Slutrapport - Kitt

EDI022 Digitala Projekt

Grupp 4
Fredrik Bondza
Per Edgren
Handledare: Bertil Lindvall

18 maj 2009

Sammanfattning

This report describes the process of constructing an automated car that can follow a line on the floor. The result of this process is discussed and unforeseen problems are highlighted.

Innehåll

1 Inledning

Denna rapport beskriver en linjeföljande bil som konstruerats av två studenter på Datateknik-programmet. Bilen använder lysdioder och fotodioder för att följa en bana tejpad på golvet. Då de tidiga kurserna på D-programmet främst fokuserar på mjukvara innebar detta att hårdvarubiten blev den största utmaningen. Detta har gjort att större delen av rapporten behandlar hårdvaran.

1.1 Beskrivning av projektet - före

Det ursprungliga syftet med projektet var att konstruera en bil som skulle följa en utstakad bana (i form av tejp på golvyta). Dessutom skulle bilen kunna undvika eventuella hinder samt optimera körbanan efter en lyckad runda.

1.2 Beskrivning av projektet - efter

Bilen kan följa en utstakad bana av hög svårighetsgrad. Den stannar vid eventuellt hinder och fortsätter köra om hindret avlägsnas. Vägen sparas under körning och kan återskapas i efterhand.

2 Komponenter

Följande kompontenter användes:

Komponent	*Antal*
Processor Motorola MC68008	1
Logikkrets PALCE22V10	2
Oscillator EXO3 4D	1
D-vippa 74HC74N	1
H-brygga L298	1
IRdiod LD271	4
Fotodiod SFH203	4
Avståndsmätare SHARP 2YOA02	1
Parallax servo	1
A/D-omvandlare ADC0804LCN	1
Spänningskomparator LM339N	1
SRAM 128KB	1
EPROM 128KB	1
EEPROM 128KB	1
Frekvensdelare 74LS292N	1
Spänningsregulator LP3855ET	1
Latch 74HC373	2

3 Hårdvara

3.1 Processor

Den centrala enheten i konstruktionen är en Motorola 68008-processor. Detta är en renodlad mikroprocessor, utan inbyggt minne, ad-omvandlare eller dylikt. Detta leder till att användaren tvingas hantera fler externa kretsar. Processorn körs i 10MHz och periodiska avbrott genereras varje millisekund genom att dela ner frekvensen från processorn.

3.2 EPROM

Programkoden lagras i ett EPROM-minne. Denna typ av minne måste programmeras med hjälp av särskild utrustning, och måste raderas innan det kan programmeras på nytt. Detta gör det svårare att testa så att hårdvaran kring minnet fungerar och gör även testningen av programvara till en lång process.

3.3 SRAM

SRAM-minne kräver ingen speciell utrustning för att spara data till eller radera data från. Detta gör SRAM-minne enklare att handskas med än minnen av typen EPROM. För att laga eventuell data på SRAM krävs konstant ström. Spara data på SRAM är inga problem skrivningar och läsningar görs på direkta adresser. Stacken och heapen ligger på SRAM, vilket innebär att eventuella variabler och datastrukturer kommer sparas på SRAM.

3.4 EEPROM

EEPROM-minnet består av 1024 sidor med 128 byte minne per sida. Att skriva till EEPROM är mer komplicerat än att skriva till SRAM. Vid skrivning till EEPROM skrivs alltid en hel sida åt gången, därför krävs det att alla 128 byte buffras upp och skrivs till minnet på en gång. Att läsa från EEPROM skiljer sig däremot ej från att läsa från SRAM-minne. För att lagra körbanan användes ett EEPROM-minne då detta inte kräver någon strömförsörjning för att hålla kvar datan. Detta gör att bilen kan upprepa banan utan tejp efter att den varit avstängd.

3.5 H-brygga

För styra riktning och hastighet på motorerna skickas pulser till en H-brygga. Strömmen från IC-kretsarna är ej tillräcklig för att driva motorerna och därför används H-bryggan. Beroende på vilka insignaler som skickas till bryggan kan den antingen släppa igenom eller hindra ström till motorerna. Detta gör det även möjligt att köra motorerna med högre spänning än IC-kretsarna klarar.

3.6 Dioder

För att avgöra hur bilen är placerad i förhållande till tejpen används fyra par av dioder av två olika typer. En IR-diod för att sända ut IR-ljus som reflekteras av golv och tejp. För att registrera vilken mängd IR-ljus som reflekterats används fotodioder vilka släpper igenom olika mycket ström beroende på mängden ljus som upptas. För att se om en diod registrerar tejp eller inte jämförs spänningen med en referensspänning i en komparator. Utgångarna från de fyra komparatorerna bildar ett bitmönster vilket gör det enkelt att bestämma bilens position relativt tejpen. Referensspänningen kan justeras med hjälp av en vridpotentiometer för att kunna anpassas efter underlag och ljusförhållande.

3.7 A/D-omvandlare

För att omvandla den analoga spänningen från avståndsmätaren till ett digitalt värde användes en enkanalig A/D-omvandlare. Genom att återkoppla EOC-signalen (End Of Conversion) till Start-signalen fås en kontinuerlig omvandling av det uppmätta avståndet. På så sätt finns det alltid ett aktuellt värde när programmet frågar efter det.

3.8 Spänningsregulator

Batteriet som används för att driva bilen ger en spänning på 6 Volt. Det går ej att driva IC-kretsarna med spänningen från batteriet direkt, då logisk etta representeras av +5 V. Spänningsregulatorn transformerar en godtycklig spänning till 5 V. På så sätt kan IC-kretsarna drivas med en spänning på 5 V och spänningen till motorerna bestäms av vilken spänning källan levererar.

4 Mjukvara

All sampling, reglering och styrning görs i mjukvara, och därför var detta en stor del av projektet.

4.1 Avbrottsrutin

Då motorerna och servot styrs med pulsmodulering (PCM) krävdes periodiska avbrott för att kunna garantera att de tidskritiska kraven uppfylldes. Ett typiskt avbrott innehåller följande steg:

- Avståndsmätning
- Servostyrning
- Motorstyrning
- Uppdatering av positionshistorik

- Lagring av motorläge

4.2 Bakgrundsprocess

När processorn är i normalläge utförs följande uppgifter:

- Positionsläsning
- Derivataberäkning
- PD-reglering
- Skrivning till EEPROM

4.3 PD-reglering

För att få så bra styrning som möjligt implementerades en PD-regulator. Eftersom processen innehåller många snabba förändringar krävdes en derivatadel för att förhindra att bilen börjar svänga okontrollerat. Viktning mellan proportionell del och derivatadel påverkar bilens beteende och var en stor utmaning. Derivatan approximerades med hjälp av ett antal äldre mätpunkter.

5 Problematik

Vid projektets slut blev det tydligt att alla de uppsatta målen (kraven) inte skulle kunna nås inom tidsramen för projektet. Svårigheterna med att få hårdvaran att samspela och att effektivt reglera styrprocessen hade underskattats vid projektstart.

Genom projektets gång har många av de ursprungliga idéerna fått ge vika för nya lösningar. Detta har gjort att tidsåtgången för hela projektet varit större än väntat. Ett exempel på detta är positionsläsningssystemet; från början skulle en flerkanalig A/D-omvandlare användas för att avgöra om dioderna är över tejpen eller inte. När idéen med komparatorn föddes innebar det att en del av systemet fick designas om.

När servot kopplades in visade det sig att styrningen krävde pulser med högre upplösning än vad som kunde genereras av konstruktionen. Servot kan nu svepa 180 grader men ej fixeras vid ett bestämt gradtal. Bilen kan alltså inte undvika hinder då hindrens position inte kan bestämmas. Istället stannar bilen när ett föremål är i vägen och fortsätter om hindret avlägsnas.

6 Avslutning

Detta projekt har bestått av design och implementation av både hårdvara och mjukvara. Det har gett en bra inblick i hur IC-kretsar kan användas och kombineras för att utföra en specifik uppgift. Då beräkningskraft och minne

varit begränsade har en större förståelse för programmering i mindre system erhållts.

Eftersom projektet gick ut på att bygga en prototyp helt från grunden har en helhetsbild av utvecklingsprocessen för ett system givits. En viktig erfarenhet är att alla beslut som tas under hårdvaruutvecklingen påverkar (mer eller mindre) mjukvaruutvecklingen senare i projektet. På grund av den ringa erfarenheten av hårdvarunära projekt var den ursprungliga tidsuppskattningen inte realistisk. Detta i kombination med höga krav gjorde att vissa mål inte kunde uppfyllas helt.

7 Appendix - Programkod

7.1 kitt.h

```
/*
 *   Uncomment this line to enable debug mode. In this mode,
 *   motors are disabled and printouts can be made to the
 *   it68 development environment.
 */
/*#define DEBUG*/

/*
 *   Uncomment this line to disable the distance sensor and
 *   the servo.
 */
#define SENSE

/* Define addresses for I/O */
#define ADDR_DIODES 0xC0000
#define ADDR_MOTORS 0x80000
#define ADDR_AD 0xE0000
#define ADDR_EEPROM 0x20000
#define PAGE_SIZE 128

/* Motor modes */
#define LEFT 0x38
#define RIGHT 0x32
#define FREE_RUNNING 0x00

/* Number of motor pulses (resolution of motor speed) */
#define NBR_PULSES 10

/* History size */
#define NBR_STORED_OFFSETS 45

/* Number of interrupts for each step in a transition */
#define TIME_CONSTANT 20

/* Circular list structure used to store offset history */
typedef struct listnode listnode;
struct listnode
{
   listnode *next;
   listnode *prev;
   signed short val;
};

/* Global variables and data structures */
unsigned short *motors;
unsigned short *diodes;
unsigned short *ad;
unsigned short *eeprom;
unsigned short pulses[NBR_PULSES];
```

```
listnode history[NBR_STORED_OFFSETS];
listnode *history_head;
unsigned short last_offset;
unsigned short latch_val;
unsigned short pulse_counter;
unsigned int power_counter;
unsigned int servo_counter;
unsigned int servo_pulser;
unsigned short power_left;
unsigned short power_right;
unsigned short motor_pause;
unsigned int pause_counter;
unsigned short page_entries[PAGE_SIZE];
unsigned short curr_column;
unsigned int curr_page;
unsigned short page_full;
unsigned short replay_mode;

/* Help functions */
void init(void);
void pulse_servo(void);
void step_playback(void);
void step_normal(void);
void push_to_history(signed short);
signed int calc_derivative(void);
void regulate_PD(signed short, signed int);
signed short read_offset(void);
void set_pulse_array(void);
void set_motors(unsigned short);
void set_diode(unsigned short);
void set_servo(unsigned short);
void write_to_motor_latch(void);
void store_value(unsigned short);
void write_to_eeprom(void);

#ifdef DEBUG
unsigned short sum_history(void);
void print_short(signed short);
#endif
```

7.2 kitt.c

```
#include "kitt.h"

int main(void)
{
   init();

   /* Self-test of AD converter */
   if (255 != *ad)
   {
      unsigned int i = 0;
      set_diode(0xFF);
      write_to_motor_latch();
      while (i < 50000) i++;
      set_diode(0x00);
      write_to_motor_latch();
   }

   /* Go to replay mode if no tape */
   if (0x0F == *diodes)
   {
      replay_mode = 1;
      set_diode(0xFF);
   }

   _avben();

   while (replay_mode) { }

   while (1)
   {
      signed int dxdt;
      /* Perform page write with interrupts disabled */
      if (page_full)
      {
         set_diode(0xFF);
         write_to_motor_latch();
         _avbdi();
         write_to_eeprom();
         _avben();
         set_diode(0x00);
         write_to_motor_latch();
      }
      /* Find P and D components and feed to regulator */
      last_offset = read_offset();
      dxdt = calc_derivative();
      regulate_PD(last_offset, dxdt);
   }
}

/*
 *   Initializes all global variables and data structures.
 */
```

```
void init(void)
{
   unsigned int i;

   motors = (unsigned short*)ADDR_MOTORS;
   diodes = (unsigned short*)ADDR_DIODES;
   ad = (unsigned short*)ADDR_AD;
   eeprom = (unsigned short*)ADDR_EEPROM;

   for (i = 0; i < NBR_PULSES; i++)
   {
      pulses[i] = FREE_RUNNING;
   }

   for (i = 0; i < NBR_STORED_OFFSETS; i++)
   {
      listnode node;
      listnode *tmp = &node;
      tmp->val = 0;
      tmp->next = &history[i + 1];
      tmp->prev = &history[i - 1];
      history[i] = *tmp;
   }
   history[NBR_STORED_OFFSETS - 1].next = &history[0];
   history[0].prev = &history[NBR_STORED_OFFSETS - 1];
   history_head = &history[0];

   last_offset = 0;
   latch_val = 0;
   pulse_counter = 0;
   power_counter = 0;
   servo_counter = 0;
   servo_pulser = 3;
   power_left = 0;
   power_right = 0;
   motor_pause = 0;
   pause_counter = 0;

   for (i = 0; i < PAGE_SIZE; i++)
   {
      page_entries[i] = 0;
   }
   curr_column = 0;
   curr_page = 0;
   page_full = 0;
   replay_mode = 0;
}

/*
 *   The periodic interrupt routine.
 */
exp2()
{
#ifdef SENSE
```

```
   /* Read value of AD converter */
   unsigned short ad_val = 0;
   if (10 == pulse_counter)
   {
      ad_val = *ad;
   }
   /* Set PCM sequence for servo */
   pulse_servo();
#endif
   if (!motor_pause)
   {
      if (replay_mode)
      {
         step_playback();
      }
      else
      {
         step_normal();
      }
      pulse_counter++;
   }
#ifdef SENSE
   else
   {
      pause_counter++;
      if (2000 == pause_counter)
      {
         motor_pause = 0;
         pause_counter = 0;
      }
      else if (ad_val >= 130)
      {
         pause_counter = 0;
      }
   }
   if (ad_val >= 130)
   {
      motor_pause = 1;
      set_motors(FREE_RUNNING);
   }
#endif
   write_to_motor_latch(); /* flush contents of latch_val */
}

/*
 *   This function controls the PCM pulse to the servo. Should be
 *   called once during each interrupt.
 */
void pulse_servo(void)
{
   if (23 == servo_pulser)
   {
      set_servo(0xFF);
      if (50 == servo_counter)
```

```
      {
         servo_counter = 0;
      }
      servo_counter++;
      servo_pulser = 3;
   }
   else if (22 == servo_pulser && 25 < servo_counter)
   {
      set_servo(0xFF);
   }
   else
   {
      set_servo(0x00);
   }
   servo_pulser++;
}

/*
 *   Sets the motors for this cycle and steps to next
 *   address in memory.
 */
void step_playback(void)
{
   set_motors(*(eeprom + curr_page + curr_column));
   curr_column++;
   if (128 == curr_column)
   {
      curr_page += 128;
      curr_column = 0;
   }
   if (10 == pulse_counter)
   {
      pulse_counter = 0;
   }
}

/*
 *   Sets the motors for this cycle and stores the offset
 *   in the history list.
 */
void step_normal(void)
{
   if (10 == pulse_counter)
   {
      pulse_counter = 0;
      set_pulse_array();
      push_to_history(last_offset);
   }
   set_motors(pulses[pulse_counter]);
}

/*
 *   Saves an offset into the circular history list.
 */
```

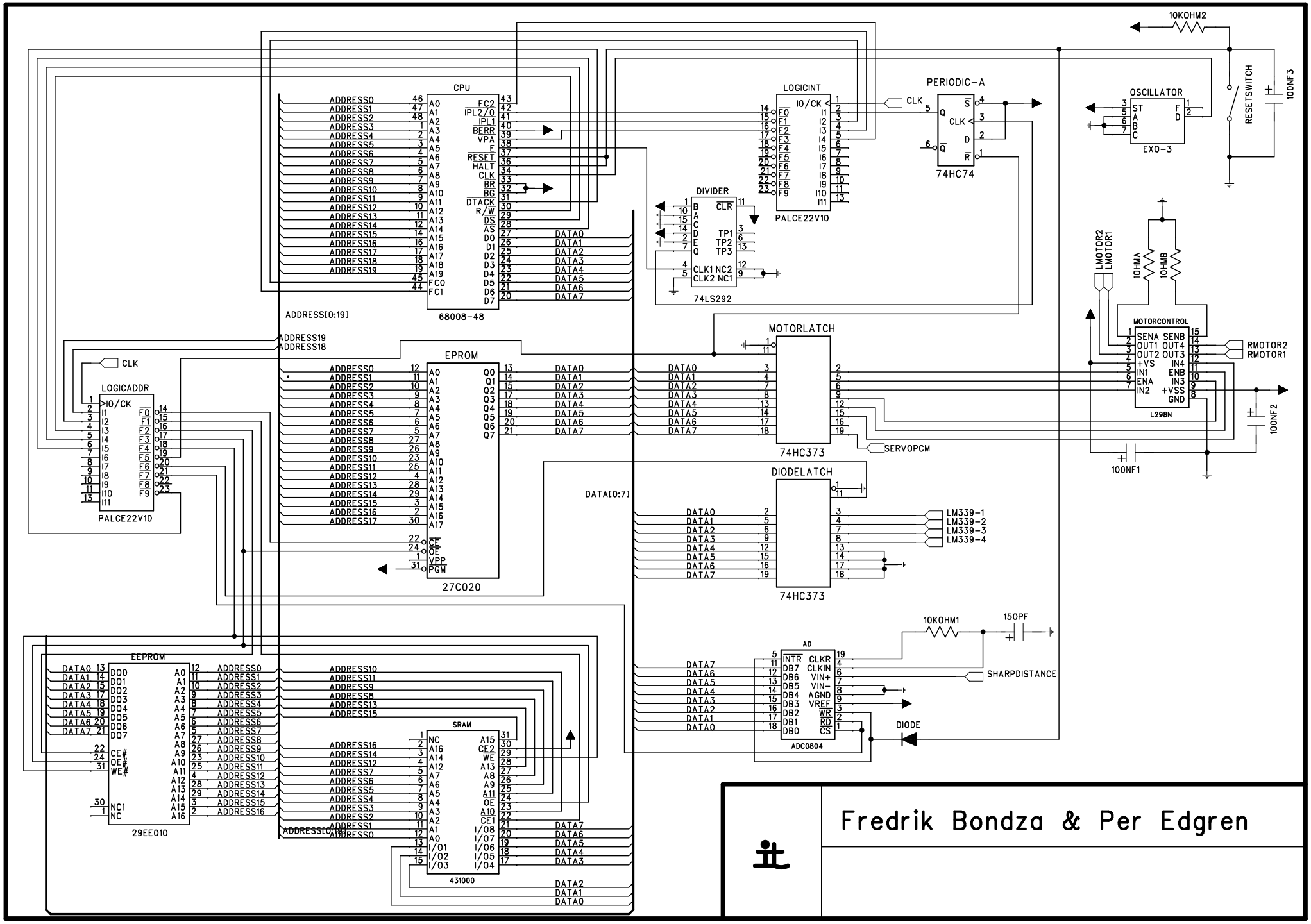
```
void push_to_history(signed short offset)
{
   history_head = history_head->next;
   history_head->val = offset;
}

/*
 *   Returns an approximation of the derivative:
 *   1) Check for the latest change in offset.
 *   2) Compute derivative so that larger offset gives greater
 *      derivative and longer time decreases the value.
 */
signed int calc_derivative(void)
{
   listnode *curr = history_head;
   signed short t0_val = history_head->val;
   unsigned short t = 1;
   do
   {
      if (t0_val - curr->val)
      {
         break;
      }
      curr = curr->prev;
      t++;
   } while (curr != history_head);
   if (t != NBR_STORED_OFFSETS)
   {
      return (t0_val - curr->val) * (NBR_STORED_OFFSETS - t);
   }
   else
   {
      return 0;
   }
}

/*
 *   Sets the target power for both motors using a proportional
 *   part and a derivative part.
 */
void regulate_PD(signed short P, signed int D)
{
   signed short D_effect;
   signed short left = 6; /* initial power */
   signed short right = 6;

   if (-127 == P) /* lost tape, continue in same direction */
   {
      return;
   }
   else if (126 == P) /* in the air, stop motors */
   {
      power_left = 0;
      power_right = 0;
```

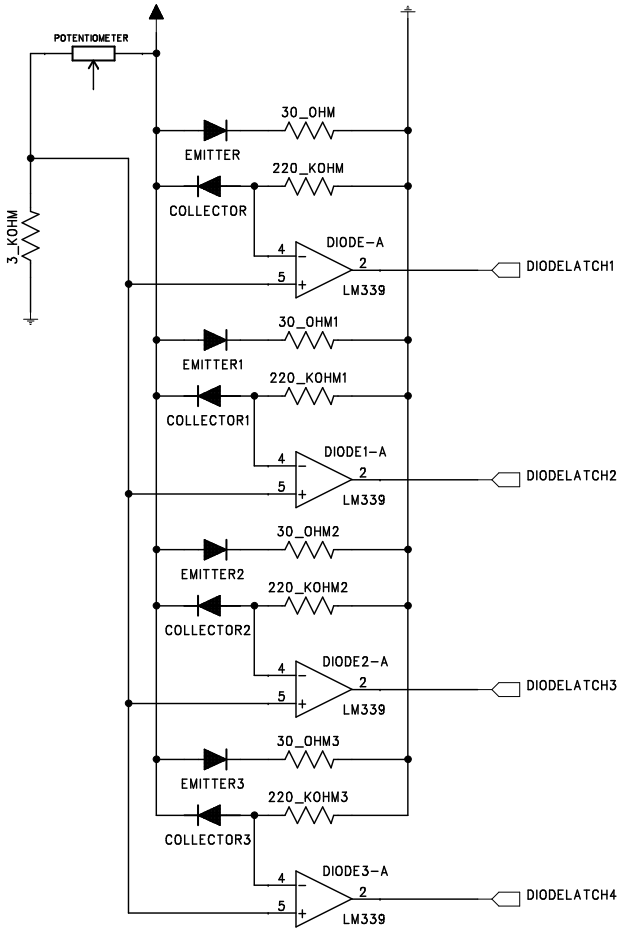
```
      return;
   }

   /* Do the P */
   left += P;
   right -= P;

   /* Do the D */
   if (D > 135) D = 135; /* max(D) = 90 */
   else if (D < -135) D = -135; /* min(D) = -90 */
   D_effect = D / 34; /* D_effect [-3,3] */

   left += (D_effect - 1); /* D_effect [-4,2] */
   right -= (D_effect + 1); /* D_effect [-2,4] */

   if (left > 10) left = 10;
   else if (left < 0) left = 0;
   if (right > 10) right = 10;
   else if (right < 0) right = 0;

   power_left = (unsigned short)left;
   power_right = (unsigned short)right;
}

/*
 *   Sets up the pulses vector. Decides the mode of each motor
 *   for the NBR_PULSES upcoming interrupts.
 */
void set_pulse_array(void)
{
   int i;
   for (i = 0; i < NBR_PULSES; i++)
   {
      pulses[i] = FREE_RUNNING;
   }
#ifndef DEBUG
   for (i = 0; i < power_left; i++)
   {
      pulses[i] |= LEFT;
   }
   for (i = 0; i < power_right; i++)
   {
      pulses[i] |= RIGHT;
   }
#endif
}

/*
 *   Reads the value of the sensor diodes and returns a decimal
 *  representation.
 */
signed short read_offset(void)
{
   unsigned short int pattern = *diodes;
```

```
   pattern ^= 0x0F; /* this makes active diode binary 1 */

   switch (pattern)
   {
      case 0x00:   return -127;   /* lost tape, continue in the same direction */
      case 0x08:   return 5;
      case 0x0C:   return 3;
      case 0x04:   return 1;
      case 0x02:   return -1;
      case 0x03:   return -3;
      case 0x01:   return -5;
      case 0x0F:   return 126;      /* in the air (probably) */
      default:   return 0;
   }
}

/* Append first 6 bits of pattern to motor/diode latch */
void set_motors(unsigned short pattern)
{
   if (!motor_pause && !replay_mode)
   {
      store_value(pattern);
   }
   latch_val = (latch_val & 0xC0) | (pattern & 0x3F);
}

/* Append bit 7 of pattern to motor/diode latch */
void set_diode(unsigned short pattern)
{
   latch_val = (latch_val & 0xBF) | (pattern & 0x40);
}

/* Append bit 8 of pattern to motor/diode latch */
void set_servo(unsigned short pattern)
{
   latch_val = (latch_val & 0x7F) | (pattern & 0x80);
}

/* Write latch_val to motor/diode latch */
void write_to_motor_latch(void)
{
   *motors = latch_val;
}

/*
 *   Queue a value for writeback to eeprom. The value will not be
 *   written until the page is full.
 */
void store_value(unsigned short value)
{
   page_entries[curr_column] = value;
   curr_column++;
   if (curr_column == 128)
   {
```

```
      page_full = 1;
      curr_column = 0;
   }
}

/*
 *   Write the active page to eeprom.
 */
void write_to_eeprom(void)
{
   int i;
   /* disable protection */
   *(eeprom + 0x5555) = 0xAA;
   *(eeprom + 0x2AAA) = 0x55;
   *(eeprom + 0x5555) = 0xA0;
   /* write page */
   for (i = 0; i < PAGE_SIZE; i++)
   {
      *(eeprom + curr_page + i) = page_entries[i];
   }
   curr_page += 128;
   page_full = 0;
}

/*
 *   The following functions can be used if DEBUG mode is active.
 *   This is set in the header file.
 */
#ifdef DEBUG

/*
 *   Returns the sum of the nodes in the history list.
 *   CAUTION: if NBR_STORED_OFFSETS is large, the sum may not
 *   fit in 8 bits.
 */
unsigned short sum_history(void)
{
   listnode *tmp;
   unsigned short sum, i;
   sum = 0;
   tmp = history_head;
   for (i = 0; i < NBR_STORED_OFFSETS; i++)
   {
      sum += tmp->val;
      tmp = tmp->prev;
   }
   return sum;
}

/*
 *   Function for printing a signed short to the it68
 *   development environment.
 */
void print_short(signed short val)
```

```
{
   char outbuf[6];
   int i = 0;
   int j = 0;
   char tmp;
   if (val < 0)
   {
      outbuf[i] = '-';
      i++;
      j++;
      val = -val;
   }
   do /* find the chars */
   {
      outbuf[i] = (char)(48+ val%10);
      i++;
   } while ((val /= 10) > 0);
   outbuf[i] = '\n';
   outbuf[i + 1] = '\0';
   i--;
   do /* swap the chars */
   {
      tmp = outbuf[j];
      outbuf[j] = outbuf[i];
      outbuf[i] = tmp;
      j++;
      i--;
   } while (i - j > 1);
   _print_str(outbuf);
}
#endif
```


CPU
68008-48
LOGICINT
PALCE22V10
PERIODIC-A
74HC74
DIVIDER
74LS292
OSCILLATOR
EXO-3
10KOHM2
RESETSWITCH
100NF3
LMOTOR2
LMOTOR1
10HMA
10HMB
MOTORCONTROL
L298N
RMOTOR2
RMOTOR1
100NF2
100NF1
LOGICADDR
PALCE22V10
EPROM
27C020
MOTORLATCH
74HC373
SERVOPCM
DIODELATCH
74HC373
LM339-1
LM339-2
LM339-3
LM339-4
EEPROM
29EE010
SRAM
431000
AD
ADC0804
10KOHM1
150PF
SHARPDISTANCE
DIODE
ADDRESS[0:19]
DATA[0:7]
Fredrik Bondza & Per Edgren



Fredrik Bondza & Per Edgren