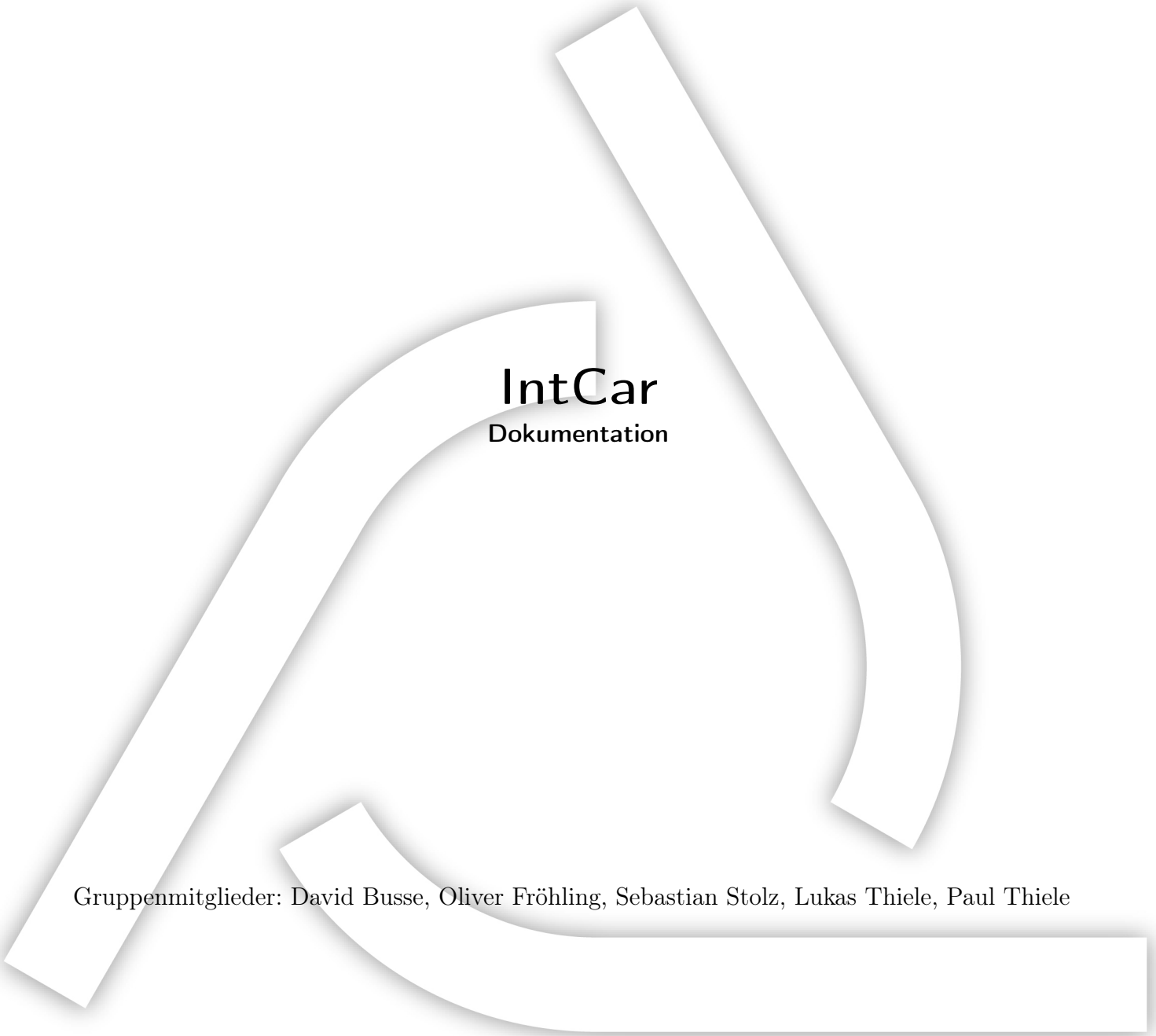
Städt. Louise-Schroeder-Gymnasium München Februar 2011



IntCar

Dokumentation

Gruppenmitglieder: David Busse, Oliver Fröhling, Sebastian Stolz, Lukas Thiele, Paul Thiele

Inhaltsverzeichnis

1 Einleitung

1.1 Die Gruppe

Unsere Gruppe besteht aus 5 Schülern des städtischen Louise-Schroeders-Gymnasium in München. Vier der Schüler besuchen die Oberstufe und einer die Mittelstufe. Auf der Suche nach neuen interessanten Herausforderungen haben wir schon erfolgreich letztes Jahr am EADS-Astrium-Wettbewerb teilgenommen.
Nach langem überlegen entschlossen wir uns auch dieses Jahr an einem Wettbewerb teilzunehmen. Wir entschieden uns nach längeren Diskussionen dafür eine Anwendung mit der Energiespar-MCU EFM32 von Energy Micro zu entwerfen.

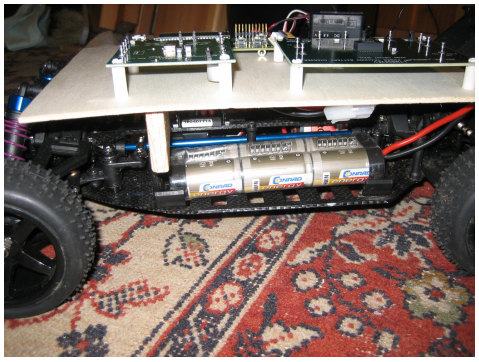


Abbildung 1.1: Nahaufnahme der linken Seite

Am Anfang das Projektes standen wir vor einigen Hindernissen. So hatten wir kaum bis keinerlei Erfahrung mit Schaltplänen oder hardwarenaher Programmierung, in C sondern nur mit PHP, Mysql, HTML auf unserem Schulserver. Da wir uns aber sicher waren, dass wir programmiertechnische Herausforderungen sehr gut lösen könnten, wählten wir trotzdem ein programmierlastiges Projekt.

1.2 Idee

Nach längeren Überlegen was wir mit dem EFM32 anfangen könnten, haben wir uns auf ein Modellauto geeinigt, das einerseits wegen unterstützendem Fahren die Rechenleistung und andererseits wegen der Versorgung über Solarzellen die Energieeffizienz des EFM32 benötigt.
Weiterhin ist eine Steuerung über Bluetooth mittels einem Handy/Smartphone/Notebook geplant. Dazu soll Java für die Steuersoftware benutzt werden, um Plattformenunabhängigkeit zu gewährleisten. Jedoch konnten wegen Zeitknappheit Möglichkeiten der Steuerung nur angedacht werden. Beispiele wären eine Steuerung durch Neigen bzw. Kippen, durch die Verwendung eines virtuellen Joystick oder durch Abfahren einer gezeichneten Route.

1.3 Verwendungszweck

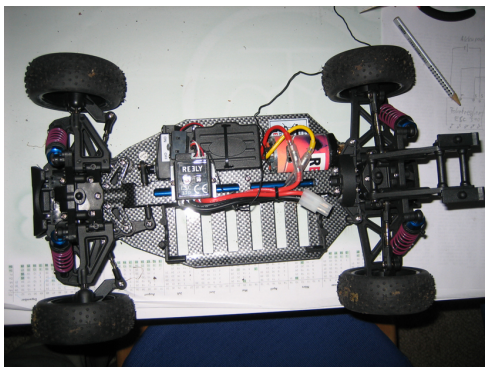


Abbildung 1.2: Grundgerüst des Fahrzeugs: Batterie, Motor und Servo

Langfristig soll der EFM32 dazu dienen, dem Auto eine „Intelligenz“ zu verleihen, sodass es sich autonomer bewegen kann, als es mit einem klassischen Fernsteuerauto möglich ist. Es ist vergleichbar mit dem durch einen Computer unterstützten Fahren in modernen „richtigen“ Autos, nur dass der Fahrer über Fernsteuerung lenkt.
Dabei ist die Effizienz des EFM32 wichtig, um einen möglichst großen Teil der Solarzellen- und Akkuleistung für den Antrieb nutzen zu können und das Gewicht gering zu halten, ohne Abstriche bei der Rechenleistung machen zu müssen. So sollen zukünftig weitere Sensoren angebaut und ausgewertet werden, um mehr Autonomie zu erreichen. Geplant sind eigenständiges Fahren auf einer vorgegebenen Karte und effektives Reagieren auf die Umwelt.
Weitblickend kann das Modell tatsächlich als eine Simulation für ein „richtiges“ Auto, das den Fahrer konstruktiv unterstützt und menschliche Fehler vermeiden kann.

2 Das Modellauto

2.1 Schaltpläne

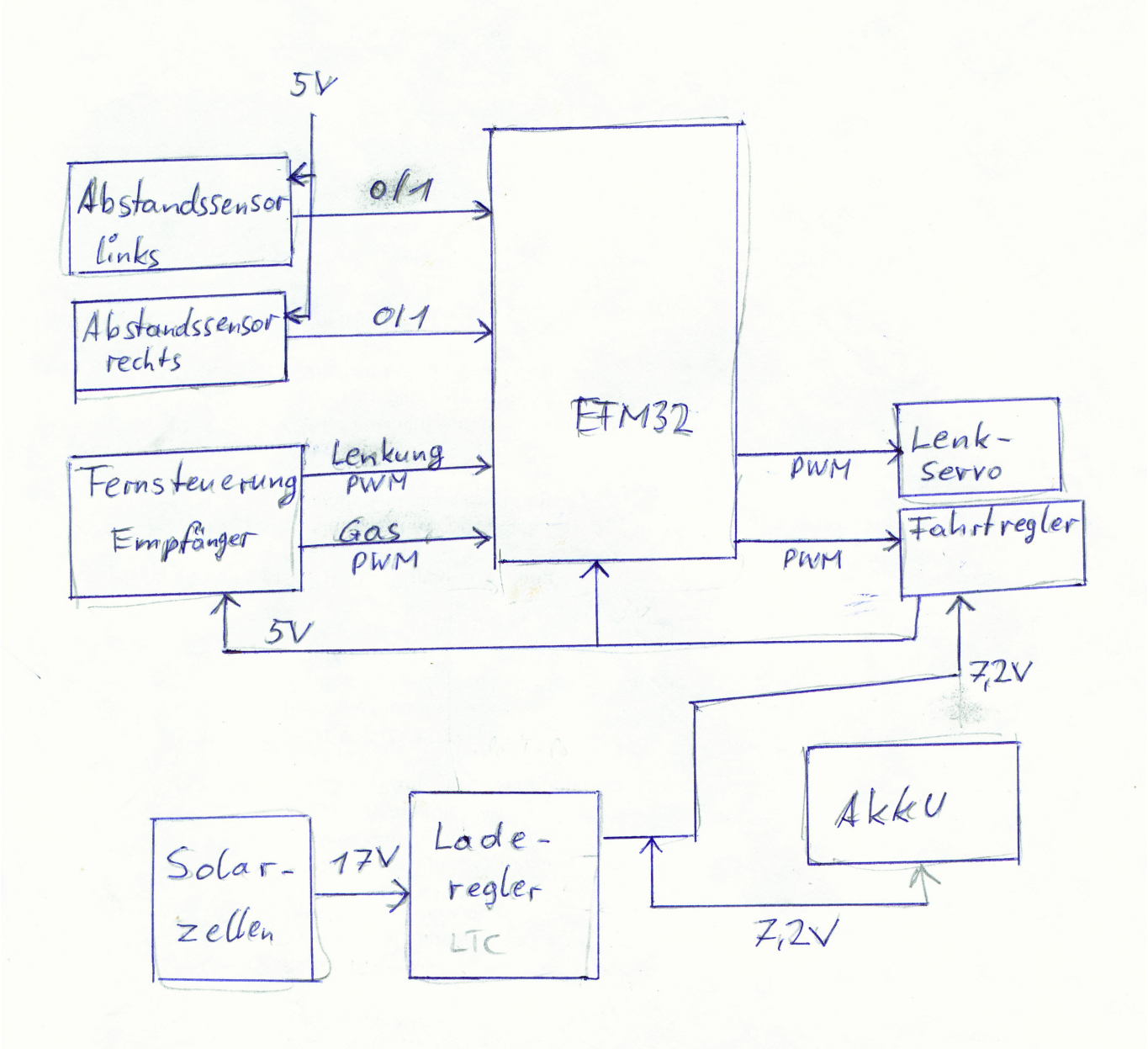


Abbildung 2.1: aktueller Energie-Schaltplan

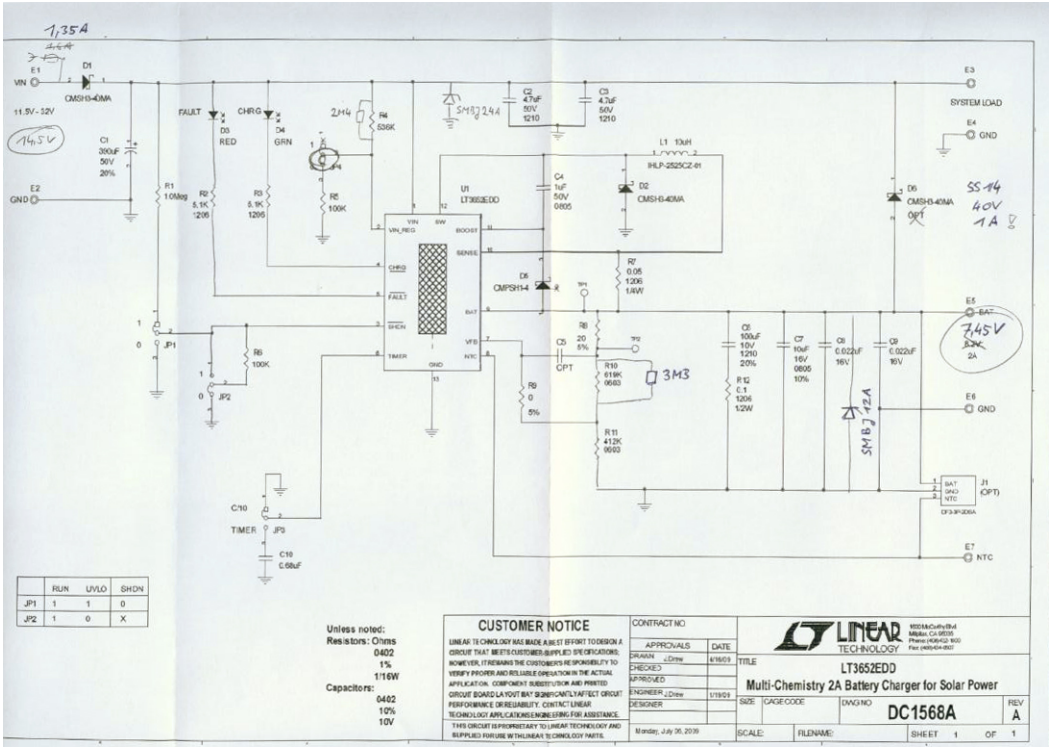


Abbildung 2.2: Schaltplan des Ladereglers

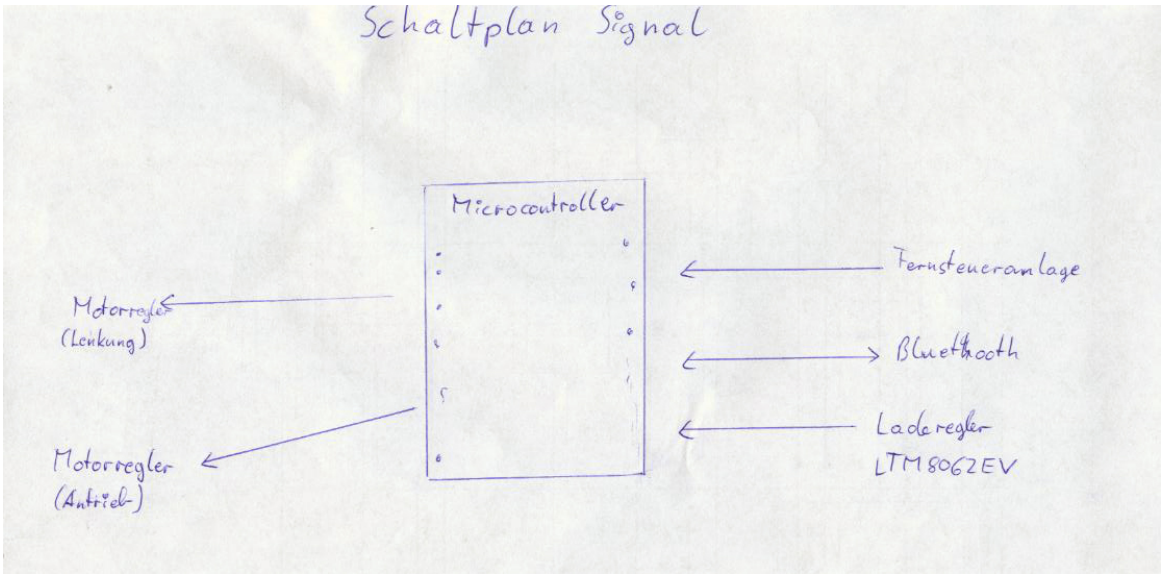


Abbildung 2.3: Schaltplan zu den Informationswegen des Modellautos

2.2 Daten

2.2.1 Kit

EFM32 Gecko Starter Kit:
http://www.energymicro.com/tools/efm32-gecko-starter-kit
Der Chip ist ein G890F128ES von Energy micro, auf dem Starter Kit EFM-32-G8XX-STK. Er bildet die Grundlage unseres IntCars und wertet alle Informationen aus. Der Code der ausgeführt wird, ist im Anhang zu finden.

2.2.2 Sensoren

GP2Y0D02YK:
http://www.conrad.de/ce/de/product/185350/DISTANZ-SENSOR-GP2Y0D02YK
Dieser Chip meldet, ob ein Hindernis in 80cm Entfernung vorhanden ist oder nicht dies meldet er an den Chip weiter, welcher das Ergebnis auswertet und verwendet. Die Sensoren sind einmal links und rechts am Auto zu finden.

2.2.3 Solarzellen

Die Solarzellen besitzen eine Leerlaufspannung von ca. 0,5V pro Zelle. Immer 34 Zellen wurden in Serie geschalten, drei dieser Serien wurden parallel geschalten. Insgesamt sind also 102 Zellen verbaut.
Die Solarenergie stand im Mittelpunkt des Schülerwettbewerbs "Powered by Space", den das Referat für Bildung und Sport und das Referat für Gesundheit und Umwelt gemeinsam mit dem Bildungszentrum für Solartechnik und dem Raumfahrtunternehmen EADS Astrium ausrichtete. Mit ihnen ist es möglich die Akkulaufzeit zu verlängern, um wie viel sie verlängert wird hängt von mehreren Faktoren ab:

1. Fahrweise des Autos
2. Die Leistung, die die Solarzelle aufnehmen kann. Das hängt von ihrem Funktionsprinzip, dem Wetter und weiteren Faktoren ab.

2.2.4 Ladechip

LT3652EDD:
http://www.linear.com/demo/dc1568a
Der Chip wurde auf einem Test-Kit montiert, welches leicht modifiziert wurde. Es sucht den Maximum Power-Point von den Solarzellen und regelt die Spannung die Spannung auf 7,2 V

2.2.5 Datenblatt: Buggy Ranger von Reely und Sensoren

Fahrzeug	
Maßstab	1:10
Betriebsspannung	Akkupack mit 7.2 V (6 Zellen)
Antrieb	Elektromotor Typ 540 Allrad-Antrieb über Kardanwelle Kugelgelagerter Antrieb Differenzial in Vorder- und Hinterachse
Federung	Einzelradaufhängung, mit Spiralfedern/Stoßdämpfern
Abmessungen (L x B x H)	400 x 240 x 160mm
Reifen-Abmessungen (B x Ø)	34 x 87mm
Radstand	268mm
Gewicht (ohne Akku)	1370g
Sender	
Betriebsspannung	9.6 - 12V (8 Zellen AA/Mignon)
Sendefrequenz	27MHz
Kanäle	2
Sensoren	
Typ	GP2Y0D02YK
Anzahl	2
Messbereich	(Erfassungsbereich) 20 - 150 cm
Betriebsspannung	5 V/DC

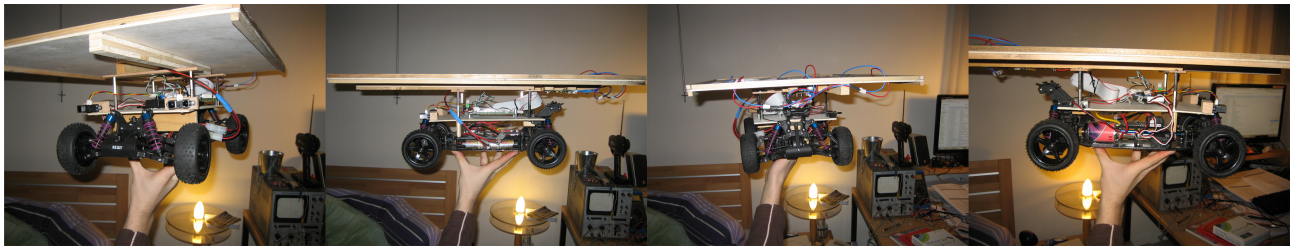


Abbildung 2.4: Das Modellauto von allen Seiten

3 Anhang

3.1 Quellcode

Listing 3.1: Quellcode

```
1   #include <stdint.h>
2   #include <stdbool.h>
3   #include <stdio.h>
4   #include <string.h>
5
6   #include "efm32.h"
7   #include "efm32_chip.h"
8   #include "efm32_system.h"
9   #include "vddcheck.h"
10  #include "segmentlcd.h"
11  #include "efm32_cmu.h"
12  #include "efm32_emu.h"
13  #include "efm32_gpio.h"
14  #include "efm32_prs.h"
15  #include "efm32_timer.h"
16
17  bool vboost = false;
18
19  uint32_t outputPWM_ST = 15;
20  uint32_t outputPWM_ACC = 15;
21  /* TOP_ST (Steuerung) reset value is 0xFFFF so it doesn't need
22     to be written for this example */
23  #define TOP_ST 0xFFFF
24  /* TOP_ACC (Beschleunigung) reset value is 0xFFFF so it doesn't need
25     to be written for this example */
26  #define TOP_ACC 0xFFFF
27
28  /* Define TOP_PWM value */
29  #define TOP_PWM_ST 1000
30  #define TOP_PWM_ACC 1000
31
32  /* 54687,5 Hz -> 14Mhz (clock frequency) / 256 (prescaler) */
33  #define TIMER_FREQ 54688
34
35  /* 54687,5 Hz -> 14Mhz (clock frequency) / 256 (prescaler) */
36  #define TIMER_FREQ_PWM 54688
37
38  #define FILTERCONST 3
39
40  long int count = 0, totalTime_ST = 150, totalTime_ACC = 150;
41  char totalTimeString[7], overFlowString[8];
42  long int totalTime_ST_read, totalTime_ACC_read;
43  long int totalTime_ST_raw, totalTime_ACC_raw;
44
45  /* ========   MAIN   ========*/
46  int main(void)
47  {
48    CHIP_Init();
49    SystemCoreClockUpdate();
50    VddCheckInit();
51    if (VddCheckLowVoltage(2.9))
52    {
53      vboost = true;
54    }
55    else
56    {
57      vboost = false;
58    }
59
60    /* ==========   PULSE-WIDTH CAPTURE   ========*/
61
62    CMU_ClockEnable(cmuClock_GPIO, true);
63    CMU_ClockEnable(cmuClock_TIMER1, true);
64    CMU_ClockEnable(cmuClock_PRS, true);
65
66    GPIO_PinModeSet(gpioPortC, 0, gpioModePushPull, 1);
67    GPIO_PinModeSet(gpioPortC, 1, gpioModePushPull, 1);
68    GPIO_PinModeSet(gpioPortC, 2, gpioModePushPull, 1);
69    GPIO_PinModeSet(gpioPortC, 3, gpioModePushPull, 1);
70
71    /* Configure PB9 as an input for PB0 button with filter and pull-up (dout = 1)*/
72    GPIO_PinModeSet(gpioPortB, 9, gpioModeInputPullFilter, 1);
73    /* Configure PB10 as an input for PB1 button with filter and pull-up (dout = 1)*/
74    GPIO_PinModeSet(gpioPortB, 10, gpioModeInputPullFilter, 1);
75    /* Select PB9 as external interrupt source*/
76    GPIO_IntConfig(gpioPortB, 9, false, false, false);
77    /* Select PB10 as external interrupt source*/
78    GPIO_IntConfig(gpioPortB, 10, false, false, false);
79
```

```
80      /* Enable PRS sense on GPIO and disable interrupt sense */
81      GPIO_InputSenseSet(GPIO_INSENSE_PRS, _GPIO_INSENSE_RESETVALUE);
82
83      /* Select GPIO as source and GPIOPIN9 as signal for PRS channel 0 */
84      PRS_SourceSignalSet(0, PRS_CH_CTRL_SOURCESEL_GPIOH, PRS_CH_CTRL_SIGSEL_GPIOPIN9, prsEdgeOff);
85      /* Select GPIO as source and GPIOPIN10 as signal for PRS channel 1 */
86      PRS_SourceSignalSet(1, PRS_CH_CTRL_SOURCESEL_GPIOH, PRS_CH_CTRL_SIGSEL_GPIOPIN10, prsEdgeOff);
87
88      /* Select CC channel parameters */
89      TIMER_InitCC_TypeDef timerCCInit_ST =
90      {
91        .eventCtrl  = timerEventEveryEdge,
92        .edge       = timerEdgeFalling,
93        .prsSel     = timerPRSSELCh0,
94        .cufoa      = timerOutputActionNone,
95        .cofoa      = timerOutputActionNone,
96        .cmoa       = timerOutputActionNone,
97        .mode       = timerCCModeCapture,
98        .filter     = true,
99        .prsInput   = true,
100       .coist      = false,
101       .outInvert  = false,
102     };
103
104     /* Configure CC channel 0 */
105     TIMER_InitCC(TIMER1, 0, &timerCCInit_ST);
106
107     /* Select CC channel parameters */
108     TIMER_InitCC_TypeDef timerCCInit_ACC =
109     {
110       .eventCtrl  = timerEventEveryEdge,
111       .edge       = timerEdgeFalling,
112       .prsSel     = timerPRSSELCh1,
113       .cufoa      = timerOutputActionNone,
114       .cofoa      = timerOutputActionNone,
115       .cmoa       = timerOutputActionNone,
116       .mode       = timerCCModeCapture,
117       .filter     = true,
118       .prsInput   = true,
119       .coist      = false,
120       .outInvert  = false,
121     };
122
123     /* Configure CC channel 1 */
124     TIMER_InitCC(TIMER1, 1, &timerCCInit_ACC);
125
126     /* Select timer parameters */
127     TIMER_Init_TypeDef timerInit =
128     {
129       .enable     = false,
130       .debugRun   = true,
131       .prescale   = timerPrescale256,
132       .clkSel     = timerClkSelHFPerClk,
133       .fallAction = timerInputActionNone,
134       .riseAction = timerInputActionReloadStart,
135       .mode       = timerModeUp,
136       .dmaClrAct  = false,
137       .quadModeX4 = false,
138       .oneShot    = false,
139       .sync       = false,
140     };
141
142     /* Enable overflow and CC0 and CC1 interrupt */
143     TIMER_IntEnable(TIMER1, TIMER_IF_OF | TIMER_IF_CC0 | TIMER_IF_CC1); // CC1 evtl. rausnehmen???
144
145     /* Enable TIMER1 interrupt vector in NVIC */
146     NVIC_EnableIRQ(TIMER1_IRQn);
147
148     /* Configure timer */
149     TIMER_Init(TIMER1, &timerInit);
150
151
152     /* ==========  PWM  ========*/
153     /* Enable clock for TIMER0 module */
154     CMU_ClockEnable(cmuClock_TIMER0, true);
155
156     /* Set CC0 location 3 pin (PD1) as output */
157     GPIO_PinModeSet(gpioPortD, 1, gpioModePushPull, 0);
158     /* Set CC1 location 3 pin (PD2) as output */
159     GPIO_PinModeSet(gpioPortD, 2, gpioModePushPull, 0);
160
161     /* Select CC channel parameters */
162     TIMER_InitCC_TypeDef timerCCInit_PWM_ST =
163     {
164       .eventCtrl  = timerEventEveryEdge,
165       .edge       = timerEdgeBoth,
166       .prsSel     = timerPRSSELCh0,
167       .cufoa      = timerOutputActionNone,
168       .cofoa      = timerOutputActionNone,
169       .cmoa       = timerOutputActionToggle,
170       .mode       = timerCCModePWM,
171       .filter     = false,
172       .prsInput   = false,
173       .coist      = false,
174       .outInvert  = false,
175     };
176
177     /* Configure CC channel 0 */
178     TIMER_InitCC(TIMER0, 0, &timerCCInit_PWM_ST);
179     /* Route CC0 to location 3 (PD1) and enable pin */
180     TIMER0->ROUTE |= (TIMER_ROUTE_CC0PEN | TIMER_ROUTE_LOCATION_LOC3);
```

```
181
182       /* Select CC channel parameters */
183       TIMER_InitCC_TypeDef timerCCInit_PWM_ACC =
184       {
185         .eventCtrl  = timerEventEveryEdge,
186         .edge       = timerEdgeBoth,
187         .prsSel     = timerPRSSELCh1,
188         .cufoa      = timerOutputActionNone,
189         .cofoa      = timerOutputActionNone,
190         .cmoa       = timerOutputActionToggle,
191         .mode       = timerCCModePWM,
192         .filter     = false,
193         .prsInput   = false,
194         .coist      = false,
195         .outInvert  = false,
196       };
197
198       /* Configure CC channel 1 */
199       TIMER_InitCC(TIMER0, 1, &timerCCInit_PWM_ACC);
200       /* Route CC1 to location 3 (PD2) and enable pin */
201       TIMER0->ROUTE |= (TIMER_ROUTE_CC1PEN | TIMER_ROUTE_LOCATION_LOC3);
202
203       /* Set Top_PWM_ST Value */
204       TIMER_TopSet(TIMER0, TOP_PWM_ST);
205
206       /* Set compare value starting at 0 - it will be incremented in the interrupt handler */
207       TIMER_CompareBufSet(TIMER0, 0, 0);
208       TIMER_CompareBufSet(TIMER0, 1, 0);
209
210       /* Select timer parameters */
211       TIMER_Init_TypeDef timerInit_PWM =
212       {
213         .enable     = true,
214         .debugRun   = true,
215         .prescale   = timerPrescale256,
216         .clkSel     = timerClkSelHFPerClk,
217         .fallAction = timerInputActionNone,
218         .riseAction = timerInputActionNone,
219         .mode       = timerModeUp,
220         .dmaClrAct  = false,
221         .quadModeX4 = false,
222         .oneShot    = false,
223         .sync       = false,
224       };
225
226       /* Enable overflow interrupt */
227       TIMER_IntEnable(TIMER0, TIMER_IF_OF);
228
229       /* Enable TIMER0 interrupt vector in NVIC */
230       NVIC_EnableIRQ(TIMER0_IRQn);
231
232       /* Configure timer */
233       TIMER_Init(TIMER0, &timerInit_PWM);
234
235       /* ==========   DISTANCE SENSORS   ======= */
236
237       /* Enable GPIO in CMU */
238       CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO; // remove?
239
240       /* Configure PB11 as an input with filter enabled (out = 1)*/
241       GPIO_PinModeSet(gpioPortB, 11, gpioModeInput, 1);
242
243       /* Configure PB12 as an input with filter enabled (out = 1)*/
244       GPIO_PinModeSet(gpioPortB, 12, gpioModeInput, 1);
245
246
247
248
249       /* ==========   LCD   ======= */
250
251       SegmentLCD_Init(vboost);
252
253       SegmentLCD_Write("LSG");
254
255       while(1)
256       {
257         /* Go to EM1 */
258         EMU_EnterEM1();
259       }
260     }
261
262     /**************************************************************************//**
263     * @brief TIMER1_IRQHandler
264     * Interrupt Service Routine TIMER1 Interrupt Line
265     *****************************************************************************/
266     void TIMER1_IRQHandler(void)
267     {
268       uint16_t intFlags = TIMER_IntGet(TIMER1);
269
270       TIMER_IntClear(TIMER1, TIMER_IF_OF | TIMER_IF_CC0);
271       TIMER_IntClear(TIMER1, TIMER_IF_OF | TIMER_IF_CC1);
272
273       /* Overflow interrupt occured */
274       if(intFlags & TIMER_IF_OF)
275       {
276         // shouldn't occur
277         // do nothing
278       }
279
280       /* Capture interrupt occured on CC0 */
281       if(intFlags & TIMER_IF_CC0)
```

```
282       {
283         /* Calculate total time of signal */
284         totalTime_ST_read = TIMER_CaptureGet(TIMER1, 0);
285         /* Multiply by 100000 to avoid floats */
286         totalTime_ST_raw = (totalTime_ST_read * 100000) / TIMER_FREQ;
287         if ((totalTime_ST_raw < 90) || (totalTime_ST_raw > 210))
288           totalTime_ST_raw = 150;
289         totalTime_ST = (totalTime_ST * FILTERCONST + totalTime_ST_raw ) / (FILTERCONST + 1);
290
291         outputPWM_ST = totalTime_ST;
292        }
293       /* Capture interrupt occured on CC1 */
294        if(intFlags & TIMER_IF_CC1)
295       {
296         /* Calculate total time of signal */
297         totalTime_ACC_read = TIMER_CaptureGet(TIMER1, 1);
298         /* Multiply by 100000 to avoid floats */
299         totalTime_ACC_raw = (totalTime_ACC_read * 100000) / TIMER_FREQ;
300         if ((totalTime_ACC_raw < 90) || (totalTime_ACC_raw > 210))
301           totalTime_ACC_raw = 150;
302         totalTime_ACC = (totalTime_ACC * FILTERCONST + totalTime_ACC_raw ) / (FILTERCONST + 1);
303
304         outputPWM_ACC = totalTime_ACC;
305        }
306     }
307
308     /**************************************************************************//**
309     * @brief TIMER0_IRQHandler
310     * Interrupt Service Routine TIMER0 Interrupt Line
311     *****************************************************************************/
312     void TIMER0_IRQHandler(void)
313     {
314       /* clear flag for TIMER0 overflow interrupt */
315       TIMER_IntClear(TIMER0, TIMER_IF_OF);
316       if(!GPIO_PinInGet(gpioPortB, 11)) {
317         outputPWM_ST = 15;
318         outputPWM_ACC = 15;
319         GPIO_PinOutSet(gpioPortC,2);
320       } else {
321         GPIO_PinOutClear(gpioPortC,2);
322       }
323       if(!GPIO_PinInGet(gpioPortB, 12)) {
324         outputPWM_ST = 15;
325         outputPWM_ACC = 15;
326         GPIO_PinOutSet(gpioPortC,3);
327       } else {
328         GPIO_PinOutClear(gpioPortC,3);
329       }
330       TIMER_CompareBufSet(TIMER0, 0, (outputPWM_ST*TIMER_FREQ_PWM)/100000);
331       TIMER_CompareBufSet(TIMER0, 1, (outputPWM_ACC*TIMER_FREQ_PWM)/100000);
332       sprintf(totalTimeString, "%3d %3d", totalTime_ACC, totalTime_ST);
333       SegmentLCD_Write(totalTimeString);
334
335     }
```