An Overview of the ZSPä Architecture

For more information contact:
Karen Suty
LSI Logic Corporation
Public Relations
Tel: (408)433-6855
Fax: (408)433-8989

Email: suty@lsil.com September 27, 1999

The ZSP Architecture

The ZSP Architecture offers the optimum solution for many next-generation DSP applications. The Architecture leverages design techniques from advanced microprocessors and is optimized for digital communications and wireless network applications.

Signal Processing Performance

- Superscalar DSP architecture
- Dispatches up to four instructions per cycle
- Dual Multiply Accumulate (MAC) and dual Arithmetic Logic Unit (ALU)
- Short five-stage pipeline minimizes penalties of branch misprediction
- RISC-based instruction set architecture: ALL instructions execute in a single cycle
- Fixed-length instructions allow efficient pre-fetch and decoding to increase throughput
- Load/Store architecture: De-couples load and store operations from instructions

Ease of Use

- Register-based orthogonal instruction set
- Highly programmable: compiler AND programmer friendly
- Familiar programming paradigm minimizes software risk
- Hardware scheduling: eliminates conflicts and ensures deterministic behavior
- Compiler produces efficient DSP assembly from high-level languages
- Hardware caches and pre-fetch logic minimize programmer exposure to memory access bottlenecks

Cost-effective

- High code efficiency: High performance without unrolling inner loops
- Microprocessor-based programming paradigm for control applications
- Efficient context save and a multi-level interrupt structure for multitasking

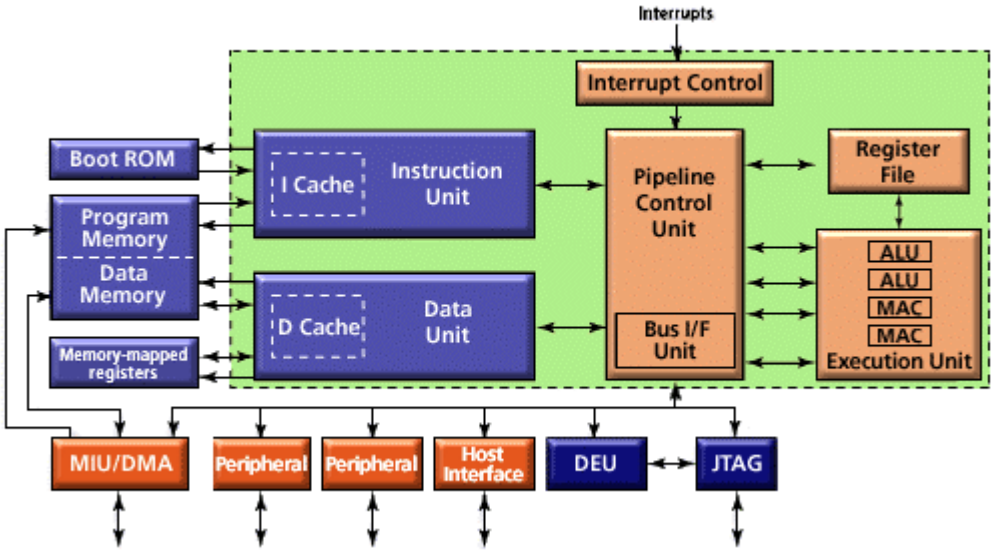
Flexibility

- High performance enables programmable solutions for new applications
- Hidden pipeline enables software compatibility across ZSP Processor families
- Scalable: Architecture may be reconfigured for varying processing needs without affecting the programming model

Functional Units

The ZSP Architecture comprises the Instruction Unit, Data Unit, Pipeline Control Unit, two MAC units, two ALUs, and a register file. Figure 1 shows an example of a complete system with the core of the ZSP Architecture indicated by a dotted line. The complete system includes boot ROM, dual access internal RAM, memory interface unit (MIU) with DMA controller, serial port, host processor interface(HPI), debugger unit (DEU), and JTAG interface.

Figure 1. ZSP Processor Architecture



The Instruction Unit (IU)

The Instruction Unit comprises a direct-mapped instruction cache, pre-fetch unit, and the instruction dispatch unit. The tasks performed by the IU every cycle include the pre-fetch of four instructions, instruction decode and dispatch, and the placing of instructions into cache. The pre-fetch unit utilizes static branch prediction and pre-fetch techniques to minimize cache miss penalties and reduce pipeline stalls. Typically, DSP programs execute in tight loops and the instruction cache reduces memory accesses, thereby reducing power consumption. The ZSP Architecture places no restriction on the size of cacheable loops that are nestable and interruptible. The Architecture also provides hardware-supported zero overhead loops.

The Data Unit (DU)

The DU comprises the direct mapped data cache, data pre-fetch unit, circular buffer unit, and load/store arbiter. It is the task of the DU to pre-fetch and buffer four data words per cycle. This unit also has the ability to write two data words per cycle, if required. Like the instruction cache the data cache reduces power dissipation by minimizing memory accesses. The Data Unit provides hardware for the implementation of two circular buffers and for the sustained data throughput required by DSP applications.

The Pipeline Control Unit (PCU)

The PCU's role is to group instructions for parallel execution. In this task, the PCU resolves data and resource dependencies in the program sequence. Stated another way, this hardware schedules instructions for execution by the four functional units (two MACs and two ALUs), simplifying the tasks of the programmer or the compiler. The PCU also synchronizes the entire operation of the pipeline, arranges operand bypass, and processes interrupt requests.

The ZSP Processor is four-way scalar and employs a five-stage pipeline. At any time, there may be a maximum of twenty instructions in various stages of execution in the pipeline. The five pipeline stages of this machine are Fetch/Decode (F/D), Group (G), Read (R), Execute (E), and Write (W).

The Fetch/Decode stage is where instructions are fetched, decoded, and issued. The Group stage is where instructions are grouped for parallel execution after thorough checking of dependencies. The operand register file and the data cache are read, and functional unit bypassing is performed during the Read stage. Bypassing allows a functional unit to access the result of the previous instruction without waiting for the result to be written to the operand register file. The Execution stage is where ALU, MAC, and any internal memory access operations are completed. Finally, the operand register file is written during the Write stage, and store data is written to memory.

As previously stated, the PCU processes interrupt requests. The interrupt structure provides four user-defined priority levels. When an interrupt occurs and is enabled (unmasked), its priority is compared to the current execution priority level of the machine. If the new interrupt is of equal or higher priority, its level is saved as the current execution priority level and the interrupt is taken. If the new interrupt is of lower priority than the execution priority level of the machine, the new interrupt will remain pending until the execution priority level of the machine drops.

The user can change the interrupt priority of an interrupt source "on-the-fly" by explicitly writing a new priority level to the appropriate field of the Interrupt Priority registers. This method can be used to raise the priority level of another interrupt source while executing a lower priority interrupt routine. The user can also change the execution priority level of the machine "on-the-fly" by writing to the Interrupt Priority register, thereby enabling interrupts of lower priority to be taken without modifying any of the assigned interrupt priorities. This flexible and programmable interrupt mechanism enables quick response to real-time tasks and without any risk of task starvation.

It takes five cycles for the processor to respond to an interrupt. The double word load and store instructions enable fast context saving during interrupt.

The Multiply Accumulate Unit (MAC)

The ZSP Architecture has two MAC units that can work together or independently. Independently, they can each perform a 16-bit by 16-bit multiply with a single 40-bit accumulation in a single cycle.

Together, they work to perform a single 32-bit by 32-bit multiply with 40-bit accumulation in a single cycle. The MACs also contain hardware support for complex multiplies and the functionality to perform a single-cycle add-compare-select for Viterbi decoding. The MACs also support parallel add and parallel subtraction.

MAC operations affect the **v** (32-bit overflow), **gv** (40-bit overflow), **c** (carry) and **ge** (greater than or equal to zero) flags, and results are immediately available in the next cycle for any other functional unit while being written to the register file. There are "sticky" overflow flags for 32- and 40-bit overflow conditions. The sticky flags can only be cleared by software. The Functional Mode register controls most MAC operations in terms of rounding, saturation on overflow, and fractional number support.

The Arithmetic Logic Unit (ALU)

The ZSP Architecture provides two identical 16-bit ALUs that can work independently in parallel or together to form a single 32-bit ALU. In addition to traditional ALU functionality, these ALUs also provide bit manipulation and normalization capability.

All ALU operations are single-cycle, and affect the **gt** (greater than zero), **ge** (greater than or equal to zero), **z** (zero), **v** (overflow) and **c** (carry) flags. The result of any ALU operation is available for any functional unit on the next cycle while being written to the register file. The Functional Mode register controls rounding, saturation, and fractional number support of most ALU operations.

The Register Files

There are two types of register files: the operand register file (ORF) and the control register file (CRF). All registers are 16-bit, memory-mapped, and can be read or written by the user.

The operand register file (ORF) has a total of sixteen 16-bit registers. Any of these registers may be used as the input or output of any ALU operation, or as a pointer to memory for register indirect addressing. For MAC and MUL instructions, all the registers can be used as input but only a subset of these can be destination registers. In addition, any operand register may be used as a stack pointer. The registers are denoted by **rX**, where $0 \leq X \leq 15$. The ORF is accessed via load/store instructions.

The ZSP Architecture also provides support for extended precision (32-bit) operations. In these operations (typically denoted in the instruction set with a "**.e**" extension), registers are used in pairs and are referenced by the lower, even numbered register. For example, the instruction

add.e r2, r4

describes the addition of the 32-bit operands contained in register pairs {r3 r2} and {r5 r4}. The result of this operation is stored in {r3 r2}.

The control register file (CRF) provides mode control as well as status and flag information. The CRF contains thirty-two 16-bit registers.

System Memory and the Memory Interface Unit (MIU)

The system memory is logically segmented into internal program, external program, internal data, and external data spaces. Each segment is 64Kwords in size (16-bit address). The total program and data external address space is extended from 16-bit to 20-bit via the use of a page register, giving a total of 1Mword of address space.

The example from Figure 1 demonstrates an internal memory of 2Kwords boot ROM and 62 Kwords of dual access RAM for instruction and data. This dual access RAM is connected to both the IU and DU by separate 64-bit data buses and can be simultaneously accessed from both the instruction and data memory spaces with no speed penalty. These buses provide the bandwidth to fetch four instructions and four 16-bit data words per cycle.

The Memory Interface Unit (MIU) is a pin-multiplexed 20-bit address, 32-bit data external memory interface for glueless external memory or peripheral expansion. The MIU provides bus arbitration logic for easy interface to external shared memory. The pin multiplexing allows the sharing of instruction and data address lines to reduce the pin count. The 4-bit page register resides in MIU to enable the memory address space extension to 20-bits.

The ZSP Architecture can also support Harvard (separate program and data) and low-power, high-density memory systems based upon single-port memories.

Programming Example

The ZSP Instruction Set is based on a load/store RISC paradigm. All instructions can use any of the general-purpose registers as the source and destination. The entire instruction set provides the following features:

- Single word (16-bit) instruction
- Single cycle execution of all instructions
- Any two 16-bit ALU operations
- Any 32-bit ALU operations
- Two 16-bit X 16-bit MUL with a single 40-bit accumulation
- One 32-bit X 32-bit MUL with 40-bit accumulation
- Exponent detection for 16/32 bit variable
- Square of 16/32 bit variable
- 16/3- bit min and max instructions
- Majority of basic functions defined by ETSI for speech coding applications
- Perform two additions or subtractions in MAC units to support ALU intensive code
- Single-cycle compare-select instruction that supports two cycle Viterbi butterfly operation
- 16-bit complex multiplication or multiply-accumulate in two cycles
- Extensive bit manipulation instructions, bit reversal, bit shift, bit set, bit clear, and bit invert
- Excellent conditional branch support
- Zero overhead looping
- Circular buffer support

The following assembly code implements the vector add, Z = X + Y, where X, Y, Z are 32-bit vectors.

```
.segment "text"
.global  main
```

```
main:           lda r14, ARRAY_X        /* r14 is a pointer to input data X */
                lda r15, ARRAY_Y        /* r15 is a pointer to input data Y */
                lda r13, RESULT/* r13 points to result array Z */
                mov %loop0, 7           /* execute this loop 8 times */
vector_add_32:
                lddu r4, r14, 2         /* r4=M(r14), r5=M(r14+1), r14=r14+2 */
                lddu r8, r15, 2         /* r8=M(r15), r9=M(r15+1), r15=r15+2 */
                add.e r4, r8            /* (r5 r4) = (r5 r4) + (r9 r8) */
                stdu r4, r13, 2         /* M(r13)=r4, M(r13+1)=r5, r13=r13+2 */
                agn0 vector_add_32      /* test, decrement counter, then branch */
end_vector_add:
                halt
                .segment "data"
RESULT:         .wspace 16              /* data space for result array Z */
```

Development Tools

The ZSP Processor family is fully supported by a GNU-based compiler, linker and assembler, available for Windows 95, Windows NT, and Solaris 2 platforms. The ZSP Architecture enables the C compiler to produce code unrivaled in code density and execution speed by any DSP in its class, offering fast to time to market with minimal compromise on performance and cost.

On-chip debug is facilitated by the JTAG port of ZSP devices, interfaced to host PC platforms via a convenient PCMCIA interface.

Development platforms are available, offering the following features:

- Integrated Debug Environment
- FLASH EPROM
- RS232C and JTAG-based host communication and code download
- Expansion to 64K words data and program memory
- Two voice-band codecs
- Analog I/O interfaces