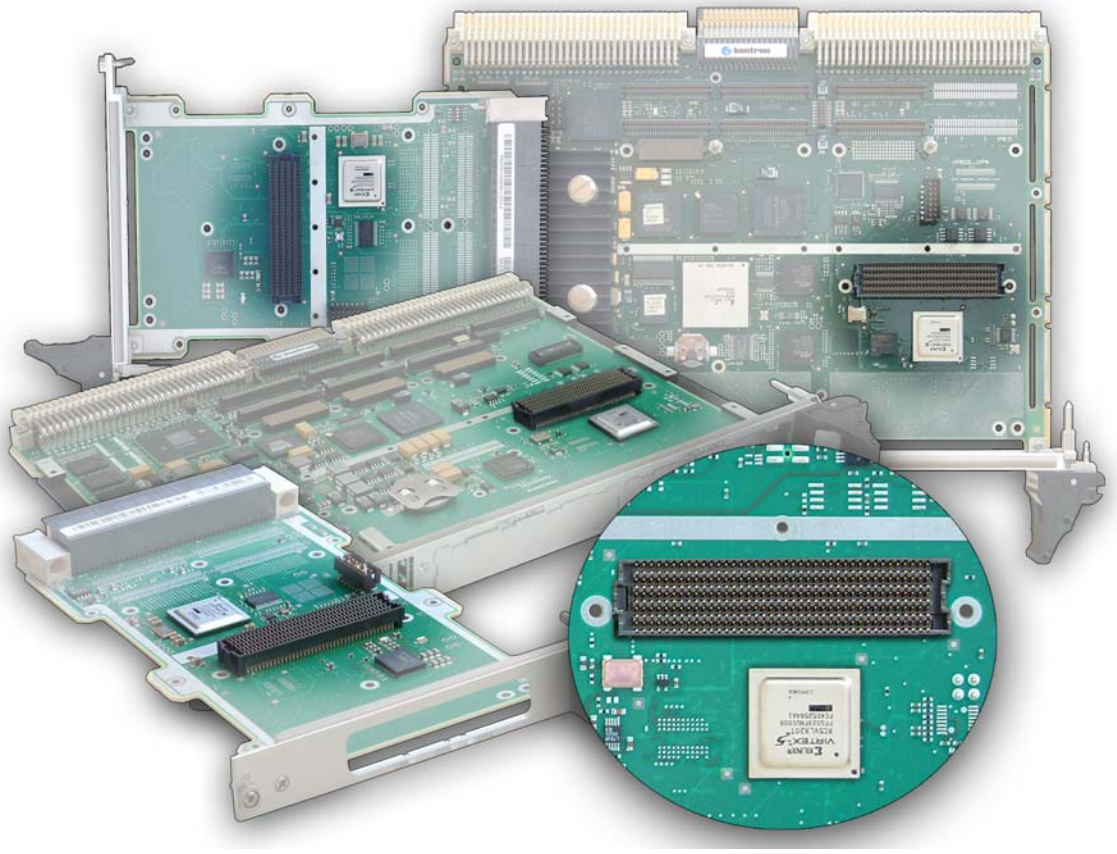
kontron

» VITA 57 «



Virtex-5 Development Kit User's Guide

SD.DT.F79-3e - June 2014

Revision History

Publication Title:	VITA 57 Development Kit User's Guide	
Doc. ID:	SD.DT.F79-3e	
Rev.	Brief Description of Changes	Date of Issue
3e	Complete Revision of the Document including “VITA 57 FPGA Configuration” (SD.DT.F74)	06-2014
2e	Update of Section 4.1 Add of Section 5.3 Kit Software Installation on a VM6250 SBC	09-2013
1e	Update of the Chapter 3. Deployment	07-2012
0e	Initial Version	05-2011

Proprietary Note

This document contains information proprietary to Kontron. It may not be copied or transmitted by any means, disclosed to others, or stored in any retrieval system or media without the prior written consent of Kontron or one of its authorized agents.

The information contained in this document is, to the best of our knowledge, entirely correct. However, Kontron cannot accept liability for any inaccuracies or the consequences thereof, or for any liability arising from the use or application of any circuit, product, or example shown in this document.

Kontron reserves the right to change, modify, or improve this document or the product described herein, as seen fit by Kontron without further notice.

Trademarks

This document may include names, company logos and trademarks, which are registered trademarks and, therefore, proprietary to their respective owners.

Environmental Protection Statement

This product has been manufactured to satisfy environmental protection requirements where possible. Many of the components used (structural parts, printed circuit boards, connectors, batteries, etc.) are capable of being recycled.

Final disposition of this product after its service life must be accomplished in accordance with applicable country, state, or local laws or regulations.

Environmental protection is a high priority with Kontron.

Kontron follows the DEEE/WEEE directive.

You are encouraged to return our products for proper disposal.

The Waste Electrical and Electronic Equipment (WEEE) Directive aims to:

- reduce waste arising from electrical and electronic equipment (EEE)
- make producers of EEE responsible for the environmental impact of their products, especially when they become waste
- encourage separate collection and subsequent treatment, reuse, recovery, recycling and sound environmental disposal of EEE
- improve the environmental performance of all those involved during the lifecycle of EEE

Conventions

This guide uses several types of notice: Note, Caution, ESD.

Note: this notice calls attention to important features or instructions.


Caution: this notice alert you to system damage, loss of data, or risk of personal injury.


ESD: This banner indicates an Electrostatic Sensitive Device.

All numbers are expressed in decimal, except addresses and memory or register data, which are expressed in hexadecimal. The prefix `0x' shows a hexadecimal number, following the `C' programming language convention.

The multipliers `k', `M' and `G' have their conventional scientific and engineering meanings of $*10^3$, $*10^6$ and $*10^9$ respectively. The only exception to this is in the description of the size of memory areas, when `K', `M' and `G' mean $*2^{10}$, $*2^{20}$ and $*2^{30}$ respectively.

When describing transfer rates, `k' `M' and `G' mean $*10^3$, $*10^6$ and $*10^9$ *not $*2^{10}$ $*2^{20}$ and $*2^{30}$.*

In PowerPC terminology, multiple bit fields are numbered from 0 to n, where 0 is the MSB and n is the LSB. PCI and CompactPCI terminology follows the more familiar convention that bit 0 is the LSB and n is the MSB.

Signal names ending with an asterisk (*) or a hash (#) denote active low signals; all other signals are active high.

Signal names follow the PICMG 2.0 R3.0 CompactPCI Specification and the PCI Local Bus 2.3 Specification.

For Your Safety

Your new Kontron product was developed and tested carefully to provide all features necessary to ensure its compliance with electrical safety requirements. It was also designed for a long fault-free life. However, the life expectancy of your product can be drastically reduced by improper treatment during unpacking and installation. Therefore, in the interest of your own safety and of the correct operation of your new Kontron product, you are requested to conform with the following guidelines.

High Voltage Safety Instructions


Warning!
All operations on this device must be carried out by sufficiently skilled personnel only.


Caution, Electric Shock!
Before installing a not hot-swappable Kontron product into a system always ensure that your mains power is switched off. This applies also to the installation of piggybacks. Serious electrical shock hazards can exist during all installation, repair and maintenance operations with this product. Therefore, always unplug the power cable and any other cables which provide external voltages before performing work.

Special Handling and Unpacking Instructions

ESD Sensitive Device!
Electronic boards and their components are sensitive to static electricity. Therefore, care must be taken during all handling operations and inspections of this product, in order to ensure product integrity at all times

Do not handle this product out of its protective enclosure while it is not used for operational purposes unless it is otherwise protected.

Whenever possible, unpack or pack this product only at EOS/ESD safe work stations. Where a safe work station is not guaranteed, it is important for the user to be electrically discharged before touching the product with his/her hands or tools. This is most easily done by touching a metal part of your system housing.

It is particularly important to observe standard anti-static precautions when changing piggybacks, ROM devices, jumper settings etc. If the product contains batteries for RTC or memory backup, ensure that the board is not placed on conductive surfaces, including anti-static plastics or sponges. They can cause short circuits and damage the batteries or conductive circuits on the board.

General Instructions on Usage

In order to maintain Kontron's product warranty, this product must not be altered or modified in any way. Changes or modifications to the device, which are not explicitly approved by Kontron and described in this manual or received from Kontron's Technical Support as a special handling instruction, will void your warranty.

This device should only be installed in or connected to systems that fulfill all necessary technical and specific environmental requirements. This applies also to the operational temperature range of the specific board version, which must not be exceeded. If batteries are present, their temperature restrictions must be taken into account.

In performing all necessary installation and application operations, please follow only the instructions supplied by the present manual.

Keep all the original packaging material for future storage or warranty shipments. If it is necessary to store or ship the board, please re-pack it as nearly as possible in the manner in which it was delivered.

Special care is necessary when handling or unpacking the product. Please consult the special handling and unpacking instruction.

Table Of Contents

Chapter 1 - Reference Document

1.1 Kontron Documents

ID	Title	Origin	Reference
H1	VITA 57 compliant FPGA Mezzanine Card (FMC)	Kontron	Datasheet FMC-SER0 #03012013MB
H2	VX3830 3U VPX Xilinx Virtex-5 FPGA Processor with FMC-Site- User's Guide	Kontron	CA.DT.A80
H3	VM6050 6U VME SBC User's Guide	Kontron	CA.DT.A93
H4	VM6250 6U VME SBC User's Guide	Kontron	CA.DT.A65

1.2 Use cases

ID	Title	Origin	Reference
S1	Platform Management FRU Information HP Storage Definition	Intel HP NEC Dell	FRU1011.PDF
S2	VITA 57.1 FPGA Mezzanine Card (FMC) Standard	ANSI	AV57DOT1.PDF

Chapter 2 - Introduction

2.1 VITA 57 Standard

The purpose of the VITA 57 standard is to create an I/O mezzanine module, which works intimately with a FPGA processing device (www.vita.com). The focus is to create a mezzanine module that minimizes the handling and formatting of the transceived data. This standard takes a new approach on interface protocols by removing the need to inject protocol data into the raw data to be processed. It assumes that the FPGA has a unique closeness with the I/O mezzanine module. This enables modification of the FPGA to process the raw data formats that the module sources and sinks.

In this way, the VITA 57 FMC standard goes beyond the mechanical definition of the FMC mezzanine and the electrical specification of the interface, and describes for instance the mechanism of identification of the FMC.

This document describes the Kontron's FPGA and Software tools available to develop and test VITA 57 functions on Kontron's products which support a VITA 57 FMC feature

2.2 Kontron VITA 57 Development Kit

The objective of the Kontron's VITA 57 Development kit is to provide hardware and software tools for the customer to design his own features.

Kontron's COTS catalogue products which support an FMC slot feature an FPGA which is loaded at first power-on with a reduced features configuration.

This document outlines such a default FPGA configuration which is delivered as an example.

- Kontron does not provide any support on the VITA 57 FPGA configuration.
- Kontron invites the customers to contact the FPGA silicon manufacturer Xilinx for any information related to the programming, use and support of the silicon device.

2.3 Prerequisites

- Knowledge and use of the Xilinx development and design tools like ISE Design Suite version 11.1.
- Knowledge and use of Verilog-XL coding.
- User's knowledge and use of Linux: the drivers and tests tools (binaries and sources) delivered as examples, are available for Linux distributions and are coded in C and Python language.

2.4 Hardware Compatibility

Kontron's products which support a VITA 57 FMC feature associated with a Xilinx Virtex-5 as build options are:

- VM6250: 6U VME Dual-Core PowerPC Single Board Computer
- VX3830: 3U VPX FMC Carrier
- VM6050: 6U VME Core i7 Single Board Computer

2.5 Use Cases

This document meets the following cases:

- New FPGA application development.
- New FPGA application deployment.
- New FPGA application test.

All of these cases will be developed hereafter.

Chapter 3 - VITA 57 Development Kit FPGA

3.1 Introduction

This chapter describes the FPGA Development Kit package for the Kontron Virtex-5 VX303x and VM6x50 SBC boards family.

It is composed from the following components:

- Verilog Code Source Example.
- Simulation TestBench.
- Synthesis Constraint Files to generate an FPGA programming file.

The VITA 57 FPGA Code example is designed with respect to:

- **On the host CPU side**: PCI-Express interface using the hardened PCI-Express blocks of the Virtex-5 FPGA
- **On the FMC side**: Interface to Kontron's FMC-SER0, a VITA 57 FMC

3.1.1 System Requirements

To install the VITA 57 Development Kit FPGA package, you need:

- Memory: 1 GB of RAM or greater.
- Operating System: Windows 2000/XP/Vista/7.
- Hard Disk: 1 GB for Core installation and component design.

3.1.2 EDA Tools Requirements

- Modelsim PE or SE 6.4
- Xilinx ISE 11.1

3.1.3 Documentation

- Xilinx Virtex-5 Logicore Endpoint Block Plusv1.13 for PCIe: UG343 of December 2, 2009
- Memory interface Guide (MIG): UG086 (v3.0) February 23, 2009

3.2 Installation

Unzip the package delivered. Files are extracted to your hard drive in a directory named VITA57_XC5VLX20T_RZZ, where “ZZ” is the Core version number and “XC5VLX20T” is the supported FPGA.

3.2.1 Directory Structure

VITA57_XC5VLX20T_Rzz

- DOCS: Kontron Documentation
- Design:
  - RTL: Verilog-XL files and synthesis constraint files
  - SIM: Modelism simulation environment
    - board: Board emulation code
    - dsport: External PCIe model
    - functional: Scripts for simulation
    - tests: Test Bench examples
  - SYNTH: Synthesis directory
    - Ipcore_dir :
      - ddr2: Customized Xilinx DDR2 IP
      - EndPoint_Pcie: Xilinx PCI Express Hard Core IP
    - Programming Files: Xilinx FPGA bit-stream files
    - XilinxISE11:
      - VM6050: Synthesis Directory for VM6050 board
      - VM6250: Synthesis Directory for VM6250 board
      - VX3830: Synthesis Directory for VM3830 board

3.3 Verilog-XL Example Design

3.3.1 Functions

The main functionalities implemented in the FPGA are:

- PCI Express interface using Xilinx PCIe Endpoint Block
- Xilinx Programmed Input Output (PIO) example design (Modified by Kontron)
- Xilinx DDR2 controller (connected to memory only on VX3830 board)
- Kontron SPI Master Module connected to the USER FLASH and the RESCUE FLASH devices which are used to fetch the FPGA firmware at power on time.
- Kontron I2C interface for access to EEPROM located on FMC-SER0
- Kontron GPIOs Modules for VPX Backplane connections
- Kontron GPIOs Modules for FMC Front panel connections
- Kontron Registers module

All Verilog source files are located in the `RTL` directory except Xilinx Cores such as DDR2 Controller and PCIe EndPoint which are located in the `SYNTH/ipcore_dir` directory

I2C and SPI Module are dedicated to Kontron boards so any changes could break some functionality. Modifying some FPGA code IP's like I2C, SPI is highly inadvisable.
All modifications will be done at customer's own risk.

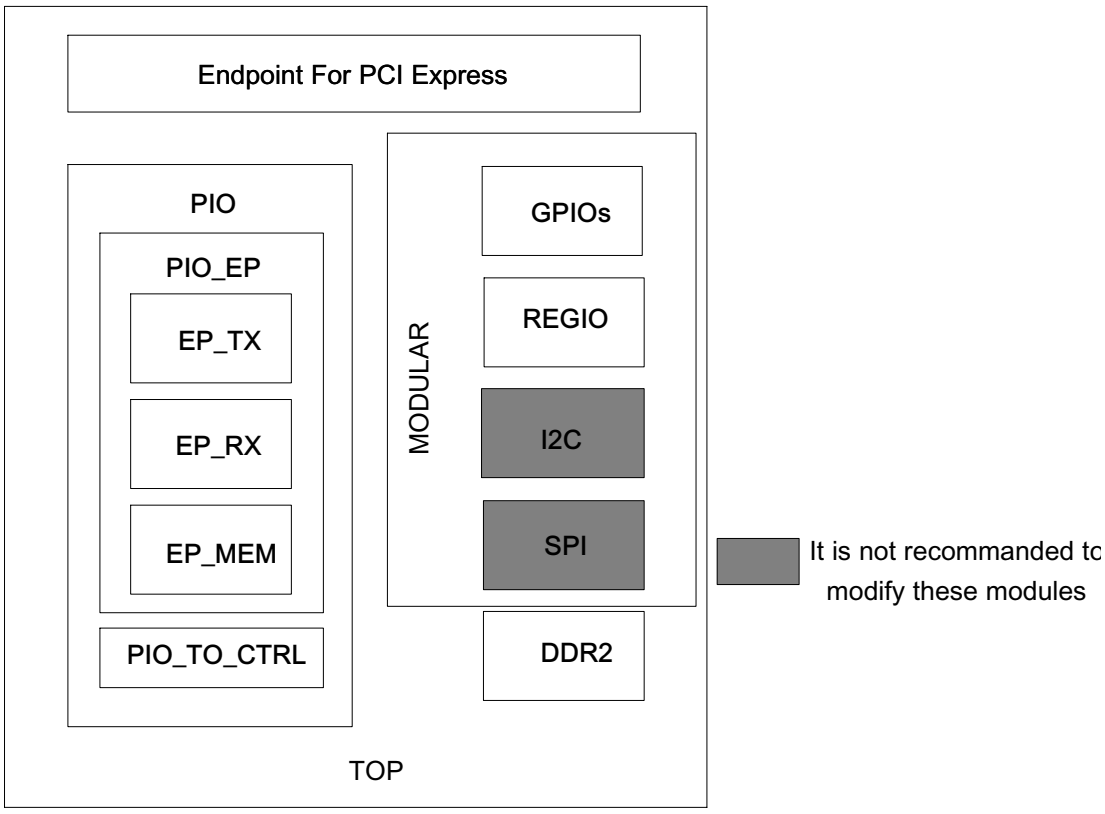


Figure 1: Verilog-XL Hierarchical Structure of the VITA 57 FPGA

3.3.2 Code Identification

Verilog-XL Top file is named `Vita57_Cxx_Fyy.v` where `xx` identifies the Carrier Board ID and `yy` is FMC ID as follows:

- `C01`: VM6250
- `C02`: VX3830
- `C03`: VM6050
- `Other`: Reserved
- `F01`: FMC-SER0
- `Other`: Reserved

To avoid mismatch with Kontron official releases we recommend to use `F9x` codes for your FMC developments.

Carrier ID and FMC ID are defined with the PCie SubDevice_ID parameter coded in the Top Level Verilog file `Vita57_Cxx_Fyy.v`:

```
parameter    SubDevice_ID = 32'h0000_0101;  // bits [15:8] => Carrier ID
                                            // bits [7:0] => FMC ID
```

» Example of Customization :

If you want to develop an FPGA code for a new FMC associated with the VM6050 board the steps are the following:

- Copy `Vita57_C03_F01.v` to `Vita57_C03_F90.v` (`F90` identifying your FMC)
- Edit `Vita57_C03_F90.v` and change Module name with `Vita57_C03_F90`
- Search `SubDevice_ID` parameters and change `bits[7:0]` to `0x90` (for this example `SubDevice_ID` becomes `32'h0000_0390`)
- Copy constraint file `Vita57_C03_F01.ucf` to `Vita57_C03_F90.ucf` and update this file for the new FMC

3.3.3 PCIe Interface

Programmed Input Output (PIO) transactions are generally used by a PCI Express system host CPU to access Memory Mapped Input Output (MMIO) and Configuration Mapped Input Output (CMIO) locations in the PCI Express fabric.

Endpoints for PCI Express accept Memory and IO Write transactions and respond to Memory and IO Read transactions with Completion with Data transactions.

3.3.3.1 PIO Design

The PIO example design (PIO design) delivered by Xilinx has been included in the Kontron FPGA Example Design, which allows users to easily bring up their system board with a known established working design to verify the link and functionality of the board.

The Xilinx PIO design is a simple target-only application that interfaces with the Endpoint for PCIe core's Transaction (TRN) interface and is provided as a starting point for customers to build their own designs.

The following features are included:

- Two transaction-specific 2 KBytes target regions using the internal Xilinx FPGA block RAMs.
- Supports single DWORD payload Read and Write PCI Express transactions to 32/64-bit address memory spaces and IO space with support for completion TLPs

The PIO design generates a completion with 1 DWORD of payload in response to a valid Memory Read 32 TLP, Memory Read 64 TLP, or IO Read TLP request presented to it by the core. In addition, the PIO design returns a completion without data with successful status for IO Write TLP request.

The PIO design processes a Memory or IO Write TLP with 1 DWORD payload by updating the payload into the target address in the FPGA block RAM space.

The Xilinx Example Design has been customized by Kontron to add a multi-Channels Interface to access GPIOs, SPI, I2C and Register Spaces.

The example Design implemented by Kontron has one of the 2 KBytes RAM target spaces replaced by a GPIO, I2C, SPI and Register Space.

3.3.3.2 PCIe Base Address Registers (BAR)

The PIO design supports three discrete target spaces of 2 KBytes represented by a separate Base Address Register (BAR). Each of these spaces corresponds to one 2 kB address region in the PIO design.

TLP Traffic is decoded as follows:

TLP Transaction Type	BAR	Ressources	Size
64-bit address Memory TLP transactions	0-1	Internal RAM	2 KBytes
32-bit address Memory TLP transactions	2	Register Space	2 KBytes

3.3.3.3 PCIe Configuration Space Header

Name	Offset	Value	Note
Vendor ID	0x00	0x1059	KONTRON
Device ID	0x02	0x9050	VITA 57 FPGA
Revision ID	0x08	0x01	
Subsystem ID [15:8]	0x2E	0x00 Reserved 0x01 VM6250 0X02 VX3830 0x03 VM6050	Carrier Supported
Subsystem ID [7:0]	0x2E	0x00 Reserved 0x01 FMC-SER0	FMC Supported

3.3.4 Simulation Files

All Simulation files are located in the SIM directory.

The simulation example is based on a Top level file named `Topserial.v` so it is necessary to copy your `RTL/Vita57_CXX_FYY.v` top to `RTL/Topserial.v` before launching Modelsim simulation.

If you have new files in your design you must add a path to them in the `SIM/functional/ board_rtl_x04.f` file.

`SIM/functional/run.do` can be used to run a simulation example.

3.4 Synthesis and Bitstream Generation

The worklib directory used by the Xilinx ISE tool is located at `SYNTH/XilinxISE11/"Board_name"` where `"Board_name"` identifies the carrier board (VM6050, VM6250 or VX3830).

3.4.1 Programming File Generation (.bit File)

For example, if you want to generate a bitstream related to your `Vita57_C03_F90` code you can use the ISE Graphical User Interface as follows :

- Launch `ISE` tool in graphical mode
- Click **Open Project** and select `VM6050.xise` on the synthesis directory `SYNTH/XilinxISE11/VM6050`
- click "**Remove**" and select `Vita57_C03_F01.ucf` in the Hierarchy panel (old version)
- click "**Remove**" and select `Vita57_C03_F01.v` in the Hierarchy panel (old version)
- click "**Add Source**" and select the new file `Vita57_C03_F90.v` in the RTL directory and define it as Top level synthesis module
- click "**Add Source**" and select your `Vita57_C03_F90.ucf` file located in the `RTL` directory
- If you have other files to add, repeat the "**Add Source**" sequence
- To generate the FPGA Programming file, select `Vita57_C03_F90.v` in the Hierarchy view and run "**Generate Programming File**". The generation process is launched. If no error appears, `Vita57_C03_F90.bit` file will be generated into the `SYNTH/XilinxISE11/VM6050/Worklib` directory.

3.4.2 Conversion of Programming File into Intel MCS format (.mcs File)

This bitstram file (`.bit`) generated previously must be converted into a `.mcs` file before it can be loaded into the User Flash :

- Create a `.tcl` file named `PromGen.tcl` in the synthesis directory containing the following commands:

```
setMode -pff
addConfigDevice  -name Vita57_C03_F90
setSubmode -pffspi
addDesign -version 0 -name "0"
addDeviceChain -index 0  setAttribute -configdevice -attr compressed -value "FALSE"
setAttribute -configdevice -attr autoSize -value "FALSE"
setAttribute -configdevice -attr fileFormat -value "mcs"
setAttribute -configdevice -attr fillValue -value "FF"
setAttribute -configdevice -attr swapBit -value "FALSE"
setAttribute -configdevice -attr dir -value "UP"
setAttribute -configdevice -attr multiboot -value "FALSE"
setAttribute -configdevice -attr spiSelected -value "TRUE"
addPromDevice -p 1 -size 2048
setAttribute -design -attr name -value "0000"
addDevice -p 1 -file Worklib/vita57_c03_f90.bit
generate
quit
```

- Open `cmd` window in the synthesis
- Launch Xilinx iMPACT tool in batch mode by typing for example:

```
E:\Xilinx\10.1\ISE_DS\ISE\bin\nt64\impact.exe -batch PromGen.tcl
```

- `vita57_c03_f90.mcs` file will be created in the synthesis directory.

3.5 Registers

3.5.1 Register Mapping

Register Space is split into 64-Byte areas named “Channels” and then each register is mapped at an address defined by BAR2 + Channel Base Address (CBA) + Register Offset

Channel	Channel Base Address (CBA)	Register Offset	Register Name	Size	Comment
0	0x000	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		
1	0x040	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		
2	0x080	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		

Channel	Channel Base Address (CBA)	Register Offset	Register Name	Size	Comment
3	0x0C0	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		
4	0x100	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		
5	0x140	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		
6	0x180	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		

Channel	Channel Base Address (CBA)	Register Offset	Register Name	Size	Comment
7	0x1C0	0x000	GPIO_Control	4	GPIO Function
		0x004	GPIO_DataOut	4	
		0x008	GPIO_In	4	
		0x00C	GPIO_IntMask	4	
		0x010	GPIO_IntPolarity	4	
		0x014	GPIO_IntMode	4	
		0x018	GPIO_IntToggle	4	
		0x01C	GPIO_IntStatus	4	
		0x020-0x03C	Reserved		
8	0x200	0x000	GPIO_Control	2	GPIO Function
		0x004	GPIO_DataOut	2	
		0x008	GPIO_In	2	
		0x00C	GPIO_IntMask	2	
		0x010	GPIO_IntPolarity	2	
		0x014	GPIO_IntMode	2	
		0x018	GPIO_IntToggle	2	
		0x01C	GPIO_IntStatus	2	
		0x020-0x03C	Reserved		
9	0x240	0x000	GPIO_Control	10	GPIO Back-End or Front
		0x004	GPIO_DataOut	10	
		0x008	GPIO_In	10	
		0x00C	GPIO_IntMask	10	
		0x010	GPIO_IntPolarity	10	
		0x014	GPIO_IntMode	10	
		0x018	GPIO_IntToggle	10	
		0x01C	GPIO_IntStatus	10	
		0x020-0x03C	Reserved		
10	0x280	0x000	GPIO_Control	14	GPIO Back-End
		0x004	GPIO_DataOut	14	
		0x008	GPIO_In	14	
		0x00C	GPIO_IntMask	14	
		0x010	GPIO_IntPolarity	14	
		0x014	GPIO_IntMode	14	
		0x018	GPIO_IntToggle	14	
		0x01C	GPIO_IntStatus	14	
		0x020-0x03C	Reserved		

Channel	Channel Base Address (CBA)	Register Offset	Register Name	Size	Comment
11	0x2C0	0x000	GPIO_Control	5	
		0x004	GPIO_DataOut	5	
		0x008	GPIO_In	5	
		0x00C	GPIO_IntMask	5	
		0x010	GPIO_IntPolarity	5	
		0x014	GPIO_IntMode	5	
		0x018	GPIO_IntToggle	5	
		0x01C	GPIO_IntStatus	5	
		0x020-0x03C	Reserved		
12	0x300	0x000	GPIO_Control	5	
		0x004	GPIO_DataOut	5	
		0x008	GPIO_In	5	
		0x00C	GPIO_IntMask	5	
		0x010	GPIO_IntPolarity	5	
		0x014	GPIO_IntMode	5	
		0x018	GPIO_IntToggle	5	
		0x01C	GPIO_IntStatus	5	
		0x020-0x03C	Reserved		
13	0x340	0x000	GPIO_GlobalStatus	14	
14	0x380	0x000	REGI0 General Purpose Register	7	PWRON_VADJ, VADJ_VID[2:0]
		0x004	REGI1 General Purpose Register	1	Read Only Register (Debug)
		0x008	REGI2 General Purpose Register	1	R/W one bit register
		0x00C	REGI3 General Purpose Register	1	R/W one bit register
		0x010	REGI4 General Purpose Register	1	R/W one bit register
		0x014	REGI5 General Purpose Register	1	R/W one bit register
		0x01C	REGI6 General Purpose Register	1	R/W one bit register
15	0x3C0	0x000	I2C_Start	1	
		0x004	I2C_Stop	1	
		0x008	I2C_ Read	32	
		0x00C	I2C_Write	8	
		0x010	I2C_CtrlStatus	4	
16	0x400	0x000	SPI Control	32	
		0x004	SPI Status	2	
		0x008	SPI Transmit Data	8	
		0x00C	SPI Receive Data	32	

3.5.2 GPIOs Functions

The GPIO interface is designed with 13 Channels (4 to 14 GPIOs per channel).

Each GPIO in a Channel can be individually configured as an input, output or bi-directional and can generate Interrupt on Level or Edge when configured as input.

Configuration and monitoring of GPIOs are available through 6 Control Registers for each channel.

Registers are mapped on addresses defined by BAR2 + Channel Base Address + Register Offset

» GPIOs Interface: Channels Mapping

The number of GPIO ports in each Channel and Base Addresses of the channels are defined as follows:

Channel	Channel Base Address	Size (nb of GPIOs)	FMC Connector Pin Name associated to FPGA Channel's GPIOs	FMC-SER0 Transceiver Control Signals	Back-End	Front-end	FMC-SER0 IOs
0	0x000	4	{LA_N[1:0], LA_P[1:0]}	{DY, RB, DZ, RA}	X	X	{TX1,RX2,TX2,RX1}
1	0x040	4	{LA_N[3:2], LA_P[3:2]}	{DY, RB, DZ, RA}	X	X	{TX3,RX4,TX4,RX3}
2	0x080	4	{LA_N[5:4], LA_P[5:4]}	{DY, RB, DZ, RA}	X	X	{TX5,RX6,TX6,RX5}
3	0x0C0	4	{LA_N[7:6], LA_P[7:6]}	{DY, RB, DZ, RA}	X	X	{TX7,RX8,TX8,RX7}
4	0x100	4	{LA_N[9:8], LA_P[9:8]}	{DY, RB, DZ, RA}	X	X	{TX9,RX10,TX10,RX9
5	0x140	4	{LA_N[11:10], LA_P[11:10]}	{DY, RB, DZ, RA}	X	X	{TX11,RX12,TX12,RX11
6	0x180	4	{LA_N[13:12], LA_P[13:12]}	{DY, RB, DZ, RA}	W	W	{TX13,RX14,TX14,RX13
7	0x1C0	4	{LA_N[15:14], LA_P[15:14}	{DY, RB, DZ, RA}			{TX15,RX16,TX16,RX15
8	0x200	2	({LA_N[16], LA_P[16]}	{DXEN15_16, DXEN}	X	X	
9	0x240	10	{LA_N/P[21:17]}		X		FMC-GPIO[10 : 1]
10	0x280	14	{LA_N/P[28:22]}		W	W	FMC-GPIO[24 : 11]
11	0x2C0	5	{LA_P[32], LA_N[30], LA_P[30], LA_N[29], LA_P[29]}	CAN_Enable, CAN2_TX/RX , CAN1_TX/RX			CAN2+/-, CAN1+/-
12	0x300	5	({LA_N[32], LA_P[33], LA_N[33], LA_P[31], LA_N[31]}	Test_Point, LED2_RED/GRN, LED1_RED/GRN			Test_Point,
13	0x340	14	This Channel contains only **Global Status Register** and does not control any GPIOs				

W: Wrapping is used to choice **between RS232/435 Transceivers and CAN Bus**

X: Routing is always available

Examples :

- Bit GPIO[0] of Channel 9 controls FMC-GPIO[1] signal which is an output of FMC-SER0
- Bit GPIO[2] of Channel 5 controls RB signal of transceiver RX/TX11-12
- Bit GPIO[1] of Channel 8 controls 435/232b mode for all transceivers

		Register SIZE (number of GPIOs)					
Register Name	Register Address Offset	Channel 0 to 7	Channel 8	Channel 9	Channel 10	Channel 11	Channel 12
GPIO_Control	0x00	4	2	10	14	5	5
GPIO_DataOut	0x04	4	2	10	14	5	5
GPIO_In	0x08	4	2	10	14	5	5
GPIO_IntMask	0x0C	4	2	10	14	5	5
GPIO_IntPolarity	0x10	4	2	10	14	5	5
GPIO_IntMode	0x14	4	2	10	14	5	5
GPIO_IntToggle	0x18	4	2	10	14	5	5
GPIO_IntStatus	0x1C	4	2	10	14	5	5

		Register SIZE
Register Name	Register Address Offset	Channel 13
GPIO_GlobalStatus	0x00	14

		Channel 0 to 7 (SIZE = 4)	Channel 8 (SIZE = 2)	Channel 11 (SIZE = 5)	Channel 12 (SIZE = 5)
Register Name	Register Address Offset	Recommended Value {DY, RB, DZ, RA}	Recommended Value {435/232b, DXEN}	Recommended Value {CAN_Enable, CAN2_TX, CAN2_RX, CAN1_TX, CAN1_RX}	Recommended Value {Test_Point, LED2_RED, LED2_GRN, LED1_RED, LED1_GRN}
GPIO_Control	0x00	1 0 1 0	1 1	1 1 0 1 0	X 1 1 1 1
GPIO_DataOut	0x04	X 0 X 0	X X	X X 0 X 0	X X X X X
GPIO_In	0x08	N.A	N.A	N.A	N.A
GPIO_IntMask	0x0C	1 X 1 X	1 1	1 1 X 1 X	X 1 1 1 1
GPIO_IntPolarity	0x10	0 X 0 X	0 0	0 0 X 0 X	X 0 0 0 0
GPIO_IntMode	0x14	0 X 0 X	0 0	0 0 X 0 X	X 0 0 0 0
GPIO_IntToggle	0x18	0 X 0 X	0 0	0 0 X 0 X	X 0 0 0 0
GPIO_IntStatus	0x1C	0 X 0 X	0 0	0 0 X 0 X	X 0 0 0 0

3.5.3 GPIOs Registers Space

GPIOs Interface: Control Register

The control register enables or disables the output and bi-directional modes of operation for each GPIO of the selected Channel (Size parameter depending on the channel)

GPIO_Control @Channel_BaseAdd + 0x0 - R/W - Reset Value = 0		
Bit#	Name	Description
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_Control[Size-1]	Control for GPIO[Size-1] of the selected Channel 1: GPIO is configured as output 0: GPIO is configured as input
...	...	...
n	GPIO_Control[n]	Control for GPIO[n] of the selected Channel 1: GPIO is configured as output 0: GPIO is configured as input
...	...	...
0	GPIO_Control[0]	Control for GPIO[0] of the selected Channel 1: GPIO is configured as output 0: GPIO is configured as input

GPIOs Interface: Data Out Register

This register specifies the Data driven by each GPIO of the selected Channel.

GPIO_DataOut @Channel_BaseAdd + 0x4 - R/W - Reset Value = 0		
Bit#	Name	Description
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_DataOut[Size-1]	Data to drive to GPIO[Size-1] of the selected Channel
...	...	...
n	GPIO_ DataOut [n]	Data to drive to GPIO[n] of the selected Channel
...	...	...
0	GPIO_ DataOut [0]	Data to drive to GPIO[0] of the selected Channel

GPIOs Interface: GPIO In Register

Access to this register reads Data from the GPIOs of the selected Channel.

GPIO_In @Channel_BaseAdd + 0x8 – Read Only - Reset Value = N.A		
Bit#	Name	Description
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_In [Size-1]	**Data from** GPIO[Size-1] of the selected Channel
...	...	...
n	GPIO_ In [n]	**Data from** GPIO[n] of the selected Channel
...	...	...
0	GPIO_ In [0]	**Data from** GPIO[0] of the selected Channel

GPIOs Interface: Interrupt Mask Register

The interrupt mask register defines which GPIO input of the selected Channel can generate an interrupt to the PCIe interface.

GPIO_IntMask @Channel_BaseAdd + 0xC- R/W - Reset Value = FFF		
Bit#	Name	Description
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_IntMaskl[Size-1]	**Interrupt Mask** for GPIO[Size-1] of the selected Channel 1: GPIO can not generate an interrupt 0: GPIO can generate an interrupt
...	...	...
n	GPIO_IntMaskl[n]	**Interrupt Mask** for GPIO[n] of the selected Channel 1: GPIO can not generate an interrupt 0: GPIO can generate an interrupt
...	...	...
0	GPIO_IntMaskl[0]	**Interrupt Mask** for GPIO[0] of the selected Channel 1: GPIO can not generate an interrupt 0: GPIO can generate an interrupt

GPIOs Interface: Interrupt Polarity Register

The Interrupt Polarity Register defines the active level of a GPIO input which can generate an interrupt.

GPIO_GPIO_IntPolarity @Channel_BaseAdd + 0x10 - R/W - Reset Value = 0		
Bit#	**Name**	**Description**
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_IntPolarity[Size-1]	**Interrupt Polarity** for GPIO[Size-1] of the selected Channel 1: Level 1 or Rising edge on GPIO generates an interrupt 0: Level 0 or Falling edge on GPIO generates an interrupt
...	...	...
n	GPIO_IntPolarity[n]	**Interrupt Polarity** for GPIO[n] of the selected Channel 1: Level 1 or Rising edge on GPIO generates an interrupt 0: Level 0 or Falling edge on GPIO generates an interrupt
...	...	...
0	GPIO_IntPolarity[0]	**Interrupt Polarity** for GPIO[0] of the selected Channel 1: Level 1 or Rising edge on GPIO generates an interrupt 0: Level 0 or Falling edge on GPIO generates an interrupt

GPIOs Interface: Interrupt Mode Register

The Interrupt Mode Register defines if interrupt is generated on edge or level as defined in GPIO_IntPolarity.

GPIO_IntMode @Channel_BaseAdd + 0x14 - R/W - Reset Value = 0		
Bit#	**Name**	**Description**
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_IntMode[Size-1]	**Interrupt Mode** for GPIO[Size-1] of the selected Channel 1: Interrupt generation on GPIO is **Edge** Sensitive 0: Interrupt generation on GPIO is **Level** Sensitive
...	...	...
n	GPIO_IntMode[n]	**Interrupt Mode** for GPIO[n] of the selected Channel 1: Interrupt generation on GPIO is **Edge** Sensitive 0: Interrupt generation on GPIO is **Level** Sensitive
...	...	...
0	GPIO_IntMode[0]	**Interrupt Mode** for GPIO[0] of the selected Channel 1: Interrupt generation on GPIO is **Edge** Sensitive 0: Interrupt generation on GPIO is **Level** Sensitive

GPIOs Interface: Interrupt Toggle Register

When a GPIO is selected in Toggle Mode by this register GPIO_IntMode and GPIO_Int_polarity are not taken in consideration and interruption will be generated if GPIO changes state and it is not masked.

GPIO_IntToggle @Channel_BaseAdd + 0x18 - R/W - Reset Value = 0		
Bit#	Name	Description
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_IntToggle[Size-1]	**Interrupt Toggle Mode** for GPIO[Size-1] of the selected Channel 1: Interrupt is generated on GPIO Toggle 0: No interrupt generated
...	...	...
n	GPIO_ IntToggle [n]	**Interrupt Toggle Mode** for GPIO[n] of the selected Channel 1: Interrupt is generated on GPIO Toggle 0: No interrupt generated
...	...	...
0	GPIO_IntToggle [0]	**Interrupt Toggle Mode** for GPIO[0] of the selected Channel 1: Interrupt is generated on GPIO Toggle 0: No interrupt generated

GPIOs Interface: Interrupt Status Register

For each channel the interrupt Status Register defines which GPIO has generated an Interrupt.

Write 0 to Reset Status bits.

GPIO_IntStatus @Channel_BaseAdd + 0x1C - R/W - Reset Value = 0		
Bit#	Name	Description
32-Size	Reserved	Not used – always read as 0's, writes ignored
Size-1	GPIO_IntStatus[Size-1]	**Interrupt Status** for GPIO[Size-1] of the selected Channel 1: GPIO has generated an interrupt 0: GPIO has not generated an interrupt
...	...	...
n	GPIO_IntStatusl[n]	**Interrupt Status** for GPIO[n] of the selected Channel 1: GPIO has generated an interrupt 0: GPIO has not generated an interrupt
...	...	...
0	GPIO_IntStatusl[0]	**Interrupt Status** for GPIO[0] of the selected Channel 1: GPIO has generated an interrupt 0: GPIO has not generated an interrupt

3.5.4 Global Status Function

GPIOs Interface: Global Status Register

This register can be acceded by selecting **Channel 13 with offset 0x00**. Bits 12 to 0 are Status information associated with channels 12-0. Bit 13 is a Global Status bit which indicates that an interrupt has been generated on the PCIe Interface.

GPIO_GlobalStatus @0x340 - R/W - Reset Value = 0		
Bit#	**Name**	**Description**
32-14	Reserved	Not used – always read as 0's, writes ignored
13	Interrupt Asserted	**Interrupt Assertion Flag** 1: Interrupt has been generated and not yet acknowledged. No more interrupts can be asserted. 0: No Interrupt pending. Write 0 to clear Interrupt
12	GPIO_GlobalStatus[12]	**Channel 12 Status** 1: At least one GPIO in this channel has generated an Interrupt 0: No Interrupt activated in this channel
11	GPIO_GlobalStatus[11]	**Channel 11 Status** (Read Only)
10	GPIO_GlobalStatus[10]	**Channel 10 Status** (Read Only)
9	GPIO_GlobalStatus[9]	**Channel 9 Status** (Read Only)
8	GPIO_GlobalStatus[8]	**Channel 8 Status** (Read Only)
7	GPIO_GlobalStatus[7]	**Channel 7 Status** (Read Only)
6	GPIO_GlobalStatus[6]	**Channel 6 Status** (Read Only)
5	GPIO_GlobalStatus[5]	**Channel 5 Status** (Read Only)
4	GPIO_GlobalStatus[4]	**Channel 4 Status** (Read Only)
3	GPIO_GlobalStatus[3]	**Channel 3 Status** (Read Only)
2	GPIO_GlobalStatus[2]	**Channel 2 Status** (Read Only)
1	GPIO_GlobalStatus[1]	**Channel 1 Status** (Read Only)
0	GPIO_GlobalStatus[0]	**Channel 0 Status** (Read Only)

3.5.5 REGIO Function

» REGIO 0: VADJ Control

This register is used to set VADJ according to the FMC plugged into the VITA 57 connector (see section 4.5.4.3 - Power Configuration Mode, page 38).

REGIO 0 @0x380 + 0x000 – RW - Reset Value=0		
Bit#	Name	Description
31-4	Reserved	Not used – always read as 0's, writes ignored
3	PWR on-VADJ	**VADJ Power Controller State**: 0: Power_Off 1: Power_On
2-0	VADJ-VID	**VADJ Setting for FMC IOs**: 000: 3.3V 010: 1.8 V 100: 2.5 V 110: 1.5 V Other: reserved

3.5.6 I2C Function

The I2C interface provides access to the EEPROM located on the VITA 57 FMC Module.

» I2C Interface: Start Register

I2C generates START condition when write access is done to this register.

I2C_Start @0x3C0 + 0x000 – (RW) - Reset Value: 0		
Bit#	Name	Description
31-0	I2C_Start	Always read as 0's, writes of any value initiates an I2C Start condition

» I2C Interface: Stop Register

I2C generates a STOP condition when write access is done to this register.

I2C_Stop @0x3C0 + 0x004 – (RW) - Reset Value = 0		
Bit#	Name	Description
31-0	I2C_Stop	Always read as 0's, writes of any value initiates an I2C Stop condition

I2C Interface: Read Register

This register is used to specify number of bytes to read at once (1 to 4) and reads the bytes received.

After Write Start and the addressing phase a write to this register with the number of Data bytes to read starts the READ I2C cycle.

I2C_Read @0x3C0 + 0x004 – (RW) - Reset Value = 0		
Bit#	**Name**	**Description**
		Read Access
31-0	I2C_DataRead	Contains **Bytes received** during I2C Read Transfer (max 4). First byte is in bits 7-0.
		Write Access
31-5	Reserved	Not used – writes ignored
4	I2C_ReadAckn	Indicates if the Read cycle must be Acknowledged (0) or not (1)
3-0	I2C_NbBytes	Number of Bytes to read on I2C bus (max 4)

I2C Interface : Write Register

Write to this register starts the WRITE I2C cycle with the specified DATA.

I2C_WriteCmd @0x3C0 + 0x00C – Write Only - Reset Value = N.A		
Bit#	**Name**	**Description**
7-0	I2C_Data	Data to write on I2C bus The first write to this register after a Start Condition Data is the **Slave Address and RW bit**

I2C Interface : Control and Status Register

I2C_CtrlStatus @0x3C0+ 0x010 - R/W - Reset Value = 0		
Bit#	**Name**	**Description**
31-4	Reserved	Not used – always read as 0's, writes ignored
3	I2C_Speed	**Speed Control (RW)** 1: Fast-speed mode (400 kbits/s) 0: Low-speed mode (100 kbits/s)
2	I2C_Held	**Held Flag (Read Only)** Reserved
1	I2C_Ack	**Ack Flag (Read Only)** 1: Transfer has not been Acknowledged by slave 0: Transfer Acknowledged by slave
0	I2C_Busy	**Busy Flag (Read Only)** 1: Indicates that I2C bus is busy 0: I2C bus not busy

3.5.7 SPI Function

SPI Interface: Control Register

SPI_Control @0x400 + 0x000 – RW - Reset Value : 0x0000_0000		
Bit#	Name	Description
31-24	SPI_Length	Number of bytes for the SPI cycle
23-11	Reserved	Not used – writes ignored
10-8	SPI_Factor	Clock Divider Factor 0 : SCK is 62.5 MHz 1: SCK is 31,25 MHz 2: SCK is 15.625 MHz n: SCK is 62.5/n MHz
7-6	Reserved	Not used – writes ignored
5	SPI_SelectOnReset	Indicates which Flash Device was selected on reset (Read Only) 1: Rescue Flash 0: Standard Flash
4	SPI_SelectConf	FLASH Select (Read Only) Indicates which Flash Device is currently 1: Rescue Flash 0: Standard Flash
3	SPI_DeviceSel	Flash Device Selection when Configuration Mode is set to “Software Mode” (bit 2 of this register set to 1) 1: Rescue Flash 0: Standard Flash
2	SPI_SelectMode	Configuration Mode 1: Flash Device Selection is done by bit 3 of this register (Soft Config) 0: Flash Device Selection is done by external hardware
1	SPI_CPHA	CPHA value
0	SPI_CPOL	CPOL value

The SPI protocol defines four combinations of SCK phase and polarity with respect to the data, they are referred to as CPOL (clock polarity) and CPHA (clock phase).

CPOL	CPHA	Transfer
0	0	SCK rising-edge transfer SCK transitions in middle of bit timing
1	0	SCK falling-edge transfer SCK transitions in middle of bit timing
0	1	SCK falling-edge transfer SCK transitions in beginning of bit timing
1	1	SCK rising-edge transfer SCK transitions in beginning of bit timing

Serial Clock: This pin is used to provide a clock to the device and is used to control the flow of data to and from the device. Command, address, and input data present on the SI pin is always latched on the rising edge of SCK, while output data on the SO pin is always clocked out on the falling edge of SCK.

SPI Interface: Status Register

SPI_Status @0x400 + 0x004 – Read Only - Reset Value = 0		
Bit#	Name	Description
31-2	Reserved	Not used – writes ignored
1	SPI_Status	0: Indicates that global transfer is done 1: Transfer pending
0	SPI_Status	0: Indicates that byte transfer is done 1: Transfer pending

SPI Interface: Transmit Register

SPI _Transmit @0x400 + 0x008 – RW - Reset Value = 0		
Bit#	Name	Description
31-8	Reserved	Not used – writes ignored
7-0	SPI_DataOut	DATA to write on SPI bus

SPI Interface: Receive Register

SPI _Receive @0x400 + 0x00C – Read Only - Reset Value = N.A		
Bit#	Name	Description
31-0	Reserved	Contains DATA received during SPI Read Transfer Bits 0-7 hold the frist byte.

Chapter 4 - VITA 57 Development Kit Software

4.1 Introduction

This chapter describes the VITA 57 software package for the Virtex-5 SBC boards family.

It is composed from the following components:

> An I2C bus driver to access the FMC-SER0 I2C EEPROM.

> A userland tool e2fmc handles the following tasks related to the EEPROM:
  - From a binary file that fits [S1] requirements, generate an inf style config file
  - From an inf style config file generate a binary file that fits [S1] requirements
  - From a binary file that fits [S1] requirements, extract IPMI information and configure the VADJ value (as defined in [S2] and [S1])

> An SPI bus driver enables the download/upload of the VITA 57 FPGA flash memory content.

> A demo driver for the FMC-SER0 board:
  - Userland tool to configure and drive GPIOs
  - Kernel driver which handles GPIOs interruptions.

4.2 Installation on a VM6050 SBC

4.2.1 Overview

The goal of this document is to help you through the installation process of the VITA 57 distribution on the Kontron VME VM6050 board.

4.2.2 Release Content

The release is made of one 64 bit CD-ROM, reference: VITA 57 Development Kit for FEDORA14 x86_64 on VM6050 SBC.

This CD-ROM contents 2 packages:

- A binary package: `vita57-1.1-2.6.35.6_11245.ki7.fc14.x86_64.12058.x86_64.rpm`
- A source package: `vita57-1.1-2.6.35.6_11245.ki7.fc14.x86_64.12058.src.rpm`

4.2.3 Hardware Requirements

- A VM6050 board equipped with an FMC-SER0 board.
- A USB DVD-ROM device.
- A graphical display, a USB keyboard and mouse OR a VT100 console plugged to the tty on the front serial line.
- A SATA disk connected to the SATA connectors available on the VM6050-RTM board.

OR

- An optional onboard USB Flash Disk

4.2.4 Software Requirements

A Fedora 14 X86 64-bit release installed on one of the following bootable devices: SATA disk or USB flash disk.

At first, verify the Fedora release installed on your disk is compatible with this VITA 57 release.

Once you have booted your system, run the following command:

```
[root@localhost ~]# cat /etc/fedora-release
Fedora release 14 (Laughlin)
```

revealing that the current version is Fedora14

```
[root@localhost ~]#       uname -a
                          Linux lnx7.ariane.local 2.6.35.6-11245.ki7.fc14.x86_64 #1
SMP Fri Sep 2 18:46:11 CEST
2011 x86_64 x86_64 x86_64 GNU/Linux
```

revealing the current version of the kernel: `2.6.35.6-11245.ki7.fc14.x86_64`

4.2.5 Installation

The distribution consists of a single CD-ROM: **VITA 57 Development Kit for FEDORA14 x86_64 on VM6050 SBC IDXXXX.**

Follow the procedure below to install the VITA 57 package onto an existing bootable media (SATA disk or USB flash):

- Insert the Kontron VITA 57 CD-ROM into the USB DVD-ROM drive.
- Boot the VM6050 board on the SATA disk previously installed with the Fedora 14 64 bits.
- Login as root and run following commands:

```
[root@localhost ~]#  mount /dev/sr0 /mnt
[root@vx6060 ~]# cd /mnt
```

If the kernel version of your target is `2.6.35.6-11245.ki7.fc14.x86_64`, you have simply to install the binary package by:

```
root@localhost ~]# rpm -i vita57-1.1-2.6.35.6_11245.ki7.fc14.x86_64.12058.x86_64.rpm
```

The installation process will take few seconds. If you have another version of the kernel, follow this procedure, install the source package by running:

```
[root@localhost ~]# rpm -ivh vita57-1.1-2.6.35.6_11245.ki7.fc14.i686.PAE.12052.src.rpm
```

Then rebuild the binary package by:

```
[root@localhost ~]# cd /root/rpmbuild/SPECS
[root@localhost ~]#  rpmbuild -bb VITA 57.spec
```

Finally install the new RPM:

```
[root@localhost ~]# rpm -ivh /usr/src/redhat/RPMS/i386/vita-1.1*.rpm
```

Reboot the machine to take into account the VITA 57 package installation. The VITA 57 software is ready to be used. To verify the installation has been successful, run the following command:

```
[root@localhost ~]# rpm -ql vita57
/etc/rc.d/init.d/spi-flash-vita57
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/i2c/busses/i2c-vita57.ko
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/input/gpio-event-vita57.ko
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/mtd/mtd_dataflash.ko
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/spi/spi-vita57.ko
/usr/local/bin/e2fmc
/usr/local/bin/e2fmc-vita57.pyc
/usr/local/bin/eeprog
/usr/local/bin/gpio-vita57
/usr/local/bin/gpio-vita57.pyc
/usr/local/bin/ipmi-vita57.pyc
/usr/local/bin/mcs_to_bin-vita57
/usr/local/bin/mcs_to_bin-vita57.pyc
/usr/local/bin/mtd_info
/usr/local/share/vita57/fmc_ser0.bin
/usr/local/share/vita57/fmc_ser0.inf
/usr/local/share/vita57/fpga_flash.bin
/usr/local/share/vita57/fpga_flash.mcs
/etc/rc.d/init.d/i2c-vita57
/etc/rc.d/init.d/spi-flash-vita57
```

which indicates the content of the VITA 57 kit package.

4.3 Installation on a VM6250 SBC

4.3.1 Overview

The goal of this chapter is to help you through the installation process of the VITA 57 distribution on the Kontron VME VM6250 board.

4.3.2 Release Content

The release is made of one CD-ROM, reference: VITA 57 Development Kit for Fedora 9 PPC on VM6250 SBC.

This CD-ROM contents 2 packages:

- A binary package: `vita57-1.1-2.6.25_09350.vm6250.fc9.ppc.smp.13246.ppc.rpm`
- A source package: `vita57-1.1-2.6.25_09350.vm6250.fc9.ppc.smp.13246.src.rpm`

4.3.3 Hardware Requirements

- A VM6250 board equipped with an FMC-SER0 board.
- A USB DVD-ROM device.
- A VT100 console plugged to the tty on the front serial line.
- A SATA disk connected to the SATA connectors available on the VM6250-RTM board.

  or
- An optional onboard USB Flash Disk

4.3.4 Software Requirements

Fedora 9 PPC release installed on one of the following bootable devices: SATA disk or USB flash disk.

At first, verify the Fedora release installed on your disk is compatible with this VITA 57 release.

Once you have booted your system, run the following command:

```
[root@localhost ~]# cat /etc/fedora-release
Fedora release 9 (Sulphur)
```

revealing that the current version is Fedora 9

```
[root@localhost ~]# uname -a
Linux Nice 2.6.25-09350.vm6250.fc9.ppc.smp #3 SMP Thu Feb 21 17:25:43
CET 2013 ppc ppc ppc GNU/Linux
```

revealing the current version of the kernel: `2.6.25-09350.vm6250.fc9.ppc.smp`

4.3.5 Installation

The distribution consists of a single CD-ROM: **VITA 57 Development Kit for Fedora 9 PPC on VM6250 SBC ID=13246.**

Follow the procedure below, to install the VITA 57 package onto an existing bootable media (SATA disk or USB-flash):

- Insert the Kontron VITA 57 CD-ROM into the USB DVD-ROM drive.
- Boot the VM6250 board on the SATA disk previously installed with the Fedora 9 PPC.
- Login as root and run following commands:

```
[root@localhost ~]# mount /dev/sr0 /mnt
[root@localhost ~]# cd /mnt
```

If the kernel version of your target is, `2.6.25-09350.vm6250.fc9.ppc.smp` you have simply to install the binary package by:

```
root@localhost ~]# sh install_vita57.sh
root@localhost ~]# reboot
...
```

The installation process will take few seconds. If you have another version of the kernel, follow this procedure, install the source package by running:

```
[root@localhost ~]# rpm -ivh
vita57-1.1-2.6.25_09350.vm6250.fc9.ppc.smp.13246.src.rpm
```

Then rebuild the binary package by:

```
[root@localhost ~]# cd /root/rpmbuild/SPECS
[root@localhost ~]# rpmbuild -bb VITA 57.spec
```

Finally install the new RPM:

```
[root@localhost ~]# rpm -ivh /usr/src/redhat/RPMS/i386/vita-1.1*.rpm
```

Reboot the machine to take into account the VITA 57 package installation.

The VITA 57 software is ready to be used. To verify the installation has been successful, run the following command:

```
# rpm -ql vita57
```

The result should be:

```
/etc/rc.d/init.d/e2fmc
/etc/rc.d/init.d/spi-flash-vita57
/lib/modules/2.6.25-09350.vm6250.fc9.ppc.smp/kernel/drivers/i2c/busses/i2c-vita57.ko
/lib/modules/2.6.25-09350.vm6250.fc9.ppc.smp/kernel/drivers/input/gpio-event-vita57.ko
/lib/modules/2.6.25-09350.vm6250.fc9.ppc.smp/kernel/drivers/mtd/mtd_dataflash.ko
/lib/modules/2.6.25-09350.vm6250.fc9.ppc.smp/kernel/drivers/spi/spi-vita57.ko
/usr/local/bin/e2fmc
/usr/local/bin/e2fmc.pyc
/usr/local/bin/eeprog
/usr/local/bin/gpio
/usr/local/bin/gpio.pyc
/usr/local/bin/ipmi.pyc
/usr/local/bin/mcs_to_bin
/usr/local/bin/mcs_to_bin.pyc
/usr/local/bin/mtd_info
/usr/local/share/vita57/VM6050_vita57_C03_F01_R02.bin
/usr/local/share/vita57/VM6050_vita57_C03_F01_R02.mcs
/usr/local/share/vita57/fmc_ser0.bin
/usr/local/share/vita57/fmc_ser0.inf
/usr/local/share/vita57/fpga_flash.bin
/usr/local/share/vita57/fpga_flash.mcs
/usr/local/share/vita57/test/evdev.pyc
/usr/local/share/vita57/test/gpio.pyc
/usr/local/share/vita57/test/gpio_tests.pyc
/usr/local/share/vita57/test/logged_commands.pyc
/usr/local/share/vita57/test/sis_unittest.pyc
/usr/local/share/vita57/test/test_constants.pyc
/usr/local/share/vita57/test/utils.pyc
```

Which indicates the content of the VITA 57 kit package.

4.4 I2C Driver / Access to the I2C FMC-SER0 EEPROM

The VITA 57 I2C bus driver is named `i2c-vita57` and enables the access to the I2C bus of the FMC connector site.

To know the characteristics of this module, simply run:

```
 [root@localhost ~]#  modinfo i2c-vita57
For example, on a VM6050  board, the result should be similar to :
filename:
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/i2c/busses/i2c-vita57.ko
license:        GPL
description:    I2C-Bus adapter for VITA57
author:         Kontron <support-kom-sa@kontron.com>
srcversion:     BDC1DE8F514951F54BCCB2A
depends:        i2c-core
vermagic:       2.6.35.6-11245.ki7.fc14.x86_64 SMP mod_unload
parm:          speed:I2C bus speed [0 for slow = 100 kbits/s, 1 for fast = 400 kbits/s]
(int)
parm:           debug:set to 1 to enable debugging (int)
parm:           timeout:set to non 0 to change timeout (int)
```

To download this module, use the following command:

```
[root@localhost ~]# modprobe i2c-vita57
```

A service `i2c-vita57` is also available.

As a result it is possible to download automatically this module during the Linux boot stage by running:

```
[root@localhost ~]# chkconfig --add e2fmc
[root@localhost ~]# chkconfig e2fmc on
[root@localhost ~]# service e2fmc start
```

Once the `i2c-vita57` driver has been downloaded, an I2C bus should have been created.

To know the number of this bus, run the following command after installation of `i2c-tools.ppc` package:

```
[root@localhost ~]# i2cdetect -l | grep VITA
```

The output should be like:

```
i2c-9        i2c          VITA57 I2C Bus adapter at f09003c0         I2C adapter
```

In this particular example, the I2C bus number is 9. It could be another one depending on the particular board you have.

Once you know the bus number value, you can use the whole standard I2C commands such as:

> `i2cdetect`, `i2cdump` ...

For example, to know the location of the FMC-SER0 EEPROM, run:

```
[root@localhost ~]# i2cdetect -y 9
```

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Which reveals the location of the EEPROM attached to the FMC board at the address 0x50.

4.5 e2fmc Tool

4.5.1 Overview

The `e2fmc` tool enables the following operations:

> Download and parse the IPMI data contained in an I2C EEPROM attached to an FMC board. According to these data, configure the VADJ power value.

> Decode binary IPMI data to text file.

> Encode IPMI text file information to binary IPMI data which can be uploaded to an I2C EEPROM attached to an FMC board.

The following IPMI structures defined in [S1] are supported:

> Common Header

- Chassis Info
- Board Info
- Product Info
- MultiRecord Area
  - DC Output Record (Type 0x01)
  - DC Load Record (Type 0x2)
  - OEM Record FMC Subtype 0: Base Definition (refer to [S2] for more)

4.5.2 Command Line Reference

Here is the full command line reference for the e2fmc tool:

```
[root@localhost ~]# e2fmc -h
                Usage:
                e2fmc [options]
                hint: see -h or --help for options
                Options:
                -h, --help
                show this help message and exit

                -e ENCODE_FROM_FILE, --encode_from_file=ENCODE_FROM_FILE
           Specify to create binary EEPROM content file from a config file [output file is specified with
                --output_file_name]
                -d DECODE_FROM_FILE, --decode_from_file=DECODE_FROM_FILE
                   Specify to create a config file from a binary EEPROM content file
                -v, --vpd
                   Specify to display VPD [Vital Product Data]
                -o OUTPUT_FILE_NAME, --output_file_name=OUTPUT_FILE_NAME
                   Specify the output file name
                -a, --power_configuration_from_file
                    Specify to read and decode EEPROM binary content file and then apply VITA57 power
configuration of the FPGA (Setting VADJ value) [needs]
         -w DOWNLOAD_EEPROM_DATA, --download_eeprom_data=DOWNLOAD_EEPROM_DATA
                   Specify the file to store eeprom binary data
                -W, --wide_eeprom_16_bit
                   Specify the size of the eeprom default to 8bit for safety if this is not set
                -D, --debug
                   Enable debug output N

WARNING: For operation involving accessing the FMC I2C EEPROM (« -v », « -a » and « -w »), it is
mandatory to set the correct size mode (ie: 8 or 16bit wide) in order to access the device. The default
is to use the 8bit mode (no « -W » switch), this way if a 16bit wide EEPROM is accessed in this mode
the communication will failed but the EEPROM content will not be damaged. But do not access an 8bit
wide EEPROM with the « -W » switch as the EEPROM device will mess a pointer address setting
operation with a write operation and the EEPROM content will be changed/damaged.
```

4.5.3 Decoding Mode

In this mode the e2fmc tool is used for decoding raw binary IPMI data to an «inf» style file containing the decoded information. Here is an example of using the e2fmc tool in this mode:

```
[root@localhost ~]# e2fmc -d eeprom.bin -o eeprom.inf
```

If no file is specified for output, the result is printed on the standard output as seen in the following output:

```
[root@localhost ~]# e2fmc -d /usr/local/share/vita57/fmc_ser0.bin
[BoardInfo]
language code = [0]
mfg. date / time = 01/01/2010 00:00:00
board manufacturer = KONTRON
board product name = FMC-SER0
board serial number = ******************
board part number = MYBOARDPARTNUMBER
fru file id = 0

[ProductInfo]
language code = [0]
manufacturer name = KONTRON MODULAR COMPUTERS SAS
product name = FMC-SER0
product part/model number = FMC-SER0
product version = 1.0
product serial number = ******************
asset tag = TAG0
fru file id = 0

[MultiRecord-0-FMCRecord]
manufacturer id = [162, 18, 0]
data = [0, 28, 68, 64, 0, 0, 0, 0]

MultiRecord-1-DCLoadRecord]
voltage required = [0]
nominal voltage (10 mv) = 330
spec´d minimum voltage (10 mv) = 322
spec´d maximum voltage (10 mv) = 347
spec´d ripple and noise pk-pk 10hz to 30 mhz (mv) = 50
minimum current load (ma) = 0
maximum current load (ma) = 1000

[MultiRecord-2-DCLoadRecord]
voltage required = [1]
nominal voltage (10 mv) = 0
spec´d minimum voltage (10 mv) = 0
spec´d maximum voltage (10 mv) = 0
spec´d ripple and noise pk-pk 10hz to 30 mhz (mv) = 0
minimum current load (ma) = 0
maximum current load (ma) = 0
```

```
[MultiRecord-3-DCLoadRecord]
voltage required = [2]
nominal voltage (10 mv) = 0
spec´d minimum voltage (10 mv) = 0
spec´d maximum voltage (10 mv) = 0
spec´d ripple and noise pk-pk 10hz to 30 mhz (mv) = 0
minimum current load (ma) = 0
maximum current load (ma) = 0

[MultiRecord-4-DCOutputRecord]
output information = [3]
nominal voltage (10 mv) = 0
maximum negative voltage deviation (10 mv) = 0
maximum positive voltage deviation (10 mv) = 0
ripple and noise pk-pk 10hz to 30 mhz (mv) = 0
minimum current draw (ma) = 0
maximum current draw (ma) = 0

[MultiRecord-4-DCOutputRecord]
output information = [3]
nominal voltage (10 mv) = 0
maximum negative voltage deviation (10 mv) = 0
maximum positive voltage deviation (10 mv) = 0
ripple and noise pk-pk 10hz to 30 mhz (mv) = 0
minimum current draw (ma) = 0
maximum current draw (ma) = 0

[MultiRecord-5-DCOutputRecord]
output information = [4]
nominal voltage (10 mv) = 0
maximum negative voltage deviation (10 mv) = 0
maximum positive voltage deviation (10 mv) = 0
ripple and noise pk-pk 10hz to 30 mhz (mv) = 0
minimum current draw (ma) = 0
maximum current draw (ma) = 0

[MultiRecord-6-DCOutputRecord]
output information = [0]
nominal voltage (10 mv) = 0
maximum negative voltage deviation (10 mv) = 0
maximum positive voltage deviation (10 mv) = 0
ripple and noise pk-pk 10hz to 30 mhz (mv) = 0
minimum current draw (ma) = 0
maximum current draw (ma) = 0
```

4.5.4 Encoding Mode

In this mode the e2fmc tool is used for encoding an «inf» style file filled with IPMI structure information in text format to raw binary IPMI data.

Here is an example of using the e2fmc tool in this mode:

```
[root@localhost]# cd /tmp ; e2fmc -e /usr/local/share/vita57/fmc_ser0.inf -o eeprom.bin
[root@localhost]#         cat eeprom.bin | xxd
0000000: 0100 0001 0914 00e1 0108 0040 5e70 074b  ...........@^p.K
0000010: 4f4e 5452 4f4e 0846 4d43 2d53 4552 3012  ONTRON.FMC-SER0.
0000020: 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a  **********
0000030: 2a2a 114d 5942 4f41 5244 5041 5254 4e55  **.MYBOARDPARTNU
0000040: 4d42 4552 0130 c17b 010b 001d 4b4f 4e54  MBER.0.{....KONT
0000050: 524f 4e20 4d4f 4455 4c41 5220 434f 4d50  RON MODULAR COMP
0000060: 5554 4552 5320 5341 5308 464d 432d 5345  UTERS SAS.FMC-SE
0000070: 5230 0846 4d43 2d53 4552 3003 312e 3012  R0.FMC-SER0.1.0.
0000080: 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a 2a2a  **********
0000090: 2a2a 0454 4147 3001 30c1 0000 0000 00ab  **.TAG0.0.......
00000a0: fa02 0bac 4da2 1200 001c 4440 0000 0000  ....M.....D@....
00000b0: 0202 0df9 f600 4a01 4201 5b01 3200 0000  ......J.B.[.2...
00000c0: e803 0202 0dff f001 0000 0000 0000 0000  ................
00000d0: 0000 0000 0202 0dfe f102 0000 0000 0000  ................
00000e0: 0000 0000 0000 0102 0dfd f303 0000 0000  ................
00000f0: 0000 0000 0000 0000 0102 0dfc f404 0000  ................
0000100: 0000 0000 0000 0000 0000 0182 0d00 7000  ..............p.
0000110: 0000 0000 0000 0000 0000 0000            ............
```

4.5.4.1 IPMI Data Fetch Mode

In this mode the e2fmc tool does the following steps:

1. Compute the total size of the IPMI data stored in EEPROM by reading some bytes from different IPMI header structure.
2. Then download the whole IPMI data block from the EEPROM.

Here is an example output of using e2fmc in IPMI Data Fetch mode:

```
[root@localhost ~]# e2fmc -w ipmi.bin -W
0000000: 0100 0001 0b10 00e3 010a 0000 0000 154d  ...............M
0000010: 7920 426f 6172 6420 4d61 6e75 6661 6374  y Board Manufact
0000020: 7572 6572 154d 7920 426f 6172 6420 5072  urer.My Board Pr
0000030: 6f64 7563 7420 4e61 6d65 164d 7920 426f  oduct Name.My Bo
0000040: 6172 6420 5365 7269 616c 204e 756d 6265  ard Serial Numbe
0000050: 7200 00c1 0000 00d2 0105 0000 1343 4841  r............CHA
0000060: 4d45 4c45 4f4e 2046 4d43 2053 4552 2030  MELEON FMC SER 0
0000070: 0000 0446 4d43 3000 00c1 0000 0000 0040  ...FMC0........@
0000080: fa02 0b4c ad00 12a2 0000 0000 0000 0000  ...L............
0000090: 0202 0db5 3a00 4a01 0000 0000 0000 0000  ....:.J.........
00000a0: 0000 0202 0d50 9f01 b4fb 0000 0000 0000  .....P..........
00000b0: 0000 0000 0202 0d4f a002 b4fb 0000 0000  .......O........
00000c0: 0000 0000 0000 0102 0dfd f303 0000 0000  ................
00000d0: 0000 0000 0000 0000 0102 0dfc f404 0000  ................
00000e0: 0000 0000 0000 0000 0000 0182 0dfb 7505  ..............u.
00000f0: 0000 0000 0000 0000 0000 0000            ............
```

4.5.4.2 VPD Data Mode

In this mode the e2fmc tool does the following steps:

1. Download the IPMI data (see IPMI Data Fetch Mode)
2. Decode the board Info IPMI data structure
3. Display the following fields of this structure:
   - BoardInfo->'board product name'
   - BoardInfo->'board serial number'
   - BoardInfo->'board part number'

Here is an example output of using e2fmc in vpd mode:

```
[root@localhost ~]# e2fmc -v -W
Product Name [My Board Product Name]
Serial Number             [My Board Serial Number]
Part Number  []
```

4.5.4.3 Power Configuration Mode

The e2fmc tool's principal function is to:

1. Download the IPMI Data (see IPMI Data Fetch Mode)
2. Decode this data and parse the DCLoad record with output number "0" which must give the VADJ for P1 3.
3. According to the value specified in this record the VADJ can or cannot be set:
   - If the nominal value of the record fits a possible settable value of VADJ, this value is set.
   - If a possible settable value of VADJ falls in the maximum and minimum range defined in the record this value is set.

The following command line will make all the previous steps:

```
[root@localhost ~]# e2fmc -a -W
Nominal value [3.3 V] is OK
Setting VADJ to [3.3 V]
Setting VADJ_VID to [0x00000008]
```

4.5.5 EEPROG Tool

The e2fmc tool uses EEPROG as a back-end for reading EEPROM data on the VITA 57 I2C bus using the i2c-vita57 bus driver.

Here is the full reference of this command line tool:

```
[root@localhost ~]#  eeprog -h eeprog 0.7.5, a 24Cxx EEPROM reader/writer Copyright (c)
2003 by Stefano Barbato - All rights reserved.
Usage: eeprog [-fqxdh] [-16|-8] [ -r addr[:count] | -w addr ]  /dev/i2c-N  i2c-address

Address modes:
             -8           Use 8bit address mode for 24c0x...24C16 [default]
             -16          Use 16bit address mode for 24c32...24C256
Actions:
             -r addr[:count]      Read [count] (1 if omitted) bytes from [addr]
                                  and print them to the standard output
             -w addr              Write input (stdin) at address [addr] of
                                  the EEPROM
             -h                   Print this help
Options:
             -x                   Set hex output mode
             -d                   Dummy mode, display what *would* have been done
             -f                   Disable warnings and don´t ask confirmation
             -q                   Quiet mode

The following environment variables could be set instead of the command
line arguments:
             EEPROG_DEV           device name(/dev/i2c-N)
             EEPROG_I2C_ADDR      i2c-address

             Examples
             1- read 64 bytes from the EEPROM at address 0x54 on bus 0 starting
             at address 123 (decimal)
                          eeprog /dev/i2c-0 0x54 -r 123:64
             2- prints the hex codes of the first 32 bytes read from bus 1
                at address 0x22
                          eeprog /dev/i2c-1 0x51 -x -r 0x22:0x20
             3- write the current timestamp at address 0x200 of the EEPROM on
                bus 0 at address 0x33
                          date | eeprog /dev/i2c-0 0x33 -w 0x200
```

Here is a typical example of using EEPROG to write a data file to an I2C EEPROM on bus i2c-9 at address 0x50:

```
[root@localhost ~]#   cat /usr/local/share/vita57/fmc_ser0.bin | eeprog -w 0 -f -16
/dev/i2c-9 0x50
eeprog 0.7.5, a 24Cxx EEPROM reader/writer
Copyright (c) 2003 by Stefano Barbato - All rights reserved.
  Bus: /dev/i2c-9, Address: 0x50, Mode: 16bit
  Writing stdin starting at address 0x0
................................................................................
............................................................
```

Here the “-16” switch is used as the EEPROM is a 16-bit wide addressing device. For an 8-bit address EEPROM use "-8" switch instead of "-16".

Here is a typical example of using EEPROG to read the content of an I2C EEPROM on bus i2c-9 at address 0x50 to a data file:

```
root@localhost ~]# eeprog -16 -r 0:512 /dev/i2c-9 0x50 -f > test.bin
eeprog 0.7.5, a 24Cxx EEPROM reader/writer
Copyright (c) 2003 by Stefano Barbato - All rights reserved.
Bus: /dev/i2c-9, Address: 0x50, Mode: 16bit
Reading 512 bytes from 0x0
```

4.6 FPGA SPI Flash Driver

4.6.1 Overview

The VITA 57 Virtex-5 FPGA component is connected to a flash device which is used to fetch FPGA firmware at power on time.

The Flash device can be uploaded by using software tools (MTD Data Flash Driver associated with the SPI Bus Driver) described in this chapter or by using hardware tools (Xilinx HW-USB-II Platform cable and iMPACT software) described in chapter 5 - FPGA Flash Update - Deployment, page 44.

4.6.2 SPI Bus Driver

To access the flash device in order to download/upload FPGA firmware, the FPGA package contains an SPI bus driver which is named “spi-vita57”.

To know the characteristics of this module, simply run:

```
[root@localhost ~]# modinfo spi-vita57
For example, on a VM6050  board, the result should be similar to :
filename:
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/spi/spi-vita57.ko
license:        GPL
description:    SPI-Bus adapter for VITA57
author:         Kontron <support-kom-sa@kontron.com>
srcversion:     CCC97EF6EC602AADCCFB493
depends:
vermagic:       2.6.35.6-11245.ki7.fc14.x86_64 SMP mod_unload
parm:           clock_factor:Set the clock divider factor [default to 0] [min=0, max =
7] (int)
parm:           debug:set to 1 to enable debugging [default to 0] (int)
parm:           timeout:set spi timeout [default to 100] (int)
parm:           cpol:set clock polarity [default to 0]  [0 or 1] (int)
parm:           cpha:set clock phase [default to 0]  [0 or 1] (int)
```

To load it use the following command:

```
root@localhost ~]#modprobe spi-vita57 clock_factor=2
```

The clock_factor=2 option sets the SPI clock frequency to 15.625 MHz. Please refer to document [H1] for more information.

4.6.3 MTD DataFlash Driver

To access the flash device, the `mtd_dataflash` module is needed. It will use the spi-vita57 bus driver to handle the flash component (detect the device, reading/writing/erasing by means of a block device). For more information on this driver please refer to the Linux kernel documentation.

The `mtd_dataflash` module is automatically loaded as soon as the spi-vita57 module is loaded as seen in the following output:

```
root@localhost ~]#   modprobe spi-vita57 clock_factor=2
[root@localhost ~]# tail -f -n 10 /var/log/messages
Feb 16 18:29:55 lnx7 kernel: [22387.477867] Unloading device ...
Feb 16 18:29:58 lnx7 kernel: [22390.522319] Probing device ...
Feb 16 18:29:58 lnx7 kernel: [22390.525374] Device found ...
Feb 16 18:29:58 lnx7 kernel: [22390.528387] spi_master spi0: VITA57 SPI Bus adapter at
0xf0900400
Feb 16 18:29:58 lnx7 kernel: [22390.534457] vita57_spi_setup ...
Feb 16 18:29:58 lnx7 kernel: [22390.537682] VITA57_SPI: Standard Flash selected
Feb 16 18:29:58 lnx7 kernel: [22390.542261] mtd_dataflash spi0.0: AT45DB161x (2112 KBytes)
pagesize 528 bytes (OTP)
Feb 16 18:29:58 lnx7 kernel: [22390.549981] spi_master spi0: Dataflash detected
```

The `mtd_dataflash` module auto-detects the type of flash used and sets up an `mtd` device accordingly.

The following command lists all `mtd` devices on the system:

```
root@localhost ~]# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00210000 00000210 "spi0.0-AT45DB161x"
```

In this example the FPGA Flash is listed as “`spi0.0-AT45DB161x`" .

Do not access other mtd devices. The VITA 57 FPGA flash device is “`spi0.0- AT45DB161x`" thus in this case “`/dev/mtd0`”

The validity of the device can be verified with the `mtd_debug` command from the `mtd-utils` package as seen in the following output:

```
root@localhost ~]# modprobe mtdchar
root@localhost ~]#mtd_debug info /dev/mtd0
mtd.type = MTD_DATAFLASH
mtd.flags = MTD_CAP_NANDFLASH
mtd.size = 2162688 (2M)
mtd.erasesize = 528
mtd.writesize = 528
mtd.oobsize = 0
regions = 0
```

Then the resulting `mtd` device can be used like another block device but the maximum SPI Bytes transferred in one message (while CS keeps enabled) is 255 bytes. Thus when using the “`dd`” command to access the `mtd` device use “`ibs`” and “`obs`” options with a maximum value of 200 bytes (because there are command bytes before data is actually read or written to the device).

Here is an example of writing the flash device with the correct `obs` option (obs<250):

```
root@localhost~]# dd if=/usr/local/share/vita57/fpga_flash.bin of=/dev/mtd0 obs=128
1526+1 records in
6105+0 records out
781440 bytes (781 kB) copied, 147.864 s, 5.3 kB/s
```

Using `dd` for uploading/downloading code to the FPGA must be done with care as it uses low level access to the `mtd block` device.

It is hazardous to break the transfer before its end.

4.6.4 mcs_to_bin Tool

As the `dd` command is difficult to use for the end user, there is a tool in the BSP named "`mcs_to_bin`" which can be used to do two important things when working with FPGA firmware deployment:

- Convert an Intel MCS file to a raw binary file.
- Convert an Intel MCS file and write it on the fly to the `mtd block` device.

Here is the full reference of the `mcs_to_bin` tool:

```
[root@localhost ~]#  mcs_to_bin -h
Options:
  -h, --help            show this help message and exit
  -i INPUT_MCS_FILE, --input_mcs_file=INPUT_MCS_FILE
                        Specify the mcs input file
  -o OUTPUT_BIN_FILE, --output_bin_file=OUTPUT_BIN_FILE
                        Specify the bin output file
```

Here is an example on how to convert an Intel MCS file to a binary file which can be downloaded to the `mtd` device using `raw dd` command:

```
[root@localhost ~]# cd /tmp; mcs_to_bin -i /usr/local/share/vita57/fpga_flash.mcs -o
fpga.bin
Writing [fpga.bin] from [/usr/local/share/vita57/fpga_flash.mcs]
[100 %]
Complete !
```

It is hazardous to break the transfer before its end. Using the `mcs_to_bin` to upload the FPGA firmware in the flash is much secure than using `raw dd` command as the MCS file is parsed one line at a time and the correct seek() in the block device is done to write at the correct address. This can even be used to deploy firmware patches (i.e. only write code in some flash memory address to fix an existing firmware). The drawback is that the write access of the tool are not optimized, thus it is slower than using `raw dd` command with optimized "`obs`" option.

Chapter 5 - FPGA Flash Update - Deployment

Two dedicated FLASH devices are available to contain the FPGA image which is fetched at Power-On time:

- The standard FPGA FLASH (User Flash).
- The rescue FPGA FLASH (Rescue Flash).

Initially both of them contain the same image: the FMC-SER0 FPGA test image.

During the deployment stage, you need to update one or both of these flash devices. For this you can use the software tool (MTD Data Flash Driver associated with the SPI Bus Driver) or JTAG tool (Xilinx Platform cable USB-II associated with Xilinx iMPACT software).

5.1 Flash Selection (Rescue or User Flash)

Selection of the FLASH you want to update must be done by setting a microswitch on the board (see User's Guide of the board) :

Board	Microswitch Number	Contact Number	ON function	OFF function
VX3830	SW2	1	Rescue Flash	User Flash
VM6050	SW4	4	Rescue Flash	User Flash
VM6250	SW5	4	Rescue Flash	User Flash

On VM6050 and VM6250 boards a Flash Write Protection (WP) must be disabled to permit Flash update.

Board	Microswitch Number	Contact Number	ON function	OFF function
VM6050	SW4	1	User Flash WP Disabled	User Flash WP Enabled
		2	Rescue Flash WP Disabled	Rescue Flash WP Enabled
VM6250	SW5	1	User Flash WP Disabled	User Flash WP Enabled
		2	Rescue Flash WP Disabled	Rescue Flash WP Enabled

Don't forget to set microswitches to the correct position before using the board.

5.2 JTAG/SPI Selection for the Xilinx Platform Cable

The Xilinx Platform cable can be used in JTAG mode or in SPI mode depending on configuration of a micro-switch on the board.

The JTAG mode is used to configure directly the FPGA with the `.bit` file generated from ISE (image is not written in Flash and will be lost after Power-off).

The SPI mode is used to write in User Flash or rescue Flash (`.mcs` file).

The setting of JTAG or SPI mode is done as follows:

Board	Microswitch Number	Contact Number	ON function	OFF function
VX3830	SW2	3	JTAG access to the FPGA	SPI access to the User /Rescue FLASH
VM6050	SW4	3		
VM6250	SW5	3		

5.3 Upload Flash with Software Tool (SPI Driver)

After you have selected the FLASH (User or Rescue) with the microswitches located on the board (see 5.2), use the following command to update the Flash:

```
root@localhost ~]# dd if=fpga_flash.bin of=/dev/mtd0 obs=128
```

where `fpga_flash.bin` represents your `fpga binary` image (see section 4.6.4 “mcs_to_bin Tool” page 43 to know how generate a `.bin` file)

The result should look like:

```
1526+1 records in
6105+0 records out
781440 bytes (781 kB) copied, 147.864 s, 5.3 kB/s
```

Refer to the section 4.6 “FPGA SPI Flash Driver“ page 41, for more details concerning the access to the FPGA flash.

5.4 Upload Flash with Xilinx Platform Cable USB II

The Flash uploading needs to use a programming file in Intel format (.mcs) and can be done with the Xilinx Platform Cable associated with iMPACT software used in Direct SPI mode.

To be able to program User or Rescue Flash, the Virtex-5 FPGA needs to have been loaded with a valid image because the SPI bus is shared between the FPGA, the FLASH and the P0401 board connector used for the JTAG/SPI Xilinx Platform Cable. If the Flash loading fails you must first reload the FPGA with a valid image from the Rescue Flash or User Flash, or program the FPGA directly using the JTAG mode on the Xilinx Platform cable.

Uploading an FPGA image when the rescue mode is activated will break the rescue image

5.4.1 Requirements

Due to the fact that the Direct SPI mode has been removed in iMPACT 12.1, it is necessary to install iMPACT 10.1 on your computer to be able to program User or Rescue Flash with the Xilinx Platform Cable.

All the documentation associated with the iMPACT tool can be found at the Xilinx web site http://www.xilinx.com/support/documentation/index.htm.

The application note **xapp951.pdf** can help the user program the flash in SPI mode.

The programming file in Intel Format (.mcs) must be generated from a «.bit» image obtained with Xilinx ISE tool.

To do this conversion you must use the Xilinx iMPACT tool as described in section 3.3.2 “Code Identification” page 7 (Note that the iMPACT release used is unimportant for this operation)

5.4.2 Upload

Flash Selection must have been done as described in section 5.1 “Flash Selection (Rescue or User Flash)” page 44 (refer to the mother board -SBC or Carrier- User’s Guide to know where are located the dip switches mentioned)

- Plug the JTAG probe into the JTAG connector.

Figure 2: Example of Plug-in of Xilinx Platform Cable at P0401 board connector (VX3830)

- Plug the board into the Backplane and Power-On
- Start Xilinx iMPACT version 10.1.
- Select the "**Direct SPI Configuration**" option in the top left window
- Then in the main window, right click and select "**Cable setup**"
- In the window "**Cable Communication Setup**", check that all the parameters are OK for the cable used (ie Xilinx USB Cable, …).
- Correct parameters if necessary and then press "**OK**" button
- In the main window, right click and select "**Add SPI Device …**"
- In the window "**Add Device**", select the file you want to program in the SPI user flash of the VX3830 board. The tool looks for files with "***.mcs**"
- In the window "**Select Device Part Name**", select "**AT45DB161D**" for the flash part name, then press "**OK**" button.
- Click on the SPI flash device to have it in green color, then right click and select "**Program**"
- The programming process starts, a window «**Progress Dialog**» shows the progress of the process.
- The window at the bottom indicates the operation on the flash device.
- The operation of programming is successful when the main window shows the message "**Program Succeeded**" in blue color.
- Now your flash is programmed with your FPGA code. To load the FPGA with this code, turn off the power, remove the programming connector and turn on the power.
- Now at each Power_On, the FPGA code will be loaded with the new code.
- If the FPGA load is successful, the LED "FPGA", passes from red color to green color (VX3830).
- If the FPGA fails, the LED "FPGA" remains red color (VX3830)

5.5 Upload Flash on FMC

The VITA 57 standard also allows the flash containing the FPGA code to be on the FMC. Kontron does not implement flash on current FMC so this feature is not available in the kit.

5.6 GPIO FMC Demo

5.6.1 User Land Driver

Here is the complete command line reference of the GPIO user land driver:

```
[root@localhost ~]#  gpio -h
Usage:  gpio [options]
hint: see -h or --help for options


Options:
  -h, --help            show this help message and exit
  -c CHANNEL_SELECT, --channel_select=CHANNEL_SELECT
                        Select the gpio channel to use (0 to 13)
  -u, --read_and_display
                        Read all the device registers then print out the
                        registers
  -s SELECT_REGISTER, --select_register=SELECT_REGISTER
                        Select the register to updated by the value specified
                        by -v option
  -v VALUE, --value=VALUE
                        Set the value for register operations 0xXXXXXXXX for
                        hexa value or 0bXXXXXXXX for binary value
  -t SELECT_REGISTER_BIT, --select_register_bit=SELECT_REGISTER_BIT
                        Select the register/bit [example: "-t -t
                        GPIO_Control.1"] to be set/reset bit 1 of the
                        GPIO_Control register by the value specified by -b
                        option
  -b BIT_VALUE, --bit_value=BIT_VALUE
                        Set the value for register.bit operation specified by
                        -t option [can be 0 or 1]
  -g READ_REGISTER, --read_register=READ_REGISTER
                        Read the value of the specified register
  -d, --dont_mask_unused_bit
                        Specify to do not mask printing of unused gpio
                        register bit(s)
  -e, --debug           Activate debug mode
```

Here is an example of requesting status of GPIO channel 0:

```
[root@localhost ~]# gpio -c 0 -u
*****************************************************************
Channel: 0 => 4 GPIOs
*****************************************************************
GPIO_Control         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1000 0x00000008
GPIO_DataOut         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1000 0x00000008
GPIO_In              XXXXXXXXXXXXXXXXXXXXXXXXXXXX1011 0x0000000B
GPIO_IntMask         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1110 0x0000000E
GPIO_IntPolarity     XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntMode         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1111 0x0000000F
GPIO_IntToggle       XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntStatus       XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
*****************************************************************
GPIO_GlobalStatus    00000000000000000000000000000000

*************************************************************
```

To change the value of a register use a command of the form "`gpio -s REGISTER_NAME -v 0xXX -c CHANNEL -u`" where "`REGISTER_NAME`" is the name of the register and "`0xXX`" is the value to write to the register. "`Channel`" is the number of the GPIO channel.

This, for example, sets the I/O direction of the channel 0 to output:

```
[root@localhost ~]# gpio -c 0 -u -s GPIO_Control -v 0xFF
-------------------------------------------------------------
*****************************************************************
Channel: 0 => 4 GPIOs
*****************************************************************
GPIO_Control         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1010 0x0000000A
GPIO_DataOut         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1000 0x00000008
GPIO_In              XXXXXXXXXXXXXXXXXXXXXXXXXXXX1001 0x00000009
GPIO_IntMask         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1110 0x0000000E
GPIO_IntPolarity     XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntMode         XXXXXXXXXXXXXXXXXXXXXXXXXXXX1111 0x0000000F
GPIO_IntToggle       XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntStatus       XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
*****************************************************************
GPIO_GlobalStatus    00000000000000000000000000000000
```

As the FMC-SER0 have restrictions on I/O direction, writing `0xFF` to the GPIO_Control register only changes the direction of GPIO1 and GPIO3 of the channel 0. Writing `0x0A` will have the same effect as only these two GPIOs can be output (please refer to document [H1] for more information). If a specific register bit need to be changed without affecting other bit values of the same register, use a command of the form "`gpio -t REGISTER_NAME.BIT_NAME -b X -c CHANNEL -u`" where "`REGISTER_NAME.BIT_NAME`" is the complete reference of the bit to update with the value specified by "`X`" (0 or 1).

For example to set the third bit of the GPIO_Control register of the channel 0, use the following command:

```
[root@localhost ~]# gpio -c 0 -u -t GPIO_Control.3 -b 0
---------------------------------------------------------------
***************************************************************
Channel: 0 => 4 GPIOs
***************************************************************
GPIO_Control          XXXXXXXXXXXXXXXXXXXXXXXXXXXX0010 0x00000002
GPIO_DataOut          XXXXXXXXXXXXXXXXXXXXXXXXXXXX1000 0x00000008
GPIO_In               XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntMask          XXXXXXXXXXXXXXXXXXXXXXXXXXXX1110 0x0000000E
GPIO_IntPolarity      XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntMode          XXXXXXXXXXXXXXXXXXXXXXXXXXXX1111 0x0000000F
GPIO_IntToggle        XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
GPIO_IntStatus        XXXXXXXXXXXXXXXXXXXXXXXXXXXX0000 0x00000000
***************************************************************
GPIO_GlobalStatus     00000000000000000000000000000000
```


When using bit operations ("`-t -b`"), the GPIO tool reads the content of the register then masks it with the bit to update (according to the `-b` switch), then writes the result to the register thus other register bits remain unchanged by this operation.

To read a register use the "`-g`" command along with the register name as in the following example:

```
 [root@localhost ~]#  gpio -c 0 -g GPIO_Control
GPIO_Control -> 0x00000002
```


Always set the GPIO_IntMode register bits to 1 before enabling interrupt on the GPIO_IntMask register.

5.6.2 GPIO IRQ Driver

This is an event Linux kernel driver which handles interrupts from the GPIO of the FPGA. The driver exposes a `/dev/input/eventX` interface like a keyboard or a joystick using the standard Linux input subsystem. When it receives an interrupt from the GPIO line(s), corresponding key codes are output to the `/dev/event/XXX` device. This driver assume that the GPIO FPGA IP registers are configured. This driver only handles the interrupt reception and not configuration of GPIO FPGA IP registers. Typical use of this driver is to make a blocking read to the `/dev/event/XXX`, when an input_event structure is received (blocking read returns) the GPIO registers can be accessed to read the value of input pins. Each time an interrupt is received, an input_event structure corresponding to the interrupt source is output on the `/dev/input/eventX` device. The LSB GPIO bit of channel 0 triggers a KEY_ESC event then the next GPIO bit triggers an event of KEY_ESC+1. Thus each GPIO bit of each channel has a dedicated EV_KEY value. Please refer to the <linux/input.h> header for complete reference.


The state of the reported key (0 or 1) is irrelevant, only the fact that we receive an input_event structure is relevant.

To know the characteristics of this module, simply run:

```
[root@localhost ~]#  modinfo vita57_gpio_event  (or vita57_gpio_event_vita57 if VM6250)
For example, on a VM6050  board, the result should be similar to :
filename:
/lib/modules/2.6.35.6-11245.ki7.fc14.x86_64/kernel/drivers/input/vita57_gpio_event.ko
license:        GPL
author:         Kontron <support-kom-sa@kontron.com>
srcversion:   D1D2608C364919BB9633ABD
depends:
vermagic:       2.6.35.6-11245.ki7.fc14.x86_64 SMP mod_unload
parm:           debug:set to 1 to enable debugging (int)
parm:           irq_force:set to non 0 value to force IRQ (int)
```

To load the driver:

```
[root@localhost ~]#  modprobe vita57_gpio_event
```

5.6.3 GPIO Function Global Test

In order to test the GPIO function in an exhaustive way, a test tool has been developed called: `gpio_test.py.`
Before running this test, be sure the loop-back cable has been correctly plugged into the FMC SER0 front panel.
Then launch the test by running:

```
[root@localhost ~]#  python /usr/local/bin/vita57/ (or /usr/local/share/vita57/ if VM6250)
```

The result should be:

```
test001_gpio_lines (__main__.gpio_tests) ... ok
test001_serial_lines (__main__.gpio_tests) ... ok
test003_gpio_driver_irq (__main__.gpio_tests) ...ok
Ran 3 tests in ...s
```

Revealing that all the tests run correctly.

Chapter 6 - Additional Information

6.1 Loopback Connector on FMC-SER0 Front Panel

This connector is to be plugged into the FMC-SER0 front panel connector to perform some tests on IOs and check that the hardware is OK.

Pin number (50 pin connector)	Signal	
1	GND1	Not connected
2	TX1	Connected to RX1
3	RX2	Connected to TX2
4	GND2	Not connected
5	TX3	Connected to RX3
6	RX4	Connected to TX4
7	GND3	Not connected
8	TX5	Connected to RX5
9	RX6	Connected to TX6
10	GND4	Not connected
11	TX7	Connected to RX7
12	RX8	Connected to TX8
13	GND5	Not connected
14	TX9	Connected to RX9
15	RX10	Connected to TX10
16	GND6	Not connected
17	TX11	Connected to RX11
18	RX12	Connected to TX12
19	GND7	Not connected
20	TX13	Connected to RX13
21	RX14	Connected to TX14
22	GND8	Not connected
23	TX15	Connected to RX15
24	RX16	Connected to TX16
25	GPIO10	1Kohms to GPIO9
26	GPIO9	Already connected
27	TX16	Already connected
28	GPIO8	1Kohms to GPIO7
29	RX15	Already connected

Pin number (50 pin connector)	Signal	
30	TX14	Already connected
31	GPIO7	Already connected
32	RX13	Already connected
33	TX12	Already connected
34	GPIO6	1Kohms to GPIO5
35	RX11	Already connected
36	TX10	Already connected
37	GPIO5	Already connected
38	RX9	Already connected
39	TX8	Already connected
40	GPIO4	1Kohms to GPIO3
41	RX7	Already connected
42	TX6	Already connected
43	GPIO3	Already connected
44	RX5	Already connected
45	TX4	Already connected
46	GPIO2	1Kohms to GPIO1
47	RX3	Already connected
48	TX2	Already connected
49	GPIO1	Already connected
50	RX1	Already connected

6.2 IO Routing with a VM6250 SBC Board as FMC Carrier

FPGA Pin	FMC Connector Pin from VM6250 (VITA57 / Schematics)	FMC-SER0 Signal from VM6250	FMC-SER0 I/O Name	FMC-SER0 Signal to VM6250	FMC Connector Pin to VM6250 (VITA57 / Schematics)	VM6250 PMCB_IO Signal	VM6250 VME P2 Connector Pin	RTM J14 Pin	Test Loopback Connector on RTM J14
N18	G6 / 57	LA0P	RX1	HA14P	J15 / 149	1	C1	1	to pin 2
M18	G7 / 67	LA0N	RX2	HA14N	J16/159	3	C2	3	to pin 4
L13	D9 / 84	LA1N	TX1	HA16P	E15 / 145	2	A1	2	
M14	D8 / 74	LA1P	TX2	HA16N	E16 / 155	4	A2	4	
H13	H7 / 68	LA2P	RX3	HA17P	K16 / 160	9	C5	9	to pin 10
J14	H8 / 78	LA2N	RX4	HA17N	K17 / 170	11	C6	11	to pin 12
G18	G10 / 97	LA3N	TX3	HA15P	F16 / 156	10	A5	10	
H17	G9 / 87	LA3P	TX4	HA15N	F17 / 166	12	A6	12	
H18	H10 / 98	LA4P	RX5	HA21P	K19 / 190	13	C7	13	to pin 14
J18	H11 / 108	LA4N	RX6	HA21N	K20 / 200	15	C8	15	to pin 16
H16	D12 / 114	LA5N	TX5	HA19P	F19 / 186	14	A7	14	
H15	D11 / 104	LA5P	TX6	HA19N	F20 / 196	16	A8	16	
J17	C10 / 93	LA6P	RX7	HA23P	K22 / 220	21	C11	21	to pin 22
K17	C11 / 103	LA6N	RX8	HA23N	K23 / 230	23	C12	23	to pin 24
K15	H14 / 138	LA7N	TX7	HB5P	E24 / 235	22	A11	22	
J15	H13 / 128	LA7P	TX8	HB5N	E25 / 245	24	A12	24	
L14	G12 / 117	LA8P	RX9	HB1P	J24 / 239	25	C13	25	to pin 26
K14	G13 / 127	LA8N	RX10	HB1N	J25 / 249	27	C14	27	to pin 28
L17	D15 / 144	LA9N	TX9	HB9P	E27 / 265	26	A13	26	
L18	D14 / 134	LA9P	TX10	HB9N	E28 / 275	28	A14	28	
N16	C14 / 133	LA10P	RX11	HB11P	J30 / 299	33	C17	33	to pin 34
M16	C15 / 143	LA10N	RX12	HB11N	J31 / 309	35	C18	35	to pin 36
M15	H17 / 168	LA11N	TX11	HB19P	E33 / 325	34	A17	34	
N15	H16 / 158	LA11P	TX12	HB19N	E34 / 335	36	A18	36	
N13	G15 / 147	LA12P	RX13	HB15P	J33 / 329	37	C19	37	to pin 38
M13	G16 / 157	LA12N	RX14	HB15N	J34 / 339	39	C20	39	to pin 40
P15	D18 / 174	LA13N	TX13	HB21P	E36 / 355	38	A19	38	
P14	D17 / 164	LA13P	TX14	HB21N	E37 / 365	40	A20	40	
R15	C18 / 173	LA14P	RX15 (4)	HB6P	K28 / 280	45	C23	45	to pin 46
R16	C19 / 183	LA14N	RX16 (4)	HB6N	K29 / 290	47	C24	47	to pin 48
N17	H20 / 198	LA15N	TX15 (4)	HB4P	F25 / 246	46	A23	46	
P18	H19 / 188	LA15P	TX16 (4)	HB4N	F26 / 256	48	A24	48	
F16	D20 / 194	LA17P	GPIO1	HA20P	E18 / 175	6	A3	6	1k to pin 8
G16	D21 / 204	LA17N	GPIO2	HA20N	E19 / 185	8	A4	8	
C15	C22 / 213	LA18P	GPIO3	HB3P	E21 / 205	18	A9	18	1k to pin 20
B15	C23 / 223	LA18N	GPIO4	HB3N	E22 / 215	20	A10	20	
A18	H22 / 218	LA19P	GPIO5	HB13P	E30 / 295	30	A15	30	1k to pin 32
A17	H23 / 228	LA19N	GPIO6	HB13N	E31 / 305	32	A16 (1)	32	

FPGA Pin	FMC Connector Pin from VM6250 (VITA57 / Schematics)	FMC-SER0 Signal from VM6250	FMC-SER0 I/O Name	FMC-SER0 Signal to VM6250	FMC Connector Pin to VM6250 (VITA57 / Schematics)	VM6250 PMCB_IO Signal	VM6250 VME P2 Connector Pin	RTM J14 Pin	Test Loopback Connector on RTM J14
A11	G21 / 207	LA20P	GPIO7	HB2P	F22 / 216	42	A20	42	1k to pin 44
A12	G22 / 217	LA20N	GPIO8	HB2N	F23 / 226	44	A21	44	
B11	H25 / 248	LA21P	GPIO9	HB10P	K31 / 310	49	C25	49	1k to pin 50
C11	H26 / 258	LA21N	GPIO10	HB8P	F28 / 276	50	A25	50	
B13	G24 / 237	LA22P	GPIO11	HB10N	K32 / 320	51	C26	51	1k to pin 52
A13	G25 / 247	LA22N	GPIO12	HB8N	F29 / 286	52	A26	52	
B14	D23 / 224	LA23P	GPIO13	HB14P	K34 / 340	53	C27	53	1k to pin 55
A14	D24 / 234	LA23N	GPIO14	HB14N	K35 / 350	55	C28	55	
D13	H28 / 278	LA24P	GPIO15	HB12P	F31 / 306	54	A27	54	1k to pin 56
D14	H29 / 288	LA24N	GPIO16	HB12N	F32 / 316	56	A28	56	
E14	G27 / 267	LA25P	GPIO17	HB17P	K37 / 370	57	C29	57	1k to pin 59
F14	G28 / 277	LA25N	GPIO18	HB17N	K38 / 380	59	C30	59	
E15	D26 / 254	LA26P	GPIO19	HB20P	F37/366	58	A29	58	1k to pin 60
D15	D27 / 264	LA26N	GPIO20	HB20N	F38 / 376	60	A30	60	
E17	C26 / 253	LA27P	GPIO21	HB18P	J36/359	61	C31	61	1k to pin 63
E16	C27 / 263	LA27N	GPIO22	HB18N	J37 / 369	63	C32	63	
F18	H31 / 308	LA28P	GPIO23	HB16P	F34 / 336	62	A31	62	1k to pin 64
F17	H32 / 318	LA28N	GPIO24	HB16N	F35 / 346	64	A32	64	
N12	G18/177	LA16P	DXEN (2)						
M11	G19 / 187	LA16N	DXEN1516 (2)						
G14	G33 / 327	LA31P	LED1R_SW1 (3)						
G15	G34 / 337	LA31N	LED1G_SW2 (3)						
D12	G36 / 357	LA33P	LED2R_SW3 (3)						
E12	G37 / 367	LA33N	LED2G_SW4 (3)						
D18	G30 / 297	LA29P	CAN1_RXD (4)						
D17	G31 / 307	LA29N	CAN1_TXD (4)						
C18	H34/338	LA30P	CAN2_RXD (4)						
B18	H35 / 348	LA30N	CAN2_TXD (4)						
B16	H37 / 368	LA32P	CAN_STB						
A16	H38 / 378	LA32N	TB0201						
			GND	HA18P	J18 / 179	5	C3	5	
			GND	HA18N	J19 / 189	7	C4	7	
			GND	HA22P	J21 / 209	17	C9	17	
			GND	HA22N	J22 / 219	19	C10	19	
			GND	HB7P	J27 / 269	29	C15	29	
			GND	HB7N	J28 / 279	31	C16 (1)	31	
			GND	HB0P	K25 / 250	41	C21	41	
			GND	HB0N	K26 / 260	43	C22	43	

(1):
- P2 pin A16 may be REAR2_RTS+ or HB13N depending on VX6060 equipment
- P2 pin C16 may be REAR1_RTS+ or HA7N depending on VX6060 equipment

(2): With FMC-SER0 PCB A:
- DXEN: enables TXn outputs when set to level 1
- DXEN1516: is used as a MODE signal for all TXn/RXn buffers: RS422 or 485 when set to level 1; RS232 if set to 0

With FMC-SER0 PCB B: these signals control TXn outputs of buffers (mandatory for RS485 mode) :

level 1: enabled

level 0: disabled
- DXEN: TX1,TX2,TX3,TX4,TX5,TX6,TX7,TX8,TX9,TX10,TX11,TX12,TX13,TX14
- DXEN1516: TX15/TX16 pair

(3): With FMC-SER0 PCB A: these signals are only used to drive the LEDs, and the mode of TXn/RXn buffers (232 or 422/485) is set by the MODE signal (DXEN1516)

With FMC-SER0 PCB B: these signals control the LEDs and also override the default mode of TXn/RXn buffers set by the switches :

level 1: RS422 or 485

level 0: RS232
- LED1R_SW1: TX1, RX1, TX2, RX2, TX3, RX3, TX4, RX4
- LED1G_SW2: TX5, RX5, TX6, RX6, TX7, RX7, TX8, RX8
- LED2R_SW3: TX9, RX9, TX10, RX10, TX11, RX11, TX12, RX12
- LED2G_SW4: TX13, RX13, TX14, RX14, TX15, RX15, TX16, RX16

(4): Depending on FMC-SER0 configuration :
- HB6P may be connected to RX15 or CAN1+
- HB6N may be connected to RX16 or CAN2+
- HB4P may be connected to TX15 or CAN1-
- HB4N may be connected to TX16 or CAN2-

MAILING ADDRESS
Kontron Modular Computers S.A.S.
150 rue Marcelin Berthelot - BP 244
ZI TOULON EST
83078 TOULON CEDEX - France

TELEPHONE AND E-MAIL
+33 (0) 4 98 16 34 00
Sales: Order-ATD-Toulon@Kontron.com
Support: GSS-ATD-Toulon@Kontron.com

For further information about other Kontron products, please visit our Internet web site: www.kontron.com.