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

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;
```

3 API Functions

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

```
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_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_SetParam (int ParamterId, int value);
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);
```

3.1 mir_sdr_Init

mir_sdr_ErrT mir_sdr_Init(int gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
mir_sdr_If_kHzT ifType, int *samplesPerPacket)

Description:
Initiates the API for the specified centre frequency.

Parameters:

gRdB	Gain reduction in dB, see default gain reduction tables, section 5
fsMHz	Specifies the ADC sample frequency in MHz, typical values between 0.5MHz and 12MHz
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.
samplesPerPacket	Pointer to an unsigned integer which returns the number of samples that will be returned for this configuration in each call to mir_sdr_ReadPacket()

Return:

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

3.2 mir_sdr_Uninit

mir_sdr_ErrT mir_sdr_Uninit(void)

Description:
Uninitialises the API.

Parameters:
None.

Return:

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

3.3 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)

3.4 mir_sdr_ReadPacket

```
mir_sdr_ErrT mir_sdr_ReadPacket(short *xi, short *xq, unsigned int *firstSampleNum,
                                int *grChanged, int *rfChanged, int *fsChanged)
```

Description:
This function is used to retrieve data from the hardware. The data is returned as 16bit left justified integers

Parameters:

xi	Pointer to an array (of minimum size samplesPerPacket * sizeof(short)) in which the I samples will be returned. If a non-zero IF is used, this array will contain the sampled IF data.
xq	Pointer to an array (of minimum size samplesPerPacket * sizeof(short)) in which the Q samples will be returned.
firstSampleNum	Pointer to an unsigned integer in which the sample count (modulo 2^32) of the first sample of the retrieved data will be returned.
grChanged	Pointer to an integer which indicates if the gain reduction has changed: 0 → no change 1 → changed.
rfChanged	Pointer to an integer which indicates if the RF frequency has changed: 0 → no change 1 → changed.
fsChanged	Pointer to an integer which indicates if the sample frequency has changed: 0 → no change 1 → changed.

Return:

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

3.5 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)

3.6 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)

3.7 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)

3.8 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)

3.9 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)

3.10 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)

3.11 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)

3.12 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)

3.13 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)

3.14 mir_sdr_SetParam

mir_sdr_ErrT mir_sdr_SetParam (int ParamterId, int value)

Description:
A command designed to allow the setting of various hardware control parameters. Currently the command only accepts one parameter ID which is 101 and this allows the hardware to configure a change to the 1st LO frequency. This command must be executed before mir_sdr_Init to configure the hardware prior to initialization.

Parameters:
ParamterId 101 → Update 1st LO Frequency

value →
19200000 – 1st LO Frequency 168MHz
22000000 – 1st LO Frequency 144MHz
24576000 – 1st LO Frequency 120MHz

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

3.15 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 mir_sdr_ReadPacket 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 mir_sdr-Readpacket 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)

4 API Usage

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

```
// Data declarations
mir_sdr_ErrT err;
int sps;
short *xi;
short *xq;
float *ver;
unsigned int fs;
int grc;
int grChangedAfter = 0;
int newGr = 40;
int oldGr = 40;
int done = 0;
int syncUpdate = 0;   // initially use asynchronous updates
...

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

// 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
// used for DAB type signals
err = mir_sdr_Init(newGr, 2.048, 222.064, mir_sdr_BW_1_536, mir_sdr_IF_Zero, &sps);

// Alocate data buffers (NUM_PKTS needs to be defined by the user)
xi = (short *)malloc(sps * NUM_PKTS * sizeof(short));
xq = (short *)malloc(sps * NUM_PKTS * sizeof(short));

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

// Processing loop
while(!done)
{
  ...
  // fill local buffer with IQ data and check for updates
  for (i = 0; i < NUM_PKTS; i++)
  {
    err = mir_sdr_ReadPacket(&xi[i * sps], &xq[i * sps], &fs, &grc, &rfc, &fsc);
    // Check if any outstanding gain changes have been applied
    if (grc)
    {
      grChangedAfter = (i + 1) * sps;      // indicate where change occurred
      grc = 0;
    }
    // Check if any outstanding center frequency changes have been applied
    ...
    // Check if any outstanding sample frequency changes have been applied
    ...
  }
  // dc offset removal (usually needs to be done before AGC)
  ...
  // agc
  if (grChangedAfter > 0)              // don't do AGC if update pending
  {
    newGr = doAgc(xi, xq, grChangedAfter);  // call AGC loop and indicate change
    grChangedAfter = 0;                  // reset flag in case no gain change is required
    if (newGR != oldGr)                  // only do update if change required
    {
```

```
        err = mir_sdr_SetGr(newGr, 1, syncUpdate);    // use absolute value
        grChangedAfter = -1;                // inhibit loop until this change completes
        oldGr = newGr;
      }
    }
    ...
    // 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_Uninit();
```

5 Default Gain Reduction Tables

5.1 100kHz – 59.999999MHz band

The table below shows the default gain reduction scheme used when operating between the frequencies of 100kHz and 59.999999MHz. The table also shows the default LNA and Mixer thresholds.

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.

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.

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.

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.

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.

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.

- 100kHz – 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