The R65F11 and F68K Single-Chip Forth Computers

Randy M. Dumse

New Micros, Inc.
808 W. Dalworth
Grand Prairie, TX 75050

Abstract

The R65F11, introduced mid year 1983, was the first commercially available single-chip Forth "engine." It has remained relatively unchallenged in its market position for over one year. Other single-chip systems are under development. In all probability, near the end of 1984, the F68K will become the second single-chip engine for the Forth language, with an anticipated 20 times execution speed improvement in performance over its predecessor. This paper focuses on the design methodologies used in the development of the R65F11 and the new F68K.

Introduction

A processor that has a native machine code language that is optimized for a particular language is commonly called a language "engine." Conventional high level language engines get their title by having a micro code that is optimized for interpreting a particular high level language's pseudo code. There is only one level of interpretation involved. The conventional language engine must implement the pseudo code of the high level language as its own machine code if there is to be any advantage in an "engine" over a standard instruction set processor.

The advantages a conventional high level language engine can have over a processor with an external pseudo code interpreter in ROM are speed of execution and reduced program memory usage.

The most notable entry in the field of conventional high level language engines came from Western Digital in the form of their Pascal Engine. The device was a processor (CPU only), rather than a complete computer on a chip. (The actual CPU was made up of five chips in total.) It directly executed the P code from the Pascal compilers. The part did not receive wide acceptance due to technical problems, questionable performance benefits, and an unresponsive marketplace. [1]

Current generation high level languages have a closed set of command structures that are patterned after very limited subsets of English. They allow the programmer to enter something that looks like English. The language compiler or interpreter converts the input into computer executable machine code.

If Forth were a conventional language it would have to be said that there are no single-chip Forth "engines" in existence at the time of this writing. In the case of Forth engines, however, the variation in the concept is significant. A discussion of the comparison of Forth to conventional languages will have to be made to understand the significance of this distinction. [2] [3]

Although the language starts with a limited subset of the English language, the programmer can add both functions and interpreters of functions to the language. Programming in the language occurs on several levels. The software is multi-dimensional. There is no limit to the dimensions of interpretation possible. No matter how completely the original language may be accommodated in an engine implementation, the user will create new interpreters and problem interpretations on top of the original language. Until self modifying hardware can keep up with a dynamically changing language set, there can be no perfect Forth engine.

The most important aspect of creating a Forth engine with current technology is to create the virtual Forth machine in hardware. Only the software can be extensible. Therefore, the essence of the Forth engine exists whenever the virtual Forth processor is implemented, at any level. Although the speed of the interpretation of the Forth language depends on the level to which the language is reduced to micro code, any Forth engine will offer the following advantages over conventional processors:

1. Reduced program memory usage.
2. Extensibility.

These features, then, can be used to measure whether a single-chip computer is a Forth engine or not.

Implementation Approach

The methods by which the Forth engine can be implemented (as per the above definition) can vary greatly. The processor itself must contain sufficient features to support at least the common inner interpreters of Forth to be a virtual Forth machine. This can be accomplished by:

1. implementing the inner interpreters and some primitives in internal ROM,
2. micro coding some of the lower functions of the inner interpreters as well (DOCOL, NEXT and SEMIS, for example),
3. or ideally, by micro coding all of the primitives of the language to a unified machine code instruction set.

The method used on the R65F11 was the first. The inner interpreters (DOCOL, DOUSER, DOVAR and DODOES), along with 133 primitive and higher level words, were placed in internal ROM. Since there were no modifications to the instruction set of the host processor, the R65F11 has no speed advantage over a conventional processor running Forth from an external ROM. No external run time ROM is needed, however, with the R65F11. From an etic point of view, the R65F11 has reduced memory requirements as compared to a conventional processor running machine code, and has all the advantages of the Forth Operating System with extensibility strongly supported. These are exactly the characteristics required of a Forth engine. [4]

The F68K follows the same philosophy. All the inner interpreters of the original language are included in the internal ROM of the part. As in the R65F11 the run time kernel words are also included in ROM. Due to the higher efficiency of the 16-bit architecture of the F68K and the expanded 4K bytes of ROM memory, the F68K has many additional words included in the internal set. The micro monitor concept first implemented in the R65F11 is expanded upon in the F68K to the point that it has become a respectable outer interpreter. The Operating System functions are thereby greatly enhanced. Of course both qualifying marks of a Forth engine, extensibility and reduced memory requirements, are fully realized in the F68K.

Both the R65F11 and F68K use external development ROMs to complete the normal set of Forth words found in the standard implementations of the languages. The R65F11 is based on fig-Forth, while the F68K is expected to have two separate Development ROMs, one for the 79 and one for the 83 Standard.

Case History: The R65F11

The R65F11 was the first single-chip computer Forth "engine." The extensibility of Forth makes it possible to categorize the R65F11 as a high level language "engine," and the effective nature of its Forth Operating System gives the R65F11 "usability" that exceeds the current interpretive Tiny BASIC chips.

The R65F11 was intended to be used as a dedicated controller which could be rapidly programmed in a high level language. It was to be a versatile, complete system needing only the addition of a user's program in EEPROM and a small amount of "glue" to make a useful target computer system. It was also intended that the combination of language, Operating System and the chip itself would allow development of programs and systems without the need for expensive development systems.

BANKEXECUTE	BANKEEC!	BANKC@	BANKC!
EEC!	?	.	.R
D.	D.R	#S	#
SIGN	#>	<#	SPACES
SEEK	INIT	DWRITE	DREAD
SELECT	DISK	M/MOD	*/
*/MOD	MOD	/	/MOD
*	M/	M*	MAX
MIN	DABS	ABS	D+-
+-	S->D	COLD	(NUMBER)
HOLD	BLANKS	ERASE	FILL
QUERY	EXPECT	(.")	-TRAILING
TYPE	COUNT	DECIMAL	HEX
-DUP	SPACE	PICK	ROT
>	<	U<	=
-	2-	1-	2+
1+	PAD	C/L	HLD
DPL	IN	CLD/WRM	BASE
UR/W	UPAD	UC/L	R0
S0	TIB	BL	4
3	2	1	0
C!	!	C@	@
TOGGLE	+!	BOUNDS	2DUP
DUP	SWAP	2DROP	DROP
OVER	DNEGATE	NEGATE	D+
+	0<	NOT	0=
R	R>	>R	LEAVE
;S	RP@	RP!	SP!
SP@	XOR	OR	AND
U/	U*	CMOVE	CR
?TERMINAL	KEY	EMIT	ENCLOSE
(FIND)	DIGIT	I	(DO)
(+LOOP)	(LOOP)	0BRANCH	BRANCH
EXECUTE	CLIT	LIT	

Listing 1. Included words of the R65F11 kernel [5]

Language in ROM

In order to install the conventional Forth outer interpreter in the R65F11, the dictionary would have to be inside the chip. Also, the dictionary control and searching words would have to be included. A study of the problem showed that the outer interpreter called about 80% of Forth when run. Putting the conventional outer interpreter inside with the kernel was clearly out of the question. Only the kernel of Forth could be fit into its 3Kbytes of internal ROM. The kernel was a selected list of words that included all the run time functions and language primitives as shown in Listing 1. The words that could not be included inside were put in an external development ROM.

The words included in the kernel are all the run time functions of the language. Putting the compile time functions in the external Development ROM is a very workable solution. The combination of 3K-bytes of internal ROM and 8K-bytes of external Development ROM made for a very complete Forth implementation, with a full assembler and many other extensions especially created to make programming dedicated applications easier.

The method of separating the dictionary heads from the code bodies of the internal kernel words also gave rise to a feature unique to the Forth implementation on the R65F11. It is possible to target compile (i.e., generate headerless code) a program and test the code with the separated heads located in an entirely separate section of memory. Usually testing target compiled code interactively is an unknown luxury. On the R65F11 it is as easy as typing HHHH H/C, where HHHH is the hex address where the heads are to be placed, and H/C is the word that initiates target compilation. [6]

The NEXT function in the R65F11 is basically the same as the one provided in the fig-Forth Model as shown here in Listing 2. High level interrupts are provided for. This slows the overall language down, but was considered a very desirable feature in a high level language environment (i.e., if the user was so good at machine code why would he use a high level language?).

```
NEXT   BIT  INTFLG
       BVS  INTRTN
NEXT1  LDY  # 1
       LDA  (IP),Y
       STA  W+1
       DEY
       LDA  (IP),Y
       STA  W
       CLC
       LDA  IP
       ADC  # 2
       STA  IP
       BNE  NEXT3
       INC  IP+1
NEXT3  JMP  (W−1)
```

Listing 2.

The above NEXT takes 31 bytes and nominally 41 microseconds to execute.

The "Micro Monitor"

The desire to have some sort of interactive monitor inside the kernel still remained. Without a command interpreter of some kind the chip could hardly be considered a stand-alone system. The ability to autostart a user's program in ROM certainly gave the flavor of an operating system. Further, the ability to boot programs from disk if no such autostart ROMs were found in the memory map, exceeded the capabilities of all other "systems on a chip." Yet, without the interpretive interaction of the outer interpreter the system just did not "feel" like a Forth system.

There was very little room left for such an addition. Most interactive monitors require at least 2 Kbytes of program to provide minimal features. All of the desired functions of a useful monitor were available in the kernel. If only there were a way to access them interactively the problem would be solved. This train of thought led to the inclusion of the "micro monitor", a small program that performs well as a small outer interpreter. (The original concept for a stand-alone, minimal interpreter germinated during a talk given by Owen Thomas. [7])

The way of handling the entry of words to be interpreted by the monitor without using the dictionary is the key to the monitor. Instead of using the ASCII names of the word to be executed, words are identified by their Code Field Addresses.

In order to distinguish between what is a word and what is a number, the "micro monitor" requires that hexadecimal entries be preceded by either an "N", meaning the entry is a number, or a "W", meaning it is a word to be executed. Even though the "micro monitor" is limited to only one entry per line, it has the interactive feel of the full outer interpreter. With the help of a look-up table detailing the CFA's of any words to be executed, an operator can fully exercise the power of the chip. All of the 133 words in the kernel of the single-chip Forth computer can be selectively executed by command from the system terminal using the "micro monitor". The scheme used with this interpreter does not require the dictionary portions of the language to function.

In the R65F11, the start-up portion of COLD is in the kernel. Neither ABORT nor QUIT is in the kernel. (Remember, the words that they in turn call amount to 80% of the language.) The code for the "micro monitor" is shown in Listing 3.

```
HEX
: MON            ( MICRO-MONITOR )
  BEGIN          ( START A NEVER ENDING INTERPRETATION LOOP )
    CR           ( MOVE TO NEW LINE )
    ." >"        ( OUTPUT A PROMPT, SAYING READY FOR INPUT )
    HEX          ( ALL ENTRIES TAKEN IN HEXIDECIMAL )
    QUERY        ( GET LINE OF INPUT FROM TERMINAL )
    0 0 TIB @
    (NUMBER)     ( PROCESS THE LINE TO GET NUMBER FROM LINE )
    2DROP        ( THROW AWAY ERROR INFO )
    TIB @ C@     ( GET THE FIRST CHARACTER FROM THE LINE )
    57 =         ( IS LETTER A "W"? )
    IF           ( IF IT IS, ITS NOT A NUMBER, BUT A WORD )
      EXECUTE    ( EXECUTE IT IMMEDIATELY )
    THEN         ( OTHERWISE THE NUMBER IS LEFT ON STACK )
  AGAIN ;        ( FOR USE BY WORD NEXT TIME AROUND )
```

Listing 3.

To enter the "micro monitor" the CTRL-R key combination is typed after power is applied and before a second reset is issued. The "micro monitor" prompts the operator with a ">" at the beginning of a new line. If the operator enters an "N" (actually any key other than a "W") the characters following to the null are interpreted as a number to be placed on the stack. If the first character is a "W", the number following is taken to be an address. The word at that address will be executed. After it finishes, control is passed back to the "micro monitor" which goes once again around the interpretive loop. Listing 4 shows an example of the system being exercised by the "micro monitor". [2]

```
>N1234          The number hex 1234 is entered
>WFEE4 1234     Stack is printed, FEE4 is CFA of .
>N0002          2 entered
>N0003          3 entered
>WF778          CFA of +
>WFEE4 5        Result is printed
>               Ready for next entry
```

Listing 4.

R6500/11 Host

The R65F11 is a ROM coded version of the R6500/11 single-chip microcomputer. This microcomputer chip can be a complete stand-alone computer system. It has on board processor, ROM, RAM, I/O components and special feature hardware. A unique feature of the R6500/11 that made it suitable to host the Forth kernel and become the R65F11, was its ability to maintain all of its microcomputer features and still address external memory. Under software control the R6500/11 can select a 16 Kbyte external memory space. It is difficult to say whether the R65F11 in final form should be classified as a microprocessor or microcomputer. From an internal viewpoint, it has all the elements of a stand-alone microcomputer including an internal mask ROM'd program. From an external view, it is a microprocessor that runs high level Forth code. The issue is further confused by its ability to boot programs from disk storage if an external user program cannot be found, which is a common attribute of advanced microcomputers with disk operating systems.

Processor Features. The processor of the host R6500/11 is an 8-bit CPU based on the proven 6502 architecture. The design is von Neumann in nature. Pipelining of instruction execution gives speed advantages over other 8-bit processor designs. It compares well in benchmark tests against other similar generation processors. It does not compare well against the new generation 16-bit processors. However, it requires very little silicon by comparison, which means that additional features can be added to the same chip (i.e., the kernel of Forth in ROM and the stacks in RAM). The CPU portion of the R65F11 occupies only 110 by 90 mils of silicon.

Register Set. Besides the Program Counter, there are no 16-bit registers in the R65F11's CPU. There are five 8-bit registers that are used for various functions in implementing the virtual Forth machine. The Accumulator is used to perform mathematical and logical functions. The Status Register is used to determine conditions for branches, etc. The Y register is used for various general purpose functions within the machine coded routines of the interpreters and primitives. Most notably the Y Register is used as part of an indirect addressing mode instruction that allows zero page RAM bytes to be used as concatenated 16-bit registers, allowing functions such as NEXT to be performed efficiently. The machine's stack pointer, the S Register, is used as the Forth return stack pointer. It operates in zero page, where the internal RAM of the R65F11 is located. The R65F11 has only one stack pointer so the X Register is used with indexing to form a software controlled stack. Although it would appear at first glance that this would considerably hinder the speed of operation, it does not make a great deal of difference. A pull of a byte from the normal processor stack takes 4 machine cycles. A load using indexing from zero page takes 4 cycles and an incrementation of the X Register another 2.

Instruction Set. The R6500/11 CPU is actually an enhanced version of the 6502, having four new instruction types added. These are the Bit Set (SMB), Bit Clear (RMB), Branch on Bit Set (BBS) and Branch on Bit Clear (BBC) instructions. Although these are useful instructions in a dedicated controller environment, they offered no advantage to the implementation of the Forth language and are not used in the Forth kernel.

Memory Features. The availability of both internal and external memory made the R65F11 possible. The sacrifice of two of the four available ports of the host R6500/11 microcomputer gives a multiplexed address and data bus capable of addressing 16 Kbytes of external memory. One TTL latch is required external to the part to capture the multiplexed address data. Internally the R65F11 has both permanent and temporary memory storage.

ROM. Current versions of the R65F11 have only 3 Kbytes of internal ROM. The Forth language, complete with dictionary and dictionary control words, requires about 8 Kbytes of permanent storage. The kernel of Forth, however, requires under 2.5 Kbytes. The ROM of the R65F11 contains the kernel and the RSC Forth Operating System, a total of 133 run time words built in.

The dictionary names and linkages are not included with the ROM. They are not needed at run time. The dictionary control words are of no use without a dictionary to manipulate, so they were omitted likewise. The kernel is otherwise surprisingly complete. All mathematical and logical operators, stack control, run time structures and even input/output formatting words are included.

A few other special feature words were included that are beyond the normal scope of a Forth kernel. They include a simple CASE statement handler, PROM programming support words and bank switching words to compensate for the limited addressing space. A primitive disk handler is also resident within the chip.

RAM. The R65F11 has 192 bytes of internal RAM. This is sufficient room to accommodate 30 levels of return stack storage, 50 levels of data stack entries and a number of system variables. There was not enough room to justify trying to use the area for the terminal input buffer or the user area, which are assigned an external address of hex 0300. Similarly, the user dictionary would be too limited if attempted to be kept internally. It is therefore assigned to be set externally at hex 0400 by default.

The internal RAM is used in a disk boot operation. The intention was to rely on as little external circuitry as possible in order to be able to load an operating boot program. The first 128 bytes after the system variables in RAM are used for that purpose.

Input/Output Features. Having onboard input/output features is a requirement of a microcomputer but not of a microprocessor. Thereby, the I/O features do not contribute to the R65F11's credibility as a Forth engine. They are, however, extremely important for the final applications the R65F11 was targeted for.

A fully duplex, advanced feature, hardward serial channel is provided on the R65F11. Baud rates and bit patterns are programmable. Virtually any of the standard asynchronous baud rates are attainable with proper crystal selection. The Operating System initializes it to communicate at 1200 baud (assuming a 1 MHz crystal) with one start bit, seven data bits and two stop bits with no parity.

Two eight-bit I/O ports are available to the user. They can be used individually as single inputs or outputs, or in parallel, as a printer port for instance. Two of these parallel lines are shared by the serial port. When the serial channel is in use the number of parallel lines available for other uses is decreased.

Special Features. The R65F11 has a number of other features that can only be labeled as special features. Like the other I/O features, these special features are not an essential part of the Forth engine portion of the R65F11 but do add considerably to the usability of the part.

Two 16-bit multi-mode counter/timers are provided on the R65F11. They can be used for a number of important real world functions, such as pulse counting, pulse generation, interval timing, as a source for periodic interrupts, etc.

Four edge sensitive lines, two positive and two negative, are implemented on port lines. These are useful in data communications tasks.

One of the sidelights of the R65F11 being hosted on a single chip microcomputer is that the ports used as the address and data bus can under program control be once again turned into ports. This unique feature means the R65F11 can "manually" take control of all its bus signal lines. It is this very feature that allows the R65F11 to program EPROM's in circuit with no external latches or special one shots.

Case Summary

At the time of the writing of this paper the R65F11 has been in production for only one year, yet several companies are already in volume production with Forth microcomputer based products. This is relatively unusual for a brand new processor. There are several board level development systems available, that cost less than the CRT terminals used to "talk" to them. They are being used for more product developments. This remarkably rapid design-in record is testimonial to the power of the "Forth on a chip" concept.

Selection of the Second Generation

Although there are many single-chip computers on the market, selecting one to be the second generation to the R65F11 was difficult. It was imperative to its success that the new chip be a significant architectural advancement over its predecessor. The other single-chip computers that were contemporaries of the R6500/11 would have all been down-featured when compared to the R65F11.

The 68200 is the first single-chip computer since the R6500/11 was introduced that is truly advanced enough to be considered as a Forth engine candidate. It has one of the most comprehensive sets of features offered in a single-chip computer to date. The memory space of the 68200 is program and data unified in structure. The 16-bit CPU has eight 16-bit data registers and six 16-bit address registers as well as 16-bit Stack Pointer, Status Register and Program Counter registers, and fast hardware multiply and divide instructions. Three 16-bit counter/timers are included as well as a full duplex serial channel. Onboard ROM in the initial version will be 2K words (4Kbytes). RAM provided in that version will be 128 words (256 bytes). Average instruction times are .5, 1 or 1.5 microseconds. Clearly this is a very suitable candidate to be the next generation ROM-coded Forth single-chip engine. The following benchmark figures were provided by Mostek. A complete benchmark is available from that source that shows a more complete range of comparisons. The relative performance marks are consistent with these, however.

A Simple Comparison of Processors

	SPEED (μsec)			SPACE (bytes)		
	68200	8096	8051	68200	8096	8051
MOVE	0.5	1.0	1.0	2	3	2
JMP	1.2	2.0	2.0	2	2	2
ADD	0.5	1.0	3.0	2	3	3
TOTAL	2.2	4.0	6.0	6	8	7

Case History: The F68K

The 68200 development was started approximately three years ago when Mostek clearly saw that the 68000 type architecture was superior to the then planned designs for other microcontrollers. Realizing software costs are a great deal of the development cost of a project, they decided to develop a high speed, high power processor/controller in single-chip form that had advanced programmability features. When the emulators became available in mid-1984, New Micros began preparing the ROM code for a targeted fall/winter 1984 F68K product introduction.

Language in ROM

The approach taken in the ROM code of the F68K closely follows that of the R65F11 philosophy. The headerless words of the Forth run time kernel are coded in internal ROM. The higher memory efficiency of the 16-bit instruction set and the additional 1 Kbyte of ROM available in the F68K (4K versus 3K) makes it possible to go beyond the functions of the R65F11. The "critical mass" of required memory in the R65F11 was about 2.5 Kbytes. After the basic run time functions of the language were in place in that amount of memory, many other functions were added to the remaining .5 Kbytes. These included CASE statements, bank switching, PROM programming, disk read and write primitives and the micro monitor. The F68K uses the extra space to accommodate enhanced versions of all these functions. Many of the primitives usually written in high level language are re-written in machine code to enhance the system operating speed.

The F68K version of NEXT far outperforms that of the R65F11 as shown in listing 5.

```
MOVE (IP)+,W
MOVE (W)+,A0
JMPA (A0)
```

Listing 5.

This version of NEXT is three instructions and six bytes long. It executes in 3 microseconds and can be placed inline in the definition, rather than jumped to, for additional time savings. It is almost 15 times faster than the NEXT of the R65F11.

As a comparison, if NEXT were micro-coded in the F68K, it would of course be faster. The performance improvement is not earthshattering, however. Note that an indirect threaded NEXT must make three memory references, one for the NEXT op code and two to access the next word's CFA. The normal machine code does the same memory references, but uses three op codes. In other words the normal machine code does two extra memory references. This is only one microsecond longer than the proposed micro-coded version. Would the 33% speed improvement justify the added complexity of modifying the microcode? This is in the range of performance that could easily be achieved by using a slightly faster crystal. The F68K approaches the limit of what could be expected of any Forth engine having a similar memory access time.

68200 Host

The F68K is a ROM coded version of the 68200 single-chip microcomputer. Like the R6500/11, this microcomputer chip can be a complete stand-alone computer system. It has on board processor, ROM, RAM, I/O components and special feature hardware. It also has a special feature similar to that of te R6500/11 that makes it suitable to host the Forth kernel, which is its ability to maintain all of its microcomputer features and still address external memory. Under software control the 68200 can select a 16 Kbyte external memory space.

The F68K is even more difficult to classify in final form as either a microprocessor or a microcomputer. Like the R65F11, it has all the elements of a stand-alone microcomputer including an internal mask ROM'd program. From an external view, it is a microprocessor that runs high level Forth code. Unlike the R65F11, the F68K can actually be programmed in high level language with no other chips required. All RAM pointers are directed to internal memory on power up. A limited dictionary can be written internally with the improved micro monitor program interactively.

Processor Features. The F68K CPU is designed as a 16-bit revision of the 68000. Although it is actually a significantly different processor, the philosophy of 68000 design was closely adhered to. It compares well in benchmark tests against all other 8- and 16-bit processors. The post incrementing and pre-decrementing address modes appreciably improve the parts performance. It has approximately the same number of registers as its namesake, although they are half as long (16 as opposed to 32 bits).

The F68K has seventeen 16-bit registers in its CPU. There are eight 16-bit data registers and six 16-bit address registers that are used for various functions in implementing the virtual Forth machine. Two address registers are used for the Instruction Pointer and W. Another address register is used to maintain the data stack. The machine's stack pointer is used as the Forth return stack pointer.

The availability of both internal and external memory made this Forth engine implementation possible. The sacrifice of 24 I/O lines of the 40 available in the host F68KT microcomputer gives a multiplexed 16-bit address and data bus capable of addressing 16 or 64 (or no) Kbytes of external memory depending on the mode selected. Two TTL latches are required external to the part to capture the multiplexed address data. Internally the F68K has both permanent and temporary memory storage.

Current versions of the F68K have 4 Kbytes of internal ROM. The Forth language, complete with dictionary and dictionary control words requires about 8 Kbytes of permanent storage. The kernal of Forth, however, requires under 2.5 Kbytes. The ROM of the F68K contains the kernel and the Forth Operating System.

The F68K has 256 bytes of internal RAM. This is sufficient room to accommodate 30 levels of return stack storage, 50 levels of data stack entries and a number of system variables. Unlike the R65F11, this area is also used for the terminal input buffer and the user area at start-up time, as determined by the operating system. Determination of a lack of a useful external memory map by COLD will cause all pointers to be initialized to internal RAM. Although crowded, there is enough RAM to be usable for interactive system checkout and limited test word programming.

Input/Output Features. As was the case with the R65F11, the I/O features of the F68K do not contribute to its credibility as a Forth engine. They are, however, extremely important for the final applications the chip was targeted for.

A fully duplex, advanced feature, hardware serial channel is provided on the F68K. Baud rates and bit patterns are programmable. Virtually any of the standard asynchronous baud rates are attainable with proper crystal selection. Asynchronous baud rates up to 375K bps and synchronous rates up to 1.5Mbps are possible.

Two 8-bit I/O ports are available to the user. They can be used individually as single inputs or outputs, or in parallel, as a printer port for instance. Two of these parallel lines are shared by the serial port. When the serial channel is in use the number of parallel lines available for other uses is decreased.

Special Features. Like the R65F11, the F68K has a number of other features that can only be labeled as special features. Like the other I/O features, these special features are not an essential part of the Forth engine portion of the chip but do add considerably to the usability of the part.

There are 16-bit multi-mode counter/timers provided on the F68K. They can be used for a number of important real world functions, such as pulse counting, pulse generation, interval timing, as a source for periodic interrupts, etc.

Four edge sensitive lines, two positive and two negative, are implemented on port lines. These are useful in data communications tasks.

Because the F68K is a single-chip microcomputer it can use its ports as programmable address and data busses. This means the F68K can "manually" take control of all its bus signal lines, just as the R65F11 did. It is this very feature that allows the EEPROM programming in circuit with no extra external latches or special one shots, etc. Beyond the capabilities of the R65F11, the F68K has bus arbitration circuitry built in with bus request and bus grant signals that makes it a natural for multiple processor systems and DMA target transfers. This is particularly attractive when it is remembered that the majority of time the CPU will be accessing internal ROM, running internal Forth primitives, accessing the external bus for data and CFA's from high level language words' parameter fields.

Conclusion

The R65F11 was the first commercially available single-chip Forth engine. The F68K will soon join it as the next generation Forth engine. The special features of these parts make them ideal for small, dedicated applications. Without having these features built into the chip, external hardware would have to be added to make a system. If larger systems had to be built, where would be the advantage of an engine over a conventional processor with the language in ROM? (Remember the ill-fated Pascal engine.) In such situations, speed is the primary, if not the only, advantage of an engine. By far the greatest number of applications that need Forth are more features critical than time critical. Now the F68K with its advanced 16-bit architecture can run Forth code about as fast as the R65F11 executed its assembly language. Would "Forth machine code" be even faster? By the size of effort required to start a chip design from scratch to achieve micro-coded engine performance, the anticipated advantages must be very significant to justify it. Certainly the F68K performance makes it an excellent real time Forth engine.

References

[1]. Western Digital Product Handbook, "WD9000 Pascal Microprocessor Chip Set".
[2]. Dumse, R., "High Level Language in Single Chip Microcomputers", MIDCON 81 Professional Program Session, Record 25.
[3]. Dumse, R., "New Programming Philosophy for Dedicated Applications", ELECTRO 82 Professional Program Session, Record 25.
[4]. Dumse, R. and Smith, D., "High Level Language Solutions for Dedicated Applications", WESCON 82 Professional Program Session, Record 17B.
[5]. *RSC-FORTH User's Manual*, Document #29761n51, October 1983, Rockwell International.
[6]. Dumse, R., "The Smallest Outer Interpreter", *Rochester FORTH Conference Proceedings*, 1984.
[7]. Thomas, O. and Amantneek, K., "Basis of FORTH", *1982 FORTH Modification Laboratory Proceedings*.

Manuscript received May 1984.

Mr. Dumse graduated from the University of North Iowa in 1975 with a B.A. in Physics. He headed development of the Rockwell R65F11 and is interested in dedicated computer systems and microprocessor technology. As president of New Micros, Inc., he developed a board level version of the Rockwell chip, the "100 Squared", and is currently involved in the development of the F68K.