Journal of VLSI Signal Processing 42, 127–138, 2006

DOI: 10.1007/s11265-005-4178-5

A Compact DSP Core with Static Floating-Point Arithmetic*

TAY-JYI LIN, HUNG-YUEH LIN, CHIE-MIN CHAO AND CHIH-WEI LIU
Department of Electronics Engineering, National Chiao Tung University, Taiwan

CHEIN-WEI JEN
SoC Technology Center, Industrial Technology Research Institute, Taiwan

Received November 11, 2004; Revised February 18, 2005; Accepted February 23, 2005

Abstract. A multimedia system-on-a-chip (SoC) usually contains one or more programmable digital signal processors (DSP) to accelerate data-intensive computations. But most of these DSP cores are designed originally for standalone applications, and they must have some overlapped (and redundant) components with the host microprocessor. This paper presents a compact DSP for multi-core systems, which is fully programmable and has been optimized to execute a set of signal processing kernels very efficiently. The DSP core was designed concurrently with its automatic software generator based on high-level synthesis. Moreover, it performs lightweight arithmetic—the static floating-point (SFP), which approximates the quality of floating-point (FP) operations with the hardware similar to that of the integer arithmetic. In our simulations, the compact DSP and its auto-generated software can achieve 3*X* performance (estimated in cycles) of those DSP cores in the dual-core baseband processors with similar computing resources. Besides, the 16-bit SFP has above 40 dB signal to round-off noise ratio over the IEEE single-precision FP, and it even outperforms the hand-optimized programs based on the 32-bit integer arithmetic. The 24-bit SFP has above 64 dB quality, of which the maximum precision is identical to that of the single-precision FP. Finally, the DSP core has been implemented and fabricated in the UMC 0.18μm 1P6M CMOS technology. It can operate at 314.5 MHz while consuming 52mW average power. The core size is only 1.5 mm×1.5 mm including the 16 KB on-chip memory and the AMBA AHB interface.

1. Introduction

Today's multimedia and communication systems demand more and more computations that can no longer be satisfied by a general-purpose microprocessor (μP) at acceptable cost or power consumption [1]. The most popular solution without sacrificing the flexibility is to accompany the μP with a digital signal processor (DSP) [2, 3]. Figure. 1 shows an example of the dual-core processors (or dual-processor cores), where the DSP core handles the data-intensive tasks efficiently with a dataflow engine and the μP takes the control-oriented and interactive tasks via maintaining huge state machines. In general, these two cores are designed individually for standalone uses, and they have overlapped functionalities and thus some redundant components. Recent μP architectures are enhanced for digital signal processing by incorporating single-cycle multiply-accumulators (MAC), SIMD (MMX-like) datapaths, or some specific functional units [4] to reduce the needs for an additional core, but their signal processing performance is still far behind that of a DSP with similar computing resources [5]. This is because data-intensive tasks are very distinct from general-purpose computations [6], and more importantly, the memory subsystem

*This work was supported by the National Science Council, Taiwan under Grant NSC93-2220-E-009-017. Besides, the authors would like to thank the National Chip Implementation Center (CIC) for chip fabrication.

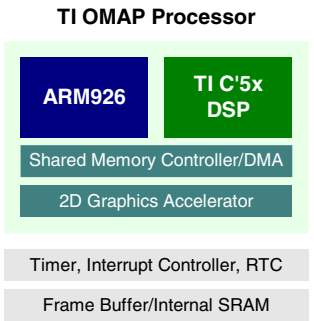


Figure 1. Example of dual-core media processor.

of the single-core system is difficult to optimize for the two different types of tasks simultaneously.

Alternatively, this paper discusses the compaction of the DSP core, and the µP is left unchanged for software compatibility. The DSP core and its automatic code generator have been developed and tuned in parallel, where software techniques are extensively investigated to reduce the hardware complexity as the design principles of VLIW processors [7]. Moreover, we have proposed the static floating-point (SFP) arithmetic to emulate expensive floating-point (FP) [8] DSP operations with hardware as simple as that for the integer arithmetic. In the SFP arithmetic, data are represented as normalized fractional, which are similar to the *mantissa*, but the normalization factors are kept in the analysis software only, in contrast to the *exponents* attached to each FP number. SFP helps to shrink the DSP core effectively without significant overheads of scaling and normalization in other fixed-point implementations. In our simulations of the 2-D DCT in JPEG [9], the 16-bit SFP has above 40 dB signal to round-off noise ratio over the IEEE 754 single-precision FP arithmetic, which even outperforms the hand-optimized 32-bit integer code from the Independent JPEG Group (IJG) [10]. The 24-bit SFP has above 64 dB with the identical maximum precision to that of the FP arithmetic. Finally, the compact DSP core with the SFP arithmetic has about 3*X* better performance in the execution cycles than those conventional DSP cores in the dual-core multimedia processors, such as the Analog Devices ADSP-218x [11]. *DSP-lite* is a 16-bit prototype of the compact DSP core in the UMC 0.18 µm 1P6M CMOS technology. The operating frequency of DSP-lite can achieve 314.5 MHz, while its average power dissipation is only 52 mW. DSP-lite has the standard AMBA AHB interface to simplify the system integration, and its core size is 1.5×1.5 mm^2 including the 16 KB on-chip memory.

The rest of this paper is organized as follows. Section 2 reviews the synchronous dataflow graph (SDFG) and high-level synthesis, and describes an abstract computing model for the configurable DSP datapaths that execute SDFG. Section 3 illustrates our proposed SFP arithmetic and the conversion of an FP SDFG into an SFP one. Section 4 shows the architecture and development software of the DSP-lite core. The simulation and implementation results are available in Section 5. Section 6 concludes this work and outlines our future researches.

2. Preliminaries

2.1. *Synchronous Dataflow Graphs (SDFG) and High-level Synthesis*

Most DSP kernels can be efficiently described in a synchronous dataflow graph (SDFG) [12,13]. An SDFG lists the operations of a repetitive DSP algorithm for an iteration, where the nodes represent computations (e.g. arithmetic operators such as multiplications and additions in our case), and an edge describes the dependency between two nodes with a non-negative weight w that indicates the number of iterations of the dependency. For example, assume there are two nodes U and V connected with an edge from U to V. It describes that the computation of V requires the result from U before w_{UV} iterations. Figure 2(a) shows an illustrating SDFG for $y[n] = ax[n] + bx[n-1]$, where A and B represent multiplication and C represents addition respectively. In this example, "the result of A in the same iteration of C" and "the result of B in the previous iteration" will be added together by C to produce an output y.

High-level synthesis is the automated hardware generation from a high-level description such as the aforementioned SDFG. It performs optimizations as module selection, scheduling, allocation, and so on [14]. Assume an SDFG (i.e. that describes an iteration of a repetitive DSP algorithm) is executed in N clock cycles. For an edge from U to V, the variable U (i.e. the output of an operator or an input of the DSP algorithm), which appears at time slot $u+P_U$, needs to be kept for $D_F(U \rightarrow V) = N \cdot w_{UV} + v - u - P_U$ cycles before V consumes it [12]. D_F is calculated as the number of iterations of the dependency (w_{UV}) multiplied by the number of cycles for a single iteration (N), and adjusted by the scheduled indices v and u for V and U within the N cycles. P_U denotes the latency (due to registered I/O or pipelining) of the functional unit that

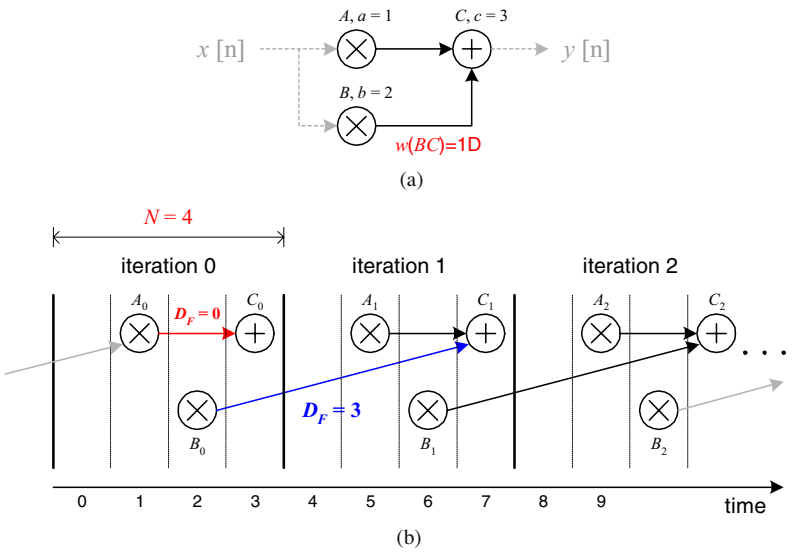


Figure 2. SDFG example.

performs U and is subtracted from D_F to obtain the final result. The time diagram of Fig. 2(a) is depicted in Fig. 2(b), where an iteration needs four clock cycles and the computations A, B, and C are scheduled into the time slots $a = 1$, $b = 2$, and $c = 3$ respectively. Assume all functional units have an identical two-cycle latency for their registered I/O ports (i.e. $P_A = P_B = P_C = 2$). Take the edge BC as an example. The result of B from a multiplier needs three-cycle buffering before it reaches the adder to perform C.

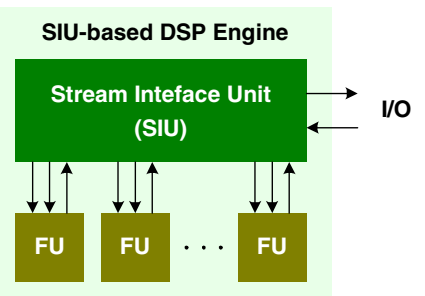


Figure 3. SIU-based computing machines.

2.2. *Configurable DSP Datapath*

Figure 3 shows the abstract model of our proposed computing engine that executes SDFG, where the data generation is completely decoupled from the computations. The concurrent functional units need not care about which data they are going to process and where to find them, for the stream interface unit (SIU) in the computing engine will deliver the required data to these functional units. Moreover, the SIU also takes the responsibility to gather the computed results for output or further processing. In other words, the SIU is the data generator of our proposed computing engine, of which the input ports are connected to the output of each functional unit and the inputs to the computing engine. Similarly, the output ports of the SIU are connected to the inputs of each functional unit and the outputs of the engine.

To execute an SDFG, the SIU needs to retrieve a datum from the output of the functional unit that performs U at time $u+P_U$ for an edge from U to V. The datum is forwarded to the functional unit that performs V after $D_F\,(U \rightarrow V) = N \cdot w_{UV} + v - u - P_U$ cycles. Generally, the latency of U (i.e. P_U) will not be smaller than two for its I/O registers, which allow a full clock cycle for SIU routing. Let's take the SDFG in Fig. 2 as an example. To perform the operation C_1 (i.e. C in the iteration 1 of cycle 4∼7), the SIU needs to retrieve the outputs of A_1 and of B_0 from the multiplier at time slots 7 and 4 respectively. B_0 is buffered in the SIU for three cycles (i.e. $D_F\,(B \rightarrow C)$) and then forwarded to the adder with A_1 at time 7.

Figure 4 shows two basic SIU architectures—one with output queues and the other with input queues, where the data are buffered at the outputs or the inputs of the functional units accordingly. The dashed lines illustrate how these two extreme architectures buffer a multiplier output for three cycles and forward it to

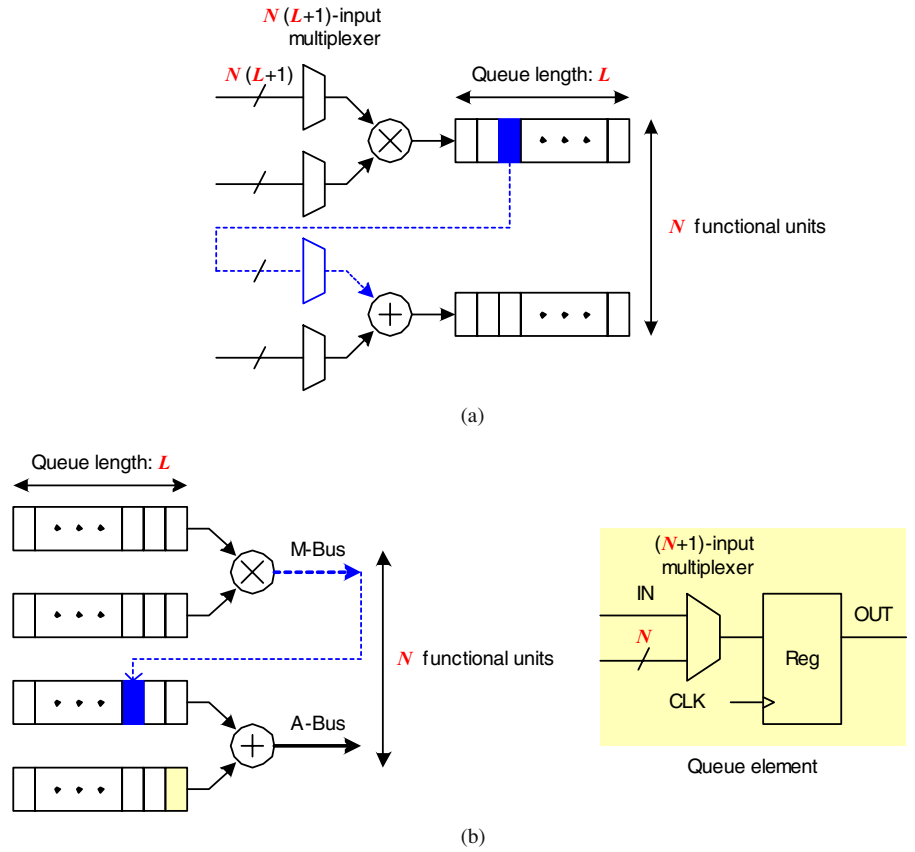


Figure 4. (a) Output-queue and (b) input-queue SIU-based architectures.

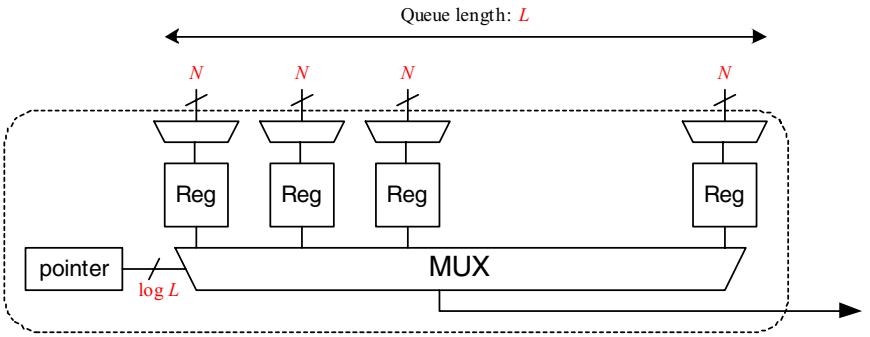


Figure 5. Static input queue.

the adder, as the B variables in the previous example. Assume the queue length is L, and the SIU with output queues will need NL registers and $2N$ $N(L+1)$-input multiplexers. On the other hand, the SIU with input queues will require $2NL$ registers and $2N(L+1)$ $(N+1)$-input multiplexers. To reduce the power dissipation, the data movements on these "dynamic" queues can be replaced by updating the pointers on the "static" queues. Figure 5 shows a static input queue, which is functionally equivalent to the dynamic input queues shown in Fig. 4(b). Note the static queues can be implemented with multi-port memory modules of

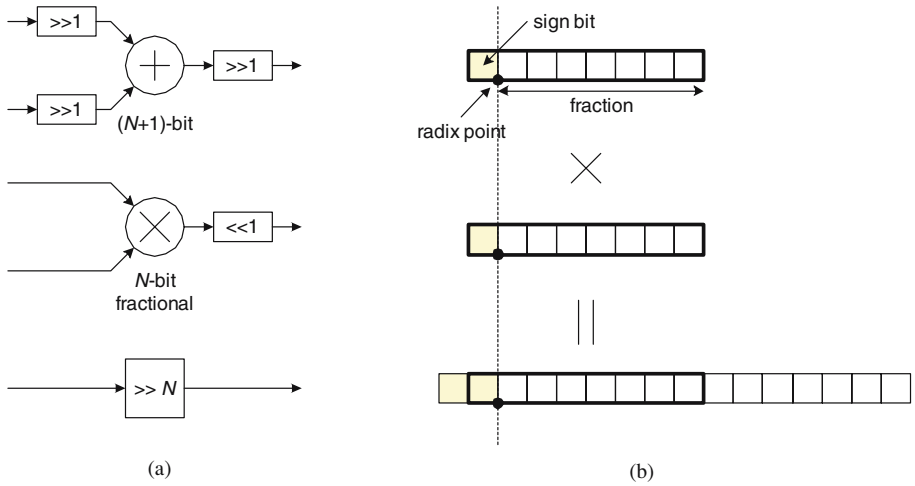


Figure 6. (a) Baseline SFP units for linear operations, and (b) 8-bit fractional multiplication.

compact layout. An input queue can be implemented using an N-write/1-read memory module and similarly, an output queue can be implemented with an N-read/1-write memory module. By the way, the memory-based SIU architectures need not access the data variables in sequence (i.e. as queue) any more.

3. Static Floating-Point Arithmetic

Digital signal processing demands high precision for quality and enough dynamic ranges to prevent overflow. The floating-point (FP) arithmetic provides the full precision of *mantissa* and a huge dynamic range with *exponent* [8], and all data variables and intermediate results have a constant wordlength with automatic rounding and normalization. Therefore, FP is very suitable for algorithm development and simulation. However, the cost of FP hardware, which dynamically aligns the operands and normalizes the results for every operation, is prohibitively high for most embedded applications, in terms of power consumption, speed, silicon area. Besides, signal ranges in most well-designed DSP algorithms are modest and do not vary very much. Embedded DSP designers usually adopt the integer arithmetic and manually scale down the variables to prevent overflow. But the conversion from FP to integer is ad-hoc, which requires extensive & time-consuming simulations, and it usually results in less optimal designs.

We have made a compromise between the FP and the integer arithmetic and propose the static FP (SFP) arithmetic in this paper. The SFP arithmetic performs data alignment and normalization for each operation as FP, but it decides and schedules the shifts at the design time instead (i.e. the so-called "static"). To improve the hardware utilization and parallelism, the internal shifters in the FP units (for operand alignment and result normalization) are shrunk as 1-bit pre-scalers and normalizers in the SFP units (SFPU). An additional barrel shifter is integrated to carry out the shifting over multiple (>1) bits. Figure 6(a) shows the baseline SFPU for linear operations, of which the hardware cost is similar to that of the integer arithmetic, except that SFP performs fractional multiplications as the FP multipliers. Figure 6(b) shows an example where the insignificant bits are rounded off autonomously [15].

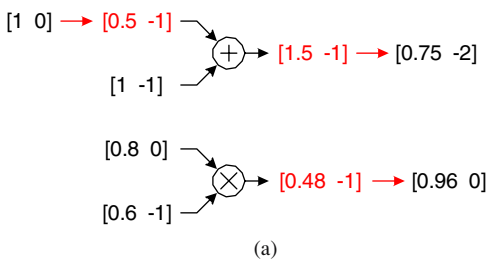
The following describes the static analysis of the SFP arithmetic on a DSP kernel represented in SDFG. First, each node of the FP SDFG (including inputs, arithmetic operators, and outputs) is associated with a peak estimation vector (PEV) [$\boldsymbol{M}$ $\boldsymbol{r}$] to perform the worst-case FP simulation, where $\boldsymbol{M}$ denotes the maximum magnitude that may occur on the node output and $\boldsymbol{r}$ is used to track the radix point (i.e. as the exponent). $\boldsymbol{M}$ should be kept as a value between 0.5 and 1 for fractional operations (with the data ranges between -1 and 1) to prevent overflow while maximizing the precision. Assume the inputs are normalized fractional numbers (i.e. PEV $=$ [1 0]), and the PEV of the remnant nodes are calculated by following the three rules:

- Keep $\boldsymbol{M}$ between 0.5 and 1 by carrying out "$\boldsymbol{M}$ divided (multiplied) by 2" and "$\boldsymbol{r}$ minus (plus) 1" simultaneously

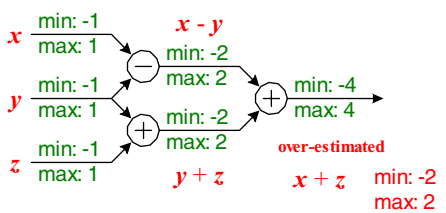
- $\boldsymbol{r}$ (radix point) should be identical before summation or subtraction
- $[\boldsymbol{M_1}\ \boldsymbol{r_1}]\times[\boldsymbol{M_2}\ \boldsymbol{r_2}] = [\boldsymbol{M_1}\times\boldsymbol{M_2}\ \boldsymbol{r_1}+\boldsymbol{r_2}]$

Figure 7(a) shows two illustrating examples of the PEV calculations. After the PEV analysis, shifts are inserted to align the data operands and to normalize the intermediate results. Note that the pre-scaling and the normalization in the two examples can be carried out in the embedded 1-bit shifters of the adder and the multiplier without invoking the additional barrel shifter. Note that the above PEV analysis may over-estimate the data ranges because it neglects the correlations between variables, such as the example given in Fig. 7(b). It can be improved by recording the intermediate variables in the affine forms [16]:

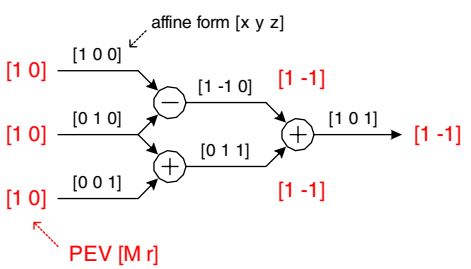
$$\sum \alpha_i \cdot x_i$$



(a)



(b)



(c)

Figure 7. Examples: (a) PEV analysis; (b) range over-estimation; and (c) affine PEV analysis.

where α_i denotes the contribution of each independent variable x_i (e.g. an input node of linear transforms). The magnitude $\boldsymbol{M}$ of a node is calculated with its corresponding α_i instead of the input $\boldsymbol{M}$ directly. Figure 7(c) shows an example for the PEV calculation with the affine data representations. By the way, for non-linear operations such as multiplication of two variables, our analysis software creates independent variables to simplify the range estimation.

4. DSP-lite Core

4.1. Core Architecture

DSP-lite is our first prototype of the proposed compact DSP core. It is equipped with the 16-bit static floating-point units (SFPU), including:

- a 17-bit adder/subtractor with two 1-bit input scalers and one 1-bit output normalizer,
- a 16-bit fractional multiplier with a 1-bit output normalizer, and
- a 16-bit barrel shifter with sign-extension capability.

A fully configurable stream interface unit (SIU) is integrated to generate data streams for the concurrent computations as described in Section 2. Table 1 compares several SIU architectures. The 2nd and the 3rd rows summarize the complexities of buffering and routing of dynamic queues. Then, the silicon areas of the SIU architectures for four functional units (i.e. three for the baseline SFPU and one for the load/store unit) are listed, and these designs are all implemented in the 0.18 µm 1P6M CMOS technology. The numbers in parentheses show the equivalent gate counts. The input queues with doubled registers do not consume twice larger area than the output queues, for their interconnections are localized and each register has only one fanout (i.e. one multiplexer

Table 1. Comparison of SIU architectures.

	Output queue		Input queue	
Storage	$\boldsymbol{NL}$ registers		$\boldsymbol{2NL}$ registers	
Interconnect	$\boldsymbol{2N}$ multiplexers; each has $\boldsymbol{N(L+1)}$ inputs		$\boldsymbol{2N(L+1)}$ multiplexers; each has $\boldsymbol{N+1}$ inputs	
Dynamic	280,900 µm^2	(16,803)	313,600 µm^2	(23,255)
Static	336,400 µm^2	(20,359)	360,000 µm^2	(26,702)
Reduced	220,900 µm^2	(16,030)	**144,400 µm^2**	(10,968)

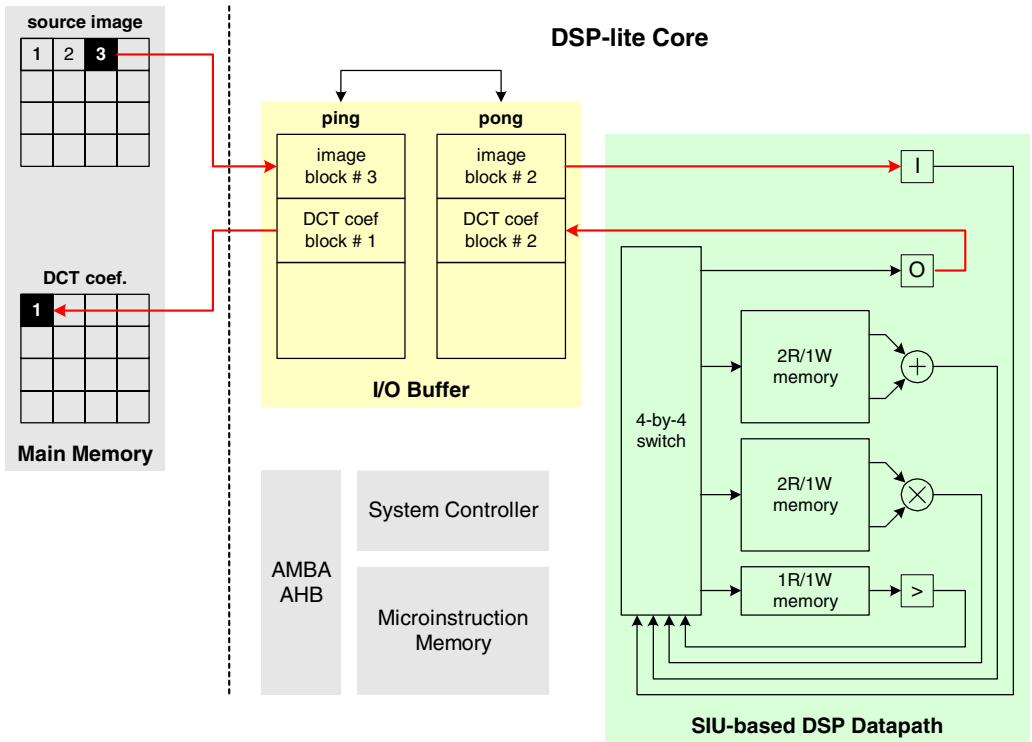


Figure 8. DSP-lite core performing the 2-D DCT.

input for the cascaded register). Finally, the last row compares two reduced SIU architectures based on 1-read/1-write memory modules, instead of those with full *N*-write/1-read (for input queue) or *N*-read/1-write (for output queue) access ports, where the memory conflicts are resolved in software via an optimal operation scheduling described later. Note that the SIU with the reduced input queues has the smallest area, for its load/store queue is redundant and has already been removed. Therefore, the reduced input queue has been chosen for our DSP-lite core.

DSP-lite has a microinstruction memory that stores the addresses to access the SIU memory modules and some control signals, such as those to enable the aligners and the normalizers of the SFPU. Besides, it contains a ping-pong I/O buffer for efficient data exchanging with the µP and external I/O devices. Moreover, the standard AMBA AHB interface is integrated to simplify the system integration. Figure 8 shows the core architecture of DSP-lite and illustrates the dataflow to perform the 2-D discrete cosine transform (DCT) [9]. When the SIU-based DSP engine performs DCT on the 2nd 8-by-8 image block, the system DMA is busy storing back to the main memory the DCT coefficients for the 1st image block and transferring the 3rd image block to the I/O buffer for the next iteration DCT.

In order to simplify the control, all memory modules of the I/O buffer and of the SIU have the mechanism to remap the virtual addresses (i.e. specified in microinstructions) to access different physical memory locations in different iterations. The address remapper consists of a stride register, a bound register, and an iteration counter, where the virtual addresses are decremented with the stride number for each iteration. Besides, the remapped addresses are modulo of a bound value specified in the bound register, and thus the addresses are rotating. Figure 9 shows an illustrating example. The virtual address “3” is remapped to the physical addresses “3”, “2”, and “1” in the 1st, 2nd, and 3rd iteration respectively, when the remapper is enabled and the stride number is set as “1”. A data item is transferred across two iterations efficiently by writing it to the virtual address “3” and retrieving it from the virtual address “5” iteratively. Note that additional memory locations are sometimes required to prevent overwriting live variables, such as the virtual address “4” in this example.

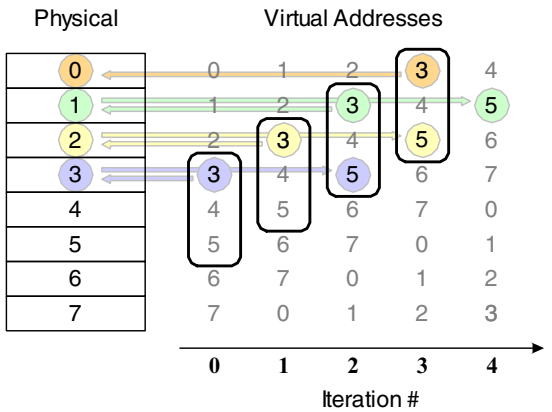


Figure 9. Memory re-mapping example.

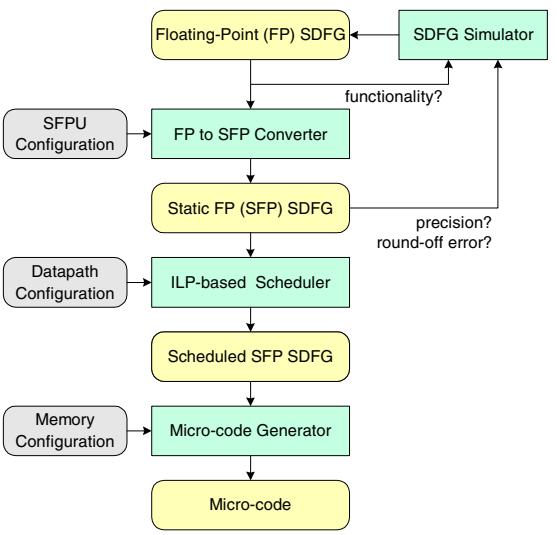


Figure 10. Software flow.

4.2. *Development Software*

Figure 10 depicts the software development flow of the DSP-lite core, and we have developed a complete tool chain to simulate the algorithmic descriptions and to compile them automatically from the FP SDFG into the SFP executables. First, the SDFG simulator is bit-true and supports both the FP and the SFP arithmetic, and the designers can develop and verify their DSP algorithms easily. Note that the FP SDFG can also be derived from the C/C++ descriptions via the SUIF compiler [17]. Then, the FP-to-SFP converter translates the FP SDFG into an SFP one by applying the PEV analysis and the shift insertion described in Section 3. The operations of the SFP SDFG (i.e. additions, multiplications, shifts, and I/O) are scheduled with integer linear programming (ILP).

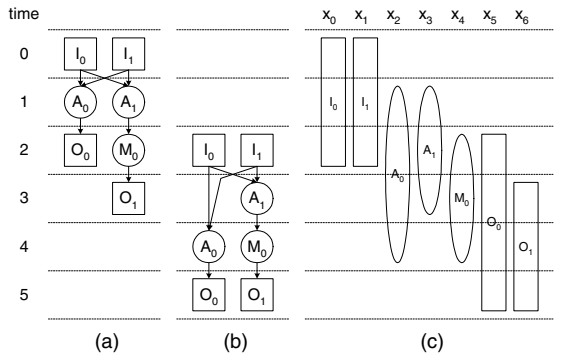


Figure 11. Example (a) ASAP; (b) ALAP; (c) scheduling range.

ILP is a formal and comprehensive approach to describe and solve the scheduling problem. In this paper, we use periodic scheduling for simplicity, where only the intra-iteration dependency is considered and the edges with non-zero weights (i.e. dependency across iterations) are first removed from the SDFG. The scheduling ranges of each operation can be determined using the basic as-soon-as-possible (ASAP) and the as-late-as-possible (ALAP) scheduling algorithms [14].

Figure 11 shows an example to construct the ILP model, where all functional units have a single-cycle latency. Let a Boolean variable $x_{i,j}$ denotes whether a node i is scheduled into the time j, and the following three types of constraints must be satisfied [12, 14].

Resource constraints (operations cannot exceed the resources)

$x_{0.0}+x_{1.0} \leq 1; x_{0.1}+x_{1.1} \leq 1; x_{0.2}+x_{1.2} \leq 1$ (for input)
$x_{2.1}+x_{3.1} \leq 1; x_{2.2}+x_{3.2} \leq 1; x_{2.3}+x_{3.3} \leq 1$ (for adder)
$x_{5.3}+x_{6.3} \leq 1; x_{5.4}+x_{6.4} \leq 1; x_{5.5}+x_{6.5} \leq 1$ (for output)

Allocation constraints (each node executes only once)

$$x_{0.0}+x_{0.1}+x_{0.2} = 1$$
$$x_{1.0}+x_{1.1}+x_{1.2} = 1$$
$$x_{2.1}+x_{2.2}+x_{2.3}+x_{2.4} = 1$$
$$x_{3.1}+x_{3.2}+x_{3.3} = 1$$
$$x_{4.2}+x_{4.3}+x_{4.4} = 1$$
$$x_{5.2}+x_{5.3}+x_{5.4}+x_{5.5} = 1$$
$$x_{6.3}+x_{6.4}+x_{6.5} = 1$$

Dependency constraints

$$x_{0.0} + 2\,x_{0.1} + 3\,x_{0.2} - 2\,x_{2.1} - 3\,x_{2.2} - 4\,x_{2.3} - 5\,x_{2.4} \leq -1$$
$$x_{1.0} + 2\,x_{1.1} + 3\,x_{1.2} - 2\,x_{2.1} - 3\,x_{2.2} - 4\,x_{2.3} - 5\,x_{2.4} \leq -1$$
$$x_{0.0} + 2\,x_{0.1} + 3\,x_{0.2} - 2\,x_{3.1} - 3\,x_{3.2} - 4\,x_{3.3} \leq -1$$

$x_{1.0} + 2\,x_{1.1} + 3\,x_{1.2} - 2\,x_{3.1} - 3\,x_{3.2} - 4\,x_{3.3} \le -1$
$2\,x_{2.1} + 3\,x_{2.2} + 4\,x_{2.3} + 5\,x_{2.4} - 3\,x_{5.2} - 4\,x_{5.3}$
$- 5\,x_{5.4} - 6\,x_{5.5} \le -1$
$2\,x_{3.1} + 3\,x_{3.2} + 4\,x_{3.3} - 3\,x_{4.2} - 4\,x_{4.3} - 5\,x_{4.4} \le -1$
$3\,x_{4.2} + 4\,x_{4.3} + 5\,x_{4.4} - 4\,x_{6.3} - 5\,x_{6.4} - 6x_{6.5} \le -1$

As mentioned in Section 4.1, the DSP-lite core uses the reduced input queues in its SIU, and it may suffer from port conflicts when multiple functional units simultaneously write their results into the same input queue. The following are the constraints to prevent multiple operations that write to an identical queue from being scheduled into the same time slot.

Port constraints

$x_{0.0} + x_{1.0} \le 1$; $x_{0.1} + x_{1.1} \le 1$; $x_{0.2} + x_{1.2} \le 1$; (for adder)

$x_{2.2} + x_{4.2} \le 1$; $x_{2.3} + x_{4.3} \le 1$; $x_{2.4} + x_{4.4} \le 1$; (for output)

We find an optimal operation schedule under the above constraints with a commercial ILP solver [18]. Lifetime analysis is then performed on the variables to allocate memory of the SIU. Whenever an input queue overflows (i.e. no free memory space), additional load/store operations are inserted to spill the variables of long lifetimes. Finally, the 64-bit microinstructions are synthesized based on the memory addresses and the control signals for the given DSP algorithm.

5. Simulation and Implementation Results

5.1. *Comparison of Round-off Error*

We have several implementations of the 2-D DCT from the independent JPEG group (IJG) [10] to evaluate the effectiveness of our proposed static floating-point (SFP) arithmetic. Table 2 summarizes the comparisons. The 2nd and the 3rd columns compare the round-off error as the PSNR over the single-precision floating-point (FP) arithmetic. The results are obtained from simulations on two natural images—Lena and Baboon. Then, the 4th column shows the number of execution cycles to perform an 8-by-8 2-D DCT on the SIU-based DSP datapaths with the baseline SFP units. Assume all functional units are single-cycle with registered I/O (i.e. with 2-cycle latency), which implies the FP units would have much longer cycle time than the integer and the SFP units.

The first three rows summarize the results for jfdctflt.c, jfdctfst.c, and jfdctint.c, which are respectively the single-precision FP, the 16-bit, and the 32-bit integer C codes from IJG. The last two rows are for the 16-bit and the 24-bit SFP respectively, both of which are derived from the FP jfdctflt.c via the affine PEV analysis. The 16-bit SFP even outperforms the hand-optimized 32-bit integer 2D-DCT from IJG. Note that the results have been improved by representing the PEV in the affine form—from 36.0510 to 40.1802 dB for Lena and from 35.9318 to 40.0468 dB for Baboon. Moreover, the 24-bit SFP has about 64 dB PSNR, which has the same maximum precision as the single-precision FP (i.e. with the 23-bit mantissa). Note that the four embedded 1-bit shifters in the SFPU for input alignment or output normalization significantly reduce the execution cycles from 784 to 656.

Table 2. Comparison of 2D-DCT on various arithmetic units.

	PSNR (dB)		
	Lena	Baboon	Cycle count
Single-Precision FP	–	–	624
16-bit Integer	29.4440	29.5183	848
32-bit Integer	33.9867	33.8020	656
16-bit SFP	**40.1802**	**40.0468**	
24-bit SFP	**64.2456**	**64.1210**	**656**

5.2. *Performance Evaluation*

In this subsection, the impacts of SIU architectures on the performance of DSP-lite are evaluated in terms of execution cycles of some popular DSP algorithms. All the functional units of the SFPU in the DSP-lite core have registered I/O ports. The adder and the shifter are single-cycle and thus with 2-cycle latency, while the multiplier is pipelined into two stages and thus has 3-cycle latency. Table 3 summarizes the results obtained with the optimal ILP-based scheduler described in Section 4.2.

As described in Section 2.2, the full SIU architectures with a complete set of multiplexers have identical external behaviors, despite whether the queues are placed at the inputs or the outputs of the functional units; or whether they are dynamic (i.e. with actual data movements) or static (i.e. with pointer updates instead), and their results are summarized in the 2nd column. Then, the execution cycles for the SIU architectures with reduced output queues and reduced input queues

Table 3. Performance evaluation of DSP-lite core.

	SIU-based architectures			ADI	TI
		Reduced		ADSP-218x	C'55
	Full	Output Q	Input Q	[11]	[19]
Lattice filter	11	12	**12**	32	12
Biquad filter	10	11	**16**	13	5
Complex FFT	207	270	**268**	874	367
2-D DCT	672	784	**688**	2,452	1,082

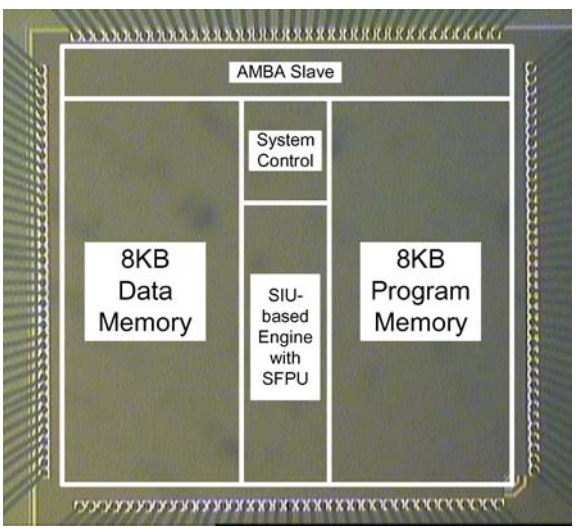


Figure 12. Die photo of the DSP-lite core.

are listed in the 3rd and the 4th columns respectively. Finally, the last two columns give the reference performances of two commercial DSP cores [11, 19], both of which have already been integrated in some dual-core processor designs. ADI ADSP-218x has similar computing resources to our DSP-lite core, including an ALU, a multiply- accumulator (MAC), and a barrel shifter, while TI C'55 has one more MAC unit. The cycle counts are all excerpted from their application notes.

The reasons that DSP-lite has such significant performance improvements over conventional DSP architectures can be summarized as follows. First, its data-driven computing engine and code generator are developed in parallel based on high-level synthesis to extensively exploit the inherent parallelism of DSP algorithms, and the performance can therefore be very close to that of customized ASIC designs. Then, the SIU enables smooth dataflow with its internal crossbar network and relatively plenty registers (note that the complexity is much less than that of a plain register file in the general-purpose processors). Moreover, the four embedded 1-bit shifters in the SFPU also help the reduction of the execution cycles. But DSP-lite does not perform the recursive DSP algorithms as well because its functional units have registered I/O and therefore longer latency (both C'55 and ADSP-218x have zero-latency functional units). However, the clock speed of DSP-lite is much faster and thus the performance is still comparable while measured in the absolute time. By the way, there exist some effective transformation techniques (e.g. lookahead [12]) that can reduce the iteration bounds and thus improve the DSP-lite performance for the algorithms with feedback loops.

5.3. Implementation and System Integration

DSP-lite is delivered as a synthesizable core. We have synthesized the design using Synopsys Design Compiler with the Artisan 0.18 μm cell library, and placed

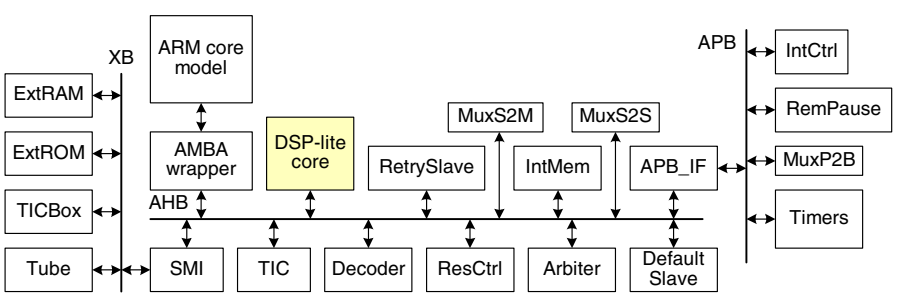


Figure 13. DSP-lite integration example on the ARM 922T platform.

and routed the net-lists using Cadence SoC Encounter. Figure 12 shows the layout of the DSP-lite core, of which the core size is 1.5×1.5 mm^2 including the standard AMBA AHB interface. The chip has been fabricated in the UMC 1P6M CMOS process, and it can operate at 314.5 MHz while consuming only 52 mW average power. Figure 13 shows an ARM-922T-based multimedia platform, and the DSP-lite core is integrated via the AMBA AHB interface. We have constructed the simulation environments both in SystemC and Seamless CVE. The SystemC model for DSP-lite is cycle-accurate, which significantly accelerates the time-consuming RTL simulation for task partitioning between ARM and DSP-lite. A JPEG encoding system has been ported on this platform successfully, where DSP-lite speeds up the 2-D DCT by a factor of 8.13 while running at the same 100 MHz as the ARM core. The dual-core platform effectively improves the JPEG encoding on an ARM-alone system by 41.78%.

6. Conclusions

This paper presents a compact DSP core for multi-core media SoC. The fully-programmable DSP core and its automatic code generator have been developed and tuned in parallel. Software techniques are extensively investigated to reduce the hardware complexity as the principles of VLIW processors. The static floating-point (SFP) arithmetic is also proposed to emulate expensive floating-point (FP) DSP operations with the hardware resources similar to those of the integer arithmetic. In the simulations, the 24-bit SFP has above 64 dB signal to round-off noise ratio over the single-precision FP under identical maximum precision (i.e. 24-bit SFP versus 23-bit mantissa in FP), and the 16-bit SFP has about 40 dB, which even outperforms the hand-optimized codes based on the 32-bit integer arithmetic. The compact DSP core with the auto-generated codes has about thrice the performance of the commercial DSP architectures found in the dual-core multimedia processors. Finally, the compact DSP core has been implemented and fabricated in the UMC 0.18 μm 1P6M CMOS process. The operating frequency can achieve 314.5 MHz, and the average power consumption is 52 mW in the meanwhile. The core size is only 1.5×1.5 mm^2 including the 16 KB on-chip memory.

We are now developing a list-scheduler to avoid the port conflicts with significantly reduced complexity than the preliminary ILP-based scheduler described in this paper. Besides, we are now investigating the distributed microinstruction memories with the JTAG-like configuration interface [20] to release some global routing. Finally, we will study the saturated arithmetic [21] in the future to further improve the round-off error of the proposed SFP arithmetic for short data wordlengths.

References

1. A. Gatherer et al, "DSP-based Architectures for Mobile Communications: Past, Present and Future," *IEEE Communications*, vol. 38, Jan. 2000, pp. 84–90.
2. *Intel PXA800F Cellular Processor Development Manual*, Intel Corp., Feb. 2003.
3. *OMAP5910 Dual Core Processor Technical Reference Manual*, Texas Instruments, Jan. 2003.
4. M. Levy, "ARM picks up performance," *Microprocessor Report*, 4/7/03-01.
5. R.A. Quinnell, "Logical combination? Convergence Products Need Both RISC and DSP Processors, but Merging them may not be the Answer," *EDN*, 1/23/2003.
6. *TriCore 2-32-bit Unified Processor Core v.2.0 Architecture—Architecture Manual*, Infineon Technology, June 2003.
7. J.L. Hennessy and D.A. Patterson, *Computer Architecture—A Quantitative Approach*, 3rd Edition, Morgan Kaufmann, 2002.
8. *IEEE Standard for Binary Floating-Point Arithmetic*, *IEEE Standard* 754, 1985.
9. W.B. Pennebaker and J.L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.
10. Independent JPEG Group, http://www.ijg.org.
11. *Digital Signal Processing Using the ADSP-2100 Family*, Analog Device Inc., 1990.
12. K.K. Parhi, *VLSI Digital Signal Processing Systems—Design and Implementation*, John Wiley & Sons, 1999.
13. E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing," *IEEE Trans. Computers*, vol. 36, 1987, pp. 24–35.
14. D.D. Gajski et al., *High Level Synthesis—Introduction to Chip and System Design*, Kluwer Academic Publisher, 1992.
15. P. Lapsley, J. Bier and E.A. Lee, *DSP Processor Fundamentals—Architectures and Features*, IEEE Press, 1996.
16. F. Fang, R. Rutenbar, M. Puschel and T. Chen, "Toward Efficient Static Analysis of Finite-precision Effects in DSP Applications via Affine Arithmetic Modeling," in *Proc. DAC*, 2003, pp. 496–501.
17. The SUIF Compiler Infrastructure, http://suif.stanford.edu/.
18. *LINDO API User 's Manual*, LINDO System Inc., 2002.
19. *TMS320C55x DSP Programmer's Guide*, Texas Instruments Inc., July 2000.
20. *IEEE Standard for In-System Configuration of Programmable Devices*, IEEE Standard 1532, 2002.
21. G.A. Constantinides, P.Y.K. Cheung and W. Luk, "Synthesis of Saturation Arithmetic Architectures," *ACM Trans. Design Automation of Electronic Systems*, vol. 8, no. 3, 2003, pp. 334–354.

Tay-Jyi Lin received the BS degree in electrical and control engineering from National Chiao Tung University, Taiwan, in 1998. He is working toward the PhD degree in the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University. His current researches include the heterogeneous computing platform for embedded multimedia systems, complexity-aware architecture design, and high-performance/low-power digital signal processors.

Hung-Yueh Lin received the BS and the MS degrees in electronics engineering from National Chiao Tung University, Taiwan, in 2002 and 2004, respectively. He is now with MediaTek, Inc., Hsinchu, Taiwan. His research interests include lightweight computer arithmetic and DSP architecture.

Chie-Min Chao received the BS degree in electronics engineering from National Chiao Tung University, Taiwan, in 2003, where he is currently pursuing his MS degree. His researches include system software development, VLSI system design, and DSP architecture.

Chih-Wei Liu received the BS and the PhD degrees in electrical engineering from National Tsing Hua University, Taiwan, in 1991 and 1999, respectively. From 1999 to 2000, he was an integrated circuit design engineer at the Electronics Research and Service Organization (ERSO) of Industrial Technology Research Institute (ITRI), Taiwan. Then, near the end of 2000, he started to work for the SoC Technology Center (STC) of ITRI as a project leader and eventually left ITRI at the end of Oct., 2003. He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Taiwan, as an assistant professor. His current research interests include SoC and VLSI system design, processor architecture, digital signal processing, digital communications, and coding theory.

Chein-Wei Jen received the BS degree from National Chiao Tung University, Taiwan, in 1970, the MS degree from Stanford University in 1977, and the PhD degree from National Chiao Tung University in 1983. From 1981 to 2004, he was with the Department of Electronics Engineering and the Institute of Electronics at National Chiao Tung University. Dr Jen was given the Outstanding Electrical Engineering Professor Award by the Chinese Institute of Electrical Engineering in 2002. He is currently the General Director of the SoC Technology Center at Industrial Technology Research Institute, the Adviser of National SoC Program, and the Managing Director of the Board of the Taiwan IC Design Society. His research interests include SoC design, VLSI architectures, multimedia processing, and design automation. He holds seven patents and has published over 50 journal and 100 conference papers in these areas.