Labjournaal

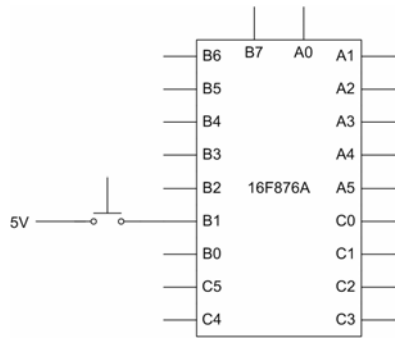
Practicum ‘Embedded Systems’

Mark van Driel
Vrije Universiteit, Amsterdam
mvdriel@cs.vu.nl
studentnummer (VU): 1101137
studentnummer (UvA): 192767

Lab Assignment I: Polling & Interrupts and measuring time

Assignment 1a: Polling.

Connect a pushbutton to pin B1 of the microcontroller. Test the status of this pin frequently and sent the message "Pushbutton 1 pressed" to the screen of the PC every time you push the button. Sent this message once, independent the time you push the button. Remark: A bad solution is to wait for input (PIN_B1) is high and after that to wait for input (PIN_B1) is low. If you implement this bad solution a satisfied implementation of assignment 1b is not possible. See page 45 of the "Reference Manual" to return the state of the indicated pin.

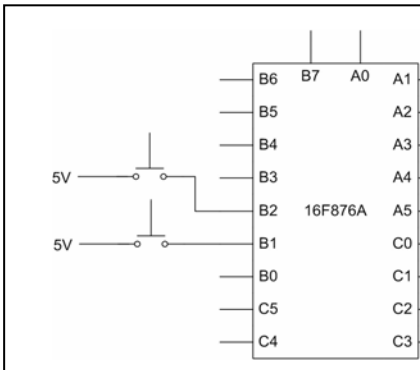


Figuur 1: Polling

Een drukknop heb ik aangesloten op pin B1 van de microcontroller, zoals te zien is in figuur 1. In de broncode (zie: appendix A) is te zien dat continue de status van pin B1 wordt bekeken. Wanneer deze uitgelezen status veranderd is ten opzichte van de vorige keer uitlezen dan wordt deze nieuwe status opgeslagen. Als deze status positief is dan is de drukknop ingedrukt en wordt dit aan de gebruiker gemeld.

Assignment 1b: Polling two devices.

Connect a second pushbutton to a free pin of the microcontroller (don't use pin B0, B7, B6 and B3). Test the status of this pin frequently and sent the message "Pushbutton 2 pressed" to the screen of the PC every time you push the button. The order you push the buttons must be irrelevant, every button sends his own message once.

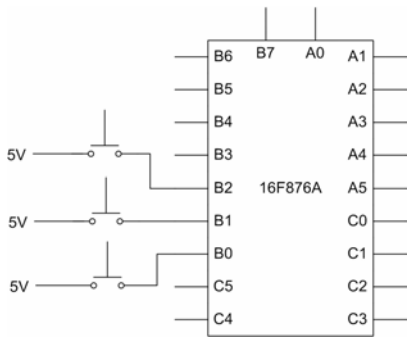


Figuur 2: Polling two devices

In figuur 2 is te zien dat ik één drukknop heb aangesloten op pin B1 en één op pin B2 van de microcontroller. Omdat er in deze opgave twee invoermogelijkheden zijn, moeten continue de waardes van twee pinnen worden uitgelezen. Dit gebeurt door in een oneindige loop zowel de status van de ene pin als van de andere pin uit te lezen. Hierbij mag niet worden gewacht op een bepaalde waarde bij het uitlezen van de waarde van een van beide pinnen, omdat anders een verandering van de andere pin niet opgemerkt kan worden. Bij de vorige opgave had dit nog wel gekund, omdat daar slechts één invoermogelijkheid was. De code is te vinden in appendix B.

Assignment 1c: Polling & Interrupt.

Connect a third pushbutton to pin B0. Write an interrupt handler that sent the message "Pushbutton 3 pressed" every time you press the button. The interrupt vector of this pin is "#int_ext". Try to interrupt the message "Pushbutton 1 pressed". See page 21 and 101 of the "Reference Manual".



Figuur 3: Polling & Interrupt

Zoals in figuur 3 te zien is, wordt er een derde drukknop aangesloten op pin B0 van de microcontroller. Bij de vorige opgave maakte het nog niet uit voor welke pin werd gekozen (mits deze maar gelijk was aan de pin die in de broncode werd gebruikt), maar nu moet het pin B0 zijn, omdat er in deze opgave gebruik gemaakt wordt van een interrupt handler. Om gebruik te kunnen maken van interrupts, moeten softwarematig interrupts aangezet worden. Dat gebeurt met de volgende regels:

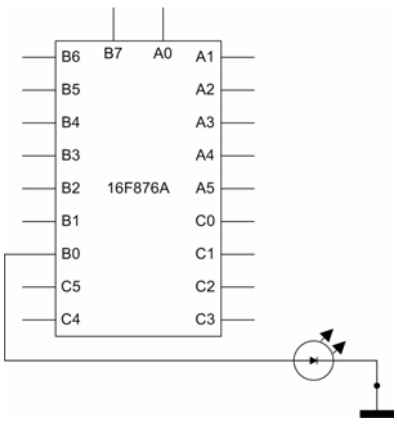
```
enable_interrupts(INT_EXT); // make specific interrupt (extern) possible

enable_interrupts(GLOBAL);  // make interrupts possible
```

Er moet natuurlijk ook een interrupt handler aanwezig zijn. Dit is een stuk code dat slechts dan wordt uitgevoerd wanneer er een bepaalde gebeurtenis plaatsvindt. In dit geval is deze speciale gebeurtenis een verandering van de status van pin B0. In mijn broncode (zie: Appendix C) is de interrupt handler boven de “main”-methode te vinden.

Assignment 1d: Signal generation.

Generate a square wave signal with a period of 2 seconds to steer a LED. Don't make use of the function "delay". See page 48/49, 57 and 101 of the "Reference Manual".



Figuur 4: Signal generation

In deze opgave maak ik weer gebruik van een interrupt handler. In dit geval moet de handler echter niet worden uitgevoerd wanneer bepaalde invoer plaatsvindt, maar op basis van een klok. De handler wordt dertig keer per seconde aangeroepen, vervolgens wordt in de code van de handler een teller bijgehouden waarmee wordt bepaald of er dertig events zijn geweest en er dus actie moet worden ondernomen. Actie ondernemen is niets meer dan het aanzetten van de LED, wanneer deze uit is en vice versa.

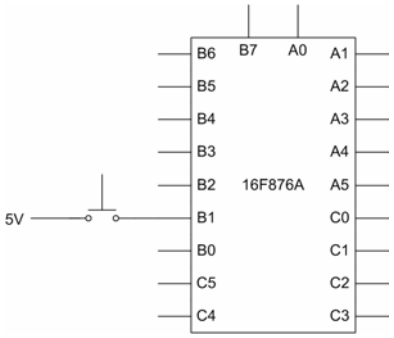
In de broncode, die te vinden is in Appendix D is te zien dat het regelen van de klok en interrupt handler gebeurt in de volgende code:

```
set_rtcc(0);                                  // set RTCC
setup_counters(RTCC_INTERNAL, RTCC_DIV_256); // initialise RTCC counter
```

```
  enable_interrupts(RTCC_ZERO);                 // make RTCC interrupt
possible
  enable_interrupts(GLOBAL);                    // make interrupts possible
```

Assignment 1e: Time measurement and fault tolerance.

Push a button during 1 to 10 seconds. Measure the time you push the button with an accuracy of better then 0.5 % and sent the result to the PC. Hint: Make use of the 16 bit timer of the microcontroller. Make only use of "floats" in the printf-statement. Make an estimation of the accuracy of your implementation.



Figuur 5: Time measurement and fault tolerance

Ook in deze opgave maak ik gebruik van de interrupt handler. Deze keer is de taak van de handler zeer eenvoudig, namelijk het ophogen van een teller. De teller “int_count” houdt het aantal interrupts bij sinds de laatse keer dat de drukknop werd ingedrukt. Op het moment dat de drukknop vervolgens wordt losgelaten, kan heel eenvoudig de tijdsduur van het indrukken worden bepaald door het aantal interrupts (“int_count”) te delen door het aantal interrupts per seconde. Omdat de RTCC counter is geïnitialiseerd met de waarde “RTCC_DIV_32” is het aantal interrupts per seconde gelijk aan: 80000000/(4*32*256) ≈ 244. Hierdoor komt de gemeten nauwkeurigheid op 1/244 ≈ 0,004 seconde. Bij een tijdsduur van 10 seconde komt de nauwkeurigheid nu uit op 0,004/10*100 = 0,04 %. Bij een tijdsduur van 1 sec komt deze uit op: 0,004/1*100 = 0,4%.

Mijn broncode van deze opgave is te vinden in appendix E.

Appendix A: broncode opdracht 1a

```
////////////////////////////////////////////////////////////////////////
// Filename:    opdr11.c
// Revision:    1.0
// Created:     17-4-2002
// Project:     Pidac876
// Device:      PIC16F876
// Development: MPLAB / CCS PCM
// Tab setting: 2
// Author:      Mark van Driel
// Description: Polling
////////////////////////////////////////////////////////////////////////

#include <D:\PICC\EXAMPLES\16F876.H>
#include <D:\PICC\EXAMPLES\CTYPE.H>

// Inform the compiler the clock frequency is 8 MHz
#use delay(clock=8000000)

// Setup the RS232 communication
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,  bits=8)

main()
{ short int state = 0;     // variable to store the current state
  short int new_state = 0; // variable to store the new state

  delay_ms(10);            // initialisation

  do
  {                        // do forever
    // get the state of pin B1
    new_state = input(PIN_B1);

    // see the state op pin B1 is changed since last check
    if(new_state != state)
    { // update the state with the new state
      state = new_state;

      // check the state of the pin
      if(state)
      { // the state of the pin is down
        printf("Pushbutton 1 pressed\n\r");
      }
    }
  }
  while(TRUE);
}
```

Appendix B: broncode opdracht 1b

```
////////////////////////////////////////////////////////////////////////
// Filename  :      opdr1b.c
// Revision  :      1.0
// Created   :      17-4-2002
// Project   :      Pidac876
// Device           :      PIC16F876
// Development      :      MPLAB / CCS PCM
// Tab setting      :      2
// Author           :      Mark van Driel
// Description      :      Polling two devices
////////////////////////////////////////////////////////////////////////

#include <D:\PICC\EXAMPLES\16F876.H>
#include <D:\PICC\EXAMPLES\CTYPE.H>

// Inform the compiler the clock frequency is 8 MHz
#use delay(clock=8000000)

// Setup the RS232 communication
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,  bits=8)

main()
```

```
{ short int state_b1 = 0;     // variable to store the current state of pin B1
  short int new_state_b1 = 0; // variable to store the new state of pin B1
  short int state_b2 = 0;     // variable to store the current state of pin B2
  short int new_state_b2 = 0; // variable to store the new state of pin B2

  delay_ms(10);               // initialisation

  do
  {                           // do forever
    // get the state of pin B1
    new_state_b1 = input(PIN_B1);

    // see the state op pin B1 is changed since last check
    if(new_state_b1 != state_b1)
    { // update the state with the new state
      state_b1 = new_state_b1;

      // check the state of the pin
      if(state_b1)
      { // the state of the pin is down
        printf("Pushbutton 1 pressed\n\r");
      }
    }

    // get the state of pin B2
    new_state_b2 = input(PIN_B2);

    // see the state op pin B2 is changed since last check
    if(new_state_b2 != state_b2)
    { // update the state with the new state
      state_b2 = new_state_b2;

      // check the state of the pin
      if(state_b2)
      { // the state of the pin is down
        printf("Pushbutton 2 pressed\n\r");
      }
    }
  }
  while(TRUE);
}
```

Appendix C: broncode opdracht 1c

```
////////////////////////////////////////////////////////////////////////
// Filename:    opdr1c.c
// Revision:    1.0
// Created:     17-4-2002
// Project:     Pidac876
// Device:      PIC16F876
// Development: MPLAB / CCS PCM
// Tab setting: 2
// Author:      Mark van Driel
// Description: Polling & Interrupt
////////////////////////////////////////////////////////////////////////

#include <D:\PICC\EXAMPLES\16F876.H>
#include <D:\PICC\EXAMPLES\CTYPE.H>

// Inform the compiler the clock frequency is 8 MHz
#use delay(clock=8000000)

// Setup the RS232 communication
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,  bits=8)

#INT_EXT            // this the handler of INT_EXT interrupts
extern_interrupt_handler()
{
  // get the state of pin B0
  if(input(PIN_B0))
  {
    printf("Pushbutton 3 pressed\n\r");
  }
}

main()
```

```
{ short int state_b1 = 0;     // variable to store the current state of pin B1
  short int new_state_b1 = 0; // variable to store the new state of pin B1
  short int state_b2 = 0;     // variable to store the current state of pin B2
  short int new_state_b2 = 0; // variable to store the new state of pin B2

  delay_ms(10);               // initialisation

  enable_interrupts(INT_EXT); // make specific interrupt (extern) possible
  enable_interrupts(GLOBAL);  // make interrupts possible

  do
  {                                                 // do forever
    // get the state of pin B1
    new_state_b1 = input(PIN_B1);

    // see the state op pin B1 is changed since last check
    if(new_state_b1 != state_b1)
    { // update the state with the new state
      state_b1 = new_state_b1;

      // check the state of the pin
      if(state_b1)
      { // the state of the pin is down
        printf("Pushbutton 1 pressed\n\r");
      }
    }

    new_state_b2 = input(PIN_B2);
    if(new_state_b2 != state_b2)
    { // update the state with the new state
      state_b2 = new_state_b2;

      // check the state of the pin
      if(state_b2)
      { // the state of the pin is down
        printf("Pushbutton 2 pressed\n\r");
      }
    }
  }
  while(TRUE);
}
```

Appendix D: broncode opdracht 1d

```
////////////////////////////////////////////////////////////////////////
// Filename  :      opdr1d.c
// Revision  :      1.0
// Created   :      17-4-2002
// Project   :      Pidac876
// Device            :      PIC16F876
// Development       :      MPLAB / CCS PCM
// Tab setting       :      3
// Author            :      Mark van Driel
// Description       :      Signal generation
////////////////////////////////////////////////////////////////////////

#include <D:\PICC\EXAMPLES\16F876.H>
#include <D:\PICC\EXAMPLES\CTYPE.H>

// Inform the compiler the clock frequency is 8 MHz
#use delay(clock=8000000)

// Setup the RS232 communication
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7,  bits=8)

#define INTS_PER_SECOND 30       // (80000000/(4*256*256))

unsigned int int_count = 0;      // counter of interrupts
int led_state = 0;         // led is off by default

#INT_RTCC           // this the handler of INT_RTCC interrupts
timer_interrupt_handler()
{ // update counter
  int_count++;

  if(int_count > (INTS_PER_SECOND))
```

```
  { // 1 second elapsed
    if(led_state)
    { // turn the led off
      output_low(PIN_B0);

      // update led state
      led_state = 0;
    }
    else
    { // turn the led on
      output_high(PIN_B0);

      // update led state
      led_state = 1;
    }
    // reset the counter
    int_count = 0;
  }
}

main()
{ delay_ms(10);               // initialisation

  set_rtcc(0);                                  // set RTCC
  setup_counters(RTCC_INTERNAL, RTCC_DIV_256); // initialise RTCC counter
  enable_interrupts(RTCC_ZERO);                 // make specific interrupt (RTCC)
possible
  enable_interrupts(GLOBAL);                    // make interrupts possible

  do
  {  // do forever
     // do nothing
     // the interrupt-handler will do everything
  }
  while(TRUE);
}
```