Albert-Ludwigs-Universität Freiburg
Institut für Informatik

Lehrstuhl für Rechnerarchitektur
Professor Dr. Bernd Becker

Studienarbeit

Das Picee++ System

Integration von Microcontroller und FPGA
zu einer Testumgebung

Heiko Falk

25. März 2003

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Ziele

Ziel dieser Studienarbeit ist die Vorbereitung des Hardwarepraktikums auf Grundlage der Picee Entwicklungsumgebung. Hierbei soll eine Erweiterungskarte für die Picee Entwicklungsumgebung entworfen werden, mit deren Hilfe die Studierenden in der Lage sind, Versuche zu logischen Schaltungen auf Transistor- und Gatterebene durchzuführen. Sowie eine Erweiterungskarte, die eine Verbindung zwischen der Picee Entwicklungsumgebung und FPGAs der Baureihe MAX7000S der Firma Altera ermöglicht.
Desweiteren sollen Vorschläge gemacht werden, wie sich die Erweiterungen im Rahmen des Hardware-Praktikums nutzen lassen, sowie konkrete Programm-Beispiele und wiederverwendbare Programm-Module entwickelt werden.

1.2 Vorgehensweise

Im 2. Kapitel wird anhand von Beispielprogrammen gezeigt, wie sich die Picee Entwicklungsumgebung mit 4MHz betreiben läßt. Desweiteren wird untersucht, welche Funktionen für spätere Versuchsreihen nutzbar sind.
Im 3. Kapitel wird die erste Erweiterungsplatine zum Aufbau logischer Schaltungen entwickelt, sowie konkrete Versuchsaufbauten beschrieben.
Im 4. Kapitel wird die Erweiterungsplatine zur Kommunikation zwischen dem auf der Picee-Platine vorhandenem PIC16F84 und dem FPGA, sowie dazugehörigem Protokoll entwickelt. Auch hierzu werden konkrete Versuchsaufbauten beschrieben.

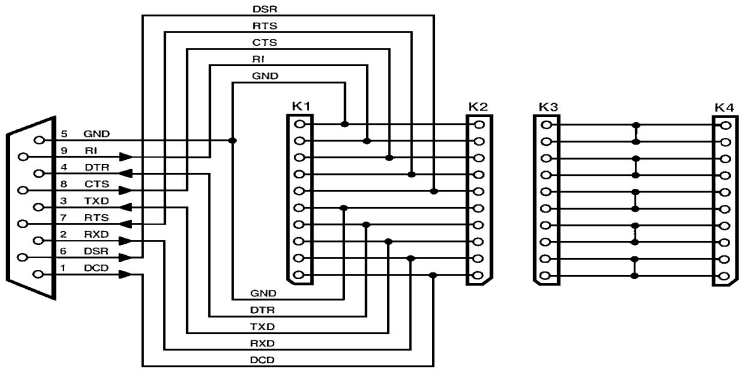

Abbildung 1.1: Schaltplan des Elektronik-Experimentier-Sets (ELEXS)

1.3 Das Hardware-Praktikum im Überblick

Das Hardware-Praktikum war bisher in drei Versuchsreihen unterteilt. Die Versuche der ersten Versuchsreihe beschäftigten sich mit den Grundlagen der Analog- und Digitaltechnik. Hierzu wurde mittels der seriellen Schnittstelle des PCs eine kleine Experimentierplatine (Schaltplan siehe Abbildung 1.1) angesteuert. Auf ihr konnten kleine Schaltungen auf Transistorebene realisiert werden, die sich dann mit den Signalen der seriellen Schnittstelle steuern ließen. Das Problem hierbei waren die oft sehr unterschiedlichen Spezifikationen der seriellen Schnittstelle bei verschiedenen Rechnern. Die Abweichungen waren teilweise so stark, dass einige Schaltungen nur bedingt lauffähig waren. Um dieses Problem in Zukunft zu umgehen, soll diese Versuchsreihe auf die Picee Entwicklungsumgebung (siehe Abbildung 2.1) übertragen werden, um allen Studierenden die gleichen Voraussetzungen zu bieten.

Die zweite Versuchsreihe behandelte den Aufbau kombinatorischer und sequentieller Schaltkreise. Diese konnten mit Hilfe der Software MAX+plus II Baseline [5] der Firma Altera grafisch oder aber mittels VHDL entwickelt werden. Wegen der ständig steigenden Studentenzahlen ist es allerdings nicht mehr möglich, alle Versuche im Praktikumsraum durchzuführen. Daher gab es im letzten Hardware-Praktikum nur einen Präsenztermin, bei dem die Studierenden die Möglichkeit hatten, ihre Schaltungsentwürfe auf einem FPGA der Baureihe MAX7000S [4] der Firma Altera zu testen. Alle anderen Versuche konnten nur in der Simulation nachvollzogen werden. Um zukünftig alle Versuche wieder direkt an der Hardware durchführen zu können, soll eine kostengünstige Lösung gefunden werden, welche die Studierenden mit nach Hause nehmen können. Um hierbei auch die Interaktion mit anderen Hardwarekomponenten kennen zu lernen, soll auch diese Lösung auf der Picee Entwicklungsumgebung basieren.

Ziel der dritten Versuchsreihe war es, die Mikroprozessor-Programmierung kennenzulernen. Diese Versuchsreihe wurde auf Basis der Picee Entwicklungsumgebung durchgeführt. Diese hat sich bewährt und eignet sich somit als Grundlage für weitere Praktika.

Durch die Integration dieser drei Versuchsreihen zu einer Testumgebung auf Basis des Picee ist es nun möglich, alle Versuche hardwarenah auch zu Hause durchzuführen. Somit ist es möglich den Studierenden auch zukünftig eine hohe Qualität der Lehre anzubieten, unabhängig von der Anzahl der Teilnehmer eines Praktikums.

Kapitel 2

Picee

2.1 Überblick

Die Picee Entwicklungsumgebung (siehe Abbildung 2.1) besteht im wesentlichen aus dem Microprozessor PIC16F84 der Firma Microchip [18], einem LCD-Display basierend auf dem LCD-Controller HD44780u der Firma Hitachi [12], sowie diversen Tastern und Leuchtdioden. Der Prozessor wird über die serielle Schnittstelle des PCs programmiert, hierbei ist darauf zu achten, dass der Jumper S1 richtig gesetzt ist. Die Entwicklung der Programme erfolgt mit der Entwicklungsumgebung "MPLAB" der Firma Microchip [20] [19], es ist jedoch auch möglich mit einem beliebigen Editor den Assemblercode zu erstellen und diesen dann mit der Software "MPASM for Windows" zu assemblieren. Die Übertragung der hieraus gewonnenen hex-Files zum PIC erfolgt mit der Software "IC Programmer" von Bonny Gijzen [10]. Hierbei sollte darauf geachtet werden, dass die Optionen WDT, PWRT und CP auf der rechten Fensterseite korrekt eingestellt sind. Um zu vermeiden, dass bei der Übertragung diese Optionen falsch eingestellt sind, empfiehlt es sich, diese direkt im Assemblercode einzustellen.

```
;-----------------------------------------------------------------
          org       2007      ; setzen der Configuration Bits
                              ; bei Adresse 2007
          de        b'11001'  ; CP=off, PWRTE=enable; WDT=disable
                              ; bit1,0:     11 =RC-Oscillator
                              ;             01 =XT-Oscillator
;-----------------------------------------------------------------
```

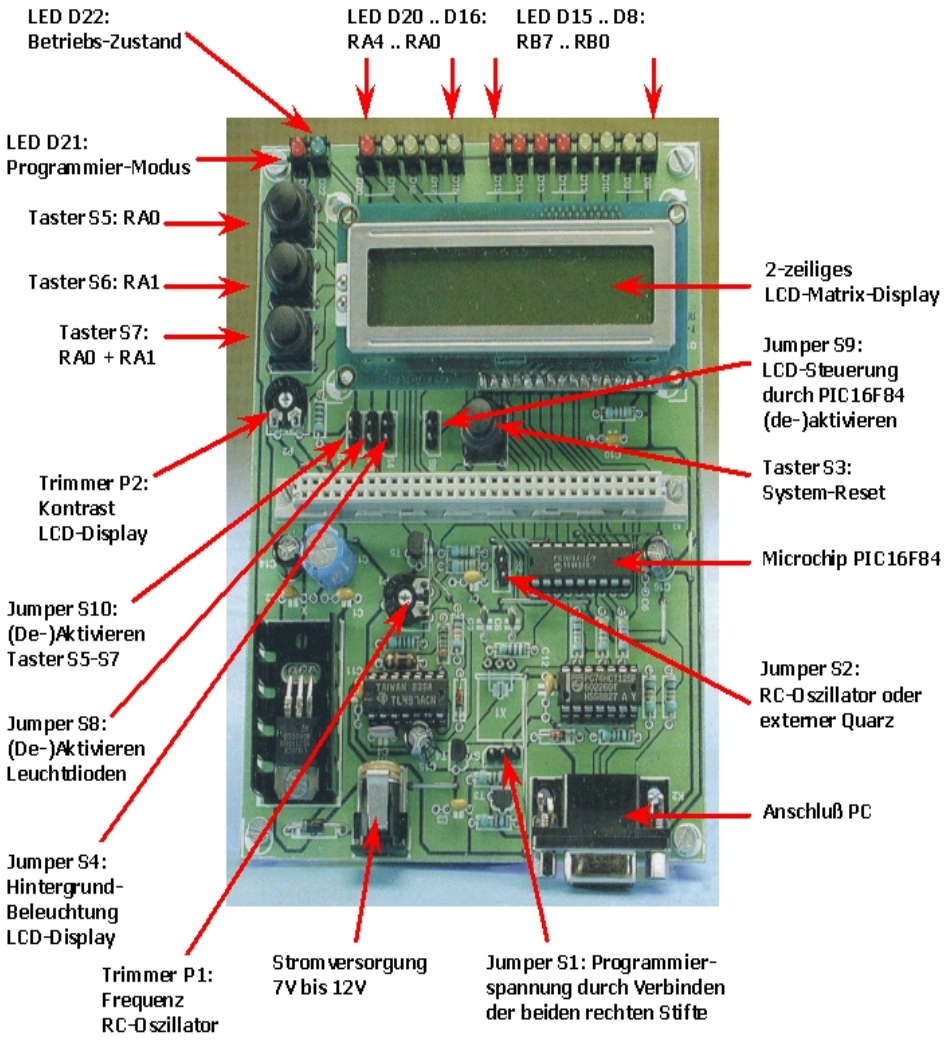

Abbildung 2.1: Picee Entwicklungsumgebung

2.2 LCD-Display

Wenn die Picee Entwicklungsumgebung mit 4MHz betrieben wird, muss darauf geachtet werden, dass bei den Operationen, welche das Display betreffen, die Timing-Vorschriften eingehalten werden. Die nötigen Informationen hierzu befinden sich im Datenblatt zum Display Controller HD44780U [12]. Die wichtigsten Funktionen sind in der Datei lcd.inc zusammengefasst, so dass diese schnell in neue Projekte zu integrieren sind, ohne sie jedesmal neu zu schreiben. Da des öfteren kurze Pausen in den Programmen benötigt werden, werden auch dazu einige Funktionen in der Datei pause.inc bereitgestellt. Das Display ist mit dem Microprozessor über dessen Ports verbunden. PortB ist für den eigendlichen Datenaustausch verantwortlich, über ihn werden Befehle an den Controller des Displays gesendet. Von PortA werden nur zwei Pins zur Steuerung des Displays verwendet. PortA2 wird zum Aktivieren des Displays gebraucht, PortA3 zur Wahl des Zielregisters (Instruction oder Data Register, siehe hierzu [12]).

2.3 Tasten

In der Datei tasten.inc werden zwei Funktionen bereitgestellt, die es ermöglichen den Programmablauf solange zu unterbrechen, bis die entsprechende Taste gedrückt wird.

2.4 Timer

Der PIC16F84 besitzt einen Timer TMR0, dessen Takt vom externen Takt und dem vorgeschalteten Prescaler abhängt. Dabei handelt es sich um einen 8-Bit-Zähler, der je nach Einstellung den Takt teilt. Dieser "neue" Takt wird anschließend noch mit dem Systemtakt synchronisiert und bewirkt, dass das Register TMR0 um Eins hochgezählt wird. Die genaue Funktionsweise ist im Datenblatt zum PIC16F84 [18] ersichtlich.

2.5 Service-Routinen

Datei	Funktion	Beschreibung
lcd.inc	_lcdinit	initialisiert das Display
	_w2lcd	transportiert die Daten aus WReg zum Displaycontroller
	_shiftleft	bewegt den Cursor nach links
	_shiftright	bewegt den Cursor nach rechts
	_clearlcd	löscht das Display
	_line1	setzt den Cursor in die erste Zeile

	_line2	setzt den Cursor in die zweite Zeile
pause.inc	_pause40us	eine Pause von $40\mu s$
	_pause1ms	eine Pause von $1ms$
	_pause2ms	eine Pause von $2ms$
	_pause4ms	eine Pause von $4ms$
	_pause5ms	eine Pause von $5ms$
	_pause10ms	eine Pause von $10ms$
	_pause50ms	eine Pause von $50ms$
	_pause100ms	eine Pause von $100ms$
	_pause1s	eine Pause von $1s$
tasten.inc	_tasteS5	wartet bis der Taster S5 gedrückt wird
	_tasteS6	wartet bis der Taster S6 gedrückt wird
port.inc	_initportA	initialisiert PortA
	_initportBout	initialisiert PortB als Ausgang
	_initportBin	initialisiert PortB als Eingang
pic_fpga.inc	_initportBout_fpga	initialisiert PortB als Ausgang und Pin7-6 als Steuerleitung
	_initportBin_fpga	initialisiert PortB als Eingang und Pin7-6 als Steuerleitung
	_w2fpga_adr	transportiert die Daten aus WReg zum FPGA und speichert diese im Adress Register
	_getfpga	empfängt die Daten vom FPGA und speichert diese in der Variable "fpga" ab
	_w2fpga_data	transportiert die Daten aus WReg zum FPGA und speichert diese im Daten Register

2.6 Beispiel: Textausgabe auf dem Display

Im nachfolgenden Beispiel wird gezeigt, wie man mit Hilfe der Funktionen aus der Datei lcd.inc Zeichen auf das Display ausgeben kann und was dabei zu beachten ist.

Bei der Textausgabe auf dem Display ist es angenehm, den Text, den man ausgeben möchte, in einer Tabelle abzulegen. Der Befehl

```
dt "Hallo"
```

wird vom Assembler in folgende Tabelle (d. h. in die entsprechenden Befehle) umgewandelt:

```
retlw  'H'
retlw  'a'
retlw  'l'
retlw  'l'
retlw  'o'
```

Um das gewünschte Zeichen zu erhalten, ist es nötig, die korrekte Speicherposition in den Program Counter zu laden. Dabei ist darauf zu achten, dass der PIC16F84 über einen 13-Bit breiten Program Counter verfügt, im PCL aber nur die 8 low-Bits gespeichert sind. Die 5 high-Bits werden aus dem PCLATH ("Program Counter Latch high") geladen.
Daher werden im Beispiel zuerst die 8 low-Bits berechnet und anschließend der benötigte Wert der high-Bits in das PC Latch geschrieben. Diese 5 Bits werden erst bei dem Befehl

```
movwf  PCL
```

in den Program Counter übernommen [18] [15]. Ist man sich sicher, dass die Table-Operation nicht über eine Seitengrenze hinweg geht, kann man auf die Berechnung der 5 high-Bits verzichten, was den Aufwand erheblich reduziert. Um dies zu erreichen kann man mit dem Befehl org die Tabellen gezielt im Speicher ablegen. Unter einer Speicherseite versteht man hier den Speicherbereich, welcher durch die 8 low-Bits des Program Counters adressierbar sind. Um den ganzen Speicher zu adressieren, benötigt man noch die high-Bits, diese geben die Speicherseite an, in welcher man sich gerade befindet. Von einem Übergang der Seitengrenze spricht man also, wenn sich die high-Bits ändern.

Ausschnitte aus der Datei text.asm:

```
;-------- Displaytexte ----------------------------------------
 _txtmenue1
          movwf     PCL                 ; lade Speicherpos.
                                        ; des naechsten Zeichens
          dt        "Hallo Welt      "
                   ;"0123456789abcdef"  ; 15 Stellen
                                        ; Bit4 = 1 => Abbruch
;--------------------------------------------------------------


;-------- Menue -----------------------------------------------
_menue1   clrf      LCDPos              ; loesche LCDPos
          call      _line1
__menue1
          movf      LCDPos,W            ; berechne offset
          addlw     1
          movwf     offset

          movlw     LOW _txtmenue1      ; berechne speicherpos.
```

```
          addwf     offset,W            ; des naechsten zeichens
          movwf     offset

          movlw     HIGH _txtmenue1     ; bei Seitenuebergang
          btfsc     STATUS,C            ; wird PCLATH angepasst
          addlw     0x01
          movwf     PCLATH

          movf      offset,W            ; lade Speicherpos.
                                        ; in WReg
          call      _txtmenue1


          call      _w2lcd              ; gib Zeichen auf LCD aus
          incf      LCDPos,F            ; erhoehe Pos. Zaehler
          btfss     LCDPos,4            ; teste ob alle Zeichen
          goto      __menue1            ; ausgegeben
          return
;-----------------------------------------------------------------
```

2.7 Beispiel: Stoppuhr

Nachfolgend wird der Timer des Microcontrollers, sowie die Interruptbehandlung vorgestellt. Beide Komponenten spielen bei späteren Programmen, wie z.B. der Messung der Kondensator-Ladezeit, eine wichtige Rolle.

Das Beispiel Stoppuhr [1] zeigt, wie man den Timer des PIC16F84 dazu verwenden kann, eine Stoppuhr zu programmieren, die sekundengenau die Zeit misst.
Zuerst wird der Prescaler auf $\frac{1}{4}$ eingestellt, so dass der Timer noch mit einem Takt von 1MHz arbeitet.

```
;-------- Option Register setzen --------------------------------
_opinit
          bsf       STATUS,RP0          ; select Bank 1
          movlw     b'00000001'         ; Prescaler 1:4
                                        ; Prescaler assigned
                                        ; to TMR0
          movwf     OPTION_REG
          bcf       STATUS,RP0          ; select Bank 0
          return
```

[1] stoppuhr.asm

```
;-----------------------------------------------------------------
```

Im Hauptteil des Programmes, wird der Startwert des Timers gesetzt. Der Wert 8 wurde experimentell mit dem Oszilloskop ermittelt, um die gewünschte Zeit exakt zu erhalten.

```
        movlw     0x8                ; setze Startwert
                                     ; fuer TMR0
        movwf     TMR0               ; bewirkt, dass TMR nur
                                     ; ca. 250 mal bis zum
                                     ; Interrupt hochzaehlt
```

Durch den neu erzeugten Takt von 1MHz ergibt sich eine Periode von 1us. Da ein Befehlszyklus 4 Takte benötigt, wird der Registerinhalt von TMR0 im Intervall von $4\mu s$ um Eins erhöht. Durch den Startwert des Registers ergibt sich somit eine Zeit von $1ms$ bis das Register überläuft und einen Interrupt auslöst.
Jetzt wird noch mit Hilfe der beiden Variablen Counter1 und Counter2 auf 1000 gezählt, um dann jede Sekunde den Wert der Stoppuhr zu erhöhen. Dies geschieht im Unterprogramm _time, hier werden die Variablen für die Sekunden und Minuten gesetzt und auf das Display ausgegeben.

Kapitel 3

Logische Schaltungen

Bisher wurde im Praktikum das "Elektronik Experimentier Set" (ELEXS) (siehe Abbildung 1.1) zum Aufbau logischer Schaltungen verwendet. Ausgehend von dieser Platine soll nun eine Erweiterungsplatine für die Picee Entwicklungsumgebung entworfen werden. Der Gedanke hierbei ist die Interaktion zwischen Microcontroller und den selbst entworfenen logischen Schaltungen, sowie die Vermeidung von Problemen, welche durch unterschiedliche Spezifikationen der seriellen Schnittstelle an verschiedenen Rechnern hervorgerufen werden (z.B. durch unterschiedliche Schwellspannungen).

3.1 Erweiterungsplatine

In Abbildung 3.1 ist die fertige Erweiterungsplatine 1 zu sehen. Auf ihr befinden sich eine Experimentierfläche, auf der man einfache logische Schaltungen auf Transistorebene aufbauen kann. Des Weiteren sind diverse IC-Fassungen, sowie zwei Taster vorhanden.
Die Schnittstelle zur Picee Entwicklungsumgebung befindet sich am unteren Teil der Platine und ist identisch mit der Schnittstelle auf der Erweiterungsplatine 2. Am linken Rand sitzt der Program-Enable Schalter, er legt fest, ob sich die Picee Entwicklungsumgebung im Programmierstatus oder im Arbeitsmodus befindet. Weiter sind die Anschlüsse der Picee Entwicklungsumgebung nach oben geführt und stehen in Form von Steckleisten zur Verfügung. Am rechten Rand befindet sich eine Steckerleiste mit der Versorgungsspannung V_{Bat}, die aus Sicherheitsgründen über einen Jumper abschaltbar ist.

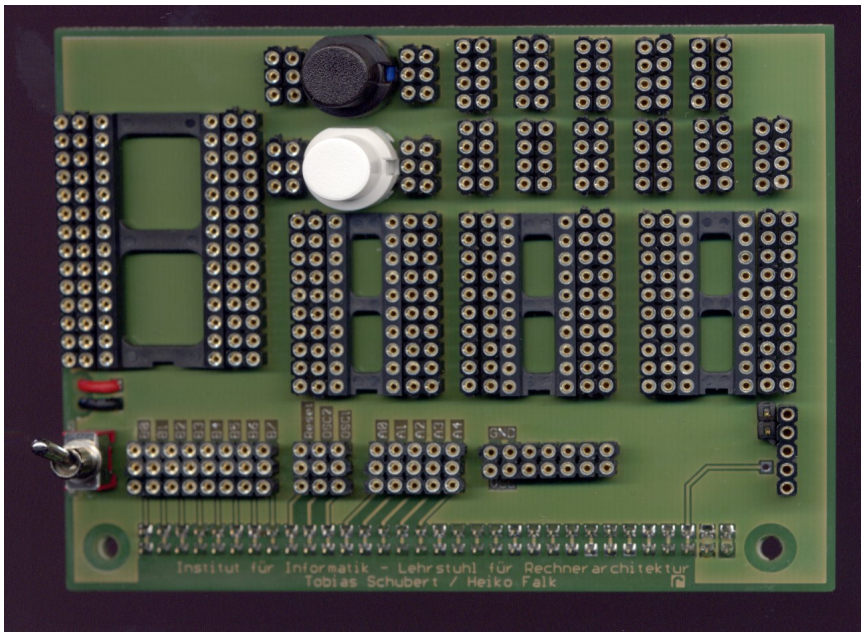

Abbildung 3.1: Erweiterungsplatine 1

3.2 Beispiel: Kondensatorladezeit bestimmen

Im Hardwarepraktikum gibt es einen Versuch, bei dem die Kapazität eines Kondensatores bestimmt werden soll. Dazu wird die Zeit gemessen, welche der Kondensator benötigt, um eine gewisse Schwellspannung zu erreichen. Abbildung 3.2 veranschaulicht den zeitlichen Ablauf des Ladevorgangs eines Kondensators. Die Ladefunktion hat an der Stelle $T = R \cdot C$ etwa 63% ihres Endzustandes erreicht. Bei einer Versorgungsspannung von $5V$ entspricht dies $3.15V$. Der Schmitt-Trigger SN74HC14 [22], welcher der Schaltung (siehe Abbildung 3.3) vorgeschaltet ist, hat bei steigender Flanke eine Schwellspannung von $3V$. Schaltet nun der Schmitt-Trigger um, so kann man näherungsweise mit der Formel $C = \frac{T}{R}$ die Kapazität des Kondensators bestimmen. Wählt man $R = 1000\Omega$ so entspricht die gemessene Zeit in Millisekunden dem Betrag nach der Kapazität des Kondensators in Mikrofarad.
Im Beispiel [1] wird zu Beginn der Zeitmessung PortB1 auf high gesetzt, um den Ladevorgang des Kondensators zu starten. Anschließend wird durch Setzen der Interrupt-Bits die Interruptbehandlung aktiviert. Von jetzt an läuft die Zeitmessung bis PortA1 auf low gezogen wird. Für den normalen Eingang des PIC16F84 sind nur die Schwellspannungen für 0 bzw. 1 festgelegt, dazwischen ist ein undefinierter Bereich. Der Schmitt-Trigger SN74HC14 [22]

[1] kondzeit.asm

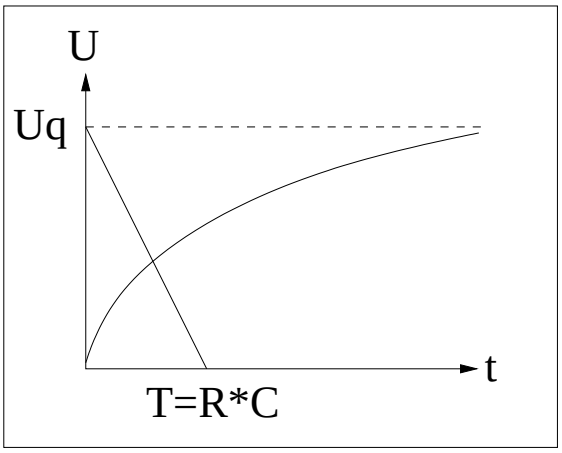

Abbildung 3.2: Ladekurve eines Kondensators

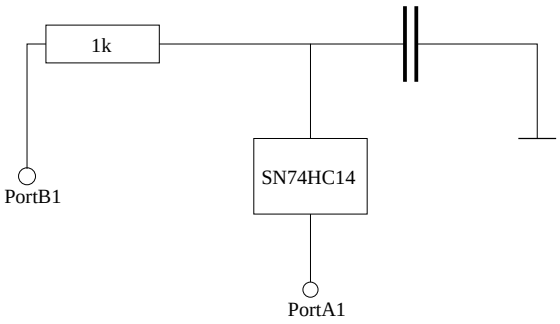

Abbildung 3.3: Versuchsaufbau Kondensatormessung

vor PortA1 stellt nun sicher, dass die Schwellspannung, bei welcher PortA1 auf low gezogen wird, fest definiert ist (3V).
Die Messung wird dann im Abstand von ca. 5 Sekunden, solange braucht der Kondensator um sich zu entladen, wiederholt.

Ausschnitt aus der Datei kondzeit.asm, welcher zeigt, wie der Ladevorgang gestartet wird:

```
;-----------------------------------------------------------------
          bsf       PORTB,1             ; start des Ladevorgangs
          bsf       INTCON,5            ; set TMR0 Interrupt Bit
          bsf       INTCON,7            ; set Global Int. Bit

_p        btfsc     PORTA,1             ; warte bis PortA1 = 0
          goto      _p
;-----------------------------------------------------------------
```

Kapitel 4

FPGA

Bisher wurde im Praktikum eine Experimentierumgebung auf Basis des PicoMax Boards [9] der Firma El Camino verwendet, mit dessen Hilfe es möglich war, die FPGAs der Firma Altera [4] zu programmieren und einfache Versuche durchzuführen. Aufgrund der Größe und der Kosten dieser Experimentierumgebung ist es allerdings nicht möglich, jedem Studierenden diese für den Zeitraum des Praktikums auszuleihen. Daher soll auf Grundlage des PicoMax Boards eine Erweiterungsplatine für die Picee Entwicklungsumgebung entworfen werden, die kostengünstiger und handlicher ist als die bisher eingesetzte Lösung. Der Gedanke ist auch hierbei die Interaktion zwischen Microcontroller und FPGA um abwechslungsreichere Versuchsaufbauten zu ermöglichen und den Studierenden die Möglichkeit zu geben sich mit dem Zusammenspiel verschiedenen Hardware-Komponenten auseinanderzusetzen.

4.1 Erweiterungsplatine

In Abbildung 4.1 ist die fertige Erweiterungsplatine 2 zu sehen. Auf ihr befindet sich das FPGA EPM7128SLC84-10 [4] der Firma Altera. Des Weiteren ist der Treiberbaustein SN74LS244 [21] sowie der Schmitt-Trigger [22], auf der Platine installiert. Beide Bausteine werden für die korrekte Ansteuerung des FPGAs benötigt [9]. Die Abbildungen 4.2 4.3 zeigen den Schaltplan hierzu. Da nicht alle Einheiten des Schmitt-Triggers gebraucht werden, stehen 3 für eigene Schaltungen auf der Platine zur Verfügung.
Die Schnittstelle zur Picee Entwicklungsumgebung befindet sich am unteren Teil der Platine und ist identisch mit der Schnittstelle auf der Erweiterungsplatine 1. Am linken Rand sitzt der Program-Enable Schalter, er legt fest, ob sich die Picee Entwicklungsumgebung im Programmierstatus oder im Arbeitsmodus befindet. Weiter sind die Anschlüsse der Picee Entwicklungsumgebung nach obengeführt und stehen in Form von Steckleisten

zur Verfügung. Am rechten Rand befindet sich eine Steckerleiste mit der Versorgungsspannung V_{Bat}, die aus Sicherheitsgründen über einen Jumper abschaltbar ist.

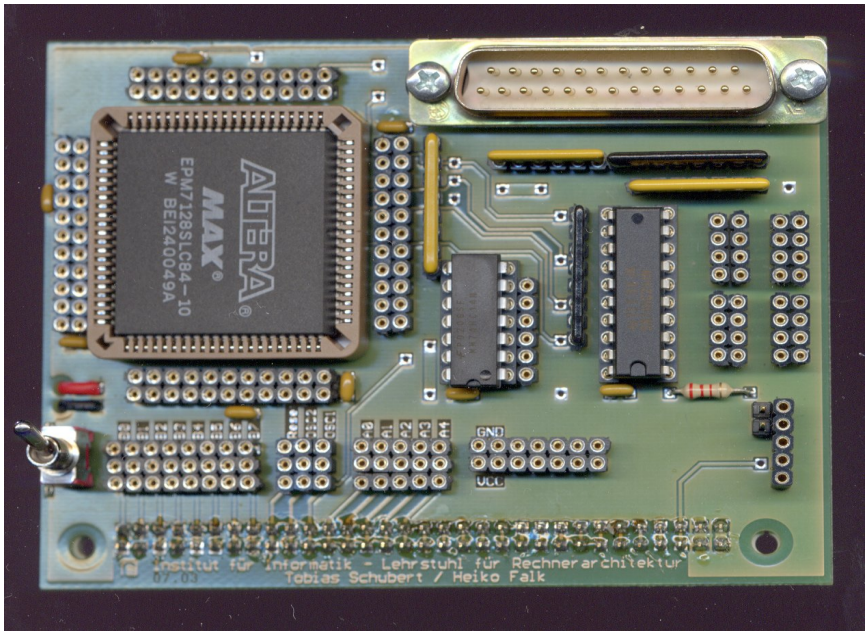

Abbildung 4.1: Erweiterungsplatine 2

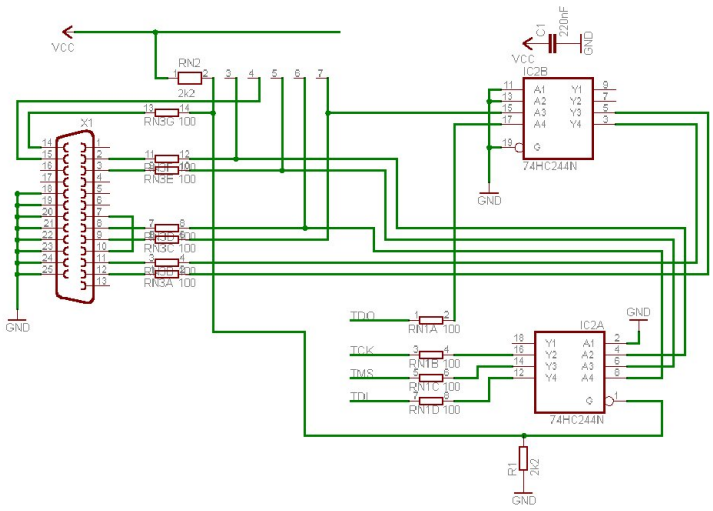

Abbildung 4.2: Schaltplan Erweiterungsplatine 2 / Teil 1

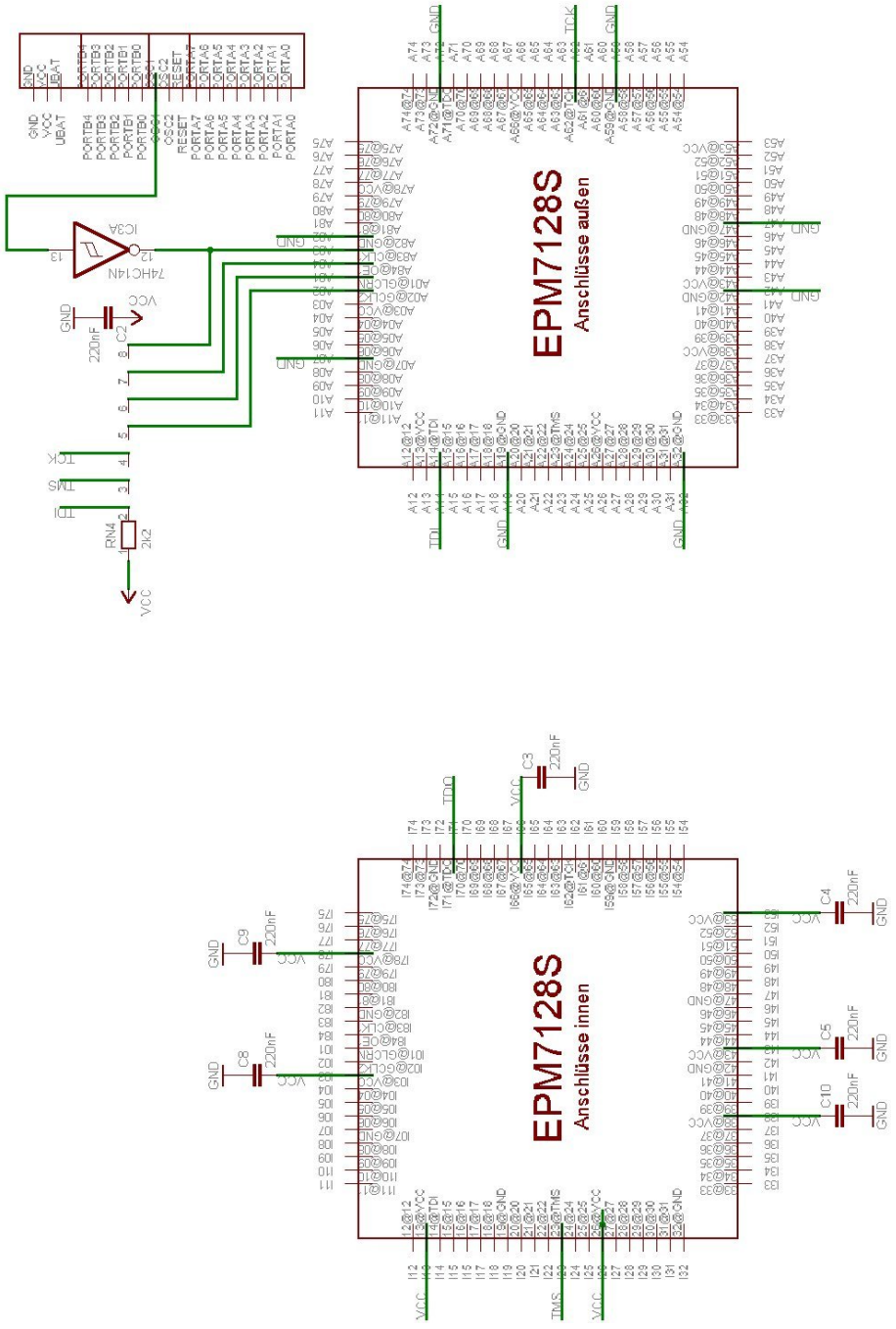

Abbildung 4.3: Schaltplan Erweiterungsplatine 2 / Teil 2

4.2 Protokoll zur Kommunikation PIC ↔ FPGA

Zur Kommunikation zwischen FPGA und der Picee Entwicklungsumgebung (Abbildung 2.1) ist es notwendig, ein Protokoll zu entwerfen, welches den korrekten Datenaustausch gewährleistet. Da der PIC16F84 nur über zwei Ports verfügt und über diese bereits das Display angesteuert wird, muss zwischen den Daten für das FPGA und den Daten für das Display unterschieden werden. Das Display wird über PortA2 enabled. Um also zu entscheiden, ob die Daten für das Display oder für das FPGA bestimmt sind, ist es notwendig den Status von PortA2 abzufragen. Die Daten, welche auf das Display übertragen werden sollen, müssen allerdings schon anliegen, bevor das Display aktiviert wird. Der Befehl _w2lcd aus der Hilfsdatei lcd.inc leistet das Gewünschte und überträgt die Daten aus dem Register W in den Display-Controller. Er sorgt also dafür, dass die Daten auf PortB anliegen und nach einer gewissen Zeit das Display aktiviert wird. Genau diese Zeitspanne ist die kritische Phase, da hier noch nicht klar ist, ob die Daten für das Display oder aber für das FPGA bestimmt sind. Es muss also auch von Seiten des FPGAs gewartet werden, ob das Display aktiviert wird. Erst wenn sichergestellt ist, das die Daten nicht für das Display bestimmt sind kann der FPGA mit der Verarbeitung der Daten beginnen.
Diese Aufgabe erledigt ein Interface, welches der Datei interface.vhd zugrunde liegt. In Abbildung 4.4 ist grob skizziert, welche Ein- und Ausgänge dieses Interface besitzt.

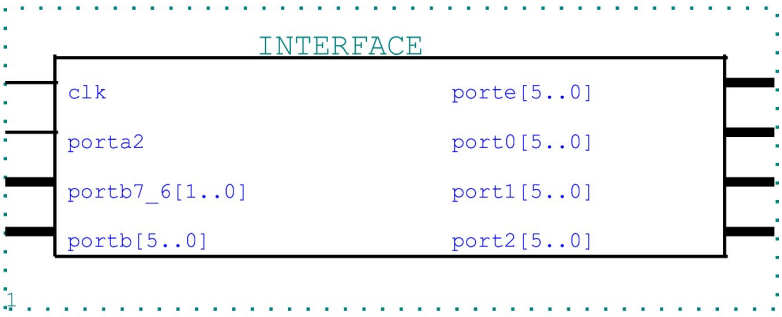

Abbildung 4.4: Grafische Darstellung des Interfaces

Der Eingang PortB7_6 des Interfaces wird mit PortB6 und PortB7 des PICs verbunden, über sie wird die Steuerung des Interfaces vorgenommen. PortB wird mit PortB5 bis 0 des PICs verbunden, über sie werden die Daten gesendet. Die Ausgänge Port0, Port1 und Port2 werden dazu verwendet, die Daten für die Schaltungen, welche auf dem FPGA betrieben werden sollen, bereitzustellen. An den Eingang PortE werden dann die Ergebnisse der Schaltungen übergeben, so dass diese wieder über PortB zum PIC gesendet

werden können. Die Kommunikation zwischen PIC und Interface basiert auf dem Protokoll, das durch den Automaten in Abbildung 4.5 beschrieben wird.

Zu Beginn befindet sich das Interface im Zustand "start". Wird nun auf portb7_6 die Belegung verändert, wird der Zustand dementsprechend gewechselt. Hierbei ist darauf zu achten, dass der Zustandsübergang erst nach einer gewissen Zeit stattfinden darf, da sichergestellt werden muss ob die Daten für das Display oder das FPGA bestimmt sind. Dies wird duch einen im Interface integrierten Zähler geleistet. Dieser wird sobald sich die Daten auf PortB7_6 ändern, neu gestartet und zählt dann bis 6, was $6\mu s$ entspricht. Wird in dieser Zeit die Belegung von PortB7_6 erneut geändert, so beginnt auch hier der Zählvorgang von Neuem. Die Befehle des PICs haben eine Ausführungszeit von $4\mu s$, so dass diese Zeitspanne ausreicht, um sicherzustellen für wen die Daten bestimmt sind.
Im Zustand "adr" wird die Adresse des Registers gespeichert, in welches man als nächstes Daten ablegen möchte. In die Zustände d0, d1, d2 gelangt man, wenn an PortB7_6 der Wert "10" anliegt und zuvor die entsprechende Adresse gespeichert wurde. In diesen Zuständen werden die Daten in den jeweiligen Registern gespeichert und liegen dann an Port0, Port1 bzw. Port2 des Interfaces an. Im Zustand "erg" werden die Daten, welche an PortE des Interfaces anliegen über PortB zum PIC übertragen.

Auf Seiten des PICs stehen Funktionen in der Datei pic_fpga.inc bereit, welche die Kommunikation mit dem FPGA unter Berücksichtigung der Timingvorschriften realisieren. Die wichtigsten Funktionen sind _w2fpga_adr, welche eine neue Adresse im FPGA speichert und _w2fpga_data, welche Daten in das Register speichert, das durch die gespeicherte Adresse gewählt wurde. Mit dem Befehl _getfpga wird das Ergebnis, das an PortE des Interfaces anliegt zum PIC übertragen und in der Variablen "fpga" gespeichert.
In Abbildung 4.6 ist ein Timingdiagramm einer Sequenz dieser drei Befehle zu sehen.

4.3 Wiederverwendbarkeit von Komponenten am Beispiel eines einfachen Rechners

In diesem Beispiel [1] wird gezeigt, wie man bestehende Komponenten in neue Projekte einbinden kann. Im Architecture-Teil wird zu Beginn mit dem Schlüsselwort component das Interface eingebunden. Im eigentlichen "Kern" des Architecture-Teils werden dann die Ports des Interfaces mit den Ports und Signalen des Rechners verbunden (siehe folgenden Ausschnitt der

[1] rechner.asm und rechner.vhd

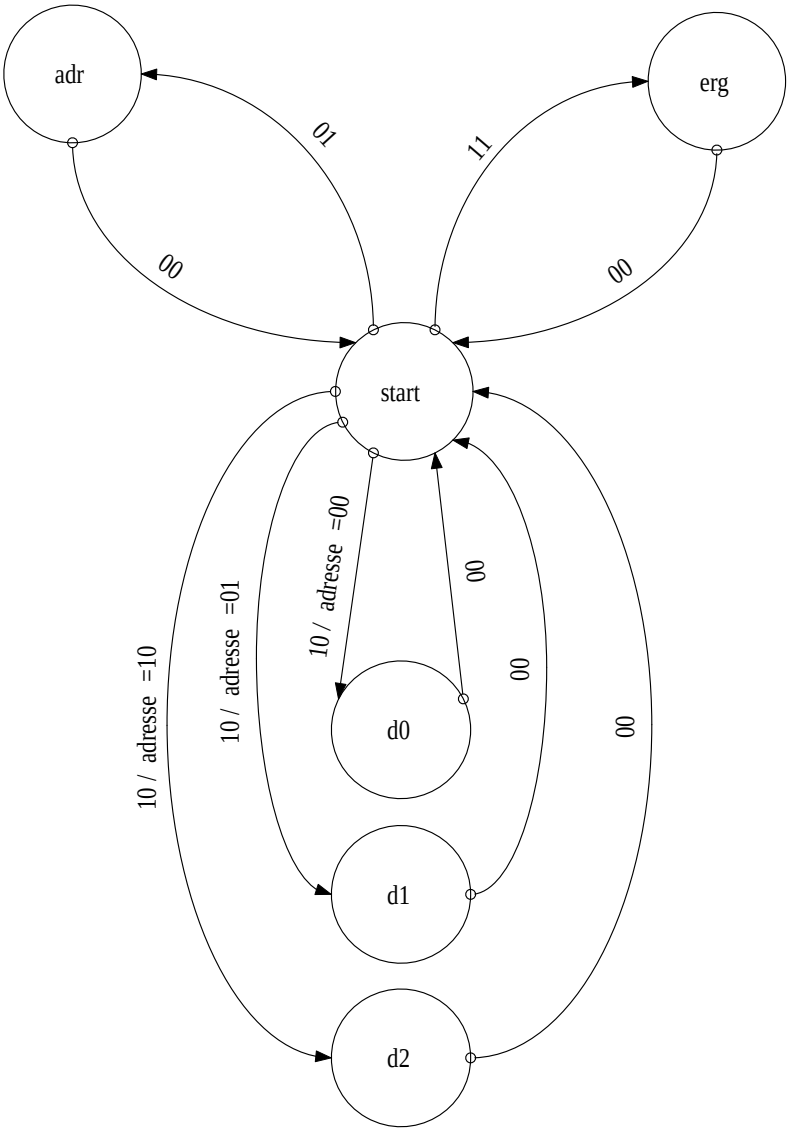

Abbildung 4.5: Automatenmodell des Protokolls

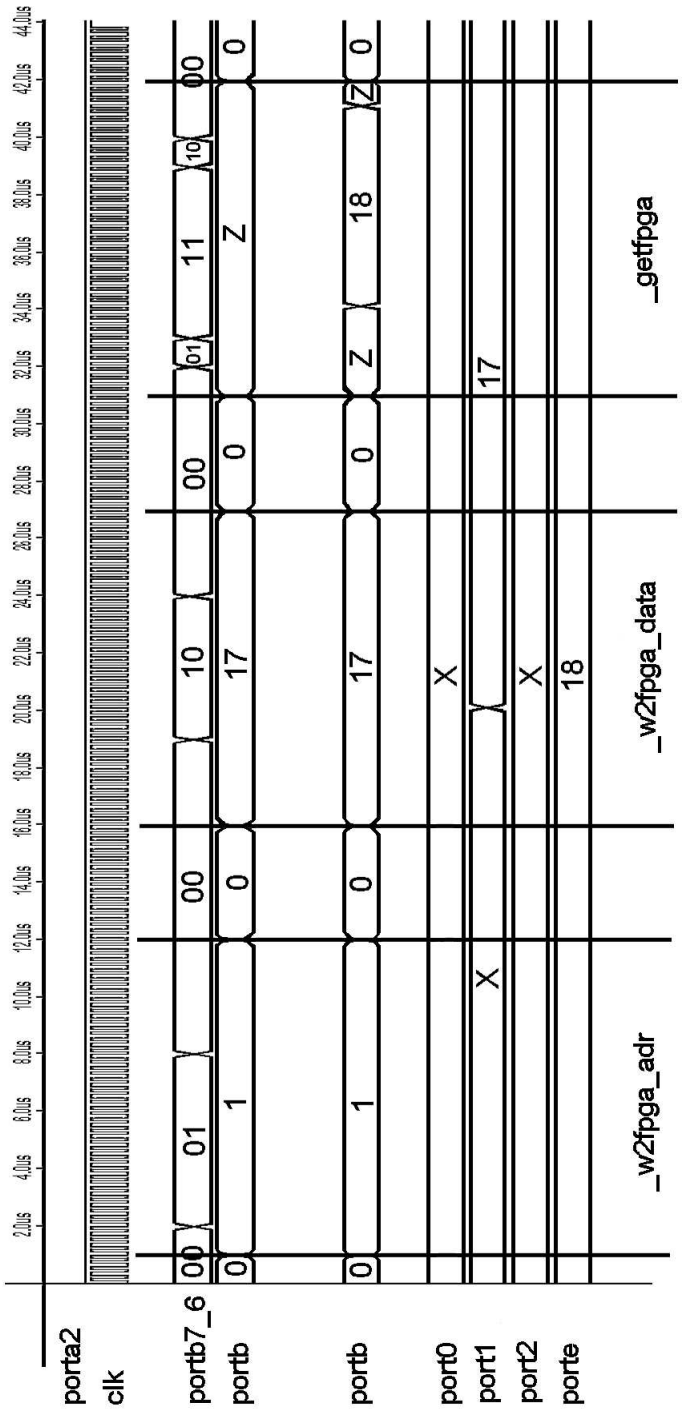

Abbildung 4.6: Timingdiagramm einer Befehlssequenz

Datei rechner.asm).

Zur Funktionsweise des Rechners:
Zuerst werden die Daten vom PIC zum Interface transportiert. An Port0 und Port1 werden die Operanden, an Port2 der Code für die auszuführende Operation gesendet. Liegen alle Werte am Interface vor, beginnt der Rechner je nach Wert an Port2 mit der Berechnung und an PortE liegt schließlich das Ergebnis vor. Dieses wird nun wieder zurück an den PIC gesendet.

Ausschnitt Quellcode rechner.vhd :

```
;-----------------------------------------------------------------
architecture rtl of rechner is

  component interface
    port( clk,porta2          : in       std_logic;
          portb7_6            : in       std_logic_vector(1 downto 0);
          portb               : inout    std_logic_vector(5 downto 0);
          porte               : in       std_logic_vector(5 downto 0);
          port0, port1, port2 : out      std_logic_vector(5 downto 0));
  end component;

  signal porte, port0, port1, port2 : std_logic_vector(5 downto 0);

begin

comp_interface : interface port map (clk, porta2, portb7_6, portb,
                                     porte, port0, port1, port2);

process
begin
  case port2 is
  when "000000" => porte <= port0 + port1;
  when "000001" => porte <= port0 - port1;
  when others  => porte <= "000000";
  end case;
end process;

end rtl;
;-----------------------------------------------------------------
```

Ausschnitt Quellcode rechner.asm:

```
;-----------------------------------------------------------------
          movlw     d'0'                ; Zahl1 zum FPGA
          call      _w2fpga_adr
          movfw     zahl1
          call      _w2fpga_data

          movlw     d'1'                ; Zahl2 zum FPGA
          call      _w2fpga_adr
          movfw     zahl2
          call      _w2fpga_data

          movlw     d'2'                ; Operation zum FPGA
          call      _w2fpga_adr
          movfw     operator
          call      _w2fpga_data

          call      _getfpga            ; hole Ergebnis

          call      _erg                ; speichere Ergebnis in
                                        ; erg1 und erg2

          movf      erg2,W              ; gib Ergebnis aus
          call      _ziffer
          call      _w2lcd

          movf      erg1,W
          call      _ziffer
          call      _w2lcd
;-----------------------------------------------------------------
```

Kapitel 5

Ausblick

Auf Grundlage der in dieser Studienarbeit entwickelten Erweiterungsplatinen, ist es nun möglich das Praktikum hardwarenah auch in Heimarbeit durchzuführen. Die Studierenden sind nicht mehr nur auf simulierte Ergebnisse angewiesen, sondern können ihre Versuche direkt an der Hardware testen. Da die Picee Entwicklungsumgebung hier als Grundlage dient, ist es nun auch möglich die Funktionsweise des PIC16F84 näher zu betrachten, da er durchgehend in allen Versuchen verwendet wird. Durch die Integration von FPGA und Microprozessor zu einer Testumgebung, können die Studierenden das Zusammenspiel verschiedener Hardware-Komponenten in diversen Versuchen nachvollziehen.
Für zukünftige Projekte bietet es sich an, weitere Module für die Picee Entwicklungsumgebung zu entwerfen und so die Anwendungsmöglichkeiten auf neue Technologien auszudehnen. Denkbar wäre z.B. ein aufsteckbares Interface, über welches zwei Picee-Boards miteinander kommunizieren können oder ein Interface, mit dessen Hilfe man den Datenverkehr auf Bussen verfolgen kann.
Ein interessantes Gebiet wäre z.B. auch die Steuer- und Regelungstechnik. So könnte man verschiedene Sensoren auf ein aufsteckbares Modul integrieren und beispielsweise bei Dämmerung eine Lichtschaltung aktivieren.

Literaturverzeichnis

[1] Altera. *Application Note: In-System Programmability in MAX Devices*. an095.pdf.

[2] Altera. *Application Note: Understanding MAX 7000 Timing*. an094.pdf.

[3] Altera. *Datasheet: EPM7128E & EPM7128S Dedicated Pin-Outs*. epm7128e.pdf.

[4] Altera. *Datasheet: MAX 7000 Programmable Logic Device Family*. max7000.pdf.

[5] Altera. *MAX+PLUS II Getting Started*. maxplusiigettingstarted.pdf.

[6] Altera. *Technical Brief 34: MAX 7000S Power Consumption*. tb34.pdf.

[7] Ashenden, P. J. *The VHDL Cookbook*. thevhdlcookbook.pdf.

[8] CadSoft. *Eagle Trainingshandbuch*. eagletutorial.pdf.

[9] El Camino. *DIGILAB picoMAX Handbuch*. picomax.pdf.

[10] Gijzen, B. *IC Programmer, Homepage: http://www.ic-prog.com*.

[11] Heinkel, U. *The VHDL Reference : A practical guide to computer-aided integrated circuit design including VHDL-AMS*. WILEY, 2000.

[12] Hitachi. *Datasheet: HD44780U (LCD-II)*. hd44780u.pdf.

[13] Mackensen, E. *Kurzeinführung Leiterplattendesign*. layoutregeln_full.pdf.

[14] Microchip. *Application Note: Crystal Oscillator Basics and Crystal Selection for rfPIC and PICmicro Devices*. an826.pdf.

[15] Microchip. *Application Note: Implementing a Table Read*. an556.pdf.

[16] Microchip. *Application Note: PICmicro Microcontroller Oscillator Design Guide*. an588.pdf.

[17] Microchip. *BASIC PIC16/17 OSCILLATOR DESIGN GUIDE*. fact001.pdf.

[18] Microchip. *Datasheet: PIC16F84*. pic16f84datasheet.pdf.

[19] Microchip. *MPASM Users Guide with MPLINK and MPLIB*. mpasmusersguide.pdf.

[20] Microchip. *MPLAB IDE, Simulator, Editor User's Guide*, 2000. mplabusersguide.pdf.

[21] Motorola. *Datasheet: OCTAL BUFFER/LINE DRIVER WITH 3-STATE OUTPUTS SN54/74LS244*. sn74244.pdf.

[22] Texas Instruments. *Datasheet: SN74HC14 HEX-Schmitt-Trigger Inverter*. sn74hc14.pdf.

Anhang A

Platinenlayouts

A.1 Erweiterungsplatine 1

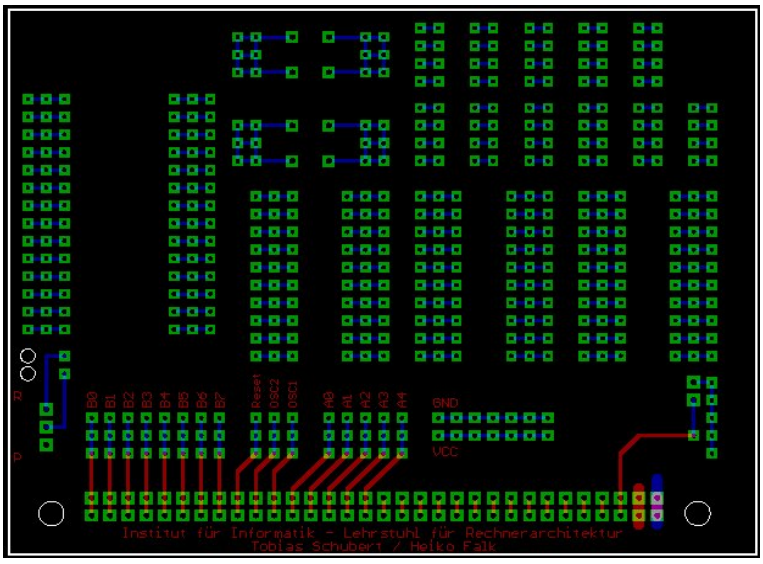

Abbildung A.1: Layout Erweiterungsplatine 1

A.2 Erweiterungsplatine 2

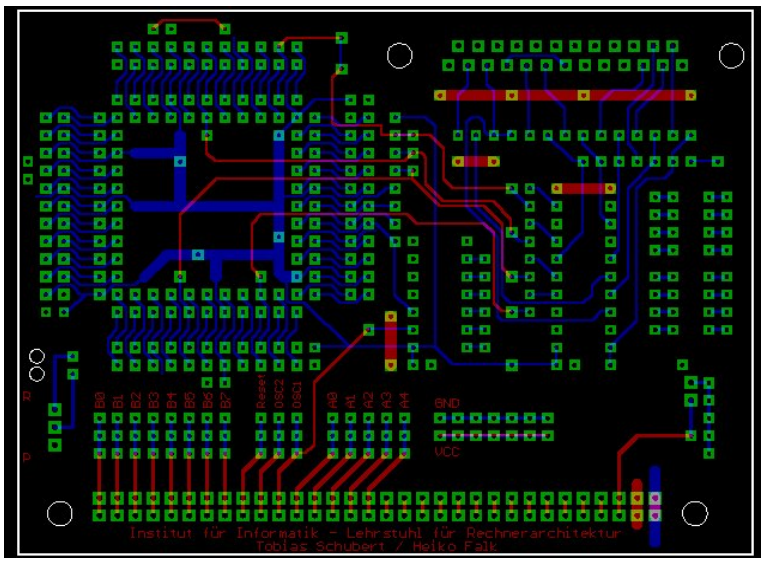

Abbildung A.2: Layout Erweiterungsplatine 2

Anhang B

FPGA-Sockel Pinbelegung

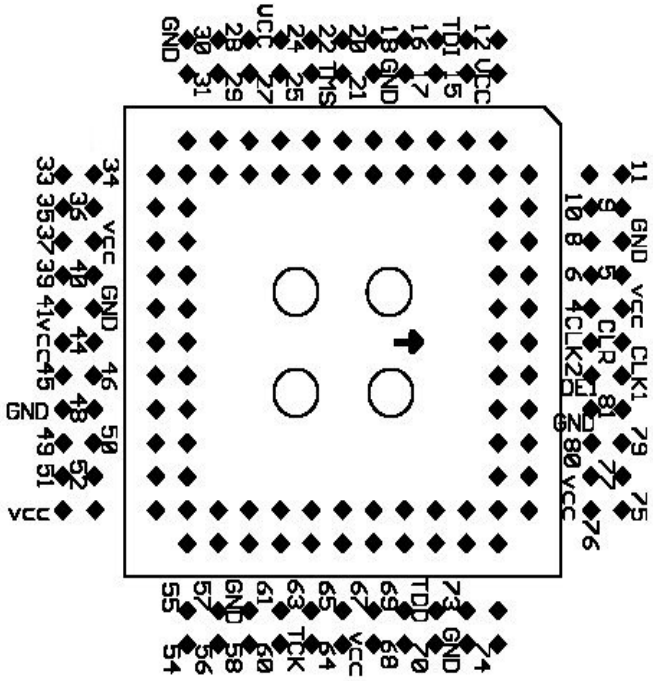

Abbildung B.1: FPGA-Sockel Pinbelegung

Anhang C

Inhalt der CD-Rom

Auf der beiliegenden CD-Rom befindet sich diese Dokumentation, Quelltexte der erstellten Programme und das Layout der Platinen. Im Verzeichnis Literatur befinden sich Datenblätter und Anleitungen die zur Erstellung der Studienarbeit verwendet wurden. Im Verzeichnis Software befinden sich die wichtigsten Programme.

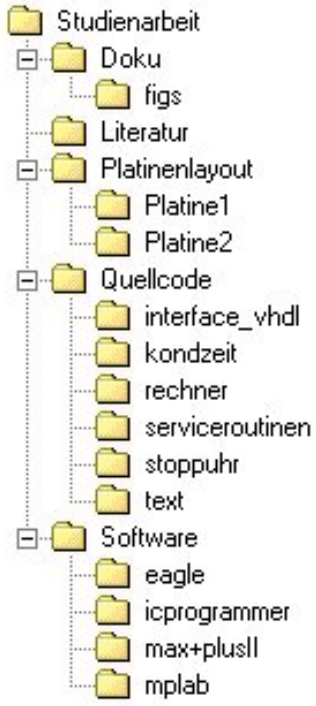

Abbildung C.1: Verzeichnisstrucktur der CD-Rom