Hard- und Softwaresimulation eines eingebetteten Microcontrollers unter Visual Micro Lab

Panagiotis Chatzichrisafis
(Email: contact@lysario.de)

31. Januar 2008

Die weit verbreitete Orcad® Design Suite von Cadence™ gibt die Möglichkeit einen Mikrocontroller in ein Layout für ein PCB einzusetzen, ein Schaltungsbild unter Schematic® zu erzeugen und sie ermöglicht auch spezifische digitale Signale zu erzeugen welche es erlauben die Hardwarereaktion externer Komponenten auf die Signale eines Mikrokontrollers zu testen. Eine Möglichkeit zusätzlich das Softwareprogramm in Zusammenarbeit mit der Hardware zu testen ist aber mit der Orcad® Design Suite leider nicht möglich.

Eine Frage die sich für den Entwickler stellt ist welche Simulationstools benutzt werden können um auch die Interaktion des auf dem Mikrokontroller ausgeführten Programms mit der angeschlossenen Hardware, bis zu einem gewissen Maß zu Simulieren, und dadurch das dynamische Verhalten des Systems zu untersuchen.

Für die AVR® Familie von ATMEL® existiert zusätzlich zur kostenpflichtigen Software Proteus VSM auch eine gänzlich frei erhältliche Entwicklungsumgebung VMLAB (Visual Micro Lab) welche eine Software-Hardware Kosimulation bis zu einem gewissen Maße erlaubt. Im folgenden Artikel soll anhand eines einfachen Beispiels eine kurze Einführung in die VMLAB Umgebung geliefert werden um dadurch die Fähigkeiten sowie die Limitationen von VMLAB zu erörtern.

Als Vorgabe für die Durchführung des Experimentes soll ein C Programm für einen ATMEL8 Mikrocontroller getestet werden. Zusätzlich dazu soll die Interaktion des Mikrocontrollers mit einer vorgeschriebenen Hardwareumgebung simuliert werden.

1 Installation der Entwicklungsumgebung

Die Entwicklungsumgebung für das folgende Beispiel besteht aus den Programmen
WINAVR™ (ausgesprochen WHENEVER) und VMLAB. Beide Programme lassen sich von den entsprechenden ([6],[1]) Internetseiten herunterladen und kostenlos installieren.

2 Problemstellung (Nutzung von Schaltern, Widerständen und Operationsverstärkern) unter VMLAB

In dem folgendem Beispiel soll der Microcontroller durch die Bedienung von zwei Schaltern einen Zähler jeweils Aufwärts (+) bzw. Abwärts (-) zählen können. Dadurch soll eine Ausgangsspannung erzeugt werden welche Analog zur Zählervariable sein soll. Die Zählervariable soll dabei den Wertebereich von 0 bis 7 annehmen und beim Erreichen des maximalen Wertes durch die Bedienung des (+) Schalters den maximalen Wert nicht überschreiten. Beim erreichen des kleinsten Wertes soll dieser durch die

Bedienung des (-) Schalters nicht unterschritten werden. Die Hardware welche zur Problemlösung dienen soll wird in der Abbildung 1 gegeben. Dabei werden die Anschlüße PD2 und PD3 als externe Interruptquellen benutzt, während die internen (Pull-Up) Widerstände an diesen Anschlüssen aktiviert werden müssen. Der INT0 Schalter soll dabei die (+) Funktion übernehmen und der INT1 Schalter die (-) Funktion. Die Widerstände sollen folgende Werte annehmen: $R_{N1} = 100k\Omega$, $R_f = 100k\Omega$, $R_{P0} = 400k\Omega$,$R_{P1} = 200k\Omega$, $R_{P2} = 100k\Omega$ und $R_P = 57k\Omega$.

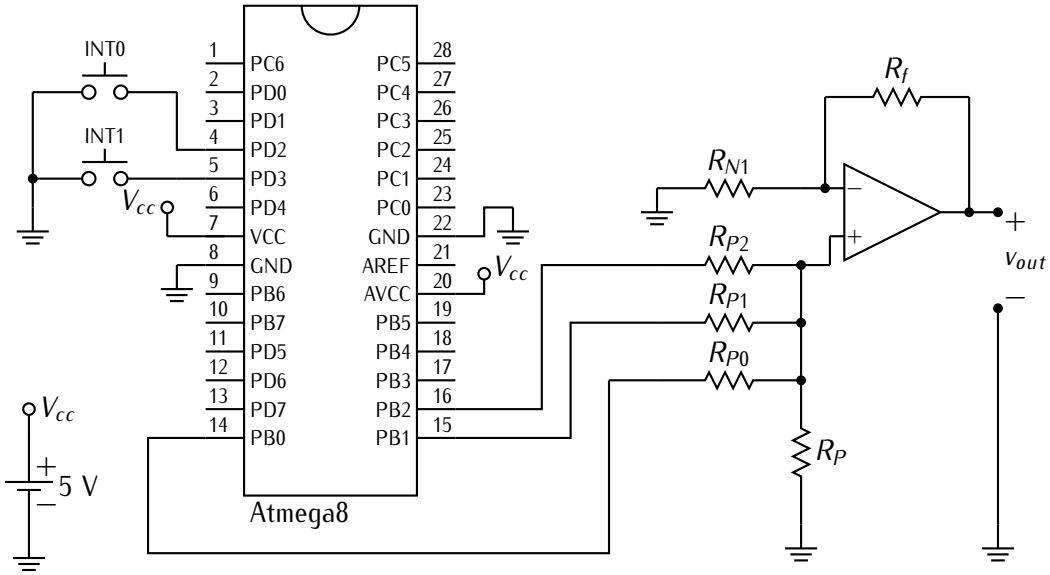

Abbildung 1: Schaltbild für das Hardware - Software Simulationsbeispiel unter VMLAB

3 VMLAB für die Beschreibung des Schaltbildes

Um das Schaltbild der Abbildung 1 in VMLAB zu realisieren muss zunächst VMLAB gestartet werden. Es sollte sich ein ähnliches Bild wie in der Abbildung 2 zeigen. Auf dem Arbeitsplatz des Hauptfensters von VMLAB sind die Fenster *Messages*, *Watch*, *Scope* und *Control Panel* zu sehen.

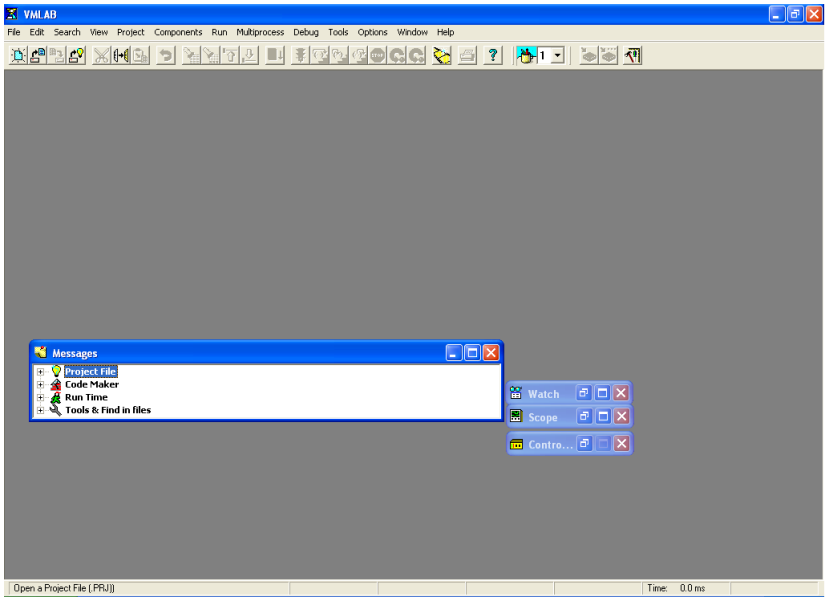

Abbildung 2: VMLAB Desktop nach dem Starten des Programms

Als nächstes muss ein neues Projekt erstellt werden. Dies geschieht durch die Auswahl des Menupunktes Project -> New Project . Dabei Öffnet sich ein Optionsmenu wie in der Abbildung 3

1. Unter Step 1 muss der gewünschte Ordner und Namen für das Projekt eingegeben werden.

2. Unter Step 2 muss man den gewünschten Microcontroller aussuchen. In diesem Beispiel wird der ATMEGA8 benutzt.

3. Unter Step 3 wird der GNU C Compiler ausgewählt und in GCC Path der Pfad eingeben in dem WINAVR™ installiert wurde. VMLAB benutzt als Default den Pfad „C:\WinAVR“. Wenn WINAVR™ in ein anderes Verzeichnis installiert wurde dann kann man diesen voreingestellten wert unter Options->Code Maker beim Reiter „GCC“ ändern. Einfach unter „Default Installation Directory“ den Pfad eingeben in dem WINAVR™ installiert wurde.

4. Unter Step 4 wird automatisch ein Name für die Haupt C Datei erstellt den man nach Wunsch ändern kann. Damit diese Datei auch in das Projekt aufgenommen wird muss man auf den Button „Add this“ drücken. Will man bereits existierende Dateien einbeziehen so wählt man diese mit dem Browse+add Button aus. Vergisst man auf „Add this“ zu drücken und ist auch sonst keine andere C Datei in das Projekt einbezogen so erscheint eine Fehlermeldung.

Ist alles richtig eingetragen erscheinen zwei neue Fenster auf dem Arbeitsplatz: Example1.prj und Example1.c . Die Datei Example1.c wird benutzt um den C Code einzutragen welcher dann von vom GCC AVR Compiler übersetzt wird und letztendlich vom Microprocessor ausgeführt werden soll. In der

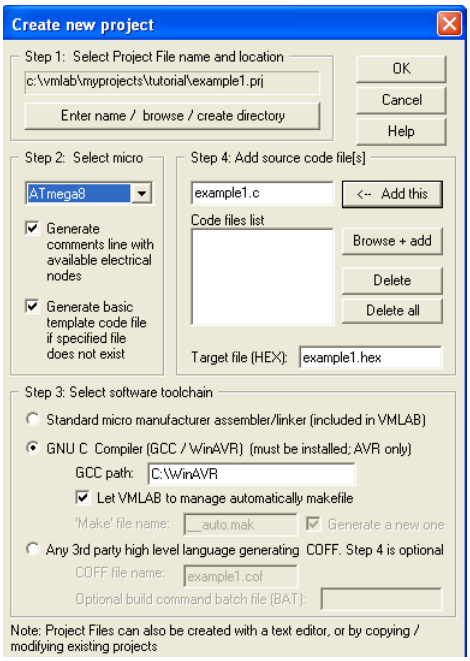

Abbildung 3: Das Menu zum erstellen eines neuen Projektes

Datei Example1.prj werden die Hardwarespezifischen Einstellungen definiert, und hier muss auch das Schaltbild der Abbildung 1 definiert werden.

Die von VMLAB erstellte Datei Example1.prj enthält folgenden Text:

```
; *************************************************************
; PROJECT:
; AUTHOR:
; *************************************************************

; Micro + software running
; -------------------------------------------------------------
.MICRO "ATmega8"
.TOOLCHAIN "GCC"
.GCCPATH   "D:\Program Files\WINAVR20071221"
.GCCMAKE   AUTO
.TARGET    "example1.hex"
.SOURCE    "example1.c"

.TRACE             ; Activate micro trace

; Following lines are optional; if not included
; exactly these values are taken by default
; -------------------------------------------------------------
.POWER VDD=5 VSS=0  ; Power nodes
.CLOCK 1meg         ; Micro clock
.STORE 250m         ; Trace (micro+signals) storage time

; Micro nodes: RESET, AREF, PB0-PB7, PC0-PC6, PD0-PD7, AC0, TIM10VF, ADC6, ADC7
; Define here the hardware around the micro
; -------------------------------------------------------------
```

VMLAB benutzt einen SPICE *ähnlichen* Syntax um Schaltungen zu beschreiben. Im VMLAB Syntax wird, im Gegensatz zum SPICE Syntax, ein Semikolon benutzt um einen Kommentar, der bis zum Zeilenende gültig ist, einzuleiten. Die ersten Zeilen in der Example1.prj Datei dienen demnach dazu den Namen des Autors und den Projekt Titel einzutragen.

Kommandos mit einem Punkt davor (.MICRO .TOOLCHAIN .GCCPATH .GCCMAKE .TARGET .SOURCE .TRACE .POWER .CLOCK .STORE .PLOT usw.) sind Befehle welche die Einstellungen und Ausgaben des VMLAB Simulators beeinflussen, so genannte Direktiven. So wird nach Befehl .MICRO der Mikrocontroller angegeben der Simuliert werden soll. Nach dem Befehl .POWER werden zwei der drei Vordefinierten Spannungsknoten (GND, VDD und VSS) definiert, wobei VDD die höchste positive Referenzspannung der Schaltung darstellt (Logische 1) und VSS die negativste Referenzspannung (Logische 0). Der Knoten GND oder 0 (null) ist die Referenz (Erde) zu der die Spannungen VSS / VDD negativ bzw. positiv sind und dieser kann nicht verändert werden.

Im Kommentar:

```
;Micro nodes:RESET,AREF,PB0-PB7,PC0-PC6,PD0-PD7,AC0,TIM1OVF,ADC6,ADC7
```

werden die vom System definierten Knoten des Mikroprozessors angegeben. Dabei sind außer TIM1OVF und AC0 alle anderen Knoten auch als Anschlüsse beim Tatsächlichen Mikrocontroller zu finden. AC0 und TIM1OVF sind so genannte *Pseudopins* welche den Inhalt von internen Registern wiedergeben.

- AC0 gibt den Ausgang des Analogen Komparators aus (ATMEGA8 Handbuch Seite 193-194) welches im Register ACSR im fünften bit abgespeichert wird.
- TIM1OVF hingegen ist das zweite bit im Timer/Counter Interrupt Flag Register (TIFR) das gesetzt wird, wenn vom TIMER1 ein Overflow erzeugt wird (ATMEGA8 Handbuch Seite 103).

Durch diese Pseudopins kann man also den Zustand der entsprechenden Register prüfen.

Der Anschluss VCC des Microcontrollers wird automatisch mit VDD verbunden und steht, wie aus der aufgeführten Kommentarzeile hervor geht, nicht zur Verfügung. Der GND Anschluss des ATMEGA8 wird ebenfalls automatisch mit dem vordefinierten Knoten GND verbunden und steht ebenfalls nicht zur Verfügung.

Will man nun den externen Schaltkreis der Abbildung 1 Simulieren so muss man die Auslegung der Komponenten in der Example1.prj Datei beschreiben.

Für die Definition eines Widerstandes wird folgender Syntax benutzt:

R[<Name>] <KnotenName1> <KnotenName2> <Wert>

Der Buchstabe R weist somit VMLAB darauf hin das in der folgenden Zeile die Position und der Wert eines Widerstandes definiert werden. Der Widerstand kann zusätzlich noch einen vom Anwender definierten Namen bekommen. Die Stelle, an der sich der Widerstand im Schaltbild befindet, wird durch die Zuweisung der zwei Knoten an denen der Widerstand angeschlossen ist definiert. Will man Beispielsweise einen $10k\Omega$ Widerstand, der den Namen 07 trägt, zwischen PB0 und PB7 schalten so kann man folgenden Befehl benutzen:

R07 PB0 PB7 10K

In dem Schaltbild von Abbildung 1 befinden sich sechs Widerstände.Vier davon sind am nicht invertierenden Ende des Operationsverstärkers angebracht. Knoten die nicht vordefiniert sind kann man einen beliebigen Namen zuordnen. Da in der Schaltung 1 die Knoten des invertierenden und nichtinvertierenden Eingangs sowie der Ausgang des Operationsverstärkers undefiniert sind wollen wir diesen Knoten entsprechend die Namen OPAMPM, OPAMPP und OPAMPOUT zuweisen. Folgende

Befehle müssen in die Example1.prj Datei Eingefügt werden um die Widerstände in die Schaltung einzufügen und Gleichzeitig die Knoten zu definieren:

```
RN1 GND OPAMPM 100k
RF OPAMPM OPAMPOUT 100k
RP0 PB0 OPAMPP 400k
RP1 PB1 OPAMPP 200k
RP2 PB2 OPAMPP 100k
RP  OPAMPP GND 57k
```

Um die Schaltung der Abbildung 1 vollständig zu beschreiben fehlen noch die Schalter welche die INT0 und INT1 Interrupts auslösen und der Operationsverstärker.

Die Schalter werden in VMLAB mit dem folgenden Syntax eingefügt:

```
K{0 - 9, A - F} <nodeName1> <nodeName2> [{NORMAL, LATCHED, MONOSTABLE(<timeValue>)}]
```

Schalter können im Gegensatz zu den Widerständen nicht beliebige Namen erhalten. Da Schalter über den *Control Panel* angesteuert werden, können nur Zeichen die auch im Control Panel zur Verfügung (0-9 und A-F) stehen, verwendet werden. Der Schalter wird dann durch das betätigen des entsprechenden Schalters auf dem *Control Panel* ausgelöst. Es werden drei Typen von Schaltern zur verfügung gestellt:

- „NORMAL“ ist dabei ein Schalter der so lange er betätigt wird geschlossen bleibt. Das Schlüsselwort „NORMAL“ sollte man dabei nicht benutzen. Diese Funktionsweise wird per Voreinstellung benutzt wenn nichts weiteres angegeben wird. Das angeben dieses Schlüsselwortes kann eventuell zu einer Fehlermeldung führen.
- „LATCHED“ ist ein Schalter der seinen Zustand ändert wenn man ihn drückt. Ist der Schalter auf und wird er gedrückt bleibt er zu. Ist er zu und wird gedrückt so wird der Schalter geöffnet.
- „MONOSTABLE“ ist der Schalter welcher für den angegebenen Zeitraum schließt. Die Zeiteinheit wird dabei direkt hinter dem Schlüsselwort in Klammern angegeben. Will man zum Beispiel einen solchen Schalter zwischen den Knoten A und B schließen der bei Betätigung des Kontrollfeldes F im *Control Panel*, 5 mikro sekunden geschlossen bleiben soll, so definiert man diesen Schalter durch den Befehl `KF A B MONOSTABLE(5u)`.

Der Syntax für die Zeiteinheiten von VMLAB ist der gleiche der von SPICE benutzt wird. So steht beispielsweise MEG oder meg für Mega,M oder m für milli, dabei werden Sekunden vorausgesetzt und Groß- sowie Kleinschreibung akzeptiert.

Für das einsetzen der Schalter des Schaltbildes 1 werden nun folgende Befehle benutzt:

```
K1 GND PD2 MONOSTABLE(15u)
K2 GND PD3 MONOSTABLE(15u)
```

Zusätzlich wollen wir den fehlenden Operationsverstärker einfügen. Der Operationsverstärker steht in VMLAB als so genanntes makro Modell zur Verfügung und der Aufruf dieses erfolgt ähnlich wie der Aufruf von benutzerdefinierten Modellen:

```
X[<instName>] OPAMP <nodePlus> <nodeMinus> <nodeOut>
```

Das X weist somit VMLAB daraufhin das ein Makromodell aufgerufen wird welches im Verlauf definiert wird. Nach dem X wird der vom Nutzer gegebene Name eingefügt und danach der Name

des Makromodells, hier OPAMP. Die darauf folgenden Werte geben die Knotennamen an, welche das Makromodell benutzt.

Der Operationsverstärker der Abbildung 1 kann somit mit dem Befehl

```
Xopamp1  OPAMP OPAMPP OPAMPM OPAMPOUT
```

eingefügt werden. Wobei der hier gewählte Name opamp1 nach belieben verändert werden kann.

Somit wären sämtliche Hardwarespezifischen Komponenten der Schaltung definiert. Um VMLAB mitzuteilen das man die Spannung an einen bestimmten Knoten im *Scope* Fenster beobachten will benutzt man die Direktive `.Plot`. Will man zum Beispiel den Spannungsausgang an den Knoten PB0 beobachten, so kann man den Befehl `.Plot V(PB0)` benutzen. Mehrere Knoten können auch angegeben indem auch die Spannungen der anderen Knoten angibt.

In diesem Beispiel sollen die Spannungen an den Knoten PD2,PD3,PB0-PB3, OPAMPM und OPAMPOUT beobachtet werden. Der Vollständige Code der nun in die Example1.prj Datei nach den von VMLAB generierten Einträgen eingefügt werden sollte ist folgender:

```
RN1 GND OPAMPM 100k
RF OPAMPM OPAMPOUT 100k
RP0 PB0 OPAMPP 400k
RP1 PB1 OPAMPP 200k
RP2 PB2 OPAMPP 100k
RP  OPAMPP GND 57k
K1 GND PD2 MONOSTABLE(15u)
K2 GND PD3 MONOSTABLE(15u)
Xopamp1  OPAMP OPAMPP OPAMPM OPAMPOUT
.Plot V(PD2) V(PB0) V(PB1) V(PB2) V(OPAMPM) V(OPAMPOUT)
```

4 Das Software Programm

Der nachfolgende Text soll die benötigten Code-Zeilen für das erfolgreiche ausführen des Simulationsbeispiels liefern. Eine Einführung in die C Programmierung mit AVR-GCC kann man auf den Seiten von www.mikrokontroller.net finden.

Die Software des Microcontrollers muss zunächst einmal zwei Interrupt Routinen bereitstellen welche beim bedienen des Schalters INT0 bzw. INT1 eine Zählervariable nach oben bzw. nach unten zählt.

Da diese Zählervariable ohnehin beim PORTB ausgegeben wird, benutzen wir demnach die vordefinierte Variable PORTB des Mikrocontrollers:

```
ISR(INT0_vect) {
            if (PORTB<=6){
            PORTB+=1;
            }
   }

ISR(INT1_vect){
            if (PORTB>=1){
            PORTB-=1;
            }
     }
```

Zusätzlich müssen

- der Data Direction Register von den D PINS auf Eingang gestellt werden,
- die internen Pull Up Widerstände von PD2 und PD3 bereitgestellt werden ,
- die Interrupts INT0 und INT1
- sowie die Interrupt Status Flag aktiviert werden um Interrupts generell zuzulassen.

Dies geschieht in der `main` Funktion mit den Befehlen:

```
//Set DataDirectionRegisterD as Input
DDRD=(1<<DDB0)|(1<<DDB1)|(1<<DDB2)|(1<<DDB3)|(1<<DDB4)|(1<<DDB5)|(1<<DDB6)|(1<<DDB7);
//Enable internal Pull up Resistors
PORTD= (1<<PD2)|(1<<PD3);
//General interrupt control register set up
GICR|=(1<<INT0)|(1<<INT1);
.
.
.
//Enable Interrupts Status register flag ...
sei(); //Set Interrupt Status Flag
```

Mehr über die möglichen Einstellungen für die externe Interrupts kann man im ATMEGA8 Handbuch auf Seite 66 erfahren.

Um ein Auf- bzw. Abzählen beim drücken der Schalter zu erreichen (fallende Flanke) müssen zusätzlich die ISC bits des MCUCR entsprechend eingestellt werden. Weiterhin müssen die PB Anschlüße als Ausgänge eingestellt werden:

```
//Enable Interrupts on INT0 and INT1 for falling edge in the MCUCR register
MCUCR|=(1<<ISC11)|(0<<ISC10)|(1<<ISC01)|(0<<ISC00);
//Set PortB to out
DDRB=(1<<DDB0)|(1<<DDB1)|(1<<DDB2)|(1<<DDB3)|(1<<DDB4)|(1<<DDB5)|(1<<DDB6)|(1<<DDB7);
```

Der ganze Code welcher nun in der Example1.c Datei zu finden sein sollte nimmt also folgende oder ähnliche Form an:

```
# include <avr\io.h>              // Most basic include files
# include <avr\interrupt.h>       // Add the necessary one here
// Define here the global static variables
//
// Interrupt handler example for INT0 and INT1
//
ISR(INT0_vect) {
             if (PORTB<=6){
             PORTB+=1;
             }
   }
ISR(INT1_vect){
             if (PORTB>=1){
             PORTB-=1;
             }
     }
// It is recommended to use this coding style to
// follow better the mixed C-assembly code in the
// Program Memory window
```

```
//
// *********************************************************
// Main program
//
int main(void) {
//Set DataDirectionRegisterD as Input
DDRD=(1<<DDB0)|(1<<DDB1)|(1<<DDB2)|(1<<DDB3)|(1<<DDB4)|(1<<DDB5)|(1<<DDB6)|(1<<DDB7);
//Enable internal Pull up Resistors
PORTD= (1<<PD2)|(1<<PD3);
//General interrupt control register set up
GICR|=(1<<INT0)|(1<<INT1);
//Enable Interrupts on INT0 and INT1 for falling edge in the MCUCR register
MCUCR|=(1<<ISC11)|(0<<ISC10)|(1<<ISC01)|(0<<ISC00);
//First set PortB to out
DDRB=(1<<DDB0)|(1<<DDB1)|(1<<DDB2)|(1<<DDB3)|(1<<DDB4)|(1<<DDB5)|(1<<DDB6)|(1<<DDB7);
//Enable Interrupts Status register flag ...
sei();
   while(1) {             // Infinite loop;
            }
}
```

5 Simulation des Systems

Um die Simulation zu starten führt man zunächst das Kommando „Build“ aus. Dieses ist unter Projekt->Build zu finden kann aber auch alternativ über die F9 Taste oder dem Menuleisten Button auf dem ein türkisfarbenes Dokument und ein Pfeil nach unten zu sehen sind, aufgerufen werden. Das *Message* Fenster sollte nun hervorspringen mit dem Hinweis „Success! all ready to run“. Am besten man arrangiert nun das *Scope* und *Control Panel* Fenster so dass beide zu sehen sind. Sollten beide auf der VMLAB Oberfläche nicht zu sehen sein so kann man die Fenster unter dem Menupunkt „View“ aufrufen. Die Simulation wird gestartet in dem man den Befehl Run-> Go/Continue aufruft oder alternativ die F5 Taste oder das Ampel-Icon auf der Menuleiste klickt. Durch drücken der Tasten 1 bzw. 2 auf dem *Control Panel* kann man nun beobachten wie die Ausgangsspannung des Operationsverstärkers variiert. Eine Demonstration von den Spannungsverläufen wird in Abbildung 4 gegeben.

VMLAB bietet weitere Features die das Debuggen des Codes erleichtern. Zum Beispiel kann man sich den Inhalt der Verschiedenen Register anzeigen lassen (Über View-> Register Flags oder View-> Data Memory usw.), den Code Zeile um Zeile durchführen (Run-> Step Over, Run-> Step Into etc.), Haltepunkte (Breakpoints) im Code Notebook setzen (indem man die grünen Vierecke neben den Programzeilen anklickt) oder bestimmte Variablen über das Watch Window beobachten. VMLAB bietet alles was man von einer anständigen Programmierumgebung erwartet. Dennoch weist VMLAB bezüglich der Hardware Simulation schwächen auf wobei zusätzlich zu der limitierten Bibliothek, die Einschränkung der Spannungen zwischen VSS und VDD eine davon ist. Jede Spannung an den verschiedenen Spannungsknoten wird auf VDD gesetzt wenn diese über VDD liegt. Ist die Tatsächliche Spannung kleiner als VSS so wird diese auf VSS gesetzt. Setzt man zum Beispiel $R_f = 500k\Omega$ ein so wird man beobachten dass der Ausgang des Operationsverstärkers bereits nach dreimaliger Betätigung der Taste „1“ bei 5Volt verbleibt und nicht mehr weitersteigt. Dies liegt daran das VMLAB alle Spannungen über VDD=5 Volt auf VDD setzt.

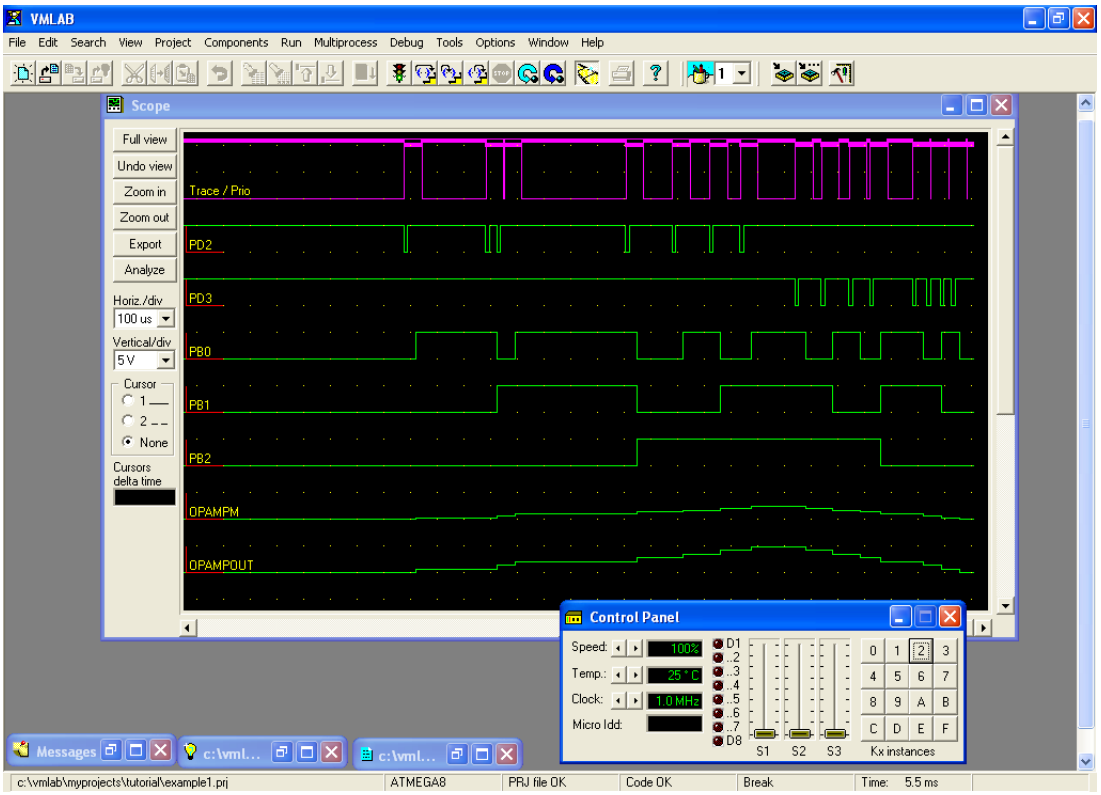

Abbildung 4: Beispiel des Spannungsverlaufs der Knoten (grüne Verlaufsgraphen von oben nach unten) PD2,PD3, PB0-PB3, OPAMPM und OPAMPOUT im *Scope* Fenster von VMLAB. Durch betätigen der Tasten 1 bzw. 2 im *Control Panel* werden die Interrupts aktiviert und der Ausgang an den PB Pins verändert.

6 Zusammenfassung

VMLAB bietet eine kostenfreie Entwicklungsumgebung für die Programmierung von Mikrocontrollern und die Simulation von einfachen Hardwareschaltungen in denen der Mikrokontroller integriert ist. Die Mikrocontroller können direkt in Assembler Code oder mit Hilfe eines externen C Compilers, wie den AVR-GCC, programmiert werden. Die Erstellung des Schaltbildes erfolgt dabei mit einem SPICE ähnlichen Syntax. Eine graphische Methode die Schaltung zu definieren wird nicht geliefert. Limitationen wie zum Beispiel die Einschränkung der Spannungswerte zwischen VDD und VSS müssen berücksichtigt werden um simulierte Daten korrekt zu interpretieren. Die Hardwarebibliothek von VMLAB lässt sich durch eigene, in C Verfasste, Makromodelle ergänzen und kann somit auch benutzt werden um eine breitere Palette von Funktionen zu Testen. Auch wenn die Hardwaresimulation von VMLAB nicht perfekt ist, können einfache Modelle erstellt werden um das dynamische Verhalten des eingebetteten Systems zu Untersuchen. Die Entwicklungsumgebung für die Programmerstellung und Testung ist hingegen sehr einfach zu bedienen und bietet im Vergleich zu anderen frei erhältlichen Systemen einen großen Komfort.

7 Über dieses Dokument

Dieses Dokument wurde mit LaTeX erstellt. Das elektrische Schaltbild der Abbildung 1 wurde mit Hilfe der M4 Makromodelle von J. D. Aplevich und DPIC.EXE erstellt [2]. Zusätzlich wurden die Pakete tikz, graphicx, graphics, hyperref, babel, xspace, fontenc, inputenc, color, kurier und a4wide mit LaTeX benutzt um das finale Dokument in PDF mittels pdflatex zu erstellen. Mehr Informationen über TeX und LaTeX und entsprechende Pakete kann man auf den Seiten der CTAN finden [4].

Literatur

[1] AMCTools - VMLAB. Website. http://www.amctools.com/download.htm, letzter Zugriff am 31.01.2008.

[2] Circuit Macros Version 6.22. Website. http://www.ece.uwaterloo.ca/~aplevich/Circuit_macros/index.html, letzter Zugriff am 31.01.2008.

[3] Labcenter Electronics Homepage - Proteus VSM. Website. http://www.labcenter.co.uk/products/vsm_overview.cfm, letzter Zugriff am 31.01.2008.

[4] The Comprehensive TeX Archive Network. Website. http://www.ctan.org/, letzter Zugriff am 31.01.2008.

[5] The SPICE Home Page. Website. http://www.amctools.com/download.htm, letzter Zugriff am 31.01.2008.

[6] WINAVR. Website. http://winavr.sourceforge.net/, letzter Zugriff am 31.01.2008.

[7] mikrocontroller.net - AVR. Website, 2006. www.mikrocontroller.net, letzter Zugriff am 31.01.2008.

[8] Advanced Micro Tools. *VMLAB User-defined Components Programming Interface*, Januar 2005. Das Dokument kann über das Hilfe Menu von VMLAB aufgerufen werden.

[9] ATMEL, Atmel Corporation 2325 Orchard Parkway San Jose, CA 95131 USA. *8-bit AVR with 8K Bytes In-System Programmable Flash*, 8 2007. http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf, letzter Zugriff am 31.01.2008.