XIPHOS Q7

Q7 User Manual

Doc No. XSC-1542-6025

May 31th 2021

Submitted To	N/A
Submitted By	Xiphos Systems Corporation 3981 boul. St-Laurent, Suite 500 Montréal QC H2W 1Y5

Xiphos Technologies

Product Document Approval

Title:	**XIPHOS Q7: Q7 User Manual**
XSC Filename:	
Document Type:	**Technical Note**
Contract Number:	***INTERNAL***
Submission Date:	May 31th 2021
Contact Information:	Xiphos Systems Corporation 3981 boul. St-Laurent, Suite 500 Montréal QC H2W 1Y5 email: luq@xiphos.ca tel: 514-847-9474

Product/Document Originator: Lucile Quirion	Signature and Date (signed) May 31th 2021
Product/Document Reviewer: Detlev Casanova	Signature and Date (signed) May 31th 2021

Contents

1 Change History

Version	Release Date	Notes
a	3 December 2019	Initial Release.
b	12 February 2020	Fix Mezzanine connector's pin table
c	14 April 2020	Add remote diagnostic port section
		Update Building software section (drop eSDK)
d	27 July 2020	Add SSH server section
		Detail application watchdog API
		Add USB-to-Ethernet dongle model
e	25 November 2020	Add Health section
		Add Memory Scrubbing (space) section
f	27 January 2021	Add Boot Selection section
		Fix RAM size
g	31 March 2021	Libpa3 programming example
h	18 May 2021	Clear up health monitoring and logging

2 Acronyms

ESD	Electrostatic Discharge	12
MPSoC	Multi-Processor System-on-Chip	31
NOR	Not OR (electronic logic gate)	17
PIM	Product Integration Module	18
QSPI	Queued Serial Peripheral Interface	17
SEU	Single Event Upset	14
VDC	Volts Direct Current	17

3 Applicable Documents

The following documents are considered as an integral part of this document to the extent specified herein.

Ref	Title	Doc No.	Publisher

4 Reference Documents

The following documents provide additional information or guidelines that either may clarify the contents or are pertinent to the history of this document.

Ref	Title	Doc No.	Publisher
RD01	Q7 Specifications	XTI-2001-2019-d	XSC
RD02	Q7_REV_A_16.6.stp (Step model for the Q7 RevA and RevB)	XSC-1542-0101	XSC
RD03	Q7_PIM_REV_A_10L.stp (Step model for the Q7-PIM)	XSC-1542-0102	XSC
RD04	Q7-Hardware-Pinout-Customers	XSC-1542-6015-a01-flr	XSC
RD05	IHLP Standards of Manufacture	First Edition 2008	Vishay Electronics Inductor Divisions
RD06	QFN/DFN Inspection of Solder Joints	Not available	Linear Technology Corporation

5 Introduction

The Q7 is built on the flight and development heritage of the Q4, Q5 and Q6 processors, and is based on a Xilinx Zynq FPGA. Key design objectives for the Q7 included providing a faster CPU, faster RAM and double-precision floating point, while maintaining many of the qualities of the Q6 such as "hybrid" processing, dual SRAM/FLASH FPGA, low power (<1 W) capabilities, and a wide range of available interfaces.

6 Q7 Overview

6.1 Architecture Overview

The high-level architecture of the Q7 is presented in the figure below. Details of the interfaces available on the Q7 are presented in the Q7/Interfaces section of this document.

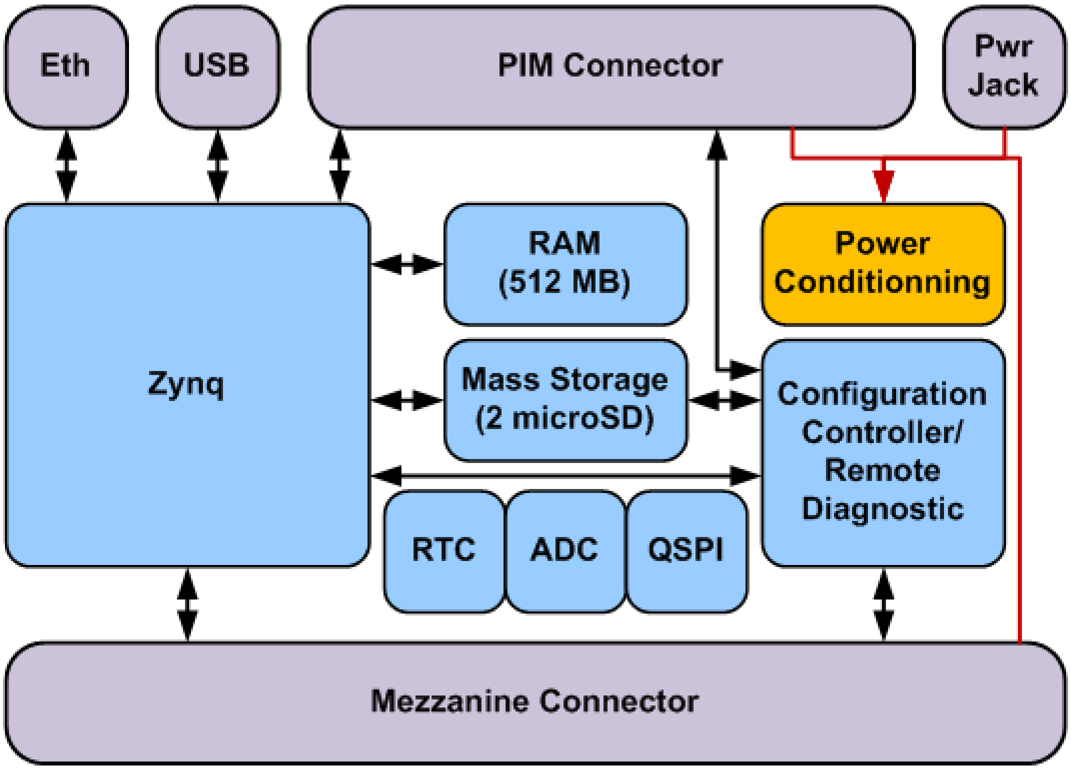


Figure 1: High level architecture

The Q7 is provided with a companion board called the Product Interface Module ("PIM") that is normally included in all sales to new customers. This provides additional features that are not required for space missions but that are helpful for development, debugging, and terrestrial applications. Such features include JTAG connectors, power/debug LEDs, RTC battery, a power on/off buttons, CAN, 1-Wire, USB serial console. The image below is of a PIM connected to a Q7.

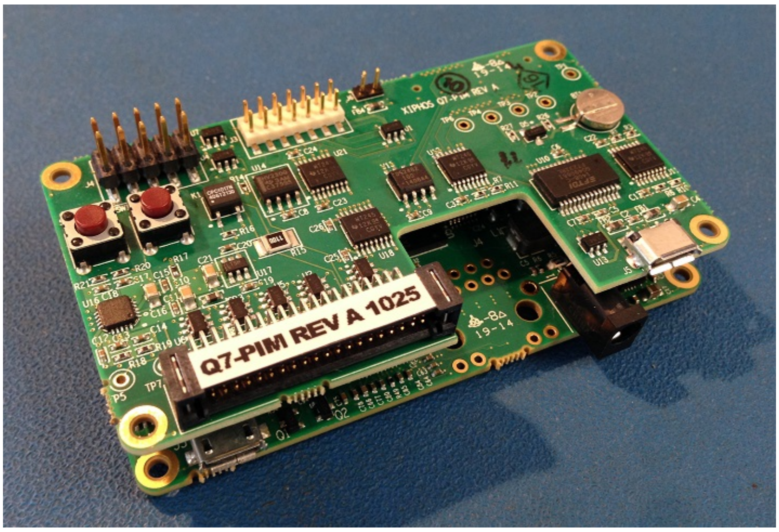


Figure 2: PIM

The Zynq is the "hybrid" processing FPGA (FPGA + CPU).

It provides the core functionality of the system:

- runs the Linux operating system
- controls the interfaces
- runs application programs, processes data, etc.
- houses the custom application logic

The PA3 is the control FPGA.

It is robust to radiation effects. It is used to implement critical features such as:

- power control
- processing unit Firmware configuration
- remote diagnostic port, and
- watchdog timers.

The Q7 can be "stacked": expansion daughterboards can be directly connected to the Q7. Examples of potential generic daughterboards supported by the Q7 include:

- Basic Board
- 1x wide range input power (5.5mm barrel jack)
- 1x gigabit Ethernet (RJ45 with discrete magnetics)

- 1x CAN (DB9)
- 1x USB (type A receptacle) master
- USB hub board:
- 1x Wide range input power (5.5mm barrel jack)
- 8x USB connectors (type A receptacle)
- Ethernet Switch board connectors
- 1x Wide range input power (5.5mm barrel jack)
- 4x gigabit Ethernet (RJ45)
- Video board:
- HDMI in
- HDMI out
- Camera Link (MDR26)
- Camera board:
- FireWire (6-pin plug)
- SpaceWire (microDB9)
- Mass storage board
- No connectors
- $>$ 32 GB storage
- $>$ 100 Mbps throughput

6.2 Summary of Changes

6.2.1 Hardware

Q7 Rev A and Rev B hardware is identical, with one exception: Rev B has 2x128MB QSPI storage space instead of 2x64MB.

6.2.2 Software

Some Q6 customers have asked us for a summary of changes in software architecture. The design and software interfaces for the core operating system have not changed substantially between Q6 and Q7. Most high-level (i.e. not assembler) language code can be compiled for either platform without any change. For example, most Xiphos-developed Q6 code was compiled for Q7, both kernel and userspace, without any code modification. The following list describes the major differences.

- Boot Loader
- Q6: qboot (Xiphos Proprietary, Limited functionality, $<$ 128kB)
- Q7: u-boot (Open Source, Many Features available, $>$ 128kB)
- Operating System
- Q6: Microblaze Linux 3.12
- Q7 1.x: ARM Linux 4.4

- Q7 2.x: ARM Linux 4.14
- Userspace (Root filesystem, task management etc.)
- Q6: Buildroot
- Q7 1.x: Yocto 2.1 (krogoth)
- Q7 2.x: Yocto 2.5 (sumo)
- Toolchain:
- Q6: GCC 4.6.4 (single source, Xilinx)
- Q7 1.x: GCC 5.2.1 (Standard Xilinx SDK version 2016.3)
- Q7 2.x: GCC 7.3.0 (Yocto SDK)
- Debug Facilities:
- Q6: Limited. GDB on non-threaded applications sometimes works, no strace.
- Q7: Standard facilities available. strace, gdb. Good threading support.

This document provides the User Manual for Q7 RevB 2.x devices.

Note that on Q7 RevB 2.x devices, Xiphos has introduced the following library:

- LibXiphos which replace userspace tools used to
- boot on another copy
- update the firmware
- writeprotect the NOR Flash

7 Board Overview

7.1 Electrostatic Discharge Caution

CAUTION! Electrostatic Discharge (ESD) can damage electronic components when they are improperly handled and can result in total or intermittent failures. Always follow ESD-prevention procedures when removing and replacing components.

Disconnect the power source before moving, cabling, or performing any set up procedures. Inappropriate handling may cause damage to the board.

7.2 Board Component Location

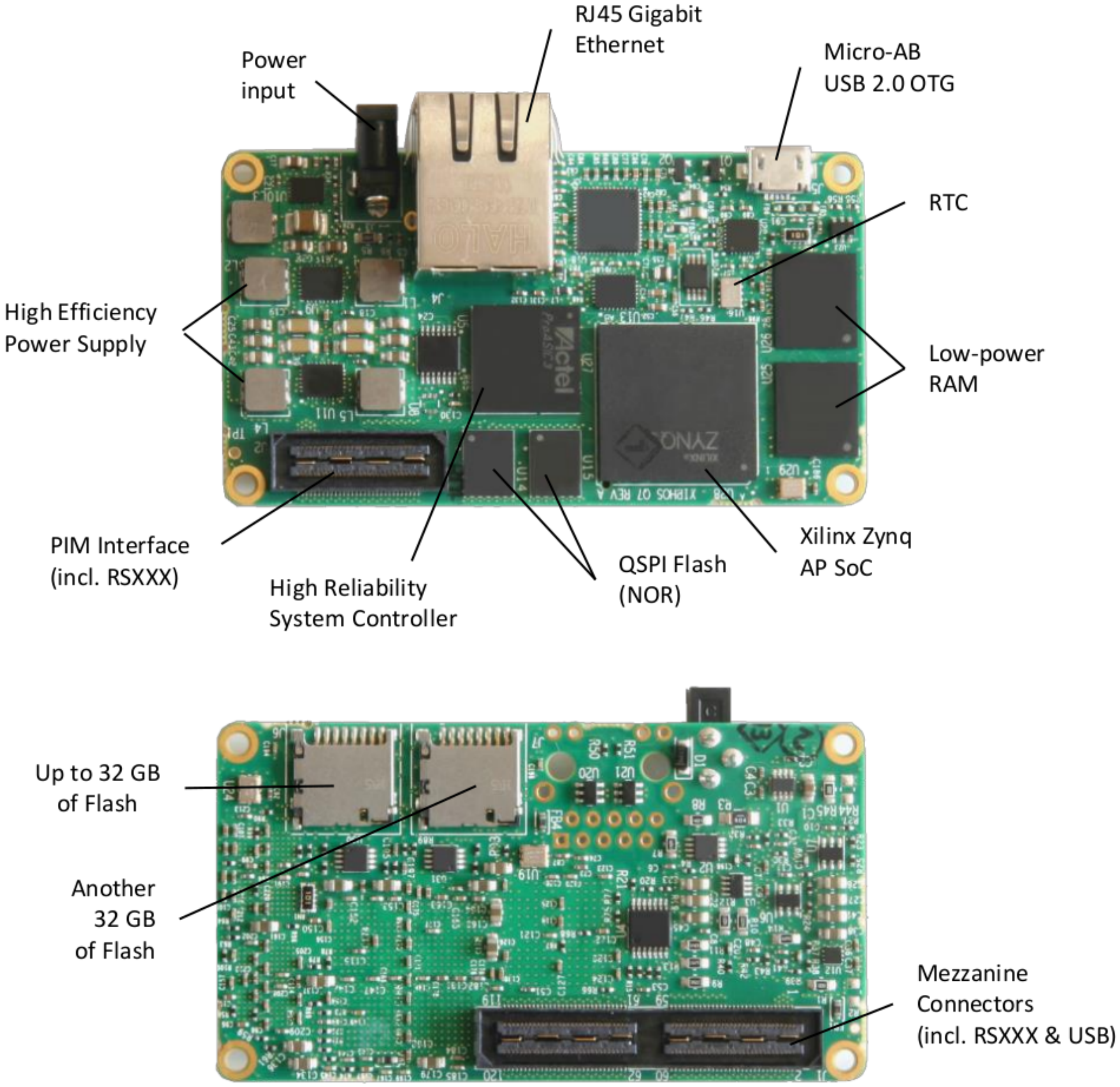


Figure 3: Q7 Board Components

7.3 Q7 Robust Design

The Q7 has been designed to provide robustness against radiation effects without using space grade components. The following features have been incorporated in the design:

- Global latch-up detection and protection circuit
- Dedicated MicroSD latch-up detection and protection circuits
- Dual Firmware mass storage chips
- Dual MicroSD cards
- Dual RAM chips
- Use of non-programmable oscillators (programmable oscillators can be damaged by Single Event Upsets (SEUs))

The global latch-up protection circuits detect overcurrent conditions and automatically power cycle the whole Q7 before damage can occur. The current limit is adjustable.

Two dedicated latch-up protection circuits protect the MicroSD against "low current latch-up" that is possible with NAND FLASH controllers. These over-currents are below the global threshold but can damage the MicroSD if not handled properly.

The Zynq firmware (Logic, bootloader, Linux and user applications) is redundantly stored in the two independent mass storage chips. Therefore the Q7 is operational even if one of the two chips is damaged or its contents are corrupted.

Two independent MicroSD cards are present for data storage. They can be used in a redundant way or independently.

Two RAM chips are connected to the Zynq: one to the PS and one to the PL. In normal operation the PL RAM chip is used only as a local buffer for the logic but it can be used by the Zynq processor if the PS RAM chip is damaged.

Programmable oscillators store information in FLASH or EEPROM bits that can flip under radiation. The Q7 uses non-programmable oscillators to prevent this issue.

7.4 Q7 Hardware Telemetry

In addition to the robustness features, the following telemetry values are available to monitor the Q7 health:

- Input voltage
- Global current consumption
- MicroSD current consumption
- Zynq die temperature
- PCB temperature

8 Q7S "Space" Features

Xiphos has developed multiple robustness enhancements to handle Single-Event Effects (SEE). These are implemented in the Q7S.

8.1 Robust Boot

- Four firmware images are stored in the two 128 MB NOR FLASH devices
- At power-up the ProASIC3 verify image integrity and tries another copy if integrity is compromised
- The boot process is constantly monitored by the ProASIC3, therefore a failed boot is quickly interrupted and the system restarts with a different firmware image
- An integrity check is performed on all code before it is executed

8.2 FLASH scrubbing

- BCH codes are stored in both NOR FLASH devices.
- When requested, the FLASH chips are scrubbed. If an error is present and correctable, it is fixed.

8.3 Health / error monitoring and logging

Multiple Q7 parameters are monitored and available through standard linux interfaces:

- `lm-sensors`
  - Power consumption
  - Die Temperature
  - Input and onboard voltages
  - see `sensors` command for an overview
- `procfs` (`/proc`)
  - CPU load and memory usage: trough `/proc`, or standard commands `top`, `free`, etc.)
  - use `xsc_info_proc` for a summary
- `sysfs` (`/sys`)
  - RAM correctable and uncorrectable bit flips (also logged in journal): `/sys/devices/system/edac/mc/mc0/`
  - RTC temperature: `/sys/class/rtc/rtc0/temperature`
- `journalctl`
  - Logic configuration bit flips (also available through an NNG Publish/Subscribe mechanism)

8.4 Watchdog

A Zynq watchdog is implemented in the ProASIC3.

8.5 RAM ECC

Zynq RAM controller and Linux are configured to use ECC on the main memory.

8.6 Logic Scrubbing

Xilinx SEM IP core is used to detect and correct configuration bit flip errors caused by SEU.

8.7 Console logging

A dedicated buffer is present in the ProASIC3 to log the last 8KB of data output from the Q7 debug port. This information can be queried to diagnose software anomalies.

9 Q7 Environmental

9.1 Q7 Radiation Effects

Xiphos has flight heritage with the architecture and many components of the Q7. Xiphos has also performed two radiation test campaigns on key components of the Q7 and on the Q7 itself. Test reports are available upon request.

In the public domain, Xilinx has published several papers on radiation induced errors in Series-7 parts at each of the last several IEEE Workshop on Silicon Errors in Logic - Systems Effects (SELSE).

- 2012: Single-Event Effects in 28 Nanometer Configuration and Dual-Port RAM Block Memories
- 2013: Soft Error Study of Memory Cells at 28 Nanometers
- 2014: Soft Error Study of ARM SoC at 28 Nanometers

Detailed reliability information, including radiation cross-section, for all Xilinx components is published with quarterly updates in Xilinx Document UG116, Xilinx Device Reliability Report.

9.2 Q7 Thermal Cycling

The Q7 has been formally tested over 200 cycles from -40 to 85°C. Test results are available upon request.

9.3 Q7 TVAC

The Q7 has been formally tested for cold and hot start in thermal vacuum. Test results are available upon request.

9.4 Q7 Shock & Vibration

The Q7 has been formally tested to shock and vibration targets based on a composite test profile generated by the University of Toronto Institute for Aerospace Studies Space Flight Laboratory.

This encompasses the requirements of the Polar Satellite Launch Vehicle (PSLV), DNEPR, Soyuz-Fregat (in various configurations), and Vega launch vehicles.

Test results are available upon request.

10 Quick Start

10.1 Board Setup

The Q7 comes with its firmware installed on Queued Serial Peripheral Interface (QSPI) Not OR (NOR) Flash.

- Connect the 6 Volts Direct Current (VDC) power supply.

The Q7 board is now ready to be powered on and off.

10.2 USB-to-Ethernet Dongle

As the RJ-45 connector is often unpopulated, especially for flight boards, a USB-to-Ethernet dongle can be used to connect the Q-card to the network and transfer files.

The driver for the Linksys USB3GIGV1 USB-to-Ethernet device is included in the kernel of the Q-card, as it is used for factory-testing.

To reduce the rootfs size, most kernel modules are not installed and loaded, as such, other USB-to-Ethernet dongles are not supported by default.

For more information on how to rebuild the kernel and its modules, refer to the article Install Yocto SDK.

Note that USB is powered off by default, to power it on, refer to the article Enabling USB and SD Cards.

10.3 Host workstation configuration

For development and testing purpose, we recommend that you use Ubuntu 18.04. The following tools are used for testing purposes: picocom, ssh, etc.

10.4 Booting into Linux

- Connect the Q7-Product Integration Module (PIM) micro USB port to your host machine. You can get the port information with:

```
$ dmesg | tail | grep tty
[4749833.015233] usb 1-1.3.4.1: FTDI USB Serial Device converter now attached to
     ttyUSB4
```

- Run the following command to start a tty session on the target, press Enter and Login (login: root, no password):

```
$ picocom -b 115200 /dev/ttyUSB4

q7-revb login: root
```

- Print firmware version and system information

  ```
  $ cat /etc/os-release
  [...]
  $ uname -a
  [...]
  ```

10.5 Transferring Files

The easiest way to transfer files from/to your workstation is to use the scp command, assuming that the Q7 has a network connection.

Use the following command to verify that the Q7 is connected to the network:

```
$ ip addr show
```

Then from your workstation, copy files to the /tmp directory on target with:

```
$ scp FILE root@${IP_ADDR}:/tmp
```

10.6 Enabling USB and SD Cards

To save on power consumption, the Q7 USB port as well as its SD Cards are not powered by default. They must be manually powered in order to use any device connected to them.

10.7 Enabling USB

The USB port is turned ON by using "q7hw" command. The command usage for USB can be shown by typing "q7hw usb help".

Examples for powering ON USB:

```
q7hw usb set on
```

Note that this script is non-blocking (i.e. it does not wait for devices to show up). Any USB device connected will be detected by the kernel and listed at that point (assuming they are supported).

The port can be turned by OFF with the same command:

```
q7hw usb set off
```

10.8 Enabling SD Cards

The micro-SD cards are turned ON by using "q7hw" command. The command usage for SD cards can be shown by typing "q7hw sd help".

Examples for powering ON SD cards:

```
# Turning on SD Card 0
q7hw sd set 0 on
# Turning on SD Card 1
q7hw sd set 1 on
# Turning both cards on at once
q7hw sd set all on

# At this point, the card partitions will appear at the path /dev/mmcblk<SDCARD>
    p<PARTITION_NUMBER>
# They can be mounted with a command like "mount /dev/mmcblk0p1 /mnt"
```

The card can be turned off with the same commands:

```
# Turning off SD Card 0
q7hw sd set 0 off
# Turning off SD Card 1
q7hw sd set 1 off
# Turning both card off at once
q7hw sd set all off
```

It is safer to unmount the partitions before turning off the SD cards.

Note that the following errors will show up on the console while performing those commands:

```
mmc0: Got command interrupt 0x00030000 even though no command operation was in
    progress.
mmc0: Got command interrupt 0x00030000 even though no command operation was in
    progress.
mmc0: Got command interrupt 0x00030000 even though no command operation was in
    progress.
mmc0: Got command interrupt 0x00030000 even though no command operation was in
    progress.
mmc0: Got command interrupt 0x00030000 even though no command operation was in
    progress.
...
```

These lines are normal and are to be expected. They can be safely ignored.

10.9 Persistent Write

10.9.1 Flash

Note that QSPI chips are mounted read-only by default. For applications to be permanently installed on the QSPI or to perform persistent modifications of the root file system, the write protection will have to be disabled during the writes and enabled once the writes have finished.

Please look at the section Writeprotect to understand how to enable or disable the protection. Please look at the section Ugrading the image to see how to update various components. Please look at the section Updating Files to understand how to update files on the root file system.

11 Building Software Components

11.1 Prerequisites

This chapter assumes that you are:

- Experienced with embedded software development
- Familiar with ARM architecture
- Familiar with Xilinx development tools such as the Vivado Integrated Design Environment (IDE), the Xilinx software developers kit (SDK), compilers, debuggers, and operating systems.

11.2 Install Yocto SDK

Along with Q-Cards, Xiphos provides a Software Development Kit that enables customers to cross-compile binaries for the Q-card from a Linux Host. First, the SDK has to be installed on a Linux host machine. This is done by running the SDK installation script.

```
$ ./ark-glibc-x86_64-xsc-image-minimal-<ARCH>-toolchain-nodistro.0.sh
ark (Xiphos Systems Reference Distribution) SDK installer version nodistro.0
==========================================================================
Enter target directory for SDK (default: /opt/xiphos/sdk/ark):…
[]
```

You can then load the build environment and tools with the following command:

```
source /opt/xiphos/sdk/ark/environment-setup-<ARCH>-xiphos-linux
```

11.3 Building the Kernel

11.3.1 Downloading the kernel

The kernel can be downloaded from:

https://github.com/XiphosSystemsCorp/linux-xlnx/tree/xiphos-4.14

Note that the Xiphos kernel is based on the Xilinx fork of the Linux kernel https://github.com/Xilinx/linux-xlnx

You can use git to download the latest version of the kernel:

```
git clone https://github.com/XiphosSystemsCorp/linux-xlnx.git --branch xiphos
    -4.14
```

11.4 Getting the Kernel Configuration

The kernel configuration may change depending on what the Q-card supports. The easiest way to obtain the correct configuration is to extract it from your Q-card.

To extract this configuration, you must boot your Q-card to its Linux system and type the following command:

```
# On your Q-card
$ zcat /proc/config.gz > /tmp/kernel-config
```

The easiest way to transfer this file to your workstation is to use the scp command, assuming your workstation has SSH and that your Q-card has a network connection.

11.4.1 Preparing the kernel to cross-compile

Before starting the compilation, you need to configure the kernel's various configuration options.

There are several ways to do this. The easiest is to use `make menuconfig` to bring up the configuration menu.

Note that you MUST have the toolchain installed for this to work; you also need to have its "Development Environment Variables" set up before proceeding. Refer to the article Install Yocto SDK on how to do this.

The following assumes that you used git to clone the kernel in the "linux-xlnx" folder. It also assumes that the kernel configuration copied from the Q-card is now in /tmp/kernel-config on your workstation.

```
# Assuming you installed the SDK in /opt/xiphos/sdk/ark
# Source the SDK environment setup script
$ . /opt/xiphos/sdk/ark/environment-setup-<ARCH>-xiphos-linux
# Assuming you cloned your kernel in linux-xlnx
$ cd linux-xlnx
$ mv /tmp/kernel-config .config
$ make menuconfig
```

You can make the modifications you need in this menu; once you are done, just "Save" your configuration and quit.

11.4.2 Compiling the kernel

Once the configuration is done, you can compile your kernel by using the command:

```
# In the "linux-xlnx" folder you cloned the kernel in
make LDFLAGS="" Image
```

If the compilation goes well, an "Image" file will be created.

The Image file is the uncompressed kernel image.

You can use the following command to build the uImage:

```
make LDFLAGS="" UIMAGE_LOADADDR=0x8000 uImage
```

11.5 Compiling a Kernel Module

Optionally, you can also build the kernel modules by using the following commands:

```
# Assuming you installed the SDK in /opt/xiphos/sdk/ark
# Source the SDK environment setup script
. /opt/xiphos/sdk/ark/environment-setup-<ARCH>-xiphos-linux

# Force to install the modules in the folder "/tmp/modules" locally
export INSTALL_MOD_PATH=/tmp/modules

# In the "linux-xlnx" folder where you compiled the kernel previously
make LDFLAGS="" modules
make LDFLAGS="" modules_install
```

At this point, the modules will be present in the folder "/tmp/modules".

You can use the following commands to create a modules.tgz file.

```
# Make sure the modules folder match the kernel version
# This assumes the modules are installed in the "modules" local folder
mv custom_modules/lib/modules/4.14.0-xilinx-* modules/lib/modules/4.14.0-xiphos

# Create the .tgz from "modules" local folder
tar -C /tmp/modules -czf modules.tgz .
```

Modules need to be transferred manually to the Q-card, for example by using the Ethernet port.

11.6 Compiling Userspace Application Programs

Load the build environment from the SDK (see Install Yocto SDK).

```
# Assuming you installed the SDK in /opt/xiphos/sdk/ark
# Source the SDK environment setup script
$ . /opt/xiphos/sdk/ark/environment-setup-<ARCH>-xiphos-linux

$ make hello_world
```

The application is now ready to be transferred and tested on the Q-card.

11.7 Building IP Cores

Refer to Logic Design Overview for information on how to generate and install IP Cores on the Q-card.

12 XDI Image Format

A standard Xiphos image file is called a "Xiphos Deployment Image" and has the .xdi file extension. It contains a single copy of the image.

The whole image is a single tar archive, compressed and containing at least the following files:

- A devicetree blob, with a U-Boot header (dtb-xiphos.img),
- A bootable Xilinx firmware (fw.bin), this is the processing unit's entry point. It is composed of system.bit, fsbl.elf and u-boot.elf package.
- U-boot environment (q7-env.bin)
- Initramfs minimal Linux system (q7-initramfs.img)
- A compressed kernel image file (uImage),
- A ubi image file (rootfs.ubi) with a single volume containing a ubifs root filesystem.

These files are also added to the XDI image for debugging purpose:

- Xilinx bootloader (fsbl.elf)
- system.bit (Xilinx generated .bit information)
- A U-Boot image (u-boot.elf)

These files can be extracted with the tar command:

```
tar -C /WHERE/TO/EXTRACT -xJf /WHERE/IMAGE/IS.xdi
```

13 Upgrading the image

13.1 Partition Layout

The Q-card has two NOR Flash chips with 256 MB per device, for a total of 512 MB. In space configuration, each chip is also divided in two 128 MB sections. There are 2 chips, each with two copies:

- qspi0 nominal
- qspi0 gold
- qspi1 nominal
- qspi1 gold

13.2 Boot Selection

After a power cycle or a reboot, the Q-card will automatically attempt to boot qspi0 nominal. In case the Q-card fails to boot into Linux, the boot watchdog in the ProASIC3 will kick in and a second attempt to boot the copy is done. If the boot fails twice, the Q-card assumes that the copy is corrupted and attempt to boot another copy.

Here is the order in which booting is attempted: qspi0 nominal, qspi1 nominal, then qspi0 gold and finally qspi1 gold.

13.3 Manual Boot Selection

To manually force a boot on a specified copy, the tool `xsc_boot_copy` should be used:

It has the following interface:

```
$ xsc_boot_copy -h

Usage: xsc_boot_copy [OPTIONS] [<chip> <copy>]

With 0 arguments return currently booted chip and copy.
With 0 arguments and -r reboot current copy.
With 2 arguments reboot on the specified copy

Possible combinations
0 0 -> qspi0-nom
0 1 -> qspi0-gold
1 0 -> qspi1-nom
1 1 -> qspi1-gold

Options:
  -h, --help:       display this message and exit
  -v, --version:    display the version and exit
  -r, --reboot:     reboot the current copy
  -f, --force:      force boot even if not writeprotected
  -w, --writeprotect:   writeprotect before booting
```

To find out which copy is currently booted, execute the tool with no arguments:

```
$ xsc_boot_copy
0 0
```

To reboot on the same copy:

```
$ xsc_boot_copy -r
re-booting in 3 2 1
```

To reboot on any copy:

```
$ xsc_boot_copy 1 0
re-booting in 3 2 1
```

The tool will first verify that the copy is write-protected. In case it is not, by default, it will print a warning and abort:

```
$ writeprotect 0 0 0
[  271.677153] UBIFS (ubi0:0): background thread "ubifs_bgt0_0" started, PID
   3329
$ xsc_boot_copy -r
Warning! Chip 0, copy 0 is unlocked!
```

By using the “-f” option, the tool can force the reboot anyway.

By using the “-w” option, the tool can write-protect the target image prior to rebooting:

```
$ xsc_boot_copy -w 0 0
Warning! Chip 0, copy 0 is unlocked!
[  422.002492] UBIFS (ubi0:0): background thread "ubifs_bgt0_0" stops
Warning: cannot detach ubi0
It is probably in use
md5: read 4194304 out of 56492032 bytes
md5: read 8388608 out of 56492032 bytes
md5: read 12582912 out of 56492032 bytes
md5: read 16777216 out of 56492032 bytes
md5: read 20971520 out of 56492032 bytes
md5: read 25165824 out of 56492032 bytes
md5: read 29360128 out of 56492032 bytes
md5: read 33554432 out of 56492032 bytes
md5: read 37748736 out of 56492032 bytes
md5: read 41943040 out of 56492032 bytes
md5: read 46137344 out of 56492032 bytes
md5: read 50331648 out of 56492032 bytes
md5: read 54525952 out of 56492032 bytes
md5: read 56492032 out of 56492032 bytes
re-booting in 3 2 1
```

13.4 Updating the firmware

Once the Q-card has booted, the XDI file needs to be transferred to the Q-card to be installed (e.g. in /tmp/xsc-image*.xdi). Once transferred, the flash can be updated using the `update_xdi` tool. The user must select:

- `bootloader` indicating the firmware
- the target NOR chip (0 or 1)
- the target copy to update: 0 for nominal, 1 for gold
- the source XDI file

```
$ update_xdi bootloader 0 0 /tmp/xsc-image*.xdi # This will update the chip 0
    nominal firmware.
$ update_xdi bootloader 1 0 /tmp/xsc-image*.xdi # This will update the chip 1
    nominal firmware.
$ update_xdi bootloader 0 1 /tmp/xsc-image*.xdi # This will update the chip 0
    gold firmware.
$ update_xdi bootloader 1 1 /tmp/xsc-image*.xdi # This will update the chip 1
    gold firmware.
```

For more information about `update_xdi` see LibXiphos Tools - update_xdi.

13.5 Updating the rootfs

Similar to the firmware, the Linux root flash partition can be updated using the `update_xdi` tool.

Note: The rootfs of an active copy cannot be updated. If your Q-card has booted normally, you cannot update rootfs on nominal 0. This operation has to be done from another copy (i.e. nominal 1). More generally, if the target flash partition is attached to a ubi device, that flash partition must first be detached. In this case the `update_xdi` tool will exit without performing the update.

```
$ update_xdi rootfs 0 0 /tmp/xsc-image*.xdi # Update the chip 0 nominal rootfs.
$ update_xdi rootfs 1 0 /tmp/xsc-image*.xdi # Update the chip 1 nominal rootfs.
$ update_xdi rootfs 0 1 /tmp/xsc-image*.xdi # Update the chip 0 gold rootfs.
$ update_xdi rootfs 1 1 /tmp/xsc-image*.xdi # Update the chip 1 gold rootfs.
```

For more information about `update_xdi` see LibXiphos Tools - update_xdi.

13.6 Updating the whole image

Running `update_xdi` using `all` will sequentially update a copy completely.

Note: as for the rootfs case, the currently running copy cannot be updated.

```
$ update_xdi all 0 0 /tmp/xsc-image*.xdi # Update the chip 0 nominal copy.
$ update_xdi all 1 0 /tmp/xsc-image*.xdi # Update the chip 1 nominal copy.
$ update_xdi all 0 1 /tmp/xsc-image*.xdi # Update the chip 0 gold copy.
$ update_xdi all 1 1 /tmp/xsc-image*.xdi # Update the chip 1 gold copy.
```

For more information about `update_xdi` see LibXiphos Tools - update_xdi.

13.7 Updating the Bitstream, device-tree or kernel

Note: Unlike the rootfs partition case, these files are not in use when Linux is running, since they have already been read completely into memory. As such they can be updated on the flash copy currently booted.

Note: All files can be updated independently but it should be done with caution. If unsure, please update all three files using compatible versions.

Example updating the kernel:

```
$ update_xdi kernel 0 0 /tmp/xsc-image*.xdi # Update kernel on chip 0 nominal.
```

Example updating the device-tree:

```
$ update_xdi devicetree 0 0 /tmp/xsc-image*.xdi # Update device-tree on chip 0
    nominal.
```

Example updating the logic:

```
$ update_xdi logic 0 0 /tmp/xsc-image*.xdi # Update logic on chip 0 nominal.
```

For more information about `update_xdi` see LibXiphos Tools - update_xdi.

13.8 Updating files

To properly mount a flash copy root partition for writing, these steps must be performed in the correct order:

1. Turning off the NOR write protection on that copy only
2. Attaching (if not already attached) the MTD to a UBI device
3. Creating a mountpoint if necessary, and mounting, or, in case it is mounted already, remounting the UBIFS filesystem as read-write.

To facilitate executing these steps, the `xsc_mount_copy` tool is available.

Example mounting the copy “qspi1-gold”:

```
$ xsc_mount_copy 1 1
```

This can be done when booted from any copy, including the one to be mounted.

The user can then access the filesystem at the location specified in the output of the tool. Typically: /tmp/mnt<...>/

Once all the modifications have been performed, it is necessary to reverse the previous steps before rebooting. This is achieved using `writeprotect` with 1 as the lock argument.

Example locking the copy `qspi1-gold`

```
$ writeprotect 1 1 1
```

13.9 Locking and Unlocking a NOR copy: writeprotect

The `writeprotect` command can lock/unlock a flash copy:

The flash copy is specified as for the other commands in the first two arguments. The third argument is optional. Without it, the command returns the state of the copy: locked or unlocked. With a third argument set to 0, the command unlocks the flash copy, and with 1 it locks it.

```
$ # Get the state:
$ writeprotect 0 0 # qspi0-nom
$ writeprotect 1 0 # qspi1-nom
$ writeprotect 0 1 # qspi0-gold
$ writeprotect 1 1 # qspi1-gold

$ # Unlock:
$ writeprotect 0 0 0 # qspi0-nom
$ writeprotect 1 0 0 # qspi1-nom
$ writeprotect 0 1 0 # qspi0-gold
$ writeprotect 1 1 0 # qspi1-gold

$ # Lock:
$ writeprotect 0 0 1 # qspi0-nom
$ writeprotect 1 0 1 # qspi1-nom
$ writeprotect 0 1 1 # qspi0-gold
$ writeprotect 1 1 1 # qspi1-gold
```

The `writeprotect` command ensures that the rootfs partition is unmounted or remounted read-only before locking the flash, such that any pending operating system cached operations are flushed to the NOR. This may take some time.

The `writeprotect` command computes the MD5 hash of the rootfs partition of the specified copy when locking. It then updates the xscinfo partition of the copy with that md5sum. This takes some time.

If the rootfs filesystem is mounted RW, the MD5 of the underlying NOR partition changes due to the internal thread of UBIFS. It is required to `writeprotect` the copy prior to booting or power cycling.

If you forget to `writeprotect` the copy prior to booting, the rootfs checksum will not be recalculated and stored. Therefore, on the next boot the Q-card will think that the image is corrupted and try to boot a different copy. For example, if you make the mistake on “qspi0 nominal” the Q-card will boot “qspi1 nominal”. Once booted on “qspi1 nominal” you can `writeprotect` “qspi0 nominal”. Then the next boot will be from “qspi0 nominal”.

14 How To

14.1 Enter U-Boot

Assuming your Q-card is in working order, you can use the following steps to get into U-Boot.

- Connect to the Q-card using your favorite serial terminal tool (e.g., picocom).
- Power cycle (turn off/turn on) your Q-card. You should see u-boot output:

When the prompt “Hit any key to stop autoboot:” appears, hit “space”

Quickly type: `run config_done`. You have about 3 seconds for performing all these steps. Restart at step 2 if you missed. The unit is now booted into U-Boot.

14.2 Add a root password

The system supports multiusers and passwords. However, the tools to add users, manage groups and passwords are not installed by default.

You can edit the Linux standard file /etc/shadow to change the root password.

The /etc/shadow file stores actual password in encrypted format with additional properties related to the user's password.

14.3 Permanent SSH key

At each boot, a new SSH key is generated. See SSH Server for instructions on how to use a persistent key.

15 ProASIC3 Features

15.1 Instant-On

The ProASIC3 can be used to provide 'instant-on' functionality for customers' payloads. The device is a ProASIC3-1000 and there is about 50% of the capacity available for mission-specific use. This type of functionality would be managed and integrated by Xiphos with input (as needed) from the customer. Examples of functionality that Xiphos has previously included are:

- CAN bus communication (CANOpen)
- Radio payload
- Closed loop thermal control
- Threshold detection/fail-safe triggering

15.2 Remote Diagnostic Port

The ProASIC3 provides a remote diagnostic port that can be used to:

- Monitor temperature and power consumption
- Manually configure the Zynq (provide a safe mode)
- Read from and write to QSPI FLASH
- Access scratch and circular buffers (if Space functionality is present)

By default the remote diagnostic port is on the Q-card PS0 port and the RSXXX port, but it can be configured to be connected to any interface used between the Q-card and satellite OBC.

For more information, please refer to the Remote Diagnostic Port section.

15.3 Resistance to Single Event Upset (SEU)

The ProASIC3 logic is highly robust against SEU because:

- It is immune to configuration bit upsets
- Logic is compiled in TMR mode (if Space functionality is present)

For this reason, it is used to implement critical Q-card functionality:

- Zynq robust configuration
- Serial backdoor access for remote diagnostic and firmware update
- Watchdog

It can also be used to implement critical user logic which would not be well-suited for the Zynq. To do so, custom logic must be designed and compiled. Currently this can only be performed by Xiphos. Therefore, for any customization request, please contact us.

16 Logic Design Overview

The Zynq UltraScale+ is the Q-card's processing FPGA. It provides the processing and logic resources required to implement mission-specific software and logic.

Custom Zynq US+ logic can be developed by Xiphos from customers' requirements or directly by customers themselves. This page explains how a user can generate and test logic for the Q-card.

16.1 Requirements

In order to generate custom IP Cores, please refer for instructions to Xilinx Zynq UltraScale+ Multi-Processor System-on-Chip (MPSoC) & Zynq-7000 AP SoC Design Flow Wiki: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841738/Getting+Started

The Xilinx SDK is free and can be downloaded from the Xilinx website https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2018-2.html. The Q-card requires Vivado v2018.2. Other versions may work but are not supported by Xiphos.

Building all Q-card boot files requires a Linux host with the following tools:

- The v2018.2 Xilinx SDK (installed with Vivado)
- The devicetree compiler, `dtc`
- The `mkimage` tool (from the U-Boot project).

16.2 Logic IP Core Design

Any software or method can be used to design logic IP cores as long as the following criteria are respected:

- If the IP core needs to communicate with the Zynq US+ processor, it must have an AXI slave interface
- If the IP core requires direct Q-card RAM access, it needs an AXI master interface
- If an interrupt line is required, it must follow the Zynq US+ interrupt rules

16.3 Generating Q-card files

Xiphos provides customers with a skeleton Vivado project which contains:

- A Vivado project and delivered IP Cores
- A constraint file
- Makefiles
- FSBL source code

The Skeleton project is shipped as a zip file and can be extracted with the following command:

```
$ unzip -q ${PROJECT_NAME}_skeleton.zip
$ cd ${PROJECT_NAME}
```

The Vivado project can be generated with the following command:

```
$ source /opt/Xilinx/SDK/2018.2/settings64.sh
$ cd ${PROJECT_NAME}/XilinxVivado/outputs/
$ make project
```

You should now be able to open the Vivado project file:

```
$ source /opt/Xilinx/SDK/2018.2/settings64.sh
$ cd ${PROJECT_NAME}/XilinxVivado/outputs/
$ make gui
```

Or open `${PROJECT_NAME}/XilinxVivado/${PROJECT_NAME}/${PROJECT_NAME}.xpr` with Vivado

After making changes to the project, export the hdf file (including the bitstream) to the outputs directory and run the following command to generate the files

```
$ source /opt/Xilinx/SDK/2018.2/settings64.sh
$ cd ${PROJECT_NAME}/XilinxVivado/outputs/
$ make fw.bin devicetree.img system.bit
$ tar cJf skeleton.xdi fw.bin devicetree.img system.bit
```

The generated skeleton.xdi file can be used to update the bootloader, kernel, logic and devicetree binaries.

We also provide a Makefile within the release archive which can be used to regenerate a full image.xdi

```
# Extract the release archive
$ mkdir ${PROJECT_DIR}
$ tar -C ${PROJECT_DIR} -xf xsc-release-[...].tar
$ cd ${PROJECT_DIR}
# Display the makefile usage
$ make help
Usage:
------
  calling `make image.xdi` with no other action will extract
  necessary files from the release XDI and rebuild a rootfs
  based on the release root filesystem.
  If you wish to use your own files when rebuilding the XDI,
  simply copy the files in the current directory using the
  appropriate name.
    - firmware: fw.bin
    - devicetree: devicetree.img
    - Kernel: Image.gz
    - FPGA bitstream: system.bit

  If you wish to append custom files to the root filesystem
  create a tar.xz archive containing all files with appropriate
  paths (see creating userfs.tar.xz below) and run

    make USERFS=${YOUR_TAR_FILE}.tar.xz image.xdi
[...]
```

A full step-by-step example to generate a custom 'overlayfs' is provided in the Makefile help.

See LibXiphos Tools - update_xdi for more information on how to install these files on a Q-card.

17 Software Features

17.1 Software Overview

The following list describes the major software components:

- Boot Loader: Based on Xilinx U-boot
- Linux Kernel: Based on Xilinx Linux 2018.02 (Linux 4.14 LTS)
- Operating System: Yocto 2.5 (sumo)
  - busybox, executable which provides many common UNIX utilities
  - systemd, init system and system processes manager
- Toolchain: GCC 7.3.0 (Yocto sumo SDK)
- PA3 programmable logic: Custom
- Zynq programmable logic: Custom

17.2 Features Overview

The Q-card comes in two different variations: ground (the default) and space.

For convenience purposes, the space configuration is a superset of the ground configuration. This means that it has everything the ground configuration provides plus extra features.

The default configuration includes:

- A robust boot and application watchdog
- A fallback mechanism implemented via:
  - Two redundant firmware images
  - Firmware image data corruption detection code
- A robust firmware update mechanism

The extra features include:

- Four redundant firmware images
- A remote diagnostic port
- Triple-mode redundancy in the PA3 logic
- ECC'd RAM
- A SEM-IP core
- A Safe Mode
- A UART buffer
- A scratch buffer
- QSPI FLASH scrubbing (available on demand)
- RAM scrubbing (available on demand)

17.3 Watchdog

The Q-card provides a software watchdog implemented in the ProASIC3 logic. If it detects a software hang-up or other abnormal behaviour it reboots the Q-card.

The Zynq watchdog is a feature implemented by:

- the ProASIC3 logic
- a generic Linux kernel driver developed by Xiphos
- the standard Linux kernel watchdog API
- and a C interface in libxiphos

The standard Linux kernel watchdog API usable from user-space is described in: `<kernel repository >/Documentation/watchdog/watchdog-api.txt.`

Watchdog duration is user selectable between 0.001 and 171 seconds.

17.3.1 IOCTL Watchdog Standard API

The standard Watchdog API uses IOCTLs to interface with the kernel and must be called on the file `/dev/xsc_logic_wdt_v1_0`.

The following standard IOCTLs are supported by our driver:

- `WDIOC_GETSUPPORT`: Information and supported options
- `WDIOC_GETSTATUS`: Gives the `RESET` status (this is used to know if the watchdog had to reboot the Q-card)
- `WDIOC_GETBOOTSTATUS`: Same as `WDIOC_GETSTATUS`
- `WDIOC_SETOPTIONS`: Used to enable or disable the watchdog
- `WDIOS_ENABLECARD`
- `WDIOS_DISABLECARD`
- `WDIOC_KEEPALIVE`: This will reset the watchdog counter (this ioctl is used to tell the WD that the application is still alive)
- `WDIOC_SETTIMEOUT`: Set the timeout value.
- `WDIOC_GETTIMEOUT`: Get the current timeout value
- `WDIOC_GETTIMELEFT`: Get the time left before the watchdog resets the board.

For more information on the IOCTLs, please refer to the Linux kernel watchdog API documentation linked above.

17.3.2 Xiphos Extra Watchdog Configurations

Xiphos also provides watchdog information that are not available through the standard API through the sysfs.

- `status` (Read only): Used to check the state of the watchdog. It returns a 3-bits value (0-7).
- Bits 0-1: indicate the status. b11 means that the watchdog is enabled. All other values mean disabled.
- Bit 2: indicate the nowayout status. 1 means enabled, 0 means disabled.
- `nowayout` (Write only): Usually, the nowayout configuration is set at compile time. To ease testing we provide an API to set the `nowayout` value. Writing 1 to this file enables the `nowayout` feature. It cannot be deactivated unless the system is rebooted.
- `timeout`, `counter`: They can be accessible through this API but it is recommended to use the standard Watchdog API.
- `magic_close`: This is used to tell the driver to stop the watchdog when the file is closed.

It is used to make the driver stop the Watchdog when the file is closed. This is equivalent to setting the `WDIOS_DISABLECARD` option then closing the file. It is triggered by writing `V` in the watchdog file.

17.3.3 libxiphos' watchdog C interface

Xiphos provides libxiphos which provides helper functions to interface with the watchdog APIs.

These are the provided functions:

```
/**
 * \brief Take control of the watchdog and activate it
 * \param handle   Returned handle
 * \return 0 on success, != 0 on failure
 * */
int xsc_watchdog_init(struct watchdog_s **handle);

/**
 * \brief Release the watchdog handle: only stops the watchdog if
 * magic close is set to 1
 *
 * \param handle The handle to release
 * \param magic_close set to 1 to also stop the watchdog
 * \return 0 on success, -1 otherwise
 **/
int xsc_watchdog_close(struct watchdog_s *handle, int magic_close);

/**
 * \brief Keep the watchdog alive
 *
 * \param handle The handle to the initialized watchdog
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_keepalive(struct watchdog_s *handle);

/**
 * \brief Disable the watchdog
 *
 * \param handle The handle to the initialized watchdog
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_disable(struct watchdog_s *handle);

/**
 * \brief Enable the watchdog
 *
 * \param handle The handle to the initialized watchdog
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_enable(struct watchdog_s *handle);

/**
 * \brief Set watchdog timeout after which the ProASIC will
 * reset the SoC
 *
 * \param handle The handle to the initialized watchdog
 * \param timeout_seconds The number of seconds after which the
 *                        watchdog will reset
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_set_timeout(struct watchdog_s *handle, int timeout_seconds);

/**
 * \brief Get watchdog timeout after which the ProASIC will
 * reset the SoC
 *
 * \param handle The handle to the initialized watchdog
 * \param timeout_seconds The number of seconds after which the
 *                        watchdog will reset
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_get_timeout(struct watchdog_s *handle, int *timeout_seconds);

/**
 * \brief Set the no-way-out flag which will prohibit disabling the
 * watchdog
 *
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_set_nowayout();

/**
 * \brief Get the status (enabled or disabled) and nowayout flag (set or not)
 *
 * \param nowayout  The nowayout flag status (0 = not set, 1 = set)
 * \param status    The status flag (0 = watchdog disabled, 1 = watchdog active)
 * \return 0 on success, -1 otherwise
 * */
int xsc_watchdog_get_status(int *nowayout, int *status);
```

This is the interface that we recommand to use.

17.4 Hibernate

/dev/rtc0 is the hardware Real Time Clock (RTC) which interfaces to the Q-card via I2C.

You can use standard tools to test the hardware RTC:

```
# synchronize the HARDWARE clock with system clock
/sbin/hwclock -w -f /dev/rtc0
```

```
# shut down system and wake up from hibernate in 10 seconds
/usr/sbin/rtcwake -u -d /dev/rtc0 -s10 --mode off
```

17.5 Fallback Mechanism

The integrity of the images is verified during the boot process via CRC and md5sum.

This ensures the detection of firmware image data corruption and makes it possible to boot on a safe firmware image.

17.6 Robust Firmware Upgrade Mechanism

The Q-card provides an application that can upgrade partial or complete firmware images.

17.6.1 LibXiphos update_xdi

The update_xdi tool is able to update a Q-card flash copy, using an XDI image as the source.

The tool has the following interface:

```
$ update_xdi -h

Usage: update_xdi [OPTIONS] <type> <chip> <copy> <xdifile>
Update a NOR copy with contents from the xdi file
<type>: all, rootfs, bootloader, kernel, logic, devicetree

Options:
  -h, --help:        display this message and exit
  -v, --version:     display the version and exit
```

The tool performs the following actions:

1. First the tool verifies that the needed components can be found in the specified xdi file.
2. Second, the tool makes sure that the flash copy is unlocked properly, as needed before the operation.
3. Third, the tool prepares the target device: for partitions it erases them, and for files, it attaches (UBI) and mounts the target rootfs partition.
4. Fourth, the tool writes the file to the device.
5. Lastly, the tool unmounts the rootfs partition, or remounts it read-only, detaches the UBI from the MTD if possible, and write-protects the NOR chip: computing the checksum of the rootfs, and then locking the flash chip.

17.7 SSH Server

The Q-card comes with `dropbear` as SSH server.

For security, the server needs to use an SSH key that is generated at boot time. As the rootfs is Read-Only, a new key is generated at each boot.

If a persistent key is needed (to avoid error messages on the SSH client after each reboot of the Q-card), it can be added in `/etc/dropbear/` and must be named `dropbear_rsa_host_key`.

For example, to use the temporary generated key as a persistent key on the nominal copy of the first chip, you need to do this:

```
writeprotect 0 0 0
cp /var/lib/dropbear/dropbear_rsa_host_key /etc/dropbear/
writeprotect 0 0 1
```

This needs to be executed after each XDI image upgrade.

17.8 Time management

The Q-card comes with `ntpd` as a time synchronization daemon.

When the network is available, NTP synchronizes the system clock and the hardware clock with the remote time server.

The configuration file is `/etc/ntp.conf` is set for ground use and calibration and is set to synchronize the clock to NRC (National Research Council Canada) time (server time.nrc.ca).

17.8.1 Systemd services

The systemd services used to manage the time are hwclock.service (Save/Restore time from Real Time Clock) and ntpd.service (Network Time Service).

At power on, the system time is synchronized with the hardware clock (RTC). Then the NTP daemon is started and attempts to reach the network time server. In case of success, the system clock and hardware clock are synchronized with this server.

When the device is shutdown, the hardware clock is synchronized with the system time.

Note that the RTC must be powered to keep track of time while the Q-card is off (see Real Time Clock).

17.9 Triple-Mode Redundancy (Space)

Q-card Space ProASIC3 logic is synthesized with TMR to protect it against SEU.

17.10 RAM ECC (Space)

The Q-card Space has ECC protection on RAM which mitigates temporary and permanent bit flips due to radiation effects.

17.11 Memory Scrubbing (Space)

In addition to ECC RAM, the Q-card performs memory scrubbing to avoid single bit flips becoming dual-bit errors that are uncorrectable.

The error correction in itself is performed seamlessly in hardware by the memory controller. The scrubbing (or scan) is performed in software and these parameters are configurable:

- Scrub Priority: priority of the process in Linux
- Scrub Frequency: time between each scan
- Scrub Duration: duration of a memory scan

These parameters are controlled via systemd's timer and service:

- `/lib/systemd/system/mempatrol.service`
- `/lib/systemd/system/mempatrol.timer`

If an uncorrectable error is found in memory, a kernel panic can be triggered. This is not enabled by default.

17.12 Safe Mode (Space / Mission Specific)

The Q-card Space can halt its boot process until it receives commands through its backdoor interface. This Safe Mode feature is required to prevent boot loop or boot crashes on remote environments.

This is a factory configurable mode into which the Q-card Space goes once power is applied or the PA3 is reset. In safe mode:

- The Zynq is kept in reset
- Backdoor access is possible
- A command to boot can be received by the ProASIC3

17.13 Remote Diagnostic Port (Space)

The Q-Card Space has a remote diagnostic port (RDP) in the ProASIC3 that can be used to perform operations if the boot process fails.

The port allows the user to:

- Read and Write to QSPI Flash
- Manually configure the Zynq
- Monitor temperature and power consumption
- Access scratch and circular buffer

The remote diagnostic port is usually connected to the satellite main OBC. It can also be connected directly to a radio.

Although the RDP is not required for proper Q-card operation, it is highly recommended to implement one in the satellite system: it provides a backdoor to access the Q-card even when everything else is not working.

> Note: The remote diagnostic port access to NOR FLASH and telemetry is not functional if the Q-Card Xilinx FPGA is configured.

17.13.1 Default Communication Interface

The base version of the RDP uses a UART interface with the following characteristics:

- Baud rate: 115200 bps
- Data bits: 8
- Parity: No
- Stop bits: 1
- Duplex: Full

The RDP can also be configured for half-duplex operation. If this option is required, please contact XSC.

A custom packet protocol is implemented on top of the UART. To abstract the communication layer, XSC provides a C library that can be used by customers to add the functionality to their application running on some computer other than the Q-Card, for example on the satellite OBC.

The RDP can be independent or shared with another UART connected to the Xilinx FPGA. The Q-Card default configuration shares the RDP with the Zynq US+ PS0 UART and the RSXXXX port.

By default, the RDP is in bypass mode and does not interfere with regular communications untils it detects a 250ms or more “break” signal (standard signal for UART). Once the signal is detected, it will start to listen on the port and interpret incoming data. Once the communication with the RDP is not required anymore it can be returned to the bypass mode.

If the “break” signal cannot be implemented by the customer application, please contact XSC for alternative methods.

17.13.2 Software Tools

The following sections assumes that the default communication interface is used.

17.13.3 Library

XSC provides a C library that implements the serial communication protocol.

17.13.3.1 Usage Example

Read, Erase and Write 4096 bytes in the Flash and get the corresponding crc at each step.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdbool.h>
#include <unistd.h> // For usleep

#include "pa3sercomm.h"   // Main libpa3 commands
#include "pa3q7cmd.h"     // zynq reset
#include "pa3qspiflash.h" // Flash related commands

void crc_poll_sleep() { usleep(100); }

int main(){

    const uint8_t FLASH_CHIP = 0;
    const uint32_t OFFSET = 0;
    const uint32_t SIZE = 0x1000; // 4096 bytes

    uint8_t buffer[SIZE];
    uint16_t read_crc, erase_crc, write_crc;

    /* Initialize serial communication */
    struct pa3sercomm_s *pa3sc = pa3sc_construct();
    pa3sc_init_options(pa3sc);
    pa3sc_set_baud(pa3sc, 115200);
    pa3sc_set_port(pa3sc, "/dev/ttyUSB0");
    pa3sc_init(pa3sc);
    /* Obtain flash structure */
    qspi_flash_t *qspi_flash = qspi_flash_construct(pa3sc);

    /* Bypass of the serial port to the pa3  */
    pa3sc_bypass(pa3sc, true);

    /* Enable the pa3 writes */
    pa3sc_write_access(pa3sc, true);

    /* Put Zynq in reset (Must be done to avoid bus contention) */
    q7cmd_zynq_reset(pa3sc, true);

    /* Unlock Flash */
    qspi_flash_set_lock(qspi_flash, FLASH_CHIP, false);

    /* Reset flash (Required to set in 3-bit addressing) */
    qspi_flash_reset(qspi_flash, FLASH_CHIP);

    /* Read to buffer and get CRC */
    qspi_flash_read(qspi_flash, FLASH_CHIP, OFFSET, SIZE, buffer, NULL);
    qspi_flash_get_crc(qspi_flash, FLASH_CHIP, OFFSET, SIZE, &read_crc,
                       crc_poll_sleep);
    printf("CRC␣at␣read␣0x%04X\n", read_crc);

    /* Erase and get CRC*/
    qspi_flash_erase(qspi_flash, FLASH_CHIP, OFFSET, SIZE, NULL);
    qspi_flash_get_crc(qspi_flash, FLASH_CHIP, OFFSET, SIZE, &erase_crc,
                       crc_poll_sleep);
    printf("CRC␣after␣erase␣0x%04X\n", erase_crc);

    /* Write from buffer and get crc */
    qspi_flash_write(qspi_flash, FLASH_CHIP, OFFSET, SIZE, buffer, NULL);
    qspi_flash_get_crc(qspi_flash, FLASH_CHIP, OFFSET, SIZE, &write_crc,
                       crc_poll_sleep);
    printf("CRC␣after␣write␣0x%04X\n", write_crc);

    /* Lock Flash */
    qspi_flash_set_lock(qspi_flash, FLASH_CHIP, true);

    /* Release control */
    q7cmd_zynq_reset(pa3sc, false);
    pa3sc_write_access(pa3sc, false);
    pa3sc_bypass(pa3sc, false);

    pa3sc_shutdown_options(pa3sc);
    free(buffer);
    return 0;
}
```

17.13.4 pa3tool

The pa3tool software can be used to communicate with the ProASIC3 FPGA from a PC on the ground for test and debug purposes.

This tool is present in the SDK. Refer to the article Install Yocto SDK on how to install and load the SDK.

The pa3tool `--help` option displays the list of available commands.

> Note: The serial communication port cannot be used simultaneously for the console and remote diagnostic. Please close all connections to this port before running pa3tool.

17.13.4.1 Enable/Disable the Remote Diagnostic mode

These commands can be used to switch from the console mode to the remote diagnostic mode (and vice versa):

```
pa3tool --port ${TTY} --switch-to-pa3
pa3tool --port ${TTY} --switch-to-console
```

17.13.4.2 Enable/Disable control FPGA Registers Write Access

These commands can be used to enable or disable Write access to the control FPGA registers:

```
pa3tool --port ${TTY} --enable-writes
pa3tool --port ${TTY} --disable-writes
```

17.13.4.3 Test Remote Diagnostic Mode

This command can be used to verify that the tool can read and write the control FPGA's registers through the Remote Diagnostic Port.

```
pa3tool --port ${TTY} --switch-to-pa3
pa3tool --port ${TTY} --test
```

17.13.4.4 Usage example

Read and Write 4096 bytes in the Flash

```
# Set the Zynq in Reset before any flash operation
pa3tool --port ${TTY} --q7-proasic-reset

# Backup 4096 bytes from the flash
pa3tool --port ${TTY} --flash-unlock 0
```

```
pa3tool --port ${TTY} --flash-read-to-file 0 0 0x1000 /tmp/qspi0.fw

# Request CRC for the 4096 bytes
pa3tool --port ${TTY} --flash-crc-size 0 0 16

# Erase the flash before any write operation
pa3tool --port ${TTY} --flash-erase-size 0 0 0x1000

# After the erase command, the CRC should be equal to 0xfe1
pa3tool --port ${TTY} --flash-crc-size 0 0 16

# Restore the flash content
pa3tool --port ${TTY} --flash-write-from-file 0 0 /tmp/qspi0.fw
```

17.13.5 UART Buffer (Space)

This feature is used for remote debugging of crashes/hangs.

The Q-card Space provides a UART buffer with the following characteristics:

- Backdoor accessible (read and write)
- Zynq accessible (read and write)
- 8 kB
- Retains information as long as power is applied (no reset possible)
- Circular buffer (pointer to last value written)
- User selectable input source

The circular buffer implemented in the ProASIC3 provides a buffer of 8192 bytes that records, by default, the console interface output.

Here are the instructions to read the circular buffer logs:

```
# activate the remote diagnostic port
pa3tool --port ${TTY} --switch-to-pa3

# select the circular buffer to read and validate value
pa3tool --port ${TTY} --circ-buf-mux-set 0
pa3tool --port ${TTY} --circ-buf-mux-get

# read the circular head pointer (this command may take a long time)
pa3tool --port ${TTY} --circ-buf

# reactivate the console
pa3tool --port ${TTY} --switch-to-console
```

17.13.6 Scratch Buffer (Space)

This feature is used to store values that must survive a reboot.

The Q-card Space provides a scratch buffer with the following characteristics:

- Backdoor accessible (read and write)
- Zynq accessible (read and write)
- 1 kB or more
- Retains information as long as power is applied (no reset possible)

17.14 SEM IP (Space) / FPGA Bitstream Scrubbing

The Q-card Space uses the Xilinx SEM IP core to scrub the FPGA bitstream. In addition, Linux contains a driver to control the scrubber and report errors.

Note: BCH codes can be used to correct radiation induced bit flips in QSPI flash.

17.14.1 SEM Monitor

The SEM IP device provides an interface to report detected errors and if they were correctable or not.

A SEM monitor service is used to report those errors and take actions if needed. It is run by SystemD at boot time and reports errors in the SystemD journal.

The reported error is a line like this one:

```
TYPE=SED_OK PA=0x00001916 LA=0x000006e0 COR=[(0x00,0x00)]
```

That indicates the type of error, the physical and linear addresses where it occured and the list of bits that have been corrected. If the list is empty, it means the error could not be corrected.

Each element of the list is in the form `(WA,BA)` where: * WA: Word Address. (8 bits) * BA: Bit Address. (8 bits)

for the given address.

The type of the error can be any of: * `ECC` * `CRC` * `BFR` * `SED_OK` * `SED_NG` * `DED`

Please refer to [0] and [1] for more information on error types and bit addresses.

17.14.2 NNG Reporting

Detected error are reported through an NNG publisher that is listening for new subscribers at the address given to the `-P` argument (by default on port `4242`).

It allows subscribers to connect to the monitor via TCP port 4242 to receive error messages.

The messages will have the following structure:

```
SEM_IP;<CORRECTION_TYPE>
```

where `CORRECTION_TYPE` is either: * `CORRECTED`: The error was corrected * `UNCORRECTABLE`: The error could not be corrected

17.14.3 Error Injection

A tool is provided to inject errors in the SEM IP device. That is useful to check if errors are correctly reported.

Any number of errors can be injected with the `xsc_sem_ctrl` tool. It takes any number of error addresses in argument and will send them to the monitor to inject them.

Inject a 2 bit error:

```
# Check that the service is running
systemctl is-active sem_monitor
active

# Inject errors
xsc_sem_ctrl injection -i C0006E0000 -I "tcp://localhost:4243"
```

Using `journalctl`, 1 error should have been logged with 1 bit correction:

```
journalctl -u sem_monitor
```

The `TS` corresponds to a timestamp at which the error was detected. The `COR` is the correction list.

17.14.4 References

[0] : https://www.xilinx.com/support/documentation/ip_documentation/sem/v4_1/pg036_sem.pdf

[1] : https://www.xilinx.com/support/documentation/ip_documentation/sem_ultra/v3_1/pg187-ultrascale-sem.pdf

18 Q7 Components

18.1 Control FPGA

The Microsemi ProASIC3 FPGA is robust against radiation effects and is used to implement critical features.

18.2 Processing FPGA

Xilinx Zynq 7020 FPGA is used to implement the processing features.

18.3 Processing FPGA CPU RAM

256MiB of LPDDR2 ECC RAM available to the processing FPGA CPU. Or 512MiB of LPDDR2 RAM, no ECC.

18.4 Non-volatile Firmware Storage

2x 128MB non-volatile Mass storage chips that provides redundant firmware storage.

> Note: The Q7 will be used in harsh environment conditions where one firmware storage chip can be corrupted or permanently damaged.

The NOR flash supports at least 100,000 ERASE cycles per sector. Note that each write operation is preceeded with an erase sector.

18.5 Data Mass Storage

2x micro-SD cards are available to the processing unit for data storage.

These chips can be left unpowered to reduce power consumption and SEE risks, for more information see power control.

18.6 Real Time Clock (RTC)

The RTC allows the Q7 to keep track of time even when powered off. The RTC must be externally powered via the VCC_BAT input on the Q7-PIM or mezzanine bottom connector if the main power is off. Otherwise the time will be lost.

19 Q7 Electrical Interfaces

The electrical interfaces on the Q7 are described in the following table. The connector types and pin callouts are found in the Q7 Connectors section. Connectors are categorized as “Nominal” (included in standard product delivery) or “Special” (available but not recommended for spaceflight).

Name	Interface	Nominal or Special	Purpose or Notes
Q7-J1	Mezzanine	Nominal	Interface to daughterboards; > 90 IO
Q7-J2	PIM	Nominal	Interface to PIM; No ESD protection
Q7-J3	Input Power	Special	4-34 V unregulated power (1.3x4.2 mm barrel connector) input on ground
Q7-J4	Ethernet	Special	Gigabit Ethernet (RJ-45) for network or payload connection on ground
Q7-J5	USB	Nominal	USB 2.0 (480 mbps) (Micro-AB) for network or payload connection on ground
Q7-J6	Micro-SD	Nominal	SD Card for storage
Q7-J7	Micro-SD	Nominal	SD Card for storage

Note:
Power and Ethernet connectors are generally not used for space flight.
Ethernet signals can be routed from the Q7 to a custom daughterboard connector by installing 0.1” header in the Q7 RJ45 connector footprint and 0.1” socket on the daughterboard.

19.1 Power Input Details

The Q7 generates all of its required internal voltages from a single source.

Minimum voltage is dependent on the use of USB and the optional provision of 5 V on the mezzanine connector. If one or both are required, the minimum voltage is 6 V, otherwise the minimum voltage is 4 V.

By default, the maximum input voltage is 15 V. In this mode all internal voltages are directly generated from the input voltage which provides a very high efficiency. If an input voltage of up to 34 V is required, the 5 V regulator output can be routed (using a jumper) to feed the other regulators. Power consumption will be slightly higher because the Q7 will be then using two-stage regulation.

The Q7 is designed for transients of up to 42 V.

19.2 Power-on inrush current Details

The Q7 can have a high current spike at boot when the input voltage reaches 3.0V. Measured with an external isolated current sensor, the spike can be up to **1.03A** during **1.7ms** .
The inrush current rate, calculated from the moment power is applied to the card, represents a rate of 65.4A/s .

19.3 Power Consumption

Q7 power consumption is dependent on processor load, logic usage, clock frequencies, and use of interfaces; it can range from 1 W to 5 W. Typical power consumption is < 2 W.

As a "real world" example, we recently completed a "hybrid" implementation of a vision processing algorithm (ICP) for a customer, and compared the performance running on the Q7 versus a PC. The power consumption during those tests is summarized in the table below. For context, the mean processing time on the Q7 with logic acceleration matched the PC (core i870 @ 2.93 GHz).

State	Power Consumption	Increase over Idle
Idle	1.88 W	
Load Average	1.99 W	0.11 W (6%)
Load Peak	2.41 W	0.53 W (28%)

19.4 ESD Protection

USB, Ethernet and RSXXX interfaces have 8kV (HBM) or better ESD protection.

19.5 USB host Current Limit

By default the Q7 limits the USB port current to 500 mA. The value can be modified by changing a 0402 resistor (R56) which sets the USB power switch current limit.

The following table presents some commonly used falues. For additional values please contact Xiphos.

Resistor	Current Limit (A)
49.9k	0.5
30.0k	0.85
20.0k	1.25

This image shows the resistor position on the Q7:

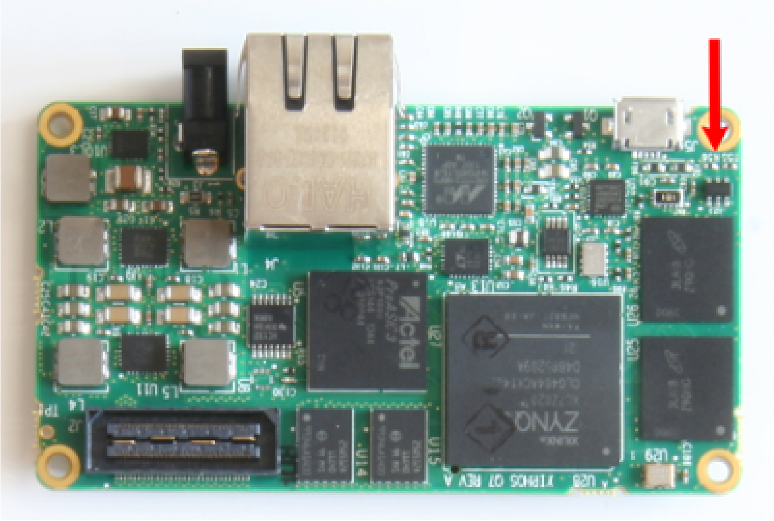


Figure 4: USB resistors

19.6 Mezzanine Connector

The Q7 mezzanine connector is used to interface with daughterboards. It provides:

Number of pins	Description	Connection
50	Flexible voltage (1.8, 2.5, 3.3 V) I/O (up to 24 LVDS pairs)	Zynq-PL
36	3.3V GPIO	ProASIC3, Zynq-PL (34 pins only)
5	1.8V GPIO	Zynq-PL
12	1.8V MIO	Zynq-PS
3	USB port (DP, DM, VCC)	Zynq-PS
4	RSXXX port (1 RS422 or 1 RS485 or 2 RS232)	ProASIC3, Zynq-PS
2	Power control pins (on/off)	Power circuit
3	Power input	Power circuit
1	1.8 V Power output (available to daughterboard)	
1	3.3 V Power output (available to daughterboard)	
1	5.0 V Power output (available to daughterboard)	

Number of pins	Description	Connection
1	Q7 RTC auxiliary input (to keep track of time and when Q7 is unpowered with an external battery)	
1	VCC flex	Zynq-PL Flexible voltage bank

20 Q7 Mechanical Interfaces

20.1 Volume

The volume of the Q7 depends on its configuration. The table below summarizes typical delivery configurations:

Configuration	Length (mm)	Width (mm)	Height (mm)	Notes
Q7 with Ethernet, Power, USB, PIM	78	43	22	Recommended development configuration, with PIM
Q7 with Ethernet, Power, USB	78	43	19	Recommended development configuration, without PIM
Q7 without Ethernet	78	43	11.5	Recommended flight configuration
Q7 without Ethernet and Power	78	43	8.5	Recommended flight configuration

20.2 Interface Bolt Pattern and Contact Surfaces

The interface bolt pattern is shown in the mechanical drawing below. The four mounting holes in the corners accept M2 or 2-56 screws. A STEP 3D model can be provided upon request. Xiphos can also provide a Cadence PCB editor footprint of the Q7 to help develop daughterboards.

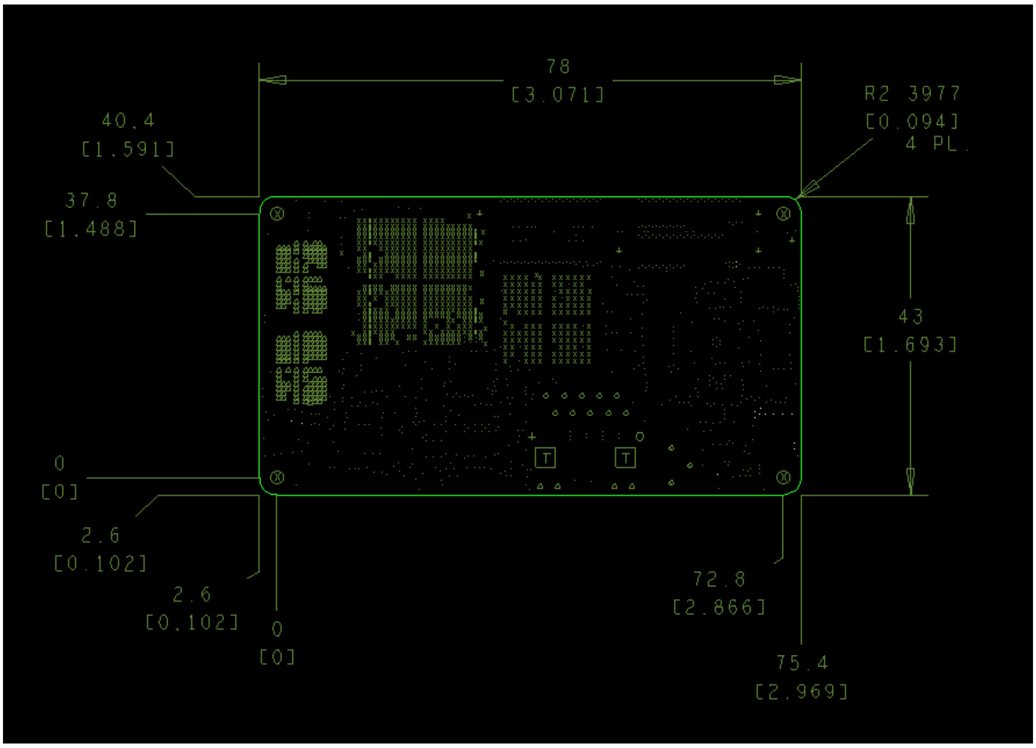


Figure 5: Mechanical drawing

20.3 Mass

The Q7 mass depends on its configuration. The table below summarizes component mass:

Component	Card Mass (g)	Notes
Q7	24 (+/-2)	USB and Power connector are included in Q7 mass.
Q7 + Ethernet	32 (+/-2)	
PIM	12 (+/-2)	

20.4 Thermal Interfaces

20.4.1 Heat Rejection Paths

The estimated power consumption of the Q7 is dependent on processor load, and can range from 1 W to 5 W. Below 2 W, the heat can be removed entirely through a conductive path of the mounting interface to the host system. For additional caution, in cases where high processor loads may be sustained, a cold finger can be fixed directly from the Xilinx Zynq to the host system.

20.5 Q7 Serial Interface

The Q7 has a multi-protocol RS232/RS422/RS485 transceiver that can be configured in several different manners. When the PIM is connected, an FTDI USB-Serial converter is also available through the micro USB.

20.5.1 RS232

When connecting in RS-232 format, the following diagrams describe the possible connections.

20.5.1.1 Q7/PIM RS232

The Q7/PIM provides RS232 levels through the multipurpose connector

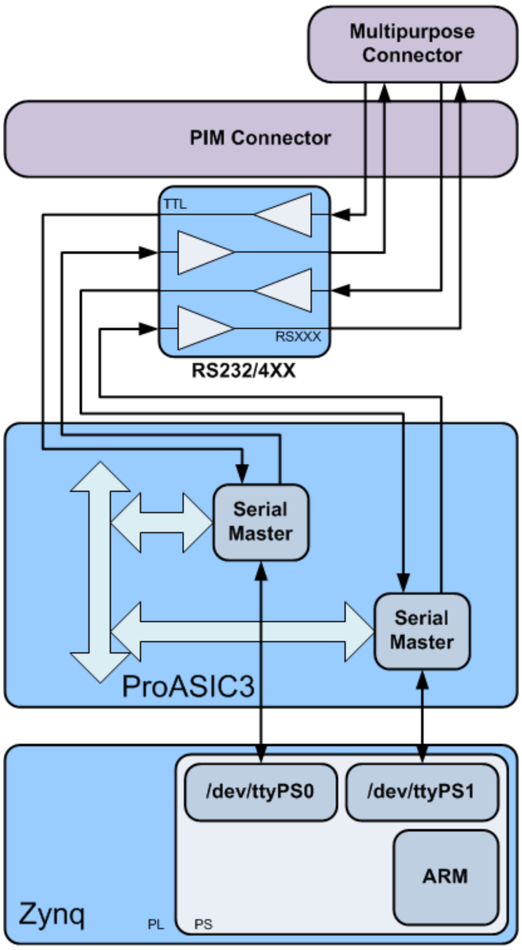


Figure 6: Serial Port RS232

20.5.1.2 Q7/Q-STRIP Mezzanine

The Q7 provides RS232 level connections through the Q-STRIP Mezzanine connector.

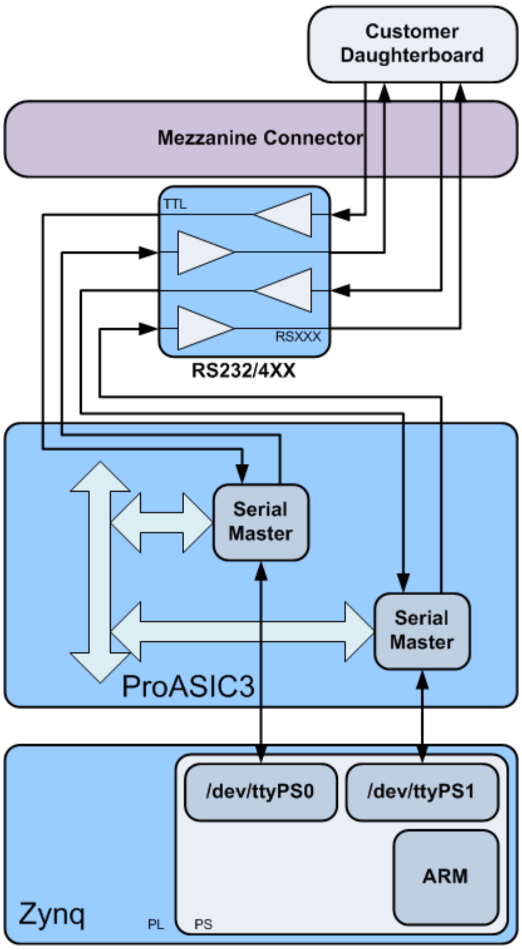


Figure 7: Serial Port QSTRIP RS232

20.5.2 USB-Serial

There are two options for connection to the Q7 as a serial slave. This requires an external USB host such as a workstation or laptop.

20.5.2.1 Q7/PIM FTDI

The Q7/PIM has an FTDI serial port.

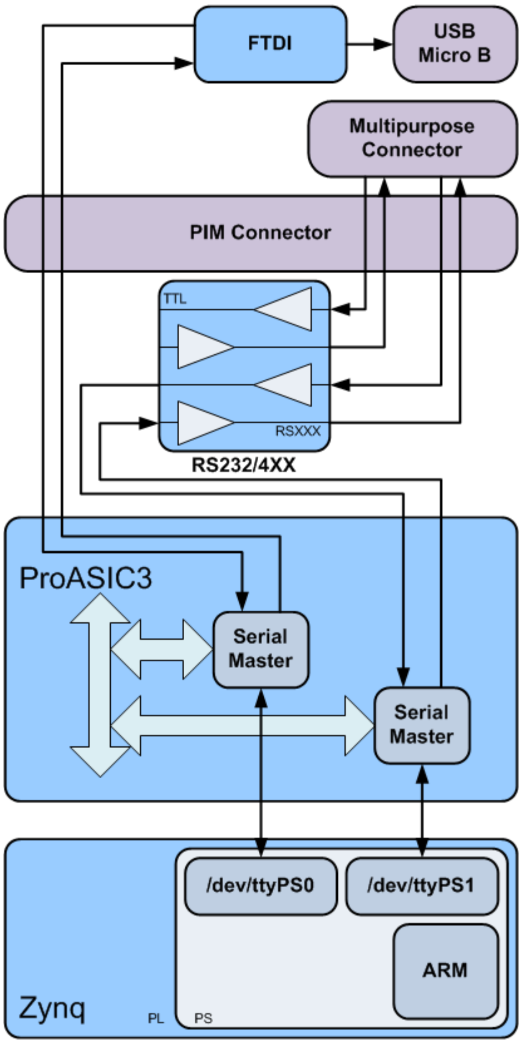


Figure 8: Serial Port PIM FTDI

20.5.2.2 Q7 OTG Serial Gadget

The Q7 USB device is capable of acting as a serial USB gadget. This feature is available upon request.

20.6 Q7 MicroSD

The Q7 has two MicroSD slots connected through programmable logic to the Zynq SOC SDHCI master.

20.6.1 Q7 MicroSD Power Control

The power supply for the Q7 MicroSD cards are monitored and controlled by the ProASIC3.

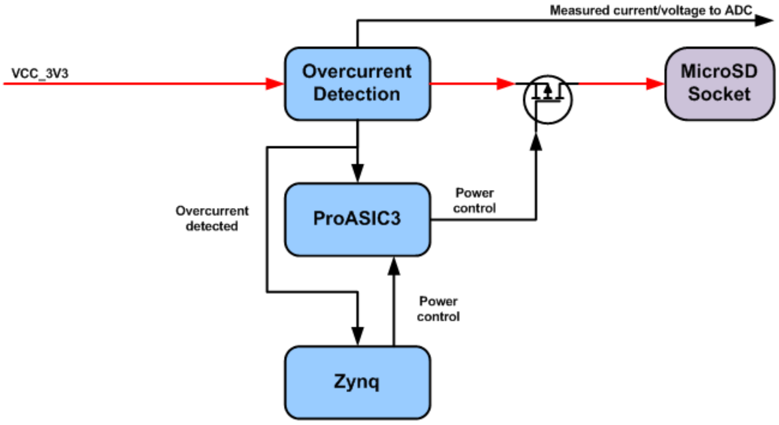


Figure 9: SD Power Control Overview

There are several internal components that can be used to exercise this control and monitoring:

- ProASIC3 GPIO Bank for SD Control
- Q7 FPGA GPIO Bank for SD Monitoring
- Q7 ADC for SD Current Monitoring
- Zynq SDHCI Registers for re-initializing a MicroSD card that has been powered on.

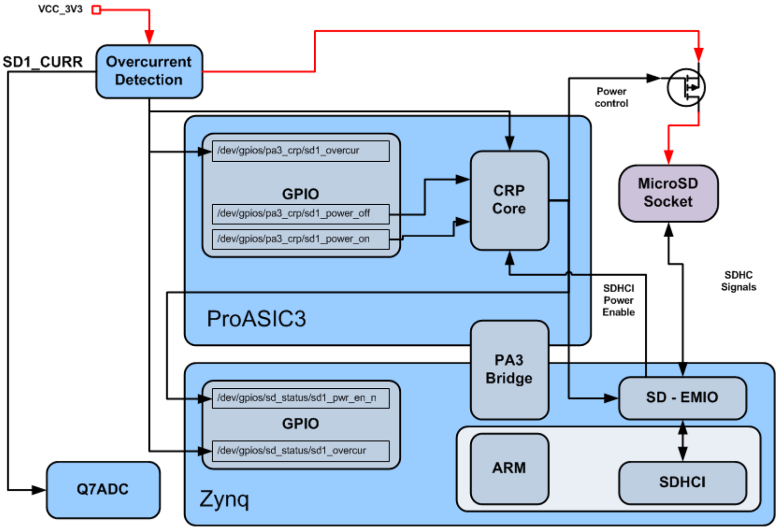

VCC_3V3
SD1_CURR
Overcurrent Detection
Power control
/dev/gpios/pa3_crp/sd1_overcur
GPIO
/dev/gpios/pa3_crp/sd1_power_off
/dev/gpios/pa3_crp/sd1_power_on
CRP Core
MicroSD Socket
SDHCI Power Enable
SDHC Signals
ProASIC3
PA3 Bridge
/dev/gpios/sd_status/sd1_pwr_en_n
GPIO
/dev/gpios/sd_status/sd1_overcur
SD - EMIO
ARM
SDHCI
Q7ADC
Zynq

Figure 10: SD Power control

20.6.2 Q7 MicroSD Power Control - Zynq/PA3

For the Q7 system developer, there are two banks of GPIO lines that are useful for controlling and monitoring the power supply to the MicroSD.

20.7 Q7 Connectors

20.7.1 Q7 Mezzanine Connector J1

Part number: Samtec QSH-060-01-C-D-A

Mating connector: Samtec QTH-060-0X-C-D-A (X select mating height 5mm to 25 mm available)

Pinout: Pinout of the Q7 mezzanine connector is provided in RD04 (worksheet "Bottom mezzanine"):

20.7.2 Q7 PIM Connector J2

Part number: Samtec QSH-030-01-C-D-A

Mating connector: Samtec QTH-030-0X-C-D-A (X select mating height 5mm to 25 mm available)

Pinout: Pinout of the Q7 PIM connector is provided in RD04 (worksheet "Top mezzanine"):

20.7.3 Q7 Power Connector J3

Part number: CUI PJ-031D

Mating connector: Any standard 1.35mm ID, 3.5mm OD barrel connector plug

Pinout: Outside: GND_D Inside: VCC_IN

20.7.4 Q7 RJ45 Ethernet Connector J4

Part number: Halo HFJ11-E1G16E-L12RL

Mating connector: Any standard RJ45 plug

Pinout: Standard Gigabit Ethernet

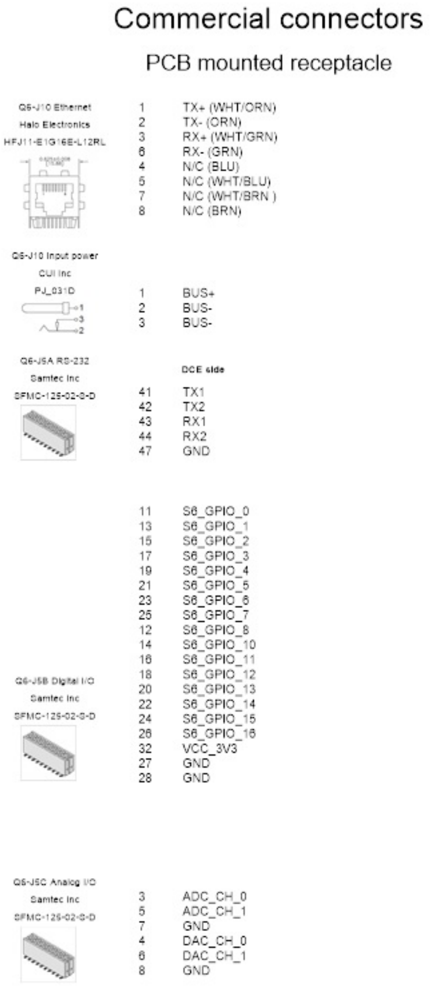
20.7.5 Q7 USB Connector J5

Part number: FCI 10104111-0001LF (Micro-AB) OR 10104110-0001LF (Micro-B)

Mating cable: Any standard USB Micro-A/Micro-B cable

Pinout: Standard USB 2.0 Micro-AB/Micro-B pinout

20.7.6 Q7 interface connector specifications



Commercial connectors

PCB mounted receptacle

Q6-J10 Ethernet Halo Electronics HFJ11-E1G16E-L12RL	1	TX+ (WHT/ORN)
	2	TX- (ORN)
	3	RX+ (WHT/GRN)
	6	RX- (GRN)
	4	N/C (BLU)
	5	N/C (WHT/BLU)
	7	N/C (WHT/BRN)
	8	N/C (BRN)
Q6-J10 Input power CUI Inc PJ_031D	1	BUS+
	2	BUS-
	3	BUS-
Q6-J5A RS-232 Samtec Inc SFMC-125-02-S-D		DCE side
	41	TX1
	42	TX2
	43	RX1
	44	RX2
	47	GND
Q6-J5B Digital I/O Samtec Inc SFMC-125-02-S-D	11	S6_GPIO_0
	13	S6_GPIO_1
	15	S6_GPIO_2
	17	S6_GPIO_3
	19	S6_GPIO_4
	21	S6_GPIO_5
	23	S6_GPIO_6
	25	S6_GPIO_7
	12	S6_GPIO_8
	14	S6_GPIO_10
	16	S6_GPIO_11
	18	S6_GPIO_12
	20	S6_GPIO_13
	22	S6_GPIO_14
	24	S6_GPIO_15
	26	S6_GPIO_16
	32	VCC_3V3
	27	GND
	28	GND
Q6-J5C Analog I/O Samtec Inc SFMC-125-02-S-D	3	ADC_CH_0
	5	ADC_CH_1
	7	GND
	4	DAC_CH_0
	6	DAC_CH_1
	8	GND

Figure 11: Commercial connectors

21 Hardware FAQ

21.1 U3 PIN 5

Question: U3 pin 5 on the Q7-PIM is snipped. Is it normal?

Answer: U3 pin 5 is snipped to fix a small design error on the PIM board. It is not an issue.

21.2 Oxidation

Question: Is the oxidation on Q7 inductors normal?

Answer: Oxidation on the inductors is caused by the water wash process used to clean the board after assembly. "Water wash" is used for the Q7 instead of "no clean" to leave no residues on the board. The oxidation is only an aesthetic issue. From the manufacturer's documentation:

> In the case that oxidation does occur, the effects are contained to the surface of the component and do not penetrate into the core material. No electrical effects have ever been documented due to oxidation of the IHLP product. Oxidation should never be considered a reliability risk.

Please refer to the manufacturer document:

IHLP_customer_guide.pdf RD05

21.3 QFN package pins

Question: Wetting on QFN packages side pins is not optimal. It is a problem?

Answer: QFN package pins are under the package with only a small portion showing on the side therefore, as long as the soldering is good under the chip, the connection will be fine. From the chip manufacturer's documentation:

> NON-CRITICAL AREA: SURFACE C (TERMINAL SIDES), NOT INTENDED TO BE A WETTABLE SURFACE BY DESIGN

Please refer to the manufacturer document:

Linear_Technology_QFN_DFN_Solder_Joint_Package_Inspection.pdf RD06

22 Q7-PIM

22.1 Overview

The Q7 companion board called the Product Interface Module (“PIM”) is normally included in all sales to new customers.

It provides additional features that are not required for space missions but that are helpful for development, debugging, and terrestrial applications.

22.2 Real Time clock Battery

The PIM battery is used to keep the Q7 real-time clock powered while power is not applied to the Q7.

It is a rechargeable battery that can last multiple months and is recharged automatically when the Q7 is powered.

> Notes:
> If a PIM connected to a Q7 is not powered for very long period of time the battery voltage may become too loo for the RTC. Therefore the Q7 will lose track of time.
> If the PIM is disconnected from a non-powered Q7 even for a brief period of time, the Q7 RTC will lose track of time.

22.3 Status LED

The PIM includes 6 status LEDS From left to right:

Color	Function	Notes
Yellow	Serial console RX/TX activity	Blinks when data is sent or received on the USB serial console
Blue	Power Indicator	Indicates that Q7 3.3V power regulator is running
Green	User defined	Can be used by developer
Green	User defined	Can be used by developer
Orange	User defined	Can be used by developer

Color	Function	Notes
Orange	User defined	Can be used by developer

Notes:
The blue LED stays off if power is applied to the Q7 but the Q7 is off (power off command from Linux or off switch used)

22.4 USB Serial Console

The Q7 debug console is routed through the PIM to a serial-to-USB converter. This permits connecting the PIM to any PC with a USB port.

Standard serial terminal software like putty for Windows or picocom for Linux can be used to access the Q7 console.

By default the bootloader and Linux console are available on the debug console.

The default ports settings are:

- Baud rate: 115200
- Bits: 8
- Parity: No
- Stop bit: 1

22.4.1 Zynq JTAG header

The Zynq JTAG header is used to connect the standard Xilinx JTAG programming cable to the Q7.

The Xilinx JTAG programming cable is useful for debugging and can be used to update the Q7 firmware.

22.4.2 ProASIC3 JTAG header

The ProASIC3 JTAG header is used to connect standard Microsemi FlashPro 4 cable to the Q7.

The FlashPro 4 is used to update the ProSIC3 logic. This operation is usually performed only by Xiphos.

22.4.3 Power Switches

Two push button switches can be used to quickly power on or of the Q7:

- SW1 can be used to power off the Q7
- SW2 can be used to power on the Q7

22.4.4 Analog Inputs

The PIM includes an Analog devices AD7689 ADC with the following characteristics:

- Resolution: 16 bits
- Speed: 250 ksps
- Number of inputs: 4 (on the multipurpose connector)
- Input type: single ended, referenced to analog ground
- Input range: 0 to 4.096V
- Input filter: RC with a cut-off frequency of 1600 Hz

The 4 channels can be used to read analog signals connected to the multipurpose connector.

The cut-off frequency can be modified by changing resistor values on the PIM. Please contact Xiphos for more information.

22.4.5 CAN

The PIM include a CAN 2.0B transceiver connected to one of the two Zynq CAN controllers.

The CAN port is available on the multipurpose connector. The PIM also includes a software selectable 120 Ohm termination (off by default).

Both the CAN controller and the termination can be controlled through Linux.

22.4.6 1-Wire

The PIM includes a 3.3 V 1-Wire master transceiver connected to the Zynq. It can be accessed through Linux.

> Notes:
> The 3.3V present on the multipurpose connector can be used to power the 1-Wire bus
> The 1-Wire port requires a dedicated 3.3V line (no power over data line)

22.4.7 3.3V GPIO

The PIM multipurpose connector provides access to 12 Q7 GPIO (Q7_GPIO_3V3_0 to Q7_GPIO_3V3_11).

Please note that these GPIO are shared with the Q7 mezzanine (bottom) connector therefore they cannot be used if a daughterboard connected under the Q7 also uses them.

22.5 Multipurpose Connector Pinout

PIM connector: Samtec TFM-120-01-L-D-RE1-WT

Mating cables:

- Teflon: Samtec SFSDT-20-28-G-24.00-DR-NUS or equivalent
- PVC: Samtec SFSD-20-28C-F-24.00-DR-NUS or equivalent

The mating cables can be purchased directly from Samtec

Pin	Pin Name	Notes	Pin	Pin Name	Notes
1	PIM_ADC_IN_2	ADC7689 channel 4	2	PIM_ADC_IN_3	ADC7689 channel 6
3	PIM_ADC_IN_0	ADC7689 channel 0	4	PIM_ADC_IN_1	ADC7689 channel 2
5	GND_A	Analog ground	6	GND_A	Analog ground
7	GND_D	Digital ground	8	GND_D	Digital ground
9	Q7_GPIO_3V3_10	Q7 3.3V GPIO	10	Q7_GPIO_3V3_11	Q7 3.3V GPIO
11	Q7_GPIO_3V3_08	Q7 3.3V GPIO	12	Q7_GPIO_3V3_09	Q7 3.3V GPIO
13	Q7_GPIO_3V3_06	Q7 3.3V GPIO	14	Q7_GPIO_3V3_07	Q7 3.3V GPIO
15	GND_D	Digital ground	16	GND_D	Digital ground
17	Q7_GPIO_3V3_04	Q7 3.3V GPIO	18	Q7_GPIO_3V3_05	Q7 3.3V GPIO
19	Q7_GPIO_3V3_02	Q7 3.3V GPIO	20	Q7_GPIO_3V3_03	Q7 3.3V GPIO
21	Q7_GPIO_3V3_00	Q7 3.3V GPIO	22	Q7_GPIO_3V3_01	Q7 3.3V GPIO
23	GND_D	Digital ground	24	GND_D	Digital ground
25	Q7_RSXXX_TX_N	Q7 RSXXX Port	26	Q7_RSXXX_RX_N	Q7 RSXXX Port
27	Q7_RSXXX_TX_P	Q7 RSXXX Port	28	Q7_RSXXX_RX_P	Q7 RSXXX Port
29	GND_D	Digital ground	30	GND_D	Digital ground
31	PIM_CAN_H	PIM CAN port	32	PIM_CAN_L	PIM CAN port
33	GND_D	Digital ground	34	PIM_ONE_WIRE	PIM 1-Wire port
35	VCC_5V0	5 V generated by the Q7	36	VCC_3V3	3.3 V generated by the Q7
37	PWR_ON_EXT_N	Q7 power on control	38	PWR_OFF_EXT_N	Q7 power off control
39	VCC_IN	Q76 power input	40	GND_D	Digital ground

When looking at the PIM front side:

- Pin 1 is bottom left
- Pin 2 is top left
- Pin 39 is bottom right
- Pin 40 is top right

The following pins are shared with the Q7 mezzanine connector therefore they cannot be used if they are used by a daughterboard:

- Q7_GPIO_3V3_XX
- PWR_ON_EXT_N
- PWR_OFF_EXT_N
- Q7_RSXX_XX_X
- VCC_IN
- VCC_5V0
- VCC_3V3

22.5.1 Power input:

The VCC_IN pin can be used to power the Q7 from the multipurpose connector.

It is electrically connected to the Q7 barrel power connector therefore the power input requirements are the same as for the Q7.

22.5.2 Power output:

Up to 100 mA is available on VCC_5V0 and VCC_3V3 to power external devices through the multipurpose connector.

23 Q7-CameraBoard Information

The Q7 Camera Board is a daughterboard for Xiphos' Q7 hybrid processor card.

The Q7 Camera Board allows the Q7 to be inserted into existing systems with high-bandwidth video or imagery streams.

High performance image processing algorithms can then be executed on the Q7 in an optimal combination of tightly-integrated CPU and programmable logic resources.

The Q7 Camera Board is designed to be versatile:

- Supports several different camera interfaces

- Can be used for development or flight applications, with only minor modifications
- Designed in a small form factor to fit most spacecraft and payload volume constraints.

23.1 Features

23.1.1 Interfaces

- 2x Cameralink (2xBase, or 1xMedium, or 1xFull)
- 4x SpaceWire (min 200 Mpbs per LVDS pair)
- 1x HDMI ADV7611 Input / ADV7511 Output
- 4x USB2.0 Master Ports (480 Mbps)
- 1x 1-Wire (factory selectable 3.3V or 5V)
- 1x RS232/422/485 interface header

Interfaces cannot all be used concurrently – details of configuration options are available upon request.

23.1.2 Size and Mass information

Together with the Q7, the Q7 Camera Board extends the form factor to 110 mm x 110 mm x 15 mm, with a mass of 100 g.

23.1.3 Power

The Q7 Camera Board can provide userselectable voltage (3-28V) to camera, up to 1.5A per camera. Xiphos supports the PoCL standard or traditional camera power inputs.