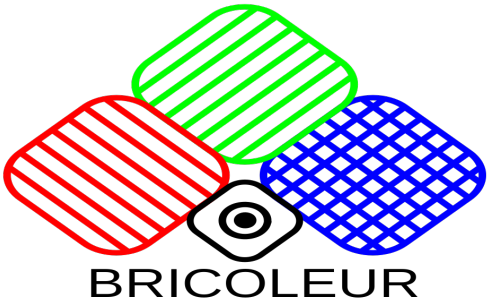
NTNU
Norwegian University of
Science and Technology

TDT4295 - Computer Design Project

User Manual

Collision Detection Group:
Eirik Vale Aase
Richard Bachmann
Einar Hov
Marius Kohmann
Joakim Lier
Mathias Lundteigen Mohus
Fritz-Olav Myrvang
Valentin Plotkin
Eirik Smithsen
Ola Toft



November 21, 2018
`https://github.com/NTNU-TDT4295/BRICOLEUR/`

Contents

1 Acknowledgements

Odd Rune Lykkebø for teaching us the ways of hardware.
Omega Verksted for providing the optimal work environment and lending their expertise.

2 Introduction

This is the collision detection group's final submission for the course TDT4295 Computer Design Project 2018. This course aims to give students an introduction to the process of designing physical computer solutions, from start to finish. The project was completed on a budget of 10000 NOK, of which 5400 NOK were spent.

2.1 What is BRICOLEUR?

A bricoleur is someone who starts building something with no clear plan, adding bits here and there, cobbling together a whole while flying by the seat of their pants.

The overarching goal of the project was to develop a stationary system which is able to detect moving objects, estimate their velocity and direction of motion, and finally predict a potential collision. In order to create concrete requirements for the minimum viable product, this was further reduced to the system being able to "predict collision with a single high-contrast object of reasonable size, moving in a straight line, and going no faster than 20 $^{m}/_{s}$".

2.1.1 Additional requirements

In addition to solving the problem described above, the project was to follow these constraints:

- The FPGA on the PYNQ must be used to perform meaningful acceleration of calculations.
- The system must make use of a custom printed circuit board, which houses a micro control unit that is actively involved in the problem's solution.

2.2 Basic outline of solution

The solution makes use of one camera and multiple ultrasonic sensors to detect incoming objects. The main code runs in Python on the PYNQ CPU, and is receiving images from the accelerator implemented on the FPGA. When the CPU has detected an incoming object it will send a signal to the microprocessor on the PCB that something is incoming. The ultrasonic sensors report directly to the microprocessor. The microprocessor will then output a byte value on UART, which indicates how sure it is that the object will collide both from the data from the CPU and from the ultrasonic sensors.

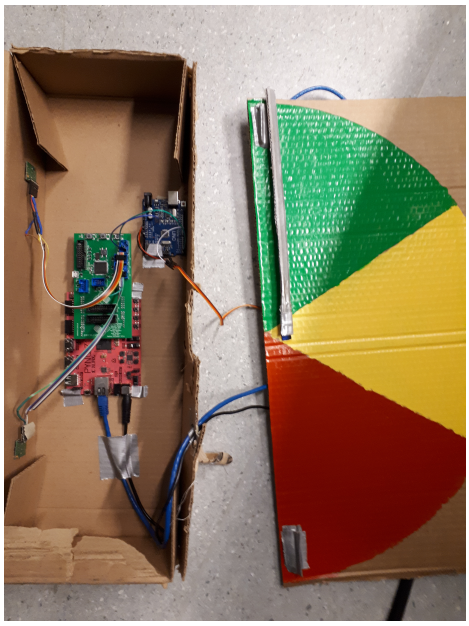


Figure 1: The contents of the final deliverable

3 Setup - Step by step

Building the project from nothing can be a quite involved process. Reserve plenty of time and prepare a cup of tea.

3.1 Getting started

To get started with the PYNQ one must first set it up properly and hook it up to a computer. Start by following this guide:
`https://pynq.readthedocs.io/en/latest/getting_started.html`

3.2 Installing dependencies

If the latest PYNQ image (8) is used, all of the dependencies used in the project come pre-installed.

However, if for some reason OpenCV for Python or Numpy are unavailable, it is highly recommended that you hook the PYNQ up to an Internet connection in order to download the dependencies easily.

3.3 Development environment

On your development computer you will need to install Vivado 2018.2 to continue development. This can be downloaded from here: `https://www.xilinx.com/`

`support/download.html`. And good luck.

3.4 Scala and Chisel 3 Modules

The accelerator modules are written using Chisel 3. You will first need to set up Chisel 3 and Scala. Follow the guide in the Chisel repository: `https://github.com/freechipsproject/chisel3`.

To be able to add your Chisel module into the Vivado project you need to compile the Chisel code into Verilog. The Verilog files will be imported into the Vivado project when it is created.

3.5 Loading project into Vivado

In the BRICOLEUR GitHub repository you will find a `pipeline.tcl` file that you will need to import our modules into the Vivado project. The rough outline below will demonstrate how this is set up:

- Start Vivado 2018.2
- Create a new RTL project
- During the wizard it will ask for a set of source files, which are the Verilog files compiled in 3.4.
- After creating the project, go into the Tools menu and click `Run TCL Script` and locate the aforementioned `pipeline.tcl` file.
- Make sure to set the correct project constraints compatible with the PYNQ.

3.6 Exporting Vivado project

When you have added your new module to the project and connected your module in the block design you can generate the bitstream. Press `Generate bitstream` and wait.

When the bitstream is created you are ready to export it for execution on the PYNQ. Click on `File` and then `Export bitstream` and save it somewhere. You will now have to transfer this file onto the PYNQ to be utilized within the Python environment.

3.7 Run on the PYNQ

Run the python script on the PYNQ system and make sure to import your newly updated bitstream.

To upload to the PYNQ, run these commands:

```
scp <pathToBitstream>/design.bit xilinx@192.168.2.99:/home/xilinx/
scp <pathToTclScript>/design.tcl xilinx@192.168.2.99:/home/xilinx/
```

3.8 Preparing the BRICOLEUR PCB

3.8.1 Programming the MCU

You will need:

- 1 USB-UART device + USB cable
- Female-female jumper cables
- A way to supply power to the board (USB-micro cable/PYNQ board)

The EFM32GG980 can be programmed using the bootloader which comes pre-installed on the chip. Compile the MCU code to produce the desired `.bin` file. Depending on your method of development, make sure that the loader sets a base offset of 0x4000, to respect the bootloader's space. Also reduce the available flash memory accordingly. Then prepare the PCB for flashing.

1. Connect the USB-UART device to your computer, make sure it works.
2. Connect the UART device to the UART_BOOT port, between the MCU and the logo, on the PCB. Pin 6 should be connected to RX, and pin 4 to TX.
3. Using a jumper, connect pin 9 on the PCB debug connector (top left corner) to pin 1 (VMCU). This enables launch to bootloader instead of an existing program.
4. Connect a power source to the PCB.
5. Launch your serial communication program of choice, such as Screen, MiniCom or Putty.
6. Push the reset button, located closest to the MCU.
7. Quickly send the character 'U' to initialize bootloader communication.
8. Then send 'u' to perform a non-destructive upload.
9. Using Xmodem-CRC, transfer the binary. In 'screen' this would be done by pressing Ctrl-a, followed by ':', and the command:

```
exec !! sx bricoleur.bin
```

10. Verify that the program runs as expected by sending 'b'.

3.9 Putting everything together

Assemble the boards as shown on the image. The BRICOLEUR system can be configured in multiple ways. However, in order to reproduce the demo-build, connect it as follows:

1. Connect U1 to the left ultrasonic sensor, and U4 to the right one.
2. Connect the top right IDC (J4) to an external unit.
3. ~~Connect the OV5642 camera to U5.~~ Connect the ethernet cable to the PYNQ instead.
4. Connect a jumper or physical switch to the desired power source pin, in this case the PYNQ, and the VMCU pin (bottom left on switch area).
5. Put the PCB on top of the PYNQ by joining the matching headers.

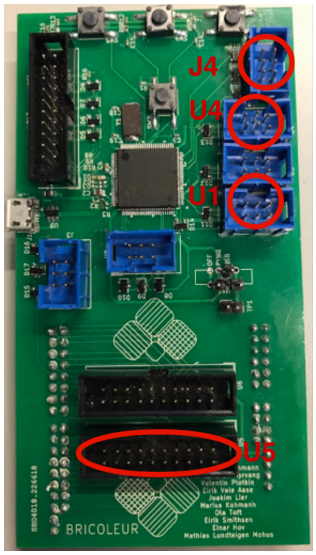


Figure 2: Connections for basic functionality

3.9.1 Sensor placement

The positioning of the ultrasonic sensors is based on two main variables: Area of overlap and distance between the sensors. Both should be maximised to achieve the best results.

A larger area of overlap means there is a larger area where both sensors will detect the object, and therefore be able to do meaningful calculations. A larger distance `D` between the sensors gives a larger difference in the distance from each sensor to object `O`. This in turn makes it easier to calculate an accurate X-coordinate of the object, but it also lowers the area of overlap.

The demo build chose a distance of 30.5 cm.

Figure 3: Things to consider when placing sensors

4 Usage

Once the system is assembled, place it on a flat surface and connect the power supply for the PYNQ. Optionally connect a visualization unit. Turn on the power supply for the PCB. The MCU will then calibrate the ultrasonic sensors, and begin normal operation.

4.1 Video force feeding

In the event that the project's camera should fail,
it is possible to send externally recorded video to the PYNQ. For instance, a pre-recorded video, or even the feed from a laptop webcam, may be used for this purpose. The Python script performing the tracking algorithm reads raw frames from stdin, so the data can be piped in from f.ex. a file or netcat. The following shell commands were used to successfully pipe a laptop's webcam through the algorithm on the PYNQ.

```
# On the PYNQ
sudo sh -c 'nc -l -p 420 | ./video-mp.py'
# On the laptop
ffmpeg -i /dev/video0 -f rawvideo -s 320x240 -pix_fmt gray8 - \
    | nc xilinx@192.168.2.99 420
```

4.2 Observations

The ultrasonic sensors read distances correctly, however the employed algorithm proved to be somewhat unstable. The camera worked better, but still struggled with larger distances to the target object. Sadly, the delay between the ultrasonic and camera readings made it difficult to combine their results without sacrificing latency.

4.3 Output

Bricoleur output can be read from the AUX-port. By default it will be transmitted through UART with a baudrate of 115200. The output is a byte value, ranging from 0 to 255. Here 0 indicates no sensed danger of collision, and 255 indicates certain collision. This output is weighted combination of the PYNQ's output and the conclusion of the ultrasonic analysis.

The PYNQ will also output video frames—annotated with bounding boxes of tracked objects—to a file. The file can either be analyzed afterwards, or used for real-time monitoring of the system. During testing, we have played back the annotated video real-time on a connected laptop by using the following shell commands before starting the Python script.

```
# On the laptop
```

```
nc -l -p 4200 | ffplay  -f rawvideo -pixel_format gray8 \
    -framerate 144 -video_size 318x238 -
# On the PYNQ
mkfifo debugframes.raw
nc 192.168.2.1  4200 < debugframes.raw
```

4.3.1 Visualisation suggestion

For the demonstration an external system was used to turn the output byte value into something more tangible. This device consisted of an Arduino Uno, which read the transmitted serial data, and a servo, which pointed a needle to indicate level of panic.

5 System description

5.1 The sensors

During the project's early stages, an evaluation of potential sensors was performed. It was then concluded that a combination of up to two cameras in addition to a set of ultrasonic sensors would be the most reliable tools to complete the task. The exact number of each sensor was determined at a later time, when the group had a clearer concept of the FPGA's strengths and limitations. In the end it was settled that the final system should use a single camera and two ultrasonic sensors.

5.1.1 CMOS OV5642 Camera Module

The camera sensor is the system's primary sensor. The purpose of the camera is to deliver images in 320x240 resolution to the accelerator so that it can pre-process the images before the CPU calculates on them. For this task the OV5642 was selected because of accessible hardware interface, comparatively low cost, and support for high frame rates. After starting to work with the camera, it became clear that the documentation for the camera module which was accessible was insufficient for configuring it to our purpose. Configuring registers over I^2C and reading raw sensor data over the DVP bus works, as does cropping the output. However, attempts to change color format or scaling the output image failed. In the end, the camera module was abandoned, and video files and laptop cameras were used as a workaround.

5.1.2 Maxbotix Ultrasonic Rangefinder

We use the ultrasonic sensors Maxbotix Ultrasonic Rangefinders, model LV-MaxSonar-EZ0. Their purposes is to determine the direction of a moving object and send the result to the microprocessor on the PCB. Although their data are intended to supplement information derived from the camera, the ultrasonic sensors may also function as a backup, should the camera fail.

5.2 Accelerator

The custom accelerator is designed to process images from the camera and give the images to the CPU to ease the work for the CPU. The reason for this accelerator being needed is that the algorithm which is used for detection of moving objects needs the images to be Gaussian blurred before it is able to determine if there is an moving object. Because this process is a time consuming task for the CPU it needs to be blazingly fast, so that the CPU can process frames on a high frame rate. The overview of the accelerator in the final design is shown in Figure 4. The intended design is shown in Figure 5.

The Gaussian Blur accelerator is modelled after [1] and looks like figure [4].

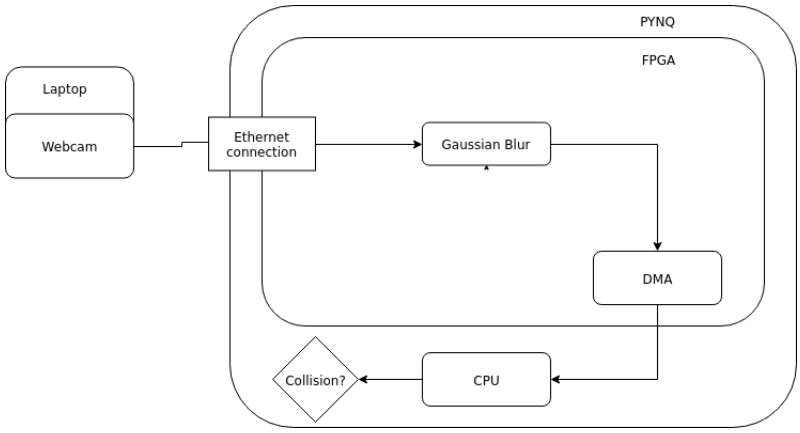


Figure 4: Final FPGA design overview

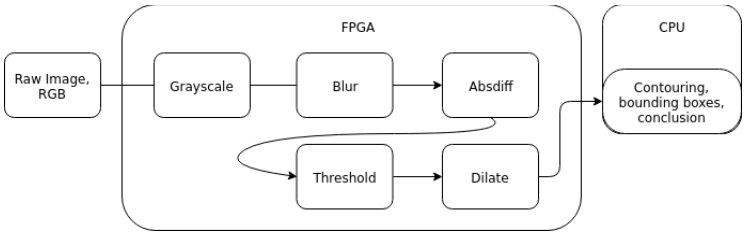


Figure 5: Intended accelerator overview

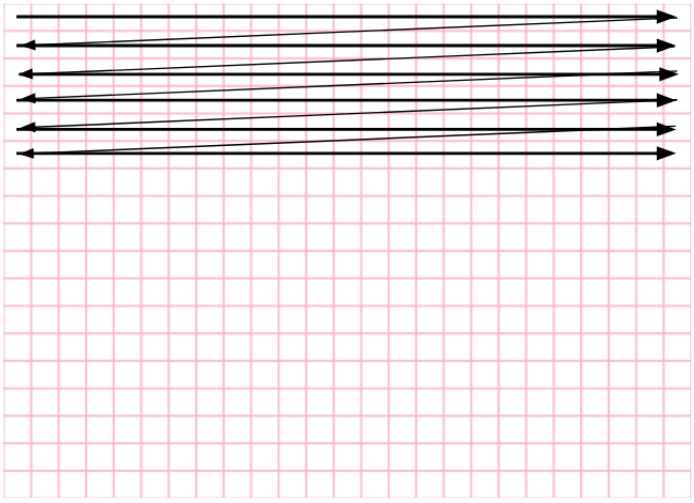


Figure 6: Real pipeline

5.3 Accelerator pipeline

The accelerator's pipeline follows a sequential pattern [6], meaning the pixels are inputted from left to right, top to bottom. The reason for this is avoiding unnecessary complexity, since the intended camera model outputs the pixel-values in this fashion. Another reason is that this method is able to get the throughput needed, it has the least amount of latency, and uses the least amount of resources.

The alternative method to a sequential pipeline would be a way to partly load pixel-values vertically, instead of horizontally. Figure 7 shows this pattern. This, however, makes no sense without hardware support, which the camera does not have. If the camera would be able to output pixels in chunks equal to the Gaussian Blur kernel, it would make sense to use this, to lower the latency from the Gaussian Blur filter.

5.3.1 Video interface

The video interface receives signals from the camera module and produces a stream of pixels. The communication between the camera and the video interface is clocked by the pixel clock from the camera. A vertical synchronization signal signifies the end of a frame and the start of a new frame. The href signal signifies that the 8-bit data bus is currently holding valid pixel data. Using these signals and the data bus, the video interface reads image frames from the camera module.

As the camera module was abandoned due to difficulties configuring it, the video interface is not used in the final product.

Figure 7: Alternative pipeline

5.3.2 Grayscale

The grayscale component is the first step of the pipeline after pixel values are sent from the data source. It grayscales an image with the following equation:

$$o = r \times 0.3 + g \times 0.59 + b \times 0.11 \tag{1}$$

where r, g and b are the respective RGB channel values and o is the output, namely the resulting grayscaled value.

It incrementally produces this sum, outputting the grayscaled value when the blue value arrives, then resets and starts over again with the next pixel. This ensures that the grayscale only has 3 clock-cycles of latency, and a throughput of one third of the input, meaning no data is lost.

This module is not used in the final product, as converting the image to grayscale before piping the data to the PYNQ saved two thirds of the bandwidth used over the network interface.

5.3.3 Gaussian blur

The Gaussian blur part of the accelerator takes grayscaled values as input and blurs them using a computational kernel. Because Gaussian blur needs to calculate the resulting pixel value from pixels directly adjacent to it, it is necessary to store the pixels for later use. This is done by creating a series of queues, as outlined in Figure 13, and outputting the correct values through the filter kernel.

5.3.4 Absdiff

Absdiff is utilized to find the absolute difference in value between the same pixel from consecutive images. The module takes grayscale-values as input, and outputs the absolute difference.

It does this by inserting all values of the current frame into a queue. The next input value will then be the first value of the next frame, and the absolute difference between them is output by the module.

Due to unresolved bugs, this module was not incorporated in the final design.

5.3.5 Threshold

Threshold is used to filter out any noise from the Absdiff by creating a binary image. Small values from Absdiff are more likely to come from random noise than from an actual object moving in the picture.

Threshold is implemented by checking the input values, and output a value of 255 if it exceeds the threshold, and a value of 0 if it does not exceed it.

This module was also not used in the final design, as the module directly before it was excluded from the design.

5.3.6 Dilate

Dilation is an image filtering operation where in a given area on the image the maximum value is chosen and replaces the center value in the chosen area. This module was implemented, tested and was proven functional, however including it in the final design was deemed impractical, due to its position in the pipeline.

5.3.7 DMA

When the Gaussian blur component is finished doing its work, the pixel values are passed in to the DMA for fast transfer into DDR memory and direct access by the CPU. The DMA module is an IP block provided by the Vivado IP library.

5.4 The PYNQ CPU

The Cortex-A9 CPU is used on the PYNQ to further process the images from the FPGA. The CPU runs Python code that reads images from memory and calculates whether there is an incoming object or not. The images which are read from memory have been pre-processed by the custom accelerator for the CPU to work with. The images are already grayscaled and Gaussian blurred to make it easier to detect movement. The algorithm for the detection of incoming objects works as follows:

1. Two pre-processed images from the camera are compared to each other to find out where something has been moving

2. The size of the area that has been changed is calculated and stored.

3. When a new image is received in the next frame, the processor can determine if the new area that is changed is bigger or smaller than the previously stored value. If the area is bigger something is incoming.

OpenCV is utilized to perform these calculations. When an incoming object is detected, a signal is sent to the MCU through a 3-bit data bus to the PCB. These lines transmit a value ranging from 0 to 7 indicating the estimated probability of a collision.

5.5 The Microprocessor

5.5.1 Specification

The EFM32GG980F1024-QFP100 was selected to be the PCB's MCU because of its flexibility and feature-richness. This made it possible to progress quickly on other parts of the design, before the exact details of its operation were decided upon. The chips most important qualities for this project were:

- Comparatively high processing power
- Support for a number of protocols such as UART, USART and USB
- Familiarity to group members
- QFP100 14x14mm package (simpler to solder than alternatives)

5.5.2 Device Communication

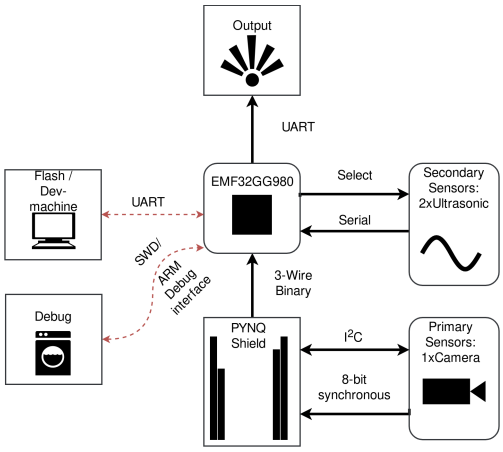


Figure 8: Communication on the BRICOLEUR PCB

5.5.3 Ultrasonic analysis

Each of the ultrasonic sensors will detect a different distance to an object. This difference can be used to calculate the position of the object relative to the sensors with a technique known as trilateration:

- Let each sensor be the center of a circle, and let the distance to the object be the radius of the circle. Then the position of the object may be found by calculating the intersection of the circles. This is illustrated in Figure 9.
- Please note that for two circles there will be two solutions, but one of these solutions will be behind the sensors, and can therefore be ignored.
- Also note that two circles will only give a 2D position, but this is sufficient to meet the assumption of an object moving along the ground. If the system should be expanded to handle objects thrown through the air, there would be a need for three sensors in order to find the 3D intersection point of three spheres. This would have four solutions, but only one of the solutions would be both in front of the sensors and above the ground.
- Once the position of the object at a given time is found, it is possible to measure the change in position over time. These positions can then be used to estimate velocity and interpolate a line, which in turn can used to estimate the future position of the object. This is illustrated in Figure 10.
- By looking at where the object will intersect the x-axis (the system is at $y = 0$), one can conclude whether the object will collide heads-on, hit the edge of our system, or misses it completely. This means that even if an object is moving toward the sensors, it is possible to detect that it will actually miss the system.

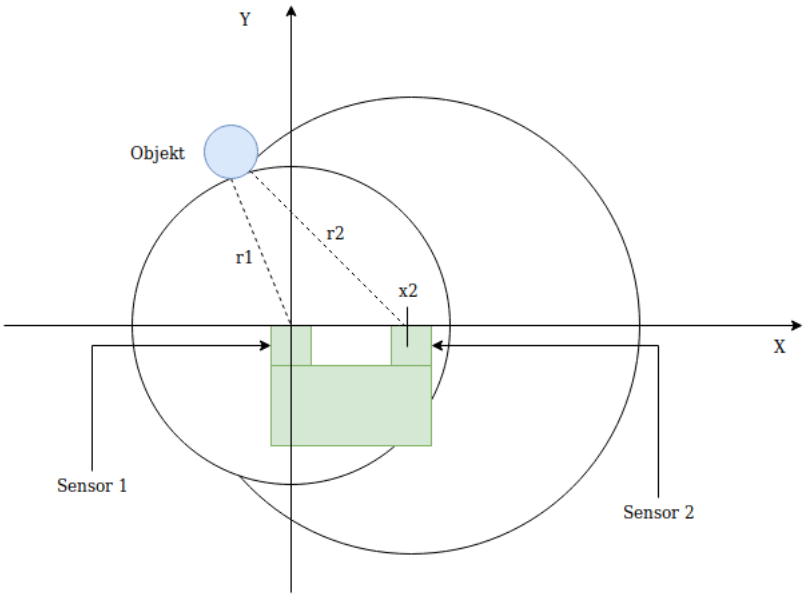


Figure 9: Ultrasonic: The difference in distance is used to find the position of the object

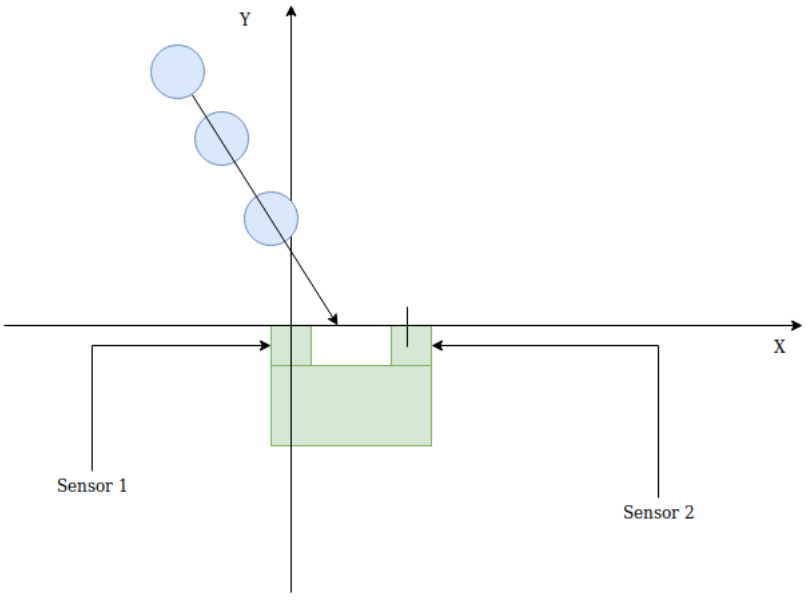


Figure 10: Ultrasonic: The different positions over time are used to estimate the future position of the object

5.5.4 Combining the conclusions from the camera and ultrasonic sensors

Both the camera system and the ultrasonic system output a probability of collision. In order to combine these conclusions, we have to take into account that the delay of the systems might be different. If we simply take a (weighted) average, we could risk that one of the systems output a high probability while the other outputs a low probability, and shortly after the opposite happens. This would then only lead to a medium value, even if both the systems report a high probability of collision, just with a delay between them.

Because of this, we simply take the maximum value of the two conclusions, which ensures that the final conclusion is not lower than it should be.

5.6 The PCB

In order to cleanly connect the project's various components, a custom PCB was designed. The design behind the BRICOLEUR PCB was inspired by SiliconLabs' STK3700 board, in part because of the similar MCUs. Schematics and a complete list of components can be found in appendix A: Schematics and B: Components.

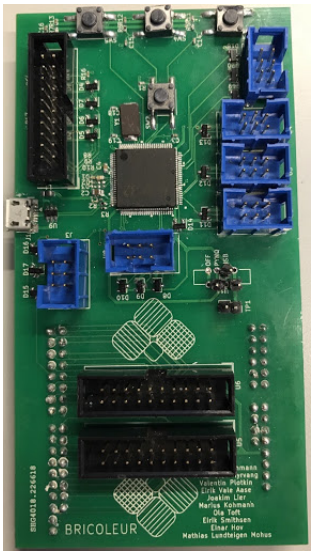


Figure 11: Assembled PCB

5.6.1 Features

- Three switchable power sources: USB, PYNQ and external.
- J-Link compatible debug header
- Two camera headers
- Three ultrasonic sensor connectors
- Header for UART-based bootloader communication
- Two UART/USART compatible input/output headers
- Arduino-compatible main header
- ESD protection on all inputs
- Debug LEDs and buttons

5.6.2 Design

The PCB was designed to be in accordance with Silicon Labs' application notes. The following documents were found to be of particular relevance:

- AN0002 - Hardware Design considerations
- AN0003 - UART Bootloader
- AN0042 - USB/UART Bootloader
- AN0046 - USB Hardware Design Guide

5.6.3 Power supply

The BRICOLEUR PCB is designed to operate at 3.3V. Other voltages are not officially supported, and all parts connected to the board are expected to operate at this voltage level. Power may be supplied to the board and its components in three ways:

1. USB: A microUSB cable can be connected directly to the board from a power supplying USB device. The MCU's internal voltage regulator converts the USB standard's 5V to 3.3V, which is then output on all power lines. Note that the PCB will not be able to power other devices through USB, even if it is provided power from another source.
2. PYNQ: The PYNQ Z1 Arduino Uno-style header contains a power pin which can output 3.3V directly to the PCB. This is expected to be the main mode of operation for the system.
3. Header: A dedicated power pin has been exposed. It is intended primarily for debugging and as a backup. This point of entry has no safeguards: Be careful to ONLY apply 3.3V.

5.6.4 PCB bricolage

Shortly after placing the PCB order it was discovered that the way in which the ultrasonic sensors were connected would require some additional hardware, namely diodes. The reason for this was the pin-saving way in which the sensor outputs were wired. Since each ultrasonic sensor would be polled in order, out of fear for interference, it was thought that they could safely share a single MCU input pin. However, this opened up for the possibility that the high signal from one sensor could be sent to another sensor whose pin was currently driven low, causing a short circuit. Diodes solve this issue by limiting the current direction. Because the board had no additional space for diodes, the cables were instead selected to be their mounting point.

Additionally, it turns out that the footprint intended for the power switch was poorly dimensioned. This has been amended by replacing the switch with a set of header pins and an externally wired power switch. A jumper between the correct pins can perform the switch's duty as well.

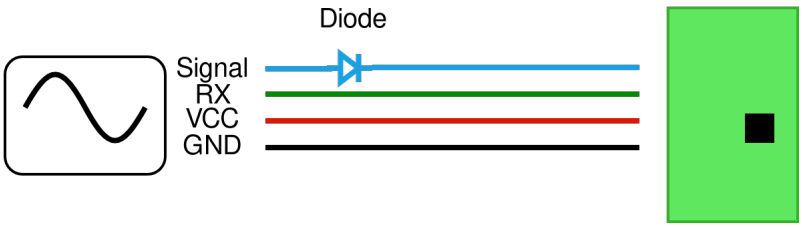
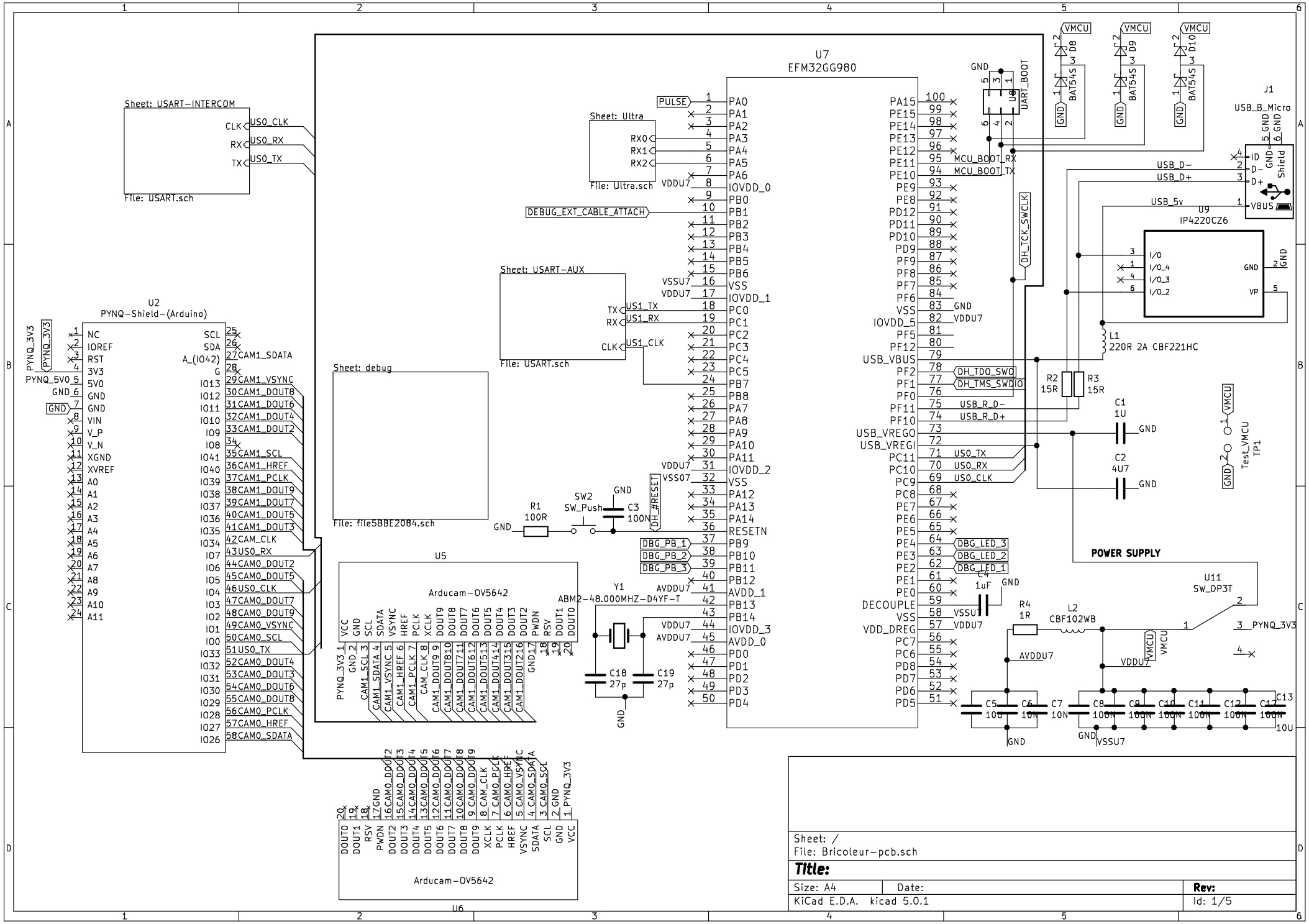
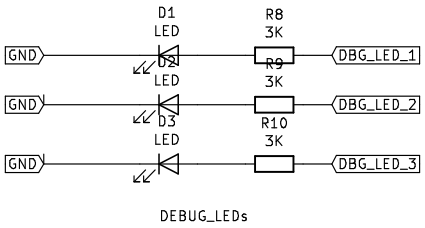
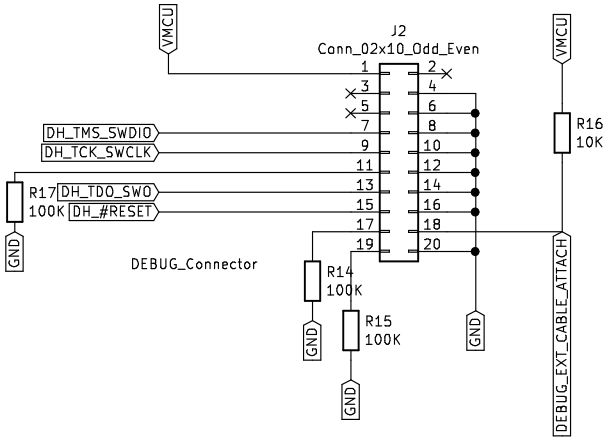


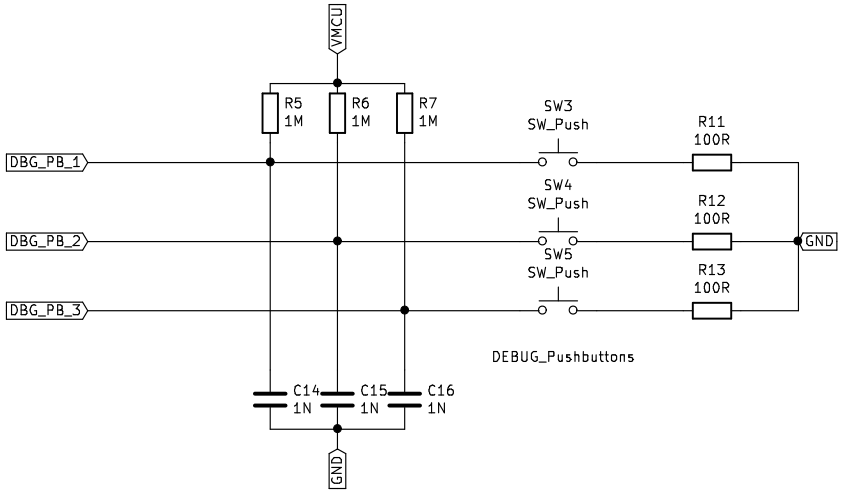
Figure 12: The cables connecting the ultrasonic sensors to the PCB

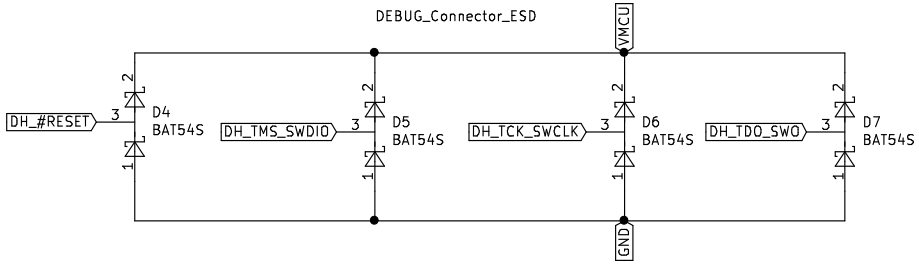
6 Appendix A: Schematics


Sheet: USART-INTERCOM
CLK
RX
TX
US0_CLK
US0_RX
US0_TX
File: USART.sch
U7
EFM32GG980
Sheet: Ultra
RX0
RX1
RX2
File: Ultra.sch
PULSE
DEBUG_EXT_CABLE_ATTACH
Sheet: USART-AUX
TX
RX
CLK
US1_TX
US1_RX
US1_CLK
File: USART.sch
Sheet: debug
File: file5BBE2084.sch
U2
PYNQ-Shield-(Arduino)
U5
Arducam-OV5642
U6
Arducam-OV5642
SW2
SW_Push
R1
100R
C3
100N
GND
DH_#RESET
Y1
ABM2-48.000MHZ-D4YF-T
C18
27p
C19
27p
DBG_PB_1
DBG_PB_2
DBG_PB_3
UART_BOOT
MCU_BOOT_RX
MCU_BOOT_TX
DH_TCK_SWCLK
DH_TDO_SWO
DH_TMS_SWDIO
USB_R_D-
USB_R_D+
US0_TX
US0_RX
US0_CLK
VMCU
D8
D9
D10
BAT54S
J1
USB_B_Micro
USB_D-
USB_D+
USB_5v
U9
IP4220CZ6
L1
220R 2A CBF221HC
R2
15R
R3
15R
C1
1U
C2
4U7
TP1
Test_VMCU
POWER SUPPLY
U11
SW_DP3T
C4
1uF
R4
1R
L2
CBF102WB
C5
10U
C6
10N
C7
10N
C8
100N
C9
100N
C10
100N
C11
100N
C12
100N
C13
10U
VDDU7
AVDDU7
VSSU7
PYNQ_3V3
PYNQ_5V0
Sheet: /
File: Bricoleur-pcb.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 5.0.1
Id: 1/5

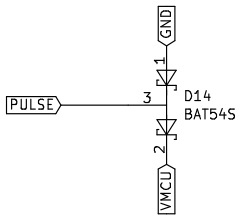


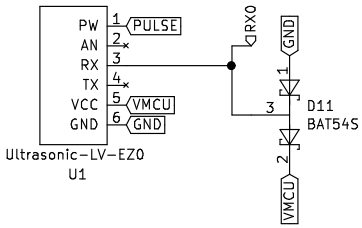


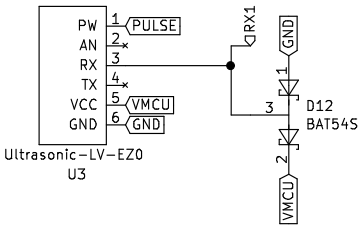


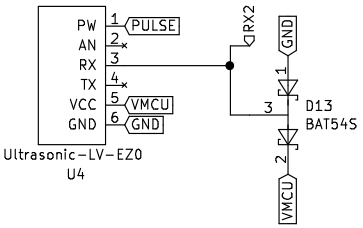


Sheet: /debug/ File: file5BBE2084.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.0.1		Id: 2/5

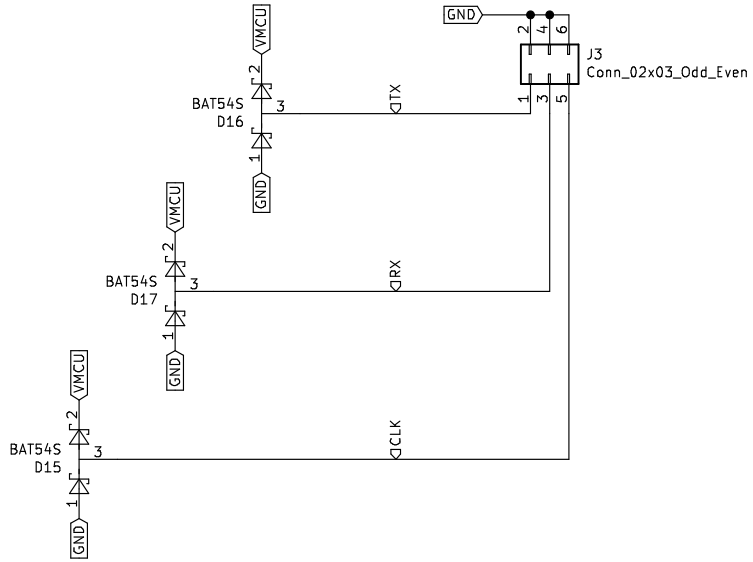




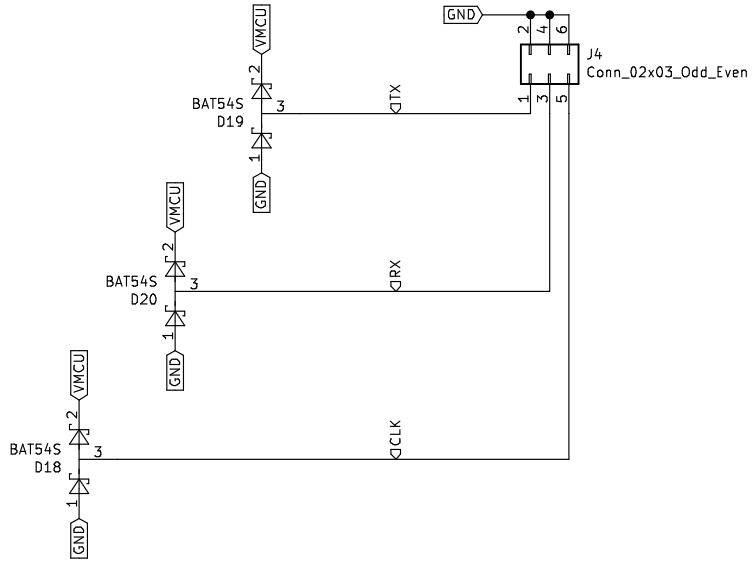




Sheet: /Ultra/		
File: Ultra.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.0.1		Id: 3/5



Sheet: /USART–INTERCOM/		
File: USART.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.0.1		Id: 4/5



Sheet: /USART-AUX/		
File: USART.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.0.1		Id: 5/5

7 Appendix B - Components

The following is a list of all components used in the assembly of one copy of the BRICOLEUR system.

7.1 Pre-made components

Component	Quantity
CMOS OV5642 Camera Module	1
LV-MaxSonar-EZ0 MB1000	2
PYNQ-Z1	1
Cardboard container	1

7.2 PCB BOM

Designation	Footprint	Stat	Distributor	Distributor ID	Qty
U7	QFP100	EFM32GG980	Mouser	634-32GG980F1024-E	1
C18,C19	0402	27pF	Digikey	478-11777-1-ND	2
C17,C3,C8,C9, C10,C11,C12	0402	100nF	Digikey	1276-1004-1-ND	7
C1	0201	1μF	Digikey	490-7229-1-ND	1
C4	0603	1μF	Digikey	1276-6524-1-ND	1
C5, C13	0805	10μF	Digikey	1276-6456-1-ND	2
C6, C7	0402	10nF	Digikey	490-4762-1-ND	2
C14,C15,C16	0402	1nF	Digikey	490-6348-1-ND	3
C2	0805	4.7μF	Digikey	399-3134-1-ND	1
R14,R15,R17	0402	100kΩ	Digikey	MCS0402-100K-CFCT-ND	3
R1,R11,R12,R13	0402	100Ω	Digikey	RR05P100DCT-ND	4
R10,R9, R8	0402	3kΩ	Digikey	RR05P3.0KDCT-ND	3
R2, R3	0402	15Ω	Digikey	P15DDCT-ND	2
R4	0402	1Ω	Digikey	311-1.00LRCT-ND	1
R5,R6,R7	0402	1MΩ	Digikey	311-1.00MLRCT-ND	3
R16	0402	10kΩ	Digikey	RR05P10.0KDCT-ND	1
L1	0805	2200nH	Digikey	732-1614-1-ND	1
L2	0603	10μH	Digikey	732-4484-1-ND	1
SW2,SW3,SW4,SW5	6x6mm	switch	Farnell	2065105	4
U9	3.1x3.0mm	1pF 1MHz	Digikey	1727-3578-1-ND	1
J1	8.4x6.4mm	USB_Micro-B	Digikey	609-4050-1-ND	1
U1,U3,U4,J3,J4	9.4x15.8mm	IDC 2x3	Digikey	609-2845-ND	5
U5, U6,	33.5x9.4mm	IDC 2x10	Digikey	ED10524-ND	2
D1, D2, D3	0603	LED	Digikey	732-4981-1-ND	3
D9,D18,D15,D16, D17,D6,D4,D19, D20,D11,D12,D13, D14,D7,D8,D10, D5	SOT-23	ESD	Farnell	1081194RL	17
Header pins	–	2.54mm	ΩVerksted	–	60

7.3 Connectors

Designation	Stat	Distributor	Distributor ID	Qty
IDC-Female small	2x3pin	Digikey	649-71600-306LF	6
IDC-Female med.	2x10pin	Digikey	ED10503-ND	3
Flat cable	6 lines	Digikey	732-11799-ND	1m
Flat cable	20 lines	Digikey	732-11806-ND	1m
Diode	100MHz	ΩVerksted	–	2

8 Appendix C - Software and Tools

Vivado 2018.2 Used for importing Chisel 3 components in the form of Verilog files, wiring them up and connecting to other parts of the PYNQ board, like the DMA controller.

Simplicity Studio 4 Silicon Labs' standard IDE for EFM chips.

KiCAD 5.0.1 Primary PCB design tool.

Macaos PCB verification and ordering through intermediate supplier.

efm32-base An EFM32 base project that allows development on the command line. `https://github.com/ryankurte/efm32-base/`

Chisel 3.1.3. Library used to write hardware components that are imported into Vivado.

SBT 1.1.1. Used to compile and test Chisel project.

Scala 2.11.12. Programming language that Chisel components are written in.

Xilinx PYNQ PYNQ-Z1 v2.3

gcc 7.2.1

arm-none-eabi toolchain 7.2_2017q4

GNU netcat For piping data between laptop and PYNQ over TCP

ffmpeg To decode video to raw grayscale and to play the debug video output

9 Appendix E - Pipeline figures

9.1 Gaussian Blur

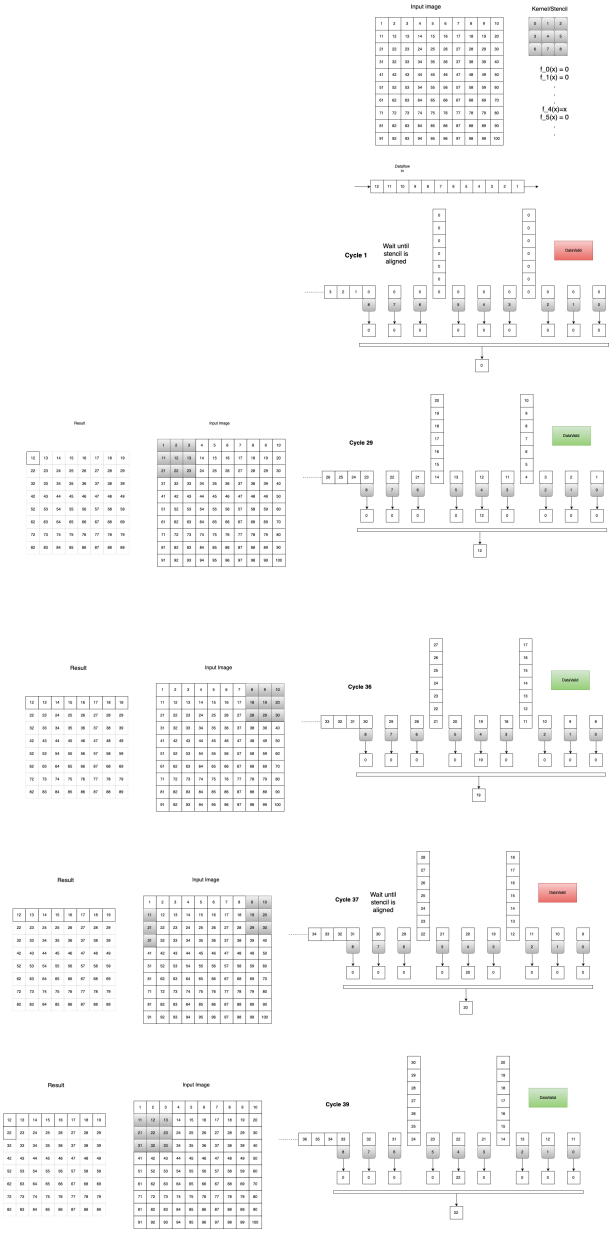


Figure 13: Illustration of the logical functionality of the Gaussian Blur component.

References

[1] Jason Cong, Peng Li, Bingjun Xiao, and Peng Zhang. An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers. In *Proceedings of the 51st annual design automation conference*, pages 1–6. ACM, 2014.