Design and Evaluation
of a Single Instruction Processor

Master Thesis
Division of Electronics Systems
Department of Electrical Engineering
Linköping Institute of Technology
Linköping University, Sweden

By
Rongzeng Mu

Reg nr: LiTH-ISY-EX-3502-2003

Supervisor:
Thomas Johansson

Examiner:
Lars Wanhammar

2003-09-24
Title
Design och utveckling av en eninstruktions processor
Design and Evaluation of a Single Instruction Processor
Author
Rongzeng Mu

Abstract
A new path of DSP processor design is described in this thesis with an example, to design a FFT processor. It is an innovative concept for DSP processor design developed by the Electronic Systems Division in the department of Electrical Engineer department in Linköping University.

The project described in this thesis is to design a Sande-Tukey FFT processor step by step. It will go through all steps from the simplest MATLAB specification to the final synthesizable VHDL specification. The steps should be as small as possible in order to avoid error and MATLAB should be used as as possible.

Keyword
FFT, processor, specification, algorithm, input, compute, output
Abstract

A new path of DSP processor design is described in this thesis with an example, to design a FFT processor. It is an innovative concept for DSP processor design developed by the Electronic Systems Division in the department of Electrical Engineer department in Linköping University.

The project described in this thesis is to design a Sande-Tukey FFT processor step by step. It will go through all steps from the simplest MATLAB specification to the final synthesizable VHDL specification. The steps should be as small as possible in order to avoid error and MATLAB should be used as far as possible.

Keywords: FFT, processor, specification, algorithm, input, compute, output
Acknowledgements

I should like to thank all of the people who have contributed to the creation thesis.

The first people I should thank is Lars Wanhammar, a professor at the Electronics Systems Division in the Electrical Engineering Department at Linköping University, who gave me the opportunity to do the project. His book helps me complete the design work.

I should like to thank my supervisor Thomas Johansson for giving me many valuable suggestions and comments.

Finally my family who made significant sacrifices so that I could study abroad.
Table of contents

Chapter 1 Introduction ........................................................................................................ 3
  1.1 Background ........................................................................................................... 3
  1.2 Overview .............................................................................................................. 3
  1.3 Design Tools ......................................................................................................... 5
  1.4 Reading Guideline ............................................................................................... 6
  1.5 Abbreviations ....................................................................................................... 6
Chapter 2 Basic Model ..................................................................................................... 7
  2.1 Algorithm ............................................................................................................. 7
    2.11 Discrete Fourier Transform (DFT) .................................................................. 7
    2.12 Fast Fourier Transform (FFT) ......................................................................... 8
  2.2 Design & Result .................................................................................................. 10
Chapter 3 First Partition ............................................................................................... 12
  3.1 Specification ....................................................................................................... 12
  3.2 Design & Result .................................................................................................. 13
Chapter 4 Second Partition ............................................................................................ 16
  4.1 Specification ....................................................................................................... 16
  4.2 Design & Result .................................................................................................. 16
Chapter 5
Transform from MATLAB to VHDL .......................................................................... 19
  5.1 Specification ....................................................................................................... 19
  5.2 Design & Result .................................................................................................. 20
Chapter 6
VHDL Model Using 2 RAMs ..................................................................................... 24
  6.1 Specification ....................................................................................................... 24
  6.2 Design & Result .................................................................................................. 26
Chapter 7 Pipelining .................................................................................................... 29
  7.1 Specification ....................................................................................................... 29
  7.2 Design & Result .................................................................................................. 30
    7.2.1 Input Part ..................................................................................................... 31
    7.2.2 Compute Part .............................................................................................. 31
    7.2.4 Output Part ................................................................................................ 33
    7.2.5 Stage_ctrl ................................................................................................... 34
    7.2.6 Result .......................................................................................................... 35
Chapter 8 Final Architecture .......................................................................................... 36
8.1 Specification ........................................................................................................ 36
8.2 Design & Result .................................................................................................... 37
Chapter 9 Test Bench .............................................................................................. 40
  9.1 Test in MATLAB ............................................................................................ 40
  9.2 Test in VHDL .................................................................................................. 40
Chapter 10 Conclusion .......................................................................................... 42
References ............................................................................................................ 43
Chapter 1 Introduction

1.1 Background

The ASIC processors are applied in more and more applications along with the booming development of the semiconductor technology now. The increase of the ASIC processor will continue in the future. The time to market is very critical for the commercial success of a product. There are various approaches to designing a processor. The development time is different. There is a developed concept for the design of an ASIC processor, which can simplify and clear the design path down to hardware, at Electronic Systems Division in the department of Electrical Engineering department at Linköping University. This approach is to start from a MATLAB description down to a synthesizable VHDL description and partition the processor to subsets as far as possible in MATLAB model.

The evaluation of this concept is necessary by designing an ASIC processor using it. The work in this thesis is to evaluate it by designing a specific processor, FFT processor. It went through the whole design path from a MATLAB description down to a synthesizable VHDL description and sought the better break point where the transfer from MATLAB model to VHDL model should be done.

1.2 Overview

The project in this thesis is to design an FFT processor. All design steps is illustrated in Figure 1-1. The design flow goes through seven steps. Three of them are MATLAB models, the others VHDL models. All models described in MATLAB use 1 memory. We describe seven steps briefly in the following.

1. Basic FFT Model, the input of this step is the Sande_Tukey FFT algorithm. The output is the basic FFT model. The purpose of this step only is to verify the Sande_Tukey FFT algorithm.
2. First Partition, the input of this step is Basic FFT Model. In this step the basic
FFT model is divided into 3 parts, Input, Compute, Output. Only one butterfly is used in the output of this step, then we call the output model of this step is 1-Butterfly Model.

3. Second Partition, the input of this step is 1-Butterfly Model. In this step the compute part of the FFT model is divided continuously. Two butterflies is used in the output of this step, so that we call the output model of this step is 2-Butterfly Model.

4. Transfer from MATLAB to VHDL, the input of this step is 2-Butterfly Model. In this step the FFT process is transferred to VHDL. Because a memory is used in the output model of this step, the output model of this step is named as 1- Memory Model.

5. VHDL Model using two RAMs, the input of this step is 1- Memory. In this step another RAM is added to the FFT processor. Because there are two memories in the output model of this step, the output model of this step is named as 2- Memory Model.

6. Pipelining, the input of this step is 2- Memory. In this step all three parts of the processor are pipelined. The output model of this step is called as Pipelined Model.

7. Final Architecture, the input of this step is Pipelined Model. In this step all components of three parts of the processor are re-ranged and combined. The output model of this step is called as Final Architecture Model.
1.3 Design Tools

MATLAB, C language, Mentor Graphics are used in this project. MATLAB is used to construct the former three MATLAB models. It has an FFT function, which is the golden model used to verify the basic model. The HDL designer and the
simulator Modelsim in Mentor Graphics is used when design the last four VHDL models. The C language is used to transfer the data used in MATLAB models into binary data used in the VHDL model, compare the output data file of 1- Memory Model with that of 2- Butterfly Model and verify all outputs in the VHDL models with the outputs of the preceding model.

1.4 Reading Guideline

This thesis consists of ten chapters. Chapter 1 is Introduction. Chapter 2 to 8 will describe the seven steps in the design path in detail. In every chapter of these seven chapters how to design, why the architecture is chosen will be presented. Chapter 9 will describe how to verify the 7 models in both MATLAB and VHDL models. Chapter 10 is the conclusion of the thesis. The result of the evaluation and what is important when using this approach will be discussed in this chapter.

1.5 Abbreviations

FFT  Fast Fourier Transform
DFT  Discrete Fourier Transform
VHDL Very High Speed Integrated Circuit Hardware Description Language
ASIC Application Specific Integrated Circuit
RAM  Random Access Memory
ROM  Read-Only Memory
DUT  Design Unit for Test
Chapter 2 Basic Model

2.1 Algorithm

2.11 Discrete Fourier Transform (DFT)

The Fourier Transform is defined as
\[ f(v) = F_t[f(t)](v) = \int_{-\infty}^{\infty} f(t) e^{-2\pi ivt} dt. \]

Now consider generalization to the case of a discrete function, \( f(t) \to f(t_k) \) by letting \( f_k \equiv f(t_k) \), where \( t_k \equiv k\Delta \), with \( k = 0, \ldots, N - 1 \). Choose the frequency step such that \( \Delta = \frac{\pi}{N} \) with \( n = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2} \). There are \( N + 1 \) values of \( n \), so there is one relationship between the frequency components. Writing this out as
\[
F^N_t[f(t)](n) = \sum_{k=0}^{N-1} f_k e^{-2\pi i(n/N)k\Delta} \\
= \Delta \sum_{k=0}^{N-1} f_k e^{-2\pi i nk / N},
\]
so write
\[
F_n = \sum_{k=0}^{N-1} f_k e^{-2\pi i nk / N}
\]

The inverse transform is then
\[ f_k = \frac{1}{N} \sum_{n=-N/2}^{N/2} F_n e^{2\pi i n k / N} \]

Note that \( F_{-n} = F_{N-n} \), \( n = 1, 2, \ldots \), so an alternate formulation is

\[ f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{2\pi i n k / N} \]

The discrete Fourier transform can be computed using a fast Fourier transform.

### 2.12 Fast Fourier Transform (FFT)

The fast Fourier transform (FFT) is a discrete Fourier transform algorithm which reduces the number of computations needed for \( N \) points from \( 2N^2 \) to \( 2N \log_2 N \), where \( \log \) is the base-2 logarithm.

Fast Fourier transform algorithms generally fall into two classes: decimation in time, and decimation in frequency. The Cooley-Tukey FFT algorithm first rearranges the input elements in bit-reversed order, then, builds the output transform (decimation in time). The Sande-Tukey algorithm first transforms, then rearranges the output values (decimation in frequency).

In this project the Sande-Tukey algorithm is used. The flow chart is illustrated as Figure 2.1. In the project \( N \) is fixed as 1024, so that \( M \) equals to 10. Unscramble is rearranging the output value.
Figure 2.1 The Sande-Tukey FFT flow chart

Figure 2.2 shows the signal flow graph for ST-FFT for N = 4. It is clearly shown
that there are $\lg(4) = 2$ stages and $2^4 \cdot \lg(4) = 16$ computations in the signal flow graph.

The Sande-Tukey butterfly can be derived from the signal flow graph shown in Figure 2.2. It is shown in Figure 2.3.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{sande-tukey_butterfly}
\caption{Sande-Tukey FFT for $N=4$}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{sande-tukey_butterfly_2}
\caption{The Sande-tukey butterfly}
\end{figure}

\section*{2.2 Design & Result}

In this step the only work is to design the basic model to verify the Sande-Tukey FFT algorithm. The MATLAB program (fft_version_1_1) is written according the flow chart illustrated in Figure 2.1. When the program is run, the input is a
complex array with 1024 length generated randomly. The output is compared with
the output of the fft function of MATLAB, the maximum error is less than 10e-13.
It is reasonable for MATLAB. The approach of the verification will be described in
Charter 9.
Chapter 3 First Partition

3.1 Specification

The first partition will be described in detail in this chapter. In this step the FFT processor is divided into 3 parts, which is Input, Compute and Output. There is no memory part because I use an array as a memory. Only 1 butterfly is used in this model. The critical work in this step is to combine the loop2 and the loop3 into a new loop, loop2&3 to simple the architecture of the processor.

As shown in Figure 2.1,

\[ 1 \leq q \leq \frac{N}{2*Ns} \]
\[ 1 \leq n \leq Ns \]

In this project, \( N = 1024 \), \( Ns = \frac{N}{2^{stage}} \), so that

\[ 1 \leq q \leq 2^{\text{stage}-1} \]
\[ 1 \leq n \leq \frac{N}{2^{\text{stage}}} \]

It is easy to derive

\[ 1 \leq q*n \leq \frac{N}{2} = 512 \]

512 is the new number of circles in the now loop. It is a constant. It is good news because the loop is easier to be synthesized in hardware than when the iteration number is variable. We introduce a new integer variable \( m \),

\[ 0 \leq m \leq \frac{N}{2} - 1 = 511 \]

In loop2 and loop3 there are 2 variables, \( p \) and \( k \). \( p \) can be derived from \( k \), then the only thing we need to do is to derive \( k \) from \( m \). At first in the old model,

\[ k = 2Ns(q-1) + (n-1) = 2Ns + q' + n', \]

where two new integers, \( q' \) and \( n' \), are introduced. We can regard the 3
invariable \( m, q' \) and \( n' \) as Figure 3.1.

\[
\begin{array}{c|c|c|c|c|c|c|c}
\multicolumn{2}{c}{} & m & q' & n' & Ns & 2Ns+1 & \ldots & \ldots \\
\hline
m & 0, 1 \ldots & Ns - 1, & Ns & Ns + 1 & 2Ns - 1, & 2Ns, & 2Ns + 1 & \ldots & 2^{\text{stage}} - 1, & 2^{\text{stage}} Ns \\
q' & 0, 1, 2, \ldots, & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

Figure 3.1 \( m, q' \) and \( n' \)

So that, \( m = q'Ns + n' \). Now can derive,

\[
q' = \left\lfloor \frac{m}{Ns} \right\rfloor \\
n' = m \mod (Ns) \\
k = 2Ns \left\lfloor \frac{m}{Ns} \right\rfloor + m \mod (Ns)
\]

Thus, the loop2 and loop3 are replaced by the new loop named loop2&3, which is only controlled by the integer \( m \).

### 3.2 Design & Result

The processor does the input, compute, output works serially. At first the processor is divided into 3 parts as Figure 3.2. Because MATLAB is a mathematic tool, it is not able to describe the components as hardware. I use the 1*1024 array Memory as the memory of the processor. The Input is separated from the Basic Model. Its
responsibility is only to copy the input array to Memory. The Compute does not change in this sub step. The Output part includes digital reversal and output. This is the program fft_version_2_1.

For stage in 1 to 10
  loop 1
    k = 0
    Ns = Ns/2
    For q in 1 to 2^{(stage-1)}
      loop2
        For n in 1 to Ns
          loop3
            Twiddle_Factor
            butterfly
            Next n
            k = k + 1
          End loop3
          k = k + Ns
        End loop2
      Next q
    Next stage
  End loop1
  Output

Figure 3.3 The Compute part of the FFT processor(3 loops)

The second, the twiddle factor WP generator and Butterfly are divided as components in the Compute part, because these two parts are complex in hardware. The program architecture is showing in Figure 3.3. The program is fft_version_2_2.
At last, the 2_loop architecture will be generated. How to combine the loop2 and loop3 is described in the specification above. The architecture is shown in Figure 3.4. The program of the FFT processor at this sub step is fft_version_2_3.

The maximum error of all versions of the processor in this step is less than 10e-13 when the outputs are compared with that of the MATLAB FFT function.

![Diagram](image)

Figure 3.4 The Compute part of the FFT processor(1-butterfly model)
Chapter 4 Second Partition

4.1 Specification

We continue to partition the FFT processor to the model that uses 2 butterflies. Since in the compute part 2 butterflies are performed concurrently, the range of \( m \) is reduced to \( 0 \leq m < N/4 = 256 \). In addition we have to generate 2 indexes \( k \) in parallel: \( k_1 \) and \( k_2 \). There are 2 different alternative approaches to generate \( k_1 \) and \( k_2 \). An approach is that

\[
\begin{align*}
   k_1 &= 2Ns \left\lfloor \frac{m}{Ns} \right\rfloor + m \mod(Ns) \\
   k_2 &= \begin{cases} 
   k_1 + N/4 & \text{stage} = 1 \\
   k_1 + N/2 & \text{stage} \geq 2 
   \end{cases}
\end{align*}
\]

The other is that

\[
\begin{align*}
   k_1 &= 4Ns \left\lfloor \frac{m}{Ns} \right\rfloor + m \mod(Ns) \\
   k_2 &= \begin{cases} 
   k_1 + Ns/2 & \text{stage} = 1 \\
   k_1 + 2Ns & \text{stage} \geq 2 
   \end{cases}
\end{align*}
\]

It is described in reference (1) how to derive these two methods. As a reason to be chosen, in both alternatives the 2 concurrent butterflies will use coefficients that either are identical or can easily be computed from each other. Because when I was working on the project in this step, I had not chosen which index generator, I constructed 2 architectures for both of alternatives. The only difference between them is in their address (indexes) generator parts.

4.2 Design & Result

In the program fft_version_3_1_1 and the program fft_version_3_1_2, the architectures that use 2 butterflies are constructed. The fft_version_3_1_1 uses the first index generator, while the fft_version_3_1_2 use the second. Because index generator is a complex part in hardware, it is separated from the Compute part as the address generator. The refined models are the program fft_version_3_2_1 and the program fft_version_3_2_2.
Because it is impossible that functions run concurrently in a single program, we can only divide the processor into components as the architecture in the hardware. The final architecture of the Second Partition is shown in Figure 4.1. In the architecture, the ‘addresses’ generates the k1, kNs1, k2, kNs2, the Twiddle_Factor generates the coefficients for the two butterflies, the upper butterfly uses k1 and kNs1 as the memory addresses, the nether one uses k2 and kNs2.

Figure 4.1 The final architecture of the FFT in 2-butterfly model
When the outputs are compared with that of MATLAB FFT function, the maximum error of all versions of the processor is less than $10^{-13}$ in this step.
Chapter 5

Transform from MATLAB to VHDL

5.1 Specification

In this step the FFT processor will be transferred from MATLAB model to VHDL model. The Mentor Graphics is used from this model. Because there is no difference between the two final versions of the 2-butterfly Model except that of addresses parts, the fft_version_3_2_2 is used as the input of this step.

Figure 5.1 the architecture of the 1-memory model
The output of this step is a hardware description, so that the architecture of the processor will be changed somewhere. At first a RAM is added to replace the array Memory in the 2-butterfly model. Then a MUX is designed to switch the signals between Input, Compute, Output and RAM. The architecture is shown in Figure 5.1.

5.2 Design & Result

In the processor the Input, Compute and Output perform sequentially. It is the same as that in MATLAB model. However, how to transfer between each other of them? It is a new problem for the design work. There are 2 approaches to resolve to the problem. One of them is that they generate the state signal (input_s, compute_s and output_s) by themselves as shown in Figure 5.2, the other one use a stage generator to generate 3 states according to the periods the 3 parts perform. The most different point between of the two approaches is that the constant periods are necessary in the later approach, while not in the former one. Because of adaptive periods the approach that three parts generate state signals by themselves is chosen.

![Figure 5.2 The state transform flow chart](image)

When the signal “input_available” equals ‘1’, “input_s” equal ‘1’. After the Input receives 1024 data, “input_s” turns to ‘0’. The signal “compute_s” turns to ‘1’ at the falling edge of the “input_s” and turn to ‘0’ after Compute finishes all the computations. The signal “output_s” turns to ‘1’ at the falling edge of the “compute_s” and turn to ‘0’ after Output outputs all results in the memory. At the
falling edge of the output, the “inv_input_e” is signed to ‘0’. It turns to ‘1’, after the Input receives 1024 data.

The main function of the Input is to generate the address and transfer the input data from 32-bit words (16 bits are the real part, the others are the image part) to 48-bit words, because the word length is 48 bits (24 bits are the real part, the others are the image part) in order to reduce the quantification error. Addresses are the indexes of the input data.

The Compute serially performs two butterfly computations in a period of 9 clock units. Its computation flow chart is show as Figure 5.3. The k1, kNs1, k2 and kNs2 are addresses. The “op1” and “op2” is the operands, while the “r1” and “r2” are results.

![Figure 5.3 schedule of the compute part for 1 memory part.](image)

From the schedule shown in Figure 5.3, the 2 butterflies in the processor do not perform concurrently. Because it is not final model of the project, the additional work that reschedule to the concurrent time plan is not necessary.

Another import problem is how to generate twiddle factors. It is impossible to use a cosine processor in the FFT processor, because it would waste both hardware and time resources. We use a ROM to save computed factors that are computed by MATLAB. We know
so that a ROM with 512 length is enough to save the factors. Maybe the size of the ROM can be smaller. We do not discuss that here. We set the word length of the twiddle factor 42 bits, 21 bits for real, 21 bits for image. We use a constant array of 42*512 represent the twiddle factor ROM.

The Output is responsible to generate the address and transfer the input data from 48-bit words to 32-bit words. Addresses are digit reversal of the indexes of the Output.

The MUX is to switch address signals and data signals and to generate the r_w signal for the RAM.

Because we use a large number of adders and multipliers, the scaling is necessary for Compute part to avoid the saturation. Safe scaling is used in the processor. In the input part we divide the input data by 2 before they are written to the RAM. In the Output part we multiply data by 2 before they are sent to output port. In the Compute part we divide results of the butterfly computations by 2. We can combine this computation into the butterfly. Then, the new butterfly is illustrated in Figure 5.4. We modify the butterfly by save $W_p/2$ instead $W_p$ in the twiddle factor ROM. After scaling the output is reduced to 1/1024 of the output before scaling.

\[ p = (k1*(2^{stage-1}))) \mod(N/2), \]

Figure 5.4 The scaling butterfly
So far the 1-memory model has been constructed. The FFT_PROCESSOR is the version in this model. When the output of the FFT_PROCESSOR is compared to that of fft_version_3_2_2, the quantization error is “0000,0000,0000,1111” which is equal to 4.783155e-4. We will explain why it is reasonable in Chapter 9.
Chapter 6

VHDL Model Using 2 RAMs

6.1 Specification

Figure 6.1 The architecture of 2-memory model
This step is mainly to divide the memory to 2 RAMs. Its architecture is shown in Figure 6.1.

The main work in this step is to design the addressing components of all 3 parts, Input, Compute and Output. The Compute determines the memory assignment and resource assignment. Because the Compute performs in_place computation, its memory assignment is the base of the addressing of the Input and Output. In this step we also have to decide how to connect butterflies to RAMs to avoid that an operand of a butterfly is fetched from 2 RAMs. It can avoid switches in hardware.

![Figure 6.2](image)

It seems simple to assign the first $N/2$ variables ($0 \leq i < N/2$) to a memory RAM0 and the other $N/2$ variables ($N/2 \leq i < N$) to another memory RAM1. However, it will store the two input and output values to a butterfly in a single memory. There is a feasible memory assignment in reference (1), which is always to store the two results of the butterfly in different memories. It is derived with the exclusion graph. The detail derivation is shown in Chapter 7 of the reference (1). We only describe the approach here.

A variable with index $i$ is assigned to $\text{RAM}_{P(i)}$ where
\[ P(i) = i_0 \otimes i_1 \otimes i_2 \otimes \cdots \otimes i_9. \]

Memory assignments of various FFTs are shown in Figure 6.2.

The butterfly assignment is derived by the same way to derive the memory assignment. As described in reference (1), “the assignment can be determined from the binary representation of the row number. A butterfly operation in row \( r \) is assigned to the PE \( P(r) \) where

\[ P(r) = r_0 \otimes r_1 \otimes r_2 \otimes \cdots \otimes r_L \]

Where \( L = \log_2 (N/2) - 1 \).”

6.2 Design & Result

In the Input part the main new function is to assign the memory according the indexes. According to the formula \( P(i) = i_0 \otimes i_1 \otimes i_2 \otimes \cdots \otimes i_9 \), the value with the index “0xxxxxxxxx” and that with the index “1xxxxxxxxx” must be assigned different memories, so that we can use “xxxxxxxxx” as the address. The \( \text{ram}_\text{sel} \) is determined by \( P(i) \).

In the Compute part, we use the addressing algorithm described in Chapter 9 of reference (1). The variable \( m \) is the index.

When \( P(m) = 0 \),

\[ k_1 \text{ from } \text{RAM}_0; \text{ address } k_1 \bmod(N/2), \text{ to } \text{PE}_0 \]
\[ k_1 Ns \text{ from } \text{RAM}_1; \text{ address } k_1 Ns \bmod(N/2), \text{ to } \text{PE}_0 \]
\[ k_2 \text{ from } \text{RAM}_1; \text{ address } k_2 \bmod(N/2), \text{ to } \text{PE}_1 \]
\[ k_2 Ns \text{ from } \text{RAM}_0; \text{ address } k_2 Ns \bmod(N/2), \text{ to } \text{PE}_1 \]
When \( P(m) = 1 \),

\[
k_1 \text{ from RAM}_i; \text{ address } k_1 \mod(N/2), \text{ to PE}_i
\]

\[
k_{1Ns} \text{ from RAM}_0; \text{ address } k_{1Ns} \mod(N/2), \text{ to PE}_i
\]

\[
k_2 \text{ from RAM}_0; \text{ address } k_2 \mod(N/2), \text{ to PE}_0
\]

\[
k_{2Ns} \text{ from RAM}_i; \text{ address } k_{2Ns} \mod(N/2), \text{ to PE}_0
\]

In this algorithm the \( m \) need not to be incremented in binary order. We let \( m \) be incremented in Gray Code order to avoid computing the function \( P(m) \) for every \( m \), because Gray Code has the property that only one bit changes once. We generate the index \( m \) by encoding \( i \) generated by a binary counter to Gray code. Then the function \( P(m) \) is equal to the least significant bit of the \( i \), \( i_0 \).

According the algorithm for any PE, \( k_1 \) and \( k_2 \) are from an identical memory, while \( k_{1Ns} \) and \( k_{2Ns} \) are from another. It means that there is no switch between the RAMs and butterflies. It satisfies the demand in the specification.

The Compute performs two butterfly computations in a period of 7 clock units. Its computation flow chart is show as Figure 6.3. The addr0 is the function to compute addresses in the RAM0, A00 and A10, while the addr1 is the function to compute addresses in the RAM1, A10 and A11. The “op00” and “op01” is the operands of the butterfly0, while the “r00” and “r01” are results. The “op10” and “op11” is the operands of the butterfly1, while the “r10” and “r11” are results. The addr0 is the address signal for RAM0, the addr1 is the address signal for RAM1.
In this step, the Output part has the same problem as that the Input part has. The ram_sel signal is determined by

\[ P(\text{digital\_reversal}(i)) \].

The address is

\[ \text{digital\_reversal}(i) \mod(N/2) \].

The MUX here is similar to that of last model except that it is connected to two RAMs. The size of RAM reduces to 1/2 of 1-memory model.

These are the main works in this step. The FFT_PROCESSOR\_ver\_2\_1 is the VHDL description in this model. The output of the FFT_PROCESSOR\_ver\_2\_1 is the same as that of FFT_PROCESSOR, when we use the same data array as the inputs.
Chapter 7 Pipelining

7.1 Specification

Because there is a delay of some clock signals, it is difficult to control the
generation of the state signals. We use the second approach mentioned in Chapter 5 to generate the state signals. Then, we add a new component, stage_ctrl, to the FFT processor to generate the state signals, Input, Compute and Output signals. The architecture of the output of this step is shown in Figure 7.1.

In the step we will pipeline the three parts separately. It is simply to fold the schedule of the Input and Output, so that we do not discuss the pipeline algorithm here. We will focus on the Compute part.

Because of the property of the memory that it can be read or written at most once in a clock unit, the minimum period of Compute performance is 4 clock units. We can read the memory twice and write twice only in the order, read_write_read_write. Then the new schedule for pipelined Compute part in shown in Figure 7.2. The addr0' represents that the address of RAM0 and that of RAM1 are respectively set to the values addr00, and addr01 after 5 clock unit delay. The addr1' represents that the address of RAM0 and that of RAM1 are respectively set to the values addr10, and addr11 after 5 clock unit delay.

Figure 7.2 The schedule for pipelined compute part

### 7.2 Design & Result

Because the architectures of components are complex, we describe the designs of
the Input, Compute, Output and stage_ctrl separately. There is no change in other component except these four parts.

### 7.2.1 Input Part

The Input part includes three processes, index_gen, addr_gen, idata as the architecture shown in Figure 7.3. The index_gen process begins to generate indexes from 1 to 1023 as soon as the rising edge of the input_s. The addr_gen process generates the address and ram_sel signals according the index $i$ and $P(i)$, when the input_s is equal to 1. The idata process converts the input data from 16 bits to 24 bits and divides them by 2, when the input_s is equal to 1.

![Figure 7.3 The architecture of Input part of Pipelined Model](image)

### 7.2.2 Compute Part

As the architecture shown in Figure 7.4, the Compute part include 7 processes that perform according the schedule illustrated in Figure 7.2. They will be described in detail one by one.
The counter_stage_gen process generates the basic counter signal that ranges from 0 to 3 and the stage signal. Its performance begins at rising edge of the compute. Because $N_s = N / 2^{stage - 1} = 2^{10 - stage}$ and both of $N_s$ and stage signals are used in the other processes, we use $N_s$ to replace the stage to simple the architecture. Then the process counter_stage_gen generates $N_s$ instead stage.

The process m_gen generates the basic index signal $k_1$, the memory assignment signal $pm = P(m)$ and the end_compute signal which controls the beginning and end of the writing operation in the process R_W_PE. There is a gray counter that consists of a binary counter $i$ and a gray encoder to generate the $m$ in the process. The detail description in VHDL is in FFT_PROCESSOR_ver_2_2.

The address_gen process computes 4 addressing signals, addr00, addr01, addr10 and addr11. The input is $N_s$, $k_1$ and $pm$. The algorithm is same as the addressing algorithm in the 2-memory Model.

The process R_W_PE reads the operands R1, R2, R3 and R4 of the butterflies form the RAMs and writes the results W1, W2, W3 and W4 to the RAMs. It outputs the address signals, addr0 and addr1, and read/write control signals, r_w0 and r_w1. It exchanges the data with RAMs through the inout ports data0 and data1, where data0 for RAM0, data1 for RAM1.

The process WP_gen process generates the twiddle factors for bufferfly0 and butterfly1. At first, it computes the twiddle factor’s address in the ROM with data0 for $N_s$ and $k_1$. Then it reads out the twiddle factor from the ROM, which is the twiddle factor of butterfly 0. At last the twiddle factor of butterfly1 is derived from it according $N_s$ and $pm$.

The process butterfly_PE0 and butterfly_PE1 complete the computation of two parallel butterflies.
7.2.4 Output Part

As the architecture shown in Figure 7.5, the output part in this step consists of 5 processes. The index_gen is implemented as a binary counter range from 1 to 1023 and generates output control signal in addition. It begins at the rising edge of the output_s. The process D_reverse performs the digital reversal operation. The addr_gen process sends the addresses to RAMs and generates RAM select signal ram_sel. The O_data process reads the data from memories, transforms them from 24 bits to 16 bits at the same time when they are divided by 2, output them to output port. The out_control generates the out_available, which means the data on the
7.2.5 Stage_ctrl

We have described the architectures of the Input, Compute and Output elements. We begin to analysis the time property of then respectively. A whole input period goes through index_gen, addr_gen, and write_ram included in RAM component. It is pipelined into 3 separated stages. And it begins at rising edge of input_s, so that the input has to last at least \( N + 3 + 1 = 1028 \) clock units. According the schedule shown in Figure 7.2, the period of the computation of a stage is \((256 + 2 + 1) * 4 = 1036\) clock units. There are 10 stages in the Compute and the computation start at rising edge of the compute_s, hence the performance of the Compute part has to last 10361 clock units. We set the period to 10366 clock units. A single output period lasts clock units, when it performs index_gen, D_reverse, addr_gen, read_data and output. It begins at rising edge of output_s, then the Output part lasts \( N + 5 + 1 = 1030\) clock units. So that we generate the periodical state signals as shown in Figure 7.6. The period of the FFT processor is 12425 clock units.
7.2.6 Result

The FFT_PROCESSOR_ver_2_2 is the VHDL description in this model. The output of the FFT_PROCESSOR_ver_2_2 is the same as that of FFT_PROCESSOR_ver_2_1, when we use the same data array as the inputs.
Chapter 8 Final Architecture

8.1 Specification

In this step we will break the Input, Compute, Output component, rearrange the
performances and combine the parts that have same functions to reconstruct a new architecture that is simpler and more hardware-saved than that we designed before. The new architecture is shown in Figure 8.1. How to use the FFT processor? At a clock after detecting the first DO=“01”, the user begins to input the input data. After 1024 data are input to the processor, the user needs to be waiting for the next DO transform from “10” or “00” to “01”.

8.2 Design & Result

The Final Architecture Model is the improvement of the Pipelined Model. All the components are derived from the Pipelined Model.

The stage_PE is the stage_ctrl in FFT_PROCESSOR_ver_2_2 added a function to generate the stages of the computation. DO is the 2_bit state signal. “01” means Input stage, “11” is Compute stage and “10” is Output stage. Ns represent the sub stage of the Compute.

The base_index_gen combines all the index generations in FFT_PROCESSOR_ver_2_2. During the Input and Output stage, it generates the index $i$. During the Compute stage, it outputs $k_1$, PM, end_compute and counter.

The digit_reversal is the same as the D_reverse process in the Output part in previous model.

The addr_gen controls the memories in 3 stages and generates ram_sel and output_en signals in the Output stage that control the output. This element performs the function of addr_gens in all 3 parts and part of the function of R_W_PE in the Compute part.

The cache_ctrl0 and cache_ctrl1 work on transferring input data from 16-bit to 24-bit in the Input stage, converting output data from 24-bit to 16-bit in the Output stage, read operands of butterflies from and write results to RAMs. It performs the function of idata in the Input part and O_data in the Output part and partly function of R_W_PE in the Compute part in previous model.
The output_ctrl element selects which data output. The other elements do not change in this step.

The FFT_PROCESSOR_ver_3_1 is the VHDL description of the Final Architecture Model. The output of the FFT_PROCESSOR_ver_3_1 is exactly same as that of FFT_PROCESSOR_ver_2_1, when we use the same data array as the inputs.
Chapter 9 Test Bench

9.1 Test in MATLAB

There is a FFT function in MATLAB. It is used as the gold model in the Test Bench shown in Figure 9.1. The input array A is a 1*1024 complex array generated randomly. And

\[ |A| < 1. \]

![Figure 9.1 Test Bench in MATLAB Model](image)

We compute the FFT by the FFT function in MATLAB and the FFT processor that we designed separately, then compute \( Z = |X - Y| \). If \( Z < \text{tolerance} = 1e^{-13} \) is satisfied, the design unit is verified in tolerance. The tolerance is set to \( 1e^{-13} \).

When the fft_version_1_1, fft_version_2_1, fft_version_2_2, fft_version_2_3, fft_version_3_1_1, fft_version_3_1_2, fft_version_3_2_1 and fft_version_3_2_2 are tested with the Test Bench, all of them are verified in tolerance.

9.2 Test in VHDL

As show in Figure 9.2, the Test Bench in VHDL is similar to that in MATLAB. The
The difference between them is that the gold model in MATLAB is identical FFT MATLAB function and that in VHDL is the nearest previous model designed. The input is read from a text file and the output is written to a text file. All variables in 

\[ Z = |X - Y| \]

are in decimal type. Both of Outputs X and Y are complex array.

![Figure 9.2 Test Bench in VHDL Model](image)

The input array A1 here is an array of binary numbers translated from the input array \( A \) in MATLAB model. The previous model of the FFT_PROCESSOR is fft_version_3_2_2 that is a MATLAB model, and because of safe scaling in VHDL model, so that we divide the output of fft_version_3_2_2 by 1024 as the test vector.

Because in \( Z = |X - Y| \) all variables are in decimal type, we translate the output of the FFT_PROCESSOR from binary type to decimal type. The \( \text{MAX}(Z) \) is equal to 4.783155e-4. It is reasonable. The quantization error of Sande-Turkey FFT is described in detail in Chapter 5 of the reference (1).

\[
\delta^2 = (N - 1)\delta_b^2
\]

Here, \( \delta \) is the error of the processor, \( \delta_b \) is the round-off error. Because the word length of the input is 16, \( \delta_b = 2^{-16} \). Hence, \( \delta^2 = (N - 1)\delta_b^2 = 1023 \times 2^{-16 \times 2} \),

\[
\delta = 4.87984 - 4
\]

The \( \text{MAX}(Z) \)s for FFT_PROCESSOR_ver_2_1, FFT_PROCESSOR_ver_2_2 and FFT_PROCESSOR_ver_3_1 are equal to 0.
Chapter 10 Conclusion

So far, the FFT processor is constructed by the concept mentioned at the beginning of the thesis. The effort on the MATLAB model is approximately 40% of all design work. It is feasible to use MATLAB as far as possible when a processor is designed. It can simplify and accelerate the design path. But, the best point where the transform from MATLAB to VHDL is to be done is unclear. Because MATLAB has no time characteristics, it is not able to describe the concurrent and pipelined architecture. It is not convenient to perform the bit operation with MATLAB. Then, I transfer form MATLAB to VHDL before 2-memory model, because memory assignment needs bit operations.
References

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/
copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Rongzeng Mu]