Skriftlig Redovisning

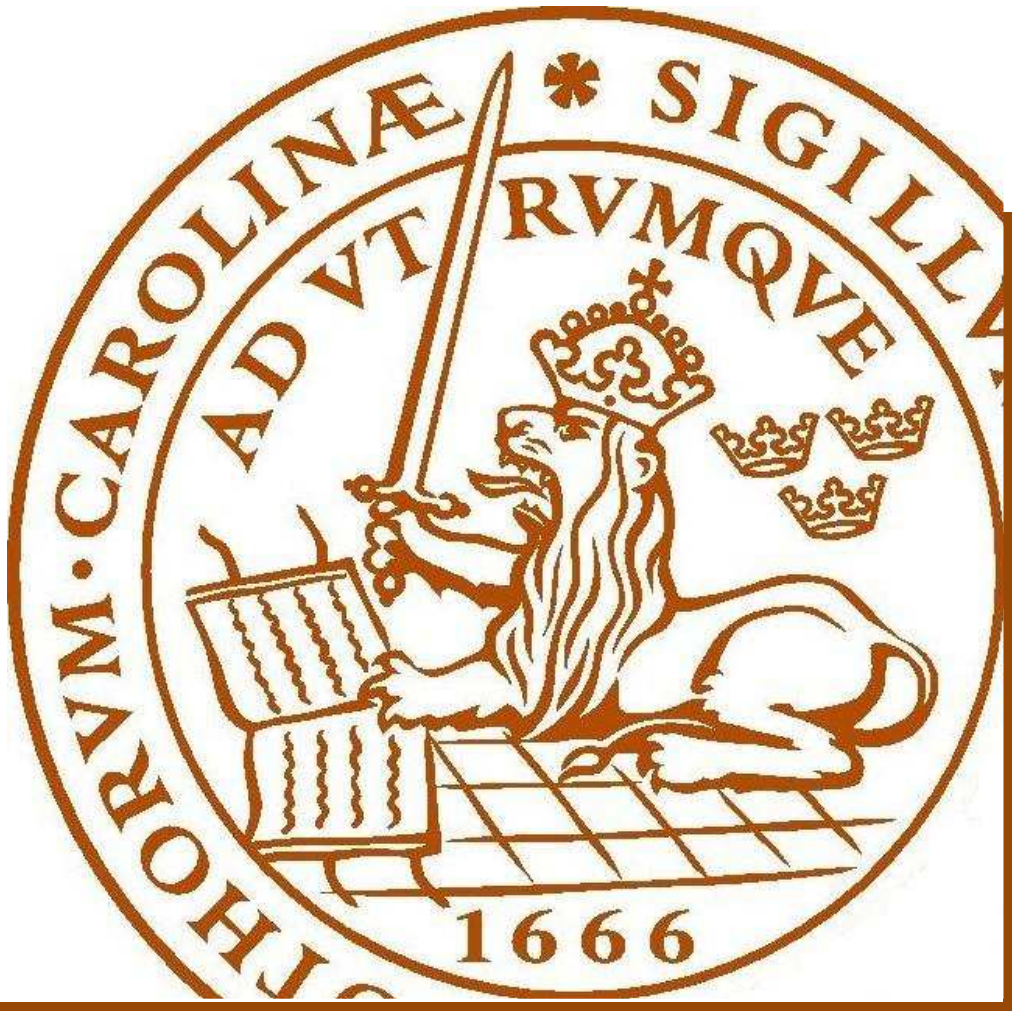
DIGPSK5

Version 1 2009-05-18

Handläggare Bertil Lindvall Bertil.Lindvall@eit.lth.s

Dokument Ansvarig Dan Kvelstad dt06dk5@student.lth.se

Gruppmedlem Anders Linden dt06al3@student.lth.se



Abstract

This digital project is an exercise in developing a prototype electronic system for the course EDI021 at Lunds Tekniska Högskola. The project was planned, designed, built and demonstrated by the students with minimal intervention of a supervisor. The target construction should be able to present pictures on a computer by utilizing an analogue camera and rs232 protocol over a standard comport. However, the project did not result in a working construction due to the design choice of explicitly using Lattice circuits instead of a microprocessor.

Innehållsförteckning

1 Syftet med denna Skrift 5

2 Planering 6

2.1 Ursprungliga Kravspecifikationen 6

2.1.1 Kommunikationsmedel 6

2.1.2 Analog Kamera 6

2.2 Slutgiltig Kravspecifikation 6

2.3 Dimensionering 7

2.3.1 A/D sampling 7

2.4 Mellanlagring 7

3 Design 8

3.1 Dataflöde 8

3.1.1 Den analoga signalen 8

3.1.2 A/D omvandlingen 8

3.1.3 Mellanlagring 8

3.1.4 Sändning 8

3.2 Kontroll 9

3.2.1 Video Synkroniserare 9

3.2.2 Singal Generator 9

3.2.3 Lattice: UART 10

3.2.4 Lattice: Klocka 10

3.2.5 Lattice: Minne 11

3.3 Datorprogramvara 11

4 Resultat 12

4.1 Den andra konstruktionen 12

4.2 Den slutgiltiga konstruktionen 12

4.3 Programvara 13

5 Utvärdering 14

5.1 Elektronisk design kontra Mjukvara 14

5.2 Planering 14

5.3 Design 14

5.4 Konstruktion 15

Appendix A Lattice 16
A.1 Clock 16
A.2 UART Controller 17
A.3 Memory 18
Appendix B JAVA UML 19
Appendix C JAVA kod 20
C.1 Controller 20
C.2 View 21
C.3 Model 23
C.4 ImageCreator 24
C.5 Reciever 25
C.6 Port 26
C.7 RS232 26
5.5 ModelTest 28

1 Syftet med denna Skrift

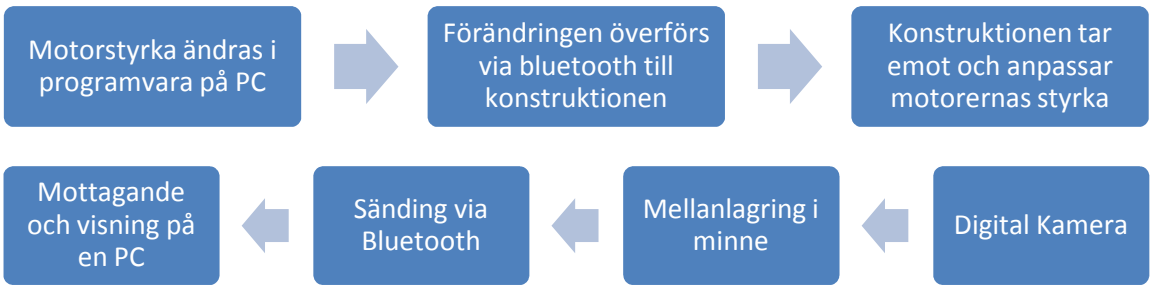
Rapporten ämnar sammanfatta projektet genom att beskriva de olika stegen genom vilka det planerades, designades och genomfördes kompletterat med motivering till valen samt erfarenheter. Tanken är att en ny grupp skall, efter att läst igenom denna rapport, kunna fortsätta utvecklingen av denna produkt och kunna utveckla en fungerande prototyp.

Det mer konkreta syftet är att ge en kollegiegrupp möjligheten och medlen att utföra en opponering samt presentera arbetet för handläggaren som sedan kan utvärdera om gruppen har uppfyllt kraven för att bli godkända på kursen.

2 Planering

2.1 Ursprungliga Kravspecifikationen

Projektets ursprungliga målsättning bestod av att bygga en hjulbaserad robot med möjligheten att styras via dator över Bluetooth. Operatören skulle kunna anpassa styrkan på höger respektive vänster hjul för att på så vis manipulera färdriktningen. I förlängningen var även tanken att en svart/vit bild skulle skickas via samma länk så att operatören kunde se vart den manövrerade.



2.1.1 Kommunikationsmedel

Att sända informationen via Bluetooth visade sig vara orealistiskt då de 8bitarsanpassade Bluetooth modulerna som finns att tillgå på marknaden är tämligen dyra och därav inte tillgängliga i kursen. Efter diskussion med handläggaren om detta så enades det om att låta roboten få en navelsträng i form av en USB kabel då modulerna för dessa är liknande i användning med 8bitars inmatning.

De USB moduler som fanns att tillgå genom kursen visade sig dock kräva seriell inmatning och inte parallell vilket var den ursprungliga tanken. Efter vidare studier av modulerna som fanns att tillgå fastställdes att de behöver en förbehandlas till RS-232 protokollet. Detta är ett protokoll som kan användas i sig självt via comport.

Ett följdproblem av att använda RS232 för kommunikation är att den har en långsammare hastighet. Ursprungligen planerades användningen av två minnesmoduler för att kunna arbeta växelvis med varandra. Det blev dock uppenbart att detta var överflödigt för denna konstruktion.

2.1.2 Analog Kamera

Då den ursprungliga kravspecifikationen tecknades fanns enbart information om att kameror fanns att tillgå. Det togs för givet att kamerorna var digitala och kravspecifikationen baserades på detta. Det framgick dock senare att detta inte var fallet utan att de var analoga. Här tillkom följaktligen att dessa måste AD omvandlas till digital information och att synkronisering måste göras av deltagarna.

2.2 Slutgiltig Kravspecifikation

Med anledning av 2.1.1 och 2.1.2 så eskalerade arbetsbördan och systemets komplexitet. Till följd av detta så enades gruppen tillsammans med handledaren att övergå till den mindre kursen och enbart göra en konstruktion med sändningen av bilden till datorn via comport. Ingen ny formell kravspecifikation formulerades utan blev en muntlig överenskommelse med handledare.

2.3 Dimensionering

Även om konstruktionen i sig inte kommer att innefatta Bluetooth så är tanken att den är dimensionerad för detta. Modulerna som finns på marknaden innefattar en symbolhastighet på 3Mbit per sekund och datagenomströmning på 2,1Mbit per sekund. Den senare siffran är den styrande faktorn på hur snabbt konstruktionen kan bli av med data.

2.3.1 A/D sampling

När hastigheten med vilken konstruktionen kan bli av med informationen nu är fastställd så är det läge att studera kameran. Då det inte finns en specifik klassificering på den och den är av ett anonymt fabrikat så utgås det för dimensionerings skull att den följer en 625radig PAL standard. I tabellen nedan visas mängden information som denna genererar, dels ifrån fabrikat och dels de värden som konstruktionen kommer att använda (s.k. optimerad). Läsaren uppmanas att observera att de optimerade värdena har viktats mot efterföljande tabeller för att få ett rimligt värde.

Kamera (VC36)	Enhet	Fabrik	Optimerad
Kol (Beräknat 4:3)	Pixlar	833	139
Rad (enligt PAL)	Pixlar	625	104
Antalet pixlar i en bild	Pixlar / Bild	520 833	14 468
Bilder /Sek	Bilder /Sek	25	2
Antalet pixlar / Sek	Pixlar / Sek	13 020 833	28 935

Efter en jämförelse med Bluetooth genomströmningen på 2,1Mbit/sek så ses spontant att konstruktionen inte rimligen kan ta hänsyn till all information som kameran ger (13MPixel/sek). För att effektivt jämföra dessa värden så måste dock den analoga signalen tolkas som en digital motsvarighet genom a/d omvandling.

A/D (TDA8703)		Fabrik	Optimerad
Samplingsfrekvens	Hz	40 000 000	28 935
Upplösning	Bitar / Pixel	8	8
En Kol i Digitalformat	Bitar	6 667	1 111
En Rad i Digitalformat	Bitar	5 000	833
En Bild i Digitalformat	Bitar	33 333 333	925 926
En Bild i Digitalformat	Bytar	4 166 667	115 741
En Sek i Digitalformat	Bitar / Sek	833 333 333	1 851 852

Genom att använda dessa värden skulle konstruktionen kunna prestera två bilder i sekunden med en storlek på 139*104 pixlar och klara av den begränsade genomströmningen med god marginal. Detta kommer dock att medföra att endast var sjätte rad i en bild skall samplas med frekvensen 29kHz. Vidare är comporten som används långsammare än Bluetooth och antalet bilder per sekund kommer att bli lidande.

2.4 Mellanlagring

En buffert är nödvändig för att lagra informationen innan den sänds vidare. Denna buffert måste klara av att lagra en bild åt gången.

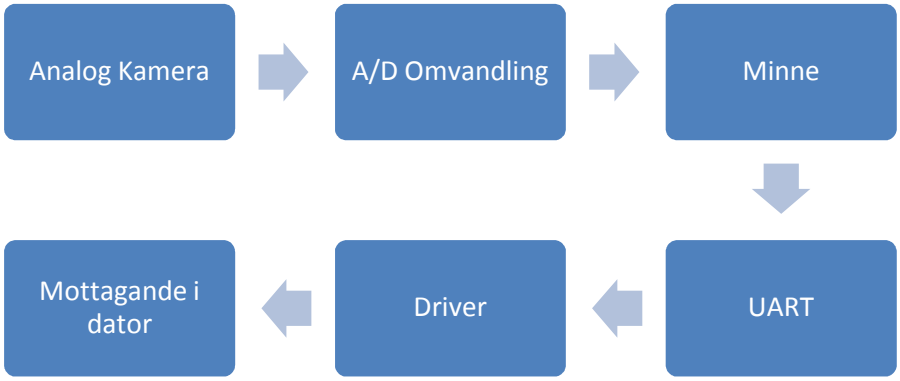
Minne (IS61C1024)		Fabrik	Optimerad
En Bilds Storlek	Bytar	4 166 667	115 741
Minnets Storlek	Bytar	131 072	131 072
Antal Access på en Sek	Access/Sek	28 571 429	28 571 429

Den valda kretsen är stor nog och antalet access på en sekund är avsevärt större än datamängden.

3 Design

3.1 Dataflöde

Till följd av den reviderade kravspecifikationen så ser dataflödet för en bild, utan hänsyn till kontroll, ut som i figuren nedan.



3.1.1 Den analoga signalen

Den analoga kameran förser A/D omvandlaren med en analog signal som är formaterad enligt PAL. Detta medför att ljusstyrkan i varje punkt framför kameran representeras som variation på spänningen i denna signal där det saturerade värdet är vitt och ett gränsvärde runt 1v är svart. För att skilja raderna åt så används ett nollvärde och bilderna skiljs åt med flera nollvärden.

3.1.2 A/D omvandlingen

Kretsen som valdes för A/D omvandlingen heter TDA8703 och är specifikt anpassad för omvandling av videosignaler. Denna krets samplar en punkt varje klockpuls och presenterar resultatet som ett 8 bitar stort tal, dvs från 0 till 255 decimalt. Konstruktionen kommer dock inte att kunna använda hela denna då den analoga signalen från kameran inte rör sig genom hela det mätbara området för AD omvandlaren. För att förbättra bildkvalitén skulle detta kunna åtgärdas med förstärkare.

3.1.3 Mellanlagring

A/D omvandlingen sker med en långt högre hastighet än vad dataöverföringen är kapabel till och således måste informationen mellanlagras. Detta sker genom att först sampla en bild och spara varje pixel i ett minne för att när sedan slutet av bilden nås, sluta sampla och börja sända. Minneskretsen som valdes är en 131KBytes stor höghastighets SRAM modul. Adressering av minnet är 17 bitar stort ($2^{17} = 131072$) och data skrivs/läses som bytes via de 8 databenen.

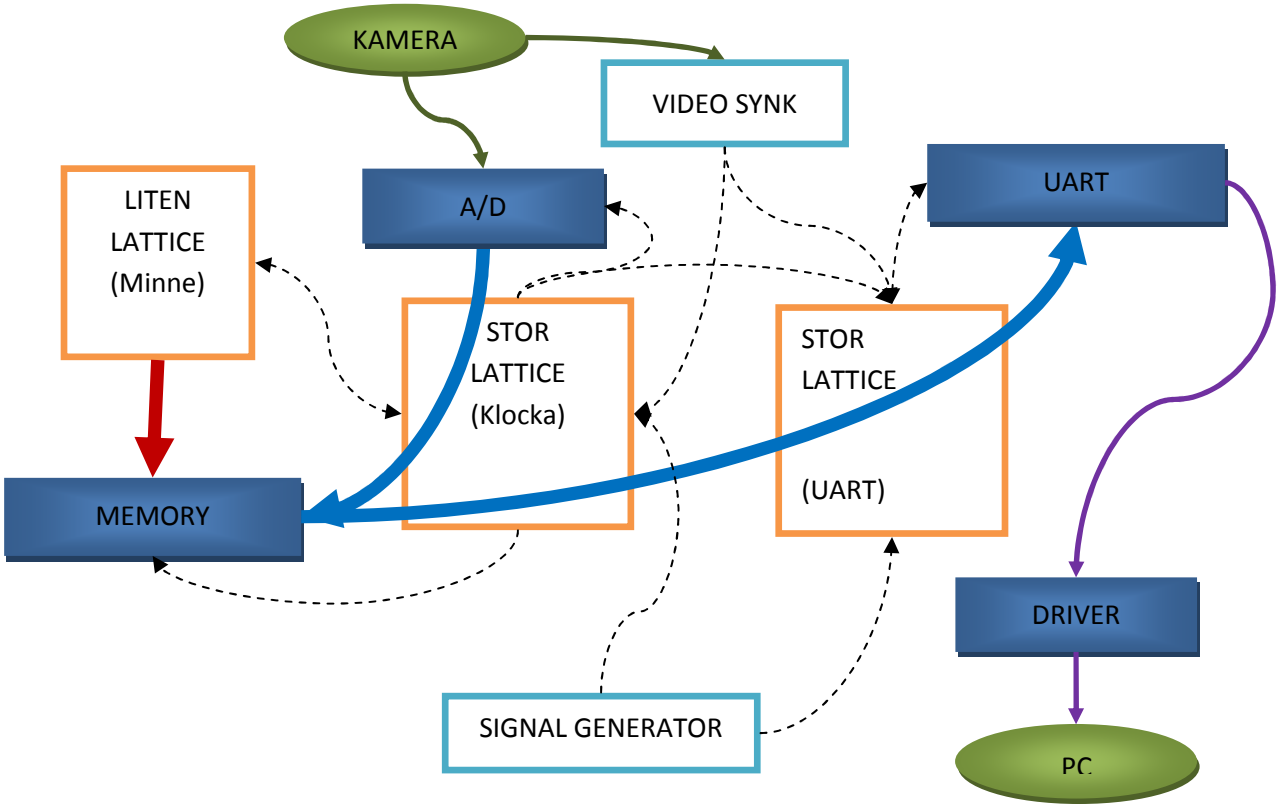
3.1.4 Sändning

UARTen tar emot 8bitar och omvandlar dessa till seriell form för att sändas via RS232 protokollet till en dator. Denna sändning görs konstant och när en bild läses från kameran skickas ett konstant värde från Latticen. Kretsen som valdes för detta är PC16550D då det var den enda enkelkanaliga kretsen som fanns att erhålla.

Utsignalen från UARTEN använder rätt protokoll men är inte stark nog för att användas via comport. För att lyckas med detta så används en driver som använder spänningspumpar för att förstärka signalerna.

3.2 Kontroll

Med hänseende till att dataflödet är enkelriktat och snabbt i förhållande till en mikroprocessor gjordes designvalet, i samråd med handledare, att genomföra detta med hårdkodad logik i Lattice kretsar. För detta ändamål valdes, slutligen, två större och en mindre Lattice krets och renderar en konstruktion enligt bilden nedan.



De solida kropparna känns igen från databeskrivningen i kapitlet innan och de nytillkomna elementen är kontrollelement. Kameran renderar den analoga signalen och de grova pilarna från A/D till minnet vidare till UART är 8 parallella bitar motsvarande en pixel. Den grova pilen från Adress Lattice till minnet styr adressen som den aktuella pixeln skall sparas i via 17bitar. De streckade pilarna motsvarar en eller flera kontrollsignaler. Slutligen sänds en seriell signal till drivern och vidare till PC.

3.2.1 Video Synkroniserare

Detta är en mindre krets vid namn lm1883 som från PAL insignalen extraherar radslut och bildslut. Denna information används för att styra dataflödet och därmed sändning eller skrivning av data. Dessutom sänds denna information vidare till datorn för att underlätta konstruktionen av bilden.

3.2.2 Singal Generator

I den ursprungliga designen planerades det använda en kristall på 1.83Mhz som klocka. I labbsalen fanns dock en signalgenerator som anpassades till samma frekvens. Detta är en frekvens som är anpassad mot UART för att uppnå en minimal felfrekvens vid sändning.

3.2.3 Lattice: UART

Se A.2 för källkod till denna latticekrets.

UART kretsen måste konfigureras genom adressering av och skrivning till register. Detta görs via adress benen (A0, A1, A2) i samråd med databenen (D0 – D7) samtidigt som en låg till hög klockpuls ges. Eftersom Lattice ser till att signalerna är konstanta under en klockcykel så ansluts UART Adress Strobe till logisk nolla. Den önskade funktionaliteten ur UART är följande.

Egenskap	Värde
Baudrate	9600
Stopbits	1
Paritybits	0
Flowcontrol	I startläget ingen, detta kan ändras till hårdvara när allt annat fungerar.

Totalt 6 register måste sättas enligt tabellen nedan för att ge önskad funktion som ovan. Viktigt att förstå är att kretsen tar emot en extern klocka på 1.9 MHz, denna signal använder sedan värdet i divisor latch registret för att skapa timingen för den seriella utsignalen. Eftersom Divisor Latch registret har samma adress som Send och Recieve registret så styr bit7 i Line Control registret vilken som används.

Register	MSB A2	A1	LSB A0	MSB D0	D1	D2	D3	D4	D5	D6	LSB D7
Line Control	0	1	1	11 8bitar/tecken		0 stopbit	0 0 0 No Parity Bits			0 Break?	1 DivLat?
Divisor Latch LSB	0	0	0	0	0	0	0	1	1	0	0
Line Control	0	1	1	11 8bitar/tecken		0 stopbit	0 0 0 No Parity Bits			0 Break?	0 DivLat?
FIFO	0	1	0	1 Enable	1 Reset Recieve	1 Reset Transmitt	1 Mode	0 0 Reserved		1 1 Reciever Trigger Level	
Modem	1	0	0	1 DTR	1 RTS	0 OUT1	1 OUT2	0 Loop		0 0 0 Reserved	
Send	0	0	0	X	X	X	X	X	X	X	X

Eftersom ingen flödeshantering används kan det tyckas onödigt att sätta modemregistret, det underlättar dock avsevärt för att säkerställa att registren sätts korrekt då detta register påverkar dedikerade ben på kretsen.

Genom att låta varje bit vara en egen, en-av-flera till en, mux med gemensam styrlogik så kan dessa kommandon sättas på registren i kretsen.

3.2.4 Lattice: Klocka

Se A.1 för källkoden till denna krets.

Denna krets har fyra uppgifter. Där den första är att räkna med ingångsfrekvensen, 1.9 MHz, och raderna för att utvärdera om och när klocksignal skall skickas till AD omvandlaren. AD omvandlaren är tänkt att sampla med frekvensen 28KHz.

$$\frac{1{,}9 \cdot 10^6}{28 \cdot 10^3} = 66$$

Av sambandet ovan ges att när 66 pulser från originalklockan har getts så skall en puls ges till AD omvandlaren. Logiken i Latticeprogrammet kommer att ge hög signal mellan 32 och 63 och låg resten för att ge stabilitet. Signalen är tänkt att vara positivt flanktriggad vilket rättfärdigar den asymmetriska fördelningen av hög och låg signal i tiden.

Nästa uppgift för kretsen är att utvärdera om konstruktionen skall läsa eller skriva till minnet. Detta värde hålls i en 1bits latch med klockan vsync, om värdet av latchen är ett så ska minnet skrivas till och annars läsas från. Om latchen är låg, läsläge, så kommer muxen att vidarebefordra ett om läsningen är klar och nästa vsync blir latchens värde hög. Om latchen är hög, skrivläge, kommer kretsen att ändra till läs nästa vsync puls.

Kretsen renderar även pulserna till Latticen för minnet. Dessa styrs om av att faktorer som om minnet är i läsläge om UART är kapabel att ta emot data och ingångsklockan är hög eller om AD klockan är hög.

Den slutliga uppgiften är att vid läsning vidarebefordra data från minnet till UART Latticen och vid skrivning vidarebefordra AD värdena till minnet.

3.2.5 Lattice: Minne

Se A.1A.3 för källkoden till denna krets.

Detta är den enklaste av de tre Lattice kretsarna. Den består av två 17bits uppräknare och ansvarar för adresseringen av minnet. Endast en räknare kan räkna åt gången och den ena 17bits räknaren räknar upp pixel för pixel tills en hel bild är inläst och den andra är under tiden stilla. När en bild är klar så står den första räknaren still och den andra räknar upp tills den kommer till samma värde som den första. Då detta inträffar så nollställs räknarna och förloppet upprepas.

3.3 Datorprogramvara

Då projektdeltagarna har mest vana med java så valdes detta programmeringsspråk till standard för att ta emot rå data från konstruktionen, omvandla denna till en bild och slutligen redovisa bilden. Programmet konstruerades i enlighet med designmönstret MVC (Model View Controller) med två controllers: Controller och Reciever.

I Appendix B återfinns ett UML diagram av programvaran och i Appendix B återfinns programkoden.

Huvudapplikationen renderas av de fyra klasserna Controller, Model, View och Reciever. Reciever körs i en egen tråd och använder sig av RS232 genom interfacet Port. RS232 måste utökas med att hantera den konvention som överenskommes i konstruktionen. Tanken bakom interfacet är att ge stöd för att ansluta till konstruktionen via ett annat gränssnitt än comport. RS232 använder sig av stödklassen ImageCreator för att omvandla en vektor med nummer i byte format till en bild som sedan ritas upp i View.

4 Resultat

Konstruktionen fungerar inte.

Konstruktionen byggdes om totalt tre gånger där den första var en fullständig katastrof och behandlas inte i denna rapport. De två efterföljande byggdes med två olika förfarandesätt men med samma slutdesign i åtanke. En design som logiskt borde fungera men som var oväntat svår att implementera praktiskt med Lattice.

4.1 Den andra konstruktionen

I detta försök byggdes allt på en gång. Förhoppningen var att följa dataflödet från kameran, anpassa kontrollen där det inte fungerade, och på så vis målbogsera bilden. Det verkade fungera att sampla bilden med AD och spara i minnet men sen tog det stop. Problemet blev att det är för mycket information för att bekräfta med oscilloskop och dart och att för varje fel behövdes tre Latticekretsar felsökas. Om en Lattice kompilerades om så flyttades dess ben vilket i sin tur påverkade de andra Latticekretsarna. Partitioneringen på Latticelogiken medförde även att för att ändra en funktion så behövdes två kretsar omprogrammeras vilket medförde att så gott som hela kretsen behövde kopplas om. Med 17 addressben, 24 databen och minst lika många logikben placerade, för en människa, slumpvis över ett hundratal ben så betyder detta en hel del arbete med hög felkopplingsrisk. Slutligen insågs att denna metod var hopplös och en omkonstruering behövdes.

4.2 Den slutgiltiga konstruktionen

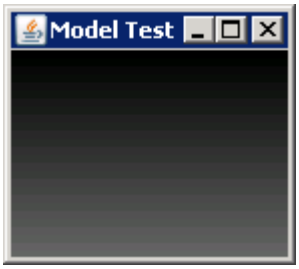
Slutkonstruktionen byggdes efter principen om verifierbarhet framför allt. Detta medförde att målet var först att skicka skräpdata, hårdkodad, från konstruktionen. När detta fungerade kunde resten kopplas upp för att skicka bildinformationen istället. Detta medför att endast en Lattice behöver vara involverad tillsammans med UART och driver. När detta fungerar fås ett tydligt och användbart resultat till datorn. Att sedan koppla upp resten stegvis sågs som enklare eftersom det övriga verkade fungera hjälpligt i instans två och endast behövdes justeras och buggjagas. För att minska konsekvenserna av benflyttning vid kompilering så användes även mellankopplingsstationer utförligt.

Problemet som uppkom var registersättningen . Lattice kretsarna hemsöktes av problem som att ett ben inte sände information, vilket visade sig vara trasigt. Då Lattice flyttar benplaceringen mellan kompileringar så var detta ett fel som inte var så lättfunnet som kan tänkas då felsökning görs systematiskt men felet är sporadiskt.

Vid ett läge sattes RTS, DTR, Out1 och Out2 till rätt värden vilket tyder på att det näst sista, modem control, av registren var satt rätt. UART kretsen gav dock enbart hög signal ut. Detta är nära rätt resultat då skräpdata som skickades var 255 men stopbiten är 0 vilket visar på fel i UART. Det visades att Lattice verkar räkna fel. Ett absurt påstående men baserat på utmatning av samtliga påverkande faktorer och sänkning av hastigheten till en puls i sekunden. Detta medförde den slutgiltiga Lattice logiken för UART där varje räknare har en egen klocka i ett försök att komma åt detta. Denna logik räknar inte alls upp registren.

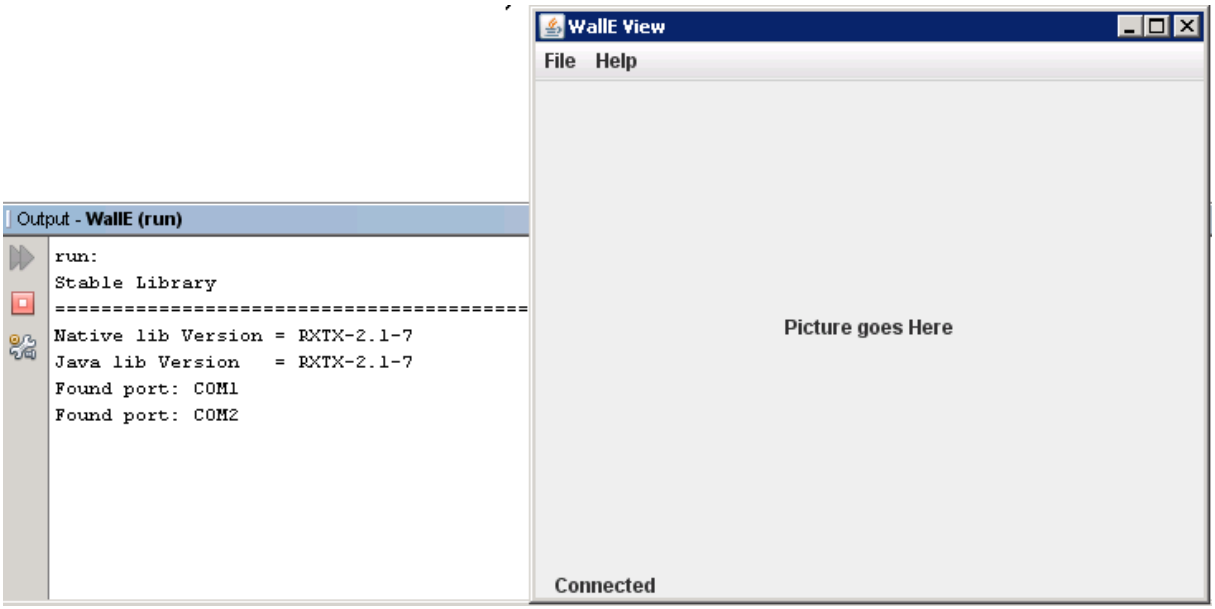
4.3 Programvara

Eftersom konstruktionen inte fungerar så kan inte programvaran anpassas efter den. Programvaran är dock testad och fungerar med testdata. Appendix JAVA Test resulterar i två fönster enligt nedan.



Dessa bilder är uppritade med en, förvisso perfekt, byte array. En del anpassning mot den faktiska arrayen skulle behövas då antalet pixlar på en rad kan variera en liten aning och radbryt, bildbryt ges som värden i vektorn något som inte tas hänsyn till i nuvarande implementering.

Nedan presenteras applikationen som används för den faktiska bildvisningen.



I exemplet ovan fanns två comportar på värddatorn och därav anslöts programmet till den som hittades först, com1. Programmet är nu i väntläget och redo att ta emot data från konstruktionen.

5 Utvärdering

Resultatet av projektet beror på värdering. Ur synpunkten för konstruktionen så är det inte lyckat men ur perspektivet att målet är att lära så är det en stor succé för deltagarna. Det mest givande upplevs som känslan för hur elektronik fungerar, detta var första tillfället då polletten på allvar trillade ner om grundläggande begrepp. Störningar i kretsar, decoupling, referenspunkter, signaler osv… Även planering och design av sådana system var givande då ingen tidigare kunskap fanns om detta. Grundläggande för projektet upplevdes begrepp som ”try-a-error” och ”gör om, gör rätt”, vilket är tidsödande och slitsamt men lärorikt.

5.1 Elektronisk design kontra Mjukvara

Den stora skillnaden elektronisk- och mjukvarudesign är att det senare vanligen byggs uppifrån och ner. Det vill säga att ett skelettprogram skapas och enskilda detaljer läggs åt sidan tills de behövs då de implementeras. Denna princip är helt felaktlig för att skapa ett elektroniskt system, där detaljerna är allt och där tillgången till komponenter styr om ett projekt är genomförbart överhuvudtaget.

Ett koncept som är viktigt inom bägge fälten är testning och verifiering men implementeringen skiljer. I mjukvarusammanhang är testning vanligen något som görs efteråt och om extra stöd behövs ändras helt enkelt systemet för det, det finns metodiker som vänder på detta bl.a. XP. Vid design av elektroniska system måste testbarheten stå i fokus långt innan kablar börjar dras. Att tänka ”detta borde fungera.” är fundamentalt fel och borde vara ”hur testar jag detta?”.

5.2 Planering

Kalkylen som utfördes i början av projektet, delar av vilket presenterades i tabellform i planeringskapitlet, underlättade planering och design avsevärt. Att ta fram krav specifikation för komponenter och relationerna mellan dessa som de måste uppfylla är en stor hjälp för att senare välja dem.

Att bygga ett system med en uppsättning komponenter med tanken att senare byta ut dessa kommer enbart att komplicera till det. Hitta komponenterna du tänker använda och använd dem eller dess substitut. Använd inte substitut och förvänta dig att senare byta till originalet. Detta är fallet för UART och rs232 i detta projekt som avsågs bytas ut mot Bluetooth.

Fråga efter andras åsikter speciellt om du är osäker på ett specifikt område eller generellt, expertis finns runtom. Ge dig inte in i ett projekt som känns bra eller logiskt utan att egentligen grunda det på annat än ditt egna sunda förnuft.

5.3 Design

Ett grundläggande designfel begicks i projektet, det att inte ta med en mikroprocessor. Tanken bakom var att inte komplicera designen med ytterligare komponentsorter utan att använda fler av de redan existerande Lattice kretsarna. Det som en mikroprocessor dock förenklar avsevärt är enkelheten att förändra och testa.

Använd inte datoriserade ritprogram för kretsar, á la Power Logic, de är tidsödande, komplicerade, buggiga och gamla. Dessutom kommer det alltid bli problem med komponenterna, finns de eller finns de inte i databasen? Stämmer pinnarna s layout eller inte? Stämmer numreringen? Risken är stor att efter ha ritat ett program i powerlogic upptäcka att man ändå måste sitta med databladen istället och att programmet inte hjälpte till med något. Vidare försvårar de kommunikationen mellan grupper, en grupp som skall ge feedback är kanske inte insatt i programmet eller dess specifika

funktioner. Detta var fallet för denna grupp då kommunikationen med handledaren försvårades eftersom gruppen använde en abstraheringsfunktion i powerlogic kallad hierarkisk symbol. En långt överlägsen metod är att rita med papper och penna kommunikationen mellan komponenterna liknande ritningen i 3.2. När sedan en komponent skall kopplas upp tas databladet fram och pinnumreringen löses på plats tillsammans med för kretsen egna komponenter som decoupling.

5.4 Konstruktion

Det kan vara lockande att välja ett mindre kretsbord då det när man placerar ut kretsarna ser tomt ut på det större. Undvik dock detta, en ratnest-baserad prototyp behöver inte vara liten utan fullt tvärtom. Det ett bevis av koncept och härav hörs att tydligheten och enkelhet att koppla upp och följa är viktigare. Ett praktexempel av detta är att pin layouten är inte alltid lätt att memorera eller hitta. Minnesmodulen hade t.ex. 17 addressben som inte var placerade i nummerordning. Att dra dessa till 17 parallella pinnar och sedan dra från dessa vidare underlättar avsevärt.

Den utrustning som finns att tillgå är värt att lära sig hantera.

Eftersom Lattice är uppbyggt runt GLB's så måste kompilatorn bestämma pinnumret för de logiska portarna. Dessa kan byta plats om en omkompilering måste göras! Detta gör Lattice till en extremt otymplig plattform att implementera större logik. En metod att minska konsekvenserna av detta är att använda flera mindre kretsar även om de tryckas in i en större. Som det är nu i projektet så innebär en mindre förändring i Lattice att ett antal kablar med stor sannolikhet behövs flyttas, vilket är tidsödande och en stor felkälla. Detta hjälptes i konstruktionen med att skapa mellankopplingstationer mellan kretsar och Lattice.

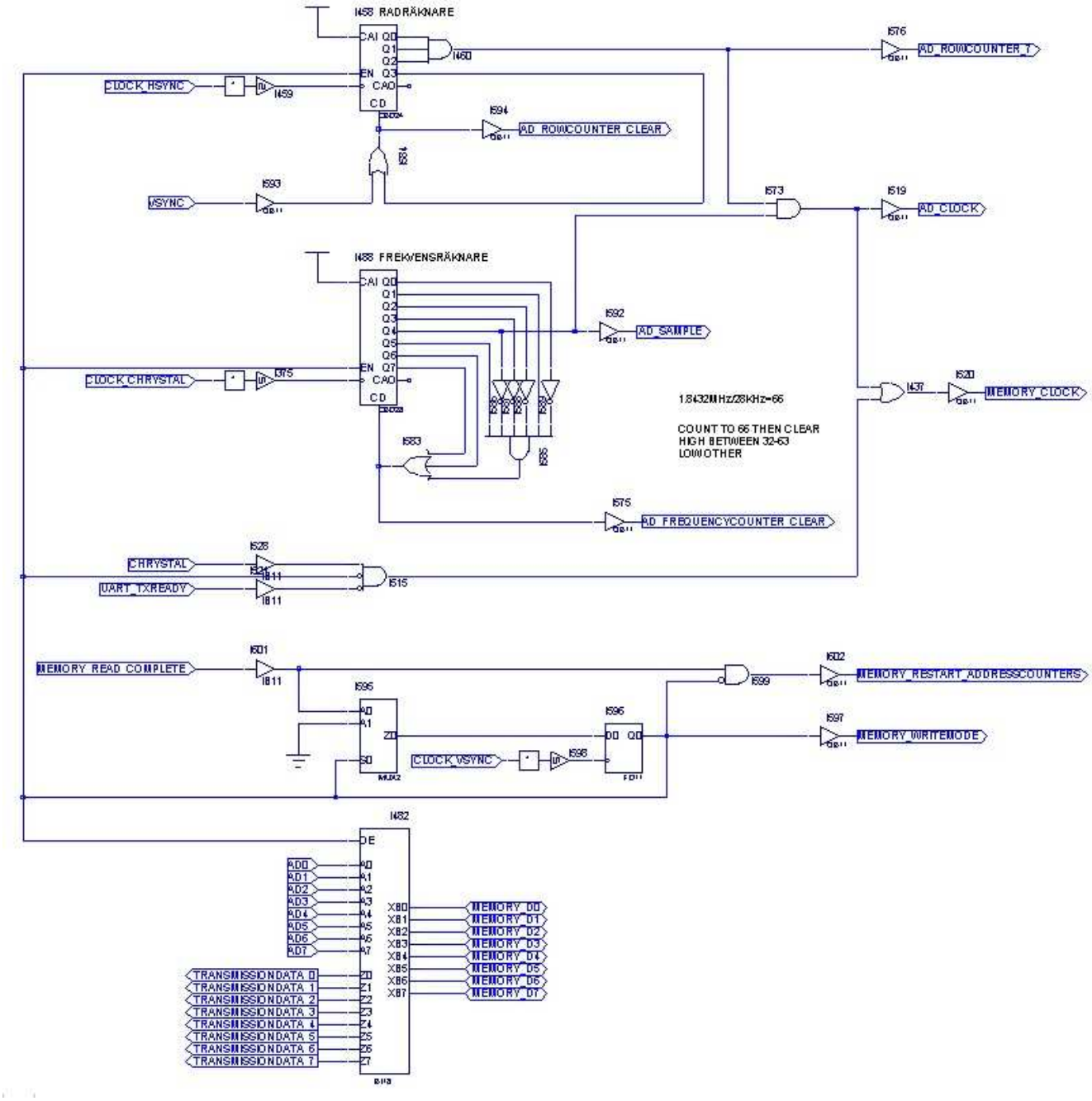
För att kunna verifiera och testa vad Lattice gör måste användaren lägga in ett stort antal utgångar i logiken. Utöver att göra designen stor och otydlig så är det överhuvudtaget kanske inte är möjligt p.g.a. antalet tillgängliga portar. Ofta leder felsökning av denna sort till att ett antal platser kontrolleras, inga fel hittas, och nästa platser måste undersökas. Med undersökningsportarna flyttas även funktionspinnarna. Då Lattice kretsen nu behövs kopplas om så kan de tidigare testfallen nu vara fel utan möjlighet att mäta dem.

I Lattice använder alla räknare, latchar mm en klockpuls. Dessa är speciella ingångar och de är inte många. Den mindre Lattice kretsen har 2externa klockor och 1 intern och dem större har 3externa och 1 intern. Detta försvårar avsevärt. Då en gemensam styrlogik för logiken måste införas för att kunna använda samma klocka vid olika pulser istället för att ha egna klockor. Detta är dessutom utanför vad hjälpdokumenten behandlar då det handlar om timing och inte direkt funktionalitet av macro. Detta inträffade ofta och som exempel kan en räknare efter en räknare tas:

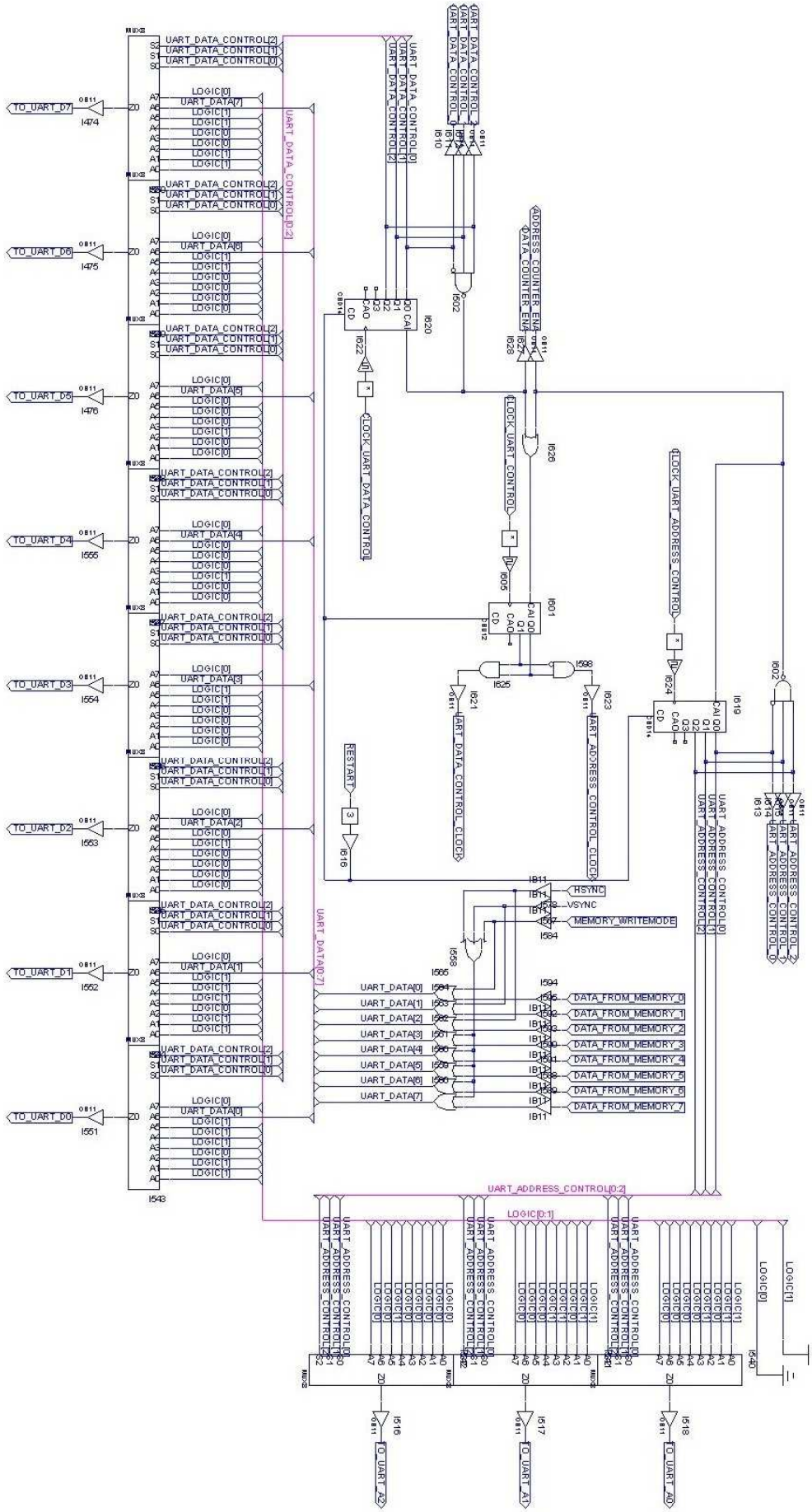
Om räknare ett räknar upp till 10 och räknare två skall ökas med ett då räknare ett är 4. Det enkla hade varit att ansluta bit 3 från räknare ett till klockan för räknare två. Detta är dock inte möjligt eftersom klockorna är ett eget nätverk skilt från GLB där räknarna finns. Ett alternativ är att ansluta bit 3 från räknare ett till enable för räknare två och låta dem använda samma klocka. Här kommer dock nästa otydlighet in. Kommer räknare två att räkna upp då räknare ett är 4 eller 5? Kommer den att räkna en eller två gånger beroende på när enable sätts? Klockorna är flanktriggade… Logiken från lattice är en heavisides steg funktion. Detaljer som dessa uppträder alltid i Lattice och inga svar verkar finnas varken i datablad eller på internet.

Appendix A Lattice

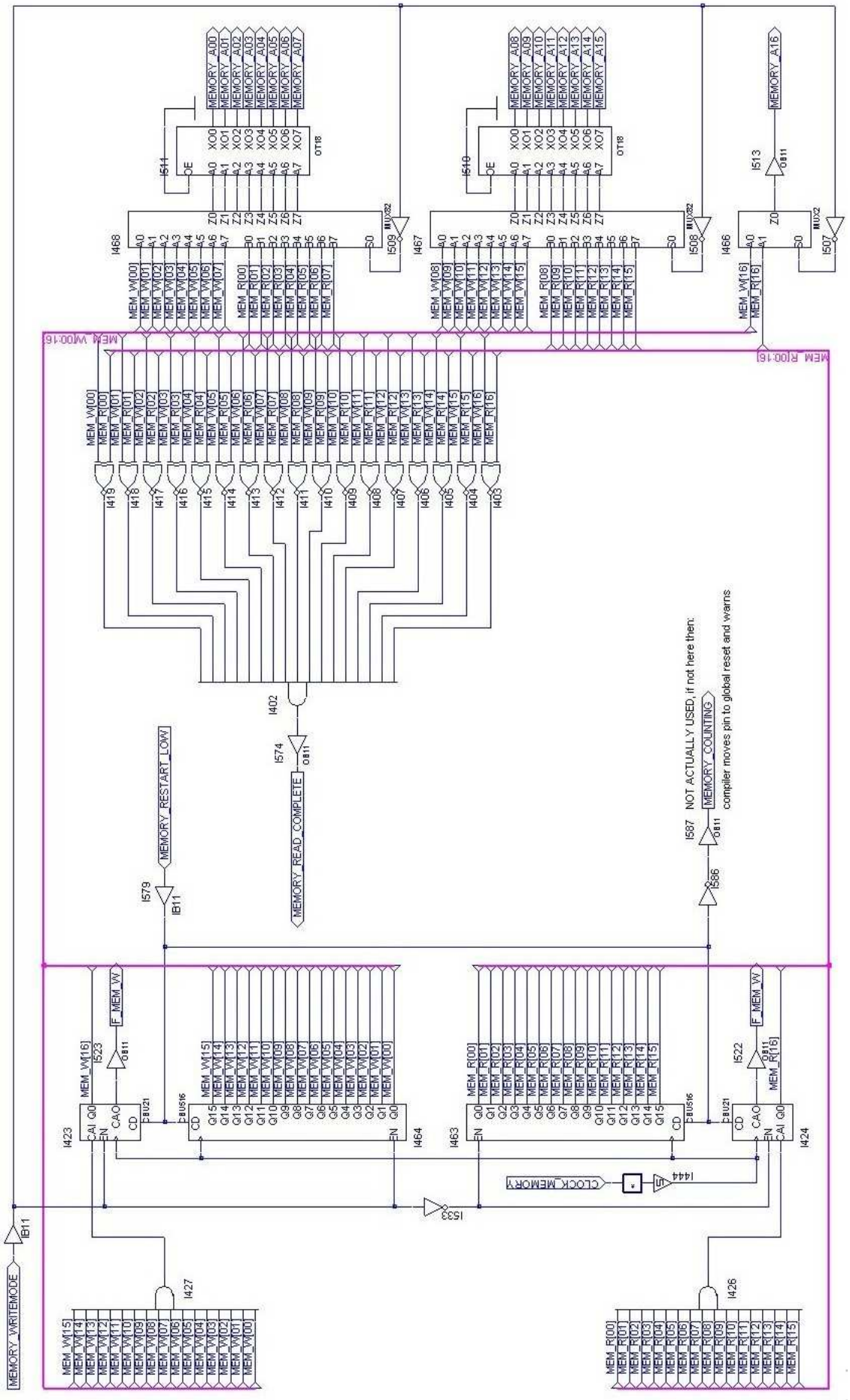
A.1 Clock



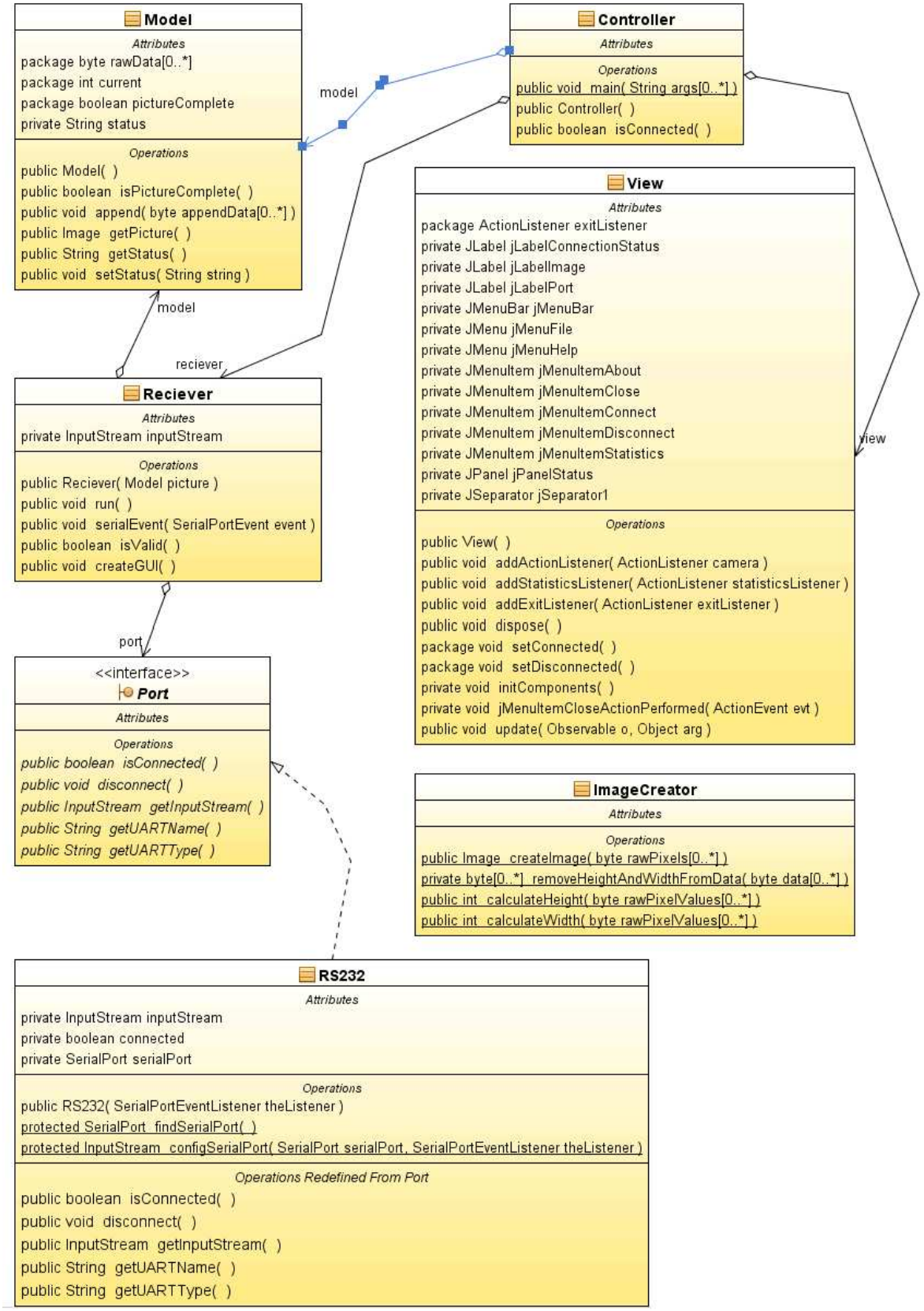
A.2 UART Controller



A.3 Memory


MEMORY_A00
MEMORY_A01
MEMORY_A02
MEMORY_A03
MEMORY_A04
MEMORY_A05
MEMORY_A06
MEMORY_A07
MEMORY_A08
MEMORY_A09
MEMORY_A10
MEMORY_A11
MEMORY_A12
MEMORY_A13
MEMORY_A14
MEMORY_A15
MEMORY_A16
I514
I540
I513
I468
I467
I466
I509
I508
I507
MEM_W[00]
MEM_W[01]
MEM_W[02]
MEM_W[03]
MEM_W[04]
MEM_W[05]
MEM_W[06]
MEM_W[07]
MEM_R[00]
MEM_R[01]
MEM_R[02]
MEM_R[03]
MEM_R[04]
MEM_R[05]
MEM_R[06]
MEM_R[07]
MEM_W[08]
MEM_W[09]
MEM_W[10]
MEM_W[11]
MEM_W[12]
MEM_W[13]
MEM_W[14]
MEM_W[15]
MEM_R[08]
MEM_R[09]
MEM_R[10]
MEM_R[11]
MEM_R[12]
MEM_R[13]
MEM_R[14]
MEM_R[15]
MEM_W[16]
MEM_R[16]
MEM_W[00:16]
MEM_R[00:16]
I419
I418
I417
I416
I415
I414
I413
I412
I411
I410
I409
I408
I407
I406
I405
I404
I403
I402
I574
MEMORY_READ_COMPLETE
I579
MEMORY_RESTART_LOW
I587
MEMORY_COUNTING
NOT ACTUALLY USED, if not here then:
compiler moves pin to global reset and warms
I586
F_MEM_W
I523
I522
I423
I464
I463
I424
I444
CLOCK_MEMORY
I533
MEMORY_WRITEMODE
I427
I426

Appendix B JAVA UML



Appendix C JAVA kod

C.1 Controller

```
package walle;

import communication.Reciever;
import java.awt.event.ActionEvent;

public class Controller {
    private Model       model;
    private View        view;
    private Reciever    reciever;

    public static void main(String[] args) {

        walle.Controller  theController   = new walle.Controller();

    }
    public Controller(){
        model   = new Model();
        view    = new View();

        view.addActionListener(new Camera());
        view.addStatisticsListener(new StatisticsListener());
        view.addExitListener(new ExitListener());
        view.setVisible(true);
    }
    public boolean isConnected(){
        if(reciever != null)
            return reciever.isValid();

        return false;
    }
    class Camera implements java.awt.event.ActionListener{
        public void actionPerformed(ActionEvent e) {
            if(e.getActionCommand().compareTo("Connect") == 0){
                reciever = new Reciever(model);
                if(reciever.isValid()){
                    reciever.start();
                    view.setConnected();
                }else
                    view.setDisconnected();
            }else if(e.getActionCommand().compareTo("Disconnect")== 0){
                reciever.interrupt();
                view.setDisconnected();
            }
        }
    }
    class StatisticsListener implements java.awt.event.ActionListener{
        public void actionPerformed(ActionEvent e) {
            if(reciever != null)
                reciever.createGUI();
        }
    }
```

```
    class ExitListener implements java.awt.event.ActionListener{
        public void actionPerformed(ActionEvent e) {

            if(e.getActionCommand().compareTo("exit") == 0){
                if(reciever != null)
                    reciever.interrupt();
                System.exit(0);
            }
        }
    }
}
```

C.2 View

```
package walle;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observable;
import javax.swing.ImageIcon;

public class View extends javax.swing.JFrame implements java.util.Observer {

    ActionListener exitListener;

    /** Creates new form View */
    public View() {
        initComponents();
    }

    public void addActionListener(ActionListener camera) {
        jMenuItemConnect.addActionListener(camera);
        jMenuItemDisconnect.addActionListener(camera);
    }

    public void addStatisticsListener(ActionListener statisticsListener) {
        jMenuItemStatistics.addActionListener(statisticsListener);
    }

    public void addExitListener(ActionListener exitListener){
        this.exitListener = exitListener;
    }

    @Override
    public void dispose(){
        exitListener.actionPerformed(new ActionEvent(this, 0, "exit"));
        super.dispose();
    }

    void setConnected() {
        jMenuItemDisconnect.setEnabled(true);
        jMenuItemConnect.setEnabled(false);
        jLabelConnectionStatus.setText("Connected");
    }

    void setDisconnected() {
        jMenuItemDisconnect.setEnabled(false);
        jMenuItemConnect.setEnabled(true);
        jLabelConnectionStatus.setText("Disconnected");
    }
```

```
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {
          //BORTTAGEN FRÅN RAPPORTEN EFTERSOM DEN ÄR 4SIDOR OCH INTE ÄR AV VIKT.

    private void jMenuItemCloseActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jMenuItemCloseActionPerformed
        dispose();
    }//GEN-LAST:event_jMenuItemCloseActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JLabel jLabelConnectionStatus;
    private javax.swing.JLabel jLabelImage;
    private javax.swing.JLabel jLabelPort;
    private javax.swing.JMenuBar jMenuBar;
    private javax.swing.JMenu jMenuFile;
    private javax.swing.JMenu jMenuHelp;
    private javax.swing.JMenuItem jMenuItemAbout;
    private javax.swing.JMenuItem jMenuItemClose;
    private javax.swing.JMenuItem jMenuItemConnect;
    private javax.swing.JMenuItem jMenuItemDisconnect;
    private javax.swing.JMenuItem jMenuItemStatistics;
    private javax.swing.JPanel jPanelStatus;
    private javax.swing.JSeparator jSeparator1;
    // End of variables declaration//GEN-END:variables

    public void update(Observable o, Object arg) {
        Model model = (Model) o;

        jLabelConnectionStatus.setText(model.getStatus());

        if(model.isPictureComplete())
            jLabelImage.setIcon(new ImageIcon(model.getPicture()));


    }

}
```

C.3 Model

```
package walle;

import support.*;
import java.awt.Image;

public class Model extends java.util.Observable{
    byte[]  rawData;
    int     current;
    boolean pictureComplete;
    private String status;

    public Model(){
        rawData         = new byte[20000];
        current         = 0;
        pictureComplete = false;
        status          = "Not Connected";
    }

    public synchronized boolean isPictureComplete() {
        return pictureComplete;
    }

    public synchronized void append(byte[] appendData){
        for(int i = 0; i < appendData.length; i++, current++){
            rawData[current] = appendData[i];
            if(appendData[i] >= 255){
                pictureComplete = true;
                setChanged();
                notifyObservers(getPicture());
                return;
            }
        }
    }
     public synchronized Image getPicture(){
        current = 0;
        pictureComplete = false;
        return ImageCreator.createImage(rawData);
    }

    public synchronized String getStatus() {
        return status;
    }

    public synchronized void setStatus(String string) {
        status = string;

        setChanged();
        notifyObservers(string);
    }
}
```

C.4 ImageCreator

```
package support;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import java.awt.image.Raster;

public class ImageCreator {
    public static Image createImage(byte[] rawPixels) {
        int width               = calculateWidth(rawPixels);
        int height              = calculateHeight(rawPixels);
        byte[] refinedPixels    = removeHeightAndWidthFromData(rawPixels);

        //create image canvas with support for greyscale images
        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_BYTE_GRAY);

        //fill canvas with actual data
        image.setData(Raster.createWritableRaster(
                image.getColorModel().createCompatibleSampleModel(width, height),
                new DataBufferByte(refinedPixels, refinedPixels.length),
                null));

        return image;
    }

    private static byte[] removeHeightAndWidthFromData(byte[] data){
        byte[] refined = new byte[data.length-calculateHeight(data)];
        int i, j;

        for(i = 0, j = 0; i < data.length; i++)
            if(data[i]+255 >= 254){
                refined[j]=data[i];
                j++;
            }

        return refined;
    }
     public static int calculateHeight(byte[] rawPixelValues) {
        //camera encodes 255 as a new frame
        for(int i = 0; i < rawPixelValues.length; i++){
            if(rawPixelValues[i]+256 == 255)
                return i/calculateWidth(rawPixelValues)-1;
        }

        return 0;
    }

    public static int calculateWidth(byte[] rawPixelValues) {
        //camera encodes 254 as a new line
        for(int i = 0; i < rawPixelValues.length; i++)
            if(rawPixelValues[i]+256 == 254)
                return i;

        return 0;
    }

}
```

C.5 Reciever

```
package communication;

import communication.ports.Port;
import communication.ports.RS232;
import gnu.io.*;
import java.io.IOException;
import java.io.InputStream;
import walle.Model;

public class Reciever extends Thread implements SerialPortEventListener{
    private Port        port;
    private Model       model;
    private InputStream inputStream;

    public Reciever(walle.Model picture){
        port       = new RS232(this);
        inputStream = port.getInputStream();
    }

    @Override
    public void run() {
        if(port.isConnected()){
            try {
                while (!interrupted()) {
                    sleep(1000);
                }
            } catch (InterruptedException e) {

                interrupt();
            }
            port.disconnect();
        }
    }
    public void serialEvent(SerialPortEvent event) {
        switch (event.getEventType()) {
            case SerialPortEvent.BI:
            case SerialPortEvent.OE:
            case SerialPortEvent.FE:
            case SerialPortEvent.PE:
            case SerialPortEvent.CD:
            case SerialPortEvent.CTS:
            case SerialPortEvent.DSR:
            case SerialPortEvent.RI:
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                break;
            case SerialPortEvent.DATA_AVAILABLE:
                byte[] readBuffer = new byte[20];

                try {
                    while (inputStream.available() > 0) {
                        int numBytes = inputStream.read(readBuffer);
                        System.out.println(numBytes);
                    }
                    model.append(readBuffer);
                } catch (IOException e) {}

                break;
        }
    }
```

```
    public boolean isValid() {
        return port.isConnected();
    }
    public void createGUI() {
        View view = new View(null, false);

        view.setUARTName(port.getUARTName());
        //view.setUARTType(rs232.getUARTType());

        view.setVisible(true);
    }
}
```

C.6 Port

```
package communication.ports;

import java.io.InputStream;

public interface Port {
    public boolean      isConnected();
    public void         disconnect();
    public InputStream  getInputStream();
    public String       getUARTName();
    public String       getUARTType();
}
```

C.7 RS232

```
package communication.ports;

import communication.*;
import java.io.IOException;
import java.io.InputStream;
import java.util.Enumeration;
import java.util.TooManyListenersException;
import gnu.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;


public class RS232 implements Port {

    private InputStream inputStream;
    private boolean     connected;
    private SerialPort  serialPort;

    public RS232(SerialPortEventListener theListener){


        connected   = false;
        serialPort  = findSerialPort();

        if(serialPort != null){
            inputStream = configSerialPort(serialPort, theListener);
            if(inputStream != null)
                connected   = true;
        }
    }
```

```
    public boolean isConnected() {
        return connected;
    }
    public void disconnect(){
        connected = false;
    }
    public InputStream getInputStream(){
        return inputStream;
    }
    public String getUARTName(){
        if(serialPort != null)
            return serialPort.getName();
        else
            return "Not Connected";
    }
    public String getUARTType(){
        try {
            return serialPort.getUARTType();
        } catch (UnsupportedCommOperationException ex) {
            Logger.getLogger(RS232.class.getName()).log(Level.SEVERE, null, ex);
        }

        return "";
    }
    protected static SerialPort findSerialPort() {
        SerialPort          serialPort  = null;
        CommPortIdentifier  portId;
        Enumeration         portList;

        portList = CommPortIdentifier.getPortIdentifiers();

        while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                System.out.println("Found port: "+portId.getName());
                try {
                    serialPort  = (SerialPort) portId.open("WebCam a la WallE",
2000);
                } catch (PortInUseException e) {}
            }
        }
        return serialPort;
    }
    protected static InputStream configSerialPort(SerialPort serialPort,
SerialPortEventListener theListener){
        try {
            serialPort.addEventListener(theListener);
            serialPort.notifyOnDataAvailable(true);
            serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8,
                                           SerialPort.STOPBITS_1,
                                           SerialPort.PARITY_NONE);

            return serialPort.getInputStream();
        } catch (IOException e) {
        } catch (TooManyListenersException e) {
        } catch (UnsupportedCommOperationException e) {
        }
        return null;
    }
}
```

5.5 ModelTest

```
import java.awt.Image;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import walle.Model;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class ModelsTest {
    private byte[]  rawData;
    private int     nLines;
    private int     nCols;

    public ModelsTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }
    @Before
    public void setUp() {
        int i   = 0;
        int j   = 0;
        nLines  = 104;
        nCols   = 139;
        rawData = new byte[nLines*nCols];

        for(i = 0; i < nLines; i++){
            for(j = 0; j < nCols; j++)
                rawData[i*nCols+j] = (byte) i;
            rawData[i*nCols+j-1] = (byte) 254;
        }
        rawData[rawData.length-1] = (byte) 255;
    }
    @After
    public void tearDown() {
    }
    @Test
    public void testCalculateWidth() {
        int calculatedWidth = support.ImageCreator.calculateWidth(rawData)+1;
        assertTrue( "supposed to be: "+nCols+", but was:
"+String.valueOf(calculatedWidth),
                    calculatedWidth == nCols);
    }
```

```
    @Test
    public void testCalculateHeight() {
        int calculatedHeight = support.ImageCreator.calculateHeight(rawData)+1;
        assertTrue( "supposed to be: "+nLines+", but was:
"+String.valueOf(calculatedHeight),
                    calculatedHeight == nLines);
    }
    @Test
    public void testImageCreator() {
        Image image     = support.ImageCreator.createImage(rawData);
        assertNotNull("Image is null", image);

        testUsingImage("ImageCreator Test",
support.ImageCreator.createImage(rawData));
    }
    @Test
    public void testModel() {
        int i = 0;
        Model model = new Model();

        while(i < rawData.length){
            int nRecieved = (int) (Math.random() * 10);
            byte[] recieved = new byte[nRecieved];
            for(int j = 0; j < nRecieved && i < rawData.length; j++, i++)
                recieved[j] = rawData[i];
            model.append(recieved);
        }

        Image image     = model.getPicture();
        assertNotNull("Image is null", image);

        testUsingImage("Model Test",image);
    }
    private static void testUsingImage(String origin, Image image){
        try {
            System.out.println( "User have to verify correctness, \n" +
                                "should show a window with a sunset");
            TestFrame theFrame = new TestFrame(image);
            theFrame.setTitle(origin);
            theFrame.setVisible(true);
            Thread.sleep(2500);
        } catch (InterruptedException ex) {
            Logger.getLogger(ModelsTest.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
    private static class TestFrame extends javax.swing.JFrame {
        public TestFrame(Image image){
            JLabel theImage = new JLabel(new ImageIcon(image));
            theImage.setSize(image.getWidth(null), image.getHeight(null));
            add(theImage);
            pack();
        }
    }

}
```