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.

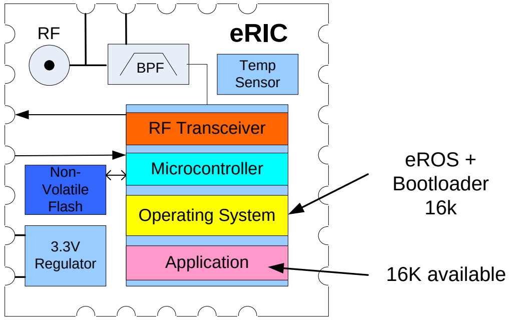


Figure 1 eRIC Transceiver Block Diagram

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 RF 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 easyRadio Companion Vx.x.x setup program on the USB stick (or download from www.lprs.co.uk) and double click to install on the PC.
- Download latest eRICxeasyRadioVx.x .zip file from www.lprs.cp.uk
- 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 CCS Project'. (Figure 2)
- Select 'Archive' file and browse to eRICxeasyRadioVx.x.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 easyRadio Comapnion Vx.x.x software tool.

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

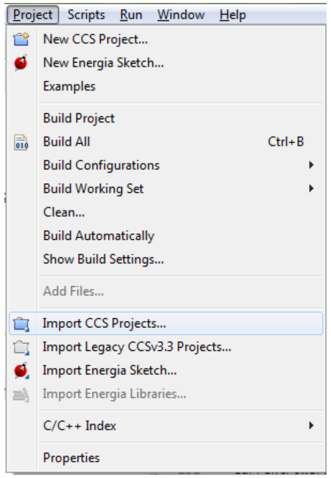


Figure 2 Import CCS Projec

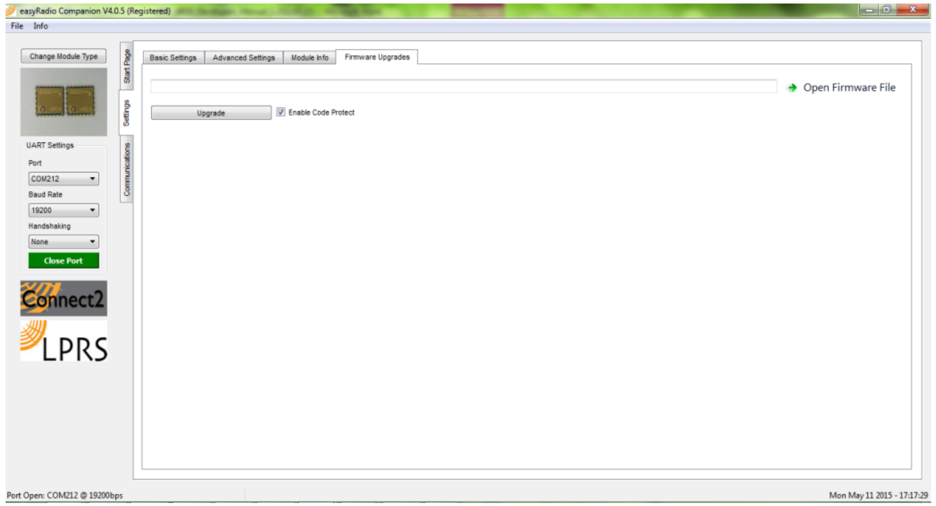


Figure 3 easyRadio Companion V4.0.5

easyRadio Companion Vx.x.x

- 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 **easyRadio Companion Vx.x.x**
- Switch the Development Board 'On' and momentarily press the 'Reset' push button switch.
- Select eRIC module and click OK.
- Select the baudrate (19200 default) and click Open Port.

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_SetUncalibratedModulationCarrierOn();	None	Sets the Modulated Carrier On without calibrating frequency		eRIC V1.4
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_Buffer[];	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 data		eROS 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 0-8 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
		8 0% Complete turn off radio receiver and puts radio in idle and sleep .
			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 4.1

eRIC_TxPowerLevel	Char . only 0-8 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.
		
		When eRIC_RxPowerLevel is 8 and eRIC_TxPowerLevel is 8 then radio is put in idle and sleep
			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 4.1

eRIC_AES_Key[];	Char 17 bytes. First byte being the mode and rest 16 bytes being the key	This need filling accordingly before calling eRIC_AES_ChangeKey(); or eRIC_AES_SetKey();		eROS 4.1
eRIC_AES_Data[];	Char 16 bytes.	This is used before or after eRIC_AES_Run();		eROS 4.1
* 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_SetAdcPin(eRICPinNumber);	eRICpinNumber can only be 1,2,3,4,5 and 22	Assigns ADC functionality to the Pin passed in		eRIC V1.4
eRIC_SetAdcRefVoltage(eRIC_ReferenceVoltage,RefOut_OnorOff_Pin22);	There are 3 eRIC_ReferenceVoltage : 1)eRICADCRef_1_5v 2)eRICADCRef_2_0v 3)eRICADCRef_2_5v RefOut_OnorOff_Pin22 = 0 or 1	The selected reference voltage can be output on Pin22 when RefOut_OnorOff_pin22 =1		eRIC V1.4

eRIC_ReadAdc();
Was
	None
	Was
		Reads 12 bit Digital Adc value on eRIC pin passed in and with Reference voltage.
		For eg:
		eRIC_SetAdcPin(1);
		eRIC_SetAdcRefVoltage(eRICADCRef_1_5v,1);
		int Temp = eRIC_ReadAdc();
		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. The reference voltage can also be seen on Pin22 as RefOut_OnorOff_Pin22 is set as 1.
			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 V1.4

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();	None	UartA Rx interrupt can be Enabled and Disabled	Interrupts can be handled using Pragma	

eRIC_UartARxInteruptEnable();			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(); eRIC_UartATxInteruptEnable();	None	UartA tx interrupt can be enabled or disabled		
eRIC_UartATxIsEnabled();	None	Test to see if UartA tx interrupt enabled		
eRIC_UartA_SyncMode();	None	Synchronous mode is selected for UartA		eRIC V1.4
eRIC_UartA_AsyncMode();	None	ASynchronous mode is selected for UartA		eRIC V1.4
eRIC_UartA_2StopBits();	None	Enables two stop bits		eRIC V1.4
eRIC_UartA_1StopBit();	None	Enables one stop bit		eRIC V1.4
eRIC_UartA_7Bit();	None	Enables 7Bit packet		eRIC V1.4
eRIC_UartA_8Bit();	None	Enables 8Bit packet		eRIC V1.4
eRIC_UartA_MsbFirst();	None	MSB first is enabled in UartA transmit or receive		eRIC V1.4
eRIC_UartA_LsbFirst();	None	LSB first is enabled in UartA transmt or receive		eRIC V1.4
eRIC_UartA_EvenParity();	None	EvenParity mode is selected		eRIC V1.4
eRIC_UartA_OddParity();	None	Odd parity mode is selected		eRIC V1.4
eRIC_UartA_ParityEnable();	None	Parity is enabled		eRIC V1.4
eRIC_UartA_ParityDisable();	None	Parity is Disabled		eRIC V1.4
eRIC_UartATxSetInterruptFlag();	None	Set the Tx interrupt flag		eRIC V1.5
eRIC_UartATxClearInterruptFlag();	None	Clears the Tx interrupt flag		eRIC V1.5
eRIC_UartATxHasInterrupted();	None	Is the Tx interrupt triggered, or flag set?		eRIC

				V1.5
eRIC_UartARxSetInterruptFlag();	None	Set the Rx interrupt flag		eRIC V1.5
eRIC_UartARxClearInterruptFlag();	None	Clears the Rx interrupt flag		eRIC V1.5
eRIC_UartARxHasInterrupted();	None	Is the Rx interrupt triggered, or flag set?		eRIC V1.5
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. Similarly SpiB can also be configured by replacing eRIC_SpiB in place of eRIC_SpiA	
eRIC_SpiARead(Data);	Data is dummy data to read SPIdata	Gets Spi Data after sending a dummy byte	Similarly SpiB can also be configured	
eRIC_SpiAWrite(Data);	Data is any unsigned char	Send a byte of data through SPI	Similarly SpiB can also be configured	
eRIC_SpiATxInteruptEnable();	None	Enables the SpiA Tx interrupt	Similarly SpiB can also be configured	
eRIC_SpiATxInteruptDisable();	None	Disables the SpiA Tx interrupt	Similarly SpiB can also be configured	
eRIC_SpiATxIsEnabled();	None	Returns Non Zero if Tx is enabled	Similarly SpiB can also be configured	
eRIC_SpiATxBufferIsEmpty();	None	Returns Non Zero if Tx buffer is empty	Similarly SpiB can also be configured	
eRIC_SpiATxBufferIsBusy();	None	Returns Non zero if Tx buffer is busy	Similarly SpiB can also be configured	
eRIC_SpiARxInteruptEnable();	None	Enables Rx interrupt	Similarly SpiB can also be configured	
eRIC_SpiARxInteruptDisable();	None	Disables Rx interrupt	Similarly SpiB can also be configured	
eRIC_SpiARxIsEnabled();	None	Returns Non Zero if Rx is enabled	Similarly SpiB can also be configured	
eRIC_SpiARxBufferIsEmpty();	None	Returns Non Zero if Rx is empty	Similarly SpiB can also be configured	
eRIC_SpiARxBufferIsBusy();	None	Return Non Zero if Rx is busy	Similarly SpiB can also be configured	
eRIC_SpiASendByte(Data);	Data can be one byte of data	Send data if Tx buffer is not busy	Similarly SpiB can also be configured	
eRIC_SpiAReceiveByte();	None	Returns one byte of data if SpiA receives it	Similarly SpiB can also be configured	
eRIC_SpiASyncMode();	None	Synchronous mode is selected for Spi communications	Similarly SpiB can also be configured	
eRIC_SpiA_ASyncMode();	None	Asynchonous mode is selected for Spi communications	Similarly SpiB can also be configured	
eRIC_SpiA_3PinMode();	None	3pin mode is selected for Spi communications	Similarly SpiB can also be configured	
eRIC_SpiA_4Pin_SteActiveHigh();	None		Similarly SpiB can also be configured	
eRIC_SpiA_4Pin_SteActiveLow();	None		Similarly SpiB can also be configured	
eRIC_SpiA_MasterMode();	None	The device is set as Master for I2C	Similarly SpiB can also be configured	
eRIC_SpiA_SlaveMode();	None	The device is set as Slave for I2C	Similarly SpiB can also be configured	
eRIC_SpiA_7Bit();	None	7Bit data packet is enabled	Similarly SpiB can also be configured	
eRIC_SpiA_8Bit();	None	8Bit data packet is enabled	Similarly SpiB can also be configured	
eRIC_SpiA_MsbFirst();	None	MSb as first bit of data is enabled	Similarly SpiB can also be configured	
eRIC_SpiA_LsbFirst();	None	LSB as first bit of data is enabled	Similarly SpiB can also be configured	
eRIC_SpiA_ClkIdleHigh();	None		Similarly SpiB can also be configured	
eRIC_SpiA_ClkIdleLow();	None		Similarly SpiB can also be configured	

eRIC_SpiA_DataOn1stUclkEdge();	None	Data on first clock edge	Similarly SpiB can also be configured	
eRIC_SpiA_DataOn2ndUclkEdge();	None	Data on second clock edge	Similarly SpiB can also be configured	
eRIC_I2CB_Initialise(I2CClock,MasterorSlave,Address);	I2CClock is a clock frequency in Hertz. MasterorSlave can be 0 or 1. Address is selected as Slave address or Own address automatically based on Master or Slave selection.	For Eg:eRIC_I2CClock(100000,1,0x0A); Masteror Slave is 1 if Master or 0 if Slave		eRIC V1.4
eRIC_I2CB_TxInteruptEnable();	None	Enables the I2CB Tx interrupt		eRIC V1.4
eRIC_I2CB_TxInterruptDisable();	None	Disables the I2CB Tx interrupt		eRIC V1.4
eRIC_I2CB_TxIsEnabled();	None	Returns Non Zero if Tx is enabled		eRIC V1.4
eRIC_I2CB_TxBufferIsEmpty();	None	Returns Non Zero if Tx buffer is empty		eRIC V1.4
eRIC_I2CB_TxBufferIsBusy();	None	Returns Non zero if Tx buffer is busy		eRIC V1.4
eRIC_I2CB_RxInteruptDisable();	None	Disables Rx interrupt		eRIC V1.4
eRIC_I2CB_RxInteruptEnable();	None	Enables Rx interrupt		eRIC V1.4
eRIC_I2CB_RxIsEnabled();	None	Returns Non Zero if Rx is enabled		eRIC V1.4
eRIC_I2CB_RxBufferIsEmpty();	None	Returns Non Zero if Rx is empty		eRIC V1.4
eRIC_I2CB_RxBufferIsBusy();	None	Return Non Zero if Rx is busy		eRIC V1.4
eRIC_I2CB_SendByte(Data);	Data can be one byte of data	Send data if Tx buffer is not busy		eRIC V1.4
eRIC_I2CB_ReceiveByte();	None	Returns one byte of data if I2C receives it		eRIC V1.4
eRIC_I2CB_SyncMode();	None	Synchronous mode is selected for I2C communications		eRIC V1.4
eRIC_I2CB_ASynchMode();	None	Asynchonous mode is selected for I2C communications		eRIC V1.4
eRIC_I2CB_MasterMode();	None	The device is set as Master for I2C		eRIC V1.4
eRIC_I2CB_SlaveMode();	None	The device is set as Slave for I2C		eRIC V1.4
eRIC_I2CB_MultiMasterMode();	None	Multimaster mode is enabled for I2C communications		eRIC

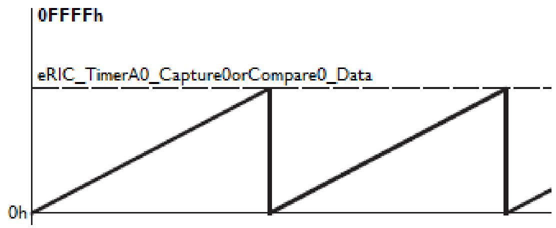
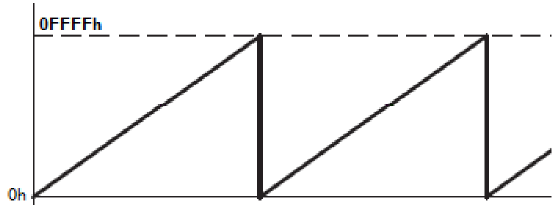
				V1.4
eRIC_I2CB_SingleMasterMode();	None	Single Master mode is enabled for I2C communications		eRIC V1.4
eRIC_I2CB_10BitSlaveAddress();	None	10Bit slave address is enabled		eRIC V1.4
eRIC_I2CB_7BitSlaveAddress();	None	7Bit slave address is enabled		eRIC V1.4
eRIC_I2CB_10BitOwnAddress();	None	10Bit Own address is enabled		eRIC V1.4
eRIC_I2CB_7BitOwnAddress();	None	7Bit Own address is enabled		eRIC V1.4
eRIC_I2CB_SoftwareResetEnable();	None	Software reset is enabled for I2C		eRIC V1.4
eRIC_I2CB_SoftwareResetDisable();	None	Software reset is disabled for I2C		eRIC V1.4
eRIC_I2CB_IsStartActive();	None	Returns Non zero if start condition is On		eRIC V1.4
eRIC_I2CB_Start();	None	Starts the I2C communication		eRIC V1.4
eRIC_I2CB_Stop();	None	Stops the I2C communication		eRIC V1.4
eRIC_I2CB_IsStopConditionOn();	None	Returns Non zero if stop condition is still on		eRIC V1.4
eRIC_I2CB_NackOn();	None	Slave acknowledging with Nack is turned on		eRIC V1.4
eRIC_I2CB_NackOff();	None	Slave acknowledging with Nack id turned off		eRIC V1.4
eRIC_I2CB_AsTransmitter();	None	Sets the device as transmitter		eRIC V1.4
eRIC_I2CB_AsReceiver();	None	Sets the device as receiver		eRIC V1.4
eRIC_I2CB_IsSCL_Low();	None	Returns Non zero if SCL line is pulled low		eRIC V1.4
eRIC_I2CB_IsBusBusy();	None	Returns Non zero if I2C bus is busy		eRIC V1.4
eRIC_I2CB_OwnAddress	None	Can be used to assign address For Eg: eRIC_I2CB_OwnAddress = 0x0A;		eRIC V1.4
eRIC_I2CB_SlaveAddress	None	Can be used to assign address For Eg: eRIC_I2CB_SlaveAddress = 0x0A;		eRIC V1.4
eRIC_I2CB_WaitforACK();	None	Returns Non zero if ACK is not received or start condition is still on		eRIC V1.4

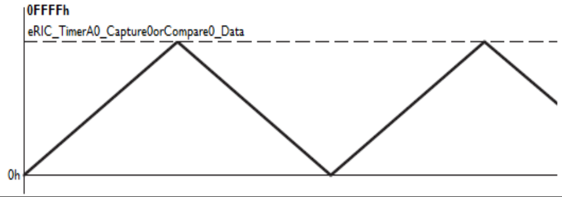
eRIC_I2CB_NackReceived();	None	Returns Non zero if Nack is received		eRIC V1.4
eRIC_Eeprom_Read(Address);	Address can range from 0-255. As EEprom is only 256 bytes size.	Reads EEprom data from the address passed in		
eRIC_Eeprom_Write(Address,Data) ;	Address can range from 0-255. As EEprom is only 256 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_ChangeKey();	None .Needs eRIC_AES_Key[17] filling first	Fill eRIC_AES_Key[0-17], first byte being mode(0 encryption,1decryption) and rest 16 Bytes being Key, before calling eRIC_AES_ChangeKey();.
		This will change AES key and also encrypts key using another discrete key and store the encrypted key in discrete location. This will also set key at the end of this function.
			For example:
			To encrypt data:
			Setting key first:
			char i =0;
			while(i<17) //0 mode as encryption and 1-
			//16 as key
			eRIC_AES_Key[i++];
			eRIC_AES_ChangeKey();
			i =0;
			while(i<16)
			eRIC_AES_Data[i++];
			eRIC_AES_Run();
			//After this encrypted16 bytes of data is
			//available in eRIC_AES_Data[16]
			
			To decrypt data:
			eRIC_AES_Key[0]=1; //decryption mode
			eRIC_AES_SetKey();//as key is same, don't
			//change key
			//store encrypted data in
			eRIC_AES_Data[16];
			
			//and call
			eRIC_AES_Run();
			//Decrypted 16 bytes of data will be
			//available in eRIC_AES_Data[16]
				eROS 4.1

eRIC_AES_SetKey();	None..Needs eRIC_AES_Key[17] filling first	Fill eRIC_AES_Key[0-17], first byte being mode(0 encryption,1decryption) and rest 16 Bytes being Key, before calling eRIC_AES_SetKey(); This will set the AES key with encryption or decryption.		eROS 4.1
eRIC_AES_Run();	None. .Needs eRIC_AES_Data[16] filling first	To encrypt data fill eRIC_AES_Data[16] and call eRIC_AES_Run();. To decrypt data call eRIC_AES_Run(); and data is available in eRIC_AES_Data[16].		eROS 4.1
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 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	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 interrupt vector if handled.	eROS4
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	eROS4

			can be monitored in code	
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_RadioSleep();	None	Sends the radio into Idle and sleep state		eRIC V1.4
eRIC_TimerA0_Setup(CompleteSetup);	CompleteSetup can be addition of clocksourse,clockdivider,Mode,interrupt etc eRICTimer_Div_1 eRICTimer_Div_2 eRICTimer_Div_4 eRICTimer_Div_8 eRICTimer_Cs_32k eRICTimer_Cs_Cpu eRICTimer_Stop eRICTimer_UpMode eRICTimer_ContinousMode eRICTimer_UpdownMode eRICTimer_Reset eRICTimer_InterruptEnable eRICTimer_InterruptDisable	TimerA0 can be setup configured in one function. For example: eRIC_TimerA0_setup(eRICTimer_Cs_Cpu+ eRICTimer_Div_1+ eRICTimer_UpMode+ eRICTimer_Reset+);		eRIC V1.5

eRIC_TimerA0_Cs(Clocksource);	Clocksource can be either of two below: eRICTimer_Cs_32k eRICTimer_Cs_Cpu	This will choose clocksource for TimerA0.		eRIC V1.4
eRIC_TimerA0_ClockDIvider(Clock divider);	Clockdivider can be any of the following: eRICTimer_Div_1 eRICTimer_Div_2 eRICTimer_Div_4 eRICTimer_Div_8	This will further divide the clock by 1,2,4,8		eRIC V1.4
eRIC_TimerA0_Stop();	None	Halts the timer		eRIC V1.4
eRIC_TimerA0_UpMode();	None	This is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of eRIC_TimerA0_Capture0orCompare0_Data, which defines the period. The number of timer counts in the period is eRIC_TimerA0_Capture0orCompare0_Data + 1. When the timer value equals eRIC_TimerA0_Capture0orCompare0_Data, the timer restarts counting from zero. 		eRIC V1.4
eRIC_TimerA0_ContinousMode();	None	In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero. 		eRIC V1.4
eRIC_TimerA0_UpdownMode();	None	This mode is used if the timer period must be different		eRIC

		from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of eRIC_TimerA0_Capture0orCompare0_Data and back down to zero. The period is twice the value in eRIC_TimerA0_Capture0orCompare0_Data. 		V1.4
eRIC_TimerA0_Reset();	None	Resets the timer		eRIC V1.4
eRIC_TimerA0_InterruptEnable();	None	This enabled the TImerA0 interrupt		eRIC V1.4
eRIC_TimerA0_interruptDisable();	None	This disables the TImerA0 interrupt		eRIC V1.4
eRIC_TimerA0_IsInterruptEnabled();	None	This returns non zero if interrupt flag is set and interrupt has been triggered		eRIC V1.4
eRIC_TimerA0_InterruptFlag_set();	None	This sets the TimerA0 interrupt flag		eRIC V1.4
eRIC_TimerA0_InterruptFlag_Clear( );	None	This clears the TimerA0 interrupt flag		eRIC V1.4
eRIC_TimerA0_HasInterrupted();	None	This returns non zero if interrupt flag is set and interrupt has been triggered		eRIC V1.4
eRIC_TimerA0_Count_Read();	None	This returns the 16bit count of the TimerA0		eRIC V1.4
eRIC_TimerA0_Count_Set(intcount number);	None	This will write to count register of TImerA0		eRIC V1.4
eRIC_TimerA0_Capture0orCompare0Setup(CompleteSetup);	CompleteSetup is used to select Capture orcompare mode, Capture edge,interrupt and compareoutputmode. The following can be used: eRICTimer_CaptureMode eRICTimer_CompareMode eRICTimer_CaptureNothing eRICTImer_CaptureOnRising eRICTimer_CaptureOnFalling	This is used to setup capture compare modes For Eg: eRIC_TimerA0_Capture0orCompare0Setup(eRICTimer_CompareMode+ eRICTimer_CaptureNothing+ eRICTimer_OutputMode_Toggle_Set+ eRICTimer_CCInterruptDisable);		eRIC V1.4

	eRICTimer_CaptureOnBothFallRise eRICTimer_OutputMode_Outputonly eRICTimer_OutputMode_Set eRICTimer_OutputMode_Toggle_Reset eRICTimer_OutputMode_Set_Reset eRICTimer_OutputMode_Toggle eRICTimer_OutputMode_Reset eRICTimer_OutputMode_Toggle_Set eRICTimer_OutputMode_Reset_Set eRICTimer_CCInterruptEnable eRICTimer_CCInterruptDisable			
eRIC_TimerA0_CaptureXorCompareX_Data	X can be 0,1,2,3,4			eRIC V1.5
eRIC_TimerA0_CaptureXorComapreXInterruptEnable();	X can be 0,1,2,3,4	Enable capture or compare interrupt		eRIC V1.5
eRIC_TimerA0_CaptureXorComapreXInterruptDisable();	X can be 0,1,2,3,4	Disables capture or compare interrupt		eRIC V1.5
eRIC_PWM_Setup(ClockSource,ClockDivider,UpDownContinousMode, Period);	The following can be used as paramters eRICPWM_Cs_32k eRICPWM_Cs_CPU eRICPWM_DIV_1 eRICPWM_DIV_2 eRICPWM_DIV_4 eRICPWM_DIV_8 eRICPWM_Stop eRICPWM _UpMode eRICPWM_ContinousMode eRICPWM_UpdownMode	For Eg; eRIC_PWM_Setup(eRICPWM_Cs_CPU,eRICPWM_DIV_1,eRICPWM_UpMode,255); //Set PWM with clock source, Clock divider, Period Mode and period		eRIC V1.4
eRIC_PWM_Cs(Clocksource);	Clocksource can be either of two below: eRICPWM_Cs_32k eRICPWM_Cs_CPU	This will choose clocksource for PWM		eRIC V1.4
eRIC_PWM_UpDownContinousMode(UpDownContinousMode);	UpDownContinousMode can be one of the following eRICPWM_Stop eRICPWM _UpMode eRICPWM_ContinousMode eRICPWM_UpdownMode	This will set different modes for PWM. Please refer TimerA0 modes for detail explanantion		eRIC V1.4

eRIC_PWM_ClockDivider(Clockdivider);	Clockdivider can be one of the following eRICPWM_DIV_1 eRICPWM_DIV_2 eRICPWM_DIV_4 eRICPWM_DIV_8	Clockdivider is used to divide further the clock:		eRIC V1.4
eRIC_PWM_Reset();	None	Resets the PWM		eRIC V1.4

eRIC_PWM1_DutyCycle(DutyCycle, LogicOutput);
Similarly
eRIC_PWM2_DutyCycle(DutyCycle, LogicOutput);
eRIC_PWM3_DutyCycle(DutyCycle, LogicOutput);
eRIC_PWM4_DutyCycle(DutyCycle, LogicOutput);
	DutyCycle can be 16Bit number and LogicOutput can be one of the following:
	eRICPWM_OutputMode_Set
	eRICPWM_OutputMode_Toggle_Reset
	eRICPWM_OutputMode_Set_Reset
	eRICPWM_OutputMode_Toggle
	eRICPWM_OutputMode_Reset
	eRICPWM_OutputMode_Toggle_Set
	eRICPWM_OutputMode_Reset_Set
		This is used to set the DutyCycle of each PWM and also the mode of when to set reset or toggle.		eRIC V1.4

eRIC_PWM_Period(Period);	Period can be any 16Bit number	This is used to set the period of the PWM		eRIC V1.4
eRIC_CRC_Initialise(Data);	Data is any 8Bit data	CRC module is initialised with first byte of data		eRIC V1.4
eRIC_CRC_FirstByte(Data);	Data is any 8Bit data	First Byte should be send to CRC module using this		eRIC V1.4
eRIC_CRC_NextByte(Data);	Data is any 8Bit data	The following Bytes of data can be sent using this		eRIC V1.4
eRIC_CRC_Result();	None	The result of the CRC is returned using this function		eRIC V1.4
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_SetGDOSIgnal(SelectSignalType);	SelectSignalType can be one of the below signals: eRICGDO_SyncWordSignal eRICGDO_PacketReceivedWithCRCOKSignal eRICGDO_CarrierSenseSignal eRICGDO_RadioTransmitSignal eRICGDO_RadioReceiverSIgnal eRICGDO_RadioRSSIValidSignal eRICGDO_RadioRXTimeoutSignal eRICGDO_RadioClock32Signal Other signals can be found on Page 712 of SLAU259 datasheet	This function is used to assign one of the signals listed beside to GDO. Once a signal is assigned, a Pin needs to be mapped to this GDO using PinX_FunctionGDOSignal(); . Once this is done, a signal is checked on pin.	For example: To check when the receiver is turning ON and OFF or when the receiver is receving data, we need to do the following steps: eRIC_SetGDOSignal(eRICGDO_RadioReceiverSIgnal); Pin19_FunctionGDOSIgnal(); Now Pin19 will stay High when the Receiver is OFF and will go low whenever Radio receiver is ON or receives anything.	eROS 4.1 eRIC V1.5
eRIC_SetPMMVCoreLevel(Level)	Level can be 0,1,2,3,4. When 4 is used, PMM is turned off.	This is to set the Voltage level for power management module. This needs to be set based on Cpu frequency, and radio being used. Genereally it is set to level 3.	Refer SLAU259 datasheet for more details	eROS 4.1 eRIC V1.5
eRIC_RadioRegWrite(RegAddress,Data);	RegAddress is register address. Data is the value that can go into register address		Refer SLAU259 datasheet for more details	eROS 4.1 eRIC V1.5
eRIC_RadioRegRead(RegAddress);	RegAddress is register address.		Refer SLAU259 datasheet for more details	eROS 4.1 eRIC V1.5

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();	None	Enable or Disable the Pull up/pull down resistor on pin	Where 'X' is one of the 18 available pins	

PinX_PullDownEnable(); PinX_PullUpDisable();				

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_Toggle();	None	Toggles Pin output state from 1 to 0 or 0 to 1 Eg:Pin1_Toggle();//Toggles Pin1 Output state.	Where 'X' is one of the 18 available pins	eRIC V1.4

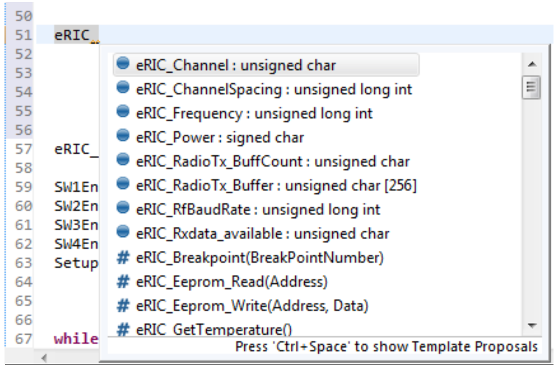
PinX_SetAsInput();
PinX_Read();
	None	Set Pin as input and read the state of the input on the pin
		E.g. Pin1_SetAsInput();
		If(Pin1_Read() == 1)
		{
		//Do something;
		}
			Where 'X' is one of the 18 available pins	

PinX_HighDriveStrength_15mA(); PinX_LowDriveStrength_6mA();	None	Set Pin high and low maximum drive current (mA source/sink) of each pin individually Default = Low 6mA	Where 'X' is one of the 18 available pins	
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_FunctionUartATxOUT();	None	Maps the pin to Uart transmit	Where 'X' is one of the 18 available pins	
PinX_FunctionUartARxD();	None	Maps the pin to Uart Receive	Where 'X' is one of the 18 available pins	
PinX_FunctionSPIA_MI();	None	Maps the pin to SPIA Master in. Similarly SPIB can also be mapped	Where 'X' is one of the 18 available pins	
PinX_FunctionSPIA_MO();	None	Maps the pin to SPIA Master out. Similarly SPIB can also be mapped	Where 'X' is one of the 18 available pins	
PinX_FunctionSPIA_SI();	None	Maps the pin to SPIA Slave in. Similarly SPIB can also be mapped	Where 'X' is one of the 18 available pins	
PinX_FunctionSPIA_SO();	None	Maps the pin to SPIA Slave out. Similarly SPIB can also be mapped	Where 'X' is one of the 18 available pins	

PinX_FunctionSPIA_SCLK();	None	Maps the pin to SPIA Clock out. Similarly SPIB can also be mapped	Where 'X' is one of the 18 available pins	
PinX_FunctionSPIA_STE();	None	Maps the pin to SPIA Transmit enable. Similarly SPIB can also be mapped	Where 'X' is one of the 18 available pins	
PinX_FunctionI2CB_SCl();	None	Maps the pin to i2c clock	Where 'X' is one of the 18 available pins	
PinX_FunctionI2CB_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_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
PinX_FunctionTA0CompareOut0();	None	Sets the Pin as Timer0 CompareOutput	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CompareOut1();	None	Sets the Pin as Timer0 CompareOutput	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CompareOut2();	None	Sets the Pin as Timer0 CompareOutput	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CompareOut3();	None	Sets the Pin as Timer0 CompareOutput	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CompareOut4();	None	Sets the Pin as Timer0 CompareOutput	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionPWM1();	None	Sets the Pin as Pulsewidthmodulation 1	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionPWM2();	None	Sets the Pin as Pulsewidthmodulation 2	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionPWM3();	None	Sets the Pin as Pulsewidthmodulation 3	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionPWM4();	None	Sets the Pin as Pulsewidthmodulation 4	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CaptureIn0();	None	Sets the Pin as Timer0 CaptureInput0	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CaptureIn1();	None	Sets the Pin as Timer0 CaptureInput1	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CaptureIn2();	None	Sets the Pin as Timer0 CaptureInput2	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CaptureIn3();	None	Sets the Pin as Timer0 CaptureInput3	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionTA0CaptureIn4();	None	Sets the Pin as Timer0 CaptureInput4	Where 'X' is one of the 18 available pins	eRIC V1.4
PinX_FunctionGDOSignal();	None	Sets the pin as GDO signal which can be assigned to other signals using eRIC_SetGDOSignal(SelectSignalType);	Where 'X' is one of the 18 available pins	eRIC V1.5

				eROS V4.1

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
    eRIC_SetAdcPin(22);       //Sets ADC on Pin22 .Added in V1.4
   eRIC_SetAdcRefVoltage(eRICADCRef_1_5v,0);  //Sets ADC reference with 1.5v and Ref out is not selected . Added in V1.4
    while(1)
      {
       LED4_Set();              //Led4 which is pin19 is turned on

       //
        AdcResult = eRIC_ReadAdc();//To read ADC value on pin 22 at reference 1.5v (0) . Added in V1.4


       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
      }

}
```