THE ST9 MCU FAMILY

PRODUCT OVERVIEW

THE ST9 8-16/BIT MCU FAMILY: INNOVATIVE SOLUTIONS FOR EMBEDDED CONTROL

The rapidly growing area of real-time applications represents one of the most exacting operating environments for today's microcontrollers. Processors are required to execute complex control algorithms, within a defined minimum response time. With the increasing complexity of embedded control applications, a significant increase in CPU performance and peripheral functionality over conventional 8-bit controllers is required.

Designed to meet market needs for cost-effective, high performance MCUs, the ST9's family bridges the gap with the worlds of 8 and 16-bit microcontrollers and covers a large range of requirements in the high-end 8-bit and low-end 16-bit applications. With an ST9 microcontroller you have the 16-bit performance (sophisticated data manipulation, real time event handling) and the 8-bit advantages (price, noise, power consumption,...).

With the ST9 family, STMicroelectronics offers significant performance and flexibility advantages over traditional 8-bit microcontrollers: it is the unequalled solution for more performance. It provides innovative answers to yours embedded control requirements with competitive MCU solutions for today and tomorrow.

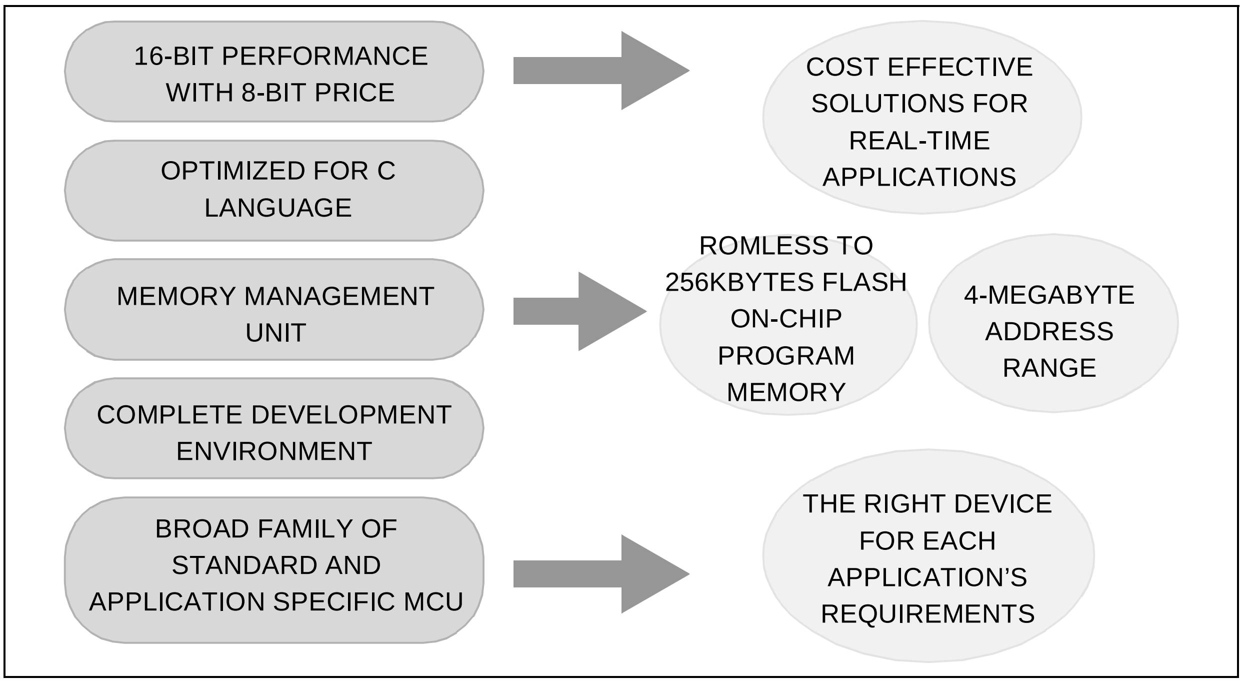
The ST9 family is one of the most powerful range of 8/16-bit MCUs available on the market. You combine the performance of a 16-bit microcontroller with the cost of a 8-bit microcontroller, with to a unique set of benefits for realtime applications.

Rev. 1.1

Table of Contents

Table of Contents

Figure 1. A unique set of benefits for real-time embedded control


16-BIT PERFORMANCE WITH 8-BIT PRICE
OPTIMIZED FOR C LANGUAGE
MEMORY MANAGEMENT UNIT
COMPLETE DEVELOPMENT ENVIRONMENT
BROAD FAMILY OF STANDARD AND APPLICATION SPECIFIC MCU
COST EFFECTIVE SOLUTIONS FOR REAL-TIME APPLICATIONS
ROMLESS TO 256KBYTES FLASH ON-CHIP PROGRAM MEMORY
4-MEGABYTE ADDRESS RANGE
THE RIGHT DEVICE FOR EACH APPLICATION'S REQUIREMENTS

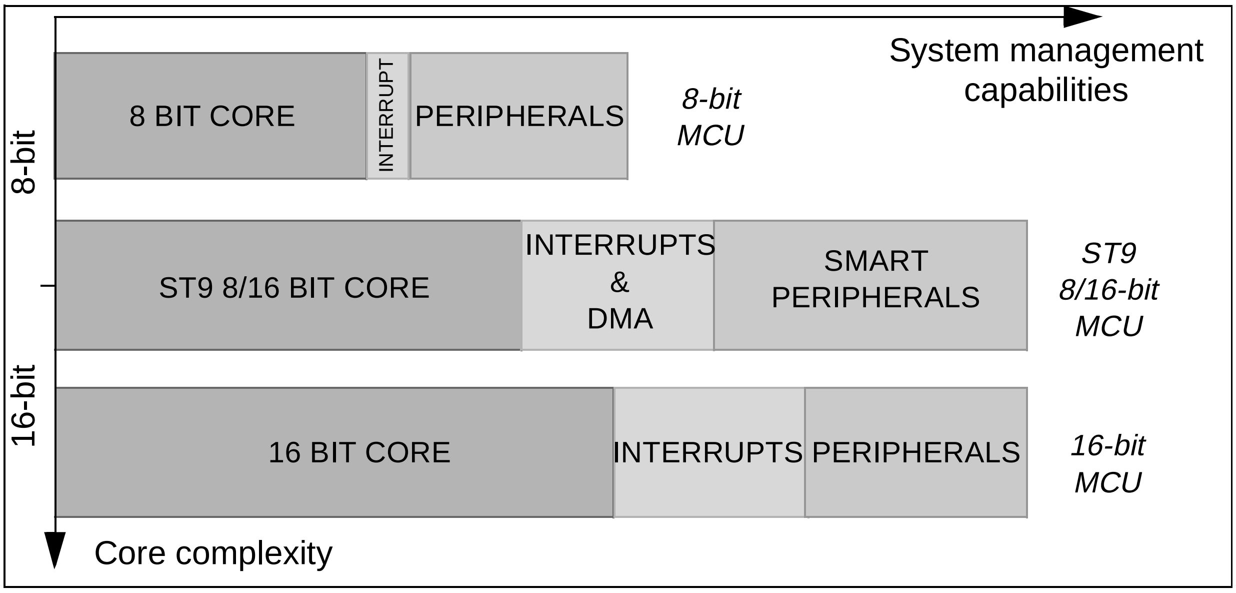
The performance of the ST9 microcontroller derives from its architecture. It has a Register-File based core optimized for sophisticated data manipulation and real-time event handling, together with a range of application-aware intelligent peripherals with the power to carry out most tasks with the minimum processor overhead.

In the ST9, the intelligence is distributed between the core and its peripherals. The core includes the Central Processing Unit (CPU), the Register File, the interrupt and Direct Memory Access (DMA) controller, and the Memory Management Unit (MMU). The MMU allows addressing of up to 4 Megabytes of program and data mapped into a single linear space. Instructions have been added to facilitate large program and data handling through the MMU, as well as to improve the performance and code density of C Function calls. 14 addressing modes are available, including powerful indirect addressing capabilities. The management of the stack which is divided into a system stack for interrupts and subroutine calls and a user stack provide optimized support for C language.

When you compare different microcontrollers, you can estimate the relative computing power of the core, and also of the peripherals (if they include some intelligence). In some architectures, the peripherals make intensive use of the core and thus take up a part of its computing power. Many microcontrollers available on the market have a relatively powerful core, surrounded by very simple peripherals. This approach has the advantage of making the peripherals easy to use and configure but at the expense of the overall computing power and system management capability.

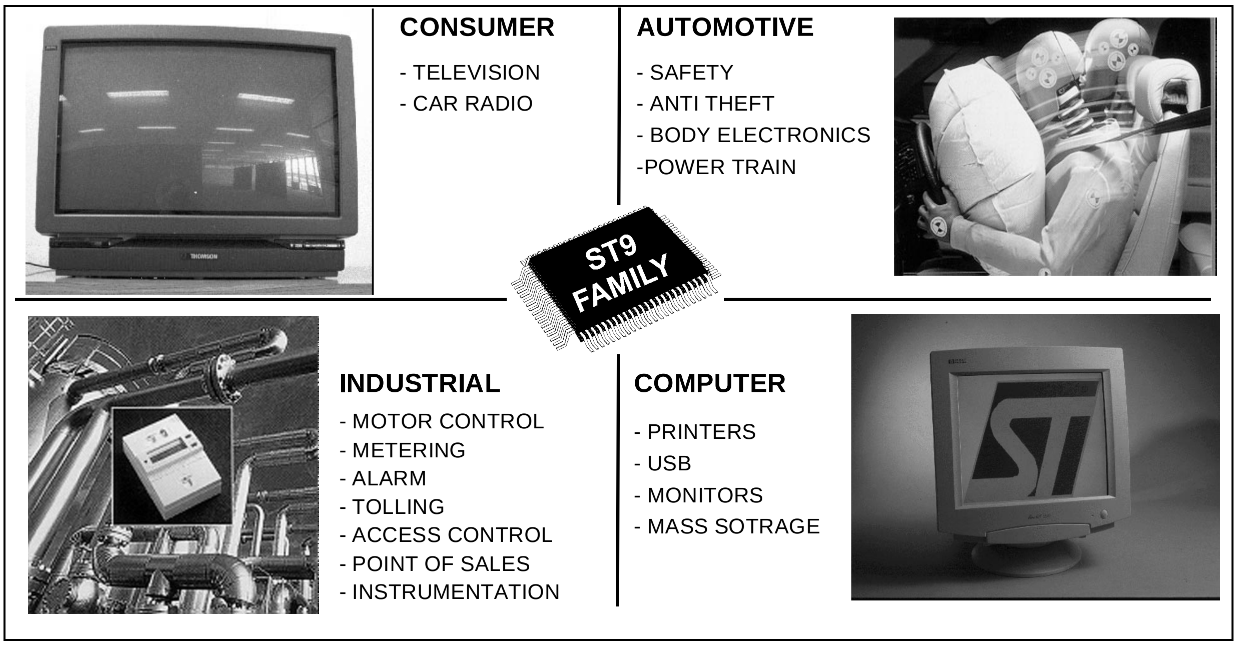
The ST9 is an example of a radically different compromise. Its core is comparable to the best 8-bit microprocessors on the market, and it boasts an impressive speed, (more than five instructions per microsecond). It is assisted (rather than just surrounded) by peripheral blocks most of which can perform complex tasks without the intervention of the core. The net result is a powerful machine that can even perform impressive tasks just using its peripherals.

Figure 2. 16-bit performance with 8-bit cost and flexibility



The ST9 family covers a large range of markets. ST9 MCU devices fit in a range of applications from automotive to industrial, consumer and computer.

Figure 3. A broad range of applications



With their large set of peripherals and memory variants, the ST9 family members integrate all the functionality needed to control a large range of real time systems.

Table 1. 8/16-Bit high performance core for fast real time management

Device	Program Memory Type					Prog. (Bytes)	RAM (Bytes)	Data E^2PROM (Bytes)	A/D Inputs	Timer Functions		Serial Interface	I/Os (High Current *))	Packages	Special Features
	Flash	OTP	FAST	ROM	EPROM					16-Bit (IC/ OC/ PWM)	Others				
ST90135M5				●	●	24K	768		8x8-Bit	3 (4/4/5)	WDG	SPI/SCI	67	TQFP80/ PQFP80	PLL Clock
ST90135M6				●	●	32K	1K		8x8-Bit	3 (4/4/5)	WDG	SPI/SCI	67		
ST90158M7				●	●	48K	1.5K		8x8-Bit	4 (6/6/7)	WDG	SPI/2xSCI	67		
ST90158M9		●		●	●	64K	2K		8x8-Bit	4 (6/6/7)	WDG	SPI/2xSCI	67		
ST90R158						None	2K		8x8-Bit	4 (6/6/7)	WDG	SPI/2xSCI	51		ROMless, PLL Clock
ST92F124	❍					60K	2K	1K	8x10-Bit	3 (4/4/5)	WDG	SPI/SCI/I^2C	48	TQFP64	PLL Clock
ST92F150J	❍					128K	4K	1K	16x10-Bit	5 (8/8/7)	WDG	SPI/2xSCIs/ I^2C	77	PQFP100	PLL Clock, J1850 BLPD
ST92F150C	❍					128K	4K	1K	16x10-Bit	3 (4/4/5) 5 (8/8/7)	WDG	SPI/2SCIs/ I^2C/CAN	48/ 77	TQFP64/ PQFP100 TQFP100	CAN 2.0B active, PLL Clock
ST92F150JD	❍					128K	6K	1K	16x10-Bit	5 (8/8/7)	WDG	SPI/2xSCIs/ I^2C/CAN	77	P/TQFP100	Two CAN 2.0B active, PLL Clock, J1850 BLPD
ST92F250C	❍					256K	8K	1K	16x10-Bit	5 (8/8/7)	WDG	SPI/2xSCIs/ 2xI^2Cs/CAN	80	P/TQFP100	CAN 2.0B active, PLL Clock
ST92141K4		●	●		●	16K	512		6x8-Bit	2 (2/2/2)	WDG	SPI	15 (4)	SDIP32/SO34	Asynchro-nous 3-phase Mo-tor Control-ler, PLL clock
ST92163R4		●	●	●	●	20K	2K		6X8-Bit	1 (2/2/2)	WDG	I^2C/SCI/USB	44 (8)	TQFP64	16 full-speed USB endpoints, PLL clock

*) Number of high current pins included in the number of I/O pins.

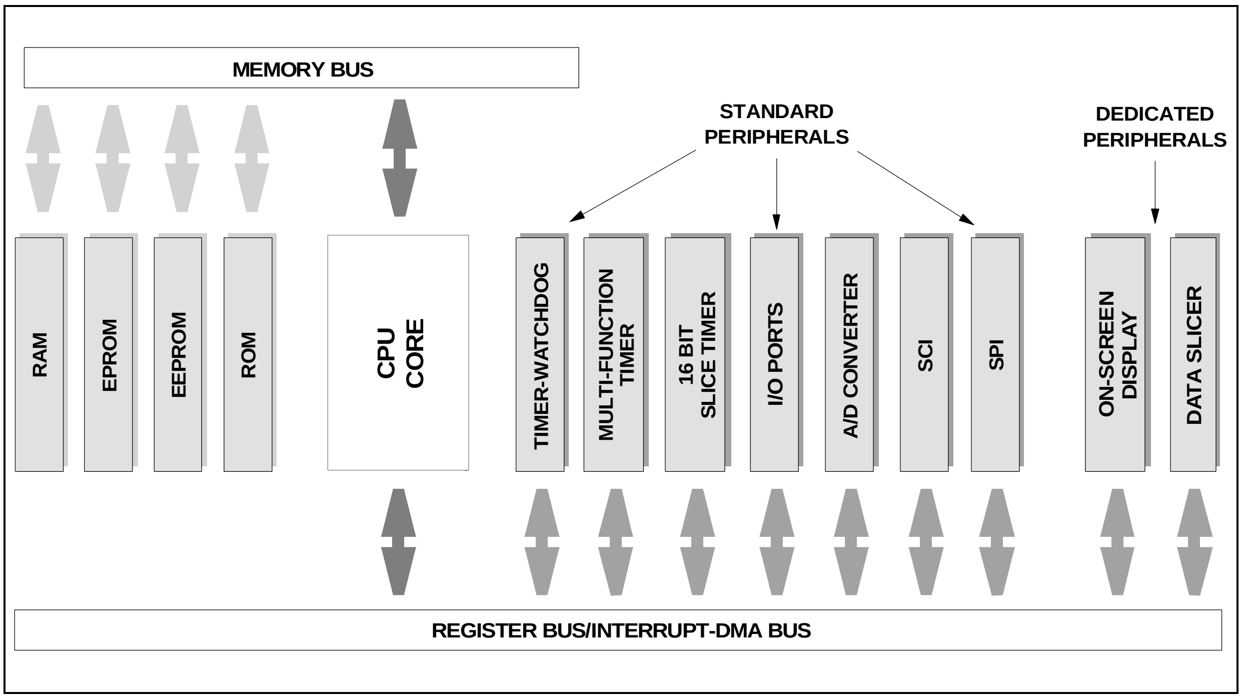
Other family members include or will soon include multi purpose as well as application specific devices designed for Automotive (CAN, J1850), Consumer (TV), Computer (Monitors, USBus), Industrial (Motor Control, Electronic Tolling and Access Control) systems.

Table 2. ST9 selection list

ST9 Family	ROM	RAM	DATA E²PROM	Peripherals	Package	Target Market
ST90135	24/32K	768/ 1K	-	SPI, SCI, 3 Timers, Watchdog, ADC, PLL Clock	TQFP80/ PQFP80	Consumer, Automo-tive, Industrial, Telecom
ST90158	48/64K	1.5K/2K	-	SPI, 2 SCIs, 4 Timers, Watch-dog, ADC, PLL Clock	TQFP80/ PQFP80	Consumer, Automo-tive, Industrial, Telecom
ST90R158	-	2K	-	SPI, 2 SCIs, 4 Timers, Watch-dog, ADC, PLL Clock	TQFP80/ PQFP80	Consumer, Automo-tive, Industrial, Telecom
ST92F124	60K	2K	1K	SPI, SCI, I²C, 3 Timers, Watch-dog, ADC, PLL Clock	TQFP64	Consumer, Automo-tive, Industrial, Telecom
ST92F150	128K	4K/6K	1K	SPI, 2 SCIs, I²C, CAN, 5 Timers, Watchdog, ADC, PLL Clock	TQFP64/ TQFP100/ PQFP100	Consumer, Automo-tive, Industrial, Telecom
ST92F250	256K	8K	1K	SPI, 2 SCIs, 2 I²Cs, CAN, 5 Tim-ers, Watchdog, ADC, PLL Clock	PQFP100	Consumer, Automo-tive, Industrial, Telecom
ST92141	16K	512	-	2 Timers, Motor Controller, SPI, ADC, PLL Clock	PSDIP32/ SO34	Motor Control
ST92R195 *) ST92195 *)	ROMLess 32K/64K	8K/ 12K	-	OSD, Teletext	SDIP56	TV Applications
ST92163	20K	2K	-	I²C, SCI, USB Functions, Watch-dog, PLL Clock, ADC, Timer	TQFP64/ SDIP56	USB bus PC Peripherals
ST92175*)	60/96/128K	2/2.5/3K	-	Timers, Sync Processor, SCI, I2C/DDC, PWM, A/D	SDIP56, TQFP64	Monitors

With the ST9's **modular architecture** and the unique core technology, you can easily up or downgrade within the same product family keeping software investment. You can choose the right device for the right task, at the right price from a broad product folio.

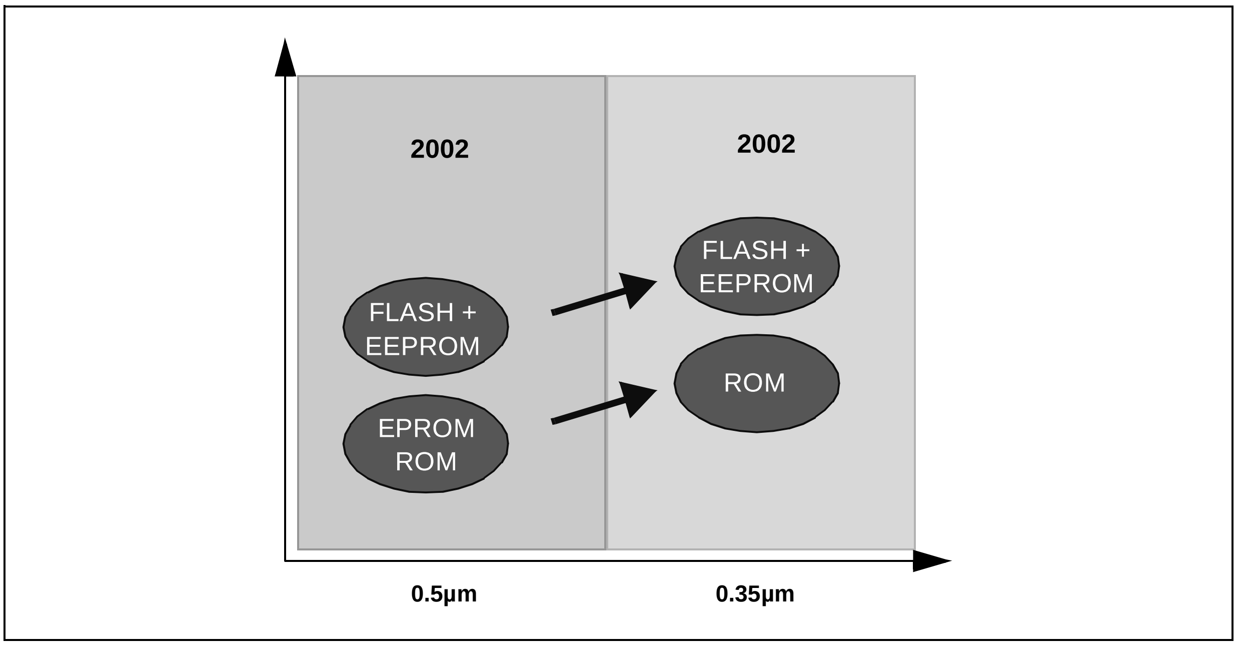
Figure 4. ST9 modular architecture



In addition, the ST9 can be easily customized to specific requirements of high volume applications

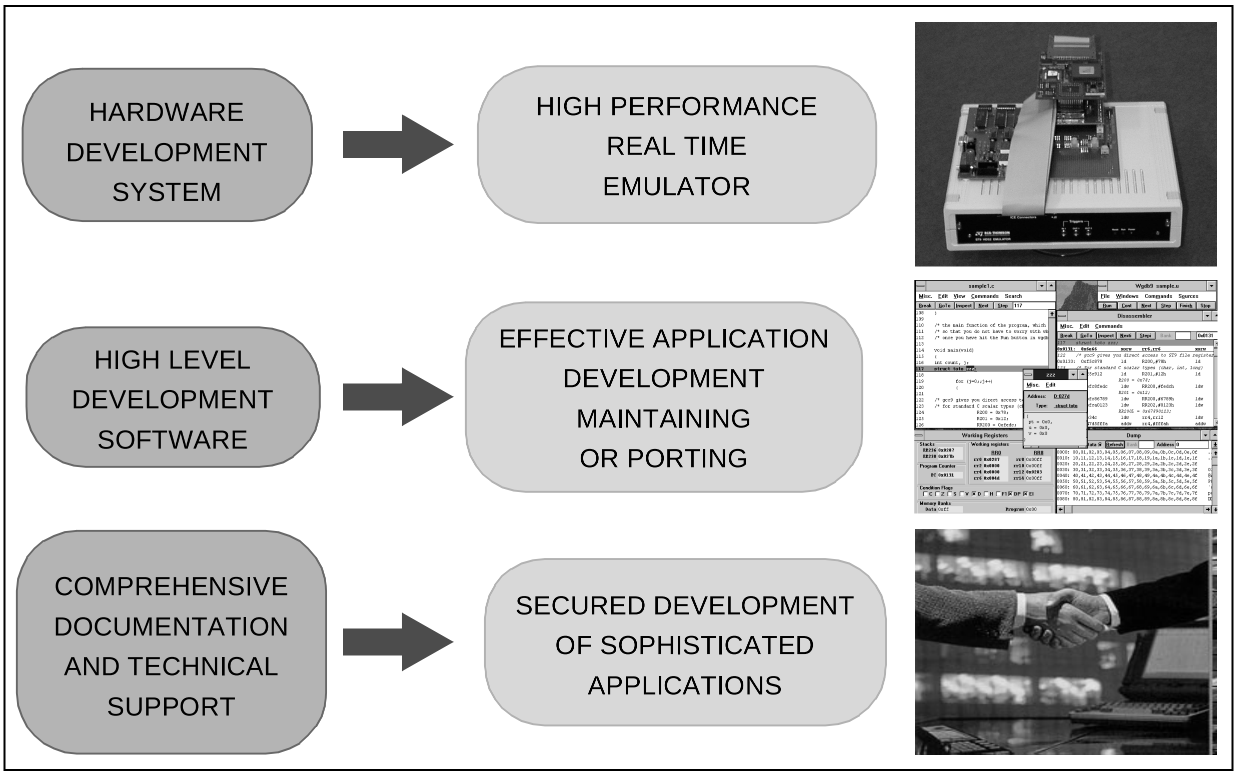
ST9 devices are manufactured using state of the art technology, allowing low power consumption, maximum performance, cost efficiency and reliability.

Figure 5. Competitive technologies for today and tomorrow



The ST9 family of MCUs is supported by a comprehensive range of development tools: a software package (C-compiler, assembler, linker, archiver, debugger, Real Time Kernel, CAN SW Drivers) and a set of hardware tools (emulators, programmers).

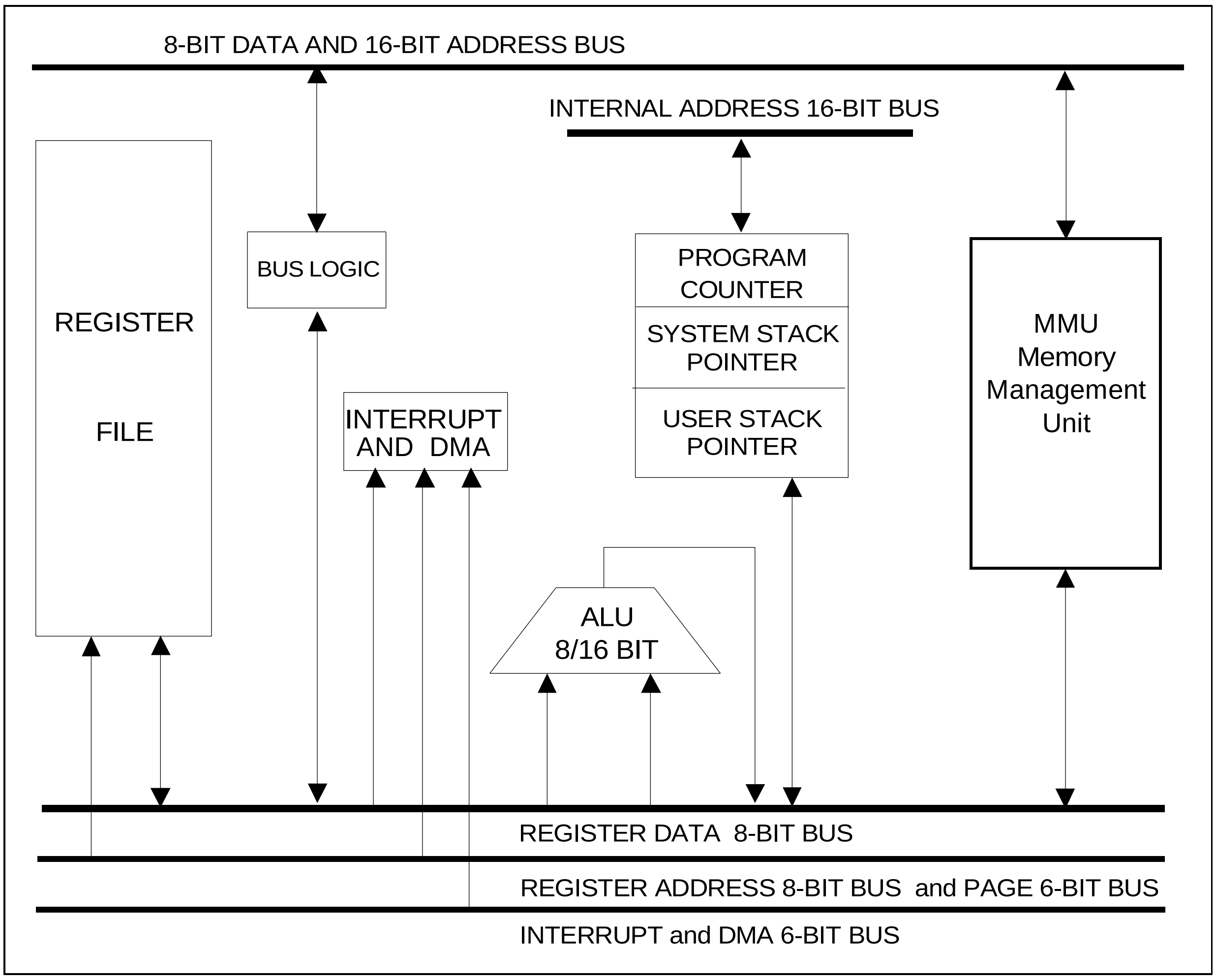
Figure 6. A complete development environment


HARDWARE DEVELOPMENT SYSTEM
HIGH PERFORMANCE REAL TIME EMULATOR
HIGH LEVEL DEVELOPMENT SOFTWARE
EFFECTIVE APPLICATION DEVELOPMENT MAINTAINING OR PORTING
COMPREHENSIVE DOCUMENTATION AND TECHNICAL SUPPORT
SECURED DEVELOPMENT OF SOPHISTICATED APPLICATIONS

1 THE ST9 REGISTER BASED CORE

1.1 DESCRIPTION

Figure 7. ST9 core block diagram


8-BIT DATA AND 16-BIT ADDRESS BUS
INTERNAL ADDRESS 16-BIT BUS
REGISTER FILE
BUS LOGIC
INTERRUPT AND DMA
PROGRAM COUNTER
SYSTEM STACK POINTER
USER STACK POINTER
MMU Memory Management Unit
ALU 8/16 BIT
REGISTER DATA 8-BIT BUS
REGISTER ADDRESS 8-BIT BUS and PAGE 6-BIT BUS
INTERRUPT and DMA 6-BIT BUS

The ST9 core consists of:

– the **Central Processing Unit** with an 8-bit Arithmetic Logic Unit, the brain of the system that processes all data and controls the internal busses.

– the **Register File**, a set of 256 registers including:

  – 224 general purpose registers available as 8 or 16-bit accumulators, index registers or address pointers

  – 16 system registers (stack pointers, flags, modes, interrupts,...)

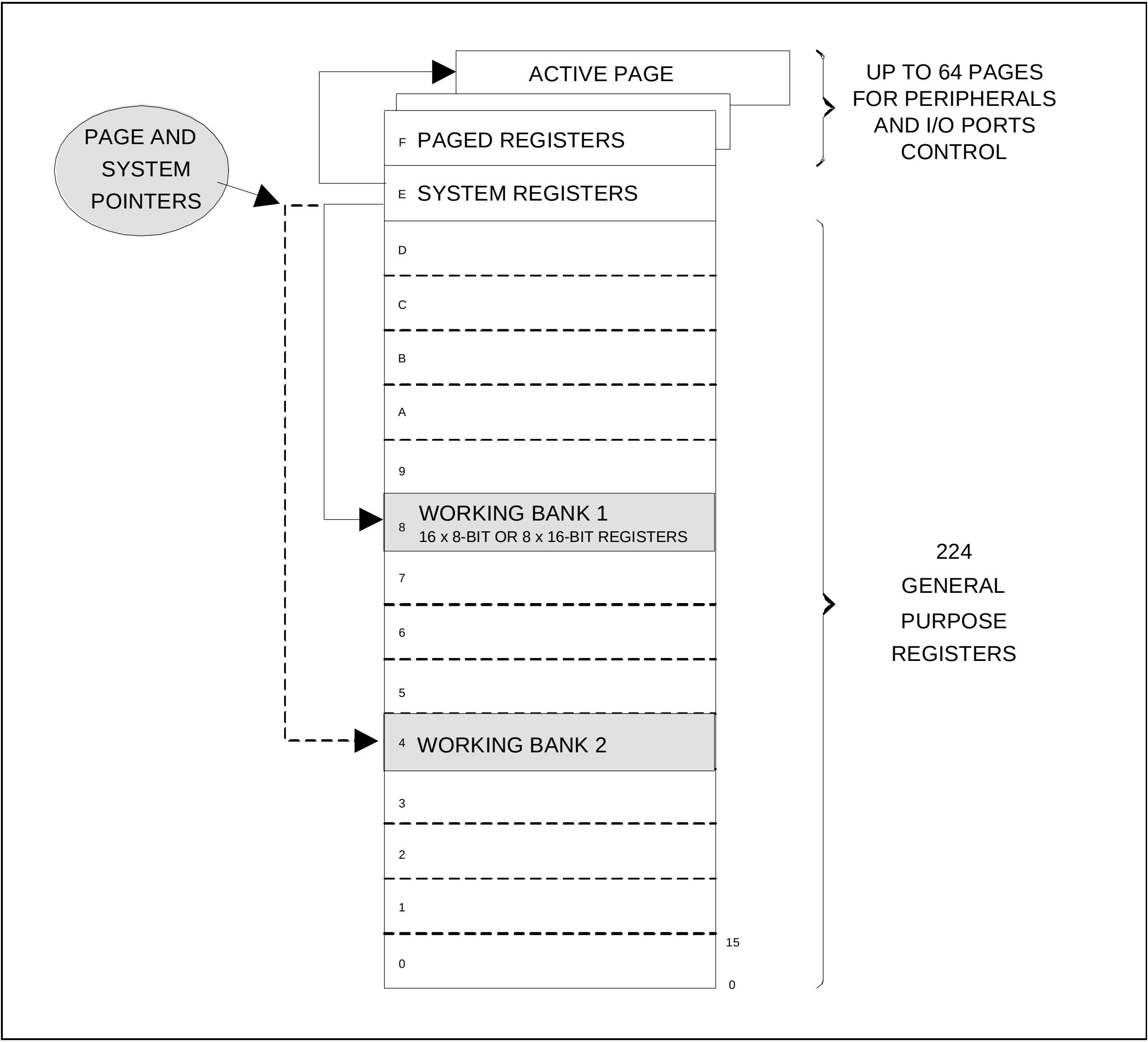
  – 64 pages of 16 registers each for peripheral management (generally, one page for each peripheral)

– a **Memory Management Unit** which allows addressing of up to 4 Mbytes of program and data mapped into a single linear space

– 6-bit **Interrupt and DMA** bus connected to each peripheral for interrupts and to the SCI, MFT, I2C and USB for DMA

The ST9 core has an unique and powerful structure. Its architecture is built around a set of registers called **Register File**. This allows efficient bit, byte and word data handling and supports high level languages more efficiently than traditional accumulator machines.

Figure 8. Register File structure



The **Direct Memory Access** and the multiple priority **Interrupt Controller** further increase the ST9's capability to manage sophisticated real time, task and communications-intensive applications effectively.

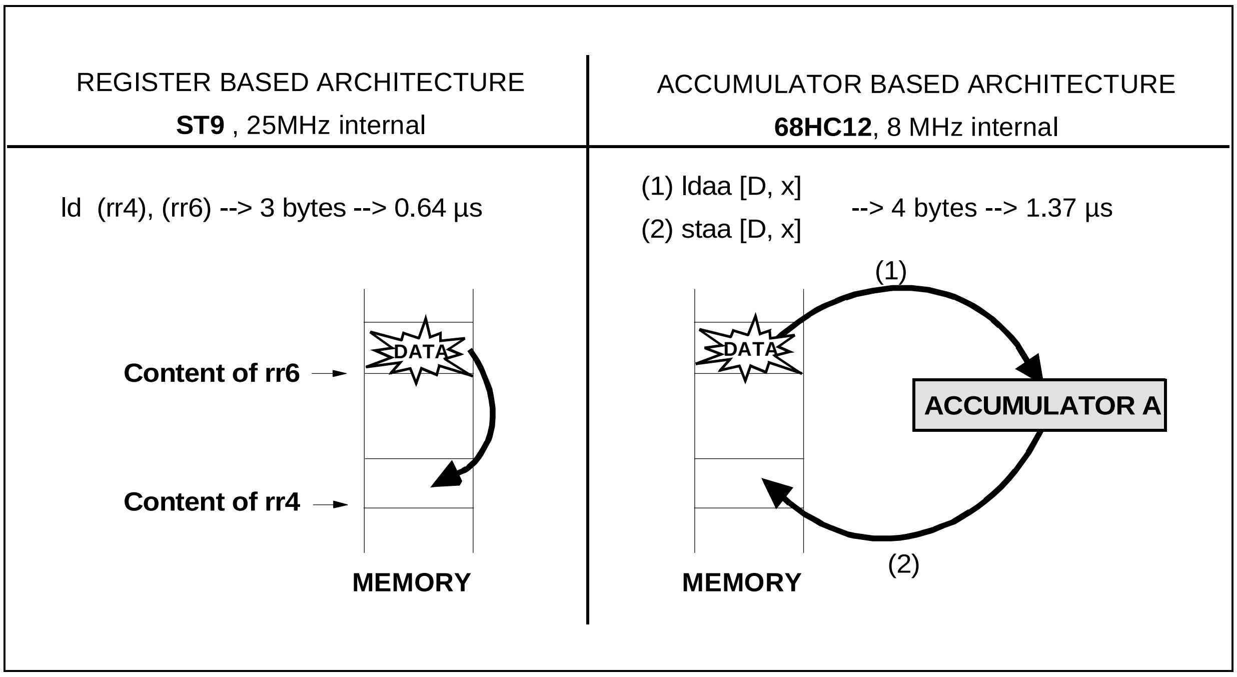
1.2 BENEFITS OF THE REGISTER BASED ARCHITECTURE

The usual microprocessor core structure is based on an accumulator. The accumulator is the one register that holds the data to work on and the results of the arithmetic or logical operations applied to it. This classic structure is characterised by its simplicity: the internal data paths of the microprocessor all converge to the accumulator. The instruction set is simple, since you need to specify only one memory address in a data move instruction, the other being implicit: the accumulator itself.

This simplicity has its drawbacks: the accumulator is the computation bottleneck, since to move data from one place in memory to another place, you have to do it through the accumulator. The simplest transfer involves at least two instructions: one to get the data, the other one to store it.

Register oriented models, in contrast, allow you to move data directly from one place to another in a single instruction as illustrated in the example below.

Figure 9. Example of 8-bit memory-memory transfer



Data can come from a register or from a memory address and can go to either to a register or a memory address. You can code the addresses in the instruction, or store them in registers referenced by the instruction. This allows you to optimize your code by choosing to store frequently used data in registers, leaving less frequently used data in memory.

The register based architecture saves execution time and code lines because there are less save and restore operations of data and pointers.The example given in Figure 10 shows the difference between the two architectures when you want to add two 8-bit operands from memory and to store the result in memory.

Figure 10. Example of adding two 8-bit memory operands

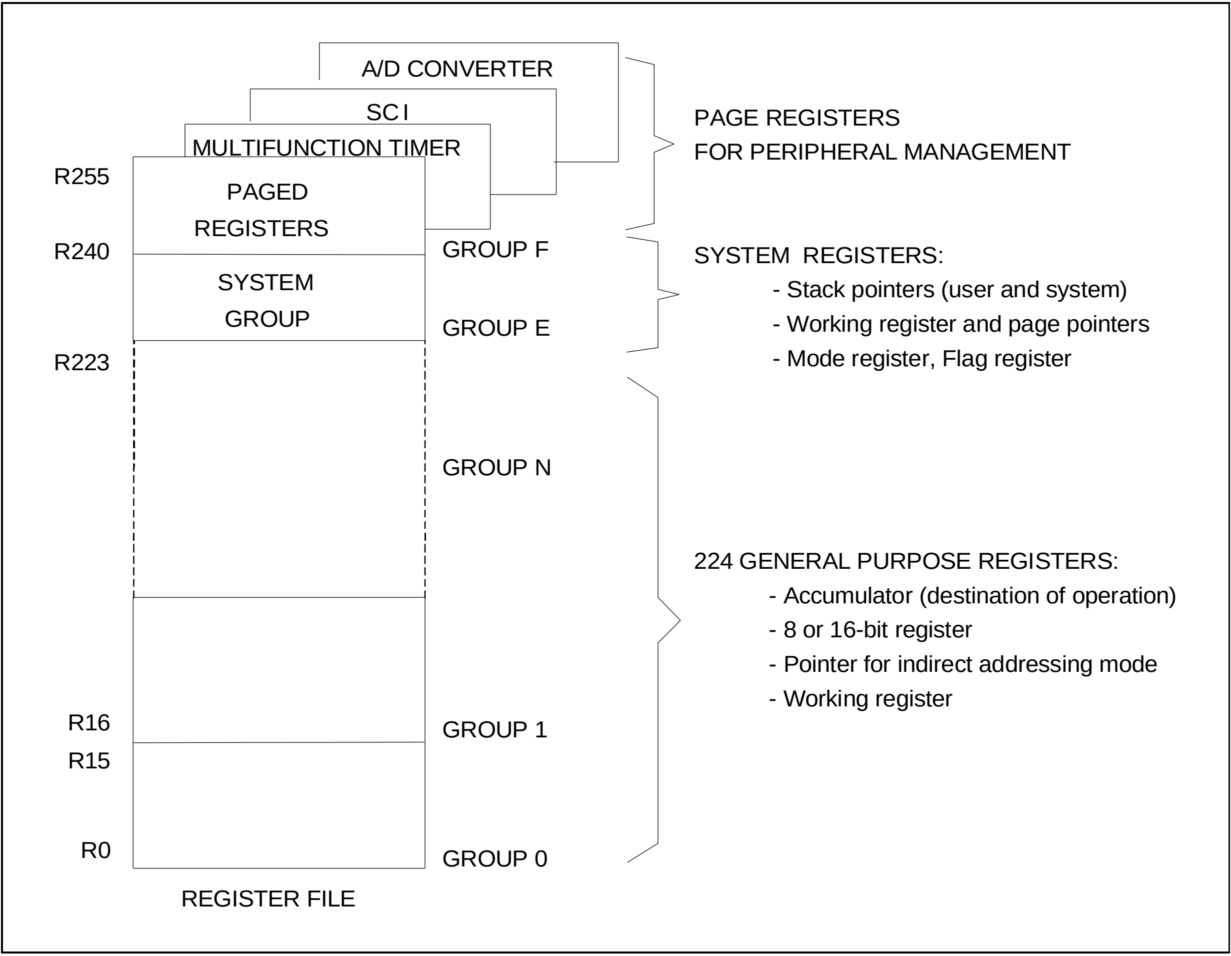
MICROCONTROLLER	ARCHITECTURE	INTERNAL CLOCK SPEED	CODE	BYTES	EXECUTION TIME
ST9	8/16-BIT REGISTER MACHINE	**25 MHz**	add (rr), (rr) (16-bit indirect)	**3**	**0.56 µs**
68HC12	16-BIT ACCUMULATOR MACHINE	**8 MHz**	ldaa [D, x]	2	750 ns
			adda [D, y]	2	750 ns
			staa [D, x] (16-bit indexed)	2	625 ns
				6	**2.12 µs**

The advantage of register based architecture is obvious in number of lines and bytes and in execution time.

1.3 THE ST9 REGISTER FILE

The ST9 has a special addressing space for registers, providing 256 different register addresses. This large amount of registers gives you considerable flexibility in allocating variables. Register addresses are coded using one byte. You can use any of these registers to hold data or as a pointer either to other registers or to bytes in memory.

Figure 11. ST9 Register File organization


A/D CONVERTER
SC I
MULTIFUNCTION TIMER
R255
PAGED REGISTERS
R240
SYSTEM GROUP
R223
R16
R15
R0
REGISTER FILE
GROUP F
GROUP E
GROUP N
GROUP 1
GROUP 0
PAGE REGISTERS FOR PERIPHERAL MANAGEMENT
SYSTEM REGISTERS:
- Stack pointers (user and system)
- Working register and page pointers
- Mode register, Flag register
224 GENERAL PURPOSE REGISTERS:
- Accumulator (destination of operation)
- 8 or 16-bit register
- Pointer for indirect addressing mode
- Working register

1.3.1 Reduced code size with working register concept

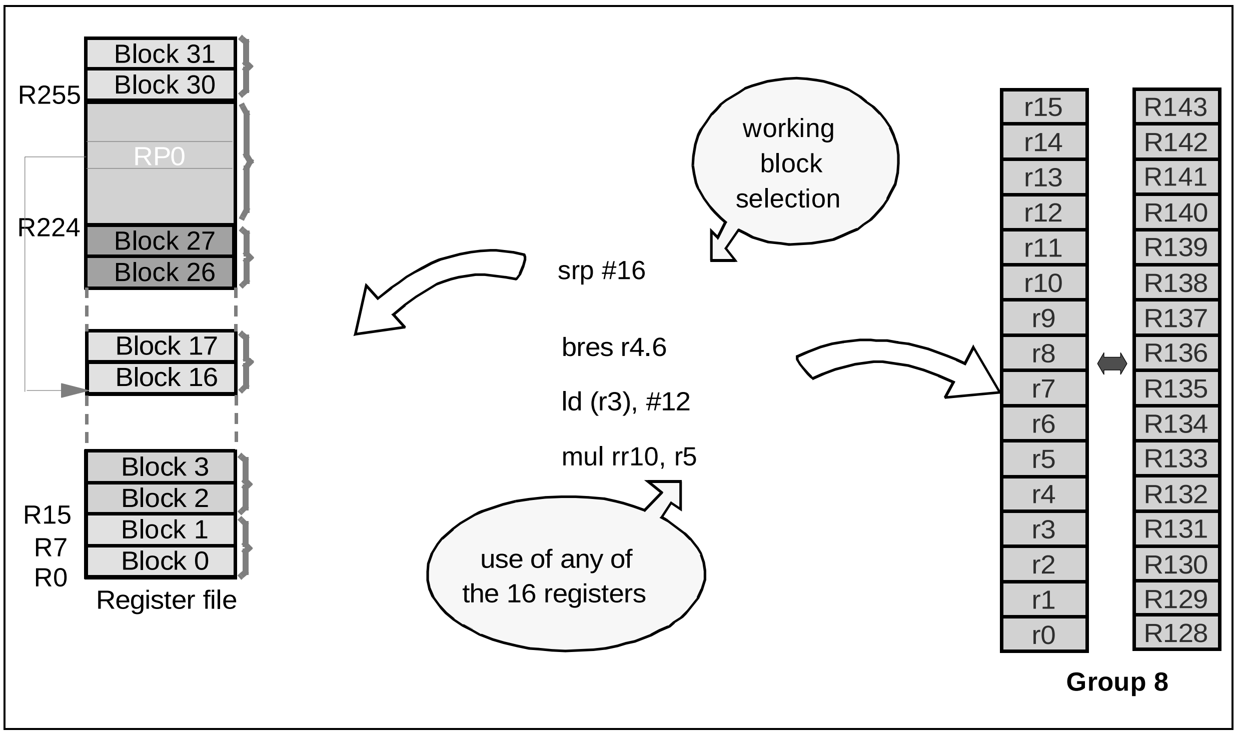
To further improve coding efficiency, a special mechanism has been created: the concept of working register. This mechanism is a short direct addressing mode which provides faster execution and more compact code. It reduces to just 16 bytes the register space accessible by the instructions in the so-called working register addressing mode. Only four bits are required to address this space, allowing both the source and the destination of a data move to be coded in a single byte, thus saving both **code size** and **execution time**.

Figure 12. Example of the benefits of working registers

OPCODE	INSTRUCTION FORMAT	CODE SIZE	EXECUTION TIME
CP r, r	[opc] [dst | src]	2 bytes	160 ns
CP R, R	[opc] [src] [dst]	3 bytes	240 ns

Working register mechanism is mandatory for bit manipulation instructions, to use a register as a pointer in most indirect addressing modes, for multiply and divide instructions.

Figure 13. Example of using working registers



1.3.2 Quick peripheral access with paging mechanism

All internal peripherals are mapped in the register space and you can access them with a fast 8-bit bus. Most of them have a multitude of features and can be configured in different ways. This implies that they have a large number of registers. Since only the last group of 16 registers is allocated for peripherals, a special scheme must be used to overcome this problem. It is called paging.

The last group of registers actually addresses one page of 16 registers that belongs to one of the peripherals. Which page of which peripheral depends on the value of a register called the Page Pointer Register. There can be as many as 64 different pages, providing plenty of space for accessing peripherals.

Although the handling of the peripheral pages requires extra bytes, the 8-bit ST9 register file address bus saves overall **execution time** and keeps some software compatibility between devices with different peripheral sets.

1.3.3 Fast context switching using working registers and register pointers

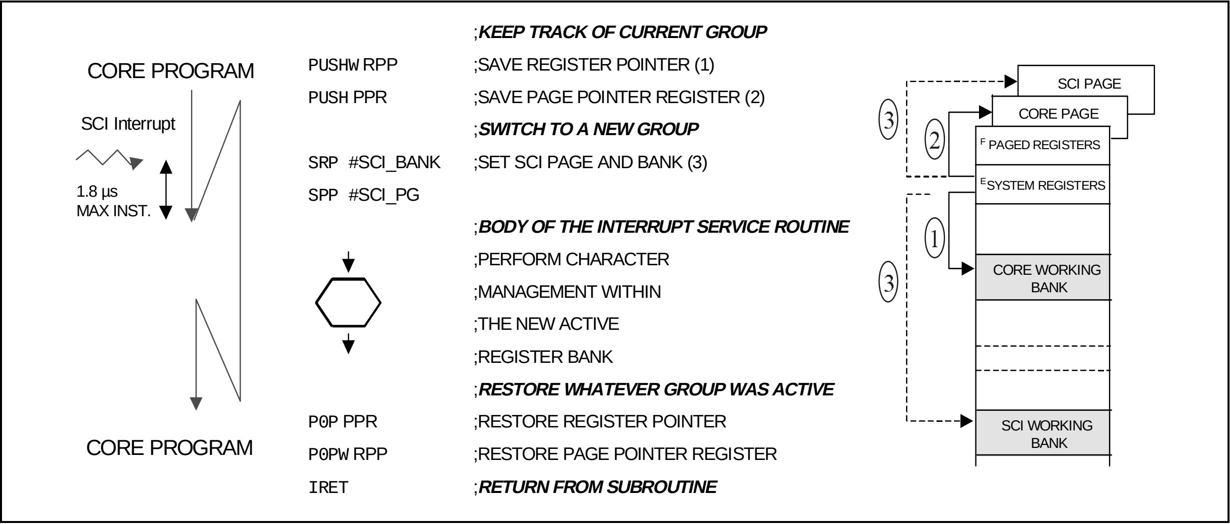
Another benefit of the ST9 Register File is the capability of effectively switching context on asynchronous events by simply pointing to a new active page and bank after saving the current values in the stack.

The working registers offer a workspace of 16 bytes. This is sufficient for most applications, and much more convenient than a single accumulator. However, in some applications, this is still not enough. In this case you can easily allocate more than one register group to a particular program module. Since any register can be accessed directly, it is up to you to decide whether you want to switch working register groups or not to access the other groups of registers.

Since changing the current group involves only one instruction, the concept of working registers can greatly reduce **context switching** time, for example in the case of an interrupt service routine. Doing this preserves the contents of the whole group, and the reverse operation restores them, as in the example below which shows the reception of a character from the Serial Interface.

While the critical parameters of the event are managed in a reserved “SCI working bank”, the key parameters of the interrupted task (the core program or another interrupt routine of lesser priority) are left untouched in the “Core working bank”. This means that critical information for core program or routines remains in the register file. Only register and page pointers need to be stacked. Once the incoming character has been managed, the pointers in the system bank are restored to their previous values and the interrupted task is resumed, resulting in minimum data manipulation and maximum execution speed.

Figure 14. Example of reading one character from SCI



```
                    ;KEEP TRACK OF CURRENT GROUP
PUSHW RPP           ;SAVE REGISTER POINTER (1)
PUSH PPR            ;SAVE PAGE POINTER REGISTER (2)
                    ;SWITCH TO A NEW GROUP
SRP  #SCI_BANK      ;SET SCI PAGE AND BANK (3)
SPP  #SCI_PG
                    ;BODY OF THE INTERRUPT SERVICE ROUTINE
                    ;PERFORM CHARACTER
                    ;MANAGEMENT WITHIN
                    ;THE NEW ACTIVE
                    ;REGISTER BANK
                    ;RESTORE WHATEVER GROUP WAS ACTIVE
P0P PPR             ;RESTORE REGISTER POINTER
P0PW RPP            ;RESTORE PAGE POINTER REGISTER
IRET                ;RETURN FROM SUBROUTINE
```

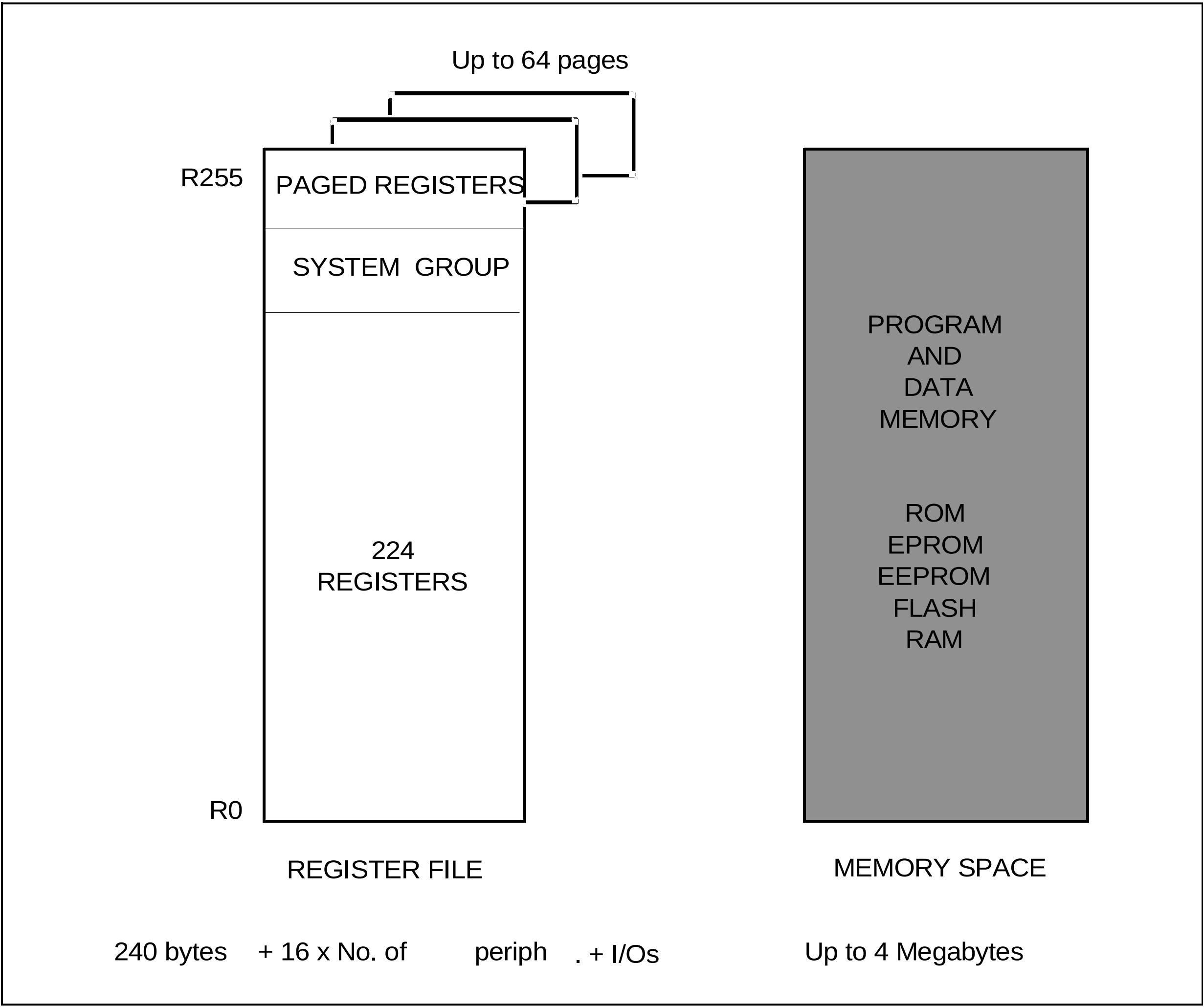
Supposing you could not switch working registers, you could have to push 16 bytes to the stack to ensure that the contents of the working area have been preserved, and pop them back before returning. Obviously the example above is more efficient, both in code and data memory size, and also in execution time.

1.4 MEMORY ORGANISATION AND MANAGEMENT

1.4.1 Memory description

The ST9 devices provides two different address spaces: the **Register File** and a single linear **Memory Space** accommodating both programs and data.

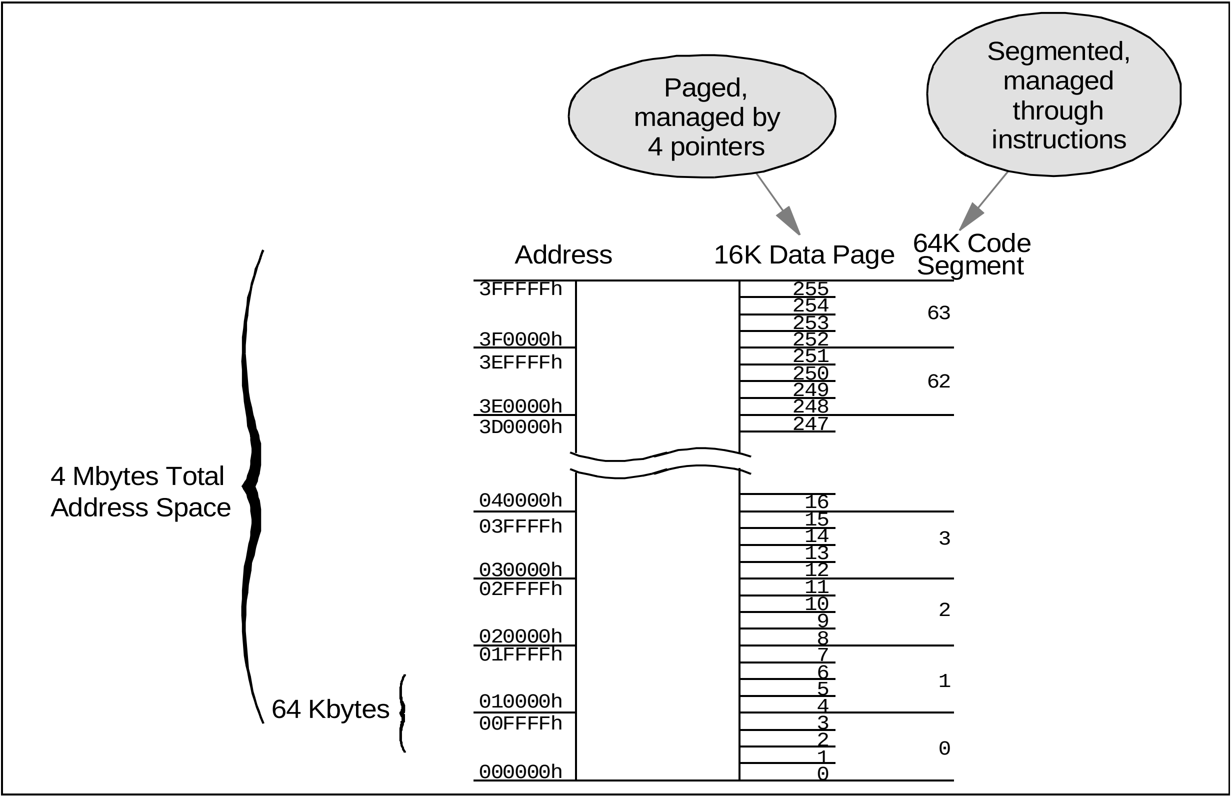
Figure 15. ST9 memory organization



The **Register File** draws its power from its size: 256 registers of which 224 are uncommitted, and from the fact that it can hold data pointers to data that reside in any of the two spaces.

All of the physically separate memory areas, including the internal ROM, internal RAM and external memory are mapped in the common address space which is the **Memory Space.** A total addressable memory space of 4 Mbytes is available. This address space is arranged as 64 segments of 64 Kbytes. Each segment is further subdivided into four pages of 16 Kbytes, as illustrated in Figure 16.

Figure 16. Memory Space organization



1.4.2 Memory Management Unit

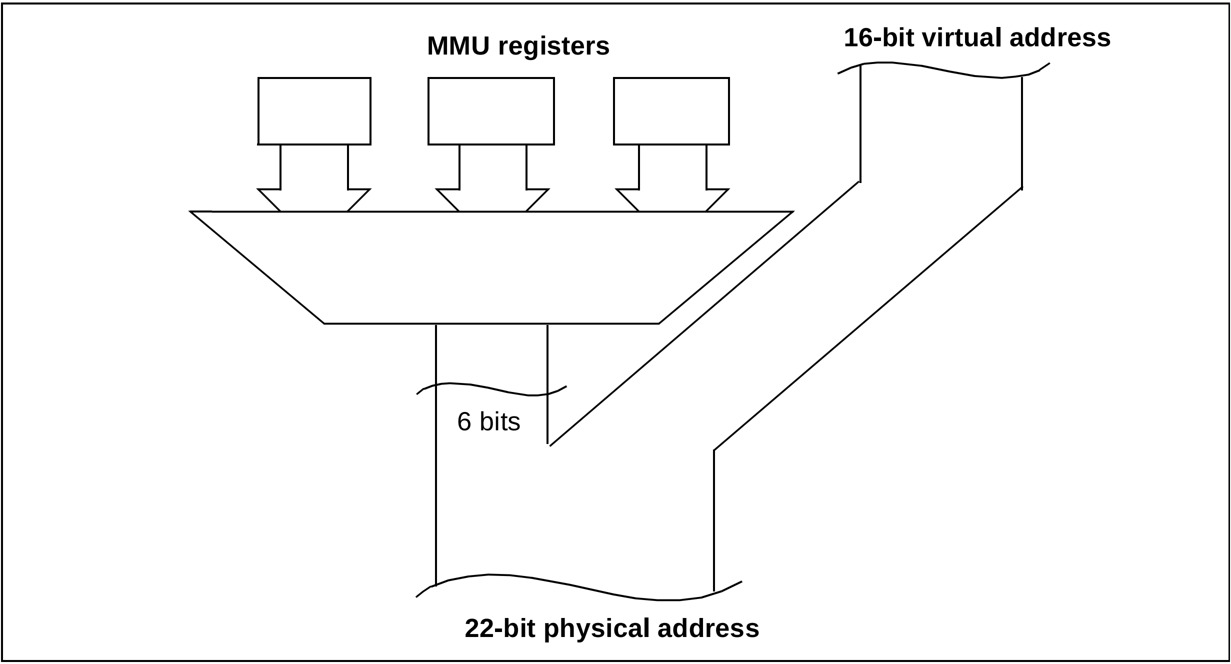
1.4.2.1 4 Mbytes of address space with the MMU

The ST9 Core includes a Memory Management Unit (MMU) which allows the addressing space to be extended to 4 Mbytes.

The MMU is controlled by 7 registers. These registers may be sub-divided into 2 main groups: a first group of four 8-bit registers (DPR0-3) used to extend the address during Data Memory access, and a second group of three 6-bit registers (CSR, DMASR, ISR) used to manage Program and Data Memory accesses during Code execution, Interrupts, and DMA service routines.

To manage 4 Mbytes of addressing space it is necessary to have 22 address bits. The MMU adds 6 bits to the usual 16-bit address, thus translating a 16-bit virtual address into a 22-bit physical address. There are 2 different ways to do this depending on the memory involved and on the operation being performed.

Figure 17. Addressing via MMU registers



1.4.2.2 Advantages of using the MMU

In typical microcontroller applications, less than 16 Kbytes of RAM are used, so just one of the four Data space pages is normally sufficient. It may be useful however to map part of the ROM to the data space if it contains strings, tables, bit maps, etc.

The Management Memory Unit lets interrupt service routines access the whole 4-Mbyte address space. The drawback is that the interrupt response time is slightly increased, because of the need to also save additional registers on the stack.

The MMU allows a DMA controller which uses two different registers for Program memory and Data memory accesses will always find its memory segment(s), no matter what segment changes the application has performed.

1.4.3 External memory interface

In the event of an application requiring more ROM space than available on-chip, or for easier program management and customization with external memory or peripherals, the ST9 microcontroller supports an external memory interface on some devices. The external memory interface provides the memory lines and timing and status control signals, plus enhanced features including programmable memory wait cycles, bus request/acknowledge cycles and shared memory bus access control.

The ST9 Memory Control Unit automatically recognizes if a memory location belongs to on-chip memory or not and works accordingly.

1.5 INTERRUPT MANAGEMENT

1.5.1 Interrupt management description

The ST9 devices respond to, and control peripheral events and external events through their interrupt channels. When such an event occurs, if previously enabled and according to a priority mechanism, the current program execution can be suspended to allow the ST9 to execute a specific response routine. If the event generates an interrupt request, the current program status is saved after the current instruction is completed and the CPU control passes to the Interrupt Service Routine.

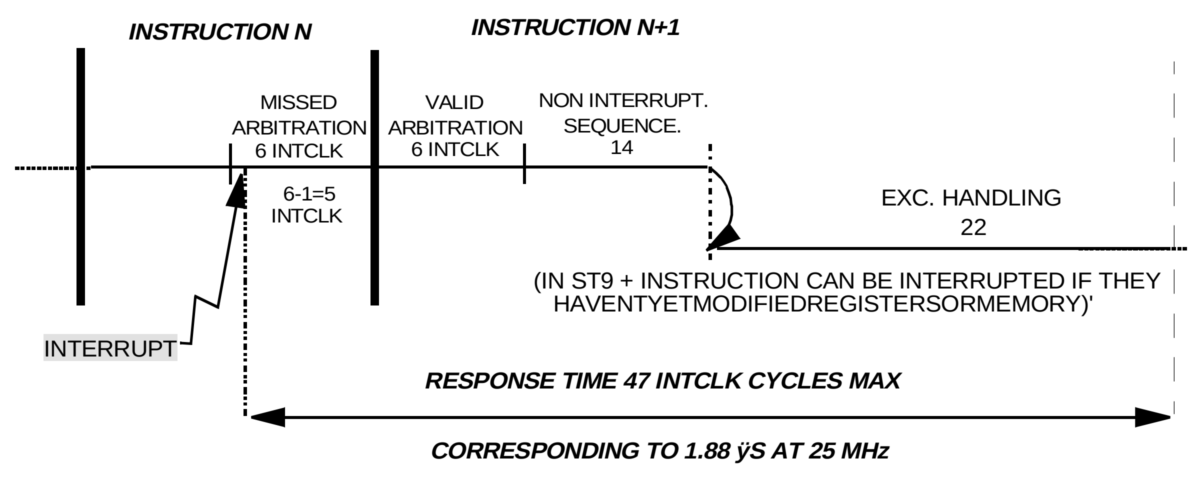
The ST9 CPU can receive requests from the following type of sources: on-chip peripherals, external pins, top level non maskable interrupt.

Up to eight external interrupt channels, with programmable input trigger edge, are available. In addition, a dedicated interrupt channel, set to the Top-level priority, can be assigned either to the external pin NMI (to provide a Non-Maskable-Interrupt) or to the Watchdog Timer. Interrupt service routines are addressed through a vector table mapped in Program Memory. This Interrupt vector table, up to 128 vectors, allows additional peripherals for more modularity.

1.5.2 A powerful system for real time applications

To achieve maximum performance, the ST9 family offers a powerful solution to the response requirements of real time systems with its advanced interrupt structure. The interrupt system allows you to handle various asynchronous events and to build very efficient programs with excellent interrupt response times. With a maximum interrupt response time of 1,88 µs, the ST9 family is particularly suited for real time applications.

Figure 18. Interrupt response time in the worst case, with MMU intersegment jump


INSTRUCTION N
INSTRUCTION N+1
MISSED ARBITRATION 6 INTCLK
VALID ARBITRATION 6 INTCLK
NON INTERRUPT. SEQUENCE. 14
6-1=5 INTCLK
EXC. HANDLING 22
(IN ST9 + INSTRUCTION CAN BE INTERRUPTED IF THEY HAVENTYETMODIFIEDREGISTERSORMEMORY)'
INTERRUPT
RESPONSE TIME 47 INTCLK CYCLES MAX
CORRESPONDING TO 1.88 ÿS AT 25 MHz

1.5.3 Interrupt vectors

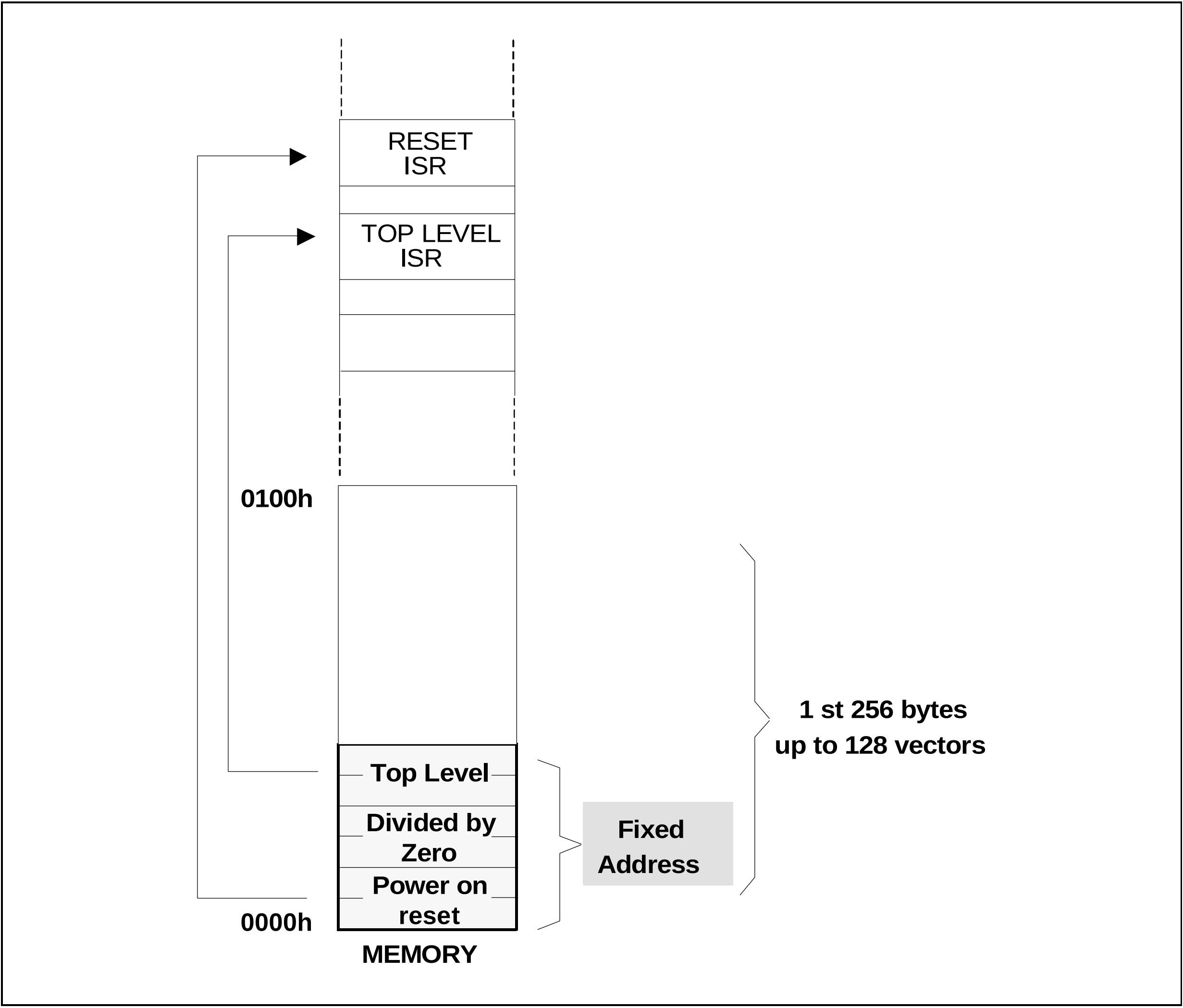
The ST9 implements an interrupt vectoring structure that allows the on-chip peripheral to automatically identify the location of the first instruction of the Interrupt Service Routine (ISR).

When the interrupt request is acknowledged, the peripheral interrupt module provides, through its Interrupt Vector Register (IVR), a vector to point into the vector table of locations containing the start addresses of the Interrupt Service Routines (defined by the programmer).

Each peripheral has a specific IVR mapped within its Register File pages.

The Interrupt Vector table, containing the list of the addresses of the Interrupt Service Routines, is located in the first 256 locations of the Program Memory (ROM).

Figure 19. Vector table organization



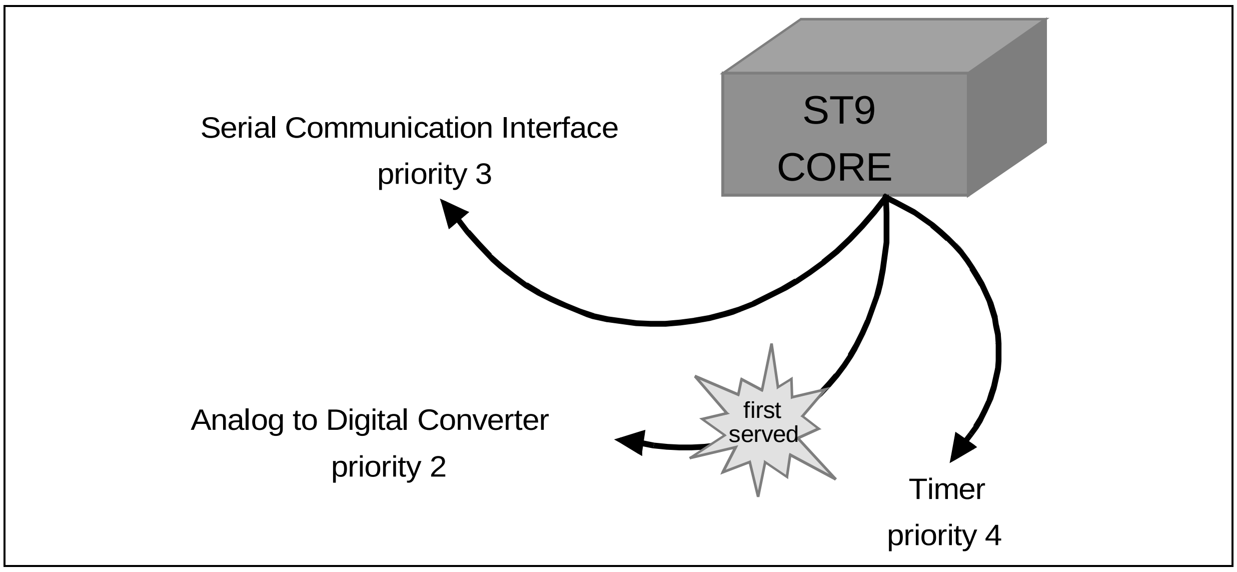
1.5.4 Interrupt priorities

In any microprocessor-based system, there is a trade-off between the computational power of the main program and the interrupt latency time. Expressed simply, the less the main program is disturbed, the sooner it finishes its job. In the other hand, the CPU often needs to serve interrupt requests generated by the peripherals as quickly as possible. A compromise must be found to give both enough power to the main program while still staying as responsive as possible to interrupts.

A typical microcontroller will leave the burden of this work almost entirely with the programmer. On the ST9, powerful interrupt management is available to significantly help the programmer.

The ST9 supports a fully programmable interrupt priority structure. Nine priority levels are available to define the channel priority relationship. Each channel has a PRiority Level (PRL), that defines its priority level among eight programmable levels for interrupt requests. The ninth level (Top Level Priority) is reserved for the internal Watchdog Timer or the external Non-Maskable-Interrupt. The on-chip peripheral channel and the eight external interrupt sources can be programmed within eight priority levels: level 7 has the lowest priority, level 0 has the highest priority.

Figure 20. Interrupt priorities for peripherals

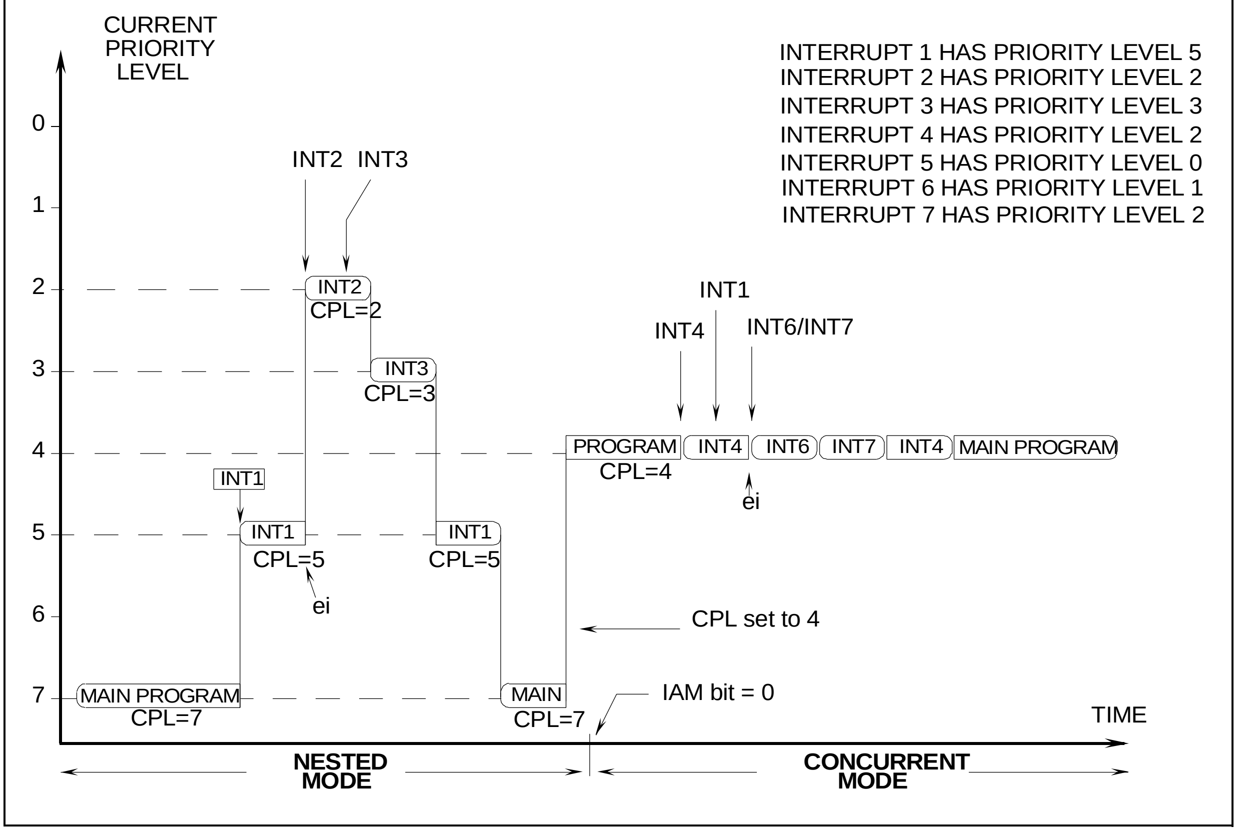


The priority mechanism is driven by the Current Priority Level parameter. At a given time, the part of the program being executed runs under a certain level. You can change the level by writing a different value in the Central Interrupt Control Register (CICR). You can assign a priority level to each interrupt source. At initialisation time, this value is written in one of the control registers specific to the corresponding peripheral.

When a peripheral requests an interrupt, the built-in interrupt controller compares the priority level of the interrupt request to the Current Priority Level. The interrupt is only acknowledged if its priority is greater than the Current Priority Level. This allows you to filter out interrupt requests according to their degree of importance or of urgency according to the current activity of the program. If several requests have the same priority level, an internal daisy chain, fixed for each ST9 device, defines the priority relationship within that level. The Non-Maskable Interrupt input (NMI) is hard wired with a higher priority than any level, and thus is acknowledge immediately in all circumstances.

ST9 provides two interrupt arbitration modes: Concurrent and Nested modes. Concurrent mode is the standard interrupt arbitration mode while Nested mode improves the effective interrupt response time when nested interrupts are required.

Figure 21. Example of nested and concurrent arbitration modes


CURRENT PRIORITY LEVEL
INTERRUPT 1 HAS PRIORITY LEVEL 5
INTERRUPT 2 HAS PRIORITY LEVEL 2
INTERRUPT 3 HAS PRIORITY LEVEL 3
INTERRUPT 4 HAS PRIORITY LEVEL 2
INTERRUPT 5 HAS PRIORITY LEVEL 0
INTERRUPT 6 HAS PRIORITY LEVEL 1
INTERRUPT 7 HAS PRIORITY LEVEL 2
0
1
2
3
4
5
6
7
INT2
INT3
INT2
CPL=2
INT3
CPL=3
INT1
INT1
CPL=5
ei
INT1
CPL=5
MAIN PROGRAM
CPL=7
MAIN
CPL=7
INT1
INT4
INT6/INT7
PROGRAM
CPL=4
INT4
INT6
INT7
INT4
MAIN PROGRAM
ei
CPL set to 4
IAM bit = 0
TIME
NESTED MODE
CONCURRENT MODE

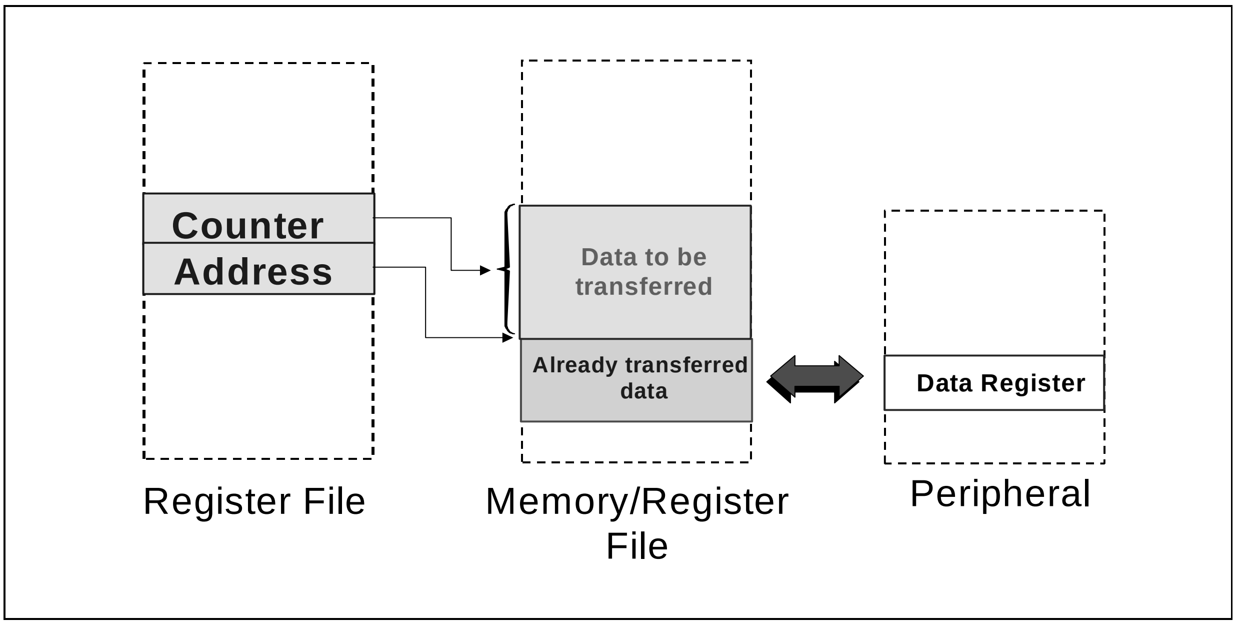
1.6 DIRECT MEMORY ACCESS

1.6.1 General description

Direct Memory Access capability is a feature seldom found in 8-bit MCUs. It allows the micro-controller to handle input/output data flow without using core instruction cycles.This feature allows you to boost the system performance.

Direct Memory Access consists of a transfer between memory and a peripheral, in either direction. DMA transfers are implemented by means of a DMA controller which stops the instruction execution and perform the data transfer. The core can then restart the instruction execution at the end of the transfer. The DMA controllers acts as part of the peripherals. They use an indirect addressing mechanism to DMA Pointers (address pointers which contain the address of the DMA table) and Counter Registers (registers which contain the number of bytes to transfer) stored in the Register File.

Figure 22. DMA overview



Assuming the peripheral is configured to handle externally supplied data or to provide data to external circuits, two steps are needed for a transfer to occur:

– The transfer must be requested by some event or condition

– A mechanism must handle the reading of the data from one part and the writing to the other part

The term DMA transfer represents the transfer of a single byte (or word) of data. Usually, more than one byte is transferred and the transfer occurs in bursts. Thus, a third step is involved:

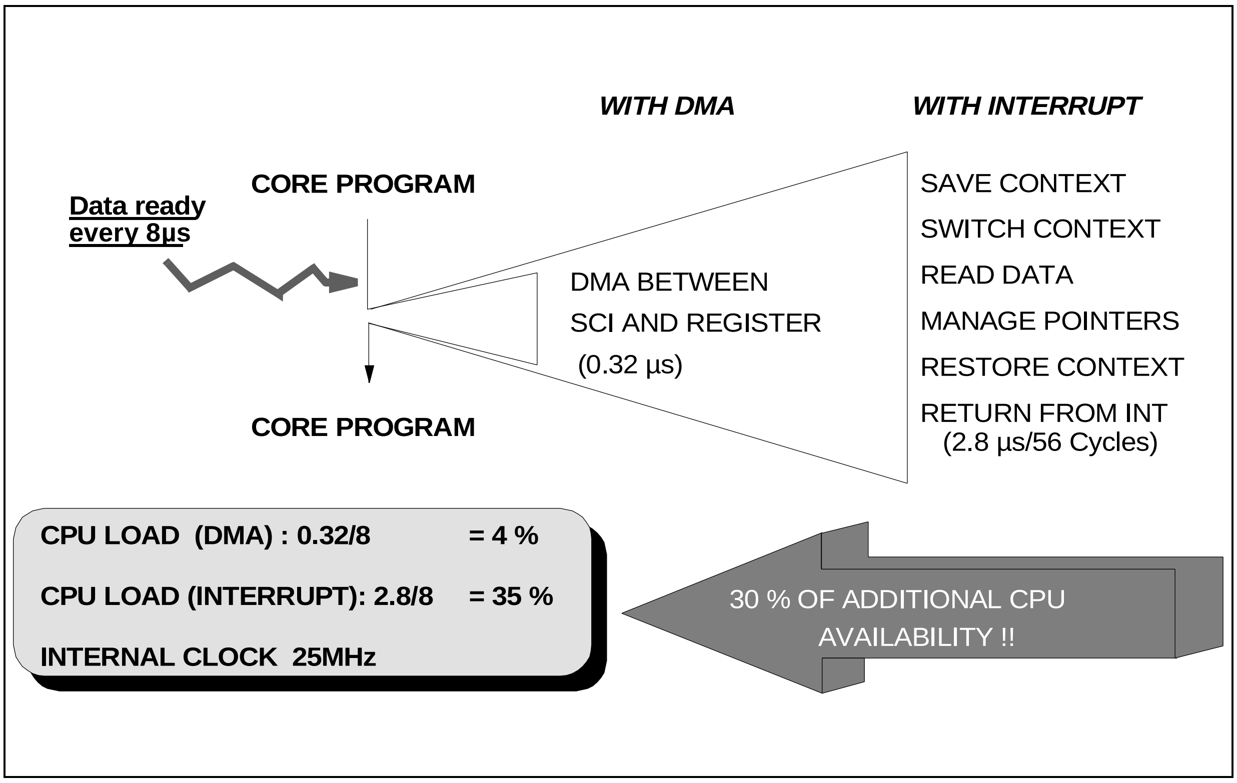
– A mechanism that counts the transfers and stops them when the count is finished

1.6.2 High speed system performance

Once properly initialised, the DMA controller allows peripherals to exchange data either with memory or the register file, with no use of core resources other than the memory cycles stolen in order to transfer the data. The maximum number of bytes that can be transferred per transaction by each DMA channel is 222 when accessing the Register File, or 65536 (64 Kbytes) when accessing Memory. The time needed to transfer information between memory or registers and peripherals is reduced to less than 1 µs. A DMA transfer with the Register file requires 8 CPUCLK cycles or 0.32 µs at 25 MHz and a DMA transfer with memory requires 16 CPUCLK cycles or 0.64 µs, plus any required wait states.

DMA can dramatically improve system performance in communications-intensive applications. For example, reading a character every 8 µs from the SCI received at 1Mbit per second requires 35% of the CPU time just to manage the interrupt routine. With DMA, up to 30% of additional CPU time is available for other control functions

Figure 23. Example of the benefit of DMA when reading a character from the SCI


WITH DMA
WITH INTERRUPT
Data ready every 8µs
CORE PROGRAM
DMA BETWEEN SCI AND REGISTER (0.32 µs)
CORE PROGRAM
SAVE CONTEXT
SWITCH CONTEXT
READ DATA
MANAGE POINTERS
RESTORE CONTEXT
RETURN FROM INT (2.8 µs/56 Cycles)
CPU LOAD (DMA) : 0.32/8 = 4 %
CPU LOAD (INTERRUPT): 2.8/8 = 35 %
INTERNAL CLOCK 25MHz
30 % OF ADDITIONAL CPU AVAILABILITY !!

1.6.3 DMA priority levels

The 8 priority levels used for interrupts are also used to prioritize DMA requests, which are arbitrated in the same arbitration phase as interrupt requests. If the event requires a DMA transaction, this will take place at the end of the current instruction execution. When an interrupt and a DMA request occur simultaneously, on the same priority level, the DMA request is serviced before the interrupt because DMA is faster (only a few cycles) compared to interrupt.

DMA requests are serviced if their priority level is equal to or higher than the Current Priority Level. DMA transactions are not interruptable. However DMA requests are not acknowledged during a top level interrupt routine.

An interrupt priority request must be higher than the Current Priority Level (CPL) value in order to be acknowledged, whereas, for a DMA transaction request, it must be equal to or higher than the CPL value in order to be executed. Thus, only DMA transaction requests can be acknowledged when the CPL = 7.

DMA requests do not modify the CPL value, since a DMA transaction is not interruptable.

1.6.4 SWAP mode

An additional DMA feature which may be found on some peripherals (i.e the MultiFunction Timer) is Swap mode. This feature allows transfer from two DMA tables alternatively. All the DMA descriptors in the Register File are thus duplicated. Two DMA transaction counters and two DMA address pointers allow the definition of two fully independent tables (they only have to belong to the same space, Register File or Memory). The DMA transaction is programmed to start on one of the two tables (say table 0) and, at the end of the block, the DMA controller automatically swaps to the other table (table 1) by pointing to the other DMA descriptors. In this case, the DMA mask (DM bit) control bit is not cleared, but the End Of Block interrupt request is generated to allow the optional updating of the first data table (table 0).

As long as swap mode is enabled, the DMA controller will continue to swap between DMA Table 0 and DMA Table 1.

1.7 INSTRUCTIONS SET AND ADDRESSING MODES

1.7.1 Overview of Instruction Set and Addressing Modes

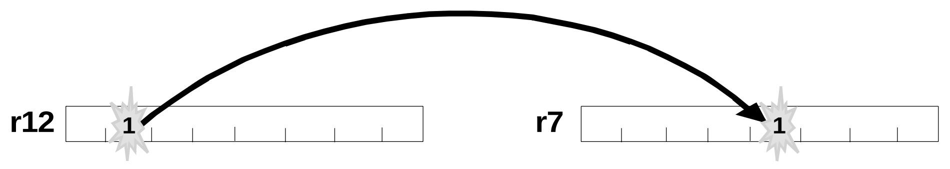
The ST9 is referred to as an 8/16-bit microcontroller. This means that although the size of the internal registers and the width of the data bus are 8 bits, the instruction set includes instructions that handle a pair of registers or a pair of bytes in memory at once. These instructions represent roughly one half of the total instructions, which means that the ST9 can be programmed with the same ease as if it were a full 16-bit machine. This is why it is so well suited for C programming.

The ST9 instruction set consists of 94 instruction types which can be divided into eight groups: Load (two operands), Arithmetic & logic (two operands), Arithmetic Logic and Shift (one operand), Stack (one or two operands), Multiply & Divide (two or three operands), Boolean (one or two operands), Program Control (zero to three operands), Miscellaneous (zero to two operands). The ST9 can operate with a wide range of data lengths from single bits, 4-bit nibbles which can be in the form of Binary Coded Decimal (BCD) digits, 8-bit bytes, and 16-bit words. A particularly notable feature is the comprehensive “Any Bit, Any Register” (ABAR) addressing capability of the Boolean instructions.

Figure 24. Example of direct bit addressing mode

bld r7.3, r12.6

this instruction loads the bit 6 of the working register 12 in bit 3 of working register 7.



Powerful addressing modes such as indirect, indirect with increment or decrement, indexed shorten the code needed to access data even in complex structures or arrays. They also facilitate access to local variables created on the stack on entering functions. This full set of addressing modes allows simple access to complex data arrays of structures (high level language) and less pointer calculation using ALU.

Working registers, that benefit from the most powerful instructions and addressing modes, are heavily used by the compiler. In fact, the GNU9 compiler does not always translate the source code. There are optimization schemes that save execution time and/or memory by judiciously allocating the working registers, so that in many cases arguments are not pushed to the stack but merely to an available working register.

1.7.2 High speed computation

The relocatable working register bank of either 16x8-bit or 8x16-bit register locations supports the fastest available instructions. With the extensive set of instructions and addressing modes, the ST9 is thus able to handle complex calculations (for example a large array of data in the memory), much faster than any 8-bit accumulator machine.

For example, the complex instruction below, which can be used to search a character in a table, has the following characteristics at 25 MHz:

Figure 25. Example of reduced code size and fast execution time capabilities

OPCODE	EXECUTION TIME		CODE SIZE
	No Jump	Jump	
CPJTI r, (rr), N	0.56 µs	0.64 µs	3 BYTES

1.7.3 Effective high level language support

In addition, the optimizer of the ST9 GNU C Compiler make full use of the available set of instructions and addressing modes, and optimizes the register allocation for function parameter within the working register bank.

The compiler generates very fast and compact code, even from sophisticated data manipulation routines.

Critical parts of the software can be further optimized by interleaving C statements with assembly language to benefit from the effectiveness of the instruction set (for example, the implementation of a character search in a table with a single opcode).

1.7.4 Addressing modes

The ST9 offers a wide variety of established and new addressing modes and combinations to facilitate full and rapid access to the address spaces while reducing program length. The available addressing modes are shown in Figure 26.

Figure 26. Addressing Modes

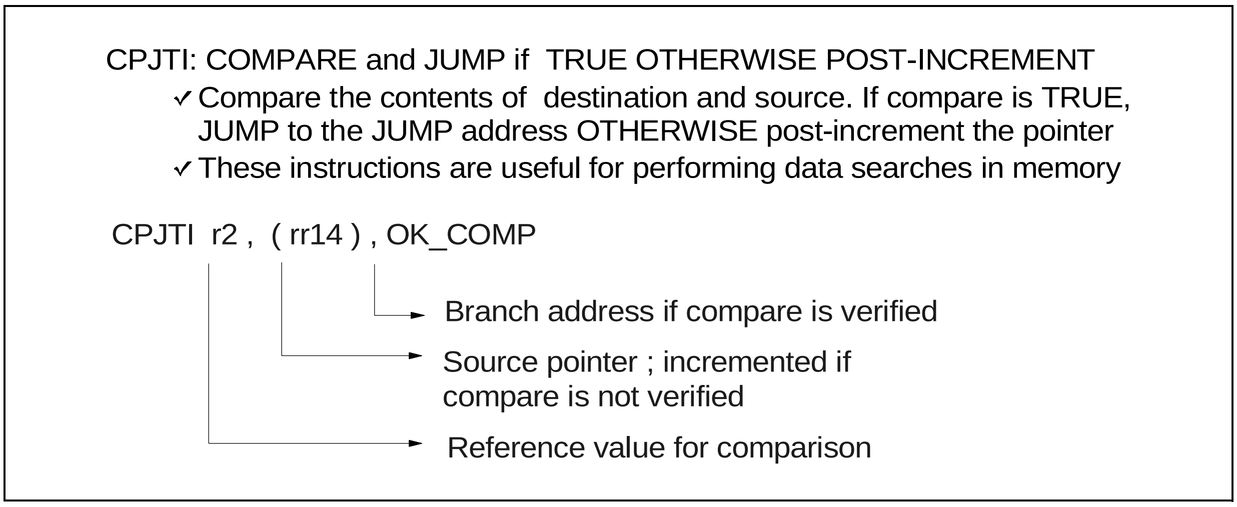
Operand is in	Addressing Mode	Destination Location	Notation
Instruction	Immediate	Byte Word	#N #NN
Register File	Direct	Byte Word	r rr
	Indirect	Byte/Word	(r)
	Indexed	Byte/Word	N(r)
	Indirect Post-Increment	Byte	(r)+
Program or Data Memory	Direct	Byte/Word	NN
	Indirect	Byte/Word	(rr)
	Indirect Post-Increment	Byte/Word	(rr)+
	Indirect Pre-Decrement	Byte/Word	-(rr)
	Short Indexed	Byte/Word	N(rr)
	Long Indexed	Byte/Word	NN(rr)
	Register Indexed	Byte/Word	rr(rr)
Any bit of any working register	Direct	Bit	r.b
Any bit in program or data memory	Indirect	Bit	(rr).b

Single operand arithmetic, logic and shift byte instructions have direct register and indirect register addressing modes. For a full list of the possible combinations for each instruction type, please refer to the ST9 Programming Manual.

1.7.5 A good alternative to more costly 16-bit MCUs

The wide range of instructions eases use of the register file and address spaces, reducing operation times, while the register pointer mechanism gives an unmatched code efficiency.

Figure 27. Example of a powerful instruction


CPJTI: COMPARE and JUMP if TRUE OTHERWISE POST-INCREMENT

✓ Compare the contents of destination and source. If compare is TRUE, JUMP to the JUMP address OTHERWISE post-increment the pointer

✓ These instructions are useful for performing data searches in memory

```
CPJTI  r2 ,  ( rr14 ) , OK_COMP
```

Branch address if compare is verified
Source pointer ; incremented if compare is not verified
Reference value for comparison

This instruction set facilitates large program and data handling through the MMU, and it improves the performance and code density of C function calls. The 8 and 16-bit data manipulation offered by these instructions gives a cost effective alternative to 16-bit MCUs.

Figure 28. ST9 vs 68HC12 on a 16-bit memory-memory transfer

MICROCONTROLLER	ARCHITECTURE	INTERNAL CLOCK SPEED	CODE	BYTES	EXECUTION TIME
ST9	8/16-BIT REGISTER MACHINE	**25 MHz**	ldw (rr), (rr) (16-bit indirect)	2	0.64 µs
68HC12	16-BIT ACCUMULATOR MACHINE	**8 MHz**	ldd [opr16, x] std [opr16, x] (16-bit indexed-indirect)	4 4 8	750 ns 625 ns 1.37 µs

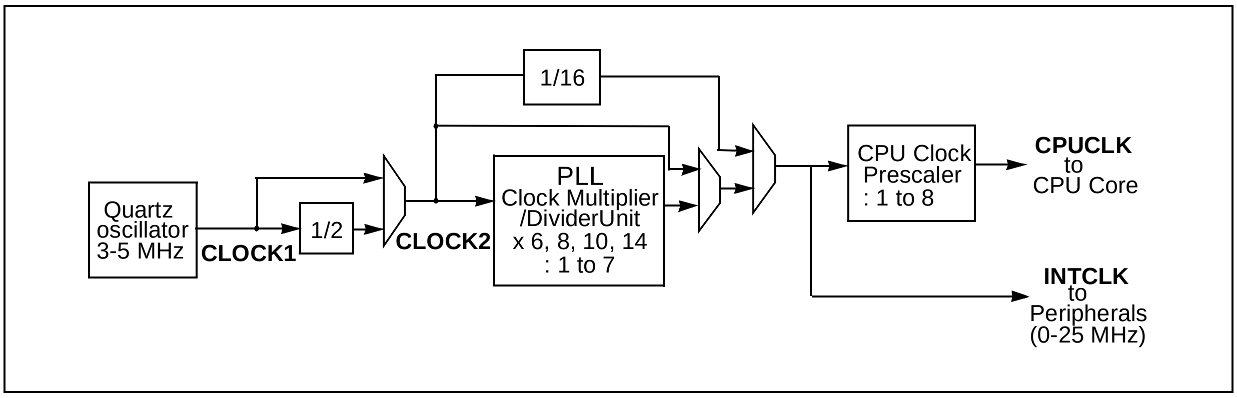
1.8 ST9 OPERATING MODES

To provide for real time management capability, the ST9 is designed to operate at a higher internal clock speed compared to traditional 8-bit architecture.

The advantage of the PLL is that, a low cost 4 MHz crystal or ceramic resonator can be used to generate an internal operating frequency of up to 25 MHz.

To optimize the performance versus power consumption of the application, ST9 devices support a range of operating modes that can be dynamically selected to meet the performance and functionality requirements of the application at a given moment.

Figure 29. Clock control unit simplified block diagram



RUN MODE: This is the full speed execution mode with CPU and peripherals running at the maximum clock speed delivered by the Phase Locked Loop (PLL) of the Clock Control Unit (CCU), or any frequency obtained from the appropriate programming of the control registers.

SLOW MODE: Power consumption can be significantly reduced by running the CPU and the peripherals at reduced clock speed using the CPU Prescaler and CCU Clock Divider.

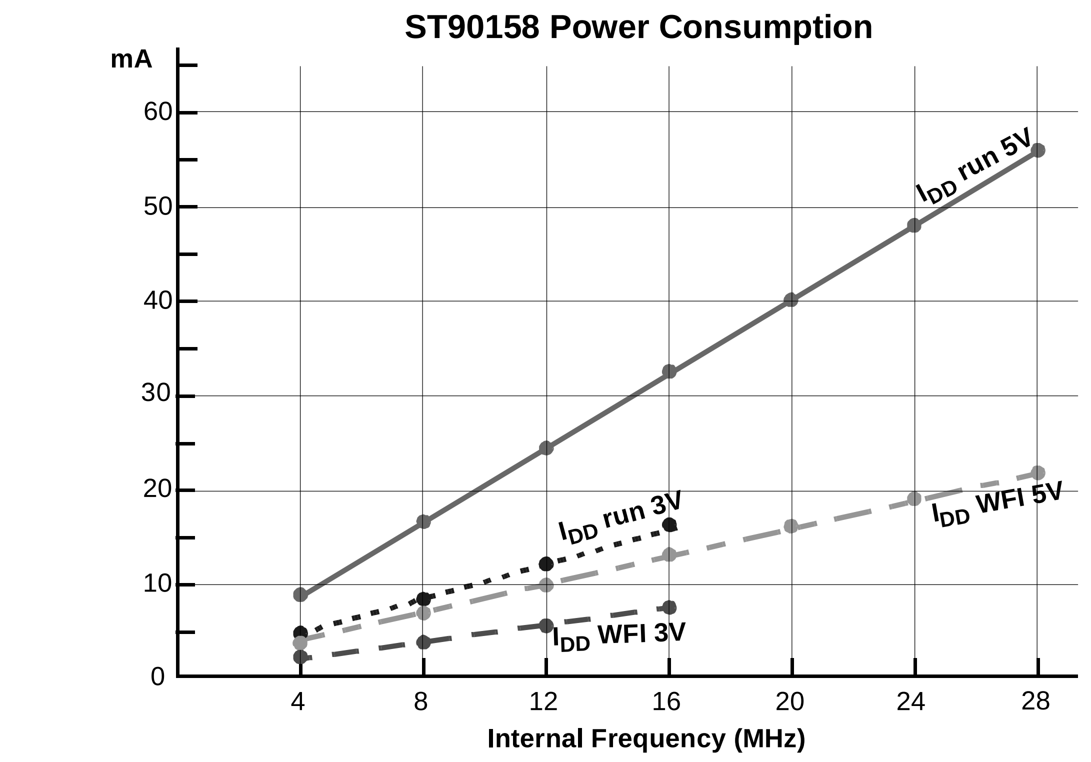
In this mode, using a 4MHz Crystal, the ST9 operates down to 125KHz and draws down to **1mA** typical.

WAIT FOR INTERRUPT MODE: The Wait For Interrupt (WFI) instruction suspends program execution until an interrupt request is acknowledged. During WFI, the CPU clock is halted while the peripheral and interrupt controller keep running at a frequency depending on the CCU programming.

HALT MODE: When executing the HALT instruction and if the Watchdog Timer is not programmed as a watchdog, the CPU and its peripherals stop operation and the I/O ports enter high impedance mode. A typical power consumption of less than 1µA is achieved

An external Reset is necessary to exit from Halt mode.

Figure 30. Typical power consumption of the ST90158 In RUN and Wait For Interrupt



2 PERIPHERALS AND I/O PORTS

The ST9 family devices include a range of powerful peripherals and I/O Ports.

Table 3. Main standard peripherals

Name	Function
Multi-Function Timer	All counting and timing functions. Includes auto-reload on condition, interrupt generation, DMA transfer, two inputs for frequency measurement or pulse counting, two outputs that can change on condition. Conditions include: overflow/underflow, comparison with one or two compare registers. Capture registers allow recording transitions on inputs with their time of occurrence.
Serial Communication Interface	Asynchronous transfer with either internal bit-rate generation or an external clock. Parity generation/detection. Address recognition feature that can request an interrupt on match of an input character. DMA transfer.
Serial Peripheral Interface	Serial input or output register, with internal or external clock. Intended for I/O expansion, or synchronous serial external device such as serial EEPROM.
Watchdog Timer	Can be used either as a watchdog or as a timer with input and output capable of pulse counting or waveform generation.
Input/Output port	Parallel input/output port. Each bit individually configurable as input, output, bi-directional, or alternate function. Inputs can be high impedance or with pull-up, CMOS or TTL-level. Outputs can have open drain or push-pull configuration.
Analog to Digital Converter	Eight-bit analog to digital converter. One to eight channels can be converted in a row. On each of two of the eight channels, an Analog Watchdog function defines upper and lower thresholds. When exceeded, an interrupt is generated.

Most peripherals of the ST9 have sufficient built-in intelligence to be able to perform even complex jobs on their own, freeing the core almost entirely from basic peripheral management. They include powerful control and data management functions that drastically reduce CPU overhead. This capability to work independently from the CPU allows the core to be fully utilized for the most complex microprocessing tasks.

A typical example is the Analog-to-Digital Converter (A/D) that automatically monitors two channels and interrupts the CPU when one of the inputs is outside a predefined voltage range (for example the speed variation of a motor or a drop in the power supply).

2.1 FLEXIBLE I/O PORTS

The basic functionality of the parallel input-outputs is very straightforward. Once initialised, they appear as a register that can be written or read. However in many cases, direct byte-wide I/O is not sufficient. Bit-oriented I/O is often what is used in microcontroller systems.

A powerful feature of the ST9 devices is that you can address the eight bits of each port individually to provide digital and analog input/output, or to connect input/output signals to the on-chip peripherals as alternate pin functions. The ST9 also provides the external pins of the other peripherals (timers, UARTs, etc.) by diverting some bits from the parallel I/O ports. The flexibility of the ST9 I/O pins allow designers to match the MCU to the application, and not the application to the MCU.

The ST9 family devices have up to 10 parallel I/O ports which have an additional very flexible feature. You can independently configure each bit as:

- an input with two variants (TTL or CMOS),
- an output with also two variants (open-drain or push-pull),
- a bidirectional port with either a weak pull-up or an open-drain output side,
- an alternate function output (that is the output pin of an internal peripheral) with also either open-drain or push-pull output driver.

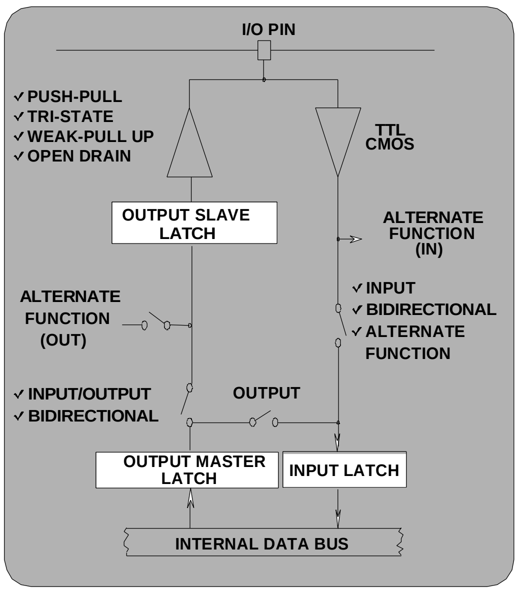
Figure 31. ST9 I/O ports configurations

INPUT	Analog Input CMOS or TTL level Schmitt Trigger
OUTPUT	Push-pull Open Drain
BIDIRECTIONAL	Weak Pull-up Open Drain
ALTERNATE FUNCTION	Push-pull Open Drain

CMOS: Vil (max) : 0.3 Vcc Vih (min) = 0.7 Vcc

TTL: Vil (max) : 0.7 Volt Vih (min) = 2.0 Volt

Vil input low level Vih input high level



For some peripherals (e.g. ADC), the port that provides the input pins has a special alternate function mode. This mode disconnects the input buffer from the pin and shorts the buffer input to the ground. The output buffer is put in high-impedance mode. The pin is permanently connected to the input of the peripheral, thus allowing its voltage to be read at any time.

Each port is associated with a data register and three control registers. These define the port configuration and allow dynamic configuration changes during program execution. Port data and control registers are mapped in the Register File and are treated just like any other general purpose register. There are no special instructions for port manipulation: any instruction that can address a register, can address the ports. Data can be directly accessed in the port register, without passing through other memory or “accumulator” locations.

2.2 TIMERS

The timer or timing system makes it possible to measure and time external and internal events without the need to do this with time critical software loops.

2.2.1 Standard Timer (STIM)

The standard timer includes a programmable 16-bit down counter and an associated 8-bit prescaler with Single and Continuous counting modes capability. It uses an input pin (STIN) and an output pin (STOUT). These pins, when available, may be independent pins or connected as alternate functions of an I/O port bit.

STIN can be used in one of four programmable input modes: event counter, gated external input mode, triggerable input mode, retriggerable input mode.

STOUT can be used to generate a Square Wave or Pulse Width Modulated signal.

The input clock to the prescaler can be driven either by an internal clock equal to INTCLK divided by 4, or by CLOCK 2 derived directly from the external oscillator, divided by 64 or 128, thus providing a stable time reference independent from the PLL programming.

The standard timer end of count condition is able to generate an interrupt which is connected to one of the external interrupt channels.

Table 4. Main features of the STIM

Functional modes	- Single count down - Continuous count down
Timer architecture	- One 8-bit Prescaler - One 16-bit Counter - One Control Logic Register
Timer accuracy	- Min.: 160 ns at 25 MHz - Max.: 2.68 s at 25 MHz
Timer input clock	- INTCLK/4 - External input clock - Max. = 25/4 MHz
Timer input pin	- One configurable input pin - Not on all ST9
Input modes	- External clock - Gated input - Retriggerable input - Triggerable input
Timer output pin	- One output pin configurable as AF or IO - Not on all ST9
Output modes	- Square wave generation - PWM
Interrupts	- Interrupt on INTA1 external interrupt channel

Four registers are used to control the standard timer. You can write in timer registers at any time while the timer is running. A Debugger option allows you to stop the timer during the Emulation Trap.

2.2.2 Watchdog Timer (WDT)

The Watchdog Timer is similar to the standard timer when working in timer mode. When watchdog mode is started, only reset can exit this mode.

The main features are almost the same as for the standard timer. Timer registers can be written at any time even if the timer is running. The counter is also read at any time but the new value written to the counter is taken into account only at the timer start or at the EOC.

Table 5. Main features of the WDT

Functional modes	- Watchdog - Normal Timer - Single count down - Continuous count down
Timer architecture	- One 8-bit Prescaler - One 16-bit Counter - One Control Logic Register
Timer accuracy	- Min.: 250 ns at 16 MHz - Max.: 4.19 s at 16 MHz
Timer input clock	- INTCLK/4 - External input clock - If available
Timer input pin	- One configurable input pin - If available
Input modes	- External clock - Gated input - Retriggerable input - Triggerable input
Timer output pin	- One output pin configurable as AF or IO
Output modes	- Square wave generation - PWM (software is needed)
Interrupts	- End of count interrupt - Top level interrupt - Watchdog reset
Watchdog mode selection	- Software - Hardware fixed (metal option) - Only on certain devices - By external input pin

The watchdog timer can be used to:

– Generate periodic interrupts

– Measure input signal pulse widths

– Request an interrupt after a set number of events

– Generate an output signal waveform

– Act as a watchdog timer to monitor system integrity

The watchdog timer provides a means of graceful recovery from a system problem. This could be a software fault, usually generated by external interference or by unforeseen logical conditions, or a hardware problem that prevents the program from operating correctly. This fault causes the application program to abandon its normal sequence of operation.

If the program fails to reset the watchdog at some predetermined interval, a hardware reset will be initiated. The bug may still exist, but at least the system has a way to recover. This is especially useful for unattended systems.

2.2.3 Multifunction timer (MFT)

The Multi-Function Timer is the most powerful of the ST9 on-chip peripherals. It offers powerful timing capabilities and features 12 operating modes, including automatic PWM generation and frequency measurement. This allows the ST9 devices to cover most application timing requirements.

The MFT comprises a 16-bit up/down counter driven by an 8-bit programmable prescaler. The input clock may be INTCLK/3 or an external source. The timer features two 16-bit comparison registers, and two 16-bit capture/load/reload registers. Two input pins and two alternate function output pins are available and independently configurable.

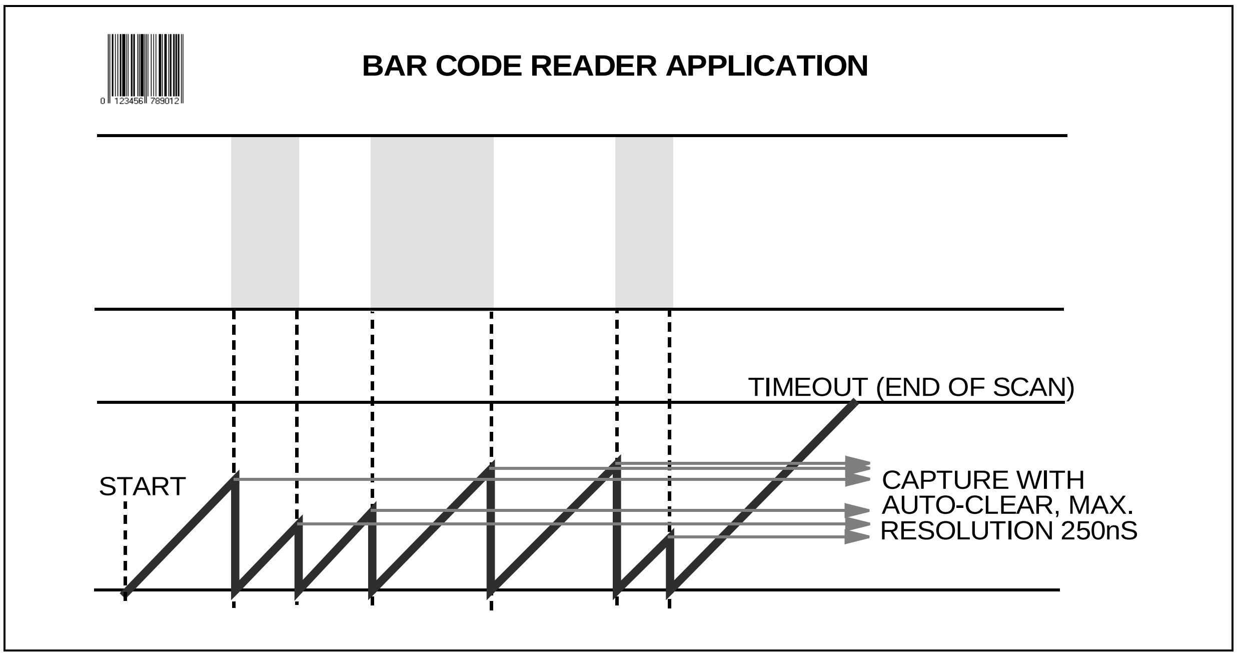
Two input pins, programmable as external clock, gate or trigger, allow 16 modes of operation, including autodiscrimination of the direction of externally generated signals.

Pulse Width Generation can easily be implemented, using the overflow/underflow signal and the two 16-bit comparison registers, each of them able to independently set, reset, toggle or ignore two output bits.

The Multi-Function Timer outputs may also generate interrupts for system scheduling, and trigger DMA transactions of a data byte to or from a data table in memory.

When two timers are present in an ST9 device, a combined operating mode is available.

Figure 32. An example of the ST9 Multi-Function Timer capabilities

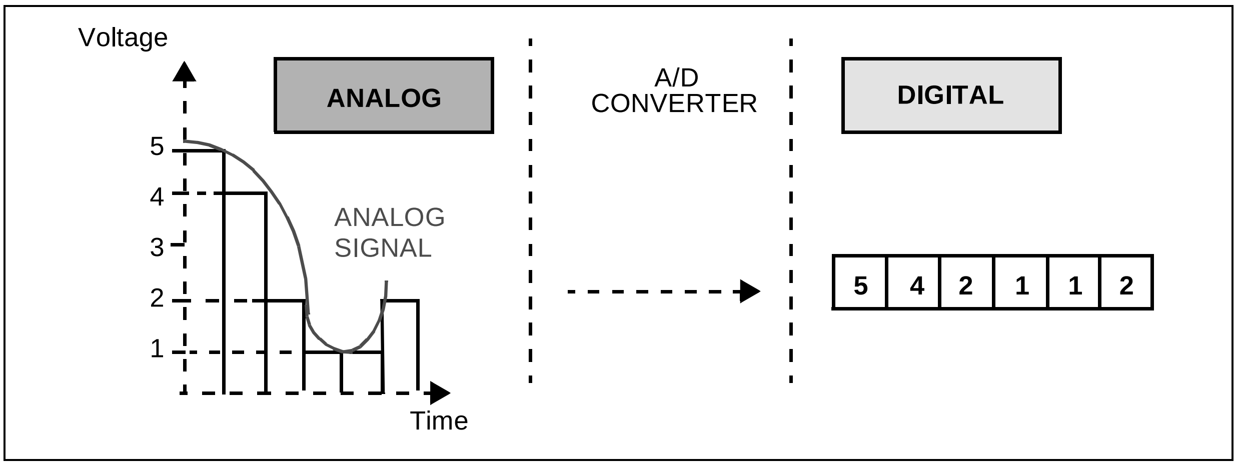


With the MFT, a complex application such as a bar code reader can be implemented with no CPU intervention by using the DMA and the timer autoclear mode in order to measure sophisticated waveforms.

2.3 FAST ANALOG TO DIGITAL CONVERTER (ADC)

The Analog to Digital Converter converts an external analog signal (typically relative to voltage) applied to one of eight inputs into a digital representation using an 8-bit successive approximation Analog to Digital Converter. The ST9 devices with this feature can be used for instrumentation, environmental data logging, or any application that lives in analog world.

Figure 33. Analog to Digital Converter



Two type of ADC exist in the ST9 family: a simple ADC and a more sophisticated one (ADC8). Only the last one is described here.

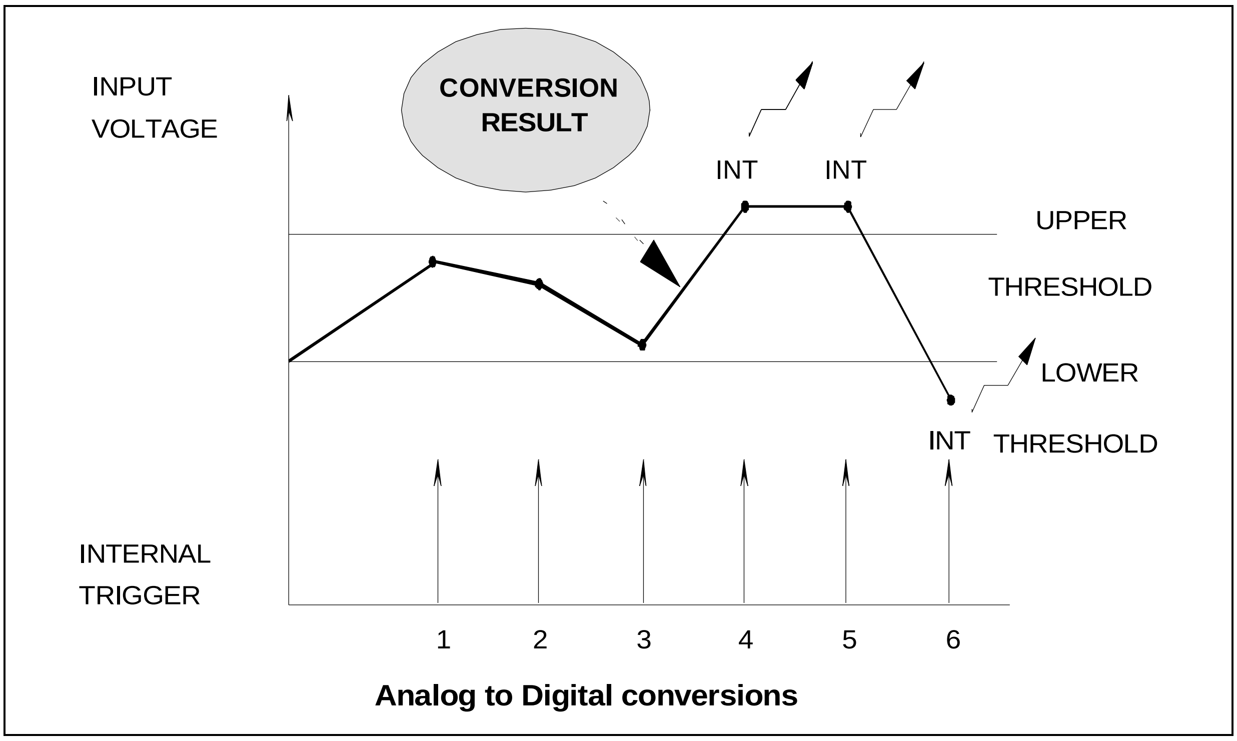
The 8-channel Analog to Digital Converter (ADC8) comprises an input multiplex channel selector feeding a successive approximation converter. Conversion requires 138 INTCLK cycles (of which 87,5 are required for sampling), conversion time is thus a function of the INTCLK frequency; for instance, for a 24MHz clock rate, conversion of the selected channel requires 5,75µs. This time includes the 3,64µs required by the built-in Sample and Hold circuitry, which minimizes the need for external components and allows quick sampling of the signal to minimise warping and conversion error. Conversion resolution is 8 bits, with ±1/2 LSB maximum non-linearity error between V_{SS} and the analog V_{DD} reference.

The converter uses a fully differential analog input configuration for the best noise immunity and precision performance. Two separate supply references are provided to ensure the best possible supply noise rejection and to allow the use of analog reference voltages lower than the digital V_{DD} supply. In fact, the converted digital value, is referred to the analog reference voltage which determines the full scale converted value. Naturally, Analog and Digital VSS MUST be common.

Up to 8 multiplexed Analog Inputs are available, depending on the ST9 device type. A group of signals can be converted sequentially by simply programming the starting address of the first analog channel to be converted and using the AUTOSCAN feature.

Two Analog Watchdogs channels are provided, allowing continuous hardware monitoring of two input channels. An interrupt request is generated whenever the converted value of either of these two analog inputs is outside the upper or lower programmed threshold values, as in the example below. The comparison result is stored in a dedicated register. This gives you fast analog data acquisition with minimum CPU intervention.

Figure 34. Analog watchdog for monitoring voltage min./tax levels



Single and continuous conversion modes are available. Conversion may be triggered by an external signal or, internally, by the Multifunction Timer.

A Power-Down programmable bit allows the ADC to be set in idle mode to reduce the power consumption of theST9.

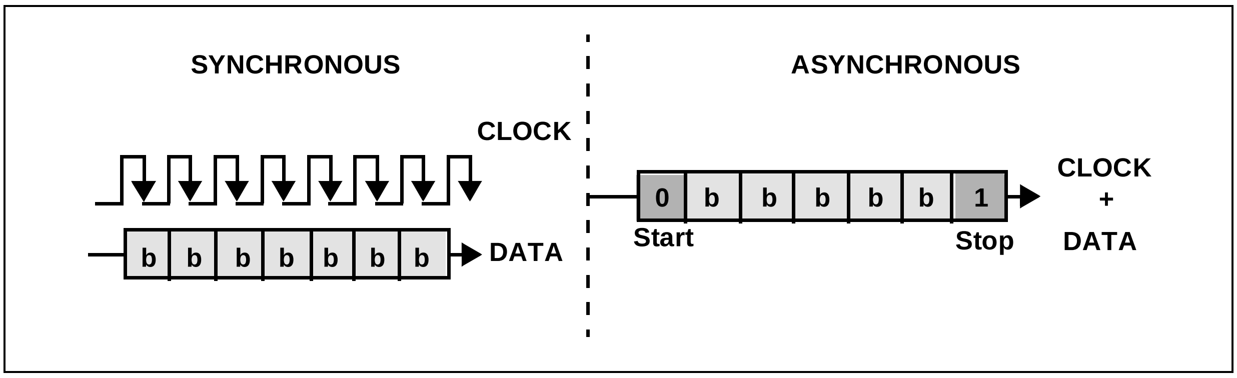
The ADC’s Interrupt Unit provides two maskable channels (Analog Watchdog and End of Conversion) with a hardware fixed priority, and up to 7 programmable priority levels.

2.4 SERIAL INTERFACE

Serial interfaces are used to exchange data with the external world. ST9 microcontrollers have both asynchronous and synchronous on-chip communications peripherals. The asynchronous interface is called Serial Communication Interface (SCI) and the synchronous interface is called Serial Peripheral Interface (SPI). A typical SCI application is for connecting a PC for debugging purposes while a typical SPI application is for connecting an external EEPROM.

A synchronous bus includes a separate line for the clock signal which simplifies the transmitter and receiver but is more susceptible to noise when used over long distances. With an asynchronous bus the transmitter and receiver clocks are independent, and a resynchronization is performed for each byte at the start bit.

Figure 35. Synchronous and asynchronous communications



2.4.1 Universal Serial Peripheral Interface (SPI)

The Serial Peripheral Interface is a synchronous input-output port that you can configure in various modes, including S-bus. It has many uses, of which two are: interfacing with serial-access EEPROMs, and interfacing with a liquid-crystal display.

The ST9's universal Serial Peripheral Interface, providing basic I²C-bus, Microwire-Bus and S-Bus functionality, allows efficient communication with low-cost external peripherals or serial access memories such as EEPROMs.

The main block of the SPI is an 8-bit shift register which can be read or written in parallel through the internal data bus of the ST9, and that can shift the data in or out on two separate pins, named SDI and SDO, respectively. The serial transfer is initiated with a write to the SPI Data Register (SPIDR). Input and output are done simultaneously, each most significant bit being output on SDO while the level at SDI becomes the least significant bit. Each time a bit is transferred, a pulse is output on the SCK pin. When eight bits are transferred, eight pulses have been sent on SCK, and the process stops. Depending on the interrupt control bits set in the SPI Control Register (SPICR), an interrupt can be requested on end of transmission. To summarise:

- Transfers are started by writing a byte into the SPI data register

- Input and output are done at the same time
- Input and output are done most-significant bit first

2.4.2 Comprehensive serial communication Interface (SCI)

The SCI is the association of a UART which offers all the usual functions for asynchronous transfer and a complex logic that handles tasks such as character recognition and DMA. It can also work as a simple serial expansion port that then resembles the SPI.

Serial communication is easily implemented, using formats and facilities offered by the ST9 Serial Communication Interface. This peripheral provide full flexibility in character format (5, 6, 7, 8 databits), odd, even or no parity, address bit, 1, 1.5 or 2 stop bits in asynchronous mode, and an integral baud rate generator allowing communication at up to 370 kbaud in asynchronous mode or 1.5 Mbytes/s in synchronous mode.

Industrial, telecom and communication systems users can also benefit from the self-test and address bit wake-up facility offered by the character search mode. Using the SCI, sophisticated high speed serial data communication can be implemented by simply selecting of the built in operating modes. For example, the SCI is able to automatically search for a character, or for its own address in a network environment.

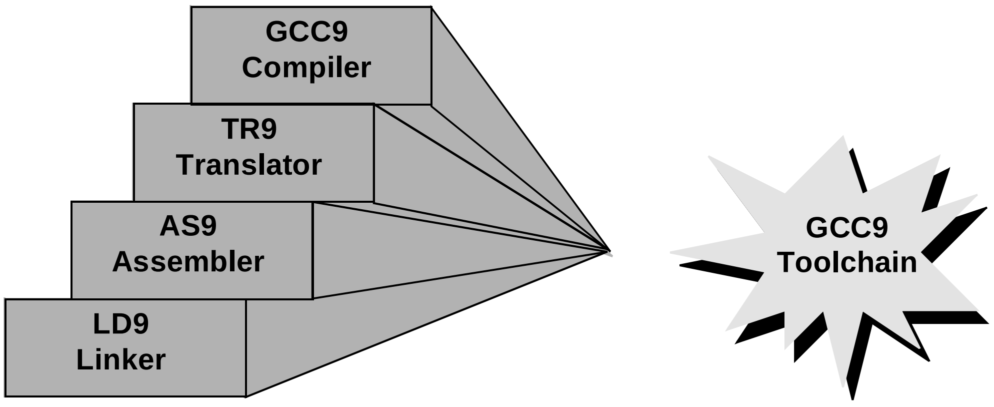
3 DEVELOPMENT TOOLS

3.1 SOFTWARE TOOLS: GNU C TOOLCHAIN

The programming tools available for the ST9 are known as GNU-9 tools. This GNU toolchain offers you a full set of resources for the development of code for the ST9 microcontroller. This consists of the optimized GNU C compiler, the macro-assembler, the linker/loader and the library archiver. Program debugging is made easier with the C language source level debugging which runs under Windows.

The GNU-9 is a set of MS-DOS programs that can be driven from a single program called GCC9. This program is capable of calling the following programs in turn:

Main block name	File name	Block name	Action
C-compiler	gcc9	C-compiler	Translates C-code into assembly source text.
Macro-assembler	tr9	Assembler pre-processor	Expands the macros, inserts the include files, removes the disabled conditional compilation blocks.
	as9	Assembler	Translates assembly language into machine code.
Linker	ld9	Linker	Links the different object files, positions the code at predefined addresses in memory.



3.1.1 GNU C compiler

The GNU C compiler for the ST9 allows you to write C source code using traditional C (Kernighan & Ritchie), ANSI C, or GNU extensions and to produce assembly language source code. You can use all standard types (char, int, short, long, signed or unsigned, float and double) in your code. The libraries which are delivered with the compiler, include string handling, conversion, I/O routines and mathematics. You have direct access to the Register File of the ST9, allowing access to all registers and on-chip peripherals.There are also some options for generating code for one or two memory spaces, one or two stacks and interrupt routines.

The C compiler allows inclusion of assembly language instructions with access to C program symbols. This means that the generated assembly source file may include interleaved C lines and assembly languages lines, and provide information for source-level debugging.

When used with the Assembler and Linker, it allows the generation of executable object code for all members of the ST9 family.

3.1.2 Assembler

The Assembler pre-processor allows macro substitution, file inclusion, conditional assembly, pseudo-instructions and pseudo-macros. Source level debugging information is generated with the object file by the assembler.

3.1.3 Linker

The Linker combines object code files issued by the assembler. It resolves references to external symbols and searches libraries for the required modules to produce an output file in a binary format, downloadable by the debugger to the ST9 emulator.

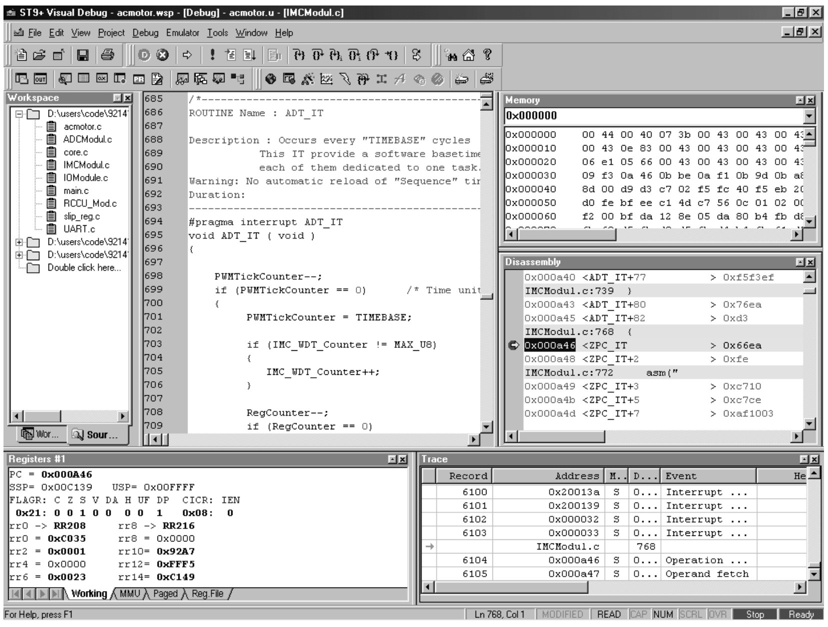
A map file is generated, including all mapping information on sections, files, and symbols. Separate files are produced to support the ST9 MMU mechanism.

3.1.4 Visual Debugger

The ST9 Visual Debugger (STVD9) allows source level debugging of C language and assembly language programs, even with optimized C language programs.

The debugger is able to generate trace information, with hardware information interleaved with source lines, and to display the local symbols of the current C procedure and the stack based on the C language source level.

Figure 36. Windows based Debugger



```
/*-----------------------------------------------------
ROUTINE Name : ADT_IT

Description : Occurs every "TIMEBASE" cycles
           This IT provide a software basetime
           each of them dedicated to one task.
Warning: No automatic reload of "Sequence" tir
Duration:
-----------------------------------------------------
#pragma interrupt ADT_IT
void ADT_IT ( void )
{

    PWMTickCounter--;
    if (PWMTickCounter == 0)      /* Time unit
    {
         PWMTickCounter = TIMEBASE;

         if (IMC_WDT_Counter != MAX_U8)
         {
            IMC_WDT_Counter++;
         }

         RegCounter--;
         if (RegCounter == 0)
```

```
0x000000
0x000000    00 44 00 40 07 3b 00 43 00 43 00 43
0x000010    00 43 0e 83 00 43 00 43 00 43 00 43
0x000020    06 e1 05 66 00 43 00 43 00 43 00 43
0x000030    09 f3 0a 46 0b be 0a f1 0b 9d 0b a8
0x000040    8d 00 d9 d3 c7 02 f5 fc 40 f5 eb 20
0x000050    d0 fe bf ee c1 4d c7 56 0c 01 02 00
0x000060    f2 00 bf da 12 8e 05 da 80 b4 fb d8
```

```
0x000a40 <ADT_IT+77          > 0xf5f3ef
IMCModul.c:739  }
0x000a43 <ADT_IT+80          > 0x76ea
0x000a45 <ADT_IT+82          > 0xd3
IMCModul.c:768  {
0x000a46 <ZPC_IT             > 0x66ea
0x000a48 <ZPC_IT+2           > 0xfe
IMCModul.c:772      asm("
0x000a49 <ZPC_IT+3           > 0xc710
0x000a4b <ZPC_IT+5           > 0xc7ce
0x000a4d <ZPC_IT+7           > 0xaf1003
```

```
PC = 0x000A46
SSP= 0x00C139   USP= 0x00FFFF
FLAGR: C Z S V DA H UF DP  CICR: IEN
 0x21: 0 0 1 0 0  0 0  1   0x08:  0
rr0 -> RR208      rr8 -> RR216
rr0 = 0xC035      rr8 = 0x0000
rr2 = 0x0001      rr10= 0x92A7
rr4 = 0x0000      rr12= 0xFFF5
rr6 = 0x0023      rr14= 0xC149
```

	Record	Address	M..	D...	Event	He
	6100	0x20013a	S	0...	Interrupt ...	
	6101	0x200139	S	0...	Interrupt ...	
	6102	0x000032	S	0...	Interrupt ...	
	6103	0x000033	S	0...	Interrupt ...	
→		IMCModul.c		768		
	6104	0x000a46	S	0...	Operation ...	
	6105	0x000a47	S	0...	Operand fetch	

The Windows based debugger provides a user friendly and highly flexible interface which may be configured to precisely match the user's requirement. Breakpoints allow the MCU to be halted when the application software accesses specific addresses or addresses within a selected range or on data fetch cycles. You may then read and modify any register and memory location. An on line assembler/disassembler is also available to ease debugging.

An important feature of the ST9 development system is that true source level debugging is possible, meaning code may be viewed at source level and breakpoints may be set on high level code, rather than on disassembled target code. This is much more meaningful to the user and results in a friendly and productive development environment.

3.2 HARDWARE TOOLS

The development system is controlled by a Host PC on which the Windows based debugger runs. The Host PC is simply connected to the mainframe by means of a parallel port.

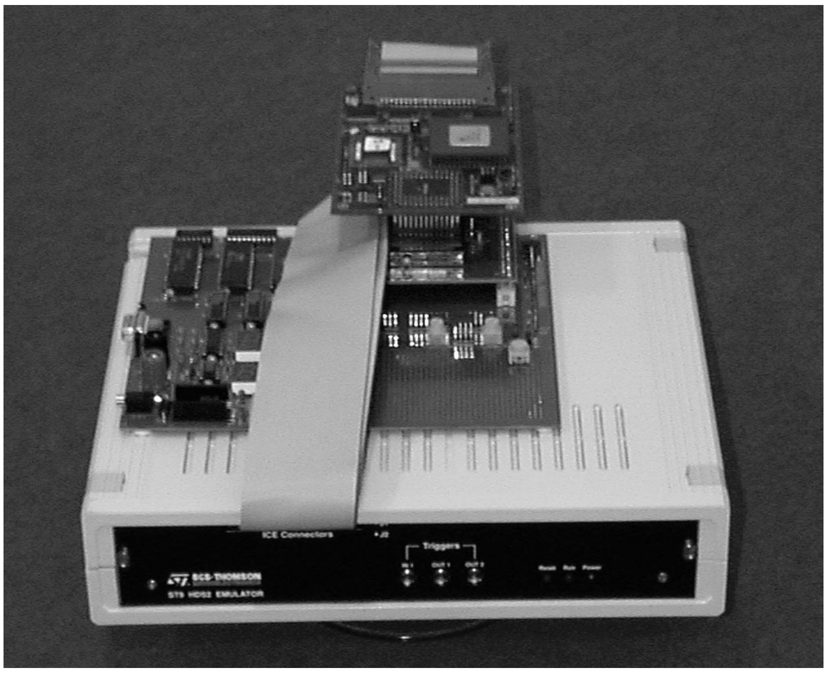
Once assembled, and/or compiled and linked, the application software may be downloaded to the real-time emulation memory, which you can configure, map and modify as required. The device probe is then connected to the application target hardware in place of the MCU and real-time emulation of the target application can begin, thus allowing sophisticated testing and debugging of both application hardware and software

3.2.1 Real-time development tools

The ST9 real-time development tools consist of various hardware and software components, which together form a flexible and powerful system designed to provide comprehensive development support for the ST9 family of MCUs.

The Hardware Development System (HDS2) mainframe, in conjunction with the probes for specific devices, allows emulation and development for the ST9 device family.

Figure 37. Emulator with its probe plugged in an application board



The WGDB9 Windows GNU Debugger software suite is supplied with the Emulator hardware, in addition to the conventional DOS ST9 Software suite, which includes a macroassembler, a linker/loader. The Windows based debugger provides a user friendly and highly flexible interface which may be configured to precisely match the user's requirements. All emulator settings are accessible via the control software.

A completely integrated trace facility is available. This hardware implemented function features 4KByte of 44-bit wide trace memory; sequential conditions may be defined on memory events; one breakpoint on a data value may also be set. One external signals is input on a subclic connector which can generate a breakpoint.

Trace memory events may be used as breakpoints or to enable or disable the trace recording feature. This powerful tool enables the user to detect and trap virtually any pattern, and thus rapidly debug the target application.

Log files offer the ability to send any screen display to a text file. In particular, log files are very useful to save the contents of the logic analyser and/or the contents of data registers to be subsequently analysed or printed.

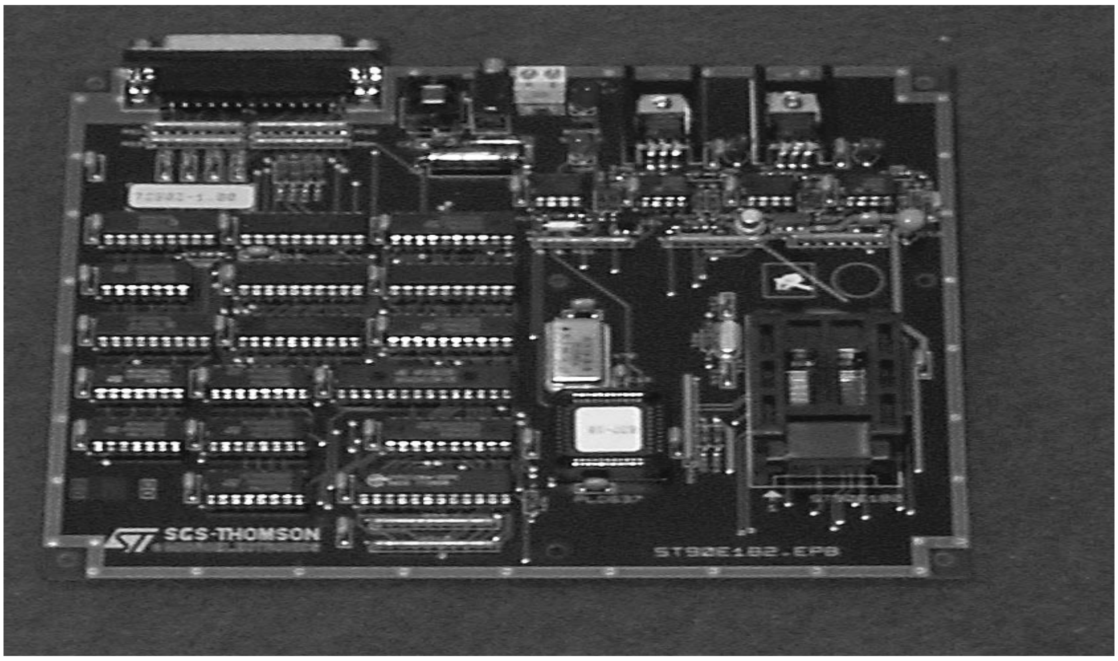
Command files can be used to execute a set of debugger commands in batch mode, to simplify and speed up the emulation session.

Finally, when the target program is fully debugged, the appropriate ST9 EPROM/OTP programming board can be used to program the EPROM/OTP version of the target device to allow stand-alone testing and evaluation.

3.2.2 EPROM programming boards

These boards are a programming tool for EPROM, OTP and FLASH members of the ST9 family.

Figure 38. EPROM programming board



The EPROM programming board is designed to program the non volatile versions of micro-controller, including the ceramic windowed and plastic OTP packages, and the FLASH. Several sockets are provided to receive the different existing packages types.

The EPROM programming board uses a RAM in which your code is downloaded. The EPROM device will be programmed from the contents of this RAM.

The board can perform three operations:

- verify the blank state of the microcontroller EPROM;
- program microcontroller with the content of hexadecimal file;
- verify the microcontroller.

IN SITU Programming of FLASH devices is also supported.

NOTES: