



LABORATION

TSEA22 DIGITALTEKNIK D

Konstruktion av mindre digitala system med CPLD

Version: 1.5

2015 (OVA, MK)

1. Inledning

Syftet med laborationen är dels att öva på konstruktion av mindre digitala system, och dels att ge en utökad inblick i moderna konstruktionshjälpmedel för digital konstruktion.

Efter genomförd laborations ska ni:

- Bland annat genom att rita blockschema, kunna konstruera mindre digitala system med hjälp av programmerbar logik.
- Ha befäst grunderna i det hårdvarubeskrivande språket VHDL.
- Kunna använda ModelSim för enklare simuleringar.
- Ha insikt i modern systemutvecklingsmetodik.

Laborationsutrustningen kommer att kompletteras med en modul som innehåller en CPLD (Complex Programmable Logic Device) av typen XC9572 tillverkad av Xilinx. Ytterligare moduler kommer att tillföras efter behov.

I detta lab-PM hittar ni bland annat följande avsnitt:

- Laborationsuppgifter
- Diverse kodexempel som kan användas som ”labskelett”
- Diverse datablad

Förberedelseuppgift: Det ska finnas ett ritat blockschema till varje uppgift som ni har för avsikt att göra på laborationstiden. Till uppgift 20 finns det även en simuleringsövning som ska redovisas.

Laborationsuppgift 20, samt minst en av uppgifterna 21-23 är obligatoriska medans resterande uppgifter är extrauppgifter.

Eftersom utrustningen som behövs till uppgift 22 i dagsläget bara finns i ett exemplar är det max två grupper/labtillfälle som får en möjlighet att prova på denna uppgift. Först till kvarn gäller via en obligatorisk förhandsbokning.

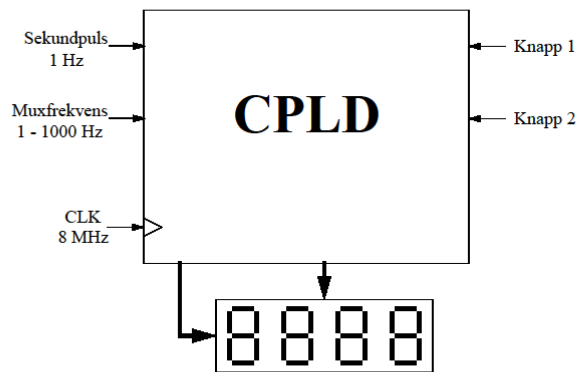
I den här kursen ligger fokus på grundläggande digitalteknik med enkla tillämpningar som exemplifierar kursinnehållet. Den VHDL som ingår i kursen är därför bara ett absolut minimum för att minska antalet sladdar som behöver kopplas. För att bli godkänd på lab 4 krävs att alla uppgifter redovisas tillsammans med ett detaljerat blockschema över konstruktionen där man på ett tydligt sätt kan se kopplingen till er VHDL-kod. Endast de VHDL-konstruktioner som lärs ut i kursen får användas. Lämpliga typer av block: Räknare, Register, Komparatorer, Multiplexrar, PROM, Vippor och grindar, Kombinatoriska funktioner samt Sekvenskretsar med tillhörande tillståndsdigram. Exempel på blockschemor finns i avsnitt 3, Kodexempel.

VHDL är ett stort språk som bland annat tillåter hierarkiskt ordnade komponenter. **Man bör dock inte använda alltför djupa hierarkier och inte heller skriva alltför små komponenter.** Det krävs då mycket kod för att binda samman komponenterna och risken är därmed stor att koden blir svårläst. Det påstås också att syntesresultatet i vissa fall kan bli sämre. De uppgifter som ska lösas i denna kurs är så pass små så att man med fördel ska lägga allt i en och samma VHDL-fil.

Användningen av ”components” kommer att läras ut i senare kurser där behovet är mer påtagligt. **Man bör också undvika alltför stora processsatser**, dvs en processats/block brukar vara lagom.

2. Laborationsuppgifter

Uppgift 20: Ett stoppur som styrs av två knappar ska konstrueras. Uret ska i sitt grundutförande visa minuter och sekunder, samt att den ena knappen används för start/stopp och den andra för nollställning. Om tiden är stoppad och startas igen fortsätter räkningen från visad tid.



Konstruera det digitala systemet. Använd en CPLD, en multiplexad display, dvs den ni använde i uppgift 4 på laboration 1, samt valfria knappar. Insignalerna måste synkroniseras.

Använd kristalloscillatorn inställd på 8 MHz som systemklocka. För multiplexning används med fördel den variabla klockpulsgeneratoren, och för tidtagningen en tidbasmodul. **Alla vippor ska klockas med systemklockan**, varför sekundpuls och muxfrekvens ska behandlas som "count enable"-signaler.

Multiplexning innebär att man visar en siffra i taget och om detta sker tillräckligt fort kommer ögat att uppfatta detta som att alla siffror visas samtidigt. Mät med hjälp av en frekvensräknare upp den displayuppdateringsfrekvens som krävs för att erhålla en flimmerfri visning av siffrorna.

Resultat:.....

Som laborationsförberedelse ska en simulering i ModelSim över två kaskadkopplade BCD-räknare genomföras, välj sekundsiffrorna. (Jämför gärna med uppgift 5.3 på laboration 1 där ni kopplade upp tre kaskadkopplade BCD-räknare.) Resultatet ska redovisas under labtid med hjälp av kod och en sparad bild från simuleringen. (Man ska alltså kunna se en rippel-carry vid omslaget 59->00.)

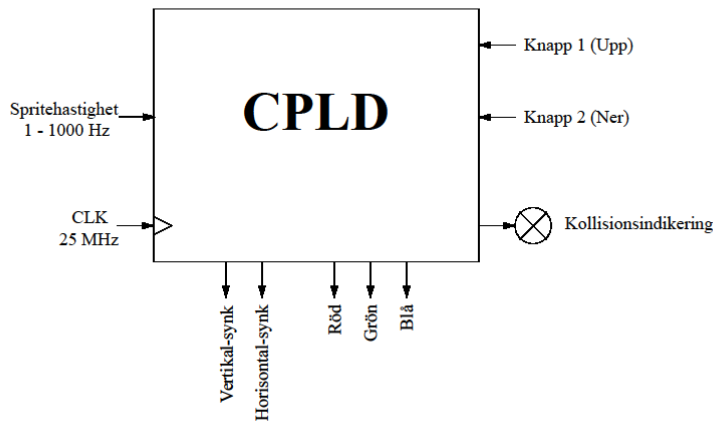
Logiskt blockschema:

Uppgift 21: Utöka konstruktionen från uppgift 20 med ytterligare funktionalitet:
(Uppgifterna redovisas i tur och ordning.)

- a) Komplettera uppgift 20 med utökad precision till minuter, sekunder och hundradelar. Använd en skjutomkopplare för att välja vilka fyra siffror som ska visas, dvs minuter+sekunder eller sekunder+hundradelar.
- b) Komplettera uppgift 20 med en mellantidsfunktion. Två tryckomkopplare ska användas. Den ena används för start/stopp som tidigare. Den andra används för mellantid/nollställning på så sätt att om klockan går visas mellantid efter en tryckning samt återgång efter nästa tryckning. Om klockan står still ska uret nollställas. Om uret stoppas medans mellantid visas ska sluttiden visas efter en tryckning på mellantidsknappen.
- c) Komplettera uppgift 21b med en finess så att inledande nollor ska vara helt släckta.

Logiskt blockschema:

Uppgift 22: Konstruera en så kallad "sprite" med hjälp av VHDL och en CPLD. Till er hjälp har ni kodexempel i avsnitt 3. En sprite används vanligtvis i lite enklare datorspel där man vill få geometriska figurer att kunna röra sig över en skärm och där hårdvaran är specialanpassad för just den applikationen.



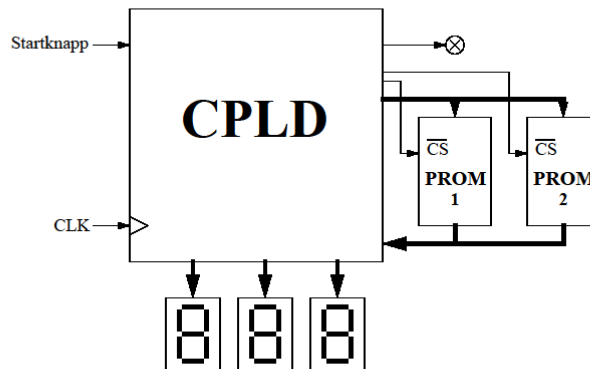
- Skapa tre fasta objekt på skärmen. En grön kvadrat (70x70 pixlar), en blårandig rektangel (40x120 pixlar) samt en tjock vit vertikal linje (10x250 pixlar). Den vertikala linjen ska ligga mitt på övre halvan av skärmen, medans övriga objekt får ligga på valfritt ställe. Den randiga rektangeln ska ha minst tio ränder.
- Behåll den vita linjen samt ersätt de två andra fyrkanterna med en kvadrat (30x30) som rör sig i X-led över hela skärmen. Då kvadraten passerar linjen ska en lysdiod vara tänd för att markera "kollision". Hastigheten på rörelsen ska kunna ändras med hjälp av den variabla klockpulsgeneratoren. En tryckomkopplare ska kunna välja Y-koordinaten till antingen övre eller undre halvan av skärmen. TIPS: Det behövs två vippor för att få till en bra och stabil kollisionsindikering.
- Utöka antalet vertikala linjer till minst två i övre respektive undre halvan så att en liten slalombana uppstår. Använd kollisionsindikeringen för att inverta hela skärmen vid "kollision". Inverta betyder i detta fall att de synliga pixlars färg ska ändras till deras komplementfärg.
- Låt nu Y-koordinaten kunna styras upp eller ner med hjälp av två tryckknappar. Eftersom den lilla CPLD som står till ert förfogande nu börjar bli väl utnyttjad, krävs det ett litet "trick" för att få syntesverktyget att klara av att lösa uppgiften, se kodexemplet.

För ett eventuellt behov av felsökning så går det INTE att klocka manuellt. Lämpliga hjälpmedel är därför ModelSim och/eller Oscilloskop.

Den här uppgiften ska betraktas som lite svårare!

Logiskt blockschema:

Uppgift 23: Minnesinnehållet i labsatsens PROM ligger kvar även vid spänningsbortfall. Konstruera ett digitalt system som räknar det totala antalet ettor i två parallellkopplade PROM, se figur. Resultatet ska presenteras i BCD-form. Varje räkning ska startas med en knappnedtryckning, vilken alltså inkluderar såväl nollställning som start. En lysdiod ska vara tänd under tiden som räkningen pågår.



Använd VHDL och en CPLD samt lämpliga in och utsignaler. Klockfrekvensen bör vara max 100 Hz, ty vi vill kunna se när konstruktionen arbetar. Osynkroniserade insignaler måste synkroniseras. Eftersom adressen till PROM:n är synkron och klockfrekvensen relativt låg, kan utdata från PROM:n räknas som synkrona, dvs PROM räknas som en rent kombinatorisk krets. Parallellkopplade PROM innebär att vi behöver utnyttja PROM:ns "tri-state" funktionalitet på datautgången, vilket styrs med "chip select"-signalerna.

TIPS: Börja gärna med ett PROM och två displayer för att därefter utöka konstruktionen.

Logiskt blockschema:

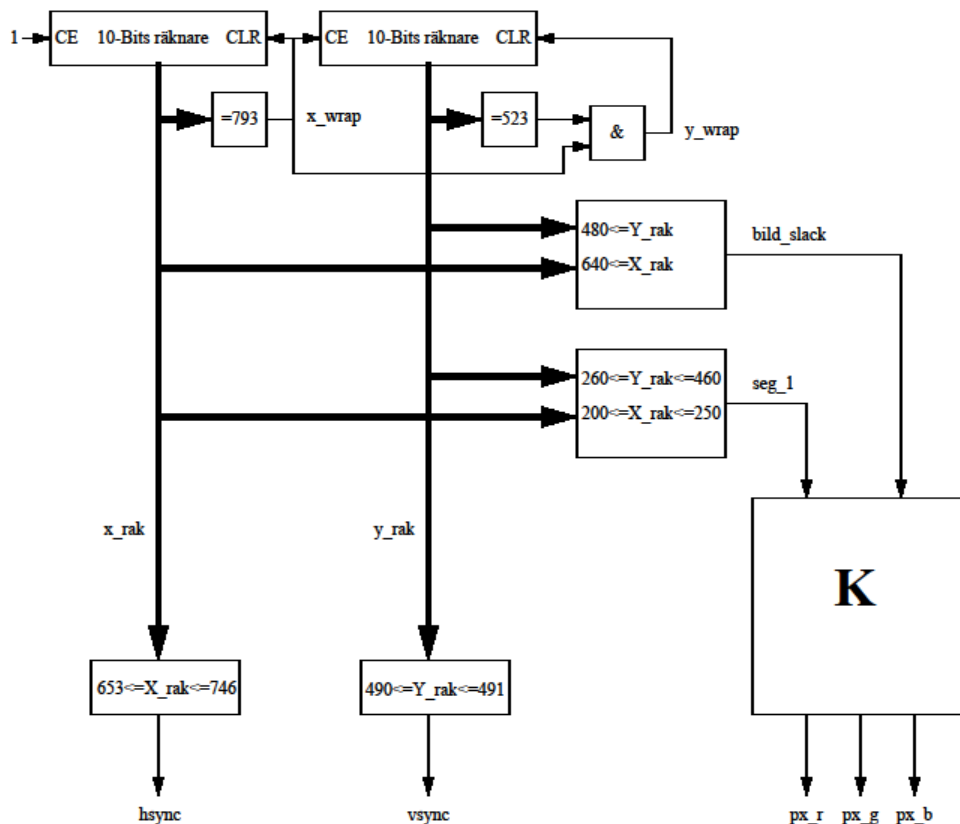
3. Diverse kodexempel i VHDL

Exempel-1 handlar om att få upp en stabil bild med pixelräknare, synkpulser och bildsläcksignal. Den synliga delen av bilden är 640x480 pixlar, men på grund av kompatibilitet med den gamla TV-tekniken så går det åt en del osynliga pixlar för att få tid till synkpulser, dvs råbilden är 794x524 pixlar.

Det vi skapar är en VGA-signal och för de som vill simulera i ModelSim kan det vara bra att veta att det krävs "rätt" timing för signalerna hsync och vsync. Eftersom kunskap om VGA ligger utanför kursen ges en hel del exempelkod "gratis". Bilden byggs upp som ett koordinatsystem där pixelräknaren bestämmer vilken punkt på skärmen som vi just för tillfället ritar ut. Utritningen sker rad för rad, och för den aktuella skärmen som kommer att användas ritas hela bilden ungefär 60 gånger/sekund.

Nollpunkten ligger i det övre vänstra hörnet av skärmen. X-koordinaten växer åt höger samt Y-koordinaten växer nedåt i bilden. Den delen av råbilden som ligger utanför den synliga delen måste vara svart, och för detta ändamål finns signalen bild_slack definierad.

Exempelkoden visar också hur en vit rektangel kan skapas med hjälp av en stor komparator. De tre färgerna röd, grön och blå tillsammans med synkpulserna måste slutligen anpassas till rätt impedans, vilket sker i den anslutningskabel som tillhandahålls i labbet.



Exempel-1: VGA-signaler.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity VGA_Net is
    Port (clk : in std_logic;           -- 25 MHz
          hsync, vsync : out std_logic;
          px_r, px_g, px_b : out std_logic);
end VGA_Net;

architecture equations1 of VGA_Net is
    signal bild_slack, seg_1, x_wrap, y_wrap : std_logic;
    signal x_rak, y_rak : std_logic_vector(9 downto 0);
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if (x_wrap = '1') then
                x_rak <= "0000000000";
            else
                x_rak <= x_rak + 1;
            end if;
        end if;
    end process;
    x_wrap <= '1' when (x_rak = 793) else '0';

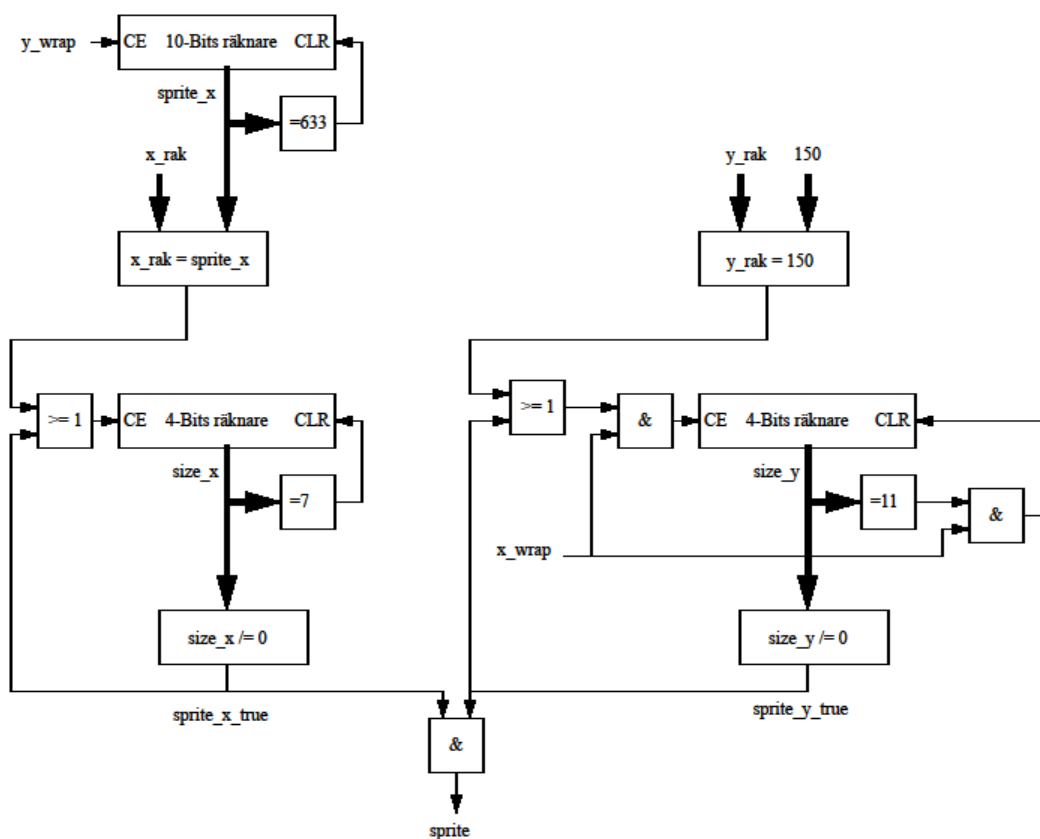
    process(clk)
    begin
        if rising_edge(clk) then
            if (y_wrap = '1') then
                y_rak <= "0000000000";
            elsif (x_wrap = '1') then
                y_rak <= y_rak + 1;
            end if;
        end if;
    end process;
    y_wrap <= '1' when ((y_rak = 523) and (x_wrap = '1')) else '0';

    bild_slack <= '1' when ((x_rak >= 640) or (y_rak >= 480)) else '0';
    hsync <= '0' when (x_rak >= 653) and (x_rak <= 746) else '1';
    vsync <= '0' when (y_rak >= 490) and (y_rak <= 491) else '1';
    -----
    seg_1 <= '1' when ((x_rak >= 200) and (x_rak <= 250) and
                      (y_rak >= 260) and (y_rak <= 460)) else '0';

    px_r <= '0' when (bild_slack = '1') or (seg_1 = '0') else '1';
    px_g <= '0' when (bild_slack = '1') or (seg_1 = '0') else '1';
    px_b <= '0' when (bild_slack = '1') or (seg_1 = '0') else '1';
end equations1;

```

Exempel-2 visar en enkel "sprite" med storleken 7x11 pixlar som åker åt höger med ca 60 Hz. I exempel-1 såg vi att man kan använda en komparator för att avgöra var på skärmen en geometrisk figur ska ritas upp, men om vi nu ska göra platsen på skärmen variabel, kommer komparatorn att ta för mycket resurser i anspråk. Däremot är "lika med" operatoren relativt billig att implementera, och lösningen blir därmed att använda den för att peka ut det övre vänstra hörnet samt att därefter använda två mindre räknare för att få till rätt storlek. Spriten blir alltså den area som uppstår om man snittar dess x-koordinater (sprite_x_true) med dess y-koordinater (sprite_y_true).



Exempel-2: En "Sprite".

```

architecture equations2 of vga is

    signal sprite_x : std_logic_vector(9 downto 0);
    signal size_x, size_y : std_logic_vector(3 downto 0);
    signal sprite, sprite_x_true, sprite_y_true : std_logic;

begin
    process(clk)
    begin
        if rising_edge(clk) then
            if (sprite_x = 633) then
                sprite_x <= "0000000000";
            elsif (y_wrap = '1') then
                sprite_x <= sprite_x + 1;
            end if;
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            if size_x = 7 then
                size_x <= "0000";
            elsif ((x_wrap = '1') or (sprite_x_true = '1')) then
                size_x <= size_x + 1;
            end if;
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            if ((x_wrap = '1') and (size_y = 11)) then
                size_y <= "0000";
            elsif ((x_wrap = '1') and (((y_wrap = 150) or (sprite_y_true =
'1'))
                size_y <= size_y + 1;
            end if;
        end if;
    end process;

    sprite_x_true <= '1' when (size_x /= 0) else '0';
    sprite_y_true <= '1' when (size_y /= 0) else '0';
    sprite <= sprite_x_true and sprite_y_true else '0';

end equations2;

```

Exempel-3 introducerar attributet KEEP. CPLD:n XC9572 består av makroceller med tillhörande logik i två grindnivåer. Mängden tillgänglig logik räknas i antalet produkttermer, och syntesverktyget försöker därför att ta fram en minimal SP-form. Optimeringen sker med avseende på snabbhet, om inte något annat specificeras, vilket medför att komplexa villkor som ingår på flera ställen kommer att kopieras till varje signal och därmed förbruka onödigt många produkttermer. För att minska kostnaden kan sådana deluttryck brytas ut, vilket kostar en makrocell/deluttryck samt att systemet blir något långsammare. Om vi har råd med en lägre hastighet kan det bli lönsamt att bryta ut sådana deluttryck, och till viss del klarar syntesverktyget av att själv göra detta, men ibland behöver verktyget få lite hjälp beträffande val av signaler som inte ska reduceras bort. Alla direktiv till verktyget görs som attribut, och i detta fall heter attributet "keep". Attributet i sig måste först deklarerats som en sträng innan det kan appliceras på de signaler som inte ska reduceras bort. Det är en fördel med att vara restriktiv med attribut eftersom man då begränsar verktygets möjligheter.

Exempel-3:

```
architecture equations3 of vga is
    signal sprite : std_logic;

    attribute keep : string;
    attribute keep of sprite : signal is "TRUE";

begin

end equations3;
```

4. Diverse datablad

Här beskrivs moduler som kommer att tillföras labsatsen efter behov.

4.1. Tidbasmodul

Tidbasmodulen används för att kunna generera de förutbestämda frekvenserna: 1 Hz, 10 Hz, 100 Hz samt 1 kHz.

4.2. VGA-anslutning (Pin List)

1. Red
2. Gnd
3. Green
4. Gnd
5. Blue
6. Gnd
7. H-Sync
8. Gnd
9. V-Sync
10. Gnd

4.3. Kristalloscillator 25 MHz (Pin List)

1. Enable
4. Gnd
13. Clock Out
16. Vcc