1 Setting up the eDVS

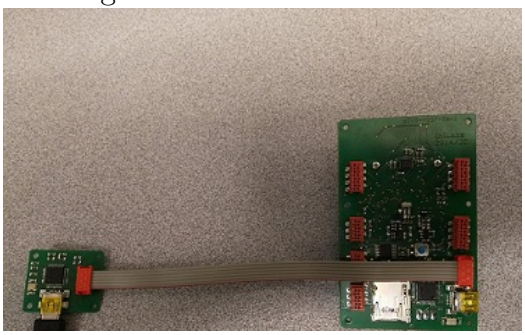
Figure 1: eDVS camera

Packets(events) can be brought out of the eDVS[?] using SPI or UART protocols. In this project we chose UART protocol. We have to first understand the format of the packets. We have to send a character 'E+' serially to the eDVS so that it can start streaming the events. According to inilabs[?] user guide for eDVS and Pushbot there are 5 types of events. Any one of the 5 possible streaming options can be selected by sending the character '!Ex' serially to the eDVS. x varies from 0 to 4. Different formats for data streaming are as follows:

!E0 - 2 bytes per event, binary: 1yyyyyyy.pxxxxxxx (default) (p = polarity)
!E1 - 3..6 bytes per event; the above address format followed by 1..4 bytes delta- timestamp (7bits each)
!E2 - 4 bytes per event (as !E0 followed by 16bit absolute timestamp)
!E3 - 5 bytes per event (as !E0 followed by 24bit absolute timestamp)
!E4 - 6 bytes per event (as !E0 followed by 32bit absolute timestamp)

Connect eDVS to the computer using the USB to UART module given. The connection should look like that of in the picture below.

Figure 2: eDVS connections



In windows open the device manager and see the port number of the newly connected device. In ubuntu run the command dmesg—grep -i tty to see the port name. Run the following code.

```
import time
import serial
from binascii import hexlify

ser = serial.Serial(
port='COM6',
baudrate=4000000,
parity=serial.PARITY_NONE,
stopbits=serial.STOPBITS_ONE,
bytesize=serial.EIGHTBITS
)

ser.isOpen()
ser.write('E+ \n')
ser.write('!E2 \n')
for i in range(0,10):
    data=ser.read()
    hex1=hexlify(data)
    num=(int(hex1,16))
    num1= format(num, '08b')
    print num1
ser.close()
```

Here we are using the option !E2 for packets. The resulting packets should look like 1yyyyyyy.pxxxxxxx.tttttttt.tttttttt
Run the code and see the data streaming. Switch off everything and re run the code. Now you will see that the first 4 bytes are as follows.
01000101
00101011
00100000
00001010
The next bytes should look like below:
11110111
00011011
00000001
01111110
11111111
00001111

00000001
10011011
10100001
01101000
00000100
00011011
11110111
00000001
00001101
After every 4 bytes the new byte starts with the bit 1. This ensures that the data is in accordance with the prescribed format for the option !E2. First 4 bytes in ascii denote E+. This ends the setup of the eDVS and understanding the "events" that are streamed out.

2 SpiNNaker Packets and the SpiNNaker Link

2.1 SpiNNaker Packets[?] :

Each processor node on SpiNNaker includes a communications controller which is responsible for generating and receiving packets to and from the communications network. There are 4 types of packets:

- multicast (MC) neural event packets routed by a key provided at the source;
- point-to-point (P2P) packets routed by destination address;
- nearest-neighbour (NN) packets routed by arrival port;
- fixed-route (FR) packets routed by the contents of a register.

Multicast packets are our interest. For more information on the other packets see the SpiNNaker datasheet[?] .

2.2 Neural event multicast (MC) packets (type 0)

Neural event packets include a control byte and a 32-bit routing key inserted by the source. In addition they may include an optional 32-bit payload:

8 bits	32 bits	32 bits
control	routing key	optional payload

The 8-bit control field includes packet type (bits[7:6] = 00 for multicast packets), emergency routing and time stamp information, a payload indicator, and error detection (parity) information:

7	6	5	4	3	2	1	0
0	0	emergency routing	emergency routing	time stamp	time stamp	payload	parity

Payload and Parity are our interest, other bits can be set to zero. For more information see SpiNNaker datasheet. If the 32 bit optional payload is used in the 72 bit packet then payload bit should be set to 1 in the control register otherwise it should be set to 0. Parity can be set by taking the XNOR of the whole packet. Other bits should be set to 0. In our application the 2 time stamp bytes can be assigned to the optional payload section. Finally, a 72 bit packet should look like below:

32 bits	16 bits	1 bit	15 bits	7 bits	1 bit
payload	chip address	1	coords	0000000	parity

32 bits payload should have time stamp information. In this case it should look like:
00000000.00000000.Most significant 8 bits. Least significant 8 bits.
The 16 bit chip address is the external node(FPGA) address. In this case it is 16'hfefe. For more information on this see AppNote 8 - Interfacing AER devices to SpiNNaker using an FPGA. The 15 bit coords can be formed as pyyyyyyy.xxxxxxx, here p is polarity which is defined in the eDVS section.

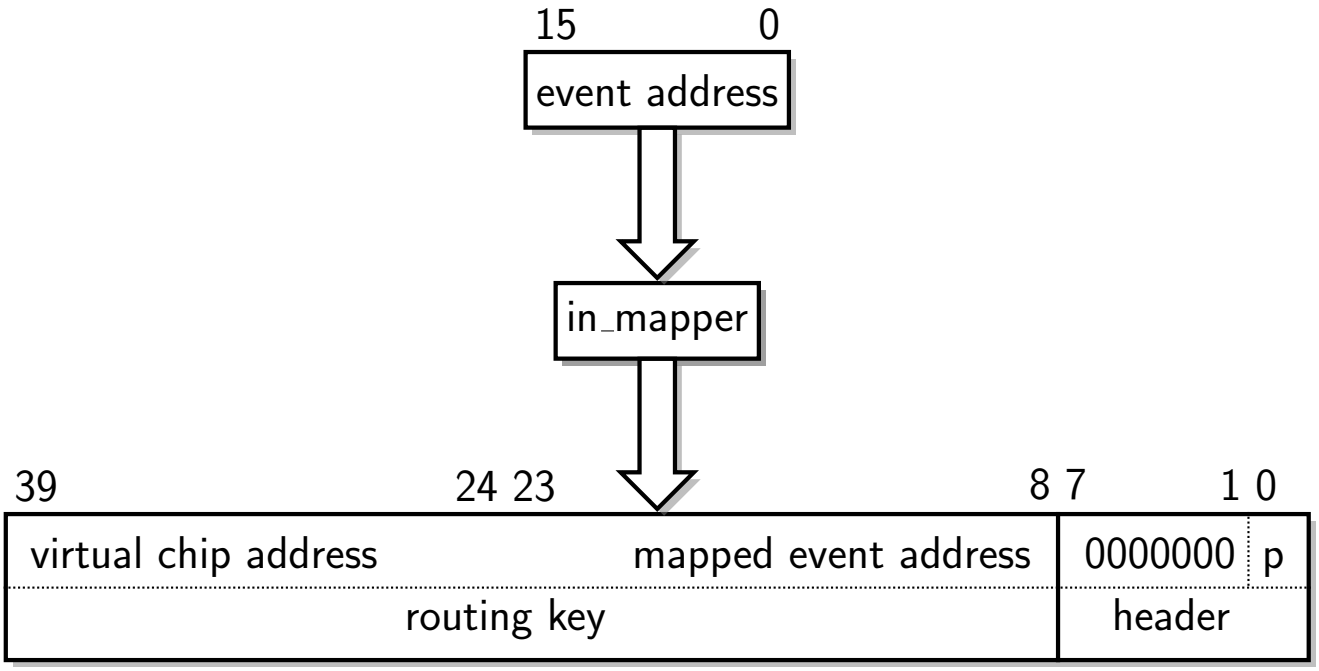
2.3 SpiNNaker Link[?] :

SpiNNaker Link is a 34 pin header found on the SpiNNaker PCB. SpiNNaker supports injection of packets using 3 different methods. Ethernet interface, SpiNNakerLink and SpiNNLink. SpiNNaker link is of our interest.

For more information on other two methods see Lab manual SpiNNaker interfacing external devices[?] and AppNote 7 - SpiNNaker Links.[?] SpiNNaker link follows a 2-of-7 protocol.[?] Verilog module for SpiNNaker Link section is posted on github[?] by University of Manchester. The goal of this project is to re-design packets that are obtained from eDVS so that they can be injected into SpiNNaker. Without going into much details the verilog module can be treated as a blackbox.

3 Rearranging the eDVS packets

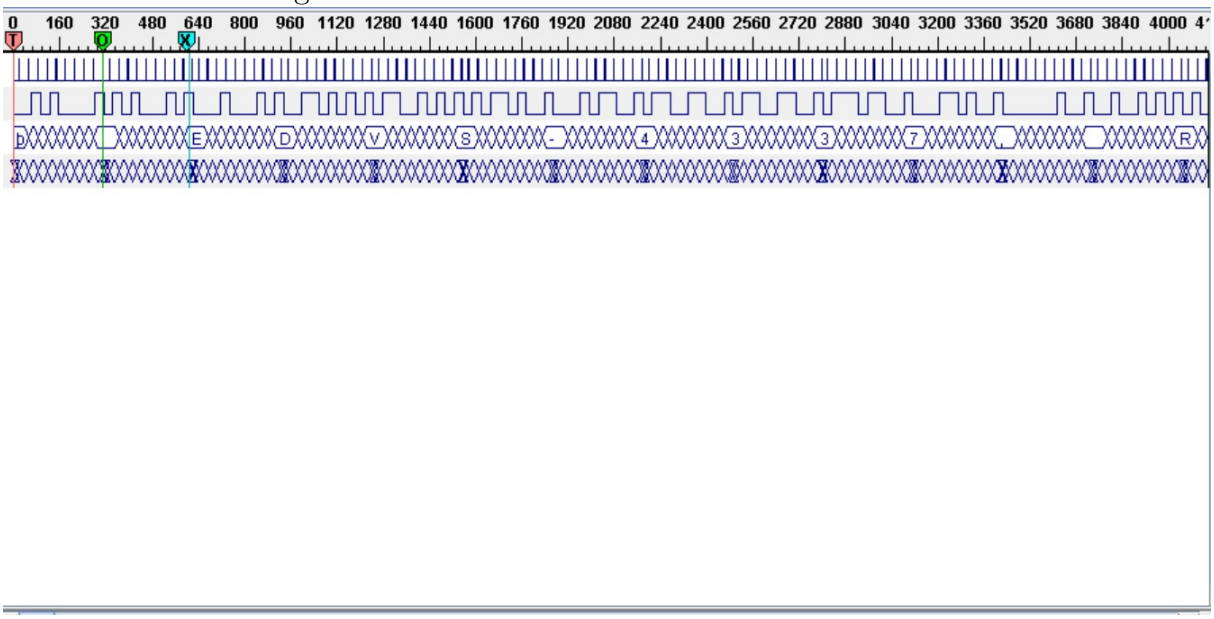
We have seen above that the eDVS[?] and SpiNNaker[?] work on different protocols. This section briefs about converting the eDVS[?] packets into SpiNNaker[?] packets. Since we are using the UART functionality of the eDVS to receive events, the goal of this project is is to write a driver for UART. We chose $4M$ baudrate on the eDVS[?] as suggested by inilabs[?]. The sub sampling on the UART code was chosen so that the FPGA samples almost at the middle of the every new bit arriving. As discussed earlier we chose $!E2$ format for the packets, so a 32 bit register was used to store the incoming events(32 bits each). Then the 32 bits were rearranged into 72 bit SpiNNaker[?] packets as discussed above. Handshaking is implemented on both the sides *i.e* eDVS[?] and SpiNNaker[?]. These 72 bit SpiNNaker[?] packets were dumped into a circular buffer. Every time the buffer crosses a pre-set watermark or if it is full the CTS pin on the UART[?] is pulled low so that the eDVS[?] stops sending the events. If buffer is empty RDY signal to the spinnaker_link verilog module is pulled low so that the SpiNNaker[?] stops reading the packets from the buffer. The mapping should look like this.



4 Simulation and implementation using Xilinx ISE

Firstly, UART interface for eDVS camera was implemented on a FPGA. Available modules from SpiNNakerManchester/spio were used for implementation. Necessary changes were made to the code to suit our requirements. The basic UART module was implemented on Zynq board from Xilinx. Data which was received from the eDVS looks like below. Xilinx's Chipscope was used to observe the data. Initial few bytes can be seen in the picture. EDVS-4337... these are the first few bytes that the eDVS sends out before starting to stream out the actual events data. Initially a 8 bit accumulator register was used to accumulate the incoming serial data. Ones this was achieved accumulator register's width width was increased to 32 bits so that a full event can be captured. Ones a full event is captured a valid signal and a 72 bit SpiNNAker packet was generated and placed into a circular buffer. This process continues for every clock cycle. A buffer is used here because SpiNNaker and the eDVS operate on different frequencies.

Figure 3: UART data from the eDVS



A screenshot of the simulation using a testbench is shown at the end of this section.
The timing diagram captures the whole protocol conversion procedure happening in the FPGA. Some of the important signals are:

- SPIKE[71 : 0];
- fifo_occupancy_i[3 : 0];
- Data_OUT_temp[31 : 0];

SPIKE[71 : 0] is used to represent the SpiNNaker packet, fifo_occupancy_i[3 : 0] is used to represent the buffer occupancy, Data_OUT_temp[31 : 0] is used to represent the accumulated 32 bit event data from the eDVS.
Some important signals within the verilog module for fifo[First in First Out] buffer.

- IN_RDY_OUT → Low if the buffer is full;
- IN_VLD_IN → To tell if incoming data from the camera is valid or not;

- OUT_RDY_IN → Ready from spinn_link module;
- OUT_VLD_OUT →Low if buffer is empty, spinn_link shouldn't read from the buffer;
- IN_DATA_IN → Store the SPIKE in the buffer.

The watermark level for fifo buffer was set at 8. We can see in the attached screenshot that the CTS signal was pulled to 0 as soon as the fifo occupancy is 8.

Figure 4: Simulation result for SpiNNaker packets