DEPARTMENT OF INFORMATION TECHNOLOGY

Digitala Projekt

Redovisning av Projekt - Grupp 14

Carl Hoffstedt (c03cho@student.lth.se) & Gustaf Lund (d02gl@student.lth.se)

5/19/2007

How can you construct an embedded clock with minimal computer performance and show the result on a graphical display without any delays? This was the base question of our project, and the result was a graphical clock based on the ATmega 16 connected to a 128x64 pixels Batron display. The ATmega 16 only has 1 kb of runtime memory and 15 kb of program memory. A lot of optimizations were really the only solution to get the clock work properly. At the end we had about 1 byte of memory left!

Index

Introduktion

Den största anledning till att läsa kursen Digitala Projekt är att den knyter samman flera av de kurser man läst tidigare. Man får även praktisk träning i att lösa främmande problem utan att ha en manual vid sin sida.

Vår uppgift blev att producera en digital klocka med analog och digital visning på en display. Styrningen av displayen skedde med hjälp av en ATmega 16. För att ställa klockan använder vi oss av 3 knappar för att öka timmar, minuter och sekunder.

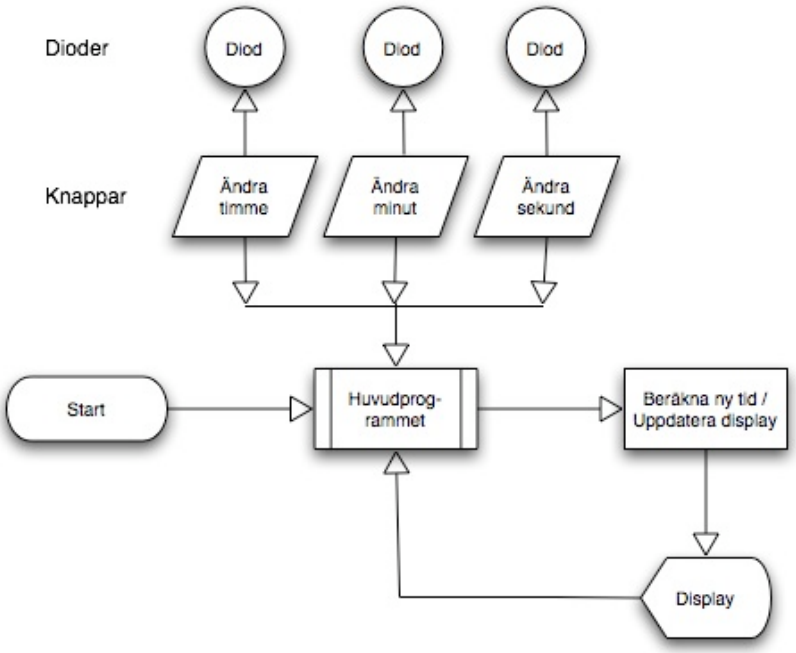
Klockan har även kompletterats med en alarmfunktion. Ställningen av larmet sker med samma knappar som klockan. En switch bestämmer om det är klockan eller larmet som ska ställas. När larmet slår till genererar en summer en pulserande ljudsignal.

Problem

Eftersom vi arbetar med en grafisk display så måste minnesmängden räcka för all grafik. Att fylla hela displayen med pixlar om man skriver en byte i taget tar ca 2-3 sekunder. Det är därför viktigt att hastigheten för att rita om de delar av displayen som vi använder uppnås.

Flödesschema

Huvudprogrammet ligger och anropar en subrutin som skriver till skärmen. Samtidigt kan avbrott för att ställa in klockan inkomma till processorn.



Resultat

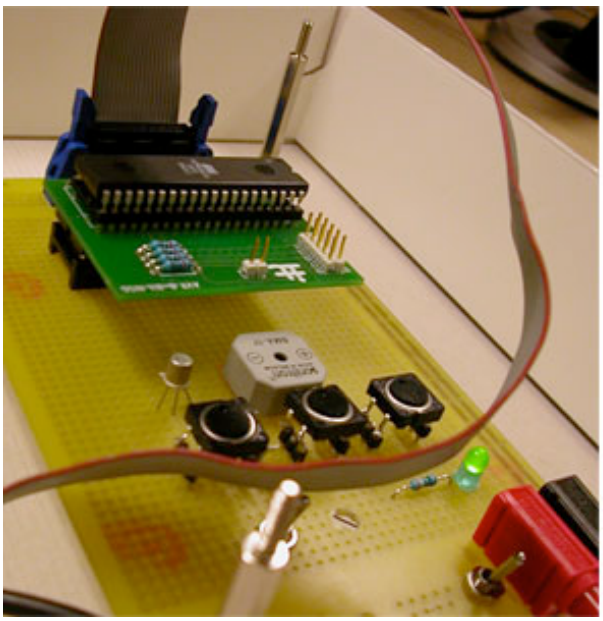
De mål som sattes upp för projektet slutfördes och de problem som beskrevs tidigare löstes. En fullt funktionerande klocka med alarm har producerats.

Hårdvara

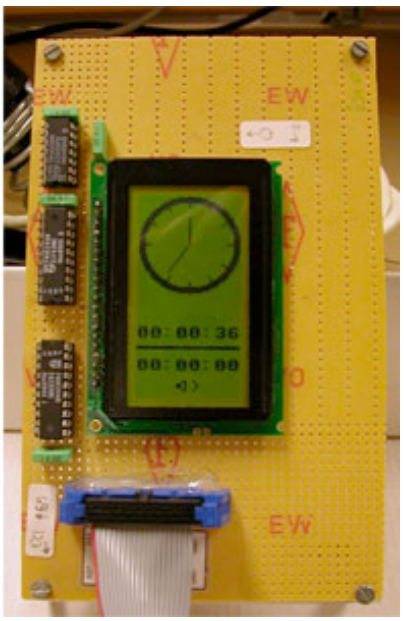
Här följer en lista med de ingående komponenterna:

- Enchipsdator ATmega16
- Batron 128*64 LCD skärm (BT128064B)
- 3 st knappar
- 1 st lysdiod
- 2 st switchar (AV/PÅ)
- 1 (piezo)summer som opererar vid 5V.
- 1 transistor för att förse summern med matarspänning
- + Lämpliga motstånd till komponenter

Kärnan i vår konstruktion består av vår ATmega16 som innehåller processor, minne, AD-omvandlare, avbrottshantering och mycket annat. Chipet har 40 ben där 32 av benen används för in och utdata. Displayen tog direkt 15 av dessa och knapparna och switchen för larmet använde sig av var sin ingång. För övrigt används några ben till att ladda upp programkod till minnet med hjälp av JTAG.

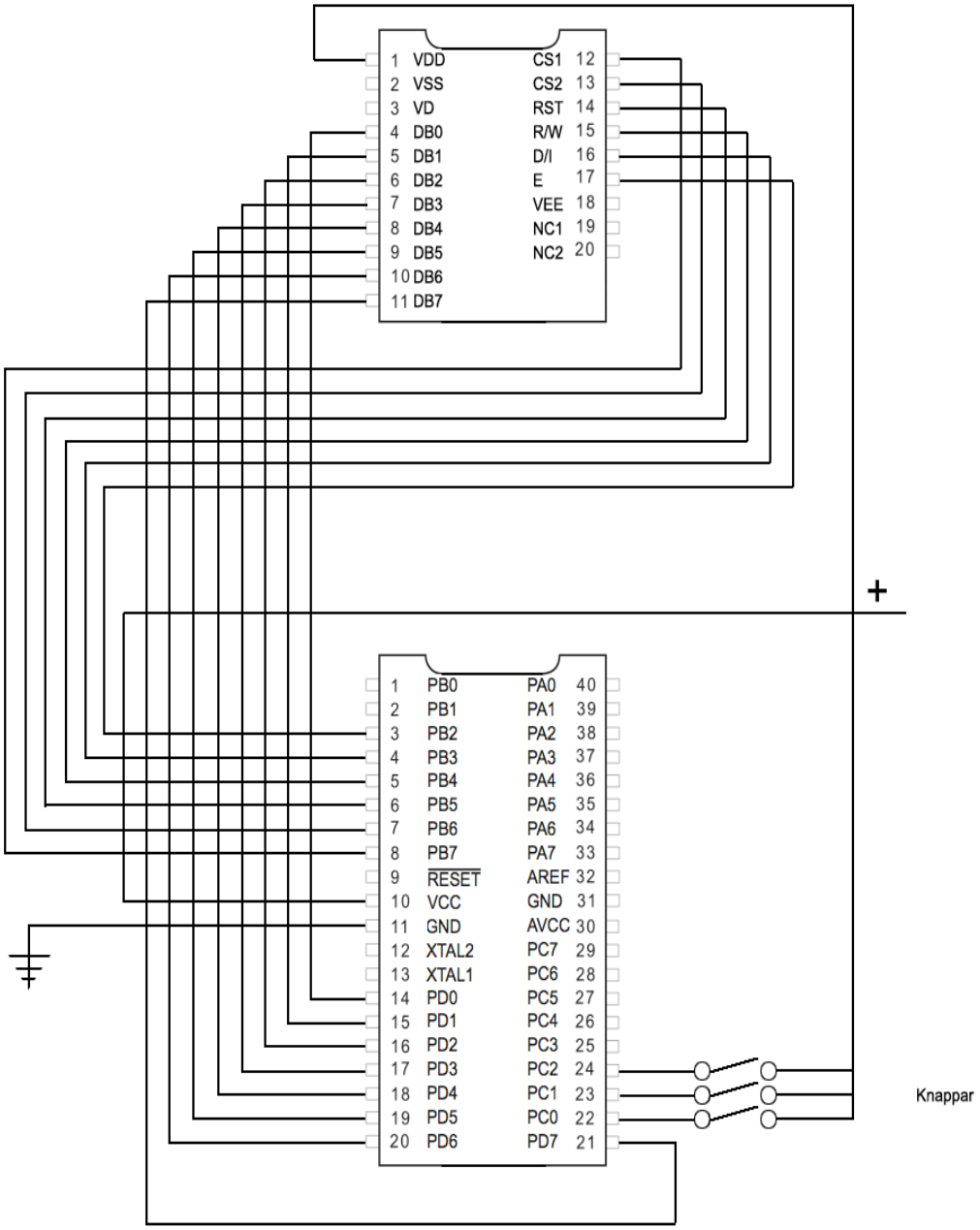


ATmega 16 med knappar och summer.



Batron display (128x64)

Kopplingsschema



Mjukvara

All programkod är skriven i C. Vi har i så stor utsträckning som möjligt försökt att använda oss av de inbyggda avbrottsrutiner som finns för att detektera knapptryckningar. Det finns även en inbyggd klocka som vi aktiverade via mjukvaran som gav ett avbrott vid ett frekvent tidsintervall.

Grafik

Den grafik vi använder oss av finns delvis lagrad i datorns programminne. I den innefattas den analoga klockan, alla siffror samt alarm-symbolen. Då all grafik tar mycket minne så har många optimeringar gjorts, dels för att lagra grafiken men också för att rita upp den. Mer om det i nästa avsnitt.

Optimeringar

Med endast 1024 bytes dataminne återfanns den största problematiken under projektets gång i just minnesoptimeringar. Mega16-processorn saknade även enheter för flyttalsberäkningar och även detta kom att uppbringa vissa problem under projektets utveckling och speciellt uppritning av den analoga klockan. I slutet av projektet utnyttjades 99% av dataminnet och redovisning för de olika typer av optimeringar följer nedan med tillhörande kodbitar.

För att rita den analoga displayen (övre halvan av skärmen) krävdes per pixel access till ritytan. Detta visade sig skapa viss problematik i programmeringen, främst då LCD-displayen endast skriver pixlar en byte åt gången (8st pixlar i x-led). Vi behövde således en buffer för att kunna mappa in enstaka pixlar. För att hålla hela skärmen i buffert krävdes 128*64 bitar = 1024 bytes. Detta var naturligtvis inte ett alternativ, då dataminnet skulle ta slut direkt.

Lösningen blev således att en buffert skapades endast för den analoga delen av displayen = 512 bytes. Vid varje pixel som skrevs till skrämen slogs de omliggande pixlarna upp i bufferten, mappades om till en byte och skrevs till displayens minne.

Kod för pixelritning av övre halvan av skärmen:

```
static byte screen_buf[8][64];
void put_pixel(int x, int y, int plot)
{
        int byteposx = (x / 8);
        int bitposx = 7 - (x - byteposx * 8);

        if (plot)
                screen_buf[byteposx][y] = (1 << bitposx) | screen_buf[byteposx][y];
        else
                screen_buf[byteposx][y] = ~(1 << bitposx) & screen_buf[byteposx][y];

        if (allow_draw)
                write_byte(byteposx, y, screen_buf[byteposx][y]);
}
```

Genom att låta den nedre delen av skärmen endast innehålla text kunde vi undvika användandet av buffer, och då med en teckenbredd på 8 pixlar.

All grafik i nedre delen sparades således i en tecken-tabell. Endast de tecken som var nödvändiga sparades i bytelistor, eg. tecken 0-9, ” ” samt 3 tecken för grafik.

För att rita upp visarna krävdes naturligtvis sin()- och cos()-funktioner och processorns nästa begränsning blev uppenbar. Trots hyffsad klockhastighet saknade processorn FPU-enhet och de trigometriska beräkningarna blev allt för prestandakrävande. Uppritningen av de tre visarna tog helt enkelt för lång tid för att ske varje sekund.

Problemet löstes genom att genera cos och sin-värden i förväg i en tabell. 60*2 flyttalsvärden krävdes för uppritning av alla nödvändiga vinklar och tillhörande funktioner:

```
static float cos_t[60] = {[cut]};
static float sin_t[60] = {[cut]};

float cos_ft(int a) {
	return cos_t[a];
}

float sin_ft(int a) {
	return sin_t[a];
}
```

Oturligt nog fanns det inte i närheten tillräckligt dataminne för dessa tabeller. Genom att utnyttja symetrin i enhetscirkeln halverade vi tabellerna. Det vill säga att vi helt enkelt inverterade värderna när vinkeln passerade x- resp y-axeln:

```
float cos_ft(int a) {
	return a >= 30 ? -1 * cos_t[a - 30] : cos_t[a];
}

float sin_ft(int a) {
	return a >= 30 ? -1 * sin_t[a - 30] : sin_t[a];
}
```

Dock räckte inte ens denna optimering. 1024 byte var alledes för lite för både trigometriska värdetabeller, nödvändiga variabler samt display-buffer. Sinus tabellen slopades helt enkelt och beräkningarna av sin-värden togs från cosinus-tabellen, fast med 45 graders justering (15 steg i tabellen). Nu hade vi minskat tabellerna från 120st flyttal till 30st:

```
static float cos_t[30] = {
	0,0.10453,0.20791,0.30902,0.40674,0.5,0.58779,0.66913,0.74314,0.80902,0.86603,
	0.91355,0.95106,0.97815,0.99452,1,0.99452,0.97815,0.95106,0.91355,0.86603,
	0.80902,0.74314,0.66913,0.58779,0.5,0.40674,0.30902,0.20791,0.10453
};

float cos_ft(int a) {
	a %= 60;
	return a >= 30 ? -1 * cos_t[a - 30] : cos_t[a];
}

float sin_ft(int a) {
	return -cos_ft(a + 15);
}
```

Slutligen kunde vi hantera ritningen av grafik, både minnesmässigt, prestandamässigt och samtidigt undvika grafikfel som ritandet av 8 simultana pixlar medförde.

Förbättringar

Utifrån våra krav finns det inte speciellt mycket förbättringar som resulterar i att vår klocka skulle uppföra sig på ett bättre sätt. Man hade dock kunnat tänka sig att ansluta ett batteri, en minneskrets samt en extern klock-krets för att klockan skulle få rätt tid då den startades om. En annan lösning hade varit att ansluta en radiomottagare till klockan som skulle förse den med en korrekt tid via DCF77 som är en signal som sänds ut från Frankfurt i Tyskland med aktuell tid.