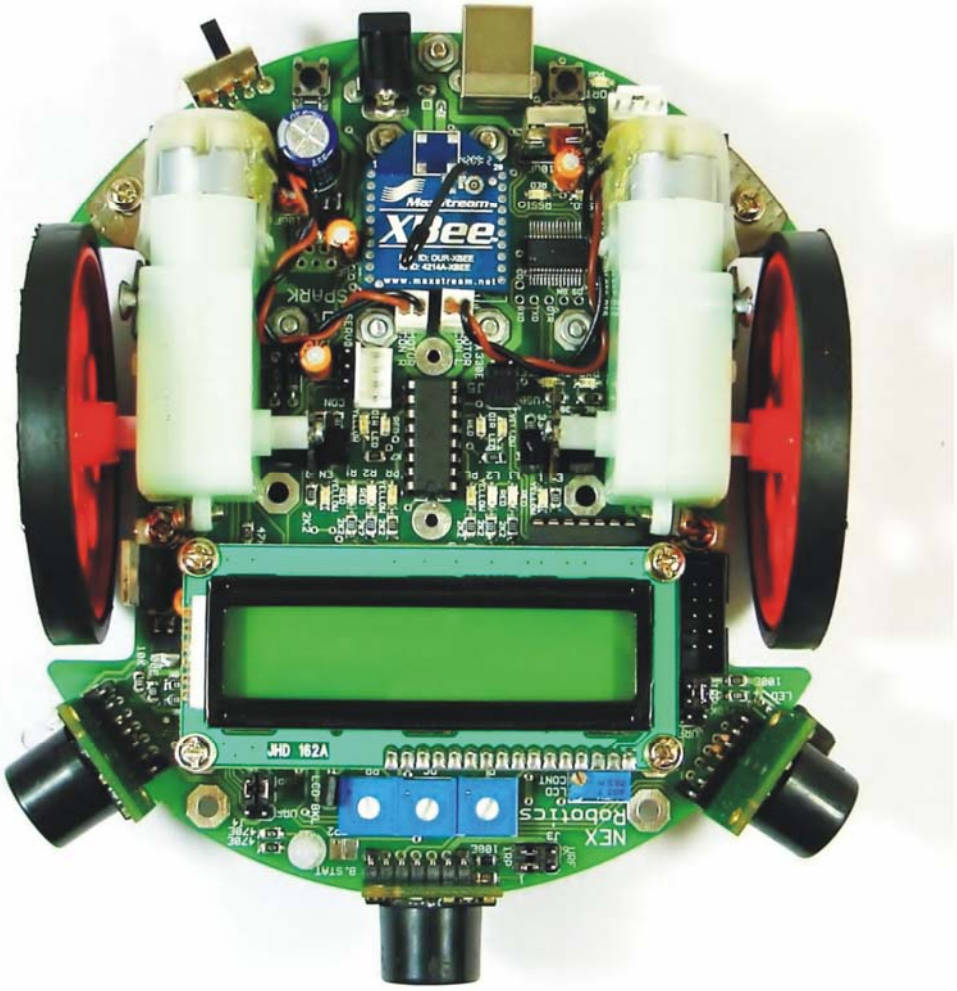
SPARK V

ATMEGA16
ROBOTIC RESEARCH PLATFORM
Software Manual



Designed By:

ERTS Lab, CSE, IIT Bombay
www.it.iitb.ac.in/~erts

NEX ROBOTICS
www.nex-robotics.com

Manufactured By: NEX Robotics Pvt. Ltd.

SPARK V

SOFTWARE MANUAL

Version 1.12
November 06, 2010

Documentation author
Sachitanand Malewar, NEX Robotics Pvt. Ltd.
Vinod Desai, NEX Robotics Pvt. Ltd.

Credits:
All the team of NEX Robotics

Notice

The contents of this manual are subject to change without notice. All efforts have been made to ensure the accuracy of contents in this manual. However, should any errors be detected, NEX Robotics welcomes your corrections. You can send us your queries / suggestions at
info@nex-robotics.com

⚠ **Robot's electronics is static sensitive. Use robot in static free environment.**
⚠ **Read the hardware and software manual completely before start using this robot**

Recycling:

Almost all of the robot parts are recyclable. Please send the robot parts to the recycling plant after its operational life. By recycling we can contribute to cleaner and healthier environment for the future generations.

Index

1. Introduction

Spark V is a low cost robot designed for robotics hobbyists and enthusiasts. It is jointly designed by NEX Robotics with Department of Computer Science and Engineering, IIT Bombay. Spark V will help you get acquainted with the world of robotics and embedded systems. Thanks to its innovative architecture and adoption of the 'Open Source Philosophy' in its software and hardware design, you will be able to create and contribute to, complex applications that run on this platform, helping you acquire expertise as you spend more time with them.

Spark V robot is based on ATMEGA16 microcontroller. It has 3 analog white line sensors, 3 analog IR Proximity sensors, 3 directional light intensity sensors and Battery voltage sensing. Robot has support for 3 MaxBotix EZ series ultrasonic range sensors. It also has support for the servo mounted sensor pod which can be used to make 180 degrees scan for the map making. Robot can be powered by 6 AA size rechargeable NiMH batteries. Robot has built-in Smart Battery Controller which charges the battery in intelligent way and also monitors the battery charge level when robot is in operation. Robot has onboard FT232 based true USB to serial TTL converter. Robot programming is done using NEX Robotics Bootloader via USB port. There is no need to use external programmer. Robot has 2x16 alphanumeric LCD, Lots of LED indicators and Buzzer etc. for quick debugging. Robot has onboard socket for XBee wireless module for multi robot and robot to PC communication. Robot has two low power 60 RPM DC geared motors which are powered by L293D motor driver with the top speed of 66cm/second.

Note: You need to buy MaxBotix EZ series ultrasonic range sensors, XBee wireless module and rechargeable NiMh Batteries separately.

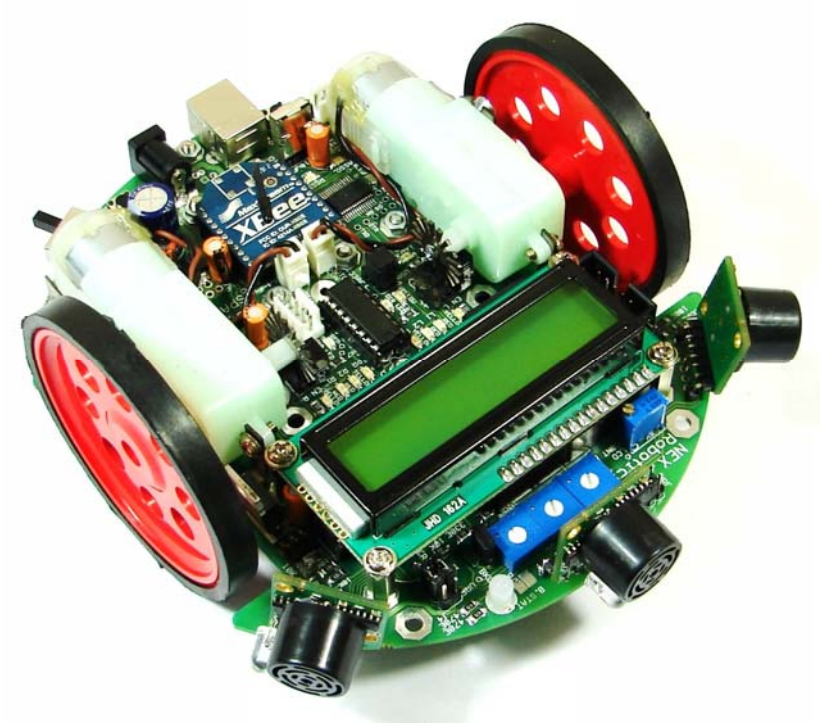


Figure 1.1: SPARK V Robot

1.1 SPARK V Block Diagram:

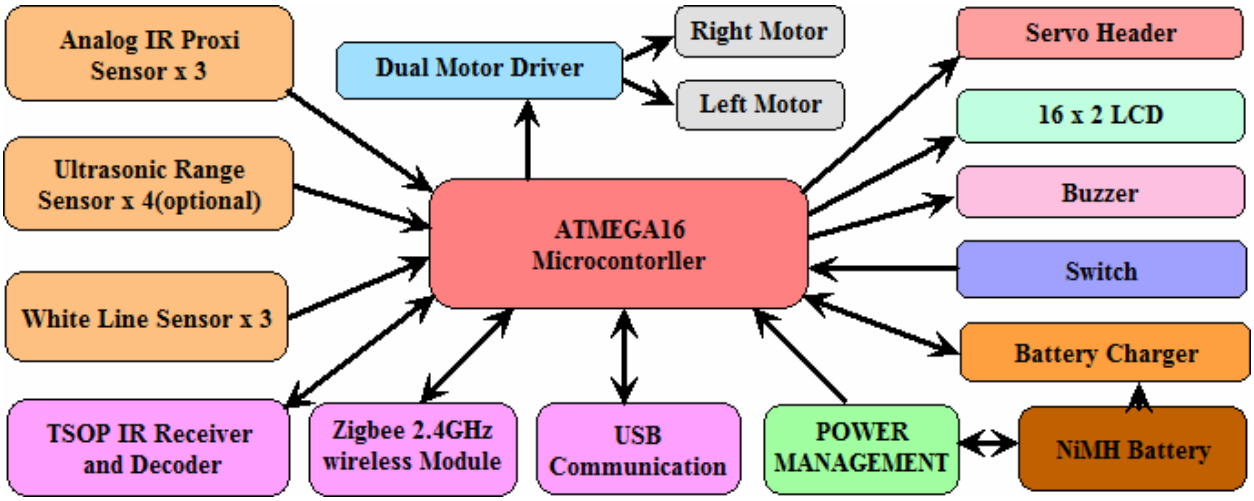


Figure 1.2: Spark V ATMEGA16 robot block diagram

1.2 SPARK V ATMEGA16 technical specification

Microcontroller: ATMEL ATMEGA16

Programming: Using NEX Robotics Boot loader via USB port (no need of separate programmer)

Sensors:
Three white line sensors
Three IR proximity sensors
Three directional light intensity sensors
Two Position Encoders (optional)
MaxBotix EZ series ultrasonic range sensors (optional)
Servo mounted Ultrasonic Range Sensor (optional)
Battery voltage sensing

Indicators:
2 x 16 Characters LCD
Indicator LEDs
Buzzer
Battery low indication

Locomotion:
Two 60 RPM DC geared motors and caster wheel as support
Built-in clutch for protection of the motors from non continuous wheel stalling.
Top Speed: 66cm/second

Operational Modes:
Standalone (Autonomous Control)
PC as master and robot as slave
Distributed (multi robot communication)

Communication:
USB Communication using FT232 USB to Serial Converter
Simplex infrared communication (From infrared remote to robot)
ZigBee (IEEE 802.15.4) (Wireless) (Robots to Robots and Robots to PCs)(Optional)

Dimensions:
Diameter: 15cm
Height: 7cm

Power:
6 AA size NiMH rechargeable batteries (Batteries not included)
Onboard Smart Battery Controller charges the battery in intelligent way and also monitors the battery charge level when robot is in operation.

Locomotion:
Two 60 RPM DC geared motors and caster wheel as support
Built-in clutch protection for the motors from non continuous stalling of the wheel
Top Speed: 66cm/second

Optional Accessories:
Servo mounted Ultrasonic range sensor for 180 degree scan
Servo mounted directional light intensity sensor for 180 degree scan
Two position encoders
MaxBotix EZ series ultrasonic range sensors
XBee wireless module

Software Support:
GUI Based control, AVR studio, WINAVR
Microsoft robotics studio Visual Programming Language (will be launched shortly)

Requires:
AC adaptor with exact 12VDC with 1Amp. current rating for battery charging.
6 NiMH rechargeable batteries

Note: Refer to Chapter 4 from the Hardware Manual for loading firmware on the robot

2. Input / Output Operations on the Robot

ATMEGA16 microcontroller has four 8 bit ports from PORT A to PORT D. Input/output operations are the most essential, basic and easy operations.

We will need frequent I/O operations mainly to do following tasks:

Function	Pins	Input / Output	Recommended Initial State
Robot Direction control	PB0 to PB3	Output	Logic 0
LCD display control	PC0 to PC2 PC4 to PC7	Output	Logic 0
Boot / I/O switch	PD6	Input	Pulled up*
Buzzer	PC3	Output	Logic 0

Table 2.1

Note:
* In the AVR microcontrollers while pin is used as input it can be pulled up internally by using software enabled internal pull-up resistor. This internal pull-up as name indicates pulls up the floating pin towards Vcc. This makes input pin less susceptible to noise.

2.1 Registers for using I/O PORTs of the ATMEGA16 Microcontroller

Each pin of the port can be addressed individually and can be configured as input or output. While pin is input it can be kept floating or pulled up by using internal pull-up. While pin is in the output mode it can be logic 0 or logic 1. To configure these ports as input or output each of the port has three associated I/O registers. These are Data Direction Register (DDRx), Port Drive Register (PORTx) and Port pins register (PINx) where 'x' is A to D indicating particular port name.

A. Data Direction Register (DDRx)

Data direction register is used to set which bits of the port are used for input and which bits are used for output. If logic one is written to the pin location in the DDRx, then corresponding port pin is configured as an output pin. If logic zero is written to the pin location in the DDRx, then corresponding port pin is configured as an input pin.

```
DDRA = 0xF0; //sets the 4 MSB bits of PORTA as output port and
             //4 LSB bits as input port
```

B. Port Drive Register (PORTx)

If the port is configured as output port, then the PORTx register drives the corresponding value on output pins of the port.

```
DDRA = 0xFF; //set all 8 bits of PORTA as output
PORTA = 0xF0; //output logic high on 4 MSB pins and logic low on 4 LSB pins
```

For pins configured as input, we can instruct the microcontroller to apply a pull up register by writing logic 1 to the corresponding bit of the port driver register.

```
DDRA = 0x00; //set all 8 bits of PORTA as input
PORTA = 0xF0; //pull-up registers are connected on 4 MSB pins and 4 LSB pins are floating
```

C. Port pins register (PINx)

Reading from the input bits of port is done by reading port pin register

```
x = PINA; //read all 8 pins of port A
```

DDRx	PORTx	I/O	Pull-up	Comments
0	0	Input	No	floating input
0	1	Input	Yes	Will source current if externally pulled low
1	0	Output	No	Output Low (Sink)
1	1	Output	No	Output High (source)

Table 2.2

Note:

- 'X' represents port name – A, B, C, D
- Tri-State is the floating pin condition.
- For more details, refer to ATMEGA16 datasheet which is located in the "Datasheets" folder in the documentation CD.

Example:

Make PORTA 0-3 bits as output and PORTA 4-7 bits input.
Add pull-up to pins PORTA 4 and PORTA 5.
Output of PORTA 0 and PORTA 2 = 1; PORT A 1 and PORT A 3 = 0;

Pin	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
DDRA	0 (i/p)	0 (i/p)	0 (i/p)	0 (i/p)	1 (o/p)	1 (o/p)	1 (o/p)	1 (o/p)
PORTA	0	0	1 (↑)	1 (↑)	0	1	0	1
Status	i/p Floating	i/p Floating	i/p Pull-up	i/p Pull-up	o/p Low	o/p High	o/p Low	o/p High

Table 2.3

```
{
unsigned char k;
DDRA = 0x0F; //Make PA4 to PA7 pins input and PA0 to PA3 pins output
PORTA = 0x35; //Make PA7, PA6 floating; PA5, PA4 pulled-up; PA3, PA1 logic 1;
PA3, PA1 //logic 0;
k = PINA; //Reads all the data from PORTA
while (1);
}
```

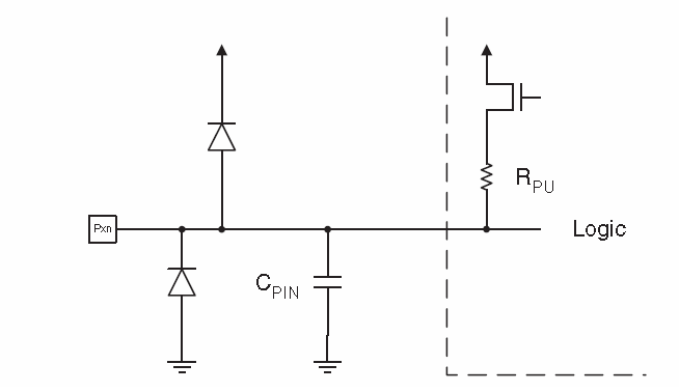


Figure 2.1: I/O pin equivalent schematic.
Source: ATMEGA16 datasheet

All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance. All I/O pins have protection diodes to both VCC and Ground as indicated in Figure 2.1.

D. To disable pull-ups of all the ports we need to set Bit 2 of SFIOR to logic one.

Special Function I/O register – SFIOR

Pin	TSM	-	-	-	ACME	PUD	PSR0	PSR321
Read/ Write	R/W	R	R	R	R/W	R/W	R/W	R/W
Initial Val	0	0	0	0	0	0	0	0

Table 2.4

Bit 2-PUD: Pull-Up Disable

When this bit is written to one, the pull-ups in all the I/O ports are disabled even if the DDRxn and PORTxn Registers are configured to enable the pull-ups ({DDRxn, PORTxn} = 0b01).

E. Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn.
Note that the SBI instruction can be used to toggle one single bit in a port. Where ‘x’ is the port name and ‘n’ is the bit number.

2.2 ATMEGA16 microcontroller pin configuration

PINNO	Pin name	USED FOR	Status
1	(XCO/T0)PB0	Logic output 1 for Left motor (Left back)	Output
2	(T1)PB1	Logic output 2 for Left motor (Left back)	Output
3	(INT2/AIN0)PB2	Logic output 1 for Right motor (Right back)	Output
4	(OC0/AIN1)PB3	Logic output 2 for Right motor (Right back)	Output
5	(SS)PB4	ISP (In System Programming)	Output
6	(MOSI)PB5		Output
7	(MISO)PB6		Input
8	(SCK)PB7		Output
9	RESET	Microcontroller reset	Default
10	VCC	5V	--
11	GND	Ground	--
12	XTAL1	Crystal 7.3728 MHz	Default
13	XTAL2		Default
14	(RXD)PD0	UART Receive*	Input
15	(TXD)PD1	UART Transmit*	Output
16	(INT0)PD2	Position Encoder input for Left Motor, TSOP1738 output**	Input
17	(INT1)PD3	Position Encoder input for Right Motor	Input
18	(OC1B)PD4	PWM output for Left Motor	Output
19	(OC1A)PD5	PWM output for Right Motor	Output
20	(ICP1)PD6	Ultrasonic Trigger Input Left (sensor no. 1) ultrasonic sensor***	Output
21	(OC2)PD7	Boot loader switch / Servo Pod output	Input / Output
22	PC0(SCL)	LCD control line RS (Register Select)	Output
23	PC1(SDA)	LCD control line RW(Read/Write Select)	Output
24	PC2(TCK)	LCD control line EN(Enable Signal)	Output
25	PC3(TMS)	Buzzer	Output
26	PC4(TDO)	LCD data lines (4-bit mode)	Output
27	PC5(TDI)		
28	PC6(TOSC1)		
29	PC7(TOSC2)		
30	AVCC	5V	--
31	AGND	Ground	--
32	AREF	ADC reference voltage pin (5V external) ****	--
33	PA7 (ADC7)	ADC input for External ultrasonic sensor	Input*****
34	PA6(ADC6)	ADC input for battery voltage monitoring	Input*****
35	PA5(ADC5)	ADC input for white line sensor Right	Input*****
36	PA4(ADC4)	ADC input for white line sensor Center	Input*****
37	PA3(ADC3)	ADC input for white line sensor Left	Input*****
38	PA2(ADC2)	ADC input for right side analog IR proximity sensor or ultrasonic range sensor	Input*****
39	PA1(ADC1)	ADC input for center side analog IR proximity sensor or	Input*****

		ultrasonic range sensor	
40	PA0(ADC0)	ADC input for left side analog IR proximity sensor or ultrasonic range sensor	Input*****

Table 2.5: ATMEGA16 microcontroller pin configuration

* UART can be connected between FT232 USB to Serial converter or XBee wireless module using jumper J5.

** Output of the Left position encoder and TSOP1738 IR receiver are open collector and both share the same 10K ohm pull-up resistor.

*** Ultrasonic sensors are connected in daisy chain for trigger synchronizing. For more details refer to chapter 3.

**** AREF can be obtained from the 5V microcontroller

***** All the ADC pins must be configured as input and floating

2.3 Application example Buzzer Beep

Located in the folder "Experiments \ 1_Buzzer_Beep" folder in the documentation CD.

This experiment demonstrates the simple operation of Buzzer ON/OFF with one second delay. Buzzer is connected to PORTC 3 pin of the ATMEGAM16

Concepts covered: Output operation, generating exact delay

Note: Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: atmega16
Frequency: 7372800
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 4.22 in the hardware manual)

```
//Buzzer is connected at the third pin of the PORTC
//To turn it on make PORTC 3rd pin logic 1

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//Function to initialize Buzzer
void buzzer_pin_config (void)
{
 DDRC = DDRC | 0x08;            //Setting PORTC 3 as output
 PORTC = PORTC & 0xF7;          //Setting PORTC 3 logic low to turnoff buzzer
```

```
}

void port_init (void)
{
 buzzer_pin_config();
}

void buzzer_on (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore | 0x08;
 PORTC = port_restore;
}

void buzzer_off (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore & 0xF7;
 PORTC = port_restore;
}

void init_devices (void)
{
 cli(); //Clears the global interrupts
 port_init();
 sei(); //Enables the global interrupts
}

//Main Function
int main(void)
{
        init_devices();
        while(1)
        {
                buzzer_on();
                _delay_ms(1000);                //delay
                buzzer_off();
                _delay_ms(1000);                //delay
        }
}
```

In this code, first three lines represent the header file declaration. The # include directive is used for including header files in the existing code. The syntax for writing header file is as follows: **#include <avr/io.h>**

This # include directive will add the already existing io.h header file stored in avr folder under winavr folder. The same way other header files are also included in the main program so that we can use various utilities defined in the header files.

In all the codes we will configure pins related to any particular module in the *xxxx_pin_config()* functions. In this example code we have used the function

buzzer_pin_config(). Buzzer is connected to the PORTC 3 pin of the microcontroller. PORTC 3 is configured as output with the initial value set to logic 0 to keep buzzer off at the time of port initialization. All the *xxxx_pin_config()* functions will be initialized in the *port_init()* function in all the codes as a convention. Function *init_devices()* will be used to initialize all the peripherals of the microcontroller as a convention.

In the above code buzzer is turned on by calling function *buzzer_on()*.
_delay_ms(1000) introduces delay of 1 second.
Buzzer is turned off by calling function *buzzer_off()*.
Again *_delay_ms(1000)* introduces delay of 1 second.
All these statements are written in *while(1)* loop construct to make buzzer on-off periodically.

2.4 Application example Simple Input – Output operation

Located in the folder "Experiments \ 2_IO_Interfacing" folder in the documentation CD.

This experiment demonstrates simple Input and Output operation. When switch is pressed buzzer gets turned on. When switch is released buzzer gets turned off. Refer to folder "Experiments \ 2_IO_Interfacing" folder in the documentation CD to look at the program.

Concepts covered: Input and Output operations

Connections:
Buzzer: PORTC 3
Input switch: PORTD 7

Note:
1. Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: atmega16
Frequency: 7372800
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

2.5 Robot direction control

Located in the folder "Experiments \ 4_Motion_Control_Simple" in the documentation CD.

Hardware aspects of the motion control are covered in detail in the chapter 3 in the hardware manual. Robot's motors are controlled by L293D motor controller from ST Microelectronics. Using L293D, microcontroller can control direction and velocity of both of the motors. To change the direction appropriate logic levels (High/Low) are

applied to IC L293D's direction pins. Velocity control is done using pulse width modulation (PWM) applied to Enable pins of L293D IC.

Microcontroller pin connections

Microcontroller Pin	Function
PD5 (OCR1AL)	Pulse width modulation for the left motor (velocity control)
PD4 (OCR1BL)	Pulse width modulation for the right motor (velocity control)
PB0	Left motor direction control
PB1	Left motor direction control
PB2	Right motor direction control
PB3	Right motor direction control

Table 2.6: Pin functions for the motion control

LED Direction Indications

Direction	L1 (Red) (I/P) PB0	L2 (Red) (I/P) PB1	PL (Yel.) (I/P) PD3	R1 (Red) (I/P) PB2	R2 (Red) (I/P) PB3	PR (Yel.) (I/P) PD4	Direction Output (Red / Yel.)	
							Left Motor	Right Motor
FORWARD	0	1	1	1	0	1	Yellow	Yellow
REVERSE	1	0	1	0	1	1	Red	Red
RIGHT *(Left wheel forward, Right wheel backward)*	0	1	1	0	1	1	Yellow	Red
LEFT *(Left wheel backward, Right wheel forward,)*	1	0	1	1	0	1	Red	Yellow
SOFT RIGHT *(Left wheel forward,, Right wheel stop)*	0	1	1	0	0	1	Yellow	Off
SOFT LEFT *(Left wheel stop, Right wheel forward,)*	0	0	1	1	0	1	Off	Yellow
SOFT RIGHT 2 *(Left wheel stop, Right wheel backward)*	0	0	1	0	1	1	Off	Red
SOFT LEFT 2 *(Left wheel backward, Right wheel stop)*	1	0	1	0	0	1	Red	Off
HARD STOP	0	0	1	0	0	1	Off	Off
SOFT STOP (Free running stop)	1	1	1	1	1	1	Off	Off

Table 2.7: Direction and PWM LED indications

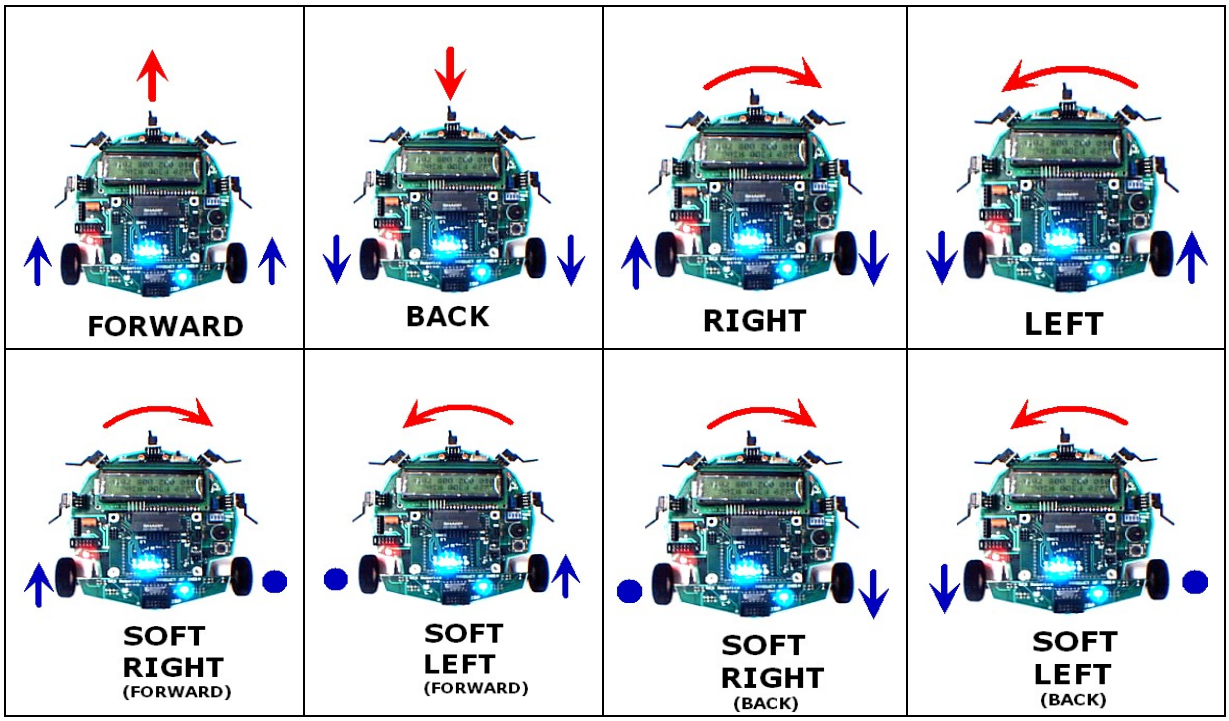


Figure 2.2: Robot direction interpretation

Note:

- All the soft turns should be used when you need more accuracy during turning
- Soft left 2 and Soft right 2 motions are very useful in grid navigation

Logic levels for setting direction and velocity

DIRECTION	LEFT BWD (LB) PB0	LEFT FWD(LF) PB1	RIGHT FWD(RF) PB2	RIGHT BWD(RB) PB3	PWM PD5 for left motor PD4 for right motor
FORWARD	0	1	1	0	As per velocity requirement
REVERSE	1	0	0	1	As per velocity requirement
RIGHT *(Left wheel forward, Right wheel backward)*	0	1	0	1	As per velocity requirement
LEFT*(Left wheel backward, Right wheel forward,)*	1	0	1	0	As per velocity requirement
SOFT RIGHT*(Left wheel forward,, Right wheel stop)*	0	1	0	0	As per velocity requirement
SOFT LEFT*(Left wheel stop, Right wheel forward,)*	0	0	1	0	As per velocity requirement
SOFT RIGHT 2 *(Left wheel stop, Right wheel backward)*	0	0	0	1	As per velocity requirement
SOFT LEFT 2 *(Left wheel backward, Right wheel stop)*	1	0	0	0	As per velocity requirement
HARD STOP	0	0	0	0	As per velocity requirement
SOFT STOP (Free running stop)	X	X	X	X	0

Table 2.8: Logic levels for setting direction and velocity

Application example: Robot direction control
Located in the folder "Experiments \4_Motion_Control_Simple" in the documentation CD.

This experiment demonstrates simple motion control.

Concepts covered: Simple motion control using I-O interfacing

There are two components to the motion control:
1. Direction control using pins PORTB0 to PORTB3
2. Velocity control by PWM on pins PD5 and PD4 using OC1A and OC1B of timer 1.

In this experiment for the simplicity PD5 and PD4 are kept at logic 1.

Connections:
Refer to table 2.6

Note:
1. Make sure that in the configuration options following settings are done for proper operation of the code
Microcontroller: atmega16
Frequency: 7372800
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 4.22 in the hardware manual)

2.6 Functions used by the robot for configuring various ports of the ATMEGA16 microcontroller

2.6.1 Buzzer

Buzzer is connected to the PORTC 3 pin of the microcontroller.
Buzzer is turned on of logic 1 is applied at the PORTC 3 pin. For more information on the hardware refer to section 3.13 in the hardware manual.

2.6.1.1 buzzer_pin_config()

PORTC 3 pin is configured as output with the initial state set at logic 0 to keep the buzzer off.

```
void buzzer_pin_config (void)
{
 DDRC = DDRC | 0x08;              //Setting PORTC 3 as output
 PORTC = PORTC & 0xF7;            //Setting PORTC 3 logic low to turnoff buzzer
}
```

2.6.1.2 buzzer_on()

Turns on the buzzer by setting PORTC 3 pin to logic 1.

```
void buzzer_on (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore | 0x08;
 PORTC = port_restore;
}
```

2.6.1.3 buzzer_off()

Turns off the buzzer by setting PORTC 3 pin to logic 0.

```
void buzzer_off (void)
{
 unsigned char port_restore = 0;
 port_restore = PINC;
 port_restore = port_restore & 0xF7;
 PORTC = port_restore;
}
```

2.6.4 Robot motion control

Refer to table 2.6 for the hardware connection details, table 2.8 for control logic and from the hardware manual section 3.6 for more information on the hardware.

2.6.4.1 motion_pin_config()

Sets the directions and logic levels of the pins involved in the motion control.

```
void motion_pin_config (void)
{
 DDRB = DDRB | 0x0F;  //set direction of the PORTB 3 to PORTB 0 pins as output
 PORTB = PORTB & 0xF0; // set initial value of the PORTB 3 to PORTB 0 pins to logic 0
 DDRD = DDRD | 0x30;  //Setting PD5 and PD4 pins as output for PWM generation
 PORTD = PORTD | 0x30; //PD5 and PD4 pins are for velocity control using PWM
}
```

2.6.4.2 motion_set()

Used for setting appropriate logic values for controlling robots direction. It is called by other functions to set robot's direction.

```
void motion_set (unsigned char Direction)
{
 unsigned char PortBRestore = 0;

 Direction &= 0x0F;              // removing upper nibble as it is not needed
 PortBRestore = PORTB;           // reading the PORTB's original status
 PortBRestore &= 0xF0;           // setting lower direction nibble to 0
 PortBRestore |= Direction;      // adding lower nibble for direction command and
                                 // restoring the PORTB status
 PORTB = PortBRestore;           // setting the command to the port
}
```

2.6.4.3 Robot direction set functions

Sets robot's direction

```
void forward (void) //both wheels forward
{
 motion_set(0x06);
}

void back (void) //both wheels backward
{
 motion_set(0x09);
}

void left (void) //Left wheel backward, Right wheel forward
{
 motion_set(0x05);
}

void right (void) //Left wheel forward, Right wheel backward
{
 motion_set(0x0A);
}

void soft_left (void) //Left wheel stationary, Right wheel forward
{
motion_set(0x04);
}

void soft_right (void) //Left wheel forward, Right wheel is stationary
{
motion_set(0x02);
}

void soft_left_2 (void) //Left wheel backward, right wheel stationary
{
motion_set(0x01);
}

void soft_right_2 (void) //Left wheel stationary, Right wheel backward
{
motion_set(0x08);
}

void hard_stop (void)      //hard stop(stop suddenly)
{
motion_set(0x00);
}

void soft_stop (void)      //soft stop(stop slowly)
{
motion_set(0x0F);
}
```

2.6.4 Functions for configuring Position encoder pins

2.6.4.1 left_encoder_pin_config()

```
//Function to configure INT1 (PORTD 3) pin as input for the left position encoder
void left_encoder_pin_config (void)
{
 DDRD  = DDRD & 0xF7;  //Set the direction of the PORTD 3 pin as input
 PORTD = PORTD | 0x08;  //Enable internal pull-up for PORTD 3 pin
}
```

2.6.4.2 right_encoder_pin_config()

```
//Function to configure INT0 (PORTD 2) pin as input for the right position encoder
void right_encoder_pin_config (void)
{
 DDRD  = DDRD & 0xFB;  //Set the direction of the PORTD 2 pin as input
 PORTD = PORTD | 0x04;  //Enable internal pull-up for PORTD 2 pin
}
```

Note: To get data from the position encoders interrupts are used which are covered in the chapter 3.

2.6.5 lcd_port_config()

```
void lcd_port_config (void)
{
 DDRC = DDRC | 0xF7;      //all the LCD pin's direction set as output
 PORTC = PORTC & 0x80; // all the LCD pins are set to logic 0 except PORTC 7
}
```

2.6.6 adc_pin_config()

```
void adc_pin_config (void)
{
 DDRA = 0x00;   //set PORTA direction as input
 PORTA = 0x00; //sett PORTA pins floating
}
```

3. Robot Position Control Using Interrupts

SPARK V incorporates various interrupt handling mechanisms such as timer overflow interrupts, timer compare interrupts, serial interrupts and external interrupts for doing specific tasks. In this chapter, we will have a brief overview of interrupt concept and will implement external hardware interrupts for position estimation of robots using position encoders.

Interrupts *interrupt* the flow of the program and cause it to branch to ISR (Interrupt Service Routine). ISR does the task that needs to be done when interrupt occurs. Whenever position encoder moves by one tick it interrupts the microcontroller and ISR does the job of tracking position count.

Each interrupt has a vector address assigned to it low in program memory. The compiler places the starting address of the associated interrupt service routine and a relative jump instruction at the vector location for each interrupt. When the interrupt occurs, the program completes executing its current instruction and branches to the vector location associated with that interrupt. The program then executes the relative jump instruction to the interrupt service routine (ISR) and begins executing the ISR. For more information on the interrupt vectors refer to table 18 in the ATMEGA16 datasheet which is located in the "datasheet" folder in the documentation CD.

When an interrupt occurs, the return address is stored on the system stack. The RETI assembly language instruction causes the return address to be popped off the stack and continue program execution from the point where it was interrupted.

3.1 Using Interrupts

Interrupts needs to be initialized before they become active. Initializing interrupt is a three step process. The first step is to select the trigger type for the interrupt. We are using falling edge trigger. This is selected by setting bits in MCUCR and MCUCSR registers. Second step is to enable the interrupt that we want to use in the GICR register. In the third step we globally enable all the unmasked interrupts. To enable unmasked interrupts we need to set global interrupt enable bit in the status register (SREG). This is done by instruction "sei ();".

3.1.1 Registers involved

3.1.1.1 MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0
	-	**0**	-	-	**ISC11**	**ISC10**	**ISC01**	**ISC00**
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 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0
The External Interrupt 1 is activated by the external pin INT1 if the SREG I-bit and the corresponding interrupt mask in the GICR are set. The level and edges on the external

INT1 pin that activate the interrupt are defined in Table 3.1. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

ISC11	ISC100	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any edge of INT1 generates asynchronously an interrupt request.
1	0	The falling edge of INT1 generates asynchronously an interrupt request.
1	1	The rising edge of INT1 generates asynchronously an interrupt request.

Table 3.1: Interrupt 1 Sense Control

Bit 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0
The External Interrupt 0 is activated by the external pin INT0 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT0 pin that activate the interrupt are defined in Table 3.2. The value on the INT0 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any edge of INT0 generates asynchronously an interrupt request.
1	0	The falling edge of INT0 generates asynchronously an interrupt request.
1	1	The rising edge of INT0 generates asynchronously an interrupt request.

Table 3.2: Interrupt 0 Sense Control

3.1.1.2 MCUCSR – MCU Control and Status Register

Bit	7	6	5	4	3	2	1	0
	-	**ISC2**	-	-	-	-	-	-
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 6 – ISC2: Interrupt Sense Control 2

The Asynchronous External Interrupt 2 is activated by the external pin INT2 if the SREG I-bit and the corresponding interrupt mask in GICR are set. If ISC2 is written to zero, a falling edge on INT2 activates the interrupt. If ISC2 is written to one, a rising edge on INT2 activates the interrupt. Edges on INT2 are registered asynchronously. Pulses on INT2 wider than the minimum pulse width given in Table 3.3 will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. When changing the ISC2 bit, an interrupt can occur. Therefore, it is recommended to first disable INT2 by clearing its Interrupt Enable bit in the GICR Register. Then, the ISC2 bit can be changed. Finally, the INT2 Interrupt Flag should be cleared by writing a logical one to its Interrupt Flag bit (INTF2) in the GIFR Register before the interrupt is re-enabled.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
t_{INT}	Minimum Pulse width for asynchronous external interrupt	-	-	50	-	ns

Table 3.3: Asynchronous External Interrupt Characteristics

3.1.1.3 GICR – General Interrupt Control Register

Bit	7	6	5	4	3	2	1	0
	INT1	**INT0**	**INT2**	**-**	**-**	**-**	**-**	**-**
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 – INT1: External Interrupt Request 1 Enable
When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the MCU General Control Register (MCUCR) define whether the External Interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from the INT1interrupt Vector.

Bit 6 – INT0: External Interrupt Request 0 Enable
When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the MCU General Control Register (MCUCR) define whether the External Interrupt is activated on rising and/or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from the INT0 interrupt vector.

Bit 5 – INT2: External Interrupt Request 2 Enable
When the INT2 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control2 bit (ISC2) in the MCU

Control and Status Register (MCUCSR) defines whether the External Interrupt is activated on rising or falling edge of the INT2 pin. Activity on the pin will cause an interrupt request even if INT2 is configured as an output. The corresponding interrupt of External Interrupt Request 2 is executed from the INT2 Interrupt Vector.

3.1.1.4 GIFR – General Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
	INTF1	**INTF0**	**INTF2**	**-**	**-**	**-**	**-**	**-**
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 – INTF1: External Interrupt Flag 1
When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 becomes set (one). If the I-bit in SREG and the INT1 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT1 is configured as a level interrupt.

Bit 6 – INTF0: External Interrupt Flag 0
When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 becomes set (one). If the I-bit in SREG and the INT0 bit in GICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. This flag is always cleared when INT0 is configured as a level interrupt.

Bit 5 – INTF2: External Interrupt Flag 2
When an event on the INT2 pin triggers an interrupt request, INTF2 becomes set (one). If the I-bit in SREG and the INT2 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared by writing a logical one to it. Note that when entering some sleep modes with the INT2 interrupt disabled, the input buffer on this pin will be disabled. This may cause a logic change in internal signals which will set the INTF2 flag. See “Digital Input Enable and Sleep Modes” on page 51 for more information.

3.1.2 Functions for configuring interrupt pins (called inside the “port_init()” function)

```
//Function to configure INT1 (PORTD 3) pin as input for the left position encoder
void left_encoder_pin_config (void)
{
 DDRD  = DDRD & 0xF7; //Set the direction of the PORTD 3 pin as input
 PORTD = PORTD | 0x08; //Enable internal pull-up for PORTD 3 pin
}
```

```
//Function to configure INT0 (PORTD 2) pin as input for the right position encoder
void right_encoder_pin_config (void)
{
 DDRD = DDRD & 0xFB; //Set the direction of the PORTD 2 pin as input
 PORTD = PORTD | 0x04; //Enable internal pull-up for PORTD 2 pin
}
```

3.1.3 Functions for configuring external interrupts for position encoders

```
void left_position_encoder_interrupt_init (void) //Interrupt 1 enable
{
 cli(); //Clears the global interrupt
 MCUCR = MCUCR | 0x08; // INT1 is set to trigger with falling edge
 GICR = GICR | 0x80;  // Enable Interrupt INT1 for left position encoder
 sei();   // Enables the global interrupt
}

void right_position_encoder_interrupt_init (void) //Interrupt 0 enable
{
 cli(); //Clears the global interrupt
 MCUCR = MCUCR | 0x02 // INT0 is set to trigger with falling edge
 GICR = GICR | 0x40; // Enable Interrupt INT0 for right position encoder
 sei();   // Enables the global interrupt
}
```

3.1.4 Function for initialization of interrupts

```
//Function to initialize all the devices
void init_devices()
{
 cli(); //Clears the global interrupt
 left_position_encoder_interrupt_init();
 right_position_encoder_interrupt_init();
 sei();   // Enables the global interrupt
}
```

3.1.5 Interrupt Service Routine (ISR)

After initializing interrupts, the next step is to define the Interrupt Service Routine (ISR). ISR in AVR Studio can be written in two different ways.

a. ISR (INT0_vect)
b. SIGNAL(SIG_INTERRUPT0)

Both of these formats are valid syntactically but we will be using *ISR (INT0_vect)*

Various syntaxes for ISR are described in datasheet of Atmega16 microcontroller and also in < iom16.h> files in WinAVR-20090313\avr\include\avr folder.

```
//ISR for right position encoder
ISR(INT0_vect)
{
 //Your code
}
```

```
//SR for left position encoder
ISR(INT1_vect)
{
//Your code
}
```

3.2 Robot position control using interrupts

Position encoders give position / velocity feedback to the robot. It is used in closed loop to control robot's position and velocity. Position encoder consists of slotted disc which rotates between optical encoder (optical transmitter and receiver). When slotted disc moves in between the optical encoder we get square wave signal whose pulse count indicates position and time period / frequency indicates velocity. For more details on the hardware refer to section 3.8 from the hardware manual.

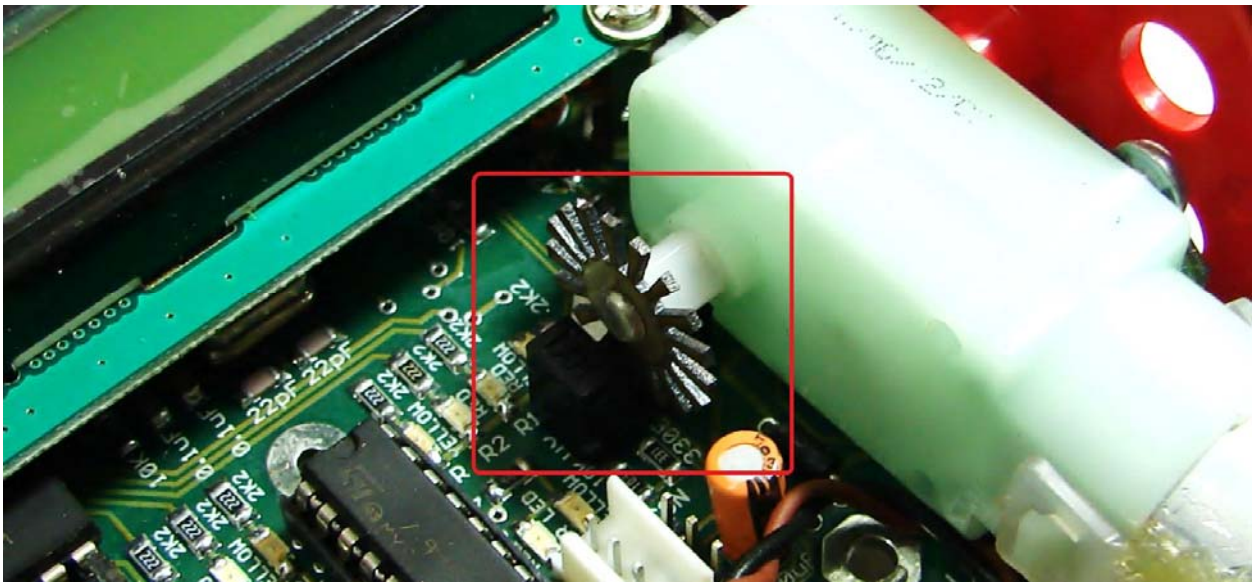


Figure 4.1: Position encoder assembly in the Spark V robot

3.2.1 Calculation of position encoder resolution:

Case 1: Robot is moving forward or backward (encoder resolution is in mm)

Wheel diameter: 7cm
Wheel circumference: 7cm * 3.14 = 21.980cm = 219.80mm
Number slots on the encoder disc: 17
Position encoder resolution: 219.80 mm / 17 = 12.92mm / pulse.

Case 2: Robot is turning with one wheel rotating clockwise while other wheel is rotating anti clockwise. Center of rotation is in the center of line passing through wheel axel and both wheels are rotating in opposite direction (encoder resolution is in degrees)

Distance between Wheels = 11.6cm

Radius of Circle formed in 360^0 rotation of Robot = Distance between Wheels / 2
= 5.8 cm

Distance Covered by Robot in 360^0 Rotation = Circumference of Circle traced
= 2 x 5.8 x 3.14
= 36.4 cm or 364mm

Number of wheel rotations in 360^0 rotation of robot
= Circumference of Traced Circle / Circumference of Wheel
= 364 / 219.80
= 1.65

Total pulses in 360^0 Rotation of Robot
= Number of slots on the encoder disc / Number of wheel rotations of in 360^0 rotation of robot
= 17 x 1.65
= 28.05 (approximately 28)

Position Encoder Resolution in Degrees = 360 / 28
= 12.85 degrees per count

Case 3: Robot is turning with one wheel stationary while other wheel is rotating clockwise or anti clockwise. Center of rotation is center of the stationary wheel (encoder resolution is in degrees)

In this case only one wheel is rotating and other wheel is stationary so robot will complete its 360^0 rotation with stationary wheel as its center.

Radius of Circle formed in 360^0 rotation of Robot = Distance between Wheels
= 11.6 cm

Distance Covered by Robot in 360^0 Rotation = Circumference of Circle traced
= 2 x 11.6 x 3.14
= 72.848 cm or 728 mm

Number of wheel rotations of in 360^0 rotation of robot
= Circumference of Traced Circle / Circumference of Wheel
= 728 / 219.80
= 3.312

Total pulses in 360^0 Rotation of Robot
= Number of slots on the encoder disc / Number of wheel rotations of in 360^0 rotation of robot
= 17 x 3.312
= 56.304 (approximately 56)

Position Encoder Resolution in Degrees = 360 /56
= 6.42 degrees per count

3.2.2 Interrupt service routine for position encoder

3.2.2.1 ISR for right position encoder

```
//ISR for right position encoder
ISR(INT0_vect)
{
 ShaftCountRight++; //increment right shaft position count
}
```

3.2.2.2 ISR for left position encoder

```
//ISR for left position encoder
ISR(INT1_vect)
{
 ShaftCountLeft++; //increment left shaft position count
}
```

3.2.3 Functions for robot position control

3.2.3.1 Function for rotating robot by specific degrees

```
//Function used for turning robot by specified degrees
void angle_rotate(unsigned int Degrees)
{
 float ReqdShaftCount = 0;
 unsigned long int ReqdShaftCountInt = 0;

 ReqdShaftCount = (float) Degrees/ 12.85; // division by resolution to get shaft count
 ReqdShaftCountInt = (unsigned int) ReqdShaftCount;
 ShaftCountRight = 0;
 ShaftCountLeft = 0;

 while (1)
 {
  if((ShaftCountRight >= ReqdShaftCountInt) | (ShaftCountLeft >= ReqdShaftCountInt))
  break;
 }
 stop(); //Stop robot
}
```

3.2.3.2 Function for moving robot forward and back by specific distance

```
//Function used for moving robot forward by specified distance
void linear_distance_mm(unsigned int DistanceInMM)
{
 float ReqdShaftCount = 0;
 unsigned long int ReqdShaftCountInt = 0;

 ReqdShaftCount = DistanceInMM / 12.92; // division by resolution to get shaft count
 ReqdShaftCountInt = (unsigned long int) ReqdShaftCount;

 ShaftCountRight = 0;
 while(1)
 {
  if(ShaftCountRight > ReqdShaftCountInt)
  {
      break;
```

```
  }
 }
 stop(); //Stop robot
}
```

3.2.3.3 Forward in mm

```
void forward_mm(unsigned int DistanceInMM)
{
 forward();
 linear_distance_mm(DistanceInMM);
}
```

3.2.3.4 Backward in mm

```
void back_mm(unsigned int DistanceInMM)
{
 back();
 linear_distance_mm(DistanceInMM);
}
```

3.2.3.5 left in degrees

```
void left_degrees(unsigned int Degrees)
{
// 28 pulses for 360 degrees rotation 12.92 degrees per count
 left(); //Turn left
 angle_rotate(Degrees);
}
```

3.2.3.6 right in degrees

```
void right_degrees(unsigned int Degrees)
{
// 28 pulses for 360 degrees rotation 12.92 degrees per count
 right(); //Turn right
 angle_rotate(Degrees);
}
```

3.2.3.7 soft left in degrees

```
void soft_left_degrees(unsigned int Degrees)
{
 // 56 pulses for 360 degrees rotation 6.42 degrees per count
 soft_left(); //Turn soft left
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

3.2.3.8 soft right in degrees

```
void soft_right_degrees(unsigned int Degrees)
{
 // 56 pulses for 360 degrees rotation 6.42 degrees per count
 soft_right();  //Turn soft right
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

3.2.3.9 soft left 2 in degrees

```
void soft_left_2_degrees(unsigned int Degrees)
{
 // 56 pulses for 360 degrees rotation 6.42 degrees per count
 soft_left_2(); //Turn reverse soft left
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

3.2.3.10 soft right 2 in degrees

```
void soft_right_2_degrees(unsigned int Degrees)
{
 // 56 pulses for 360 degrees rotation 6.42 degrees per count
 soft_right_2();  //Turn reverse soft right
 Degrees=Degrees*2;
 angle_rotate(Degrees);
}
```

3.2.4 Application example of robot position control

Located in the folder "Experiments \ Position_Control_Interrups" folder in the documentation CD.

This experiment demonstrates use of position encoders.

Concepts covered: External Interrupts, Position control

Connections:
PORTB3 to PORTB0: Robot direction control
PD2, PD3: Robot velocity control. Currently set to 1 as PWM is not used
PD3 (INT1): External interrupt for left motor position encoder
PD2 (INT0): External interrupt for the right position encoder

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

    Microcontroller: atmega16
    Frequency: 7372800Hz
    Optimization: -O0
    (For more information read section: Selecting proper optimization options below figure 2.22 in the software manual)

4. Timer / Counter Operations on the Robot

ATMEGA16 has 2 eight bit timers (timer 0 and timer 2) and 1 sixteen bit timer (timer1). All the timers have independent Output Compare Units with PWM support. These timers can be used for accurate program execution timing (event management) and output signal generation.

SPARK V uses these timers mainly for the following applications:

- Velocity control – Timer 1 is used to generate PWM for robot's velocity control.
- Servo motor control.
- Event scheduling – Timer1 with timer overflow interrupt is used for event scheduling.

In the SPARK V ATMEGA16 robot Timer 1 is used to generate PWM for robot velocity control. All other timers are free and can be used for other purposes.

Note: Theory content of this chapter is based on the ATMEGA16 datasheet which is located in the "datasheet" folder in the documentation CD.

General features of the 8 bit timers 0 and 2

- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0, TOV2, OCF2)

General features of the 16 bit timers 1

- True 16-bit Design (i.e., Allows 16-bit PWM)
- Three independent Output Compare Units
- Double Buffered Output Compare Registers
- One Input Capture Unit
- Input Capture Noise Canceller
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- External Event Counter
- independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1)

4.1 Important terms involved in the timers:

BOTTOM: The counter reaches the BOTTOM when it becomes 0x0000.

MAX: The counter reaches its MAX value when it becomes 0xFF (decimal 255) for 8 bit timer or 0xFFFF (decimal 65535) for 16 bit timer.

TOP: The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCRnA or ICRn Register. The assignment is dependent of the mode of operation. where n is 0,1,2 for the timer used.

Accessing 16-bit Registers
The TCNT1, OCRA/B, and ICR1 are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same Temporary Register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

Not all 16-bit accesses use the Temporary Register for the high byte. Reading the OCR1A/B/C 16-bit registers does not involve using the Temporary Register. To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte. The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCR1A/B/C and ICR1 Registers.

Modes of operation in timers:

1. Normal mode
2. Clear timer on compare match (CTC) mode
3. Fast PWM mode
4. Phase correct PWM mode
5. Phase and frequency correct PWM mode

For more information on the timer operation refer to ATMEGA16 datasheet.

4.2 16 bit Timer Registers

Clock source for the Timers

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select CS12:0 bits located in the Timer/Counter control Register B (TCCR1B).

The Timer/Counter can be clocked directly by the system clock (by setting the CS12:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency (fCLK_I/O). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either fCLK_I/O/8, fCLK_I/O/64, fCLK_I/O/256, or fCLK_I/O/1024.

4.2.1 TCCR1A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0
	COM1A1	**COM1A0**	**COM1B1**	**COM1B0**	**FOC1A**	**FOC1B**	**WGM11**	**WGM10**
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 7:6 – COM1A1:0: Compare Output Mode for Channel A
Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B

The COM1A1:0 and COM1B1:0 control the Output Compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the *Data Direction Register* (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver. When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 4.1 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

COMnA1 COMnB1	COMnA0 COMnB0	Description
0	0	Normal port operation, OC1A, OC1B disconnected
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OCnB disconnected (normal port operation).
1	0	Clear OCnA/OCnB on compare match, set OCnA/OCnB at BOTTOM (non-inverting mode).
1	1	Set OCnA/OCnB on compare match, clear OCnA/OCnB at BOTTOM (inverting mode).

Table 4.1: COMnX1:0 bit functionality when the WGM13:0 bits are set to fast PWM mode.

Bit 1:0 – WGM11:0: Waveform Generation Mode
Combined with the WGM13:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 5.3. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes.

For more information on the different modes, refer “Modes of Operation” on page 94 of the ATMEGA16 datasheet.

4.2.2 TCCR1B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0
	ICNC1	**ICES1**	**-**	**WGM13**	**WGM12**	**CS12**	**CS11**	**CS10**
Read / Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – ICNC1: Input Capture Noise Canceller
Setting this bit (to one) activates the Input Capture Noise Canceller. When the Noise canceller is activated, the input from the Input Capture Pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The input capture is therefore delayed by four Oscillator cycles when the noise canceller is enabled.

Bit 6 – ICES1: Input Capture Edge Select
This bit selects which edge on the Input Capture Pin (ICP1) that is used to trigger a capture event. When the ICES5 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture. When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled. When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected and consequently the input capture function is disabled.

Bit 5 – Reserved Bit
This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

Bit 4:3 – WGM13:2: Waveform Generation Mode
See TCCR1A Register description and refer to table 4.3

Bit 2:0 – CS12:0: Clock Select
The three clock select bits select the clock source to be used by the Timer/Counter.

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge
1	1	1	External clock source on T1 pin. Clock on rising edge

Table 4.2: Clock select bit description

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Table 4.3: Waveform generation mode bit description

4.2.4 TIMSK – Timer/Counter n Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
	-	-	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	-	-
Read / Write			R/W	R/W	R/W	R/W		
Initial Value			0	0	0	0		

Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture Interrupt is enabled. The corresponding Interrupt vector (See “Interrupts” on page 45 in the ATMEGA2560 datasheet) is executed when the ICF1 flag, located in TIFR, is set.

Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 45 in the ATMEGA2560 datasheet) is executed when the OCF1A flag, located in TIFR, is set.

Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 45 in the ATMEGA2560 datasheet) is executed when the OCF1B flag, located in TIFR, is set.

Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow Interrupt is enabled. The corresponding

Interrupt Vector (See "Interrupts" on page 45 in the ATMEGA2560 datasheet) is executed when the TOV1 flag, located in TIFR, is set.

4.2.5 TIFR – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
	-	-	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	-	-
Read / Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 5 – ICF1: Timer/Counter1, Input Capture Flag
This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 flag is set when the counter reaches the TOP value. ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag
This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A). Note that a Forced Output Compare (FOC1A) strobe will not set the OCF1A flag. OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag
This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B). Note that a forced output compare (FOC1B) strobe will not set the OCF1B flag. OCF1B is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

Bit 2 – TOV1: Timer/Counter1, Overflow Flag
The setting of this flag is dependent of the WGM13:0 bits setting. In normal and CTC modes, the TOV1 flag is set when the timer overflows. Refer to Table 5.3 on page 78 for the TOV1 flag behavior when using another WGM13:0 bit setting. TOV1 is automatically cleared when the Timer/Counter1 Overflow interrupt vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

4.3 Velocity control using PWM

4.3.1 Concept of PWM

Pulse width modulation is a process in which duty cycle of constant frequency square wave is modulated to control power delivered to the load i.e. motor.

Duty cycle is the ratio of **'T-ON/ T'**. Where **'T-ON'** is ON time and **'T'** is the time period of the wave. Power delivered to the motor is proportional to the **'T-ON'** time of the signal. In case of PWM the motor reacts to the time average of the signal.

PWM is used to control total amount of power delivered to the load without power losses which generally occur in resistive methods of power control.

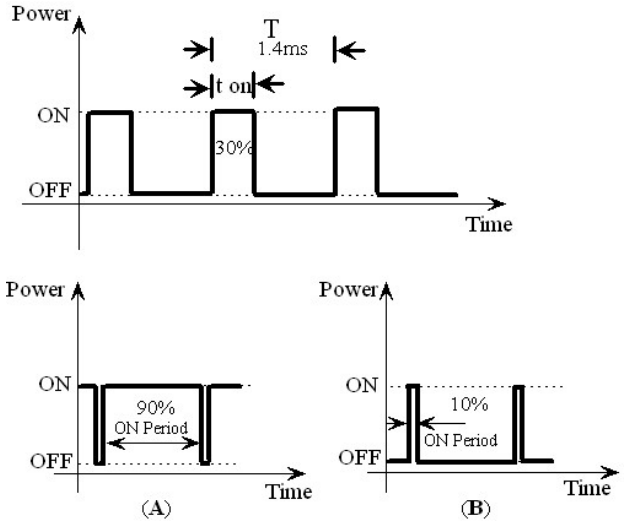


Figure 4.1: Pulse Width Modulation (PWM)

Above figure shows the PWM waveforms for motor velocity control. In case (A), ON time is 90% of time period. This wave has more average value. Hence more power is delivered to the motor. In case (B), the motor will run slower as the ON time is just 10% of time period.

Microcontroller Pin	Function
PD5 (OCR1AL)	Pulse width modulation for the left motor (velocity control)
PD4 (OCR1BL)	Pulse width modulation for the right motor (velocity control)
PB0	Left motor direction control
PB1	Left motor direction control
PB2	Right motor direction control
PB3	Right motor direction control

Table 4.4: Pin functions for the motion control

4.3.2 PWM generation using Timer

PWM using Timer 1
All 16 bit timers are identical in nature. We are using timer 1 for PWM as input pins of the motor driver IC L293D are connected to PD5 (OC1A) and PD4 (OC1B).

For robot velocity control Timer 1 is used in 8 bit fast PWM generation mode. In the non inverting compare output mode.

The counter counts from BOTTOM to MAX and again restarts from BOTTOM. In non-inverting compare output mode, the output compare (OC1X) is cleared on the compare match between TCNT1 and OCR1X, and set at BOTTOM. Where X is A or B. In inverting compare output mode output (OC1X) is set on compare match and cleared at BOTTOM.

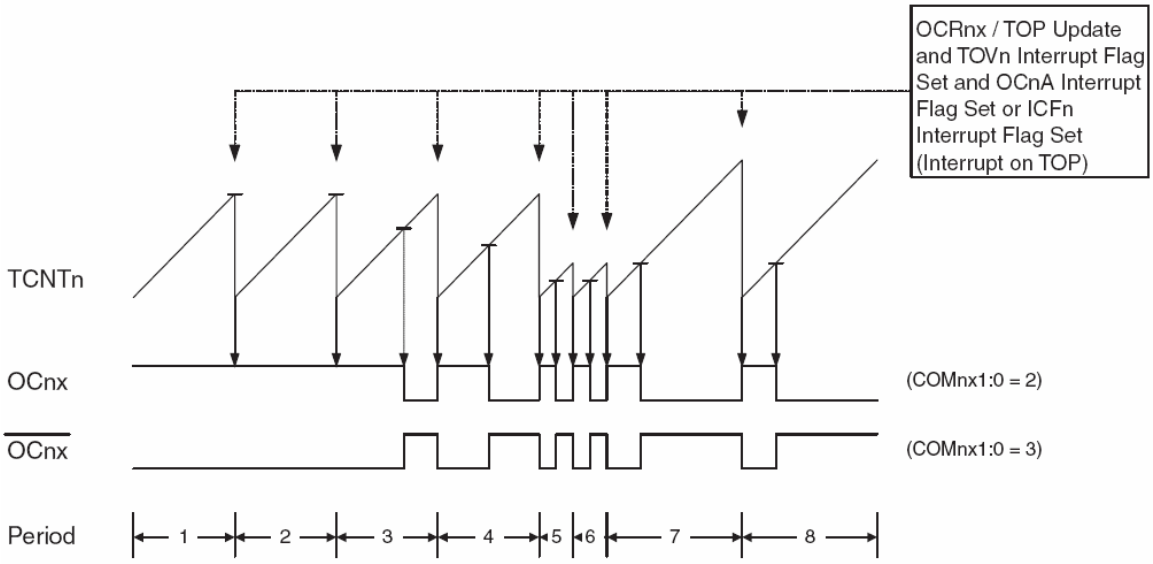


Figure 4.2: Time diagram for fast PWM mode

In 8 bit fast PWM mode the counter is incremented until the counter value matches either fixed value of 0x00FF hex and then value is rolled over again to 0. In the non-inverting PWM mode output on the output compare pins (OC1A and OC1B in this case) is logic 0 when counter starts at 0. When counter value is matched with OCR1x (in this case OCR1AL and OCR1BL output) corresponding compare pins (OC1A and OC1B in this case) becomes logic 1. It stays at logic 1 till counter rolls over from 0xFF to 0. At the roll over value of these OCR1x pins is set to logic 0. To change the duty cycle of the PWM form 0 to 100% duty cycle in the 8 bit fast PWM generation mode value of OCR1x can be set between 0 to 255 (0x00 to 0xFF).

4.3.3 Timer 1 configuration in 8 bit fast PWM mode

Function Timer1_init() function initializes the function in 8 bit fast PWM generation mode.

```
// Timer 5 initialized in PWM mode for velocity control
// Prescale: 64
// PWM 8bit fast, TOP=0x00FF
// Timer Frequency:450.000Hz
void timer1_init()
{
        TCCR1B = 0x00;          //Stop
        TCNT1H = 0xFF;          //Counter higher 8-bit value to which OCR1xH value is compared with
        TCNT1L = 0x01;          //Counter lower 8-bit value to which OCR1xH value is compared with
        OCR1AH = 0x00;          //Output compare register high value for Left Motor
        OCR1AL = 0xFF;          //Output compare register low value for Left Motor
        OCR1BH = 0x00;          //Output compare register high value for Right Motor
        OCR1BL = 0xFF;          //Output compare register low value for Right Motor
        TCCR1A = 0xA1;          //COM1A1=1, COM1A0=0; COM1B1=1, COM1B0=0;
//For Overriding normal port functionality to OCR1A outputs. WGM11=0, WGM10=1 Along With GM12
//in TCCR1B for Selecting FAST PWM 8-bit Mode
        TCCR1B = 0x0D;          //WGM12=1; CS12=0, CS11=1, CS10=1 (Prescaler=64)
}
```

PWM frequency calculation:

PWM frequency = System Clock / N (1 + TOP)
= 7.3728MHz / 64 (1 + 255)
= 450.000 Hz

Where
System clock = Crystal frequency = 7.3728MHz
Prescale = N = 64
TOP = 255 (8 bit resolution)

System Clock / Prescale	8-bit (TOP = 255)
System Clock	F_{pwm} = 28.799 KHz
System Clock / 8	F_{pwm} = 3.600 KHz
System Clock / 64	F_{pwm} = 450.000Hz
System Clock / 256	F_{pwm} = 112.500 Hz
System Clock / 1024	F_{pwm} = 28.125 Hz

Table 4.2: 8 bit PWM fast frequency for different prescale options

4.3.4 Function for timer 1 initialization

```
void init_devices (void) //use this function to initialize all devices
{
 cli(); //disable all interrupts
 timer1_init();
 sei(); //re-enable interrupts
}
```

cli(); disables all the interrupts and sei(); enables all the interrupts.

It is very important that all the devices should be configured after disabling all the interrupts. All the peripherals of the microcontroller will be configured inside init_devices() function.

4.3.5 Functions for PWM output pin configuration and robot's velocity control

4.3.5.1 Functions for PWM output pin configuration (called inside the "port_init()" function)

```
void motion_pin_config (void)
{
DDRB = DDRB | 0x0F; //set direction of the PORTB 3 to PORTB 0 pins as output
PORTB = PORTB & 0xF0; // set initial value of the PORTB 3 to PORTB 0 pins to logic 0
DDRD = DDRD | 0x30;   //Setting PD5 and PD4 pins as output for PWM generation
PORTD = PORTD | 0x30; //PD5 and PD4 pins are for velocity control using PWM
}
```

4.3.5.2 Function for robot's velocity control

```
void velocity (unsigned char left_motor, unsigned char right_motor)
{
OCR1AL = left_motor;
OCR1BL = right_motor;
}
```

This function takes velocity for left motor and right motor as input parameter and assigns them to output compare register OCR1A and OCR1B. Channel A is used for left motor and channel B is used for right motor. Since we are using PWM in 8 bit resolution we only load lower byte of the OCR1A and OCR1B registers.

4.3.6 Application example for robot velocity control

Located in the folder "Experiments \ 5_Velocity_Control_using_PWM" folder in the documentation CD.

This experiment demonstrates robot velocity control using PWM.

Concepts covered: Use of timer to generate PWM for velocity control

There are two components to the motion control:
1. Direction control using pins PORTB0 to PORTB3
2. Velocity control by PWM on pins PD5 and PD4 using OC1A and OC1B of timer 1.

Connections: Refer to table 4.4 for connection details.

Note:
1. Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: atmega16
Frequency: 7372800
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 4.22 in the software manual)

5. LCD Interfacing

To interface LCD with the microcontroller requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To save number of I/Os required for LCD interfacing we can use 3 control signals with 4 data lines. This is known as 4 bit interfacing mode and it requires 7 I/O lines. We are using 4 bit interfacing mode to reduce number of I/O lines. In this mode higher nibble and lower nibble of commands/data set needs to be sent separately. Figure 5.1 shows LCD interfacing in 4 bit mode. The three control lines are referred to as EN, RS, and RW.

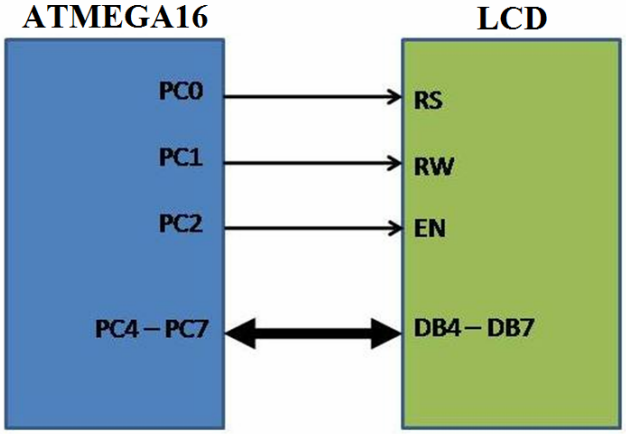


Figure 5.1: LCD interfacing in 4 bit mode

Microcontroller	LCD PINS	Description
VCC	VCC	Supply voltage (5V).
GND	GND	Ground
PC0	RS (Control line)	Register Select
PC1	R/W (Control line)	READ /WRITE
PC2	EN (Control Line)	Enable
PC4 to PC7	D4 to D7 (Data lines)	Bidirectional Data Bus
	LED+, LED-	Backlight control

Table 5.1: LCD pin mapping with the microcontroller

The EN line is called "Enable" and it is connected to PC2. This control line is used to tell the LCD that microcontroller has sent data to it or microcontroller is ready to receive data from LCD. This is indicated by a high-to-low transition on this line. To send data to the LCD, program should make sure that this line is low (0) and then set the other two control lines as required and put data on the data bus. When this is done, make EN high (1) and wait for the minimum amount of time as specified by the LCD datasheet, and end by bringing it to low (0) again.

The RS line is the "Register Select" line and it is connected to PC0. When RS is low (0), the data is treated as a command or special instruction by the LCD (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is treated as text data which should be displayed on the screen.

The RW line is the "Read/Write" control line and it is connected to PC1. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading from) the LCD.

The data bus is bidirectional, 4 bit wide and is connected to PC4 to PC7 of the microcontroller. The MSB bit (DB7) of data bus is also used as a Busy flag. When the Busy flag is 1, the LCD is in internal operation mode, and the next instruction will not be accepted. When RS = 0 and R/W = 1, the Busy flag is output on DB7. The next instruction must be written after ensuring that the busy flag is 0.

We are using LCD in 4-bit mode, in 4-bit mode the data is sent in nibbles with higher nibble sent first followed by the lower nibble. Initialization of LCD in 4-bit mode is done only after setting the LCD for 4-bit mode. LCD reset sequence include following steps.

1. Wait for about 20ms.
2. Send the first value 0x30.
3. Wait for about 10ms.
4. Send the second value 0x30.
5. Wait for about 1ms.
6. Send the third value 0x30.
7. Wait for about 1ms.
8. Send 0x20 for selecting 4-bit mode.
9. Wait for 1ms.

Before we can display any data on the LCD we need to initialize the LCD for proper operation. The first instruction we send must tell the LCD that we will be communicating with it using 4-bit data bus. Remember that the RS line must be low if we are sending a command to the LCD. In the second and third instruction we clear and reset the display of the LCD. The fourth instruction sets the display and cursor ON. In fifth instruction we place the cursor at the start. Check the lcd_init( ) function to see how all this is put in code.

The function lcd_reset() and lcd_init completes the initialization of LCD in 4-bit mode. Now following steps are followed to send the command/data in 4-bit mode.

1. Mask lower 4-bits.
2. Send command/data to the LCD port.
3. Send enable signal to EN pin.
4. Mask higher 4-bits.
5. Shift bits left by 4 positions (to bring lower bits to upper bits position).
6. Send command/data to the LCD port.
7. Send enable signal to EN pin.

The function lcd_wr_command() and lcd_wr_char() are for sending the command and data respectively to the LCD.

For using the busy flag (polling method) the LCD is read in the similar way, i.e. nibble by nibble, here we are not using the polling method and instead we are providing the necessary delay between the commands.

After the initialization of LCD in 4-bit mode is complete, then for sending the data in nibbles there is no need of providing any delay between two nibbles of same byte, the most significant nibble (higher 4-bits) is sent first, immediately followed by the least significant nibble (lower 4-bits).

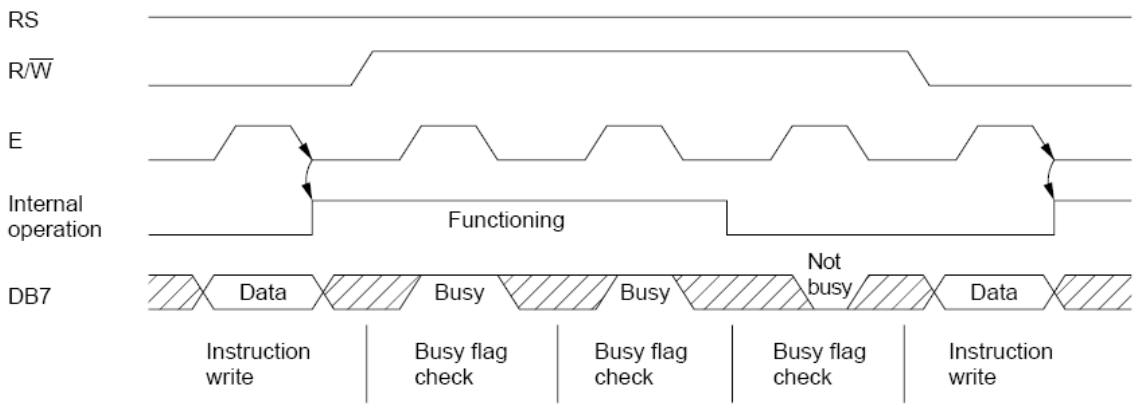


Figure 5.2: LCD interface timing diagram

For more details on the LCD, refer to "hd44780u.pdf" in the folder "datasheet" in the documentation CD.

5.1 Functions for the use of LCD

Note: All the functions are defined in the lcd.c file. It is located inside the "Experiments" folder inside the documentation CD.

5.1.1 LCD port configure (called inside the "port_init()" function)

```
void lcd_port_config (void)
{
 DDRC = DDRC | 0xF7; //all the LCD pin's direction set as output
 PORTC = PORTC & 0x80; // all the LCD pins are set to logic 0 except PORTC 7
}
```

5.1.2 Setting LCD in 4 bit mode

```
void lcd_set_4bit()
{
        _delay_ms(1);

        cbit(lcd_port,RS);          //RS=0 --- Command Input
        cbit(lcd_port,RW);          //RW=0 --- Writing to LCD
        lcd_port = 0x30;            //Sending 3 in the upper nibble
        sbit(lcd_port,EN);          //Set Enable Pin
        _delay_ms(5);               //delay
        cbit(lcd_port,EN);          //Clear Enable Pin
```

```
        _delay_ms(1);

        cbit(lcd_port,RS);      //RS=0 --- Command Input
        cbit(lcd_port,RW);      //RW=0 --- Writing to LCD
        lcd_port = 0x30;        //Sending 3 in the upper nibble
        sbit(lcd_port,EN);      //Set Enable Pin
        _delay_ms(5);           //delay
        cbit(lcd_port,EN);      //Clear Enable Pin

        _delay_ms(1);

        cbit(lcd_port,RS);      //RS=0 --- Command Input
        cbit(lcd_port,RW);      //RW=0 --- Writing to LCD
        lcd_port = 0x30;        //Sending 3 in the upper nibble
        sbit(lcd_port,EN);      //Set Enable Pin
        _delay_ms(5);           //delay
        cbit(lcd_port,EN);      //Clear Enable Pin

        _delay_ms(1);

        cbit(lcd_port,RS);      //RS=0 --- Command Input
        cbit(lcd_port,RW);      //RW=0 --- Writing to LCD
        lcd_port = 0x20;        //Sending 2 in the upper nibble to initialize LCD 4-bit mode
        sbit(lcd_port,EN);      //Set Enable Pin
        _delay_ms(5);           //delay
        cbit(lcd_port,EN);      //Clear Enable Pin
}
```

5.1.3 LCD initialization function

```
//Function to Initialize LCD
void lcd_init()
{
        _delay_ms(1);
        lcd_wr_command(0x28);  //4-bit mode and 5x8 dot character font
        lcd_wr_command(0x01);  //Clear LCD display
        lcd_wr_command(0x06);  //Auto increment cursor position
        lcd_wr_command(0x0E);  //Turn on LCD and cursor
        lcd_wr_command(0x80);  //Set cursor position
}
```

5.1.4 Function to write command on LCD

```
//Function to write command on LCD
void lcd_wr_command(unsigned char cmd)
{
        unsigned char temp;
        temp = cmd;
        temp = temp & 0xF0;
        lcd_port &= 0x0F;
        lcd_port |= temp;
        cbit(lcd_port,RS);
        cbit(lcd_port,RW);
        sbit(lcd_port,EN);
        _delay_ms(5);
        cbit(lcd_port,EN);

        cmd = cmd & 0x0F;
        cmd = cmd<<4;
        lcd_port &= 0x0F;
        lcd_port |= cmd;
        cbit(lcd_port,RS);
        cbit(lcd_port,RW);
        sbit(lcd_port,EN);
        _delay_ms(5);
        cbit(lcd_port,EN);
}
```

5.1.4 Function to write data on LCD

```
//Function to write data on LCD
void lcd_wr_char(char letter)
{
        char temp;

        temp = letter;
        temp = (temp & 0xF0);
        lcd_port &= 0x0F;
        lcd_port |= temp;
        sbit(lcd_port,RS);
        cbit(lcd_port,RW);
        sbit(lcd_port,EN);
        _delay_ms(5);
        cbit(lcd_port,EN);

        letter = letter & 0x0F;
        letter = letter<<4;
        lcd_port &= 0x0F;
        lcd_port |= letter;
        sbit(lcd_port,RS);
        cbit(lcd_port,RW);
        sbit(lcd_port,EN);
        _delay_ms(5);
        cbit(lcd_port,EN);
}
```

5.1.5 Function for LCD home

```
void lcd_home()
{
        lcd_wr_command(0x80);
}
```

5.1.6 Function to Print String on LCD

```
void lcd_string(char *str)
{
        while(*str != '\0')
        {
                lcd_wr_char(*str);
                str++;
        }
}
```

5.1.7 Position the LCD cursor at "row", "column"

```
//Position the LCD cursor at "row", "column"
void lcd_cursor (char row, char column)
{
        switch (row) {
                case 1: lcd_wr_command (0x80 + column - 1); break;
                case 2: lcd_wr_command (0xc0 + column - 1); break;
                case 3: lcd_wr_command (0x94 + column - 1); break;
                case 4: lcd_wr_command (0xd4 + column - 1); break;
                default: break;
        }
}
```

5.1.8 Function to print any input value up to the desired digit on LCD

```
// Function to print any input value up to the desired digit on LCD
void lcd_print (char row, char coloumn, unsigned int value, int digits)
{
        unsigned char flag=0;
        if(row==0||coloumn==0)
        {
                lcd_home();
        }
        else
        {
                lcd_cursor(row,coloumn);
        }
        if(digits==5 || flag==1)
        {
```

```
		million=value/10000+48;
		lcd_wr_char(million);
		flag=1;
	}
	if(digits==4 || flag==1)
	{
		temp = value/1000;
		thousand = temp%10 + 48;
		lcd_wr_char(thousand);
		flag=1;
	}
	if(digits==3 || flag==1)
	{
		temp = value/100;
		hundred = temp%10 + 48;
		lcd_wr_char(hundred);
		flag=1;
	}
	if(digits==2 || flag==1)
	{
		temp = value/10;
		tens = temp%10 + 48;
		lcd_wr_char(tens);
		flag=1;
	}
	if(digits==1 || flag==1)
	{
		unit = value%10 + 48;
		lcd_wr_char(unit);
	}
	if(digits>5)
	{
		lcd_wr_char('E');
	}
}
```

5.2 Application examples

5.2.1 Application example to print string on the LCD

Located in the folder “Experiments \ 6_LCD_interfcing” folder in the documentation CD.

This program shows how to write string on the LCD

Note:

1. Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: atmega16
Frequency: 7372800
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 4.22 in the hardware manual)

2. Buzzer is connected to PC3. Hence to operate buzzer without interfering with the LCD, buzzer should be turned on or off only using buzzer function

5.2.2 Application example to print sensor data on the LCD

It involves concept of ADC. It will be covered in chapter 7.

6. Analog to Digital Conversion

SPARK V has three white line sensors, Three Analog IR proximity sensors, Battery voltage sensing and optional ultrasonic range sensors. All these sensors give analog output. We need to use ATMEGA16 microcontroller's ADC (Analog to Digital Converter) to convert these analog values in to digital values.

Due to limited number of ADC channels with the Microcontroller ATMEGA16, either we can use IR Proximity sensors or Ultrasonic Range Sensors by setting jumpers J2, J3, J4 on the Robot. For jumper settings refer to section 3.9 and 3.10 from the hardware manual.

The ATMEGA16 has a 10-bit successive approximation Analog to Digital Converter (ADC). The ADC block is connected to a 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs from the pins of PORTA. The minimum value represents GND and the maximum value represents the voltage on the AREF pin (5 Volt in the case of SPARK V).

6.1 ADC Resolution

The resolution of the ADC indicates the number of discrete values it can produce over the range of analog values. The values are usually stored electronically in binary form, so the resolution is usually expressed in bits. In consequence, the number of discrete values available, or "levels", is usually a power of two. For example, an ADC with a resolution of 8 bits can encode an analog input to one in 256 different levels, since $2^8 = 256$. The values can represent the ranges from 0 to 255 (i.e. unsigned integer) or from -128 to 127 (i.e. signed integer), depending on the application.

ATMEGA16 microcontroller has ADC with 10 bit resolution.

$V_{resolution} = V_{full\ scale} / 2^n - 1$
Where $V_{full\ scale}$ = 5V; n = 10 or 8

Case 1: n = 10 (10 bit resolution)
$V_{resolution} = 5V / 2^{10} - 1$
$V_{resolution} = 4.8875mV$

Case 2: n = 8 (8 bit resolution)
$V_{resolution} = 5V / 2^8 - 1$
$V_{resolution} = 19.6078mV$

6.2 Registers for ADC

6.2.1 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	ADEN	**ADSC**	**ADATE**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**
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 – ADEN: ADC Enable
Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress will terminate this conversion.

Bit 6 – ADSC: ADC Start Conversion
In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC. ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

Bit 5 – ADATE: ADC Auto Trigger Enable
When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

Bit 4 – ADIF: ADC Interrupt Flag
This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set.ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

Bit 3 – ADIE: ADC Interrupt Enable
When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Table 6.1 ADC prescaler selections

6.2.2 ADMUX– ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
	REFS1	**REFS0**	**ADLAR**	**MUX4**	**MUX3**	**MUX2**	**MUX1**	**MUX0**
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 7:6 – REFS1:0: Reference Selection Bits
These bits select the voltage reference for the ADC, as shown in Table 6.2. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal VREF turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Table 6.2: Voltage reference selection for ADC

Bit 5 – ADLAR: ADC Left Adjust Result
The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see "ADCL and ADCH – The ADC Data Register" in the section 7.2.5.

Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits
The value of these bits selects which combination of analog inputs is connected to the ADC. See Table 6.3 for details. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set)

MUX4:0	ADC pin	Pin function	Pin status
000000	PA0/ADC0	ADC input for Left IR proximity analog sensor (or Left Ultrasonic Range Sensor)	Input (Floating)
000001	PA1/ADC1	ADC input for Centre IR proximity analog sensor (or Centre Ultrasonic Range Sensor)	Input (Floating)
000010	PA2/ADC2	ADC input for Right IR proximity analog sensor (or Right Ultrasonic Range Sensor)	Input (Floating)

000011	PA3/ADC3	ADC input for Left white line sensor	Input (Floating)
000100	PA4/ADC4	ADC input for Centre white line sensor	Input (Floating)
000101	PA5(ADC5)	ADC input for Right white line sensor	Input (Floating)
000110	PA6/(ADC6)	ADC input for battery voltage monitoring	Input (Floating)
000111	PA7(ADC7)	ADC input for 4th Ultrasonic Range Sensor	Input (Floating)

Table 6.3 Input channel selection and functions

6.2.4 ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0
	ACD	**ACBG**	**ACO**	**ACI**	**ACIE**	**ACIC**	**ACIS1**	**ACIS0**
Read / Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – ACD: Analog Comparator Disable
When this bit is written logic one, the power to the Analog Comparator is switched off. This bit can be set at any time to turn off the Analog Comparator. This will reduce power consumption in Active and Idle mode. When changing the ACD bit, the Analog Comparator Interrupt must be disabled by clearing the ACIE bit in ACSR. Otherwise an interrupt can occur when the bit is changed.

Bit 6 – ACBG: Analog Comparator Band gap Select
When this bit is set, a fixed band gap reference voltage replaces the positive input to the Analog Comparator. When this bit is cleared, AIN0 is applied to the positive input of the Analog Comparator. When the band gap reference is used as input to the Analog Comparator, it will take a certain time for the voltage to stabilize. If not stabilized, the first conversion may give a wrong value. For more information see “Internal Voltage Reference” on page 39 of the ATMEGA16 datasheet.

Bit 5 – ACO: Analog Comparator Output
The output of the Analog Comparator is synchronized and then directly connected to ACO. The synchronization introduces a delay of 1 - 2 clock cycles.

Bit 4 – ACI: Analog Comparator Interrupt Flag
This bit is set by hardware when a comparator output event triggers the interrupt mode defined by ACIS1 and ACIS0. The Analog Comparator interrupt routine is executed if the ACIE bit is set and the I-bit in SREG is set. ACI is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ACI is cleared by writing a logic one to the flag.

Bit 3 – ACIE: Analog Comparator Interrupt Enable
When the ACIE bit is written logic one and the I-bit in the Status Register is set, the Analog Comparator interrupt is activated. When written logic zero, the interrupt is disabled.

Bit 2 – ACIC: Analog Comparator Input Capture Enable
When written logic one, this bit enables the input capture function in Timer/Counter1 to be triggered by the Analog Comparator. The comparator output is in this case directly connected to the input capture front-end logic, making the comparator utilize the noise canceller and edge select features of the Timer/Counter1 Input Capture interrupt. When written logic zero, no connection between the Analog Comparator and the input capture function exists. To make the

comparator trigger the Timer/Counter1 Input Capture interrupt, the ICIE1 bit in the Timer Interrupt Mask Register (TIMSK1) must be set.

Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select
These bits determine which comparator events that trigger the Analog Comparator interrupt. The different settings are shown in Table 6.4.

ACIS1	ACIS0	Interrupt mode
0	0	Comparator Interrupt on Output Toggle
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge
1	1	Comparator Interrupt on Rising Output Edge

Table 6.4: ACIS1/ACIS0 settings

6.2.5 ADCL and ADCH – The ADC Data Register

Case 1: ADLAR = 0;

Initial value	0	0	0	0	0	0	0	0
Read / Write	R	R	R	R	R	R	R	R
Bit	15	14	13	12	11	10	9	8
ADCH							**ADC9**	**ADC8**
ADCL	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADC1**	**ADC0**
Bit	7	6	5	4	3	2	1	0
Read / Write	R	R	R	R	R	R	R	R
Initial value	0	0	0	0	0	0	0	0

Case 2: ADLAR = 1; (Left adjust)

Initial value	0	0	0	0	0	0	0	0
Read / Write	R	R	R	R	R	R	R	R
Bit	15	14	13	12	11	10	9	8
ADCH	**ADC9**	**ADC8**	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**
ADCL	**ADC1**	**ADC0**						
Bit	7	6	5	4	3	2	1	0
Read / Write	R	R	R	R	R	R	R	R
Initial value	0	0	0	0	0	0	0	0

When an ADC conversion is complete, the result is found in these two registers. If differential channels are used, the result is presented in two's complement form. When ADCL is read, the ADC Data Register is not updated until ADCH is read. Consequently, if the result is left adjusted and no more than 8-bit precision (7 bit + sign bit for differential input channels) is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH. The ADLAR bit in ADMUX and the MUXn bits in ADMUX affect the way the result is read from the registers. If ADLAR is set, the result is left adjusted. If ADLAR is cleared (default), the result is right adjusted.

6.3 Functions for ADC

6.3.1 Function to configure pins for ADC (called inside the "port_init()" function)

```
//ADC pin configuration
void adc_pin_config (void)
{
```

```
 DDRA = 0x00; //set PORTA direction as input
 PORTA = 0x00; //set PORTA pins floating
}
```

6.3.2 Function to configure ADC

```
//Function to Initialize ADC
void adc_init()
{
        ADCSRA = 0x00;
        ADMUX = 0x20;           //Vref=5V external --- ADLAR=1 --- MUX4:0 = 0000
        ACSR = 0x80;
        ADCSRA = 0x86;          //ADEN=1 --- ADIE=1 --- ADPS2:0 = 1 1 0
}
```

6.3.3 Function to initialize ADC

```
void init_devices (void)
{
 cli(); //Clears the global interrupts
 port_init();
 adc_init();
 sei(); //Enables the global interrupts
}
```

6.3.4 Function to get ADC value

```
//This Function accepts the Channel Number and returns the corresponding Analog Value
unsigned char ADC_Conversion(unsigned char Ch)
{
        unsigned char a;
        Ch = Ch & 0x07;
        ADMUX= 0x20| Ch;
        ADCSRA = ADCSRA | 0x40;      //Set start conversion bit
        while((ADCSRA&0x10)==0);     //Wait for ADC conversion to complete
        a=ADCH;
        ADCSRA = ADCSRA|0x10;        //clear ADIF (ADC Interrupt Flag) by writing 1 to it
        ADCSRB = 0x00;
        return a;
}
```

6.4 Application examples

6.4.1 Application example to display ADC sensor data on the LCD(IR Proximity sensor)

Located in the folder “Experiments \ 7A_ADC_IRWLBAT_Sensor_display_On_LCD” folder in the documentation CD.

6.4.1 Application example to display ADC sensor data on the LCD (Ultrasonic Range Sensor)

Located in the folder “Experiments \ 7B_ADC_URSWLBAT_Sensor_display_On_LCD” folder in the documentation CD.

6.4.2 Application example to follow white line following

Located in the folder “Experiments \8_ White_Line_Following” folder in the documentation CD.

6.4.3 Application example to perform Adaptive Cruise Control (ACC) while following the white line with front obstacle detection using IR Proximity sensor

Located in the folder “Experiments \ 9A_Adaptive_Cruise_Control_IR” folder in the documentation CD.

Note for all the application examples:

1. Make sure that in the configuration options following settings are done for proper operation of the code

    Microcontroller: atmega16
    Frequency: 7372800
    Optimization: -O0
    (For more information read section: Selecting proper optimization options below figure 4.22 in the hardware manual)

2. Make sure that you copy the lcd.c file in your folder

3. Put the Jumper on J2, J3 and J4 connector setting for selecting the either IR Proximity Sensor or Ultrasonic Range Sensor. Refer to section 3.9 and 3.10 from the hardware manual for the correct jumper settings.

4. For more details on sensor interfacing and hardware connection of jumper setting refer the chapter 3 of “SPARKV Hardware Manual” in the documentation CD.

7. Serial Communication

Serial Communication using USART

The SPARK V can communicate with other robots / devices serially using either wired link or wireless module. Serial communication is done in asynchronous mode. In the asynchronous mode, the common clock signal is not required at both the transmitter and receiver for data synchronization.

SPARKV Robot Support following two mode of serial communication

1. USB Communication using onboard FT232 USB to serial converter.
2. ZigBee Wireless Communication with ZigBee wireless module is installed on ZigBee holders.

ATMEGA16 have single USART port available for serial communication. Serial port of the ATMEGA16 can be switched between USB port and Xbee wireless module using jumper setting connector J5. Please refer the chapter 8 in Hardware manual for more details about USB/ ZigBee communication interfacing and connection details.

7.1 Registers involved in the serial communication

7.1.1 UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	RXC	**TXC**	**UDRE**	**FE**	**DOR**	**UPE**	**U2X**	**MPCM**
Read / Write	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – RXC: USART Receive Complete
This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (i.e., does not contain any unread data). If the Receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit in the section 7.1.2).

Bit 6 – TXC: USART Transmit Complete
This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see descript ion of the TXCIE bit in the section 7.1.2).

Bit 5 – UDRE: USART Data Register Empty
The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data

Register Empty interrupt (see description of the UDRIE bit). UDRE is set after a reset to indicate that the Transmitter is ready.

Bit 4 – FE: Frame Error
This bit is set if the next character in the receive buffer had a Frame Error when received. I.E., when the first stop bit of the next character in the receive buffer is zero. This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.

Bit 3 – DOR: Data OverRun
This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the Receive Shift Register, and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

Bit 2 – UPE: USART Parity Error
This bit is set if the next character in the receive buffer had a Parity Error when received and the Parity Checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

Bit 1 – U2X: Double the USART Transmission Speed
This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation. Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.

Bit 0 – MPCM: Multi-processor Communication Mode
This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCM setting. For more detailed information see “Multi-processor Communication Mode” on page 154 of the ATMEGA16 datasheet.

7.1.2 UCSRB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0
	RXCIE	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**UCSZ2**	**RXB8**	**TXB8**
Read / Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit 7 – RXCIE: RX Complete Interrupt Enable
Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

Bit 6 – TXCIE: TX Complete Interrupt Enable
Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable
Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

Bit 4 – RXEN: Receiver Enable
Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RXD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and UPE Flags.

Bit 3 – TXEN: Transmitter Enable
Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxD port.

Bit 2 – UCSZ2: Character Size
The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

Bit 1 – RXB8: Receive Data Bit 8
RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.

Bit 0 – TXB8: Transmit Data Bit 8
TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. Must be written before writing the low bits to UDR.

7.1.3 UCSRC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0
	URSEL	**UMSEL**	**UPM1**	**UPM0**	**USBS**	**UCSZ1**	**UCSZ0**	**UCP0L**
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

The UCSRC Register shares the same I/O location as the UBRRH Register. See the "Accessing UBRRH/ UCSRC Register" on page 155 in Atmega16 datasheet which describes how to access this register.

Bits 7 – URSEL: Register Select
These bits selects between the UCSRC or UBRRH Registers. It is read as one when Reading UCSRC. The URSEL must be one when writing to UCSRC.

Bits 6 – UMSEL: USART Mode Select
These bits select the mode of operation of the USART as shown in Table 7.1.

UMSEL	Mode
0	Asynchronous USART
1	Synchronous USART

Table 7.1: UMSEL Bit settings

Bits 5:4 – UPM1:0: Parity Mode
These bits enable and set type of parity generation and check. If enabled, the Transmitter will automatically generate and send the parity of the transmitted data bits within each frame. The Receiver will generate a parity value for the incoming data and compare it to the UPM setting. If a mismatch is detected, the UPE Flag in UCSRA will be set.

UPM1	UPM0	Parity mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Table 7.2: UPM Bits settings

Bit 3 – USBS: Stop Bit Select
This bit selects the number of stop bits to be inserted by the Transmitter. The Receiver ignores this setting.

USBS	Stop Bit(s)
0	1-bit
1	2-bit

Table 7.3: USBS bit settings

Bit 2:1 – UCSZ1:0: Character Size
The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

UCSZ2	UCSZ1	UCSZ0	Character size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit

1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Table 7.4: UCSZ Bits Settings

Bit 0 – UCPOL: Clock Polarity

This bit is used for synchronous mode only. Write this bit to zero when asynchronous mode is used. The UCPOL bit sets the relationship between data output change and data input sample, and the synchronous clock (XCK).

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RXD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

Table 7.5: UCPOL Bit Settings

7.1.4 UBRRL and UBRRH – USART Baud Rate Registers

Initial value	0	0	0	0	0	0	0	0
Read / Write	R	R	R	R	R/W	R/W	R/W	R/W
Bit	15	14	13	12	11	10	9	8
ADCH	-	-	-	-	**UBRR11**	**UBRR10**	**UBRR9**	**UBRR8**
ADCL	**UBRR7**	**UBRR6**	**UBRR5**	**UBRR4**	**UBRR3**	**UBRR2**	**UBRR1**	**UBRR0**
Bit	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
Initial value	0	0	0	0	0	0	0	0

Baud rate calculation:

Crystal frequency: 7372800 Hz

Required baud rate: 9600 bits per second

UBRR = (System Clock / (16 * baud rate)) – 1
= (7372800Hz / (16 * 9600)) – 1
= 47
= 0x2F (hex)

UBRRH = 0x00

UBRRL = 0x2F

Baud rate	2400	4800	9600	14.4k	19.2k	28.8k	38.4k	57.6k	76.8k	115.2k
UBRR	191	95	47	31	23	15	11	7	5	3

Table 7.6: Value of UBRR for different baud rate for 7.3728MHz crystal

For 7.3728MHz crystal frequency, the most commonly used baud rates for asynchronous operation can be generated by using the UBRR settings as shown in the table 8.6.

Note:

While loading values in the UBRR register load values in the UBRRH resistor first and then in UBRRL register.

7.1.5 UDR – USART I/O Data Register

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the Transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use Read-Modify-Write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

7.2 Functions used in serial communication

7.2.1 Function to configure UART

```
//Function To Initialize UART
//desired baud rate:9600
//actual baud rate:9600 (error 0.0%)
//char size: 8 bit
//parity: Disabled
void uart_init(void)
{
 UCSRB = 0x00; //disable while setting baud rate
 UCSRA = 0x00;
 UCSRC = 0x86;
 UBRRL = 0x2F; //set baud rate lo
 UBRRH = 0x00; //set baud rate hi
 UCSRB = 0x98;
}
```

7.2.2 Function to initialize UART

```
void init_devices()
{
 cli(); //Clears the global interrupts
 port_init();  //Initializes all the ports
 uart_init(); //Initialize UART for serial communication
 sei();   //Enables the global interrupts
}
```

7.2.3 Receive complete ISR

When UART receives eight data bits on receive pin of the microcontroller, RXC flag is set. If RXCIE interrupt is enabled then receive complete interrupt triggers ISR. This ISR then reads valid data from UDR and stores it in a separate variable before next character is received and overwritten. It is always recommended to save data read from UDR in a separate variable as next character received will overwrite and destroy the existing data in UDR.

```
SIGNAL(SIG_UART_RECV)       // ISR for receive complete interrupt
{
        receive_data = UDR;       //making copy of data from UDR in 'data' variable
        //Insert your coder here
}
```

7.2.4 Data register empty ISR

The transmitter side of the UART is double buffered containing UDRn to hold the data written from the program and transmit register to actually transmit parallel data sequentially bit-by-bit on the transmit pin. The data written to UDR is transferred to transmit register. At this point, the UDR is available to accept next data word from the program. This sets UDRE flag and if UDRIE interrupt is enabled then UDR data register empty interrupt triggers ISR. This ISR then loads next data byte to be transmitted into UDR.

```
SIGNAL(SIG_UART_RECV)
{
        UDR = tx_data;
         // Insert your code here…….
}
```

7.2.5 Transmit complete ISR

In the case of packet based data communication it is necessary to know when a byte has been completely transmitted out of microcontroller. The TXC flag is provided to indicate that the transmit register is empty and no new data is waiting to be transmitted. If transmit register is empty it sets TXC flag and if TXCIE interrupt is enabled then Transmit complete interrupt triggers ISR. This ISR can be used as a confirmation of the byte that was loaded in UDR is successfully transmitted out of the microcontroller transmit pin. This interrupt can be used to check if all the bytes in a packet transmission are transmitted successfully.

```
SIGNAL (SIG_USART_TRANS)
{
        //Insert your code here……..
}
```

7.3 Application example for serial communication

Note:
All the application examples are identical in nature.
Robot can be controlled using wired or wireless link using PC with these application examples.
Refer to chapter 8 from the hardware manual for using these application examples.

7.3.1 USB communication using FT232 USB to serial converter

Located in the folder "Experiments \ 10A_Serial_Communication_USB-RS232" folder in the documentation CD.

7.3.2 Serial communication over wireless using ZigBee wireless module

Located in the folder "Experiments \ 10B_Serial_Communication_ZigBee_wireless" folder in the documentation CD.

Note for all the application examples:
1. Make sure that in the configuration options following settings are done for proper operation of the code

Microcontroller: atmega16
Frequency: 7372800
Optimization: -O0
(For more information read section: Selecting proper optimization options below figure 4.22 in the hardware manual)

2. Also put the Jumper on J5 connector setting for selecting the either USB Module or ZigBee Wireless Module.

3. For more details on sensor interfacing and hardware connection of jumper setting refer the chapter 3 of "SPARKV Hardware Manual" in the documentation CD.