MICROCONTROLLER INTERFACE FOR EMBEDDED VISION SENSOR

eingereichtes
PROJEKTPRAKTIKUM
von

cand. ing. Razvan Tiganu

Tel.: 01776796318

cand. ing. Roman Ursu

Tel.: 017698436632

Lehrstuhl für
STEUERUNGS- UND REGELUNGSTECHNIK
Technische Universität München
Univ.-Prof. Dr.-Ing../Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Betreuer:	cand. Dr. Cristian Axenie
Beginn:	22.10.2012
Abgabe:	18.01.2013

Abstract

This paper describes the development of the practical course Microcontroller Interface for embedded Vision as part of the Neuro-Robotics Program. The core of the project is the development of a printed circuit board featuring a neuro-biologically inspired even-based vision sensor and a high performance microcontroller. The starting point and the goals are presented at first. The details of the hardware and software development are then explained. Finally, the actual results are compared to the expected ones.

Zusammenfassung

Es handelt sich in diesem Bericht um die Beschreibung des Projektpraktikums Microcontroller Interface for embedded Vision das im Bereich des Neuro-Robotics ausgeführt wurde. Die Hauptaufgabe des Projekts besteht aus der Entwicklung einer electronischen Schaltung für die Kopplung eines biologisch inspirierten Sensors mit einem Microcontroller. Am Anfang werden die Ausgangspunkte sowie die Ziele vorgestellt. Dann werden die unterschiedliche Aspeckte der Entwickung des Systems erläutert. Schliesslich, werden die tatsächliche Ergebnisse mit den erwarteten Zielen vergleicht.

Contents

1 Introduction

One of the most interesting aspects of the world of robotics is vision. The ability to perceive the surrounding environment visually opens the path to solving many problems such as object detection, obstacle avoidance, motion tracking etc.

The starting point in tackling any of these problems is the method by which the input data is obtained, more precisely the vision sensor. The most common approach is the use of technology borrowed from image and video recording devices designed to be used by humans. The first problem that arises is that these types of sensors provide large amounts of information which, while useful for the very complex human sight, manage to overburden systems designed with simple goals in mind, such as motion detection or pattern recognition.

This document presents an alternative to the conventional vision sensors and describes the progress of a small development project dealing with the control system of the new sensor.

2 Project Plan

2.1 Requirements and Goals

The starting point of this project is an already designed and implemented control system that makes use of the first generation of the Dynamic Vision Sensors. This system makes use of a microcontroller unit to retrieve measurement data from the sensor and forward it to other devices via a serial interface.

The new system will operate the second generation of the DVS which presents changes in existing functionality as well as additional features. The microcontroller unit to be used will also be replaced by one faster and better in terms of computational power which will be accompanied by an external SRAM module. The components are connected as depicted in the block diagram below.

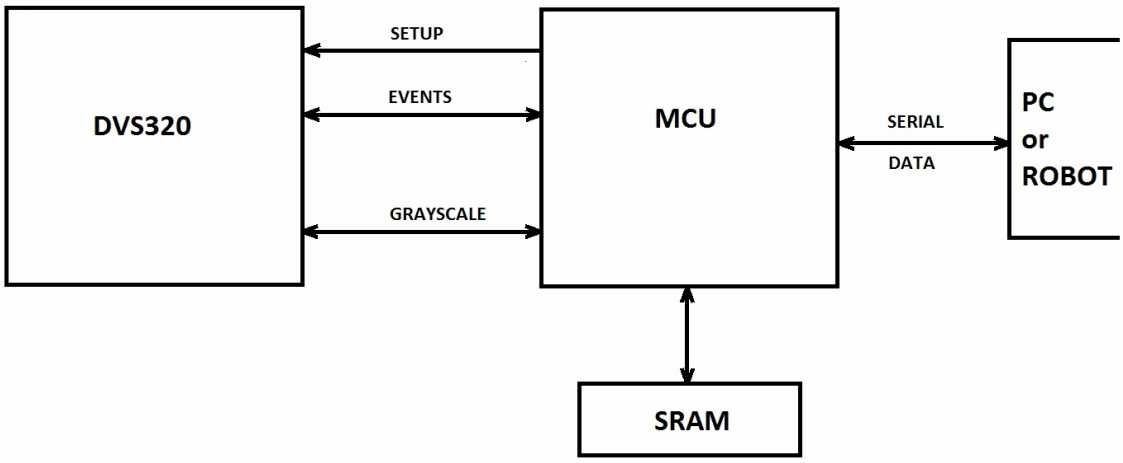

Figure 2.1: General diagram of the system

Additionally, the existing software from the old system will be ported and enhanced with additional features and, as optional goals, attempts must be made to increase the data throughput of the system. Finally the new system will have to be characterized in terms of noise robustness and speed processing.

2.2 Task and Work Division

The tasks to be performed are divided primarily into two: redesigning the hardware and developing the software to be used on it. As the team that works on the project is composed of 2 persons, these tasks may be performed in parallel, at least in the

initial phase, as long as some considerations are made. By making use of an evaluation board that houses the microcontroller unit to be used in the project, the software implementation may begin even before the hardware has been built.

3 Development

3.1 Resources

The sensor to be used is called Dynamic Vision Sensor 320 and it represents an improved version of the DVS128. Unlike conventional vision sensors whi0ch deliver a sequence of frames at a fixed rate, the DVS sensors send only local pixel changes caused by movement in a scene at the time they occur. Typically, the sensitive element of the DVS can be seen as a 2D array where each pixel has an x address and an y address. The DVS sends then the addresses of the pixels experiencing a change in the radiation intensity.

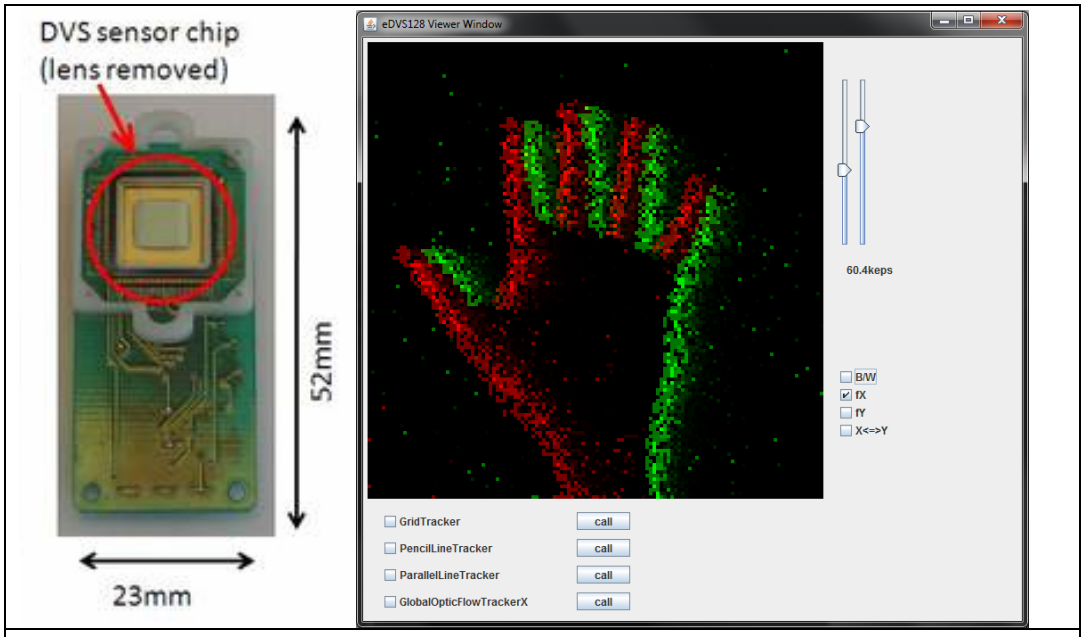

Figure 3.1: The existing system

This event based system yields a stream of data at millisecond time resolution which is considerably easier to process as it does not contain redundant information such as background. Despite the very high data rate of this sensor type, discarding a considerable amount of information from the scene due to its redundancy leads to great improvements in power consumption, data storage, computational requirements and dynamic range.

Because the exposure time for each pixel is variable, adaptive and independent of the others, the sensor is able to easily deal with very bright or very dark environments.

Aside from a better resolution (320x240 vs. 128x128), the DVS320 can also deliver a grayscale image (snapshot) in parallel to the pixel events extending functionality towards static scenes.

3.2 Hardware Development

3.2.1 Concept

The DVS320 sensor delivers two kinds of measurement data: the x and y coordinates of pixels who detect movement through an event triggered parallel bus and pixel-wise light intensity measurement as an analogue signal. Both communication interfaces require active participation from the microcontroller side. Additionally, the microcontroller must provide setup data for the bias capacitors used by the sensor. These capacitors are used to control the sensitivity level of the sensor.

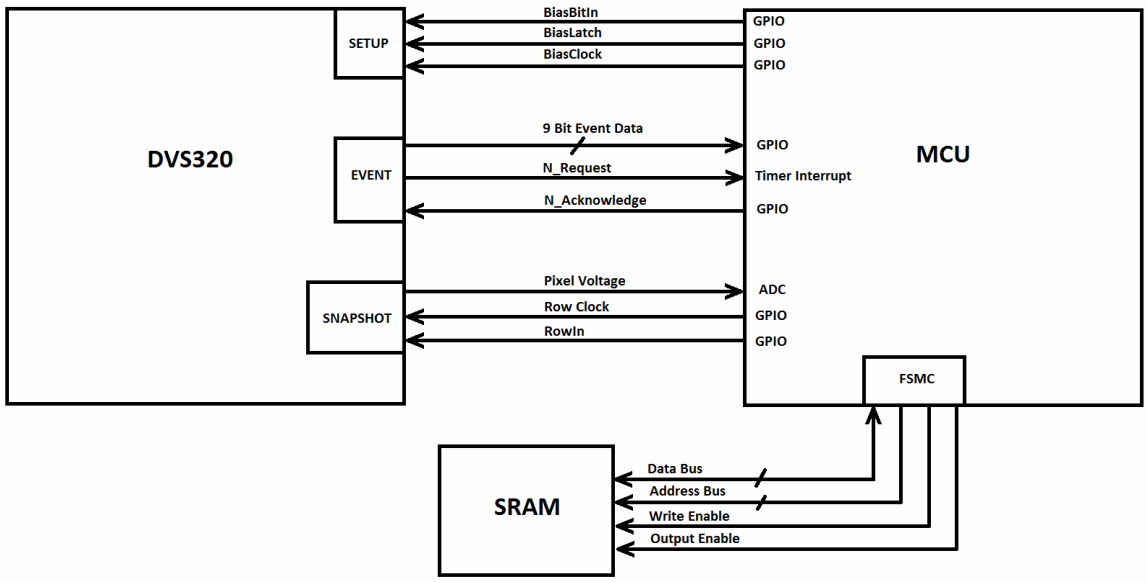

Figure 3.2: DVS-microcontroller communication

The SRAM module, on the other hand, simply needs to be connected to the correct pins of the microcontroller's Flexible Static Memory Controller (FSMC). Additional SRAM is desired for future applications that will be implemented on the microcontroller as it is assumed that its internal memory might not suffice, for the software developed in this project it is not a hard requirement, however.

The design is to be made using CadSoft's Eagle Light Edition. The PCB will be built using a milling machine and will be populated by hand.

3.2.2 Schematics

Since the sensor has numerous pins with various functionalities, to make connecting to the microcontroller an easier process, additional effort has been invested

into making a schematics symbol with named pins which have been grouped by functionality. To operate properly, the sensor needs to receive setup parameters from the microcontroller. This information is stored in the capacitors that need to be connected to the sensor. The image below depicts the resulting symbol together with the connected peripherals.

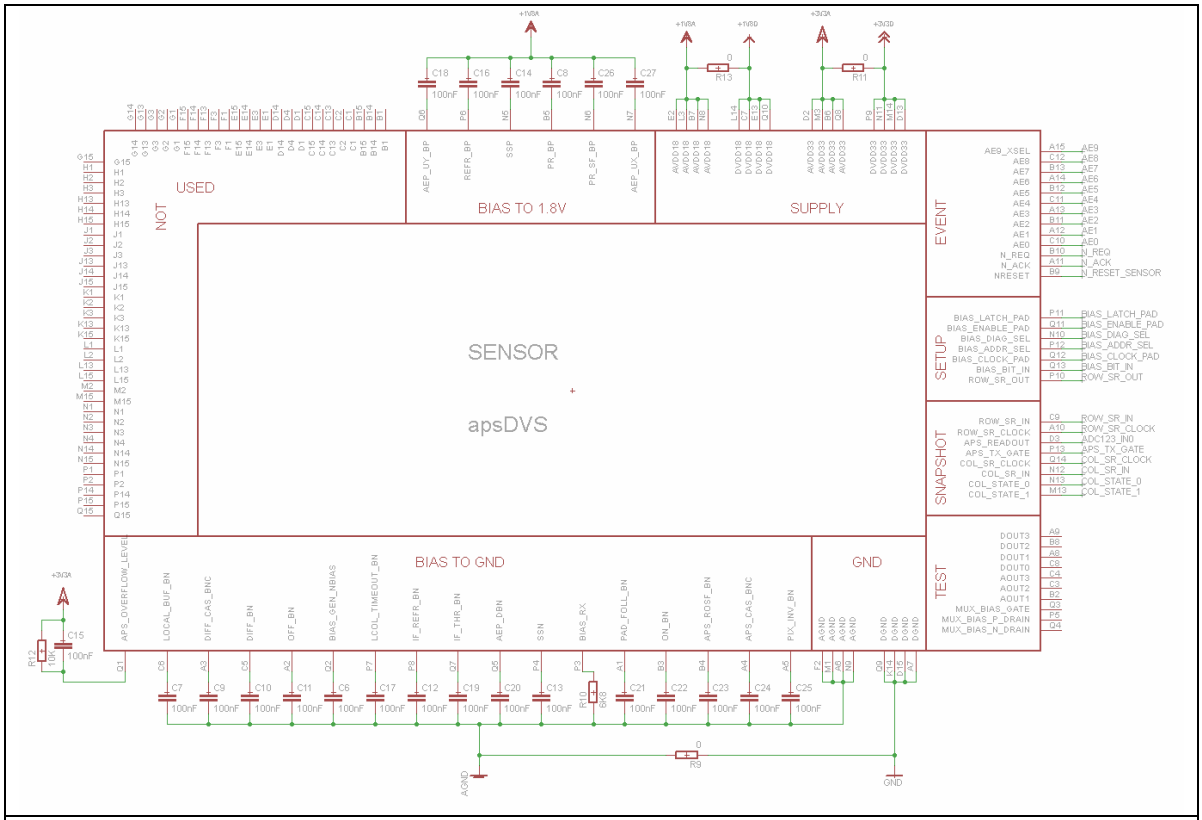

Figure 3.3: Schematics : Sensor part

The analogue and digital supply lines are split by a 0 Ohm resistor which can be later replaced with an inductor in case noise reduction is required.

Already knowing the connections to be made between the sensor and the microcontroller, the second part of the schematic involves simply picking from the controller's available pins while avoiding the ones needed by the FSMC. The peripherals required for proper microcontroller operation are added, together with two connectors for the serial and programming interfaces, as well as buttons for reset and boot mode and two LEDs for debug and signalling purposes.

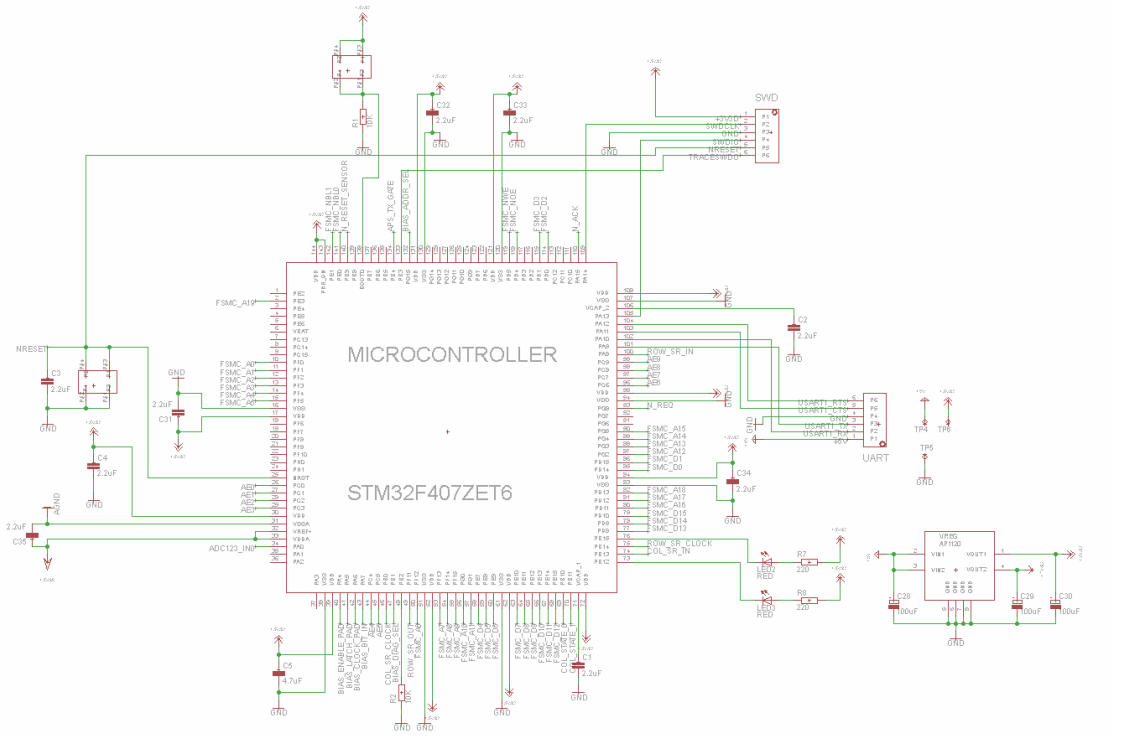

Figure 3.4: Schematic : Microcontroller part

The microcontroller is supplied by 3.3V and the board's main supply line comes from the serial interface and equals 5V. Because the DVS requires a 1.8V supply line in addition to the 3.3V, a dual output voltage regulator is included.

When connecting the SRAM a strategy is used to facilitate later PCB design: it makes no functional difference if the address signals are interchanged and this is also valid for the data signals. While this causes address and data bits shuffling when writing, the exact transformation is made backwards when reading.

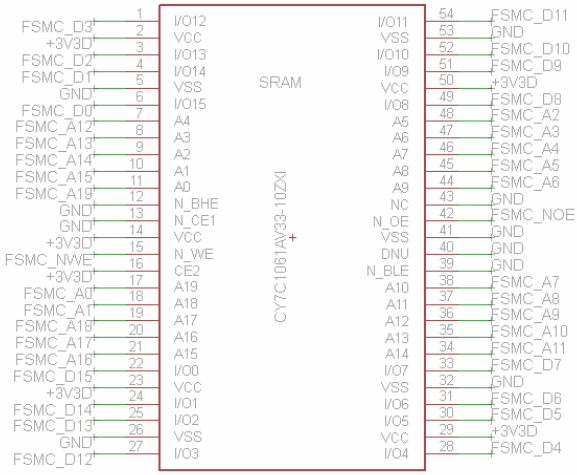

Figure 3.5: Schematic : SRAM

The first prototype of the system will not include the SRAM module, but once it has been tested and presents correct functionality, the SRAM will be added. This choice was made because the sheer number of connections that adding the module entails, would prolong the implementation of the first prototype.

3.3 Software Development

3.3.1 Concept

The main purpose of the software we had to develop was to initialize the sensor, control and fetch the event acquisition and send the events to a display application. In order to fulfill these requirements some basic software routines that allow the interface microcontroller to capture the events produced by the DVS and to process them. In our case the processing comes down to acquiring raw events from the sensor, apply timestamps for further sequencing and sending the events in an appropriate format to a PC in order to be displayed on the screen.

More precisely, the first critical thing our software does is sending the proper values to the DVS's internal Bias Current Generator. This is the initialization phase in which we can set the sensitivity level of the sensor. This Bias Generator sets then the DVS bias capacitor voltages to the proper levels which enables the DVS to work properly. Secondly, the program has to be able to detect the signals sent by the DVS when the intensity of the light arriving on a pixel changes (the generation of an event). These events have to be stored in the microcontroller's embedded memory and sent to a PC at a given rate.

Our main program consists of an infinite loop. Inside this loop some global variables are compared to some reference values and if the value of a variable matches the reference value, a block of instructions is conditionally executed or a function is called (for a better understanding, one can have look at the below). Moreover some inputs of the

microcontroller which are connected to the outside world (DVS and the PC) are overseen by dedicated peripherals of the microcontroller and generate interrupts.

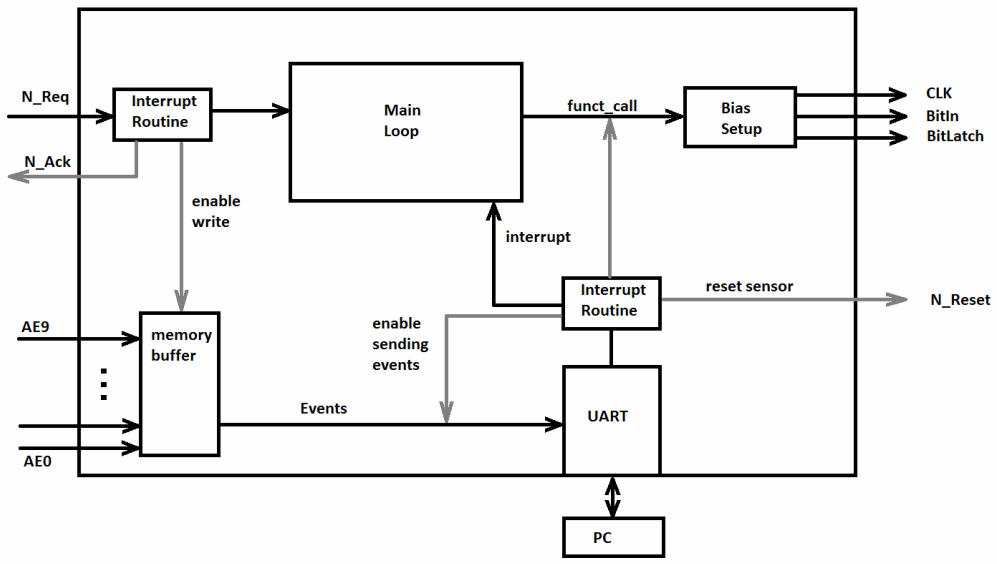

Figure 3.6: General overview of the software

There are two types of interrupts. The first type of interrupt is generated by the timer peripheral when the DVS requests to send an event to the microcontroller via the N_Request signal figure above. If such an interrupt occurs, the execution of the main loop is suspended and an interrupt routine is called (see again the fig). The interrupt routine reads the corresponding port of the microcontroller and gets this way the address of the pixel **which generated an event**. The address is saved in the internal memory on the microcontroller.

The other type of interrupt is generated by the USART peripheral of the microcontroller when the user sends a character to the microcontroller via the PC. In fact, the microcontroller is connected to a PC via an USB-FTDI-UART line what allows the user to give commands to the microcontroller by typing single characters on the keyboard. If such an interrupt occurs, the execution of the main lop is suspended and an interrupt routine is called which, if a proper character was sent by the user, changes the values of some global variables of the entire program. Theses variables, which are continuously assessed in the main lop, allow different blocks of instructions or independent functions to be conditionally executed. This way, the user can lunch the Bias routine which sends the bias values to the DVS or allow the microcontroller to send to the PC the events it stored in its memory.

For future developments using the DVS, additional memory will certainly be needed by the microcontroller to carry out more complex data processing (a possible practical application is to use the DVS for robot navigation purpose which will need more memory to store processed data). It was thus foreseen to include in our program a routine for the initialisation of an external memory block (SRAM) and make it ready to use by the microcontroller. Nevertheless at the date of the report writing, this last part

was not done. The same holds regarding the capture and the processing of the snapshots which was not done.

During the software development we had access to the software which runs on another microcontroller connected to an older version of the DVS. The software we developed is mainly based on the same procedure and it was very helpful to us.

3.3.2 Tools

For producing the software, we used the Red Suite 4 and Red Suite 5 Integrated Development Environments from Code Red Technologies.

The microcontroller used on the PCB is a STM32F407xx featuring an ARM Cortex M4 CPU. The testing of the software was performed using a STM32F4-Discovery development board (see figure below) featuring the same microcontroller as the one used on the PCB. Finally, the programming of the microcontroller was performed by a Red Probe+ debug probe from Red-Code-Technologies (se fig…).

The development of the software was considerably made easier first of all by the access to the software from another microcontroller (from the LPC210x family) which interacts with a DVS128, an older version of our DVS.

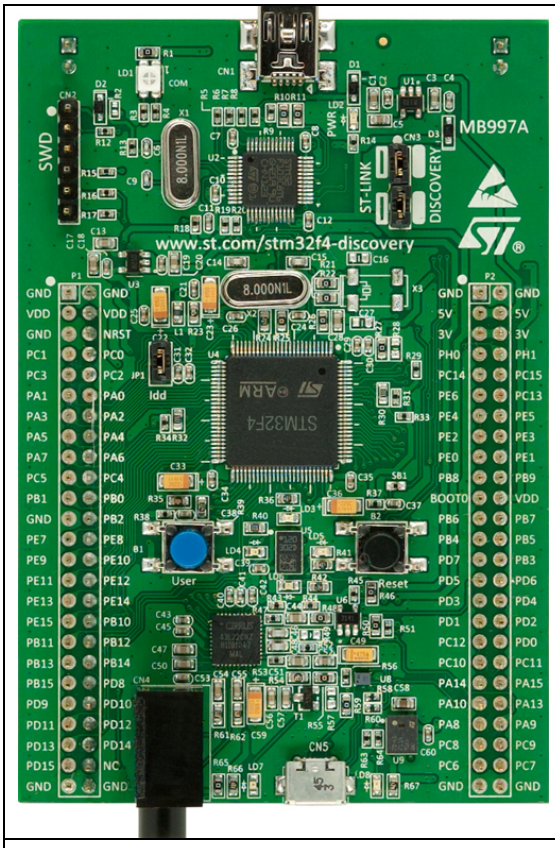

Figure 3.7: Dicovery development board

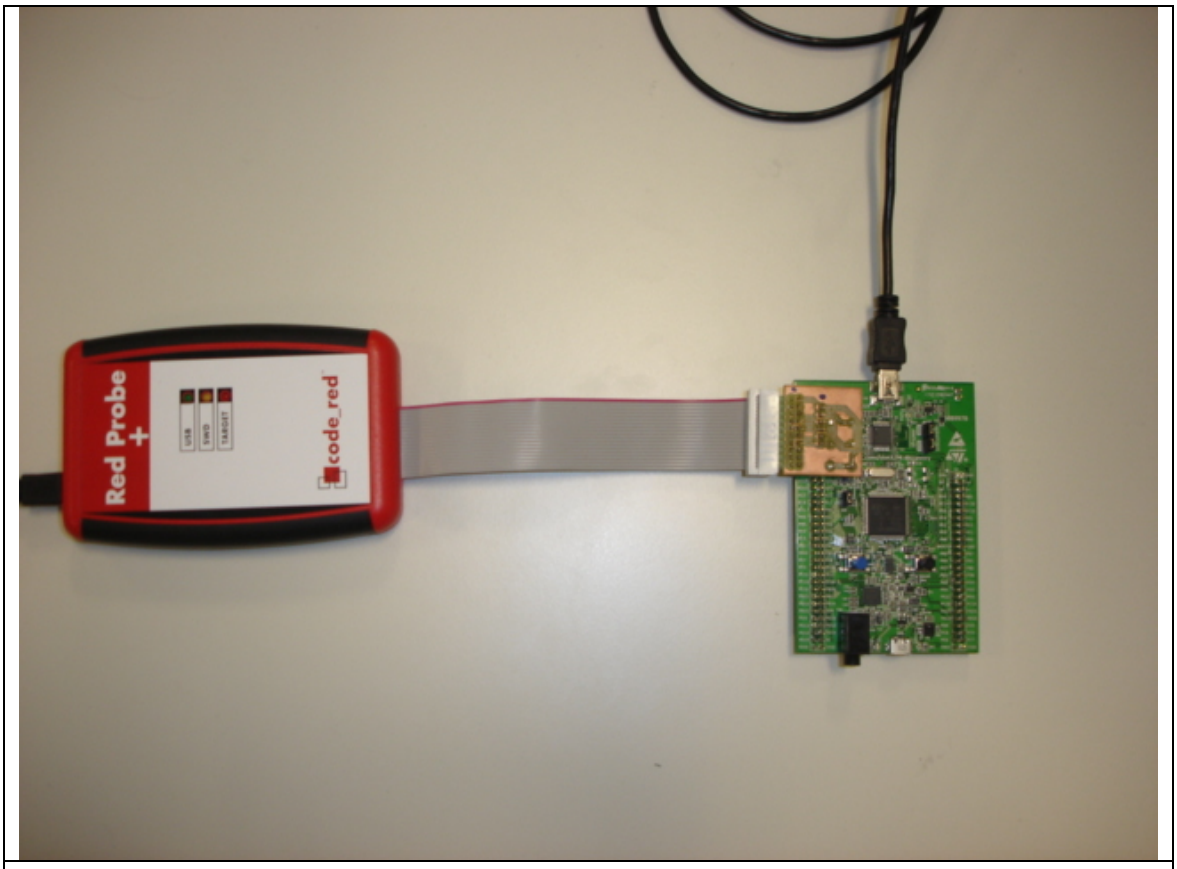

Figure 3.8: Discovery development board with a conected Red Probe +

4 Implementation

4.1 Hardware Implementation

4.1.1 PCB Layout

As mentioned before the first prototype will not include the external memory in order to simplify the routing process. The signals are routed by hand in Eagle, and, by setting the correct parameters in the DRC rules, errors due to the milling machine's limitations can be easily avoided. Vias represent a high cost resource as they have to be soldered in by hand. The PCB design does not include a ground plane to facilitate soldering pads.

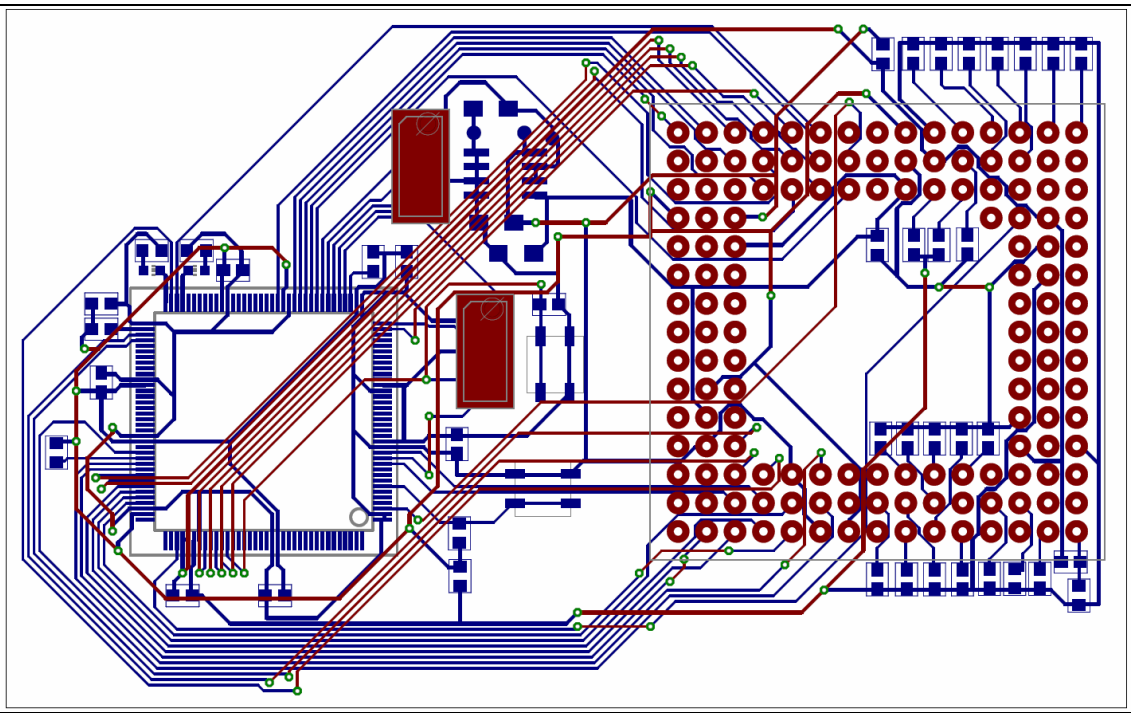

Figure 4.1: Overall schematic

In the above image, the microcontroller location can be seen on the left and it is placed on the bottom side of the board. The sensor connects through a socket which is placed on the right side of the board. The sensor pads are not connected from top to bottom layer. For this reason, the sensor pins are only routed from the bottom side of the board.

4.1.2 Assembly

Once the PCB layout is complete it is exported to the milling machine control software and the board can be manufactured. All components are added by hand including vias.

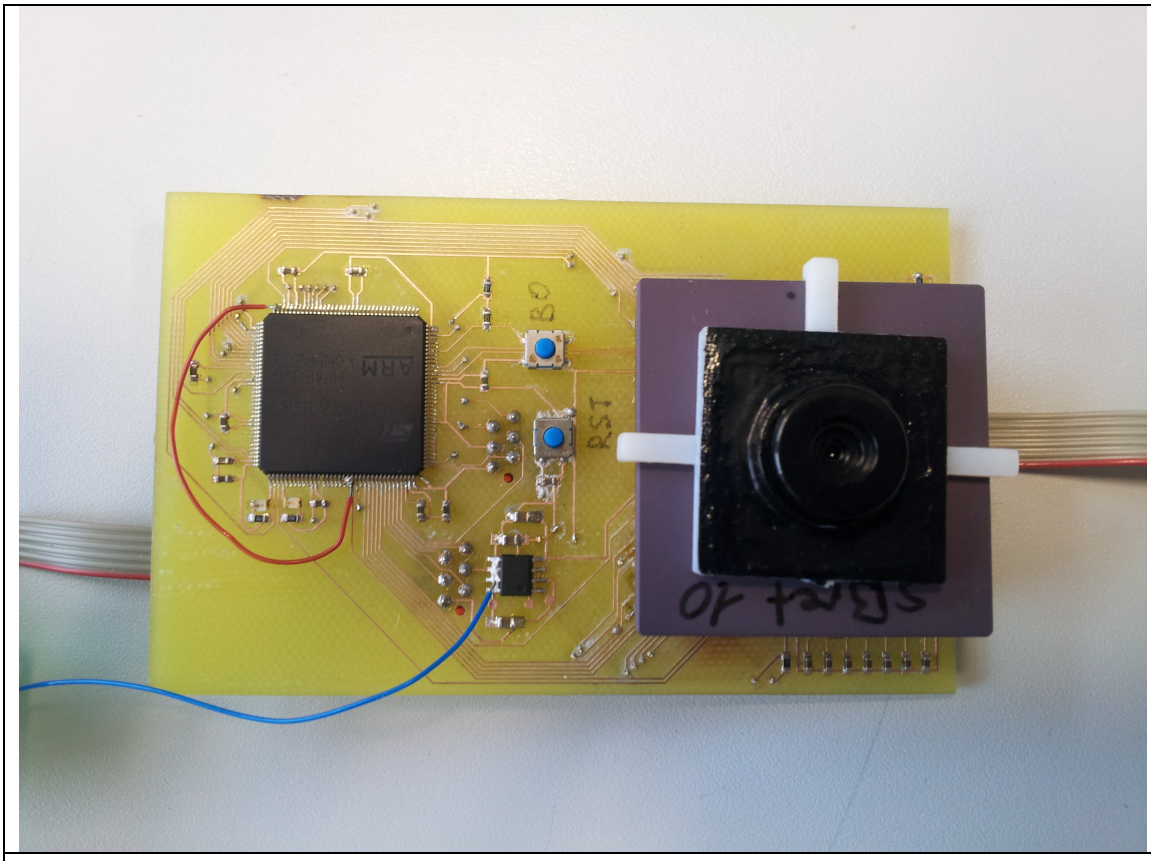

Figure 4.2: The buit system

Above is a picture of the assembled prototype with connected data cables for the programming tool as well as a serial-to-USB interface. The “green wire fix” (red in the picture) is due to a post assembly change in desired pin functionality on the microcontroller side.

4.2 Software Implementation

In this section we present our software routines in a more detailed way.

4.2.1 Main Loop

The program running on the microcontroller consists of a continuously executed loop (see figure below).

As we can see on the figure below during the execution of the loop, some variables are assessed and depending on their values, additional instruction blocks or functions are executed. There are three such variables: ***enable_bias***, ***enable_event_sending*** and ***new_event_arrived***. These variables are modified by the interrupt routines.

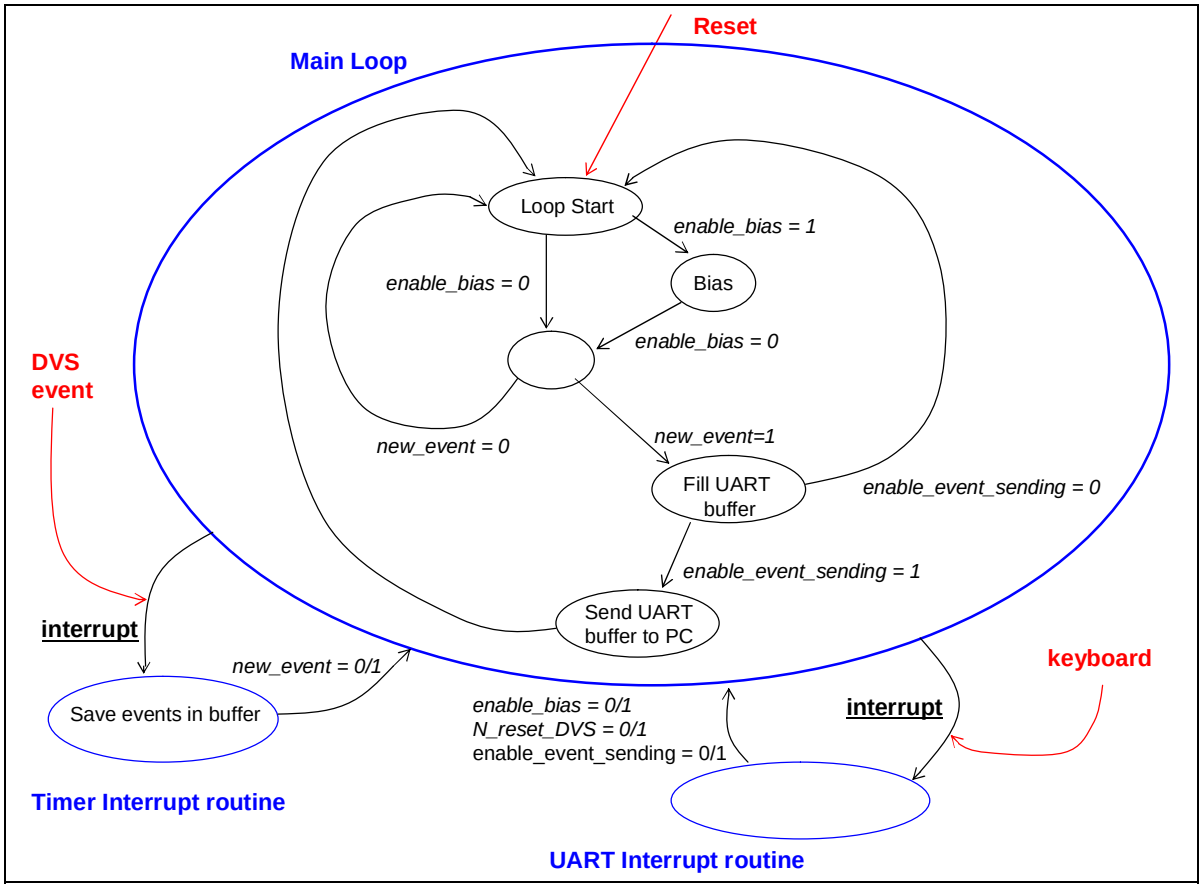

Figure 4.3: Diagram of the software

4.2.1.1 Bias

The bias function sends to the DVS a sequence of digital values. Inside the DVS, the Bias Current Generator receives these values and generates the corresponding analogue currents which will then be used to configure the bias capacitors inside the DVS.

Basically, three signals are required to send digital bias values to the DVS (see General overview of the software in the concept part). The digital values are sent as sequences of bits to the DVS through the **BitIn** line. A clocking signal **Clk** is also required: the logical level detected by the Bias Current Generator corresponds to the logical level of the **BitIn** line at the rising edge of the clocking signal. Finally, a latching signal **Latch** is needed. In fact, there are 22 bias values to load into the DVS in order to make it function properly. In order to specify to the Bias Current Generator that a particular bias value

has been completely loaded, the latching signal has to be set to low for a certain time (see figure below). The latching signal is used to validate the transfer of each bias value.

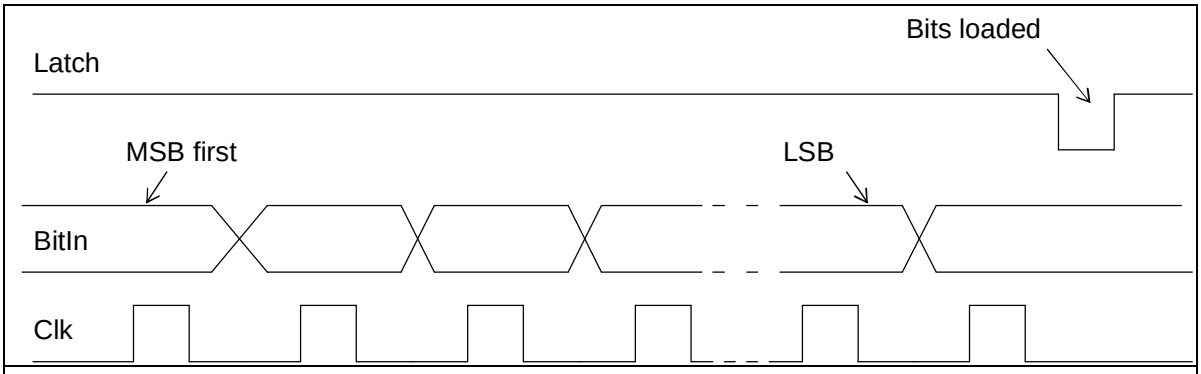

Figure 4.4: Bias signals

The bias procedure is slightly more complicated because each bias value has to be sent to a specific address. That means that for sending a specific bias value to the Bias Current Generator, the Bias function first sends the address which corresponds to the bias value and then the value.

4.2.1.2 The influence of the user

For debug purposes, the user is able to influence the execution of the main loop. In fact, the user can modify the values of the variables: ***enable_bias***, and ***enable_event_sending*** by typing specific characters on the keyboard. Once the PCB is powered on, the user can lunch the bias routine by typing **B** followed by **'return'**. This sets the ***enable_bias*** variable to 1. The Bias function is then executed only once since at the end of this function, enable***_bias*** is reset to 0.

Once the biases were set and the DVS is ready to use, typing **S 'return'** enables the DVS to send events to the microcontroller when an event occurs. The microcontroller saves then the received events in its memory.

Finally, **+ 'return'**, sets the ***enable_event_sending*** variable and the microcontroller starts to send the events from its memory to the PC. The instruction – 'return' sets ***enable_event_sending*** to 0.

The UART generates an interrupt as soon as a character is typed on the keyboard. The interrupt calls a specific interrupt routine which modifies the values of the above variables.

4.2.1.3 Events Capture and Processing

The execution of the main loop is interrupted when an event comes from the DVS. This interrupt is generated by the timer peripheral. In fact the control of the event flow from the DVS to the microcontroller is managed by two low active signals: **N_Request** and **N_Acknowledge** (see figure General overview of the software in the Concept part). **N_Request** is used by the DVS to inform the microcontroller that an event has been detected and is ready to be sent, **N_Request** is connected to a timer peripheral of the microcontroller which generates an interrupt when the level of

N_Request changes. **N_Acknowledge** is used by the microcontroller to inform the DVS it is ready to receive events. If the DVS has no events to send to the microcontroller, the signal **N_Request** is set to high and if the microcontroller is ready to receive events, **N_Acknowledge** is set to high.

When an event occurs, the DVS sets the x or y address of the pixel affected by the change in intensity on a parallel bus :

$(AE_0, AE_1, AE_2, AE_3, AE_4, AE_5, AE_6, AE_7, AE_8, AE_9)$

and sets **N_Request** to low. The timer detects the change of the **N_Request** signal, saves at which time the event occurred and lunches a specific interrupt routine which sets **N_Acknowledge** to low (preventing this way the DVS from sending new data) and reads the address of the pixel affected by the change in intensity. Once the address of the pixel was saved, the interrupt routine sets **N_Acknowledge** to high and the DVS can send new addresses.

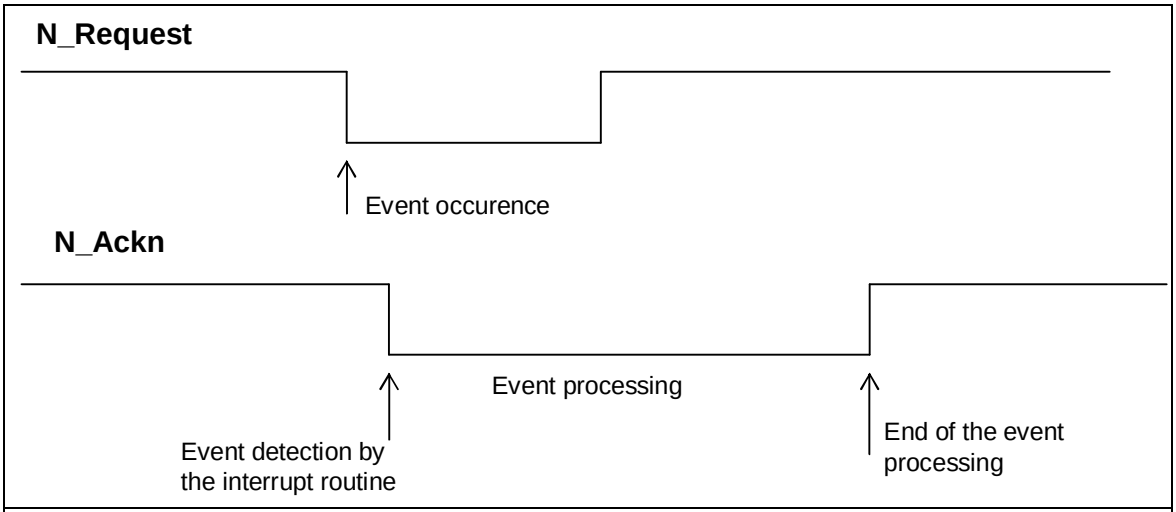

Figure 4.5: The behaviour of the N_Request and N_Acknowledge signals

The interrupt routine reads the address sent by the DVS and stores it in a buffer. The addresses are sent by the DVS in a particular way. The x and y addresses are not sent in the same time : the DVS sends an x address or an y address. If at some point in time, an y address is sent, the next addresses are x addresses corresponding to the pixels with the same initial value of the y address. In as much as an event is defined by an x address, an y address and the corresponding polarity (the intensity raises or lessens), once an y address is received, the interrupt routine saves it in a dedicated buffer and waits for the corresponding x addresses. When an x address is detected, the interrupt routine combines the previously stored y address with the received x address and stores the whole event in the microcontroller's event buffer. The event buffer is continuously read by the main loop and if a new event was put in, it is loaded in the UART buffer and transmitted (if the transmission is enabled) to the PC (see the main loop diagram).

```
Timer Interrupt routine
{
        Set Ackn to 0;

        address = read (AE….AE9);

        if (address is y)
        {
                save (address) in Y_ADDR;
        }

        if (address is x)
        {
                y = read (Y_ADDR);
                save (address, y) in Event_Buffer;
        }

        Set Ackn to 1;

}
```

Figure 4.6: Timer Interrupt Routine

Finally, the structure of the addresses sent by the DVS is presented below.

For an x address: the AE_0 signal gives the polarity of the event, the $AE_1,....,AE_8$ signals give the address and $AE_9 = 1$.

For an y address: the $AE_0,....,AE_7$ signals give the address and $AE_9 = 0$.

5 System Verification and Tests

The verification of the PCB began after its realisation and before the integration of the DVS. During the integration of the DVS on the PCB some problems arose as it sometimes happens when dealing with devices not yet under industrial production. One of these problems required to replace the DVS's socket on the opposite side of the PCB, in contrast to our initial plans.

As the DVS is not an industrial product, its robustness is not properly specified in data sheets. Thus, the integration of the DVS on the PCB had to be done very carefully in order not to damage it by an inappropriate signal coming from a well engineered microcontroller.

The testing of the software was also painful but it has to be acknowledged that in that case the problem came more from the bugs in our software than from the uncertainty regarding some properties of the DVS.

It can be stated that the most relevant tests which would decide whether the project achieved its goals are on the one hand the test of the ability of the system to provide expected data which are then displayed on a PC and on the other hand the data throughput should not be below the data throughput of the previous system.

Regarding the overall functionality of the system, it can be stated that the produced system achieves the task of providing correct data to the PC (see figure below).

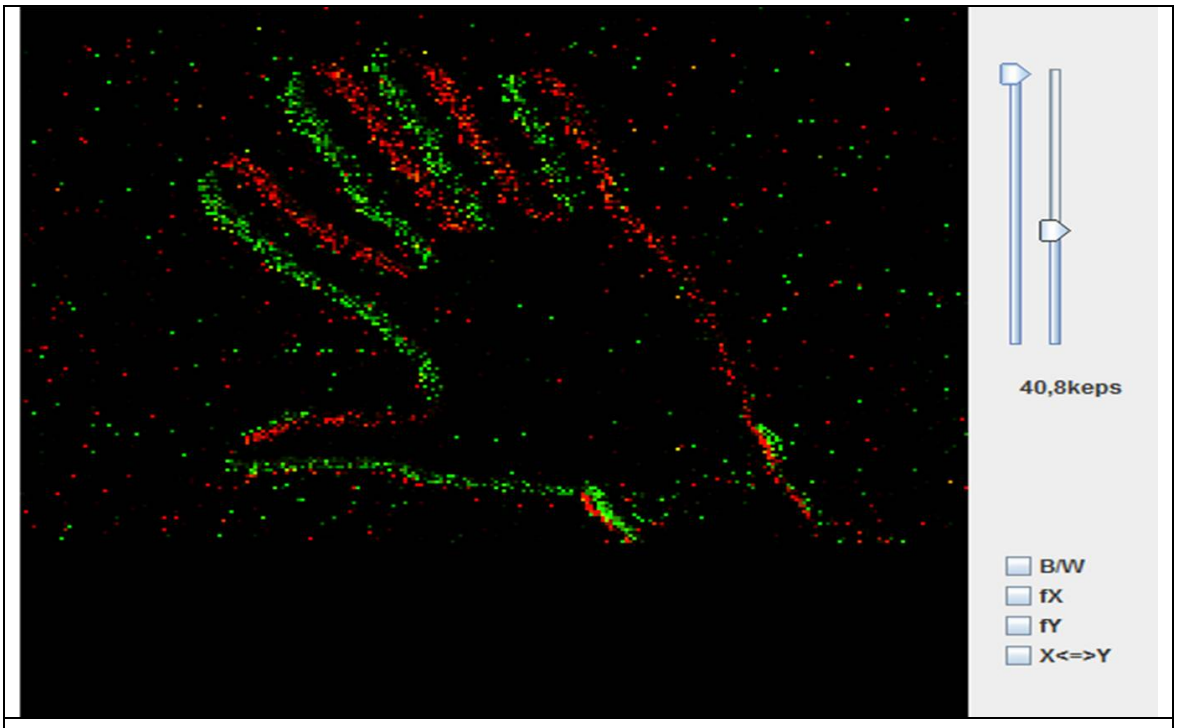

Figure 5.1: Image produced by our system

Regarding the data throughput and the noise robustness, improvements have to be carried out. Unfortunately it was not possible for us to quantify the noise robustness. The measurements of the voltages at some points on the PCB reveal the presence of an undesirable level of noise.

Regarding the data throughput, the system behaves worse than the previous system. The data throughput is defined in our case as the number of events the system can send to a PC. The maximum data throughput is about $40 \cdot 10^3$ events per second which is at least half the value of the previous versions. Suitable data throughputs are around $100 \cdot 10^3$ events per second.

6 Conclusion

6.1 Achievements

At the time of writing the present document many of the goals of the project have been achieved. On the hardware level, the prototype stands as proof of the correctness of the design concept as far as interfacing the sensor with the microcontroller goes. A new version still has to be built to include the external memory module, however, its connection to the MCU is very straight forward, thus no problems are anticipated.

On the software level the necessary basic routines were implemented and tested. However, the data throughput of the system doesn't achieve the expected level and therefore some optimisations of the software have to be done. Moreover, the software block for the handling of static images has not been implemented although it was specified in our goals at the beginning. This is due to the slow development of the main software routines.

6.2 Outlook

The design, while usable for its intended purposes, can be further optimised. The PCB layout can be minimized especially given that future versions of the DVS320 will be smaller in size, and additional software components may be added. With respect to the latter, the additional memory may facilitate implementation of more complex computational algorithms such that the host device connected to the sensor board may be freed from part of the processing that practical usages of the eDVS will entail.

List of Figures

Bibliography

[1] https://wiki.lsr.ei.tum.de/nst/index
[2] http://www.stm32f0.com.cn/pdfs/user_manual/02_STM32F0%20reference%20manual.pdf
[3] http://www.st.com/internet/mcu/product/252148.jsp
[4] Conradt, J., Tevatia, G., Vijayakumar, S., & Schaal, S. (2000). On-line Learning for Humanoid Robot Systems, International Conference on Machine Learning (ICML2000), Stanford, USA.
[5] Denk C., Llobet-Blandino F., Galluppi F., Plana LA., Furber S., and Conradt, J. (2013) Real-Time Interface Board for Closed-Loop Robotic Tasks on the SpiNNaker Neural Computing System, International Conf. on Artificial Neural Networks (ICANN), preprint.
[6] Conradt, J., Simon, P., Pescatore, M., and Verschure, PFMJ. (2002). Saliency Maps Operating on Stereo Images Detect Landmarks and their Distance, Int. Conference on Artificial Neural Networks (ICANN2002), p. 795-800, Madrid, Spain.