SP 2034

SERIAL PORT COMMUNICATION WITH AVR 8535-8 BIT MICROCONTROLLER

BY

AMIT MAGO-021072
PRAMOD KUMAR-021033
PRAVEEN KR. GUPTA-021061
ARVIND GUPTA-021098
SHAUNAK VYAS -021064

JAYPEE UNIVERSITY OF INFORMATION & TECHNOLOGY
juit
विद्या तत्व ज्योतिसम:
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

MAY- 2006

Submitted in partial fulfillment of the requirements of the degree of Bachelor of Technology

**DEPARTMENT OF ELECTRONICS & COMMUNICATION
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY – WAKNAGHAT**

CERTIFICATE

This is to certify that the work entitled, **"SERIAL PORT COMMUNICATION WITH AVR 8535-8 BIT MICROCONTROLLER"** submitted by Amit Mago (021072), Pramod Kumar (021033), Praveen Kr. Gupta (021061), Arvind Kumar (021098), and Shaunak R. Vyas (021064) in partial fulfillment for the award of degree of Bachelor of Technology in May 2006 of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Vivek Sehgal

Mr. Vivek Sehgal
Sr.Lecturer, Dept. of ECE

ACKNOWLEDGMENT

The team wishes to express their sincere appreciation for Mr. Vivek Sehgal, Lecturer, Dept. of Electronics & Communication Engineering for his guidance and whose familiarity with the needs and ideas of this topic was helpful during the programming phase of the project "**SERIAL PORT COMMUNICATION WITH AVR 8535-8 BIT MICROCONTROLLER**". We are also grateful to Prof. S. V. Bhooshan, H.o.D, Dept. of ECE, for giving us an opportunity to undertake this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

RISC-	Reduced Instruction Set Computing
CISC-	Complex Instruction Set Computing
ARM-	Acorn RISC Machine
ALU-	Arithmetic Logical Unit.
EEPROM-	Electrically Erasable Program Read Only Memory
SRAM-	Static Random Access Memory
SP-	Stack Pointer.
SPL-	Stack Pointer Low.
SPH-	Stack Pointer High.
UART-	Universal Asynchronous Receiver Transmitter

ABSTRACT

The ATmega8535 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture having 8K Bytes in-system programmable flash. By executing instructions in a single clock cycle, the ATmega8535 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed. In this project we have used ATmega8535 Microcontroller to build a robot car capable of moving in all the directions.At its simplest, a robot is machine that can be programmed to perform a variety of jobs, which usually involve moving or handling objects. Robots can range from simple machines to highly complex, computer-controlled devices. We have shown that it is possible to make a robot in low cost if selection of microcontroller is done properly as we did in this project.

CHAPTER 1. INTRODUCTION

The AVRs are a family of RISC microcontrollers from Atmel. Their internal architecture was conceived by two students: Alf-Egil Bogen and Vergard Wollan, at the Norwegian Institute of Technology (NTH) and further developed at Atmel Norway, a subsidiary founded by the two architects.

The AVR is a **Harvard architecture** machine with programs and data stored separately. Program instructions are stored in semi-permanent Flash memory which loads and manipulates data in the volatile SRAM. The AVRs have thirty-two single-byte registers.

The term **Harvard architecture** originally referred to computer architectures that used physically separate storage and signal pathways for their instructions and data (in contrast to the *von Neumann architecture*). The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape (24-bits wide) and data in relay latches (23-digits wide). These early machines had very limited data storage, entirely contained within the data processing unit, and provided no access to the instruction storage as data (making loading, modifying, etc. of programs entirely an offline process).

In a computer with von Neumann architecture, the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same signal pathways and memory. In a computer with Harvard architecture, the CPU can read both an instruction and data from memory at the same time. A computer with Harvard architecture can be faster because it is able to fetch the next instruction at the same time it completes the current instruction. Speed is gained at the expense of more complex electrical circuitry.

Modern high performance CPU chip designs incorporate aspects of both Harvard and von Neumann architecture. On chip cache memory is divided into an instruction cache and a data cache. Harvard architecture is used as the CPU accesses the cache. In the case of a cache miss, however, the data is retrieved from the main memory, which is not divided into separate instruction and data sections. Thus von Neumann architecture is used for off chip memory access.

Harvard architectures are also frequently used in specialized DSPs, or digital signal processors, commonly used in audio or video processing products.

CHAPTER 2. WHY NOT ARM?

ARM processors are characterized by having small amounts of program and data memory, and take advantage of the Harvard architecture and reduced instruction set (RISC) to ensure that most instructions can be executed within only one machine cycle. The separate storage means the program and data memories can be in different bit depths. For example, the PIC microcontrollers have an 8-bit data word but (depending on specific range of PICs) a 12-bit, 14-bit, or 16-bit program word. This allows a single instruction to contain a full-size data constant. Other RISC architectures, like the ARM, typically have to use at least two instructions to load a full-size constant.

2.1 ATmega8535

The ATmega8535 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing instructions in a single clock cycle, using RISC architecture. By executing instructions in a single clock cycle, the ATmega8535 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega8535 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes

EEPROM, 512 bytes SRAM, 32 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain in TQFP package, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM; Timer/Counters, SPI port, and interrupt system to continue functioning.

The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the asynchronous timer continue to run.

The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega8535 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega8535 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, In-Circuit Emulators, and evaluation kits.

Figure 1. Pinout ATmega8535

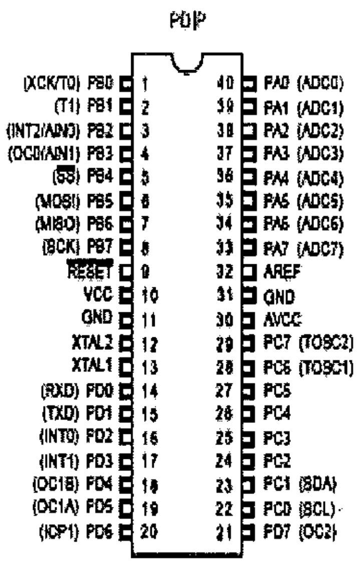
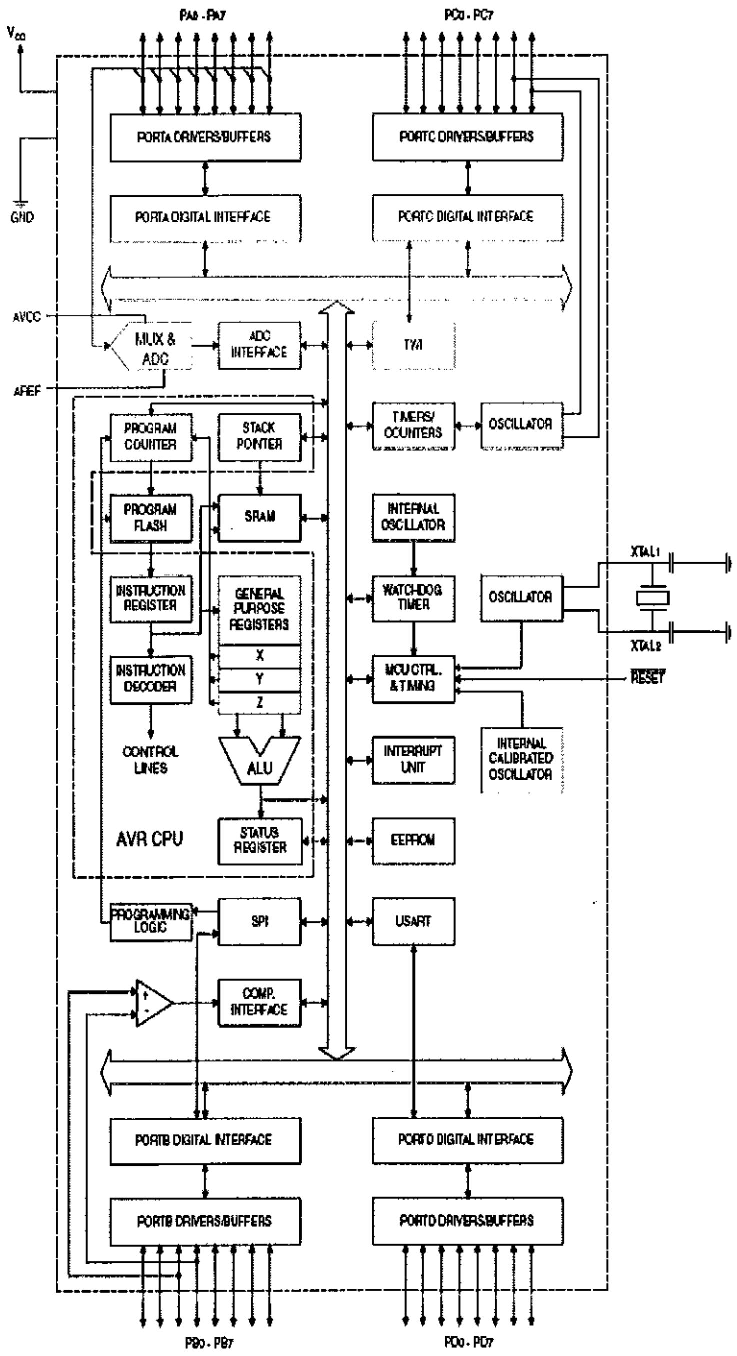


Figure 2. Block Diagram



BLOCK DIAGRAM OF ATMEGA 8535

CHAPTER 3. PIN DESCRIPTIONS

VCC	Digital supply voltage.
GND	Ground.

Port A (PA7...PA0) Port A serves as the analog inputs to the A/D Converter. Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output **buffers** have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7...PB0) Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port C (PC7...PC0) Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri- stated when a reset condition becomes active, even if the clock is not running.

Port D (PD7...PD0) Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

RESET Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running.

XTAL1 Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2. Output from the inverting Oscillator amplifier

AVCC AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.

AREF AREF is the analog reference pin for the A/D Converter.

CHAPTER 4. AVR ARCHITECTURE

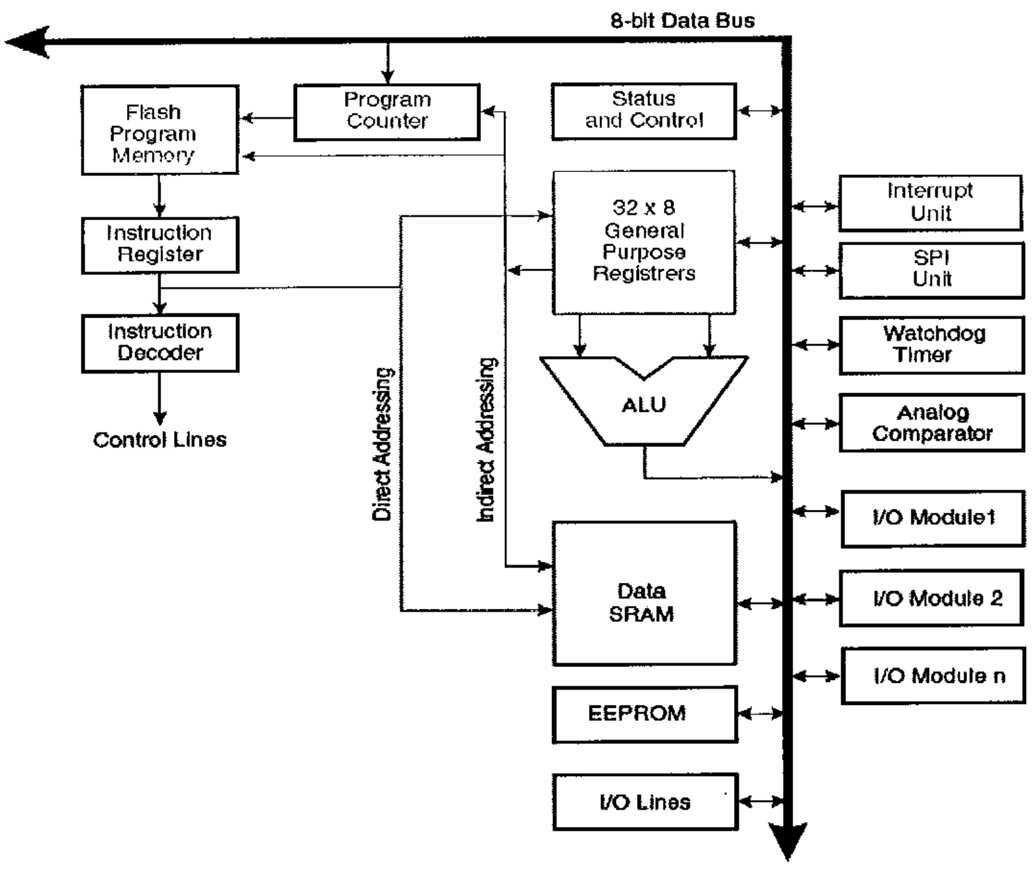
This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts. In order to maximize performance and parallelism, the AVR uses Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Re-Programmable Flash memory.

The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect addresses register pointers for Data Space addressing – enabling efficient address calculations. One of these address pointers can also be used as an address pointer for look up tables in Flash program memory. These added function registers are the 16-bit X-, Y-, and Z-registers, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation. Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction. Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section. During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The Stack Pointer SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture. The memory spaces in the AVR architecture are all linear and regular memory maps. A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority. The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F.

BLOCK DIAGRAM AVR ARCHITECTURE



CHAPTER 5. ALU – AIRTHMETIC LOGIC UNIT

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format.

5.1 STATUS REGISTER

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will, in many cases, remove the need for using the dedicated compare instructions, resulting in faster and more compact code. The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SRI
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 – I: Global Interrupt Enable

The Global Interrupt Enable bit must be set for the interrupts to be enabled.

• Bit 6 – T: Bit Copy Storage

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register file can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

• Bit 5 – H: Half Carry Flag

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half carry is useful in BCD arithmetic.

• Bit 4 – S: Sign Bit, $S = N \oplus V$

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V.

• Bit 3 – V: Two's Complement Overflow Flag

The Two's Complement Overflow Flag V supports two's complement arithmetic.

• Bit 2 – N: Negative Flag

The Negative Flag N indicates a negative result in an arithmetic or logic operation.

• Bit 1 – Z: Zero Flag

The Zero Flag Z indicates a zero result in an arithmetic or logic operation.

• Bit 0 – C: Carry Flag

The Carry Flag C indicates a carry in an arithmetic or logic operation.

5.2 STACK POINTER

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

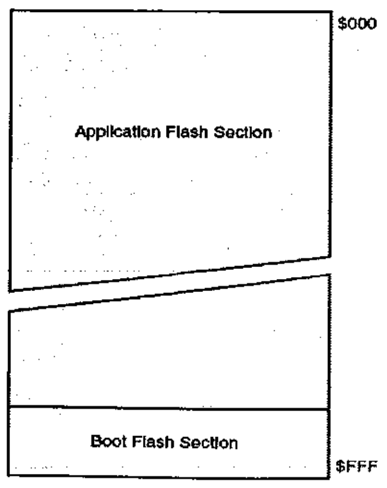
CHAPTER 6. AVR ATMEGA8535 MEMORIES

This section describes the different memories in the ATmega8535. The AVR architecture has two main memory spaces, the **Program Memory** and the **Data Memory** space. In addition, the ATmega8535 features an **EEPROM Memory** for data storage.

6.1 PROGRAM MEMORY

The ATmega8535 contains 8K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 4K x 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section. The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega8535 Program Counter (PC) is 12 bits wide, thus addressing the 4K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in "Boot Loader Support – Read-While-Write Self-Programming". "Memory Programming"contains a detailed description on Flash Programming in SPI or Parallel Programming

Program Memory Map



mode.

6.2 SRAM DATA MEMORY

The 608 Data Memory locations address the Register File, the I/O Memory, and the internal data SRAM. The first 96 locations address the Register File and I/O Memory, and the next 512 locations address the internal data SRAM. The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers. The direct addressing reaches the entire data space. The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register. When using register indirect addressing modes with automatic pre-decrement and post increment, the address registers X, Y, and Z are decremented or incremented. The 32 general purpose working registers, 64 I/O Registers, and the 512 bytes of internal data SRAM in the ATmega8535 are all accessible through all these addressing modes.

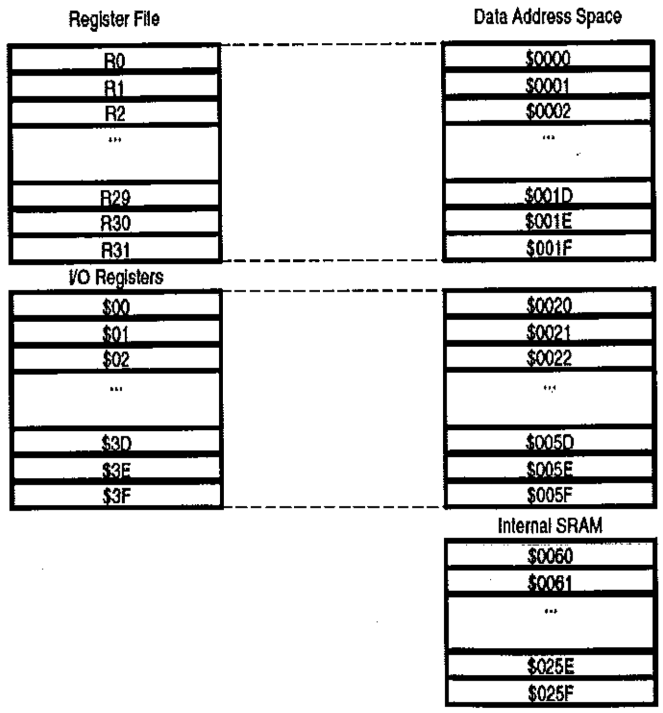


Figure for Data Memory

6.3 EEPROM DATA MEMORY

The ATmega8535 contains 512 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 Write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

CHAPTER 7. INTERRUPTS

This section describes the specifics of the interrupt handling as performed in ATmega8535.

Interrupt Vectors in ATmega8535

Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	INT2	External Interrupt Request 2
20	0x013	TIMER0 COMP	Timer/Counter0 Compare Match
21	0x014	SPM_RDY	Store Program Memory Ready

CHAPTER 8. INSTRUCTION SET NOMENCLATURE

Status Register (SREG)

SREG:	Status register
C:	Carry flag in status register
Z:	Zero flag in status register
N:	Negative flag in status register
V:	Two's complement overflow indicator
S:	$N \oplus V$, for signed tests
H:	Half Carry flag in the status register
T:	Transfer bit used by BLD and BST instructions
I:	Global interrupt enable/disable flag

Registers and Operands

Rd:	Destination (and source) registers in the register file
Rr:	Source register in the register file
R:	Result after instruction is executed
K:	Constant data
k:	Constant address
b:	Bit in the register file or I/O register (3 bit)
s:	Bit in the status register (3 bit)
X, Y, Z:	Indirect address register (X=R27:R26, Y=R29:R28 and Z=R31:R30)
A:	I/O location address
q:	Displacement for direct addressing (6 bit)

CHAPTER 9. I/O REGISTERS

RAMPX, RAMPY, RAMPZ

Registers concatenated with the X, Y and Z registers enabling indirect addressing of the whole data space on MCUs with more than 64K bytes data space, and constant data fetch on MCUs with more than 64K bytes program space.

RAMPD

Register concatenated with the Z register enabling direct addressing of the whole data space on MCUs with more than 64K bytes data space.

EIND

Register concatenated with the instruction word enabling indirect jump and call to the whole program space on MCUs with more than 64K bytes program space.

STACK

STACK: Stack for return address and pushed registers

SP: Stack Pointer to STACK

FLAGS

⇔: Flag affected by instruction

0: Flag cleared by instruction

1: Flag set by instruction

-: Flag not affected by instruction

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
$Rd > Rr$	$Z\bullet(N \oplus V) = 0$	BRLT[1]	$Rd \leq Rr$	$Z+(N \oplus V) = 1$	BRGE*	Signed
$Rd \geq Rr$	$(N \oplus V) = 0$	BRGE	$Rd < Rr$	$(N \oplus V) = 1$	BRLT	Signed
$Rd = Rr$	$Z = 1$	BREQ	$Rd \neq Rr$	$Z = 0$	BRNE	Signed
$Rd \leq Rr$	$Z+(N \oplus V) = 1$	BRGE[1]	$Rd > Rr$	$Z\bullet(N \oplus V) = 0$	BRLT*	Signed
$Rd < Rr$	$(N \oplus V) = 1$	BRLT	$Rd \geq Rr$	$(N \oplus V) = 0$	BRGE	Signed
$Rd > Rr$	$C + Z = 0$	BRLO[1]	$Rd \leq Rr$	$C + Z = 1$	BRSH*	Unsigned
$Rd \geq Rr$	$C = 0$	BRSH/BRCC	$Rd < Rr$	$C = 1$	BRLO/BRCS	Unsigned
$Rd = Rr$	$Z = 1$	BREQ	$Rd \neq Rr$	$Z = 0$	BRNE	Unsigned
$Rd \leq Rr$	$C + Z = 1$	BRSH[1]	$Rd > Rr$	$C + Z = 0$	BRLO*	Unsigned
$Rd < Rr$	$C = 1$	BRLO/BRCS	$Rd \geq Rr$	$C = 0$	BRSH/BRCC	Unsigned
Carry	$C = 1$	BRCS	No carry	$C = 0$	BRCC	Simple
Negative	$N = 1$	BRMI	Positive	$N = 0$	BRPL	Simple
Overflow	$V = 1$	BRVS	No overflow	$V = 0$	BRVC	Simple
Zero	$Z = 1$	BREQ	Not zero	$Z = 0$	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test. i.e. CP Rd,Rr $\rightarrow$ CP Rr,Rd

CHAPTER 10. INSTRUCTION SET SUMMARY

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
		Arithmetic and Logic Instructions			
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1
ADIW	Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1
SBIW	Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V,S	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V,S	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd × Rr (UU)	Z,C	2
MULS	Rd,Rr	Multiply Signed	R1:R0 ← Rd × Rr (SS)	Z,C	2
MULSU	Rd,Rr	Multiply Signed with Unsigned	R1:R0 ← Rd × Rr (SU)	Z,C	2
FMUL	Rd,Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd × Rr)<<1 (UU)	Z,C	2
FMULS	Rd,Rr	Fractional Multiply Signed	R1:R0 ← (Rd × Rr)<<1 (SS)	Z,C	2
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd × Rr)<<1 (SU)	Z,C	2

Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
Branch Instructions					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2
EIJMP		Extended Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	2
JMP	k	Jump	PC ← k	None	3
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3 / 4
ICALL		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3 / 4
EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	4
CALL	k	Call Subroutine	PC ← k	None	4 / 5
RET		Subroutine Return	PC ← STACK	None	4 / 5
RETI		Interrupt Return	PC ← STACK	I	4 / 5
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,S,H	1
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	A, b	Skip if Bit in I/O Register Cleared	if(I/O(A,b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	A, b	Skip if Bit in I/O Register Set	If(I/O(A,b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC ←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC ←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2

Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
		Data Transfer Instructions			
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
STS	k, Rr	Store Direct to data space	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2

Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
ESPM		Extended Store Program Memory	(RAMPZ:Z) ← R1:R0	None	-
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
		Bit and Bit-test Instructions			
LSL	Rd	Logical Shift Left	Rd(n+1)←Rd(n),Rd(0)←0,C←Rd(7)	Z,C,N,V,H	1
LSR	Rd	Logical Shift Right	Rd(n)←Rd(n+1),Rd(7)←0,C←Rd(0)	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)←Rd(n),C←Rd(7)	Z,C,N,V,H	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)←Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	2
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	2
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1

Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clock Note
SEI		Global Interrupt Enable	$I \leftarrow 1$	I	1
CLI		Global Interrupt Disable	$I \leftarrow 0$	I	1
SES		Set Signed Test Flag	$S \leftarrow 1$	S	1
CLS		Clear Signed Test Flag	$S \leftarrow 0$	S	1
SEV		Set Two's Complement Overflow	$V \leftarrow 1$	V	1
CLV		Clear Two's Complement Overflow	$V \leftarrow 0$	V	1
SET		Set T in SREG	$T \leftarrow 1$	T	1
CLT		Clear T in SREG	$T \leftarrow 0$	T	1
SEH		Set Half Carry Flag in SREG	$H \leftarrow 1$	H	1
CLH		Clear Half Carry Flag in SREG	$H \leftarrow 0$	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR)	None	1

CHAPTER 11. MAX232N

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply TIA/EIA-232-F voltage levels from a single 5-V supply. Each receiver converts TIA/EIA-232-F inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V, a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into TIA/EIA-232-F levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC library.

Function Tables

EACH DRIVER

INPUT TIN	OUTPUT TOUT
L	H
H	L

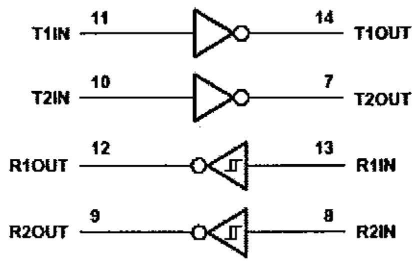
H = high level, L = low level

EACH RECEIVER

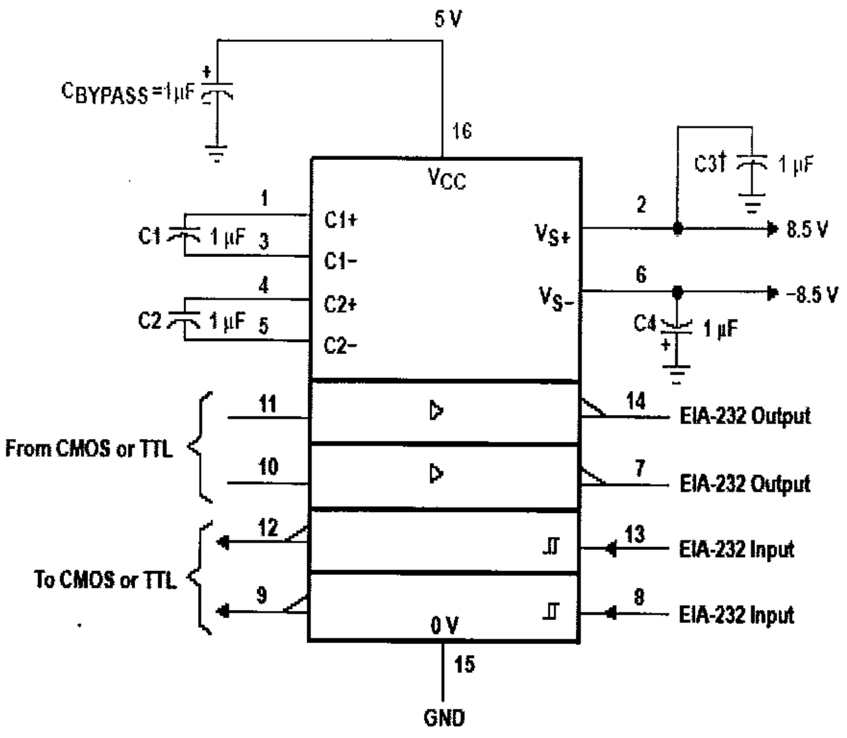
INPUT RIN	OUTPUT ROUT
L	H
H	L

H = high level, L = low level

logic diagram (positive logic)



APPLICATION INFORMATION



† C3 can be connected to V_{CC} or GND.

NOTES: A. Resistor values shown are nominal.

B. Nonpolarized ceramic capacitors are acceptable. If polarized tantalum or electrolytic capacitors are used, they should be connected as shown. In addition to the 1-µF capacitors shown, the MAX202 can operate with 0.1-µF capacitors.

Figure 4. Typical Operating Circuit

CHAPTER 12. CODE

```
.include "C:\Atmel\AVR Tools\AvrAssembler\Appnotes\m8535def.inc"

 .def temp = r16                        \\ Defines temp and puts its value
equal to r16
 .def mode = r21                        \\ Defines mode and puts its value
equal to r21

 .equ forward = 0b00000001
 .equ reverse = 0b00000010
 .equ left    = 0b00000100
 .equ right   = 0b00001000

.org 0x00
 rjmp reset                            \\ Relative jump to reset
.org.OVF0addr                          \\ Overflow0 Interrupt Vector Address
 rjmp timer                            \\ Relative jump to timer
.org URXCaddr                          \\ UART Receive Complete Interrupt Vector
Address
 rjmp receive                          \\ Relative jump to recieve

reset:

ldi temp,high(ramend)                  \\ Set temp to high initialize SP
out SPH,temp                           \\ Writes high to i/o port SP
ldi temp,low(ramend)                   \\ Sets temp to low
out SPL,temp                           \\ Writes low to i/o port

ldi temp,0xFF                          \\ Sets temp to 0xff
```

```
 out DDRB,temp                          \\ Writes 0xff to i/o port Data Direction
Register PORTB
 ldi temp,0xFF                          \\ Sets temp to 0xff
 out PORTB,temp                         \\ Writes 0xff to i/o port

 ldi temp,0xFF                          \\ Sets temp to 0xFF
 out DDRA,temp                         \\ Writes 0xff to i/o port
 ldi temp,0x00                         \\ Sets temp to 0x00
 out PORTA,temp                         \\ Writes 0x00 to i/o port

 ldi temp,0xFF                          \\ Sets temp to 0xFF
 out DDRC,temp                          \\ Writes 0xff to i/o port
 ldi temp,0x00                          \\ Sets temp to 0x00
 out PORTC,temp                         \\ Writes 0x00 to i/o port

 ldi temp,0                            \\ Clears temp
 out UBRRH,temp                        \\ Writes 0 to i/o port BAUD RATE
 ldi temp,51                           \\ Sets temp to 51
 out UBRRL,temp                        \\ Writes 51 to i/o port
; UBRR to 51 implies 9600 baud with an 8 MHz crystal.
 ldi temp,0b10010000                   \\ Sets temp to 0b10010000
 out UCSRB,temp                        \\ Writes 0b10010000 to i/o port

 cbr mode,left                        \\ Clears mode
 cbr mode,right
 cbr mode,forward
 cbr mode,reverse

 ldi r26,0                          \\ Clears r26
 ldi r27,0                          \\ Clears r27
 sei                                \\ Enables interrupts
```

```
loop:

 bst mode,0                        \\ Stores bit 0 of mode in T flag
 brts poll                         \\ Branch if this bit was reset
 bst mode,1                        \\ Stores bit 1 of mode in T flag
 brts backs                        \\ Branch if this bit was set
 rjmp endout                       \\ Relative jump to endout


poll:

 cbi PORTB,0                       \\ Clear bit 0 in port B

 sbi PORTA,0                       \\ Set read bit in port A
 cbi PORTA,1                       \\ Clear bit  1 in port A
 cbi PORTA,2          ; speed      \\ Clear bit 2 in port A
 cbi PORTA,3                       \\ Clear bit 3 in port A

 rcall delay                       \\ Relative call to subroutine - delay

 cbi PORTA,0                       \\ Clear bit 0 in port A
 sbi PORTA,1                       \\ Set bit 1 in port A
 cbi PORTA,2                       \\ Clear bit 2 in port A
 cbi PORTA,3                       \\ Clear bit 3 in port A

 rcall delay                       \\ Relative call to subroutine - delay

 cbi PORTA,0                       \\ Clear bit 0 in port A
 cbi PORTA,1                       \\ Clear bit 1 in port A
 cbi PORTA,2                    \\ Clear bit 2 in port A
 sbi PORTA,3                    \\ Set bit 3 in port A

 rcall delay                    \\ Relative call to subroutine - delay

 cbi PORTA,0                    \\ Clear bit 0 in port A
 cbi PORTA,1                    \\ Clear bit 1 in port A
 sbi PORTA,2                    \\ Set bit 2 in port A
 cbi PORTA,3                    \\ Clear bit 3 in port A

 rcall delay                    \\ Relative call to subroutine - delay

endout:
 sbi PORTB,0                    \\ Set read bit in port B

 rjmp loop                      \\ Relative jump to loop

backs:
 cbi PORTB,0                    \\ Clear bit 0 in port B
 cbi PORTA,0                    \\ Clear bit 0 in port A
 cbi PORTA,1                    \\ Clear bit 1 in port A
 sbi PORTA,2                    \\ Set bit 2 in port A
 cbi PORTA,3                    \\ Clear bit 3 in port A

 rcall delay                    \\ Relative call to subroutine - delay

 cbi PORTA,0                    \\ Clear bit 0 in port A
 cbi PORTA,1                    \\ Clear bit 1 in port A
 cbi PORTA,2                    \\ Clear bit 2 in port A
 sbi PORTA,3                    \\ Set bit 3 in port A
```

```
rcall delay                \\ Relative call to subroutine - delay

cbi PORTA,0               \\ Clear bit 0 in port A
sbi PORTA,1               \\ Set bit 1 in port A
cbi PORTA,2               \\ Clear bit 2 in port A
cbi PORTA,3               \\ Clear bit 3 in port A

rcall delay                 \\ Relative call to subroutine - delay

sbi PORTA,0                 \\ Set read bit in port A
cbi PORTA,1                 \\ Clear bit 1 in port A
cbi PORTA,2       ; speed   \\ Clear bit 2 in port A
cbi PORTA,3                 \\ Clear bit 3 in port A

rcall delay                \\ Relative call to subroutine - delay
sbi PORTB,0                \\ Set read bit in port B
rjmp loop                  \\ Relative jump to loop


;************************************************


recieve:
 cpi temp,'w'             \\ Compare temp with 'w'
 breq run_forward         \\ Branch if equal
 cpi temp,'s'             \\ Compare temp with 's'
 breq run_backward        \\ Branch if equal
 cpi temp,'h'             \\ Compare temp with 'h'
 breq stop                \\ Branch if equal
 cpi temp,'a'             \\ Compare temp with 'a'
 breq lean_left           \\ Branch if equal
 cpi temp,'d'             \\ Compare temp with 'd'
 breq lean_right              \\ Branch if equal
 rjmp outloop                 \\ Relative jump to outloop
 run_forward:
 sbr mode,forward
 cbr mode, reverse
 rjmp outloop                  \\ Relative jump to outloop
run_backward:
 sbr mode,reverse
 cbr mode, forward
 rjmp outloop                  \\ Relative jump to outloop
stop:
 cbr mode,reverse
 cbr mode,forward
 cbi PORTA,0                   \\ Clear bit 0 in port A
 cbi PORTA,1                   \\ Clear bit 1 in port A
 cbi PORTA,2                  \\ Clear bit 2 in port A
 cbi PORTA,3                  \\ Clear bit 3 in port A
 rjmp outloop                 \\ Relative jump to outloop
lean_left:
 ldi temp,0b00000001          \\ Set temp to 0b00000001
 out TCCR0,temp               \\ Store Temp to i/o port
 ldi temp,0x01                \\ Set temp top 0x01
 out TIMSK,temp               \\ Store temp to i/o port TIMSK
 sbr mode,left
 cbr mode,right
 rjmp outloop                 \\ Relative jump to outloop
lean_right:
 ldi temp,0b00000001          \\ Set temp to 0b00000001
 out TCCR0,temp               \\ Store temp to i/o port TCCR0
 ldi temp,0x01                \\ Set temp to 0x01
 out TIMSK,temp               \\ Store temp to i/o port TIMSK
 sbr mode,right
 cbr mode,left
 rjmp outloop                 \\ Relative jump to outloop
straight:
 ldi temp,0b00000001          \\ Set temp to 0b00000001
 out TCCR0,temp               \\ Store temp to i/o port TCCR0
 ldi temp,0x01                \\ Set temp to 0x01
 out TIMSK,temp               \\ Store temp to i/o port TIMSK
 cbr mode,right
 cbr mode,left
 sbr mode,strait
 rjmp outloop                 \\ Relative jump to outloop

outloop:
 reti

;**************************************************
timer:

 bst mode,2                   \\ Store bit 2 of mode in T
 brts turn_left               \\ Branch if this bit was set
 bst mode,3                   \\ Store bit 3 of mode in T
 brts turn_right              \\ Branch if this bit was set
 bst mode, 4                  \\ Store bit 4 of mode in T
 brts position0               \\ Branch if this bit was set
 rjmp out_timer               \\ Relative jump to out_timer

turn_left:

 inc r26                      \\ Increment r26
```

```
    dec r27                 \\ Decrement r27

    cbr mode,left
    cbr mode,right
    cbi PORTC,4             \\ Clear bit 4 in port C
    sbi PORTC,1             \\ Set bit 1 in port C
    cbi PORTC,2             \\ Clear bit 2 in port C
    cbi PORTC,3             \\ Clear bit 3 in port C
    rcall delay             \\ Relatice call subroutine - delay
    cbi PORTC,4             \\ Clear bit 4 in port C
    cbi PORTC,1             \\ Clear bit 1 in port C
    sbi PORTC,2             \\ Set bit 2 in port C
    cbi PORTC,3             \\ Clear bit 3 in port C
    rcall delay             \\ Relative call to subroutine - delay
    sbi PORTC,4             \\ Set bit 4 in port C
    cbi PORTC,1             \\ Clear bit 1 in port C
    cbi PORTC,2             \\ Clear bit 2 in port C
    cbi PORTC,3             \\ Clear bit 3 in port C
    rcall delay             \\ Relative call to subroutine - delay
    cbi PORTC,4             \\ Clear bit 4 in port C
    cbi PORTC,1             \\ Clear bit 1 in port C
    cbi PORTC,2             \\ Clear bit 2 in port C
    sbi PORTC,3             \\ Set bit 4 in port C
    rcall delay             \\ Relative call to subroutine - delay
    rjmp out_timer          \\ Relative jump to out_timer
turn_right:

    inc r27                 \\ Increment r27
    cbr mode,left
    cbr mode,right
    ldi temp,0              \\ Clear temp
```

```
out TCCR0,temp           \\ Store temp in i/o port TCCR0

cbi PORTC,4              \\ Clear bit 4 in port C
cbi PORTC,1              \\ Clear bit 1 in port C
cbi PORTC,2              \\ Clear bit 2 in port C
sbi PORTC,3              \\ Set bit 3 in port C
rcall delay              \\ Relative call to subroutine - delay
sbi PORTC,4              \\ Set bit 4 in port C
cbi PORTC,1              \\ Clear bit 1 in port C
cbi PORTC,2              \\ Clear bit 2 in port C
cbi PORTC,3              \\ Clear bit 3 in port C
rcall delay              \\ Relative call to subroutine - delay
cbi PORTC,4              \\ Clear bit 4 in port C
cbi PORTC,1              \\ Clear bit 1 in port C
sbi PORTC,2              \\ Store bit 2 in port C
cbi PORTC,3              \\ Clear bit 3 in port C
rcall delay              \\ Relative call to subroutine - delay
cbi PORTC,4              \\ Clear bit 4 in port C
sbi PORTC,1              \\ Store bit 1 in port C
cbi PORTC,2              \\ Clear bit 2 in port C
cbi PORTC,3              \\ Clear bit 3 in port C
rcall delay              \\ Relative call to subroutine - delay

rjmp out_timer           \\ Relative jump to out_timer

position0:

 cbr mode,strait

 mov temp,r26            \\ Copy r26 to temp
 subi temp,0             \\ Substract 0 from temp
```

```
  brmi turned_right    \\ Branch if minus
  breq out_timer       \\ Branch if equal
  mov temp,r27         \\ Copy r27 to temp
  subi temp,0          \\ Substract 0 from temp
  brmi turned_left     \\ Branch if minus
  breq out_timer       \\ Branch if equal
  rjmp out_timer       \\ Relative jump to out_timer

turned_left:

  cbi PORTC,4          \\ Clear bit 4 in port C
  cbi PORTC,1          \\ Clear bit 1 in port C
  cbi PORTC,2          \\ Clear bit 2 in port C
  sbi PORTC,3          \\ Store bit 3 in port C
  rcall delay          \\ Relative call to subroutine - delay
  sbi PORTC,4          \\ Store bit 4 in port C
  cbi PORTC,1          \\ Clear bit 1 in port C
  cbi PORTC,2          \\ Clear bit 2 in port C
  cbi PORTC,3          \\ Clear bit 3 in port C
  rcall delay          \\ Relative call to subroutine - delay
  cbi PORTC,4          \\ Clear bit 4 in port C
  cbi PORTC,1          \\ Clear bit 1 in port C
  sbi PORTC,2          \\ Store bit 2 in port C
  cbi PORTC,3          \\ Clear bit 3 in port C
  rcall delay          \\ Relative call to subroutine - delay
  cbi PORTC,4          \\ Clear bit 4 in port C
  sbi PORTC,1          \\ Store bit 1 in port C
  cbi PORTC,2          \\ Clear bit 2 in port C
  cbi PORTC,3          \\ Clear bit 3 in port C
  rcall delay          \\ Relative call to subroutine - delay
  ldi temp,0           \\ Clear temp
```

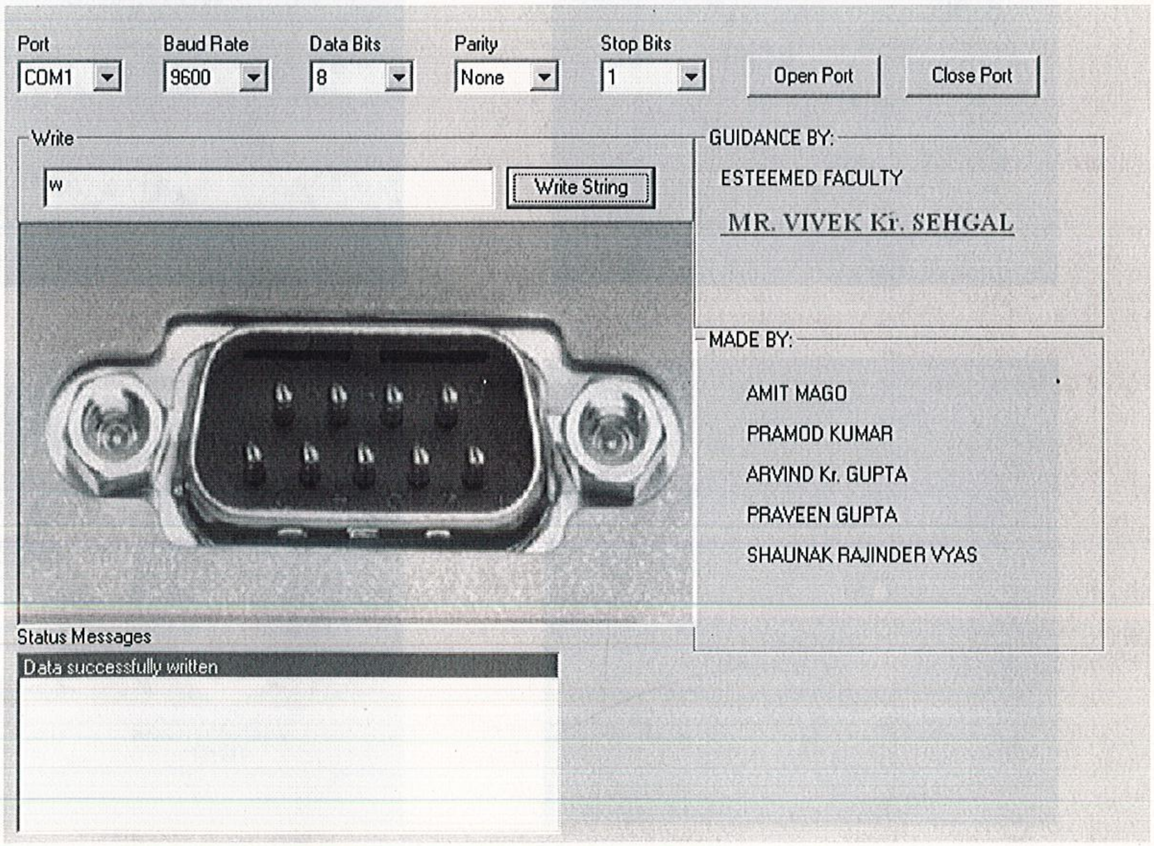
```
out TCCR0,temp      \\ Store temp in i/o port TCCR0
dec r26             \\ Decrement r26
brne turned_left    \\ Branch if not equal
ldi r26,0           \\ Clear r26
ldi r27,0           \\ Clear r27

rjmp out_timer      \\ Relative jump to out_timer


turned_right:

cbi PORTC,4         \\ Clear bit 4 in port C
sbi PORTC,1         \\ Set bit 1 in port C
cbi PORTC,2         \\ Clear bit 2 in port C
cbi PORTC,3         \\ Clear bit 3 in port C
rcall delay         \\ Relative call to subroutine - delay
cbi PORTC,4         \\ Clear bit 4 in port C
cbi PORTC,1         \\ Clear bit 1 in port C
sbi PORTC,2         \\ Set bit 2 in port C
cbi PORTC,3         \\ Clear bit 3 in port C
rcall delay         \\ Relative call to subroutine - delay
sbi PORTC,4       ;set 0 position
cbi PORTC,1         \\ Clear bit 1 in port C
cbi PORTC,2         \\ Clear bit 2 in port C
cbi PORTC,3         \\ Clear bit 3 in port C
rcall delay         \\ Relative call to subroutine - delay
cbi PORTC,4         \\ Clear bit 4 in port C
cbi PORTC,1         \\ Clear bit 1 in port C
cbi PORTC,2         \\ Clear bit 2 in port C
sbi PORTC,3         \\ Set bit 3 in port C
rcall delay         \\ Relative call to subroutine - delay
```

```
 ldi temp,0           \\ clear temp
 out TCCR0,temp       \\ Store temp to i/o port TCCR0
 dec r27              \\ Decrement r27
 brne turned_right    \\ Branch if not equal
 ldi r26,0            \\ Clear r26
 ldi r27,0            \\ Clear r27
out_timer:
 ldi temp,0           \\ Clear temp
 out TCCR0,temp       \\ Store temp to i/o port TCCR0

 reti

;**************************************************

delay:

 ldi r25,0xFF         \\ Store 0xFF to r25
lop1:
 ldi r24,0xFF         \\ Store 0xFF to r24
lop:
 dec r24              \\ Decrement r24
 brne lop dec r25     \\ Branch if not equal
 brne lop1            \\ Branch if not equal

ret
;**************************************************
```

CHAPTER 13. SERIAL PORT COMMUNICATION

If you use the UART to talk to a PC, the UART should be connected to a COM port of the PC. You can use a simple terminal program, such as HyperTerminal, on the PC. The terminal program should be set to 9600 baud, no parity, one stop bit, and no flow control. The connection to the PC for the AVR development board assumes a RS232 cable with straight-through connection. It transmits a 16 bit ASCII value of the data through the RS232 port serially. We designed the same in Visual Basic.

THE SCREENSHOT


Port: COM1
Baud Rate: 9600
Data Bits: 8
Parity: None
Stop Bits: 1

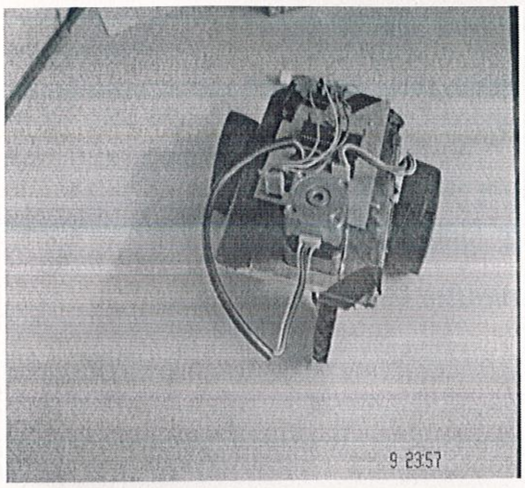
w

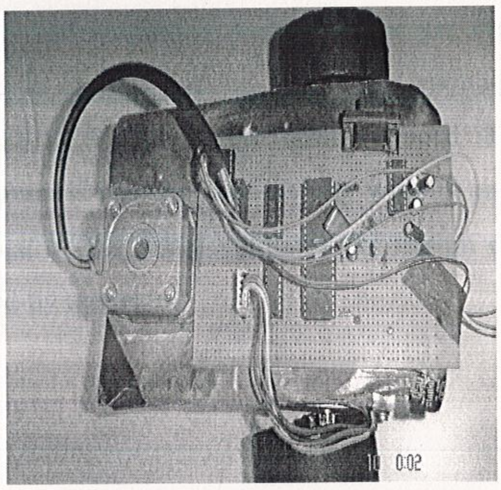
GUIDANCE BY:
ESTEEMED FACULTY
MR. VIVEK Kr. SEHGAL

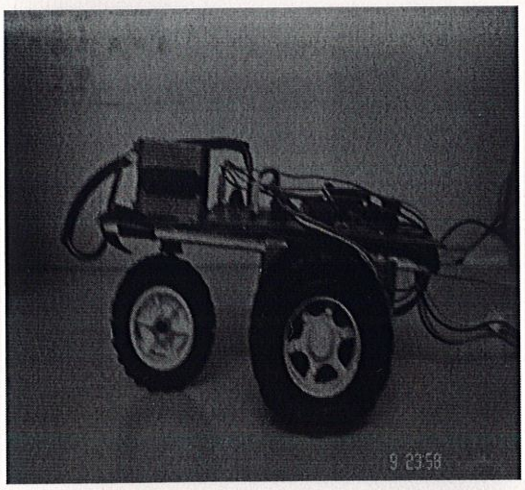
MADE BY:
AMIT MAGO
PRAMOD KUMAR
ARVIND Kr. GUPTA
PRAVEEN GUPTA
SHAUNAK RAJINDER VYAS

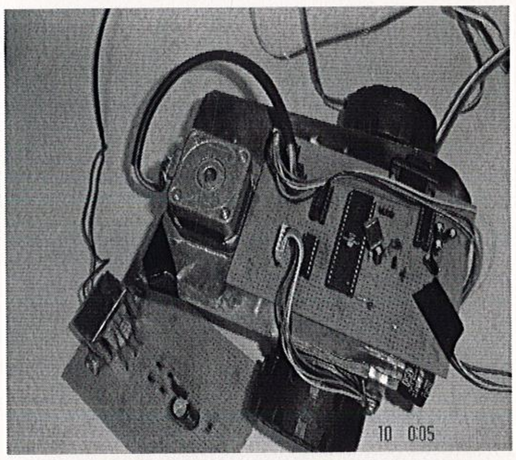
Status Messages
Data successfully written

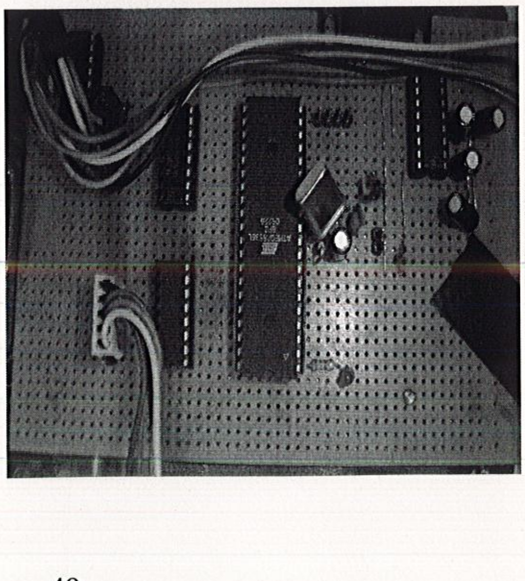
CHAPTER 14. SNAPSHOTS











CONCLUSION

Target of our team were achieved and responses were well as per our algorithm. This robot car is tested in various terrains and found running smoothly. The careful and economical design by our team reduced the overall cost of the project. The program is running successfully without any error and warning. The performance of the robot validates the design of hardware and software, implemented by the team.

CHAPTER 16 .BIBLIOGRAPHIES

1. Microprocessor Architecture Programming and Application with 8085. By:- Ramesh Gaonkar
2. Microprocessor Programming and Interfacing in 8086

   By: - B.Brey
3. CMOS Digital Integrated Circuits Analysis and Design

   By:- Sung-Mo Kang and Yusuf Leblebici
4. www.atmel.com
5. www.8051.com