EURASIP Journal on Applied Signal Processing 2003:13, 1355–1370
© 2003 Hindawi Publishing Corporation

Low-Power Embedded DSP Core for Communication Systems

Ya-Lan Tsao
Department of Electrical Engineering, National Central University, 300 Jung-Da Road, Jung-Li City, Taoyuan 320, Taiwan
Email: alan@ee.ncu.edu.tw

Wei-Hao Chen
Department of Electrical Engineering, National Central University, 300 Jung-Da Road, Jung-Li City, Taoyuan 320, Taiwan
Email: ee021053@ee.ncu.edu.tw

Ming Hsuan Tan
Department of Electrical Engineering, National Central University, 300 Jung-Da Road, Jung-Li City, Taoyuan 320, Taiwan
Email: ee892012@ee.ncu.edu.tw

Maw-Ching Lin
Department of Electrical Engineering, National Central University, 300 Jung-Da Road, Jung-Li City, Taoyuan 320, Taiwan
Email: mclin@ee.ncu.edu.tw

Shyh-Jye Jou
Department of Electrical Engineering, National Central University, 300 Jung-Da Road, Jung-Li City, Taoyuan 320, Taiwan
Email: jerry@ee.ncu.edu.tw

Received 2 February 2003 and in revised form 14 July 2003

This paper proposes a parameterized digital signal processor (DSP) core for an embedded digital signal processing system designed to achieve demodulation/synchronization with better performance and flexibility. The features of this DSP core include parameterized data path, dual MAC unit, subword MAC, and optional function-specific blocks for accelerating communication system modulation operations. This DSP core also has a low-power structure, which includes the gray-code addressing mode, pipeline sharing, and advanced hardware looping. Users can select the parameters and special functional blocks based on the character of their applications and then generating a DSP core. The DSP core has been implemented via a cell-based design method using a synthesizable Verilog code with TSMC 0.35 μm SPQM and 0.25 μm 1P5M library. The equivalent gate count of the core area without memory is approximately 50 k. Moreover, the maximum operating frequency of a 16×16 version is 100 MHz (0.35 μm) and 140 MHz (0.25 μm).

Keywords and phrases: digital signal processor, embedded system, dual MAC, subword multiplier.

1. INTRODUCTION

During the past few years, digital signal processor (DSP) has become the fastest growing segment in the processor industry [1]. Today, almost all wireless handsets and base stations are DSP-based systems. Not only technological trends make DSP cheaper and more powerful, but DSP-based systems are also more cost effective and have shorter time to market than other systems [2].

Some DSPs can achieve high throughput by exploiting parallelism with specialized data paths at moderate clock frequency. For example, very long instruction word (VLIW) and single instruction multiple data (SIMD) approaches can be used to further enhance processor performance [3]. However, these approaches are not economical for dedicated application in area and power terms. Consequently, these structures are not suitable for embedded communication applications, in which small area and low-power consumption are critical factors. Instead, an application-specific concept is used while maintaining a focus on the targeted application of the processor. Accordingly, the DSP architecture and bus structure have been set to optimize

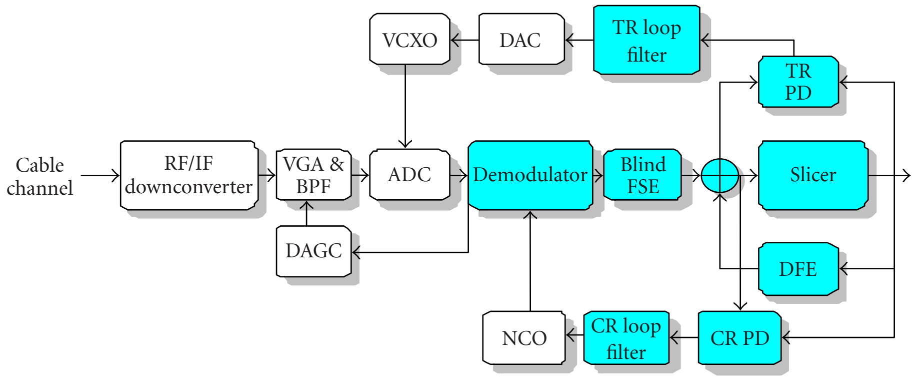


Figure 1: Typical block diagram of the demodulation and synchronization in the receiver of communication system.

the performance of DSP processors for the target applications. Some special function blocks also influence the performance of application-specific DSPs. Notably, special functional blocks such as square-distance-and-accumulate for vector quantization, add-compare-select for the Viterbi algorithm, and the Galois field operation for forward error-control coding are provided in certain DSPs for baseband operations [4, 5, 6, 7, 8]. For example, Lucent's DSP 1618 performs Viterbi decoding using a coprocessor, which supports various decoding modes with control registers [5]. A special function, called the mobile communication accelerator (MCA), is incorporated into the design of MDSP-II to accelerate the complex MAC operation [8].

Consequently, combining a dedicated, high performance DSP core with some special functional blocks to produce a highly integrated system is a current trend [9, 10, 11, 12]. The proposed design is parameterized and configurable and thus can meet system requirements easily. The proposed DSP core contains special blocks such as Hamming distance unit, subword multiplier, dual MAC unit, rounded/saturation mode, fixed-coefficient FIR filter, and slicer unit. The proposed DSP core is designed to support the calculations in the demodulation/synchronization part of the receiver. Figure 1 illustrates the typical block diagram of the demodulation and synchronization in the receiver. Thus, this DSP core supports operations such as scaling, digital FIR filtering (both fixed-coefficient filter for pulse shaping and adaptive filter for equalization), symbol slicing, looping, complex multiplication, and so on.

In the aspect of low-power design, the memory access operation is clearly the most power-consuming action in DSPs. Various low-power techniques are also used in the DSP developed here, including gray-code addressing and advanced hardware looping; pipeline sharing and low-power data-path design are used to reduce power consumption. The remainder of this paper is organized as follows: Section 2 presents the architecture of the proposed DSP. Section 3 then shows the design of the parameterized architecture and the special functional blocks. Next, Section 4 discusses some low-power design techniques used in this DSP core. Subsequently, implementation and design results are demonstrated in Section 5. Finally, Section 6 makes some conclusions.

2. ARCHITECTURE OF THE DSP CORE

Figure 2 illustrates the overall architecture of the proposed NCU_DSP [9]. The NCU_DSP is a fixed-point DSP core. The grey blocks in Figure 2 are the special functional blocks and are optional blocks that can be chosen by the user. The DSP processor core itself is parameterized with several independent parameters. Users can set the parameters so that the DSP core fits the applications.

2.1. Bus and memory architecture

One of the characteristics of the DSP processor is that it can move large amounts of data to or from memory rapidly and efficiently. DSP processor has this characteristic because it needs to process numerous calculations simultaneously. Taking FIR as an example, one tap operation must make three accesses to memory, namely, coefficient access, data access, and write-back data. If the memory bandwidth is not wide enough, an operation must be split into several suboperations before it can be completed. Consequently, memory architecture is an important determinant of processor performance.

Figure 3 illustrates the modified Harvard architecture used in our work. The modified architecture contains one program-memory bank and one data-memory bank with separate program and data bus. The program and data memories are single-port and dual-port RAM, respectively. The dual-port RAM indicates that the DSP processor simultaneously can make two accesses to RAM. This arrangement provides a maximum of one program access and two data accesses per instruction cycle to enhance memory access capacity.

Most of the DSP processors include one or more dedicated data-address generation units (DAGU) for calculating data address. NCU_DSP supports three addressing modes, namely, the indirect addressing, register direct addressing, and immediate addressing modes, as listed in Table 1. The indirect addressing mode requires one additional register

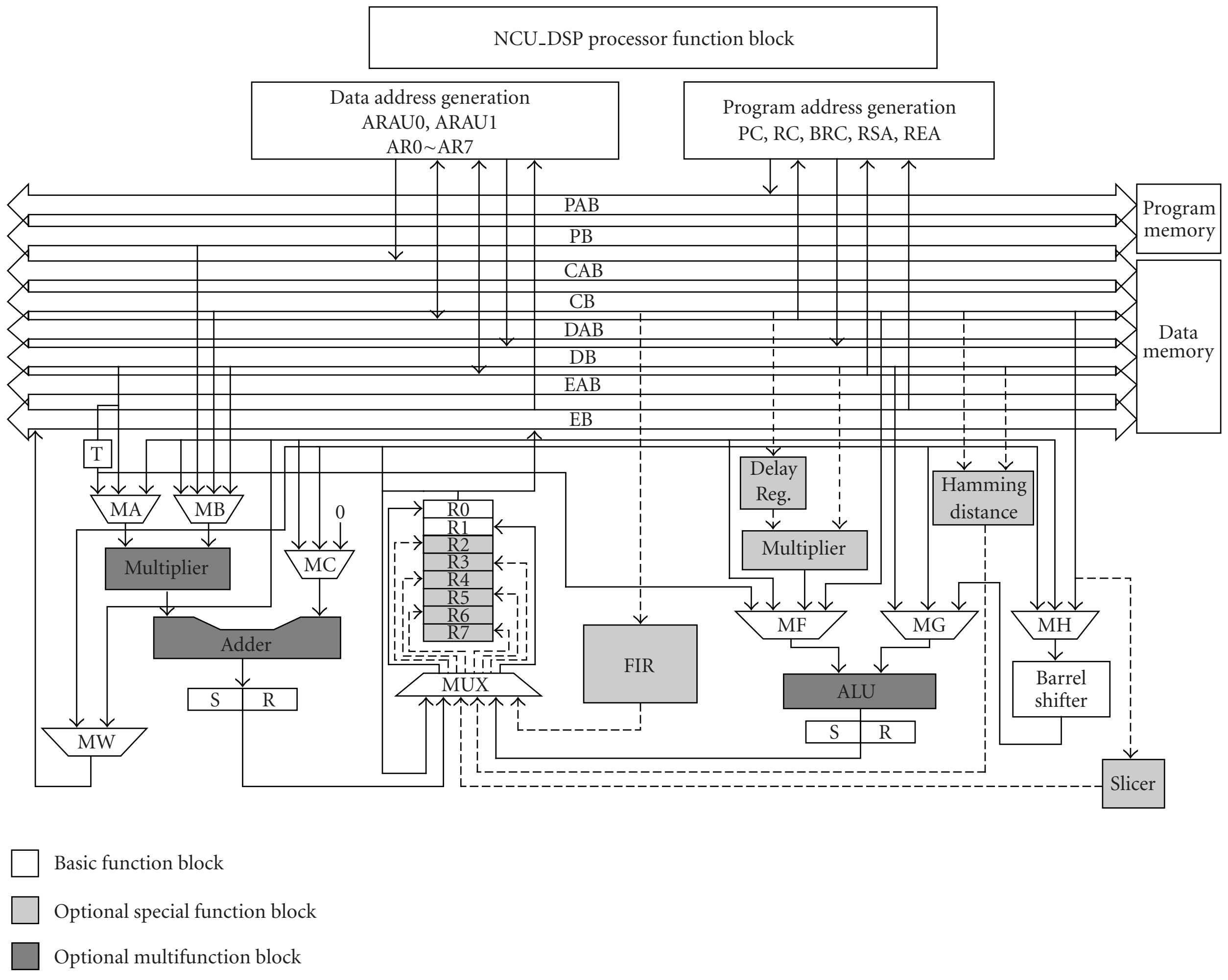


Figure 2: The block diagram of NCU_DSP.

file, called the auxiliary register (ARx), for storing data-memory address. Moreover, DSP processors usually need to access data using special addressing methods in many DSP algorithms. Hence, NCU_DSP supports linear addressing, circular addressing, and bit-reversed address in the indirect addressing mode. The circular addressing mode can be used to operate the FIR filter, and convolution and correlation algorithm, while the FFT algorithm uses bit-reversed addressing. These specialized functions not only reduce the programming burden but also enhance the performance of DSP under conditions of smooth data access. This enhanced performance is why the indirect addressing mode is the most important addressing mode in the DSP cores.

Figure 4a shows the straightforward method for calculating the bit-reversed addressing value. In Figure 4a, "A" represents the current address pointer value and "Step" represents the offset value, which is added to or subtracted from the current pointer value. The internal carry propagation is from MSB to LSB, differing from normal addition. Notably, the bit-reversed address is calculated by adding or subtracting the step value from MSB to LSB (if the step is +1, the address value will be 0000, 1000, 0100, 1100, 0010, 1010, . . .). This circuit in Figure 4a uses a ripple adder to construct the reversed carry propagation from MSB to LSB. However, the circuit has n full-adder (FA) delay time. This delay time of ripple adder makes the instruction decode (ID) stage become the critical path of DSP core. Figure 4b illustrates the new bit-reversed addressing generation architecture. In Figure 4b, "A" and "Step" are reordered by reversed connecting. The ripple adder is replaced by a parallel adder which has less delay time with respect to ripple adder. Finally, the output of the parallel adder is the reversed order of the bit-reversed value. The proposed new structure, Figure 4b, has smaller delay time than that of Figure 4a.

2.2. *I/O interface*

Required transmission methods differ with data type. The I/O interface of NCU_DSP contains three categories, the direct data access (DMA) mode, the handshaking mode, and

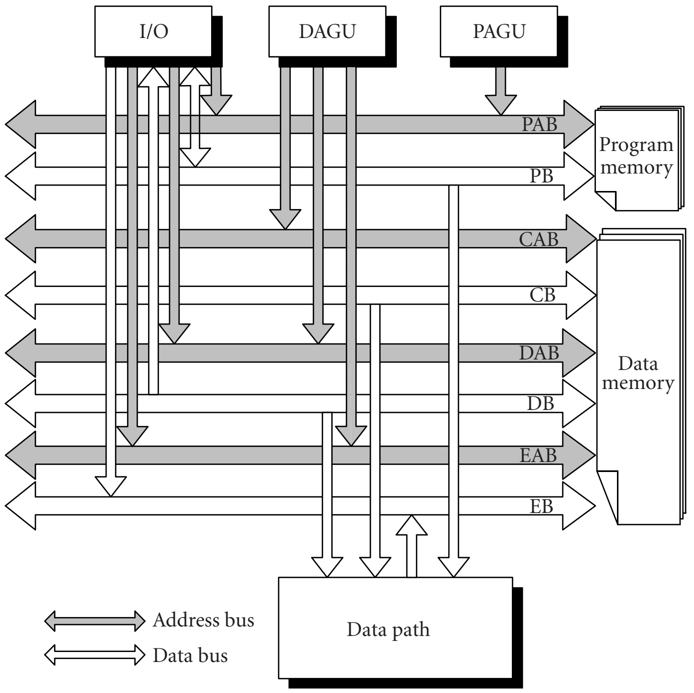


Figure 3: The bus architecture of NCU_DSP.

Table 1: Data addressing modes.

Type	Operation Syntax	Function	Description
Indirect addressing mode	*ARx	Address = ARx	ARx contains the data-memory address.

	*ARx±	Address = ARx
		ARx = ARx±
			After access, the address in ARx is incremented or decremented by 1.

	*ARx ± 0B	Address = ARx
		ARx = B(ARx ± AR0)
			After access, AR0 is added to or subtracted from ARx with reverse carry propagation.

	*ARx ± 0	Address = ARx
		ARx = ARx ± AR0
			After access, AR0 is added to or subtracted from ARx.

	*ARx ± 0%	Address = ARx
		ARx = circ(ARx ± AR0)
			After access, AR0 is added to or subtracted from ARx with circular addressing.

Register direct addressing mode	ADD R0, R1, R2	R2 = R0 + R1	Access the content of register as operand directly.
Immediate addressing mode	LAR # 1000 h AR0	AR0 = 1000 h	Give the destination register or memory a value directly.

the merge mode. The DMA mode is to transfer data directly from the outside of the DSP to the data memory of the DSP core. The DMA mode is provided for transferring these data quickly and conveniently. Notably, the DMA mode transfer batch data. The transfer rate is the same with the clock in the DSP core. The handshaking mode is for real-time data but the data rate is not regular. The handshaking signals are required to perform the data transfer in this mode. The merge mode is to transfer data in regular clock rate which is slower than the internal clock of DSP core. In DMA mode, the DSP core is halt until the data transfer is finished. The DSP core is running when data are transferred in merge mode and handshaking mode. Notably, the data transfer in the handshaking and merge modes occurs between the data outside the NCU_DSP core and the host programmable interface (HPI) memory. The HPI memory resembles a buffer of data memory.

2.3. *Pipeline stage*

The NCU-DSP contains six pipeline stages, namely, instruction fetch (IF) stage, ID stage, operand fetch (OP) stage,

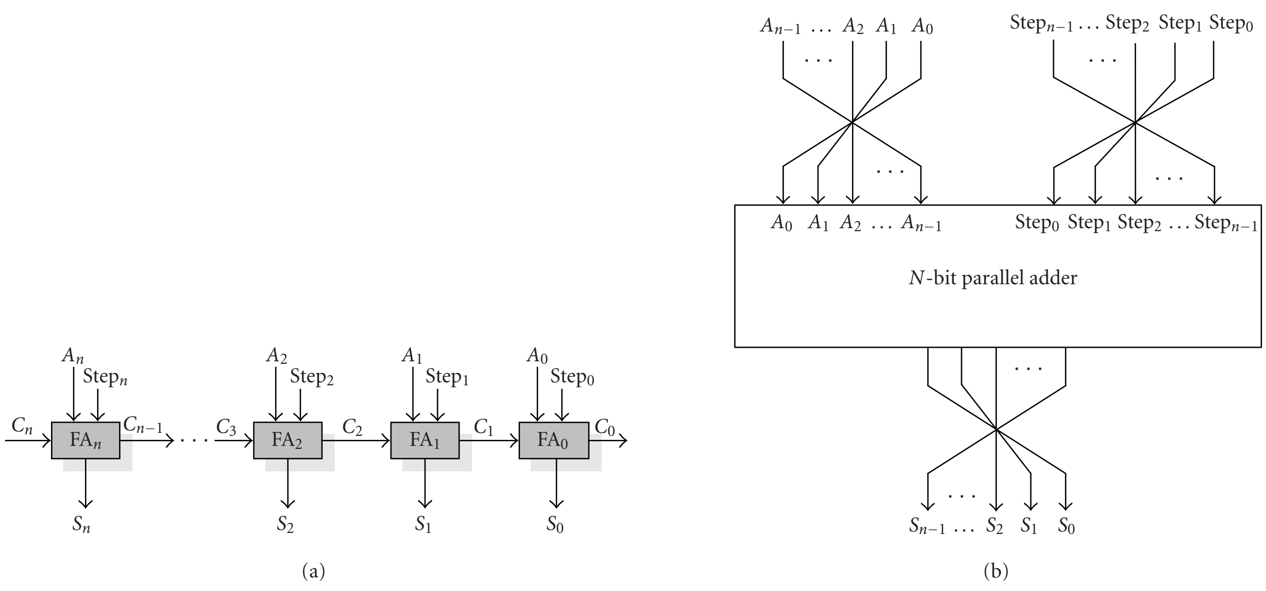

A_n
Step_n
C_n
FA_n
C_{n-1}
C_3
A_2
Step_2
FA_2
C_2
A_1
Step_1
FA_1
C_1
A_0
Step_0
FA_0
C_0
S_n
S_2
S_1
S_0
(a)
$A_{n-1} \dots A_2\ A_1\ A_0$
$\text{Step}_{n-1} \dots \text{Step}_2\ \text{Step}_1\ \text{Step}_0$
$A_0\ A_1\ A_2 \dots A_{n-1}$
$\text{Step}_0\ \text{Step}_1\ \text{Step}_2 \dots \text{Step}_{n-1}$
N-bit parallel adder
$S_{n-1} \dots S_2\ S_1\ S_0$
(b)

FIGURE 4: (a) Previous and (b) new bit-reversed addressing generator architecture in NCU_DSP.

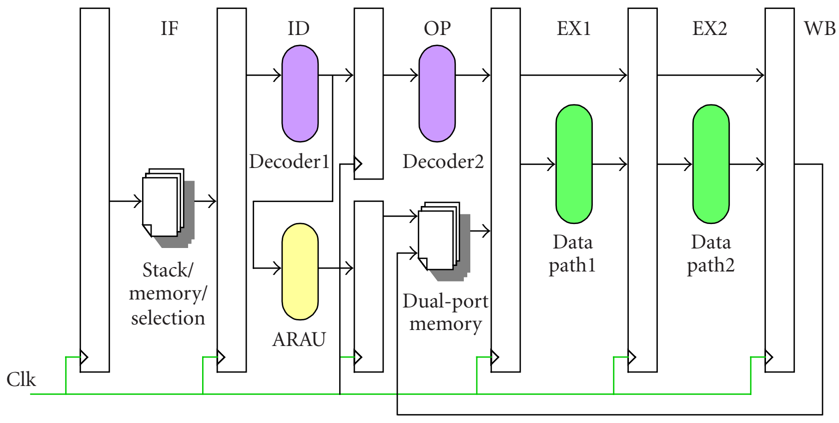


FIGURE 5: The pipeline stages of NCU_DSP.

execution one (EX1) stage, execution two (EX2) stage, and write-back (WB) stage, as shown in Figure 5. To accelerate the performance of NCU-DSP, data-path calculation was split into the EX1 and EX2 stages. The most troublesome problems encountered using the pipelining technique were data hazards [13]. Data hazards occur when the next instruction needs to use data that is still being calculated by the present instruction. Six clock cycles are required for the present instruction to calculate the data and write it back to memory. The next instruction fetches the data just three stages behind (OP stage). Consequently, the programmer needs to insert some useless instructions (e.g., NOP) to avoid the data hazard. To reduce the penalties arising from data hazards, this work adopts the data-forwarding technique in [13, 14].

The following example describes an example of data hazard:

```
.........
STL     A, *AR3
MAC2    *AR3, *AR2, A
.........
```

The *__AR__3 is not ready until "STL" completes in the sixth stage. Thus, three NOPs must be added between "STL" and "MAC2." Figure 6 illustrates the data-forwarding technique, which reduces the required number of NOPs to just one.

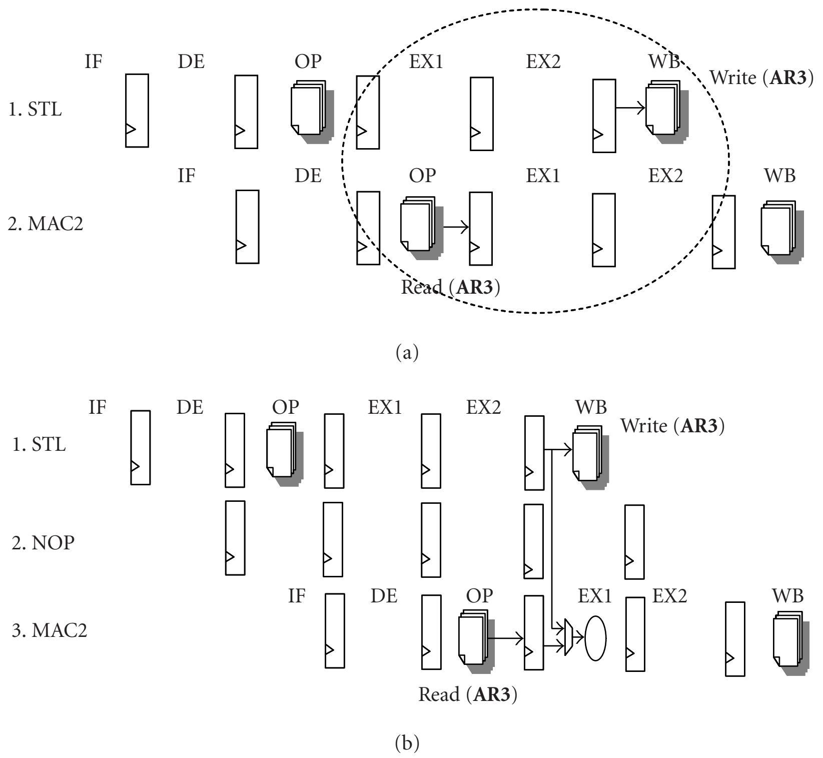


FIGURE 6: (a) An example of data hazard. (b) Using of data-forwarding technique to resolve the data hazard.

3. DATA PATH AND SPECIAL FUNCTIONAL BLOCKS

3.1. *Dual MAC architecture*

The MAC data-path operation, which is the most important instruction in DSP, is the key to enhancing DSP operation performance. Millions of multiply accumulates per second (MMACS) is more relevant than millions of instructions per second (MIPS). Here, "dual" indicates two MACs per cycle. For example, the FIR algorithm is the most apparent usage of the dual MAC unit. The following equation can express the operation of the FIR filter process:

$$Y(n) = \sum_{i=0}^{N-1} h(i) \cdot X(n-i), \tag{1}$$

where $Y(n)$ denotes the output sample, $h(i)$ represents the coefficient, and $X(n-i)$ is the input data. Two consecutive output samples can be listed for analysis:

$$\begin{aligned} Y(0) &= h(0) \cdot X(0) + h(1) \cdot X(-1) \\ &\quad + h(2) \cdot X(-2) + \cdots + h(N-1) \cdot X(1-N), \\ Y(1) &= h(0) \cdot X(1) + h(1) \cdot X(0) \\ &\quad + h(2) \cdot X(-1) + \cdots + h(N-1) \cdot X(2-N). \end{aligned} \tag{2}$$

Each output sample of N taps filter will take N instruction cycles in the single MAC path. To accelerate performance, this work established another MAC path in the DSP data path, as shown in Figure 7. This second MAC path comprises a newly added multiplier along with the original ALU block. Regarding the data-flow consideration, a delay register is added between the single MAC path and the second MAC path to create the data source. This approach can save data access requests where the coefficient remains the same for each arm (Table 2). This architecture can be used to obtain two output samples simultaneously. Therefore, only around $N/2$ instruction cycles are required to complete the same operations in the single MAC architecture. Meanwhile, the dual MAC unit also reduces memory access power consumption. The single MAC unit needs $2N$ memory accesses, while the

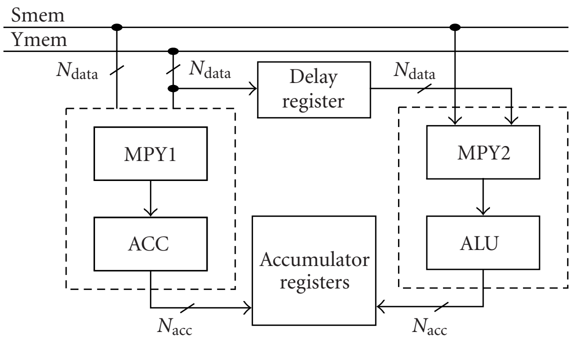


FIGURE 7: The dual MAC data-path block diagram.

Table 2: The performance comparison of MAC structures in FIR algorithm [15].

	Single MAC	Dual MAC with 3 access bus	Dual MAC with Reg.
No. of MAC operation	N	N	N
No. of memory read	$2N$	$2N$	N
No. of instruction cycle	N	$N/2$	$N/2$

Table 3: The overhead of the special block.

	Clock cycle (without special block)	Process	Overhead	Overhead* (percentage)
Hamming distance calculator (16 bits)	1(33)	0.35 μm	174 (gate count)	0.35%
		0.25 μm	238 (gate count)	0.38%
Multilevel slicer (16 bits)	1($2N; N = \log$(symbol level))	0.35 μm	65 (gate count)	0.13%
		0.25 μm	175 (gate count)	0.28%
Subword MAC unit (16 bits)	1(4)	0.35 μm	1 396 (gate count)	2.79%
		0.25 μm	1 669 (gate count)	2.70%
Dedicated FIR filter with N taps	1(N at single MAC; $N/2$ dual MAC)		Depends on spec. of FIR filter	
Advance hardware looping	1(5)	0.35 μm	850 (gate count)	1.70%
	1(5)	0.25 μm	988 (gate count)	1.60%
Dual MAC (additional multiplier)	1(2)	0.35 μm	2 488 (gate count)	4.98%
		0.25 μm	1 337 (gate count)	2.17%

*Area of whole DSP exclude memory in 0.35 μm is estimated as 49 895 gates and area in 0.25 μm process is estimated as 61 751 gates.

dual MAC unit only requires N memory accesses [15]. The dual MAC architecture is an optional special function. In our dual MAC architecture, the hardware overhead is only one delay register and one multiplier (MPY2). The critical delay path of dual MAC structure is the same with the single MAC architecture. The user can select the function as optional. Table 3 lists the overheads of additional multipliers in different technologies.

3.2. Subword MAC

The subword process architecture can partition an n-bit data into two $n/2$-bit data so that data processing can be accelerated, as in the I/Q channel data processing of the communication system. Furthermore, in certain cases, parts of the system do not need to operate at high resolution, meaning data can be expressed using a half-word length. Based on the half-word-length representation, a set of parallel subword process paths can be designed rather than a full-word process path. Subword parallelism process is also highly efficient for application-specific data processors [16, 17]. For example, the subword MAC unit can reduce the complex MAC operation [17]. The complex vector of operations includes real and image parts. Multiplying complex numbers requires four multiplication operations. Notably, the subword MAC unit achieved four MAC operations in a single cycle. The subword process can be further divided into two parts, namely, the subword multiplier and the subword accumulator.

A subword multiplier is designed to complete three different types of multiplications: subword multiplication, conventional full-word multiplication, and complex-word multiplication, as illustrated in Figure 8. The first mode, namely, the subword mode, is designed to multiply the upper and lower half of operands, respectively. Moreover, the second mode, conventional full-word multiplication, is implemented to help the subword multiplier maintain the capability to perform a full-word process. Finally, the third mode, complex-word multiplication, assumes that a full-word comprises both real-part and image-part subwords.

This design assumes that both full-word and subword data are coded in the two's complement system. This work uses AH to indicate the upper half and AL to indicate the lower half of a word; "AH@AL" represents "$\text{AH} \times 2^{N_{\text{sub}}} + \text{AL}$," where N_{sub} is the number of bits in a subword.

The multiplier mainly comprises four subword multipliers (with outputs of carry and summation words to avoid the requirement of carry-propagation adder) followed by a carry-save-adder (CSA) tree, as shown in Figure 9. Each subword multiplier is designed as a booth multiplier, meaning that it only deals with signed data. In the subword multiplication mode, operands need to be sent to the multiplier, and then the corresponding results are selected from subword multipliers.

Regarding the second mode, the conventional full-word multiplication, the arrangement displayed in Figure 8b is

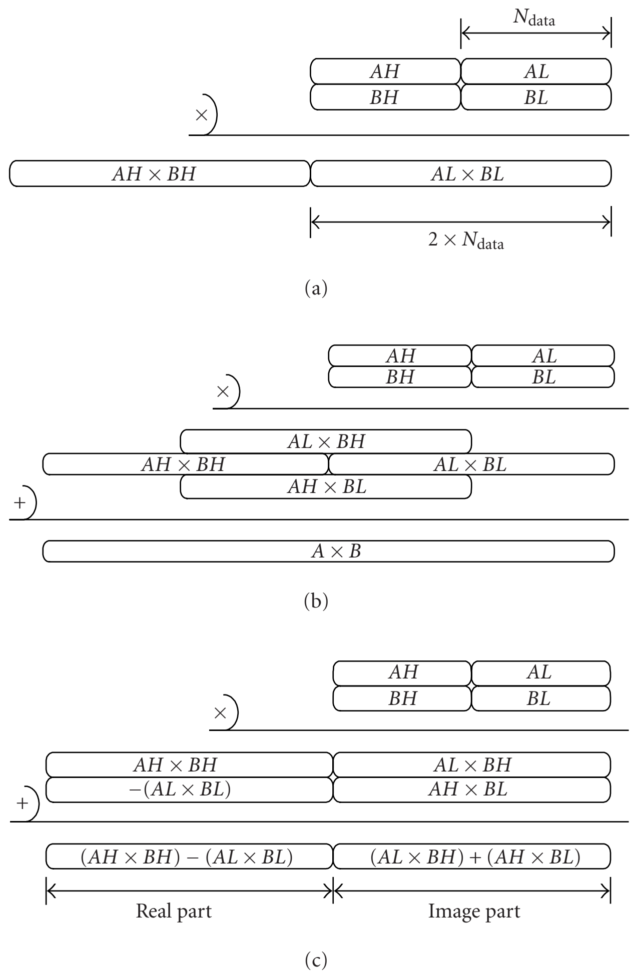

N_{data}
AH AL
BH BL
$\times$
$AH \times BH$ $AL \times BL$
$2 \times N_{\text{data}}$
(a)
AH AL
BH BL
$\times$
$AL \times BH$
$AH \times BH$ $AL \times BL$
$AH \times BL$
$+$
$A \times B$
(b)
AH AL
BH BL
$\times$
$AH \times BH$ $AL \times BH$
$-(AL \times BL)$ $AH \times BL$
$+$
$(AH \times BH) - (AL \times BL)$ $(AL \times BH) + (AH \times BL)$
Real part
Image part
(c)

FIGURE 8: (a) Subword multiplication in subword multiplier. (b) Full-word multiplication in subword multiplier. (c) Complex-word multiplication.

impractical, and thus a compensation term is required to correct the computation results. This situation exists because the operand A in the two's complement number system cannot be represented using AH@AL directly. The problem in full-word multiplication can be expressed as follows:

$$\begin{aligned}
B &= -B_{N_{\text{data}-1}} \times 2^{N_{\text{data}-1}} + \sum_{i=0}^{N_{\text{data}-2}} B_i \times 2^i, \\
B_H &= -B_{N_{\text{data}-1}} \times 2^{N_{\text{data}-1}} + \sum_{i=N_{\text{sub}}}^{N_{\text{data}-2}} B_i \times 2^i, \\
B_L &= -B_{N_{\text{sub}-1}} \times 2^{N_{\text{sub}-1}} + \sum_{i=0}^{N_{\text{sub}-2}} B_i \times 2^i, \\
B &= BH@BL + B_{N_{\text{sub}-1}} \times 2^{N_{\text{sub}}}.
\end{aligned} \tag{3}$$

According to booth multiplier characteristic, the operand A which acts as the multiplicand should be modified as

$$\text{AH} = \sum_{i=N_{\text{sub}}}^{N_{\text{data}-1}} A_i \times 2^{i-N_{\text{sub}+1}} + A_{N_{\text{sub}-1}}. \tag{4}$$

Thus, $A \times B$ can be expressed as

$$\begin{aligned}
A \times B &= A \times \left(BH@BL + B_{N_{\text{sub}-1}} \times 2^{N_{\text{sub}}}\right) \\
&= A \times (BH@BL) + A \times B_{N_{\text{sub}-1}} \times 2^{N_{\text{sub}}}.
\end{aligned} \tag{5}$$

The term $A \times B_{N_{\text{sub}-1}} \times 2^{N_{\text{sub}}}$ is the compensation term that should be implemented. The subword multiplier developed here can compensate this term in the CSA tree following the multipliers. Notably, the compensation term exists because the critical path presented here is located on the MAC path. The MAC function must be balanced between EX1 and EX2. The compensation term of the multiplier is left to the next stage, EX2. Figure 9 also illustrates that the CSA tree is in the EX2 stage. Figure 10 shows the basic data arrangement in the case of complex mode. A special arrangement occurs in the subtraction in the real-part computation. The subtraction in the two's complement must perform the one's complement of the subtraction first, and then add the complement and compensation terms to the minuend. Similarly, the addition of the compensation term is implemented in the CSA tree. The word length of the subword MAC structure can be selected by the user and parameterized from 8 to 32 bits. The subword resolution ranges from 4 to 16 bits, respectively. In [17, 18] the MAC data path is accomplished in one pipeline stage. In our design, it is arranged across two pipeline stages. Each multiplier has two outputs to avoid carrying propagation in the final stage of multiplier. The multiplier outputs, compensation term, and accumulation output are summed by CSA tree to speed up the computation. Moreover, in the full-word multiplication mode, due to the compensation-term arrangement, we do not have to cascade two multipliers operation as those used in [18].

The subword MAC and non-subword MAC are both synthesized and evaluated. The area of 0.35 μm design (16×16) is approximately 12132 gate counts (subword) and 10736 gate counts (nonsubword). The delay times of both cases are synthesized to meet 5.56 nanoseconds. The overhead is 1396 gate counts, approximately 13% compared to the non-subword MAC. In the 0.25 μm technology design, the subword and non-subword MAC were synthesized to meet the 5 nanoseconds delay time. The area of subword MAC (16×16) is about 8669 gate counts, while that of non-subword MAC is around 7000 gate counts. The overhead of the subword MAC is approximately 23.8% with respect to the non-subword MAC. The subword MAC generally consumes 20% more power than the non-subword MAC.

3.3. Optional special functional blocks and parameters

The special function blocks are merged into the NCU-DSP described here for two important reasons. First, in some

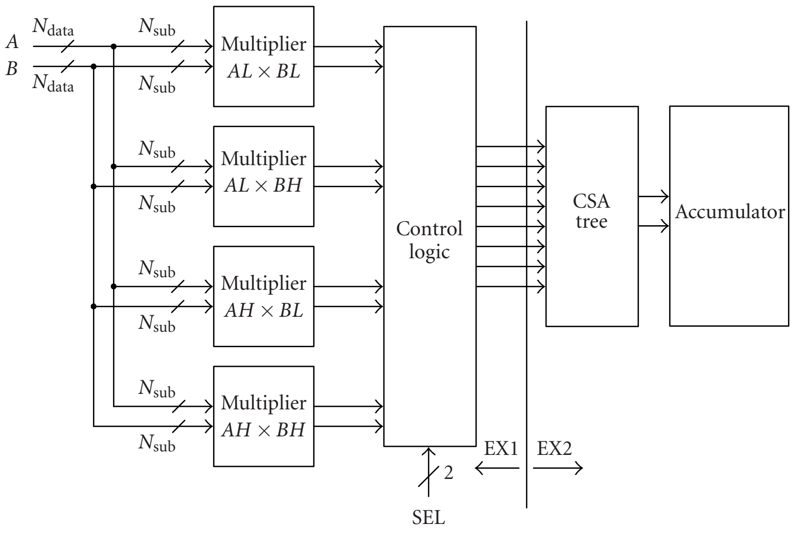

A
B
N_{data}
N_{data}
N_{sub}
Multiplier $AL \times BL$
Multiplier $AL \times BH$
Multiplier $AH \times BL$
Multiplier $AH \times BH$
Control logic
CSA tree
Accumulator
2
SEL
EX1
EX2

FIGURE 9: Subword multiplier architecture.

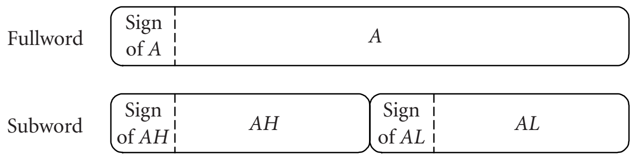


FIGURE 10: Subword data arrangement.

applications with high sampling rate, special functional blocks represent the only reasonable approach. If the DSP processor can provide special data paths that comprise these functional blocks and do not increase overheads significantly, then the provision of these paths is worthwhile. Second, communication systems usually use some special function units with a small area compared to the overall DSP gate counts. For example, the multilevel slicer unit can reduce the instruction cycles of a symbol mapper operation in communication system. It can reduce the symbol mapper operation from $2N$, $N = \log(\text{symbol level})$, to just one instruction cycle. The circuit area overheads of the functional block are only 0.13% (0.35 μm) and 0.28% (0.25 μm) of the whole DSP core (excluding memory). For reasons of performance and flexibility, these blocks are also merged in our NCU-DSP. Based on the above two reasons, some special function circuits are offered for selection, as listed in Table 4. Table 3 lists the hardware overheads and acceleration factors of these circuits.

Recently, a more flexible DSP core has been proposed, namely, the so-called parameterized DSP core [20]. The parameterized DSP core is parameterized using several independent parameters. Table 5 lists the parameters of NCU_DSP. The most important parameter in the table is the "data word." This parameter exerts the biggest influence on the chip size and performance of the NCU-DSP. For users to whom chip area is important, care must be taken regarding the parameters data address length (DAL) and program address length (PAL). Care is necessary because memory generally occupies a large part of the total area of the DSP processor. Some parameters are related to one another. For example, ALU is used to calculate the operands from the multiplier or accumulator. Consequently, the word length of ALU must be related to the multiplier and accumulator. The related functional blocks must have the same data length.

4. LOW-POWER DESIGN

Various methods of saving power have been proposed for use in the design of DSP. These methods include bus segmentation, data access reduction, program memory access reduction, gray-code addressing, and pipeline register reduction [21, 22]. The following sections address some key low-power design methods used in the DSP presented here.

4.1. Gray-code addressing

The advantage of gray code compared to straight binary code is that gray code changes by only one bit while changing from one number to the next number. That is, if the memory access pattern is a sequence of consecutive address, each memory access changes by only one bit at its address bits. Owing to instruction locality during program execution, the program memory accesses in DSP applications are mostly sequential. Therefore a significant number of bit switching can be eliminated via gray-code addressing [21]. For example, the sequence of number from 0 to 15 are 26 bits switched when the number is encoded in binary representation, and are only 15 bits switched when the number is encoded in gray-code representation. This arrangement

Table 4: The optional special function and multifunction blocks.

Function name	Range (bit)	Description
Dedicated CSD FIR	8–32	For fixed-coefficient FIR [19]
Slicer	8–32	For multilevel symbol system
Hamming distance	8–32	For code distance applications
Subword multiplier	8–32	For complex operation, etc.
Dual MAC	8–32	For communication applications
Rounded/saturation mode	—	For accumulation applications
Buffered hardware loop	—	For loop computations

Table 5: Parameters of NCU_DSP.

Parameter	Range	Default	Relation	Description
Data word	$8 \sim 32$ bits	16 bits	m bits	Data word length
IO_width	$14 \sim 32$ bits	16 bits	$\max(14, m)$ bits	Input/Output operand width
M_width	$8 \sim 32$ bits	16 bits	m bits	Multiplier word length
G_width	$0 \sim 16$ bits	8 bits	n bits	Guard bit length
A_width	$16 \sim 80$ bits	40 bits	$(2^* m + n)$ bits	The word length of accumulator
HPIA_length	$8 \sim 16$ bits	9 bits	$\max(N_D, N_P, N_{\text{HPI}})$ bits	The word length of HPIA
R-num	$2 \sim 8$	2	N_a	Number of accumulator register
M-num	$1 \sim 2$	1	N_m	Number of multiplier
AR-num	$2 \sim 8$	8	N_{AR}	Number of auxiliary register
Loop-num	—	3	N_{NL}	Maximum nested loop number
PC_stack_size	—	8	N_{PCS}	The depth of PC stack
Buffer_size	—	16	N_{EB}	The size of Embedded buffer
DAL	$8 \sim 16$ bits	9 bits	N_D	Data-memory address length
PAL	$8 \sim 16$ bits	9 bits	N_P	Program-memory address length
HPIAL	$5 \sim 13$ bits	6 bits	N_{HPI}	Host port interface memory address length

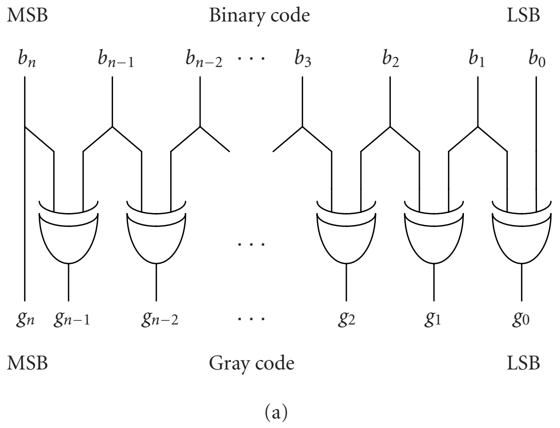
reduces the switching activity of the bus and thus also reduces the power consumption of the bus driver.

Figure 11a displays the block diagram of the binary-to-gray (B2G) coding circuit [22]. The hardware of the conversion circuit is approximately 21 gates, each loading capacitor of standard gate is about 15 fF. The loading capacitor of the program bus line driving from PAGU to the program memory (0.5 K word) is about 0.32 pF, totally about 2.88 pF. Our design discards the gray-to-binary (G2B) circuit, which consumes twice the power of the B2G circuit, to increase power savings. The program instructions in the program memory are stored using a gray-coding arrangement. Thus, the PAGU to program-memory interface is shown in Figure 11b. The switching activity of gray coding is about half of the binary coding in sequential memory access [22]. The power consumption (P) is proportional to the switched capacitor, $P = \alpha C f V^2$, where α is a switching probability, C is capacitance of circuit, f is frequency, V is supply voltage. In our design, the bus loading capacitor and B2G-circuit loading are 2.88 pF. Thus, the power saving in this case is about $(0.315 + 1.44)/2.88 = 60.9\%$.

4.2. *Advanced hardware looping*

The hardware looping circuit can reduce program size and execution cycles [9]. The hardware looping circuit reduces the number of instructions and clock cycles by using a hardware circuit instead of software instructions for the looping. Table 6 shows the processing sequences that distinguish hardware looping from software looping. However, the DSP processor still needs to fetch the program memory for each instruction, despite the instruction having already been fetched in the last execution of looping time. Each IF needs to pass signals through memory and interconnect system elements, buses, multiplexers, and buffers, consuming a significant percentage of total power [23]. Accordingly, this work designs an advanced hardware looping circuit.

The key objective of the advanced hardware looping is to save the repetition of instructions in the instruction register or instruction buffer (IB). Accordingly, the program memory is not accessed while instructions are repeated, and the value on the bus connected to program memory remains


MSB
Binary code
LSB
b_n b_{n-1} b_{n-2} ··· b_3 b_2 b_1 b_0
g_n g_{n-1} g_{n-2} ··· g_2 g_1 g_0
MSB
Gray code
LSB
(a)

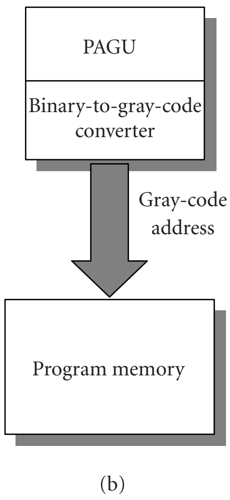


Figure 11: (a) Binary code to gray-code conversion circuit. (b) The conversion circuit in PAGU.

Table 6: Implementation of 1024 times ADD with hardware looping and software looping.

	Hardware looping	Software looping
Instruction	RPTZ Reg0,1023 ADD *AR1+, *AR2+, *AR3+	STM 1023, AR0 ADD *AR1+, *AR2+, *AR3+ ADD 1, *AR4 CMPR CC, AR4 BC pmad, cond
No. of instructions	$1024 + 1 = 1025$	$1024 \times 4 + 1 = 4097$

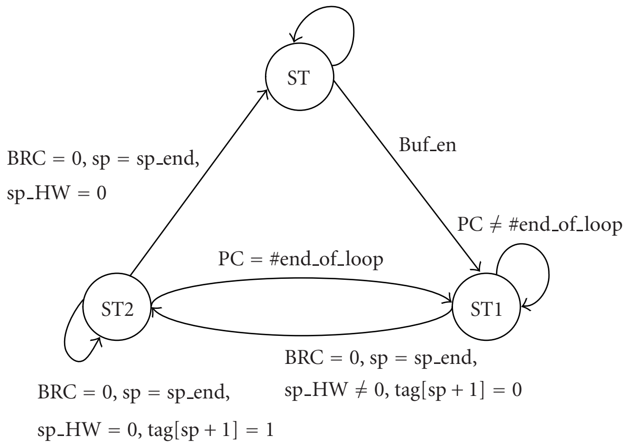

ST
ST1
ST2
Buf_en
PC ≠ #end_of_loop
PC = #end_of_loop
BRC = 0, sp = sp_end, sp_HW = 0
BRC = 0, sp = sp_end, sp_HW ≠ 0, tag[sp + 1] = 0
BRC = 0, sp = sp_end, sp_HW = 0, tag[sp + 1] = 1

Figure 12: The state diagram of hardware looping.

unchanged. This approach can reduce the power consumption of the program memory and related buses.

The operation of repeating a block of instructions with IB can be divided into three phases, as displayed in Figure 12. Phase ST0 means that the hardware looping is inactive, and thus no instructions need to be repeated. Meanwhile, in phase ST1, the hardware looping is active and IB receives instructions from the program memory. Simultaneously, the instructions are stored in the IB. But when the circuit is in phase ST2, the program memory is switched off and IR accesses instructions from IB until the content of block-repeat counter (BRC) becomes zero. This scheme means that program memory only needs to be accessed once.

To implement nested loop, this work adds a loop stack to store the repeat-start address (RSA) register, repeat-end address (REA) register, and BRC of the current loop. This work focuses on the nested loop with the form illustrated in Figure 13a, and creates a new instruction, RPTBX. In case of other forms, nested loop still can be implemented using extra instructions such as PUSHHW and POPHW, as shown in Figure 13b. Since some applications may be concerned with chip area rather than power consumption, the nested loop circuit and IB are regarded as an optional module. Furthermore, the size of IB is also parameterized. Users can select the size of IB. The design in our DSP differs from the popular so-called IB in [24, 25, 26]. The key difference is that IB does not work if the instruction is not looping. Moreover, IB only stores the instructions contained in the loop. Additionally, in our design the IB involves no hitting rate or instruction

```
STM#end_loop1, REA
RPTBX No_loop1
. . .
STM#end_loop2, REA
RPTBX No_loop2
. . .
STM#end_loop3, REA
RPTBX No_loop3
. . .
#end_loop3
. . .
#end_loop2
. . .
#end_loop1
. . .
```

(a)

```
STM#end_loop1, REA
RPTB No_loop1
. . .
PUSHHW
STM#end_loop2, REA
RPTB No_loop2
. . .
#end_loop2
POPHW
. . .
PUSHHW
STM#end_loop3, REA
RPTB No_loop3
. . .
#end_loop3
POPHW
. . .
#end_loop1
. . .
```

(b)

Figure 13: Examples of nested loop. (a) Loop that can be coded using instruction RPTBX. (b) Loop that can be coded using instructions RPTB, PUSHHW, and POPHW.

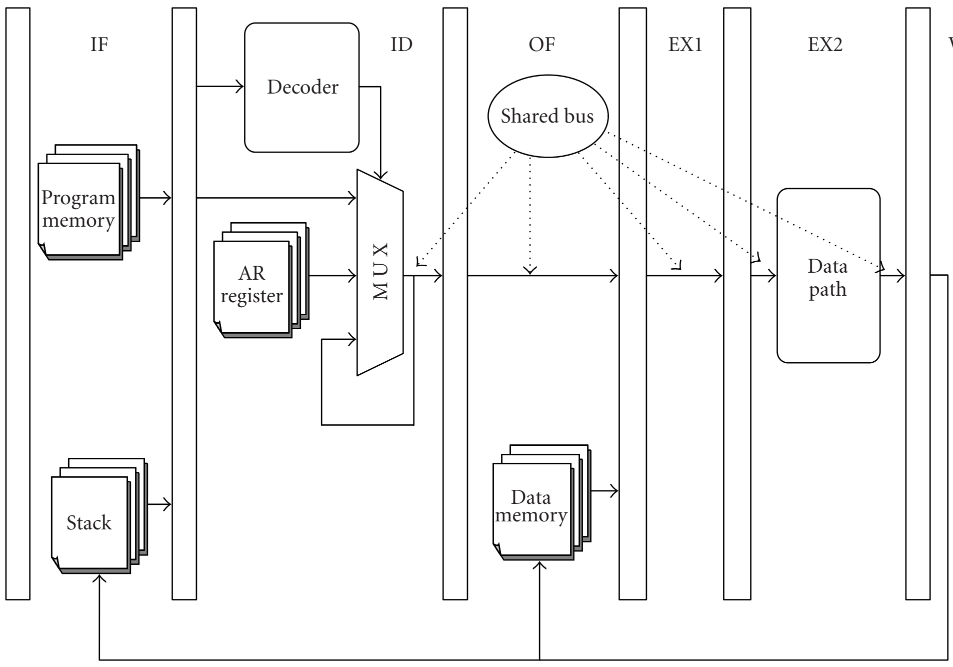


Figure 14: The block diagram of pipeline sharing.

cycle penalties. Furthermore, the structure has negligible overheads compared to hardware looping without IB in other DSPs. The control circuit of this advanced hardware looping is only 1.6% overhead compared with the whole DSP area. IB size is a parameter that can be varied according to application demands.

Table 7: Instruction examples of pipeline sharing.

Instruction	Operation	Data	No. of stages passed
CC Cond, #lk	$\#lk \longrightarrow PC, PC + 1 \longrightarrow$ stack while Cond is true	PC + 1	4

ADDM Xmem, Ymem
ORM Xmem, Ymem
	Ymem = (Xmem) + (Ymem)
	Ymem = (Xmem)|(Ymem)
		[ARx]	4

STH src, SHIFT, Ymem STL src, SHIFT, Ymem	Ymem = src $\ll$ SHIFT	[ARx]	4

Table 8: Example of assembly code for a 16-tap FIR filter.

Address	Instruction	Comment
	/********& program data start************/	
0	LAR # 180 h, AR6	;input buffer
1	LAR # 1 h, AR0	;index
2	LAR # 4 fh, AR1	;data pointer
3	LAR # 80 h, AR2	;coefficient pointer
4	LAR # 100 h, AR7	;output buffer
	/*********FIR_task*********/	
5	STM # FIR_filter_loop, REA	;repeat block last position
6	STM # 17, BK1	;for buffer size
7	STM # 17, BK2	;for buffer size
8	RPTB # 255	;hardware looping
9	LD # AR6+,C	
a	LD # AR6+,D	
b	LD # AR6+,B	
	/*********FIR_filter*********/	
c	STL C, *AR1+,%	
d	STL D, *AR1+,%	
e	RPTZ A, 17	;data forwarding
f	MAC//MAC *AR1+0%, *AR2+0%, A, B	;dual MAC instruction
10	STH A, *AR7+	;AR7 is outbuffer
11	STL B, *AR7+	;AR7 is outbuffer
	/*********FIR_filter_loop*********/	

4.3. *Pipeline sharing*

In pipeline architecture, the pipeline registers contribute significantly to area and power consumption. Some signals simply pass through the pipeline stages without being used. Therefore, the pipeline sharing technique was adopted here to reduce the number of pipeline registers and thus reduce power and area.

Figure 14 shows the block diagram of pipeline sharing. Table 7 lists instructions that do not use the data address (ARi) and program address (PCi) until they are transmitted to the last pipeline stage. Therefore, these data occupy many unnecessary pipeline registers. For example, the instruction ADDM performs the addition of two memory operands and then stores the result in the memory that holds the value of ARi until the final pipeline stage (WB). On the other hand, the instruction CC, conditional call, maintains the value of PCi plus one until the final stage. The values of PCi and ARi share the same buses and pipeline registers. The multiplexer determines which data are loaded into the shared bus. If some instructions do not use the shared bus and associated pipeline registers, the value of buses and pipeline registers can be held without passing through. The unpassing signals contribute zero transition on the registers and buses to reduce power consumption. The area overhead associated with the pipeline sharing technique is a multiplexer and increases the complexity of the instruction decoder. In this parameterized DSP, the size of the program and data memory may differ. Accordingly, the length of the shared bus should be the maximum of the address bus in terms of program and data memory. The pipeline sharing method can be considered as a direct and simple way to save power consumption in the

Table 9: The design results.

Features	NCU_DSP	NCU_DSP
Technology	TSMC 0.35 μm SPQM	TSMC 0.25 μm 1P5M
Cell library	Avant! 0.35 cell library	Artisan 0.25 cell library
Power supply	3.3 V	2.5 V
Pipeline	6-stage	6-stage
Max. operating frequency	100 MHz	140 MHz
Die size	4 152 $\times$ 4 152 μm^2	1 281 $\times$ 1 281 μm^2 (estimated)
Core area (memory excluded)	51 349 gate counts	1 131 581 μm^2 (65 485 gate counts)
On-chip memory	512 $\times$ 24 bit (program), 512 $\times$ 16 bit (data), 64 $\times$ 16 bit (HPI)	512 $\times$ 24 bit (program), 16 $\times$ 24 bit (Instruction buffer), 512 $\times$ 16 bit (data), 64 $\times$ 16 bit (HPI)
Special function blocks	Indirect addressing mode	Indirect addressing mode subword Multiplier Hamming distance slicer

data path circuit. The bus segment method performs well in saving power [27] which is dealing with data bus and address bus. The example in [27] requires a more complicated control circuit than pipeline sharing.

This pipeline sharing approach reduced four 16-bit pipeline registers and 64 wires out of eight 16-bit pipeline registers and 128 wires in the example with a 16-bit word structure. The overhead associated with this approach include a multiplexer and a slight increase in the complexity of the instruction decoder.

5. IMPLEMENTATION RESULTS AND EXAMPLE

5.1. FIR filter function example

Since the proposed dual MAC architecture, Figure 7, supports two parallel operation of MAC, it can accelerate FIR operation by a factor of two. Table 8 displays the example of assembly code. The instructions #14~#15 (address d and address e) in Table 8 are an example of data forwarding for two NOP, saving. Significantly, the dual MAC structure requires only 18 instructions to complete the example. In contrast, if only one MAC is used, it requires 35 instructions to complete the function.

5.2. Chip verification

To verify the NCU_DSP, a 16-bit DSP core with an instruction set of 24-bits word is designed. This architecture contains three memory blocks on chip, one 24-bit*512-word two-port RAM for the program memory, one 16-bit*512-word dual-port SRAM for the data memory, and one 16-bits*64-word dual-port SRAM for the HPI memory. The word length of the accumulator is 40 bits, and the guard bits are relatively eight. The synthesis result demonstrates that the maximum frequency is 140 MHz with 0.25 μm cell-library implementation, and the critical path is the EX2 stage. Figure 15 displays the area of each stage with the 0.35 μm and 0.25 μm cell library. Moreover, Figure 16 shows the timing comparison of each stage with the 0.35 μm and 0.25 μm cell library. Table 9 lists the features in the first (0.35 μm) and second (0.25 μm) versions of NCU_DSP. This work uses the cell-based design flow to implement the DSP core. The 0.35 μm has been taped out and the post-layout simulation reveals that it operates effectively at 100 MHz with 75 mW. Figure 17 shows the die photo of our design in 0.35 μm.

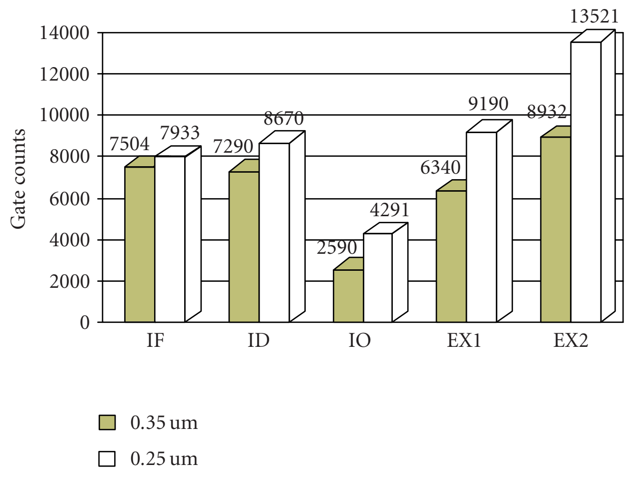


Figure 15: The gate counts of each stage with 0.35 μm and 0.25 μm cell library.

6. CONCLUSIONS

This work presented a parameterized embedded DSP core for demodulation/synchronization in a communication system. The parameterized structure is easily embedded in systems with different system requirements. The special functional blocks of this DSP core can achieve improved performance and flexibility with minimum area overhead. Furthermore, NCU_DSP is designed using several low-power methods to reduce power consumption. The proposed DSP core can meet the cost/performance in mostly DSP-based applications.

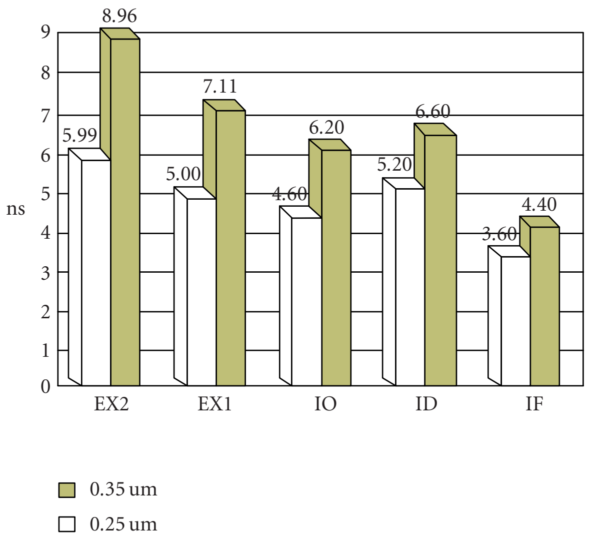


FIGURE 16: Timing comparison of each stage with 0.35 μm and 0.25 μm cell library.

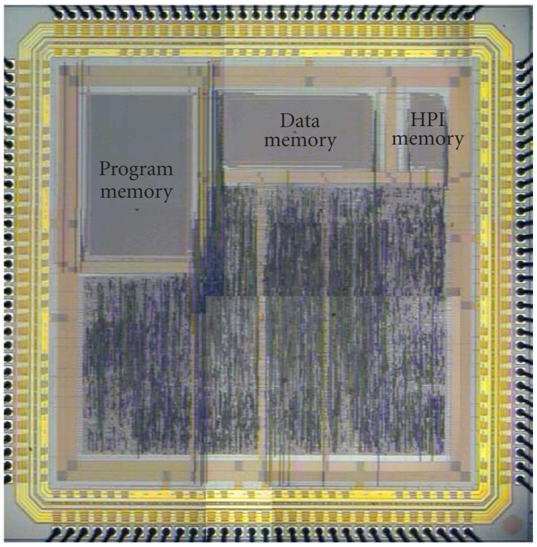


FIGURE 17: The die photo of NCU_DSP (0.35 μm).

REFERENCES

[1] D. D. Clark, E. A. Feigenbaum, D. P. Greenberg, et al., "Innovation and obstacles: the future of computing," *IEEE Computer*, vol. 31, no. 1, pp. 29–38, 1998.

[2] G. Frantz, "Digital signal processor trends," *IEEE Micro*, vol. 20, no. 6, pp. 52–59, 2000.

[3] J. Turley and H. Hakkarainen, "TI's new 'C6x DSP screams at 1600 MIPS," *Microprocessor Report*, vol. 11, pp. 14–17, 1997.

[4] I. Verbauwhede and M. Touriguian, "A low power DSP engine for wireless communications," *Journal of VLSI Signal Processing Systems*, vol. 18, no. 2, pp. 177–186, 1998.

[5] AT&T Data Sheet, DSP1618 digital signal processor, February 1994.

[6] Texas Instruments, TMS320C54X DSP Reference Set, Volume 1: CPU and Peripherals, 1997.

[7] M. Alidina, G. Burns, C. Holmqvist, E. Morgan, and D. Rhodes, "DSP16000: A high performance, low power dual-MAC DSP core for communication applications," in *Proc. IEEE Custom Integrated Circuits Conference (CICC '98)*, pp. 119–122, Santa Clara, Calif, USA, May 1998.

[8] B. W. Kim, J. H. Yang, C. S. Hwang, et al., "MDSP-II: A 16-bit DSP with mobile communication accelerator," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, pp. 397–404, 1999.

[9] Y. Tsao, S. Jou, H. Lee, Y. Chen, and M. Tan, "An embedded DSP core for wireless communication," in *Proc. International Symposium on Circuit and System (ISCAS '02)*, vol. 4, pp. 524–527, Scottsdale, Ariz, USA, May 2002.

[10] R. Mehra, L. M. Guerra, and J. M. Rabaey, "A partitioning scheme for optimizing interconnect power," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 3, pp. 433–443, 1997.

[11] M. Kuulusa, J. Nurmi, J. Jakala, P. Ojala, and H. Herranen, "A flexible DSP core for embedded systems," *IEEE Design & Test of Computers*, vol. 14, no. 4, pp. 60–68, 1997.

[12] A. Gierlinger, R. Forsyth, and E. Ofner, "GEPARD: A parameterisable DSP core for ASICS," in *Proc. International Conference on Signal Processing Applications & Technology (ICSPAT '97)*, pp. 203–207, Scottsdale, Ariz, USA, 1997.

[13] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann Publishers, San Francisco, Calif, USA, 2nd edition, 1998.

[14] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalom, "Exploiting data forwarding to reduce the power budget of VLIW embedded processors," in *Proc. Conference and Exhibition on Design, Automation and Test in Europe (DATE '2001)*, pp. 252–257, Munich, Germany, March 2001.

[15] I. Verbauwhede and C. Nicol, "Low power DSP's for wireless communications," in *Proc. International Symposium on Low-Power Electronics and Design (ISLPED '00)*, pp. 303–310, Rapallo, Italy, July 2000.

[16] J. Fridman, "Sub-word parallelism in digital signal processing," *IEEE Signal Processing Magazine*, vol. 17, no. 2, pp. 27–35, 2000.

[17] Y. Huang and T. D. Chiueh, "A sub-word parallel digital signal processor for wireless communication systems," in *Proc. IEEE Asia-Pacific Conference on ASIC (AP-ASIC '02)*, pp. 287–290, Taipei, Taiwan, August 2002.

[18] C. K. Chen, P. C. Tseng, Y. C. Chang, and L. G. Chen, "A digital signal processor with programmable correlator array architecture for third generation wireless communication system," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 12, pp. 1110–1120, 2001.

[19] M. C. Lin, C. L. Chen, D. Y. Shin, C. H. Lin, and S. J. Jou, "Low-power multiplierless FIR filter synthesizer based on CSD code," in *Proc. International Symposium on Circuit and System (ISCAS '01)*, vol. 4, pp. 666–669, Sydney, Australia, May 2001.

[20] J. Nurmi and J. Takala, "A new generation of parameterized and extensible DSP Cores," in *Proc. IEEE Workshop on Signal Processing Systems: Design and Implementation (SiPS '97)*, pp. 320–329, Leicester, Midlands, UK, November 1997.

[21] C. L. Su, C. Y. Tsui, and A. M. Despain, "Low power architecture design and compilation techniques for high-performance processors," in *Proc. IEEE International Computer Conference (COMPCON '94)*, pp. 489–498, San Francisco, Calif, USA, February–March 1994.

[22] H. Mehta, R. M. Owens, and M. J. Irwin, "Some issues in gray code addressing," in *Proc. 6th Great Lakes Symposium on VLSI (GLS '96)*, pp. 178–181, Des Moines, Iowa, USA, March 1996.

[23] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals*, IEEE Press, New York, NY, USA, 1997.

[24] C. Wu and T. T. Hwang, "Instruction buffering for nested loops in low power design," in *Proc. International Symposium*

on Circuit and System (ISCAS '02), vol. 4, pp. 81–84, Scottsdale, Ariz, USA, May 2002.

[25] M. Lewis and L. Brackenbury, "An instruction buffer for a low-power DSP," in *Proc. 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '00)*, pp. 176–186, Eilat, Israel, April 2000.

[26] R. S. Bajwa, M. Hiraki, H. Kojima, et al., "Instruction buffering to reduce power in processors for signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 417–424, 1997.

[27] J. Y. Chen, W. B. Jone, J. S. Wang, H.-I. Lu, and T. F. Chen, "Segmented bus design for low-power systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 25–29, 1999.

Ya-Lan Tsao was born in Taiwan in 1967. He received his B.S. degree from the Department of Electrical Engineering, Chung Cheng Institute of Technology in 1991 and M.S. degrees from the Department of Electronics Engineering, National Central University in 1996. He joined the Department of Electronic, Chung-Shan Institute of Science and Technology, Tao-Yuan, Taiwan in 1991. Since 2000, he is a Ph.D. student in the Department of Electronics Engineering, National Central University. His research interests include high-speed, low-power digital integrated circuits design communication integrated circuits.

Wei-Hao Chen was born in Taiwan in 1977. He received his B.S. and M.S. degrees from the Department of Electronics Engineering, National Central University in 2001 and 2003. He joined VIA Technologies, Inc., Taipei, Taiwan, right after his master course. His research interests include high-speed, low-power digital integrated circuits design and DSP system.

Ming Hsuan Tan was born in Taiwan in 1975. He received his B.S. and M.S. degrees from the Department of Electronics Engineering, National Central University in 1999 and 2003. He joined VIA Technologies, Inc., Taipei, Taiwan, right after his master course. His research interests include high-speed, low-power digital integrated circuits design and DSP system.

Maw-Ching Lin was born in Taiwan in 1962. He received his B.S. degree in electronic engineering from Feng Chia University, Tai Chung, in 1985 and M.S. degree in biomedical engineering from Chung Yuan Christian University, Chung Li, in 1994. He is presently working toward the Ph.D. degree in electrical engineering at National Central University, Chung Li. Since 1985, he has been working at Chung Shan Institute of Science and Technology, Lung Tan, Taiwan. Currently, he is an Associate Scientist. His research interests include high-speed, low-power digital circuit design and architecture optimization in digital signal processing.

Shyh-Jye Jou was born in Taiwan in 1960. He received his B.S. degree in electrical engineering from National Chen-Kung University in 1982, and M.S. and Ph.D. degrees in electronics from National Chiao-Tung University in 1984 and 1988, respectively. He joined the Department of Electrical Engineering, National Central University, Chung Li, Taiwan, in 1990 and is currently a Professor. He was a Visiting Research Associate Professor in the Coordinated Science Laboratory at the University of Illinois during 1993–1994 academic years. He served on the technical program committee of 1994–1996 Custom Integrated Circuits Conference and was Technical Program Cochair of the First, Second, and Third IEEE Asia Pacific Conference on ASIC, 1999, 2000, and 2002. In the summer of 2001, he was a Visiting Research Consultant in the Communication Circuits & Systems Research Laboratory of Agere Systems, USA. His research interests include high-speed, low-power digital integrated circuits design, communication-integrated circuits, and simulation and analysis tools for integrated circuits.