University of Zurich[UZH]

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CPG project: FPGA report

September 24, 2020

1 Introduction

This document describes the interface between the computer and the CPG chip through FPGA. This project includes two main parts : the API python interface and the FPGA project. In the following, at first, the project design specification will be defined. Then the high-level block diagram of the AER protocol on FPGA will be introduced. Finally, the CPG_project that includes all the design sources will be itemized in more detail.

2 Technical specification

— Hadrware Description Language : VHDL
— Simulator tool : Vunit, GHDL
— Synthesizer tool : Xilinx Vivado 18.02
— Operating system : Linux (Ubuntu)
— FPGA development board : XEM7310
— FPGA device : Xilinx Artrix-7 XC7A75T
— Application programming interface (API) : python 3.6

3 Communication protocols

Address-event representation (AER) was proposed in 1991 by Sivilotti [1] for transferring the state of an array of neurons from one chip to another. AER is a communication protocol originally proposed as a means to communicate sparse neural events between neuromorphic chips. The block diagram in Figure 1 shows the organization of the various AER blocks within two-dimensionally organized transmitter and receiver chips. The signal flow between any two communicating blocks is carried out through the Request (Req) and Acknowledge (Ack) signals. Figure 2 illustrates the 4-phases handshaking protocol which we used in this project. Steps : 1) Both Req and Ack are initially in zero states. 2) A new data word is put on the bus, and Req is raised, and control is passed to the receiver. 3) When ready, the receiver accepts the data and raises Ack. 4) Req and Ack are brought back to their initial state in sequence. We use the same AER communication protocol to send the neuromorphic biases as well as monitor the events using a PC. Therefore we need to interface the chip and the PC using an FPGA which hosts an AER core.

3.1 Implementing AER on FPGA

AER core is designed using VHDL hardware description language and implemented on Xilinx Artix-7 XC7A75T FGG484 hosted on Opal kelly XEM7310.
As depicted in figure 3, AER is composed of two main parts : One part is for getting event packets from the PC and sending them to the CPG. The other part is for receiving events from CPG and reporting to PC.
The first part is composed of “Deserializer” and “Sequencer”. The Deserializer receives input packets from a pipe-In endpoint sent by the PC, and Sequencer is responsible for sending events to the neuromorphic chip based on their timing sequences.
The second part contains “Reciever” and “Serializer”. The Receiver gets events produced by CPG, and the Serializer manages them in a few packets and sends them to the PC using a pipe-out endpoint. Based on the synthesized results from Table 1, the entire system uses only a small number of all available resources.

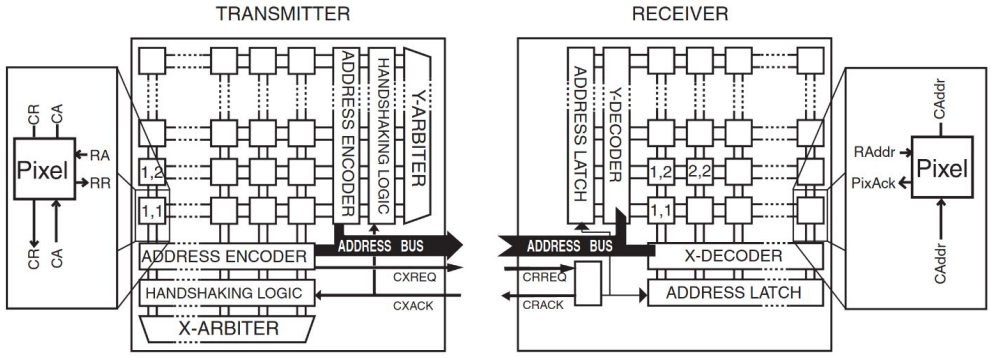

FIGURE 1: Block diagram of AER blocks on a 2D transmitter chip and 2D receiver chip. Details of signaling are described here [2]. The signals CXREQ and CXACK can be connected to CRREQ and CRACK, if no translation of addresses from the transmitter are needed

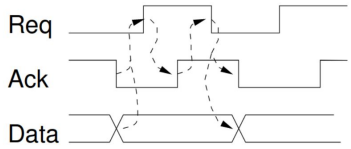

FIGURE 2: A 4-phase bundled-data protocol

4 CPG Project manual

The CPG_project contains all required design sources for implementing the AER protocol on FPGA. This project includes a few subfolders :

- AER_CPG_wrapper : This folder contains the FPGA project created using Xilinx Vivado 18.02 for implementing on Xilinx Artix-7 XC7A75T FGG484 device.
- CPG_comm :This folder contains python level API for controlling sending and receiving packets to/from chip thorough FPGA.
- src : This folder includes all design sources, constraint file, opall kelly libraries and all required files for testing simulations.
  - Design libraries : XEM7310-A75
  - Design constraint file : xem7310.xdc
  - Design sources :
    - AER_CPG_wrapper.vhd
    - InputEventsChain.vhd
    - OutputEventsChain.vhd
    - BtPipeDeserializer.vhd
    - BtPipeSerializer.vhd
    - EventReceiver.vhd
    - EventsSequencer.vhd
    - InferredBlockRam.vhd
    - Reset_Controller.vhd
    - counter.vhd
    - CustomFifo.vhd
  - Simulation libraries :

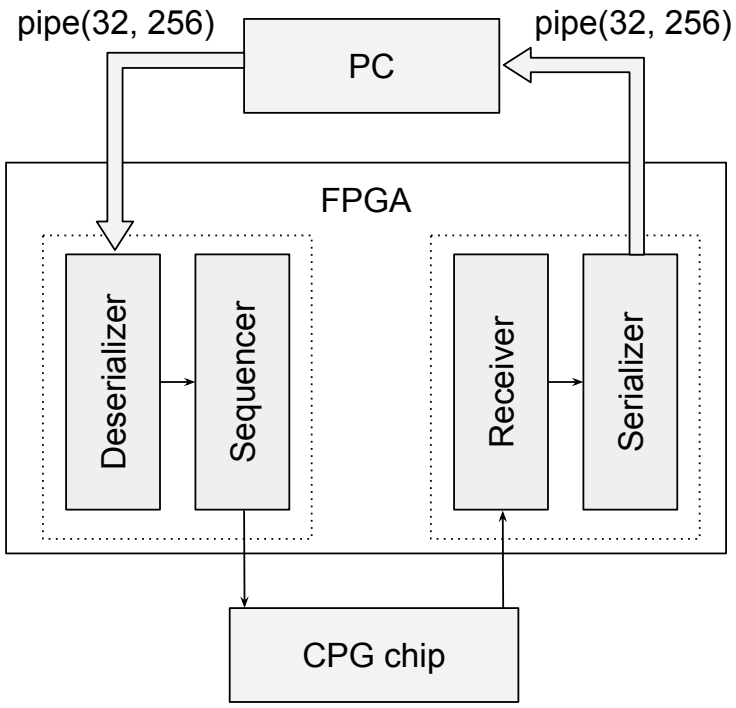

FIGURE 3: System block diagram including PC, FPGA, and CPG-chip : PC and CPG-chip communicate together through FPGA.

  - xilinx-vivado
  - mappings.vhd
  - okBTPipeIn.vhd
  - okBTPipeOut.vhd
  - okHost.vhd
  - okLibrary_sim.vhd
  - okPipeIn.vhd
  - okPipeOut.vhd
  - okRegisterBridge.vhd
  - okTriggerIn.vhd
  - okTriggerOut.vhd
  - okWireIn.vhd
  - okWireOR.vhd
  - okWireOut.vhd
  - parameters.vhd
- Running the test environment : run.py
- Test-Bench file
  - AER_CPG_wrapper_ok_tb.vhd : This test-bench simulates the design in the Opal-kelly environment.

Table 1: Results of the system synthesized by Xilinx Vivado tool.

Resource	Utilization	Available	Utilization%
LUT	1296	47200	2.745
LUTRAM	72	19000	0.378
FF	1043	94400	1.10
BRAM	6	105	5.71
IO	80	285	28.07
BUFG	4	32	12.5
MMCM	1	6	16.66

5 FPGA Closed-loop testing

PC and FPGA communicate by sending packets through pipe-endpoints. Each packet contains a header (the first word) and maximally 256 words (each word has 32 bits). The header specifies each packet carries how many words. Each event is defined by 2 words, one for timestamp and the other for the address.

To test the AER implemented on the FPGA independently, a closed-loop system is designed and shown in figure 4. This closed-loop testing setup can be done both in the simulation and implementation levels.

The data provided for testing is a packet that contains some of the CPG's parameters called "biasgens". The whole input packet and the relevant expected outputs are illustrated in table 2.

Fig. 3 shows the waveform of the system in the simulated Opalkelly's environment.

Table 2: Results of the system synthesized by Xilinx Vivado tool.

Input packet	Biasgens	Expected outputs
0x12,0x7A,0x08,0x00	Header	
0x0A,0x00,0x00,0x00	Timestamp	
0xBD,0x7A,0x08,0x00	PADBias	87ABD
0x0C,0x00,0x00,0x00	Timestamp	
0x15,0x08,0x09,0x00	IDC	90815
0x0E,0x00,0x00,0x00	Timestamp	
0x0B,0x08,0x0C,0x00	VolSat	C080B
0x11,0x00,0x00,0x00	Timestamp	
0x15,0x88,0x08,0x00	NaSat	88815
0x13,0x00,0x00,0x00	Timestamp	
0x6F,0xE4,0x08,0x00	gleak	8E46F
0x15,0x00,0x00,0x00	Timestamp	
0x01,0x82,0x0C,0x00	AhpSet	C8201
0x17,0x00,0x00,0x00	Timestamp	
0x01,0x12,0x0C,0x00	Calu	C1201
0x19,0x00,0x00,0x00	Timestamp	
0x01,0x62,0x08,0x00	PUTHresh	86201
0x1B,0x00,0x00,0x00	Timestamp	
0x09,0xF2,0x08,0x00	Eleak	8F209

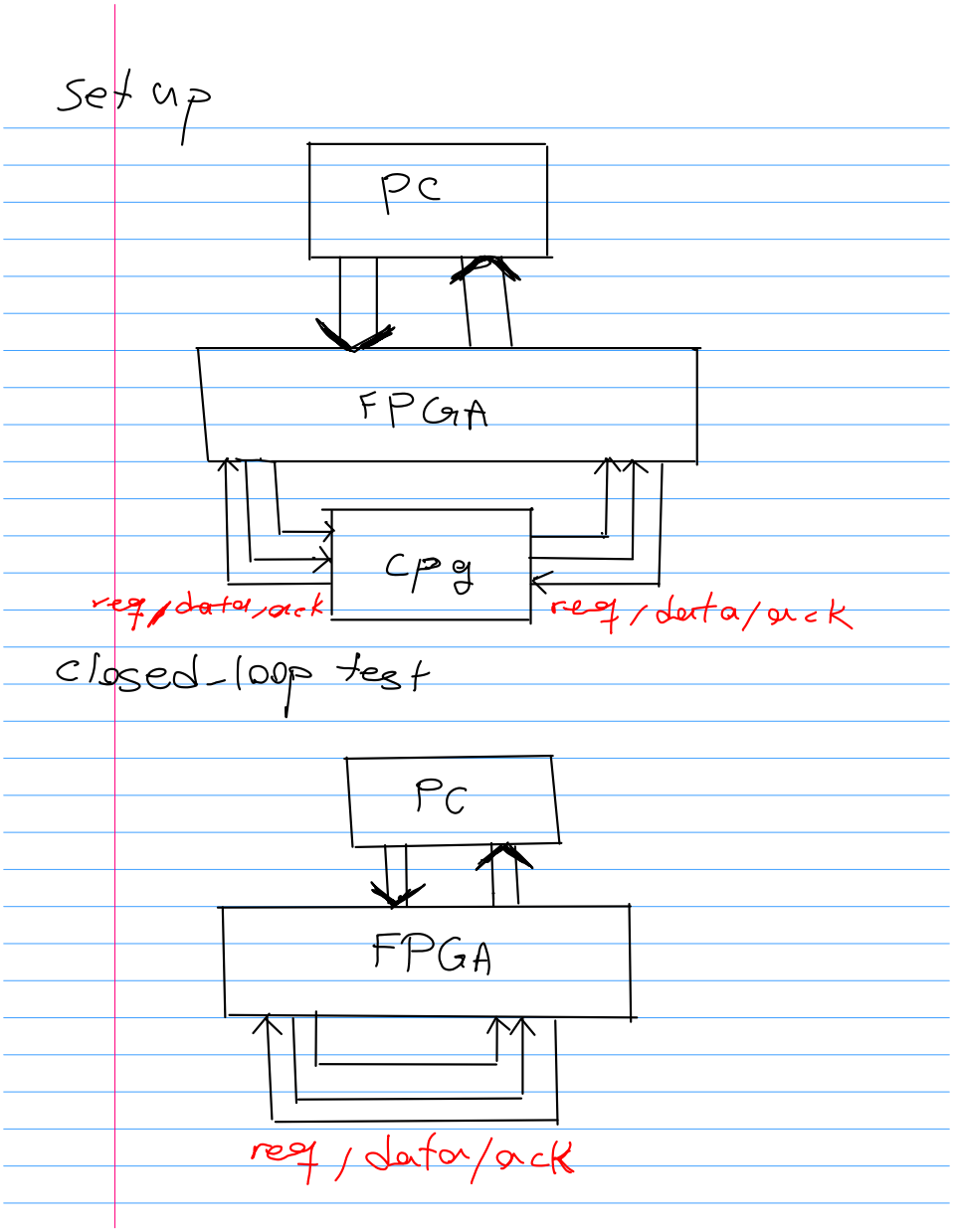

FIGURE 4: Closed-loop testing setup

Références

[1] M. Sivilotti, "Wiring Considerations in Analog VLSI Systems, with applications to Field-Programmable Networks," Ph.D. dissertation, California Institute of Techno-

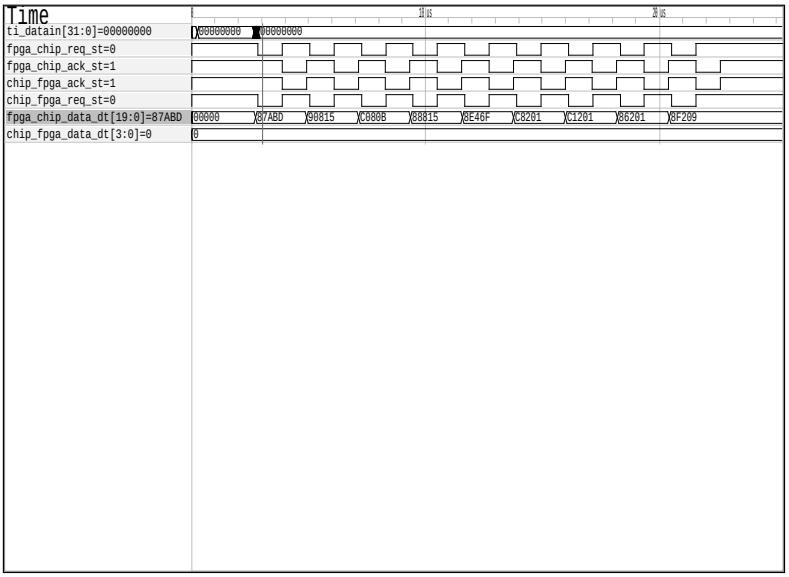

FIGURE 5: Closed-loop testing setup

logy, 1991.

[2] S.-C. L. and Tobi Delbruck and Giacomo Indiveri and Adrian Whatley and Rodney Douglas, *Event-Based Neuromorphic Systems*. John Wiley & Sons, Ltd, 2015.