Sammanfattning

På astronomiska observatoriet i Uppsala så har man nyligen installerat ett nytt teleskop som skall användas för att göra astronomiska observationer. För att på ett tillfredställande sätt kunna observera med teleskopet så krävs en hel del kringutrustning. En väsentlig del av den utrustningen är den kupol som teleskopet står monterat inuti. Projektet går ut på att med hjälp av en mikrokontroller samt diverse annan elektonik automatisera kupolens rörelser vid användande av teleskopet. För att det skall var möjligt att observera med teleskopet så finns det två ställbara luckor för att öppna kupolen. Dessa luckor, som styrs av två växelströmsmotorer, avgör hur stor spaltöppningen i kupolen skall vara för tillfället. Kupolen kan givetvis även roteras 360º horisontellt. Denna rörelse sker med hjälp av två andra växelströmsmotorer. Kupolen skall ompositioneras när den befinner sig utanför de gränser kring teleskopets koordinater som i förväg definierats och på så sätt synkroniseras med teleskopets rörelser. Projektet baseras på mikrokontrollern, teleskopets styrdator samt utrustning för att ompositionera kupolen. Kontrollern får kommandon från teleskopdatorn, tolkar dessa och skickar sedan signaler till motorernas reläsystem för att ompositionera kupolen.

Förord

Denna rapport utgör den skriftliga redovisningen av mitt examensarbete för programmet Elektroteknik (systemteknik - datorsystem) vid Uppsala Universitet.
Arbetet har utförts på astronomiska observatoriet vid institutionen för astronomi och rymdfysik i Uppsala under perioden mars till augusti 2004. I Elektoingengörsutbildningen ingår endast ett 10 poängs arbete, men när det var lättare att hitta ett på 20 poäng så beslutade jag mig för att göra det i stället

Jag vill här ta tillfället i akt att tacka följande personer:
mina handledare Nikolai Piskunov, Zahm Ottosson och Johan Warell vid astonomiska observatoriet, Sven-Erik Jansson vid institutet för rymdfysik, min ämnesgranskare Kjell Staffas och examinatorn Nora Masszi vid institutionen för signaler och system. Jag vill även rikta ett särskilt tack till min kursare och kamrat Alexander Åhman som har varit till stor hjälp vid programutvecklingsingsarbetet.

Kort om institutionen

Bild 1. Astonomiska observatoisiet

Astonomiska observatoriet är beläget på institutionen för astronomi och rymdfysik som i sin tur ligger på Ångströmlaboratoriet. Avdelningen för astronomi, som jag har varit på, finns i hus sex våning tre. Själva observatoriet med kupol och teleskop, som för övrigt syns på bilden ovan, finns i änden av taket på själva huvudbyggnaden. Observatoriet har vid tillfället för mitt examensarbete inte satts i fullt bruk utan var inne i någon sorts installationsfas.

Innehållsförteckning

1. Inledning

1.1 ***Bakgrund***

Jag heter Pär Hedblom och har under perioden 2002 – 2004 studerat på Elektroingenjörsprogrammet vid Uppsala universitet. Under perioden Mars till Augusti 2004 avslutade jag denna utbildning genom att utföra det examensarbete som du nu håller i din hand. Arbetet utfördes på astronomiska obervatoriet vid institutionen för astronomi och rymdfysik vid Uppsala universitet.

1.2 ***Syfte***

Syftet med examensarbetet är att studenten skall få möjlighet att pröva sin förmåga att arbeta som ingenjör samt få tillfälle att knyta kontakter med eventuella framtida arbetsgivare. Det skall även vara en första kontakt med de arbetsmiljöer och arbetsuppgifter som studenten i framtiden kommer att ställas inför. Examensarbetet kan också vara en möjlighet för arbetsgivare att på ett smidigt sätt rekrytera ny personal. Samtidigt som de ser vad studenten går för så får de ett arbete utfört till ett reducerat pris.

1.3 ***Projekt/Rapport-beskrivning***

Med tanke på att detta projekt har varit relativt praktiskt till sin natur så har mycket tid gått till att sätta sig in i problemet, studera manualer, lära sig nya program samt en hel del praktiskt arbete som montering och felsökning. Rapporten kan därför tyckas lite snäv eller knapp. Den tar dock upp det mesta som rör mikrodatorers uppbyggnad samt programmets konstruktion men ägnar mindre tid åt att beskriva endel övriga tekniska detaljer som radiokommunikation, trefasmotorer och motorstyrning m.m., som egentligen inte ingick i min uppgift men ändå blev en del av den.

1.4 Problemformulering

Institutionen för astronomi och rymdfysik behövde hjälp med att utveckla, implementera samt dokumentera ett system för automatisering av deras teleskopkupol. Innan arbetet utfördes så styrdes teleskopet och kupolen av separata system, eller kupolen styrdes egentligen helt manuellt med hjälp av strömbrytare på motorerna.Uppdraget var att rationalisera bort dessa strömbrytare och konstruera ett automatiskt system för att synkronisera kupolens och teleskopets position.

Examensarbetet gick som sagt ut på att konstruera ett system för automation av teleskopkupolen som hör till det nya BW-teleskopet*.
Teleskopet kontrolleras av två datorer, en för själva styrningen av teleskopet och en för att göra astronomiska observationer samt samla mätdata. Förutom styrdatorn till teleskopet så används två mikro-kontrollers för styrning av kupolen. Systemet baseras i huvudsak på dessa programmerbara inbyggda mikrokontrollers (RabbitCore RCM3200/RCM3220).
För att kunna rotera och öppna /stänga kupolen används fyra stycken växelströmsmotorer. För att hålla reda på kupolens position används en enkoder (pulsgivare) samt en magnetisk givare för detektering av nolläge (kalibrering).

För att teleskopet skall kunna används på ett tillfredtällande sätt måste öppningen (spalten) i kupolen alltid följa teleskopets position. Öppningen i kupolen kan, genom två ställbara luckor, varieras mellan 5 och 87 graders altitud och hela kupolen kan roteras 360º.

Teleskopets styrdator skickar med ett förutbestämt intervall ut koordinater, i form av altitud och azimut, via serieporten. Mikrokontrollerns uppgift är att med hjälp av dessa koordinater styra kupolen till önskat läge.

1.5 Mål

Målet är att skapa ett helt automatiskt system för positionering av kupolen i förhållande till hur teleskopet är riktat. Spalten i kupolen skall alltid befinna sig i fältet för teleskopet. När teleskopet närmar sig kanten på spalten så ompositioneras kupolen.

Samt att jag skall bli behörig att ta ut en examen.

*1.6 **Problem /Hinder***

Motorerna för att styra de båda luckorna sitter monterade på kupolen och snurrar därför med när kupolen rör sig. Matningen till dessa motorer sker via ringar som sitter moterade längs kupolens insida. Dessa ringar kopplas i sin tur ihop med motorerna via släpkontakter.

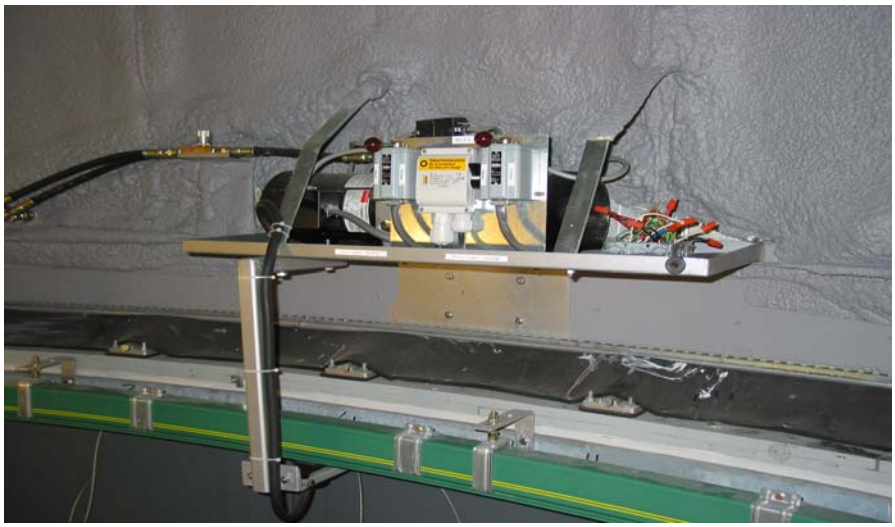

Bild .2 Motor som roterar med kupolen

Styrningen av dessa motorer skedde innan ombyggnaden manuellt via treläges vippströmbrytare som satt monterade på motorerna (se bild 18). I dag, efter ombyggnaden, sköts till och från-slag av motorerna med hjälp av kontrollern och ett reläsystem, via radiokommunikation . Huvudkontrollern sitter monterad i ett skåp på kupolens fundament, och slavkontrollern sitter i ett skåp på insidan av kupolen. Dessa två moduler kommunicerar med hjälp av radiomoduler.
Detta var i början, innan beslutet att installera radiostyrning, ett av de större problem jag stötte på under projektets gång. När jag väl hade radiomodulerna så uppstod en hel del problem innan jag slutligen fick dem att kommunicera med varandra. Förutom att de första moduler jag fick var felprogrammerade från leverantören så visade det sig att dom va mycket känsliga för hur antennen va kostruerad. Detta tog en stor del av tiden att konstatera och var en del av arbetet som jag inte hade räknat med innan jag började.

1.7 Alt-Azmonterat Casselgrainteleskop

Teleskopet är 90 cm i diameter och designat av Casselgrain. Det är monterat i en gaffel med så kallad Alt-Az* montering. För att styra teleskopet samt för att skicka koordinater och kommandon till kupolen används programmet AutoSlew.

Bild .3 Alt-Azmonterat Cassergrain teleskop

*Alt-Az = Altitud -Azimut

För ytterligare information angående teleskopet se bilaga.

1.8 ***Ash-dome***

Kupolen är levererad av ett företag som heter Ash-dome och är 6,24 meter i diameter och 2,8 meter hög.

Den övre luckan öppnas resp. stängs på ca: 145sek
Den nedre luckan öppnas och stängs på ca: 65sek
Att rotera kupolen kupolen ett helt varv tar ca: 133sek

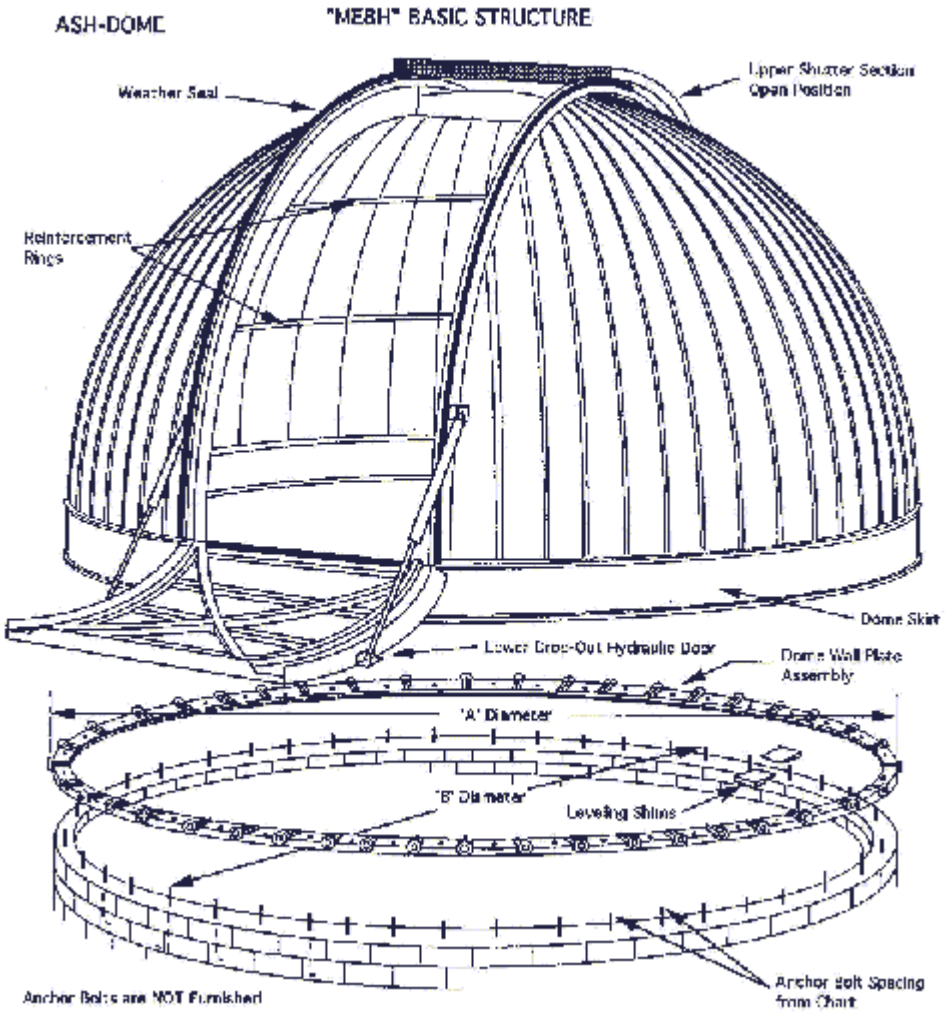

Bild .4 Ash-dome

Vi börjar med en generell genomgång av de viktigaste delarna i en dators hårdvarumässiga uppbyggnad, sedan övergår vi till att studera hur en mikrodator är konstuerad för att slutligen titta på den specifika mikrokontrollern som har använts i projektet.

2. Generell datorarkitektur

För att förstå hur en dator fungerar, kan vi titta på en mycket förenklad modell av en dator, se figuren nedan:

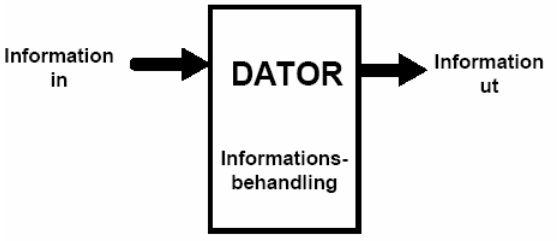

Bild 5 förenklad modell av en dator

Man kan säga att det är denna modell vi ser dagligen omkring oss. Med andra ord ser vi alltså inte särskilt mycket av själva datorn, utan märker istället att den gör något. Ta exempelvis en modern bilradio med RDS. Vi knappar in önskad station, och mikrodatorn ser till att högtalarna så bra som möjligt återger det vi önskar. Vad som finns inuti behöver inte den vanlige radiolyssnaren veta. En dator kan beskrivas som en maskin, oftast byggd med elektronik, som kan ta emot information, behandla den och sedan ge ifrån sig annan information. Informationsbehandlingen bör vara någorlunda intelligent, dvs. datorn ska naturligtvis utföra något nyttigt. Kännetecknande för en dator är att informationsbehandlingen styrs av ett program och just detta är till stor fördel när maskinen ska konstrueras. Om vi nu tar en noggrannare titt på innehållet i en dator så kan vi göra en naturlig uppdelning i tre delar: in- och utenhet, centralenhet (CPU) och minne.

2.1 In- och Utenheten

In- och Utenheten som i figuren nedan betecknas I/O-enhet (eng.: In/ Out), är datorns gränssnitt mot omvärlden, och det är detta användaren kommer i kontakt med när han använder datorbestyckade konstruktioner.

2.2 *Minnet*

Minnet är uppdelat i ett programminne och ett så kallat arbetsminne. I programminnet finns en arbetsbeskrivning för CPU:n. Arbetsbeskrivningen är uppbyggd av instruktioner som gör att datorn kan behandla den inkommande informationen steg för steg och ge ifrån sig önskade utsignaler. Arbetsminnet är ett minne för både skrivning och läsning och används av programmet bland annat för lagring av insamlade och beräknade värden.

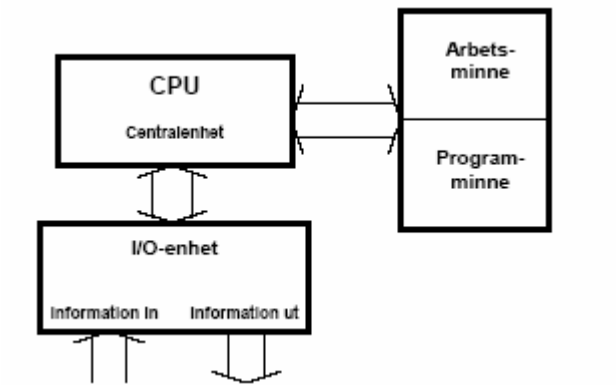

Bild .6 CPU, mine och I/O-enhet

CPU:n och minnet kan vara gemensamma för många vitt skilda instruktioner. Detta innebär att en och samma dator kan utföra många olika uppgifter.

Vad måste då CPU:n innehålla för att den ska kunna göra datorn till en användbar maskin?

Om man ger den förmågan att göra någon sorts beräkningar och jämförelser och sedan utifrån dessa kan fatta beslut genom att kunna välja bland alternativ, så bör man kunna få något uträttat. Den måste naturligtvis också kunna läsa de instruktioner man givit den. De två viktigaste delarna är därför en beräkningsenhet (aritmetisk/logisk enhet) och en instruktionsavkodare.

2.3 CPU

CPU:n eller centralenheten innehåller själva 'hjärnan' i datorn och det är här som det logiska arbetet utförs. CPU är förkortning för Central Processing Unit.

Datorns arbete sköts av CPU:n som innehåller ett antal urskiljbara delar:

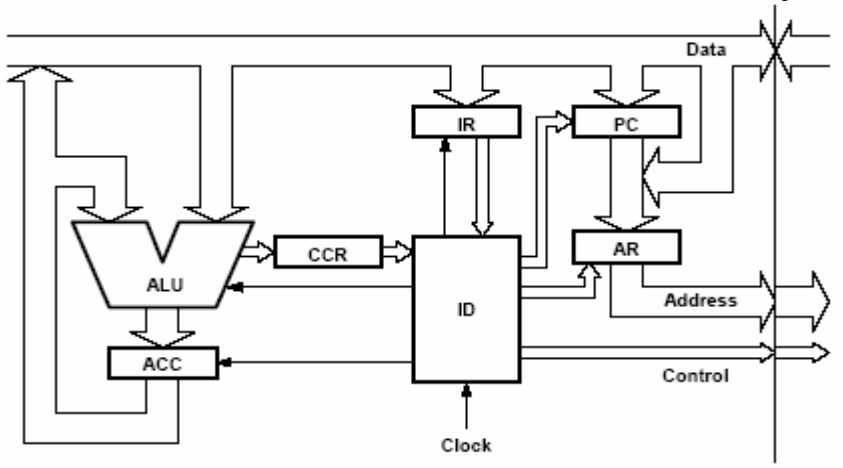

Bild .7 CPU

Bilden ovan visar hur en enkel CPU kan vara uppbyggd. Normalt är den mer komplex men här nöjer jag mig med att presentera de mest väsentliga delarna.

.

2.3.1 PC, Program Counter, programräknare

Programräknaren används till att peka ut instruktion för instruktion i det program som körs. Detta sker genom att programräknarens innehåll läggs ut på adressbussen som är kopplad till programminnet.

2.3.2 IR, Instruction Register, instruktionsregister

Här hamnar koden som bestämmer vad varje enskild instruktion ska utföra. Till instruktionen hör ibland en adress eller ett värde. I så fall laddas dessa in i AR eller ALU (se nedan). Varje instruktion initierar en sekvens av händelser i instruktionsavkodaren.

2.3.3 ID, Instruction Decoder. Instruktionsavkodare

En kod från IR startar en serie händelser som styr vad varje instruktion ska utföra. Denna kod läses in från programminnet via databussen. Händelser skapas genom att instruktionsavkodaren är byggd som en sekvensmaskin med ett antal insignaler och ett antal utsignaler. Koden från IR kan alltså kombineras med exempelvis information från flaggregistret,CCR, för att kunna utföra ett villkorligt hopp. Vi ser att styrsignaler går från instruktionsavkodaren till andra delar av CPU:n. Klocksignalen ser till att sekvensmaskinen uppdaterar sitt tillstånd med en viss frekvens. När instruktionen är utförd hämtas automatiskt nästa, varefter denna utförs. En dator arbetar alltså i 2-takt: hämta, utför, hämta, utför osv.

2.3.4 ALU, Arithmetic Logic Unit. Aritmetisk/logisk enhet

Här sker datorns beräkningsarbete på det sätt som instruktionsavkodaren bestämmer. Det kan tex. vara en subtraktion eller en logisk eller-funktion. Beräkning sker alltid mellan värdet i ackumulatorn (ACC) och ett värde i minnet.

2.3.5 ACC, Accumulator. Ackumulator

Resultatregister för ALU:n. Det uträknade värdet hamnar här.

2.3.6 CCR, Condition Code Register. Flaggregister

Flaggregistret avspeglar resultatet så att tex. likhet kan detekteras. Tillsammans med signaler från instruktionsregistret styr de instruktionsavkodaren.

2.3.7 AR, Address Register. Adressregister

Detta register innehåller en adress som pekar ut en speciell plats i det externa minnet. Adressen kommer antingen från PC (om den ska peka ut en instruktion) eller laddas in från databussen (om den ska peka ut en plats där tex. en skrivning ska ske).

3. Vad är en enchipsdator?

En dator i en krets med inbyggda minnen och periferienheter kallas enchipsdator eller mikrostyrkrets. En enchipsdator utgörs generellt av ett antal enheter som vi går igenom nedan.

3.1 ***Watchdog***

En watchdog är en speciell timer som oftast används för att starta om datorn då programmet har hängt sig. Det är en mekanism som ser till att programmet återstartas med en reset om man inte kvitterar den genom att utföra ett par speciella skrivningar till ett speciellt register. Den kommer att utföras till exempel om man fastnar i en oändlig loop där denna kvittering saknas.

3.2 ***Timer/räknare***

Timers eller räknare är vanliga binärräknare som utgör tidbas för alla enheter som skall kunna räkna händelser, mäta tid etc. Den går inte att stoppa, men kan programmeras att arbeta i olika hastigheter. En ”overflowfunktion” gör att en 8-bitars räknare lätt kan skarvas ihop till exempelvis 16 eller 24 bitar.

3.3 ***Portar***

Mikrodatorns kontakt med omvärlden.

Kretsar som sköter kommunikationen in och ut ur datorn kallas för I/O (Input/Output). Dessa kretsar kan innehålla en eller flera portar som i princip fungerar som ett register. Den data som skickas in eller ut ur datorn passerar detta register. Processorn kan skriva eller läsa på samma sätt som till en minnescell.

Rabbiten(RCM 3000) har sju stycken 8-bitars parallellportar. Pinnarna som används för parallellportar är också delade med ett antal andra funktioner som seriekommunikation, pulsbredds modulering m.m. Dessa portar konfigureras, för att fungera för önskat ändamål, genom att sätta ett antal kontrollregister.

3.4 ***Avbrottshantering***

Ett avbrott kan avbryta en programkörning då någon speciell händelse inträffar. Vid avbrottsstyrd I/O utnyttjas avbrottssignaler. Avbrott är en inbyggd funktion som gör det möjligt att bygga upp programdelar som fungerar oberoende av det övriga programmet. Med ett avbrott kan programkörningen styras bort från sin normala verksamhet till en bestämd subrutin när en speciell händelse inträffar. En händelse som absolut inte får missas kan tas omhand direkt. Miljön kring en mirokontroller innehåller ofta sådana händelser.

3.5 Polling

Polling är den kommunikationsmetod som är enklast att implementera. Denna metod kallas också programmerad I/O. För att processorn ska få reda på om I/O-kretsen behöver betjäning så måste den hela tiden fråga I/O-kretsen om detta, till skillnad från avbrottsstyrd I/O där avbrottet själv meddelar när nått skall utföras.

3.6 Stackpekare

En del av minnet används som en stack. Där kan man tillfälligt lagra data. Stacken är speciellt användbar vid subrutinsavbrott (se Avbrottshantering) m.m. En stack är av typen LIFO (Last In First Out). Detta innebär att den data som senast lagts på stacken också är den man först måste plocka bort. Den pekare (adressen i datorns minne) som pekar ut toppen på stacken kallas stackpekare.

3.7 In och utmatning

Kretsar för in och utmatning, det vill säga I/O-kretsar, sköter kommunikationen in och ut ur datorn. Denna kommunikation sker via portar(se 3.3). Man skiljer på inportar och utportar. En port är ett antal pinnar som är kopplade till ett register i I/O-kretsen. Inmatning sker genom att signaler som kommer på inportens pinnar hamnar i inportens register. Mikroprocessorn kan sedan läsa av detta register.

3.8 Seriekommunikation

Seriekommunikation kan ske antingen synkront, där bitarna klockas ut/in med hjälp av en timer, eller asynkront utan timer. Synkron dataöverföring kan ske i mycket högre hastighet än den asynkrona men den asynkrona överföringen kan i stället användas över längre sträckor.

3.8.1 RS-232

RS-232 är en standard som ofta används vid seriell dataöverföring. Den bygger på att man använder sig av två UART:s. UART:en sitter i datorn och sköts av ett antal kontrollregister. Dessa register måste ställas inför att kommunikation skall kunna ske mellan dem. Kommunikation sker med ett speciellt protokoll där man i förväg har bestämt hur man skall koda meddelanden, spänningsnivåer för etta respektive nolla m.m. Med seriell kommunikation menas att man med en ledning skickar endas en bit åt gången mellan enheterna. En timer används för att klocka ut bitarna på utgången eller för att ta emot bitar på ingången. En överföring består av en startbit, åtta databitar, eventuell paritetsbit samt en eller två stoppbitar. Starbiten representeras med en nolla medan stoppbiten representeras av en etta. Pariteten indelas i jämn och udda paritet. Udda paritet innebär att paritetsbiten blir ett om ett udda antal ettor skickas och jämn paritet motsvarars om jämt antal ettor skickas.

3.9 ***Bussar***

Bussar används för att skicka data mellan olika enheter. Det finns olika typer av signaler, grupperade i bussar t.ex. databuss & adressbuss. En buss kan ses som en grupp parallella signaler som på något sätt är relaterade till varandra.

De två vanligaste bussarkitekturerna är:

> *Von Neuman:* Alla typer av minnen och övriga komponenter delar på samma adress- och databuss.
>
> *Harvard:* Flera parallella adress- och databussar. Vanligast är att programminnet har egna bussar och att dataminne och övriga komponenter har delat minne.

Antalet ledningar i adressbussen avgör hur många minnesceller eller andra enheter som CPU:n kan hantera eller hur många bitar som kan flyttas samtidigt till eller från adresserad enhet.

Vanligt i enchipsdatorsammanhang är en adressbuss på upp till 16 bitar och en databuss på 8 bitar.

3.10 ***Minnestyper***

3.10.1 **Programminne/arbetsminne**

Innehållet måste bevaras utan strömförsörjning.
De vanligaste typerna av programminne är:

> **ROM**(Read Only Memory)
> Tillverkas med ett visst minnesinnehåll. Programminnet ligger i den del av adressrymden som har de högsta adresserna.
>
> **EPROM**(Erasable Programmable ROM)
> Program kan lagras med en speciell EPROM-brännare genom att ettor kan ändras till nollor där så krävs. Radering görs med UV-ljus.
>
> **FLASH**
> Kan programmeras och raderas elektriskt. Brännaren byggs ofta in i chipet. Radering görs oftast i stora minnesblock. Kan raderas ca 1000 gånger innan de är förbrukade.

3.10.2 **Dataminne**

Läsning och skrivning bör gå lika snabbt och i samma hastighet som CPU:n.
De vanligaste typerna av dataminne är

SRAM (Static Random Access Memory)
Minnesinnehållet lagras i D-vippor och bevaras så länge strömförsörjning finns.

EEPROM (Electrical Erasable Programmable ROM)
Skrivning tar längre tid än läsning. Minnesinnehållet bevaras dock utan spänning.

3.11 **Programmeringsspråk**

En enchipsdators eget språk är lågnivåspråket Assembler. Genom att skriva applikationer direkt i Assembler kan man få effektiva program där man hela tiden har full kontroll på vad som händer och fulla möjligheter att utnyttja alla resurser.
När Assembler ligger relativt långt från människans sätt att tänka så används i stället vanligen någon form av högnivåspråk. Det vanligaste språket bland högnivåspråken är C. Detta beror främst på att C är relativt effektivt och ger kort och snabb kod.

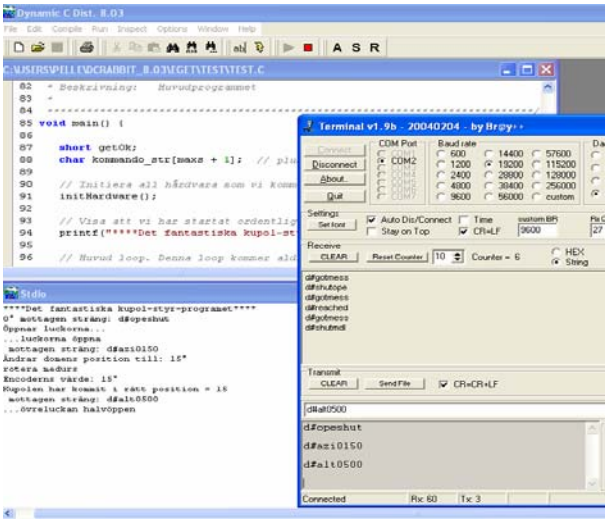

Bild .8 DynamicC

C är det högnivåspråk som har lägst nivå och ligger närmast Assembler. Detta gör att koden kan skrivas hårdvarunära och därför ej blir så prestandakrävande.

3.12 **Multitasking**

De flesta inbyggda systemapplikationer består av flera deluppgifter som skall utföras ”parallellt” (ett tecken skall tolkas samtidigt som en lucka skall öppnas). Deluppgifterna kan var mer eller mindre tidskritiska, vilket i stor utsträckning styr programstrukturen. Tidskritiska deluppgifter måste utföras inom ett snävt tidsintervall. Tidskritiska deluppgifter kan vara så väl periodiska som icke periodiska. Oftast finns både periodiska som icke periodiska i samma applikation.
Icke tidskritiska uppgifter kan utföras när man får tid över. Programmet måste konstrueras så att tid som blir över automatiskt används för dessa uppgifter.

4. RabbitCore

Vi skall nu övergå till att studera den specifika mikrodatorn som har använts i projektet. RCM3200/3220 skiljer sig något från vad man generellt brukar kalla enchipsdatorer. RabbitCore-moduler är snarare enkortsdatorer, med processor minnen socklar och kontakter monterade på ett litet kretskort, än enchipsdator där allt är inbyggt i en kapsel.

4.1 ***RabbitCore RCM3200***

Detta projekt baseras på Z-worlds mikrodatormodul RCM3200. Det är en 8-bitars processor som arbetar i 44,2MHz**.**
RCM3200 är en avancerad modul som är bestyckad med gott om såväl flashminne som statiskt RAM-minne. Den är utrustad med serie- och parallellportar samt möjlighet att ansluta den till ett nätverk via en 10/100Base-T Ethernetport

Bild .9 RabbitCore-modul RCM3200

Modulen är konstruerad för att sitta på ett egetkonstruerat moderkort som förser modulen med ström samt innehåller interface för anslutning av kringutrustning till I/O portarna.

För styrningen av spaltluckorna har det använts en något enklare RabbitCore-modul utan ethernet anslutning (RCM3220).

4.2 Blockschema över RabbitCore 3000 processorn

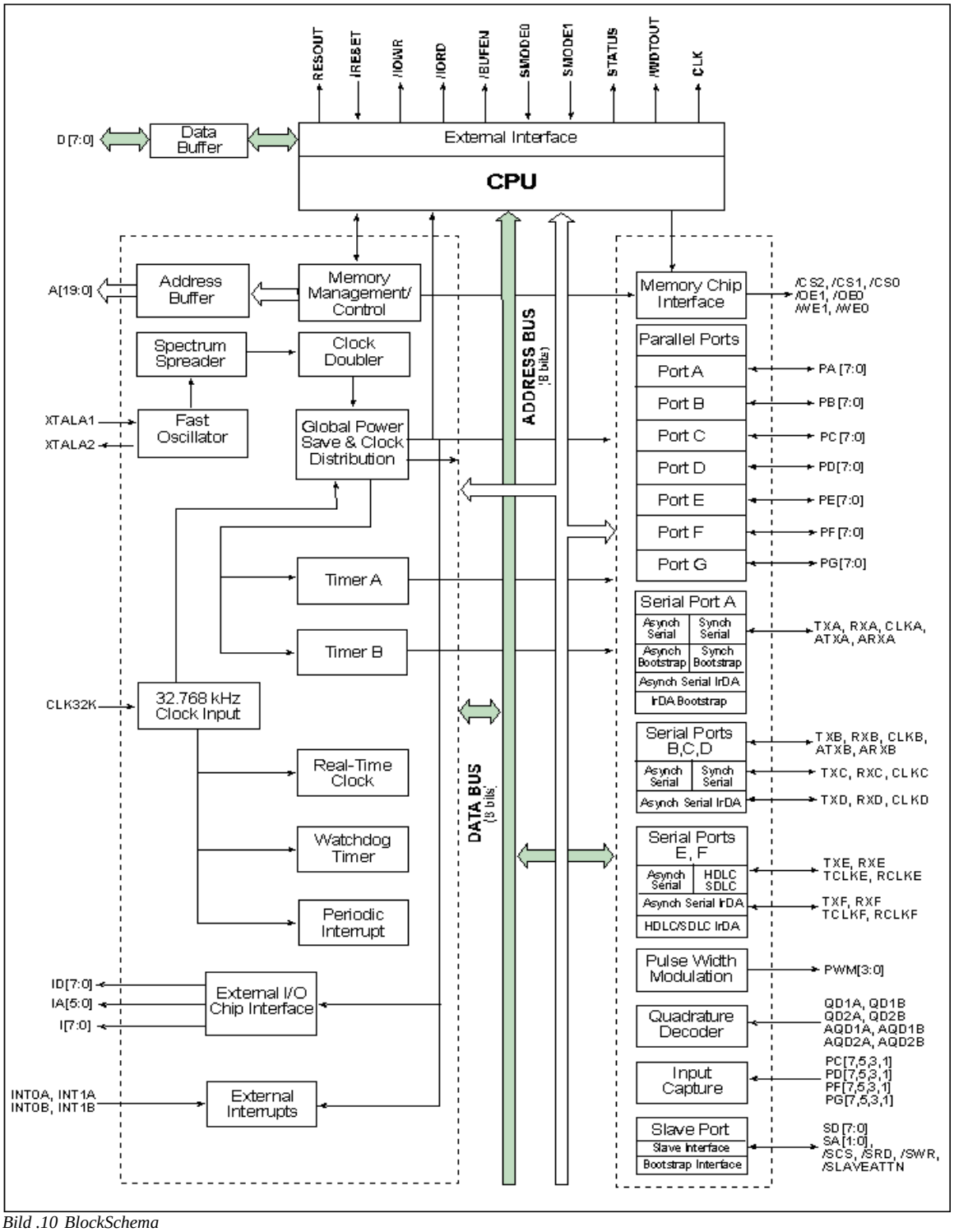

Bild .10 BlockSchema

5. Kringutrustning

Kupolens azimut ges av en enkoder (pulsgivare) som sitter monterad på motoraxeln till en av motorerna som roterar kupolen. Motoraxeln roterar 75 varv på ett varv på kupolen och enkodern ger 500 pulser per varv. Detta ger 75×500 = 37500 pulser/varv. Detta är dock betydligt högre upplösning än vad som behövs vid denna applikation. Därför skalas värdet ner, till att kunna visa 360 jämna grader per varv, i programvaran.

5.1 ***Roterande inkrementell pulsgivare***

Tillverkningstekniken för optiska pulsgivare har utvecklats snabbt. Det innebär att pulsgivare med hög upplösning har blivit billiga.
I digitala system används signalen från givaren som den är, i analoga via en D/A-omvandlare som ger pulståg ifrån sig vid rörelse. Roterande pulsgivare kan ge mellan några få och många tusen pulser per varv. De har ofta två kanaler vars pulståg är förskjutna 90 grader. På så sätt kan elektroniken känna åt vilket håll pulsgivaren roterar. De har dessutom ofta en nollpuls (indexpuls). Denna tredje kanal består av en puls per varv och används som referenspunkt. Indexpulsen är lika smal som de andra kanalernas pulser eller smalare. Indexpulsen på en pulsgivare med 2 000 linjer/varv är därför inte bredare än ca 1/8 000-dels varv. Indexpulsen ger därför en hemmaposition med hög precision och repeternoggrannhet.

5.1.1 **Pulståg från inkrementell pulsgivare**

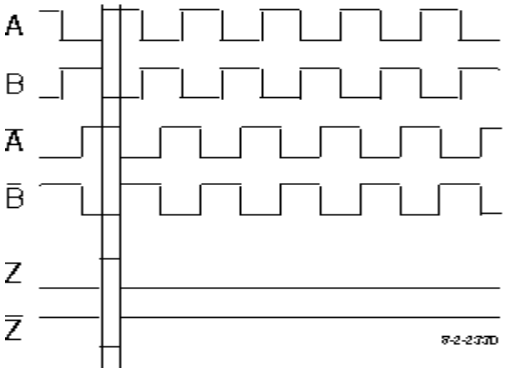

Bild .11 Pulståg från enkoder

5.1.2 Uppbyggnaden hos en inkrementell pulsgivare

Ljuskällan i pulsgivare består oftast lysdioder och pulsgeneratorn av en ljuskänslig krets. Den slitsade skivan består antingen av en stansad metallskiva eller av ett genomskinligt material som belagts med en film med slitsat mönster. Plast-, mylar- och glasskivor används.

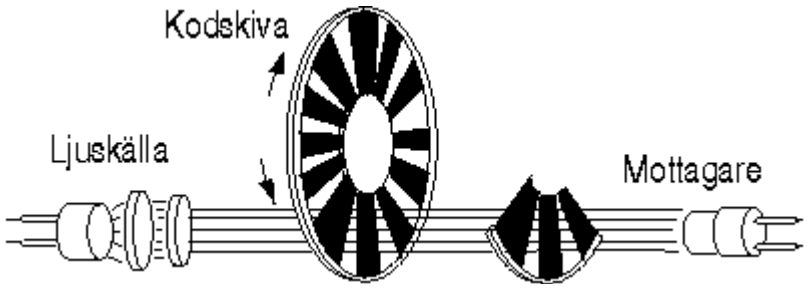

Bild .12 Uppbyggnaden hos en enkoder

Det finns pulsgivare som ger olika typer av signaler. I utrustning med små avstånd och låga störnivåer används ofta vanlig 5 V TTL-logik. I industriella sammanhang används i allmänhet differentiell 5 V-logik eller 24 V-logik.

5.2 *Nollägesgivare*

För att kunna kalibrera azimuten användes en magnetisk givare som sitter monterad i kupolen. Den sitter i parkeringsläget där azimuten 0 grader är fördefinierat. Varje gång kupolen passerar nollägesgivaren nollas värdet på quadraturenkodern och värdet på dome.realpos (i programkoden).

Bild .13 Magnetisk nollägesgivare

5.3 Radiokommunikation

Motorerna för att styra luckorna sitter monterade på insidan av den roterande delen av kupolen och kräver därför trådlös kommunikation för att kunna manövreras. Kommunikationen mellan basenheten och kontrollern till motorrelän för luckorna sker via radio. Radiokommunikationen baseras på en FM transiver där man skickar RS232 data till sändaren och får ut RS232 från mottagaren. Till transivern på motorerna är det ytterligare en mikrokontroller för att tolka den mottagna datan och utföra önska operation.

Bild .14 Radiomodul(transiver)

Transivern som används har beteckningen TRXQ1 och kommer från ett företag som heter rfSolutions. TRXQ1 transivermodulen kan upprätthålla tillförlitlig trådlös kommunikation för överföringshastigheter upp till 20Kbaud och avstånd upp till 200 m. Den har två kanaler och arbetar i 433.92MHz respektive 434.33MHz banden.

5.4 *Trefasmotorer*

Kupolen styrs med hjälp av fyra trefasmotorer. Två för att rotera kupolen samt en till vardera lucka.

Trefas asynkronmotorn är idag den vanligaste motorn i industritillämpningar. Detta tack vare att de är driftsäkra, billiga och kräver väldigt lite underhåll. De finns även inom ett väldigt stort effektregister från några få watt till megawatt storlek. Nackdelarna med denna typ av maskin är att de drar stora startströmmar, typiskt 6-8ggr fullastström, samt att de tidigare varit svårare att hastighetsreglera än DC-motorer.
Den vanligaste typen av asynkronmotor, den med kortsluten rotor, består utav en stator med lindningar och en rotor utav kopparstavar. Denna typ brukar kallas kortsluten burrotor på grund av likheten med en bur. Namnet asynkronmotor kommer av att rotorn roterar med ett asynkront varvtal mot statorns magnetfält (rotorn släpar efter).
Motorerna som sitter i kupolen är av märket Leeson.

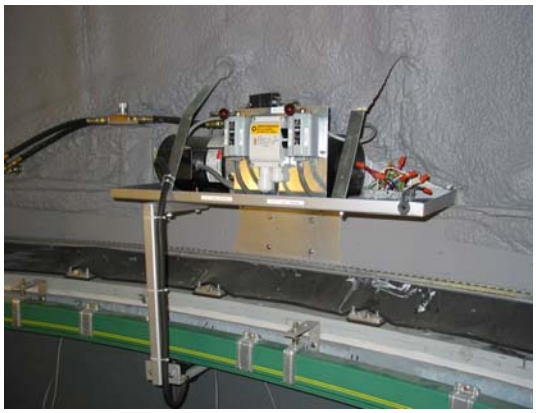

Bild .15 Motor för att öppna lucka

Bild .16 Motor för att rotera kupolen

Hur styrs motorerna?

Motorerna styrs av mikrokontrollern via ett reläsystem och en radiomodul.Det finns även möjlighet att köra motorerna manuellt via treläges vippströmbrytare i fall radiokommunikationen eller mikrokontrollern slås ut på ett eller annat sätt.

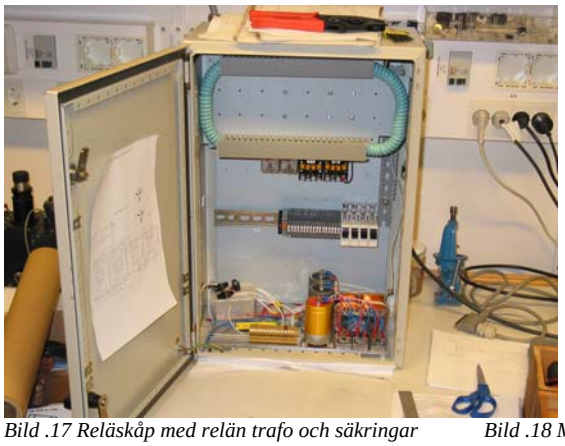

Bild .17 Reläskåp med relän trafo och säkringar

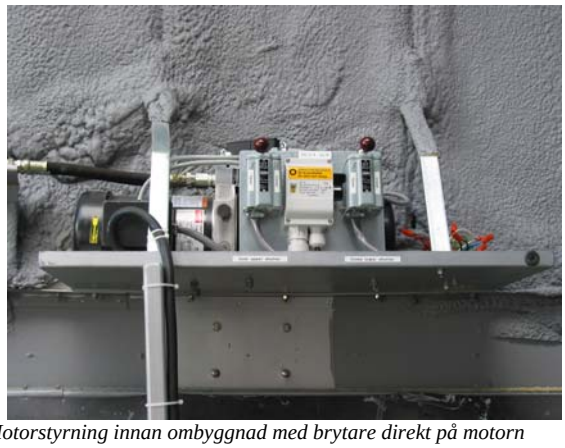

Bild .18 Motorstyrning innan ombyggnad med brytare direkt på motorn

5.5 ***Programvara: Autoslew, TheSky***

TheSky är ett program som används för astronomiska observationer och Autoslew är programmet som arbetar under TheSky för att styra teleskopet samt skicka kommandon till mikrokontrollern för kupolstyrningen.

6. Utförande

6.1 ***Projektplan***

Innan det praktiska arbetet påbörjades så arbetades en projektplan fram i samråd med handledaren. Under projektets fortsatta gång så följdes planen med undantag för vissa modifieringar, vid väntetid för leverans av beställd utrustning. Projektplanen i sin helhet kan ses i appendix A.

6.2 ***Analys och metodbeskrivning***

Det tre första veckorna ägandes åt att studera uppgiften och lägga upp en plan för hur den enklast skulle utföras. Jag studerade hur det befintliga systemet med teleskop och kupol fungerade innan ombyggnaden och hur uppdragsgivaren vill att den skall fungera när projektet är utfört. Jag satte mig in i hur den tekniska kringutrustningen, som enkoder, nollägesgivare, reläsystem m.m. fungerar.

De tre efterföljande veckorna ägnades åt att skriva ett prototypprogram. Programmet utvecklades och testades i den utvecklingsmiljö (Dynamic C) som följer med ett utvecklingskit från Z-world. Motorernas rörelser simulerades genom att manuellt vrida på enkodern och signalerna från mikrokontrollern till motorerna simulerades genom att tända och släcka lysdioder.

På bilden syns modulen som sitter monterad på ett utvecklingskort, lysdioderna för att simulera motorerna samt enkodern för att detektera kupolens rotationsposition

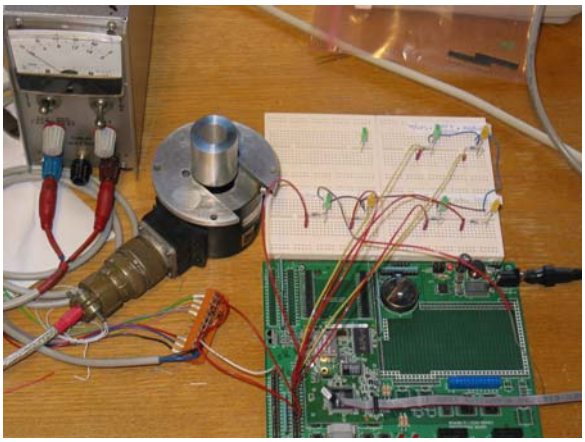

Bild .19 Pototypkoppling

Vecka sju och åtta väntade vi på leverans av radiomoduler och mikrokontroller och jag använde då i stället tiden till att påbörja skrivandet av rapporten.
Veckorna nio till elva ägnade jag är praktiskt arbete, som att montera transformatorer och relän, tillverkade fästen för kretskort och givare m.m.

Vecka elva och tolv ägnades åt att designa kretskortet (moderkortet) för rabbitmodulen, radiomodulen m.m. Samt att jag försökte sätta upp en prototypkopling för att testa radiomodulen. Efter fyra dagars labbande så ringde vi till tillverkaren för att få råd och tips. Då fick vi istället reda på att modulen var feltilverkad från början och fyra dagars arbete hade gått till spillo.
När vi slutligen hade all utrustning, och kopplade samman allt i labbet innan vi slutligen skulle installera systemet i kupolen, så kunde vi konstatera att mycket av utrusningen inte fungerade riktigt som jag hade tänkt. Det blev en hel del felsökning och en hel del modifieringar fick göras innan vi slutligen fick ett system som fungerade på ett någorlunda tillfredställande sätt.

6.3 Hur löste jag problemet?

Ett kommando skickas, till mikrokontrollern(Huvudmodulen) från teleskopets styrdator, via RS – 232. Meddelandet som kommer i form av en char-sträng tas emot av kontrollern och tolkas i funktionen parseCmd(); . Det mottagna kommandot jämförs sedan med ett antal kända kommandon. Är det ett giltigt kommando så returneras en nolla och sedan utförs det i CheckIO() , som är den funktion där allting utförs. Ifall det inkommande kommandot i stället inte stämmer överens med något av de kända kommandona så returneras i stället en etta (För mer ingående beskrivning se programkoden i appendix B).

6.4 Mikrokontrollern

Mikrokontrollern har i uppgift att ta emot kommandon, som kan vara för att öppna eller rotera kupolen (se tabell nedan), via en serieport och med hjälp av dessa kommandon styra kupolens motorer. För att kontrollern ”alltid” skall veta kupolens nuvarande läge används en enkoder för att ge feedback för azimuten samt en magnetisk givare för att kunna kalibrera enkodern. Spaltluckorna ger däremot ingen feedback till kontrollern utan styrs i stället med hjälp av delay och ändlägesbrytare. Jag har helt enkelt tagit tid på luckorna, och sedan styrs luckorna genom att köra motorerna en förutbestämd tid.

6.5 *Spaltluckorna*

Den övre luckan, som för övrigt är den som tar längst tid att öppna, har tre lägen: ett läge då den är stängd, ett läge då den står på gläntoch ett då den är helt öppen.
Då luckan är i läge ”stängd” och ska till läge ”öppen” startas motorn i riktning upp och körs sedan tills delayet har gått ut. För att alltid vara säker på att luckan är helt öppen så har delayet satts lite längre än den egentliga tiden för att öppna luckan, detta medför inga problem för motorerna då det sitter ändlägesbrytare monterade i både läge öppen och läge stängd. För att komma till läge ”mellan” körs motorerna på samma sätt med hjälp av delay, men man kan då inte vara lika säker på att ”mellan” alltid innebär exakt samma läge. Det kan hända att motorerna eller drivsystemet i övrigt fungerar olika effektivt beroende på om det är varmt eller kallt ute eller andra yttre omständigheter. På den övre luckan sitter en klack som låser den nedre luckan. Därför måste man, för att överhuvudtaget kunna öppna den nedre spaltluckan, först öppna den övre spaltluckan cirka två decimeter så att den nedre luckan kan löpa fritt.

6.6 *Offset*

För att kupolen skall rotera så sällan som möjligt behövs en offset mellan teleskopets koordinat och koordinaten för mittpunkten på spalten. När kupolen får en ny koordinat från teleskopet skall den rotera tills teleskopet befinner sig i kanten på spalten i kupolen. Sedan kan man observera ända tills teleskopet når den andra kanten av spalten och måste då rotera kupolen igen. Om man observerar ett och samma objekt behöver kupolen uppskattningsvis rotera var 5:e minut.
Eftersom jordens rotationsriktning förhoppningsvis alltid är den samma och stjärnhimlen då alltid kommer att rotera så att observation sker medurs skall offseten implementeras så att teleskopet alltid hamnar i vänster kant av spalten, sett innefrån.

Det enklaste sättet att implementera denna offset är att ha parkerings/utgångs-läge för teleskop och kupol förskjutna med just så mycket man vill att offsetten skall vara. Då har man lämpligtvis kupolens nolläge förskjutet till höger om teleskopets. När teleskopet sedan skickar nya azimutkoordinater så ställer sig kupolen alltid i det läge där spaltens vänstra kant sammanfaller med just den koordinaten.
Detta visade sig vara en mindre bra lösning eftersom offsetten alltid blir till vänster om teleskopets koordinat. Detta resulterar i att när teleskopet ändrar position moturs så kommer teleskopet att titta in i kupolväggen i stället för att titta ut genom spalten. Detta problem löstes genom att i stället implementera en mjukvaru-offset som tar hänsyn till om kupolen roterar medurs eller moturs(mer om det kan läsas i den kommenterade koden).

Teleskopet skickar sedan en ny azimutkoordinat till kupolen endast om den nuvarande koordinaten är XX grader skiljd från den senast skickade, där XX lämpligtvis motsvarar vinkel given av spalten sedd från teleskopet.

6.7 *Kommandon mellan teleskop och kupol*

Autoslew kontrollerar om kupolen behöver ompositioneras. Under normala omständigheter kommer kupolen få ett nytt kommando från teleskopet ungefär var femte minut för att rotera spalten framför teleskopet. Detta beror givetvis på vilken altitud man för tillfället observerar.

Kommandon från teleskop till kupol	**Svar från kupolen till teleskopet**
”d#getshut” – för läget på luckorna just nu	”d#shutclo”- om kupolen är stängd ”d#shutmdl”- om kupolen står på glänt ”d#shutope” –om kupolen är helt öppen
”d#opeshut”- för att öppna kupolen	Kupolen svarar med ”d#gotmess” Efter att den har nått läge öppen så svarar den med ”d#shutope”
”d#closhut” – för att stänga kupolen	Kupolen svarar med ”d#gotmess” Efter att den har nått läge stängd så svarar den med ”d#shutclo”
”d#aziXXXX”- för att gå till given azimut	Kupolen svarar med ”d#gotmess” Och roterar sedan till önskad azimut-position. När den har nått positionen så skickar den ”d#reached” till teleskopet
”d#altXXXX” – för att öppna kupolen så att det går att observera på given altitud	Kupolen svarar med ”d#gotmess” Och kör sedan luckorna till önskad altitud-positon (beroende på hur man har definierat läge mitten). När den har nått positionen så skickar den ”d#reached” till teleskopet.
”d#getazim”- för att ge nuvarande azimuth	Kupolen svara med att skicka tillbaks nuvarande position, i hela grader.

All kommunikation mellan teleskopet och kupolen sker seriellt via RS-232, med baudrate 19200 utan paritet med 8 databitar och en stoppbit.

6.8 ***Programmering***

Programmet till kontrollern är skrivet i Z-Worlds egen version av C, Dynamic C.
C och Dynamic C är högnivåspråk som ligger på en relativt låg nivå för att komma nära hårdvaran och lätt kunna kommunicera med perifera enheter.

För att kunna hantera flera uppgifter(multitasking - läsa kommando från serieporten, skicka signaler till motorer m.m.) så nära parallellt som möjligt så har Dynamic C en språkkonstruktion som kallas ”costates” och ”cofunctions”. Dessa kan vara användbara när man, som i detta fallet, skall skriva ett program som skall köras på en processor som inte har hjälp av något operativsystem (ett så kallat ”naket” system).
Med hjälp av dessa kan man konstruera ett system som nästan kan sägas ha realtidsegenskaper.

Koden är uppdelad i fem olika delar header, main, initHardware, checkIO och parseCmd.

Den första delen är en headerliknande del som innehåller funktionsprototyper, variabel deklarationer samt makron.
Sedan följer en main-del som innehåller själva huvudprogrammet. Från denna del anropas sedan de resterande tre delarna.

I initHardware så initieras portar och lägesvariablerna för luckornas tillstånd nollställs.

I checkIO utförs alla kommandon som behöver vänta på att något ska uppfyllas/bli klart, det vill säga alla kommandon som inte ögonblickligen blir klara. Denna funktion måste anropas ofta och regelbundet för att kunna uppdatera lägesvariabler från givare med mera.

ParseCmd tar som argument en sträng och jämför denna med ett antal kända kommandon (se tabell nedan). Är det ett giltigt kommando kommer detta att utföras. Kommandot 'utförs' dock inte i ParseCmd, utan i checkIO(). Här sätts endast de variabler som får checkIO att utföra det valda kommandot. Om strängen beskriver ett giltigt kommando och detta blev utfört returneras en nolla och om inget giltigt kommando kunde hittas returneras en etta.

Huvudprogrammet ligger på huvudkontrollern som sitter i skåpet på fundamentet. Det programmet tar hand om inkommande kommandon från teleskopdatorn och tolkar dessa, utför det som kommandot innebär och skickar tillbaka en bekräftelse till teleskopdatorn. Denna kontroller skickar även vidare kommandon till kontrollern som styr spaltluckorna, om det är detta kommandot innebär.

Kontrollern för luckorna innehåller ingen CheckIo();-del utan utför kommandot direkt i ParseCmd()-delen. Den tar helt enkelt emot kommandon på serieport C och utför detta direkt genom att sätta motsvarande pinne, till reläsystemet, hög eller låg för att öppna respektive stänga luckorna. Den skickar heller ingen bekräftelse tillbaka till huvudmodulen utan det är en ren envägskommunikation

Kalibrering: Varje gång programmet startas om(vid strömavbrott eller om någon har tryckt på reset-knappen) så börjar kupolen automatiskt att rotera medurs. Den roterar till det att den har kommit till nollägesgivaren för att få en bekräftelse på att man verkligen befinner sig på rätt azimut.
Detta sker endast vid ny uppstart av programmet.
För ytterligare upplysningar se den kommenterade koden, appendix X

7 Slutsatser

7.1 ***Resultat***

Resultatet av detta projekt är ett fullt fungerande styrsystem för teleskopkupolen vid astronomiska observatoriet. Systemet fungerar enligt den förutbestämda kravspecifikationen

7.2 ***Diskussion***

När projektet påbörjades var målet att skapa ett system för att kupolen automatiskt skulle följa teleskopets position. Min uppgift var att skapa ett program till mikrokontrollern. Men allt efter projektets gång tillkom andra uppgifter så som: installation av reläsystem och enkoder, konstruktion av kretskort, installation av radiokommunikationsutrustning m.m. Även fast detta inte ingick i min uppgift så anser jag att det har varit mycket lärorikt och av stort såväl praktiskt som teoretiskt intresse att arbeta även med dessa uppgifter. Det är trots allt troligen så det kommer att fungera i min framtida yrkesroll.

En stor del av mitt arbete har gått ut på att samla fakta genom att studera facklitteratur och söka på Internet samt felsökning. Jag har även haft stor hjälp av personer utanför institutionen. Jag anser att det är en viktig egenskap i ingenjörens arbete att vara lyhörd och kunna ta hjälp av kunniga människor i sin omgivning.

7.3 ***Referenser***

- Föreläsningsanteckningar från kursen Mät & mikrodatorteknik med Lars Eriksson
- Mikrodatorer bit för bit av Rune Körnefors
- Vägen till C av Ulf Bilting och Jan Skansholm
- Samtliga manualer tillhörande RabbitCore 3000 och Dynamic C
- Bygg och programmera med enchipsdatorn Stefan Nyman

7.4 ***Problem under projektets gång***

I början var det svårt att få någon tydlig specifikation på hur uppdragsgivaren ville att systemet skulle fungera. Eftersom observatoriet ännu inte satts i bruk vid tidpunkten för projektet så fanns det en del svårigheter i att veta hur man skulle komma att använda det i framtiden. Även problemet med att kommunicera med motorerna till spaltluckorna ställde till en del problem innan beslutet att installera radiokommunikation fattades.
Det har som sagt även varit en hel del felsökning och modifieringar av utrustningen. Detta är en direkt följd av de misstag jag gjorde vid konstruktionen av systemet. Detta kan ju anses som fullt normala problem när det ändå var första gången jag arbetade med den här utrustningen och dessutom aldrig tidigare har konstruerat ett system av den här storleken.

7.5 ***Förslag till fortsatt utvecklingsarbete – förbättringar som kan/bör göras***

Med en mer avancerad motorstyrning kan man lätt få motorerna att köras mjukare. Med ett drivsystem som ger mindre vibrationer i byggnaden skulle det vara fullt möjligt att konstruera ett system som gör att kupolen kontinuerligt följer teleskopet, i stort sett steglöst, istället för som idag så sällan som möjligt. Det skulle då även vara nödvändigt att montera någon form av givare på luckorna, för att exakt kunna veta vart luckorna befinner sig och därmed ständigt kunna ge respons till huvudmodulen.

8 Slutord

Genomförandet av projektet har gett mig en hel del erfarenhet vad det beträffar planering, systemering, programmering, caddning m.m. Detta är erfarenheter som jag tror att jag kommer ha stor nytta av i min framtida yrkesroll. Jag är tacksam att institutionen för astronomi och rymdfysik har givit mig möjligheten att utföra mitt examensarbete och hoppas att dom i framtiden skall ha nytta av det arbete jag har utfört med att automatisera teleskopkupolen.

9 Bilagor

Appendix A
Tidsplan
Specific task and time table:

- Study of the telescope orientation principles, observational procedures and capabilities of the controller (based on telescope and controller description).
  week 1-3
- Setting up a simulator attached to a PC based on the controller prototyping board.
  Week 4-6.
- Programming the embedded controller to perform the following operations:
  1. Opening of the dome
  2. Closing of the dome
  3. Automatic following the telescope position

  week 7-11
- Develop a zero point calibration of the dome azimuth
  week 12-15
- Implementation and testing of the whole system
  week 16-18
- Writing documentation and final report
  19-20

Vecka 4 till 15 vart som en enda lång utveklingsperiod, där de två sista veckorna gick till att installera kringutrustningen istället för att utveckla nollpunktskalibrering som i stället utfördes på betydligt kortare tid i början av programutvecklingen. Vecka 15 till 19 bestod mest av felsökning och modifieringar av utrustningen. Vecka 19 och några veckor framåt gick åt till att installera systemet och färdigställa rapporten och dokumentationen.

Appendix B
Teleskopspecifikation

Position	Short Description	Price (SEK)
90cm telescope		**1.488.000**
Optics	- Lomo sitall optic - 900mm free diameter - 920mm mechanical diameter	
Tube	- Truss tube - Lasalle main mirror support - Dual Nasmyth foucus design	
Mount	- AltAZ mount - High accuracy friction servo-drives - External Heidenhain-encoders with <0,08”reslution	
Telescopecontrol system and software	- Win 98 PC, <700MHz with all necessarily PC cards, mouse and keyboard, Monitor for the TCS Autoslew - 32bit Win control software Autoslew - Microcontrolled weather station	
Eyepieces	- 75mm Rodestock Heligon with adapter - 35mm Panoptic - 21mm Pentax XL	**10.500**
Guiding Camera	- LISÄÄ Autoguider - 659×494 Pixel - 14 bit ADC - Control with Autoslew	**16.000**
Offset Guider	- Off-Axis Guider for f/5 focus - Off-Axis Guider for f/10 focus -	**9.000**
Dual Filter Wheel Nasmyth focus #1	- Custom made to hold 25kg - 2×12 positions for 48mm filter thread	**33.000**
Filter wheel for Nasmyth focus #2	- Custom made to hold 25kg - 1×12 positions for 48mm filter thread	**22.500**
Installation costs 90cm	- Including packing, transportation, insurance	**90.000**

Utdrag ur offerten i samband med leverans och installation av teleskopet.

Appendix C
Blockschema