High Performance Controller in Information Control Applications

National Semiconductor
Application Note 585
Steve McRobert
June 1989

ABSTRACT

This paper describes National Semiconductor's HPC™ family of High Performance microControllers. Included are two examples showing how the devices are used in actual Information Control applications.

The architecture, technology, and instruction set of the HPC family are presented, with emphasis on how these features are appropriate for use in microcontroller based information control systems. Two example applications are given, the first being the use of a single chip mode HPC as an I/O processor and interrupt handler in a laser beam printer. In this case the HPC acts as a slave to the main 32-bit CPU in the printer, freeing it from the many tasks which require fast interrupt response and thus improves system throughput. The second example shows the HPC used in expanded mode as the sole microprocessor in an ESDI to SCSI bridge adapter card. The operations performed by the HPC in this application are used as an example of how the instruction set and addressing modes work together to achieve high throughput. The paper concludes with a brief discussion of the future of the HPC family of devices.

INTRODUCTION

The HPC (High Performance Controller) family of microcontrollers was designed by National Semiconductor as the first of a new generation of 16-bit CMOS microcontrollers.

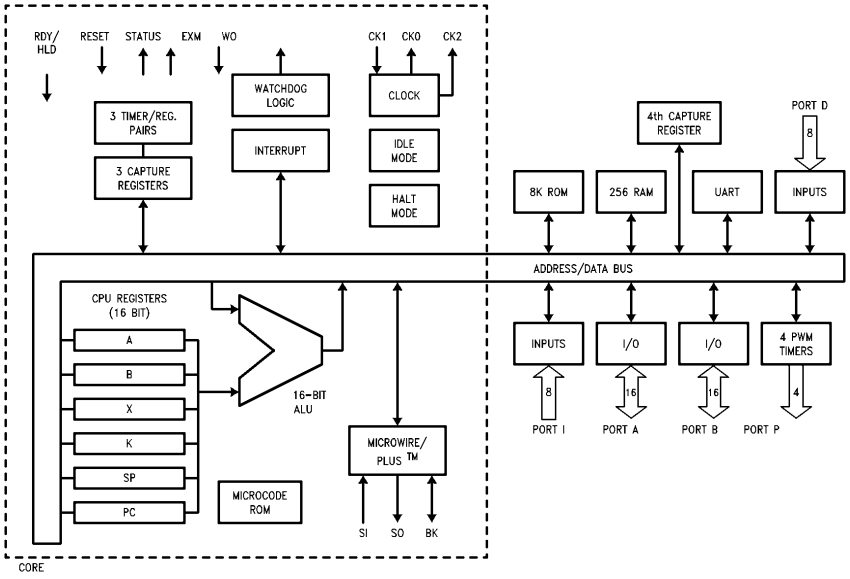
The intention was to start afresh, using the experience gained from earlier device families and, without software compatibility constraints, to create an architecture sufficiently advanced to be competitive for 10 years or more. Other design goals were to minimize device complexity, thus allowing for dependable, economical, high volume production, and to make HPC easy to understand so that system designers could readily convert designs to use the new family's advanced features.

These goals have been met, and, since the first device was sampled in early 1986, the HPC family has developed into a well proven solution to many design problems.

ARCHITECTURE

The HPC family is based on a core concept. All devices share a common core including the CPU and a base set of peripherals such as timer/counters etc. *Figure 1* shows a block diagram of the HPC16083 with the core emphasized at left. HPC uses a memory-mapped Von Neuman architecture, in which all registers, I/O ports, peripherals etc. are assigned memory locations in one uniform address space.

This includes the CPU registers *(Figure 1)*, allowing all HPC instructions to operate on every register in the programmer's model. Such uniformity simplifies the work of the assembly language programmer and the writer of the C compiler, making the HPC a particularly efficient microcontroller for running programs written in ''C''.



TL/DD/10346–1

FIGURE 1. HPC16083 Block Diagram

HPC™ is a trademark of National Semiconductor Corporation.

RRD-B30M105/Printed in U. S. A.

The core is connected to peripherals and on-chip memory by a 16-bit address/data bus, which is multiplexed to reduce die size. This bus is brought out on the A port when the device is used in expanded and/or ROMless modes, allowing off-chip devices to be accessed in exactly the same fashion as on-chip memory or peripherals.

When writing assembly language or C instructions the programmer perceives no difference between on-chip and off-chip memories, but both assembler and compiler take account of two key differences. When the HPC is run at high oscillator frequencies (up to 30 MHz on current production devices) a wait state must be applied for accesses to external memories or peripherals, but are never applied to on-chip RAM or registers. The other difference is that accesses to on-chip locations with addresses below 100 hexadecimal (called basepage accesses) require only a one byte address, so are thus shorter and faster than accesses to non-basepage locations *(Figure 2)*.

FFF:FFF0	INTERRUPT VECTORS	HPC16083 ON-CHIP ROM SPACE
FFEF:FFD0	JSRP VECTORS	
FFCF:E000	GENERAL PURPOSE ROM	
DFFF:0200	EXPANDED MODE ADDRESS SPACE	EXTERNAL USER MEMORY
0IFF:0IC0	ON-CHIP RAM	ON-CHIP RAM AND REGISTERS
0IBF:00C0	ON-CHIP REGISTERS	
00BF:0000	ON-CHIP RAM	

FIGURE 2

The programmer must choose which variables to put into on chip RAM or the basepage to achieve maximum performance and code efficiency.

Basepage RAM, because it is very fast and efficient to use, provides many of the benefits of the register file architecture used on some other microcontrollers. The HPC is different, however, in that it has a small set of registers: Accumulator, B pointer, X pointer and K (or limit) register. These registers all have addresses and can be used as general purpose memory locations, but are best used for their special functions. Many HPC instructions have two operands, the source and the destination. If the Accumulator (A) register is used as the destination, this is implied in the opcode and the address of A need not be included in the instruction, thus making it shorter and faster than instructions using another memory location as the destination. If the address of the source is contained in the B register then this too can be implied from the opcode and the whole instruction becomes one byte long.

Most HPC instructions thus have a single-byte form, using the B or X register as a pointer to the memory location being accessed.

The use of the K register will be discussed in the next section.

The primary objective when designing the architecture and instruction set of HPC was to minimize code size, an approach which can reduce throughput if unlimited bus bandwidth is available. In typical microcontroller applications the use of external memory is undesirable for board space and cost reasons. If the code is too large for mask ROM, the best solution in terms of space and cost is a single, relatively slow, EPROM.

In this situation of low bus bandwidth, the high byte efficiency of the HPC goes hand-in-hand with good performance.

ADDRESSING MODES

In keeping up with the HPC philosophy of being simple and quick to understand, the HPC instruction set *(Figure 3)* has relatively few mnemonics. This is because for those instructions with one or two addressable operands the same mnemonic is used regardless of the addressing mode, operand size (byte or word) or address size (depending upon whether each operand is in the basepage or not). Each individual memory location may be addressed using one of the following addressing modes:

Direct: The 8- or 16-bit address is included in the series of bytes that make up the instruction.

Indirect: The 8-bit address of a word in the base page is included in the instruction. The contents of this word are used as a pointer to the variable to be accessed.

Mnemonic	Description	Action	
ARITHMETIC INSTRUCTIONS			
ADD	Add	MA + MemI ⟶ MA	carry ⟶ C
ADC	Add with carry	MA + MemI + C ⟶ MA	carry ⟶ C
ADDS	Add short imm8	MA + imm8 ⟶ MA	carry ⟶ C
DADC	Decimal add with carry	MA + MemI + C ⟶ MA (Decimal)	carry ⟶ C
SUBC	Subtract with carry	MA − MemI + C ⟶ MA	carry ⟶ C
DSUBC	Decimal subtract w/carry	MA − MemI + C ⟶ MA (Decimal)	carry ⟶ C
MULT	Multiply (unsigned)	MA*MemI ⟶ MA & X, 0 ⟶ K, 0 ⟶ C	
DIV	Divide (unsigned)	MA/MemI ⟶ MA, rem. ⟶ X, 0 ⟶ K, 0 ⟶ C	
DIVD	Divide Double Word (unsigned)	(X & MA)/MemI ⟶ MA, rem ⟶ X, 0 ⟶ K, carry ⟶ C	
IFEQ	If equal	Compare MA & MemI, Do next if equal	
IFGT	If greater than	Compare MA & MemI, Do next if MA > MemI	
AND	Logical and	MA and MemI ⟶ MA	
OR	Logical or	MA or MemI ⟶ MA	
XOR	Logical exclusive-or	MA xor MemI ⟶ MA	
MEMORY MODIFY INSTRUCTIONS			
INC	Increment	Mem + 1 ⟶ Mem	
DECSZ	Decrement, skip if 0	Mem − 1 ⟶ Mem, Skip next if Mem = 0	

FIGURE 3. HPC Instruction Set Description

Mnemonic	Description	Action
BIT INSTRUCTIONS		
SBIT	Set bit	1 → Mem.bit
RBIT	Reset bit	0 → Mem.bit
IFBIT	If bit	If Mem.bit is true, do next instr.
MEMORY TRANSFER INSTRUCTIONS		
LD	Load	MemI → MA
	Load, incr/decr X	Mem(X) → A, X ±1 (or 2) → X
ST	Store to Memory	A → Mem
X	Exchange	A ↔ Mem
	Exchange, incr/decr X	A ↔ Mem(X), X ±1 (or 2) → X
PUSH	Push Memory to Stack	W → W(SP), SP+2 → SP
POP	Pop Stack to Memory	SP−2 → SP, W(SP) → W
LDS	Load A, incr/decr B, Skip on condition	Mem(B) → A, B ±1 (or 2) → B, Skip next if B greater/less than K
XS	Exchange, incr/decr B, Skip on condition	Mem(B) ↔ A,B ±1 (or 2) → B, Skip next if B greater/less than K
REGISTER LOAD IMMEDIATE INSTRUCTIONS		
LD B	Load B immediate	imm → B
LD K	Load K immediate	imm → K
LD X	Load X immediate	imm → X
LD BK	Load B and K immediate	imm → B,imm → K
ACCUMULATOR AND C INSTRUCTIONS		
CLR A	Clear A	0 → A
INC A	Increment A	A + 1 → A
DEC A	Decrement A	A − 1 → A
COMP A	Complement A	1's complement of A → A
SWAP A	Swap nibbles of A	A15:12 ← A11:8 ← A7:4 ↔ A3:0
RRC A	Rotate A right thru C	C → A15 → . . . → A0 → C
RLC A	Rotate A left thru C	C ← A15 ← . . . ← A0 ← C
SHR A	Shift A right	0 → A15 → . . . → A0 → C
SHL A	Shift A left	C ← A15 ← . . . ← A0 ← 0
SC	Set C	1 → C
RC	Reset C	0 → C
IFC	IF C	Do next if C = 1
IFNC	IF not C	Do next if C = 0
TRANSFER OF CONTROL INSTRUCTIONS		
JSRP	Jump subroutine from table	PC → [SP],SP+2 → SP W(table#) → PC
JSR	Jump subroutine relative	PC → [SP],SP+2 → SP,PC+# → PC (#is +1025 to −1023)
JSRL	Jump subroutine long	PC → [SP],SP+2 → SP,PC+# → PC
JP	Jump relative short	PC+# → PC(# is +32 to −31)
JMP	Jump relative	PC+# → PC(#is +257 to −255)
JMPL	Jump relative long	PC+# → PC
JID	Jump indirect at PC + A	PC+A+1 → PC
JIDW		then Mem(PC)+PC → PC
NOP	No Operation	PC + 1 → PC
RET	Return	SP−2 → SP,[SP] → PC
RETSK	Return then skip next	SP−2 → SP,[SP] → PC, & skip
RETI	Return from interrupt	SP−2 → SP,[SP] → PC, interrupt re-enabled

Note: W is 16-bit word of memory

MA is Accumulator A or direct memory (8 or 16-bit)

Mem is 8-bit byte or 16-bit word of memory

MemI is 8- or 16-bit memory or 8 or 16-bit immediate data

imm is 8-bit or 16-bit immediate data

imm8 is 8-bit immediate data only

FIGURE 3. HPC Instruction Set Description

```
        LD  X,#0100              ; Point to beginning of source code
        LD  BK,# 0400, #0600;P   ; Point to beginning & end of target
LOOP:   LD  A, [X+].W            ; Get word from source block
        XS  A, [B+].W            ; Store it at target
        JP  LOOP
```

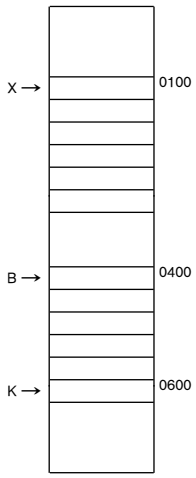


FIGURE 4. Word Block Move

Indexed: As Indirect, but with an 8- or 16-bit immediate offset added to the pointer.

Register Indirect: As indirect, but the B or X registers are used as pointers, with their addresses implied in the opcode.

Immediate: Only for the source in two-operand instructions. An 8- or 16-bit immediate value is included in the instruction.

The first four addressing modes are used both for single operand instructions e.g. bit set, bit clear, bit test, increment, decrement, and two operand instructions such as ADD and LD.

Direct and immediate modes can be used in combination, allowing operations to be performed directly on memory or registers without using the accumulator.

Two variables, each byte or word, each located anywhere in memory, can be compared, added, divided or have any of the other two-address instructions performed on them. This improves the byte-efficiency of the HPC, and enhances the power of the instruction set in that it takes less lines of assembly code to perform a given function than it would for earlier, completely accumulator-based CPUs.

An important benefit provided by the indirect and indexed modes is that any of the 96 words of RAM or the basepage registers, such as port A or the accumulator, may be used as pointers.

There are two special addressing modes which are used only with the LD and X (exchange) instructions. These modes are called auto increment/decrement and auto increment/decrement with conditional skip, and their use is illustrated by the example shown in *Figure 4*.

This example uses the B pointer, the X pointer and the K register to move a block of data one word at a time. Some points to note are that the LD BK instruction initializes both registers with one instruction, and that both the LD and XS instructions increment the pointer by two because two bytes (one word) are moved. The S in XS signifies the conditional skip. After A has been exchanged with the word pointed to by B, B is incremented, then compared with K. If B is greater than K (or, for an XS A, [B−] instruction, if B is less than K) the next statement is skipped over, thus terminating the loop. This example epitomizes the approach taken in designing the HPC family.

String operations are built up from simple data movement instructions, allowing them to be interrupted at any time with no need for complex re-start or recovery schemes.

INSTRUCTION SET

The HPC instruction set is noticeably different from other 16-bit controllers, in that many of its instructions are single byte. How this is achieved can be seen by looking at the opcode map *(Figure 5)*.

Instructions such as bit manipulation operations and single byte jumps (JP) use many opcodes for the same mnemonic. This is because information, such as the jump length for JP, is coded into the opcode.

This makes these instructions very efficient, and enhances the performance of the HPC in information control applications, where decision making and bit manipulation operations tend to be important.

All of the arithmetic, comparison, logical and data movement instructions have a single byte form using register indirect addressing mode. The opcode space ''used up'' by having many opcodes for a few instructions is restored by using addressing mode prefixes for the less commonly used addressing modes. These make instructions using these modes one byte longer, but the use of these prefixes allows all of the two address instructions to use all of the addressing modes. Without the prefixes the HPC would run out of opcode space and restrictions would have to be placed on some instructions, making the assembly language much harder to use and the C compiler harder to write. Examples are given in *Figure 6* of several combinations of instructions and addressing modes, with execution times for systems using low cost external memories.

C.13 HPC OPCODE MAP
LSB/MSB →

	0	1	2	3	4	5	6	7
0	CLR A	IFBIT 0	JSRP 0	JSR +	JP +1*	JP +17	JP 0	JP −16
1	COMP A	IFBIT 1	JSRP 1	JSR +	JP +2	JP +18	JP −1	JP −17
2	SC	IFBIT 2	JSRP 2	JSR +	JP +3	JP +19	JP −2	JP −18
3	RC	IFBIT 3	JSRP 3	JSR +	JP +4	JP +20	JP −3	JP −19
4	INC A	IFBIT 4	JSRP 4	JSR −	JP +5	JP +21	JP −4	JP −20
5	DEC A	IFBIT 5	JSRP 5	JSR −	JP +6	JP +22	JP −5	JP −21
6	IFNC	IFBIT 6	JSRP 6	JSR −	JP +7	JP +23	JP −6	JP −22
7	IFC	IFBIT 7	JSRP 7	JSR −	JP +8	JP +24	JP −7	JP −23
8	SBIT 0	RBIT 0	JSRP 8	RBIT X	JP +9	JP +25	JP −8	JP −24
9	SBIT 1	RBIT 1	JSRP 9	SBIT X	JP +10	JP +26	JP −9	JP −25
A	SBIT 2	RBIT 2	JSRP 10	IFBIT X	JP +11	JP +27	JP −10	JP −26
B	SBIT 3	RBIT 3	JSRP 11	SWAP A	JP +12	JP +28	JP −11	JP −27
C	SBIT 4	RBIT 4	JSRP 12	RET	JP +13	JP +29	JP −12	JP −28
D	SBIT 5	RBIT 5	JSRP 13	RETSK	JP +14	JP +30	JP −13	JP −29
E	SBIT 6	RBIT 6	JSRP 14	RETI	JP +15	JP +31	JP −14	JP −30
F	SBIT 7	RBIT 7	JSRP 15	POP	JP +16	JP +32	JP −15	JP −31
	8	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	Dir−Dir	LD A,i	Dir−Dir	LD A,ii	LDS [B+].b	LD [X+],b	LDS [B+].w	LD [X+].w
1	Dir−Dir	LD K,i	Dir−Dir	LD K,ii	XS [B+].b	X [X+],b	XS [B+].w	X [X+].w
2	Imm−Dir	LD B,i	Index	LD B,ii	LDS [B−].b	LD [X−],b	LDS [B−].w	LD [X−].w
3	Imm−Dir	LD X,i	—	LD X,ii	XS [B−].b	X M[X−],b	XS [B−].w	X [X−].w
4	Dir−Dir	JMP+	Dir−Dir	JMPL	LD [B].b	LD [X].b	LD [B].w	LD [X].w
5	Dir−Dir	JMP−	Dir−Dir	JSRL	X [B].b	X [X].b	X [B].w	X [X].w
6	Imm−Dir	Direct	Index	Direct	ST [B].b	ST [X].b	ST [B].w	ST [X].w
7	Imm−Dir	LD bd,i	LD BK,ii	LD wd,ii	SHR A	RRC A	SHL A	RLC A
8	LD A,bd	ADD A,i	LD A,wd	ADD A,ii	ADC A,b	ADD A,b	ADC A,w	ADD A,w
9	INC bd	AND A,i	INC wd	AND A,ii	DADC A,b	AND A,b	DADC A,w	AND A,w
A	DECSZ bd	OR A,i	DECSZ wd	OR A,ii	DSUBC A,b	OR A,b	DSUB A,w	OR A,w
B	ST A,bd†	XOR A,i	ST A,wd†	XOR A,ii	SUBC A,b	XOR A,b	SUBC A,w	XOR A,w
C	LD bd,bd	IFEQ A,i	LD wd,wd	IFEQ A,ii	JID	IFEQ A,b	JIDW	IFEQ A,w
D	LD BK,i	IFGT A,i	Indirect	IFGT A,ii	—	IFGT A,b	—	IFGT A,w
E	X A,bd	MULT A,i	X A,wd	MULTA,ii	—	MULT A,b	—	MULT A,w
F	XIndirect	DIV A,i	PUSH	DIV A,ii	DIVD A,b	DIV A,b	DIVD A,w	DIV A,w

— = opcode is reserved for future use.

b = byte of memory

bd = direct byte of memory

i = 8-bit immediate value

w = word of memory

wd = direct word of memory

ii = 16-bit immediate value

Dir−Dir, Imm−Dir, Index, Direct, Indirect and XIndirect are all Addressing Mode directives.

Notes:

*NOP is the same as JP + 1 and has the same opcode.

†These opcodes are LD if prefixed by Dir−Dir or Imm−Dir directive.

FIGURE 5

		20 MHz	30 MHz
CLR	A	300 ns	200 ns
RRC	A	400 ns	267 ns
LD	B, H'3CF2	600 ns	400 ns
IFBIT	7,[B].B	800 ns	533 ns
ST	A,38.W	900 ns	600 ns
JSR		1.10 μs	733 μs
JSRL		1.40 μs	933 μs
ADC	[H'10].W, [H'20].W	1.70 μs	1.13 μs
DSUBC	[H'A0].W, [H'B0].W	2.00 μs	1.33 μs
MULT	A, [B].W	5.90 μs	3.93 μs
DIVD	A,[X].W	6.40 μs	4.27 μs

Times Calculated with 1 Wait State Inserted

FIGURE 6. Typical Execution Times

There are many more powerful features of the HPC instruction set, but space does not permit describing them here. For more information see the documents listed in the references section.

TECHNOLOGY

The HPC family and nearly all other new National Semiconductor analog and digital VLSI devices are fabricated in an advanced double metal process called M2CMOS. This is a very high speed process, as shown by the current production two micron (drawn) HPC46083, which is available as a 30 MHz version.

The HPC family has been migrated to a 1.5 micron (drawn) process for the first part with an analog to digital converter on chip, the HPC46164.

National Semiconductor already manufactures the NS32532 microprocessor in 1.25 micron M2CMOS, and will shrink this process still further in the future. The HPC devices will be migrated to these smaller geometries and will benefit from other process developments such as on chip EPROM.

INFORMATION CONTROL APPLICATIONS

Laser Beam Printer Front End Processor

This section describes a customer's application for an HPC46083 used in single chip mode. It makes use of the Universal Peripheral interface (UPI) port which is a feature of all HPC devices with on-chip mask ROM.

The UPI port allows an HPC device to be used as a peripheral to a host processor, connected to the host via its data bus. The HPC in UPI mode appears to the host to be a peripheral device such as a UART, but provides additional processing power, relieving the host of interrupt-intensive tasks and thus improving the host's performance.

The UPI port of the HPC provides status signals to both the HPC CPU and that of the host which ensure that no data is lost when the CPUs communicate.

In the laser beam pointer (LBP) application *(Figure 7)*, the HPC handles the serial and Centronics interfaces of the printer, buffering received characters and interrupting the host CPU when a block of up to 128 characters has been received. When the host CPU (a National Semiconductor NS32CG16 printer/display controller) is interrupted it then transfers the whole block of data into its own memory very rapidly.

This approach reduces the number of interrupts received by the 32CG16 by a factor of over 100 compared to a solution using a conventional UART while being simpler, cheaper and offering higher system performance than using a DMA approach. These overhead reductions are very important in LBP systems, because the main CPU must keep up with the paper movement, otherwise image data will be lost.

In addition to improving printer performance, the HPC reduces the system cost by providing functions that would otherwise need extra devices. The HPC acts as the interrupt controller for the 32CG16, generating an interrupt signal to it and then placing the interrupt vector on the UPI port when the 32CG16 acknowledges the interrupt. Another function provided by the HPC is an intelligent interface to the printer front panel displays and push buttons controlling such functions as LCD contrast. Finally, the HPC implements a serial interface to the electronic subsystem of the printer engine itself, providing diagnostic capability to the 32CG16. For all of these functions, the HPC performs first-level error checking, further relieving the main CPU of minor tasks.

The LBP is at one extreme of the range of HPC applications, where the HPC uses virtually nothing but its on-chip peripherals and memories.

The next section deals with an application towards the other end of the range.

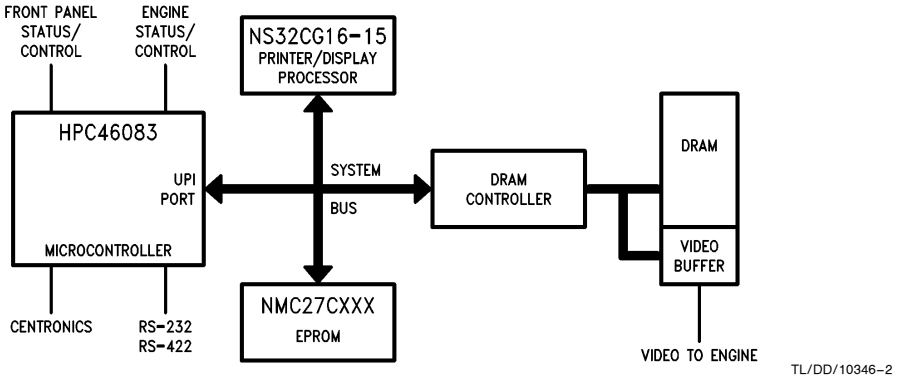
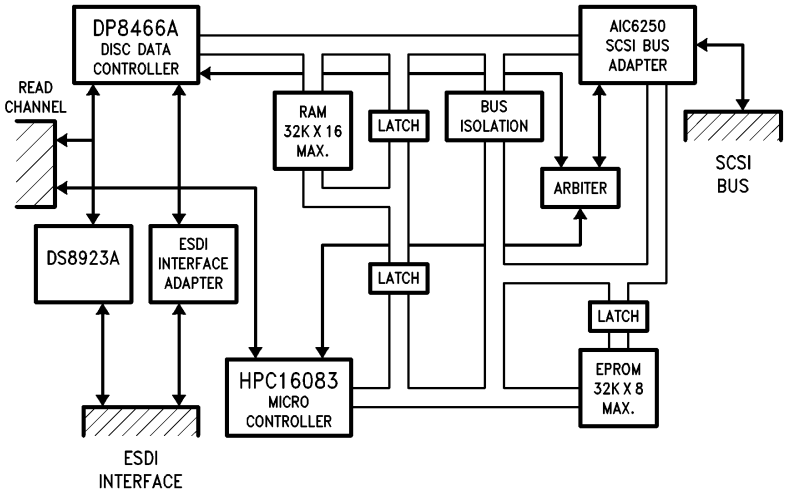


FIGURE 7



TL/DD/10346–3

FIGURE 8

SCSI Bridge Adapter

The fast growing usage of Winchester disk drives in the Small Computer System Interface (SCSI) environment has provided another important market for the HPC family.

The HPC architecture is well suited for use in embedded SCSI systems, as the peripherals such as the SCSI interface device may be memory mapped into the HPC address space, allowing bit and byte manipulation operations to be performed directly on the registers of the peripheral using single assembly language instructions. Many SCSI interface devices are relatively unintelligent, requiring the CPU to perform many bit test, set, and clear operations to set up a data transfer operation. Most other microcontrollers need up to three instructions to set a bit in one of these peripherals, thus reducing drive performance.

National Semiconductor has produced an ESDI-to-SCSI bridge adapter board, which demonstrates the use of the HPC46003 and the DP8466A disk data controller in a real synchronous SCSI system. A software package has been written in HPC assembly language which implements the SCSI common command set and is available in source code form to companies wishing to use the HPC in embedded SCSI or host adapter designs.

The code was written in HPC assembly language because for very high volume, cost sensitive designs, like a disk drive, the extra development cost of writing in assembler is outweighed by the advantages of reduced code size and improved performance.

The adapter board design *(Figure 8)* uses the HPC46003 running in 8-bit mode with a single EPROM providing program memory. Data memory is provided by the 256 bytes of on-chip RAM which provides fast scratch pad and stack space.

One important function in embedded SCSI disk drives is logical to physical address conversion, in which a logical address (typically 24 bits) is divided twice by constants, the result and the two remainders being the head, cylinder and sector numbers.

The HPC is capable of dividing a 32-bit number by a 16-bit number in under four microseconds, thus providing a dramatic improvement in logical to physical address conversion time compared to earlier 8-bit microcontroller solutions. As a final point in this necessarily brief discussion, the HPC uses very little power due to its advanced CMOS manufacturing technology. This is important in disk drive applications, where low power consumption is an important performance parameter for the end product.

CONCLUSION AND FUTURE DEVELOPMENTS

This paper has discussed the design of the HPC family and described two actual applications in important market areas. The development work performed for these and other projects has shown that the HPC architecture provides very high performance in embedded control applications.

The plans for future products are to take the high performance core and add various combinations of peripherals, thus allowing the family to reach a wide range of markets. *Figure 9* shows some of the current and future devices.

HPC16083	8K ROM, 256 RAM
HPC16003	ROMless, 256 RAM
HPC16400	256 RAM, 2 HDLC + 4 DMA Channels
HPC16164	16K ROM, 512 RAM, 8 Channel ADC
HPC16064	16K ROM, 512 RAM
HPC16104	ROMless, 8 Channel ADC
HPC16004	ROMless, 512 RAM
HPC167164	16K EPROM, 512 RAM, 8 Channel ADC

FIGURE 9. HPC Family Devices Principal Features

REFERENCES

National Semiconductor Application Note AN-510: *Assembly Language programming for the HPC.*

National Semiconductor Publication Number 424410897-001A July 1987: *HPC16083/HPC16043/HPC16003 User's Manual.*

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408