



# Sommarstugekoll

*EDA234 - Digital konstruktion, projektkurs*

Grupp 1

Christian Isaksson

Jan Pettersson

Christoffer Öjeling

## Sammanfattning

Sommarstugekoll är en prototyp för avläsning av temperatur och reglering av element via SMS-kommunikation. Prototypen bygger på en FPGA, en extern temperatursensor och en mobiltelefon med serieinterface (RS232). Genom att skicka fördefinierade kommandon finns möjligheten att slå av/på element eller begära temperaturen. När ett sms tagits emot avkodas dess innehåll för att identifiera eventuella kommandon som getts till systemet och därefter utförs de. Prototypen avslutar med att skicka en status rapport till avsändaren av sms:et.

# Innehåll

<b>1</b>	<b>Systemspecifikation</b>	<b>1</b>
<b>2</b>	<b>Systembeskrivning</b>	<b>2</b>
2.1	Blockschema . . . . .	2
<b>3</b>	<b>Beskrivning av delblocken</b>	<b>3</b>
3.1	Styrenhet . . . . .	3
3.2	Temperatur . . . . .	4
3.2.1	DS18S20 . . . . .	4
3.2.2	Onewire . . . . .	6
3.3	7-Segmentdisplay . . . . .	12
3.3.1	segment-temperature . . . . .	12
3.4	Kommunikation . . . . .	14
3.5	Element . . . . .	19
<b>4</b>	<b>Hårdvara</b>	<b>21</b>
4.1	Digilent Nexys3 - FPGA . . . . .	21
4.2	DS18S20 - Temperatursensor . . . . .	21
<b>5</b>	<b>Felanalys</b>	<b>23</b>
<b>A</b>	<b>Kommandon</b>	<b>i</b>
<b>B</b>	<b>Signallista</b>	<b>i</b>
B.1	DS18S20 . . . . .	i
B.2	segment-temperature . . . . .	ii
B.3	Com . . . . .	ii
B.4	Styrenhet . . . . .	iii
B.5	Element . . . . .	iv
<b>C</b>	<b>Kretsschema</b>	<b>v</b>
<b>D</b>	<b>Syntesschema</b>	<b>vi</b>
<b>E</b>	<b>Komponentlista</b>	<b>vii</b>
<b>F</b>	<b>Pinlayout</b>	<b>viii</b>
<b>G</b>	<b>Programkod (VHDL)</b>	<b>ix</b>
<b>H</b>	<b>Arbetsredogörelse</b>	<b>x</b>
H.1	Christian Isaksson . . . . .	x
H.2	Christoffer Öjeling . . . . .	x
H.3	Jan Pettersson . . . . .	xi

## 1. Systemspecifikation

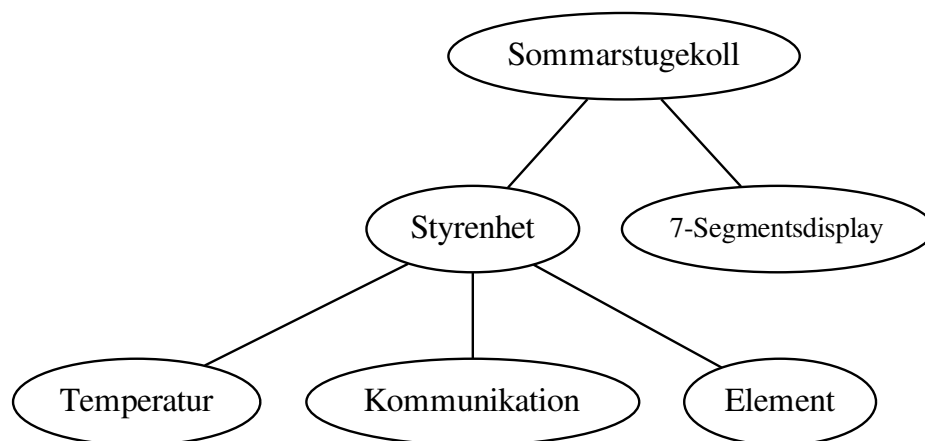
Sommarstugekoll är ett system för övervakning och uppvärmning utav sommarstugor. Systemet tillåter ägare att via sms få reda på inomhustemperatur och vilka element som är igång, samt möjligheten att slå av/på element i sommarstugan förutsatt att de är sammankopplade med detta system.

När systemet får ett sms analyseras innehållet i sms:et för att identifiera eventuella kommandon till systemet. De kommandon som skickats till systemet kommer att utföras ett i taget, vilket innebär att om sms:ets avsändare både vill veta inomhustemperaturen och slå av/på element kommer först temperaturen att hämtas och därefter regleras elementen. När alla kommandon utförts kommer huvudenheten att svara avsändaren med inomhustemperatur och alla elements nya status beroende på vilka kommandon avsändaren angett i sitt sms. Sen återgår systemet till vänteläget där det ligger och inväntar ett nytt sms.

I det fall då sms:et saknar giltiga kommandon raderas det och huvudenheten går direkt tillbaka till vänteläget.

Systemet består huvudsakligen utav tre delar, en huvudenhet, en extern temperatursensor och en mobiltelefon med serieinterface (*RS232*).

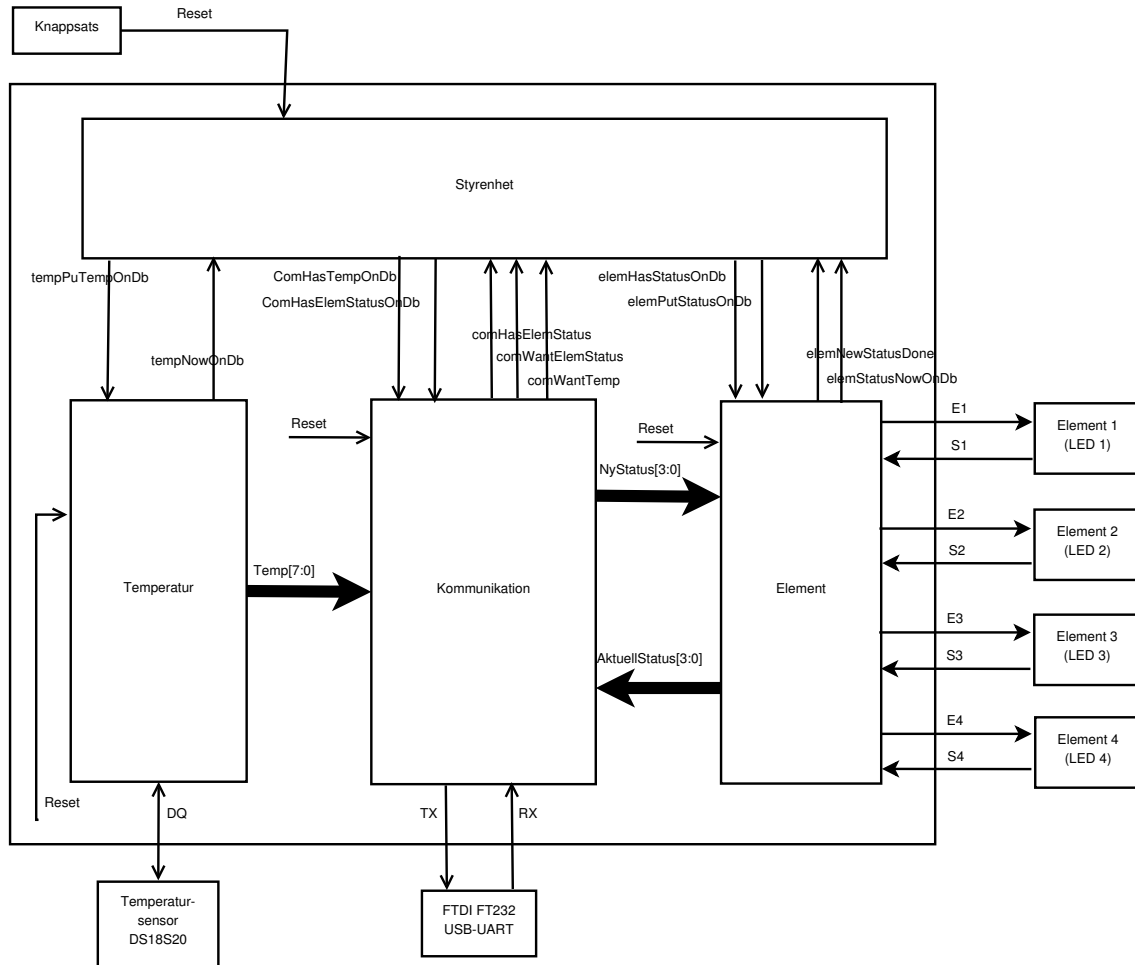
Huvudenheten är uppbyggd utav en *Field-Programmable Gate Array* (FPGA) innehållande all logik.



Figur 1: Övergripande struktur för projektet

## 2. Systembeskrivning

### 2.1. Blockschema



Figur 2: Övergripande blockschema för huvudenheten som visar alla styrsignaler inklusive datavägar mellan de olika blocken.

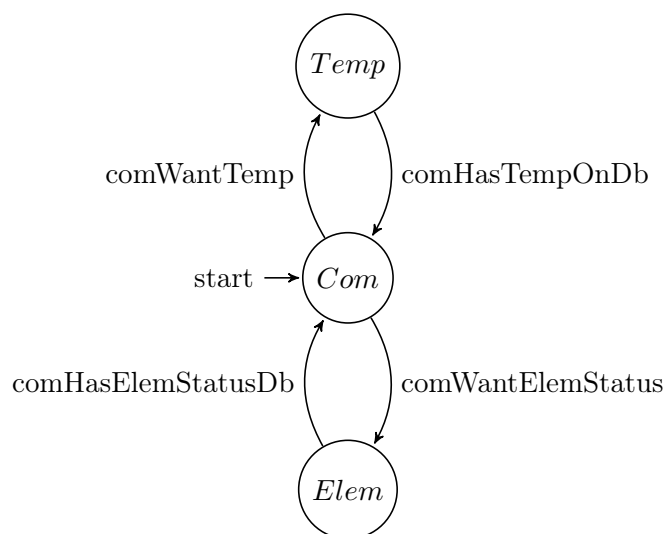
### 3. Beskrivning av delblocken

#### 3.1. Styrenhet

Styrenheten tar emot insignaler från tre de delblocken: *Element*, *Kommunikation* och *Temperatur*.

Utifrån mottagna insignaler bestämmer styrenheten med hjälp av sina styrsignaler vilket delblock som ska läsa respektive skriva till databussen och om ett delblock förväntas skriva till databussen anger även styrsignalerna vad som ska skrivas.

Styrenheten bygger på en tillståndsmaskin med tre tillstånd: **Com**, **Temp** och **Elem**, där vardera av de tre tillstånd är kopplade till ett, och endast ett, utav de tre delblocken.



Figur 3: Händelseförlopp för styrenheten.

Namnen på bågarna mellan olika tillstånd anger signaler som ettställs vid tillståndsförändringen.

För att förtydliga skillnaden mellan delblock, tillstånd och databussar i nedanstående stycken är delblocken markerade i *kursivt*, tillstånd i **fetstilt** och databussar i *kursivt fetstilt*.

I **Com** ligger styrenheten och väntar på kommandon från delblocket *Kommunikation*. Om en begäran att få veta inomhustemperaturen fås, kommer styrenheten att byta tillstånd till **Temp** och med hjälp av en styrsignal meddela delblocket *Temperatur* om att temperaturen ska inhämtas och skrivas till databussen *temp*.

I det fall en begäran fås om att ändra elementens status, dvs. slå av/på ett eller flera element, skriver delblocket *Kommunikation* den nya elementstatusen till databussen *nyStatus* och sedan meddelas styrenheten om att ny status finns att läsa på databussen *nyStatus*. Därefter ändras tillstånd till **Elem**.

Om enbart information om elements aktuella status begärs, sätts styrenhetens styrsignaler till att meddela *Element* att enbart aktuella element status behöver skrivas till databussen samtidigt som en tillståndsovergång till **Elem** sker.

I **Temp** väntar styrenheten på kvittens från *Temperatur* om att temperaturen skrivits till databussen *temp* och går att läsa.

När kvittens fått återgår styrenheten till tillståndet **Com**, samtidigt som en styrsignal meddelar delblocket *Kommunikation* att temperaturen nu finns att hämta på databussen *temp*.

I **Elem** väntar styrenheten på kvittens från delblocket *Element* om att elementen har reglerats och/eller att aktuell elementstatus finns på databussen *aktuellStatus*.

När kvittens fått återgår styrenheten till tillståndet **Com**, samtidigt som styrsignaler meddelar delblocket *Kommunikation* att elementen reglerats och/eller aktuell elementstatus finns att hämta på databussen *aktuellStatus*.

## 3.2. Temperatur

Logiken för avläsning av temperatur är uppdelad i ett antal delblock. Dels för att vara lättöverskådligt, men även så att man lätt ska kunna lägga till funktionalitet i efterhand. Slut användaren använder endast DS18S20 direkt. Onewire modulen är inte bunden till just DS18S20, utan kan även användas till andra enheter som använder sig utav 1-wire protokollet.

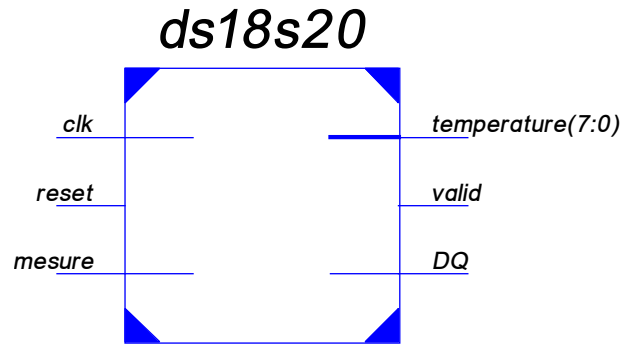
All logik använder sig utav VHDL-standardbiblioteket `numeric_std` för hantering av tal med och utan tecken.

### 3.2.1. DS18S20

**Interface** DS18S20 exponerar ett interface för mätning och avläsning av nuvarande temperatur. Mätningen initieras genom att `measure` sätts till '1'. När temperaturmätningen är klar kommer `valid` sättas till '1'. Då finns temperaturen att avläsa på `temperature` som ett binärt 8 bitars tal på tvåkomplementsform där bit 7–1 är heltalsdelen och bit 0 decimaldelen. `valid` fortsätter att vara '1' tills en ny mätning initieras genom att `measure` återigen sätts till '1'.

En temperaturmätning tar upp till 750ms.

**Implementation** Själva logiken i sig består av en cirkulär tillståndsmaskin (Figure 5). Tillståndsmaskinen växlar mellan att ge olika kommandon till onewire-blocket, och sedan vänta på att kommandot ska utföras. En mer utförlig beskrivning när data avläses och hur det samplas finns under 3.2.2 Onewire. DS18S20 har stöd för flera sensorer på samma buss. De delar då DQ och varje sensor har unikt serienummer för identifiering. I denna konstruktion används endast en sensor och logik för identifiering av flera sensorer utelämnas.



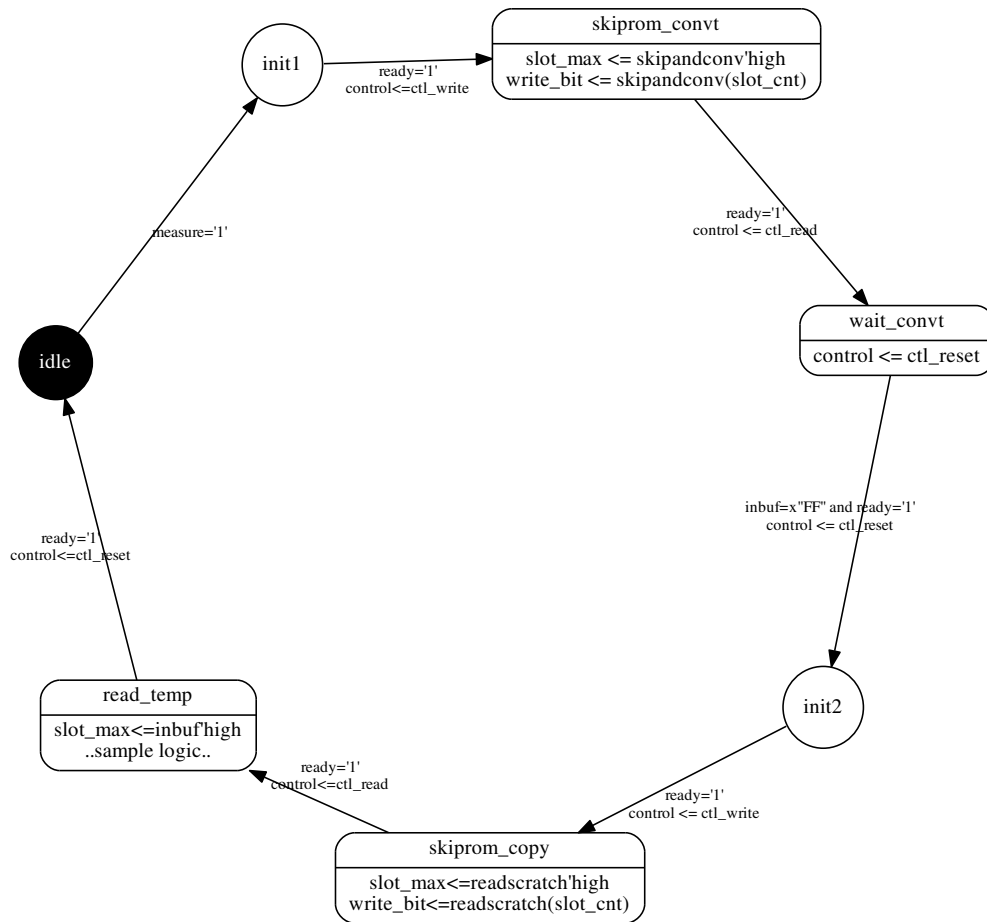
Figur 4: Delblocket DS18S20

VHDL koden är uppbyggd efter tvåprocessmodellen, med en klockad och en kombinatorisk process.

Master	Data	Kommentar
T <sub>x</sub>	Reset	Reset puls.
R <sub>x</sub>	Presence	Sensorn svarar med en presence puls.
T <sub>x</sub>	0x44	Skip ROM command. Eftersom det bara finns en sensor på bussen skippas identifiering.
T <sub>x</sub>	0x44	Convert T command. Sensorn mäter och sparar nuvarande temperatur till sitt interna minne.
R <sub>x</sub>		Sensorn pollas kontinuerligt tills den svarar '1', vilket indikerar att mätningen är klar.
T <sub>x</sub>	Reset	Reset puls.
R <sub>x</sub>	Presence	Sensorn svarar med en presence puls.
T <sub>x</sub>	0x44	Skip ROM command. Eftersom det bara finns en sensor på bussen skippas identifiering.
T <sub>x</sub>	0xBE	Read scratchpad command. Läser sensorns interna minne.
R <sub>x</sub>	<1 byte>	Läser första byten vilket är temperaturen.

Tabell 1: Kontrollsekvens för mätning och avläsning av temperatur





Figur 5: Tillståndsmaskin för delblocket DS18S20

### 3.2.2. Onewire

**Interface** Delblocket onewire sköter lågnivå kommunikationen med temperatursensorn, och exponerar ett högre nivå interface med fyra kontroll-kommandon och en ready signal. För att initiera en operation sätts `control` signalen till ett av följande värden när `ready='1'`:

**ctl\_idle** Gör ingenting. `ready` kommer konstant vara '1'.

**ctl\_read** Läs `slot_max` antal bitar från sensorn. `slot_cnt` är en räknare som indikerar vilken bit som läses just nu. När `sample_now='1'` Förväntas användaren spara biten som under den klockcykeln finns på `read_bit`. När alla bitar är lästa kommer `ready` sättas till '1'.

**Exempel:**

```

if rising_edge(clk) then
    if sample_now = '1' then
        in_buffer(slot_cnt) <= read_bit;
    end if;
end if;

```

**ctl\_write** Skriv `slot_max` antal bitar till sensorn. `slot_cnt` är en räknare som indikerar vilken bit som skrivs just nu. Användaren förväntas lägga biten som ska skrivas till sensorn på `write_bit`. När alla bitar är skrivna kommer `ready` sättas till '1'.

**Exempel:**

```

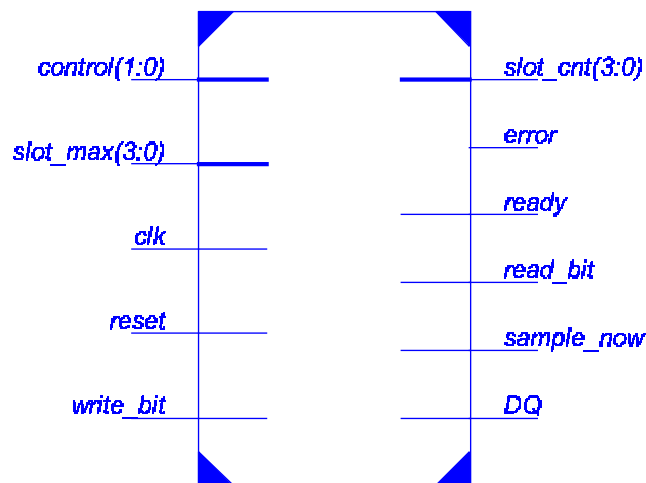
write_bit <= out_buffer(slot_cnt);

```

**ctl\_reset** Återställer sensorns tillståndsmaskin. När reset-sekvensen är klar kommer `ready` sättas till '1'.

Under en pågående operation kommer `ready` vara '0'. Observera att efter "power on" eller "master reset" kommer onewire utföra reset-sekvensen för sensorn, och användaren måste vänta på `ready='1'` innan ett kontrollkommando kan ges. Det finns även en **error** signal som kommer gå hög under en klockcykel om inte temperatursensorn svarar under resetsekvensen.

## *onewire\_proto*



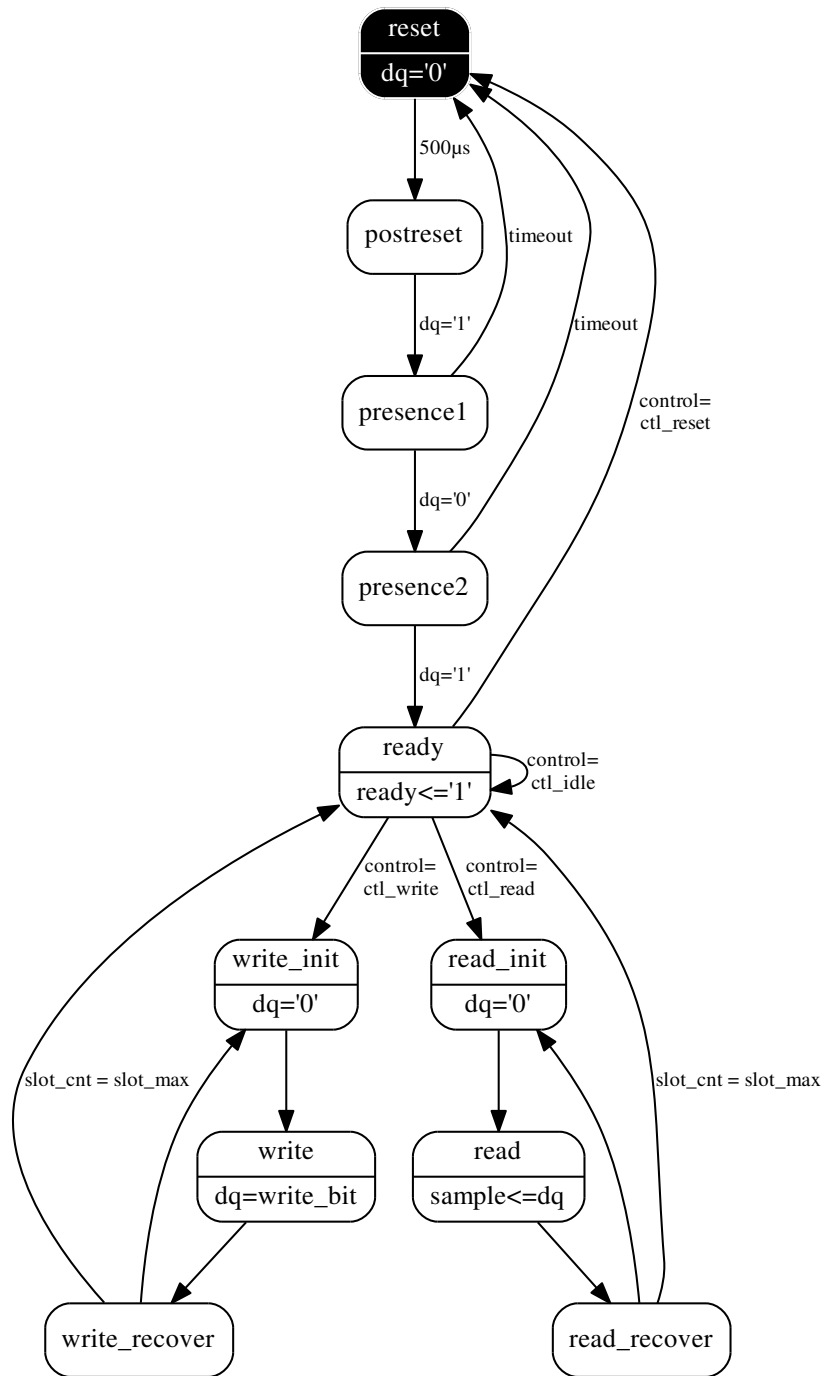
Figur 6: Delblocket onewire

**Implementation** Onewire delblocket implementerar Dallas 1-wire protokoll. För 1-wire används enbart en pin (`DQ`) för kommunikation. Ingen gemensam klocka finns. 1-wire bygger på master-slave principen, med sensorn som slave och kontrollen som master. Till `DQ` är en 5KΩ pullup resistor kopplad. Kommunikation sker via *write slots* och *read*

*slots*. Mastern initierar all kommunikation. 1-wire är *open drain*, vilket innebär att pullup resistorn driver DQ hög när det inte är någon aktivitet på bussen. All data skrivs och avläses med den minst signifikanta biten först (LSB).

1-wire har stöd för sk. "parasite power", där DQ driver temperatursensorn. Onewire delblocket använder sig dock inte utav denna funktion, utan sensorn drivs genom  $V_{dd}$ .

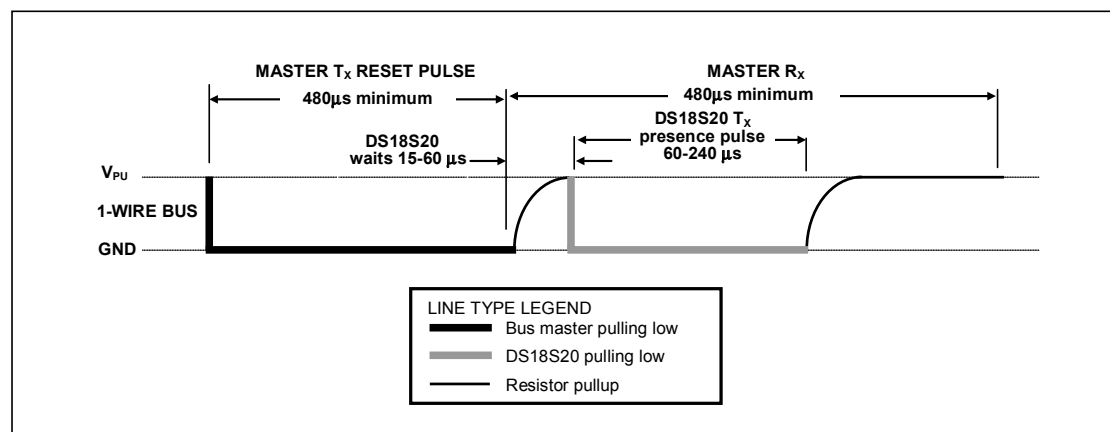
Kontrollen är uppbyggd av en tillståndsmaskin, se Figure 7. VHDL koden är uppbyggd efter tvåprocessmodellen, med en klockad och en kombinatorisk process.



Figur 7: Tillståndsmaskin för delblocket onewire

**Reset** Resetfasen korresponderar till *reset* → *postreset* → *presence1* → *presence2* → *ready* i onewires tillståndsmaskin (Figure 7).

Vid FPGA:ns *power on*, eller när den globala, asynkrona **reset** signalen går från hög till låg kommer tillståndsmaskinen börja i **reset**. Initieringssekvensen för temperatursensorn DS18S20 kommer då inledas. Se Figure 8. Om temperatursensorn svarar med en korrekt *presence pulse* inom accepterade tidsintervall kommer onewire att försättas i tillstånd **ready** och invänta vidare kommandon. Vid felaktigt eller uteblivet svar kommer **error** vara '1' under en klockcykel. Tillståndsmaskinen kommer sedan återgå till **reset** och börja om initieringssekvensen.



Figur 8: Tidsspecifikation för 1-wire resetsekvens

**Skrivning med write slots** Skrivfasen korresponderar till *ready* → *write\_init* → *write* → *write\_recover* i onewires tillståndsmaskin (Figure 7).

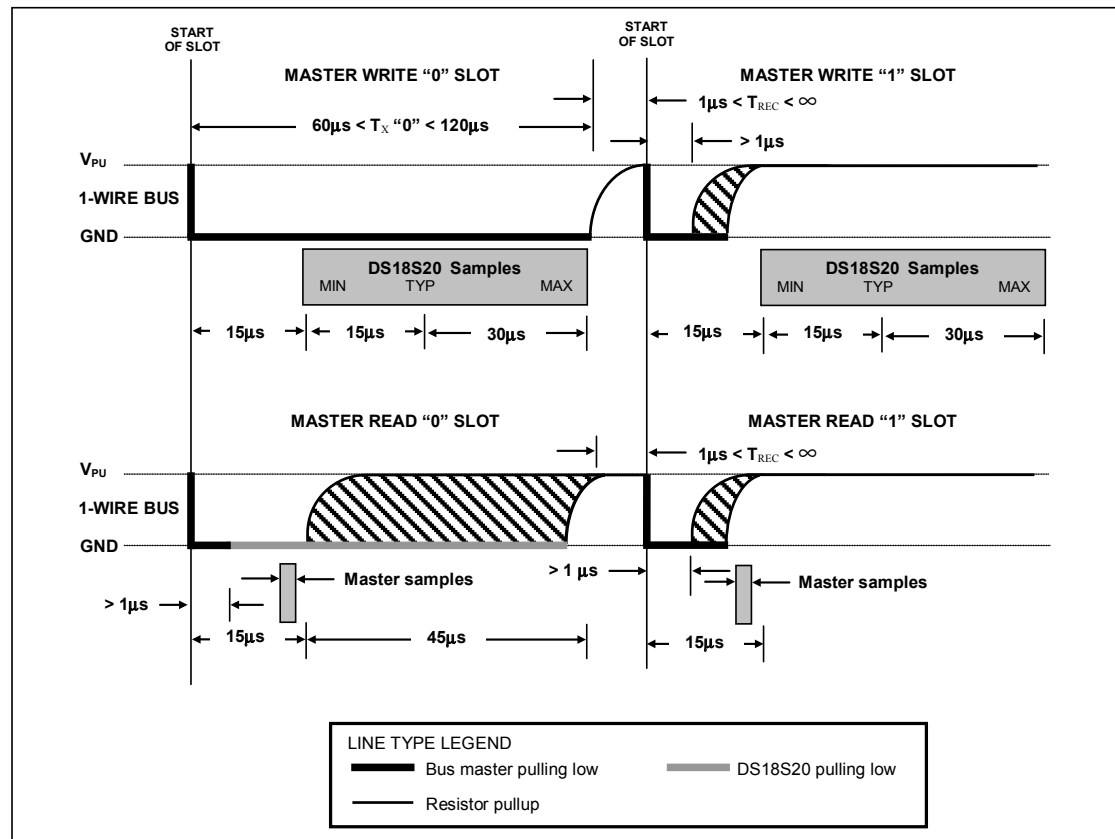
För att påbörja en skrivning sätts **control** till **ctl\_write** när **ready**='1'. **slot\_max** bitar kommer då skrivas till sensorn. 1-wire protokollet använder *write slots* för att skriva bitar till sensorn. Flera bitar skrivs genom flera efterföljande write slots.

En write slot börjar med att buss-mastern driver DQ låg 1–15µs. Sensorn kommer sampla värdet på bussen 15–60µs efter att DQ gick från hög till låg. Efter varje write slot behövs >1µs återhämtningstid.

För att skriva '0' driver kontrollen DQ låg i  $60\mu s < T_x < 120\mu s$ . För att skriva '1' släpper kontrollen DQ maximalt 15µs efter att write slot påbörjades. Se Figure 9.

**Läsning med read slots** Läsfasen korresponderar till *ready* → *read\_init* → *read* → *read\_recover* i onewires tillståndsmaskin (Figure 7).

För att påbörja en läsning sätts **control** till **ctl\_read** när **ready**='1'. **slot\_max** bitar kommer då läsas från sensorn. 1-wire protokollet använder *read slots* för att skriva bitar till sensorn. Flera bitar skrivs genom flera efterföljande read slots. En read slot initieras alltid av mastern genom att driva DQ låg i  $1\mu s < T_x < 15\mu s$ . Sensorn kommer efter att DQ gått från '1' → '0' lägga ut '1' eller '0' på DQ. Data är giltig upp till 15µs efter det att master initierar read slot. Se Figure 9.



Figur 9: Write och read slots tidsspecifikation för '0' respektive '1'.

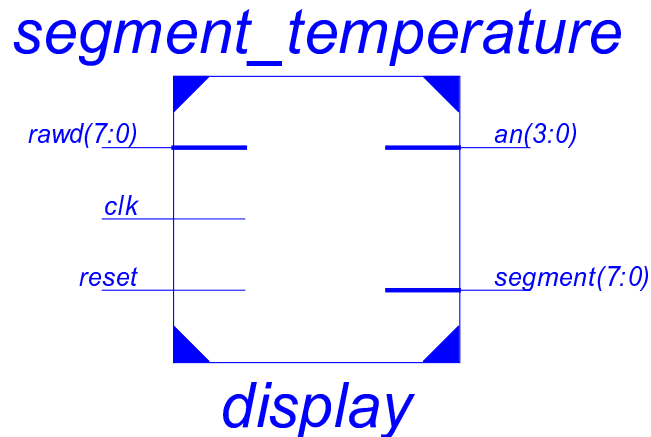
### 3.3. 7-Segmentdisplay

Delblocket ansvarar för att visa ett 8 bitars binärt tal på tvåkomplementsform där bit 7–1 är heltalsdelen och bit 0 decimaldelen på en 7-segmentdisplay med basen 10.

Det finns stöd för att visa tal mellan  $-99 - 999$ , med en decimalsiffra som antingen är .0 eller .5. Det är tillräckligt för temperatursensorn som är specifierad mellan  $-75 - -125 \pm 5$ .

#### 3.3.1. segment-temperature

**Interface** Komponenten tar ett binärt tal som input och ger utsignalerna till 7-segmentdisplayerna.



Figur 10: Delblocket segment-temperature

**Implementation** De fyra fysiska 7-segmentdisplayerna har en gemensam databuss. Varje display har även en enable signal. Komponenten växlar mellan att visa de olika displayerna med 1kHz, vilket för ögat upplevs som att alla lyser konstant.

**bcd** Funktion som gör om ett binärt tal utan tecken till bcd-form. “Double Dabble” algoritmen används internt i form utav ett kombinatoriskt nät.

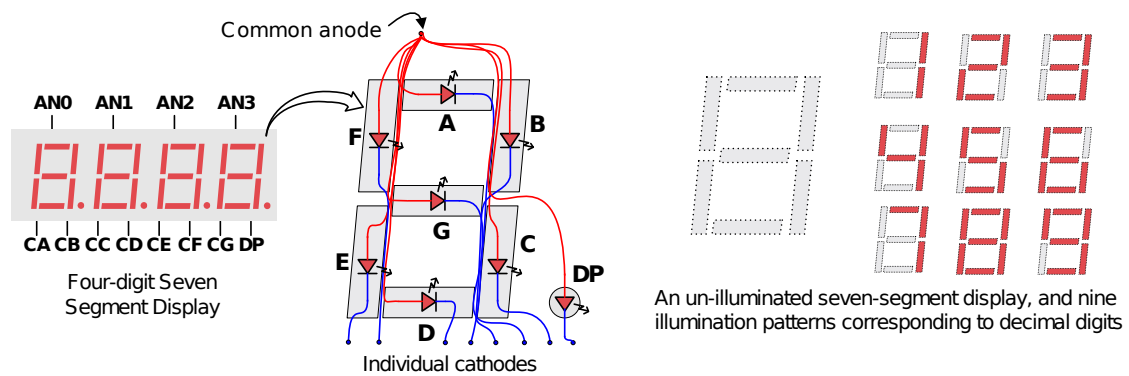
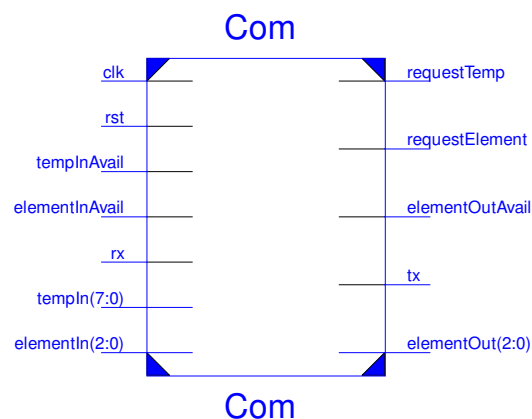


Figure 11: 7-segments display



### 3.4. Kommunikation

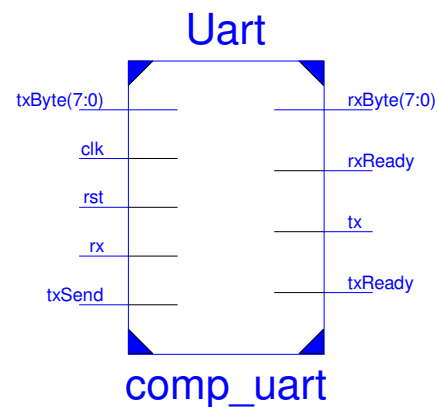
Kommunikationsmodulen tolkar styrmeddelanden från en yttre GSM-enhet och vidarebefodrar dessa till styrenheten. Därefter kodar den data från sensorer så som temperaturmodulen och skickar tillbaka den till GSM-enhet. Kommunikation styrs med hjälp av AT- kommandon som innehåller ett fåtal kodord (se appendix A) som GSM-modulen skickar/mottar via SMS från operatören. Com-moulens består av delblocken Uart samt AT. Uart sköter seriel kommunikationen med GSM-modulen. AT-blocket omtolkar AT kommandon och kodord till styrsignaler och vice versa. Tolkningen av kodorden består i att omvandla dem till relevanta styrsignaler och extrahera data. Vidare kodar den styrsignaler och data till AT meddelanden som ska skickas till GSM-modulen.



Figur 12: kommunikationsmodul

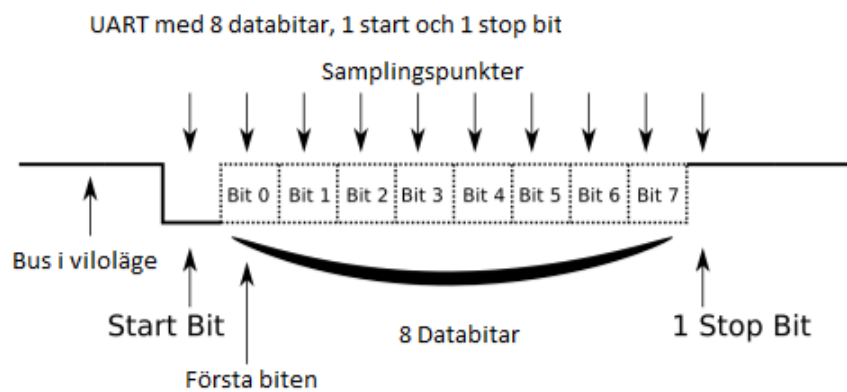
Com enheten har två portar för seriell kommunikation tx och rx. Dessa är direkt kopplade till motsvarande på Uart blocket. Den har tre databussar, elementIn, elementOut och tempIn. De två första bussarna är kopplade till elementmodulen och används för att indikera vilka element som ska vara på respektive vilka som är på. Temperaturbussen håller temperaturen från temperaturmodulen på binär form (signed). Tre styrsignaler requestTemp, requestElement och elementAvail går till styrenheten. De har följande betydelse (i samma ordning). Ett kodord har avkodats som ber om temperaturdata, elementdata samt att elementData finns på elementOut- bussen.

**Uart** Uarten är uppdelad i de fristående blocken txControl och rxControl som skickar respektive tar emot seriel data på var sin port och jobbar därmed i full duplex.



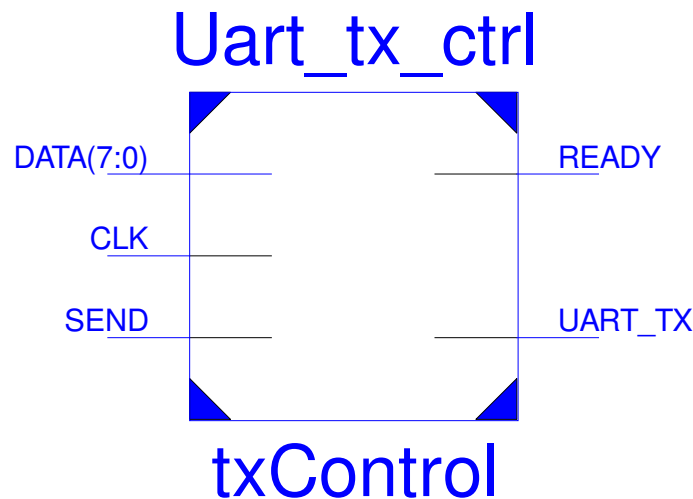
Figur 13: Uart modul för seriekommunikation

Kommunikationen sker enligt följande specifikation. Datan skickas med 10 bitar i taget där den första biten är en sk startbit (låg) och den sista är stopbiten (hög). Därimellan skickas åtta databitar. Ingen paritetskontroll används. Bitarna skickas med 9600bps vilket innebär att det tar ca 1042us mellan startbitens positiva flank och stopbitens negativa flank.



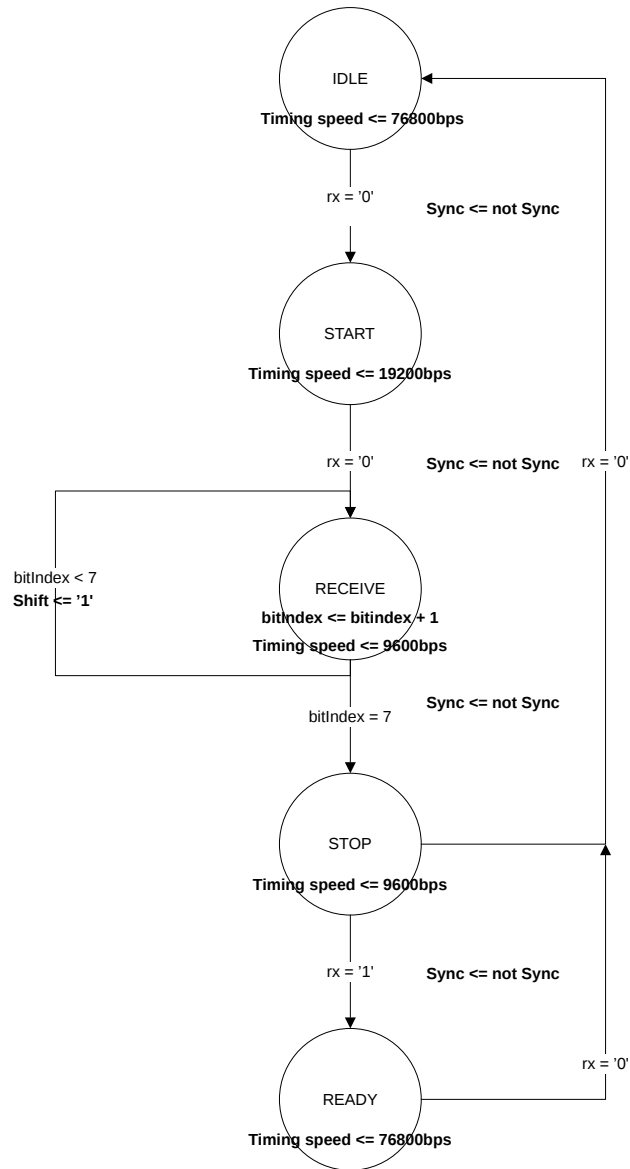
Figur 14: Insignalen synkas på startbiten för att därefter kunna avläsas i mitten av varje databit.

**uart\_tx\_ctrl** Komponenten serialiserar en byte i taget och skickar ut datan med 9600bps enligt Uart specifikationen. Den åstakommer detta genom att lägga till start och stopbit till DATA-byten. Sedan stegar den helt enkelt igenom data vektorn bit för bit. I takt med att en klockräknare når 10416 cykler så skickas en bit ut. Med en 100mhz klocka ger detta 9600bps. För att skicka en byte så lägger man ut denna på DATA porten och väntar tills READY är hög. Då driver man SEND hög under en klockpuls för att skicka 10 bitar seriellt. READY signalen är låg under detta förlopp.



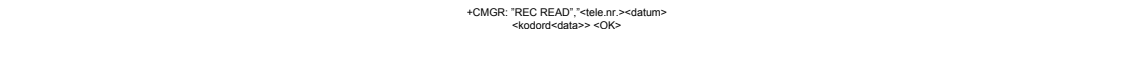
Figur 15: Uart modul för seriekommunikation

**uart\_rx\_ctrl** Eftersom det inte går att veta när en ny databit lagts ut på signallinan så måste man synka mottagningen. Detta gör man genom att använda sig av ett översnkomet startläge. I detta fall låg signal. Eftersom vi vet antalet bitar därefter så kan vi med rätt timing sampla signalen vid varje bit i meddelandet. Blocket består av en cirkulär tillståndsmaskin samt ett initieringstillstånd, IDLE, som bara ingås vid väntan på första startbiten efter reset. I detta tillstånd väntar man på att datasignalen ska gå låg vilket indikerar startbiten och tillståndet övergår då till START. I start-tillståndet så startas en timer som ger en puls efter halva startbiten. Är den fortfarande låg så har signalen synkats framgångsrikt. Därefter ingås RECEIVE tillståndet där signalen samplas. Timern startas om med 9600bps cykellängd. Eftersom FPG:an jobbar i 100mhz motsvarar detta  $((100\text{MHz} / 9600) - 1) = 10416$  klockcykler. Signalen samplas åtta gånger enligt specifikationen och bitarna shifas in i ett shiftregister. Därefter ingås STOP tillståndet. Om den sista biten samplas hög så anses en byte data framgångsrikt mottagen och tillståndet ändras till READY. I detta tillstånd inväntas en ny startbit och utsignalen byteReady drivs hög för att indikera att en byte data ligger på byteOut porten.



Figur 16: State ändras endast då timing=1. Timing hastigheten ändras beroende på tillstånd. Sync är en toggle-signal som resttar timing räknaren. I ready tillståndet så är byteOut giltig.

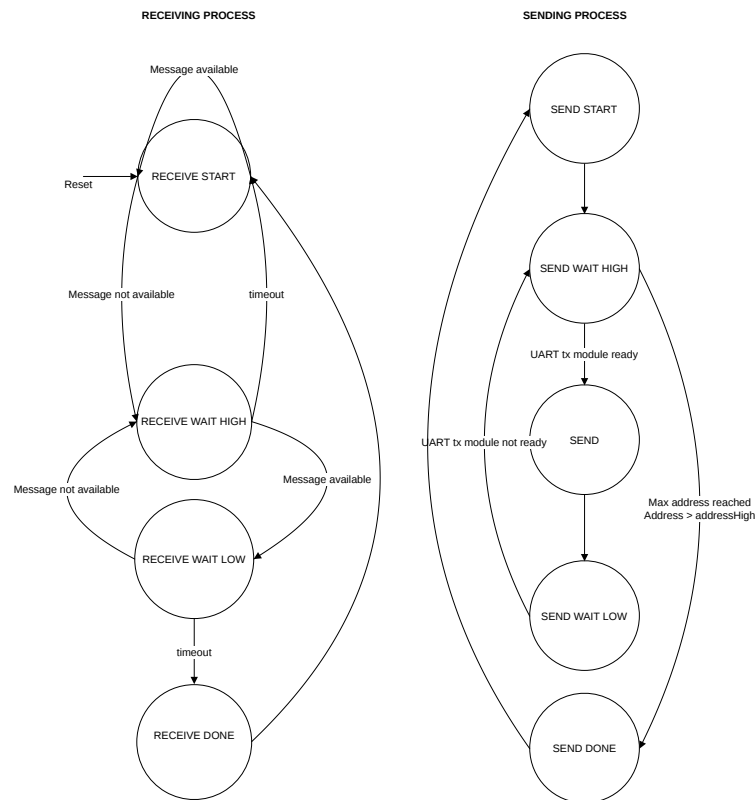
**AT** AT enheten jobbar i en cykel som inbegriper flera omgångar av både ingående och utgående meddelanden till GSM-enheten. En korrekt sekvens ska se ut enligt flödesschemat i figur 16.



```
+CMGR: "REC READ", "<tele.nr><datum>
<kodord<data>> <OK>
```

Figur 17: Till vänster visas tillstånden enheten går igenom, till höger de AT meddelanden som skickas fram och tillbaka mellan AT-enheten och GSM-modulen.

AT modulen innehåller en minneskomponent där alla inkommande meddelanden sparas för att senare avkodas till styrsignaler. Utgående meddelanden skrivs till samma minne för att sedan skickas byte för byte till UARTen för serialisering. AT enheten drivs av en tillståndsmaskin som i sin tur aktiverar delprocesser som i tur och ordning sköter inläsning, avkodning, kodning samt skickande av data. Dessa processer har egna interna tillstånd vilka visas i figur 17.



Figur 18: Tillståndsmaskiner för processerna send och receive i AT-modulen. Receive väntar på inkommande bytes från Uarten och skriver dem till minnet. Send skickar innehållet i minnet till Uarten.

### 3.5. Element

Detta block hanterar allt som har med elementen att göra.

Blocket sparar även sitt tillstånd, vilket innebär att det själv är medvetet om vilka element som är av- och påslagna.

Först kontrolleras insignalerna för att avgöra om ett eller flera element ska regleras och/eller om elementens aktuella status ska skrivas till databussen.

### 3 BESKRIVNING AV DELBLOCKEN

---

Om elementen ska regleras hämtas ny status från databussen och sedan slås elementen av/på beroende på vad som står i datan som hämtats från databussen.

Önskas aktuell elementstatus, eller ny elementstatus i det fall att elementen har reglerats, skrivs den till databussen.

Efter att elementen har reglerats och/eller aktuell elementstatus skrivits till databussen sätts ut signaler för att tala om exakt vad som gjorts.

## 4. Hårdvara

Systemet är uppbyggt utav huvudsakligen tre komponenter: ett Digilent Nexys3 utvecklingskort, en temperatursensor från Maxim och en mobiltelefon med serieinterface (*RS232*).

### 4.1. Digilent Nexys3 - FPGA

Nexys3 är ett utvecklingskort från Digilent som bygger på en Spartan-6 FPGA från Xilinx. **Nexys3 har bland annat:**

- Xilinx Spartan-6 XC6SLX16 CSG324C.
- Klockfrekvens på 100MHz.
- 48MB externt minne, varav 32MB är ickeflyktigt.
- Mikro USB-port för programmering av FPGA och strömförsörjning.
- USB-UART, genom en mikro USB-port kopplad vid en FTDI FT232 krets.
- USB Host-kontroller för anslutning utav externa USB-enheter, ex. mus, tangentbort, osv.
- 10-100 Mbit ethernet-anslutning.
- 4st 7-segmentdisplayer.
- 8st binära DIP switchar.
- 8st ytmonterade lysdioder
- 4st kontaktstycken för externa I/O-enheter (dubbelbreda Pmod anslutningar).

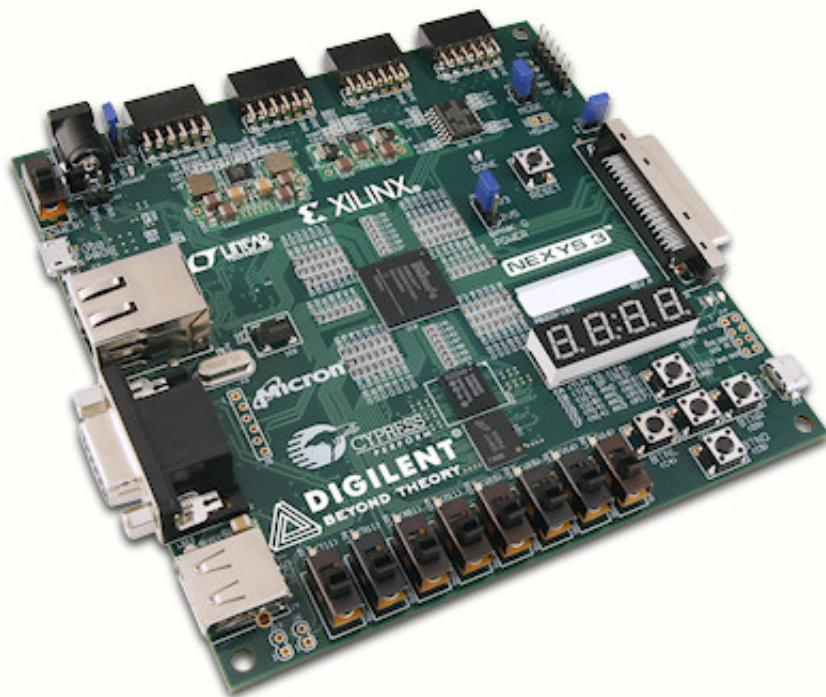
### 4.2. DS18S20 - Temperatursensor

DS18S20 är en temperatursensor tillverkad av Dallas Semiconductor (numera Maxim) som enbart använder sig utav 1 pin för kommunikation.

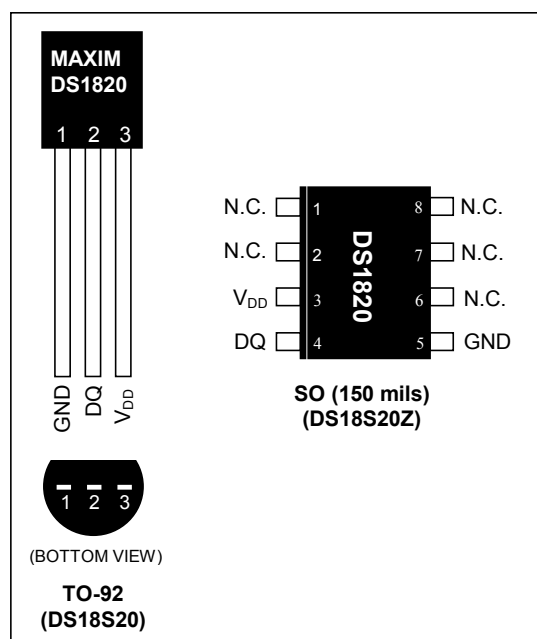
**Sensorn har:**

- Temperaturmätning från -75°C till 125°C med  $\pm 5$  precision.
- Alarmfunktion med icka-flyktigt minne.
- Max 750ms för temperaturmätning
- Flera sensorer kan dela på en buss.
- Ett unikt för varje enhet 64 bitars serienummer.
- Endast två pinnar behövs om "parasite power" används. Då laddar sensorn upp en kondensator när DQ drivs aktivt hög.





Figur 19: Digilent Nexys 3

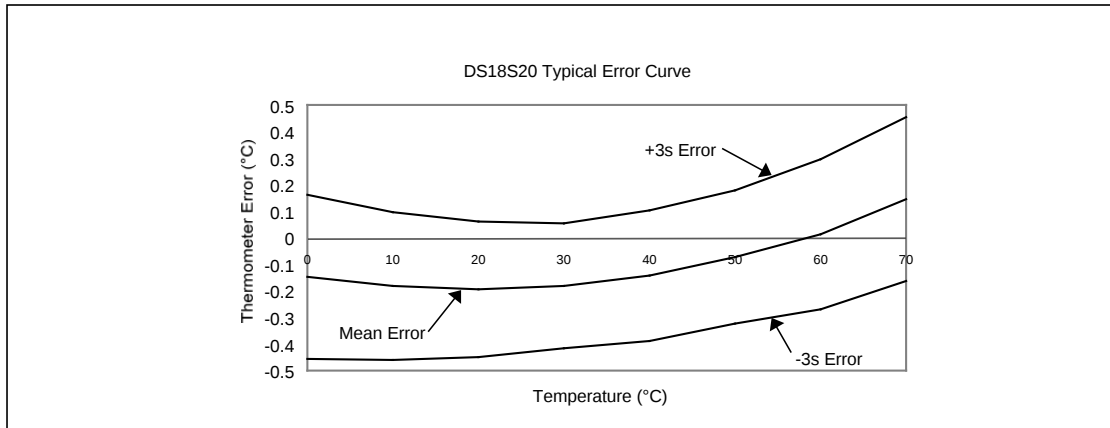


Figur 20: Dallas DS18S20

## 5. Felanalys

VHDL-modulen för temperatursensorn har testats noggrant genom att genomföra flera mätningar på olika platser. Kontroller att temperaturen är koncis med andra temperatursensorer har även genomförts. Se Figure 21 för felmarginalsspecifikationer.

Temperaturmätningen har testats i inomhusmiljö ( $\approx 23^\circ\text{C}$ ) samt utomhusmiljö ( $\approx -10^\circ\text{C}$ ) med goda resultat. Kalibrering av tidsintervall har simulerats samt testats empiriskt för att hitta optimala värden. Den slutgiltiga revisionen är testad upp till ca. 1000 korrekta mätningar i rad, där den kan anses vara helt fungerande.



Figur 21: Visar hur DS18S20s mätfel varierar beroende på temperaturförändringar.

## A. Kommandon

Kodord	Funktion
get element	Begär antal element som är igång.
get temp	Begär nuvarande temperatur
set element:<element[int]	Sätter antal element som ska vara igång
temp:<temp[int]>	Anger nuvarande temperatur
element:<element[int]>	Anger antal element igång

Avsändare	AT-kommando	Funktion
GSM-enhet	+CMTI=<mem. location>,<index>	Anger index för nytt meddelande.
AT-modul	AT+CMGF=1	Ställer in 'text mode' på GSM modulen.
AT-modul	AT+CMGR=<index>	Begär meddelandet med angivet index.
GSM-enhet	+CMGR:"REC READ",<telefon nr.>"<datum>" <kodord> <OK>	Meddelandet med data och kodord från användaren.
AT-modul	AT+CMGS="<telefon nummer>"<data>	Meddelande innehållandes svarsdata till användaren.

## B. Signallista

### B.1. DS18S20

Namn	Typ	Kommentar
clk	in std_ulogic	Global klocka, 100MHz
reset	in std_ulogic	Global asynkron reset
measure	in std_ulogic	Påbörja temperaturavläsning
valid	buffer std_ulogic	Temperaturavläsning klar. Giltig så länge valid = '1'
DQ	inout std_logic	Pinne till sensor
temperature	buffer signed(7 downto 0)	Temperatur i tvåkomplements binärform

**B.2. segment-temperature**

<b>Namn</b>	<b>Typ</b>	<b>Kommentar</b>
clk	in std_ulogic	Global klocka, 100MHz
reset	in std_ulogic	Global asynkron reset
rawd	in signed(7 downto 0)	Temperatur i tvåkomplements binärform
valid	out std_ulogic	Temperaturavläsning klar. Giltig så länge <code>valid = '1'</code>
an	buffer std_ulogic_vector(3 downto 0)	7 Segment-enable
segment	buffer std_ulogic_vector(7 downto 0)	Utsignal till 7-segmentsbussen

**B.3. Com**

<b>Namn</b>	<b>Typ</b>	<b>Kommentar</b>
clk	in std_ulogic	Global klocka, 100MHz
rst	in std_ulogic	Global asynkron reset
tempInAvail	in std_logic	Temperatur finns tillgänglig på bus
elementInAva	in std_logic	Elementdata finns tillgänglig på bus
requestTemp	out std_logic	Ber styrenheten om tempdata till bus
requestEleme	out std_logic	Ber styrenheten om elementdata till bus
elementOutA	out std_logic	Meddelar styrenheten, elementdata på bus
tempIn	in std_logic_vector(7 downTo 0)	Databus från temperaturmodul
elementIn	in std_logic_vector(1 downTo 0)	Databus från elementmodul
elementOut	out std_logic_vector(7 downTo 0)	Databus till elementmodul
tx	out std_logic	Seriell port ut till GSM enhet ut från AT-modul
rx	in std_logic	Seriell port in till AT-modul från GSM-modul

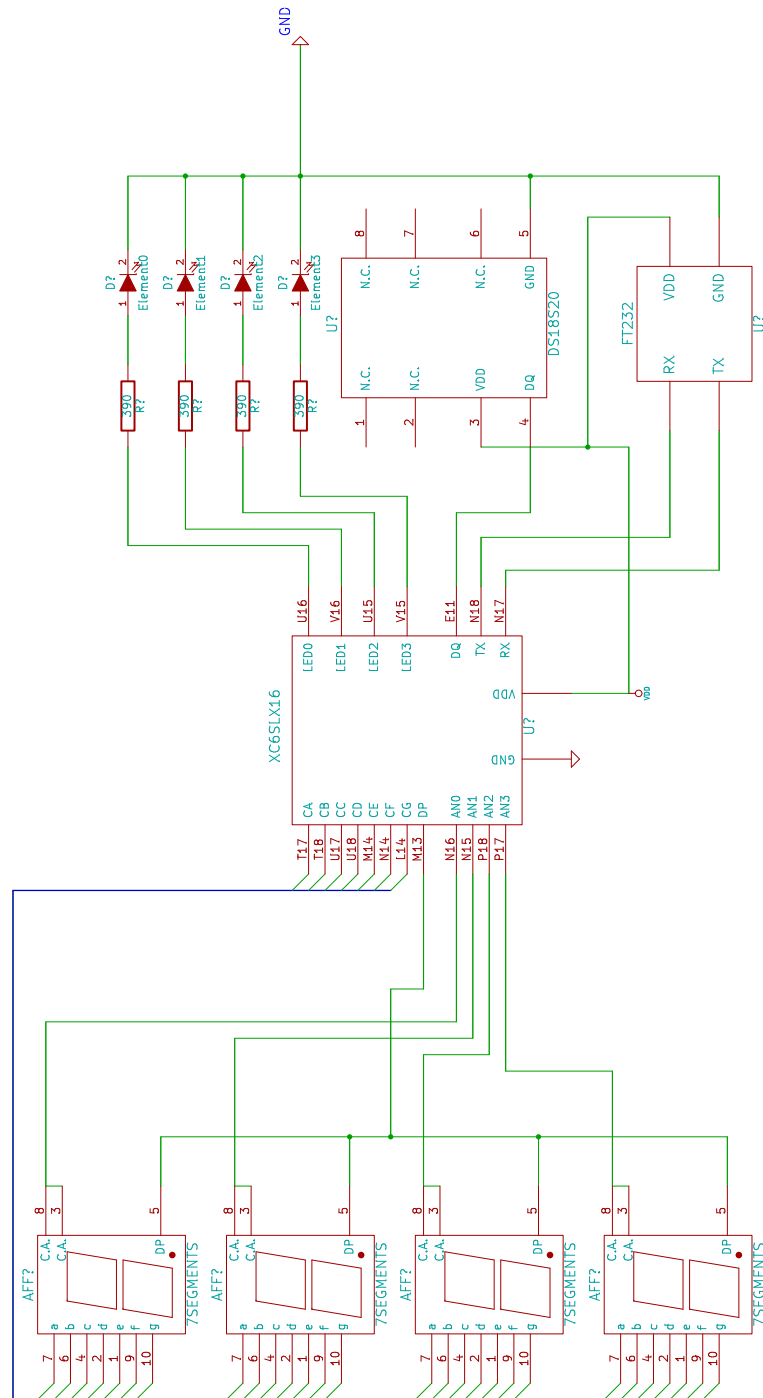
**B.4. Styrenhet**

<b>Namn</b>	<b>Typ</b>	<b>Kommentar</b>
clk	in std_logic	Global klocka, 100MHz
reset	in std_logic	Global asynkron reset
tempPutTempOnDb	out std_logic	Tala om för Temperatur-blocket att lägga ut temperaturen på databussen
tempNowOnDb	in std_logic	Temperaturen finns tillgänglig på databussen
comHasTempOnDb	out std_logic	Talar om för Kommunikations-blocket att temperaturen finns på databussen
comHasElemStatusOnDb	out std_logic	Talar om för Kommunikations-blocket att aktuell elementstatus finns på databussen
comHasElemStatus	in std_logic	Meddelar styrenheten att Element-blocket har ny elementstatus på bussen
comWantElemStatus	in std_logic	Talar om för styrenheten att aktuell elementstatus önskas på databussen
comWantTemp	in std_logic	Talar om för styrenheten att temperaturen önskas på databussen
elemHasStatusOnDb	out std_logic	Talar om att ny elementstatus finns på databussen
elemPutStatusOnDb	out std_logic	Talar om att aktuell elementstatus ska skrivas till databussen
elemNewStatusDone	in std_logic	Talar om att elementstatus har uppdaterats till det som fanns på databussen
elemNewStatusOnDb	in std_logic	Talar om att aktuell elementstatus finns tillgänglig på databussen

**B.5. Element**

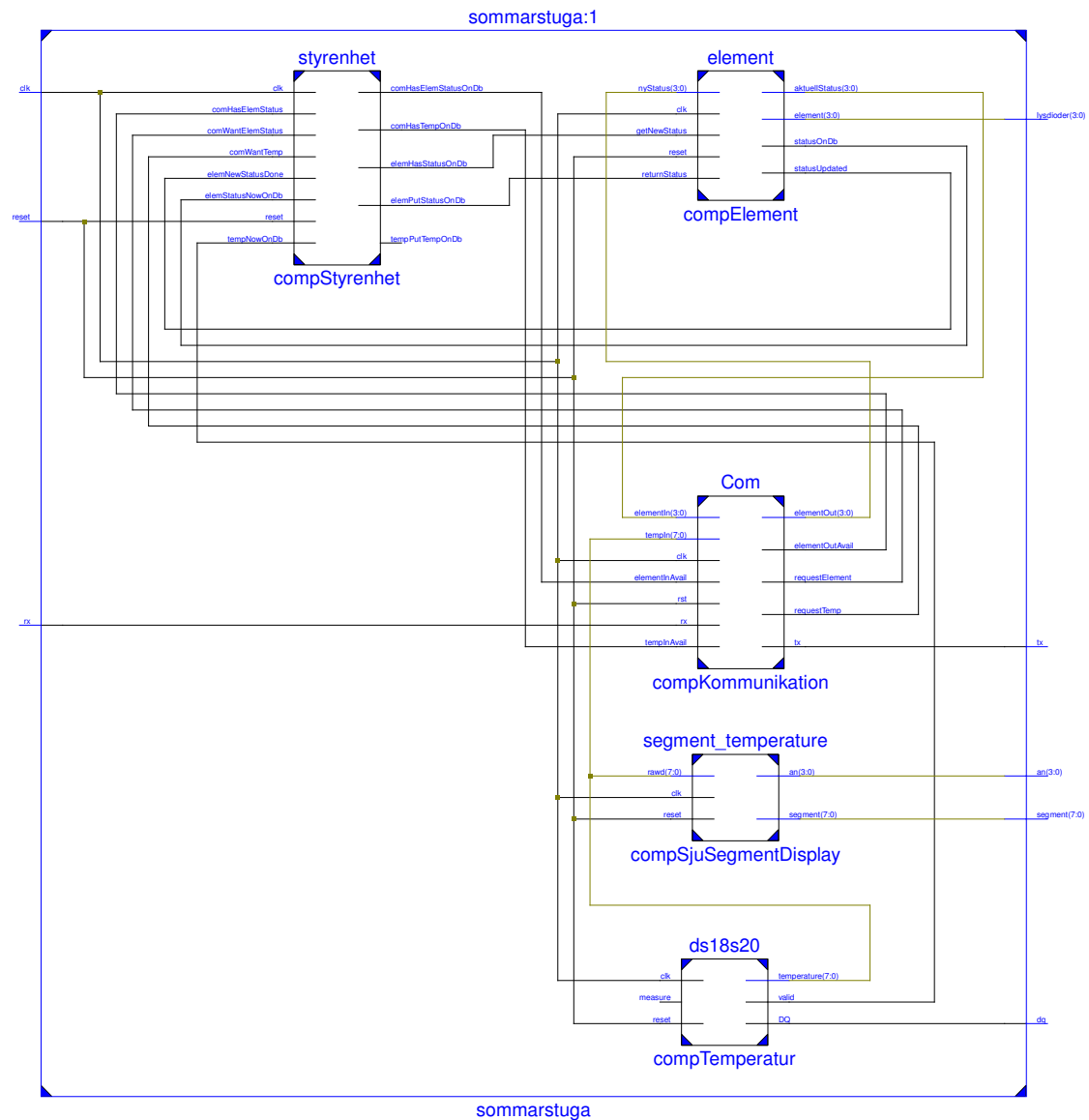
Namn	Typ	Kommentar
clk	in std_logic	Global klocka, 100MHz
reset	in std_logic	Global asynkron reset
setNewStatus	in std_logic	Reglera elementen enligt ny elementstatus från databussen
returnStatus	in std_logic	Skriv aktuell elementstatus till databussen
statusOnDb	out std_logic	Talar om att aktuell elementstatus skrivits till databussen
updatedStatus	out std_logic	Talar om att elementstatus nu ändrat till det som fanns på databussen
input	in std_logic_vector(3:0)	Databuss för inkommand elementstatus
output	out std_logic_vector(3:0)	Databuss för utgående aktuell elementstatus

## C. Kretsschema



Figur 22: Förenklat kretsschema, över de komponenter som används, baserat på Digilents kretsschema för Nexys3.

## D. Syntesschema



Figur 23: Genererat syntesschema över hela prototypen



## E. Komponentlista

Namn	Beteckning
Utvecklingskort	Digilent Nexys3
Temperatursensor	Maxim DS18S20
Motstånd	4.7 k $\Omega$
GSM-Modul	Emuleras med serieterminal

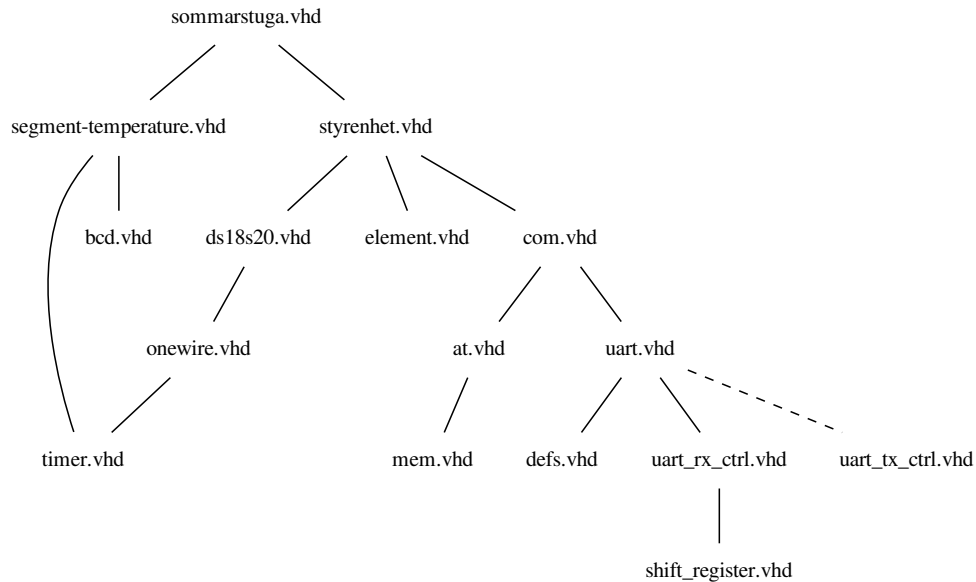
## F. Pinlayout

Pin	Beskrivning	Från	Till
T17	Katod för segment A	FPGA	7-segmentdisplayen
T18	Katod för segment B	FPGA	7-segmentdisplayen
U17	Katod för segment C	FPGA	7-segmentdisplayen
U18	Katod för segment D	FPGA	7-segmentdisplayen
M14	Katod för segment E	FPGA	7-segmentdisplayen
N14	Katod för segment F	FPGA	7-segmentdisplayen
L14	Katod för segment G	FPGA	7-segmentdisplayen
M13	Katod för segment DP	FPGA	7-segmentdisplayen
N16	Gemensam anod för 7-seg AN0	FPGA	7-segmentdisplayen
N15	Gemensam anod för 7-seg AN1	FPGA	7-segmentdisplayen
P18	Gemensam anod för 7-seg AN2	FPGA	7-segmentdisplayen
P17	Gemensam anod för 7-seg AN3	FPGA	7-segmentdisplayen
U16	Aktivera lysdiod LD0	FPGA	LED0
V16	Aktivera lysdiod LD1	FPGA	LED1
U15	Aktivera lysdiod LD2	FPGA	LED2
V15	Aktivera lysdiod LD3	FPGA	LED3
N18	Tx	FPGA	USB-UART FTDI FT232
N17	Rx	FPGA	USB-UART FTDI FT232

## G. Programkod (VHDL)

All programkoden inklusive testfiler finns i ett git-källkodsrepository:

<https://github.com/Cadynum/sommarstuga.git>



Figur 24: Övergripande bild hur VHDL-moduler relaterar till varandra. Observera att `uart_tx_ctrl.vhd` ej är egenproducerad.

## H. Arbetsredogörelse

### H.1. Christian Isaksson

- Jag har gjort blockschemat, tagit fram hur databussarna ska se ut (bredden på dem), hur många som behövs och var de ska vara någonstans.
- Skrivit koden för:
  - Styrenheten.
  - Hantering utav elementen.
  - Översta lagret av delblocket Kommunikation.
- Jag lödde fast den ytmonterade temperatursensorn, som sen visade sig vara fel krets.
- Jag har satt ihop själva grundstrukturen för denna rapport och skrivit flertalet av kapitlen.
- Varit med och löst problemet med UART-mottagaren, beträffande när inkommande data ska börja att samplas för att kunna avgöra huruvida en bit är 1 eller 0 och hur länge datan ska samplas för att göra avgörandet.
- Tagit fram strukturen för hur AT-kommandona ska hanteras. Hur gsm-modulen förmedlar att ett nytt sms inkommit och hur innehållet i sms:et sedan hämtas ut ur gsm-modulens minne.
- Jag har skapat pinlayouten samt ritat kretsschemat, med hjälp av ett CAD program. Både pinlayouten och kretsschemat finns i appendix. Har även skapat signallistan för styrenheten och elementen.

### H.2. Christoffer Öjeling

- Alla Komponenter för mätning och avläsning av temperatur från DS18S20 inklusive hjälpfunktioner.
- Lött den korrekta DS18S20-konstruktionen på ett mönsterkort.
- Visning av ett binärt tal på FPGA:ns 7-segmentdisplay.
- Testning och verifikation av DS18S20 kontrollen.
- Utskriften av temperatur via UART ("Temperatur: 31C").
- Felanalys-delen i rapporten
- Allt i rapporten som relaterar till temperatursensorn DS18S20 och 7-segmentdisplayen.
- Följande VHDL-moduler:

- onewire.vhd
- ds18s20.vhd
- segment-temperature.vhd
- timer.vhd
- bcd.vhd

### H.3. Jan Pettersson

- Kommunikationsmodul implementering
- UART modul
- AT modul
- Avkodning / kodning av AT-kommando
- Allt i rapporten som relaterar till kommunikationsmodulen.
- Följande kodfiler
  - Defs.vhd
  - Uart.vhd
  - At.vhd
  - Mem.vhd
  - Shift\_register.vhd
  - Uart\_rx\_ctrl.vhd