J Sign Process Syst
DOI 10.1007/s11265-008-0162-1

Architecture and Evaluation of an Asynchronous Array of Simple Processors

Zhiyi Yu · Michael J. Meeuwsen · Ryan W. Apperson · Omar Sattari · Michael A. Lai · Jeremy W. Webb · Eric W. Work · Tinoosh Mohsenin · Bevan M. Baas

Received: 13 March 2007 / Accepted: 17 January 2008

Abstract This paper presents the architecture of an asynchronous array of simple processors (AsAP), and evaluates its key architectural features as well as its performance and energy efficiency. The AsAP processor calculates DSP applications with high energy-efficiency, is capable of high-performance, is easily scalable, and is well-suited to future fabrication technologies. It is composed of a two-dimensional array of simple single-issue programmable processors interconnected by a reconfigurable mesh network. Processors are designed to capture the kernels of many DSP algorithms with very little additional overhead. Each processor contains its own tunable and haltable clock oscillator, and processors operate completely asynchronously with respect to each other in a globally asynchronous locally synchronous (GALS) fashion. A 6×6 AsAP array has been designed and fabricated in a 0.18 μm CMOS technology. Each processor occupies 0.66 mm^2, is fully functional at a clock rate of 520–540 MHz at 1.8 V, and dissipates an average of 35 mW per processor at 520 MHz under typical conditions while executing applications such as a JPEG encoder core and a complete IEEE 802.11a/g wireless LAN baseband transmitter. Most processors operate at over 600 MHz at 2.0 V. Processors dissipate 2.4 mW at 116 MHz and 0.9 V. A single AsAP processor occupies 4% or less area than a single processing element in other multi-processor chips. Compared to several RISC processors (single issue MIPS and ARM), AsAP achieves performance 27–275 times greater, energy efficiency 96–215 times greater, while using far less area. Compared to the TI C62x high-end DSP processor, AsAP achieves performance 0.8–9.6 times greater, energy efficiency 10–75 times greater, with an area 7–19 times smaller. Compared to ASIC implementations, AsAP achieves performance within a factor of 2–5, energy efficiency within a factor of 3–50, with area within a factor of 2.5–3. These data are for varying numbers of AsAP processors per benchmark.

Keywords array processor · chip multi-processor · digital signal processing · DSP · globally asynchronous locally synchronous · GALS · many-core · multi-core · programmable DSP

Z. Yu (✉) · M. J. Meeuwsen · R. W. Apperson · O. Sattari · M. A. Lai · J. W. Webb · E. W. Work · T. Mohsenin · B. M. Baas
ECE department, UC Davis, Davis, CA 95616 USA
e-mail: zhyyu@ucdavis.edu

B. M. Baas
e-mail: bbaas@ucdavis.edu

1 Introduction

Applications that require the computation of complex DSP workloads are becoming increasingly commonplace. These applications often comprise multiple DSP tasks and are found in applications such as: wired and wireless communications, multimedia, sensor signal processing, and medical/biological processing. Many are embedded and are strongly energy-constrained. In addition, many of these workloads require very high throughputs and often dissipate a significant portion of the system power budget and are therefore of considerable interest.

Increasing clock frequencies and an increasing number of circuits per chip has resulted in modern chip

performance being limited by power dissipation rather than circuit constraints. This implies a new era of high-performance design that must now focus on energy-efficient implementations [1]. Future fabrication technologies are expected to have large variations in devices and wires, and "long" wires are expected to significantly reduce maximum clock rates. Therefore, architectures that enable the elimination of long high-speed wires will likely be easier to design and may operate at higher clock rates [2].

The asynchronous array of simple processors (AsAP) computes the aforementioned complex DSP application workloads with high performance and high energy-efficiency, and is well suited for future technologies. The AsAP system comprises a two-dimensional array of simple programmable processors interconnected by a reconfigurable mesh network. Processors are each clocked by fully independent haltable oscillators in a globally asynchronous locally synchronous (GALS) [3] fashion. Several of AsAP's key features distinguish it from other broadly similar work:

- *A chip multiprocessor architecture* achieves high performance through parallel computation. Many DSP applications are composed of a collection of cascaded DSP tasks, so an architecture that allows the parallel computation of independent tasks will likely be more efficient.
- *Small memories and simple single-issue architecture* for each processor achieves high energy efficiency. Since large memories—which are normally used in modern processors [4, 5]—dissipate significant energy and require larger delays per memory transaction, architectures that minimize the need for memory and keep data near or within processing elements are likely to be more efficient. Along with reduced memory sizes, the datapath and control logic complexity of AsAP are also reduced.
- *GALS clocking style* is suitable for future fabrication technologies and can achieve high energy efficiency due to the fact that global clock circuits have become increasingly difficult to design and they consume significant power.
- *Nearest neighbor communication* is used to avoid global wires to make it suitable for future fabrication technologies, due to the fact that global chip wires will dramatically limit performance if not properly addressed since their delay is roughly constant when scaled [2].

A prototype 6 × 6 AsAP chip has been implemented in 0.18 μm CMOS and is fully functional [6]. In this paper, we discuss AsAP's architectural design and investigate how the key features affect system results. In addition, we present a thorough evaluation of its performance and energy efficiency for several DSP applications.

2 The AsAP Processor System

2.1 Architecture of the AsAP Processor

The AsAP array consists of a large number of simple uniform processing elements operating asynchronously with respect to each other and connected through a reconfigurable network. The processors are optimized to efficiently compute DSP algorithms individually as well as in conjunction with neighboring processors. Figure 1 contains diagrams of the fabricated processing array and a single AsAP processor.

Each AsAP processor is a simple single-issue processor with a 64-word 32-bit instruction memory (IMEM),

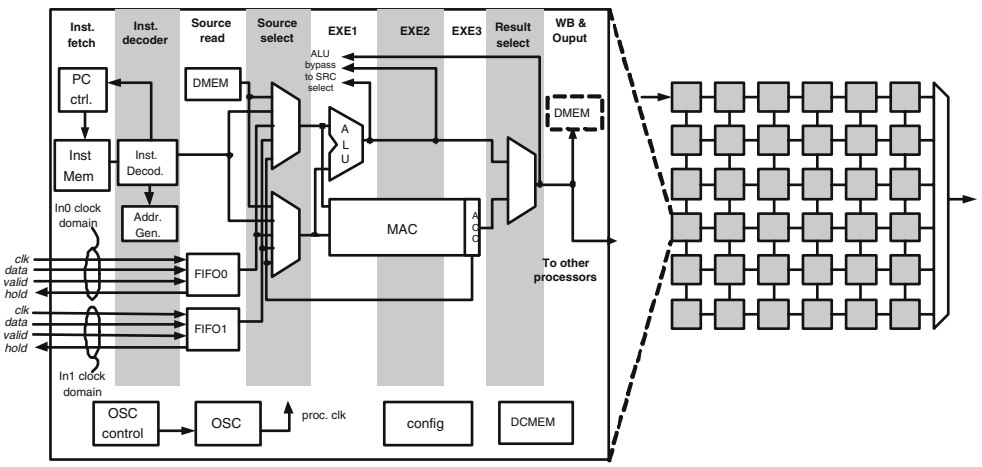


Figure 1 Block diagram of an AsAP processor and the 6 × 6 chip. Vertical gray bars indicate the nine pipeline stages.

Table 1 AsAP 32-bit instruction types and fields.

Instruction type	6 bits	8 bits	8 bits	8 bits	2 bits
General	opcode	dest	src1	src2	NOPs
Immediate	opcode	dest	immediate		NOPs
Branch	opcode	–	–	target	NOPs

a 128-word 16-bit data memory (DMEM), a dynamic configuration memory (DCMEM), a 16×16-bit multiplier with a 40-bit accumulator, a 16-bit ALU, and four address generators. It utilizes a memory-to-memory architecture with no register file. No support is provided for branch prediction, out of order execution, or speculative operation. During the design phase, hardware was added only when it significantly increased performance and/or energy-efficiency for our benchmarks. A nine stage pipeline is implemented as shown in Fig. 1. All control signals are generated in the instruction decode stage, and pipelined appropriately. Interlocks are not implemented in hardware, so all code is scheduled prior to execution by the compiler.

2.1.1 Instruction Set

AsAP supports 54 32-bit instructions with three broad instruction formats. A summary of the 54 instructions is given in Tables 1 and 2. *General* instructions select two operands from memories, accumulator, FIFOs, and three ALU bypass routes; and they select one destination from memories, accumulator and output ports. *Immediate* instructions receive input from a 16-bit immediate field. *Branch* instructions include a number of conditional and unconditional branch functions. Two bits in each instruction define how many NOP operations (from 0 to 3) should follow after instruction processing, which allows inserting NOPs to avoid pipeline hazards without requiring additional NOP instructions.

Other than a bit-reverse instruction and a bit-reverse mode in the address generators, no algorithm-specific instructions or hardware are implemented. While single-purpose hardware can greatly speed computation for specific algorithms, it can prove detrimental to the performance of a complex multi-algorithmic system and limits performance for future presently-unknown algorithms—which is one of the key domains for programmable processors.

2.1.2 Data Addressing

AsAP processors fetch data at pipeline stage *Mem Read*, using the addressing modes listed in Table 3. Three methods are supported to address data memory. *Direct* memory addressing uses immediate data as the address to access static memory locations; four *address pointers* access memory according to the value in special registers located in DCMEM; and four *address generators* provide automatic addressing with special-purpose hardware to accelerate many tasks. In addition to the data memory, AsAP processors can also fetch data from another six locations: (1) short immediate data (6 bits) can be used in dual-source instructions, (2) long immediate data (16 bits) can be used in the move immediate instruction, (3) the DCMEM's configuration information can be read or written by instructions, (4) three bypass paths from the ALU and MAC units can be used as sources to accelerate execution, (5) the two processor input FIFOs are available as general instruction sources, and (6) the lowest 16 bits of the accumulator register can also be a source for execution.

Figure 2 shows the logic diagram for one address generator. Each address generator contains a *count* register which is used as the memory pointer, and several inputs define how to change its value after each memory access. *Start_addr* defines the start address of the *count* register. When the counter is enabled (*enable* = 1), it will be increased or decreased (determined by *direction*) by the amount of the value *stride*. The *count* register is reloaded to the start address when it reaches the end address (*end_addr*). The other control signals are primarily used to accelerate FFTs. Each address generator occupies about 3700 μm^2 in a 0.18 μm technology, and the four address generators occupy only 2% of the processor's area.

2.1.3 Completely Independent Clocking and Circuits for Crossing Asynchronous Clock Domains

Each processor has its own digitally programmable clock oscillator which occupies only about 0.5% of the processor's area. There are no phase-locked loops (PLLs), delay-locked loops (DLLs), or global frequency or phase-related signals, and the system is

Table 2 Classes of the 54 supported instructions.

Instruction class	Number of instructions
Addition	7
Subtraction	7
Logic	11
Shift	4
Multiply	2
Multiply–accumulate	6
Branch	13
Miscellaneous	4

Table 3 Data fetch addressing modes.

Addressing mode	Example	Meaning
Direct	Move Obuf DMEM 0	Obuf ← DMEM[0]
Address pointer	Move Obuf aptr0	Obuf ← DMEM[DCMEM]
Address generator	Move Obuf ag0	Obuf ← DMEM[generator]
Short immediate	Add Obuf #3 #3	Obuf ← 3+3
Long immediate	Move Obuf #256	Obuf ← 256
DCMEM	Move Obuf DCMEM 0	Obuf ← DCMEM[0]
Bypassing	Move Obuf regbp1	Obuf ← first bypass
FIFOs	Move Obuf Ibuf0	Obuf ← FIFO 0
ACC	Move Obuf Acc	Obuf ← ACC[15:0]

fully GALS. While impressive low clock skew designs have been achieved at multi-GHz clock rates, the effort expended in clock tree management and layout is considerable [7]. Placing a clock oscillator inside each processor reduces the size of the clock tree circuit to a fraction of a square millimeter—the size of a processing element. Large systems can be made with arrays of processing elements with no change whatsoever to clock trees (that are wholly contained within processing elements), simplifying overall design complexity and scalability.

The reliable transfer of data across unrelated asynchronous clock domains is accomplished by dual-clock FIFOs [8]. The FIFO's write clock and data are supplied in a source-synchronous fashion by the upstream processor and its read clock is supplied by the downstream processor—which is the host for the dual-clock FIFO in AsAP.

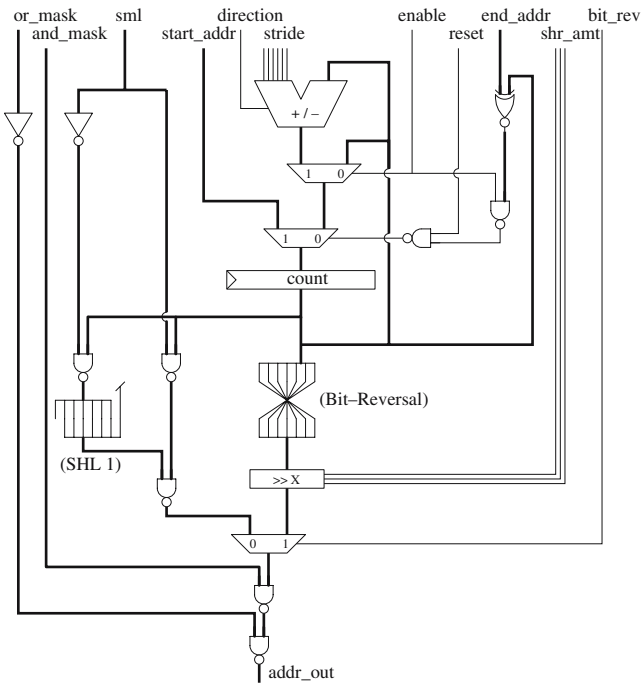


Figure 2 Address generator; *thin lines* represent one-bit wires, and *thick lines* represent seven-bit wires.

Special clock control circuits enable processing elements to power down completely—dissipating leakage power only—if no work is available for nine clock cycles. The local oscillator is fully restored to full speed in less than one cycle after work again becomes available.

2.1.4 Reconfigurable Two-Dimensional Mesh Network

Processors connect via a configurable two-dimensional mesh. To maintain link communication at full clock rates, inter-processor connections are made to nearest-neighbor processors only. A number of architectures including wavefront [9], RAW [10], and TRIPS [11], have specifically addressed this concern and have demonstrated the advantages of a tile-based architecture. AsAP's nearest neighbor connections result in no high-speed wires with a length greater than the linear dimension of a processing element. The inter-processor delay decreases with advancing fabrication technologies and allows clock rates to scale upward. Longer distance data transfers in AsAP are handled by routing through intermediary processors or by "folding" the application's data flow graph so that communicating processing elements are placed adjacent or near each other—for example, the *Pilot Insert* processor and the first *G.I. Wind.* processor in Fig. 5b.

Each AsAP processor has two asynchronous input data ports and can connect each port to any of its four nearest neighboring processors. Because changing

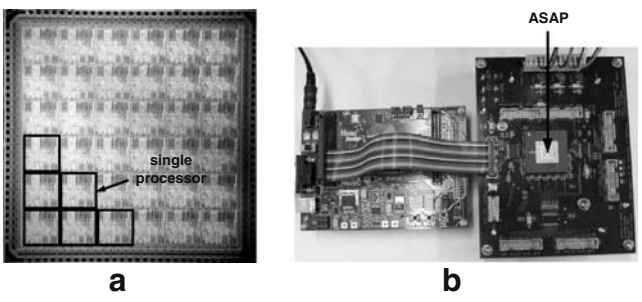


Figure 3 Micrograph of the **a** 6 × 6 AsAP chip and **b** its test environment.

active clock signals can cause runt clock pulses, a processor may change its input connection only during times when both input clocks are guaranteed to both be low—which is normally only during power-up. On the other hand, output port connections can be changed among any combination of the four neighboring processors at any time through software.

2.2 AsAP Implementation

The first generation AsAP 6 × 6 processor array has been implemented using TSMC 0.18 μm CMOS technology [6]. The left part of Fig. 3 shows the die micrograph. A standard cell based design flow was used from verilog source code. All circuits were synthesized, except the programmable oscillator. A single processor tile and the entire chip were placed and routed by CAD tools.

The right part of Fig. 3 shows the test environment for the AsAP prototype including a printed circuit board hosting an AsAP processor and a supporting FPGA board to interface between AsAP and a host PC. AsAP's SPI-style serial port receives configuration information and programs for each processor.

Table 4 Required instruction memory and data memory sizes for various DSP tasks on a simple single-issue processor.

Task	IMem size (words)	DMem size (words)
16-Tap FIR filter	6	33
Level-shifting for JPEG	8	1
8-Point DCT	40	16
8 × 8 two-dimensional DCT	154	72
Quantization for 64 elements	7	66
Zig-zag re-ordering for JPEG	68	64
Huffman encoding for JPEG	203	334
Scrambling for 802.11a/g	31	17
Padding OFDM bitstream	49	25
Convolutional coding ($k = 7$)	29	14
Interleaving 1 for 802.11a/g	35	30
Interleaving 2 for 802.11a/g	51	31
Modulation for BPSK, QPSK, 16QAM, 64QAM	53	33
Pilot insertion for OFDM	47	68
Training symbol generation for 802.11a/g	31	76
64-point complex FFT	97	192
Guard interval insertion for OFDM	44	74
2× upsampling + 21-tap Nyquist FIR filter	40	128
Bubble sort	20	1
N-element merge sort	50	*N*
Square root	62	15
Exponential	108	32

```
void main() {
  int i;
  int a, b, c;
  sat int d;          /* saturating integer */
  while(1) {          /* loop */
    a = Ibuf0;        /* read value from FIFO 0 */
    b = Ibuf1;        /* read value from FIFO 1 */
    c = (a + b) >> 1;/* AddHigh instruction */
    for (i = 0; i < 10; i++) {
      d = c + i;      /* saturating instruction */
      OBuf = d;       /* OBuf is proc.'s output */
    }
  }
}
```

Figure 4 An example C language program for an AsAP processor.

2.3 Software

Programming the AsAP processor presents significant challenges. Programming involves taking advantage of all levels of parallelism easily available to simplify the coding of small kernels, including task-level parallelism, data-level parallelism, and address-data parallelism. Partly due to the natural partitioning of applications by task-level parallelism, we have found the task less challenging than first expected. This is supported by data in Table 4 showing the memory requirements of common DSP tasks.

A high level language (which we call called AsAP-C) and its corresponding compiler were developed to generate code for each individual AsAP processor. AsAP-C contains most standard C language functions such as arithmetic calculations (addition, subtraction, multiplication, etc.), logic calculations (AND, OR, NOT, etc.), and control functions (while loops, for loops, etc.). A saturating integer type is defined to support DSP integer calculations which are commonly used in high level DSP languages [12]. Additionally, the language contains several functions specific for AsAP such as FIFO reads and direct inter-processor communication. Both inputs and outputs are mapped into the language through the reserved variable names: `Ibuf0`, `Ibuf1`, and `Obuf`. Figure 4 shows one example of an AsAP-C program which fetches data from two FIFOs and sends its result to the processor's output port.

The job of programming processors also includes the mapping of processor graphs to the two-dimensional planar array. While this is normally done at compile time, an area of current work is tools for the automatic mapping of graphs to accommodate rapid programming and to recover from hardware faults and extreme variations in circuits, environment, and workload.

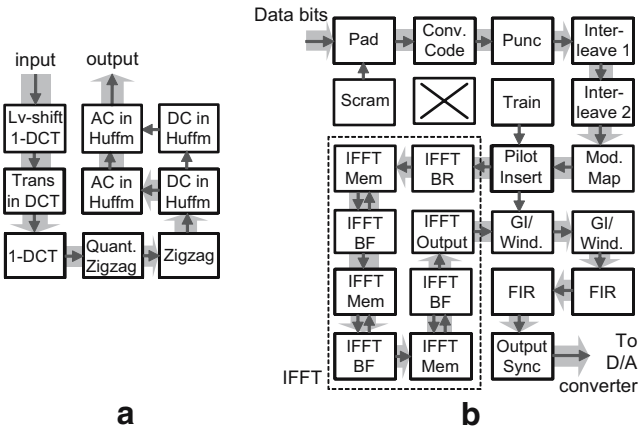


Figure 5 Block diagram of the **a** 9-processor JPEG encoder and **b** 22-processor 802.11a/g implementation. *Thin arrows* show all paths and *wide arrows* show the primary data flow. The processor marked with an "×" is unused and powered down.

2.4 Task and Application Implementations

In addition to the tasks listed in Table 4, we have completed the implementation and further analysis of several complex applications, including a JPEG encoder and an IEEE 802.11a/g wireless LAN baseband transmitter as shown in Fig. 5—both are fully functional on the fabricated chip. The JPEG encoder principally consists of five sub-tasks: level shift, 8 × 8 discrete cosine transform (DCT), quantization, zig-zag scanning, and Huffman encoding. The fully-compliant 802.11a/g transmitter implementation operates over all 8 data rates, includes additional upsampling and filtering functions not specified by the standard [13], and sustains transmissions at 30% of the full 54 Mb/s rate at a clock speed of 300 MHz on 22 processors [14].

3 Analysis of the Key Features

One of the most important variables in chip multiprocessor architectures is the level of granularity of each processing element. A wide range of granularities are possible as shown in Fig. 6 [6, 10, 15, 16]. The coarse grain two-core Itanium [16] contains large wide-issue processors each close to 300 mm^2 in 90 nm technology, while the fine grain AsAP contains single-issue processors less than 1 mm^2 in 0.18 μm technology. Size differences of factors of tens and hundreds make strong impacts on system behavior.

Most chip multiprocessors target a broad range of applications, and each processor in such systems normally contains powerful computational resources—such as large memories, wide issue processors [16], and powerful inter-processor communication [10]—to support widely varying requirements. Extra computational resources can enable systems to provide high performance to a diverse set of applications, but they reduce energy efficiency for tasks that can not make use of those specialized resources. Most DSP applications AsAP targets are made up of computationally intensive tasks with very small instruction and data kernels, which makes it possible to use extremely simple computational resources—small memory, simple single issue datapath, and nearest neighbor communication—to achieve high energy efficiency while maintaining high performance.

In this section, we analyze these key features of the AsAP processor which justify its fine grain architecture. We also briefly analyze AsAP's GALS clocking style.

3.1 Small Memory

A clear trend among all types of programmable processors is not only an increasing amount of on-chip memory, but also an increasing percentage of die area used for memory. For example, the TI C64x [4] and third generation Itanium processor [5] use approximately 75% and 70% of their area for memory respectively. Since large memories dissipate more energy and require larger delays per transaction, we seek architectures that minimize the need for memory and keep data near or within processing elements.

3.1.1 Inherent Small Memory Requirement for DSP Applications

A notable characteristic of the targeted DSP tasks is that many have very limited memory requirements compared to general-purpose tasks. The level of *required* memory must be differentiated from the amount of memory that can be used or is typically used to calculate these kernels. For example, an N-tap filter may be programmed using a vast amount of memory though the base kernel requires only $2N$ data words. Table 4 lists the actual amounts of instruction and data memory required for 22 common DSP tasks and shows the very small required memory sizes compared to memories commonly available in modern processors.

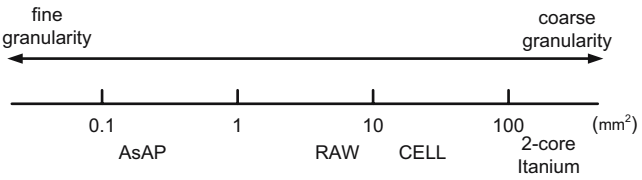


Figure 6 Size of a single processing element in several chip multiprocessor systems. Data are scaled to 0.13 μm CMOS technology.

This analysis assumes a simple single-issue processor like AsAP. Although programs were hand written in assembly code, little effort was placed on optimizing them such as scheduling instructions for the pipeline or using forwarding paths.

3.1.2 Finding the Optimal Memory Size for DSP Applications

Once the amount of required instruction and data memory is known, it is worthwhile to consider what size of memory per processor is optimal in terms of total processing element area. We begin our analysis with several assumptions:

1) The non-memory processor size is 0.55 mm^2 in 0.18 μm CMOS and is not a function of memory size,
2) Memory area scales linearly with capacity and the area is 400 μm^2 for a 16-bit word and 800 μm^2 for a 32-bit word,
3) A fixed partitioning overhead is added each time a task is split onto multiple processors—this overhead is estimated per task and varies from two to eight instructions and from 0–30% of the total space, and
4) Additional processors used only for routing data may be needed for designs using a large number of processors, but are neglected.

Figures 7 and 8 show the total circuit areas for several representative tasks listed in Table 4, while varying the instruction memory and data memory sizes respectively.

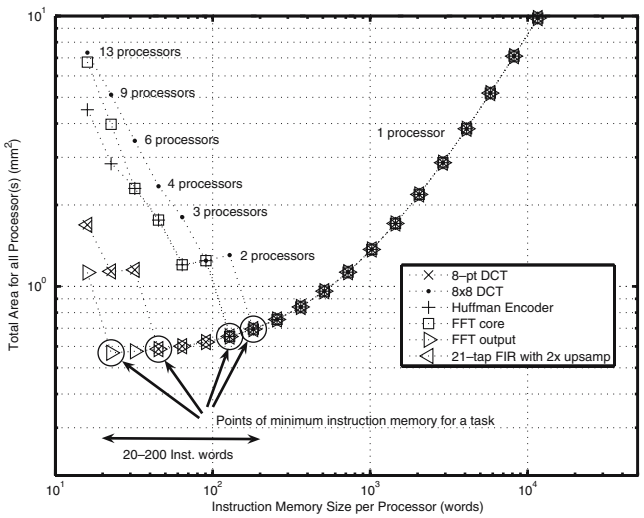


Figure 7 Total area required for representative tasks mapped onto one or multiple 0.18 μm CMOS simple processors, as a function of the size of the instruction memory per processor. Minimum total area is achieved with approximately 20–200 instructions per processor.

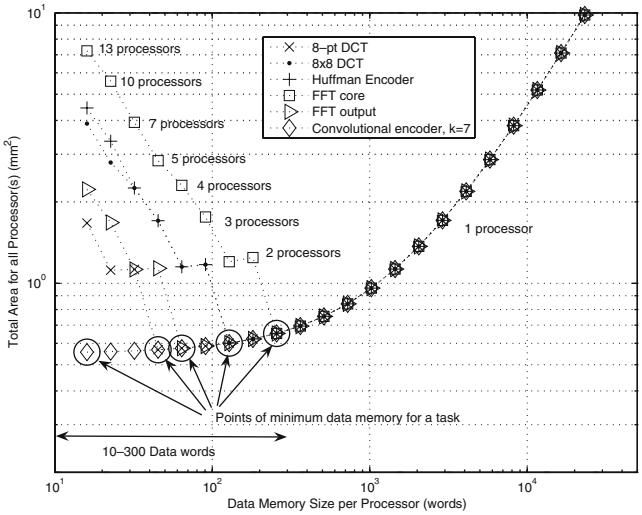


Figure 8 Analysis similar to that shown in Fig. 7 but for data memory. The minimum total area is achieved with approximately 10–300 data words per processor.

These analyses show that processors with memories of a few hundred words will likely produce highly energy efficient systems due to their low overall memory power and their very short intra-processor wires. On the negative side, processors with very small memories that require parallelization of tasks across processors may require greater communication energy and present significant programming challenges.

3.1.3 Several Architectural Features Help Reduce Memory Requirement

In addition to the inherent small instruction memory requirement of DSP applications, address generators help reduce the required instruction memory for applications since they can handle many complex addressing functions without any additional instructions. The upper figure of Fig. 9 shows the estimated relative instruction cost for a system using three addressing modes to fulfill the same functions. Compared to systems primarily using direct memory addressing and address pointers, the system containing address generators reduces the number of required instructions by 60% and 13% respectively. Also, using address generators can increase system performance. As shown in the lower figure of Fig. 9, it comes within 15% of the performance of a system using direct addressing with pre-calculated addresses, and approximately two times higher performance compared to a system using address pointers alone.

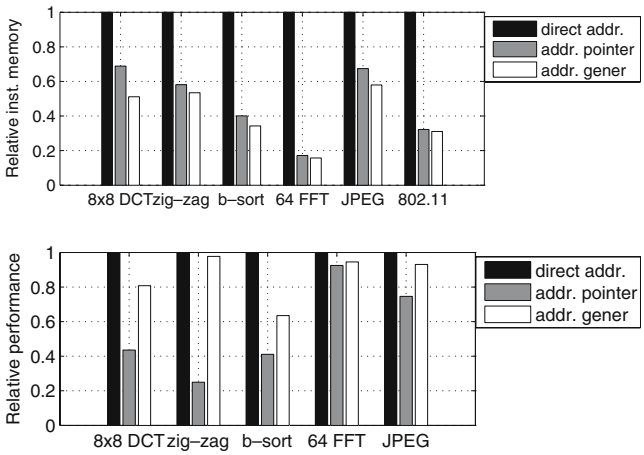


Figure 9 Comparison of relative instruction memory requirements and relative performance for three different addressing modes. Comparisons are made against the *direct address* case which uses straight line code with pre-calculated addresses only.

The embedded NOP instruction field described in Section 2.1.1 also helps reduce instruction memory requirements since it dramatically reduces the number of explicit NOP instructions. Figure 10 shows that instruction memory requirements are reduced by approximately 30% for 9 applications.

In addition to the inherent small data memory requirements of DSP applications, task cascading also helps to reduce the required data memory size. As shown in Fig. 11, a system with many processors can use separate processors to compute individual tasks in an application, and the intermediate data can be streamed between processors instead of buffering them in a large memory.

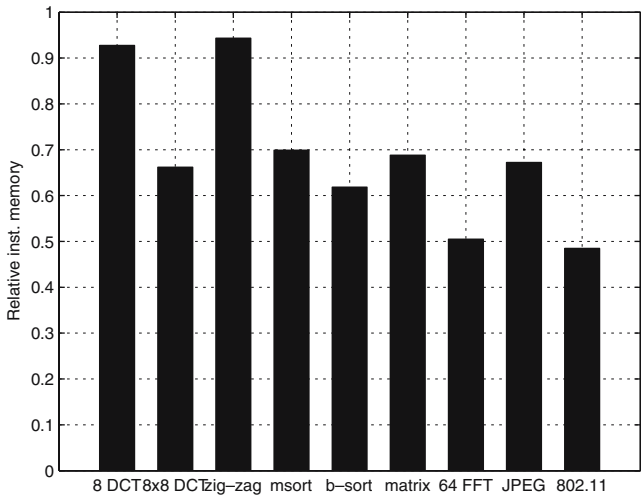


Figure 10 Relative instruction memory reductions by using a 2-bit embedded NOP field in each instruction.

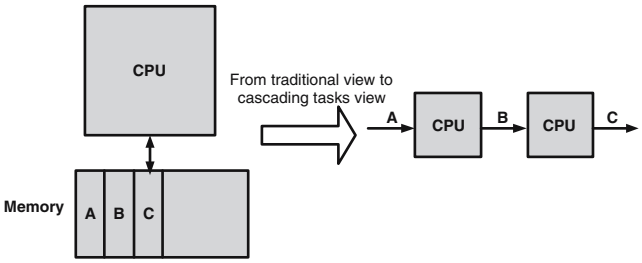


Figure 11 Traditional large memory versus small memory cascading tasks views.

3.2 Datapath—Wide Issue vs. Single Issue

The datapath, or execution unit, plays a key role in processor computation, and also occupies a considerable amount of chip area. Uniprocessor systems are shifting from single issue architectures to wide issue architectures in which multiple execution units are available to enhance system performance. For chip multiprocessor systems, there remains a question about the trade-off between using many small single-issue processors, versus larger but fewer wide-issue processors.

A large wide-issue processor has a centralized controller, contains more complex wiring and control logic, and its area and power consumption increase faster than linearly along with the number of execution units. One model of area and power for processors with different issues derived by J. Oliver et al. [17] shows using wide-issue processors consumes significantly more area and power than using multiple single-issue processors. Their work shows a single 32-issue processor occupies more than two times the area and dissipates approximately three times the power of 32 single-issue processors.

However, chip multiprocessor systems composed of single-issue processors will not always have higher area and energy efficiency—much depends on specific applications. Wide-issue processors work well when instructions fetched during the same cycle are highly independent and can take full advantage of functional unit parallelism, but this is not always the case. Multiple single-issue processors such as AsAP are less efficient if the application is not easy to partition, but it can perform particularly well on many DSP applications since they are often made up of complex components exhibiting task level parallelism so that tasks are easily spread across multiple processors. Large numbers of simple processors also introduce extra inter-processor communication overhead, which we discuss further in Section 3.3.

Figure 12 shows how throughput scales for four single tasks relative to the throughput of a single

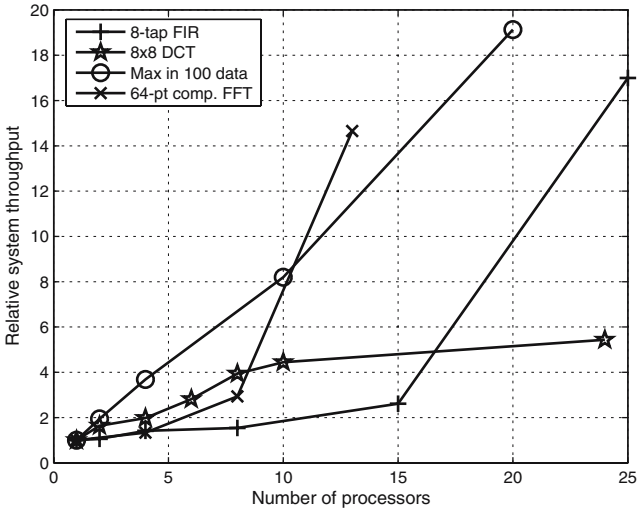


Figure 12 Increase in system throughput with increasing number of processors.

processor. Programs were written in assembly by hand but are lightly optimized and unscheduled. The memory requirement for the 8×8 DCT and 64-pt complex FFT exceeds the available memory of a single AsAP processor, so data points using one processor are estimated assuming one single processor had a large enough memory. An analysis of scaling results of a 16-tap FIR filter implemented in 85 different designs using from 1–52 processors shows a factor of 9 variation in throughput per processor over this space [18].

When all processors have a balanced computational load with little communication overhead, the system throughput increases close to linearly with the number of processors, such as for the task of finding the maximum value of a data set (*Max in 100 data* in Fig. 12).

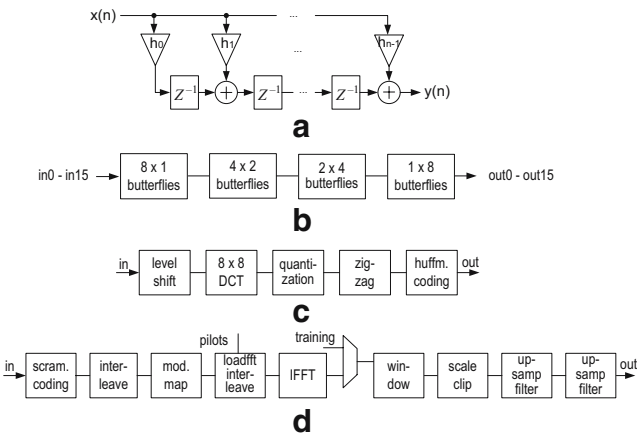


Figure 13 Common DSP tasks and applications exhibiting their abilities to be linearly pipelined: **a** transpose form FIR filter, **b** 16-point radix-2 FFT, **c** JPEG encoder, and **d** IEEE 802.11a/g wireless LAN transmitter.

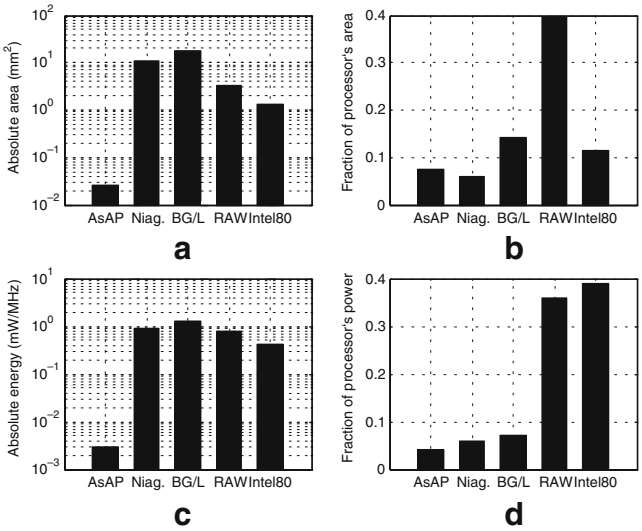


Figure 14 Comparison of communication circuit area and power dissipation for five chip multiprocessors: **a** absolute area, **b** fraction of processor's circuit area used for communication, **c** absolute energy dissipation, and **d** fraction of processor's power dissipation due to communication circuits. Values are scaled to 0.13 μm CMOS technology.

Clearly, applications that are difficult to parallelize show far less scalability at some point. For example, the performance of the 8×8 DCT increases well up to 10 processors where 4.4 times higher performance is achieved, but after that, little improvement is seen and only 5.4 times higher performance is seen using 24 processors. However, there is significant improvement in the FIR filter and FFT after a certain number of processors is reached. The reason for this is because increasing the number of processors in these applications avoids extra computation in some cases. For example, the FFT avoids the calculation of data and coefficient addresses when each processor is dedicated to one stage of the FFT computation. On average, 10 processor and 20 processor systems achieve 5.5 times and 12.3 times higher performance compared to a single processor system, respectively.

Table 5 Estimates for a 13×13 mm AsAP array implemented in various semiconductor technologies.

CMOS tech (nm)	Processor size (mm^2)	Num Procs per Chip	Clock freq (GHz)	Peak system processing (Tera-Op)
180	0.66	256	0.51	0.14
130	0.34	500	0.66	0.33
90	0.16	1,050	1.02	1.07
45	0.04	4,200	2.04	8.57

Table 6 Area, performance and power comparison of various processors for several key DSP kernels and applications; all data are scaled to 0.18 μm technology assuming a $1/s^2$ reduction in area, a factor of s increase in speed, and a $1/s^2$ reduction in power consumption.

Benchmark	Processor	Processor style	Scaled area (mm^2)	Scaled clock freq. (MHz)	Clock cycles	Scaled execution time (ns)	Scaled power (mW)	Scaled energy (nJ)
40-tap FIR filter	AsAP	Array (8 proc.)	5.28	510	10	20	730	15
	MIPS VR5000 [23, 28]	RISC processor.	N/A	347	430	1,239	2,600	3,222
	TI C62x [23, 28]	VLIW DSP	> 100	216	20	92	3,200	296
	PipeRench [29]	Parallel processor	55	120	2.87	24	1,873	45
8x8 DCT	AsAP	Array (8 proc.)	5.28	510	254	498	390	194
	NMIPS [30]	RISC processor	N/A	78	10,772	137,000	177	24,400
	CDCT6 [30]	Enhanced RISC	N/A	178	3,208	18,000	104	1,950
	TI C62x [23, 28]	VLIW DSP	> 100	216	208	963	3,200	3,078
	DCT processor [31]	ASIC	1.72	555	64	115	520	60
Radix-2 complex 64-pt FFT	AsAP	Array (13 proc.)	8.6	510	845	1657	730	1,209
	MIPS VR5000 [28]	RISC proc.	N/A	347	15,480	44,610	2,600	115,988
	TI C62x [23, 28]	VLIW DSP	> 100	216	860	3,981	3,200	12,739
	FFT processor [32]	ASIC	3.5 (core)	27	23	852	43	37
JPEG encoder (8 × 8 block)	AsAP	Array (9 proc.)	5.94	300	1,443	4,810	224	1,077
	ARM [33]	RISC processor	N/A	50	6,372	127,440	N/A	N/A
	TI C62x [28, 33]	VLIW DSP	> 100	216	840	3,900	3,200	12,400
	ARM+ASIC [33]	ASIC + RISC	N/A	50	1,023	20,460	N/A	N/A
802.11a/g transmitter (1 symbol)	AsAP	Array (22 proc.)	14.52	300	4,000	13,200	407	5,372
	TI C62x [28, 34]	VLIW DSP	> 100	216	27,200	126,800	3,200	405,760
	Atheros [35]	ASIC	4.8 (core)	N/A	N/A	4,000	24.2	96.8

The area is the chip core area when available.

3.3 Nearest Neighbor Communication

Currently, most chip multiprocessors target broad general purpose applications and use complex inter-processor communication strategies [10, 19–22]. For example, RAW [10] uses a separate complete processor to provide powerful static routing and dynamic routing functions, BlueGene/L [21] uses a torus network and a collective network to handle inter-processor communication, and Niagara [22] uses a crossbar to connect 8 cores and memories. These methods provide flexible communication abilities, but consume a significant portion of the area and power in communication circuits.

The DSP applications which AsAP targets have specific regular features and make it possible to use a simple nearest neighbor communication scheme to achieve high area and energy efficiency, without a large performance loss. As can be seen from several popular industry-standard DSP benchmarks such as TI [23], BDTI [24], and EMBC [25], the most common tasks include FIR and IIR filtering, vector operations, the Fast Fourier Transform (FFT), and various control and data manipulation functions. These tasks can normally be *linearly pipelined*, as shown in the upper two examples in Fig. 13, and the result from one stage can be sent directly to the next stage without complex global communication. Complete applications containing multiple DSP tasks also have similar features, as two examples shown in Fig. 13c and d for the JPEG encoder and the 802.11a/g baseband transmitter. All these examples can be handled efficiently by nearest neighbor inter-processor communication.

Nearest neighbor communication simplifies the inter-processor circuitry and two dual-clock FIFOs present the major cost in this case, which results in low area and high energy efficiencies. Figure 14 compares AsAP to four other chip multiprocessors (Niagara [22], BlueGene/L [21], RAW [20], and Intel 80-core [26]). The communication circuitry in the AsAP processor occupies less than 0.08 mm^2 in 0.18 μm CMOS, which is approximately 8% of the processor area, and is more than 50 times smaller than the others. Under the worst case conditions when maximizing possible communication, the communication circuity in the AsAP processor consumes around 4 mW at 475 MHz, which is about 4% of the processor power, and the energy efficiency is more than 100 times higher than the others.

3.4 GALS

The GALS clocking style simplifies the clock tree design and provides the opportunity for a joint clock/voltage scaling method to achieve very high energy efficiency. However, at the same time, it introduces an extra performance penalty since it requires extra circuitry to handle asynchronous boundaries which introduce additional latency. It has been shown that the performance penalty from a GALS chip multiprocessor architecture like AsAP can be highly reduced, due to its localized computation and less frequent communication loops. Simulation results show the performance penalty of the AsAP processor is less than 1% compared to the corresponding synchronous system [27].

4 Evaluation of the AsAP Processor

This section provides a detailed evaluation and discussion of the AsAP processor including performance, area, and power consumption.

Each processor occupies 0.66 mm^2 and the 6 × 6 array occupies 32.1 mm^2 including pads. Due to its small memories and simple architecture, each AsAP processor's area is divided as follows: 8% for communication circuitry, 26% for memory circuitry, and a favorable 66% for the remaining core.

Processors operate at 520–540 MHz under typical conditions. The average power consumption for each processor is 35 mW when processors are executing applications such as a JPEG encoder or an 802.11a/g baseband transmitter, and they consume 94 mW when 100% active at 520 MHz. At a supply voltage of 2.0 V, most processors operate at clock frequencies over 600 MHz.

4.1 High Speed, Small Area, and High Peak Performance

Small memories and simple processing elements enable high clock frequencies and high system performance. The AsAP processor operates at frequencies among the highest possible for a digital system designed using a particular design approach and fabrication technology. The clock frequency information listed in Table 6 supports this assertion.

AsAP is also highly area efficient. AsAP has a processing element density about 23–100 times greater than that of other broadly-similar projects [6], and thus each AsAP processor occupies 4% or less area compared to other reported processing elements.

High clock speeds and small area result in a high peak performance density with a fixed chip size. With advancing semiconductor fabrication technologies, the number of processors will increase with the square of the scaling factor and clock rates will increase approximately linearly—resulting in a total peak system

throughput that increases with the *cube* of the technology scaling factor. Table 5 summarizes area and performance estimates for several technologies with the corresponding peak performance. It shows that in 90 nm technology, an AsAP array can achieve 1 Tera-op/s with a 13 × 13 mm chip. Real applications would unlikely be able to sustain this peak rate, but tremendous throughputs are nonetheless expected.

4.2 High Performance and Low Power Consumption for DSP Applications

Table 6 lists area, performance, and power data for a number of general-purpose, programmable DSP, and ASIC processors for which we could obtain data. We choose the TI C62x as the reference programmable DSP processor since it belongs to the TI VLIW C6000 series which is TI's highest performance series. The enhanced TI C64x VLIW DSP processor [4] is also in the C6000 series and has an architecture similar to the C62x, but it contains substantial circuit level optimizations that achieve more than four times higher performance with less than half the power consumption compared to the C62x. We feel the C62x is a fair comparison with the first version AsAP processor and thus a better comparison at the architecture level, without tainting from circuit level optimizations.

In support of our assertion that the AsAP prototype has significant room for improvement, we note that measurements show approximately 2/3 of AsAP's power is dissipated in its clocking system. This is largely due to the fact that we did not implement clock gating in this first prototype. All circuits within each processor are clocked continuously—except during idle periods when the oscillator is halted.

The area used by AsAP, shown in Table 6, is the combined area required for all processors including those used for communication. Data for the FIR, 8 × 8 DCT, and FFT are deduced from measured results of larger applications. We estimated the performance of the JPEG encoder on the TI C62x by using the relative performance of the C62x compared to MIPS processors [28], and a reported similar ARM processor [33].

Figure 15 compares the relative performance and power of an AsAP processor to other processors in Table 6. These comparisons use 8, 8, 13, 9, and 22 AsAP processors—which clearly do not make full use of the chip's 36 processors. Utilizing a larger numbers of processors (through further parallelization) would increase performance further. Nevertheless,

- AsAP achieves 27–275 times higher performance and 96–215 times higher energy efficiency than RISC processors (single issue MIPS and ARM);
- Compared to a high-end programmable DSP (TI C62x), AsAP achieves 0.8–9.6 times higher performance and 10–75 times higher energy efficiency; and
- Compared to ASIC implementations, AsAP achieves performance within a factor of 2–5 and energy efficiency within a factor of 3–50 with an area within a factor of 2.5–3.

Another source of AsAP's high energy efficiency comes from its haltable clock, which is greatly aided by the GALS clocking style. Halting clocks while processors are even momentarily inactive results in power reductions of 53% for the JPEG core and 65% for the 802.11a/g baseband transmitter.

Supply voltage scaling can be used to further improve power savings. Processors dissipate an average of 2.4 mW at a clock rate of 116 MHz using a supply voltage of 0.9 V while executing the described applications.

5 Related Work

There have been many other styles of parallel processors. The key features of the AsAP processor are

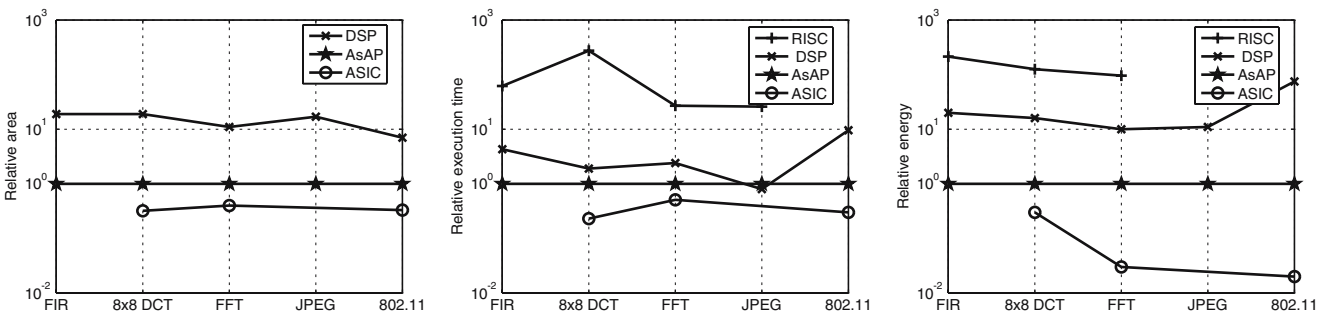


Figure 15 Relative area, execution time and energy for various implementations of several key DSP kernels and applications. Source data are available in Table 6.

Table 7 Comparison of key features of selected parallel processor architectures.

Processor	Single element	Clock style	Inter-proc network	Applications
Pleiades [36]	Heterogeneous (ASIC + proc.)	Handshake GALS	Hierarchical network	Wireless apps
Picochip [37]	Heterogeneous (DSP + coproc.)	Synchronous	Hierarchical network	Wireless apps
Cradle [38]	Heterogeneous (RISC + DSP)	Synchronous	Global bus	Multimedia apps
Imagine [39]	ALU cluster	Synchronous	Hierarchical switch	Stream apps
RaPiD [40]	Multiple execution units	Synchronous	Linear array	DSP apps
PipeRench [29]	Execution unit stripe	Synchronous	Linear array	DSP apps
TRIPS [11]	Wide-issue processor	Synchronous	Two-dimensional mesh; dynamic route	All apps
Intel 80-core [26]	VLIW processor	Mesochronous	Two-dimensional mesh; dynamic route	All apps
Synchroscalar [41]	SIMD processor	Rationally related	Global interconnect	DSP apps
CELL [15]	SIMD processor	Synchronous	High-bandwidth ring bus	Multimedia apps
ClearSpeed [42]	SIMD processor	Synchronous	N/A	High perf. apps
Sandbridge [43]	SIMD processor with cache	Synchronous	N/A	Wireless apps
Smart Memories [19]	Single-issue proc. with 128 KB mem	Synchronous	Packet dynamic route	All apps
RAW [10]	Single-issue proc. with 128 KB mem	Synchronous	Static + dynamic route	All apps
AsAP	Single-issue proc. with 512 B mem	GALS	2D reconfig. mesh	DSP apps

a small memory, a simple processor, GALS clocking style, and reconfigurable nearest-neighbor mesh network. These features distinguish it from other previous and current parallel processors.

The transputer [44] is a popular parallel processor originally developed in the 1980's. It shares the philosophy of using multiple relatively simple processors to achieve high performance. The transputer is designed for a multiple processor board, where each transputer processor is a complete standalone system. It uses a bit serial channel for inter-processor communication which can support communication of different word lengths to save hardware, but with dramatically reduced communication speeds.

Systolic processors and wavefront processors are two more classic parallel architectures. Systolic processors [45] contain synchronously-operating processors which send and receive data in a highly regular manner [46]. Wavefront array processors [47] are similar to systolic processors but rely on dataflow properties for inter-processor data synchronization. Previous designs were optimized for simple and single algorithm workloads such as matrix operations [9] and image processing kernels [48].

More parallel processor projects have appeared recently. Table 7 compares the key features of other projects to AsAP. Most parallel processors can be easily differentiated by their processing element architectures which can be categorized into three broad types—heterogeneous, multiple execution units (often similar to classic SIMD), and multiple processors (MIMD). A heterogeneous style such as the one used by Pleiades [36], Picochip [37] and Cradle [38] makes the system efficient for specific applications, but results in a non-regular layout and difficult scaling. Recent processors such as RAW [10], CELL [15], TRIPS [11], and Synchroscalar [41], also use MIMD architectures, but can be easily distinguished from AsAP by their larger processing element granularity alone. One of the main reasons for their increased processor granularity is because they target a wider range of applications. Most other projects use a fully synchronous clocking style. Pleiades and FAUST [49] use GALS but with handshaking flow control, which is quite different from the source-synchronous interprocessor communication used in AsAP that is able to sustain full-rate communication of one word per cycle at high clock rates. The Intel 80-core chip [26] employs mesochronous clocking where each processor has the same clock frequency while the clock phase is allowed to vary.

6 Conclusion

The AsAP platform is well-suited for the computation of complex DSP workloads comprised of many DSP tasks, as well as single highly-parallel computationally demanding tasks. By its very nature of having independent clock domains, very small processing elements, and short interconnects, it is highly energy-efficient and capable of high throughput.

Measured results show that on average, AsAP can achieve several times higher performance and ten times higher energy efficiency than a high performance DSP processor, while utilizing an area more than ten times smaller.

Areas of interesting future work include: mapping a broader range of applications to AsAP; developing algorithms and hardware for intelligent clock and voltage scaling; automatic software mapping tools to optimize utilization, throughput, and power; C compiler enhancements; connecting large memories when more memory is needed; and automatic fault detection and recovery.

Acknowledgements The authors gratefully acknowledge support from Intel, UC Micro, NSF Grant 0430090 and CAREER Award 0546907, SRC GRC Grant 1598, Intellasys, S Machines, MOSIS, Artisan, and a UCD Faculty Research Grant; and thank D. Truong, M. Singh, R. Krishnamurthy, M. Anders, S. Mathew, S. Muroor, W. Li, and C. Chen.

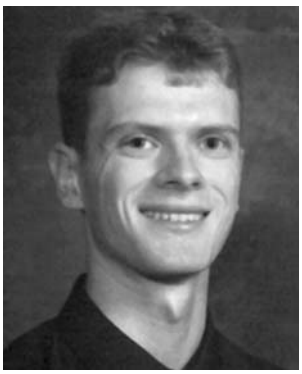
References

1. Horowitz, M., & Dally, W. (2004). How scaling will change processor architecture. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 132–133) (February).
2. Ho, R., Mai, K. W., & Horowitz, M. A. (2001). The future of wires. *Proceedings of the IEEE* (pp. 490–504) (April).
3. Chapiro, D. M. (1984). *Globally-asynchronous locally-synchronous Systems*, PhD thesis. Stanford, CA: Stanford University (October).
4. Agarwala, S., Anderson, T., Hill, A., et al. (2002). A 600-MHz VLIW DSP, *IEEE Journal of Solid-State Circuits (JSSC)* (pp. 1532–1544) (November).
5. Stinson, J., & Rusu, S. (2003). A 1.5 GHz third generation Itanium 2 processor. In *Design Automation Conference (DAC)* (pp. 706–710) (June).
6. Yu, Z., Meeuwsen, M., Apperson, R., et al. (2006). An asynchronous array of simple processors for DSP applications. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 428–429) (February).
7. Bindal, N., et al. (2003). Scalable sub-10ps skew global clock distribution for a 90 nm multi-GHz IA microprocessor. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 346–347) (February).
8. Apperson, R., Yu, Z., Meeuwsen, M., Mohsenin, T., & Baas, B. (2007). A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15*(10), 1125–1134 (October).
9. Kung, S. Y. (1985). VLSI array processors. In *IEEE ASSP Magazine* (pp. 4–22) (July).
10. Taylor, M., et al. (2003). A 16-issue multiple-program-counter microprocessor with point-to-point scalar operand network. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 170–171) (February).
11. Keckler, S., et al. (2003). A wire-delay scalable microprocessor architecture for high performance systems. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 168–169) (February).
12. Glossner, J., Moreno, J., Moudgill, M., et al. (2000). Trends in compilable DSP architecture. In *IEEE Workshop on Signal Processing Systems (SiPS)* (pp. 181–199) (October).
13. IEEE Computer Society (1999). Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: High speed physical layer in the 5 GHz band. In *Standard for Information Technology*. Institute of Electrical and Electronics Engineers.
14. Meeuwsen, M. J., Sattari, O., & Baas, B. M. (2004) A full-rate software implementation of an IEEE 802.11a compliant digital baseband transmitter. In *IEEE Workshop on Signal Processing Systems (SiPS)* (pp. 124–129) (October).
15. Pham, D., et al. (2005) The design and implementation of a first-generation CELL processor. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 184–185) (February).
16. Naffziger, S., et al. (2005). The implementation of a 2-core multi-threaded itanium family processor. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 182–183) (February).
17. Oliver, J., et al. (2006). Tile size selection for low-power tile-based architecture. In *ACM Computing Frontiers* (pp. 83–94) (May).
18. Baas, B., (2003). A parallel programmable energy-efficient architecture for computationally-intensive DSP systems. In *Asilomar Conference on Signals, Systems and Computers* (pp. 2185–2189) (November).
19. Mai, K., et al. (2000). Smart memories: A modular reconfigurable architecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (pp. 161–171) (June).
20. Sungtae, J., et al. (2003). Energy characterization of a tiled architecture processor with on-chip network. In *International Symposium on Low Power Electronics and Design (ISLPED)* (pp. 424–427) (August).
21. Bright, A. A., et al. (2005). Creating the BlueGene/L supercomputer from low-power SOC ASICs. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 188–189) (February).
22. Leon, A. S., et al. (2006). A power-efficienct high-throughput 32-thread SPARC processor. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 98–99) (February).
23. Texas Instruments, DSP platforms benchmarks, Tech. Rep., http://www.ti.com/.
24. Berkeley Design Technology (2000). *Evaluating DSP Processor Performance*. Berkeley, CA, USA.
25. The Embedded Microprocessor Benchmark Consortium (2006). *Data sheets*, www.eembc.org.
26. Vangal, S., Howard, J., Ruhl, G., et al. (2007). An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 98–99) (February).
27. Yu, Z., et al. (2006). Performance and power analysis of globally asynchronous locally synchronous multi-processor

systems. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 378–384) (March).
28. Kozyrakis, C., et al. (2002). Vector vs. superscalar and vliw architectures for embedded multimedia benchmarks. In *Micro* (pp. 283–289) (November).
29. Schmit, H., et al. (2002). PipeRench: A virtualized programmable datapath in 0.18 micron technology. In *IEEE Custom Integrated Circuits Conference (CICC)* (pp. 63–66) (May).
30. Gorjiara, B., et al. (2005). Custom processor design using NISC: A case-study on DCT algorithm. In *ESTIMedia* (pp. 55–60) (September).
31. Matsui, M., et al. (1994). A 200 MHz 13 mm^2 2-D DCT macrocell using sense-amplifying pipeline flip-flop scheme. In *IEEE Journal of Solid-State Circuits (JSSC)* (pp. 1482–1490) (December).
32. Maharatna, K., et al. (2004) A 64-point fourier transform chip for high-speed wireless LAN application using OFDM. In *IEEE Journal of Solid-State Circuits (JSSC)* (pp. 484–493) (March).
33. Lin, T., & Jen, C. (2002) Cascade—Configurable and scalable DSP environment. In *IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 26–29) (May).
34. Tariq, M., et al. (2002). Development of an OFDM based high speed wireless LAN platform using the TI C6x DSP. In *IEEE International Conference on Communications (ICC)* (pp. 522–526) (April).
35. Thomson, J., et al. (2002). An Integrated 802.11a Baseband and MAC Processor. In *IEEE International Solid-State Circuits Conference (ISSCC), 45*, 126–127, 451.
36. Zhang, H., et al. (2000). A 1-V heterogeneous reconfigurable DSP IC for wireless baseband digital signal processing. In *IEEE Journal of Solid-State Circuits (JSSC)* (pp. 1697–1704) (November).
37. Baines, R., et al. (2003). A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers. In *IEEE Communication Magazine* (pp. 105–113) (January).
38. Cradle Technologies, Multiprocessor DSPs: Next stage in the evolution of media processor DSPs, Tech. Rep., http://www.cradle.com/.
39. Khailany, B., et al. (2002). VLSI design and verification of the imagine processor. In *IEEE International Conference on Computer Design (ICCD)* (pp. 289–294) (September).
40. Cronquist, D. C., et al. (1999). Architecture design of reconfigurable pipelined datapaths. In *Advanced research in VLSI (ARVLSI)* (pp. 23–40) (March).
41. Oliver, J., et al. (2004). Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (pp. 150–161) (June).
42. ClearSpeed, CSX600: Advanced product, Tech. Rep., http://www.clearspeed.com/.
43. Sandbridge, The sandbridge sandblaster convegence platform, Tech. Rep., http://www.sandbridgetech.com/.
44. Whitby-Strevens, C. (1990). Transputers-past, present and future. In *IEEE Micro* (pp. 16–19) (December).
45. Kung, H. T. (1982). Why systolic architectures? In *Computer Magazine* (pp. 37–46) (January).
46. Kung, H. T. (1988). Systolic communication. In *International Conference on Systolic Arrays* (pp. 695–703) (May).
47. Kung, S., et al. (1982). Wavefront array processor: Language, architecture, and applications. *IEEE Transactions on Computers, C-31*(11), 1054–1066 (November).
48. Schmidt, U., & Mehrgardt, S. (1990). Wavefront array processor for video applications. In *IEEE International Conference on Computer Design (ICCD)* (pp. 307–310) (September).
49. Lattard, D., Beigne, E., Bernard, C., et al. (2007). A telecom baseband circuit based on an asynchronous network-on-chip. In *IEEE International Solid-State Circuits Conference (ISSCC)* (pp. 258–259) (February).

Zhiyi Yu received the B.S. and M.S. degrees in Electrical Engineering from Fudan University, Shanghai, China, in 2000 and 2003, respectively, and the Ph.D. degree in Electrical and Computer Engineering from University of California, Davis, in 2007.

Dr. Yu is currently a Hardware Engineer with IntellaSys Corporation, headquartered in Cupertino, CA. His research interests include high-performance and energy-efficient digital VLSI design, architectures, and processor interconnects, with an emphasis on many-core processors. He was a key designer of the 36-core asynchronous array of simple processors (AsAP) chip, and one of the designers of the 150+ core second generation computational array chip.



Michael J. Meeuwsen received the B.S. degrees with honors in Electrical Engineering and Computer Engineering (both summa cum laude) from Oregon State University, Corvallis, and the M.S. in Electrical and Computer Engineering from the University of California, Davis.

He is currently a Hardware Engineer with Intel Digital Enterprise Group, Hillsboro, OR, where he works on CPU hardware design. His research interests include digital circuit design and IEEE 802.11a/g algorithm mapping.

Ryan W. Apperson received the B.S. in Electrical Engineering (magna cum laude) from the University of Washington, Seattle, and the M.S. degree in Electrical and Computer Engineering from the University of California, Davis.

He is currently an IC Design Engineer with Boston Scientific CRM Division, Redmond, WA. His research interests include multiclock domain systems and SRAM design.

Michael A. Lai received the B.S. and M.S. degrees in Electrical and Computer Engineering from the University of California, Davis.

He is currently a Design Engineer at Altera Corporation working on next generation transceiver products. His research interests include the design of high-speed arithmetic units and control.

Omar Sattari received the B.S. and M.S. degrees in Electrical and Computer Engineering from the University of California, Davis.

He is currently a Software Engineer at CornerTurn. His research interests include FFT and DSP algorithms and digital hardware design.

Jeremy W. Webb received the B.S. degree in Electrical and Computer Engineering from the University of California, Davis.

He is currently a M.S. student in Electrical and Computer Engineering at the University of California, Davis, and a hardware engineer at Centellax. His research interests include high-speed board design and system interfacing.

Eric W. Work received the B.S. degree from the University of Washington, and the M.S. degree in Electrical and Computer Engineering from the University of California, Davis.

He is currently a Software Engineer at S Machine Corporation. His research interests include the mapping of arbitrary task graphs to processor networks and software tool flow.

Tinoosh Mohsenin received the B.S. degree in Electrical Engineering from Sharif University, Tehran, Iran, and the M.S. degree in Electrical and Computer Engineering from Rice University, Houston, TX. She is currently pursuing the Ph.D. degree in Electrical and Computer Engineering from the University of California, Davis.

She is the designer of the split-row and multi-split-row low density parity check (LDPC) decoding algorithms. Her research interests include energy efficient and high performance signal processing and error correction architectures including multi-gigabit full-parallel LDPC decoders and many-core processor architecture design.

Bevan M. Baas received the B.S. degree in Electronic Engineering from California Polytechnic State University, San Luis Obispo, in 1987, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 1990 and 1999, respectively.

In 2003 he became an Assistant Professor with the Department of Electrical and Computer Engineering at the University of California, Davis. He leads projects in architecture, hardware, software tools, and applications for VLSI computation with an emphasis on DSP workloads. Recent projects include the asynchronous array of simple processors (AsAP) chip, applications, and tools; low density parity check (LDPC) decoders; FFT processors; viterbi decoders; and H.264 video codecs.

From 1987 to 1989, he was with Hewlett-Packard, Cupertino, CA, where he participated in the development of the processor for a high-end minicomputer. In 1999, he joined Atheros Communications, Santa Clara, CA, as an early employee and served as a core member of the team which developed the first IEEE 802.11a (54 Mbps, 5 GHz) Wi-Fi wireless LAN solution. During the summer of 2006 he was a Visiting Professor in Intel's Circuit Research Lab.

Dr. Baas was a National Science Foundation Fellow from 1990 to 1993 and a NASA Graduate Student Researcher Fellow from 1993 to 1996. He was a recipient of the National Science Foundation CAREER Award in 2006 and the Most Promising Engineer/Scientist Award by AISES in 2006. He is an Associate Editor for the *IEEE Journal of Solid-State Circuits* and has served as a member of the Technical Program Committee of the IEEE International Conference on Computer Design (ICCD) in 2004, 2005, and 2007. He also serves as a member of the Technical Advisory Board of an early stage technology company.