Data Stream Generation for Concurrent Computation in VLSI Signal Processors*

Tay-Jyi Lin, Chein-Wei Jen

Dept. of Electronics Engineering
National Chiao Tung University, Hsinchu, Taiwan
Fax: 886-3-5710580, Email: {tjlin, cwjen}@ee.nctu.edu.tw

Abstract

Signal processing usually requires extremely high computing power. Fortunately, with advance in VLSI technology, the required performance could be achieved with more functional units performing concurrent computations on a chip. Supplying demanding data streams to these computational units soon becomes the system-performance bottleneck because of slow off-chip I/O and memory with less improvement speed. We have proposed a data stream generation (DSG) scheme that explores data reuse property existing in most signal processing algorithms while supplying appropriate data sequences. This scheme can make the VLSI signal processors much more latency and cost efficient. In the illustrating example, our DSG supplies the specified streams with 4-cycle setup time at the beginning (latency), instead of 48 cycles for each block in conventional FIFO approaches and only requires 1/6-storage elements.

1. Introduction

Standard microprocessors today cannot provide a cost-efficient platform with enough computing power for dramatically growing multimedia applications with intensive-computation requirements. Application-specific circuits are an obvious choice, but lack flexibility. Digital signal processors with separate data/program accesses, adapted interconnections and functional units could only satisfy speech-/audio-processing requirements. With an embedded micro-controller and customized hardware accelerators, application specific instruction-set processors (ASIPs) [1] with both flexibility and computing power are very attractive. Most VLSI signal-processing applications adopt this ASIP model as the basic underlying platform for multimedia computing.

Conventional ASIPs consist of one embedded micro-controller to synchronize all operations and handle some sequential tasks, some customized hardware accelerators such as array coprocessors, SIMD and MMX-like datapath that explore the concurrency in DSP algorithms to provide computing power boost. A memory subsystem is also included as shown in figure 1(a). Data-hungry concurrent computations usually block the memory accesses by other functional units, or lose some performance themselves due to bandwidth limitation on system bus. Address generation and data formation, possibly at bit level, might exhaust the micro-controller.

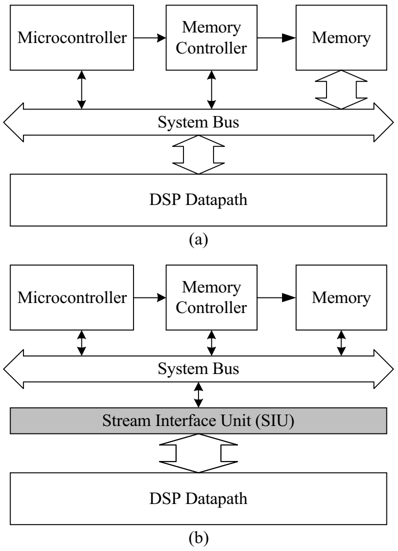


Figure 1 (a) Conventional ASIP Model (b) Modified ASIP Model with our Proposed SIU

*This work was supported by the NSC, Taiwan under Grant NSC89-2215-E009-003

A stream interface unit (SIU) is inserted as an alternative [2], to decouple the accelerating DSP datapath from the system bus as shown in figure 1(b). The customized SIU datapath handles data format conversions if necessary and support a very high data rate to the accelerating DSP datapath while maintaining the lowest system bus occupation and hence less off-chip data acquisition. The asymmetric data rate property at the SIU I/O comes from massive data reuse.

Design methodologies for algorithm-specific array processors in accelerating DSP datapath were well developed but performance might be limited by I/O constraints. Our proposed data stream generation (DSG) scheme can release these I/O constraints via the customized SIU that could be automatically synthesized. In next section we describe the DSG problem. SIU implementation issues and our proposed two-stage procedure for automatic circuit generation are summarized in section 3. Section 4 concludes our work.

2. Data Stream Generation

Our stream interface unit (SIU) supplies huge data streams required by concurrent computation while keeping low data acquisition rate from the system bus. The asymmetric SIU I/O-rate could be in the form of different clock rates, different data widths or both. For simplicity, we assume the single-input / multiple-output under one synchronous clock, i.e. the SIU receives scalar inputs and generates array outputs.

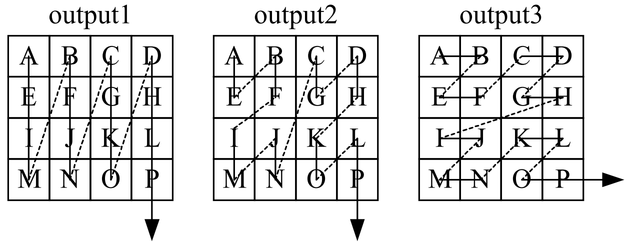


Figure 2 Output Sequence Specification

Suppose there are three tasks in sequence, which access identical 4*4 data block with elements A~P but in different orders as figure 2 — three widely-used scanning ways in image applications. Parallelizing these tasks directly triples the system bandwidth. We can eliminate redundant memory accesses to keep the same bandwidth as the original sequential hardware by introducing a small latency and some storage elements. The proposed SIU for 3-channel data stream generation is shown in figure 3. The hardware cost could be measured as the number of storage elements together with the routing complexity for the MUX-based interconnection network.

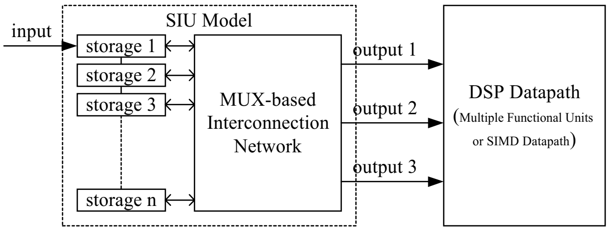


Figure 3 3-Channel Data Stream Generation

We first assume that the SIU acquires data in raster-scan order from the system bus. Each channel is equivalent to an individual data format conversion (DFC) problem. The DFC methodology, which applies lifetime analysis followed by forward-backward register allocation [3], is generalized here to handle asymmetric I/O rates with multiple data outputs. To maintain a smooth data flow into the parallel functional units, a small latency is introduced, which equals the most negative difference between T_{in} (data arriving time) and T_{zl} (data consumed time with zero latency). Lifetime for each variable is then derived from its *birth* (T_{in}) till *death* (the last consumed time of the three channels, i.e. MAX [T_{out}]). Minimum register count is then obtained, which equals the maximum number of concurrent live variables, via lifetime analysis. The 3-channel SIU requires 9-cycle latency and 13 storage elements as shown in figure 4. Internal copies of identical data samples are not allowed for simplicity.

3. Stream Interface Unit (SIU) Implementation

Following (1) *I/O specification* (2) *circuit mapping*, the SIU could be systematically generated. More sophisticated I/O specification than raster-scan can efficiently reduce latency and SIU complexity with some assistance from the microcontroller and the on-chip memory subsystem. Minimizing the storage and routing complexity in the MUX-based interconnection network while keeping an acceptable low latency leads the I/O specification to an NP-complete problem. A linear-time heuristic scheduler is shown to efficiently reduce almost half latency and storage elements in figure 5 while re-transmitting the same data item is not allowed for minimum bandwidth requirement.

Forward-backward register allocation [3] allocates the variables to the storage elements that are initially self-organized as a FIFO. The routing problem is significantly simplified based on this underlying FIFO structure with some additional feedback paths. Figure 6 shows the allocation table and the SIU hardware for scheduled input. The SIU hardware consists of 8 storage elements; one 7-input, two 6-input MUX for output ports and only one 2-input MUX for data feedback in the MUX-based interconnection network. Various register allocation schemes [4-7] could be utilized here as well with some adaptation of the input scheduling heuristics.

The methodology for SIU generation can be easily packaged as a soft-IP (silicon Intellectual Property [11]) that finds applications in many fields. Once the data streams are specified, tradeoff among latency, register count and interconnection complexity can be evaluated via various heuristic schedulers and register allocation schemes. Automatic pattern generation for functional verification and testing can be also supported. Users can easily instantiate our SIU module in the top-level Verilog design.

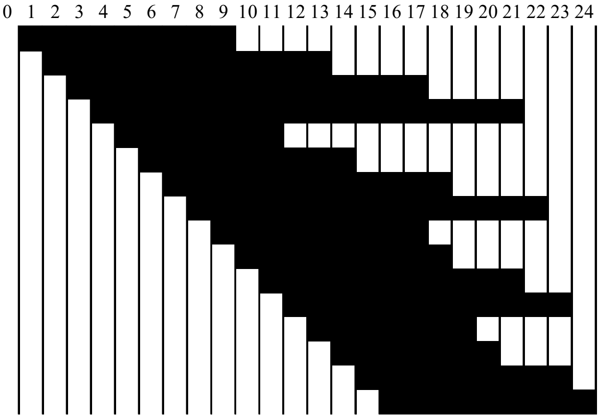
4. Conclusion

The concept of System-on-Chip (SoC) and increasing gates available makes heterogeneous architectures more practical. We propose an alternative general computation model for embedded systems with data-flow intensive tasks in this paper. The proposed data stream generation (DSG) scheme makes data-driven computing blocks much more efficient in terms of power, speed, and memory usage with an additional customized stream interface unit (SIU). A two-step procedure with simplified algorithms is summarized for automatic SIU circuit generation. We use a 3-channel SIU example to demonstrate the validity of our DSG scheme. The resulting hardware shows great improvement both in area and latency over conventional heterogeneous VLSI signal processors.

Reference

1. I. Kuroda, T. Nishitani, “Multimedia processors,” *Proceedings of the IEEE*, June 1998
2. M. Schonfeld, M. Schwiegershausen, P. Pirsch, “Synthesis of Intermediate Memories for the Data Supply to Processor Arrays,” *Algorithms and Parallel VLSI Architectures II*, 1992
3. K. K. Parhi, “Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation,” *IEEE Trans. Circuits Syst. II*, July 1992
4. Kazuhito Ito, et al. “ILP-Based Cost-Optimal DSP Synthesis with Module Selection and Data Format Conversion,” *IEEE Trans. VLSI*, December 1998
5. K. Srivatsan, C. Chakrabarti, and L. Lucke, “Low Power Data Format Converter Design Using Semi-Static Register Allocation,” *ICCD*, 1995
6. J. Bae, V. K. Prasanna, “Synthesis of Area-Efficient and High Throughput Rate Data Format Converters,” *IEEE Trans. VLSI*, December 1998
7. M. Majumdar, K. K Parhi, “Design of Data Format Converters Using Two-Dimensional Register Allocation,” *IEEE Trans. Circuits Syst. II*, April 1998
8. Francky Catthoor, et al, “Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design,” *Kluwer Academic Publishers*, 1998
9. Preeti Ranjan Panda, et al, “Memory Issues in Embedded Systems-On-Chip,” *Kluwer Academic Publishers*, 1999
10. K. K. Parhi, "VLSI Digital Signal Processing Systems: Design and Implementation," *John Wiley & Sons*, 1999
11. M. Keating, P. Bricaud, "Reuse Methodology Manual for System-on-a-Chip Designs," *Kluwer Academic Publisher*, 1998

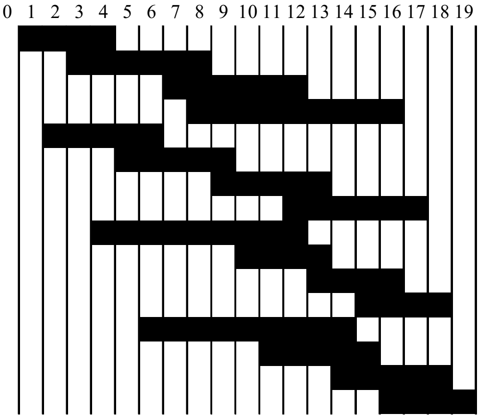
	Tinput	Tzl1	Tdiff1	Tout1	Tzl2	Tdiff2	Tout2	Tzl3	Tdiff3	Tout3	Life Period	
A	0	0	0	9	0	0	9	0	0	9	0	9
B	1	4	3	13	2	1	11	1	0	10	1	13
C	2	8	6	17	8	6	17	4	2	13	2	17
D	3	12	9	21	10	7	19	5	2	14	3	21
E	4	1	-3	10	1	-3	10	2	-2	11	4	11
F	5	5	0	14	3	-2	12	3	-2	12	5	14
G	6	9	3	18	9	3	18	6	0	15	6	18
H	7	13	6	22	11	4	20	7	0	16	7	22
I	8	2	-6	11	4	-4	13	8	0	17	8	17
J	9	6	-3	15	6	-3	15	9	0	18	9	18
K	10	10	0	19	12	2	21	12	2	21	10	21
L	11	14	3	23	14	3	23	13	2	22	11	23
M	12	3	$\boxed{-9}$	12	5	-7	14	10	-2	19	12	19
N	13	7	-6	16	7	-6	16	11	-2	20	13	20
O	14	11	-3	20	13	-1	22	14	0	23	14	23
P	15	15	0	24	15	0	24	15	0	24	15	24



number of live variables: 1 2 3 4 5 6 7 8 9 9 10 10 11 11 11 12 12 10 8 7 6 4 3 1
loop overhead: 0 1 2 3 4 5 6 7 8
concurrent live variables: 1 2 3 4 5 6 7 8 9 9 10 10 11 11 11 12 $\boxed{13}$ 12 11 11 11 10 10 9

Figure 4 Lifetime Analysis for Raster-Scan Input: latency = |MIN (Tdiff1~3)| and life period = Tinput ~ MAX (Tout1~3). The SIU needs 13 storage elements and 9-cycle latency.

	Tinput	Tzl1	Tdiff1	Tout1	Tzl2	Tdiff2	Tout2	Tzl3	Tdiff3	Tout3	Life Period	
A	0	0	0	4	0	0	4	0	0	4	0	4
B	2	4	2	8	2	0	6	1	-1	5	2	8
C	6	8	2	12	8	2	12	4	-2	8	6	12
D	7	12	5	16	10	3	14	5	-2	9	7	16
E	1	1	0	5	1	0	5	2	1	6	1	6
F	4	5	1	9	3	-1	7	3	-1	7	4	9
G	8	9	1	13	9	1	13	6	-2	10	8	13
H	11	13	2	17	11	0	15	7	$\boxed{-4}$	11	11	17
I	3	2	-1	6	4	1	8	8	5	12	3	12
J	9	6	-3	10	6	-3	10	9	0	13	9	13
K	12	10	-2	14	12	0	16	12	0	16	12	16
L	14	14	0	18	14	0	18	13	-1	17	14	18
M	5	3	-2	7	5	0	9	10	5	14	5	14
N	10	7	-3	11	7	-3	11	11	1	15	10	15
O	13	11	-2	15	13	0	17	14	1	18	13	18
P	15	15	0	19	15	0	19	15	0	19	15	19



number of live variables: 1 2 3 4 4 5 5 6 6 6 7 8 7 6 6 6 4 3 1
loop overhead: 1 2 3
concurrent live variables: 1 2 3 4 4 5 5 6 6 6 7 $\boxed{8}$ 7 6 6 6 5 5 4

Figure 5 Lifetime Analysis for Scheduled Input: "*Small MIN [T_{zl}] first*" reduces latency. When variables with identical MIN [T_{zl}] exist, latency does not depend on how these variables are scheduled. "*Small MAX [T_{zl}] first*" schedules late-dying ones last to efficiently decrease their lifetime and hence storage requirement. The scheduled-input SIU only requires 8 storage elements and 4-cycle latency.

T	input	reg 1	reg 2	reg 3	reg 4	reg 5	reg 6	reg 7	reg 8	output 1	output 2	output 3
0	A											
1	E	A										
2	B	E	A									
3	I	B	E	A								
4	F	I	B	E	A(1,2,3)					A	A	A
5	M	F	I	B(3)	E(1,2)					E	E	B
6	C	M	F	I(1)	B(2)	E(3)				I	B	E
7	D	C	M(1)	F(2,3)	I	B				M	F	F
8	G	D	C(3)	M	F	I(2)	B(1)			B	I	C
9	J	G	D(3)	C	M(2)	F(1)	I			F	M	D
10	N	J(1,2)	G(3)	D	C	M		I		J	J	G
11	H(3)	N(1,2)	J	G	D	C	M		I	N	N	H
12	K	H	N	J	G	D	C(1,2)	M	I(3)	C	C	I
13	O	K	H	N	J(3)	G(1,2)	D		M	G	G	J
14	L	O	K(1)	H	N			D(2)	M(3)	K	D	M
15	P	L	O(1)	K	H(2)	N(3)			D	O	H	N
16		P	L	O	K(2,3)	H			D(1)	D	K	K
17		■	P	L(3)	O(2)		H(1)			H	O	L
18		■	■	P	L(1,2)	O(3)				L	L	O
19		■	■	■	P(1,2,3)					P	P	P

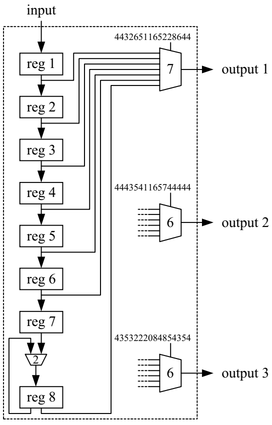


Figure 6 Forward Backward Register Allocation for Scheduled Input: The MUX-based network requires one 7-input, two 6-input MUX in I/O and one 2-input MUX in datapath.