Goals and tradeoffs in the design of the MC68881 floating point coprocessor

by JOEL BONEY
Motorola Inc.
Austin, Texas

ABSTRACT

This paper describes the goals and tradeoffs in the design of the MC68881 Floating Point Coprocessor. The Motorola MC68881 is a complete implementation of the proposed IEEE floating point standard on a single VLSI chip. It is a coprocessor for the MC68020 microprocessor and is a peripheral processor for other M68000 family processors.

The design of the MC68881 was guided by a set of goals. This paper discusses the major goals of the MC68881 project and their impact on the design. During the definition of the architecture of the MC68881 many engineering tradeoffs were made by the design team. This paper also documents how some of these tradeoffs affected our decisions. Lastly, the paper gives enough of an overview of the MC68881 to make the discussions of the goals and tradeoffs meaningful.

INTRODUCTION

No design project should be undertaken without a good set of clear goals that are the guiding information allowing the designers to make the necessary tradeoffs during the design process. This paper documents the design goals and some of the architectural tradeoffs of the MC68881 design project. This VLSI design project will take about 4 years from the first preliminary specification to first silicon (which is expected about the time this paper is published).

The Motorola MC68881 is a complete implementation of the proposed IEEE floating point standard on a single VLSI chip.[1] It is a coprocessor for the MC68020 microprocessor and is a peripheral processor for other M68000 family processors. Since it will be necessary to have some knowledge of the MC68881 in order to understand the goals and tradeoffs, this paper also includes an overview of the MC68881. More specific detail about the MC68881 can be obtained from other papers and articles published by the design team.[2,3,4]

AN OVERVIEW OF THE MC68881

The MC68881 is a high performance floating point unit designed to interface with the 32-bit MC68020 as a coprocessor. It can also be used as a peripheral processor with some performance degradation, in systems where the MC68020 is not the main processor (e.g. MC68000, MC68008, MC68010). The configuration of the MC68881 as a coprocessor or a peripheral processor can be completely transparent to user software.

The MC68881 utilizes the general purpose M68000 family coprocessor interface to provide a logical extension of the CPU's instruction set and register set such that it is transparent to the programmer. The programmer is *never* aware that the coprocessor and main processor are implemented on two separate chips.

Internally the MC68881 is divided into two processing elements, the Bus Interface Processor (BIP) which handles the coprocessor interface and the Arithmetic Processor (AP). All interaction with the main processor is handled by the BIP while the AP executes all MC68881 instructions.[4]

Bus Interface Processor

All interprocessor transfers are initiated by the MC68020. During the processing of an MC68881 instruction, the MC68020 transfers instruction information and data to the coprocessor via standard M68000 write bus cycles using a unique CPU function code and receives data, requests for service, and status information from the coprocessor via standard M68000 read bus cycles.

The MC68881 contains a number of coprocessor interface registers which are addressed like memory by the MC68020's micro-machine. These registers are not part of the programmer visible register set.

Reserved opcodes in the M68000 instruction map that formerly trapped out to an exception routine (Line 1111 Emulator Trap) are now defined as coprocessor instructions. Only the MC68020 tracks the instruction stream. When it detects a coprocessor instruction, it writes the next word in the instruction stream to the coprocessor and reads the coprocessor's response. The BIP encodes in the response any additional action required of the main processor on behalf of the coprocessor. A typical request for service is "evaluate the effective address and transfer N bytes of data to the coprocessor interface operand register."

The coprocessor interface permits the MC68881 to execute most floating point instructions concurrent with the MC68020's execution of non-floating point instructions.

The MC68881 is designed to operate over 8-, 16-, or 32-bit data buses. The part is packaged in a 64-pin DIP or 68-pin Pin-Grid-Array package.

The coprocessor interface is fully compatible with the MC68020's on-chip instruction cache and virtual memory architecture. The interface insures that *all* coprocessor execution time exceptions, including instruction single-step, are handled identically to main processor execution time exceptions. Both the MC68020 and the MC68881 are designed for 16.67-Mhz operation. Since the interface is based solely on standard M68000 *asynchronous* bus cycles, the MC68881 need not run at the same clock speed as the main processor.

Arithmetic Processor

Once the BIP has decoded an instruction and requested any operands it needs, the microcode in the Arithmetic Processor is started to acquire the operands and to perform the requested operation. The AP is implemented as a pseudo two-level micro-machine much like the MC68000.[7]

Architecture Overview

The architecture of the MC68881 appears to the user as a logical extension of the M68000 family architecture. It is a register oriented, one-and-a-half-address processor similar to the MC68000 and its derivatives.[6]

Programmer's model

The MC68881 adds the following registers to the programmer's model of the M68000 family:

1. Eight 80-bit floating point data registers analogous to the M68000 integer data registers.
2. A 32-bit control register contains enable bits for each class of exception trap, and mode bits to select rounding mode and rounding precision.
3. A 32-bit status register contains the floating point condition codes, quotient bits set by remainder and modulo, and exception status information.
4. A 32-bit instruction address register contains the address in memory of the last floating point instruction. This address is used in exception trap handling.

Data formats

The MC68881 supports the following data formats:

1. Byte, word, and long word integers,
2. Single, double, and extended precision binary real,
3. Decimal real string (packed BCD).

The three integer data formats are identical to those supported by M68000 family processors. The floating point data formats, single precision (32-bits), and double precision (64-bits) are as defined by the IEEE standard.[2]

The extended precision data format is also in conformance with the IEEE standard, but the standard does not specify this format to the bit level as it does for single and double. The format on the MC68881 consists of 96 bits, 3 long words, with an explicit most significant mantissa bit. Only 80 bits are actually used, the other 16 bits are left for future expandability.

The decimal real string format consists of a signed 3-digit base 10 exponent and a signed 17-digit base 10 mantissa. All digits are packed BCD so that a whole string fits in 96 bits.

Integer, single precision, double precision, and decimal real string format operands are always converted to an extended precision floating point number prior to participating in an MC68881 operation. The floating point data registers always contain extended precision values, and all internal computations are performed to extended precision.

Instruction set

The instruction set of the MC68881 can be subdivided as follows:

1. Moves; register to register, external operand to register, and register to external operand forms are provided. The external operand may be any of the 7 data formats supported, and may be specified by any MC68020 addressing mode.
2. Arithmetic and Transcendental Operations; register to register and external operand to register forms are provided. The external operand may be any of the 7 data formats supported, and may be specified by any MC68020 addressing mode. The result is always placed in the specified floating point data register.
3. Miscellaneous; move multiples (in and out) branches, set on condition, trap on condition, save context, restore context, etc.

The arithmetic and transcendental operations are listed in Figure 1. Dyadic operations (those requiring two operands) are listed first followed by the monadic operations.

Add	IEEE Remainder
Compare	Scale Exponent
Divide	Single Precision Divide
Modulo	Single Precision Multiply
Multiply	Subtract
Absolute Value	Log Base 2
Arc Cosine	Log Base e
Arc Sine	Log Base e of x + 1
Arc Tangent	Negate
Hyperbolic Arc Tangent	Sine
Cosine	Sine and Cosine
Hyperbolic Cosine	Hyperbolic Sine
e to the x Power	Square Root
e to the x Power − 1	Tangent
Get Exponent	Hyperbolic Tangent
Get Mantissa	10 to the x Power
Integer Part	Test
Log Base 10	2 to the x Power

Figure 1—Supported operations

All operations required by the IEEE standard are provided on the MC68881 plus many more. All instructions support all IEEE defined special values (normalized, zeroes, infinities, denormalized numbers, and 'not-a-numbers'), and return the IEEE specified results with accuracy as specified in the standard.

Following the precedent set by the orthogonal instruction set in the M68000 family of processors, MC68881 instructions are provided for move, arithmetic, and transcendental operations using any data format and any addressing mode. The domain of an operand in a given data format is unrestricted for all operations. *No* operations require software envelopes to conform to the standard. Similarly, for the transcendentals, all argument reduction is performed *on-chip*.

The MC68881's conditional instructions utilize 32 floating point conditional predicates encoded in five bits. The four possible relations between two floating point numbers, greater than, equal, less than, or unordered, are encoded into four bits. The fifth bit, as required by the proposed standard, indicates whether an exception should be raised if the predicate evaluation yields an unordered relationship.

GOALS AND TRADEOFFS

Goals

There were five major goals for the MC68881 project given in the following priority:

1. The MC68881 should have the same style of architecture as the other members of the M68000 family
2. Performance
3. Functionality and user friendliness
4. Reduce design time and long term design costs
5. Producibility

M68000 Family Style of Architecture

Since we felt that the functionality of the MC68881 would eventually be moved onto the same die as the main CPU, an important goal was to insure that the architecture of the MC68881 fit in with the rest of the family. The MC68881 should expand the instruction set of the main CPU in an orthogonal manner that was transparent to the programmer (i.e., the user should not be aware that the MC68020/MC68881 consisted of two devices).

The coprocessor interface scheme is crucial to achieving this goal. The philosophy was to split the work done by the coprocessor interface between the main CPU and the coprocessor such that each element does what it can do best. For example, the MC68020 decodes the original instruction and determines that it is a coprocessor instruction. It then informs the coprocessor by writing a coprocessor defined operation word to the coprocessor. The coprocessor decodes this word and requests that the main CPU do the effective address calculation and transfer operands of 'n' bytes to the coprocessor. Or if a floating point exception occurred, the coprocessor might ask the main CPU to commence exception processing. Thus it can be seen that the MC68020 does what it already knows how to do: decide basic instructions, calculate effective addresses, and take exceptions. The coprocessor knows about its defined operation and knows what kind and size of data it wants from the main CPU or if an exception occurred.

A tradeoff was made in the coprocessor interface scheme to use standard asynchronous M68000 bus cycles for communication between the main CPU and the coprocessor. There was a minor speed penalty for this method when the MC68881 was used as a coprocessor for the MC68020, but it allowed the MC68881 to be used by all other M68000 family members as a peripheral.

This decision, along with the decision to *not* make the MC68881 a bus master (i.e., the MC68881 does not fetch its own operands; they are fetched by the main CPU and passed to the MC68881) greatly simplifies the system hardware interface to the MC68881 and allows flexibility.

Another tradeoff/decision made by the MC68881 design team was the selection of a register based one-and-a-half address architecture. In this type of architecture one of the operands typically comes from memory while the other operand comes from a register with the result going to the register or memory. This architecture is consistent with the architecture of the other M68000 family members. Further, since the M68000 processors have 8 integer data registers, the decision was made to have 8 additional floating point data registers. Studies have indicated that 8 registers are optimal for expression evaluation, etc.; and by having the same number of integer and floating point data registers compiler writers should be able to use the same register allocation algorithms for integers and floating point.

Orthogonality across the instruction set and addressing modes is a feature of the M68000 family that was preserved by the MC68881. All the addressing modes of the MC68020 are available for accessing floating point operands. Further, the safety features supported by the M68000 processor such as illegal instruction and illegal addressing mode traps are also supported by the MC68020/MC68881 pair.

Performance

Within the constraints of M68000 family architectural consistency, performance was the next most important design goal for the MC68881. Both the MC68020 and the MC68881 were designed for a clock speed of 16.67 Mhz. Even though the HCMOS process results in a slightly larger die, it was selected for both projects because of speed and low power consumption.

Performance of the basic functions, add, subtract, multiply, and divide, was emphasized. Special hardware was added to the execution unit to speed up these basic operations. Table I gives the execution times for the register to register forms of these operations on a MC68020/MC68881 pair. These times do not reflect the potential throughput increase from concurrency.

The single multiply and single divide operations assume that their operands are single precision, and produce a single precision result (while maintaining the range of extended). These operations are provided for special applications where multiply and divide performance is more important than loss of significance.

Even though we wanted the operations to be very fast on the average, one tradeoff we made was to insure that the worst case execution times would not be significantly different from the best case times. In some applications the only important item would be the average execution time, but in real-time applications the whole system usually has to be designed using the worst case time. Floating point units that depend on slow *software envelopes* to handle special cases will be very hard to use in real-time applications.

All calculations in the MC68881 are done internally to full IEEE extended precision. Even though we might have achieved marginally faster single and/or double precision times by including special hardware for single and double precision, we decided to concentrate our efforts in making extended precision very fast. This gives us very competitive times for all operand size not just single or double.

The last major performance-related tradeoff was the deci-

TABLE I—Execution times

Operation (reg-reg)	Clock Cycles	Time (μsec) @ 16.67 Mhz
Add	40	2.4
Subtract	40	2.4
Multiply	60	3.6
Divide	92	5.5
Single Mul	46	2.8
Single Div	58	3.5

sion to support concurrent operation. Concurrency means that once an instruction is started in the MC68881 the MC68020 is free to continue executing other non-MC68881 instructions. Thus the two processors overlap their execution, which increases the overall throughput of the pair. The support of concurrency did cost some silicon area and added some complexity, but we felt that the potential benefits outweighed the silicon costs.

Functionality and User Friendliness

Probably the biggest tradeoff we made toward functionality and user friendliness was the decision to support the proposed IEEE standard in its entirety in silicon.[1] As participants in the standardization process we felt the accuracy and safety provided by the standard greatly outweighed the minor impact it had on die size and hence, cost. Many people seem needlessly frightened by the complexity of the standard. If all the defaults of the standard are selected, the user is hardly aware of it except that he gets better results and has fewer problems with his algorithms blowing up than with conventional floating point implementations.[5] Most of the special modes are included for the expert numerical analysts and can be ignored by the average user.

Conformance to the standard involves much more than just conformance to the specified data formats. The standard specifies what operations must be supplied in a conforming implementation, and what accuracy is required for the operations. Further, it defines exceptions, specifies their detection, and specifies the results of exceptional operations in both trapping and non-trapping environments. The standard specifies *special* data types within each format (signed zeroes and infinities, not-a-numbers, denormalized numbers) and specifies the results of operations involving these special data types. It also specifies user selectable modes for rounding mode and precision. Any floating point hardware element that does not support all these requirements does *not* conform to the IEEE standard.

In addition to the functions required by the standard we decided to support many additional functions including a complete set of transcendental functions. As with the IEEE required functions, no software envelope is required to make the functions work correctly. The transcendentals even do the argument reduction on chip.

A slightly more efficient use of silicon would have been made if we had just implemented a set of primitive transcendentals on the chip. All the functions we support can be derived from a subset of primitives. There are perhaps a few hundred people in the world who know how to derive these correctly. It took us several years to figure it out. We didn't want our customers to have to go through what we did to become numerical experts in order to use our part, nor did we want to ship a large, slow software envelope with every part. The silicon impact was minimal, so we just put everything on the chip.

Another major tradeoff we made was whether to support all of the data types supported by the M68000 family in addition to the floating point data types and conversions required by the standard. We decided to support all data types including a decimal real string type. This feature along with the fact that all internal operations are done to full extended precision makes the MC68881 very easy to use and very accurate. The old FORTRAN problem of mixed modes goes away when an MC68881 is used since all sizes and types of data can take part in a floating point calculation with maximum accuracy.

As mentioned previously, we decided to support concurrency for performance reasons; however, we made a lot of minor design tradeoffs to insure that the concurrency is completely transparent to the programmer.

Reduce Design Time and Long Term Design Cost

As VLSI chips have gotten bigger, the time it takes to do the architectural design, the circuit design, and the layout has increased dramatically. We therefore made many tradeoffs in the design to reduce the design complexity. The MC68881 is implemented as a pseudo two-level microcode machine. It has a very wide control word with very little residual control.[7] Several PLAs are used for microcode address generation and for the coprocessor responses.[4]

Nearly all the cost of implementing the IEEE standard is contained in several PLAs and a small amount of microcode. There is almost no random logic used to implement the IEEE standard or for that matter any of the other functionality improvements of the MC68881. The only time we used random logic was in the performance paths in the execution unit for the basic four functions and in parts of the BIP. The MC68881 is the most regular non-memory VLSI microprocessor device we have ever produced.

As for long term design cost, we felt that no manufacturer could afford to make a whole family of floating point coprocessors—the market just isn't big enough to justify the cost. Because we felt this way, we were more likely to include extra functionality on the MC68881 so that we don't have to do an enhanced version later. Further, the general purpose coprocessor interface insures us that we won't have to do a new version of the MC68881 for each existing M68000 family member nor will we have to do a new version for any new family members. Therefore, we may have put more design effort and cost into the original MC68881 design, but we feel we greatly reduced the long term design cost to Motorola.

Producibility

The best paper design in the world is useless unless it can be produced cheaply in volume. Although at times we did tradeoff die size for regularity and functionality, the final die size is producible in the HCMOS process. And if processing improvements continue at the pace they have in the past, in a few years the MC68881 will seem like a tiny die.

In fact, testing and package costs will dominate the device cost over time. To this end we will package the MC68881 in a 64-pin DIP or 68-pin Pin-Grid-Array package. Both of these packages will be high volume packages. For testing, the MC68881 has extensive on-chip test logic to reduce test costs that I am not free to discuss in this paper.

SUMMARY

This paper has attempted to provide a glimpse into the thought processes of the designers of the MC68881. The project had more goals than the 5 mentioned and there were an endless number of tradeoffs made daily with only the major ones mentioned here. Of course, dozens of people participate in the design of any VLSI device from the initial marketers who gave us customer input to the final layout draftsmen who put it on silicon. Rarely were any of the decisions mentioned in this paper made by one or two people, but rather by groups.

REFERENCES

1. IEEE Computer Society Microprocessor Standards Committee Task P754. "A Proposed Standard for Binary, Floating Point Arithmetic, Draft 10.0." January 1983. A copy may be obtained now from Richard Karpinski, UCSF U-76, San Francisco, Calif. 94143, and ultimately from IEEE, 345 East 47th St., New York, NY. Draft 10.0 is a substantial revision of Draft 8.0 published in *Computer,* March, 1981.
2. Boney, J., P. Harvey, and V. Shahan. "Floating Point Power for the M68000 Family." *Proceedings of 1983 Mini/Micro West,* November 1983, Session 16, paper #5.
3. Cawthron, D. and C. Huntsman. "The MC68881: Motorola's Floating-Point Solution." *IEEE Micro,* December 1983.
4. Shahan, V. "The MC68881: The IEEE Floating Point Standard Reduced to One VLSI Chip." *Proceedings of COMPCON, Spring 84.*
5. Kahan, W. "The Proposed IEEE Standard p754 for Floating Point Arithmetic: What Good Is It?" *Proceedings of 1983 Mini/Micro West,* November 1983, Session 16, paper #1.
6. Zolnowsky, J. and N. Tredennick. "Design and Implementation of System Features for the MC68000." *Proceedings of COMPCON, Fall 79,* September 1979,
pp. 2–9.
7. Stritter, E. and N. Tredennick. "Microprogrammed Implementation of a Single-Chip Microprocessor." *Proceedings of the 11th Annual Workshop on Microprogramming* (Micro-11), November 1978, pp. 8–16.