Mirics Limited.
Software Defined Radio API
Applications
Revision History

Revision	Release Date:	Reason for Change:	Originator
1.0	26 May 2013	Pre-Release 0.0.1	APC
1.1	08 August 2013	Add reset function	APC
1.2	31 July 2014	Extended frequency range	APC
1.3	27 March 2015	Updated error codes & tidy up	APC
1.7	13 August 2015	Added New commands & update mir_sdr_SetDcMode to add missing modes	IMH
1.8	22 December 2015	DC/IQ compensation & new gain map	APC
1.8.1	15 January 2016	Tidy up text	APC
1.94	11th July 2016	Range of new functions	APC
1.97	18th September 2016	Added new AGC + DC offset schemes	APC

Contents

1 Introduction

This document provides a description of the Mirics Ltd Software Defined Radio API. This API provides a common interface to the Mirics USB bridge device (MSi2500) and the multi-standard tuner (MSi001).

2 API Data Types

The header file mir_sdr.h provides the definitions of the external data types provided by this API.

2.1 Error Code Enumerated Type

```
typedef enum
{
  mir_sdr_Success             = 0,
  mir_sdr_Fail                = 1,
  mir_sdr_InvalidParam        = 2,
  mir_sdr_OutOfRange          = 3,
  mir_sdr_GainUpdateError     = 4,
  mir_sdr_RfUpdateError       = 5,
  mir_sdr_FsUpdateError       = 6,
  mir_sdr_HwError             = 7,
  mir_sdr_AliasingError       = 8,
  mir_sdr_AlreadyInitialised  = 9,
  mir_sdr_NotInitialised      = 10
} mir_sdr_ErrT;
```

2.2 Band Width Enumerated Type

```
typedef enum
{
  mir_sdr_BW_0_200 = 200,
  mir_sdr_BW_0_300 = 300,
  mir_sdr_BW_0_600 = 600,
  mir_sdr_BW_1_536 = 1536,
  mir_sdr_BW_5_000 = 5000,
  mir_sdr_BW_6_000 = 6000,
  mir_sdr_BW_7_000 = 7000,
  mir_sdr_BW_8_000 = 8000
} mir_sdr_Bw_MHzT;
```

2.3 IF Enumerated Type

```
typedef enum
{
  mir_sdr_IF_Zero    = 0,
  mir_sdr_IF_0_450  = 450,
  mir_sdr_IF_1_620  = 1620,
  mir_sdr_IF_2_048  = 2048
} mir_sdr_If_kHzT;
```

2.4 Reinit Enumerated Type

Used to specify the reason for a reinitialise request - values should be or'd together if there are multiple reasons for the request.

```
typedef enum
{
  mir_sdr_CHANGE_NONE       = 0x00,
  mir_sdr_CHANGE_GR         = 0x01,
  mir_sdr_CHANGE_FS_FREQ    = 0x02,
  mir_sdr_CHANGE_RF_FREQ    = 0x04,
  mir_sdr_CHANGE_BW_TYPE    = 0x08,
  mir_sdr_CHANGE_IF_TYPE    = 0x10,
  mir_sdr_CHANGE_LO_MODE    = 0x20
} mir_sdr_ReasonForReinitT;
```

2.5 LO Mode Enumerated Type

Used to specify the up-converter LO frequency for AM for use between frequencies below 60MHz and between 250MHz and 420MHz.

```
typedef enum
{
  mir_sdr_LO_Undefined      = 0,
  mir_sdr_LO_Auto           = 1,     // LO is automatically selected to provide appropriate coverage
                                     // across all RF frequency ranges
  mir_sdr_LO_120MHz         = 2,     // LO is set to 120MHz (coverage gap between 370MHZ and 420MHz)
  mir_sdr_LO_144MHz         = 3,     // LO is set to 144MHz (coverage gaps between 250MHZ and 255MHz
                                     // additional between 400MHz and 420MHz)
  mir_sdr_LO_168MHz         = 4      // LO is set to 168MHz (coverage gap between 250MHZ and 265MHz)
} mir_sdr_LoModeT;
```

2.6 Band Enumerated Type

Used to specify the requested band.

```
typedef enum
{
  mir_sdr_BAND_AM_LO        = 0,     //  0.01 <= Freq <    12 MHz
  mir_sdr_BAND_AM_MID       = 1,     //    12 <= Freq <    30 MHz
  mir_sdr_BAND_AM_HI        = 2,     //    30 <= Freq <    60 MHz
  mir_sdr_BAND_VHF          = 3,     //    60 <= Freq <   120 MHz
  mir_sdr_BAND_3            = 4,     //   120 <= Freq <   250 MHz
  mir_sdr_BAND_X            = 5,     //   250 <= Freq <   420 MHz
  mir_sdr_BAND_4_5          = 6,     //   420 <= Freq <  1000 MHz
  mir_sdr_BAND_L            = 7      //  1000 <= Freq <  2000 MHz
} mir_sdr_BandT;
```

2.7 AGC Control Enumerated Type

Used to specify AGC control method.

```
typedef enum
{
  mir_sdr_AGC_DISABLE       = 0,
  mir_sdr_AGC_100HZ         = 1,
  mir_sdr_AGC_50HZ          = 2,
  mir_sdr_AGC_5HZ           = 3
} mir_sdr_AgcControlT;
```

3 API Functions

The header mir_sdr.h defines the external function prototypes provided by this API. All functions are blocking.

3.1 Supported Functions

```
mir_sdr_ErrT mir_sdr_StreamInit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType, mir_sdr_If_kHzT ifType,
                    int LNAEnable, int *gRdBsystem, int useGrAltMode, int *samplesPerPacket,
                    mir_sdr_StreamCallback_t StreamCbFn, mir_sdr_GainChangeCallback_t GainChangeCbFn, void *cbContext);
mir_sdr_ErrT mir_sdr_StreamUninit(void);
mir_sdr_ErrT mir_sdr_SetRf(double drfHz, int abs, int syncUpdate);
mir_sdr_ErrT mir_sdr_SetFs(double dfsHz, int abs, int syncUpdate, int reCal);
mir_sdr_ErrT mir_sdr_SetGr(int gRdB, int abs, int syncUpdate);
mir_sdr_ErrT mir_sdr_SetGrParams(int minimumGr, int lnaGrThreshold);
mir_sdr_ErrT mir_sdr_SetDcMode(int dcCal, int speedUp);
mir_sdr_ErrT mir_sdr_SetDcTrackTime(int trackTime);
mir_sdr_ErrT mir_sdr_SetSyncUpdateSampleNum(unsigned int sampleNum);
mir_sdr_ErrT mir_sdr_SetSyncUpdatePeriod(unsigned int period);
mir_sdr_ErrT mir_sdr_ApiVersion(float *version);
mir_sdr_ErrT mir_sdr_ResetUpdateFlags(int resetGainUpdate, int resetRfUpdate, int resetFsUpdate);
mir_sdr_ErrT mir_sdr_DownConvert(short *in, short *xi, short *xq, unsigned int samplesPerPacket, mir_sdr_If_kHzT ifType,
                                 unsigned int M, unsigned int preReset);
mir_sdr_ErrT mir_sdr_SetPpm(double ppm);
mir_sdr_ErrT mir_sdr_SetLoMode(mir_sdr_LoModeT loMode);
mir_sdr_ErrT mir_sdr_SetGrAltMode(int *gRdB, int LNAenable, int *gRdBsystem, int abs, int syncUpdate);
mir_sdr_ErrT mir_sdr_DCoffsetIQimbalanceControl(unsigned int DCenable, unsigned int IQenable);
mir_sdr_ErrT mir_sdr_DecimateControl(unsigned int enable, unsigned int decimationFactor, unsigned int wideBandSignal);
mir_sdr_ErrT mir_sdr_AgcControl(mir_sdr_AgcControlT enable, int setPoint_dBfs, int knee_dBfs, unsigned int decay_ms,
                                   unsigned int hang_ms, int syncUpdate, int lnaEnable);
mir_sdr_ErrT mir_sdr_GetGrByFreq(double rfMHz, mir_sdr_BandT *band, int *gRdB, int LNAenable, int *gRdBsystem,
                                 int useGrAltMode);
mir_sdr_ErrT mir_sdr_Reinit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType, mir_sdr_If_kHzT ifType,
                            mir_sdr_LoModeT loMode, int LNAEnable, int *gRdBsystem, int useGrAltMode, int *samplesPerPacket,
                            mir_sdr_ReasonForReinitT reasonForReinit);
mir_sdr_ErrT mir_sdr_DebugEnable(unsigned int enable);
```

3.2 Deprecated Functions

These functions are no longer supported. They will still work but will be removed in a future version of the API.

```
mir_sdr_ErrT mir_sdr_Init(int gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType, mir_sdr_If_kHzT ifType,
                          int *samplesPerPacket);
mir_sdr_ErrT mir_sdr_Uninit(void);
mir_sdr_ErrT mir_sdr_ReadPacket(short *xi, short *xq, unsigned int *firstSampleNum, int *grChanged, int *rfChanged,
                                int *fsChanged);
mir_sdr_ErrT mir_sdr_SetParam(int ParamterId, int value);
```

3.3 Callback Function Prototypes

Stream callback:

```
typedef void (*mir_sdr_StreamCallback_t)(short *xi, short *xq, unsigned int firstSampleNum, int grChanged, int rfChanged,
                                         int fsChanged, unsigned int numSamples, unsigned int reset, void *cbContext);
```

AGC gain change callback:

```
typedef void (*mir_sdr_GainChangeCallback_t)(unsigned int gRdB, unsigned int lnaGRdB, void *cbContext);
```

3.4 mir_sdr_StreamInit

mir_sdr_ErrT mir_sdr_StreamInit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType, mir_sdr_If_kHzT ifType,
int LNAEnable, int *gRdBsystem, int useGrAltMode, int *samplesPerPacket,
mir_sdr_StreamCallback_t StreamCbFn, mir_sdr_GainChangeCallback_t GainChangeCbFn,
void *cbContext)

Description:
Replaces mir_sdr_Init(). It sets up a thread (or chain of threads) inside the API which will perform the processing chain, and then use the callback function to return the data to the calling application.
Processing chain (in order):

ReadPacket()	- fetch packet of IQ samples from USB interface
DownConvert()	- automatically enabled if the parameters shown for mir_sdr_DownConvert() are selected
Decimate()	- disabled by default
DCoffsetCorrection()	- enabled by default
IQimbalanceCorrection()	- enabled by default
Agc()	- disabled by default

Parameters:

gRdB	Initial gain reduction in dB, see default gain reduction tables, section 5, returns current IF gain value.
fsMHz	Specifies the ADC sample frequency in MHz, typical values between 2MHz and 10MHz. Decimation can be used to obtain lower sample rates.
rfMHz	Specifies the tuner centre frequency in MHz, see frequency allocation tables, section 6
bwType	Specifies the bandwidth to be used, see list in enumerated type for supported modes.
ifType	Specifies the IF to be used, see list in enumerated type for supported modes.
LNAEnable	Specifies if the LNA is enabled (if useGrAltMode == 1)
gRdBsystem	Input value ignored, returns overall system gain reduction value (if useGrAltMode == 1)
useGrAltMode	Specifies gain mode to use 0 → use mir_sdr_SetGr() to set initial gain reduction 1 → use mir_sdr_SetGrAltMode()
samplesPerPacket	Pointer to an unsigned integer which returns the number of samples that will be returned for the current configuration.
StreamCbFn	Specifies the callback function to use to send processed data
GainChangeCbFn	Specifies the callback function to use when an AGC gain change happens to notify the application of the current gain reduction settings.
cbContext	Pointer to a context passed that will be returned as a parameter in the callbacks.

Return:

mir_sdr_ErrT Error code (SUCCESS = 0, FAIL = non 0) – see enum definition.

3.5 mir_sdr_StreamUninit

mir_sdr_ErrT mir_sdr_Uninit(void)

Description:
Stops the stream and uninitialises the API.

Parameters:
None.

Return:

mir_sdr_ErrT Error code (SUCCESS = 0, FAIL = non 0) – see enum definition

3.6 mir_sdr_SetRf

mir_sdr_ErrT mir_sdr_SetRf(double drfHz, int abs, int syncUpdate)

Description:
Adjusts the nominal tuner centre frequency maintained in the internal state of the API. Depending on the state of the abs parameter, the drfHz parameter is either applied as an offset from the internally stored state of the API or is used in an absolute manner to modify the internally stored state. This command will only permit frequency changes that fall within the restrictions of the frequency allocation tables shown in section 5

Parameters:

drfHz	Frequency in Hz
abs	Indicates if drfHz is an absolute value or offset from previously set value: 0 → Offset Mode 1 → Absolute Mode
syncUpdate	Indicates if the RF frequency update is to be applied immediately or delayed until the next synchronous update point as configured in calls to mir_sdr_SetSyncUpdateSampleNum() and mir_sdr_SetSyncUpdatePeriod(). 0 → Immediate 1 → Synchronous

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.7 mir_sdr_SetFs

mir_sdr_ErrT mir_sdr_SetFs(double dfsHz, int abs, int syncUpdate, int reCal)

Description:
Adjusts the nominal ADC sampling frequency maintained in the internal state of the API. Depending on the state of the abs parameter, the dfsHz parameter is either applied as an offset from the internally stored state of the API or is used in an absolute manner to modify the internal stored API state. This command will typically permit only small changes in ADC sample frequency in the order of ±1000ppm. For large ADC sample frequency changes a mir_sdr_Uninit and mir_sdr_Init at the new sample rate must be performed.

Parameters:

dfsHz	Sample frequency or sample frequency offset in Hz
abs	Indicates if dfsHz is an absolute value or offset from previously set value 0 → Offset Mode 1 → Absolute Mode
syncUpdate	Indicates if the ADC sample frequency update is to be applied immediately or delayed is delayed until the next synchronous update point as configured in calls to mir_sdr_SetSyncUpdateSampleNum() and mir_sdr_SetSyncUpdatePeriod(). 0 → Immediate 1 → Synchronous
reCal	Recalibration of the PLL. Note: this is normally done only when the nominal sample frequency is set in mir_sdr_Init() and should be set to 0 elsewhere. 0 → no recalibration is made 1 → force a recalibration of the PLL

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.8 mir_sdr_SetGr

mir_sdr_ErrT mir_sdr_SetGr(int gRdB, int abs, int syncUpdate)

Description:
Programs the gain reduction required in the tuner. The abs parameter is used to determine whether the value specified is absolute gain reduction or an offset from the current gain value. The internal state is updated irrespective of what abs parameter is set to.

Parameters:

gRdB	Required gain reduction in dB, see default gain reduction tables.
abs	Indicates if gRdB is an absolute value or offset from previously set value 0 → Offset Mode 1 → Absolute Mode
syncUpdate	Indicates if the gain reduction is to be applied immediately or delayed until the next synchronous update point as configured in calls to mir_sdr_SetSyncUpdateSampleNum() and mir_sdr_SetSyncUpdatePeriod(). 0 → Immediate 1 → Synchronous

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.9 mir_sdr_SetGrParams

mir_sdr_ErrT mir_sdr_SetGrParams(int minimumGr, int lnaGrThreshold)

Description:
Modifies the default gain reduction parameters required in the tuner. A table of the hard coded defaults can be found in section 5

Parameters:

minimumGr	Minimum gain reduction in dB that can be programmed.
lnaGrThreshold	Threshold at which the LNA will be switched in.

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.10 mir_sdr_SetDcMode

mir_sdr_ErrT mir_sdr_SetDcMode(int dcCal, int speedUp)

Description:
Sets the DC offset correction mode for the tuner.

Parameters:

dcCal	DC offset correction mode: 0 → static 1 → Periodic 1 (Correction applied periodically every 6mS) 2 → Periodic 2 (Correction applied periodically every 12mS) 3 → Periodic 3 (Correction applied periodically every 24mS) 4 → one shot mode (correction applied each time gain update performed) 5 → continuous
speedUp	Speed up mode: 0 → disabled 1 → enabled

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.11 mir_sdr_SetDcTrackTime

mir_sdr_ErrT mir_sdr_SetDcTrackTime(int trackTime)

Description:
Set the time period over which the DC offset is tracked when in one-shot mode.

Parameters:

trackTime	Tracking time period – valid range is 1 to 63 and the duration can be calculated as: Duration (us) = 3 * trackTime

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.12 mir_sdr_SetSyncUpdateSampleNum

mir_sdr_ErrT mir_sdr_SetSyncUpdateSampleNum(unsigned int sampleNum)

Description:
Configures the sample number of the next synchronous update point. This is typically determined from the use of the firstSampleNum parameter returned in the mir_sdr_ReadPacket() function call. If the latency incurred over the USB causes this sample number to be set too late, the hardware will adjust automatically to correct for this.

Parameters:

sampleNum	Sample number of next synchronous update point.

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.13 mir_sdr_SetSyncUpdatePeriod

mir_sdr_ErrT mir_sdr_SetSyncUpdatePeriod(unsigned int period)

Description:
The value set in this call is automatically added to the sample number of the last synchronous update point to determine the next one. Note – this function should be called before mir_sdr_SetSyncUpdateSampleNum().

Parameters:

period	Defines the period between synchronous update points can be set between 1 and 1000000 samples

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.14 mir_sdr_ApiVersion

mir_sdr_ErrT mir_sdr_ApiVersion(float *version)

Description:
This function checks that the version in the include file is consistent with the dll version.

Parameters:

version	Pointer to a float which returns the version of the dll.

Return:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.15 mir_sdr_ResetUpdateFlags

mir_sdr_ErrT mir_sdr_ResetUpdateFlags(int resetGainUpdate, int resetRfUpdate, int resetFsUpdate)

Description:
If it is detected that an update to one or more of Gain Reduction, Rf Frequency or Sampling Frequency has not completed within some application specific timeout period, the logic prohibiting further updates can be reset using this function. More than one update type can be reset in each call, and once reset, new updates can be scheduled.

Parameters:
resetGainUpdate Reset Gain Reduction update logic:
0 → do not reset
1 → reset
resetRfUpdate Reset RF Frequency update logic:
0 → do not reset
1 → reset
resetFsUpdate Reset Sampling Frequency update logic:
0 → do not reset
1 → reset

Return:
mir_sdr_ErrT Error code (SUCCESS = 0, FAIL = non 0)

3.16 mir_sdr_Downconvert

mir_sdr_ErrT mir_sdr_DownConvert(short *in, short *xi, short *xq, unsigned int samplesPerPacket, mir_sdr_If_kHzT ifType, unsigned int M, unsigned int preReset)

Description:
A command which converts the sampled IF data obtained from streamed data to I and Q data in a zero IF format. The functions converts from low IF to zero IF by mixing, filtering and decimating the sampled IF data. The function will only operate correctly for the parameters detailed in the table below.

IF Frequency	IF Bandwidth	Input Sample Rate	Output Sample Rate	Decimation Factor
450kHz	200kHz	2MS/s	0.5MS/s	4
450kHz	300kHz	2MS/s	0.5MS/s	4
450kHz	600kHz	2MS/s	1MS/s	2
2048kHz	1536kHz	8.192MS/s	2.048MS/s	4

Parameters:
In Pointer to an array of size (samplesPerPacket * sizeof(short)) in which the I samples returned from streamed data are contained. If a non-zero IF mode this array will contain the sampled IF data.
xi Pointer to an array (of minimum size ((samplesPerPacket/M) * sizeof(short)) in which the down-converted I samples will be returned.
xq Pointer to an array (of minimum size ((samplesPerPacket/M) * sizeof(short)) in which the down-converted Q samples will be returned.
samplesPerPacket An unsigned integer which contains the number of samples that are contained within input IF sampled data array (in)
ifType Specifies the IF bandwidth that has been configured, see list in enumerated type for supported modes.
M Desired decimation factor, see attached table for list of applicable values
preReset If preReset is equal to 1 then the filtering state will be reset prior to any filtering operation

Return:
mir_sdr_ErrT Error code (SUCCESS = 0, FAIL = non 0)

3.17 mir_sdr_SetPpm

mir_sdr_ErrT mir_sdr_SetPpm(double ppm)

Description:
To specify a correction factor used to account for offsets from the nominal in the crystal oscillator.

Parameters:

ppm	Parts per million offset (e.g. +/- 1 ppm specifies a +/- 24Hz error for a 24MHz crystal.

Return:

mir_sdr_ErrT	mir_sdr_Success (no error checking performed)

3.18 mir_sdr_SetLoMode

mir_sdr_ErrT mir_sdr_SetLoMode(mir_sdr_LoModeT loMode)

Description:
Allows a particular up-converted LO frequency to be specifed or selects automatic mode which allows the API to determine the most appropriate LO frequency across all RF frequency ranges.

Parameters:

loMode	Specifies LO frequency (see mir_sdr_LoModeT for possible values)

Result:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.19 mir_sdr_SetGrAltMode

mir_sdr_ErrT mir_sdr_SetGrAltMode(int *gRdB, int LNAenable, int *gRdBsystem, int abs, int syncUpdate)

Description:
Alternative method (to mir_sdr_SetGR()) to set the gain reduction based on internal gain tables based on current band, requested gain reduction and whether the LNA is enabled or not.

Parameters:

gRdB	Input value is requested gain reduction, returns IF gain reduction value.
LNAenable	Tuner LNA control 0 → disable LNA (increases gain reduction) 1 → enable LNA (decreases gain reduction)
gRdBsystem	Input value ignored, returns overall system gain reduction.
abs	How the input value is interpreted 0 → gRdB is a delta to be added to the previous value. 1 → gRdB is an absolute value.
syncUpdate	When to update the gain reduction change. 0 → immediate update 1 → synchronous update (see mir_sdr_SetGR() for more details)

Result:

mir_sdr_ErrT	Error code (SUCCESS = 0, FAIL = non 0)

3.20 mir_sdr_DCoffsetIQimbalanaceControl

mir_sdr_ErrT mir_sdr_DCoffsetIQimbalanceControl(unsigned int DCenable, unsigned int IQenable)

Description:
Individual enables for DC offset correction and IQ imbalance correction.

Parameters:

DCenable	DC offset correction control: 0 → DC correction disabled 1 → DC correction enabled (default)
IQenable	IQ correction control: 0 → IQ correction disabled 1 → IQ correction enabled (default)

Result:

mir_sdr_ErrT mir_sdr_Success (no error checking performed)

3.21 mir_sdr_DecimateControl

mir_sdr_ErrT mir_sdr_DecimateControl(unsigned int enable, unsigned int decimationFactor,
unsigned int wideBandSignal)

Description:
Used to control whether decimation is enabled or not. Valid decimation factors are 2, 4, 8, 16, 32 or 64 only. If other values are specified then decimation will not be enabled. If wide band mode is selected, the decimation algorithm uses a sequence of half-band filters to achieve the required decimation, otherwise a box-filter is used which is much more efficient but may cause roll-off in the passband of the received signal depending on bandwidth. Otherwise, a simple block averaging is used to reduce the CPU load, but with increased in-band roll-off.

Note: Requires internal stream thread to have been created via mir_sdr_StreamInit() for decimation to be enabled. Also for IQ output in Low IF mode, enable must = 0 as the decimation is automatic within the API.

Parameters:

enable	Decimation control 0 → Decimation disabled (default) 1 → Decimation enabled
decimationFactor	Decimation factor (2, 4, 8, 16, 32 or 64 only)
widebandSignal	Filter control: 0 → Use averaging (default) 1 → Use half-band filters

Result:

mir_sdr_ErrT mir_sdr_Success (no error checking performed)

3.22 mir_sdr_AgcControl

mir_sdr_ErrT mir_sdr_AgcControl(mir_sdr_AgcControlT enable, int setPoint_dBfs, int knee_dBfs, unsigned int decay_ms, unsigned int hang_ms, int syncUpdate, int lnaEnable)

Description:
Used to control whether AGC is enabled or not and parameters to allow the AGC to be configured.

Note: Requires internal stream thread to have been created via mir_sdr_StreamInit() for AGC to be enabled.

Parameters:

enable	Specifies the AGC mode required. See enumerated types for valid values. Standard mode is 100Hz (mir_sdr_100HZ)
setPoint_dBfs	Specifies the required set point in dBfs
knee_dBfs	Not currently used, set to 0
decay_ms	Not currently used, set to 0
hang_ms	Not currently used, set to 0
syncUpdate	Update control 0 → immediate update 1 → synchronous update (see mir_sdr_SetGR() for more details)
lnaEnable	indicates the LNA state to use in gain updates when AGC uses mir_sdr_SetGrAltMode() as specified when calling mir_sdr_StreamInit()

Result:

mir_sdr_ErrT	mir_sdr_Success (no error checking performed)

3.23 mir_sdr_GetGrByFreq

mir_sdr_ErrT mir_sdr_GetGrByFreq(double rfMHz, mir_sdr_BandT *band, int *gRdB, int LNAenable, int *gRdBsystem, int useGrAltMode)

Description:
Get the gain reduction settings for a gain reduction request for a particular RF frequency. This function does not update the hardware. It also determines the band the RF frequency is in and returns it.

Parameters:

rfMHz	Tuner RF frequency
band	Input value ignored, returns band for requested RF frequency
gRdB	On input, gain reduction value, returns IF gain reduction value (if useGrAltMode == 1)
LNAenable	Tuner LNA control (if useGrAltMode == 1): 0 → disable LNA (increases gain reduction) 1 → enable LNA (decreases gain reduction)
gRdBsystem	Input value ignored, returns overall system gain reduction (if useGrAltMode == 1)
useGrAltMode	Gain reduction mode control: 0 → use mir_sdr_SetGr() to set initial gain reduction 1 → use mir_sdr_SetGrAltMode()

Result:

mir_sdr_ErrT	mir_sdr_Success (no error checking performed)

3.24 mir_sdr_Reinit

```
mir_sdr_ErrT mir_sdr_Reinit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType, mir_sdr_If_kHzT ifType,
                            mir_sdr_LoModeT loMode, int LNAEnable, int *gRdBsystem, int useGrAltMode,
                            int *samplesPerPacket, mir_sdr_ReasonForReinitT reasonForReinit)
```

Description:
Used to change any combination of values of the parameters. If required it stop the stream, change the values and then start the stream again, otherwise it will make the changes directly. Only those parameters required for the requested reason(s) need to be provided – as specified below:

mir_sdr_CHANGE_GR: gRdB, LNAenable, gRdBsystem, useGrAltMode (does not require Uninit/Init)	
mir_sdr_CHANGE_FS_FREQ: fsMHz	(requires Uninit/Init)
mir_sdr_CHANGE_RF_FREQ: rfMHz	(requires Uninit/Init if changing band)
mir_sdr_CHANGE_BW_TYPE: bwType	(requires Uninit/Init)
mir_sdr_CHANGE_IF_TYPE: ifType	(requires Uninit/Init)
mir_sdr_CHANGE_LO_MODE: loMode	(requires Uninit/Init)

Parameters:

gRdB	Initial gain reduction in dB, see default gain reduction tables, section 5, returns current IF gain value.
fsMHz	Specifies the ADC sample frequency in MHz, typical values between 2MHz and 10MHz. Decimation can be used to obtain lower sample rates.
rfMHz	Specifies the tuner centre frequency in MHz, see frequency allocation tables, section 6
bwType	Specifies the bandwidth to be used, see list in enumerated type for supported modes.
ifType	Specifies the IF to be used, see list in enumerated type for supported modes.
LNAEnable	Specifies if the LNA is enabled (if useGrAltMode == 1)
gRdBsystem	Input value ignored, returns overall system gain reduction value (if useGrAltMode == 1)
useGrAltMode	Specifies gain mode to use 0 → use mir_sdr_SetGr() to set initial gain reduction 1 → use mir_sdr_SetGrAltMode()

samplesPerPacket Pointer to an unsigned integer which returns the number of samples that will be returned for the current configuration.

reasonforReinit Used to specify the reason for a reinitialise request - values should be or'd together if there are multiple reasons for the request.

Result:

mir_sdr_ErrT mir_sdr_Success (no error checking performed)

3.25 mir_sdr_DebugEnable

```
mir_sdr_ErrT mir_sdr_DebugEnable(unsigned int enable)
```

Description:
Used to enable debug message output.

Parameters:

enable	Debug output control: 0 → messages disabled (default) 1 → messages enabled

Result:

mir_sdr_ErrT mir_sdr_Success (no error checking performed)

3.26 Stream callback

```
typedef void (*mir_sdr_StreamCallback_t)(short *xi, short *xq, unsigned int *firstSampleNum, int *grChanged,
                    int *rfChanged, int *fsChanged, unsigned int numSamples, unsigned int reset, void *cbContext);
```

Description:
This callback is triggered when there are samples to be processed.

Parameters:

xi	Pointer to real data in the buffer.
xq	Pointer to the imaginary data in the buffer.
firstSampleNum	Number of first sample in buffer (used for synchronous updates). Note that this is divided in the API by the decimationFactor, to make it relative to the output sample rate. The values specified for sample number and period for synchronous updates should also be relative to the output sample rate.
grChanged	Indicates if the gain reduction value was changed somewhere in the current buffer.
rfChanged	Indicates if the RF frequency was changed somewhere in the current buffer.
fsChanged	Indicates if the sample frequency value was changed somewhere in the current buffer.
numSamples	The number of samples in the current buffer.
reset	Indicates if a re-initialisation has occurred and that the local buffering should be reset.
cbContext	Pointer to context passed into mir_sdr_StreamInit()

3.27 AGC Gain Change callback

```
typedef void (*mir_sdr_GainChangeCallback_t)(unsigned int gRdB, unsigned int lnaGRdB, void *cbContext);
```

Description:
This callback is triggered when a gain change occurs in the AGC loop.

Parameters:

gRdB	New IF gain reduction value applied by AGC
lnaGRdB	LNA gain reduction value
cbContext	Pointer to context passed into mir_sdr_StreamInit()

4 API Usage

The example code below shows how the calls are typically used - note that no error processing is shown.

```
void callback(short *xi, short *xq, unsigned int numSamps, unsigned int reset)
{
  // put data in a buffer to be used in main loop
  ...
  // signal semaphore (pseudo code)
  ReleaseSemaphore(dataAvailSema);
  return;
}

void callbackGC(unsigned int gRdB, unsigned int lnaGRdB)
{
  // do something with updated gain information if required
  ...
  return;
}

int main(void)
{
        // data declarations
        mir_sdr_ErrT err;
        int sps;
        float *ver;
        int newGr = 40;
        int sysGr = 40;
        int done = 0;
        int syncUpdate = 0;   // initially use asynchronous updates
        semaphore dataAvailSema = CreateSemaphore();  // create semaphore (pseudo code)
        ...

        // check API version
        err = mir_sdr_ApiVersion(&ver);
        if (ver != MIR_SDR_API_VERSION)
        {
           // include file does not match dll. Deal with error condition.
        }

        // enable debug output
        mir_sdr_DebugEnable(1);

        // disable DC offset and IQ imbalance correction (default is for these to be enabled - this just shows
        // how to disable if required)
        mir_sdr_DCoffsetIQimbalanceControl(1, 0);

        // disable decimation and set decimation factor to 4  (this is for information only as decimation is
        // disabled by default)
        mir_sdr_DecimateControl(0, 4, 0);

        // enable AGC with a setPoint of -30dBfs
        mir_sdr_AgcControl(1, -30, 0, 0, 0, 0);

        // initialise API and hardware for DAB demodulation: initial gain reduction of 40dB, sample
        // rate of 2.048MHz, centre frequency of 222.064MHz, double sided bandwidth of 1.536MHz and
        // a zero-IF

        // this also configures the API to use the new mir_sdr_SetGrAltMode() to
        // control the gain and disables the LNA. The overall system gain is returned in sysGr

        // used for DAB type signals
        err = mir_sdr_StreamInit(&newGr, 2.048, 222.064, mir_sdr_BW_1_536, mir_sdr_IF_Zero, 0, &sysGr, 1, &sps, callback,
                                      callbackGC, (void *)NULL);
        if (err != mir_sdr_Success)
        {
           // Error detected. Deal with error condition.
        }
```

```
        // configure DC tracking in tuner
        err = mir_sdr_SetDCmode(4);             // select one-shot tuner DC offset correction
        err = mir_sdr_SetDCTrackTime(63);       // with maximum tracking time
        ...

        // processing loop
        while(!done)
        {
            ...
            // wait on semaphore for samples to be received in callback  (pseudo code)
            WaitSemaphore(dataAvailSema);
            // get data from buffer
            ...
            // rf frequency tracking
            ...
            // sample frequency tracking
            ...
            // demodulate data (pass I and Q data to demod)
            ...
            // for DAB, it may be required to do all updates during NULL period, so once demodulator has
            // established where this is, pass the details to the hardware and switch to using
            // synchronous updates
            mir_sdr_SetSyncUpdatePeriod(period);          // set period first
            mir_sdr_SetSyncUpdateSampleNum(sampleNum);    // then set sample number at start of NULL
            syncUpdate = 1;
            ...
        }
        // At exit
        err = mir_sdr_StreamUninit();
}
```

5 Default Gain Reduction Tables

5.1 10kHz – 59.999999MHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 10kHz and 59.999999MHz. The table also shows the default LNA and Mixer thresholds. This is only valid for the default gain map.

gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction
0	0	0	0	35	11	24	0	70	46	24	0
1	1	0	0	36	12	24	0	71	47	24	0
2	2	0	0	37	13	24	0	72	48	24	0
3	3	0	0	38	14	24	0	73	49	24	0
4	4	0	0	39	15	24	0	74	50	24	0
5	5	0	0	40	16	24	0	75	51	24	0
6	6	0	0	41	17	24	0	76	52	24	0
7	7	0	0	42	18	24	0	77	53	24	0
8	8	0	0	43	19	24	0	78	54	24	0
9	9	0	0	44	20	24	0	79	55	24	0
10	10	0	0	45	21	24	0	80	56	24	0
11	11	0	0	46	22	24	0	81	57	24	0
12	12	0	0	47	23	24	0	82	58	24	0
13	13	0	0	48	24	24	0	83	40	24	19
14	14	0	0	49	25	24	0	84	41	24	19
15	15	0	0	50	26	24	0	85	42	24	19
16	16	0	0	51	27	24	0	86	43	24	19
17	17	0	0	52	28	24	0	87	44	24	19
18	18	0	0	53	29	24	0	88	45	24	19
19	19	0	0	54	30	24	0	89	46	24	19
20	20	0	0	55	31	24	0	90	47	24	19
21	21	0	0	56	32	24	0	91	48	24	19
22	22	0	0	57	33	24	0	92	49	24	19
23	23	0	0	58	34	24	0	93	50	24	19
24	24	0	0	59	35	24	0	94	51	24	19
25	25	0	0	60	36	24	0	95	52	24	19
26	26	0	0	61	37	24	0	96	53	24	19
27	27	0	0	62	38	24	0	97	54	24	19
28	28	0	0	63	39	24	0	98	55	24	19
29	29	0	0	64	40	24	0	99	56	24	19
30	30	0	0	65	41	24	0	100	57	24	19
31	31	0	0	66	42	24	0	101	58	24	19
32	32	0	0	67	43	24	0	102	59	24	19
33	33	0	0	68	44	24	0				
34	34	0	0	69	45	24	0				

5.2 60MHz – 119.999999MHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 60MHz and 119.999999MHz. The table also shows the default LNA and Mixer thresholds. This is only valid for the default gain map.

gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction
0	0	0	0	35	11	24	0	70	46	24	0
1	1	0	0	36	12	24	0	71	47	24	0
2	2	0	0	37	13	24	0	72	48	24	0
3	3	0	0	38	14	24	0	73	49	24	0
4	4	0	0	39	15	24	0	74	50	24	0
5	5	0	0	40	16	24	0	75	51	24	0
6	6	0	0	41	17	24	0	76	52	24	0
7	7	0	0	42	18	24	0	77	53	24	0
8	8	0	0	43	19	24	0	78	54	24	0
9	9	0	0	44	20	24	0	79	55	24	0
10	10	0	0	45	21	24	0	80	56	24	0
11	11	0	0	46	22	24	0	81	57	24	0
12	12	0	0	47	23	24	0	82	58	24	0
13	13	0	0	48	24	24	0	83	40	24	19
14	14	0	0	49	25	24	0	84	41	24	19
15	15	0	0	50	26	24	0	85	42	24	19
16	16	0	0	51	27	24	0	86	43	24	19
17	17	0	0	52	28	24	0	87	44	24	19
18	18	0	0	53	29	24	0	88	45	24	19
19	19	0	0	54	30	24	0	89	46	24	19
20	20	0	0	55	31	24	0	90	47	24	19
21	21	0	0	56	32	24	0	91	48	24	19
22	22	0	0	57	33	24	0	92	49	24	19
23	23	0	0	58	34	24	0	93	50	24	19
24	24	0	0	59	35	24	0	94	51	24	19
25	25	0	0	60	36	24	0	95	52	24	19
26	26	0	0	61	37	24	0	96	53	24	19
27	27	0	0	62	38	24	0	97	54	24	19
28	28	0	0	63	39	24	0	98	55	24	19
29	5	24	0	64	40	24	0	99	56	24	19
30	6	24	0	65	41	24	0	100	57	24	19
31	7	24	0	66	42	24	0	101	58	24	19
32	8	24	0	67	43	24	0	102	59	24	19
33	9	24	0	68	44	24	0				
34	10	24	0	69	45	24	0				

5.3 120MHz – 249.999999MHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 120MHz and 249.999999MHz. The table also shows the default LNA and Mixer thresholds. This is only valid for the default gain map.

gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction
0	0	0	0	35	11	24	0	70	46	24	0
1	1	0	0	36	12	24	0	71	47	24	0
2	2	0	0	37	13	24	0	72	48	24	0
3	3	0	0	38	14	24	0	73	49	24	0
4	4	0	0	39	15	24	0	74	50	24	0
5	5	0	0	40	16	24	0	75	51	24	0
6	6	0	0	41	17	24	0	76	52	24	0
7	7	0	0	42	18	24	0	77	53	24	0
8	8	0	0	43	19	24	0	78	54	24	0
9	9	0	0	44	20	24	0	79	55	24	0
10	10	0	0	45	21	24	0	80	56	24	0
11	11	0	0	46	22	24	0	81	57	24	0
12	12	0	0	47	23	24	0	82	58	24	0
13	13	0	0	48	24	24	0	83	40	24	19
14	14	0	0	49	25	24	0	84	41	24	19
15	15	0	0	50	26	24	0	85	42	24	19
16	16	0	0	51	27	24	0	86	43	24	19
17	17	0	0	52	28	24	0	87	44	24	19
18	18	0	0	53	29	24	0	88	45	24	19
19	19	0	0	54	30	24	0	89	46	24	19
20	20	0	0	55	31	24	0	90	47	24	19
21	21	0	0	56	32	24	0	91	48	24	19
22	22	0	0	57	33	24	0	92	49	24	19
23	23	0	0	58	34	24	0	93	50	24	19
24	24	0	0	59	35	24	0	94	51	24	19
25	25	0	0	60	36	24	0	95	52	24	19
26	26	0	0	61	37	24	0	96	53	24	19
27	27	0	0	62	38	24	0	97	54	24	19
28	28	0	0	63	39	24	0	98	55	24	19
29	5	24	0	64	40	24	0	99	56	24	19
30	6	24	0	65	41	24	0	100	57	24	19
31	7	24	0	66	42	24	0	101	58	24	19
32	8	24	0	67	43	24	0	102	59	24	19
33	9	24	0	68	44	24	0				
34	10	24	0	69	45	24	0				

5.4 250MHz – 419.999999MHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 250MHz and 419.999999MHz. The table also shows the default LNA and Mixer thresholds. This is only valid for the default gain map.

gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction
0	0	0	0	35	11	24	0	70	46	24	0
1	1	0	0	36	12	24	0	71	47	24	0
2	2	0	0	37	13	24	0	72	48	24	0
3	3	0	0	38	14	24	0	73	49	24	0
4	4	0	0	39	15	24	0	74	50	24	0
5	5	0	0	40	16	24	0	75	51	24	0
6	6	0	0	41	17	24	0	76	52	24	0
7	7	0	0	42	18	24	0	77	53	24	0
8	8	0	0	43	19	24	0	78	54	24	0
9	9	0	0	44	20	24	0	79	55	24	0
10	10	0	0	45	21	24	0	80	56	24	0
11	11	0	0	46	22	24	0	81	57	24	0
12	12	0	0	47	23	24	0	82	58	24	0
13	13	0	0	48	24	24	0	83	40	24	19
14	14	0	0	49	25	24	0	84	41	24	19
15	15	0	0	50	26	24	0	85	42	24	19
16	16	0	0	51	27	24	0	86	43	24	19
17	17	0	0	52	28	24	0	87	44	24	19
18	18	0	0	53	29	24	0	88	45	24	19
19	19	0	0	54	30	24	0	89	46	24	19
20	20	0	0	55	31	24	0	90	47	24	19
21	21	0	0	56	32	24	0	91	48	24	19
22	22	0	0	57	33	24	0	92	49	24	19
23	23	0	0	58	34	24	0	93	50	24	19
24	24	0	0	59	35	24	0	94	51	24	19
25	25	0	0	60	36	24	0	95	52	24	19
26	26	0	0	61	37	24	0	96	53	24	19
27	27	0	0	62	38	24	0	97	54	24	19
28	28	0	0	63	39	24	0	98	55	24	19
29	29	0	0	64	40	24	0	99	56	24	19
30	30	0	0	65	41	24	0	100	57	24	19
31	31	0	0	66	42	24	0	101	58	24	19
32	32	0	0	67	43	24	0	102	59	24	19
33	33	0	0	68	44	24	0				
34	34	0	0	69	45	24	0				

5.5 420MHz – 999.999999MHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 420MHz and 999.999999MHz. The table also shows the default LNA and Mixer thresholds. This is only valid for the default gain map.

gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction
0	0	0	0	35	28	7	0	70	44	7	19
1	1	0	0	36	29	7	0	71	45	7	19
2	2	0	0	37	30	7	0	72	46	7	19
3	3	0	0	38	31	7	0	73	47	7	19
4	4	0	0	39	32	7	0	74	48	7	19
5	5	0	0	40	33	7	0	75	49	7	19
6	6	0	0	41	34	7	0	76	50	7	19
7	7	0	0	42	35	7	0	77	51	7	19
8	8	0	0	43	36	7	0	78	52	7	19
9	9	0	0	44	37	7	0	79	53	7	19
10	10	0	0	45	38	7	0	80	54	7	19
11	11	0	0	46	39	7	0	81	55	7	19
12	5	7	0	47	40	7	0	82	56	7	19
13	6	7	0	48	41	7	0	83	57	7	19
14	7	7	0	49	42	7	0	84	58	7	19
15	8	7	0	50	43	7	0	85	59	7	19
16	9	7	0	51	44	7	0				
17	10	7	0	52	45	7	0				
18	11	7	0	53	46	7	0				
19	12	7	0	54	47	7	0				
20	13	7	0	55	48	7	0				
21	14	7	0	56	49	7	0				
22	15	7	0	57	50	7	0				
23	16	7	0	58	51	7	0				
24	17	7	0	59	52	7	0				
25	18	7	0	60	53	7	0				
26	19	7	0	61	54	7	0				
27	20	7	0	62	55	7	0				
28	21	7	0	63	56	7	0				
29	22	7	0	64	57	7	0				
30	23	7	0	65	58	7	0				
31	24	7	0	66	40	7	19				
32	25	7	0	67	41	7	19				
33	26	7	0	68	42	7	19				
34	27	7	0	69	43	7	19				

5.6 1GHz – 2GHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 1GHz and 2GHz. The table also shows the default LNA and Mixer thresholds. This is only valid for the default gain map.

gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction	gRdB	Baseband Gain Reduction	LNA Gain Reduction	Mixer Gain Reduction
0	0	0	0	35	28	7	0	70	44	7	19
1	1	0	0	36	29	7	0	71	45	7	19
2	2	0	0	37	30	7	0	72	46	7	19
3	3	0	0	38	31	7	0	73	47	7	19
4	4	0	0	39	32	7	0	74	48	7	19
5	5	0	0	40	33	7	0	75	49	7	19
6	6	0	0	41	34	7	0	76	50	7	19
7	7	0	0	42	35	7	0	77	51	7	19
8	8	0	0	43	36	7	0	78	52	7	19
9	9	0	0	44	37	7	0	79	53	7	19
10	3	7	0	45	38	7	0	80	54	7	19
11	4	7	0	46	39	7	0	81	55	7	19
12	5	7	0	47	40	7	0	82	56	7	19
13	6	7	0	48	41	7	0	83	57	7	19
14	7	7	0	49	42	7	0	84	58	7	19
15	8	7	0	50	43	7	0	85	59	7	19
16	9	7	0	51	44	7	0				
17	10	7	0	52	45	7	0				
18	11	7	0	53	46	7	0				
19	12	7	0	54	47	7	0				
20	13	7	0	55	48	7	0				
21	14	7	0	56	49	7	0				
22	15	7	0	57	50	7	0				
23	16	7	0	58	51	7	0				
24	17	7	0	59	52	7	0				
25	18	7	0	60	53	7	0				
26	19	7	0	61	54	7	0				
27	20	7	0	62	55	7	0				
28	21	7	0	63	56	7	0				
29	22	7	0	64	57	7	0				
30	23	7	0	65	58	7	0				
31	24	7	0	66	40	7	19				
32	25	7	0	67	41	7	19				
33	26	7	0	68	42	7	19				
34	27	7	0	69	43	7	19				

6 Frequency Allocation Tables

When using the init command any frequency range supported by the hardware can be programmed and this will configure the front end accordingly. Once the desired frequency has been programmed the mir_sdr_setRf command can be used to alter the centre frequency. It should be noted though that the mir_sdr_setRf command can only change the frequency within a set of predefined bands. If a frequency is desired that falls outside the current band then a mir_sdr_Uninit command must be issued followed by a mir_sdr_Init command at the new frequency to force reconfiguration of the front end. The table below shows the frequency bands over which the mir_sdr_setRf commands will permit operation.

- 10kHz – 11.999999MHz
- 12MHz – 29.999999MHz
- 30MHz – 59.999999MHz
- 60MHz – 119.999999MHz
- 120MHz – 249.999999MHz
- 250MHz – 419.999999MHz
- 420MHz – 999.999999MHz
- 1GHz – 2GHz

For more information contact:

Mirics Limited

info@mirics.com
www.mirics.com