Title of script: Basic IO interfacing

Author: Saurav Shandilya

Keywords: ports, buzzer, interfacing

|  |  |
| --- | --- |
| Slides | Narration |
| 01.  Title | * Hello Friends. * Welcome to the spoken tutorial on Firebird V Robotics Research Platform. * This platform is based on LPC 2148 microcontroller which belongs to ARM architecture based microcontroller family. * In this tutorial, we will learn the **Basic IO interfacing on Firebird V robot**.  |  | | --- | |  |   **(press next)** |
| 02.  Agenda | * Now let us see the agenda for discussion in this tutorial. * The presentation will start with the discussion of Input/Output ports of LPC 2148, which includes the overview of the ports and method of accessing the ports with the help of examples. Then we will learn how to write our first Embedded C program for the Buzzer Interfaced on the Firebird V robot.   **(press next)** |
| 03.  What are Ports? | * So, what are ports? **(press next)** * Ports can be defines as junction where peripherals devices can be connected. **(press next)** * Peripherals devices can be classified as firstly **(press next)** * input devices; Input devices are the devices which gives signals to the microcontroller. Example of input devices are switch, sensors etc.. **(press next)** * Other peripherals device is called as output devices. So Output devices are devices to which the microcontroller will send the signals example of output devices are buzzer, LCD, Motors, Led etc.   **(press next)** |
| 04.  Ports in Atmega2560 | * Let us now see the ports present in LPC 2148. **(press next)** * **LPC** stands Low Power consumption. * LPC 2148 is a 64 pin micro-controller. **(press next)** * Out of which 45 pins can be used as Input or output Pins for I/O interfacing. **(press next)** * These pins are grouped together and are called as Ports. The pins are called as ports because if you want to access any of the peripheral devices you have to do it by putting the address of it, that is to which pin particular input/output device is connected. To address 45 pins is very difficult so we have grouped the pins and have called them as Ports. **(press next)** * In LPC 2148 we have two 32-bit ports named as PORTx where x is name of the port, and x can be as 0 or 1. **(press next)** * All Port can be configured individually as input or output pin as per device connected to it.   **(press next)** |
| 05.  Accessing Ports | * Now let us see how to access the port. **(press next)** * In AVR architecture each port has 3 registers associated with it. So we have to use these 3 registers to define each port which we are going to use. **(press next)** * The first register is DDRx register where x can be replaced by the name if any of the port register. i.e.A to H and J,K,L **(press next)** * The second register is PORTx register **(press next)** * and similarly we have third register which is called as PINx register. So now let us study each of the register in detail.   **(press next)** |
| 06.  Understanding DDRx Register | * So the first register is DDRx register **(press next)** * DDRx register stands for Data Direction Register . **(press next)** * The purpose of this register is to set the PORT pins as input or output. **(press next)** * If we assign the value Zero to DDRx register, and we set the PORTx pins as input pin * Similarly If we assign the value One to DDRx register, we set the PORTx pins as output pin. **(press next)** * Let us see a example for getting more clearer concept. Suppose we need to make PORTB Lower nibble as input and upper nibble as output. So for this what should be the logic? **(press next)** * So the logic for it would be to assign the value 0 to port pin D0 to D3 and the value 1 to pin number D4 to D7. We represent the value in Hexadecimal notation which will give the convenient and standard notation while programming in C-Program. **(press next)** * So here the value is written as DDRB= 0xF0.   **(press next)** |
| 07.  Understanding PINx register | * Now let us consider the PINx register. **(press next)** * The purpose of this register is to read the value from the devices connected on the PORT pins. **(press next)** * If we define a PORT as input port then we read the values present on the input device using the PINx register. The value read from the PINx register is to be saved in the user defined variable. **(press next)** * Let us consider an example, We need to read the data from the PORTC register. **(press next)** * Suppose PORTC as following bit-pattern. So the step would be to define a variable, here we are calling it as x **(press next)** * Then we perform x = PINC. So the value of the register PORTC will be stored in variable x . **(press next)** * In this case the value in x would be 0xF0.So you can read any value stored in the register using the PIN register.   **(press next)** |
| 08  Understanding PORTx register  (case 1) | * Now let us see the third register which is PORTx register. PORTx register operates in two different ways depending on the way in which the port is defined. **(press next)** * First case is if port is defined as the output **(press next)** * we use PORT register to output desired value to the output device. **(press next)** * For example: **(press next)** * Here we have defined the portA as output by assigning **(press next)** * the value of DDRA=0xFF. **(press next)** * In this case if we assign the value as PORTA=0xFF which means the logic high to the device connected to PORTA .Now suppose if the device is LED then the LED will glow on the logic high. Similarly, if PORTA is written as 0x00, LEDs will turn off.   **(press next)** |
| 09  Understanding PORTx register  (case 2) | * Now let us see the second case **(press next)** * In this case when the port is defined as the input port. **(press next)** * In such cases the PORTx register is used to activate or deactivate the pull-up resistor. So what are pull-up resistor? The pull-up resistors are used to assures that the input to the pin is maintained at a desired logic level even if no active devices are connected to it .This is done to avoid any external noise or interference to our readings and hence to minimize the error. **(press next)** * So to activate the pull-up resistor we assign the value as PORTx= 1 **(press next)** * and to deactivate we assign the value as PORTx=0. **(press next)** * Let us consider example now **(press next)** * in this we make port A as input port using DDRA=0x00 **(press next)** * then to activate the pull-up we assign the value PORTA=0xFF. **(press next)** * So the pull up resistors is activated for all the pins of port A.   **(press next)** |
| 10  Examples  (example 1) | * So we now have covered the overview of the ports and how to define the ports using the register .Now let us understand the concept with few more examples. **(press next)** * Consider example 1 where we need to make PORTD as output port and send bit pattern of D5 to it. So the steps involved in the example will be as follows**(press next)** * First step will be to make Port D as Output port **(press next)** * So value in DDRD register will be **(press next)** * 0xFF. **(press next)** * Then second step will be to Put data on the Port D register **(press next)** * So the value in PORTD register will be **(press next)** * 0xD5   **(press next)** |
| 11.  Example (Cont..) | * Let us now consider the second example. **(press next)** * In this we have to make port A as input port with pull-up activated on all the pins. So the steps involved in the example will be as follows: **(press next)** * First step will be to make Port A as Input port **(press next)** * So value in DDRA register will be **(press next)** * 0x00 **(press next)** * Then the second step will be to activate Pull-up Resistor on Port A**(press next)** * So value in PORTA will be **(press next)** * 0xFF   **(press next)** |
| 12  Example (Cont..) | * Let us now consider the next example. **(press next)** * In this we have to make upper nibble of PORTD as output and lower nibble as input. Then we have to Output value 3 on upper nibble and read back data using PIND. So the steps involved in the example will be as follows **(press next)** * First Step will be to define port D Pin7 to Pin4 as output and Pin3 to Pin0 as input pins **(press next)** * Therefore value of DDRD register will be **(press next)** * 0xF0. **(press next)** * Then the second step will be to put data on Port D Pin7 to Pin4 and enable pull-up for Pin3 to Pin0 **(press next)** * So the value of PORTD register will be **(press next)** * 0x3F. **(press next)** * The last step will be to read the data from Port D register. This will be done using PIN register. **(press next)** * So we can write x = PIND. **(press next)** * Therefore value of x will be 0x3F. **(press next)** * With this we complete the configuration of ports in detail and we can now begin with our first program.   **(press next)** |
| 13.  Buzzer interfacing in Firebird V | * The first program in this tutorial is to interface the buzzer on Firebird V platform. **(press next)** * Buzzer Connected to PortC pin 3. **(press next)** * This is simple diagram of buzzer connection, in which buzzer is connected to PC3 as shown in the figure. **(press next)** * To Turn on the buzzer**(press next)** * We send logic HIGH on pin3 of PortC. **(press next)** * And to Turn off the buzzer**(press next)** * We must send logic LOW on pin3 of PortC.   **(press next)** |
| 14  Buzzer Program | * Now let us proceed with writing the program **(press next)** * we should first configure PORTC, pin 3 as output **(press next)** * So this is done by using DDRC register **(press next)** * So value of DDRC should be 0x08. **(press next)** * Now to turn ON the buzzer set Port C pin 3 as hi**(press next)**gh * So value of PORTC will be **(press next)** * 0x08. **(press next)** * And to turn OFF the buzzer set Port C pin3 as low **(press next)** * So value of PORTC will be **(press next)** * 0x00.   **(press next)** |
| 15  AVR Programming  Tools | * Now let us look at how we are going to write the program. For writing microcontroller program and making it work on the robot we require few tools, here we will have brief introduction to tools required for programming the AVR microcontroller. **(press next)** * The Software required are **(press next)** * ATMEL STUDIO 6, **(press next)** * It is an IDE which stands for Integrated Development Environment (IDE). An IDE typically consist of an editor, debugger, compiler and an interpreter. Atmel Studio supports Developing and Debugging of AVR and ARM based microcontroller application. We will write the program in C language and compile it in the Atmel studio to generate the hex file which is the machine language understandable by the microcontroller. Atmel studio can be downloaded from the given link. **(press next)** * After generating the hex file, we need to load it into memory of microcontroller. For this we need a hardware device. **(press next)** * So the hex file can be loaded into the microcontroller using the hardware such as bootloader **(press next)** * or AVR Programmers such as AVR MKII, AVRDude, Pony-Programmer, etc.   **(press next)** |
| 16  Syntax | * Let us now begin writing the C program. In coming slides we put the syntax for writing the C program. **(press next)** * So the standard C program begin with including the header files. **(press next)** * Here in this program we include some of header files which are   #include <avr/io.h>  #include <avr/interrupt.h>  #include <util/delay.h> **(press next)**   * Next we need to configure the ports pin as input or output. **(press next)** * We are using a function defined as buzzer\_pin\_config. Then we assign the value to DDRC register in a way as it was explained earlier to make buzzer as output device and value of PORTC will have the initial value as 0x00 i.e. initially switch the buzzer off. We have intentionally not written the values here. You should fill it yourself.   **(press next)** |
| 17.  C-program | * Now let us define the Main program. **(press next)** * In the main function initially we have called the buzzer\_pin\_config function. Next we have called an infinite while loop in which we call the function buzzer\_on (), to switch on the buzzer. Next is the delay function writtenas \_delay\_ms() which has the parameters to be sent in the milliseconds to introduce the required delay for which buzzer should be high. Next we call the buzzer\_off and then again give necessary delay. So this function will be called in the while loop making the buzzer on and off at particular interval depending upon the delay. **(press next)** * Now let look at the functions defined in the main program. It is always a good programming practice to write the code by making different function. **(press next)** * So the first function is of buzzer\_on which will have the high logic stored in PORTC register. **(press next)** * The second function is of buzzer\_off. which will have low logic stored in PORTC register.   Now let us look at the demonstration. We will start with writing the C-program for buzzer interfacing in ATMEL STUDIO. So Open the Atmel Studio by clicking on the icon.  **(Minimize the presentation and go to desktop)**  So let me do it. You can click the icon from here.  **(Open Atmel Studio window)** So we have already created the project for buzzer programming. You can refer to the tutorial on Using ATMEL STUDIO for knowing more details on how to create projects and other relevant details.  **(On Screen: Editor of ATMEL STUDIO)**SoI have here written the program in editor. You can write the code by yourself on the editor which you are using. As you can here, code is same as explained to you in slides. I will be explaining you each of the line individually. So the first line here #define F\_CPU 14745600 this line sets the clock frequency at which microcontroller is working. Its value is determined by the crystal oscillator attached to the controller. In Firebird V crystal frequency is 14745600Hz. You can change the value if crystal oscillator has some different value. Next instructions are for including various header file. I have included three header files. Then this is the function for buzzer pin config, so it configures the buzzer pin where buzzer is connected as output. This is the concept of masking used here. So masking is done to set only single pin while we are working and rest of the pins is left in previous state unconnected. This is the function for switching on the buzzer. So here we have written PortC =0x08, so this will put a logic high i.e a value of one on my portC pin number 3. Then this is the buzzer off function. So this will put zero value on PortC pin number 3. Then this function for init\_devices. This is a standard function which we will be using in all programs. This function calls functions which configure pin as output or input and it call which are used for hardware initialization. So in this case we are not using any hardware initilizatio. Later when you use PWM or ADC, at that time you will see how we use those also. Now after this we have a main function. So in main function I have called this init devices, the first statement and then inside an infinite loop, I have written buzzer on. So this will switch on my buzzer and then a delay. So buzzer will remain on for One second and then buzzer will go off and it will be off for one second and after one second again buzzer will be on and so this loop will go on infinitely. You can change the logic by changing the program. You can make buzzer ring for ten time by using for loop etc.  Now after completing writing the code, next step is to build the program for generating the hex file. For this click on the Build tab at top, inside that click on build buzzer. Once you have clicked the build buzzer you can see the process going on in output window at the bottom. So Once build is complete you can view output window which shows the status. As you can see here, it is showing 1 succeeded. So one file has been successfully build. Now after building, we are generating a hex file. You can look in the folder where hex file has been generated. Now let me just introduce a error here and let me show you how to tackle those situation. So I have removed the semicolon from here and now I will try to build it again. So as you can see here, a error is coming. So double click on this error, it will give you the location where the error is. So you can see here, it is pointing that we have missed a semi-colon, so just introduce the semicolon and then try building it again. So you will see that build will succeed. So in this way you can troubleshoot various program by looking at the output window and try to troubleshoot the error which are coming.  Now the hex file for the code is generated. So you can view that hex file by locating it in the folder you created for the program. We will now burn this program into the microcontrollers memory. This is done using Bootloader. You can refer to spoken tutorial on how to use Bootloader technique. So let us open AVR Bootloader software, provide all details such as COM Port number, Baud Rate should be 115200, Select microcontroller as ATmega2560, browse the hex file. Now switch on the robot and invoke the boot mode for loading the program using Bootloader. Boot mode is invoked by pressing and releasing reset switch with boot switch pressed. Release boot switch. Now click on Program button in AVR Bootloader. You can see the led blinking, which means program is getting loaded. Once hex file is completely loaded, press reset button or switch off-on you robot. The program will now work and you can see its execution on the robot. If in case you face problem in hex file getting loaded using AVR Bootloader, try troubleshooting by checking com port number or re-inserting the usb cable and restarting the Bootloader software.  So here we have successfully understood the concept of port operatiojn in VAR based microcontroller and have written a program for ringing the buzzer. |
| 18.  Thank you | With this we have come to end to this tutorial. Thank you for listening. For any queries or doubts you can visit <http://qa.e-yantra.org/>  This is Saurav Shandilya Signing off!! |