Embedded Systems 1 - Dokumentation

Kevin Fritz

5. April 2017, Aalen

Inhaltsverzeichnis

Einleitung **3**

1 Getting Started **4**
1.1 Aufgabenstellung 4
1.2 Lösung 4

2 Rechenleistung **10**
2.1 Aufgabenstellung 10
2.2 Lösung 10

3 IO-Bibliothek **15**
3.1 Aufgabenstellung 15
3.2 Lösung 15

4 Mitschrift aus Vorlesung **16**
4.1 Vorlesung 04.04.2017 - IO-Bibliothek 16
4.2 Lösung 16

5 Kapitelbezeichnung **17**
5.1 Aufgabenstellung 17
5.2 Lösung 17

6 Quellcode **18**
6.1 C-Code 18

Einleitung

Einleitung muss noch verfasst werden.

1 Getting Started

1.1 Aufgabenstellung

1. Nehmen Sie das Programm „HelloWorld2“ in Betrieb.
2. Entfernen Sie die Verzögerungsfunktion und messen Sie die Frequenz, mit der die LED angesteuert wird. Überprüfen Sie das Ergebnis durch Analyse des generierten Assembler-Codes. Wie groß ist die Rechenleistung in MIPS?
3. Erhöhen Sie die CPU-Frequenz auf den maximal möglichen Wert. Weisen Sie durch eine Messung nach, dass die CPU-Frequenz tatsächlich erhöht wurde. Wie groß ist die Rechenleistung in MIPS?

1.2 Lösung

1. Programm „HelloWorld2“ in Betrieb nehmen. Der Programmcode ist in Listing 1 zu sehen.
2. Die Verzögerungsfunktion wurde auskommentiert. Der Oszi-Aufnahme aus Abbildung 1 kann entnommen werden, dass die Zeitdauer um einen Port ein- bzw. auszuschalten jeweils ungefähr $5,75us$ beträgt. Daraus ergibt sich eine Frequenz von $f = \frac{1}{T} = \frac{1}{5,75us} \approx 174kHz$. Aus Listing 2 geht hervor das zum Toggeln der LED 16 Assemblerbefehle benötigt werden.
$$MIPS = \frac{16}{5,75us} * 10^{-6} \approx 2,783MIPS \quad (1)$$
Die Einheit MIPS gibt an, wie viele Maschinenbefehle (Instruktionen) ein Mikroprozessor pro Sekunde ausführen kann. 1 MIPS bedeutet, er kann eine Million Maschinenbefehle pro Sekunde ausführen. MIPS ist eine ungenaue Einheit, da verschiedene Assemblerbefehle verschieden viel Zeit benötigen.
3. Um die CPU-Frequenz auf den Maximalen Wert zu erhöhen wird die auf dem Board verbaute PLL verwendet. Die Parameter zur Konfiguration der PLL sind dem Datenblatt (Abbildung 2) zu entnehmen. Der Wertebereich der PLL Parameter kann Abbildung 3 entnommen werden. Die Parameter wurden (mit einem Excel Sheet, Abbildung 4) so ausgelegt, dass sich eine Taktfrequenz F_{OSC} von $140MHz$ ergibt. Aus dem Oszillator Modul (online zu finden bei mikrochip) kann eine Code-Sequenz entnommen werden wie die jeweiligen PLL-Parameter zu setzen sind. Der Ausschnitt aus dem Datenblatt wurde für unsere Zwecke angepasst (Listing 3).
Nach Konfigurieren der PLL wurde wieder die Zeitdauer zum toggeln der LED gemessen ($302ns$ für 16 Assembler-Befehle), hieraus ergibt sich eine Rechenleistung von:
$$MIPS = \frac{16}{302ns} * 10^{-6} \approx 52,980MIPS \quad (2)$$
Setzt man die ausgerechneten MIPS ins Verhältnis, kommt man zu dem Entschluss das die gemessenen Werte plausibel sind, da: $2,783 * \frac{140}{7,37} \approx 52,867$.

```
// Check for Project Settings
#ifndef __dsPIC33EP512MU810__
#error "Wrong Controller"
#endif
#include <xc.h>    //Include appropriate controller specific
   headers
#include <stdint.h>//Standard typedefs
// Oscillator Configuration
_FOSCSEL(FNOSC_FRC); //Initial Oscillator:  Internal Fast RC
_FOSC(POSCMD_NONE);  //Primary Oscillator disabled (not used)

/* Substitute for stdlib.h */
#define EXIT_SUCCESS    0
#define EXIT_FAILURE    1

/* Hardware  */
#define _LED200 LATBbits.LATB8

void delay_ms(uint16_t u16milliseconds){
        uint16_t ui16_i=0;
        while(u16milliseconds){
                for (ui16_i=0;ui16_i<331;ui16_i++){//1 ms delay
                        __asm__ volatile("nop \n\t"
                        "nop \n\t"
                        "nop \n\t");
                }//for
        u16milliseconds--;
        }//while
}
int main() {
/* Port Configurations */ // DS70616G-page 209
// ODCB (open drain config) unimplemented (DS70616G, Table 4-56)
ANSELBbits.ANSB8=0;      //Digital I/O
CNENBbits.CNIEB8=0;      //Disable change notification interrupt
CNPUBbits.CNPUB8=0;      //Disable weak pullup
CNPDBbits.CNPDB8=0;      //Disable weak pulldown
TRISBbits.TRISB8=0;      //Pin B8: Digital Output
LATBbits.LATB8=0;        //Pin B8: Low
/* Endless Loop */
while(1){
        /* LATBbits.LATB8 = !(LATBbits.LATB8); //Toggle Pin B8 */
        _LED200=!_LED200; //Toggle LED
        delay_ms(500);
}//while
return (EXIT_SUCCESS);  //never reached
} //main()
```

Listing 1: Quellcode HelloWorld2

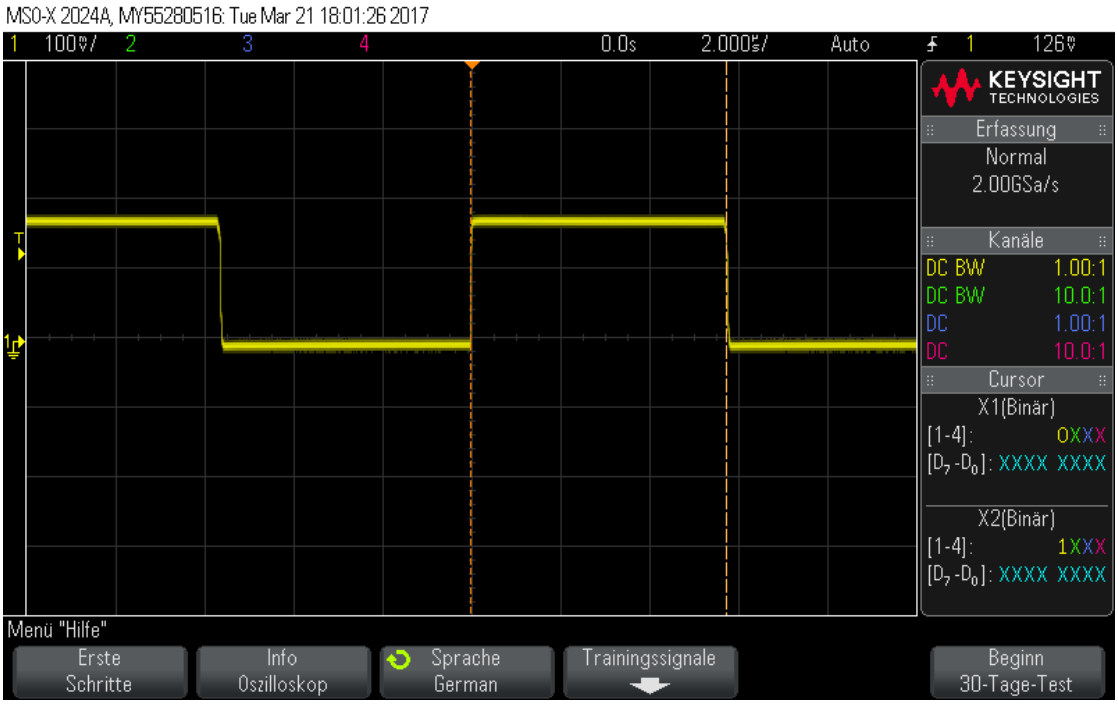


Abbildung 1: Ansteuerungsfrequenz der LED

```
//while(1){
// _LED200=!_LED200; //Toggle LED
00033E  8070A1      MOV LATB, W1
000340  201000      MOV #0x100, W0
000342  608000      AND W1, W0, W0
000344  A7F000      BTSC W0, #15
000346  EA0000      NEG W0, W0
000348  E90000      DEC W0, W0
00034A  DE004F      LSR W0, #15, W0
00034C  784000      MOV.B W0, W0
00034E  FB8000      ZE W0, W0
000350  600061      AND W0, #0x1, W0
000352  DD0048      SL W0, #8, W0
000354  8070A1      MOV LATB, W1
000356  A18001      BCLR W1, #8
000358  700001      IOR W0, W1, W0
00035A  8870A0      MOV W0, LATB
//}//while
00035C  37FFF0      BRA 0x33E
//return (EXIT_SUCCESS);  //never reached
//} //main()
```

Listing 2: Assembler Befehle zum toggeln

```
// Select Internal FRC at POR
_FOSCSEL(FNOSC_PRIPLL); //Initial Oscillator: Primary Oscillator
    (XT, HS, EC) with PLL
_FOSC(POSCMD_HS);  //HS Crystal Oscillator Mode

int main()
{
// Configure PLL prescaler, PLL postscaler, PLL divisor
PLLFBD=455; // PLLDIV
CLKDIVbits.PLLPOST=2;
CLKDIVbits.PLLPRE=2;

// Wait for PLL to lock
while (OSCCONbits.LOCK!= 1);


while(1)
{
//endless loop
}

return 1; //never reached
}
```

Listing 3: Code Example for Using PLL with 7.37 MHz Internal FRC

9.1 CPU Clocking System

The dsPIC33EPXXX(GP/MC/MU)806/810/814 and PIC24EPXXX(GP/GU)810/814 family of devices provides seven system clock options:

- Fast RC (FRC) Oscillator
- FRC Oscillator with Phase-Locked Loop (PLL)
- Primary (XT, HS or EC) Oscillator
- Primary Oscillator with PLL
- Secondary (LP) Oscillator
- Low-Power RC (LPRC) Oscillator
- FRC Oscillator with postscaler

Instruction execution speed or device operating frequency, FCY, is given by Equation 9-1.

EQUATION 9-1: DEVICE OPERATING FREQUENCY

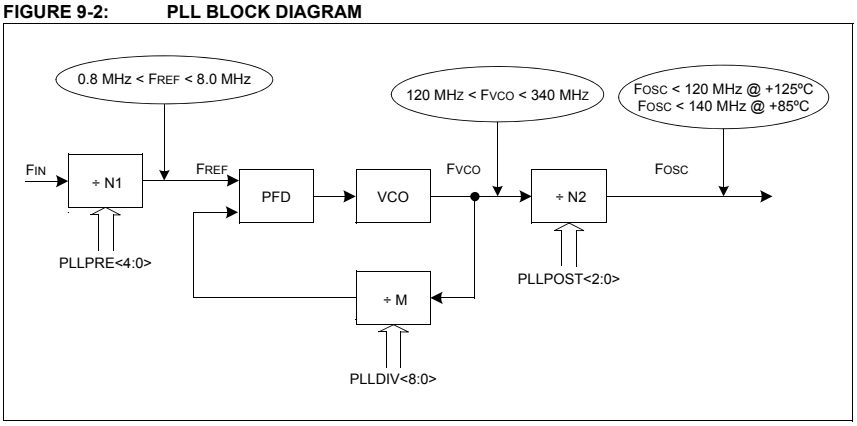
$$FCY = FOSC/2$$

Figure 9-2 is a block diagram of the PLL module.

Equation 9-2 provides the relation between input frequency (FIN) and output frequency (FOSC).

Equation 9-3 provides the relation between input frequency (FIN) and VCO frequency (FVCO).

FIGURE 9-2: PLL BLOCK DIAGRAM



EQUATION 9-2: FOSC CALCULATION

$$FOSC = FIN \times \left(\frac{M}{N1 \times N2}\right) = FIN \times \left(\frac{(PLLDIV + 2)}{(PLLPRE + 2) \times 2(PLLPOST + 1)}\right)$$

Where,

$N1 = PLLPRE + 2$

$N2 = 2 \text{ x } (PLLPOST + 1)$

$M = PLLDIV + 2$

EQUATION 9-3: FVCO CALCULATION

$$FVCO = FIN \times \left(\frac{M}{N1}\right) = FIN \times \left(\frac{(PLLDIV + 2)}{(PLLPRE + 2)}\right)$$

Abbildung 2: CPU Clocking System mit PLL Block Diagramm

PLLPOST<1:0>: PLL VCO Output Divider Select bits (also denoted as 'N2', PLL postscaler)
11 = Output divided by 8
10 = Reserved
01 = Output divided by 4 (default)
00 = Output divided by 2

Unimplemented: Read as '0'

PLLPRE<4:0>: PLL Phase Detector Input Divider Select bits (also denoted as 'N1', PLL prescaler)
11111 = Input divided by 33
•
•
•
00001 = Input divided by 3
00000 = Input divided by 2 (default)

PLLDIV<8:0>: PLL Feedback Divisor bits (also denoted as 'M', PLL multiplier)
111111111 = 513
•
•
•
000110000 = 50 (default)
•
•
•
000000010 = 4
000000001 = 3
000000000 = 2

Abbildung 3: Wertebereich der PLL Parameter

B8 =B2*(B4+2)/((B5+2)*2*(B6+1))

	A	B	C	D	E
1			Einheit	Wertebereich	
2	FIN	7,37	[MHz]		
3					
4	PLLDIV	455		2...513	
5	PLLPRE	2		2...33	
6	PLLPOST	2		2;4;8	
7					
8	FOSC	140,337083	[MHz]		

Abbildung 4: PLL Parameter Excel

2 Rechenleistung

2.1 Aufgabenstellung

1. Ermitteln Sie die durchschnittliche Laufzeit einschließlich Streuung arithmetischer Grundoperationen für verschiedene vom XC-16-Compiler unterstützten Datentypen. Wie erklären Sie die Unterschiede?

2. Berechnen Sie die ersten 10 Primzahlen, die größer als 1.000.000 (1E6) sind. Implementieren Sie denselben Algorithmus auf einem PC und vergleichen Sie die Rechenzeiten.

2.2 Lösung

1. Der Programmcode wird so umgeändert wie in Listening 4 zu sehen. Zuerst wird mit dem Oszi nur die Zeit gemessen wie lange eine LED aus ist (ohne Rechenoperation, $29,1ns$). Anschließend kann man mit dem Oszi messen wie lange eine Grundoperation (mit Zufallszahlen) inklusiv LED ein/ausschalten benötigt. Die folgende Auflistung beinhaltet nur die Zeitdauer für die jeweilige Rechenoperation (ohne LED ein/aus).

Datentyp	Operation	Zeitdauer		Datentyp	Operation	Zeitdauer
uint8_t	+	43,7 ns		int8_t	+	57,4 ns
	-	58,1 ns			-	57,4 ns
	*	72,1 ns			*	71,9 ns
	/	72,5 ns			/	343,9 ns
uint16_t	+	43,4 ns		int16_t	+	43,3 ns
	-	43,5 ns			-	43,4 ns
	*	86,2 ns			*	87,4 ns
	/	330,9 ns			/	329,9 ns
uint32_t	+	115,4 ns		int32_t	+	142,9 ns
	-	114,9 ns			-	112,9 ns
	*	245,9 ns			*	230,9 ns
	/	7,559 us			/	7,95 us
uint64_t	+			int64_t	+	254,9 ns
	-				-	226,9 ns
	*				*	1,5 us
	/				/	131 us
float	+			long double	+	
	-				-	
	*				*	
	/				/	

2. Der geforderte Algorithmus ist in Listening 5 abgebildet. Bei dem verfügbaren Computer (Intel Core i7 2.6GHz, 16GB RAM, 64Bit Windows 10) ergab sich eine Laufzeit von ungefähr $5us$. (Gemessen mit CodeBlocks, $50s$ für 10^7 Durchläufe)

   Die Laufzeit des selben Programms (angepasst auf die Hardware) benötigte auf dem Mikrocontroller Board $47,8ms$. Der Code hierzu ist in Listing 6 abgebildet.

   Damit ist der Computer ca 10.000 mal schneller als der μC. ($\frac{47,8ms}{5us} = 9560$, Abbildung 5).

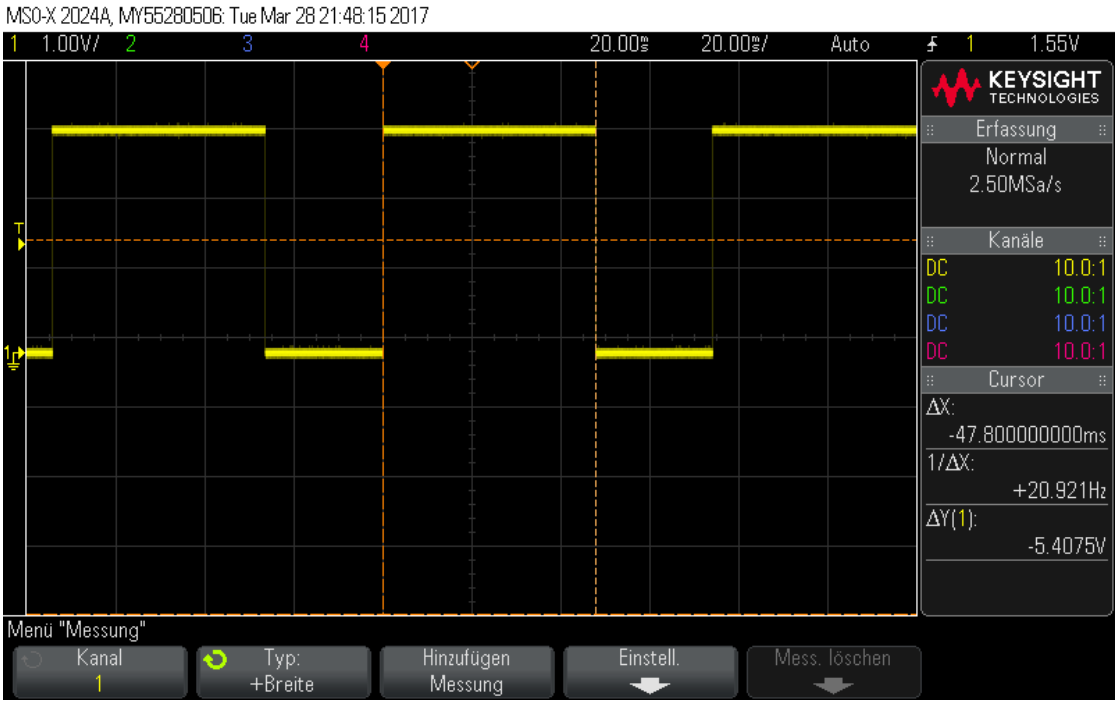


Abbildung 5: Laufzeit der Primzahlenberechnung

```
/* Endless Loop */
while(1){
_LED200=0;
ui8Var1 *= ui8Var2;
_LED200=1;
}//while
```

Listing 4: Bestimmen der Rechenleistung

```
#include<stdint.h>
#include<stdlib.h>
#include<math.h>

uint8_t isPrim(uint32_t ui32Number);

int main()
{
        uint32_t ui32Number= 1e6;  //start value
        uint16_t ui8PrimeCounter=0; //counts the number of
            calculated prime numbers
        const uint16_t ui8PrimMax=100;

        for(; ui8PrimeCounter<ui8PrimMax; ui32Number++)
                if(isPrim(ui32Number)) //check if the number is
                    prime
                {
                //printf("%d\t%d\n",ui8PrimeCounter,ui32Number);
                ui8PrimeCounter++; //increase PrimeCounter, if
                    the number is prime
                }
        return 0;
}
uint8_t isPrim(uint32_t ui32Number){
        uint32_t ui32Divider;
        uint32_t ui32SqrtNumber =((uint32_t)
            sqrt((double)(ui32Number)))+1;
        for(ui32Divider=2; ui32Divider<ui32SqrtNumber;
            ui32Divider++)
        {
                if((ui32Number%ui32Divider) == 0)
                {
                        return 0; //uiNumber32 isn't a prime
                            number
                }
        }
        return 1; //ui32Number is a Prime Number
}
```

Listing 5: Algorithmus zur Berechnung der ersten 100 Primzahlen größer als 1E6

```
int main() {    //scope_25 47,8ms

        PLLFBD = 418;
        CLKDIVbits.PLLPOST = 2;
        CLKDIVbits.PLLPRE = 2;

        /* Port Configurations */
        // DS70616G-page 209
        // ODCB (open drain config) unimplemented (DS70616G, Table 4-56)
        ANSELBbits.ANSB8=0;     //Digital I/O
        CNENBbits.CNIEB8=0;     //Disable change notification interrupt
        CNPUBbits.CNPUB8=0;     //Disable weak pullup
        CNPDBbits.CNPDB8=0;     //Disable weak pulldown
        TRISBbits.TRISB8=0;     //Pin B8: Digital Output
        LATBbits.LATB8=0;       //Pin B8: Low
        _LED200 = 1;
        //uint32_t    ui32Var1=2;
        //uint32_t    ui32Var2=2;
        //uint32_t    ui32Var3=2;
        while (OSCCONbits.LOCK!= 1);
        /* Endless Loop */

        uint32_t ui32Number= 1000000;     //start value
        uint16_t ui8PrimeCounter=0;       //counts the number of calculated prime numbers
        const uint16_t ui8PrimMax=10;

        while(1){

                _LED200=1;      //hard on/off 28,5ns  scope_15

                ui32Number= 1000000;     //start value
                ui8PrimeCounter=0;       //counts the number of calculated prime numbers
                //ui8PrimMax=10;

                for(; ui8PrimeCounter<ui8PrimMax; ui32Number++)
                if(isPrim(ui32Number)) //check if the number is prime
                {
                        //printf("%d\t%d\n",ui8PrimeCounter,ui32Number);
                        ui8PrimeCounter++; //increase PrimeCounter, if the number is prime
                }

                _LED200=0;
                delay_ms(500);
```

```
44
45          }//while
46
47          return (EXIT_SUCCESS);  //never reached
48  } //main()
49
50  uint8_t isPrim(uint32_t ui32Number){
51
52          if(ui32Number==0 || ui32Number==1)
53          return 0;
54
55
56          if((ui32Number%2)==0)
57          {
58                  if(ui32Number==2)
59                  {
60                          return 1;
61                  }
62                  else
63                  {
64                          return 0;
65                  }
66          }
67          uint32_t ui32Divider;
68          uint32_t ui32SqrtNumber =((uint32_t) sqrt((long
                double)(ui32Number)))+1;
69
70          for(ui32Divider=3; ui32Divider<ui32SqrtNumber;
                ui32Divider+=2)
71          {
72                  if((ui32Number%ui32Divider) == 0) //check if
                        ui32Divider is a in whole divider of ui32Number
73                  {
74                          return 0; //uiNumber32 isn't a prime
                                number
75                  }
76          }
77          return 1; //ui32Number is a Prime Number
78  }
```

Listing 6: Algorithmus zur Berechnung der ersten 100 Primzahlen größer als 1E6 auf dem uC

3 IO-Bibliothek

3.1 Aufgabenstellung

Die populäre Arduino-Plattform (https://www.arduino.cc/en/Reference/) kapselt die Pinkonfiguration und -ansteuerung mit folgenden Funktionen:

- pinMode()
- digitalRead()
- digitalWrite()

Übertragen Sie dieses Konzept auf das EDA-Board. Anwendungsbeispiele:

- pinMode(SW1, IPUT_PULLUP)

soll den Pin, an den SW1 angeschlossen ist, als digitalen Eingang konfigurieren und den Pullup-Widerstand einschalten.

- digitalWrite(LED203, HIGH)

soll an dem Pin, an den LED203 angeschlossen ist, einen High-Pegel ausgeben. Verwenden Sie die Bezeichner aus dem Schaltplan. Modularisieren Sie Ihre Software, verwenden Sie dazu die Dateinamen *edaPIC33Hardware.h* und *edaPIC33Hardware.c*.

Dokumentieren Sie die Funktionen mit Doxygen.

Messen Sie die Zeit, die zur Ansteuerung eines Ausgangspins mit den IO-Bibliotheksfunktionen notwendig ist und vergleichen Sie diese mit einem direkten Schreiben in die entsprechenden Hardwareregister.

3.2 Lösung

1. Lösungsansatz 1
2. Lösungsansatz 2
3. Lösungsansatz 3
4. Lösungsansatz 4

```
1 //Quellcode
```

Listing 7: Listening Bezeichnung

4 Mitschrift aus Vorlesung

4.1 Vorlesung 04.04.2017 - IO-Bibliothek

- jeder uC is universell aufgebaut und muss für jeden Fall individuell konfiguriert werden
- kleine Bibliothek zum anpassen der IO-Ports ist praktisch
- Lese und Schreib Funktionalität soll realisiert werden
- Datenblatt Figure 11-1 -> Pins müssen über Treiber realisiert werden, Schmitt-Trigger ist enthalten
- Pegel um Treiber zu aktivieren? nachlesen Schaltbild könnte falsch sein
- Lesezugriff über Port, Schreibezugriff über Latch
- Erkenntnis: Lesen des Datenblattes liefert Aufschluss wie die Pins zu beschalten sind.

4.2 Lösung

1. Lösungsansatz 1
2. Lösungsansatz 2
3. Lösungsansatz 3
4. Lösungsansatz 4

```
1 //Quellcode
```

Listing 8: Listening Bezeichnung

5 Kapitelbezeichnung

5.1 Aufgabenstellung

1. Aufgabe 1
2. Aufgabe 2
3. Aufgabe 3
4. Aufgabe 4

5.2 Lösung

1. Lösungsansatz 1
2. Lösungsansatz 2
3. Lösungsansatz 3
4. Lösungsansatz 4

```
1 //Quellcode
```

Listing 9: Listening Bezeichnung

6 Quellcode

6.1 C-Code

Anhang C-Code Syntax highlighting.

```
1
2  void delay_ms(uint16_t u16milliseconds){
3      uint16_t ui16_i=0;
4      while(u16milliseconds){
5          for (ui16_i=0;ui16_i<331;ui16_i++){     //1 ms delay loop
6              __asm__ volatile("nop \n\t"
7                               "nop \n\t"
8                               "nop \n\t");
9          }//for
10         u16milliseconds--;
11     }//while
12 }
13
14 int16_t i16_x;
```

Listing 10: Das Listing zeigt C Quellcode

```
1 //leeres listing
```

Listing 11: leeres listing