Sommarstugekoll

EDA234 - Digital ponstruktion, 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

1 Systemspecifikation **1**

2 Systembeskrivning **2**

2.1 Blockschema . . . 2

3 Beskrivning av delblocken **3**

3.1 Styrenhet . . . 3

3.2 Temperatur . . . 4

3.2.1 DS18S20 . . . 4

3.2.2 Onewire . . . 5

3.3 7 Segments display . . . 11

3.3.1 segment_temperature . . . 11

3.4 Kommunikation . . . 11

3.5 Element . . . 11

4 Hårdvara **13**

4.1 Digilent Nexys3 - FPGA . . . 13

4.2 DS18S20 - Temperatursensor . . . 13

5 Resultat och diskussion **15**

5.1 Resultat . . . 15

5.2 Felanalys . . . 15

5.3 Diskussion . . . 15

A Signallista **i**

A.1 DS18S20 . . . i

A.2 segment_temperature . . . i

A.3 Com . . . ii

B Kretsschema **iii**

C Syntesschema **iv**

D Komponentlista **v**

E Pinlayout **vi**

F Programkod (VHDL) **vii**

G Arbetsredogörelse **xiv**

G.1 Christian . . . xiv

G.2 Christoffer Öjeling . . . xiv

G.3 Jan . . . xiv

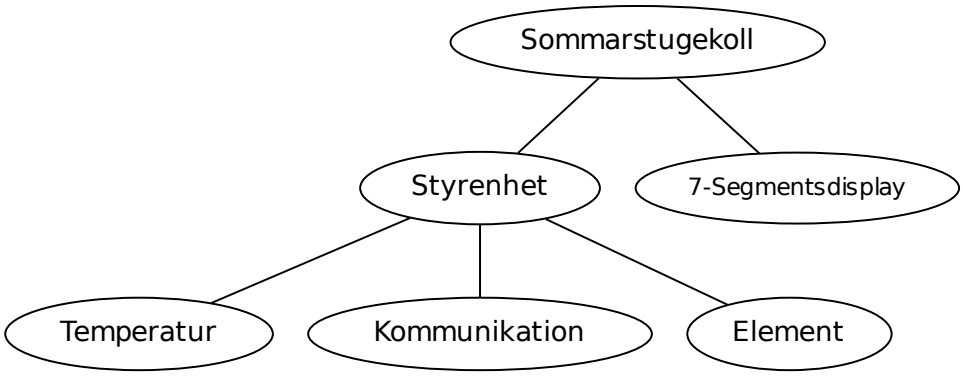
1. Systemspecifikation

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*).

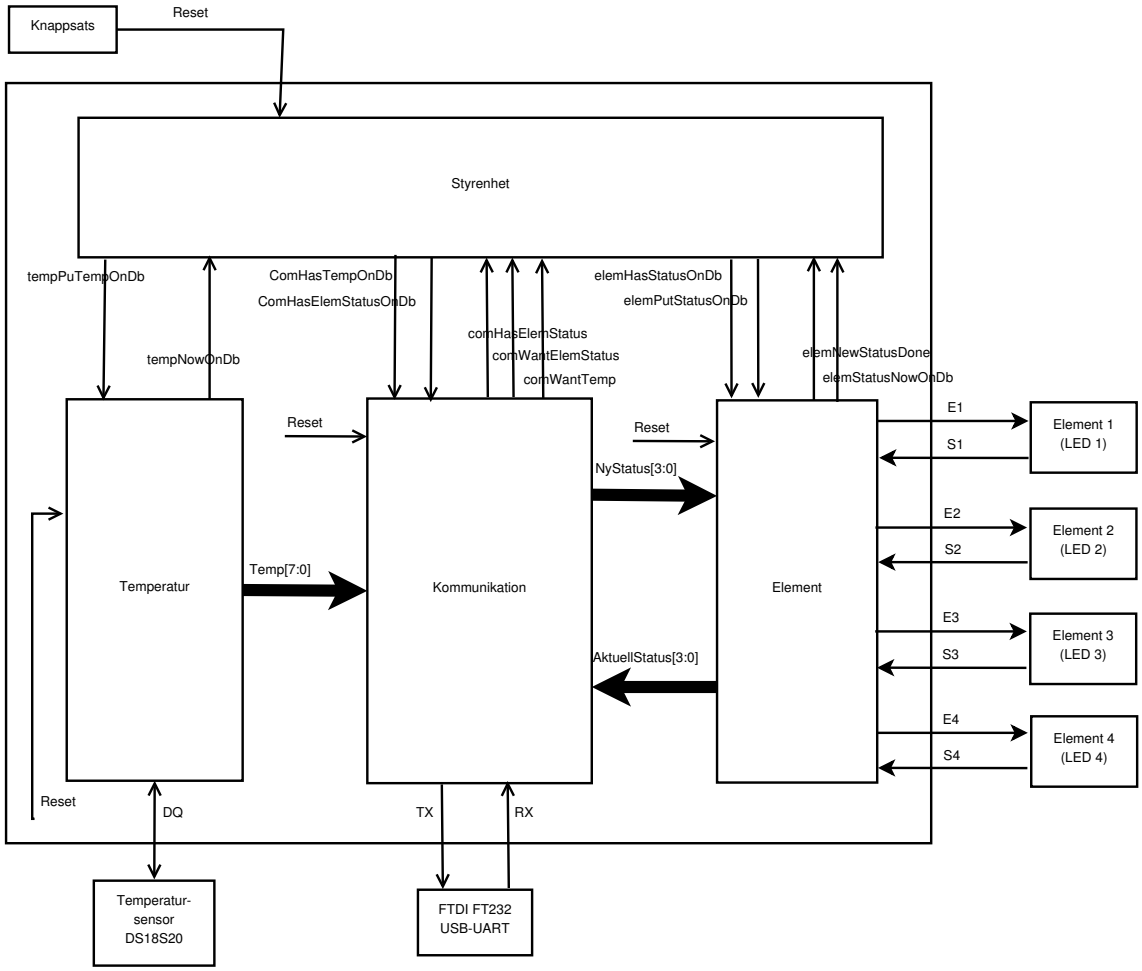
Huvudenheten är uppbyggd utav en *Field-Programmable Gate Array* (FPGA) innehållande all logik.



Figur 1: Övergripande struktur för projektet

2. Systembeskrivning

2.1. Blockschema



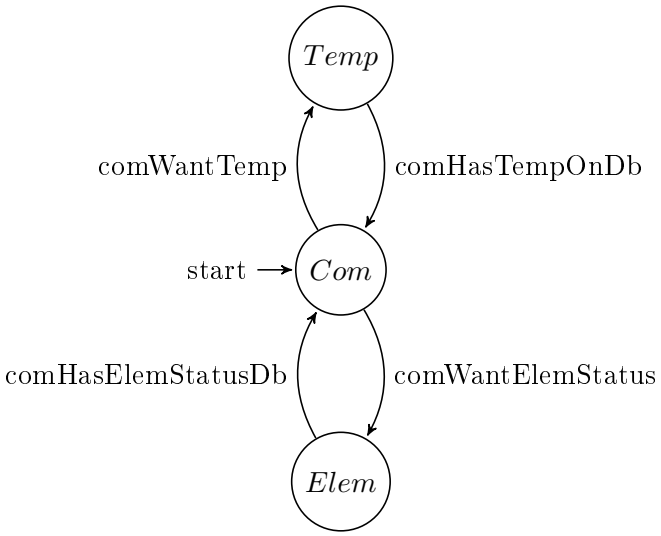
Figur 2: Övergripande blockschema för huvudenheten.

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.

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.
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 och sedan meddelas styrenheten om att ny status finns att läsa på databussen. 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 tillståndsövergång till **Elem** sker.

I **Temp** väntar styenheten på kvittens från *Temperatur* om att temperaturen skrivits till databussen och går att läsa.
När kvittens fåtts återgår styrenheten till tillståndet **Com**, samtidigt som en styrsignal meddelar delblocket *Kommunikation* att temperaturen nu finns att hämta på databussen.

I **Elem** väntar styrenheten på kvittens från delblocket *Element* om att elementen har reglerats och/eller att aktuell elementstatus finns på databussen.
När kvittens fåtts återgår styrenheten till tillståndet **Com**, samtidigt som styrsignaler meddelar delblocket *Kommunikation* att elementen reglerats och/eller aktuell elementstatus finns att hämta på databussen.

3.2. Temperatur

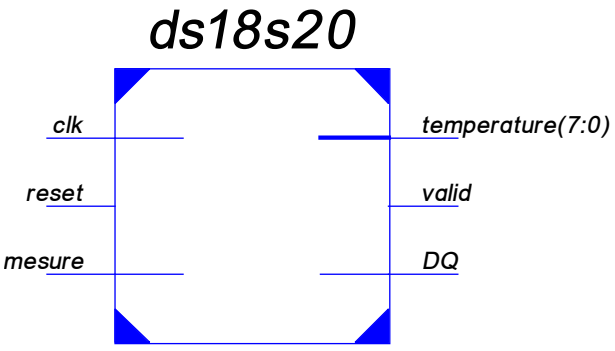
Logiken för avläsning av temperatur är uppdelad i ett antal delblock. Dels för att vara lättöverskådligt, men även så att man lätt ska kunna lägga till funktionalitet i efterhand. Slutanvändaren använder endast DS18S20 direkt. Onewire modulen är inte bunden till just DS12S20, utan kan även användas till andra 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 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.



Figur 4: Delblocket DS18S20

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.

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

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

Tabell 1: Kontrollsekvens för mätning och avläsning av temperatur

3.2.2. Onewire

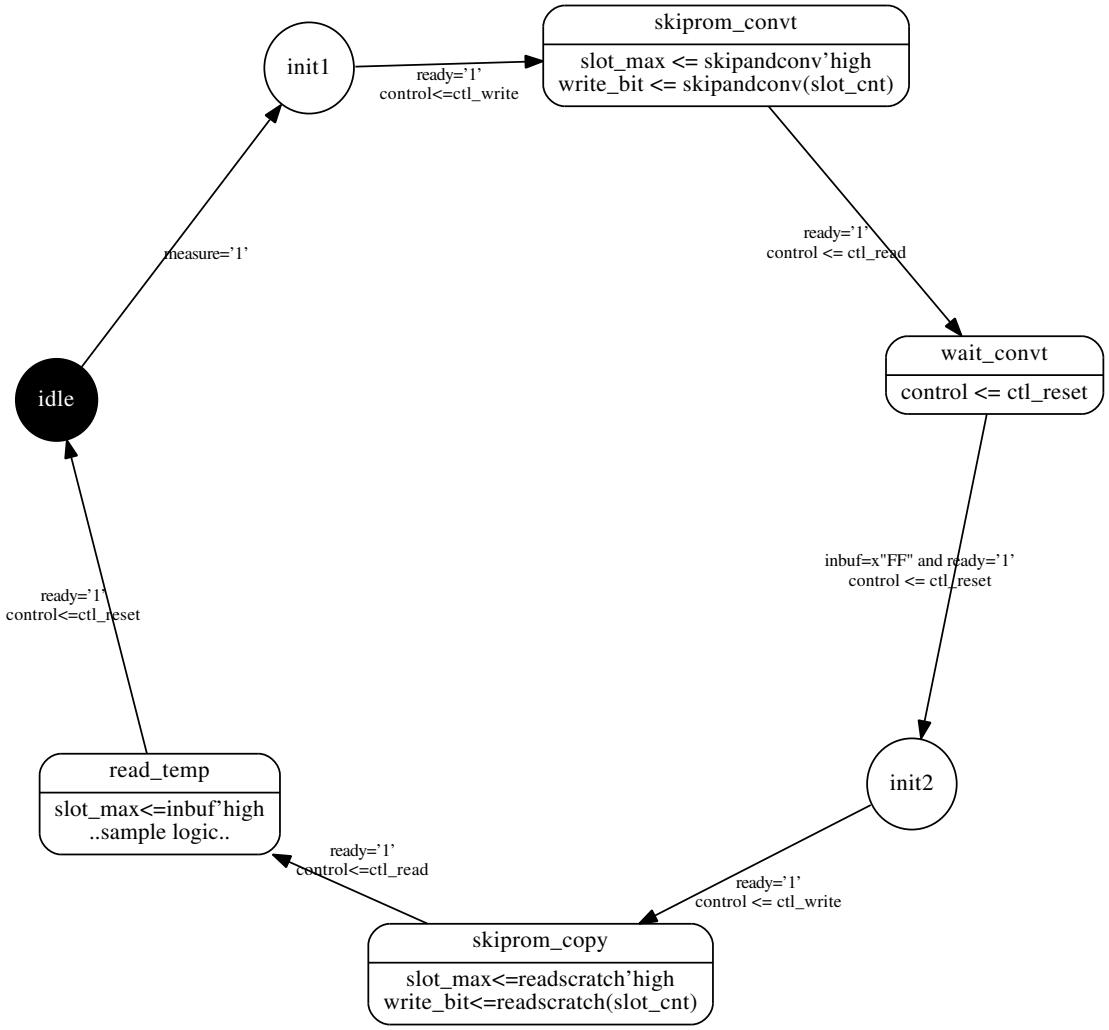
Interface Delblocket onwire sköter lågnivå kommunikationen med temperatursensorn, och exponerar ett högre nivå interface med fyra kontroll-kommandon och en ready signal. För att initiera en operation sätts `control` signalen till ett av följande värden när `ready='1'`:

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**



Figur 5: Tillståndsmaskin för delblocket DS18S20

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

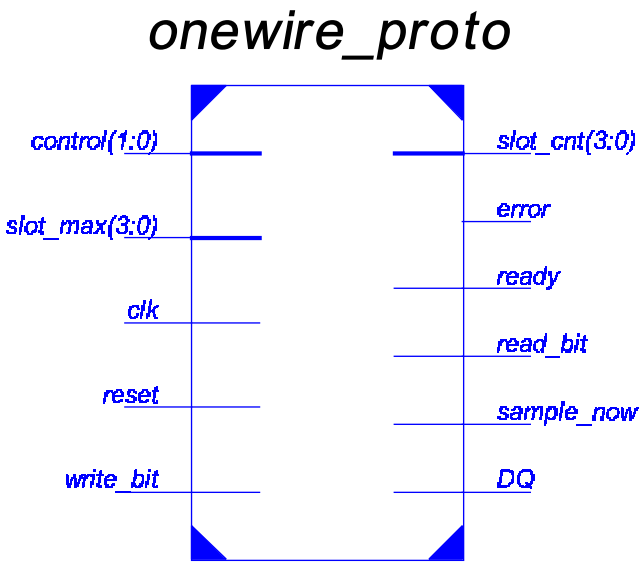
ctl_write Skriv `slot_max` antal bitar till sensorn. `slot_cnt` är en räknare som indikerar vilken bit som skrivs just nu. Användaren förväntas lägga biten som ska skrivas till sensorn på `write_bit`. När alla bitar är skrivna kommer `ready` sättas till `'1'`.

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.

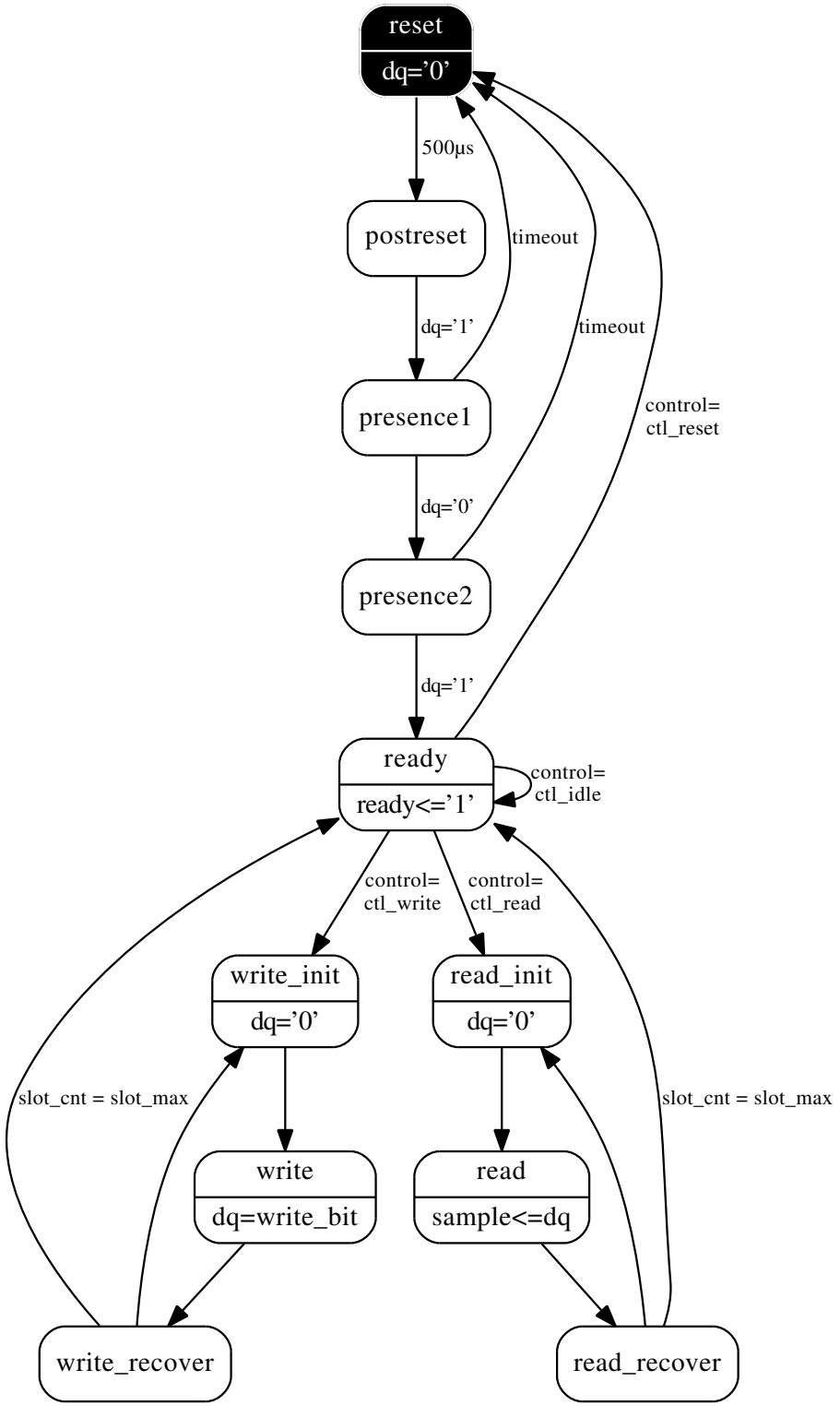


Figur 6: Delblocket onewire

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 5KΩpullup 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}.

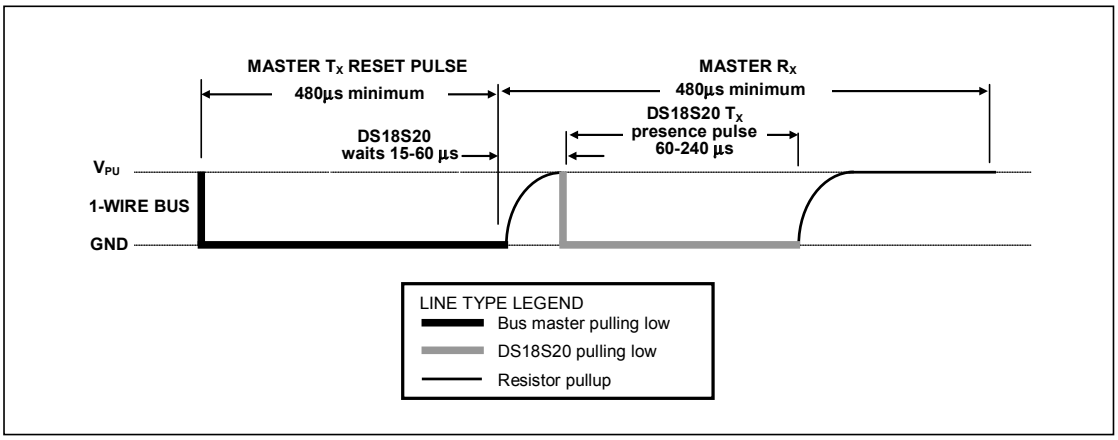
Kontrollen är uppbyggd av en tillståndsmaskin, se Figure 7. VHDL koden är uppbyggd efter tvåprocessmodellen, med en klockad och en kombinatorisk process.



Figur 7: Tillståndsmaskin för delblocket onewire

Reset Resetfasen korresponderar till *reset* → *postreset* → *presence*1 → *presence*2 → *ready* i onewires tillståndsmaskin (Figure 7).

Vid FPGAns *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 felaktikt eller uteblivet svar kommer `error` vara `'1'` under en klockcykel. Tillståndsmaskinen kommer sedan återgå till **reset** och börja om initieringssekvensen.



Figur 8: Timings för 1-wire resetsekvens

Skrivning med *write slots* Skrivfasen korresponderar till *ready* → *write_init* → *write* → *write_recover* i onewires tillståndsmaskin (Figure 7).

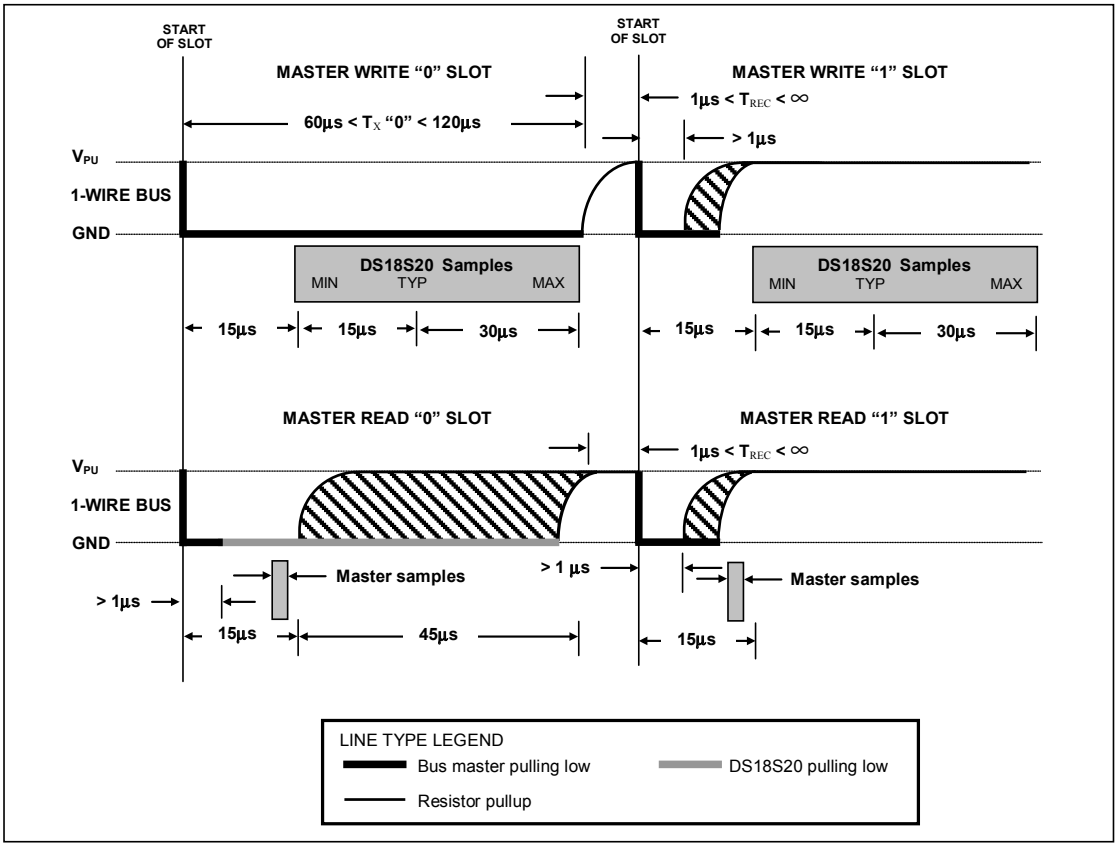
För att påbörja en skrivning sätts `control` till `ctl_write` när `ready='1'`. `slot_max` bitar kommer då skrivas till sensorn. 1-wire protokollet anvander *write slots* för att skriva bitar till sensorn. Flera bitar skrivs genom flera efterföljande write slots.

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

För att skriva `'0'` driver kontrollen DQ låg i 60μs $< T_x <$ 120μs. 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 anvander *read slots* för att skriva bitar till sensorn. Flera bitar skrivs genom flera efterföljande read slots. En read slot intitieras alltid av mastern genom att driva DQ låg i 1μs $< T_x <$ 15μs. 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 timings för '0' respektive '1'.

3.3. 7 Segments display

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 segments display 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 $-75--125\pm5$.

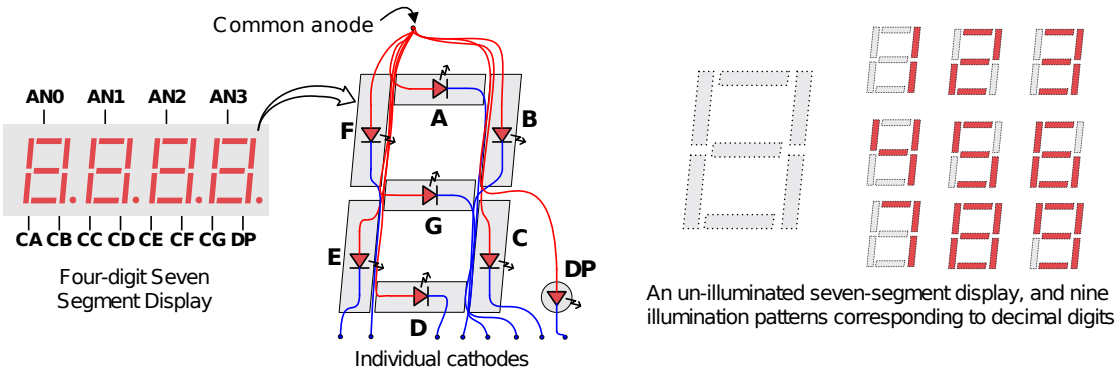
3.3.1. segment_temperature

Interface Komponenten tar ett binärt tal som input och ger utsignalerna till 7 segments displayerna.

<bild>

Implementation De fyra fysiska 7 segmentsdisplayerna 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.



Figur 10: 7-segments display

3.4. Kommunikation

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 elemenet ska regleras och/eller om elementens aktuella status ska skrivas till databussen.
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 utsignaler för att tala om exakt vad som gjorts.

4. Hårdvara

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

4.1. Digilent Nexys3 - FPGA

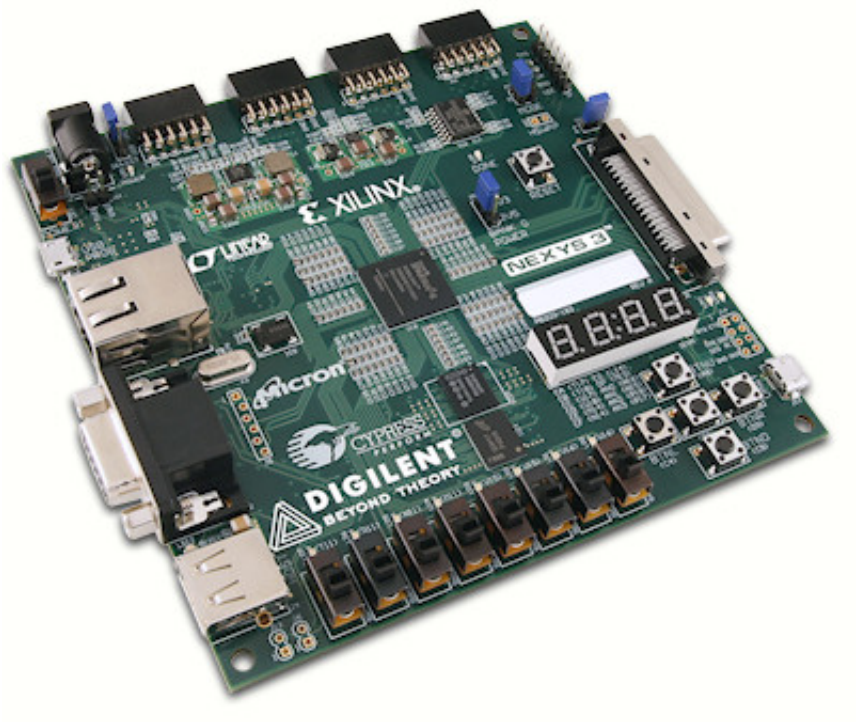
Nexys3 är ett utvecklingkort från Digilent som bygger på en Spartan-6 FPGA från Xilinx.
Nexys3 har bland annat:

- 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 sjusegments displayer.
- 8st binära DIP switchar.
- 8st ytmonterade lysdioder
- 4st kontaktstycken för externa I/O-enheter (dubbelbreda Pmod anslutningar).

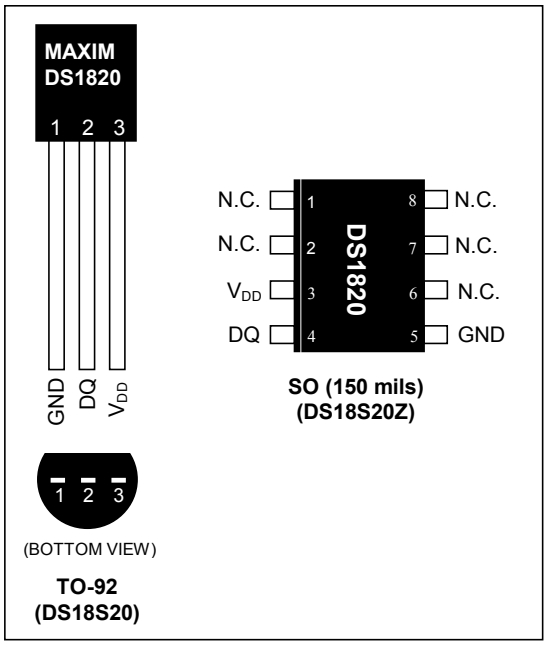
4.2. DS18S20 - Temperatursensor

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

- Temperaturmätning från -75°C till 125°C med ±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" anvands. Då laddar sensorn upp en kondensator när DQ drivs aktivt hög.



Figur 11: Digilent Nexys 3



Figur 12: Dallas DS18S20

5. Resultat och diskussion

5.1. Resultat

5.2. Felanalys

5.3. Diskussion

A. Signallista

A.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 tvåkompliments binärform

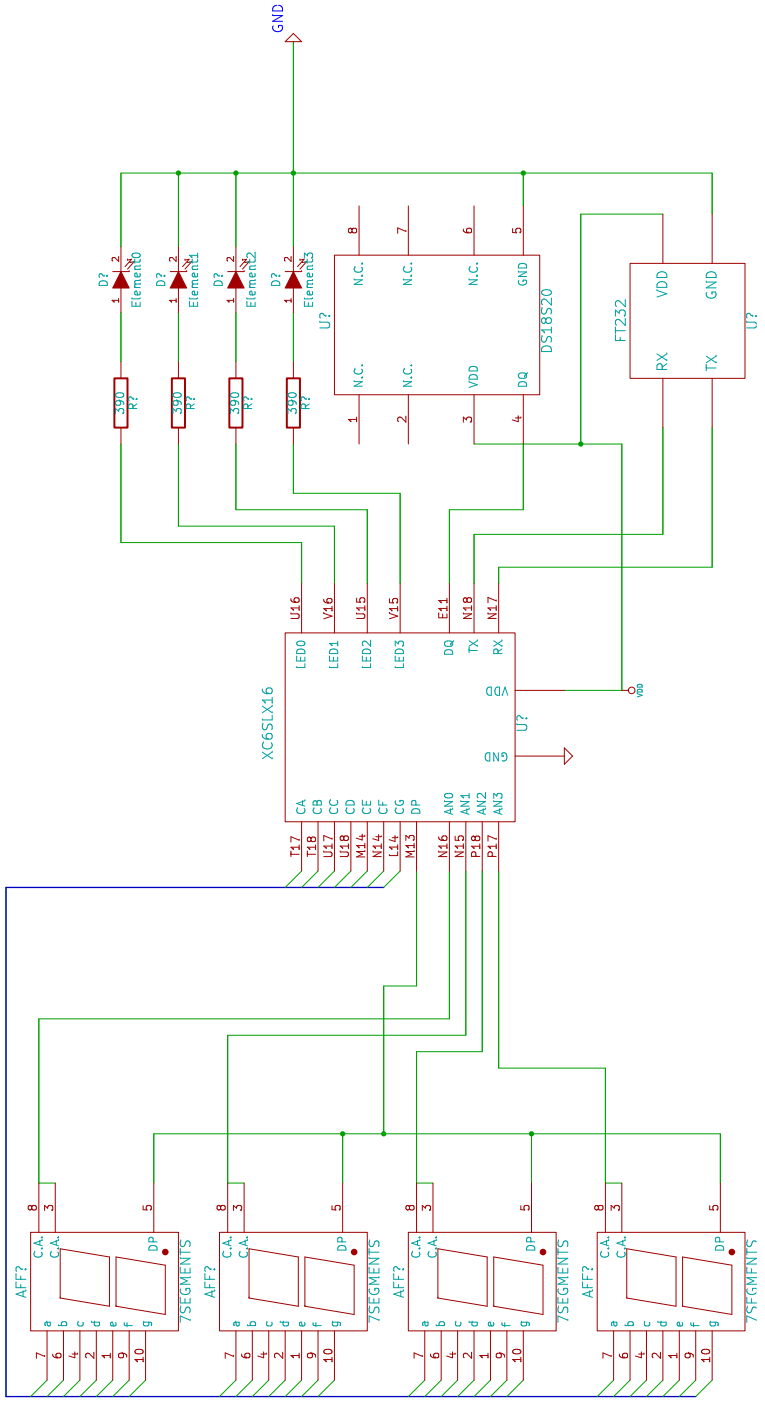
A.2. segment_temperature

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

A.3. Com

Namn	Typ	Kommentar
clk	in std_ulogic	Global klocka, 100MHz
rst	in std_ulogic	Global asynkron reset
tempInAvail	in std_logic	Temperatur finns tillgänlig på bus
elementInAva	in std_logic	Elementdata finns tillgänlig 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	Seriel port ut till GSM enhet ut från AT-modul
rx	in std_logic	Seriel port in till AT-modul från GSM-modul

B. Kretsschema



Figur 13: Förenklat kretsschema, över de komponenter som används, baserat på Digilents kretsschema för Nexys3.

C. Syntesschema

D. Komponentlista

Namn	**Beteckning**
Utvecklingskort	Digilent Nexys3
Temperatursensor	Maxim DS18S20
Motstånd	4.7 kΩ

E. Pinlayout

Pin	Beskrivning	Från	Till
T1	Pin	f	t

F. Programkod (VHDL)

```
package onewire is
   type control_t is (ctl_idle, ctl_read, ctl_write, ctl_reset);
   subtype slot_t is integer range 0 to 15;
end package;


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.onewire.all;


entity onewire_proto is
   port    ( clk, reset : in std_ulogic
           ; ready : buffer std_ulogic
           ; control : in control_t
           ; write_bit : in std_ulogic
           ; read_bit, sample_now : buffer std_ulogic
           ; slot_max : in slot_t
           ; slot_cnt : buffer slot_t
           ; error : buffer std_ulogic
           ; DQ : inout std_logic
           );
end entity;

architecture a of onewire_proto is
   -- // Constants. Times are in usec //
   -- min: 480 max: 960
   constant Treset_pulse : integer := 500;

   -- Longest time allowed for slave to aswers with a presence pulse
   constant Tpresence_timeout : integer := 60;

   constant Tpresence_min : integer := 60;
   -- Longest allowed duration of response pulse plus 1
   constant Tpresence_max : integer := 240;

   --Smallest duration of presence operations
   constant Tpostpresence : integer := Treset_pulse - Tpresence_min;
```

```
-- Write and read slots. Each slot have a >60usec duration, with >1usec
--   obligatory recovery time.
-- Values calibrated against one sensor

-- min: 1 max: inf
constant Trc : integer := 5;
-- min: 60 max: 120 (for 0 write slot), otherwise inf
constant Tslot : integer := 70;

-- min: 1 max: 15
constant Twrite_init : integer := 5;
-- min/max: 60
constant Twrite_hold : integer := Tslot - Twrite_init;

-- Read slot
-- The slave will get the value within 15us after the bus master pulls DQ low.
constant Tread_init : integer := 5;
constant Tread_sample : integer := 13 - Tread_init;
constant Tread_recover : integer := Tslot + Trc - Tread_sample - Tread_init;



-- // Types //
type state_t is ( Sreset
                , Spostreset
                , Spresence1
                , Spresence2
                , Spostpresence
                , Sready
                , Swrite_init
                , Swrite
                , Swrite_recover
                , Sread_init
                , Sread
                , Sread_recover
                );

-- / Signals //
signal DQO, DQId, DQI : std_ulogic;
signal DQFlipFlop : std_ulogic := '1';

signal state, next_state_signal : state_t;
signal reset_timer, pulse : boolean;
```

```
    signal cnt : integer range 0 to 512-1;

    signal slot_cnt_inc, slot_cnt_reset : boolean;
begin
    usec_timer:
    entity work.timer port map (clk, reset, reset_timer, pulse);

    user_counter:
    process (clk, reset) begin
        if reset = '1' then
            cnt <= 0;
        elsif rising_edge(clk) then
            if reset_timer then
                cnt <= 0;
            elsif pulse then
                cnt <= cnt + 1;
            end if;
        end if;
    end process;


    slot_counter:
    process (clk, reset) begin
        if reset = '1' then
            slot_cnt <= 0;
        elsif rising_edge(clk) then
            if slot_cnt_reset then
                slot_cnt <= 0;
            elsif slot_cnt_inc then
                slot_cnt <= slot_cnt + 1;
            end if;
        end if;
    end process;


    DQ <= '0' when DQFlipFlop = '0' else 'Z';

    -- Flip flop input and output for glitch free operation.
    process (clk, reset) begin
        if reset = '1' then
            state <= Sreset;
            DQFlipFlop <= '1';
        elsif rising_edge(clk) then
            state <= next_state_signal;
```

```
        DQFlipFlop <= DQO;
        DQId <= DQ;
        DQI <= DQId;
    end if;
end process;
read_bit <= DQI;


process (cnt, slot_cnt, slot_max, DQI, state, write_bit, control, pulse) is
    variable next_state : state_t;
begin
    error <= '0';
    reset_timer <= false;
    slot_cnt_reset <= false;
    slot_cnt_inc <= false;
    ready <= '0';
    sample_now <= '0';
    next_state := state;
    DQO <= '1';
    case state is
        -- Pull the pin low to reset the sensor(s)
        when Sreset =>
            DQO <= '0';
            if (cnt = Treset_pulse-1) and pulse then
                next_state := Spostreset;
            end if;

        -- wait for DQ to go high
        when Spostreset =>
            reset_timer <= true;
            if DQI = '1' then
                next_state := Spresence1;
            end if;

        -- wait for the slave to start the presence pulse
        when Spresence1 =>
            if DQI = '0' then
                next_state := Spresence2;
            elsif (cnt = Tpresence_timeout-1) and pulse then
                next_state := Sreset;
                error <= '1';
            end if;

        -- wait for the slave to finish the presence pulse
```

```
-- 107us for one sensor
when Spresence2 =>
    if DQI = '1' then
        if (cnt >= Tpresence_min-1) and pulse then
            next_state := Spostpresence;
        elsif pulse then
            next_state := Sreset;
            error <= '1';
        end if;
    elsif (cnt = Tpresence_max-1) and pulse then
        next_state := Sreset;
        error <= '1';
    end if;

when Spostpresence =>
    if (cnt = Tpostpresence-1) and pulse then
        next_state := Sready;
    end if;

-- Ready and waiting for commands
when Sready =>
    reset_timer <= true;
    slot_cnt_reset <= true;
    ready <= '1';
    case control is
        when ctl_idle => null;
        when ctl_write => next_state := Swrite_init;
        when ctl_read => next_state := Sread_init;
        when ctl_reset => next_state := Sreset;
    end case;

-- start a write slot by pulling the pin low
when Swrite_init =>
    DQO <= '0';
    if (cnt = Twrite_init - 1) and pulse then
        next_state := Swrite;
    end if;

-- write the bit (LSB first)
when Swrite =>
    DQO <= write_bit;
    if (cnt = Twrite_hold - 1) and pulse then
        next_state := Swrite_recover;
    end if;
```

```
        -- recover from the write slot
        when Swrite_recover =>
            if (cnt = Trc - 1) and pulse then
                if slot_cnt = slot_max then
                    next_state := Sready;
                else
                    slot_cnt_inc <= true;
                    next_state := Swrite_init;
                end if;
            end if;

        -- Start a read slot by pulling the pin low
        when Sread_init =>
            DQO <= '0';
            if (cnt = Tread_init - 1) and pulse then
                next_state := Sread;
            end if;

        -- Sample the data (LSB first)
        when Sread =>
            if (cnt = Tread_sample -1) and pulse then
                sample_now <= '1';
                next_state := Sread_recover;
            end if;

        -- Recover from the read slot
        when Sread_recover =>
            if cnt = Tread_recover then
                if slot_cnt = slot_max then
                    next_state := Sready;
                else
                    slot_cnt_inc <= true;
                    next_state := Sread_init;
                end if;
            end if;
    end case;

    -- Always reset the timer for the next state
    if next_state /= state then
        reset_timer <= true;
    end if;
    next_state_signal <= next_state;
end process;
```

end architecture;

G. Arbetsredogörelse

G.1. Christian

G.2. Christoffer Öjeling

- Komponenten för mätning och avläsning av temperatur från DS18S20. Inkluside hjälpfunktioner.
- Lött DS18S20-konstruktionen på ett mönsterkort.
- Visning av ett binärt tal på FPGAns 7 segment display.
- Testing och verifikation av DS18S20 kontrollen.
- Binär till BCD konverterare.
- Allt i rapporten som relaterar till temperatursensorn DS18S20 och 7 segments displayn:
  - 3.2 Temperatur
  - 3.3 7 Segments display
  - 4.2 DS18S20 - Temperatursensor
  - A.1 DS18S20
  - A.2 segment_temperature

G.3. Jan

- 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
  - Uart_tx_ctrl.vhd (kod tagen från diligents hemsida)

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.