Introduction to eROS (easyRadio Operating System)

eROS, the easyRadio Operating System is used within eRIC, the easy Radio Integrated Controller RF transceiver module.

eRIC's processor memory (32k) is partitioned and eROS provides a simplified and elegant means of configuring and programming a complex microcontroller and the multiple control registers of the RF transceiver. The other partition provides an optional user accessible application code area.

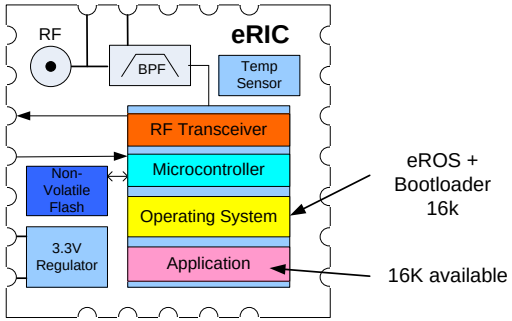


Figure 1 eRIC Transceiver Block Diagram

Radio parameters such as frequency, channel, output power and data rate are passed to the OS by the application code and radio data is sent and received by simply calling predefined functions.

The eROS API replaces low level chip specific code with intuitive pin commands that allow the multiple general purpose I/O pins and internal function blocks to be configured and interfaced to external hardware. These built in functions make customisation easy for the novice and powerful for advanced programmers.

Code is written in 'C' and currently supports the CC4305137 System-on-Chip (SoC) RF transceiver IC from Texas Instruments (TI).

This architecture eliminates the need for a separate application microcontroller and thus minimises cost and power consumption for simple 'sense and control' RF nodes such as might be employed within the 'Internet of Things'.

eRIC modules incorporating eROS offer the following features:

- 250 byte radio transmit/receive buffers
- Precise frequency control
- Adjustable RF Power from -30 to +12dBm
- Over air RF data rates of up to 500kbps
- Power saving modes
- Built in Temperature Sensor
- 18 General Purpose Input/Output Pins (GPIO)
- UART, SPI, A-D convertor
- 256Bytes of EEPROM *
- 2K user RAM
- Dynamic CPU clock speed control

* Flash memory emulated as EEPROM

Software Development

Getting started:

- Locate the latest 'eRIC_Flash_Setup_x.exe' setup program on the USB stick (or download from www.plrs.co.uk) and double click to install on the PC.
- Download and install the latest Texas Instruments Code Composer Studio (CCS) from: http://processors.wiki.ti.com/index.php/Category:CCS

- Run the CCS program and from the 'Project' tab select 'Import Existing CCS Eclipse Project'. (Figure 2)
- Select 'Archive' file and browse to C:\Program Files (x86)\LPRS\eRIC1.1 folder where you will find the eRICxeasyRadioV1_1.zip archive.
- Select the Discovered project and click Finish.
- Modify the source code as required and compile/build.
- The program can then be 'flashed' to the module using the eRIC Flash Programmer software tool.

Further information on programming is provided within the eRIC Tutorials 1, 2 and 3.

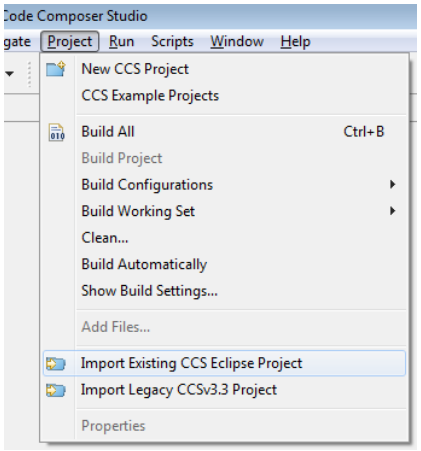


Figure 2 Import Existing CCS Eclipse Project

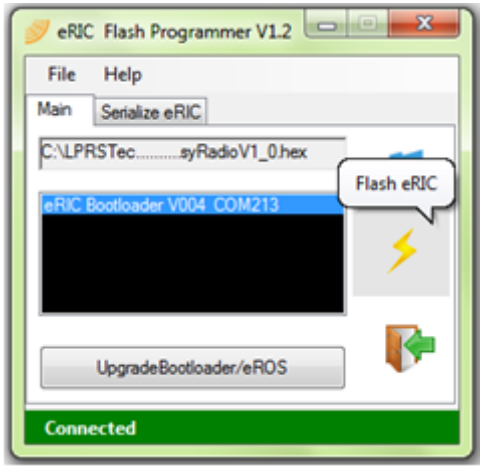


Figure 3 eRIC Flash Programmer

eRIC Flash Programmer

- On the Development Board bridge JP1 (Bootloader Enable) with the supplied jumper.
- Connect the Development Board to the PC using the supplied USB cable.
- Run the installed eRIC Flash Programmer V1.x. The program window lower banner will initially flash red and report 'Scanning for connection'.
- Switch the Development Board 'On' and momentarily press the 'Reset' push button switch.
- When connection is established the lower banner should change to green and report 'Connected' (as shown).
- The internal black window will then display 'eRIC Bootloader', the Bootloader version number and the Com Port number which is automatically selected.

eROS Application Programmers Interface

Radio Functions

Functions	Parameters	Description	Notes	OS
eROS_Initialise(RadioFrequency);	RadioFrequency can be any frequency value e.g. eRIC_RadioInitialise(43392000); RadioFrequency = 0 when Radio is not required	Initialises the eROS and set RF registers if required and set the frequency as passed in	This MUST be done once to set up eROS All further updates to RF use the eRIC_RadioUpdate() function	
eRIC_Rx_Enable();	None	Enable the Radio receiver	If this is not enabled, Radio cannot receive any data, but can transmit data. Works only as transmitter.	
eRIC_Rx_Disable();	None	Disable Radio receiver Can be disabled at any time		
eRIC_RadioUpdate();	None Values are changed prior to call	Changes to Power, Channel, Frequency, Data Rates etc. are stored using this function		
eRIC_RfSenddata();	None	Sends 'eRIC_RadioTx_BuffCount' bytes from 'eRIC_RadioTx_Buffer' array	eRIC_RadioTx_Buffer must be loaded, and eRIC_RadioTx_BuffCount set before this call	

eRIC_ReadRfByte()	None
	Returns next unread RF byte from buffer
		E.g.
		while(eRIC_Rxdata_available)
		{
		myBuffer[i++] = eRIC_ReadRfByte();
		}		

eRIC_RadioAsyncMode();
 	
Was:
~~eRIC_RawDataModeOn();~~
	None	Turn Raw data mode on	E.g. To enable Rx Rawdata:
			eRIC_RadioAsyncMode();
			Pinx_SetAsAsyncRxData(); // x ericpin
			
			E.g. To enable Tx Rawdata:
			eRIC_RadioAsyncMode();
			Pinx_SetAsAsyncTxData(); // x ericpin
			Pinx_SetHigh(); or Pinx_SetLow(); to send data.
				eROS 4

eRIC_RadioPacketMode(); ~~Was:~~ ~~eRIC_RawDataModeOff();~~	None	Turn Raw data mode off	Enters into eRIC Packet Mode.	eROS 4
eRIC_SetModulationCarrierOn();	None	Sets the Modulated Carrier on	Transmit continuous modulated Carrier at selected Over Air data rate. Useful for checking transmitter frequency and RF Power output	
eRIC_SetHighSideCarrierOn();	None	Sets high side FSK Carrier on	Transmit continuous upper FSK Carrier Useful for checking FSK deviation limit	
eRIC_SetLowSideCarrierOn();	None	Sets low side FSK Carrier on	Transmit continuous lower FSK Carrier Useful for checking FSK deviation limit	

eRIC_SetCarrierOff();	None	Turn off transmitter Carrier		
eRIC_Tx_CarrierOn();	None	Turns on transmitter Carrier	Mostly useful in AsyncMode to turn transmitter on	eROS 4
eRIC_Tx_CarrierOff();	None	Turn off transmitter Carrier	Mostly useful in AsyncMode to turn transmitter off	eROS 4
eRIC_GetLastPacketRSSI();	None	This returns the real signed RSSI value in dBm of the last packet received (Only in Packet mode)	This value is updated every time a new message is received.. E.g. -23db or -87db etc.	eROS 4
eRIC_GetLiveRSSI();	None	This returns the signed live RSSI value in dBm	Useful in applications to find range of the receiver. E.g. -107dbm, -93dbm etc.	eROS 4
eRIC_GroupIDEnable(IDNumber);	IDNumber = 4578; Any two byte groupID		eRIC_GroupID(4578); Note: Whenever groupID is enabled, eROS CRC is also added for more secured data packet	eROS 4
eRIC_GroupIDDisable();	None	Disables GroupID	eRIC_GroupIDDisable();	eROS 4
Variables	**VariableType**	**Description**	**Example**	
eRIC_Frequency	unsigned long	Desired frequency in Hz of the radio	eRIC_Frequency = 869750000; eRIC_RadioUpdate();*	
eRIC_Power	signed char (-30 to +12)	Power level from -30 to +12dBm	eRIC_Power = -12; // (Set to -12dBm) eRIC_RadioUpdate();*	
eRIC_Channel	unsigned char (0 – 255)	Sets frequency channel (eRIC_Frequency + (eRIC_Channel x eRIC_ChannelSpacing))	eRIC_Channel = 4; // Set Channel 4 eRIC_RadioUpdate();* eRIC_Channel = 85; // Set Channel 85 eRIC_RadioUpdate();*	
eRIC_ChannelSpacing	unsigned long	Sets the space in Hz between channels Allowed values: Up to 400000 Hz	Set to 100KHz Channel Spacing: eRIC_ChannelSpacing = 100000; eRIC_RadioUpdate();*	
eRIC_RfBaudRate	unsigned long	Sets the RF data rate of the transceiver Allowed Values: 1200, 2400, 4800, 9600, 10000, 19200, 38400 (default), 76800, 100000, 175000, 250000, 500000.	Set Data Rate to 250Kbps: eRIC_RfBaudRate = 250000 ; eRIC_RadioUpdate();*	
eRIC_RadioTx_BuffCount	unsigned char	Sets the number of bytes to transmit	eRIC_RadioTx_BuffCount = 10;	
eRIC_RadioTx_Buff[];	unsigned char 250 Bytes	This is the Radio Transmit buffer and should be filled before sending	eRIC_RadioTx_Buff[0] = 'e'; eRIC_RadioTx_Buff[1] = 'R'; eRIC_RadioTx_BuffCount = 2; eRIC_RfSenddata();	
IsGroupID_Enabled();	Boolean	Return non-zero, if group id is enabled		eROS 4
IsRadio_Rx_Busy();	Boolean	Returns non-zero, if radio is busy receiving		eROS

		data		4
Is60ByteLimitEnabled();	Boolean	Returns non-zero, if cpuclock speed is less than 9 times the radio baudrate. 60 Bytes of limited data is only sent to prevent locking up the radio when clockspeed is less than necessary radiobaudrates.		eROS 4
IsAsyncModeEnabled();	Boolean	Returns non-zero when asynchronous mode is selected		eROS 4

eRIC_RxPowerLevel	Char . only 0-7 values are to be used	0 = Radio is 100% ON
		1 12.5% of the time or current of Radio ON
		2 6.25% of the time or current of Radio ON
		3 3.13% of the time or current of Radio ON
		4 1.56% of the time or current of Radio ON
		5 0.78% of the time or current of Radio ON
		6 0.39% of the time or current of Radio ON
		7 0.20% of the time or current of Radio ON
			eRIC_RxPowerLevel= 7;
			eRIC_RadioUpdate();
			
			This sets the Radio in to lowest power mode, which is about 0.2% of what it will be when the radio was completely on. This setting brings the radio current consumption down to 32uA which 0.2%of16mA
			
			Clockspeed should be always 9 times more than Baudrate of the radio to work low power modes
			CpuFrequency>=9*eRIC_RfBaudrate
				eROS 4

eRIC_TxPowerLevel	Char . only 0-7 values are to be used	This need to be set in according to the setting of the eRIC_RxPowerLevel and should follow the below equation:
		eRIC_TxPowerLevel>=
		eRIC_RxPowerLevel.
			eRIC_TxPowerLevel = 7;
			eRIC_RadioUpdate();
			Clockspeed should be always 9 times more than Baudrate of the radio to work low power modes
			CpuFrequency>=9*eRIC_RfBaudrate
				eROS 4

* Note that 'eRIC_RadioUpdate();' does not need to be called after each setting change. Multiple settings can be modified followed by a call to 'eRIC_RadioUpdate();'
 	
eRIC_Frequency = 915000000; // Set Channel 0 position to 915MHz (Base/Centre Frequency)
eRIC_ChannelSpacing = 150000; // Set Channel Spacing to 150KHz
eRIC_Channel = 1; // Set Channel 1 (915.150MHz)
eRIC_RfBaudRate = 250000 ; // Set data rate to 250Kbps
eRIC_Power = -3; // Set Power to FCC USA limit (-3dBm)
eRIC_RadioUpdate(); // Single call to update all above changes				

Non Radio Functions and Commands

Functions	Parameters	Description	Notes	OS
eRIC_SetCpuFrequency(ClockFrequency);	ClockFrequency = 10000, 20000, 32000, 40000, 50000, 60000 and 70000 to 2000000 *Improvements from previous version*	Sets the clock frequency	If this is not used, the default clock frequency set to 1048576.	eROS4
eRIC_ReadAdc(AdcPinnumber,ReferenceVoltage);	AdcPinnumber = 1, 2, 3, 4, 5 and 22. ReferenceVoltage = 0,1,2,3 Where 0 = Ref_1_5v 1 = Ref_2_0v 2 = Ref_2_5v 3 = Ref_2_5v	Reads 12 bit Digital Adc value on eRIC pin passed in and with Reference voltage. For eg: eRIC_ReadAdc(1,0); This gives a digital Adc value on pin 1 with reference voltage 1.5v. Actual voltage =((Received 12 bit digital vaule)*1.5)/4096; 1.5 because 0 is passed and 4096 because its a12 bit ADC.	This is a 2 Byte data. 12bits Adc. Before using this function, pin mapping is required for the particular pin used in the function	
eRIC_GetTemperature();	None	Gives the current temperature of the chip device. Return the real float value of temperature in decimal in degree Celsius. Accuracy is +/-3 degrees C.		eROS4
eRIC_UARTAInitialise(Baudrate);	Baudrate = 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 or can be any UART Baud	Initialise the Uart with the desired Baudrate	Before initialising, Uart_Rx and Uart_TX must be mapped to one of the secondary mapping pins on eRIC	
eRIC_UARTA_SetBaud(Baudrate);	Baudrate = 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 or can be any UART Baud	Baud rate can be changed at any time, after initialisation	Changing baud rates affects the timing of the RX and TX data, so check the timings when the baud rate is changed	
eRIC_UartAReceiveByte()	None	Gets one Byte of Uart Rx Data	Uart_Rx and Uart_TX must be mapped to one of the secondary mapping pins on eRIC	
eRIC_UartARxBufferIsBusy(); eRIC_UartARxBufferIsEmpty();	None	Test to see if RX buffer is busy or empty		
eRIC_UartARxInteruptDisable(); eRIC_UartARxInteruptEnable();	None	UartA Rx interrupt can be Enabled and Disabled	Interrupts can be handled using Pragma vectors, which can be found in eRIC.c. This can be copied into main application.	
eRIC_UartARxIsEnabled();	None	Test to see if the Rx interrupt is enabled		
eRIC_UartASendByte(Data);	Data can be any unsigned char	Transmits one byte of data on Uart TX	Uart_Rx and Uart_TX must be mapped to one of the secondary mapping pins on eRIC	
eRIC_UartATxBufferIsBusy();	None	Test to see if Tx buffer is Busy or Empty		
eRIC_UartATxBufferIsEmpty();				
eRIC_UartATxInteruptDisable();	None	UartA tx interrupt can be enabled or disabled		

eRIC_UartATxInteruptEnable();				
eRIC_UartATxIsEnabled();	None	Test to see if UartA tx interrupt enabled		
eRIC_SpiAInitialise(SpiClock);	SpiClock in Hz	Initialises Spi with desired clock speed	Spi Slave, Master and Clock pins must be mapped using Secondary mapping function before initialising	
eRIC_SpiARead(Data);	Data is dummy data to read SPIdata	Gets Spi Data after sending a dummy byte		
eRIC_SpiAWrite(Data);	Data is any unsigned char	Send a byte of data through SPI		
eRIC_Eeprom_Read(Address);	Address can range from 0-255. As EEprom is only 256 bytes size.	Reads EEprom data from the address passed in		
eRIC_Eeprom_Write(Address,Data) ;	Address can range from 0-255. As EEprom is only 256 bytes size. Data is any char	Write the data on to EEprom address passed in.		
eRIC_GetSerialNumber()	None	Return 4 bytes long serial number. This is a unique serial number to each eRIC module. Each module is tracked and licensed based on this serial number.	E.g. 400000AB 900000CB etc. The MSB tells which module it is. eRIC4 or eRIC9.	eROS4
eRIC_Delay(MilliSeconds)	MilliSeconds ranges from 1-65535ms.		eRIC_Delay(1000); //Waits for 1 sec	eROS4
eRIC_AES_SetKey(mode,*Key);	Mode = 0 for encryption, Mode = 1 for decryption . Key should be the address of the 16 Bytes char to generate key.		Everytime a mode is changed, this eRIC_AES_SetKey(mode,*Key); is used to generate key.	eROS4

eRIC_AES_Run(*Data);	16 bytes data is passed to perform encryption or decryption.		E.g. Unsigned char Key[16];//fill this buffer before generating key in next step eRIC_AES_SetKey(0,*Key);
			Unsigned char Data[16];//fill this buffer before encrypting in next step.
			eRIC_AES_Run(*Data);
			The encrypted data is available in Data variable only.
			Similarly decryption is performed in the same way
				eROS4

eRIC_WDT_Setup(Modebits);	Where Modebits=(Clocksource+Time Interval) ClockSource available are: 1) eRICWDT_Cs_CPU 2) eRICWDT_Cs_32k 3) eRICWDT_Cs_10k TimeInterval available are: 1) eRICWDT_Interval_64	Sets the Watch dog timer with selected clock source and triggers after the selected number of interval of cycles with that clock source	E.g. eRIC_WDT_Setup(eRICWDT_Cs_32k+eRICWDT_Interval_32768); This sets the watch dog timer with 32k clock and triggers the interrupt after every 32768 cycles which is 1 second. ***This watch dog timer never resets the module. It only sets the flag or triggers the	eROS4

	2) eRICWDT_Interval_512 3) eRICWDT_Interval_8192 4) eRICWDT_Interval_32768 5) eRICWDT_Interval_524288 6) eRICWDT_Interval_8388608 7) eRICWDT_Interval_134217728 8) eRICWDT_Interval_2147483648		interrupt vector if handled.	
eRIC_WDT_Stop();	None	This stops the already running WDT		eROS4
eRIC_WDT_Start();	None	This starts the WDT again with preset WDT clocksource and Interval		eROS4
eRIC_WDT_Reset();	None	This resets the WDT timer and counts again from start. If one doesn't want to trigger the WDT, care should be taken to reset WDT before the WDT timer expires		eROS4
eRIC_WDT_InterruptEnable();	None	This enables the interrupt for WDT	The code for WDT interrupt vector is available in eRIC.c . Code can be written in there or it can be copied into main. Whenever interrupt triggers, program counter jumps in to it	eROS4
eRIC_WDT_InterruptDisable();	None	This disables the WDT interrupt		eROS4
eRIC_WDT_HasInterrupted();	None	This returns non zero if interrupt flag is set and interrupt has been triggered	If WDT interrupt vector is not used, this can be monitored in code.	eROS4
eRIC_WDT_ClearInterruptFlag();	None	This clears the WDT interrupt flag	If WDT interrupt vector is not used, this can be monitored in code	eROS4
eRIC_PowerOnReset();	None	This is a software power on reset (POR) of the eRIC module		eROS4
eRIC_GlobalInterruptEnable();	None	Enables all global interrupts		eROS4
eRIC_GlobalInterruptDisable();	None	Disables all global interrupts		eROS4
eRIC_FlashProgram_Mode(Mode);	Where Mode = 0, jumps into bootloader and it needs flashing new app code to come of it.			eROS4
eRIC_LPM_Level0();	None	Turn off only MCLCK, and enter sleep mode		eROS4
eRIC_LPM_ExitLevel0();	None	Exits the sleep mode and LPMlevel0 and continues the program from where it went into sleep before		eROS4
eRIC_LPM_Level1();	None	Turns off MCLCK and SMCLCK and enter sleep mode		eROS4
eRIC_LPM_ExitLevel1();	None	Exits the sleep mode and LPMLevel1 and continues the program from where it went into sleep before		eROS4
eRIC_LPM_Level2();	None	Turns off all clocks and enters sleep mode		eROS4
eRIC_LPM_ExitLevel2();	None	Exits the sleep mode and LPMLevel2 and continues the program from where it went into sleep before		eROS4
eRIC_Stringcopy(*destination,*source,count);	Where destination is the address of destination string, source is the address of the source and count is no of bytes to be copies	Copies one string into another		eROS4

eRIC_Stringcompare(*a,*b,count);	Where a is the address of first string and b is address of second string and count is no of bytes to be compared	Compares two strings and return 0 if they are same.		eROS4
eRIC_Stringlength(*string);	Where string is the address of the string for which length needs finding	Returns no of bytes of the string.		eROS4
eRIC_Sprintf(*buff,*string,val);	Where buff is the address where formatted string is stored, string is format,val is the data to be formatted. Formats available are: %d,%d which prints decimal data with sign.	Returns the length of the formatted string		eROS4
eRIC_Print(*txt);	Where txt is the address of the string which prints on to Uart			eROS4
eRIC_Ascii2Hex(val);	Where val can be any Ascii char between 0-9,a-f,A-F.	Converts Ascii to Hex and returns hex val.		eROS4
eRIC_Int2Ascii(val);	Where val can be any Int 0-F	Converts Int to Ascii character		eROS4

eRIC Pins Functionality and Usage

eRIC has 24 Pins, of which Pin 6 is Vcc, Pin 7 is ground, Pin 8, 9 are used by JTAG only, Pin 23 Ground and Pin 24 Antenna.

18 Pins are therefore available for general purpose (I/O), secondary mapping function and interrupts.

Pins 1-5 and Pin 22 are also Analogue pins. Any ADC or Analogue function should therefore only be connected to these pins. These pins also have interrupts.

Please note 'X' can be any of the 18 pin numbers below and 'Y' can be only be Pins 1-5 and Pin 22.

Functions	Parameters	Description	Notes	OS
PinX_PullUpEnable(); PinX_PullDownEnable(); PinX_PullUpDisable();	None	Enable or Disable the Pull up/pull down resistor on pin	Where 'X' is one of the 18 available pins	

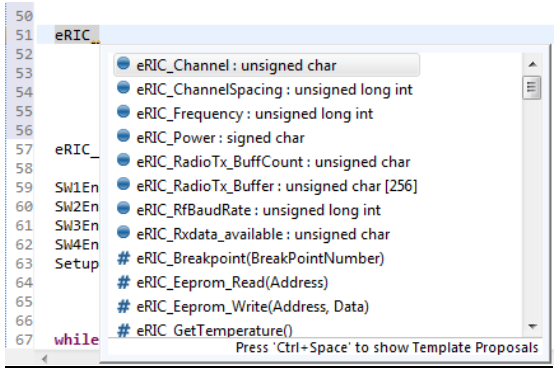
PinX_SetAsOutput();
PinX_SetHigh();
PinX_SetLow();
	None	Set Pin as output and then set high or low;
		E.g. Pin1_SetAsOutput();
		Pin1_SetHigh();//logic 1
		Pin1_SetLow();//logic 0
			Where 'X' is one of the 18 available pins	

PinX_SetAsInput();
PinX_Read();
	None	Set Pin as input and read the state of the input on the pin
		E.g. Pin1_SetAsInput();
		If(Pin1_Read() == 1)
		{
		//Do something;
		}
			Where 'X' is one of the 18 available pins	

PinX_HighDriveStrength_15mA();	None	Set Pin high and low maximum drive current (mA source/sink) of each pin	Where 'X' is one of the 18 available pins	

PinX_LowDriveStrength_6mA();		individually Default = Low 6mA		
PinX_InterruptLow2High(); PinX_InterruptHigh2Low(); PinX_InterruptDirection();	None	Pin Interrupt Edge Direction Set Interrupt Flag on Pin Low to High Set Interrupt Flag on Pin High to Low Read Interrupt Edge selection	Pins1-5 and Pin22 only can use this	
PinX_InterruptEnable(); PinX_InterruptDisable(); PinX_InterruptEnabled();	None	Pin Change Interrupt Enable/Disable Enable Pin Interrupt, only use when using Interrupt Service Routine Disable Pin Interrupt Read Interrupt Enabled status	Pins1-5 and Pin22 only can use this	
PinX_SetInterruptFlag(); PinX_ClearInterruptFlag(); PinX_HasIntterupted();	None	Set Interrupt Flag on pin Reset Interrupt flag Test if Pin has changed	Pins1-5 and Pin22 only can use this	
PinX_FunctionIO();	None	Maps the pin as general I/O	Where 'X' is one of the 18 available pins	
PinX_FunctionNone();	None	Nothing is mapped to the pin	Where 'X' is one of the 18 available pins	
PinX_FunctionAclk();	None	Maps the pin to Aclk	Where 'X' is one of the 18 available pins	
PinX_FunctionMclk();	None	Maps the pin to Mclk	Where 'X' is one of the 18 available pins	
PinX_FunctionSmclk();	None	Maps the pin to Smclk	Where 'X' is one of the 18 available pins	
PinX_FunctionTA0clkIN();	None	Maps the pin to Timer 0	Where 'X' is one of the 18 available pins	
PinX_FunctionUartTxOUT();	None	Maps the pin to Uart transmit	Where 'X' is one of the 18 available pins	
PinX_FunctionUartRxD();	None	Maps the pin to Uart Receive	Where 'X' is one of the 18 available pins	
PinX_FunctionSPI_MI();	None	Maps the pin to SPI Master in	Where 'X' is one of the 18 available pins	
PinX_FunctionSPI_MO();	None	Maps the pin to SPI Master out	Where 'X' is one of the 18 available pins	
PinX_FunctionSPI_SI();	None	Maps the pin to SPI Slave in	Where 'X' is one of the 18 available pins	
PinX_FunctionSPI_SO();	None	Maps the pin to SPI Slave out	Where 'X' is one of the 18 available pins	
PinX_FunctionSPI_SCLK();	None	Maps the pin to SPI Clock out	Where 'X' is one of the 18 available pins	
PinX_FunctionSPI_STE();	None	Maps the pin to SPI Transmit enable	Where 'X' is one of the 18 available pins	
PinX_FunctionI2C_SCI();	None	Maps the pin to i2c clock	Where 'X' is one of the 18 available pins	
PinX_FunctionI2C_SDA();	None	Maps the pin to i2c data	Where 'X' is one of the 18 available pins	
PinX_FunctionA2D();	None	Maps the pin to Analog function	Pins1-5 and Pin22 only can use this	
PinX_SetAsCarrierDetect();	None	Sets the pin as carrier detect	Where 'X' is one of the 18 available pins	eROS4
PinX_SetAsAsyncRxData();	None	Sets the pin as receiver output pin in Asynchronous mode	Where 'X' is one of the 18 available pins	eROS4
PinX_SetAsAsyncTxData();	None	Sets the pin as transmitter input pin in Asynchronous mode	Where 'X' is one of the 18 available pins	eROS4
PinX_FunctionUartABusy()	None	This is used to set a Uart busy pin. Used for handshaking (controlled by Radio functions)		eROS4

Note: Code Composer Studio uses 'autocomplete'. To complete a command or variable, press ctrl+space after first character.



Further information on programming is provided in the eRIC Tutorials 1, 2 and 3.

Sample Application Code using some of the above functions

```
#include<cc430f5137.h>
#include "eRIC.h"
#include <stdio.h>
#include <string.h>
/*
 * main.c
* This program code reads ADC value on Pin22 and also reads temperature around the module and sends over air through RF at 459600000hz frequency
* with 9dbm RF power continuously every 2 seconds
 */
int main(void)
 {

   eRIC_WDT_Stop();     //stops the watch dog timer
   eROS_Initialise(434000000);// initialse eros with 434000000
   eRIC_Rx_Enable();          //Enable radio receive mode,if not enabled can save power
   eRIC_SetCpuFrequency(4000000); //Cpu clock speed is set to 4Mhz

  eRIC_ChannelSpacing = 200000;    //Channel spacing is 200khz
  eRIC_Channel  = 128;          //Channel changed to 128        , Now frequency would be (434000000+(128*200000)) = 459600000Hz
  eRIC_RfBaudRate = 38400;         // Over air baud rate changed to 38400

  eRIC_Power = 9;                    //power is set to 9

  eRIC_RadioUpdate();               //Makes all above Radio changes in eROS
   volatile long AdcResult = 0;
   volatile float temperature = 0;        //Decalred as float because temperature will be in points
   LED4Enable();                    //Led4 which is pin19 is set as output
   Pin22_FunctionA2D();      //Map pin 22 to ADC
   while(1)
    {
      LED4_Set();               //Led4 which is pin19 is turned on

      AdcResult = eRIC_ReadAdc(22,0);//To read ADC value on pin 22 at reference 1.5v (0)

      temperature= eRIC_GetTemperature();         // to read temperature

      eRIC_RadioTx_Buffer[eRIC_RadioTx_BuffCount++] = AdcResult>>8;          // Fills Adc value into Rf transmit buffer
      eRIC_RadioTx_Buffer[eRIC_RadioTx_BuffCount++] = AdcResult;             // Fills Adc value into Rf transmit buffer

      eRIC_RadioTx_Buffer[eRIC_RadioTx_BuffCount++] = temperature;       // Fills temperature real value into Rf transmit buffer
```

```
    eRIC_RadioTx_Buffer[eRIC_RadioTx_BuffCount++] = (temperature*100);          // Fills temperature decimal value into Rf transmit buffer


    eRIC_RfSenddata();                      // sends Adc value and temperature in four bytes over air through RF


    eRIC_Delay(1000);   //1 sec delay 1000ms
    LED4_Clear();          //Led4 which is pin19 is turned Off

    eRIC_Delay(1000);   //1 sec delay 1000ms
  }

}
```