VGA-lab
TSEA83 Datorkonstruktion

Anders Nilsson

2021 version 1.3

Innehåll

1 Introduktion

1.1 Syfte

Den här laborationen kommer att beröra flera saker. Avkodning av signaler från ett PS/2-tangentbord. Användning av ett block-RAM som bildminne. Generering av VGA-signaler till en bildskärm. Dessa tre delar sätts samman i en större hierakisk konstruktion som bildar en simpel textmonitor där man kan skriva och redigera text. Alla delar kan vara nyttiga i det kommande datorkonstruktionsprojektet.

1.2 Förkunskaper

För att på ett bra sätt tillgodogöra sig laborationen behöver man ha grunderna från tidigare digitalteknikkurs. Man bör också ha gjort en föregående laboration i VHDL för att vara något bekant med språket och labmiljön.

1.3 Material

Allt material som behövs till laborationen finns i labsalen, även labinstruktionen (denna skrift), fast bara då i elektronisk form som PDF-fil från kursen websida. Vill man ha en tryckt kopia så får man skriva ut den själv. Detta är dock inte nödvändigt för att kunna utföra själva laborationen.

1.4 Förberedelser

Läs igenom följande avsnitt, Blockschemakonstruktion. Studera den givna VHDL-koden som kan laddas ned från kursens web-sida, och jämför blockschema och kod så att du förstår vad som händer. Förbered uppgifterna i avsnittet Uppgifter så att du kan använda labtiden till att prova och eventuellt felsöka i koden.

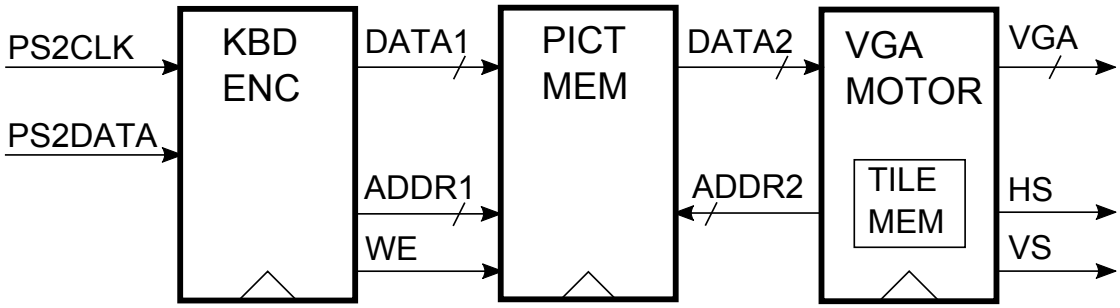
2 Blockschemakonstruktion

2.1 Förutsättningar

Textmonitorn ska visas på en VGA-skärm i upplösningen 640x480 pixlar. Detta uppnås med korrekt VGA-timing (se bilaga 4.2). Textmonitorn är konstruerad till att ha 15 rader á 20 kolumner.

2.2 Översikt

Hela konstruktionen av textmonitorn framgår av figur 2.1.



Figur 2.1: Textmonitor

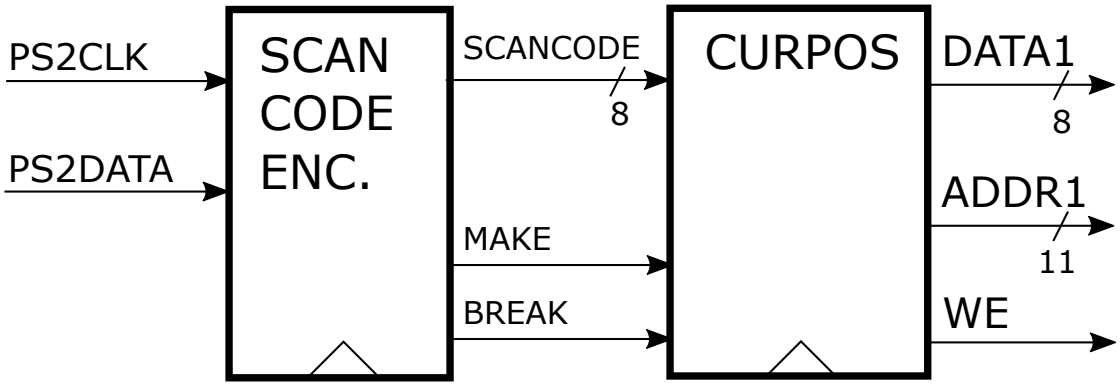
Tangentbordsavkodaren (KBD_ENC) läser av tangenbordets PS/2-signaler och skriver in lämpliga data i bildminnet (PICT_MEM). Samtidigt läser bildgeneratorn (VGA_MOTOR) ur bildminnet och skapar lämpliga synk- och VGA-signaler till en VGA-monitor.

I den följande framställningen kommer vi stegvis att dela upp konstruktionen i mindre realiserbara delar för att lättare kunna överföra dessa till fungerande VHDL-kod.

2.3 KBD_ENC : Tangentbordsavkodaren

Tangentbordsavkodarens uppgift är dels att läsa av PS/2-signalerna från tangenbordet, men även att skriva in lämpliga data i bildminnet där textmonitorns cursor befinner sig. Vi delar upp tangentbordsavkodaren i två block. En del som läser tangentbordets s k scankoder (SCAN_CODE_ENC), och en del som sköter cursorns position och själva skrivningen i bildminnet (CURPOS). Se figur 2.2

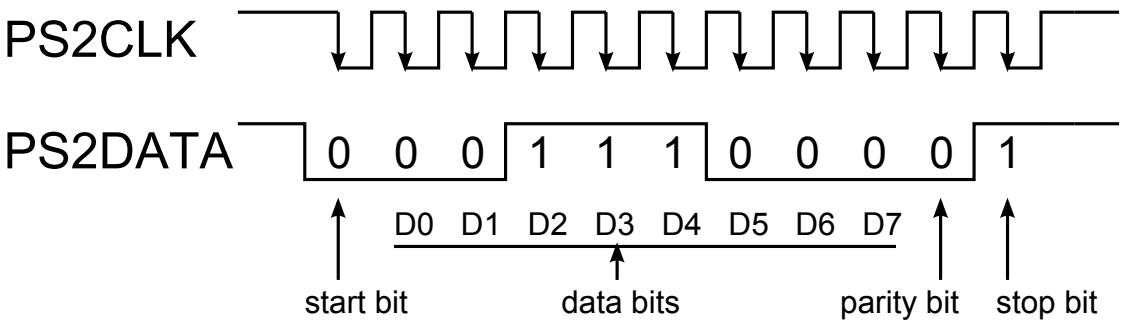
När man trycker ned en tangent på ett tangentbord så skickas data om tangettryckningen seriellt över två ledningar, PS2CLK och PS2DATA. Data som skickas utgörs



Figur 2.2: KBD_ENC, Tangentbordsavkodare

av scankoder (se appendix 4.1). Själva nedtryckningen av tangenten kallas för MAKE, och då skickas tangentens scankod. När man släpper upp tangenten kallas detta för BREAK, och då skickas koden $F0 följt av scankoden för den aktuella tangenten. För t ex tangenten A skickas scankoden $1C vid MAKE, och $F0 följt av $1C vid BREAK. Se exempel i figur 2.3. Observera att avläsningen av varje bit sker vid den negativa flanken på PS2CLK.

Vissa tangenter har utökade scankoder, dvs det skickas mer än en byte vid MAKE. I vår lösning kommer vi att bortse från det och endast hantera enstaka byte vid MAKE.

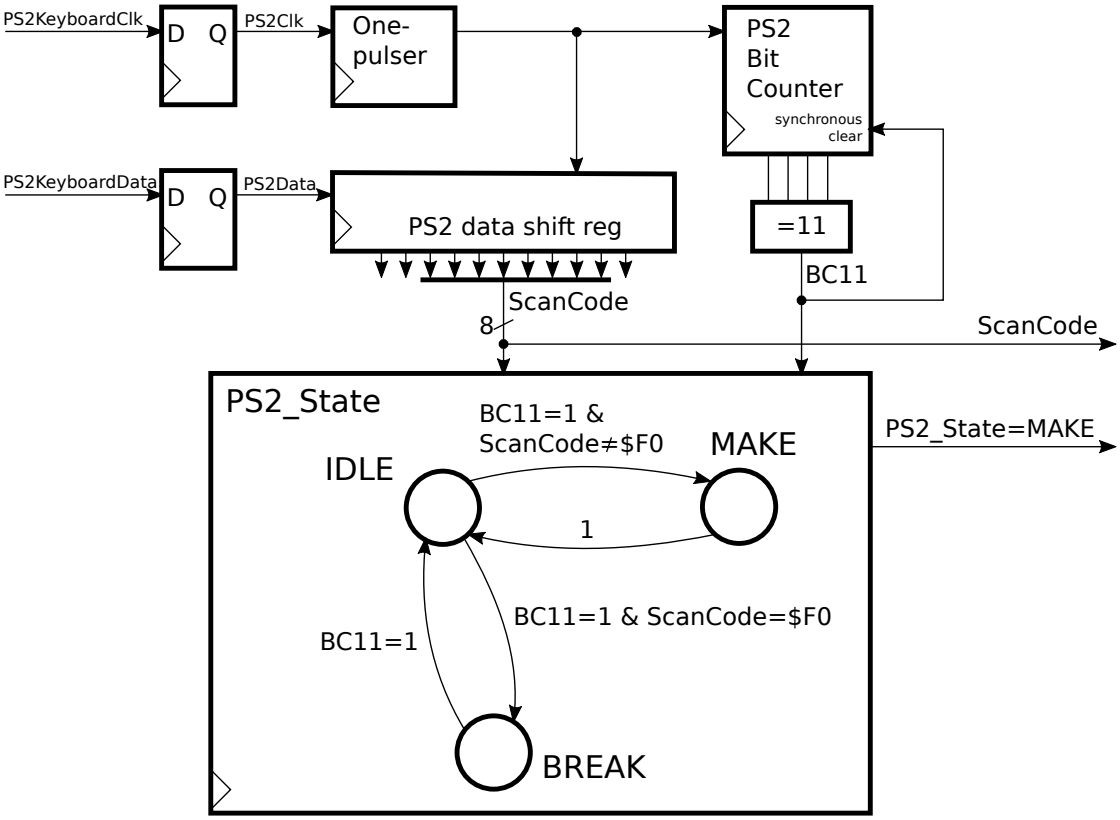


Figur 2.3: PS2, scankod för tangenten A

2.3.1 SCAN_CODE_ENC

Lösningen av tangentbordsavkodarens första block (SCAN_CODE_ENC) behöver delas upp ytterligare innan via kan göra VHDL-kod av det. Vi gör det enligt figur 2.4

Tangentbordets signaler synkroniseras med vippor. PS2-klockans negativa flank enpulsas och skiftar in databitarna i ett skiftregister samt räknar alla 11 bitarna i en bit-räknare. Själva scankoden och signalen som säger att alla 11 bitar tagits emot går in i en tillståndsmaskin (PS2_State) som avgör om MAKE eller BREAK har inträffat. Allt detta kan direkt översättas till VHDL.



Figur 2.4: SCAN_CODE_ENC

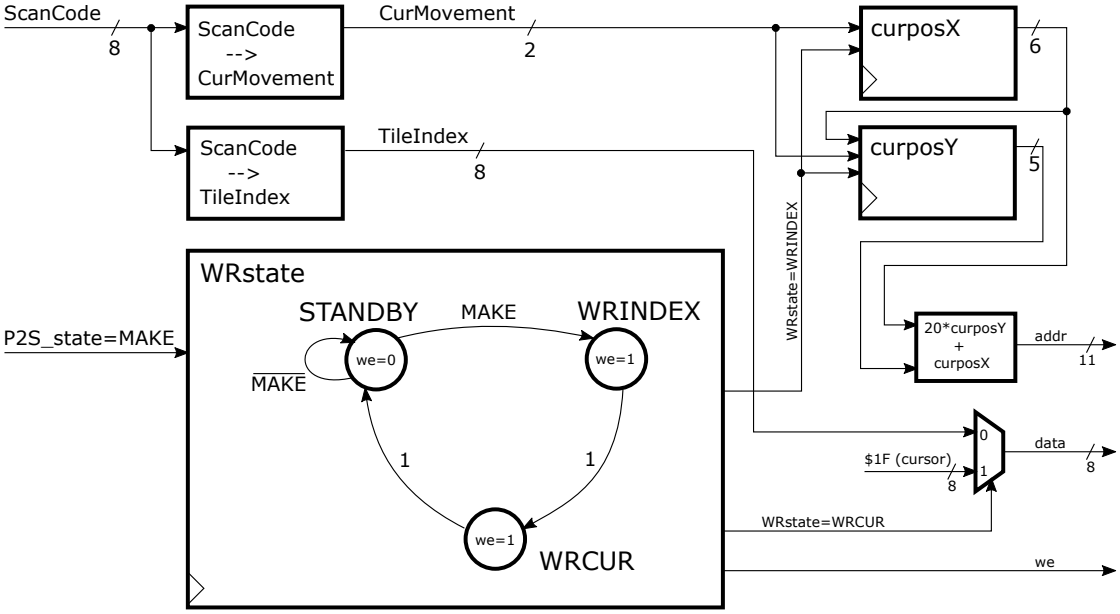
2.3.2 CURPOS

Tangentbordsavkodarens andra block, CURPOS, behöver också förtydligas. Vi gör det i figur 2.5.

Scankoden översätts i två tabeller. Dels till något som talar om hur cursorn ska röra sig (framåt, bakåt eller ny rad). Skriver man in en bokstav så ska cursorn röra sig framåt, vid backspace ska den gå bakåt och för enter-tangenten ska den gå till en ny rad. Dessutom översätts scankoden till ett uppräkningsbart index som lämpar sig bättre att skriva in i bildminnet än själva scankoden.

Var gång man trycker på en tangent så ska två skrivningar göras i bildminnet. Detta sker i tillståndsmaskinen WRState. Först ska cursorns nuvarande position skrivas över av den nya tangentens index, och därefter ska själva cursons index skrivas på cursorns nya position.

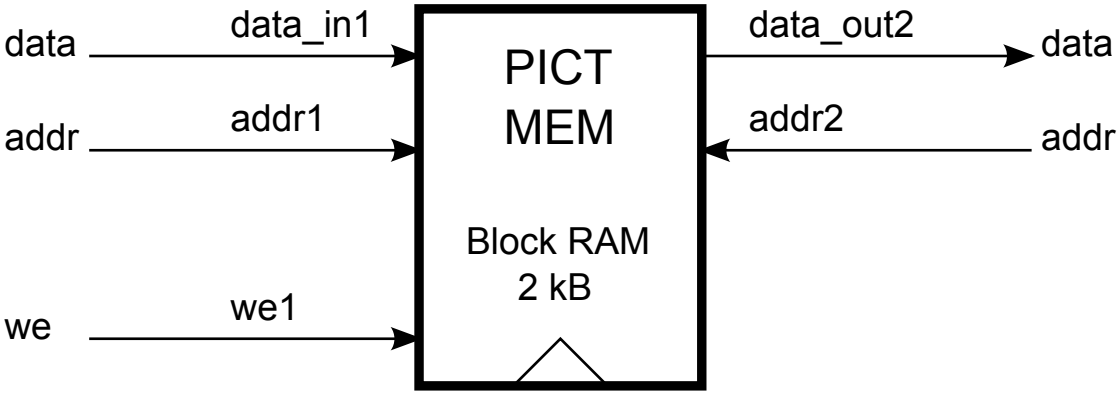
Cursons position sköts av processerna curposX och curposY. Dessa bildar tillsammans adressen till bildminnet via en enkel multiplikation och addition. Alla delar kan mer eller mindre direkt översättas till VHDL.



Figur 2.5: CURPOS

2.4 PICT_MEM : Bildminnet

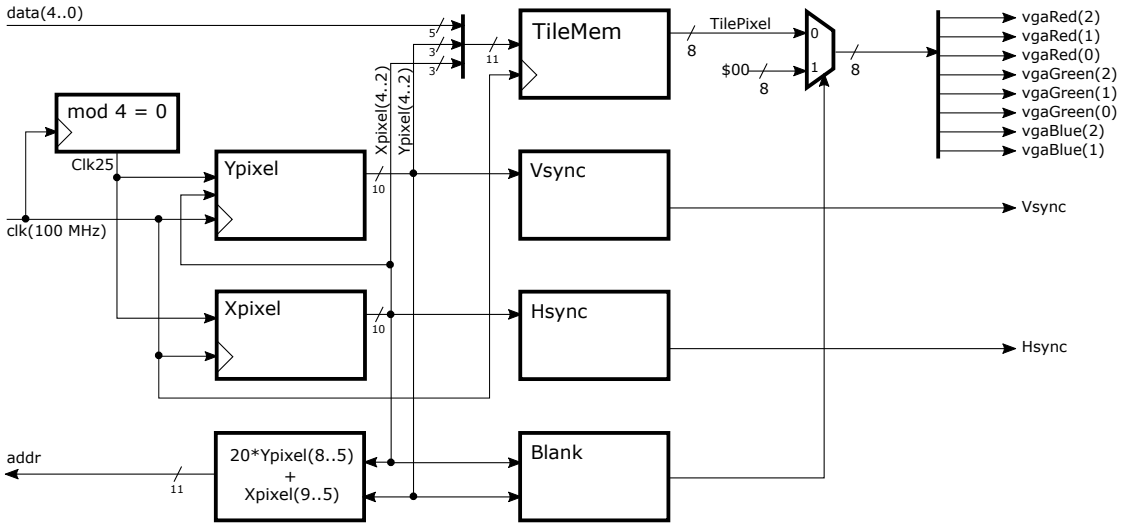
Bildminnet (PICT_MEM) består av ett 2kB stort block-RAM. Då textmonitorn består av 15 rader á 20 kolumner med ett tecken i varje position så behövs endast 300 byte, men minsta allokerbara block-RAM-storlek i vår FPGA är just 2kB, så det får bli det. Block-RAM:et är ett dubbelports-minne där tangentbordsavkodaren kan skriva in ny information på en adress via ena porten samtidigt som bildgeneratorn kan läsa ur minnet på en annan (eller samma) adress via den andra porten. Just denna egenskap gör att block-RAM:et lämpar sig mycket väl som bildminne. En block-RAMs-konstruktion kan direkt skrivas i VHDL.



Figur 2.6: Bildminnet PICT_MEM

2.5 VGA_MOTOR : Bildgeneratorn

Slutligen då bildgeneratorn (VGA_MOTOR). Dess uppgift är att läsa indexvärden ur bildminnet och rita ut motsvarande teckens utseende på bildskärmen. Varje teckens utseende finns i bildgeneratorns teckenminne (TileMem). För att VGA-monitorn ska visa en korrekt bild måste synksignaler (Vsync och Hsync) samt en släcksignal (Blank) genereras. Allt i bildgeneratorn har sitt ursprung från två pixelräknare (Xpixel och Ypixel) vilka styr hela funktionen. För att nå den tänkta bildskärmsupplösningen på 640x480 pixlar behöver dessa räknare gå i en takt av 25 MHz. Denna frekvens genereras av en modulo-4-räknare där utsignalen ligger hög under endast en klockcykels längd av den ursprungliga 100 MHz-klockan.



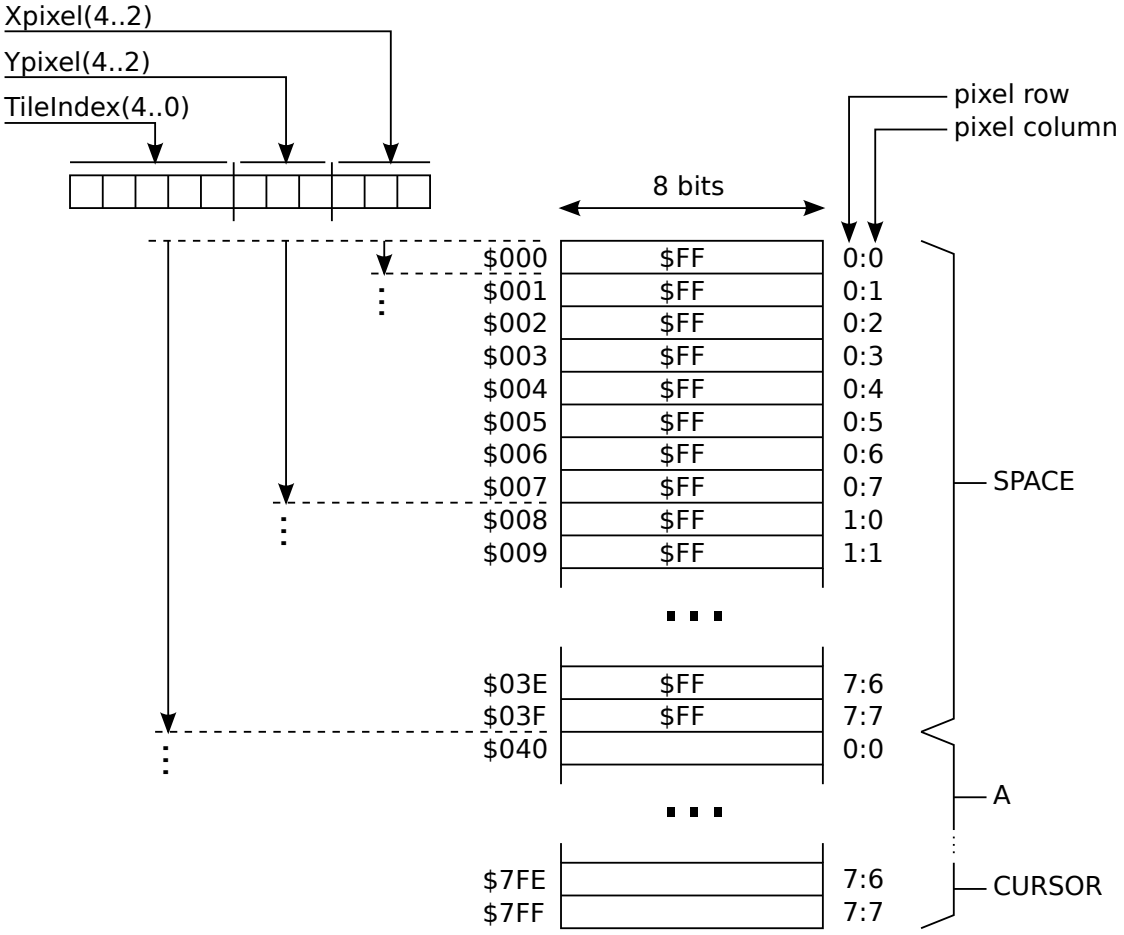
Figur 2.7: Bildgeneratorn VGA_MOTOR

2.5.1 TileMem : Teckenminnet

Teckenminnet (TileMem) ska innehålla utseendet för alla tecken som vi vill kunna visa. Det går att göra detta på många tänkbara sätt beroende på vad man vill åstadkomma. Vi vill att varje pixel i ett tecken ska kunna anta alla tillgängliga färger, dvs varje pixel kommer att motsvaras av ett 8-bitars-värde, där $FF är helt vitt och $00 är helt svart. Värden däremellan kommer att ge olika färger beroende på dom ingående bitarna för dom röda, gröna och blåa komponenterna i 8-bitars-ordet.

Teckeminnet organiseras alltså på så sätt att varje adress innehar färgvärdet för en pixel. Vi vill också att varje tecken har en storlek av 8x8 pixlar. Det medför att varje tecken upptar totalt 64 byte. De 64 första byten av teckenminnet innehåller alla värdet $FF och motsvarar alltså utseendet för SPACE (mellanslag). Därefter följer utseendet för alfabetets olika tecken A, B, C och så vidare till Å, Ä och Ö. Slutligen på dom 64 sista byten i minnet finns utseendet för själva cursorn.

Adresseringen av teckenminnet är organiserat så att varje pixel faktiskt läses inte mindre än 16 gånger, men vid olika tillfällen. Via genomsvepningen av bildskärmens upplösning på 640x480 pixlar kommer utvalda bitar från TileIndex, Ypixel och Xpixel att göra så att varje pixel i ett tecken motsvaras av 4x4 pixlar i bildskärmens upplösning, en s k storpixel. Det är på så sätt vi uppnår att textmonitorn består av 15 rader á 20 kolumner med ett tecken i varje position. Teckenminnet organiseras enligt figur 2.8.



Figur 2.8: Teckenminnet TileMem

3 Uppgifter

Hela hårdvarukonstruktionen är given enligt föregående avsnitt, Blockschemakonstruktion. Stora delar av VHDL-koden är också given. Koden finns att tanka ned från kursens hemsida för den här labben. Din uppgift är att fylla i VHDL-kod där funktionalitet saknas. Detta görs lämpligen i en viss ordning enligt följande avsnitt.

Tanka ned lab-skelettet `lab_vga.tgz` och packa upp med kommandot:
`tar xzvf lab_vga.tgz`

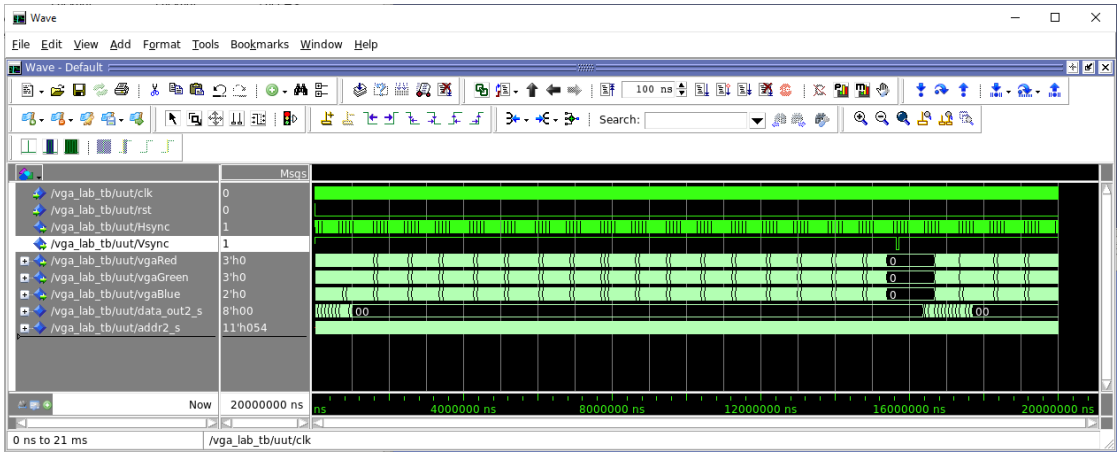
Ladda även modulen TSEA83 med kommandot:
`module add courses/TSEA83`

3.1 Uppgift 1 : Skapa bild

Gå till underkatalogen VGA_MOTOR och gör hela uppgift 1 därifrån.
Bildgeneratorn (VGA_MOTOR) saknar viss funktionalitet för att ge korrekta synksignaler. Komplettera koden i VGA_MOTOR.vhd, dvs blocken Xpixel, Ypixel, Vsync, Hsync och Blank.

Simulera VGA_MOTOR med kommandot:
`make VGA_lab.sim`
Om du kör simuleringen i 20 ms bör du få ett resultat enligt figur 3.1. Kontrollera att det kommer en `vsync` och massor med `hsync`. Det går även att mäta och kontrollera avstånd mellan och längd på `hsync`-pulserna. Simulerar du i ytterligare 20 ms kan du även kontrollera avståndet mellan `vsync`-pulserna.
Om simuleringen inte alls liknar detta kan det vara aktuellt att lägga till signalerna för `Xpixel-` och `Ypixel`-räknarna i undermodulen `U2` (vga_motorn), och kontrollera att dessa räknar som dom ska.

Syntetisera sedan till en bit-fil med kommandot:
`make VGA_lab.bitgen`
Ladda in bit-filen i Nexys3-kortet med kommandot:
`make VGA_lab.prog`
Om du gjort allt rätt ska du se en vit bakgrund med en pacman-liknande markör uppe i vänstra hörnet.



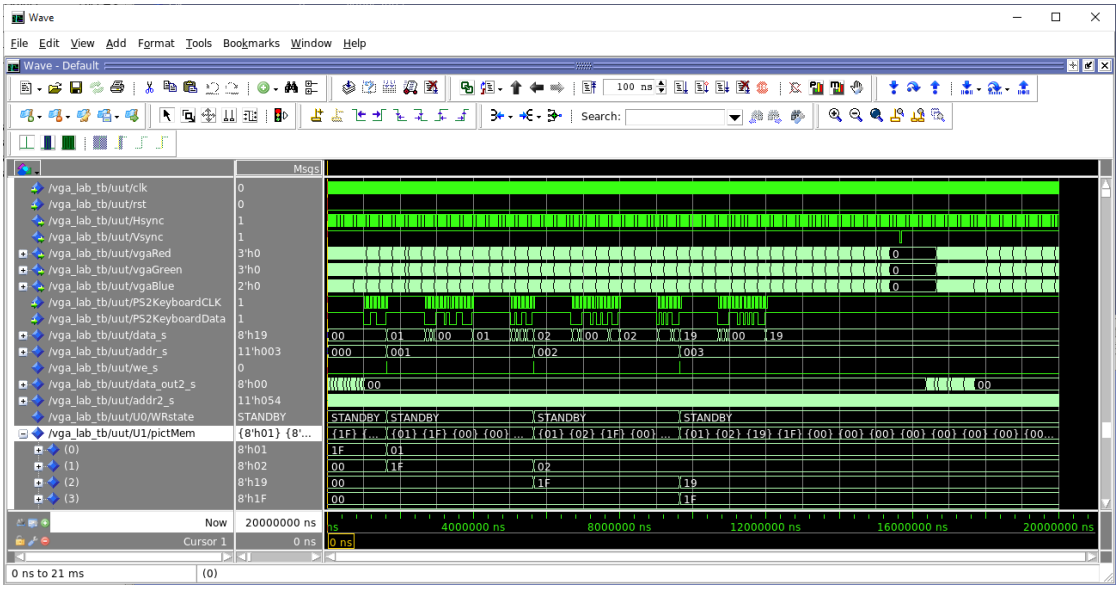
Figur 3.1: Simulering av VGA_MOTOR

3.2 Uppgift 2 : Läs av tangentbordet

Gå nu upp en nivå till katalogen VGA_lab och gör hela uppgift 2 därifrån.
Tangentbordsavkodaren (KBD_ENC) saknar viss funktionalitet för att kunna skriva in data i bildminnet (PICT_MEM). Komplettera koden för blocken PS2_bit_Counter, PS2_data_shift_reg och PS2_State i filen KBD_ENC.vhd.

Simulera VGA_lab med kommandot:
`make VGA_lab.sim`
Om du kör simuleringen i 20 ms, och lägger till signalerna `WRstate` från undermodulen `U0` och signalen `pictMem` från undermodulen `U1`, bör du få ett resultat enligt figur 3.2. Singalerna `PS2KeyboardClk` och `PS2KeyboardData` simulerar `Make` och `Break` för tangenttryckningar av `A`, `B` och `Y` i den ordningen. Om man zoomar in på signalen `WRstate` ska man kunna se hur tillståndet cyklar mellan `STANDBY`, `WRINDEX` och `WRCUR` för varje tangenttryckning. Expanderar man innehållet i signalen `pictMem` ska man kunna se hur index `01`, `02` och `19` (för tecknen `A`, `B` respektive `Y`) skrivs in i bildminnet. Index `1F` är index för tecknet som motsvarar pacman-figuren (dvs cursorn).

Syntetisera sedan till en bit-fil med kommandot:
`make VGA_lab.bitgen`
Observera! Innan du laddar in bit-filen kan du behöva slå av och på spänningen till Nexys3-kortet för att det ska fungera.
Ladda in bit-filen i Nexys3-kortet med kommandot:
`make VGA_lab.prog`
Om du gjort rätt ska du kunna skriva text (dock bara bokstäverna A, B, C, X, Y, Z, Å, Ä, Ö samt SPACE) i textmonitorn via det externa tangentbordet inkopplat till Nexys3-kortet.



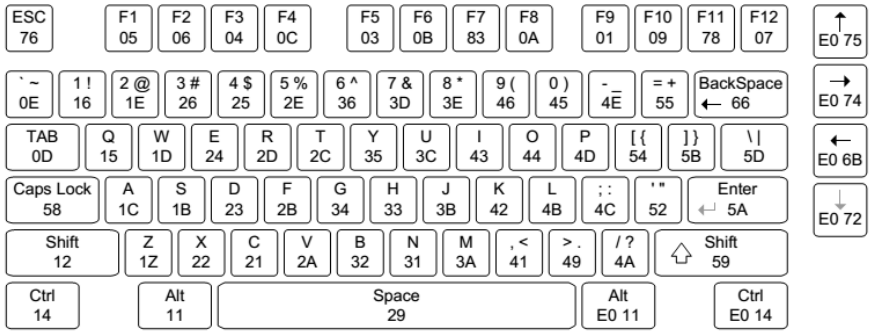
Figur 3.2: Simulering av VGA_lab

3.3 Extrauppgift : Skala om

Gör extrauppgiften i mån av tid. Du behöver inte göra den för att vara godkänd på labben.
Textmonitorn är gjord för att visa 15 rader á 20 kolumner. Skala om visningen till det dubbla, dvs 30 rader á 40 kolumner genom att ändra i VHDL-koden på lämpliga ställen. Komplettera även teckenminnet (TileMem) med utseende på odefinierade bokstäver så att du t ex kan skriva ditt namn.

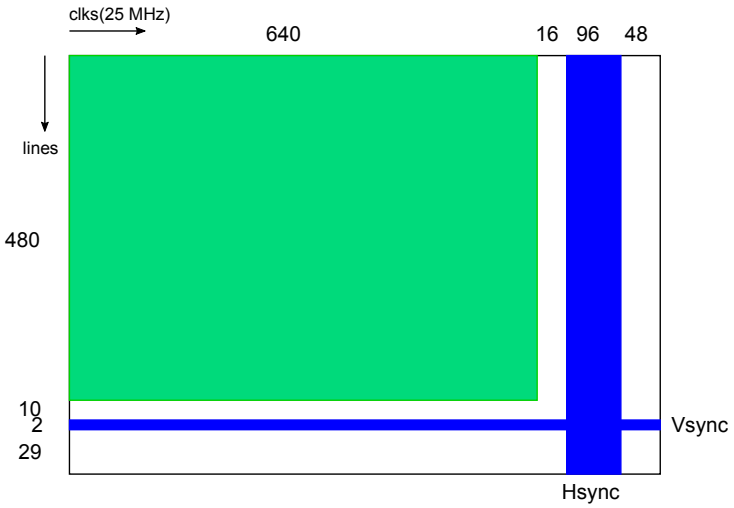
4 Bilagor

4.1 PS/2 scan codes



PS/2 Keyboard Scan Codes

4.2 VGA timing



Figur 4.1: VGA timing. Vsync och Hsync är aktivt låga.

Utöver den synliga delen av bilden (640x480 pixlar grön yta) behöver synksignaler (Vsync och Hsync, blå yta) skapas under vissa tider för att VGA-monitorn ska kunna visa bilden. Utanför den synliga delen av bilden (dvs blå och vit yta) måste videosignalen vara svart, dvs s k blanking ska aktiveras där.

5 Versionslog

2016 version 1.0	Ursprunglig version
2017 version 1.1	Diverse omritade figurer
2020 version 1.2	Figur 2.7, VGA_MOTOR, förtydligande av CLK-signal
2021 version 1.3	Simuleringsinformation med figurer och beskrivning för uppgift 1 och 2.