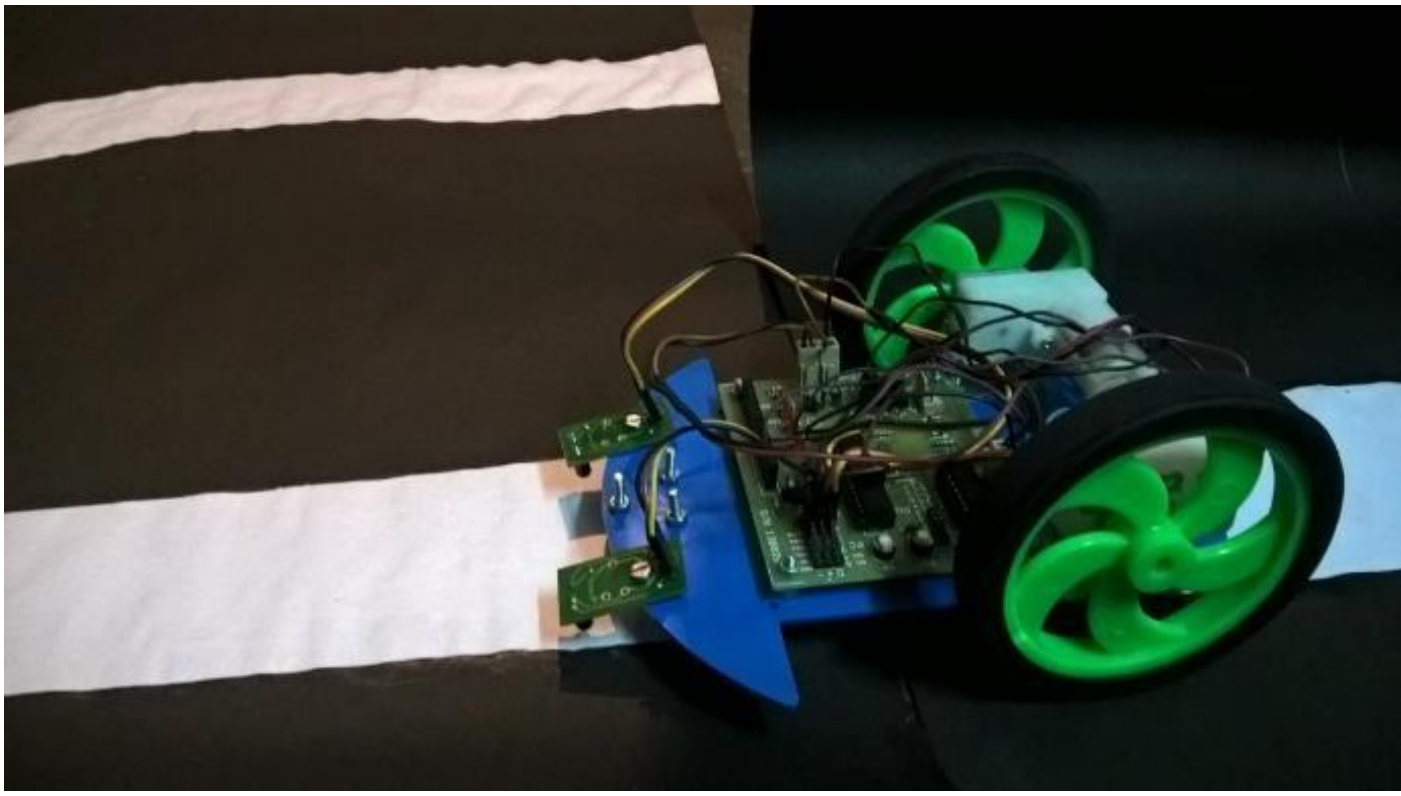
ASPIRINE nBARS

98.1



An industrial project report on

A microcontroller based line follower autonomous robot

Under the guidance of ***Mr. Utpal Kumar Paul***

Prepared and submitted by

BHAWANI SHANKAR BHARTI
RITESH KUMAR PANDEY
ANANT KUMAR
ABHINAS RAJAK
ANJEET KUMAR
NAGENDRA KUMAR
SANDEEP KUMAR
SOVICK KUMAR SHILL
ANIL KUMAR YADAV
SURAJ KUMAR SHILL

Department of **Electronics and Communication Engineering**

Certificate of recommendation

It is hereby recommended that the industrial project report with project entitled "***ASPIRINE nBARS 98.1-a microcontroller based line following autonomous robot"*** has been prepared and submitted by the mentioned group members, be accepted in partial fulfilment for the Diploma in ***Electronics and Communication Engineering (ECE)***, ***Government Polytechnic, Silli (GPS)***, Jharkhand under ***State Board of Technical Education, Jharkhand (SBTEJ)***.

Mr. Utpal Kumar Paul
Department-In-Charge
Department of ECE
Government Polytechnic, Silli

Certificate of approval

The industrial project report entitled with ***"ASPIRINE nBARS 98.1-a microcontroller based line following autonomous robot"*** is hereby approved and certified as a creditable study of technological subject carried out and presented in a manner, satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted.

It is understood that by the approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in, but approve the project report only for the purposed for which it has been submitted.

Mr. Utpal Kumar Paul
Department-In-Charge
Department of ECE
Government Polytechnic, Silli

Acknowledgement

Nothing would have been created or developed if there is no need. A person gets active when he feels hunger. The hunger of making an automatic system leaded us to create the project called "***ASPIRINE nBARS 98.1***". The word "***Aspirin***" stands for the treatment or medicine, what a person uses to reduce his pain of diseases. Same here, we reduced our pain by developing this robot. The word "***nBARS***" denotes the members of the team who created this autonomous robot. The alphabet "***e***" relates to the development in electronics world.

During the completion of the project, we came to know and got pretty experiences which are making us more excited to express. Everything happened during the project implementation, the hard work, dedication, motivation, happiness, sorrow etc. are sharable and are points to be remembered always.

We are thankful of the institutions like ***State Board of Technical Education, Jharkhand (SBTEJ)***, ***Techno India Group (TIG)***, ***Government Polytechnic, Silli (GPS)***, ***Government Engineering College, Ramgarh (GECR)*** and ***Skill-Rex Technology, Mumbai*** for their kind support and motivation.

We are again thankful of the ***Principal, Government Polytechnic, Silli, Mr. Bishnu Brata Chatopadhyay; Mr. Nilanjan Shill; Department-In-Charge, ECE, Mr. Utpal Kumar Paul; Mr. Suprakash Jana; Mr. Chayan Chakraborty; Mr. Bishaljeet Dutta; Mr. Rupam Dutta; Mr. Debasis Roy*** and all those who directly and indirectly caused the development of our plan and for their kind support.

Saturday, June 25, 2016
Silli

The team, ASPIRINE nBARS 98.1
Students of ECE
Government Polytechnic, Silli

Contents

Introduction

The project “ASPIRINE nBARS 98.1” is basically an autonomous robot that will only move on reflective path like white colour and ignore all non-reflective dark colours. It is simply called a line follower robot. This is a microcontroller based robot.

The green point indicates the line follower robot. The street coloured as white is the path in which the robot will move. Leaving this white area, the robot will not move anywhere else.

The main objective of this project is to know how to develop such self-decision making robots using microcontrollers. This initial step could lead us to a very large scale development. This type of machines will reduce human efforts as it can perform the tasks what we ask it to do.

Components

ATmega8 microcontroller

The ATmega8 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8 achieves throughputs approaching 1 MIPS per MHz, allowing the system designed to optimize power consumption versus processing speed.

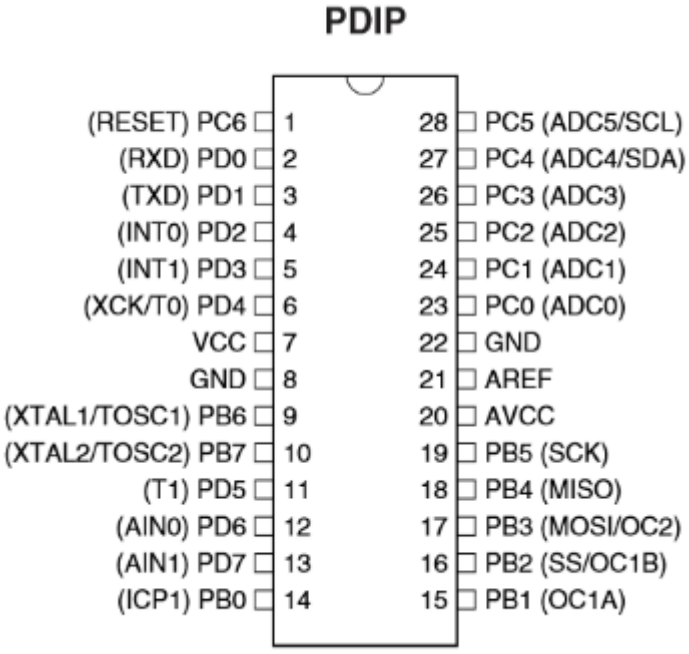
The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is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

The ATmega8 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1K byte of SRAM, 23general 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, a 6-channel ADC (eight channels in TQFP and QFN/MLF packages) with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five 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. The device is manufactured using Atmel's high density non-volatile memory technology. The Flash Program memory can be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile 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 ATmega8 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega8 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.

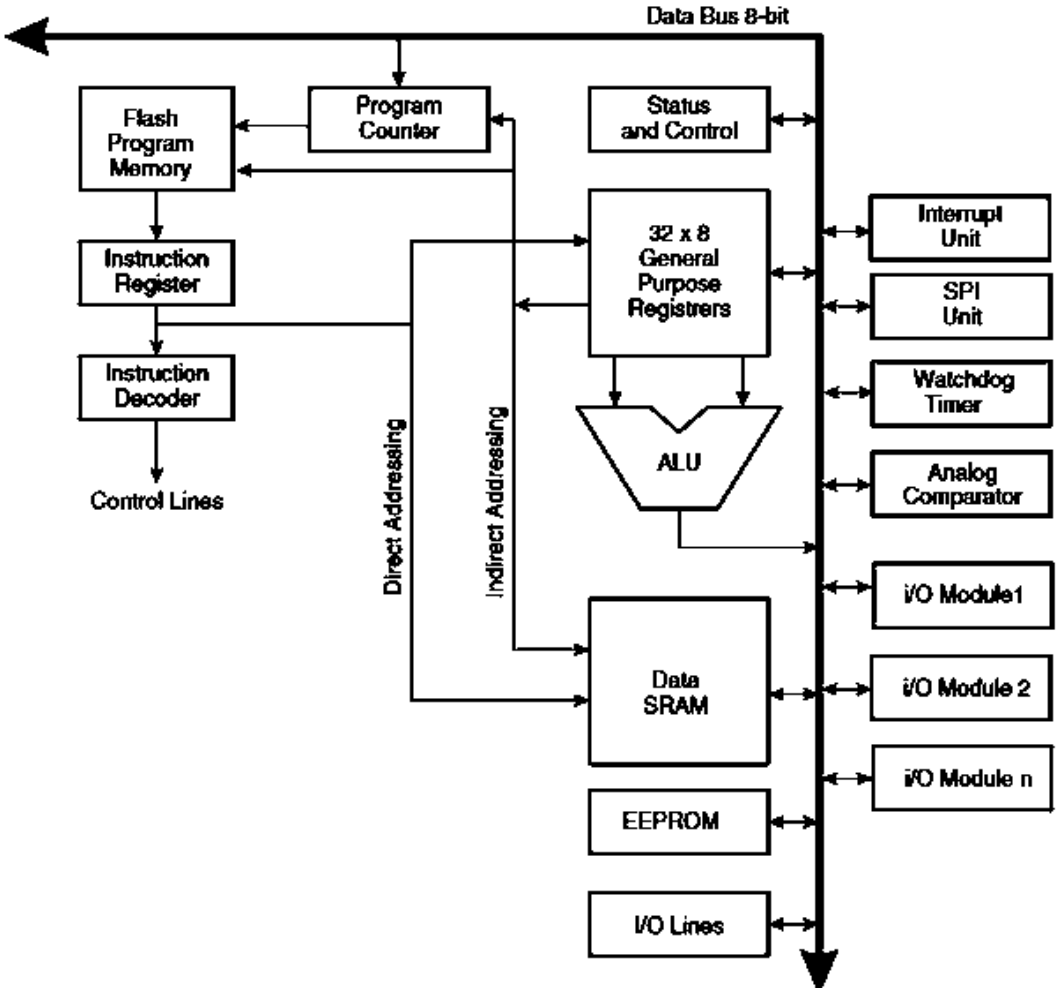
Pin configuration



VCC	Digital supply voltage.
GND	Ground.
Port B (PB7..PB0) **XTAL1/XTAL2/TOSC1/TOSC2**	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.

	Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.
Port C (PC5..PC0)	Port C is a 7-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.
PC6/RESET	If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is un-programmed, PC6 is used as a 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. The minimum pulse length is given in Table 15 on page 38. Shorter pulses are not guaranteed to generate a Reset.
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.
AVCC	AVCC is the supply voltage pin for the A/D Converter, Port C (3..0), and ADC (7..6). 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. Note that Port C (5..4) use digital supply voltage, VCC.
AREF	AREF is the analog reference pin for the A/D Converter.
ADC7..6 (TQFP and QFN/MLF Package Only)	In the TQFP and QFN/MLF package, ADC7..6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

AVR CPU core



In order to maximize performance and parallelism, the AVR uses a 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 Reprogrammable 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 address 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-register, 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.

The 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.

Arithmetic Logic Unit (ALU)

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.

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	**SRREG**
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. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The Ibit can also be set and cleared by the application with the SEI and CLI instructions, as described in the Instruction Set Reference.

- **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. See the “Instruction Set Description” for detailed information.

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two’s Complement Overflow Flag V. See the “Instruction Set Description” for detailed information.

- **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.

General purpose resource file

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input.
- Two 8-bit output operands and one 8-bit result input.
- Two 8-bit output operands and one 16-bit result input.
- One 16-bit output operand and one 16-bit result input.

The figure shows AVR general purpose working registers

	7 … 0	Addr.	
General purpose working registers	R0	0x00	
	R1	0x01	
	R2	0x02	
	…		
	R13	0x0D	
	R14	0x0E	
	R15	0x0F	
	R16	0x10	
	R17	0x11	
	…		
	R26	0x1A	X-Register Low Byte
	R27	0x1A	X-Register High Byte
	R28	0x1C	Y-Register Low Byte
	R29	0x1D	Y-Register High Byte
	R30	0x1E	Z-Register Low Byte
	R31	0x1F	Z-Register High Byte

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions. As shown, each register is also assigned a Data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer Registers can be set to index any register in the file.

X-register, X-register and X-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space.

	15 … XH		XL … 0	
X-Register	7 … 0		7 … 0	
	R27 (0x1B)		R26 (0x1A)	

	15 … YH		YL … 0	
Y-Register	7 … 0		7 … 0	
	R29 (0x1D)		R28 (0x1C)	

	15 … ZH		ZL … 0	
Z-Register	7 … 0		7 … 0	
	R31 (0x1F)		R30 (0x1E)	

In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement.

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 Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when address is popped from the Stack with return from subroutine RET or return from interrupt RETI.

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	
	SP15	S914	SP13	SP12	SP11	SP10	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	

Instruction execution timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk CPU, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

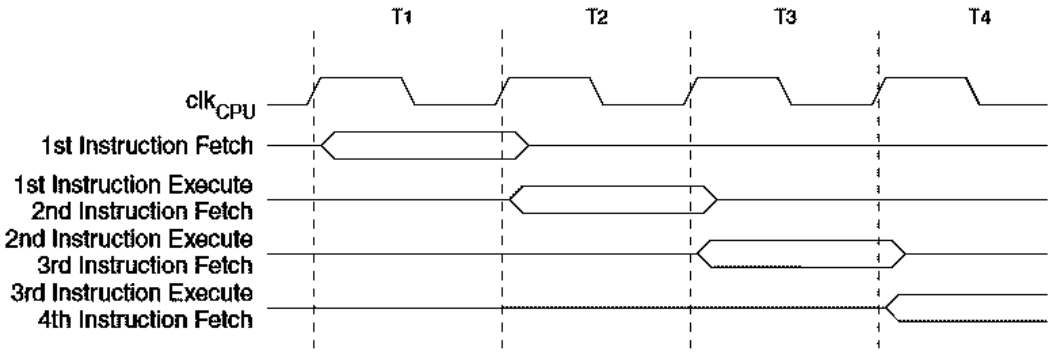
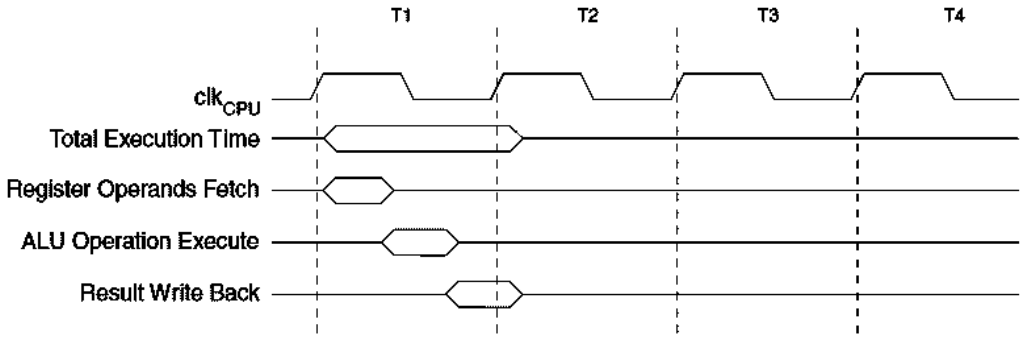


Figure shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.



Reset and interrupt handling

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate Program Vector in the Program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock Bits BLB02 or BLB12 are programmed. This feature improves software security.

The lowest addresses in the Program memory space are by default defined as the Reset and Interrupt Vectors. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the boot Flash section by setting the Interrupt Vector Select (IVSEL) bit in the General Interrupt Control Register (GICR). Refer to “Interrupts” on page 46 for more information. The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST Fuse.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed. There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the global interrupt enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in r16, SREG ; store SREG value
cli ; disable interrupts during timed sequence
sbi EECR, EEMWE ; start EEPROM write
sbi EECR, EEWE
out SREG, r16 ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG; /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMWE); /* start EEPROM write */
EECR |= (1<<EEWE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in the following example.

Assembly Code Example

```
sei ; set global interrupt enable
sleep; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```
_SEI(); /* set global interrupt enable */
_SLEEP(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

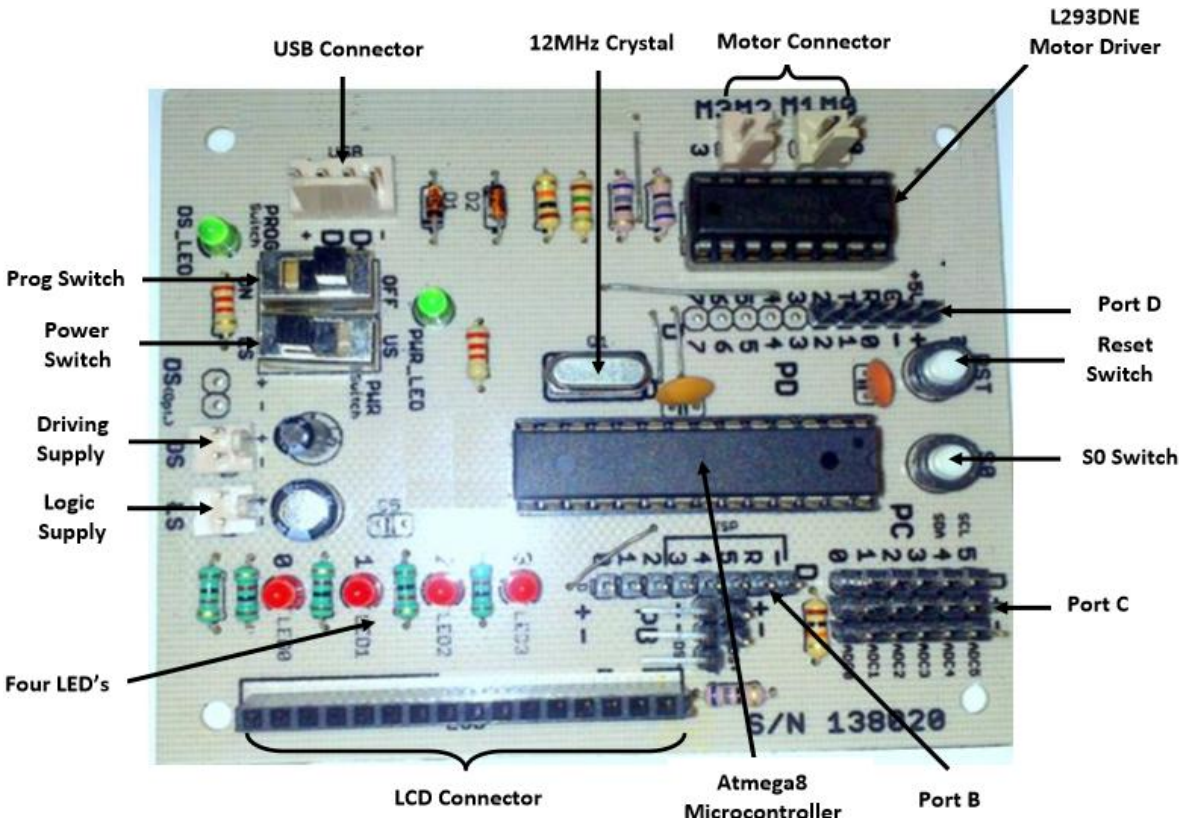
Interrupt response time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles, the Program Vector address for the actual interrupt handling routine is executed. During this 4-clock cycle period, the Program Counter is pushed onto the Stack. The Vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (2 bytes) is popped back from the Stack, the Stack Pointer is incremented by 2, and the I-bit in SREG is set.

AVR Development Board S/N: 138020

- Includes Atmel's ATmega8 Microcontroller with 8kb flash memory working at 16MIPS.
- On-board LCD interfacing option (it can also be used for any other general purpose application).
- On-board Motor Driver for connecting 2 DC motors or 1 Stepper motors.
- On-board 5v regulated power supply.
- 12 MHz external crystal.
- Exposed all 21 I/O pins.
- Exposed 6 channel I/O pins for ADC with 5V/1A power supply.
- Exposed 8 I/O channels for sensors and other peripherals with 5V/1A power supply.
- One tact switches for external input and one tact switch for reset.
- Four test LEDs for status and debugging purpose.
- Two supply indicator LEDs.
- Dual power supply through DC source (6V to 16V) or USB powered.
- On board USB programming feature.
- Exposed ISP pins for programming.
- Separate power supply option for microcontroller and motor driver.



Atmega8 IC/Chip

It is a microcomputer chip which stores user programs, executes them and takes necessary action. The chip used here is a popular AVR microcontroller of Atmel.

1117 VOLTAGE REGULATOR

It is a three terminal 5V voltage regulator IC used to provide a constant voltage supply of 5V to the microcontroller and other peripherals (i.e. sensors etc.) attached in the main board.

L293DNE

This is basically a motor driver IC which takes input from microcontroller and is able to drive the DC and stepper motors by using separate power supply.

16 X 2/16 X 1 LCD INTERFACE

The LCD interface can be used to interface any 16x2 or 16x1 character LCD displays in 4 bit mode. The LCD display can be used to display any message, status or also can be used for debugging purpose. The LCD interfacing can also be used as a general purpose input output port. The pin connections for interfacing a LCD to the board are given below:

RS	Port B4
E	Port B5
D4	Port B0
D5	Port B1
D6	Port B2
D7	Port B3

SWITCHES

One tact switch along with a Reset switch is present on the board in order to provide an external input to the board. The S0 tact switch is connected in the following manner:

S0	Port C0

RST (Reset Switch): The Reset switch is basically used to reset a running program right to the beginning; it is same as the reset switch of a PC.

RST - Reset Switch - PortC6

POWER (Power toggle Switch): It is basically a toggle switch used to provide power supply to the main board. The power can be supplied either by a battery power supply (through LS) or can be USB powered. Thus, the POWER switch can be made to toggle between LS (logic Supply) or UP (USB Supply).

PROG (Programming Switch): It is also a toggle switch for programming the microcontroller using on board USB programmer. For programming mode, it should be ON then RESET button should be pressed. For normal operation it should be OFF.

POWER SUPPLY LS (Logic Supply)

It consists of two pins one is + ve and another is – ve. A battery or an AC adaptor can be connected here to provide power supply to the mother board it provides regulated power supply to all the peripherals present in the mother board and also to the external peripherals connected to the motherboard through a voltage regulator. The DC voltage provided to this terminal should be lies in between 6 to

16 volt. To use the supply connected in LS pin the power switch should be toggled towards “LS” (Logic Supply).

DS (Driving Supply): It consists of two pins one is + ve and another is – ve. It is basically used to provide a separate high current power supply to the Motors. For operating DC motors you may provide here a Power supply of 5 to 24 volt, 1amp.

USB Connector: It is basically used for USB communication with the PC. It also provides necessary logic supply to the motherboard. In order to use the USB supply the POWER switch should be toggled towards US (USB Supply). When using the USB power some precautions should be taken such as any heavy load should not be connected to the board directly.

LED’s

Active high

LED0	Port B0
LED1	Port B1
LED2	Port B2
LED3	Port B3
POWER_LED	Logic power ON indicator
DS_LED	Driving power ON indicator

ISP (IN-SYSTEM PROGRAMMING) INTERFACE

It is the In-System Programming interface of the main board which can be used to connect any ISP programmer to download the programs in the microcontroller. It can also be used in SPI (Serial Peripheral Interface) communication. The pins provided for ISP are given below:

MO-Master in slave out	Port B3
MI-Master in slave in	Port B4
SCK-Serial clock	Port B5
RST-Reset	Reset
GND-Ground	Ground

PB (PORTB)

It is a general purpose I/O port. This port contains eight pins that can be used as digital input and digital output. Some pins of this port are in the form, DATA-VCCGROUND (denoted as D + - respectively on the board) and also one DS+ i.e. driving supply positive pin (which is used to drive the motor) is present. The Data pins are towards the microcontroller.

PC (PORTC)

It is a general purpose I/O port. This port contains seven pins that can be used as digital input, ADC in and digital output. These pins are in the form, DATA-VCCGROUND (denoted as D + - respectively on the board). The VCC and Ground pins are provided with a 5V/1A power supply.

PD (PORTD)

It is a general purpose I/O port. This port contains eight pins that can be used as digital input and digital output. The two external interrupt pins INT0 & INT1 which are there on port D at pin number 2 & 3 respectively. The UART pins are also available on this port.

MOTOR DRIVER CONNECTIONS

The motor driver is used to run the DC motors/stepper motor that may be connected to the board according to the data from the microcontroller. The motor driver links with microcontroller is as shown below:

M0	Port D4
M1	Port D5
M2	Port D6
M3	Port D7

The AVR Microcontrollers Description

The AVR is a Modified Harvard architecture 8-bit RISC single chip microcontroller which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to One-Time Programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time. Atmel's low power, high performance AVR microcontrollers handle demanding 8 and 16-bit applications. With a single cycle instruction RISC CPU, innovative Pico Power® technology, and a rich feature set, the AVR architecture ensures fast code execution combined with the lowest possible power consumption. Whether you program in C or assembly, the tuned AVR instructions decrease program size and development time. The well-defined I/O structure limits the need for external components and reduces development cost. A variety of internal oscillators, timers, UARTs, SPIs, Pulse Width Modulation, pull-up resistors, ADCs, Analog Comparators and Watch-Dog Timers are some of the features available for creative engineers. The AVR microcontrollers are divided into 4 families tiny AVR, mega AVR, XMEGA and Application specific AVR. Among these 4 families of AVR here we are going to use a microcontroller of mega AVR family “ATmega8”.

Programming

WinAVR is a suite of executable, open source software development tools for the Atmel’s AVR series of RISC microcontrollers hosted on the Windows platform. It includes the GNU GCC compiler for C and C++. Steps for writing a code using WinAVR:

1. Open the Programmer’s Notepad and write your code.
2. Create a new folder and save your code in that folder with extension name “.c” & minimize the notepad
3. Now open the make file and edit it as mentioned bellow:
   3.1 Make file→ main filename (give your file name here without extension)
   3.2 Make file→ MCU type→ ATmega8→ (choose your UC)
   3.3 Make file→ enable editing make file→ then in your make file edit the following things
   3.4 F_CPU = 12000000 (change it as for your crystal frequency)
   3.5 Save the make file in your folder (above you have created) without changing its name.
4. Now maximize the programmer’s notepad.
5. To compile your code and to generate hex file go Tools→ make all.

Motherboard S/N: 138020 configuration

16x2/16x1 LCD

RS	Port B4
E	Port B5
D4	Port B0
D5	Port B1
D6	Port B2
D7	Port B3

LED interface (active high)

RED LED	Port B0
RED LED	Port B1
RED LED	Port B2
RED LED	Port B3

Switch (active low)

S0	Port C0

Motor driver

M0	Port D4
M1	Port D5
M2	Port D6
M3	Port D7

Analog to Digital Converter (ADC)

PC0 to PC5

I2C

SCL	PC5
SDA	PC5

RS232/UART

R-Receiver	PD0
T-Transmitter	PD1
G-Ground	

ISP

MO-Master in slave out	Port B3
MI-Master in slave in	Port B4
SCK-Serial clock	Port B5
RST-Reset	Reset
GND-Ground	Ground

IR Sensors

Construction

The sensors are very simple to understand and easy to implement. The major components of these sensors include a High glow Red LED and a Photodiode. The LED emits light when energized; hence acting like an optical transmitter. Whereas the photodiode is capable of detecting light incident on it, thereby acting like an optical receiver. Thus, these sensors work on the principle of light sensing.

Working

The LED emits high intensity red light when current flows through it. If the emitted light of LED falls on any bright surface it is reflected back to the photo diode. Photo diode is a device which is able to detect the light intensity and convert it to a corresponding electric current. Other devices present in the board read the electric current generated from the photodiode and convert it to a logic voltage level which is either logic-0 or logic-1. Thus, depending on the presence or absence of the surface, logic data is developed by the light sensor.

Applications

The MPOs can be used for a number of applications, some of which have been mentioned below:

OBSTACLE SENSOR: In order to use this sensor as an Obstacle sensor mount the optical sensor on side of your robot facing them horizontally with the ground surface. Take care that your sensor is affected by any other bright light source which may affect the photodiode. Place any obstacle in front of your sensor (it is better if the obstacle is of white or any other light colour). When your sensor detects the obstacle it will send a data of logic-0 through the data pin and when the obstacle is absent, it will send a data of logic-1 through the data pin.

LINE SENSOR: In order to use this sensor as Line sensor, take a white chart paper and draw a black line on it (with water colour preferably) or take a black chart paper and draw a white line on it. Place the sensor perpendicular to the ground level facing the ground and maintain a clearance of near about 1 to 2 centimeter between the sensor and the ground. When the sensor passes over a black surface it will send a data of logic-1 on data pin and when it passes over a white surface it will send a logic-0 on data pin. Thus, a sensor or two may be able to follow a line while detecting any turn or curve along the line with a change in the logic data sent to the data pin.

LIGHT SENSOR: This sensor can also be used to detect any other source of light. In order to use this sensor to detect a light source, it is required to cover the high glow red LED so that it may detect the light due to reflection. If light falls on it, it will send a data of logic-0 on data pin and when less light falls on it, it will send a logic-1 on data pin.

Configuration

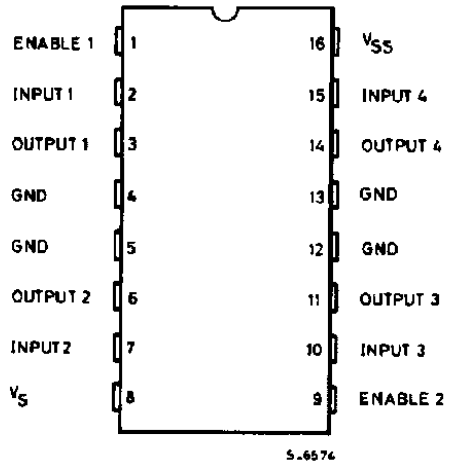
In the Multi-Purpose Optical Sensor, a 3-pin male connector or a berg strip is present. The three pins of this berg strip are D (Data), + (positive 5 V supply), - (ground). Supply to the sensor is provided by the logic supply that may be present on the mother board. In order to connect this sensor to the mother board you need a 3-band connector cable. Connect one side of the connector cable to the sensor, check the convention of D, + and -, and then connect the other side of the sensor cable to the motherboard by following the same convention of D, + and -. (IMPORTANT NOTE - While connecting the sensors, never do mistake of connecting with a wrong convention of D, + and – mentioned on motherboard and sensor.)

Driver IC (L293D)

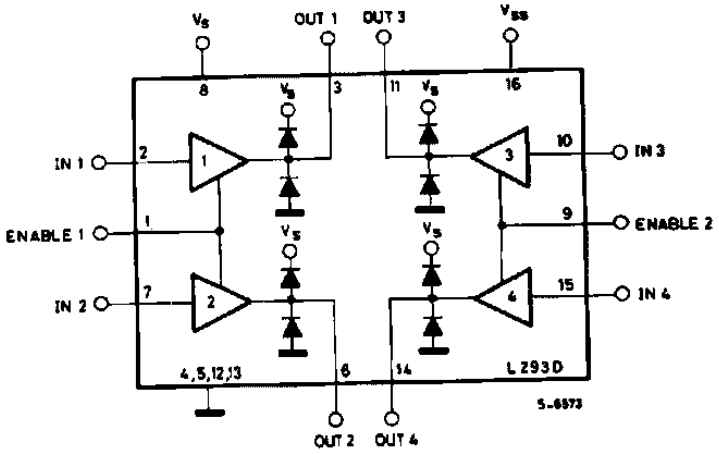
Simply knows as PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES.

The L293D is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoids, DC and stepping motors) and switching power transistors. To simplify use as two bridges is pair of channels is equipped with an enable input. A separate supply input is provided form the logic, allowing operation at a low voltage and internal clamp diodes are included.

Pin configuration



Block diagram



Servo motors

Servo motors used here are basically dc motors which can rotate in both clockwise and anticlockwise directions.

Clockwise, if V1>V2 | Anticlockwise, if V1<V2 | No rotation, if V1=V2

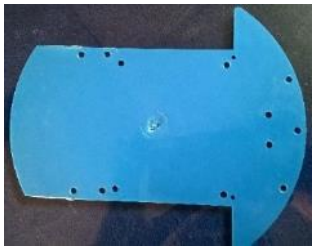
Power source

Here we have used two 9V dc cells to conduct the microcontroller motherboard, servo motors, IR sensors, servo ICs and various components of the board. LS (means Logic source) and DS (means Driver source) are two dc power sources. Since the single 9v cell is not enough capable to run both the motors and even do all circuit operations of this robot.

Physical assembly

Major parts of robot

Base plate



Motor clamp

Caster wheel

Wheel

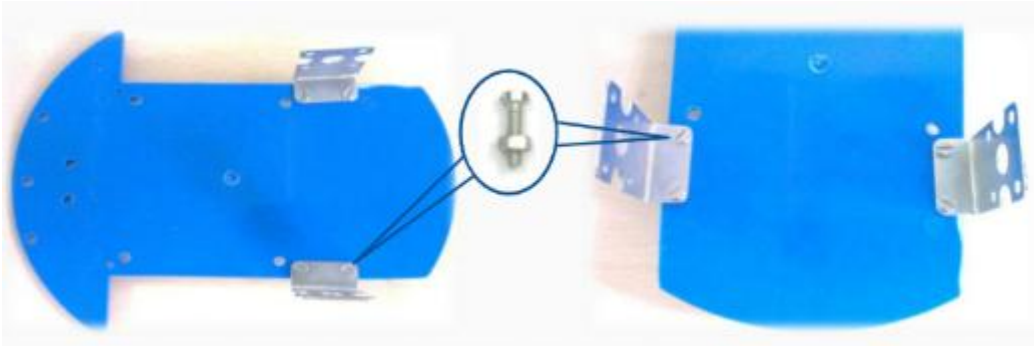
Various screws and nut

Seperators (use to mount ATmega8 motherboard with the base plate

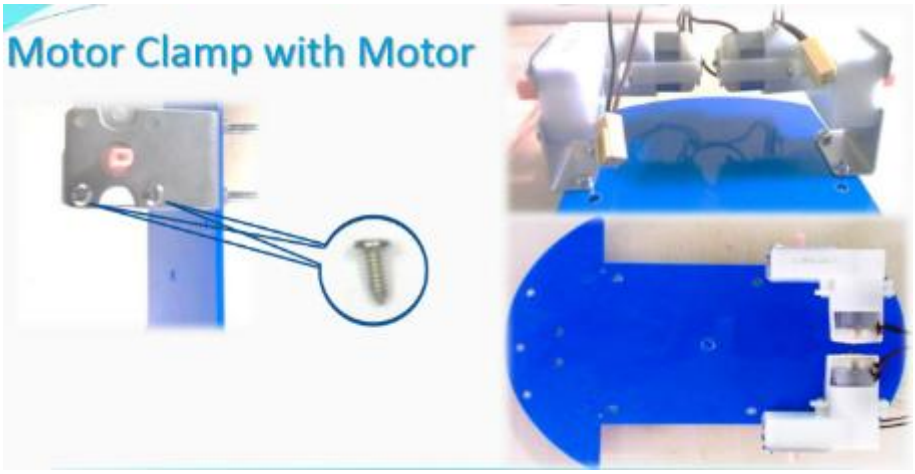
Assembly

The physical assembly of the "***ASPIRINE nBARS 98.1***" is done as per the following steps given below:

Step 1: Take Base Plate and Motor Clamp with Nut Bolts fix them.



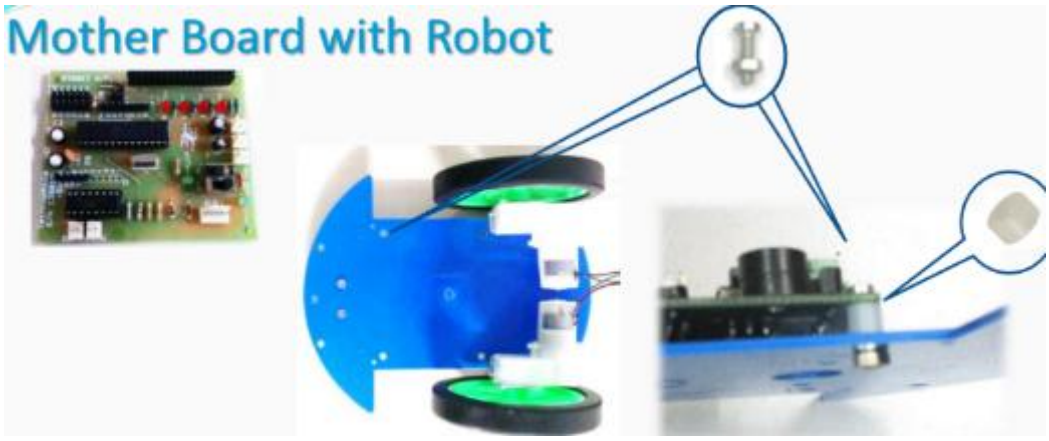
Step 2: Connect both the motors using Thick & Long Screw.



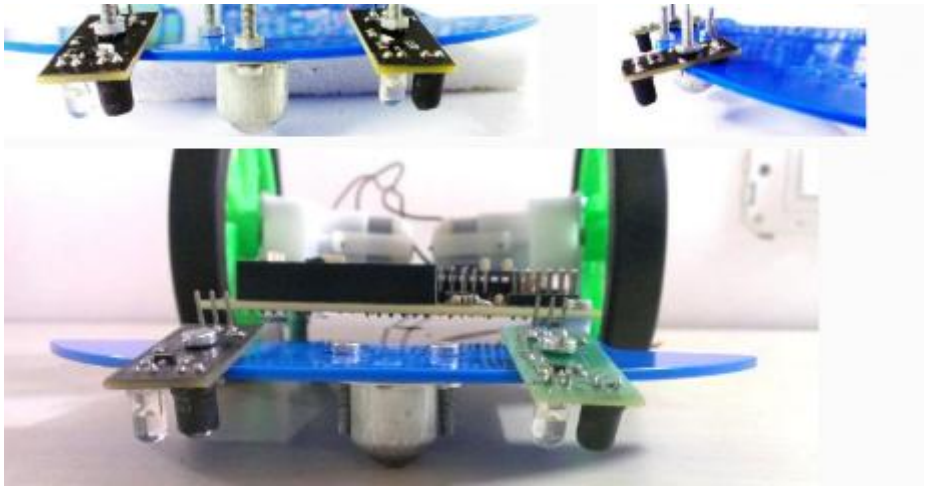
Step 3: Fix wheels to the shaft of the motors using the thin & long screw. Don't tight it more leaves it in the wheel level.

Step 4: Connect the caster wheel.

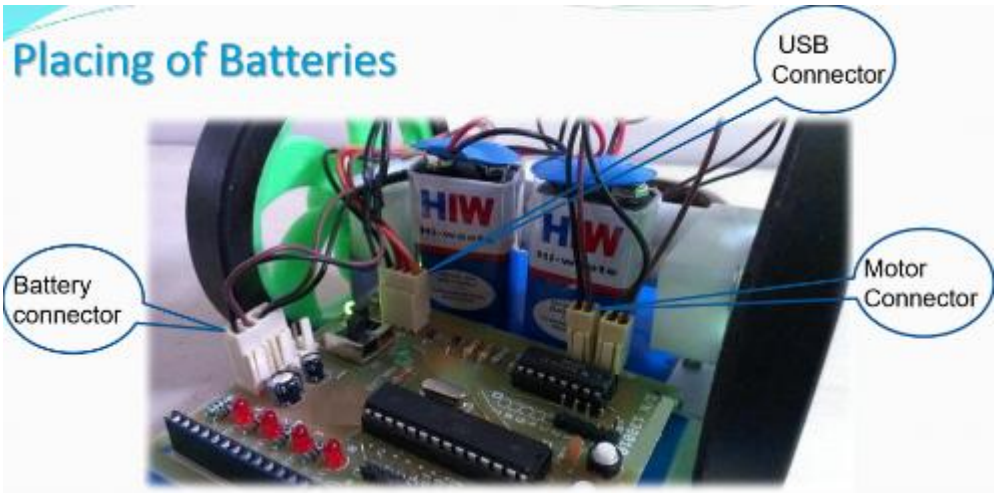
Step 5: Fix motherboard on the Base plate by using nut-bolt & spacers, make sure that motor connectors of the mother board & motor wires are in the same side.



Step 6: Connect IR sensors in front side of the robot using 3-wire ribbon.



Step 7: Place the batteries and connect all connectors as mention above.

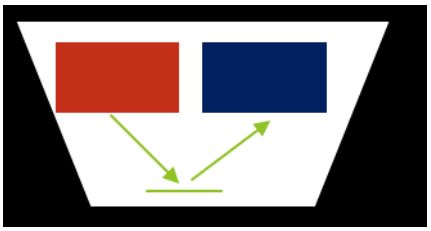


Working and explanations

Working of an IR sensor

The robot here follows the white path. Here we are using the white so that the sensor will generate the said voltage whenever it will sense the white path. This will also limit the power consumption. The motor of the robot should rotate only if the sensors (or either a sensor) senses the white path (where reflection of light source occurs).

As the working of IR sensors are, the IR LED emits (red box) an infrared light. If there is any reflective source before the sensor, the infrared light reflects and is sensed by the photodiode (blue box). This photodiode generates respective voltage as per the amount of light received.



Now when the robot is turned ON, as it senses a reflective path like white coloured part it starts moving on and stops if there is fully non-reflective path.

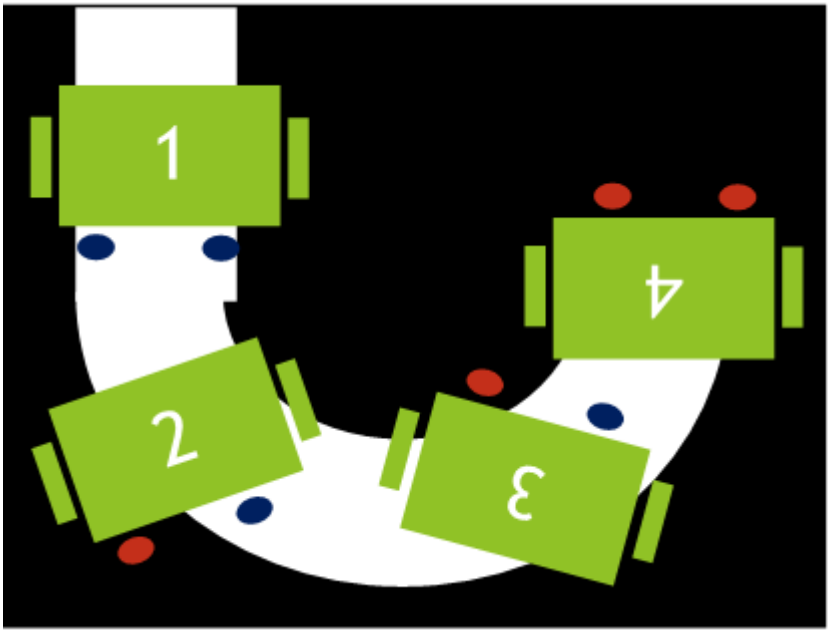
How the robot moves

Let us consider the figure shown below. Here when the robot is moving on a straight reflective path, both the sensors are receiving (both sensors are blue now) the reflected infrared light source.

In the next position, where the left sensor is blue (i.e., receiving the reflect IR light) and the right one is red (no reception of light). This case the motors will be commanded to move a left turn. Kindly remember that at every instant the microcontroller would try to make both the sensors blue (i.e., to move as in a straight line).

In the third stage, the left sensor is red and the right one is blue. This time the motor will rotate the robot towards right direction.

At last when it receives no reflective path, i.e., both sensors are red, the robot stops moving anywhere.



Robot wheels and its rotation

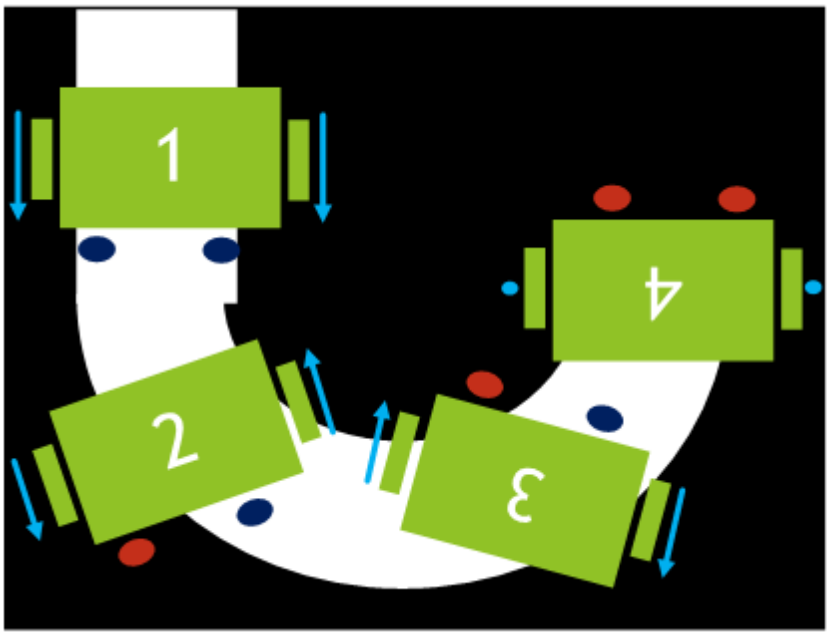
As described earlier, the dc motor requires a dc power supply to rotate either in clockwise of anti-clockwise direction. The robot consists of two dc motors connected oppositely to each other.

The figure shown below, elaborates how the motors rotate of different cases.

In case 1, both motors are moving in forward direction. Since both motors are opposite to each other then we need to provide opposite signals to each motors so that the motors could rotate in same direction.

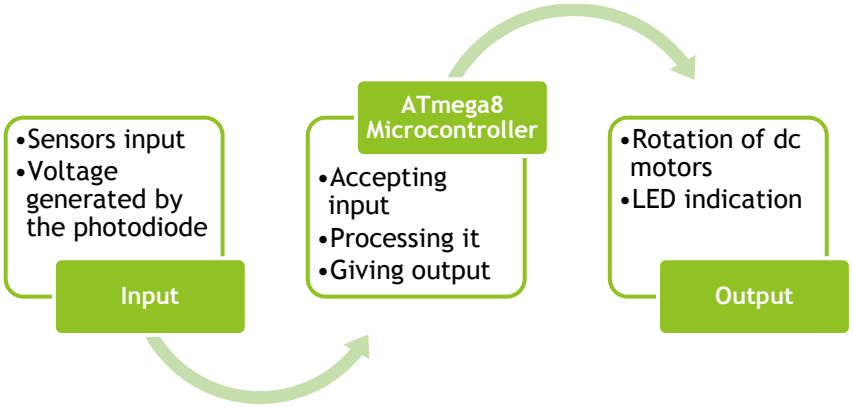
In case 2 and 3, both motors are rotating opposite to each other. So, for case 2 we will rotate the right motor in forward direction and the left motor in reverse direction. In same manner in case 3, forward direction to left motor and reverse direction for right motor.

Case 4 does not require any movement of motors.



All on the microcontroler

The all above operations will take place as per the diagram shown below



As per the input supplied by the sensors, the microcontroller will generate such output.

Software program

To actually make the robot work, we need to provide instructions to the robot. The microcontroller of the robot needs to be programmed so that it operates as we want. The machine understands basically the low level language or 0s and 1s. These 0s and 1s bring complexities to us to write number of instructions to the machines. Thus a mid-level language or a high level language helps us to program such machines easily. Mid-level languages are codes which are partly understood by the human beings. These must be kept remembered in mind. The high level language is a language which we use to talk to others. Not so pretty much, since it has some limitations. But it makes our programming easier, so that machines could understand us easily.

Syntax and hex codes

Before going to the programming following hex code generation are necessary to understand. 0 is used for input and 1 for output.

	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0
DDRX	0	0	0	0	0	0	0	0
PORTX	0	0	0	0	0	0	0	0
PINX	0	0	0	0	0	0	0	0

DDR, Data Direction Register
PORT-for O/P operations
PIN-for I/P operations
X-Port name

Sensor's responses

The following tables tells the microcontroller about the type of sensor's responses

When both sensors are experiencing

	7	6	5	4	3	2	1	0	Type	Hex code
DDRC	0	0	0	0	0	0	0	0	Input	0x00
PINC	0	0	0	0	0	0	0	0	Input	0x00

When left sensor experiences

	7	6	5	4	3	2	1	0	Type	Hex code
DDRC	0	0	0	0	0	0	0	0	Input	0x00
PINC	0	0	0	0	0	0	0	1	Input	0x01

When right sensor experiences

	7	6	5	4	3	2	1	0	Type	Hex code
DDRC	0	0	0	0	0	0	0	0	Input	0x00
PINC	0	0	0	0	0	0	1	0	Input	0x2

When both experience no reflective path/surface

	7	6	5	4	3	2	1	0	Type	Hex code
DDRC	0	0	0	0	0	0	0	0	Input	0x00
PINC	0	0	0	0	0	0	1	1	Input	0x03

Movements

Forward

	7	6	5	4	3	2	1	0	Type	Hex code
	M3	M2	M1	M0						
DDRD	1	1	1	1	0	0	0	0	Output	0xF0
PORTD	0	1	1	0	0	0	0	0	Output	0x60

Backward

	7	6	5	4	3	2	1	0	Type	Hex code
	M3	M2	M1	M0						
DDRD	1	1	1	1	0	0	0	0	Output	0xF0
PORTD	1	0	0	1	0	0	0	0	Output	0x90

Left

	7	6	5	4	3	2	1	0	Type	Hex code
	M3	M2	M1	M0						
DDRD	1	1	1	1	0	0	0	0	Output	0xF0
PORTD	0	1	0	1	0	0	0	0	Output	0x50

Right

	7	6	5	4	3	2	1	0	Type	Hex code
	M3	M2	M1	M0						
DDRD	1	1	1	1	0	0	0	0	Output	0xF0
PORTD	1	0	1	0	0	0	0	0	Output	0xA0

No movement

	7	6	5	4	3	2	1	0	Type	Hex code
	M3	M2	M1	M0						
DDRD	1	1	1	1	0	0	0	0	Output	0xF0
PORTD	0	0	0	0	0	0	0	0	Output	0x00

LED's responses

To glow 1st LED

	7	6	5	4	3	2	1	0	Type	Hex code
DDRB	0	0	0	0	1	1	1	1	Output	0x0F
PORTB	0	0	0	0	0	0	0	1	Input	0x01

To glow 2nd LED

	7	6	5	4	3	2	1	0	Type	Hex code
DDRB	0	0	0	0	1	1	1	1	Output	0x0F
PORTB	0	0	0	0	0	0	1	0	Input	0x02

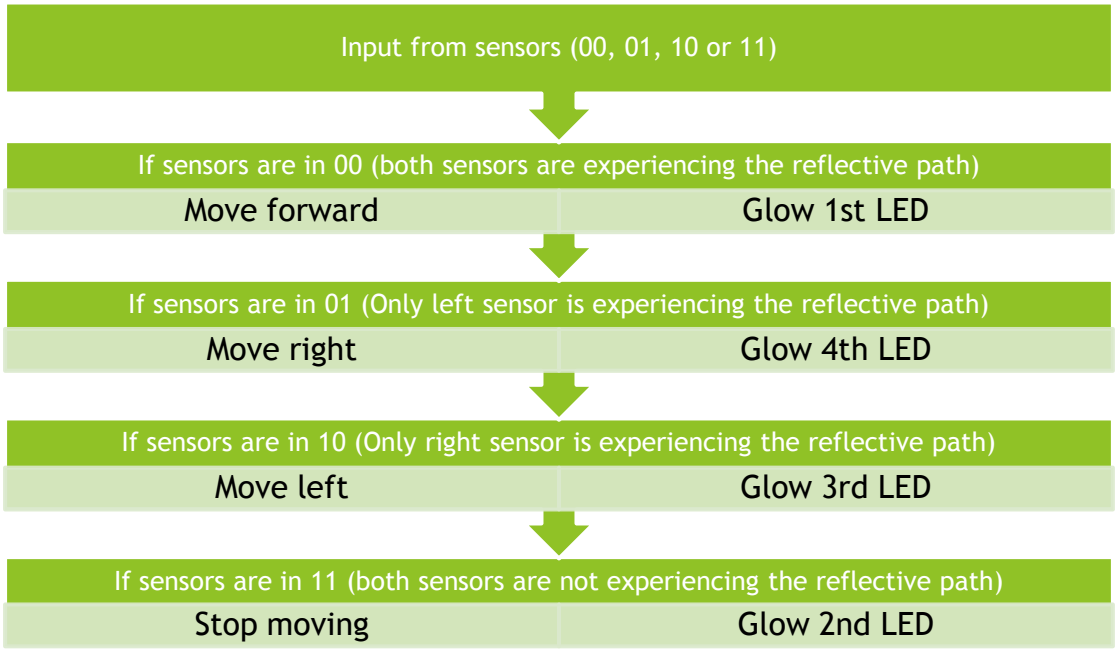
To glow 3rd LED

	7	6	5	4	3	2	1	0	Type	Hex code
DDRB	0	0	0	0	1	1	1	1	Output	0x0F
PORTB	0	0	0	0	0	1	0	0	Input	0x04

To glow 4th LED

	7	6	5	4	3	2	1	0	Type	Hex code
DDRB	0	0	0	0	1	1	1	1	Output	0x0F
PORTB	0	0	0	0	1	0	0	0	Input	0x08

Flow chart



The programming steps

Here we are using the programmer's notepad to write the codes. Of course, this is in high level language.

We have connected the motherboard (S/N: 138020) consisting the Atmega8 microcontroller of the robot to a PC using a USB cable.

After programming we saved it as "main.c" in a folder where a file named "makefile" is already existing which contains the information of various process, instructions and hardware details. Also compiled this "main.c" file.

As the robot is connected with PC, the program switch of the robot must be set to "ON" and the Power switch must be set to "US" while installing the program. When done, run "HIDBootFlash". Click on "Find Device", then after open the ".hex" file which is automatically generated in the folder where "main.c" and "makefile" were existed. Now click on "Flash Device". When done successfully, the robot has got the program and is now ready for its operation.

Program code

```
#include<avr/io.h>

intmain()
{
DDRB=0x0F;              //initialization of LEDs port (O/P device)
PORTB=0x00;             //clearing the port

DDRC=0x00;              //initialization of sensors port (I/P device)
PORTC=0x00;             //clearing the port

DDRD=0xF0;              //initialization of Motors port (O/P device)
PORTD=0x00;             //clearing the port

while(1)
{
switch(PINC & 0x03)                 //read both the sensors
          {
          case 0x00:                //if both the sensors read
          PORTB=0x01;               //glow 1st LED
          PORTD=0x60;               //move forward
          break;

          case 0x01:                //if left sensor read
          PORTB=0x08;               //glow 4th LED
          PORTD=0xA0;               //move right
          break;

          case 0x02:                //if right sensor read
          PORTB=0x04;               //glow 3rd LED
          PORTD=0x50;               // move left
          break;

          case 0x03:                //if both the sensors do not read
          PORTB=0x02;               //glow 2nd LED
          PORTD=0x00;               //stop moving
          break;
          }

}
}
```

Conclusion

All connections have been made as per the required operation. Proper programming codes have been installed. The robot is now running as per the desired operation. It is ignoring the non-reflective surfaces and moving only in white path (reflective surface, where IR is reflecting).

The project provided the practical experience of interfacing various components, features of a microcontroller and designing a particular program (i.e., writing programming codes).

Future aspects

The future is based on our needs. We are growing. We human being, are requiring more tools that reduce our efforts and help us to explore more. If we look back, then there was no device to wash clothes. But now it has been plenty of services. In various industries, number of microcontrollers are being applied to perform automated operations and to make effect the large scale productions.

The Indian army is the top most fighters all around the world. This thing is aware by views of all. The critical areas where landmines are installed, interrupts their activities and goals. As result, there could be increased activities of intruders and lots of territory activities. This type of autonomous robot may be used in such place so that the communication errors between the arm forces and the entire team could not occur. Also, we can keep these areas in track by being far away from such dangerous places.

Its application is also considerable in various industries to carry heavy items in its defined path while disturbing no one.

As the technology increases, the memory, number of I/O ports, core, speed etc. of the microcontroller is getting developed.

References

s.l. : Skill-Rex Technology.

Wikkipedia. [Online] www.wikkipedia.com.

Instructables. [Online] www.instructables.com.

Chakyaborty, Chayan. Hardware configuration.

Dutta, Bisaljeet.

Dutta, Rupam.

Jana, Suprakash.

Paul, Utpal Kumar. Microcontrollers. s.l. : Government Polytechnic, Silli.

Roy, Debasis.

Thanking you!

We are the team, **ASPIRINE nBARS 98.1**

Electronics and Communication Engineering
Government Polytechnic, Silli (GPS)
Silli, Ranchi, Jharkhand, India