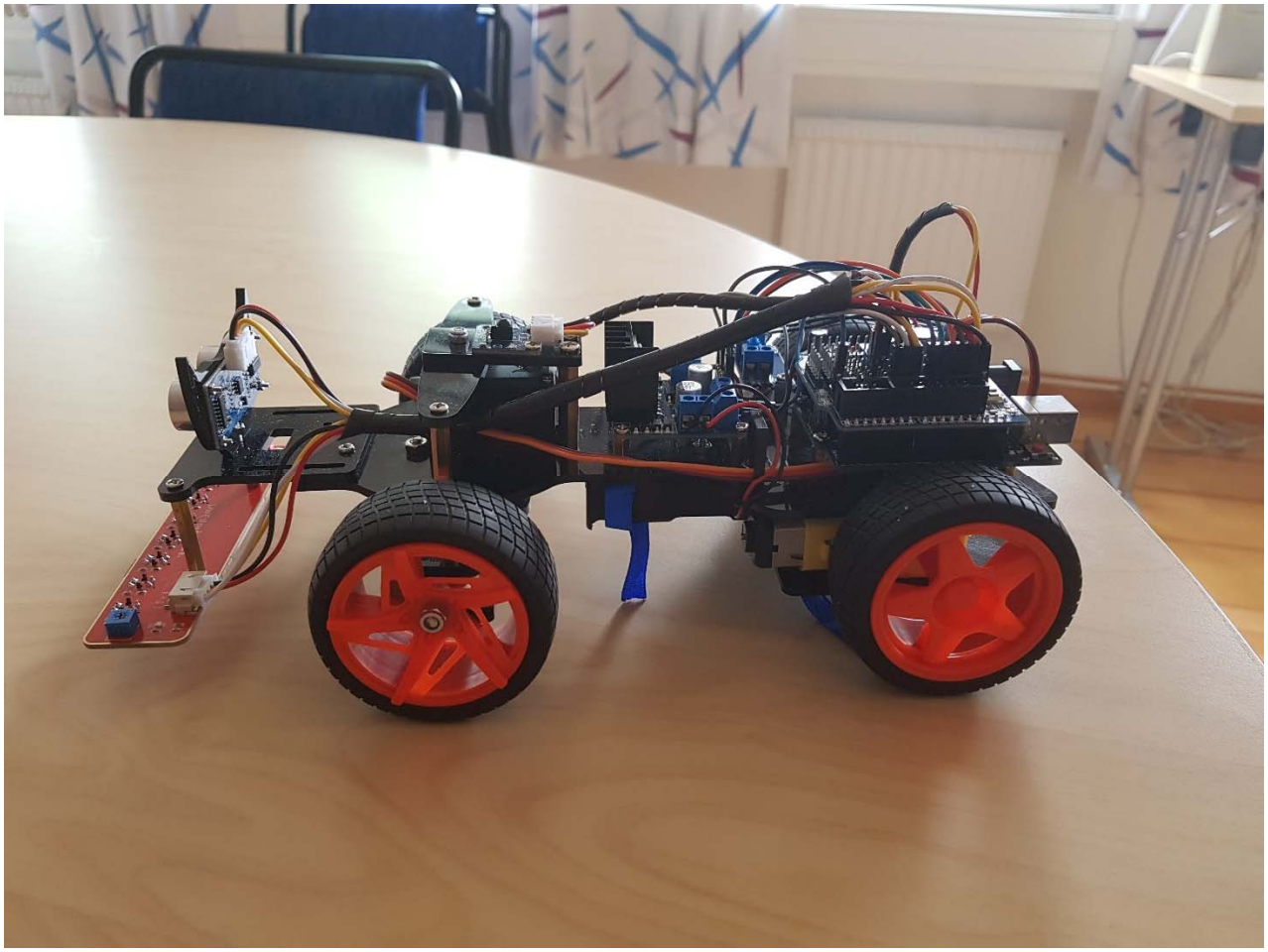
CHALMERS



# Autonom RC-bil

## via FPGA

Examensarbete inom högskoleingenjörsprogrammet Elektroingenjör

DIZDAR, MUHAMED
HARMAT, SRDJAN

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2017

# Förord

Den här rapporten är en del utav examensarbetet som utförts på institutionen för Signaler och System på Chalmers tekniska högskola. Arbetet omfattar 15 högskolepoäng, vilket är det avslutande steget inom högskoleingenjörsprogammet Elektroingenjör(180hp).

Examensarbetet utfördes hos företaget Synective Labs där en autonom radiostyrd bil med hjälp av FPGA och kamera konstruerades. Den tillbringade tiden hos företaget har varit väldigt lärorik då den har gett oss en mycket bättre förståelse hur FPGA-kort och signaler fungerar samt kunskapen hur man lägger upp ett projekt.

Vi vill tacka Håkan Dahlbom som varit handledare på företaget men även de anställda Carl Ehrenstråhle, Carl Gustafsson, Anders Petersson och Magnus Peterson som hjälpt oss under projektets gång. Vi vill dessutom tacka vår examinator Xuezhi Zeng vid Instutionen för Signaler och system för handledning under arbetets gång och skötseln av de administrativa bitarna.

Göteborg september 2017.

Muhamed Dizdar
Srdjan Harmat

# Sammanfattning

Denna rapport behandlar konstruktionen av en autonom RC-bil och dess delsystem. Företaget Synective Labs där arbetet utfördes är Nordens största leverantör av FPGA-tjänster och vill ta fram en egen version av självkörande RC-bil via FPGA och kamera. Detta för att ta fram ett demonstrationsexemplar som kan visas upp på mässor och utställningar.

Under projektets gång har det skapats en RC-bil som styrs via seriell data via en FPGA. Dessvärre är den inte autonom utan det simuleras med hjälp utav en dator tänkbara scenarion.

Styrning och kommunikation av resterande delsystem fungerar felfritt. Konstruktionen utav delsystemen är konstruerade på samma sätt som om VGA-kameran skulle ha linjedetektering implementerat. För att implementera LKA krävs det filtrering utav bilden och detta är något som kan erhållas med hjälp utav två olika algoritmer.

Under projektet gjordes det stora avgränsningar då projektets omfattning inte skulle hinna utföras inom den tänkta tidsramen.

## Abstract

This report deals with the design of an autonomous RC car and its subsystems. The company Synective Labs where the work was performed, is the Nordic region's largest provider of FPGA services and wants to produce its own version of self-driving RC car through FPGA and camera. The reason why they want to produce a demonstration copy is because they want to show the possibilities of what you can do with a FPGA. The product will then be shown at trade fairs and exhibitions.

During the project, an RC car has been built which is controlled by serial data through a FPGA. Unfortunately, the RC- car is not autonomous but imaginable scenarios where simulated.

Control and communication of the remaining subsystems works flawlessly. The subsystems are designed in the same way as the implementation of line detection. To implement LKA, it requires filtering of the image and this is something that can be obtained with the help of two different algorithms.

During the project, large boundaries were made but the scope of the project would not be carried out within the intended timeframe.

# Innehåll

# Terminologi/Förkortningar

| | |
|---|---|
| **FPGA** | Field-Programmable Gate Array |
| **SoC** | System-on-a-chip |
| **VGA** | Video Graphics Array |
| **IR** | Infrared |
| **RC** | Radio controlled |
| **RGB** | Red Green Blue |
| **LKA** | Lane keep assist |
| **LLC** | Logic level converter |
| **UART** | Universal Asynchronous Receiver/Transmitter |
| **CMOS** | Complementary metal-oxide-semiconductor |
| **RX** | Receiver |
| **TX** | Transmitter |
| **FPS** | Frames per second |
| **SSCB** | Serial Camera Control Bus |
| **PWM** | Pulse Width Modulation |
| **ASIC** | Application Specific Integrated Circuits |
| **VHDL** | Very High Speed Integrated Circuit Hardware Description Language |
| **XDSK** | Xilinx Software Development Kit |

# 1.INLEDNING

## 1.1 Bakgrund

Arbetet är utfört på företaget Synective Labs som med hjälp utav FPGA-kort utför högpresterande beräkningar. Anledningen till att företaget vill ha ett ”självkörande fordon” är att de vill ha ett projekt som skall kunna demonstreras på mässor för att kunna visa vad som kan uppnås med hjälp utav en FPGA-kort med tillhörande kamera.

## 1.2 Syfte

Syftet med arbetet är att ta fram en "autonom" Arduino baserad RC-bil som med hjälp av ett FPGA-kort, en kamera samt hjälpmedlet Lane Keep Assist ska kunna följa en utstakad bana.

## 1.3 Avgränsningar

På grund av tidsbegränsning och arbetets omfattning så har en del avgränsningar gjorts. För att kunna implementera Lane Keep Assist så krävdes att ett Sobel filter och en Hough transform konstruerades. Dessvärre fanns ej tiden för att konstruera detta. Fokus har istället lagts på att simulera ett liknande tänkbart scenario där samma signaler skickats med hjälp av en dator.

## 1.4 Precisering av frågeställningen

- På vilket sätt sker kommunikationen mellan FPGA-kortet och RC-bilens Arduino kort?

- Hur "skapas" Lane Keep Assist?

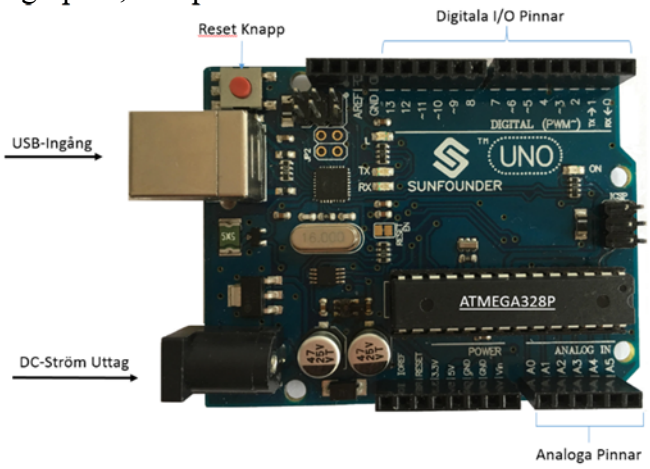
# 2. TEORI/TEKNISK BAKGRUND

Detta kapitel kommer att överskådligt behandla de komponenter och program som använts under projektets gång.

## 2.1 Arduino Board – Sunfounder Uno R3

Sunfounder Uno R3 är företaget Sunfounders egna version av kortet Arduino Uno R3, dess funktionaliteter, komponenter och specifikationer är näst intill identiska. Sunfounder Uno R3 är kompatibel med Arduino och använder sig av dess bibliotek och mjukvara.

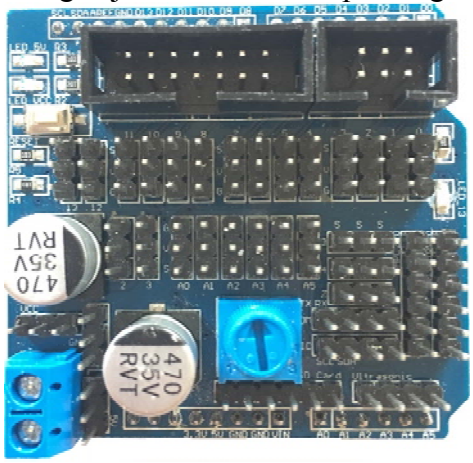
Hårdvaran Sunfounder Uno R3 är ett mikrokontroller kort vars uppgift är att förenkla och underlätta användningen av elektronik.[1] Kortet använder sig av en enkel kretsdesign med hjälp utav en mikroprocessor vid namn ATMEGA328P. Mikrokontrollerkortet har även stöd för signalhanteringen av in och utgångar samt PWM-signaler.  Samt att dess mjukvara består utav ett programmeringsspråk, kompilator samt boot loader.




*Figur 1. Sunfounder Uno R3*

### 2.1.1 Sensor Shield

Sensor Shielden är en komponent vars funktion är att agera förlängning åt Sunfounder Uno R3 kortet. [2]Signalerna som finns på Sunfounder Uno R3 kortet återfås på Sensor Shielden efter inkoppling. Fördelarna med denna metod är att antalet pinnar ökar, där bland annat varje analog och digital signal får sin egen jord och referensspänning.

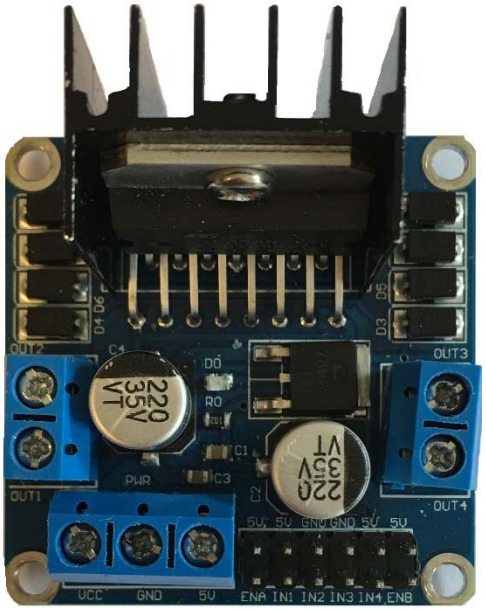



*Figur 2. Sensor shield.*

### 2.1.2 Motor driver module L298N

Motor driver modulens uppgift är att sköta vridmomentet på de två DC-motorerna som får bilen att röra sig i en framåtgående eller bakåtgående riktning. Chipet (L298N) på modulen är av hög spänning och ström karaktäristik och har 15st ben. [3] Den är kopplad med Sunfounder Uno R3 med hjälp av sex kablar som alla överför en PWM-signal. Det är utifrån PWM-signalerna som vridmomentet riktning bestäms.

PWM-Signalerna har två tillstånd, de kan vara ”höga” vilket innebär en ”etta” eller så kan de vara ”låga” vilket innebär en ”nolla”. Motorerna behöver tre signaler var, en signal som aktiverar och två signaler som utifrån antalet möjliga kombinationer skapar olika tillstånd. I detta fall är det maximala antalet kombinationer fyra stycken, vilket leder till fyra tillstånd.




*Figur 3. Visar motor driver modulen.*

*Tabell 1. Tillståndstabell för Motor A*

| ENA | IN1 | IN2 | Tillstånd av Motor A(Vänster) |
|---|---|---|---|
| **0** | X | X | Stopp |
| **1** | 0 | 0 | Bromsa |
| **1** | 0 | 1 | Rotera Medurs |
| **1** | 1 | 0 | Rotera Moturs |
| **1** | 1 | 1 | Bromsa |

*Tabell 2. Tillståndstabell för Motor B*

| ENB | IN3 | IN4 | Tillstånd av Motor B (Höger) |
|---|---|---|---|
| **0** | X | X | Stopp |
| **1** | 0 | 0 | Bromsa |
| **1** | 0 | 1 | Rotera Medurs |
| **1** | 1 | 0 | Rotera Moturs |
| **1** | 1 | 1 | Bromsa |

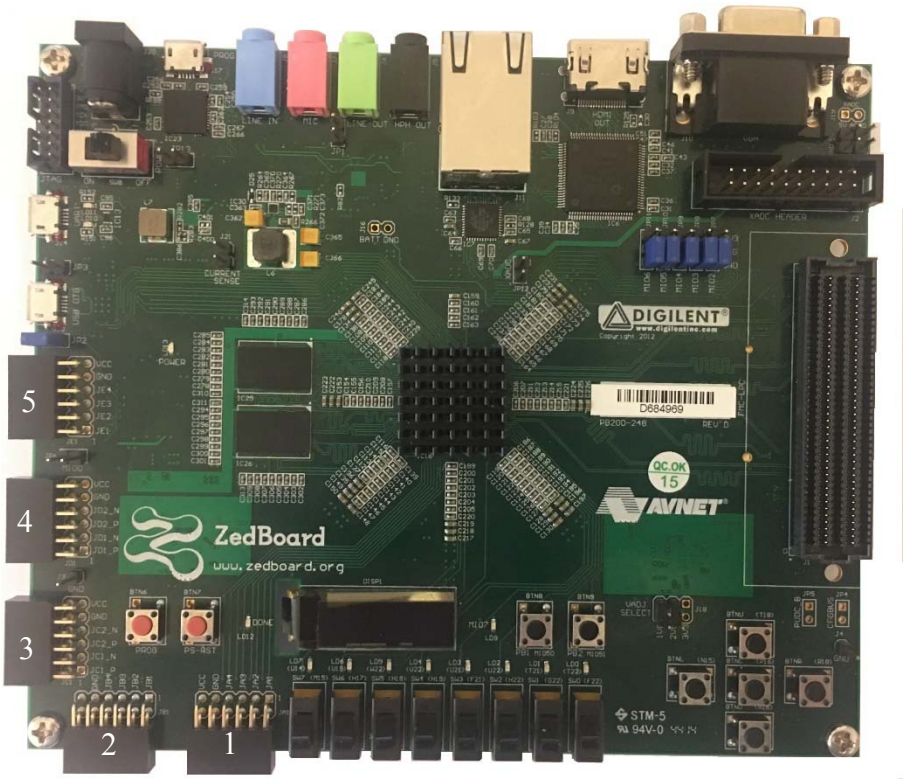
## 2.2 FPGA ZedBoard Zynq-7000 ARM/FPGA SoC Development Board

Field Programmable Gate Array mer känt som FPGA är halvledaranordningar baserade på en matris vilken innehåller konfigurerbara logikblock. [4] Dessa i sin tur är anslutna via programmerbara sammankopplingar. Det finns två typer av integrerade kretsar. FPGA kort tillhör de omprogrammerbara integrerade kretsarna och det finns många olika typer och tillverkare.

De kan programmeras om obegränsat antal gånger efter tillverkning medan den andra gruppen enbart kan programmeras en gång. Gruppen som enbart har möjligheten att programmeras om en gång tillhör gruppen ASIC. [5]

Programmeringen sköts via USB (Typ A) till micro usb kabel (Typ B) mellan dator och FPGA. De flesta och mest använda FPGA:s är SRAM-baserade och omprogrammerbara. Nackdelen med FPGA:s är att de tappar minnet så fort spänningen bryts. Detta innebär att det krävs omprogrammering varje gång spänningen har brutits. Vissa tillverkare utav FPGA:s har lyckats implementera så att det senast inprogrammerade program laddas om så fort den spänningssätts igen. Programmet för en FPGA skrivs i ett hårdvarubeskrivande språk, vilket är antingen VHDL eller Verilog. [4] [5]

I detta projekt användes ett FPGA-kort från tillverkaren Xilinx vid namn ZedBoard Zynq-7000 ARM/FPGA SoC Development Board. Det är ett lågkostnads utvecklingskort med bland annat 512 MB DDR3 RAM, HDMI- och VGA-port, dubbla ARM Cortex A9 kärnor, switchar, LEDS, knappar och en P-mod samt FMC kontakt som kan ses på bilden nedan. [6]




*Figur 4. Zedboard är det FPGA-kortet som använts i projektet. P-mod kontakter numrerade från 1 – 5.*

### 2.2.1 P-mod kontakt

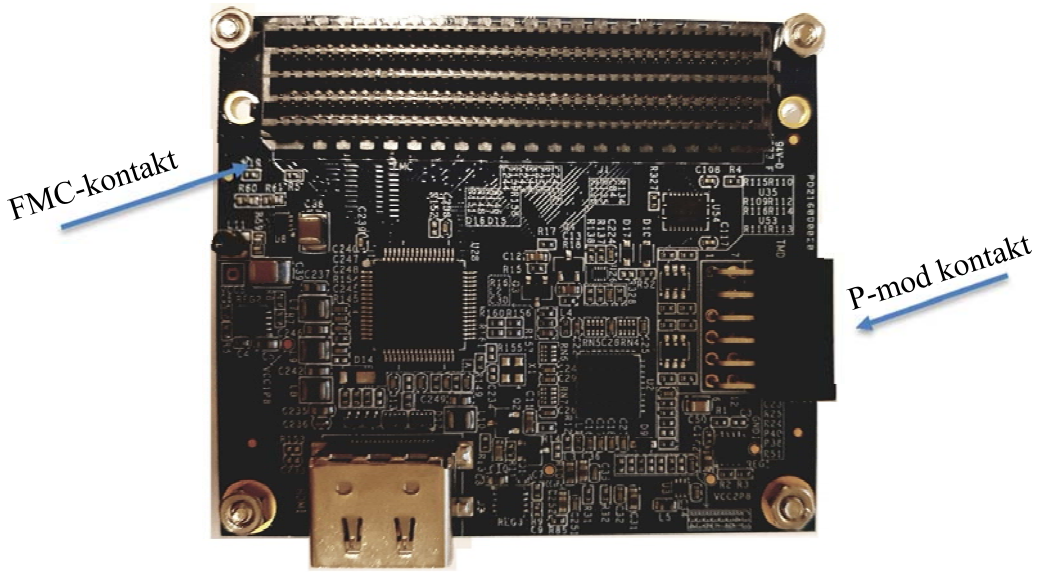
P-mod interface står för Peripheral Module interface vilket är framtaget utav Digilent Inc och används för att koppla ihop kringutrustning med bland annat FPGA:s. Det är en kontakt med antingen sex eller tolv ben. Kontakten med sex ben har fyra digitala I/O signalben och en vardera för jord och spänning medan den andra kontakten med tolv ben är två ihop staplade sex-bens kontakter. Denna typ av kontakt använder sig utav en spänning på 3,3 volt. Det finns två olika typer av spänningsstandarden och dessa är LVCMOS 3,3V eller LVTTL 3,3V. [7]

### 2.2.2 FMC - kontakt

FMC står för FPGA Mezzanine Card vilket är en ANSI/VITA standard utvecklad av flertalet företag. Zedboardet använder sig utav en standard vid namn VITA 57.1 vilket möjliggör en sammankoppling mellan expansionskort och FPGA:s av samma standard. Det är en kontakt som är av lågprofilstyp för att ha flera olika användningsområden som exempelvis kort avsedda för industristandarden. Den här typen av standard har stöd för dataöverföring med upp till 10 Gb/s och en möjlig total bandbredd på 40 Gb/s mellan expansionskort och huvudkort. [8]

### 2.2.3 D8M-FMC kamera

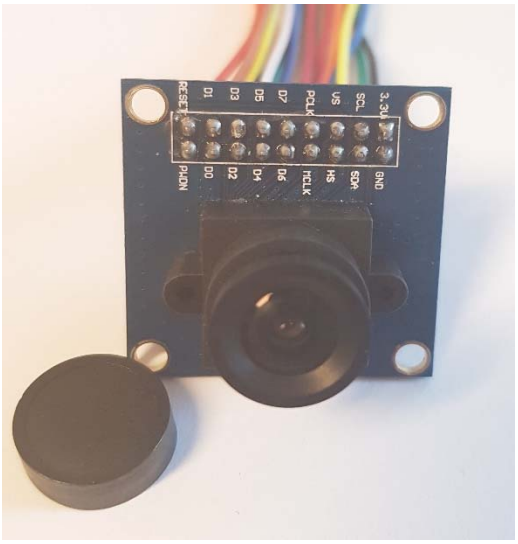
Till projektet köptes det in en kamera D8M-FMC vilken kommer från tillverkaren terasIC och är till skillnad från OV7670 en kamera med hög prestanda. Den består utav en sensor på åtta megapixel med autofokus och en HDMI sändare som kan användas till att mata ut bilden på HDMI-porten. Detta medför att man kan se den bearbetade bilden på en extern bildskärm. Kameran kan anslutas till vilket som helst FPGA-kort som har en FMC-kontakt som likt kortet följer VITA 57.1 standarden. Genom att styra kameran via I2C-gränssnittet på en FPGA kan man dessutom ändra upplösningen på bilden men även bildhastigheten (frame rate) där kameran har stöd för upp till 30 FPS i 1080p. [9]




*Figur 5. D8M-FMC kamera.*

## 2.3 VGA-kamera OV7670

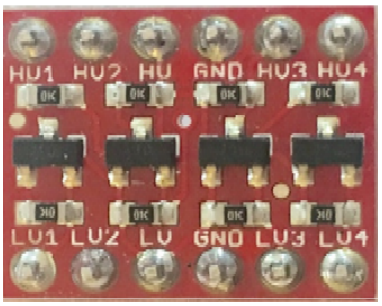
OV7670 är en CMOS-bildsensor som är uppbyggd på lågspänning. Den förser en VGA-kamera med full funktionalitet och bildbehandling i ett litet paket. Sensorn tillhandahåller full-frame, windowed men även sub-sampled 8-bitars bilder vilka kontrolleras via SSCB gränssnittet. En fullständig kontroll över bildkvalitet görs samtidigt som formatering och utmatning av data. När detta utförs samtidigt medför det en kapacitet för sensorn på upp till 30 FPS. Nödvändiga bildbehandlingsfunktioner som exempelvis vitbalans, gamma, mättnad och nyanskontroll kan programmeras via SSCB. [10]




*Figur 6. VGA-kamera OV7670*

## 2.4 Logic Level Converter Bi-Directional BoB

Logic level converter är en komponent vars uppgift är att konvertera spänningar till önskad nivå. Den används för kommunikation mellan olika anordningar/apparater som inte har samma spänningsnivå. Denna komponent möjliggör den seriella kommunikation mellan FPGA:n och Sunfounder Uno R3 vars spänningsnivåer är 3,3 V respektive 5,0 V. [11]




*Figur 7. Logic Level Converter*

## 2.5 Servomotor MG995

I projektet så används det en servomotor vid namn MG995 och kommer från tillverkaren TowerPro. [12] Den här typen av servomotor kan rotera från 0° – 180°. I detta fall så är det en mer begränsad svängradie på grund utav RC-bilens konstruktion. I projektet så är ursprungsläget för servon 90° och kan rotera i intervallet 60° - 120°. Servon är kopplad till Arduinons sensor shield på analog pinne A0 med en fast kabel. Kabeln består utav tre stycken olika kablar som går ihop i en Dupont kontakt. Det innebär att första kabeln som är brun står för jord, den andra är röd och står för spänning medan den sista orangea är till för signal.


*Figur 8. Visar hur servomotorn som användes ser ut.*

## 2.6 USB till seriell adapter (PL2303HX)

En USB till seriell adapter användes ihop med PuTTY. Det är en USB kabel som innehåller ett litet chip som heter PL2303HX, där HX står för att det är den senaste versionen. Chipet är billigt och högpresterande. Kabeln möjliggör en väldigt enkel anslutning mellan en USB- och RS232 full-duplex asynkron enhet. flexibel signalnivå så möjliggör kabeln anslutning till enheter som är mellan 1,8–3,3 Volt. [13]

## 2.7 MJUKVAROR

### 2.7.1 Vivado HL WebPack Edition och Xilinx SDK

Programvaror som använts är Vivado HL WebPack Edition vilket är en lite begränsad fri utgåva utav det fullständiga programmet Vivado HL Design Edition. [14] Programmet är framtaget utav Xilinx och är anpassat till deras egna FPGA:s arkitektur. Det används för sammanställning och analys av HDL designer samt möjliggör för utvecklare att bland annat kompilera sin design, utföra tidsanalyser och även för granskning utav RTL diagram. Programmet ersatte ett tidigare program vid namn ISE Design Suite och skiljer sig väldigt mycket från det ursprungliga. Det innehåller bland annat en inbyggd logisk simulator för simulering till skillnad från det tidigare som använde sig utav ModelSim för samma ändamål. Vivado innehåller dessutom ett verktyg som kan omvandla C-kod till programmerbar logik, Programmet är omfattande och innehåller många behövliga saker som t.ex. algoritmer. [15]

Ett tillval till Vivado finns i form av XSDK, men det är ett krav och installera för att kunna programmera FPGA:n. XSDK används till för att skapa inbyggda applikationer på någon utav Xilinx mikroprocessorer och levererar homo- och heterogen design med flera processer, felsökning och prestationsanalyser. [16]

### 2.7.2 Arduino Software (IDE)

Arduino Integrated Development Environment eller Arduino Software (IDE) innehåller bland annat en texteditor där man skriver kod, ett fönster där man kan se exempelvis felmeddelanden men även ett verktygsfält med knappar och underliggande menyer som innehåller vissa behövliga funktioner. Programvaran används till att programmera Arduinon, men även till att kommunicera med den. Det finns en seriell monitor i programmet som kan användas i felsökningssyfte för att se om exempelvis koden fungerar som tänkt. [17]

### 2.7.3 PuTTY

PuTTY är ett program som möjliggör seriell kommunikation mellan två olika enheter. Det är en terminal liknande kommandotolken på en Windows dator. I programmet sätts kommunikationen upp genom att välja vilken COM-port på datorn enheten som skall sköta utbytet av data är kopplad till. En till viktig sak i programmet är att ställa in rätt dataöverföringshastighet i bytes/s, så kallat baud rate. Ställer man inte in de båda enheterna på samma dataöverföringshastighet så kommer det ej att mottas någon som helst data. [18]

# 3. METOD

Detta kapitel avser att på ett tydligt sätt beskriva projektets arbetsgång, kravspecifikation och felsökningsmetod.

## 3.1 Projektets arbetsgång

Det första som gjordes var att tillsammans med handledaren på företaget diskutera vilka krav och produkter som skulle användas samt så lades en plan upp på hur man skulle gå tillväga. Utifrån detta påbörjades informationssökning om berörda produkter för att säkerställa att rätt produkter beställdes. Informationsinhämtningen är något som pågick under hela projektets gång i form av litteratur, internet och videor. Det gjordes även sökningar på liknande projekt för att få en bättre överblick av olika sätt att angripa problemet.
Planen för projektet var att dela upp uppgiften i flera delblock och fokusera på ett delblock i taget för att sedan slå ihop det till ett helt system. Varje delblock utfördes med en liknande procedur. Första steget var att information inhämtades och bearbetades. Därefter konstruerades och testades delsystemet stegvis för att få en bättre förståelse och för att säkerställa att det som konstruerats fungerar så som det är tänkt. När väl delblocket ansågs vara färdigt så testades det för att se att önskad handling utfördes.

## 3.2 Kravspecifikation

Projektet innehöll inga specifika krav på produkternas prestanda, dock så var urvalet av produkter begränsat. Kraven som fanns var på bilens storlek och utseende då denna skulle visas upp som exemplar för företaget vid mässor och konferenser. Det fanns ytterligare ett krav, att en FPGA skulle användas ihop med en kamera. Detta för att kunna implementera LKA.

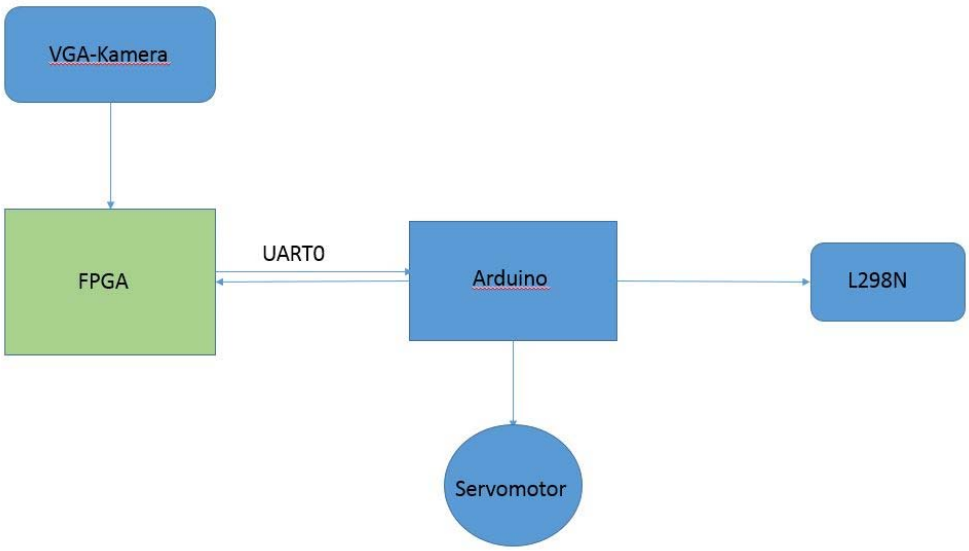
## 3.3 Säkerställning av data

Felsökning pågick ständigt under projektet. Detta för att säkerhetsställa att det inte fanns några som helst felaktiga delblock. Det användes en mängd olika felsökningsmetoder t.ex. så användes en USB till seriell adapter kopplad till en dator där data visades i programmet PuTTY. Ett oscilloskop användes även flertalet gånger för att se om rätt data eller signal överfördes, samt användning av en multimeter när sammanslagning av delblock gjordes. När det gäller felsökning inom kod/programskrivande så användes terminalerna i den mjukvara de skrevs i en hel del för att se att rätt värde skrevs ut.

# 4. KONSTRUKTIONSPROCESS AV SYSTEM OCH DELBLOCK

Systemet är uppbyggt av flera delblock som tillsammans utgör en helhet. Detta kapitel ska förklara processen för konstruktionen av varje delblock samt inkludera resultat efter programkörning.

## 4.1 Överblick av system




*Figur 9. Visar en överblick av system.*

## 4.2 Konstruktion av UART

För att veta hur kommunikation mellan FPGA och Arduino skulle konstrueras inhämtades och studerades information om olika tillgängliga kommunikationsprotokolls alternativ. Detta gjordes för att kunna utesluta de alternativ som inte ansågs uppfylla kravet. Kravet för denna kommunikation var att använda så få kablar som möjligt då antalet pinnar på Arduinon är begränsat.

Det smidigaste alternativet var att koppla FPGA:n via USB-kabel från USB-porten UART1 till Arduinon. Detta alternativ gick dock inte att utföra av den anledningen att Arduinon i projektet enbart hade en USB-ingång som redan användes till programmering av den.

Efter samtal och diskussioner med personer på företaget så föreslogs och förklarades UART:s funktioner och möjligheter djupgående. UART1 är något som är hårdkodat/inbyggt på FPGA:n och är bra att ha tillgängligt för felsökning. Under samtalet framkom det även att det går att konstruera ytterligare en UART genom att ”porta” UART1 till valfri P-mod kontakt på FPGA:n. Med ”porta” syftas det på att du gör samma signal tillgänglig på ett annat ställe. Den UART som gick att konstruera kallas för UART0.

UART fungerar på så sätt att parallell data omvandlas och skickas vidare som seriell data. Valet att använda UART innebär att information sänds utan att vara beroende av en yttre klocka. Denna typ av protokoll kräver dock att flertalet processer uppfylls och behandlas istället för använda sig av yttre klocka. Detta för att kunna säkerställa att information som skickas tas emot korrekt.

För att kunna överföra data från FPGA till Arduino kortet så måste data sändas och mottas vid samma dataöverföringshastighet (Antal bitar per sekund). Inläsning av information på Arduinon är möjligt tack vare att den har två stycken pinnar som sköter inläsning och sändning av information, även kallat Rx och Tx pinne.

Det måste även finnas något som signalerar när data ska läsas in samt längden på data. Detta regleras med hjälp av en startbit och en stoppbit. Startbiten ligger före data och signalerar till mottagarsidan vilken hastighet den överförs med. Stoppbiten signalerar när data är slut och att invänta nästa datapaket.

Konstruktionen av UART0 gjordes utifrån en guide. [19]

Metoden för att skapa UART0 kan visuellt följas i guiden. Först och främst skapades det ett nytt RTL projekt i programvaran Vivado. Nästa steg var att det lades till ett ”IP-block”. IP-blocket som lades till hette ZYNQ7 Processing System och är Zedboardet ARM del. Att ha med detta block i varje blockdesign är ett grundkrav, då detta är platsen som program skrivs till vid programmering av FPGA i programvaran XDSK.

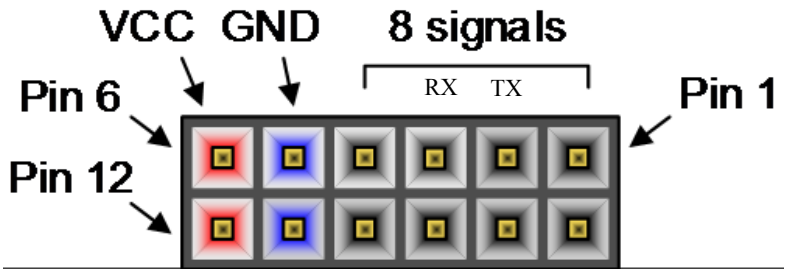
En ”Run block automation” gjordes genom högerklickning på processing system. Programmet kopplar automatisk ihop två signaler FIXED_IO och DDR och gör de externa. Nästa steg är att läggs det till ett AXI Timer block vars uppgift är skötseln av PWM signalen. PWM-signalen kommer även den att göras extern dock efter att funktionen att automatisk koppla och lägga till block i Vivado vid namn Connection Automation utförts. Connection Automation var det sista steget för konstruktion av blockdesignen.

Genom dubbelklickning på processing system blocket kan olika kommunikationsprotokoll aktiveras. I projektet aktiverades protokollen SPI, I2C och UART0. Under projektet användes dock enbart UART0 av den anledningen att den krävde minst antal kablar då den skickar informationen seriellt.

När de hade aktiverats så var de dags och välja vart de skulle portas vidare. Valet föll på EMIO pinnarna. Det är de pinnar som återfinns på Zedboardets P-mod kontakter. När allt var portat till rätt pinnar så dök alla kommunikationsprotokoll upp i processing system blocket. Där gjordes alla kommunikationsprotokoll externa.

Den nu färdiga blockdesignen har ett par steg kvar innan den kan programmeras över till FPGA:an. Det krävs att en bitstream genereras. Bitstream är den fil som krävs för att man skall kunna exportera projektet till hårdvara. Filen kan först genereras då vi valt vilka I/O portar som skall portas vart. Valet av I/O portar avgör vilken P-mod kontakt som används samt på vilket ben signalerna finns tillgängliga på.

I projektet så portades I/O portarna till P-mod kontakten JC1. De tillgängliga signalerna på JC1 fanns på var Tx (Pin 2) och Rx(Pin 3). P-mod kontakten kan ha upp till 8 signaler tillgängliga (VCC och GND exkluderade).

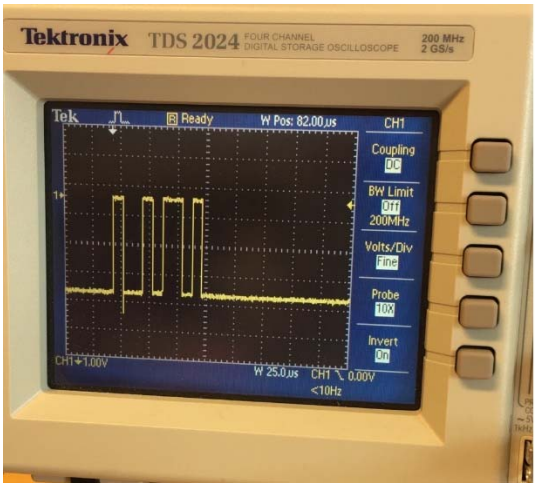



*Figur 10. P-mod kontakten [20]*

| Pmod | Signal Name | Zynq pin |
|---|---|---|
| JC1 Differential | JC1_N | AB6 |
| | JC1_P | AB7 |
| | JC2_N | AA4 |
| | JC2_P | Y4 |
| | JC3_N | T6 |
| | JC3_P | R6 |
| | JC4_N | U4 |
| | JC4_P | T4 |

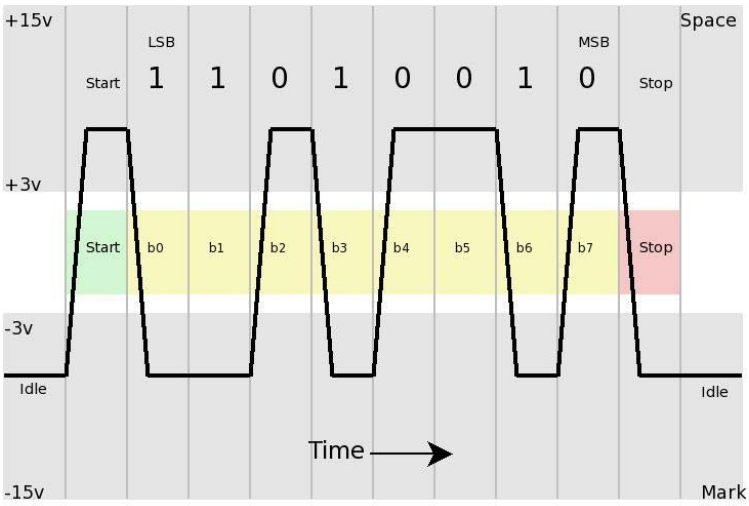
*Figur 11. Visar P-mod JC1:s signalnamn samt vilka som används till UART0 TX och RX. [21]*

För att verifiera och se att konstruktionen av UART0 var lyckad skrevs ett program i Vivado Programmet fungerade så att det skickade ut ett förbestämt ASCII-tecken på Tx-pinnen på JC1. Under testet valdes ASCII-tecknet 0x4b, vilket motsvarar ett stort K. För att se att det förbestämda ASCII tecknet skickades ut mättes det med hjälp av ett Oscilloskop på Tx-pinnen. Vid mätning mottogs rätt information.




*Figur 112. Visar hur det ser ut när ett "K" skickas.*

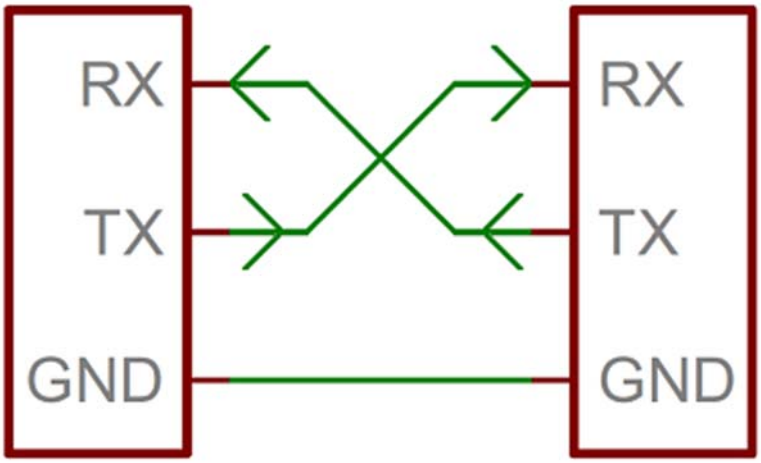
Den binära tolkningen för ett ”K” är 0100 1011 och genom att tolka signalen på oscilloskopet från höger till vänster så kan man se att det är det värdet som erhålls. Den mest signifikanta biten är den till höger i figuren. Som en jämförelse att det verkligen är ett ”K” som mottages så kommer vår signal att jämföras med figur 13 nedan.




*Figur 13. Visar hur det skall se ut när ett "K" skickas seriellt. [22]*

## 4.3 Kommunikation mellan FPGA och Arduino

De två signaler (Tx,Rx) som gjorts tillgängliga på P-mod JC1 kopplades till Arduino med hjälp av kablar. Signalerna var dock tvungna att konverteras upp med hjälp av LLC innan de kunde anslutas till Arduinon. Detta pga. att referensspänningen i P-moden ligger på 3,3 V, medan Arduinons referensspänning ligger på 5V. Kopplingen av signalerna konstruerades enligt figur 11.




*Figur 14. Visar hur det är kopplat mellan Arduino och FPGA.*

## 4.4 Implementation av OV7670 kamera ihop med Zedboard

Informationssökningen för att implementera kameramodulen D8M-FMC var väldigt begränsad då det fanns oerhört lite underlag kring denna komponent. Det enda som påträffades var tre olika demonstrationsexempel där det användes en helt annan tillverkare av FPGA-kort. [23] En implementation via programmeringsspråket Verilog skulle kunna vara möjlig genom användning av dessa exempel. Tyvärr fanns det varken tid eller kunskap för inlärning av språket Verilog.

Då beslut togs att inte använda sig utav FMC-kameran söktes nu nya alternativ till att koppla en kamera till FPGA-kortet. Efter informationssökning beslutades istället användning av VGA-kameran OV7670. Det är inte en lika högupplöst kamera men är av tillräcklig kvalitet för implementation av linjedetektering.

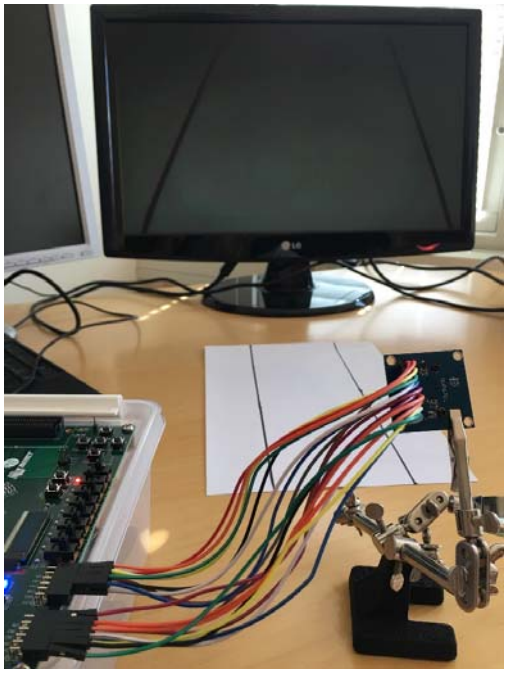
En implementation utav OV7670 ihop med FPGA-kortet Zedboard möjliggjordes genom att använda sig utav offentligt delad kod som modifierats. [24] Se bilaga A-A.6.

Konstruktionen mellan dessa två enheter skapades via kabelförbindelser. Tabellen nedan visar hur signalerna är kopplade.

Tabell 3. Lista över signalnamn mellan OV7670 och Zedboard.

| Signalnamn OV7670 | Signalnamn P-mod JA | Zynq Pin |
|---|---|---|
| **OV7670_PWDN** | JA1 | Y11 |
| **OV7670_RESET** | JA5 | AB11 |
| **OV7670_D0** | JA2 | AA11 |
| **OV7670_D1** | JA6 | AB10 |
| **OV7670_D2** | JA3 | Y10 |
| **OV7670_D3** | JA7 | AB9 |
| **OV7670_D4** | JA4 | AA9 |
| **OV7670_D5** | JA8 | AA8 |
| **Signalnamn OV760** | **Signalnamn P-mod JB** | **Zynq Pin** |
| **OV7670_D6** | JB1 | W12 |
| **OV7670_D7** | JB5 | V12 |
| **OV7670_XCLK** | JB2 | W11 |
| **OV7670_PCLK** | JB6 | W10 |
| **OV7670_HREF** | JB3 | V10 |
| **OV7670_VSYNC** | JB7 | V9 |
| **OV7670_SIOD** | JB4 | W8 |
| **OV7670_SIOC** | JB8 | V8 |

Efter att kameran kopplats ihop med FPGA:n utfördes ett test genom att koppla in en extern bildskärm på FPGA:ns VGA-kontakt. Detta för att säkerhetsställa att det fungerade korrekt. Ett körfält ritades upp på ett papper för att se hur bra skärpa det var från kameran. Figur 12 nedan visar hur det såg ut under testet.



*Figur 12. Demonstration av sammankoppling.*

## 4.5 Styrning av RC-bilen

I paketet med RC-bilen som användes i projektet medföljde flertalet färdiga Arduinoprogram. Dessa användes i början av projektet för att testa konstruktionen. I takt med att projektet växte fram skulle alternativ styrning implementeras. Den valda metoden för utomstående styrning utfördes via UART. Det medförde att motor driver modulen L298N samt servomotor MG995 fick implementeras på ett annorlunda sätt.

Implementering utav L298N utfördes på ett sådant sätt att det skapades kabelförbindelse mellan sensor shielden på Arduionon och L298N. Signalerna för kabelförbindelsen valdes utifrån ett krav på PWM signaler. Detta eftersom att L298N är den enhet som styr motorerna vilka tar stryk vid för höga konstanta spänningar. Genom användning utav PWM signaler skickas det istället för konstant spänning små pulser för att skydda komponenterna. De pin nummer och signaler som valts för styrning utav L298N återfinns i tabell 4.

Tabell 4. Signalnamn och pin nummer.

| **Signalnamn L298N** | **Pin nummer sensor shield** |
|---|---|
| **IN1** | 2 |
| **IN2** | 3 |
| **IN3** | 4 |
| **IN4** | 5 |
| **ENA** | 6 |
| **ENB** | 9 |

För att åstadkomma styrning av L298N via UART krävdes det ett egenskrivet bibliotek som importerades i projektet. Biblioteket hämtades från användaren Ivica_Matic [25] vilket la grunden till en fortsatt utveckling av programkod för Arduinon.

Användning av biblioteket möjliggjorde via L298N styrning av motorerna genom skapandet av en seriell förbindelse mellan dator och Arduino. Det här kompletterades sedan med ett annat bibliotek, SoftwareSerial.h för att skapa en virtuell port. Den virtuella porten användes till att öppna upp möjligheterna för utomstående styrning.

Ytterligare ett bibliotek adderades till programkoden i slutändan, det var servo.h vilket är det som möjliggör styrning av servon. Genom att kombinera ihop dessa tre bibliotek och skriva programkod erhölls det en RC-bil som kunde styras genom seriell data skickad via UART.

Styrningen utav bilen som erhölls sköttes via ett FPGA-kort som slumpvis skickade olika värden. Dessa värden tolkades i Arduinon som utförde olika handlingar.

## 4.6 Konstruktion av system

När alla delsystem var konstruerade så kunde de sättas samman till en och samma enhet. Hela systemet bestod då utav en FPGA, Arduino, VGA-kamera och RC-bil.

För att få hela systemet att arbeta ihop så krävdes det programkod i både Arduino- och FPGAmiljö. Programkoden för Arduino finns tillgänglig i bilaga B1. Eftersom FPGA:n i projektet implementerades till att agera sändare samtidigt som Arduinon utsågs till mottagare så var det olika upplägg på programkoden.

I Arduinokoden skapades det ett program som väntar på inkommande data på den virtuella Rx porten. När data mottagits så skrivs först värdet ut i en terminal i felsökningssyfte för att sedan gå vidare i programmet. Där hamnar man vid en switch case, vilket innebär att Arduinon står inför ett val beroende på vilket värde som mottages. Värdet som skall tas emot skickas via UART och kommer från FPGA:ns Tx port. Varje värde som tagits emot har en unik tolkning vilket innebär olika handlingar.

Programkoden för FPGA:n är lättläst och kan ses i bilaga B2. Den är konstruerad på ett sådant sätt att det skapas en array som innehåller 15 olika värden. Varje position i arrayen tillsätts ett förvalt värde. Det är något av dessa värden som skickas ut till Arduinon. Värdena skickas inte ut i rätt ordning utan slumpas med hjälp av en funktion som heter rand. Efter simulation utav slumpfunktionen konstaterades det att det krävdes 45 programkörningar innan alla värden hade skickats minst en gång. Efter varje programkörning fördröjs programmet dessutom med tre sekunder innan det skickas ett nytt värde. Detta för att säkerhetsställa att alla delsystem hinner köra igenom programmet. I tabell 5 nedan kan de slumpgenererade värdena följas.

**Tabell 5. Visar slumpgenererade värden från programkörning.**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **8** | *3* | **2** | *S* | **0** | *3* | **7** | *6* | **C** | *4* | **6** | *C* | **3** | *5* | **3** |
| **8** | *0* | **6** | *S* | **2** | *7* | **2** | *2* | **8** | *A* | **E** | *9* | **D** | *9* | **4** |
| **6** | *2* | **A** | *9* | **0** | *5* | **S** | *5* | **7** | *S* | **3** | *6* | **0** | *6* | **1** |

Det finns även i projektet implementerat en VGA-kamera som fungerar ihop med FPGA:n. Kameran fångar en bildsekvens och skickar via FPGA:ns VGA-port ut den till en extern bildskärm. Det var tänkt att kameran skulle användas till linjedetektering men valdes bort på grund utav tidsbrist och svårighetsgrad.

# 5. RESULTAT

Resultatet i detta projekt är själva konstruktionen av de enskilda delblocken men även sammansättning av de olika blocken för att erhålla en RC-bil som styrs via FPGA.

Projektet startade så att UART0 konstruerades i Vivado. Detta för att skapa ett sätt att överföra data/information mellan FPGA:n och Arduinon. Med hjälp utav UART0 portat till P-mod på Zedboardet så kunde FPGA:n kopplas samman med Arduinon via två kablar, en för Rx och den andra för Tx.

Utbytet utav data sker dock inte per automatik utan det krävs extern kontroll över RC-bilens komponenter så att de går och styra via seriell data. Det innebär att man med andra ord anpassar komponenten för att kommunicera på det sättet som efterfrågas.

För att åstadkomma detta så skrevs det olika programkoder för vardera komponent, när man kände sig nöjd så testades det via en förbindelse med datorn om det fungerade som tänkt. När alla behövliga komponenter kunde styras på det utvalda sättet kombinerades flertalet skrivna programkoder ihop till en och samma. Det här i sin tur skapade en grund för hur FPGA:n skulle kunna styra Arduinon. Eftersom data skulle skickas från FPGA:n så konstruerades programkoden på ett sådant sätt att den skickade ut data kontinuerligt. Arduinon i sin tur agerade mottagare och den programkoden väntade på inkommande data.

Då tanken var att det skulle implementeras linjedetektering via kamera så konstruerades det en förbindelse mellan FPGA:n och en VGA-kamera och den fungerar på så sätt att den ”fångar” upp en bildström, för att visa vad kameran ”fångat” så läggs hela bildströmmen ut på FPGA:ns VGA-port som kopplas in på en extern bildskärm. Kameran detekterar inga linjer då inga filter som krävs implementerats. En fullt fungerande yttre styrning av RC-bilen och dess komponenter via FPGA:n har uppnåtts. Utöver det har även kameran för framtida implementation av linjedetektering erhållits.

# 6. LÖSNINGSALTERNATIV

Projektet fungerar på det sätt att FPGA:n just nu slumpar ett antal värden och skickar ut de via UART0 på P-mod kontakten vidare till Arduinon. De slumpade värdena tolkas på olika sätt i Arduinokoden. Simuleringar som täcks är t.ex. om kameran skulle missa en linje och är på väg att åka utanför, då skall den bromsa ner och backa tillbaka där den kom ifrån. För att sedan hitta tillbaka till linjen så skall den svänga motsatt håll från vad den gjorde när den åkte utanför. Den skall om den känner att den är på väg att förlora den detekterade linjen bromsa in och svänga tills den hittat ett stabilt läge och sedan kan den accelerera upp igen. Skulle det vara så att den har på tok för hög hastighet och missar linjen helt så skall bromsa ner och sakta backa tillbaka. Eftersom detta bara är en simulering kring några tänkbara scenarion som kan ske så täcks inte alla möjligheter in.

Tanken med projektet var att det skulle användas en kamera till detektering. Kameran finns implementerad ihop med FPGA:n. Men för att man med hjälp utav kamera skall kunna detektera linjer så krävs det att man filtrerar bilden. Då det inte fanns tid eller kunskap att implementera dessa filter i VHDL som tanken var så läggs det upp som en alternativ lösning. Att implementera filter är något som kan göras på flera olika sätt, men med tanke på att LKA skall implementeras så finns det två stycken algoritmer som vanligen används. Dessa två är Sobel operator och Hough transform.

Man använder vanligen dessa två algoritmer ihop eftersom att de kompletterar varandra på ett sådant sätt att bilden och linjen blir mycket tydligare när de arbetar ihop. En teoretisk beskrivning kring hur dessa två filter är uppbyggda kommer i detta avsnitt att förklaras för att möjliggöra en implementation i framtiden. Implementation av dessa filter går och göra i olika programmeringsspråk men rekommenderat är att man använder sig utav C-kod då det finns mycket mer hjälp att tillgå.

Sobel filter eller sobel operator med andra ord används inom bildbehandling och datorvision. Användningsområdet är framför allt inom kantdetekteringsalgoritmer för att förtydliga kanter i bilden. Filtret är utvecklat och uppkallat efter Irwin Sobel och Gary Feldman, två kollegor vid Stanford Artificial Intelligence Laboratory (SAIL). Den första idén presenterades som ”Isotropic 3x3 Image Gradient Operator” år 1968. [26]

Det är tekniskt sett en diskret differentieringsoperator, vilken beräknar en approximation av en gradient för bildintensitetsfunktionen. Resultatet presenteras vid varje punkt i bilden som en motsvarande gradientfaktor eller som en norm för vektorn. Operatorn är baserad på att sammansätta bilden med ett litet separat heltalsbaserat filter i både horisontell och vertikal riktning. I och med detta så är det relativt billigt i form av beräkningar medan gradient approximationen vilken skapas är tämligen dålig. Detta särskilt i form av högfrekventa variationer som uppstår i bilden.

Sobel operatorn fungerar på så sätt att den använder två stycken 3x3 kärnor som sammanfogas ihop med originalbilden för att beräkna approximationen av derivatan, både för horisontella och vertikala ändringar. Vi börjar med att definiera källbilden som **KB** och de två andra bilderna som innehåller horisontella och vertikala approximationerna för $\mathbf{G_H}$ och $\mathbf{G_V}$. Då blir beräkningarna enligt följande:

$$\boldsymbol{G_H} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \boldsymbol{KB} \quad \textbf{(1)} \qquad \boldsymbol{G_V} = \begin{bmatrix} +1 & +2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \boldsymbol{KB} \ \textbf{(2)}$$

* Visar hur operationen ser ut för sammanfogandet av den tvådimensionella signalprocessen.

De båda sobel kärnorna kan brytas ner till två produkter en medelvärdes- och en differentieringskärna. Deras uppgift är att beräkna gradienten och skrivs om till en lite behagligare form för fortsättningen:

$$\boldsymbol{G_H} = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} [+1 \quad 0 \quad -1] \text{ (3)}$$

$$\boldsymbol{G_V} = \begin{bmatrix} +1 & +2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} [+1 \quad +2 \quad +1] \text{ (4)}$$

Ökningen för $\mathbf{G_H}$ är här definierad för att röra sig från vänster till höger riktning på det horisontella planet medan $\mathbf{G_V}$ rör sig uppifrån och ner. (i vertikala planet) Vid varje punkt i bilden så kan gradient approximationerna kombineras och då erhålles gradientens storlek. [26] Detta är något som görs med hjälp av:

$$G = \sqrt{{G_H}^2 + {G_V}^2} \text{ (5)}$$

Genom att nyttja informationen från $\mathbf{G_H}$ och $\mathbf{G_V}$ så kan gradientens riktning beräknas enligt följande:

$$\Theta = \arctan\left(\frac{G_V}{G_H}\right) \text{ (6)}$$

Θ är exempelvis 0 för en vertikal kant som är ljusare på höger sida.

Intensitetsfunktionen av en digital bild är känd som diskreta punkter, detta innebär att derivatan av funktionen inte kan bli definierad förrän det antagits att det finns en underliggande kontinuerlig intensitetsfunktion som samplas vid bildpunkterna. [27] Derivatan kan med hjälp av ytterligare antaganden beräknas som en funktion på den samplade intensitetsfunktionen som med andra ord är den digitala bilden. Vid en viss punkt på nästan alla bildpunkter så är derivatan en funktion av intensitetsvärdena. Uppskattningar utav derivatans funktioner kan i dessa fall definieras med större eller mindre noggrannhet.

Användning av Sobel operatorn ger oss en tämligen felaktig approximation av bildgradienten, detta är dock vid de flesta tillfällen av tillräckligt bra kvalitet. Detta på grund utav att operatorn endast använder sig utav 3 x 3 area runt varje bildpunkt. Genom att nyttja denna area skapas en approximation av motsvarande bildgradient, som endast använder sig utav heltalsvärden för koefficienterna. Dessa värden används i sin tur sedan för att framställa gradientens approximation.

Definitionen av Sobel operatorn medför att den endast kräver åtta bildpunkter kring en punkt för att beräkna motsvarande resultat. För beräkning utav gradientvektorns approximation krävs något så enkelt som heltalsräkning, i och med detta kan operatorn på ett enkelt sätt implementeras både i hårdvara och programvara.

När dessa två filter har implementerats så kan man i första hand använda det ihop med VGA-kameran. Skulle man inte vara nöjd med kvalitén på bilden så kan man även i framtiden implementera D8M-FMC kameran. Då får man en kvalité på upp till 30 FPS i 1080p istället för 30 FPS med VGA-upplösning. Kameran D8M-FMC kan implementeras på vilket som helst FPGA-kort som innehar en FMC-kontakt som följer samma VITA 57.1 standard som kameran.

# 7. SLUTSATSER OCH DISKUSSION

Projektet har varit väldigt intressant och lärorikt. Det har varit väldigt mycket nya saker man lärt sig, saker som man i skolan lärt sig grunden på har studerats och tillämpats på en mer detaljerad och djupare nivå. Syftet med detta arbete var att få till en självkörande RC-bil.(Linjedetektering via kamera) Detta lyckades tyvärr inte att uppnås helt och hållet. Det utav syftet som har lyckats uppnås är att styra en RC-bil via en FPGA. Tidsplanen som skapades i början av projektet kunde inte följas helt då arbetet varit mycket mer omfattande och tidskrävande än vad det tidigare varit beräknat. Istället har avgränsningar under arbetet gjorts och utvecklingsmöjligheter har lämnats kvar för företaget att bygga vidare på.

Då syftet inte uppnåtts så har det försökts skapa ett scenario där bilen skulle utföra handlingar så likt som möjligt som om den styrdes via kamera och linjedetektering. Då linjedetekteringen inte implementerades så lämnas det vid en framtida lösning. Detta med hjälp utav algoritmerna Hough transform och sobel operator. Till en början kan den VGA-kamera som redan finns användas vid implementationen utav linjedetektering för att sedan övergå till att implementera D8M-FMC som redan finns inköpt. Ytterligare en eventuell framtida lösning är att vidareutveckla linjedetekteringen så att den kan hantera körfält och inte bara linjer.

Under hela projektet så har det utförts test- och verifieringar av varje delmoment för att ha ”rätt” kontroll över RC-bilen. Detta arbetssätt har varit bra ur felsökningssynpunkt, då det efter varje moment har säkerställts att allting fungerar som det skall. Testerna har varit i form av olika mätningar på delsystemen för att se att informationen överförs vid t.ex kommunikationen mellan Arduino och FPGA.

Detta har medfört att företaget nu har en bra grund för framtida vidareutveckling och påbyggnad av projektet. Under projektets gång så har även planering varit en väldigt viktig beståndsdel samt att arbeta metodiskt och strukturera då det tidigt konstaterades att tidsramen inte kommer att vara tillräcklig för att kunna möta målen och syftet.

Frågeställningen angående hur kopplingen mellan FPGA och Arduino konstruerats i projektet har besvarats i kapitel 4. Tyvärr har inte LKA behandlats då beslut tidigt under projektet togs att tagits att inte implementera det på grund utav tidsbrist utan att lämna det för framtida utveckling.

# Referenser


[1] Wikipedia, "Arduino," Wikipedia, 01 08 2017. [Online]. Available: https://sv.wikipedia.org/wiki/Arduino. [Accessed 12 05 2017].

[2] Arduino, "Shields," Arduino, 2017. [Online]. Available: https://www.arduino.cc/en/Main/arduinoShields. [Accessed 13 05 2017].

[3] STMicroelectronics, "DUAL FULL-BRIDGE DRIVER," 2000. [Online]. Available: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf. [Accessed 14 05 2017].

[4] Wikipedia, "Field-programmable gate array," Wikipedia, 24 08 2017. [Online]. Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array. [Accessed 19 05 2017].

[5] Xilinx, "What is an FPGA?," Xilinx Inc., 2017. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Accessed 19 05 2017].

[6] Xilinx, "ZedBoard Zynq-7000 ARM/FPGA SoC Development Board," Xilinx Inc, 2017. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-elhabt.html. [Accessed 23 08 2017].

[7] Digilent, "Digilent Pmod™ Interface Specification," 20 11 2011. [Online]. Available: https://www.digilentinc.com/Pmods/Digilent-Pmod_%20Interface_Specification.pdf. [Accessed 19 08 2017].

[8] Xilinx, "I/O Design Flexibility with th FPGA Mezzanine Card (FMC)," 19 08 2009. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp315.pdf. [Accessed 20 08 2017].

[9] Terasic, "D8M-FMC User manual," 24 10 2016. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=1051&FID=352bb2b3fa3a763a071ca75a2c68e2ff. [Accessed 19 08 2017].

[10] ArduCAM, "CMOS OV7670 Camera Module," 05 2015. [Online]. Available: https://www.openhacks.com/uploadsproductos/ov7670_cmos_camera_module_revc_ds.pdf. [Accessed 29 06 2017].

[11] JIMB0, "Bi-Directional Logic Level Converter Hookup Guide," SparkFun, [Online]. Available: https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide. [Accessed 04 06 2017].

[12] T. Pro, "MG995," 2014. [Online]. Available: http://www.towerpro.com.tw/product/mg995/. [Accessed 20 08 2017].

[13] P. USA, "PL2303HX (Chip Rev D) USB to Serial Bridge Controller," Profilic USA, 2010 - 2012. [Online]. Available: http://prolificusa.com/portfolio/pl2303hx-rev-d-usb-to-serial-bridge-controller/. [Accessed 01 08 2017].

[14] Xilinx, "Vivado Design Suite Evaluation and WebPACK," Xilinx, 2017. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html. [Accessed 17 06 2017].

[15] Wikipedia, "Xilinx Vivado," Wikipedia, 22 06 2017. [Online]. Available: https://en.wikipedia.org/wiki/Xilinx_Vivado. [Accessed 20 06 2017].

[16] Xilinx, "Xilinx Software Development Kit (XSDK)," Xilinx Inc, 2017. [Online].

Available: https://www.xilinx.com/products/design-tools/embedded-software/sdk.html. [Accessed 21 06 2017].

[17] Arduino, "Arduino Software (IDE)," Arduino, 2017. [Online]. Available: https://www.arduino.cc/en/Guide/Environment. [Accessed 06 07 2017].

[18] Wikipedia, "PuTTY," Wikipedia, 08 07 2017. [Online]. Available: https://en.wikipedia.org/wiki/PuTTY. [Accessed 10 06 2017].

[19] d9.idv-tech.com, "d9 Tech Blog," d9.idv-tech.com , 22 03 2014. [Online]. Available: http://blog.idv-tech.com/2014/03/22/howto-export-zynq-peripheralsi2c-spi-uart-and-etc-to-pmod-connectors-of-zedboard-using-vivado-2013-4/. [Accessed 15 06 2017].

[20] Digilent, "ChipKIT PRO MX4 Board Reference Manual," Digilent, 2015. [Online]. Available: https://reference.digilentinc.com/chipkit_pro_mx4/refmanual. [Accessed 23 06 2017].

[21] Avnet, "ZedBoard," 27 01 2014. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf. [Accessed 25 07 2017].

[22] Ktnbn, "wikipedia," wikipedia, 04 12 2007. [Online]. Available: https://commons.wikimedia.org/wiki/File:Rs232_oscilloscope_trace.jpg. [Accessed 10 07 2017].

[23] T. Inc, "D8M-FMC," 25 10 2016. [Online]. Available: http://www.terasic.com/downloads/cd-rom/d8m-fmc/D8M-FMC_v.1.0.0_SystemCD.zip. [Accessed 30 05 2017].

[24] M. Field, "hamsterworks," hamsterworks, 25 07 2013. [Online]. Available: http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7670. [Accessed 20 07 2017].

[25] Ivica_Matic, "L298n Arduino Library," CC BY-NC-SA 2.5, 02 2017. [Online]. Available: http://www.instructables.com/id/L298n-Arduino-Library/. [Accessed 02 07 2017].

[26] Wikipedia, "Sobel operator," Wikipedia, 08 06 2017. [Online]. Available: https://en.wikipedia.org/wiki/Sobel_operator. [Accessed 07 08 2017].

[27] Tutorialspoint, "Sobel operator," Tutorialspoint, 2017. [Online]. Available: https://www.tutorialspoint.com/dip/sobel_operator.htm. [Accessed 05 08 2017].

# BILAGA A. Programkod för ov7670_capture.vhd

I programkoden så finns det ett block som heter ov7670_capture och är det block som undersöker signalerna HREF och VSYNC som finns på kameran. OV7670 använder sig utav ett 16 bitars RGB pixelvärde, halva delen utav detta värde är dock det som behandlas under en klockpuls. Blocket fungerar på så sätt att det låser halva värdet som skickats och kombinerar sedan ihop två halverade värden till ett 12 bitars RGB pixelvärde. Detta värde i sin tur sparas i FPGA:ns block RAM.

```
----------------------------------------------------------------------------
-------
-- Engineer: Mike Field <hamster@snap.net.nz>
--
-- Description: Captures the pixels coming from the OV7670 camera and
--              Stores them in block RAM
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ov7670_capture is
    Port ( pclk  : in  STD_LOGIC;
           vsync : in  STD_LOGIC;
           href  : in  STD_LOGIC;
           d     : in  STD_LOGIC_VECTOR (7 downto 0);
           addr  : out STD_LOGIC_VECTOR (18 downto 0);
           dout  : out STD_LOGIC_VECTOR (11 downto 0);
           we    : out STD_LOGIC);
end ov7670_capture;

architecture Behavioral of ov7670_capture is
   signal d_latch      : std_logic_vector(15 downto 0) := (others => '0');
   signal address      : STD_LOGIC_VECTOR(18 downto 0) := (others => '0');
   signal address_next : STD_LOGIC_VECTOR(18 downto 0) := (others => '0');
   signal wr_hold      : std_logic_vector(1 downto 0)  := (others => '0');

begin
   addr <= address;
   process(pclk)
   begin
      if rising_edge(pclk) then
         -- This is a bit tricky href starts a pixel transfer that takes 3
cycles
         --        Input   | state after clock tick
         --         href   |
wr_hold    d_latch           d                 we address  address_next
         -- cycle -
1  x    |    xx      xxxxxxxxxxxxxxxx  xxxxxxxxxxxxxxxx  x   xxxx     xxxx
         -- cycle
0   1    |    x1      xxxxxxxxRRRRRGGG  xxxxxxxxxxxxxxxx  x   xxxx     addr
         -- cycle
1   0    |    10      RRRRRGGGGGGBBBBB  xxxxxxxxRRRRRGGG  x   addr     addr
         -- cycle
2   x    |    0x      GGGBBBBBxxxxxxxx  RRRRRGGGGGGBBBBB  1   addr     addr
+1

         if vsync = '1' then
```

```
            address <= (others => '0');
            address_next <= (others => '0');
            wr_hold <= (others => '0');
         else
            dout    <= d_latch(10 downto 7) & d_latch(15 downto 12) &
d_latch(4 downto 1);
            address <= address_next;
            we      <= wr_hold(1);
            wr_hold <= wr_hold(0) & (href and not wr_hold(0));
            d_latch <= d_latch( 7 downto  0) & d;

            if wr_hold(1) = '1' then
               address_next <= std_logic_vector(unsigned(address_next)+1);
            end if;

         end if;
      end if;
   end process;
end Behavioral;
```

## BILAGA A.1. Programkod för ov7670_controller.vhd

Nästa block är ett sammansatt block som består utav två olika som i VHDL designats. Dessa komponenter är i2c_sender och ov7670_registers och används/anropas i ov7670_controller som det sammansatta blocket kallas. Detta block sköter samarbetet mellan de olika komponenterna.

```
----------------------------------------------------------------------------
-------
-- Engineer: Mike Field <hamster@snap.net.nz>
--
-- Description: Controller for the OV760 camera - transferes registers to
-- the  camera over an I2C like bus
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ov7670_controller is
    Port ( clk   : in    STD_LOGIC;
        resend :in    STD_LOGIC;
        config_finished : out std_logic;
           sioc  : out   STD_LOGIC;
           siod  : inout STD_LOGIC;
           reset : out   STD_LOGIC;
           pwdn  : out   STD_LOGIC;
        xclk  : out   STD_LOGIC
);
end ov7670_controller;

architecture Behavioral of ov7670_controller is
  COMPONENT ov7670_registers
  PORT(
    clk      : IN std_logic;
    advance  : IN std_logic;
    resend   : in STD_LOGIC;
    command  : OUT std_logic_vector(15 downto 0);
    finished : OUT std_logic
```

```
    );
  END COMPONENT;

  COMPONENT i2c_sender
  PORT(
    clk   : IN std_logic;
    send  : IN std_logic;
    taken : out std_logic;
    id    : IN std_logic_vector(7 downto 0);
    reg   : IN std_logic_vector(7 downto 0);
    value : IN std_logic_vector(7 downto 0);
    siod  : INOUT std_logic;
    sioc  : OUT std_logic
    );
  END COMPONENT;

  signal sys_clk  : std_logic := '0';
  signal command  : std_logic_vector(15 downto 0);
  signal finished : std_logic := '0';
  signal taken    : std_logic := '0';
  signal send     : std_logic;

  constant camera_address : std_logic_vector(7 downto 0) := x"42"; -- 42";
-- Device write ID - see top of page 11 of data sheet
begin
   config_finished <= finished;

  send <= not finished;
  Inst_i2c_sender: i2c_sender PORT MAP(
    clk   => clk,
    taken => taken,
    siod  => siod,
    sioc  => sioc,
    send  => send,
    id    => camera_address,
    reg   => command(15 downto 8),
    value => command(7 downto 0)
  );

  reset <= '1';             -- Normal mode
  pwdn  <= '0';             -- Power device up
  xclk  <= sys_clk;

  Inst_ov7670_registers: ov7670_registers PORT MAP(
    clk      => clk,
    advance  => taken,
    command  => command,
    finished => finished,
    resend   => resend
  );

  process(clk)
  begin
    if rising_edge(clk) then
      sys_clk <= not sys_clk;
    end if;
  end process;
end Behavioral;
```

# BILAGA A.2. Programkod för i2c_sender.vhd

I2c_sender är det som sätter upp ett protokoll som används för att ställa in kameraparametrar. Protokollet SCCB som OV760 använder sig utav är i sin tur ett i2c kompatibelt gränssnitt som koden nyttjar.

```
----------------------------------------------------------------------------
-------
-- Engineer: <mfield@concepts.co.nz
--
-- Description: Send the commands to the OV7670 over an I2C-like interface
--
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity i2c_sender is
    Port ( clk   : in  STD_LOGIC;
           siod  : inout  STD_LOGIC;
           sioc  : out  STD_LOGIC;
        taken : out  STD_LOGIC;
        send  : in  STD_LOGIC;
           id    : in  STD_LOGIC_VECTOR (7 downto 0);
           reg   : in  STD_LOGIC_VECTOR (7 downto 0);
           value : in  STD_LOGIC_VECTOR (7 downto 0));
end i2c_sender;

architecture Behavioral of i2c_sender is
  signal   divider  : unsigned (7 downto 0) := "00000001"; -- this value
gives a 254 cycle pause before the initial frame is sent
  signal   busy_sr  : std_logic_vector(31 downto 0) := (others => '0');
  signal   data_sr  : std_logic_vector(31 downto 0) := (others => '1');
begin
  process(busy_sr, data_sr(31))
  begin
    if busy_sr(11 downto 10) = "10" or
       busy_sr(20 downto 19) = "10" or
       busy_sr(29 downto 28) = "10"  then
      siod <= 'Z';
    else
      siod <= data_sr(31);
    end if;
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
      taken <= '0';
      if busy_sr(31) = '0' then
        SIOC <= '1';
        if send = '1' then
          if divider = "00000000" then
         data_sr <= "100" &   id & '0'  &   reg & '0' & value & '0' & "01";
         busy_sr <= "111" & "111111111" & "111111111" & "111111111" & "11";
            taken <= '1';
          else
            divider <= divider+1; -- this only happens on powerup
          end if;
        end if;
      else
```

```
        case busy_sr(32-1 downto 32-3) & busy_sr(2 downto 0) is
          when "111"&"111" => -- start seq #1
            case divider(7 downto 6) is
              when "00"   => SIOC <= '1';
              when "01"   => SIOC <= '1';
              when "10"   => SIOC <= '1';
              when others => SIOC <= '1';
            end case;
          when "111"&"110" => -- start seq #2
            case divider(7 downto 6) is
              when "00"   => SIOC <= '1';
              when "01"   => SIOC <= '1';
              when "10"   => SIOC <= '1';
              when others => SIOC <= '1';
            end case;
          when "111"&"100" => -- start seq #3
            case divider(7 downto 6) is
              when "00"   => SIOC <= '0';
              when "01"   => SIOC <= '0';
              when "10"   => SIOC <= '0';
              when others => SIOC <= '0';
            end case;
          when "110"&"000" => -- end seq #1
            case divider(7 downto 6) is
              when "00"   => SIOC <= '0';
              when "01"   => SIOC <= '1';
              when "10"   => SIOC <= '1';
              when others => SIOC <= '1';
            end case;
          when "100"&"000" => -- end seq #2
            case divider(7 downto 6) is
              when "00"   => SIOC <= '1';
              when "01"   => SIOC <= '1';
              when "10"   => SIOC <= '1';
              when others => SIOC <= '1';
            end case;
          when "000"&"000" => -- Idle
            case divider(7 downto 6) is
              when "00"   => SIOC <= '1';
              when "01"   => SIOC <= '1';
              when "10"   => SIOC <= '1';
              when others => SIOC <= '1';
            end case;
          when others      =>
            case divider(7 downto 6) is
              when "00"   => SIOC <= '0';
              when "01"   => SIOC <= '1';
              when "10"   => SIOC <= '1';
              when others => SIOC <= '0';
            end case;
        end case;

        if divider = "11111111" then
          busy_sr <= busy_sr(32-2 downto 0) & '0';
          data_sr <= data_sr(32-2 downto 0) & '1';
          divider <= (others => '0');
        else
          divider <= divider+1;
        end if;
      end if;
```

```
    end if;
  end process;
end Behavioral;
```

## BILAGA A.3. Programkod för ov7670_registers.vhd

Ov7670_registers å sin sida är det block som sätter upp inställningarna för kamerans register.

```
----------------------------------------------------------------------------
-- Company:
-- Engineer: Mike Field <hamster@sanp.net.nz>
--
-- Description: Register settings for the OV7670 Caamera (partially from
OV7670.c
--              in the Linux Kernel
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ov7670_registers is
    Port ( clk      : in  STD_LOGIC;
           resend   : in  STD_LOGIC;
           advance  : in  STD_LOGIC;
           command  : out  std_logic_vector(15 downto 0);
           finished : out  STD_LOGIC);
end ov7670_registers;

architecture Behavioral of ov7670_registers is
  signal sreg    : std_logic_vector(15 downto 0);
  signal address : std_logic_vector(7 downto 0) := (others => '0');
begin
  command <= sreg;
  with sreg select finished  <= '1' when x"FFFF", '0' when others;

  process(clk)
  begin
    if rising_edge(clk) then
      if resend = '1' then
        address <= (others => '0');
      elsif advance = '1' then
        address <= std_logic_vector(unsigned(address)+1);
      end if;

      case address is
        when x"00" => sreg <= x"1280"; -- COM7   Reset
        when x"01" => sreg <= x"1280"; -- COM7   Reset
        when x"02" => sreg <= x"1204"; -- COM7   Size & RGB output
        when x"03" => sreg <= x"1100"; -- CLKRC  Prescaler - Fin/(1+1)
        when x"04" => sreg <= x"0C00"; -- COM3   Lots of stuff, enable
scaling, all others off
        when x"05" => sreg <= x"3E00"; -- COM14  PCLK scaling off

        when x"06" => sreg <= x"8C00"; -- RGB444 Set RGB format
        when x"07" => sreg <= x"0400"; -- COM1   no CCIR601
        when x"08" => sreg <= x"4010"; -- COM15  Full 0-255 output, RGB 565
        when x"09" => sreg <= x"3a04"; -- TSLB   Set UV ordering,  do not
auto-reset window
        when x"0A" => sreg <= x"1438"; -- COM9  - AGC Celling
        when x"0B" => sreg <= x"4fb3"; -- MTX1  - colour conversion matrix
```

```
        when x"0C" => sreg <= x"50b3"; -- MTX2  - colour conversion matrix
        when x"0D" => sreg <= x"5100"; -- MTX3  - colour conversion matrix
        when x"0E" => sreg <= x"523d"; -- MTX4  - colour conversion matrix
        when x"0F" => sreg <= x"53a7"; -- MTX5  - colour conversion matrix
        when x"10" => sreg <= x"54e4"; -- MTX6  - colour conversion matrix
        when x"11" => sreg <= x"589e"; -- MTXS  - Matrix sign and auto
contrast
        when x"12" => sreg <= x"3dc0"; -- COM13 - Turn on GAMMA and UV Auto
adjust
        when x"13" => sreg <= x"1100"; -- CLKRC  Prescaler - Fin/(1+1)

        when x"14" => sreg <= x"1711"; -- HSTART HREF start (high 8 bits)
        when x"15" => sreg <= x"1861"; -- HSTOP  HREF stop (high 8 bits)
        when x"16" => sreg <= x"32A4"; -- HREF   Edge offset and low 3 bits
of HSTART and HSTOP

        when x"17" => sreg <= x"1903"; -- VSTART VSYNC start (high 8 bits)
        when x"18" => sreg <= x"1A7b"; -- VSTOP  VSYNC stop (high 8 bits)
        when x"19" => sreg <= x"030a"; -- VREF   VSYNC low two bits

--         when x"10" => sreg <= x"703a"; -- SCALING_XSC
--         when x"11" => sreg <= x"7135"; -- SCALING_YSC
--         when x"12" => sreg <= x"7200"; -- SCALING_DCWCTR  -- zzz was 11
--         when x"13" => sreg <= x"7300"; -- SCALING_PCLK_DIV
--         when x"14" => sreg <= x"a200"; -- SCALING_PCLK_DELAY must match
COM14
--         when x"15" => sreg <= x"1500"; -- COM10 Use HREF not hSYNC
--
--         when x"1D" => sreg <= x"B104"; -- ABLC1 - Turn on auto black
level
--         when x"1F" => sreg <= x"138F"; -- COM8  - AGC, White balance
--         when x"21" => sreg <= x"FFFF"; -- spare
--         when x"22" => sreg <= x"FFFF"; -- spare
--         when x"23" => sreg <= x"0000"; -- spare
--         when x"24" => sreg <= x"0000"; -- spare
--         when x"25" => sreg <= x"138F"; -- COM8 - AGC, White balance
--         when x"26" => sreg <= x"0000"; -- spare
--         when x"27" => sreg <= x"1000"; -- AECH Exposure
--         when x"28" => sreg <= x"0D40"; -- COMM4 - Window Size
--         when x"29" => sreg <= x"0000"; -- spare
--         when x"2a" => sreg <= x"a505"; -- AECGMAX banding filter step
--         when x"2b" => sreg <= x"2495"; -- AEW AGC Stable upper limite
--         when x"2c" => sreg <= x"2533"; -- AEB AGC Stable lower limi
--         when x"2d" => sreg <= x"26e3"; -- VPT AGC fast mode limits
--         when x"2e" => sreg <= x"9f78"; -- HRL High reference level
--         when x"2f" => sreg <= x"A068"; -- LRL low reference level
--         when x"30" => sreg <= x"a103"; -- DSPC3 DSP control
--         when x"31" => sreg <= x"A6d8"; -- LPH Lower Prob High
--         when x"32" => sreg <= x"A7d8"; -- UPL Upper Prob Low
--         when x"33" => sreg <= x"A8f0"; -- TPL Total Prob Low
--         when x"34" => sreg <= x"A990"; -- TPH Total Prob High
--         when x"35" => sreg <= x"AA94"; -- NALG AEC Algo select
--         when x"36" => sreg <= x"13E5"; -- COM8 AGC Settings
        when others => sreg <= x"ffff";
      end case;
    end if;
  end process;
end Behavioral;
```

# BILAGA A.4. Programkod för vga.vhd

Ytterligare ett block som används är vga och det är det som genererar VSYNC OCH HSYNC signaler för videoutgången vilket möjliggör att bilden skickas ut på FPGA-kortets VGA-utgång.

```
----------------------------------------------------------------------------
-- Engineer: Mike Field <hamster@snap.net.nz>
--
-- Description: Generate analog 640x480 VGA, double-doublescanned from
19200 bytes of RAM
--
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity vga is
    Port (
    clk25       : in  STD_LOGIC;
    vga_red     : out STD_LOGIC_VECTOR(3 downto 0);
    vga_green   : out STD_LOGIC_VECTOR(3 downto 0);
    vga_blue    : out STD_LOGIC_VECTOR(3 downto 0);
    vga_hsync   : out STD_LOGIC;
    vga_vsync   : out STD_LOGIC;
    frame_addr  : out STD_LOGIC_VECTOR(18 downto 0);
    frame_pixel : in  STD_LOGIC_VECTOR(11 downto 0)
   );
end vga;

architecture Behavioral of vga is
   -- Timing constants
   constant hRez       : natural := 640;
   constant hStartSync : natural := 640+16;
   constant hEndSync   : natural := 640+16+96;
   constant hMaxCount  : natural := 800;

   constant vRez       : natural := 480;
   constant vStartSync : natural := 480+10;
   constant vEndSync   : natural := 480+10+2;
   constant vMaxCount  : natural := 480+10+2+33;

  constant hsync_active : std_logic := '0';
  constant vsync_active : std_logic := '0';

   signal hCounter : unsigned( 9 downto 0) := (others => '0');
   signal vCounter : unsigned( 9 downto 0) := (others => '0');
  signal address  : unsigned(18 downto 0) := (others => '0');
  signal blank    : std_logic := '1';

begin
  frame_addr <= std_logic_vector(address);

   process(clk25)
   begin
    if rising_edge(clk25) then
      -- Count the lines and rows
      if hCounter = hMaxCount-1 then
        hCounter <= (others => '0');
        if vCounter = vMaxCount-1 then
          vCounter <= (others => '0');
```

```
        else
          vCounter <= vCounter+1;
        end if;
      else
        hCounter <= hCounter+1;
      end if;

      if blank = '0' then
        vga_red   <= frame_pixel(11 downto 8);
        vga_green <= frame_pixel( 7 downto 4);
        vga_blue  <= frame_pixel( 3 downto 0);
      else
        vga_red   <= (others => '0');
        vga_green <= (others => '0');
        vga_blue  <= (others => '0');
      end if;

      if vCounter  >= vRez then
        address <= (others => '0');
        blank <= '1';
      else
        if hCounter  < 640 then
          blank <= '0';
          address <= address+1;
        else
          blank <= '1';
        end if;
      end if;

      -- Are we in the hSync pulse? (one has been added to include
frame_buffer_latency)
      if hCounter > hStartSync and hCounter <= hEndSync then
        vga_hSync <= hsync_active;
      else
        vga_hSync <= not hsync_active;
      end if;

      -- Are we in the vSync pulse?
      if vCounter >= vStartSync and vCounter < vEndSync then
        vga_vSync <= vsync_active;
      else
        vga_vSync <= not vsync_active;
      end if;
    end if;
  end process;
end Behavioral;
```

## BILAGA A.5. Programkod för clocking.vhd

En till kodsnutt som finns är clocking, det är den som sköter och fixar till de klockor som behövs. Det används olika frekvenser som 25Mhz, 50Mhz samt 100Mhz. Clocking är en komponent i programmet som används, d.v.s. programkoden genereras i programmet och är inte något som man skriver själv.

library ieee;

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

library unisim;
```

```
use unisim.vcomponents.all;

entity clocking is
port
 (-- Clock in ports
  CLK_100           : in     std_logic;
  -- Clock out ports
  CLK_50          : out    std_logic;
  CLK_25          : out    std_logic
 );
end clocking;

architecture xilinx of clocking is
  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of xilinx :

architecture is
  -- Input clock buffering / unused connectors
  signal clkin1      : std_logic;
  -- Output clock buffering / unused connectors
  signal clkfbout         : std_logic;
  signal clkfbout_buf     : std_logic;
  signal clkfboutb_unused : std_logic;
  signal clkout0          : std_logic;
  signal clkout0b_unused  : std_logic;
  signal clkout1          : std_logic;
  signal clkout1b_unused  : std_logic;
  signal clkout2_unused   : std_logic;
  signal clkout2b_unused  : std_logic;
  signal clkout3_unused   : std_logic;
  signal clkout3b_unused  : std_logic;
  signal clkout4_unused   : std_logic;
  signal clkout5_unused   : std_logic;
  signal clkout6_unused   : std_logic;
  -- Dynamic programming unused signals
  signal do_unused        : std_logic_vector(15 downto 0);
  signal drdy_unused      : std_logic;
  -- Dynamic phase shift unused signals
  signal psdone_unused    : std_logic;
  -- Unused status signals
  signal locked_unused    : std_logic;
  signal clkfbstopped_unused : std_logic;
  signal clkinstopped_unused : std_logic;
begin


  -- Input buffering
  -------------------------------------
  clkin1_buf : IBUFG
  port map
   (O => clkin1,
    I => CLK_100);


  -- Clocking primitive
  --------------------------------------
  -- Instantiation of the MMCM primitive
  --    * Unused inputs are tied off
  --    * Unused outputs are labeled unused
  mmcm_adv_inst : MMCME2_ADV
  generic map
```

```
    (BANDWIDTH            => "OPTIMIZED",
     CLKOUT4_CASCADE      => FALSE,
     COMPENSATION         => "ZHOLD",
     STARTUP_WAIT         => FALSE,
     DIVCLK_DIVIDE        => 1,
     CLKFBOUT_MULT_F      => 10.000,
     CLKFBOUT_PHASE       => 0.000,
     CLKFBOUT_USE_FINE_PS => FALSE,
     CLKOUT0_DIVIDE_F     => 20.000,
     CLKOUT0_PHASE        => 0.000,
     CLKOUT0_DUTY_CYCLE   => 0.500,
     CLKOUT0_USE_FINE_PS  => FALSE,
     CLKOUT1_DIVIDE       => 40,
     CLKOUT1_PHASE        => 0.000,
     CLKOUT1_DUTY_CYCLE   => 0.500,
     CLKOUT1_USE_FINE_PS  => FALSE,
     CLKIN1_PERIOD        => 10.000,
     REF_JITTER1          => 0.010)
   port map
     -- Output clocks
    (CLKFBOUT             => clkfbout,
     CLKFBOUTB            => clkfboutb_unused,
     CLKOUT0              => clkout0,
     CLKOUT0B             => clkout0b_unused,
     CLKOUT1              => clkout1,
     CLKOUT1B             => clkout1b_unused,
     CLKOUT2              => clkout2_unused,
     CLKOUT2B             => clkout2b_unused,
     CLKOUT3              => clkout3_unused,
     CLKOUT3B             => clkout3b_unused,
     CLKOUT4              => clkout4_unused,
     CLKOUT5              => clkout5_unused,
     CLKOUT6              => clkout6_unused,
     -- Input clock control
     CLKFBIN              => clkfbout_buf,
     CLKIN1               => clkin1,
     CLKIN2               => '0',
     -- Tied to always select the primary input clock
     CLKINSEL             => '1',
     -- Ports for dynamic reconfiguration
     DADDR                => (others => '0'),
     DCLK                 => '0',
     DEN                  => '0',
     DI                   => (others => '0'),
     DO                   => do_unused,
     DRDY                 => drdy_unused,
     DWE                  => '0',
     -- Ports for dynamic phase shift
     PSCLK                => '0',
     PSEN                 => '0',
     PSINCDEC             => '0',
     PSDONE               => psdone_unused,
     -- Other control and status signals
     LOCKED               => locked_unused,
     CLKINSTOPPED         => clkinstopped_unused,
     CLKFBSTOPPED         => clkfbstopped_unused,
     PWRDWN               => '0',
     RST                  => '0');

   -- Output buffering
   -------------------------------------
```

```
   clkf_buf : BUFG
   port map
    (O => clkfbout_buf,
     I => clkfbout);


   clkout1_buf : BUFG
   port map
    (O   => CLK_50,
     I   => clkout0);


   clkout2_buf : BUFG
   port map
    (O   => CLK_25,
     I   => clkout1);

end xilinx;
```

## BILAGA A.6. Programkod för ov7670_top.vhd

Huvudblocket i programkoden är ov7670_top och är det som ligger högst upp i hierarkin. Det är en komponent där deklaration och definitioner utav underliggande blocks portar finns

```
----------------------------------------------------------------------------
-- Engineer: Mike Field <hamster@snap.net.nz>
--
-- Description: Top level for the OV7670 camera project.
----------------------------------------------------------------------------
-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

Library UNISIM;
use UNISIM.vcomponents.all;

entity ov7670_top is
    Port (
    clk100        : in    STD_LOGIC;
    OV7670_SIOC   : out   STD_LOGIC;
    OV7670_SIOD   : inout STD_LOGIC;
    OV7670_RESET  : out   STD_LOGIC;
    OV7670_PWDN   : out   STD_LOGIC;
    OV7670_VSYNC  : in    STD_LOGIC;
    OV7670_HREF   : in    STD_LOGIC;
    OV7670_PCLK   : in    STD_LOGIC;
    OV7670_XCLK   : out   STD_LOGIC;
    OV7670_D      : in    STD_LOGIC_VECTOR(7 downto 0);

    LED           : out    STD_LOGIC_VECTOR(7 downto 0);

    vga_red       : out   STD_LOGIC_VECTOR(3 downto 0);
    vga_green     : out   STD_LOGIC_VECTOR(3 downto 0);
    vga_blue      : out   STD_LOGIC_VECTOR(3 downto 0);
    vga_hsync     : out   STD_LOGIC;
    vga_vsync     : out   STD_LOGIC;

    btn          : in    STD_LOGIC
   );
```

```
end ov7670_top;

architecture Behavioral of ov7670_top is

  COMPONENT debounce
  PORT(
    clk : IN std_logic;
    i : IN std_logic;
    o : OUT std_logic
    );
  END COMPONENT;

component clocking
port
 (-- Clock in ports
  CLK_100           : in     std_logic;
  -- Clock out ports
  CLK_50          : out    std_logic;
  CLK_25          : out    std_logic
 );
end component;

  COMPONENT ov7670_controller
  PORT(
    clk   : IN    std_logic;
    resend: IN    std_logic;
    config_finished : out std_logic;
    siod  : INOUT std_logic;
    sioc  : OUT   std_logic;
    reset : OUT   std_logic;
    pwdn  : OUT   std_logic;
    xclk  : OUT   std_logic
    );
  END COMPONENT;

  COMPONENT frame_buffer
  PORT (
    clka  : IN  STD_LOGIC;
    wea   : IN  STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN  STD_LOGIC_VECTOR(18 DOWNTO 0);
    dina  : IN  STD_LOGIC_VECTOR(11 DOWNTO 0);
    clkb  : IN  STD_LOGIC;
    addrb : IN  STD_LOGIC_VECTOR(18 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
  );
  END COMPONENT;

  COMPONENT ov7670_capture
  PORT(
    pclk : IN std_logic;
    vsync : IN std_logic;
    href  : IN std_logic;
    d     : IN std_logic_vector(7 downto 0);
    addr  : OUT std_logic_vector(18 downto 0);
    dout  : OUT std_logic_vector(11 downto 0);
    we    : OUT std_logic
    );
  END COMPONENT;


  COMPONENT vga
```

```
  PORT(
    clk25     : IN std_logic;
    vga_red   : OUT std_logic_vector(3 downto 0);
    vga_green : OUT std_logic_vector(3 downto 0);
    vga_blue  : OUT std_logic_vector(3 downto 0);
    vga_hsync : OUT std_logic;
    vga_vsync : OUT std_logic;

    frame_addr  : OUT std_logic_vector(18 downto 0);
    frame_pixel : IN  std_logic_vector(11 downto 0)
    );
  END COMPONENT;

  signal frame_addr      : std_logic_vector(18 downto 0);
  signal frame_pixel     : std_logic_vector(11 downto 0);

  signal capture_addr    : std_logic_vector(18 downto 0);
  signal capture_data    : std_logic_vector(11 downto 0);
   signal capture_we      : std_logic_vector(0 downto 0);
  signal resend          : std_logic;
  signal config_finished : std_logic;

  signal clk_feedback  : std_logic;
  signal clk50u        : std_logic;
  signal clk50         : std_logic;
  signal clk25u        : std_logic;
  signal clk25         : std_logic;
  signal buffered_pclk : std_logic;

begin

btn_debounce: debounce PORT MAP(
    clk => clk50,
    i   => btn,
    o   => resend
  );

  Inst_vga: vga PORT MAP(
    clk25       => clk25,
    vga_red     => vga_red,
    vga_green   => vga_green,
    vga_blue    => vga_blue,
    vga_hsync   => vga_hsync,
    vga_vsync   => vga_vsync,
    frame_addr  => frame_addr,
    frame_pixel => frame_pixel
  );

fb : frame_buffer
  PORT MAP (
    clka  => OV7670_PCLK,
    wea   => capture_we,
    addra => capture_addr,
    dina  => capture_data,

    clkb  => clk50,
    addrb => frame_addr,
    doutb => frame_pixel
  );

  led <= "0000000" & config_finished;
```

```
capture: ov7670_capture PORT MAP(
    pclk  => OV7670_PCLK,
    vsync => OV7670_VSYNC,
    href  => OV7670_HREF,
    d     => OV7670_D,
    addr  => capture_addr,
    dout  => capture_data,
    we    => capture_we(0)
  );

controller: ov7670_controller PORT MAP(
    clk   => clk50,
    sioc  => ov7670_sioc,
    resend => resend,
    config_finished => config_finished,
    siod  => ov7670_siod,
    pwdn  => OV7670_PWDN,
    reset => OV7670_RESET,
    xclk  => OV7670_XCLK
  );

your_instance_name : clocking
  port map
   (-- Clock in ports
    CLK_100 => CLK100,
    -- Clock out ports
    CLK_50 => CLK50,
    CLK_25 => CLK25);

end Behavioral;
```

# BILAGA B1. Programkod för Arduino.

```
#include "l298n_lib.h"
#include <SoftwareSerial.h>
#include <Servo.h>

// software serial #1: RX = digital pin 10, TX = digital pin 11
SoftwareSerial portOne(10, 11);
int inByte;
Servo servo1;


void setup() {  // Open serial communications and wait for port to
open:
  set_pins(2, 3, 4, 5, 6, 9); // Set pinconfiguration for IN1, IN2, IN3,
IN4, ENA, ENB
   pinMode(1,OUTPUT);
  servo1.attach(14); //Attach servo1 to analog pin 0
  servo1.write(0); // Calibrate servo1, set position to 0°
  delay(1000);
  servo1.write(90); // Calibrate servo1, set position to 90°

  Serial.begin(9600); // Opens serial with the data rate of 9600bps.
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port
only
  }

  // Start each software serial port
  portOne.begin(9600);
}

void loop() {
   portOne.listen(); // Listen on portOne

  while (portOne.available() != 0) {  // if serial data is
available,
write the read value from portOne to char inByte.
    char inByte = portOne.read();
    Serial.write(inByte);

    Serial.println();  // Print the received value on the Arduino
terminal
    delay(1000); // Delay of 1 sec

  switch (inByte){  // Depending on which value is received, do the
right case

    case '0' : {
      lmot('F',64); // Left motor A rotate forward with the speed of
64 rpm
      dmot('F',127); // Right motor B rotate forward with the speed
of 127 rpm
     servo1.write(60);
      break;
```

```
    }
    case '1' : {
      lmot('F',127); // F stands for forward
      dmot('F',191);
     servo1.write(70);
      break;
      }

    case '2' : {
      lmot('F',191);
      dmot('F',255);
    servo1.write(80);
      break;
      }

    case '3' :{
      lmot ('F',255);
      dmot ('F',255);
     servo1.write(90);
      break;
      }

    case '4':{
      lmot ('F',255);
      dmot ('F',191);
     servo1.write(100);
     break;
       }

    case '5':{
      lmot ('F',191);
      dmot ('F',127);
     servo1.write(110);
      break;
      }
    case '6':{
      dmot( 'F',127);
      lmot( 'F',64);
     servo1.write(120);
      break;
      }
    case '7':{
      lmot ('B',100); // Left motor rotate backwards with the speed
of 100 rpm
      dmot ('B',50); // Right motor rotate backwards with the speed
of 50 rpm
   servo1.write(120);
      break;
      }
    case '8':{
      lmot ('B',144); // B stand for backwards
      dmot ('B',100);
   servo1.write(110);
      break;
      }
     case '9':{
```

```
        lmot ('B',191);
        dmot ('B',144);
    servo1.write(100);
        break;
        }
      case 'A':{
        lmot ('B',191);
        dmot ('B',191);
    servo1.write(90);
        break;
        }
      case 'B':{
        lmot ('B',144);
        dmot ('B',191);
    servo1.write(80);
        break;
        }
      case 'C':{
        lmot ('B',100);
        dmot ('B',144);
    servo1.write(70);
        break;
        }
      case 'D':{
        lmot ('B',50);
        dmot ('B',100);
    servo1.write(60);
        break;
        }
     case 'S' :{
        dmot_stop();  // Right motor stop
        lmot_stop(); // Left motor stop
        }
     }

   }

}
```

# BILAGA B2. Programkod för FPGA.

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <stdlib.h>
#include <time.h>


int main()
{
  init_platform();
    print("\n\rProgram start\n\r");
    int i = 0; // Skapa en integer som används till en räknare

//Array med alla datadresser som senare slumpas
const char* array[15];

array[0] = '0';  // Position 0 innehar värdet 0
array[1] = '1';
array[2] = '2';
```

```
array[3] = '3';
array[4] = '4';
array[5] = '5';
array[6] = '6';
array[7] = '7';
array[8] = '8';
array[9] = '9';
array[10] = 'A';
array[11] = 'C';
array[12] = 'D';
array[13] = 'E';
array[14] = 'S';

    for (int i = 0; i < 45; i++ ) { //Körs 45 gånger för att få med alla värden minst en
gång
    char r = array[rand() % 15]; //Funktion som rör om värdena
      printf("%s\n", r);
      XUartPs_SendByte(0xe0000000, r); // Skicka ett slumpmässigt värde till UART0
      sleep(3); // Vänta 3 sek innan nästa värde skickas igen

    };

    print("\n\rProgram end\n\r");

    cleanup_platform();
    return 0;
}
```