FX8010 - A DSP Chip Architecture for Audio Effects

Steve Hoge

Joint E-mu/Creative Technology Center

steveh@emu.com, http://www.emu.com, http://www.creaf.com

Abstract

FX8010 is a DSP chip architecture specifically designed for time-domain 3D audio and effects processing. It is a 32-channel, 32-bit integer design that can deliver 100MIPS at a 50KHZ audio sample rate. It features powerful delay memory and I/O engines that execute in parallel with and are decoupled from microprogram execution. Its highly regular architecture supports the simultaneous execution of large numbers of separately compiled and downloaded programs with zero-overhead signal patching. A compiler for FX8010 programs generates code from C-style expressions and control-flow constructs. FX8010 has been implemented in two different ASICs for PC multimedia and professional audio applications.

1. Introduction

FX8010 is a real-time digital signal processing architecture specifically designed to implement time-domain digital audio effects and multichannel mixing. By coupling a highly regular four-operand, 32-bit integer architecture with independent delay memory and I/O engines, FX8010 delivers 100MIPS at 50KHZ sample rate and is capable of simultaneously executing up to eight high-quality reverberators or dozens of simpler algorithms.

The FX8010 architecture has already been implemented in two different ASICs. The *EMU10K1*, a PCI-based wavetable synthesizer, DirectSound accelerator and audio interface chip, uses the FX8010 as part of its 3D and environmental audio effects engine and is a major element of the Creative Technology SBLive! and E-mu Audio Production Studio products. The *RChip* is a dedicated effects processor for embedded musical instrument and professional audio applications, and is one of several custom DSPs used in E-mu's Mantis digital audio mixing system. Numerous patents are pending on unique aspects of the FX8010 architecture.

2. Basic Architecture

The FX8010 design comprises a 32/67 bit execution unit, a 1K memory array of 32-bit General Purpose Registers (GPRs) for signals, coefficients, and addresses, and 32 channels of 32-bit signal I/O via double-buffered I/O GPRs. Microprogram storage is an array of 1K instruction words, each of which specifies one opcode and four GPR operand addresses.

The processor's opcodes include fractional and integer multiply/accumulate instructions, linear interpolation, bit-wise logical operations, conditional instruction execution and data movement, and single-cycle logarithmic and exponential conversion. The multiply/accumulate unit uses a 67-bit accumulator including 4 guard bits and can accommodate double-precision operations.

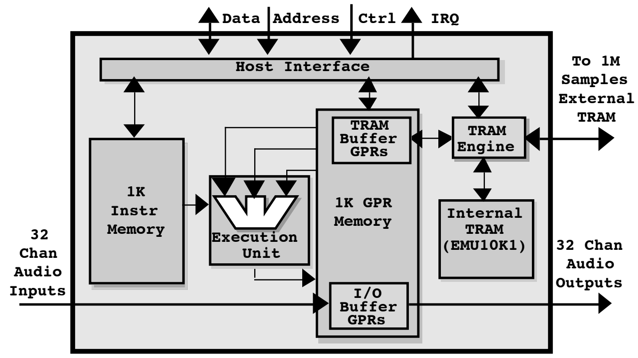

Figure 1. Basic FX8010 Architecture

Internal and external "tank" memory (TRAM) for audio delay lines and table look-up is managed by a TRAM engine that operates independently of and in parallel with microprogram execution. This engine transfers audio samples between external memory in a 1M word off-chip address space and internal dual-ported data buffer GPRs shared between the TRAM engine and the FX8010 execution unit. Up to 256 delay line or table accesses each sample period are implemented by this addressing and data move engine.

In contrast to most commercial DSP chips, the FX8010 is sample-locked and runs without jumps or branches, though conditional data movement is available and block-oriented control flow constructs can be implemented using conditional instruction execution. The architecture is specifically designed to support the execution of multiple simultaneous but independently compiled and loaded effects programs, a capability that is facilitated by the conditional execution mechanism, a lack of exposed pipelining, and the direct addressing of GPR operands.

Since FX8010 cannot operate alone but is designed to be controlled by a conventional microprocessor, a high-bandwidth host interface provides mapping of the FX8010's internal GPR and microinstruction memory directly into the host's address space. Interrupts from the FX8010 to the host can be generated under control of DSP programs and when signal saturation (clipping) occurs. A debug facility allows the processor to be run in single-step mode.

3. Execution Unit

3.1 Arithmetic

The FX8010's 32-bit integer arithmetic exceeds the accuracy of single-precision floating point, and has sufficient dynamic range and precision for almost any audio processing or filtering operation. Total dynamic range is over 192dB, and the center frequency resolution of a biquad filter using 32-bit coefficients is less than 1HZ across the entire audio range. While all fractional coefficientsmust lie in the range of [-1.0..1.0], sufficient footroom exists to normalize most filter topologies so that their coefficients fall within this range. Limit cycles that can arise in recursive filters from asymmetric truncation of results towards -∞ are not typically an issue in FX8010 due to the small magnitude of the truncation error.

The accessibility of both the MS and LS halves of the FX8010 67-bit accumulator allow the multiply/accumulator unit to perform either fractional or integer arithmetic, depending on which half is retrieved as the result operand and how it is saturated. This accessibility also makes possible double-precision operations, if necessary. With both integer and fractional multiplication available, the FX8010 compiler can generate coefficients that implement conventional left and right shift operators (<< and >>).

3.2 GPR Operand Architecture

The FX8010 execution unit is connected directly to a 1K GPR address space for operand storage. Each FX8010 instruction includes an opcode and four independent GPR addresses that define the instruction's three input operands A, X and Y and the result operand R. For multiply/accumulate instructions, A is the accumulator while X and Y are symmetrical multiplier inputs. There are no visible hazards in the FX8010 operand pipeline, so the result operand of one instruction can become any of the input operands on the next instruction cycle.

Many DSP architectures place input registers ahead of their execution units that must be kept filled with operands by the programmer (often with parallel move operations) in order to extract maximum compute bandwidth from the processor. Keeping these registers full at all times is a challenging problem of operand sequencing and sometimes even memory layout, and becomes one of the arcane skills of the DSP programmer. By contrast, the FX8010 math unit has no such input registers; instead, all instructions fetch their operands by directly addressing GPR memory. In this respect the FX8010 architecture is very programmer-friendly.

Similarly, most DSPs also have volatile accumulators or output registers that must be saved with data move instructions or recycled as "special" input operands. While the FX8010 also has a volatile accumulator output register, FX8010 result operands are typically ordinary GPRs whose data movement is implicit in the result operand address. This accumulator *can* be reused by explicitly specifying its GPR-mapped address as the A operand, but is only necessary when extra headroom (the accumulator guard bits) or precision (the LS 32 of the accumulator's 67 bits) need to be retained through the next multiply/accumulate instruction.

The accumulator is one of several special registers that are GPR-mapped, including the condition code register (CCR), interrupt register, read-only delay line and table base address registers, and noise (dither) sources. In addition, FX8010 maps a collection of useful ROM constants into GPR space that are used as implicit operands in many instructions.

3.3 Instruction Set

Opcode	Operation
MAC	Fractional multiply/add/subtract with optional saturation/word wrap
MACINT	Integer multiply/add/subtract with optional saturation/word wrap
ACC3	Accumulate 3 inputs with saturation
MACMV	Multiply/accumulate with additional data move
SKIP	Conditionally skip over instructions
ANDXOR	Multi-purpose bitwise logical instruction
TSTNEG	Test and conditionally negate the result
LIMIT	Test and conditionally output a higher/lower threshold
LOG	Convert linear to logarithmic representation
EXP	Convert logarithmic to linear representation
INTERP	Linear interpolate between two values

Table 1: Typical FX8010 opcodes

As seen in the table of opcodes, the FX8010's instructions implement traditional DSP arithmetic as well as some more unusual operations:

MACMV performs multiply/accumulation on X, Y and the accumulator, while in parallel moving the A operand to R. This simultaneously accomplishes the MAC and data shift required for FIR filtering.

ANDXOR (R = A & X ^ Y) allows the FX8010 compiler to take advantage of the 4-operand architecture and built-in ROM constant GPRs to synthesize bitwise AND, XOR, NOR, NAND, NOT and OR operations from a single opcode.

LOG and *EXP* perform transformations to and from a sign|exponent|mantissa representation with a programmable maximum exponent. Applications are data compression, dB conversion, waveshaping and log domain arithmetic approximating division and roots. Interesting distortion effects are also possible, especially by modulating the exponent size.

INTERP, which performs the linear interpolation r = a*x + y*(1-x) allows single-instruction lowpass filters, parameter smoothing, and inversely proportional signal mixing (e.g., wet/dry or pan control.)

LIMIT and *TSTNEG* are both forms of conditional move instructions, useful for threshold detection and control signal generation. The compiler synthesizes ABS() and SIGN() from TSTNEG by the right choice of GPR operands and ROM constants.

4. I/O Engine

32-channel signal input and output is accomplished in FX8010 through buffers which are mapped into GPR space. Since I/O is fully double-buffered, new input signals appear synchronously at the beginning of each sample period in the 32 input GPRs, and the contents of the 32 output GPRs disappear off -chip.

In EMU10K1, physical input signals originate in the wavetable synthesizer and various AC97, I^2S and S/PDIF codecs, and are output through codecs or back across PCI to the host. The RChip also supports I^2S and S/PDIF, but mainly uses *EMU32*, a serial 32-bit, 32-channel interface, for I/O connections with other DSPs.

5. TRAM Engine

5.1 Circular Delay Addressing

The TRAM engine transfers samples between GPR-mapped buffers in the FX8010 and TRAM memory in a 1M off-chip address space. TRAM is used for delay lines as well as indexed table look-up. For delay lines, the TRAM engine uses a circular addressing mechanism, computing the absolute address of each TRAM access by adding a relative delay offset to a global base address counter modulo the entire delay address space, and decrementing the counter once per sample period. Since all delay lines recirculate within the same physical memory, modulo-addressing of individual delay lines is not required.

5.2 Decoupled TRAM Execution

Each TRAM access is implemented using a pair of buffer registers mapped into GPR memory and a third register that contains flag bits that control the type of TRAM access. One buffer GPR stores the TRAM address offset and the other stores an incoming or outgoing sample word (for reads or writes, respectively.) By GPR-mapping the buffers, FX8010 programs can operate on TRAM data like any other GPR operand and can compute new TRAM offsets for modulated delay effects or table-lookups. Triples composed of these data, offset and flag registers are organized in an array of contiguous memory locations where they are operated on sequentially by the TRAM engine every sample period. If the flag register is viewed as an opcode and the data and address buffers as operands, then the TRAM engine can be seen as an independent execution unit that iterates its own simple microprogram once each sample period.

A dual-port memory architecture ensures that accesses by the execution unit and TRAM engine do not collide. Since operation of the TRAM engine is decoupled from program execution, 100% TRAM bandwidth utilization is guaranteed by design without stalling the execution unit or requiring the FX8010 programmer to manage memory transactions.

A hardware mechanism that returns zeros from each delay line until it contains valid data obviates the need for the time-consuming TRAM zeroing that would otherwise be required at program load time.

5.3 Physical TRAM Implementation

Variations in TRAM architecture constitute the major differences between the RChip and the EMU10K1 implementations of FX8010. TRAM on the RChip is implemented exclusively with external SRAMs, but in the EMU10K1 this off-chip memory can include system DRAM accessed across the PCI bus in the host processor's address space. To preserve PCI bandwidth, it has an additional block of TRAM located in a separate address space on-chip.

All TRAM in the EMU10K1 is 16-bits wide, but the RChip can be programmed to accommodate 16, 24 or 32-bit wide memory. TRAM less than 32-bits wide can be accessed using a hardware-based encoding scheme that is transparent to the DSP programmer and extends the TRAM's effective dynamic range. This encoding contributes greatly to a low noise floor in recursive algorithms like reverberators, which are often plagued with noticeable truncation distortion and noise-like limit cycles, especially as feedback coefficients are increased.

While all GPRs are 32-bits wide, address offsets occupy only the top 21-bits of the TRAM buffer GPR. These MS 21 bits and the remaining LS 11 bits can be thought of as the integer and fractional part of the address, respectively. In a single MACINT instruction, the LS bits can be masked and left-shifted to become the coefficient to an INTERP instruction in order to implement a linear-interpolated delay line.

6. Microsequencer

6.1 Microprogram Control Flow

FX8010 microprograms are stored on-chip in an array of wide microinstruction memory which cannot be written to by the execution unit. The FX8010 executes in sample-locked fashion, so that the instruction rate is a fixed multiple of the sample rate. The FX8010 runs straight through its entire microinstruction array each sample period without jumps, branches, or subroutine calls, so that it is impossible to fall out of real-time operation.

While straight-line execution is an appropriate model for implementing linear time-invariant filters, many common audio effects require event processing at a regular sub-audio control rate or even asynchronously. FX8010 accommodates this by providing conditional move operations and *conditional execution* using the SKIP instruction. With this technique, the FX8010 does not actually skip forward over sequences of instructions but, based on tests of its Condition Code Register (CCR), converts these sequences into NOPs. This conditional mechanism also supports multiprogramming by skipping over areas where new programs are being loaded without disturbing other executing programs.

6.2 Condition Code Register

The FX8010 CCR holds 5 bits, some or all of which are updated after each instruction cycle:

- Z - set if the result is zero
- M - set if the result is negative (minus)
- N - set if a normalized result (MSB=next MSB)
- S - set if the result saturated or wrapped
- B - set if a borrow occurred in a subtraction

Typical SKIP conditions such as M + Z (less than or equal to zero) or more exotic ones such as M + (S • ~M) (negative or saturated positive) can be specified by CCR test masks. By computing CCR masks and their inverses along with the proper skip counts, the compiler is able to generate if/then/else constructs nested to any depth from the familiar C-language source code syntax. While the CCR mask and skip count are typically compiler-generated constants, since they are stored in ordinary GPRs they can also be computed by the microprogram.

7. Effects Development

7.1 FX8010 Compiler

fxasm, the FX8010 program compiler, accepts source code files in a C-like expression syntax and generates a portable object code format. Currently, fxasm is integrated into an effects development system using Microsoft Developer Studio.

FX8010 programmers combine executable statements with declarations of input and output ports, GPRs, mix registers, constants, tables, and delay lines. These objects are linked symbolically to the high-level parameter control code running on the host processor through #include symbol files. All GPR, microcode and TRAM addresses emitted from the compiler are virtual; virtual-to-physical translation happens both at program load time and at run-time, when real-time parameter updates and queries are relocated on-the-fly.

7.2 Drivers

A driver stack written in C manages multi-program resource allocation, loading, patching and parameter control in real-time across multiple FX8010s. Pools of temporary GPRs and other resources are maintained for shared use by all loaded programs. An abstraction layer allows the same effectsmanagement code to run efficiently on top of both of the current FX8010 hardware implementations. For example, on a P166 running Win95, relocating and loading a typical reverberator requires approximately 2ms.

Above the driver stack are self-contained effects software plug-ins which encapsulate the FX8010 program and the host code necessary for parameter control. In Win95 these plug-ins take the form of independent registered COM objects which can be further encapsulated in ActiveX wrappers for use by DirectShow-aware applications.

7.3 Benchmarks

The efficiency of the FX8010 architecture allows instruction counts for most algorithms to be estimated simply by the number of multiplies required. Thus a reverberator allpass filter takes two instructions and a direct form biquad requires five, as shown in the sample listings of FX8010 source code.

The implementation of FX8010 in the EMU10K1 is sufficiently powerful that the E-mu Audio Production Studio product is able to simultaneously run high-quality reverb and chorus algorithms as well as a flanger, echo, "auto-wah" envelope filter, distortion, compressor/limiter, pitch-shifter, 4 parametric EQs, 4 shelving EQs, and a large mixing matrix, all with smoothed parameter updates and ditherable outputs.

```
// Allpass[ ] is automatically
// allocated in TRAM
DELAY allpass[ 50msec ] ;
GPR in, out ;

out       = in * .6 + Allpass[ 40msec ] ;
Allpass[ ] = in - out * .6 ;
```

Listing 1: FX8010 source code for Reverb allpass

```
// Output is in state[2]
GPR state[4], in ;
GPR a[3], b[2] ; // A/B Coefficients

                ACC =   input  *a[0];
state[1]=state[2],ACC+=state[1]*a[2];
state[0]=input,  ACC +=state[0] *a[1];
state[3]=state[2],ACC+=state[3]*b[1];
state[2]=        ACC + state[2] *b[0];
```

Listing 2: FX8010 source code for Direct Form 1 Biquad

```
//dB peak meter in 2 instr. (LOG+LIMIT)
GPR in, peak ;
TEMP GPR tmp ;
tmp  = ABS( LOG( in )) ;
peak = tmp > peak ? tmp : peak ;
```

Listing 3: FX8010 Source code for peak VU meter