DigiMMIC Driver

dmdriver-src-3.3.201019.986

Generated by Doxygen 1.8.14

Contents

Chapter 1

Introduction

1.1 DigiMMIC driver

The DigiMMIC driver implements an Application Programming Interface (API) to control one or more ADAR6901/↩ ADAR6902 (DigiMMIC) devices. The driver runs on a host processor connected to the ADAR6901/ADAR6902 devices via SPI and GPIO for control, and either MIPI CSI-2 (ADAR6901) or LVDS (ADAR6902) to receive ADC data.

The driver is distributed as host processor and operating system independent standard C99 source code. This independence is achieved through the use of a Hardware Abstraction Layer (HAL), a clearly defined API that abstracts the host resources which must be provided by the user. The HAL is described in Hardware Abstraction Layer (HAL).

The API implemented by the driver may be divided into high-level and low-level routines. High-level routines provide a significant abstraction of the ADAR6901/ADAR6902 devices and are described in High-level Interface. The driver can perform built in self tests (BISTs) which require additional high-level routines described in Built-in-self-tests (BISTs). Low-level routines provide direct access to device registers and memories and are described in Low-level Interface.

The driver supports configurations including one ADAR6901/ADAR6902 or two or four ADAR6901/ADAR6902 devices connected as described in the "Cascading" section of [2]. The representation of multiple devices within the API is described in Multi-DigiMMIC support.

Compile time definitions of the API appear in the header file adi_dmdriver.h, which can be included in user application source code. The driver sources should be compiled and linked into the user application. To get started, a "Makefile" is included in the release package to build a library from driver source code. This will need to be adapted to the customer toolchain.

Identifiers in the driver are prefixed to ease integration with user code at build and link time. All global identifiers and all identifiers defined in public header files are prefixed with `ADI_` or `adi_`.

Host independence, language standards and automotive coding conventions impose a certain style on the driver. In particular:

- No threads are created.
- There is no interfacing with concurrency control primitives at all, client code is responsible for avoiding race conditions.
- Dynamic memory is not used.
- Errors are signalled by return codes only. See Error handling
- Setjmp/longjmp or C extensions are not used.

A minimal driver for the ADP5140 Power Management IC (PMIC) is included in the release, sufficient for control of the PMIC on our development boards. This is defined in adi_pmic_driver.h.

Refer to [2] for detailed description of the ADAR6901/ADAR6902 hardware and functionality.

Refer to [1] for details of the commands supported by the firmware that runs on the ADAR6901/ADAR6902.

1.2 High-level Interface

The high level API provides functions for control tasks. These may be used to initialize the ADAR6901/ADAR6902 in the following steps:

1. Initialize Driver

- adi_dm_InitDriver — Initialize driver.

2. Power Up Device

- adi_dm_PowerUp — Power up device.

This function initializes ADCPLL and RFPLL, providing initial ramp start frequency and ramp bandwidth parameters to the firmware. Subsequent changes to these parameters may be made by calling adi_dm_RfpllReconfig.

3. Configure Dataport

- adi_dm_MipiSetup — Configure Dataport on ADAR6901.
- adi_dm_LvdsSetup — Configure Dataport on ADAR6902.

4. Optionally configure Analog Front End (AFE) and TX Channels

- adi_dm_AfeSetup — Optionally configure AFE.
- adi_dm_TxSetup — Optionally configure TX Channels.

At this point adi_dm_WriteSysCalRx and adi_dm_WriteSysCalTx should be called if required. If using the default AFE and Tx settings they should be called after adi_dm_PowerUp if required. See Built-in-self-tests (BISTs).

5. Program Ramp Generator

The Hardware Reference Manual [2]. describes two methods for programming the ramp generator. The simplest loads the RAMPGEN registers directly and uses builtin MIMO mode.

- adi_dm_BuiltinMimoSetup — Program RAMPGEN using Builtin MIMO Mode.

A more powerful method is to store the values to be loaded to RAMPGEN registers to the device's memory and use DMA to load the registers during transmission, so each ramp can have a different profile. The UDMA engine in the ADAR6901/ADAR6902 is general purpose and there many possible ways to organize these values in memory. The driver supports one set of possibilities which it abstracts as a data type called a burst profile.

A ramp profile is a single set of values to be loaded into the registers. Each time a ramp is transmitted the next ramp profile is loaded into the RAMPGEN registers. A burst profile consists of a sequence of ramp profiles which is repeated until the end of the burst. Multiple burst profiles are stored in memory, and a current burst profile is specified. When a burst is triggered the rampgen registers are loaded from the current burst profile.

- adi_dm_BurstProfileSetup — Stores a burst profile to device memory.
- adi_dm_DmaRampSetup — Programs RAMP_CONFIG registers, DMA, and sets the current burst profile.

Between bursts a new current burst profile can be selected by calling adi_dm_SelectBurstProfile which switches profile with just a few SPI writes.

In the above function calls ramps are represented by adi_dm_ramp_profile_t which uses the same representation of values as the RAMPGEN registers themselves. See [2] for register descriptions. The adi_dm_ramp_profile_t representation can be generated from a higher level description in which ramp times are measured in microseconds and bandwidth in hertz using adi_dm_CalcRamp which can be run offline.

Because of hardware limitations, the ramp profile may not *exactly* match what was requested. In particular, the "AFE Timing" subsection of [2] states that *the duration of each ramp must be an integer multiple of the CLK and AFE_CLK periods.*

- adi_dm_CalcRamp — Calculate ramp profile from high level parameters.

After programming the ramps and before any ramps are triggered adi_dm_WriteRfpllPeriod, adi_dm_PowerDetectorMeasTask, and adi_dm_LockConfig should be called if required. See Built-in-self-tests (BISTs).

6. Trigger a burst of ramps

- adi_dm_Trigger — Trigger a single burst by calling the firmware trigger command.

Bursts can also be triggered using the TRIG pin. See "Input/Output Pad Control and General-Purpose Input/Output" in [2].

7. Collect ADC data

If the above steps are followed, once a burst is triggered ADC data will be transmitted to the host processor via the MIPI CSI-2 or LVDS connection. The details of how this data is received depends entirely upon host hardware and drivers so is not abstracted in the driver source code. This must be coded in the user application.

8. Low power state

It is possible to save power by entering a low power state between bursts. This state must be exited before executing any subsequent driver calls.

- adi_dm_ManualSleep — Enter low power state.
- adi_dm_ManualWake — Exit low power state.

9. Prepare for next burst

Between bursts it is usually necessary to perform recalibration and reacquisition.

- adi_dm_PeriodicCalibration — Perform periodic firmware (re-)calibration as recommended by [1].

This function can also perform BISTs. See Built-in-self-tests (BISTs).

10. Reconfiguration

The application may also require a change to the configuration. Prior to reconfiguring adi_dm_UnlockConfig should be called if required. See Built-in-self-tests (BISTs).

- adi_dm_SelectBurstProfile — change current burst profile.
- adi_dm_RfpllReconfig — change ramp start frequency and firmware ramp bandwidth parameter.
- adi_dm_RfpllLock — change ramp start frequency only.

Other aspects of the ramp configuration can be changed by calling adi_dm_BuiltinMimoSetup or adi_dm_DmaRampSetup again.

11. Power down device

- adi_dm_PowerDown — Power down ADAR6901/ADAR6902 devices.

12. Exit application

- adi_dm_FiniDriver — Release any resources used by driver.

Notes on High-level Interface

In an effort to avoid long parameter lists, a struct is commonly passed to functions. *All* elements of such a struct should be filled in. This is most conveniently done by zero-initializing the entire structure, for instance with a call to `memset`. This ensures that as the API evolves and new structure members are added existing code initializes these to zero. The addition of a new parameter structure member which invokes the previous functionality when its value is zero is considered a compatible change.

1.3 Tasklists

The high level APIs write to registers and call firmware tasks which takes many SPI transactions. The same result can be achieved more efficiently using tasklists which are lists of firmware tasks stored to device memory which can be executed with a single firmware command. Currently only a limited low level interface is provided for building tasklist. Refer to examples/example_tasklists.c for examples of use and a higher level interface to tasklist construction.

- adi_dm_TasklistPoolSetup — Write tasklists to device memory.
- adi_dm_Tasklist — Execute a tasklist.
- adi_dm_TasklistNoBlock — Execute a tasklist without waiting for it to finish.

1.4 Built-in-self-tests (BISTs)

In addition to calibrating the device adi_dm_PowerUp and adi_dm_PeriodicCalibration can execute BISTs to ensure the device is running correctly. adi_dm_PowerUp runs BISTs by default as the power-up BISTs do not depend upon additional state. adi_dm_PeriodicCalibration only runs BISTs if the parameter member `run_checks` is set true. Furthermore it will only run BISTs that depend upon adi_dm_PowerDetectorMeasTask if the parameter member `run_power_checks` is set true. Note run_power_checks is ignored when run_checks is false. If these options are selected further API calls are required.

Locking the configuration

Some BISTs check the device memory has not been corrupted, using CRCs, so an API is provided to indicate that the ADAR6901/ADAR6902 devices are fully configured and memory is no longer expected to change.

When adi_dm_PeriodicCalibration is called parameter member `run_checks` is set true the following APIs must be used.

- adi_dm_LockConfig — Indicate devices are fully configured and memory is not expected to change.
- adi_dm_UnlockConfig — Indicate reconfiguration is about to start and memory is expected to change.

Power Tests

Some BISTs check power levels during a burst of ramps and so require initialization before a burst is triggered.

When adi_dm_PeriodicCalibration is called parameter member `run_checks` and `run_power_checks` are set true the following APIs must be used.

- adi_dm_PowerDetectorMeasTask — Initialize gathering of Tx power levels for subsequent BISTs.

The parameter to this function describes the current burst, and may be computed by adi_dm_CalcPwrDetCfg.

RFPLL Period Check

BIST 103c compares a counter in the RFPLL against an estimate based on burst length and frequency.

When adi_dm_PeriodicCalibration is called parameter member `run_checks` and `run_rfpll_period_chk` are set true the following APIs must be used.

- adi_dm_WriteRfpllPeriod — Write the user estimated RFPLL period count for the current burst.

The estimate may be computed by calling adi_dm_CalcRfpllPeriod.

'End of line' t0 measurements (SysCal)

A further set of BISTs compare against so called 't0 measurements' characterizing the performance of the entire system at the time it was built. These measurements need to be collected after each system is built and stored for use at runtime. These measurements are also called 'end of line' because they are run at the end of the factory production line where the system is manufactured, and miscalled 'system calibrations'.

When adi_dm_PeriodicCalibration is called parameter member `run_checks` and `run_power_checks` are set true the following APIs must have been called after.

- adi_dm_WriteSysCalRx — Initialize Rx side t0 power measurements for subsequent BISTs.
- adi_dm_WriteSysCalTx — Initialize Tx side t0 power measurements for for subsequent BISTs.

Additional APIs are provided specifically for use in the software that gathers t0 measurements. The following APIs are specifically for use in the t0 measurement software.

- adi_dm_SetSysCal — Enable t0 measurement mode which changes the function of some firmware calls.
- adi_dm_ReadSysCalRx — Read Rx side t0 power measurements from the devices in a form to pass to adi_dm_WriteSysCalRx.
- adi_dm_ReadSysCalTx — Read Tx side t0 power measurements from the devices in a form to pass to adi↩ _dm_WriteSysCalTx.

See the example `example_syscal.c` for an overview of the function of t0 measurement software. Essentially it configures the devices as in normal mode but calls adi_dm_SetSysCal to change the function of some firmware calls. It then runs bursts with setting that match the configurations to be used in normal mode, in the field. After the data has been gathered it is read back by calling adi_dm_ReadSysCalTx and adi_dm_ReadSysCalRx.

If multiple usecases are to be used in normal operation where the tx configuration is different, then when in syscal mode, a separate syscal tx profile should be saved for each use case and adi_dm_WriteSysCalTx should be called with new values when switching between use cases.

1.5 Low-level Interface

The low level API enables direct reads and writes of the ADAR6901/ADAR6902 address space and direct calls to the firmware running on the device. These functions are used to implement the high-level API and may also be called directly from application code to provide full control over the device.

- adi_dm_CallFW — Execute a firmware task. See [1].
- adi_dm_Write — Write a word to the ADAR6901/ADAR6902 address space.
- adi_dm_Read — Read a word from the ADAR6901/ADAR6902 address space.
- adi_dm_RMW — Read-modify-write.
- adi_dm_BlockWrite — Write a memory region in the ADAR6901/ADAR6902 address space.
- adi_dm_BlockRead — Read a memory region from the ADAR6901/ADAR6902 address space.

As a convenience a number of typed memory access functions are implemented in terms adi_dm_Write and adi_dm_Read above.

- adi_dm_WriteF32 — Write `float` to the DigiMMIC address space.
- adi_dm_WriteF64 — Write `double` to the DigiMMIC address space.
- adi_dm_WriteU64 — Write `uint64_t` to the DigiMMIC address space.
- adi_dm_ReadF32 — Read `float` from the DigiMMIC address space.
- adi_dm_ReadF64 — Read `double` from the DigiMMIC address space.
- adi_dm_ReadU64 — Read `uint64_t` value from the DigiMMIC address space.

Another convenience function is provided to control the function of an IO pad.

- adi_dm_PinMux — Control function of an IO pad.

Knowledge of [1] and [2] is needed to use the low-level interface effectively. The information from [1] is summarized in adar690x_fw.h.

1.6 Hardware Abstraction Layer (HAL)

The PMIC and DigiMMIC drivers call an API to use hardware and operating systems resources of the host system. The API is defined in the file adi_dmhal.h.

These functions must be provided by user.

- adi_dm_WaitGPIO — Wait for a GPIO to assume a particular value, with timeout.
- adi_dm_WriteGPIO — Set a GPIO to a particular value.
- adi_dm_ReleaseGPIO — Tri-state a GPIO.
- adi_dm_DelayNS — Delay for specified time.
- adi_dm_SPI — Execute a SPI transfer.
- adi_dm_Log — Write some tracing.
- adi_dm_PowerUpSupplies — Power up power supplies.
- adi_dm_PowerDownSupplies — Power down power supplies.

adi_dm_SPI is a single function that transmits a byte string. This can be a simple interface to implement, however sometimes host SPI devices and drivers impose further constraints on the transmitted data and it is more convenient to replace the code that formats byte strings for the ADI SPI slave IP used in DigiMMIC and PMIC devices.

These functions are implemented in the file `spicmd.c` and may be replaced by the user as an alternative to providing adi_dm_SPI if that is more convenient. Both PMIC and DigiMMIC driver call this high-level interface to format byte strings and communicate with devices with ADI SPI slave IP over the SPI bus.

- adi_dm_WriteSPI — Write to a remote SPI device with ADI SPI slave IP.
- adi_dm_ReadSPI — Read from a remote SPI device with ADI SPI slave IP.
- adi_dm_ResetSPIConnection — Set local model of remote ADI SPI slave IP to power on state.
- adi_dm_InitSPIConnection — Initialize the connection to remote ADI SPI slave IP.

1.7 Multi-DigiMMIC support

Multiple DigiMMIC chips cascaded together are supported in software. However, high- and low-level APIs look slightly different.

The high-level API programs all the DigiMMICs as a single unit. Where different parameters may be applied to individual devices they are split out into a parameter array indexed by device number, which is one of `ADI_DM_↩ MASTER`, `ADI_DM_SLAVE1`, `ADI_DM_SLAVE2` or `ADI_DM_SLAVE3`.

The low-level API functions take an argument that specifies which device to use. The device may be specified directly as `ADI_DM_MASTER`, `ADI_DM_SLAVE1`, `ADI_DM_SLAVE2` or `ADI_DM_SLAVE3`. The special value `ADI_DM_ALL_DIGIMMICS` selects all DigiMMICs for adi_dm_Write, adi_dm_BlockWrite, adi_dm_RMW, adi_dm_CallFW and adi_dm_PinMux but not adi_dm_Read and adi_dm_BlockRead.

In both APIs the number of devices controlled by the driver is determined statically at build time with the constant ADI_DM_NUM_DIGIMMIC and at run time by the global variable adi_dm_active_digimmics.

1.8 Error handling

Errors are signalled by return codes only. The values are members of the adi_dm_err_t enum. Setjmp/longjmp or C extensions are not used in case customers want to harden the code to MISRA guidelines.

Chapter 2

Driver Examples

Source Files

Source File	Comment
example_dma_ramp.c	Full example transmitting ramps using DMA mode.
example_tasklist.c	A functionally equivalent example using tasklists.
example_syscal.c	Example of system calibration and t0 measurement.
example_init.c	Common initialization routine used by all examples.
example_burst_loop.c	Measurement loop routine used by example_dma_ramp.c.
example_main.c	A main() to call the examples.
application.h	Declarations for the examples.
example_burst_loop.c	Measurement loop routine used by example_dma_ramp.c.
tasklist_util.c	Utilities for building tasklists.
tasklist_util.h	Header file for utilities for building tasklists.
f2decl.c	Utility to convert binary file to C initialization.
platforms/win/*	Files to build the example with Visual Studio on Windows
platforms/win/example.sln	*VisualStudio* Solution file to build example and driver
platforms/win/dummy_hal.c	Stub Hardware Abstraction Layer (HAL) routines
platforms/win/dummy_↩ platform.c	Stub Platform routines
platforms/win/platform.h	Platform specific declarations.

The examples

Three examples of the driver API are provided in the directory `driver/examples`. `example_dma_ramp.c` shows how to generate ramps using DMA mode using the driver API. In particular adi_dm_PeriodicCalibration is used to run calibrations between bursts. `example_tasklist.c` also generates ramps using DMA but uses tasklists to run the calibrations between bursts and to switch between burst profiles. `example_syscal.↩ c` shows how to perform 'end of line system calibration' to collect t0 (time zero) measurements at the end of the systems production line. This data is used in some built-in self-tests (BISTS). Much of these examples are the same and have been broken out into common subroutines, `example_init.c` contains the common initialization and `example_burst_loop.c` contains the steady-state code for `example_dma_ramp.c`.

Platform specific code

The driver API is only concerned with controlling ADAR690x devices. Other parts of the application are concerned with controlling the host platform. These are represented by calls in the example code to routines with names beginning `platform_`.

Routine	Comment
platform_init	Initialize drivers etc. on the host platform
platform_data_plane_init	Further initialization once device is initialized
platform_error	Report an error and terminate program
platform_prep_host_for_trigger	Called before each trigger to prepare host for next burst
platform_start_timer	Start a timer running in the background
platform_wait_for_timer	Wait for the above timer or return immediately if expired

The Hardware Abstraction Layer (HAL) contains the platform specific code called by the driver itself.

Firmware in the examples

The firmware must be loaded into the host platform memory before calling adi_dm_PowerUp(). The declaration of this memory region may depend upon how the platform code loads the firmware so it has to appear in the file `platform.h` included by `application.h`.

The host is an embedded system so it may not be convenient to load the firmware at runtime. The program `f2decl.c` which converts a binary file to C declarations is provided to support this scenario. Nothing is assumed about the host system except it has a C compiler so the program is distributed as C source code.

Example of `f2decl.c` use:

1. Compile and link f2decl.c to f2decl.exe
2. f2decl.exe adar690x_ICCMRAM.bin > adar690x_ICCMRAM.bin.c
3. f2decl.exe adar690x_DCCMRAM.bin > adar690x_DCCMRAM.bin.c

The example *VisualStudio* solution "examples\platforms\win\example.sln" performs these steps automatically.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 adi_dm_actual_ramp_shape_t Struct Reference

High level description of ramp shape output by adi_dm_CalcRamp()

```
#include <adi_dmdriver.h>
```

Data Fields

- float slope0_time_us

  Actual time for first slope of ramp, in microseconds.
- float slope1_time_us

  Actual time for second slope of ramp, in microseconds.
- float delay0_time_us

  Actual time for delay before `slope0`, in microseconds.
- float delay1_time_us

  Actual time for delay between `slope0` and `slope1`, in microseconds.
- float delay2_time_us

  Actual time for delay after `slope1`, in microseconds.
- float ramp_bw_Mhz

  Actual difference between frequency of `delay1` and `delay0`, in megahertz.
- float afe_start_off_time_us

  Duration of `afe_stat_off_time` in microseconds.
- float filter_valid_delay_time_us

  Duration of `filter_valid_delay` in microseconds.
- float sample_time_us

  Duration of the sampling period in microseconds.
- float sample_start_time_us

  Actual time to start sampling.
- float afe_ramp_time_us

  Ramp time calculated from AFE clk.
- bool pga_shunt_en

  PGA Shunt enabled.
- float pga_shunt_start_del_us

  Actual start delay for PGA Shunt, in microseconds.

- float pga_shunt_stop_del_us

  Actual stop delay for PGA Shunt, in microseconds.
- float afe_freq_Mhz

  AFE CLK frequency, in megahertz.
- float sample_rate_Mhz

  Sample rate in megahertz.

5.1.1 Detailed Description

High level description of ramp shape output by adi_dm_CalcRamp()

See also

adi_dm_CalcRamp

5.1.2 Field Documentation

5.1.2.1 afe_freq_Mhz

`float adi_dm_actual_ramp_shape_t::afe_freq_Mhz`

AFE CLK frequency, in megahertz.

5.1.2.2 afe_start_off_time_us

`float adi_dm_actual_ramp_shape_t::afe_start_off_time_us`

Duration of `afe_stat_off_time` in microseconds.

5.1.2.3 delay0_time_us

`float adi_dm_actual_ramp_shape_t::delay0_time_us`

Actual time for delay before `slope0`, in microseconds.

5.1.2.4 delay1_time_us

```
float adi_dm_actual_ramp_shape_t::delay1_time_us
```

Actual time for delay between `slope0` and `slope1`, in microseconds.

5.1.2.5 delay2_time_us

```
float adi_dm_actual_ramp_shape_t::delay2_time_us
```

Actual time for delay after `slope1`, in microseconds.

5.1.2.6 filter_valid_delay_time_us

```
float adi_dm_actual_ramp_shape_t::filter_valid_delay_time_us
```

Duration of `filter_valid_delay` in microseconds.

5.1.2.7 pga_shunt_en

```
bool adi_dm_actual_ramp_shape_t::pga_shunt_en
```

PGA Shunt enabled.

5.1.2.8 pga_shunt_start_del_us

```
float adi_dm_actual_ramp_shape_t::pga_shunt_start_del_us
```

Actual start delay for PGA Shunt, in microseconds.

5.1.2.9 pga_shunt_stop_del_us

```
float adi_dm_actual_ramp_shape_t::pga_shunt_stop_del_us
```

Actual stop delay for PGA Shunt, in microseconds.

5.1.2.10 ramp_bw_Mhz

```
float adi_dm_actual_ramp_shape_t::ramp_bw_Mhz
```

Actual difference between frequency of `delay1` and `delay0`, in megahertz.

5.1.2.11 sample_rate_Mhz

```
float adi_dm_actual_ramp_shape_t::sample_rate_Mhz
```

Sample rate in megahertz.

5.1.2.12 sample_start_time_us

```
float adi_dm_actual_ramp_shape_t::sample_start_time_us
```

Actual time to start sampling.

5.1.2.13 sample_time_us

```
float adi_dm_actual_ramp_shape_t::sample_time_us
```

Duration of the sampling period in microseconds.

5.1.2.14 slope0_time_us

```
float adi_dm_actual_ramp_shape_t::slope0_time_us
```

Actual time for first slope of ramp, in microseconds.

5.1.2.15 slope1_time_us

```
float adi_dm_actual_ramp_shape_t::slope1_time_us
```

Actual time for second slope of ramp, in microseconds.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.2 adi_dm_afe_setup_t Struct Reference

IN parameter to `adi_dm_AfeSetup`

```
#include <adi_dmdriver.h>
```

Data Fields

- uint32_t rx_channels

  If zero, settings are applied to all RX channels.
- adi_dm_hpf_fc_t hpf_fc

  High pass filter (HPF) corner frequency.
- adi_dm_hpf_gain_t hpf_gain

  High pass filter (HPF) gain.
- bool hpf_bypass

  If true, high pass filter (HPF) is bypassed, otherwise it is enabled.
- adi_dm_pga_mux_t **pga_mux**
- adi_dm_pga_gain_t pga_gain

  Selects input to programmable gain amplifier (PGA)
- adi_dm_adc_gain_t adc_gain

  programmable gain amplifier (PGA) gain.

5.2.1 Detailed Description

IN parameter to `adi_dm_AfeSetup`

See also

adi_dm_AfeSetup

5.2.2 Field Documentation

5.2.2.1 adc_gain

`adi_dm_adc_gain_t adi_dm_afe_setup_t::adc_gain`

programmable gain amplifier (PGA) gain.

5.2.2.2 rx_channels

```
uint32_t adi_dm_afe_setup_t::rx_channels
```

If zero, settings are applied to all RX channels.

If non-zero, settings are applied to RX channels for which a bit is set. In a multi-device system the first 4 bits correspond to the 4 Rx channels of ADI_DM_MASTER, the next 4 to the channels for ADI_DM_SLAVE1 etc. So, for instance, if `rx_channels` is 0x21 then, as bits 0 and 5 are set, the settings are applied to ADI_DM_MASTER Rx0 and ADI_DM_SLAVE1 Rx1.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.3 adi_dm_builtin_mimo_setup_t Struct Reference

IN parameter to adi_dm_BuiltinMimoSetup()

```
#include <adi_dmdriver.h>
```

Data Fields

- adi_dm_ramp_config_t ramp_config

  Burst invariant ramp configuration.
- adi_dm_ramp_profile_t ramp_profile

  Ramp parameters.
- unsigned mimo_seq_len:2

  Length of MIMO sequences.
-

  struct {
    unsigned mimo_seq_val:12
      Tx and phase pattern for each device.
  } **dm** [ADI_DM_NUM_DIGIMMIC]

5.3.1 Detailed Description

IN parameter to adi_dm_BuiltinMimoSetup()

Register values for ramp generation using built-in MIMO mode. See "Built-in MIMO Mode" subsection of [2].

See also

adi_dm_BuiltinMimoSetup

5.3.2 Field Documentation

5.3.2.1 mimo_seq_len

```
unsigned adi_dm_builtin_mimo_setup_t::mimo_seq_len
```

Length of MIMO sequences.

See "Built-in MIMO Mode" subsection of [2]

5.3.2.2 mimo_seq_val

```
unsigned adi_dm_builtin_mimo_setup_t::mimo_seq_val
```

Tx and phase pattern for each device.

See "Built-in MIMO Mode" subsection of [2]

5.3.2.3 ramp_config

```
adi_dm_ramp_config_t adi_dm_builtin_mimo_setup_t::ramp_config
```

Burst invariant ramp configuration.

5.3.2.4 ramp_profile

```
adi_dm_ramp_profile_t adi_dm_builtin_mimo_setup_t::ramp_profile
```

Ramp parameters.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.4 adi_dm_burst_profile_t Struct Reference

Burst Profile.

```
#include <adi_dmdriver.h>
```

Data Fields

- uint32_t num_ramps_in_seq

  Number of ramps in a sequence and number of elements in ramp array.
- uint32_t num_seq_in_burst

  Number of sequences in a burst.
- adi_dm_ramp_profile_t ∗ ramp_profile

  Array of ramp profiles.
- uint32_t tx_overlay_len

  Length of tx_overlay array.
- adi_dm_tx_overlay_t ∗ tx_overlay [ADI_DM_NUM_DIGIMMIC]

  Array of bit fields to overlay ramp_gen_tx register.

5.4.1 Detailed Description

Burst Profile.

Data type corresponding to the burst description that gets stored in memory for a DMA generated burst. Note that in order to accomodate adi_dm_PowerDetectorMeasTask, the minimum number of ramps in a burst should be 228.

5.4.2 Field Documentation

5.4.2.1 num_ramps_in_seq

```
uint32_t adi_dm_burst_profile_t::num_ramps_in_seq
```

Number of ramps in a sequence and number of elements in ramp array.

5.4.2.2 num_seq_in_burst

```
uint32_t adi_dm_burst_profile_t::num_seq_in_burst
```

Number of sequences in a burst.

5.4.2.3 ramp_profile

```
adi_dm_ramp_profile_t* adi_dm_burst_profile_t::ramp_profile
```

Array of ramp profiles.

5.4.2.4 tx_overlay

```
adi_dm_tx_overlay_t* adi_dm_burst_profile_t::tx_overlay[ADI_DM_NUM_DIGIMMIC]
```

Array of bit fields to overlay ramp_gen_tx register.

per device

5.4.2.5 tx_overlay_len

```
uint32_t adi_dm_burst_profile_t::tx_overlay_len
```

Length of tx_overlay array.

Zero for no array.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.5 adi_dm_calc_rfpll_period_t Struct Reference

IN parameter to `adi_dm_CalcRfpllPeriod`

```
#include <adi_dmdriver.h>
```

Data Fields

- adi_dm_burst_profile_t ∗ burst_profile

  Burst for which RFPLL period is to be calculated.
- uint64_t ramp_start_freq_hz

  Start frequency of burst.

5.5.1 Detailed Description

IN parameter to `adi_dm_CalcRfpllPeriod`

See also

adi_dm_CalcRfpllPeriod

5.5.2 Field Documentation

5.5.2.1 burst_profile

```
adi_dm_burst_profile_t* adi_dm_calc_rfpll_period_t::burst_profile
```

Burst for which RFPLL period is to be calculated.

5.5.2.2 ramp_start_freq_hz

```
uint64_t adi_dm_calc_rfpll_period_t::ramp_start_freq_hz
```

Start frequency of burst.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.6 adi_dm_dma_ramp_setup_t Struct Reference

IN parameter to adi_dm_DmaRampSetup()

```
#include <adi_dmdriver.h>
```

Data Fields

- adi_dm_ramp_config_t ramp_config

  Burst invariant ramp configuration.
- uint32_t bpid

  Select first profile to use, from adi_dm_BurstProfileSetup().

5.6.1 Detailed Description

IN parameter to adi_dm_DmaRampSetup()

Register writes for ramp generation using DMA, including writing ramp_config registers and initialization of DMA controller. See "DMA Interfacing" subsection of [2].

See also

adi_dm_DmaRampSetup

5.6.2 Field Documentation

5.6.2.1 bpid

```
uint32_t adi_dm_dma_ramp_setup_t::bpid
```

Select first profile to use, from adi_dm_BurstProfileSetup().

5.6.2.2 ramp_config

```
adi_dm_ramp_config_t adi_dm_dma_ramp_setup_t::ramp_config
```

Burst invariant ramp configuration.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.7 adi_dm_lvds_setup_t Struct Reference

IN parameter to `adi_dm_LvdsSetup`

```
#include <adi_dmdriver.h>
```

Data Fields

- bool crc32_en

  Enable CRC32 field.
- bool status_en

  Enable status field.
- unsigned filter_output_bitwidth:2

  Bits per sample.
- bool dual_fs_mode

  Select dual frame sync mode.
- bool fs_active_low

  Frame sync active low.
- bool streaming_mode_1

  Select streaming mode 1.
- bool payload_dis

  Disable transmission of RX channel data.
- bool id_wc_en

  Enable dataid and word count fields.
- bool ls_byte_first

  Send least significant byte first.
- bool ls_bit_first

  Send least significant bit first.
- adi_dm_lvds_clk_t lvds_clk

  Select lvds_clk.
-

struct {
  uint8_t userval0
    Appears in rx0 status field.
  uint8_t userval1
    Appears in rx1 status field.
  uint8_t userval2
    Appears in rx2 status field.
  uint8_t userval3
    Appears in rx3 status field.
  uint8_t dataid0
    Appears in rx0 dataid field.
  uint8_t dataid1
    Appears in rx1 dataid field.
  uint8_t dataid2
    Appears in rx2 dataid field.
  uint8_t dataid3
    Appears in rx3 dataid field.
} **dm** [ADI_DM_NUM_DIGIMMIC]

5.7.1 Detailed Description

IN parameter to `adi_dm_LvdsSetup`

See also

adi_dm_LvdsSetup

5.7.2 Field Documentation

5.7.2.1 crc32_en

`bool adi_dm_lvds_setup_t::crc32_en`

Enable CRC32 field.

See "CRC32" in "LVDS Data Frame Content" in [2].

5.7.2.2 dataid0

`uint8_t adi_dm_lvds_setup_t::dataid0`

Appears in rx0 dataid field.

See "Data ID and Byte Count" in "LVDS Data Frame Content" in [2].

5.7.2.3 dataid1

```
uint8_t adi_dm_lvds_setup_t::dataid1
```

Appears in rx1 dataid field.

See "Data ID and Byte Count" in "LVDS Data Frame Content" in [2].

5.7.2.4 dataid2

```
uint8_t adi_dm_lvds_setup_t::dataid2
```

Appears in rx2 dataid field.

See "Data ID and Byte Count" in "LVDS Data Frame Content" in [2].

5.7.2.5 dataid3

```
uint8_t adi_dm_lvds_setup_t::dataid3
```

Appears in rx3 dataid field.

See "Data ID and Byte Count" in "LVDS Data Frame Content" in [2].

5.7.2.6 dual_fs_mode

```
bool adi_dm_lvds_setup_t::dual_fs_mode
```

Select dual frame sync mode.

See "LVDS Data Framing" in [2].

5.7.2.7 filter_output_bitwidth

```
unsigned adi_dm_lvds_setup_t::filter_output_bitwidth
```

Bits per sample.

Must be one of ADI_DM_OUTPUT_BITWIDTH_16, ADI_DM_OUTPUT_BITWIDTH_14, or ADI_DM_OUTPUT_BITWIDTH_12

5.7.2.8 fs_active_low

```
bool adi_dm_lvds_setup_t::fs_active_low
```

Frame sync active low.

See "LVDS Data Framing" in [2].

5.7.2.9 id_wc_en

`bool adi_dm_lvds_setup_t::id_wc_en`

Enable dataid and word count fields.

See "Data ID and Byte Count" in "LVDS Data Frame Content" in [2].

5.7.2.10 ls_bit_first

`bool adi_dm_lvds_setup_t::ls_bit_first`

Send least significant bit first.

See "LVDS Data Formatting" in [2].

5.7.2.11 ls_byte_first

`bool adi_dm_lvds_setup_t::ls_byte_first`

Send least significant byte first.

See "LVDS Data Formatting" in [2].

5.7.2.12 lvds_clk

`adi_dm_lvds_clk_t adi_dm_lvds_setup_t::lvds_clk`

Select lvds_clk.

See "LVDS Clock selection ..." in [2].

5.7.2.13 payload_dis

`bool adi_dm_lvds_setup_t::payload_dis`

Disable transmission of RX channel data.

See "Rx Channel Data" in "LVDS Data Frame Content" in [2].

5.7.2.14 status_en

`bool adi_dm_lvds_setup_t::status_en`

Enable status field.

See "Status Data" in "LVDS Data Frame Content" in [2].

5.7.2.15 streaming_mode_1

`bool adi_dm_lvds_setup_t::streaming_mode_1`

Select streaming mode 1.

See "LVDS Streaming Modes" in [2].

5.7.2.16 userval0

`uint8_t adi_dm_lvds_setup_t::userval0`

Appears in rx0 status field.

See "Status Data" in "LVDS Data Frame Content" in [2].

5.7.2.17 userval1

`uint8_t adi_dm_lvds_setup_t::userval1`

Appears in rx1 status field.

See "Status Data" in "LVDS Data Frame Content" in [2].

5.7.2.18 userval2

`uint8_t adi_dm_lvds_setup_t::userval2`

Appears in rx2 status field.

See "Status Data" in "LVDS Data Frame Content" in [2].

5.7.2.19 userval3

`uint8_t adi_dm_lvds_setup_t::userval3`

Appears in rx3 status field.

See "Status Data" in "LVDS Data Frame Content" in [2].

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.8 adi_dm_mask_faults_t Struct Reference

IN parameter to `adi_dm_MaskFaults`

`#include <adi_dmdriver.h>`

Data Fields

- uint32_t **fault_status0_mask**
- uint32_t fault_status1_mask

  Bits set correspond to faults to ignore in REG_FAULTCTL_FAULT_STATUS0.
- uint32_t fault_status2_mask

  Bits set correspond to faults to ignore in REG_FAULTCTL_FAULT_STATUS1.
- uint32_t sw_fault0_mask

  Bits set correspond to faults to ignore in REG_FAULTCTL_FAULT_STATUS2.
- uint32_t sw_fault1_mask

  Bits set correspond to faults to ignore in REG_FAULTCTL_SOFTWARE_FAULT_0.
- uint32_t sw_fault2_mask

  Bits set correspond to faults to ignore in REG_FAULTCTL_SOFTWARE_FAULT_1.
- uint32_t sw_fault3_mask

  Bits set correspond to faults to ignore in REG_FAULTCTL_SOFTWARE_FAULT_2.

5.8.1 Detailed Description

IN parameter to `adi_dm_MaskFaults`

See also

adi_dm_MaskFaults

5.8.2 Field Documentation

5.8.2.1 fault_status1_mask

```
uint32_t adi_dm_mask_faults_t::fault_status1_mask
```

Bits set correspond to faults to ignore in REG_FAULTCTL_FAULT_STATUS0.

5.8.2.2 fault_status2_mask

```
uint32_t adi_dm_mask_faults_t::fault_status2_mask
```

Bits set correspond to faults to ignore in REG_FAULTCTL_FAULT_STATUS1.

5.8.2.3 sw_fault0_mask

```
uint32_t adi_dm_mask_faults_t::sw_fault0_mask
```

Bits set correspond to faults to ignore in REG_FAULTCTL_FAULT_STATUS2.

5.8.2.4 sw_fault1_mask

```
uint32_t adi_dm_mask_faults_t::sw_fault1_mask
```

Bits set correspond to faults to ignore in REG_FAULTCTL_SOFTWARE_FAULT_0.

5.8.2.5 sw_fault2_mask

```
uint32_t adi_dm_mask_faults_t::sw_fault2_mask
```

Bits set correspond to faults to ignore in REG_FAULTCTL_SOFTWARE_FAULT_1.

5.8.2.6 sw_fault3_mask

```
uint32_t adi_dm_mask_faults_t::sw_fault3_mask
```

Bits set correspond to faults to ignore in REG_FAULTCTL_SOFTWARE_FAULT_2.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.9 adi_dm_mipi_setup_t Struct Reference

IN parameter to `adi_dm_MipiSetup`

```
#include <adi_dmdriver.h>
```

Data Fields

- bool crc32_en

  Enable CRC32 field.
- bool status_en

  Enable status field.
- unsigned filter_output_bitwidth:2

  Bits per sample.
- uint32_t ref_freq_hz

  Reference frequency, in hertz.
- adi_dm_mipi_clk_t mipi_clk

  MIPI CSI-2 clock.
- bool continuous_clock

  Continuous clock mode.
- adi_dm_num_mipi_lanes_t **num_lanes**
- adi_dm_mipi_data_type_t **mipi_data_type**
- bool ls_byte_first

  Send least significant byte first.
- bool byte_interleaving

  See "Data Interleaving & Byte Order (endianness)" in [2].
-

  struct {
    uint8_t userval0
      Appears in rx0 status field.
    uint8_t userval1
      Appears in rx1 status field.
    uint8_t userval2
      Appears in rx2 status field.
    uint8_t userval3
      Appears in rx3 status field.
  } **dm** [ADI_DM_NUM_DIGIMMIC]

5.9.1 Detailed Description

IN parameter to `adi_dm_MipiSetup`

See also

adi_dm_MipiSetup

5.9.2 Field Documentation

5.9.2.1 byte_interleaving

```
bool adi_dm_mipi_setup_t::byte_interleaving
```

See "Data Interleaving & Byte Order (endianness)" in [2].

5.9.2.2 crc32_en

`bool adi_dm_mipi_setup_t::crc32_en`

Enable CRC32 field.

See "CRC32" in "Data Frame Content" in [2].

5.9.2.3 filter_output_bitwidth

`unsigned adi_dm_mipi_setup_t::filter_output_bitwidth`

Bits per sample.

Must be one of ADI_DM_OUTPUT_BITWIDTH_16, ADI_DM_OUTPUT_BITWIDTH_14, or ADI_DM_OUTPUT_BITWIDTH_12

5.9.2.4 ls_byte_first

`bool adi_dm_mipi_setup_t::ls_byte_first`

Send least significant byte first.

See "Data Interleaving & Byte Order (endianness)" in [2].

5.9.2.5 mipi_clk

`adi_dm_mipi_clk_t adi_dm_mipi_setup_t::mipi_clk`

MIPI CSI-2 clock.

5.9.2.6 ref_freq_hz

`uint32_t adi_dm_mipi_setup_t::ref_freq_hz`

Reference frequency, in hertz.

The same value passed to adi_dm_PowerUp().

5.9.2.7 status_en

`bool adi_dm_mipi_setup_t::status_en`

Enable status field.

See "Status word" in "Data Frame Content" in [2].

5.9.2.8 userval0

```
uint8_t adi_dm_mipi_setup_t::userval0
```

Appears in rx0 status field.

See "Status Data" in "MIPI Data Frame Content" in [2].

5.9.2.9 userval1

```
uint8_t adi_dm_mipi_setup_t::userval1
```

Appears in rx1 status field.

See "Status Data" in "MIPI Data Frame Content" in [2].

5.9.2.10 userval2

```
uint8_t adi_dm_mipi_setup_t::userval2
```

Appears in rx2 status field.

See "Status Data" in "MIPI Data Frame Content" in [2].

5.9.2.11 userval3

```
uint8_t adi_dm_mipi_setup_t::userval3
```

Appears in rx3 status field.

See "Status Data" in "MIPI Data Frame Content" in [2].

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.10 adi_dm_periodic_calibration_t Struct Reference

IN parameter to `adi_dm_PeriodicCalibration`

```
#include <adi_dmdriver.h>
```

Data Fields

- bool no_rfpll_bow_cal

  Do not run ADI_ADAR690x_FW_RFPLL_BOW_CAL.
- bool no_lochain_cal

  Do not run ADI_ADAR690x_FW_LOCHAIN_CAL.
- bool no_adcpll_align

  Do not run ADI_ADAR690x_FW_ADCPLL_ALIGN.
- bool no_rxgain_cal

  Do not run ADI_ADAR690x_FW_RXGAIN_CAL.
- bool no_adc_phase_cal

  Do not run ADI_ADAR690x_FW_ADC_PHASE_CAL.
- bool no_pa_adj_cal

  Do not run ADI_ADAR690x_FW_TXPA_ADJ.
- bool no_hpf_cal

  Do not run ADI_ADAR690x_FW_HPF_CAL.
- bool run_checks

  Run built-in-self-test (BIST) tasks to check for proper functioning of device.
- bool run_power_checks

  Run BISTs that require power detection during burst.
- bool run_rfpll_period_chk

  Run BISTs that require adi_dm_WriteRfpllPeriod() to have been called.

5.10.1 Detailed Description

IN parameter to `adi_dm_PeriodicCalibration`

See also

adi_dm_PeriodicCalibration

5.10.2 Field Documentation

5.10.2.1 run_checks

```
bool adi_dm_periodic_calibration_t::run_checks
```

Run built-in-self-test (BIST) tasks to check for proper functioning of device.

See adi_dm_LockConfig()

5.10.2.2 run_power_checks

```
bool adi_dm_periodic_calibration_t::run_power_checks
```

Run BISTs that require power detection during burst.

Needs `run_checks` to be set also. See adi_dm_PowerDetectorMeasTask() and adi_dm_WriteSysCalTx()

5.10.2.3 run_rfpll_period_chk

```
bool adi_dm_periodic_calibration_t::run_rfpll_period_chk
```

Run BISTs that require adi_dm_WriteRfpllPeriod() to have been called.

Needs `run_checks` to be set also.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.11 adi_dm_power_detector_meas_task_t Struct Reference

IN parameter to `adi_dm_PowerDetectorMeasTask`

`#include <adi_dmdriver.h>`

Data Fields

-

  struct {
    bool **tx_active** [ADI_DM_NUM_TX]
    unsigned tx0_inactive_when_all_inactive:1
      Tx channels active during bursts for Power detector measurements.
    unsigned **tx0_inactive_when_tx1_active**:1
    unsigned **tx0_inactive_when_tx2_active**:1
    unsigned **tx0_inactive_when_tx1_tx2_active**:1
    unsigned **tx1_inactive_when_all_inactive**:1
    unsigned **tx1_inactive_when_tx0_active**:1
    unsigned **tx1_inactive_when_tx2_active**:1
    unsigned **tx1_inactive_when_tx0_tx2_active**:1
    unsigned **tx2_inactive_when_all_inactive**:1
    unsigned **tx2_inactive_when_tx0_active**:1
    unsigned **tx2_inactive_when_tx1_active**:1
    unsigned **tx2_inactive_when_tx0_tx1_active**:1
  } **dm** [ADI_DM_NUM_DIGIMMIC]

5.11.1 Detailed Description

IN parameter to `adi_dm_PowerDetectorMeasTask`

See also

adi_dm_PowerDetectorMeasTask

5.11.2 Field Documentation

5.11.2.1 tx0_inactive_when_all_inactive

`unsigned adi_dm_power_detector_meas_task_t::tx0_inactive_when_all_inactive`

Tx channels active during bursts for Power detector measurements.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.12 adi_dm_power_up_t Struct Reference

IN parameter to `adi_dm_PowerUp`

`#include <adi_dmdriver.h>`

Data Fields

- bool power_part_only

  Power up the part but do not initialize it.
- bool power_part_load_files_only

  Power up the part and load the firmware but do not initialize it.
- bool is_standalone_master

  Execute init sequence for cascaded master.
- bool is_standalone_slave

  Execute init sequence for cascaded slave.
- bool is_lo_right [ADI_DM_NUM_DIGIMMIC]

  Setting of lo_left_right for each device.
- adi_dm_timing_comp_setting_t timing_comp_setting [ADI_DM_NUM_DIGIMMIC]

  ADI_ADAR690x_CFG_TIMING_COMP_EN setting.
- const uint32_t ∗ firmware_image

  Firmware code image in host memory.
- uint32_t firmware_sz

  Size of firmware code image.
- const uint32_t ∗ firmware_constants_image

  Firmware constants image in host memory.
- uint32_t firmware_constants_sz

  Size of firmware constants.
- uint32_t ref_freq_hz

  Reference clock.
- uint32_t rfpll_loop_bw_hz

  RF PLL Loop Bandwidth.
- uint64_t ramp_start_freq_hz

  Tx output frequency at start of burst.
- float ramp_bw_Mhz

  Maximum ramp bandwidth.
- bool enable_clkhost

  Enable the CLKHOST pin.
- adi_dm_clkoutctrl_t clkoutctrl

  Signal on CLKHOST pin if enabled.

5.12.1 Detailed Description

IN parameter to `adi_dm_PowerUp`

See also

adi_dm_PowerUp

5.12.2 Field Documentation

5.12.2.1 clkoutctrl

`adi_dm_clkoutctrl_t adi_dm_power_up_t::clkoutctrl`

Signal on CLKHOST pin if enabled.

See "Reference input Section" in [2].

5.12.2.2 enable_clkhost

`bool adi_dm_power_up_t::enable_clkhost`

Enable the CLKHOST pin.

5.12.2.3 firmware_constants_image

`const uint32_t* adi_dm_power_up_t::firmware_constants_image`

Firmware constants image in host memory.

5.12.2.4 firmware_constants_sz

`uint32_t adi_dm_power_up_t::firmware_constants_sz`

Size of firmware constants.

5.12.2.5 firmware_image

```
const uint32_t* adi_dm_power_up_t::firmware_image
```

Firmware code image in host memory.

5.12.2.6 firmware_sz

```
uint32_t adi_dm_power_up_t::firmware_sz
```

Size of firmware code image.

5.12.2.7 is_lo_right

```
bool adi_dm_power_up_t::is_lo_right[ADI_DM_NUM_DIGIMMIC]
```

Setting of lo_left_right for each device.

5.12.2.8 is_standalone_master

```
bool adi_dm_power_up_t::is_standalone_master
```

Execute init sequence for cascaded master.

Only applies if driver built for a single DigiMMIC.

5.12.2.9 is_standalone_slave

```
bool adi_dm_power_up_t::is_standalone_slave
```

Execute init sequence for cascaded slave.

Only applies if driver built for a single DigiMMIC.

5.12.2.10 power_part_only

```
bool adi_dm_power_up_t::power_part_only
```

Power up the part but do not initialize it.

Exits after RTWO power up.

5.12.2.11 timing_comp_setting

`adi_dm_timing_comp_setting_t adi_dm_power_up_t::timing_comp_setting[ADI_DM_NUM_DIGIMMIC]`

ADI_ADAR690x_CFG_TIMING_COMP_EN setting.

This BIST ensures ramp timings on cascaded devices are consitent by comparing local ramp time with STAT1 monitor input which is generated by another device's STAT0 output. The recommended wiring connects leaves one device's input unconnected so the comparator should be disabled on that device.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.13 adi_dm_ramp_config_t Struct Reference

Ramp Configuration.

`#include <adi_dmdriver.h>`

Data Fields

- unsigned ramp_count:12

  Number of ramps in burst.
- unsigned pga_shunt_en:1

  Enable PGA Shunt signal to power off PGA during slope1.
- uint8_t pga_shunt_start_del

  Time in ref clocks before slope1 to power off PGA.
- uint8_t pga_shunt_stop_del

  Time in ref clocks after slope1 to power on PGA.
- unsigned filter_decim_ratio:9

  Decimation Ratio.
- unsigned filter_valid_delay:9

  Number of samples to suppress at start of ramp.
-

  struct {
    unsigned vga_sync_data_valid:1
      Synchronize VGA gain activation with data valid signal.
    unsigned vga_on_patt:6
      Bitset with parts of ramp where VGAs are enabled.
    unsigned pa_off_time:5
      Time between PAs Activated and Deactivated, in REF_CLK cycles.
    unsigned vga_gauss_dis:1
      Disable VGA Gain Gaussian Shape.
    unsigned vga_gain_steps:3
      Number of steps in VGA gain shape is 2^ vga_gain_steps.
    unsigned vga_gain_step_div:6
      Time of each step in VGA gain shape, in REF_CLK cycles.
    unsigned phase_mod_en:1
      Enable phase modulation during ramp generation.
    unsigned phase_delay_en:1
      Start phase modulation with data_valid.
  } **dm** [ADI_DM_NUM_DIGIMMIC]

5.13.1 Detailed Description

Ramp Configuration.

Data type with the common parameters to adi_dm_BuiltinMimoSetup() and adi_dm_DmaRampSetup() which are mainly written to fields of RFPLL_RAMP_CONFIG registers.

5.13.2 Field Documentation

5.13.2.1 filter_decim_ratio

```
unsigned adi_dm_ramp_config_t::filter_decim_ratio
```

Decimation Ratio.

5.13.2.2 filter_valid_delay

```
unsigned adi_dm_ramp_config_t::filter_valid_delay
```

Number of samples to suppress at start of ramp.

5.13.2.3 ramp_count

```
unsigned adi_dm_ramp_config_t::ramp_count
```

Number of ramps in burst.

5.13.2.4 vga_gain_steps

```
unsigned adi_dm_ramp_config_t::vga_gain_steps
```

Number of steps in VGA gain shape is 2^vga_gain_steps.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.14 adi_dm_ramp_profile_t Struct Reference

Ramp Profile.

```
#include <adi_dmdriver.h>
```

Data Fields

- unsigned del_0:22

  Delay before starting the first slope of the ramp.
- unsigned del_1:20

  Delay between the two slopes of the ramp.
- unsigned del_2:22

  Delay before starting the next ramp.
- uint32_t ramp_steps_0

  Number of steps for the first slope of the ramp.
- uint32_t ramp_steps_1

  Number of steps for the second slope of the ramp.
- int32_t ramp_dev_0

  Change in frequency at each step of the first slope.
- int32_t ramp_dev_1

  Change in frequency at each step of the second slope.
- uint16_t afe_start_off_time

  Time for which the ADC is off after the start of the ramp, in AFE_CLK cycles.
- uint16_t afe_ramp_time

  Total ramp time, in AFE_CLK cycles.
- unsigned num_samples:13

  Number of ADC samples to take per ramp.
-

  struct {

  unsigned tx_pattern:3

  Transmitters to enable for the ramp.

  unsigned ramp_stat_bit_0:1

  See "Ramp Status Pins" subsection of [2].

  unsigned ramp_stat_bit_1:1

  See "Ramp Status Pins" subsection of [2].

  unsigned ramp_stat_bit_2:1

  See "Ramp Status Pins" subsection of [2].

  unsigned pa0_phase:7

  Initial phase index for Tx0.

  unsigned pa1_phase:7

  Initial phase index for Tx1.

  unsigned pa2_phase:7

  Initial phase index for Tx2.

  unsigned phase_step:5

  If non-zero, delay before incrementing phase, in AFE_CLK cycles.

  unsigned pa0_phase_dev:7

  Increment to phase index for Tx0 each phase step.

  unsigned pa1_phase_dev:7

  Increment to phase index for Tx1 each phase step.

  unsigned pa2_phase_dev:7

  Increment to phase index for Tx2 each phase step.

  } **dm** [ADI_DM_NUM_DIGIMMIC]

5.14.1 Detailed Description

Ramp Profile.

Data type equivalent to contents of the RAMPGEN registers across a cascade of DigiMMICs. Fields describing the shape of the ramp are common across cascaded devices. Other fields are duplicated for each device.

5.14.2 Field Documentation

5.14.2.1 pa0_phase

```
unsigned adi_dm_ramp_profile_t::pa0_phase
```

Initial phase index for Tx0.

5.14.2.2 pa0_phase_dev

```
unsigned adi_dm_ramp_profile_t::pa0_phase_dev
```

Increment to phase index for Tx0 each phase step.

5.14.2.3 pa1_phase

```
unsigned adi_dm_ramp_profile_t::pa1_phase
```

Initial phase index for Tx1.

5.14.2.4 pa1_phase_dev

```
unsigned adi_dm_ramp_profile_t::pa1_phase_dev
```

Increment to phase index for Tx1 each phase step.

5.14.2.5 pa2_phase

```
unsigned adi_dm_ramp_profile_t::pa2_phase
```

Initial phase index for Tx2.

5.14.2.6 pa2_phase_dev

```
unsigned adi_dm_ramp_profile_t::pa2_phase_dev
```

Increment to phase index for Tx2 each phase step.

5.14.2.7 phase_step

```
unsigned adi_dm_ramp_profile_t::phase_step
```

If non-zero, delay before incrementing phase, in AFE_CLK cycles.

5.14.2.8 ramp_stat_bit_0

```
unsigned adi_dm_ramp_profile_t::ramp_stat_bit_0
```

See "Ramp Status Pins" subsection of [2].

5.14.2.9 ramp_stat_bit_1

```
unsigned adi_dm_ramp_profile_t::ramp_stat_bit_1
```

See "Ramp Status Pins" subsection of [2].

5.14.2.10 ramp_stat_bit_2

```
unsigned adi_dm_ramp_profile_t::ramp_stat_bit_2
```

See "Ramp Status Pins" subsection of [2].

5.14.2.11 tx_pattern

```
unsigned adi_dm_ramp_profile_t::tx_pattern
```

Transmitters to enable for the ramp.

A bitset.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.15 adi_dm_ramp_shape_t Struct Reference

High level description of ramp shape input to adi_dm_CalcRamp()

`#include <adi_dmdriver.h>`

Data Fields

- float slope0_time_us

  Time for first slope of ramp, in microseconds.
- float slope1_time_us

  Time for second slope of ramp, in microseconds.
- float delay0_time_us

  Time for delay before `slope0`, in microseconds.
- float delay1_time_us

  Time for delay between `slope0` and `slope1`, in microseconds.
- float delay2_time_us

  Time for delay after `slope1`, in microseconds.
- float ramp_bw_Mhz

  Difference between frequency of `delay1` and `delay0`, in megahertz.
- uint32_t num_samples

  Number of samples.
- float sample_start_time_us

  Time to start sampling, measured from start of `delay0` in microseconds.
- bool pga_shunt_en

  Enable PGA Shunt.
- float pga_shunt_start_del_us

  Start delay for PGA Shunt, in microseconds.
- float pga_shunt_stop_del_us

  Stop delay for PGA Shunt, in microseconds.
- uint32_t filter_valid_delay

  Decimation filter group delay.
- uint32_t decim_ratio

  Decimation ratio.
- uint32_t ref_freq_hz

  Reference frequency, in hertz.
- unsigned ramp_count:12

  Copied through to adi_dm_CalcRamp() ramp_config output parameter.

5.15.1 Detailed Description

High level description of ramp shape input to adi_dm_CalcRamp()

See also

adi_dm_CalcRamp

5.15.2 Field Documentation

5.15.2.1 decim_ratio

`uint32_t adi_dm_ramp_shape_t::decim_ratio`

Decimation ratio.

See [2] "Decimation filter" subsection. Must be one of 24, 32, 48, 64, 96, 128, 192 or 256.

5.15.2.2 delay0_time_us

`float adi_dm_ramp_shape_t::delay0_time_us`

Time for delay before `slope0`, in microseconds.

5.15.2.3 delay1_time_us

`float adi_dm_ramp_shape_t::delay1_time_us`

Time for delay between `slope0` and `slope1`, in microseconds.

5.15.2.4 delay2_time_us

`float adi_dm_ramp_shape_t::delay2_time_us`

Time for delay after `slope1`, in microseconds.

5.15.2.5 filter_valid_delay

`uint32_t adi_dm_ramp_shape_t::filter_valid_delay`

Decimation filter group delay.

See [2] "Frequency Ramp" subsection. If this is zero, a default value is returned. Note this value must be the same for every ramp in a burst.

5.15.2.6 num_samples

```
uint32_t adi_dm_ramp_shape_t::num_samples
```

Number of samples.

5.15.2.7 pga_shunt_en

```
bool adi_dm_ramp_shape_t::pga_shunt_en
```

Enable PGA Shunt.

See [2] "Frequency Ramp" subsection. Note this value must be the same for every ramp in a burst.

5.15.2.8 pga_shunt_start_del_us

```
float adi_dm_ramp_shape_t::pga_shunt_start_del_us
```

Start delay for PGA Shunt, in microseconds.

If `pga_shunt_en` is true and this is zero a default value is returned. Note this value must be the same for every ramp in a burst.

5.15.2.9 pga_shunt_stop_del_us

```
float adi_dm_ramp_shape_t::pga_shunt_stop_del_us
```

Stop delay for PGA Shunt, in microseconds.

If `pga_shunt_en` is true and this is zero a default value is returned. Note this value must be the same for every ramp in a burst.

5.15.2.10 ramp_bw_Mhz

```
float adi_dm_ramp_shape_t::ramp_bw_Mhz
```

Difference between frequency of `delay1` and `delay0`, in megahertz.

5.15.2.11 ramp_count

```
unsigned adi_dm_ramp_shape_t::ramp_count
```

Copied through to adi_dm_CalcRamp() ramp_config output parameter.

5.15.2.12 ref_freq_hz

```
uint32_t adi_dm_ramp_shape_t::ref_freq_hz
```

Reference frequency, in hertz.

Must be a value in the range 40 MHz to 80 MHz

5.15.2.13 sample_start_time_us

```
float adi_dm_ramp_shape_t::sample_start_time_us
```

Time to start sampling, measured from start of `delay0` in microseconds.

This time includes the decimation filter group delay. See `filter_valid_delay`. When `sample_start_↩ time_us` is zero `afe_init_del` is calculated by placing the sample window as far right in `slope0` as possible.

5.15.2.14 slope0_time_us

```
float adi_dm_ramp_shape_t::slope0_time_us
```

Time for first slope of ramp, in microseconds.

5.15.2.15 slope1_time_us

```
float adi_dm_ramp_shape_t::slope1_time_us
```

Time for second slope of ramp, in microseconds.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.16 adi_dm_rfpll_reconfig_t Struct Reference

IN parameter to adi_dm_RfpllReconfig()

```
#include <adi_dmdriver.h>
```

Data Fields

- uint64_t ramp_start_freq_hz
  Tx output frequency at start of burst.
- float ramp_bw_Mhz
  Maximum ramp bandwidth.

5.16.1 Detailed Description

IN parameter to adi_dm_RfpllReconfig()

See also

adi_dm_RfpllReconfig

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.17 adi_dm_tasklist_pool_setup_t Struct Reference

IN parameter to `adi_dm_TasklistPoolSetup`

`#include <adi_dmdriver.h>`

Data Fields

- uint32_t num_tasklists

  Number of tasklists to setup.
- uint32_t ** tasklists [ADI_DM_NUM_DIGIMMIC]

  Array of arrays containing tasklist in memory format.
- uint32_t * tasklist_length [ADI_DM_NUM_DIGIMMIC]

  Array of arrays giving the number of words in each tasklist.

5.17.1 Detailed Description

IN parameter to `adi_dm_TasklistPoolSetup`

The *tasklists* array contains *num_tasklists* tasklists for each device. The *tasklist_length* array gives the length of each tasklist on each device.

The underlying representation of a tasklist is the machine representation:

- A tasklist consist of a tasklist control word followed by a number of tasks.
- Tasklist control word is a uint32_t encoded:
  - bit 31: WDTOUT control
    - 0 does not pulse at beginning of tasklist.
    - 1 pulses at beginning of tasklist.
  - bits 30:16 reserved.
  - bits 15:0 number of tasks.
- A task is a task control word followed by a variable number of parameters and the watchdog timer window.
  - Task control word is a uint32_t encoded:

    * bit 31: WDTOUT configuration. ()
        - 0: no WDTOUT.
        - 1: pulse WDTOUT.
        - This bit along with 16bit task ID acts as future key for the task. Watchdog key = (task control word <<1)|(task control word >>31) If LSB of the key is set, WDTOUT is pulsed out on servicing the watchdog.
    * bits 30:20 reserved.
    * bits 19:16 number of dynamic parameter words.
    * bits 15:0 Task ID. The 16 cmd id defined in ::adar690x_fw.h.
- params: as many uint32_t as specified in the task control word.
    * If no dynamic parameters are specified the parameters are read from the configuration section, othewise the right number of parameters for the task must be provided. Currently this must be deduced from examples of firmware calls with parameter in the driver.
- uint32_t: Min cycles for watchdog timer
- uint32_t: Max cycles for watchdog timer

This is somewhat low level. See examples/tasklist_util.c for a more abstract interface to building this data structure.

See also

adi_dm_TasklistPoolSetup

5.17.2 Field Documentation

5.17.2.1 tasklist_length

`uint32_t* adi_dm_tasklist_pool_setup_t::tasklist_length[ADI_DM_NUM_DIGIMMIC]`

Array of arrays giving the number of words in each tasklist.

5.17.2.2 tasklists

`uint32_t** adi_dm_tasklist_pool_setup_t::tasklists[ADI_DM_NUM_DIGIMMIC]`

Array of arrays containing tasklist in memory format.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.18 adi_dm_temperature_t Struct Reference

IN parameter to `adi_dm_TemperatureGet`

`#include <adi_dmdriver.h>`

Data Fields

-
  ```
  struct {
     float tx_temp_c [ADI_DM_NUM_TX]
     float rx_temp_c [ADI_DM_NUM_RX]
        Temperature at each Tx in Celsius.
     float lochain_temp_c
        Temperature at each Rx in Celsius.
  } dm [ADI_DM_NUM_DIGIMMIC]
  ```

5.18.1 Detailed Description

IN parameter to `adi_dm_TemperatureGet`

See also

adi_dm_TemperatureGet

5.18.2 Field Documentation

5.18.2.1 lochain_temp_c

`float adi_dm_temperature_t::lochain_temp_c`

Temperature at each Rx in Celsius.

5.18.2.2 rx_temp_c

`float adi_dm_temperature_t::rx_temp_c[ADI_DM_NUM_RX]`

Temperature at each Tx in Celsius.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.19 adi_dm_tx_overlay_t Struct Reference

Tx Overlay.

`#include <adi_dmdriver.h>`

Data Fields

- unsigned pa0_phase:7

  Initial phase index for Tx0.
- unsigned pa1_phase:7

  Initial phase index for Tx1.
- unsigned pa2_phase:7

  Initial phase index for Tx2.
- unsigned tx_pattern:3

  Transmitters to enable for the ramp.
- unsigned phase_step:5

  If non-zero, delay before incrementing phase, in AFE_CLK cycles.
- unsigned ramp_stat_bit_0:1

  See "Ramp Status Pins" subsection of [2].
- unsigned ramp_stat_bit_1:1

  See "Ramp Status Pins" subsection of [2].
- unsigned ramp_stat_bit_2:1

  See "Ramp Status Pins" subsection of [2].

5.19.1 Detailed Description

Tx Overlay.

Data type equivalent to the contents of the RAMP_GEN_TX register used for the tx_overlay field of adi_dm_burst_profile_t. A denser encoding of a burst profile which only differs in these elements can be achieved by specifying many tx_overlays rather than many ramp_profiles

5.19.2 Field Documentation

5.19.2.1 pa0_phase

```
unsigned adi_dm_tx_overlay_t::pa0_phase
```

Initial phase index for Tx0.

5.19.2.2 pa1_phase

```
unsigned adi_dm_tx_overlay_t::pa1_phase
```

Initial phase index for Tx1.

5.19.2.3 pa2_phase

```
unsigned adi_dm_tx_overlay_t::pa2_phase
```

Initial phase index for Tx2.

5.19.2.4 phase_step

```
unsigned adi_dm_tx_overlay_t::phase_step
```

If non-zero, delay before incrementing phase, in AFE_CLK cycles.

5.19.2.5 ramp_stat_bit_0

```
unsigned adi_dm_tx_overlay_t::ramp_stat_bit_0
```

See "Ramp Status Pins" subsection of [2].

5.19.2.6 ramp_stat_bit_1

```
unsigned adi_dm_tx_overlay_t::ramp_stat_bit_1
```

See "Ramp Status Pins" subsection of [2].

5.19.2.7 ramp_stat_bit_2

```
unsigned adi_dm_tx_overlay_t::ramp_stat_bit_2
```

See "Ramp Status Pins" subsection of [2].

5.19.2.8 tx_pattern

```
unsigned adi_dm_tx_overlay_t::tx_pattern
```

Transmitters to enable for the ramp.

A bitset.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.20 adi_dm_tx_setup_t Struct Reference

IN parameter to `adi_dm_TxSetup`

```
#include <adi_dmdriver.h>
```

Data Fields

-
  struct {
    unsigned tx_enable:3
      Bitset.
    unsigned continuous_pa:3
      Continuous Tx.
    int32_t pa_gain_backoff_db [ADI_DM_NUM_TX]
      Tx gain setting specified as negative number of dB to add to max power.
  } **dm** [ADI_DM_NUM_DIGIMMIC]

5.20.1 Detailed Description

IN parameter to `adi_dm_TxSetup`

Alters firmware parameters for Tc=x channels.

See also

adi_dm_TxSetup

5.20.2 Field Documentation

5.20.2.1 continuous_pa

```
unsigned adi_dm_tx_setup_t::continuous_pa
```

Continuous Tx.

Bitset. Put Tx channel for which corresponding bit is set into continuous mode.

5.20.2.2 pa_gain_backoff_db

```
int32_t adi_dm_tx_setup_t::pa_gain_backoff_db[ADI_DM_NUM_TX]
```

Tx gain setting specified as negative number of dB to add to max power.

A value between 0 and -10.

5.20.2.3 tx_enable

`unsigned adi_dm_tx_setup_t::tx_enable`

Bitset.

Enable Tx channel for which corresponding bit is set.

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.21 adi_dm_write_rfpll_period_t Struct Reference

IN parameter to `adi_dm_WriteRfpllPeriod` *OUT* parameter to `adi_dm_CalcRfpllPeriod`

`#include <adi_dmdriver.h>`

Data Fields

- uint32_t rfpll_period_low_limit

  Low limit for BIST103c: RFPLL period check.
- uint32_t rfpll_period_high_limit

  High limit for BIST103c: RFPLL period check.

5.21.1 Detailed Description

IN parameter to `adi_dm_WriteRfpllPeriod` *OUT* parameter to `adi_dm_CalcRfpllPeriod`

See also

adi_dm_WriteRfpllPeriod, adi_dm_CalcRfpllPeriod

The documentation for this struct was generated from the following file:

- adi_dmdriver.h

5.22 adi_pmic_freq_config_t Struct Reference

Freq Spread Spectrum config register.

`#include <adi_pmic_driver.h>`

Data Fields

- uint32_t enable

  Enable the Frequency Spread Spectrum 0:Disable; 1:Enabled.
- adi_pmic_sweep_depth_t sweep_depth

  Sweep Depth.
- adi_pmic_sweep_freq_t sweep_freq

  Sweep Freq.
- uint32_t sync_en

  Enable Sync Function 0:Disabled; 1:Enabled and pin direction determined by sync_dir.
- uint32_t sync_div

  Setting frequency on SYNC Pin when it configured as output 0:fsw; 1:fsw/5.
- uint32_t sync_dir

  Setting SYNC Pin Direction 0: Input; 1: Output.

5.22.1 Detailed Description

Freq Spread Spectrum config register.

The documentation for this struct was generated from the following file:

- adi_pmic_driver.h

5.23 adi_pmic_qa_ctrl_t Struct Reference

QA Watchdog timer control register.

```
#include <adi_pmic_driver.h>
```

Data Fields

- uint32_t fast_window

  Set the fast window of the QA watchdog.
- uint32_t slow_window

  Set the slow window of the QA watchdog.
- uint32_t fault_threshold

  If the fault counter $>=$ fault threshold, a fault event happens.
- uint32_t pre_scale

  Set the scale factor which is used to caluclate the fast and slow windows.
- uint32_t enable

  Enable or disable the QA watchdog.

5.23.1 Detailed Description

QA Watchdog timer control register.

The documentation for this struct was generated from the following file:

- adi_pmic_driver.h

5.24 adi_pmic_qa_status_t Struct Reference

QA Watchdog timer status register.

```
#include <adi_pmic_driver.h>
```

Data Fields

- uint32_t fault_counter

  The current number of QA watchdog fault counter.
- uint32_t bad_answer

  A flag to indicate a bad answer to the QA watchdog.
- uint32_t single_fail

  A flag to indicate a bad watchdog feed to QA watchdog.

5.24.1 Detailed Description

QA Watchdog timer status register.

The documentation for this struct was generated from the following file:

- adi_pmic_driver.h

5.25 adi_pmic_warn_fault_settings_t Struct Reference

Warn/Fault Window setup.

```
#include <adi_pmic_driver.h>
```

Data Fields

- adi_pmic_warn_fault_window_t **warnWindow**
- adi_pmic_threshold_values_t thresholdLevels

  <Modify the warn window or the fault window 0: faultwindow; 1: warnWindow
- adi_pmic_blank_times_t blankTime

  <The threshold to set the window to

5.25.1 Detailed Description

Warn/Fault Window setup.

The documentation for this struct was generated from the following file:

- adi_pmic_driver.h

Chapter 6

File Documentation

6.1 adar690x_fw.h File Reference

Public C interface to the firmware.

Macros

- #define ADI_ADAR690x_CMD_CONFIG_COMPLETE 0xF001

  Install/uninstall scheduler configuration file.
- #define ADI_ADAR690x_CMD_TASKLIST 0xF003

  Execute a tasklist from scheduler file.
- #define ADI_ADAR690x_FW_BOOTPARSE 0x101

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_ADCPLL_INIT 0x201

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_ADCPLL_MUX_OUT 0x203

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_RFPLL_INIT 0x301

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_RFPLL_MUX_OUT 0x303

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_RFPLL_LOCK 0x304

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_RAMP_TRIG 0x30A

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_CHIP_INIT 0xF01

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_TEMP_MEASURE 0x1001

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_CAL_READ 0x1101

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_ADCPLL_ALIGN 0x1501

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_POWER_MANAGE 0x1701

  See Table 1 in [1].

- #define ADI_ADAR690x_FW_RMW 0x1901

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_MEAS_PWR_DET 0x1E01

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_MEAS_PWR_DET_OFF 0x1E02

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_MEAS_PWR_DET_CLR 0x1E03

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_EXT_TRIG_EN 0x1F01

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_EXT_TRIG_DIS 0x1F02

  See Table 1 in [1].
- #define ADI_ADAR690x_FW_RFPLL_BOW_CAL 0x307

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_RFPLL_RAMP_SETUP 0x30C

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_LOCHAIN_CAL 0x602

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_FLASH_ADC_CAL 0x705

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_HPF_CAL 0x805

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_HPF_CAL_SINGLE_CHAN 0x806

  Undocumented internal function.
- #define ADI_ADAR690x_FW_ADC_PHASE_CAL 0xA05

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_PGA_CAL 0xB05

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_TXPA_CAL 0xC01

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_TXPA_ADJ 0xC02

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_RXGAIN_CAL 0x1801

  See Table 2 in [1].
- #define ADI_ADAR690x_FW_PWR_SUP_CHK 0xD01

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_CRC_CHK 0xD02

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_RX_BASEBAND_CHK 0xD03

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_RX_CHAIN_CHK 0xD04

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_TX_PWR_CHK 0xD06

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_AUXADC_DIAG_CHK 0xD07

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_PWR_DET_FAULT_CHK 0xD0B

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_TX_ISOL_CHK 0xD0E

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_TX_LOAD_CHK 0xD0F

See Table 3 in [1].

- #define ADI_ADAR690x_FW_RX_BASEBAND_LATENT_CHK 0xD12

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_ADCPLL_CHK 0xD13

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_RFPLL_CHK 0xD14

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_CRC_CALC_CHK 0xD15

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_RX_FILTER_CHK 0xD16

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_RX_OVERFLOW_CHK 0xD17

  See Table 3 in [1].
- #define ADI_ADAR690x_FW_RFPLL_PERIOD_CHK 0xD19

  See Table 3 in [1].
- #define ADI_ADAR690x_MINCYC_BOOTPARSE 226170

  Minimum cycle time for ADI_ADAR690x_FW_BOOTPARSE.
- #define ADI_ADAR690x_MAXCYC_BOOTPARSE 226170

  Maximum cycle time for ADI_ADAR690x_FW_BOOTPARSE.
- #define ADI_ADAR690x_MINCYC_ADCPLL_INIT 347774

  Minimum cycle time for ADI_ADAR690x_FW_ADCPLL_INIT.
- #define ADI_ADAR690x_MAXCYC_ADCPLL_INIT 7784856

  Maximum cycle time for ADI_ADAR690x_FW_ADCPLL_INIT.
- #define ADI_ADAR690x_MINCYC_ADCPLL_MUX_OUT 1685

  Minimum cycle time for ADI_ADAR690x_FW_ADCPLL_MUX_OUT.
- #define ADI_ADAR690x_MAXCYC_ADCPLL_MUX_OUT 1693

  Maximum cycle time for ADI_ADAR690x_FW_ADCPLL_MUX_OUT.
- #define ADI_ADAR690x_MINCYC_RFPLL_INIT 388013

  Minimum cycle time for ADI_ADAR690x_FW_RFPLL_INIT.
- #define ADI_ADAR690x_MAXCYC_RFPLL_INIT 8260402

  Maximum cycle time for ADI_ADAR690x_FW_RFPLL_INIT.
- #define ADI_ADAR690x_MINCYC_RFPLL_MUX_OUT 1704

  Minimum cycle time for ADI_ADAR690x_FW_RFPLL_MUX_OUT.
- #define ADI_ADAR690x_MAXCYC_RFPLL_MUX_OUT 1713

  Maximum cycle time for ADI_ADAR690x_FW_RFPLL_MUX_OUT.
- #define ADI_ADAR690x_MINCYC_RFPLL_LOCK 45935

  Minimum cycle time for ADI_ADAR690x_FW_RFPLL_LOCK.
- #define ADI_ADAR690x_MAXCYC_RFPLL_LOCK 88802

  Maximum cycle time for ADI_ADAR690x_FW_RFPLL_LOCK.
- #define ADI_ADAR690x_MINCYC_RAMP_TRIG 1571

  Minimum cycle time for ADI_ADAR690x_FW_RAMP_TRIG.
- #define ADI_ADAR690x_MAXCYC_RAMP_TRIG 1571

  Maximum cycle time for ADI_ADAR690x_FW_RAMP_TRIG.
- #define ADI_ADAR690x_MINCYC_CHIP_INIT 222952

  Minimum cycle time for ADI_ADAR690x_FW_CHIP_INIT.
- #define ADI_ADAR690x_MAXCYC_CHIP_INIT 222952

  Maximum cycle time for ADI_ADAR690x_FW_CHIP_INIT.
- #define ADI_ADAR690x_MINCYC_TEMP_MEASURE 55328

  Minimum cycle time for ADI_ADAR690x_FW_TEMP_MEASURE.
- #define ADI_ADAR690x_MAXCYC_TEMP_MEASURE 55328

  Maximum cycle time for ADI_ADAR690x_FW_TEMP_MEASURE.

- #define ADI_ADAR690x_MINCYC_CAL_READ 37766

    Minimum cycle time for ADI_ADAR690x_FW_CAL_READ.
- #define ADI_ADAR690x_MAXCYC_CAL_READ 37766

    Maximum cycle time for ADI_ADAR690x_FW_CAL_READ.
- #define ADI_ADAR690x_MINCYC_ADCPLL_ALIGN 0

    Minimum cycle time for ADI_ADAR690x_FW_ADCPLL_ALIGN.
- #define ADI_ADAR690x_MAXCYC_ADCPLL_ALIGN 0

    Maximum cycle time for ADI_ADAR690x_FW_ADCPLL_ALIGN.
- #define ADI_ADAR690x_MINCYC_POWER_MANAGE 228479

    Minimum cycle time for ADI_ADAR690x_FW_POWER_MANAGE.
- #define ADI_ADAR690x_MAXCYC_POWER_MANAGE 228479

    Maximum cycle time for ADI_ADAR690x_FW_POWER_MANAGE.
- #define ADI_ADAR690x_MINCYC_RMW 1650

    Minimum cycle time for ADI_ADAR690x_FW_RMW.
- #define ADI_ADAR690x_MAXCYC_RMW 1650

    Maximum cycle time for ADI_ADAR690x_FW_RMW.
- #define ADI_ADAR690x_MINCYC_MEAS_PWR_DET 0

    Minimum cycle time for ADI_ADAR690x_FW_MEAS_PWR_DET.
- #define ADI_ADAR690x_MAXCYC_MEAS_PWR_DET 0

    Maximum cycle time for ADI_ADAR690x_FW_MEAS_PWR_DET.
- #define ADI_ADAR690x_MINCYC_MEAS_PWR_DET_OFF 50624

    Minimum cycle time for ADI_ADAR690x_FW_MEAS_PWR_DET_OFF.
- #define ADI_ADAR690x_MAXCYC_MEAS_PWR_DET_OFF 50624

    Maximum cycle time for ADI_ADAR690x_FW_MEAS_PWR_DET_OFF.
- #define ADI_ADAR690x_MINCYC_MEAS_PWR_DET_CLR 1988

    Minimum cycle time for ADI_ADAR690x_FW_MEAS_PWR_DET_CLR.
- #define ADI_ADAR690x_MAXCYC_MEAS_PWR_DET_CLR 1988

    Maximum cycle time for ADI_ADAR690x_FW_MEAS_PWR_DET_CLR.
- #define ADI_ADAR690x_MINCYC_EXT_TRIG_EN 2277

    Minimum cycle time for ADI_ADAR690x_FW_EXT_TRIG_EN.
- #define ADI_ADAR690x_MAXCYC_EXT_TRIG_EN 2277

    Maximum cycle time for ADI_ADAR690x_FW_EXT_TRIG_EN.
- #define ADI_ADAR690x_MINCYC_EXT_TRIG_DIS 2277

    Minimum cycle time for ADI_ADAR690x_FW_EXT_TRIG_DIS.
- #define ADI_ADAR690x_MAXCYC_EXT_TRIG_DIS 2277

    Maximum cycle time for ADI_ADAR690x_FW_EXT_TRIG_DIS.
- #define ADI_ADAR690x_MINCYC_RFPLL_BOW_CAL 69353

    Minimum cycle time for ADI_ADAR690x_FW_RFPLL_BOW_CAL.
- #define ADI_ADAR690x_MAXCYC_RFPLL_BOW_CAL 111609

    Maximum cycle time for ADI_ADAR690x_FW_RFPLL_BOW_CAL.
- #define ADI_ADAR690x_MINCYC_RFPLL_RAMP_SETUP 187343

    Minimum cycle time for ADI_ADAR690x_FW_RFPLL_RAMP_SETUP.
- #define ADI_ADAR690x_MAXCYC_RFPLL_RAMP_SETUP 382398

    Maximum cycle time for ADI_ADAR690x_FW_RFPLL_RAMP_SETUP.
- #define ADI_ADAR690x_MINCYC_LOCHAIN_CAL 375766

    Minimum cycle time for ADI_ADAR690x_FW_LOCHAIN_CAL.
- #define ADI_ADAR690x_MAXCYC_LOCHAIN_CAL 375766

    Maximum cycle time for ADI_ADAR690x_FW_LOCHAIN_CAL.
- #define ADI_ADAR690x_MINCYC_FLASH_ADC_CAL 173965

    Minimum cycle time for ADI_ADAR690x_FW_FLASH_ADC_CAL.
- #define ADI_ADAR690x_MAXCYC_FLASH_ADC_CAL 173965

Maximum cycle time for ADI_ADAR690x_FW_FLASH_ADC_CAL.

- #define ADI_ADAR690x_MINCYC_HPF_CAL 223669

  Minimum cycle time for ADI_ADAR690x_FW_HPF_CAL.
- #define ADI_ADAR690x_MAXCYC_HPF_CAL 384914

  Maximum cycle time for ADI_ADAR690x_FW_HPF_CAL.
- #define ADI_ADAR690x_MINCYC_HPF_CAL_SINGLE_CHAN 0

  Minimum cycle time for ADI_ADAR690x_FW_HPF_CAL_SINGLE_CHAN.
- #define ADI_ADAR690x_MAXCYC_HPF_CAL_SINGLE_CHAN 0

  Maximum cycle time for ADI_ADAR690x_FW_HPF_CAL_SINGLE_CHAN.
- #define ADI_ADAR690x_MINCYC_ADC_PHASE_CAL 175759

  Minimum cycle time for ADI_ADAR690x_FW_ADC_PHASE_CAL.
- #define ADI_ADAR690x_MAXCYC_ADC_PHASE_CAL 175759

  Maximum cycle time for ADI_ADAR690x_FW_ADC_PHASE_CAL.
- #define ADI_ADAR690x_MINCYC_PGA_CAL 28941

  Minimum cycle time for ADI_ADAR690x_FW_PGA_CAL.
- #define ADI_ADAR690x_MAXCYC_PGA_CAL 28941

  Maximum cycle time for ADI_ADAR690x_FW_PGA_CAL.
- #define ADI_ADAR690x_MINCYC_TXPA_CAL 438563

  Minimum cycle time for ADI_ADAR690x_FW_TXPA_CAL.
- #define ADI_ADAR690x_MAXCYC_TXPA_CAL 438563

  Maximum cycle time for ADI_ADAR690x_FW_TXPA_CAL.
- #define ADI_ADAR690x_MINCYC_TXPA_ADJ 438563

  Minimum cycle time for ADI_ADAR690x_FW_TXPA_ADJ.
- #define ADI_ADAR690x_MAXCYC_TXPA_ADJ 438563

  Maximum cycle time for ADI_ADAR690x_FW_TXPA_ADJ.
- #define ADI_ADAR690x_MINCYC_RXGAIN_CAL 1962

  Minimum cycle time for ADI_ADAR690x_FW_RXGAIN_CAL.
- #define ADI_ADAR690x_MAXCYC_RXGAIN_CAL 1962

  Maximum cycle time for ADI_ADAR690x_FW_RXGAIN_CAL.
- #define ADI_ADAR690x_MINCYC_PWR_SUP_CHK 5341

  Minimum cycle time for ADI_ADAR690x_FW_PWR_SUP_CHK.
- #define ADI_ADAR690x_MAXCYC_PWR_SUP_CHK 74929

  Maximum cycle time for ADI_ADAR690x_FW_PWR_SUP_CHK.
- #define ADI_ADAR690x_MINCYC_CRC_CHK 800

  Minimum cycle time for ADI_ADAR690x_FW_CRC_CHK.
- #define ADI_ADAR690x_MAXCYC_CRC_CHK 400550

  Maximum cycle time for ADI_ADAR690x_FW_CRC_CHK.
- #define ADI_ADAR690x_MINCYC_RX_BASEBAND_CHK 701027

  Minimum cycle time for ADI_ADAR690x_FW_RX_BASEBAND_CHK.
- #define ADI_ADAR690x_MAXCYC_RX_BASEBAND_CHK 798531

  Maximum cycle time for ADI_ADAR690x_FW_RX_BASEBAND_CHK.
- #define ADI_ADAR690x_MINCYC_RX_CHAIN_CHK 430846

  Minimum cycle time for ADI_ADAR690x_FW_RX_CHAIN_CHK.
- #define ADI_ADAR690x_MAXCYC_RX_CHAIN_CHK 527713

  Maximum cycle time for ADI_ADAR690x_FW_RX_CHAIN_CHK.
- #define ADI_ADAR690x_MINCYC_TX_PWR_CHK 35780

  Minimum cycle time for ADI_ADAR690x_FW_TX_PWR_CHK.
- #define ADI_ADAR690x_MAXCYC_TX_PWR_CHK 35780

  Maximum cycle time for ADI_ADAR690x_FW_TX_PWR_CHK.
- #define ADI_ADAR690x_MINCYC_AUXADC_DIAG_CHK 9283

  Minimum cycle time for ADI_ADAR690x_FW_AUXADC_DIAG_CHK.

- #define ADI_ADAR690x_MAXCYC_AUXADC_DIAG_CHK 9283

  Maximum cycle time for ADI_ADAR690x_FW_AUXADC_DIAG_CHK.
- #define ADI_ADAR690x_MINCYC_PWR_DET_FAULT_CHK 20799

  Minimum cycle time for ADI_ADAR690x_FW_PWR_DET_FAULT_CHK.
- #define ADI_ADAR690x_MAXCYC_PWR_DET_FAULT_CHK 20799

  Maximum cycle time for ADI_ADAR690x_FW_PWR_DET_FAULT_CHK.
- #define ADI_ADAR690x_MINCYC_TX_ISOL_CHK 26608

  Minimum cycle time for ADI_ADAR690x_FW_TX_ISOL_CHK.
- #define ADI_ADAR690x_MAXCYC_TX_ISOL_CHK 26608

  Maximum cycle time for ADI_ADAR690x_FW_TX_ISOL_CHK.
- #define ADI_ADAR690x_MINCYC_TX_LOAD_CHK 55730

  Minimum cycle time for ADI_ADAR690x_FW_TX_LOAD_CHK.
- #define ADI_ADAR690x_MAXCYC_TX_LOAD_CHK 55730

  Maximum cycle time for ADI_ADAR690x_FW_TX_LOAD_CHK.
- #define ADI_ADAR690x_MINCYC_RX_BASEBAND_LATENT_CHK 353504

  Minimum cycle time for ADI_ADAR690x_FW_RX_BASEBAND_LATENT_CHK.
- #define ADI_ADAR690x_MAXCYC_RX_BASEBAND_LATENT_CHK 353504

  Maximum cycle time for ADI_ADAR690x_FW_RX_BASEBAND_LATENT_CHK.
- #define ADI_ADAR690x_MINCYC_ADCPLL_CHK 202665

  Minimum cycle time for ADI_ADAR690x_FW_ADCPLL_CHK.
- #define ADI_ADAR690x_MAXCYC_ADCPLL_CHK 202665

  Maximum cycle time for ADI_ADAR690x_FW_ADCPLL_CHK.
- #define ADI_ADAR690x_MINCYC_RFPLL_CHK 219808

  Minimum cycle time for ADI_ADAR690x_FW_RFPLL_CHK.
- #define ADI_ADAR690x_MAXCYC_RFPLL_CHK 219808

  Maximum cycle time for ADI_ADAR690x_FW_RFPLL_CHK.
- #define ADI_ADAR690x_MINCYC_CRC_CALC_CHK 475

  Minimum cycle time for ADI_ADAR690x_FW_CRC_CALC_CHK.
- #define ADI_ADAR690x_MAXCYC_CRC_CALC_CHK 475

  Maximum cycle time for ADI_ADAR690x_FW_CRC_CALC_CHK.
- #define ADI_ADAR690x_MINCYC_RX_FILTER_CHK 7078

  Minimum cycle time for ADI_ADAR690x_FW_RX_FILTER_CHK.
- #define ADI_ADAR690x_MAXCYC_RX_FILTER_CHK 7078

  Maximum cycle time for ADI_ADAR690x_FW_RX_FILTER_CHK.
- #define ADI_ADAR690x_MINCYC_RX_OVERFLOW_CHK 248

  Minimum cycle time for ADI_ADAR690x_FW_RX_OVERFLOW_CHK.
- #define ADI_ADAR690x_MAXCYC_RX_OVERFLOW_CHK 248

  Maximum cycle time for ADI_ADAR690x_FW_RX_OVERFLOW_CHK.
- #define ADI_ADAR690x_MINCYC_RFPLL_PERIOD_CHK 238

  Minimum cycle time for ADI_ADAR690x_FW_RFPLL_PERIOD_CHK.
- #define ADI_ADAR690x_MAXCYC_RFPLL_PERIOD_CHK 238

  Maximum cycle time for ADI_ADAR690x_FW_RFPLL_PERIOD_CHK.
- #define ADI_ADAR690x_MINCYC_CONFIG_COMPLETE 0

  Minimum cycle time for ADI_ADAR690x_FW_CONFIG_COMPLETE.
- #define ADI_ADAR690x_MAXCYC_CONFIG_COMPLETE 0

  Maximum cycle time for ADI_ADAR690x_FW_CONFIG_COMPLETE.
- #define ADI_ADAR690x_MINCYC_TASKLIST 0

  Minimum cycle time for ADI_ADAR690x_FW_TASKLIST.
- #define ADI_ADAR690x_MAXCYC_TASKLIST 0

  Maximum cycle time for ADI_ADAR690x_FW_TASKLIST.
- #define ADI_ADAR690x_OFF_FW_ICCM_LOAD_ADDR 0x1FDF4

Offset of ICCM file load address in ICCM file.
- #define ADI_ADAR690x_OFF_FW_DCCM_LOAD_ADDR 0x1FDF8

    Offset of DCCM file load address in ICCM file.
- #define ADI_ADAR690x_OFF_FW_VERSION 0x1FDFC

    Offset of firmware version in ICCM file.
- #define ADI_ADAR690x_DMA_BASE_POINTER 0x80008400UL

    *See **DMA (DIRECT MEMORY ACCESS)** in [1].*
- #define ADI_ADAR690x_DMA_AREA_BASE 0x80008600UL

    *See **DMA (DIRECT MEMORY ACCESS)** in [1].*
- #define ADI_ADAR690x_DMA_AREA_SIZE 14848UL

    *See **DMA (DIRECT MEMORY ACCESS)** in [1].*
- #define ADI_ADAR690x_CFG_SPI_CMD 0x80000800

    *See **SENDING COMMANDS OVER SPI** in [1].*
- #define ADI_ADAR690x_CFG_BASE 0x80006000

    *See **CONFIGURATION TABLE** in [1].*
- #define ADI_ADAR690x_STS_BASE 0x80004800

    *See **STATUS TABLE** [1].*
- #define ADI_ADAR690x_CFG_REF_FREQ_HZ (ADI_ADAR690x_CFG_BASE+0x0)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_RFPLL_LOOP_BW_HZ (ADI_ADAR690x_CFG_BASE+0x4)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_RAMP_START_FREQ_HZ (ADI_ADAR690x_CFG_BASE+0x8)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_RAMP_BW_MHZ (ADI_ADAR690x_CFG_BASE+0x10)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CASCADED (ADI_ADAR690x_CFG_BASE+0x20)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_POWER_DOWN_AUTO_US (ADI_ADAR690x_CFG_BASE+0x24)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0x28)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0x2C)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0x30)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0x34)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0x38)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0x3C)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0x40)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0x44)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0x48)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0x4C)

    See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0x50)

    See Table 4 in [1].

- #define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0x54)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PWR_TX0 (ADI_ADAR690x_CFG_BASE+0x5C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PWR_TX1 (ADI_ADAR690x_CFG_BASE+0x60)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PWR_TX2 (ADI_ADAR690x_CFG_BASE+0x64)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CRC_USER_AREA_ADDR (ADI_ADAR690x_CFG_BASE+0x68)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CRC_USER_AREA_COUNT (ADI_ADAR690x_CFG_BASE+0x6C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_TIMING_COMP_EN (ADI_ADAR690x_CFG_BASE+0x70)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_VDDIO_MIN_VOLT (ADI_ADAR690x_CFG_BASE+0x88)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_VDDIO_MAX_VOLT (ADI_ADAR690x_CFG_BASE+0xB4)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0xDC)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0xE0)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0xE4)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0xE8)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PWR_SUP_CHK (ADI_ADAR690x_CFG_BASE+0xEC)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PAT_SEL_TEST_TONE (ADI_ADAR690x_CFG_BASE+0x100)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CHANNEL_SEL_TEST_TONE (ADI_ADAR690x_CFG_BASE+0x104)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_POWER_MANAGE (ADI_ADAR690x_CFG_BASE+0x108)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CONTINUOUS_PA (ADI_ADAR690x_CFG_BASE+0x10C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PWRDN_INTERRUPT (ADI_ADAR690x_CFG_BASE+0x110)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_0 (ADI_ADAR690x_CFG_BASE+0x13C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_1 (ADI_ADAR690x_CFG_BASE+0x140)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_2 (ADI_ADAR690x_CFG_BASE+0x144)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_3 (ADI_ADAR690x_CFG_BASE+0x148)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_4 (ADI_ADAR690x_CFG_BASE+0x14C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_5 (ADI_ADAR690x_CFG_BASE+0x150)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_OP_PWR_TX0 (ADI_ADAR690x_CFG_BASE+0x154)

See Table 4 in [1].
- #define ADI_ADAR690x_CFG_OP_PWR_TX1 (ADI_ADAR690x_CFG_BASE+0x158)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_OP_PWR_TX2 (ADI_ADAR690x_CFG_BASE+0x15C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_TX_LOAD_T0_TX0 (ADI_ADAR690x_CFG_BASE+0x16C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_TX_LOAD_T0_TX1 (ADI_ADAR690x_CFG_BASE+0x170)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_TX_LOAD_T0_TX2 (ADI_ADAR690x_CFG_BASE+0x174)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_RFPLL_PERIOD_LOW_LIM (ADI_ADAR690x_CFG_BASE+0x180)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_RFPLL_PERIOD_HIGH_LIM (ADI_ADAR690x_CFG_BASE+0x184)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_RX_CHAIN_CHK (ADI_ADAR690x_CFG_BASE+0x19C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_PWR_DET_MEAS (ADI_ADAR690x_CFG_BASE+0x1A0)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_SYS_CAL_ENABLE (ADI_ADAR690x_CFG_BASE+0x1A4)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_DMA_OFFSETS_SEL (ADI_ADAR690x_CFG_BASE+0x1A8)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_LOCHAIN_MST_SLV_SYNC (ADI_ADAR690x_CFG_BASE+0x1B0)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CRC_GROUP_SELECT (ADI_ADAR690x_CFG_BASE+0x304)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CRC_SIGNATURE_SELECT (ADI_ADAR690x_CFG_BASE+0x308)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CONFIG_BLOCK_LEN (ADI_ADAR690x_CFG_BASE+0x30C)

  See Table 4 in [1].
- #define ADI_ADAR690x_CFG_CONFIG_BLOCK_CRC (ADI_ADAR690x_CFG_BASE+0x310)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_REF_FREQ_HZ (40000000UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_REF_FREQ_HZ (80000000UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_RFPLL_LOOP_BW_HZ (100000UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_RFPLL_LOOP_BW_HZ (1000000UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_RAMP_START_FREQ_HZ (76000000000ULL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_RAMP_START_FREQ_HZ (81000000000ULL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_RAMP_BW_MHZ (-5000.0F)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_RAMP_BW_MHZ (5000.0F)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_CASCADED (0UL)

  See Table 4 in [1].

- #define ADI_ADAR690x_MAX_CASCADED (1UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_POWER_DOWN_AUTO_US (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_POWER_DOWN_AUTO_US (1000000UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN0 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN0 (6UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN1 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN1 (6UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN2 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN2 (6UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN3 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN3 (6UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN0 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN0 (2UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN1 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN1 (2UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN2 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN2 (2UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN3 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN3 (2UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN0 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN0 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN1 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN1 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN2 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN2 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN3 (0UL)

See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN3 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PWR_TX0 (-10L)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PWR_TX0 (0L)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PWR_TX1 (-10L)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PWR_TX1 (0L)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PWR_TX2 (-10L)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PWR_TX2 (0L)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_TIMING_COMP_EN (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_TIMING_COMP_EN (1UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN0 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN0 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN1 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN1 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN2 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN2 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN3 (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN3 (3UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_PAT_SEL_TEST_TONE (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_PAT_SEL_TEST_TONE (63UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_CHANNEL_SEL_TEST_TONE (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_CHANNEL_SEL_TEST_TONE (15UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_SYS_CAL_ENABLE (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_SYS_CAL_ENABLE (1UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MIN_DMA_OFFSETS_SEL (0UL)

  See Table 4 in [1].
- #define ADI_ADAR690x_MAX_DMA_OFFSETS_SEL (2UL)

  See Table 4 in [1].

- #define ADI_ADAR690x_STS_TX0_TEMP (ADI_ADAR690x_STS_BASE+0x0)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_TX1_TEMP (ADI_ADAR690x_STS_BASE+0x4)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_TX2_TEMP (ADI_ADAR690x_STS_BASE+0x8)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX0_TEMP (ADI_ADAR690x_STS_BASE+0xC)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX1_TEMP (ADI_ADAR690x_STS_BASE+0x10)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX2_TEMP (ADI_ADAR690x_STS_BASE+0x14)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX3_TEMP (ADI_ADAR690x_STS_BASE+0x18)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_LOCHAIN_TEMP (ADI_ADAR690x_STS_BASE+0x1C)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_TX_LOAD_T0_TX0 (ADI_ADAR690x_STS_BASE+0x34)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_TX_LOAD_T0_TX1 (ADI_ADAR690x_STS_BASE+0x38)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_TX_LOAD_T0_TX2 (ADI_ADAR690x_STS_BASE+0x3C)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_0 (ADI_ADAR690x_STS_BASE+0x214)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_1 (ADI_ADAR690x_STS_BASE+0x218)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_2 (ADI_ADAR690x_STS_BASE+0x21C)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_3 (ADI_ADAR690x_STS_BASE+0x220)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_4 (ADI_ADAR690x_STS_BASE+0x224)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_5 (ADI_ADAR690x_STS_BASE+0x228)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_OP_PWR_TX0 (ADI_ADAR690x_STS_BASE+0x230)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_OP_PWR_TX1 (ADI_ADAR690x_STS_BASE+0x234)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_OP_PWR_TX2 (ADI_ADAR690x_STS_BASE+0x238)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_VDD_DCO_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x248)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_AVDD_09P_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x24C)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_DVDD_09P_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x258)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_AVDD_1P8_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x264)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_VDDIO_1P8_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x274)
  See Table 5 in [1].
- #define ADI_ADAR690x_STS_VDDIO_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x27C)

See Table 5 in [1].

- #define ADI_ADAR690x_STS_CRC_PASS (ADI_ADAR690x_STS_BASE+0x3D4)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_CRC_FAIL (ADI_ADAR690x_STS_BASE+0x3D8)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP0 (ADI_ADAR690x_STS_BASE+0x3FC)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP1 (ADI_ADAR690x_STS_BASE+0x400)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP2 (ADI_ADAR690x_STS_BASE+0x404)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP3 (ADI_ADAR690x_STS_BASE+0x408)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP4 (ADI_ADAR690x_STS_BASE+0x40C)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP5 (ADI_ADAR690x_STS_BASE+0x410)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP6 (ADI_ADAR690x_STS_BASE+0x414)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP7 (ADI_ADAR690x_STS_BASE+0x418)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP8 (ADI_ADAR690x_STS_BASE+0x41C)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP9 (ADI_ADAR690x_STS_BASE+0x420)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP10 (ADI_ADAR690x_STS_BASE+0x424)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP11 (ADI_ADAR690x_STS_BASE+0x428)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP12 (ADI_ADAR690x_STS_BASE+0x42C)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP13 (ADI_ADAR690x_STS_BASE+0x430)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP14 (ADI_ADAR690x_STS_BASE+0x434)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP15 (ADI_ADAR690x_STS_BASE+0x438)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP16 (ADI_ADAR690x_STS_BASE+0x43C)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_ADCPLL_FREQ_HZ (ADI_ADAR690x_STS_BASE+0x4B8)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_RFPLL_FREQ_HZ (ADI_ADAR690x_STS_BASE+0x4D0)

  See Table 5 in [1].
- #define ADI_ADAR690x_STS_DMA_OFFSETS_LUT (ADI_ADAR690x_STS_BASE+0xE1C)

  See Table 5 in [1].
- #define ADI_ADAR690x_BITP_PWR_DET_MEAS_LO 0

  *See **ADI_ADAR690x_FW_MEAS_PWR_DET** in [1].*
- #define ADI_ADAR690x_BITP_PWR_DET_MEAS_TXACTIVECOUPLED 1

  *See **ADI_ADAR690x_FW_MEAS_PWR_DET** in [1].*
- #define ADI_ADAR690x_BITP_PWR_DET_MEAS_TXACTIVEREFLECTED 4

  *See **ADI_ADAR690x_FW_MEAS_PWR_DET** in [1].*

- #define ADI_ADAR690x_BITP_PWR_DET_MEAS_CLRPREV 19

    *See **ADI_ADAR690x_FW_MEAS_PWR_DET** in [1].*
- #define ADI_ADAR690x_BITP_PWR_DET_MEAS_PDMEASPERCHIRP 20

    *See **ADI_ADAR690x_FW_MEAS_PWR_DET** in [1].*
- #define ADI_ADAR690x_BITP_POWER_RX0 0

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_RX1 1

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_RX2 2

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_RX3 3

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_AFE0 8

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_AFE1 9

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_AFE2 10

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_AFE3 11

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_TX0 16

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_TX1 17

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_TX2 18

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_RFPLL 20

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_LO 22

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_LOAMP 23

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_AUXADC 24

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_DATAPORT 25

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_PERMANENT 27

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_DOWN_REQUEST 29

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_RESTORE_REQUEST 30

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_POWER_UP_REQUEST 31

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_CONTINUOUS_PA_TX0 0

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_CONTINUOUS_PA_TX1 1

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITP_CONTINUOUS_PA_TX2 2

    *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_RX0 0x00000001

*See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*

- #define ADI_ADAR690x_BITM_POWER_RX1 0x00000002

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_RX2 0x00000004

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_RX3 0x00000008

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_AFE0 0x00000100

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_AFE1 0x00000200

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_AFE2 0x00000400

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_AFE3 0x00000800

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_TX0 0x00010000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_TX1 0x00020000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_TX2 0x00040000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_RFPLL 0x00100000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_LO 0x00400000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_LOAMP 0x00800000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_AUXADC 0x01000000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_DATAPORT 0x02000000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_PERMANENT 0x08000000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_DOWN_REQUEST 0x20000000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_RESTORE_REQUEST 0x40000000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_POWER_UP_REQUEST 0x80000000

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_CONTINUOUS_PA_TX0 0x00000001

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_CONTINUOUS_PA_TX1 0x00000002

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*
- #define ADI_ADAR690x_BITM_CONTINUOUS_PA_TX2 0x00000004

  *See **ADI_ADAR690x_CFG_POWER_MANAGE** in [1].*

6.1.1 Detailed Description

Public C interface to the firmware.

6.1.2 Macro Definition Documentation

6.1.2.1 ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0xDC)
```

See Table 4 in [1].

6.1.2.2 ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0xE0)
```

See Table 4 in [1].

6.1.2.3 ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0xE4)
```

See Table 4 in [1].

6.1.2.4 ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_CFG_ADC_GAIN_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0xE8)
```

See Table 4 in [1].

6.1.2.5 ADI_ADAR690x_CFG_BASE

```
#define ADI_ADAR690x_CFG_BASE 0x80006000
```

See **CONFIGURATION TABLE** in [1].

6.1.2.6 ADI_ADAR690x_CFG_CASCADED

```
#define ADI_ADAR690x_CFG_CASCADED (ADI_ADAR690x_CFG_BASE+0x20)
```

See Table 4 in [1].

6.1.2.7 ADI_ADAR690x_CFG_CHANNEL_SEL_TEST_TONE

```
#define ADI_ADAR690x_CFG_CHANNEL_SEL_TEST_TONE (ADI_ADAR690x_CFG_BASE+0x104)
```

See Table 4 in [1].

6.1.2.8 ADI_ADAR690x_CFG_CONFIG_BLOCK_CRC

```
#define ADI_ADAR690x_CFG_CONFIG_BLOCK_CRC (ADI_ADAR690x_CFG_BASE+0x310)
```

See Table 4 in [1].

6.1.2.9 ADI_ADAR690x_CFG_CONFIG_BLOCK_LEN

```
#define ADI_ADAR690x_CFG_CONFIG_BLOCK_LEN (ADI_ADAR690x_CFG_BASE+0x30C)
```

See Table 4 in [1].

6.1.2.10 ADI_ADAR690x_CFG_CONTINUOUS_PA

```
#define ADI_ADAR690x_CFG_CONTINUOUS_PA (ADI_ADAR690x_CFG_BASE+0x10C)
```

See Table 4 in [1].

6.1.2.11 ADI_ADAR690x_CFG_CRC_GROUP_SELECT

```
#define ADI_ADAR690x_CFG_CRC_GROUP_SELECT (ADI_ADAR690x_CFG_BASE+0x304)
```

See Table 4 in [1].

6.1.2.12 ADI_ADAR690x_CFG_CRC_SIGNATURE_SELECT

```
#define ADI_ADAR690x_CFG_CRC_SIGNATURE_SELECT (ADI_ADAR690x_CFG_BASE+0x308)
```

See Table 4 in [1].

6.1.2.13 ADI_ADAR690x_CFG_CRC_USER_AREA_ADDR

```
#define ADI_ADAR690x_CFG_CRC_USER_AREA_ADDR (ADI_ADAR690x_CFG_BASE+0x68)
```

See Table 4 in [1].

6.1.2.14 ADI_ADAR690x_CFG_CRC_USER_AREA_COUNT

```
#define ADI_ADAR690x_CFG_CRC_USER_AREA_COUNT (ADI_ADAR690x_CFG_BASE+0x6C)
```

See Table 4 in [1].

6.1.2.15 ADI_ADAR690x_CFG_DMA_OFFSETS_SEL

```
#define ADI_ADAR690x_CFG_DMA_OFFSETS_SEL (ADI_ADAR690x_CFG_BASE+0x1A8)
```

See Table 4 in [1].

6.1.2.16 ADI_ADAR690x_CFG_HPF_FC_RX_CHAN0

```
#define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0x28)
```

See Table 4 in [1].

6.1.2.17 ADI_ADAR690x_CFG_HPF_FC_RX_CHAN1

```
#define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0x2C)
```

See Table 4 in [1].

6.1.2.18 ADI_ADAR690x_CFG_HPF_FC_RX_CHAN2

```
#define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0x30)
```

See Table 4 in [1].

6.1.2.19 ADI_ADAR690x_CFG_HPF_FC_RX_CHAN3

```
#define ADI_ADAR690x_CFG_HPF_FC_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0x34)
```

See Table 4 in [1].

6.1.2.20 ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0x38)
```

See Table 4 in [1].

6.1.2.21 ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0x3C)
```

See Table 4 in [1].

6.1.2.22 ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0x40)
```

See Table 4 in [1].

6.1.2.23 ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_CFG_HPF_GAIN_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0x44)
```

See Table 4 in [1].

6.1.2.24 ADI_ADAR690x_CFG_LOCHAIN_MST_SLV_SYNC

```
#define ADI_ADAR690x_CFG_LOCHAIN_MST_SLV_SYNC (ADI_ADAR690x_CFG_BASE+0x1B0)
```

See Table 4 in [1].

6.1.2.25 ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_0

```
#define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_0 (ADI_ADAR690x_CFG_BASE+0x13C)
```

See Table 4 in [1].

6.1.2.26 ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_1

```
#define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_1 (ADI_ADAR690x_CFG_BASE+0x140)
```

See Table 4 in [1].

6.1.2.27 ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_2

```
#define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_2 (ADI_ADAR690x_CFG_BASE+0x144)
```

See Table 4 in [1].

6.1.2.28 ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_3

```
#define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_3 (ADI_ADAR690x_CFG_BASE+0x148)
```

See Table 4 in [1].

6.1.2.29 ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_4

```
#define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_4 (ADI_ADAR690x_CFG_BASE+0x14C)
```

See Table 4 in [1].

6.1.2.30 ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_5

```
#define ADI_ADAR690x_CFG_MEAS_REL_GAIN_MISMATCH_5 (ADI_ADAR690x_CFG_BASE+0x150)
```

See Table 4 in [1].

6.1.2.31 ADI_ADAR690x_CFG_OP_PWR_TX0

```
#define ADI_ADAR690x_CFG_OP_PWR_TX0 (ADI_ADAR690x_CFG_BASE+0x154)
```

See Table 4 in [1].

6.1.2.32 ADI_ADAR690x_CFG_OP_PWR_TX1

```
#define ADI_ADAR690x_CFG_OP_PWR_TX1 (ADI_ADAR690x_CFG_BASE+0x158)
```

See Table 4 in [1].

6.1.2.33 ADI_ADAR690x_CFG_OP_PWR_TX2

```
#define ADI_ADAR690x_CFG_OP_PWR_TX2 (ADI_ADAR690x_CFG_BASE+0x15C)
```

See Table 4 in [1].

6.1.2.34 ADI_ADAR690x_CFG_PAT_SEL_TEST_TONE

```
#define ADI_ADAR690x_CFG_PAT_SEL_TEST_TONE (ADI_ADAR690x_CFG_BASE+0x100)
```

See Table 4 in [1].

6.1.2.35 ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN0 (ADI_ADAR690x_CFG_BASE+0x48)
```

See Table 4 in [1].

6.1.2.36 ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN1 (ADI_ADAR690x_CFG_BASE+0x4C)
```

See Table 4 in [1].

6.1.2.37 ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN2 (ADI_ADAR690x_CFG_BASE+0x50)
```

See Table 4 in [1].

6.1.2.38 ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_CFG_PGA_GAIN_RX_CHAN3 (ADI_ADAR690x_CFG_BASE+0x54)
```

See Table 4 in [1].

6.1.2.39 ADI_ADAR690x_CFG_POWER_DOWN_AUTO_US

```
#define ADI_ADAR690x_CFG_POWER_DOWN_AUTO_US (ADI_ADAR690x_CFG_BASE+0x24)
```

See Table 4 in [1].

6.1.2.40 ADI_ADAR690x_CFG_POWER_MANAGE

```
#define ADI_ADAR690x_CFG_POWER_MANAGE (ADI_ADAR690x_CFG_BASE+0x108)
```

See Table 4 in [1].

6.1.2.41 ADI_ADAR690x_CFG_PWR_DET_MEAS

```
#define ADI_ADAR690x_CFG_PWR_DET_MEAS (ADI_ADAR690x_CFG_BASE+0x1A0)
```

See Table 4 in [1].

6.1.2.42 ADI_ADAR690x_CFG_PWR_SUP_CHK

```
#define ADI_ADAR690x_CFG_PWR_SUP_CHK (ADI_ADAR690x_CFG_BASE+0xEC)
```

See Table 4 in [1].

6.1.2.43 ADI_ADAR690x_CFG_PWR_TX0

```
#define ADI_ADAR690x_CFG_PWR_TX0 (ADI_ADAR690x_CFG_BASE+0x5C)
```

See Table 4 in [1].

6.1.2.44 ADI_ADAR690x_CFG_PWR_TX1

```
#define ADI_ADAR690x_CFG_PWR_TX1 (ADI_ADAR690x_CFG_BASE+0x60)
```

See Table 4 in [1].

6.1.2.45 ADI_ADAR690x_CFG_PWR_TX2

```
#define ADI_ADAR690x_CFG_PWR_TX2 (ADI_ADAR690x_CFG_BASE+0x64)
```

See Table 4 in [1].

6.1.2.46 ADI_ADAR690x_CFG_PWRDN_INTERRUPT

```
#define ADI_ADAR690x_CFG_PWRDN_INTERRUPT (ADI_ADAR690x_CFG_BASE+0x110)
```

See Table 4 in [1].

6.1.2.47 ADI_ADAR690x_CFG_RAMP_BW_MHZ

```
#define ADI_ADAR690x_CFG_RAMP_BW_MHZ (ADI_ADAR690x_CFG_BASE+0x10)
```

See Table 4 in [1].

6.1.2.48 ADI_ADAR690x_CFG_RAMP_START_FREQ_HZ

```
#define ADI_ADAR690x_CFG_RAMP_START_FREQ_HZ (ADI_ADAR690x_CFG_BASE+0x8)
```

See Table 4 in [1].

6.1.2.49 ADI_ADAR690x_CFG_REF_FREQ_HZ

```
#define ADI_ADAR690x_CFG_REF_FREQ_HZ (ADI_ADAR690x_CFG_BASE+0x0)
```

See Table 4 in [1].

6.1.2.50 ADI_ADAR690x_CFG_RFPLL_LOOP_BW_HZ

```
#define ADI_ADAR690x_CFG_RFPLL_LOOP_BW_HZ (ADI_ADAR690x_CFG_BASE+0x4)
```

See Table 4 in [1].

6.1.2.51 ADI_ADAR690x_CFG_RFPLL_PERIOD_HIGH_LIM

```
#define ADI_ADAR690x_CFG_RFPLL_PERIOD_HIGH_LIM (ADI_ADAR690x_CFG_BASE+0x184)
```

See Table 4 in [1].

6.1.2.52 ADI_ADAR690x_CFG_RFPLL_PERIOD_LOW_LIM

```
#define ADI_ADAR690x_CFG_RFPLL_PERIOD_LOW_LIM (ADI_ADAR690x_CFG_BASE+0x180)
```

See Table 4 in [1].

6.1.2.53 ADI_ADAR690x_CFG_RX_CHAIN_CHK

```
#define ADI_ADAR690x_CFG_RX_CHAIN_CHK (ADI_ADAR690x_CFG_BASE+0x19C)
```

See Table 4 in [1].

6.1.2.54 ADI_ADAR690x_CFG_SPI_CMD

```
#define ADI_ADAR690x_CFG_SPI_CMD 0x80000800
```

See **SENDING COMMANDS OVER SPI** in [1].

6.1.2.55 ADI_ADAR690x_CFG_SYS_CAL_ENABLE

```
#define ADI_ADAR690x_CFG_SYS_CAL_ENABLE (ADI_ADAR690x_CFG_BASE+0x1A4)
```

See Table 4 in [1].

6.1.2.56 ADI_ADAR690x_CFG_TIMING_COMP_EN

```
#define ADI_ADAR690x_CFG_TIMING_COMP_EN (ADI_ADAR690x_CFG_BASE+0x70)
```

See Table 4 in [1].

6.1.2.57 ADI_ADAR690x_CFG_TX_LOAD_T0_TX0

```
#define ADI_ADAR690x_CFG_TX_LOAD_T0_TX0 (ADI_ADAR690x_CFG_BASE+0x16C)
```

See Table 4 in [1].

6.1.2.58 ADI_ADAR690x_CFG_TX_LOAD_T0_TX1

```
#define ADI_ADAR690x_CFG_TX_LOAD_T0_TX1 (ADI_ADAR690x_CFG_BASE+0x170)
```

See Table 4 in [1].

6.1.2.59 ADI_ADAR690x_CFG_TX_LOAD_T0_TX2

```
#define ADI_ADAR690x_CFG_TX_LOAD_T0_TX2 (ADI_ADAR690x_CFG_BASE+0x174)
```

See Table 4 in [1].

6.1.2.60 ADI_ADAR690x_CFG_VDDIO_MAX_VOLT

```
#define ADI_ADAR690x_CFG_VDDIO_MAX_VOLT (ADI_ADAR690x_CFG_BASE+0xB4)
```

See Table 4 in [1].

6.1.2.61 ADI_ADAR690x_CFG_VDDIO_MIN_VOLT

```
#define ADI_ADAR690x_CFG_VDDIO_MIN_VOLT (ADI_ADAR690x_CFG_BASE+0x88)
```

See Table 4 in [1].

6.1.2.62 ADI_ADAR690x_DMA_AREA_BASE

```
#define ADI_ADAR690x_DMA_AREA_BASE 0x80008600UL
```

See **DMA (DIRECT MEMORY ACCESS)** in [1].

6.1.2.63 ADI_ADAR690x_DMA_AREA_SIZE

```
#define ADI_ADAR690x_DMA_AREA_SIZE 14848UL
```

See **DMA (DIRECT MEMORY ACCESS)** in [1].

6.1.2.64 ADI_ADAR690x_DMA_BASE_POINTER

```
#define ADI_ADAR690x_DMA_BASE_POINTER 0x80008400UL
```

See **DMA (DIRECT MEMORY ACCESS)** in [1].

6.1.2.65 ADI_ADAR690x_FW_ADC_PHASE_CAL

```
#define ADI_ADAR690x_FW_ADC_PHASE_CAL 0xA05
```

See Table 2 in [1].

6.1.2.66 ADI_ADAR690x_FW_ADCPLL_ALIGN

```
#define ADI_ADAR690x_FW_ADCPLL_ALIGN 0x1501
```

See Table 1 in [1].

6.1.2.67 ADI_ADAR690x_FW_ADCPLL_CHK

```
#define ADI_ADAR690x_FW_ADCPLL_CHK 0xD13
```

See Table 3 in [1].

6.1.2.68 ADI_ADAR690x_FW_ADCPLL_INIT

```
#define ADI_ADAR690x_FW_ADCPLL_INIT 0x201
```

See Table 1 in [1].

6.1.2.69 ADI_ADAR690x_FW_ADCPLL_MUX_OUT

```
#define ADI_ADAR690x_FW_ADCPLL_MUX_OUT 0x203
```

See Table 1 in [1].

6.1.2.70 ADI_ADAR690x_FW_AUXADC_DIAG_CHK

```
#define ADI_ADAR690x_FW_AUXADC_DIAG_CHK 0xD07
```

See Table 3 in [1].

6.1.2.71 ADI_ADAR690x_FW_BOOTPARSE

```
#define ADI_ADAR690x_FW_BOOTPARSE 0x101
```

See Table 1 in [1].

6.1.2.72 ADI_ADAR690x_FW_CAL_READ

```
#define ADI_ADAR690x_FW_CAL_READ 0x1101
```

See Table 1 in [1].

6.1.2.73 ADI_ADAR690x_FW_CHIP_INIT

```
#define ADI_ADAR690x_FW_CHIP_INIT 0xF01
```

See Table 1 in [1].

6.1.2.74 ADI_ADAR690x_FW_CRC_CALC_CHK

```
#define ADI_ADAR690x_FW_CRC_CALC_CHK 0xD15
```

See Table 3 in [1].

6.1.2.75 ADI_ADAR690x_FW_CRC_CHK

```
#define ADI_ADAR690x_FW_CRC_CHK 0xD02
```

See Table 3 in [1].

6.1.2.76 ADI_ADAR690x_FW_EXT_TRIG_DIS

```
#define ADI_ADAR690x_FW_EXT_TRIG_DIS 0x1F02
```

See Table 1 in [1].

6.1.2.77 ADI_ADAR690x_FW_EXT_TRIG_EN

```
#define ADI_ADAR690x_FW_EXT_TRIG_EN 0x1F01
```

See Table 1 in [1].

6.1.2.78 ADI_ADAR690x_FW_FLASH_ADC_CAL

```
#define ADI_ADAR690x_FW_FLASH_ADC_CAL 0x705
```

See Table 2 in [1].

6.1.2.79 ADI_ADAR690x_FW_HPF_CAL

```
#define ADI_ADAR690x_FW_HPF_CAL 0x805
```

See Table 2 in [1].

6.1.2.80 ADI_ADAR690x_FW_HPF_CAL_SINGLE_CHAN

```
#define ADI_ADAR690x_FW_HPF_CAL_SINGLE_CHAN 0x806
```

Undocumented internal function.

6.1.2.81 ADI_ADAR690x_FW_LOCHAIN_CAL

```
#define ADI_ADAR690x_FW_LOCHAIN_CAL 0x602
```

See Table 2 in [1].

6.1.2.82 ADI_ADAR690x_FW_MEAS_PWR_DET

```
#define ADI_ADAR690x_FW_MEAS_PWR_DET 0x1E01
```

See Table 1 in [1].

6.1.2.83 ADI_ADAR690x_FW_MEAS_PWR_DET_CLR

```
#define ADI_ADAR690x_FW_MEAS_PWR_DET_CLR 0x1E03
```

See Table 1 in [1].

6.1.2.84 ADI_ADAR690x_FW_MEAS_PWR_DET_OFF

```
#define ADI_ADAR690x_FW_MEAS_PWR_DET_OFF 0x1E02
```

See Table 1 in [1].

6.1.2.85 ADI_ADAR690x_FW_PGA_CAL

```
#define ADI_ADAR690x_FW_PGA_CAL 0xB05
```

See Table 2 in [1].

6.1.2.86 ADI_ADAR690x_FW_POWER_MANAGE

```
#define ADI_ADAR690x_FW_POWER_MANAGE 0x1701
```

See Table 1 in [1].

6.1.2.87 ADI_ADAR690x_FW_PWR_DET_FAULT_CHK

```
#define ADI_ADAR690x_FW_PWR_DET_FAULT_CHK 0xD0B
```

See Table 3 in [1].

6.1.2.88 ADI_ADAR690x_FW_PWR_SUP_CHK

```
#define ADI_ADAR690x_FW_PWR_SUP_CHK 0xD01
```

See Table 3 in [1].

6.1.2.89 ADI_ADAR690x_FW_RAMP_TRIG

```
#define ADI_ADAR690x_FW_RAMP_TRIG 0x30A
```

See Table 1 in [1].

6.1.2.90 ADI_ADAR690x_FW_RFPLL_BOW_CAL

```
#define ADI_ADAR690x_FW_RFPLL_BOW_CAL 0x307
```

See Table 2 in [1].

6.1.2.91 ADI_ADAR690x_FW_RFPLL_CHK

```
#define ADI_ADAR690x_FW_RFPLL_CHK 0xD14
```

See Table 3 in [1].

6.1.2.92 ADI_ADAR690x_FW_RFPLL_INIT

```
#define ADI_ADAR690x_FW_RFPLL_INIT 0x301
```

See Table 1 in [1].

6.1.2.93 ADI_ADAR690x_FW_RFPLL_LOCK

```
#define ADI_ADAR690x_FW_RFPLL_LOCK 0x304
```

See Table 1 in [1].

6.1.2.94 ADI_ADAR690x_FW_RFPLL_MUX_OUT

```
#define ADI_ADAR690x_FW_RFPLL_MUX_OUT 0x303
```

See Table 1 in [1].

6.1.2.95 ADI_ADAR690x_FW_RFPLL_PERIOD_CHK

```
#define ADI_ADAR690x_FW_RFPLL_PERIOD_CHK 0xD19
```

See Table 3 in [1].

6.1.2.96 ADI_ADAR690x_FW_RFPLL_RAMP_SETUP

```
#define ADI_ADAR690x_FW_RFPLL_RAMP_SETUP 0x30C
```

See Table 2 in [1].

6.1.2.97 ADI_ADAR690x_FW_RMW

```
#define ADI_ADAR690x_FW_RMW 0x1901
```

See Table 1 in [1].

6.1.2.98 ADI_ADAR690x_FW_RX_BASEBAND_CHK

```
#define ADI_ADAR690x_FW_RX_BASEBAND_CHK 0xD03
```

See Table 3 in [1].

6.1.2.99 ADI_ADAR690x_FW_RX_BASEBAND_LATENT_CHK

```
#define ADI_ADAR690x_FW_RX_BASEBAND_LATENT_CHK 0xD12
```

See Table 3 in [1].

6.1.2.100 ADI_ADAR690x_FW_RX_CHAIN_CHK

```
#define ADI_ADAR690x_FW_RX_CHAIN_CHK 0xD04
```

See Table 3 in [1].

6.1.2.101 ADI_ADAR690x_FW_RX_FILTER_CHK

```
#define ADI_ADAR690x_FW_RX_FILTER_CHK 0xD16
```

See Table 3 in [1].

6.1.2.102 ADI_ADAR690x_FW_RX_OVERFLOW_CHK

```
#define ADI_ADAR690x_FW_RX_OVERFLOW_CHK 0xD17
```

See Table 3 in [1].

6.1.2.103 ADI_ADAR690x_FW_RXGAIN_CAL

```
#define ADI_ADAR690x_FW_RXGAIN_CAL 0x1801
```

See Table 2 in [1].

6.1.2.104 ADI_ADAR690x_FW_TEMP_MEASURE

```
#define ADI_ADAR690x_FW_TEMP_MEASURE 0x1001
```

See Table 1 in [1].

6.1.2.105 ADI_ADAR690x_FW_TX_ISOL_CHK

```
#define ADI_ADAR690x_FW_TX_ISOL_CHK 0xD0E
```

See Table 3 in [1].

6.1.2.106 ADI_ADAR690x_FW_TX_LOAD_CHK

```
#define ADI_ADAR690x_FW_TX_LOAD_CHK 0xD0F
```

See Table 3 in [1].

6.1.2.107 ADI_ADAR690x_FW_TX_PWR_CHK

```
#define ADI_ADAR690x_FW_TX_PWR_CHK 0xD06
```

See Table 3 in [1].

6.1.2.108 ADI_ADAR690x_FW_TXPA_ADJ

```
#define ADI_ADAR690x_FW_TXPA_ADJ 0xC02
```

See Table 2 in [1].

6.1.2.109 ADI_ADAR690x_FW_TXPA_CAL

```
#define ADI_ADAR690x_FW_TXPA_CAL 0xC01
```

See Table 2 in [1].

6.1.2.110 ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN0 (3UL)
```

See Table 4 in [1].

6.1.2.111 ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN1 (3UL)
```

See Table 4 in [1].

6.1.2.112 ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN2 (3UL)
```

See Table 4 in [1].

6.1.2.113 ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_MAX_ADC_GAIN_RX_CHAN3 (3UL)
```

See Table 4 in [1].

6.1.2.114 ADI_ADAR690x_MAX_CASCADED

```
#define ADI_ADAR690x_MAX_CASCADED (1UL)
```

See Table 4 in [1].

6.1.2.115 ADI_ADAR690x_MAX_CHANNEL_SEL_TEST_TONE

```
#define ADI_ADAR690x_MAX_CHANNEL_SEL_TEST_TONE (15UL)
```

See Table 4 in [1].

6.1.2.116 ADI_ADAR690x_MAX_DMA_OFFSETS_SEL

```
#define ADI_ADAR690x_MAX_DMA_OFFSETS_SEL (2UL)
```

See Table 4 in [1].

6.1.2.117 ADI_ADAR690x_MAX_HPF_FC_RX_CHAN0

```
#define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN0 (6UL)
```

See Table 4 in [1].

6.1.2.118 ADI_ADAR690x_MAX_HPF_FC_RX_CHAN1

```
#define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN1 (6UL)
```

See Table 4 in [1].

6.1.2.119 ADI_ADAR690x_MAX_HPF_FC_RX_CHAN2

```
#define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN2 (6UL)
```

See Table 4 in [1].

6.1.2.120 ADI_ADAR690x_MAX_HPF_FC_RX_CHAN3

```
#define ADI_ADAR690x_MAX_HPF_FC_RX_CHAN3 (6UL)
```

See Table 4 in [1].

6.1.2.121 ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN0 (2UL)
```

See Table 4 in [1].

6.1.2.122 ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN1 (2UL)
```

See Table 4 in [1].

6.1.2.123 ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN2 (2UL)
```

See Table 4 in [1].

6.1.2.124 ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_MAX_HPF_GAIN_RX_CHAN3 (2UL)
```

See Table 4 in [1].

6.1.2.125 ADI_ADAR690x_MAX_PAT_SEL_TEST_TONE

```
#define ADI_ADAR690x_MAX_PAT_SEL_TEST_TONE (63UL)
```

See Table 4 in [1].

6.1.2.126 ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN0 (3UL)
```

See Table 4 in [1].

6.1.2.127 ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN1 (3UL)
```

See Table 4 in [1].

6.1.2.128 ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN2 (3UL)
```

See Table 4 in [1].

6.1.2.129 ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_MAX_PGA_GAIN_RX_CHAN3 (3UL)
```

See Table 4 in [1].

6.1.2.130 ADI_ADAR690x_MAX_POWER_DOWN_AUTO_US

```
#define ADI_ADAR690x_MAX_POWER_DOWN_AUTO_US (1000000UL)
```

See Table 4 in [1].

6.1.2.131 ADI_ADAR690x_MAX_PWR_TX0

```
#define ADI_ADAR690x_MAX_PWR_TX0 (0L)
```

See Table 4 in [1].

6.1.2.132 ADI_ADAR690x_MAX_PWR_TX1

```
#define ADI_ADAR690x_MAX_PWR_TX1 (0L)
```

See Table 4 in [1].

6.1.2.133 ADI_ADAR690x_MAX_PWR_TX2

```
#define ADI_ADAR690x_MAX_PWR_TX2 (0L)
```

See Table 4 in [1].

6.1.2.134 ADI_ADAR690x_MAX_RAMP_BW_MHZ

```
#define ADI_ADAR690x_MAX_RAMP_BW_MHZ (5000.0F)
```

See Table 4 in [1].

6.1.2.135 ADI_ADAR690x_MAX_RAMP_START_FREQ_HZ

```
#define ADI_ADAR690x_MAX_RAMP_START_FREQ_HZ (81000000000ULL)
```

See Table 4 in [1].

6.1.2.136 ADI_ADAR690x_MAX_REF_FREQ_HZ

```
#define ADI_ADAR690x_MAX_REF_FREQ_HZ (80000000UL)
```

See Table 4 in [1].

6.1.2.137 ADI_ADAR690x_MAX_RFPLL_LOOP_BW_HZ

```
#define ADI_ADAR690x_MAX_RFPLL_LOOP_BW_HZ (1000000UL)
```

See Table 4 in [1].

6.1.2.138 ADI_ADAR690x_MAX_SYS_CAL_ENABLE

```
#define ADI_ADAR690x_MAX_SYS_CAL_ENABLE (1UL)
```

See Table 4 in [1].

6.1.2.139 ADI_ADAR690x_MAX_TIMING_COMP_EN

```
#define ADI_ADAR690x_MAX_TIMING_COMP_EN (1UL)
```

See Table 4 in [1].

6.1.2.140 ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN0 (0UL)
```

See Table 4 in [1].

6.1.2.141 ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN1 (0UL)
```

See Table 4 in [1].

6.1.2.142 ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN2 (0UL)
```

See Table 4 in [1].

6.1.2.143 ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_MIN_ADC_GAIN_RX_CHAN3 (0UL)
```

See Table 4 in [1].

6.1.2.144 ADI_ADAR690x_MIN_CASCADED

```
#define ADI_ADAR690x_MIN_CASCADED (0UL)
```

See Table 4 in [1].

6.1.2.145 ADI_ADAR690x_MIN_CHANNEL_SEL_TEST_TONE

```
#define ADI_ADAR690x_MIN_CHANNEL_SEL_TEST_TONE (0UL)
```

See Table 4 in [1].

6.1.2.146 ADI_ADAR690x_MIN_DMA_OFFSETS_SEL

```
#define ADI_ADAR690x_MIN_DMA_OFFSETS_SEL (0UL)
```

See Table 4 in [1].

6.1.2.147 ADI_ADAR690x_MIN_HPF_FC_RX_CHAN0

```
#define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN0 (0UL)
```

See Table 4 in [1].

6.1.2.148 ADI_ADAR690x_MIN_HPF_FC_RX_CHAN1

```
#define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN1 (0UL)
```

See Table 4 in [1].

6.1.2.149 ADI_ADAR690x_MIN_HPF_FC_RX_CHAN2

```
#define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN2 (0UL)
```

See Table 4 in [1].

6.1.2.150 ADI_ADAR690x_MIN_HPF_FC_RX_CHAN3

```
#define ADI_ADAR690x_MIN_HPF_FC_RX_CHAN3 (0UL)
```

See Table 4 in [1].

6.1.2.151 ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN0 (0UL)
```

See Table 4 in [1].

6.1.2.152 ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN1 (0UL)
```

See Table 4 in [1].

6.1.2.153 ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN2 (0UL)
```

See Table 4 in [1].

6.1.2.154 ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_MIN_HPF_GAIN_RX_CHAN3 (0UL)
```

See Table 4 in [1].

6.1.2.155 ADI_ADAR690x_MIN_PAT_SEL_TEST_TONE

```
#define ADI_ADAR690x_MIN_PAT_SEL_TEST_TONE (0UL)
```

See Table 4 in [1].

6.1.2.156 ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN0

```
#define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN0 (0UL)
```

See Table 4 in [1].

6.1.2.157 ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN1

```
#define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN1 (0UL)
```

See Table 4 in [1].

6.1.2.158 ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN2

```
#define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN2 (0UL)
```

See Table 4 in [1].

6.1.2.159 ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN3

```
#define ADI_ADAR690x_MIN_PGA_GAIN_RX_CHAN3 (0UL)
```

See Table 4 in [1].

6.1.2.160 ADI_ADAR690x_MIN_POWER_DOWN_AUTO_US

```
#define ADI_ADAR690x_MIN_POWER_DOWN_AUTO_US (0UL)
```

See Table 4 in [1].

6.1.2.161 ADI_ADAR690x_MIN_PWR_TX0

```
#define ADI_ADAR690x_MIN_PWR_TX0 (-10L)
```

See Table 4 in [1].

6.1.2.162 ADI_ADAR690x_MIN_PWR_TX1

```
#define ADI_ADAR690x_MIN_PWR_TX1 (-10L)
```

See Table 4 in [1].

6.1.2.163 ADI_ADAR690x_MIN_PWR_TX2

```
#define ADI_ADAR690x_MIN_PWR_TX2 (-10L)
```

See Table 4 in [1].

6.1.2.164 ADI_ADAR690x_MIN_RAMP_BW_MHZ

```
#define ADI_ADAR690x_MIN_RAMP_BW_MHZ (-5000.0F)
```

See Table 4 in [1].

6.1.2.165 ADI_ADAR690x_MIN_RAMP_START_FREQ_HZ

```
#define ADI_ADAR690x_MIN_RAMP_START_FREQ_HZ (76000000000ULL)
```

See Table 4 in [1].

6.1.2.166 ADI_ADAR690x_MIN_REF_FREQ_HZ

```
#define ADI_ADAR690x_MIN_REF_FREQ_HZ (40000000UL)
```

See Table 4 in [1].

6.1.2.167 ADI_ADAR690x_MIN_RFPLL_LOOP_BW_HZ

```
#define ADI_ADAR690x_MIN_RFPLL_LOOP_BW_HZ (100000UL)
```

See Table 4 in [1].

6.1.2.168 ADI_ADAR690x_MIN_SYS_CAL_ENABLE

```
#define ADI_ADAR690x_MIN_SYS_CAL_ENABLE (0UL)
```

See Table 4 in [1].

6.1.2.169 ADI_ADAR690x_MIN_TIMING_COMP_EN

```
#define ADI_ADAR690x_MIN_TIMING_COMP_EN (0UL)
```

See Table 4 in [1].

6.1.2.170 ADI_ADAR690x_OFF_FW_DCCM_LOAD_ADDR

```
#define ADI_ADAR690x_OFF_FW_DCCM_LOAD_ADDR 0x1FDF8
```

Offset of DCCM file load address in ICCM file.

6.1.2.171 ADI_ADAR690x_OFF_FW_ICCM_LOAD_ADDR

```
#define ADI_ADAR690x_OFF_FW_ICCM_LOAD_ADDR 0x1FDF4
```

Offset of ICCM file load address in ICCM file.

6.1.2.172 ADI_ADAR690x_OFF_FW_VERSION

```
#define ADI_ADAR690x_OFF_FW_VERSION 0x1FDFC
```

Offset of firmware version in ICCM file.

6.1.2.173 ADI_ADAR690x_STS_ADCPLL_FREQ_HZ

```
#define ADI_ADAR690x_STS_ADCPLL_FREQ_HZ (ADI_ADAR690x_STS_BASE+0x4B8)
```

See Table 5 in [1].

6.1.2.174 ADI_ADAR690x_STS_AVDD_09P_PWR_SUP_VOLT

```
#define ADI_ADAR690x_STS_AVDD_09P_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x24C)
```

See Table 5 in [1].

6.1.2.175 ADI_ADAR690x_STS_AVDD_1P8_PWR_SUP_VOLT

```
#define ADI_ADAR690x_STS_AVDD_1P8_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x264)
```

See Table 5 in [1].

6.1.2.176 ADI_ADAR690x_STS_BASE

```
#define ADI_ADAR690x_STS_BASE 0x80004800
```

See **STATUS TABLE** [1].

6.1.2.177 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP0

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP0 (ADI_ADAR690x_STS_BASE+0x3FC)
```

See Table 5 in [1].

6.1.2.178 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP1

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP1 (ADI_ADAR690x_STS_BASE+0x400)
```

See Table 5 in [1].

6.1.2.179 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP10

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP10 (ADI_ADAR690x_STS_BASE+0x424)
```

See Table 5 in [1].

6.1.2.180 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP11

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP11 (ADI_ADAR690x_STS_BASE+0x428)
```

See Table 5 in [1].

6.1.2.181 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP12

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP12 (ADI_ADAR690x_STS_BASE+0x42C)
```

See Table 5 in [1].

6.1.2.182 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP13

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP13 (ADI_ADAR690x_STS_BASE+0x430)
```

See Table 5 in [1].

6.1.2.183 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP14

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP14 (ADI_ADAR690x_STS_BASE+0x434)
```

See Table 5 in [1].

6.1.2.184 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP15

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP15 (ADI_ADAR690x_STS_BASE+0x438)
```

See Table 5 in [1].

6.1.2.185 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP16

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP16 (ADI_ADAR690x_STS_BASE+0x43C)
```

See Table 5 in [1].

6.1.2.186 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP2

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP2 (ADI_ADAR690x_STS_BASE+0x404)
```

See Table 5 in [1].

6.1.2.187 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP3

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP3 (ADI_ADAR690x_STS_BASE+0x408)
```

See Table 5 in [1].

6.1.2.188 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP4

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP4 (ADI_ADAR690x_STS_BASE+0x40C)
```

See Table 5 in [1].

6.1.2.189 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP5

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP5 (ADI_ADAR690x_STS_BASE+0x410)
```

See Table 5 in [1].

6.1.2.190 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP6

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP6 (ADI_ADAR690x_STS_BASE+0x414)
```

See Table 5 in [1].

6.1.2.191 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP7

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP7 (ADI_ADAR690x_STS_BASE+0x418)
```

See Table 5 in [1].

6.1.2.192 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP8

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP8 (ADI_ADAR690x_STS_BASE+0x41C)
```

See Table 5 in [1].

6.1.2.193 ADI_ADAR690x_STS_COMPUTED_CRC_GROUP9

```
#define ADI_ADAR690x_STS_COMPUTED_CRC_GROUP9 (ADI_ADAR690x_STS_BASE+0x420)
```

See Table 5 in [1].

6.1.2.194 ADI_ADAR690x_STS_CRC_FAIL

```
#define ADI_ADAR690x_STS_CRC_FAIL (ADI_ADAR690x_STS_BASE+0x3D8)
```

See Table 5 in [1].

6.1.2.195 ADI_ADAR690x_STS_CRC_PASS

```
#define ADI_ADAR690x_STS_CRC_PASS (ADI_ADAR690x_STS_BASE+0x3D4)
```

See Table 5 in [1].

6.1.2.196 ADI_ADAR690x_STS_DMA_OFFSETS_LUT

```
#define ADI_ADAR690x_STS_DMA_OFFSETS_LUT (ADI_ADAR690x_STS_BASE+0xE1C)
```

See Table 5 in [1].

6.1.2.197 ADI_ADAR690x_STS_DVDD_09P_PWR_SUP_VOLT

```
#define ADI_ADAR690x_STS_DVDD_09P_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x258)
```

See Table 5 in [1].

6.1.2.198 ADI_ADAR690x_STS_LOCHAIN_TEMP

```
#define ADI_ADAR690x_STS_LOCHAIN_TEMP (ADI_ADAR690x_STS_BASE+0x1C)
```

See Table 5 in [1].

6.1.2.199 ADI_ADAR690x_STS_OP_PWR_TX0

```
#define ADI_ADAR690x_STS_OP_PWR_TX0 (ADI_ADAR690x_STS_BASE+0x230)
```

See Table 5 in [1].

6.1.2.200 ADI_ADAR690x_STS_OP_PWR_TX1

```
#define ADI_ADAR690x_STS_OP_PWR_TX1 (ADI_ADAR690x_STS_BASE+0x234)
```

See Table 5 in [1].

6.1.2.201 ADI_ADAR690x_STS_OP_PWR_TX2

```
#define ADI_ADAR690x_STS_OP_PWR_TX2 (ADI_ADAR690x_STS_BASE+0x238)
```

See Table 5 in [1].

6.1.2.202 ADI_ADAR690x_STS_RFPLL_FREQ_HZ

```
#define ADI_ADAR690x_STS_RFPLL_FREQ_HZ (ADI_ADAR690x_STS_BASE+0x4D0)
```

See Table 5 in [1].

6.1.2.203 ADI_ADAR690x_STS_RX0_TEMP

```
#define ADI_ADAR690x_STS_RX0_TEMP (ADI_ADAR690x_STS_BASE+0xC)
```

See Table 5 in [1].

6.1.2.204 ADI_ADAR690x_STS_RX1_TEMP

```
#define ADI_ADAR690x_STS_RX1_TEMP (ADI_ADAR690x_STS_BASE+0x10)
```

See Table 5 in [1].

6.1.2.205 ADI_ADAR690x_STS_RX2_TEMP

```
#define ADI_ADAR690x_STS_RX2_TEMP (ADI_ADAR690x_STS_BASE+0x14)
```

See Table 5 in [1].

6.1.2.206 ADI_ADAR690x_STS_RX3_TEMP

```
#define ADI_ADAR690x_STS_RX3_TEMP (ADI_ADAR690x_STS_BASE+0x18)
```

See Table 5 in [1].

6.1.2.207 ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_0

```
#define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_0 (ADI_ADAR690x_STS_BASE+0x214)
```

See Table 5 in [1].

6.1.2.208 ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_1

```
#define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_1 (ADI_ADAR690x_STS_BASE+0x218)
```

See Table 5 in [1].

6.1.2.209 ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_2

```
#define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_2 (ADI_ADAR690x_STS_BASE+0x21C)
```

See Table 5 in [1].

6.1.2.210 ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_3

```
#define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_3 (ADI_ADAR690x_STS_BASE+0x220)
```

See Table 5 in [1].

6.1.2.211 ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_4

```
#define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_4 (ADI_ADAR690x_STS_BASE+0x224)
```

See Table 5 in [1].

6.1.2.212 ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_5

```
#define ADI_ADAR690x_STS_RX_SIGCHAIN_MISMATCH_5 (ADI_ADAR690x_STS_BASE+0x228)
```

See Table 5 in [1].

6.1.2.213 ADI_ADAR690x_STS_TX0_TEMP

```
#define ADI_ADAR690x_STS_TX0_TEMP (ADI_ADAR690x_STS_BASE+0x0)
```

See Table 5 in [1].

6.1.2.214 ADI_ADAR690x_STS_TX1_TEMP

```
#define ADI_ADAR690x_STS_TX1_TEMP (ADI_ADAR690x_STS_BASE+0x4)
```

See Table 5 in [1].

6.1.2.215 ADI_ADAR690x_STS_TX2_TEMP

```
#define ADI_ADAR690x_STS_TX2_TEMP (ADI_ADAR690x_STS_BASE+0x8)
```

See Table 5 in [1].

6.1.2.216 ADI_ADAR690x_STS_TX_LOAD_T0_TX0

```
#define ADI_ADAR690x_STS_TX_LOAD_T0_TX0 (ADI_ADAR690x_STS_BASE+0x34)
```

See Table 5 in [1].

6.1.2.217 ADI_ADAR690x_STS_TX_LOAD_T0_TX1

```
#define ADI_ADAR690x_STS_TX_LOAD_T0_TX1 (ADI_ADAR690x_STS_BASE+0x38)
```

See Table 5 in [1].

6.1.2.218 ADI_ADAR690x_STS_TX_LOAD_T0_TX2

```
#define ADI_ADAR690x_STS_TX_LOAD_T0_TX2 (ADI_ADAR690x_STS_BASE+0x3C)
```

See Table 5 in [1].

6.1.2.219 ADI_ADAR690x_STS_VDD_DCO_PWR_SUP_VOLT

```
#define ADI_ADAR690x_STS_VDD_DCO_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x248)
```

See Table 5 in [1].

6.1.2.220 ADI_ADAR690x_STS_VDDIO_1P8_PWR_SUP_VOLT

```
#define ADI_ADAR690x_STS_VDDIO_1P8_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x274)
```

See Table 5 in [1].

6.1.2.221 ADI_ADAR690x_STS_VDDIO_PWR_SUP_VOLT

```
#define ADI_ADAR690x_STS_VDDIO_PWR_SUP_VOLT (ADI_ADAR690x_STS_BASE+0x27C)
```

See Table 5 in [1].

6.2 adi_dmdriver.h File Reference

Public C interface to the driver.

```
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include "adi_dmhal.h"
```

Data Structures

- struct adi_dm_ramp_profile_t

  Ramp Profile.
- struct adi_dm_tx_overlay_t

  Tx Overlay.
- struct adi_dm_ramp_config_t

  Ramp Configuration.
- struct adi_dm_burst_profile_t

  Burst Profile.
- struct adi_dm_power_up_t

  IN parameter to `adi_dm_PowerUp`
- struct adi_dm_lvds_setup_t

  IN parameter to `adi_dm_LvdsSetup`
- struct adi_dm_mipi_setup_t

  IN parameter to `adi_dm_MipiSetup`
- struct adi_dm_afe_setup_t

  IN parameter to `adi_dm_AfeSetup`
- struct adi_dm_tx_setup_t

  IN parameter to `adi_dm_TxSetup`
- struct adi_dm_ramp_shape_t

  High level description of ramp shape input to adi_dm_CalcRamp()
- struct adi_dm_actual_ramp_shape_t

  High level description of ramp shape output by adi_dm_CalcRamp()
- struct adi_dm_builtin_mimo_setup_t

  IN parameter to adi_dm_BuiltinMimoSetup()
- struct adi_dm_dma_ramp_setup_t

  IN parameter to adi_dm_DmaRampSetup()
- struct adi_dm_periodic_calibration_t

  IN parameter to `adi_dm_PeriodicCalibration`
- struct adi_dm_rfpll_reconfig_t

  IN parameter to adi_dm_RfpllReconfig()
- struct adi_dm_mask_faults_t

  IN parameter to `adi_dm_MaskFaults`
- struct adi_dm_temperature_t

  IN parameter to `adi_dm_TemperatureGet`
- struct adi_dm_power_detector_meas_task_t

  IN parameter to `adi_dm_PowerDetectorMeasTask`
- struct adi_dm_write_rfpll_period_t

  IN parameter to `adi_dm_WriteRfpllPeriod` OUT parameter to `adi_dm_CalcRfpllPeriod`
- struct adi_dm_calc_rfpll_period_t

  IN parameter to `adi_dm_CalcRfpllPeriod`
- struct adi_dm_tasklist_pool_setup_t

  IN parameter to `adi_dm_TasklistPoolSetup`

Enumerations

- enum adi_dm_err_t {
ADI_DM_SUCCESS = 0, ADI_DM_FAIL = -1, ADI_DM_TIMEDOUT = -2, ADI_DM_PARAMETER_ERROR = -3,
ADI_DM_FIRMWARE_ERROR = -4, ADI_DM_SPI_SCRATCHPAD_ERROR = -5, ADI_DM_MISC_SCRATCHPAD_ERROR = -6, ADI_DM_SPI_CRC_ERROR = -7,
ADI_DM_SPI_COUNTER_ERROR = -8, ADI_DM_CASCADED_TX_PWR_CHK_ERROR = -9 }

  Possible error codes.
- enum adi_dm_gpio_t {
ADI_DM_RESETB, **ADI_DM_GPIO0**, **ADI_DM_GPIO1**, **ADI_DM_FAULT0B**,
ADI_DM_FAULT1, **ADI_DM_TRIG**, **ADI_DM_PWDNB**, **ADI_DM_GPIO6**,
ADI_DM_GPIO7, **ADI_DM_GPIO8**, **ADI_DM_GPIO9**, **ADI_DM_GPIO10**,
ADI_DM_GPIO11, **ADI_DM_NUM_GPIO** }

  Names for GPIO pins.

Functions

Driver initialization

- void adi_dm_InitDriver (void)

  Initialize driver and host side, do not touch device.
- void adi_dm_FiniDriver (void)

  Release any resources used by driver.
- const char ∗ adi_dm_DriverVersion (void)

  Return a string denoting driver release or "unreleased".

Typed memory access

These are convenience functions for memory access.

They are not strictly required, but may avoid errors in customer code.

The following data types (other than uint32_t) are supported:

- *uint64_t*
- *float*
- *double*

- static adi_dm_err_t adi_dm_WriteU64 (adi_dm_num_t dm_num, uint32_t addr, uint64_t value)

  Write uint64_t to the DigiMMIC address space, checking for CRC errors.
- static adi_dm_err_t adi_dm_WriteF32 (adi_dm_num_t dm_num, uint32_t addr, float value)

  Write float to the DigiMMIC address space, checking for CRC errors.
- static adi_dm_err_t adi_dm_ReadF32 (adi_dm_num_t dm_num, uint32_t addr, float ∗value)

  Read float from the DigiMMIC address space, checking for CRC errors.
- static adi_dm_err_t adi_dm_ReadU64 (adi_dm_num_t dm_num, uint32_t addr, uint64_t ∗value)

  Read uint64_t value from the DigiMMIC address space, checking for CRC errors.
- static adi_dm_err_t adi_dm_ReadF64 (adi_dm_num_t dm_num, uint32_t addr, double ∗value)

  Read double from the DigiMMIC address space, checking for CRC errors.
- static adi_dm_err_t adi_dm_WriteF64 (adi_dm_num_t dm_num, uint32_t addr, double value)

  Write double to the DigiMMIC address space, checking for CRC errors.

Low-level

}

This is the low-level API. All functions take a `dm_num` parameter specifying which DigiMMIC to use. Valid values for this is the contiguous range from ADI_DM_MASTER (0) to (ADI_DM_NUM_DIGIMMIC - 1) inclusive.

- enum adi_dm_num_t {
  ADI_DM_MASTER = 0, ADI_DM_SLAVE1 = 1, ADI_DM_SLAVE2 = 2, ADI_DM_SLAVE3 = 3,
  ADI_DM_ALL_DIGIMMICS = 0xFF }
- adi_dm_num_t adi_dm_active_digimmics

  Number of devices actively controlled by the driver.
- uint32_t **adi_dm_sw_fault0_mask**
- uint32_t **adi_dm_sw_fault1_mask**
- uint32_t **adi_dm_sw_fault2_mask**
- uint32_t **adi_dm_sw_fault3_mask**
- uint32_t **adi_dm_fault_status0_mask**
- uint32_t **adi_dm_fault_status1_mask**
- uint32_t **adi_dm_fault_status2_mask**
- adi_dm_err_t adi_dm_Write (adi_dm_num_t dm_num, uint32_t addr, uint32_t value)

  Write a word to the DigiMMIC address space, checking for CRC error.
- adi_dm_err_t adi_dm_Read (adi_dm_num_t dm_num, uint32_t addr, uint32_t ∗value)

  Read a word from the DigiMMIC address space, checking for CRC error.
- adi_dm_err_t adi_dm_RMW (adi_dm_num_t dm_num, uint32_t addr, uint32_t mask, uint32_t bits)

  Read, modify, write a word in the DigiMMIC address space, checking for CRC error.
- adi_dm_err_t adi_dm_BlockWrite (adi_dm_num_t dm_num, uint32_t addr, size_t sz, const void ∗data)

  Write a memory region in the DigiMMIC address space, checking for CRC error.
- adi_dm_err_t adi_dm_BlockRead (adi_dm_num_t dm_num, uint32_t addr, size_t sz, void ∗data)

  Read a memory region in the DigiMMIC address space, checking for CRC error.
- adi_dm_err_t adi_dm_BlockCallFW (adi_dm_num_t dm_num, size_t sz, const void ∗data)

  Execute a firmware task.
- static adi_dm_err_t adi_dm_CallFW (adi_dm_num_t dm_num, uint32_t cmd)

  Execute a firmware task.
- static adi_dm_err_t adi_dm_CallFW2 (adi_dm_num_t dm_num, uint32_t cmd, uint32_t p1, uint32_t p2)

  Execute a firmware task.
- static adi_dm_err_t **adi_dm_CallFWU64** (adi_dm_num_t dm_num, uint32_t cmd, uint64_t p)
- adi_dm_err_t adi_dm_PinMux (adi_dm_num_t dm_num, adi_dm_gpio_t pin, uint8_t fer, uint8_t mux)

  Control function of an IO pad.

High-level

This is the high-level API.

- #define **ADI_DM_NUM_TX** 3 /∗∗ Number of Tx channels on each part. ∗/
- #define **ADI_DM_NUM_RX** 4 /∗∗ Number of Rx channels on each part. ∗/
- #define ADI_DM_REF_CLK_DIV_2 0U

  Set CLKOUT pin to ref_clk / 2.
- #define ADI_DM_REF_CLK_DIV_4 1U

  Set CLKOUT pin to ref_clk / 4.
- #define ADI_DM_REF_CLK_DIV 2U

Set CLKOUT pin to ref_clk.
- #define ADI_DM_TIMING_COMP_DISABLED 0U

  Disabled.
- #define ADI_DM_TIMING_COMP_GPIO7_GPIO9 1U

  GPIO7 and GPIO9 connected.
- #define ADI_DM_TIMING_COMP_GPIO7_ONLY 2U

  Only GPIO7 connected.
- #define ADI_DM_TIMING_COMP_GPIO9_ONLY 3U

  Only GPIO9 connected.
- #define ADI_DM_OUTPUT_BITWIDTH_16 0U

  Set filter_output_bitwidth to 16.
- #define ADI_DM_OUTPUT_BITWIDTH_14 1U

  Set filter_output_bitwidth to 14.
- #define ADI_DM_OUTPUT_BITWIDTH_12 2U

  Set filter_output_bitwidth to 12.
- #define ADI_DM_ADCCLK_DIV_6 1U

  Set lvds_clk to ADCCLK / 6.
- #define ADI_DM_ADCCLK_DIV_1_5 2U

  Set lvds_clk to ADCCLK / 1.5.
- #define ADI_DM_ADCCLK_DIV_2 3U

  Set lvds_clk to ADCCLK / 2.
- #define ADI_DM_ADCCLK_DIV_3 4U

  Set lvds_clk to ADCCLK / 3.
- #define ADI_DM_ADCCLK_DIV_4 5U

  Set lvds_clk to ADCCLK / 4.
- #define ADI_DM_ADCCLK_DIV_8 6U

  Set lvds_clk to ADCCLK / 8.
- #define ADI_DM_MIPI_CLK_1200MHZ 0UL

  Set mipi_clk to 1.2 GHz.
- #define ADI_DM_MIPI_CLK_1000MHZ 1UL

  Set mipi_clk to 1.0 GHz.
- #define ADI_DM_MIPI_CLK_800MHZ 2UL

  Set mipi_clk to 800 MHz.
- #define ADI_DM_MIPI_CLK_80MHZ 3UL

  Set mipi_clk to 80 MHz.
- #define ADI_DM_MIPI_CLK_650MHZ 4UL

  Set mipi_clk to 650 MHz.
- #define ADI_DM_MIPI_CLK_NUM 5UL

  Used to check adi_dm_mipi_clk_t range.
- #define ADI_DM_1_MIPI_LANE 0UL

  Enable 1 lane.
- #define ADI_DM_2_MIPI_LANES 1UL

  Enable 2 lanes.
- #define ADI_DM_4_MIPI_LANES 2UL

  Enable 4 lanes.
- #define ADI_DM_YUV422_8B 0x1EUL

  YUV422_8B.
- #define ADI_DM_RGB444 0x20UL

  RGB444.
- #define ADI_DM_RGB555 0x21UL

  RGB555.

- #define ADI_DM_RGB565 0x22UL

  RGB565.
- #define ADI_DM_RAW6 0x28UL

  RAW6.
- #define ADI_DM_RAW7 0x29UL

  RAW7.
- #define ADI_DM_RAW8 0x2AUL

  RAW8.
- #define ADI_DM_RAW12 0x2CUL

  RAW12.
- #define ADI_DM_RAW14 0x2DUL

  RAW14.
- #define ADI_DM_BITP_MIMO_SEQ_VAL_PAT0 0

  Bit position of active antennas during sequence step 0.
- #define ADI_DM_BITM_MIMO_SEQ_VAL_PAT0 0x7

  Bit mask of active antennas during sequence step 0.
- #define ADI_DM_BITP_MIMO_SEQ_VAL_PHASE0 3

  Bit position of phase during sequence step 0.
- #define ADI_DM_BITM_MIMO_SEQ_VAL_PHASE0 0x8

  Bit mask of phase during sequence step 0.
- #define ADI_DM_BITP_MIMO_SEQ_VAL_PAT1 4

  Bit position of active antennas during sequence step 1.
- #define ADI_DM_BITM_MIMO_SEQ_VAL_PAT1 0x70

  Bit mask of active antennas during sequence step 1.
- #define ADI_DM_BITP_MIMO_SEQ_VAL_PHASE1 7

  Bit position of phase during sequence step 1.
- #define ADI_DM_BITM_MIMO_SEQ_VAL_PHASE1 0x80

  Bit mask of phase during sequence step 1.
- #define ADI_DM_BITP_MIMO_SEQ_VAL_PAT2 8

  Bit position of active antennas during sequence step 2.
- #define ADI_DM_BITM_MIMO_SEQ_VAL_PAT2 0x700

  Bit mask of active antennas during sequence step 2.
- #define ADI_DM_BITP_MIMO_SEQ_VAL_PHASE2 11

  Bit position of phase during sequence step 2.
- #define ADI_DM_BITM_MIMO_SEQ_VAL_PHASE2 0x800

  Bit mask of phase during sequence step 2.
- enum adi_dm_hpf_fc_t {
  ADI_DM_HPF_FC_125_KHZ = 0, **ADI_DM_HPF_FC_250_KHZ** = 1, **ADI_DM_HPF_FC_500_KHZ** = 2, **A↵
  DI_DM_HPF_FC_1_MHZ** = 3,
  ADI_DM_HPF_FC_2_MHZ = 4, **ADI_DM_HPF_FC_4_MHZ** = 5, **ADI_DM_HPF_FC_8_MHZ** = 6 }

  Enumeration for high pass filter (HPF) corner frequency.
- enum adi_dm_hpf_gain_t { **ADI_DM_HPF_GAIN_6_DB** = 0, **ADI_DM_HPF_GAIN_12_DB** = 1, **ADI_DM↵
  _HPF_GAIN_18_DB** = 2 }

  Enumeration for high pass filter (HPF) gain.
- enum adi_dm_pga_gain_t {
  ADI_DM_PGA_GAIN_0_DB = 0, **ADI_DM_PGA_GAIN_6_DB** = 1, **ADI_DM_PGA_GAIN_12_DB** = 2, **AD↵
  I_DM_PGA_GAIN_18_DB** = 3,
  ADI_DM_PGA_GAIN_24_DB = 4 }

  Enumeration for programmable gain amplifier (PGA) in RX channel.
- enum adi_dm_adc_gain_t { **ADI_DM_ADC_GAIN_0_DB** = 0, **ADI_DM_ADC_GAIN_3_DB** = 2 }

  Enumeration for programmable gain amplifier (PGA) in RX channel.

- enum adi_dm_pga_mux_t { **ADI_DM_PGA_MUX_HPF_OUTPUT** = 0, **ADI_DM_PGA_MUX_HPF_INPUT** = 1, **ADI_DM_PGA_MUX_DISCONNECTED** = 2 }

  Enumeration for programmable gain amplifier (PGA) input.
- typedef uint8_t adi_dm_clkoutctrl_t

  Effective enumeration for CLKOUTCTRL clock selection.
- typedef uint8_t adi_dm_timing_comp_setting_t

  Values for ADI_ADAR690x_CFG_TIMING_COMP_EN setting for BIST56.
- typedef uint8_t adi_dm_lvds_clk_t

  Effective enumeration for LVDS clock selection.
- typedef uint32_t adi_dm_mipi_clk_t

  Effective enumeration for MIPI clock selection.
- typedef uint32_t adi_dm_num_mipi_lanes_t

  Effective enumeration for number of MIPI lanes to enable.
- typedef uint32_t adi_dm_mipi_data_type_t

  Effective enumeration for payload MIPI data type.
- adi_dm_err_t adi_dm_PowerUp (const adi_dm_power_up_t ∗p)

  Power up device.
- adi_dm_err_t adi_dm_PowerDown (void)

  Power down device.
- adi_dm_err_t adi_dm_LvdsSetup (const adi_dm_lvds_setup_t ∗p)

  Dataport configuration for LVDS parts.
- adi_dm_err_t adi_dm_MipiSetup (const adi_dm_mipi_setup_t ∗p)

  Dataport configuration for MIPI parts.
- adi_dm_err_t adi_dm_AfeSetup (const adi_dm_afe_setup_t ∗p)

  Analog front end (AFE) configuration.
- adi_dm_err_t adi_dm_TxSetup (const adi_dm_tx_setup_t ∗p)

  Transmitter path (Tx) configuration.
- adi_dm_err_t adi_dm_CalcRamp (const adi_dm_ramp_shape_t ∗in, adi_dm_ramp_profile_t ∗ramp_profile, adi_dm_ramp_config_t ∗ramp_config, adi_dm_actual_ramp_shape_t ∗actual)

  Calculate ramp profile from high level parameters.
- adi_dm_err_t adi_dm_BuiltinMimoSetup (const adi_dm_builtin_mimo_setup_t ∗p)

  Register writes for ramp generation using built-in MIMO mode.
- adi_dm_err_t adi_dm_BurstProfileSetup (const adi_dm_burst_profile_t ∗p, uint32_t ∗dma_mem_limit, uint32_t ∗bpid)

  Memory writes for ramp generation using DMA.
- adi_dm_err_t adi_dm_DmaRampSetup (const adi_dm_dma_ramp_setup_t ∗p)

  Register writes for ramp generation using DMA.
- adi_dm_err_t adi_dm_SelectBurstProfile (uint32_t bpid)

  Select the burst profile to use for subsequent bursts.
- adi_dm_err_t adi_dm_Trigger (void)

  Issue software trigger for a single burst.
- adi_dm_err_t adi_dm_PeriodicCalibration (const adi_dm_periodic_calibration_t ∗p)

  Perform periodic firmware (re-)calibration as recommended by [1].
- adi_dm_err_t adi_dm_RfpllReconfig (const adi_dm_rfpll_reconfig_t ∗p)

  Reconfigure RFPLL.
- adi_dm_err_t adi_dm_RfpllLock (uint64_t ramp_start_freq_hz)

  Set the RFPLL lock frequency.
- adi_dm_err_t adi_dm_LockConfig (void)

  Setup for checks that configuration does not change by accident.
- adi_dm_err_t adi_dm_UnlockConfig (void)

  Tear down for checks that configuration does not change by accident.

Miscellaneous

These functions provide functionality which does not quite fit into the other categories.

- #define ADI_DM_SYSCAL_TX_SIZE 24

  Dimension for BIST54 t0 measurements.
- #define ADI_DM_SYSCAL_RX_SIZE 24

  Dimension for BIST15 t0 measurements.
- uint32_t adi_dm_Crc32ADI (uint8_t ∗p, int n, int step)

  Calculate a CRC/checksum.
- adi_dm_err_t adi_dm_MaskFaults (adi_dm_mask_faults_t ∗p)

  Ignore faults triggered by hardware or software errors.
- adi_dm_err_t adi_dm_TemperatureGet (adi_dm_temperature_t ∗out)

  Get temperature of parts.
- adi_dm_err_t adi_dm_SetLvdsTestPattern (uint16_t pat[ADI_DM_NUM_DIGIMMIC][ADI_DM_NUM_RX])

  Set test mode on LVDS devices.
- adi_dm_err_t adi_dm_ClearLvdsTestPattern (void)

  Unset test mode on LVDS devices.
- adi_dm_err_t adi_dm_ManualSleep (void)

  Enter low power state suitable for sleep between bursts.
- adi_dm_err_t adi_dm_ManualWake (void)

  Exit the low power state suitable for sleep between bursts.
- adi_dm_err_t adi_dm_PowerDetectorMeasTask (adi_dm_power_detector_meas_task_t ∗p)

  Configures Power Detector Measurement tasks and sets measurement for next burst.
- adi_dm_err_t adi_dm_CalcPwrDetCfg (adi_dm_burst_profile_t ∗p, adi_dm_power_detector_meas_task_t ∗out)

  Calculate power detector configuration for a burst, required by BIST15: Rx chain diagnostic check, BIST18: LO chain output monitor, BIST19: Tx output transmit power check, BIST44: Power detector rationality check, BIST53: Tx isolation monitor check, and BIST54: Tx output load monitor check.
- adi_dm_err_t adi_dm_WriteRfpllPeriod (adi_dm_write_rfpll_period_t ∗p)

  Calculate expected RFPLL period count for a burst, required by BIST103c 'RFPLL period check'.
- adi_dm_err_t adi_dm_CalcRfpllPeriod (adi_dm_calc_rfpll_period_t ∗p, adi_dm_write_rfpll_period_t ∗out)

  Calculate expected RFPLL period count for a burst, required by BIST103c 'RFPLL period check'.
- adi_dm_err_t adi_dm_SetSysCal (bool enable_sys_cal)

  Set system calibration mode.
- adi_dm_err_t adi_dm_WriteSysCalRx (const uint8_t in[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_RX_SIZE])

  Write RX System Calibration values.
- adi_dm_err_t adi_dm_WriteSysCalTx (const uint8_t in[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_TX_SIZE])

  Write TX System Calibration values.
- adi_dm_err_t adi_dm_ReadSysCalRx (uint8_t out[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_RX_SIZE])

  Read RX System Calibration values.
- adi_dm_err_t adi_dm_ReadSysCalTx (uint8_t out[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_TX_SIZE])

  Read TX System Calibration values.
- adi_dm_err_t adi_dm_PwndnInterrupt (void)

  Send PWDNb interrupt.
- adi_dm_err_t adi_dm_Tasklist (adi_dm_num_t dm_num, uint8_t listno)

  Execute a tasklist.
- adi_dm_err_t adi_dm_TasklistNoBlock (adi_dm_num_t dm_num, uint8_t listno)

  Execute a tasklist without blocking.
- adi_dm_err_t adi_dm_TasklistPoolSetup (const adi_dm_tasklist_pool_setup_t ∗p, uint32_t ∗dma_mem_limit)

  Create a tasklist pool.

6.2.1 Detailed Description

Public C interface to the driver.

6.2.2 Macro Definition Documentation

6.2.2.1 ADI_DM_1_MIPI_LANE

```
#define ADI_DM_1_MIPI_LANE 0UL
```

Enable 1 lane.

See adi_dm_num_mipi_lanes_t.

6.2.2.2 ADI_DM_2_MIPI_LANES

```
#define ADI_DM_2_MIPI_LANES 1UL
```

Enable 2 lanes.

See adi_dm_num_mipi_lanes_t.

6.2.2.3 ADI_DM_4_MIPI_LANES

```
#define ADI_DM_4_MIPI_LANES 2UL
```

Enable 4 lanes.

See adi_dm_num_mipi_lanes_t.

6.2.2.4 ADI_DM_ADCCLK_DIV_1_5

```
#define ADI_DM_ADCCLK_DIV_1_5 2U
```

Set lvds_clk to ADCCLK / 1.5.

See also

adi_dm_lvds_clk_t.

6.2.2.5 ADI_DM_ADCCLK_DIV_2

```
#define ADI_DM_ADCCLK_DIV_2 3U
```

Set lvds_clk to ADCCLK / 2.

See also

adi_dm_lvds_clk_t.

6.2.2.6 ADI_DM_ADCCLK_DIV_3

```
#define ADI_DM_ADCCLK_DIV_3 4U
```

Set lvds_clk to ADCCLK / 3.

See also

adi_dm_lvds_clk_t.

6.2.2.7 ADI_DM_ADCCLK_DIV_4

```
#define ADI_DM_ADCCLK_DIV_4 5U
```

Set lvds_clk to ADCCLK / 4.

See also

adi_dm_lvds_clk_t.

6.2.2.8 ADI_DM_ADCCLK_DIV_6

```
#define ADI_DM_ADCCLK_DIV_6 1U
```

Set lvds_clk to ADCCLK / 6.

See also

adi_dm_lvds_clk_t.

6.2.2.9 ADI_DM_ADCCLK_DIV_8

```
#define ADI_DM_ADCCLK_DIV_8 6U
```

Set lvds_clk to ADCCLK / 8.

See also

adi_dm_lvds_clk_t.

6.2.2.10 ADI_DM_MIPI_CLK_1000MHZ

```
#define ADI_DM_MIPI_CLK_1000MHZ 1UL
```

Set mipi_clk to 1.0 GHz.

See adi_dm_mipi_clk_t.

6.2.2.11 ADI_DM_MIPI_CLK_1200MHZ

```
#define ADI_DM_MIPI_CLK_1200MHZ 0UL
```

Set mipi_clk to 1.2 GHz.

See adi_dm_mipi_clk_t.

6.2.2.12 ADI_DM_MIPI_CLK_650MHZ

```
#define ADI_DM_MIPI_CLK_650MHZ 4UL
```

Set mipi_clk to 650 MHz.

See adi_dm_mipi_clk_t.

6.2.2.13 ADI_DM_MIPI_CLK_800MHZ

```
#define ADI_DM_MIPI_CLK_800MHZ 2UL
```

Set mipi_clk to 800 MHz.

See adi_dm_mipi_clk_t.

6.2.2.14 ADI_DM_MIPI_CLK_80MHZ

```
#define ADI_DM_MIPI_CLK_80MHZ 3UL
```

Set mipi_clk to 80 MHz.

See adi_dm_mipi_clk_t.

6.2.2.15 ADI_DM_MIPI_CLK_NUM

```
#define ADI_DM_MIPI_CLK_NUM 5UL
```

Used to check adi_dm_mipi_clk_t range.

6.2.2.16 ADI_DM_OUTPUT_BITWIDTH_12

```
#define ADI_DM_OUTPUT_BITWIDTH_12 2U
```

Set filter_output_bitwidth to 12.

See also

adi_dm_lvds_setup_t and adi_dm_mipi_setup_t.

6.2.2.17 ADI_DM_OUTPUT_BITWIDTH_14

```
#define ADI_DM_OUTPUT_BITWIDTH_14 1U
```

Set filter_output_bitwidth to 14.

See also

adi_dm_lvds_setup_t and adi_dm_mipi_setup_t.

6.2.2.18 ADI_DM_OUTPUT_BITWIDTH_16

```
#define ADI_DM_OUTPUT_BITWIDTH_16 0U
```

Set filter_output_bitwidth to 16.

See also

adi_dm_lvds_setup_t and adi_dm_mipi_setup_t.

6.2.2.19 ADI_DM_RAW12

```
#define ADI_DM_RAW12 0x2CUL
```

RAW12.

See adi_dm_mipi_data_type_t.

6.2.2.20 ADI_DM_RAW14

```
#define ADI_DM_RAW14 0x2DUL
```

RAW14.

See adi_dm_mipi_data_type_t.

6.2.2.21 ADI_DM_RAW6

```
#define ADI_DM_RAW6 0x28UL
```

RAW6.

See adi_dm_mipi_data_type_t.

6.2.2.22 ADI_DM_RAW7

```
#define ADI_DM_RAW7 0x29UL
```

RAW7.

See adi_dm_mipi_data_type_t.

6.2.2.23 ADI_DM_RAW8

```
#define ADI_DM_RAW8 0x2AUL
```

RAW8.

See adi_dm_mipi_data_type_t.

6.2.2.24 ADI_DM_REF_CLK_DIV

```
#define ADI_DM_REF_CLK_DIV 2U
```

Set CLKOUT pin to ref_clk.

See also

adi_dm_clkoutctrl_t.

6.2.2.25 ADI_DM_REF_CLK_DIV_2

```
#define ADI_DM_REF_CLK_DIV_2 0U
```

Set CLKOUT pin to ref_clk / 2.

See also

adi_dm_clkoutctrl_t.

6.2.2.26 ADI_DM_REF_CLK_DIV_4

```
#define ADI_DM_REF_CLK_DIV_4 1U
```

Set CLKOUT pin to ref_clk / 4.

See also

adi_dm_clkoutctrl_t.

6.2.2.27 ADI_DM_RGB444

```
#define ADI_DM_RGB444 0x20UL
```

RGB444.

See adi_dm_mipi_data_type_t.

6.2.2.28 ADI_DM_RGB555

```
#define ADI_DM_RGB555 0x21UL
```

RGB555.

See adi_dm_mipi_data_type_t.

6.2.2.29 ADI_DM_RGB565

```
#define ADI_DM_RGB565 0x22UL
```

RGB565.

See adi_dm_mipi_data_type_t.

6.2.2.30 ADI_DM_SYSCAL_RX_SIZE

```
#define ADI_DM_SYSCAL_RX_SIZE 24
```

Dimension for BIST15 t0 measurements.

See also

adi_dm_power_up_t0_data_t

6.2.2.31 ADI_DM_SYSCAL_TX_SIZE

```
#define ADI_DM_SYSCAL_TX_SIZE 24
```

Dimension for BIST54 t0 measurements.

See also

adi_dm_power_up_t0_data_t

6.2.2.32 ADI_DM_TIMING_COMP_DISABLED

```
#define ADI_DM_TIMING_COMP_DISABLED 0U
```

Disabled.

See also

adi_dm_timing_comp_setting_t.

6.2.2.33 ADI_DM_TIMING_COMP_GPIO7_GPIO9

```
#define ADI_DM_TIMING_COMP_GPIO7_GPIO9 1U
```

GPIO7 and GPIO9 connected.

See also

adi_dm_timing_comp_setting_t.

6.2.2.34 ADI_DM_TIMING_COMP_GPIO7_ONLY

```
#define ADI_DM_TIMING_COMP_GPIO7_ONLY 2U
```

Only GPIO7 connected.

See also

adi_dm_timing_comp_setting_t.

6.2.2.35 ADI_DM_TIMING_COMP_GPIO9_ONLY

```
#define ADI_DM_TIMING_COMP_GPIO9_ONLY 3U
```

Only GPIO9 connected.

See also

adi_dm_timing_comp_setting_t.

6.2.2.36 ADI_DM_YUV422_8B

```
#define ADI_DM_YUV422_8B 0x1EUL
```

YUV422_8B.

See adi_dm_mipi_data_type_t.

6.2.3 Typedef Documentation

6.2.3.1 adi_dm_clkoutctrl_t

```
typedef uint8_t adi_dm_clkoutctrl_t
```

Effective enumeration for CLKOUTCTRL clock selection.

Value must be one of the following.

- ADI_DM_REF_CLK_DIV_2
- ADI_DM_REF_CLK_DIV_4
- ADI_DM_REF_CLK_DIV

6.2.3.2 adi_dm_lvds_clk_t

```
typedef uint8_t adi_dm_lvds_clk_t
```

Effective enumeration for LVDS clock selection.

Value must be one of the following.

- ADI_DM_ADCCLK_DIV_1_5
- ADI_DM_ADCCLK_DIV_2
- ADI_DM_ADCCLK_DIV_3
- ADI_DM_ADCCLK_DIV_4
- ADI_DM_ADCCLK_DIV_6
- ADI_DM_ADCCLK_DIV_8

6.2.3.3 adi_dm_mipi_clk_t

```
typedef uint32_t adi_dm_mipi_clk_t
```

Effective enumeration for MIPI clock selection.

Value must be one of the following.

- ADI_DM_MIPI_CLK_80MHZ
- ADI_DM_MIPI_CLK_650MHZ
- ADI_DM_MIPI_CLK_800MHZ
- ADI_DM_MIPI_CLK_1000MHZ
- ADI_DM_MIPI_CLK_1200MHZ

6.2.3.4 adi_dm_mipi_data_type_t

```
typedef uint32_t adi_dm_mipi_data_type_t
```

Effective enumeration for payload MIPI data type.

Value must be one of the following.

- ADI_DM_YUV422_8B
- ADI_DM_RGB444
- ADI_DM_RGB555
- ADI_DM_RGB565
- ADI_DM_RAW6
- ADI_DM_RAW7
- ADI_DM_RAW8
- ADI_DM_RAW12
- ADI_DM_RAW14

6.2.3.5 adi_dm_num_mipi_lanes_t

```
typedef uint32_t adi_dm_num_mipi_lanes_t
```

Effective enumeration for number of MIPI lanes to enable.

Value must be one of the following.

- ADI_DM_4_MIPI_LANES
- ADI_DM_2_MIPI_LANES
- ADI_DM_1_MIPI_LANE

6.2.3.6 adi_dm_timing_comp_setting_t

```
typedef uint8_t adi_dm_timing_comp_setting_t
```

Values for ADI_ADAR690x_CFG_TIMING_COMP_EN setting for BIST56.

This BIST ensures ramp timings on cascaded devices are consistent. The check compares STAT1 monitor input on GPIO7 has the same timing as local ramp. The input some from STAT0 signal output on GPIO9 of the neighbour device. The recommended board layout connects GPIO9 with GPIO7 in a chain, leaving one device with GPIO7 unconnected and another, at the other end of the chain, with GPIO9 unconnected. These values tell the driver how the GPIO7 and GPIO9 pins of each device are connected.

Value must be one of the following.

- ADI_DM_TIMING_COMP_DISABLED
- ADI_DM_TIMING_COMP_GPIO7_GPIO9
- ADI_DM_TIMING_COMP_GPIO7_ONLY
- ADI_DM_TIMING_COMP_GPIO9_ONLY

6.2.4 Enumeration Type Documentation

6.2.4.1 adi_dm_err_t

```
enum adi_dm_err_t
```

Possible error codes.

Enumerator

ADI_DM_SUCCESS	ok
ADI_DM_FAIL	Generic failure code.
ADI_DM_TIMEDOUT	Operation took too long.
ADI_DM_PARAMETER_ERROR	Parameter validation failed.
ADI_DM_FIRMWARE_ERROR	Firmware signalled an error condition.
ADI_DM_SPI_SCRATCHPAD_ERROR	A test write to SPI failed on power up. Often indicates

6.2.4.2 adi_dm_num_t

```
enum adi_dm_num_t
```

Enumerator

ADI_DM_MASTER	DigiMMIC number for the master DigiMMIC in a cascade.
ADI_DM_SLAVE1	First slave in a cascade.
ADI_DM_SLAVE2	Second slave in a cascade.
ADI_DM_SLAVE3	Third slave in a cascade.
ADI_DM_ALL_DIGIMMICS	Pseudo-DigiMMIC number to address *all* DigiMMICs, master and slaves, in a cascade.

6.2.5 Function Documentation

6.2.5.1 adi_dm_AfeSetup()

```
adi_dm_err_t adi_dm_AfeSetup (
             const adi_dm_afe_setup_t * p )
```

Analog front end (AFE) configuration.

Parameters

in	*p*	parameters. See adi_dm_afe_setup_t.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.2 adi_dm_BlockCallFW()

```
adi_dm_err_t adi_dm_BlockCallFW (
             adi_dm_num_t dm_num,
             size_t sz,
             const void * data )
```

Execute a firmware task.

With arbitrary parameters.

Parameters

in	*dm_num*	The DigiMMIC on which to run the task. If `dm_num` is ADI_DM_ALL_DIGIMMICS, task is executed on all DigiMMICs.
in	*sz*	Length of data in bytes.
in	*data*	The firmware command and parameters. Should be of a length that is written with one SPI frame.

Returns

ADI_DM_SUCCESS for success, an error code for errors.

Hardware faults can be ignored by defining adi_dm_fault_status0_mask, adi_dm_fault_status1_mask and adi_↩ dm_fault_status2_mask in the user application.

6.2.5.3 adi_dm_BlockRead()

```
adi_dm_err_t adi_dm_BlockRead (
            adi_dm_num_t dm_num,
            uint32_t addr,
            size_t sz,
            void * data )
```

Read a memory region in the DigiMMIC address space, checking for CRC error.

Parameters

in	*dm_num*	DigiMMIC to read from. `dm_num` may not be ADI_DM_ALL_DIGIMMICS.
in	*addr*	Address to read fromin the DigiMMIC address space.
in	*sz*	Size of block to read from the DigiMMIC. Must be a multiple of 4 bytes.
in	*data*	Block of data read from the DigiMMIC.

Returns

ADI_DM_SUCCESS for success, an error code for errors including ADI_DM_SPI_CRC_ERROR for CRC error.

6.2.5.4 adi_dm_BlockWrite()

```
adi_dm_err_t adi_dm_BlockWrite (
            adi_dm_num_t dm_num,
            uint32_t addr,
            size_t sz,
            const void * data )
```

Write a memory region in the DigiMMIC address space, checking for CRC error.

Parameters

in	*dm_num*	DigiMMIC to write to. If `dm_num` is ADI_DM_ALL_DIGIMMICS, a write is broadcast to all DigiMMICs.
in	*addr*	Address to write to in the DigiMMIC address space.
in	*sz*	Size of block written the DigiMMICs. Must be a multiple of 4 bytes.
in	*data*	Block of data written to the DigiMMICs.

Returns

ADI_DM_SUCCESS for success, an error code for errors.

6.2.5.5 adi_dm_BuiltinMimoSetup()

```
adi_dm_err_t adi_dm_BuiltinMimoSetup (
             const adi_dm_builtin_mimo_setup_t * p )
```

Register writes for ramp generation using built-in MIMO mode.

See "Built-in MIMO Mode" subsection of [2].

Parameters

in	*p*	parameters. See adi_dm_builtin_mimo_setup_t.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.6 adi_dm_BurstProfileSetup()

```
adi_dm_err_t adi_dm_BurstProfileSetup (
             const adi_dm_burst_profile_t * p,
             uint32_t * dma_mem_limit,
             uint32_t * bpid )
```

Memory writes for ramp generation using DMA.

See "DMA Interfacing" subsection of [2].

Parameters

in	*p*	Burst Profile to be written to memory.
in,out	*dma_mem_limit*	Pass in 0 for first burst profile. Pass in returned dma_mem_limit for subsequent profiles.
out	*bpid*	"Burst Profile Identifier" identifies this profile to adi_dm_DmaRampSetup() and adi_dm_SelectBurstProfile().

Returns

ADI_DM_SUCCESS for success, an error code for errors.

6.2.5.7 adi_dm_CalcPwrDetCfg()

```
adi_dm_err_t adi_dm_CalcPwrDetCfg (
             adi_dm_burst_profile_t * p,
             adi_dm_power_detector_meas_task_t * out )
```

Calculate power detector configuration for a burst, required by BIST15: Rx chain diagnostic check, BIST18: LO chain output monitor, BIST19: Tx output transmit power check, BIST44: Power detector rationality check, BIST53: Tx isolation monitor check, and BIST54: Tx output load monitor check.

Parameters

in	*p*	burst profile for which the expected count is being calculated.
out	*out*	rfpll_period_low_limit, rfpll_period_high_limit set to limits for the expected RFPLL period count.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

See also

adi_dm_PowerDetectorMeasTask

6.2.5.8 adi_dm_CalcRamp()

```
adi_dm_err_t adi_dm_CalcRamp (
             const adi_dm_ramp_shape_t * in,
             adi_dm_ramp_profile_t * ramp_profile,
             adi_dm_ramp_config_t * ramp_config,
             adi_dm_actual_ramp_shape_t * actual )
```

Calculate ramp profile from high level parameters.

Although this function is hardware-independent, it is part of the driver library so it can be called by user code. It is also strictly optional as the hardware-level parameters can just be passed from user code to `adi_dm_Builtin↩ MimoSetup` and `adi_dm_BurstProfileSetup`. This is by design.

Note

Because of hardware limitations, the ramp profile may not *exactly* match what was requested. In particular, the "Ramp Generator - AFE Timing" subsection of [2] states that

"the duration of each ramp must be an integer multiple of the CLK and AFE_CLK periods."

Parameters

in	*in*	Required timings and other parameters for the ramp.
out	*ramp_profile*	Ramp profile shape fields are filled in, other fields are zeroed.
out	*ramp_config*	Ramp config shape fields are filled in, other fields are zeroed.
out	*actual*	Actual values for timings etc. corresponding to the generated values.

Returns

ADI_DM_SUCCESS for success, an error code for errors

See also

adi_dm_BuiltinMimoSetup
adi_dm_BurstProfileSetup

6.2.5.9 adi_dm_CalcRfpllPeriod()

```
adi_dm_err_t adi_dm_CalcRfpllPeriod (
            adi_dm_calc_rfpll_period_t * p,
            adi_dm_write_rfpll_period_t * out )
```

Calculate expected RFPLL period count for a burst, required by BIST103c 'RFPLL period check'.

Parameters

in	*p*	burst profile and frequency for which the expected count is being calculated.
out	*out*	expected limits of RFPLL period counter for this burst.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

See also

adi_dm_WriteRfpllPeriod

6.2.5.10 adi_dm_CallFW()

```
static adi_dm_err_t adi_dm_CallFW (
            adi_dm_num_t dm_num,
            uint32_t cmd )  [inline], [static]
```

Execute a firmware task.

No parameters.

Parameters

in	*dm_num*	The DigiMMIC on which to run the task. If `dm_num` is ADI_DM_ALL_DIGIMMICS, task is executed on all DigiMMICs.
in	*cmd*	The firmware command to execute.

Returns

ADI_DM_SUCCESS for success, an error code for errors.

Hardware faults can be ignored by defining adi_dm_fault_status0_mask, adi_dm_fault_status1_mask and adi_↩ dm_fault_status2_mask in the user application.

6.2.5.11 adi_dm_CallFW2()

```
static adi_dm_err_t adi_dm_CallFW2 (
            adi_dm_num_t dm_num,
            uint32_t cmd,
            uint32_t p1,
            uint32_t p2 )  [inline], [static]
```

Execute a firmware task.

Two parameters.

Parameters

in	*dm_num*	The DigiMMIC on which to run the task. If `dm_num` is ADI_DM_ALL_DIGIMMICS, task is executed on all DigiMMICs.
in	*cmd*	The firmware command to execute.

Returns

ADI_DM_SUCCESS for success, an error code for errors.

Hardware faults can be ignored by defining adi_dm_fault_status0_mask, adi_dm_fault_status1_mask and adi_↩ dm_fault_status2_mask in the user application.

6.2.5.12 adi_dm_ClearLvdsTestPattern()

```
adi_dm_err_t adi_dm_ClearLvdsTestPattern (
            void  )
```

Unset test mode on LVDS devices.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.13 adi_dm_Crc32ADI()

```
uint32_t adi_dm_Crc32ADI (
            uint8_t * p,
            int n,
            int step )
```

Calculate a CRC/checksum.

Note

This is not the standard IEEE CRC32 but it does do the same computation as LVDS dataport.

Parameters

in	*p*	Pointer to start of data.
in	*n*	Length of data.
in	*step*	Step to next sample in the channel in bytes. For instance 4∗2 for a buffer in transmission order.

Returns

The CRC/checksum.

6.2.5.14 adi_dm_DmaRampSetup()

```
adi_dm_err_t adi_dm_DmaRampSetup (
            const adi_dm_dma_ramp_setup_t * p )
```

Register writes for ramp generation using DMA.

See "DMA Interfacing" subsection of [2].

Parameters

in	*p*	parameters. See adi_dm_dma_ramp_setup_t.

Returns

ADI_DM_SUCCESS for success, an error code for errors.

6.2.5.15 adi_dm_LockConfig()

```
adi_dm_err_t adi_dm_LockConfig (
            void )
```

Setup for checks that configuration does not change by accident.

Must be called if checks are to be run by adi_dm_PeriodicCalibration(). i.e. if `run_checks` member of adi_dm_periodic_calibration_t is set true. Must only be called after all setup is complete.

Returns

ADI_DM_SUCCESS for success, an error code for errors

See also

adi_dm_UnlockConfig

6.2.5.16 adi_dm_LvdsSetup()

```
adi_dm_err_t adi_dm_LvdsSetup (
             const adi_dm_lvds_setup_t * p )
```

Dataport configuration for LVDS parts.

Parameters

in	*p*	parameters. See adi_dm_lvds_setup_t

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.17 adi_dm_ManualSleep()

```
adi_dm_err_t adi_dm_ManualSleep (
             void  )
```

Enter low power state suitable for sleep between bursts.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.18 adi_dm_ManualWake()

```
adi_dm_err_t adi_dm_ManualWake (
             void  )
```

Exit the low power state suitable for sleep between bursts.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.19 adi_dm_MaskFaults()

```
adi_dm_err_t adi_dm_MaskFaults (
            adi_dm_mask_faults_t * p )
```

Ignore faults triggered by hardware or software errors.

Faults that are not ignored may cause any driver function to return early with ADI_DM_FIRMWARE_ERROR.

Parameters

in	*p*	Input parameters.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.20 adi_dm_MipiSetup()

```
adi_dm_err_t adi_dm_MipiSetup (
            const adi_dm_mipi_setup_t * p )
```

Dataport configuration for MIPI parts.

Parameters

in	*p*	parameters. See adi_dm_mipi_setup_t

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.21 adi_dm_PeriodicCalibration()

```
adi_dm_err_t adi_dm_PeriodicCalibration (
            const adi_dm_periodic_calibration_t * p )
```

Perform periodic firmware (re-)calibration as recommended by [1].

Parameters

in	*p*	parameters. See adi_dm_periodic_calibration_t.

Returns

ADI_DM_SUCCESS for success, an ADI_DM_PARAMETER_ERROR if insufficient memory

6.2.5.22 adi_dm_PinMux()

```
adi_dm_err_t adi_dm_PinMux (
            adi_dm_num_t dm_num,
            adi_dm_gpio_t pin,
            uint8_t fer,
            uint8_t mux )
```

Control function of an IO pad.

Selects the function of the named pad as described in "Input/Output Pad Control and General-Purpose Input/Output" [2]

Parameters

in	*dm_num*	The DigiMMIC to which the IO pad belongs. If `dm_num` is ADI_DM_ALL_DIGIMMICS, task is executed in parallel on all DigiMMICs.
in	*pin*	Ihe IO pad to change function.
in	*fer*	FER value from Table 52 in [2].
in	*mux*	MUX value from Table 52 in [2].

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.23 adi_dm_PowerDetectorMeasTask()

```
adi_dm_err_t adi_dm_PowerDetectorMeasTask (
            adi_dm_power_detector_meas_task_t * p )
```

Configures Power Detector Measurement tasks and sets measurement for next burst.

Must be called before trigger if power checks are to be run by adi_dm_PeriodicCalibration(). i.e. if `run_power↵_checks` member of adi_dm_periodic_calibration_t is set true. Cannot be used with adi_dm_BuiltinMimoSetup()

Parameters

in	*p*	containing Power detector measurement configuration.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

See also

adi_dm_CalcPwrDetCfg

6.2.5.24 adi_dm_PowerDown()

```
adi_dm_err_t adi_dm_PowerDown (
            void  )
```

Power down device.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.25 adi_dm_PowerUp()

```
adi_dm_err_t adi_dm_PowerUp (
            const adi_dm_power_up_t * p )
```

Power up device.

Parameters

in	*p*	parameters see adi_dm_power_up_t.

Returns

ADI_DM_SUCCESS on success, an error code on failure,

6.2.5.26 adi_dm_PwndnInterrupt()

```
adi_dm_err_t adi_dm_PwndnInterrupt (
            void  )
```

Send PWDNb interrupt.

Abort either ramp or tasklist, returning as-soon-as-possible to a known state.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.27 adi_dm_Read()

```
adi_dm_err_t adi_dm_Read (
            adi_dm_num_t dm_num,
            uint32_t addr,
            uint32_t * value )
```

Read a word from the DigiMMIC address space, checking for CRC error.

Access to SPI registers is supported by special rules for addresses in the range 0 to 255. If the address is in the range 0 to 255 a byte is read from the address over SPI and zero extended before assigning to value. Otherwise the address must be modulo 4, 4-bytes are read over SPI and assigned to value.

Parameters

in	*dm_num*	DigiMMIC to read from. `dm_num` may not be ADI_DM_ALL_DIGIMMICS.
in	*addr*	Address to read from in the DigiMMIC address space.
out	*value*	The 32-bit value read.

Returns

ADI_DM_SUCCESS for success, an error code for errors including ADI_DM_SPI_CRC_ERROR for CRC error.

6.2.5.28 adi_dm_ReadF32()

```
static adi_dm_err_t adi_dm_ReadF32 (
            adi_dm_num_t dm_num,
            uint32_t addr,
            float * value ) [inline], [static]
```

Read float from the DigiMMIC address space, checking for CRC errors.

Parameters

in	*dm_num*	DigiMMIC to read from. `dm_num` may not be ADI_DM_ALL_DIGIMMICS.
in	*addr*	Address to read from in the DigiMMIC address space.
out	*value*	The 32-bit floating-point value read.

Returns

ADI_DM_SUCCESS for success, an error code for errors including ADI_DM_SPI_CRC_ERROR for CRC error.

6.2.5.29 adi_dm_ReadF64()

```
static adi_dm_err_t adi_dm_ReadF64 (
            adi_dm_num_t dm_num,
            uint32_t addr,
            double * value )  [inline], [static]
```

Read double from the DigiMMIC address space, checking for CRC errors.

Parameters

in	*dm_num*	DigiMMIC to read from. `dm_num` may not be ADI_DM_ALL_DIGIMMICS.
in	*addr*	Address to read from in the DigiMMIC address space.
out	*value*	The 64-bit floating-point value read.

Returns

ADI_DM_SUCCESS for success, an error code for errors including ADI_DM_SPI_CRC_ERROR for CRC error.

6.2.5.30 adi_dm_ReadSysCalRx()

```
adi_dm_err_t adi_dm_ReadSysCalRx (
            uint8_t out[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_RX_SIZE] )
```

Read RX System Calibration values.

Used to read system calibration values obtained during system calibration. Refer to system calibration example on how to use this function.

Parameters

out	*out*	Output system calibration values

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.31 adi_dm_ReadSysCalTx()

```
adi_dm_err_t adi_dm_ReadSysCalTx (
            uint8_t out[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_TX_SIZE] )
```

Read TX System Calibration values.

Used to read system calibration values obtained during system calibration. Refer to system calibration example on how to use this function.

Parameters

out	*out*	Output system calibration values

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.32 adi_dm_ReadU64()

```
static adi_dm_err_t adi_dm_ReadU64 (
            adi_dm_num_t dm_num,
            uint32_t addr,
            uint64_t * value )  [inline], [static]
```

Read uint64_t value from the DigiMMIC address space, checking for CRC errors.

Parameters

in	*dm_num*	DigiMMIC to read from. `dm_num` may not be ADI_DM_ALL_DIGIMMICS.
in	*addr*	Address to read from in the DigiMMIC address space.
out	*value*	The 64-bit unsigned integer value read.

Returns

ADI_DM_SUCCESS for success, an error code for errors including ADI_DM_SPI_CRC_ERROR for CRC error.

6.2.5.33 adi_dm_RfpllLock()

```
adi_dm_err_t adi_dm_RfpllLock (
            uint64_t ramp_start_freq_hz )
```

Set the RFPLL lock frequency.

Sets the ramp start frequency and calls the firmware command to set the RFPLL's frequency accordingly.

Parameters

in	*ramp_start_freq_hz,the*	ramp start frequency in Hertz

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.34 adi_dm_RfpllReconfig()

```
adi_dm_err_t adi_dm_RfpllReconfig (
             const adi_dm_rfpll_reconfig_t * p )
```

Reconfigure RFPLL.

Calls the firmware calibrations recommended by [1] when one of the ramp start frequency or ramp bandwidth is changed.

Parameters

in	*p*	parameters, See adi_dm_rfpll_reconfig_t.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.35 adi_dm_RMW()

```
adi_dm_err_t adi_dm_RMW (
             adi_dm_num_t dm_num,
             uint32_t addr,
             uint32_t mask,
             uint32_t bits )
```

Read, modify, write a word in the DigiMMIC address space, checking for CRC error.

The effect is similar to the following (on the DigiMMIC):

```
uint32_t reg = *addr;
reg &= mask;
reg |= bits;
*addr = reg;
```

Parameters

in	*dm_num*	DigiMMIC to read from and write to. If `dm_num` is ADI_DM_ALL_DIGIMMICS, the read-modify-wite is performed on all DigiMMICs.
in	*addr*	Address in the DigiMMIC address space.
in	*mask*	Bitmask and-ed with value read from DigiMMIC address space.
in	*bits*	Bits or-ed with masked value prior to writing back to the DigiMMIC address space.

Returns

ADI_DM_SUCCESS for success, an error code for errors including ADI_DM_SPI_CRC_ERROR for CRC error.

6.2.5.36 adi_dm_SelectBurstProfile()

```
adi_dm_err_t adi_dm_SelectBurstProfile (
             uint32_t bpid )
```

Select the burst profile to use for subsequent bursts.

Requires adi_dm_DmaRampSetup() to have been called. Used to select a profile to replace the one passed to adi_dm_DmaRampSetup(). Only needs to be called if the current burst profile changes.

Parameters

in	*bpid*	"Burst Profile Identifier" identifies selected profile, from adi_dm_BurstProfileSetup().

Returns

ADI_DM_SUCCESS for success, an ADI_DM_PARAMETER_ERROR if insufficient memory

6.2.5.37 adi_dm_SetLvdsTestPattern()

```
adi_dm_err_t adi_dm_SetLvdsTestPattern (
             uint16_t pat[ADI_DM_NUM_DIGIMMIC][ADI_DM_NUM_RX] )
```

Set test mode on LVDS devices.

In this mode all data transmitted over the dataport is replaced by a known test pattern. The rest of the part should be programmed as normal and ramps triggered by calling adi_dm_Trigger() but all data transmitted will be replaced by the test pattern.

Parameters

in	*pat*	The test pattern to be used. A separate 16-bit value for each channel of each device.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.38 adi_dm_SetSysCal()

```
adi_dm_err_t adi_dm_SetSysCal (
             bool enable_sys_cal )
```

Set system calibration mode.

Used to enable or disable system calibration mode. This function should be called after adi_dm_PowerUp.

Parameters

in	*enable_sys_cal*	Boolean to enable system calibration

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.39 adi_dm_Tasklist()

```
adi_dm_err_t adi_dm_Tasklist (
             adi_dm_num_t dm_num,
             uint8_t listno )
```

Execute a tasklist.

Execute a list of firmware tasks with a single SPI command and wait until execution completes. The list must be in the have been added by adi_dm_TasklistPoolSetup()

Parameters

in	*dm_num*	Device on to run the tasklist. If ADI_DM_ALL_DIGIMMICS run on all active devices.
in	*listno*	Index of list in in tasklist pool.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

See also

adi_dm_TasklistPoolSetup, adi_dm_TasklistNoBlock

6.2.5.40 adi_dm_TasklistNoBlock()

```
adi_dm_err_t adi_dm_TasklistNoBlock (
             adi_dm_num_t dm_num,
             uint8_t listno )
```

Execute a tasklist without blocking.

Initiates execution a list of firmware tasks with a single SPI command without waiting until execution completes. The list must be in the have been added by adi_dm_TasklistPoolSetup()

Parameters

in	*dm_num*	Device on to run the tasklist. If ADI_DM_ALL_DIGIMMICS run on all active devices.
in	*listno*	Index of list in in tasklist pool.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

See also

adi_dm_TasklistPoolSetup, adi_dm_Tasklist

6.2.5.41 adi_dm_TasklistPoolSetup()

```
adi_dm_err_t adi_dm_TasklistPoolSetup (
            const adi_dm_tasklist_pool_setup_t * p,
            uint32_t * dma_mem_limit )
```

Create a tasklist pool.

Copies all tasklists down to device. Must be called after all calls to adi_dm_BurstProfileSetup().

Parameters

in	*p*	Tasklists to be written to memory.
in,out	*dma_mem_limit*	The same conventions as adi_dm_BurstProfileSetup().

Returns

ADI_DM_SUCCESS for success, an error code for errors.

6.2.5.42 adi_dm_TemperatureGet()

```
adi_dm_err_t adi_dm_TemperatureGet (
            adi_dm_temperature_t * out )
```

Get temperature of parts.

Parameters

`out`	*out*	Temperatures read from parts.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

6.2.5.43 adi_dm_Trigger()

```
adi_dm_err_t adi_dm_Trigger (
              void  )
```

Issue software trigger for a single burst.

Return immediately.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.44 adi_dm_TxSetup()

```
adi_dm_err_t adi_dm_TxSetup (
              const adi_dm_tx_setup_t * p )
```

Transmitter path (Tx) configuration.

Parameters

`in`	*p*	parameters. See adi_dm_tx_setup_t.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.45 adi_dm_UnlockConfig()

```
adi_dm_err_t adi_dm_UnlockConfig (
              void  )
```

Tear down for checks that configuration does not change by accident.

Must be called after adi_dm_LockConfig() before changing configuration. Note adi_dm_PeriodicCalibration() must not be called with `run_checks` member of adi_dm_periodic_calibration_t is set true when configuration is 'unlocked'.

Returns

ADI_DM_SUCCESS for success, an error code for errors

See also

adi_dm_UnlockConfig

6.2.5.46 adi_dm_Write()

```
adi_dm_err_t adi_dm_Write (
            adi_dm_num_t dm_num,
            uint32_t addr,
            uint32_t value )
```

Write a word to the DigiMMIC address space, checking for CRC error.

Access to SPI registers is supported by special rules for addresses in the range 0 to 255. If the address is in the range 0 to 255 the low 8-bits of value are written to the address using a byte SPI write. Otherwise the address must be modulo 4 and all 32-bits of value are written using a 4-byte SPI write.

Parameters

in	*dm_num*	DigiMMIC to write to. If `dm_num` is ADI_DM_ALL_DIGIMMICS, a write is broadcast to all DigiMMICs.
in	*addr*	Address to write to in the DigiMMIC address space.
in	*value*	32-bit value to write.

Returns

ADI_DM_SUCCESS for success, an error code for errors.

6.2.5.47 adi_dm_WriteF32()

```
static adi_dm_err_t adi_dm_WriteF32 (
            adi_dm_num_t dm_num,
            uint32_t addr,
            float value )  [inline], [static]
```

Write float to the DigiMMIC address space, checking for CRC errors.

Parameters

in	*dm_num*	DigiMMIC to write to. If `dm_num` is ADI_DM_ALL_DIGIMMICS, a write is broadcast to all DigiMMICs.
in	*addr*	Address to write to in the DigiMMIC address space.
in	*value*	The 32-bit floating-point value to write.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.48 adi_dm_WriteF64()

```
static adi_dm_err_t adi_dm_WriteF64 (
            adi_dm_num_t dm_num,
            uint32_t addr,
            double value )  [inline], [static]
```

Write double to the DigiMMIC address space, checking for CRC errors.

Parameters

in	*dm_num*	DigiMMIC to write to. If `dm_num` is ADI_DM_ALL_DIGIMMICS, a write is broadcast to all DigiMMICs.
in	*addr*	Address to write to in the DigiMMIC address space.
in	*value*	The 64-bit floating-point value to write.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.49 adi_dm_WriteRfpllPeriod()

```
adi_dm_err_t adi_dm_WriteRfpllPeriod (
            adi_dm_write_rfpll_period_t * p )
```

Calculate expected RFPLL period count for a burst, required by BIST103c 'RFPLL period check'.

It is assumed this has been called if adi_dm_PeriodicCalibration() is called with `run_rfpll_period_chk` member of adi_dm_periodic_calibration_t set true.

Parameters

in	*p*	estimated RFPLL period limits based upon current burst.

Returns

ADI_DM_SUCCESS for success, and an error code on failure.

See also

adi_dm_CalcRfpllPeriod

6.2.5.50 adi_dm_WriteSysCalRx()

```
adi_dm_err_t adi_dm_WriteSysCalRx (
             const uint8_t in[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_RX_SIZE] )
```

Write RX System Calibration values.

Used to write system calibration values which were obtained during system calibration. This function should be called after adi_dm_PowerUp.

Parameters

in	*in*	Input system calibration values

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.51 adi_dm_WriteSysCalTx()

```
adi_dm_err_t adi_dm_WriteSysCalTx (
             const uint8_t in[ADI_DM_NUM_DIGIMMIC][ADI_DM_SYSCAL_TX_SIZE] )
```

Write TX System Calibration values.

Used to write system calibration values which were obtained during system calibration. This function should be called after adi_dm_PowerUp.

Parameters

in	*in*	Input system calibration values

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.5.52 adi_dm_WriteU64()

```
static adi_dm_err_t adi_dm_WriteU64 (
             adi_dm_num_t dm_num,
             uint32_t addr,
             uint64_t value )  [inline], [static]
```

Write uint64_t to the DigiMMIC address space, checking for CRC errors.

Parameters

in	*dm_num*	DigiMMIC to write to. If `dm_num` is ADI_DM_ALL_DIGIMMICS, a write is broadcast to all DigiMMICs.
in	*addr*	Address to write to in the DigiMMIC address space.
in	*value*	64-bit unsigned integer value to write.

Returns

ADI_DM_SUCCESS for success, an error code for errors

6.2.6 Variable Documentation

6.2.6.1 adi_dm_active_digimmics

`adi_dm_num_t adi_dm_active_digimmics`

Number of devices actively controlled by the driver.

Must be a number between 1 and ADI_DM_NUM_DIGIMMIC inclusive. Must be set before calling adi_dm_InitDriver.

6.3 adi_dmhal.h File Reference

Public C hardware abstraction layer.

Macros

- #define ADI_DM_SPI_IS_LSBFIRST 0

  Set this define to 1 if the low order bit of each byte will be transmitted first by your implementation of adi_dm_SPI.
- #define ADI_DM_NUM_DIGIMMIC 1

  Set this define to the number of DigiMMICs to be controlled by the driver.
- #define ADI_DM_NUM_SPI_SLAVES (ADI_DM_NUM_DIGIMMIC + 1)

  Set this define to the number of SPI slaves to be controlled by the driver.
- #define ADI_DM_DIGIMMIC_MASTER_SPI_ADDR 0

  The slave parameter the driver should pass to adi_dm_SPI to access the 1st DigiMMIC under its control.
- #define ADI_DM_DIGIMMIC_SPI_SLAVE1 1

  The slave parameter the driver should pass to adi_dm_SPI to access the 2nd DigiMMIC under its control.
- #define ADI_DM_DIGIMMIC_SPI_SLAVE2 2

  The slave parameter the driver should pass to adi_dm_SPI to access the 3rd DigiMMIC under its control.
- #define ADI_DM_DIGIMMIC_SPI_SLAVE3 3

  The slave parameter the driver should pass to adi_dm_SPI to access the 4th DigiMMIC under its control.
- #define ADI_DM_FIRST_PMIC_SPI_SLAVE ADI_DM_NUM_DIGIMMIC

  The slave parameter the driver should pass to adi_dm_SPI to access the 1st PMIC under its control.
- #define ADI_DM_MAX_DATA_BYTES_PER_SPI_COMMAND 64

  The DigiMMIC SPI peripheral accepts commands with 1, 4, 16 and 64 data bytes.

Functions

Hardware Abstraction Layer

- adi_dm_err_t adi_dm_WaitGPIO (uint8_t dm_num, adi_dm_gpio_t hPin, bool bValue, uint32_t nTimeout↩NS)

  Wait for a GPIO to assume a particular value, with timeout.
- void adi_dm_WriteGPIO (uint8_t dm_num, adi_dm_gpio_t hPin, bool bValue)

  Set a GPIO to a particular value.
- void adi_dm_ReleaseGPIO (uint8_t dm_num, adi_dm_gpio_t hPin)

  Tri-state a GPIO.
- void adi_dm_DelayNS (uint32_t nTimeNS)

  Delay for nanoseconds.
- void adi_dm_SPI (uint8_t slave, uint8_t mosi[ ], uint8_t miso[ ], uint32_t count)

  Execute a SPI transfer.
- void adi_dm_Log (const char ∗msg,...)

  Write some tracing.
- adi_dm_err_t adi_dm_PowerUpSupplies (uint8_t dm_idx)

  Power up the supplies.
- adi_dm_err_t adi_dm_PowerDownSupplies (uint8_t dm_idx)

  Power down the supplies.

High level SPI driver

These functions issue SPI commands suitable for communicating with a remote DigiMMIC style SPI peripheral, as found in both the DigiMMIC and PMIC.

The code supplied with the DigiMMIC driver calls adi_dm_SPI to pass the formmated command to a local SPI device. The user may opt to provide an implementation for adi_dm_SPI or for this interface.

Note the interface currently assumes communication with only one peripheral at a time.

- void adi_dm_WriteSPI (uint8_t spi_slave, uint32_t addr, int_fast16_t bytes, const uint32_t ∗data)

  Write to a remote SPI device with ADI SPI slave IP.
- adi_dm_err_t adi_dm_ReadSPI (uint8_t spi_slave, uint32_t addr, int_fast16_t bytes, uint32_t ∗data)

  Read from a remote SPI device with ADI SPI slave IP.
- void adi_dm_ResetSPIConnection (uint8_t spi_slave)

  Set local model of remote ADI SPI slave IP to power on state.
- void adi_dm_InitSPIConnection (uint8_t spi_slave, bool disable_crc)

  Initialize the connection to remote ADI SPI slave IP.

6.3.1 Detailed Description

Public C hardware abstraction layer.

These functions are to be implemented by the user for their chosen platform.

6.3.2 Macro Definition Documentation

6.3.2.1 ADI_DM_DIGIMMIC_MASTER_SPI_ADDR

```
#define ADI_DM_DIGIMMIC_MASTER_SPI_ADDR 0
```

The slave parameter the driver should pass to adi_dm_SPI to access the 1st DigiMMIC under its control.

This must always be the (DigiMMIC) master in a cascaded system.

6.3.2.2 ADI_DM_MAX_DATA_BYTES_PER_SPI_COMMAND

```
#define ADI_DM_MAX_DATA_BYTES_PER_SPI_COMMAND 64
```

The DigiMMIC SPI peripheral accepts commands with 1, 4, 16 and 64 data bytes.

Change this define if the host SPI peripheral is restricted in number of data bytes supported. The value must be at least 4.

6.3.2.3 ADI_DM_NUM_DIGIMMIC

```
#define ADI_DM_NUM_DIGIMMIC 1
```

Set this define to the number of DigiMMICs to be controlled by the driver.

This count includes both the master and slaves.

6.3.2.4 ADI_DM_NUM_SPI_SLAVES

```
#define ADI_DM_NUM_SPI_SLAVES (ADI_DM_NUM_DIGIMMIC + 1)
```

Set this define to the number of SPI slaves to be controlled by the driver.

These slaves must support the ADI SPI protocol and may be accessed with adi_dm_WriteSPI and adi_dm_ReadSPI. Must be at least 1. The default value of 2 is intended to support a DigiMMIC and PMIC.

6.3.3 Function Documentation

6.3.3.1 adi_dm_DelayNS()

```
void adi_dm_DelayNS (
             uint32_t nTimeNS )
```

Delay for nanoseconds.

Parameters

in	*nTimeNS*	time to delay for

6.3.3.2 adi_dm_InitSPIConnection()

```
void adi_dm_InitSPIConnection (
             uint8_t spi_slave,
             bool disable_crc )
```

Initialize the connection to remote ADI SPI slave IP.

Parameters

in	*spi_slave*	The slave number of the connection to be reset.
in	*disable_crc*	If true disable the CRC checks on the connection.

6.3.3.3 adi_dm_Log()

```
void adi_dm_Log (
             const char * msg,
              ... )
```

Write some tracing.

For example, this function could output the message to UART. Note though that this function is a debugging tool for the user, there is no *requirement* for it to do anything. It could just return instantly although that would probably make development slower.

Parameters

in	*msg*	message to write

6.3.3.4 adi_dm_PowerDownSupplies()

```
adi_dm_err_t adi_dm_PowerDownSupplies (
             uint8_t dm_idx )
```

Power down the supplies.

This is the bottom half of Figures 12 an 13 in [2] .

Parameters

in	*dm_idx*	DigMMIC device index

Returns

ADI_DM_SUCCESS for success, error code for errors

6.3.3.5 adi_dm_PowerUpSupplies()

```
adi_dm_err_t adi_dm_PowerUpSupplies (
             uint8_t dm_idx )
```

Power up the supplies.

This is the bottom half of Figures 10 and 11 in [2] .

On boards with PMICs it may just be a matter of calling adi_pmic_PowerADAR690x().

Parameters

in	*dm_idx*	DigMMIC device index

Returns

ADI_DM_SUCCESS for success, error code for errors

6.3.3.6 adi_dm_ReadSPI()

```
adi_dm_err_t adi_dm_ReadSPI (
             uint8_t spi_slave,
             uint32_t addr,
             int_fast16_t bytes,
             uint32_t * data )
```

Read from a remote SPI device with ADI SPI slave IP.

Parameters

in	*spi_slave*	The SPI Slave for the remote device. A number between 0 and ADI_DM_NUM_SPI_SLAVES-1.
in	*addr*	32-bit address on remote device to be read from.
in	*bytes*	Number of data bytes to be received. Must be a number supported by remote device.
in	*data*	Array of data to be received.

Returns

ADI_DM_SUCCESS on success or

6.3.3.7 adi_dm_ReleaseGPIO()

```
void adi_dm_ReleaseGPIO (
             uint8_t dm_num,
             adi_dm_gpio_t hPin )
```

Tri-state a GPIO.

Parameters

in	*dm_num*	device index
in	*hPin*	pin that we want to tri-state

6.3.3.8 adi_dm_ResetSPIConnection()

```
void adi_dm_ResetSPIConnection (
            uint8_t spi_slave )
```

Set local model of remote ADI SPI slave IP to power on state.

Parameters

in	*spi_slave*	The slave number of the connection to be reset.

6.3.3.9 adi_dm_SPI()

```
void adi_dm_SPI (
            uint8_t slave,
            uint8_t mosi[ ],
            uint8_t miso[ ],
            uint32_t count )
```

Execute a SPI transfer.

The same number of bytes are written and read. Called by HLSPIDriver. The user may opt to provide an implementation for this or for HLSPIDriver.

Parameters

in	*slave*	index of the slave to be accessed
in	*mosi*	buffer containing bytes to transmit
out	*miso*	buffer to hold recieved bytes
in	*count*	the number of bytes to transfer

6.3.3.10 adi_dm_WaitGPIO()

```
adi_dm_err_t adi_dm_WaitGPIO (
            uint8_t dm_num,
            adi_dm_gpio_t hPin,
            bool bValue,
            uint32_t nTimeoutNS )
```

Wait for a GPIO to assume a particular value, with timeout.

Parameters

in	*dm_num*	device index

Parameters

in	*hPin*	pin to test
in	*bValue*	value to wait for
in	*nTimeoutNS*	how long to wait

Returns

ADI_DM_SUCCESS for success, ADI_DM_TIMEOUT for timeout, error codes for other errors

6.3.3.11 adi_dm_WriteGPIO()

```
void adi_dm_WriteGPIO (
            uint8_t dm_num,
            adi_dm_gpio_t hPin,
            bool bValue )
```

Set a GPIO to a particular value.

Parameters

in	*dm_num*	device index
in	*hPin*	pin to write
in	*bValue*	new state of pin

6.3.3.12 adi_dm_WriteSPI()

```
void adi_dm_WriteSPI (
            uint8_t spi_slave,
            uint32_t addr,
            int_fast16_t bytes,
            const uint32_t * data )
```

Write to a remote SPI device with ADI SPI slave IP.

Parameters

in	*spi_slave*	The SPI Slave for the remote device. A number between 0 and ADI_DM_NUM_SPI_SLAVES-1.
in	*addr*	32-bit address on remote device to be written to.
in	*bytes*	Number of data bytes to be sent. Must be a number supported by remote device.
in	*data*	Array of data to be sent.

6.4 adi_pmic_driver.h File Reference

Public C interface to the pmic driver.

```
#include "adi_dmdriver.h"
```

Data Structures

- struct adi_pmic_qa_status_t

  QA Watchdog timer status register.
- struct adi_pmic_qa_ctrl_t

  QA Watchdog timer control register.
- struct adi_pmic_freq_config_t

  Freq Spread Spectrum config register.
- struct adi_pmic_warn_fault_settings_t

  Warn/Fault Window setup.

Macros

- #define **ADI_PMIC_UNLOCK_KEY** 0x5F6A8C3DUL
- #define **ADI_PMIC_UNLOCK_ADDRESS** 0x001CUL
- #define **ADI_PMIC_FREQ_CONFIG** 0x8008UL
- #define **ADI_PMIC_BUCK_ONE_VOUT_SETTING_ADDRESS** 0x800CUL
- #define **ADI_PMIC_BUCK_ONE_VOUT_ADDRESS** 0x8010UL
- #define **ADI_PMIC_BUCK_FOUR_ADDRESS** 0x8014UL
- #define **ADI_PMIC_BUCK_DVS_ADDRESS** 0x8018UL
- #define **ADI_PMIC_LDO_ONE_ADDRESS** 0x801CUL
- #define **ADI_PMIC_LDO_TWO_ADDRESS** 0x8020UL
- #define **ADI_PMIC_LDO_THREE_ADDRESS** 0x8024UL
- #define **ADI_PMIC_LDO_FOUR_ADDRESS** 0x8028UL
- #define **ADI_PMIC_LDO_FIVE_ADDRESS** 0x802CUL
- #define **ADI_PMIC_LDO_SIX_ADDRESS** 0x8030UL
- #define **ADI_PMIC_LDO_SEVEN_ADDRESS** 0x8034UL
- #define **ADI_PMIC_WARN_WINDOW_ADDRESS** 0x8038UL
- #define **ADI_PMIC_FAULT_WINDOW_ADDRESS** 0x803CUL
- #define **ADI_PMIC_VOLTAGE_BLANK_TIME0_ADDRESS** 0x80B8UL
- #define **ADI_PMIC_VOLTAGE_BLANK_TIME1_ADDRESS** 0x80BCUL
- #define ADI_PMIC_LDO_ONE 1UL /∗∗ < Select LDO 1 ∗/

  List of LDO's that the user can change.
- #define **ADI_PMIC_LDO_TWO** 2UL /∗∗ < Select LDO 2 ∗/
- #define **ADI_PMIC_LDO_THREE** 3UL /∗∗ < Select LDO 3 ∗/
- #define **ADI_PMIC_LDO_FOUR** 4UL /∗∗ < Select LDO 4 ∗/
- #define **ADI_PMIC_LDO_FIVE** 5UL /∗∗ < Select LDO 5 ∗/
- #define **ADI_PMIC_LDO_SIX** 6UL /∗∗ < Select LDO 6 ∗/
- #define **ADI_PMIC_LDO_SEVEN** 7UL /∗∗ < Select LDO 7 ∗/
- #define ADI_PMIC_1P76_V 0UL /∗∗ < Set the LDO1 Land LDO2 voltage to 1.76V ∗/

  Possible voltage settings for LDO1 and LDO2.
- #define **ADI_PMIC_1P80_V** 1UL /∗∗ < Set the LDO1 Land LDO2 voltage to 1.80V ∗/
- #define **ADI_PMIC_1P85_V** 2UL /∗∗ < Set the LDO1 Land LDO2 voltage to 1.85V ∗/
- #define **ADI_PMIC_1P89_V** 3UL /∗∗ < Set the LDO1 Land LDO2 voltage to 1.89V ∗/

- #define **ADI_PMIC_1P94_V** 4UL /∗∗ < Set the LDO1 Land LDO2 voltage to 1.94V ∗/
- #define **ADI_PMIC_1P98_V** 5UL /∗∗ < Set the LDO1 Land LDO2 voltage to 1.98V ∗/
- #define **ADI_PMIC_2P03_V** 6UL /∗∗ < Set the LDO1 Land LDO2 voltage to 2.03V ∗/
- #define **ADI_PMIC_2P07_V** 7UL /∗∗ < Set the LDO1 Land LDO2 voltage to 2.07V ∗/
- #define **ADI_PMIC_LDO_1_2_INVALID** 8UL /∗∗ < Maximum value that the user can enter for LDO1 and LDO2 ∗/
- #define ADI_PMIC_0P86_V 0UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 0.86V ∗/

  Possible voltage settings for LDO3, LDO4 and LDO5.
- #define **ADI_PMIC_0P90_V** 1UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 0.90V ∗/
- #define **ADI_PMIC_0P94_V** 2UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 0.94V ∗/
- #define **ADI_PMIC_0P97_V** 3UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 0.97V ∗/
- #define **ADI_PMIC_1P01_V** 4UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 1.01V ∗/
- #define **ADI_PMIC_1P04_V** 5UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 1.04V ∗/
- #define **ADI_PMIC_1P08_V** 6UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 1.08V ∗/
- #define **ADI_PMIC_1P12_V** 7UL /∗∗ < Set the LDO3, LDO4 and LDO5 voltage to 1.12V ∗/
- #define **ADI_PMIC_LDO_3_4_5_INVALID** 8UL /∗∗ < Maximum value that the user can enter for LDO3, LDO4 and LDO5 ∗/
- #define ADI_PMIC_3P20_V 0UL /∗∗ < Set the LDO6 Land LDO7 voltage to 3.20V ∗/

  Possible voltage settings for LDO6 and LDO7.
- #define **ADI_PMIC_3P30_V** 1UL /∗∗ < Set the LDO6 Land LDO7 voltage to 3.30V ∗/
- #define **ADI_PMIC_3P35_V** 2UL /∗∗ < Set the LDO6 Land LDO7 voltage to 3.35V ∗/
- #define **ADI_PMIC_3P40_V** 3UL /∗∗ < Set the LDO6 Land LDO7 voltage to 3.40V ∗/
- #define **ADI_PMIC_LDO_6_7_INVALID** 4UL /∗∗ < Maximum value that the user can enter for LDO61 and LDO7 ∗/
- #define ADI_PMIC_OP55_V 0UL /∗∗ < Set the Buck4 voltage to 0.55V ∗/

  Possible voltage settings for Buck4.
- #define **ADI_PMIC_OP60_V** 1UL /∗∗ < Set the Buck4 voltage to 0.60V ∗/
- #define **ADI_PMIC_OP65_V** 2UL /∗∗ < Set the Buck4 voltage to 0.65V ∗/
- #define **ADI_PMIC_OP70_V** 3UL /∗∗ < Set the Buck4 voltage to 0.70V ∗/
- #define **ADI_PMIC_OP75_V** 4UL /∗∗ < Set the Buck4 voltage to 0.75V ∗/
- #define **ADI_PMIC_OP80_V** 5UL /∗∗ < Set the Buck4 voltage to 0.80V ∗/
- #define **ADI_PMIC_OP85_V** 6UL /∗∗ < Set the Buck4 voltage to 0.85V ∗/
- #define **ADI_PMIC_OP90_V** 7UL /∗∗ < Set the Buck4 voltage to 0.90V ∗/
- #define **ADI_PMIC_OP95_V** 8UL /∗∗ < Set the Buck4 voltage to 0.95V ∗/
- #define **ADI_PMIC_1P00_V** 9UL /∗∗ < Set the Buck4 voltage to 1.00V ∗/
- #define **ADI_PMIC_1P05_V** 10UL /∗∗ < Set the Buck4 voltage to 1.05V ∗/
- #define **ADI_PMIC_1P10_V** 11UL /∗∗ < Set the Buck4 voltage to 1.10V ∗/
- #define **ADI_PMIC_1P15_V** 12UL /∗∗ < Set the Buck4 voltage to 1.15V ∗/
- #define **ADI_PMIC_1P20_V** 13UL /∗∗ < Set the Buck4 voltage to 1.20V ∗/
- #define **ADI_PMIC_BUCK_4_INVALID** 14UL /∗∗ < Maximum value that the user can enter for voltages for PMIC ∗/
- #define ADI_PMIC_FB1_WARN_WINDOW_OFFSET 0UL /∗∗ < Offset into register for PMIC to set the FB1 warn/fault threshold ∗/

  Warn/Fault Window Offset bits.
- #define **ADI_PMIC_FB2_WARN_WINDOW_OFFSET** 2UL /∗∗ < Offset into register for PMIC to set the FB2 warn/fault threshold ∗/
- #define **ADI_PMIC_FB3_WARN_WINDOW_OFFSET** 4UL /∗∗ < Offset into register for PMIC to set the FB3 warn/fault threshold ∗/
- #define **ADI_PMIC_FB4_WARN_WINDOW_OFFSET** 6UL /∗∗ < Offset into register for PMIC to set the FB4 warn/fault threshold ∗/
- #define **ADI_PMIC_FB5_WARN_WINDOW_OFFSET** 8UL /∗∗ < Offset into register for PMIC to set the FB5 warn/fault threshold ∗/
- #define **ADI_PMIC_FB6_WARN_WINDOW_OFFSET** 10UL/∗∗ < Offset into register for PMIC to set the FB6 warn/fault threshold ∗/

- #define **ADI_PMIC_FB7_WARN_WINDOW_OFFSET** 12UL/∗∗ < Offset into register for PMIC to set the FB7 warn/fault threshold ∗/
- #define **ADI_PMIC_FB8_WARN_WINDOW_OFFSET** 14UL/∗∗ < Offset into register for PMIC to set the FB8 warn/fault threshold ∗/
- #define **ADI_PMIC_FB9_WARN_WINDOW_OFFSET** 16UL/∗∗ < Offset into register for PMIC to set the FB9 warn/fault threshold ∗/
- #define **ADI_PMIC_FB10_WARN_WINDOW_OFFSET** 18UL/∗∗ < Offset into register for PMIC to set the FB10 warn/fault threshold ∗/
- #define **ADI_PMIC_FB11_WARN_WINDOW_OFFSET** 20UL/∗∗ < Offset into register for PMIC to set the FB11 warn/fault threshold ∗/
- #define **ADI_PMIC_FB12_WARN_WINDOW_OFFSET** 22UL/∗∗ < Offset into register for PMIC to set the FB12 warn/fault threshold ∗/
- #define **ADI_PMIC_VM0_WARN_WINDOW_OFFSET** 24UL/∗∗ < Offset into register for PMIC to set the VM0 warn/fault threshold ∗/
- #define **ADI_PMIC_VM1_WARN_WINDOW_OFFSET** 26UL/∗∗ < Offset into register for PMIC to set the VM1 warn/fault threshold ∗/
- #define **ADI_PMIC_INVALID_WINDOW_OFFSET** 27UL/∗∗ < Max offset into register for PMIC to set the warn/fault thresholds ∗/
- #define ADI_PMIC_4_PERCENT 0UL /∗∗ < If the output volatge exceeds 4%, a warn/fault event is triggered ∗/

  Valid Thresholds for Warn/Fault Windows.
- #define **ADI_PMIC_5_PERCENT** 1UL /∗∗ < If the output volatge exceeds 5%, a warn/fault event is triggered ∗/
- #define **ADI_PMIC_6_PERCENT** 2UL /∗∗ < If the output volatge exceeds 6%, a warn/fault event is triggered ∗/
- #define **ADI_PMIC_8_PERCENT** 3UL /∗∗ < If the output volatge exceeds 8%, a warn/fault event is triggered ∗/
- #define **ADI_PMIC_INVALID_THRESHOLD** 4UL /∗∗ < Maximum value that the user can enter for % for fault/warn windows ∗/
- #define ADI_PMIC_16_US 0UL /∗∗ < Blank time setting of 16us ∗/

  Blank Times for Warn Fault Windows.
- #define **ADI_PMIC_32_US** 1UL /∗∗ < Blank time setting of 32us ∗/
- #define **ADI_PMIC_48_US** 2UL /∗∗ < Blank time setting of 48us ∗/
- #define **ADI_PMIC_64_US** 3UL /∗∗ < Blank time setting of 64us ∗/
- #define **ADI_PMIC_80_US** 4UL /∗∗ < Blank time setting of 80us ∗/
- #define **ADI_PMIC_96_US** 5UL /∗∗ < Blank time setting of 96us ∗/
- #define **ADI_PMIC_112_US** 6UL /∗∗ < Blank time setting of 112us ∗/
- #define **ADI_PMIC_128_US** 7UL /∗∗ < Blank time setting of 128us ∗/
- #define **ADI_PMIC_144_US** 8UL /∗∗ < Blank time setting of 144us ∗/
- #define **ADI_PMIC_160_US** 9UL /∗∗ < Blank time setting of 160us ∗/
- #define **ADI_PMIC_176_US** 10UL /∗∗ < Blank time setting of 176us ∗/
- #define **ADI_PMIC_192_US** 11UL /∗∗ < Blank time setting of 192us ∗/
- #define **ADI_PMIC_208_US** 12UL /∗∗ < Blank time setting of 208us ∗/
- #define **ADI_PMIC_240_US** 13UL /∗∗ < Blank time setting of 240us ∗/
- #define **ADI_PMIC_288_US** 14UL /∗∗ < Blank time setting of 288us ∗/
- #define **ADI_PMIC_352_US** 15UL /∗∗ < Blank time setting of 352us ∗/
- #define **ADI_PMIC_INVALID_BLANK_TIME** 16UL /∗∗ < Maximum value that the user can enter for Blank time for PMIC ∗/
- #define ADI_PMIC_BUCK1 1UL /∗∗ < Select Buck1 to modify settings for ∗/

  Possible Buck Choices.
- #define **ADI_PMIC_BUCK4** 2UL /∗∗ < Select Buck4 to modify settings for ∗/
- #define **ADI_PMIC_INVALID_BUCK** 3UL /∗∗ < Maximum value that the user can enter for Buck selection ∗/
- #define ADI_PMIC_INTERVAL_TIME_10US 0UL /∗∗ < 10us Dynamic Voltage Scaling Interval Time ∗/

  Output DVS Interval Times.

- #define **ADI_PMIC_INTERVAL_TIME_20US** 1UL /∗∗ < 20us Dynamic Voltage Scaling Interval Time ∗/
- #define **ADI_PMIC_INTERVAL_TIME_30US** 2UL /∗∗ < 30us Dynamic Voltage Scaling Interval Time ∗/
- #define **ADI_PMIC_INTERVAL_TIME_40US** 3UL /∗∗ < 40us Dynamic Voltage Scaling Interval Time ∗/
- #define **ADI_PMIC_INVALID_INTERVAL** 4UL /∗∗ < Maximum value that the user can enter for DVS interval time ∗/
- #define ADI_PMIC_SWEEP_DEPTH_2_PERCENT 0UL /∗∗ < 2% Sweep Depth of the Frequecny Spread Spectrum ∗/

  Sweep Depth of the Frequency Spread Spectrum.
- #define **ADI_PMIC_SWEEP_DEPTH_4_PERCENT** 1UL /∗∗ < 4% Sweep Depth of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_DEPTH_6_PERCENT** 2UL /∗∗ < 6% Sweep Depth of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_DEPTH_8_PERCENT** 3UL /∗∗ < 8% Sweep Depth of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_DEPTH_10_PERCENT** 4UL /∗∗ < 10% Sweep Depth of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_DEPTH_INVALID** 5UL /∗∗ < Maximum value that the user can enter for Sweep Depth ∗/
- #define ADI_PMIC_SWEEP_FREQ_5_KHz 0UL /∗∗ < 5KHz Sweep Freq of the Frequecny Spread Spectrum ∗/

  Sweep Frequency of the Frequency Spread Spectrum.
- #define **ADI_PMIC_SWEEP_FREQ_10_42_KHz** 1UL /∗∗ < 10.42KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_15_63_KHz** 2UL /∗∗ < 15.63KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_20_83_KHz** 3UL /∗∗ < 20.83KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_25_KHz** 4UL /∗∗ < 25KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_31_25_KHz** 5UL /∗∗ < 31.25KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_41_67_KHz** 6UL /∗∗ < 41.67KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_62_5_KHz** 7UL /∗∗ < 62.5KHz Sweep Freq of the Frequecny Spread Spectrum ∗/
- #define **ADI_PMIC_SWEEP_FREQ_INVALID** 8UL /∗∗ < Maximum value that the user can enter for Sweep Freq ∗/
- #define **ADI_PMIC_FAULT_WINDOW** 0x00UL /∗∗ <Define for setting the Fault window ∗/
- #define **ADI_PMIC_WARN_WINDOW** 0x01UL /∗∗ <Define for setting the warn window ∗/
- #define **ADI_PMIC_FEEDBACK1** 0x00UL
- #define **ADI_PMIC_FEEDBACK2** 0x01UL
- #define **ADI_PMIC_FEEDBACK3** 0x02UL
- #define **ADI_PMIC_FEEDBACK4** 0x03UL
- #define **ADI_PMIC_FEEDBACK5** 0x04UL
- #define **ADI_PMIC_FEEDBACK6** 0x05UL
- #define **ADI_PMIC_FEEDBACK7** 0x06UL
- #define **ADI_PMIC_FEEDBACK8** 0x07UL
- #define **ADI_PMIC_FEEDBACK9** 0x08UL
- #define **ADI_PMIC_FEEDBACK10** 0x09UL
- #define **ADI_PMIC_FEEDBACK11** 0x0AUL
- #define **ADI_PMIC_FEEDBACK12** 0x0BUL
- #define **ADI_PMIC_VM0** 0x0CUL
- #define **ADI_PMIC_VM1** 0x0DUL
- #define **ADI_PMIC_INVALID_VOLTAGE** 0x0EUL

Typedefs

- typedef uint32_t **adi_pmic_ldo_selection_t**
- typedef uint32_t **adi_pmic_ldo_1_2_voltages_t**
- typedef uint32_t **adi_pmic_ldo_3_4_5_voltages_t**
- typedef uint32_t **adi_pmic_ldo_6_7_voltages_t**
- typedef uint32_t **adi_pmic_buck_4_voltages_t**
- typedef uint32_t **adi_pmic_warn_fault_offset_t**
- typedef uint32_t **adi_pmic_threshold_values_t**
- typedef uint32_t **adi_pmic_blank_times_t**
- typedef uint32_t **adi_pmic_buck_sel_t**
- typedef uint32_t **adi_pmic_dvs_interval_times_t**
- typedef uint32_t **adi_pmic_sweep_depth_t**
- typedef uint32_t **adi_pmic_sweep_freq_t**
- typedef uint32_t **adi_pmic_warn_fault_window_t**
- typedef uint32_t **adi_pmic_voltages_t**

Enumerations

- enum adi_pmic_err_t { ADI_PMIC_SUCCESS = 0, ADI_PMIC_FAIL = -1, ADI_PMIC_PARAMETER_ERROR = -3 }

  Possible PMIC Error Codes.

Functions

- adi_pmic_err_t adi_pmic_PowerADAR690x (void)

  The recommended sequence for powering up an ADAR6901/2 device.
- adi_pmic_err_t adi_pmic_ReadChipId (uint32_t ∗id)

  Read the ID of the PMIC.
- adi_pmic_err_t adi_pmic_SetQaWdCtrl (adi_pmic_qa_ctrl_t ctrl)

  Set the ctrl register of the QA WDT.
- adi_pmic_err_t adi_pmic_GetQaWdCtrl (adi_pmic_qa_ctrl_t ∗ctrl)

  Get the ctrl register of the QA WDT.
- adi_pmic_err_t adi_pmic_GetQaWdStatus (adi_pmic_qa_status_t ∗status)

  Get the status of the QA Watchdog.
- adi_pmic_err_t adi_pmic_ServiceQaWatchdog (void)

  Service the QA watchdog.
- adi_pmic_err_t adi_pmic_SetLdoOneTwoVout (adi_pmic_ldo_selection_t ldo, adi_pmic_ldo_1_2_voltages↩_t vout)

  Set the voltage level for LOD1 and LDO2.
- adi_pmic_err_t adi_pmic_GetLdoOneTwoVout (adi_pmic_ldo_selection_t ldo, adi_pmic_ldo_1_2_voltages↩_t ∗vout)

  Get the voltage level for LOD1 and LDO2.
- adi_pmic_err_t adi_pmic_SetLdoThreeFourFiveVout (adi_pmic_ldo_selection_t ldo, adi_pmic_ldo_3_4_5_↩voltages_t vout)

  Set the voltage level for LOD3, LDO4 and LDO5.
- adi_pmic_err_t adi_pmic_GetLdoThreeFourFiveVout (adi_pmic_ldo_selection_t ldo, adi_pmic_ldo_3_4_5↩_voltages_t ∗vout)

  Get the voltage level for LOD3, LDO4 and LDO5.
- adi_pmic_err_t adi_pmic_SetLdoSixSevenVout (adi_pmic_ldo_selection_t ldo, adi_pmic_ldo_6_7_↩voltages_t vout)

Set the voltage level for LOD6 and LDO7.
- adi_pmic_err_t adi_pmic_GetLdoSixSevenVout (adi_pmic_ldo_selection_t ldo, adi_pmic_ldo_6_7_↩ voltages_t ∗vout)

  Get the voltage level for LOD6 and LDO7.
- adi_pmic_err_t adi_pmic_SetBuckOneVout (uint32_t vout)

  Set the voltage level for Buck1.
- adi_pmic_err_t adi_pmic_GetBuckOneVout (uint32_t ∗vout)

  Get the voltage level for Buck1.
- adi_pmic_err_t adi_pmic_SetBuckFourVout (adi_pmic_buck_4_voltages_t vout)

  Set the voltage level for Buck4.
- adi_pmic_err_t adi_pmic_GetBuckFourVout (adi_pmic_buck_4_voltages_t ∗vout)

  Get the voltage level for Buck4.
- adi_pmic_err_t adi_pmic_SetWarnFaultWindow (adi_pmic_voltages_t voltage, adi_pmic_warn_fault_settings_t cfg)

  Set the warning thresholds for the voltage supply.
- adi_pmic_err_t adi_pmic_GetWarnFaultWindow (adi_pmic_voltages_t voltage, adi_pmic_warn_fault_settings_t ∗cfg)

  Get the warning thresholds register value.
- adi_pmic_err_t adi_pmic_SetBuckDVS (adi_pmic_buck_sel_t sel, adi_pmic_dvs_interval_times_t interval)

  Set the DVS interval time for either Buck1 or BUCK4.
- adi_pmic_err_t adi_pmic_GetBuckDVS (adi_pmic_buck_sel_t sel, adi_pmic_dvs_interval_times_t ∗interval)

  Get the DVS interval time for either Buck1 or BUCK4.
- adi_pmic_err_t adi_pmic_SetFreqSpreadCfg (adi_pmic_freq_config_t cfg)

  Set the config register for the Frequency Spread Sprectrum.
- adi_pmic_err_t adi_pmic_GetFreqSpreadCfg (adi_pmic_freq_config_t ∗cfg)

  Get the config register for the Frequency Spread Sprectrum.
- void adi_pmic_Write (uint32_t addr, uint32_t value)

  Write a word to the PMIC address space.
- uint32_t adi_pmic_Read (uint32_t addr)

  Read a word from the PMIC address space.

6.4.1 Detailed Description

Public C interface to the pmic driver.

6.4.2 Enumeration Type Documentation

6.4.2.1 adi_pmic_err_t

```
enum adi_pmic_err_t
```

Possible PMIC Error Codes.

Enumerator

ADI_PMIC_SUCCESS	ok
ADI_PMIC_FAIL	Generic failure code.
ADI_PMIC_PARAMETER_ERROR	Parameter validation failed.

6.4.3 Function Documentation

6.4.3.1 adi_pmic_GetBuckDVS()

```
adi_pmic_err_t adi_pmic_GetBuckDVS (
            adi_pmic_buck_sel_t sel,
            adi_pmic_dvs_interval_times_t * interval )
```

Get the DVS interval time for either Buck1 or BUCK4.

Parameters

in	*sel*	- which buck to select
in	*interval*	- the interval time in US

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.2 adi_pmic_GetBuckFourVout()

```
adi_pmic_err_t adi_pmic_GetBuckFourVout (
            adi_pmic_buck_4_voltages_t * vout )
```

Get the voltage level for Buck4.

Parameters

out	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.3 adi_pmic_GetBuckOneVout()

```
adi_pmic_err_t adi_pmic_GetBuckOneVout (
            uint32_t * vout )
```

Get the voltage level for Buck1.

Parameters

out	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.4 adi_pmic_GetFreqSpreadCfg()

```
adi_pmic_err_t adi_pmic_GetFreqSpreadCfg (
            adi_pmic_freq_config_t * cfg )
```

Get the config register for the Frequency Spread Sprectrum.

Parameters

out	*cfg*	- the config structure

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.5 adi_pmic_GetLdoOneTwoVout()

```
adi_pmic_err_t adi_pmic_GetLdoOneTwoVout (
            adi_pmic_ldo_selection_t ldo,
            adi_pmic_ldo_1_2_voltages_t * vout )
```

Get the voltage level for LOD1 and LDO2.

Parameters

in	*ldo*	- the ldo enum
out	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.6 adi_pmic_GetLdoSixSevenVout()

```
adi_pmic_err_t adi_pmic_GetLdoSixSevenVout (
            adi_pmic_ldo_selection_t ldo,
            adi_pmic_ldo_6_7_voltages_t * vout )
```

Get the voltage level for LOD6 and LDO7.

Parameters

in	*ldo*	- the ldo enum
out	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.7 adi_pmic_GetLdoThreeFourFiveVout()

```
adi_pmic_err_t adi_pmic_GetLdoThreeFourFiveVout (
            adi_pmic_ldo_selection_t ldo,
            adi_pmic_ldo_3_4_5_voltages_t * vout )
```

Get the voltage level for LOD3, LDO4 and LDO5.

Parameters

in	*ldo*	- the ldo enum
out	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.8 adi_pmic_GetQaWdCtrl()

```
adi_pmic_err_t adi_pmic_GetQaWdCtrl (
            adi_pmic_qa_ctrl_t * ctrl )
```

Get the ctrl register of the QA WDT.

Parameters

in	*ctrl*	- structure of the ctrl register for the QA WDT

6.4.3.9 adi_pmic_GetQaWdStatus()

```
adi_pmic_err_t adi_pmic_GetQaWdStatus (
            adi_pmic_qa_status_t * status )
```

Get the status of the QA Watchdog.

Parameters

out	*status*	- structure of the current state of the QA WDT

6.4.3.10 adi_pmic_GetWarnFaultWindow()

```
adi_pmic_err_t adi_pmic_GetWarnFaultWindow (
            adi_pmic_voltages_t voltage,
            adi_pmic_warn_fault_settings_t * cfg )
```

Get the warning thresholds register value.

Parameters

out	*res*	- The contents of the Warn Window register

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.11 adi_pmic_PowerADAR690x()

```
adi_pmic_err_t adi_pmic_PowerADAR690x (
            void )
```

The recommended sequence for powering up an ADAR6901/2 device.

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.12 adi_pmic_Read()

```
uint32_t adi_pmic_Read (
            uint32_t addr )
```

Read a word from the PMIC address space.

Parameters

in	*addr*	32-bit address to write to in the PMIC address space
out	*value*	32-bit value to write

6.4.3.13 adi_pmic_ServiceQaWatchdog()

```
adi_pmic_err_t adi_pmic_ServiceQaWatchdog (
             void  )
```

Service the QA watchdog.

The processor calls this functions which reads the token and writes back the answer

6.4.3.14 adi_pmic_SetBuckDVS()

```
adi_pmic_err_t adi_pmic_SetBuckDVS (
             adi_pmic_buck_sel_t sel,
             adi_pmic_dvs_interval_times_t interval )
```

Set the DVS interval time for either Buck1 or BUCK4.

Parameters

in	*sel*	- which buck to select
in	*interval*	- the interval time in US

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.15 adi_pmic_SetBuckFourVout()

```
adi_pmic_err_t adi_pmic_SetBuckFourVout (
             adi_pmic_buck_4_voltages_t vout )
```

Set the voltage level for Buck4.

Parameters

in	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.16 adi_pmic_SetBuckOneVout()

```
adi_pmic_err_t adi_pmic_SetBuckOneVout (
            uint32_t vout )
```

Set the voltage level for Buck1.

Parameters

in	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.17 adi_pmic_SetFreqSpreadCfg()

```
adi_pmic_err_t adi_pmic_SetFreqSpreadCfg (
            adi_pmic_freq_config_t cfg )
```

Set the config register for the Frequency Spread Sprectrum.

Parameters

in	*cfg*	- the config structure

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.18 adi_pmic_SetLdoOneTwoVout()

```
adi_pmic_err_t adi_pmic_SetLdoOneTwoVout (
            adi_pmic_ldo_selection_t ldo,
            adi_pmic_ldo_1_2_voltages_t vout )
```

Set the voltage level for LOD1 and LDO2.

Parameters

in	*ldo*	- the ldo enum
in	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.19 adi_pmic_SetLdoSixSevenVout()

```
adi_pmic_err_t adi_pmic_SetLdoSixSevenVout (
            adi_pmic_ldo_selection_t ldo,
            adi_pmic_ldo_6_7_voltages_t vout )
```

Set the voltage level for LOD6 and LDO7.

Parameters

in	*ldo*	- the ldo enum
in	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.20 adi_pmic_SetLdoThreeFourFiveVout()

```
adi_pmic_err_t adi_pmic_SetLdoThreeFourFiveVout (
            adi_pmic_ldo_selection_t ldo,
            adi_pmic_ldo_3_4_5_voltages_t vout )
```

Set the voltage level for LOD3, LDO4 and LDO5.

Parameters

in	*ldo*	- the ldo enum
in	*vout*	- the selected voltage

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.21 adi_pmic_SetQaWdCtrl()

```
adi_pmic_err_t adi_pmic_SetQaWdCtrl (
             adi_pmic_qa_ctrl_t ctrl )
```

Set the ctrl register of the QA WDT.

Parameters

in	*ctrl*	- structure of the ctrl register for the QA WDT

6.4.3.22 adi_pmic_SetWarnFaultWindow()

```
adi_pmic_err_t adi_pmic_SetWarnFaultWindow (
             adi_pmic_voltages_t voltage,
             adi_pmic_warn_fault_settings_t cfg )
```

Set the warning thresholds for the voltage supply.

Parameters

in	*faultwindowOffset*	- The offset into the register to get the correct voltage supply
in	*thresholdlevel*	- window percentages for threshold levels
in	*blankTime*	- Voltage monitor blank time

Returns

ADI_PMIC_SUCCESS (0) for success, ADI_PMIC_FAIL on failure

6.4.3.23 adi_pmic_Write()

```
void adi_pmic_Write (
             uint32_t addr,
             uint32_t value )
```

Write a word to the PMIC address space.

Parameters

in	*addr*	32-bit address to write to in the PMIC address space
in	*value*	32-bit value to write

Bibliography

[1] Analog Devices Inc., Norwood, MA, USA. *ADAR6901/ADAR6902 Firmware Reference Manual UG-1002*, 1.3 edition, October 2020. 1, 3, 6, 7, 59, 60, 61, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 116, 136, 142

[2] Analog Devices Inc., Norwood, MA, USA. *ADAR6901/ADAR6902 Hardware Reference Manual UG-1485*, prd edition, August 2020. 1, 2, 3, 7, 20, 21, 24, 26, 27, 28, 29, 32, 33, 34, 38, 42, 44, 46, 47, 52, 53, 130, 131, 134, 137, 153, 154

Index