Institutionen för Elektro- och Informationsteknik

Projektrapport inom kursen EITF11 Digitala Projekt

Projektarbete

Das Auto

Rickard Karlsson
Anders Martinsson
Niclas Bergström

2012-05-09

Abstract

This project was for the course Digital Systems, EITF11 at LTH. The purpose with this course is to give students an insight in the industrial and technological development of digital systems. Our choice of project has been to build a radio controlled car that listens to commands from an external remote control, to turn left and right and to go backwards and forwards. The car uses a IR-decoder, a ATmega16 microprocessor, a servo and a lego motor to accomplish this. The problems that we have encountered during the project has been handled as it progressed and it forced us to gain the knowledge we needed to proceed. In this report we will discuss more about the construction and the coding but also reflect over the different phases of the project. All together it has been an interesting project and it has given us an insight into how similar projects are being conducted in real life.

Innehållsförteckning

1. Inledning

Inom kursen EITF11 Digitala Projekt på LTH genomfördes under LP2 VT2012 detta projekt där vi tre studenter från Industriell Ekonomi (I09/I09/I07) undertog oss utmaningen att tillverka en fjärrstyrd bil. Syftet med kursen som ingår i teknikprofilen System och Programvaruutveckling på I-programmet är att ge studenterna en inblick i verkligt industriellt utvecklingsarbete samt realisera digitala system. Valet av att tillverka en fjärrstyrd bil ligger i ett för gruppen bakomliggande mekaniskt intresse och ett driv efter ett rörligt resultat. Målet med projektet var att inom utsatt tidsram få bilen att bli styrbar framåt, bakåt samt höger och vänster med hjälp av en fjärrtv-kontroll och ir-mottagare.

Rapporten är disponerad i enlighet med de rekommendationer som ges för E, D och F-programmet på LTH enligt följande upplägg: inledning med teori kring använd hårdvara, genomgång av metod och projektgenomförande, sammanfattande resultat, en utvärderande diskussion kring genomförandet samt ett avslutande appendix med kopplingsschema och källkod.

2. Kravspecifikation

- Bilen skall kontrolleras med en TV-fjärrkontroll
- Styrningen ska skötas med hjälp av ett servo
- Bilen ska vara byggd i LEGO dacta

3. Teori

Använd hårdvara

Lego
Byggsats i Lego av version Dacta

Motor 9V - Lego 74569
9V motor årsmodell 1990. Oväxlad med hög hastighet och lågt vridmoment.

Servo, Goteck GD-9257
Detta servo är utformat för att ge en snabbt ställhastighet utan att offra vridmomentet. Den har en kärnlös motor för högre prestanda och två kullager för minskad friktion och hastigheten vid 6 V är 0.07 sec/60° vilket är önskvärt för vårt styrservo. Servot styrs med hjälp av PWM (Pulse with Modulation) där pulsers bredd resulterar i olika ställvinklar på servot.

Processor: ATMega16
8-bitars mikrokontroller med 40 pinnar. ATMega16 har inbyggt återskrivningsbart flashminne på 16kb där program kan lagras och köras. Totalt finns 30 I/O pinnar till förfogande.

H-Brygga: "Dual full-bridge driver" - L298
H-bryggan använder logiska signaler från processorn för att stoppa eller släppa igenom den spänning som driver motorn i olika riktningar. Den har kapacitet att hantera två separata motorer, i detta projekt används två motorer som är

sammankopplade till samma utgångar på H-bryggan. De logiska pinnar på L298 som används i detta projekt är:
Enable B: Används för att tillåta eller neka aktivitet i motor B, oberoende från input 3 och 4.
Input 3 och 4: Används för att bestämma riktning på spänningen, eller stanna motorn.

IR-dekoder: Philips "Infrared remote control decoder" - SAA3049A
Tolkar information från IR-mottagaren och presenterar denna på vippar. SAA3049A har kapacitet att tolka information i protokollen RC-5 och RECS-80. I detta projekt används RC-5. SAA3049A har kapacitet att ta emot adresser vilket möjliggör styrning av flera enheter via samma mottagare och dekoder. I detta projekt används ej adresser.

De logiska pinnar på SAA3049A som används är:
Output 1-4: Siffrorna 1-9 på det senaste mottagna kommandot presenteras i binär-form på dessa.
T0: Byter läge då en signal har tagits emot.

IR-mottagare: Elrim8608
Tar emot IR-signaler och skickar vidare dessa i digital form till en IR-dekoder.

Spänningsregulator LP3852ET
Ger en spänning på 5V, vilket används till projektets logiska komponenter. Fungerar med strömkälla på mellan 2,5V till 7V. Är nödvändigt efterssom batteriet är på 6V.

Batteri 6V
Nickel-metallhydridackumulator av vanlig typ.

Fjärrkontroll Philips digital RC 5914
Standard TV-fjärrkontroll, använder protokollet RC-5.

JTAG - Joint Test Action Group
Med JTAG kan man koppla in sig till processorn för att exekvera och ladda mjukvara. Det är alltså en högnivådebugger vilket gör det lätt att hitta fel i källkoden som är skriven i C. Den kan även användas till att hitta kortslutningar eller felaktiga kretskortsdragningar. Hårdvaran som ännändes till detta var JTAGICE mkII vilken kopplades till AVR-processorn där emulering skedde i realtid. Specifikationer:

- Exekverar med målsystemets kristallfrekvens (upp till 16MHz)
- Emuleringsminne 16kb

Mjukvara

Powerlogic 5.0
Powerlogic används för att rita kopplingschema. Komponentera i projektet kunde hittas i programmets bibliotek.

AVR studie 4 med avr-gcc kompilator.

Används för att programmera och debugga koden till processorn. AVR studio är Atmel, tillverkaren av processorn, egna utvklingsmiljö. Avr-gcc möjliggör för programmering i C.

Mjukvaran skrevs i programkoden C.
När processorn får ström så körs metoden main. I denna konfigueras processorn för att:
Generera en pwm signal.

Beskrivning av bilens mjukvara:

main
I denna metod konfigureras processorn med hjälp av fyra metoder, initPorts, initInts, initPwm och determineEdge. Därefter körs en oändlig loop utan instruktioner, där programmet ligger under hela tiden bilen har spänning, med undantag för avbrott.

initPorts
Pinnar som används till output respektive input specifieras.

initPwm
Konfigurerar processorns inbyggda PWM(pulsbreddmodulering), till att skicka en regelbunden puls till servot med rätt frekvens, som vars pulsbredd styrs av OCR2.

initInts
Aktiverar interupts och definerar de pinnar som används för detta. I denna metod körs även determineEdge.

ISR(BADISR_vect)
Denna metod körs när ett oväntat avbrott sker, inga instruktioner ligger här. När denna metod är definierad töms inte samtliga variabler vid okända avbrott. Detta är önskvärt.

ISR(INT2_vect)
I denna metod utförs två funktioner. determineEdge och checkPressedButton.

delayDriveTime
En fördröjning som varar den önskade tiden som motorn kör fram eller bak, innan den stängs av igen, efter användaren tryckt på 8 eller 2.

determineEdge
När en knapp är tryckt på fjärrkontrollen, så ställer IR-dekodern om en pin som kallas T0. Med denna metod sätts kriterierna för avbrott så att nästa avbrott sker när T0 lämnar sitt nuvarande läge. Processorn skall, när T0 är 1, skapa nästa avbrott när T0 *faller* från 1 till 0, och vice versa skapa nästa avbrott då T0 *stiger* om T0 är 0.

checkPressedButton
Denna metod styr motorn och servot, beroende på vilken signal som skickats från fjärrkontrollen. En case-sats tolkar den binära talföljd som presenteras på fyra av

dekoderns outputpinnar. Knapparna på fjärrkontrollen ger följande utfall:

2: Motorn kör fram, väntar en tid, stannar.
8: Motorn kör bak, väntar en tid, stannar.
4: Servot ställer framhjulen till vänster.
5: Servot ställer framhjulen rak fram. Motorn stannar.
6: Servot ställer framhjulen till höger.
1: Funktion för att vända i trånga utrymmen.

4. Genomförande/metod

Förberedelser inför projektet

Med vår begränsade erfarenhet inom elektronik och dess tillämpning genomgick vi inför projektstart ett antal föreläsningar och labbar som snabbintroduktion till verktyg och kunskap vi kom att behöva i projektet. Denna introduktion innefattade grundläggande kunskaper i ellära, metodik för uppritning av kopplingschema samt introduktion till programmeringsspråket C.

Uppritning av kopplingsschema

Med hjälp av att noga studera de hårdvarusspecifika datablad som finns tillgängliga för alla hårdvarukomponenter vi använde oss kunde vi rita upp ett schema över de kopplingar vi ämnade att göra.

Lödning och koppling

Genom att utgå från kopplingschemat och studera de olika komponenternas specifikationer så kopplades allt samman på ett kretskort genom lödning och koppling. De sladdar som användes för högspänning var större och löddes fast. De som användes för logiska signaler var mindre, och virades fast.

Legobygge

Genom att studera ritningar samt använda kunskaper som inhämtats under barndomen byggdes en bil med tre däck. Av dessa utgjorde de två främre styrningen och det bakre drevs av två motorer med hjälp av kedjor kopplade till kugghjul på en axel (se bifogad bild). Dessa var byggda på en legoplatta där även kretskortet och servot fästes.

Påbörjan av programmering

När programmeringen påbörjades skrevs enkla program för att bekräfta att de grundläggande funktionerna, som att kunna läsa insignaler och skicka utsignaler. Logikpennan användes för att bekräfta att det vi gjorde på skärmen även skedde på riktigt.

Vidare programmering, felsökning och justering i hårdvaran.

Utan möjligheter för gruppen att jobba parallellt med bilens olika funktioner, fick vi börja i en ände. I följande ordning behandlade gruppen programmets möjlighet att:

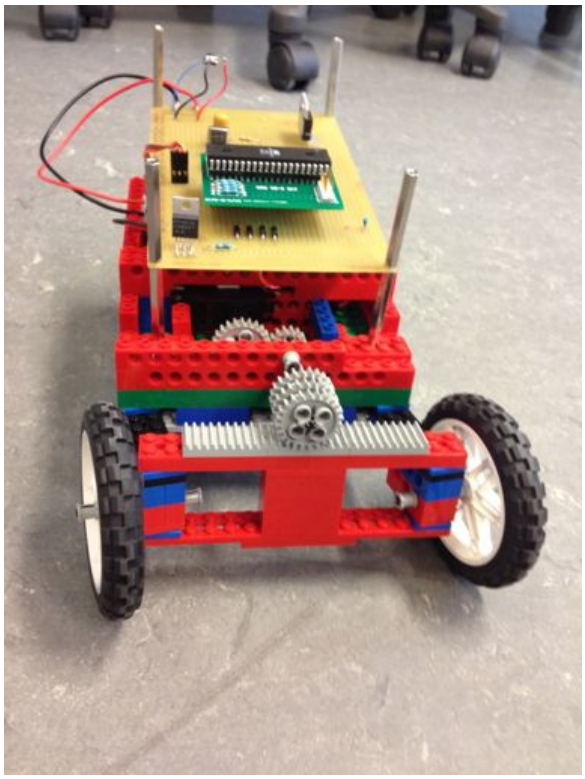
1. Läsa signaler från dekodern, och tolka dessa så att programmet kan förstå vilken knapp på fjärrkontrollen som senast har registrerats hos dekodern.

2. Styra motorn via h-bryggan, och kartlägga hur olika utsignaler påverkar motorn.
3. Använda processorns pulsbreddsmoduleringsfunktioner för att skicka lämpliga signaler och med hjälp av dessa kunna styra servot, till önskade vinklar.
4. Reagera på knapptryckningar på fjärrkontrollen, med hjälp av avbrott.
5. Använda ovanstående funktioner till att låta en användare köra bilen via fjärrkontrollen.

Detta arbete medförde en upprepande process med felsökning och åtgärder. Ofta var det de fysiska kopplingarna som inte visade sig fungera som tänkt. Även här hade vi stor hjälp av logikpenna med vilken vi, allteftersom kodning pågick, testade pinnar och signaler. Andra gånger var det okunskap om programmeringsspråket, processorn och utvecklingsmiljön som gjorde att vi inte kom vidare. Kursledaren bistod med förslag och hjälp.

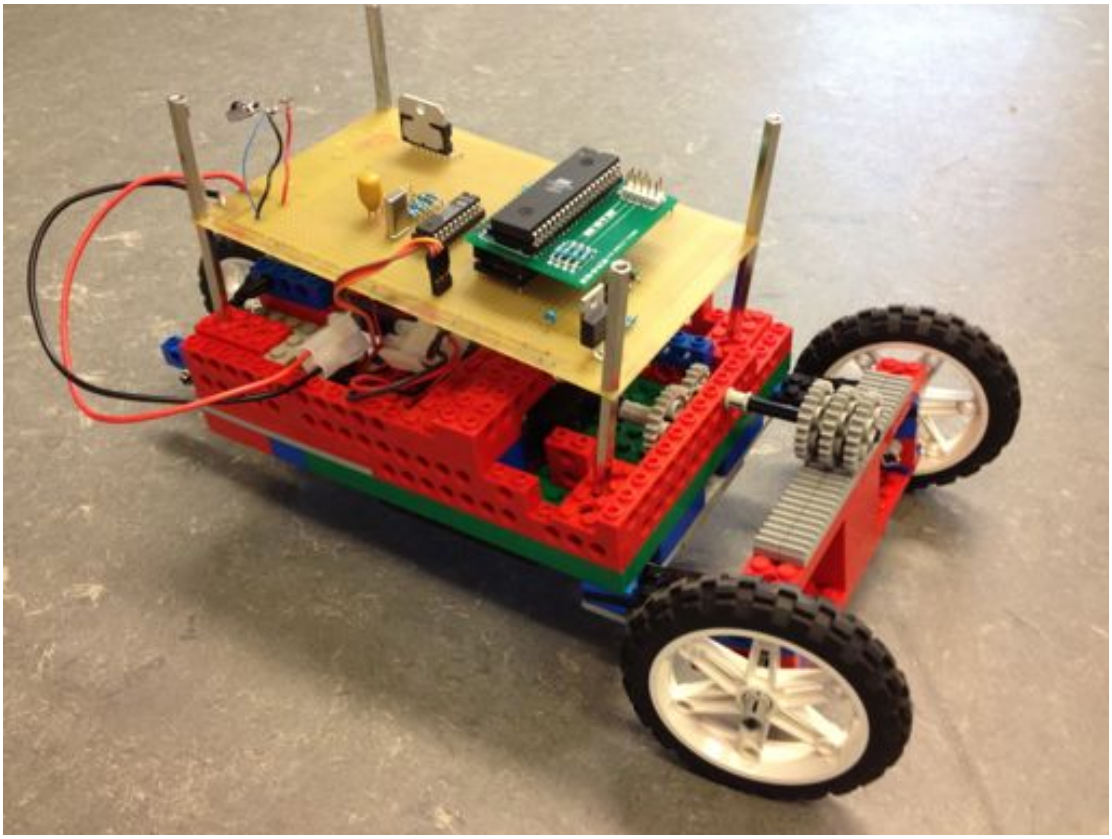
Resultat

Vi har genom detta projekt fått en djupare insikt och förståelse för hur utvecklingen av digitala produkter och projekt ser ut. Nedan följer bilder på hur vår bil till slut blev:



Figur 1. Bilen framifrån

Figur 2. Drivaxeln från motorerna



Figur 3. Bilen från sidan

5. Utvärderande diskussion

Vi fick till största grad själva bestämma vad projektet skulle bestå av. Vi valde medvetet att bygga ett projekt med tydliga mekaniska inslag. Att faktiskt kunna observera resultat, till exempel en surrande motor och ett vridande servo, har varit motiverande. Vi hade även en önskan att göra ett projekt som med en stor variation på komponterna som ingår, nu fick vi vitt skilda utmaningar som rörde IR-protokoll, pulsviddsmodulation, motorstyrning med mera.

Att vi använde lego tillät oss experimentera med utformning av chassi, styrning och den utväxling som krävdes för både servo och motor.

Eftersom systemet bara använder de fyra första av de sex pinnar på dekodern som presenterar senaste knapptryckningen, tolkar bilen 100000 och 1000011 likadant. Detta kan ge upphov till att knappar på fjärrkontrollen som ej i första hand är tänkta att kunna styra bilen, ändå kan komma att få bilen att till exempel svänga.

6. Referenslista

Manualer:

ATMega16 - Processor

SAA3049A - Infrared remote control decoder

L298 - Dual full-bridge driver

Elrim 8608 - Remote control detector

LP3855 - 1.5A Fast Response Ultra Low Dropout Linear Regulators

GD-9257 - Mini high-speed digital servo

7. Appendix

Programkod:

```
#include <avr/interrupt.h>
#include <util/delay.h>
#include <util/delay_basic.h>
#define rightOCR 5;
#define leftOCR 16;
#define midOCR 11;
#define forward 66;
#define backward 65;
#define stop 00;

volatile
int constant;
void
checkPressedButton(){

      switch (PIND & 0b00001111) {
      case (0b00001011): { //Tvåa(framåt)
            PORTC = forward;
            delayDriveTime();
            PORTC = stop;
      }
      break;
      case (0b00001110): { //Åtta(bakåt)
            PORTC = backward;
            delayDriveTime();
            PORTC = stop;
```

```
} break;
case (0b00001101): { //Fyra(vänster)
      OCR2
      = leftOCR;
      _delay_ms
      (1000);
      OCR2
      = leftOCR;

} break;

case (0b00001001): { //Sexa(höger)
      OCR2
      = rightOCR;
      _delay_ms
      (1000);
      OCR2
      = rightOCR;

} break;

case (0b00000101): { //Femma(mittåt)
      OCR2
      = midOCR;
      _delay_ms
      (1000);
      OCR2
      = midOCR;
      PORTC
      = stop;

} break;

case (0b00000111): { //Etta(special)

      OCR2
      = rightOCR; // Höger och bak
      _delay_ms
      (1000);
      PORTC
      = backward;
      _delay_ms
      (10000);
      PORTC
      = stop;

      PORTC
```

```
            = forward;
            _delay_ms
            (1000);
            OCR2
            = leftOCR; //Vänster och fram
            _delay_ms
            (1000);
            PORTC
            = forward;
            _delay_ms
            (10000);
            PORTC
            = backward;
            _delay_ms
            (1000);
            PORTC
            = stop;

      } break;

      }

      return;
}

void
determineEdge(){
      GICR
      = (0<<INT2); // Stänger av INT2 för att kunna ändra
mcucsr utan att ett interuppt skapas
      _delay_ms
      (100);

      if ((PINB & 0b0000100) == 0b0000100){ // T0 är 1
            MCUCSR
            = (0<<ISC2); // Interrupt on falling edge

      } else {
            MCUCSR
            = (1<<ISC2); // Interrupt on rising edge

      }
      GICR
      = (1<<INT2); // Aktivera interupts från INT2;
      GIFR
      = (1<<INTF2); // Clearar intf i gifr för interupts
```

```
	return;
}
void
delayDriveTime(){
	_delay_ms
	(200000);
	return;
}
void
initInt(){
	SREG
	= SREG |= 0b10000000; // I-bit i statusregistret,
aktiverar interupts från INT2;
	GICR
	= (0<<INT2); // Stänger av INT2 för att kunna ändra
mcucsr utan att ett interuppt skapas
	determineEdge
	(); // Lyssna på B
	GICR
	= (1<<INT2); // Aktivera interupts från INT2;
	GIFR
	= (1<<INTF2); // Clearar intf i gifr för interupts
	sei
	(); // Bra för interupts
	return;
}
void
initPwm(){
	TCNT2
	= 0;
	TCCR2
	|= (1<<COM21); // phase correct mode
	TCCR2
	|= (1 << CS22)|(1 << CS21) ; // set prescale to 256
	TCCR2
	|= (1<<WGM20); // phase correct pwm
	TIMSK
	|= (1<<OCIE2);
	OCR2
	= 0; // initialize counter
	TIMSK
	|= (1 << TOIE2); // enable overflow interrupt
	return;
}

void
initPorts(){
```

```
	DDRB
	= 0b00000000; // Lyssna på samtliga B
	DDRD
	= 0b10100000; // Skriv på PD5 och PD7, läs på resten av
d.
	DDRA
	= 0b00000000; // Lyssna på samtliga A
	DDRC
	= 0b11000011; // Skriv på samtliga C portar där detta är
möjligt.
	return;
}
ISR
(BADISR_vect)
{
	constant
	= OCR2; //Meningslös instruktion så att avbrottet inte
försvinner i kompileringen.
}
ISR
(INT2_vect){

	//Avbrott på INT2
	determineEdge
	();
	checkPressedButton
	();
}
void
main(void) {
	initPorts
	();
	initInt
	();
	initPwm
	();

	while (1){
		constant
		= OCR2; //Meningslös instruktion så att loopen inte
försvinner i kompileringen.

	}

	return 1;

}
```

Kopplingsschema: