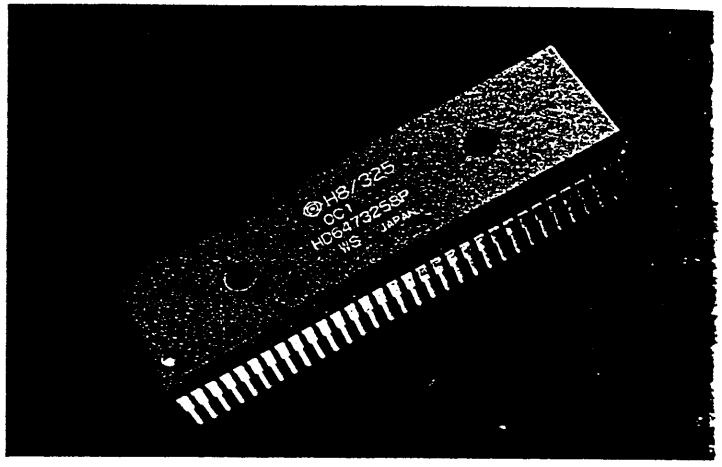
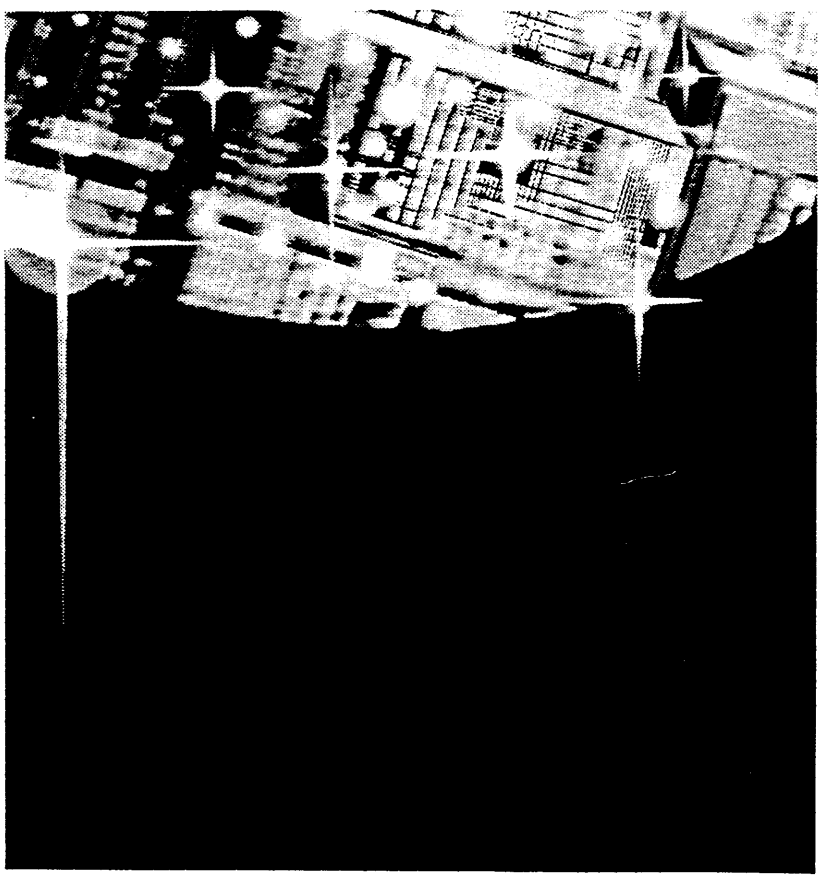
H8/300 SERIES
microcontrollers



HITACHI

H8–300
MICROCONTROLLERS



INDEX

H8 / 300

Introduction

Microcontrollers, in many shapes and forms, have now penetrated many aspects of everyday life. You will often find microcontrollers in the car or train that gets you to work, perhaps in your company's coffee machine, and much of your workplace's office and production equipment.

Indeed, it is safe to say that you will own at least one piece of equipment which contains a microcontroller, whether its a sophisticated Hi-Fi, a feature phone or an automatic washing machine.

The embedded software within such a system is particularly important, as it provides the fuctionality, advanced features and sometimes, unfortunately, short-comings. So the quality of this software is crucial to your products success.

To ensure that programmers produce top quality software for their microcontrollers, many companies would now like to base their developments on a High Level Language (HLL), such as "C" or Pascal.

The advantages of using HLL's include programmer productivity and easier software maintenance.

Using a HLL, code can be generated much faster than when using assembler as the code is written in English-like statements (such as IF, ELSE, WHILE). So using a HLL you are able to write application code much more efficiently.

HLL's also offer another great advantage - portability! The movement of application code, from one project to the next, and even to a new microcontroller can be easily achieved using a HLL, by selecting a different compiler to suit your new choice of architecture. This means embedded software, which is very costly to develop can be used in many projects; saving development resource and reducing the time to market for your products.

But the requirement for HLL in a microcontroller system is not always straightforward.

Traditionally microcontrollers have been limited to very simple accumulator bound CPU architectures, and a HLL can only produce large, slow and unwieldy code to run on them.

This is not the case with the H8/300 series of microcontrollers. As well as offering all the features you would expect in a microcontroller, such as on chip memory and peripherals, it also contains a powerful 8 bit CPU. This CPU has an architecture and instruction set well tuned to run code produced by modern HLL compilers, especially "C" compilers.

As the Hitachi H8/300 family

couples extremely high memory and peripheral integration, with the capability of efficient execution of the "C" language. You can now seriously consider moving your development up to the C level.

The ZTAT ™ Concept

ZTAT™ (Zero Turn Around Time) is the Hitachi way to offer its customers total flexibility when taking their products from conception through to final production.

ZTAT™ provides both one time programmable CMOS EPROM on chip and mask programmable devices, in identical packages. By offering ZTAT we enable our customers to easily progress from the prototype stage, through to pilot production (both of which can use ZTAT). Final production could use either ZTAT for small/ medium volume or a mask ROM part for larger volumes.

The advantages of ZTAT extend even further than this. There is no mask charge, reduced lead time and changes in software can be accommodated quickly just by "blowing" a new microcontroller.

And of course we haven't forgotten the period of development where you seem to go through a new version of software every day. Blowing a one time programmable device for each revision of your software could soon exceed your development budget!

For this situation Hitachi can offer uv-erasable devices (EPROM), which are ideally suited to initial development purposes.

H8/300 CPU Overview

When Hitachi's designers began work on the H8/300 CPU they threw away the rule book, and began with a blank sheet of paper. Therefore H8/300 was designed with no compatibility constraints, and it certainly shows! This enabled the designers to accommodate the requirements of HLL compilers, and to achieve outstanding performance.

CPU Model

The H8/300 CPU is based around what is known as a general register architecture. The CPU model is shown in *Figure 1*. Each register shown can be used for any purpose - hence the name (The H8/300 has either 16 general registers that are 8 bits wide, or 8 general registers that are 16 bits wide).

Each CPU register can be used as an accumulator, an index register, an address pointer or just high speed local storage. It can be said that H8/300 has 16 accumulators in comparison with previous architectures which only have 1 or possibly 2 !

From the point of view of a compiler writer, having so many available accumulators makes for a much more efficient compiler. Additionally all of these accumulators can be used as index registers or address pointers. The compiler can calculate the address of a variable in a

Figure 1 - H8 / 300 CPU Model

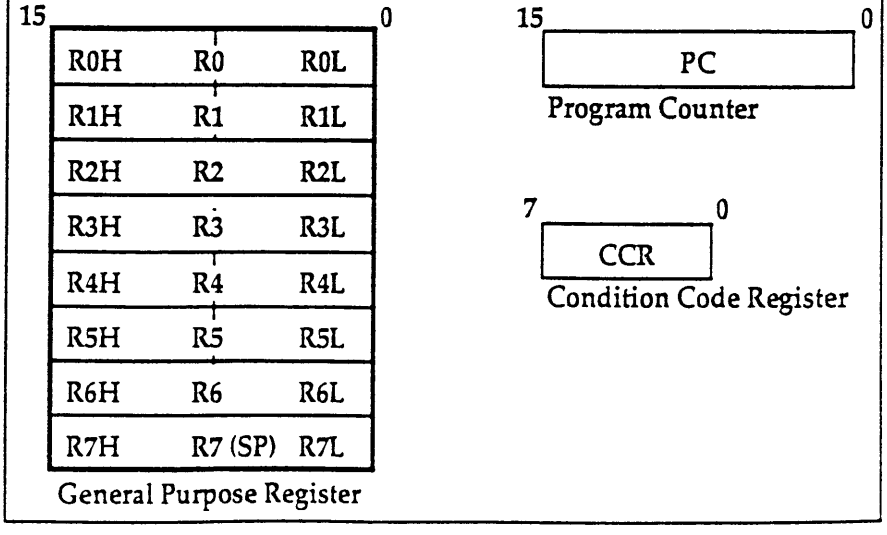


Figure 2 - H8/300 Addressing Modes

NO.	ADDRESSING MODE	MNEMONIC
1	Register direct	Rn
2	Register indirect	@ Rn
3	Register indirect with 16-bit displacement	@ (d : 16, Rn)
4	Register indirect with pre-decrement or post increment	@ _Rn @ Rn+
5	Immediate (3-, 8-, or 16-bit data)	#xx : 3, #xx : 8, #xx : 16
6	Absolute address (8 or 16 bits)	@ aa : 8, @ aa : 16
7	PC-relative (8-bit displacement)	@ (d : 8, PC)
8	Memory indirect	@@ aa : 8

register, and in the next instruction use that same register as an address pointer to access the variable. This significantly reduces the amount of code that is required to perform such an operation compared to a traditional architecture which has separate accumulators and index registers.

Internal Buses

Although the H8/300 series is based on an 8 bit CPU, it can be seen that its registers and memory can be accessed as 16 bit locations. To further increase the CPU performance the internal data bus of the H8/300 series are all 16 bits wide, giving fast, 2 state word access to internal memory.

Addressing Modes

Another way a CPU architecture can support the compiler writer is by providing powerful, flexible addressing modes. A CPU which only has rudimentary addressing modes makes a compiler inefficient in accessing variables, thereby increasing both execution time and code size.

To ease the operation of the compiler, the H8/300 CPU provides 8 addressing modes which are shown in *Figure 2*. These modes range from simple direct and indirect operations through to indirect plus displacement.

Supporting array and stack data types H8/300 has indirect addressing with either a post-increment or pre-decrement. These modes support both byte and word data (±1 or 2).

For accessing local variables (which are kept on the stack) the addressing mode indirect plus displacement is used. The operation of this mode is shown in *Figure 3*. It allows stack based variables to be accessed in one instruction. This signficantly reduces the code size and exection time when running a compiled language.

H8/300 has 2 direct addressing modes, using either an 8 or 16 bit absolute address. The 16 bit mode has the capability to access any location in the devices 64K address space using a single instruction. By its nature any operation which uses

Figure 3 - Effective Address Calculation

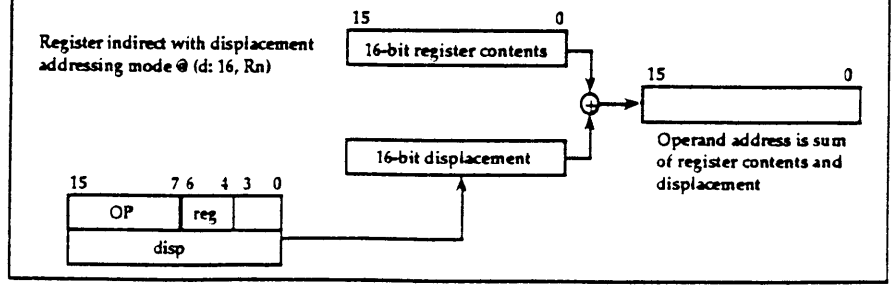


Figure 4 - H8 / 300 Instruction Set

FUNCTION	INSTRUCTIONS	TYPES
Data transfer	MOV, MOVFPE, MOVTPE, POP, PUSH	3
Arithmetic operations	ADD, SUB, ADDX, SUBX, INC, DEC, ADDS, SUBS, DAA, DAS, MULXU, DIVXU, CMP, NEG	14
Logic operations	AND, OR, XOR, NOT	4
Shift	SHAL, SHAR, SHLL, SHLR, ROTL, ROTR, ROTXL, ROTXR	8
Bit manipulation	BSET, BCLR, BNOT, BTST, BAND, BIAND, BOR BIOR, BXOR, BIXOR, BLD, BILD, BST, BIST	14
Branch	Bcc, JMP, BSR, JSR, RTS	5
System control	RTE, SLEEP, LDC, STC, ANDC, ORC, XORC, NOP	8
Block data transfer	EEPMOV	1
	Total :	57

this mode must have a 16 bit address field.

For more compact access to variables the 8-bit direct mode has been provided. This will access any byte within the top 256 byte page of the H8/300s memory when this mode is used. (It does this by making the most significant byte on the address bus H'FF).

The 8-bit mode is particulary useful when accessing I/O registers. The H8/300 microcontrollers have this area located within the top 256 byte page, which can therefore be accessed by these shortened addresses.

Instruction Set

H8/300 has a streamlined instruction set, well suited to the needs of a HLL, and also embedded applications in general.

The instruction set comprises 57 instructions, which are either 2 or 4 bytes long.

All of the most frequently used instructions execute in just 2 states. That is 200 nS if the device is clocked at 10Mhz. Indeed even complex instructions such as an 8 * 8 multiplication only take 14 states, giving the H8/300 unrivalled 8 bit CPU performance.

Figure 4 shows the complete H8/300 instruction set, and from this you can see the full extent of the H8/300s instruction power.

Bit Processing

In microcontroller applications it is often necessary to manipulate data on a bit by bit basis. A good example would be where an I/O pin needs to be set, to switch on a lamp or other device.

The H8/300 CPU has 14 separate bit processing instructions, which allow the programmer to manipulate bit data very easily. Also it is possible to perform boolean algebra on bit data. The H8/300 CPU provides a bit accumulator using its carry flag and a full set of boolean operations (AND, OR, XOR and NOT) for bit data.

Another feature of the bit processing instructions of the H8/300 CPU is its ability to access bits indirectly, using the value from a general purpose register as a bit pointer. This mechanism is shown in *Figure 5* and is useful for scanning a byte for set or cleared bits.

H8/300 Exception Processing

In any microcontroller environment, it is important to handle asynchronous events as efficiently as possible. These events may come from a number of different sources, such as motion

Figure 5 - Indirect Bit Access

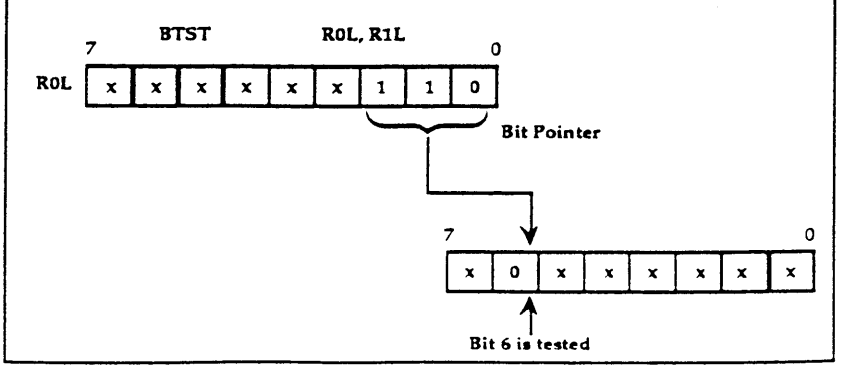
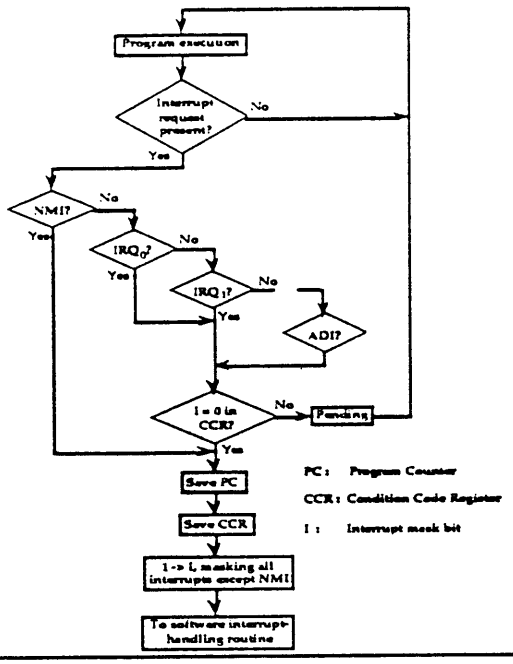


Figure 6 - Interrupt Handling Sequence



sensors, frequency sources or the microcontroller's internal peripherals.

These events are best handled using interrupts, and a powerful interrupt structure allows a number of events to be handled easily.

H8/300 has a novel approach to interrupts, in which an interrupt vector is assigned to each separate event that the microcontroller can process.

So instead of the most commonly applied method in which one interrupt vector is provided for each separate peripheral, H8/300 microcontrollers have several vectors per peripheral. This speeds up the execution of any interrupt service routine, as there is no need for the service routine to check the peripherals status register to determine where the interrupt came from.

For example, the serial port of the H8/330 microcontroller can generate three separate interrupts, rx done, tx done and rx error. To handle these events three interrupts are provided.

Once an interrupt is signalled to the interrupt controller it follows the flow shown in *Figure 6*.

The controller's current interrupt status is checked first, and then, if interrupts are currently enabled (I = 0), the appropriate interrupt vector is placed into the program counter (after saving the PC and CCR registers) and execution begins from this address, with all interrupts (except NMI) disabled.

To disable (mask) individual interrupts each peripheral has its own control register which allows each interrupt to be enabled separately. This mechanism is described in the block diagram of the interrupt controller shown in *Figure 7*.

Interrupt Response Time

As well as being able to handle multiple interrupts it is also important to handle them quickly. The interrupt response time of the H8/300 is very fast. A single chip system can start execution of an interrupt service routine in a minimum of 17 states (1.7µS at 10MHz) from the event occurrence.

Interrupt response timings are shown in *Figure 8*.

Power Down Modes

When running, the H8/300

Figure 7 - Interrupt Controller Block Diagram

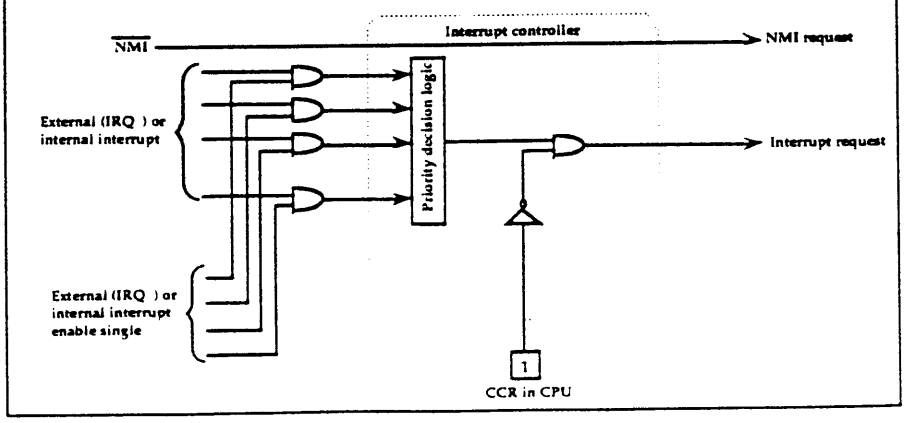


Figure 8 - H8 / 300 Interrupt Response Time

No.	Reason for wait	On-chip memory	External memory
1	Interrupt priority decision	2	2
2	Wait for completion of current instruction	1 to 13	5 to 17
3	Save PC and CCR	4	12
4	Fetch vector	2	6
5	Fetch instruction	4	12
6	Internal processing	4	4
	TOTAL	17 to 29 *	41 to 53 *

* Clock cycles

microcontroller family displays a very low power dissipation due to its CMOS structure. However, at certain times even this is not low enough, especially when a battery backup is required.

To provide for such eventualities, H8/300 devices have three low power modes: sleep, software standby and hardware standby. Each offers different advantages for various application needs.

Sleep Mode

In this mode the device switches off the clock to the CPU, but all of the on-board peripherals remain active. Sleep mode is entered via the "sleep" instruction, and the CPU can exit this mode whenever an enabled interrupt occurs.

A useful application for this mode is to reduce the average power consumed by a system, using a timer to "wake" the CPU after a period of slumber.

Once woken, the CPU can process for a period of time and then after loading the timer again the sleep instruction can be executed, lowering the power consumption.

When sleep mode is entered the devices current consumption is reduced by approximately one third.

Software Standby Mode

Again, this mode can be entered using the sleep instruction, but in this case the on chip oscillator is stopped completely, putting the device into standby.

Standby current is very low with a maximum value of 5 µA. This is coupled with a data retention voltage of 2V, allowing the microcontrollers internal ram contents to be maintained using just two 1.5V battery cells, or a large "reservoir" capacitor.

During software standby mode, the microcontroller maintains the value of the I/O ports, so outputs can be set to the values the system requires during power down, with the knowledge that they will remain stable during software standby mode.

To exit from this mode, any external interrupt can be used, and a specialised timer circuit is provided to ensure that the on chip oscillator has started and is stable before execution of the interrupt service routine begins.

Hardware Standby Mode

As with software standby, hardware standby enters a state where the on chip oscillator has been switched off, and current consumption is minimal.

However, hardware standby is entered depending on the state of an external pin. This mode is ideal for putting the device into standby from some form of external signal, for example a power fail warning.

Unlike software standby this mode has the effect of tri-stating all the I/O ports, and so is an effective method of taking the microcontroller off the bus.

The sole method of leaving hardware standby is by first resetting the device ($\overline{RES}$ = 0) and then taking the standby pin high. When reset is taken high also (after the clock oscillator has stabilised) the device begins execution from the reset vector.

H8/300 Peripherals

A crucial part of any microcontroller's feature set is its on chip peripherals. No microcontroller can boast

that it can provide a "single chip solution" if it needs additional devices to complete a system's requirements.

But, just integrating a high density of peripherals onto the die of a microcontroller is not enough, functionality within these peripherals is also vital.

For example, a timer designed with no mechanisms to off-load the CPU's involvement when producing timed outputs, or measuring in-coming signals can seriously limit the performance of a system.

Within the H8/300 family, a varied range of peripheral functions are provided, including timers, A/D converters and serial interfaces and some more unusual features, such as dual port RAM. These features combine to make H8/300 a true single chip solution.

16 Bit Free Running Timers (FRT)

Three types of 16 bit FRT are specified within the H8/300 family. Each variety differs according to the number and type of inputs and outputs that are available to the FRT.

Figure 9 shows the FRT which is included in the H8/330 microcontroller.

All the different FRT's are based on a 16 bit count register (FRC). This can be incremented either from some derivatives of the system clock (timer mode) or from an external pin (event counter mode). The maximum resolution of a FRT in timer mode is 200nS (when the system clock (ø) = 10 Mhz).

Each type of FRT has the ability to produce complex output events to a very high degree of accuracy, using what are known as output comparators.

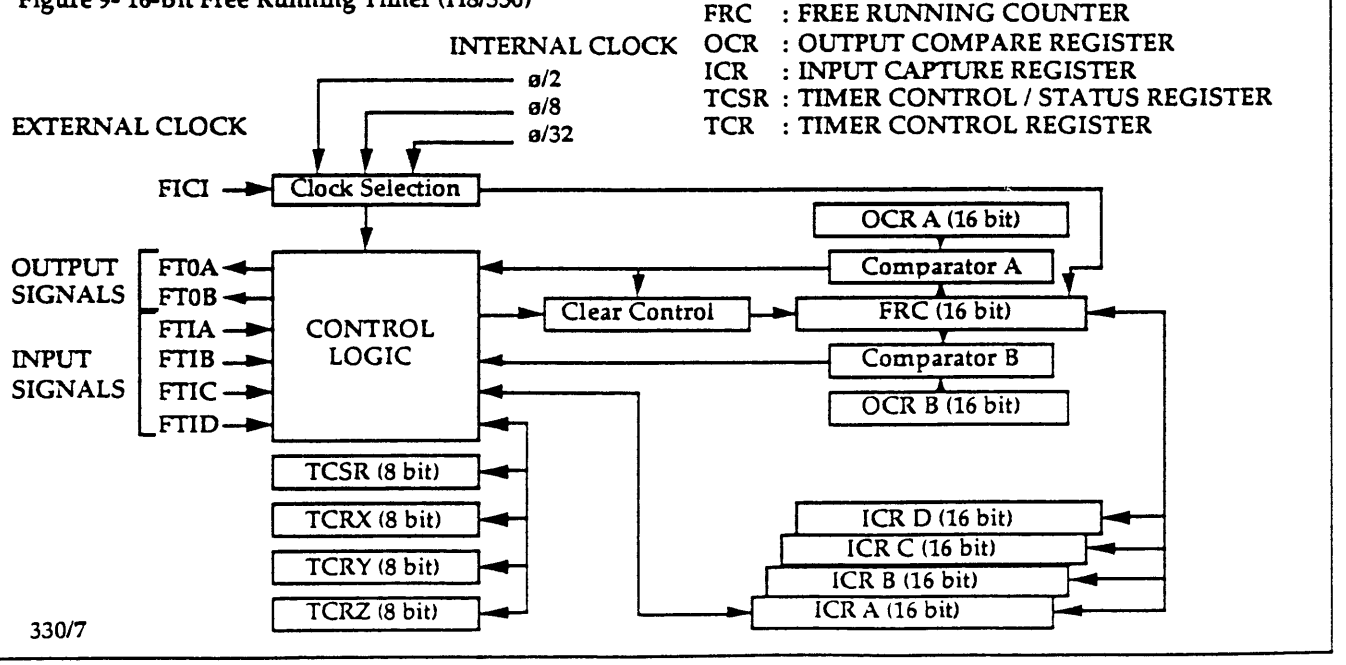
These comparators are used to compare the 16 bit value in a timer register with the value in the FRC for every timer increment. When the two values are equal a compare match is said to have occurred, and an event will be produced.

This event can be anything from simply setting a status flag in the timer's control register, to causing a transition on a timer output pin. Another useful event allows the FRC to be cleared when a match occurs. These events can also be programmed to cause an interrupt to the CPU.

Referring to the diagram in Figure 9, it can be seen that the H8/330 microcontroller has two of these comparators (OCRA and OCRB), and two corresponding output pins.

Apart from producing timed outputs, the FRTs in the H8/300 family can also be used to measure the occurrence of edges on

Figure 9- 16-Bit Free Running Timer (H8/330)



incoming signals, using what is known as input capture registers.

An input capture register is a 16 bit register, which can "capture" the value from the FRC when an external event provides a pre-programmed stimulus. This stimulus can be either a positive or negative going edge on the capture input pin.

When this stimulus is detected on a capture input pin the value currently contained within the FRC is transferred into the appropriate capture register, so giving a 16 bit " timestamp " for an input transition.

By using a capture input to measure timing values, a very accurate measurement can be made, as the time is captured without any software intervention and no allowance need be made for interrupt latentcy.

An input capture can also be programmed to produce an interrupt when the programmed event occurs.

As a FRT has a number of separate functions, several interrupts are provided from each channel of FRT. For example the H8/330's FRT can produce 7 separate interrupts. These interrupts service 2 output comparators, 4 input captures and FRC overflow.

8 Bit Timer

This timer type offers similar functionality to the 16 bit FRT, but with an 8 bit resolution. This timer is capable of producing a high frequency output waveform with no CPU intervention.

As the block diagram in *Figure 10* shows this peripheral can be used as an event counter, or a timer. If timer mode is selected several derivatives of the internal clock are available.

To produce timed outputs this timer has two output compare registers (TCORA and TCORB), with a single output shared between the two registers. It is this sharing which enables the device to produce high frequency outputs.

This is done by configuring the comparators so that a match on TCORA will set the timer output pin, and a match on TCORB will clear the timer pin, and also clear the timer. This in effect will produce an output signal with a period equal to the value in TCORB and a mark equal to the value in TCORA.

The 8 bit timer's count value can also be cleared by an external signal at the TMRI pin.

The 8-bit timers have the capability to produce interrupts, from either of the comparator registers or when the timer overflows.

PWM Timers

Pulse Width Modulation, (PWM), is a technique

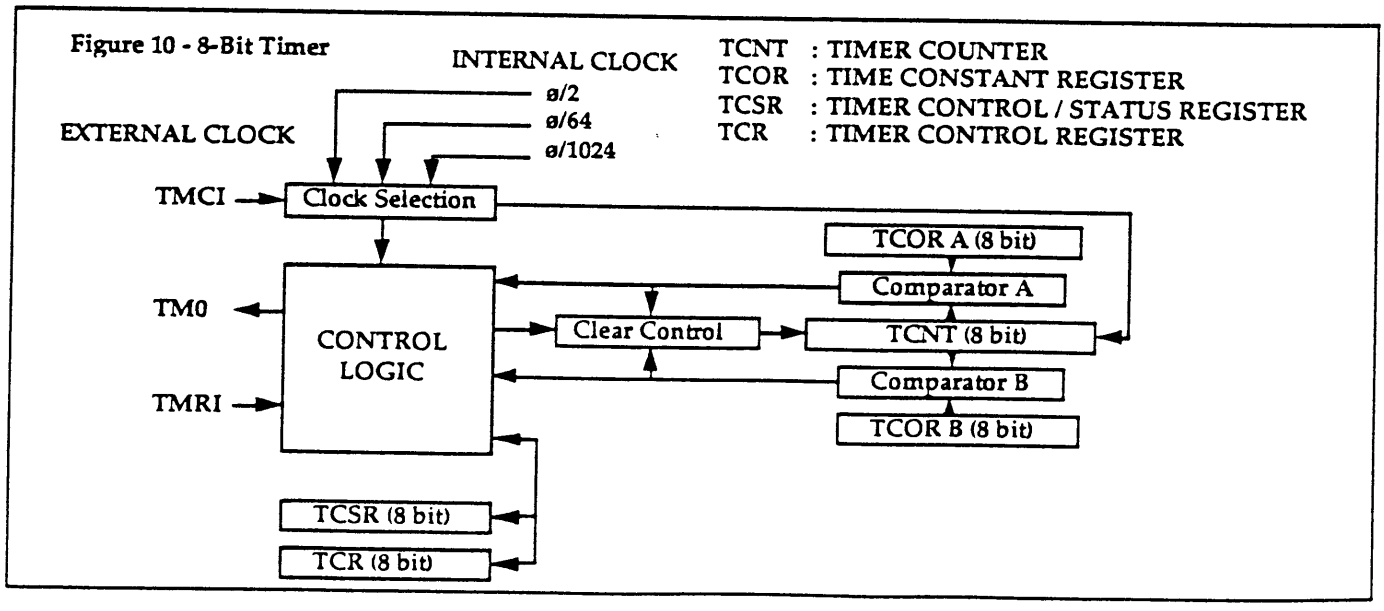
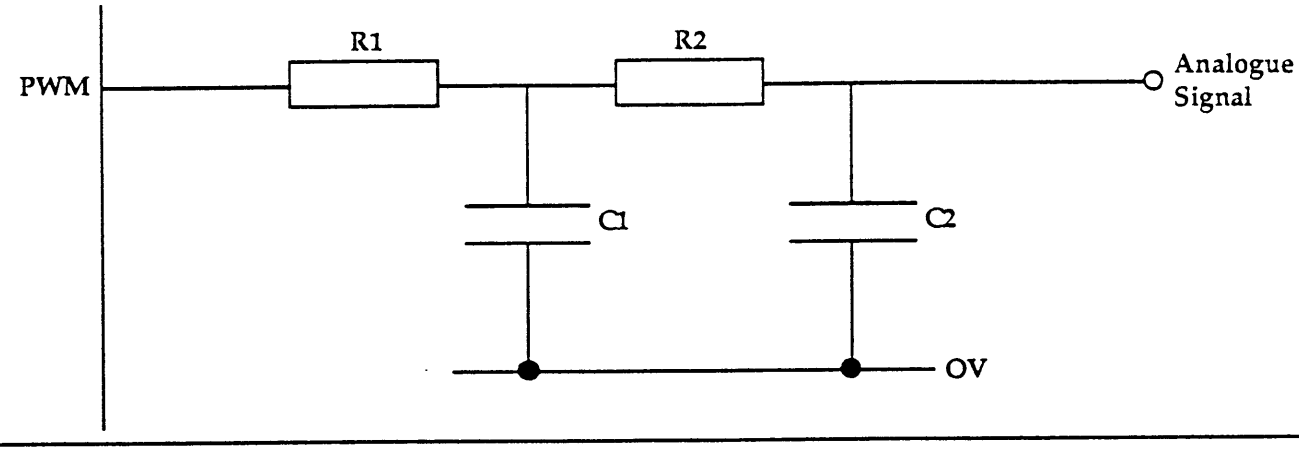


Figure 10 - 8-Bit Timer

Figure 11 - Using the PWM Output as a D to A



which is commonly used to drive mechanical devices such as solenoids or motors.

The characteristics of PWM is a waveform with a fixed period and a variable duty cycle. By changing the duty cycle (or mark to space ratio) the average voltage to the device being driven changes.

As well as driving mechanical devices a PWM output can also be used as an analogue output. This can be achieved by feeding the output into a low pass filter (integrator) as shown in *Figure 11*.

H8/300 family members can have PWM timers with resolutions of 8-bits (H8/330), or 14-bits, (H8/350). The 8-bit timers can also be used as extra PWM outputs if necessary.

Servo Control Timers

To control servo motors in a closed loop system some very specialised timers can be used to monitor the motor's current position.

The H8/350 includes up/down counters ,which can be used to gather position information from a quadrature encoder, and a single channel of 14-bit timer, which can produce a precision drive signal to a servo motor.

Other specialised timers on the H8/350 include a 19 bit free running timer with the same sort of functionality as the previously described 16-bit FRT.

Serial Communications

This form of communications is very often used in a microcontroller environment, to talk to other microcontrollers or to peripheral devices. The H8/300 family includes very powerful peripherals to facilitate this type of interface.

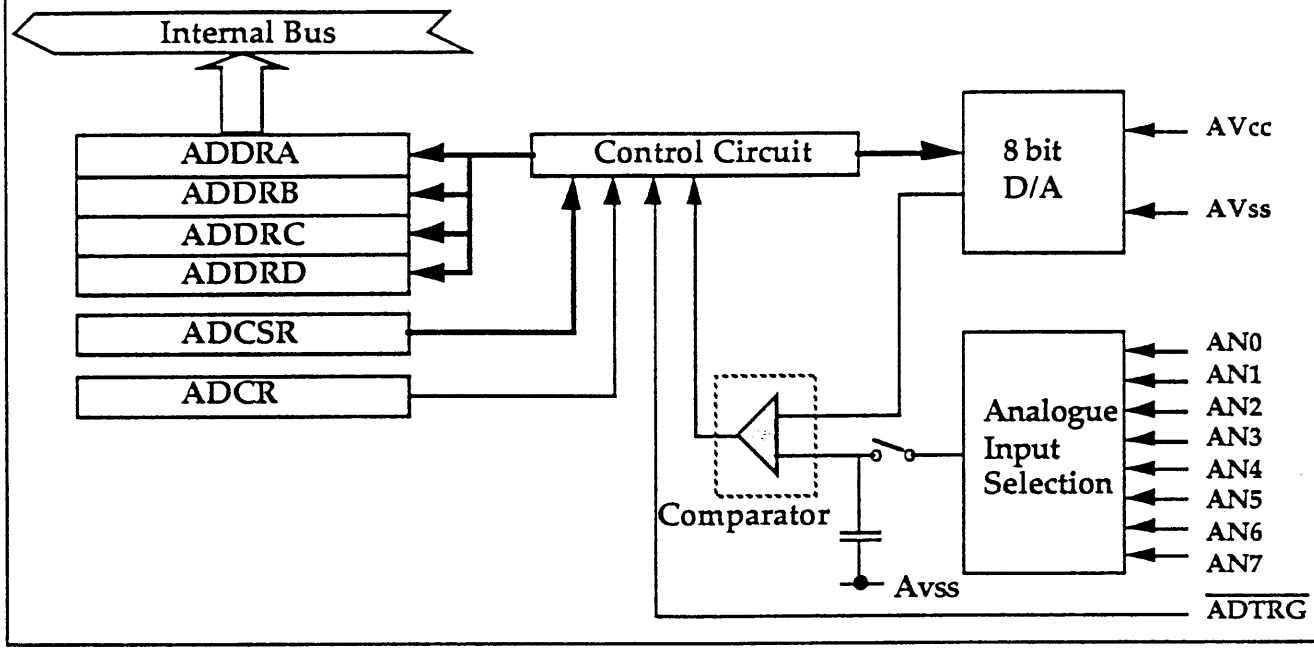
These serial communications interfaces (SCI) can perform either synchronous or asynchronous communications, and data rates of up to 2.5 MBit/Sec are supported (sync mode).

Each channel of SCI has its own baud rate generation timer, so the use of serial communications does not impact the number of timers available in the microcontroller. By programming a control register and loading the baud rate register (BRR) a wide range of data rates can be achieved from one source of microcontroller clock.

In asynchronous mode several data formats are catered for, including the provision of odd or even parity.

To indicate various conditions occurring within the SCI a status register is provided which contains flags for receive buffer full, transmit buffer full or receive error. Each of these flags has an interrupt associated with it to indicate the occurrance of such a condition.

Figure 12 - A/D Converter (H8/330)



Each of the SCI data registers (TDR and RDR) is double buffered so it is possible to transmit and receive data in a " back to back " manner.

Interface Peripherals

As well as serial communications the H8/300 family also has the facility to provide a parallel interface to other devices. In the case of H8/330 using a dual port RAM, and via a handshaking interfacing on the H8/32X series.

The dual port ram of the H8/330 comprises a set of 15 registers which can be accessed from the internal CPU, or from another device via an SRAM-like interface.

Using this facility, the H8/330 can be used as an intelligent peripheral in a large system.

A/D Converter

Some variants within the H8/300 family have Analogue / Digital converters which allow the microcontroller to monitor voltage or other "real world" values and therefore control such variables.

The H8/330 has an 8 channel A/D converter, which is shown in *Figure 12*. The H8/350 has 16 channels of A/D in keeping with its role as a servo microcontroller.

These A/D converters have resolutions of 8 bits, and they achieve their specified number of channels using an analogue multiplexer (8:1 or 16:1). After the multiplexer a sample and hold circuit is included. Once a conversion has begun its result will not be affected by any changes on the external input.

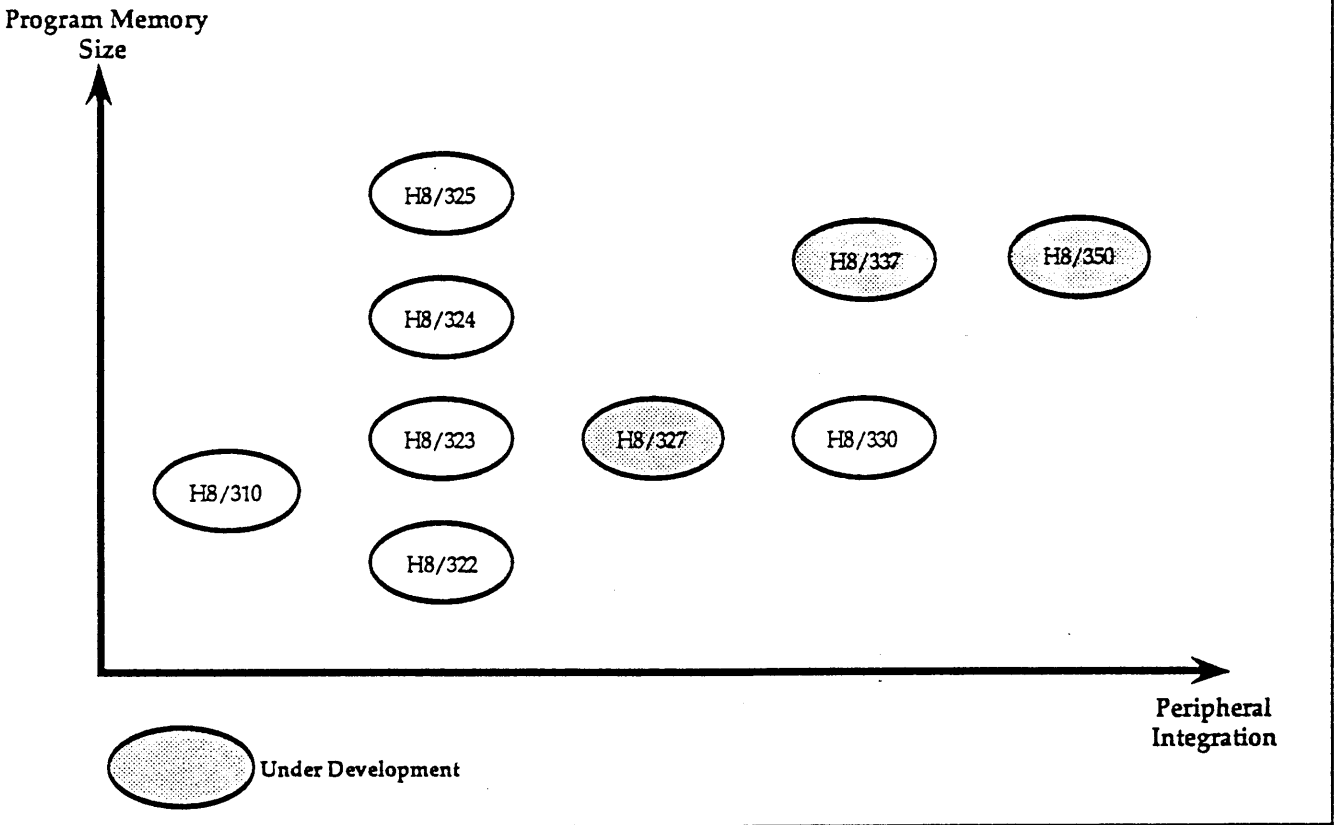
Conversion is performed very quickly, taking either 12.2µS or 24.2µS (programmable) if the system clock frequency is 10MHz.

To remove responsibility for servicing the A/D converter from the CPU a scan mode is provided. When scan mode is the selected a number of channels (1-4) will be converted sequentially. This operation is continuous, and the A/D converter has 4 data registers (ADDRA - ADDRD) to store each result.

Another useful feature of the H8/300 family's A/D converter is its ability to begin a conversion when it receives an external stimulus, using a trigger pin (ADTRG).

THE H8 / 300 FAMILY OF MICROCONTROLLERS

H8 / 300 FAMILY MIGRATION



H8 / 300 Part Numbers

HD64 — 8 / 7 / 3 / 1 — XXX8 — CP / S / CG / C / F — 6 / 8 / 10

8 = EEPROM	CP = PLCC 84	6 = 6MHz System Clock
7 = ZTAT Memory	CG = LCC 84	8 = 8MHz System Clock
3 = Mask ROM	F = QFP 80A	10 = 10MHz System Clock
1 = ROM Less	C = ceramic dil (shrink)	
	S = plastic dil (shrink)	

For Example :

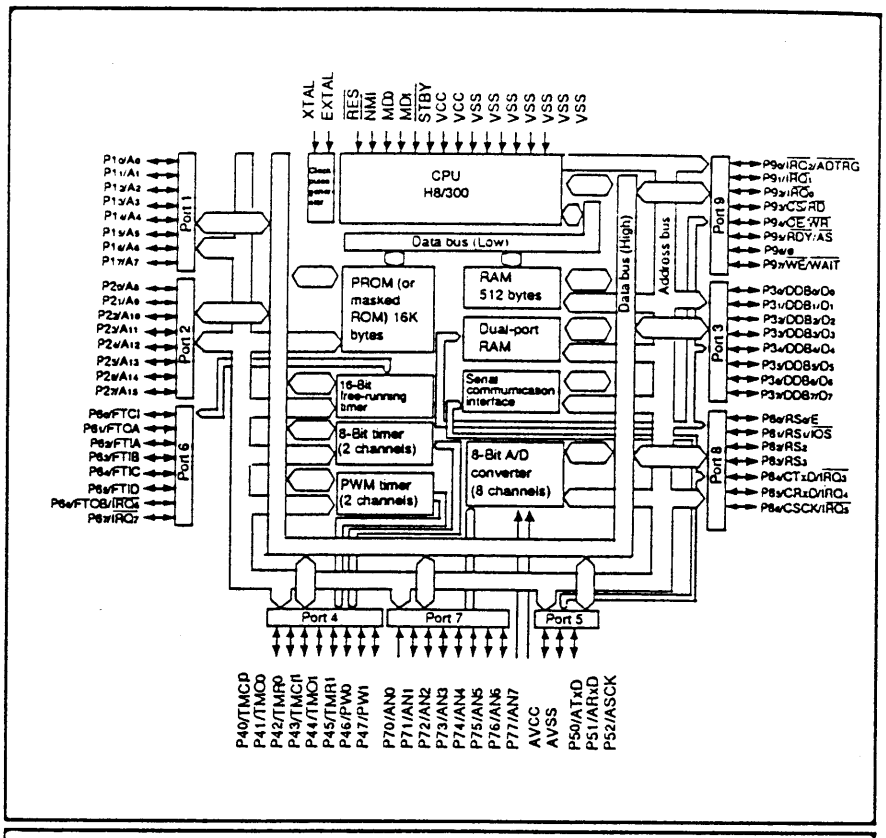
HD6473308F10 = H8 / 330, QFP - 80A, 10MHz

H8 / 330

This member of the H8/300 family is an ideal general purpose microcontroller, due to its high level of peripheral integration and large pin count.

Its features include :-

16 KBytes ROM/PROM/EPROM
512 Bytes RAM
16 Bit Counter / Timer - 1 channel
8 Bit Counter Timer - 2 channels
8 Bit PWM Timer - 2 channels
8 Bit A/D Converter - 8 channels
Async/Sync Serial Port
58 Input / Output Lines
8 Input Only Lines
15 Bytes Dual-Port Ram



H8/32X Family

This family of devices all share an identical pin out and peripheral set. The differentiation between the 4 family members is made by the internal memory size, from 32 KBytes ROM/PROM and 1 KBytes of Ram to 8K Bytes of ROM/EPROM and 256 Bytes of RAM.

The H8/32X Family Features are :

16 Bit Counter/Timer - 1 channel
8 bit Counter/Timer - 2 channels
Async/Sync Serial Port- 2 channels
Parallel Handshaking Interface

See *Table 1* For Memory Sizes

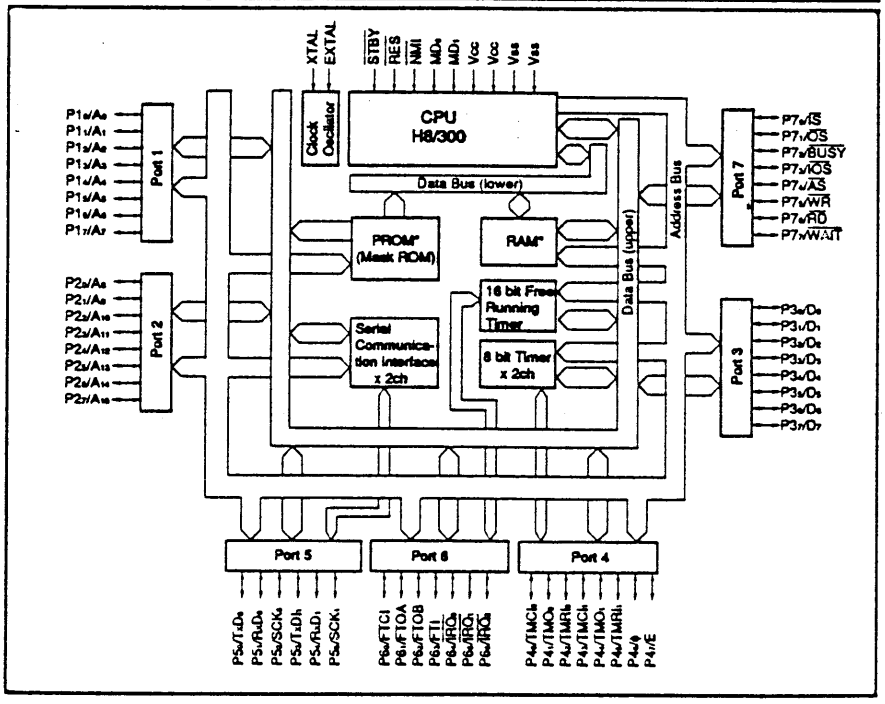


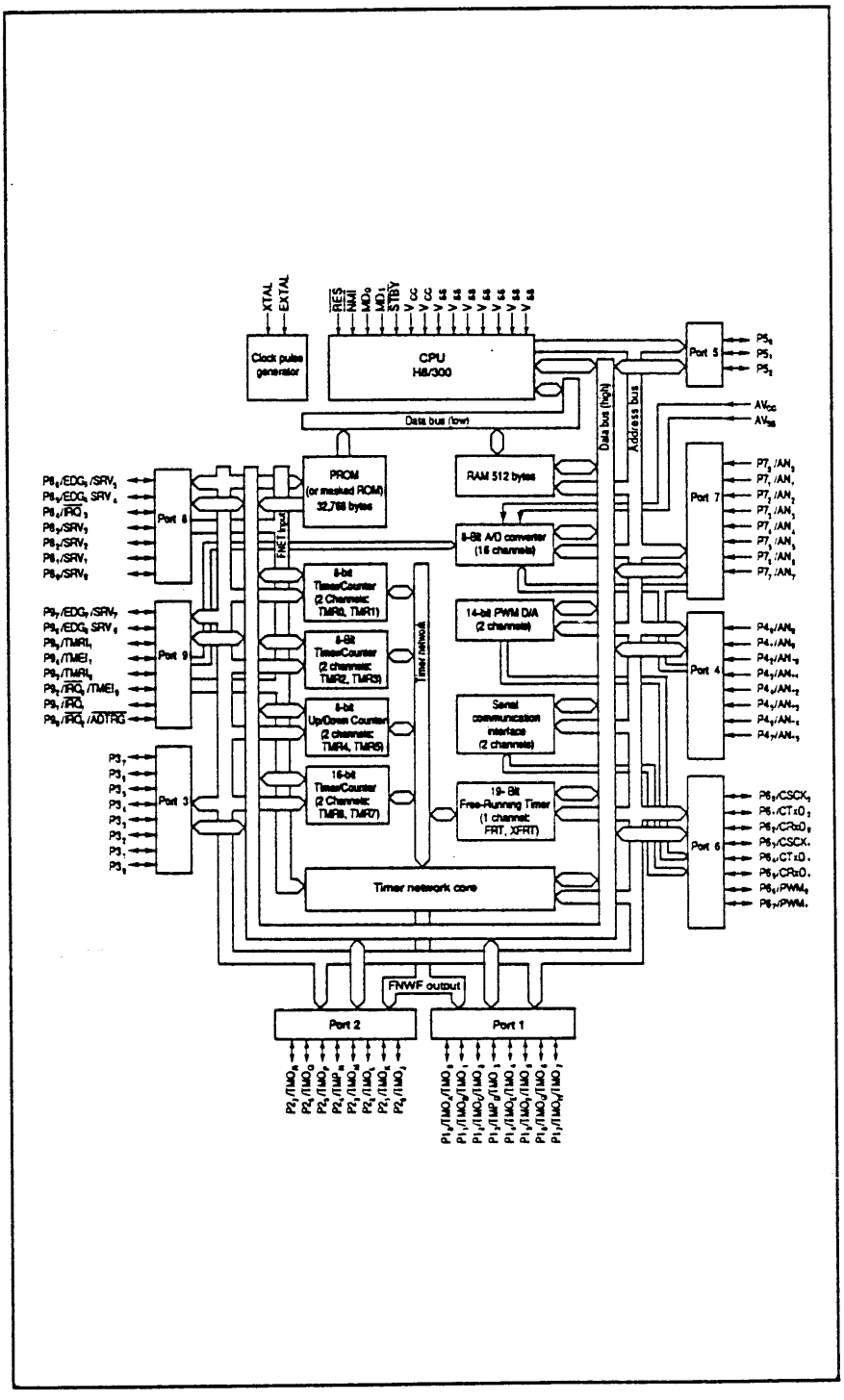
Table 1 : H8 / 32X Memory Variants

DEVICE	ROM	PROM	EPROM	RAM
H8 / 325	32K	32K	32K	1K
H8 /324	24K	-	-	1K
H8 / 323	16K	16K	-	512B
H8 / 322	8K	8K	-	256B

H8/350

When an application needs a variety of timer functions the H8/350 really comes into its own. Designed as a servo microcontroller this device boast no less than 12 seperate timer channels, each with their own special features.

To make these timers even more useful a network core allows inputs and outputs of the timers to be connected, so the range of a timer channel can easily be extended.



The H8/350 Features :

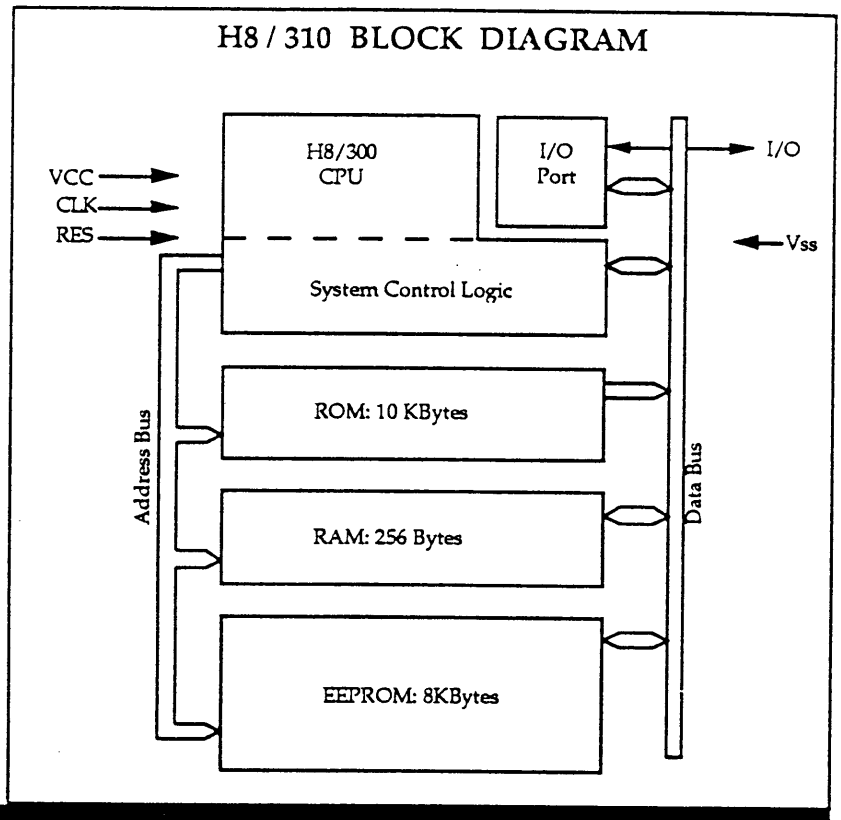
32 KBytes ROM, PROM or EPROM
512 Bytes of RAM
19-Bit Counter/Timer-1 channel
16-Bit Counter/Timer-2 channels
8-Bit Counter/Timer-4 channels
8-Bit Up/Down Counter-2 channels
14-Bit PWM Timer-2 channels
Async/Sync Serial Port-1 channel
Sync Serial Port-1 channel
8-Bit A/D Converter-16 channels
50 Input/Output Lines
16 Input Only Lines

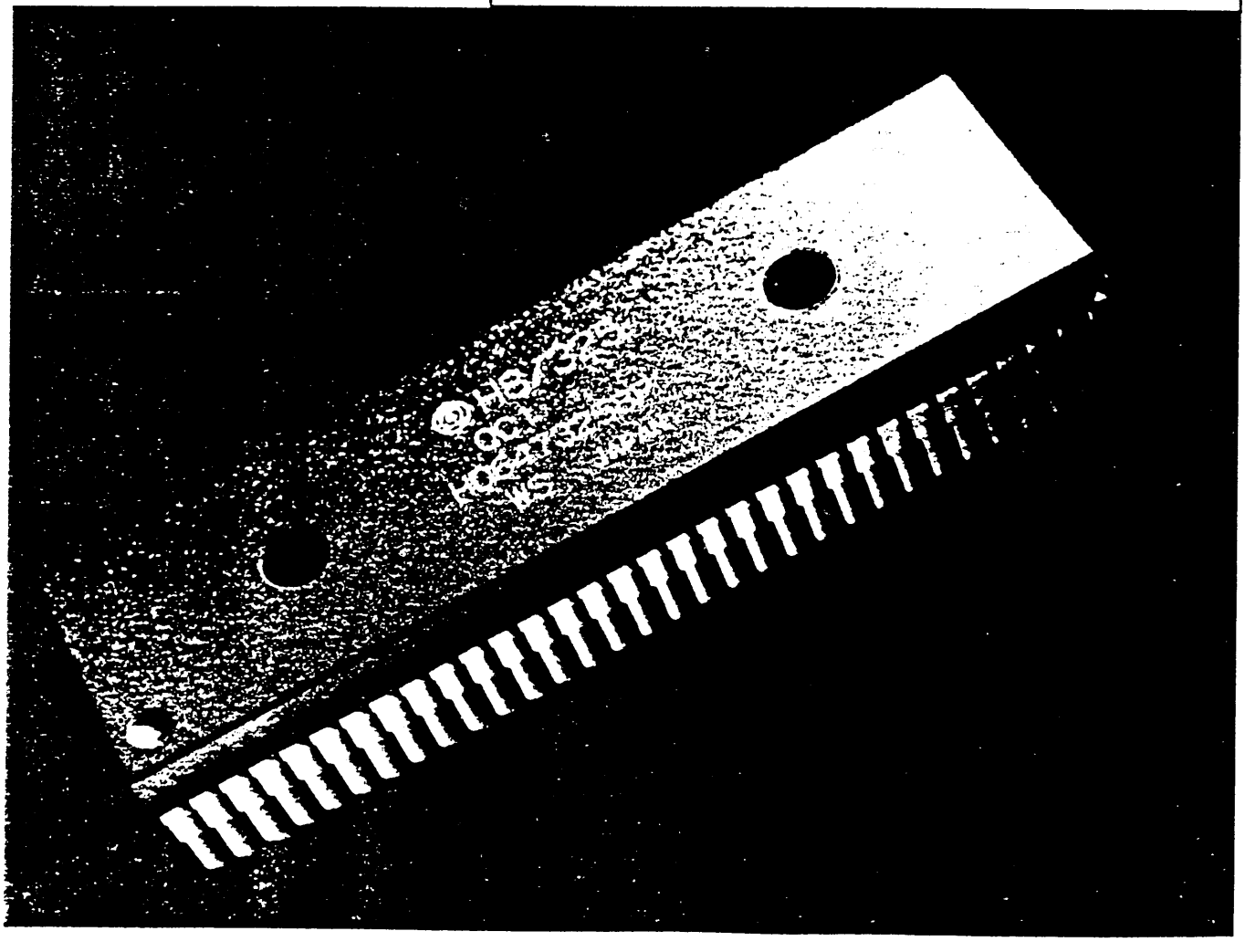
H8/310

This device is aimed at smart card applications, especially with its large area of on board EEPROM. For this reason it is available as either a die, or in chip on board packaging (COB). More recently this device has also been made available in a 10 pin small outline package.

The H8/310 features :-

10 KBytes ROM
256 Bytes Ram
8 KBytes EEPROM
1 I / O Line





H8 / 300 - A New Look at Packaging

With the progressive march towards smaller end equipment, the pressure on the allocation of space inside the equipment is increasing very rapidly.

It is no longer enough to offer a highly integrated microcontroller, it also has to come in the smallest package possible.

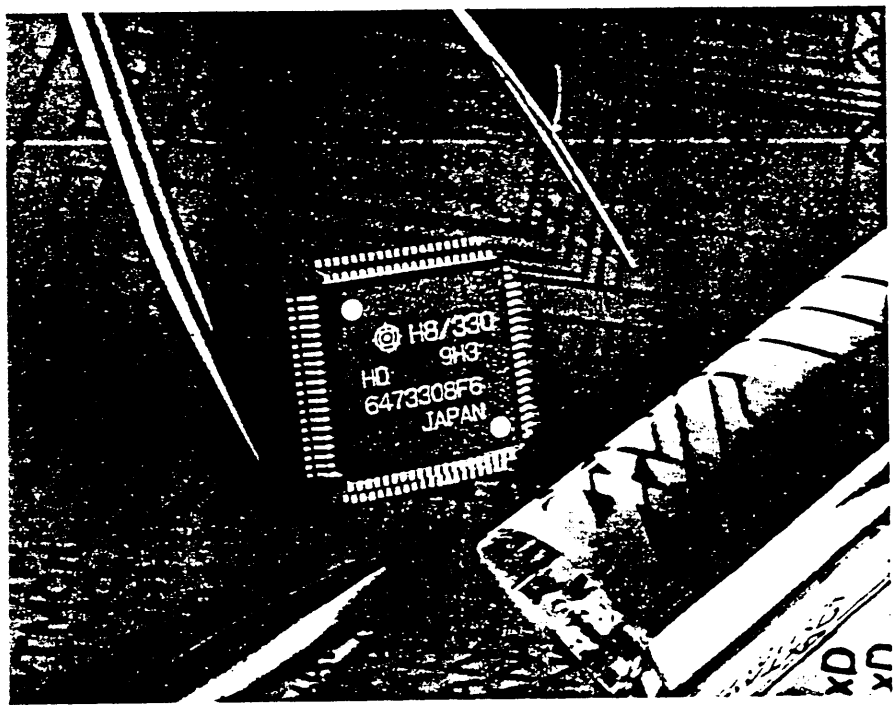
The H8/300 family lets you keep one jump ahead of this trend, offering unbeatable integration in a wide range of package options.

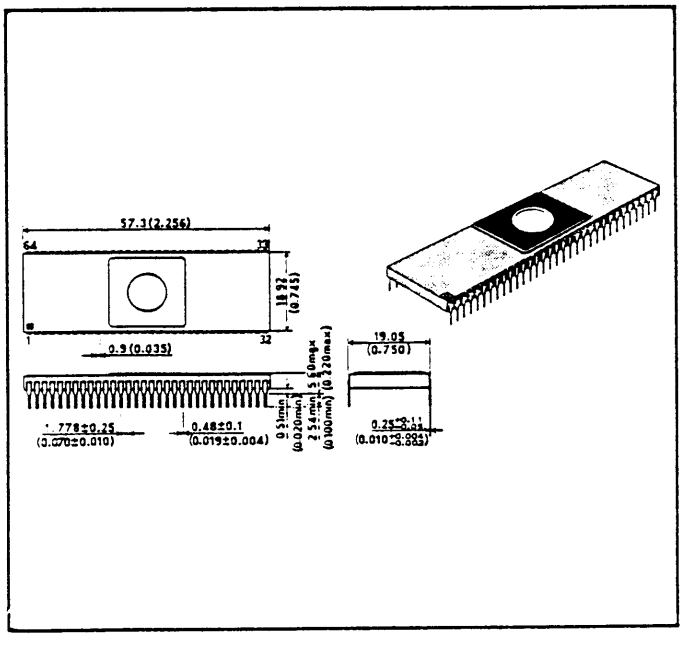
Witness the H8/325 with 32K of OTP PROM and 1K of RAM, all contained within a tiny flat package, measuring just 17.2mm by 17.2mm across it's gull wings.

Aside from these extremely small packages, the H8/300 family is also available in more traditional packaging, including PLCC and DIL.

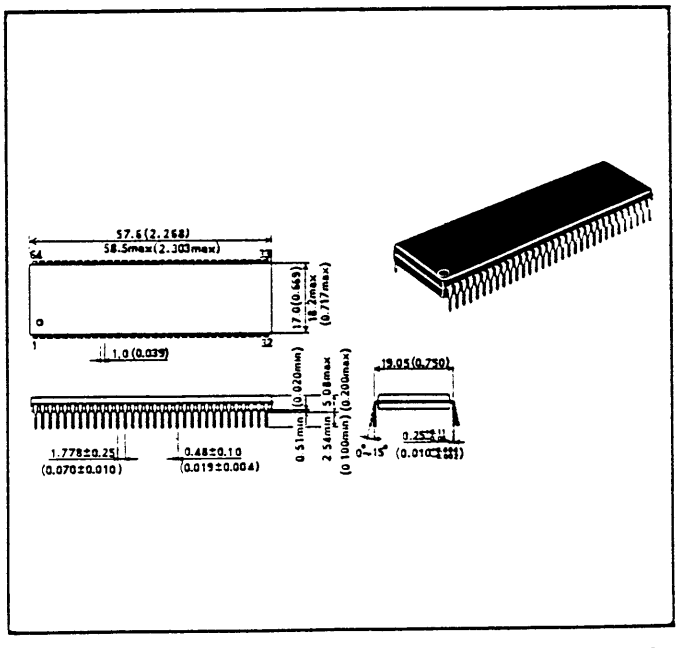
H8 / 300 Package Options

DEVICE \ PACKAGE	SOP-10	DC-64S	DP-64S	QFP-64A	PLCC-84	QFP-80A	LCC-84	DIE
H8 / 310	•							•
H8 / 322 FAMILY			•	•				
H8 / 323			•	•				
H8 / 324			•	•				
H8 / 325		•	•	•				
H8 / 330					•	•	•	
H8 / 350					•	•	•	

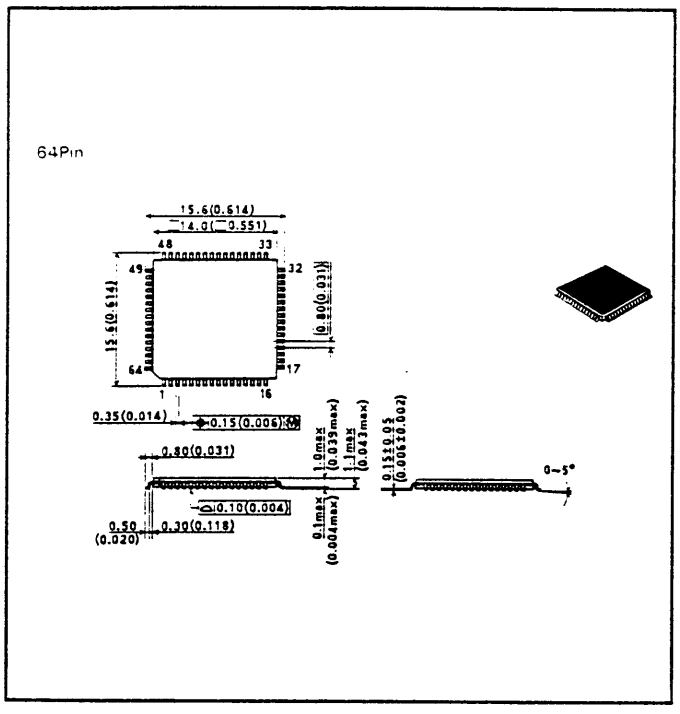




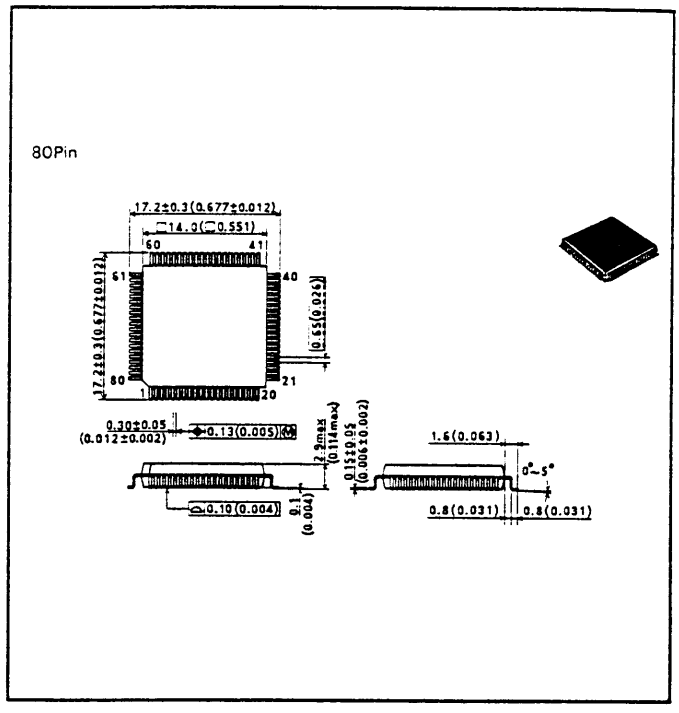
DC - 645



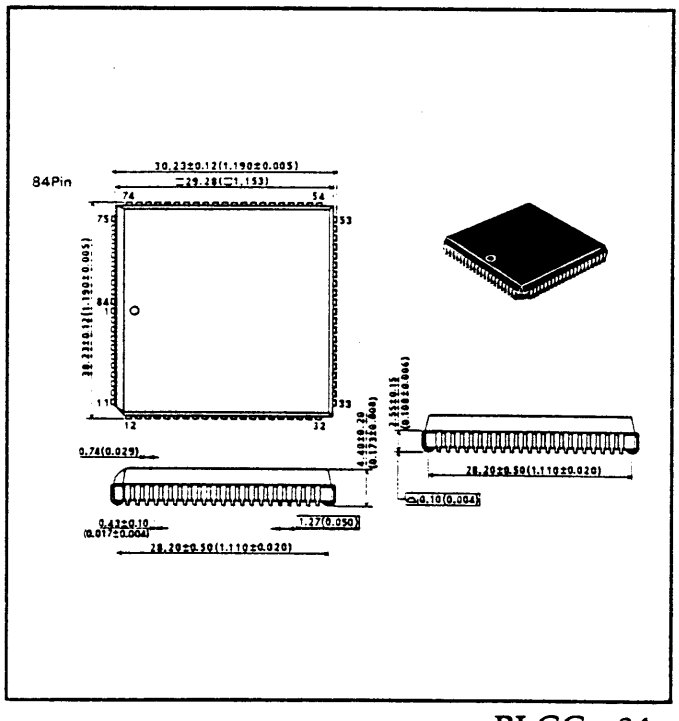
DP - 645



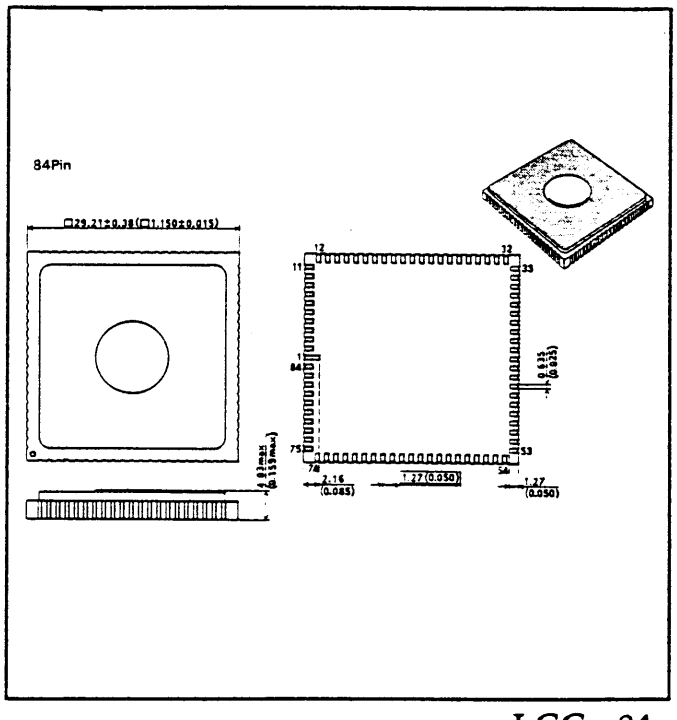
QFP - 64A



QFP - 80A

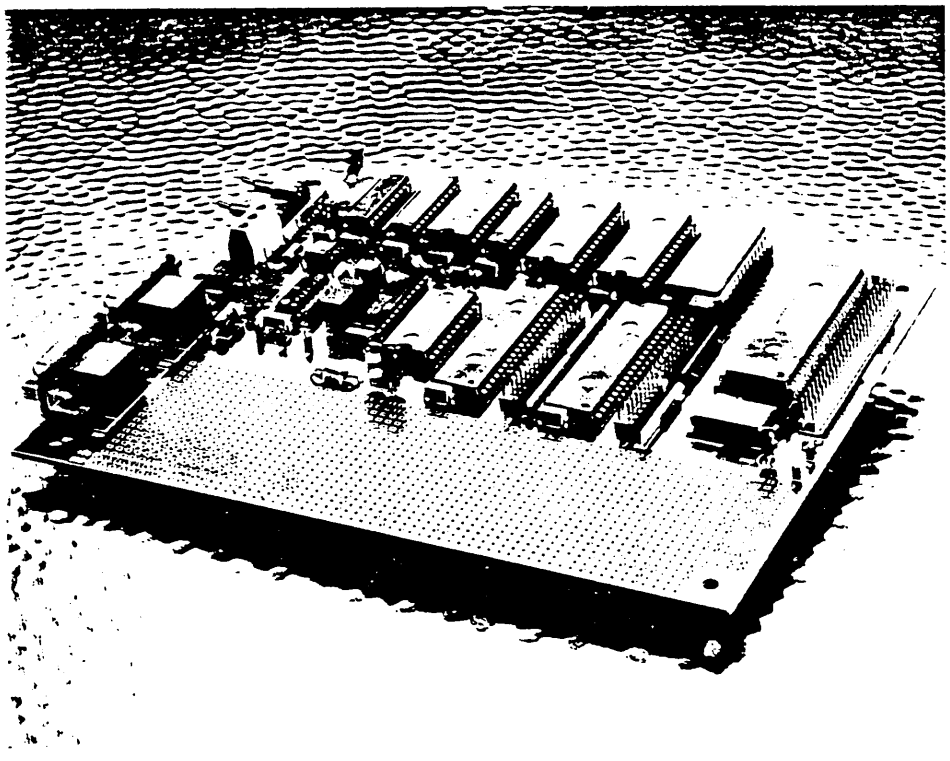


PLCC - 84



LCC - 84

H8/300 Support Tools



To achieve a development schedule using a microcontroller it is important that the tools used for developing the hardware, and often more importantly, the embedded software are of the right quality and functionality. For example if a high level language compiler is used then the quality of the code produced can have a significant impact on the development cycle.

In introducing the H8/300 series Hitachi has recognised the importance of such development tools, and a great deal of effort has been put into ensuring that the right mix of products is available, to suit any engineers requirements.

This range includes full ANSI C Compilers, evaluation boards and full in circuit emulation systems. The host supported computers include standard 640K personal computers running DOS and VAX/VMS machines.

Ansi C Compiler

The H8/300 compiler has been designed for ease of use and efficiency, both during development and in the resulting code. This ultra-fast memory-based software complies totally to the full ANSI C proposal, while still generating efficient optimised and, when requested, PROMable code in one single pass.

Data Representation

The H8/300 C Compiler supports all ANSI C basic elements. 'Float', 'double' and 'long double' objects are represented in the IEEE format. The table below shows the size in bytes for the different objects.

char	short	int	long	float	double	long double	pointer / address
1	2	2	4	4	4/8	4/8	2

The basic floating point operators (+ - * /) support a numeric range according to the following table:

Type	Precision	Smallest Value	Largest Value
IEEE 32 bit IEEE 64 bit	7 Digits 16 Digits	±1.18 E-38 ±2.23 E-308	±3.39 E+38 ±1.79 E+308

Absolute values below the smallest limit will be regarded as zero while overflow conditions give undefined results.

Bit-fields are always based on 'int' element size.

Memory Models

There are two memory models available (large and banked), selectable with the -m switch. The compiler will by default (or selected by -ml) generate code for a system equipped with up to 64K of RAM / EPROM (including single-chip mode). This model is called large. The bank-switched model (called 'banked' and selected by -mb) is identical to the large model when it

comes to variable allocation. However, it extends the accessible code area beyond the 64K limit with the help of a simple memory mapping scheme supplied in assembly source. The following table shows which memory model to select for a certain hardware configuration :

Memory Model	Large (default or -ml)	Banked (-mb)
Code Area	64K	1M
Variable Area	< 64K	< 64K
IEEE 32 bit IEEE 64 bit	cl831_r20 cl831d_r20	cl83b_r20 cl835bd_r20

Memory Mapped I/O

It is possible to access specific memory locations directly from a C-Program, making it possible to read / write to for example I/O ports, without having to write special assembly routines. The following example shows the technique :

```
#define PORT (*(char *)
(0x4000)) /* Ptr to addr
4000H */

void read_write (char c)
{
 PORT = c;  /* Write c to
absolute address */
 c = PORT;  /* Read of
absolute address */
}
```

Configuration

Usually the C run-time system has to be adapted for the actual hardware that the software is eventually going to run on. This is the case for stack and heapsize as well as input and output addresses, eg. for a keypad or an LCD display.

Stack Size

The basic system requires about 20 - 40 bytes of run-time stack. Apart from the basic needs of the C run-time system, stack requirements are dependent on the worst case function nesting, interrupts and calls to C Library functions. The minimum stack space required by a function is the sum of the formal parameters, local variables and return address location. By default the run-time stack size is set to 512 bytes.

Interrupt Routines

It is possible to write interrupt routines in either C or assembler.

H8/300 Specific Extensions

ICCH8300, the H8/300 C Compiler, is equipped with a set of C extensions for controlling the H8/300 hardware. This coupled with Memory Mapped I/O optimisation decreases the need for assembly language routines considerably. The in-line functions conceptually work like the ANSI-type function declarations shown in *Figure 13*.

C Run-Time-Library

Included with the compiler are four different libraries. Also included are three different formatters for the printf and sprintf routines, and two formatters for scanf and sscanf routines. These are selectable at link-time, to optimise the performance / size ratio.

Linking

The XLINK linker has been designed for ease of use and versatility. Built into it is both the linker, locator and format generator to produce PROMable code as fast as possible without any intermediary files.

Figure 13 - H8/300 Specific Extentions

```
void set_interrupt_mask (ICE mask);
/* loads mask into the I bit in the CCR */

unsigned char read_E_port (unsigned short address);
/* reads to E port */

void write_E_port (unsigned short address, unsigned char value);
/* writes to the E port */

void sleep (void);
/* executes the sleep instruction */
```

XLINK and ICCH8300 together perform a very thorough typecheck of the generated system.

With simple but powerful commands it is possible to link for any memory configuration and generate any of the 30 different output formats to fit most emulators and PROM -programmers. Detailed cross reference listings with module, segment and / or symbol information, even including index for symbols, are easily generated.

Library Manager

With the XLIB librarian it is possible to build new libraries, add or remove functions in existing libraries, etc.

C Library Functions

Character Handling <ctype.h>:
isnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper.

Variable Arguments <stdarg.h>:
va_arg, va_end, va_start.

Non_Local Jumps <setjmp.h>:
longjmp, setjmp.

Input / Output <stdio.h>:
getchar, gets, printf, putchar, scanf, sscanf, sprintf.

General Utilities <stlib.h>
atof, atoi, atol, exit, calloc, free, malloc, realloc.

String Handling <string.h>:
strcat, strcmp, strcpy, strlen, strncat, strncmp, strncpy.

Mathematics <math.h>:
atan, atan2, cos, exp, log, log10, modf, pow, sin, sqrt, tan.

Low-Level Routines <icclbutl.h>:
_formatted_write,
_formatted_read.

C Level Debug

C-SPY is a window oriented C debugger, designed for ease-of-use while not compromising its power. C-SPY is memory-based to be as fast as possible, and also has the option to work in a filebound fashion.

C-SPY has been provided with a complete knowledge of the C language. This allows some unique features, such as debugging by statement rather than by line, since C is built by statement.

All definitions allowed in C are debuggable in C-SPY: stack based or global variables, integers strings, structures, pointers, #define and enum symbols. It even allows for the debug of code in #include- files.

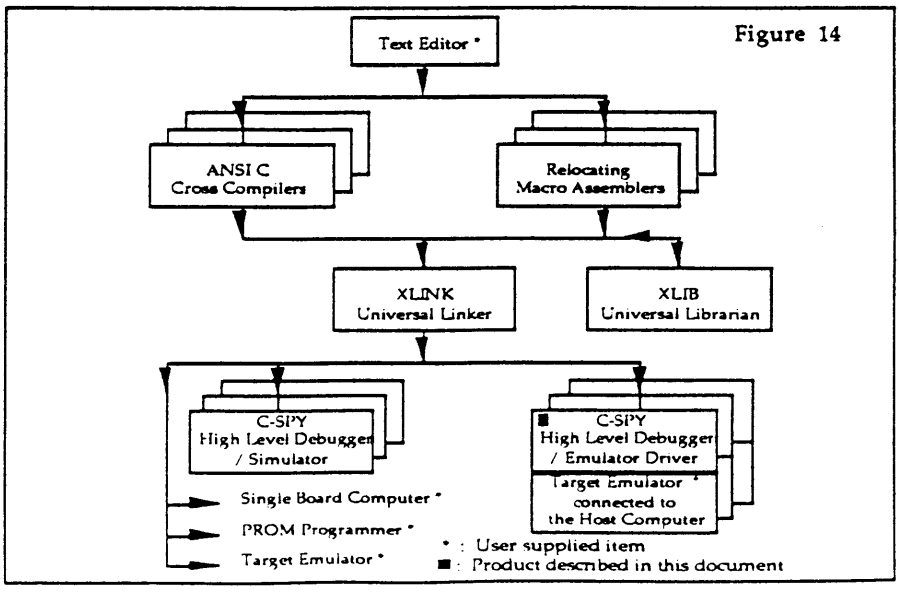
Each version of C-SPY uses the same small set of powerful commands independent of host and target. To improve the ease-of-use further, a two-level on-line help is available. To make usage as comfortable as possible, all commands may be abbreviated and most frequent commands have been defined as function keys.

The simulation of the H8/300 processor is extremely fast with support for both memory models. C-SPY provides debugging capability, both at the C source level and the assembler/ memory/ register level if necessary. This gives the user total access to his programs interaction with the CPU.

System Overview

Figure 14 shows how C-SPY fits together with the other

Figure 14



components of the H8/300 software tool box.

Simulator Architecture

The simulator is essentially a processor with 64K of memory. It is also capable of handling bank-switch code. The memory needed is allocated dynamically to C-SPY by the host.

Emulator Architecture

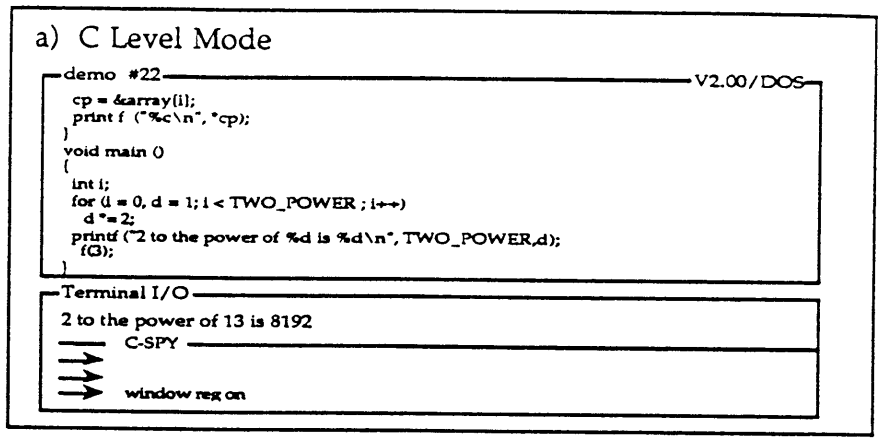
To allow C-SPY to interactively debug real time software, an emulator version is also available. This version of C-SPY provides an interface front end to Hitachi EM83XX emulators.

The user is therefore able to debug software at the C level, while the code is executing in real time, in the actual target system.

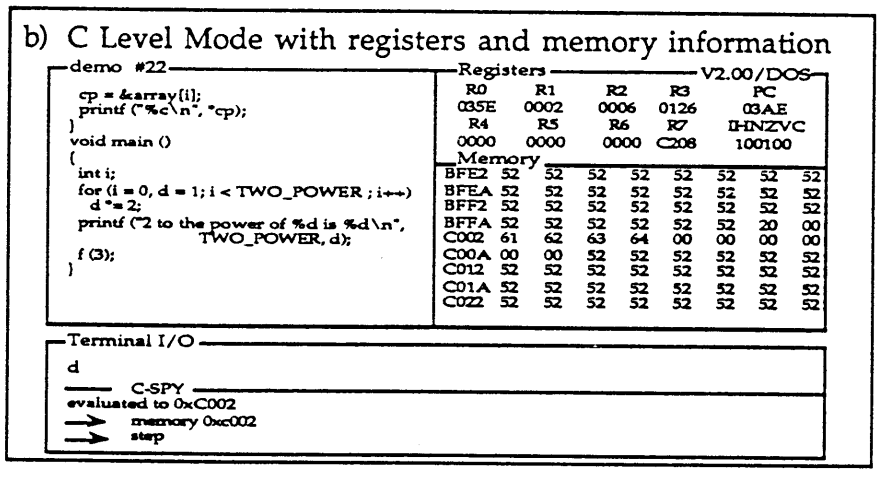
The C-SPY Screen

The screen has three default modes as follows :

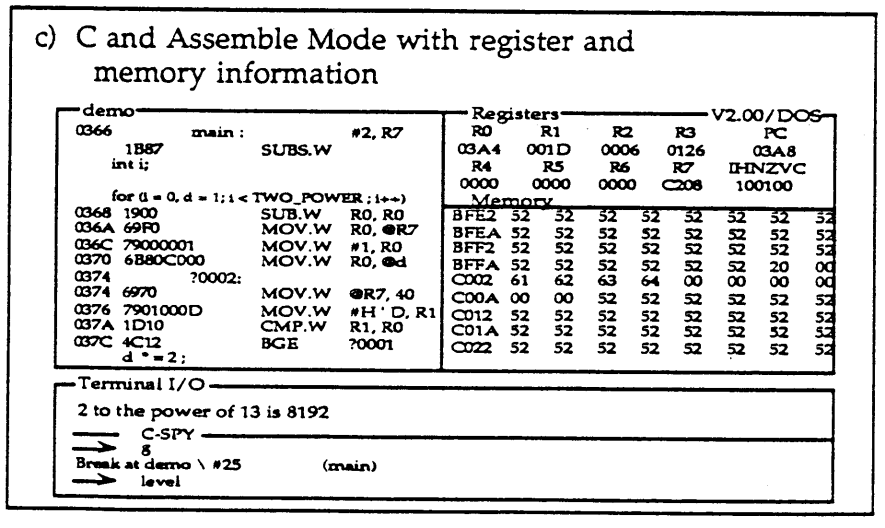
a) C Level Mode



b) C Level Mode with registers and memory information



c) C and Assemble Mode with register and memory information



High Specification Emulator EM83XX

This system provides full real time emulation of the H8/300 family. If used in conjunction with the appropriate C-SPY package this emulator can also provide C source level debugging, whilst retaining its real time emulation capability.

The EM83XX is based around three key components: a main system unit, a target probe and an analyser probe. Interface to the host computer is performed via a high speed serial link.

The microcontroller specific component of the emulator is the target probe, and this circuit uses the evaluation chip of the appropriate H8/300 device. This EV Chip is used, as it has all of the internal buses of the microcontroller bonded out, and therefore allows true real time emulation without the loss of any of the devices internal resources (interrupts, memory etc.).

By changing this probe different members of the H8/300 series can be emulated.

The emulator supports fully symbolic debugging, both in the command line and when viewing disassembled code or listings.

User Interface

To control and drive the emulator a sophisticated menu driven front end software package known as MimeView is provided.

As well as offering user friendly windows, MimeView provides a number of other features which can significantly shorten the learning curve of a new user. These features include a command line which can accept abbreviated keywords and prompt the user with the next available option whilst a command line is being typed.

To prevent lengthy commands having to be constantly re-typed an extensive command line recall and edit facility is also provided.

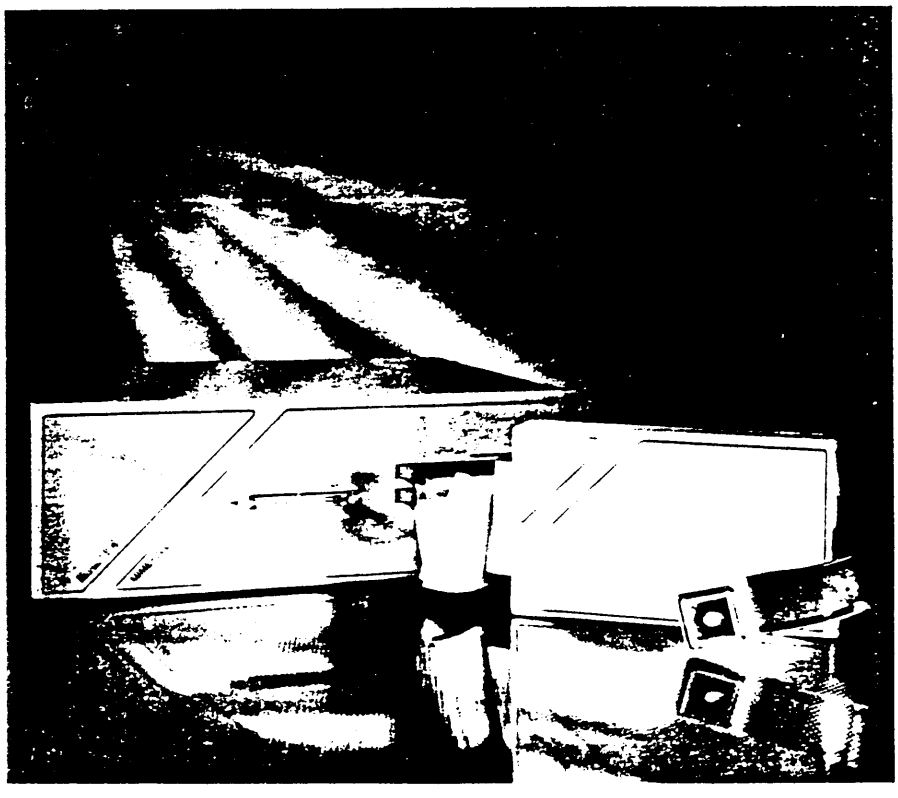
Emulation Memory

To allow full speed emulation of the target processor a large quantity of zero wait state memory is provided (256K standard fit) which can be mapped as either read write memory, write protected or guarded.

The memory mapping command allows blocks of memory to be allocated in blocks of 512 Bytes, and by using this facility the users target system can be accurately emulated.

Breakpoints

The EM83XX emulators have been designed with the maximum breakpoint flexibility possible by virtue of a dynamic breakpoint manager. This part of the system can be used to give the user 3 types of control



points- **break** (program stops), **view** (program stops momentarily, and this fact is reported to the screen) and **trigger** (a trigger pulse is produced from a BNC connector at the rear of the emulator).

These control points can be placed at any memory location, and although they are activated by specified addresses, it is also possible to qualify the break as execution, data read or data write. Using the dynamic break point manager ranges of breaks are also very easy to set.

Events

To enable the user to set the emulator up to perform complex monitoring tasks it has been designed with 4 hardware comparators, which enable emulator functions to be triggered by complex conditions on the various buses within the device (address, data and control) as well as by external events using the sixteen inputs from the analyser probe.

To add a further level of capability to the event handling each comparator has two counters associated with it. These counters are a sixteen bit pass counter and an eight bit delay counter.

The pass counter allows the event to be triggered after a programmed number of event matches, so for example it is possible to trigger an event after the character "a" has been written to the serial port ten times.

Using the pass counter events can be produced a set number of cpu cycles after the initial match condition occured.

Other facilities provided by the event circuitry include sequencing, combination (ANDED) and re-arming. The re-arming facility is particulary useful, as one event can be used to reset another, and tasks such as filtering the bus cycles that are traced can be implemented.

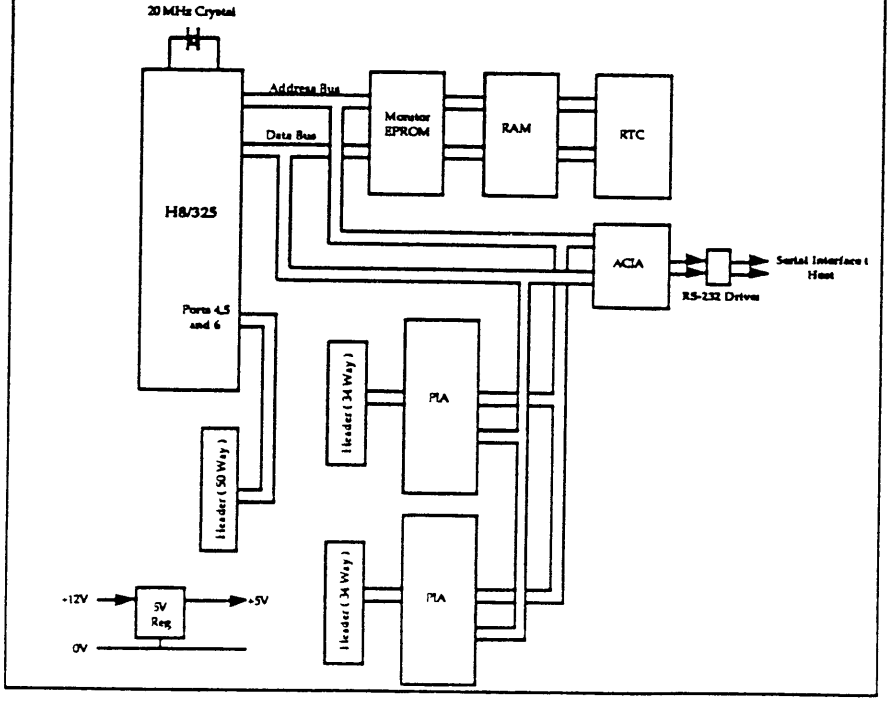
Trace

Each EM83XX emulator comes with an 8K x 128 bit trace memory with can capture address, data and control bus values every cycle. As well as bus information the trace collects 16 user lines (from the analyser probe) and 48 bits of timestamp data.

The timestamping data is used to calculate execution time, either cumulatively from program continuation or relatively from the last instruction.

The trace facility is further enhanced by the events described earlier which allow the user to be very selective about which execution data is collected.

Figure 15 - LEV8325 Block Diagram



Low Cost Evaluation Boards (LEV83XX)

The LEV83XX series of evaluation boards is targetted at providing a cost effective method in which to evaluate a H8/300 microcontrollers suitability for an application.

Aside from this the functionality of the LEV boards can also be utilised to bring a product all the way from initial feasibility studies through to production.

Each LEV83XX board contains a microcontroller running at its full speed, plus some external memory and peripheral devices. A block diagram of the LEV8325 is shown in *Figure 15.*

To ensure that the evaluation board has the same number of I/O resources as a single chip device available to the user the I/O ports used as address / data lines are reconstructed using PIA devices. To allow the board to communicate with a host computer without leaving the microcontroller short of serial I/O an external uart device is used.

Monitor Functions

The microcontroller on the LEV board is running a monitor program known as the Extended Monitor System (EMS) which allows the user to download code to the memory on the board, and to debug it once it has been loaded.

The command set supported by EMS is shown if *Figure 16,* it can support download from a personnel computer in the S-Record object format.

Figure 16 - EMS Commands

COMMAND	DESCRIPTION
A	Assemble instructions
B	Set, display, or cancel breakpoints
BS	Set, display, or cancel break sequence
CV	Convert data
D	Dump memory contents
DA	Disassemble memory
DC	Data change
DS	Data search
F	Fill memory
G	Go
HE	Help information
IL	Load data from host computer
IS	Save data to host computer
IV	Verify memory against file
M	Display or modify memory
MV	Move memory contents
R	Display register contents
S	Single step
SI	Step information
. (full stop)	Modify register value

This format is supported by the cross assemblers and compilers available for the H8/300 family.

LEV83XX Hardware

As has been mentioned earlier great care has been taken when designing the LEV series of boards to ensure that all hardware resources of the microcontroller are preserved, using external components where necessary.

Aside from this circuitary the design of the LEV is very simple, with only the minimum external hardware possible used.

To allow the user to connect the microcontroller to target hardware all of the device's I/O lines are accessible via headers. To make things even easier, each LEV also has a small wire wrap area on which some interfacing hardware can be fabricated easily.

The LEV boards have a dual purpose, as they are each also equipped with a peripheral device, a HD64160 Real Time Clock on the LEV8325 and a Universal Pulse Processor on the LEV8330. These boards are therefore excellent tools with which to evaluate these peripherals.

Development Tools - Ordering Information

PACKAGE	PART NO. DOS	PART NO. VAX / VMS
Ansi C Compiler Cross Assembler Utilities	SE083CPC	SE083CVAX
C-SPY High level language debugger (simulator)	SE083CSPC-S	SE083CSVAX
H8/300 Cross Assembler Utilities	SE083PC	SE083VAX
Assembler level H8/330 simulator	SE083SIMPC	-
Software Designer kit containing : Ansi C Compiler Cross Assembler C Level Debugger Assembler Level Simulator (H8/330)	SE083SDKPC	-
H8/32X evaluation board	LEV8325	-
H8/330 evaluation board	LEV8330	-
H8/325 in circuit emulator	EM8325	-
H8/330 in circuit emulator	EM8330	-
H8/325 system kit (contains emulator plus software design kit)	S3-8325	-
H8/330 system kit (as H8/325)	S3-8330	-

Technical Questions and Answers

Product	H8/300 CPU	Q&A No.	QA8300-027A
Topic	Debug information		

Question

1. If I link a program with the /DEBUG option and store it as an absolute module, then use the converter to convert it from SYSROF type format to S type or HEX type format, will debug information be included?

Classification—H8/300

	Registers
	Read timing
	Write timing
	Interrupts
	Reset
	External expansion
	Power-down state
	Instructions
○	Software
	Development tools
	Miscellaneous

Answer

1. Debug information is included only in the SYSROF type format.

   No debug information is included in the S type or HEX type format.

Related Manuals

Manual Title

Other Technical Documentation

Document Name

Related Microcomputer Technical Q&A

Title

Additional Information