PANIMALAR ENGINEERING COLLEGE

(A CHRISTIAN MINORITY INSTITUTION)

JAISAKTHI EDUCATIONAL TRUST

ACCREDITED BY NATIONAL BOARD OF ACCREDITATION (NBA)

Bangalore Trunk Road, Varadharajapuram, Nasarathpettai,

Poonamallee, Chennai – 600 123.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

EC6711 – EMBEDDED LABORATORY

IV ECE- VII SEMESTER

2017-2018(ODD SEMESTER)

DEPARTMENT OF ECE

VISION

To emerge as a centre of excellence in providing quality education and produce technically competent Electronics and Communication Engineers to meet the needs of industry and Society.

MISSION

M1: To provide best facilities, infrastructure and environment to its students, researchers and faculty members to meet the Challenges of Electronics and Communication Engineering field.

M2: To provide quality education through effective teaching – learning process for their future career, viz placement and higher education.

M3: To expose strong insight in the core domains with industry interaction.

M4: Prepare graduates adaptable to the changing requirements of the society through life long learning.

PROGRAMME EDUCATIONAL OBJECTIVES

1. To prepare graduates to analyze, design and implement electronic circuits and systems using the knowledge acquired from basic science and mathematics.
2. To train students with good scientific and engineering breadth so as to comprehend, analyze, design and create novel products and solutions for real life problems.
3. To introduce the research world to the graduates so that they feel motivated for higher studies and innovation not only in their own domain but multidisciplinary domain.
4. Prepare graduates to exhibit professionalism, ethical attitude, communication skills, teamwork and leadership qualities in their profession and adapt to current trends by engaging in lifelong learning.
5. To practice professionally in a collaborative, team oriented manner that embraces the multicultural environment of today's business world.

PROGRAMME OUTCOMES

1. **Engineering Knowledge:** Able to apply the knowledge of Mathematics, Science, Engineering fundamentals and an Engineering specialization to the solution of complex Engineering problems.
2. **Problem Analysis:** Able to identify, formulate, review research literature, and analyze complex Engineering problems reaching substantiated conclusions using first principles of Mathematics, Natural sciences, and Engineering sciences.
3. **Design / Development of solutions:** Able to design solution for complex Engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Able to use Research - based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Able to create, select and apply appropriate techniques, resources, and modern Engineering IT tools including prediction and modeling to complex Engineering activities with an understanding of the limitations.
6. **The Engineer and society: A**ble to apply reasoning informed by the contextual knowledge to access societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Engineering practice.
7. **Environment and sustainability:** Able to understand the impact of the professional Engineering solutions in societal and environmental context, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Able to apply ethical principles and commit to professional ethics and responsibilities and norms of the Engineering practice.

9. **Individual and Team work:** Able to function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Able to communicate effectively on complex Engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and Finance:** Able to demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life – long learning:** Able to recognize the needs for, and have the preparation and ability to engage in independent and life-long learning in the broadest contest of technological change.

PROGRAMME SPECIFIC OUTCOMES

1. Graduates should demonstrate an understanding of the basic concepts in the primary area of Electronics and Communication Engineering, including: analysis of circuits containing both active and passive components, electronic systems, control systems, electromagnetic systems, digital systems, computer applications and communications.
2. Graduates should demonstrate the ability to utilize the mathematics and the fundamental knowledge of Electronics and Communication Engineering to design complex systems which may contain both software and hardware components to meet the desired needs.
3. The graduates should be capable of excelling in Electronics and Communication Engineering industry/Academic/Software companies through professional careers.

SYLLABUS

EC6711 EMBEDDED LABORATORY **L T P C 0 0 3 2**

OBJECTIVES: The student should be made to:

- ✓ Learn the working of ARM processor
- ✓ Understand the Building Blocks of Embedded Systems
- ✓ Learn the concept of memory map and memory interface
- ✓ Know the characteristics of Real Time Systems
- ✓ Write programs to interface memory, I/Os with processor
- ✓ Study the interrupt performance

LIST OF EXPERIMENTS

1. Study of ARM evaluation system
2. Interfacing ADC and DAC.
3. Interfacing LED and PWM.
4. Interfacing real time clock and serial port.
5. Interfacing keyboard and LCD.
6. Interfacing EPROM and interrupt.
7. Mailbox.
8. Interrupt performance characteristics of ARM and FPGA.
9. Flashing of LEDS.
10. Interfacing stepper motor and temperature sensor.
11. Implementing zigbee protocol with ARM.

OUTCOMES: At the end of the course, the student should be able to:

- ✓ Write programs in ARM for a specific Application
- ✓ Interface memory and Write programs related to memory operations
- ✓ Interface A/D and D/A convertors with ARM system
- ✓ Analyze the performance of interrupt
- ✓ Write programs for interfacing keyboard, display, motor and sensor.
- ✓ Formulate a mini project using embedded system

TABLE OF CONTENT

S.No	PARTICULARS	PAGE No
1	Basics for Embedded Systems for Software Development	4
2	How to develop embedded software for LPC 1768 using Keil IDE	9
3	How to download and Run program using Flash Magic into LPC1768(Target)	11
4	Hardware Description of ADT V1.1(Development Board)	12
5	ARM CORTEX M3 – LPC1768	13
6	Pin Connections and Configurations of LPC1768 with I/O Peripherals in ADTV1.1 (Development Board)	44
7	RTOS on LPC1768 (Implementing Mail Box)	47

LAB EXPERIMENTS

S.No	PARTICULARS	PAGE No
1	Interfacing LEDS with LPC1768	55
2	Interfacing Buzzer with LPC1768	57
3	Interfacing Switch(Digital Input and output) with LPC1768	58
4	Interfacing ADC with LPC1768	60
5	Interfacing DAC with LPC1768	62
6	Interfacing LED and PWM (RGB) with LPC1768	64
7	Interfacing Real Time Clock on Serial UART	66
8	Interfacing I2C based EEPROM with LPC1768	70
9	Interfacing LCD with LPC1768	72
10	External Interrupt of LPC1768	74
11	Interfacing Stepper Motor with LPC1768	76
12	Interfacing LM35 with LPC1768	79
13	Interfacing ZIGBEE with LPC1768	81
14	Interfacing Matrix Keyboard with LPC1768	83
ADDITIONAL EXPERIMENTS		
1	Interfacing Bluetooth module	84
2	Interfacing GSM module	86
3	Interfacing GPS module	89
ARM7 LPC2148 DEVELOPMENT BOARD		91
PYTHON PROGRAMMING PRACTICE		96
VIVA VOCE QUESTIONS		100
THE FLOWCHART SYMBOLS AND THEIR USAGE		105

BASICS FOR EMBEDDED SYSTEMS SOFTWARE DEVELOPEMENT

DECIMAL	BINARY	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Example								
Decimal	$(139)_{10}$							
Hexadecimal	$(8B)_{16}$							
Binary	1	0	0	0	1	0	1	1
	Seventh bit (MSB)	Sixth bit	Fifth bit	Fourth bit	Third bit	Second bit	First bit	Zeroth bit (LSB)
1 BYTE(8 Bits)								

Basic C Operator

Arithmetic Operators: The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then −

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators: The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators: Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.

!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators: Bitwise operators work on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100 B = 0000 1101	A&B = 0000 1100 A|B = 0011 1101 A^B = 0011 0001 ~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, i.e,. 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators: The following table lists the assignment operators supported by the C language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary: Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Operators Precedence in C: Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

How to develop embedded software for LPC 1768 using Keil IDE

About "Project":

A project is a file in which **Keil uVision5** stores all information related to an application. E.g. it stores the name of 'C' and/or Assembler source file, memory size to be used and other options for compiler, assembler and linker.

Opening a project:

To open an existing project file, select **Project / Open Project** from the menu.

Creating a new project:

To create a new project, select **Project / New uVision Project** from the menu.

Closing project settings:

To close the project, select **Project / Close Project** from the menu.

	Procedure
Step 1	Start the **Keil uVision5** program (i.e. the Integrated Development Environment) from start\Programs**Keil uVision5**.
Step 2	From Project menu, select **Close project** (if any project is open).
Step 3	From Project menu, select **New Project**. The Open dialog window will be displayed.Select the desired path where you wish to create this new project. (For example, C:\SPJ). **CAUTION**: The path and filename must not contain space or other special characters such as tab, comma, semicolon etc. In the "File name" field, type the name of the project, without any extension. For example, you may type "**PROG1**". Then click on the "**Open**" button.
Step 4	The action in the previous step will display the "**Select device for Target 'Target 1'** " dialog window. Here you have to select the micro-controller you are going to use. The target microcontroller (must be a member of ARM family) is known, You have to load **"Legacy Device Database (RTE)"** and then select the device you may select the appropriate Manufacturer from the list; and then select the appropriate micro-controller from the device list. If the target microcontroller is not known or if you cannot find it in the list, then you may simply search " **LPC1768**"or select "**NXP**" as the manufacturer and "**LPC1768**" as the micro-controller.
Step 5	Then a dialog box will appear asking for adding Startup.s file in to the project. Click on the **"Yes"** to it.

Step 6	Click on "**Source Group 1**" to display that part of the dialog window. This window will indicate that IDE has automatically added 1 files in this new project: **STARTUP.S** The STARTUP.s file is automatically created by the IDE and is required for all C projects. Now add the file select **"File / New"** then a dialog box will appear, give name to your file with extension (E.g. PROG1.c) Then add files in this project, then right click on the "**Source Group 1**" Folder,click on **"Add existing Files to Group 'Source Group1' "** select the desired filename and then click on "**Add**" button. Now the Project Settings dialog will indicate that selected file has been added into the project. When all necessary files have been added to the project, click "**OK**" button to create this newproject.
Step 7	Now create the "**Target**" for the completion of the project, right click on the **"Target 1"** folder in the project explorer window. then click on "**options for Target 'Target 1'** " ,Then a dialog appears having 10 parts first is **Device** and last **Utilities**. Now select **Output** and Checked the options **"Create Hex File".** then go to "**Linker**" and check the option "**Use memory layout from target dialog.**"
Step 8	From the **Target** menu, select **Build**. This will invoke the Compiler to compile the file PROG1.C;and further (assuming no errors) invoke the linker to create the **.HEX** file. If there are any errors or warnings during the process of compiling, assembling or linking, then those will be displayed in the output window (below the editor window). If there are errors, then you may correct those by making appropriate changes to the program; select Save from File menu to save the changes and then again select Build from Compile menu. Repeat this until there are no errors.
Step 9	You may inspect contents of the folder where your project files reside. When there are no errors and build has completed successfully and then you will see a file name with same name as the project name and extension .HEX (in above example, **PROG1.HEX**). This is the file that you will need to use to program your micro-controller.

How to download and Run program using Flash Magic into LPC1768

- ✓ After installation of Flash Magic, open it.
- ✓ In Flash Magic go to Options -> Advanced Options-> Communications. Check High Speed Communications and keep Maximum Baud Rate as 19200. Click on OK
- ✓ Again in Flash Magic go to Options -> Advanced Options-> Hardware Config. "Use DTR and RTS to control RST and P0.14" option should be checked. Click on OK.
- ✓ (After doing above mentioned settings, Flash Magic stores it means for the next time just verify if these setting are proper or not. If they are proper then you can directly follow below mentioned procedure)

Step 1 Connect the J1/UART0 connector of ADT V1.1 board to COM1 or COM2 of a PC, using the serial communication cable (supplied with the board).

Step 2 Keep S2 switch in ON position. (You can keep S2 switch continuously ON) Switch ON power to the ADT V1.1.

Step 3 Do proper settings in Flash Magic (COM Port: COM1 (if other choose it), Baud Rate: 19200,Device: LPC1768, Interface: None (ISP), Enable "Erase blocks used by Hex File", Browse the file which you want to download) and click on Start button.

Step 4 Flash Magic will download the program. Wait till Finished comes.

Step 5 After downloading Flash Magic automatically resets the ADT V1.1 board and program executes. You can see output according to the program.

Step 6 If again you want to **"Reset"** the board then press **"RST"** switch on SM-2148 board. You can see output according to the program. **Note**: Flash Magic can be used to download the program into other Philips Micro-controllers also. See the list in Flash Magic itself.

HARDWARE DESCRIPTION OF ADT V1.1

Unpacking:
You will find following items in the package:
• ADT V1.1 board
• Serial communication cable (Straight 9-Pin)
• Power adapter with cable
• SPJ_TOOLS' CD-ROM

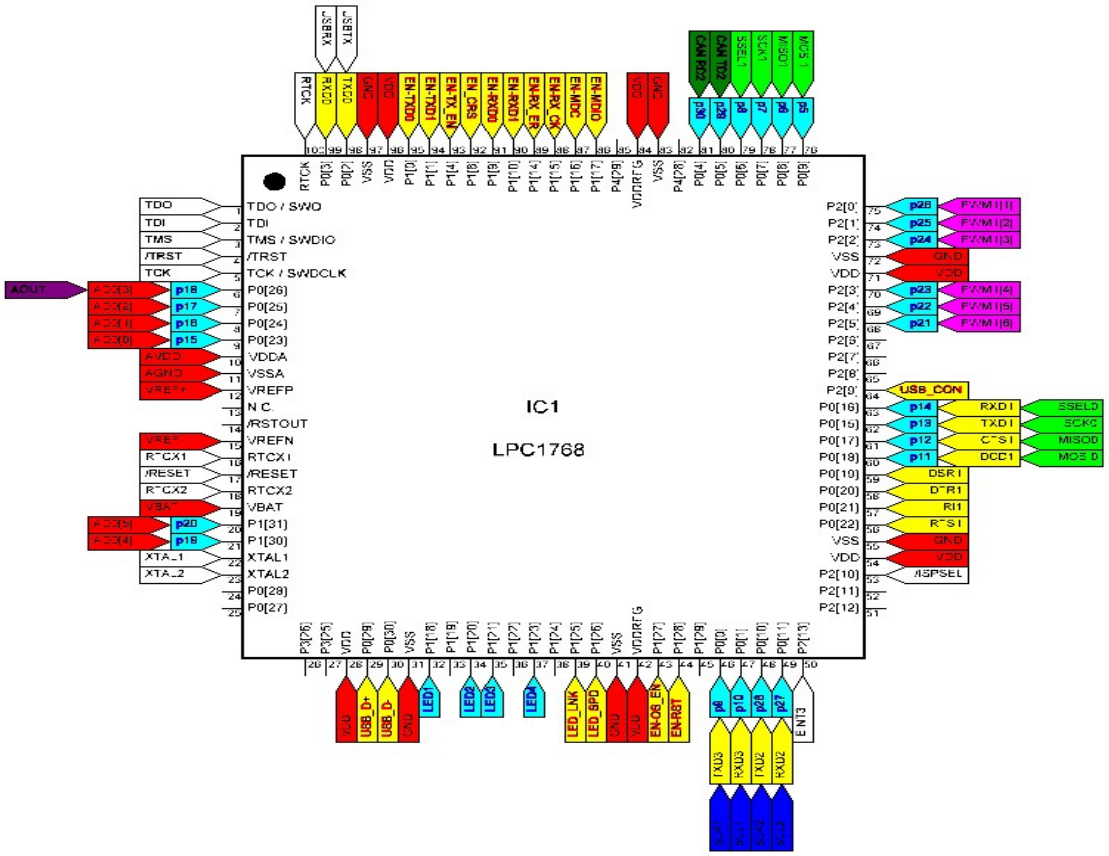
Power Supply Requirements:
The power adapter works with 230Volts AC. It produces approximately 9 Volts DC, and the ADT V1.1 uses on-board regulators to provide 5 Volts and 3.3 Volts DC to all components on the board.

Connecting the system:
The serial communication cable supplied with the board should be used to connect the board to a PC running Windows95/98/NT/ ME/2000/XP/Vista Operating System. Connect one end of the serial cable to UART0 of ADT V1.1 board and other end to PCs serial port.

Powering ON:
After connecting the serial communication cable as described above, you may insert the power adapter output jack into the on-board power socket. Plug the power adapter into 230VAC mains outlet and turn it on. Now press on-board power switch, power-on indication Green LED will turn on.

CAUTION: Please do not connect or disconnect the serial communication cable while the board is powered ON. Doing so can damage the serial port of the ADT V1.1 board and/or PC.

ARM CORTEX M3-LPC1768



PIN DESCRIPTION

Pin Number	Symbol	Pinsel reg	Pinsel ix	I/O	Pin Function	Function Block	Description
1	TDO/SWO			O	TDO	JTAG	Test Data out for JTAG interface.
					SWO	SWDEBUG	Serial wire trace o/p.
2	O			I	TDI	JTAG	Test Data in for JTAG interface.

3	TMS/SWDIO			I	TMS	JTAG	Test Mode Select for JTAG interface.
				I	SWDIO	SWDEBUG	Serial wire debug data input/output.
4	!TRST			I	!TRST	JTAG	Test Reset for JTAG interface.
5	TCK/SWDCLK			I	TCK	JTAG	Test Clock for JTAG interface.
				I	SWDCLK	SWDEBUG	Serial wire clock.
6	P0.26/AD0.3/AOUT/RXD3	pinsel1 21:20 x	0	I/O	P0.26	GPIO 0	General purpose digital input/output pin. When configured as an ADC input or DAC output, the digital section of the pad is disabled.
			1	I	AD0.3	ADC	A/D converter 0, input 3.
			2	O	AOUT	DAC	D/A converter output.
			3	I	RXD3	UART 3	Receiver input for UART3.
7	P0.25/AD0.2/I2SRX_SDA/TXD3	pinsel1 19:18 x	0	I/O	P0.25	GPIO 0	General purpose digital input/output pin. When configured as an ADC input, digital section of the pad is disabled.
			1	I	AD0.2	ADC	A/D converter 0,

							input 2.
			2	I/O	I2SRX_SDA	I2S	Receive data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the I2S bus specification.
			3	O	TXD3	UART 3	Transmitter output for UART3.
8	P0.24/AD0.1/I2SRX_WS/CAP3.1	pinsel1 17:16 x	0	I/O	P0.24	GPIO 0	General purpose digital input/output pin. When configured as an ADC input, digital section of the pad is disabled.
			1	I	AD0.1	ADC	A/D converter 0, input 1.
			2	I/O	I2SRX_WS	I2S	Receive Word Select. It is driven by the master and received by the slave. Corresponds to the signal WS in the I2S bus specification.
			3	I	CAP3.1	Timer 3	Capture input for Timer 3, channel 1.
9	P0.23/AD0.0/I2SRX_CLK/CAP3.0	pinsel1 15:14 x	0	I/O	P0.23	GPIO 0	General purpose digital input/output pin. When

							configured as an ADC input, digital section of the pad is disabled.
			1	I	AD0.0	ADC	A/D converter 0, input 0.
			2	I/O	I2SRX_CLK	I2S	Receive Clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the I2S bus specification.
			3	I	CAP3.0	Timer 3	Capture input for Timer 3, channel 0.
10	VDDA			P	VDDA	ADC DAC	analog 3.3 V pad supply voltage: This can be connected to the same supply as VDD(3V3) but should be isolated to minimize noise and error. This voltage is used to power the ADC and DAC. Note: this pin should be tied to 3.3v if the ADC and DAC are not used.
11	VSSA			P	VSSA	ADC DAC	analog ground: 0 V reference. This should be the same voltage as VSS, but

					should be isolated to minimize noise and error.
12	VREFP	I	VREFP	ADC DAC	ADC positive reference voltage: This should be nominally the same voltage as VDDA but should be isolated to minimize noise and error. The voltage level on this pin is used as a reference for ADC and DAC. Note: this pin should be tied to 3.3v if the ADC and DAC are not used.
13	n.c.	-	n.c.		not connected
14	!RSTOUT	O	!RSTOUT	Main	This is a 3.3 V pin. A LOW on this pin indicates that the LPC17xx is in a Reset state.
15	VREFN	I	VREFN	ADC DAC	ADC negative reference voltage: This should be the same voltage as VSS but should be isolated to minimize noise and error. Level on this pin is used as a

						reference for ADC and DAC.
16	RTCX1			I	RTCX1	RTC	Input to the RTC oscillator circuit.
17	!RESET			I	!RESET	Main	External reset input: A LOW on this pin resets the device, causing I/O ports and peripherals to take on their default states, and processor execution to begin at address 0. This is a 5 V tolerant pad with a 20 ns glitch filter, TTL levels and hysteresis.
18	RTCX2			O	RTCX2	RTC	Output from the RTC oscillator circuit.
19	VBAT			I	VBAT	RTC	RTC domain power supply: 3.3 V on this pin supplies the power to the RTC peripheral.
20	P1.31/SCK1/AD 0.5	pinsel3 31:30 x	0	I/ O	P1.31	GPIO 1	General purpose digital input/output pin. When configured as an ADC input, digital section of the pad is disabled.
			2	I/ O	SCK1	SSP 1	Serial Clock for SSP1.

			3	I	AD0.5	ADC	A/D converter 0, input 5.
21	P1.30/VBUS/AD 0.4	pinsel3 29:28 x	0	I/ O	P1.30	GPIO 1	General purpose digital input/output pin. When configured as an ADC input, digital section of the pad is disabled.
			2	I	VBUS	USB-Device	Monitors the presence of USB bus power. Note: This signal must be HIGH for USB reset to occur.
			3	I	AD0.4	ADC	A/D converter 0, input 4.
22	XTAL1			I	XTAL1	Main	Input to the oscillator circuit and internal clock generator circuits.
23	XTAL2			O	XTAL2	Main	Output from the oscillator amplifier.
24	P0.28/SCL0/USB_SCL	pinsel1 25:24 x	0	I/ O	P0.28	GPIO 0	General purpose digital input/output pin. Open-drain 5 V tolerant digital I/O pad, compatible with I2C-bus specifications for 100 kHz standard mode, 400 kHz Fast Mode, and 1

							MHz Fast Mode Plus. This pad requires an external pull-up to provide output functionality. When power is switched off, this pin connected to the I2C-bus is floating and does not disturb the I2C lines. Open-drain configuration applies to all functions on this pin.
			1	I/O	SCL0	I2C 0	I2C0 clock input/output. Open-drain output (for I2C-bus compliance).
			2	I/O	USB_SCL	USB-OTG	USB port I2C serial clock (OTG transceiver).
25	P0.27/SDA0/USB_SDA	pinsel1 23:22 x	0	I/O	P0.27	GPIO 0	General purpose digital input/output pin. Open-drain 5 V tolerant digital I/O pad, compatible with I2C-bus specifications for 100 kHz standard mode, 400 kHz Fast Mode, and 1 MHz Fast Mode Plus. This pad requires an

							external pull-up to provide output functionality. When power is switched off, this pin connected to the I2C-bus is floating and does not disturb the I2C lines. Open-drain configuration applies to all functions on this pin.
			1	I/O	SDA0	I2C 0	I2C0 data input/output. Open-drain output (for I2C-bus compliance).
			2	I/O	USB_SDA	USB-OTG	USB port I2C serial data (OTG transceiver).
26	P3.26/STCLK/MAT0.1/PWM1.3	pinsel7 21:20 x	0	I/O	P3.26	GPIO 3	General purpose digital input/output pin.
			1	I	STCLK	Tick	System tick timer clock input.
			2	O	MAT0.1	Timer 0	Match output for Timer 0, channel 1.
			3	O	PWM1.3	PWM	Pulse Width Modulator 1, channel 3 output.
27	P3.25/MAT0.0/PWM1.2	pinsel7 19:18 x	0	I/O	P3.25	GPIO 3	General purpose digital input/output pin.

			2	O	MAT0.0	Timer 0	Match output for Timer 0, channel 0.
			3	O	PWM1.2	PWM	Pulse Width Modulator 1, channel 2 output.
28	VDD(3V3)			I	VDD(3V3)	Main	3.3 V supply voltage: This is the power supply voltage for I/O other than pins in the Vbat domain.
29	P0.29/USB_D+	pinsel1 27:26 x	0	I/O	P0.29	GPIO 0	General purpose digital input/output pin. Pad provides digital I/O and USB functions. It is designed in accordance with the USB specification, revision 2.0 (Full-speed and Low-speed mode only).
			1	I/O	USB_D+	USB-Device USB-Host USB-OTG	USB bidirectional D+ line.
30	P0.30/USB_D-	pinsel1 29:28 x	0	I/O	P0.30	GPIO 0	General purpose digital input/output pin. Pad provides digital I/O and USB functions. It is designed in accordance with

							the USB specification, revision 2.0 (Full-speed and Low-speed mode only).
			1	I/O	USB_D-	USB-Device USB-Host USB-OTG	USB bidirectional D-line.
31	VSS			I	VSS	Main	ground: 0 V reference.
32	P1.18/USB_UP_LED/PWM1.1/CAP1.0	pinsel3 5:4 x	0	I/O	P1.18	GPIO 1	General purpose digital input/output pin.
			1	O	USB_UP_LED	USB-Device USB-Host USB-OTG	USB GoodLink LED indicator. It is LOW when device is configured (non-control endpoints enabled). It is HIGH when the device is not configured or during global suspend.
			2	O	PWM1.1	PWM	Pulse Width Modulator 1, channel 1 output.
			3	I	CAP1.0	Timer 1	Capture input for Timer 1, channel 0.
33	P1.19/MCOA0/!USB_PPWR/CAP1.1	pinsel3 7:6 x	0	I/O	P1.19	GPIO 1	General purpose digital input/output pin.

			1	O	MCOA0	Motor-PWM	Motor control PWM channel 0, output A.
			2	O	!USB_PPWR	USB-Host	Port Power enable signal for USB port.
			3	I	CAP1.1	Timer 1	Capture input for Timer 1, channel 1.
34	P1.20/MCI0/PWM1.2/SCK0	pinsel3 9:8 x	0	I/O	P1.20	GPIO 1	General purpose digital input/output pin.
			1	I	MCI0	Motor-PWM	Motor control PWM channel 0 input. Also Quadrature Encoder Interface PHA input.
			2	O	PWM1.2	PWM	Pulse Width Modulator 1, channel 2 output.
			3	I/O	SCK0	SSP 0	Serial clock for SSP0.
35	P1.21/!MCABORT/PWM1.3/SSEL0	pinsel3 11:10 x	0	I/O	P1.21	GPIO 1	General purpose digital input/output pin.
			1	I	!MCABORT	Motor-PWM	Motor control PWM, active low fast abort.
			2	O	PWM1.3	PWM	Pulse Width Modulator 1, channel 3 output.
			3	I/O	SSEL0	SSP 0	Slave Select for SSP0.

36	P1.22/MCOB0/USB_PWRD/MAT1.0	pinsel3 13:12 x	0	I/O	P1.22	GPIO 1	General purpose digital input/output pin.
			1	O	MCOB0	Motor-PWM	Motor control PWM channel 0, output B.
			2	I	USB_PWRD	USB-Host	Power Status for USB port (host power switch).
			3	O	MAT1.0	Timer 1	Match output for Timer 1, channel 0.
37	P1.23/MCI1/PWM1.4/MISO0	pinsel3 15:14 x	0	I/O	P1.23	GPIO 1	General purpose digital input/output pin.
			1	I	MCI1	Motor-PWM	Motor control PWM channel 1 input. Also Quadrature Encoder Interface PHB input.
			2	O	PWM1.4	PWM	Pulse Width Modulator 1, channel 4 output.
			3	I/O	MISO0	SSP 0	Master In Slave Out for SSP0.
38	P1.24/MCI2/PWM1.5/MOSI0	pinsel3 17:16 x	0	I/O	P1.24	GPIO 1	General purpose digital input/output pin.
			1	I	MCI2	Motor-PWM	Motor control PWM channel 2 input. Also Quadrature Encoder Interface INDEX input.

			2	O	PWM1.5	PWM	Pulse Width Modulator 1, channel 5 output.
			3	I/O	MOSI0	SSP 0	Master Out Slave In for SSP0.
39	P1.25/MCOA1/ MAT1.1	pinsel3 19:18 x	0	I/O	P1.25	GPIO 1	General purpose digital input/output pin.
			1	O	MCOA1	Motor-PWM	Motor control PWM channel 1, output A.
			3	O	MAT1.1	Timer 1	Match output for Timer 1, channel 1.
40	P1.26/MCOB1/PWM1.6/CAP0.0	pinsel3 21:20 x	0	I/O	P1.26	GPIO 1	General purpose digital input/output pin.
			1	O	MCOB1	Motor-PWM	Motor control PWM channel 1, output B.
			2	O	PWM1.6	PWM	Pulse Width Modulator 1, channel 6 output.
			3	I	CAP0.0	Timer 0	Capture input for Timer 0, channel 0.
41	VSS			I	VSS	Main	ground: 0 V reference.
42	VDD(REG)(3V3)			I	VDD(REG)(3V3)	Main	3.3 V voltage regulator supply voltage: This is the supply voltage for the on-chip voltage regulator only.

43	P1.27/CLKOUT/ !USB_OVRCR/ CAP0.1	pinsel3 23:22 x	0	I/ O	P1.27	GPIO 1	General purpose digital input/output pin.
			1	O	CLKOUT	Main	Clock output pin.
			2	I	!USB_OV RCR	USB-Host	USB port Over-Current status.
			3	I	CAP0.1	Timer 0	Capture input for Timer 0, channel 1.
44	P1.28/MCOA2/P CAP1.0/MAT0.0	pinsel3 25:24 x	0	I/ O	P1.28	GPIO 1	General purpose digital input/output pin.
			1	O	MCOA2	Motor-PWM	Motor control PWM channel 2, output A.
			2	I	PCAP1.0	PWM	Capture input for PWM1, channel 0.
			3	O	MAT0.0	Timer 0	Match output for Timer 0, channel 0.
45	P1.29/MCOB2/P CAP1.1/MAT0.1	pinsel3 27:26 x	0	I/ O	P1.29	GPIO 1	General purpose digital input/output pin.
			1	O	MCOB2	Motor-PWM	Motor control PWM channel 2, output B.
			2	I	PCAP1.1	PWM	Capture input for PWM1, channel 1.
			3	O	MAT0.1	Timer 0	Match output for Timer 0, channel 1.

46	P0.0/RD1/TXD3 /SDA1	pinsel0 1:0 x	0	I/ O	P0.0	GPIO 0	General purpose digital input/output pin.
			1	I	RD1	CAN 1	CAN1 receiver input.
			2	O	TXD3	UART 3	Transmitter output for UART3.
			3	I/ O	SDA1	I2C 1	I2C1 data input/output (this pin is not fully compliant with the I2C-bus specification, see Section 19–4 for details).
47	P0.1/TD1/RXD3 /SCL1	pinsel0 3:2 x	0	I/ O	P0.1	GPIO 0	General purpose digital input/output pin.
			1	O	TD1	CAN 1	CAN1 transmitter output.
			2	I	RXD3	UART 3	Receiver input for UART3.
			3	I/ O	SCL1	I2C 1	I2C1 clock input/output (this pin is not fully compliant with the I2C-bus specification, see Section 19–4 for details).
48	P0.10/TXD2/SD A2/MAT3.0	pinsel0 21:20 x	0	I/ O	P0.10	GPIO 0	General purpose digital input/output pin.
			1	O	TXD2	UART 2	Transmitter output for UART2.

			2	I/O	SDA2	I2C 2	I2C2 data input/output (this is not an open-drain pin).
			3	O	MAT3.0	Timer 3	Match output for Timer 3, channel 0.
49	P0.11/RXD2/SCL2/MAT3.1	pinsel0 23:22 x	0	I/O	P0.11	GPIO 0	General purpose digital input/output pin.
			1	I	RXD2	UART 2	Receiver input for UART2.
			2	I/O	SCL2	I2C 2	I2C2 clock input/output (this is not an open-drain pin).
			3	O	MAT3.1	Timer 3	Match output for Timer 3, channel 1.
50	P2.13/!EINT3/I2STX_SDA	pinsel4 27:26 x	0	I/O	P2.13	GPIO 2	General purpose digital input/output pin. 5 V tolerant pad with 5 ns glitch filter providing digital I/O functions with TTL levels and hysteresis.
			1	I	!EINT3	Main	External interrupt 3 input.
			3	I/O	I2STX_SDA	I2S	Transmit data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the I2S bus

							specification.
51	P2.12/!EINT2/I2STX_WS	pinsel4 25:24 x	0	I/O	P2.12	GPIO 2	General purpose digital input/output pin. 5 V tolerant pad with 5 ns glitch filter providing digital I/O functions with TTL levels and hysteresis.
			1	I	!EINT2	Main	External interrupt 2 input.
			3	I/O	I2STX_WS	I2S	Transmit Word Select. It is driven by the master and received by the slave. Corresponds to the signal WS in the I2S bus specification.
52	P2.11/!EINT1/I2STX_CLK	pinsel4 23:22 x	0	I/O	P2.11	GPIO 2	General purpose digital input/output pin. 5 V tolerant pad with 5 ns glitch filter providing digital I/O functions with TTL levels and hysteresis.
			1	I	!EINT1	Main	External interrupt 1 input.
			3	I/O	I2STX_CLK	I2S	Transmit Clock. It is driven by the master and received by the slave. Corresponds to

							the signal SCK in the I2S bus specification.
53	P2.10/!EINT0/NMI	pinsel4 21:20 x	0	I/O	P2.10	GPIO 2	General purpose digital input/output pin. 5 V tolerant pad with 5 ns glitch filter providing digital I/O functions with TTL levels and hysteresis. Note: A LOW on this pin while !RESET is LOW forces the on-chip bootloader to take over control of the part after a reset and go into ISP mode. See Section 32–1.
			1	I	!EINT0	Main	External interrupt 0 input.
			2	I	NMI	Main	Non-maskable interrupt input.
54	VDD(3V3)			I	VDD(3V3)	Main	3.3 V supply voltage: This is the power supply voltage for I/O other than pins in the Vbat domain.
55	VSS			I	VSS	Main	ground: 0 V reference.
56	P0.22/RTS1/TD1	pinsel1 13:12 x	0	I/O	P0.22	GPIO 0	General purpose digital input/output pin.

			1	O	RTS1	UART 1	Request to Send output for UART1. Can also be configured to be an RS-485/EIA-485 output enable signal.
			3	O	TD1	CAN 1	CAN1 transmitter output.
57	P0.21/RI1/RD1	pinsel1 11:10 x	0	I/O	P0.21	GPIO 0	General purpose digital input/output pin.
			1	I	RI1	UART 1	Ring Indicator input for UART1.
			3	I	RD1	CAN 1	CAN1 receiver input.
58	P0.20/DTR1/SCL1	pinsel1 9:8 x	0	I/O	P0.20	GPIO 0	General purpose digital input/output pin.
			1	O	DTR1	UART 1	Data Terminal Ready output for UART1. Can also be configured to be an RS-485/EIA-485 output enable signal.
			3	I/O	SCL1	I2C 1	I2C1 clock input/output (this pin is not fully compliant with the I2C-bus specification, see Section 19–4 for details).
59	P0.19/DSR1/SD	pinsel1 7:6	0	I/	P0.19	GPIO 0	General purpose digital

	A1	x		O			input/output pin.
			1	I	DSR1	UART 1	Data Set Ready input for UART1.
			3	I/O	SDA1	I2C 1	I2C1 data input/output (this pin is not fully compliant with the I2C-bus specification, see Section 19–4 for details).
60	P0.18/DCD1/MOSI0/MOSI	pinsel1 5:4 x	0	I/O	P0.18	GPIO 0	General purpose digital input/output pin.
			1	I	DCD1	UART 1	Data Carrier Detect input for UART1.
			2	I/O	MOSI0	SSP 0	Master Out Slave In for SSP0.
			3	I/O	MOSI	SPI	Master Out Slave In for SPI.
61	P0.17/CTS1/MISO0/MISO	pinsel1 3:2 x	0	I/O	P0.17	GPIO 0	General purpose digital input/output pin.
			1	I	CTS1	UART 1	Clear to Send input for UART1.
			2	I/O	MISO0	SSP 0	Master In Slave Out for SSP0.
			3	I/O	MISO	SPI	Master In Slave Out for SPI.
62	P0.15/TXD1/SCK0/SCK	pinsel0 31:30 x	0	I/O	P0.15	GPIO 0	General purpose digital input/output pin.

			1	O	TXD1	UART 1	Transmitter output for UART1.
			2	I/O	SCK0	SSP 0	Serial clock for SSP0.
			3	I/O	SCK	SPI	Serial clock for SPI.
63	P0.16/RXD1/SSEL0/SSEL	pinsel11:0 x	0	I/O	P0.16	GPIO 0	General purpose digital input/output pin.
			1	I	RXD1	UART 1	Receiver input for UART1.
			2	I/O	SSEL0	SSP 0	Slave Select for SSP0.
			3	I	SSEL	SPI	Slave Select for SPI.
64	P2.9/USB_CONNECT/RXD2/ENET_MDIO	pinsel419:18 x	0	I/O	P2.9	GPIO 2	General purpose digital input/output pin.
			1	O	USB_CONNECT	USB-Device	Signal used to switch an external 1.5 kΩ resistor under software control. Used with the SoftConnect USB feature.
			2	I	RXD2	UART 2	Receiver input for UART2.
			3	I/O	ENET_MDIO	Ethernet	Ethernet MIIM data input and output.
65	P2.8/TD2/TXD2/ENET_MDC	pinsel417:16 x	0	I/O	P2.8	GPIO 2	General purpose digital input/output pin.
			1	O	TD2	CAN 2	CAN2 transmitter

							output.
			2	O	TXD2	UART 2	Transmitter output for UART2.
			3	O	ENET_MDC	Ethernet	Ethernet MIIM clock.
66	P2.7/RD2/RTS1	pinsel4 15:14 x	0	I/O	P2.7	GPIO 2	General purpose digital input/output pin.
			1	I	RD2	CAN 2	CAN2 receiver input.
			2	O	RTS1	UART 1	Request to Send output for UART1. Can also be configured to be an RS-485/EIA-485 output enable signal.
67	P2.6/PCAP1.0/RI1/TRACECLK	pinsel4 13:12 x	0	I/O	P2.6	GPIO 2	General purpose digital input/output pin.
			1	I	PCAP1.0	PWM	Capture input for PWM1, channel 0.
			2	I	RI1	UART 1	Ring Indicator input for UART1.
			3	I	TRACECLK	Trace	Trace Clock.
68	P2.5/PWM1.6/DTR1/TRACEDATA0	pinsel4 11:10 x	0	I/O	P2.5	GPIO 2	General purpose digital input/output pin.
			1	O	PWM1.6	PWM	Pulse Width Modulator 1, channel 6 output.

			2	O	DTR1	UART 1	Data Terminal Ready output for UART1. Can also be configured to be an RS-485/EIA-485 output enable signal.
			3	O	TRACEDATA0	Trace	Trace data, bit 0.
69	P2.4/PWM1.5/DSR1/TRACEDATA1	pinsel4 9:8 x	0	I/O	P2.4	GPIO 2	General purpose digital input/output pin.
			1	O	PWM1.5	PWM	Pulse Width Modulator 1, channel 5 output.
			2	I	DSR1	UART 1	Data Set Ready input for UART1.
			3	O	TRACEDATA1	Trace	Trace data, bit 1.
70	P2.3/PWM1.4/DCD1/TRACEDATA2	pinsel4 7:6 x	0	I/O	P2.3	GPIO 2	General purpose digital input/output pin.
			1	O	PWM1.4	PWM	Pulse Width Modulator 1, channel 4 output.
			2	I	DCD1	UART 1	Data Carrier Detect input for UART1.
			3	O	TRACEDATA2	Trace	Trace data, bit 2.
71	VDD(3V3)			I	VDD(3V3)	Main	3.3 V supply voltage: This is the power supply voltage for I/O other than pins

						in the Vbat domain.
72	VSS		I	VSS	Main	ground: 0 V reference.
73	P2.2/PWM1.3/CTS1/TRACEDATA3	pinsel4 5:4 x	0	I/O	P2.2	GPIO 2	General purpose digital input/output pin.
			1	O	PWM1.3	PWM	Pulse Width Modulator 1, channel 3 output.
			2	I	CTS1	UART 1	Clear to Send input for UART1.
			3	O	TRACEDATA3	Trace	Trace data, bit 3.
74	P2.1/PWM1.2/RXD1	pinsel4 3:2 x	0	I/O	P2.1	GPIO 2	General purpose digital input/output pin.
			1	O	PWM1.2	PWM	Pulse Width Modulator 1, channel 2 output.
			2	I	RXD1	UART 1	Receiver input for UART1.
75	P2.0/PWM1.1/TXD1	pinsel4 1:0 x	0	I/O	P2.0	GPIO 2	General purpose digital input/output pin.
			1	O	PWM1.1	PWM	Pulse Width Modulator 1, channel 1 output.
			2	O	TXD1	UART 1	Transmitter output for UART1.
76	P0.9/I2STX_SDA/MOSI1/MAT2	pinsel0 19:18	0	I/	P0.9	GPIO 0	General purpose digital

	.3	x		O			input/output pin.
			1	I/O	I2STX_SDA	I2S	Transmit data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the I2S bus specification.
			2	I/O	MOSI1	SSP 1	Master Out Slave In for SSP1.
			3	O	MAT2.3	Timer 2	Match output for Timer 2, channel 3.
77	P0.8/I2STX_WS /MISO1/MAT2.2	pinsel0 17:16 x	0	I/O	P0.8	GPIO 0	General purpose digital input/output pin.
			1	I/O	I2STX_WS	I2S	Transmit Word Select. It is driven by the master and received by the slave. Corresponds to the signal WS in the I2S bus specification.
			2	I/O	MISO1	SSP 1	Master In Slave Out for SSP1.
			3	O	MAT2.2	Timer 2	Match output for Timer 2, channel 2.
78	P0.7/I2STX_CLK/SCK1/MAT2.1	pinsel0 15:14 x	0	I/O	P0.7	GPIO 0	General purpose digital input/output pin.
			1	I/O	I2STX_CLK	I2S	Transmit Clock. It is driven by

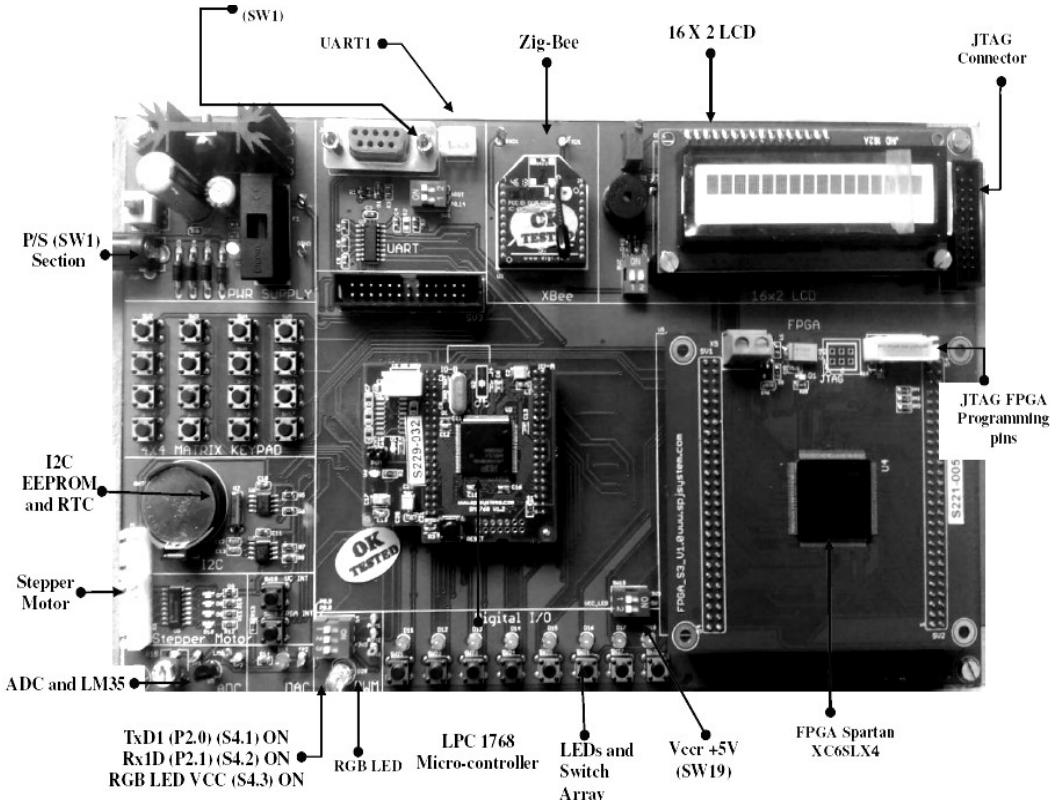
							the master and received by the slave. Corresponds to the signal SCK in the I2S bus specification.
			2	I/O	SCK1	SSP 1	Serial Clock for SSP1.
			3	O	MAT2.1	Timer 2	Match output for Timer 2, channel 1.
79	P0.6/I2SRX_SDA/SSEL1/MAT2.0	pinsel0 13:12 x	0	I/O	P0.6	GPIO 0	General purpose digital input/output pin.
			1	I/O	I2SRX_SDA	I2S	Receive data. It is driven by the transmitter and read by the receiver. Corresponds to the signal SD in the I2S bus specification.
			2	I/O	SSEL1	SSP 1	Slave Select for SSP1.
			3	O	MAT2.0	Timer 2	Match output for Timer 2, channel 0.
80	P0.5/I2SRX_WS/TD2/CAP2.1	pinsel0 11:10 x	0	I/O	P0.5	GPIO 0	General purpose digital input/output pin.
			1	I/O	I2SRX_WS	I2S	Receive Word Select. It is driven by the master and received by the slave. Corresponds to the signal WS in

							the I2S bus specification.
			2	O	TD2	CAN 2	CAN2 transmitter output.
			3	I	CAP2.1	Timer 2	Capture input for Timer 2, channel 1.
81	P0.4/I2SRX_CLK/RD2/CAP2.0	pinsel0 9:8 x	0	I/O	P0.4	GPIO 0	General purpose digital input/output pin.
			1	I/O	I2SRX_CLK	I2S	Receive Clock. It is driven by the master and received by the slave. Corresponds to the signal SCK in the I2S bus specification.
			2	I	RD2	CAN 2	CAN2 receiver input.
			3	I	CAP2.0	Timer 2	Capture input for Timer 2, channel 0.
82	P4.28/RX_MCLK/MAT2.0/TXD3	pinsel9 25:24 x	0	I/O	P4.28	GPIO 4	General purpose digital input/output pin.
			1	O	RX_MCLK	I2S	I2S receive master clock.
			2	O	MAT2.0	Timer 2	Match output for Timer 2, channel 0.
			3	O	TXD3	UART 3	Transmitter output for UART3.
83	VSS			I	VSS	Main	ground: 0 V

							reference.
84	VDD(REG)(3V3)			I	VDD(REG)(3V3)	Main	3.3 V voltage regulator supply voltage: This is the supply voltage for the on-chip voltage regulator only.
85	P4.29/TX_MCLK/MAT2.1/RXD3	pinsel9 27:26 x	0	I/O	P4.29	GPIO 4	General purpose digital input/output pin.
			1	O	TX_MCLK	I2S	I2S transmit master clock.
			2	O	MAT2.1	Timer 2	Match output for Timer 2, channel 1.
			3	I	RXD3	UART 3	Receiver input for UART3.
86	P1.17/ENET_MDIO	pinsel3 3:2 x	0	I/O	P1.17	GPIO 1	General purpose digital input/output pin.
			1	I/O	ENET_MDIO	Ethernet	Ethernet MIIM data input and output.
87	P1.16/ENET_MDC	pinsel3 1:0 x	0	I/O	P1.16	GPIO 1	General purpose digital input/output pin.
			1	O	ENET_MDC	Ethernet	Ethernet MIIM clock.
88	P1.15/ENET_REF_CLK	pinsel2 31:30 x	0	I/O	P1.15	GPIO 1	General purpose digital input/output pin.
			1	I	ENET_REF_CLK	Ethernet	Ethernet reference clock.

89	P1.14/ENET_RX_ER	pinsel2 29:28 x	0	I/O	P1.14	GPIO 1	General purpose digital input/output pin.
			1	I	ENET_RX_ER	Ethernet	Ethernet receive error.
90	P1.10/ENET_RXD1	pinsel2 21:20 x	0	I/O	P1.10	GPIO 1	General purpose digital input/output pin.
			1	I	ENET_RXD1	Ethernet	Ethernet receive data.
91	P1.9/ENET_RXD0	pinsel2 19:18 x	0	I/O	P1.9	GPIO 1	General purpose digital input/output pin.
			1	I	ENET_RXD0	Ethernet	Ethernet receive data.
92	P1.8/ENET_CRS	pinsel2 17:16 x	0	I/O	P1.8	GPIO 1	General purpose digital input/output pin.
			1	I	ENET_CRS	Ethernet	Ethernet carrier sense.
93	P1.4/ENET_TX_EN	pinsel2 9:8 x	0	I/O	P1.4	GPIO 1	General purpose digital input/output pin.
			1	O	ENET_TX_EN	Ethernet	Ethernet transmit data enable.
94	P1.1/ENET_TXD1	pinsel2 3:2 x	0	I/O	P1.1	GPIO 1	General purpose digital input/output pin.
			1	O	ENET_TXD1	Ethernet	Ethernet transmit data 1.
95	P1.0/ENET_TXD0	pinsel2 1:0 x	0	I/O	P1.0	GPIO 1	General purpose digital input/output pin.
			1	O	ENET_TXD0	Ethernet	Ethernet transmit data 0.

96	VDD(3V3)			I	VDD(3V3)	Main	3.3 V supply voltage: This is the power supply voltage for I/O other than pins in the Vbat domain.
97	VSS			I	VSS	Main	ground: 0 V reference.
98	P0.2/TXD0/AD0.7	pinsel0 5:4 x	0	I/O	P0.2	GPIO 0	General purpose digital input/output pin. When configured as an ADC input, digital section of the pad is disabled.
			1	O	TXD0	UART 0	Transmitter output for UART0.
			2	I	AD0.7	ADC	A/D converter 0, input 7.
99	P0.3/RXD0/AD0.6	pinsel0 7:6 x	0	I/O	P0.3	GPIO 0	General purpose digital input/output pin. When configured as an ADC input, digital section of the pad is disabled.
			1	I	RXD0	UART 0	Receiver input for UART0.
			2	I	AD0.6	ADC	A/D converter 0, input 6.
100	RTCK			O	RTCK	JTAG	JTAG interface control signal.



PICTURE OF ADTV1.1

PIN CONNECTIONS AND CONFIGURATIONS OF LPC1768 WITH I/O PERIPHERAL	
DIP Switches Details:	**S1:** Turn ON this switch to connect NRST and P2.10(PGM pin) to LPC1768. **S4.1:** Turn ON this switch to connect UART1 Tx to P2.0 of LPC1768. **S4.2:** Turn ON this switch to connect UART1 Rx to P2.1 of LPC1768. **S4.3:** Turn ON this switch to connect 3.3V to RGB PWM of LPC1768. **S11.1:** Turn ON this switch to connect Buzzer to P0.26 of LPC1768. **S11.1:** Turn ON this switch to connect DAC/TP2 to DACOut/P0.26 of LPC1768. **S19.1:** Turn ON 8 LEDs, It connects 3.3V to VCC of LEDs. **S19.2:** Turn ON 8 LEDs, It connects 5V to VCC of LEDs.

Push-Button Switches Details

When pressed, the switches are grounded. When Idle, they are pulled**:**high

Push-Button	Signal Name
SW1(RST/RESET)	Reset to LPC1768
SW2 to SW17:	Matrix Keypad P1.18 to P1.25
SW18	P2.10/ EINT1
SW20	FPGA Interrupt
SW21	P1.18
SW22	P1.19
SW23	P1.20
SW24	P1.21
SW25	P1.22
SW26	P1.23
SW27	P1.24
SW28	P1.25

LEDs and Buzzer Details

All the LEDs are connected by common anode method. That means the positive leg of each LED is connected to Vcc and negative leg to the port pins of the microcontroller. A logic 0 on the port pin will make LED ON and logic 1 will make it OFF.

LED	Signal Name
D2/ P.ON	+5V Power ON/OFF
D11	P0.4
D12	P0.5
D13	P0.6
D14	P0.7
D15	P0.8
D16	P0.9
D17	P0.10
D18	P0.11
D19	P0.26/DAC
D20	RGB LED
Buzzer	P0.26

UART0

This is a DB9 female connector, used for RS232 serial communication with the PC:
Pin 2 = UART0 RS232 TxD (output of μC)

	Pin 3 = UART0 RS232 RxD (input to µC) Pin 4 = RS232 DTR Pin 5 = Ground Pin 7 = RS232 RTS All other pins of J1/UART0 are unused.
UART1	This is a 3pin Relimate connector, used for RS232 serial communication with the PC: Pin 1 = UART1 RS232 TxD (output of µC) Pin 2 = UART1 RS232 RxD (input to µC)
16x2 LCD	This is a 16 pin, single line connector, designed for connection to standard, text LCD modules. The pin/signal correspondence is designed to be matching with that required by such LCD modules. Pin 1 = GND Pin 2 = +5V Pin 3 = Vlcd Pin 4 = P1.27 (Used as RS of LCD) Pin 5 = GND Pin 6 = P1.26 (Used as EN of LCD) Pin 7 = P0.4 (Used as D0 of LCD) Pin 8 = P0.5 (Used as D1 of LCD) Pin 9 = P0.6 (Used as D2 of LCD) Pin 10 = P0.7 (Used as D3 of LCD) Pin 11 = P0.8 (Used as D4 of LCD) Pin 12 = P0.9 (Used as D5 of LCD) Pin 13 = P0.10 (Used as D6 of LCD) Pin 14 = P0.11 (Used as D7 of LCD) Pin 15 = Back lighting Pin 16 = GND
I2C Connector	This standard 4 pin I2C connector provides support for interfacing of I2C based peripherals to LPC21xx.This connector is mounted on middle-bottom side of the board as shown in figure1. **4 pin straight cable** can be used to connect LPC21xx to the board having I2C based peripheral devices. The pin-out of I2C Connector is given below: **Pin Signal name** 1 ---------- P0.20 (SCL1) 2 ---------- P0.19 (SDA1) 3 ---------- +5V 4 ---------- DGND
Stepper Motor Connector	This standard 6 pin connector provides support for interfacing Stepper motor to LPC1768. The pin-out of Stepper motor Connector is given below: **Pin Signal name** 1 ---------- P0.0

	2 ---------- P0.1 3 ---------- P0.16 4 ---------- P2.3 5 ---------- Vcc (+5V) 6 ---------- GND

RTOS and its Codes

The RTX kernel message objects are simply pointers to a block of memory where the relevant information is stored. There is no restriction regarding the message size or content. The RTX kernel handles only the pointer to this message.

Sending 8-bit, 16-bit, and 32-bit values

Because the RTX kernel passes only the pointer from the sending task to the receiving task, we can use the pointer itself to carry simple information like passing a character from a serial receive interrupt routine. An example can be found in the **serial.c** interrupt driven serial interface module for the **Traffic example**

You must cast the char to a pointer like in the following example:

os_mbx_send (send_mbx, (void *)c, 0xffff);

Sending fixed size messages

To send fixed size messages, you must allocate a block of memory from the dynamic memory pool, store the information in it, and pass its **pointer** to a mailbox. The receiving task receives the pointer and restores the original information from the memory block, and then releases the allocated memory block.

Fixed Memory block memory allocation functions

RTX has very powerful **fixed memory block** memory allocation routines. They are **thread safe** and fully **reentrant**. They can be used with the RTX kernel with no restriction. It is better to use the fixed memory block allocation routines for sending fixed size messages. The memory pool needs to be properly initialized to the size of message objects:

_ **32-bit values**: initialize to 4-byte block size.
_init_box (mpool, sizeof(mpool), 4);

_
_ **any size messages**: initialize to the size of message object.
_init_box (mpool, sizeof(mpool), sizeof(struct message))

Create New RTX Application

This section describes how to create a new application that uses the RTX kernel.
_ ZIn the *Create New Project* window, select a new directory for your project and enter a name for your project.
_ In the *Select Device for Target* window, select your target ARM device and click OK. Allow µVision to copy and add the device startup file to your project. This creates a basic µVision project.
_ Now **setup the project** to use the RTX kernel.

To do this, select **Project —> Options for Target**.
Then select RTX Kernel for the **Operating system** and click OK.

_ Copy the RTX configuration file for your target device from the **\Keil\ARM\RL\RTX\Config** directory and rename it to**RTX_Config.c**:
_ for **ARM7™/ARM9™**devices, copy the configuration file for your specific device.

If the file does not exist for your specific device, then copy the RTX_Conf_LPC21xx.c as a template and modify it to suit your device.
_ for **Cortex-M™**devices, copy **RTX_Conf_CM.c** configuration file.
_ Modify the device startup file for **ARM7™/ARM9™**devices to enable SWI_Handler function (no change required for **Cortex-M™**devices):
_ Comment out the following line from the startup file:
SWI_Handler B SWI_Handler
_ Add the following line to the startup file:
IMPORT SWI_Handler
This change prevents the code from sitting in a loop when a SWI interrupt occurs. The change allows the right function to run when a SWI interrupt occurs.
_ Copy the **retarget.c** file from **\Keil\ARM\Startup** to your project directory, and add it to your project. The main purpoof this file is to avoid the use of semihosting SWIs.

Thus the file must contain the following:

```
#include <rt_misc.h>
#pragma import(__use_no_semihosting_swi)
void _ttywrch(int ch) {
// Not used (No Output)
}
void _sys_exit(int return_code) {
label: goto label; /* endless loop */
}
```

_ Depending on your application, you might have to retarget more functions. For example if you use the RL-FlashFS library, you can obtain retarget.c from the **\Keil\ARM\RL\FlashFS\SRC**\ directory.

Now the project is setup to use the RTX kernel.

Note
For**MicroLIB** run-time library you do not need a **retarget.c** in your project.
_ Now you must **configure the RTX kernel** for the needs of your application by making the required changes in the**RTX_Config.c** file.

_ **Create the application source files** if they do not already exist. Add these source files to the project. You can do this in the project workspace of µVision by right clicking on the Source Group and selecting **Add Files to Group**.

_ When you write programs for RL-RTX, you can define RTX tasks using the __task keyword. You can use the RTX kernel routines whose prototypes are declared in RTL.h.

1. Build your application using **Project —> Build Target**.
2. If you project builds successfully, you can download it to your hardware or run it using the µVision Simulator. You can alsodebug the application using **Debug —> Start Debug Session**.

_ **Configuring RL-RTX**
The RTX kernel is easy to customize for each application you create. This section describes how you can configure the RTX kernel's features for your applications. It contains:

_ Configuration Options

The RTX kernel must be configured for the embedded applications you create. All configuration settings are found in the **RTX_Config.c** file, which is located in the **\Keil\ARM\RL\RTX\Config** directory. **RTX_Config.c**is configured differently for the different ARM devices. Configuration options in **RTX_Config.c** allow you to:

_ Specify the number of concurrent running task.
_ Specify the number of tasks with user-provided stack
_ Specify the stack size for each task
_ Enable or disable the stack checking
_ Enable or disable running tasks in privileged mode
_ Specify the CPU timer number used as the system tick timer
_ Specify the input clock frequency for the selected timer
_ Specify the timer tick interval
_ Enable or disable the round-robin task switching
_ Specify the time slice for the round-robin task switching
_ Define idle task operations
_ Specify the number of user timers
_ Specify code for the user timer callback function
_ Specify the FIFO Queue size
_ Specify code for the runtime error function

There is no default configuration in the RL-RTX library. Hence, you must add the **RTX_Config.c**configuration file to each project you create.

To customize the RTX kernel's features, you must change the configurable settings in **RTX_Config.c**.

_ Idle Task Customization

When no tasks are ready to run, the RTX kernel executes the **idle task** with the name **os_idle_demon()**.

By default this task is an empty end-less loop that does nothing. It only waits until another task becomes ready to run.

You may change the code of **os_idle_demon()** to put the CPU into a power-saving or idle mode. Most**RTX_Config.c** files define the macro _idle_() that contains the code to put the CPU into a power-saving mode.

Example:

```
/*--------------------------- os_idle_demon ---------------------------------*/
__task void os_idle_demon (void) {
/* The idle demon is a system task. It is running when no other task is */
/* ready to run (idle situation). It must not terminate. Therefore it */
/* should contain at least an endless loop. */
for (;;) { _idle_(); /* enter low-power mode */
}
}
```

Note:

_ On some devices, the IDLE blocks debugging via the JTAG interface. Therefore JTAG debuggers such as ULINK may not work when you are using CPU power-saving modes.
_ For using power-saving modes, some devices may require additional configuration (such as clock configuration settings).

_ Error Function Customization

Some system error conditions can be detected during runtime. If RTX kernel detects a **runtime error**, it calls the **os_error()** runtime error function.

```
void os_error (U32 err_code) {
/* This function is called when a runtime error is detected. */
OS_TID err_task;
switch (err_code) {
case OS_ERR_STK_OVF:
/* Identify the task with stack overflow. */
err_task = isr_tsk_get();
break;
case OS_ERR_FIFO_OVF:
break;
case OS_ERR_MBX_OVF:
break;
}
for (;;);
```

The **error code** is passed to this function as a parameter *err_code*:
Error Code Description

Error Code	Description
OS_ERR_STK_OVF	The stack checking has detected a stack overflow for the currently running task.
OS_ERR_FIFO_OVF	The ISR FIFO Queue buffer overflow is detected.
OS_ERR_MBX_OVF	The mailbox overflow is detected for **isr_mbx_send()** function.

The runtime error function must contain an **infinite loop** to prevent further program execution. You can use an emulator to step over infinite loop and trace into the code introducing a runtime error. For the overflow errors this means you need to increase the size of the object causing an overflow.

_ Create New RTX_Config file (for ARM7/ARM9 library)
RL-ARM provides several versions of the **RTX_Config.c** file for **ARM7™/ARM9™**RTX Kernel library. Each one configures the RTX kernel for a specific ARM device variant that RL-ARM supports. However the ARM family of devices is growing quickly, and it is possible that RL-ARM does not contain the configuration file for the device you use. In this case, you can take the **RTX_Config.c** filefor the NXP device as a template and modify it for your particular ARM device. RTX Kernel library for **Cortex™-M** has only one configuration file **RTX_Conf_CM.c**, which iscommon for all Cortex™-M device variants.
HWResources Required
In order to run the RTX kernel, the following hardware resources are required from an ARM device:

_ Peripheral Timer for generating periodic ticks. It is better to use a peripheral timer with an autoreload function. RTX also supports timers with manual timer (or counter) reload. However, this can generate jitter and inaccuracy in the long run. The RTX kernel needs a **count-up** timer. If the timer used is a count-down timer, you need to convert the timer value.

_ Timer Interrupts to interrupt the execution of a task and to start the system task scheduler.

_ **Forced Interrupts** to force a timer interrupt when **isr_** functions are used. If an **isr_** function is called, the kernel forces the timer interrupt immediately after the interrupt ends. The forced timer interrupt activates the task scheduler. It is possible that a task has become ready. If this task has a higher priority than the currently running task, a task switch must occur.

All hardware dependent definitions are extracted from the code and defined with **configuration macros**.
This makes it possible to customize the configuration without modifying the code.

Note:
The RTX configuration files are located in the **\Keil\ARM\RL\RTX\Config** directory.

_ Alternate Tick Timer Configuration (for Cortex-M library)
RTX Library version for **Cortex™-M** devices uses SysTick timer as RTX tick timer. The SysTick timer is Cortex-M core timer, thus common for all Cortex-M device variants. Some new dual core devices, such as LPC4300 devices, do not implement the SysTick timer in both cores. In this case, an alternate tick timer must be used for the core without SysTick timer.

The following functions provide an interface for an Alternate Tick Timer:
_ os_tick_init()
initializes hardware timer as system tick timer,
_ os_tick_irqack()
acknowledges hardware timer interrupt,
_ OS_Tick_Handler()
handles RTX tick interrupts and updates task scheduler.

Configuration
To configure an alternate peripheral timer as system tick timer, you have to:
1. implement the **os_tick_init()** and **os_tick_irqack()** functions in the RTX_Config.c configuration file.
2.Replace the alternate timer interrupt vector with the **OS_Tick_Handler** in the Interrupt Vector Table in startup file.

Note:
An **RTX_Blinky_RIT** example, located in **\Keil\ARM\Boards\MCB1700** directory, is a demo example, configured for alternate tick timer in Cortex-M3.

_ Low Power RTX Configuration
The Low Power RTX extension allows using power-saving modes efficiently and building RTX applications for power constrained devices such as battery powered devices.
The operation of the OS task scheduler in Low Power mode is different than normal operation. Instead of executing periodic system tick interrupts when all active tasks are suspended, the system enters a powerdown mode. It calculates how long it can stay in power-down mode and disables power for peripherals and the CPU. The wake-up timer must remain powered. The time is responsible to wake-up the system after the power-down period expires.

The following functions provide a Low Power RTX extension:
_ os_suspend()
suspends OS task scheduler,
_ os_resume()
resumes OS task scheduler.

Configuration
The Low Power RTX is controlled from the **idle task**. The peripheral wake-up timer must be initialized before the system enters an endless loop.
The function **os_suspend()** calculates the timeout until the first suspended task becomes ready, and returns the timeout to the user.

```
for (;;) {
sleep = os_suspend();
```
The user sets-up a peripheral timer to *sleep* timeout and starts the timer. The timeout is measured in system ticks.

```
if (sleep) {
/* Setup the wake-up timer ... */
```
When the wake-up timer is set-up and running, the user puts the system in power-down mode. The wake-up timer must run also in power-down mode. All other peripherals and the CPU may power-down to reduce power.

```
/* Power-down the system ... */
SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
__WFE();
```
The wake-up timer, when expired, generates the interrupt and wakes-up the system. Hence, it must run also in power-down mode. The system resumes operation and needs to call the function **os_resume()**.
This function restores the RTX and re-enables the OS task scheduler.
```
/* After Wake-up */
sleep = (tc - LPC_WWDT->TV) / 250;
}
os_resume(sleep);
```

If, for any reason, the system does not wake up immediately after the wake-up interrupt, the actual *sleep* time is checked and adjusted.
RTX_LowPower demo examples configured for low power RTX:
_ Usage of the WFE instruction for entering sleep mode is demonstrated in the RTX_LowPower example located in the directory **\Keil\ARM\Boards\Keil\MCBSTM32F200**.

_ Usage of the WFI instruction for entering sleep mode and early wake-up mechanism is demonstrated in RTX_LowPower example located in the directory **\Keil\ARM\Boards\MCB1000\MCB11U10**.

_ First Example RTX Application
This section demonstrates an example of using the RTX kernel for a simple application. The example is located in the folder **\Keil\ARM\RL\RTX\Examples\RTX_ex1**. The application must perform two activities. The first activity must continuously repeat 50 ms after the second activity completes. The second activity must repeat 20 ms after the first activity completes.

Hence, you can implement these activities as two separate tasks, called task1 and task2:

1. Place the code for the two activities into two separate functions (task1 and task2). Declare the two functions as tasks using the keyword **__task** (defined in RTL.H) which indicates a RTX task.

```
__task void task1 (void) {
```

```
.... place code of task 1 here ....
}
__task void task2 (void) {
.... place code of task 2 here ....
}
```

2. When the system starts up, the RTK kernel must start before running any task. To do this, call the **os_sys_init** function in the C **main** function. Pass the function name of the first task as the parameter to the **os_sys_init** function. This ensures that after the RTX kernel initializes, the task starts executing rather than continuing program execution in the**main** function. In this example, task1 starts first. Hence, task1 must create task2. You can do this using the **os_tsk_create** function.

```
__task void task1 (void) {
                os_tsk_create (task2, 0);
.... place code of task 1 here ....
}
__task void task2 (void) {
.... place code of task 2 here ....
}
void main (void) {
os_sys_init (task1);
}
```

3. Now implement the timing requirements. Since both activities must repeat indefinitely, place the code in an endless loop in each task. After the task1 activity finishes, it must send a signal to task2, and it must wait for task2 to complete. Then it must wait for 50 ms before it can perform the activity again. You can use the **os_dly_wait** function to wait for a number of system intervals.

The RTX kernel starts a system timer by programming one of the on-chip hardware timers of the ARM processors. By default, the system interval is 10 ms and timer 0 is used (this is configurable). You can use the **os_evt_wait_or** function to make task1 wait for completion of task2, and you can use the **os_evt_set**function to send the signal to task2. This examples uses bit 2 (position 3) of the event flags to inform the other task when it completes.
task2 must start 20 ms after task1 completes. You can use the same functions in task2 to wait and send signals to task1. The listing below shows all the statements required to run the example:

```
/* Include type and function declarations for RTX. */
#include <rtl.h>
/* id1, id2 will contain task identifications at run-time. */
OS_TID id1, id2;
/* Forward declaration of tasks. */
__task void task1 (void);
__task void task2 (void);
__task void task1 (void){
/* Obtain own system task identification number. */
id1 = os_tsk_self();
/* Create task2 and obtain its task identification number. */
id2 = os_tsk_create (task2, 0);
for (;;) {
/* ... place code for task1 activity here ... */
/* Signal to task2 that task1 ha
/*Wait for completion of task2 activity. */
```

```
s completed. */
os_evt_set(0x0004, id2);
/* 0xFFFF makes it wait without timeout. */
/* 0x0004 represents bit 2. */
os_evt_wait_or(0x0004, 0xFFFF);
/*Wait for 50 ms before restarting task1 activity. */
os_dly_wait(5);
}
}
__task void task2 (void) {
for (;;) {
/*Wait for completion of task1 activity. */
/* 0xFFFF makes it wait without timeout. */
/* 0x0004 represents bit 2. */
os_evt_wait_or(0x0004, 0xFFFF);
/*Wait for 20 ms before starting task2 activity. */
os_dly_wait(2);
/* ... place code for task2 activity here ... */
/* Signal to task1 that task2 has completed. */
os_evt_set(0x0004, id1);
}
}
void main (void) {
/* Start the RTX kernel, and then create and execute task1. */
os_sys_init(task1);
}
```

4. Finally, to compile the code and link it with the RTX library, you must select the RTX operating system for the project. From the main menu, select **Project —> Options for Target**. Select the **Target** tab. Select RTX Kernel for the **Operating system**. Build the project to generate the absolute file. You can run the object file output from the linker either on your target or on the µVision Simulator

Ex. No. 1 **Interfacing LEDS with LPC1768**

Date:

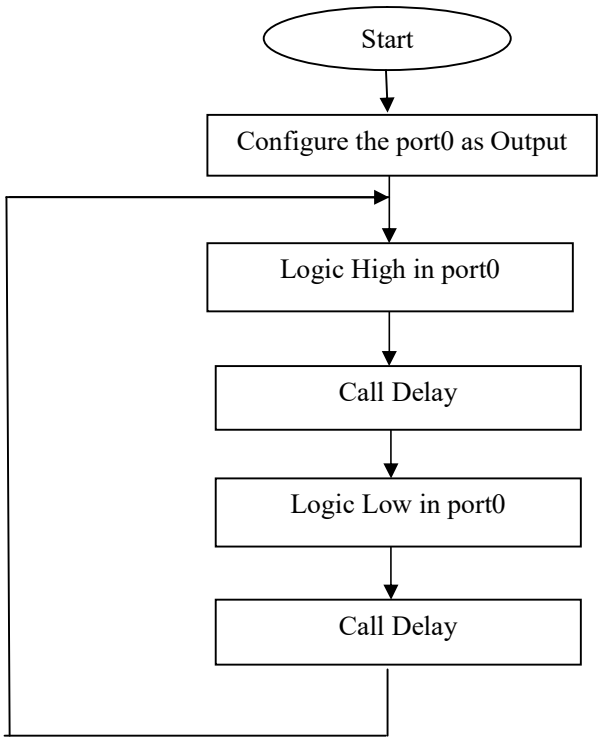
Aim :

To interface the LEDs with LPC1768

Algorithm :

Step 1: Start the program
Step 2: Configure Port0 as GPI0
Step 3: Interfacing Port0 as Output
Step 4: Logic High on Port0
Step 5: Call Delay
Step 6: Logic Low on Port0
Step 7: Call Delay
Step 8: Go to step 4

Flow Chart:



Program:

```
#include "LPC17xx.h"

int main(void)                    // Main Function
{

LPC_PINCON->PINSEL3 = (LPC_PINCON->PINSEL3 &~ 0x000FFFF0) ;
// Configure P0.16 to P0.23 as GPIO

LPC_GPIO1->FIODIR  = (LPC_GPIO1->FIODIR |~ 0x03FC0000) ;
        // Configure P0.16 to P0.23 as Input


LPC_PINCON->PINSEL0 = (LPC_PINCON->PINSEL0 &~ 0x00FFFF00) ;
// Configure P1.16 to P1.23 as GPIO

LPC_GPIO0->FIODIR  = (LPC_GPIO0->FIODIR | 0x00000FF0) ;
// Configure P1.16 to P1.23 as Output

while(1)
{
LPC_GPIO0->FIOCLR = ~((LPC_GPIO1->FIOPIN & 0x03FC0000)>>14);
// Status of Input Switches are copied to LEDs,
                                                                                    //
Respected LEDs pressed switched are On.

LPC_GPIO0->FIOSET = ((LPC_GPIO1->FIOPIN & 0x03FC0000) >>14);
// Switches which are not pressed are made Off.

                }

}
```

Peripheral Interface: Keep SW19.2 switch in ON position. 8 LEDs (D11 to D18) present on ADT are connected to P0.4,P0.5,P0.6,P0.7,P0.8,P0.9,P0.10 and P0.11 respectively by Common Anode method.

Output:
You can see blinking of LEDs.
Note: Keep SW19.2 switch in OFF position to save power, after execution of program.

Result:
Thus interfacing of leds with LPC1768 is executed and verified successfully.

Exercise:

1. Develop an embedded C code to blink LED with different delays (5 secs ,10 secs) using the software delay.
2. Develop an embedded C code to blink LED with different sequence.

Ex. No. 2 **INTERFACING BUZZER (ON-OFF) WITH LPC1768**
Date:

Aim :

To interface Buzzer (ON-OFF) with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Configure PORT0 as output
Step 3: Set logic high on PORT0
Step 4: Call Delay
Step 5: Set logic low on PORT0
Step 6: Call Delay
Step 7: Go to step3

Program:

```
#include "LPC17xx.h"
#include <stdio.h>
volatile unsigned int delay;
void MyDelay(void)
{
        int i;
        for(i=0;i<10;i++)
        for(delay=0;delay<65000;delay++);
}
int main (void)
{

            LPC_GPIO0->FIODIR = 0x04000000 ;                    // Configure P0.26 as output

        while(1)
        {
                LPC_GPIO0->FIOSET = 0x04000000 ; //Buzzer ON
                MyDelay();
                LPC_GPIO0->FIOCLR = 0x04000000 ; //Buzzer OFF
                MyDelay();
        }
}
```

Peripheral Interface: Keep S11.1 switch in ON position.
Buzzer is connected to P0.26.

Output:

Buzzer will turn ON and OFF.
Note: Keep S11.1 switch in OFF position, after execution of program.

Result:

Thus, interfacing of Buzzer with LPC1768 is executed and verified successfully.

Exercise:

1. Develop an embedded C code to enable buzzer with different tones.

Ex. No. 3 **INTERFACING DIGITAL INPUT AND OUTPUT WITH LPC1768**
Date:

Aim :

To interface Digital Input and Output with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Configure P0.16 to P0.23 as GPIO
Step 3: Configure P0.16 to P0.23 as GPIO using LPC_PINCON structure and PINSEL3 element
Step 4: Configure pins P0.16 to P0.23 as input
Step 5: Configure pins P0.16 to P0.23 as GPIO
Step 6: Configure pins P0.16 to P0.23 as output
Step 7: Copy the states of the input switches to LED's
Step 8: Stop

Program:

```
#include "LPC17xx.h"

int main(void)              // Main Function
{

LPC_PINCON->PINSEL3 = (LPC_PINCON->PINSEL3 &~ 0x000FFFF0) ;
// Configure P0.16 to P0.23 as GPIO
LPC_GPIO1->FIODIR  = (LPC_GPIO1->FIODIR |~ 0x03FC0000) ;
// Configure P0.16 to P0.23 as Input
LPC_PINCON->PINSEL0 = (LPC_PINCON->PINSEL0 &~ 0x00FFFF00) ;
// Configure P1.16 to P1.23 as GPIO
LPC_GPIO0->FIODIR  = (LPC_GPIO0->FIODIR | 0x00000FF0) ;
// Configure P1.16 to P1.23 as Output

while(1)
                    {
LPC_GPIO0->FIOCLR = ~((LPC_GPIO1->FIOPIN & 0x03FC0000)>>14);
   // Status of Input Switches are copied to LEDs,


 // Respected LEDs pressed switched are On.
LPC_GPIO0->FIOSET = ((LPC_GPIO1->FIOPIN & 0x03FC0000) >>14);
 // Switches which are not pressed are made Off.
                    }

}
```

Peripheral Interface: Keep S19.2 switch in ON position.
Push-button switches SW21 to SW8 are connected from P1.18 to P1.25.
LEDs D9 to D12 are connected to P0.18 to P0.25 and D11 to D18 are connected from P0.4 to P0.11.

Output:
After pressing any switch from SW21 to SW28, its corresponding LED (D11 to D18) will become ON otherwise it will be OFF.

Result:

Thus interfacing of digital inputs and outputs with LPC1768 is executed and verified successfully.

Exercise:

1. Develop an embedded C code to blink LED when switch is pressed

Ex. No. 4 **INTERFACING ADC WITH LPC 1768**
Date:

Aim :

To interface ADC with LPC 1768.

Algorithm :

Step 1: Start the program
Step 2: Declare the variable to hold ADC converted value
Step 3: Function is declared and defined for ADC initialization
Step 4: Configure PO.23 as ADC input using PINSELI
Step 5: Declare the variable ADC data
Step 6: Select clock for ADC to start of conversion
Step 7: Check end of conversion and read result
Step 8: Return the 12 bit result to main function
Step 9: Call the functions TargetResetInit(), ADC_Init(), InitUART0()
Step 10: Display ADC test
Step 11: Read the ADC value to convert
Step 12: Convert to volts using adc=(5*adc)/1023
Step 13: Display the result on UART
Step 14: Stop

Program:

```
#include "LPC17xx.h"
#include "stdio.h"
#include "UART0.h"
#include "target.h"
#define PCADC 0x00001000

char adcreading[16] ;     // Variable to hold the ADC converted Value
//static inline void __enable_irq()          { asm volatile ("cpsie i"); }
void ADC_Init (void)                // ADC initialation function
{
        LPC_SC->PCONP |= PCADC ;
        LPC_PINCON->PINSEL1 = (LPC_PINCON->PINSEL1 | 0x00004000 );
        // P0.23 is configured as ADC Input
}

unsigned int ADC_GetAdcReading ()
{
        unsigned int adcdata;

LPC_ADC->ADCR =0x01200301 ;
        // Select AD0.0, Select clock for ADC, Start of conversion,

while(!((adcdata = LPC_ADC->ADGDR) & 0x80000000))
      // Check end of conversion (Done bit) and read result
        {

        }

        return((adcdata >> 4) & 0x3ff) ; // Return 12 bit result
```

```
}
int main (void)
{
        unsigned int delay;
        float adc;

                TargetResetInit();
                ADC_Init() ;                    // Initiate ADC Setting
                InitUart0();
                printf("ADC Test");             // Display ADC Test

        while(1)
        {
                adc = ADC_GetAdcReading();  // Read AD0.7 reading
                adc=(5*adc)/1023;// Convertion of ADC value in Volts
                sprintf(adcreading,"\n ADC0 CH1= %.2f V",adc);
          // Convert result into ASCII to display it on LCD
                UART0Puts(adcreading) ;// Display result on UART
                for(delay=0;delay<60000;delay++);
        }
}
```

Peripheral Interface:

To give analog input from Potentiometer R15 present in Analog Input region on ADTV1.1. Analog input range is from 0 to 1023.

Output:

You can see output on Hyper Terminal.
Therefore Open **Hyper Terminal**. Go to **Start**->**All Programs**->**Accessories**->**Communications**->**Hyper Terminal**->Assign the Respective port-> Settings. Do proper settings (**Baud Rate: 19200**, Data
Bits: **8**, Stop Bits: **1**, Echo: **Off**, Parity: **None**, Com Port: Com 1 (if other choose it)). Click on OK. Go to
Port -> Open. If required Reset the ADT board. Now vary the R15 POT and hence see the change in voltage on Hyper Terminal.

Result:

Thus interfacing of ADC with LPC1768 is executed and verified successfully.

Exercise:

1. Develop an embedded C code to interface ADC with LM35(Temperature sensor)

Ex. No. 5 **INTERFACING DAC WITH LPC 1768**
DATE:

Aim :

To interface DAC with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Declare the variable to store output of DAC
Step 3: Call Target Reset Init() function
Step 4: Clock selection for DAC is done
Step 5: Select the DAC output pin
Step 6: Call delay for sometime
Step 7: Aout value is shifted right by 6bits for the loop to count from 0 to 1024
Step 8: Call Delay
Step 9: Again the loop count for 1024 times
Step 10: Aout value is again shifted right by 6times
Step 11: Stop

Program:

```
#include "LPC17xx.h"
#include <stdio.h>
#include "target.h"
unsigned int delay,AoutValue = 0 ;
        // Variable to store the output value of DAC

int main (void)  // Main function
{
        TargetResetInit();
         //LPC_SC->PCLKSEL0 |= 0x00400000;
         //Peripheral clock selection for DAC as PCLK_peripheral = CCLK
LPC_PINCON->PINSEL1 |= 0x00200000 ;
        // to use P0.26 as DAC output pin.

while(1)                    // Infinite for loop for continues operation
        {
        for(AoutValue = 0 ; AoutValue < 1024 ; AoutValue ++) // For loop to count the 1024 times
        {
                for(delay=0;delay<1000;delay++);
                LPC_DAC->DACR = AoutValue << 8 ;                                    //
AoutValue is shifted right by 6 bits
                }
                                                                                    //
End of nested for loop
        for(AoutValue = 1024 ; AoutValue > 0 ; AoutValue --)
        // For loop to count the 1024 times
        {
        for(delay=0;delay<1000;delay++);
        LPC_DAC->DACR = AoutValue << 8 ;
        // AoutValue is shifted right by 6 bits
                }
```

```
        }       // End of outer For loop
}               // End of Main Function
```

Connect one pin of oscilloscope to TP2/DAC and another to GND.

Output:

You can see ramp wave on oscilloscope.

Note: Keep S11.2 switch in OFF position, after execution of program

Result:

Thus interfacing of DAC with LPC1768 is executed and verified successfully.

Exercise:

1. Develop an embedded C code to generate sine wave form using DAC.

Ex. No. 6 **INTERFACING LED AND PWM (RGB) WITH LPC1768**
DATE:

Aim :

To interface LED and PWM(RGB) with LPC1768 .

Algorithm :

Step 1: Start the program
Step 2: Set the GPIO for all pins by PINSEL on PWM2 for RED color in PWM_Init() function
Step 3: Read the Latch enable resistor and enable PWM2 by setting PWM control resister
Step 4: If the color is Green set the pins on PWM4
Step 5: Load the LED and enable PWM4 by setting PWM control register
Step 6: If the color is Blue, set the pins of PWM on PWM6
Step 7: Load LED and enable PWM6 by control register
Step 8: If the color is equal to all set the pins on AWM0
Step 9: Load LED and enable PWM2, PWM4 by setting PWM control register
Step 10: Control is Reset and count frequency = Fp clks
Step 11: For some Cycle one color is enabled and i+ continuously changing for every clock cycle
Step 12: Stop

Program:

```
#include "LPC17xx.h"
#include <stdint.h>
#include "PWM.h"
#include "lpc_types.h"


unsigned int PWM_Init( enum color Color, unsigned int cycle )
{
	if(Color == RED)
	{
		LPC_PINCON->PINSEL4 = 0x00000001 ;
	// Set GPIOs for all PWM pins on PWM2
		LPC_PWM1->LER = LER0_EN | LER1_EN ;
	// Loading the Latch enable resister
		LPC_PWM1->PCR = PWMENA1 ;                                    //
Enabling PWM2 by setting PWM control register
	}
	else if (Color == GREEN)
	{
		LPC_PINCON->PINSEL4 = 0x00000004 ;                           //
set GPIOs for all PWM pins on PWM4
		LPC_PWM1->LER = LER0_EN | LER2_EN ;                          //
Loading the Latch enable resister
		LPC_PWM1->PCR = PWMENA2  ;
	// Enabling PWM4 by setting PWM control register
	}
	else if(Color == BLUE)
	{
		LPC_PINCON->PINSEL4 = 0x0000010 ;                            //
set GPIOs for all PWM pins on PWM6
```

```
                LPC_PWM1->LER = LER0_EN | LER3_EN ;                                    //
Loading the Latch enable resister
                LPC_PWM1->PCR = PWMENA3  ;
        // Enabling PWM6 by setting PWM control register
        }
        else if(Color == ALL)
        {
                LPC_PINCON->PINSEL4 = 0x00000015 ;                                     //
set GPIOs for all PWM pins on PWM0
        LPC_PWM1->LER = LER0_EN | LER2_EN | LER2_EN | LER3_EN;
// Loading the Latch enable resister
        LPC_PWM1->PCR = PWMENA1 | PWMENA2 | PWMENA3;
      // Enabling PWM2,PWM4,PWM6 by setting PWM control register
        }
        LPC_PWM1->TCR = TCR_RESET;      // Counter Reset
        LPC_PWM1->PR = 0x00;                    // count frequency : Fpclk
        LPC_PWM1->MCR = PWMMR0I;
        // interrupt on PWMMR0, reset on PWMMR0, reset

                        // TC if PWM0 matches
        LPC_PWM1->MR0 = cycle;// set PWM cycle
        LPC_PWM1->MR1 = cycle * 5/6;
        LPC_PWM1->MR2 = cycle * 5/6; // cycle * 2/3;
        LPC_PWM1->MR3 = cycle * 5/6 ;// cycle * 1/2;
        LPC_PWM1->TCR = TCR_CNT_EN | TCR_PWM_EN;
                // counter enable, PWM enable

  return (1);
}
```

Peripheral Interface:

Keep S4.3 switch in ON position.

Output:

You can see RED,BLUE,GREEN color blinking on RGB Led.

Note: Keep S4.3 switch in OFF position, after execution of program

Result:

Thus interfacing of LEDS and PWM(RGB) with LPC1768 is executed and verified successfully

Ex. No. 7 INTERFACING REAL TIME CLOCK ON SERIAL UART
DATE:

Aim :

To interface Real Time Clock on Serial UART with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Enable Divisor latch access ,set 8 bit word length, stop bit no parity, disable break transmission
Step 3: Set UPB bus clock divisor
Step 4: Set UART0 divisor latch(LSB)
Step 5: Set UART0 divisor latch(MSB)
Step 6: Clear TxFIFO and enable Rx and TxFIFOS
Step 7: Wait until Trasnsmit holding register is empty
Step 8: Store to transmit holding register.
Step 9: Wait until Trasnsmit holding register is empty
Step 10: Then Store to transmit holding register.
Step 11: Wait until there's a character table read
Step 12: Then read from the receiver buffer register.

Program:

```
#include "LPC17xx.h"
#include "target.h"
#include "UART0.h"
#include <stdio.h>
#include <string.H>
#include "I2C_RTC.h"
#include "Timer.h"
 char MAIN_u8buffer1[20];
char MAIN_u8buffer2[20];
#define PCF8523 1
#ifdef PCF8523
#define OFFSET 3
#endif
#ifdef DS1307
#define OFFSET 0
#endif
#define CONTROL_ADDR0 0
#define CONTROL_ADDR1 1
#define CONTROL_ADDR2 2
#define SECONDS_ADDR      3
#define MINUTES_ADDR      4
#define HOURS_ADDR        5
#define DATE_ADDR         6
#define DAY_ADDR          7
#define MONTH_ADDR        8
#define YEAR_ADDR         9
```

```
#define BYTE0 0x00
#define BYTE1 0x00
#define BYTE2 0x88
unsigned char MAIN_u8buffer[20] ;
volatile unsigned int delay,i;
unsigned char date, mon, year;
unsigned char hh,mm,ss ;
unsigned int idate, imon, iyear ;
unsigned int ihh,imm,iss ;
char rtcdate[16] ;
char rtctime[16] ;

int main(void)
{
        TargetResetInit();
        InitUart0();
        I2C_Init();
        TIMER_Init();
        printf("Clock:\nPress 'S' (to set RTC) Within 5 seconds, any other to discard") ;
        for(delay=0; delay<=800000;delay++)
        {
                        if (UART0_IsReady())
                        break;
        }

        if(UART0_IsReady() && (UART0getchar())=='S')
        {
                printf("\nEnter date and time in dd mm yy hh mm ss format\n") ;
      scanf("%02x%02x%02x%02x%02x%02x",&idate,&imon,&iyear,&ihh,&imm,&iss);
                        date = idate ;
                        mon = imon ;
                        year = iyear ;
                        hh = ihh ;
                        mm = imm ;
                        ss = iss ;
                        printf("You entered:\n%02x-%02x-%02x
%02x:%02x:%02x\n",date,mon,year,hh,mm,ss) ;
                        printf("Press Y to store these values to RTC, any other to discard\n") ;
                        if (UART0getchar() == 'Y')
                        {
                                I2C_WriteToRTC(CONTROL_ADDR0,BYTE0,1) ;
                                I2C_WriteToRTC(CONTROL_ADDR1,BYTE1,1) ;
                                I2C_WriteToRTC(CONTROL_ADDR2,BYTE2,1) ;
                                I2C_WriteToRTC(DATE_ADDR,date,1) ;
                                I2C_WriteToRTC(MONTH_ADDR,mon,1) ;
                                I2C_WriteToRTC(YEAR_ADDR,year,1) ;
```

```
                    I2C_WriteToRTC(HOURS_ADDR,hh,1) ;
                    I2C_WriteToRTC(MINUTES_ADDR,mm,1) ;
                    I2C_WriteToRTC(SECONDS_ADDR,ss,1) ;
                }
                else;
        }

        printf("\n");

        while(1)
        {
            if (!(I2C_ReadFromRTC(OFFSET, MAIN_u8buffer, 7)))
            printf("Main Buffer %s \n",MAIN_u8buffer);
            printf("\r\nClock : ");
          printf("Date : %02x",MAIN_u8buffer[DATE_ADDR-OFFSET]& 0x3F);
          printf("-%02x",MAIN_u8buffer[MONTH_ADDR-OFFSET]& 0x1F);
          printf("-%02x",MAIN_u8buffer[YEAR_ADDR-OFFSET]);
            printf("    Time : %02x",MAIN_u8buffer[HOURS_ADDR-OFFSET]& 0x1F);
          printf(":%02x",MAIN_u8buffer[MINUTES_ADDR-OFFSET]& 0x7F);
            printf(":%02x",MAIN_u8buffer[SECONDS_ADDR-OFFSET]& 0x7F);
        while(1)
                                {
            ss = MAIN_u8buffer[SECONDS_ADDR-OFFSET] & 0x7F;
              if (!I2C_ReadFromRTC(OFFSET, MAIN_u8buffer, 7))
                printf("\nMemory Read error..$");
                if (ss != (MAIN_u8buffer[SECONDS_ADDR-OFFSET] & 0x7F))
                break;
              }
         }
}
```

Peripheral Interface:
No switches are there to turn ON/OFF I2C RTC

Output:
You can see output on Hyper Terminal.
Therefore Open **Hyper Terminal**. Go to **Start->All Programs->Accessories->Communications->Hyper Terminal->**Assign the Respective port-> Settings. Do proper settings (**Baud Rate: 19200**, Data Bits: **8**, Stop Bits: **1**, Echo: **Off**, Parity: **None**, Com Port: Com 1 (if other choose it)). Click on OK. Go to Port -> Open. If required Reset the ADT board.On terminal after reset, press "Shift-S' to set DD MM YY HH MM SS format and press enter. To save the RTC press "Shift-Y",now you can see the RTC getting updated.

Result:

Thus interfacing of real time clock on serial UART with LPC1768 is executed and verified successfully.

Exercise:

1. Develop an embedded C code to design a stop watch.

Ex. No. 8 **INTERFACING I2C BASED EEPROM**
DATE:

Aim :

To interface I2C based EEPROM with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Power on LPCI2C TO I2peripheral
Step 3: Initialize timer, define port pin as SDA&SCL
Step 4: Clear all I2C configure bits and set I2EN
Step 5: Wait for the status if status returns true or false
Step 6: Read data from RTC
Step 7: Ready for device address & ready for data bytes
Step 8: Transmit start address and if received return ACK
Step 9: Clear SI flag
Step 10: Write data to RTC
Step 11: Check the upper limit & write data byte wise
Step 12: After writing clear all except I2REN
Step 13: Clear all except I2N
Step 14: Transmit start address & port for write down
Step 15: Generate stop condition

Program:

```
#include "LPC17xx.h"
#include <Stdio.h>
#include "UART0.h"
#include "I2C.h"
#include "TIMER.h"
#include "target.h"
unsigned char *MAIN_u8buffer1;
unsigned char MAIN_u8buffer2[20] ;
unsigned char MAIN_u8temp = 0;
unsigned char MAIN_u8temp12[2][2]={0,1,2,3};
int main()
{
TargetResetInit();
        LPC_PINCON->PINSEL0 |= 0x00000005 ;
        InitUart0();
        I2C_Init();
        TIMER_Init();
        MAIN_u8temp=MAIN_u8temp12[0][0];
        MAIN_u8buffer1 = (unsigned char*)MAIN_u8temp12;
        while(1)
        {
                if (!I2C_WriteToEEPROM(0,  MAIN_u8buffer1, 10))
                        printf("\nMemory write error..");

                if (!I2C_ReadFromEEPROM(0, MAIN_u8buffer2, 15))
                        printf("\nMemory Read error..");
                printf("\n%u",*MAIN_u8buffer2);
                printf("\n%u",MAIN_u8buffer2[0]);
```

```
                    printf("\n%x",MAIN_u8buffer2[1]);
                    printf("\n%x",MAIN_u8buffer2[2]);
                    printf("\n%x",MAIN_u8buffer2[3]);
                    while(1);

        }
}
```

Peripheral Interface:

No switches are there to turn ON/OFF I2C EEPROM

Output:

You can see output on Hyper Terminal.
Therefore Open **Hyper Terminal**. Go to **Start**->**All Programs**->**Accessories**->**Communications**->**Hyper Terminal**->Assign the Respective port-> Settings. Do proper settings (**Baud Rate: 19200**, Data
Bits: **8**, Stop Bits: **1**, Echo: **Off**, Parity: **None**, Com Port: Com 1 (if other choose it)). Click on OK. Go to Port -> Open. If required Reset the ADT board. It will display 2 option Read and Write. We have to first select Write. Type something max of 20 character length. And reset the board and select Read option.

Result:

Thus interfacing I2C based EEPROM with LPC1768 is executed and verified successfully

Exercise:

1.Develop an embedded C code to transfer image file & store it in EEPROM & retrieve it

Ex. No. 9 **INTERFACING OF LCD**
DATE:

Aim :

To interface LCD with LPC1768

Algorithm :

Step 1: Start the pogram
Step 2: Write delay function for small delay and large delay
Step 3: Send command on the data lines
Step 4: Clear Reset (RS) and enable (EN)
Step 5: Set the enable PIN and call delay
Step 6: Clear the enable (EN) and call delay
Step 7: Send the data on data lines (D0 to D7)
Step 8: After sending the data Reset (RS) is set and call delay
Step 9: Clear the pin EN and call delay
Step 10: Set the pin EN and Call delay
Step 11: Clear the pin EN and call delay
Step 12: Configure the pins PO.15 (D4), PO.17 (D5), PO.22 (D6) and PO.30 (D7) as output
Step 13: Clear all the pins and configure the pins as GPIO and clear all pins
Step 14: Call the function LCD cmd and call delay
Step 15: Delay function is written which has pass pointer to character string
Step 16: Display the character using display data
Step 17: Stop

Program:

```
/*
        This file contains LCD related sub-routines for EDU-ARM7-2148 Trainer Kit.
        LCD connections:
        P1.26   P1.27   P0.11   P0.10   P0.9    P0.8    P0.7    P0.6    P0.5    P0.4
        RS              EN              D7              D6              D5              D4
        D3              D2              D1              D0
        8 bit interface is used.

 */

#include "LPC17xx.h"
#include <stdio.h>
#include "LCD.h"
#include "target.h"
int main ()
{
        volatile int l;
        TargetResetInit();
        LcdInit();
        while(1)
                {
                        DisplayRow (1,"  SPJ EMBEDDED  ");
                        DisplayRow (2,"  Technologies  ");
                }
```

```
}
```

Peripheral Interface:

No switches are there to turn ON/OFF I2C EEPROM

Output:
In this program after pressing any key from SW2 to SW17, its code will be displayed on 16x2 Text LCD.

Result:

Thus interfacing of LCD with LPC1768 is executed and verified successfully

Exercise:

1. Develop an embedded C code to scroll the text from left to right in LCD display.

Ex. No. 10 **INTERFACING EXTERNAL INTERRUPT**
DATE:

Aim :

To interface External Interrupt with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Main function
Step 3: Initialize INIT interrupt pin
Step 4: Level sensitive mode on EINT1
Step 5: Low active or negative edge sensitive
Step 6: Clear EINT0 interrupt flag
Step 7: Set flag
Step 8: Clear EINT1 interrupt flag
Step 9: Check flag and clear flag
Step 10: Stop

Program:

```
#include "LPC17xx.h"
#include "target.h"
//#define EINT0_IRQn (1<<18)

void EINT0_IRQHandler(void)__irq;
unsigned char Eint0Detect = 0;
unsigned int DisplayTime = 0 ;


void InitEint0(void)
{

        LPC_PINCON->PINSEL4 |= 0x00100000 ;     // Initialize EINT1 Interrupt Pin
        LPC_SC->EXTMODE     |= 0x00 ;           // Level sensitive mode on EINT1
        LPC_SC->EXTPOLAR    |= 0x00 ;           // Low-active or Negative edge sensitive
        LPC_SC->EXTINT = 0x02;                  // Clear EINT0 interrupt flag
}

void EINT0_IRQHandler(void)__irq
{
        LPC_GPIO0->FIOCLR |= 0x00000010;
        Eint0Detect = 1 ;                       // Set flag
        LPC_SC->EXTINT = 0x01;                  // Clear EINT1 interrupt flag

}

int main (void)
{
        TargetResetInit();
        __enable_irq();
        LPC_GPIO0->FIODIR |=  0x00000010 ;      // Configure P1.16 to P1.23 as Output
        LPC_GPIO0->FIOSET |= 0x00000010;
```

```
        NVIC_EnableIRQ(EINT0_IRQn);

        InitEint0();                                    // Initialise EINT0

         while(1)
         {
                if(Eint0Detect)                         // Check flag
                {
                        Eint0Detect = 0 ;               // Clear flag
                        LPC_GPIO0->FIOSET = (LPC_GPIO0->FIOSET | 0x00000010);

                }
         }
}
```

Peripheral Interface:

To connect SW19.2 switch to EINT1 of LPC1768, keep it in ON position

Output:

EINT1 is configured as Low-active, Level Sensitive. Therefore if SW18 switch is pressed then D11 LED
will glow for that much of time only.

Note: Keep SW19.2 switch in OFF position, after execution of program.

Result:

Thus interfacing of external interrupt with LPC1768 is executed and verified successfully

Exercise:

1. Develop an embedded C code to produce multiple interrupts.

Ex. No. 11 **INTERFACING STEPPER MOTOR**
DATE:

Aim :

To interface Stepper Motor with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Assign the values for the phase A,B,C,D
Step 3: Configure the pins as GPIO and assign the pin as output in Port0 and set the LED as OFF
Step 4: Configure the pins P0.28 to P0.31 as GPIO and also as output
Step 5: Set the LED OFF in Port1
Step 6: Configure the pins P0.28 to p0.31 as GPIO and as output in Port2
Step 7: Set the LEDs OFF
Step 8: Move the Stepper Motor in clockwise direction
Step 9: Phase D is excited to rotate in clockwise direction
Step 10: Phase C is excited to rotate motor and call delay
Step 11: Phase B is excited to rotate the motor and call delay
Step 12: Phase A is excited to rotate motor and call delay
Step 13: Stop

Program:

```
#include "LPC17xx.h"

#define PHASEA 0x00000009                    // Phase A value 1001
#define PHASEB 0x00000005                    // Phase B value 0101
#define PHASEC 0x00000006                    // Phase C value 0110
#define PHASED 0x0000000A                    // Phase D value 1100

unsigned int delay ;                         // Delay function definition

int main(void)
{
LPC_PINCON->PINSEL0 = LPC_PINCON->PINSEL0 & ~ 0x0000000F ;
                    // Configure P0.28 to P0.31 as GPIO
LPC_GPIO0->FIODIR = LPC_GPIO0->FIODIR | 0x00000003 ;
          // Configure P0.28 to P0.31 as Output
LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00000003 ;
// SET (1) P0.28 to P0.31, LEDs OFF
LPC_PINCON->PINSEL1 = LPC_PINCON->PINSEL1 & ~ 0x00000003 ;
// Configure P0.28 to P0.31 as GPIO
LPC_GPIO0->FIODIR = LPC_GPIO0->FIODIR | 0x00010000 ;
// Configure P0.28 to P0.31 as Output
LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00010000 ;
// SET (1) P0.28 to P0.31, LEDs OFF

LPC_PINCON->PINSEL4 = LPC_PINCON->PINSEL4 & 0XFFFFFF3F ;
// Configure P0.28 to P0.31 as GPIO
LPC_GPIO2->FIODIR = LPC_GPIO2->FIODIR | 0x00000008 ;
// Configure P0.28 to P0.31 as Output
LPC_GPIO2->FIOSET = LPC_GPIO2->FIOSET | 0x00000008 ;
```

```
// SET (1) P0.28 to P0.31, LEDs OFF

        while(1)
        {
                // Move stepper motor in anti clockwise direction

LPC_GPIO0->FIOCLR = LPC_GPIO0->FIOCLR | 0x00000001 ;
LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00000002 ;
LPC_GPIO0->FIOCLR = LPC_GPIO0->FIOCLR | 0x00010000 ;
LPC_GPIO2->FIOSET = LPC_GPIO2->FIOSET | 0x00000008 ;
                //LPC_GPIO2->FIOSET = PHASED ;
// Phase D is excited to rotate the motor
                //LPC_GPIO2->FIOCLR = (~PHASED) & 0x0000000F ;
// Phase D is cleared
                for(delay=0; delay<80000; delay++) ;
                // Small time Delay

LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00000001 ;
LPC_GPIO0->FIOCLR = LPC_GPIO0->FIOCLR | 0x00000002 ;
LPC_GPIO0->FIOCLR = LPC_GPIO0->FIOCLR | 0x00010000 ;
LPC_GPIO2->FIOSET = LPC_GPIO2->FIOSET | 0x00000008 ;

//LPC_GPIO2->FIOSET = PHASEC ;
// Phase C is excited to rotate the motor
                //LPC_GPIO2->FIOCLR = (~PHASEC) & 0x0000000F ;
// Phase C is cleared
                for(delay=0; delay<80000; delay++) ;
// Small time Delay

LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00000001 ;
LPC_GPIO0->FIOCLR = LPC_GPIO0->FIOCLR | 0x00000002 ;
LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00010000 ;
LPC_GPIO2->FIOCLR = LPC_GPIO2->FIOCLR | 0x00000008 ;

                //LPC_GPIO2->FIOSET = PHASEB ;
                                                // Phase B is excited to rotate the motor
                //LPC_GPIO2->FIOCLR = (~PHASEB) & 0x0000000F ;
// Phase B is cleared
                for(delay=0; delay<80000; delay++) ;
// Small time Delay

LPC_GPIO0->FIOCLR = LPC_GPIO0->FIOCLR | 0x00000001 ;
LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00000002 ;
LPC_GPIO0->FIOSET = LPC_GPIO0->FIOSET | 0x00010000 ;
LPC_GPIO2->FIOCLR = LPC_GPIO2->FIOCLR | 0x00000008 ;
                //LPC_GPIO2->FIOSET = PHASEA ;
// Phase A is excited to rotate the motor
                //LPC_GPIO2->FIOCLR = (~PHASEA) & 0x0000000F ;
// Phase A is cleared
                for(delay=0; delay<80000; delay++) ;
                // Small time Delay
        }
}
```

Peripheral Interface:

No Switches are there to Stepper Motor is connected to P0.28, P0.29, P0.30 and P0.31 through LEDs D7 to D10.Connect the Stepper motor at connector X3.

Output:

You can see stepper motor moving in a particular direction and corresponding phase changes you can observe on LEDs D7 to D10.

Result:

Thus interfacing of stepper motor with LPC1768 is executed and verified successfully

Exercise:

1. Develop an embedded C code to move the stepper motor in different directions.

Ex. No. 12 **INTERFACING LM35 (TEMPERATURE SENSOR)**
DATE:

Aim :

To interface LM35 (Temperature Sensor) with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Declare the variables delay, ADC, Temp and Temperature
Step 3: Call the function TargetResetInit()
Step 4: Call the function InitUART0()
Step 5: Call the function ADC_Init()
Step 6: UART0 puts ("ADC LM 35 Test") function is called
Step 7: Read ADC value
Step 8: Calculate Temperature using Temp, (adc*3.3)
Temperature = (Temp/1023)*100
Step 9: Display the temperature on LCD convert the result for ASCII to delay
Step 10: Delay

Program:

```
#include "LPC17xx.h"
#include "stdio.h"
#include "UART0.h"
#include "target.h"

#define PCADC 0x00001000

char adcreading[16] ;                          // Variable to hold the ADC converted Value

void ADC_Init (void)
                  // ADC initialation function
{
LPC_SC->PCONP |= PCADC ;
LPC_PINCON->PINSEL1 = (LPC_PINCON->PINSEL1 |= 0x00010000 );
        // P0.4 is configured as ADC Input
}

unsigned int ADC_GetAdcReading ()
{
        unsigned int adcdata;

LPC_ADC->ADCR = 0x01200302 ;// Select AD0.7, Select clock for ADC, Start of conversion,
        while(!((adcdata = LPC_ADC->ADGDR) & 0x80000000))
// Check end of conversion (Done bit) and read result
        {
        }
        return((adcdata >> 6) & 0x3ff) ;          // Return 10 bit result
}
int main (void)
{
        unsigned int delay,adc;
        float Temp, Temperature;
```

```
	//float adc;
	TargetResetInit();
		InitUart0();
		ADC_Init() ;
			// Initiate ADC Setting
		UART0Puts("ADC LM35 Test");
	// Display ADC Test

	while(1)
	{
		adc = ADC_GetAdcReading();                                    //
Read AD0.7 reading
		// Calculate Temperature
		Temp = (adc * 3.3);
		Temperature = (Temp / 1023) * 100;

		// Display Temperature on LCD

		sprintf(adcreading,"\nTemp =%0.03f *C",Temperature) ;
// Convert result into ASCII to display it on LCD
		UART0Puts(adcreading) ;                                       //
Display Temperature on LCD

		for(delay=0;delay<60000;delay++);
	}
}
```

Peripheral Interface:

To interface LM35 (present in Analog Input region on ADT Board) with P0.24 .

Output:

You can see output on Hyper Terminal.

Therefore Open **Hyper Terminal**. Go to **Start**->**All Programs**->**Accessories**->**Communications**->**Hyper Terminal**->Assign the Respective port-> Settings. Do proper settings (**Baud Rate: 19200**, Data Bits: **8**, Stop Bits: **1**, Echo: **Off**, Parity: **None**, Com Port: Com 1 (if other choose it)). Click on OK. Go to Port -> Open. If required Reset the ADT board. You can see the temperature on Hyper Terminal changing if you give some heat(even rubbing the sensor produces heat).

Result:

Thus interfacing of LM35 (Temperature Sensor) with LPC1768 is executed and verified successfully

Exercise:

1. Develop an embedded C code to enable the buzzer when temperature exceeds.

Ex. No. 13 **INTERFACING ZIGBEE ON SERIAL UART**
DATE:

Aim :

To Interface ZIGBEE on Serial UART with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Enable divisor latch access set 8bit word length, 1 stop bit, no parity bit
Step 3: Disable break transmission
Step 4: Enable bus clock divider
Step 5: Enable divisor latch (LSC)&(MSB)
Step 6: Disable divisor latch access
Step 7: Clear TxFIFO and enable Tx and Tx FIFOs
Step 8: Wait until transmit holding register is empty
Step 9:Then store to transmit holding register and wait until character is to be read
Step 10:Then read from the receiver buffer register
Step 11:stop

Program:

```
#include "LPC17xx.h"
#include <stdio.h>
#include <string.h>
#include "target.h"
#include "UART0.h"
#include "UART1.h"
#include "lpc_types.h"
int main(void)
{	//char ch1;
	char ch;
	TargetResetInit();
	InitUart0();
	InitUart1();
	UART0Puts("\nWelcome to Xee-Bee Testing\n");
		while(1)
		{
			if(UART1_IsReady())
			{
				UART0putchar(UART1_getchar());
			}
			if(UART0_IsReady())
			{
				ch = UART0getchar();
				printf("%c",ch);
			}
			}
}
```

Peripheral Interface: (S2 Zigbee)
Mount Zigbee on Zigbee socket/U3 .Switch ON S4.1 and S4.2(for UART1)
Cxx Indicates Co-Ordinator Zigbee.
Rxx Indicates Router Zigbee.
You should use two different Zigbees(Rxx and Cxx)(Ex. R1 and C1) on two different ADT boards.

Output:
You can see output on Hyper Terminal.
Therefore Open **Hyper Terminal**. Go to **Start**->**All Programs**->**Accessories**->**Communications**->**Hyper Terminal**->Assign the Respective port-> Settings. Do proper settings (**Baud Rate: 19200**, Data
Bits: **8**, Stop Bits: **1**, Echo: **Off**, Parity: **None**, Com Port: Com 1 (if other choose it)). Click on OK. Go to
Port -> Open. If required Reset the ADT board.
Press reset on both boards type something on keyboard of PC that respective key pressed is transmitted from that PC to another PC using Zigbee.

Result:

Thus interfacing of ZIGBEE on Serial UART with LPC1768 is executed and verified successfully

Ex. No. 14 **INTERFACING MATRIX KEYBOARD**
DATE:

Aim :

To interface Matrix Keyboard with LPC1768.

Algorithm :

Step 1: Start the program
Step 2: Define the ports P1.21, P1.20, P1.19, P1.18 for RET pins
Step 3: For scanning assign pins P2.25, p1.26, P1.23, P1.22
Step 4: Assign crystal frequency 12KHz and PLL multiplier as 1
Step 5: Direct the Rs, Rw of LCD as GPIO
Step 6: Call the delay
Step 7: Check whether the key is down or up
Step 8: If the key is down return 1
Step 9: If key is not pressed check for other key
Step 10: if already the key is pressed wait until it is pressed
Step 11: Return the key has been released and it is enable 10-1 also return 0
Step 12: Stop

Program:

```
#include "LPC17xx.h"
#include <stdio.h>
#include <string.h>
#include "target.h"
#include "UART0.h"
#include "KBD.h"
int i8ch ;
char szTemp[16] ;
int main (void)
{
TargetResetInit();
 InitUart0();
UART0Puts("Keypad Test \n");
           while(1)
           {
           i8ch = KBD_rdkbd() ;    // Read Keyboard
         sprintf(szTemp,"\nKeyCode = %02X",i8ch);          // Convert keycode into ASCII to display
it on LCD
        UART0Puts(szTemp) ;   // Display keycode on 2nd line of LCD
           }
}
```

Result:

Thus interfacing of matrix keyboard is executed and verified successfully.

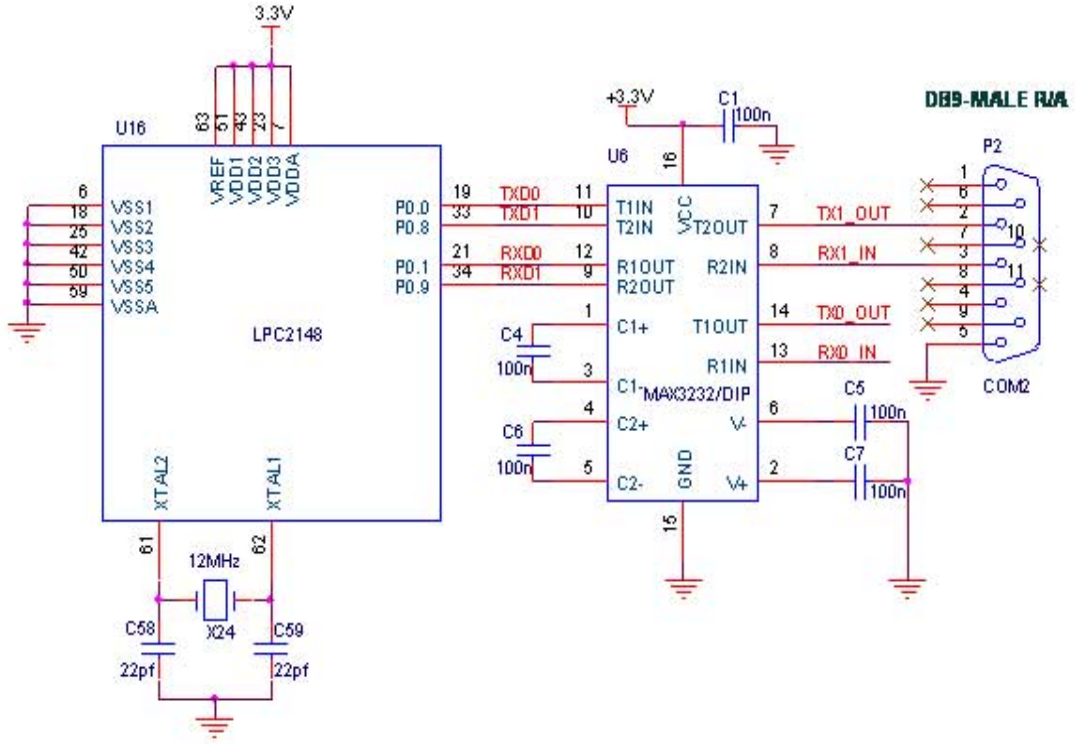
ADDITIONAL EXPERIMENTS

Ex. No. 1 **INTERFACING BLUETOOTH MODULE**
DATE:

Aim :

To write an embedded C code for interfacing Bluetooth Module with ARM CORTEX M3-LPC2148.

Circuitdiagram



Procedure:

Give +3.3V power supply to LPC2148 Primer Board; connect the 5V adapter with Bluetooth module which is connected with the LPC2148 Primer Board. There are two Bluetooth modules are required. One is connected with LPC2148 Primer Board; other one is connected with PC.First connect the serial cable between LPC2148 Primer board & PC. Then open the Hyper Terminal screen, select which port you are using and set the default settings. Now the screen should show some text messages. If the messages are correctly displayed in Hyper Terminal, then only connect the Bluetooth modules in LPC2148 Primer Board UART0 & PC.If you are not reading any data from UART0, then you just check the jumper connections & just check the serial cable is working. Otherwise you just check the code with debugging mode in Keil.

Program:

```
#define CR 0x0D
#include <LPC21xx.H>
void init_serial (void);
int putchar (int ch);
int getchar (void);
unsigned char test;
int main(void)
{
char *Ptr = "*** UART0 Demo ***\n\n\rType Characters to be echoed!!\n\n\r";
VPBDIV = 0x02; // Divide Pclk by two
init_serial();
while(1)
{
while (*Ptr) { putchar(*Ptr++);
} putchar(getchar()); // Echo terminal
}
}
void init_serial (void)
{
PINSEL0 = 0x00000005; // Enable RxD0 and TxD0
U0LCR = 0x00000083; //8 bits, no Parity, 1 Stop bit
U0DLL = 0x000000C3; //9600 Baud Rate @ 30MHz VPB Clock
U0LCR = 0x00000003;
}
int putchar (int ch)
{
if (ch == '\n')
{
while (!(U0LSR & 0x20));
U0THR = CR;
 }
while (!(U0LSR & 0x20));
return (U0THR = ch);
}
int getchar (void)
{
while (!(U0LSR & 0x01));
 return (U0RBR);
}
```

Result:

Thus, interfacing of GSM module with ARM CORTEX M3- LPC2148 using embedded C code was executed and verified successfully.

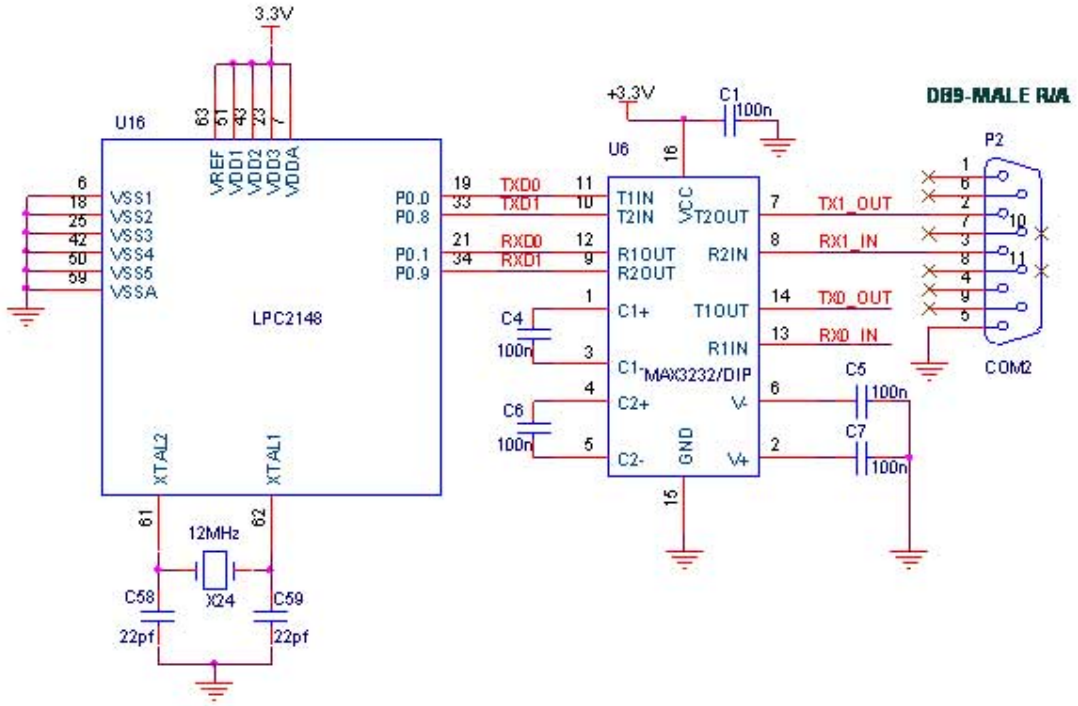
Ex. No. 2 **INTERFACING GSM MODULE**

DATE:

Aim :

To write an embedded C code for interfacing GSM Module with ARM CORTEX M3-LPC2148.

Circuit Diagram



Procedure:

Give +3.3V power supply to LPC2148 Primer Board; connect the +5V adapter with GSM module which is connected with LPC2148 Primer Board through UART0. Open the Hyper Terminal screen, select which port you are using and set the default settings. Now the screen should show some text messages.

The following Commands and sequence of events performed for sending text message to a mobile phone through GSM Modem interfaced with microcontroller:

- First select the text mode for SMS by sending the following AT Command to GSM Modem : AT+CMGF = 1 . This command configures the GSM modem in text mode.
- Send the following AT Command for sending SMS message in text mode along with mobile number to the GSM Modem : AT+CMGS =+923005281046 . This command sends the mobile number of the recipient mobile to the GSM modem.
- Send the text message string ("hello!") to the GSM Modem This is a test message from UART".
- Send ASCII code for CTRL+Z i.e., 0x1A to GSM Modem to transmit the message to mobile phone. After message string has been sent to the modem, send CTRL+Z to the micro-controller, which is equivalent to 0x1A (ASCII value).

If you not reading any text from UART0, then you just check the jumper connections & just check the serial cable is working. Otherwise you just check the code with debugging mode in Keil. If you want to see more details about debugging just see the videos in below link.

Program:

```
#define CR 0x0D
#include <LPC21xx.H>
#include <stdio.h>
 void getstring(unsigned char *);
int getchar (void) /* Read character from Serial Port */
void status_ok(void);
void Serial_Init(void);
void delay(unsigned int n);
void main(void)
{
unsigned int
cnt=0x80,m; char xx;
Serial_Init();
delay(50);
while(1)
{
printf("AT\r"); // AT COMMAND FOR INITIALING status_ok();
printf("AT+IPR=9600\r"); // AT COMMAND FOR BAUD RATE status_ok();
printf("AT+CMGR=2\r"); // Reading the message detail // at Index 1 with phone
number, data and time status_ok(); delay(250); printf("ATD9790550124;\r");//AT
COMMAND FOR CALL DIALING delay(250); status_ok();
delay(500);
 delay(500);
delay(500);
delay(500);
delay(500);
delay(500);
printf("ATH\r"); // AT COMMAND FOR CALL DISCONNECTING delay(250);
status_ok();
delay(500);
delay(500);
printf("ATDL\r"); // AT COMMAND FOR REDIALING delay(250); status_ok();
delay(500);
delay(500);
printf("ATH\r"); // AT COMMAND FOR ANSWERING THE CALL delay(250);
status_ok();
delay(500);
delay(500);
}
}
void getstring(unsigned char *array)
{
unsigned char temp=0, i=0;
do { temp = getchar();
*array++ = temp;
}
while((temp != '\r') && (temp != '\n'));
```

```
 *array = '\0';
}
int getchar (void) /* Read character from Serial Port */
{
while (!(U0LSR & 0x01));
 return (U0RBR);
}

void status_ok(void)
{
getstring(y);
 while(!(strstr(y,"OK")))
 getstring(y);
pointr = strstr(y,"OK");
lcd_cmd(0xc0);
lcd_data(*pointr++);
 lcd_data(*pointr);
delay(500);
 lcd_cmd(0x01);
}
void Serial_Init(void)
{
PINSEL0 |= 0X00000005; //Enable Txd0 and Rxd0
U0LCR = 0x00000083; //8-bit data, no parity, 1-stop bit
U0DLL = 0x00000061; //for Baud rate=9600,DLL=82
U0LCR = 0x00000003; //DLAB = 0;
}
void delay(unsigned int n)
{
unsigned int i,j;
for(i=0;i<n;i++)
{
for(j=0;j<12000;j++)
{;
}
}
```

Result:

Thus ,interfacing of GSM module with ARM CORTEX M3- LPC2148 using embedded C code was executed and verified successfully

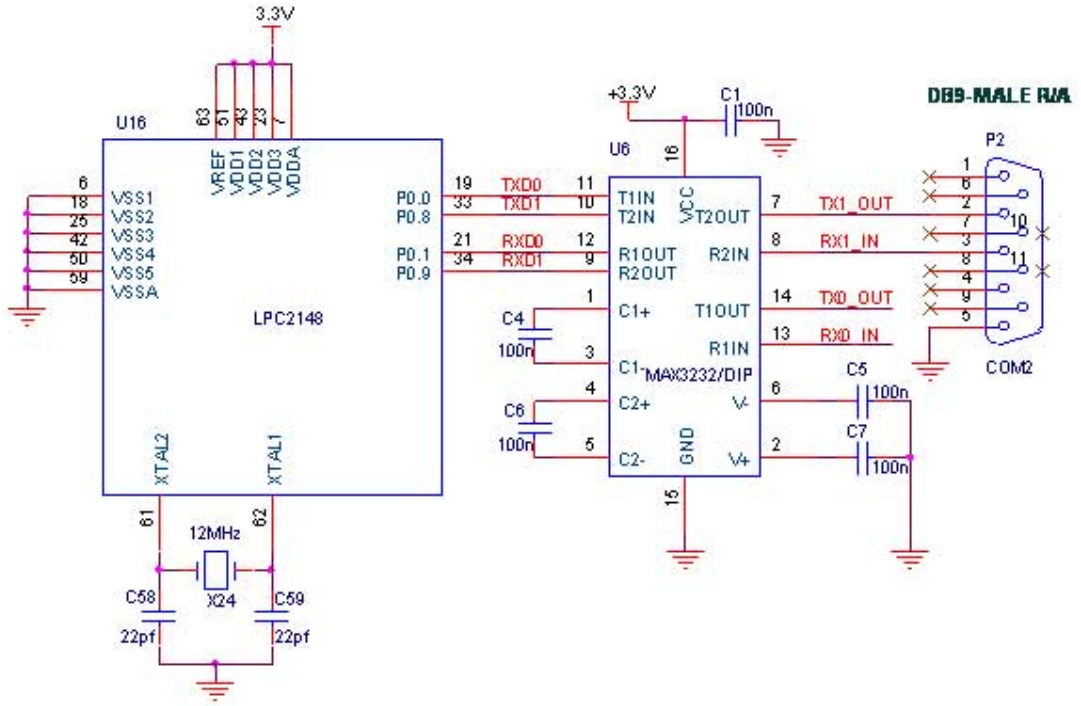
Ex. No. 3 **INTERFACING GPS MODULE**

DATE:

Aim :

To write an embedded C code for interfacing GPS Module with ARM CORTEX M3-LPC2148.

Circuit diagram



Procedure:

Give +3.3V power supply to LPC2148 Primer Board; connect +5V adapter with GPS module is connected with the LPC2148 Primer Board. Open the Hyper Terminal screen, select which port you are using and set the default settings. Now the screen should show some text messages.

If you are not reading any data from UART0, then you just check the jumper connections & just check the serial cable is working. Otherwise you just check the code with debugging mode in Keil. If you want to see more details about debugging just see the videos in below link.

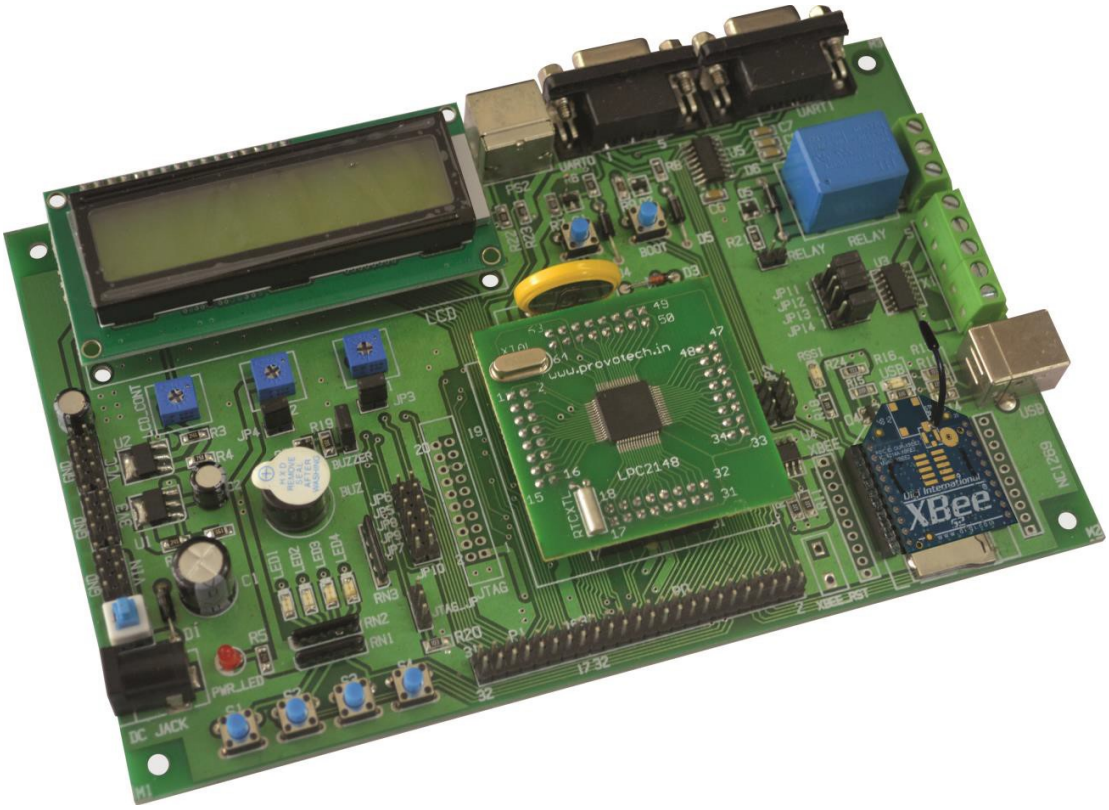
Program:

```
#define CR 0x0D
#include <LPC21xx.H>
void init_serial (void);
int putchar (int ch);
int getchar (void);
unsigned char test;
int main(void)
{
char *Ptr = "*** UART0 Demo ***\n\n\rType Characters to be echoed!!\n\n\r";
VPBDIV = 0x02; // Divide Pclk by two init_serial();
while(1)
{
while (*Ptr)
{
putchar(*Ptr++);
}
 putchar(getchar()); // Echo terminal
}
}
 void init_serial (void)
{
PINSEL0 = 0x00000005; // Enable RxD0 and TxD0
U0LCR = 0x00000083; //8 bits, no Parity, 1 Stop bit
    U0DLL = 0x000000C3; //9600 Baud Rate @ 30MHz VPB Clock
    U0LCR = 0x00000003;
}
int putchar (int ch)
{
if (ch == '\n')
{
while (!(U0LSR & 0x20));
 U0THR = CR;
}
while (!(U0LSR & 0x20));
return (U0THR = ch);
}
int getchar (void)
{
while (!(U0LSR & 0x01));
return (U0RBR);
}
```

Result:

Thus ,interfacing of GPS module with ARM CORTEX M3- LPC2148 using embedded C code was executed and verified successfully

ARM7 LPC2148 Development Board



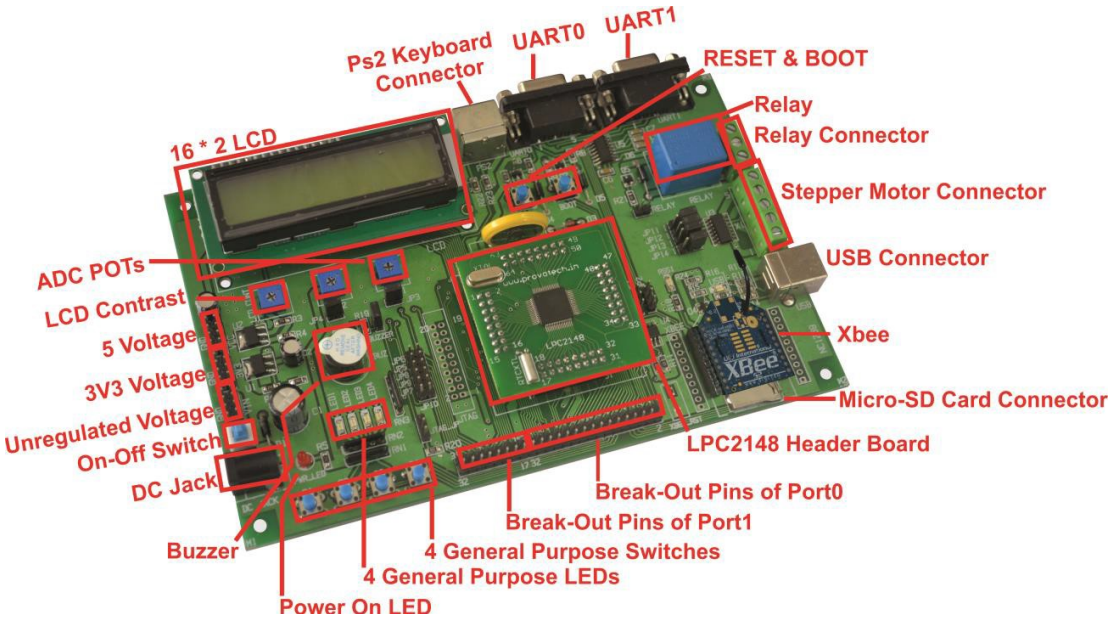
Introduction :

LPC2148 microcontrollers are based on a 32-bit ARM7TDMI-S CPU with real time emulation and embedded trace support that combine microcontroller with embedded high-speed flash memory 512 kB. A 128-bit wide memory interface and an unique accelerator architecture enable 32-bit code execution at the maximum clock rate. In-System Programming/In-Application Programming (ISP/IAP) via on-chip boot loader software. 10-bit ADCs provide a total of 6/14 analog inputs, with conversion times as low as 2.44 µs per channel. Single 10-bit DAC provides variable analog output Two 32-bit timers/external event counters (with four capture and four compare channels each), PWM unit (six outputs) and watchdog. Low power Real-Time Clock (RTC) with independent power and 32 kHz clock input. Multiple serial interfaces including two UARTs, two Fast I2C-bus (400 kbit/s), SPI and SSP with buffering and variable data length capabilities. Vectored Interrupt Controller (VIC) with configurable priorities and vector addresses. Up to 5 V tolerant fast general purpose I/O pins in a tiny LQFP64 package.

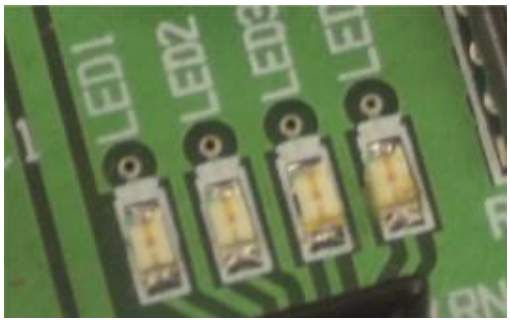
Onboard Features:

2. All port's Break-out
3. Power supply pin Break-out
4. 2 Analog potentiometer connected to ADC
5. 4 General Purpose Switches
6. 4 General Purpose LEDs

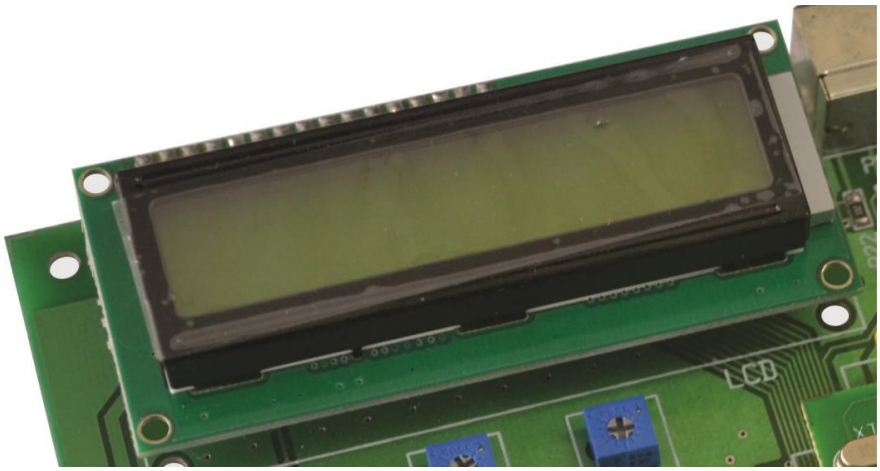
7. 16 *2 LCD
8. PS\2 Keyboard connector
9. UART0 and UART1 connector
10. Reset and Boot switches
11. RTC back-up Battery Cell
12. USB B type Connector
13. ULN2003 Stepper Motor driver
14. I2C based EEPROM
15. Micro-SD card connector
16. Buzzer and Relay
17. Relay connector
18. Stepper motor driver connector
19. XBEE
20. On-chip boot loader



LED: LEDs are connected to Port P0.19 – P0.22 in active LOW Configuration so to turn ON the LED user has to apply logic low (0) signal at the LED PIN and to turn OFF the LED apply logic high (1) signal at the LED pin.

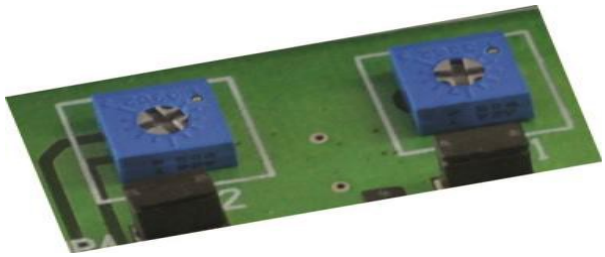


LCD: LCD (16*2LCD) is placed as shown in picture. Potentiometer is there for controlling the contrast of LCD. LCD

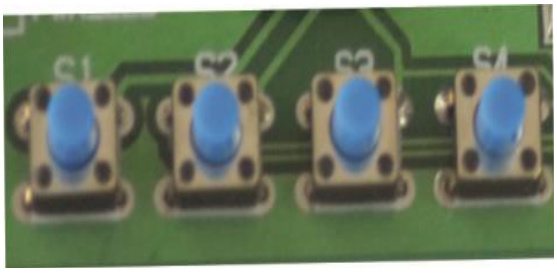


Buzzer: To enable Buzzer have to connect the jumper as shown. Buzzer is connected to pin P1.24.

Analog Input: This board is features with two external Potentiometer**s** which used to give precision analog input to ADC pins of ARM microcontroller. These potentiometer can be connected through jumper(JP3 and JP4) connected to ADC pins P0.28 – AD0.1 and P0.29 – AD0.2.

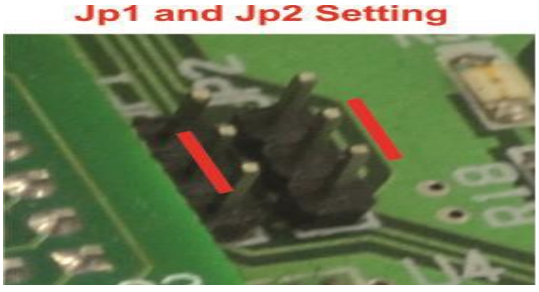


Switches: Switches are connected to Port P0.10 – P0.13 in active LOW Configuration, Pulled up by Resistor network, when pressed logic '0' gives else logic '1'.

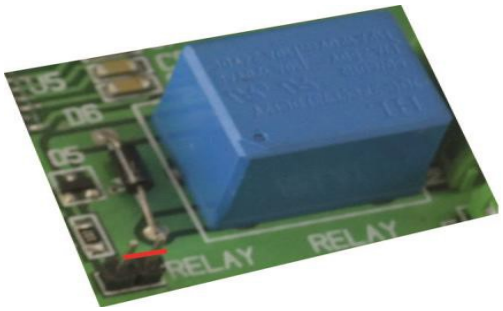


UART0 and UART1: UART0 is used for programming the LPC2148 with on chip boot loader. It can also be used as general purpose Communication Port/serial port.
UART1 is general purpose Communication Port/serial port.

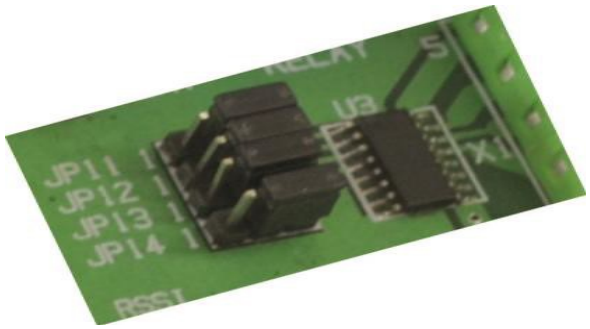
XBEE: XBEE is a wireless communication device transceiver. This board has functionality to mount xbee directly to UART1, for that user has to change the jumper setting as shown.



Relay: To enable Relay have to connect the jumper as shown. Buzzer is connected to pin P1.25.



Stepper motor driver: to enable stepper motor driver change the jumper (JP11 – JP14) as shown in figure.



PYTHON PROGRAMMING PRACTICE

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. Using third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, so Python-based software can be distributed to, and used on, those environments with no need to install a Python interpreter.

First Python Program

Let us execute programs in different modes of programming.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt –

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text at the Python prompt and press the Enter:

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!");**. However in Python version 2.4.3, this produces the following result:

```
Hello, Python!
```

Example Project :1

Python program to display calendar of given month of the year

```
# import module
import calendar

# ask of month and year
yy = int(input("Enter year: "))
mm = int(input("Enter month: "))

# display the calendar
print(calendar.month(yy,mm))
```

Example Project : 2

Program to make a simple calculator that can add, subtract, multiply and divide using functions

```
# define functions
def add(x, y):
   """This function adds two numbers"""

   return x + y

def subtract(x, y):
   """This function subtracts two numbers"""

   return x - y

def multiply(x, y):
   """This function multiplies two numbers"""

   return x * y

def divide(x, y):
   """This function divides two numbers"""

   return x / y

# take input from the user
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

choice = input("Enter choice(1/2/3/4):")

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

if choice == '1':
```

```
    print(num1,"+",num2,"=", add(num1,num2))

  elif choice == '2':
    print(num1,"-",num2,"=", subtract(num1,num2))

  elif choice == '3':
    print(num1,"*",num2,"=", multiply(num1,num2))

  elif choice == '4':
    print(num1,"/",num2,"=", divide(num1,num2))
  else:
    print("Invalid input")
```

Example Project :3

```
  # Python program to convert decimal number into binary, octal and hexadecimal number system

  # Take decimal number from user
  dec = int(input("Enter an integer: "))

  print("The decimal value of",dec,"is:")
  print(bin(dec),"in binary.")
  print(oct(dec),"in octal.")
  print(hex(dec),"in hexadecimal.")
```

Example Project :4

```
  # Program to add two matrices using nested loop

  X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]

  Y = [[5,8,1],
     [6,7,3],
     [4,5,9]]

  result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

  # iterate through rows
  for i in range(len(X)):
    # iterate through columns
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]

  for r in result:
    print(r)
```

Example Project :5

Python Code for "Guess the Number"

```
import random

guesses_made = 0

name = raw_input('Hello! What is your name?\n')

number = random.randint(1, 20)
print 'Well, {0}, I am thinking of a number between 1 and 20.'.format(name)

while guesses_made < 6:

    guess = int(raw_input('Take a guess: '))

    guesses_made += 1

    if guess < number:
        print 'Your guess is too low.'

    if guess > number:
        print 'Your guess is too high.'

    if guess == number:
        break

if guess == number:
    print 'Good job, {0}! You guessed my number in {1} guesses!'.format(name, guesses_made)
else:
    print 'Nope. The number I was thinking of was {0}'.format(number)
```

VIVA-VOCE QUESTIONS

1) Explain what is embedded system in a computer system?

An embedded system is a special purpose computer system which is completely encapsulated by device it controls. It is a programmed hardware device in which the hardware chip is programmed with specific function. It is a combination of hardware and software.

2) Mention what are the essential components of embedded system?

Essential components of embedded system includes

- **Hardware**
- Processor
- Memory
- Timers
- I/O circuits
- System application specific circuits
- **Software**
- It ensures the availability of System Memory
- It checks the Processor Speed availability
- The need to limit power lost when running the system continuously
- **Real Time Operating System**
- It runs a process as per scheduling and do the switching from one process to another

3) Mention how I/O devices are classified for embedded system?

The I/O devices of embedded system are classified into two categories

- Serial
- Parallel

4) Why embedded system is useful?

With embedded system, it is possible to replace dozens or even more of hardware logic gates, input buffers, timing circuits, output drivers, etc. with a relatively cheap microprocessor.

5) Explain what are real-time embedded systems?

Real-time embedded systems are computer systems that monitor, respond or control an external environment. This environment is connected to the computer system through actuators, sensors, and other input-output interfaces.

6) Explain what is microcontroller?

The microcontroller is a self-contained system with peripherals, memory and a processor that can be used as embedded system.

7) Mention what is the difference between microprocessor and microcontroller?

Microprocessor is managers of the resources (I/O, memory) which lie outside of its architecture

Microcontroller have I/O, memory, etc. built into it and specifically designed for control

8) What does DMA address will deal with?

DMA address deals with physical addresses. It is a device which directly drives the data and address bus during data transfer. So, it is purely physical address.

9) Explain what is interrupt latency? How can you reduce it?

Interrupt latency is a time taken to return from the interrupt service routine post handling a specific interrupt. By writing minor ISR routines, interrupt latency can be reduced.

10) Mention what are buses used for communication in embedded system?

For embedded system, the buses used for communication includes

- **I2C**: It is used for communication between multiple ICs
- **CAN**: It is used in automobiles with centrally controlled network
- **USB**: It is used for communication between CPU and devices like mouse, etc.

While ISA, EISA, PCI are standard buses for parallel communication used in PCs, computer network devices, etc.

11) List out various uses of timers in embedded system?

Timers in embedded system are used in multiple ways

- Real Time Clock (RTC) for the system
- Initiating an event after a preset time delay
- Initiating an even after a comparison of preset times
- Capturing the count value in timer on an event
- Between two events finding the time interval
- Time slicing for various tasks
- Time division multiplexing
- Scheduling of various tasks in RTOS

12) Explain what is a Watchdog Timer?

A watchdog timer is an electronic device or electronic cards that execute specific operation after certain time period if something goes wrong with an electronic system.

13) Explain what is the need for an infinite loop in embedded systems?

Embedded systems require infinite loops for repeatedly processing or monitoring the state of the program. For instance, the case of a program state continuously being verified for any exceptional errors that might just happen during run-time such as memory outage or divide by zero, etc.

14) List out some of the commonly found errors in Embedded Systems?

Some of the commonly found errors in embedded systems are

- Damage of memory devices static discharges and transient current
- Address line malfunctioning due to a short in circuit
- Data lines malfunctioning
- Due to garbage or errors some memory locations being inaccessible in storage
- Inappropriate insertion of memory devices into the memory slots
- Wrong control signals

15) Explain what is semaphore?

A semaphore is an abstract data type or variable that is used for controlling access, by multiple processes to a common resource in a concurrent system such as multiprogramming operating system. Semaphores are commonly used for two purposes

- To share a common memory space
- To share access to files

16) When one must use recursion function? Mention what happens when recursion functions are declared inline?

Recursion function can be used when you are aware of the number of recursive calls is not excessive. Inline functions property says whenever it will call; it will copy the complete definition of that function. Recursive function declared as inline creates the burden on the compilers execution.

17) What are the Processors in Embedded systems?

- General Purpose Processor (GPP) Microprocessor. Microcontroller. ...
- Application Specific System Processor (ASSP)
- Application Specific Instruction Processors (ASIPs)
- GPP core(s) or ASIP core(s) on either an Application Specific Integrated Circuit (ASIC) or a Very Large Scale Integration (VLSI) circuit.

18)What is the need for an infinite loop in Embedded systems?

- Infinite Loops are those program constructs where in there is no break statement so as to get out of the loop, it just keeps looping over the statements within the block defined.

Example:

```
While(Boolean True) OR for(;;);
{
//Code
}
```

19)What are the characteristics of embedded system?

The Characteristics of the embedded systems are as follows-

1. Sophisticated functionality
2. Real time behavior
3. Low manufacturing cost
4. Low power consumption
5. User friendly
6. Small size

20) What are the types of embedded system?

They are of 4 types

1. General computing
2. Control System
3. Digital Signal Processing
4. Communication and network

21) Why we use embedded systems?

Embedded systems avoid lots of electronic components and they have rich built in functionality. They reduce the cost and maintenance cost and the probability of failure of embedded system is less so embedded system are in very much use now a days.

22) What are the languages used in embedded system?

Assembly language and C are basically used for embedded system. Java and ADA are also preferred.

23) What are the commonly found errors in Embedded Systems?

- Damage of memory devices due to transient current and static discharges.
- Malfunctioning of address lines due to a short in the circuit
- Malfunctioning of Data lines.
- Some memory locations being inaccessible in storage due to garbage or errors.
- Improper insertion of Memory devices into the memory slots
- Faulty control signals.

24) What are the examples of embedded system?

An example of embedded system includes ATMs, cell phones, printers, thermostats, calculators, and videogame consoles. Handheld computers or PDAs are also considered embedded devices because of the nature of their hardware design, even though they are more expandable in software terms.

25) What is embedded system programming?

Embedded programming is a specific type of programming that supports the creation of consumer facing or business facing devices that don't operate on traditional operating systems the way that full-scale laptop computers and mobile devices do.

26) What is an embedded developer?

Embedded software is computer software, written to control machines or devices that are not typically thought of as computers. It is typically specialized for the particular hardware that it runs on and has time and memory constraints.

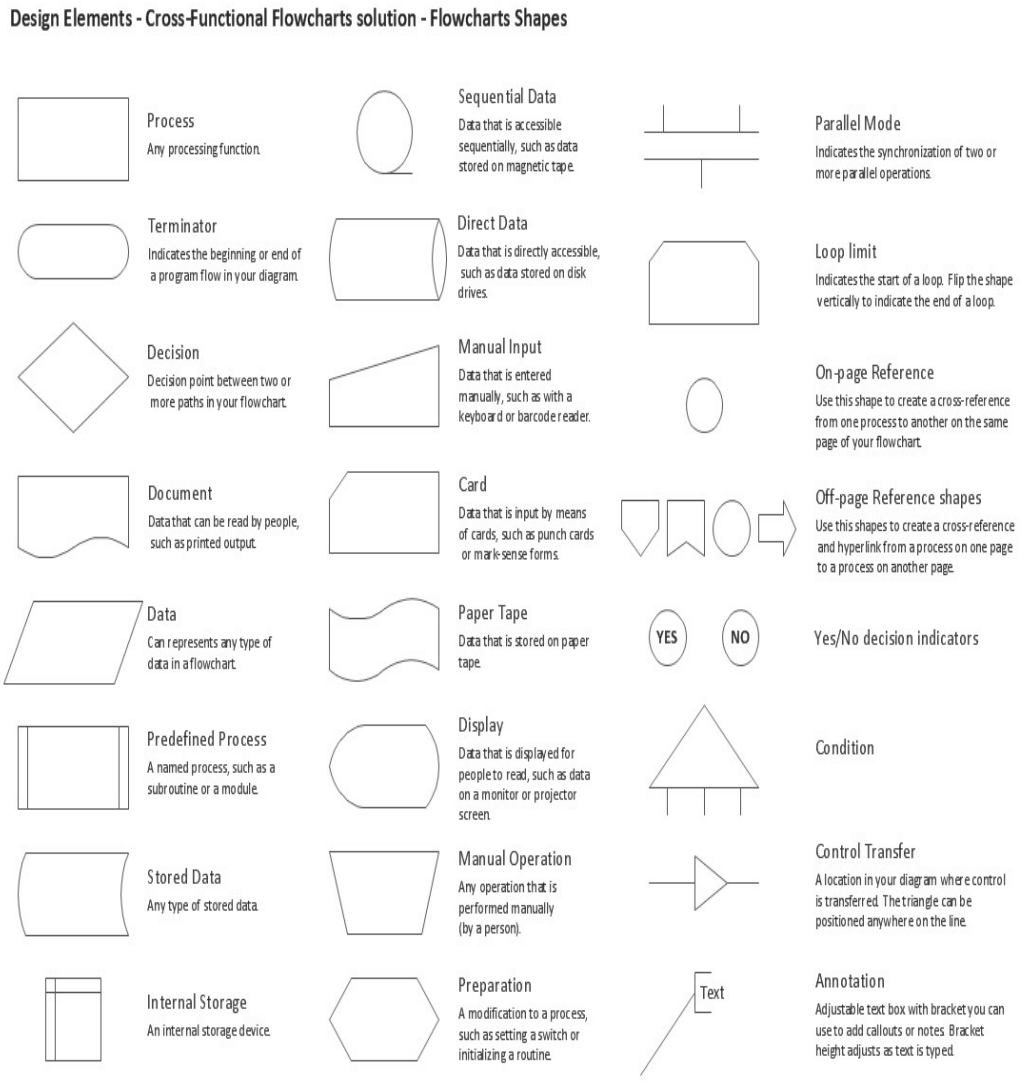
27)What is embedded c?

The C standard doesn't care about embedded, but vendors of embedded systems usually provide standalone implementations with whatever amount of libraries they're willing to provide. C is a widely used general purpose high level programming language mainly intended for system programming.

28) What is main difference between C and embedded C?

As, embedded C is generally an extension of the C language, they are more or less similar. However, some differences do exist, such as: C is generally used for desktop computers, while embedded C is for microcontroller based applications. C can use the resources of a desktop PC like memory, OS, etc.

THE FLOWCHART SYMBOLS AND THEIR USAGE



.