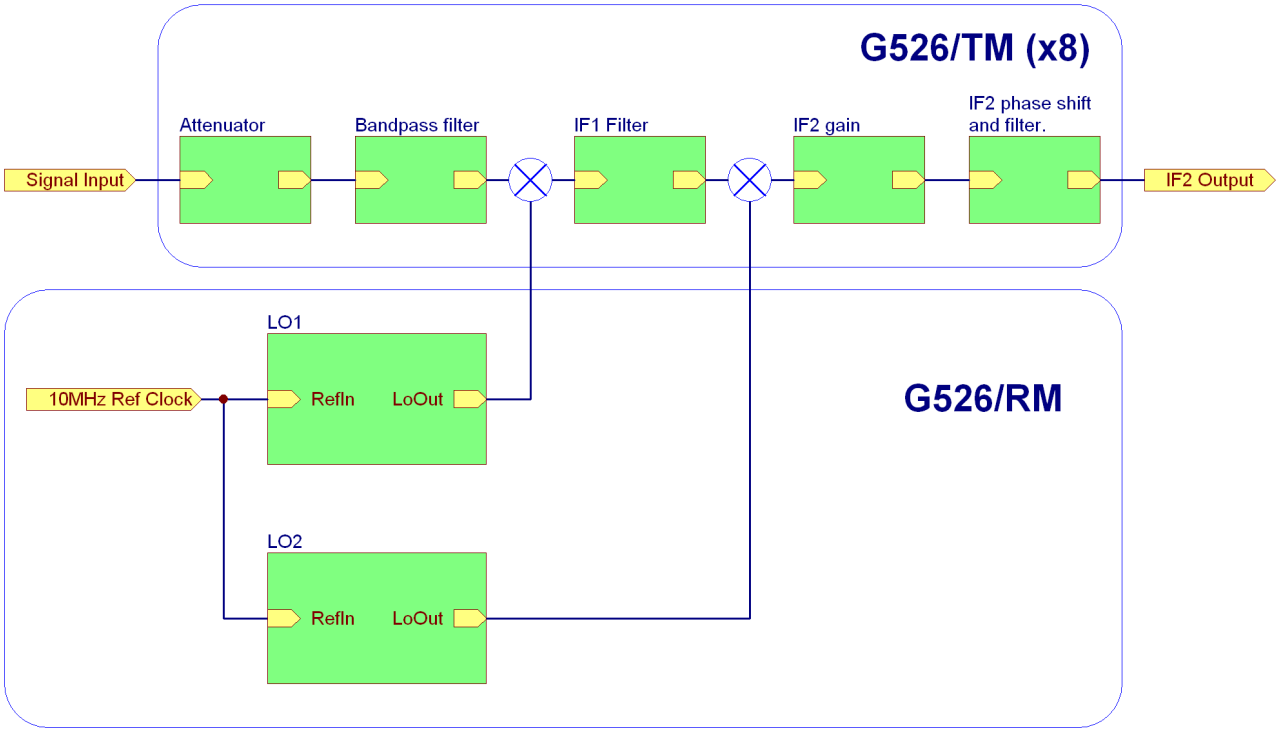
WR-G526 API for Linux

Introduction

This document describes the software interface to control a WR-G526 receiver system.

A WRG526 system consists of one master unit (G526e/RM or G526e/RMF) and one to eight slave units (G526e/TM). A “combined” version is available, which incorporates one RM and one TM into a single module (G526e/CM).

The entire system can be visualized from this block diagram:



Software Requirements

Linux WR-G526 API is implemented as shared object (wrg526api.so) that can be loaded by an application dynamically. It can work in 32bit (i386) and 64bit (x86_64) Linux environment. In 32bit Linux the API is stored in /usr/lib directory. In 64bit Linux 32bit version of the API is stored in /usr/lib directory and 64bit version of the API is stored in /usr/lib64 directory.

The system can be controlled by any Linux software which is able to load shared object dynamically.

Your application must provide storage space for some data structures used by the API. This is described in detail below.

The following section describes the usual order of events for starting a WR-G526 system, and setting a frequency. This is followed by a summary of all API calls, with short descriptions, finally followed by in-depth documentation for each call.

The data structures used are shown in the Appendix A, in C/C++ format.

All the functions provided by the API are thread safe and C calling conventions are used for these functions.

Typical usage

To initialize

Reserve three scratch areas in memory:

```
G5_RADIO RadioList[9];     //List of each individual radio, and per-radio data
G5_STATUS g5stat={0};      //High level intermediate calculation data
G5_HW g5hw={0};            //Low level hardware settings
```

There is no need to initialize these variables. **RadioList** is fully initialized by **G5_FindRadios, g5stat** and **g5hw** must be initialized by zeros before first call of any API function.

Call **G5_FindRadios** to populate the **RadioList** array. (Entry zero is the master, and subsequent entries are the slaves, in alphabetic order by serial number)

Save the return value (the number of radios found), to pass as **NumberOfRadios** to all other calls.

Call **G5_OpenRadios** to open communication with all the radios.

Set the **Power** field of each radio in **RadioList** to 1, then call **G5_SetPower** to turn on all radios.

Call **G5_SetRFAttn** and **G5_SetIFGain** (described below), to set the initial state of the slave radios.

To tune a new frequency

Call **G5_SetFrequency** to start tuning

If phase adjust is required, set the desired phase-adjust DAC value into the **Phase** field of each slave radio, then call **G5_SetPhase.**

Loop calling **G5_GetHwStatus** until it indicates that all radios are idle, and both LO PLLs are locked. (Bits 0-15 clear, bits 16 & 17 set.) (Not required for G526e/RMF)

To adjust the RF attenuator

Call **G5_SetRFAttn** with the desired attenuation level (0-30 dB).

To adjust the IFGAIN

Call **G5_SetIFGain** with 0 for normal, or 1 for -20dB gain. (This disables a 20dB amplifier).

To adjust the LO1 PLL settling speed

Call **G5_SetPllSpeed** with 0 for normal, or 1 for slow settling speed. Selecting “slow” will improve the phase noise. This is automatically set back to “fast” whenever a new frequency is tuned. (Not required for G526e/RMF)

To exit

If you want to turn all radios off, set the **Power** field of each radio in **RadioList** to 0, then call **G5_SetPower** to turn off all radios.

Call **G5_CloseRadios**

To check error status

If any hardware errors occur, API function returns -1 and the **Status** fields in the **RadioList** array will reveal which radio had the problem (zero ⇒ ok. Non-zero ⇒ error)

WRG526 API call summary

G5_FindRadios

```
int32_t G5_FindRadios(G5_RADIO *Radios uint32_t MaxNumberOfRadios,uint32_t
*RadiosFound)
```

Fills the user supplied array with a list of all WR-G526 class receivers connected to the PC.

The first entry will be the master receiver. Subsequent entries will be the slaves, sorted alphabetically by serial number.

See Appendix C for a description of model IDs and hardware differences.

G5_OpenRadios

```
int32_t G5_OpenRadios(G5_RADIO *Radios,uint32_t NumberOfRadios)
```

Call once to open communications with the radios.

G5_CloseRadios

```
int32_t G5_CloseRadios(G5_RADIO *Radios,uint32_t NumberOfRadios)
```

Call once to close communications with the radios.

G5_SetPower

```
int32_t G5_SetPower(G5_RADIO *Radios,uint32_t NumberOfRadios)
```

Each radio is switched off or on, according to the matching the power field in the array.

G5_SetFrequency

```
int32_t G5_SetFrequency(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t Frequency,
G5_STATUS *g5stat, G5_HW *g5hw,uint8_t Force)
```

Set all radios to receive the specified frequency.

Debugging information is returned in **g5stat**, and the contents of hardware registers in **g5hw**.

If bit 0 of **Force** is 1, then all hardware modules are reprogrammed.

If bit 0 of **Force** is 0, then only hardware modules which have changed are written to (which speeds up the tuning operation slightly)

If bit 7 of **Force** is 0, then the LO2 oscillator will be automatically toggled between high-side or low-side mixing to avoid spurii in the output.

If bit 7 of **Force** is 1, then the LO2 oscillator mixing mode will be set to the opposite of the automatic setting.

Additionally for G526e/RMF modules:

If bit 1 of **Force** is 1, the G5_HW data is sent directly to the hardware. The “Frequency” value is ignored.

If bit 2 of **Force** is 1, then the “spur-killing” circuitry is automatically activated at frequencies which might require it.

If bit 6 of **Force** is 1, the LO2 mixing mode tracks LO1 to ensure the output spectrum does not “invert” as the receive frequency is changed.

G5_SetRFAttn

```
int32_t G5_SetRFAttn(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t attn,
G5_STATUS *g5stat, G5_HW *g5hw)
```

Sets the level of attenuation in all slave radios, from 0 to 30dB in 2dB steps.

G5_SetIFGain

```
int32_t G5_SetIFGain(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t gain,
G5_STATUS *g5stat,G5_HW *g5hw)
```

Switches off or on a 20dB amplifier in the final IF stage of all slave radios.

G5_SetPhase

```
int32_t G5_SetPhase(G5_RADIO *Radios,uint32_t NumberOfRadios)
```

Set the phase-adjust DAC on each slave radio individually.

G5_GetHwStatus

```
int32_t G5_GetHwStatus(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t *Status)
```

This allows you to check if the master's PLLs are locked, and if any radios have been disconnected.

G5_UpdateHw

```
int32_t G5_UpdateHw(G5_RADIO *Radios,uint32_t NumberOfRadios,G5_HW *g5hw,uint8_t force)
```

Reprogram all hardware modules with data in **g5hw**.

WR-G526 API call detail

G5_FindRadios

Declaration

```
int32_t G5_FindRadios(G5_RADIO *Radios,uint32_t MaxNumberOfRadios,uint32_t
*RadiosFound);
```

Parameters

Radios

[in/out] Pointer to user supplied table which is initialized by this call. This parameter cannot be NULL.

MaxNumberOfRadios

[in] Max size of table (number of items, not bytes).

RadiosFound [out]

[out] Pointer to a variable that receives number of found G526 receivers. This parameter cannot be NULL.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero.

Description

The first entry of *Radios* will always be the master radio.
Subsequent entries will all be slaves, sorted alphabetically by serial number.
The *Handle*, *Status*, *Power* and *Phase* fields are all zeroed by this call.

G5_OpenRadios

Declaration

```
int32_t G5_OpenRadios(G5_RADIO *Radios,uint32_t NumberOfRadios);
```

Parameters

Radios

[in,out] Pointer to user suplied table, initialized by *G5_FindRadios*. This parameter cannot be NULL.

NumberOfRadios

[in] Number of items in *Radios* table (master + slaves).

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

Opens communication with receivers. The function changes value of *Handle* field of items in *Radios*. Non-zero value of *Handle* indicates the receiver is opened. The *Handle* field is zero for all receiver which failed.

G5_CloseRadios

Declaration

```
void G5_CloseRadios(G5_RADIO *Radios,uint32_t NumberOfRadios);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slave).

Return value

No return value.

Description

Closed communication with receivers. It sets *Handle* field of items in *Radios* to zero.

G5_SetPower

Declaration

```
int32_t G5_SetPower(G5_RADIO *Radios,uint32_t NumberOfRadios);
```

Parameters

Radios

[in] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

The function switches power off or on of each radio, according to the *Power* field of items in *Radios* table. To switch all the radios off or on, you have to set all the *Power* fields to 0 or 1 before the function is called.

G5_SetFrequency

Declaration

```
int32_t G5_SetFrequency(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t Frequency,
G5_STATUS *g5stat,G5_HW *g5hw,uint8_t Force);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

Frequency

[in] New receiver frequency, specified in Hertz.

g5stat

[in/out] Pointer to a G5_STATUS structure. This structure is updated to show the internal state of the receiver.
This parameter cannot be NULL.

g5hw

[in/out] Pointer to a G5_HW structure. This structure is updated to show the raw values written to receiver hardware. You must preserve this data, and pass it to the next G5_SetFrequency call for optimum tuning.
This parameter cannot be NULL.

Force

[in] Bit 0 set: force reprogram of all hardware.
Bit 7 set: toggle to alternate IF2 mixing mode.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

Set all radios to receive the specified frequency.

G5_SetRFAttn

Declaration

```
int32_t G5_SetRFAttn(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t attn,G5_STATUS
*g5stat,G5_HW *g5hw);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

attn

[in] New attenuation value, from 0 to 30 dB (in 2dB steps).

g5stat

[in/out] Pointer to a G5_STATUS structure. This structure is updated to show the internal state of the receiver.
This parameter cannot be NULL.

g5hw

[in/out] Pointer to a G5_HW structure. This structure is updated to show the raw values written to receiver hardware.
This parameter cannot be NULL.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

Sets the level of attenuation in all slave radios, from 0 to 30dB in 2dB steps.

G5_SetIFGain

Declaration

```
int32 G5_SetIFGain(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t gain,G5_STATUS
*g5stat,G5_HW *g5hw);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

gain

[in] If zero, normal gain.
If non zero, reduce IF gain by 20fB.

g5stat

[in/out] Pointer to a G5_STATUS structure. This structure is updated to show the internal state of the receiver.
This parameter cannot be NULL.

g5hw

[in/out] Pointer to a G5_HW structure. This structure is updated to show the raw values written to receiver hardware.
This parameter cannot be NULL.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

Switches off or on a 20dB amplifier in the final IF stage of all slave radios.

G5_SetPhase

Declaration

```
int32_t G5_SetPhase(G5_RADIO *Radios,uint32_t NumberOfRadios);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

Set the phase-adjust DAC on each slave radio individually. *Phase* field of items in *Radios* has to be set before the function is called.

G5_GetHwStatus

Declaration

```
int32_t G5_GetHwStatus(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t *Status);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

Status

[out] Pointer to a variable that receives hardware status.
Bits 0-15 set if radio N is busy. Bit 16 set if LO1 is locked.
Bit 17 set if LO2 is locked. Bits 18-31 unused.
This parameter cannot be NULL.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

The function allows to check if the master's PLLs are locked

G5_UpdateHw

Declaration

```
int32_t G5_UpdateHw(G5_RADIO *Radios,uint32_t NumberOfRadios,G5_HW *g5hw);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slaves).

g5hw

[in] Pointer to a G5_HW structure. All modules will be reprogrammed with data from this structure.
This parameter cannot be NULL.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

The function reprograms all hardware with data from the *g5hw* structure.

G5_SetPllSpeed

Declaration

```
int32_t G5_SetPllSpeed(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t SlowSpeed,
G5_STATUS *g5stat,G5_HW *g5hw);
```

Parameters

Radios

[in/out] Pointer to user supplied table, as used by G5_OpenRadios.
This parameter cannot be NULL.

NumberOfRadios

[in] Number of radios (master + slave).

SlowSpeed

[in] If zero, normal settling speed on LO1. If non-zero, slower settling speed on LO1 (less phase noise).

g5stat

[in/out] Pointer to a G5_STATUS structure. This structure us updated to show the internal state of the receiver.
This parameter cannot be NULL.

g5hw

[in/out] Pointer to a G5_HW structure. This structure is updated to show the raw values written to receiver hardware.
This parameter cannot be NULL.

Return value

If the function success the return value is zero, otherwise if function fails the return value is less than zero and non-zero value of *Status* field of items in *Radios* indicates which radios had the problem.

Description

The function sets LO1 PLL speed to normal or slow speed. Selecting slow speed will improve the phase noise. The PLL speed is automatically set back to normal whenever a new frequency is tuned. The function does nothing on version 1, 4, 5 and 6 radios (see Appendix C).

Appendix A: WR-G526 data structure definition – C/C++ format

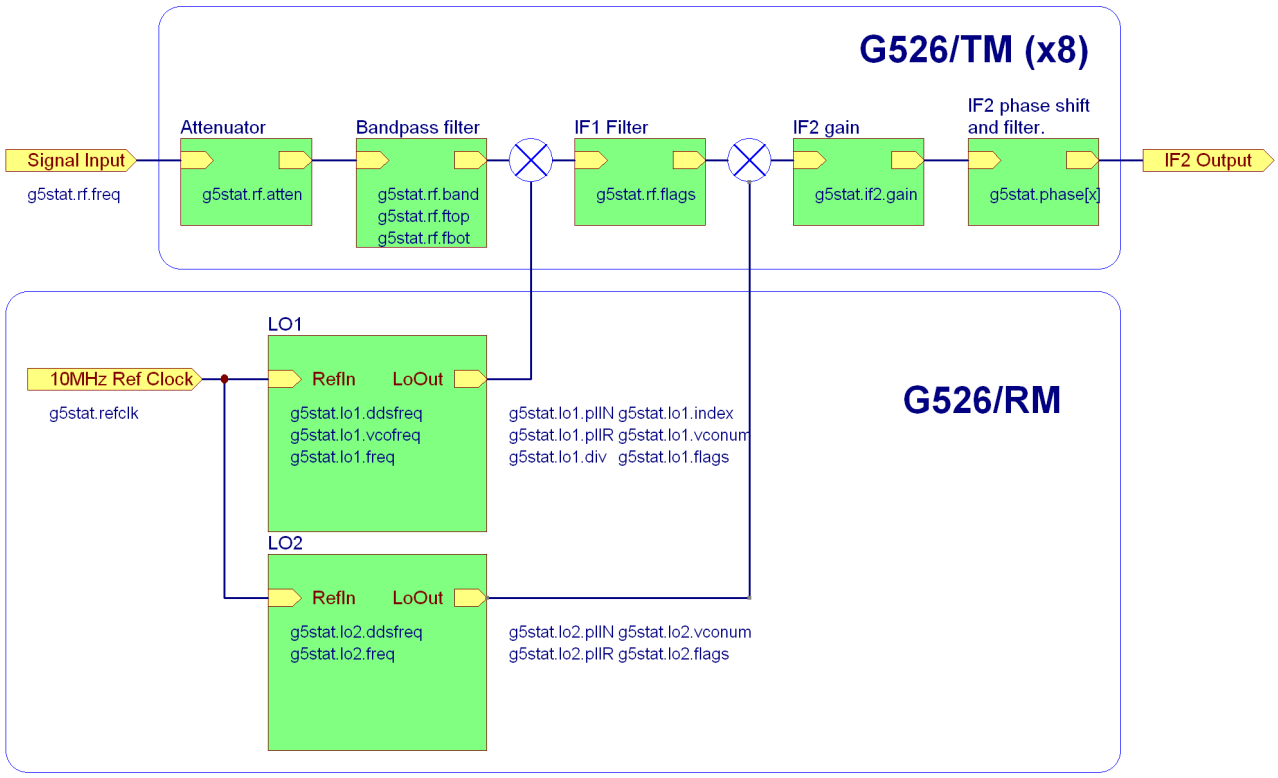
```
#pragma pack(push,1)

typedef struct
{
       char     SerNum[9];        // Serial Number + NULL
       uint8_t  Model;            // Bit0=1 => new PLL, Bit1=1 => combined
       uint8_t  Spare;            // unused
       int32_t  Handle;           // index into the D->Receiver[] table
       uint16_t Status;           // bit status
       uint8_t  Power;            // 0=off, 1=on
       uint8_t  Phase;            // phase adjustment (G526/TM only)
} G5_RADIO;

typedef struct
{
    uint32_t freq;     // receive frequency
    uint32_t freq_hi; // (reserved to extend frequency range)
    uint32_t fbot;     // bottom of current band filter
    uint32_t fbot_hi; // (reserved to extend frequency range)
    uint32_t ftop;     // top of current band filter
    uint32_t ftop_hi; // (reserved to extend frequency range)
    uint8_t  band;     // band#
    uint8_t  atten;    // Attenuation in dB
    uint8_t  filter;   // tuneable filter value
    uint8_t  flags;    // 0: 1 for preamp on,
                       // 1: 1 if manual filter setting,
                       // 2: 0=860MHz IF, 1=480MHz IF
} WR_RF;

typedef struct
{
    uint32_t freq;     // LO module output freq
    uint32_t ddsfreq; // dds output freq
    uint32_t vcofreq; // vco output freq
    uint16_t pllN;     // PLL main divider
    uint16_t pllR;     // PLL reference divider
    uint8_t  index;    // index in VCO lookup table
    uint8_t  vconum;   // which VCO
    uint8_t  div;      // always=0 => not used
    uint8_t  flags;    // 0: 0 for hi-side, 1 for lo-side
                       // 1: settling speed
                       // 2: 0 for 480MHz IF, 1 for 860 (LO2 only)
} WR_LO;

typedef struct
{
    uint8_t gain;
    uint8_t spare[3];             pad out to 4 bytes
} WR_IF;

typedef struct
{
    uint32_t refclk;
    WR_RF    rf;
    WR_LO    lo1;
    WR_LO    lo2;
    WR_IF    if2;
    uint8_t  phase[8];
} G5_STATUS;
```

```
typedef struct
{
	uint32_t LO1_PLL;		// MASTER	(i2c-0xC0)
	uint32_t LO1_DDS;		// MASTER	(not used)
	uint32_t LO2_PLL;		// MASTER	(i2c-0xC2)
	uint32_t LO2_DDS;		// MASTER	(CS-IF2)
	uint8_t  RF_BAND;		// SLAVES	(SR LSB)
	uint8_t  RF_ATTN;		// SLAVES	(SR MSB)
	uint8_t  IF2_GAIN;		// SLAVES	(IF2-STB)
	uint8_t  LO1_PCA;		// MASTER
	uint32_t LO1_DACS;		// MASTER
	uint32_t LO2_DACS;		// MASTER
	uint32_t SPURA;			// MASTER
	uint32_t SPURB;			// MASTER

} G5_HW;

#pragma pack(pop)
```

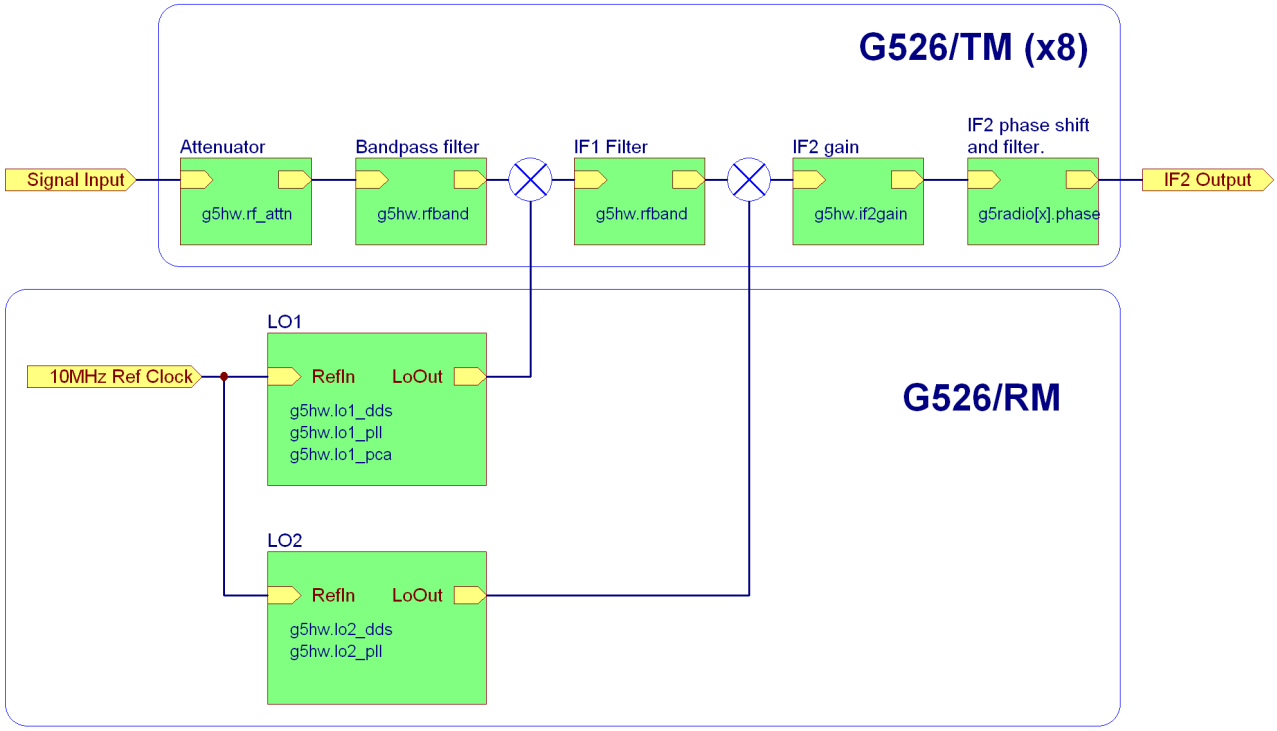
Appendix B: Data structure detail

This diagram illustrates which module each item in the G5_STATUS structure applies to.

Note that TM modules are all programmed with the same data, except for the phase adjust register.



This diagram illustrates which module each item in the G5_HW structure applies to.



Appendix C: Model variations

There have been several hardware variants of the WR-G526/RM as we strive to improve this product.

The byte returned in the "Model" field of the first entry in the **G5_RADIO** structure indicates which version it is. All WR-G526/TM (slave) units produced to date will return zero in this field.

Currently, bit-1 of the model field is set if this is a "combined" unit (WR-G526/SC), which incorporates one reference module, and one tuner module into a single unit.

Bits 0, 2, and 3 indicate the version as follows:

Bits	LO1 version	Comments
00x0	1 (PLL only)	Tuning resolution is 1 MHz
00x1	2 (DDS & PLL)	Tuning resolution is 10 Hz. Small steps by adjusting LO1-DDS Fast/slow PLL settling option present, to optimise either tuning speed or phase noise.
01x0	3 (PLL only)	Tuning resolution is 10 Hz. Small steps by adjusting LO2-DDS Fast/slow PLL settling option present
01x1	3b (PLL only)	This is a small variation on version 3
11x0	4 (PLL only, new loop filter)	Tuning resolution is 10 Hz. Small steps by adjusting LO2-DDS Fast/slow PLL settling option NOT present (Always fast, with improved phase noise also.) LO2 also has a new loop filter for reduced phase noise.

Unused bit combinations are reserved for future use.

Bits 4, 5, 6, and 7 are always zero, and are reserved for future use.

Note that for version 3 and later LO1, frequency steps between 1MHz multiples are implemented by shifting LO2 slightly.

Version 2 and version 3 have a slow/fast option on the LO1 PLL. This is automatically set to "fast" whenever a new frequency is selected to give fast stepping. Call **G5_SetPllSpeed** with a parameter of 1 to select "slow" for reduced phase noise when you are stopped on a frequency.

Version 4 has an improved loop filter, which gives fast stepping with reduced phase noise at all times. **G5_SetPllSpeed** has no effect on version 4 boards.

Appendix D: Output Spectrum Inversion

The WR-G526 is a dual conversion receiver.

To minimise the output of internal spurious frequencies ("birdies"), the local oscillators automatically switch between "high-side" mixing and "low-side" mixing.

If both LOs are mixing from the same side, the output spectrum will be "normal". If one is "high-side" and the other is "low-side", then the output spectrum will be "mirrored".

The G5_STATUS structure contains a bit for each LO indicating which mixing side is being used. This is bit 0 of **lo1.flags** and **lo2.flags** respectively. A Boolean XOR of these two bits will give 0 for true output, or 1 for mirrored output.

Appendix E: Loading API

Header file (wrg526api.h)

```
#ifndef __WRG526API_H__
#define __WRG526API_H__

#include <stdint.h>

#pragma pack(push,1)

typedef struct
{
    char        SerNum[9];
    uint8_t     Model;
    uint8_t     Spare;
    int32_t     Handle;
    uint16_t    Status;
    uint8_t     Power;
    uint8_t     Phase;
} G5_RADIO;

typedef struct
{
    uint32_t freq;    // receive frequency
    uint32_t freq_hi; // (reserved to extend frequency range)
    uint32_t fbot;    // bottom of current band filter
    uint32_t fbot_hi; // (reserved to extend frequency range)
    uint32_t ftop;    // top of current band filter
    uint32_t ftop_hi; // (reserved to extend frequency range)
    uint8_t  band;    // band#
    uint8_t  atten;   // Attenuation in dB
    uint8_t  filter;  // tuneable filter value
    uint8_t  flags;   // 0: 1 for preamp on,
                      // 1: 1 if manual filter setting,
                      // 2: 0=860MHz IF, 1=480MHz IF
} WR_RF;

typedef struct
{
    uint32_t freq;    // LO module output freq
    uint32_t ddsfreq; // dds output freq
    uint32_t vcofreq; // vco output freq
    uint16_t pllN;
    uint16_t pllR;
    uint8_t  index;   // index in VCO lookup table
    uint8_t  vconum;  // which VCO
    uint8_t  div;     // always=0 => not used
    uint8_t  flags;   // 0: 0 for hi-side, 1 for lo-side
                      // 1: settling speed
                      // 2: 0 for 480MHz IF, 1 for 860 (LO2 only)
} WR_LO;

typedef struct
{
    uint8_t gain;
    uint8_t spare[3];
} WR_IF;

typedef struct
{
    uint32_t refclk;
    WR_RF    rf;
    WR_LO    lo1;
    WR_LO    lo2;
    WR_IF    if2;
    uint8_t  phase[8];
} G5_STATUS;
```

```
typedef struct
{
    uint32_t   LO1_PLL;   // MASTER  (i2c-0xC0)
    uint32_t   LO1_DDS;   // MASTER  (not used in ver1)
    uint32_t   LO2_PLL;   // MASTER  (i2c-0xC2)
    uint32_t   LO2_DDS;   // MASTER (CS-IF2)
    uint8_t    RF_BAND;   // SLAVES (SR LSB)
    uint8_t    RF_ATTN;   // SLAVES (SR MSB)
    uint8_t    IF2_GAIN;  // SLAVES (IF2-STB)
    uint8_t    LO1_PCA;   //MASTER (only for ver2)
    uint32_t   LO1_DACS;
    uint32_t   LO2_DACS;
    uint32_t   SPURA;
    uint32_t   SPURB;
} G5_HW;

#pragma pack(pop)

#ifdef __cplusplus
extern "C"
{
#endif

typedef int32_t (*G5_FIND_RADIOS)(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t
*RadiosFound);
typedef int32_t (*G5_OPEN_RADIOS)(G5_RADIO *Radios,uint32_t NumberOfRadios);
typedef void (*G5_CLOSE_RADIOS)(G5_RADIO *Radios,uint32_t NumberOfRadios);
typedef int32_t (*G5_SET_POWER)(G5_RADIO *Radios,uint32_t NumberOfRadios);
typedef int32_t (*G5_SET_FREQUENCY)(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t
Frequency,G5_STATUS *Status,G5_HW *Hw,uint8_t Force);
typedef int32_t (*G5_SET_RF_ATTN)(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t
Attn,G5_STATUS *Status,G5_HW *Hw);
typedef int32_t (*G5_SET_IF_GAIN)(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t
Gain,G5_STATUS *Status,G5_HW *Hw);
typedef int32_t (*G5_SET_PHASE)(G5_RADIO *Radios,uint32_t NumberOfRadios);
typedef int32_t (*G5_SET_PLL_SPEED)(G5_RADIO *Radios,uint32_t NumberOfRadios,uint8_t
SlowSpeed,G5_STATUS *Status,G5_HW *Hw);
typedef int32_t (*G5_GET_HW_STATUS)(G5_RADIO *Radios,uint32_t NumberOfRadios,uint32_t
*Status);
typedef int32_t (*G5_UPDATE_HW)(G5_RADIO *Radios,uint32_t NumberOfRadios,G5_HW *Hw);

#ifdef __cplusplus
};
#endif

extern G5_FIND_RADIOS G5_FindRadios;
extern G5_OPEN_RADIOS G5_OpenRadios;
extern G5_CLOSE_RADIOS G5_CloseRadios;
extern G5_SET_POWER G5_SetPower;
extern G5_SET_FREQUENCY G5_SetFrequency;
extern G5_SET_RF_ATTN G5_SetRFAttn;
extern G5_SET_IF_GAIN G5_SetIFGain;
extern G5_SET_PHASE G5_SetPhase;
extern G5_SET_PLL_SPEED G5_SetPllSpeed;
extern G5_GET_HW_STATUS G5_GetHwStatus;
extern G5_UPDATE_HW G5_UpdateHw;

uint8_t LoadG526API(void);
void UnloadG526API(void);

#endif
```

C file

```
#include "wrg526api.h"
#include <dlfcn.h>
#include <stdio.h>
#include <errno.h>

G5_FIND_RADIOS G5_FindRadios=NULL;
G5_OPEN_RADIOS G5_OpenRadios=NULL;
G5_CLOSE_RADIOS G5_CloseRadios=NULL;
G5_SET_POWER G5_SetPower=NULL;
G5_SET_FREQUENCY G5_SetFrequency=NULL;
G5_SET_RF_ATTN G5_SetRFAttn=NULL;
G5_SET_IF_GAIN G5_SetIFGain=NULL;
G5_SET_PHASE G5_SetPhase=NULL;
G5_SET_PLL_SPEED G5_SetPllSpeed=NULL;
G5_GET_HW_STATUS G5_GetHwStatus=NULL;
G5_UPDATE_HW G5_UpdateHw=NULL;

void *WRG526API=NULL;

#define GET_PROC(t,f)               \
    f=(t)dlsym(WRG526API,"" #f);    \
    if(f==NULL)                     \
    {                               \
                                    \
        fprintf(stderr,"Unable to load valid wrg526api.so library.\n"); \
        fprintf(stderr,"Unable to find " #f " procedure.\n");\
        dlclose(WRG526API);         \
        WRG526API=NULL;             \
        return 0;                   \
    }

//LoadG526API return 1 if success, 0 if failed
uint8_t LoadG526API(void)
{
    if(WRG526API==NULL)
    {
        WRG526API=dlopen("wrg526api.so",RTLD_LAZY);

        if(WRG526API==NULL)
        {
            fprintf(stderr,"Unable to load wrg526api.so shared library
(%s).\n",dlerror());
            return 0;
        }
        else
        {
            GET_PROC(G5_FIND_RADIOS,G5_FindRadios)
            GET_PROC(G5_OPEN_RADIOS,G5_OpenRadios)
            GET_PROC(G5_CLOSE_RADIOS,G5_CloseRadios)
            GET_PROC(G5_SET_POWER,G5_SetPower)
            GET_PROC(G5_SET_FREQUENCY,G5_SetFrequency)
            GET_PROC(G5_SET_RF_ATTN,G5_SetRFAttn)
            GET_PROC(G5_SET_IF_GAIN,G5_SetIFGain)
            GET_PROC(G5_SET_PHASE,G5_SetPhase)
            GET_PROC(G5_SET_PLL_SPEED,G5_SetPllSpeed)|
            GET_PROC(G5_GET_HW_STATUS,G5_GetHwStatus)
            GET_PROC(G5_UPDATE_HW,G5_UpdateHw)

            return 1;
        }
    }
    else
    {
        return 1;
    }
}
```

```
void UnloadG526API(void)
{
    if(WRG526API!=NULL)
    {
        dlclose(WRG526API);
        WRG526API=NULL;
    }
}
```