Spelkonsol: Meteorstorm

Lars Bengtsson
lars.bengtsson@physics.gu.se

2014-12-12

Sammanfattning: En spelprototyp har konstruerats för en VGA-skärm. Spelet heter MeteorStorm och går ut på att skydda ett antal städer under en meteorsvärm. En ”skyddsskärm” styrs med två knappar på DE2-kortet och på VGA-skärmen registreras antalet meteoriter som återstår och hur många meteoriter som träffat någon stad. Skyddsskärmen blir allt mindre för varje meteorit som fångas vilket gör det allt svårare att skydda städerna. Ett antal drivrutiner har utvecklats levereras i två moduler och inkluderats i ”board support packagen”. Till VGA-modulen finns en detaljerad beskrivning av varje drivrutin. Dessa har också verifierats, validerats och analyserats i detalj. Koden har optimerats med avseende på både storlek och snabbhet.

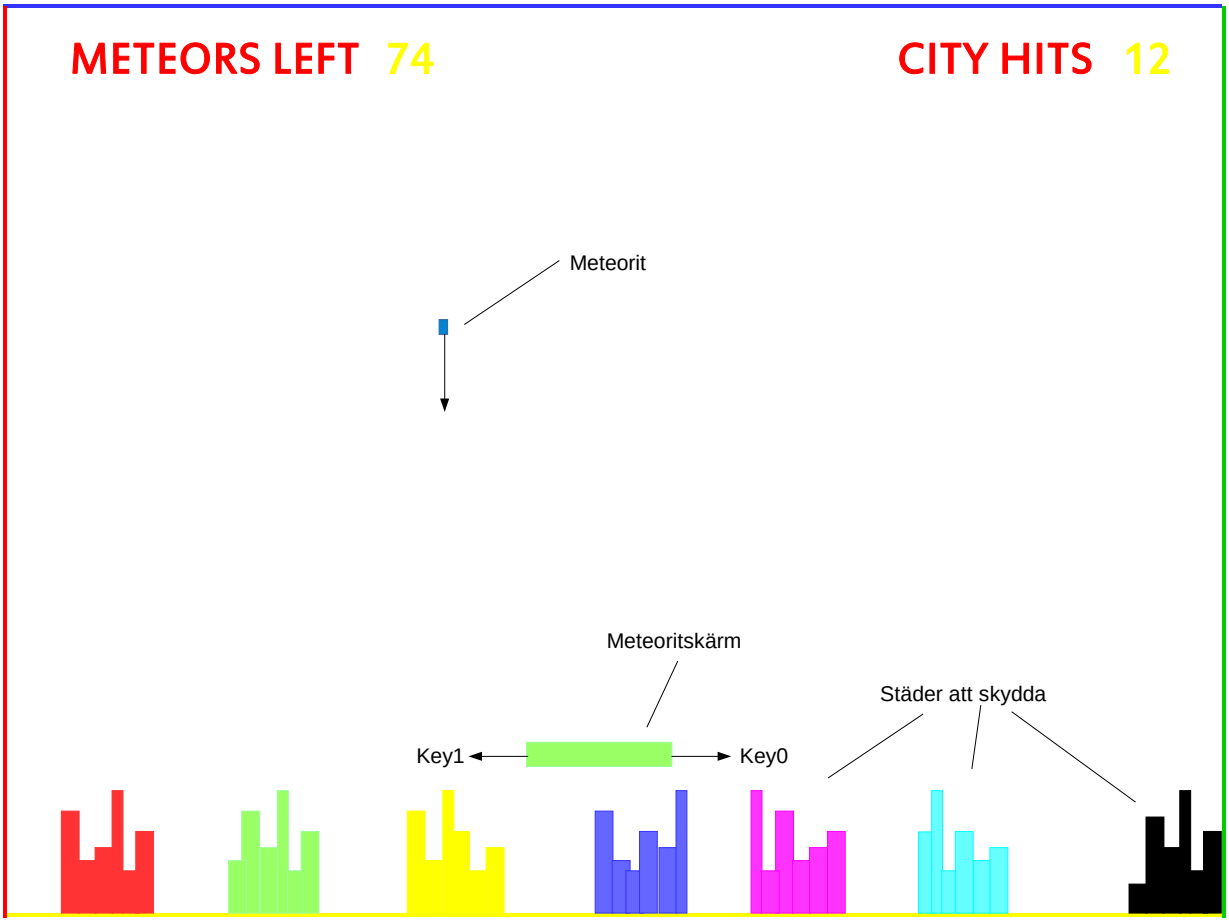
Innehåll

1 Inledning

I spelet Meteorstorm utsätts jorden för en meteoritsvärm. 100 stycken meteoriter är på väg mot jorden och för att skydda jordens städer har en satellitskärm konstruerats. Denna skärm kan flyttas över himlen och blockera de inkommande meteoriterna. Skärmens egenskaper försämras dock för varje meteorit som fångas, varför den bästa taktiken är att bara skydda städerna.

I spelet är det din uppgift att flytta skärmen åt höger och vänster (med Key0- respektive Key1-knappen på DE2-kortet) för att fånga upp de meteoriter som ser ut att träffa en stad. Antalet meteoriter som träffat en stad (=”City Hits”) registreras i övre högra hörnet och det gäller att minimera denna siffra. Tänk på att skärmens egenskaper försämras för varje meteorit som fångas. Därför gäller det att hushålla med skärmen och inte fånga meteoriter i onödan, dvs meteoriter som inte ser ut att träffa en stad.

Figur 1 visar en bild över spelplanen.



Figur 1 Spelplanen

Meteoriterna faller ner slumpvis från himlen med slumpvisa färger. Antalet återstående meteoriter visas i övre vänstra hörnet och antalet meteoriter som träffat en stad visas i övre högra hörnet. Samma information visas också på den alfanumeriska displayen på DE2-kortet.

Två C-moduler har konstruerats för spelet; en lcd-modul (liten) och en vga-modul (stor) med alla funktioner för att hantera grafiken på skärmen.

2 Kravspecifikation

I tabellen nedan visas en sammanställning av kundens krav samt en sammanfattning företagets kommentarer.

Tabell 1 Kundkraven

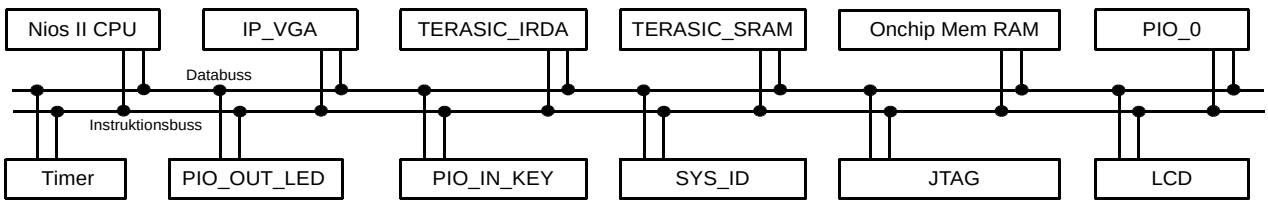
Kundkrav	**Företagets kommentarer**
Filhuvud i varje fil.	☑ Sex filer bifogas: main()-funktionen, lcd- och vga-modulerna (.h och .c) samt tcl-filen till vga-modulen. Se bilagorna.
Namn på variabler ska vara tydliga	☑ Genomgående har camelCase använts.
Kontroll av gränsvärden ska ske, t ex att värdet ligger utanför skärmen, då ska ett felmeddelande skrivas ut.	☑
Koden ska vara rikligt kommenterad.	☑
Alla drivrutiner ska finnas i BSP.	☑ Se kapitel 8.
Strukturerad kod, inte bara en applikationsfil för hela spelet.	☑ Två moduler har konstruerats (lcd och vga) med funktioner som anropas från main()-funktionen.
Spelspecifikation (spelmanual).	☑ Se bilaga E.
Testprotokoll (minst ett fall från varje krav)	☑ Se avsnitt 5.
SW-arkitekturen.	☑ Se avsnitt 4.2.
Hierarkin.	☑ Se avsnitt 4.1.
Minneskarta.	☑ Se avsnitt 3.1.
Alla I/O-enheterna och de viktiga funktionerna.	☑ Se avsnitt 3.1 och 3.2.
Tidsanalys av den färdiga koden. Hur mycket ledig tid finns i systemet?	☑ Se avsnitt 7.2 och 7.3.
Alla drivrutinerna ska finnas i komponenterna under HAL/src och inc. tcl-filerna ska definiera alla drivrutinerna för respektive komponent så att de kommer med i BSP-foldern.	☑ Se kapitel 8.
Beställda drivrutiner med kort manual för VGA-drivrutinerna. Även exekveringstiden ska vara ”defilerad” för varje funktion. Koden ska vara robust skriven.	☑ Se avsnitt 5 för manual till VGA-drivrutinerna. ☑ Se tabell 3 i avsnitt 7 för exekveringstiden. (Oklart vad kunden menar med ”defilerad” men alla funktionernas exekveringstid är uppmätta och redovisas i rapporten.)
Beskriv hur koden har optimerats för hastighet och storlek. Beskriv minnets arkitektur och motivera varför vissa delar av koden/data har länkats till olika minnen.	☑ Optimering av storlek, se avsnitt 6. ☑ För tidsoptimeringen, se avsnitt 7.2.
Spela in en kort film på Youtube som valideringsbevis på att spelet fungerar. Filmen ska börja med en kort förklaring av spelet, därefter en översiktsbild av SW-konstruktionen och till sist ett smakprov på spelet.	☑ Se Youtube: http://youtu.be/mXwvTMQqRDA
Leveransen ska bestå av tre foldrar: 1) Konstruktionsbeskrivning 2) Spel och 3) Diverse	☑

3 Hårdvaran

Hårdvaran kan delas upp i två delar; arkitekturen i FPGA:n och hela systemet. Den FPGA-arkitektur som användes levererades av kunden och beskrivs i avsnitt 3.1 och avsnitt 3.2 beskriver hela systemet, dvs hela spelkonsolen.

3.1 FPGA:n

Arkitekturen i FPGA:n framgår av figur 2 och i tabell 2 redovisas minneskartan (från system.h-filen).



Figur 2 Hårdvaran

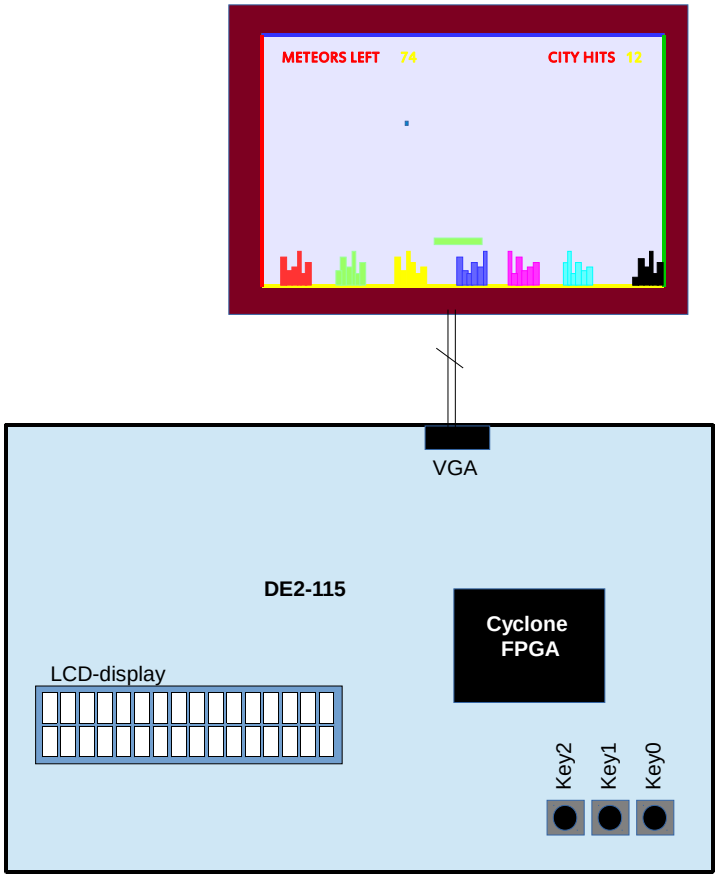
Tabell 2 Minneskartan

Enhet	Basadressens namn	Adressområde
Nios II CPU	-	-
IP_VGA	MY_VGA_BASE	0x000000 – 0x1FFFFF
TERASIC_SRAM	TERASIC_SRAM_BASE	0x200000 – 0x2FFFFF
OnChip Mem Ram	ON_CHIP_RAM_BASE	0x340000 – 0x36FFFF
PIO_0	PIO_0_BASE	0x381000 – 0x38100F
Timer	MY_TIMER_BASE	0x381010 – 0x38101F
PIO_OUT_LED	PIO_OUT_LED_BASE	0x381020 – 0x38102F
PIO_IN_KEY	PIO_IN_KEY_BASE	0x381030 – 0x38103F
SYS_ID	SYSID_BASE	0x381040 – 0x381047
JTAG	JTAG_UART_BASE	0x381048 – 0x38104F
TERASIC_IRDA	TERASIC_IRDA_REC_0_BASE	0x381050 – 0x381053
LCD	CHARACTER_LCD_BASE	0X381054 – 0X381055

Av dessa komponenter har inte PIO_0, PIO_OUT_LED och TERASIC_IRDA använts i det här projektet.

3.2 Spelkonsolen/systemet

Figur 3 visar en figur över hela systemet (med endast de delar av DE2-kortet som används).

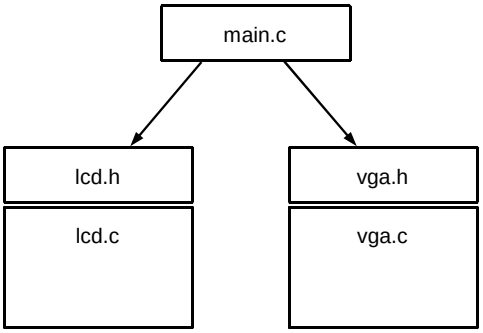


Figur 3 Hela systemet

Key0 används för att flytta skyddsskärmen åt höger och Key1 flyttar skärmen åt vänster. Key2 används för att starta ett nytt spel (SW17 resettar systemet).

4 SW-arkitekturen

Mjukvaran består av ett huvudprogram och två moduler; en lcd-modul och en vga-modul, se figur 4.



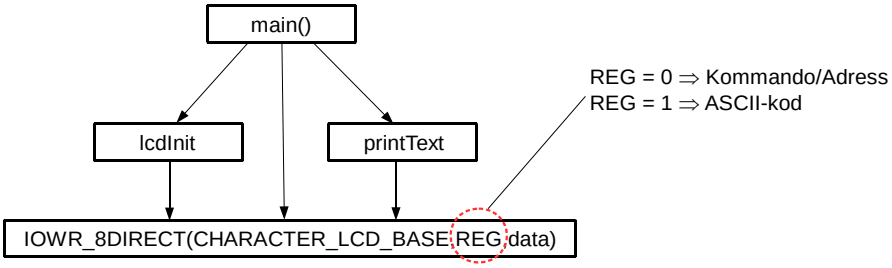
Figur 4 Programuppbyggnaden

lcd-modulen innehåller de tjänster som kommunicerar med LCD-displayen på DE2-kortet och vga-modulen innehåller tjänsterna mot VGA-skärmen. Dessa beskrivs i detalj nedan.

4.1 Hierarkin

4.1.1 LCD-modulen

lcd-modulen är mycket enkel och består endast av två funktioner; en som initierar displayen och en som skriver den ”konstanta” texten till displayen (”Meteors left:” och ”City Hits:”). Dessa funktioner anropas endast vid initieringen.

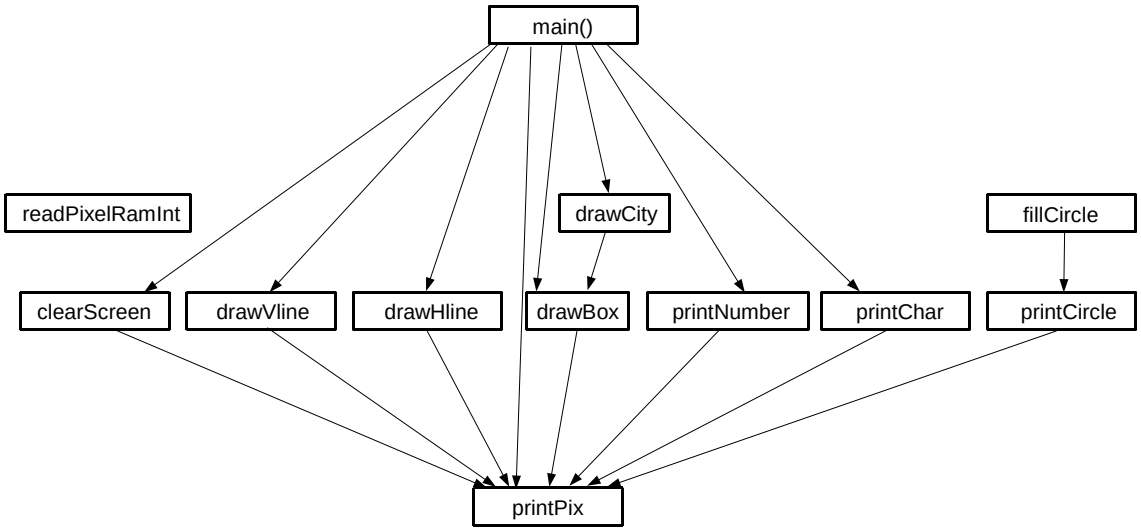


Figur 5 LCD-modulen

Eftersom kommunikationen med LCD-displayen är så okomplicerad finns ingen särskild funktion för att skriva ut data (i form av antal meteoritträffar och antal återstående meteoriter) utan detta sker genom att *main()*-funktionen kommunicerar direkt med LCD-displayen med IOWR_8DIRECT()-funktionen, se figur 5 ovan. Genom att bara ändra värdet på adress-offsetten (REG) skriver man till kommando/adress-registret eller till data-registret (ASCII-koder).

4.1.2 VGA-modulen

vga-modulen är betydligt mer komplicerad och ”call graphen” framgår av figur 6. Av figur 6 framgår först och främst att det finns tre tjänster i vga-modulen som aldrig anropas av *main()*-funktionen; *readPixelRamInt*, *fillCircle* och *printCircle*. Dessa har ändå tagits fram och inkluderats i vga-modulen enligt kravspecifikationen.



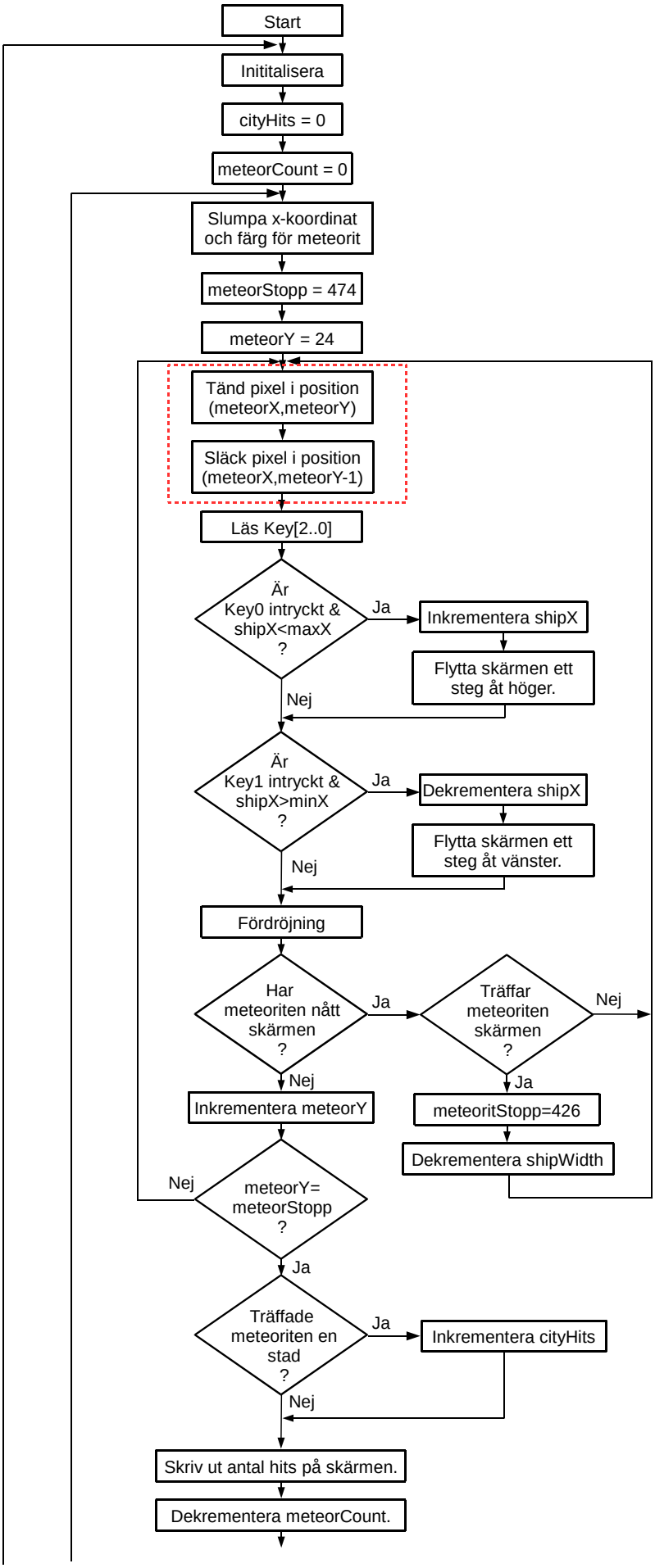
Figur 6 Call graphen för vga-modulen

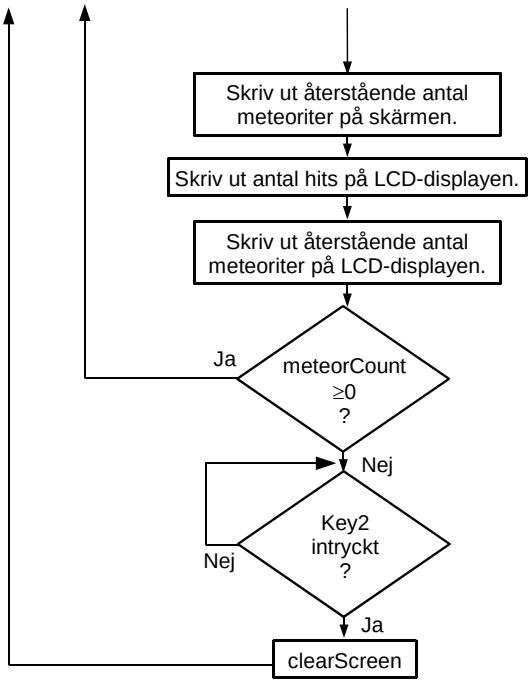
Funktionerna i vga-modulen presenteras i detalj i avsnitt 5.

4.2 Flödesschema

I figur 7 visas flödesschemat för hela programmet. För att läsa flödesschemat är följande till hjälp:

1) Betydelsen av variabelnamnen:
    a. *shipX* = den rörliga skyddsskärmens position i horisontalled (vänstra kanten)
    b. *shipWidth* = skyddsskärmens bredd (som minskas för varje meteoritträff)
    c. *cityHits* = antalet meteoriter som träffat någon stad
    d. *meteorCount* = totala antalet meteoriter som återstår (av totalt 100)
    e. *meteorX* = meteorens X-koordinat (slumpas fram)
    f. *meteorY* = meteorens Y-koordinat (inkrementeras)
    g. *meteorStopp* = den Y-koordinat som meteoriten ska stanna på (ändras beroende på om den träffar skyddsskärmen eller inte)
    h. *maxX* och *minX* = anger gränsen för höger respektive vänster bildkant
2) I första rutan, ”Initiera”, initieras alla variabler och städerna och spelplanen ritas upp.
3) De två rutorna i flödesschemat som omges av en röd streckad ram, utförs endast varannan gång (för att bromsa hastigheten på meteoriterna).
4) I flödesschemat kan det se ut som att meteoriten stannar på Y-koordinat 476 och detta kan se konstigt ut eftersom skärmen bara har 240 pixels i Y-led, men meteoritens Y-koordinat divideras med en faktor två i programmet. Anledningen är att meteoriterna annars blir för snabba och nästan omöjliga att fånga. 476/2 = 238, dvs meteoriterna stannar exakt på marken vilket innebär att det syns tydligt var de har hamnat.


Start
Inititalisera
cityHits = 0
meteorCount = 0
Slumpa x-koordinat och färg för meteorit
meteorStopp = 474
meteorY = 24
Tänd pixel i position (meteorX,meteorY)
Släck pixel i position (meteorX,meteorY-1)
Läs Key[2..0]
Är Key0 intryckt & shipX<maxX ?
Ja
Inkrementera shipX
Flytta skärmen ett steg åt höger.
Nej
Är Key1 intryckt & shipX>minX ?
Ja
Dekrementera shipX
Flytta skärmen ett steg åt vänster.
Nej
Fördröjning
Har meteoriten nått skärmen ?
Ja
Träffar meteoriten skärmen ?
Nej
Nej
Ja
Inkrementera meteorY
meteoritStopp=426
Dekrementera shipWidth
Nej
meteorY= meteorStopp ?
Ja
Träffade meteoriten en stad ?
Ja
Inkrementera cityHits
Nej
Skriv ut antal hits på skärmen.
Dekrementera meteorCount.



Figur 7 Flödesschemat

5 VGA-drivrutinerna

5.1 Inledning

I det här kapitlet beskrivs alla funktionerna i vga-modulen. För varje funktion finns en kortfattad beskrivning av dess parametrar och vad den utför. I samtliga fall har verifieringen utförts med testutskrifter på VGA-skärmen (och i konsol-fönstret i Nios). Exakt hur verifiering utförts och resultaten av detta redovisas efter varje funktionsbeskrivning. Samtliga funktioner finns i detalj i vga.c-filen i bilaga C.2.

5.2 printPix

5.2.1 Beskrivning

```
void printPix(unsigned int x,unsigned int y,unsigned char rgb);          //se bilaga C.2
```

Funktionen *printPix()* är den fundamentala funktionen som samtliga andra funktioner i vga-modulen använder sig av. Den tänder helt enkelt en pixel på skärmen. Den har tre inparametrar som samtliga är ”unsigned”:

x	:x-koordinaten, $0 \leq x \leq 319$
y	:y-koordinaten, $0 \leq y \leq 239$
rgb	:färgen, $0 \leq rgb \leq 7$

Om x- eller y-koordinaten ligger utanför skärmen skrivs ett felmeddelande ut i konsolfönstret.

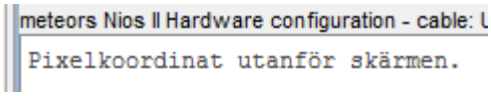
5.2.2 Verifiering

Funktionen verifierades med följande kodrader:

```
printPix(160,120,green);        //ger en grön prick mitt på skärmen
printPix(360,120,green);        //ger felmeddelande i konsolfönstret
while(1);                       //vänta
```

Testkörningen gav en grön prick mitt på skärmen (se figur 8) och ett felmeddelande i konsolfönstret (se figur 9):

Figur 8 VGA-skärmen



Figur 9 Konsolfönstret

5.3 clearScreen

5.3.1 Beskrivning

void clearScreen(void);

clearScreen()-funktionen gör hela skärmen svart genom att helt enkelt anropa *printPix()*-funktionen upprepade gånger och skriva en svart pixel till varje pixel på hela skärmen.

5.3.2 Verifiering

Funktionen verifierades med följande kodrader:

```
clearScreen();          //släck skärmen
while(1);               //vänta
```

Detta gav en svart skärm.

5.4 drawVline

5.4.1 Beskrivning

***void drawVline(unsigned int** x_pos,**unsigned int** y_start, **unsigned int** y_stopp, **unsigned char** rgb);*

drawVline() drar en vertikal linje på skärmen på X-koordinat *x_pos*, från *y_start* till *y_stopp* med färgen *rgb*. Om någon del av linjen riskerar att hamna utanför skärmen skrivs inte linjen ut alls utan ett felmeddelande skrivs ut i konsolfönstret.

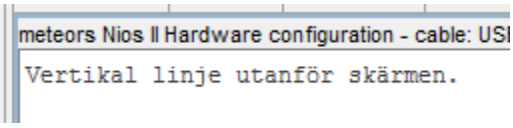
5.4.2 Verifiering

Funktionen verifierades med följande kodrader:

```
drawVline(100,3,60,green);      //grön linje från y=3 till y=60 på x=100
drawVline(100,3,260,green);     //utanför skärmen: ger felmeddelande
while(1);                       //vänta
```

Detta gav skärmen i figur 10 och konsolfönstret i figur 11:

Figur 10 VGA-skärmen



Figur 11 Konsolfönstret

5.5 drawHline

5.5.1 Beskrivning

***void* drawHline(*unsigned int* x_start,*unsigned int* x_stopp, *unsigned int* y_pos, *unsigned char* rgb);**

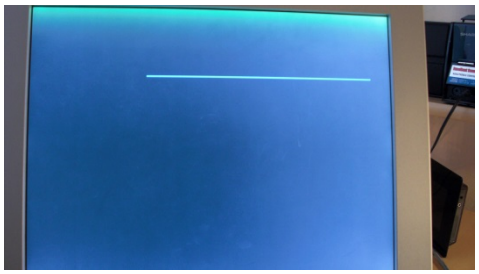
drawHline() drar en horisontell linje på skärmen på Y-koordinat *y_pos*, från *x_start* till *x_stopp* med färgen *rgb*. Om någon del av linjen riskerar att hamna utanför skärmen skrivs inte linjen ut alls utan ett felmeddelande skrivs ut i konsolfönstret.

5.5.2 Verifiering

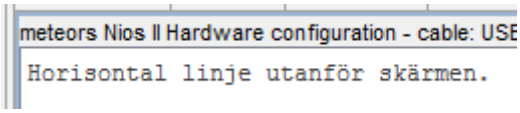
Funktionen verifierades med följande kodrader:

```
drawHline(100,300,60,blue);     //blå linje från x=100 till x=300 på y=60
drawHline(100,3,260,green);     //utanför skärmen: ger felmeddelande
while(1);                       //vänta
```

Detta gav följande skärm och konsolfönster:



Figur 12 VGA-skärmen



Figur 13 Konsolfönstret

5.6 drawBox

5.6.1 Beskrivning

***void* drawBox(*unsigned* x1, *unsigned* x2, *unsigned* y1, *unsigned* y2, *unsigned char* rgb);**

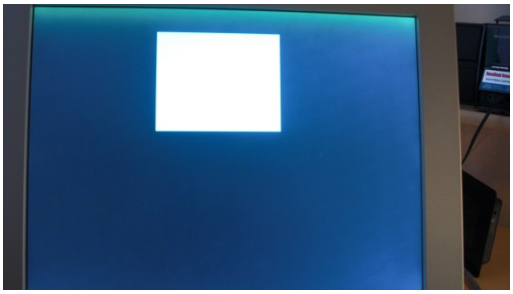
drawBox() fyller en rektangel på skärmen på med färgen *rgb*. Rektangeln går från x_1 till x_2 i horisontell led och från y_1 till y_2 i vertikal led. Om någon del av boxen riskerar att hamna utanför skärmen ritas inte boxen ut alls utan ett felmeddelande skrivs ut i konsolfönstret.

5.6.2 Verifiering

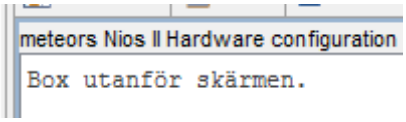
Funktionen verifierades med följande kodrader:

```
drawBox(100,200,20,100,cyan);   //cyanfärgad box: övre vänstra hörnet=(100,20), nedre högra hörnet= (200,100)
drawBox(100,350,20,100,cyan);   //utanför skärmen: ger felmeddelande
while(1);                       //vänta
```

Detta gav följande skärm och konsolfönster:



Figur 14 VGA-skärmen



Figur 15 Konsolfönstret

5.7 drawCity

5.7.1 Beskrivning

void drawCity(void);

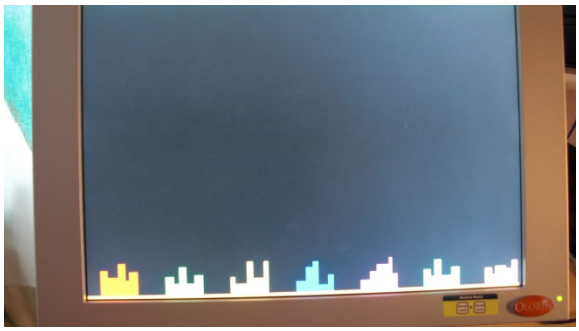
drawCity() ritar upp städernas skyline på skärmens nedersta rad genom att anropa *drawBox()* ett antal gånger. Den här funktionen har inga parametrar alls utan för att ändra skyline eller städernas storlek/antal, måste funktionen editeras direkt i vga.c-filen. Eftersom den bygger på *drawBox()*, skulle en feleditering i vga.c (dvs, en husbyggnad utanför skärmen), ge samma felmeddelande som *drawBox()*. Däremot sker ingen kontroll av att huskropparna verkligen är placerade på marken utan detta måste noga säkerställas vid ändringen av skylinen.

5.7.2 Verifiering

Funktionen verifierades med följande kodrader:

```
drawCity();        //rita upp default skyline
while(1);          //vänta
```

Detta gav följande skärm:



Figur 16 VGA-skärmen

5.8 printNumber

5.8.1 Beskrivning

void printNumber(unsigned int x,unsigned int y, unsigned char digit);

printNumber() ritar entalssiffran *digit* på skärmen där siffrans övre vänstra hörn bestäms av *x* och *y*. I den här funktionen bestäms färgen på siffran av ett #define-makro i vga.h:

```
#define c 3                //c=color, 3 ger gul färg
```

Genom att ändra siffran i define-makrot kan siffrans färg ändras. Beroende på inparametern *digit* läser funktionen *printNumber()* in ett vektorfält definierat i vga.h. Om t ex *digit* = 5, läser den in följande vektorfält till en vektorvariabel:

```
unsigned char fem[42]=
        {       c,c,c,c,c,c,
                c,c,0,0,0,0,
                c,c,c,c,c,0,
                0,0,0,0,c,c,
                0,0,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0};
```

Funktionen anropar sedan *printPix()* 42 gånger till rätt koordinater och skriver ut informationen i vektorfältet i tur och ordning på varje koordinat på skärmen. Färgen bestäms alltså av konstanten *c* som ändras i header-filen.

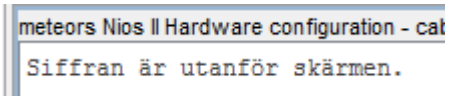
5.8.2 Verifiering

Funktionen verifierades med följande kodrader:

```
for (i=0;i<10;i++)
    printNumber(20+i*10,50,i);      //skriv ut alla siffrorna
printNumber(400,50,i):              //ger felutskrift
while(1);                           //vänta
```

Detta gav följande skärm och konsolfönster:

Figur 17 VGA-skärmen



Figur 18 Konsolfönstret

5.9 printChar

5.9.1 Beskrivning

***void printChar(unsigned int** x,**unsigned int** y, **unsigned char** bg_color,**unsigned char** color,**unsigned char** letter);*

printChar() ritar bokstaven *letter* på skärmen där bokstavens övre vänstra hörn bestäms av *x* och *y*. Giltiga bokstäver är endast versalerna A-Z och bokstaven ritas ut med färgen *color* på bakgrundsfärgen *bg_color*.

Beroende på inparametern *letter* läser funktionen *printChar()* in ett vektorfält definierat i vga.h. Om t ex *letter* = A, läser den in följande vektorfält till en vektorvariabel:

```
unsigned char A[42]=
            {       0,1,1,1,1,0,
                    1,1,0,0,1,1,
                    1,1,0,0,1,1,
                    1,1,0,0,1,1,
                    1,1,1,1,1,1,
                    1,1,0,0,1,1,
                    1,1,0,0,1,1};
```

Funktionen anropar sedan *printPix()* 42 gånger till rätt koordinater och skriver ut informationen i vektorfältet i tur och ordning på varje koordinat på skärmen. Om informationen i vektorfältet är en 1:a skrivs färgen *color* ut, om informationen i vektorfältet är en 0:a skrivs färgen *bg_color* ut.

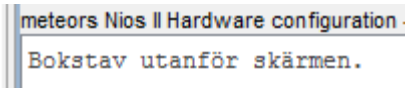
5.9.2 Verifiering

Funktionen verifierades med följande kodrader:

```
printChar(20,50,black,yellow,'T');      //gult T på svart bakgrund i koordinat 20,50
printChar(20,60,green,blue,'E');        //blått E på grön bakgrund i koordinat 20,60
printChar(20,70,cyan,red,'I');          //rött I på cyan bakgrund i koordinat 20,70
printChar(20,80,magenta,white,'S');     //vitt S på magenta bakgrund i koordinat 20,80
printChar(20,300,magenta,white,'S');    //ger felutskrift
while(1);                               //vänta
```

Detta gav följande skärm och konsolfönster:

Figur 20 VGA-skärmen



Figur 21 Konsolfönstret

5.10 printCircle

5.10.1 Beskrivning

***void printCircle(unsigned int** radie, **unsigned int** xMitt, **unsigned int** yMitt, **unsigned char** color);*

printCircle() ritar en cirkel på skärmen med centrum i koordinaten (*xMitt*,*yMitt*) med radien *radie* och färgen *color*.

Det här var det svåraste problemet att lösa på hela projektet. Att använda den ”vanliga” matematiska cirkelformeln skulle vara mycket resurskrävande eftersom det skulle innebära både kvadrering och kvadratrot vilket skulle ta lång tid. En Internetsökning gav att det fanns flera olika alternativa datoralgoritmer för cirkelritning. Algortimen som används i den här funktionen kallas för ”midpoint circle algorithm” och är hämtad från Code-for-Funs hemsida (http://codenfunz.blogspot.se/2013/02/cc-program-for-printing-circle-using.html). Algoritmen är inte ”uppenbar” på något sätt men fungerar utmärkt och är mycket resurssnål jämfört med andra alternativ (se appendix C.2).

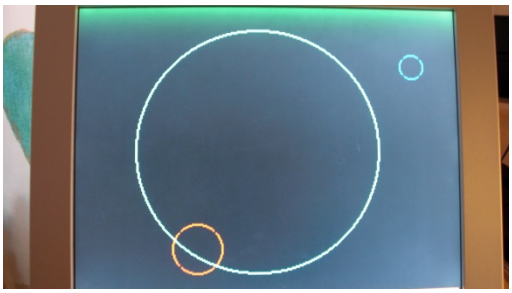
Om någon del av cirkeln riskerar att hamna utanför skärmen skrivs inte cirkeln ut utan ett felmeddelande genereras i konsolfönstret.

5.10.2 Verifiering

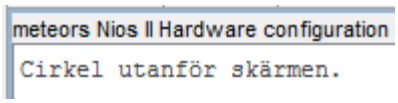
Funktionen verifierades med följande kodrader:

```
printCircle(20,100,200,red);        //röd cirkel med radien 20 med centrum i (100,200)
printCircle(100,150,120,cyan);      //cyan cirkel med radien 100 med centrum i (150,120)
printCircle(10,280,50,blue);        //blå cirkel med radien 10 med centrum i (280,50)
printCircle(100,280,50,blue);       //ger felmeddelande
while(1);                           //vänta
```

Detta gav följande skärm och konsolfönster:



Figur 22 VGA-skärmen



Figur 23 Konsolfönstret

5.11 fillCircle

5.11.1 Beskrivning

***void* fillCircle(*unsigned int* radie, *unsigned int* xMitt, *unsigned int* yMitt, *unsigned char* color);**

fillCircle() ritar en fylld cirkel på skärmen med centrum i koordinaten (*xMitt*,*yMitt*) med radien *radie* och färgen *color*.

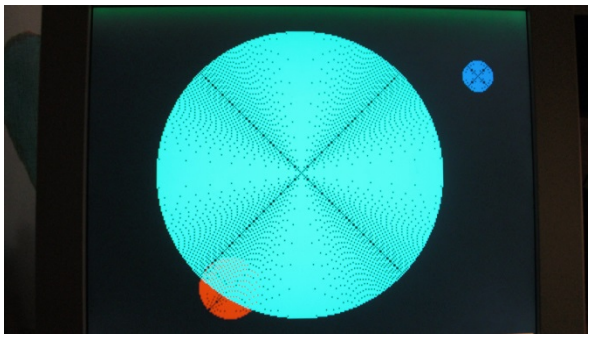
Den här funktionen anropar bara funktionen *drawCircle()* ett antal gånger med radien varierande från 0 till *radie* vilket ger en fylld cirkel.

5.11.2 Verifiering

Funktionen verifierades med följande kodrader:

```
fillCircle(20,100,200,red);          //fylld röd cirkel med radien 20 med centrum i (100,200)
fillCircle(100,150,120,cyan);        //fylld cyan cirkel med radien 100 med centrum i (150,120)
fillCircle(10,280,50,blue);          //fylld blå cirkel med radien 10 med centrum i (280,50)
while(1);                            //vänta
```

Detta gav följande skärm:



Figur 24 VGA-skärmen

Kommentarer: På grund av den algoritm som används i *drawCircle()* ger inte *fillCircle()* en helt fylld cirkel. Som framgår av figuren ovan blir det symmetriska ”hål” i cirkeln. Detta kan ordnas med en annan algoritm, men en annan algoritm skulle kräva betydligt större systemresurser och för övrigt ger ”hålen” en viss ”karaktär” åt cirkeln som ger ett mera estetiskt tilltalande intryck. Behöver kunden en helt fylld cirkel kommer detta att ingå i en kompletterande leverans utan extra kostnad (förutom kostnaden för större systemresurser).

5.11 readPixelRamInt

5.11.1 Beskrivning

***unsigned char* readPixelRamInt(*unsigned int* x, *unsigned int* y);**

Funktionen läser tillbaks pixelinformationen i koordinat (*x*,*y*) från pixelminnet.

5.11.2 Verifiering

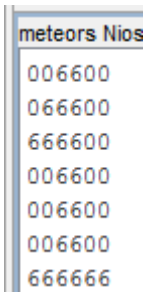
När t ex en 1:a skrivs ut på skärmen skrivs följande data in i pixel-RAM:et (på rätt adresser):

```
unsigned char ett[42]=
        {   0,0,c,c,0,0,                    //c bestämmer färgen och sätts till 6 i ett #define-makro
            0,c,c,c,0,0,
            c,c,c,c,0,0,
            0,0,c,c,0,0,
            0,0,c,c,0,0,
            0,0,c,c,0,0,
            c,c,c,c,c,c};
```

För att verifiera att *readPixelRamInt()* fungerar, skrevs en cyan-färgad 1:a (*c* = 6) till en viss position på skärmen och sedan lästes samma information ut från minnet och skrevs ut i konsolfönstret. Testkoden såg ut på följande sätt:

```
printNumber(10,0,1);
for (r=0;r<7;r++){
        for (k=0;k<6;k++)
                printf("%d",readPixelRamInt(k+10,r));
        printf("\n");
}
while(1);          //vänta
```

Detta gav följande utskrift i konsolfönstret:



Figur 25 VGA-skärmen

Med *c* = 6 överensstämmer detta exakt med innehållet i vektorfältet *ett* på föregående sida, varför funktionen *readPixelRamInt()* verifierats.

6 Kodoptimering

Ända från början användes reducerade device drivers och ”små” ANSI-C-bilbiotek utan support för C++. Detta gav en kodstorlek på 22,7 kByte (.text från objektsdumpfilen). När koden fungerade och verifierats och validerats, optimerades koden för att minimera storleken. Följande åtgärder vidtogs:

1) Kodoptimeringen sattes till maxvärdet.
2) Clean Exit ”disablades”.
3) Exit ”disablades”.
4) ”Lightweight device driver API” aktiverades.
5) *printf()* byttes ut mot *alt_printf()*.
6) Konstanter ersattes med #define-deklarationer där det var möjligt.

Alla dessa åtgärder resulterade i att kodstorleken kunde reduceras till 14,5 kByte.

> **Företagets kommentar:** Kodstorleken skulle kunna reduceras ytterligare om drivrutinerna till de hårdvarumoduler som inte används togs bort ur bsp:n. Detta gäller i första hand drivrutinera för pio_0, lysdioderna och IR-modulen. Det går att välja i BSP editorn (under ”Driver tab”) att inte ladda in dessa. Detta gav dock ingen märkbar reducering av kodstorleken.

7 Tidsanalys

7.1 Testprogrammet

Alla funktionerna i VGA-modulen har tidsanalyserats med hjälp av följande testprogram

```
TIMER_RESET;
TIMER_START;
offset=TIMER_READ;
TIMER_RESET;
TIMER_START;

//Testfunktionen här

time=TIMER_READ;
tidsbruk=time-offset;
printf("tidsbruk=%d\n",tidsbruk);
while(1);
```

Variabeln *tidsbruk* blir alltså ett mått på testfunktionens exekveringstid och denna skrivs ut i konsolfönstret.

7.2 Testresulaten

I tabell 3 nedan redovisas resultatet. Notera följande:

1) Tidsenheten i tabell 3 är ”counts”. För att få ”sekunder” multipliceras ”counts” med processorklockans periodtid (som är 1/(100 MHz) enligt kundens specifikation av hårdvaran).
2) Exekveringshastigheten för vissa av funktionerna beror av inparametrarnas värden och i dessa fall blir det redovisade tidsvärdet bara ett typiskt exempel på hur lång tid denna funktion kan ta.

Dessutom: Innan exekveringshastigheten mättes upp, optimerades exekveringshastigheten genom att en del av koden (.text och .heap) flyttades från externt RAM-minne till internt, ”on-chip” RAM. Till exempel reducerade detta exekveringstiden för funktionen *printPix()* med 40%.

Tabell 3 Exekveringstider

Funktion	Exekveringstid	Kommentar
printPix(100,200,5)	620	Oberoende av inparametrarna
clearScreen();	49385240	Oberoende av inparametrarna
drawVline(318,0,237,green)	157896	Beroende av inparametrarna
drawHline(0,318,237,yellow)	228999	Beroende av inparametrarna
drawBox(130,175,210,215,green)	163550	Beroende av inparametrarna
drawCity()	1773248	Oberoende av inparametrarna
printNumber(103,3,1)	36905	Oberoende av inparametrarna
printChar(6,3,black,red,’M’)	37665	Oberoende av inparametrarna
printCircle(20,30,208,cyan)	80055	Beroende av inparametrarna
fillCircle(20,200,50,red)	860007	Beroende av inparametrarna
readPixRamInt(20,30)	448	Oberoende av inparametrarna

Kommentarer:

1) Snabbast är förstås *printPix()* och *readPixRamInt()* eftersom de bara skriver till/läser från en enda pixeladress. Teoretiskt borde de kanske ta exakt lika lång tid, men det beror på exakt hur *IOWR*-respektive *IORD*-funktionerna implementerats i assembler.

2) Längst tid tar naturligtvis *clearScreen()* eftersom denna funktion skriver till exakt varje pixel på skärmen, men notera att med en systemklocka på 100 MHz, tar det ändå bara ca en halv sekund att ”cleara” skärmen.

7.3 Belastningsgraden

Därefter analyserades var programmet spenderar tiden, dvs i vilka funktioner programmet uppehåller sig. GNU profiler-verktyget implementerades[1], men den outputfil (gmon.out) som skulle presentera resultaten genererades inte (troligen på grund problem med *exit()*-funktionen (?)). Detta spelar mindre roll eftersom det som GNU profilern gör är ganska enkelt att implementera själv i mjukvaran. GNU profiler räknar hur många gånger som en funktion anropas och detta kan implementeras ”manuellt” genom att inleda varje funktion med att inkrementera en global variabel och sedan skriva ut dess värde i konsolfönstret efter en programkörning. På så vis analyserades koden i det här arbetet.

Resultatet framgår av tabell 4 nedan. Mittenkolumnen anger hur många gånger som varje funktion anropades under en spelomgång (= 100 meteoriter) och i den högra kolumen har detta översatts till tid genom att multiplicera med tidsåtgången för varje funktion (från tabell 3 och en systemklocka på 100 MHz).

Tabell 4 Tidsanalys av koden

Funktion	Antal anrop	Tid (sekunder)
printPix()	1275089	7,905
clearScreen();	2	0,988
drawVline()	9982	15,761
drawHline()	4	0,00915
drawBox()	10014	16,378
drawCity()	2	0,0355
printNumber()	343	0,127
printChar()	54	0,0204
Total tid		41,2 sekunder

Notera följande: Antalet anrop av vissa funktioner under en spelomgång beror på hur aktiv spelare är med skyddsskärmen. Ju mer aktivitet desto oftare måste boxen ritas om och ju fler meteoriter som fångas med skärmen, desto fler gånger måste t ex *printNumber()* anropas för att uppdatera ”cityHits”. Siffrorna i tabell 4 är alltså bara ett typiskt exempel på hur det kan se ut.

Totala tiden som spenderades i någon av VGA-rutinerna var alltså 41,2 sekunder. Hela speltiden mättes till 2 minuter och 20 sekunder. Det innebär att 1 minut och 39,8 sekunder spenderades i *main()*-funktionen, någon delay-funktion eller i någon av lcd-modulens funktioner.

8 Pakteteringen

Enligt kravspecifikationen ska alla drivrutiner levereras i Board Support Packagen (bsp:n). Som framgår av figur 4 har två moduler utvecklats för MeteorStorm. Drivrutinerna i lcd-modulen hör till den hårdvara som i konstruktionen heter ”avalon_character_lcd_driver”. Till denna hårdvara finns redan drivrutiner med tillhörande tcl-fil. För att få in de nya drivrutinerna från lcd-modulen i figur 4, skrevs därför ingen ny tcl-fil utan den befintliga tcl-filen editerades; följande två rader lades till i den befintliga tcl-filen (som heter altera_up_avalon_character_lcd_sw.tcl):

add_sw_property c_source HAL/src/lcd.c

[1] *Profiling Nios II Systems*, Altera Corp., application note AN-391-3.0, San Jose, Juli 2011.

add_sw_property include_source HAL/inc/lcd.h

De nya drivrutinerna *lcd.c* och *lcd.h*, placerades i samma katalog som de befintliga drivrutinerna (HAL/src respektive HAL/inc).

Drivrutinera för vga-skärmen hör till den hårdvara som i konstruktionen kallas för ”ip_vga” och för att få in dessa i bsp:n krävdes en helt ny tcl-fil. Först döptes filerna om till *altera_avalon_ip_vga_regs.c* respektive *altera_avalon_ip_vga_regs.h* och placerades i katalogerna IP/IP_VGA/HAL/src respektive IP/IP_VGA/HAL/inc.

Därefter skapades en tcl-fil med namnet *ip_vga_sw.tcl*, för att få in drivrutinerna i bsp:n. Hela tcl-filen finns i appendix D.

9 Slutkommentarer

Spelets funktioner har verifierats så som beskrivs denna rapport. För validering hänvisas till en demonstrationsfilm på Youtube (http://youtu.be/mXwvTMQqRDA).

Spelet har utvärderats av en testpanel och de har varit övervägande positiva till spelet, men har också framfört följande synpunkter:

1) Det är svårt att se de röda meteoriterna.
2) Det är svårt att precisionsplacera skyddsskärmen.
3) Skärmen ”tar slut” innan meteoriterna tar slut.

Dessa synpunkter bör utvärderas ytterligare av kunden, men alla dessa synpunkter är lätta att åtgärda om så önskas. De röda meteoriterna kan helt enkelt uteslutas och antalet meteoriter kan enkelt minskas eller så kan antalet städer reduceras (eller göras mindre).

Notera också att det är ganska lätt att utveckla spelet till flera ”nivåer”. I en högre ”nivå” kan t ex hastigheten på meteoriterna ökas eller så kan två meteoriter komma samtidigt. Det går också att utveckla det till ett ”tvåmannaspel” där spelarna har varsin skyddsskärm att styra. Den ena skärmen kan ligga lite högre upp än den andra och då krävs det lite ”team work” för att skydda alla städerna.

Bilagor

A main()-funktionen

```
/*----------------- main-funktionen i Meteorstorm -----------------------------
 *
 * Företag:              TEIS AB
 * Skriven av:           Lars Bengtsson
 *
 * Skapad:               2014-12-08
 * Målkrets:             Altera Cyclone IV EBC4E115F29C7
 * Verktyg:              Quartus II (v14.0) och DE2-115-kortet
 *
 * Beskrivning:          Denna kod innehåller main()-funktionen för spelet
 *                       Meteorstorm.
 * ------------------------------------------------------------------------------
*/
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <alt_types.h>
#include <system.h>
#include <unistd.h>
#include "lcd.h"
#include "vga.h"
#include "altera_avalon_pio_regs.h"


int main()
{
  alt_u32 key_in;
  unsigned int meteorX,meteorY,shipX,shipWidth,meteorStopp;
  unsigned char meteorRGB,cityHit,directionFlag;
  signed meteorCount;

  lcdInit();
  printTextLcd();

  clearScreen();                                  //släck skärmen
  shipX=130;                                      //initiera skydddsskärmens postion
  shipWidth=45;                                   //initiera skyddsskärmens bredd
  drawVline(0,0,237,red);                         //rita en röd vänsterkant
  drawVline(318,0,237,green);                     //rita en grön högerkant
  drawHline(0,318,0,blue);                        //rita en blå överkant
  drawHline(0,318,237,yellow);                    //rita en gul nederkant
  drawBox(shipX,shipX+shipWidth,210,215,green);   //rita upp skeppet
  drawCity();                                     //rita upp "city skyline"

  printNumber(103,3,1);                           //skriv ut "100"
  printNumber(110,3,0);
  printNumber(117,3,0);
  printChar(6,3,black,red,'M');                   //Skriv ut "METEORS LEFT"
  printChar(14,3,black,red,'E');
  printChar(22,3,black,red,'T');
  printChar(30,3,black,red,'E');
  printChar(38,3,black,red,'O');
  printChar(46,3,black,red,'R');
  printChar(54,3,black,red,'S');
  printChar(70,3,black,red,'L');
  printChar(78,3,black,red,'E');
  printChar(86,3,black,red,'F');
  printChar(94,3,black,red,'T');

  printNumber(300,3,0);                           //Skriv ut "0" (startvärde för City Hits)
  printChar(208,3,black,red,'C');                 //Skriv ut "CITY HITS"
  printChar(216,3,black,red,'I');
  printChar(224,3,black,red,'T');
```

```
printChar(232,3,black,red,'Y');
printChar(248,3,black,red,'H');
printChar(256,3,black,red,'I');
printChar(264,3,black,red,'T');
printChar(272,3,black,red,'S');


while(1) {
        cityHit=0;                                          //initiera hit-räknaren
        printNumber(300,3,cityHit);
        for(meteorCount=99;meteorCount>=0;meteorCount--){   //key_in=IORD_32DIRECT(PIO_IN_KEY_BASE,0);

                meteorX=1+rand()%317;                       //generera en slump-koordinat för meteoriten
                meteorRGB=1+rand()%7;                       //generera slumpfärg
                meteorStopp=474;                            //ange var meteoriten ska stanna

                for (meteorY=24;meteorY<meteorStopp;meteorY++){
                        printPix(meteorX,meteorY/2,meteorRGB);  //tänd meteorit i ny position (div med två halverar hastigheten)
                        printPix(meteorX,meteorY/2-1,black);    //släck meteorit i gammal position

                        key_in=IORD_32DIRECT(PIO_IN_KEY_BASE,0);                //läs tryckknapparna på DE2-kortet
                        if (((key_in & 0x00000001)==0) && (shipX<(318-shipWidth))) {
                                shipX++;
                                directionFlag=0;
                                drawBox(shipX,shipX+shipWidth,210,215,green);   //flytta skärmen ett steg åt höger
                                drawVline(shipX-1,210,215,black);               //släck vänsterkanten
                        }

                        if (((key_in & 0x0000002) == 0) && (shipX>1)) {
                                shipX--;
                                directionFlag=1;
                                drawBox(shipX,shipX+shipWidth,210,215,green);   //flytta skärmen ett steg åt vänster
                                if ((shipX+shipWidth)<317)                      //förstör inte den gröna linjen i ramen
                                        drawVline(shipX+shipWidth+1,210,215,black);     //släck högerkanten
                        }
                        usleep(2000);

                        if (meteorY==420) {
                                if ((meteorX>=shipX) && meteorX<=(shipX+shipWidth)){  //träffar meteoriten skärmen?
                                        meteorRGB=black;                        //släck meteoriten
                                        meteorStopp=426;                        //stoppa meteoriten i skärmen
                                        if (directionFlag)
                                                drawVline(shipX+shipWidth,210,215,black);               /släck högerkanten
                                        else
                                                drawVline(shipX+shipWidth-1,210,215,black);             //släck högerkanten
                                        shipWidth--;

                                }
                        }
                }

                //Träffar meteoriten en stad?
                if (meteorY==474 && ((meteorX>9 && meteorX<38)||(meteorX>56 && meteorX<84)||(meteorX>103 &&
meteorX<132)||(meteorX>150 && meteorX<179)||(meteorX>197 && meteorX<226)||(meteorX>244 &&
meteorX<272)||(meteorX>291 && meteorX<318)))
                        cityHit++;
                if (cityHit<10)
                        printNumber(300,3,cityHit);             //skriv ut antalet träffar på städerna
                else {
                        printNumber(293,3,cityHit/10);          //skriv ut antalet träffar på städerna,10-talet
                        printNumber(300,3,cityHit%10);          //skriv ut antalet träffar på städerna,1-talet
                }

                if(meteorCount==99)
                        drawBox(103,109,3,10,black);            //släck 1:an i "100"
                printNumber(110,3,meteorCount/10);              //skriv ut antalet återstående meteoriter,10-tal
                printNumber(117,3,meteorCount%10);              //skriv ut antalet återstående meteoriter,1-tal
```

```
            //skriv ut på LCD-displayen:
            IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x8E);                    //Flytta till första raden (0x0E)
            IOWR_8DIRECT(CHARACTER_LCD_BASE,1,(meteorCount/10)+0x30);
            IOWR_8DIRECT(CHARACTER_LCD_BASE,1,(meteorCount%10)+0x30);
            IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0xCE);                    //Flytta till andra raden (0x4E)
            IOWR_8DIRECT(CHARACTER_LCD_BASE,1,(cityHit/10)+0x30);
            IOWR_8DIRECT(CHARACTER_LCD_BASE,1,(cityHit%10)+0x30);

        }

        while ((key_in & 0x00000004)!=0)                    //Vänta tills Key2 tycks in för omstart
            key_in=IORD_32DIRECT(PIO_IN_KEY_BASE,0);

        //Återupprepa initieringen när Key2 trycks in
        clearScreen();
        shipX=130;
        shipWidth=45;
        drawVline(0,0,237,red);
        drawVline(318,0,237,green);
        drawHline(0,318,0,blue);
        drawHline(0,318,237,yellow);
        drawBox(shipX,170,210,215,green);
        drawCity();
        printNumber(103,3,1);
        printNumber(110,3,0);
        printNumber(117,3,0);
        printChar(6,3,black,red,'M');
        printChar(14,3,black,red,'E');
        printChar(22,3,black,red,'T');
        printChar(30,3,black,red,'E');
        printChar(38,3,black,red,'O');
        printChar(46,3,black,red,'R');
        printChar(54,3,black,red,'S');
        printChar(70,3,black,red,'L');
        printChar(78,3,black,red,'E');
        printChar(86,3,black,red,'F');
        printChar(94,3,black,red,'T');

        printNumber(300,3,0);
        printChar(208,3,black,red,'C');
        printChar(216,3,black,red,'I');
        printChar(224,3,black,red,'T');
        printChar(232,3,black,red,'Y');
        printChar(248,3,black,red,'H');
        printChar(256,3,black,red,'I');
        printChar(264,3,black,red,'T');
        printChar(272,3,black,red,'S');
        usleep(1000000);

        //Börja om
    }

    return 0;
}
```

B lcd-modulen

B.1 headerfilen

```
/*----------------- h-filen till lcd-modulen i Meteorstorm -------------------------
 *
 * Företag:          TEIS AB
 * Skriven av:       Lars Bengtsson
```

```
 *
 * Skapad:            2014-12-04
 * Målkrets:          Altera Cyclone IV EBC4E115F29C7
 * Verktyg:           Quartus II (v14.0) och DE2-115-kortet
 * Filnamn:           lcd.h
 * Beskrivning:       h-filen till lcd-modulen (alfanumeriska displayen)
 * -------------------------------------------------------------------------------
 */
#ifndef LCD_H_
#define LCD_H_

void lcdInit(void);
void printTextLcd(void);

#endif /* LCD_H_ */
```

B.2 sourcefilen

```
/*----------------- .c-filen till lcd-modulen i Meteorstorm -------------------------
 *
 * Företag:           TEIS AB
 * Skriven av:        Lars Bengtsson
 *
 * Skapad:            2014-12-04
 * Målkrets:          Altera Cyclone IV EBC4E115F29C7
 * Verktyg:           Quartus II (v14.0) och DE2-115-kortet
 * Filnamn:           lcd.h
 * Beskrivning:       Här definieras funktionerna i lcd-modulen
 * -------------------------------------------------------------------------
 */
#include <stdio.h>
#include <io.h>
#include <unistd.h>
#include <system.h>

void lcdInit(void) {
        usleep(15000);                                  //vänta minst 15 ms
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x38);        //8-bit, 2 rader, 5x7 mod
        usleep(4100);                                   //vänta 4.1 ms
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x08);        //Display off
        usleep(100);                                    //vänta 100 mikrosekunder
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x0C);        //Display on
        usleep(100);
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x06);        //Increment cursor, skifta inte displayen
        usleep(100);
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x02);        //Cursorn till "home" position
        usleep(2000);                                   //Vänta 2 ms
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0x01);        //Clear display
        usleep(2000);
}

void printTextLcd(void) {
        int i;
        char message1[] = "Meteors left:";
        char message2[]= "City hits:";

        for (i=0;i<13;i++) {
                IOWR_8DIRECT(CHARACTER_LCD_BASE,1,message1[i]);          //Skriv ut "Meteors left:" på första raden
                usleep(100);
        }
        IOWR_8DIRECT(CHARACTER_LCD_BASE,0,0xC0);                        //Flytta till andra raden (0x40)
        usleep(400);
        for (i=0;i<10;i++) {
                IOWR_8DIRECT(CHARACTER_LCD_BASE,1,message2[i]);          //Skriv ut "City Hits:" på andra raden.
                usleep(100);
        }
}
```

C vga-modulen

C.1 headerfilen

```
/*----------------- main-funktionen i Meteorstorm -------------------------
 *
 * Företag:                TEIS AB
```

```
 * Skriven av:              Lars Bengtsson
 *
 * Skapad:                  2014-12-06
 * Målkrets:                Altera Cyclone IV EBC4E115F29C7
 * Verktyg:                 Quartus II (v14.0) och DE2-115-kortet
 * Filnamn:                 vga.h
 * Beskrivning:             Detta är h-filen till vga-modulen i Meteorstorm
 * ----------------------------------------------------------------------------
 */

#ifndef VGA_H_
#define VGA_H_

#define black 0
#define red 1
#define green 2
#define yellow 3
#define blue 4
#define magenta 5
#define cyan 6
#define white 7
#define c 3                     //gul färg

void clearScreen(void);
void printPix(unsigned x,unsigned y,unsigned char rgb);
void drawVline(unsigned x_pos,unsigned y_start, unsigned y_stopp, unsigned rgb);
void drawHline(unsigned x_start,unsigned x_stopp, unsigned y_pos, unsigned rgb);
void drawBox(unsigned x1, unsigned x2, unsigned y1, unsigned y2, unsigned char rgb);
void drawCity(void);
void printNumber(unsigned x,unsigned y, unsigned char digit);
void printChar(unsigned x,unsigned y, unsigned char bg_color,unsigned char color,unsigned letter);
void printCircle(unsigned int radie, unsigned int xMitt, unsigned int yMitt, unsigned char color);
void fillCircle(unsigned int radie, unsigned int xMitt, unsigned int yMitt, unsigned char color);
unsigned char readPixelRamInt(unsigned int x, unsigned int y);

//---------- Definiera alla entalsiffror ----------
unsigned char nolla[42]=
        {       0,c,c,c,c,0,
                c,c,0,0,c,c,
                c,c,0,c,c,c,
                c,c,c,c,c,c,
                c,c,c,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0};

unsigned char ett[42]=
        {       0,0,c,c,0,0,
                0,c,c,c,0,0,
                c,c,c,c,0,0,
                0,0,c,c,0,0,
                0,0,c,c,0,0,
                0,0,c,c,0,0,
                c,c,c,c,c,c};

unsigned char tva[42]=
        {       0,c,c,c,c,0,
                c,c,0,0,c,c,
                0,0,0,0,c,c,
                0,0,c,c,c,0,
                0,c,c,0,0,0,
                c,c,0,0,0,0,
                c,c,c,c,c,c};

unsigned char tre[42]=
        {       0,c,c,c,c,0,
                c,c,0,0,c,c,
                0,0,0,0,c,c,
                0,0,c,c,c,0,
                0,0,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0};

unsigned char fyra[42]=
        {       0,0,0,c,c,0,
                0,0,c,c,c,0,
                0,c,c,c,c,0,
```

```
                c,c,0,c,c,0,
                c,c,c,c,c,c,
                0,0,0,c,c,0,
                0,0,0,c,c,0};

unsigned char fem[42]=
        {       c,c,c,c,c,c,
                c,c,0,0,0,0,
                c,c,c,c,c,0,
                0,0,0,0,c,c,
                0,0,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0};

unsigned char sex[42]=
        {       0,0,c,c,c,0,
                0,c,c,0,0,0,
                c,c,0,0,0,0,
                c,c,c,c,c,0,
                c,c,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0};

unsigned char sju[42]=
        {       c,c,c,c,c,c,
                0,0,0,0,c,c,
                0,0,0,c,c,0,
                0,0,c,c,0,0,
                0,c,c,0,0,0,
                0,c,c,0,0,0,
                0,c,c,0,0,0};

unsigned char atta[42]=
        {       0,c,c,c,c,0,
                c,c,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0,
                c,c,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,0};

unsigned char nio[42]=
        {       0,c,c,c,c,0,
                c,c,0,0,c,c,
                c,c,0,0,c,c,
                0,c,c,c,c,c,
                0,0,0,0,c,c,
                0,0,0,c,c,0,
                0,c,c,c,0,0};
//-------------------------------------------------

//------ Definiera alla versaler ------------------
unsigned char A[42]=
        {       0,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,1,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1};

unsigned char B[42]=
        {       1,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,1,1,0};

unsigned char C[42]=
        {       0,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
```

```
                1,1,0,0,1,1,
                0,1,1,1,1,0};

unsigned char D[42]=
        {       1,1,1,1,0,0,
                1,1,0,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,1,1,0,
                1,1,1,1,0,0};

unsigned char E[42]=
        {       1,1,1,1,1,1,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,1,1,1,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,1,1,1,1};

unsigned char F[42]=
        {       1,1,1,1,1,1,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,1,1,1,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0};

unsigned char G[42]=
        {       0,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,0,0,
                1,1,1,1,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,1};

unsigned char H[42]=
        {       1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,1,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1};

unsigned char I[42]=
        {       0,1,1,1,1,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,1,1,1,1,0};

unsigned char J[42]=
        {       0,0,1,1,1,1,
                0,0,0,1,1,0,
                0,0,0,1,1,0,
                0,0,0,1,1,0,
                1,1,0,1,1,0,
                1,1,0,1,1,0,
                0,1,1,1,0,0};

unsigned char K[42]=
        {       1,1,0,0,1,1,
                1,1,0,1,1,0,
                1,1,1,1,0,0,
                1,1,1,0,0,0,
                1,1,1,1,0,0,
                1,1,0,1,1,0,
                1,1,0,0,1,1};

unsigned char L[42]=
```

```
        {       1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,1,1,1,1};

unsigned char M[42]=
        {       0,1,0,0,1,0,
                1,1,1,1,1,1,
                1,1,1,1,1,1,
                1,0,1,1,0,1,
                1,0,0,0,0,1,
                1,0,0,0,0,1,
                1,0,0,0,0,1};

unsigned char N[42]=
        {       1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,0,1,1,
                1,1,1,1,1,1,
                1,1,0,1,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1};

unsigned char O[42]=
        {       0,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0};

unsigned char P[42]=
        {       1,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,1,1,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0,
                1,1,0,0,0,0};

unsigned char Q[42]=
        {       0,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0,
                0,0,0,0,1,0};

unsigned char R[42]=
        {       1,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,1,1,1,0,
                1,1,1,1,0,0,
                1,1,0,1,1,0,
                1,1,0,0,1,1};

unsigned char S[42]=
        {       1,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,0,0,
                0,1,1,1,1,0,
                0,0,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0};

unsigned char T[42]=
        {       1,1,1,1,1,1,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
```

```
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0};

unsigned char U[42]=
        {       1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0};

unsigned char V[42]=
        {       1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0,
                0,0,1,1,0,0};

unsigned char X[42]=
        {       1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0,
                0,0,1,1,0,0,
                0,1,1,1,1,0,
                1,1,0,0,1,1,
                1,1,0,0,1,1};

unsigned char Y[42]=
        {       1,1,0,0,1,1,
                1,1,0,0,1,1,
                1,1,0,0,1,1,
                0,1,1,1,1,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0,
                0,0,1,1,0,0};

unsigned char Z[42]=
        {       1,1,1,1,1,1,
                0,0,0,0,1,1,
                0,0,0,1,1,0,
                0,0,1,1,0,0,
                0,1,1,0,0,0,
                1,1,0,0,0,0,
                1,1,1,1,1,1};
//----------------------------------------


#endif /* VGA_H_ */
```

C.2 sourcefilen

```
/*------------------ .c-filen i vga-modulen i Meteorstorm --------------------------
 *
 * Företag:                 TEIS AB
 * Skriven av:              Lars Bengtsson
 *
 * Skapad:                  2014-12-06
 * Målkrets:                Altera Cyclone IV EBC4E115F29C7
 * Verktyg:                 Quartus II (v14.0) och DE2-115-kortet
 * Filnamn:                 vga.c
 * Beskrivning:             Här definieras funktionerna i vga-modulen till Meteorstorm
 * ----------------------------------------------------------------------------
 */
#include <stdio.h>
#include "sys/alt_stdio.h"
#include <io.h>
#include <alt_types.h>
#include <system.h>
#include "altera_avalon_pio_regs.h"

extern unsigned char nolla[42];
extern unsigned char ett[42];
```

```
extern unsigned char tva[42];
extern unsigned char tre[42];
extern unsigned char fyra[42];
extern unsigned char fem[42];
extern unsigned char sex[42];
extern unsigned char sju[42];
extern unsigned char atta[42];
extern unsigned char nio[42];
extern unsigned char A[42];
extern unsigned char B[42];
extern unsigned char C[42];
extern unsigned char D[42];
extern unsigned char E[42];
extern unsigned char F[42];
extern unsigned char G[42];
extern unsigned char H[42];
extern unsigned char I[42];
extern unsigned char J[42];
extern unsigned char K[42];
extern unsigned char L[42];
extern unsigned char M[42];
extern unsigned char N[42];
extern unsigned char O[42];
extern unsigned char P[42];
extern unsigned char Q[42];
extern unsigned char R[42];
extern unsigned char S[42];
extern unsigned char T[42];
extern unsigned char U[42];
extern unsigned char V[42];
extern unsigned char X[42];
extern unsigned char Y[42];
extern unsigned char Z[42];

void printPix(unsigned x,unsigned y,unsigned char rgb) {
	if (x<320 && y<240)
		IOWR_ALTERA_AVALON_PIO_DATA(MY_VGA_BASE+x*4+(y+2)*1280,rgb);	//tänd pixel
	else
		alt_printf("Pixelkoordinat utanför skärmen.\n");
}

void clearScreen(void){
	unsigned int x,y;
	for (y=0;y<240;y++) {
		for (x=0;x<320;x++) {
			printPix(x,y,0);			//släck en rad
		}
	}
}

void drawVline(unsigned x_pos,unsigned y_start, unsigned y_stopp, unsigned rgb){
	unsigned int y;
	if (x_pos<320 && y_start<240 && y_stopp<240) {
		for (y=y_start;y<=y_stopp;y++) {
			printPix(x_pos,y,rgb);		//tänd alla pixlar mellan y_start och y_stopp
		}
	}
	else
		alt_printf("Vertikal linje utanför skärmen.\n");
}

void drawHline(unsigned x_start,unsigned x_stopp, unsigned y_pos, unsigned rgb){
	unsigned int x;
	if (x_start<320 && x_stopp<320 && y_pos<240) {
		for (x=x_start;x<=x_stopp;x++) {
			printPix(x,y_pos,rgb);		//tänd alla pixlar mellan x_start och x_stopp
		}
	}
	else
		alt_printf("Horisontal linje utanför skärmen.\n");
}

void drawBox(unsigned x1, unsigned x2, unsigned y1, unsigned y2, unsigned char rgb) {
	unsigned int i,j;
	if (x1<320 && x2<320 && y1<240 && y2<240) {
		for (i=x1;i<x2;i++) {
```

```
                for (j=y1;j<y2;j++)
                        printPix(i,j,rgb);              //tänd alla pixlar inom x1->x2 och y1->y2
        }
    }
    else
        alt_printf("Box utanför skärmen.\n");
}


void printNumber(unsigned x,unsigned y, unsigned char digit) {
    unsigned int i,j;
    unsigned char number[42];
    if (x<320 && y<240){
        switch (digit) {                                //välj siffra
            case 0:
                for (i=0;i<42;i++)
                    number[i]=nolla[i];
                break;

            case 1:
                for (i=0;i<42;i++)
                    number[i]=ett[i];
                break;

            case 2:
                for (i=0;i<42;i++)
                    number[i]=tva[i];
                break;

            case 3:
                for (i=0;i<42;i++)
                    number[i]=tre[i];
                break;

            case 4:
                for (i=0;i<42;i++)
                    number[i]=fyra[i];
                break;

            case 5:
                for (i=0;i<42;i++)
                    number[i]=fem[i];
                break;

            case 6:
                for (i=0;i<42;i++)
                    number[i]=sex[i];
                break;

            case 7:
                for (i=0;i<42;i++)
                    number[i]=sju[i];
                break;

            case 8:
                for (i=0;i<42;i++)
                    number[i]=atta[i];
                break;

            case 9:
                for (i=0;i<42;i++)
                    number[i]=nio[i];
                break;
        }

        for (j=0;j<7;j++) {                             //Skriv ut siffran
            for (i=0;i<6;i++) {
                printPix(x+i,y+j,number[i+6*j]);
            }
        }
    }
    else
        alt_printf("Siffran är utanför skärmen.\n");

}
```

```
void drawCity(void) {                                   //skriv ut "city skylines" = ett antal boxar av olika storlek
        drawBox(10,15,220,238,1);
        drawBox(15,23,225,238,1);
        drawBox(23,28,215,238,1);
        drawBox(28,31,227,238,1);
        drawBox(31,37,221,238,1);

        drawBox(57,63,225,238,2);
        drawBox(63,67,230,238,2);
        drawBox(67,72,218,238,2);
        drawBox(72,78,230,238,2);
        drawBox(78,84,224,238,2);

        drawBox(104,107,226,238,3);
        drawBox(107,116,231,238,3);
        drawBox(116,120,215,238,3);
        drawBox(120,128,227,238,3);
        drawBox(128,131,215,238,3);

        drawBox(151,158,230,238,4);
        drawBox(158,163,220,238,4);
        drawBox(163,167,216,238,4);
        drawBox(167,174,233,238,4);
        drawBox(174,178,226,238,4);

        drawBox(198,205,231,238,5);
        drawBox(205,210,225,238,5);
        drawBox(210,219,220,238,5);
        drawBox(219,221,215,238,5);
        drawBox(221,225,227,238,5);

        drawBox(245,250,225,238,6);
        drawBox(250,253,230,238,6);
        drawBox(253,258,218,238,6);
        drawBox(258,266,230,238,6);
        drawBox(266,272,224,238,6);

        drawBox(292,297,226,238,7);
        drawBox(297,300,231,238,7);
        drawBox(300,308,225,238,7);
        drawBox(308,313,227,238,7);
        drawBox(313,317,221,238,7);
}

void printChar(unsigned x,unsigned y, unsigned char bg_color,unsigned char color,unsigned letter) {
        unsigned int i,j;
        unsigned char bokstav[42];

        if (x<320 && y<240) {
                switch (letter) {                       //välj bokstav
                        case 'A':
                                for (i=0;i<42;i++)
                                        bokstav[i]=A[i];
                                break;

                        case 'B':
                                for (i=0;i<42;i++)
                                        bokstav[i]=B[i];
                                break;

                        case 'C':
                                for (i=0;i<42;i++)
                                        bokstav[i]=C[i];
                                break;

                        case 'D':
                                for (i=0;i<42;i++)
                                        bokstav[i]=D[i];
                                break;

                        case 'E':
                                for (i=0;i<42;i++)
                                        bokstav[i]=E[i];
                                break;

                        case 'F':
```

```
            for (i=0;i<42;i++)
                  bokstav[i]=F[i];
            break;

      case 'G':
            for (i=0;i<42;i++)
                  bokstav[i]=G[i];
            break;

      case 'H':
            for (i=0;i<42;i++)
                  bokstav[i]=H[i];
            break;

      case 'I':
            for (i=0;i<42;i++)
                  bokstav[i]=I[i];
            break;

      case 'J':
            for (i=0;i<42;i++)
                  bokstav[i]=J[i];
            break;

      case 'K':
            for (i=0;i<42;i++)
                  bokstav[i]=K[i];
            break;

      case 'L':
            for (i=0;i<42;i++)
                  bokstav[i]=L[i];
            break;

      case 'M':
            for (i=0;i<42;i++)
                  bokstav[i]=M[i];
            break;

      case 'N':
            for (i=0;i<42;i++)
                  bokstav[i]=N[i];
            break;

      case 'O':
            for (i=0;i<42;i++)
                  bokstav[i]=O[i];
            break;

      case 'P':
            for (i=0;i<42;i++)
                  bokstav[i]=P[i];
            break;

      case 'Q':
            for (i=0;i<42;i++)
                  bokstav[i]=Q[i];
            break;

      case 'R':
            for (i=0;i<42;i++)
                  bokstav[i]=R[i];
            break;

      case 'S':
            for (i=0;i<42;i++)
                  bokstav[i]=S[i];
            break;

      case 'T':
            for (i=0;i<42;i++)
                  bokstav[i]=T[i];
            break;

      case 'U':
            for (i=0;i<42;i++)
```

```
                                        bokstav[i]=U[i];
                                break;

                        case 'V':
                                for (i=0;i<42;i++)
                                        bokstav[i]=V[i];
                                break;

                        case 'X':
                                for (i=0;i<42;i++)
                                        bokstav[i]=X[i];
                                break;

                        case 'Y':
                                for (i=0;i<42;i++)
                                        bokstav[i]=Y[i];
                                break;

                        case 'Z':
                                for (i=0;i<42;i++)
                                        bokstav[i]=Z[i];
                                break;

                }

                for (j=0;j<7;j++) {                                     //skriv ut bokstav
                        for (i=0;i<6;i++) {
                                if (bokstav[i+6*j])
                                        printPix(x+i,y+j,color);
                                else
                                        printPix(x+i,y+j,bg_color);
                        }
                }
        }
        else {
                alt_printf("Bokstav utanför skärmen.\n");
        }
}


void printCircle(unsigned int radie, unsigned int xMitt, unsigned int yMitt, unsigned char color) {
        int x=0,y,p;

        if ((xMitt+radie)<320 && (xMitt-radie)>0 && (yMitt+radie)<240 && (yMitt-radie)>0) {
                y=radie;
                p=1-radie;
                while(x<y) {                                            //"midpoint circle algorithm"
                        printPix(xMitt+x,yMitt+y,color);
                        printPix(xMitt+x,yMitt-y,color);
                        printPix(xMitt-x,yMitt+y,color);
                        printPix(xMitt-x,yMitt-y,color);
                        printPix(xMitt+y,yMitt+x,color);
                        printPix(xMitt-y,yMitt+x,color);
                        printPix(xMitt+y,yMitt-x,color);
                        printPix(xMitt-y,yMitt-x,color);
                        x++;
                        if(p<0) {
                                p=p+2*x+1;
                        }
                        else {
                                y--;
                                p=p+2*(x-y)+1;
                        }
                }
        }
        else {
                alt_printf("Cirkel utanför skärmen.\n");
        }
}

void fillCircle(unsigned int radie, unsigned int xMitt, unsigned int yMitt, unsigned char color) {
        unsigned int r;
        for (r=0;r<=radie+1;r++)
                printCircle(r,xMitt,yMitt,color);               //fyller cirkeln genom att rita allt större cirklar
}
```

```
unsigned char readPixelRamInt(unsigned int x,unsigned int y) {
        unsigned char pixData;
        pixData=(IORD_ALTERA_AVALON_PIO_DATA(MY_VGA_BASE+x*4+(y+2)*1280));
        return pixData;
}
```

D Tcl-filen till vga-modulen

Nedan återges den tcl-fil som användes för att få in vga-rutinerna i bsp:n.

```
# IP_VGA_sw.tcl)
# Create a new driver
create_driver vga_driver

# Associate it with some hardware known as "ip_vga"
set_sw_property hw_class_name ip_vga

# The version of this driver
set_sw_property version 1

# This driver may be incompatible with versions of hardware less
# than specified below. Updates to hardware and device drivers
# rendering the diver incompatible with older versions of
# hardware are noted with this property assignment.
# Multiple-version compatibility was introduced in version 7.1;
# prior-versions are therefore excluded.
set_sw_property min_compatible_hw_version 1.0

# Initialize the driver in alt_sys_init() - not in this case
# set_sw_property auto_initialize true

# location in generated BSP that above sources will be copied into
set_sw_property bsp_subdirectory drivers
#
# Source file listings...
#
# C/C++ source files
add_sw_property c_source HAL/src/altera_avalon_ip_vga_regs.c

# Include files
add_sw_property include_source HAL/inc/altera_avalon_ip_vga_regs.h

# This driver supports HAL type
add_sw_property supported_bsp_type HAL

# Enf of file
```

E Spelmanual

E.1 Inledning

I spelet MeteorStorm har jorden råkat ut för en meteoritstorm. Meteoritstormen består av exakt 100 meteoriter som alla riskerar att träffa någon stad på jorden. För att skydda städerna mot meteoriterna har en sattelitskärm konstruerats som kan flyttas över himlen och fånga upp meteoriterna.

Meteoritskärmen alstrar också ett magnetfält när den rör sig som saktar ner meteoriterna vilket gör det lättare att fånga dem.

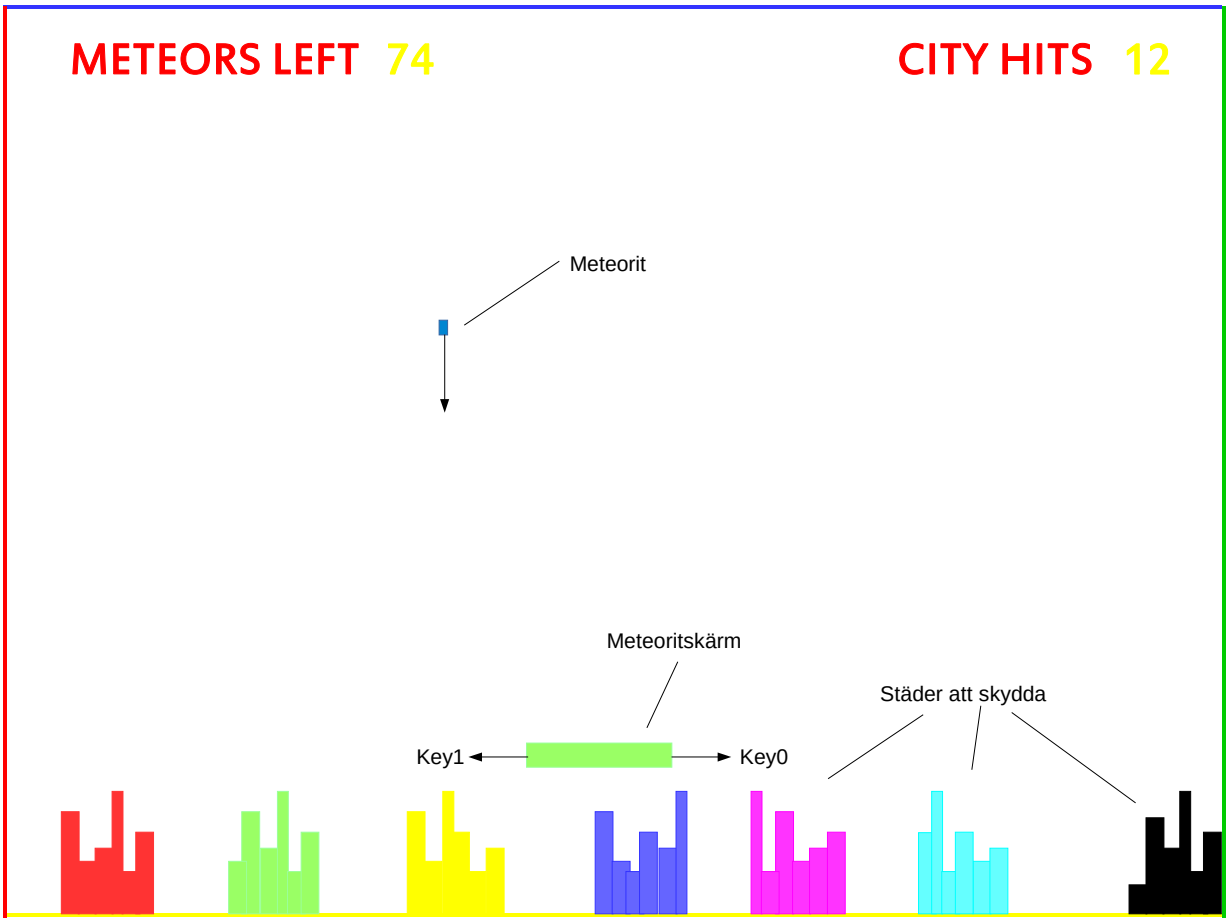
Tyvärr försämras meteoritskärmens egenskaper för varje meteorit som den träffas av; den blir mindre och magnetfältet den alstrar minskar med storleken. För varje meteorit som fångas blir det alltså svårare att fånga nästa meteorit eftersom skyddsskärmen blir mindre och dess förmåga att bromsa meteoriterna minskar.

Det innebär att det är viktigt att hushålla med skärmen, dvs fånga bara de meteoriter som ser ut att träffa någon av städerna.

För varje meteorit som slinker igenom skärmen och träffar en stad registreras en ”hit” och det gäller alltså att få så få ”hits” som möjligt. Meteoriter som missar en stad påverkar inte ”hit”-listan.

E.2 Spelplanen

Spelplanen ser ut på följande sätt:



Figur E.1 Spelplanen

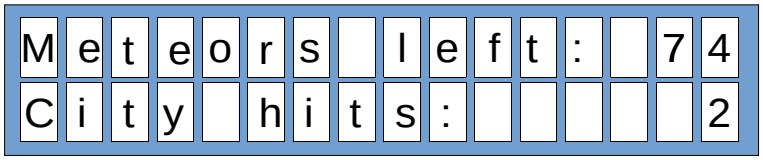
Som framgår av figur 1 finns det alltså sju städer att skydda. Meteoritskärmen flyttas åt höger med Key0 på DE2-kortet och åt vänster med Key1. En ny omgång startas med ett tryck på Key2.

Längst upp i vänstra hörnet visas hur många meteoriter som återstår och längst upp i högra hörnet visas hur många träffar som städerna råkat ut för. Det är denna siffra spelaren ska minimera.

Speltips:

1) Meteoriterna dyker upp slumpmässigt över hela skärmen med slumpvisa färger. Var beredd på att de röda är lite svårare att se än de andra.
2) Låt bli de meteoriter som inte kommer att träffa en stad! De försämrar bara skärmens egenskaper om du fångar dem. Din uppgift är att försvara städerna! Inte att fånga så många meteoriter som möjligt! De meteoriter som träffar jorden utanför städerna prioriteras bort i det här spelet.

Informationen om antalet meteoriter som återstår och antal ”hits” visas också på den alfanumeriska displayen:



Figur E.2 LCD-displayens utseende under spelets gång