Implementing a ReboT server on a MicroBlaze.

G.Varghese, L. Butkowski, R. Rybaniec, M. Killenberg, N. Shehzad, A. Dworzanski, K. Czuba

***Abstract*—Data acquisition over an IP network is convenient for diagnostics, monitoring and control applications. The ReboT protocol (Register Based Device Access Over TCP) extends the ChimeraTK DeviceAccess library. It brings in hardware access over TCP/IP. Using ReboT, the Python and Matlab bindings provided by the library give application developers a convenient way to access hardware over the network.**

The ReboT server side is implemented on a MicroBlaze softcore which runs on a Xilinx Spartan 6 FPGA with an AXI Ethernet Lite Media Access Controller solution. We present our experience implementing the code on the MicroBlaze using FreeRTOS and the Netconn API of the LWIP stack and report on the achieved data throughput.

I. Introduction

THE 'REgister Based device access Over Tcp' (ReboT) is an in house payload format for TCP/IP that grew form the need to communicate with the Temperature Monitor and Control Board (TMCB). The TMCB is intended as a data acquisition and control device with ADCs for monitoring and DACs for control voltage generation. This board was developed at DESY in collaboration with Instrumentation Technologies[1]. It provides a 100 BASE-T Ethernet interface for data transfer and a serial interface for board management. On board processing is provided through a Xilinx Spartan 6 FPGA which hosts a Xilinx Microblaze soft core and a Xilinx AXI Media Access Controller on it.

The interface to the ADCs and DACs on the TMCB is exposed through a register space provided by the VHDL code on the FPGA. The individual registers in this memory space may be written to in order trigger specific actions, or they may be read from to retrieve collected data. Having such a register based interface to the hardware lets us integrate with the ChimeraTK DeviceAccess library[2][3]. The list of exposed register locations and their attributes, like size and address offsets are made available to the library through a DeviceAccess map file and this lets us access TMCB registers easily in our C++, Python or Matlab code (Fig. 1).

This register space is made available by the ReboT server running on the TMCB MicroBlaze. It maps payload over TCP/IP, into instructions that indicate read or write on a register of the VHDL register space of the TMCB. On the DeviceAccess library side, we have a ReboT backend that parses the response from the server received over TCP and extracts requested information from it thus letting clients of the DeviceAccess library interact with the TMCB registers.

Manuscript received on June 14, 2016

G. Varghese, L. Butkowski, R. Rybaniec, M. Killenberg are with DESY Hamburg, Germany

A. Dworzanski, K. Czuba are with the University of Warsaw, Poland

Corresponding author email: geogin.varghese@desy.de

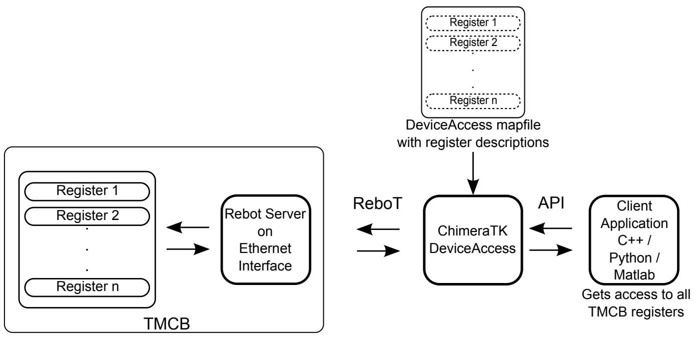
Register 1
Register 2
Register n
DeviceAccess mapfile with register descriptions
Register 1
Register 2
Register n
Rebot Server on Ethernet Interface
TMCB
ReboT
ChimeraTK DeviceAccess
API
Client Application C++ / Python / Matlab
Gets access to all TMCB registers

Fig. 1. TMCB access using ReboT and DeviceAccess library

II. ReboT server on the MicroBlaze

Use cases for the ReboT protocol involves single word (4 bytes) writes for board control, single word reads and multi word block reads from the hardware register space. The ReboT server expects commands from its clients over a TCP connection. These commands have the structure shown in Fig. 2. The server processes received commands sequentially, with the next command executed only after the current command has finished processing and a response has been sent to the client. This results in blocking calls on the the DeviceAccess API when trying to access the TMCB registers. The ReboT protocol in its current state is simplistic and limited, though it currently works for our purpose. A quick initial implementation of the server, based on the Xilinx Kernel and the lwIP socket API gave single word read speeds of around 19 ms and a TCP throughput of around 773 Kbits/sec. However these numbers were found to be inadequate and the firmware implementation was revised.

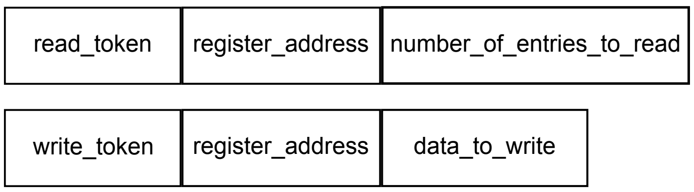

Fig. 2. Structure of ReboT commands

The revised MicroBlaze firmware implementation runs FreeRTOS v8.2[4] as the embedded operating system and uses the Netconn API of lwIP v1.4[5] for the network stack. FreeRTOS provides a framework for implementing user applications on parallel tasks, with the possibility to assign priority levels for individual tasks. For the revised implementation, the TCP/IP thread of the network stack runs on a higher priority task, with the ReboT server and the management console of the TMCB running as applications in other lower priority tasks. New applications are brought in to the firmware by implementing them on separate tasks and integrating these with the existing base firmware.

lwIP offers three API variants to work with. These are the Raw API, the Netconn API and the FreeBSD style socket API. Any of these three API options may be used for application development. The Raw API gives the best throughput, but is not thread safe and was primarily intended to be used for micro controller applications with no operating systems.The Netconn API and FreeBSD style socket API are thread safe and are convenient to use with an operating system like FreeRTOS that supports threading. The FreeBSD style socket API gives portable code compared to the Netconn API, but the Netconn API can be used to access the stacks internal buffer pointers which can be taken advantage of to gain better performance. This was a motivation for preferring this API for the ReboT server implementation.

The throughput of a TCP connection on the revised firmware was measured using the iPerf tool[6] (Fig. 3). For this tool to work, the server side should sink the received TCP payload over the connection. This was realized by implementing an iPerf dummy server on the MicroBlaze. This task accepts TCP connections and discards the payload it receives. Using this server, the throughput for our current TMCB hardware configuration was measured on a 100 BASE-T interface. The measured data throughput of a TCP connection is shown in Fig. 3.

```
------------------------------------------------
TCP window size: 85.0 KByte (default)
------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.1 sec   148 MBytes  20.6 Mbits/sec
```

Fig. 3. Throughput of TCP connection captured using iPerf tool. Maximum connection bitrate limited to 100 Mbits/sec.

TABLE I
AVERAGE TIME FOR READ AND WRITE ON TMCB REGISTERS THROUGH THE DEVICEACCESS LIBRARY (INTERNALLY USES REBOT).

	Revised Firmware (FreeRTOS, lwIP)	Old Firmware (Xilinx kernel, lwIP)
Read 4 byte register	1.12 $\pm$ 0.01 ms	19.70 $\pm$ 0.01 ms
Write 4 byte register	1.15 $\pm$ 0.04 ms	19.75 $\pm$ 0.01 ms
Read 4096 byte register	5.52 $\pm$ 0.04 ms	59.80 $\pm$ 0.01 ms

A comparison on read and write times between the reimplemented ReboT server and the initial implementation is shown in Table I. It is observed that single word reads and writes on the reimplemented version of the server are almost 20 times faster. Fig. 4 shows ReboT register access times as a function of the register size for the reimplemented server. Access times are in the order of 1-2 ms for register sizes below 1460 bytes. The growth profile till this limit is flat. This is because the fetched data is below the TCP Maximum Segment Size of 1460 bytes and hence it can be sent over one IP datagram, thus incurring only the minimal TCP/IP header overheads. This is in contrast to the reads on larger registers, where the access times tend to increase linearly because of data fragmentation over multiple IP packets bringing in increased header overheads. Fig. 5 shows the throughput of the ReboT read command with the register sizes. Single word reads give the lowest data transfer rates for the protocol while multi word read requests achieve better transfer rates. The maximum observed data rates for the protocol levels off at around 900 KBytes/sec.

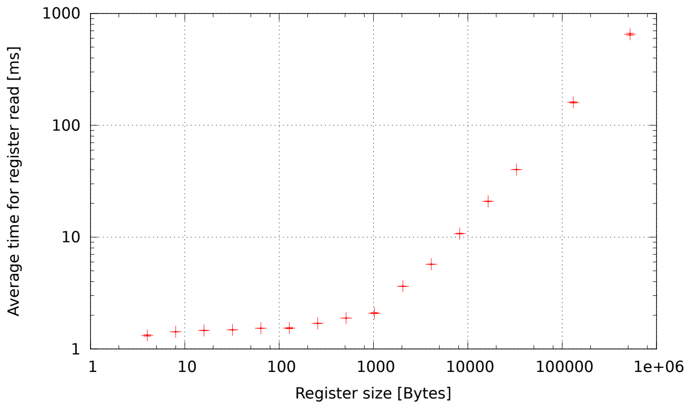

Fig. 4. ReboT read times plotted against the register size

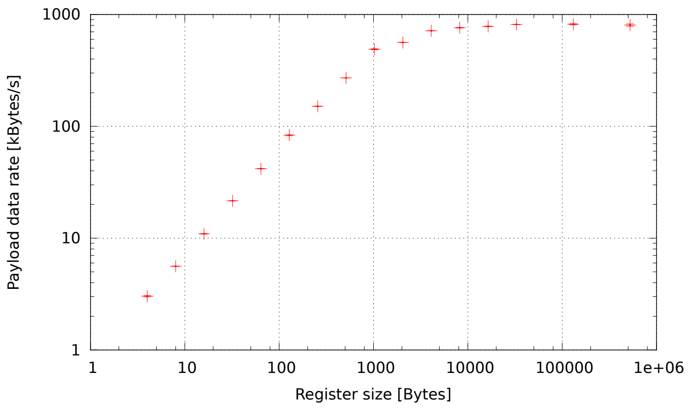

Fig. 5. Transfer rate for different register sizes

III. SUMMARY AND FUTURE WORK

The discussion presents an overview of the work done to set up a ReboT server on a Spartan 6 FPGA with the Xilinx AXI Ethernet Lite Media Access Controller solution. The MicroBlaze on the Spartan 6 used the lwIP stack with FreeRTOS to implement a ReboT server. This allows the ChimeraTK DeviceAccess library to perform single word reads (4 bytes), single word writes and block reads on the hardware over the network. The implementation was able to achieve around 800 single word reads per second and a data transfer rate of around 900 KBytes/sec for multi word reads using the ReboT protocol. These observed values are almost 20 times better that our initial implementation and are sufficient for our current needs.

REFERENCES

[1] *Instrumentation Technologies* http://www.i-tech.si/

[2] N. Shehzad et al., *Modular Software for MicroTCA.4 Based Control Applications*, *These Proceedings*, 20th IEEE Real Time Conference, Padova, Italy, 2016

[3] *DeviceAccess: ChimeraTK core library: Data Acquisition on Hardware devices*. https://github.com/ChimeraTK/DeviceAccess

[4] *FreeRTOS: A Cross Platform Real Time Operating System* http://www.freertos.org/

[5] *lwIP: A Lightweight TCP/IP stack* http://savannah.nongnu.org/projects/lwip/

[6] *iPerf2: The network bandwidth measurement tool* https://iperf.fr/