Universal Configuration Manager

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.1
Date	17.09.2018

Copyright Notice

Disclaimer

The information available to you in this document/code may contain errors and is subject to periods of interruption. While NetTimeLogic GmbH does its best to maintain the information it offers in the document/code, it cannot be held responsible for any errors, defects, lost profits, or other consequential damages arising from the use of this document/code.

NETTIMELOGIC GMBH PROVIDES THE INFORMATION, SERVICES AND PRODUCTS AVAILABLE IN THIS DOCUMENT/CODE "AS IS," WITH NO WARRANTIES WHATSOEVER. ALL EXPRESS WARRANTIES AND ALL IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS ARE HEREBY DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT SHALL NETTIMELOGIC GMBH BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL AND EXEMPLARY DAMAGES, OR ANY DAMAGES WHATSOEVER, ARISING FROM THE USE OR PERFORMANCE OF THIS DOCUMENT/CODE OR FROM ANY INFORMATION, SERVICES OR PRODUCTS PROVIDED THROUGH THIS DOCUMENT/CODE, EVEN IF NETTIMELOGIC GMBH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF YOU ARE DISSATISFIED WITH THIS DOCUMENT/CODE, OR ANY PORTION THEREOF, YOUR EXCLUSIVE REMEDY SHALL BE TO CEASE USING THE DOCUMENT/CODE.

Overview

NetTimeLogic's Universal Configuration Manager is an open source solution for configuring and supervising all NetTimeLogic's IP cores. It allows to configure the configuration registers of the individual cores and allows to supervise the status of the cores. Some cores allow real-time monitoring of status information and can show this in a graph (e.g. PTP). The connection between the host and the target is done via UART (often USB USART) or 100Mbit Ethernet and has its own protocol running on it. The GUI can detect all instantiated cores in the systems and their AXI base addresses at runtime and will provide tabs for the individual cores. The solution consists of two parts, an FPGA part and a GUI part. The FPGA part allows the access to the registers, provides information about the cores in the system and makes a protocol and interface conversion between UART/Ethernet and AXI. The GUI part is the frontend for the user, it abstracts the communication interface and the individual registers and does the data presentation. Multiple instances of the tool can run in parallel and allow configuration and monitoring of multiple systems. Multiple instances of the same core in a system are handled and can be configured individually.

Key Features:

- Open Source GUI
- HW/SW co-solution
- Configuration of the cores via UART or Ethernet
- Status monitoring of the cores via UART or Ethernet
- Register access to all AXI addresses in the system (also 3rd party)
- Auto detection of available cores and base addresses
- Proprietary protocol for the UART/Ethernet connection, can also be done from a terminal (UART only)
- Multiple systems and multiple cores in a system support
- Loading of configurations from a file (plain ASCII)
- Logging of all accesses
- QT based

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	23.02.2017	First draft
1.0	16.08.2017	First release
1.1	17.09.2018	Added Ethernet interface

Table 1: Revision History

Content

Definitions

Definitions	
Counter Clock	A counter based clock that count in the period of its frequency in nanoseconds
PI Servo Loop	Proportional–Integral servo loop, allows for smooth corrections
Offset	Phase difference between clocks
Drift	Frequency difference between clocks

Table 2: Definitions

Abbreviations

Abbreviations	
AXI	AMBA4 Specification (Stream and Memory Mapped)
IRQ	Interrupt, Signaling to e.g. a CPU
PPS	Pulse Per Second
TS	Timestamp
CLK	Clock
CC	Counter Clock
ETH	Ethernet
TB	Testbench
LUT	Look Up Table
FF	Flip Flop
PWM	Pulse Width Modulation
RAM	Random Access Memory
ROM	Read Only Memory
FPGA	Field Programmable Gate Array
VHDL	Hardware description Language for FPGA's

Table 3: Abbreviations

1 Introduction

1.1 Context Overview

NetTimeLogic's Universal Configuration Manager is meant as a solution for configuring and supervising all NetTimeLogic's IP cores. It allows to configure the configuration registers of the individual cores and allows to supervise the status of the cores. The connection between the host and the target is done either via UART (often USB USART) or 100Mbit Ethernet and has its own protocol running on it. The solution consists of two parts, an FPGA part and a GUI part. The FPGA part allows the access to the registers, provides information about the cores in the system and makes a protocol and interface conversion between UART/Ethernet and AXI. The GUI part is the frontend for the user, it abstracts the communication interface and the individual registers and does the data representation. Multiple instances of the tool can run in parallel and allow configuration and monitoring of multiple systems. Multiple instances of the same core in a system are handled and can be configured individually.

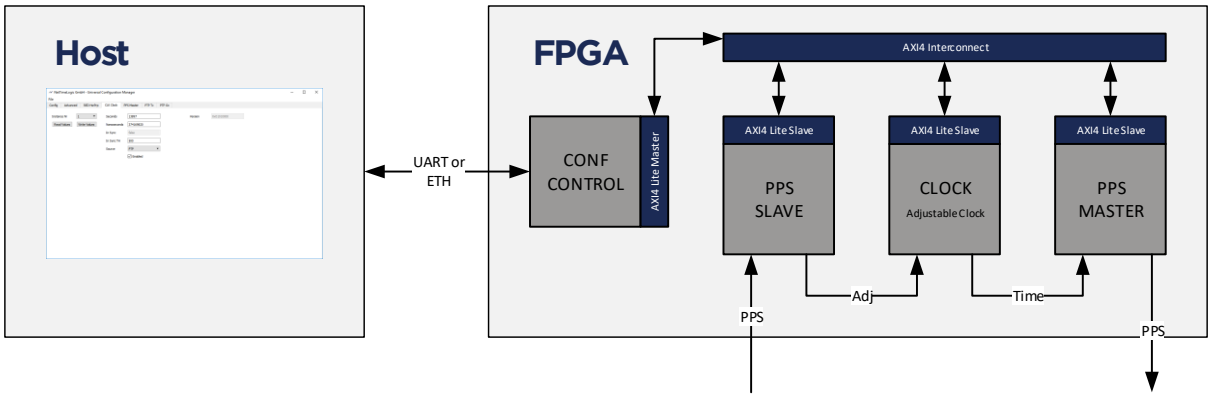

Figure 1: Context Block Diagram

1.2 Function

The Universal Configuration Manager allows to read and write registers via an FPGA configuration block which converts between a proprietary UART/Ethernet protocol and AXI. It first tries to connect to the configuration core and asks for a specific acknowledge (if in UART mode also baudrate). If it received the expected acknowledge it reads the configuration ROM in the configuration core to get the information about the instantiated cores like base address and instance number.

This register map is then shown and the individual tabs of the instantiated cores are shown. Then in the individual tabs the registers can be written and read. The registers are shown as fields with a meaningful value and therefore are abstracted from the individual addresses and bits.
For some of the cores also an auto-refresh functionality is available which polls the registers in a fixed interval and updates graphs if available.

1.3 FPGA Architecture

The core is split up into different functional blocks for reduction of the complexity, modularity and maximum reuse of blocks. The interfaces between the functional blocks are kept as small as possible for easier understanding of the core.

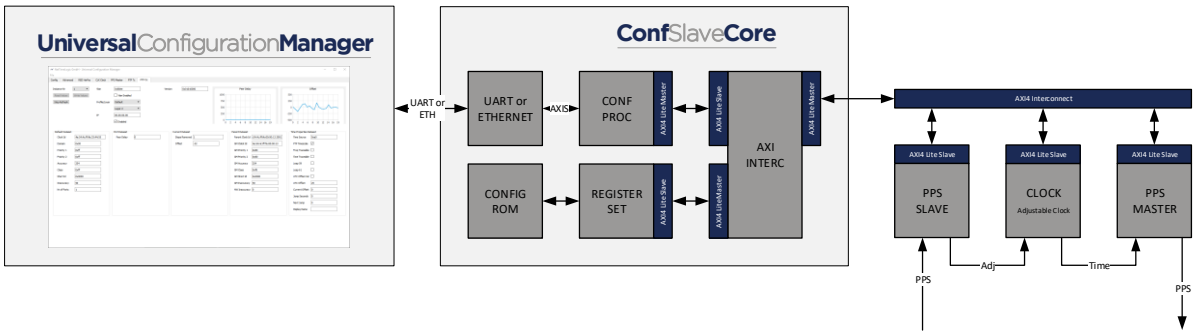

Figure 2: Architecture Block Diagram

UART or ETHERNET

This block converts the UART or Ethernet data stream into AXIS and vice versa.

Conf Processor

This block parses the protocol data received from the UART/ETH block, converts it into AXI access and generates responses towards the host via the UART/ETH block.

AXI interconnect

This block connects the internal Registerset with the AXI Master in the Conf Processor and connects to an external AXI interconnect for accessing all other registers.

Register Set

This block allows reading the configuration from the Config ROM.

Config ROM

This block stores all the information about the instantiated slaves in the ROM. The configuration has to be passed to the Conf Slave core via a structure via generics.

1.4 SW Architecture

The core is split up into different functional blocks for reduction of the complexity, modularity and maximum reuse of blocks. The interfaces between the functional blocks are kept as small as possible for easier understanding of the core.

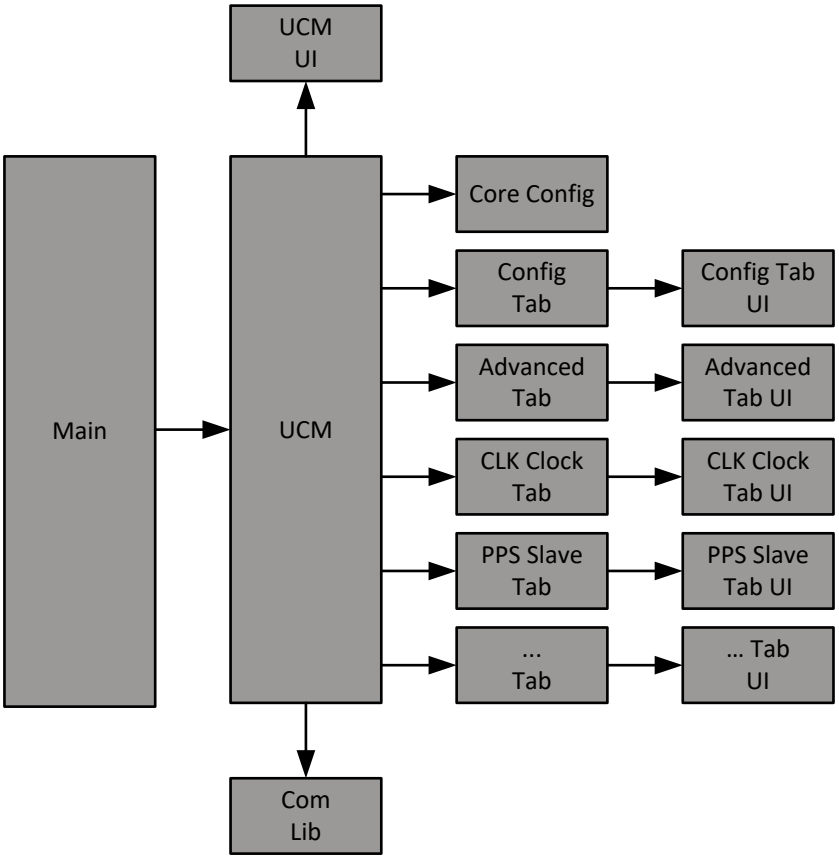

Figure 3: Architecture Block Diagram

Main

This block is the programs entry point, the only thing it does is to instantiate a Universal Configuration Manager class.

Universal Configuration Manager (UCM)

This class has references to all other Tabs, the Communication Lib, the Core Config and to the Universal Configuration Manager UI. It is the interconnection block for all other Tabs.

Universal Configuration Manager UI

This is the UI of the UCM, it basically is only the main window of the UI with an empty Tab.

Communication Lib

This class gives access to the registers and abstract the underlying protocol and UART or Ethernet interface. When a connection is opened the library checks with a connect command if a counterpart is available on this link.

Core Config

This class is a list of the cores as read from the Config ROM.

Config Tab

This class has a reference to the Config Tab UI. It is the first of the only Tabs which are active at the beginning. Its purpose is to open and close a connection to the target. It lists all available UART ports in the system so the user can choose to which system he wants to connect. When it opens a connection it immediately reads the Config ROM in the FPGA. Based on the information in the Config ROM it adds the corresponding core Tabs to the main window UI Tab and enables the tabs. Also the information from the ROM is stored in the Core Config and shown in the Address Map window. When the connection is closed all Tabs are removed from the main window UI Tab and disabled. When the connection is closed, no read and writes to registers is possible anymore

Config Tab UI

This is the UI of the Config Tab, it contains all GUI elements used for the Config Tab.

Advanced Tab

This class has a reference to the Advanced Tab UI. It is the second of the only Tabs which are active at the beginning. Its purpose is to log all activities (transfers) in

the system and allows to read and apply a configuration file for stored configs. In addition, it gives access to any AXI register in the system. Be aware that accessing an address which is not available (or doesn't respond) in the system will lead to a blocking system since there is no timeout on AXI and the AXI master will block the AXI Bus infinitely. It also allows to load and save config files and to save log files

Advanced Tab UI

This is the UI of the Advanced Tab, it contains all GUI elements used for the Advanced Tab.

CLK Clock Tab, PPS Slave Tab, ... Tab

This class has a reference to the ... Tab UI. These are the Tabs containing the functionalities according to the cores they represent.

CLK Clock Tab UI, PPS Slave Tab UI, ... Tab UI

This is the UI of the ... Tab, it contains all GUI elements used specific core's functionality.

2 Interface and Protocol Basics

2.1 UART Interface

For the communication between the FPGA and the Host a UART interface can be used. Often this UART interface is done via an USB UART. The following parameters are used:

- 1 Start bit
- 8 Data bits
- 1 Stop bit
- No Parity bit
- Baudrate 115200

2.2 ETHERNET Interface

For the communication between the FPGA and the Host also an 100Mbit Ethernet interface can be used. The following parameters are used:

- 100Mbit only
- Access via Broadcast or Unicast MAC and IP
- IPv4
- TTL: 128
- UDP Port: 0xBEEF

Data is encapsulated into a UDP/IPv4 frame as one command per frame. It also expects ASCII character and does the padding and cut off of the padding.

2.3 Protocol

The protocol run on the UART and Ethernet is a proprietary protocol defined by NetTimeLogic.
It is a simple protocol with no retransmission and therefore also not failsafe. The protocol uses ASCII character so it can also be entered directly from a terminal.
A couple of extra characters are used in the Data stream to allow synchronization of start and end of the commands as well as separation of the individual fields.
The command always starts with a '$' character followed by a two-character command code. Then individual fields can follow, each field is separated by a ',' character. After the fields a '*' character indicates the end of the command and that a checksum is followed, the two characters of checksum are followed. The command is ended with a <CR><LF> (carriage return and line feed) combination. The checksum is optional for the host and can be left away, in this case the '*' character is

also left away. The checksum XOR combines all received bytes between the '$' and '*' characters (not including) starting with 0x00 as starting value. If a checksum is present, the checksum is checked and an error is signaled by the FPGA to the host if the checksum is not correct and the command ignored.
The protocol engine in the FPGA allows empty lines and comments be transferred also via UART. A comment line starts with "--" characters. This functionality, and the fact that the checksum is optional can be used if the whole content of a file containing not only commands but also comments is copied to a terminal. The Universal Configuration Manager will always send only commands from the host to the FPGA and always with a checksum.

2.3.1 Write Command and Write Response

The two messages described here are used for writing a register.

The format of the write command looks the following:

$WC,<ADDRESS>,<DATA>*<CHECKSUM><CR><LF>

e.g. : $WC,0x50000000,0x40000001*14

A write command is always issued by the host.
The write command starts with the command identifier of the two characters "WC". Following the identifier, the 32bit AXI address to be written in hexadecimal format is added. Following the address, 32bit of write data in hexadecimal format is added. Both address and data have to start with "0x" followed by 8 hexadecimal characters.

A write command will always trigger a write response in the FPGA. If something goes wrong an error response is sent containing an error code.

The format of the write response looks the following:

$WR,<ADDRESS>*<CHECKSUM><CR><LF>

e.g. : $WR,0x50000000*64

A write response is always issued by the FPGA.
The write response starts with the command identifier of the two characters “WR”. Following the identifier, the 32bit AXI address written in hexadecimal format is added which is the address which was written in the FPGA (as in the examples, 0x50000000). The address has to start with “0x” followed by 8 hexadecimal characters.

2.3.2 Read Command and Write Response

The two messages described here are used for reading a register.

The format of the read command looks the following:

$RC,<ADDRESS>*<CHECKSUM><CR><LF>

e.g. : $RC,0x50000000*70

A read command is always issued by the host.
The read command starts with the command identifier of the two characters “RC”. Following the identifier, the 32bit AXI address to be read in hexadecimal format is added. The address has to start with “0x” followed by 8 hexadecimal characters.

A read command will always trigger a read response in the FPGA. If something goes wrong an error response is sent containing an error code.

The format of the read response looks the following:

$RR,<ADDRESS>,<DATA>*<CHECKSUM><CR><LF>

e.g. : $RR,0x50000000,0x00000001*04

A read response is always issued by the FPGA.
The read response starts with the command identifier of the two characters “RR”. Following the identifier, the 32bit AXI address read in hexadecimal format is added which is the address which was read in the FPGA (as in the examples, 0x50000000). Following the address, 32bit of read data read in hexadecimal format is added. Both address and data have to start with “0x” followed by 8 hexadecimal characters.

2.3.3 Connect Command and Connect Response

The two messages described here are used for testing the connection, for e.g. to figure out if a system is connected that supports this protocol.

The format of the connect command looks the following:

$CC*<CHECKSUM><CR><LF>

e.g. : $CC*00

A connect command is always issued by the host.
The connect command starts with the command identifier of the two characters "CC".

A connect command will always trigger a connect response in the FPGA. If something goes wrong an error response is sent containing an error code.

The format of the connect response looks the following:

$CR*<CHECKSUM><CR><LF>

e.g. : $CR*11

A read response is always issued by the FPGA.
The read response starts with the command identifier of the two characters "CR".

2.3.4 Error Response

The error messages described here is used when something goes wrong. It is always issued as reaction to another command.

The format of the error response looks the following:

$ER,<ERROR CODE>*<CHECKSUM><CR><LF>

e.g. : $ER,0x00000003*70

An error response is always issued by the FPGA.
The error response starts with the command identifier of the two characters “ER”.
Following the identifier, a 32bit error code in hexadecimal format is added.
The enumeration of the errors as of today is as following:

- 0x00000000: CRC error
- 0x00000001: Unknown command
- 0x00000002: Read error on AXI
- 0x00000003: Write error on AXI

3 Delivery Structure

```
UCM                                 -- UCM core folder
 |-Binary                           -- UCM binary
 |-Doc                              -- UCM documentations
 |-Library                          -- UCM sources
 |-Tools                            -- UCM build tools
```

4 Run

If you do not want to build the application yourself a prebuilt binary is located in Binary\UniversalConfigurationManager.exe.

5 Build

To build the core there are two possibilities, a static build for redistribution and a dynamic build which needs that you have the QT runtime installed.

For building the application QT 5.7 with MinGW 5.3.0 is used. In principal also later QT versions should be able to build the project.

5.1 Dynamic Build

For a dynamic build only two steps are needed:

1. Open the project in Library\UniversalConfigurationManager\UniversalConfigurationManager.pro with QT Creator
2. Just press the run button and it will build and lunch the application

The application can be also started from the MinGW shell

5.2 Static Build

For a static build some additional steps are needed, since first a static library of QT has to be built. You can either follow the instructions here https://wiki.qt.io/Building_a_static_Qt_for_Windows_using_MinGW or use the scripts provided:

1. Open a Windows PowerShell
2. Run the script Tools\windows-build-qt-static.ps1 (This takes several hours: ~4h)

For running the script you need and internet connection, admin rights in the PowerShell and you need 7-Zip to be installed. Also it expects that QT is installed in C:\Qt

Once the static library is built, you can build the application also with a script:

1. Run the script Tools\Ucm_ReleaseScript.bat
2. Run the application from Binary\UniversalConfigurationManager.exe

A List of tables

B List of figures