PING PONG BUFFER BASED WARPLAB EXTENSION

MATTHIAS THOMAS SCHULZ
TECHNISCHE UNIVERSITÄT DARMSTADT
SECURE MOBILE NETWORKING LAB
EMAIL: MSCHULZ@SEEMOO.TU-DARMSTADT.DE

Abstract

WARPLab is a framework to connect WARP [1] nodes to MATLAB and perform experiments. To do this, WARPLab uses buffers in the WARP's FPGA to store samples for transmission and reception. In WARPLab version 7.4.0 those buffers can hold up to 2^{15} samples.

In our ping pong buffer based WARPLab extension we increase the amount of storable samples per WARP node to 2^{28}. Instead of relying on BRAMs, we use the WARP's DDR RAM to store all samples that do not fit into the BRAM based buffers. During transceiving we use the common principle of ping pong buffers to alternately fill or read one of two halfs of the BRAM based buffers.

In this work, we present the changes performed to the original WARPLab version and we give an introduction on how to use our extension.

1 DISCLAIMER

Our software and the FPGA image is only a proof of concept and still contains many bugs. The authors take no responsibilities for possible damage due to neither the use of our software nor the FPGA image.

2 INTRODUCTION

With our extension we increase the possible number of samples a WARP node can store for either transmission or reception. The idea is as follows: To transceive up to 2^{15} samples, WARPLab stores them in the existing BRAM buffers. As soon as more samples should be transceived, WARPLab stores them in the WARP's DDR RAM of which 1 GB is accessible by the processor. During transceiving, the software running on the WARP node constantly checks the address of the tranceive buffers. Whenever the address modulo 2^{15} exceeds 2^{14} samples, the software orders the DMA to refill the first half of the buffer with samples from DDR RAM. As soon as the address counter wraps around and the buffer is again read from the beginning, the software orders the DMA to refill the second half of the buffer. Which parts of the DDR RAM are used to store samples for each buffer can be freely configured from MATLAB using custom WARPLab commands. This allows to allocate buffer space depending on the demands of the application.

3 CHANGES IN THE FPGA IMAGE

To perform the ping pong buffering, we extend the *w3_warplab_buffers*-pcore. The length of the address counters has to be increased to count up to 2^{28}. The outcoming address is sliced to 15 bits and passed to the existing buffers. The full address is constantly written to a shared register *Addr* which is read by the processor to coordinate the refilling of the buffers.

4 CHANGES IN THE MICROBLAZE SOFTWARE

The software is executed on the FPGA to coordinate the communication with MATLAB and fill the buffers. To be able to write to and read from the DDR RAM using MATLAB we intend to use the existing functions that are also used to exchange data with the BRAM buffers. Therefore, we define a new interface identifier $BB\_DDR = 0x10$ and implement operations to write to and read from DDR RAM using the DMA in *wl_baseband.c*. Whenever, a transmission is triggered, the software waits until the transmission is over. This is done in function *trigmngr_triggerIn* in *wl_trigger_manager.c*. As we need to update the buffers' contents, we implement our logic in the loop that waits for the transmission to be over. Here we constantly check the *Addr* register introduced in Section 3 and instruct the DMA to transfer data between buffers and DDR RAM. To define which parts of the DDR RAM are allocated for which buffer, we store addresses that can be configured from MATLAB. Therefore, we extended the *wl_user.c* code with custom commands.

5 CHANGES IN MATLAB CODE

In MATLAB we disable multiple checks that avoid to transfer more than 2^{15} samples to the WARP nodes. Additionally, we extend the *write_IQ* baseband command to set an offset that allows us to write to arbitrary positions in the DDR RAM. We also implement custom commands to manage the memory allocation of the DDR RAM in *user_extension_example_class.m*.

6 TESTING OUR DESIGN

We built our design based on the four antenna WARPLab version. To make it run on WARPs without an extension board, we simply set *WARPLAB_CONFIG_4RF* to zero but did not recompile the FPGA hardware.

You can find ready to use *.bin* files for your SD card under *WARP_SD_IMAGES*. To interface this design and use the new features the modified WARPLab version coming with this work is required. Therefore, run the *wl_setup.m*. You might need to recompile the MEX file for your system. It is based on SVN rev. 3296 with minor changes. Under *ddr_transfer_example* there is a *pingpongtest.m* script which uses one WARPv3 node to transmit a 2^{22} samples long linearly increasing waveform from interface A to interface B.

7 KNOWN PROBLEMS

- Currently, MATLAB crashes when the DDR offset for RX buffers is different from 0 when using the MEX file.
- Not using the MEX file, the number of transferable numbers is more limited than with the MEX file as is comes to a timeout due to the long time required to transfer data to and from the WARP.
- MATLAB behaves wired when the number of samples is increased to more than roughly 2^{25}.

8 DO YOU USE OUR EXTENSION?

If you (intend to) use this WARPLab extension, please write us an email and let us know. We just want to see how many researchers are interested in such an extension. If you fix open bugs in the code, please make the bug fixes available to us or the public.

[1] (2013) Rice university WARP project. [Online]. Available: http://warp.rice.edu