FMC216

Reference Firmware For VC707

Revision History

Date	Revision	Revision
2015-9-03	Initial release	1.0

4DSP LLC, USA.

Email: support@4dsp.com

Table of Contents

1 Introduction

This document describes the reference firmware for the FMC216 daughter card connected to FMC1 HPC of a VC707 carrier board. The hardware topology is depicted in Figure 1.

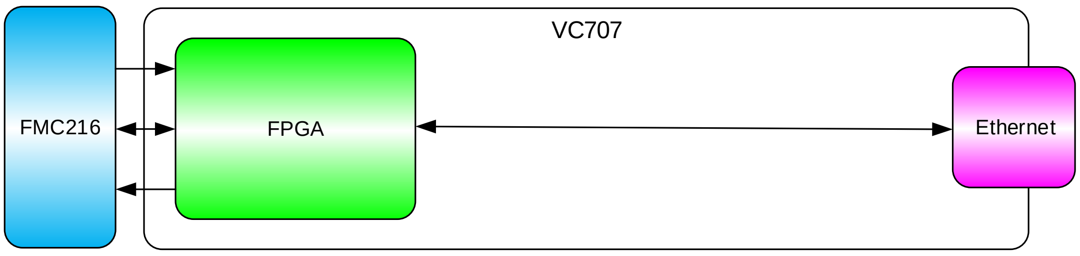

Figure 1: Hardware topology VC707 with FMC216

This document describes the firmware for the FPGA on the VC707, which is a Virtex-7 device. This design flow is based on 4DSP's Stellar IP flow. A description of the Stellar IP flow is beyond the scope of this document, but can be found in Stellar IP user manual. In order to read this document it is important to remember the following terminology:

-	**Star**	A functional block with a specific task.
-	**Wormhole**	A connection between two stars. A wormhole comprises of one or more signals in either one or both directions.
-	**Constellation**	A collection of stars that forms the top level of a firmware design.

This document only gives a global description of each star. Detailed information can be found in the Star Specification Documents available per star.

2 Constellation Design Description

A top level of the constellation is depicted in Figure 2. The constellation offers the following functionality:

- Load data into the DAC waveform memories (over Ethernet)
- Transfer data to the DACs using the JESD204B protocol

The constellation has the following stars:

1) Extended Constellation ID Star	(sip_cid_ex)
2) MAC Engine Star	(sip_vc707_mac_engine_sgmii)
3) FMC216 8 Lane Star	(sip_fmc216_8lane)
4) I^{2}C Master Star	(sip_i2c_master)
5) 1-to-16 Router Star	(sip_router_s1d16)
6) Command Multiplexer Star	(sip_cmd12_mux)

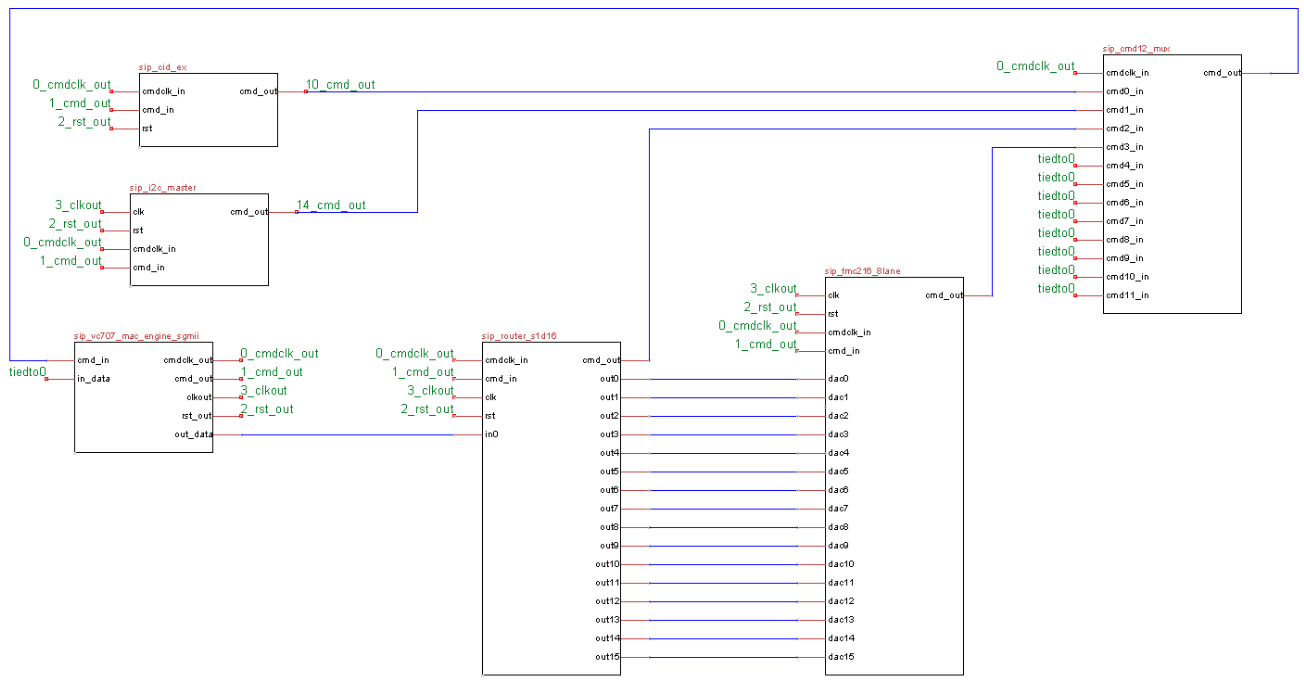

Figure 2: FPGA constellation block diagram.

2.1 Extended Constellation ID Star

The constellation ID star holds information about the constellation (constellation ID, star ID's, star address ranges, etcetera). The memory is filled by the Stellar IP tool and can be accessed by register read cycles.

2.2 MAC Engine Star

The MAC engine star distributes register read and register write commands coming from the Ethernet MAC. In addition the star transfers data to/from the host through the Ethernet MAC.

This star also generates and distributes the clock and reset signals for the other stars in the constellation.

2.3 FMC216 8 Lane Star

The FMC216 8 Lane star takes care of communication with the FMC216 daughter card using 8 transceiver lanes. Furthermore it sends data to the D/A converter and provides sampling control (burst size, etc.). Since the high bandwidth requirement the FMC216 star has an internal sample FIFO in in the internal waveform memory (WFM) in the D/A path.

2.4 I^{2}C Master Star

The I^{2}C master star acts as master on the I^{2}C bus which is connected to the voltage and temperature monitoring circuit on the FMC216 daughter card.

2.5 1-to-16 Router Star

The 1-to-16 router star routes data from its input port to one of the sixteen output ports.

2.6 Command Multiplexer Star

The command multiplexer star merges the command output wormholes of all stars and connects a single command output wormhole to the MAC Engine Star. This star does not require to be controlled by the user.

3 Limitations

The FMC216 has four DAC38J84 ICs, each having 4 DACs for a total 16 channels. The firmware is configured for JESD204B L=2, M=4,F=4, S=1,HD=0. The VC707 has a -2 Virtex-7 FPGA with a limit on it's tranceivers to 10.312 Gbps. Two transceiver lanes are used for each DAC IC giving a total of 257.8125 samples per second per converter maximum data we can send.

4 Unit System Support

The Vivado version of this constellation supports software working with the Unit API and is completely backwards compatible to legacy software. Each star that has a SIP command interface is assigned with a Unit Number (sip_cid_ex is an exception) which can automatically be detected by the Unit API. The design units are listed in Table 1.

Star	Unit number
Host Interface Star	0
I2C Master	4
FMC216	1

Table 1: Design Units

The design defines sixteen datapaths that will also be detected by the Unit API. The datapaths are described in Table 2.

Datapath	Starting point	Destination	Direction (used by software)
0	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC0)	Output
1	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC1)	Output
2	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC2)	Output
3	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC3)	Output
4	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC4)	Output
5	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC5)	Output
6	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC6)	Output
7	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC7)	Output
8	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC8)	Output
9	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC9)	Output
10	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC10)	Output
11	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC11)	Output
12	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC12)	Output
13	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC13)	Output
14	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC14)	Output
15	VP780_HOST_IF, Unit 0	FMC216, Unit 1 (DAC15)	Output

Table 2: Datapath descriptions

5 Register map

The register map is automatically generated by the Stellar IP tools. After running the Stellar-IP flow a file called “cid_package.vhd” can be found in the output directory.

6 Simulating the Reference Firmware

The reference firmware comes with an XSIM test bench that uses a behavioural model for VC707 host interface.

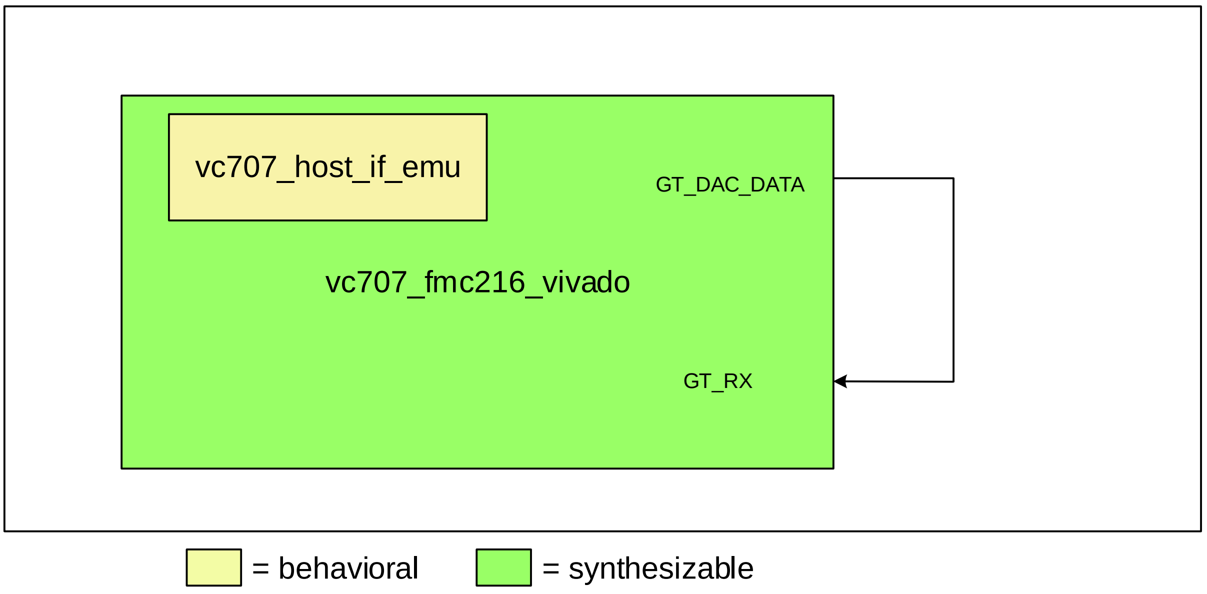

Figure 3: Test bench top level architecture

The VC707 host interface model emulates the communication through the Ethernet interface (register access, data transfer). This behavioural model reads instructions from a Stellar IP simulation script called “sip script”. Refer to section 6.3 for detailed information about the sip script language.

6.1 Running the simulation

In order to setup the simulation, perform the following steps:

- Start with the creation of the Vivado project. Refer to the StellarIP section in the 4FM Getting Started Guide.
- Use Vivado to open the project from
  - “\527_vc707_fmc216\output\vc707_fmc216_vivado\”.
- From the Tcl Console (View → Panels → Tcl Console) run the following command:
  - source ../../../simulate/xsim/xsim.tcl
- In the Flow Navigator click “Run Simulation” or from the Flow menu select “Run simulation → Run Behavioral Simulation.
- XSIM will launch and the simulation can be run.

6.2 What is simulated?

The reference design comes with a Stellar IP simulation script (section 6.3) that uses most of the available features. The sip script starts with waiting 4 us (power-up reset time) and then:

- Reset the transceiver
- Configure the S1D16 router

- Configure DACs
- Push data to the waveform repeat block for each DAC
- Enable waveform repeat for each DAC
- Repeat

6.3 Stellar IP simulation script (sip script)

The sip script is and has to be located in the simulation root folder with the file name “sip_cmd.sip”. The behavioural model parses the sip script line-by-line supporting the following instructions:

- REG_SET
- REG_SET_ACK
- REG_GET
- DMAPUSH
- DMAPULL
- WAIT

A semicolon (;) can be used as **first** character on a line to add comments to the script or to prevent a line from execution. Blank lines are allowed.

6.3.1 REG_SET

The REG_SET instruction performs a register write access to the constellation. Two arguments are required; the register address (1st) and the register data (2nd), for example:

REG_SET = 0x00000006 0x01234567

The arguments must be hexadecimal notation starting with 0x followed by 8 ASCII characters (0-9|A-F). It is required to keep the equal sign and spaces as in the example.

6.3.1 REG_SET_ACK

The REG_SET_ACK instruction performs a register write access to the constellation and waits until it receives a write acknowledge from the target in the constellation. Two arguments are required; the register address (1st) and the register data (2nd), for example:

REG_SET_ACK = 0x00000006 0x01234567

The arguments must be in hexadecimal notation starting with 0x followed by 8 ASCII characters (0-9|A-F). It is required to keep the equal sign and spaces as in the example.

The REG_SET_ACK instruction is blocking, which means that the next instruction is not executed until a write acknowledge response has been received from the constellation.

6.3.2 REG_GET

The REG_GET instruction performs a register read access to the constellation. One argument is required; the register address, for example:

REG_SET = 0x00000006

The argument must be hexadecimal notation starting with 0x followed by 8 ASCII characters (0-9|A-F). It is required to keep the equal sign and spaces as in the example.

The REG_GET instruction is blocking, which means that the next instruction is not executed until a response has been received from the constellation.

6.3.3 DMAPUSH

The DMAPUSH instruction performs a block write access to the constellation. One argument is required; the block size in bytes, which must be a multiple of 8, for example:

DMAPUSH = 0x00000400

The argument must be hexadecimal notation starting with 0x followed by 8 ASCII characters (0-9|A-F). It is required to keep the equal sign and spaces as in the example.

The data transferred to the constellation is read from the file "input.txt" located in the simulation root directory. This is a text file with one 64-bit value per line, hexadecimal notation using 16 ASCII characters (0-9|A-F).

The DMAPUSH instruction is blocking, which means that the next instruction is not executed until all bytes has been transferred to the constellation.

6.3.4 DMAPULL

The DMAPULL instruction performs a block read access to the constellation. One argument is required; the block size in bytes, which must be a multiple of 8, for example:

DMAPULL = 0x00000400

The argument must be hexadecimal notation starting with 0x followed by 8 ASCII characters (0-9|A-F). It is required to keep the equal sign and spaces as in the example.

The data transferred from the constellation is written to the file "output.txt" located in the simulation root directory. This is a text file with one 64-bit value per line, hexadecimal notation using 16 ASCII characters (0-9|A-F).

The DMAPULL instruction is blocking, which means that the next instruction is not executed until all bytes has been transferred from the constellation.

6.3.5 WAIT

In many cases the constellation will not be ready for access until the resets fall off and the clocks are stable. Therefore the sip script supports the WAIT instruction. One argument is required; the number of microseconds to wait, for example:

WAIT = 250

The argument must be integer notation. It is required to keep the equal sign and spaces as in the example.

7 Implementing the Reference Firmware

When implementation is going to take place, constraints become an essential requirement for a proper result. The most important constraints are those that indicate where to tie the input and outputs top entity's ports into the target FPGA. Also for a good implementation timing constraints are important.

When a reference design is created in StellarIP the XDC files will be also important at the moment of generating the project. Constraints files are required when the star has an external wormhole property, for example the start sip_fmc216_8lane has I/O ports associated to it.

During generation StellarIP will search for all the XDC files to merge them into one single XDC file. This file, by default is set to implementation only which means the constraints will be taken into account during implementation.

After the implementation is satisfactory, (meaning when all errors are solve and all warnings are handled) the programming file can be generated. The resulting file <project_name>.bit extension and can be found in:

Vivado:

\527_vc707_fmc216\output\vc707_fmc216_vivado\vivado\vc707_fmc216_vivado.runs\impl_1\ vc707_fmc216_vivado.bit