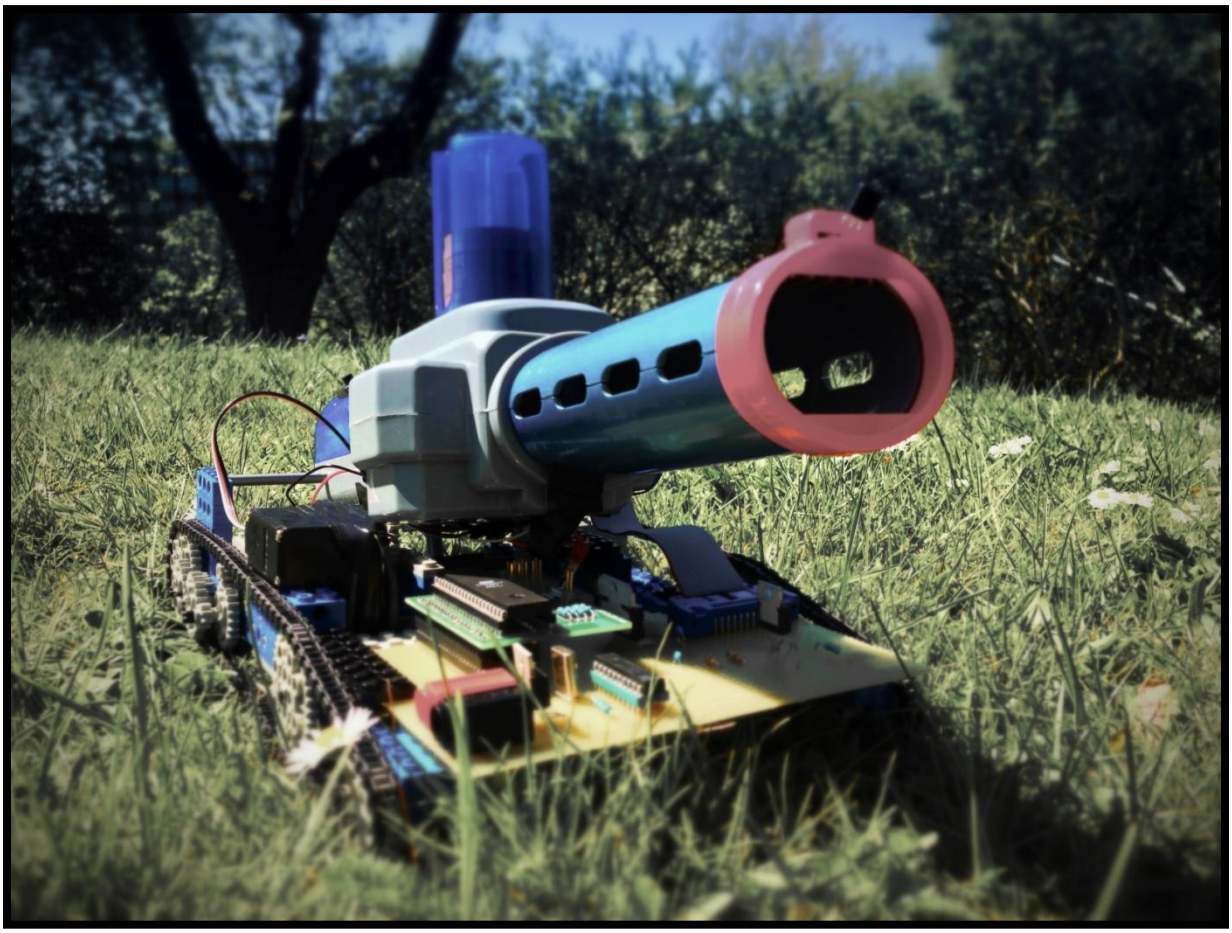
Maj-2013

EITF11: BANDKANON

Grp 05



KRISTOFER ADOLFSSON
JOHN KARLSSON
ERIK LAGERBERG
HANDLEDARE: BERTIL LINDVALL

ABSTRACT

This project was planned and completed during the course EITF11 at Lund Tekniska Högskola april-may 2013. The goal of this project was to build a Lego self-propelled gun controlled by an IR remote. We used two Lego motors, an ATMega16 microprocessor, a servo, an IR-decoder, a dual full-bridge driver and a RapidFire™ Gun to accomplish this.

INNEHÅLL

INLEDNING

BAKGRUND

Vi insåg ganska tidigt att vi ville göra något som kunde röra sig. Den positiva, triumfatoriska känslan när delar rör på sig överstiger vida känslan en konstruktör får då en skärm visar rätt information. Det var flera andra grupper tidigare år som valt att göra en bil eller något väldigt snarlikt en bil. Därför kände att vi inte bara kunde göra en sådan. Vår konstruktion skulle kunna vara kapabel till större dåd än så. Med anledning av den senare tidens debatt i media om Sveriges bristande försvarsförmåga kände vi att vi ville dra vårt strå till stacken för att trygga Sveriges självständighet. Valet att bygga en bandkanon var således tämligen självklart.

MÅLSÄTTNING

Syftet med vårt projekt är att bygga en fungerande bandkanon. Denna ska styras med en IR-länk. Funktioner som bandkanonen ska ha är:

- Köra framåt
- Köra bakåt
- Rotera åt både höger och vänster
- Skjuta

Bandkanonen ska dessutom vara tillräckligt hållbar för att kunna krocka med hinder och ta emot fiendeeld utan att försättas ur stridbart skick. Målsättningen är att vagnen ska framdrivas på larvfötter.

GENOMFÖRANDE

HÅRDVARA

MEKANIK

Lego

Bandkanonen byggdes i Lego. Två teknikset av version Dacta användes.

Kanon

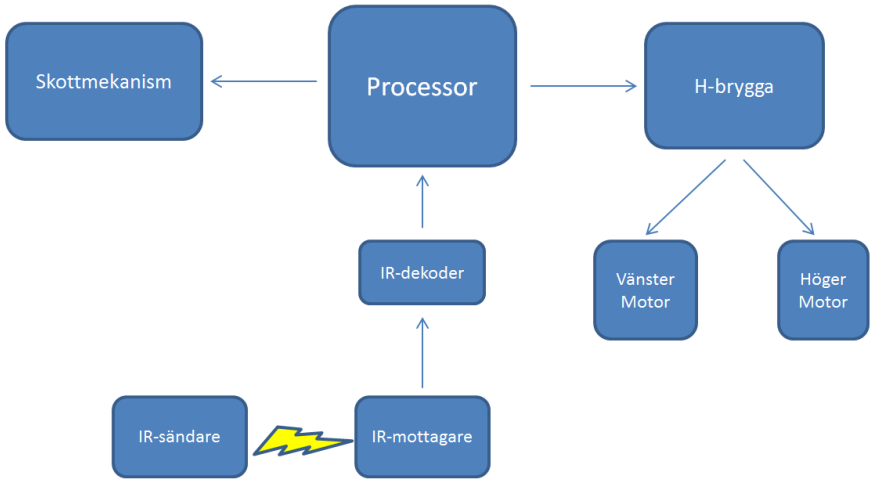
Bandkanonen ska kunna beskjuta sina fiender och vi har därför införskaffat ett leksaksgevär av märke Rapid Fire från Teknikmagasinet. Geväret skjuter skumgummiringar genom att nämnda ringar matas mot en roterande motor som skickar iväg dem med en betydande hastighet. Matningen av ammunition sker då en avtryckare trycks in.

För att geväret ska kunna mata ammunition utan mänsklig inblandning har vi satt in ett servo som kunde aktivera avtryckningen via kommando på fjärrkontrollen.

Motorn i geväret kräver en matningsspänning på 3V och är kopplad till batteriet via en spänningsregulator. Vi har också kopplat en on/off-switch till motorn i geväret för att kunna säkra vapnet om så skulle behövas.

ELEKTRONIK

Ett blockschema med bandkanonens elektroniska komponenter kan beskådas nedan. För en mer detaljerad bild av kopplingar samt hur strömförsörjningen fungerar, se bilaga 2.



Processor – ATMega16

Processorn som använts är en Atmel AVR ATMega16 med 8-bitars mikroprocessor. Den har en mängd integrerade funktioner som intern klocka, avbrottsrutiner, 32 programmerbara I/O-pinnar och 16kb programmerbart minne. Den fungerar i spänningsintervallet 4,5-5,5 V.

Programmeringen av processorn skedde genom anslutning till dator via en JTAG, koden skrevs i AVR Studio 4 i programspråket C.

Processorns uppgift är att delegera och styra signalerna som skickas i bandkanonen.

IR-sändare

Vi använde oss av en vanlig fjärrkontroll som använder standardprotokollet RC-5. Fjärrkontrollen drivs av 2 st 1,5 V-batterier.

IR-mottagare - IRM-8608S

För att ta emot IR-signaler använde vi oss av en IR-diod av modell IRM-8608S. Den tar emot signaler från IR-sändaren (fjärrkontrollen) och skickar vidare signalerna till IR-dekodern.

IR-dekoder - SAA3049A

Dekodningen av IR-signalerna sköts av en Philips SAA3049A-dekoder. IR-signalerna som mottas av IR-mottagaren måste först omvandlas innan de kan användas av processorn. Dekodern kan programmeras att använda ett av två olika IR-protokoll; i detta projekt har vi använt RC5-protokollet.

Motorer 9V – Lego 74569

Bandvagnen drivs av två 9V-motorer från Lego som kopplas till varsin larvfot. På grund av motorernas höga hastighet och låga drivmoment har vi växlat ned dem för att göra dem kapabla att framdriva bandvagnen.

För att köra framåt och bakåt sätts båda motorerna igång åt respektive håll, för rotering vänster/höger sätts den ena motorn igång framåt och den andra bakåt.

H-brygga – L928N

För att styra motorerna så att de kör i rätt riktning används en H-brygga. H-bryggan använder logiska signaler från processorn för att styra motorerna oberoende av varandra – spänningen från batteriet släpps antingen igenom, körs i omvänd riktning eller stoppas.

Spänningsregulatorer – LP3852ET och LP3855ET

Strömförsörjningen till vår bandvagn kommer från ett 6V-batteri. Eftersom alla våra digitala komponenter behöver en matningsspänning på 5V behöver vi reglera spänningen från batteriet med en spänningsregulator av typ LP3852ET.

Motorn i kanonen behöver en matningsspänning på 3V så den kräver en annan spänningsregulator som omvandlar 6V till 3V. Till detta använde vi komponenten LP3855ET.

Parallax Standard Servo

Eftersom avfyrningsmekanismen till geväret krävde mekanisk avfyrning var vi tvungna att använda en servomotor. Motorn opererar i spänningsintervallet 4,8V – 6V. Servot kontrolleras genom att olika långa pulser skickas genom det.

JTAG – Joint Test Action Group

En JTAG användes för att kunna koppla processorn till datorn och där kunna utföra debugging och programmering i AVR Studio.

MJUKVARA

Mjukvaran till bandvagnen skrevs i C och överfördes till processorn via programmet AVR Studio, Atmels egna utvecklingsmiljö. Se bilaga 1 för att se den slutgiltiga koden.

ARBETSGÅNG

Innan vi inledde arbetet med vår bandkanon gjorde vi en kravspecifikation som skulle klargöra det slutgiltiga målet med projektet. Kraven gjordes med avsikt luddiga då vi inte visste mycket om vad som väntade oss i form av tekniska utmaningar och begränsningar. Detta visade sig vara klokt då det underlättade revisioner av bandkanonens design under projektets senare delar. Därefter läste vi på om programmet vi skulle arbeta i och om verktygen vi skulle använda.

Första steget i arbetet var att göra ett kopplingsschema i programmet Powerlogic. Då vi var ganska gröna på elektronik gjordes en del misstag här som vi fick rätta till senare då vi stötte på problem. I huvudsak följdes dock kopplingsschemat.

Därefter började processen med att löda och vira ihop alla komponenter på ett kretskort. Strömmen anslöts och strömförsörjningssladdarna löddes fast vid alla komponenter. Sladdarna som användes för logiska signaler var mindre och virades fast. Här upptäckte vi att vi förbisett viktiga delar av kopplingen då vi ritade upp vårt kopplingsschema så vissa revisioner fick göras. Vi såg till att alla komponenter hade ström och att de gick att styra via datorn.

Under tiden komponenterna kopplades ihop konstruerades chassit och nedväxlingen av motorerna i Lego. Leksaksgeväret togs isär och skruvades ihop igen utan kolv och främre magasin för att lättare kunna monteras på chassit. Lite svarvning och borrning gjorde att geväret kunde fästas på chassit. Vid bygget tog vi hänsyn till att kretskort och batteripaket skulle behöva få plats på bandkanonen.

För att kunna styra bandkanonen med en fjärrkontroll behövde vi därefter skriva programkod. En lång stund ägnades åt att lära oss grunderna i C då vi aldrig använt det förut. Logikpennan var vår bästa vän då vi skulle avgöra vilka pinnar som skickade vilka signaler. Det tog lång tid att få till rätt kod då vi upptäckte att vissa pinnar på processorn var felkopplade och att vi hade svårt att aktivera våra interupts. En mängd tidigare misstag uppdagades under denna process och mycket fick tänkas om.

Efter att styrningen via fjärrkontroll visat sig fungera började vi med projektets sista del – att aktivera avtryckaren med hjälp av servot. Detta visade sig ta längre tid än förväntat då koden var mer komplicerad att skriva än vad vi väntat oss. Dessutom hade vi problem att få servot att aktivera avtryckaren och att få servot att starta i rätt utgångsläge. Detta löstes genom att förstärka och

förlänga avtryckarmekanismen och finjustering av utgångsläge och elevation på kanonen.

Slutligen kunde kanonen både köra och skjuta – konstruktionen var klar.

RESULTAT

Vårt projekt resulterade i en fungerande bandkanon. Vi var inte tvungna att minska ambitionsnivån någon gång under projektets gång. Vi är mycket nöjda med att ha kunnat uppfylla de krav vi formulerade vid projektets start.

SLUTSATSER

LÄRDOMAR

Till vår förtret gick bandkanonen ofta sönder under vårt slutarbete med finjustering och montering av alla komponenter. En fördel med att använda Lego till att bygga med är att det är lätt att ta isär och tänka om konstruktionen om något är fel, en nackdel är Lego i alla lägen inte är särskilt hållbart vilket visade sig i slutändan. För att förbättra detta borde vi ha tänkt på att bygga ett starkare chassi redan från början för att slippa göra sista-minutenjusteringar.

FÖRBÄTTRINGSFÖRSLAG

Om vi hade mer tid så skulle vi försökt bygga bandkanonen med fler material än bara Lego för att öka stabiliteten och förmågan att motstå fiendeeld. Kretskortet ligger relativt oskyddat så att bygga en låda till det skulle också vara en förbättring.

Kanonen visade sig vara ganska tung. Att hitta en lättare version skulle alltså kunna öka bandkanonens hastighet och minska belastningen på chassit.

KÄLLFÖRTECKNING

Datablad till följande komponenter:

- ATMEL (u.d.) ATmega16 Datasheet
- H-brygga L928N
- IR-mottagare - IRM-8608S
- IR-dekoder - SAA3049A
- Spänningsregulatorer – LP3852ET och LP3855ET
- Parallax Standard Servo

Samtliga datablad hämtades från kurshemsidan.

BILAGA 1: KODEN

```
#include <avr/interrupt.h>
#include <util/delay.h>
#include <util/delay_basic.h>
#include <avr/io.h>
#define forward 0b01111001 //knapp 2 på fjärrkontrollen
#define backward 0b01111011 //knapp 0 på fjärrkontrollen
#define left 0b00111010 //knapp 5 på fjärrkontrollen
#define right 0b00111000 //knapp 7 på fjärrkontrollen
#define stop 0b00111001 //knapp 6 på fjärrkontrollen
#define fire 0b01111010 //knapp 1 på fjärrkontrollen
#define fireOCR 0b00001100 // värde på OCR2
#define postFireOCR 0b00101000 // värde på OCR2

volatile int constant;

void command()
{
           switch(PINB & 0b01111011)
           {
                      case stop:
                                 //!Knapp!(6)
                                 PORTA = 0x00;
                                 break;
                      case backward:
                                 //!Knapp!(0)
                                 PORTA = 0x36;
                                 break;
                      case forward:
                                 //!Knapp!(2)
                                 PORTA = 0x1B;
                                 break;
                      case left:
                                 //!Knapp!(5)
                                 PORTA = 0x33;
                                 break;
                      case right:
                                 //!Knapp!(7)
                                 PORTA = 0x1E;
                                 break;
                      case fire:
                                 //!Knapp!(1)
                                 OCR2 = fireOCR;
                                 _delay_ms(80000);
                                 OCR2 = postFireOCR;
                                 break;
                      }
}

void initPorts(){ //för att sätta portarna till rätt startvärden
           DDRA = 0b00111111;
           PORTA = 0b00000000;
           DDRB = 0b00000000;
```

```
            PORTB = 0b01111111;
            DDRC = 0b00000001;
            PORTC = 0b00000001;
            DDRD = 0b10011000;
            PORTD = 0b00000000;
            return;
}

void determineEdge(){ //gör att det genereras interrupt om värdet på T0
ändras
            GICR = (0<<INT2); // Stänger av INT2 för att kunna ändra mcucsr
utan att ett interuppt skapas
            if ((PINB & 0b00000100) == 0b00000100){ // 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 initInt(){ //möjliggör interupts
             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 interrupts
     return;
}

void initPwm(){ //gör en PWM-puls
     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;
}

int main(void)        { //main-metod
            initPorts();
            initInt();
            initPwm();
            while(1){
               constant = OCR2; //loop där programmet hamnar i väntan på
interrupts.
            }
            return 1;
}
```

```
ISR(BADISR_vect) {
  constant  = OCR2;
}



ISR(INT2_vect){  //Avbrott på INT2
           determineEdge();
           command();
}
```

BILAGA 2: KOPPLINGSSCHEMA