


# Chapter 5

## Implementation of the Proposed FFT Architecture

This chapter is discusses our architecture. First, we explain the reason for our algorithmic and architectural design selection. Next, we introduce the physical design for our FFT architecture including the purpose and principles of each individual design circuit.

### 5.1 Algorithmic and Architectural Design



Some types of algorithms have already been discussed in Chapter 3. We choose the radix-8 algorithm to realize our design to achieve high speed. However, this algorithm results in large die area in fabrication. So we restructure the signal flow of the radix-8 butterfly operation resulting in four complex multipliers.

To achieve high throughput and small area, we propose a memory-based variable length FFT architecture despite the fact that pipeline architectures are commonly used in academic research. The reason is that the latter require a large hardware cost for long length FFTs. Our architecture uses a dual port synchronous SRAM and a pipelined butterfly operation and thus can satisfy the high-speed requirement. To require a minimum amount of memory, we use an effective in-place memory addressing scheme for both fixed radix and mixed radix algorithms. We also consider a method to realize a suitable address pointer generator for both fixed and mixed radix algorithms concurrently.

## 5.2 Proposed Architecture Design

We implement an FFT processor that operates on variable data lengths of 512, 1024, 2048 and 4096 points to be used in the DMT based VDSL system with TSMC 0.25  $\mu\text{m}$  CMOS Technology. The typical choice of the required wordlength (with sign bit) for input and output data is 16 bits. The internal FFT architecture for our design is composed of the following individual components:

- FFT control block
- Serial to parallel module
- Radix-8 butterfly processing element
- Complex multiplier
- Reduced twiddle factor ROM table
- Static RAM (SRAM)
- Address pointer generator
- Data reordering unit

The following sections will discuss the purpose and principle of the detailed design circuit. Figure 5.1 illustrates the internal architecture of the FFT Processor.

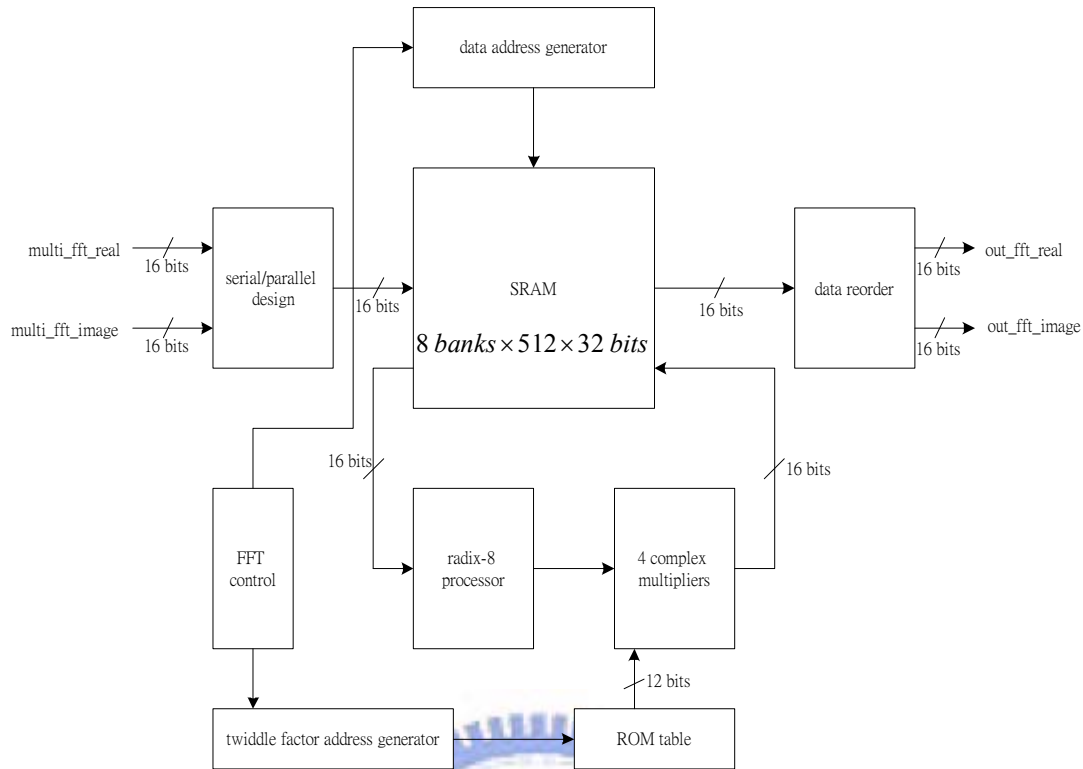


Figure 5.1 Schematic block diagrams of internal architecture for proposed architecture

### 5.3 FFT Control Block Design

This design includes a state machine to generate the required control signals. The *fft\_mode* signal selects the input data length  $N$ .

- *fft\_mode* = 00,  $N=512$
- *fft\_mode* = 01,  $N=1024$
- *fft\_mode* = 10,  $N=2048$
- *fft\_mode* = 11,  $N=4096$

The *fft\_start* signal controls the overall system operation. The signal *first\_valid*=1 indicates that data is being written to the SRAM. The signal *fft\_core\_valid*=1 indicates that the radix-8 butterfly operation is being performed. The signal *last\_valid*=1 indicates that data reordering and data output is complete. Figure

5.2 shows a simulation waveform involving these signals.

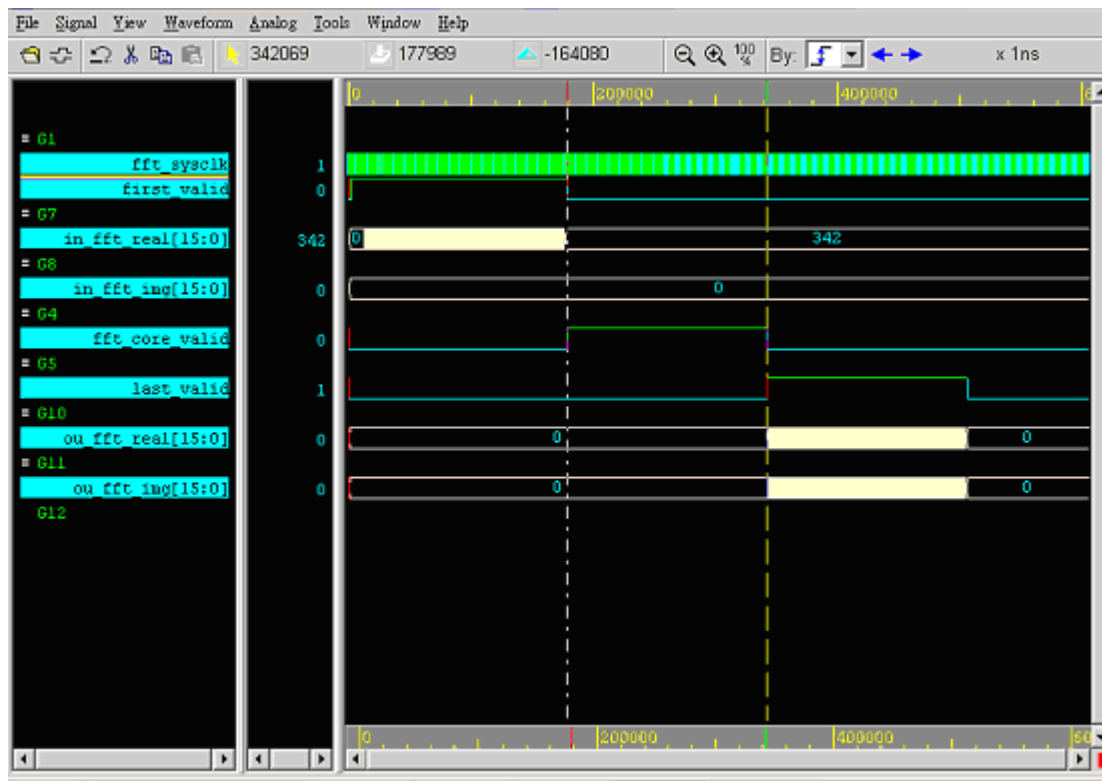


Figure 5.2 Simulation of FFT Processor control signal

## 5.4 Serial to Parallel Design

The block diagram of the serial to parallel module is shown in Figure 5.3. The function of this module is to load the first incoming sequential data into SRAM before the FFT operation begins. We must transform the sequential data to parallel data so that the correct data can be written into memory. After transformation, we must write all data into SRAM. So we use the write commutator shown in Figure 5.24 to rotate data one position in each column address at the same time. The objective of this method is to complete successful reading of the required data in one clock cycle to satisfy the condition of no data conflict. The block *Butterfly count* in Figure 5.3

determines the value of *block\_index*. The *block\_index* determines which commutator is used. This means that if *block\_index*=0, we use the commutator in Figure 5.24 (a) to correctly order the data before writing it into SRAM. If the *block\_index*=1, we use the commutator in Figure 5.24 (b). If the *block\_index*=7, we use the commutator in Figure 5.24 (g), and so forth.

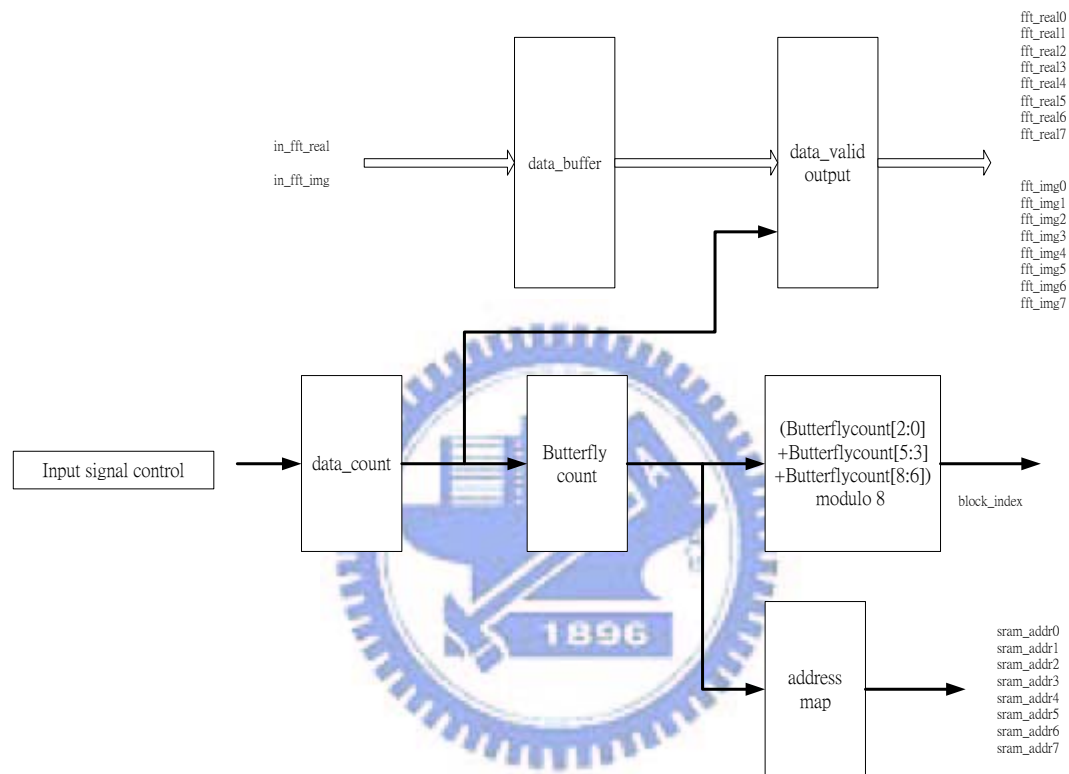


Figure 5.3 Block diagram of serial to parallel design

## 5.5 Radix-8 Butterfly Processing Element

A butterfly processing element (BF\_PE) is the fundamental component of the FFT architecture. We will explain the details of our butterfly operation and how to share the addition hardware in the following subsection.

## 5.5.1 Butterfly Computation

The proposed butterfly processing is based on the radix-8 algorithm and is also capable of mixed-radix butterfly processing (i.e. processing when the FFT transform length is not a power of 8). As shown in Figure 5.4, the radix-8 butterfly operation can be divided into three parts in order to realize the mixed radix butterfly calculation using the radix-8 hardware. These parts are designated BF\_PE1, BF\_PE2 and BF\_PE3. Eight data values are read from the SRAM and processed by the butterfly module. The system uses the in-place memory addressing method. Therefore, after completion of the butterfly calculation, the results are stored in the same memory locations that were used by the input data.

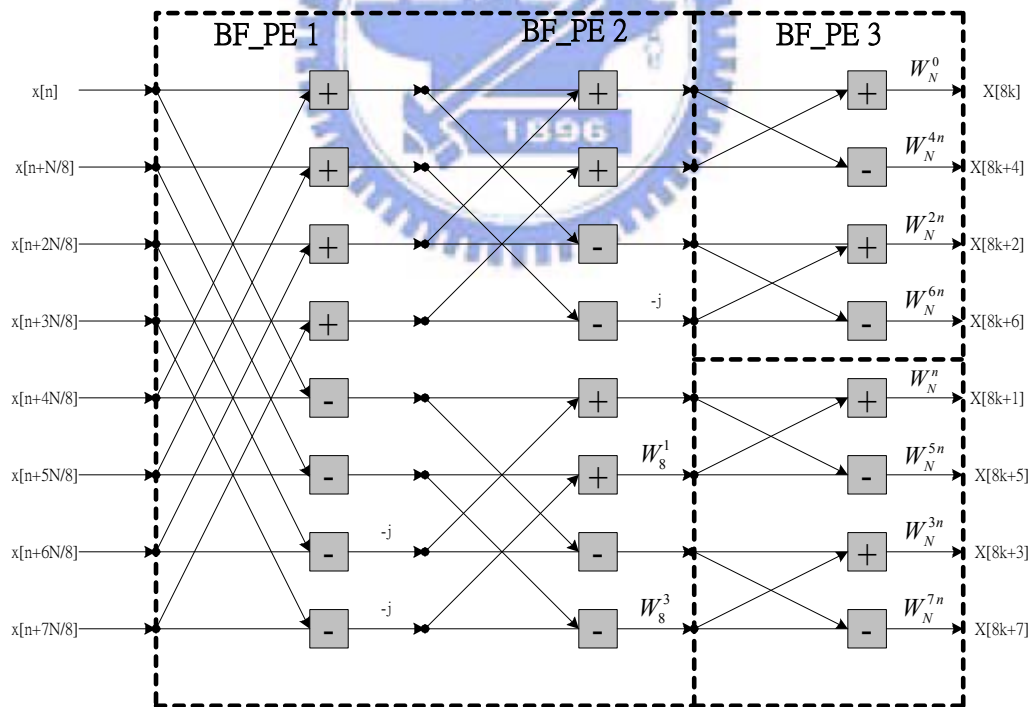


Figure 5.4 Signal flow of the radix-8 butterfly operation

Figure 5.5 depicts the proposed hardware architecture with support for multiple radix operations. The BF\_PE1 block computes the radix-2 butterfly. Both the BF\_PE1 and BF\_PE2 blocks compute the radix-4 butterfly. So additional hardware is not required for radix-2 and radix-4 butterfly computations. Radix-8 computation is collectively performed by BF\_PE1, BF\_PE2 and BF\_PE3.

The function of *data latch* is only required for the radix-8 butterfly operation. The *data latch* module latches eight data values to provide static data for the following BF\_PE3 block.

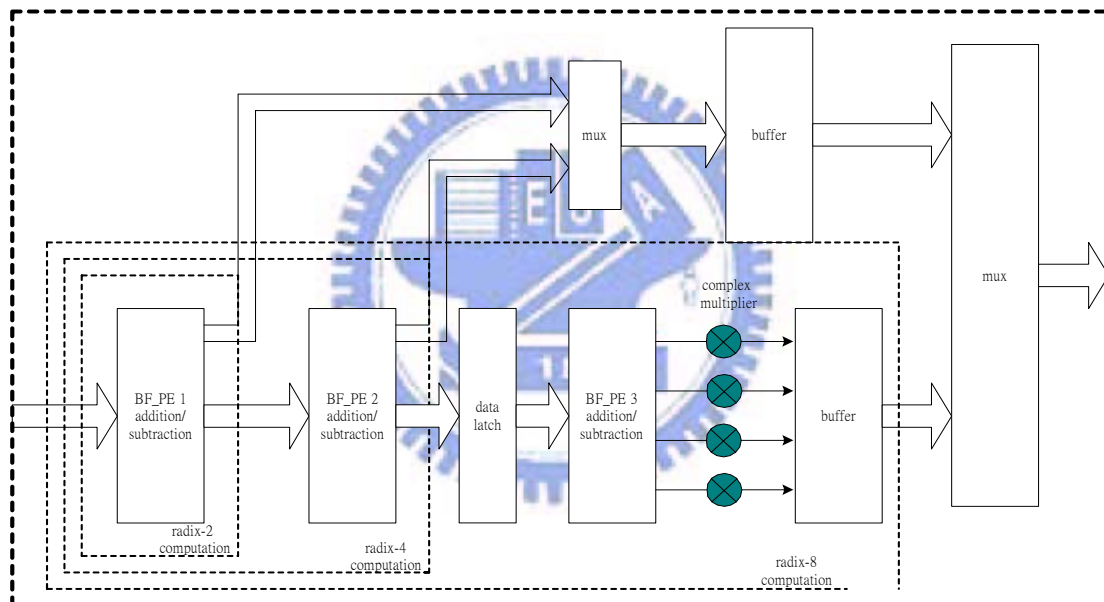


Figure 5.5 Block diagrams of implementation hardware with butterfly operation

Figures 5.6~5.8 describe the proposed pipelined butterfly data processing algorithms. The pipelined radix-8 butterfly data processing algorithm is shown in Figure 5.6 and described as follows:

- (1) In the first cycle, the first eight data inputs are read out from SRAM.
- (2) In the second cycle, BF\_PE1 and BF\_PE2 process these eight data inputs. Each output of these blocks is separated into an *upper half* and a *lower half*.

- (3) In the third cycle, BF\_PE3 processes the upper halves. At the same time, the next eight data values are read from the memory.
- (4) In the fourth cycle, BF\_PE3 processes the lower halves. At the same time, BF\_PE1 and BF\_PE2 process the next eight data values.
- (5) In the fifth cycle, all results are written back to SRAM. At the same time, BF\_PE3 processes the upper halves of the results from the fourth cycle.
- (6) In the sixth cycle, BF\_PE3 processes the lower halves of the results from the fourth cycle.
- (7) In the seventh cycle, the results from steps (5) and (6) are written back to SRAM, and so on.

We can iterate the foregoing procedure to complete the  $N$ -point butterfly operation. Similarly, the proposed pipelined radix-4 and radix-2 butterfly data processing operations are described in Figure 5.7 and Figure 5.8, respectively. The procedures of both the radix-4 and radix-2 operations are similar to that of radix-8.

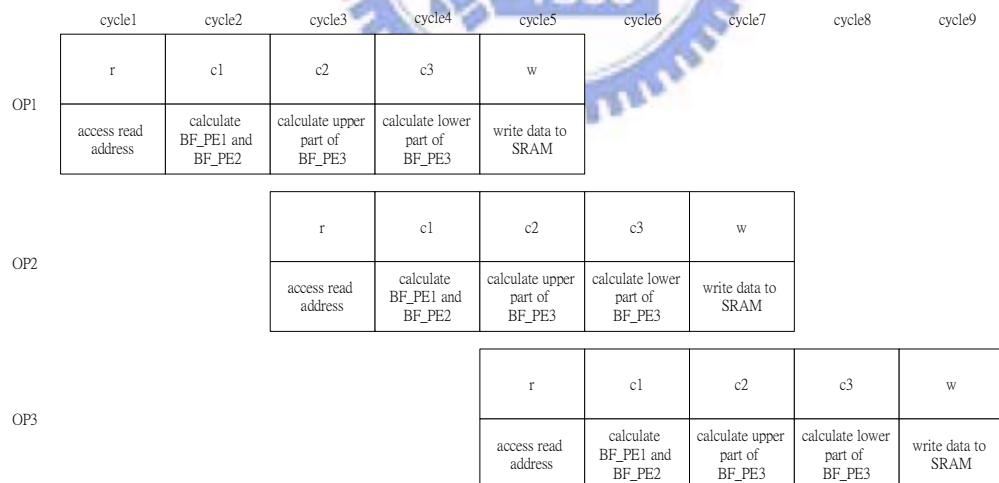


Figure 5.6 Schematic data path of the pipelined butterfly radix-8 processing element



	cycle1	cycle2	cycle3	cycle4	cycle5
OP1	r	c	w		
	access read address	calculate BF_PE1 and BF_PE2	write to SRAM		
OP2		r	c	w	
		access read address	calculate BF_PE1 and BF_PE2	write to SRAM	
OP3			r	c	w
			access read address	calculate BF_PE1 and BF_PE2	write to SRAM

Figure 5.7 Schematic data path of the pipelined butterfly radix-4 processing element

	cycle1	cycle2	cycle3	cycle4	cycle5
OP1	r	c	w		
	access read address	calculate BF_PE1	write data to SRAM		
OP2		r	c	w	
		access read address	calculate BF_PE1	write data to SRAM	
OP3			r	c	w
			access read address	calculate BF_PE1	write data to SRAM

Figure 5.8. Schematic data path of the pipelined butterfly radix-2 processing element

## 5.5.2 Sharing the Addition Operator Hardware

Figure 5.9 shows sharing the addition operator hardware among different radix modes of BF\_PE1. Figure 5.10 shows sharing addition operator hardware among different radix modes of BF\_PE2. When  $sel\_mux=100$ , the adder performs radix-2 addition. When  $sel\_mux=010$ , the adder performs radix-4 addition. When  $sel\_mux=001$ , the adder performs radix\_8 addition. To better understand these schematic diagrams, we can examine Table 5.1~5.3 listed according to radix mode and data organization.



(1) BF\_PE1 addition operator sharing architecture with radix-2,4,8

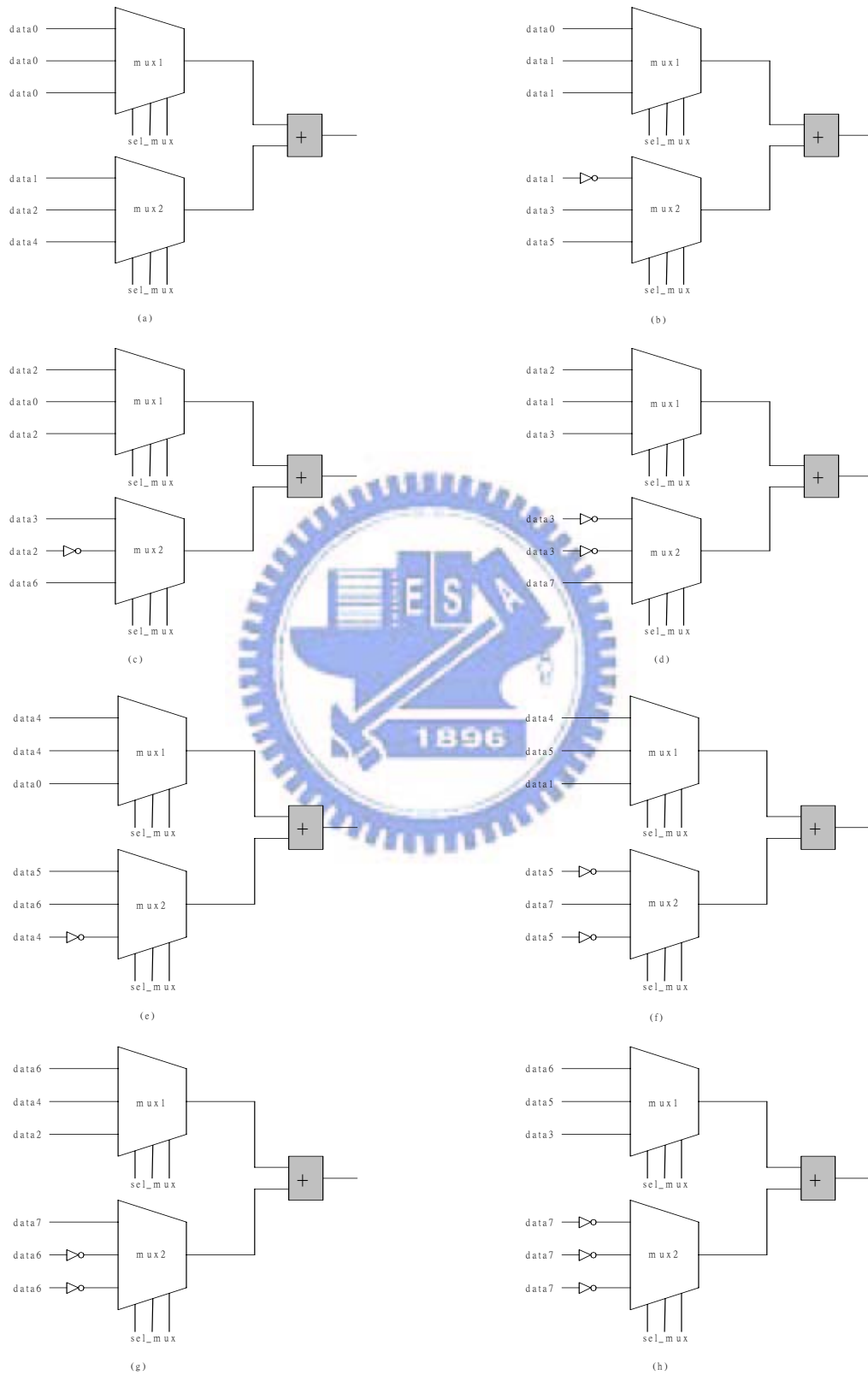


Figure 5.9 Sharing addition hardware with radix- 2, 4, 8

Table 5.1 Select addition control signal with radix 2, 4, 8

	<i>sel_mux</i> [2:0]			Input number (mux1,mux2)	Figure
	(2,	1,	0)		
Radix-2	1	0	0	(data0,data1)	Figure 5.9(a)
Radix-4	0	1	0	(data0,data2)	
Radix-8	0	0	1	(data0,data4)	
Radix-2	1	0	0	(data0,data1)	Figure 5.9(b)
Radix-4	0	1	0	(data1,data3)	
Radix-8	0	0	1	(data1,data5)	
Radix-2	1	0	0	(data2,data3)	Figure 5.9(c)
Radix-4	0	1	0	(data0,data2)	
Radix-8	0	0	1	(data2,data6)	
Radix-2	1	0	0	(data2,data3)	Figure 5.9(d)
Radix-4	0	1	0	(data1,data3)	
Radix-8	0	0	1	(data3,data7)	
Radix-2	1	0	0	(data4,data5)	Figure 5.9(e)
Radix-4	0	1	0	(data4,data6)	
Radix-8	0	0	1	(data0,data4)	
Radix-2	1	0	0	(data4,data5)	Figure 5.9(f)
Radix-4	0	1	0	(data5,data7)	
Radix-8	0	0	1	(data1,data5)	
Radix-2	1	0	0	(data6,data7)	Figure 5.9(g)
Radix-4	0	1	0	(data4,data6)	
Radix-8	0	0	1	(data2,data6)	
Radix-2	1	0	0	(data6,data7)	Figure 5.9(h)
Radix-4	0	1	0	(data5,data7)	
Radix-8	0	0	1	(data3,data7)	

(2) BF\_PE2 addition operator sharing architecture with radix-4,8

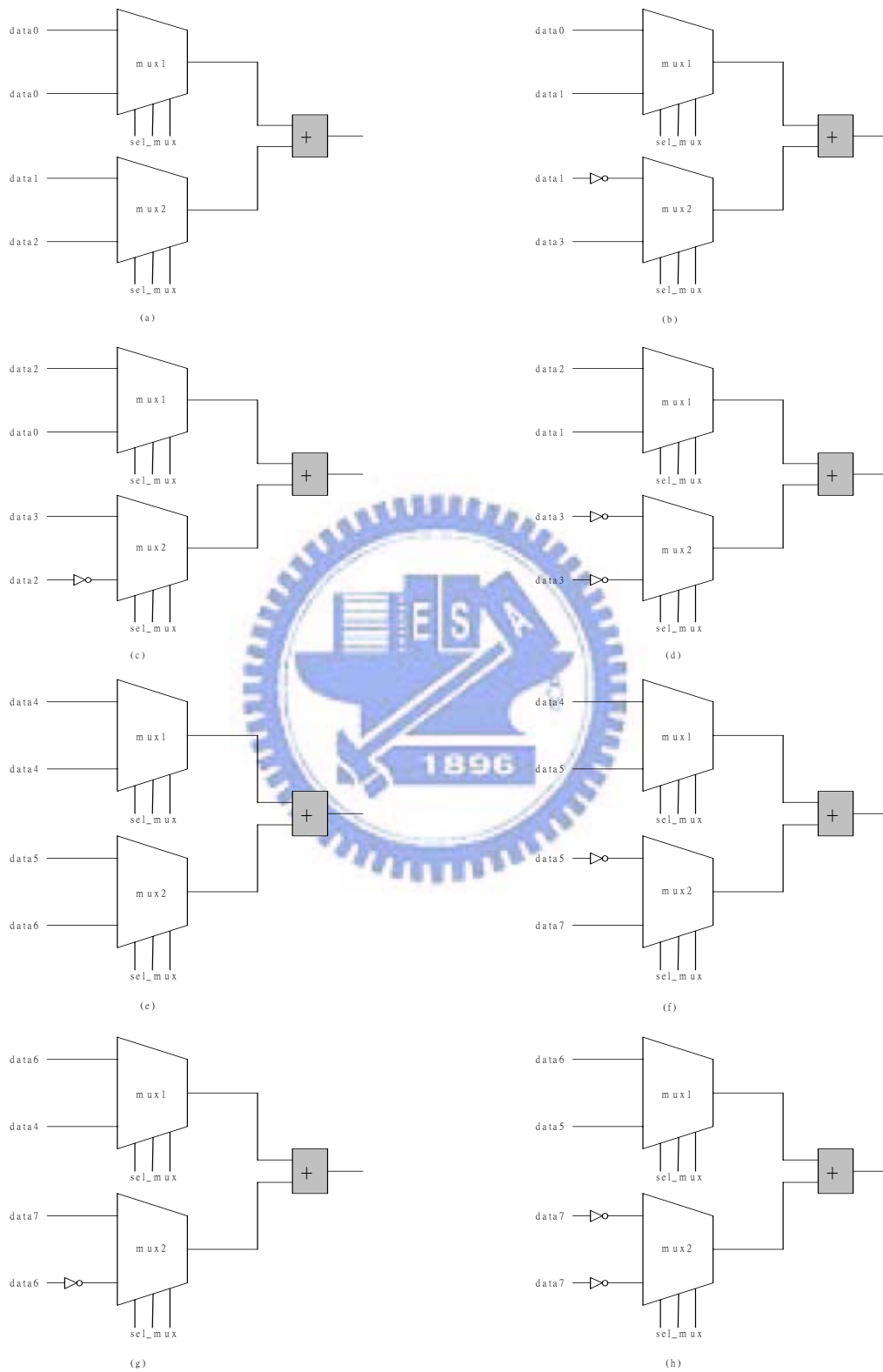


Figure 5.10 Sharing addition hardware with radix- 4, 8

Table 5.2 Select addition control signal with radix 4, 8

	<i>sel_mux</i> [2:0]			Input number (mux1,mux2)	Figure
	(2,	1,	0)		
Radix-4	0	1	0	(data0,data1)	Figure 5.10(a)
Radix-8	0	0	1	(data0,data2)	
Radix-4	0	1	0	(data0,data1)	Figure 5.10(b)
Radix-8	0	0	1	(data1,data3)	
Radix-4	0	1	0	(data2,data3)	Figure 5.10(c)
Radix-8	0	0	1	(data0,data2)	
Radix-4	0	1	0	(data2,data3)	Figure 5.10(d)
Radix-8	0	0	1	(data1,data3)	
Radix-4	0	1	0	(data4,data5)	Figure 5.10(e)
Radix-8	0	0	1	(data4,data6)	
Radix-4	0	1	0	(data4,data5)	Figure 5.10(f)
Radix-8	0	0	1	(data5,data7)	
Radix-4	0	1	0	(data6,data7)	Figure 5.10(g)
Radix-8	0	0	1	(data4,data6)	
Radix-4	0	1	0	(data6,data7)	Figure 5.10(h)
Radix-8	0	0	1	(data5,data7)	

(3) BF\_PE3 addition operator sharing architecture with radix-8

Table 5.3 Select addition control signal with radix 8

	Input number (mux1,mux2)
Radix-8	(data0,data1)
Radix-8	(data2,data3)
Radix-8	(data4,data5)
Radix-8	(data6,data7)

### 5.5.3 Complex Multiplier Hardware Design

Figure 5.4 demonstrates that the butterfly operation requires three types of multiplication. One is the multiplication by  $-j$ . The second is multiplication by a constant twiddle factor. The third is multiplication by a complex twiddle factor.

#### (1) Multiplication by $-j$

The three multiplications by  $-j$  can be realized with no extra hardware cost by simply interchanging the real and imaginary part of the product as shown in Equation (5.1).

$$(a + jb) \times (-j) = b - ja \quad (5.1)$$

#### (2) Multiplication by constant twiddle factor

The two constant multiplications  $W_8^1$  and  $W_8^3$  can be implemented with 10 complex additions by shift adders as shown in Figure 5.11. These multiplications with constant twiddle factor can be expressed in mathematical form as in Equation (5.2)(a) and (b). The value,  $\frac{\sqrt{2}}{2}$ , can also be replaced by shift adders and four real adders as given in Figure 5.12.

$$(p + jq) \times W_8^1 = (p + jq) \times \left( \frac{\sqrt{2}}{2} - j \frac{\sqrt{2}}{2} \right) \quad (5.2)(a)$$

$$(p + jq) \times W_8^3 = (p + jq) \times \left( -\frac{\sqrt{2}}{2} - j \frac{\sqrt{2}}{2} \right) \quad (5.2)(b)$$

where  $\frac{\sqrt{2}}{2} = 0.70710678 = 2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-8}$

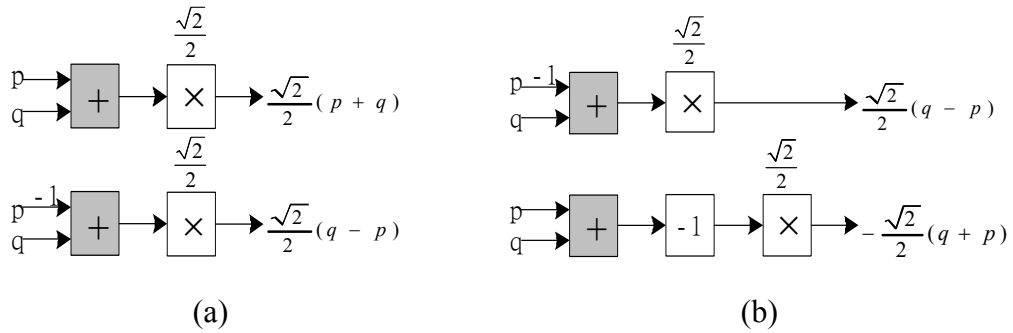


Figure 5.11 (a) Multiplication with constant twiddle factor  $W_8^1$

(b) Multiplication with constant twiddle factor  $W_8^3$

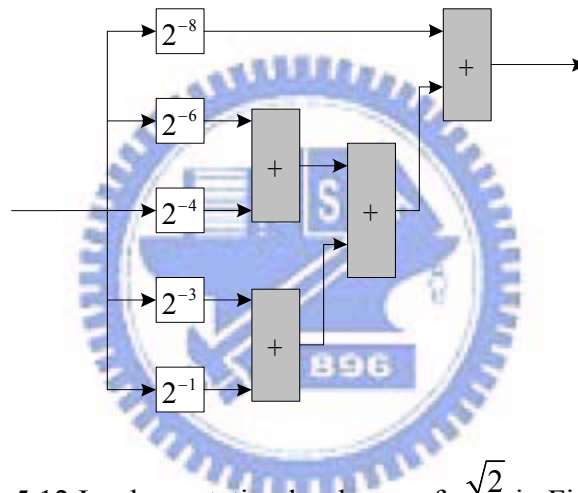


Figure 5.12 Implementation hardware of  $\frac{\sqrt{2}}{2}$  in Figure 5.11

### (3) Multiplication by complex twiddle factor

One complex multiplier can be realized by four real multiplications and two real additions as shown in Figure 5.13. Its mathematical form can be expressed as  $(p + jq)(r + js) = (pr - qs) + j(ps + qr)$ . One complex multiplier occupies large chip area in hardware implementations. Fortunately, complex multiplication can also be performed by three real multiplications and five real additions as shown in Figure

5.14 and expressed mathematically as

$$(p + jq)(r + js) = \{r(p - q) + q(r - s)\} + j\{s(p + q) + q(r - s)\} .$$



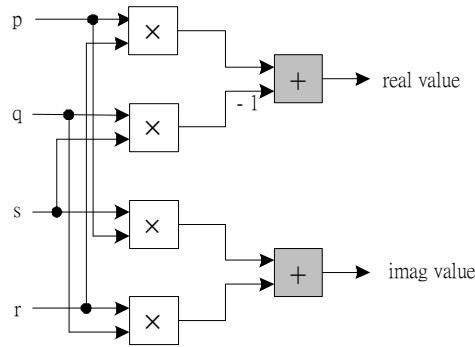


Figure 5.13 Complex multiplier with four real multiplications and two real additions

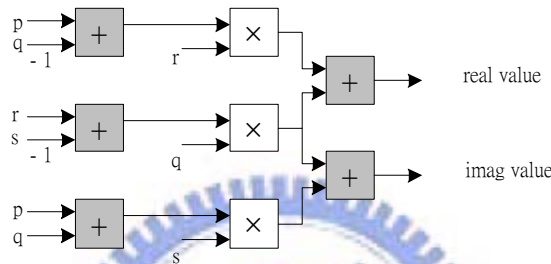


Figure 5.14 Complex multiplier with three real multiplications and five real additions

## 5.6 Twiddle Factor ROM Design

In this section, we first introduce the reduced twiddle factor coefficient design. Next, we describe how to design the twiddle factor circuit design for the fixed length FFT. Finally we introduce an implementation for the variable length FFT.

### 5.6.1 Reduced Twiddle Factor ROM Design

$N$ -point radix-8 FFT implementations can require seven complex twiddle factor coefficients  $W_N^n, W_N^{2n}, \dots, W_N^{7n}$ . Such implementations can require a twiddle factor ROM table to store the real and imaginary parts of these values which have phase

angles in the range  $(0, 2\pi)$  in the complex plane. If we store all required coefficient values in a ROM table, we must use a large chip area. Thus, this subsection presents a method to reduce the size of the twiddle factor ROM table.

Our method is modified from the techniques in [21] and is also an efficient design for long length FFTs. It is only necessary to store the twiddle factor coefficients between the interval  $0 \sim N/8$ . We denote the interval  $0 \sim N/8$  as region 0. The remaining interval regions are listed in Figure 5.15 and Table 5.4. The storage coefficients in region 0 are only in  $\left(0, \frac{\pi}{4}\right)$  to save hardware cost because it can represent all the angles in  $(0, 2\pi)$  by exploiting the symmetry of the sine and cosine functions. This means that the sine of elements in  $\left(0, \frac{\pi}{4}\right)$  are equal to the cosine of elements in  $\left(\frac{\pi}{4}, \frac{\pi}{2}\right)$  and vice versa. Thus, if the values in region 0 are known (stored in a reduced size ROM), the values from all the regions can be computed.

Table 5.4 Interval regions of twiddle factor design

No	region	Interval	boundary
(a)	0	$0 \leq m \leq \frac{N}{8}$	boundary0=0 boundary1=N/8
(b)	1	$\frac{N}{8} + 1 \leq m \leq \frac{N}{4} - 1$	boundary2=(N/4)-1
(c)	2	$\frac{N}{4} \leq m \leq \frac{3N}{8}$	boundary3=3N/8
(d)	3	$\frac{3N}{8} + 1 \leq m \leq \frac{N}{2} - 1$	boundary4=(N/2)-1
(e)	4	$\frac{N}{2} \leq m \leq \frac{5N}{8}$	boundary5=5N/8
(f)	5	$\frac{5N}{8} + 1 \leq m \leq \frac{3N}{4} - 1$	boundary6=(3N/4)-1
(g)	6	$\frac{3N}{4} \leq m \leq \frac{7N}{8}$	boundary7=7N/8
(h)	7	$\frac{7N}{8} + 1 \leq m \leq N - 1$	

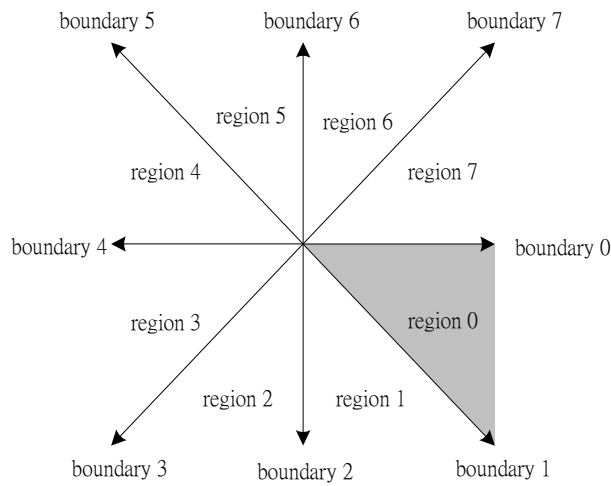


Figure 5.15 Twiddle factor boundary diagram

When the twiddle factor address generator calculates the addresses in each region, the corresponding addresses of region0 are given by Equation (5.3) through Equation (5.10). Table 5.5 lists the parameters used in Equations (5.3) through (5.10).

Table 5.5 Description of twiddle factor parameters

	Description
$W_N^m = W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$	Twiddle factor coefficient
$N$	FFT length size
$m$	The actual address
$t$	$\log_2 \frac{N}{2}$
$R_{region0\_addr}$	The real data value of the region0 address
$I_{region0\_addr}$	The image data value of the region0 address
'~'	A complement operation

$$\text{No.(a)} \quad \text{region0\_addr} = \text{actual\_addr}[t-2:0] \quad 0 \leq m \leq \frac{N}{8}$$

$$\text{actual\_data} = [R_{\text{region0\_addr}}] + j[I_{\text{region0\_addr}}] \quad (5.3)$$

$$\text{No.(b)} \quad \text{region0\_addr} = \sim \text{actual\_addr}[t-2:0] + 1 \quad \frac{N}{8} + 1 \leq m \leq \frac{N}{4} - 1$$

$$\text{actual\_data} = [\sim I_{\text{region0\_addr}}] + j[\sim R_{\text{region0\_addr}}] \quad (5.4)$$

$$\text{No.(c)} \quad \text{region0\_addr} = \text{actual\_addr}[t-2:0] \quad \frac{N}{4} \leq m \leq \frac{3N}{8}$$

$$\text{actual\_data} = [I_{\text{region0\_addr}}] + j[\sim R_{\text{region0\_addr}}] \quad (5.5)$$

$$\text{No.(d)} \quad \text{region0\_addr} = \sim \text{actual\_addr}[t-2:0] + 1 \quad \frac{3N}{8} + 1 \leq m \leq \frac{N}{2} - 1$$

$$\text{actual\_data} = [\sim R_{\text{region0\_addr}}] + j[I_{\text{region0\_addr}}] \quad (5.6)$$

$$\text{No.(e)} \quad \text{region0\_addr} = \text{actual\_addr}[t-2:0] \quad \frac{N}{2} \leq m \leq \frac{5N}{8}$$

$$\text{actual\_data} = [\sim R_{\text{region0\_addr}}] + j[\sim I_{\text{region0\_addr}}] \quad (5.7)$$

$$\text{No.(f)} \quad \text{region0\_addr} = \sim \text{actual\_addr}[t-2:0] + 1 \quad \frac{5N}{8} + 1 \leq m \leq \frac{3N}{4} - 1$$

$$\text{actual\_data} = [I_{\text{region0\_addr}}] + j[R_{\text{region0\_addr}}] \quad (5.8)$$

$$\text{No.(g)} \quad \text{region0\_addr} = \text{actual\_addr}[t-2:0] \quad \frac{3N}{4} \leq m \leq \frac{7N}{8}$$

$$\text{actual\_data} = [\sim R_{\text{region0\_addr}}] + j[I_{\text{region0\_addr}}] \quad (5.9)$$

$$\text{No.(h)} \quad \text{region0\_addr} = \sim \text{actual\_addr}[t-2:0] + 1 \quad \frac{7N}{8} + 1 \leq m \leq N - 1$$

$$\text{actual\_data} = [R_{\text{region0\_addr}}] + j[\sim I_{\text{region0\_addr}}] \quad (5.10)$$

In order to understand more clearly, we explain the concept of the reduction of the twiddle factor ROM table with the help of a 64-point FFT. Table 5.6 describes some various coefficient values from different regions. When the actual address  $m$  is 010100, the region is 2. So we use Equation (5.5) (which corresponds to region 2) to

obtain the corresponding address in region 0.

Example:

$$region0\_addr = actual\_addr[3:0] \quad 16 \leq m \leq 24$$

$$actual\_data = [I_{region0\_addr}] + j[\sim R_{region0\_addr}]$$

When we know the corresponding address of region 0, we obtain the coefficient values in region 2 by complementing the real and imaginary parts of those values at this address in region 0. For example, if the coefficient value in region 0 is  $0.923880+j(-0.382683)$ , then the coefficient values in region 2 would be  $(-0.382683)+j(-0.923880)$  (from Table 5.6).

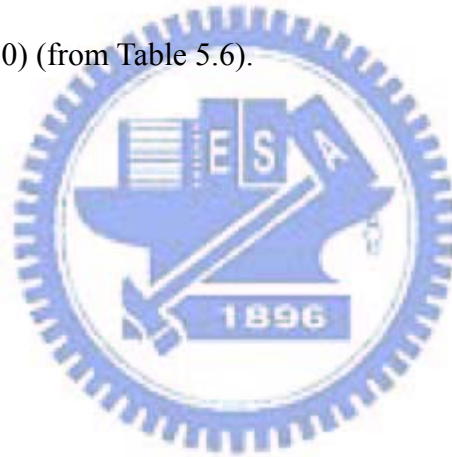


Table 5.6 Some various coefficient values with different regions for 64-point FFT

address ( <i>m</i> )	Coefficient		12bit quantized coefficient		region
	real	Imag	real	imag	
0	1.000000	0.000000	400	000	region 0
1	0.995185	-0.098017	3FB	F9C	
2	0.980785	-0.195090	3EC	F38	
3	0.956940	-0.290285	3D4	ED7	
4	0.923880	-0.382683	3B2	E78	
5	0.881921	-0.471397	387	E1D	
6	0.831470	-0.555570	353	DC7	
7	0.773010	-0.634393	318	D76	
8	0.707107	-0.707107	2D4	D2C	
9	0.634393	-0.773010	28A	CE8	region 1
10	0.555570	-0.831470	239	CAD	
11	0.471397	-0.881921	1E3	C79	
12	0.382683	-0.923880	188	C4E	
13	0.290285	-0.956940	129	C2C	
14	0.195090	-0.980785	0C8	C14	
15	0.098017	-0.995185	064	C05	
16	0.000000	-1.000000	000	C00	region 2
17	-0.098017	-0.995185	F9C	C05	
18	-0.195090	-0.980785	F38	C14	
19	-0.290285	-0.956940	ED7	C2C	
20	-0.382683	-0.923880	E78	C4E	
21	-0.471397	-0.881921	E1D	C79	
22	-0.555570	-0.831470	DC7	CAD	
23	-0.634393	-0.773010	D76	CE8	
24	-0.707107	-0.707107	D2C	D2C	
25	-0.773010	-0.634393	CE8	D76	region 3
26	-0.831470	-0.555570	CAD	DC7	
27	-0.881921	-0.471397	C79	E1D	
28	-0.923880	-0.382683	C4E	E78	
29	-0.956940	-0.290285	C2C	ED7	
30	-0.980785	-0.195090	C14	F38	
31	-0.995185	-0.098017	C05	F9C	
...	...	...	...	...	...
...	...	...	...	...	...

## 5.6.2 Twiddle Factor Circuit Design for Fixed FFT Length

Section 5.6.1 presented the mathematical description of the reduced twiddle factor ROM table design. In this section, we propose a hardware implementation for it. Our hardware design for the fixed-length FFT twiddle factor as shown in Figure 5.16 is required only for the storage coefficient values in region0. All remaining twiddle factors from other regions are generated by first mapping into region0. This mapping is illustrated in Figure 5.17 and described in the following procedure:

Step (1) Decide the boundary region according to the actual address  $m$ . (Figure 5.15 and Table 5.4 in Section 5.6.1)

Step (2) Convert address (First line in Equation (5.3~5.10) in Section 5.6.1)

Step (3) Convert data (Second line in Equation (5.3~5.10) in Section 5.6.1)

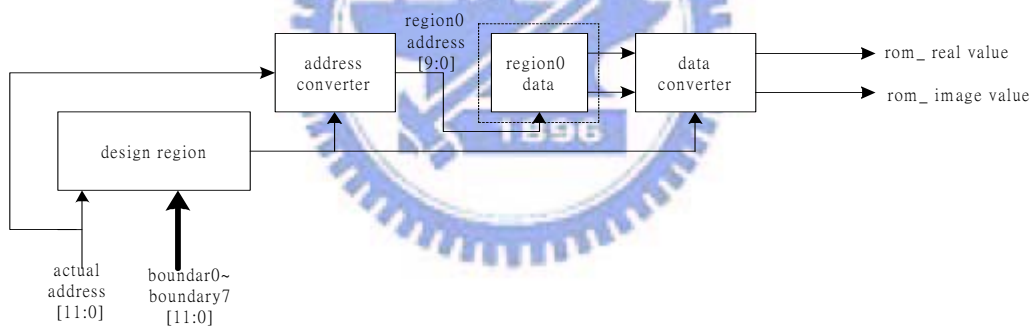


Figure 5.16 Block diagram of a twiddle factor circuit design for fixed FFT length

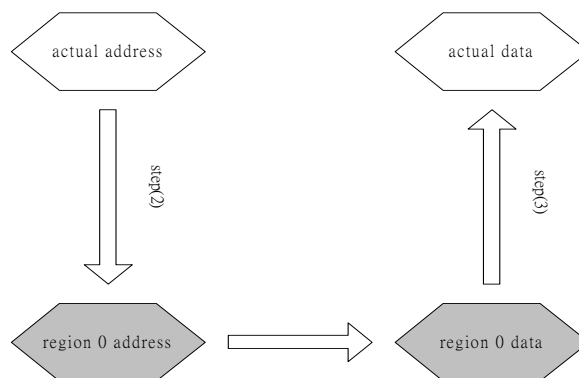


Figure 5.17 Twiddle factor mapping diagram for fixed FFT length

### 5.6.3 Twiddle Factor Design for Our Variable FFT Length

For use with variable lengths of 512, 1024, 2048, and 4096 points, we modified the dotted block in Figure 5.16 to become the dotted block in Figure 5.18. The 4096-point FFT needs 4096 twiddle factor coefficients. Using the concepts from Section 5.6.1, we design ROM table hardware for the FFT where we only store the 512 coefficients in region0. The remaining coefficients can be computed from these stored values.

Again, we introduce how to share the ROM table hardware with variable FFT length architectures of 512, 1024 and 2048 points. The concept of the twiddle factor calculation for variable FFT length is illustrated in Figure 5.19. First, we decide the actual address  $m$  of any FFT length and which interval region  $m$  is in, as shown in Figure 5.15 and Table 5.4 in Section 5.6.1. Second, we convert from the actual address to the address in region0, given in the first line of Equation (5.3~5.10). Third, we perform the operation shown in the dotted box of Figure 5.19 in order to get the required coefficient values with variable FFT length. The meaning of this method is as follows.

For a 512–point FFT operation, we read out coefficients with multiple of eight address values from the ROM. For 1024–point FFT operation, we read out coefficients with multiple of four address values from the ROM. For 2048–point FFT operation, we read out coefficients with multiple of two address values from the ROM. For 4096–point FFT operation, we read out all coefficients in the ROM.

Finally, we convert the data from region0 to the desired actual data by using the second lines in each of the Equations (5.3 ~ 5.10).



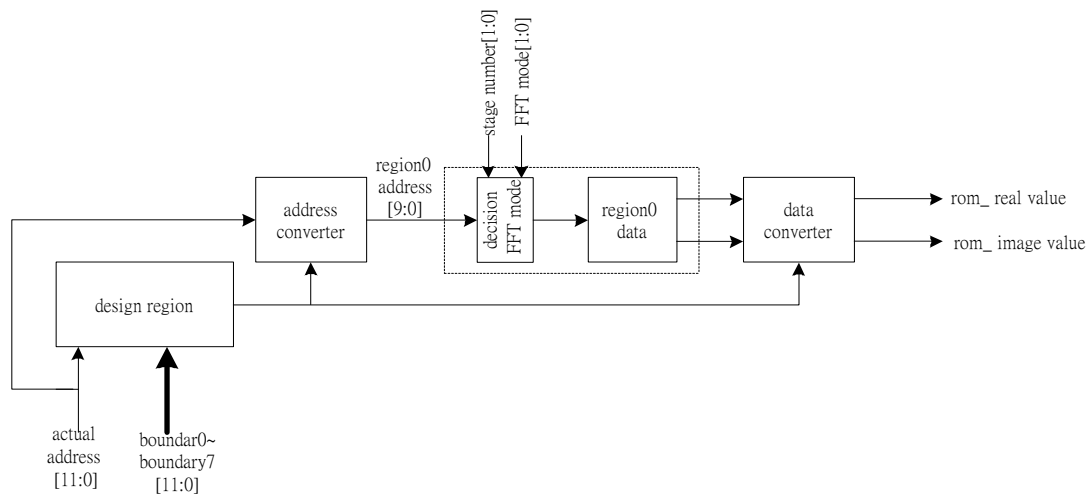


Figure 5.18 Block diagram of a twiddle factor circuit design for variable FFT length

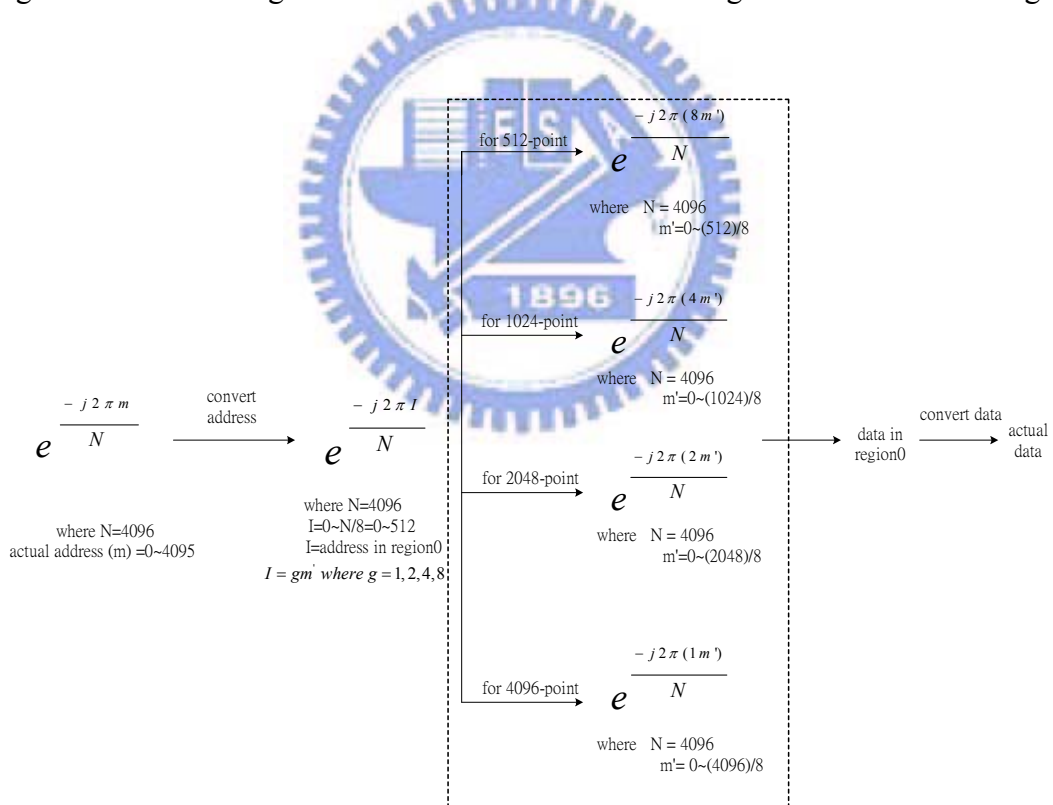


Figure 5.19 Concept of twiddle factor interchanging for variable FFT length

## 5.7 SRAM Model Design

Figure 5.20 is a block diagram of our SRAM model. The input signals  $fft\_real$  and  $fft\_image$  are from the serial to parallel circuit design. When  $first\_op=1$ , the input data is written to SRAM. When  $first\_op=0$ , the data from the radix-8 butterfly operation has already finished writing back to SRAM. The signal ports of the read/write data commutator are connected to the memory. The objective of the read/write enable signals is to control the input data assigned to the correct memory bank. The *data mapping* block in Figure 5.20 is only used for the mixed radix algorithm.

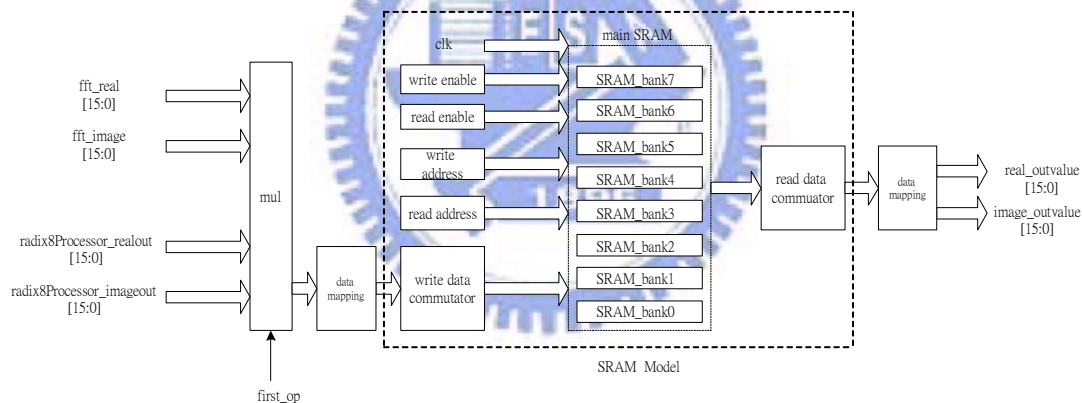


Figure 5.20 Sketch block diagram of our SRAM model

### 5.7.1 SRAM Design

This section discusses details of the main memory (Figure 5.20) of our FFT design. The data memory is configured as dual port synchronous SRAM using TSMC 0.25um fabrication technology. There are two main types of RAM: dynamic RAM (DRAM) and static RAM (SRAM). Since the control of SRAM is simple, we choose

to use SRAM (Appendix). Each RAM is available as single port or dual port. When we design the radix-8 butterfly with pipeline unit, read/write operations may be executed simultaneously. To avoid a conflict in memory access, it is necessary to use one read address port and one write address port. For this reason, we use a dual port SRAM. However, dual port memory cells have one read port and one write port and so require 33% more chip area than a single port memory.

The dual port SRAM has seven control signals as shown in Figure 5.21 and described as the following Table 5.7.

Table 5.7 Description of SRAM parameters

<i>CEn</i>	chip select control
<i>rd_Addr</i>	read address control
<i>wr_Addr</i>	write address control
<i>WEn</i>	write enable
<i>REn</i>	read enable
<i>data_in</i>	data input
<i>data_out</i>	data output

The SRAM for our FFT design requires  $N$  complex memory locations. The word-length for each memory location has a 4byte (32bits) complex datum. This means that the real components are high byte ( $x_{31}x_{30}x_{29}, \dots, x_{16}$ ) and the image components are low byte ( $x_{15}x_{14}x_{13}, \dots, x_0$ ).

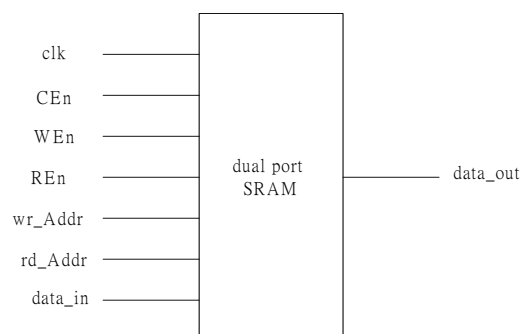


Figure 5.21 Basic block diagram of dual port SRAM

In accordance with the foregoing analysis of Chapter 3, we select the radix-8 algorithm. Therefore, we separate the SRAM into 8 memory banks (SRAM\_bank0, SRAM\_bank1, SRAM\_bank2,..., SRAM\_bank7), as shown in Figure 5.20. Each bank of SRAM has  $512 \times (4\text{byte}/\text{word}) = 2048\text{byte}$  for variable-length FFT (512, 1024, 2048, 4096). Therefore, the total number of words in the SRAM is  $8 \times 2048\text{ byte} = 16384\text{ byte}$ .

We perform concurrently eight data read/ write operations from/to the memory at an assigned address. For this reason, each bank of SRAM has its own read address ( $rd\_Addr0, rd\_Addr1, rd\_Addr2, \dots, rd\_Addr7$ ) and write address ( $wr\_Addr0, wr\_Addr1, wr\_Addr2, \dots, wr\_Addr7$ ), as shown in Figure 5.22. When we do each iteration read/write operation, each bank address can be controlled together. The design of address generators for data and twiddle factor coefficients will be discussed in the next section.

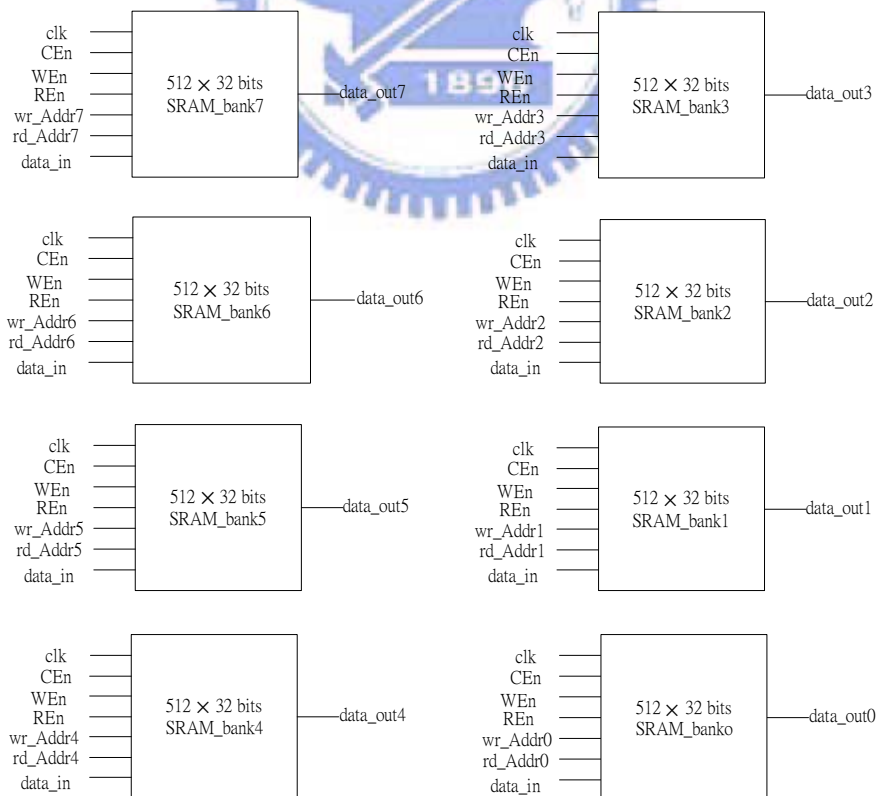


Figure 5.22 Detailed block diagram of our SRAM design

## 5.7.2 Read /Write Data Commutator

Eight types of commutators for the data-in port and eight types of commutators for the data-out port (corresponding to the eight banks of SRAM) are necessary for implementation of the proposed radix-8 operation because eight separate data reordering configurations are required as depicted in Figure 5.24. Since the data reorderings are different from one another, we utilize eight different commutators to exchange the required data so that computation of the radix-8 butterfly is convenient. We explain the commutator operation with the help of a 64-point data index diagram shown in Figure 5.23. We assume that the data index order reading from each bank of SRAM is  $\{57,1,9,17,25,33,41,49\}$ , but the butterfly computation needs the index order  $\{1,9,17,25,33,41,49,57\}$ . Therefore, we use the correct commutator to exchange the desired data ordering. To understand more clearly, the necessary butterfly orderings along with the eight read/write index orderings and corresponding commutators are listed in Table 5.8. This method is also similar to other transform lengths of 512, 1024, 2048 and 4096.

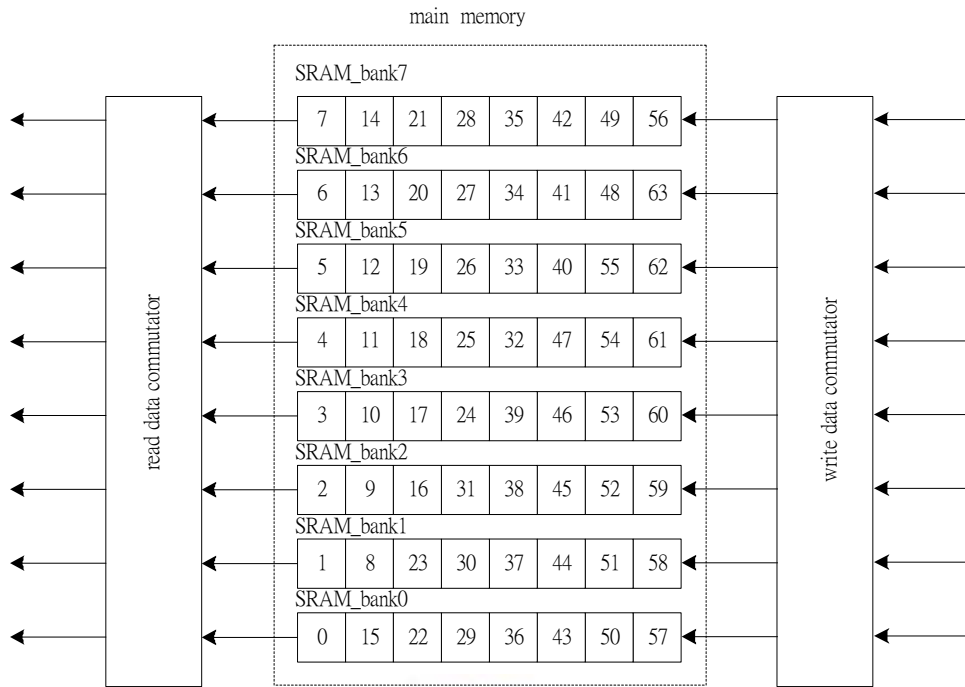


Figure 5.23 Function of commutator with the help of 64-point data index

Table 5.8 Function of eight types of commutator with the help of 64-point FFT

Figure	read data index from SRAM	butterfly necessary order	write data index to SRAM
Figure 5.24 (a)	{0,8,16,24,32,40,48,56}	{0,8,16,24,32,40,48,56}	{0,8,16,24,32,40,48,56}
Figure 5.24 (b)	{57,1,9,17,25,33,41,49}	{1,9,17,25,33,41,49,57}	{57,1,9,17,25,33,41,49}
Figure 5.24 (c)	{50,58,2,10,18,26,34,42}	{2,10,18,26,34,42,50,58}	{50,58,2,10,18,26,34,42}
Figure 5.24 (d)	{43,51,59,3,11,19,27,35}	{3,11,19,27,35,43,51,59}	{43,51,59,3,11,19,27,35}
Figure 5.24 (e)	{36,44,52,60,4,12,20,28}	{4,12,20,28,36,44,52,60}	{36,44,52,60,4,12,20,28}
Figure 5.24 (f)	{29,37,45,53,61,5,13,21}	{5,13,21,29,37,45,53,61}	{29,37,45,53,61,5,13,21}
Figure 5.24 (g)	{22,30,38,46,54,62,6,14}	{6,14,22,30,38,46,54,62}	{22,30,38,46,54,62,6,14}
Figure 5.24 (h)	{15,23,31,39,47,55,63,7}	{7,15,23,31,39,47,55,63}	{15,23,31,39,47,55,63,7}

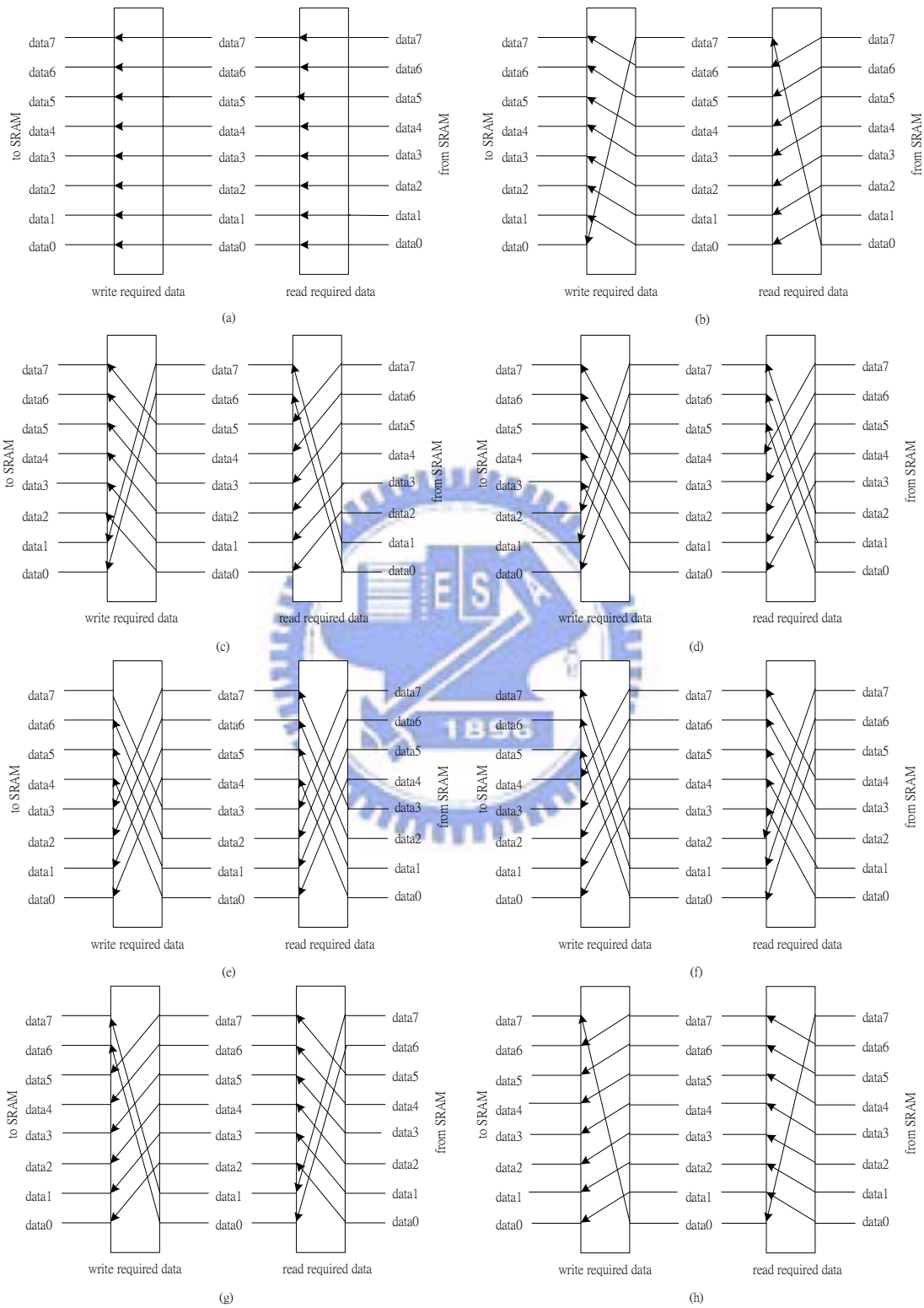


Figure 5.24 Eight types of read/write commutators

## 5.8 Address Pointer Generation

The purpose of address generators is to control the input and output data from SRAM with the correct addresses. There are two address generators. One is the data address generator and the other is the twiddle factor coefficient address generator. The details of their functions will be discussed in the following subsection.

### 5.8.1 Data Address Pointer Generation

The data address pointer generation is based on a 12-bit shift register. Different FFT modes generate variable transform lengths. The schematic diagram of variable-length FFT for the data address generator is shown in Figure 5.25. The address generator keeps track of which butterfly is being performed in which stage number and generates the correct address.

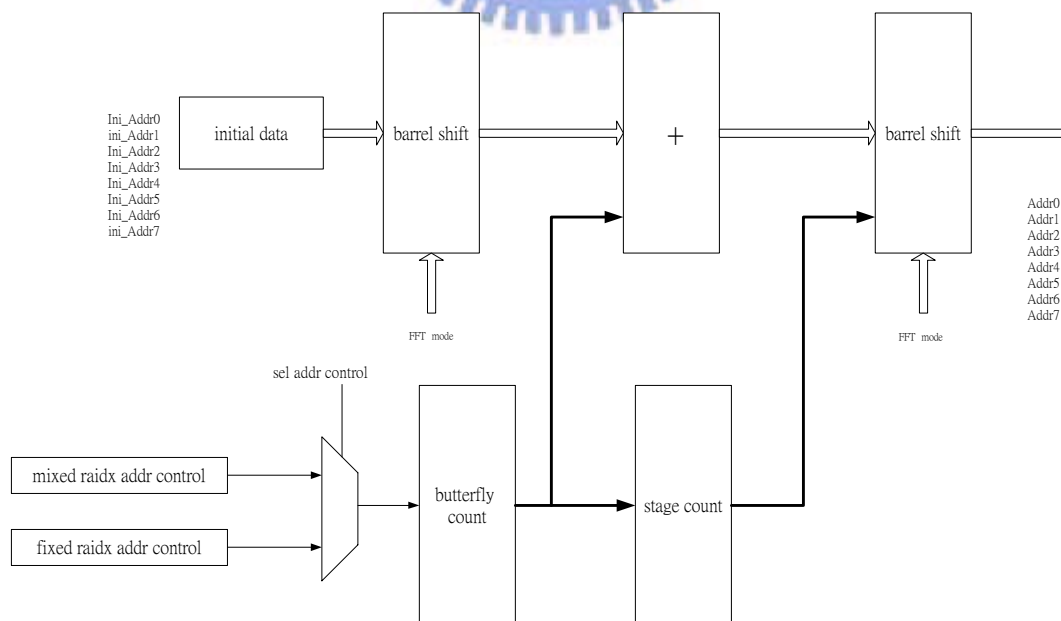


Figure 5.25 Schematic diagram of data address generator for variable length



In Table 5.9, FFT mode 00 stands for the 512-point FFT operation. For 512-point, there are three stages based on radix-8 computation and each stage contains 64 butterflies. Figure 5.26 represents the data address generation scheme with different stages of the 512-point FFT. We can see that all data addresses are 3-bit shifted. This bit-shifted property is performed in every stage except for the first stage. A 4096-point FFT is selected with FFT mode 11. The method of its address generation (Figure 5.27) is the same as the 512-point mode.

FFT mode 01 is the 1024-point FFT. For this mode, the first three stages use the radix-8 algorithm and the final stage uses the radix-2 algorithm. The data address generation for stage1 is obtained by 3-bits shifted from the stage0 address and that for stage2 is performed by 3-bit shifted from the stage1 address. But for the final stage, the address generates a one bit shifted, as shown in Figure 5.28.

When FFT mode 10 is selected, the FFT operation is computed with 2048 points. There are four stages altogether; three stages with radix-8 and one stage with radix-4. Figure 5.29 represents the data address generation scheme for different stages of the 2048-point FFT. The address generation for radix-4 is 2-bit shifted and that for radix-8 is still 3-bit shifted except for the first stage.

To better understand the forgoing description, the detailed tables that show the particular address generation with 512, 1024 and 2048-point FFT are illustrated in Table 5.10, 5.11 and 5.12.

Table 5.9 Mode control of FFT with variable length

FFT transform length	FFT mode	stage number				algorithm
		0	1	2	3	
512	00	Radix-8	Radix-8	Radix-8	-	Fixed radix
1024	01	Radix-8	Radix-8	Radix-8	Radix-2	Mixed radix
2048	10	Radix-8	Radix-8	Radix-8	Radix-4	Mixed radix
4096	11	Radix-8	Radix-8	Radix-8	Radix-8	Fixed radix

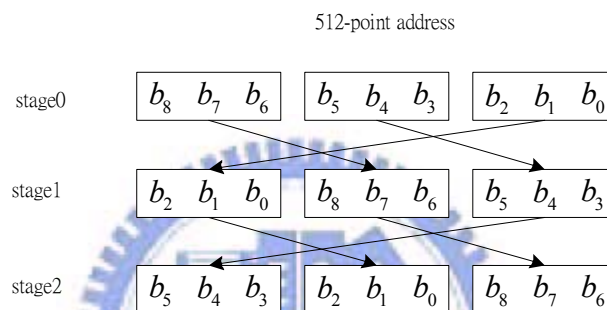


Figure 5.26 Address generation scheme with different stages of 512-point FFT

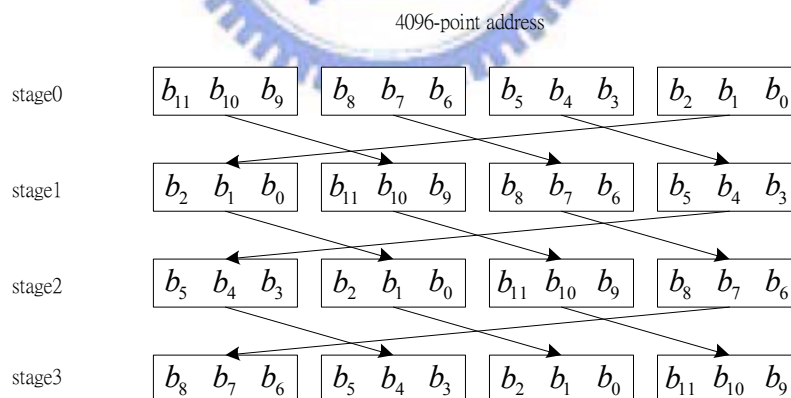


Figure 5.27 Address generation scheme with different stages of 4096-point FFT

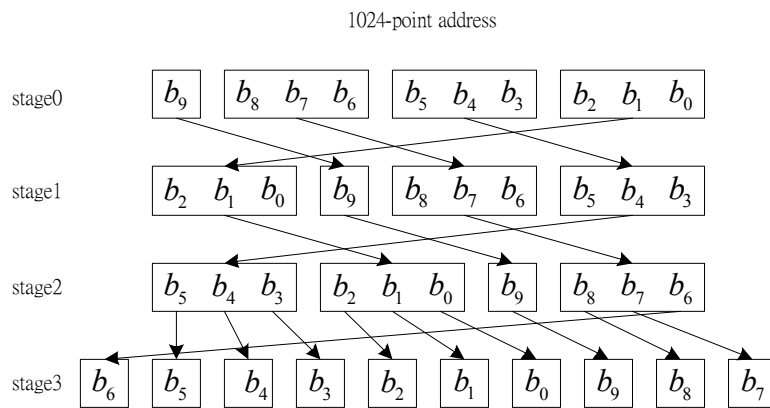


Figure 5.28 Address generation scheme with different stages of 1024-point FFT

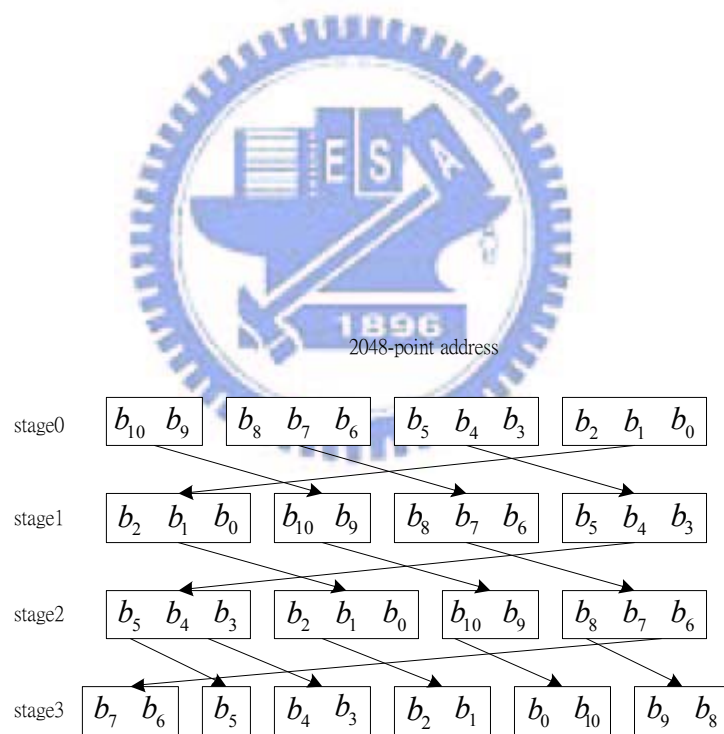


Figure 5.29 Address generation scheme with different stages of 2048-point FFT

Table 5.10 Representation of data address generation scheme at different stages of fixed radix-8 for 512-point FFT

stage	address in decimal form								address in Octal form								note
0	448	449	450	...	...	...	510	511	700	701	702	...	...	...	776	777	a base-8 digit shifted address
	384	385	386	...	...	...	446	447	600	601	602	...	...	...	676	677	
	320	321	322	...	...	...	382	383	500	501	502	...	...	...	576	577	
	256	257	258	...	...	...	318	319	400	401	402	...	...	...	476	477	
	192	193	194	...	...	...	254	255	300	301	302	...	...	...	376	377	
	128	129	130	...	...	...	190	191	200	201	202	...	...	...	276	277	
	64	65	66	...	...	...	126	127	100	101	102	...	...	...	176	177	
	0	1	2	...	...	...	62	63	000	001	002	...	...	...	076	077	
1	56	120	184	...	...	...	447	511	070	170	270	...	...	...	677	777	a base-8 digit shifted address
	48	112	176	...	...	...	439	503	060	160	260	...	...	...	667	767	
	40	104	168	...	...	...	431	495	050	150	250	...	...	...	657	757	
	32	96	160	...	...	...	423	487	040	140	240	...	...	...	647	747	
	24	88	152	...	...	...	415	479	030	130	230	...	...	...	637	737	
	16	80	144	...	...	...	407	471	020	120	220	...	...	...	627	727	
	8	72	136	...	...	...	399	463	010	110	210	...	...	...	617	717	
	0	64	128	...	...	...	391	455	000	100	200	...	...	...	607	707	
2	7	15	23	...	...	...	503	511	007	017	027	...	...	...	767	777	a base-8 digit shifted address
	6	14	22	...	...	...	502	510	006	016	026	...	...	...	766	776	
	5	13	21	...	...	...	501	509	005	015	025	...	...	...	765	775	
	4	12	20	...	...	...	500	508	004	014	024	...	...	...	764	774	
	3	11	19	...	...	...	499	507	003	013	023	...	...	...	763	773	
	2	10	18	...	...	...	498	506	002	012	022	...	...	...	762	772	
	1	9	17	...	...	...	497	505	001	011	021	...	...	...	761	771	
	0	8	16	...	...	...	496	504	000	010	020	...	...	...	760	770	

Table 5.11 Representation of data address generation scheme at different stages of mixed radix-8+2 for 1024-point FFT

stage	address in decimal form								address in Octal form								note
0	896	897	898	...	...	1022	1023	1600	1601	1602	...	...	1776	1777	a base-8 digit shifted address		
	768	769	770	...	...	894	895	1400	1401	1402	...	...	1576	1577			
	640	641	642	...	...	766	767	1200	1201	1202	...	...	1376	1377			
	512	513	514	...	...	638	639	1000	1001	1002	...	...	1176	1177			
	384	385	386	...	...	510	511	600	601	602	...	...	776	777			
	256	257	258	...	...	382	383	400	401	402	...	...	576	577			
	128	129	130	...	...	254	255	200	201	202	...	...	376	377			
	0	1	2	...	...	126	127	000	001	002	...	...	176	177			
1	112	240	368	...	...	895	1023	160	360	560	...	...	1577	1777	a base-8 digit shifted address		
	96	224	352	...	...	879	1007	140	340	540	...	...	1557	1757			
	80	208	336	...	...	863	991	120	320	520	...	...	1537	1737			
	64	192	320	...	...	847	975	100	300	500	...	...	1517	1717			
	48	176	304	...	...	831	959	060	260	460	...	...	1477	1677			
	32	160	288	...	...	815	943	040	240	440	...	...	1457	1657			
	16	144	272	...	...	799	927	020	220	420	...	...	1437	1637			
	0	128	256	...	...	783	911	000	200	400	...	...	1417	1617			
2	14	30	46	...	...	1006	1022	16	216	416	...	...	1757	1777	one bit shifted address		
	12	28	44	...	...	1004	1020	14	214	414	...	...	1755	1775			
	10	26	42	...	...	1002	1018	12	212	412	...	...	1753	1773			
	8	24	40	...	...	1000	1016	10	210	410	...	...	1751	1771			
	6	22	38	...	...	998	1014	6	26	46	...	...	1747	1767			
	4	20	36	...	...	996	1012	4	24	44	...	...	1745	1765			
	2	18	34	...	...	994	1010	2	22	42	...	...	1743	1763			
	0	16	32	...	...	992	1008	0	20	40	...	...	1741	1761			
3	15	31	47	...	...	1015	1023	7	17	27	...	...	1767	1777	one bit shifted address		
	13	29	45	...	...	1014	1022	6	16	26	...	...	1766	1776			
	11	27	43	...	...	1013	1021	5	15	25	...	...	1765	1775			
	9	25	41	...	...	1012	1020	4	14	24	...	...	1764	1774			
	7	23	39	...	...	1011	1019	3	13	23	...	...	1763	1773			
	5	21	37	...	...	1010	1018	2	12	22	...	...	1762	1772			
	3	19	35	...	...	1009	1017	1	11	21	...	...	1761	1771			
	1	17	33	...	...	1008	1016	0	10	20	...	...	1760	1770			

Table 5.12 Representation of data address generation scheme at different stages of mixed radix-8+4 for 2048-point FFT

stage	address in decimal form								address in Octal form								note
0	1792	1793	1794	...	...	2046	2047	3400	3401	3402	...	...	3776	3777	a base-8 digit shifted address     ←		
	1536	1537	1538	...	...	1790	1791	3000	3001	3002	...	...	3376	3377			
	1280	1281	1282	...	...	1534	1535	2400	2401	2402	...	...	2776	2777			
	1024	1025	1026	...	...	1278	1279	2000	2001	2002	...	...	2376	2377			
	768	769	770	...	...	1022	1023	1400	1401	1402	...	...	1776	1777			
	512	513	514	...	...	766	767	1000	1001	1002	...	...	1376	1377			
	256	257	258	...	...	510	511	400	401	402	...	...	776	777			
	0	1	2	...	...	254	255	000	001	002	...	...	376	377			
1	224	480	736	...	...	1791	2047	340	740	1340	...	...	3377	3777	a base-8 digit shifted address     ←		
	192	448	704	...	...	1759	2015	300	700	1300	...	...	3337	3737			
	160	416	672	...	...	1727	1983	240	640	1240	...	...	3277	3677			
	128	384	640	...	...	1695	1951	200	600	1200	...	...	3237	3637			
	96	352	608	...	...	1663	1919	140	540	1140	...	...	3177	3577			
	64	320	576	...	...	1631	1887	100	500	1100	...	...	3137	3537			
	32	288	544	...	...	1599	1855	040	440	1040	...	...	3077	3477			
	0	256	512	...	...	1567	1823	000	400	1000	...	...	3037	3437			
2	28	60	92	...	...	2015	2047	34	74	134	...	...	3737	3777	a base-4 digit shifted address     ←		
	24	56	88	...	...	2011	2043	30	70	130	...	...	3733	3773			
	20	52	84	...	...	2007	2039	24	64	124	...	...	3727	3767			
	16	48	80	...	...	2003	2035	20	60	120	...	...	3723	3763			
	12	44	76	...	...	1999	2031	14	54	114	...	...	3717	3757			
	8	40	72	...	...	1995	2027	10	50	110	...	...	3713	3753			
	4	36	68	...	...	1991	2023	4	44	104	...	...	3707	3747			
	0	32	64	...	...	1987	2019	0	40	100	...	...	3703	3743			
3	7	15	23	...	...	2039	2047	7	17	27	...	...	3767	3777	←		
	6	14	22	...	...	2038	2046	6	16	26	...	...	3766	3776			
	5	13	21	...	...	2037	2045	5	15	25	...	...	3765	3775			
	4	12	20	...	...	2036	2044	4	14	24	...	...	3764	3774			
	3	11	19	...	...	2035	2043	3	13	23	...	...	3763	3773			
	2	10	18	...	...	2034	2042	2	12	22	...	...	3762	3772			
	1	9	17	...	...	2033	2041	1	11	21	...	...	3761	3771			
	0	8	16	...	...	2032	2040	0	10	20	...	...	3760	3770			

## 5.8.2 Twiddle Factor Coefficient Address Pointer Generation

The coefficient address pointer generator of Figure 5.30 is similar to the data address generator which keeps track of which butterfly is being performed in which stage. Our design uses the radix-8 butterfly operation with four complex multipliers and so having 8 parallel input data and 4 output data. The radix-8 algorithm requires seven complex twiddle factor coefficients  $W_N^n, W_N^{2n}, \dots, W_N^{7n}$  where  $N$  is transform length, excluding  $W_N^{0n}$ . So we need a multiplexer to select the values of  $k$  where  $k=0,1,2,3,4,5,6,7$ . This means when  $ctrl=0$ , the butterfly output data is multiplied by  $W_N^{0n}, W_N^{4n}, W_N^{2n}, W_N^{6n}$ . When  $ctrl=1$ , the butterfly output data is multiplied by  $W_N^{1n}, W_N^{5n}, W_N^{3n}, W_N^{7n}$ .

The symbol  $n$  in  $W_N^n, W_N^{2n}, \dots, W_N^{7n}$  represents the interval  $0 \sim N/8$  and is represented by the signal  $romaddr\_mul$ . The bit rotation scheme for these values is charted in Figure 5.31. For the 512-point FFT, the  $n$  interval is from 0 to 63 and so it needs 6 bits. All bit numbers in stage1 are obtained from the bit numbers in stage0 by dividing by eight. Similarly, all bit numbers in stage2 are obtained from the bit numbers in stage1 by dividing by eight. This can be seen in Figure 5.31 (a).

The bit rotation scheme is similar to other cases such as the 1024, 2048 and 4096-point FFTs given in Figure 5.31 (b) (c) (d). To better understand the description in Figure 5.31, the detailed tables that show the particular coefficient address generation with 512, 1024, 2048 and 4096-point FFTs are illustrated in Table 5.14 through Table 5.17.

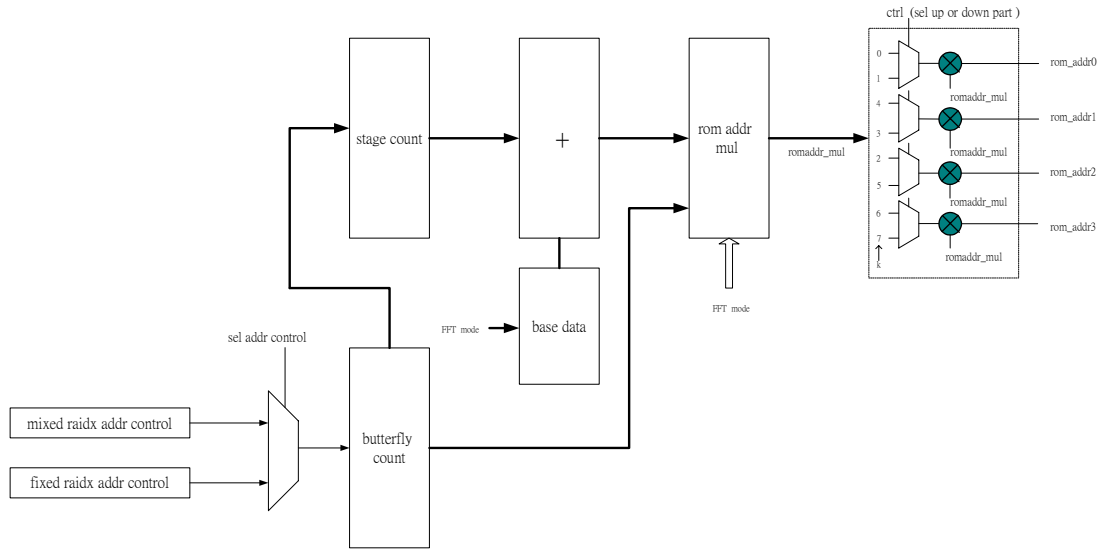


Figure 5.30 Block diagram of twiddle factor coefficient address generation

Table 5.13 Description of coefficient address generation parameters in Figure5.30

Parameter	Description
rom_addr0 rom_addr1 rom_addr2 rom_addr3	values of $W_N^m = W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$
<i>romaddr_mul</i>	<i>n</i> (its interval: 0~N/8-1)
(0,1,2,3,4,5,6,7) in dotted block	values of <i>k</i>

*stage0*:  $b_5 b_4 b_3 b_2 b_1 b_0$   
*stage1*: 0 0 0  $b_5 b_4 b_3$   
*stage2*: 0 0 0 0 0 0

(a) for 512-point FFT

*stage0*:  $b_6 b_5 b_4 b_3 b_2 b_1 b_0$   
*stage1*: 0 0 0  $b_6 b_5 b_4 b_3$   
*stage2*: 0 0 0 0 0 0  $b_6$   
*stage3*: 0 0 0 0 0 0 0

(b) for 1024-point FFT

*stage0*:  $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$   
*stage1*: 0 0 0  $b_7 b_6 b_5 b_4 b_3$   
*stage2*: 0 0 0 0 0 0  $b_7 b_6$   
*stage3*: 0 0 0 0 0 0 0 0

(c) for 2048-point FFT

*stage0*:  $b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$   
*stage1*: 0 0 0  $b_8 b_7 b_6 b_5 b_4 b_3$   
*stage2*: 0 0 0 0 0 0  $b_8 b_7 b_6$   
*stage3*: 0 0 0 0 0 0 0 0

(d) for 4096-point FFT

Figure 5.31 Rotation bit number scheme for *romaddr\_mul* (*n*) with different stages



Table 5.14 Address ( $m$ ) generation in  $W^m$  at different stages for 512-point FFT

stage	romaddr_mul ( $n$ )	rom_addr0, rom_addr1, rom_addr2, rom_addr3 ( $m$ )		Note
		$ctrl=0$ k=(0,4,2,6)	$ctrl=1$ k=(1,5,3,7)	
		0	0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	61	0,244,122,366	61,305,183,427	
	62	0,248,124,372	62,310,186,434	
	63	0,252,126,378	63,315,189,441	
1	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	3	...	...	
	4	...	...	
	5	0,20,10,30	5,25,15,35	
	6	0,24,12,36	6,30,18,42	
	7	0,28,14,42	7,35,21,49	
2	0	0,0,0,0	0,0,0,0	

Table 5.15 Address ( $m$ ) generation in  $W^m$  at different stages for 1024-point FFT

Stage	$romaddr\_mul$ ( $n$ )	rom_addr0, rom_addr1, rom_addr2, rom_addr3 ( $m$ )		Note
		$ctrl=0$ $k=(0,4,2,6)$	$ctrl=1$ $k=(1,5,3,7)$	
		0	0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	125	0,500,250,750	125,625,375,875	
	126	0,504,252,756	126,630,378,882	
	127	0,508,254,762	127,635,381,889	
1	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	13	0,52,26,78	13,65,39,91	
	14	0,56,28,84	14,70,42,98	
	15	0,60,30,90	15,75,45,105	
2	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
3	0	0,0,0,0	0,0,0,0	

Table 5.16 Address ( $m$ ) generation in  $W^m$  at different stages for 2048-point FFT

Stage	$romaddr\_mul$ ( $n$ )	$rom\_addr0, rom\_addr1, rom\_addr2, rom\_addr3$ ( $m$ )		Note
		$ctrl=0$ $k=(0,4,2,6)$	$ctrl=1$ $k=(1,5,3,7)$	
		0	0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	253	0,1012,506,1518	253,1265,759,1771	
	254	0,1016,508,1524	254,1270,762,1778	
	255	0,1020,510,1530	255,1275,765,1785	
1	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	29	0,116,58,174	29,145,87,203	
	30	0,120,60,180	30,150,90,210	
	31	0,124,62,186	31,155,93,217	
2	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	3	0,12,6,18	3,15,9,21	
3	0	0,0,0,0	0,0,0,0	

Table 5.17 Address ( $m$ ) generation in  $W^m$  at different stages for 4096-point FFT

Stage	$romaddr\_mul$ ( $n$ )	$rom\_addr0, rom\_addr1, rom\_addr2, rom\_addr3$ ( $m$ )		Note
		$ctrl=0$ $k=(0,4,2,6)$	$ctrl=1$ $k=(1,5,3,7)$	
		0	0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	509	0,2036,1018,3054	509,2545,1527,3563	
	510	0,2040,1020,3120	510,2550,1530,3570	
	511	0,2044,1022,3066	511,2555,1533,3577	
1	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	...	...	...	
	...	...	...	
	61	0,244,122,366	61,305,183,427	
	62	0,248,124,372	62,310,186,434	
	63	0,252,126,378	63,315,189,441	
2	0	0,0,0,0	0,0,0,0	
	1	0,4,2,6	1,5,3,7	
	2	0,8,4,12	2,10,6,14	
	3	0,12,6,18	3,15,9,21	
	4	0,16,8,24	4,20,12,28	
	5	0,20,10,30	5,25,15,35	
	6	0,24,12,36	6,30,18,42	
	7	0,56,28,84	7,35,21,49	
3	0	0,0,0,0	0,0,0,0	

## 5.9 Hardware Implementation Rules for Address and Data Control

In section 5.8.1, we introduced how to generate the data address. Here, we will illustrate how to obtain the addresses for each bank of SRAM using our design method.

The following steps are the workflows of the implementation for the 512-point address generator, depicted as the solid line in Figure 5.32. The workflows for the 4096-point address generator are similar to that of the 512-point one and so we will not explain further.

Step1 (Figure 5.32(a)): Give initial address  $\{0,64,128,192,256,320,384,448\}$ .

These initial addresses are added to every counter and then generated all addresses in the first stage. When we know the address in the first stage, we use the bit-shifted property to generate the other stages. (See Table 5.10)

Step2 (Figure 5.32(c)): If we assume that the address index from address generation in stage0 is  $\{1,65,129,193,257,321,385,449\}$ , we must use the address commutator in Figure 5.35(h) to get the practical assigned address  $\{449,1,65,129,193,257,321,385\}$  in each bank of SRAM. The SRAM memory location can be calculated by Equation (4.1). Similarly, the remaining address indices can also use the address commutator in Figure 5.35 to get the corresponding bank index assignment. Some memory address assignment for the 512-point FFT is in Table 5.18.

Step3 (Figure 5.32(f)): The data index order read from SRAM is {449,1,65,129,193,257,321,385}. However, FFT calculation requires the order {1,65,129,193,257,321,385,449}. For this reason, we use the commutator in Figure 5.24(b) to switch to the required order. The method of data index writing to SRAM is equal to that of read from SRAM.

Table 5.18 Memory assignment for a 512-point FFT

data index	memory bank index	memory address index
385	7	48
321	6	40
257	5	32
193	4	24
129	3	16
65	2	8
1	1	0
449	0	56

The workflows of the implementation for 1024-points are given by the dotted line in Figure 5.32.

Step1 (Figure 5.32(a)): Give initial address {0,128,256,384,512,640,768,896}.

These initial addresses are added to every counter and then all addresses are generated in the first stage. When we know the address in the first stage, we use the bit-shifted property to generate other stages. (See Table 5.11)

Step2 (Figure 5.32(b)): If the address index from address generation in stage0 is {2,130,258,386,514,642,770,898}, we use the address mapping in Figure 5.33 to reorder address index {2,514,130,642,258,770,386,898}. The first three stages for the 1024-point FFT butterfly use the address index mapping scheme shown in Figure 5.33(a) and the last stage uses

the scheme shown in Figure 5.33(b).

Step3 (Figure 5.32(c)): Using the results {2,514,130,642,258,770,386,898} from Step2, we again use the address commutator in Figure 5.35(g) to get the practical assigned address {386,898,2,514,130,642,258,770} in each bank of SRAM, as listed in Table 5.19.

Table 5.19 Memory assignment for a 1024-point FFT

data index	memory bank index	memory address index
770	7	96
258	6	32
642	5	80
130	4	16
514	3	64
2	2	0
898	1	112
386	0	48

Step4 (Figure 5.32(f)): The data index read order from SRAM is {386,898,2,514,130,642,258,770}. We use the read commutator in Figure 5.24(c) to switch to the desired order {2,514,130,642,258,770,386,898}.

Step5 (Figure 5.32(d)): But the final order in step4 cannot support the butterfly operation. To support this operation, we need to achieve index order {2,130,258,386,514,642,770,898} by performing the data mapping strategy of Figure 5.36(a).

Step6 (Figure 5.32(e)): After the butterfly computation, we employ the data mapping scheme of Figure 5.36(b) to switch the data again to arrive at the address index ordering {2,514,130,642,258,770,386,898}.

Step7 ((Figure 5.32(f)): Finally, we use the write commutator of Figure 5.24(c) to exchange the address index in order to write back to SRAM with the practical assigned address {386,898,2,514,130,642,258,770}.

The workflows of the implementation for 2048-points are given by the dotted line in Figure 5.32.

Step1 (Figure 5.32(a)): Give initial address  $\{0,256,512,768,1024,1280,1536,1792\}$ . These initial addresses are added to every counter and then generate all addresses in the first stage. When we know the address in the first stage, we use the bit shifted property to generate other stages. (See Table 5.12)

Step2 (Figure 5.32(b)): If the address index from address generation in stage0 is  $\{2,258,514,770,1026,1282,1538,1794\}$ , we use the address mapping of Figure 5.34 to reorder the address index  $\{2,514,1026,1538,258,770,1282,1794\}$ . The first three stages for the 2048-point FFT butterfly uses the address index mapping scheme of Figure 5.34(a) and the last stage uses the scheme of Figure 5.34(b).

Step3 (Figure 5.32(c)): Using the results  $\{2,514,1026,1538,258,770,1282,1794\}$  from in Step2, we again use the address commutator in Figure 5.35(g) to get the practical assigned address  $\{1282,1794,2,514,1026,1538,258,770\}$  in each bank of SRAM, given in Table 5.20.

Table 5.20 Memory assignment for a 2048-point FFT

data index	memory bank	memory address
770	7	96
258	6	32
1538	5	192
1026	4	128
514	3	64
2	2	0
1794	1	224
1282	0	160



Step4 (Figure 5.32(f)): The data index order read from SRAM is {1282,1794,2,514,1026,1538,258,770}. We use the read commutator in Figure 5.24(c) to switch the desired order to {2,514,1026,1538,258,770,1282,1794}.

Step5 (Figure 5.32(d)): But the final order in step4 cannot do butterfly calculation. So, we have to perform index order {2,258,514,770,1026,1282,1538,1794} that can operate butterfly by doing mapping data strategy in Figure 5.37(a).

Step6 (Figure 5.32(e)): After the FFT computation, we employ the data mapping scheme of Figure 5.37(b) to switch the data again that get the address index {2,514,1026,1538,258,770,1282,1794}.

Step7 ((Figure 5.32(f)): Finally, we use the write commutator in Figure 5.24(c) to exchange the address index in order to write back to the SRAM with the practical assigned address {1282,1794,2,514,1026,1538,258,770}.

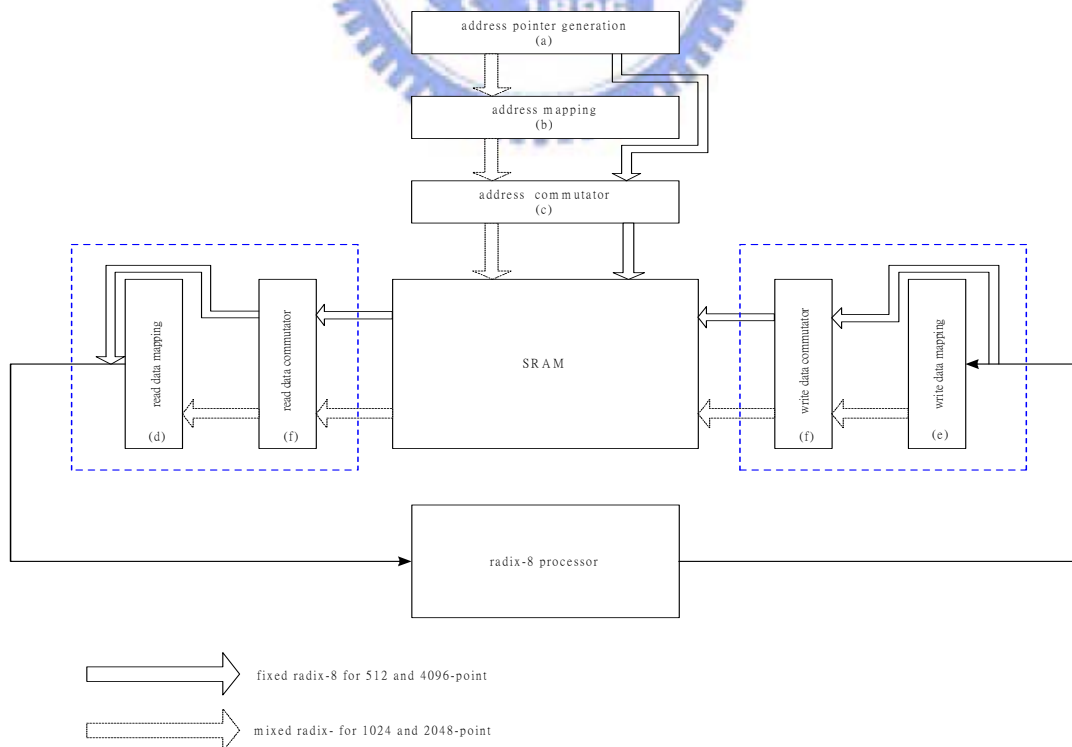


Figure 5.32 Workflow of the implementation hardware rules for address control

## (1) Address mapping

- For mixed radix-8+2 case,

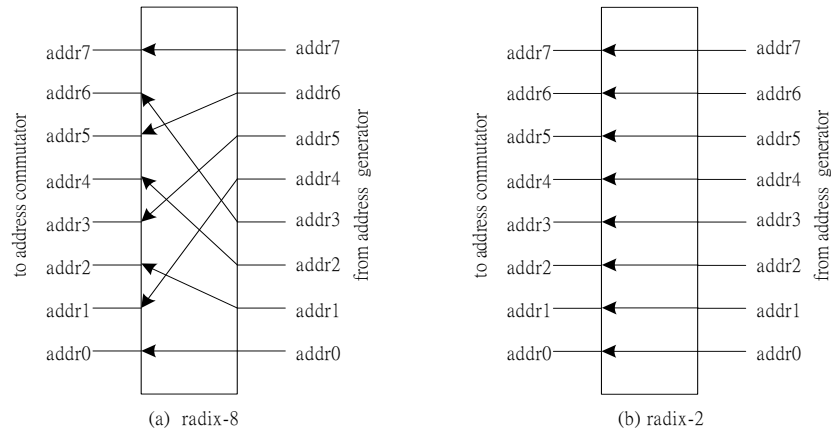


Figure 5.33 Address mapping for mixed radix-8+2 case

- For mixed radix-8+4 case,

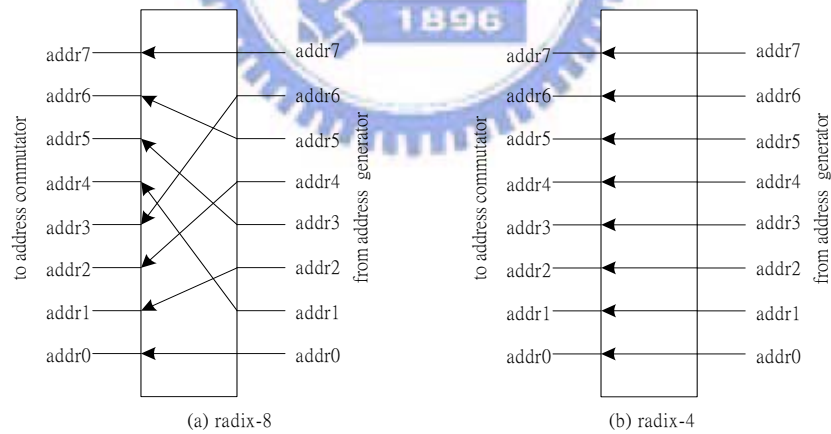


Figure 5.34 Address mapping for mixed radix-8+4 case

## (2) Address commutator

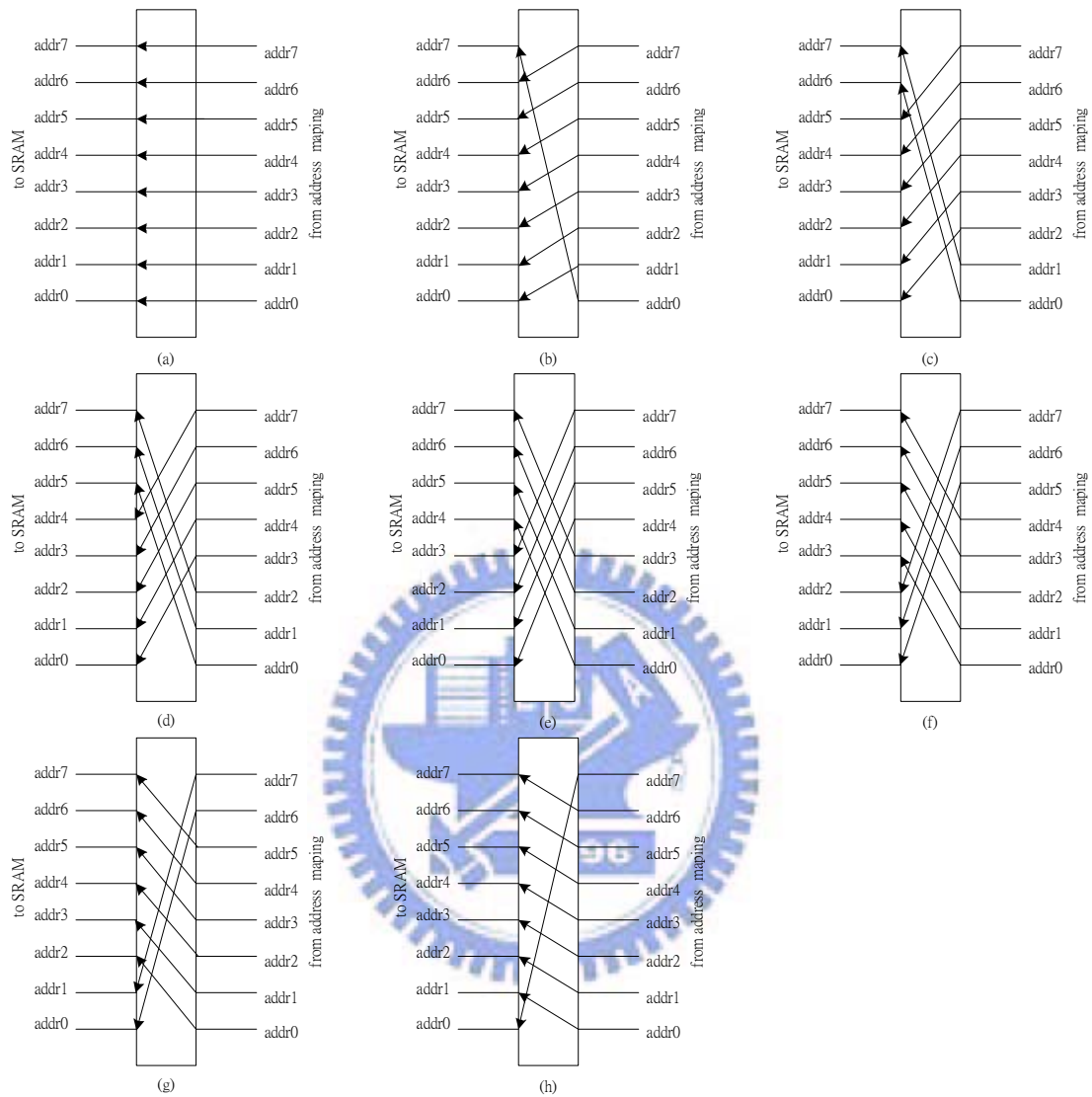


Figure 5.35 Eight types of address commutator

### (3) Read /write data mapping

- For mixed radix-8+2 case,

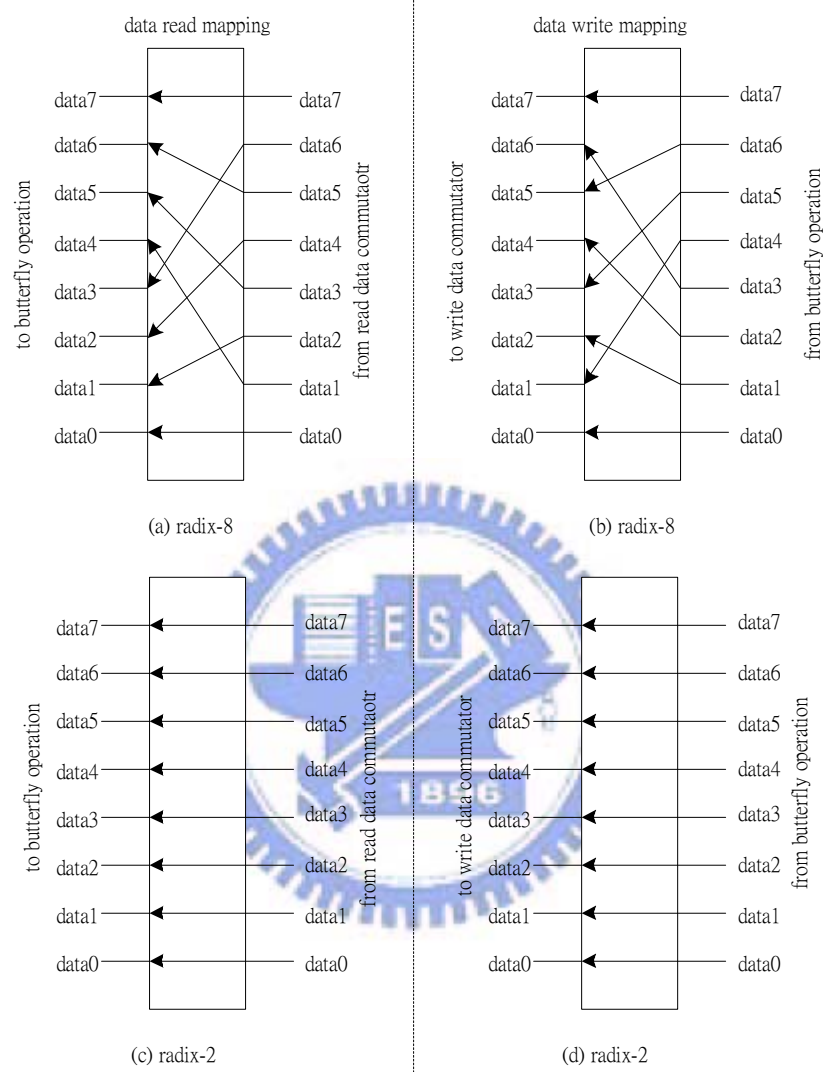


Figure 5.36 Read and write data mapping for mixed radix8+2 algorithm

- For mixed radix-8+4 case,

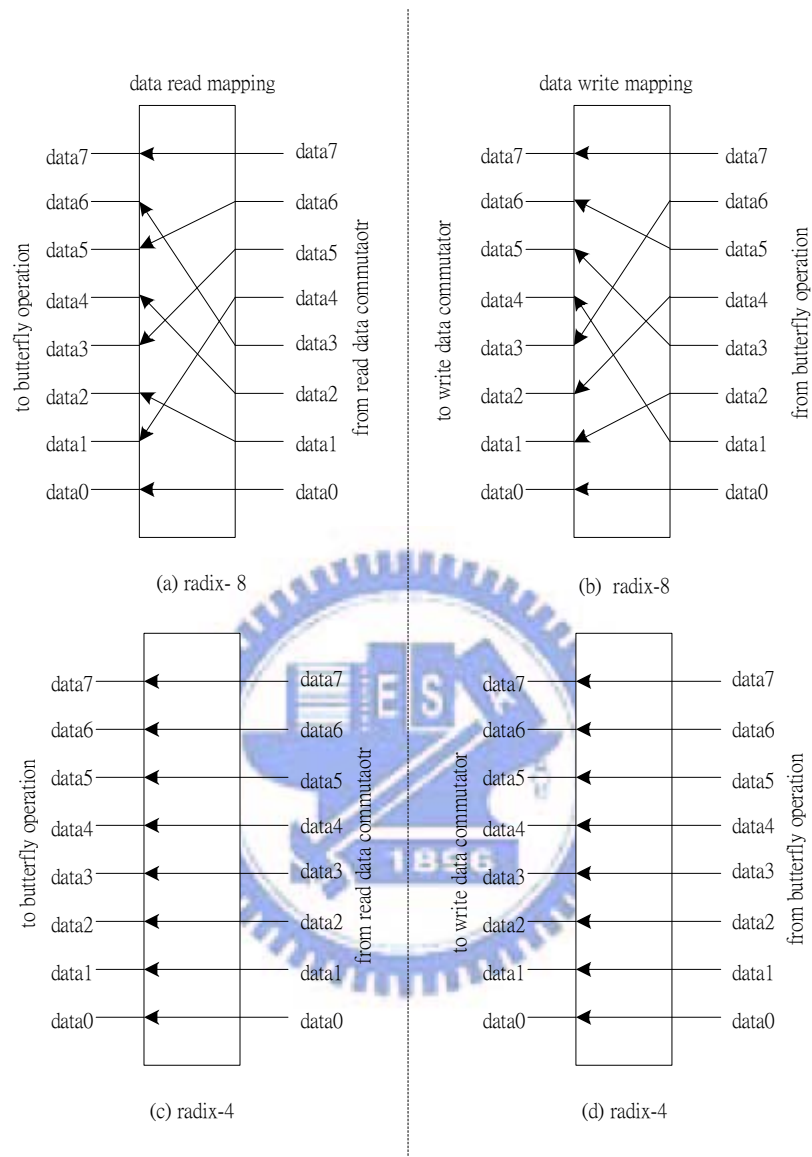


Figure 5.37 Read and write data mapping for mixed radix8+4 algorithm

## 5.10 Data Reorder Design

The DIF radix-8 FFT butterfly uses the in-place computation method. Thus, it has the natural input data order and the bit-reversed output data order. But the output data of our modified radix-8 operation is in natural order. We need to design the hardware implementation in order to obtain the bit-reversed output order. The way to accomplish this is to reverse the binary address bits. That is, if the address bits of the output data are  $x_6x_5x_4x_3x_2x_1x_0$ , then its reorder output becomes  $x_0x_1x_2x_3x_4x_5x_6$ .

After reordering the output data (represented by *data\_reorder*), we have to read out the necessary data from the SRAM. Figure 5.38 details our scheme to read out data from SRAM after performing the bit-reversed reordering.

The first step is to read the necessary column address of SRAM by using

$$\text{column address index} = \left\lfloor \frac{\text{data\_reorder}[11:0]}{8} \right\rfloor \quad (5.11)$$

Next, from the resulting column address in the first step, we use a multiplexer to select the SRAM bank index we need according to

$$\text{bank index} = (\text{data\_reorder}[2:0] + \text{data\_reorder}[5:3] + \text{data\_reorder}[8:6] + \text{data\_reorder}[11:9]) \bmod 8 \quad (5.12)$$

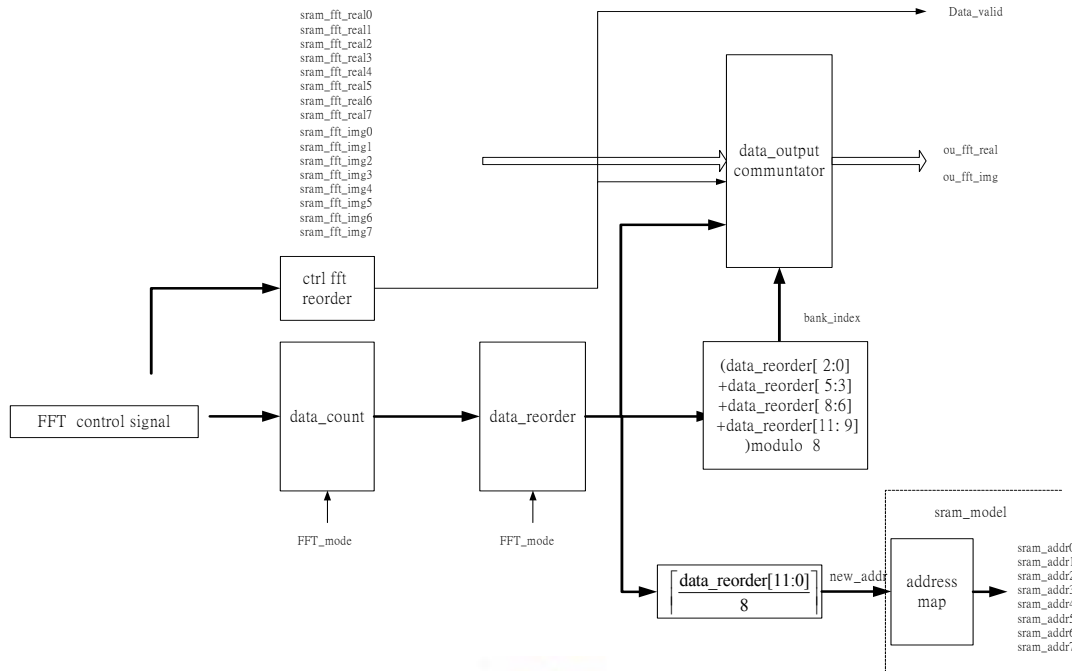


Figure 5.38 Data reorder design

## 5.11 Performance Evaluation

The comparison between our proposed design and the other designs is depicted in Table 5.21. It completes a 512-point FFT in 384 clock cycles. It takes 896 clock cycles to compute a 1024-point FFT and takes 1792 clock cycles for a 2048-point FFT. To compute a 4096-point FFT operation, it requires 4096 clock cycles. Under the constraints of the VDSL requirements, the maximum operation clock frequency for all points is 50 MHz.

Table 5.21 Performance comparisons

	radix(r)	number of clock cycles for $T_{FFT}$	512-point	1024-point	2048-point	4096-point	8192-point	
Hsin-Fu Lo[20]	radix-2	$\frac{N}{2} (\log_2 N)$	2306	5122	11266	24578	53250	power of 2
Byung S.Son[18]	radix-4	$\frac{N}{4} (\log_4 N)$	-	1280	-	6144	-	power of 4
Sang_Chul Moon[23]	radix-4	$\frac{N}{8} (\log_2 N - 2) + \frac{N}{4}$	576	1280	2816	6144	13312	power of 4 and not power of 4
proposed	radix-8	$\frac{N}{8} (\log_8 N) \times 2$	384	-	-	4096	-	power of 8 and not power of 8
	mixed radix	$\frac{N}{8} \lceil \log_8 N \rceil \times 2 + \frac{N}{8}$	-	896	1792	-	9216	