國立交通大學

電機資訊學院 電子與光電學程

碩士論文

一個兼具高速及小面積之

JPEG2000 平行區塊編碼器

Design of a High-Speed and Small-Area Pass-Parallel Context Formation Encoder for JPEG2000

研 究 生：張義孟

指導教授：陳紹基 博士

中 華 民 國 九 十 五 年 十一 月

一個兼具高速及小面積之

JPEG2000 平行區塊編碼器

Design of a High-Speed and Small-Area Pass-Parallel Context Formation Encoder for JPEG2000

研 究 生：張義孟　　Student：Yi-Meng Chang

指導教授：陳紹基 博士　　Advisor：Dr. Sau-Gee Chen

國 立 交 通 大 學

電機資訊學院 電子與光電學程

碩 士 論 文

A Thesis

Submitted to Degree Program of Electrical Engineering and Computer Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master of Science in

Electronics and Electro-Optical Engineering

November 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十一月

一個兼具高速及小面積之 JPEG2000 平行區塊編碼器

學生：張義孟　　指導教授:陳紹基 博士

國立交通大學電機資訊學院 電子與光電學程（研究所）碩士班

摘　　要

隨著網際網路和數位相機快速發展，靜態數位影像廣泛地使用於儲存與傳輸媒介。JPEG2000 是一種較新的靜態影像壓縮標準，它比目前使用的JPEG標準有更好的壓縮率，並且提供很多實用特徵。然而這些有用特徵相對的也需要更複雜的運算量與硬體資源。對於JPEG2000 編碼器而言，EBCOT Tier-1 的Context Formation Encoder是複雜度最高的模組。為了要改善它的效能，Pass-Parallel 架構是目前最有效率的方法之一。在本論文中，我們提出一個高效能且低功率的JPEG2000 Context Formation 方塊編碼器架構。我們藉由三種加速方法並使用管線化技巧來實現新的硬體架構。使用Dual Column Pass1 generation method，Pass-Parallel Column-Based區塊編碼器所需的context window與目前存在的技術比較，面積可降低 25%。使用Dual Column Pass1 generation method，all coding pass and significance change generation method 和 sample-parallel column-based coding method，可將整個系統架構的critical path降低。所提的新硬體架構可加快運算效率和減少編碼所需的硬體電路。最後我們利用Verilog硬體描述語言描述我們的架構並且使用Synopsys Design Compiler以TSMC CMOS 0.25μm製程合成後，pre-layout晶片面積大小為 40037μm^2，工作頻率可以到達 330 MHz，處理一張 2304×1728 的灰階影像時，編碼時間為 0.021 秒。

Design of a High-Speed and Small-Area Pass-Parallel Context Formation Encoder for JPEG2000

Student：Yi-Meng Chang　　Advisor：Dr. Sau-Gee Chen

Degree Program of Electrical Engineering Computer Science

National Chiao Tung University

ABSTRACT

As the prompt development of Internet and digital still camera (DSC), still image is broadly used as storage and transmission contents. JPEG2000 is a relatively new still image compression standard. It has better compression performance than conventional JPEG standard, and it provides many useful features. However, these features require more complex computations and hardware resources. The Context Formation Encoder of EBCOT tire-1 is of high complexity in a JPEG2000 encoder. To improve performance, the Pass-Parallel architecture is one of the most efficient methods. In this thesis, a high performance and low-power hardware architecture design of Context Formation encoder for JPEG2000 is proposed. The new hardware architecture is implemented by three speedup methods and pipeline technique. The area of context window of Pass-Parallel Column-Based context formation encoder is reduced by 25% using the proposed Dual Column Pass1 generation method in comparison with existing techniques. The critical path of overall system architecture is reduced employing dual column pass1 generation, all coding pass and significance change generation method and sample-parallel column-based coding method. The new architecture is proposed to improve the computation efficiency and reduce hardware area in pass coding operations. Finally, Our design is described with Verilog HDL code and synthesized by Synopsys Design Compiler using TSMC CMOS 0.25μm process. The pre-layout synthesized area is 40037 μm^2. In our simulation, the operation clock frequency can reach 330 MHz. With this clock frequency, it needs 0.021 second to encode an image with 2304 x 1728 image size.

誌　　謝

能夠順利完成這篇論文。要感謝的人很多，首先要感謝我的指導教授陳紹基博士，在研究的過程中，陳教授提供我很多寶貴的意見，讓我能夠確定論文方向，在老師嚴格訓練與耐心指導下，使我學習到如何解決問題，同時讓我在課業與研究上獲益良多。

另外要感謝同學、同事，以及朋友們鼓勵，在課業及豐富經驗業上給予我的幫助，在寫論文這段時間中，大家互相討論，精益求精，使我產生很多創意，為我日後的研究奠定深厚基礎。

最後，要感謝我的家人。對於妻子玉芳，長久以來的支持、體諒，以及一肩挑起照顧兒子宸安的重擔，讓我無後顧之憂的專注於工作與求學。

張 義 孟

於交大

民國九十五年十一月

Contents

List of Tables

List of Figures

CHAPTER 1 INTRODUCTION

With the development of information technology and application, image compression becomes more and more important. To address this need in the specific domain of still image coding, a new standard, so-called “JPEG2000”, is developed by the International Standards Organization (ISO) and International Electrotechnical Commission (IEC). The details of JPEG200 standard are described in [1] to [4]. Compared to the traditional JPEG standard, JPEG2000 not only has better compression performance than JPEG standard does, but also provides more features than JPEG. In this chapter, JPEG2000 encoder overview and research motivation are described in section 1.1 and section 1.2, respectively. Thesis organization is presented in section 1.3.

1.1 JPEG2000 Encoder Overview

JPEG2000 is the newest standard for still image compression. It provides not only high image quality performance but also new functionality. The features of this standard are as follows :

1. Lossless and lossy compression
2. Better low bit-rate compression performance than JPEG.
3. Progressive transmission by pixel accuracy and resolution
4. Bi-level compression
5. Random codestream access and processing
6. Better error resilience
7. Region-of-Interest Coding (ROI)

The system structure of JPEG2000 encoder is depicted in Figure 1-1. The JPEG2000 coding system can be separated into four main parts: forward discrete wavelet transform (FDWT), quantization, EBCOT Tier-1 Encoder (Embedded Block Coding with Optimized Truncation) and EBCOT Tier-2 Encoder.

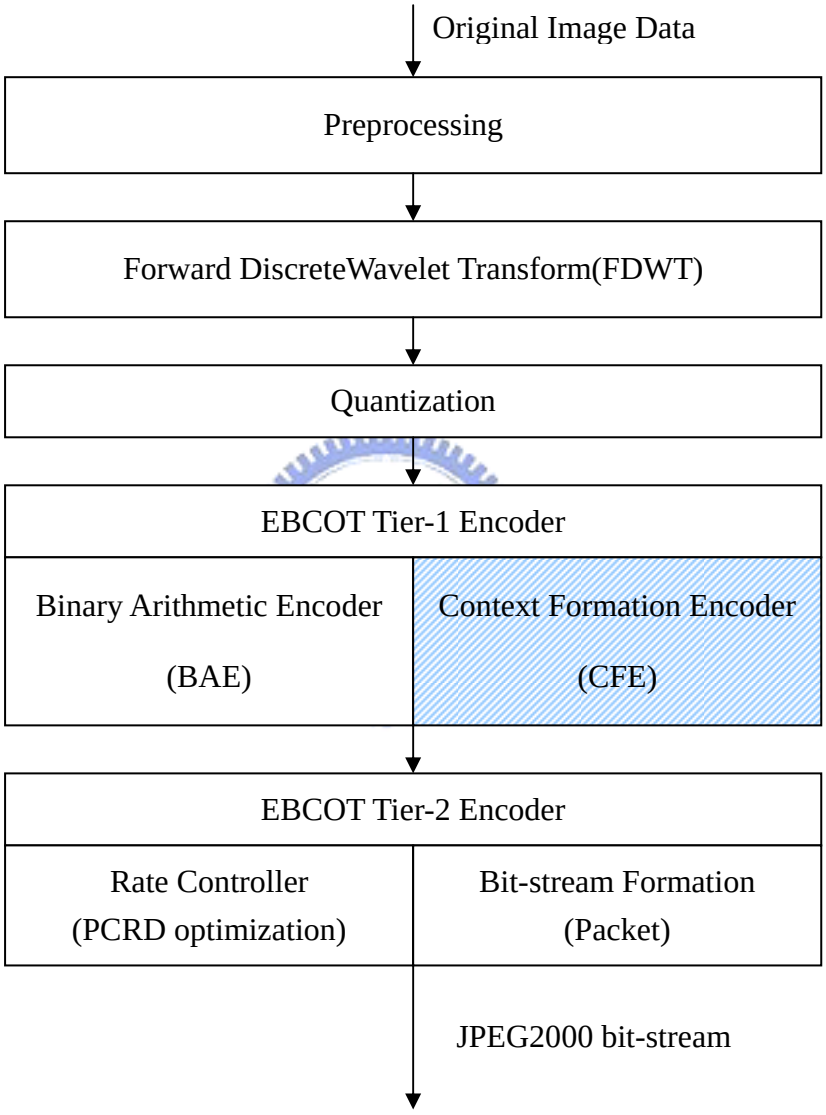


Figure 1-1. The system structure of JPEG2000 encoder

The preprocessing part include DC level shift and forward color transform. The DC level shift converted source image data with unsigned format to two's complement format. The forward color transform converted RGB image to YUV or YCbCr format in order to reduce the correlations among RGB colors.

The transform part uses Forward Discrete Wavelet Transform (FDWT) to convert source image data into several sub-bands and transform image from spatial domain to frequency domain. By using FDWT instead of DCT technique to process images, JPEG2000 not only overcomes the block effect of JPEG standard, but also achieves higher compression ratio with subjective image quality.

The quantization part is performed for lossy compression. After FDWT and quantization , the FDWT coefficients of each wavelet sub-band are partitioned into several non-overlapped code blocks (typically 64x64 or 32x32 in dimension), which are independently coded by EBCOT.

As shown in Figure 1-1, EBCOT is composed of EBCOT Tier-1 encoder and EBCOT Tier-2 encoder. EBCOT Tier-1 encoder is a context-based adaptive arithmetic encoder and compresses each code block into a respective sub-bit-stream by applying the context-based adaptive arithmetic coding technique. Code blocks are independently coded by EBCOT Tier-1 encoder. The wavelet coefficients of a code-block are coded bit-plane by bit-plane from most significant bit-plane to less significant bit-plane instead of coefficient by coefficient. EBCOT Tier-1 encoder is composed of Context Formation Encoder (CFE) and Binary Arithmetic Encoder (BAE). Before executing CFE, every coefficient in code blocks is converted into sign-magnitude representation. Context Formation Encoder scans all bits in a bit plane of a code-block in stripe-oriented scanning pattern, and generates corresponding contexts for each bit by checking the status of the neighborhood bits. CFE sends context label (CX) and decision (D) to BAE for generating bit stream. The Binary Arithmetic Encoder (BAE) then encodes each bit according to the adaptively estimated probabilities from its contexts. The outputs of Binary Arithmetic Encoder are sub-bit-streams of each compressed code-block data.

EBCOT Tier-2 encoder is composed of Rate Controller and Bit-stream Formation. The rate controller is executed by Post-Compression Rate-Distortion Optimization algorithm (PCRD Optimization). After getting all the compressed sub-bit-streams from EBCOT Tier-1 encoder, the rate controller of EBCOT Tier-2 encoder takes charge of optimized rate of bit-streams under a given bit-rate and sends bit-streams to Bit-stream Formation. The Bit-stream Formation collects bit-streams by packet.

1.2 Research Motivation

Although JPEG2000 has good SNR performance and new functionality, its computational complexity is much higher than JPEG. In order to achieve high performance features and various applications, we must implement JPEG2000 encoder with low cost and high throughput rate for still image. Before design JPEG2000, we first analyze computation time of components of JPEG2000 encoder. Table1-1 [7] shows that time consumed by each component of JPEG2000 encoder represents the ratio of total computation time. From Table1-1, Context Formation Encoder (CFE) of JPEG2000 encoder system occupies highest computation time.

Table 1-1. Profile of Run Time Percentage (%) of JPEG2000

Operation	Gray Scale Image		Color Image (RGB)	
	Lossless	Lossy	Lossless	Lossy
Color transform	N.A.	N.A.	0.91	14.12
DWT	10.81	26.38	11.9	23.97
Quantization	N.A.	6.42	N.A.	5.04
EBCOT Tier 1	71.63	52.26	69.29	43.85
CF Encoder (Context Formation)	51.88	37.91	49.96	31.79
BA Encoder (Binary Arithmetic)	19.75	14.35	19.33	12.06
EBCOT Tier 2	17.56	14.95	17.9	13.01
Layer Formation	10	9.52	9.94	7.95
Marker Insertion	7.56	5.43	7.96	5.06

The reason of the highest computation time of the context formation encoder is that the operations are bit-level processing. N samples need to be processed in the context formation encoder of EBCOT tier-1 for an N-bit word. Some designs of context formation encoder of EBCOT tier-1 in JPEG2000 have been announced in [5] to [17]. In this thesis, we focus on the analysis of context formation encoder in EBCOT Tier-1 for JPEG2000 and propose a new architecture to improve the computation efficiency and reduce hardware area.

1.3 Thesis Organization

The organization of this thesis is as follows:

Chapter 1:

We introduce overview and features of JPEG2000 and research motivation. Four main function block of JPEG2000 encoder is described.

Chapter 2:

We will detail structure of basic EBCOT Tier-1 Context Formation Encoder.

Chapter 3:

We will detail architecture of pass-parallel EBCOT Tier-1 Context Formation Encoder.

Chapter 4:

High-speed pass-parallel methods proposed are introduced. We propose three new methods to improve the computation efficiency and reduce hardware area.

Chapter 5:

In this chapter, we introduce the overall system architecture of the proposed fast pass-parallel context-formation (CF) encoder, detailed internal signals of the proposed context formation encoder, analysis of system critical path, design flow, verification and experiment results.

Chapter 6:

A brief conclusion is described in this chapter.

CHAPTER 2 ANALYSIS OF BASIC EBCOT TIER-1 CONTEXT FORMATION ENCODER

2.1 Coding Scan Order and Five Coding State Variables

The coding scan order is based on stripe-oriented scanning pattern in every bit plane. This pattern is described in section 2.1.1. Five coding state memories are needed in coding operations. The meaning of five coding state memories is described in section 2.2.2.

2.1.1 Coding Scan Order

After the DWT (Discrete Wavelet Transform) and quantization, each sub-band is partitioned into code block (64x64 or 32x32 in dimension). As shown in Figure 2-1, a code-block is composed of many bit planes. The coding scan order of each code block is bit plane by bit plane, from the most significant bit plane (MSB) with at least a non-zero element to the least significant bit plane (LSB).

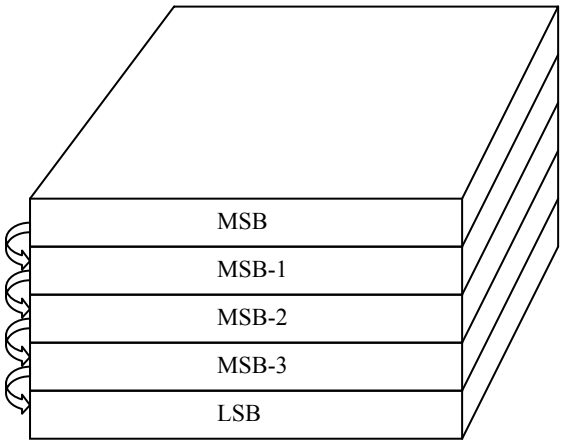


Figure 2-1. Bit planes in a code block and scan order

As shown in Figure 2-2, a bit plane is composed of many stripes. In every bit plane, the coding scan order is stripe by stripe from top to bottom.

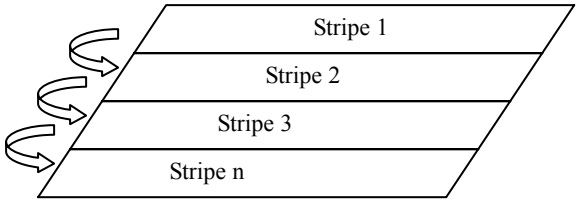


Figure 2-2. Stripes in a bit plane and scan order

As shown in Figure 2-3, a stripe is composed of many columns. And a column is composed of four samples. In every stripe, the coding scan order is column by column from left to right. In every column, the coding scan order is sample by sample from top to bottom. Each sample in each bit plane is coded by bit plane coder (BPC) and sent to the binary arithmetic coder (BAC) with a decision and context. The context decision (D,CX) of each sample would be decided by five coding state variables, four coding operation, and three coding passes.

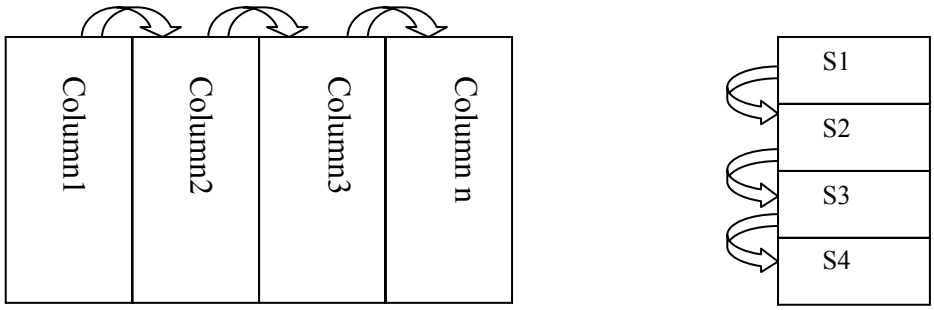


Figure 2-3. (a) Columns in a stripe and scan order

(b) Samples in a column and scan order

As shown in Figure 2-4, all quantized wavelet coefficients of each code block are expressed in sign-magnitude representation (in 1's complement) and divided into one sign bit plane and several magnitude bit planes.

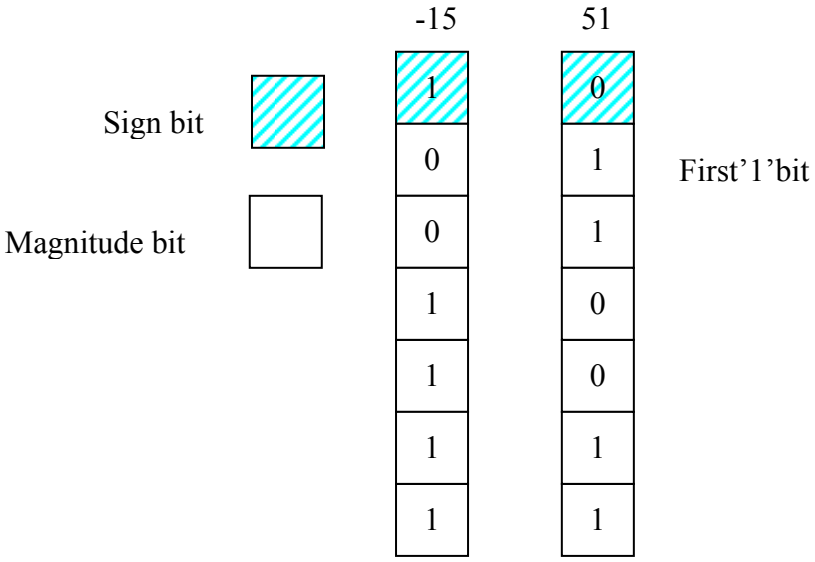


Figure 2-4. Wavelet coefficients in sign-magnitude representation

2.1.2 Five Coding State Variables

There are five state variables for block coding in context formation module. They are the sign state variable (X), magnitude state variable (V), significance state variable (σ), the refinement state variable (R), and already-coded state variable (η).

Sign state variable (X)

The sign bit of every sample is recorded in the sign states. The sign state of every sample is the same across all bit planes. From Figure 2-4, a one bit in sign bit represents a negative number and a zero bit in sign bit represents a positive number. In the beginning of a coding process for each code block, the sign bit of every sample is loaded to this state variable. Since every wavelet coefficient in code blocks is converted into sign-magnitude representation, the sign state is the same when coding every coefficient across all bit planes. All the sign state bits are kept across all bit planes.

Magnitude state variable (V)

The magnitude state variable is used to record magnitude bit of every coefficient in the current coding bit plane. The magnitude state is different in every bit plane for a coefficient generated from DWT. All magnitude state bits are loaded at the start of coding each bit plane.

Significance state variable (σ)

If a sample in context window meet significance change in pass1 or pass3 of current bit plane, the significance state variable records significance change of a sample. In coding operation from MSB to LSB, the first magnitude bit of a coefficient is called significance change (‘1’), and is called insignificance (‘0’) before the first magnitude bit of a coefficient appears, as illustrated in Figure 2-4. The significance state variable of every code block is all insignificance (‘0’) in setting initial value. The significance state variable changes from zero to one when the magnitude bit of the coefficient is the first ‘1’ in pass1 or pass3 coding operations. All the significance state bits are reset at the start of coding each code block and changed according to the coding operation of pass1 or pass3.

Refinement state variable (R)

In the beginning of the coding process for each code block, refinement states of all samples are set to ‘0’. If a sample becomes significant in previous bit planes, it should be coded firstly in magnitude refinement coding in this bit plane. And refinement state bit is set one after a sample is coded by magnitude refinement coding at first time in pass2 coding. All refinement state bits are reset at the start of coding each code block and changed according to the coding operation of pass2.

Coded state variable (η)

When a sample is coded in significance propagation pass (Pass1) or magnitude refinement pass (Pass 2), the coded state bit of a sample change from zero to one. The coded state indicates whether or not a sample has already been coded in a previous coding pass of the same bit plane. After the clean up pass (Pass 3), the coded state bits are all reset to zero. All coded state bits are reset after pass3 of each bit plane and changed according to whether or not a sample has already been coded in current bit plane.

2.2 Four Coding Operations

The context label (CX) and decision bit (D) of each sample is generated according to the states of its neighbors in different coding operations and current bit value. As shown in Table 2-1, the eight neighbor samples of current sample X are classified into three groups: vertical (V0, V1), horizontal (H0, H1), and diagonal (D0, D1, D2, D3). The CXD pair (context label and decision bit) of the sample X in context window is generated by zero coding (ZC), sign coding (SC), magnitude refinement coding (MRC), and run-length coding (RLC).

The four coding operations for generating context label (CX) and decision bit (D) are introduced below.

Table 2-1. Neighbor significance states for context label generation

D0	V0	D1
H0	X	H1
D2	V1	D3

Zero Coding (ZC)

Sample X in context window is insignificant in previous bit planes. And sample X of becoming significance in current bit plane will be coded by zero coding. The context label of sample X is generated by context label 0-8 shown in Table 2-2. The decision bit D of sample X is equal to the magnitude bit of the current sample in the bit plane. Zero coding is used in significant propagation pass and clean up pass. In Table 2-2, neighbor samples of sample X are classified into 9 groups, corresponding to 9 contexts. SUMH represents the sum of significant horizontal neighbors H0 and H1, SUMV represents the sum of significant vertical neighbors V0 and V1, and SUMD represents the sum of diagonal neighbor samples D0, D1, D2, and D3, and SUMHV represents the sum of significant horizontal neighbors H0, H1, V0, V1.

Table 2-2. C錯誤!ontext table for zero Coding

LL and LH sub-band			HL sub-band			HH sub-band		CX
SUMH	SUMV	SUMD	SUMH	SUMV	SUMD	SUMHV	SUMD	
2	X	X	X	2	X	X	≥3	8
1	≥1	X	≥1	1	X	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5
0	2	X	2	0	X	1	1	4
0	1	X	1	0	X	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

Sign Coding (SC)

As shown in Figure 2-5, if the first magnitude bit of sample W is one in significant propagation pass or clean up pass, it will be coded by sign coding.

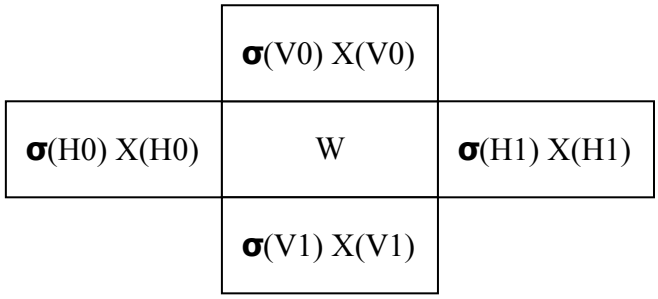


Figure 2-5. Neighbor significance and sign states for sign coding

In sign coding, the significance state and sign state of the vertical and horizontal neighbors are classified into three contribution reference marks (0, 1, -1). Mark '0' indicates that both vertical (or horizontal) neighbors are insignificant, or both vertical (or horizontal) neighbors are significant but have opposite signs. Mark '1' indicates that one or both vertical (or horizontal) neighbors are significant, and sign of significance state is positive. Mark '-1' indicates that that one or both vertical (or horizontal) neighbors are significant, and sign of significance state is negative.

As shown in Table 2-3, horizontal contribution of sample H0, H1 is obtained according to the significance state and sign state of sample H0, H1.

Table 2-3. Table of horizontal contribution

σ(H0)	**σ**(H1)	X(H0)	X(H1)	SUMH
0	0	X	X	0
0	1	X	0	1
0	1	X	1	-1
1	0	0	X	1
1	0	1	X	-1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	-1

As shown in Table 2-4, vertical contribution of sample V0, V1 is obtained according to the significance state and sign state of sample V0, V1.

Table 2-4. Table of vertical contribution

σ(V0)	**σ**(V1)	X(V0)	X(V1)	SUMV
0	0	X	X	0
0	1	X	0	1
0	1	X	1	-1
1	0	0	X	1
1	0	1	X	-1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	-1

As shown in Table 2-5, context label is generated by combination of the contribution of horizontal samples and the vertical samples. And the decision bit D is generated by exclusive-or the XOR bit and the sign bit of coded sample.

$$D = \text{sign bit} \oplus \text{XOR bit}$$

Table 2-5. Context table for sign coding

SUMH	SUMV	Context Label	XOR bit
1	1	13	0
1	0	12	0
1	-1	11	0
0	1	10	0
0	0	9	0
0	-1	10	1
-1	1	11	1
-1	0	12	1
-1	-1	13	1

Magnitude Refinement Coding (MRC)

A sample that is significant in the previous bit plane will be coded by magnitude refinement coding. If the sample is the first time coded by MRC, the context label CX is generated by the significance states of its eight neighbors. Otherwise the context label is fixed to be 16. Table 2-4 shows the three contexts for magnitude refinement coding. The decision bit is the magnitude bit of the current sample in the bit plane. MRC is only used in magnitude refinement pass.

Table 2-6. Context table for magnitude refinement coding

R(refinement)	σ (significance)	SUMH+SUMV+SUMD	Context label
1	1	X	16
0	1	≥1	15
0	1	0	14

Run-Length Coding (RLC)

If four contiguous samples in a column are all pass3, enter run-length coding mode. In run-length coding mode, there are two as follows sub-coding mode.

1. Run-length code 0 (RLC 0) : As shown in Table 2-7, if the magnitude bit of four contiguous samples in a column are all zero and their pass type are all pass3, the context label is coded to 17 and the decision bit is set to 0.
2. Run-length code 1 (RLC 1) : As shown in Table 2-8, if the magnitude bit of any sample in four contiguous samples is one and their pass type are all pass3, the context label is coded to 17 and the decision bit is set to 1. After RLC 1, two uniform coding are generated with context-decision pairs, (0,18)(0,18) or (0,18)(1,18) or (1,18)(0,18) or (1,18)(1,18). The decision values are generated according to the location of magnitude bit in a column. After uniform coding, sign coding of the first magnitude bit is generated. The rest samples in a column are coded by zero coding or zero coding and sign coding after the first magnitude bit sample. If the magnitude bit of rest sample is zero, zero coding is used. If the magnitude bit of rest sample is one, zero coding and sign coding are used.

Table 2-7. Context table for run-length code 0

Coding	column pass x1,x2,x3,x4	4 samples value x1,x2,x3,x4	D	CX
RLC0	P3,P3,P3,P3	0000	0	17

Table 2-8. Context table for run-length code 1 with uniform coding

Coding	column pass x1,x2,x3,x4	Column value x1,x2,x3,x4	D	CX	UC1 (D,CX)	UC0 (D,CX)
RLC1	P3,P3,P3,P3	1,X,X,X	1	17	0,18	0,18
RLC1	P3,P3,P3,P3	0,1,X,X	1	17	0,18	1,18
RLC1	P3,P3,P3,P3	0,0,1,X	1	17	1,18	0,18
RLC1	P3,P3,P3,P3	0,0,0,1	1	17	1,18	1,18

2.3 Three Coding Passes

There are three coding passes in each bit plane. They are significance propagation pass (SPP, Pass 1, P1), magnitude refinement pass (MRP, Pass 2, P2), and clean up pass (CUP, Pass 3, P3).

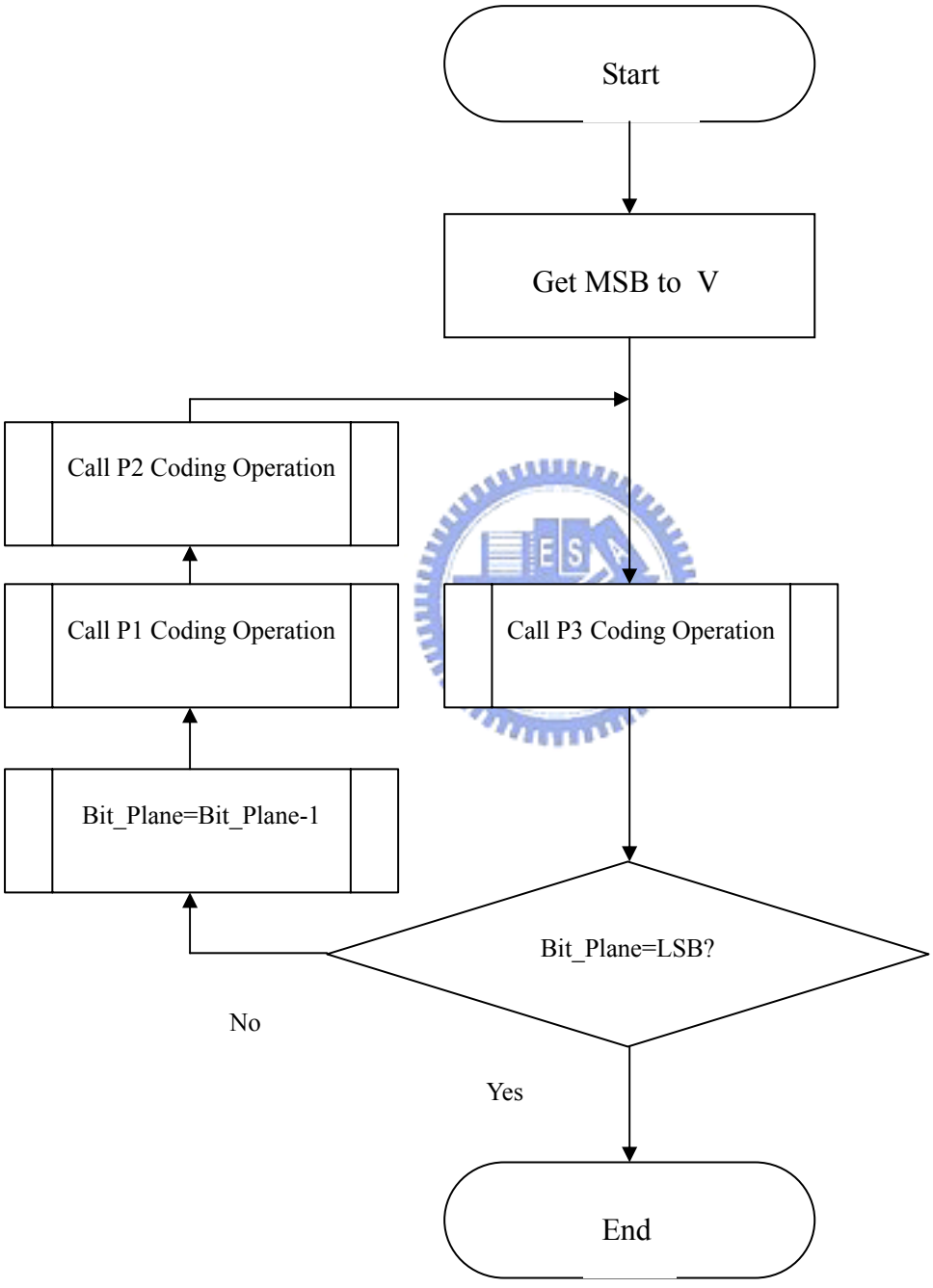


Figure 2-6. Order of coding operations of three passes

As shown in Figure 2-6, the pass3 coding operation is used in MSB. The MSB represents the first bit plane with at least one non-zero element. The LSB represents the last bit-plane. The Bit_Plane represents current bit plane. The coding operations of three passes are pass1 coding operation, pass2 coding operation and pass3 coding operation sequentially in (MSB-1) to LSB.

As shown in Table 2-1, Sample X in context window is coded in only one of the coding operations of three passes. As shown in Figure 2-7, the **σ**(X) represents significance sate of sample X in context window and the N(X) represents neighbor significance states of sample X in context window. When **σ**(X) is one, current coded sample can enter pass2(P2) coding operation. When **σ**(X) and N(X) are all zero, current coded sample can enter pass3(P3) coding operation. When **σ**(X) is zero and N(X) is one, current coded sample can enter pass1(P1) coding operation.

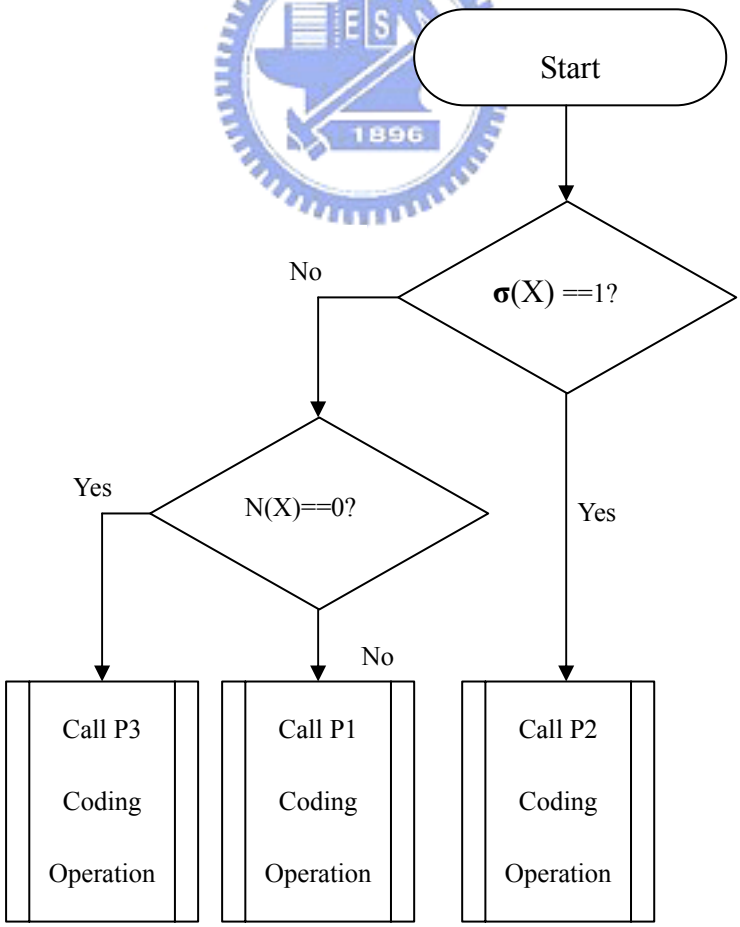


Figure 2-7. Checking of pass coding operation of sample X in context window

Significance propagation pass (SPP, Pass1 or P1)

As shown in Table 2-1, the neighbor significance sates of sample X are σ(V0), σ(V1), σ(H0), σ(H1), σ(D0), σ(D1), σ(D2), σ(D3) and the significance state of sample X is σ(X). The SUM represents the sum of neighbor significance states of sample X. When σ(X) is zero and the SUM is at least one, sample X enters pass1 coding operation. The simple hardware description language is expressed as follows:

```
SUM=σ(V0)| σ(V1)| σ(H0)| σ(H1)| σ(D0)| σ(D1)| σ(D2)| σ(D3)
if (σ(X)    == 0) & (SUM == 1)
            P1    = 1'b1;
else
            P1    = 1'b0;
```

Besides, if the magnitude bit V(X) of sample X is zero or one, the sample X is executed by zero-coding (ZC), and the coded state (η) is set to '1' immediately. Next, if the first magnitude bit V(X) of the sample X is one from MSB to LSB bit plane, the sample X is executed by sign-coding (SC), and the significance state (σ) is set to '1' immediately. The simple hardware description language is expressed as follows:

```
if (P1(X)    == 1)
    begin
          Zero-Coding;
          assign η(X)=1;
     end
    else if (P1(X)    == 1 & V(X) == 1)
    begin
                        Sign-Coding;
                        assign σ(X)=1;
     end
```

Magnitude refinement pass (MRP, Pass2 or P2)

As shown in Table 2-1, The η(X) represents coded state of sample X in current bit plane. The R(X) represents refinement state of sample X in previous bit plane. The σ'(X) represents significance state of sample X in previous bit plane. When

$\sigma'(X)$ is one and the $\eta(X)$ is zero, sample X of current bit plane enters pass2 coding operation. The simple hardware description language is expressed as follows:

```
if (σ'(X)   == 1 & η(X) ==0)
      P2    = 1'b1;
else
            P2    = 1'b0;
      end
if (P2(X)   == 1 )
   begin
      if (R(X) == 1)
      Magnitude-Refine Coding(output CXD[4:0] = 5'd16);
      else
      Magnitude-Refine Coding(output CXD[4:0] = 5'd14 or 5'd15);
    end
```

Cleanup pass (CUP, Pass3 or P3)

From the above discussion, we obtain pass3 information by pass1 and pass2 information. When the $\sigma(X)$, the SUM and the $\eta(X)$ are all zero, sample X enters pass3 coding operation. The simple hardware description language is expressed as follows:

```
SUM=σ(V0)| σ(V1)| σ(H0)| σ(H1)| σ(D0)| σ(D1)| σ(D2)| σ(D3)
if (σ(X)   == 0) & (SUM == 0) & (η(X) ==0)
            P3    = 1'b1;
else
            P3    = 1'b0;
if (P3(X)   == 1)
    begin
            Zero-Coding;
     end
    else if (P3(X)   == 1 & V(X) == 1)
    begin
                        Sign-Coding;
                        assign σ(X)=1;
     end
```

As shown in Figure 2-8, when the significance states, neighbor significance states and magnitude bits of all four consecutive samples in a column are all zero, then sample X1 to X4 enter pass3 coding operation and run length code 0 is executed.

A0	B0	C0	Stripe n-1
A1	X1	C1	
A2	X2	C2	
A3	X3	C3	Stripe n
A4	X4	C4	

Figure 2-8. Neighbor significance states of samples X1 to X4 for Pass3

Example: (column-based operation)

The simple hardware description language is expressed as follows:

SUM1= σ(A0)| σ(A1)| σ(A2)| σ(A3)| σ(A4)| σ(C0)| σ(C1)| σ(C2)| σ(C3)| σ(C4)|σ(B0)

SUM2= σ(X1)| σ(X2)| σ(X3)| σ(X4)

SUM3= η(X1)| η(X2)| η(X3)| η(X4)

SUM4=V(X1)| V(X2)| V(X3)| V(X4)

if (SUM1 == 0) & (SUM2 == 0) & (SUM3 ==0) & (SUM4 ==0)

Run-length code 0(output CXD[4:0] = 5'd17,decision= 1'b0);

When the significance states and neighbor significance states of all four consecutive samples in a column are all zero and their magnitude bits are not all zero, then sample X1 to X4 enter pass3 coding operation and run length code 1 is executed.

The simple hardware description language is expressed as follows:

if (SUM1 == 0) & (SUM2 == 0) & (SUM3 ==0) & (SUM4 ==1) begin

Run-length code 1(output CXD[4:0] = 5'd17,decision= 1'b1);

Uniform coding; end

CHAPTER 3 ANALYSIS OF PASS-PARALLEL EBCOT TIER-1 CONTEXT FORMATION ENCODER

In this chapter, we will analyze the architecture of Pass-Parallel EBCOT Tier-1 Context Formation encoder. The **Pass-Parallel** context formation structure is that three coding pass operations are executed in the same bit plane in parallel to achieve higher computation performance and less memory requirement in comparison with the basic EBCOT Tier-1 Context Formation encoder introduced in previous chapter.

3.1 The Core of Pass-Parallel Context Formation Architecture

According to the basic Context Formation (CF) encoder described in previous chapter, samples in a bit plane are coded pass by pass. The conventional three coding pass operations caused lots of clock cycles wasted. At present, the Pass-Parallel method is proposed to improve system performance and reduce memory requirement.

The Pass-Parallel context formation (CF) is to process three coding passes of the same bit plane in parallel. There are some issues to be solved. First, due to processing three coding pass operations concurrently, the samples belong to Pass 3 may become significant earlier than the two prior coding pass operations and this situation will mistake the following pass coding procedures for samples which belong to Pass 1 or Pass 2. Second, if the sample that currently coded belongs to Pass1, Pass 2 or Pass 3, the neighbor significances of the coded sample is how to be predicted.

To solve above issues, we will discuss the form of the core of Pass-Parallel Context Formation architecture. The core of Pass-Parallel Context Formation architecture is context-window shift register array of three coding pass operations. Up to now, two context-window shift register arrays proposed in papers are discussed in this section. The first one [8], [10], [13] is that context-window shift register array consists of two parts of 5X3 shift register array. The second one [14] is that context-window shift register array is 5X5 pipeline shift register array. The main purpose of context-window shift register array is to store all coded samples and neighbor status of all coded samples. Moreover, the “stripe causal” mode and column-based operation are also adopted.

As shown in Figure 3-1, the context window shift register array of Pass-Parallel Context Formation Encoder is composed of two parts. The Part1 is formed by register B, C and D. The Part2 is formed by register A, B and C. To solve the first issue described in this section, the coding pass operation for Pass 3 is delayed by one stripe column to eliminate the effect between Pass3 and other two passes. Subsequently, to eliminate the dependence of coding pass operations on the next stripe, the stripe cause mode is also adopted. The first part is responsible to process all samples that are coded by pass1 and pass2 in register C. The second part is responsible to process the rest samples coded by pass3 in register C of the first part to shift left one column to be coded in register B of the second part.

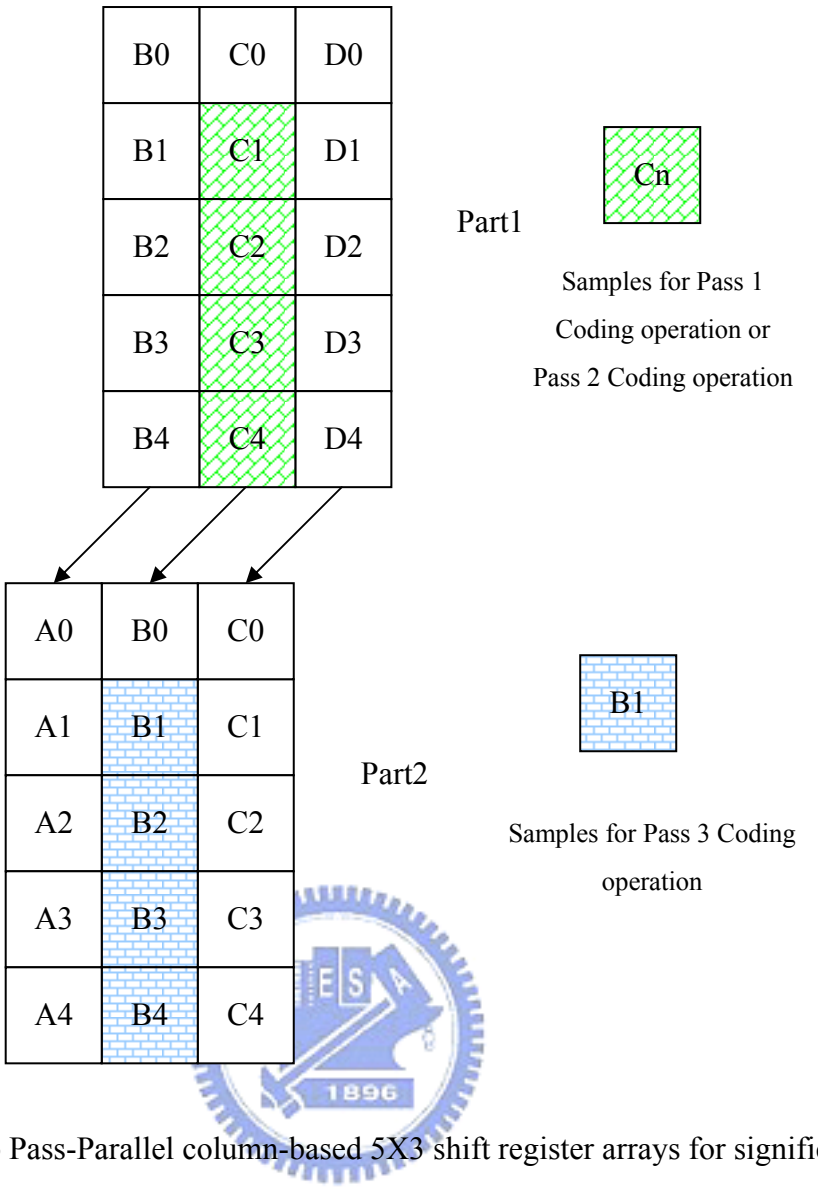


Figure 3-1. Two Pass-Parallel column-based 5X3 shift register arrays for significance states S1,S3 and sign state X0

The coding pass operations [10] of two parts of context-windows can be separated into three steps. The first step is that samples in register C belong to pass1 or pass2 and are coded by pass1 or pass2. The second step is that consecutive four samples in register D are all pass3 and are coded by run-length code. The third step is that consecutive four samples in register D are not all pass3 and are coded by zero-coding or sign-coding of pass3.

As shown in Figure 3-2, two context windows are adopted. The context window 1 is formed by register A, B and C. The context window 2 is formed by register C, D and E. These two context windows are implemented as 5 x 5 shift register array. We use context window 1 to judge whether the samples are coded in Pass1 or Pass2 or not.

We use context window 2 to judge whether the samples are coded in Pass3 or not. To solve the first issue described in this section, the coding pass operation for Pass 3 is delayed by two stripe columns to eliminate the effect between Pass3 and other two passes. Subsequently, to eliminate the dependence of coding pass operations on the next stripe, the stripe cause mode is also adopted. In stripe cause mode, the samples in the next stripe are considered to be insignificant. Take Figure 3-2 for illustration, the significance states of A5, B5, and C5 are considered to be zeros when coding sample B4. In context window 1, circuit of deciding pass1 or pass2 is executed and sample B1 to B4 of pass1 or pass2 is coded by pass1 coding operation or pass2 coding operation in a clock cycle of register B. In context window 2, circuit of deciding pass3 or pass3 run-length is executed and sample D1 to D4 of pass3 or pass3 run-length is coded by pass3 coding operation or pass3 run-length coding operation in a clock cycle of register D.

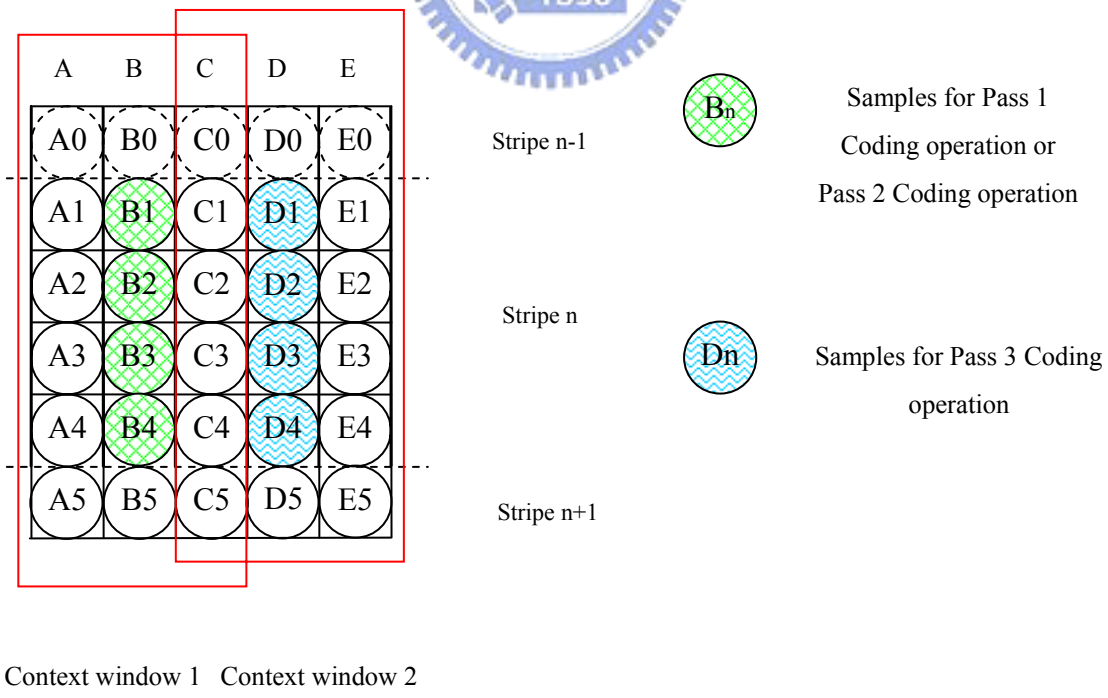


Figure 3-2. Pass-Parallel column-based shift register array (6x5)

To solve the second issue described in this section, the author [8][10][13] proposed some methods to record neighbor significance state of all four consecutive coded samples in a column and their significance state. In papers, two state variables are described, significance state 1 (S1) and significance state 3 (S3), instead of original significance state variable (σ), refinement state variable (R), and coded state variable (η) described in the previous chapter. If a sample belongs to Pass 1 and becomes significant, the state variable S1 is set to '1'. If a sample belongs to Pass 3 and becomes significant, the state variable S3 is set to '1'. Besides, both significance state variables S1 and S3 are set to '1' immediately after the sample has been coded in magnitude refinement coding of pass2.

3.2 Sample -Based Operation for Pass-Parallel

Four coding state variables are used in pass coding operations. The significance state variable S1 recorded pass1 significance change in current bit plane or pass1 significance state in previous bit plane. The significance state variable S3 recorded pass3 significance change in current bit plane or pass3 significance state in previous bit plane. The sign state variable X0 recorded sign bits of each coefficient in code block. The magnitude state variable V0 recorded magnitude bits in bit planes. As shown in Figure 3-3, a 3X5 shift register array is needed for S1, S3, X0. As shown in Figure 3-4, a 1X5 shift register array is needed for V0. If a sample becomes significant after pass1 coding operation, the significance state variable S1 is set to '1'. If a sample becomes significant after pass3 coding operation, the significance state variable S3 is set to '1'. After the

current sample is coded by pass2 coding operation, both the significance state variable S1 and the significance state variable S3 are set to'1' immediately. In other words, if any of the significance state variable S1 or the significance state variable S3 is '0', it means that the sample has not been coded by pass2 coding operation. Hence, the original refinement state variable can be replaced by

$$R = S1 \text{ XOR } S3$$

As shown in Figure 3-3, If the equation S1 | S3 of register B1 in context window 1 is zero and the any neighbor significance states of the register B1 is one, register B1 belongs to Pass 1. If register B1 in context window 1 belongs to Pass 1, the significance states of visited register C0, C1, C2, B0, A0 are equal to pass1 significance change in current bit-plane or pass1 significance state in the previous bit plane. Since register B2, A1, A2 that have not been visited may become significant in pass 3 coding operation of the previous bit plane, the significance states of not visited register B2, A1, A2 are equal to pass1 significance state or pass3 significance state in the previous bit plane.

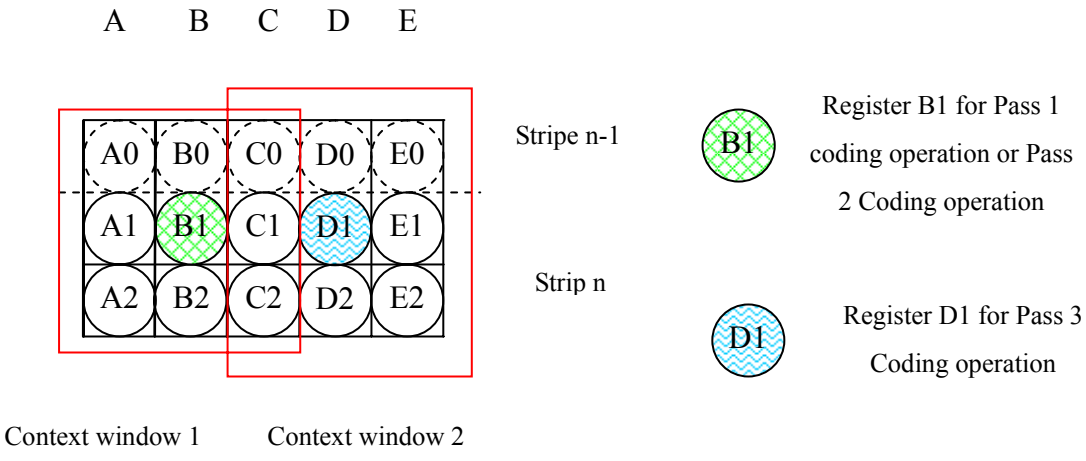


Figure 3-3. Pass-Parallel sample-based shift register array (3x5) for significance state S1, S3 and sign state X0

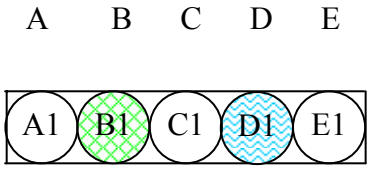


Figure 3-4. Pass-Parallel sample-based 1X5 shift register for magnitude state V0

Neighbor significance states of register B1 can be predicted by Table 3-1 for pass1.

Table 3-1. The neighbor significance states of register B1 for pass1

Register C	Register B	Register A
S1 (Register C0)	S1 (Register B0)	S1(S1 | S3) (Register A0)
S1 (Register C1)	S1 | S3 (Register B1)	S1 | S3 (Register A1)
S1 (Register C2)	S1 | S3 (Register B2)	S1 | S3 (Register A2)

If the equation S1 | S3 of register B1 in context window 1 is one and the any neighbor significance states of the register B1 do not care, Register B1 belongs to Pass 2. If register B1 in context window 1 belongs to Pass 2, the significance states of visited register C0, C1, C2, B0, A0 are equal to pass1 significance change in current bit-plane or pass1 significance state in the previous bit plane. Since register B2, A1, A2 that have not been visited may become significant in Pass 3 of the previous bit plane, the significance states of not visited register B2, A1, A2 are equal to pass1 significance state or pass3 significance state in the previous bit plane or their magnitude bit in current bit plane. Neighbor significance states of register B1 can be predicted by Table 3-2 for pass2. (V0 represents the magnitude bit)

Table 3-2. The neighbor significance states of register B1 for pass2

Register C	Register B	Register A
S1 (Register C0)	S1 (Register B0)	S1(S1 | S3 | V0) (Register A0)
S1 (Register C1)	S1 | S3 (Register B1)	S1 | S3 | V0 (Register A1)
S1 (Register C2)	S1 | S3 | V0 (Register B2)	S1 | S3 | V0 (Register A2)

If the equation S1 | S3 of register D1 in context window 2 is zero and the neighbor significance states of the register D1 are zero, register D1 belongs to Pass 3. If register D1 in context window 2 belongs to Pass 3, the significance states of visited register E0, E1, E2, D0, C0 are equal to pass1(or pass3) significance change in current bit-plane or pass1(or pass3) significance state in the previous bit plane. Since register D2, C1, C2 that have not been visited may become significant in pass 3 or pass1 of the previous bit plane, the significance states of not visited register D2, C1, C2 are equal to pass1 significance state or pass3 significance state in the previous bit plane. Neighbor significance states of register D1 can be predicted by Table 3-3 for pass3.

Table 3-3. The neighbor significance states of register D1 for pass3

Register E	Register D	Register C
S1 | S3 (Register E0)	S1 | S3 (Register D0)	S1 | S3 (Register C0)
S1 | S3 (Register E1)	S1 | S3 (Register D1)	S1 | S3 (Register C1)
S1 | S3 (Register E2)	S1 | S3 (Register D2)	S1 | S3 (Register C2)

3.3 Column -Based operation for Pass-Parallel

As shown in Figure 3-4, a 5 x 5 shift register array is composed the context window 1 and context window 2. Context window 1 is used to judge whether the samples are coded in Pass1 or Pass2 or not. Context window 2 is used to judge whether the samples are coded in Pass3 or not. The pass3 coding operation is delayed by two stripe columns to eliminate the effect between Pass3 and other two passes. Subsequently, to eliminate the dependence of coding operations on the next stripe, the stripe cause mode is also adopted. As discussed above, four coding state variables are used in pass coding operations. As shown in Figure 3-5, a 5X5 shift register array is needed for significance state variable S1, significance state variable S3 and sign state variable X0. As shown in Figure 3-6, a 4X5 shift register array is needed for magnitude state variable V0.

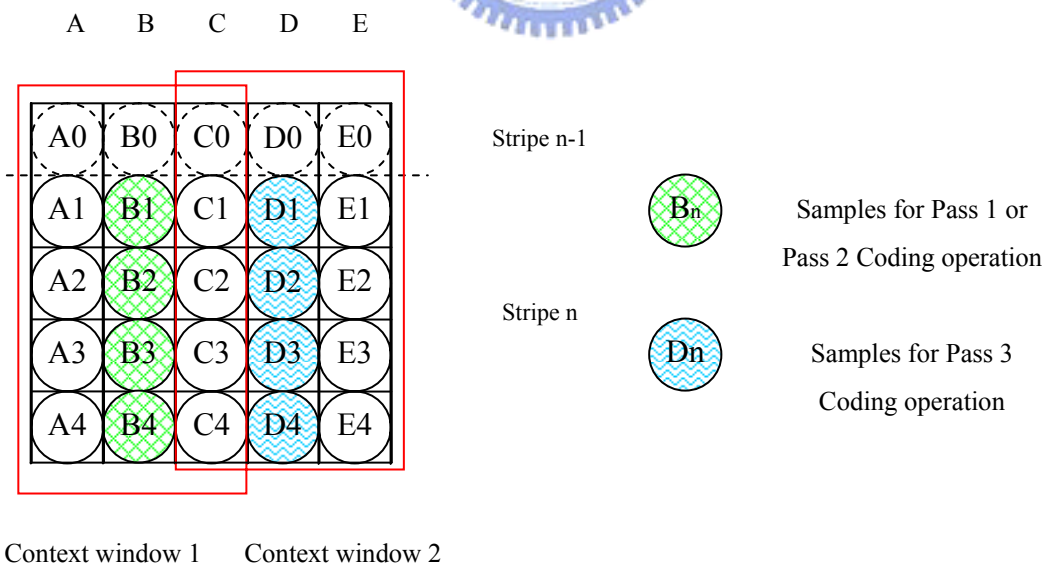


Figure 3-5. Pass-Parallel column-based shift register array (5x5) for significance states S1, S3 and sign state X0

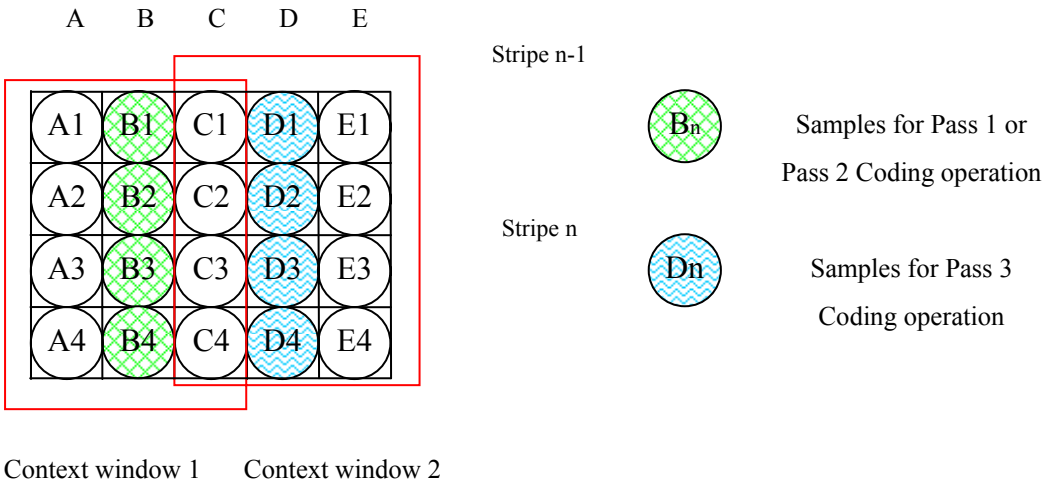


Figure 3-6. Pass-Parallel column-based shift register array (4x5) for magnitude state V0

All four consecutive samples in a column are executed by sample parallel and pass parallel method to speed up the coding performance of Pass-Parallel Sample-Based Context Formation Encoder. As shown in Figure 3-7, Context window 1 is formed by context window B1, context window B2, context window B3, context window B4. Context window B1 is responsible for checking pass type of sample B1. Context window B2 is responsible for checking pass type of sample B2. Context window B3 is responsible for checking pass type of sample B3. Context window B4 is responsible for checking pass type of sample B4. In order to code all consecutive four samples in register B in parallel and produce CXD pairs simultaneously, the state equation in upper position of each coded sample from sample B2 to B4 must be modified.

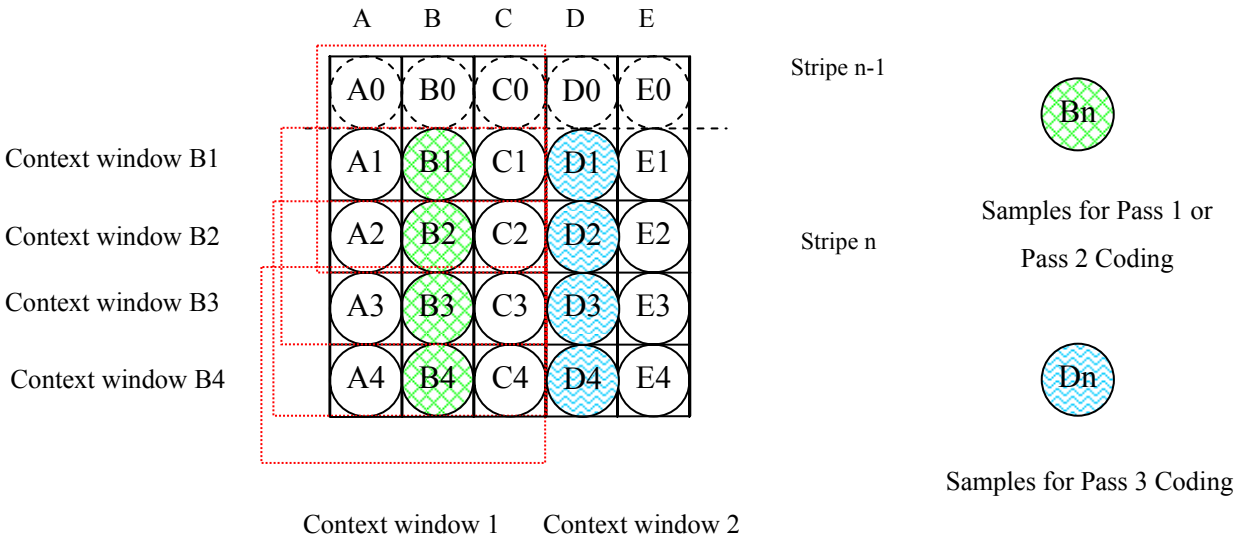


Figure 3-7. Pass-Parallel sample-parallel shift register array (5x5)

When decide pass type and context label of sample B2, both pass type and significant state of sample B1 had to be considered. And so do samples B3 and B4, respectively. According to principle described above, we obtain easily pass type and neighbor significance states of coded sample. Neighbor significance states of column-based sample B1, B2, B3, B4 can be predicted for pass1, as shown in Table 3-4, 3-5, 3-6, 3-7.

Table 3-4.The neighbor significance states of column-based sample B1 for pass1

Register C	Register B	Register A
S1_C[0] (Sample C0)	S1_B[0] (Sample B0)	S1_A[0] (Sample A0)
S1_C[1] (Sample C1)	S1_B[1] | S3_B[1] (Sample B1)	S1_A[1] | S3_A[1] (Sample A1)
S1_C[2] (Sample C2)	S1_B[2] | S3_B[2] (Sample B2)	S1_A[2] | S3_A[2] (Sample A2)

Table 3-5. The neighbor significance states of column-based sample B2 for pass1

Register C	Register B	Register A
S1_C[1] (Sample C1)	S1_B[1] | P1_S_B[1] (Sample B1)	S1_A[1] | S3_A[1] (Sample A1)
S1_C[2] (Sample C2)	S1_B[2] | S3_B[2] (Sample B2)	S1_A[2] | S3_A[2] (Sample A2)
S1_C[3] (Sample C3)	S1_B[2] | S3_B[2] (Sample B3)	S1_A[3] | S3_A[3] (Sample A3)

Table 3-6. The neighbor significance states of column-based sample B3 for pass1

Register C	Register B	Register A
S1_C[2] (Sample C2)	S1_B[2] | P1_S_B[2] (Sample B2)	S1_A[2] | S3_A[2] (Sample A2)
S1_C[3] (Sample C3)	S1_B[3] | S3_B[3] (Sample B3)	S1_A[3] | S3_A[3] (Sample A3)
S1_C[4] (Sample C4)	S1_B[4] | S3_B[4] (Sample B4)	S1_A[4] | S3_A[4] (Sample A4)

Table 3-7. The neighbor significance states of column-based sample B4 for pass1

Register C	Register B	Register A
S1_C[3] (Sample C3)	S1_B[3] | P1_S_B[3] (Sample B3)	S1_A[3] | S3_A[3] (Sample A3)
S1_C[4] (Sample C4)	S1_B[4] | S3_B[4] (Sample B4)	S1_A[4] | S3_A[4] (Sample A4)
0 (Sample C5)	0 (Sample B5)	0 (Sample A5)

Neighbor significance states of column-based sample B1, B2, B3, B4 can be predicted for pass2, as shown in Table 3-8, 3-9, 3-10, 3-11.

Table 3-8.The neighbor significance states of column-based sample B1 for pass2

Register C	Register B	Register A
S1_C[0] (Sample C0)	S1_B[0] (Sample B0)	S1_A[0] (Sample A0)
S1_C[1] (Sample C1)	S1_B[1] | S3_B[1] (Sample B1)	S1_A[1] | S3_A[1] | V0_A[1] (Sample A1)
S1_C[2] (Sample C2)	S1_B[2] | S3_B[2] | V0_B[2] (Sample B2)	S1_A[2] | S3_A[2] | V0_A[2] (Sample A2)

Table 3-9. The neighbor significance states of column-based sample B2 for pass2

Register C	Register B	Register A
S1_C[1] (Sample C1)	S1_B[1] | P1_S_B[1] (Sample B1)	S1_A[1] | S3_A[1] | V0_A[1] (Sample A1)
S1_C[2] (Sample C2)	S1_B[2] | S3_B[2] (Sample B2)	S1_A[2] | S3_A[2] | V0_A[2] (Sample A2)
S1_C[3] (Sample C3)	S1_B[3] | S3_B[3] | V0_B[3] (Sample B3)	S1_A[3] | S3_A[3] | V0_A[3] (Sample A3)

Table 3-10. The neighbor significance states of column-based sample B3 for pass2

Register C	Register B	Register A
S1_C[2] (Sample C2)	S1_B[2] | P1_S_B[2] (Sample B2)	S1_A[2] | S3_A[2] | V0_A[2] (Sample A2)
S1_C[3] (Sample C3)	S1_B[3] | S3_B[3] (Sample B3)	S1_A[3] | S3_A[3] | V0_A[3] (Sample A3)
S1_C[4] (Sample C4)	S1_B[4] | S3_B[4] | V0_B[4] (Sample B4)	S1_A[4] | S3_A[4] | V0_A[4] (Sample A4)

Table 3-11. The neighbor significance states of column-based sample B4 for pass2

Register C	Register B	Register A
S1_C[3] (Sample C3)	S1_B[3] | P1_S_B[3] (Sample B3)	S1_A[3] | S3_A[3] | V0_A[3] (Sample A3)
S1_C[4] (Sample C4)	S1_B[4] | S3_B[4] (Sample B4)	S1_A[4] | S3_A[4] | V0_A[4] (Sample A4)
0 (Sample C5)	0 (Sample B5)	0 (Sample A5)

Neighbor significance states of column-based sample B1, B2, B3, B4 can be predicted for pass3, as shown in Table 3-12, 3-13, 3-14, 3-15.

Table 3-12. The neighbor significance states of column-based sample D1 for pass3

Register E	Register D	Register C
S1_E[0] | S3_E[0] (Sample E0)	S1_D[0] | S3_D[0] (Sample D0)	S1_C[0] | S3_C[0] (Sample C0)
S3_E[1]|S1_E[1] (Sample E1)	S1_D[1] | S3_D[1] (Sample D1)	S1_C[1] | S3_C[1] (Sample C1)
S3_E[2]|S1_E[2] (Sample E2)	S1_D[2] | S3_D[2] (Sample D2)	S1_C[2] | S3_C[2] (Sample C2)

Table 3-13. The neighbor significance states of column-based sample D2 for pass3

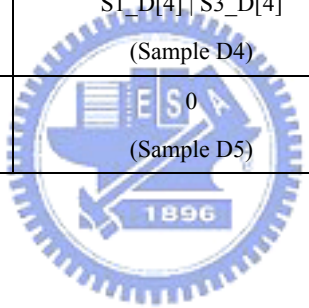
Register E	Register D	Register C
S1_E[1] | S3_E[1] (Sample E1)	S1_D[1] | S3_D[1] (Sample D1)	S1_C[1] | S3_C[1] (Sample C1)
S3_E[2]|S1_E[2] (Sample E2)	S1_D[2] | S3_D[2] (Sample D2)	S1_C[2] | S3_C[2] (Sample C2)
S3_E[3]|S1_E[3] (Sample E3)	S1_D[3] | S3_D[3] (Sample D3)	S1_C[3] | S3_C[3] (Sample C3)

Table 3-14. The neighbor significance states of column-based sample D3 for pass3

Register E	Register D	Register C
S1_E[2] | S3_E[2] (Sample E2)	S1_D[2] | S3_D[2] (Sample D2)	S1_C[2] | S3_C[2] (Sample C2)
S3_E[3]|S1_E[3] (Sample E2)	S1_D[3] | S3_D[3] (Sample D3)	S1_C[3] | S3_C[3] (Sample C3)
S3_E[4]|S1_E[4] (Sample E3)	S1_D[4] | S3_D[4] (Sample D4)	S1_C[4] | S3_C[4] (Sample C4)

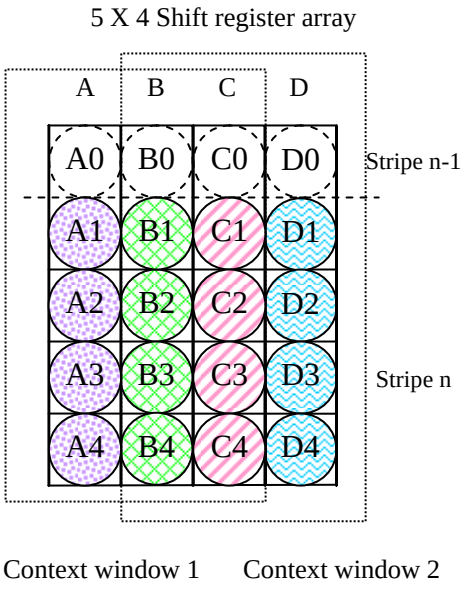
Table 3-15. The neighbor significance states of column-based sample D4 for pass3

Register E	Register D	Register C
S1_E[3] | S3_E[3] (Sample E3)	S1_D[3] | S3_D[3] (Sample D3)	S1_C[3] | S3_C[3] (Sample C3)
S3_E[4]|S1_E[4] (Sample E4)	S1_D[4] | S3_D[4] (Sample D4)	S1_C[4] | S3_C[4] (Sample C4)
0 (Sample E5)	0 (Sample D5)	0 (Sample C5)



CHAPTER 4 Proposed High Speed and Small-Area Pass-Parallel Context Formation Methods

As shown in Figure 4-1, in order to fit the pass-parallel architecture, it merges context window of three coding passes into a 5x4 shift register array. The context window 1 which is formed by registers A, B and C is responsible to generate dual column pass1 and correct coding pass and significance state for all four samples in current and next column. The context window 2 which is formed by registers B, C and D is responsible to execute parallel all coding operation and generate all coding pass. In register A, load data from sram memory to register bits A1 to A4 in positive edge of a clock cycle time. In register B, generate dual column pass1 for all four consecutive samples in current column and coding pass with significance state for all four consecutive samples in next column in positive edge of a clock cycle time. In register C, generate all coding pass and significance state



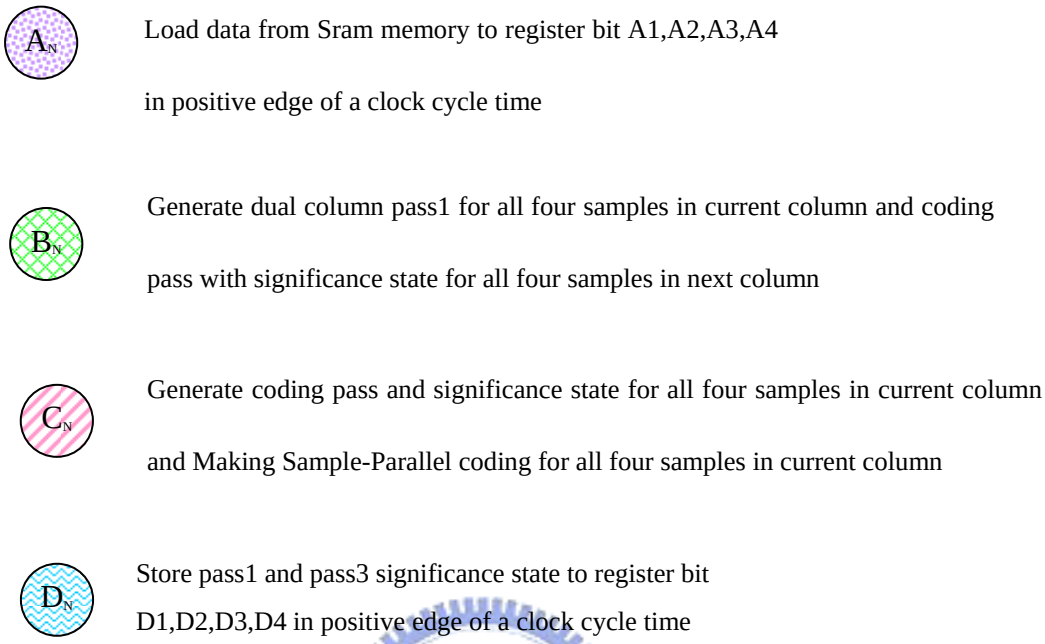


Figure 4-1. Pass-Parallel column-based shift register array (5x4) proposed for static significance state S1,S3 and sign state X0

for all four consecutive samples in current column and making Sample-Parallel coding for all four consecutive samples in current column in positive edge of a clock cycle time. In register D of state variable S1, both current pass1 significance state generated in register C and significance state corresponding column in previous bit plane implement ‘or’ operation and store the result to register bits D1 to D4 in positive edge of a clock cycle time. In register D of state variable S3, both current pass3 significance state generated in register C and significance state corresponding column in previous bit plane implement ‘or’ operation and store the result to register bit D1 to D4 in positive edge of a clock cycle time.

In the proposed architecture, column-based operation is adopted instead of sample-based operation. The fundamental concept of column-based operation is to check four consecutive samples of columns simultaneously. The shift register arrays of the significant states S1 and S3 are responsible to record the pass1 and pass3 significance state of coded samples, as shown in figure 4-1. The shift register array of sign state variable X0 is responsible to record the sign state of coded sample, as shown in figure 4-1. The refinement state variable R indicates whether or not sample coded has already been coded in magnitude refinement pass in previous bit-plane and is set to one after a sample is coded by magnitude refinement coding at first time. Both the significance state S1 and significance state S3 implemented 'Exclusive-OR' operation instead of refinement state. The shift register array of magnitude state variable V0 is recorded by the magnitude bit of coded samples in current coding bit-plane, as shown in Figure 4-2.

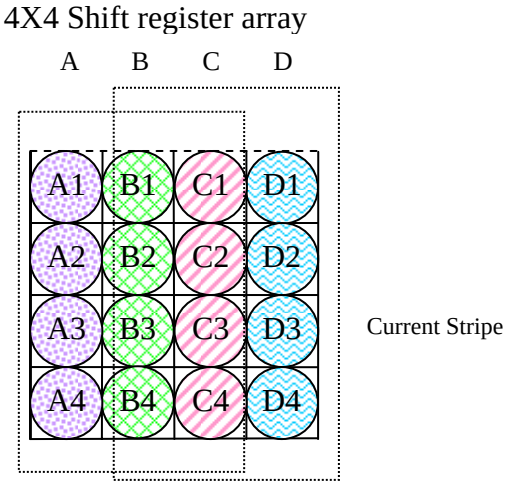


Figure 4-2. Pass-Parallel column-based shift register array (4x4) proposed for magnitude state V0

In this chapter we will propose three methods to improve the performance and reduce the hardware requirement of original Pipeline Pass-Parallel Context Formation. As shown in Figure 4-1, the first one is called **Dual Column Pass1 Generation (DCPG)** method that is used to generate pass1 and pass1 with significance state for current samples B1 to B4 in a clock cycle time of register B and for previous samples C1 to C4 in a clock cycle time of register C simultaneously. The second one is called **All Coding Pass and Significance Change Generation (ACSC)** method that is used to generate all coding pass and pass with significance state for sample C1 to C4 in a clock cycle time of register C. The third one is called **Sample-Parallel Column-Based Coding (SPCC)** method that is used to simultaneously handle all four consecutive samples with correct coding pass and pass with significance state in a clock cycle time of register C. These coding operations include run-length coding 0 (RLC0), run-length coding 1 (RLC1), uniform coding 0 (UC0), uniform coding 1 (UC1), zero coding (ZC), sign coding (SC), and magnitude refinement coding (MRC). These types of coding pass include pass1 (Significance Pass, P1), pass1 with significance state (P1_S), pass2 (Magnitude Refinement Pass, P2), pass2 with the first refinement coding (P2_1), pass3 (Clean Up Pass, P3), pass3 with run length coding 0 (P3_RLC0), pass3 with run length coding 1 and uniform coding (P3_RLC1_UC). Three methods proposed will be discussed in the following sections.

4.1 Dual Column Pass 1 Generation method

As shown in Figure 4-3, in order to simultaneously decide pass1 and pass1 with significance state for samples B1 to B4 in register B and for samples C1 to C4 in

register C in parallel, we process context window B1, context window B2, context window B3, context window B4 in parallel.

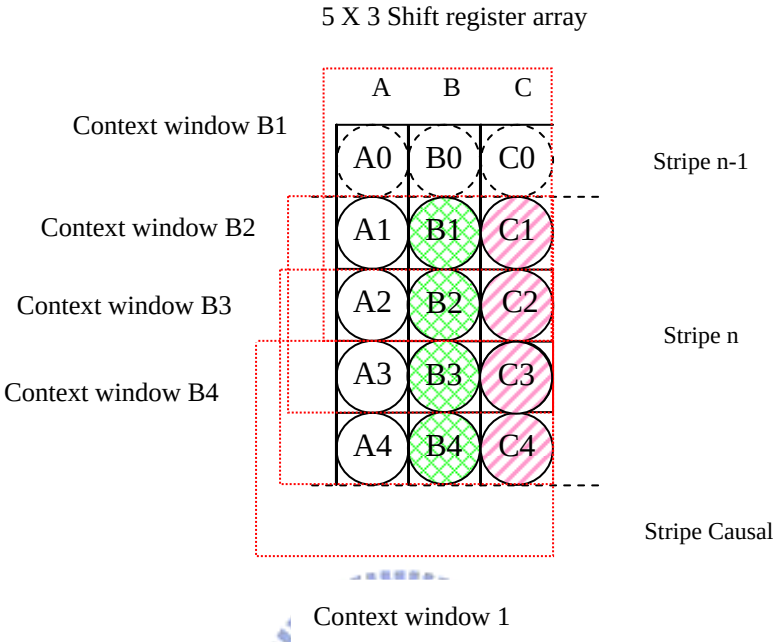


Figure 4-3. A 5X3 shift register context window for Dual Column Pass1 generation method

Figure 4-4 shows the shift register array of state variables used in the context window 1. The state variable S1 or S3 is responsible to record the pass1 or pass3 significance state change in current bit-plane for samples in register B. The shift register array of state variable V0 is responsible to record the magnitude bit of samples in register B. The pass1 significance change generated in register B is not stored in register C. The pass type information generated in register B store in temporary register in next clock cycle.

A	B	C
S1_A[0]	S1_B[0]	S1_C[0]
S1_A[1]	S1_B[1]	S1_C[1]
S1_A[2]	S1_B[2]	S1_C[2]
S1_A[3]	S1_B[3]	S1_C[3]
S1_A[4]	S1_B[4]	S1_C[4]

(a)

A	B	C
S3_A[0]	S3_B[0]	S3_C[0]
S3_A[1]	S3_B[1]	S3_C[1]
S3_A[2]	S3_B[2]	S3_C[2]
S3_A[3]	S3_B[3]	S3_C[3]
S3_A[4]	S3_B[4]	S3_C[4]

(b)

A	B	C
V0_A[0]	V0_B[0]	V0_C[0]
V0_A[1]	V0_B[1]	V0_C[1]
V0_A[2]	V0_B[2]	V0_C[2]
V0_A[3]	V0_B[3]	V0_C[3]
V0_A[4]	V0_B[4]	V0_C[4]

(c)

Figure 4-4. Shift register array of state variables used in Dual Column Pass1 generation (a) ; (b) static significance states S1, S3 ; (c) Magnitude state V0

In order to improve speed, we divided it into three steps in dual column pass1 generation method. Three steps include pre-processing for pass1 generation, horizontal processing for pass1 generation, and vertical processing for pass1 generation. These steps will be discussed in the following sections.

4.1.1 Pre-processing for Pass1 generation of samples in register B

According to the Pass-Parallel architecture discussed in previous chapter, in order to decide pass types of sample coded, it is necessary to decide the neighbor significance state of samples coded. Since we only handle pre-processing pass1 generation, horizontal and vertical direction generation is not considered in this section. The values of register C in state variables S1 are not changed after pass1 generation. It is different from the register C in state variables S1 of original pipeline pass-parallel architecture. It is necessary to decide the relationship between these variables. In next section, we need to take care of the possibility that the significance state of an immediate vertical or horizontal neighbor may change during the current pass. If the significance state of sample B1 in context window 1 is zero and the any neighbor significance state of sample B1 is one, sample B1 belongs to Pass 1. As shown in Table 4-1, the first Pass 1 generation of sample B1 is expressed as follows:

```
if ((!(S1_B[1]|S3_B[1])) & ((S1_C[0]|S1_C[1]|S1_C[2])|(S1_B[0]|S1_A[0])
        |(S1_B[2]|S3_B[2])|(S1_A[1]|S3_A[1])|(S1_A[2]|S3_A[2])))
    F1_P1_B[1]    = 1'b1;
  else
   F1_P1_B[1]    = 1'b0;
```

Neighbor significance states of sample B1 can be predicted, as shown in Table 4-1.

Table 4-1 Neighbor significance states of sample B1 for pre-processing

Register C	Register B	Register A
S1_C[0]	S1_B[0]	S1_A[0]
S1_C[1]	S1_B[1]|S3_B[1]	S1_A[1]|S3_A[1]
S1_C[2]	S1_B[2]|S3_B[2]	S1_A[2]|S3_A[2]

If the significance state of sample B2 in context window 1 is zero and the any neighbor significance state of sample B2 is one, sample B2 belongs to Pass 1. As shown in Table 4-2, the first Pass 1 generation of sample B2 is expressed as follows:

```
if ((!(S1_B[2]|S3_B[2])) &
    ((S1_C[1]|S1_C[2])|(S1_C[3]|S1_B[1])|(S1_B[3]|S3_B[3])
    |(S1_A[1]|S3_A[1])|(S1_A[2]|S3_A[2])|(S1_A[3]|S3_A[3])))
        F1_P1_B[2]   = 1'b1;
    else
      F1_P1_B[2]   = 1'b0;
```

Neighbor significance states of sample B2 can be predicted, as shown in Table 4-2.

Table 4-2 Neighbor significance states of sample B2 for pre-processing

Register C	Register B	Register A
S1_C[1]	S1_B[1]	S1_A[1]|S3_A[1]
S1_C[2]	S1_B[2]|S3_B[2]	S1_A[2]|S3_A[2]
S1_C[3]	S1_B[3]|S3_B[3]	S1_A[3]|S3_A[3]

If the significance state of sample B3 in context window 1 is zero and the any neighbor significance state of sample B3 is one, sample B3 belongs to Pass 1. As shown in Table 4-3, the first Pass 1 generation of sample B3 is expressed as follows:

```
if ((!(S1_B[3]|S3_B[3])) &
((S1_C[2]|S1_C[3])|(S1_C[4]|S1_B[2])|(S1_B[4]|S3_B[4])
```

```
|(S1_A[2]|S3_A[2])|(S1_A[3]|S3_A[3])|(S1_A[4]|S3_A[4])))
        F1_P1_B[3]    = 1'b1;
    else
    F1_P1_B[3]    = 1'b0;
```

Neighbor significance states of sample B3 can be predicted, as shown in Table 4-3

Table 4-3 Neighbor significance states of sample B3 for pre-processing

Register C	Register B	Register A
S1_C[2]	S1_B[2]	S1_A[2]|S3_A[2]
S1_C[3]	S1_B[3]|S3_B[3]	S1_A[3]|S3_A[3]
S1_C[4]	S1_B[4]|S3_B[4]	S1_A[4]|S3_A[4]

If the significance state of sample B4 in context window 1 is zero and the any neighbor significance state of sample B4 is one, sample B4 belongs to Pass 1. As shown in Table 4-4, the first Pass 1 generation of sample B4 is expressed as follows:

```
if ((!(S1_B[4]|S3_B[4])) & ((S1_C[3]|S1_C[4]|S1_B[3])
  |(S1_A[3]|S3_A[3])|(S1_A[4]|S3_A[4])))
     F1_P1_B[4]    = 1'b1;
  else
  F1_P1_B[4]    = 1'b0;
```

Neighbor significance states of sample B4 can be predicted, as shown in Table 4-4.

Table 4-4 Neighbor significance states of sample B4 for pre-processing

Register C	Register B	Register A
S1_C[3]	S1_B[3]	S1_A[3]|S3_A[3]
S1_C[4]	S1_B[4]|S3_B[4]	S1_A[4]|S3_A[4]
0	0	0

4.1.2 Horizontal Processing for pass1 generation of samples in register B

We need to take care of the possibility that the significance state of an immediate horizontal neighbor may change during the current pass. Specifically, for samples B1, B2, B3 and B4 in register B we must take into account the possible change of significance in samples C1, C2, C3 and B4 in register C, respectively during the current pass. If the result F1_P1_B[1] of sample B1 in register B is zero, the pass type of sample B1 may be changed by significance state of horizontal samples C1 and C2 in register C. As shown in Table 4-5, the second Pass 1 of sample B1 are expressed as follows:

F2_P1_B[1]= F1_P1_B[1] | P1_S_C[1] | P1_S_C[2]

Neighbor significance states of sample B1 can be predicted by Table 4-5 for horizontal processing of pass 1 generation.

Table 4-5 Neighbor significance states of sample B1 for horizontal processing

Register C	Register B	Register A
X	X	X
P1_S_C[1]	F1_P1_B[1]	X
P1_S_C[2]	X	X

If the result F1_P1_B[2] of sample B2 in register B is zero, the pass type of sample B2 may be changed by significance state of horizontal samples C1, C2 and C3 in register C. As shown in Table 4-6, the second Pass 1 of sample B2 are expressed as follows:

F2_P1_B[2]= F1_P1_B[2] | P1_S_C[1] | P1_S_C[2] | P1_S_C[3];

Neighbor significance states of sample B2 can be predicted by Table 4-6 for horizontal processing of pass 1 generation.

Table 4-6 Neighbor significance states of sample B2 for horizontal processing

Register C	Register B	Register A
P1_S_C[1]	X	X
P1_S_C[2]	F2_P1_B[2]	X
P1_S_C[3]	X	X

If the result F1_P1_B[3] of sample B3 in register B is zero, the pass type of sample B3 may be changed by significance state of horizontal samples C2, C3 and C4 in register C. As shown in Table 4-7, the second Pass 1 of sample B3 are expressed as follows:

F2_P1_B[3]= F1_P1_B[3] | P1_S_C[2] | P1_S_C[3] | P1_S_C[4];

Neighbor significance states of sample B3 can be predicted by Table 4-7 for horizontal processing of pass 1 generation.

Table 4-7 Neighbor significance states of sample B3 for horizontal processing

Register C	Register B	Register A
P1_S_C[2]	X	X
P1_S_C[3]	F2_P1_B[3]	X
P1_S_C[4]	X	X

If the result F1_P1_B[4] of sample B4 in register B is zero, the pass type of sample B4 may be changed by significance state of horizontal samples C3 and C4 in register C. As shown in Table 4-8, the second Pass 1 of sample B4 are expressed as follows:

F2_P1_B[4]= F1_P1_B[4] | (P1_S_C[3] | P1_S_C[4])

Neighbor significance states of sample B4 can be predicted by Table 4-8 for horizontal processing of pass 1 generation.

Table 4-8 New neighbor significance states of sample B4

Register C	Register B	Register A
P1_S_C[3]	X	X
P1_S_C[4]	F1_P1_B[4]	X
0	0	0

4.1.3 Vertical Processing for pass1 generation of samples in register B

We need to take care of the possibility that the significance state of an immediate vertical neighbor may change during the current pass. Specifically, for samples B2, B3, B4, we must take into account the possible change of significance in samples B1, B2, and B3, respectively during the current pass. The pass type and significance state of sample B1 in register B are not changed by sample B0 in register B, because the significance state in sample B0 belongs to the previous bit plane. As shown in Table 4-9, the real Pass 1 and pass1 with significance state of sample B1 are expressed as follows:

$$P1\_B[1] = F2\_P1\_B[1]$$
$$P1\_S\_B[1] = P1\_B[1] \ \& \ V0\_B[1]$$

Neighbor significance states of sample B1 can be predicted by Table 4-9 for vertical processing of pass 1 generation.

Table 4-9 Neighbor significance states of sample B1 for vertical processing

Register C	Register B	Register A
X	X	X
X	F2_P1_B[1]	X
X	X	X

If the result F2_P1_B[2] of sample B2 in register B is zero, the pass type of sample B2 may be changed by significance state of vertical sample B1. As shown in Table 4-10, the real Pass 1 and Pass1 with significance state of sample B2 are expressed as follows:

$$P1\_B[2] = P1\_S\_B[1] \mid F2\_P1\_B[2]$$
$$P1\_S\_B[2] = P1\_B[2] \,\&\, V0\_B[2]$$

Neighbor significance states of sample B2 can be predicted by Table 4-10 for vertical processing of pass 1 generation.

Table 4-10 Neighbor significance states of sample B2 for vertical processing

Register C	Register B	Register A
X	P1_S_B[1]	X
X	F2_P1_B[2]	X
X	X	X

If the result F2_P1_B[3] of sample B3 in register B is zero, the pass type of sample B3 may be changed by significance state of vertical sample B2. As shown in Table 4-11, the real Pass 1 and Pass1 with significance state of sample B3 are expressed as follows:

$$P1\_B[3] = P1\_S\_B[2] \mid F2\_P1\_B[3]$$
$$P1\_S\_B[3] = P1\_B[3] \,\&\, V0\_B[3]$$

Neighbor significance states of sample B3 can be predicted by Table 4-11 for vertical processing of pass 1 generation.

Table 4-11 Neighbor significance states of sample B3 for vertical processing

Register C	Register B	Register A
X	P1_S_B[2]	X
X	F2_P1_B[3]	X
X	X	X

If the result F2_P1_B[4] of sample B4 in register B is zero, the pass type of sample B4 may be changed by significance state of vertical sample B3. As shown in Table 4-12, the real Pass 1 and Pass1 with significance state of sample B4 are expressed as follows:

$$P1\_B[4] = P1\_S\_B[3] \mid F2\_P1\_B[4]$$
$$P1\_S\_B[4] = P1\_B[4] \,\&\, V0\_B[4]$$

Neighbor significance states of sample B4 can be predicted by Table 4-12 for vertical processing of pass 1 generation.

Table 4-12 Neighbor significance states of sample B4 for vertical processing

Register C	Register B	Register A
X	P1_S_B[3]	X
X	F2_P1_B[4]	X
X	X	X

4.2 All Coding Pass and Significance Change generation method

From the method discussed in the previous section, we can obtain pass1 information and pass1 with significance state for each sample in register B in parallel. And we send pass1 information to register C in rise edge of next clock cycle. We can obtain all coding pass information in the same clock cycle. These signals of coding passes information include pass1 (P1_B_C), pass1 and significance state (P1_S_C), pass2 (P2_C), pass2 and the first time magnitude refinement coding (P2_1_C), pass3 (P3_C), pass3 and significance state (P3_S_C), pass3 run length coding 0 (P3_RLC0), pass3 run length coding 1 with uniform coding bit 1 (P3_RLC1_UC1), pass3 run length coding 1 with uniform coding bit 0 (P3_RLC1_UC0). The signal ‘P1_B_C’ represents pass1 information from register B and is responsible to execute zero coding

(ZC). The signal ‘P1_S_C’ represents pass1 with significance state information in register C and is responsible to execute zero coding (ZC) and sign coding (SC). The signal ‘P2_C’ represents pass2 information generated in register C and is responsible to execute magnitude refinement coding. The signal ‘P2_1_C’ represents first time pass2 information in register C and is responsible to execute the first time magnitude refinement coding. The signal ‘P3_C’ represents pass3 information in register C and is responsible to execute zero coding (ZC). The signal ‘P3_S_C’ represents pass3 with significance state information in register C and is responsible to execute zero coding (ZC) and sign coding (SC). The signal ‘P3_RLC0’ represents that the neighbor significance states in all four samples of a column and their magnitude bit value is all zero. The signal ‘P3_RLC1_UC1’ represents the high bit of uniform coding in pass3. The signal ‘P3_RLC1_UC0’ represents the low bit of uniform coding in pass3.

As shown in Figure 4-5, we can generate simultaneously all coding pass in a clock cycle of register C.

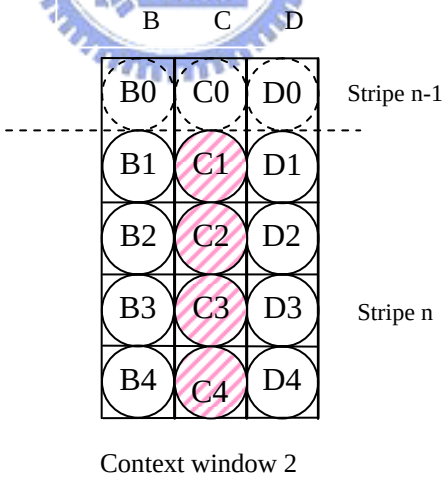


Figure 4-5. A 5X3 Shift Register Context window for generating all coding pass information and significance change of Pass1 or Pass3

Figure 4-6 shows the state variables used in the context window 2. The shift register array of the state variable S1 is responsible to record the pass1 with significance state for the sample coded in register B or register C. The shift register

array of the state variable S3 is responsible to record the pass3 with significance state for the sample coded in register B or register C. The shift register array of the state variable V0 is responsible to record the magnitude bit of sample coded in register B or register C or register D.

B	C	D
S1_B[0]	S1_C[0]	S1_D[0]
S1_B[1]	S1_C[1]	S1_D[1]
S1_B[2]	S1_C[2]	S1_D[2]
S1_B[3]	S1_C[3]	S1_D[3]
S1_B[4]	S1_C[4]	S1_D[4]

(a)

B	C	D
S3_B[0]	S3_C[0]	S3_D[0]
S3_B[1]	S3_C[1]	S3_D[1]
S3_B[2]	S3_C[2]	S3_D[2]
S3_B[3]	S3_C[3]	S3_D[3]
S3_B[4]	S3_C[4]	S3_D[4]

(b)

B	C	D
V0_B[0]	V0_C[0]	V0_D[0]
V0_B[1]	V0_C[1]	V0_D[1]
V0_B[2]	V0_C[2]	V0_D[2]
V0_B[3]	V0_C[3]	V0_D[3]
V0_B[4]	V0_C[4]	V0_D[4]

(c)

Figure 4-6 Shift register array for state variables used in generating all coding pass

(a) ; (b) Static significance states S1, S3 ; (c) Magnitude state V0

The pass1 information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
if (!RST_N | START)
begin
 P1_B_C      <= #`DELAY 5'b0;
   end
else begin
   P1_B_C  <= #`DELAY P1_B;
     end
```

The pass1 with significance state of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign   P1_S_C[1] = P1_B_C[1] & V0_C[1];
assign   P1_S_C[2] = P1_B_C[2] & V0_C[2];
assign   P1_S_C[3] = P1_B_C[3] & V0_C[3];
assign   P1_S_C[4] = P1_B_C[4] & V0_C[4];
```

The pass2 information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign    P2_C[1]   = S1_C[1] | S3_C[1];
assign    P2_C[2]   = S1_C[2] | S3_C[2];
assign    P2_C[3]   = S1_C[3] | S3_C[3];
assign    P2_C[4]   = S1_C[4] | S3_C[4];
```

The first time paas2 magnitude refinement coding information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign    P2_1_C[1]   = S1_C[1] ^ S3_C[1];
assign    P2_1_C[2]   = S1_C[2] ^ S3_C[2];
assign    P2_1_C[3]   = S1_C[3] ^ S3_C[3];
assign    P2_1_C[4]   = S1_C[4] ^ S3_C[4];
```

The pass3 information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign   P3_C[1] = !(P2_C[1] | P1_B_C[1]);
```

```
assign    P3_C[2] = !(P2_C[2] | P1_B_C[2]);
assign    P3_C[3] = !(P2_C[3] | P1_B_C[3]);
assign    P3_C[4] = !(P2_C[4] | P1_B_C[4]);
```

The paas3 with significance state of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign    P3_S_C[1] = P3_C[1] & V0_C[1];
assign    P3_S_C[2] = P3_C[2] & V0_C[2];
assign    P3_S_C[3] = P3_C[3] & V0_C[3];
assign    P3_S_C[4] = P3_C[4] & V0_C[4];
```

The pass3 run-length code (P3_RLC0, P3_RLC1) information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign    P3_RLC0_T1 = P3_C[1] & P3_C[2]& P3_C[3] & P3_C[4];
assign    P3_MAG     = V0_C[1] | V0_C[2]| V0_C[3] | V0_C[4];
assign    P3_RLC0    = P3_RLC0_T1 & (!P3_MAG);
assign    P3_RLC1    = P3_RLC0_T1 & P3_MAG;
```

The pass3 run-length code 1 with uniform code 1 (P3_RLC1_UC1) information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign    P3_RLC1_UC1_T1 = !(V0_C[4] | V0_C[3]);
assign    P3_RLC1_UC1 = !(P3_RLC1_UC1_T1 | V0_C[2] | V0_C[1]);
```

The pass3 run-length code 1 with uniform code 0 (P3_RLC1_UC0) information of samples C1 to C4 in register C can be obtained by following hardware description language:

```
assign    P3_RLC1_UC0_T1 = !(!V0_C[4] | V0_C[3]);
assign    P3_RLC1_UC0_T2 = !(P3_RLC1_UC0_T1 | V0_C[2]);
assign    P3_RLC1_UC0 = !(P3_RLC1_UC0_T2 | V0_C[1]);
```

4.3 Sample-Parallel Column-Based Coding method

From the method discussed in the previous section, we can obtain all coding pass type and pass type with significance state information for samples C1 to C4 in register C in parallel. And we can simultaneously execute all coding operation by all coding pass information generated in register C. All coding operations for all four consecutive samples in the same column are performed in context window 2. So we can generate all coding pass and execute all coding operation in a clock cycle of register C.

Figure 4-7 shows the state variables used in the context window 2. All significance states is shown for magnitude refinement coding, zero coding, sign coding in Figure 4-7 (A). The magnitude state variable is shown in Figure 4-7(B) in order to record the magnitude bit of each sample in register C. The sign state variable is shown in Figure 4-7(C) in order to record the sign bit of each sample in register C.

B	C	D
S1_B[0] S3_B[0]	S1_C[0] S3_C[0]	S1_D[0] S3_D[0]
S1_B[1] S3_B[1] P1_S_B[1]	S1_C[1] S3_C[1] P1_S_C[1] P3_S_C[1]	S1_D[1] S3_D[1]
S1_B[2] S3_B[2] P1_S_B[2]	S1_C[2] S3_C[2] P1_S_C[2] P3_S_C[2]	S1_D[2] S3_D[2]
S1_B[3] S3_B[3] P1_S_B[3]	S1_C[3] S3_C[3] P1_S_C[3] P3_S_C[3]	S1_D[3] S3_D[3]
S1_B[4] S3_B[4] P1_S_B[4]	S1_C[4] S3_C[4] P1_S_C[4] P3_S_C[4]	S1_D[4] S3_D[4]

(A)

B	C	D
V0_B[0]	V0_C[0]	V0_D[0]
V0_B[1]	V0_C[1]	V0_D[1]
V0_B[2]	V0_C[2]	V0_D[2]
V0_B[3]	V0_C[3]	V0_D[3]
V0_B[4]	V0_C[4]	V0_D[4]

(B)

B	C	D
X0_B[0]	X0_C[0]	X0_D[0]
X0_B[1]	X0_C[1]	X0_D[1]
X0_B[2]	X0_C[2]	X0_D[2]
X0_B[3]	X0_C[3]	X0_D[3]
X0_B[4]	X0_C[4]	X0_D[4]

(C)

Figure 4-7 All static significance state and dynamic significance state used in MRC, SC and ZC

As shown in Figure 4-7(a), the S1_B, S1_C, and S1_D is responsible to records the significance state for the sample coded in Pass 1 of the corresponding column of the previous bit-plane, respectively in register B, register C and register D. The S3_B, S3_C, and S3_D is responsible to records the significance state for the sample coded in Pass 3 of the corresponding column of the previous bit-plane, respectively in register B, register C and register D. These state variables are used for coding operations, including ZC, SC, MRC, RLC0, RLC1_UC1, RLC1_UC0 operations that are described in previous section. Samples in register C of context window 2 are coded concurrently by appropriate coding operations according to the pass types generated for sample C1 to sample C4 in register C.

4.3.1 MRC Coding Expression for sample C1 to C4 in Register C for Pass2

The pass types of sample C1 to C4 in register C of context window 2 can be easily decided by discussion in previous section. When coding samples C1 to C4 in

register C, the significance states of their neighbor samples are also predicted. The hard-macro of magnitude refinement coding is shown in Figure 4-8.

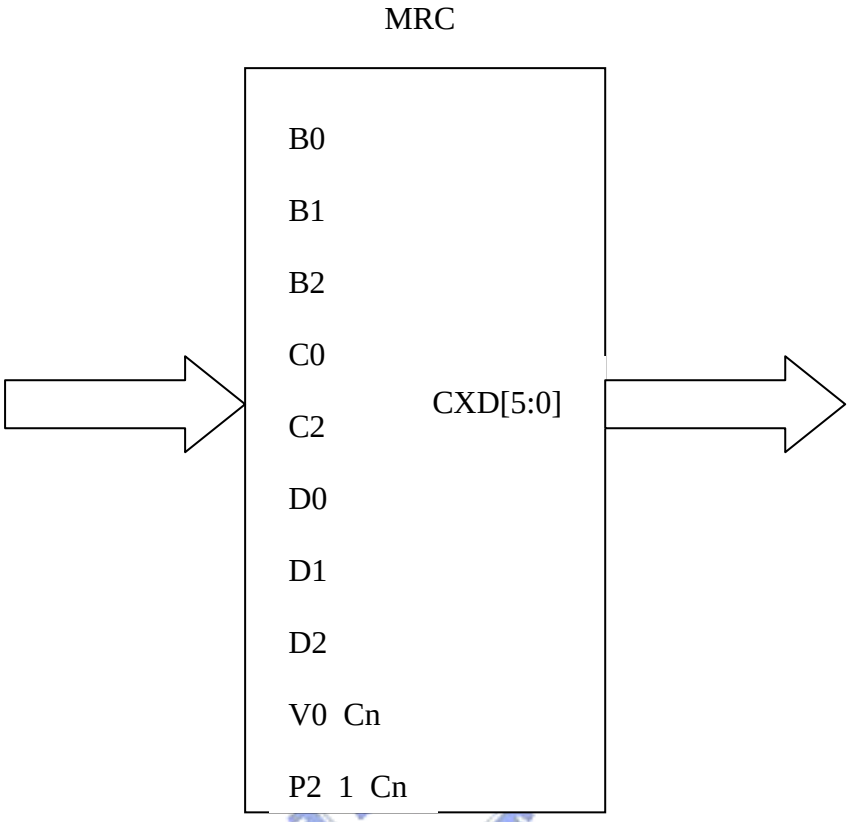


Figure 4-8 Hard-macro of magnitude refinement coding

As shown in Figure 4-8, the magnitude refinement coding of sample Cn is expressed by following hardware description language:

```
SUM = B0 | B1 | B2 | C0 | C2 |D0 | D1 | D2 ;
CXD[5] = V0_Cn;
if (P2_1_Cn == 0) begin
    CXD[4:0] = 5'd16;
end
else begin
    if ((SUM == 0)) begin
        CXD[4:0] = 5'd14;
    end
    else begin
        CXD[4:0] = 5'd15;
    end
 end
```

As shown in Table 4-13, the magnitude refinement coding of sample C1 is expressed by following hardware description language:

```
assign  B0_[1] = S1_B[0];
assign  B1_[1] = S1_B[1]|S3_B[1]|P1_S_B[1];
assign  B2_[1] = S1_B[2]|S3_B[2]|P1_S_B[2];
assign  C0_[1] = S1_C[0];
assign  C2_[1] = S1_C[2]|S3_C[2]|P1_S_C[2];
assign  D0_[1] = S1_D[0];
assign  D1_[1] = S1_D[1];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign  D2_[1] = S1_D[2];//
```

Table 4-13 Neighbor significance states of sample C1 for MRC

Register D	Register C	Register B
S1_D[0] (D0_[1])	S1_C[0] (C0_[1])	S1_B[0] (B0_[1])
S1_D[1] (D1_[1])	V0_C[1] (V0_Cn)	S1_B[1] | S3_B[1]| P1_S_B[1] (B1_[1])
S1_D[2] (D2_[1])	S1_C[2] | S3_C[2]| P1_S_C[2] (C2_[1])	S1_B[2] | S3_B[2]| P1_S_B[2] (B2_[1])

As shown in Table 4-14, the magnitude refinement coding of sample C2 are expressed by following hardware description language:

```
assign  B0_[2] = S1_B[1]|S3_B[1]|P1_S_B[1];
assign  B1_[2] = S1_B[2]|S3_B[2]|P1_S_B[2];
assign  B2_[2] = S1_B[3]|S3_B[3]|P1_S_B[3];
assign  C0_[2] = S1_C[1]|P1_S_C[1];
assign  C2_[2] = S1_C[3]|S3_C[3]|P1_S_C[3];
assign  D0_[2] = S1_D[1];
assign  D1_[2] = S1_D[2];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign  D2_[2] = S1_D[3];//
```

Table 4-14 Neighbor significance states of sample C2 for MRC

Register D	Register C	Register B
S1_D[1] (D0_[2])	S1_C[1] |P1_S_C[1] (C0_[2])	S1_B[1] | S3_B[1] | P1_S_B[1] (B0_[2])
S1_D[2] (D1_[2])	V0_C[2] (V0_C[2])	S1_B[2] | S3_B[2] | P1_S_B[2] (B1_[2])
S1_D[3] (D2_[2])	S1_C[3] | S3_C[3] | P1_S_C[3] (C2_[2])	S1_B[3] | S3_B[3] | P1_S_B[3] (B2_[2])

As shown in Table 4-15, the magnitude refinement coding of sample C3 are expressed by following hardware description language:

```
assign   B0_[3] = S1_B[2]|S3_B[2]|P1_S_B[2];
assign   B1_[3] = S1_B[3]|S3_B[3]|P1_S_B[3];
assign   B2_[3] = S1_B[4]|S3_B[4]|P1_S_B[4];
assign   C0_[3] = S1_C[2]|P1_S_C[2];
assign   C2_[3] = S1_C[4]|S3_C[4]|P1_S_C[4];
assign   D0_[3] = S1_D[2];
assign   D1_[3] = S1_D[3];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign   D2_[3] = S1_D[4];//
```

Table 4-15 Neighbor significance states of sample C3 for MRC

Register D	Register C	Register B
S1_D[2] (D0_[3])	S1_C[2] | P1_S_C[2] (C0_[3])	S1_B[2] | S3_B[2] | P1_S_B[2] (B0_[3])
S1_D[3] (D1_[3])	V0_C[3] (V0_C[3])	S1_B[3] | S3_B[3] | P1_S_B[3] (B1_[3])
S1_D[4] (D2_[3])	S1_C[4] | S3_C[3]| P1_S_C[4] (C2_[3])	S1_B[4] | S3_B[4] | P1_S_B[4] (B2_[3])

As shown in Table 4-16, the magnitude refinement coding of sample C4 are expressed by following hardware description language:

```
assign    B0_[4] = S1_B[3]|S3_B[3]|P1_S_B[3];
assign    B1_[4] = S1_B[4]|S3_B[4]|P1_S_B[4];
assign    B2_[4] = 0;
assign    C0_[4] = S1_C[3]|P1_S_C[3];
assign    C2_[4] = 0;
assign    D0_[4] = S1_D[3];
assign    D1_[4] = S1_D[4];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign    D2_[4] = 0;//
```

Table 4-16 Neighbor significance states of sample C4 for MRC

Register D	Register C	Register B
S1_D[3] (D0_[4])	S1_C[3] | P1_S_C[3] (C0_[4])	S1_B[3] | S3_B[3] | P1_S_B[3] (B0_[4])
S1_D[4] (D1_[4])	S1_C[4] | S3_C[4] (C1_[4])	S1_B[4] | S3_B[4] | P1_S_B[4] (B1_[4])
0 (D2_[4])	0 (C2_[4])	0 (B2_[4])

4.3.2 Zero Coding Expression of sharing between Pass1 and Pass3 for Sample C1 to C4 in Register C

If sample C1 to C4 in register C belongs to pass 1 or pass3 and their magnitude bit value are zero, they must be coded by zero coding (ZC). The hard-macro of zero coding is shown in Figure 4-9.

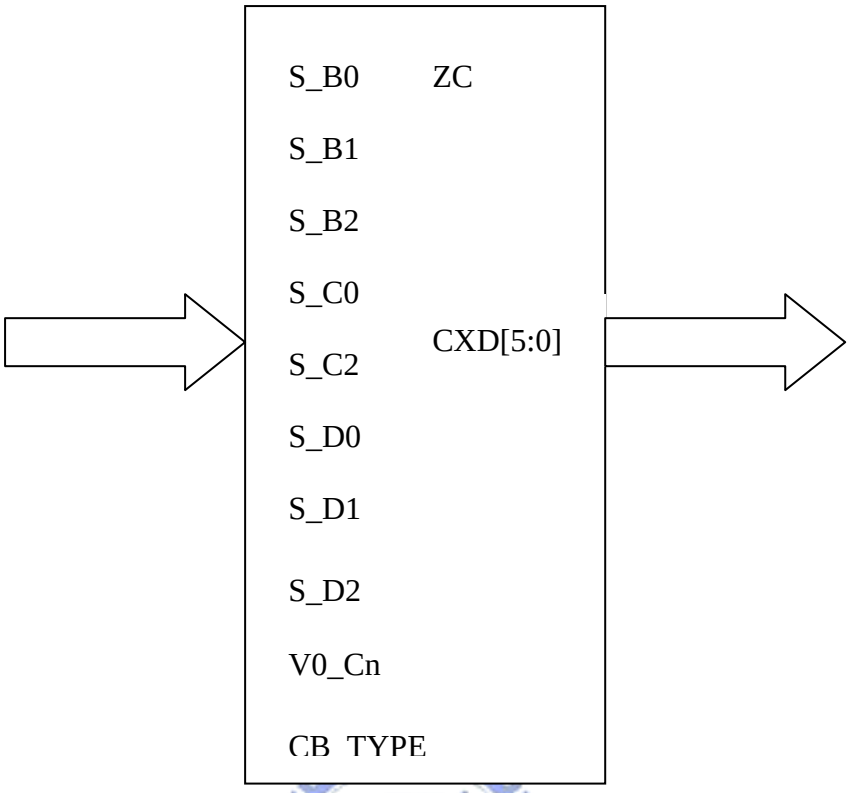


Figure 4-9 Hard-macro of zero coding

As shown in Figure 4-9, the zero coding of sample Cn is expressed as follows: (example for LL)

```
SUM_H = S_B1 + S_D1;
SUM_V = S_C0 + S_C2;
SUM_D = S_B0 + S_D0 + S_B2 + S_D2;
SUM_HV= S_B1 + S_D1 + S_C0 + S_C2;
  CXD[5]     = V0_Cn;
 if ((SUM_H == 0) & (SUM_V == 0) & (SUM_D == 0)) CXD[4:0] = 5'd0;
 if ((SUM_H == 0) & (SUM_V == 0) & (SUM_D == 1)) CXD[4:0] = 5'd1;
 if ((SUM_H == 0) & (SUM_V == 0) & (SUM_D >= 2)) CXD[4:0] = 5'd2;
 if ((SUM_H == 0) & (SUM_V == 1)              ) CXD[4:0] = 5'd3;
 if ((SUM_H == 0) & (SUM_V == 2)              ) CXD[4:0] = 5'd4;
 if ((SUM_H == 1) & (SUM_V == 0) & (SUM_D == 0)) CXD[4:0] = 5'd5;
 if ((SUM_H == 1) & (SUM_V == 0) & (SUM_D >= 1)) CXD[4:0] = 5'd6;
 if ((SUM_H == 1) & (SUM_V >= 1)              ) CXD[4:0] = 5'd7;
 if ((SUM_H == 2)                             ) CXD[4:0] = 5'd8;
```

As shown in Table 4-17, the zero coding of sample C1 is expressed by following hardware description language:

```
assign  S_T_B[0] = (P1_B_C[1])? S1_B[0] : S1_B[0]|S3_B[0];
assign  S_T_B[1] = (P1_B_C[1])? S1_B[1]|S3_B[1] : S1_B[1]|P1_S_B[1]|S3_B[1];
assign  S_T_B[2] = (P1_B_C[1])? S1_B[2]|S3_B[2] : S1_B[2]|P1_S_B[2]|S3_B[2];
assign  S_T_C[0] = (P1_B_C[1])? S1_C[0]: S1_C[0]|S3_C[0];
assign  V_T_C[1] = V0_C[1];
assign  S_T_C[2] = (P1_B_C[1])? S1_C[2]|S3_C[2] : S1_C[2]|P1_S_C[2]|S3_C[2];
assign  S_T_D[0] = (P1_B_C[1])? S1_D[0]          : S1_D[0]|S3_D[0];
assign  S_T_D[1] = (P1_B_C[1])? S1_D[1]          : S1_D[1]|S3_D[1];
assign  S_T_D[2] = (P1_B_C[1])? S1_D[2]          : S1_D[2]|S3_D[2];
```

Table 4-17 Neighbor significance states of sample C1 for ZC

Register D	Register C	Register B
S1_D[0] S1_D[0] | S3_D[0] (S_T_D[0])	S1_C[0] S1_C[0] | S3_C[0] (S_T_C[0])	S1_B[0] S1_B[0]|S3_B[0] (S_T_B[0])
S1_D[1] S1_D[1] | S3_D[1] (S_T_D[1])	V0_C[1] (V_T_C[1])	S1_B[1]|S3_B[1] S1_B[1]|P1_S_B[1] |S3_B[1] (S_T_B[1])
S1_D[2] S1_D[2] | S3_D[2] (S_T_D[2])	S1_C[2] | S3_C[2] S1_C[2]|P1_S_C[2]|S3_C[2] (S_T_C[2])	S1_B[2] | S3_B[2] S1_B[2]|P1_S_B[2] |S3_B[2] (S_T_B[2])

As shown in Table 4-18, the zero coding of sample C2 is expressed by following hardware description language:

```
assign  S_T_B[1] = (P1_B_C[2])? S1_B[1]|S3_B[1] : S1_B[1]|S3_B[1]|P1_S_B[1];
assign  S_T_B[2] = (P1_B_C[2])? S1_B[2]|S3_B[2] : S1_B[2]|S3_B[2]|P1_S_B[2];
assign  S_T_B[3] = (P1_B_C[2])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign  S_T_C[1] = (P1_B_C[2])? S1_C[1]|P1_S_C[1] :
                   S1_C[1]|S3_C[1]|P1_S_C[1]|P3_S_C[1];
assign  V_T_C[2] = V0_C[2];
assign  S_T_C[3] = (P1_B_C[2])? S1_C[3]|S3_C[3] : S1_C[3]|S3_C[3]|P1_S_C[3];
assign  S_T_D[1] = (P1_B_C[2])? S1_D[1] : S1_D[1]|S3_D[1];
assign  S_T_D[2] = (P1_B_C[2])? S1_D[2] : S1_D[2]|S3_D[2];
assign  S_T_D[3] = (P1_B_C[2])? S1_D[3] : S1_D[3]|S3_D[3];
```

Table 4-18 Neighbor significance states of sample C2 for ZC

Register D	Register C	Register B
S1_D[1] S1_D[1] | S3_D[1] (S_T_D[1])	S1_C[1]|P1_S_C[1] S1_C[1]|S3_C[1]|P1_S_C[1]|P3_S_C[1] (S_T_C[1])	S1_B[1]|S3_B[1] S1_B[1]|S3_B[1]|P1_S_B[1] (S_T_B[1])
S1_D[2] S1_D[2] | S3_D[2] (S_T_D[2])	V0_C[2] (V_T_C[2])	S1_B[2]|S3_B[2] S1_B[2]|S3_B[2]|P1_S_B[2] (S_T_B[2])
S1_D[3] S1_D[3] | S3_D[3] (S_T_D[3])	S1_C[3] | S3_C[3] S1_C[3]|S3_C[3]|P1_S_C[3] (S_T_C[3])	S1_B[3] | S3_B[3] S1_B[3]|S3_B[3]|P1_S_B[3] (S_T_B[3])

As shown in Table 4-19, the zero coding of sample C3 is expressed by following hardware description language:

```
assign  S_T_B[2] = (P1_B_C[3])? S1_B[2]|S3_B[2] : S1_B[2]|S3_B[2]|P1_S_B[2];
assign  S_T_B[3] = (P1_B_C[3])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign  S_T_B[4] = (P1_B_C[3])? S1_B[4]|S3_B[4] : S1_B[4]|S3_B[4]|P1_S_B[4];
assign  S_T_C[2] = (P1_B_C[3])? S1_C[2]|P1_S_C[2] :
                                S1_C[2]|S3_C[2]|P1_S_C[2]|P3_S_C[2];
assign  V_T_C[3] = V0_C[3];
assign  S_T_C[4] = (P1_B_C[3])? S1_C[4]|S3_C[4] : S1_C[4]|S3_C[4]|P1_S_C[4];
assign  S_T_D[2] = (P1_B_C[3])? S1_D[2] : S1_D[2]|S3_D[2];
assign  S_T_D[3] = (P1_B_C[3])? S1_D[3] : S1_D[3]|S3_D[3];
assign  S_T_D[4] = (P1_B_C[3])? S1_D[4] : S1_D[4]|S3_D[4];
```

Table 4-19 Neighbor significance states of sample C3 for ZC

Register D	Register C	Register B
S1_D[2] S1_D[2] | S3_D[2] (S_T_D[2])	S1_C[2]|P1_S_C[2] S1_C[2]|S3_C[2]|P1_S_C[2]|P3_S_C[2] (S_T_C[2])	S1_B[2]|S3_B[2] S1_B[2]|S3_B[2]|P1_S_B[2] (S_T_B[2])
S1_D[3] S1_D[3] | S3_D[3] (S_T_D[3])	V0_C[3] (V_T_C[3])	S1_B[3]|S3_B[3] S1_B[3]|S3_B[3]|P1_S_B[3] (S_T_B[3])
S1_D[4] S1_D[4] | S3_D[4] (S_T_D[4])	S1_C[4] | S3_C[4] S1_C[4]|S3_C[4]|P1_S_C[4] (S_T_C[4])	S1_B[4] | S3_B[4] S1_B[4]|S3_B[4]|P1_S_B[4] (S_T_B[4])

As shown in Table 4-20, the zero coding of sample C4 is expressed by following hardware description language:

```
assign  S_T_B[3] = (P1_B_C[4])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign  S_T_B[4] = (P1_B_C[4])? S1_B[4]|S3_B[4] : S1_B[4]|S3_B[4]|P1_S_B[4];
assign  S_T_B[5] = (P1_B_C[4])?  1'b0  :  1'b0  ;
assign  S_T_C[3] = (P1_B_C[4])?  S1_C[3]|P1_S_C[3]  :
S1_C[3]|S3_C[3]|P1_S_C[3]|P3_S_C[3];
assign  V_T_C[4] = V0_C[4];
assign  S_T_C[5] = (P1_B_C[4])?  1'b0  :  1'b0  ;
assign  S_T_D[3] = (P1_B_C[4])? S1_D[3] : S1_D[3]|S3_D[3];
assign  S_T_D[4] = (P1_B_C[4])? S1_D[4] : S1_D[4]|S3_D[4];
assign  S_T_D[5] = (P1_B_C[4])?  1'b0  :  1'b0 ;
```

Table 4-20 Neighbor significance states of sample C4 for ZC

Register D	Register C	Register B
S1_D[3] S1_D[3] | S3_D[3] (S_T_D[3])	S1_C[3]|P1_S_C[3] S1_C[3]|S3_C[3]|P1_S_C[3]|P3_S_C[3] (S_T_C[3])	S1_B[3]|S3_B[3] S1_B[3]|S3_B[3]|P1_S_B[3] (S_T_B[3])
S1_D[4] S1_D[4] | S3_D[4] (S_T_D[4])	V0_C[4] (V_T_C[4])	S1_B[4]|S3_B[4] S1_B[4]|S3_B[4]|P1_S_B[4] (S_T_B[4])
0 (S_T_D[5])	0 (S_T_C[5])	0 (S_T_B[5])

4.3.3 Sign Coding Expression of sharing between Pass1 and Pass3 for Sample C1 to C4 in Register C

If sample C1 to C4 in register C belongs to pass 1 or pass3 and their magnitude bit value are one, they must be coded by sign-coding (SC). The hard-macro of sign coding is shown in Figure 4-10.

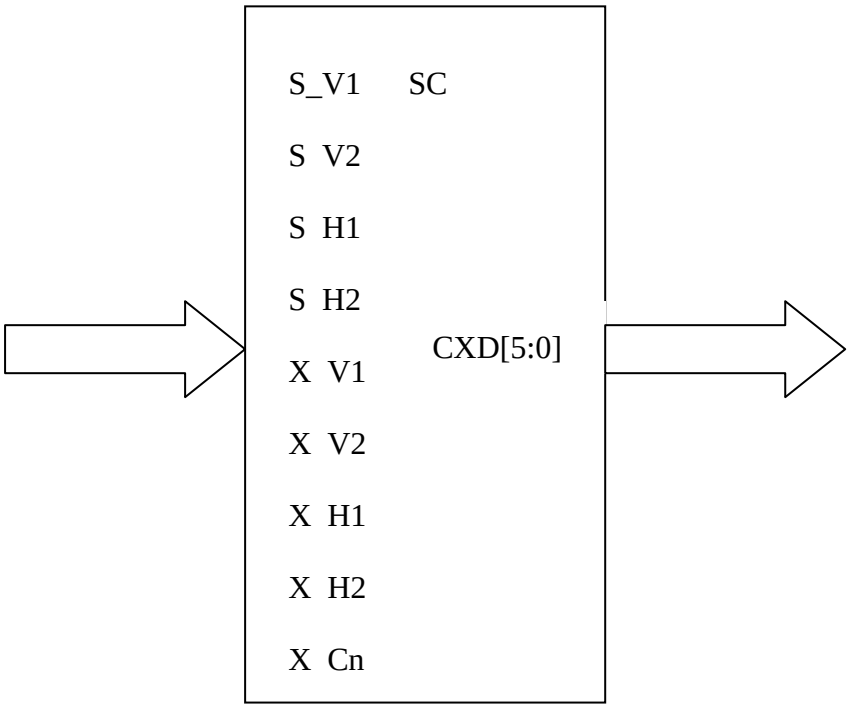


Figure 4-10 Hard-macro of sign coding

As shown in Figure 4-10, the sign coding of sample Cn is expressed as follows:

```
always @(S_V1 or S_V2 or X_V1 or X_V2)
    begin
    casex ({S_V1, S_V2, X_V1, X_V2})
        4'b00xx : SUM_V = 2'b00;
        4'b01x0 : SUM_V = 2'b01;
        4'b01x1 : SUM_V = 2'b11;
        4'b100x : SUM_V = 2'b01;
        4'b101x : SUM_V = 2'b11;
        4'b1100 : SUM_V = 2'b01;
        4'b1101 : SUM_V = 2'b00;
        4'b1110 : SUM_V = 2'b00;
        4'b1111 : SUM_V = 2'b11;
    endcase
end
// -1 = 2'b11, 0 = 2'b00, 1 = 2'b01
always @(S_H1 or S_H2 or X_H1 or X_H2)
    begin
```

```
    casex ({S_H1, S_H2, X_H1, X_H2})
        4'b00xx : SUM_H = 2'b00;
        4'b01x0 : SUM_H = 2'b01;
        4'b01x1 : SUM_H = 2'b11;
        4'b100x : SUM_H = 2'b01;
        4'b101x : SUM_H = 2'b11;
        4'b1100 : SUM_H = 2'b01;
        4'b1101 : SUM_H = 2'b00;
        4'b1110 : SUM_H = 2'b00;
        4'b1111 : SUM_H = 2'b11;
    endcase
end

always @(SUM_H or SUM_V or X_Cn) begin
    case ({SUM_H, SUM_V})
        4'b0000 : CXD = { X_Cn, 5'd9};   //    0    0
        4'b0001 : CXD = { X_Cn, 5'd10}; //    0    1
        4'b0011 : CXD = {!X_Cn, 5'd10}; //    0 -1
        4'b0111 : CXD = { X_Cn, 5'd11}; //    1 -1
        4'b1101 : CXD = {!X_Cn, 5'd11}; // -1    1
        4'b0100 : CXD = { X_Cn, 5'd12}; //    1    0
        4'b1100 : CXD = {!X_Cn, 5'd12}; // -1    0
        4'b0101 : CXD = { X_Cn, 5'd13}; //    1    1
        4'b1111 : CXD = {!X_Cn, 5'd13}; // -1 -1
        default : CXD = { X_Cn, 5'd31};
    endcase
end
```

As shown in Table 4-21 and Table 4-22, the sign coding of sample C1 is expressed by following hardware description language:

```
assign    S_V1_1 = (P1_B_C[1])? S1_C[0] : S1_C[0]|S3_C[0];
assign    S_V2_1 = (P1_B_C[1])? S1_C[2]|S3_C[2] :
                                S1_C[2]|S3_C[2]|P1_S_C[2];
assign    S_H1_1 = (P1_B_C[1])? S1_B[1]|S3_B[1] :
                                S1_B[1]|S3_B[1]|P1_S_B[1];
assign    S_H2_1 = (P1_B_C[1])? S1_D[1] : S1_D[1]|S3_D[1];
assign    X_V1_1 = X0_C[0]; // S1_D[n]=S1_C[n]|P1_S_C[n]
assign    X_V2_1 = X0_C[2]; // S3_D[n]=S3_C[n]|P3_S_C[n]
assign    X_H1_1 = X0_B[1];
```

```
assign    X_H2_1 = X0_D[1];
assign    X_C_1 = X0_C[1];
```

Table 4-21 Neighbor significance states of sample C1 for SC

Register D	Register C	Register B
X	S1_C[0] S1_C[0] | S3_C[0] (S_V1_1)	X
S1_D[1] S1_D[1] | S3_D[1] (S_H2_1)	(Sample C1)	S1_B[1]|S3_B[1] S1_B[1]|S3_B[1]|P1_S_B[1] (S_H1_1)
X	S1_C[2] | S3_C[2] S1_C[2]|S3_C[2]|P1_S_C[2] (S_V2_1)	X

Table 4-22 Neighbor sign states of sample C1 for SC

Register D	Register C	Register B
X	X0_C[0] (X_V1_1)	X
X0_D[1] (X_H2_1)	X0_C[1] (X_C_1)	X0_B[1] (X_H1_1)
X	X0_C[2] (X_V2_1)	X

As shown in Table 4-23 and Table 4-24, the sign coding of sample C2 is expressed by following hardware description language:

```
assign    S_V1_2 = (P1_B_C[2])? S1_C[1]|P1_S_C[1] :
                                S1_C[1]|S3_C[1]|P1_S_C[1]|P3_S_C[1];
assign    S_V2_2 = (P1_B_C[2])? S1_C[3]|S3_C[3] :
                                S1_C[3]|S3_C[3]|P1_S_C[3];
assign    S_H1_2 = (P1_B_C[2])? S1_B[2]|S3_B[2] :
                                S1_B[2]|S3_B[2]|P1_S_B[2];
    assign    S_H2_2 = (P1_B_C[2])? S1_D[2] : S1_D[2]|S3_D[2];
    assign    X_V1_2 = X0_C[1];
    assign    X_V2_2 = X0_C[3];
    assign    X_H1_2 = X0_B[2];
    assign    X_H2_2 = X0_D[2];
```

assign X_C_2 = X0_C[2];

Table 4-23 Neighbor significance states of sample C2 for SC

Register D	Register C	Register B
X	S1_C[1]|P1_S_C[1] S1_C[1]|S3_C[1]|P1_S_C[1]|P3_S_C[1] (S_V1_2)	X
S1_D[2] S1_D[2] | S3_D[2] (S_H2_2)	(Sample C2)	S1_B[2]|S3_B[2] S1_B[2]|S3_B[2]|P1_S_B[2] (S_H1_2)
X	S1_C[3] | S3_C[3] S1_C[3]|S3_C[3]|P1_S_C[3] (S_V2_2)	X

Table 4-24 Neighbor sign states of sample C2 for SC

Register D	Register C	Register B
X	X0_C[1] (X_V1_2)	X
X0_D[2] (X_H2_2)	X0_C[2] (X_C_2)	X0_B[2] (X_H1_2)
X	X0_C[3] (X_V2_2)	X

As shown in Table 4-25 and Table 4-26, the sign coding of sample C3 is expressed by following hardware description language:

```
assign  S_V1_3 = (P1_B_C[3])? S1_C[2]|P1_S_C[2] :
                              S1_C[2]|S3_C[2]|P1_S_C[2]|P3_S_C[2];
assign  S_V2_3 = (P1_B_C[3])? S1_C[4]|S3_C[4] :
                              S1_C[4]|S3_C[4]|P1_S_C[4];
assign  S_H1_3 = (P1_B_C[3])? S1_B[3]|S3_B[3] :
                              S1_B[3]|S3_B[3]|P1_S_B[3];
assign  S_H2_3 = (P1_B_C[3])? S1_D[3] : S1_D[3]|S3_D[3];
assign  X_V1_3 = X0_C[2];
assign  X_V2_3 = X0_C[4];
assign  X_H1_3 = X0_B[3];
assign  X_H2_3 = X0_D[3];
assign  X_C_3 = X0_C[3];
```

Table 4-25 Neighbor significance states of sample C3 for SC

Register D	Register C	Register B
X	S1_C[2]|P1_S_C[2] S1_C[2]|S3_C[2]|P1_S_C[2]|P3_S_C[2] (S_V1_3)	X
S1_D[3] S1_D[3] | S3_D[3] (S_H2_3)	(Sample C3)	S1_B[3]|S3_B[3] S1_B[3]|S3_B[3]|P1_S_B[3] (S_H1_3)
X	S1_C[4] | S3_C[4] S1_C[4]|S3_C[4]|P1_S_C[4] (S_V2_3)	X

Table 4-26 Neighbor sign states of sample C3 for SC

Register D	Register C	Register B
X	X0_C[2] (X_V1_3)	X
X0_D[3] (X_H2_3)	V0_C[3] (X_C_3)	X0_B[3] (X_H1_3)
X	X0_C[4] (X_V2_3)	X

As shown in Table 4-27 and Table 4-28, the sign coding of sample C4 is expressed by following hardware description language:

```
assign    S_V1_4 = (P1_B_C[4])? S1_C[3]|P1_S_C[3] :
                                S1_C[3]|S3_C[3]|P1_S_C[3]|P3_S_C[3];
   assign    S_V2_4 = 1'b0;
assign    S_H1_4 = (P1_B_C[4])? S1_B[4]|S3_B[4] :
                                S1_B[4]|S3_B[4]|P1_S_B[4];
   assign    S_H2_4 = (P1_B_C[4])? S1_D[4] : S1_D[4]|S3_D[4];
   assign    X_V1_4 = X0_C[3];
   assign    X_V2_4 = 1'b0     ;
   assign    X_H1_4 = X0_B[4];
   assign    X_H2_4 = X0_D[4];
   assign    X_C_4 = X0_C[4];
```

Table 4-27 Neighbor significance states of sample C4 for SC

Register D	Register C	Register B
X	S1_C[3]|P1_S_C[3] S1_C[3]|S3_C[3]|P1_S_C[3]|P3_S_C[3] (S_V1_4)	X
S1_D[4] S1_D[4] | S3_D[4] (S_H2_4)	(Sample C4)	S1_B[4]|S3_B[4] S1_B[4]|S3_B[4]|P1_S_B[4] (S_H1_4)
X	0 (S_V2_4)	X

Table 4-28 Neighbor sign states of sample C4 for SC

Register D	Register C	Register B
X	X0_C[3] (X_V1_4)	X
X0_D[4] (X_H2_4)	X0_C[4] (X_C_4)	X0_B[4] (X_H1_4)
X	0 (X_V2_4)	X

4.3.4 Run-Length Coding Expression for sample C1 to C4 in Register C for Pass3

The output signal (P3_RLC0,P3_RLC1) of Run length code is obtain by all coding pass generation described in section 4-2. When P3_RLC0 is one, the context-decision of run length code is expressed by following hardware description language:

```
assign  CXD_P3_RLC_0   = {!P3_RLC0, 5'd17};
```

When P3_RLC1 is one, the context-decision of run length code with uniform coding is expressed by following hardware description language:

```
assign  CXD_P3_RLC_1   = {P3_RLC1, 5'd17};
assign  CXD_P3_RLC1_UC0   = { P3_RLC1_UC1, 5'd18};
assign  CXD_P3_RLC1_UC1   = { P3_RLC1_UC0, 5'd18};
```

CHAPTER 5 ARCHITECTURE DESIGN AND EXPERIMENT RESULTS

In this chapter, we introduce overall system architecture of proposed pass parallel context-formation (CF) encoder, detailed internal signals of context formation encoder, analysis of critical path of system, design flow, verification and experiment results. According to three methods described in previous chapter, we divided overall system architecture into four stages. These topics above description are discussed in next sections.

5.1 Overall Architecture of Pass-Parallel Context Formation Encoder with High-Speed and Small-Area

As shown in Figure 5-1, the top level block of overall system architecture include four stages as follows:

The first stage: Load data from sram to shift register A.

The second stage: Shift the data of register A to register B and execute Dual Column Pass1 generation.

The third stage: Shift the data of register B to register C, execute all coding pass generation and sample parallel column-based coding.

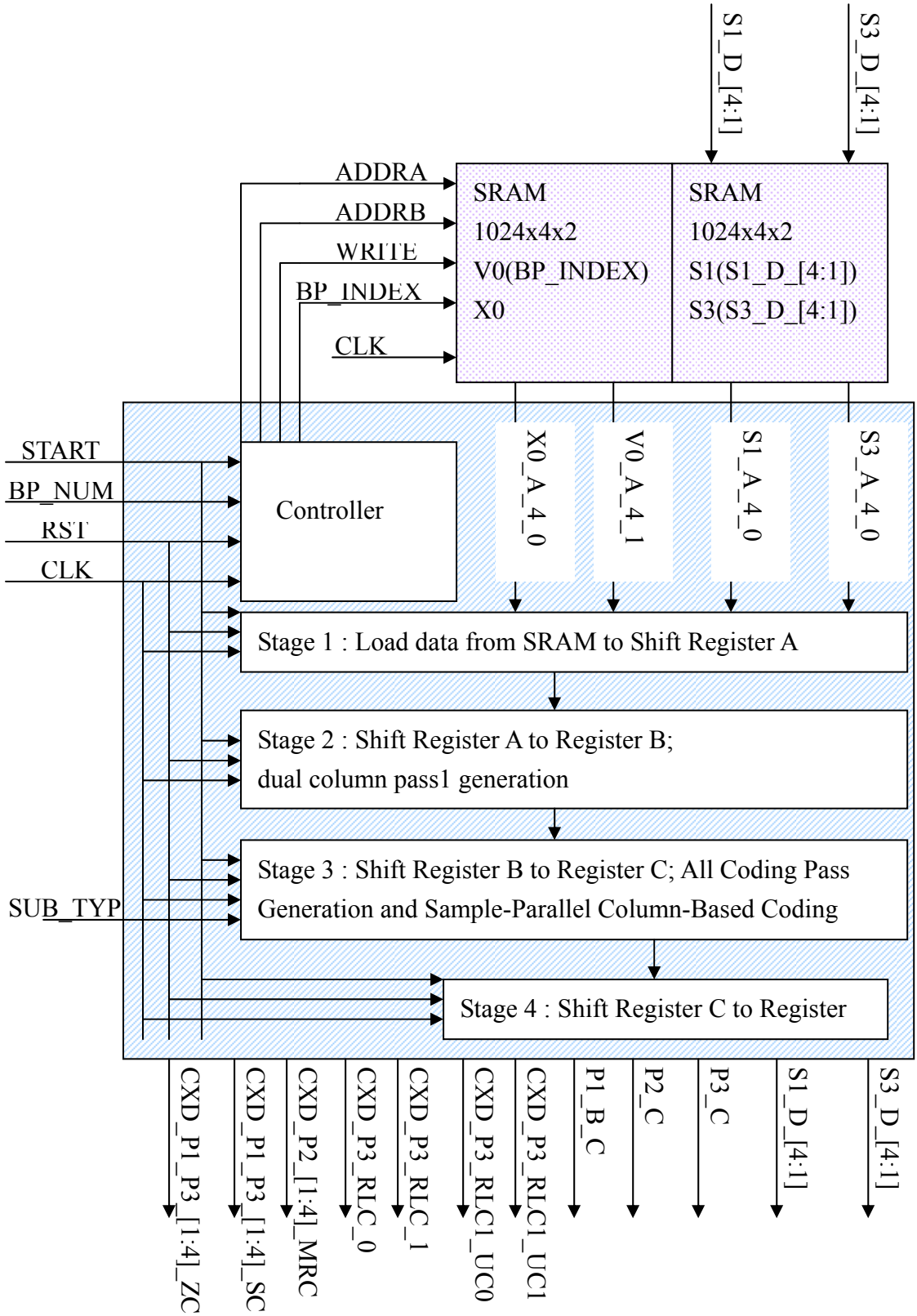


Figure 5-1 Overall architecture of Pass-Parallel Context Formation Encoder with high speed and small area

The fourth stage: Shift the data of register C to register D and store pass1 significance change or pass3 significance change to register D.

The code block of 64x64 in dimension needs sram (1024x4 bits) of magnitude state (V0), sram (1024x4 bits) of sign state, sram (1024x4x2 bits) of pass1 significance change (S1) or pass3 significance change (S3). Suppose the system can read 5 bits (data of one column, the fourth data of one column in previous stripe) once from sram (S1, S3) and write 4bits (significance change data of one column) once to sram (S1, S3).

Controller manages the overall the data flow of the system, and generates writing and reading address for pass3 significance state memory (S3), pass1 significance state memory (S1), magnitude state memory (V0), and sign state memory (X0). It controls the pipeline architecture. For example, when it sent the four reading address at the beginning, must generate writing address to move the data of the first column to memory.

The input signals of overall system architecture include ‘START’, ‘BP_NUM‘, ‘RST’, ‘CLK’, ‘SUB_TYP’, ‘X0_A_4_0’, ‘V0_A_4_1’, ‘S1_A_4_0’, ‘S3_A_4_0’. Function of input signal can be described, as shown in Table5-1.

Table 5-1 Function of input signal, proposed architecture

Input signal	Function Description
START	-If the signal is low to high, initiate the value of shift register to zero. If the signal is high to low and the signal ‘clk’ is low to high, address begin to count. Then Shift the data of the first column to the register A.
BP_NUM	- If the data of dwt coefficient is expressed by magnitude-sign format (8bits). BP_NUM is set to BP_NUM[3:0]= 4'd7

RST	- Reset pin. This is an active high signal. -If the signal is low, initiate the value of shift register to zero. - If the signal is low to high and the signal 'START' is high to low, internal counter begin to count.
CLK	- System clock of synchronization. - Force system clk to sram, controller and all shift register.
SUB_TYP	- SUB_TYP [1:0]=00, System used LL sub-band. - SUB_TYP [1:0]=01, System used LH sub-band. - SUB_TYP [1:0]=10, System used HL sub-band. - SUB_TYP [1:0]=11, System used HH sub-band.
X0_A_4_0[4:0]	-When controller send read address to memory, memory send five sign bits data (four bits from current stripe column and the fourth bit from previous stripe column) to shift register A
V0_A_4_1	-When controller send read address to memory, memory send five magnitude bits data (four bits from current stripe column and the fourth bit from previous stripe column) to shift register A
S1_A_4_0	- When controller send read address to memory, memory send five pass1 significance bits data (four bits from current stripe column and the fourth bit from previous stripe column) to shift register A
S3_A_4_0	- When controller send read address to memory, memory send five pass3 significance bits data (four bits from current stripe column and the fourth bit from previous stripe column) to shift register A

The output signals of overall system architecture include ‘CXD_P1_P3_1_ZC’,‘CXD_P1_P3_2_ZC’,‘CXD_P1_P3_3_ZC’, ‘CXD_P1_P3_4_ZC’,‘CXD_P1_P3_1_SC‘,‘CXD_P1_P3_2_SC‘, ‘CXD_P1_P3_3_SC‘,‘CXD_P1_P3_4_SC‘,‘CXD_P2_1_MRC’,‘CXD_P2_2_MRC’, ‘CXD_P2_3_MRC’,‘CXD_P2_4_MRC’,‘CXD_P3_RLC_0’,‘CXD_P3_RLC_1’, ‘CXD_P3_RLC1_UC0’,‘CXD_P3_RLC1_UC1’,‘P1_B_C’,‘P2_C’, ‘P3_C’,‘S1_D_[4:1]’,‘S3_D_[4:1]’,‘ADDRA’,‘ADDRB’,‘WRITE’,‘BP_INDEX’. Function of output signal can be described, as shown in Table5-2.

Table 5-2 Function of output signal, proposed architecture

Output signal	Function Description
CXD_P1_P3_1_ZC	-When P1_B_C[1] is high, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass1 zero coding of the first sample of coded column -When P1_B_C[1] is low, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass3 zero coding of the first sample of coded column
CXD_P1_P3_2_ZC	-When P1_B_C[2] is high, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass1 zero coding of the second sample of coded column -When P1_B_C[2] is low, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass3 zero coding of the second sample of coded column
CXD_P1_P3_3_ZC	-When P1_B_C[3] is high, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass1 zero coding of the third

	sample of coded column -When P1_B_C[3] is low, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass3 zero coding of the third sample of coded column
CXD_P1_P3_4_ZC	-When P1_B_C[4] is high, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass1 zero coding of the fourth sample of coded column -When P1_B_C[4] is low, CXD_P1_P3_1_ZC[5:0] send six bits context-decision(D,CX) for pass3 zero coding of the fourth sample of coded column
CXD_P1_P3_1_SC	-When P1_B_C[1] is high, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass1 sign coding of the first sample of coded column -When P1_B_C[1] is low, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass3 sign coding of the first sample of coded column
CXD_P1_P3_2_SC	-When P1_B_C[2] is high, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass1 sign coding of the second sample of coded column -When P1_B_C[2] is low, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass3 sign coding of the second sample of coded column
CXD_P1_P3_3_SC	-When P1_B_C[3] is high, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass1 sign coding of the third sample of coded column

	-When P1_B_C[3] is low, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass3 sign coding of the third sample of coded column
CXD_P1_P3_4_SC	-When P1_B_C[4] is high, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass1 sign coding of the fourth sample of coded column -When P1_B_C[4] is low, CXD_P1_P3_1_SC[5:0] send six bits context-decision(D,CX) for pass3 sign coding of the fourth sample of coded column
CXD_P2_1_MRC	-CXD_P2_1_MRC [5:0] send six bits context-decision(D,CX) for pass2 magnitude refinement coding of the first sample of coded column
CXD_P2_2_MRC	-CXD_P2_1_MRC [5:0] send six bits context-decision(D,CX) for pass2 magnitude refinement coding of the second sample of coded column
CXD_P2_3_MRC	-CXD_P2_1_MRC [5:0] send six bits context-decision(D,CX) for pass2 magnitude refinement coding of the third sample of coded column
CXD_P2_4_MRC	-CXD_P2_1_MRC [5:0] send six bits context-decision(D,CX) for pass2 magnitude refinement coding of the fourth sample of coded column
CXD_P3_RLC_0	- CXD_P3_RLC_0 [5:0] send six bits context-decision (D,CX)=(0,17) for pass3 run-length coding and values of magnitude bit of coded column are all zero.
CXD_P3_RLC_1	- CXD_P3_RLC_1 [5:0] send six bits context-decision

	(D,CX)=(1,17) for pass3 run-length coding and values of magnitude bit of coded column are not all zero.
CXD_P3_RLC1_UC0	-CXD_P3_RLC1_UC0[5:0] send six bits uniform coding context-decision(UC0,18) for pass3 run-length coding and values of magnitude bit of coded column are not all zero. -When magnitude bit of the first sample of coded column is one, then the value of UC0 is zero. -When magnitude bit of the second sample of coded column is one, then the value of UC0 is one. -When magnitude bit of the third sample of coded column is one, then the value of UC0 is zero. -When magnitude bit of the second sample of coded column is one, then the value of UC0 is one.
CXD_P3_RLC1_UC1	CXD_P3_RLC1_UC0[5:0] send six bits uniform coding context-decision(UC1,18) for pass3 run-length coding and values of magnitude bit of coded column are not all zero. -When magnitude bit of the first sample of coded column is one, then the value of UC1 is zero. -When magnitude bit of the second sample of coded column is one, then the value of UC1 is zero. -When magnitude bit of the third sample of coded column is one, then the value of UC1 is one. -When magnitude bit of the second sample of coded column is one, then the value of UC1 is one.
P1_B_C	-P1_B_C[4:1] send four bits information for Pass1 type stored

	in the register C of the third stage. Four bits information can be generated by executing dual column pass1 generation in register B of the second stage.
P2_C	-P2_C[4:1] send four bits information for Pass2 type generated in the register C of the third stage
P3_C	-P3_C[4:1] send four bits information for Pass3 type generated in the register C of the third stage
S1_D_[4:1]	-S1_D_[4:1] send four bits information of pass1 significance change stored in the register S1_D of the fourth stage. Four bits information can be generated by executing both original significance states(S1_C) in register C and pass1 significance state change(P1_S_C) generated in register C 'OR' operation.
S3_D_[4:1]	-S1_D_[4:1] send four bits information of pass3 significance change stored in the register S3_D of the fourth stage. Four bits information can be generated by executing both original significance states(S3_C) in register C and pass3 significance state change(P3_S_C) generated in register C 'OR' operation.
ADDRA	- ADDRA[9:0] send ten bits information for reading columns data in memory to shift register D. Ten bits information can be generated by controller.
ADDRB	- ADDRB[9:0] send ten bits information for writing columns data in shift register D to memory. Ten bits information can be generated by controller.
WRITE	- WRITE send one bit information for writing columns data in shift register D to memory. One bit information can be

	generated by controller.
BP_INDEX	-BP_INDEX [3:0] send four bits information for indicating current processing bit-plane. -When BP_INDEX [3:0]=0, indicate the sixth bit-plane(MSB).

The hardware description language of top level of overall system architecture is expressed in appendix A.1.

5.2 Detailed Internal Structure of Pass-Parallel Context Formation Encoder with High-Speed and Small-Area

As showed in Figure 5-2, the detailed internal structure of context formation encoder includes shift register A block circuit, shift register B block circuit, shift register C block circuit, shift register D block circuit, dual column pass1 generation block circuit, all coding pass generation circuit and sample parallel column-based coding block circuit. The internal signal flow of context formation encoder is shown in Figure 5-2. The neighbor significance states needed for dual column pass1 generation include S1_A[4:0], S3_A[4:0], S1_B[4:0], S3_B[4:0], V0_B[4:0], P1_S_C[4:1]. The neighbor significance states needed for sample-parallel column-based coding include S1_B[4:0], S3_B[4:0], P1_S_B[4:1], P1_S_C[4:1], P3_S_C[4:1], S1_C[4:0],S3_C[4:0], V0_C[4:1], X0_C[4:0], S1_D[4:0], S3_D[4:0]. The detailed description of dual column pass1 generation block circuit is discussed in section 5.2.1. The detailed description of all coding pass generation block circuit is discussed in section 5.2.2. The detailed description of all coding pass generation block circuit is discussed in section 5.2.3.

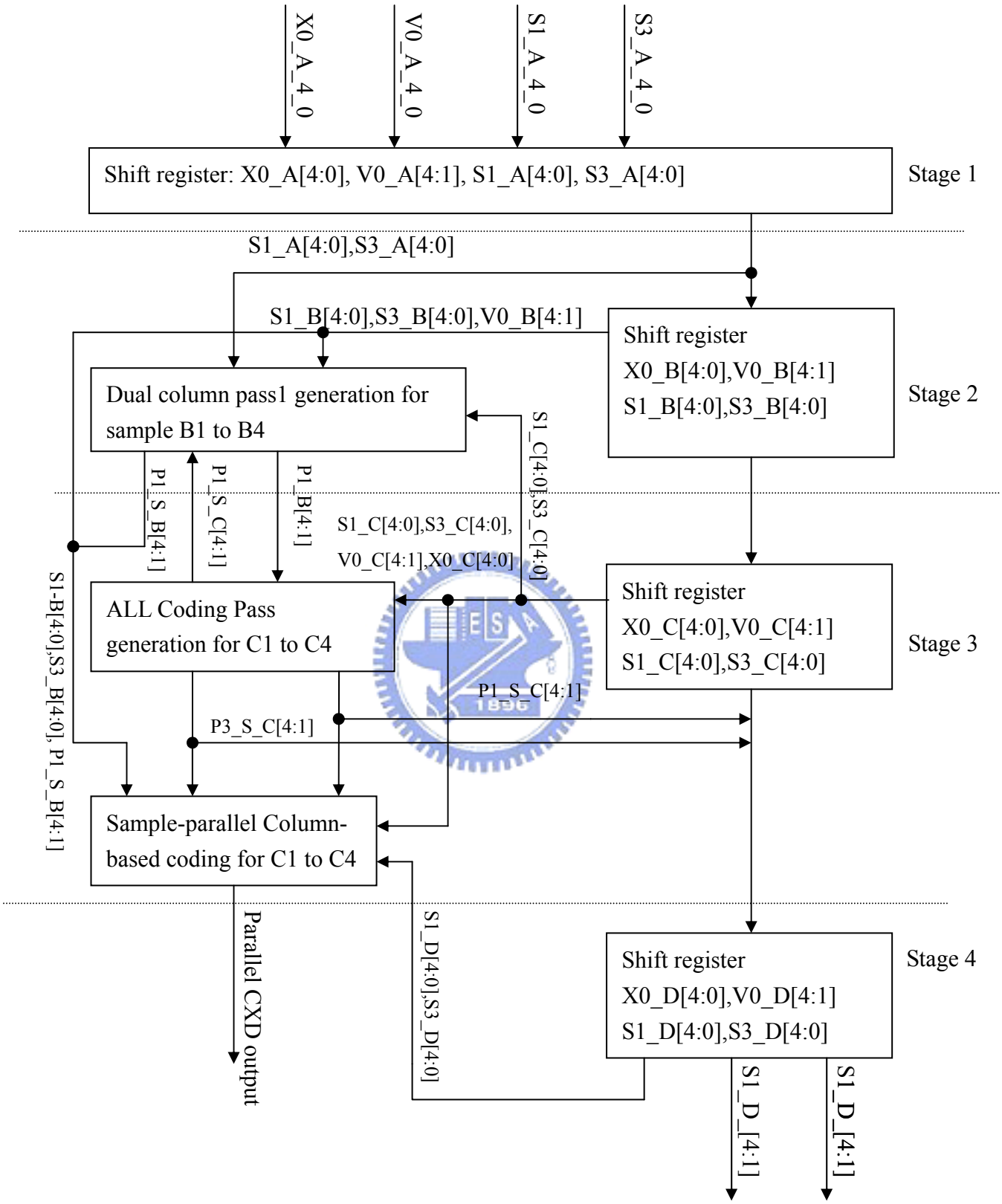


Figure 5-2 Detailed internal structure of Pass-Parallel Context Formation Encoder with high speed and small area

5.2.1 Detailed Hardware Description of the Dual Column Pass1 Generation

As showed in Figure 5-3, the detailed internal structure of dual column pass generation included pre-processing pass1 generation, horizontal pass1 generation and vertical pass1 generation. The input signals of dual pass1 generation circuit include P1_S_C[4:1], S1_A[4:0], S3_A[4:0], S1_B[4:0], S3_B[4:0], V0_B[4:1], S1_C[4:0], S3_C[4:0]. The output signals of dual pass1 generation circuit include P1_B[4:1], P1_S_B[4:1].

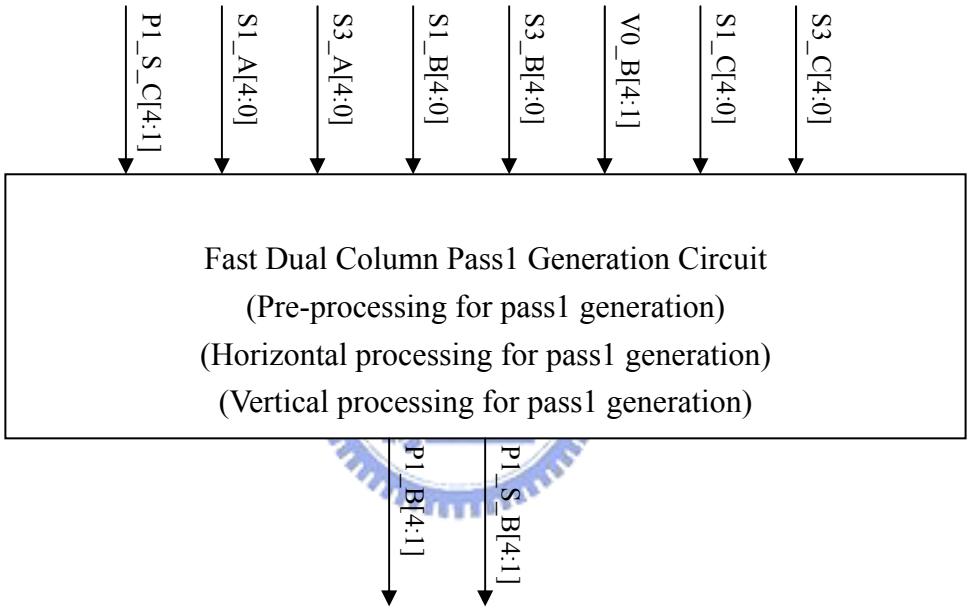


Figure 5-3 Detailed block circuit of Dual Column Pass1 generation

The hardware description language of dual column pass1 generation is expressed in appendix A.2.

5.2.2 Detailed Hardware Description of all Coding Pass and Significance Change Generation

The hardware description language of all coding pass generation and significance change is expressed in appendix A.3.

5.2.3 Detailed Hardware Description of Sample-Parallel Column-Based Coding

The hardware description language of the magnitude refinement coding is expressed in appendix A.4.

The hardware description language of the zero coding for pass1 coding operation or pass3 coding operation is expressed in appendix A.5.

The hardware description language of the sign coding for pass1 coding operation or pass3 coding operation is expressed in appendix A.6.

The hardware description language of the run-length code 0 is expressed as follows:

```
module      RLC_COLUMN_0(
              CXD_RLC,

              CXD_P3_RLC
            );
output[5:0] CXD_RLC;
input       CXD_P3_RLC;
wire  [5:0] CXD_RLC;
assign CXD_RLC  = {CXD_P3_RLC, 5'd17};
endmodule
```

The hardware description language of the run-length code 1 is expressed as follows:

```
module      RLC_COLUMN_1(
              CXD_RLC,

              CXD_P3_RLC
            );
output[5:0] CXD_RLC;
```

```
input          CXD_P3_RLC;
wire   [5:0]   CXD_RLC;
assign CXD_RLC   = {CXD_P3_RLC, 5'd17};
endmodule
```

The hardware description language of the run-length code 1 with uniform coding 0 is expressed as follows:

```
module       RLC1_UC0(
               CXD_UC,

               CXD_P3_UC
             );
output [5:0]  CXD_UC;
input         CXD_P3_UC;
wire   [5:0]  CXD_UC;
assign CXD_UC    = {CXD_P3_UC, 5'd18};
endmodule
```

The hardware description language of the run-length code 1 with uniform coding 1 is expressed as follows:

```
module       RLC1_UC1(
               CXD_UC,

               CXD_P3_UC
             );
output [5:0]  CXD_UC;
input         CXD_P3_UC;
wire   [5:0]  CXD_UC;
assign CXD_UC    = {CXD_P3_UC, 5'd18};
endmodule
```

5.3 Analysis of critical path of CF Overall Architecture Proposed

Top circuit of pass-parallel context formation encoder is shown in Figure 5-4 for searching gate delay of the second stage.

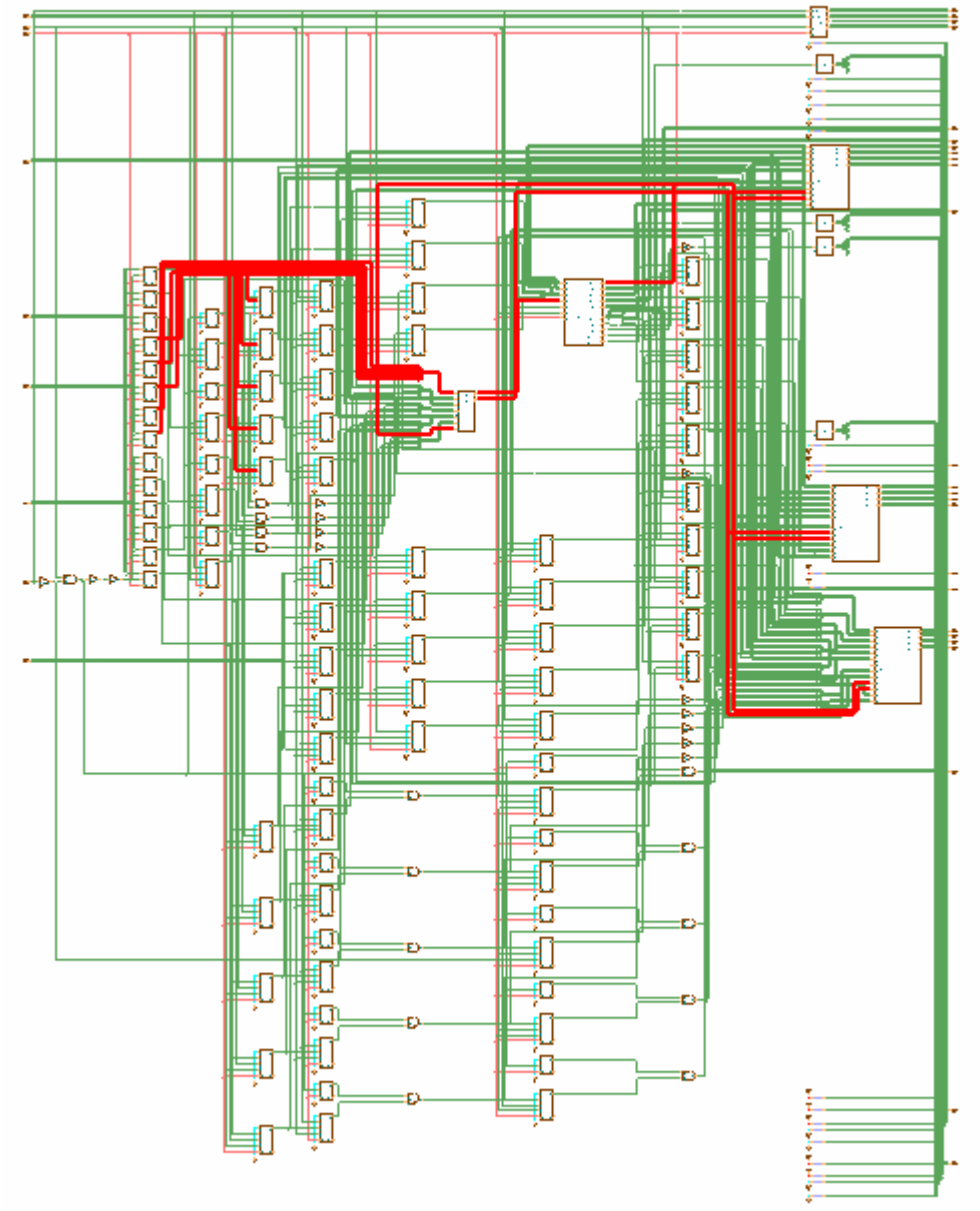


Figure 5-4 Top circuit of CF for searching gate delay of the second stage

As shown in Figure 5-5, 5-6, 5-7, the gate delay of dual column pass1 generation of the second stage is 9 gate counts.

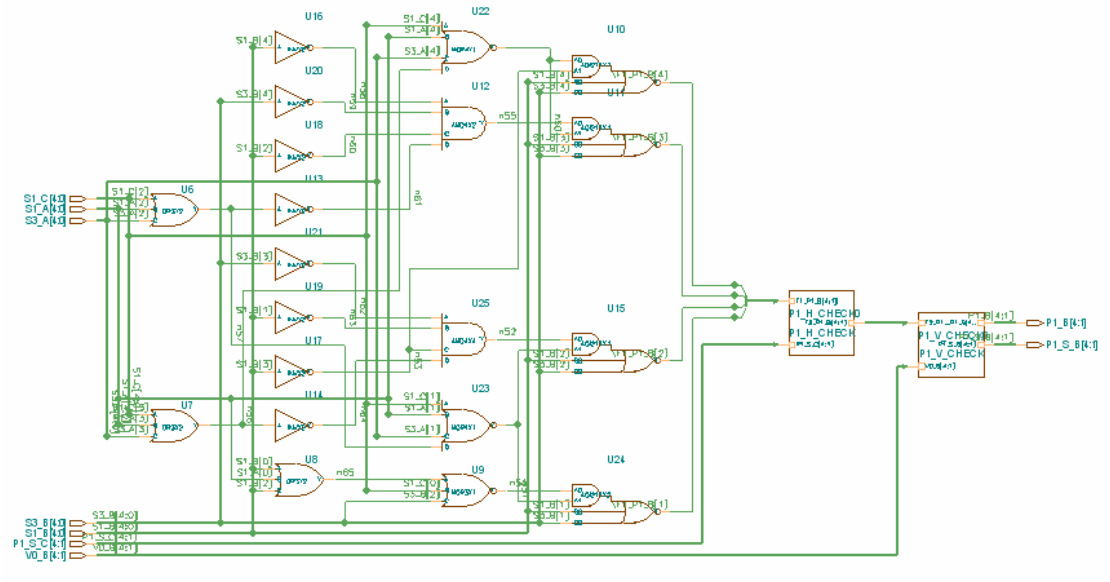


Figure 5-5 Circuit of pre-processing

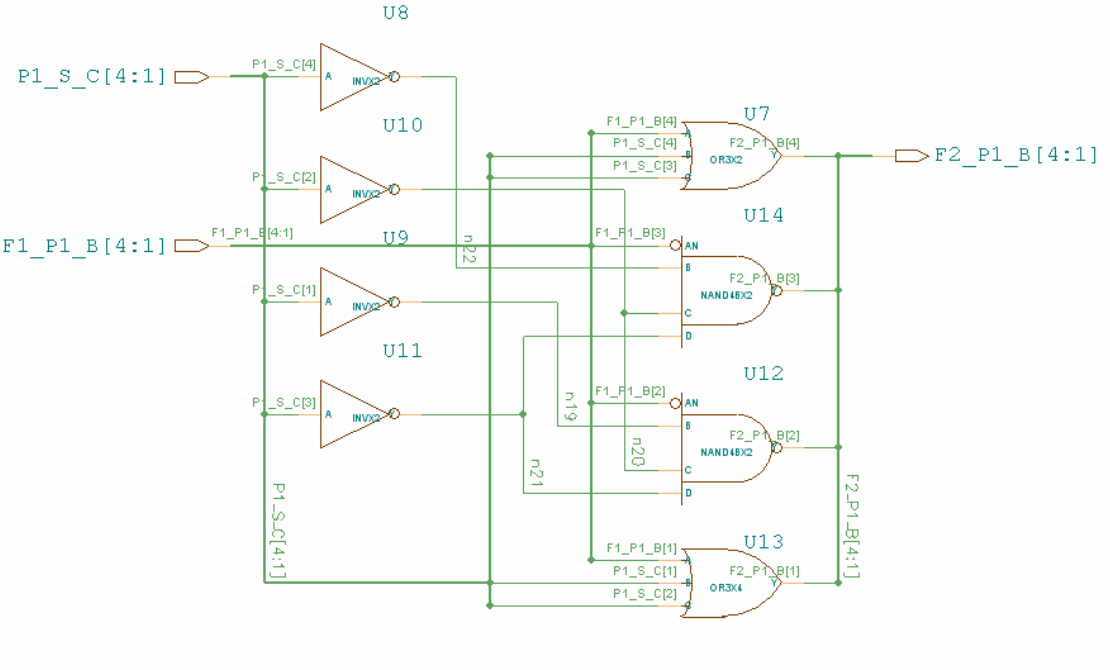


Figure 5-6 Circuit of horizontal processing

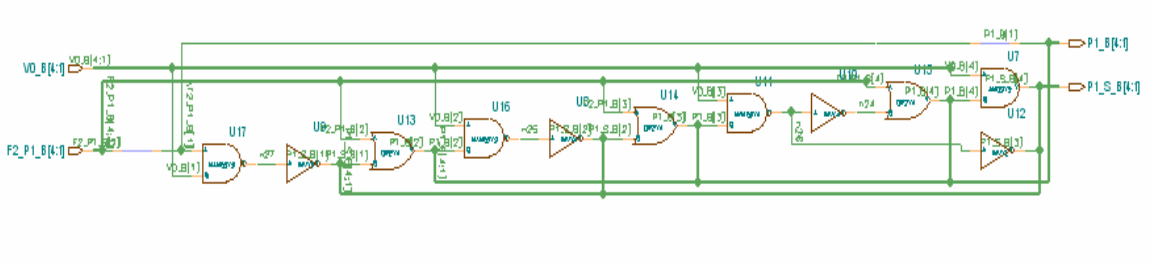


Figure 5-7 Circuit of vertical processing

Top circuit of pass-parallel context formation encoder is shown in Figure 5-8 for searching gate delay of the third stage.

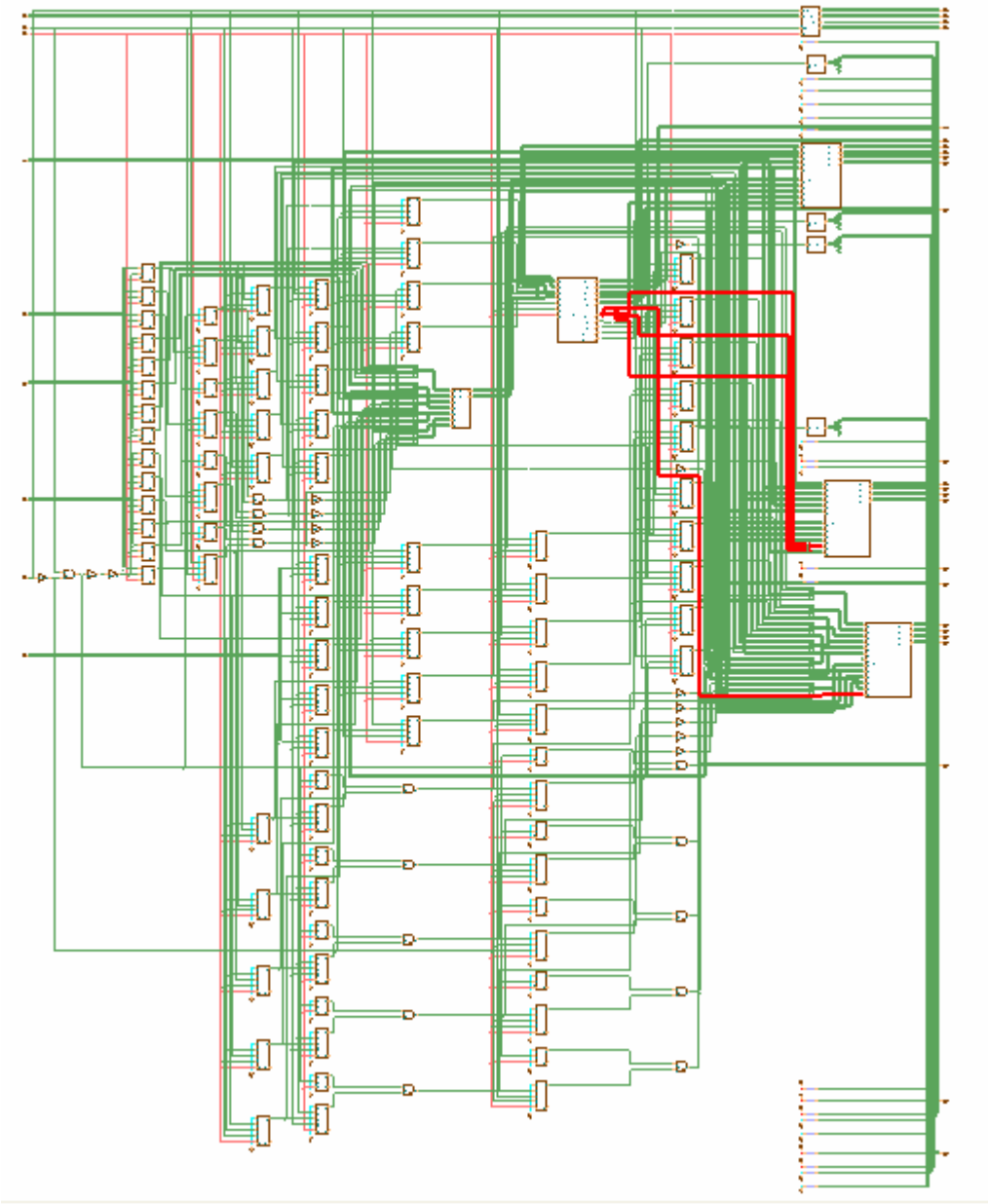


Figure 5-8 Top level circuit for searching gate delay of the third stage

As shown in Figure 5-9, the gate delay of all coding pass and significance change generation is 5 gate counts.

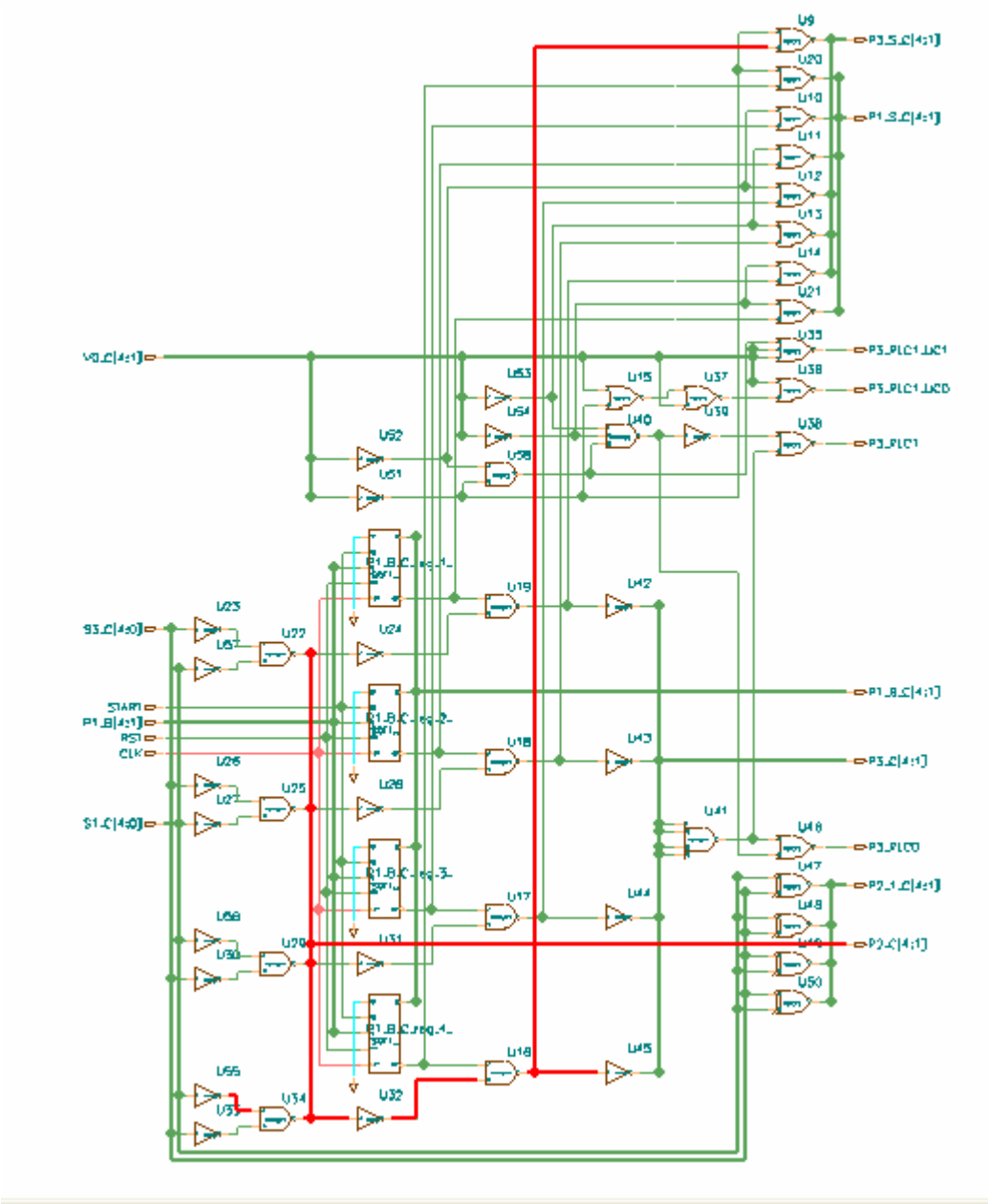


Figure 5-9 Circuit of all coding pass and significance change generation

As shown in Figure 5-10, the gate delay of selecting pass1 or pass3 is 5 gate counts for zero coding.

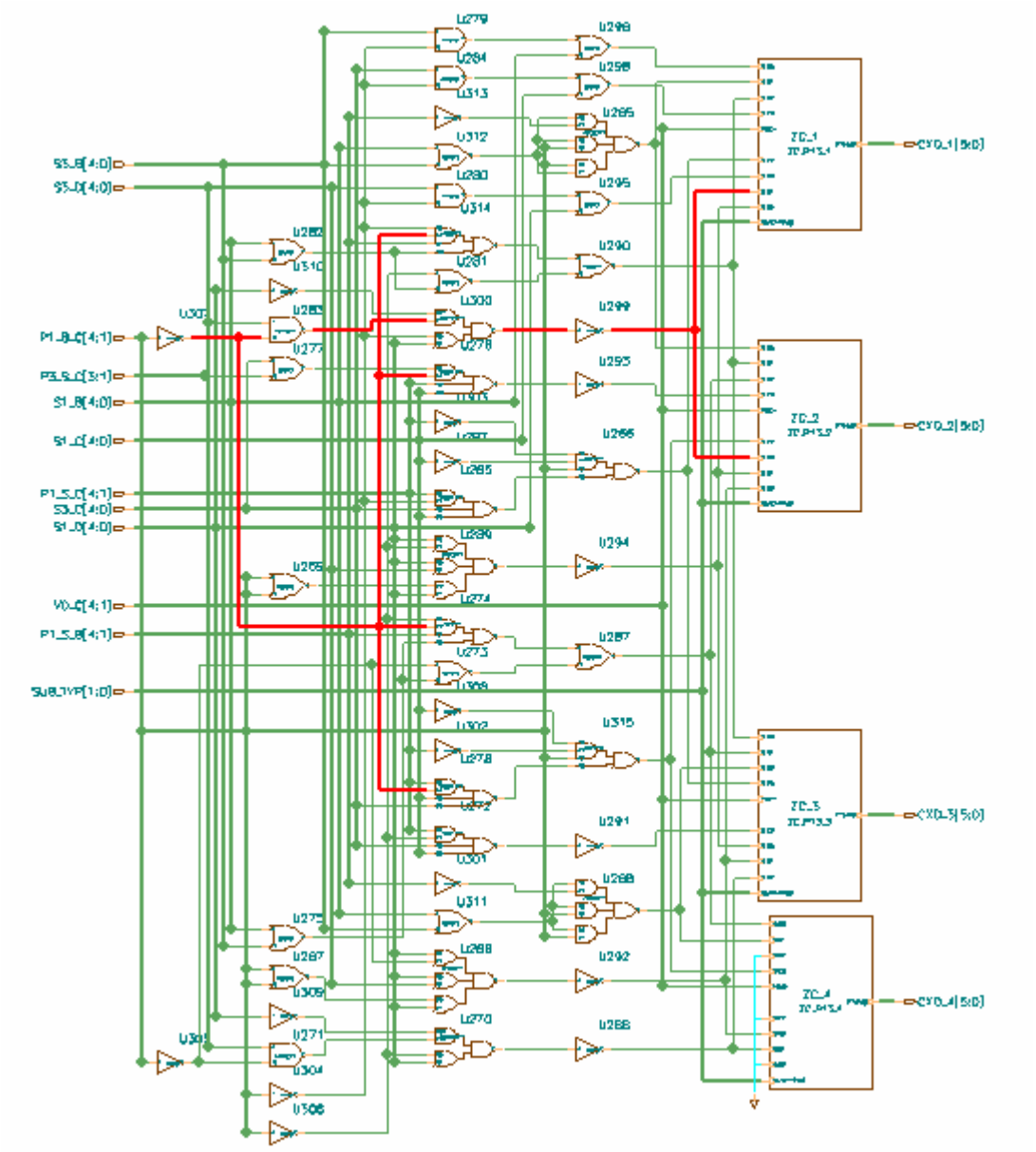


Figure 5-10 Circuit of zero coding for pass1 or pass3

As shown in Figure 5-11, the gate delay of zero coding is 8 gate counts.

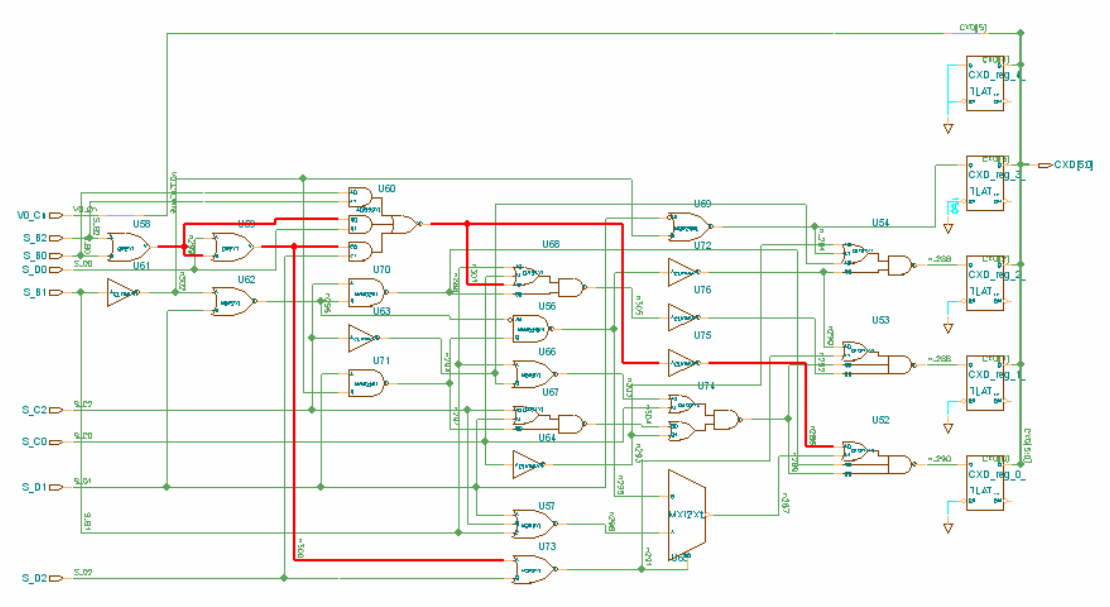


Figure 5-11 Internal circuit of zero coding

As shown in Figure 5-12, the gate delay of selecting pass1 or pass3 is 4 gate counts for zero coding.

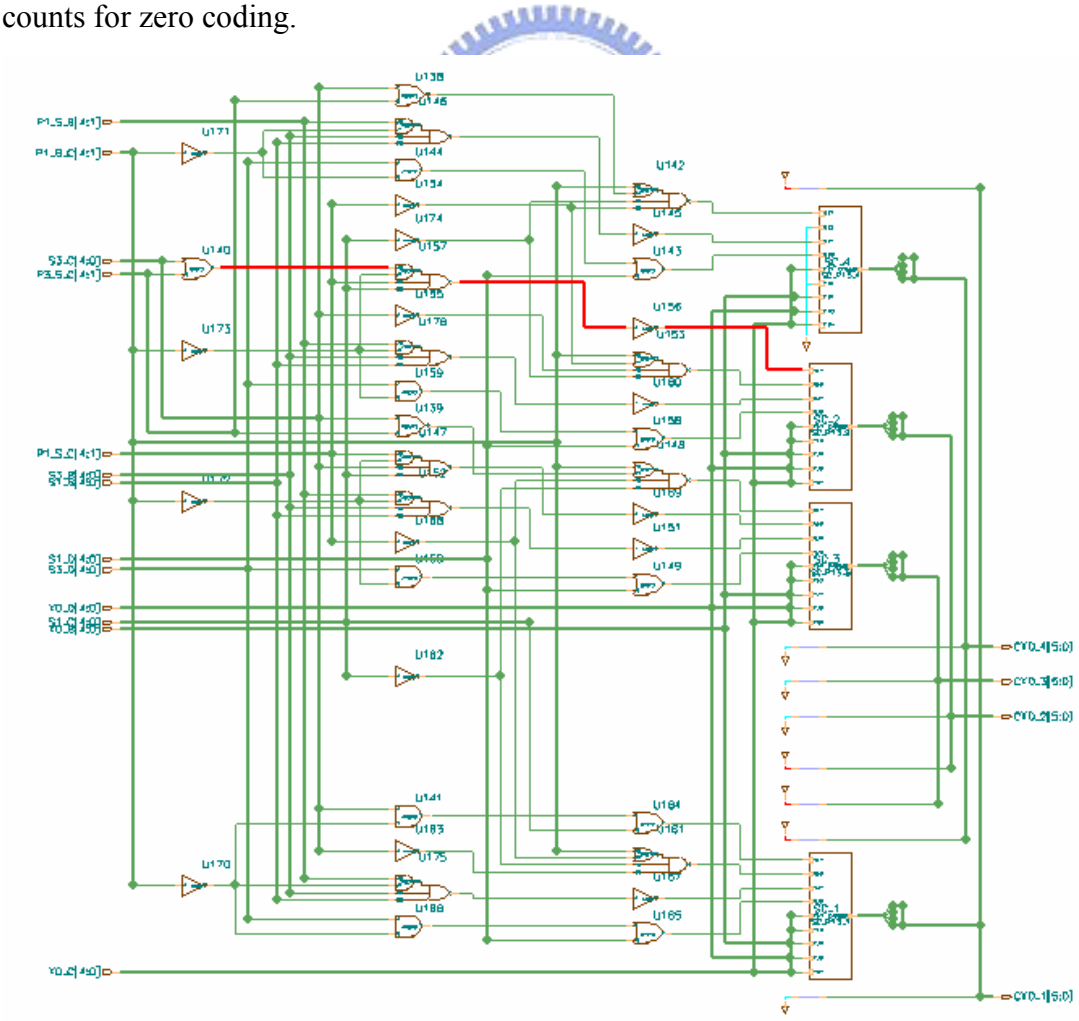


Figure 5-12 Circuit of sign coding for pass1 or pass3

As shown in Figure 5-13, the gate delay of sign coding is 9 gate counts.

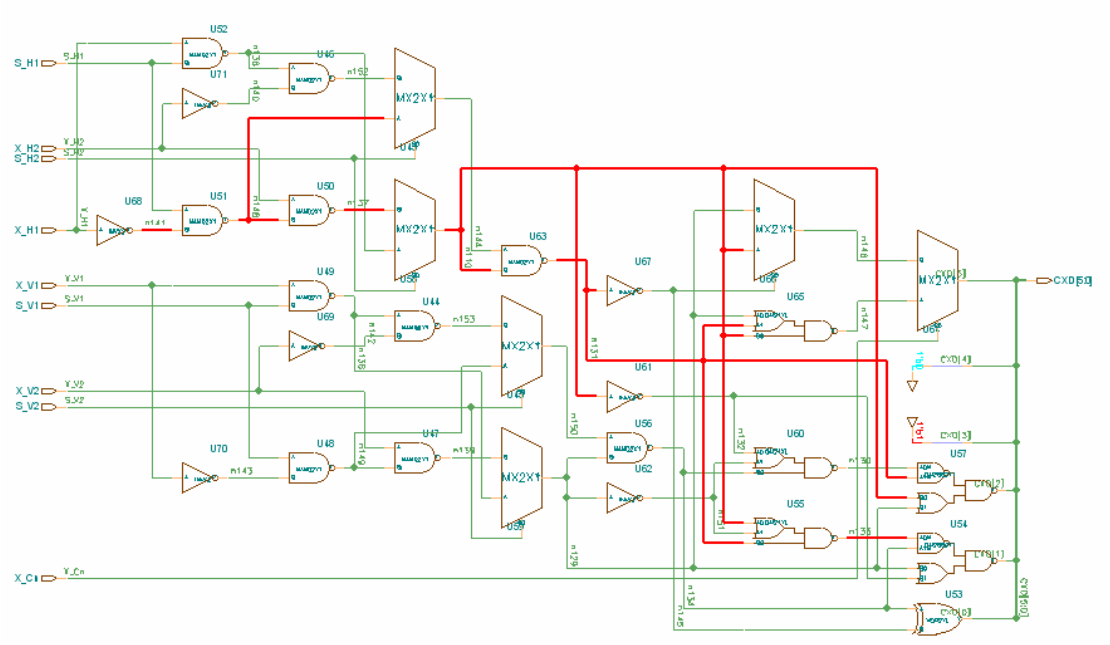


Figure 5-13 Internal circuit of sign coding

5.4 Design Flow and Verification

The overall cell-based design flow is shown in Figure 5-1.

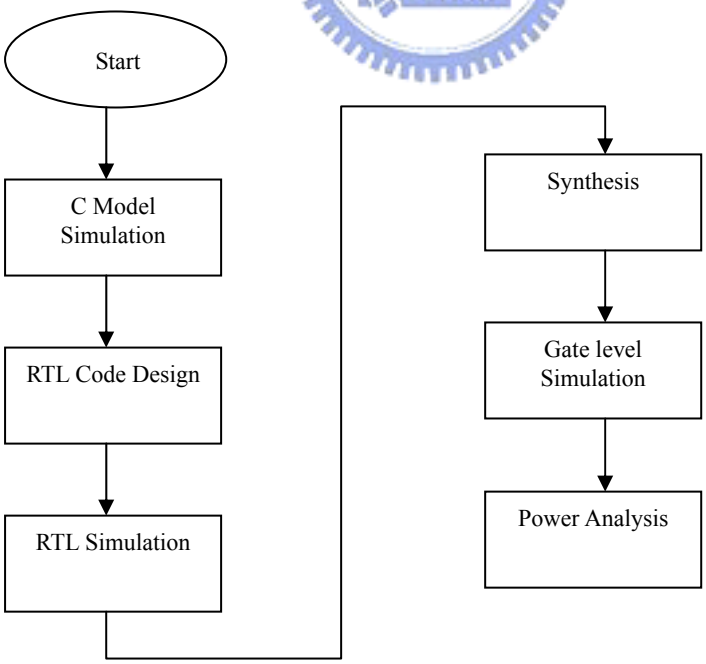


Figure 5-14 Flow chart of cell-based design

C Model Simulation

We use C language to build verification model for simulation.

RTL Code Design

We proceed to RTL(Register Transfer Level)design using verilog language.

RTL Simulation

The RTL codes are simulated through the nc-verilog simulator.

Synthesis

We adopt Synopsys Design Compiler for the logic synthesis and nc-verilog is used for the gate-level simulation.

Gate level Simulation

The gate-level of context formation encoder is simulated through nc-verilog simulator.

RTL Verification

In C model verification, we use Jasper Software [19] to verify our design. We collect the input data of CF module in Jasper as the input data in our design, and compare output CXD streams of Jasper and ours to verify correctness of our design. In RTL code verification, a C model of basic EBCOT Tier-1 context formation encoder described in chapter 2 is developed. Golden CXD pairs are generated from random samples according to this C model. These random samples are also used to test our design to see whether or not the outputs are the same as the golden CXDs. We use nc-verilog to verify our design, and get consistent results, which prove that our RTL design is correct in encoding.

Gate-level Net-list Verification

After gate-level net-list is generated by Synopsys Design Compiler, we also use nc-verilog to run simulations with the same random samples used in RTL simulation, and get the same results.

5.5 Experiment Results

Table 5-3 shows the performances of proposed design in detail.

Table 5-3. Performance of proposed design

Technology	TSMC 0.25 µm 1p5m process
Hardware Synthesizer	Synopsys® Design Compiler
Area	40037um^2
Operating Frequency (MHz) *1	333 MHz
Supply Voltage (V)	2.5v
Throughput (M Sample/Sec) *2	1332
Throughput (M Pixel/Sec) *3	190.28
Power Consumption	214mw
*1: The operating frequency is obtained from the pre-layout gate-level net-list simulation. *2: All four consecutive samples in a column are coded to CXD pairs in one clock cycle. *3: The DWT coefficient is 8-bit wide for each pixel.(2's complement format). One pixel = Seven samples	

Table 5-4 shows the typical context formation encoder performance in comparison with other designs. From Table 5-4, our design performs better than others.

Table 5-4. Performance comparison of Context Formation encoders

	Ours	NCTU[13]	NCTU[14]	[10]	NTU[7]	NCTU[11]
Technology (µm)	0.25	0.15	0.25	0.25	0.35	0.35
Mode	CF Encoder	CF Encoder	CF Codec	Tier-1 Encoder	Tier-1 Encoder	Tier-1 Encoder
Frequency (MHz)	333	600	100 (133)	185	50	142.8
Throughput (M/S)	190.28	342.8	12.32	x	x	12.32
CF gate counts	2316	2496	x	3055	x	x

NTCU[13] is implemented by CMOS 0.15μm technology. Typically, a design implemented by 0.15μm process is about 2.78x faster than by 0.25μm process.

If NTCU[13] is implemented by CMOS 0.25μm technology, the operation clock frequency is about 216 MHz. Typically, a design implemented by 0.25μm process is about 1.96x faster than by 0.35μm process. If NTCU[11] is implemented by CMOS 0.25μm technology, the operation clock frequency is about 279.8 MHz. The gate counts of ours are 2316. The gate counts of [13] are 2496. The gate counts of [10] are 3055.

CHAPTER 6 Conclusion

In this thesis, we focus on the research of Pass-Parallel Column-Based context formation encoder for JPEG2000. The EBCOT Tier-1 context formation encoder has high computational complexity, so a high performance and low power hardware architecture design of Context Formation encoder for JPEG2000 is proposed for it. The new hardware architecture is implemented by three speedup methods and pipeline technique. The architecture is proposed to improve the computation efficiency and reduce hardware area.

In our design, the overall system architecture of Pass-Parallel Column-Based context formation encoder is divided into four stages. The Dual Column Pass1 generation method is used in the second and third stages. All coding pass and significance change generation method and sample-parallel column-based coding method is used in third stage. The area of context window of Pass-Parallel Column-Based context formation encoder is reduced 25% by Dual Column Pass1 generation method in comparison with existing context window. The critical path of overall system architecture is reduced by Dual Column Pass1 generation, all coding pass and significance change generation method and sample-parallel column-based coding method. The critical path delay of overall system architecture is 19 gate counts.

Our design is described with Verilog HDL code and synthesized by Synopsys Design Compiler using TSMC CMOS 0.25μm process. The pre-layout synthesized area is 40037 μm^2. In our simulation, the operation clock frequency can reach 330 MHz. With this clock frequency, it needs 0.021 second to encode an image with 2304 x 1728 image size.

The EBCOT tier-1 context formation encoder is full with bit operation, so hardware implementation is more efficient both system throughput and power consumption. In the future, we can simultaneously execute samples of two columns in 5x5 shift register array by concept of Dual Column Pass1 generation method in order to improve the system throughput.

BIBLIOGRAPHY

[1] D. Taubman and M. W. Marcellin, JPEG2000 Image Compression Fundamentals, Standards and Practice, Kluwer Academic Publishers, 2002.

[2] ISO/IEC JTC1/SC29 WG 1 N1684, “JPEG2000 Part I Final Committee Draft Version 1.0,” March 2000.

[3] ISO/IEC JTC1/SC29 WG 1 N1894, “JPEG 2000 Verification Model 8.6,” 2000.

[4] K.Andra, C.Chakrabarti and T.Acharya, “A High Performance JPEG2000 Architecture,” Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on, vol.1,26-29 May 2002, Page(s): I-765 - I-768.

[5] D. Taubman, “High Performance scalable image compression with EBCOT,” IEEE Trans., Image Processing, vol. 9, issue 7, July 2000, Page(s):1158 –1170.

[6] Kuan-Fu Chen, Chung-Jr Lian, Hong-Hui Chen and Liang-Gee Chen, “Analysis and Architecture Design of EBCOT for JPEG 2000,” Circuits and Systems, 2001. ISCAS 2001. IEEE International Symposium on, Volume: 2, 2001, Page(s): 765 –768.

[7] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, and Liang-Gee Chen, “Analysis and Architecture Design of Block-Coding Engine for EBCOT in JPEG 2000,” Circuits and Systems for Video Technology, IEEE Transactions on, Volume: 13, Issue:3, March 2003, Page(s) : 219 –230.

[8] Jen-Shiun Chiang, Yu-Sen Lin, and Chang-Yo Hsieh, “Efficient Pass-Parallel Architecture for EBCOT in JPEG2000,” Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on, Volume: 1, 2002, Page(s): 773 -776.

[9] Hung-Chi Fang, Tu-Chih Wang, Chung-Jr Lian, Te-Hao Chang, Liang-Gee Chen, “HGH SPEED MEMORY EFFICIENT EBCOT Architecture for JPEG2000,” Circuits and Systems, 2003. ISCAS 2003. IEEE International Symposium on, Volume: 2 , 2003, Page(s): 736 -739.

[10] Jen-Shiun Chiang, Chun-Hau Chang, Yu-Sen Lin, Chang-You Hsieh, and Chih-Hsien Hsia, “High-Speed EBCOT with Dual Context-Modeling Coding Architecture for JPEG2000,” Circuits and Systems, 2004. ISCAS 2004. IEEE

International Symposium , Volume: 3, 2004, Page(s): 865 -868.

[11] Yun-Tai Hsiao, Hung-Der Lin, Kun-Bin Lee and Chein-Wei Jen, “High-Speed Memory-Saving Architecture for the Embedded Block Coding in JPEG2000,”Circuits and Systems, 2002. IEEE International Symposium on, Volume : 5, 2002, Page(s): 128 –136.

[12] Yijun Li, Ramy E. Aly, M. A. Bayoumi and S. A. Mashali, “Parallel High-Speed Architecture for EBCOT in JPEG2000,” Acoustics, Speech, and Signal Processing, 2003. Proceedings. 2003 IEEE International Conference on, Volume: 2, April 6 - 10, 2003, Page(s) 481 -484.

[13] Chi-Chin Chang, Sau-Gee Chen, “Efficient Design of JPEG2000 EBCOT TIER-I Context Formation Encoder,“ Master Thesis, NCTU, January 2006.

[14] Pei-Chun Chen, Bin-Fei Wu, “Design of the efficient Pass-Parallel Context Formation Codec for JPEG2000,“ Master Thesis, NCTU, July 2004.

[15] Amit Kumar Gupta, David Taubman, Saeid Nooshabadi, “High Speed VLSI Architecture for Bit Plane Encoder of JPEG2000,” Circuits and Systems, 2004. ISCAS 2004. IEEE International Symposium on, vol. 2, 2004, Page(s): 233 -236.

[16] Yan Xiaolang, Qin Xing, Yang Ye, Ge,Haitong, “A High Performance Architecture of EBCOT Encoder in JPEG2000 ,” Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium, Page(s) 492-495.

[17] Tsung-Han Tsai and Lian-Tsung Tsai, “JPEG2000 Encoder Architecture Design with Fast EBCOT Algorithm,” Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium, Page(s) 279-282.

[18] Yijun Li and Magdy Bayoumi, “THREE-Level Parallel High Speed Architecture for EBCOT in JPEG2000,” Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium, Page(s) V-5 – V-8.

[19] Jasper Software, http://www.ece.uvic.ca/~mdadams/jasper.

APPENDIX

A.1 The hardware description language of overall system top level of context formation proposed is expressed as follows:

```
modulePCF(
        //--------Output---------
    CXD_P1_P3_1_ZC,      // 1_ZC
    CXD_P1_P3_2_ZC,      // 2_ZC
    CXD_P1_P3_3_ZC,      // 3_ZC
    CXD_P1_P3_4_ZC,      // 4_ZC

    CXD_P1_P3_1_SC,      // 1_SC
    CXD_P1_P3_2_SC,      // 2_SC
    CXD_P1_P3_3_SC,      // 3_SC
    CXD_P1_P3_4_SC,      // 4_SC

    CXD_P2_1_MRC,        // 1_MRC
    CXD_P2_2_MRC,        // 2_MRC
    CXD_P2_3_MRC,        // 3_MRC
    CXD_P2_4_MRC,        // 4_MRC

    CXD_P3_RLC_0,// RLC_0
    CXD_P3_RLC_1,// RLC_1
    CXD_P3_RLC1_UC0,    // UC_0
    CXD_P3_RLC1_UC1,    // UC_1

    P1_B_C, // FOR PASS1
    P2_C,    // FOR PASS2
    P3_C,    // FOR PASS3

    S1_D_,
    S3_D_,

    ADDRA,
```

```
                ADDRB,
                WRITE,
                BP_INDEX,
                 // --------------Input-------
                X0_A_4_0,
                V0_A_4_1,
                S1_A_4_0,
                S3_A_4_0,

                START,
                SUB_TYP,
                BP_NUM,
                RST,
                CLK
               );
output  [5:0]    CXD_P1_P3_1_ZC;   // 1_ZC
output  [5:0]    CXD_P1_P3_2_ZC;   // 2_ZC
output  [5:0]    CXD_P1_P3_3_ZC;   // 3_ZC
output  [5:0]    CXD_P1_P3_4_ZC;   // 4_ZC
output  [5:0]    CXD_P1_P3_1_SC;   // 1_SC
output  [5:0]    CXD_P1_P3_2_SC;   // 2_SC
output  [5:0]    CXD_P1_P3_3_SC;   // 3_SC
output  [5:0]    CXD_P1_P3_4_SC;   // 4_SC
output  [5:0]    CXD_P2_1_MRC;     // 1_MRC
output  [5:0]    CXD_P2_2_MRC;     // 2_MRC
output  [5:0]    CXD_P2_3_MRC;     // 3_MRC
output  [5:0]    CXD_P2_4_MRC;     // 4_MRC
output  [5:0]    CXD_P3_RLC_0;     // RLC_0
output  [5:0]    CXD_P3_RLC_1;     // RLC_1
output  [5:0]    CXD_P3_RLC1_UC0;// UC_0
output  [5:0]    CXD_P3_RLC1_UC1;// UC_1
output  [4:1]    P1_B_C;
output  [4:1]    P2_C;
output  [4:1]    P3_C;
output           [4:1]    S1_D_;
output           [4:1]    S3_D_;
output  [9:0]     ADDRA;
output  [9:0]     ADDRB;
output           WRITE;
```

```
output  [3:0]   BP_INDEX;
input   [4:0]   X0_A_4_0;
input   [4:1]   V0_A_4_1;
input   [4:0]   S1_A_4_0;
input   [4:0]   S3_A_4_0;
input           START;
input   [1:0]   SUB_TYP;
input   [3:0]   BP_NUM;
input           RST;
input           CLK;
//Shift Register for Context Window use
reg     [4:0]   X0_A;
reg     [4:0]   X0_B;
reg     [4:0]   X0_C;
reg     [4:0]   X0_D;
reg     [4:1]   V0_A;
reg     [4:1]   V0_B;
reg     [4:1]   V0_C;
reg     [4:1]   V0_D;
reg     [4:0]   S1_A;
reg     [4:0]   S1_B;
reg     [4:0]   S1_C;
reg     [4:0]   S1_D;
reg     [4:0]   S3_A;
reg     [4:0]   S3_B;
reg     [4:0]   S3_C;
reg     [4:0]   S3_D;
//Output wire
wire    [5:0]   CXD_P1_P3_1_ZC;  // 1_ZC
wire    [5:0]   CXD_P1_P3_2_ZC;  // 2_ZC
wire    [5:0]   CXD_P1_P3_3_ZC;  // 3_ZC
wire    [5:0]   CXD_P1_P3_4_ZC;  // 4_ZC
wire    [5:0]   CXD_P1_P3_1_SC;  // 1_SC
wire    [5:0]   CXD_P1_P3_2_SC;  // 2_SC
wire    [5:0]   CXD_P1_P3_3_SC;  // 3_SC
wire    [5:0]   CXD_P1_P3_4_SC;  // 4_SC
wire    [5:0]   CXD_P2_1_MRC;    // 1_MRC
wire    [5:0]   CXD_P2_2_MRC;    // 2_MRC
wire    [5:0]   CXD_P2_3_MRC;    // 3_MRC
```

```
wire    [5:0]   CXD_P2_4_MRC;     // 4_MRC
wire    [5:0]   CXD_P3_RLC_0;     // RLC_0
wire    [5:0]   CXD_P3_RLC_1;     // RLC_1
wire    [5:0]   CXD_P3_RLC1_UC0;// UC_0
wire    [5:0]   CXD_P3_RLC1_UC1;// UC_1
wire    [4:1]   P1_B_C;
wire    [4:1]   P2_C;
wire    [4:1]   P3_C;
wire      [4:1]       S1_D_;
wire      [4:1]       S3_D_;
wire    [9:0]   ADDRA;
wire    [9:0]   ADDRB;
wire            WRITE;
wire    [3:0]   BP_INDEX;
//Internal wire
wire    [4:0]   N_X0_A;
wire    [4:1]   N_V0_A;
wire    [4:0]   N_S1_A;
wire    [4:0]   N_S3_A;
wire [4:1] P1_S_B;
wire [4:1] P1_B;
wire [4:1] P2_1_C;
wire [4:1] P1_S_C;
wire [4:1] P1_B_C;
wire [4:1] P3_S_C;
wire                P3_RLC0;
wire                P3_RLC1;
wire                P3_RLC1_UC1;
wire                P3_RLC1_UC0;
//--------------------PCF CONTROL ---
PCF_CONTROL       PCF_CONTROL_0(
                        .ADDRA(ADDRA),
                        .ADDRB(ADDRB),
                        .WRITE(WRITE),
                        .BP_INDEX(BP_INDEX),

                        .START(START),
                        .BP_NUM(BP_NUM),
                        .RST(RST),
```

```
                                .CLK(CLK)
                               );
//--------------------------------
assign  N_X0_A        = {X0_A_4_0[4:0]};
assign  N_V0_A        = {V0_A_4_1[4:1]};
assign  N_S1_A        = {S1_A_4_0[4:0]};
assign  N_S3_A        = {S3_A_4_0[4:0]};
//-Store Pass1 or Pass3 significance change

assign  S1_D_ = S1_D[4:1];
assign  S3_D_ = S3_D[4:1];
//--Shift Register Operation -----
always @(posedge CLK) begin
    if (!RST | START) begin
        X0_A  <= #`DELAY 5'b0;
        V0_B  <= #`DELAY 5'b0;
        S3_C  <= #`DELAY 5'b0;
        S3_D  <= #`DELAY 5'b0;
    end
    else begin
        X0_A  <= #`DELAY N_X0_A;
        V0_A  <= #`DELAY N_V0_A;
        S1_A  <= #`DELAY N_S1_A;
        S3_A  <= #`DELAY N_S3_A;
    end
end
always @(posedge CLK) begin
    if (!RST | START) begin
        X0_B  <= #`DELAY 5'b0;
        X0_C  <= #`DELAY 5'b0;
        X0_D  <= #`DELAY 5'b0;
        V0_B  <= #`DELAY 5'b0;
        V0_C  <= #`DELAY 5'b0;
        V0_D  <= #`DELAY 5'b0;
        S1_B  <= #`DELAY 5'b0;
        S1_C  <= #`DELAY 5'b0;
        S1_D  <= #`DELAY 5'b0;
        S3_B  <= #`DELAY 5'b0;
        S3_C  <= #`DELAY 5'b0;
```

```
        S3_D   <= #`DELAY 5'b0;
    end
    else begin
        X0_D  <= #`DELAY X0_C;
        X0_C  <= #`DELAY X0_B;
        X0_B  <= #`DELAY X0_A;

        V0_D  <= #`DELAY V0_C;
        V0_C  <= #`DELAY V0_B;
        V0_B  <= #`DELAY V0_A;

        S1_D      <= #`DELAY S1_C;
        S1_C      <= #`DELAY S1_B;
        S1_B      <= #`DELAY S1_A;

        S3_D      <= #`DELAY S3_C;
        S3_C      <= #`DELAY S3_B;
        S3_B      <= #`DELAY S3_A;

    end
end
//--- Dual Column Pass1 generation--
P1_PRE_CHECK      P1_PRE_CHECK(
                                    P1_B,
                                        P1_S_B,

                                        S1_A,
                                        S1_B,
                                        S1_C,
                                        S3_A,
                                        S3_B,
                                        V0_B,
                                        P1_S_C
                                    );

//--All Coding Pass and Significance Change Generation--------
PCPG      PCPG_0(
                        .P1_S_C(P1_S_C),
                        .P2_C(P2_C),
```

```
                    .P2_1_C(P2_1_C),
                    .P1_B_C(P1_B_C),
                    .P3_C(P3_C),
                    .P3_S_C(P3_S_C),
                    .P3_RLC0(P3_RLC0),
                    .P3_RLC1(P3_RLC1),
                    .P3_RLC1_UC1(P3_RLC1_UC1),
                    .P3_RLC1_UC0(P3_RLC1_UC0),

                    .S3_C(S3_C),
                    .S1_C(S1_C),
                    .V0_C(V0_C),
                    .P1_B(P1_B),
                    .START(START),
                    .RST(RST),
                    .CLK(CLK)
                    );

//---Sample-Parallel Column-Based Coding ----
// FOR PASS2 Magnitude Refinement Coding USE
MRC_COLUMN      MRC_COLUMN0(
                        .CXD_1(CXD_P2_1_MRC     ),
                        .CXD_2(CXD_P2_2_MRC     ),
                        .CXD_3(CXD_P2_3_MRC     ),
                        .CXD_4(CXD_P2_4_MRC     ),

                        .V0_C(V0_C),
                        .S1_B(S1_B),
                        .S1_C(S1_C),
                        .S1_D(S1_D),
                        .S3_B(S3_B),
                        .S3_C(S3_C),
                        .S3_D(S3_D),
                        .P1_S_B(P1_S_B),
                        .P1_S_C(P1_S_C),
                        .P2_1_C(P2_1_C)
                       );
// FOR PASS1 OR PASS3 ZERO CODING USE
ZC_COLUMN_P1_P3   ZC_COLUMN_P1_P3_0(
```

```
                        .CXD_1(CXD_P1_P3_1_ZC ),
                        .CXD_2(CXD_P1_P3_2_ZC ),
                        .CXD_3(CXD_P1_P3_3_ZC ),
                        .CXD_4(CXD_P1_P3_4_ZC ),

                        .V0_C(V0_C),
                        .S1_B(S1_B),
                        .S1_C(S1_C),
                        .S1_D(S1_D),
                        .S3_B(S3_B),
                        .S3_C(S3_C),
                        .S3_D(S3_D),
                        .P1_S_B(P1_S_B),
                        .P1_S_C(P1_S_C),
                        .P1_B_C(P1_B_C),
                        .P3_S_C(P3_S_C[3:1]),
                        .SUB_TYP(SUB_TYP )
                        );
// FOR PASS1 OR PASS3 SIGN CODING USE
SC_COLUMN_P1_P3    SC_COLUMN_P1_P3_0(
                        .CXD_1(CXD_P1_P3_1_SC),
                        .CXD_2(CXD_P1_P3_2_SC),
                        .CXD_3(CXD_P1_P3_3_SC),
                        .CXD_4(CXD_P1_P3_4_SC),

                        .X0_B(X0_B),
                        .X0_C(X0_C),
                        .X0_D(X0_D),
                        .S1_B(S1_B),
                        .S1_C(S1_C),
                        .S1_D(S1_D),
                        .S3_C(S3_C),
                        .S3_D(S3_D),
                        .P1_S_B(P1_S_B),
                        .P1_S_C(P1_S_C),
                        .P1_B_C(P1_B_C),
                        .P3_S_C(P3_S_C)
                       );
// FOR PASS3 Run-Length Code 0 USE
```

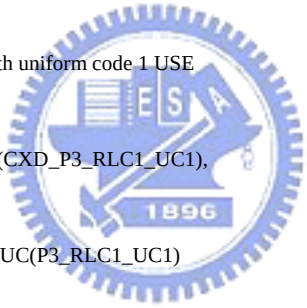
```
RLC_COLUMN_0    RLC_COLUMN_0_N(
                        .CXD_RLC(CXD_P3_RLC_0),

                        .CXD_P3_RLC(!P3_RLC0)
                       );
// FOR PASS3 Run-Length Code 1 USE
RLC_COLUMN_1    RLC_COLUMN_1_N(
                        .CXD_RLC(CXD_P3_RLC_1),

                        .CXD_P3_RLC(P3_RLC1)
                       );
// FOR PASS3 Run-Length Code 1 with uniform code 0 USE
RLC1_UC0    RLC1_UC0_N(
                        .CXD_UC(CXD_P3_RLC1_UC0),

                        .CXD_P3_UC(P3_RLC1_UC0)
                       );
// FOR PASS3 Run-Length Code 1 with uniform code 1 USE
RLC1_UC1    RLC1_UC1_N(
                        .CXD_UC(CXD_P3_RLC1_UC1),

                        .CXD_P3_UC(P3_RLC1_UC1)
                       );
//-------------------------------------------------
endmodule
```



A.2 The hardware description language of dual column pass1 generation is expressed as follows:

```
moduleP1_PRE_CHECK(
    P1_B,
    P1_S_B,
    S1_A,
    S1_B,
    S1_C,
    S3_A,
    S3_B,
    V0_B,
    P1_S_C
```

```
            );
output  [4:1]    P1_B;
output  [4:1]    P1_S_B;
input   [4:0]    S1_A;
input   [4:0]    S1_B;
input   [4:0]    S1_C;
input   [4:0]    S3_A;
input   [4:0]    S3_B;
input   [4:1]    V0_B;
input   [4:1]    P1_S_C;
reg     [4:1]    F1_P1_B;
wire    [4:1]    P1_B;
wire      [4:1]     F2_P1_B;
always @(S1_A or S1_B or S1_C or S3_B or S3_A ) begin
    if ((!(S1_B[1]|S3_B[1])) & ((S1_C[0]|S1_C[1]|S1_C[2])|(S1_B[0]|S1_A[0])
            |(S1_B[2]|S3_B[2])|(S1_A[1]|S3_A[1])|(S1_A[2]|S3_A[2])))
        F1_P1_B[1]   = 1'b1;
    else
        F1_P1_B[1]   = 1'b0;
    if ((!(S1_B[2]|S3_B[2])) & ((S1_C[1]|S1_C[2])|(S1_C[3]|S1_B[1])|(S1_B[3]|S3_B[3])
            |(S1_A[1]|S3_A[1])|(S1_A[2]|S3_A[2])|(S1_A[3]|S3_A[3])))
         F1_P1_B[2]   = 1'b1;
    else
        F1_P1_B[2]   = 1'b0;
 if ((!(S1_B[3]|S3_B[3])) & ((S1_C[2]|S1_C[3])|(S1_C[4]|S1_B[2])|(S1_B[4]|S3_B[4])
            |(S1_A[2]|S3_A[2])|(S1_A[3]|S3_A[3])|(S1_A[4]|S3_A[4])))
        F1_P1_B[3]   = 1'b1;
    else
        F1_P1_B[3]   = 1'b0;
    if ((!(S1_B[4]|S3_B[4])) & ((S1_C[3]|S1_C[4]|S1_B[3])
            |(S1_A[3]|S3_A[3])|(S1_A[4]|S3_A[4])))
        F1_P1_B[4]   = 1'b1;
    else
        F1_P1_B[4]   = 1'b0;
end
P1_V_CHECK          P1_V_CHECK0(
                                .P1_B(P1_B),
                                .P1_S_B(P1_S_B),
```

```
                                .F2_P1_B(F2_P1_B),
                                .V0_B(V0_B)
                                 );
P1_H_CHECK          P1_H_CHECK0(
                                .F2_P1_B(F2_P1_B),

                                .F1_P1_B(F1_P1_B),
                                .P1_S_C(P1_S_C)
                                );
endmodule
moduleP1_H_CHECK(
            F2_P1_B,

            F1_P1_B,
            P1_S_C
            );
output  [4:1]    F2_P1_B;
input   [4:1]    F1_P1_B;
input   [4:1]    P1_S_C;
wire    [4:1]    F2_P1_B;
assign  F2_P1_B[1]= F1_P1_B[1] | P1_S_C[1] | P1_S_C[2];
assign  F2_P1_B[2]= F1_P1_B[2] | P1_S_C[1] | P1_S_C[2] | P1_S_C[3];
assign  F2_P1_B[3]= F1_P1_B[3] | P1_S_C[2] | P1_S_C[3] | P1_S_C[4];
assign  F2_P1_B[4]= F1_P1_B[4] | (P1_S_C[3] | P1_S_C[4]);
endmodule
moduleP1_V_CHECK(
            P1_B,
            P1_S_B,

            F2_P1_B,
            V0_B
            );
output  [4:1]    P1_B;
output  [4:1]    P1_S_B;
input   [4:1]    F2_P1_B;
input   [4:1]    V0_B;
wire      [4:1]     P1_S_B;
wire    [4:1]    P1_B;
assign  P1_B[1]= F2_P1_B[1];
```

```
assign  P1_S_B[1]= P1_B[1] & V0_B[1];
assign    P1_B[2] = P1_S_B[1] | F2_P1_B[2];
assign  P1_S_B[2]= P1_B[2] & V0_B[2];
assign    P1_B[3] = P1_S_B[2] | F2_P1_B[3];
assign  P1_S_B[3]= P1_B[3] & V0_B[3];
assign    P1_B[4] = P1_S_B[3] | F2_P1_B[4];
assign  P1_S_B[4]= P1_B[4] & V0_B[4];
endmodule
```

A.3 The hardware description language of all coding pass generation and significance change is expressed as follows:

```
module  PCPG(
            P1_S_C,
            P2_C,
            P2_1_C,
            P1_B_C,
            P3_C,
            P3_S_C,
            P3_RLC0,
            P3_RLC1,
            P3_RLC1_UC1,
            P3_RLC1_UC0,
            S3_C,
            S1_C,
            V0_C,
            P1_B,
            START,
            RST,
                    CLK
            );
output   [4:1]    P1_S_C;
output   [4:1]   P2_C;
output   [4:1]    P2_1_C;
output   [4:1]   P1_B_C;
output   [4:1]    P3_C;
output   [4:1]    P3_S_C;
```

```
output              P3_RLC0;
output              P3_RLC1;
output            P3_RLC1_UC1;
output            P3_RLC1_UC0;
input    [4:0]    S3_C;
input    [4:0]    S1_C;
input    [4:1]    V0_C;
input    [4:1]     P1_B;
input    START;
input    RST;
input    CLK;
wire     [4:1]    P1_S_C;
wire      [4:1]     P2_C;
wire      [4:1]     P2_1_C;
wire      [4:1]     P3_C;
wire      [4:1]     P3_S_C;
wire                P3_RLC0;
wire                P3_RLC1;
wire                P3_MAG;
wire              P3_RLC1_UC1;
wire              P3_RLC1_UC0;
wire               P3_RLC0_T1;
wire        P3_RLC1_UC1_T1;
wire        P3_RLC1_UC0_T1;
wire        P3_RLC1_UC0_T2;
reg      [4:1]    P1_B_C;
assign     P2_C[1]    = S1_C[1] | S3_C[1];
assign     P2_C[2]    = S1_C[2] | S3_C[2];
assign     P2_C[3]    = S1_C[3] | S3_C[3];
assign     P2_C[4]    = S1_C[4] | S3_C[4];
assign     P2_1_C[1]    = S1_C[1] ^ S3_C[1];
assign     P2_1_C[2]    = S1_C[2] ^ S3_C[2];
assign     P2_1_C[3]    = S1_C[3] ^ S3_C[3];
assign     P2_1_C[4]    = S1_C[4] ^ S3_C[4];
always   @(posedge CLK)
begin
    if (!RST | START)
    begin
        P1_B_C <= #`DELAY 5'b0;
```

```
            end
        else begin
            P1_B_C <= #`DELAY P1_B;
              end
    end
    assign    P1_S_C[1] = P1_B_C[1] & V0_C[1];
    assign    P1_S_C[2] = P1_B_C[2] & V0_C[2];
    assign    P1_S_C[3] = P1_B_C[3] & V0_C[3];
    assign    P1_S_C[4] = P1_B_C[4] & V0_C[4];
    assign    P3_C[1] = !(P2_C[1] | P1_B_C[1]);
    assign    P3_C[2] = !(P2_C[2] | P1_B_C[2]);
    assign    P3_C[3] = !(P2_C[3] | P1_B_C[3]);
    assign    P3_C[4] = !(P2_C[4] | P1_B_C[4]);
    assign    P3_S_C[1] = P3_C[1] & V0_C[1];
    assign    P3_S_C[2] = P3_C[2] & V0_C[2];
    assign    P3_S_C[3] = P3_C[3] & V0_C[3];
    assign    P3_S_C[4] = P3_C[4] & V0_C[4];
    assign    P3_RLC0_T1 = P3_C[1] & P3_C[2]& P3_C[3] & P3_C[4];
    assign    P3_MAG       = V0_C[1] | V0_C[2]| V0_C[3] | V0_C[4];
    assign    P3_RLC0       = P3_RLC0_T1 & (!P3_MAG);
    assign    P3_RLC1       = P3_RLC0_T1 & P3_MAG;
    assign    P3_RLC1_UC1_T1 = !(V0_C[4] | V0_C[3]);
    assign        P3_RLC1_UC1 = !(P3_RLC1_UC1_T1 | V0_C[2] | V0_C[1]);
    assign    P3_RLC1_UC0_T1 = !(!V0_C[4] | V0_C[3]);
    assign    P3_RLC1_UC0_T2 = !(P3_RLC1_UC0_T1 | V0_C[2]);
    assign    P3_RLC1_UC0 = !(P3_RLC1_UC0_T2 | V0_C[1]);
    endmodule
```

A.4 The hardware description language of the magnitude refinement coding is expressed as follows:

```
moduleMRC_COLUMN(
                CXD_1,
                CXD_2,
                CXD_3,
                CXD_4,

                V0_C,
                S1_B,
```

```
                    S1_C,
                    S1_D,
                      S3_B,
                    S3_C,
                    S3_D,
                    P1_S_B,
                    P1_S_C,
                    P2_1_C
                    );
output [5:0]   CXD_1;
output [5:0]   CXD_2;
output [5:0]   CXD_3;
output [5:0]   CXD_4;
input  [4:1]   V0_C;
input  [4:0]   S1_B;
input  [4:0]   S1_C;
input  [4:0]   S1_D;
input  [4:0]   S3_B;
input  [4:0]   S3_C;
input  [4:0]   S3_D;
input  [4:1]   P1_S_B;
input  [4:1]   P1_S_C;
input  [4:1]   P2_1_C;
wire     [4:1]    B0_;
wire     [4:1]    B1_;
wire     [4:1]    B2_;
wire     [4:1]    C0_;
wire     [4:1]    C2_;
wire     [4:1]    D0_;
wire     [4:1]    D1_;
wire     [4:1]    D2_;
//------------FOR THE FIRST SAMPLE IN REGISTER C--------
assign   B0_[1] = S1_B[0];
assign   B1_[1] = S1_B[1]|S3_B[1]|P1_S_B[1];
assign   B2_[1] = S1_B[2]|S3_B[2]|P1_S_B[2];
assign   C0_[1] = S1_C[0];
assign   C2_[1] = S1_C[2]|S3_C[2]|P1_S_C[2];
assign   D0_[1] = S1_D[0];
assign   D1_[1] = S1_D[1];//S1_D[N]=S1_C[N]|P1_S_C[N]
```

```
assign    D2_[1] = S1_D[2];//
//-------------FOR THE SECOND SAMPLE IN REGISTER C----------
assign    B0_[2] = S1_B[1]|S3_B[1]|P1_S_B[1];
assign    B1_[2] = S1_B[2]|S3_B[2]|P1_S_B[2];
assign    B2_[2] = S1_B[3]|S3_B[3]|P1_S_B[3];
assign    C0_[2] = S1_C[1]|P1_S_C[1];
assign    C2_[2] = S1_C[3]|S3_C[3]|P1_S_C[3];
assign    D0_[2] = S1_D[1];
assign    D1_[2] = S1_D[2];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign    D2_[2] = S1_D[3];//
//-------------FOR THE THIRD SAMPLE IN REGISTER C----------
assign    B0_[3] = S1_B[2]|S3_B[2]|P1_S_B[2];
assign    B1_[3] = S1_B[3]|S3_B[3]|P1_S_B[3];
assign    B2_[3] = S1_B[4]|S3_B[4]|P1_S_B[4];
assign    C0_[3] = S1_C[2]|P1_S_C[2];
assign    C2_[3] = S1_C[4]|S3_C[4]|P1_S_C[4];
assign    D0_[3] = S1_D[2];
assign    D1_[3] = S1_D[3];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign    D2_[3] = S1_D[4];//
//-------------FOR THE FOURTH SAMPLE IN REGISTER C---------
assign    B0_[4] = S1_B[3]|S3_B[3]|P1_S_B[3];
assign    B1_[4] = S1_B[4]|S3_B[4]|P1_S_B[4];
assign    B2_[4] = 0;
assign    C0_[4] = S1_C[3]|P1_S_C[3];
assign    C2_[4] = 0;
assign    D0_[4] = S1_D[3];
assign    D1_[4] = S1_D[4];//S1_D[N]=S1_C[N]|P1_S_C[N]
assign    D2_[4] = 0;//

//-----------------------------------------------------------------

MRC1 MRC_1(
                .CXD   (CXD_1),
                .B0      (B0_[1]),
                .B1      (B1_[1]),
                .B2      (B2_[1]),
                .C0      (C0_[1]),
                .C2      (C2_[1]),
                .D0      (D0_[1]),
```

```
        .D1     (D1_[1]),
        .D2     (D2_[1]),
        .V0_Cn          (V0_C[1]),
        .P2_1_Cn    (P2_1_C[1])
    );

MRC2 MRC_2(
        .CXD    (CXD_2),
        .B0     (B0_[2]),
        .B1     (B1_[2]),
        .B2     (B2_[2]),
        .C0     (C0_[2]),
        .C2     (C2_[2]),
        .D0     (D0_[2]),
        .D1     (D1_[2]),
        .D2     (D2_[2]),
        .V0_Cn          (V0_C[2]),
        .P2_1_Cn    (P2_1_C[2])
    );

MRC3 MRC_3(
        .CXD    (CXD_3),
        .B0     (B0_[3]),
        .B1     (B1_[3]),
        .B2     (B2_[3]),
        .C0     (C0_[3]),
        .C2     (C2_[3]),
        .D0     (D0_[3]),
        .D1     (D1_[3]),
        .D2     (D2_[3]),
        .V0_Cn          (V0_C[3]),
        .P2_1_Cn    (P2_1_C[3])
    );

MRC4 MRC_4(
        .CXD    (CXD_4),
        .B0     (B0_[4]),
        .B1     (B1_[4]),
        .B2     (B2_[4]),
```

```
        .C0     (C0_[4]),
        .C2     (C2_[4]),
        .D0     (D0_[4]),
        .D1     (D1_[4]),
        .D2     (D2_[4]),
        .V0_Cn          (V0_C[4]),
        .P2_1_Cn    (P2_1_C[4])
      );

endmodule
```

A.5 The hardware description language of the zero coding for pass1 coding operation or pass3 coding operation is expressed as follows:

```
module  ZC_COLUMN_P1_P3(
          CXD_1,
          CXD_2,
          CXD_3,
          CXD_4,

          V0_C,
          S1_B,
          S1_C,
          S1_D,
          S3_B,
          S3_C,
          S3_D,
          P1_S_B,
          P1_S_C,
          P1_B_C,
          P3_S_C,
          SUB_TYP
          );
output   [5:0]   CXD_1;
output   [5:0]   CXD_2;
output   [5:0]   CXD_3;
output   [5:0]   CXD_4;
```

```
input     [4:1]    V0_C;
input     [4:0]    S1_B;
input     [4:0]    S1_C;
input     [4:0]    S1_D;
input     [4:0]     S3_B;
input     [4:0]     S3_C;
input     [4:0]    S3_D;
input     [4:1]    P1_S_B;
input     [4:1]    P1_S_C;
input     [4:1]    P1_B_C;
input     [3:1]     P3_S_C;
input     [1:0]    SUB_TYP;
wire      [5:0]        S_T_B;
wire      [5:0]        S_T_C;
wire      [5:0]        S_T_D;
wire      [4:1]        V_T_C;
//------------ FOR THE FIRST SAMPLE IN REGISTER C --------
assign    S_T_B[0] = (P1_B_C[1])? S1_B[0] : S1_B[0]|S3_B[0];
assign    S_T_B[1] = (P1_B_C[1])? S1_B[1]|S3_B[1] : S1_B[1]|P1_S_B[1]|S3_B[1];
assign    S_T_B[2] = (P1_B_C[1])? S1_B[2]|S3_B[2] : S1_B[2]|P1_S_B[2]|S3_B[2];
assign    S_T_C[0] = (P1_B_C[1])? S1_C[0]: S1_C[0]|S3_C[0];
assign    V_T_C[1] = V0_C[1];
assign    S_T_C[2] = (P1_B_C[1])? S1_C[2]|S3_C[2] : S1_C[2]|P1_S_C[2]|S3_C[2];
assign    S_T_D[0] = (P1_B_C[1])? S1_D[0]              : S1_D[0]|S3_D[0];
assign    S_T_D[1] = (P1_B_C[1])? S1_D[1]              : S1_D[1]|S3_D[1];
assign    S_T_D[2] = (P1_B_C[1])? S1_D[2]              : S1_D[2]|S3_D[2];
// S1_D[0]=S1_C[0];S3_D[0]=S3_C[0]
// S1_D[n]=S1_C[n]|P1_S_C[n]
// S3_D[n]=S3_C[n]|P3_S_C[n]
//-------------FOR THE SECOND SAMPLE IN REGISTER C----------
assign    S_T_B[1] = (P1_B_C[2])? S1_B[1]|S3_B[1] : S1_B[1]|S3_B[1]|P1_S_B[1];
assign    S_T_B[2] = (P1_B_C[2])? S1_B[2]|S3_B[2] : S1_B[2]|S3_B[2]|P1_S_B[2];
assign    S_T_B[3] = (P1_B_C[2])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign    S_T_C[1] = (P1_B_C[2])? S1_C[1]|P1_S_C[1] : S1_C[1]|S3_C[1]|P1_S_C[1]|P3_S_C[1];
assign    V_T_C[2] = V0_C[2];
assign    S_T_C[3] = (P1_B_C[2])? S1_C[3]|S3_C[3] : S1_C[3]|S3_C[3]|P1_S_C[3];
assign    S_T_D[1] = (P1_B_C[2])? S1_D[1] : S1_D[1]|S3_D[1];
assign    S_T_D[2] = (P1_B_C[2])? S1_D[2] : S1_D[2]|S3_D[2];
assign    S_T_D[3] = (P1_B_C[2])? S1_D[3] : S1_D[3]|S3_D[3];
```

```
// S3_D[1]=S3_C[1]|P3_S_C[1]
// S3_D[2]=S3_C[2]|P3_S_C[2]
// S1_D[3]=S1_C[3]|P1_S_C[3]
//-------------FOR THE THIRD SAMPLE IN REGISTER C----------
assign    S_T_B[2] = (P1_B_C[3])? S1_B[2]|S3_B[2] : S1_B[2]|S3_B[2]|P1_S_B[2];
assign    S_T_B[3] = (P1_B_C[3])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign    S_T_B[4] = (P1_B_C[3])? S1_B[4]|S3_B[4] : S1_B[4]|S3_B[4]|P1_S_B[4];
assign    S_T_C[2] = (P1_B_C[3])? S1_C[2]|P1_S_C[2] : S1_C[2]|S3_C[2]|P1_S_C[2]|P3_S_C[2];
assign    V_T_C[3] = V0_C[3];
assign    S_T_C[4] = (P1_B_C[3])? S1_C[4]|S3_C[4] : S1_C[4]|S3_C[4]|P1_S_C[4];
assign    S_T_D[2] = (P1_B_C[3])? S1_D[2] : S1_D[2]|S3_D[2];
assign    S_T_D[3] = (P1_B_C[3])? S1_D[3] : S1_D[3]|S3_D[3];
assign    S_T_D[4] = (P1_B_C[3])? S1_D[4] : S1_D[4]|S3_D[4];
//-------------FOR THE FOURTH SAMPLE IN REGISTER C----------
assign    S_T_B[3] = (P1_B_C[4])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign    S_T_B[4] = (P1_B_C[4])? S1_B[4]|S3_B[4] : S1_B[4]|S3_B[4]|P1_S_B[4];
assign    S_T_B[5] = (P1_B_C[4])?    1'b0     :      1'b0              ;
assign    S_T_C[3] = (P1_B_C[4])? S1_C[3]|P1_S_C[3] : S1_C[3]|S3_C[3]|P1_S_C[3]|P3_S_C[3];
assign    V_T_C[4] = V0_C[4];
assign    S_T_C[5] = (P1_B_C[4])?     1'b0    :          1'b0          ;
assign    S_T_D[3] = (P1_B_C[4])? S1_D[3] : S1_D[3]|S3_D[3];
assign    S_T_D[4] = (P1_B_C[4])? S1_D[4] : S1_D[4]|S3_D[4];
assign    S_T_D[5] = (P1_B_C[4])?     1'b0    :     1'b0 ;
//-------------------------------------------------------------
ZC_P13_1              ZC_1(
                    .CXD            (CXD_1        ),
                    .S_B0           (S_T_B[0]),
                    .S_B1           (S_T_B[1]),
                            .S_B2           (S_T_B[2]),
                            .S_C0           (S_T_C[0]),
                    .V0_Cn          (V_T_C[1]),
                            .S_C2           (S_T_C[2]),
                           .S_D0            (S_T_D[0]),
                    .S_D1           (S_T_D[1]),
                    .S_D2           (S_T_D[2]),
                    .SUB_TYP    (SUB_TYP)
                   );
  ZC_P13_2    ZC_2(
                    .CXD (CXD_2         ),
```

```
                .S_B0           (S_T_B[1]),
                .S_B1           (S_T_B[2]),
                        .S_B2           (S_T_B[3]),
                         .S_C0          (S_T_C[1]),
                .V0_Cn          (V_T_C[2]),
                        .S_C2           (S_T_C[3]),
                        .S_D0           (S_T_D[1]),
                .S_D1           (S_T_D[2]),
                .S_D2           (S_T_D[3]),
                .SUB_TYP        (SUB_TYP)
               );

ZC_P13_3        ZC_3(
                .CXD(CXD_3          ),

                .S_B0           (S_T_B[2]),
                .S_B1           (S_T_B[3]),
                .S_B2           (S_T_B[4]),
                .S_C0           (S_T_C[2]),
                .V0_Cn          (V_T_C[3]),
                 .S_C2          (S_T_C[4]),
                .S_D0           (S_T_D[2]),
                .S_D1           (S_T_D[3]),
                .S_D2           (S_T_D[4]),
                .SUB_TYP        (SUB_TYP)
               );

ZC_P13_4                ZC_4(
                .CXD(CXD_4          ),

                .S_B0           (S_T_B[3]),
                .S_B1           (S_T_B[4]),
                .S_B2           (S_T_B[5]),
                .S_C0           (S_T_C[3]),
                .V0_Cn          (V_T_C[4]),
                .S_C2           (S_T_C[5]),
                .S_D0           (S_T_D[3]),
                .S_D1           (S_T_D[4]),
```

```
                .S_D2          (S_T_D[5]),
                .SUB_TYP   (SUB_TYP)
               );
endmodule
```

A.6 The hardware description language of the sign coding for pass1 coding operation or pass3 coding operation is expressed as follows:

```
module  SC_COLUMN_P1_P3(
        CXD_1,
        CXD_2,
        CXD_3,
        CXD_4,

        X0_B,
        X0_C,
        X0_D,
        S1_B,
        S1_C,
        S1_D,
        S3_B,
        S3_C,
        S3_D,
        P1_S_B,
        P1_S_C,
              P1_B_C,
        P3_S_C
              );
output   [5:0]   CXD_1;
output   [5:0]   CXD_2;
output   [5:0]   CXD_3;
output   [5:0]   CXD_4;
input    [4:0]   X0_B;
input    [4:0]   X0_C;
input    [4:0]   X0_D;
```

```
input   [4:0]   S1_B;
input   [4:0]   S1_C;
input   [4:0]   S1_D;
input   [4:0]    S3_B;
input   [4:0]    S3_C;
input   [4:0]   S3_D;
input   [4:1]   P1_S_B;
input   [4:1]   P1_S_C;
input   [4:1]   P1_B_C;
input   [4:1]   P3_S_C;
wire            S_V1_1;
wire            S_V2_1;
wire            S_H1_1;
wire            S_H2_1;
wire            X_V1_1;
wire            X_V2_1;
wire            X_H1_1;
wire            X_H2_1;
wire            X_C_1;
wire            S_V1_2;
wire            S_V2_2;
wire            S_H1_2;
wire            S_H2_2;
wire            X_V1_2;
wire            X_V2_2;
wire            X_H1_2;
wire            X_H2_2;
wire            X_C_2;

wire            S_V1_3;
wire            S_V2_3;
wire            S_H1_3;
wire            S_H2_3;
wire            X_V1_3;
wire            X_V2_3;
wire            X_H1_3;
wire            X_H2_3;
wire            X_C_3;
wire            S_V1_4;
```

```
wire            S_V2_4;
wire            S_H1_4;
wire            S_H2_4;
wire            X_V1_4;
wire            X_V2_4;
wire            X_H1_4;
wire            X_H2_4;
wire            X_C_4;
//------------FOR THE FIRST SAMPLE IN REGISTER C--------
assign  S_V1_1 = (P1_B_C[1])? S1_C[0] : S1_C[0]|S3_C[0];
assign  S_V2_1 = (P1_B_C[1])? S1_C[2]|S3_C[2] : S1_C[2]|S3_C[2]|P1_S_C[2];
assign  S_H1_1 = (P1_B_C[1])? S1_B[1]|S3_B[1] : S1_B[1]|S3_B[1]|P1_S_B[1];
assign  S_H2_1 = (P1_B_C[1])? S1_D[1]          : S1_D[1]|S3_D[1];
assign  X_V1_1 = X0_C[0]; // S1_D[n]=S1_C[n]|P1_S_C[n]
assign  X_V2_1 = X0_C[2]; // S3_D[n]=S3_C[n]|P3_S_C[n]
assign  X_H1_1 = X0_B[1];
assign  X_H2_1 = X0_D[1];
assign  X_C_1 = X0_C[1];
//------------FOR THE SECOND SAMPLE IN REGISTER C--------
assign  S_V1_2 = (P1_B_C[2])? S1_C[1]|P1_S_C[1] : S1_C[1]|S3_C[1]|P1_S_C[1]|P3_S_C[1];
assign  S_V2_2 = (P1_B_C[2])? S1_C[3]|S3_C[3] : S1_C[3]|S3_C[3]|P1_S_C[3];
assign  S_H1_2 = (P1_B_C[2])? S1_B[2]|S3_B[2] : S1_B[2]|S3_B[2]|P1_S_B[2];
assign  S_H2_2 = (P1_B_C[2])? S1_D[2]          : S1_D[2]|S3_D[2];
assign  X_V1_2 = X0_C[1];
assign  X_V2_2 = X0_C[3];
assign  X_H1_2 = X0_B[2];
assign  X_H2_2 = X0_D[2];
assign  X_C_2 = X0_C[2];
//------------FOR THE THIRD SAMPLE IN REGISTER C--------
assign  S_V1_3 = (P1_B_C[3])? S1_C[2]|P1_S_C[2] : S1_C[2]|S3_C[2]|P1_S_C[2]|P3_S_C[2];
assign  S_V2_3 = (P1_B_C[3])? S1_C[4]|S3_C[4] : S1_C[4]|S3_C[4]|P1_S_C[4];
assign  S_H1_3 = (P1_B_C[3])? S1_B[3]|S3_B[3] : S1_B[3]|S3_B[3]|P1_S_B[3];
assign  S_H2_3 = (P1_B_C[3])? S1_D[3]          : S1_D[3]|S3_D[3];
assign  X_V1_3 = X0_C[2];
assign  X_V2_3 = X0_C[4];
assign  X_H1_3 = X0_B[3];
assign  X_H2_3 = X0_D[3];
assign  X_C_3 = X0_C[3];
//------------FOR THE FOURTH SAMPLE IN REGISTER C--------
```

```
assign    S_V1_4 = (P1_B_C[4])? S1_C[3]|P1_S_C[3] : S1_C[3]|S3_C[3]|P1_S_C[3]|P3_S_C[3];
assign    S_V2_4 = 1'b0;
assign    S_H1_4 = (P1_B_C[4])? S1_B[4]|S3_B[4] : S1_B[4]|S3_B[4]|P1_S_B[4];
assign    S_H2_4 = (P1_B_C[4])? S1_D[4]                : S1_D[4]|S3_D[4];
assign    X_V1_4 = X0_C[3];
assign    X_V2_4 = 1'b0     ;
assign    X_H1_4 = X0_B[4];
assign    X_H2_4 = X0_D[4];
assign    X_C_4 = X0_C[4];
//------------------------------------------------------------
SC_P13_1       SC_1(
              .CXD     (CXD_1     ),

              .S_V1       (S_V1_1    ),
              .S_V2       (S_V2_1    ),
              .S_H1       (S_H1_1    ),
              .S_H2       (S_H2_1    ),
              .X_V1       (X_V1_1    ),
              .X_V2       (X_V2_1    ),
              .X_H1       (X_H1_1    ),
              .X_H2       (X_H2_1    ),
              .X_Cn       (X_C_1     )
            );
         SC_P13_2       SC_2(
              .CXD     (CXD_2     ),

              .S_V1       (S_V1_2    ),
              .S_V2       (S_V2_2    ),
              .S_H1       (S_H1_2    ),
              .S_H2       (S_H2_2    ),
              .X_V1       (X_V1_2    ),
              .X_V2       (X_V2_2    ),
              .X_H1       (X_H1_2    ),
              .X_H2       (X_H2_2    ),
              .X_Cn       (X_C_2     )
            );
         SC_P13_3       SC_3(
              .CXD     (CXD_3     ),
```

```
        .S_V1       (S_V1_3    ),
        .S_V2       (S_V2_3    ),
        .S_H1       (S_H1_3    ),
        .S_H2       (S_H2_3    ),
        .X_V1       (X_V1_3    ),
        .X_V2       (X_V2_3    ),
        .X_H1       (X_H1_3    ),
        .X_H2       (X_H2_3    ),
        .X_Cn       (X_C_3     )
      );
    SC_P13_4       SC_4(
        .CXD      (CXD_4      ),

        .S_V1       (S_V1_4    ),
        .S_V2       (S_V2_4    ),
        .S_H1       (S_H1_4    ),
        .S_H2       (S_H2_4    ),
        .X_V1       (X_V1_4    ),
        .X_V2       (X_V2_4    ),
        .X_H1       (X_H1_4    ),
        .X_H2       (X_H2_4    ),
        .X_Cn       (X_C_4     )
      );
    endmodule
```

A.7 Design Compiler Script File

```
target_library = {typical.db}
symbol_library = {tsmc25.sdb}
synthetic_library = {dw_foundation.sldb}
link_library = {"*" slow.db dw_foundation.sldb}
/*read_verilog*/
read -format verilog PCF.v
read -format verilog PCF_CONTROL.v
read -format verilog P1_PRE_CHECK.v
read -format verilog P1_V_CHECK.v
read -format verilog P1_H_CHECK.v
read -format verilog PCPG.v
read -format verilog MRC_COLUMN.v
read -format verilog MRC1.v
```

```
read -format verilog MRC2.v
read -format verilog MRC3.v
read -format verilog MRC4.v
read -format verilog ZC_COLUMN_P1_P3.v
read -format verilog ZC_P13_1.v
read -format verilog ZC_P13_2.v
read -format verilog ZC_P13_3.v
read -format verilog ZC_P13_4.v
read -format verilog SC_COLUMN_P1_P3.v
read -format verilog SC_P13_1.v
read -format verilog SC_P13_2.v
read -format verilog SC_P13_3.v
read -format verilog SC_P13_4.v
read -format verilog RLC_COLUMN_0.v
read -format verilog RLC_COLUMN_1.v
read -format verilog RLC1_UC0.v
read -format verilog RLC1_UC1.v
current_design PCF
link
create_clock CLK -name CLK -period 3 -waveform {0 1.5}
set_dont_touch_network CLK
check_design > pcf_chk.log
compile -map_effort high -area_effort high
change_names -rules verilog -verbose -hierarchy
write -format verilog -hierarchy -output PCFALL.hv
write_sdf PCFALL.sdf
report_area > area.rpt
report_cell > areacell.rpt
report_timing > timing
remove_design -all
exit
```

A.8 Gate-Level Netlist

```
module PCF_CONTROL_DW01_inc_10_1 ( A, SUM );
input    [9:0] A;
output [9:0] SUM;
      wire carry_9, carry_4, carry_2, carry_6, carry_7, carry_3, carry_8,
           carry_5;
      XOR2X2 U5 ( .A(carry_9), .B(A[9]), .Y(SUM[9]) );
```

```
        ADDHXL U1_1_2 ( .A(A[2]), .B(carry_2), .S(SUM[2]), .CO(carry_3) );
        INVX2 U6 ( .A(A[0]), .Y(SUM[0]) );
        ADDHX2 U1_1_3 ( .A(A[3]), .B(carry_3), .S(SUM[3]), .CO(carry_4) );
        ADDHX2 U1_1_4 ( .A(A[4]), .B(carry_4), .S(SUM[4]), .CO(carry_5) );
        ADDHX2 U1_1_1 ( .A(A[1]), .B(A[0]), .S(SUM[1]), .CO(carry_2) );
        ADDHX2 U1_1_5 ( .A(A[5]), .B(carry_5), .S(SUM[5]), .CO(carry_6) );
        ADDHX2 U1_1_7 ( .A(A[7]), .B(carry_7), .S(SUM[7]), .CO(carry_8) );
        ADDHX2 U1_1_6 ( .A(A[6]), .B(carry_6), .S(SUM[6]), .CO(carry_7) );
        ADDHX2 U1_1_8 ( .A(A[8]), .B(carry_8), .S(SUM[8]), .CO(carry_9) );
endmodule
module PCF_CONTROL_DW01_sub_10_1 ( A, B, CI, DIFF, CO );
input  [9:0] A;
input  [9:0] B;
output [9:0] DIFF;
input  CI;
output CO;
        wire carry_9, carry_4, carry_6, \DIFF[0] , \DIFF[1] , carry_8, carry_7,
             carry_5;
        assign \DIFF[1]  = A[1];
        assign \DIFF[0]  = A[0];
        assign DIFF[1] = \DIFF[1] ;
        assign DIFF[0] = \DIFF[0] ;
        XNOR2XL U6 ( .A(A[9]), .B(carry_9), .Y(DIFF[9]) );
        OR2X1 U7 ( .A(A[8]), .B(carry_8), .Y(carry_9) );
        XNOR2XL U8 ( .A(carry_8), .B(A[8]), .Y(DIFF[8]) );
        OR2X1 U9 ( .A(A[7]), .B(carry_7), .Y(carry_8) );
        XNOR2XL U10 ( .A(carry_7), .B(A[7]), .Y(DIFF[7]) );
        OR2X1 U11 ( .A(A[6]), .B(carry_6), .Y(carry_7) );
        XNOR2XL U12 ( .A(carry_6), .B(A[6]), .Y(DIFF[6]) );
        OR2X1 U13 ( .A(A[5]), .B(carry_5), .Y(carry_6) );
        XNOR2XL U14 ( .A(carry_5), .B(A[5]), .Y(DIFF[5]) );
        OR2X1 U15 ( .A(A[4]), .B(carry_4), .Y(carry_5) );
        XNOR2XL U16 ( .A(carry_4), .B(A[4]), .Y(DIFF[4]) );
        OR2X1 U17 ( .A(A[3]), .B(A[2]), .Y(carry_4) );
        XNOR2XL U18 ( .A(A[2]), .B(A[3]), .Y(DIFF[3]) );
        CLKINVX1 U19 ( .A(A[2]), .Y(DIFF[2]) );
endmodule
module PCF_CONTROL_DW01_cmp2_32_0 ( A, B, LEQ, TC, LT_LE, GE_GT );
input  [31:0] A;
```

```
input    [31:0] B;
input    LEQ, TC;
output LT_LE, GE_GT;
      wire n44, n45, n46, n47, n48, n49, n212, n213, n214, n215, n216, n217,
           n218, n219, n220, n221, n222, n223, n224, n225, n226, n227, n228, n229,
           n230, n231, n232, n233, n234, n235, n236, n237, n238, n239;
      NOR2XL U8 ( .A(B[21]), .B(B[11]), .Y(n228) );
      NOR2XL U9 ( .A(B[29]), .B(B[22]), .Y(n229) );
      NOR2XL U10 ( .A(B[5]), .B(B[8]), .Y(n225) );
      NOR2XL U11 ( .A(B[26]), .B(B[6]), .Y(n220) );
      NOR2XL U12 ( .A(B[25]), .B(B[28]), .Y(n221) );
      NOR2XL U13 ( .A(B[20]), .B(B[16]), .Y(n222) );
      NOR2XL U14 ( .A(B[12]), .B(B[19]), .Y(n223) );
      NOR2XL U15 ( .A(B[18]), .B(B[14]), .Y(n234) );
      NOR2XL U16 ( .A(B[17]), .B(B[13]), .Y(n231) );
      NOR2XL U17 ( .A(B[10]), .B(B[9]), .Y(n232) );
      NAND2BX1 U18 ( .AN(n213), .B(n218), .Y(n212) );
      INVX2 U19 ( .A(B[0]), .Y(n213) );
      NAND3BXL U20 ( .AN(A[0]), .B(n216), .C(n217), .Y(n49) );
      NAND2BX1 U21 ( .AN(B[1]), .B(A[1]), .Y(n216) );
      AOI2BB2X1 U22 ( .A0N(A[3]), .A1N(n215), .B0(n219), .B1(n218), .Y(n239) );
      NOR2BXL U23 ( .AN(B[2]), .B(A[2]), .Y(n219) );
      INVX2 U24 ( .A(B[3]), .Y(n215) );
      NAND4BBX1 U25 ( .AN(n214), .BN(A[1]), .C(n218), .D(n217), .Y(n238) );
      INVX2 U26 ( .A(B[1]), .Y(n214) );
      NAND2BX1 U27 ( .AN(B[3]), .B(A[3]), .Y(n218) );
      NAND2BX1 U28 ( .AN(B[2]), .B(A[2]), .Y(n217) );
      NOR3XL U29 ( .A(n224), .B(n226), .C(n230), .Y(n47) );
      NAND4X1 U30 ( .A(n223), .B(n222), .C(n221), .D(n220), .Y(n224) );
      NAND3BXL U31 ( .AN(B[23]), .B(n227), .C(n225), .Y(n226) );
      INVX2 U32 ( .A(B[15]), .Y(n227) );
      NAND2X1 U33 ( .A(n229), .B(n228), .Y(n230) );
      NOR2XL U34 ( .A(n233), .B(n235), .Y(n46) );
      NAND2X1 U35 ( .A(n232), .B(n231), .Y(n233) );
      NAND3BXL U36 ( .AN(B[27]), .B(n236), .C(n234), .Y(n235) );
      INVX2 U37 ( .A(B[30]), .Y(n236) );
      NOR4XL U38 ( .A(B[24]), .B(B[7]), .C(B[4]), .D(B[31]), .Y(n45) );
      NOR2XL U39 ( .A(n237), .B(n48), .Y(n44) );
      NAND2X1 U40 ( .A(n238), .B(n239), .Y(n237) );
```

```
    NOR2XL U41 ( .A(n49), .B(n212), .Y(n48) );
    NAND4X1 U42 ( .A(n44), .B(n45), .C(n46), .D(n47), .Y(LT_LE) );
endmodule
module PCF_CONTROL_DW01_dec_32_0 ( A, SUM );
input  [31:0] A;
output [31:0] SUM;
    wire C_0_1_1, C_0_0_2, C_0_4_2, C_0_2_2, PG_int_0_0_1, SUM_14, C_0_3_2,
         C_0_7_0, C_0_3_0, PG_int_0_0_2, C_0_0_3, n241, SUM_23, SUM_29, n244,
         n246, n247, n248, n249, SUM_13, n251, SUM_21, n253, SUM_25, SUM_6,
         SUM_11;
    assign SUM[31] = SUM_29;
    assign SUM[30] = SUM_29;
    assign SUM[29] = SUM_29;
    assign SUM[28] = SUM_29;
    assign SUM[27] = SUM_23;
    assign SUM[26] = SUM_25;
    assign SUM[25] = SUM_25;
    assign SUM[24] = SUM_23;
    assign SUM[23] = SUM_23;
    assign SUM[22] = SUM_25;
    assign SUM[21] = SUM_21;
    assign SUM[20] = SUM_21;
    assign SUM[19] = SUM_25;
    assign SUM[18] = SUM_25;
    assign SUM[17] = SUM_21;
    assign SUM[16] = SUM_23;
    assign SUM[15] = SUM_14;
    assign SUM[14] = SUM_14;
    assign SUM[13] = SUM_13;
    assign SUM[12] = SUM_13;
    assign SUM[11] = SUM_11;
    assign SUM[10] = SUM_11;
    assign SUM[9] = SUM_11;
    assign SUM[8] = SUM_11;
    assign SUM[7] = SUM_6;
    assign SUM[6] = SUM_6;
    CLKINVX1 U4 ( .A(SUM_13), .Y(C_0_3_2) );
    NAND3BXL U5 ( .AN(A[1]), .B(n251), .C(SUM[0]), .Y(C_0_4_2) );
    NOR2XL U6 ( .A(A[3]), .B(A[2]), .Y(n247) );
```

```
    NOR3XL U7 ( .A(A[0]), .B(A[2]), .C(A[1]), .Y(n246) );
    XOR2XL U3_C_0_0_2 ( .A(PG_int_0_0_2), .B(C_0_0_2), .Y(SUM[2]) );
    OR2X2 U8 ( .A(A[1]), .B(A[0]), .Y(C_0_0_2) );
    NAND2BX1 U9 ( .AN(A[1]), .B(PG_int_0_0_2), .Y(n253) );
    NAND2BX1 U10 ( .AN(A[0]), .B(n244), .Y(C_0_0_3) );
    XOR2XL U3_C_0_0_3 ( .A(n248), .B(C_0_0_3), .Y(SUM[3]) );
    XOR2XL U3_C_0_0_1 ( .A(PG_int_0_0_1), .B(A[0]), .Y(SUM[1]) );
    INVX2 U11 ( .A(C_0_0_3), .Y(n249) );
    INVX2 U3_A_0_0_0 ( .A(A[0]), .Y(SUM[0]) );
    INVX2 U3_A_0_0_1 ( .A(A[1]), .Y(PG_int_0_0_1) );
    AND2X2 U12 ( .A(n248), .B(n249), .Y(SUM[4]) );
    INVX2 U13 ( .A(A[3]), .Y(n248) );
    INVX2 U3_A_0_0_2 ( .A(A[2]), .Y(PG_int_0_0_2) );
    INVX2 U14 ( .A(C_0_3_0), .Y(SUM_13) );
    INVX2 U15 ( .A(C_0_4_2), .Y(SUM_25) );
    INVX2 U16 ( .A(C_0_1_1), .Y(SUM_6) );
    INVX2 U17 ( .A(C_0_4_2), .Y(SUM_23) );
    NAND2BX1 U18 ( .AN(C_0_0_3), .B(n248), .Y(C_0_1_1) );
    INVX2 U19 ( .A(C_0_1_1), .Y(SUM[5]) );
    NAND2BX1 U20 ( .AN(A[3]), .B(n246), .Y(C_0_2_2) );
    INVX2 U21 ( .A(C_0_7_0), .Y(SUM_29) );
    INVX2 U22 ( .A(C_0_4_2), .Y(SUM_21) );
    INVX2 U23 ( .A(C_0_2_2), .Y(SUM_11) );
    CLKINVX1 U24 ( .A(C_0_3_2), .Y(SUM_14) );
    NAND3BXL U25 ( .AN(A[0]), .B(n247), .C(PG_int_0_0_1), .Y(C_0_3_0) );
    NAND2X1 U26 ( .A(n241), .B(SUM[0]), .Y(C_0_7_0) );
    NOR2XL U27 ( .A(A[3]), .B(n253), .Y(n241) );
    NOR2XL U28 ( .A(A[3]), .B(A[2]), .Y(n251) );
    NOR2X2 U29 ( .A(A[2]), .B(A[1]), .Y(n244) );
endmodule
module PCF_CONTROL ( ADDRA, ADDRB, WRITE, BP_INDEX, START, BP_NUM, RST, CLK );
output [9:0] ADDRA;
input  [3:0] BP_NUM;
output [9:0] ADDRB;
output [3:0] BP_INDEX;
input  START, RST, CLK;
output WRITE;
    wire COUNTER323_3, STATE377_0, ADDRB184_8, N_VP_INDEX263_1, n271_24,
        n271_17, ADDRA175_5, ADDRB184_1, n271_4, n271_29, n271_30, n135,
```

```
      ADDRA175_8, n271_9, COUNTER323_7, ADDRB184_5, ADDRA175_1, n271_13,
      n271_20, ADDRA175_3, n271_22, n271_11, n271_2, ADDRB184_7,
      COUNTER323_5, n271_18, n271_32, ADDRB184_3, COUNTER323_8, n271_6,
      n271_15, n271_26, ADDRA175_7, N_VP_INDEX263_3, COUNTER323_1,
      \COUNTER[7] , BP_INDEX362_2, \COUNTER[3] , \COUNTER[1] , \COUNTER[8] ,
      \STATE[1] , n244, \COUNTER[5] , BP_INDEX362_1, \COUNTER[4] ,
      \COUNTER[0] , \COUNTER[9] , \STATE[0] , \COUNTER[2] , n341,
      \COUNTER[6] , BP_INDEX362_3, ADDRB184_2, COUNTER323_9, n271_7,
      ADDRA175_6, n271_27, n271_14, COUNTER323_0, N_VP_INDEX263_2, n271_23,
      n271_10, ADDRA175_2, n271_3, ADDRB184_6, COUNTER323_4, n271_19,
      ADDRA175_9, n271_31, n271_28, n271_8, COUNTER323_6, n271_1, ADDRB184_4,
      ADDRA175_0, n271_12, n271_21, ADDRB184_9, COUNTER323_2, ADDRA175_4,
      n271_25, n271_16, ADDRB184_0, n271_5, n1029, n1030, n1031, n1032,
      n1033, n1034, n1035, n1036, n1037, n1038, n1039, n1040, n1041, n1042,
      n1043, n1044, n1047, n1048, n1049, n1050, n1051, n1052, n1053, n1054,
      n1055, n1056, n1057, n1058, n1059, n1060, n1061, n1062, n1063, n1064,
      n1065, n1066, n1067, n1068, n1069, n1070, n1071, n1072, n1073, n1074,
      n1075, n1076, n1077, n1078, n1079, n1080, n1081, n1082, n1083, n1084,
      n1085, n1086, n1087, n1088, n1089, n1090, n1091, n1092, n1093, n1094,
      n1095, n1096;
NAND2X1 U123 ( .A(\COUNTER[8] ), .B(\COUNTER[4] ), .Y(n1035) );
NAND3XL U124 ( .A(\COUNTER[0] ), .B(\COUNTER[9] ), .C(\COUNTER[3] ), .Y(
      n1036) );
NAND2X1 U125 ( .A(\COUNTER[6] ), .B(\COUNTER[2] ), .Y(n1033) );
NAND3XL U126 ( .A(\COUNTER[1] ), .B(\COUNTER[7] ), .C(\COUNTER[5] ), .Y(
      n1034) );
NAND2X1 U127 ( .A(n1038), .B(n1037), .Y(n341) );
NOR2XL U128 ( .A(n1034), .B(n1033), .Y(n1038) );
NOR2XL U129 ( .A(n1036), .B(n1035), .Y(n1037) );
NOR2BXL U130 ( .AN(BP_INDEX[2]), .B(n1048), .Y(n1049) );
NAND2X1 U131 ( .A(BP_INDEX[1]), .B(BP_INDEX[0]), .Y(n1048) );
NAND4BX1 U132 ( .AN(n1091), .B(n1078), .C(n1092), .D(n1094), .Y(n1095) );
NAND4X1 U133 ( .A(n1081), .B(n1082), .C(n1083), .D(n1084), .Y(n1091) );
NOR2XL U134 ( .A(\COUNTER[7] ), .B(\COUNTER[4] ), .Y(n1092) );
INVX2 U135 ( .A(n1087), .Y(n1094) );
NAND3XL U136 ( .A(n1076), .B(n1077), .C(\COUNTER[2] ), .Y(n1087) );
NOR3XL U137 ( .A(n1090), .B(n1078), .C(n1079), .Y(n1086) );
NAND3XL U138 ( .A(\COUNTER[0] ), .B(\COUNTER[2] ), .C(\COUNTER[1] ), .Y(
      n1090) );
```

```
NOR3XL U139 ( .A(n1082), .B(n1080), .C(n1081), .Y(n1085) );
NAND2X1 U140 ( .A(n1039), .B(n1040), .Y(n1043) );
NOR2XL U141 ( .A(\COUNTER[4] ), .B(\COUNTER[9] ), .Y(n1039) );
NOR2XL U142 ( .A(\COUNTER[6] ), .B(\COUNTER[8] ), .Y(n1040) );
NAND2X1 U143 ( .A(n1041), .B(n1042), .Y(n1044) );
NOR2XL U144 ( .A(\COUNTER[3] ), .B(\COUNTER[2] ), .Y(n1041) );
NOR2XL U145 ( .A(\COUNTER[7] ), .B(\COUNTER[5] ), .Y(n1042) );
NAND2X1 U146 ( .A(\STATE[1] ), .B(n1073), .Y(n1074) );
NAND2X1 U147 ( .A(\STATE[0] ), .B(n1072), .Y(n1075) );
OAI2BB2X1 U148 ( .A0N(N_VP_INDEX263_2), .A1N(n1030), .B0(n1050), .B1(n1047
      ), .Y(BP_INDEX362_2) );
XOR2XL U149 ( .A(n1047), .B(n1048), .Y(N_VP_INDEX263_2) );
NOR2X1 U150 ( .A(n1056), .B(n1057), .Y(COUNTER323_9) );
NOR2X1 U151 ( .A(n1056), .B(n1058), .Y(COUNTER323_8) );
NOR2X1 U152 ( .A(n1056), .B(n1059), .Y(COUNTER323_7) );
NOR2X1 U153 ( .A(n1056), .B(n1060), .Y(COUNTER323_6) );
NOR2X1 U154 ( .A(n1056), .B(n1061), .Y(COUNTER323_5) );
NOR2X1 U155 ( .A(n1056), .B(n1062), .Y(COUNTER323_4) );
NOR2X1 U156 ( .A(n1056), .B(n1063), .Y(COUNTER323_3) );
NOR2XL U157 ( .A(n1056), .B(n1065), .Y(COUNTER323_1) );
NOR2XL U158 ( .A(n1056), .B(n1066), .Y(COUNTER323_0) );
OAI2BB2X1 U159 ( .A0N(N_VP_INDEX263_3), .A1N(n1030), .B0(n1050), .B1(n1052
      ), .Y(BP_INDEX362_3) );
XOR2XL U160 ( .A(BP_INDEX[3]), .B(n1049), .Y(N_VP_INDEX263_3) );
OAI2BB2X1 U161 ( .A0N(N_VP_INDEX263_1), .A1N(n1030), .B0(n1050), .B1(n1051
      ), .Y(BP_INDEX362_1) );
XOR2XL U162 ( .A(BP_INDEX[1]), .B(BP_INDEX[0]), .Y(N_VP_INDEX263_1) );
NAND2X1 U163 ( .A(RST), .B(n1088), .Y(n1050) );
OAI2BB2X1 U164 ( .A0N(n1093), .A1N(n1095), .B0(n1029), .B1(n1075), .Y(
      n1088) );
INVX2 U165 ( .A(n1074), .Y(n1093) );
INVX2 U166 ( .A(n1075), .Y(n1089) );
AOI31X1 U167 ( .A0(n1053), .A1(n1029), .A2(n1054), .B0(n1055), .Y(
      STATE377_0) );
NAND4X1 U168 ( .A(START), .B(RST), .C(n1072), .D(n1073), .Y(n1053) );
INVX2 U169 ( .A(n135), .Y(n1054) );
NOR2XL U170 ( .A(n1071), .B(n1031), .Y(n1055) );
INVX2 U171 ( .A(n1053), .Y(n1071) );
AOI31X1 U172 ( .A0(n1069), .A1(n1070), .A2(n244), .B0(n1067), .Y(WRITE) );
```

```
NOR2XL U173 ( .A(BP_INDEX[3]), .B(BP_INDEX[2]), .Y(n1069) );
NOR2XL U174 ( .A(BP_INDEX[1]), .B(BP_INDEX[0]), .Y(n1070) );
NOR2XL U175 ( .A(n1044), .B(n1043), .Y(n244) );
AND2X2 U176 ( .A(ADDRB184_0), .B(n1068), .Y(ADDRB[0]) );
AND2X2 U177 ( .A(ADDRB184_1), .B(n1068), .Y(ADDRB[1]) );
AND2X2 U178 ( .A(ADDRB184_2), .B(n1068), .Y(ADDRB[2]) );
AND2X2 U179 ( .A(ADDRB184_3), .B(n1068), .Y(ADDRB[3]) );
AND2X2 U180 ( .A(ADDRB184_4), .B(n1068), .Y(ADDRB[4]) );
AND2X2 U181 ( .A(ADDRB184_5), .B(n1068), .Y(ADDRB[5]) );
AND2X2 U182 ( .A(ADDRB184_6), .B(n1068), .Y(ADDRB[6]) );
AND2X2 U183 ( .A(ADDRB184_7), .B(n1068), .Y(ADDRB[7]) );
AND2X2 U184 ( .A(ADDRB184_8), .B(n1068), .Y(ADDRB[8]) );
AND2X2 U185 ( .A(ADDRB184_9), .B(n1068), .Y(ADDRB[9]) );
INVX2 U186 ( .A(ADDRA175_0), .Y(n1066) );
NOR2XL U187 ( .A(n1067), .B(n1066), .Y(ADDRA[0]) );
INVX2 U188 ( .A(ADDRA175_1), .Y(n1065) );
NOR2XL U189 ( .A(n1067), .B(n1065), .Y(ADDRA[1]) );
NOR2XL U190 ( .A(n1067), .B(n1064), .Y(ADDRA[2]) );
INVX2 U191 ( .A(ADDRA175_2), .Y(n1064) );
INVX2 U192 ( .A(ADDRA175_3), .Y(n1063) );
NOR2XL U193 ( .A(n1067), .B(n1063), .Y(ADDRA[3]) );
INVX2 U194 ( .A(ADDRA175_4), .Y(n1062) );
NOR2XL U195 ( .A(n1067), .B(n1062), .Y(ADDRA[4]) );
INVX2 U196 ( .A(ADDRA175_5), .Y(n1061) );
NOR2XL U197 ( .A(n1067), .B(n1061), .Y(ADDRA[5]) );
INVX2 U198 ( .A(ADDRA175_6), .Y(n1060) );
NOR2XL U199 ( .A(n1067), .B(n1060), .Y(ADDRA[6]) );
INVX2 U200 ( .A(ADDRA175_7), .Y(n1059) );
NOR2XL U201 ( .A(n1067), .B(n1059), .Y(ADDRA[7]) );
INVX2 U202 ( .A(ADDRA175_8), .Y(n1058) );
NOR2XL U203 ( .A(n1067), .B(n1058), .Y(ADDRA[8]) );
INVX2 U204 ( .A(ADDRA175_9), .Y(n1057) );
NAND2X1 U205 ( .A(n1075), .B(n1074), .Y(n1068) );
INVX2 U206 ( .A(n1068), .Y(n1067) );
NOR2XL U207 ( .A(n1067), .B(n1057), .Y(ADDRA[9]) );
DFFXL COUNTER_reg_9_ ( .D(COUNTER323_9), .CK(CLK), .Q(\COUNTER[9] ), .QN(
      n1084) );
DFFXL COUNTER_reg_8_ ( .D(COUNTER323_8), .CK(CLK), .Q(\COUNTER[8] ), .QN(
      n1083) );
```

```
DFFXL COUNTER_reg_7_ ( .D(COUNTER323_7), .CK(CLK), .Q(\COUNTER[7] ), .QN(
      n1080) );
DFFXL COUNTER_reg_6_ ( .D(COUNTER323_6), .CK(CLK), .Q(\COUNTER[6] ), .QN(
      n1082) );
DFFXL COUNTER_reg_5_ ( .D(COUNTER323_5), .CK(CLK), .Q(\COUNTER[5] ), .QN(
      n1081) );
DFFXL COUNTER_reg_4_ ( .D(COUNTER323_4), .CK(CLK), .Q(\COUNTER[4] ), .QN(
      n1079) );
DFFXL COUNTER_reg_3_ ( .D(COUNTER323_3), .CK(CLK), .Q(\COUNTER[3] ), .QN(
      n1078) );
DFFX1 COUNTER_reg_2_ ( .D(COUNTER323_2), .CK(CLK), .Q(\COUNTER[2] ) );
DFFX1 COUNTER_reg_1_ ( .D(COUNTER323_1), .CK(CLK), .Q(\COUNTER[1] ), .QN(
      n1077) );
DFFX1 COUNTER_reg_0_ ( .D(COUNTER323_0), .CK(CLK), .Q(\COUNTER[0] ), .QN(
      n1076) );
DFFXL BP_INDEX_reg_3_ ( .D(BP_INDEX362_3), .CK(CLK), .Q(BP_INDEX[3]), .QN(
      n1052) );
DFFXL BP_INDEX_reg_1_ ( .D(BP_INDEX362_1), .CK(CLK), .Q(BP_INDEX[1]), .QN(
      n1051) );
JKFFXL BP_INDEX_reg_0_ ( .J(n1030), .K(n1050), .CK(CLK), .Q(BP_INDEX[0])
       );
DFFXL STATE_reg_0_ ( .D(STATE377_0), .CK(CLK), .Q(\STATE[0] ), .QN(n1073)
       );
DFFXL BP_INDEX_reg_2_ ( .D(BP_INDEX362_2), .CK(CLK), .Q(BP_INDEX[2]), .QN(
      n1047) );
AND4X2 U208 ( .A(\COUNTER[9] ), .B(\COUNTER[8] ), .C(n1085), .D(n1086),
      .Y(n1029) );
AND2X2 U209 ( .A(n1031), .B(n1029), .Y(n1030) );
AND2X2 U210 ( .A(n1089), .B(RST), .Y(n1031) );
AND3X1 U211 ( .A(n1093), .B(RST), .C(n1087), .Y(n1032) );
PCF_CONTROL_DW01_inc_10_1 r441 ( .A({\COUNTER[9] , \COUNTER[8] ,
      \COUNTER[7] , \COUNTER[6] , \COUNTER[5] , \COUNTER[4] , \COUNTER[3] ,
      \COUNTER[2] , \COUNTER[1] , \COUNTER[0] }), .SUM({ADDRA175_9,
      ADDRA175_8, ADDRA175_7, ADDRA175_6, ADDRA175_5, ADDRA175_4, ADDRA175_3,
      ADDRA175_2, ADDRA175_1, ADDRA175_0}) );
PCF_CONTROL_DW01_sub_10_1 r406 ( .A({\COUNTER[9] , \COUNTER[8] ,
      \COUNTER[7] , \COUNTER[6] , \COUNTER[5] , \COUNTER[4] , \COUNTER[3] ,
      \COUNTER[2] , \COUNTER[1] , \COUNTER[0] }), .B({1'b0, 1'b0, 1'b0, 1'b0,
      1'b0, 1'b0, 1'b0, 1'b1, 1'b0, 1'b0}), .CI(1'b0), .DIFF({ADDRB184_9,
```

```
        ADDRB184_8, ADDRB184_7, ADDRB184_6, ADDRB184_5, ADDRB184_4, ADDRB184_3,
        ADDRB184_2, ADDRB184_1, ADDRB184_0}) );
    SDFFXL STATE_reg_1_ ( .SI(1'b0), .SE(n1096), .D(1'b1), .CK(CLK), .Q(
        \STATE[1] ), .QN(n1072) );
    OAI222X1 U212 ( .A0(n1032), .A1(n1029), .B0(n1032), .B1(n1031), .C0(n1032),
        .C1(n1054), .Y(n1096) );
    PCF_CONTROL_DW01_cmp2_32_0 lt_88 ( .A({1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0,
        1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0,
        1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0,
        BP_INDEX[3], BP_INDEX[2], BP_INDEX[1], BP_INDEX[0]}), .B({n271_1,
        n271_2, n271_3, n271_4, n271_5, n271_6, n271_7, n271_8, n271_9,
        n271_10, n271_11, n271_12, n271_13, n271_14, n271_15, n271_16, n271_17,
        n271_18, n271_19, n271_20, n271_21, n271_22, n271_23, n271_24, n271_25,
        n271_26, n271_27, n271_28, n271_29, n271_30, n271_31, n271_32}), .LEQ(
        1'b0), .TC(1'b0), .LT_LE(n135) );
    PCF_CONTROL_DW01_dec_32_0 sub_88 ( .A({1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0,
        1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0,
        1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, BP_NUM[3],
        BP_NUM[2], BP_NUM[1], BP_NUM[0]}), .SUM({n271_1, n271_2, n271_3,
        n271_4, n271_5, n271_6, n271_7, n271_8, n271_9, n271_10, n271_11,
        n271_12, n271_13, n271_14, n271_15, n271_16, n271_17, n271_18, n271_19,
        n271_20, n271_21, n271_22, n271_23, n271_24, n271_25, n271_26, n271_27,
        n271_28, n271_29, n271_30, n271_31, n271_32}) );
    NOR2X2 U215 ( .A(n1056), .B(n1064), .Y(COUNTER323_2) );
    NAND3X2 U216 ( .A(RST), .B(n1068), .C(n341), .Y(n1056) );
endmodule


module P1_V_CHECK ( P1_B, P1_S_B, F2_P1_B, V0_B );
output [4:1] P1_B;
output [4:1] P1_S_B;
input  [4:1] F2_P1_B;
input  [4:1] V0_B;
    wire \F2_P1_B[1] , n24, n25, n26, n27;
    assign P1_B[1] = \F2_P1_B[1] ;
    assign \F2_P1_B[1]  = F2_P1_B[1];
    AND2X2 U7 ( .A(V0_B[4]), .B(P1_B[4]), .Y(P1_S_B[4]) );
    INVX4 U8 ( .A(n25), .Y(P1_S_B[2]) );
    INVX2 U9 ( .A(n27), .Y(P1_S_B[1]) );
```

```
    INVX4 U10 ( .A(n26), .Y(n24) );
    NAND2X4 U11 ( .A(V0_B[3]), .B(P1_B[3]), .Y(n26) );
    INVXL U12 ( .A(n26), .Y(P1_S_B[3]) );
    OR2X4 U13 ( .A(F2_P1_B[2]), .B(P1_S_B[1]), .Y(P1_B[2]) );
    OR2X4 U14 ( .A(F2_P1_B[3]), .B(P1_S_B[2]), .Y(P1_B[3]) );
    OR2X4 U15 ( .A(F2_P1_B[4]), .B(n24), .Y(P1_B[4]) );
    NAND2X2 U16 ( .A(V0_B[2]), .B(P1_B[2]), .Y(n25) );
    NAND2X2 U17 ( .A(\F2_P1_B[1] ), .B(V0_B[1]), .Y(n27) );
endmodule
module P1_H_CHECK ( F2_P1_B, F1_P1_B, P1_S_C );
output [4:1] F2_P1_B;
input  [4:1] F1_P1_B;
input  [4:1] P1_S_C;
    wire n19, n20, n21, n22;
    OR3X2 U7 ( .A(F1_P1_B[4]), .B(P1_S_C[4]), .C(P1_S_C[3]), .Y(F2_P1_B[4]) );
    INVX2 U8 ( .A(P1_S_C[4]), .Y(n22) );
    INVX2 U9 ( .A(P1_S_C[1]), .Y(n19) );
    INVX2 U10 ( .A(P1_S_C[2]), .Y(n20) );
    INVX2 U11 ( .A(P1_S_C[3]), .Y(n21) );
    NAND4BX2 U12 ( .AN(F1_P1_B[2]), .B(n19), .C(n20), .D(n21), .Y(F2_P1_B[2])
        );
    OR3X4 U13 ( .A(F1_P1_B[1]), .B(P1_S_C[1]), .C(P1_S_C[2]), .Y(F2_P1_B[1])
        );
    NAND4BX2 U14 ( .AN(F1_P1_B[3]), .B(n22), .C(n20), .D(n21), .Y(F2_P1_B[3])
        );
endmodule


module P1_PRE_CHECK ( P1_B, P1_S_B, S1_A, S1_B, S1_C, S3_A, S3_B, V0_B, P1_S_C
    );
output [4:1] P1_B;
output [4:1] P1_S_B;
input  [4:0] S1_A;
input  [4:0] S3_B;
input  [4:0] S1_C;
input  [4:0] S3_A;
input  [4:1] P1_S_C;
input  [4:0] S1_B;
input  [4:1] V0_B;
```

```
    wire \F2_P1_B[4] , \F1_P1_B[1] , \F1_P1_B[3] , \F2_P1_B[2] , \F1_P1_B[2] ,
        \F2_P1_B[3] , \F2_P1_B[1] , \F1_P1_B[4] , n50, n51, n52, n53, n54, n55,
        n56, n57, n58, n59, n60, n61, n62, n63, n64, n65;
    P1_H_CHECK P1_H_CHECK0 ( .F2_P1_B({\F2_P1_B[4] , \F2_P1_B[3] ,
        \F2_P1_B[2] , \F2_P1_B[1] }), .F1_P1_B({\F1_P1_B[4] , \F1_P1_B[3] ,
        \F1_P1_B[2] , \F1_P1_B[1] }), .P1_S_C(P1_S_C) );
    P1_V_CHECK P1_V_CHECK0 ( .P1_B(P1_B), .P1_S_B(P1_S_B), .F2_P1_B({
        \F2_P1_B[4] , \F2_P1_B[3] , \F2_P1_B[2] , \F2_P1_B[1] }), .V0_B(V0_B)
         );
    OR3X2 U6 ( .A(S1_C[2]), .B(S1_A[2]), .C(S3_A[2]), .Y(n57) );
    OR3X2 U7 ( .A(S1_C[3]), .B(S1_A[3]), .C(S3_A[3]), .Y(n56) );
    OR3X2 U8 ( .A(S1_B[0]), .B(S1_A[0]), .C(S1_B[2]), .Y(n65) );
    NOR3X1 U9 ( .A(n65), .B(S1_C[0]), .C(S3_B[2]), .Y(n54) );
    AOI211X1 U10 ( .A0(n50), .A1(n53), .B0(S1_B[4]), .C0(S3_B[4]), .Y(
        \F1_P1_B[4] ) );
    AOI211X1 U11 ( .A0(n55), .A1(n50), .B0(S1_B[3]), .C0(S3_B[3]), .Y(
        \F1_P1_B[3] ) );
    AND4X2 U12 ( .A(n58), .B(n59), .C(n60), .D(n61), .Y(n55) );
    INVX2 U13 ( .A(n57), .Y(n61) );
    INVX2 U14 ( .A(n56), .Y(n64) );
    AOI211X1 U15 ( .A0(n52), .A1(n51), .B0(S1_B[2]), .C0(S3_B[2]), .Y(
        \F1_P1_B[2] ) );
    INVX2 U16 ( .A(S1_B[4]), .Y(n58) );
    INVX2 U17 ( .A(S1_B[3]), .Y(n53) );
    INVX2 U18 ( .A(S1_B[2]), .Y(n60) );
    INVX2 U19 ( .A(S1_B[1]), .Y(n63) );
    INVX2 U20 ( .A(S3_B[4]), .Y(n59) );
    INVX2 U21 ( .A(S3_B[3]), .Y(n62) );
    NOR4XL U22 ( .A(S1_C[4]), .B(S1_A[4]), .C(S3_A[4]), .D(n56), .Y(n50) );
    NOR4X1 U23 ( .A(S1_C[1]), .B(S1_A[1]), .C(S3_A[1]), .D(n57), .Y(n51) );
    AOI211X2 U24 ( .A0(n54), .A1(n51), .B0(S1_B[1]), .C0(S3_B[1]), .Y(
        \F1_P1_B[1] ) );
    AND4X4 U25 ( .A(n62), .B(n63), .C(n53), .D(n64), .Y(n52) );
endmodule
module PCPG ( P1_S_C, P2_C, P2_1_C, P1_B_C, P3_C, P3_S_C, P3_RLC0, P3_RLC1,
    P3_RLC1_UC1, P3_RLC1_UC0, S3_C, S1_C, V0_C, P1_B, START, RST, CLK );
output [4:1] P1_S_C;
output [4:1] P1_B_C;
input  [4:0] S3_C;
```

```
input    [4:1] P1_B;
input    [4:1] V0_C;
output [4:1] P2_C;
output [4:1] P2_1_C;
output [4:1] P3_C;
input    [4:0] S1_C;
output [4:1] P3_S_C;
input    START, RST, CLK;
output P3_RLC0, P3_RLC1, P3_RLC1_UC1, P3_RLC1_UC0;
      wire n160, n183, n184, n185, n186, n187, n188, n189, n190, n191, n192,
           n193, n194, n195, n196, n197, n198, n199, n200, n201, n202, n203, n204,
           n205, n206, n207, n208, n209, n210, n211;
      NOR2XL U9 ( .A(n195), .B(n196), .Y(P3_S_C[4]) );
      NOR2X1 U10 ( .A(n191), .B(n194), .Y(P1_S_C[3]) );
      NOR2X1 U11 ( .A(n197), .B(n198), .Y(P1_S_C[2]) );
      NOR2XL U12 ( .A(n191), .B(n192), .Y(P3_S_C[3]) );
      NOR2XL U13 ( .A(n197), .B(n201), .Y(P3_S_C[2]) );
      NOR2XL U14 ( .A(n199), .B(n205), .Y(P3_S_C[1]) );
      SDFFTRX1 P1_B_C_reg_1_ ( .SI(1'b0), .SE(START), .D(P1_B[1]), .CK(CLK),
           .RN(RST), .Q(P1_B_C[1]), .QN(n200) );
      NOR2XL U15 ( .A(V0_C[3]), .B(n195), .Y(n207) );
      NAND2X1 U16 ( .A(n202), .B(n208), .Y(n196) );
      NAND2X1 U17 ( .A(n194), .B(n209), .Y(n192) );
      NAND2X1 U18 ( .A(n198), .B(n210), .Y(n201) );
      NAND2X1 U19 ( .A(n200), .B(n211), .Y(n205) );
      NOR2X1 U20 ( .A(n195), .B(n202), .Y(P1_S_C[4]) );
      NOR2X1 U21 ( .A(n199), .B(n200), .Y(P1_S_C[1]) );
      NAND2X1 U22 ( .A(n183), .B(n184), .Y(P2_C[1]) );
      INVX2 U23 ( .A(S3_C[1]), .Y(n183) );
      INVX2 U24 ( .A(P2_C[1]), .Y(n211) );
      NAND2X1 U25 ( .A(n185), .B(n186), .Y(P2_C[2]) );
      INVX2 U26 ( .A(S3_C[2]), .Y(n185) );
      INVX2 U27 ( .A(S1_C[2]), .Y(n186) );
      INVX2 U28 ( .A(P2_C[2]), .Y(n210) );
      NAND2X1 U29 ( .A(n187), .B(n188), .Y(P2_C[3]) );
      INVX2 U30 ( .A(S3_C[3]), .Y(n188) );
      INVX2 U31 ( .A(P2_C[3]), .Y(n209) );
      INVX2 U32 ( .A(P2_C[4]), .Y(n208) );
      INVX2 U33 ( .A(S3_C[4]), .Y(n190) );
```

```
    NAND2X1 U34 ( .A(n189), .B(n190), .Y(P2_C[4]) );
    NOR3XL U35 ( .A(n160), .B(V0_C[1]), .C(V0_C[2]), .Y(P3_RLC1_UC1) );
    NOR2XL U36 ( .A(V0_C[1]), .B(n193), .Y(P3_RLC1_UC0) );
    NOR2XL U37 ( .A(n207), .B(V0_C[2]), .Y(n193) );
    NOR2XL U38 ( .A(n206), .B(n203), .Y(P3_RLC1) );
    INVX2 U39 ( .A(n204), .Y(n206) );
    NAND3XL U40 ( .A(n197), .B(n199), .C(n160), .Y(n204) );
    NAND4X1 U41 ( .A(P3_C[1]), .B(P3_C[2]), .C(P3_C[3]), .D(P3_C[4]), .Y(n203)
        );
    INVX2 U42 ( .A(n205), .Y(P3_C[1]) );
    INVX2 U43 ( .A(n201), .Y(P3_C[2]) );
    INVX2 U44 ( .A(n192), .Y(P3_C[3]) );
    INVX2 U45 ( .A(n196), .Y(P3_C[4]) );
    NOR2XL U46 ( .A(n203), .B(n204), .Y(P3_RLC0) );
    XOR2XL U47 ( .A(S1_C[4]), .B(S3_C[4]), .Y(P2_1_C[4]) );
    XOR2XL U48 ( .A(S1_C[3]), .B(S3_C[3]), .Y(P2_1_C[3]) );
    XOR2XL U49 ( .A(S3_C[2]), .B(S1_C[2]), .Y(P2_1_C[2]) );
    XOR2XL U50 ( .A(S3_C[1]), .B(S1_C[1]), .Y(P2_1_C[1]) );
    INVX2 U51 ( .A(V0_C[4]), .Y(n195) );
    INVX2 U52 ( .A(V0_C[3]), .Y(n191) );
    INVX2 U53 ( .A(V0_C[2]), .Y(n197) );
    INVX2 U54 ( .A(V0_C[1]), .Y(n199) );
    INVX2 U55 ( .A(S1_C[4]), .Y(n189) );
    INVX2 U56 ( .A(S1_C[3]), .Y(n187) );
    INVX2 U57 ( .A(S1_C[1]), .Y(n184) );
    AND2X2 U58 ( .A(n191), .B(n195), .Y(n160) );
    SDFFTRX4 P1_B_C_reg_4_ ( .SI(1'b0), .SE(START), .D(P1_B[4]), .CK(CLK),
        .RN(RST), .Q(P1_B_C[4]), .QN(n202) );
    SDFFTRX4 P1_B_C_reg_3_ ( .SI(1'b0), .SE(START), .D(P1_B[3]), .CK(CLK),
        .RN(RST), .Q(P1_B_C[3]), .QN(n194) );
    SDFFTRX4 P1_B_C_reg_2_ ( .SI(1'b0), .SE(START), .D(P1_B[2]), .CK(CLK),
        .RN(RST), .Q(P1_B_C[2]), .QN(n198) );
endmodule
module MRC1 ( CXD, B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn );
output [5:0] CXD;
input  B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn;
    wire V0_Cn_wire, P2_1_Cn_wire, n75, n76, n77;
    assign CXD[5] = V0_Cn_wire;
    assign CXD[3] = P2_1_Cn_wire;
```

```
        assign CXD[2] = P2_1_Cn_wire;
        assign CXD[1] = P2_1_Cn_wire;
        assign V0_Cn_wire = V0_Cn;
        assign P2_1_Cn_wire = P2_1_Cn;
        OR4X1 U6 ( .A(D0), .B(D2), .C(n75), .D(C2), .Y(n76) );
        OR3X2 U7 ( .A(C0), .B(D1), .C(B0), .Y(n75) );
        INVX2 U8 ( .A(n77), .Y(CXD[0]) );
        OAI31X1 U9 ( .A0(B2), .A1(n76), .A2(B1), .B0(P2_1_Cn_wire), .Y(n77) );
        INVX2 U10 ( .A(P2_1_Cn_wire), .Y(CXD[4]) );
endmodule
module MRC2 ( CXD, B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn );
output [5:0] CXD;
input   B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn;
        wire V0_Cn_wire, P2_1_Cn_wire, n66, n67, n68;
        assign CXD[5] = V0_Cn_wire;
        assign CXD[3] = P2_1_Cn_wire;
        assign CXD[2] = P2_1_Cn_wire;
        assign CXD[1] = P2_1_Cn_wire;
        assign V0_Cn_wire = V0_Cn;
        assign P2_1_Cn_wire = P2_1_Cn;
        OR4X1 U6 ( .A(D0), .B(C2), .C(n67), .D(B0), .Y(n66) );
        OR3X2 U7 ( .A(D2), .B(D1), .C(C0), .Y(n67) );
        INVX2 U8 ( .A(n68), .Y(CXD[0]) );
        OAI31X1 U9 ( .A0(B2), .A1(n66), .A2(B1), .B0(P2_1_Cn_wire), .Y(n68) );
        INVX2 U10 ( .A(P2_1_Cn_wire), .Y(CXD[4]) );
endmodule
module MRC3 ( CXD, B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn );
output [5:0] CXD;
input   B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn;
        wire V0_Cn_wire, P2_1_Cn_wire, n72, n73, n74;
        assign CXD[5] = V0_Cn_wire;
        assign CXD[3] = P2_1_Cn_wire;
        assign CXD[2] = P2_1_Cn_wire;
        assign CXD[1] = P2_1_Cn_wire;
        assign V0_Cn_wire = V0_Cn;
        assign P2_1_Cn_wire = P2_1_Cn;
        OR4X1 U6 ( .A(C0), .B(D1), .C(n72), .D(B0), .Y(n73) );
        OR3X2 U7 ( .A(D0), .B(D2), .C(C2), .Y(n72) );
        INVX2 U8 ( .A(n74), .Y(CXD[0]) );
```

```
    OAI31X1 U9 ( .A0(B2), .A1(B1), .A2(n73), .B0(P2_1_Cn_wire), .Y(n74) );
    INVX2 U10 ( .A(P2_1_Cn_wire), .Y(CXD[4]) );
endmodule
module MRC4 ( CXD, B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn );
output [5:0] CXD;
input  B0, B1, B2, C0, C2, D0, D1, D2, V0_Cn, P2_1_Cn;
    wire V0_Cn_wire, P2_1_Cn_wire, n69, n70, n71;
    assign CXD[5] = V0_Cn_wire;
    assign CXD[3] = P2_1_Cn_wire;
    assign CXD[2] = P2_1_Cn_wire;
    assign CXD[1] = P2_1_Cn_wire;
    assign V0_Cn_wire = V0_Cn;
    assign P2_1_Cn_wire = P2_1_Cn;
    OR4X1 U6 ( .A(D1), .B(D0), .C(n70), .D(C0), .Y(n69) );
    OR3X2 U7 ( .A(D2), .B(B2), .C(C2), .Y(n70) );
    INVX2 U8 ( .A(n71), .Y(CXD[0]) );
    OAI31X1 U9 ( .A0(B1), .A1(B0), .A2(n69), .B0(P2_1_Cn_wire), .Y(n71) );
    INVX2 U10 ( .A(P2_1_Cn_wire), .Y(CXD[4]) );
endmodule
module MRC_COLUMN ( CXD_1, CXD_2, CXD_3, CXD_4, V0_C, S1_B, S1_C, S1_D, S3_B,
    S3_C, S3_D, P1_S_B, P1_S_C, P2_1_C );
output [5:0] CXD_1;
output [5:0] CXD_2;
input  [4:0] S3_B;
output [5:0] CXD_3;
input  [4:0] S3_D;
output [5:0] CXD_4;
input  [4:1] V0_C;
input  [4:0] S1_D;
input  [4:0] S3_C;
input  [4:0] S1_C;
input  [4:1] P1_S_C;
input  [4:1] P2_1_C;
input  [4:0] S1_B;
input  [4:1] P1_S_B;
    wire C0__2, B1__1, C0__4, C2__2, \B2_[2] , C2__1, \B2_[3] , \B2_[1] ,
        C2__3, C0__3;
    MRC1 MRC_1 ( .CXD(CXD_1), .B0(S1_B[0]), .B1(B1__1), .B2(\B2_[1] ), .C0(
        S1_C[0]), .C2(C2__1), .D0(S1_D[0]), .D1(S1_D[1]), .D2(S1_D[2]),
```

```
        .V0_Cn(V0_C[1]), .P2_1_Cn(P2_1_C[1]) );
    MRC3 MRC_3 ( .CXD(CXD_3), .B0(\B2_[1] ), .B1(\B2_[2] ), .B2(\B2_[3] ),
        .C0(C0__3), .C2(C2__3), .D0(S1_D[2]), .D1(S1_D[3]), .D2(S1_D[4]),
        .V0_Cn(V0_C[3]), .P2_1_Cn(P2_1_C[3]) );
    MRC4 MRC_4 ( .CXD(CXD_4), .B0(\B2_[2] ), .B1(\B2_[3] ), .B2(1'b0), .C0(
        C0__4), .C2(1'b0), .D0(S1_D[3]), .D1(S1_D[4]), .D2(1'b0), .V0_Cn(V0_C
        [4]), .P2_1_Cn(P2_1_C[4]) );
    MRC2 MRC_2 ( .CXD(CXD_2), .B0(B1__1), .B1(\B2_[1] ), .B2(\B2_[2] ), .C0(
        C0__2), .C2(C2__2), .D0(S1_D[1]), .D1(S1_D[2]), .D2(S1_D[3]), .V0_Cn(
        V0_C[2]), .P2_1_Cn(P2_1_C[2]) );
    OR3X2 U6 ( .A(S3_C[4]), .B(S1_C[4]), .C(P1_S_C[4]), .Y(C2__3) );
    OR2X2 U7 ( .A(S1_C[1]), .B(P1_S_C[1]), .Y(C0__2) );
    OR2X2 U8 ( .A(S1_C[3]), .B(P1_S_C[3]), .Y(C0__4) );
    OR2X2 U9 ( .A(S1_C[2]), .B(P1_S_C[2]), .Y(C0__3) );
    OR3X2 U10 ( .A(S1_B[4]), .B(S3_B[4]), .C(P1_S_B[4]), .Y(\B2_[3] ) );
    OR2X2 U11 ( .A(C0__4), .B(S3_C[3]), .Y(C2__2) );
    OR3X2 U12 ( .A(S3_B[3]), .B(S1_B[3]), .C(P1_S_B[3]), .Y(\B2_[2] ) );
    OR2X2 U13 ( .A(C0__3), .B(S3_C[2]), .Y(C2__1) );
    OR3X2 U14 ( .A(S3_B[1]), .B(S1_B[1]), .C(P1_S_B[1]), .Y(B1__1) );
    OR3XL U15 ( .A(S3_B[2]), .B(S1_B[2]), .C(P1_S_B[2]), .Y(\B2_[1] ) );
endmodule
module ZC_P13_1 ( CXD, S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2,
    SUB_TYP );
output [5:0] CXD;
input  [1:0] SUB_TYP;
input  S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2;
    wire CXD299_1, SUM_H_0, CXD299_3, SUM_HV_0, SUM_D_1, n265_3, n323, n259_3,
        V0_Cn_wire, n259_2, SUM_D_0, n265_2, SUM_D_2, SUM_V_0, CXD299_2, n375,
        n309, n300_4, CXD299_0, n551, n552, n553, n917, n918, n919, n920, n921,
        n922, n923, n924, n925, acell_4820_U147_Z_0, acell_4820_U147_Z_1, n927,
        n928, n929, n930, n931, n932, n933, n934, n935, n936, n937, n938, n939,
        n940, n941, n942, n943, n944, n945, n946, n947, n948, n949, n950, n951,
        n952, n953, n954, n955, n956, n957, n958, n959, n960, n961, n962, n963,
        n964, n965, n966, n967, add_61_3_g_array_0_0, add_61_3_g_array_1_0,
        add_61_3_pog_array_0_0, n968, n969, n970, n971, add_63_3_g_array_0_0,
        add_63_3_g_array_1_0, add_63_3_pog_array_0_0, n972, n973, n974, n975,
        n976, n977, n978, n979, n980, n981, n982, n983, n984, n985, n986, n987,
        n988, n989, n990, n991, n992, n993, n994, n995, n996, n997, n998, n999,
        n1000, n1001, n1002, n1003, n1004, n1005, n1006, n1007, n1008, n1009,
```

```
      n1010, n1011, n1012, n1013, n1014, n1015, n1016, n1017, n1018, n1019,
      n1020, n1021, n1022, n1023, n1024, n1025, n1026, n1027, n1028;
assign CXD[5] = V0_Cn_wire;
assign V0_Cn_wire = V0_Cn;
NAND2X1 U178 ( .A(S_C2), .B(S_D1), .Y(n969) );
OR2X2 U179 ( .A(S_D1), .B(S_C2), .Y(add_63_3_pog_array_0_0) );
NAND2X1 U180 ( .A(S_D2), .B(S_D0), .Y(n965) );
OR2X2 U181 ( .A(S_D0), .B(S_D2), .Y(add_61_3_pog_array_0_0) );
MXI2XL U182 ( .S0(S_B0), .B(n964), .A(n967), .Y(n259_3) );
XNOR2XL U183 ( .A(S_D2), .B(S_D0), .Y(n967) );
NOR2BXL U184 ( .AN(n965), .B(n966), .Y(n964) );
INVX2 U185 ( .A(add_61_3_pog_array_0_0), .Y(n966) );
NAND2X1 U186 ( .A(n963), .B(n961), .Y(n309) );
MXI2XL U187 ( .S0(S_C0), .B(n968), .A(n971), .Y(n265_3) );
XNOR2XL U188 ( .A(S_C2), .B(S_D1), .Y(n971) );
NOR2BXL U189 ( .AN(n969), .B(n970), .Y(n968) );
INVX2 U190 ( .A(add_63_3_pog_array_0_0), .Y(n970) );
INVX2 U191 ( .A(n969), .Y(add_63_3_g_array_0_0) );
NAND3XL U192 ( .A(n927), .B(n928), .C(n952), .Y(n948) );
XNOR2XL U193 ( .A(n259_2), .B(n924), .Y(SUM_D_1) );
INVX2 U194 ( .A(n965), .Y(add_61_3_g_array_0_0) );
INVX2 U195 ( .A(SUM_D_1), .Y(n961) );
OAI21XL U196 ( .A0(n1025), .A1(n990), .B0(n1018), .Y(n1017) );
NOR2XL U197 ( .A(n991), .B(n992), .Y(n990) );
NAND3XL U198 ( .A(n1007), .B(n1006), .C(n919), .Y(n1016) );
NAND3XL U199 ( .A(n1024), .B(n1022), .C(n1027), .Y(n1019) );
NAND2X1 U200 ( .A(n1023), .B(n1012), .Y(n1022) );
NAND4X1 U201 ( .A(SUM_HV_0), .B(n1011), .C(n1026), .D(n1001), .Y(n1027) );
INVX2 U202 ( .A(n1022), .Y(n991) );
OAI31X1 U203 ( .A0(n1021), .A1(n1001), .A2(n1002), .B0(n921), .Y(n1028) );
INVX2 U204 ( .A(n993), .Y(n1021) );
NAND2X1 U205 ( .A(n309), .B(n1020), .Y(n993) );
XOR2XL U206 ( .A(n259_3), .B(S_B2), .Y(SUM_D_0) );
NAND3XL U207 ( .A(SUM_V_0), .B(n927), .C(n375), .Y(n1006) );
AOI31X1 U208 ( .A0(n996), .A1(n994), .A2(n997), .B0(n998), .Y(n995) );
NAND2X1 U209 ( .A(n918), .B(n954), .Y(n996) );
NAND4X1 U210 ( .A(SUM_D_0), .B(n1020), .C(n1000), .D(n993), .Y(n997) );
INVX2 U211 ( .A(n1007), .Y(n998) );
NAND2X1 U212 ( .A(n954), .B(n323), .Y(n1007) );
```

```
OAI21XL U213 ( .A0(SUB_TYP[1]), .A1(n1009), .B0(n986), .Y(n1013) );
OAI211XL U214 ( .A0(n1010), .A1(n1012), .B0(n1018), .C0(n1024), .Y(n1015)
        );
NAND2X1 U215 ( .A(n1011), .B(SUM_D_0), .Y(n1010) );
XOR2XL U216 ( .A(n265_3), .B(S_B1), .Y(SUM_HV_0) );
INVX2 U217 ( .A(n1010), .Y(n1023) );
NAND2X1 U218 ( .A(n962), .B(n963), .Y(n323) );
NOR2XL U219 ( .A(SUM_D_0), .B(SUM_D_1), .Y(n962) );
NAND2X1 U220 ( .A(n918), .B(n949), .Y(n1003) );
INVX2 U221 ( .A(SUM_D_0), .Y(n1001) );
NAND4X1 U222 ( .A(SUM_H_0), .B(n927), .C(n945), .D(n928), .Y(n994) );
NAND4X1 U223 ( .A(n1026), .B(n1018), .C(n1024), .D(n1008), .Y(n953) );
NAND2X1 U224 ( .A(n918), .B(n956), .Y(n1026) );
NAND4X1 U225 ( .A(n957), .B(n1012), .C(n552), .D(n551), .Y(n1018) );
INVX2 U226 ( .A(SUM_HV_0), .Y(n1012) );
NAND3XL U227 ( .A(SUM_D_0), .B(n956), .C(n1000), .Y(n1024) );
NAND3XL U228 ( .A(n552), .B(n551), .C(n1000), .Y(n1008) );
INVX2 U229 ( .A(n1026), .Y(n992) );
INVX2 U230 ( .A(n1024), .Y(n1025) );
INVX2 U231 ( .A(n1008), .Y(n1011) );
OAI211XL U232 ( .A0(n946), .A1(n947), .B0(n919), .C0(n960), .Y(n951) );
NAND3XL U233 ( .A(n927), .B(n928), .C(n917), .Y(n946) );
OR2X2 U234 ( .A(SUM_D_2), .B(SUM_D_1), .Y(n947) );
INVX2 U235 ( .A(n1005), .Y(n960) );
NAND4X1 U236 ( .A(n948), .B(n996), .C(n1006), .D(n1007), .Y(n1005) );
INVX2 U237 ( .A(n946), .Y(n1020) );
NAND3XL U238 ( .A(SUM_H_0), .B(n945), .C(n375), .Y(n1009) );
XOR2XL U239 ( .A(S_D1), .B(S_B1), .Y(SUM_H_0) );
INVX2 U240 ( .A(n922), .Y(n945) );
INVX2 U241 ( .A(acell_4820_U147_Z_1), .Y(n972) );
NAND3XL U242 ( .A(n1002), .B(n1014), .C(n921), .Y(n950) );
NAND3XL U243 ( .A(n917), .B(n927), .C(n1000), .Y(n1002) );
INVX2 U244 ( .A(n947), .Y(n1000) );
NAND2X1 U245 ( .A(n917), .B(n920), .Y(n1014) );
INVX2 U246 ( .A(SUM_V_0), .Y(n928) );
XOR2XL U247 ( .A(S_C2), .B(S_C0), .Y(SUM_V_0) );
AOI222XL U248 ( .A0(S_B2), .A1(n935), .B0(S_D0), .B1(S_B0), .C0(S_D2),
      .C1(n936), .Y(n934) );
NAND2X1 U249 ( .A(n959), .B(n943), .Y(n935) );
```

```
INVX2 U250 ( .A(n936), .Y(n959) );
INVX2 U251 ( .A(S_D2), .Y(n943) );
NAND2X1 U252 ( .A(n941), .B(n942), .Y(n936) );
INVX2 U253 ( .A(S_D0), .Y(n941) );
INVX2 U254 ( .A(S_B0), .Y(n942) );
AOI222XL U255 ( .A0(S_C2), .A1(n932), .B0(S_C0), .B1(S_D1), .C0(S_B1),
      .C1(n933), .Y(n931) );
NAND2X1 U256 ( .A(n958), .B(n940), .Y(n932) );
INVX2 U257 ( .A(n933), .Y(n958) );
INVX2 U258 ( .A(S_B1), .Y(n940) );
NAND2X1 U259 ( .A(n938), .B(n939), .Y(n933) );
INVX2 U260 ( .A(S_C0), .Y(n938) );
INVX2 U261 ( .A(S_D1), .Y(n939) );
OAI21XL U262 ( .A0(SUB_TYP[1]), .A1(n974), .B0(n975), .Y(CXD299_3) );
NAND2X1 U263 ( .A(n922), .B(n944), .Y(n974) );
INVX2 U264 ( .A(SUM_H_0), .Y(n944) );
OAI221X1 U265 ( .A0(SUB_TYP[1]), .A1(n979), .B0(SUB_TYP[1]), .B1(n980),
      .C0(n981), .Y(CXD299_1) );
NAND3XL U266 ( .A(SUM_V_0), .B(n927), .C(n917), .Y(n979) );
INVX2 U267 ( .A(n923), .Y(n927) );
INVX2 U268 ( .A(n1004), .Y(n980) );
NAND2X1 U269 ( .A(n977), .B(n993), .Y(n1004) );
AOI221X1 U270 ( .A0(n973), .A1(n1016), .B0(n553), .B1(n1017), .C0(n1013),
      .Y(n981) );
INVX2 U271 ( .A(n979), .Y(n954) );
OAI211XL U272 ( .A0(n982), .A1(n983), .B0(n984), .C0(n985), .Y(CXD299_0)
       );
NOR2XL U273 ( .A(n995), .B(n999), .Y(n982) );
INVX2 U274 ( .A(n1006), .Y(n999) );
NAND2X1 U275 ( .A(SUB_TYP[1]), .B(n937), .Y(n983) );
INVX2 U276 ( .A(SUB_TYP[0]), .Y(n937) );
NAND3XL U277 ( .A(n987), .B(n977), .C(n1028), .Y(n984) );
AOI31X1 U278 ( .A0(n553), .A1(n1018), .A2(n1019), .B0(n1013), .Y(n985) );
OAI221X1 U279 ( .A0(SUB_TYP[1]), .A1(n976), .B0(SUB_TYP[1]), .B1(n977),
      .C0(n978), .Y(CXD299_2) );
NOR2XL U280 ( .A(n988), .B(n989), .Y(n976) );
INVX2 U281 ( .A(n1014), .Y(n988) );
INVX2 U282 ( .A(n1003), .Y(n989) );
NAND2X1 U283 ( .A(n323), .B(n949), .Y(n977) );
```

```
INVX2 U284 ( .A(n994), .Y(n949) );
AOI221X1 U285 ( .A0(n973), .A1(n1005), .B0(n553), .B1(n1015), .C0(n1013),
      .Y(n978) );
NAND4X1 U286 ( .A(n986), .B(n975), .C(n929), .D(n930), .Y(n300_4) );
NAND3XL U287 ( .A(n553), .B(n956), .C(n957), .Y(n986) );
INVX2 U288 ( .A(n931), .Y(n956) );
INVX2 U289 ( .A(n934), .Y(n957) );
NAND2X1 U290 ( .A(n973), .B(n920), .Y(n975) );
INVX2 U291 ( .A(n983), .Y(n973) );
OAI31X1 U292 ( .A0(n1004), .A1(n955), .A2(n950), .B0(n987), .Y(n929) );
INVX2 U293 ( .A(n1009), .Y(n955) );
INVX2 U294 ( .A(SUB_TYP[1]), .Y(n987) );
AOI222XL U295 ( .A0(n973), .A1(n951), .B0(n952), .B1(n987), .C0(n553),
      .C1(n953), .Y(n930) );
INVX2 U296 ( .A(n974), .Y(n952) );
TLATX1 CXD_reg_3_ ( .D(CXD299_3), .G(n300_4), .Q(CXD[3]) );
TLATX1 CXD_reg_1_ ( .D(CXD299_1), .G(n300_4), .Q(CXD[1]) );
TLATX1 CXD_reg_0_ ( .D(CXD299_0), .G(n300_4), .Q(CXD[0]) );
TLATX1 CXD_reg_2_ ( .D(CXD299_2), .G(n300_4), .Q(CXD[2]) );
OR2X2 U297 ( .A(add_63_3_g_array_1_0), .B(n925), .Y(n551) );
XOR2XL U298 ( .A(n265_2), .B(n925), .Y(n552) );
AND2X2 U299 ( .A(SUB_TYP[0]), .B(SUB_TYP[1]), .Y(n553) );
AND2X2 U300 ( .A(n945), .B(n944), .Y(n917) );
AND2X2 U301 ( .A(n1000), .B(n1001), .Y(n918) );
AND2X2 U302 ( .A(n993), .B(n994), .Y(n919) );
AND2X2 U303 ( .A(n923), .B(n928), .Y(n920) );
AND2X2 U304 ( .A(n1003), .B(n979), .Y(n921) );
INVX2 U305 ( .A(SUM_D_2), .Y(n963) );
INVX2 U306 ( .A(add_63_3_g_array_1_0), .Y(n265_2) );
AOI21X1 add_63_3_U1_4_0_0 ( .A0(add_63_3_pog_array_0_0), .A1(S_C0), .B0(
      add_63_3_g_array_0_0), .Y(add_63_3_g_array_1_0) );
INVX2 U307 ( .A(add_61_3_g_array_1_0), .Y(n259_2) );
AOI21X1 add_61_3_U1_4_0_0 ( .A0(add_61_3_pog_array_0_0), .A1(S_B0), .B0(
      add_61_3_g_array_0_0), .Y(add_61_3_g_array_1_0) );
AND2X2 U308 ( .A(S_D1), .B(S_B1), .Y(n922) );
AND2X2 U309 ( .A(S_C2), .B(S_C0), .Y(n923) );
NAND2X1 U310 ( .A(n259_3), .B(S_B2), .Y(n924) );
OAI2BB2X1 U311 ( .A0N(SUM_H_0), .A1N(SUB_TYP[1]), .B0(n973), .B1(n928),
      .Y(acell_4820_U147_Z_0) );
```

```
    OAI2BB2X1 U312 ( .A0N(n922), .A1N(SUB_TYP[1]), .B0(n973), .B1(n927), .Y(
        acell_4820_U147_Z_1) );
    NAND2X1 U313 ( .A(n265_3), .B(S_B1), .Y(n925) );
    NAND2BX1 U314 ( .AN(acell_4820_U147_Z_0), .B(n972), .Y(n375) );
    NOR2XL U315 ( .A(add_61_3_g_array_1_0), .B(n924), .Y(SUM_D_2) );
    TLATX1 CXD_reg_4_ ( .D(1'b0), .G(n300_4), .Q(CXD[4]) );
endmodule
module ZC_P13_2 ( CXD, S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2,
    SUB_TYP );
output [5:0] CXD;
input  [1:0] SUB_TYP;
input  S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2;
    wire n261_3, SUM_H_0, n325, n377, n267_2, n302_4, n311, CXD301_3, SUM_HV_0,
        SUM_D_1, CXD301_1, V0_Cn_wire, SUM_D_0, CXD301_0, SUM_D_2, SUM_V_0,
        CXD301_2, n267_3, n261_2, n780, n781, n782, n783, n784, n785, n786,
        n787, n788, n789, n790, acell_4536_U43_Z_0, acell_4536_U43_Z_1, n792,
        n793, n794, n795, n796, n797, n798, n799, n800, n801, n802, n803, n804,
        n805, n806, n807, n808, n809, n810, n811, n812, n813, n814, n815, n816,
        n817, n818, n819, n820, n821, n822, n823, n824, n825, net9887, net9888,
        net9889, net9978, net9979, net9980, n850, n851, n852, n853, n854, n855,
        n856, n857, n858, n859, n860, n861, n862, n863, n864, n865, n866, n867,
        n868, n869, n870, n871, n872, n873, n874, n875, n876, n877, n878, n879,
        n880, n881, n882, n883, n884, n885, n886, n887, n888, n889, n890, n891,
        n892, n893, n894, n895, n896, n897, n898, n899, n900, n901, n902, n903,
        n904, n905, n906, n907, n908, n909, n910, n911, n912, n913, n914, n915,
        n916;
    assign CXD[5] = V0_Cn_wire;
    assign V0_Cn_wire = V0_Cn;
    NAND2X1 U180 ( .A(S_B0), .B(S_D2), .Y(n854) );
    OR2X2 U181 ( .A(S_D2), .B(S_B0), .Y(net9979) );
    NAND2X1 U182 ( .A(n782), .B(n879), .Y(n914) );
    OR2X2 U183 ( .A(S_D1), .B(S_C2), .Y(net9888) );
    INVX2 U184 ( .A(n858), .Y(net9889) );
    NAND2X1 U185 ( .A(S_C2), .B(S_D1), .Y(n858) );
    MXI2XL U186 ( .S0(S_C0), .B(n857), .A(n860), .Y(n267_3) );
    XNOR2XL U187 ( .A(S_C2), .B(S_D1), .Y(n860) );
    NOR2BXL U188 ( .AN(n858), .B(n859), .Y(n857) );
    INVX2 U189 ( .A(net9888), .Y(n859) );
    NAND2X1 U190 ( .A(n852), .B(n850), .Y(n311) );
```

MXI2XL U191 (.S0(S_D0), .B(n853), .A(n856), .Y(n261_3));

XNOR2XL U192 (.A(S_B0), .B(S_D2), .Y(n856));

NOR2BXL U193 (.AN(n854), .B(n855), .Y(n853));

INVX2 U194 (.A(net9979), .Y(n855));

NAND3XL U195 (.A(n792), .B(n793), .C(n822), .Y(n816));

NOR2XL U196 (.A(n904), .B(n819), .Y(n815));

NAND2X1 U197 (.A(n825), .B(n810), .Y(n801));

INVX2 U198 (.A(n802), .Y(n825));

NAND2X1 U199 (.A(n808), .B(n809), .Y(n802));

INVX2 U200 (.A(S_C0), .Y(n808));

INVX2 U201 (.A(S_D1), .Y(n809));

XNOR2XL U202 (.A(n261_2), .B(n789), .Y(SUM_D_1));

INVX2 U203 (.A(n854), .Y(net9980));

INVX2 U204 (.A(SUM_D_1), .Y(n850));

OAI21XL U205 (.A0(n904), .A1(n886), .B0(n908), .Y(n907));

NOR2XL U206 (.A(n887), .B(n888), .Y(n886));

INVX2 U207 (.A(n914), .Y(n887));

NAND3XL U208 (.A(n900), .B(n899), .C(n912), .Y(n906));

NAND3XL U209 (.A(n911), .B(n913), .C(n915), .Y(n909));

NAND3XL U210 (.A(SUM_D_0), .B(n897), .C(n881), .Y(n911));

NAND3XL U211 (.A(SUM_D_0), .B(n879), .C(n782), .Y(n913));

NAND4X1 U212 (.A(SUM_HV_0), .B(n881), .C(n914), .D(n898), .Y(n915));

INVX2 U213 (.A(n911), .Y(n888));

INVX2 U214 (.A(n913), .Y(n904));

NAND4X1 U215 (.A(n823), .B(n897), .C(n781), .D(n780), .Y(n908));

XOR2XL U216 (.A(n267_3), .B(S_B1), .Y(SUM_HV_0));

INVX2 U217 (.A(n908), .Y(n819));

OAI31X1 U218 (.A0(n910), .A1(n898), .A2(n894), .B0(n786), .Y(n916));

INVX2 U219 (.A(n884), .Y(n910));

NAND2X1 U220 (.A(n311), .B(n785), .Y(n884));

NAND3XL U221 (.A(SUM_V_0), .B(n792), .C(n377), .Y(n899));

AOI31X1 U222 (.A0(n890), .A1(n885), .A2(n891), .B0(n892), .Y(n889));

NAND3XL U223 (.A(n820), .B(n898), .C(n782), .Y(n890));

NAND4X1 U224 (.A(SUM_D_0), .B(n785), .C(n782), .D(n884), .Y(n891));

INVX2 U225 (.A(n900), .Y(n892));

NAND2X1 U226 (.A(n820), .B(n325), .Y(n900));

OAI21XL U227 (.A0(SUB_TYP[1]), .A1(n901), .B0(n875), .Y(n902));

NAND2X1 U228 (.A(n851), .B(n852), .Y(n325));

NOR2XL U229 (.A(SUM_D_0), .B(SUM_D_1), .Y(n851));

```
NAND3XL U230 ( .A(n817), .B(n898), .C(n782), .Y(n895) );
NAND4X1 U231 ( .A(SUM_H_0), .B(n792), .C(n813), .D(n793), .Y(n885) );
XOR2XL U232 ( .A(n261_3), .B(S_B2), .Y(SUM_D_0) );
AOI222XL U233 ( .A0(S_B2), .A1(n798), .B0(S_D0), .B1(S_B0), .C0(S_D2),
      .C1(n799), .Y(n797) );
NAND2X1 U234 ( .A(n824), .B(n807), .Y(n798) );
INVX2 U235 ( .A(n799), .Y(n824) );
INVX2 U236 ( .A(S_D2), .Y(n807) );
NAND2X1 U237 ( .A(n805), .B(n806), .Y(n799) );
INVX2 U238 ( .A(S_D0), .Y(n805) );
NAND3XL U239 ( .A(SUM_H_0), .B(n813), .C(n377), .Y(n901) );
XOR2XL U240 ( .A(S_D1), .B(S_B1), .Y(SUM_H_0) );
INVX2 U241 ( .A(n787), .Y(n813) );
INVX2 U242 ( .A(acell_4536_U43_Z_1), .Y(n861) );
NAND3XL U243 ( .A(n894), .B(n903), .C(n786), .Y(n818) );
NAND3XL U244 ( .A(n783), .B(n792), .C(n782), .Y(n894) );
NAND2X1 U245 ( .A(n783), .B(n784), .Y(n903) );
XOR2XL U246 ( .A(S_C2), .B(S_C0), .Y(SUM_V_0) );
AOI211X1 U247 ( .A0(n782), .A1(n785), .B0(n877), .C0(n878), .Y(n796) );
INVX2 U248 ( .A(SUM_V_0), .Y(n793) );
NAND4X1 U249 ( .A(n890), .B(n816), .C(n899), .D(n900), .Y(n877) );
NAND2X1 U250 ( .A(n884), .B(n885), .Y(n878) );
INVX2 U251 ( .A(n878), .Y(n912) );
NAND2X1 U252 ( .A(SUB_TYP[0]), .B(SUB_TYP[1]), .Y(n811) );
AOI211X1 U253 ( .A0(n782), .A1(n879), .B0(n880), .C0(n881), .Y(n803) );
AOI222XL U254 ( .A0(S_C2), .A1(n801), .B0(S_C0), .B1(S_D1), .C0(S_B1),
      .C1(n802), .Y(n800) );
OAI31X1 U255 ( .A0(n814), .A1(n898), .A2(n897), .B0(n815), .Y(n880) );
INVX2 U256 ( .A(SUM_D_0), .Y(n898) );
INVX2 U257 ( .A(SUM_HV_0), .Y(n897) );
INVX2 U258 ( .A(n814), .Y(n881) );
NAND3XL U259 ( .A(n781), .B(n780), .C(n782), .Y(n814) );
INVX2 U260 ( .A(S_B1), .Y(n810) );
INVX2 U261 ( .A(S_B0), .Y(n806) );
OAI21XL U262 ( .A0(SUB_TYP[1]), .A1(n863), .B0(n864), .Y(CXD301_3) );
NAND2X1 U263 ( .A(n787), .B(n812), .Y(n863) );
INVX2 U264 ( .A(SUM_H_0), .Y(n812) );
INVX2 U265 ( .A(n863), .Y(n822) );
OAI221X1 U266 ( .A0(SUB_TYP[1]), .A1(n868), .B0(SUB_TYP[1]), .B1(n869),
```

```
      .C0(n870), .Y(CXD301_1) );
NAND3XL U267 ( .A(SUM_V_0), .B(n792), .C(n783), .Y(n868) );
INVX2 U268 ( .A(n788), .Y(n792) );
INVX2 U269 ( .A(n896), .Y(n869) );
NAND2X1 U270 ( .A(n866), .B(n884), .Y(n896) );
AOI221X1 U271 ( .A0(n862), .A1(n906), .B0(n905), .B1(n907), .C0(n902), .Y(
      n870) );
INVX2 U272 ( .A(n868), .Y(n820) );
OAI211XL U273 ( .A0(n871), .A1(n872), .B0(n873), .C0(n874), .Y(CXD301_0)
       );
NOR2XL U274 ( .A(n889), .B(n893), .Y(n871) );
INVX2 U275 ( .A(n899), .Y(n893) );
NAND2X1 U276 ( .A(SUB_TYP[1]), .B(n804), .Y(n872) );
INVX2 U277 ( .A(SUB_TYP[0]), .Y(n804) );
NAND3XL U278 ( .A(n876), .B(n866), .C(n916), .Y(n873) );
AOI31X1 U279 ( .A0(n905), .A1(n908), .A2(n909), .B0(n902), .Y(n874) );
OAI221X1 U280 ( .A0(SUB_TYP[1]), .A1(n865), .B0(SUB_TYP[1]), .B1(n866),
      .C0(n867), .Y(CXD301_2) );
NOR2XL U281 ( .A(n882), .B(n883), .Y(n865) );
INVX2 U282 ( .A(n903), .Y(n882) );
INVX2 U283 ( .A(n895), .Y(n883) );
NAND2X1 U284 ( .A(n325), .B(n817), .Y(n866) );
INVX2 U285 ( .A(n885), .Y(n817) );
AOI221X1 U286 ( .A0(n862), .A1(n877), .B0(n905), .B1(n880), .C0(n902), .Y(
      n867) );
NAND4BX1 U287 ( .AN(n794), .B(n864), .C(n795), .D(n875), .Y(n302_4) );
OAI222X1 U288 ( .A0(n803), .A1(n811), .B0(SUB_TYP[1]), .B1(n863), .C0(n796
      ), .C1(n872), .Y(n794) );
NAND2X1 U289 ( .A(n862), .B(n784), .Y(n864) );
INVX2 U290 ( .A(n872), .Y(n862) );
OAI31X1 U291 ( .A0(n896), .A1(n821), .A2(n818), .B0(n876), .Y(n795) );
INVX2 U292 ( .A(n901), .Y(n821) );
INVX2 U293 ( .A(SUB_TYP[1]), .Y(n876) );
NAND3XL U294 ( .A(n879), .B(n905), .C(n823), .Y(n875) );
INVX2 U295 ( .A(n800), .Y(n879) );
INVX2 U296 ( .A(n811), .Y(n905) );
INVX2 U297 ( .A(n797), .Y(n823) );
TLATX1 CXD_reg_3_ ( .D(CXD301_3), .G(n302_4), .Q(CXD[3]) );
TLATX1 CXD_reg_1_ ( .D(CXD301_1), .G(n302_4), .Q(CXD[1]) );
```

```
    TLATX1 CXD_reg_0_ ( .D(CXD301_0), .G(n302_4), .Q(CXD[0]) );
    TLATX1 CXD_reg_2_ ( .D(CXD301_2), .G(n302_4), .Q(CXD[2]) );
    OR2X2 U298 ( .A(net9887), .B(n790), .Y(n780) );
    XOR2XL U299 ( .A(n267_2), .B(n790), .Y(n781) );
    NOR2X1 U300 ( .A(SUM_D_2), .B(SUM_D_1), .Y(n782) );
    AND2X2 U301 ( .A(n813), .B(n812), .Y(n783) );
    AND2X2 U302 ( .A(n788), .B(n793), .Y(n784) );
    AND3X1 U303 ( .A(n792), .B(n793), .C(n783), .Y(n785) );
    AND2X2 U304 ( .A(n895), .B(n868), .Y(n786) );
    INVX2 U305 ( .A(SUM_D_2), .Y(n852) );
    INVX2 U306 ( .A(net9887), .Y(n267_2) );
    AOI21X1 U307 ( .A0(net9888), .A1(S_C0), .B0(net9889), .Y(net9887) );
    INVX2 U308 ( .A(net9978), .Y(n261_2) );
    AOI21X1 U309 ( .A0(net9979), .A1(S_D0), .B0(net9980), .Y(net9978) );
    AND2X2 U310 ( .A(S_D1), .B(S_B1), .Y(n787) );
    AND2X2 U311 ( .A(S_C2), .B(S_C0), .Y(n788) );
    NAND2X1 U312 ( .A(n261_3), .B(S_B2), .Y(n789) );
    NAND2X1 U313 ( .A(n267_3), .B(S_B1), .Y(n790) );
    OAI2BB2X1 U314 ( .A0N(SUM_H_0), .A1N(SUB_TYP[1]), .B0(n862), .B1(n793),
        .Y(acell_4536_U43_Z_0) );
    OAI2BB2X1 U315 ( .A0N(n787), .A1N(SUB_TYP[1]), .B0(n862), .B1(n792), .Y(
        acell_4536_U43_Z_1) );
    NAND2BX1 U316 ( .AN(acell_4536_U43_Z_0), .B(n861), .Y(n377) );
    NOR2XL U317 ( .A(net9978), .B(n789), .Y(SUM_D_2) );
    TLATX1 CXD_reg_4_ ( .D(1'b0), .G(n302_4), .Q(CXD[4]) );
endmodule
module ZC_P13_3 ( CXD, S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2,
    SUB_TYP );
output [5:0] CXD;
input  [1:0] SUB_TYP;
input  S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2;
    wire n261_3, SUM_H_0, n325, n377, n267_2, n302_4, n311, CXD301_3, SUM_HV_0,
        SUM_D_1, CXD301_1, V0_Cn_wire, SUM_D_0, CXD301_0, SUM_D_2, SUM_V_0,
        CXD301_2, n267_3, n261_2, n667, n668, n669, n670, n671, n672, n673,
        n674, n675, n676, n677, acell_4252_U43_Z_0, acell_4252_U43_Z_1, n679,
        n680, n681, n682, n683, n684, n685, n686, n687, n688, n689, n690, n691,
        n692, n693, n694, n695, n696, n697, n698, n699, n700, n701, n702, n703,
        n704, n705, n706, n707, n708, n709, n710, n711, n712, net9523, net9524,
        net9525, net9614, net9615, net9616, n713, n714, n715, n716, n717, n718,
```

```
      n719, n720, n721, n722, n723, n724, n725, n726, n727, n728, n729, n730,
      n731, n732, n733, n734, n735, n736, n737, n738, n739, n740, n741, n742,
      n743, n744, n745, n746, n747, n748, n749, n750, n751, n752, n753, n754,
      n755, n756, n757, n758, n759, n760, n761, n762, n763, n764, n765, n766,
      n767, n768, n769, n770, n771, n772, n773, n774, n775, n776, n777, n778,
      n779;
assign CXD[5] = V0_Cn_wire;
assign V0_Cn_wire = V0_Cn;
NAND2X1 U180 ( .A(S_B2), .B(S_D0), .Y(n717) );
OR2X2 U181 ( .A(S_D0), .B(S_B2), .Y(net9615) );
NAND2X1 U182 ( .A(n669), .B(n742), .Y(n777) );
OR2X2 U183 ( .A(S_D1), .B(S_C0), .Y(net9524) );
INVX2 U184 ( .A(n721), .Y(net9525) );
NAND2X1 U185 ( .A(S_C0), .B(S_D1), .Y(n721) );
MXI2XL U186 ( .S0(S_C2), .B(n720), .A(n723), .Y(n267_3) );
XNOR2XL U187 ( .A(S_C0), .B(S_D1), .Y(n723) );
NOR2BXL U188 ( .AN(n721), .B(n722), .Y(n720) );
INVX2 U189 ( .A(net9524), .Y(n722) );
NAND2X1 U190 ( .A(n715), .B(n713), .Y(n311) );
MXI2XL U191 ( .S0(S_D2), .B(n716), .A(n719), .Y(n261_3) );
XNOR2XL U192 ( .A(S_B2), .B(S_D0), .Y(n719) );
NOR2BXL U193 ( .AN(n717), .B(n718), .Y(n716) );
INVX2 U194 ( .A(net9615), .Y(n718) );
NAND3XL U195 ( .A(n679), .B(n680), .C(n709), .Y(n703) );
NOR2XL U196 ( .A(n767), .B(n706), .Y(n702) );
NAND2X1 U197 ( .A(n712), .B(n697), .Y(n688) );
INVX2 U198 ( .A(n689), .Y(n712) );
NAND2X1 U199 ( .A(n695), .B(n696), .Y(n689) );
INVX2 U200 ( .A(S_D1), .Y(n696) );
XNOR2XL U201 ( .A(n261_2), .B(n676), .Y(SUM_D_1) );
INVX2 U202 ( .A(n717), .Y(net9616) );
INVX2 U203 ( .A(SUM_D_1), .Y(n713) );
OAI21XL U204 ( .A0(n767), .A1(n749), .B0(n771), .Y(n770) );
NOR2XL U205 ( .A(n750), .B(n751), .Y(n749) );
INVX2 U206 ( .A(n777), .Y(n750) );
NAND3XL U207 ( .A(n763), .B(n762), .C(n775), .Y(n769) );
NAND3XL U208 ( .A(n774), .B(n776), .C(n778), .Y(n772) );
NAND3XL U209 ( .A(SUM_D_0), .B(n760), .C(n744), .Y(n774) );
NAND3XL U210 ( .A(SUM_D_0), .B(n742), .C(n669), .Y(n776) );
```

```
NAND4X1 U211 ( .A(SUM_HV_0), .B(n744), .C(n777), .D(n761), .Y(n778) );
INVX2 U212 ( .A(n774), .Y(n751) );
INVX2 U213 ( .A(n776), .Y(n767) );
NAND4X1 U214 ( .A(n710), .B(n760), .C(n668), .D(n667), .Y(n771) );
XOR2XL U215 ( .A(n267_3), .B(S_B1), .Y(SUM_HV_0) );
INVX2 U216 ( .A(n771), .Y(n706) );
OAI31X1 U217 ( .A0(n773), .A1(n761), .A2(n757), .B0(n673), .Y(n779) );
INVX2 U218 ( .A(n747), .Y(n773) );
NAND2X1 U219 ( .A(n311), .B(n672), .Y(n747) );
NAND3XL U220 ( .A(SUM_V_0), .B(n679), .C(n377), .Y(n762) );
AOI31X1 U221 ( .A0(n753), .A1(n748), .A2(n754), .B0(n755), .Y(n752) );
NAND3XL U222 ( .A(n707), .B(n761), .C(n669), .Y(n753) );
NAND4X1 U223 ( .A(SUM_D_0), .B(n672), .C(n669), .D(n747), .Y(n754) );
INVX2 U224 ( .A(n763), .Y(n755) );
NAND2X1 U225 ( .A(n707), .B(n325), .Y(n763) );
OAI21XL U226 ( .A0(SUB_TYP[1]), .A1(n764), .B0(n738), .Y(n765) );
NAND2X1 U227 ( .A(n714), .B(n715), .Y(n325) );
NOR2XL U228 ( .A(SUM_D_0), .B(SUM_D_1), .Y(n714) );
NAND3XL U229 ( .A(n704), .B(n761), .C(n669), .Y(n758) );
NAND4X1 U230 ( .A(SUM_H_0), .B(n679), .C(n700), .D(n680), .Y(n748) );
XOR2XL U231 ( .A(n261_3), .B(S_B0), .Y(SUM_D_0) );
AOI222XL U232 ( .A0(S_B2), .A1(n685), .B0(S_D0), .B1(S_B0), .C0(S_D2),
      .C1(n686), .Y(n684) );
NAND2X1 U233 ( .A(n711), .B(n694), .Y(n685) );
INVX2 U234 ( .A(n686), .Y(n711) );
INVX2 U235 ( .A(S_D2), .Y(n694) );
NAND2X1 U236 ( .A(n692), .B(n693), .Y(n686) );
INVX2 U237 ( .A(S_D0), .Y(n692) );
NAND3XL U238 ( .A(SUM_H_0), .B(n700), .C(n377), .Y(n764) );
XOR2XL U239 ( .A(S_D1), .B(S_B1), .Y(SUM_H_0) );
INVX2 U240 ( .A(n674), .Y(n700) );
INVX2 U241 ( .A(acell_4252_U43_Z_1), .Y(n724) );
NAND3XL U242 ( .A(n757), .B(n766), .C(n673), .Y(n705) );
NAND3XL U243 ( .A(n670), .B(n679), .C(n669), .Y(n757) );
NAND2X1 U244 ( .A(n670), .B(n671), .Y(n766) );
XOR2XL U245 ( .A(S_C2), .B(S_C0), .Y(SUM_V_0) );
AOI211X1 U246 ( .A0(n669), .A1(n672), .B0(n740), .C0(n741), .Y(n683) );
INVX2 U247 ( .A(SUM_V_0), .Y(n680) );
NAND4X1 U248 ( .A(n753), .B(n703), .C(n762), .D(n763), .Y(n740) );
```

```
NAND2X1 U249 ( .A(n747), .B(n748), .Y(n741) );
INVX2 U250 ( .A(n741), .Y(n775) );
NAND2X1 U251 ( .A(SUB_TYP[0]), .B(SUB_TYP[1]), .Y(n698) );
AOI211X1 U252 ( .A0(n669), .A1(n742), .B0(n743), .C0(n744), .Y(n690) );
AOI222XL U253 ( .A0(S_C2), .A1(n688), .B0(S_C0), .B1(S_D1), .C0(S_B1),
      .C1(n689), .Y(n687) );
OAI31X1 U254 ( .A0(n701), .A1(n761), .A2(n760), .B0(n702), .Y(n743) );
INVX2 U255 ( .A(SUM_D_0), .Y(n761) );
INVX2 U256 ( .A(SUM_HV_0), .Y(n760) );
INVX2 U257 ( .A(n701), .Y(n744) );
NAND3XL U258 ( .A(n668), .B(n667), .C(n669), .Y(n701) );
INVX2 U259 ( .A(S_B1), .Y(n697) );
INVX2 U260 ( .A(S_B0), .Y(n693) );
INVX2 U261 ( .A(S_C0), .Y(n695) );
OAI21XL U262 ( .A0(SUB_TYP[1]), .A1(n726), .B0(n727), .Y(CXD301_3) );
NAND2X1 U263 ( .A(n674), .B(n699), .Y(n726) );
INVX2 U264 ( .A(SUM_H_0), .Y(n699) );
INVX2 U265 ( .A(n726), .Y(n709) );
OAI221X1 U266 ( .A0(SUB_TYP[1]), .A1(n731), .B0(SUB_TYP[1]), .B1(n732),
      .C0(n733), .Y(CXD301_1) );
NAND3XL U267 ( .A(SUM_V_0), .B(n679), .C(n670), .Y(n731) );
INVX2 U268 ( .A(n675), .Y(n679) );
INVX2 U269 ( .A(n759), .Y(n732) );
NAND2X1 U270 ( .A(n729), .B(n747), .Y(n759) );
AOI221X1 U271 ( .A0(n725), .A1(n769), .B0(n768), .B1(n770), .C0(n765), .Y(
      n733) );
INVX2 U272 ( .A(n731), .Y(n707) );
OAI211XL U273 ( .A0(n734), .A1(n735), .B0(n736), .C0(n737), .Y(CXD301_0)
       );
NOR2XL U274 ( .A(n752), .B(n756), .Y(n734) );
INVX2 U275 ( .A(n762), .Y(n756) );
NAND2X1 U276 ( .A(SUB_TYP[1]), .B(n691), .Y(n735) );
INVX2 U277 ( .A(SUB_TYP[0]), .Y(n691) );
NAND3XL U278 ( .A(n739), .B(n729), .C(n779), .Y(n736) );
AOI31X1 U279 ( .A0(n768), .A1(n771), .A2(n772), .B0(n765), .Y(n737) );
OAI221X1 U280 ( .A0(SUB_TYP[1]), .A1(n728), .B0(SUB_TYP[1]), .B1(n729),
      .C0(n730), .Y(CXD301_2) );
NOR2XL U281 ( .A(n745), .B(n746), .Y(n728) );
INVX2 U282 ( .A(n766), .Y(n745) );
```

```
INVX2 U283 ( .A(n758), .Y(n746) );
NAND2X1 U284 ( .A(n325), .B(n704), .Y(n729) );
INVX2 U285 ( .A(n748), .Y(n704) );
AOI221X1 U286 ( .A0(n725), .A1(n740), .B0(n768), .B1(n743), .C0(n765), .Y(
    n730) );
NAND4BX1 U287 ( .AN(n681), .B(n727), .C(n682), .D(n738), .Y(n302_4) );
OAI222X1 U288 ( .A0(n690), .A1(n698), .B0(SUB_TYP[1]), .B1(n726), .C0(n683
    ), .C1(n735), .Y(n681) );
NAND2X1 U289 ( .A(n725), .B(n671), .Y(n727) );
INVX2 U290 ( .A(n735), .Y(n725) );
OAI31X1 U291 ( .A0(n759), .A1(n708), .A2(n705), .B0(n739), .Y(n682) );
INVX2 U292 ( .A(n764), .Y(n708) );
INVX2 U293 ( .A(SUB_TYP[1]), .Y(n739) );
NAND3XL U294 ( .A(n768), .B(n710), .C(n742), .Y(n738) );
INVX2 U295 ( .A(n698), .Y(n768) );
INVX2 U296 ( .A(n684), .Y(n710) );
INVX2 U297 ( .A(n687), .Y(n742) );
TLATX1 CXD_reg_3_ ( .D(CXD301_3), .G(n302_4), .Q(CXD[3]) );
TLATX1 CXD_reg_1_ ( .D(CXD301_1), .G(n302_4), .Q(CXD[1]) );
TLATX1 CXD_reg_0_ ( .D(CXD301_0), .G(n302_4), .Q(CXD[0]) );
TLATX1 CXD_reg_2_ ( .D(CXD301_2), .G(n302_4), .Q(CXD[2]) );
OR2X2 U298 ( .A(net9523), .B(n677), .Y(n667) );
XOR2XL U299 ( .A(n267_2), .B(n677), .Y(n668) );
NOR2X1 U300 ( .A(SUM_D_2), .B(SUM_D_1), .Y(n669) );
AND2X2 U301 ( .A(n700), .B(n699), .Y(n670) );
AND2X2 U302 ( .A(n675), .B(n680), .Y(n671) );
AND3X1 U303 ( .A(n679), .B(n680), .C(n670), .Y(n672) );
AND2X2 U304 ( .A(n758), .B(n731), .Y(n673) );
INVX2 U305 ( .A(SUM_D_2), .Y(n715) );
INVX2 U306 ( .A(net9523), .Y(n267_2) );
AOI21X1 U307 ( .A0(net9524), .A1(S_C2), .B0(net9525), .Y(net9523) );
INVX2 U308 ( .A(net9614), .Y(n261_2) );
AOI21X1 U309 ( .A0(net9615), .A1(S_D2), .B0(net9616), .Y(net9614) );
AND2X2 U310 ( .A(S_D1), .B(S_B1), .Y(n674) );
AND2X2 U311 ( .A(S_C2), .B(S_C0), .Y(n675) );
NAND2X1 U312 ( .A(n261_3), .B(S_B0), .Y(n676) );
NAND2X1 U313 ( .A(n267_3), .B(S_B1), .Y(n677) );
OAI2BB2X1 U314 ( .A0N(SUM_H_0), .A1N(SUM_TYP[1]), .B0(n725), .B1(n680),
    .Y(acell_4252_U43_Z_0) );
```

```
    OAI2BB2X1 U315 ( .A0N(n674), .A1N(SUB_TYP[1]), .B0(n725), .B1(n679), .Y(
        acell_4252_U43_Z_1) );
    NAND2BX1 U316 ( .AN(acell_4252_U43_Z_0), .B(n724), .Y(n377) );
    NOR2XL U317 ( .A(net9614), .B(n676), .Y(SUM_D_2) );
    TLATX1 CXD_reg_4_ ( .D(1'b0), .G(n302_4), .Q(CXD[4]) );
endmodule
module ZC_P13_4 ( CXD, S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2,
    SUB_TYP );
output [5:0] CXD;
input  [1:0] SUB_TYP;
input  S_B0, S_B1, S_B2, S_C0, V0_Cn, S_C2, S_D0, S_D1, S_D2;
    wire n261_3, SUM_H_0, n325, n377, n267_2, n302_4, n311, CXD301_3, SUM_HV_0,
        SUM_D_1, CXD301_1, V0_Cn_wire, SUM_D_0, CXD301_0, SUM_D_2, SUM_V_0,
        CXD301_2, n267_3, n261_2, n554, n555, n556, n557, n558, n559, n560,
        n561, n562, n563, n564, acell_3968_U43_Z_0, acell_3968_U43_Z_1, n566,
        n567, n568, n569, n570, n571, n572, n573, n574, n575, n576, n577, n578,
        n579, n580, n581, n582, n583, n584, n585, n586, n587, n588, n589, n590,
        n591, n592, n593, n594, n595, n596, n597, n598, n599, net9159, net9160,
        net9161, net9250, net9251, net9252, n600, n601, n602, n603, n604, n605,
        n606, n607, n608, n609, n610, n611, n612, n613, n614, n615, n616, n617,
        n618, n619, n620, n621, n622, n623, n624, n625, n626, n627, n628, n629,
        n630, n631, n632, n633, n634, n635, n636, n637, n638, n639, n640, n641,
        n642, n643, n644, n645, n646, n647, n648, n649, n650, n651, n652, n653,
        n654, n655, n656, n657, n658, n659, n660, n661, n662, n663, n664, n665,
        n666;
    assign CXD[5] = V0_Cn_wire;
    assign V0_Cn_wire = V0_Cn;
    NAND2X1 U180 ( .A(S_D0), .B(S_D2), .Y(n604) );
    OR2X2 U181 ( .A(S_D2), .B(S_D0), .Y(net9251) );
    NAND2X1 U182 ( .A(n556), .B(n629), .Y(n664) );
    OR2X2 U183 ( .A(S_D1), .B(S_C0), .Y(net9160) );
    INVX2 U184 ( .A(n608), .Y(net9161) );
    NAND2X1 U185 ( .A(S_C0), .B(S_D1), .Y(n608) );
    MXI2XL U186 ( .S0(S_C2), .B(n607), .A(n610), .Y(n267_3) );
    XNOR2XL U187 ( .A(S_C0), .B(S_D1), .Y(n610) );
    NOR2BXL U188 ( .AN(n608), .B(n609), .Y(n607) );
    INVX2 U189 ( .A(net9160), .Y(n609) );
    NAND2X1 U190 ( .A(n602), .B(n600), .Y(n311) );
    MXI2XL U191 ( .S0(S_B2), .B(n603), .A(n606), .Y(n261_3) );
```

XNOR2XL U192 (.A(S_D0), .B(S_D2), .Y(n606));

NOR2BXL U193 (.AN(n604), .B(n605), .Y(n603));

INVX2 U194 (.A(net9251), .Y(n605));

NAND3XL U195 (.A(n566), .B(n567), .C(n596), .Y(n590));

NOR2XL U196 (.A(n654), .B(n593), .Y(n589));

NAND2X1 U197 (.A(n599), .B(n584), .Y(n575));

INVX2 U198 (.A(n576), .Y(n599));

NAND2X1 U199 (.A(n582), .B(n583), .Y(n576));

INVX2 U200 (.A(S_D1), .Y(n583));

XNOR2XL U201 (.A(n261_2), .B(n563), .Y(SUM_D_1));

INVX2 U202 (.A(n604), .Y(net9252));

INVX2 U203 (.A(SUM_D_1), .Y(n600));

INVX2 U204 (.A(S_C0), .Y(n582));

OAI21XL U205 (.A0(n654), .A1(n636), .B0(n658), .Y(n657));

NOR2XL U206 (.A(n637), .B(n638), .Y(n636));

INVX2 U207 (.A(n664), .Y(n637));

NAND3XL U208 (.A(n650), .B(n649), .C(n662), .Y(n656));

NAND3XL U209 (.A(n661), .B(n663), .C(n665), .Y(n659));

NAND3XL U210 (.A(SUM_D_0), .B(n647), .C(n631), .Y(n661));

NAND3XL U211 (.A(SUM_D_0), .B(n629), .C(n556), .Y(n663));

NAND4X1 U212 (.A(SUM_HV_0), .B(n631), .C(n664), .D(n648), .Y(n665));

INVX2 U213 (.A(n661), .Y(n638));

INVX2 U214 (.A(n663), .Y(n654));

NAND4X1 U215 (.A(n597), .B(n647), .C(n555), .D(n554), .Y(n658));

XOR2XL U216 (.A(n267_3), .B(S_B1), .Y(SUM_HV_0));

INVX2 U217 (.A(n658), .Y(n593));

OAI31X1 U218 (.A0(n660), .A1(n648), .A2(n644), .B0(n560), .Y(n666));

INVX2 U219 (.A(n634), .Y(n660));

NAND2X1 U220 (.A(n311), .B(n559), .Y(n634));

NAND3XL U221 (.A(SUM_V_0), .B(n566), .C(n377), .Y(n649));

AOI31X1 U222 (.A0(n640), .A1(n635), .A2(n641), .B0(n642), .Y(n639));

NAND3XL U223 (.A(n556), .B(n648), .C(n594), .Y(n640));

NAND4X1 U224 (.A(SUM_D_0), .B(n556), .C(n559), .D(n634), .Y(n641));

INVX2 U225 (.A(n650), .Y(n642));

NAND2X1 U226 (.A(n594), .B(n325), .Y(n650));

OAI21XL U227 (.A0(SUB_TYP[1]), .A1(n651), .B0(n625), .Y(n652));

NAND2X1 U228 (.A(n601), .B(n602), .Y(n325));

NOR2XL U229 (.A(SUM_D_0), .B(SUM_D_1), .Y(n601));

NAND3XL U230 (.A(n556), .B(n648), .C(n591), .Y(n645));

```
XOR2XL U231 ( .A(S_B0), .B(n261_3), .Y(SUM_D_0) );
NAND4X1 U232 ( .A(SUM_H_0), .B(n566), .C(n587), .D(n567), .Y(n635) );
AOI222XL U233 ( .A0(S_B2), .A1(n572), .B0(S_D0), .B1(S_B0), .C0(S_D2),
      .C1(n573), .Y(n571) );
NAND2X1 U234 ( .A(n598), .B(n581), .Y(n572) );
INVX2 U235 ( .A(n573), .Y(n598) );
INVX2 U236 ( .A(S_D2), .Y(n581) );
NAND2X1 U237 ( .A(n579), .B(n580), .Y(n573) );
INVX2 U238 ( .A(S_D0), .Y(n579) );
NAND3XL U239 ( .A(SUM_H_0), .B(n587), .C(n377), .Y(n651) );
XOR2XL U240 ( .A(S_D1), .B(S_B1), .Y(SUM_H_0) );
INVX2 U241 ( .A(n561), .Y(n587) );
INVX2 U242 ( .A(acell_3968_U43_Z_1), .Y(n611) );
NAND3XL U243 ( .A(n644), .B(n653), .C(n560), .Y(n592) );
NAND3XL U244 ( .A(n557), .B(n566), .C(n556), .Y(n644) );
NAND2X1 U245 ( .A(n557), .B(n558), .Y(n653) );
XOR2XL U246 ( .A(S_C2), .B(S_C0), .Y(SUM_V_0) );
AOI211X1 U247 ( .A0(n556), .A1(n559), .B0(n627), .C0(n628), .Y(n570) );
INVX2 U248 ( .A(SUM_V_0), .Y(n567) );
NAND4X1 U249 ( .A(n640), .B(n590), .C(n649), .D(n650), .Y(n627) );
NAND2X1 U250 ( .A(n634), .B(n635), .Y(n628) );
INVX2 U251 ( .A(n628), .Y(n662) );
NAND2X1 U252 ( .A(SUB_TYP[0]), .B(SUB_TYP[1]), .Y(n585) );
AOI211X1 U253 ( .A0(n556), .A1(n629), .B0(n630), .C0(n631), .Y(n577) );
AOI222XL U254 ( .A0(S_C2), .A1(n575), .B0(S_C0), .B1(S_D1), .C0(S_B1),
      .C1(n576), .Y(n574) );
OAI31X1 U255 ( .A0(n588), .A1(n648), .A2(n647), .B0(n589), .Y(n630) );
INVX2 U256 ( .A(SUM_D_0), .Y(n648) );
INVX2 U257 ( .A(SUM_HV_0), .Y(n647) );
INVX2 U258 ( .A(n588), .Y(n631) );
NAND3XL U259 ( .A(n555), .B(n554), .C(n556), .Y(n588) );
INVX2 U260 ( .A(S_B1), .Y(n584) );
INVX2 U261 ( .A(S_B0), .Y(n580) );
OAI21XL U262 ( .A0(SUB_TYP[1]), .A1(n613), .B0(n614), .Y(CXD301_3) );
NAND2X1 U263 ( .A(n561), .B(n586), .Y(n613) );
INVX2 U264 ( .A(SUM_H_0), .Y(n586) );
INVX2 U265 ( .A(n613), .Y(n596) );
OAI221X1 U266 ( .A0(SUB_TYP[1]), .A1(n618), .B0(SUB_TYP[1]), .B1(n619),
      .C0(n620), .Y(CXD301_1) );
```

```
NAND3XL U267 ( .A(SUM_V_0), .B(n566), .C(n557), .Y(n618) );
INVX2 U268 ( .A(n562), .Y(n566) );
INVX2 U269 ( .A(n646), .Y(n619) );
NAND2X1 U270 ( .A(n616), .B(n634), .Y(n646) );
AOI221X1 U271 ( .A0(n612), .A1(n656), .B0(n655), .B1(n657), .C0(n652), .Y(
      n620) );
INVX2 U272 ( .A(n618), .Y(n594) );
OAI211XL U273 ( .A0(n621), .A1(n622), .B0(n623), .C0(n624), .Y(CXD301_0)
       );
NOR2XL U274 ( .A(n639), .B(n643), .Y(n621) );
INVX2 U275 ( .A(n649), .Y(n643) );
NAND2X1 U276 ( .A(SUB_TYP[1]), .B(n578), .Y(n622) );
INVX2 U277 ( .A(SUB_TYP[0]), .Y(n578) );
NAND3XL U278 ( .A(n626), .B(n616), .C(n666), .Y(n623) );
AOI31X1 U279 ( .A0(n655), .A1(n658), .A2(n659), .B0(n652), .Y(n624) );
OAI221X1 U280 ( .A0(SUB_TYP[1]), .A1(n615), .B0(SUB_TYP[1]), .B1(n616),
      .C0(n617), .Y(CXD301_2) );
NOR2XL U281 ( .A(n632), .B(n633), .Y(n615) );
INVX2 U282 ( .A(n653), .Y(n632) );
INVX2 U283 ( .A(n645), .Y(n633) );
NAND2X1 U284 ( .A(n325), .B(n591), .Y(n616) );
INVX2 U285 ( .A(n635), .Y(n591) );
AOI221X1 U286 ( .A0(n612), .A1(n627), .B0(n655), .B1(n630), .C0(n652), .Y(
      n617) );
NAND4BX1 U287 ( .AN(n568), .B(n614), .C(n569), .D(n625), .Y(n302_4) );
OAI222X1 U288 ( .A0(n577), .A1(n585), .B0(SUB_TYP[1]), .B1(n613), .C0(n570
      ), .C1(n622), .Y(n568) );
NAND2X1 U289 ( .A(n612), .B(n558), .Y(n614) );
INVX2 U290 ( .A(n622), .Y(n612) );
OAI31X1 U291 ( .A0(n646), .A1(n595), .A2(n592), .B0(n626), .Y(n569) );
INVX2 U292 ( .A(n651), .Y(n595) );
INVX2 U293 ( .A(SUB_TYP[1]), .Y(n626) );
NAND3XL U294 ( .A(n655), .B(n597), .C(n629), .Y(n625) );
INVX2 U295 ( .A(n585), .Y(n655) );
INVX2 U296 ( .A(n571), .Y(n597) );
INVX2 U297 ( .A(n574), .Y(n629) );
TLATX1 CXD_reg_3_ ( .D(CXD301_3), .G(n302_4), .Q(CXD[3]) );
TLATX1 CXD_reg_1_ ( .D(CXD301_1), .G(n302_4), .Q(CXD[1]) );
TLATX1 CXD_reg_0_ ( .D(CXD301_0), .G(n302_4), .Q(CXD[0]) );
```

```
    TLATX1 CXD_reg_2_ ( .D(CXD301_2), .G(n302_4), .Q(CXD[2]) );
    OR2X2 U298 ( .A(net9159), .B(n564), .Y(n554) );
    XOR2XL U299 ( .A(n267_2), .B(n564), .Y(n555) );
    NOR2X1 U300 ( .A(SUM_D_2), .B(SUM_D_1), .Y(n556) );
    AND2X2 U301 ( .A(n587), .B(n586), .Y(n557) );
    AND2X2 U302 ( .A(n562), .B(n567), .Y(n558) );
    AND3X1 U303 ( .A(n566), .B(n567), .C(n557), .Y(n559) );
    AND2X2 U304 ( .A(n645), .B(n618), .Y(n560) );
    INVX2 U305 ( .A(SUM_D_2), .Y(n602) );
    INVX2 U306 ( .A(net9159), .Y(n267_2) );
    AOI21X1 U307 ( .A0(net9160), .A1(S_C2), .B0(net9161), .Y(net9159) );
    INVX2 U308 ( .A(net9250), .Y(n261_2) );
    AOI21X1 U309 ( .A0(net9251), .A1(S_B2), .B0(net9252), .Y(net9250) );
    AND2X2 U310 ( .A(S_D1), .B(S_B1), .Y(n561) );
    AND2X2 U311 ( .A(S_C2), .B(S_C0), .Y(n562) );
    NAND2X1 U312 ( .A(S_B0), .B(n261_3), .Y(n563) );
    NAND2X1 U313 ( .A(n267_3), .B(S_B1), .Y(n564) );
    OAI2BB2X1 U314 ( .A0N(SUM_H_0), .A1N(SUB_TYP[1]), .B0(n612), .B1(n567),
        .Y(acell_3968_U43_Z_0) );
    OAI2BB2X1 U315 ( .A0N(n561), .A1N(SUB_TYP[1]), .B0(n612), .B1(n566), .Y(
        acell_3968_U43_Z_1) );
    NAND2BX1 U316 ( .AN(acell_3968_U43_Z_0), .B(n611), .Y(n377) );
    NOR2XL U317 ( .A(net9250), .B(n563), .Y(SUM_D_2) );
    TLATX1 CXD_reg_4_ ( .D(1'b0), .G(n302_4), .Q(CXD[4]) );
endmodule
module ZC_COLUMN_P1_P3 ( CXD_1, CXD_2, CXD_3, CXD_4, V0_C, S1_B, S1_C, S1_D,
    S3_B, S3_C, S3_D, P1_S_B, P1_S_C, P1_B_C, P3_S_C, SUB_TYP );
output [5:0] CXD_1;
output [5:0] CXD_2;
input  [4:0] S3_B;
input  [4:1] P1_B_C;
output [5:0] CXD_3;
input  [4:0] S3_D;
output [5:0] CXD_4;
input  [4:1] V0_C;
input  [4:0] S1_D;
input  [4:0] S3_C;
input  [4:0] S1_C;
input  [4:1] P1_S_C;
```

```
input    [1:0] SUB_TYP;
input    [4:0] S1_B;
input    [4:1] P1_S_B;
input    [3:1] P3_S_C;
    wire S_T_C_2, S_T_C_4, S_T_C_0, S_T_B_4, S_T_B_0, S_T_D_3, S_T_B_2,
        S_T_D_1, S_T_D_0, S_T_B_3, S_T_D_4, S_T_C_1, S_T_D_2, S_T_C_3, n1587,
        n1588, n1589, n1590, n1591, n1592, n1593, n1594, n1595, n1596, n1597,
        n1598, n1599, n1600, n1601, n1602, n1603, n1604, n1605, n1606, n1607,
        n1608, n1609, n1610, n1611, n1612, n1613, n1614, n1615, n1616, n1617,
        n1618, n1619, n1620, n1621, n1622, n1623;
    ZC_P13_1 ZC_1 ( .CXD(CXD_1), .S_B0(S_T_B_0), .S_B1(n1592), .S_B2(S_T_B_2),
        .S_C0(S_T_C_0), .V0_Cn(V0_C[1]), .S_C2(S_T_C_2), .S_D0(S_T_D_0),
        .S_D1(S_T_D_1), .S_D2(S_T_D_2), .SUB_TYP(SUB_TYP) );
    ZC_P13_2 ZC_2 ( .CXD(CXD_2), .S_B0(n1592), .S_B1(S_T_B_2), .S_B2(S_T_B_3),
        .S_C0(S_T_C_1), .V0_Cn(V0_C[2]), .S_C2(S_T_C_3), .S_D0(S_T_D_1),
        .S_D1(S_T_D_2), .S_D2(S_T_D_3), .SUB_TYP(SUB_TYP) );
    ZC_P13_3 ZC_3 ( .CXD(CXD_3), .S_B0(S_T_B_2), .S_B1(S_T_B_3), .S_B2(S_T_B_4
        ), .S_C0(S_T_C_2), .V0_Cn(V0_C[3]), .S_C2(S_T_C_4), .S_D0(S_T_D_2),
        .S_D1(S_T_D_3), .S_D2(S_T_D_4), .SUB_TYP(SUB_TYP) );
    ZC_P13_4 ZC_4 ( .CXD(CXD_4), .S_B0(S_T_B_3), .S_B1(S_T_B_4), .S_B2(1'b0),
        .S_C0(S_T_C_3), .V0_Cn(V0_C[4]), .S_C2(1'b0), .S_D0(S_T_D_3), .S_D1(
        S_T_D_4), .S_D2(1'b0), .SUB_TYP(SUB_TYP) );
    AOI222X1 U265 ( .A0(n1588), .A1(n1593), .B0(n1588), .B1(P1_B_C[1]), .C0(
        P1_B_C[2]), .C1(n1588), .Y(n1592) );
    AOI31X1 U266 ( .A0(n1599), .A1(n1600), .A2(P1_B_C[3]), .B0(n1601), .Y(
        S_T_C_2) );
    NOR2XL U267 ( .A(P1_B_C[2]), .B(P1_B_C[3]), .Y(n1608) );
    AOI222X1 U268 ( .A0(n1587), .A1(n1598), .B0(n1587), .B1(P1_B_C[3]), .C0(
        n1587), .C1(P1_B_C[4]), .Y(S_T_B_4) );
    NOR2XL U269 ( .A(P1_B_C[2]), .B(P1_B_C[1]), .Y(n1609) );
    OAI2BB2X1 U270 ( .A0N(n1618), .A1N(n1619), .B0(n1607), .B1(S1_D[4]), .Y(
        n1617) );
    NAND2X1 U271 ( .A(S3_D[4]), .B(n1612), .Y(n1619) );
    AOI211X1 U272 ( .A0(P1_S_C[4]), .A1(n1607), .B0(S3_C[4]), .C0(S1_C[4]),
        .Y(n1605) );
    NOR2XL U273 ( .A(n1612), .B(n1614), .Y(n1595) );
    AOI31X1 U274 ( .A0(n1607), .A1(n1611), .A2(P1_S_B[3]), .B0(n1614), .Y(
        n1594) );
    OR2X2 U275 ( .A(S1_B[3]), .B(S3_B[3]), .Y(n1614) );
```

```
AOI211X1 U276 ( .A0(n1613), .A1(n1611), .B0(P1_S_C[1]), .C0(S1_C[1]), .Y(
      n1603) );
OR2X2 U277 ( .A(S3_C[1]), .B(P3_S_C[1]), .Y(n1613) );
AOI211X1 U278 ( .A0(P1_S_C[3]), .A1(n1611), .B0(S1_C[3]), .C0(S3_C[3]),
      .Y(n1591) );
AND2X2 U279 ( .A(S3_B[0]), .B(n1610), .Y(n1604) );
AND2X2 U280 ( .A(S3_D[0]), .B(n1610), .Y(n1602) );
NOR2XL U281 ( .A(n1607), .B(n1615), .Y(n1597) );
OR2X2 U282 ( .A(S1_B[2]), .B(S3_B[2]), .Y(n1615) );
NAND2X1 U283 ( .A(S3_D[1]), .B(n1611), .Y(n1622) );
AND2X2 U284 ( .A(S3_C[0]), .B(n1610), .Y(n1606) );
AOI211X1 U285 ( .A0(P1_S_C[2]), .A1(n1610), .B0(S3_C[2]), .C0(S1_C[2]),
      .Y(n1601) );
OAI222X1 U286 ( .A0(S1_D[3]), .A1(n1612), .B0(S1_D[3]), .B1(S3_D[3]), .C0(
      n1608), .C1(S1_D[3]), .Y(n1623) );
NOR2X1 U287 ( .A(n1594), .B(n1595), .Y(S_T_B_3) );
INVX2 U288 ( .A(n1617), .Y(S_T_D_4) );
OAI222X1 U289 ( .A0(S1_D[2]), .A1(n1607), .B0(S1_D[2]), .B1(S3_D[2]), .C0(
      n1609), .C1(S1_D[2]), .Y(n1616) );
NOR2X1 U290 ( .A(n1596), .B(n1597), .Y(S_T_B_2) );
INVX2 U291 ( .A(n1605), .Y(S_T_C_4) );
INVX2 U292 ( .A(n1623), .Y(S_T_D_3) );
INVX2 U293 ( .A(n1603), .Y(S_T_C_1) );
INVX2 U294 ( .A(n1616), .Y(S_T_D_2) );
OR2X2 U295 ( .A(n1602), .B(S1_D[0]), .Y(S_T_D_0) );
OR2X2 U296 ( .A(n1604), .B(S1_B[0]), .Y(S_T_B_0) );
INVX2 U297 ( .A(S1_C[2]), .Y(n1600) );
OR2X2 U298 ( .A(n1606), .B(S1_C[0]), .Y(S_T_C_0) );
INVX2 U299 ( .A(n1620), .Y(S_T_D_1) );
OAI2BB2X1 U300 ( .A0N(n1621), .A1N(n1622), .B0(n1610), .B1(S1_D[1]), .Y(
      n1620) );
INVX2 U301 ( .A(P1_S_B[4]), .Y(n1598) );
INVX2 U302 ( .A(P1_S_C[3]), .Y(n1590) );
INVX2 U303 ( .A(P1_S_C[2]), .Y(n1599) );
INVX2 U304 ( .A(P1_B_C[1]), .Y(n1610) );
INVX2 U305 ( .A(P1_B_C[4]), .Y(n1612) );
INVX2 U306 ( .A(P1_B_C[3]), .Y(n1607) );
INVX2 U307 ( .A(P1_B_C[2]), .Y(n1611) );
INVX2 U308 ( .A(S1_C[3]), .Y(n1589) );
```

```
    INVX2 U309 ( .A(S1_D[4]), .Y(n1618) );
    INVX2 U310 ( .A(S1_D[1]), .Y(n1621) );
    NOR2XL U311 ( .A(S1_B[4]), .B(S3_B[4]), .Y(n1587) );
    NOR2XL U312 ( .A(S1_B[1]), .B(S3_B[1]), .Y(n1588) );
    INVX2 U313 ( .A(P1_S_B[1]), .Y(n1593) );
    AOI31XL U314 ( .A0(n1610), .A1(n1611), .A2(P1_S_B[2]), .B0(n1615), .Y(
        n1596) );
    AOI31X2 U315 ( .A0(n1589), .A1(n1590), .A2(P1_B_C[4]), .B0(n1591), .Y(
        S_T_C_3) );
endmodule
module SC_P13_1 ( CXD, S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn );
output [5:0] CXD;
input  S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn;
    wire n135, n154, n155, n156, n157, n158, n159, n161, n162, n163, n164,
        n165, n166, n167, n168, n169, n170, n171, n172, n173, n174, n175, n176,
        n177, n178;
    assign CXD[4] = 1'b0;
    assign CXD[3] = 1'b1;
    MX2X1 U43 ( .S0(S_V2), .B(n178), .A(n174), .Y(n175) );
    NAND2X1 U44 ( .A(n163), .B(n167), .Y(n178) );
    MX2X1 U45 ( .S0(S_H2), .B(n177), .A(n173), .Y(n169) );
    NAND2X1 U46 ( .A(n161), .B(n165), .Y(n177) );
    NAND2X1 U47 ( .A(X_V2), .B(n174), .Y(n164) );
    NAND2X1 U48 ( .A(S_V1), .B(n168), .Y(n174) );
    NAND2X1 U49 ( .A(X_V1), .B(S_V1), .Y(n163) );
    NAND2X1 U50 ( .A(X_H2), .B(n173), .Y(n162) );
    NAND2X1 U51 ( .A(S_H1), .B(n166), .Y(n173) );
    NAND2X1 U52 ( .A(X_H1), .B(S_H1), .Y(n161) );
    XOR2XL U53 ( .A(n159), .B(n170), .Y(CXD[0]) );
    OAI2BB2X1 U54 ( .A0N(n158), .A1N(n159), .B0(n154), .B1(n157), .Y(CXD[1])
        );
    OAI21XL U55 ( .A0(n135), .A1(n176), .B0(n156), .Y(n158) );
    NAND2X1 U56 ( .A(n175), .B(n154), .Y(n159) );
    OAI2BB2X1 U57 ( .A0N(n155), .A1N(n156), .B0(n135), .B1(n154), .Y(CXD[2])
        );
    MX2X1 U58 ( .S0(S_H2), .B(n162), .A(n161), .Y(n135) );
    MX2X1 U59 ( .S0(S_V2), .B(n164), .A(n163), .Y(n154) );
    OAI21XL U60 ( .A0(n157), .A1(n176), .B0(n159), .Y(n155) );
    INVX2 U61 ( .A(n135), .Y(n157) );
```

```
    INVX2 U62 ( .A(n154), .Y(n176) );
    NAND2X1 U63 ( .A(n169), .B(n135), .Y(n156) );
    MX2X1 U64 ( .S0(X_Cn), .B(n171), .A(n172), .Y(CXD[5]) );
    OAI21XL U65 ( .A0(n154), .A1(n156), .B0(n135), .Y(n172) );
    MX2X1 U66 ( .S0(n170), .B(n154), .A(n135), .Y(n171) );
    INVX2 U67 ( .A(n156), .Y(n170) );
    INVX2 U68 ( .A(X_H1), .Y(n166) );
    INVX2 U69 ( .A(X_V2), .Y(n167) );
    INVX2 U70 ( .A(X_V1), .Y(n168) );
    INVX2 U71 ( .A(X_H2), .Y(n165) );
endmodule

module SC_P13_2 ( CXD, S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn );
output [5:0] CXD;
input  S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn;
    wire n110, n129, n130, n131, n132, n133, n134, n136, n137, n138, n139,
         n140, n141, n142, n143, n144, n145, n146, n147, n148, n149, n150, n151,
         n152, n153;
    assign CXD[4] = 1'b0;
    assign CXD[3] = 1'b1;
    MX2X1 U43 ( .S0(S_V2), .B(n153), .A(n149), .Y(n150) );
    NAND2X1 U44 ( .A(n138), .B(n142), .Y(n153) );
    MX2X1 U45 ( .S0(S_H2), .B(n152), .A(n148), .Y(n144) );
    NAND2X1 U46 ( .A(n136), .B(n140), .Y(n152) );
    NAND2X1 U47 ( .A(X_V2), .B(n149), .Y(n139) );
    NAND2X1 U48 ( .A(S_V1), .B(n143), .Y(n149) );
    NAND2X1 U49 ( .A(X_V1), .B(S_V1), .Y(n138) );
    NAND2X1 U50 ( .A(X_H2), .B(n148), .Y(n137) );
    NAND2X1 U51 ( .A(S_H1), .B(n141), .Y(n148) );
    NAND2X1 U52 ( .A(X_H1), .B(S_H1), .Y(n136) );
    XOR2XL U53 ( .A(n134), .B(n145), .Y(CXD[0]) );
    OAI2BB2X1 U54 ( .A0N(n133), .A1N(n134), .B0(n129), .B1(n132), .Y(CXD[1])
        );
    OAI21XL U55 ( .A0(n110), .A1(n151), .B0(n131), .Y(n133) );
    NAND2X1 U56 ( .A(n150), .B(n129), .Y(n134) );
    OAI2BB2X1 U57 ( .A0N(n130), .A1N(n131), .B0(n110), .B1(n129), .Y(CXD[2])
        );
    MX2X1 U58 ( .S0(S_H2), .B(n137), .A(n136), .Y(n110) );
    MX2X1 U59 ( .S0(S_V2), .B(n139), .A(n138), .Y(n129) );
```

```
    OAI21XL U60 ( .A0(n132), .A1(n151), .B0(n134), .Y(n130) );
    INVX2 U61 ( .A(n110), .Y(n132) );
    INVX2 U62 ( .A(n129), .Y(n151) );
    NAND2X1 U63 ( .A(n144), .B(n110), .Y(n131) );
    MX2X1 U64 ( .S0(X_Cn), .B(n146), .A(n147), .Y(CXD[5]) );
    OAI21XL U65 ( .A0(n129), .A1(n131), .B0(n110), .Y(n147) );
    MX2X1 U66 ( .S0(n145), .B(n129), .A(n110), .Y(n146) );
    INVX2 U67 ( .A(n131), .Y(n145) );
    INVX2 U68 ( .A(X_H1), .Y(n141) );
    INVX2 U69 ( .A(X_V2), .Y(n142) );
    INVX2 U70 ( .A(X_V1), .Y(n143) );
    INVX2 U71 ( .A(X_H2), .Y(n140) );
endmodule
module SC_P13_3 ( CXD, S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn );
output [5:0] CXD;
input  S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn;
    wire n85, n104, n105, n106, n107, n108, n109, n111, n112, n113, n114, n115,
        n116, n117, n118, n119, n120, n121, n122, n123, n124, n125, n126, n127,
        n128;
    assign CXD[4] = 1'b0;
    assign CXD[3] = 1'b1;
    NAND2X1 U43 ( .A(n113), .B(n117), .Y(n128) );
    MX2X1 U44 ( .S0(S_V2), .B(n128), .A(n124), .Y(n125) );
    MX2X1 U45 ( .S0(S_H2), .B(n127), .A(n123), .Y(n119) );
    NAND2X1 U46 ( .A(n111), .B(n115), .Y(n127) );
    NAND2X1 U47 ( .A(X_V2), .B(n124), .Y(n114) );
    NAND2X1 U48 ( .A(S_V1), .B(n118), .Y(n124) );
    NAND2X1 U49 ( .A(X_V1), .B(S_V1), .Y(n113) );
    NAND2X1 U50 ( .A(X_H2), .B(n123), .Y(n112) );
    NAND2X1 U51 ( .A(S_H1), .B(n116), .Y(n123) );
    NAND2X1 U52 ( .A(X_H1), .B(S_H1), .Y(n111) );
    XOR2XL U53 ( .A(n109), .B(n120), .Y(CXD[0]) );
    OAI2BB2X1 U54 ( .A0N(n108), .A1N(n109), .B0(n104), .B1(n107), .Y(CXD[1])
        );
    OAI21XL U55 ( .A0(n85), .A1(n126), .B0(n106), .Y(n108) );
    NAND2X1 U56 ( .A(n125), .B(n104), .Y(n109) );
    OAI2BB2X1 U57 ( .A0N(n105), .A1N(n106), .B0(n85), .B1(n104), .Y(CXD[2]) );
    MX2X1 U58 ( .S0(S_H2), .B(n112), .A(n111), .Y(n85) );
    MX2X1 U59 ( .S0(S_V2), .B(n114), .A(n113), .Y(n104) );
```

```
    OAI21XL U60 ( .A0(n107), .A1(n126), .B0(n109), .Y(n105) );
    INVX2 U61 ( .A(n85), .Y(n107) );
    INVX2 U62 ( .A(n104), .Y(n126) );
    NAND2X1 U63 ( .A(n119), .B(n85), .Y(n106) );
    MX2X1 U64 ( .S0(X_Cn), .B(n121), .A(n122), .Y(CXD[5]) );
    OAI21XL U65 ( .A0(n104), .A1(n106), .B0(n85), .Y(n122) );
    MX2X1 U66 ( .S0(n120), .B(n104), .A(n85), .Y(n121) );
    INVX2 U67 ( .A(n106), .Y(n120) );
    INVX2 U68 ( .A(X_H1), .Y(n116) );
    INVX2 U69 ( .A(X_V2), .Y(n117) );
    INVX2 U70 ( .A(X_V1), .Y(n118) );
    INVX2 U71 ( .A(X_H2), .Y(n115) );
endmodule
module SC_P13_4 ( CXD, S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn );
output [5:0] CXD;
input  S_V1, S_V2, S_H1, S_H2, X_V1, X_V2, X_H1, X_H2, X_Cn;
    wire n78, n79, n80, n81, n82, n83, n84, n86, n87, n88, n89, n90, n91, n92,
         n93, n94, n95, n96, n97, n98, n99, n100, n101, n102, n103;
    assign CXD[4] = 1'b0;
    assign CXD[3] = 1'b1;
    MX2X1 U43 ( .S0(S_V2), .B(n103), .A(n99), .Y(n100) );
    NAND2X1 U44 ( .A(n88), .B(n92), .Y(n103) );
    INVX2 U45 ( .A(X_V2), .Y(n92) );
    MX2X1 U46 ( .S0(S_H2), .B(n102), .A(n98), .Y(n94) );
    NAND2X1 U47 ( .A(n86), .B(n90), .Y(n102) );
    NAND2X1 U48 ( .A(X_V2), .B(n99), .Y(n89) );
    NAND2X1 U49 ( .A(S_V1), .B(n93), .Y(n99) );
    NAND2X1 U50 ( .A(X_V1), .B(S_V1), .Y(n88) );
    NAND2X1 U51 ( .A(X_H2), .B(n98), .Y(n87) );
    NAND2X1 U52 ( .A(S_H1), .B(n91), .Y(n98) );
    NAND2X1 U53 ( .A(X_H1), .B(S_H1), .Y(n86) );
    XOR2XL U54 ( .A(n84), .B(n95), .Y(CXD[0]) );
    OAI2BB2X1 U55 ( .A0N(n83), .A1N(n84), .B0(n79), .B1(n82), .Y(CXD[1]) );
    OAI21XL U56 ( .A0(n78), .A1(n101), .B0(n81), .Y(n83) );
    NAND2X1 U57 ( .A(n100), .B(n79), .Y(n84) );
    OAI2BB2X1 U58 ( .A0N(n80), .A1N(n81), .B0(n78), .B1(n79), .Y(CXD[2]) );
    MX2X1 U59 ( .S0(S_H2), .B(n87), .A(n86), .Y(n78) );
    MX2X1 U60 ( .S0(S_V2), .B(n89), .A(n88), .Y(n79) );
    OAI21XL U61 ( .A0(n82), .A1(n101), .B0(n84), .Y(n80) );
```

```
    INVX2 U62 ( .A(n78), .Y(n82) );
    INVX2 U63 ( .A(n79), .Y(n101) );
    NAND2X1 U64 ( .A(n94), .B(n78), .Y(n81) );
    MX2X1 U65 ( .S0(X_Cn), .B(n96), .A(n97), .Y(CXD[5]) );
    OAI21XL U66 ( .A0(n79), .A1(n81), .B0(n78), .Y(n97) );
    MX2X1 U67 ( .S0(n95), .B(n79), .A(n78), .Y(n96) );
    INVX2 U68 ( .A(n81), .Y(n95) );
    INVX2 U69 ( .A(X_H1), .Y(n91) );
    INVX2 U70 ( .A(X_V1), .Y(n93) );
    INVX2 U71 ( .A(X_H2), .Y(n90) );
endmodule
module SC_COLUMN_P1_P3 ( CXD_1, CXD_2, CXD_3, CXD_4, X0_B, X0_C, X0_D, S1_B,
    S1_C, S1_D, S3_B, S3_C, S3_D, P1_S_B, P1_S_C, P1_B_C, P3_S_C );
output [5:0] CXD_1;
output [5:0] CXD_2;
input  [4:0] S3_B;
input  [4:1] P1_B_C;
output [5:0] CXD_3;
input  [4:0] S3_D;
output [5:0] CXD_4;
input  [4:0] X0_C;
input  [4:0] S1_D;
input  [4:0] S3_C;
input  [4:0] X0_D;
input  [4:1] P1_S_C;
input  [4:0] S1_C;
input  [4:0] X0_B;
input  [4:0] S1_B;
input  [4:1] P1_S_B;
input  [4:1] P3_S_C;
    wire S_V1_1, S_V2_3, S_H1_2, S_V2_2, S_H1_3, S_H1_4, S_H2_1, S_V1_2,
        S_H2_3, S_H1_1, S_V2_1, S_V1_3, S_V1_4, S_H2_2, val817_1, n826, n827,
        n828, n829, n830, n831, n832, n833, n834, n835, n836, n837, n838, n839,
        n840, n841, n842, n843, n844, n845, n846, n847, n848, n849;
    wire SYNOPSYS_UNCONNECTED_1 , SYNOPSYS_UNCONNECTED_2 ,
      SYNOPSYS_UNCONNECTED_3 , SYNOPSYS_UNCONNECTED_4 , SYNOPSYS_UNCONNECTED_5 ,
      SYNOPSYS_UNCONNECTED_6 , SYNOPSYS_UNCONNECTED_7 , SYNOPSYS_UNCONNECTED_8 ;
    assign CXD_1[4] = 1'b0;
    assign CXD_1[3] = 1'b1;
```

```
assign CXD_2[4] = 1'b0;
assign CXD_2[3] = 1'b1;
assign CXD_3[4] = 1'b0;
assign CXD_3[3] = 1'b1;
assign CXD_4[4] = 1'b0;
assign CXD_4[3] = 1'b1;
SC_P13_1 SC_1 ( .CXD({CXD_1[5], SYNOPSYS_UNCONNECTED_1,
      SYNOPSYS_UNCONNECTED_2, CXD_1[2], CXD_1[1], CXD_1[0]}), .S_V1(S_V1_1),
      .S_V2(S_V2_1), .S_H1(S_H1_1), .S_H2(S_H2_1), .X_V1(X0_C[0]), .X_V2(
      X0_C[2]), .X_H1(X0_B[1]), .X_H2(X0_D[1]), .X_Cn(X0_C[1]) );
SC_P13_2 SC_2 ( .CXD({CXD_2[5], SYNOPSYS_UNCONNECTED_3,
      SYNOPSYS_UNCONNECTED_4, CXD_2[2], CXD_2[1], CXD_2[0]}), .S_V1(S_V1_2),
      .S_V2(S_V2_2), .S_H1(S_H1_2), .S_H2(S_H2_2), .X_V1(X0_C[1]), .X_V2(
      X0_C[3]), .X_H1(X0_B[2]), .X_H2(X0_D[2]), .X_Cn(X0_C[2]) );
SC_P13_3 SC_3 ( .CXD({CXD_3[5], SYNOPSYS_UNCONNECTED_5,
      SYNOPSYS_UNCONNECTED_6, CXD_3[2], CXD_3[1], CXD_3[0]}), .S_V1(S_V1_3),
      .S_V2(S_V2_3), .S_H1(S_H1_3), .S_H2(S_H2_3), .X_V1(X0_C[2]), .X_V2(
      X0_C[4]), .X_H1(X0_B[3]), .X_H2(X0_D[3]), .X_Cn(X0_C[3]) );
SC_P13_4 SC_4 ( .CXD({CXD_4[5], SYNOPSYS_UNCONNECTED_7,
      SYNOPSYS_UNCONNECTED_8, CXD_4[2], CXD_4[1], CXD_4[0]}), .S_V1(S_V1_4),
      .S_V2(1'b0), .S_H1(S_H1_4), .S_H2(val817_1), .X_V1(X0_C[3]), .X_V2(
      1'b0), .X_H1(X0_B[4]), .X_H2(X0_D[4]), .X_Cn(X0_C[4]) );
NOR2XL U138 ( .A(S3_C[3]), .B(P3_S_C[3]), .Y(n844) );
NOR2XL U139 ( .A(S3_C[2]), .B(P3_S_C[2]), .Y(n840) );
OR2X2 U140 ( .A(S3_C[1]), .B(P3_S_C[1]), .Y(n847) );
AND2X2 U141 ( .A(S3_C[0]), .B(n849), .Y(n834) );
OAI211XL U142 ( .A0(P1_B_C[4]), .A1(n844), .B0(n839), .C0(n837), .Y(S_V1_4
      ) );
OR2X2 U143 ( .A(n829), .B(S1_D[4]), .Y(val817_1) );
AND2X2 U144 ( .A(S3_D[4]), .B(n845), .Y(n829) );
INVX2 U145 ( .A(n827), .Y(S_H1_4) );
AOI211X1 U146 ( .A0(P1_S_B[4]), .A1(n845), .B0(S3_B[4]), .C0(S1_B[4]), .Y(
      n827) );
AOI211X1 U147 ( .A0(P1_S_C[4]), .A1(n846), .B0(S3_C[4]), .C0(S1_C[4]), .Y(
      n833) );
OAI211XL U148 ( .A0(P1_B_C[3]), .A1(n840), .B0(n830), .C0(n831), .Y(S_V1_3
      ) );
OR2X2 U149 ( .A(n835), .B(S1_D[3]), .Y(S_H2_3) );
AND2X2 U150 ( .A(S3_D[3]), .B(n846), .Y(n835) );
```

```
    INVX2 U151 ( .A(n843), .Y(S_H1_3) );
    AOI211X1 U152 ( .A0(P1_S_B[3]), .A1(n846), .B0(S3_B[3]), .C0(S1_B[3]), .Y(
        n843) );
    OAI211XL U153 ( .A0(P1_B_C[2]), .A1(n837), .B0(n838), .C0(n839), .Y(S_V2_2
        ) );
    INVX2 U154 ( .A(P1_S_C[3]), .Y(n837) );
    INVX2 U155 ( .A(S3_C[3]), .Y(n838) );
    INVX2 U156 ( .A(n826), .Y(S_V1_2) );
    AOI211X1 U157 ( .A0(n847), .A1(n848), .B0(P1_S_C[1]), .C0(S1_C[1]), .Y(
        n826) );
    OR2X2 U158 ( .A(n828), .B(S1_D[2]), .Y(S_H2_2) );
    AND2X2 U159 ( .A(S3_D[2]), .B(n848), .Y(n828) );
    INVX2 U160 ( .A(n836), .Y(S_H1_2) );
    OAI211XL U161 ( .A0(P1_B_C[1]), .A1(n830), .B0(n831), .C0(n832), .Y(S_V2_1
        ) );
    INVX2 U162 ( .A(S1_C[2]), .Y(n831) );
    INVX2 U163 ( .A(S3_C[2]), .Y(n832) );
    OR2X2 U164 ( .A(n834), .B(S1_C[0]), .Y(S_V1_1) );
    OR2X2 U165 ( .A(n842), .B(S1_D[1]), .Y(S_H2_1) );
    AND2X2 U166 ( .A(S3_D[1]), .B(n849), .Y(n842) );
    INVX2 U167 ( .A(n841), .Y(S_H1_1) );
    INVX2 U168 ( .A(P1_S_C[2]), .Y(n830) );
    INVX2 U169 ( .A(n833), .Y(S_V2_3) );
    INVX2 U170 ( .A(P1_B_C[1]), .Y(n849) );
    INVX2 U171 ( .A(P1_B_C[4]), .Y(n845) );
    INVX2 U172 ( .A(P1_B_C[3]), .Y(n846) );
    INVX2 U173 ( .A(P1_B_C[2]), .Y(n848) );
    INVX2 U174 ( .A(S1_C[3]), .Y(n839) );
    AOI211X1 U175 ( .A0(P1_S_B[1]), .A1(n849), .B0(S3_B[1]), .C0(S1_B[1]), .Y(
        n841) );
    AOI211XL U176 ( .A0(P1_S_B[2]), .A1(n848), .B0(S3_B[2]), .C0(S1_B[2]), .Y(
        n836) );
endmodule
module RLC_COLUMN_0 ( CXD_RLC, CXD_P3_RLC );
output [5:0] CXD_RLC;
input  CXD_P3_RLC;
    wire CXD_P3_RLC_wire;
    assign CXD_RLC[5] = CXD_P3_RLC_wire;
    assign CXD_RLC[4] = 1'b1;
```

```
        assign CXD_RLC[3] = 1'b0;
        assign CXD_RLC[2] = 1'b0;
        assign CXD_RLC[1] = 1'b0;
        assign CXD_RLC[0] = 1'b1;
        assign CXD_P3_RLC_wire = CXD_P3_RLC;
endmodule
module RLC_COLUMN_1 ( CXD_RLC, CXD_P3_RLC );
output [5:0] CXD_RLC;
input    CXD_P3_RLC;
        wire CXD_P3_RLC_wire;
        assign CXD_RLC[5] = CXD_P3_RLC_wire;
        assign CXD_RLC[4] = 1'b1;
        assign CXD_RLC[3] = 1'b0;
        assign CXD_RLC[2] = 1'b0;
        assign CXD_RLC[1] = 1'b0;
        assign CXD_RLC[0] = 1'b1;
        assign CXD_P3_RLC_wire = CXD_P3_RLC;
endmodule
module RLC1_UC0 ( CXD_UC, CXD_P3_UC );
output [5:0] CXD_UC;
input    CXD_P3_UC;
        wire CXD_P3_UC_wire;
        assign CXD_UC[5] = CXD_P3_UC_wire;
        assign CXD_UC[4] = 1'b1;
        assign CXD_UC[3] = 1'b0;
        assign CXD_UC[2] = 1'b0;
        assign CXD_UC[1] = 1'b1;
        assign CXD_UC[0] = 1'b0;
        assign CXD_P3_UC_wire = CXD_P3_UC;
endmodule
module RLC1_UC1 ( CXD_UC, CXD_P3_UC );
output [5:0] CXD_UC;
input    CXD_P3_UC;
        wire CXD_P3_UC_wire;
        assign CXD_UC[5] = CXD_P3_UC_wire;
        assign CXD_UC[4] = 1'b1;
        assign CXD_UC[3] = 1'b0;
        assign CXD_UC[2] = 1'b0;
        assign CXD_UC[1] = 1'b1;
```

```
        assign CXD_UC[0] = 1'b0;
        assign CXD_P3_UC_wire = CXD_P3_UC;
endmodule
module PCF ( CXD_P1_P3_1_ZC, CXD_P1_P3_2_ZC, CXD_P1_P3_3_ZC, CXD_P1_P3_4_ZC,
        CXD_P1_P3_1_SC, CXD_P1_P3_2_SC, CXD_P1_P3_3_SC, CXD_P1_P3_4_SC,
        CXD_P2_1_MRC, CXD_P2_2_MRC, CXD_P2_3_MRC, CXD_P2_4_MRC, CXD_P3_RLC_0,
        CXD_P3_RLC_1, CXD_P3_RLC1_UC0, CXD_P3_RLC1_UC1, P1_B_C, P2_C, P3_C, S1_D_,
        S3_D_, ADDRA, ADDRB, WRITE, BP_INDEX, X0_A_4_0, V0_A_4_1, S1_A_4_0,
        S3_A_4_0, START, SUB_TYP, BP_NUM, RST, CLK );
output [5:0] CXD_P1_P3_1_ZC;
output [5:0] CXD_P1_P3_4_ZC;
output [5:0] CXD_P1_P3_4_SC;
output [5:0] CXD_P2_2_MRC;
output [5:0] CXD_P2_4_MRC;
output [4:1] P1_B_C;
output [4:1] S1_D_;
input  [3:0] BP_NUM;
input  [4:0] X0_A_4_0;
output [5:0] CXD_P1_P3_2_ZC;
output [5:0] CXD_P1_P3_1_SC;
input  [4:0] S1_A_4_0;
output [5:0] CXD_P1_P3_2_SC;
output [5:0] CXD_P2_3_MRC;
output [9:0] ADDRA;
output [3:0] BP_INDEX;
output [5:0] CXD_P1_P3_3_ZC;
output [5:0] CXD_P1_P3_3_SC;
output [5:0] CXD_P2_1_MRC;
output [5:0] CXD_P3_RLC_0;
output [4:1] P2_C;
output [4:1] S3_D_;
input  [1:0] SUB_TYP;
output [5:0] CXD_P3_RLC1_UC0;
output [5:0] CXD_P3_RLC1_UC1;
output [4:1] P3_C;
input  [4:1] V0_A_4_1;
output [5:0] CXD_P3_RLC_1;
output [9:0] ADDRB;
input  [4:0] S3_A_4_0;
```

```
input    START, RST, CLK;
output WRITE;
      wire \S3_C[2] , \S1_B[1] , \S1_A[2] , \P2_1_C[3] , \X0_B[0] , \X0_A[3] ,
           \V0_B[4] , \P1_S_C[4] , \X0_D[3] , \V0_A[3] , \X0_B[4] , S1_D_0,
           \V0_B[2] , \V0_A[1] , \X0_D[1] , \S3_C[4] , \S1_A[4] , \X0_B[2] ,
           \X0_A[1] , \P1_S_C[2] , \S1_B[3] , \S1_A[0] , \P2_1_C[1] , \S3_C[0] ,
           \V0_C[2] , \P1_B[4] , \S3_B[4] , \X0_C[2] , \S3_A[3] , \P3_S_C[2] ,
           \S3_B[0] , \S1_C[3] , \P1_S_B[2] , \S1_C[1] , \S3_B[2] , \S3_A[1] ,
           \P1_B[2] , \X0_C[0] , \V0_C[4] , \P3_S_C[4] , \P1_S_B[4] , n_1156,
           \X0_C[4] , \S3_A[4] , \S1_C[4] , \V0_C[1] , S3_D_0, \S1_C[0] ,
           \P1_S_B[1] , P3_RLC1_UC1, \S3_B[3] , \S3_A[0] , \P3_S_C[1] , \P1_B[3] ,
           P3_RLC0, \X0_C[1] , \P1_B[1] , \X0_C[3] , \S3_B[1] , \S3_A[2] ,
           \P3_S_C[3] , \S1_C[2] , \P1_S_B[3] , \V0_C[3] , n_353, \X0_A[0] ,
           \V0_A[4] , \X0_B[3] , \S1_B[2] , \P1_S_C[3] , \S1_A[1] , \S3_C[1] ,
           \X0_D[4] , \X0_A[4] , \V0_B[3] , \X0_D[0] , \P2_1_C[4] , \S1_B[4] ,
           \X0_D[2] , \V0_B[1] , \V0_A[2] , n_269, \S3_C[3] , \S1_A[3] ,
           \P1_S_C[1] , \S1_B[0] , P3_RLC1, \P2_1_C[2] , P3_RLC1_UC0, \X0_B[1] ,
           \X0_A[2] , n366, n367, n440, n441, n442, n443, n444, n445, n446, n447,
           n448, n449, n450, n451, n452, n453, n454, n455, n456, n457, n458, n459,
           n460, n461, n462, n463, n464, n465, n466, n467, n468, n469, n470, n471,
           n472, n473, n474, n475, n476, n477, n478;
      wire SYNOPSYS_UNCONNECTED_1 , SYNOPSYS_UNCONNECTED_2 ,
        SYNOPSYS_UNCONNECTED_3 , SYNOPSYS_UNCONNECTED_4 , SYNOPSYS_UNCONNECTED_5 ,
        SYNOPSYS_UNCONNECTED_6 , SYNOPSYS_UNCONNECTED_7 , SYNOPSYS_UNCONNECTED_8 ,
        SYNOPSYS_UNCONNECTED_9 , SYNOPSYS_UNCONNECTED_10 , SYNOPSYS_UNCONNECTED_11 ,
        SYNOPSYS_UNCONNECTED_12 , SYNOPSYS_UNCONNECTED_13 , SYNOPSYS_UNCONNECTED_14 ,
        SYNOPSYS_UNCONNECTED_15 , SYNOPSYS_UNCONNECTED_16 , SYNOPSYS_UNCONNECTED_17 ,
        SYNOPSYS_UNCONNECTED_18 , SYNOPSYS_UNCONNECTED_19 , SYNOPSYS_UNCONNECTED_20 ;
      assign CXD_P3_RLC_0[4] = 1'b1;
      assign CXD_P3_RLC_0[3] = 1'b0;
      assign CXD_P3_RLC_0[2] = 1'b0;
      assign CXD_P3_RLC_0[1] = 1'b0;
      assign CXD_P3_RLC_0[0] = 1'b1;
      assign CXD_P3_RLC_1[4] = 1'b1;
      assign CXD_P3_RLC_1[3] = 1'b0;
      assign CXD_P3_RLC_1[2] = 1'b0;
      assign CXD_P3_RLC_1[1] = 1'b0;
      assign CXD_P3_RLC_1[0] = 1'b1;
      assign CXD_P3_RLC1_UC0[4] = 1'b1;
```

```
assign CXD_P3_RLC1_UC0[3] = 1'b0;
assign CXD_P3_RLC1_UC0[2] = 1'b0;
assign CXD_P3_RLC1_UC0[1] = 1'b1;
assign CXD_P3_RLC1_UC0[0] = 1'b0;
assign CXD_P3_RLC1_UC1[4] = 1'b1;
assign CXD_P3_RLC1_UC1[3] = 1'b0;
assign CXD_P3_RLC1_UC1[2] = 1'b0;
assign CXD_P3_RLC1_UC1[1] = 1'b1;
assign CXD_P3_RLC1_UC1[0] = 1'b0;
PCF_CONTROL PCF_CONTROL_0 ( .ADDRA(ADDRA), .ADDRB(ADDRB), .WRITE(WRITE),
      .BP_INDEX(BP_INDEX), .START(START), .BP_NUM(BP_NUM), .RST(RST), .CLK(
      CLK) );
RLC1_UC1 RLC1_UC1_N ( .CXD_UC({CXD_P3_RLC1_UC1[5], SYNOPSYS_UNCONNECTED_1,
      SYNOPSYS_UNCONNECTED_2, SYNOPSYS_UNCONNECTED_3, SYNOPSYS_UNCONNECTED_4,
      SYNOPSYS_UNCONNECTED_5}), .CXD_P3_UC(P3_RLC1_UC1) );
MRC_COLUMN MRC_COLUMN0 ( .CXD_1(CXD_P2_1_MRC), .CXD_2(CXD_P2_2_MRC),
      .CXD_3(CXD_P2_3_MRC), .CXD_4(CXD_P2_4_MRC), .V0_C({\V0_C[4] ,
      \V0_C[3] , \V0_C[2] , \V0_C[1] }), .S1_B({\S1_B[4] , \S1_B[3] ,
      \S1_B[2] , \S1_B[1] , \S1_B[0] }), .S1_C({\S1_C[4] , \S1_C[3] , n367,
      \S1_C[1] , \S1_C[0] }), .S1_D({S1_D_[4], S1_D_[3], S1_D_[2], S1_D_[1],
      S1_D_0}), .S3_B({\S3_B[4] , \S3_B[3] , \S3_B[2] , \S3_B[1] , \S3_B[0]
      }), .S3_C({\S3_C[4] , \S3_C[3] , \S3_C[2] , \S3_C[1] , \S3_C[0] }),
      .S3_D({S3_D_[4], S3_D_[3], S3_D_[2], S3_D_[1], S3_D_0}), .P1_S_B({
      \P1_S_B[4] , \P1_S_B[3] , \P1_S_B[2] , \P1_S_B[1] }), .P1_S_C({
      \P1_S_C[4] , \P1_S_C[3] , \P1_S_C[2] , \P1_S_C[1] }), .P2_1_C({
      \P2_1_C[4] , \P2_1_C[3] , \P2_1_C[2] , \P2_1_C[1] }) );
RLC1_UC0 RLC1_UC0_N ( .CXD_UC({CXD_P3_RLC1_UC0[5], SYNOPSYS_UNCONNECTED_6,
      SYNOPSYS_UNCONNECTED_7, SYNOPSYS_UNCONNECTED_8, SYNOPSYS_UNCONNECTED_9,
      SYNOPSYS_UNCONNECTED_10}), .CXD_P3_UC(P3_RLC1_UC0) );
ZC_COLUMN_P1_P3 ZC_COLUMN_P1_P3_0 ( .CXD_1(CXD_P1_P3_1_ZC), .CXD_2(
      CXD_P1_P3_2_ZC), .CXD_3(CXD_P1_P3_3_ZC), .CXD_4(CXD_P1_P3_4_ZC),
      .V0_C({\V0_C[4] , \V0_C[3] , \V0_C[2] , \V0_C[1] }), .S1_B({\S1_B[4] ,
      \S1_B[3] , \S1_B[2] , \S1_B[1] , \S1_B[0] }), .S1_C({\S1_C[4] ,
      \S1_C[3] , n367, \S1_C[1] , \S1_C[0] }), .S1_D({S1_D_[4], S1_D_[3],
      S1_D_[2], S1_D_[1], S1_D_0}), .S3_B({\S3_B[4] , \S3_B[3] , \S3_B[2] ,
      \S3_B[1] , \S3_B[0] }), .S3_C({\S3_C[4] , \S3_C[3] , \S3_C[2] ,
      \S3_C[1] , \S3_C[0] }), .S3_D({S3_D_[4], S3_D_[3], S3_D_[2], S3_D_[1],
      S3_D_0}), .P1_S_B({\P1_S_B[4] , \P1_S_B[3] , \P1_S_B[2] , \P1_S_B[1] }
      ), .P1_S_C({\P1_S_C[4] , \P1_S_C[3] , \P1_S_C[2] , \P1_S_C[1] }),
```

```
    .P1_B_C(P1_B_C), .P3_S_C({\P3_S_C[3] , \P3_S_C[2] , \P3_S_C[1] }),
    .SUB_TYP(SUB_TYP) );
PCPG PCPG_0 ( .P1_S_C({\P1_S_C[4] , \P1_S_C[3] , \P1_S_C[2] , \P1_S_C[1] }
    ), .P2_C(P2_C), .P2_1_C({\P2_1_C[4] , \P2_1_C[3] , \P2_1_C[2] ,
    \P2_1_C[1] }), .P1_B_C(P1_B_C), .P3_C(P3_C), .P3_S_C({\P3_S_C[4] ,
    \P3_S_C[3] , \P3_S_C[2] , \P3_S_C[1] }), .P3_RLC0(P3_RLC0), .P3_RLC1(
    P3_RLC1), .P3_RLC1_UC1(P3_RLC1_UC1), .P3_RLC1_UC0(P3_RLC1_UC0), .S3_C(
    {\S3_C[4] , \S3_C[3] , \S3_C[2] , \S3_C[1] , \S3_C[0] }), .S1_C({
    \S1_C[4] , \S1_C[3] , n367, \S1_C[1] , \S1_C[0] }), .V0_C({\V0_C[4] ,
    \V0_C[3] , \V0_C[2] , \V0_C[1] }), .P1_B({\P1_B[4] , \P1_B[3] ,
    \P1_B[2] , \P1_B[1] }), .START(START), .RST(RST), .CLK(CLK) );
RLC_COLUMN_1 RLC_COLUMN_1_N ( .CXD_RLC({CXD_P3_RLC_1[5],
    SYNOPSYS_UNCONNECTED_11, SYNOPSYS_UNCONNECTED_12,
    SYNOPSYS_UNCONNECTED_13, SYNOPSYS_UNCONNECTED_14,
    SYNOPSYS_UNCONNECTED_15}), .CXD_P3_RLC(P3_RLC1) );
SC_COLUMN_P1_P3 SC_COLUMN_P1_P3_0 ( .CXD_1(CXD_P1_P3_1_SC), .CXD_2(
    CXD_P1_P3_2_SC), .CXD_3(CXD_P1_P3_3_SC), .CXD_4(CXD_P1_P3_4_SC),
    .X0_B({\X0_B[4] , \X0_B[3] , \X0_B[2] , \X0_B[1] , \X0_B[0] }), .X0_C(
    {\X0_C[4] , \X0_C[3] , \X0_C[2] , \X0_C[1] , \X0_C[0] }), .X0_D({
    \X0_D[4] , \X0_D[3] , \X0_D[2] , \X0_D[1] , \X0_D[0] }), .S1_B({
    \S1_B[4] , \S1_B[3] , \S1_B[2] , \S1_B[1] , \S1_B[0] }), .S1_C({
    \S1_C[4] , \S1_C[3] , n367, \S1_C[1] , \S1_C[0] }), .S1_D({S1_D_[4],
    S1_D_[3], S1_D_[2], S1_D_[1], S1_D_0}), .S3_B({1'b0, 1'b0, 1'b0, 1'b0,
    1'b0}), .S3_C({\S3_C[4] , \S3_C[3] , \S3_C[2] , \S3_C[1] , \S3_C[0] }),
    .S3_D({S3_D_[4], S3_D_[3], S3_D_[2], S3_D_[1], S3_D_0}), .P1_S_B({
    \P1_S_B[4] , \P1_S_B[3] , \P1_S_B[2] , \P1_S_B[1] }), .P1_S_C({
    \P1_S_C[4] , \P1_S_C[3] , \P1_S_C[2] , \P1_S_C[1] }), .P1_B_C(P1_B_C),
    .P3_S_C({\P3_S_C[4] , \P3_S_C[3] , \P3_S_C[2] , \P3_S_C[1] }) );
RLC_COLUMN_0 RLC_COLUMN_0_N ( .CXD_RLC({CXD_P3_RLC_0[5],
    SYNOPSYS_UNCONNECTED_16, SYNOPSYS_UNCONNECTED_17,
    SYNOPSYS_UNCONNECTED_18, SYNOPSYS_UNCONNECTED_19,
    SYNOPSYS_UNCONNECTED_20}), .CXD_P3_RLC(n_1156) );
P1_PRE_CHECK P1_PRE_CHECK ( .P1_B({\P1_B[4] , \P1_B[3] , \P1_B[2] ,
    \P1_B[1] }), .P1_S_B({\P1_S_B[4] , \P1_S_B[3] , \P1_S_B[2] ,
    \P1_S_B[1] }), .S1_A({\S1_A[4] , \S1_A[3] , \S1_A[2] , \S1_A[1] ,
    \S1_A[0] }), .S1_B({\S1_B[4] , \S1_B[3] , \S1_B[2] , \S1_B[1] ,
    \S1_B[0] }), .S1_C({\S1_C[4] , \S1_C[3] , \S1_C[2] , \S1_C[1] ,
    \S1_C[0] }), .S3_A({\S3_A[4] , \S3_A[3] , \S3_A[2] , \S3_A[1] ,
    \S3_A[0] }), .S3_B({\S3_B[4] , \S3_B[3] , \S3_B[2] , \S3_B[1] ,
```

```
      \S3_B[0] }), .V0_B({\V0_B[4] , \V0_B[3] , \V0_B[2] , \V0_B[1] }),
      .P1_S_C({\P1_S_C[4] , \P1_S_C[3] , \P1_S_C[2] , \P1_S_C[1] }) );
SDFFTRX1 S1_B_reg_3_ ( .SI(1'b0), .SE(START), .D(\S1_A[3] ), .CK(CLK),
      .RN(RST), .Q(\S1_B[3] ) );
SDFFTRX2 S1_C_reg_3_ ( .SI(1'b0), .SE(START), .D(\S1_B[3] ), .CK(CLK),
      .RN(RST), .Q(\S1_C[3] ) );
AND2X2 U25 ( .A(n463), .B(n464), .Y(S3_D_0) );
SDFFTRX1 S1_C_reg_4_ ( .SI(1'b0), .SE(START), .D(\S1_B[4] ), .CK(CLK),
      .RN(RST), .Q(\S1_C[4] ) );
SDFFTRX2 S1_B_reg_1_ ( .SI(1'b0), .SE(START), .D(\S1_A[1] ), .CK(CLK),
      .RN(RST), .Q(\S1_B[1] ) );
SDFFTRXL S1_D_reg_3_ ( .SI(1'b0), .SE(START), .D(\S1_C[3] ), .CK(CLK),
      .RN(RST), .Q(S1_D_[3]) );
SDFFTRXL S1_D_reg_2_ ( .SI(1'b0), .SE(START), .D(\S1_C[2] ), .CK(CLK),
      .RN(RST), .Q(S1_D_[2]) );
NAND2X1 U26 ( .A(RST), .B(n441), .Y(n440) );
NAND2X1 U27 ( .A(n473), .B(n474), .Y(n445) );
INVX2 U28 ( .A(n445), .Y(\V0_B[4] ) );
NAND2X1 U29 ( .A(n459), .B(n460), .Y(n444) );
INVX2 U30 ( .A(n444), .Y(\V0_B[3] ) );
NAND2X1 U31 ( .A(n471), .B(n472), .Y(n443) );
INVX2 U32 ( .A(n443), .Y(\V0_B[2] ) );
NAND2X1 U33 ( .A(n455), .B(n456), .Y(n442) );
INVX2 U34 ( .A(n442), .Y(\V0_B[1] ) );
NAND2X1 U35 ( .A(n467), .B(n468), .Y(n446) );
INVX2 U36 ( .A(n446), .Y(\S3_C[0] ) );
INVX2 U37 ( .A(START), .Y(n441) );
AND2X2 U38 ( .A(n465), .B(n466), .Y(S3_D_[1]) );
AND2X2 U39 ( .A(n451), .B(n452), .Y(S3_D_[2]) );
AND2X2 U40 ( .A(n475), .B(n476), .Y(S3_D_[3]) );
AND2X2 U41 ( .A(n457), .B(n458), .Y(S3_D_[4]) );
NAND2X1 U42 ( .A(n461), .B(n462), .Y(n447) );
NAND2X1 U43 ( .A(n477), .B(n478), .Y(n448) );
NAND2X1 U44 ( .A(n453), .B(n454), .Y(n449) );
NAND2X1 U45 ( .A(n469), .B(n470), .Y(n450) );
SDFFTRX2 V0_C_reg_1_ ( .SI(1'b0), .SE(n442), .D(RST), .CK(CLK), .RN(n441),
      .Q(\V0_C[1] ) );
INVX2 U46 ( .A(P3_RLC0), .Y(n_1156) );
INVX2 U47 ( .A(n450), .Y(\S3_C[4] ) );
```

```
INVX2 U48 ( .A(n449), .Y(\S3_C[3] ) );
INVX2 U49 ( .A(n448), .Y(\S3_C[2] ) );
INVX2 U50 ( .A(n447), .Y(\S3_C[1] ) );
SDFFTRX2 S1_C_reg_1_ ( .SI(1'b0), .SE(START), .D(\S1_B[1] ), .CK(CLK),
      .RN(RST), .Q(\S1_C[1] ) );
SDFFTRX1 X0_A_reg_4_ ( .SI(1'b0), .SE(START), .D(X0_A_4_0[4]), .CK(CLK),
      .RN(RST), .Q(\X0_A[4] ) );
SDFFTRX1 X0_A_reg_3_ ( .SI(1'b0), .SE(START), .D(X0_A_4_0[3]), .CK(CLK),
      .RN(RST), .Q(\X0_A[3] ) );
SDFFTRX1 X0_A_reg_2_ ( .SI(1'b0), .SE(START), .D(X0_A_4_0[2]), .CK(CLK),
      .RN(RST), .Q(\X0_A[2] ) );
SDFFTRX1 X0_A_reg_1_ ( .SI(1'b0), .SE(START), .D(X0_A_4_0[1]), .CK(CLK),
      .RN(RST), .Q(\X0_A[1] ) );
SDFFTRX1 X0_A_reg_0_ ( .SI(1'b0), .SE(START), .D(X0_A_4_0[0]), .CK(CLK),
      .RN(RST), .Q(\X0_A[0] ) );
EDFFX1 V0_B_reg_4_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n473) );
EDFFX1 V0_B_reg_3_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n459) );
EDFFX1 V0_B_reg_2_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n471) );
EDFFX1 V0_B_reg_1_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n455) );
EDFFX1 S3_C_reg_4_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n469) );
EDFFX1 S3_C_reg_3_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n453) );
EDFFX1 S3_C_reg_2_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n477) );
EDFFX1 S3_C_reg_1_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n461) );
EDFFX1 S3_C_reg_0_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n467) );
EDFFX1 S3_D_reg_4_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n457) );
EDFFX1 S3_D_reg_3_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n475) );
EDFFX1 S3_D_reg_2_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n451) );
EDFFX1 S3_D_reg_1_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n465) );
EDFFX1 S3_D_reg_0_ ( .D(1'b0), .CK(CLK), .E(n440), .Q(n463) );
EDFFX1 V0_A_reg_4_ ( .D(V0_A_4_1[4]), .CK(CLK), .E(n_353), .Q(\V0_A[4] )
      );
EDFFX1 V0_A_reg_3_ ( .D(V0_A_4_1[3]), .CK(CLK), .E(n_353), .Q(\V0_A[3] )
      );
EDFFX1 V0_A_reg_2_ ( .D(V0_A_4_1[2]), .CK(CLK), .E(n_353), .Q(\V0_A[2] )
      );
EDFFX1 V0_A_reg_1_ ( .D(V0_A_4_1[1]), .CK(CLK), .E(n_353), .Q(\V0_A[1] )
      );
EDFFX1 S1_A_reg_4_ ( .D(S1_A_4_0[4]), .CK(CLK), .E(n_353), .Q(\S1_A[4] )
      );
```

```
EDFFX1 S1_A_reg_3_ ( .D(S1_A_4_0[3]), .CK(CLK), .E(n_353), .Q(\S1_A[3] )
        );
EDFFX1 S1_A_reg_1_ ( .D(S1_A_4_0[1]), .CK(CLK), .E(n_353), .Q(\S1_A[1] )
        );
EDFFX1 S1_A_reg_0_ ( .D(S1_A_4_0[0]), .CK(CLK), .E(n_353), .Q(\S1_A[0] )
        );
EDFFX1 S3_A_reg_4_ ( .D(S3_A_4_0[4]), .CK(CLK), .E(n_353), .Q(\S3_A[4] )
        );
EDFFX1 S3_A_reg_3_ ( .D(S3_A_4_0[3]), .CK(CLK), .E(n_353), .Q(\S3_A[3] )
        );
EDFFX1 S3_A_reg_1_ ( .D(S3_A_4_0[1]), .CK(CLK), .E(n_353), .Q(\S3_A[1] )
        );
EDFFX1 S3_A_reg_0_ ( .D(S3_A_4_0[0]), .CK(CLK), .E(n_353), .Q(\S3_A[0] )
        );
SDFFTRX1 X0_B_reg_4_ ( .SI(1'b0), .SE(START), .D(\X0_A[4] ), .CK(CLK),
      .RN(RST), .Q(\X0_B[4] ) );
SDFFTRX1 X0_B_reg_3_ ( .SI(1'b0), .SE(START), .D(\X0_A[3] ), .CK(CLK),
      .RN(RST), .Q(\X0_B[3] ) );
SDFFTRX1 X0_B_reg_2_ ( .SI(1'b0), .SE(START), .D(\X0_A[2] ), .CK(CLK),
      .RN(RST), .Q(\X0_B[2] ) );
SDFFTRX1 X0_B_reg_1_ ( .SI(1'b0), .SE(START), .D(\X0_A[1] ), .CK(CLK),
      .RN(RST), .Q(\X0_B[1] ) );
SDFFTRX1 X0_B_reg_0_ ( .SI(1'b0), .SE(START), .D(\X0_A[0] ), .CK(CLK),
      .RN(RST), .Q(\X0_B[0] ) );
SDFFTRX1 X0_C_reg_4_ ( .SI(1'b0), .SE(START), .D(\X0_B[4] ), .CK(CLK),
      .RN(RST), .Q(\X0_C[4] ) );
SDFFTRX1 X0_C_reg_3_ ( .SI(1'b0), .SE(START), .D(\X0_B[3] ), .CK(CLK),
      .RN(RST), .Q(\X0_C[3] ) );
SDFFTRX1 X0_C_reg_2_ ( .SI(1'b0), .SE(START), .D(\X0_B[2] ), .CK(CLK),
      .RN(RST), .Q(\X0_C[2] ) );
SDFFTRX1 X0_C_reg_1_ ( .SI(1'b0), .SE(START), .D(\X0_B[1] ), .CK(CLK),
      .RN(RST), .Q(\X0_C[1] ) );
SDFFTRX1 X0_C_reg_0_ ( .SI(1'b0), .SE(START), .D(\X0_B[0] ), .CK(CLK),
      .RN(RST), .Q(\X0_C[0] ) );
SDFFTRX1 X0_D_reg_4_ ( .SI(1'b0), .SE(START), .D(\X0_C[4] ), .CK(CLK),
      .RN(RST), .Q(\X0_D[4] ) );
SDFFTRX1 X0_D_reg_3_ ( .SI(1'b0), .SE(START), .D(\X0_C[3] ), .CK(CLK),
      .RN(RST), .Q(\X0_D[3] ) );
SDFFTRX1 X0_D_reg_2_ ( .SI(1'b0), .SE(START), .D(\X0_C[2] ), .CK(CLK),
```

```
    .RN(RST), .Q(\X0_D[2] ) );
SDFFTRX1 X0_D_reg_1_ ( .SI(1'b0), .SE(START), .D(\X0_C[1] ), .CK(CLK),
    .RN(RST), .Q(\X0_D[1] ) );
SDFFTRX1 X0_D_reg_0_ ( .SI(1'b0), .SE(START), .D(\X0_C[0] ), .CK(CLK),
    .RN(RST), .Q(\X0_D[0] ) );
SDFFTRX1 V0_B_reg2_4_ ( .SI(1'b0), .SE(START), .D(\V0_A[4] ), .CK(CLK),
    .RN(RST), .Q(n474) );
SDFFTRX1 V0_B_reg2_3_ ( .SI(1'b0), .SE(START), .D(\V0_A[3] ), .CK(CLK),
    .RN(RST), .Q(n460) );
SDFFTRX1 V0_B_reg2_2_ ( .SI(1'b0), .SE(START), .D(\V0_A[2] ), .CK(CLK),
    .RN(RST), .Q(n472) );
SDFFTRX1 V0_B_reg2_1_ ( .SI(1'b0), .SE(START), .D(\V0_A[1] ), .CK(CLK),
    .RN(RST), .Q(n456) );
SDFFTRX1 V0_C_reg_4_ ( .SI(1'b0), .SE(n445), .D(RST), .CK(CLK), .RN(n441),
    .Q(\V0_C[4] ) );
SDFFTRX1 V0_C_reg_3_ ( .SI(1'b0), .SE(n444), .D(RST), .CK(CLK), .RN(n441),
    .Q(\V0_C[3] ) );
SDFFTRX2 V0_C_reg_2_ ( .SI(1'b0), .SE(n443), .D(RST), .CK(CLK), .RN(n441),
    .Q(\V0_C[2] ) );
SDFFTRX1 S1_B_reg_4_ ( .SI(1'b0), .SE(START), .D(\S1_A[4] ), .CK(CLK),
    .RN(RST), .Q(\S1_B[4] ) );
SDFFTRX2 S1_B_reg_0_ ( .SI(1'b0), .SE(START), .D(\S1_A[0] ), .CK(CLK),
    .RN(RST), .Q(\S1_B[0] ) );
SDFFTRX2 S1_C_reg_0_ ( .SI(1'b0), .SE(START), .D(\S1_B[0] ), .CK(CLK),
    .RN(RST), .Q(\S1_C[0] ) );
SDFFTRX1 S1_D_reg_4_ ( .SI(1'b0), .SE(START), .D(\S1_C[4] ), .CK(CLK),
    .RN(RST), .Q(S1_D_[4]) );
SDFFTRXL S1_D_reg_1_ ( .SI(1'b0), .SE(START), .D(\S1_C[1] ), .CK(CLK),
    .RN(RST), .Q(S1_D_[1]) );
SDFFTRX1 S1_D_reg_0_ ( .SI(1'b0), .SE(START), .D(\S1_C[0] ), .CK(CLK),
    .RN(RST), .Q(S1_D_0) );
SDFFTRX1 S3_B_reg_4_ ( .SI(1'b0), .SE(START), .D(\S3_A[4] ), .CK(CLK),
    .RN(RST), .Q(\S3_B[4] ) );
SDFFTRXL S3_B_reg_3_ ( .SI(1'b0), .SE(START), .D(\S3_A[3] ), .CK(CLK),
    .RN(RST), .Q(\S3_B[3] ) );
SDFFTRX1 S3_B_reg_1_ ( .SI(1'b0), .SE(START), .D(\S3_A[1] ), .CK(CLK),
    .RN(RST), .Q(\S3_B[1] ) );
SDFFTRX1 S3_B_reg_0_ ( .SI(1'b0), .SE(START), .D(\S3_A[0] ), .CK(CLK),
    .RN(RST), .Q(\S3_B[0] ) );
```

```
    SDFFTRX1 S3_C_reg2_4_ ( .SI(1'b0), .SE(START), .D(\S3_B[4] ), .CK(CLK),
        .RN(RST), .Q(n470) );
    SDFFTRX1 S3_C_reg2_3_ ( .SI(1'b0), .SE(START), .D(\S3_B[3] ), .CK(CLK),
        .RN(RST), .Q(n454) );
    SDFFTRX1 S3_C_reg2_2_ ( .SI(1'b0), .SE(START), .D(\S3_B[2] ), .CK(CLK),
        .RN(RST), .Q(n478) );
    SDFFTRX1 S3_C_reg2_1_ ( .SI(1'b0), .SE(START), .D(\S3_B[1] ), .CK(CLK),
        .RN(RST), .Q(n462) );
    SDFFTRX1 S3_C_reg2_0_ ( .SI(1'b0), .SE(START), .D(\S3_B[0] ), .CK(CLK),
        .RN(RST), .Q(n468) );
    SDFFTRX1 S3_D_reg2_4_ ( .SI(1'b0), .SE(n450), .D(RST), .CK(CLK), .RN(n441),
        .Q(n458) );
    SDFFTRX1 S3_D_reg2_3_ ( .SI(1'b0), .SE(n449), .D(RST), .CK(CLK), .RN(n441),
        .Q(n476) );
    SDFFTRX1 S3_D_reg2_2_ ( .SI(1'b0), .SE(n448), .D(RST), .CK(CLK), .RN(n441),
        .Q(n452) );
    SDFFTRX1 S3_D_reg2_1_ ( .SI(1'b0), .SE(n447), .D(RST), .CK(CLK), .RN(n441),
        .Q(n466) );
    SDFFTRX1 S3_D_reg2_0_ ( .SI(1'b0), .SE(n446), .D(RST), .CK(CLK), .RN(n441),
        .Q(n464) );
    CLKBUFX4 U51 ( .A(n440), .Y(n_269) );
    INVX2 U52 ( .A(n_269), .Y(n_353) );
    INVX2 U53 ( .A(n366), .Y(n367) );
    EDFFX4 S1_A_reg_2_ ( .D(S1_A_4_0[2]), .CK(CLK), .E(n_353), .Q(\S1_A[2] )
         );
    EDFFX4 S3_A_reg_2_ ( .D(S3_A_4_0[2]), .CK(CLK), .E(n_353), .Q(\S3_A[2] )
         );
    SDFFTRX4 S1_B_reg_2_ ( .SI(1'b0), .SE(START), .D(\S1_A[2] ), .CK(CLK),
        .RN(RST), .Q(\S1_B[2] ) );
    SDFFTRX4 S1_C_reg_2_ ( .SI(1'b0), .SE(START), .D(\S1_B[2] ), .CK(CLK),
        .RN(RST), .Q(\S1_C[2] ), .QN(n366) );
    SDFFTRX4 S3_B_reg_2_ ( .SI(1'b0), .SE(START), .D(\S3_A[2] ), .CK(CLK),
        .RN(RST), .Q(\S3_B[2] ) );
endmodule
```

A.9 Cell List

```
****************************************
Report : cell
Design : PCF
```

Version: 2000.05-1

Date : Tue Oct 31 21:44:34 2006

****

Attributes:

b - black box (unknown)

h - hierarchical

n - noncombinational

r - removable

u - contains unmapped logic

Cell	Reference	Library	Area	Attributes
MRC_COLUMN0	MRC_COLUMN		794.88	h
P1_PRE_CHECK	P1_PRE_CHECK		1123.20	h
PCF_CONTROL_0	PCF_CONTROL		6266.88	h, n
PCPG_0	PCPG		1595.52	h, n
S1_A_reg_0_	EDFFX1	typical	126.72	n
S1_A_reg_1_	EDFFX1	typical	126.72	n
S1_A_reg_2_	EDFFX4	typical	184.32	n
S1_A_reg_3_	EDFFX1	typical	126.72	n
S1_A_reg_4_	EDFFX1	typical	126.72	n
S1_B_reg_0_	SDFFTRX2	typical	149.76	n
S1_B_reg_1_	SDFFTRX2	typical	149.76	n
S1_B_reg_2_	SDFFTRX4	typical	184.32	n
S1_B_reg_3_	SDFFTRX1	typical	126.72	n
S1_B_reg_4_	SDFFTRX1	typical	126.72	n
S1_C_reg_0_	SDFFTRX2	typical	149.76	n
S1_C_reg_1_	SDFFTRX2	typical	149.76	n
S1_C_reg_2_	SDFFTRX4	typical	184.32	n
S1_C_reg_3_	SDFFTRX2	typical	149.76	n
S1_C_reg_4_	SDFFTRX1	typical	126.72	n
S1_D_reg_0_	SDFFTRX1	typical	126.72	n
S1_D_reg_1_	SDFFTRXL	typical	126.72	n
S1_D_reg_2_	SDFFTRXL	typical	126.72	n
S1_D_reg_3_	SDFFTRXL	typical	126.72	n
S1_D_reg_4_	SDFFTRX1	typical	126.72	n
S3_A_reg_0_	EDFFX1	typical	126.72	n
S3_A_reg_1_	EDFFX1	typical	126.72	n
S3_A_reg_2_	EDFFX4	typical	184.32	n
S3_A_reg_3_	EDFFX1	typical	126.72	n

S3_A_reg_4_	EDFFX1	typical	126.72	n
S3_B_reg_0_	SDFFTRX1	typical	126.72	n
S3_B_reg_1_	SDFFTRX1	typical	126.72	n
S3_B_reg_2_	SDFFTRX4	typical	184.32	n
S3_B_reg_3_	SDFFTRXL	typical	126.72	n
S3_B_reg_4_	SDFFTRX1	typical	126.72	n
S3_C_reg2_0_	SDFFTRX1	typical	126.72	n
S3_C_reg2_1_	SDFFTRX1	typical	126.72	n
S3_C_reg2_2_	SDFFTRX1	typical	126.72	n
S3_C_reg2_3_	SDFFTRX1	typical	126.72	n
S3_C_reg2_4_	SDFFTRX1	typical	126.72	n
S3_C_reg_0_	EDFFX1	typical	126.72	n
S3_C_reg_1_	EDFFX1	typical	126.72	n
S3_C_reg_2_	EDFFX1	typical	126.72	n
S3_C_reg_3_	EDFFX1	typical	126.72	n
S3_C_reg_4_	EDFFX1	typical	126.72	n
S3_D_reg2_0_	SDFFTRX1	typical	126.72	n
S3_D_reg2_1_	SDFFTRX1	typical	126.72	n
S3_D_reg2_2_	SDFFTRX1	typical	126.72	n
S3_D_reg2_3_	SDFFTRX1	typical	126.72	n
S3_D_reg2_4_	SDFFTRX1	typical	126.72	n
S3_D_reg_0_	EDFFX1	typical	126.72	n
S3_D_reg_1_	EDFFX1	typical	126.72	n
S3_D_reg_2_	EDFFX1	typical	126.72	n
S3_D_reg_3_	EDFFX1	typical	126.72	n
S3_D_reg_4_	EDFFX1	typical	126.72	n
SC_COLUMN_P1_P3_0	SC_COLUMN_P1_P3		3697.92	h
U25	AND2X2	typical	23.04	
U26	NAND2X1	typical	17.28	
U27	NAND2X1	typical	17.28	
U28	INVX2	typical	11.52	
U29	NAND2X1	typical	17.28	
U30	INVX2	typical	11.52	
U31	NAND2X1	typical	17.28	
U32	INVX2	typical	11.52	
U33	NAND2X1	typical	17.28	
U34	INVX2	typical	11.52	
U35	NAND2X1	typical	17.28	
U36	INVX2	typical	11.52	

U37	INVX2	typical	11.52	
U38	AND2X2	typical	23.04	
U39	AND2X2	typical	23.04	
U40	AND2X2	typical	23.04	
U41	AND2X2	typical	23.04	
U42	NAND2X1	typical	17.28	
U43	NAND2X1	typical	17.28	
U44	NAND2X1	typical	17.28	
U45	NAND2X1	typical	17.28	
U46	INVX2	typical	11.52	
U47	INVX2	typical	11.52	
U48	INVX2	typical	11.52	
U49	INVX2	typical	11.52	
U50	INVX2	typical	11.52	
U51	CLKBUFX4	typical	23.04	
U52	INVX2	typical	11.52	
U53	INVX2	typical	11.52	
V0_A_reg_1_	EDFFX1	typical	126.72	n
V0_A_reg_2_	EDFFX1	typical	126.72	n
V0_A_reg_3_	EDFFX1	typical	126.72	n
V0_A_reg_4_	EDFFX1	typical	126.72	n
V0_B_reg2_1_	SDFFTRX1	typical	126.72	n
V0_B_reg2_2_	SDFFTRX1	typical	126.72	n
V0_B_reg2_3_	SDFFTRX1	typical	126.72	n
V0_B_reg2_4_	SDFFTRX1	typical	126.72	n
V0_B_reg_1_	EDFFX1	typical	126.72	n
V0_B_reg_2_	EDFFX1	typical	126.72	n
V0_B_reg_3_	EDFFX1	typical	126.72	n
V0_B_reg_4_	EDFFX1	typical	126.72	n
V0_C_reg_1_	SDFFTRX2	typical	149.76	n
V0_C_reg_2_	SDFFTRX2	typical	149.76	n
V0_C_reg_3_	SDFFTRX1	typical	126.72	n
V0_C_reg_4_	SDFFTRX1	typical	126.72	n
X0_A_reg_0_	SDFFTRX1	typical	126.72	n
X0_A_reg_1_	SDFFTRX1	typical	126.72	n
X0_A_reg_2_	SDFFTRX1	typical	126.72	n
X0_A_reg_3_	SDFFTRX1	typical	126.72	n
X0_A_reg_4_	SDFFTRX1	typical	126.72	n
X0_B_reg_0_	SDFFTRX1	typical	126.72	n

X0_B_reg_1_	SDFFTRX1	typical	126.72	n
X0_B_reg_2_	SDFFTRX1	typical	126.72	n
X0_B_reg_3_	SDFFTRX1	typical	126.72	n
X0_B_reg_4_	SDFFTRX1	typical	126.72	n
X0_C_reg_0_	SDFFTRX1	typical	126.72	n
X0_C_reg_1_	SDFFTRX1	typical	126.72	n
X0_C_reg_2_	SDFFTRX1	typical	126.72	n
X0_C_reg_3_	SDFFTRX1	typical	126.72	n
X0_C_reg_4_	SDFFTRX1	typical	126.72	n
X0_D_reg_0_	SDFFTRX1	typical	126.72	n
X0_D_reg_1_	SDFFTRX1	typical	126.72	n
X0_D_reg_2_	SDFFTRX1	typical	126.72	n
X0_D_reg_3_	SDFFTRX1	typical	126.72	n
X0_D_reg_4_	SDFFTRX1	typical	126.72	n
ZC_COLUMN_P1_P3_0	ZC_COLUMN_P1_P3		14751.36	h, n

Total 125 cells 40037.76

A.10 Hardware Area Size

****

Report : area

Design : PCF

Version: 2000.05-1

Date : Tue Oct 31 21:44:34 2006

****

Library(s) Used:

typical (File: C:/synopsys/msvc50/syn/bin/typical.db)

Number of ports: 169

Number of nets: 378

Number of cells: 125

Number of references: 20

Combinational area: 25200.000000

Noncombinational area: 14837.760742

Total cell area: 40037.760742