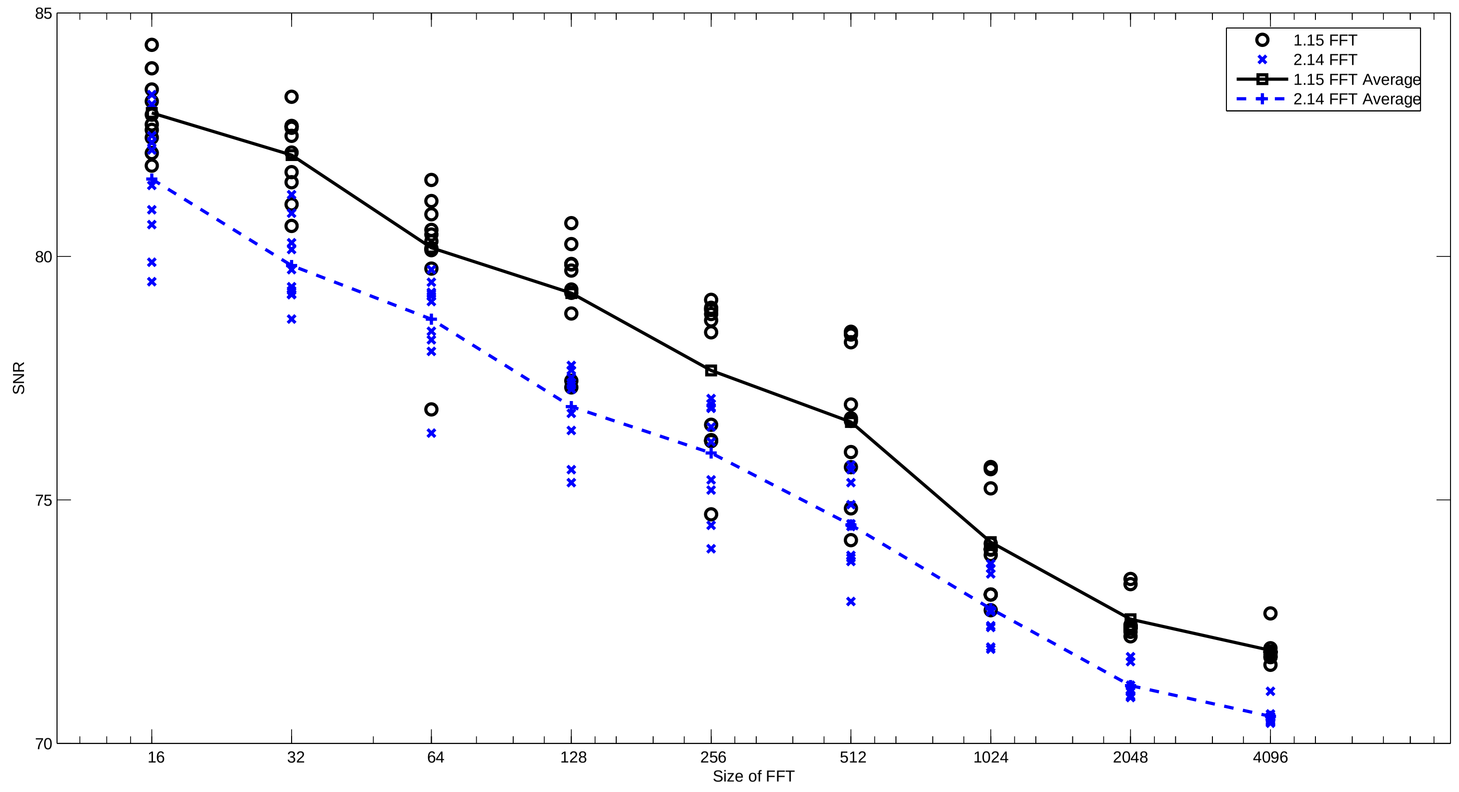

Figure 3.6: 1.15 vs. 2.14 W-ROM FFT accuracy. This figure shows the increase in accuracy seen when a 1.15 number format W-ROM table is used vs. when a 2.14 number format W-ROM table is used. Ten runs of each size FFT were made for each number format, and the average SNR was mapped for each size. The dotted line indicates the 1.15 format W-ROM SNR average, and the dashed line represents the 2.14 format W-ROM SNR average.

FFT Size	Mults. by 1	Total Mults.	% Mults. by 1
16	15	24	62.50%
32	31	64	48.44%
64	63	144	43.75%
128	127	352	36.08%
256	255	768	33.20%
512	511	1792	28.52%
1024	1023	3840	26.64%
2048	2047	8704	23.52%
4096	4095	18432	22.22%
1,048,576	1,048,575	7,864,320	13.33%
1,073,741,824	1,073,741,823	12,079,595,520	8.89%

Table 3.9: Multiplications by +1 by FFT size. This table shows the number of multiplications by +1 in for a variety of FFT sizes, and what fraction of the total multiplications in the FFT they represent. Multiplications by +1 for a 1-million point and 1-billion point FFT were included for perspective.

in an even power-of-2 FFT is (for an N-point FFT):

$$\text{Mults. per FFT(even)} = (N/4)(3)(\log_4(N)) \tag{3.34}$$

For an odd power-of-2 FFT, the number of multiplications is:

$$\text{Mults. per FFT(odd)} = (N/4)(3)(\log_4(N/2)) + N/2 \tag{3.35}$$

For even power-of-2 FFTs, the number of multiplications saved by bypassing multiplications by “+1” is:

$$\text{Mults. saved(even)} = 3(N/4 + N/4^2 + ... + N/4^{\log_4 N}) \tag{3.36}$$

For odd power-of-2 FFTs, the number of multiplications saved is:

$$\text{Mults. saved(odd)} = 3(N/4 + N/4^2 + ... + N/4^{\log_4 N/2}) + 1 \tag{3.37}$$

The percentage of multiplications saved is shown in Table 3.9. Because of the lack of multiplications by “+1” in the radix-2 stage of odd power-of-2 FFTs (1 per stage), the percentage of multiplications saved relative to even power-of-2 FFTs is somewhat reduced. Further, to achieve the same amount of accuracy while maintaining the 2.14 number format would require a 17-bit ROM table, and a 16 x 17 multiplier. Synthesizing components of these specifications results in an increase of area of 4% in the butterfly, and 6% in the twiddle factor ROM.

FFT Size	1.15 Number Format	2.14 Number Format	Difference
16	82.94 dB	81.58 dB	1.35 dB
32	82.08 dB	79.81 dB	2.26 dB
64	80.17 dB	78.71 dB	1.46 dB
128	79.24 dB	76.91 dB	2.33 dB
256	77.65 dB	75.96 dB	1.69 dB
512	76.59 dB	74.48 dB	2.11 dB
1024	74.13 dB	72.76 dB	1.36 dB
2048	72.54 dB	71.18 dB	1.36 dB
4096	71.90 dB	70.55 dB	1.35 dB

Table 3.10: Average SNR for 10 FFTs of size 16- to 4096-point for 1.15 and 2.14 W-ROM formats. This table indicates the increase in SNR for FFTs of all sizes when a 1.15 number format is used in a twiddle factor ROM.

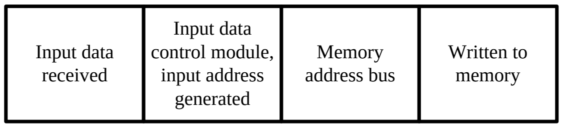

Figure 3.7: Input data pipeline. This pipeline shows the datapath of signals involved in writing input data to memory.

The increase in SNR accuracy due to the 1.15 twiddle factor format (in lieu of the 2.14 number format) is shown in Figure 3.6. Note that for the odd power-of-2 FFTs, the average SNR increase is 2.02 dB, which is greater than the average for even power-of-2 FFTs, which is 1.45 dB (see Table 3.10). This difference is due to a difference between the radix-2 butterfly and the radix-4 butterfly; in the radix-2 butterfly, the outputs are shifted only one bit, whereas in the radix-4 butterfly, outputs are shifted 2 bits to compensate for a greater data magnitude increase within the butterfly. As a result, the radix-2 outputs lose less data to truncation and rounding than the radix-4 outputs, and a greater SNR increase is seen.

3.6 Data Pipelines

Data is received from an external source and written to memory. If a valid piece of data is received, the address generator produces the next index in the FFT sequence, and both are sent to memory. The input data pipeline is seen in Figure 3.7.

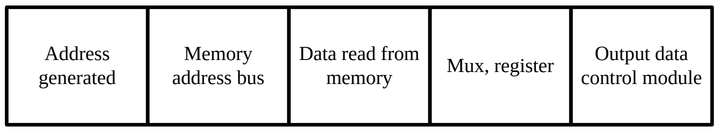

Figure 3.8: Output data pipeline. This pipeline shows the datapath of signals involved in outputting data from the FFT processor to an external source.

Similarly, to send data out to an external source, the address generator produces the next index in the sequence, and the address is sent to memory. Once the piece of data is read from memory, it is sent to the mux/register component. From there, it travels to the output data control module. The data, along with a valid signal, is then sent out to an external source as long as the stall output signal is low. When the stall output signal goes high, the data output control module will finish sending the data word it is currently sending and wait for the stall signal to go low. The output data pipeline is seen in Figure 3.8.

More complicated than the I/O pipelines is the butterfly pipeline, which highlights the datapath of signals involved in computing a butterfly within the FFT. Memory addresses corresponding to the correct data indices are generated by the address generator. Simultaneously, the address generator produces the twiddle factor ROM index. The memory addresses are pipe-delayed 6 clock cycles for use in writing the computed data back to memory (due to the in-place algorithm). The correctly encoded twiddle factors are then read from the ROM, and the butterfly input data is read from memory. The butterfly input data is sent to the mux/register module as the twiddle factors are decoded. Both the twiddle factors and the butterfly input data arrive at the butterfly at the same time. Regardless of whether the butterfly is performing as a radix-2 or radix-4 butterfly, 3 complex multiplications occur. If acting as a radix-2 butterfly, two of the complex multiplications have multiplicands of 0. In the next pipe stage, 12 complex additions are performed, with the proper rounding occurring afterwards. Finally, the pipe-delayed addresses are sent along with the butterfly output data to memory, where it is written back to the same location it was accessed from. The butterfly pipeline is seen in Figure 3.9.

Address Generator	Data addresses generated, W index generated	Address pipe-delay register 1	Address pipe-delay register 2	Address pipe-delay register 3	Address pipe-delay register 4	Address pipe-delay register 5	Address pipe-delay register 6		
W-ROM		W-index	Read encoded W-values	Decode W-values					
Memory Bank		Data addresses	Read butterfly input data					Butterfly output data, pipe-delayed data addresses	Write butterfly output data
Butterfly					W-values, butterfly input data	Complex multiplications (3)	Complex additions (12), rounding		
Mux/ Register Module				Butterfly input data					

Figure 3.9: Butterfly data pipeline. This pipeline shows the datapath of signals involved in reading data from memory, reading twiddle factors from the W-ROM, sending both to the butterfly for computation, and sending the computed data back to its original location in memory.

Chapter 4

Performance

This chip was synthesized using the ST 65 nm cell library with Design Compiler, and layout was performed with Encounter. To the best of our knowledge this is the first dedicated FFT processor to be built in a 65 nm technology. The processor's layout is shown in Figure 4.1. This chapter highlights the area and performance specifications of this work.

4.1 Area composition

Module	Synthesized Area	% of Total Area
Address Generator	2520	0.5%
Butterfly	61484	11.6%
Input Data Control	5702	1.1%
Output Data Control	5866	1.1%
Block Floating Point	1960	0.4%
Twiddle Factor ROM	18060	3.4%
Multiplexor Logic	19440	3.7%
Delay Registers	3382	0.6%
Memory (8)	412341	77.6%
Area(Total)	530755	100%

Table 4.1: Area composition of synthesized FFT processor by module, μm^2. This table shows the relative sizes of the individual modules within the FFT processor, not including the FIFO or oscillator.

The area breakdown by module of the synthesized chip is seen in Table 4.1. The memory and the butterfly account for almost 90% of the total chip area. As a result, the addition of features such as block floating point arithmetic and the variable-length address generator come at a negligible cost.

4.2 Performance & Comparison

It is difficult to compare this FFT processor with other recent FFT processors, as the technologies used in other recent processors are 2-5 generations apart from our processor. This makes normalization of performance difficult and less accurate. As a result, explicit comparisons will be made between processor features, FFT accuracy, and clock cycles per FFT only (all unrelated to the technology used), with more a general analysis of FFTs per second, clock rate, power, and chip area. Only dedicated FFT processors built in a 0.350 μm or smaller technology will be considered. An FFT processor built in an FPGA is also analyzed for additional comparison.

Smaller technologies inevitably lead to smaller chip areas, faster clock speeds, and less power within a chip. Our processor's clock speed is 1.01 GHz, and can perform a 1024-point complex FFT in 1.29 μs, or roughly 774,000 1024-point FFTs per second. Similarly, a 4096-point FFT takes only 6.11 μs, and 164,000 transforms can be performed per second. The chip, when operating at 1.3 V and 1.01 GHz, dissipates 250 mW. The total chip area for our processor is 1.01 mm^2.

Among the group of FFT processors analyzed, our chip is the only processor that is dynamically configurable. This is important, as it allows real-time flexibility in performance between applications requiring differing configurations of FFTs without having to reset the FFT chip. Dynamic configuration was one of the primary goals of this project.

FFT processor size range was another important criteria to consider in designing the FFT processor. Some of the chips in Tables 4.2, 4.3, and 4.4 have the capability to perform 8- and 8192-point FFTs, which our chip does not, but none of the chips has the capability to perform the range of sizes ours can (9 different sizes between 16- and 4096-point).

Very few of the chips listed possess a continuous flow architecture, and none with the exception of [16] perform continuous flow processing with the minimum amount of

Metric	Wang [33]	Miyamoto [34]	Lin [29]	Kuo [35]	DoubleBW [36]
CMOS Technology	0.35 μm	0.35 μm	0.35 μm	0.35 μm	0.35 μm
Clock Frequency	16 MHz	133 MHz	45 MHz, 17.8 MHz	80 MHz	128 MHz
Memory Size	8 Kword	512 word	est. 2 Kword	2 Kword	-
Complex Word Length	16 bits	36 bits	24 bits	32 bits	48 bits
Continuous Flow	No	No	No	No	-
Dynamically Configurable	-	No	No	No	-
FFT Size Range	2048, 8192	512	512-2048	64-2048	at least 1024
Time for 128-pt FFT	-	-	-	-	-
128-pt FFTs/second	-	-	-	-	-
Time for 512-pt FFT	-	23.2 μs	-	est. 9.8 μs	-
512-pt FFTs/second	-	43.1K	-	est. 102K	-
Cycles for 1024-pt FFT	-	-	-	-	-
Time for 1024-pt FFT	-	-	22.5μs @ 45 MHz 57 μs @ 17.8MHz	est. 40 μs	10 μs
1024-pt FFTs/second	-	-	44.4K @ 45 MHz 17.5K @ 17.8 MHz	est. 25K	100K
Cycles for 4096-pt FFT	-	-	-	-	-
Time for 4096-pt FFT	-	-	-	-	-
4096-pt FFTs/second	-	-	-	-	-
64-pt FFT SNR	-	-	-	-	-
Power	535 mW	-	640 mW @ 45 MHz 176 mW @ 17.8 MHz	574 mW	8000 mW
Voltage	3.3 V	-	3.3 V @ 45 MHz 2.3 V @ 17.8 MHz	3.3 V	3.3 V
Chip Size	33.75 mm^2	-	21.45 mm^2	6.76 mm^2	429 mm^2

Table 4.2: FFT Processor Specifications Table, part 1. These tables compare all known FFT processors built in a technology 0.35 μm or smaller. The table metrics vary as information given for a particular chip may differ. A '-' indicates information for a particular metric was not available.

Metric	Drey [12]	Currie [14]	Lin [37]	Power FFT [38]	Jo [16]
CMOS Technology	0.25 μm	0.25 μm	0.18 μm	0.18 μm	0.18 μm
Frequency	200 MHz	100 MHz	250 MHz	128 MHz	100 MHz
Memory Size	16 Kword	-	-	-	8 Kword
Complex Word Length	32 bits	32 bits	-	64 bits	32 bits
Continuous Flow	Yes	-	Yes	-	Yes
Dynamically Configurable	-	-	No	-	No
FFT Size Range	8-1024	2-4096	128	at least 1024	64-1024
Time for 128-pt FFT	-	-	312 ns	-	-
128-pt FFTs/second	-	-	3.2 M	-	-
Time for 512-pt FFT	-	-	-	-	-
512-pt FFTs/second	-	-	-	-	-
Cycles for 1024-pt FFT	1280	-	-	-	1280
Time for 1024-pt FFT	6.4 μs	Initial: 11 μs Subsequent: est. 5.1 μs	-	10 μs	12.8 μs
1024-pt FFTs/second	156K	Initial: 91K Subsequent: 196K	-	100K	78.1K
Cycles for 4096-pt FFT	-	-	-	-	-
Time for 4096-pt FFT	-	Initial: 42 μs Subsequent: 20.5 μs	-	-	-
4096-pt FFTs/second	-	Initial: 24K Subsequent: 49K	-	-	-
64-pt FFT SNR	-	-	-	-	65 dB
Power	-	2600 mW	175 mW	1000 mW	-
Voltage	-	2.5 V	3.3 V	1.8 V	-
Chip Size	-	400 mm^2	3.1 mm^2	-	-

Table 4.3: FFT Processor Specifications Table, part 2.

Metric	Sung [15]	Wang [13]	Amphion [20]	Chao (FPGA) [39]	TM-44 [40]	This Work
CMOS Technology	0.18 μm	0.18 μm	0.18 μm	0.13 μm	0.13 μm	65 nm
Frequency	150 MHz	164 Hz-6 MHz	100 MHz	127 MHz	100 MHz	1.01 GHz
Memory Size	8 Kword	1 Kword	4 Kword	n/a	-	8 Kword
Complex Word Length	32 bits	16-32 bits	32 bits	32 bits	64 bits	32 bits
Continuous Flow	No	No	No	No	-	Yes
Dynamically Configurable	No	No	No	No	No	Yes
FFT Size Range	2048-8192	128-1024	8-1024	1024	at least 64, 1024	16-4096
Time for 128-pt FFT	-	-	-	-	-	-
128-pt FFTs/second	-	-	-	-	-	-
Time for 512-pt FFT	-	-	-	-	-	-
512-pt FFTs/second	-	-	-	-	-	-
Cycles for 1024-pt FFT	-	-	5175	est. 1280	-	1305
Time for 1024-pt FFT	-	-	51.75 μs	10.1 μs	8.04 μs	1.29 μs
1024-pt FFTs/second	-	-	19.3K	100K	119K	774K
Cycles for 4096-pt FFT	est. 10.4K	-	-	-	-	6174
Time for 4096-pt FFT	69 μs	-	-	-	-	6.11 μs
4096-pt FFTs/second	14.5K	-	-	-	-	164K
64-pt FFT SNR	-	-	-	-	-	80 dB
Power	350 mW	0.6 mW (@10KHz)	-	-	8 W	250 mW
Voltage	1.8 V	0.35 V (@10KHz)	-	-	-	1.3 V
Chip Size	38 mm^2	5.46 mm^2	-	-	-	1.01 mm^2

Table 4.4: FFT Processor Specifications Table, part 3.

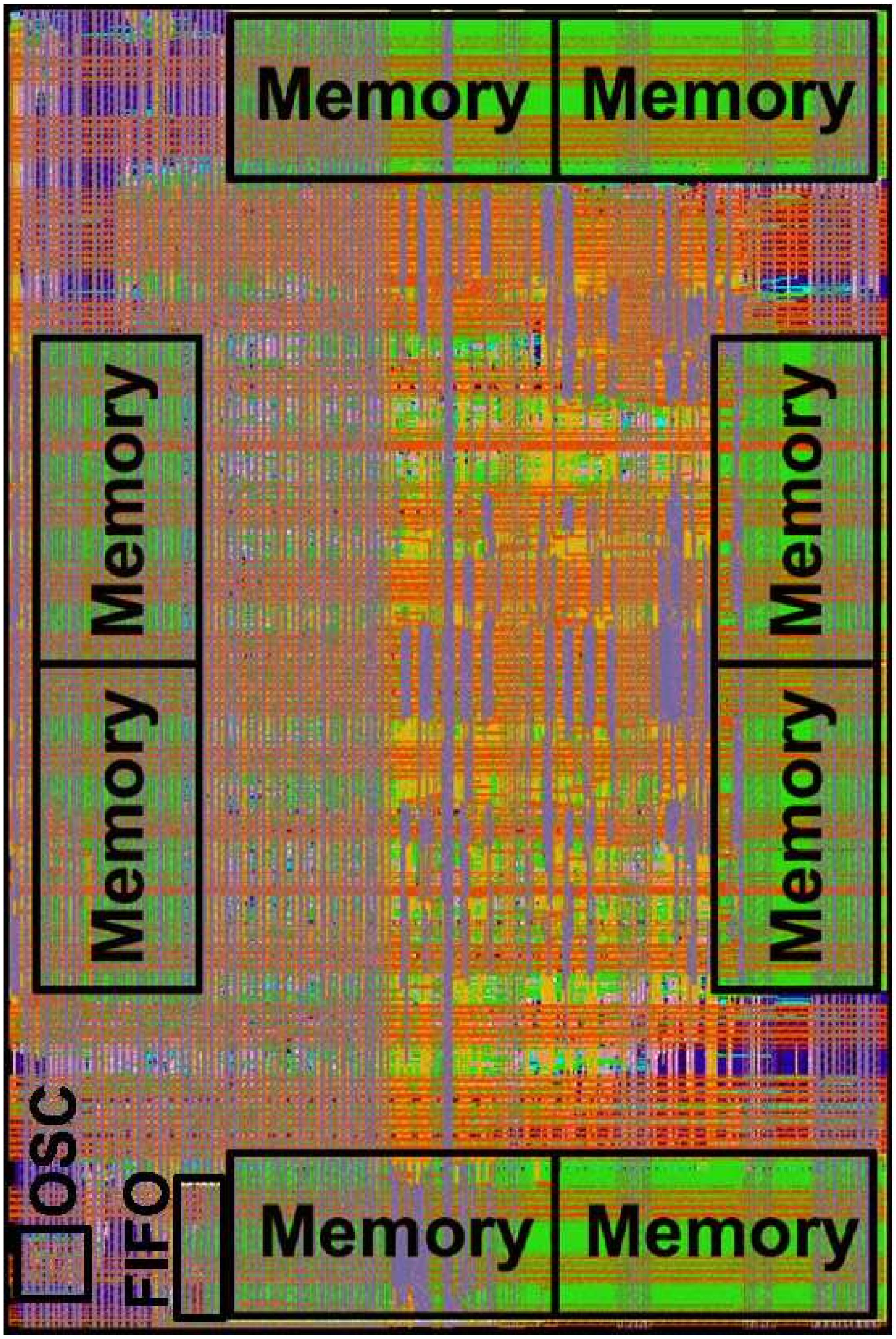

Figure 4.1: FFT processor layout. Since Encounter was used to perform layout, the only visible components in the layout are the memories (which are pre-made macros by ST), and the FIFO and oscillator (also pre-made by VCL).

FFT Size	Clock Cycles per FFT	Time per FFT (@ 1.01 GHz)	FFTs per second (@ 1.01 GHz)
16	13	12.87 ns	77.7M
32	37	36.63 ns	27.3M
64	58	57.42 ns	17.4M
128	170	168.3 ns	5.94M
256	271	268.3 ns	3.73M
512	783	775.2 ns	1.29M
1024	1305	1.289 μs	774K
2048	3609	3.573 μs	280K
4096	6174	6.113 μs	164K

Table 4.5: Performance of our FFT processor. This table breaks down the speed performance of our FFT processor at every size FFT the processor is capable of computing. The chip was run at 1.01 GHz.

memory required to do so (twice the size of the largest FFT capable of being performed).

Minimizing chip area and power were not primary considerations of this design, but due to the advanced technology used, both metrics performed very well relative to the chips built in older technologies.

The goal of being able to perform 1,000,000 1024-point FFTs per second was not met; the actual number of 1024-point FFTs capable of being performed in a second is 774,000. This is still roughly 4-5 times faster than [12], the next fastest chip. The complete breakdown of performance by FFT size is found in Table 4.5.

Chapter 5

Conclusion

This thesis introduces a dynamically configurable, continuous-flow, mixed-radix complex FFT processor. Complex FFTs and IFFTs of size 16- to 4096-point can be performed, and configuration options allow the user to turn on/off block floating point, and to divide the input data by 2 (to prevent overflow in the butterfly). This processor was built in ST's 65 nm cell library, which allows 17,400,000 64-point FFTs, 774,000 1024-point FFTs or 164,000 4096-point FFTs to be performed every second. To the best of our knowledge, this is the fastest FFT chip available academically. Therefore, this processor shows the speed potential of smaller technologies, the benefits of expanding configuration options for real-time processing, and the accuracy potential of an FFT engine.

5.1 Problems & Challenges

Several problems and challenges surfaced throughout the duration of this project. One problem in particular was the result of an improper design method that I employed. Instead of reading the literature concerning similar FFT processor design prior to designing this chip, I designed it almost exclusively around ideas that Professor Baas and I had. This oversight was crippling because I spent months developing addressing algorithms that much work had already been done on. In particular, the ideas found in Jo's work [16] could have been very useful to me had I done research in the field of FFT processor design prior to beginning my design. This typically would be crippling, but since FFT processors are a fairly new and under-utilized venture, it did not hamper my results significantly.

Another challenge that I had to overcome was the development of the addressing algorithms themselves. Knowing the desired outcome of the algorithm but not knowing how to create the algorithm was in particular challenging. I am seldom faced with a situation where the answers are known and the questions have to be invented. This style of thinking took a while to get used to, since as an engineer I am typically faced with the opposite task. Regardless, the combination of reverse-engineering and trial & error delving into the unknown equipped me with the ability to see a problem from a different perspective than I was previously able.

5.2 Future Work

In the future, it would be ideal if the source for this processor was opened to promote the advancement of this FFT processor in particular and FFT processors in general.

Additionally, there are several aspects of this processor that could be improved. The complex multiplier used in the mixed-radix butterfly could be enhanced to be more accurate and faster, as the data path was automatically generated by the synthesizer. The datapath between the memory banks and the I/O and processing element of the chip could also be designed to be more efficient, as the critical path of our chip was here. For the radix-2 butterfly processing, the radix-4 butterfly could have been modified to perform two radix-2 butterflies simultaneously, which would have significantly improved the performance of FFTs with a radix-2 stage at the end. The complex word length could be expanded to allow for greater SNR accuracy throughout the chip.

Finally, the configurability capability of the chip can be enhanced in a number of ways. FFTs smaller than 8-point and greater than 4096-point could easily be computed due to the scalability of the processor. Static configuration could be included for jobs where real-time configuration fluctuation is not needed. The I/O interface of the processor could be expanded upon to allow for a greater data throughput rate (more than one 16-bit piece of data accepted from upstream module and sent to downstream module simultaneously).

Bibliography

[1] A. Doufexi, S. Armour, M. Butler, A. Nix, D. Bell, and J. McGeehan. A comparison of the HIPERLAN/2 and IEEE 802.11a wireless LAN standards. *IEEE Communications Magazine*, 40(5):172–180, May 2002.

[2] Ming-Ju Ho, Jing Wang, Kevin Shelby, and Herman Haisch. IEEE 802.11g OFDM WLAN throughput performance. *Vehicular Technology Conference*, 4(4):2252–2256, October 2003.

[3] M.K.A. Aziz, P.N. Fletcher, and A.R. Nix. Performance analysis of IEEE 802.11n solutions combining MIMO architectures with iterative decoding and sub-optimal ML detection via MMSE and zero forcing GIS solutions. *IEEE Wireless Communications and Networking Conference, 2004*, 3:1451–1456, March 2004.

[4] Arunabha Ghosh and David R. Wolter. Broadband wireless access with WiMax/802.16: current performance benchmarks and future potential. *IEEE Communications Magazine*, 43(2):129–136, February 2005.

[5] Bernard Le Floch, Roselyne Halbert-Lassalle, and Damien Castelain. Digital sound broadcasting to mobile receivers. *IEEE Transactions on Consumer Electronics*, 35(3):493–503, August 1989.

[6] U. Reimers. DVB-T: the COFDM-based system for terrestrial television. *Electronics and Communications Engineering Journal*, 9(1):28–32, February 1997.

[7] John A.C. Bingham. *ADSL, VDSL and Multicarrier Modulation.* John Wiley and Sons, Hoboken, NJ, 2000.

[8] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. Baas. An asynchronous array of simple processors for DSP applications. In *IEEE International Solid-State Circuits Conference*, pages 428–429, February 2006.

[9] Bevan Baas, Zhiyi Yu, Michael Meeuwsen, Omar Sattari, Ryan Apperson, Eric Work, Jeremy Webb, Michael Lai, Daniel Gurman, Chi Chen, Jason Cheung, and Tinoosh Mohsenin. Hardware and applications of AsAP: An asynchronous array of simple processors. In *Proceedings of the IEEE HotChips Symposium on High-Performance Chips (HotChips 2006)*, 2006.

[10] S.S. Magar, S. Shen, G. Luikuo, M. Fleming, and R. Aguilar. An application specific DSP chip set for 100 MHz data rates. *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, pages 1989–1993, April 1988.

[11] P. A. Ruetz and M. M. Cai. A real time FFT chip set: architectural issues. *10th International Conference on Pattern Recognition, 1990*, 2:385–388, June 1990.

[12] Drey enterprise inc., Jaguar II variable-point (8-1024) FFT/IFFT specification. 1998.

[13] A. Wang and A. Chandrakasan. A 180-mv subthreshold FFT processor using subthreshold using a minimum energy design methodology. *IEEE Journal of Solid-State Circuits*, 40(1):310–319, January 2005.

[14] S. M. Currie, B. K. Gilbert, B. A. Randall, P. R. Schumacher, and E. E. Swartzlander. Implementation of a single chip, pipelined, complex, one-dimensional fast fourier transform in 0.25 um bulk cmos. *ASAP*, 2002.

[15] T.-Y. Sung. Memory-efficient and high-speed split-radix FFT/IFFT processor based on pipelined CORDIC rotations. *IEE Proceedings on Vision, Image and Signal Processing*, 153(4):405–410, August 2006.

[16] Byung G. Jo and Myung H. Sunwoo. New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(5):911–919, May 2005.

[17] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[18] Alan V. Oppenheim and Alan S. Willsky. *Signals and Systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[19] B. Baas. *An approach to low-power, high-performance, fast fourier transform processor design*. PhD thesis, Stanford University, Stanford, CA, USA, 1999.

[20] Amphion, CS2410 8-1024 point FFT/IFFT. July 2001.

[21] Jae H. Baek, Byung S. Son, Byung G. Jo, Myung H. Sunwoo, and Seung K. Oh. A continuous flow mixed-radix FFT architecture with an in-place algorithm. *International Symposium on Circuits and Systems 2003.*, 2:133–136, May 2003.

[22] Shousheng He and Mats Torkelson. Designing pipeline FFT processor for OFDM (de)modulation. *International Symposium on Signals, Systems, and Electronics.*, pages 257–262, October 1998.

[23] Wen-Chang Yeh and Chein-Wei Jen. High-speed and low-power split-radix FFT. *IEEE Transactions on Signal Processing*, 51(3):864–874, March 2003.

[24] L.G. Johnson. Conflict free memory addressing for dedicated FFT hardware. *IEEE Transactions on Circuits and Systems*, 39(5):312–316, May 1992.

[25] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn. A fast signle-chipimplementation of 8192 complex point FFT. *IEEE Journal of Solid-State Circuits*, 30(3):300–305, March 1995.

[26] Bevan M. Baas. A low-power, high-performance 1024-point FFT processor. *IEEE Journal of Solid-State Circuits*, 34(3):380–387, March 1999.

[27] R. Radhouane, P. Liu, and C. Modlin. Minimizing the memory requirement for continuous flow FFT implementation: continuous flow mixed mode FFT (CFMM-FFT). *IEEE International Symposium on Circuits and Systems*, 1:116–119, May 2000.

[28] Yun-Nan Chang and Keshab K. Parhi. An efficient pipelined FFT architecture. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 50(6):322–325, June 2003.

[29] Y.-T. Lin, P.-Y. Tsai, and T.-D. Chiueh. Low-power variable-length fast Fourier transform processor. *IEE Proceedings on Computer and Digital Techniques*, 152(4):499–506, July 2005.

[30] R. Apperson. A dual-clock FIFO for the reliable transfer of high-throughput data between unrelated clock domains. *M.S. thesis, UC Davis*, 2004.

[31] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice-Hall, Inc., Upper Saddle River, NJ, 1989.

[32] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[33] Chua-Chin Wang, Jian-Ming Huang, and Hsian-Chang Cheng. A 2k/8k mode small-area FFT processor for OFDM demodulation of DVB-T receivers. *IEEE Transactions on Consumer Electronics*, 51(1):28–32, February 2005.

[34] Naoto Miyamoto, L. Karnan, Kazuyuki Maruo, Koji Kotani, and Tadahiro Ohmi. A small-area high performance 512-point 2-dimensional FFT single-chip processor. *Proceedings of the 29th European Solid-State Circuits Conference, 2003.*, pages 603–606, September 2003.

[35] Jen-Chih Kuo, Ching-Hua Wen, and An-Yeu (Andy) Wu. Implementation of a programmable 64- to 2048-point FFT/IFFT processor for ofdm-based communication systems. *IEEE International Symposium on Circuits and Systems, 2003*, 2:121–124, May 2003.

[36] FFT processor chip information page, http://nova.stanford.edu/~bbaas/fftinfo.html. 2000.

[37] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee. A 1-GS/s FFT/IFFT processor for UWB applications. *IEEE Journal of Solid-State Circuits*, 40(8):1726–1735, August 2005.

[38] Eonic bv, Power FFT. 2002.

[39] Chu Chao, Zhang Qin, Xie Yingke, and Han Chengde. Design of a high performance FFT processor based on FPGA. In *Asia and South Pacific Design Automation Conference*, pages 920–923, January 2005.

[40] Texas memory systems, TM-44 DSP processor. 2001.

[41] Shung-Chih Chen, Chao-Tang Yu, Chia-Lian Tsai, and Jing-Jou Tang. A new IFFT/FFT hardware implementation structure for OFDM applications. In *IEEE Asia Pacific Conference on Circuits and Systems*, pages 1093–1096, December 2004.

[42] Chung-Ping Hung, Sau-Gee Chen, and Kun-Lung Chen. Design of an efficient variable-length FFT processor. In *International Symposium on Circuits and Systems*, pages 833–836, May 2004.

[43] P. Duhamel and H. Hollmann. Split radix FFT algorithm. *Electronics Letters*, 20(1):14–16, January 1984.

[44] Mark A. Richards. On hardware implementation of the split-radix FFT. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(10):1575–1581, October 1988.