ALPHA DATA

XRM(2)-DAC-D4/1G User Guide

Document Revision: 2.2
Mar 8, 2018

	Head Office	US Office
Address:	4 West Silvermills Lane, Edinburgh, EH3 5BD, UK	611 Corporate Circle Suite H Golden, CO 80401
Telephone:	+44 131 558 2600	(303) 954 8768
Fax:	+44 131 558 2700	(866) 820 9956 - toll free
email:	sales@alpha-data.com	sales@alpha-data.com
website:	http://www.alpha-data.com	http://www.alpha-data.com

Table Of Contents

List of Tables

List of Figures

1 Introduction

Alpha Data provide three variants of a fast analogue signal generation card operating at sampling frequencies up to 1 GHz, based on the DAC5681, DAC5681Z and DAC5682Z devices from Texas Instruments.

The DAC5681 provides a non-interpolating architecture for wideband signal generation.

The DAC5681Z implements an interpolating architecture and provides filtering and mixing circuitry and is essentially a single-channel version of the DAC5682Z.

The DAC5682Z also has an interpolating architecture and provides filtering and mixing circuitry for the two DACs contained within the package. Note that in this case the second DAC output in each package is not accessible, although the data can be processed and combined internally as two channels.

All versions utilise a common circuit board with build options being used to match the board configuration to the DAC fitted.

These boards differ only in the following aspects:

a) The DAC fitted - the DAC5681/ DAC5681Z version uses a single channel DAC normally aimed at producing wide bandwidth signals. The DAC5682Z has 2 full DAC channels, which allows signal generation from complex data streams.

b) Minor differences in pin functions.

c) Register addresses and bit allocations for the internal DAC registers in the DAC581Z and DAC5682Z are supersets of those in the DAC5681.

d) Inclusion of a PLL on the interpolating devices (DAC5681Z, DAC5682Z) for DAC sample clock generation from a reference clock. In normal circumstances this facility is not used since because of the limited set of frequencies that can be produced, but if used the high-frequency clocks required for the FPGA must be synthesised in the FPGA fabric using MMCM or DCM.

These XRM modules are compatible with Alpha-Data's family of FPGA cards fitted with Virtex 4, Virtex 5, Virtex 6, Kintex 7 and Virtex 7 devices.

Both configurations are referred to in this document by the generic title XRM(2)-DAC-D4-1G; where required, any DAC-specific differences will be made explicit.

The code and hardware descriptions given below reflect the functions implemented at the date of this document.

1.1 Block Diagram

The block diagram (see Figure Block Diagram) shows the major components of the XRM(2)-DAC-D4-1G board.

The DAC has its own dedicated power supplies and uses a mixture of single-ended (serial control) and differential (data, clocks and synchronisation) signals to/from the FPGA. A clock synthesis/distribution circuit is included to provide flexible clock generation options.

Dedicated serial interfaces are implemented in the VHDL code to communicate with the DAC and the synthesiser. These interfaces are initialised automatically by the FPGA as part of the reset sequence.

DAC sample data is transferred to each DAC from the FPGA via 16 LVDS pairs plus synchronisation (SYNC) and data clock (DCLK) differential pairs. The data clock (DCLK), synchronous with the data, is generated from a half-rate copy of the DAC clock (DACCLK). The DCLK signal runs at 0.5 * the DAC clock rate present on the clock input connector. Within the FPGA the DCLK is further divided by 2 for use as a global clock (FABRCLK) for data generation, so runs at 0.25 * DACCLK.

The synthesiser/distribution circuit provides three options for clocking the DAC.

a) Internal reference, internally synthesised clock, giving integer sub-divisions, including 1, of 1GHz.

b) Externally generated clock, integer sub-divisions, including 1.

c) External reference, internally synthesised clock, giving integer sub-divisions of 1GHz.

A pair of LVTTL outputs ('TRIG' and 'AUX') is provided (3V3 signal levels) via SMA connectors. In addition, two direct connections to FPGA pins via UFL connectors are also available for fast signalling interconnect between multiple DAC cards or other devices.

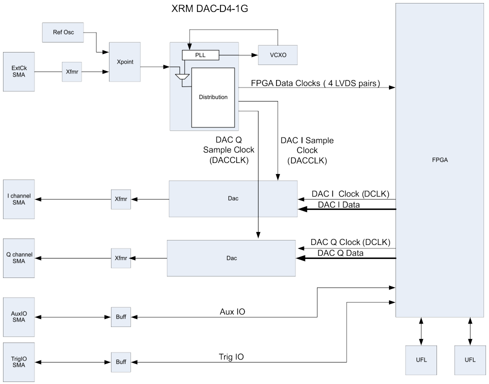


Figure 1 : Block Diagram

1.2 XRM and XRM2

The latest generation of FPGA cards from Alpha Data use a modified version of the XRM interface originally implemented on legacy FPGA (Virtex4, Virtex5) cards. From a user viewpoint these two interfaces are identical so references to 'XRM' signals refer also to XRM2 implementations. Where any differences between the two interfaces are relevant to the operation of the XRM module they will be explicitly stated in the text.

On the XRM(2)-DAC-D4 the principal difference lies in how the I/O voltages for the banks connected to the XRM are set.

1.2.1 Signalling Voltage

The signals to the DAC are mainly LVDS, with some single-ended signals for serial interfaces etc. Differential termination is used in the FPGA for clocks etc. which requires that the signalling voltage is set to a suitable level on the host FPGA card. FPGA cards using the XRM2 interface (e.g. Virtex6, Virtex7 etc.) this voltage is set automatically. On the XRM interface (boards fitted with Virtex4 or Virtex5) this should be set manually to 2v5. This voltage level is required solely to ensure correct termination values in the FPGA; the DAC board will not be damaged if this voltage is inadvertently set to 3v3.

Single-ended signals are all level-translated to hsift signals to/from the device signalling levels to theat of the FPGA I/O bank supply being used.

1.3 Build Level

The description in this document refer to release 5.0 of the xrm_dac_d4_1g code, dated 15/11/17. Current board hardware revision is rev 6 and this code supports rev 3 and later builds. Contact the factory for support for board versions earlier than rev 3.

1.4 Alpha Data SDK Versions

All VHDL code for legacy boards is built using Alpha Data's SDK version 4.9.3. This SDK version is frozen at this revision.

All VHDL code for current boards uses Alpha Data's ADMXRCG3SDK version 1.7.0.

1.5 Xilinx Tool Versions

The VHDL can be synthesised using either ISE or Vivado. Only FPGA cards fitted with Virtex7 or Kintex7 FPGAs are supported in Vivado.

The currently supported version of ISE for synthesis and bitfile generation is version 14.7.

The currently supported version of Vivado for synthesis and bitfile generation is version 2017.2.

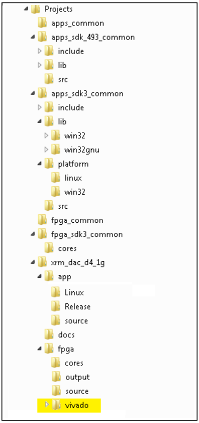
1.6 ISE Projects

1.6.1 Structure

The example code for ISE builds runs this in batch mode, using makefiles to control the various steps that are required, based on the methodology used in the both variants of Alpha Data SDKs. The files required for each FPGA card type are defined in a file with the extension 'prj'; the switches necessary for guiding synthesis, map, place and route, and bit file generation are defined for each FPGA type in files with the 'scr' extension.

The file paths defined in the prj file reflect the structure of the example code; any changes to the project structure must be reflected in the paths defined in the prj file.

The default project structure is shown below; this includes the additional folder for the Vivado version of the project (highlighted).



```
Projects
  apps_common
  apps_sdk_493_common
    include
    lib
    src
  apps_sdk3_common
    include
    lib
      win32
      win32gnu
    platform
      linux
      win32
    src
  fpga_common
  fpga_sdk3_common
    cores
  xrm_dac_d4_1g
    app
      Linux
      Release
      source
    docs
    fpga
      cores
      output
      source
      vivado
```

Figure 2 : Default Project Structure

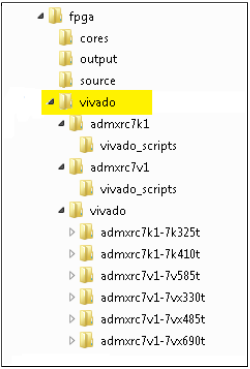
1.7 Vivado Projects

1.7.1 Vivado Folder Structure

The additional folder present in the standard release 'FPGA' folder for producing bitfiles using Vivado is highlighted. In the case shown, this additional folder is called 'vivado' and is referred to as the main Vivado folder in this discussion. This folder name be any legal name as long as the paths (relative and absolute) to the SDK files, the project source and the project core files is preserved. In other words it should be at the same level of the folder hierarchy as the 'source' and 'cores' folders. Any change in these paths will require modifications to paths defined in TCL scripts.

The generation of project files for Vivado uses scripts based on those in the Vivado examples provided by the SDK. The TCL files automate the generation of Vivado project(s), which ensures that the xpr files produced include all settings required for correct configuration of both synthesis and post-synthesis file generation. Specifying these options manually is unlikely to set all options correctly so it is **strongly** recommended that the TCL files are used to generate project files and folder structures.

This uses the same *'design-model-device'* syntax as the Alpha Data SDK examples, where the *'design'* equates to the XRM type, the *'model'* equates to the ADMXRC board type and the *'device'* equates to the specific FPGA on the FPGA card. In this case the *'device'* field is fixed since these designs are for a specific XRM.



```
fpga
    cores
    output
    source
    vivado
        admxrc7k1
            vivado_scripts
        admxrc7v1
            vivado_scripts
        vivado
            admxrc7k1-7k325t
            admxrc7k1-7k410t
            admxrc7v1-7v585t
            admxrc7v1-7vx330t
            admxrc7v1-7vx485t
            admxrc7v1-7vx690t
```

Figure 3 : Vivado Project Structure

The main Vivado folder contains three files - genxpr.bat, genxpr.tcl, makexpr.tcl plus a folder named after each supported model of board; currently there are only two models supported, the 7K1 and the 7V1.The *model* folders contain TCL files that are specific to the design.

Each model folder contains a *'vivado-scripts'* folder and an xdc file. The xdc file specifies the XRM(2)-to-admxrc connections for that model. Each *'design-model-device.tcl'* file in the *'vivado-scripts'* folder corresponds to the scr file for ISE; the *'design-model.tcl'* file in the same folder performs a similar function to the prj file. In this file the required source files (.vhd and .ngc) and the paths to them relative to the main Vivado folder are defined.

All of these files (xdc, TCL) are provided as part of the release. Normally only the *'design-model.tcl'* file is ever altered and then that only when path names need to be corrected. Note that additional (standard) xdc files from the SDK are required in order to build the bit file correctly and these are also referenced in the *'design-model.tcl'* file.

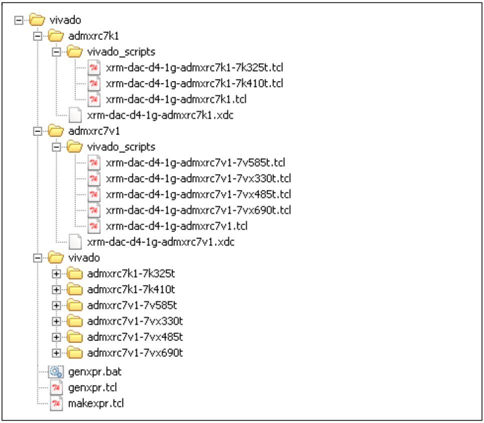
Once the batch file has been run, the main Vivado folder will contain a third folder which is created by Vivado as part of the script. This folder is named *'vivado'* (note that this is at a level **below** the main Vivado folder) and the sub-folders within, named from a combination of the valid *model-device* variables, are used as the location for working folders when projects are run.

Genxpr.bat accepts command line parameters to allow projects for single boards, single board types or all supported boards to be generated. The absence of any command line parameters is interpreted as a command to make all projects. Using the batch file allows parameters to be passed to the script, something that is not possible when using the standard TCL 'source' command from within Vivado (or other TCL interpreters).

The batch file runs genxpr.tcl in Vivado using batch mode; note that any existing xpr file is **not** overwritten unless over-write permission is **explicitly** specified on the command line. Genxpr.tcl subsequently invokes makexpr.tcl for each required model and device specified via the command line, creating the working folders for each model/ design combination specified.

Genxpr.tcl is based on the file of the same name that can be found in the SDK Vivado projects. In the SDK case, the script allows all SDK design examples (simple, uber etc.) to be generated; in the XRM case there is only a single design, the one for the XRM(2)-dac-d4. Furthermore, in most cases the top-level entity name (reflected in the vhdl file name) is not the same as the design name so an extra function has been added (gen_design_top) to handle this.

The folders thus created for each of the model types contains the xpr and a *'design-model-device_bit_post.tcl'* for that model. The projects are run from the Vivado GUI by opening the required xpr file; any files produced by Vivado are stored in the same folder to keep things neat. The *'design-model-device_bit_post.tcl'* specifies some post-processing of the bit file necessary to produce the bitfile name expected by the SDK application.



```
vivado
  admxrc7k1
    vivado_scripts
      xrm-dac-d4-1g-admxrc7k1-7k325t.tcl
      xrm-dac-d4-1g-admxrc7k1-7k410t.tcl
      xrm-dac-d4-1g-admxrc7k1.tcl
    xrm-dac-d4-1g-admxrc7k1.xdc
  admxrc7v1
    vivado_scripts
      xrm-dac-d4-1g-admxrc7v1-7v585t.tcl
      xrm-dac-d4-1g-admxrc7v1-7vx330t.tcl
      xrm-dac-d4-1g-admxrc7v1-7vx485t.tcl
      xrm-dac-d4-1g-admxrc7v1-7vx690t.tcl
      xrm-dac-d4-1g-admxrc7v1.tcl
    xrm-dac-d4-1g-admxrc7v1.xdc
  vivado
    admxrc7k1-7k325t
    admxrc7k1-7k410t
    admxrc7v1-7v585t
    admxrc7v1-7vx330t
    admxrc7v1-7vx485t
    admxrc7v1-7vx690t
  genxpr.bat
  genxpr.tcl
  makexpr.tcl
```

Figure 4 : Vivado Files

It is **strongly** recommended to retain the folder structure shown in the main Vivado folder in order to ensure that TCL files provided work correctly. Alterations to this structure will entail the need for extensive modification of paths/files embedded in the scripts.

2 Hardware

2.1 Hardware Operation

The application must first configure the FPGA with the bit stream using the standard functions provided in the SDK before any hardware or FPGA registers can be accessed via the local bus .

The various system blocks must be configured in the correct sequence in order to generate analogue signals correctly. First the clock source must be established, the synthesiser configured, then the FPGA clock generation circuitry. Once a stable DCLK signal has been established, the DAC internal registers can be configured to suit the operating frequency required.

Default register settings are written to the DAC and synthesiser following system reset/ FPGA configuration, but the stability of the clocks during this period cannot be guaranteed so the full clock configuration sequence should be explicitly run by the application prior to use.

The DAC initialisation sequence, which must be run for any change in clock frequency requiring alteration of the control bits for the DAC DLL, defaults to the sequence for the DAC5681. This can be easily changed in the VHDL to default to the DAC5682Z sequence. In both cases the assumed DAC clock frequency is 1GHz. Any change in the clock speed from this value is used to re-configure the clock multiplexing, the settings for the DCM clock used by the FPGA and the DAC registers. This application also provides code to interrogate the DAC type and implement the correct settings.

As shown in the Block Diagram, the DAC sample clock fed to DAC I and DAC Q run at the full rate (1GHz maximum); the relevant DCLK clocks are at half this rate, since the data interface is DDR. A total of four differential clock ports are available to capture the data clock reference. In practice, only three are used since the pinout of the FPGA requires a maximum of three clock regions in order to support the range of Virtex 4 and Virtex 5 boards. The fourth is connected to a counter for diagnostic purposes.

2.2 Connector Signals

There are five external connectors accessible on the D4-1G board plus a further two which are used for fast signalling. Of these, only the TRIG and AUX ports have any significant protection whilst the clock input has limited overdrive protection.

The DAC outputs are ac coupled and present a 50R output impedance, both of which factors give some limited protection.

The two UFL connectors used for fast signalling are connected directly to FPGA pins. Any signals outwith normal 2v5 LVCMOS signalling levels may cause permanent damage to the FPGA.

2.3 DAC Serial Interface

The reset state of the DAC configures the serial interface for 3-wire operation. The example code uses a 4-wire interface so the first operation following a hardware reset of the DAC (via the RESETB pin of the DAC) must be a write to DAC register CONFIG5 which sets D7 to ensure that the DAC is set to operate in 4-wire mode.

The Status and Func ports have no function on the DAC interface.

The maximum speed of the serial interface is 10 MHz (100 ns). The example code runs the state machine at LCLK/10, controlled by the END_CNT generic. This results in the interface clock running at LCLK/20, limiting the interface to less than 4 MHz for all settings of LCLK. The default rate is 1.6 MHz for LCLK= 33 MHz (virtex4, Virtex 5) or LCLK=80 MHz (Virtex6, Virtex 7, Kintex 7).

2.4 DAC Programming

The register values following a reset of the DAC require to be modified for correct operation. Firstly the interface must be set to operate in 4-wire mode as noted above. For the DAC 5681 this is essentially all that needs to be done, since the clock is set up explicitly by the application, which in turn configures the DLL settings and re-starts the DAC DLL.

The DAC5682Z and the DAC5681Z require that the PLL, FIR and CMIX blocks are disabled; the DAC5682Z also requires that the output of the second (B channel) is disabled. This requires a read of the "device id" bits to determine the device fitted to the board.

The operation of any of the DACs requires a hardware reset each time the clock frequency is changed in order to achieve lock of the DAC DLL, so this reset sequence must be followed each time.

2.5 Synthesiser Serial Interface

The synthesiser (AD9510) is programmed using a 4-wire interface which is virtually identical to that for the DAC. The maximum operating speed of the serial interface is 25 MHz. The example code has the END_CNT generic set to a value of 2, forcing the clock rate on the interface to be 0.25 *LCLK rate (~ 8 MHz for the default LCLK of 33 MHz).

The reset state of the synthesiser/distribution circuit results in incorrect divider values for the DAC and the FPGA clocks so the synthesiser's internal registers must be set explicitly to the required values.

The STATUS port on the serial interface can be used to provide real-time monitoring of various signals within the synthesiser; this is normally set to the synthesiser lock signal.

The FUNC port provides a real-time control input for the synthesiser. Note that the synthesiser treats this pin as an active-low reset by default, which must be removed in order to program the synthesiser. For this reason, this port is normally pulled high and should default high at FPGA reset if used in an application, otherwise the default configuration data will be ignored by the device.

2.6 Synthesiser Programming

The synthesiser (AD9510) provides three main functions:

a) Clock synthesis
b) Clock routeing and division
c) Clock output type

The output type is configured to suit the requirements of the DAC and FPGA clock inputs and should not be altered, although unused buffer outputs can be disabled to reduce power consumption. The clock dividers prior to the buffer outputs are normally configured via the application code but can be customised if required.

In normal use, the DAC clock input, driven by the synthesiser, is programmed to run at twice the FPGA clock input rate, although for low frequency operation this could be modified but this would probably also require modification of the FPGA code to suit the new clock ratios. There is a 125MHz lower limit on the operating frequency of the DLL, equivalent to a data rate of 250MSps. Lower clock rates may be used by disabling the DLL but are currently not supported. See needs a link :type:target:description thing here- this section Low Frequency Operationcauses a warning for more information.

Note that the output counters must be re-synchronised using the appropriate command following any re-programming of the clock circuit; this is built in to the example code.

For clock synthesis, the 1GHz VCXO is controlled by the synthesiser using a 100MHz internal reference. The clock distribution provides the capability for integer divisions of the VCXO frequency to be used as the DAC clock and the related FPFGA clocks. An external reference can also be used so that the VCXO (and therefore the DAC clock) are locked to this reference. The PLL provides only integer-N synthesis so only integer divisions of 1GHz can be used in this mode.Consult the factory for the availability of custom VCXO frequencies for applications requiring other frequencies.

The external signal source can be used in place of the VCXO as the clock driving the distribution section for the DAC so can be used as the DAC clock directly or integer divisions of this source can be used.The maximum input frequency is 1.2 GHz

See the AD9510 data sheet for further information.

2.7 DAC Selftest

The DAC has a number of self-test capabilities built in which are implemented in the example code.

2.7.1 Pattern testing

This consists of writing values of 0xAAAA and 0X5555 in succession. Error flags can be interrogated via the serial interface to determine the success or otherwise of the data transfers and thereby test the FPGA-DAC interface.

2.7.2 Fifo Test

The DAC provides the facility to check for FIFO overruns, which normally should not occur. For applications where low-level control of register bits has been implemented (e.g. FIFO_offset position), this provides confirmation of correct fifo operation.

2.7.3 Selftest

This runs an internal self-test algorithm which requires 400,000 DACCLK cycles, so completes in < 1ms for a 1GHz clock frequency. Pass/fail flags can be interrogated via the serial interface

2.8 DAC DLL Control

The DAC uses a DLL to align its input registers with DCLK and hence with the data. Any change in the DAC clock frequency (thus DCLK and FABRCLK) requires the DLL control bits in DAC register CONFIG10 to be set appropriately and the DLL re-aligned. This in turn requires the application of a hardware reset as part of the initialisation sequence so any existing settings will be lost. These settings must be explicitly re-written as part of the sequence.

The DLL control register (address 0xA) in the DAC has several different bit fields listed in the data sheet. Effectively these can be treated as a single bit field since the values to be used for each frequency are fixed (see "Electrical Characteristics" in the relevant device data sheet). Note that the frequency break points for the DAC5682Z changed from (200,300) and (300,500) to (200,325) and (325,500) in the March 09 data sheet.

DAC5681 DCLK (MHz)	DAC5681Z DCLK (MHz)	DAC5682Z DCLK (MHz)	Setting
125-150	125-150	125-150	0xCD
150-175	150-175	150-175	0xCE
175-200	175-200	175-200	0xCF
200-325	200-325	200-325	0xC8
325-500	325-500	325-500	0xC0

2.9 DAC Sync

The DAC configuration sequence requires a rising edge to be generated on the internal SYNC signal to instigate data output. This signal is normally controlled by the hardware pin (see register description below), but can be controlled via the serial interface. SYNC is set low prior to clock adjustment (and the FPGA-generated data forced to output a zero level) to minimise transients on the DAC outputs. The configuration sequence for the DAC does produce some transients however, a feature of the DAC itself rather than the board design.

2.10 Multiple DAC synchronisation

Multiple DAC synchronisation requires the use of the hardware SYNC pin; in addition the internal FIFOs of the DACs to be synchronised must also be aligned with each other. This is achieved by ensuring that bits D5 and D4 of DAC register CONFIG5 are initially cleared and the SYNC pin is pulsed high then low. Following this, D5 is set high and the drive to the DAC SYNC pin then set high. This synchronises the DAC outputs to within ± 1 DAC clock cycle. See the relevant device data sheet for further information.

2.11 Clocking on Virtex4, Virtex5

DCLK, SYNC and data must be produced synchronously for the DAC. At maximum speed, the data rate required is 500 MHz DDR. This is possible by using the 4:1 OSERDES components on Virtex4 and Virtex5 FPGAs.

On the XRC cards using Virtex4 and Virtex5 FPGAs, only regional clock inputs are available to XMC modules so global clocks must be generated using these inputs since the data input pins typically span multiple regions. There is an unknown (and unconstrainable) delay between the I/O pin and the BUFG input which must be taken into account when doing this.

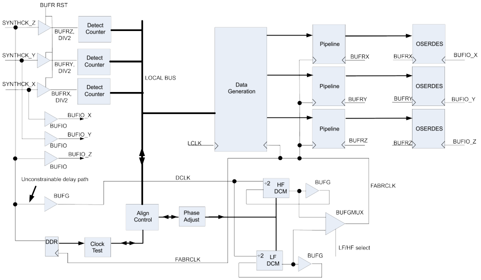


Figure 5 : Virtex 4 Virtex 5 Clocking Scheme

This delay is eliminated as shown in Figure Clocking Scheme. The clock input is fed to a pair of DCMs which generates the clock required by the OSERDES circuitry. Two DCMs are required because of the input and output clock restrictions of the DCM and the range of frequencies (125 MHz to 500MHz) which must be produced. Each DCM uses a divide-by-2 pre-scaler to produce FABRCLK at the correct rate.

The output side of the OSERDES is clocked using a copy of the FPGA clock, appropriate to the clock region occupied by the OSERDES. This is routed via a BUFIO to the OSERDES. The input side of the OSERDES is driven by the BUFR signal derived from the BUFIO clock to maintain timing alignment. The global clock signal which drives the data generation hardware (FABRCLK) runs at the same speed as this. The DCM aligns the global clock with these clocks and a constraint on the path lengths when crossing the clock domains ensures that data has the required setup and hold.

The BUFR's and BUFIO's required for each FPGA type are automatically instantiated by the clock mapping code; in most cases only two sets of BUFR's and BUFIO's are used.

Note that the clock(s) produced by the DCM do not drive the DAC directly, ensuring that any clock jitter added by the DCM is not transferred to the DAC.

Clock alignment is controlled by a few state machine components. The "ClockTest" block samples the image of the incoming clock signal via a DDR register over a number of cycles and compares the register output with the value expected for alignment, setting a pass or fail flag accordingly. The "PhaseAdjust" block shifts the DCM phase under the control of the "AlignControl" block.

When triggered via the local bus, the "AlignControl" block samples the pass/fail flag from "Clocktest" for the full range of phase shifts, determines the optimum setting and then implements this phase offset using "PhaseAdjust". Once completed, AlignControl signals back to the user application the result of the alignment operation.

This scheme also ensures that all clocks are constrained within the limits imposed by the various components in the FPGA for all speed grades of the FPGA. The DCM clock runs at half the rate of the DCLK rate. Hence for 1GHz DAC operation, the BUFIO clock (DCLK) runs at 500 MHz maximum whilst the DCM clock (FABRCLK) runs at 250 MHz maximum.

The BUFR clocks are used to provide diagnostic confirmation of the presence of the clock signals, in the same way as the spare clock signal noted above. In these cases the "divide-by-2" attribute must be set since the frequency limit for BUFR signals is 300 MHz maximum.

2.11.1 Low Frequency Operation

The above scheme is suitable for operation down to roughly 100 MHz. Below this frequency the use of the DCM becomes more problematic (e.g. lower frequency limit of 30 MHz). Interpolation techniques could be used to maintain the clock above the DCM limits whilst generating data at a much lower speed, but a simpler clocking scheme (shown in Figure Low frequency clocking scheme) can be used, which also has the benefit of simplifying the data generation requirements.

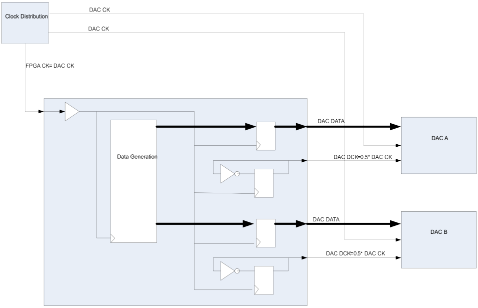


Figure 6 : Low frequency clocking scheme

Here the data generation in the FPGA runs at the same speed as the DAC sample clock fed to the DACs, so the FPGA generates 1 sample per DAC sample clock cycle instead of 4 samples at 0.25 *DAC sample clock cycle. This removes the need for OSERDES, DDR registers etc and simplifies the interface to the DAC and the data generation.

DCM alignment is no longer required since all data generation and output uses the same global clock. The unconstrained delay which results from using the clock-capable input appears as a simple phase offset, which is irrelevant since the DAC synchronises to the DCLK signal generated synchronously with the data via a toggling bit; data still changes on each edge of this clock. The DAC DLL Bypass bit is normally set when running in this mode since the operating speed is typically less than 125 MHz.

Clearly there is some overlap in the clock speed ranges which these two architectures can support which is also dependent on FPGA speed; the user should choose the one best suited to the application. Code for this style of operation is not included in the example code.

On Virtex6 and later boards, this restriction does not apply as there is no MMCM used.

2.12 Clocking on Virtex6, Kintex7 and Virtex7

On the ADMXRC cards using Virtex6, Kintex7 and Virtex7 FPGAs, clock distribution is simpler. Only one regional clock input is used as the clock source, selected to be the clock pair which can drive the clock regions above and the below the one directly clocked by the selected clock pair. This routed via a BUFMR and the BUFRs (configured to divide by 2) to clock these additional regions and the OSERDES components. The incoming clocks (at the half the DAC clock rate) are distributed via BUFIOs to drive the OSERDES components. The master BUFR clock is also routed through a global clock buffer to provide the clock for the data generation circuitry in the FPGA fabric.

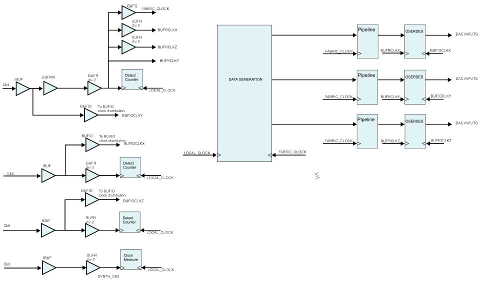


Figure 7 : Kintex 7 Virtex 7 Clocking Scheme

The remaining clock inputs are used for clock monitoring and diagnostic purposes only

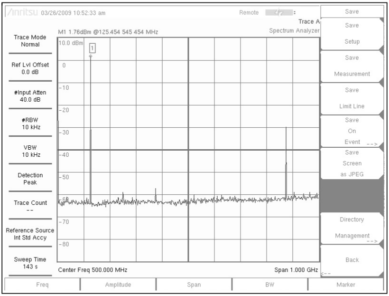
2.13 Data Generation

Each DAC receives data via a 16-bit DDR interface, plus DCLK, generated by the data source synchronously with the data. Clock speed restrictions in the FPGA force the use of OSERDES components in order to be able to run at the full rate (1G sample per second) This in turn means that the data generation circuitry must provide 4 consecutive data samples on each FABRCLK clock cycle.

In the example code this is implemented by instantiating four identical data sources for each type of waveform produced, each offset by the appropriate amount in order to provide the correct signal for each time slot.

2.14 Performance

Typical performance when producing a 125 MHz sine wave using the internal 1GHz clock is shown below. Note that the figure shows both the fundamental frequency (F_{fund}) and the image frequency (F_{sample}-F_{fund}) of the fundamental caused by sampling. The image frequency and higher components are normally filtered out by a low-pass filter which has a cut-off frequency F_{cutoff} <= F_{sample} /2, the midpoint of the figure.



Anritsu 03/26/2009 10:52:33 am

Remote

Trace A

Spectrum Analyzer

M1 1.76dBm @125.454 545 454 MHz

Trace Mode Normal

Ref Lvl Offset 0.0 dB

#Input Atten 40.0 dB

#RBW 10 kHz

VBW 10 kHz

Detection Peak

Trace Count --

Reference Source Int Std Accy

Sweep Time 143 s

Center Freq 500.000 MHz

Span 1.000 GHz

2.15 Board Layout

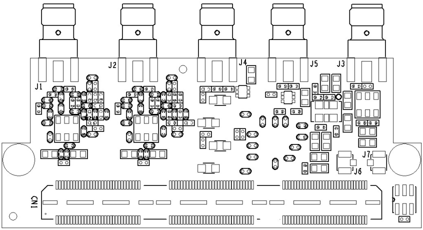


Figure 8 : XRM(2)-DAC-D4-1G Layout

Connector	Function	Signal Levels
J1	I channel DAC output	+/500 mV (+4 dBm)
J2	Q channel DAC output	+/500 mV (+4 dBm)
J3	Ext. Clock input	0dBm +6 dB
J4	AUX Dig I/0	3v3 LVTTL
J5	TRIG Dig I/O	3v3 LVTTL
J6	General Purpose Fast I/O, p side	FPGA I/O bank voltage
J7	General Purpose Fast I/O, n side	FPGA I/O bank voltage

Table 1 : SMA and UFL Connectors

Note that J6 and J7 connect directly to the FPGA pins and so use the same signalling voltage as the FPGA bank (1.8 volts or 2.5 volts). Extreme caution should be used when employing these connectors to avoid damage to the FPGA.

3 VHDL Structure

3.1 Introduction

The basic data flow is illustrated in figures 5 and 7 above. The clock components provide a clock for the output stage at the appropriate rate and the global clock for running the data generation circuitry. Each channel has its own data generation, DCLK and SYNC generation circuit under control of the host via the local bus interface.

3.2 Major HDL Components

Two top level design files are provided; one for use with ISE (xrm_dac_d4_1g_ise.vhd) and one for use with Vivado (xrm_dac_d4_1g.vhd). The ISE version includes ports for V4,V5,V6,V7 and K7 designs; unused top-level ports disappear in ISE during the process of synthesis, place-and-route and bit-file generation. In Vivado, unused top level ports generate an error. Functionally these are the same although the Vivado version also uses a wrapper component in order to simplify migration to other interface types (e.g. AXI). It is anticipated that future releases of ISE-style code will migrate to using the wrapper component. For the purposes of discussion, both styles of top-level component are implied by the term 'top level file'

The top level file contains a mixture of sequential statements and component instantiations. These implement the main blocks of code used in this design:

3.2.1 Clock generation and alignment

The DAC_Clocks component implements the clock generation and alignment code.

Restrictions on the maximum clock rates for the various buffers inside the FPGA require that the OSERDES output clock must be driven using a BUFR (or a BUFG). This in turn requires selection of the appropriate BUFR mapping based on the regional clocking capabilities of each FPGA, a function implemented in the dac_ck_map module.

The BUFRs are instantiated in the dac_ck_ip modules; there are three of these, one for each of the possible regional clock inputs available. The circuitry to generate the global clock is also contained in this module, with the generic parameter USE_AS_REF set TRUE on a single instance to instantiate a single instance of the global clock circuitry module, ck_align.vhd.

For Virtex4 and Virtex5 designs only, the ck_align module uses the dcm_align.vhd, ps_adj.vhd and align_test.vhd components plus it instantiates the DCMs appropriate to the FPGA being used. These three components implement the following tasks;

a) Sampling the input clock signal and generating a pass/fail flag with regard to the alignment of the global clock with the input clock (align_test.vhd).
b) Control of the phase setting of the DCM (ps_adj.vhd).
c) State machine to generate the handshake signals to the host, phase shift and alignment test modules, determine the optimum phase settings accordingly and then to apply this setting to align the clock (dcm_align.vhd).

Two DCMs must be used in each design, since the range of clock frequencies required spans the limits of the LF and HF modes of DCM operation. These operate with a pre-scaling divide by 2 counter to ensure that clocks are within the valid ranges for all speed grades of Virtex 4 and Virtex5. The appropriate DCM (LF/HF) is enabled by the application software, based on the clock frequency specified by the user, as part of the set-up sequence of the clock chain.

3.2.2 Data Generation and Output

The dac_channel.vhd component instantiates the data generation and output circuitry for each channel. All data generation is performed at the FABRCLK rate, with appropriate synchronisation of control signals from the local clock domain.

The FPGA data generation drives a 4:1 OSERDES, so must generate four samples of data for each cycle of FABRCLK. This is achieved by operating four identical data sources in parallel. For sine waves, four DDS generators are used, each offset in phase by the required amount to correctly interleave the data. The ramp generator produces four offset data streams in the same way and this signal is used as the basis of triangle and square wave generation. User-implemented data sources must implement the same mechanisms.

3.2.3 Local bus interface

3.2.3.1 Virtex4, Virtex5

The local bus interface consists of two components plus a small number of related processes and is based on the parallel-bus architecture on these FPGA cards.

Address decoding is performed here to enable read or write access by the host. This decoding is used to qualify read and write operations.

For writes, address-qualified write-enables are used within processes to implement the registers required to control the various elements in the design. For reads, address-qualified output-enables are used to enable retrieval of register settings and access to read-only registers.

PLXDSSM is a standard component from the Alpha-Data SDK which implements the interfacing and decoding required for the local bus signals.

XRC_CONFIG implements a number of the glue-logic functions associated with the local bus which require coding specific to the type of FPGA board being targeted. The latter is identified by including the appropriate configuration file in the project list.

3.2.3.2 Virtex6, Virtex7, Kintex7

Virtex6, Virtex7, Kintex7 boards use a serial-bus-based OCP interface for communications with the host via the bridge. The ocpbus_if.vhd component is a wrapper for adb3_ocp_simple_bus_if.vhd and the associated files from the SDK. This isolates the main XRM code from any SDK modifications by preserving a standard interface. This component also supplies the clock interfaces for the local bus and the 200 MHz reference clock. By default the local bus is run at 80 MHz although this can be run at up to 200 MHz if required.

Address decoding and register reads and writes are performed in much the same way as for the parallel local bus with a few noteworthy exceptions. Read decoding does not use an output enable signal and register addresses are aligned on 128-bit addresses instead of the 32-bit addresses used for the parallel bus. This is required in order to ensure that the bursting (four 32-bit reads or writes per access) inherent in the OCP bus operation does not cause inadvertent register accesses which might affect volatile data e.g. FIFOs.

3.2.4 Serial Control

The serial interface code (sampck_synth.vhd) is common to both synthesiser and DAC serial control and is implemented using a simple state machine which includes a programmable update clock rate to accommodate various bit-rates.

Read and write operations are initiated by generating a rising edge on the appropriate strobe port. An initialisation sequence can also be run from a single strobe signal.

The strobe signal triggers the transfer of data and address values via a state machine , with handshaking bits to synchronise with the host application.

For writes, the register address byte and write data byte specified on the external ports are used to form the serial stream sent to the device. For reads, only the address is used - any value present on the write port is ignored. At the end of the transfer process the data read from the devices is registered on the read data port at the end of the handshake sequence. This data is held until the next read sequence is triggered.

Initialisation sequences normally use only the address and data values specified in the initialisation table contained in sampck_synth_init.vhd. However, the configuration sequence for the DAC requires that part of this initialisation sequence can be varied in order to configure the DLL control bits correctly, so an additional port is used to provide this information for the instantiations driving the DACs; this port is ignored by the synthesiser instantiation which uses the same piece of VHDL code.

A further port is used specify the "device type" which controls the width of each field in the serial stream to suit the device being programmed. This must also be programmable in order to preserve code commonality, since the sequences for each of the DACs is slightly different and the field widths of the serial stream differ between the DAC and the synthesiser.

3.2.5 Digital I/O

The digital I/O pins use signalling levels of 3.3 volts and can be driven directly by the host from a dedicated register bit, one for each port, or be used as inputs. Ports can be individually configured as inputs or outputs. In addition the ARB waveform generator synchronisation marker can be output via these ports. All signals on these ports are clocked using the FABRCLK clock.

3.2.6 General Purpose I/O

The general purpose I/O pins, which are connected to J6 and J7, can be read or written as single-ended signals in the example code. Write bits are mapped to register bits with associated output enables. Direction defaults to inputs, with the state of the signal at the FPGA pins being read back in the same way as other signals.

These can also be configured to be used as a single differential signal input/output if required.

Suitable cableforms can be obtained from Samtec or Hirose director from a number of distributors (e.g. Farnell 168-8079, 168-8067).

3.2.7 Host Access via Local Bus

Data register addresses are defined in xrm_dac_d4_1g_pkg.vhd (and replicated in hwdefs.h for use by the C application) for both types of bus.Address decoding in the VHDL is divided into two parts - page addresses and register addresses within that space. The address ranges required to align register addresses on 128-bit or 32-bit boundaries as required by each card are defined in the appropriate 'xrc_build_pkg' library for the FPGA card being used. The *'xrc_build_pkg'* definitions also contain FPGA type definitions (e.g. *USE_XRC6_BOARD* etc.) which are used to control FPGA-specific build options within the VHDL code via generate statements.

All registers are accessed using offsets from the base address of the memory space, which is returned by software functions provided in the SDK. Data can be read and written via calls using dedicated functions in the SDK or by directly de-referencing the fpgaSpace pointer.

Additional SDK functions are used to configure the FPGA with the specific bit file, set various system clocks and pass command-line parameters.

The application also provides low-level test and diagnostic routines.

The PARLOCBUS_IF (Virtex4, Virtex5 and OCPBUS_IF (Virtex6, Virtex7, Kintex7) provide a standard interface for address decoding, irrespective of the type of local bus being implemented.

3.2.7.1 Virtex4, Virtex5

All registers are 32 bits wide; addresses A1 and A0 are unused, with the PLXDSSM component embedded in the PARLOCBUS_IF component doing most of the work involved in decoding local bus signals and generating the signals to ensure correct timing for reads and writes.

Address decoding uses address bits 22 to 19, with bit 23 zero and register addresses within each page are decoded using local address bits 9 to 2 for 32 bit accesses.

3.2.7.2 Virtex6, Virtex7, Kintex7

All registers are 32 bits wide but are 128-bit aligned, so addresses A1 to A3 inclusive are unused and register addresses within each page are decoded using local address bits 11 to 4.

The OCPBUS_IF component encapsulates the ADB3_OCP_SIMPLE_BUS_IF component which generates the interface and timing signals required for correct operation in the same manner as PLXDSSM above.

3.3 Waveform Generator Operation

The example code provides a number of built-in waveform generation options. In each case the fact that FABRCLK (which drives the FPGA fabric) runs at one quarter of the DAC sample clock frequency requires that four samples are produced by the generators on each FABRCLK cycle so typically four generators of each type, acting in parallel, are required.

A number of registers are shared across the various signal types to reduce the total numebr of registers required. Where applicable, this is indicated in the relevant register description in the register description section below, which aslo details the allocation of register bits.

The waveform output is selected using a total of six bits which control a number of cascaded 2:1 multiplexers. One bit is used to enable the data output to the DAC; if this bit is set to zero then all zeroes is transmitted. A second bit is used to select betwen the pattern test data sequence and the fixeed sine as the test waveform (see below).

In normal operation, the waveform output is selected from one of sine (DDS generated), ramp,triangle, pulse/ square or the arbitrary waveform, with the DDS sine being the default.

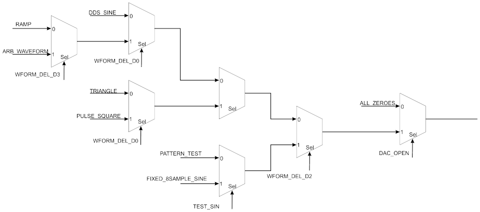


Figure 9 : Waveform Selection Diagram

The four waveform selection bits for each channel are located in the control register (D1,D28,D3,D2 for I channel, D5,D29,D7,D6 for Q channel).

3.3.1 Sine Waveform Generator

The main sine wave generator is based on a DDS core. Four cores are used in parallel to provide the four samples required on each FABRCLK cycle.

The signal frequency produced is determined by the value of the increment (rate of change of phase) specified for the phase accumulatorof each core. This phase accumulator is 32 bits wide hence 0x8000_0000 corresponds to a normalised frequency of 0.5. The actual frequency generated is given by:

$F_{out} = F_{dac} * F_{norm} = F_{dac} * \text{RegVal}/(2^{32})$

where F_{dac} = the DAC sample clock frequency (= 4 * FABRCLK frequency), F_{norm} is the normalised frequency and RegVal is the register setting.

Once the required register value *Val* is calculated as above, the actual increment applied to each core is *4*Val*, with the inital phase accumulator value for each core offset fromt the preceding one by *Val*. Thus core 0 produces the values for sample numbers 0,4,8 ..., core 1 produces the values for sample numbers 1,5,9 .., core 2 produces the values for sample numbers 2,6,10 .. and core 3 produces the values for sample numbers 3,7,11 .. etc.

In a similar fashion the starting phase of each channel (nominally 0 degrees) can be specified to produce quadrature, anti-phase or arbitrary phase offsets between the I and Q channels. The initial phase value is scaled in the same way as frequency increments

A value of 0x8000_0000 corresponds to a phase offset of 180 degrees. The initial phase offset is given by:

Offs = 360 degrees * RegVal/(2^{32})

where RegVal is the register setting

This offset is used to calculate and load the necessary offset into each core, with core 0 receiving an offset of *Offs*, core 1 receiving an offset of *2*Offs*,core 2 receiving an offset of *3*Offs* and core 3 receiving an offset of *4* Offs*.

Typically this offset is written to only one channel.

3.3.2 Ramp Waveform Generator

For simple ramp generation, the increment value *Inc* at the DAC sample clock frequency is calculated.

The increment *Inc* is determined by the normalised frequency according to the value:

Inc= 2^{16} * F_{norm}

since the DAC is a 16 bit device.

Four ramp accumulators are used in parallel, with each accumulator incremented by 4**Inc* and each offset from the preceding one by *Inc* in the same manner as ther sine wave generation.

Data simply rolls over, so non integer divisions of 2^16 will produce ramps with different starting points but the same slope on each cycle until the starting value is reached again

3.3.3 Triangle Waveform Generator

For triangle generation, 3 registers are used. The 16-bit start and end values, specified in a single 32-bit register, force the signal value at the zero-crossing points of the positive and negative ramp generation.

The 16-bit slope values (for positive and negative slopes) are used to increment the start and end values respectively and are specified in a single 32-bit register.

A third 32-bit register is used as a pair of 16-bit values specifying the number of increments applied for the positive and negative edges.

This ensures that consistent triangle waveforms are produced; note that the frequencies and slopes possible are constrained by the fact that four samples must be produced on each FABRCLK cycle **and** that waveforms must be repetitive.

The slope value is calculated based on the DAC sample clock.

Four ramp accumulators are used in parallel, with each accumulator incremented by 4*the slope value with each accumulator offset from the preceding one by the slope value in the same manner as ther sine wave generation.

3.3.4 Square/Pulse Waveform Generator

For square wave generation, the waveform output is set to the mark value for a given number of FABRCLK cycles and is then followed by the space value for separetely specified number of clock cycles.

One 32-bit register is used to specify the mark and space durations whilst a second is used to specify the mark and space levels.

Additional pulse width resolution is provided by four control bits in a separate register, which allow the mark and space durations to be adjusted in DAC smaple clock cycles to provide fine control of pulse/square waveforms.

3.3.5 Arbitrary Waveform Generator

ARB memory consists of 4k samples (16 bits wide) in each channel. ARB waveforms can be any length but should be zero-filled if not an exact multiple of 4 samples. An ARB sequence can be a single non-zero value, but the number of samples read out to the DAC is always a multiple of 4 (minimum 4 samples) because of the FABRCLK and DAC sample clock ratios. In the example code, data is automatically zero-filled to the end of the RAM.

Writing more data than the capacity of ARB memory will cause the write address to wrap around and thus over-write previously-written values (this is prevented from occurring in the example code routines).

The data writes are controlled by an 11-bit address counter, where each address references a pair of 16-bit samples. Data is written by sequential writes to the I_ARBWRITE_REG (and/or Q_ARBWRITE_REG) register as pairs of consecutive samples, each 16 bits wide.

Data pairs are written in time-order and each 32 bit write must have the earliest of the two samples in the low word and the latest sample in the upper word. The ARB busy bit pulses high during the process of writing to the ARB memory, but each write completes within the access time of the host doing the write thus eliminating any need for handshaking during this process.

The write address is auto-incremented by each write to the ARB ; the initial sample is written at address zero (the write address is reset to zero when the (FPGA) generator reset bit is pulsed high).

The length of the ARB sequence (4096 samples maximum) played out from the RAM is controlled by a programmable-length end address counter. The end address for playout is written into hardware by setting the appropriate last address value in the ARB control register and then pulsing the 'Load Enable' bit for that channel in the same register.

The 'Run Enable' bit is the global enable bit which must be set for the ARB to produce a non-zero signal on any channel.

The ARB can function in two modes - continuous or burst mode. The active operating mode is controlled by the value of the Continuous/Burst enable bit in the ARB control register; in both cases the 'Run Enable' bit must be set to obtain non-zero output.

In continuous mode, samples from the start of the RAM to the end address are played out, with the RAM address rolling over to 0 following the end address. This sample sequence is repeated continuously. Short burst-type waveforms can be created by setting the ARB length to include zero-padding samples stored in the RAM.

As an example, assume the signal is ***N*** samples of a sine wave followed by ***M*** samples of zero values. The ARB length, ***L***, is thus specified as ***(N+M)***, where ***(N+M)*** modulo 4 =0 and ***(N+M)*** <= 4096. This results in a burst of ***N*** sine samples followed by ***M*** zero samples, which repeats every ***(N+M)*** DAC sample clocks. With the same data set, specifying the ARB length ***L*** to be ***N*** (assuming ***N*** modulo 4 = 0), only the sine samples are continuously read out.

The limited length of the memory (4k samples), means that the maximum repetition rate at 1GHz DAC sample clock rate is limited to approximately 4 us.

Burst mode is used to extend the reptition rates achievable by means of a 16-bit count register ***C***, where ***C*** is the count of repetitions of the ARB length ***L*** (the RAM end address). When Burst mode is enabled, the ***L*** samples from the RAM are read out; these are then followed by ***{ (C-1)*L}*** DAC sample clocks of zero-value samples; this entire sequence is then repeated.

The ARB can also output a digital gate signal ("tick"), synchronous with the data read from RAM, for use with external hardware. The rising edge of this signal occurs at the start of the burst; the width can be specified to be a fixed number of FABRCLK cycles up to the length ***L*** of the signal in the RAM using the tick width register.

The TRIG and AUX hardware I/O ports can output this tick by first setting the multiplexer bit to select the tick signal as the drive source for the AUX or TRIG signal and also configuring the TRIG and/or AUX ports as outputs. The ARB tick for the I channel is available on the TRIG port whilst that for the Q channel is available on the AUX port

3.3.6 Self Test Pattern

The DAC provides a pattern testing mode where what is received on the data inputs is compared to the expected pattern of 0xAAAA/0x5555 and the result of this comparison soignalled via a status bit.

The FPGA must uses two 32-bit registers on each channel to implement this, with the two registers providing the four consecutive sample values required on each FABRCLK cycle.

Having two programmable registers allows the test pattern to be varied, allowing individual bits to be tested.

3.3.7 Sine Test

A simple 8-sample sine wave can be forced to be the signal source driving the DACs for test and diagnostic purposes using a single bit.

4 Register Description

All data accesses are 32 bits wide; for writes, unused bits are ignored whilst on reads unused bits are always mapped as a logic low.

A total of 30 registers are used, although not all bits are active controls.

Register Name	Address
CNTRL_REG	0X00
STATUS_REG	0X01
CNTR_STATUS_REG	0X02
I_DDS_REG	0X03
Q_DDS_REG	0X04
I_INC_REG	0X05
Q_INC_REG	0X06
SYNTH_CNTRL_REG	0X07
SYNTH_STRB_REG	0X08
IDAC_CNTRL_REG	0X09
IDAC_STRB_REG	0X0A
QDAC_CNTRL_REG	0X0B
QDAC_STRB_REG	0X0C
DEVICE_REG	0X0D
I_DDSINIT_REG	0X0E
Q_DDSINIT_REG	0X0F
IPATTERN_REG	0X10
QPATTERN_REG	0X11
IPATTERN_REG2	0X12
QPATTERN_REG2	0X13
MEAS0_VAL_REG	0X14
MEAS1_VAL_REG	0X15
MEAS2_VAL_REG	0X16
FREERUN_CNT_REG	0X17
I_ARBWRITE_REG	0X18
Q_ARBWRITE_REG	0X19
ARB _CNTRL_REG	0X1A
ARB _TICK_REG	0X1B
Not used	0X1C
Not used	0X1D
AUXCNTRL_REG	0X1E

Register Name	Address
PHASE_REG	0X1F

4.1 FPGA_CNTRL_REG (0x00)

The control register provides various bits for controlling timing and resets.

D31 to D24:	
D31	Clock align request, rising edge triggers alignment.
D30	Select HF DCM when low (default), LF DCM when high.
D29	Q channel signal select bit 2
D28	I channel signal select bit 2
D27	Unused.
D26	Sync enable for I DAC and QDAC.
D25	Clock mux control msb.
D24	Clock mux control lsb.
D23 to D16:	
D23	'GPIO_N' port direction signal, 1=output, 0=default=input.
D22	'GPIO_N 'port output signal.
D21	'GPIO_P' port direction signal, 1=output, 0=default=input.
D20	'GPIO_P' port output signal.
D19	'AUX' port direction signal, 1=output, 0=default=input.
D18	'AUX' port output signal.
D17	'TRIG' port direction signal, 1=output, 0=default=input.
D16	'TRIG' port output signal.
D15 to D8:	
D15	Alignment reset signal,1=reset active.
D14	Q channel FPGA data generator reset (1=reset active).
D13	I channel FPGA data generator reset (1=reset active).
D12	QDAC hardware reset, 1=reset active.
D11	IDAC hardware reset, 1=reset active.
D10	Determines DLL bypass bit in initialisation sequence (0=default=DLL enabled).
D9	VCXO oscillator enable (1 = oscillator running).
D8	100MHz reference oscillator enable (1 = oscillator running).
D7 to D0:	
D7	Q channel signal select bit 1
D6	Q channel signal select bit 0
D5	Q channel signal select bit 3, msb
D4	Q channel output enable; 0=all zeroes transmitted, 1=FPGA generator data transmitted to DAC.
D3	I channel signal select bit 1

D7 to D0:	
D2	I channel signal select bit 0,lsb
D1	I channel signal select bit 3, msb
D0	I channel output enable; 0 = all zeroes transmitted by FPGA, 1 = FPGA generator data transmitted to DAC.

Clock mux msb	Clock mux lsb	Synth sample clock	Synth reference clock
0	0	Extck input	Extck input
0	1	Extck input	Internal reference
1	0	Internal VCXO	Extck input
1	1	Internal VCXO	Internal reference

Table 2 : Clock Muxing (D25,D24)

4.2 FPGA_STATUS_REG (0x01)

The read-only status register provides read back of the various system status bits for control and diagnostic use.

D31 to D24:	
D31	XRM pcb revision setting, msb
D30	XRM pcb revision setting bit 2
D29	XRM pcb revision setting bit 1
D28	XRM pcb revision setting, lsb
D27	unused
D26	unused
D25	unused
D24	Q serial status flag
D23 to D16:	
D23	I serial status flag
D22	unused
D21	Q sync enable acknowledge
D20	I sync enable acknowledge
D19	unused
D18	unused
D17	Alignment fault detected if high following alignment sequence (meaningful for V4
D16	Alignment routine busy if high (meaningful for V4
D15 to D8:	
D15	DCM lock flag (1=locked
D14	DAC 3V3 regulator monitor bit, 1= power good.
D13	DAC 1V8 regulator monitor bit, 1= power good.
D12	Unused.
D11	GPIO_N (J7) input bit.
D10	GPIO_P (J6) input bit.
D9	External 'AUX' input signal.
D8	External 'TRIG' input signal.
D7 to D0:	
D7	Q channel signal generator busy (1 = busy).
D6	Q channel initialisation sequence flag (1 = init sequence running).
D5	Q channel DAC serial interface complete flag (1 = complete, set at end of transfer and cleared by start of next transfer).
D4	Q channel DAC serial interface status (1 = transfer in progress).
D3	I channel signal generator busy (1 = busy).

D7 to D0:	
D2	I channel initialisation sequence flag (1=init sequence running).
D1	I channel DAC serial interface complete flag (1=complete, set at end of transfer and cleared by start of next transfer).
D0	I channel DAC serial interface status (1 = transfer in progress).

4.3 CNTR_STAT_REG (0x02)

Data read from this read-only location returns the detection counter bits for each of the four clock inputs where instantiated in the FPGA.

D31 to D24:	
D31	unused
D30	unused
D29	unused
D28	unused
D27	unused
D26	unused
D25	unused
D24	unused
D23 to D16:	
D23	unused
D22	unused
D21	unused
D20	unused
D19	200MHz ref clock detect counter D3 msb
D18	200MHz ref clock detect counter D2
D17	200MHz ref clock detect counter D1
D16	200MHz ref clock detect counter D0 lsb
D15 to D8:	
D15	Test clock input, detect counter D3 msb
D14	Test clock input, detect counter D2
D13	Test clock input, detect counter D1
D12	Test clock input, detect counter D0, lsb
D11	Clock input Z, detect counter D3, msb
D10	Clock input Z, detect counter D2
D9	Clock input Z, detect counter D1
D8	Clock input Z, detect counter D0, lsb
D7 to D0:	
D7	Clock input Y, detect counter D3 msb
D6	Clock input Y, detect counter D2
D5	Clock input Y, detect counter D1
D4	Clock input Y, detect counter D0, lsb
D3	Clock input X, detect counter D3 msb

D7 to D0:	
D2	Clock input X, detect counter D2
D1	Clock input X, detect counter D1
D0	Clock input X, detect counter D0, lsb

4.4 I_DDS_REG (0x03)

Data written to this location sets the DDS phase accumulator increment to determine the output frequency based on the DAC sample clock frequency.

The actual output frequency is related to the signed 32-bit register value by:

$F_{out} = F_{dac} * F_{norm} = F_{dac} * RegVal/(2^{32})$

where F_{dac} = the DAC clocking frequency (= 2*DCLK=4 * FABRCLK frequency), F_{norm} is the normalised frequency and RegVal is the register setting.

D31 to D24:	
D31	DDS phase increment D31 msb
D30	DDS phase increment D30
D29	DDS phase increment D29
D28	DDS phase increment D28
D27	DDS phase increment D27
D26	DDS phase increment D26
D25	DDS phase increment D25
D24	DDS phase increment D24
D23 to D16:	
D23	DDS phase increment D23
D22	DDS phase increment D22
D21	DDS phase increment D21
D20	DDS phase increment D20
D19	DDS phase increment D19
D18	DDS phase increment D18
D17	DDS phase increment D17
D16	DDS phase increment D16
D15 to D8:	
D15	DDS phase increment D15
D14	DDS phase increment D14
D13	DDS phase increment D13
D12	DDS phase increment D12
D11	DDS phase increment D11
D10	DDS phase increment D10
D9	DDS phase increment D9
D8	DDS phase increment D8
D7 to D0:	
D7	DDS phase increment D7

D6	DDS phase increment D6
D5	DDS phase increment D5
D4	DDS phase increment D4
D3	DDS phase increment D3
D2	DDS phase increment D2
D1	DDS phase increment D1
D0	DDS phase increment D0, lsb

4.5 Q_DDS_REG (0x04)

Data written to this location sets the DDS phase accumulator increment to determine the output frequency based on the DAC sample clock frequency.

The actual output frequency is related to the signed 32-bit register value by:

$F_{out} = F_{dac} * F_{norm} = F_{dac} * RegVal/(2^{32})$

where F_{dac} = the DAC clocking frequency (= 2*DCLK=4 * FABRCLK frequency), F_{norm} is the normalised frequency and RegVal is the register setting.

D31 to D24:	
D31	DDS phase increment D31 msb
D30	DDS phase increment D30
D29	DDS phase increment D29
D28	DDS phase increment D28
D27	DDS phase increment D27
D26	DDS phase increment D26
D25	DDS phase increment D25
D24	DDS phase increment D24
D23 to D16:	
D23	DDS phase increment D23
D22	DDS phase increment D22
D21	DDS phase increment D21
D20	DDS phase increment D20
D19	DDS phase increment D19
D18	DDS phase increment D18
D17	DDS phase increment D17
D16	DDS phase increment D16
D15 to D8:	
D15	DDS phase increment D15
D14	DDS phase increment D14
D13	DDS phase increment D13
D12	DDS phase increment D12
D11	DDS phase increment D11
D10	DDS phase increment D10
D9	DDS phase increment D9
D8	DDS phase increment D8
D7 to D0:	
D7	DDS phase increment D7

D6	DDS phase increment D6
D5	DDS phase increment D5
D4	DDS phase increment D4
D3	DDS phase increment D3
D2	DDS phase increment D2
D1	DDS phase increment D1
D0	DDS phase increment D0, lsb

4.6 I_INC_REG (0x05)

Used for both simple ramp (sawtooth) and triangle generation.

Data written to this register sets the increment value for the ramp on each clock cycles. For ramp generation, the increment is determined by the normalised frequency according to the value:

Inc= $2^{16} * F_{norm}$

since the DAC is a 16 bit device.

For triangle waveforms, the upper 16 bits specify the start value for triangle generation on each positive slope at the zero crosssing point whilst the lower 16 bits specify the end value for triangle generation on each negative slope at the zero crosssing point.

D31 to D24:	
D31	Triangle start value D15, msb (zero crossing, positive slope)
D30	Triangle start value D14
D29	Triangle start value D13
D28	Triangle start value D12
D27	Triangle start value D11
D26	Triangle start value D10
D25	Triangle start value D9
D24	Triangle start value D8
D23 to D16:	
D23	Triangle start value D7
D22	Triangle start value D6
D21	Triangle start value D5
D20	Triangle start value D4
D19	Triangle start value D3
D18	Triangle start value D2
D17	Triangle start value D1
D16	Triangle start value D0, lsb
D15 to D8:	
D15	Ramp increment, Triangle end value D15 msb (zero crossing, negative slope)
D14	Ramp increment, Triangle end value D14
D13	Ramp increment, Triangle end value D13
D12	Ramp increment, Triangle end value D12
D11	Ramp increment, Triangle end value D11
D10	Ramp increment, Triangle end value D10
D9	Ramp increment, Triangle end value D9

D15 to D8:	
D8	Ramp increment, Triangle end value D8
D7 to D0:	
D7	Ramp increment, Triangle end value D7
D6	Ramp increment, Triangle end value D6
D5	Ramp increment, Triangle end value D5
D4	Ramp increment, Triangle end value D4
D3	Ramp increment, Triangle end value D3
D2	Ramp increment, Triangle end value D2
D1	Ramp increment, Triangle end value D1
D0	Ramp increment, Triangle end value D0, lsb

4.7 Q_INC_REG (0x06)

Used for both simple ramp (sawtooth) and triangle generation.

Data written to this register sets the increment value for the ramp on each clock cycle. The increment for ramp generation is determined by the normalised frequency according to the value:

Inc= $2^{16} * F_{norm}$

since the DAC is a 16 bit device.

For triangle waveforms, the upper 16 bits specify the start value for triangle generation on each positive slope at the zero crosssing point whilst the lower 16 bits specify the end value for triangle generation on each negative slope at the zero crosssing point.

D31 to D24:	
D31	Triangle start value D15, msb (zero crossing, positive slope)
D30	Triangle start value D14
D29	Triangle start value D13
D28	Triangle start value D12
D27	Triangle start value D11
D26	Triangle start value D10
D25	Triangle start value D9
D24	Triangle start value D8
D23 to D16:	
D23	Triangle start value D7
D22	Triangle start value D6
D21	Triangle start value D5
D20	Triangle start value D4
D19	Triangle start value D3
D18	Triangle start value D2
D17	Triangle start value D1
D16	Triangle start value D0, lsb
D15 to D8:	
D15	Ramp increment,Triangle end value D15 msb (zero crossing, negative slope)
D14	Ramp increment,Triangle end value D14
D13	Ramp increment,Triangle end value D13
D12	Ramp increment,Triangle end value D12
D11	Ramp increment,Triangle end value D11
D10	Ramp increment,Triangle end value D10
D9	Ramp increment,Triangle end value D9

D15 to D8:	
D8	Ramp increment,Triangle end value D8
D7 to D0:	
D7	Ramp increment,Triangle end value D7
D6	Ramp increment,Triangle end value D6
D5	Ramp increment,Triangle end value D5
D4	Ramp increment,Triangle end value D4
D3	Ramp increment,Triangle end value D3
D2	Ramp increment,Triangle end value D2
D1	Ramp increment,Triangle end value D1
D0	Ramp increment,Triangle end value D0, lsb

4.8 SYNTH_CNTRL_REG (0x07)

Used to specify the data and addresses for reads and writes via the serial interface for the AD9510 synthesiser.

The bottom 8 bits are the write data, with the next 8 bits forming the address for reads and writes. The top 16 bits of this word (read data and status bits) are read-only.

D31 to D24:	
D31	unused
D30	unused
D29	unused
D28	unused
D27	unused
D26	synth initialisation sequence busy. Copy of status reg bit 6
D25	synth serial busy signal. Copy of status reg bit 5
D24	synth serial busy signal. Copy of status reg bit 4
D23 to D16:	
D23	D7 serial read data from the synth interface following a read cycle (read-only).
D22	D6 serial read data
D21	D5 serial read data
D20	D4 serial read data
D19	D3 serial read data
D18	D2 serial read data
D17	D1 serial read data
D16	D0 serial read data
D15 to D8:	
D15	D7 serial address for serial read/writes of the synth interface.
D14	D6 serial address
D13	D5 serial address
D12	D4 serial address
D11	D3 serial address
D10	D2 serial address
D9	D1 serial address
D8	D0 serial address
D7 to D0:	
D7	D7 serial write data to write to the synth interface following a write strobe.
D6	D6 serial write data
D5	D5 serial write data

D7 to D0:	
D4	D4 serial write data
D3	D3 serial write data
D2	D2 serial write data
D1	D1 serial write data
D0	D0 serial write data

4.9 SYNTH_STRB_REG (0x08)

D31 to D24:	
D31	unused
D30	unused
D29	unused
D28	unused
D27	unused
D26	unused
D25	unused
D24	unused
D23 to D16:	
D23	unused
D22	unused
D21	unused
D20	unused
D19	unused
D18	unused
D17	unused
D16	unused
D15 to D8:	
D15	unused
D14	unused
D13	unused
D12	unused
D11	unused
D10	unused
D9	unused
D8	unused
D7 to D0:	
D7	Unused.
D6	Unused.
D5	Unused.
D4	Unused.
D3	Unused.
D2	Init strobe-setting this bit high triggers a synthesiser serial interface initialisation (write) sequence. This bit is self-clearing.

D7 to D0:	
D1	Write strobe-setting this bit high triggers a synthesiser serial interface write cycle using the values in the control register. This bit is self-clearing.
D0	Read strobe-setting this bit high triggers a synthesiser serial interface read cycle. This bit is self-clearing.

4.10 IDAC_CNTRL_REG (0x09)

Used to specify the data and addresses for reads and writes via the serial interface for the I channel DAC.

The bottom 8 bits are the write data, with the next 8 bits forming the address. The top 16 bits of this word (read data and status bits) are read-only.

D31 to D24:	
D31	unused
D30	unused
D29	unused
D28	unused
D27	unused
D26	IDAC serial initialisation sequence busy. Copy of status reg bit 6
D25	IDAC serial busy signal. Copy of status reg bit 5
D24	IDAC serial busy signal. Copy of status reg bit 4
D23 to D16:	
D23	D7 serial read data from the IDAC interface following a read cycle (read-only).
D22	D6 serial read data
D21	D5 serial read data
D20	D4 serial read data
D19	D3 serial read data
D18	D2 serial read data
D17	D1 serial read data
D16	D0 serial read data
D15 to D8:	
D15	D7 serial address for serial read/writes of the IDAC interface.
D14	D6 serial address
D13	D5 serial address
D12	D4 serial address
D11	D3 serial address
D10	D2 serial address
D9	D1 serial address
D8	D0 serial address
D7 to D0:	
D7	D7 serial write data to write to the IDAC interface following a write strobe.
D6	D6 serial write data
D5	D5 serial write data

D7 to D0:	
D4	D4 serial write data
D3	D3 serial write data
D2	D2 serial write data
D1	D1 serial write data
D0	D0 serial write data

4.11 IDAC_STRB_REG (0x0A)

Triggers serial reads, writes or forces the initialisation sequence to be active on the serial interface for the I channel DAC. Write only

D31 to D24:	
D31	Unused
D30	Unused
D29	Unused
D28	Unused
D27	Unused
D26	Unused
D25	Unused
D24	Unused
D23 to D16:	
D23	Unused.
D22	Unused.
D21	Unused.
D20	Unused.
D19	Unused.
D18	Unused.
D17	Unused.
D16	Unused.
D15 to D8:	
D15	Unused.
D14	Unused.
D13	Unused.
D12	Unused.
D11	Unused.
D10	Unused.
D9	Unused.
D8	Unused.
D7 to D0:	
D7	Unused.
D6	Unused.
D5	Unused.
D4	Unused.
D3	Unused.

D7 to D0:	
D2	Init strobe-setting this bit high triggers a IDAC serial interface initialisation (write) sequence. This bit is self-clearing.
D1	Write strobe-setting this bit high triggers a IDAC serial interface write cycle using the values in the control register. This bit is self-clearing.
D0	Read strobe-setting this bit high triggers a IDAC serial interface read cycle. This bit is self-clearing.

4.12 QDAC_CNTRL_REG (0x0B)

Used to specify the data and addresses for reads and writes via the serial interface for the Q channel DAC.

The bottom 8 bits are the write data, with the next 8 bits forming the address. The top 16 bits of this word (read data and status bits) are read-only.

D31 to D24:	
D31	unused
D30	unused
D29	unused
D28	unused
D27	unused
D26	QDAC serial initialisation sequence busy. Copy of status reg bit 6
D25	QDAC serial busy signal. Copy of status reg bit 5
D24	QDAC serial busy signal. Copy of status reg bit 4
D23 to D16:	
D23	D7 serial read data from the QDAC interface following a read cycle (read-only).
D22	D6 serial read data
D21	D5 serial read data
D20	D4 serial read data
D19	D3 serial read data
D18	D2 serial read data
D17	D1 serial read data
D16	D0 serial read data
D15 to D8:	
D15	D7 serial address for serial read/writes of the QDAC interface.
D14	D6 serial address
D13	D5 serial address
D12	D4 serial address
D11	D3 serial address
D10	D2 serial address
D9	D1 serial address
D8	D0 serial address
D7 to D0:	
D7	D7 serial write data to write to the QDAC interface following a write strobe.
D6	D6 serial write data
D5	D5 serial write data

D7 to D0:	
D4	D4 serial write data
D3	D3 serial write data
D2	D2 serial write data
D1	D1 serial write data
D0	D0 serial write data

4.13 QDAC_STRB_REG (0x0C)

Triggers serial reads, writes or forces the initialisation sequence to be active on the serial interface for the Q channel DAC. Write only

D31 to D24:	
D31	Unused
D30	Unused
D29	Unused
D28	Unused
D27	Unused
D26	Unused
D25	Unused
D24	Unused
D23 to D16:	
D23	Unused
D22	Unused
D21	Unused
D20	Unused
D19	Unused
D18	Unused
D17	Unused
D16	Unused
D15 to D8:	
D15	Unused
D14	Unused
D13	Unused
D12	Unused
D11	Unused
D10	Unused
D9	Unused
D8	Unused
D7 to D0:	
D7	Unused.
D6	Unused.
D5	Unused.
D4	Unused.
D3	Unused.

D7 to D0:	
D2	Init strobe-setting this bit high triggers a QDAC serial interface initialisation (write) sequence. This bit is self-clearing.
D1	Write strobe-setting this bit high triggers a QDAC serial interface write cycle using the values in the control register. This bit is self-clearing.
D0	Read strobe-setting this bit high triggers a QDAC serial interface read cycle. This bit is self-clearing.

4.14 DEVICE_REG (0x0D)

This register provides 4 separate bytes for controlling the initialisation sequence and the DLL setting (clock-frequency dependent) used in the DAC automatic initialisation sequence.

The top 16 bits (1 byte for each channel) are used to specify the DLL setting forced into the DAC serial initialisation stream to handle differences between different DACs.

The bottom 16 bits (1 byte for each channel) allow dynamic selection of the (device specific) DAC initialisation sequence if required.

D31 to D24:	
D31	QDACDLL configuration D7 (byte inserted into initialisation sequence triggered by strobe bit).
D30	QDACDLL configuration D6
D29	QDACDLL configuration D5
D28	QDACDLL configuration D4
D27	QDACDLL configuration D3
D26	QDACDLL configuration D2
D25	QDACDLL configuration D1
D24	QDACDLL configuration D0
D23 to D16:	
D23	IDACDLL configuration D7 (byte inserted into initialisation sequence triggered by strobe bit).
D22	IDACDLL configuration D6
D21	IDACDLL configuration D5
D20	IDACDLL configuration D4
D19	IDACDLL configuration D3
D18	IDACDLL configuration D2
D17	IDACDLL configuration D1
D16	IDACDLL configuration D0
D15 to D8:	
D15	Q channel DAC type code D7
D14	Q channel DAC type code D6
D13	Q channel DAC type code D5
D12	Q channel DAC type code D4
D11	Q channel DAC type code D3
D10	Q channel DAC type code D2
D9	Q channel DAC type code D1
D8	Q channel DAC type code D0
D7 to D0:	
D7	I channel DAC type code D7

D6	I channel DAC type code D6
D5	I channel DAC type code D5
D4	I channel DAC type code D4
D3	I channel DAC type code D3
D2	I channel DAC type code D2
D1	I channel DAC type code D1
D0	I channel DAC type code D0

Legal values for the DAC type bits (read via serial interface from DAC register CONFIG0\, bits D4-D2 inclusive) are:

DAC Type:	Device ID:	Device Code:
DAC5681	111	0x02
DAC5681Z	010	0x04
DAC5682Z	000	0x03

4.15 I_DDSINIT_REG(0x0E)

This register provides a mechanism to specify the phase offset of the I channel from that of the default 0 degrees when operating with the DDS signal (sine) selected. Data is scaled in the same way as frequency increments

The DDS generator has a 32-bit phase accumulator; a value of 0x8000_0000 corresponds to a phase offset of 180 degrees. The initial phase offset is given by:

Phase = 360 degrees * RegVal/(2^{32})

where RegVal is the register setting

D31 to D24:	
D31	D31 DDS phase msb
D30	D30 DDS phase bit
D29	D29 DDS phase bit
D28	D28 DDS phase bit
D27	D27 DDS phase bit
D26	D26 DDS phase bit
D25	D25 DDS phase bit
D24	D24 DDS phase bit
D23 to D16:	
D23	D23 DDS phase bit
D22	D22 DDS phase bit
D21	D21 DDS phase bit
D20	D20 DDS phase bit
D19	D19 DDS phase bit
D18	D18 DDS phase bit
D17	D17 DDS phase bit
D16	D16 DDS phase bit
D15 to D8:	
D15	D15 DDS phase bit
D14	D14 DDS phase bit
D13	D13 DDS phase bit
D12	D12 DDS phase bit
D11	D11 DDS phase bit
D10	D10 DDS phase bit
D9	D9 DDS phase bit
D8	D8 DDS phase bit
D7 to D0:	
D7	D7 DDS phase bit

D6	D6 DDS phase bit
D5	D5 DDS phase bit
D4	D4 DDS phase bit
D3	D3 DDS phase bit
D2	D2 DDS phase bit
D1	D1 DDS phase bit
D0	D0 DDS phase bit lsb

4.16 Q_DDSINIT_REG(0x0F)

This register provides a mechanism to specify the phase offset of the Q channel from that of the default 0 degrees when operating with the DDS signal (sine) selected. Data is scaled in the same way as frequency increments

The DDS generator has a 32-bit phase accumulator; a value of 0x8000_0000 corresponds to a phase offset of 180 degrees. The initial phase offset is given by:

Phase = 360 degrees * RegVal/(2^{32})

where RegVal is the register setting

D31 to D24:	
D31	D31 DDS phase msb
D30	D30 DDS phase bit
D29	D29 DDS phase bit
D28	D28 DDS phase bit
D27	D27 DDS phase bit
D26	D26 DDS phase bit
D25	D25 DDS phase bit
D24	D24 DDS phase bit
D23 to D16:	
D23	D23 DDS phase bit
D22	D22 DDS phase bit
D21	D21 DDS phase bit
D20	D20 DDS phase bit
D19	D19 DDS phase bit
D18	D18 DDS phase bit
D17	D17 DDS phase bit
D16	D16 DDS phase bit
D15 to D8:	
D15	D15 DDS phase bit
D14	D14 DDS phase bit
D13	D13 DDS phase bit
D12	D12 DDS phase bit
D11	D11 DDS phase bit
D10	D10 DDS phase bit
D9	D9 DDS phase bit
D8	D8 DDS phase bit
D7 to D0:	
D7	D7 DDS phase bit

D6	D6 DDS phase bit
D5	D5 DDS phase bit
D4	D4 DDS phase bit
D3	D3 DDS phase bit
D2	D2 DDS phase bit
D1	D1 DDS phase bit
D0	D0 DDS phase bit lsb

4.17 IPATTERN_REG (0x10)

This register provides two 16 bit values which are used to control data generation for triangle and pulse (square) waveforms and also serve to specify the data for self test operation on the I channel.

When generating triangle waveforms, bits D31 to D16 of this register determine the gradient (increment per DAC sample clock cycle) which controls the value applied to the DAC data pins for the negative slope. Bits D15 to D0 determine the gradient (increment per DAC sample clock cycle) which controls the value applied to the DAC data pins for the positive slope:

When generating pulse waveforms, bits D31 to D16 determine the value applied to the DAC pins for the 'space' duration and bits D15 to D0 determine the value applied to the DAC pins for the 'mark' duration in counts of FABRCLK cycles.Additional bits in the AUXCNTRL register extend this to give DAC sample clock resolution of pulse widths.

When generating the DAC pattern self test sequence, both this and the IPATTERN_REG2 register are programmed with 0xAAAA5555. These two registers are concatenated to specify the 4-sample sequence output during self-test operation. Other patterns may be used to check individual bits etc.

D31 to D24:	
D31	Triangle neg slope bit 15, Pulse Space level bit 15, msb
D30	Triangle neg slope bit 14, Pulse Space level bit 14
D29	Triangle neg slope bit 13, Pulse Space level bit 13
D28	Triangle neg slope bit 12, Pulse Space level bit 12
D27	Triangle neg slope bit 11, Pulse Space level bit 11
D26	Triangle neg slope bit 10, Pulse Space level bit 10
D25	Triangle neg slope bit 9, Pulse Space level bit 9
D24	Triangle neg slope bit 8, Pulse Space level bit 8
D23 to D16:	
D23	Triangle Neg slope bit 7, Pulse Space level bit 7
D22	Triangle Neg slope bit 6, Pulse Space level bit 6
D21	Triangle Neg slope bit 5, Pulse Space level bit 5
D20	Triangle Neg slope bit 4, Pulse Space level bit 4
D19	Triangle Neg slope bit 3, Pulse Space level bit 3
D18	Triangle Neg slope bit 2, Pulse Space level bit 2
D17	Triangle Neg slope bit 1, Pulse Space level bit 1
D16	Triangle Neg slope bit 0, Pulse Space level bit 0, lsb
D15 to D8:	
D15	Pos slope bit 15, Mark level bit 15, msb
D14	Pos slope bit 14, Mark level bit 14
D13	Pos slope bit 13, Mark level bit 13
D12	Pos slope bit 12, Mark level bit 12
D11	Pos slope bit 11, Mark level bit 11

D15 to D8:	
D10	Pos slope bit 10, Mark level bit 10
D9	Pos slope bit 9, Mark level bit 9
D8	Pos slope bit 8, Mark level bit 8
D7 to D0:	
D7	Mark level bit 7/ Pos slope bit 7
D6	Mark level bit 6/ Pos slope bit 6
D5	Mark level bit 5/ Pos slope bit 5
D4	Mark level bit 4/ Pos slope bit 4
D3	Mark level bit 3/ Pos slope bit 3
D2	Mark level bit 2/ Pos slope bit 2
D1	Mark level bit 1/ Pos slope bit 1
D0	Mark level bit 0, lsb

4.18 QPATTERN_REG (0x11)

This register provides two 16 bit values which are used to control data generation for triangle and pulse (square) waveforms and also serve to specify the data for self test operation on the Q channel.

When generating triangle waveforms, bits D31 to D16 of this register determine the gradient (increment per DAC sample clock cycle) which controls the value applied to the DAC data pins for the negative slope. Bits D15 to D0 determine the gradient (increment per DAC sample clock cycle) which controls the value applied to the DAC data pins for the positive slope:

When generating pulse waveforms, bits D31 to D16 determine the value applied to the DAC pins for the 'space' duration and bits D15 to D0 determine the value applied to the DAC pins for the 'mark' duration in counts of FABRCLK cycles.Additional bits in the AUXCNTRL register extend this to give DAC sample clock resolution of pulse widths.

When generating the DAC pattern self test sequence, both this and the QPATTERN_REG2 register are programmed with 0xAAAA5555. These two registers are concatenated to specify the 4-sample sequence output during self-test operation. Other patterns may be used to check individual bits etc.

D31 to D24:	
D31	Triangle neg slope bit 15, Pulse Space level bit 15, msb
D30	Triangle neg slope bit 14, Pulse Space level bit 14
D29	Triangle neg slope bit 13, Pulse Space level bit 13
D28	Triangle neg slope bit 12, Pulse Space level bit 12
D27	Triangle neg slope bit 11, Pulse Space level bit 11
D26	Triangle neg slope bit 10, Pulse Space level bit 10
D25	Triangle neg slope bit 9, Pulse Space level bit 9
D24	Triangle neg slope bit 8, Pulse Space level bit 8
D23 to D16:	
D23	Triangle Neg slope bit 7, Pulse Space level bit 7
D22	Triangle Neg slope bit 6, Pulse Space level bit 6
D21	Triangle Neg slope bit 5, Pulse Space level bit 5
D20	Triangle Neg slope bit 4, Pulse Space level bit 4
D19	Triangle Neg slope bit 3, Pulse Space level bit 3
D18	Triangle Neg slope bit 2, Pulse Space level bit 2
D17	Triangle Neg slope bit 1, Pulse Space level bit 1
D16	Triangle Neg slope bit 0, Pulse Space level bit 0, lsb
D15 to D8:	
D15	Pos slope bit 15, Mark level bit 15, msb
D14	Pos slope bit 14, Mark level bit 14
D13	Pos slope bit 13, Mark level bit 13
D12	Pos slope bit 12, Mark level bit 12
D11	Pos slope bit 11, Mark level bit 11

D15 to D8:	
D10	Pos slope bit 10, Mark level bit 10
D9	Pos slope bit 9, Mark level bit 9
D8	Pos slope bit 8, Mark level bit 8
D7 to D0:	
D7	Mark level bit 7/ Pos slope bit 7
D6	Mark level bit 6/ Pos slope bit 6
D5	Mark level bit 5/ Pos slope bit 5
D4	Mark level bit 4/ Pos slope bit 4
D3	Mark level bit 3/ Pos slope bit 3
D2	Mark level bit 2/ Pos slope bit 2
D1	Mark level bit 1/ Pos slope bit 1
D0	Mark level bit 0, lsb

4.19 IPATTERN_REG2 (0x12)

This register provides two 16 bit values which are used to control data generation for triangle and pulse (square) waveforms and also serve to specify the data for self test operation on the I channel.

When generating triangle waveforms, bits D31 to D16 determine determine the count of increments required for the negative slope and bits D15 to D0 determine the count of increments required for the positive slope. In both cases counts are in FABRCLK cycles.

When generating pulse waveforms, bits D31 to D16 determine the voltage level of the 'space' segment of the pulse whilst bits D15 to D0 determine the voltage level of the 'mark' segment of the pulse.

When generating the DAC pattern self test sequence, both this and the IPATTERN_REG register are programmed with 0xAAAA5555 to output the required sequence.

D31 to D24:	
D31	Neg slope bit 15,Space level bit 15 , msb
D30	Neg slope bit 14,Space level bit 14
D29	Neg slope bit 13,Space level bit 13
D28	Neg slope bit 12,Space level bit 12
D27	Neg slope bit 11,Space level bit 11
D26	Neg slope bit 10,Space level bit 10
D25	Neg slope bit 9,Space level bit 9
D24	Neg slope bit 8,Space level bit 8
D23 to D16:	
D23	Neg slope bit 7,Space level bit 7
D22	Neg slope bit 6,Space level bit 6
D21	Neg slope bit 5,Space level bit 5
D20	Neg slope bit 4,Space level bit 4
D19	Neg slope bit 3,Space level bit 3
D18	Neg slope bit 2,Space level bit 2
D17	Neg slope bit 1,Space level bit 1
D16	Neg slope bit 0,Space level bit 0 , lsb
D15 to D8:	
D15	Pos slope bit 15,Mark level bit 15 , msb
D14	Pos slope bit 14,Mark level bit 14
D13	Pos slope bit 13,Mark level bit 13
D12	Pos slope bit 12,Mark level bit 12
D11	Pos slope bit 11,Mark level bit 11
D10	Pos slope bit 10,Mark level bit 10
D9	Pos slope bit 9,Mark level bit 9
D8	Pos slope bit 8,Mark level bit 8

Note that pulse and triangle periods are constrained to be integer multiples of FABRCLK cycles.

4.20 QPATTERN_REG2 (0x13)

This register provides two 16 bit values which are used to control data generation for triangle and pulse (square) waveforms and also serve to specify the data for self test operation on the Q channel.

When generating triangle waveforms, bits D31 to D16 determine determine the count of increments required for the negative slope and bits D15 to D0 determine the count of increments required for the positive slope. In both cases counts are in FABRCLK cycles.

When generating pulse waveforms, bits D31 to D16 determine the voltage level of the 'space' segment of the pulse whilst bits D15 to D0 determine the voltage level of the 'mark' segment of the pulse.

When generating the DAC pattern self test sequence, both this and the QPATTERN_REG register are programmed with 0xAAAA5555 to output the required sequence.

D31 to D24:	
D31	Neg slope bit 15,Space level bit 15 , msb
D30	Neg slope bit 14,Space level bit 14
D29	Neg slope bit 13,Space level bit 13
D28	Neg slope bit 12,Space level bit 12
D27	Neg slope bit 11,Space level bit 11
D26	Neg slope bit 10,Space level bit 10
D25	Neg slope bit 9,Space level bit 9
D24	Neg slope bit 8,Space level bit 8
D23 to D16:	
D23	Neg slope bit 7,Space level bit 7
D22	Neg slope bit 6,Space level bit 6
D21	Neg slope bit 5,Space level bit 5
D20	Neg slope bit 4,Space level bit 4
D19	Neg slope bit 3,Space level bit 3
D18	Neg slope bit 2,Space level bit 2
D17	Neg slope bit 1,Space level bit 1
D16	Neg slope bit 0,Space level bit 0 , lsb
D15 to D8:	
D15	Pos slope bit 15,Mark level bit 15 , msb
D14	Pos slope bit 14,Mark level bit 14
D13	Pos slope bit 13,Mark level bit 13
D12	Pos slope bit 12,Mark level bit 12
D11	Pos slope bit 11,Mark level bit 11
D10	Pos slope bit 10,Mark level bit 10
D9	Pos slope bit 9,Mark level bit 9
D8	Pos slope bit 8,Mark level bit 8

Note that pulse and triangle periods are constrained to be integer multiples of FABRCLK cycles.

4.21 MEAS0_VAL_REG (0x14)

This register provides a 24 bit measurement (determined by CKMEAS_WIDTH) of the period of the unused clock input from the synthesiser for determining the frequency of the internal or external clock.

The count returned is based on a 10ms gate time (attribute gate_long= TRUE) or a 1ms gate time in the clock_meas component

Unused bits read as 0

D31 to D24:	
D31	N/A
D30	N/A
D29	N/A
D28	N/A
D27	N/A
D26	N/A
D25	N/A
D24	N/A
D23 to D16:	
D23	D23 gate-time multiplier MSB
D22	D22 gate-time multiplier
D21	D21 gate-time multiplier
D20	D20 gate-time multiplier
D19	D19 gate-time multiplier
D18	D18 gate-time multiplier
D17	D17 gate-time multiplier
D16	D16 gate-time multiplier
D15 to D8:	
D15	D15 gate-time multiplier
D14	D14 gate-time multiplier
D13	D13 gate-time multiplier
D12	D12 gate-time multiplier
D11	D11 gate-time multiplier
D10	D10 gate-time multiplier
D9	D9 gate-time multiplier
D8	D8 gate-time multiplier
D7 to D0:	
D7	D7 gate-time multiplier
D6	D6 gate-time multiplier

D7 to D0:	
D5	D5 gate-time multiplier
D4	D4 gate-time multiplier
D3	D3 gate-time multiplier
D2	D2 gate-time multiplier
D1	D1 gate-time multiplier
D0	D0 gate-time multiplier LSB

4.22 MEAS1_VAL_REG (0x15)

This register provides a 24 bit measurement (determined by CKMEAS_WIDTH) of the period of the transitions on the synthesiser STATUS signal for frequency measurements of the ref oscillator etc.

The count returned is based on a 10ms gate time (attribute gate_long= TRUE) or a 1ms gate time in the clock_meas component

Unused bits read as 0

D31 to D24:	
D31	N/A
D30	N/A
D29	N/A
D28	N/A
D27	N/A
D26	N/A
D25	N/A
D24	N/A
D23 to D16:	
D23	D23 gate-time multiplier MSB
D22	D22 gate-time multiplier
D21	D21 gate-time multiplier
D20	D20 gate-time multiplier
D19	D19 gate-time multiplier
D18	D18 gate-time multiplier
D17	D17 gate-time multiplier
D16	D16 gate-time multiplier
D15 to D8:	
D15	D15 gate-time multiplier
D14	D14 gate-time multiplier
D13	D13 gate-time multiplier
D12	D12 gate-time multiplier
D11	D11 gate-time multiplier
D10	D10 gate-time multiplier
D9	D9 gate-time multiplier
D8	D8 gate-time multiplier
D7 to D0:	
D7	D7 gate-time multiplier
D6	D6 gate-time multiplier

D7 to D0:	
D5	D5 gate-time multiplier
D4	D4 gate-time multiplier
D3	D3 gate-time multiplier
D2	D2 gate-time multiplier
D1	D1 gate-time multiplier
D0	D0 gate-time multiplier LSB

4.23 MEAS2_VAL_REG (0x16)

Address allocated for frequency measurements but unused

Unused bits read as 0

D31 to D24:	
D31	N/A
D30	N/A
D29	N/A
D28	N/A
D27	N/A
D26	N/A
D25	N/A
D24	N/A
D23 to D16:	
D23	N/A
D22	N/A
D21	N/A
D20	N/A
D19	N/A
D18	N/A
D17	N/A
D16	N/A
D15 to D8:	
D15	N/A
D14	N/A
D13	N/A
D12	N/A
D11	N/A
D10	N/A
D9	N/A
D8	N/A
D7 to D0:	
D7	N/A
D6	N/A
D5	N/A
D4	N/A
D3	N/A

D7 to D0:	
D2	N/A
D1	N/A
D0	N/A

4.24 FREERUN_CNT_REG (0x17)

This register is a free-running 32bit counter updated at 1 us intervals for timing use. All bits are read-only and are used by software routines for implementing time delays with 1 us resolution.

D31 to D24:	
D31	free-running counter MSB
D30	free-running counter
D29	free-running counter
D28	free-running counter
D27	free-running counter
D26	free-running counter
D25	free-running counter
D24	free-running counter
D23 to D16:	
D23	D23 free-running counter
D22	D22 free-running counter
D21	D21 free-running counter
D20	D20 free-running counter
D19	D19 free-running counter
D18	D18 free-running counter
D17	D17 free-running counter
D16	D16 free-running counter
D15 to D8:	
D15	D15 free-running counter
D14	D14 free-running counter
D13	D13 free-running counter
D12	D12 free-running counter
D11	D11 free-running counter
D10	D10 free-running counter
D9	D9 free-running counter
D8	D8 free-running counter
D7 to D0:	
D7	D7 free-running counter
D6	D6 free-running counter
D5	D5 free-running counter
D4	D4 free-running counter
D3	D3 free-running counter

D7 to D0:	
D2	D2 free-running counter
D1	D1 free-running counter
D0	D0 free-running counter LSB, changes at 1 us intervals

4.25 I_ARBWRITE_REG (0x18)

Writes to this register are copied into the IARB memory to form two 16 bit samples, with the earliest sample in the lower word.

D31 to D24:	
D31	bit D15 of 16-bit sample number N+1
D30	bit D14
D29	bit D13
D28	bit D12
D27	bit D11
D26	bit D10
D25	bit D9
D24	bit D8
D23 to D16:	
D23	bit D7 of 16-bit sample number N+1
D22	bit D6
D21	bit D5
D20	bit D4
D19	bit D3
D18	bit D2
D17	bit D1
D16	bit D0, LSB of 16-bit sample number N+1
D15 to D8:	
D15	bit D15 of 16-bit sample number N
D14	bit D14
D13	bit D13
D12	bit D12
D11	bit D11
D10	bit D10
D9	N/A
D8	N/A
D7 to D0:	
D7	bit D7 of 16-bit sample number N
D6	bit D6
D5	bit D5
D4	bit D4
D3	bit D3

D7 to D0:	
D2	bit D2
D1	bit D1
D0	D0, LSB of 16-bit sample number N

4.26 Q_ARBWRITE_REG (0x19)

Writes to this register are copied into the QARB memory to form two 16 bit samples, with the earliest sample in the lower word.

D31 to D24:	
D31	bit D15 of 16-bit sample number N+1
D30	bit D14
D29	bit D13
D28	bit D12
D27	bit D11
D26	bit D10
D25	bit D9
D24	bit D8
D23 to D16:	
D23	bit D7 of 16-bit sample number N+1
D22	bit D6
D21	bit D5
D20	bit D4
D19	bit D3
D18	bit D2
D17	bit D1
D16	D0, LSB of 16-bit sample number N+1
D15 to D8:	
D15	bit D15 of 16-bit sample number N
D14	bit D14
D13	bit D13
D12	bit D12
D11	bit D11
D10	bit D10
D9	N/A
D8	N/A
D7 to D0:	
D7	bit D7 of 16-bit sample number N
D6	bit D6
D5	bit D5
D4	bit D4
D3	bit D3

D7 to D0:	
D2	bit D2
D1	bit D1
D0	D0, LSB of 16-bit sample number N

4.27 ARB _CNTRL_REG (0x1A)

This register controls loading, running and synchronisation of the arbitrary waveform generators.

D31 to D24:	
D31	N/A
D30	N/A
D29	N/A
D28	N/A
D27	N/A
D26	MSB of ARB sequence length
D25	D9 ARB sequence length
D24	D8 ARB sequence length
D23 to D16:	
D23	D7 ARB sequence length
D22	D6 ARB sequence length
D21	D5 ARB sequence length
D20	D4 ARB sequence length
D19	D3 ARB sequence length
D18	D2 ARB sequence length
D17	D1 ARB sequence length
D16	LSB of ARB sequence length
D15 to D8:	
D15	QARB tick to AUX port driver mux, 1= tick as driver else trigger signal from control register.
D14	N/A
D13	N/A
D12	N/A
D11	N/A
D10	Load enable for QARB length value
D9	Burst/continuous select for QARB sequence
D8	Run enable for QARB sequence
D7 to D0:	
D7	IARB tick to TRIG port driver mux, 1= tick as driver else trigger signal from control register.
D6	N/A
D5	N/A
D4	N/A
D3	N/A

D7 to D0:	
D2	Load enable for IARB length value
D1	Burst/continuous select for IARB sequence
D0	Run enable for IARB sequence

4.28 ARB _TICK_REG (0x1B)

This register provides two 16 bit values for use with the ARB generator. The top 16 bits determine the marker width in FABRCLK cycles, although this is limites to 4096 (12 bits) max by the hardware.The bottom 16 bits determine the repetition rate of the ARB waveform, which consists of 1 cycle of data from the ARB RAM followed by ***(N-1)*** cycles of zero output, each the same length as the ARB RAM waveform.

D31 to D24:	
D31	ARB sequence marker width MSB
D30	D14 marker width
D29	D13 marker width
D28	D12 marker width
D27	D11 marker width
D26	D10 marker width
D25	D9 marker width
D24	D8 marker width
D23 to D16:	
D23	D7 marker width
D22	D6 marker width
D21	D5 marker width
D20	D4 marker width
D19	D3 marker width
D18	D2 marker width
D17	D1 marker width
D16	ARB sequence marker width LSB
D15 to D8:	
D15	MSB of 16-bit repeat rate control for ARB sequence
D14	D14 repeat rate control
D13	D13 repeat rate control
D12	D12 repeat rate control
D11	D11 repeat rate control
D10	D10 repeat rate control
D9	D9 repeat rate control
D8	D8 repeat rate control
D7 to D0:	
D7	D7 repeat rate control
D6	D6 repeat rate control
D5	D5 repeat rate control
D4	D4 repeat rate control

D7 to D0:	
D3	D3 repeat rate control
D2	D2 repeat rate control
D1	D1 repeat rate control
D0	LSB of 16-bit repeat rate control for ARB sequence.

4.29 AUXCNTRL_REG (0x1E)

Miscellaneous control bits and mark/space duration fine control for pulse waveforms.

D31 to D24:	
D31	Unused
D30	Unused
D29	Unused
D28	Unused
D27	Unused
D26	Unused
D25	Unused
D24	Unused
D23 to D16:	
D23	Unused
D22	Q channel mark duration fine control, bit 2, msb.
D21	Q channel mark duration fine control, bit 1.
D20	Q channel mark duration fine control, bit 0, lsb.
D19	Unused.
D18	I channel mark duration fine control, bit 2, msb.
D17	I channel mark duration fine control, bit 1.
D16	I channel mark duration fine control, bit 0, lsb.
D15 to D8:	
D15	Unused
D14	Unused
D13	Unused
D12	Unused
D11	Unused
D10	Unused
D9	Unused
D8	Unused
D7 to D0:	
D7	Unused
D6	Unused
D5	Unused
D4	Active-high asynchronous BUFR reset
D3	Software control of synth FUNC bit
D2	Unused

D7 to D0:	
D1	XRM board revision (high for rev2 and rev1 boards polarity correction - redundant)
D0	Phase value mux

The fine control bits for the I and Q channels allow extension of the mark duration in increments of the DACCLK period. Normally durations can only be stepped in multiples of the FABRCLK period. Setting a value other than zero modifies the data produced for the last space duration to output the mark level earlier than the 4-clock boundary. Setting a value greater than 4 effectively reduces the duration of the mark. The use of eight states allows the waveforms to maintain synchronism at the rising edge of the mark state even if differing mark/space ratios are programmed.

Note that pulse periods are still constrained to be multiples of FABRCLK cycles

Fine Adjust:	Phase1:	Phase2:
0	4 spaces	0 mark
1	3 spaces	1 marks
2	2 spaces	2 marks
3	1 spaces	3 marks
4	0 spaces	4 marks
5	1 mark	3 spaces
6	2 marks	2 spaces
7	3 marks	1 space

D1 (XRM board revision)-setting this bit high routes DAC data with polarities for rev2 boards; setting this bit low (the default) routes DAC data with polarities for rev3 boards. This should always be 0 unless a rev2 board is used.

D0 (Phase value mux) -the default value (0) selects the mid-point/width values of the valid phase window for readback. A value of selects the start-point/end-point values of the valid phase window for readback.

Sync bit (for synth) -defaults to high since default function is active-low reset of the synth.

4.30 PHASE_VALUE_REG (0x1F)

This register provides two 16 bit values which are sign-extended versions of the 9 bit phase alignment settings. By default. the values returned are the mid-point of the phase window (lower 16 bits) and the width of the window (upper 16 bits), which is centred on the mid-point value. Setting bit 0 of the AUXCNTRL_REG above switches this to provide the start point of the valid phase window (lower 16 bits) and the end point of the valid phase window (upper 16 bits).

This value is valid only for Virtex4 and Virtex5 designs. For Virtex6 and later, this register always returns a fixed value as no phase alignment is required

D31 to D24:	
D31	sign bit copy
D30	sign bit copy
D29	sign bit copy
D28	sign bit copy
D27	sign bit copy
D26	sign bit copy
D25	sign bit copy
D24	D8 (sign and magnitude) of width (end point) of valid DCM phase window
D23 to D16:	
D23	D7 of width (end point) of valid DCM phase window
D22	D6 of width (end point) of valid DCM phase window
D21	D5 of width (end point) of valid DCM phase window
D20	D4 of width (end point) of valid DCM phase window
D19	D3 of width (end point) of valid DCM phase window
D18	D2 of width (end point) of valid DCM phase window
D17	D1 of width (end point) of valid DCM phase window
D16	D0 of width (end hoint) of valid DCM phase window
D15 to D8:	
D15	sign bit copy
D14	sign bit copy
D13	sign bit copy
D12	sign bit copy
D11	sign bit copy
D10	sign bit copy
D9	sign bit copy
D8	D8 (sign and magnitude) of mid-point (start point) of valid DCM phase window
D7 to D0:	
D7	D7 mid point (start point) of valid DCM phase window

D6	D6 mid point (start point) of valid DCM phase window
D5	D5 mid point (start point) of valid DCM phase window
D4	D4 mid point (start point) of valid DCM phase window
D3	D3 mid point (start point) of valid DCM phase window
D2	D2 mid point (start point) of valid DCM phase window
D1	D1 mid point (start point) of valid DCM phase window
D0	D0 mid point (start point) of valid DCM phase window

Revision History

Date	Revision	Nature of Change
26/02/09	-	Draft
03/04/09	1.0	First issue with release 1.2 of code
12/08/09	1.1	Added signalling voltage, alternative clocking scheme and updated register bits
14/09/09	1.2	Fixed minor typos.
09/06/10	1.3	Updated to reflect code changes for release 2.0 renamed as User Guide.
15/06/10	1.4	Initial XML version of document
22/11/17	2.0	Added Virtex7, Kintex7 and Vivado information. Added ARB ram signal generation and programmable sine phase descriptions. Fixed various typos and added waveform generation description.
08/12/17	2.1	Fixed typos and updated waveform generation description.
08/03/18	2.2	Modified to be XRM/XRM2 manual

Address:	4 West Silvermills Lane, Edinburgh, EH3 5BD, UK	Address:	611 Corporate Circle Suite H Golden, CO 80401
Telephone:	+44 131 558 2600	Telephone:	(303) 954 8768
Fax:	+44 131 558 2700	Fax:	(866) 820 9956 - toll free
email:	sales@alpha-data.com	email:	sales@alpha-data.com
website:	http://www.alpha-data.com	website:	http://www.alpha-data.com