

Projektdokumentation

Prozeßdatenverarbeitung Vertiefungsveranstaltung 2006

| | |
|---------------------------|---|
| Thema: | Aufzugssteuerung per PIC18F452 |
| Projektteilnehmer: | Maxim Albertin Oskar Horch Alexander Bitz |
| Auftraggeber: | Prof. Dr. Karl-Otto Linn |

Inhaltsverzeichnis

| | |
|--|----|
| Aufgabenstellung..... | 3 |
| Zur Verfügung stehende Komponenten..... | 3 |
| Vorüberlegungen für das Zusammenspiel der Komponenten..... | 4 |
| Allgemeines..... | 4 |
| Aufzugsschacht samt Wagen..... | 4 |
| Antrieb..... | 4 |
| Steuerung..... | 4 |
| Realisierung..... | 5 |
| Fahrstuhlschacht..... | 5 |
| Antrieb..... | 6 |
| Steuerung..... | 8 |
| Allgemeines..... | 8 |
| Hardware..... | 8 |
| Microcontroller PIC18F452..... | 8 |
| Der In-Circuit Debugger | 9 |
| Der HP Druckermotor | 10 |
| Die H-Brücke..... | 10 |
| Schaltplan für die Beschaltung der H-Brücke..... | 11 |
| Software..... | 11 |
| Die Entwicklungsumgebung | 11 |
| Funktionsweise der Anwendung..... | 12 |
| Fazit..... | 12 |
| Quellen..... | 13 |
| Anhang..... | 13 |
| Quellcode..... | 13 |

Aufgabenstellung

Es ist eine Steuerung für einen Aufzug per PIC18F452 zu realisieren.

Um die Steuerung entsprechend gut illustrieren zu können, ist außerdem noch ein Fahrstuhl samt Fahrstuhlschacht zu bauen.

Zur Verfügung stehende Komponenten

Für die Projektrealisierung werden uns seitens Prof. Dr. Linn folgende Komponenten zur Verfügung gestellt:

- MPLAB ICD 2 In-Circuit Debugger (Debugger und Programmer)
- PICDEM 2 PLUS Board
- H-Brücke: LMD18200T (Steuerung der Motordrehrichtung)
- Motor mit vormontierter Schlitzscheibe samt Lichtschranke

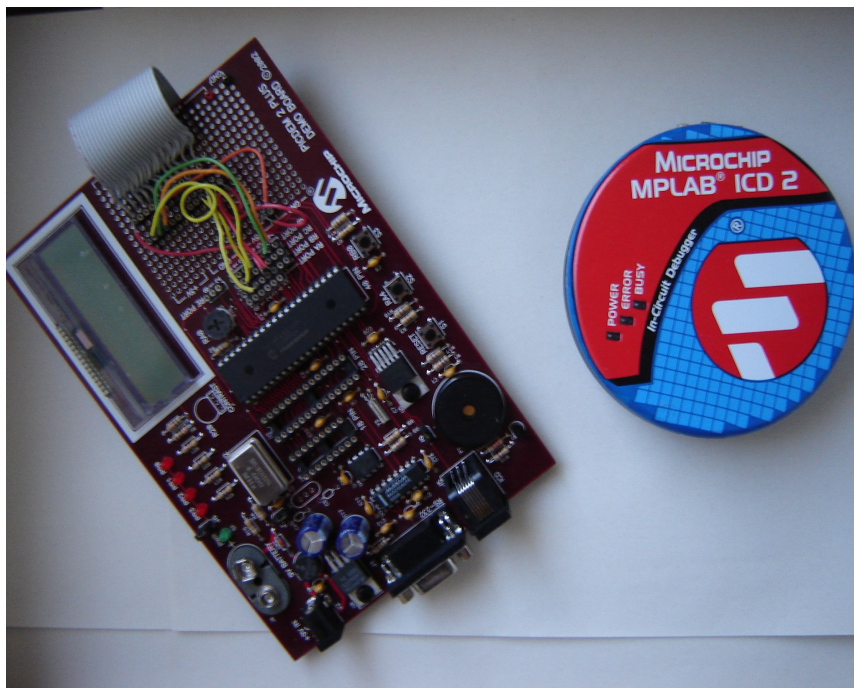


Abbildung 1: Demoboard und Debugger/Programmer

Vorüberlegungen für das Zusammenspiel der Komponenten

Allgemeines

Für die einfachere Realisierung des Projektes wurde dieses in Teile gesplittet, die auch parallel realisiert werden können (falls möglich). Wir unterteilten das Projekt grob in folgende Teile:

- Aufzugsschacht samt Wagen
- Antrieb
- Steuerung

Aufzugsschacht samt Wagen

Der Fahrstuhlschacht spielt eine zentrale Rolle im Projekt, denn dieser dient als Führung für den Fahrstuhl. Auf die Turmspitze werden zwei Führungsrollen angebracht, welche das Seil von der Wickelrolle am Fahrstuhlfuß an die Kabine leiten sollen.

Die Steuerkonsole am Fahrstuhl wird weggelassen, stattdessen wird eine Steuerkonsole auf einer externen Platine zusammengestellt, die für die Simulation völlig ausreichend ist. Der Schacht soll eine Höhe von etwa einem Meter haben.

Um bestmöglichen Demonstrationseffekt zu erzielen, sollte der Fahrstuhl innerhalb des Schachtes gut sichtbar sein.

Antrieb

Der Fahrstuhlantrieb wird außerhalb des Fahrstuhlschachtes montiert. Dieser besteht aus dem Motor, Umlenkrollen mit Zahnriemen als Kraftüberträger vom Motor auf die Wickelrolle für das Seil, an dem der Fahrstuhl hängen wird.

Steuerung

Die Fahrstuhlsteuerung besteht in unserem Fall aus Hard- und Software. Unter der Entwicklungsumgebung wird die Steuersoftware erstellt und mittels des Programmiers auf den PIC hoch geladen. Bestimmte Ausgänge des Demoboards werden mit den Eingängen der H-Brücke verbunden. Die H-Brücke ermöglicht es dem PIC die Geschwindigkeit und die Drehrichtung des Motors zu regeln. Da auf dem Demoboard nicht genügend Platz für die H-Brücke vorhanden ist, wird diese auf einer separaten Platine aufgebaut.

Außerdem wird auf einer weiteren Platine die gesamte Schaltlogik angebracht.

Realisierung

Fahrstuhlschacht

Für den Fahrstuhlschacht wurden die Maße 13x13x100 (cm b/t/h) gewählt. Zunächst wird ein Gerüst errichtet, indem 4 Alu-Winkel mit den Maßen 15x15x1000 (mm b x t x h) mit 8 15x15x130 (mm) Winkeln untereinander verschraubt werden. In das Gerüst werden von innen Plexiglasscheiben geklebt, dies macht das Gerüst stabiler und garantiert durch den Einsatz eines durchsichtigen Materials eine gute Sicht in den Fahrstuhlschacht.

In die vier Ecken des Turms werden vier zusätzliche Alu-Winkel als Führungsschienen für den Fahrkorb geklebt.

Auf dem entstandenen Turm werden an der Spitze zwei Umlenkrollen platziert. Die Wickelrolle befindet sich am Turmfuß, außerhalb des Turms. Auf die Wickelrolle wird das Seil aufgewickelt und an dieses der Fahrstuhl gehängt.

Die nachfolgende Abbildungen zeigen den Fahrstuhlschacht und die Aufwickelrolle für das Seil.



Abbildung 2: Fahrstuhlschacht

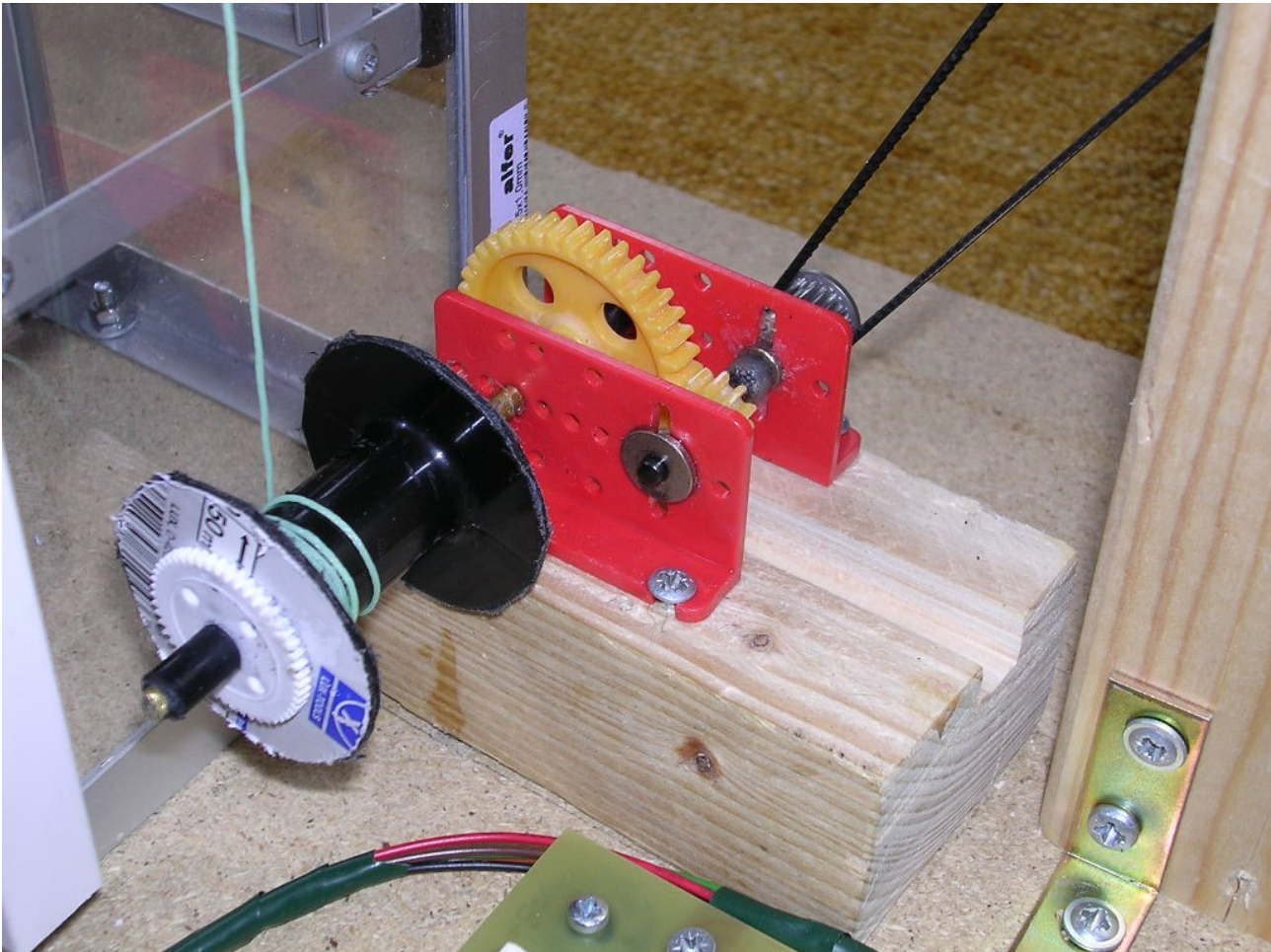


Abbildung 3: Wickelrolle

Antrieb

Da der Motor bereits mit komplett montierter Schlitzscheibe samt Lichtschranke zur Verfügung steht, ist es eine Leichtigkeit diesen in das Projekt zu integrieren, vorzugsweise auf dem Boden, da dieser ein nicht zu vernachlässigbares Eigengewicht aufweist und dadurch den gesamten Turm zum Kippen bringen kann, ist es ratsamer den Turmschwerpunkt an den Boden zu verlagern. Ist der Motor einmal montiert, muss dieser nur noch mit der Stange gekoppelt werden, die die Wickelrolle antreibt. Hierzu wird in die Führung des Zahnrades des Motors und dem der Stange ein Zahnriemen gespannt.

Die nachfolgende Abbildung zeigt unser komplettes Getriebe:

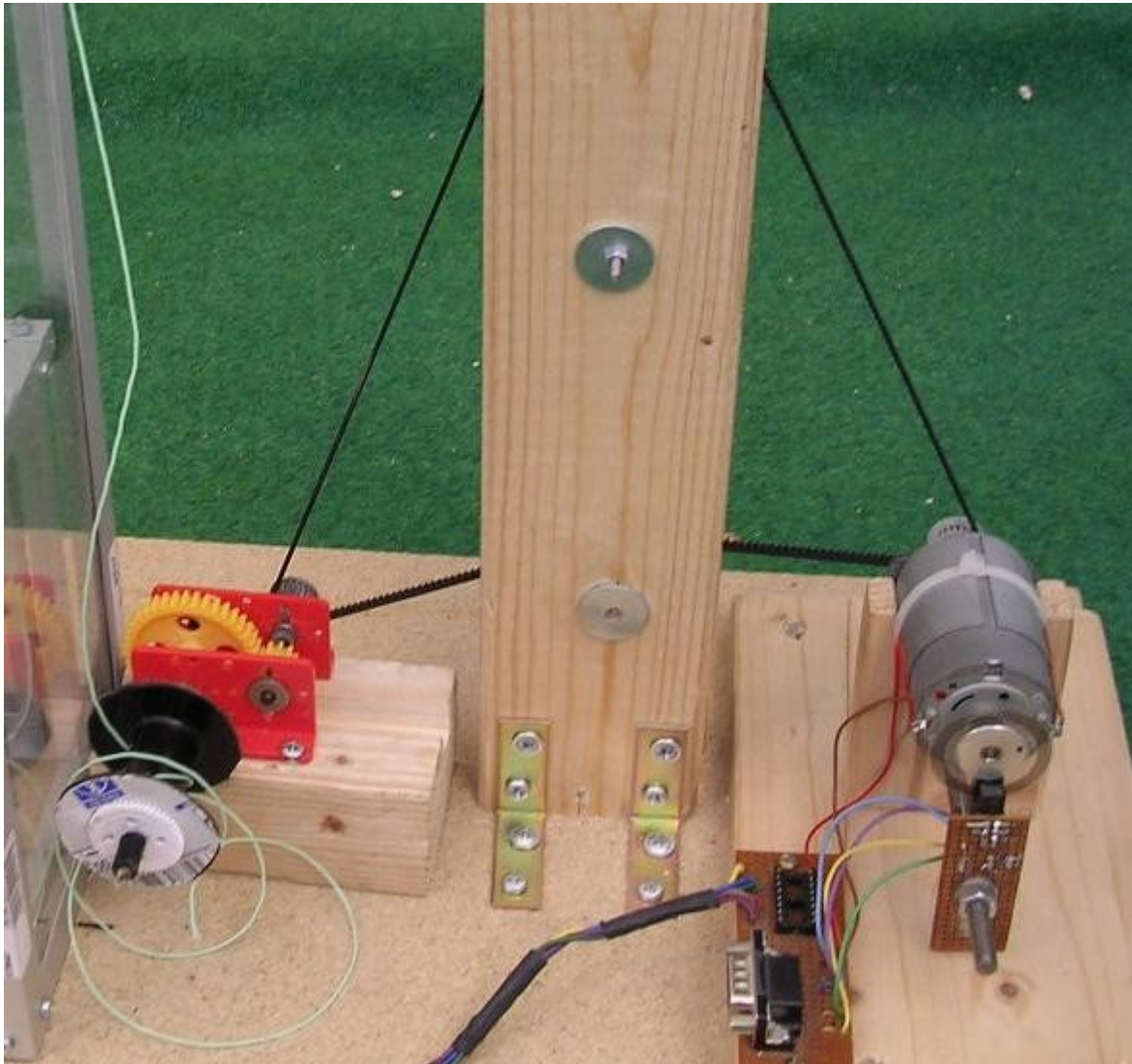


Abbildung 4: Getriebe

Steuerung

Allgemeines

Zur Realisierung der in den Vorüberlegung festgestellten Aufgaben, muss sowohl die Steuerungshardware als auch die dafür benötigte Software erstellt und evtl. konfiguriert werden.

Hardware

Microcontroller PIC18F452

Microcontroller sind preisgünstige Ein-Chip-Computersysteme, in denen alle Komponenten (CPU, ROM, RAM) auf einem einzigen Chip untergebracht sind. Microcontroller verwendet man für kleine Steuerungsprobleme die mit analogen oder diskreten digitalen Schaltungen einen hohen Aufwand erfordern würden. Sie finden in eingebetteten Systemen Verwendung z. B. ABS, Videorekorder, Waschmaschine, da sie sehr günstig sind. Ihre Grundlegende 8 Bit Struktur stammt aus den 70er Jahren, mit Speicher bis 1MB für Programmcode und 48KB für Daten. Somit kann man sie ohne viel Aufwand betreiben, aber auch nur einfache Berechnungen (8-Bit Zahlen) ausführen. Entgegen den Eprom-Microcontoller die man mit Strom beschreibt, aber nur mit UV-Strahlen löschen kann, gibt es heute immer mehr Flash-Microcontroller, die sich elektronisch löschen und wiederbeschreiben lassen. Auch der 18F452 gehört zu der Reihe der flashbaren Systemen. Die Programme für diese Microcontroller werden in Assembler oder C programmiert. Es ist angebracht dazu eine Software Entwicklungsumgebung zu verwenden, wie z.B. MPLAB der Firma Microchip. Mit einem In-Circuit-Debugger, werden die geschriebenen Programme auf den Prozessor übertragen und können im Einzelschrittmodus durchlaufen werden. Der PIC 18F452 (High Performance Enhanced Architecture) gehört zur Mikrocontroller-Familie der Firma Microchip Technology Inc. Dabei handelt es sich um einen 8 Bit RISC Mikrocontroller. Er besitzt eine Lineare Speicheradressierung, hat 79 Befehle, 8x8 Hardware-Multiplizierer, 16 Bit Befehlsbreite und 10 MIPS. Die Speicheraufteilung ist nach der Harvard-Architektur ausgeführt. Der Programmspeicher ist in Pages (Seiten) organisiert, er ist 32Kb groß und es lassen sich Programme mit 16384 Assemblerbefehlen speichern. Der Datenspeicher ist in 4 Bänke zu je 64 Bytes unterteilt. Die Bankumschaltung erfolgt durch die Bits RP0 und RP1 im STATUS Register. Der Stack ist nur als Speicher für den Befehlszähler verwendbar. Da der PIC nur 8 Stufen tief ist wird nach 8 Unterprogramm- /Interruptaufrufen wieder die erste Stufe des Stacks beschrieben, was zum Verlust der ersten Rücksprungadresse führt.

PICs besitzen zwar viele Interruptquellen, haben aber nur einen Interruptvektor, der bei Auslösung eines beliebigen Interrupts angesprungen wird. Die einzige Interruptbehandlungsroutine muss durch Abfragen der einzelnen Flag-Bits herausfinden, welcher Interrupt ausgelöst wurde (es ist möglich hier eine Prioritätssteuerung zu realisieren) und danach die dafür entsprechenden Programmschritte ausführen.

Folgende Abbildung zeigt die Pinbelegung unseres PIC:

| | | | | | |
|---------------------|------------|----|----|-----|--------------|
| | MCLR | 1 | 40 | RB7 | |
| | RA0 | 2 | 39 | RB6 | |
| | RA1 | 3 | 38 | RB5 | |
| | RA2 | 4 | 37 | RB4 | |
| | RA3 | 5 | 36 | RB3 | |
| Taster: Help | RA4 | 6 | 35 | RB2 | |
| Taster: Not-Aus | RA5 | 7 | 34 | RB1 | |
| | RE0 | 8 | 33 | RB0 | LS1 Interupt |
| | RE1 | 9 | 32 | VDD | |
| | RE2 | 10 | 31 | Vss | |
| | VDD | 11 | 30 | RD7 | TasterA4 |
| | Vss | 12 | 29 | RD6 | TasterA3 |
| | OSC1 | 13 | 28 | RD5 | TasterA2 |
| | OSC2 | 14 | 27 | RD4 | TasterA1 |
| H-Brücke: Direction | RC0 | 15 | 26 | RC7 | LED4 |
| H-Brücke: Brake | RC1 | 16 | 25 | RC6 | LED3 |
| H-Brücke: PWM | RC2 | 17 | 24 | RC5 | LED2 |
| LS2 | RC3 | 18 | 23 | RC4 | LED1 |
| TasterB1 | RD0 | 19 | 22 | RD3 | TasterB4 |
| TasterB3 | RD1 | 20 | 21 | RD2 | TasterB2 |
| | | | | | |
| | * Eingänge | | | | |
| | * Ausgänge | | | | |

Der In-Circuit Debugger

Der In-Circuit-Debugger ICD 2 der Firma Microchip arbeitet auf PC-Basis und unterstützt die Microcontroller PIC16F und PIC18F.

Sobald neue Microcontroller Versionen erhältlich sind, kann der Benutzer die entsprechende Software kostenlos in den ICD 2 laden. Download unter: www.microchip.com

Der Microchip ICD 2 wird benötigt, um das Demo-Board zu programmieren, oder um ein Programm zu debuggen. Dazu wird das Programm nicht auf dem Board selbst gespeichert, sondern auf dem Debugger. Zum Programmieren und Debuggen ist der ICD 2 über einen RJ11-Anschluss mit dem Board verbunden. Komfortables Debugging ermöglicht es dem Anwender Breakpoints zu setzen, sowie Lese und Schreibvorgänge im Speicher zu übernehmen.

Der HP Druckermotor

Der Motor in der Aufzugkonstruktion ist ein HP Druckermotor, der in der PDV schon öfter gute Dienste geleistet hatte. Die Eigenschaften des Motors passen perfekt zu dem Aufzug. Es ist ein 12 V Gleichstrommotor, der sich bequem über eine H-Brücke ansteuern lässt. Er läuft nicht besonders schnell ist aber sehr kraftvoll.

An dem Motor ist eine "Lochscheibe" mit zwei Lichtschranken angebracht, die auf helle Striche auf der Scheibe reagieren. Eine Lichtschranke haben wir an den Interupteingang des PICs geschaltet. Die Zweite an den anderen Port. In dem Programm haben wir einen Algorithmus geschrieben, der die Lichtschrankensignale behandelt. Auf diese Weise stellen wir die Position, sowie die Drehrichtung des Aufzuges fest.

Die H-Brücke

Für die Steuerung des Motors steht eine H-Brücke zur Verfügung. Um die Funktionsweise und korrekte Beschaltung der H-Brücke herauszufinden, wurde zunächst ein passendes Datenblatt von der Seite ihres Herstellers heruntergeladen und studiert. Daraufhin wurde ein für unsere Bedürfnisse passender Schaltplan erstellt. Die für die Beschaltung notwendigen Zusatzkomponenten wurden uns freundlicherweise von Herrn Grothe zur Verfügung gestellt.

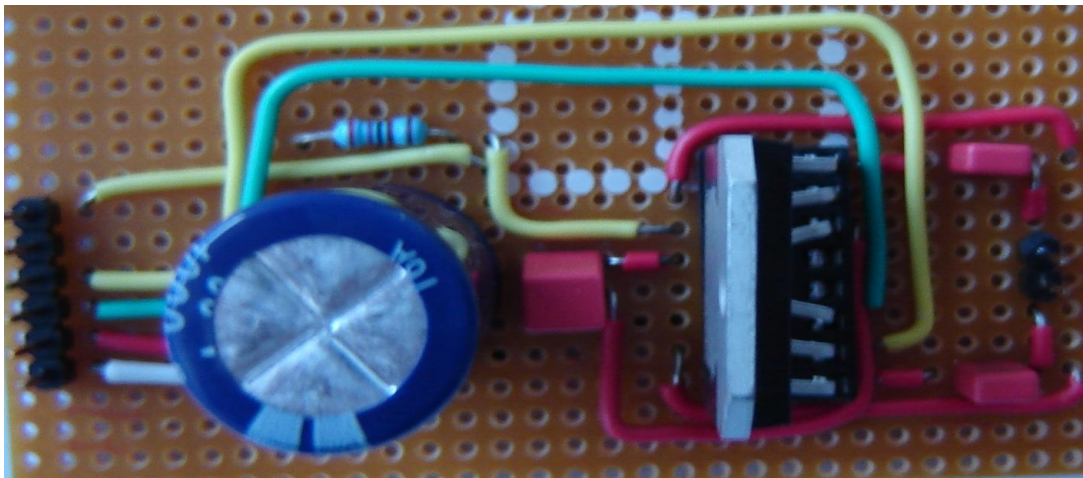
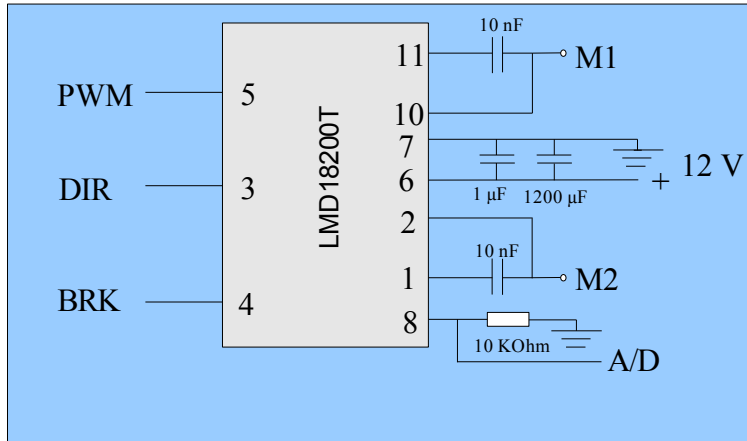


Abbildung 5: H-Brücke LMD18200T

Verwendete Komponenten für die Beschaltung der H-Brücke:

- 2 x Kondensator 10 nF
- 1 x Kondensator 1 μ F
- 1 x Kondensator 1200 μ F
- 1 x Widerstand 10 KOhm

Schaltplan für die Beschaltung der H-Brücke



| <i>Eingänge</i> | <i>Ausgänge</i> |
|--|-------------------------------|
| PWM (Pulsweitenmodulation) | M1, M2: Ausgänge für Motor |
| DIR (Drehrichtung des Motors) | A/D: Anschluss an A/D des PIC |
| BRK (Bremsen) | |
| 6 und 7: Erfordern 12V Versorgungsspannung | |

Software

Die Entwicklungsumgebung

Die 32-Bit-MPLAB-Entwicklungsumgebung (Version 7.10) arbeitet unter MS Windows und bietet die Möglichkeiten des Editierens und Debuggens des Quellcodes aus Assembler oder C. Die graphische Entwicklungsumgebung ist kostenfrei erhältlich und gestattet eine effiziente Code-Bearbeitung insbesondere einer schnellen Simulation. Für die Kompilierung des Codes verwenden wir den MPLAB- C18 Compiler.

Download MPLAB IDE V7.10: <http://www.microchip.com>

Funktionsweise der Anwendung

Das ganze Programm ist in C geschrieben und mit dem C18 Compiler kompiliert.

Das Programm ist wie folgt aufgebaut:

Programmstart -> PIC - Initialisierung -> Fahrstuhl in die Ausgangsstellung bringen(1.Stock)-> Betriebsphase(Endlosschleife)

Nachfolgend werden einzelne Aktionen kurz aufgelistet:

Betriebsphase:

- Obere Hilfs-LED leuchten lassen
- Taster abfragen
- Wenn betätigt das gewünschte Aktion ausführen

Tasteraktionen:

- Gewünschtes Stockwerk anfahren
- Not-Aus

Not-Aus-Aktion:

- PWM ist aus
- Brake ist an
- Aufzug in die Ausgangsstellung bringen(1.Stock)
- Stop-Taste noch mal drücken
- Betriebsphase

Fazit

Die Realisierung des Projektes brachte einige Probleme mit sich, denn der Einstieg in die PIC-Programmierung erinnert eher an Assembler-Programmierung, als an C.

Weiterhin bereitete unser mangelhaftes Elektrotechnik-Wissen, Probleme bei der Konzeption und Realisierung der Schaltungen.

Da wir am Anfang eine falsche Compiler-Version verwendeten, konnten wir einige wichtige Funktionen nicht benutzen, wie z. B. "openrb0int". Der Einsatz der Compiler-Version 3.0.4 brachte jedoch Abhilfe.

Da der Interrupt-Port RB0 an dem Demo-Board einen Kondensator für die Entprellung einer Taste hatte, konnte unser PIC nicht alle ankommenden Interrupts, die von der Lichtschranke ausgelöst werden, nicht schnell genug abarbeiten und verlor bei großer Drehgeschwindigkeit des Motors einige Interrupts. Das Problem wurde durch Abklemmen des Kondensators behoben.

Obwohl viele Hürden zu meistern waren, war das Projekt ein voller Erfolg. Die Aufgabenstellung

behandelte ein sehr interessantes und aktuelles Themengebiet, da sich die PIC-Programmierung immer größerer Beliebtheit erfreut, gerade dort, wo der Einsatz eines kompletten Computersystems aus Platzgründen nicht möglich ist.

Das Projekt war im Zeitlichen Rahmen gut umzusetzen.

Quellen

<http://www.microchip.com>

<http://www.datasheetcatalog.net>

<http://www.informatik.fh-wiesbaden.de/~linn/>

Anhang

Quellcode

```
/**
 * Komplette Aufzugssteuerung
 */

#include <p18cxxx.h>
#include <delays.h>
#include <pwm.h>
#include <timers.h>
#include <portb.h>
#include <stdio.h>

#define NOTAUS          PORTAbits.RA5
#define HELP            PORTAbits.RA4

#define TASTERA1        PORTDbits.RD7
#define TASTERA2        PORTDbits.RD6
#define TASTERA3        PORTDbits.RD5
#define TASTERA4        PORTDbits.RD4

#define TASTERB1        PORTDbits.RD0
#define TASTERB2        PORTDbits.RD2
#define TASTERB3        PORTDbits.RD1
#define TASTERB4        PORTDbits.RD3

#define LED1            PORTCbits.RC4
#define LED2            PORTCbits.RC5
#define LED3            PORTCbits.RC6
#define LED4            PORTCbits.RC7
#define LED_OBEN       PORTAbits.RA3
```

```
#define PWM                PORTCbits.RC2
#define brake              PORTCbits.RC1
#define richtung           PORTCbits.RC0

#define INTERRUPT_FLAG    INTCONbits.INT0IF
#define INTERRUPT_FLAG2  INTCON3bits.INT1IF
#define INTERRUPT_TIMER0 INTCONbits.TMR0IF
#define LS2                PORTCbits.RC3
#define EDGE               INTCON2bits.INTEDG0 // Eintrag im Register, der sagt auf welche flanke
                                                    // reagiert werden soll:
1:ansteigende 0:abfallende

void notAusFnkt();
void insertIntoWohin(int wohin);
void ledBlinkenLassen(int welcher);
void set_reset_FlipFlop(int welcher, int set_reset);
void LEDausFlipFlopReset();
void wohinArraySortieren();
void initFlipFlop();
void init(void);
void initAufzug();
void pwmOn(void);
void pwmOff(void);
void leseTaster();
void checkPosition();

void delay(void);
void DelayFor15ms(void);
void DelayFor5ms(void);
void wait();
void InterruptHandlerHigh(void);

static int global_power;
static long globalTicks;
static int globalState;
static int globalSchonDa;
static int globalInUse;

static int globalWohin[4]; // in dem werden die Zieletaggen festgehalten
static int globalZielStock;
static int globalStock1 = 0;
static int globalStock2 = 3677;
static int globalStock3 = 7354;
static int globalStock4 = 11031;

/**
 * Die Struktur realisiert ein R/S-FlipFlop
 * für jede einzelne Taste. Somit wird das
 * Prellen der Taster verhindert.
 */
```

```
struct FlipFlop
{
    int A1;
    int A2;
    int A3;
    int A4;
    int B1;
    int B2;
    int B3;
    int B4;
} flipFlop;

/**
 * Interrupt - Handler Assembler - Stup
 */
#pragma code InterruptVectorHigh = 0x8
void InterruptVectorHigh (void)
{
    _asm
        goto InterruptHandlerHigh
    _endasm
}

/**
 * Interrupt - Handler Implementierung für hohe Priorität
 */
#pragma code
#pragma interrupt InterruptHandlerHigh

void InterruptHandlerHigh(void)
{
    int tmp;
    if ( INTERRUPT_FLAG )
    {
        // wenn ansteigende Flanke von der 1.Lichschanke
        // muss man schauen ob die 2. logisch 0 oder 1 hat
        if( globalState == 0 )
        {
            if(LS2 == 1 )
            {
                globalTicks++;
            }else
            {
                globalTicks--;
            }
            globalState = 1;
            EDGE = 0; // beim naechsten Mal Fallende Flanke Abfangen
        }else
        {
            if(LS2 == 1 )
            {
                globalTicks--;
            }
        }
    }
}
```

```
        }else
        {
            globalTicks++;
        }
        globalState = 0;
        EDGE = 1; // beim naechsten Mal Ansteigende Flanke Abfangen
    }
    INTERRUPT_FLAG=0;
}
if(INTERRUPT_TIMER0)
{
    // Timer-Interrupt wird ausgelöst
    // beim überlauf von FFFF auf 0
    // wenn das der Faall ist die LED toglen
    // so wurde das Blinken einer LED realisiert
    LED_OBEN^=1;
    INTERRUPT_TIMER0 = 0;
}
}

/**
 * Initialisiert FlipFlop mit 0
 */
void initFlipFlop()
{
    flipFlop.A1 = 0;
    flipFlop.A2 = 0;
    flipFlop.A3 = 0;
    flipFlop.A4 = 0;

    flipFlop.B1 = 0;
    flipFlop.B2 = 0;
    flipFlop.B3 = 0;
    flipFlop.B4 = 0;
}

/**
 * Funktion zur Initialisierung des PICs
 */
void init(void)
{
    int i = 0;

    // Globale setzten
    globalTicks = 0;
    globalSchonDa = 1;
    globalZielStock = -1;
    globalState = 0; // am anfang auf ansteigende flanke reagieren

    for(i=0;i<4;i++)
        globalWohin[i] = -1;
}
```

```
// PWM, Richtung, Brake
TRISCbits.TRISC0 = 0;
TRISCbits.TRISC1 = 0;
TRISCbits.TRISC2 = 0;
richtung = 1;
brake = 1;

// LEDs
TRISCbits.TRISC7 = 0;
TRISCbits.TRISC6 = 0;
TRISCbits.TRISC5 = 0;
TRISCbits.TRISC4 = 0;
TRISAbits.TRISA3 = 0;
LED1 = 0;
LED2 = 0;
LED3 = 0;
LED4 = 0;
LED_OBEN = 1;

// Ganzes PortD als eingang
TRISDbits.TRISD7 = 1;
TRISDbits.TRISD6 = 1;
TRISDbits.TRISD5 = 1;
TRISDbits.TRISD4 = 1;
TRISDbits.TRISD3 = 1;
TRISDbits.TRISD2 = 1;
TRISDbits.TRISD1 = 1;
TRISDbits.TRISD0 = 1;
TRISAbits.TRISA5 = 1;
TRISAbits.TRISA4 = 1;

// Interupt und LS-Motor
TRISCbits.TRISC3 = 1;
RCONbits.IPEN = 1; // enable interrupt priority levels
INTCONbits.GIEH = 1; // Globale Interrupts freischalten
TRISBbits.TRISB0 = 1; // RB0 als Eingang

OpenRB0INT (PORTB_CHANGE_INT_ON & //enable the RB0/INT0 interrupt
            PORTB_PULLUPS_OFF & //configure the RB0 pin for input
            RISING_EDGE_INT); //trigger interrupt upon button

OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_8);

INTERUPT_FLAG=0; // Flag zurücksetzen
INTERUPT_TIMER0 = 0;

OpenXLCD( FOUR_BIT & LINES_5X7 );
}

void pwmOn(void)
{
```

```
    // PWM erzeugen
    OpenPWM1(0xff);
    SetDCPWM1(global_power);
}

void pwmOff(void)
{
    // PWM löschen
    ClosePWM1();
}

// prüft wo sich der Aufzug befindet
// wenn die Zieletage erreicht wurde
// stoppt den Aufzug
void checkPosition()
{
    int tmp;
    if( globalZielStock!=-1 )
    {
        tmp = globalTicks - globalZielStock*3677;
        if (tmp<0) tmp*=-1;
        if (tmp<=10)
        {
            globalSchonDa = 1;
            brake = 1;
            pwmOff();
        }
    }
}

// fügt dem Array "wohin" den Wert ein, der besagt
// wohin der Aufzug sich bewegen soll
void insertIntoWohin(int wohin)
{
    int i=1;
    int insert = -1;
    while(i<4)
    {
        // Wenn schon die Etage angefahren wird
        // braucht man keinen neuen Eintrag
        if((globalWohin[i] == wohin) || (globalZielStock==wohin) )
        {
            break;
        }
        // Nach dem freien Eintrag suchen
        if( globalWohin[i]==-1)
        {
            // die erste Stelle im Array wird
            // niemals direkt gesetzt, weil
            // in main werden die werte immer
            // nach links verschoben
        }
    }
}
```

```
                insert = i;
                break;
            }
            i++;
        }
        if(insert!=-1)
        {
            globalWohin[insert] = wohin;
        }
    }

// Veranlasst die gewünschte LED paar Mal zu blinken
// Betreffende LED bleibt dann leuchten
void ledBlinkenLassen(int welcher)
{
    int i;
    for( i = 0;i<4;i++)
    {
        switch (welcher)
        {
            case 1:
                LED1 = 0;
                DelayFor15ms();
                DelayFor15ms();
                DelayFor15ms();
                LED1 = 1;
                break;
            case 2:
                LED2 = 0;
                DelayFor15ms();
                DelayFor15ms();
                DelayFor15ms();

                LED2 = 1;
                break;
            case 3:
                LED3 = 0;
                DelayFor15ms();
                DelayFor15ms();
                DelayFor15ms();
                LED3 = 1;
                break;
            case 4:
                LED4 = 0;
                DelayFor15ms();
                DelayFor15ms();
                DelayFor15ms();
                LED4 = 1;
                break;
            default:
                //nichts
                break;
        }
    }
}
```

```
    }  
  }  
}  
  
// Setzen oder Rücksetzen von FlipFlops  
void set_reset_FlipFlop(int welcher, int set_reset)  
{  
    switch (welcher)  
    {  
        case 1:  
            flipFlop.A1 = set_reset;  
            break;  
        case 2:  
            flipFlop.A2 = set_reset;  
            break;  
        case 3:  
            flipFlop.A3 = set_reset;  
            break;  
        case 4:  
            flipFlop.A4 = set_reset;  
            break;  
        case 5:  
            flipFlop.B1 = set_reset;  
            break;  
        case 6:  
            flipFlop.B2 = set_reset;  
            break;  
        case 7:  
            flipFlop.B3 = set_reset;  
            break;  
        case 8:  
            flipFlop.B4 = set_reset;  
            break;  
        default:  
            //nichts  
            break;  
    }  
}  
  
// LED ausschalten FlipFlops zuruecksetzen  
void LEDausFlipFlopReset()  
{  
    switch (globalZielStock)  
    {  
        case 0:  
            LED1 = 0;  
            set_reset_FlipFlop(1,0);  
            set_reset_FlipFlop(5,0);  
            break;  
        case 1:  
            LED2 = 0;  
            set_reset_FlipFlop(2,0);  
            set_reset_FlipFlop(6,0);  
    }  
}
```

```
        break;
    case 2:
        LED3 = 0;
        set_reset_FlipFlop(3,0);
        set_reset_FlipFlop(7,0);
        break;
    case 3:
        LED4 = 0;
        set_reset_FlipFlop(4,0);
        set_reset_FlipFlop(8,0);
        break;
    default:
        // nichts tun
        break;
    }
}
// Liest die Taster aus und setzt wenn
// nötig die Flip-Flops. Außerdem füllt wohin-Array mit Werten
void leseTaster()
{
    if(!TASTERA1 && !flipFlop.A1 )
    {
        ledBlinkenLassen(1);
        insertIntoWohin(0);
        set_reset_FlipFlop(1,1);
    }
    if(!TASTERA2 && !flipFlop.A2 )
    {
        ledBlinkenLassen(2);
        insertIntoWohin(1);
        set_reset_FlipFlop(2,1);
    }
    if(!TASTERA3 && !flipFlop.A3 )
    {
        ledBlinkenLassen(3);
        insertIntoWohin(2);
        set_reset_FlipFlop(3,1);
    }
    if(!TASTERA4 && !flipFlop.A4 )
    {
        ledBlinkenLassen(4);
        insertIntoWohin(3);
        set_reset_FlipFlop(4,1);
    }

    if(!TASTERB1 && !flipFlop.B1 )
    {
        ledBlinkenLassen(1);
        insertIntoWohin(0);
        set_reset_FlipFlop(5,1);
    }
    if(!TASTERB2 && !flipFlop.B2 )
```

```
        {
            ledBlinkenLassen(2);
            insertIntoWohin(1);
            set_reset_FlipFlop(6,1);
        }
        if(!TASTERB3 && !flipFlop.B3 )
        {
            ledBlinkenLassen(3);
            insertIntoWohin(2);
            set_reset_FlipFlop(7,1);
        }
        if(!TASTERB4 && !flipFlop.B4 )
        {
            ledBlinkenLassen(4);
            insertIntoWohin(3);
            set_reset_FlipFlop(8,1);
        }

        if(!NOTAUS)
        {
            notAusFnkt();
        }
    }

/**
 * Diese Funktion wird aufgerufen, wenn der Not-Aus-
 * Taster betätigt wurde. Sie stoppt den Motor, schaltet
 * alle LEDs aus und initialisiert alles neu
 * Der Aufzug muss neu eingestellt werden
 */
void notAusFnkt()
{
    brake = 1;
    pwmOff();
    LED1 = 0;
    LED2 = 0;
    LED3 = 0;
    LED4 = 0;
    initFlipFlop();
    init();
    initAufzug();
}

/**
 * Sortiert das Array, wo die Zieletagen gespeichert sind,
 * auf die Weise, dass alle werte nach vorne rutschen, wenn
 * der erste Wert -1 ist
 */
void wohinArraySortieren()
{
    int i = 0;
    for( i=0;i<3;i++)
```

```
        globalWohin[i] = globalWohin[i+1];
        globalWohin[3] = -1;
    }

/**
 * Warte für 10000 Zyklen
 */
void delay (void)
{
    int i;

    for (i = 0; i < 10000; i++)
        ;
}

void DelayFor15ms(void) // minimum 15ms
{
    Delay100TCYx(0xA0); // 100TCY * 160
    return;
}

void DelayFor5ms(void) // minimum 5ms
{
    Delay100TCYx(0x36); // 100TCY * 54
    return;
}

/**
 * Warte für 20*15ms
 */
void wait()
{
    int i;
    for(i=0;i<20;i++)
        DelayFor15ms();
}

/**
 * initialisiert den Fahrstuhl
 * wartet bis die Taste Help gedrückt wird
 * dann sagt man ob der Aufzug hoch oder runter fahren soll
 * mit dem Stop-Button beendet man die Funktion
 */
void initAufzug()
{
    int i;
    int j;
    while(HELP)
    {
    }
    while(NOTAUS)
    {
    }
}
```

```
        // runter
        if(!TASTERA1)
        {
            brake = 0;
            richtung = 1;
            global_power = 750;
            pwmOn();
            DelayFor15ms();
            DelayFor15ms();
            pwmOff();
        }
        //hoch
        if(!TASTERA4)
        {
            brake = 0;
            richtung = 0;
            global_power = 800;
            pwmOn();
            DelayFor15ms();
            DelayFor15ms();
            pwmOff();
        }

        brake = 1;
        pwmOff();
    }
    pwmOff();
    brake = 1;
    globalTicks = 0;
    wait();

    for(j=0;j<3;j++)
    {
        for(i=1;i<=4;i++) ledBlinkenLassen(i);
        wait();
        LED1 = LED2 = LED3 = LED4 = 0;
    }
    wait();
}

void main (void)
{
    int diff;
    char buf[100];
    init();
    initFlipFlop();
    pwmOff();
    initAufzug();
    while (1)
    {
        checkPosition();
        leseTaster();
    }
}
```

```
if( globalSchonDa == 1 )
{
    // prüfen ob ein Eintrag gibt
    if(globalWohin[0]!=-1)
    {
        // ein wenig warten
        wait();
        globalZielStock = globalWohin[0];
        // prüfen, ob Aufzug schon da steht
        // aber der zaehler hat kleine Abweichung
        // das kann passieren, wenn Taster mehrmals
        // bestaetigt wurde
        // obwohl der Fahrstuhl schon da ist
        diff = globalTicks - globalWohin[0]*3677;
        // zuerst das vorzeichen behandeln
        if (diff<0) diff*=-1;

        if( diff>=10)
        {
            // Richtung bestimmen
            if( (globalTicks - globalWohin[0]*3677)<0)
            {
                // hoch
                global_power = 800;
                richtung = 0;
            }else
            {
                //runter
                global_power = 650;
                richtung = 1;
            }
            brake = 0;
            pwmOn();
            globalSchonDa = 0;
        }
        globalWohin[0] = -1;
    }else
    {
        wohinArraySortieren();
        brake = 1;
        pwmOff();
        // LED ausschalten FlipFlops zuruecksetzen
        LEDausFlipFlopReset();
        globalZielStock = -1;
    }
}
}
```