



**UNIVERSAL CHARGER**  
Firmware source code

Revision 1.0

Page 1 of 15

# UNIVERSAL CHARGER

Firmware source code

[www.seven-segments.com](http://www.seven-segments.com)



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 2 of 15

```
program Universal;

//=====
// PIC pin assignment
//
// PORTA.0 -> AN0 (battery voltage multiplied by R6/(R5+R6)) (analog IN)
// PORTA.1 -> AN1 (current pickup) (analog IN)
// PORTA.2 -> spare (OUT)
// PORTA.3 -> spare (Analog IN may be temperature reading)
// PORTA.4 -> BUZZER (OUT)
// PORTA.5 -> spare (OUT)
//
// PORTB.0 -> FAN cooler (OUT)
// PORTB.1 -> D4 data bus for display (OUT)
// PORTB.2 -> D5 data bus for display (OUT)
// PORTB.3 -> D6 data bus for display (OUT)
// PORTB.4 -> D7 data bus for display (OUT)
// PORTB.5 -> UP key (IN)
// PORTB.6 -> DOWN key (IN)
// PORTB.7 -> OK/ENTER key (IN)
//
// PORTC.0 -> spare (OUT)
// PORTC.1 -> PWM2 DISCHARGE (OUT)
// PORTC.2 -> PWM1 CHARGE (OUT)
// PORTC.3 -> RS register/data for display (OUT)
// PORTC.4 -> RW read/write for display (OUT)
// PORTC.5 -> E enable for display (OUT)
// PORTC.6 -> TX RS232 transmit (OUT)
// PORTC.7 -> RX RS232 receive (IN)
//=====

//=====
// Constants
//=====

const repeat_period = 16; //repeat key period -> 16*5ms = 80ms (about 12 press/sec)
      repeat_initial = 200; //initial delay for start autorepeat -> 200*5ms = 1000ms
      key_pressed = 10; //time to hit a key for a valid action -> 10*5ms = 50ms
      ctrl_period = 1; //CV charge BW control 1*5ms -> 200Hz BW (max)

// EEPROM Memory map

// 247..255 //spare locations
uni_profile = 246; //Position for the selected profile in EEPROM
uni_maxcharge = 245; //Maximum allowable charge current 255 -> 255A
uni_maxdischarge = 244; //Maximum allowable discharge current 255 -> 255A
uni_beep = 243; //beep tone semiperiod 0..255
uni_r6h = 242; //
uni_r6l = 241; //R5=0..65535 in Ohm
uni_r5h = 240; //
uni_r5l = 239; //R6=0..65535 in Ohm
uni_currh = 238; //
uni_currl = 237; //current pick up sensitivity 0..65535 -> 65535uV/A
uni_mode = 236; //idle, charge or discharge mode, see below constants
uni_seconddline = 220; //second hello line
uni_firstline = 204; //first hello line
0..203 //space for the 12 profiles

// Single profile map

profilelen = 17; //single profile length

pchemistry = 0; //parameters position in a specific profile: Chemistry
pcapacity = 1; //capacity in mAh/100, 255 -> 25500mAh
pncells = 2; //Number of cells
pcharge = 3; //charge in capacity units 255 -> 25.5*capacity
pdischarge = 4; //discharge in capacity units 255 -> 25.5*capacity
pinhibit = 5; //Inhibit in minutes for delta peak check: 255 -> 255 min

pcutoff_nicd = 6; //Cutoff in discharge for NiMh
pcutoff_nimh = 7; //Cutoff in discharge for NiCd
pcutoff_lipo = 8; //Cutoff in discharge for LiPo
pcutoff_sla = 9; //Cutoff in discharge for SLA
pdeltav_nicd = 10; //deltav in charge for NiCd
pdeltav_nimh = 11; //deltav in charge for NiMh
pmaxv_lipo = 12; //max voltage in charge for LiPo
pmaxv_sla = 13; //max voltage in charge for SLA
pfinalcurr_lipo = 14; //final current (% of initial) for LiPo
pfinalcurr_sla = 15; //final current (% of initial) for SLA
ptimeout = 16; //Timeout for charge

// Constants for battery identification

nicd = 0; //battery codes: Nickel Cadmium
nimh = 1; //Nickel Metal Hydride
lipo = 2; //Lithium Polymer
sla = 3; //Sealed Lead Acid

// Constants for charger action identification

idle = 0; //idle phase (for interrupt routine)
discharge_cc = 1; //discharge at constant current (for interrupt routine)
charge_cc = 2; //charge at constant current (for interrupt routine)
charge_cv = 3; //charge at constant voltage (for interrupt routine)

// Constants for charger last action identification

mode_idle = 0; //idle mode for power interrupt
mode_charge = 1; //charge phase for power interrupt
mode_discharge = 2; //discharge phase for power interrupt

//=====
// Display messages
//=====

msg_hello1='seven-segments.';
msg_hello2='www. .com';
msg_main1='Task select: ';
msg_main2='Profile select ';
msg_main3='Batt. charge ';
msg_main4='Batt. discharge ';
msg_main7='Profile change ';
msg_main8='PC management ';
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 3 of 15

```
msg_main9='Volt calibr.  ';\nmsg_main10='Ampere calibr.  ';\nmsg_change1='Chemistry:  ';\nmsg_change2='Battery type  ';\nmsg_change3='NiCd';\nmsg_change4='NiMh';\nmsg_change5='LiPo';\nmsg_change6='SLA  ';\nmsg_change7='Capacity:  ';\nmsg_change8='mAh per cell  ';\nmsg_change9='Cells:  ';\nmsg_change10='Number of cells  ';\nmsg_change11='Charge:  ';\nmsg_change12='mult. for capac.  ';\nmsg_change13='Discharge:  ';\nmsg_change15='Name:  ';\nmsg_pcman1='PC serial link..';\nmsg_discharge='Dsch.';\nmsg_charge='Chrg.';\nmsg_end='End!';\nmsg_display='Pack!';\nmsg_init='Initializing...  ';\n\n//=====\n// Variables\n//=====\n\nvar i:byte; //generic interrupt\n    temp:word;\n    tempi:integer;\n\n    ii,j,k,l,m,n,act:byte; //generic program\n    tw,tw1,tw2,vrat,pk,maxcap,ilim:word;\n    tlw,tlw1:longint;\n    twi:integer;\n    tmpword:string[5];\n    tmpbyte:string[3];\n\n    repeat_counter:byte; //keys\n    repeat_flag:byte;\n    repeat_k1,repeat_k2,repeat_k3:byte;\n    keys:byte;\n\n    msec,sec,minu:byte; //time\n\n    count_slow:byte; //analog values\n    fast_voltage:word;\n    slow_voltage:word;\n    acc_voltage:longint;\n    fast_current:word;\n    slow_current:word;\n    acc_current:longint;\n\n    action:byte; //task to be performed: charge, discharge...\n    duty_pwm_charge,duty_pwm_discharge:integer; //pwm control (mosfet drive)\n    target_current:word; //target current for constant current phases\n    target_voltage:word; //target voltage for constant voltage phases\n    zero_current:word;\n    mah:longint; //amount of charge\n\n    menupos:byte; //top menu position\n    selprof:byte; //profile selection index\n    selitem:byte; //item selected within the change profile routine\n    profile_point:byte; //current profile selected\n\n    eepar,eelim,eeres:byte; //eeprom variables\n\n    phase,q,ps:byte; //current phase for power interrupt handling\n    pt,pw:word;\n    bdelay:byte;\n    cbeep,dbeep,ebeep,kk:byte;\n    calibra:byte;\n\n    tempib:integer;\n    slow_voltageb,pwmb:word;\n    mahb:longint;\n    pwm_err:byte;\n\n    ctrl_co,ctrl_flag:byte;\n\n//=====\n// PWM adjust\n//=====\n\nprocedure setpwm;\nbegin\n    CCP1R1L:=duty_pwm_charge shr 2; //PWM1 is physically connected to the charge FET\n    CCP1CON.5:=duty_pwm_charge.1;\n    CCP1CON.4:=duty_pwm_charge.0;\n\n    CCP2R1L:=duty_pwm_discharge shr 2; //PWM2 is physically connected to the discharge FET\n    CCP2CON.5:=duty_pwm_discharge.1;\n    CCP2CON.4:=duty_pwm_discharge.0;\nend;\n\n//=====\n// Interrupt, every 5msec -> keyboard, sampling ANx, control loops for voltage and current, TX UART\n//=====\n\nprocedure interrupt;\nbegin\n    if INTCON.2 = 1 then\n    begin\n        TMR0 := 0x3D; //an interrupt every 5ms\n        INTCON.2:=0;\n    end;\n\n//=====\n// Internal clock\n//=====\n\n    inc(msec); //clock\n    if (msec=200) then
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 4 of 15

```
begin
  msec:=0;
  inc(sec);
end;
//one second is 200 interrupts

if (sec=60) then
begin
  sec:=0;
  inc(minu);
end;
//one minute is 60 seconds

//=====
// This unit controls the bandwidth for CV charge
//=====

inc(ctrl_co);
if (ctrl_co=ctrl_period) then
begin
  ctrl_co:=0;
  ctrl_flag:=1;
end
//When this flag is high the CV control will be performed
else
begin
  ctrl_flag:=0;
end;

//=====
// Keyboard management
//=====

inc(repeat_counter);
if (repeat_counter=repeat_period) then
begin
  repeat_counter:=0;
  repeat_flag:=1;
end
//when this flag is high is simulated a "key hit"
else
begin
  repeat_flag:=0;
end;

if PORTB.6 = 0 then
begin
  //if one key has been pressed, (this is the "DOWN" or "-" key)
  if (repeat_k1<repeat_initial) then inc(repeat_k1);
  //increase the "key pressed" counter
  if (repeat_k1=key_pressed) then
  begin
    keys.0:=1;
  end;
  if (repeat_k1=repeat_initial) then
  begin
    if (repeat_flag=1) then keys.0:=1;
  end;
end
else
begin
  repeat_k1:=0;
end;

if PORTB.5 = 0 then
begin
  //increase the "key pressed" counter (this is the "UP" or "+" key)
  if (repeat_k2<repeat_initial) then inc(repeat_k2);
  if (repeat_k2=key_pressed) then
  begin
    keys.1:=1;
  end;
  if (repeat_k2=repeat_initial) then
  begin
    if (repeat_flag=1) then keys.1:=1;
  end;
end
else
begin
  repeat_k2:=0;
end;

if PORTB.7 = 0 then
begin
  //increase the "key pressed" counter (this is the "OK" or "NEXT" key)
  if (repeat_k3<repeat_initial) then inc(repeat_k3);
  if (repeat_k3=key_pressed) then
  begin
    keys.2:=1;
  end;
  if (repeat_k3=repeat_initial) then
  begin
    if (repeat_flag=1) then keys.2:=1;
  end;
end
else
begin
  repeat_k3:=0;
end;

//=====
// sampling the cells voltage and current
//=====

tempw:=0;
ADCON0:=0x81;
delay_us(5);
for i:=0 to 63 do
begin
  ADCON0.2:=1;
  repeat until ADCON0.2=0;
  tempw:=tempw+ADRESL;
  tempw:=tempw+(ADRESH shl 8);
end;
fast_voltage:=tempw;
acc_voltage:=acc_voltage+tempw;
//voltage and current @16bits

tempw:=0;
ADCON0:=0x89;
delay_us(5);
for i:=0 to 63 do
```



# UNIVERSAL CHARGER

## Firmware source code

Revision 1.0

Page 5 of 15

```
begin
  ADCON0.2:=1;
  repeat until ADCON0.2=0;
  tempw:=tempw+ADRESL;
  tempw:=tempw+(ADRESH shl 8);
end;
fast_current:=tempw; //voltage and current @16bits
acc_current:=acc_current+tempw;

inc(count_slow);
if (count_slow=0) then
  begin
    slow_voltage:=acc_voltage shr 8; //voltage and current @16bits "high precision"
    slow_current:=acc_current shr 8; //voltage and current @16bits "high precision"
    acc_voltage:=0;
    acc_current:=0;
  end;

case (action) of //choice for different operating modes
//=====
// Discharge at constant current (all the batteries)
//=====
discharge_cc:begin //discharge control loop
  temp1:=fast_current-zero_current;
  if (temp1<0) then temp1:=0;
  mah:=mah+temp1;
  if (temp1>target_current) then //modulate the PWM following the discharge current changes
    begin
      dec(duty_pwm_discharge);
      if (duty_pwm_discharge<0) then duty_pwm_discharge:=0;
    end
  else
    begin
      inc(duty_pwm_discharge);
      if (duty_pwm_discharge>1023) then
        begin
          duty_pwm_discharge:=1023;
          pwm_err:=1; //if the PWM is at the maximum value, there is an error
        end;
    end;
  end;
end;

//=====
// Charge at constant current (NiMh and NiCd all the time, LiPo and SLA only the first period)
//=====
charge_cc:begin //charge cc control loop
  temp1:=zero_current-fast_current;
  if (temp1>0) then mah:=mah+temp1;
  if (temp1>target_current) then //modulate the PWM following the charge current changes
    begin
      dec(duty_pwm_charge);
      if (duty_pwm_charge<0) then duty_pwm_charge:=0;
    end
  else
    begin
      inc(duty_pwm_charge);
      if (duty_pwm_charge>1023) then
        begin
          duty_pwm_charge:=1023;
          pwm_err:=1; //if the PWM is at the maximum value, there is an error
        end;
    end;
  end;
end;

//=====
// Charge at constant voltage (LiPo and SLA second period)
//=====
charge_cv:begin //charge cv control loop
  temp1:=zero_current-fast_current;
  if (temp1>0) then mah:=mah+temp1;
  if (temp1>target_current) then
    begin
      dec(duty_pwm_charge); //In any case saturate at the maximum charge current
      if (duty_pwm_charge<0) then duty_pwm_charge:=0;
    end
  else
    begin
      if (ctrl_flag=1) then //decrease the bandwidth
        begin
          if (fast_voltage>target_voltage) then //modulate the PWM following the cells voltage changes
            begin
              dec(duty_pwm_charge);
              if (duty_pwm_charge<0) then duty_pwm_charge:=0;
            end
          else
            begin
              inc(duty_pwm_charge);
              if (duty_pwm_charge>1023) then
                begin
                  duty_pwm_charge:=1023;
                  pwm_err:=1; //if the PWM is at the maximum value, there is an error
                end;
            end;
          end;
        end;
    end;
  end;
end;

//=====
// Idle
//=====
else //otherwise reset all
  begin
    duty_pwm_discharge:=0;
    duty_pwm_charge:=0;
  end;
end;

setpwm; //update the PWM hardware with the most recent values
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 6 of 15

```
//=====
// during any charge or discharge TX via serial I/F the values
//=====

if (action<>0) then
begin
  i:=count_slow and 15;
  case 1 of
    0:TXREG:=0x55; //4 0x55 values for syncing with PC
    1:TXREG:=0x55;
    2:TXREG:=0x55;
    3:begin
      TXREG:=0x55; //latches the multibyte values
      if (action=1) then
      begin
        tempib:=slow_current-zero_current;
        pwmb:=duty_pwm_discharge;
      end
      else
      begin
        tempib:=zero_current-slow_current;
        pwmb:=duty_pwm_charge;
      end;
      slow_voltageb:=slow_voltage;
      mahb:=mah;
    end;
    4:TXREG:=hi(pwmb); //output of all the parameters, one for 5ms slot, total 16*5=80ms
    5:TXREG:=lo(pwmb);
    6:TXREG:=hi(tempib);
    7:TXREG:=lo(tempib);
    8:TXREG:=hi(slow_voltageb);
    9:TXREG:=lo(slow_voltageb);
    10:TXREG:=highest(mahb);
    11:TXREG:=higher(mahb);
    12:TXREG:=hi(mahb);
    13:TXREG:=lo(mahb);
    14:TXREG:=minu;
    15:TXREG:=sec;
  end;
end;

end;
end;

//=====
// Procedure delay in milliseconds
//=====

procedure delays(a:byte);
begin
  for bdelay:=1 to a do delay_ms(1);
end;

//=====
// procedure click for key feedback
//=====

procedure click;
begin
  for ebeep:=1 to 5 do
  begin
    begin
      PORTA.4:=ebeep.0;
      delays(1);
    end;
  end;
end;

//=====
// procedure single beep at user defined frequency
//=====

procedure beep;
begin
  INTCON.GIE:=0;
  cbeep:=eeprom_read(uni_beep);
  for dbeep:=1 to 201 do
  begin
    PORTA.4:=dbeep.0;
    for ii:=1 to cbeep do delay_us(10);
  end;
  INTCON.GIE:=1;
end;

//=====
// Three beep for end of charge and discharge
//=====

procedure tribeep;
begin
  for kk:=1 to 3 do
  begin
    begin
      beep;
      delays(200);
    end;
  end;
end;

//=====
// Display management routines
//=====

procedure lcd4(a,b:byte); //a command code, b=0 command, b=1 data;
begin
  PORTC.3:=b.0;
  PORTC.4:=0;
  PORTB.1:=a.4;
  PORTB.2:=a.5;
  PORTB.3:=a.6;
  PORTB.4:=a.7;
  PORTC.5:=1; //generic 4 bit command or data write
  PORTC.5:=0;
  PORTB.1:=a.0;
```



# UNIVERSAL CHARGER

## Firmware source code

Revision 1.0

Page 7 of 15

```
PORTB.2:=a.1;
PORTB.3:=a.2;
PORTB.4:=a.3;
PORTC.5:=1;
PORTC.5:=0;
delay_us(1000);
end;

procedure lcd8(a:byte);
begin
PORTC.3:=0;
PORTC.4:=0;
PORTB.1:=a.4;
PORTB.2:=a.5;
PORTB.3:=a.6;
PORTB.4:=a.7;
PORTC.5:=1;
PORTC.5:=0;
end;
//generic 8 bit command write

procedure cls;
begin
lcd4(1,0);
delays(5);
end;

procedure lcd_df_config;
begin
INTCON.GIE:=0;
delays(1);
lcd8(0x30);
delays(10);
lcd8(0x30);
delays(10);
lcd8(0x30);
delays(10);
lcd8(0x20);
delays(10);
lcd4(0x28,0);
lcd4(0x0C,0);
lcd4(6,0);
cls;
delays(250);
lcd4(0x80,0);
lcd4(32,1);
cls;
INTCON.GIE:=1;
end;
//start with 8 bit interface
//now we are on 4 bit interface
//write a space to test
//clear all

procedure lcd_df_out(a:byte;p:word);
begin
pt:=p;
lcd4(a,0);
repeat
pw:=Flash_read(pt);
q:=lo(pw);
if (q<>0) then lcd4(q,1);
inc(pt);
until (q=0);
end;

procedure lcd_df_out_eeprom(a:byte;p:byte);
begin
ps:=p;
lcd4(a,0);
for j:=0 to 15 do
begin
q:=eeprom_read(ps+j);
lcd4(q,1);
end;
end;

procedure lcd_df_char(a,c:byte); //a:command code, b:character
begin
lcd4(a,0);
lcd4(c,1);
end;

//=====
// Main menu for functions activation
//=====

procedure main_menu;
begin
repeat
lcd_df_out(128,@msg_main1);
case menupos of
0:lcd_df_out(192,@msg_main2);
1:lcd_df_out(192,@msg_main3);
2:lcd_df_out(192,@msg_main4);
3:lcd_df_out(192,@msg_main7);
4:lcd_df_out(192,@msg_main8);
5:lcd_df_out(192,@msg_main9);
6:lcd_df_out(192,@msg_main10);
end;
//clear display and write msg_main1

keys:=0;
repeat until (keys<>0);
click;
if (keys.0=1) then
begin
dec(menupos);
if (menupos=255) then menupos:=0;
end;
if (keys.1=1) then
begin
inc(menupos);
if (menupos=7) then menupos:=6;
end;
end;
end;
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 8 of 15

```
        end;
        until (keys.2=1);
end;

//=====
// Procedure display profile
//=====

procedure lcd_profile;
begin
    lcd_df_out(128,@msg_display);
    l:=selprof;
    inc(l);
    bytetostr(l,tmpbyte);
    lcd_df_char(133,tmpbyte[1]);
    lcd_df_char(134,tmpbyte[2]);
    lcd_df_char(135,':');
    l:=eeprom_read(profile_point+pchemistry);
    case l of
        nicd:lcd_df_out(136,@msg_change3);
        nimh:lcd_df_out(136,@msg_change4);
        lipo:lcd_df_out(136,@msg_change5);
        sla:lcd_df_out(136,@msg_change6);
    end;
    lcd_df_char(141,'x');

    l:=eeprom_read(profile_point+pncells);
    bytetostr(l,tmpbyte);
    lcd_df_char(142,tmpbyte[1]);
    lcd_df_char(143,tmpbyte[2]);

    l:=eeprom_read(profile_point+pcapacity);
    tw:=l*100;
    wordtostr(tw,tmpword);
    lcd_df_char(192,'K');
    lcd_df_char(193,tmpword[0]);
    lcd_df_char(194,tmpword[1]);
    lcd_df_char(195,tmpword[2]);
    lcd_df_char(196,tmpword[3]);
    lcd_df_char(197,tmpword[3]);

    l:=eeprom_read(profile_point+pcharge);
    bytetostr(l,tmpbyte);
    lcd_df_char(198,'C');
    lcd_df_char(199,tmpbyte[0]);
    lcd_df_char(200,tmpbyte[1]);
    lcd_df_char(201,'.');
    lcd_df_char(202,tmpbyte[2]);

    l:=eeprom_read(profile_point+pdischarge);
    bytetostr(l,tmpbyte);
    lcd_df_char(203,'D');
    lcd_df_char(204,tmpbyte[0]);
    lcd_df_char(205,tmpbyte[1]);
    lcd_df_char(206,'.');
    lcd_df_char(207,tmpbyte[2]);
end;

//=====
// Battery profile selection
//=====

procedure selprofile;
begin
    selprof:=eeprom_read(uni_profile);

    repeat
        lcd_profile;
        keys:=0;
        repeat until (keys<>0);
        click;
        if (keys.0=1) then
            begin
                dec(selprof);
                if (selprof=255) then selprof:=0;
                profile_point:=selprof*profilelen;
            end;
        if (keys.1=1) then
            begin
                inc(selprof);
                if (selprof=12) then selprof:=11;
                profile_point:=selprof*profilelen;
            end;
        until (keys.2=1);

        eeprom_write(uni_profile,selprof);
    end;

//=====
// procedure A to ADU and viceversa
//=====

function atoadu:word;
begin
    tw:=eeprom_read(uni_currh)*256+eeprom_read(uni_curr1);
    tlw:=tw*7629;
    result:=tlw/7629;
end;

function adutoa:word;
begin
    tw:=eeprom_read(uni_currh)*256+eeprom_read(uni_curr1);
    tlw:=tw*7629;
    result:=tlw/tw;
end;

//=====
// procedure mV to ADU and viceversa
//=====
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 9 of 15

```
function mvtoadu:word; //twl argument in mV, result in ADU
begin
  vrat:=(eeprom_read(uni_r5h) shl 8)+eeprom_read(uni_r5l); //R5
  tw:=(eeprom_read(uni_r6h) shl 8)+eeprom_read(uni_r6l); //R6
  tlw:=tw shl 12; //R6*4096
  tlw1:=tw+vrat; //R5+R6
  tlw:=tlw/tlw1; //R5*4096/(R5+R6)
  tlw:=(tw*tlw) shl 1; //mV* (R6/ (R5+R6)) *65536
  result:=tlw/625; //mV* (R6/ (R5+R6)) * (65536/5000)
end;

function adutomv:word; //twl argument in ADU,result in mV
begin
  vrat:=(eeprom_read(uni_r5h) shl 8)+eeprom_read(uni_r5l); //R5
  tw:=(eeprom_read(uni_r6h) shl 8)+eeprom_read(uni_r6l); //R6
  tlw:=vrat shl 12; //R5*4096
  tlw:=4096+(tlw/tw); //K=(1+R5/R6)*4096
  tlw:=tlw*tlw; //ADU*4096*(1+R5/R6)/4096
  result:=tlw/53687; //ADU*(1+R5/R6)*(5000/65536)
end;

//=====
// procedure mah ADU to display mah
//=====

function recallmah:word; //result in mah
begin
  tw:=highest(mah) shl 8;
  tw:=tw+higher(mah);
  tlw:=6944*tw;
  tw:=eeprom_read(uni_currh)*256+eeprom_read(uni_curr1);
  tlw:=tlw/tw;
  result:=tlw;
end;

//=====
// procedure prepare display for charge and discharge (& other)
//=====

procedure prepdis(a:byte);
begin
  lcd_df_out(128,@msg_init);
  for ii:=1 to 30 do Delaysms(100);
  zero_current:=slow_current; //sample the "no current" level

  cls;
  l:=eeprom_read(profile_point+pcapacity); //l holds the capacity divided by 100 -> 40 = 4000mAh
  tw:=l;

  if (a=0) then l:=eeprom_read(profile_point+pdischarge) //l holds the user rate multiplied by 10 -> 20 = 2.0C
  else l:=eeprom_read(profile_point+pcharge);

  tw1:=l*tw; //target current in A*100 -> 40*20=800 discharge current (8.00A)
  if (a=0) then l:=eeprom_read(uni_maxdischarge)
  else l:=eeprom_read(uni_maxcharge);
  tw2:=l*100;
  if tw1>tw2 then tw1:=tw2; //saturate if the current exceed the charger capabilities
  target_current:=atoadu; //convert the value in ADU

  lcd_df_char(143,'A');
  lcd_df_char(198,'V');

  if calibra=0 then
  begin
    if (a=0) then lcd_df_out(128,@msg_discharge)
    else lcd_df_out(128,@msg_charge);
    lcd_df_char(205,'m');
    lcd_df_char(206,'A');
    lcd_df_char(207,'h');
  end;

  act:=eeprom_read(profile_point+pchemistry); //read the chemistry
end;

//=====
// display management during charge and discharge
//=====

procedure displ;
begin
  tw:=adutoa; //current display in format xx.xx
  wordtostr(tw,tmpword);
  lcd_df_char(138,tmpword[1]);
  lcd_df_char(139,tmpword[2]);
  lcd_df_char(140,'. ');
  lcd_df_char(141,tmpword[3]);
  lcd_df_char(142,tmpword[4]);

  tw1:=slow_voltage; //voltage display in format xx.xxx
  tw:=adutomv;
  wordtostr(tw,tmpword);
  lcd_df_char(192,tmpword[0]);
  lcd_df_char(193,tmpword[1]);
  lcd_df_char(194,'. ');
  lcd_df_char(195,tmpword[2]);
  lcd_df_char(196,tmpword[3]);
  lcd_df_char(197,tmpword[4]);

  if (calibra=0) then //if not in calibration,
  begin //capacity display in format xxxxx
    tw:=recallmah;
    wordtostr(tw,tmpword);
    lcd_df_char(200,tmpword[0]);
    lcd_df_char(201,tmpword[1]);
    lcd_df_char(202,tmpword[2]);
    lcd_df_char(203,tmpword[3]);
    lcd_df_char(204,tmpword[4]);
  end;
end;
```



# UNIVERSAL CHARGER

## Firmware source code

Revision 1.0

Page 10 of 15

```
//=====
// Charge routine
//=====

procedure charge;
begin

  cls; //clear display
  eeprom_write(uni_mode,mode_charge); //set the charge flag in case the action will be interrupted
  PORTB.0:=1; //fan ON

  prepdis(1); //prepare the display for charge

  ii:=eeprom_read(profile_point+pncells);
  m:=eeprom_read(profile_point+ptimeout);

  case act of
    0:begin //read values for NiCd
      l:=eeprom_read(profile_point+pdeltav_nicd);
      n:=eeprom_read(profile_point+pinhibit);
      lcd_df_out(133,@msg_change3);
      twl:=ii*1;
      end;
    1:begin //read values for NiMh
      l:=eeprom_read(profile_point+pdeltav_nimh);
      n:=eeprom_read(profile_point+pinhibit);
      lcd_df_out(133,@msg_change4);
      twl:=ii*1;
      end;
    2:begin //read values for LiPo
      l:=eeprom_read(profile_point+pmaxv_lipo);
      n:=eeprom_read(profile_point+pfinalcurr_lipo);
      lcd_df_out(133,@msg_change5);
      tw:=3500+(l shl 2);
      twl:=ii*tw;
      end;
    3:begin //read values for SLA
      l:=eeprom_read(profile_point+pmaxv_sla);
      n:=eeprom_read(profile_point+pfinalcurr_sla);
      lcd_df_out(133,@msg_change6);
      tw:=2000+(l shl 2);
      twl:=ii*tw;
      end;
  end;

  target_voltage:=mvtoadu; //target voltage: deltaV for NiCd e NiMh, MaxV for LiPo e SLA

  j:=eeprom_read(profile_point+pcapacity); //j holds the capacity divided by 100 -> 40 = 4000mAh
  maxcap:=j*m; //maxcap holds the timeout-> cap*timeout=capacity for timeout escape

  tlw:=target_current*n;
  ilim:=tlw/100; //ilim holds the value of current for end of charge in the CV phase

  action:=charge_cc; //starts the control
  mah:=0; //reset the variables
  sec:=0;
  minu:=0;
  pk:=0;
  keys:=0;
  pwm_err:=0;
  repeat

    twi:=zero_current-slow_current; //calculate the effective current
    if twi<0 then twi:=0;
    twl:=twi;
    displ; //display the values

    case act of
      0,1:begin //if NiXX charge, wait for end of inhibition
        if minu>n then
          begin
            INTCON.GIE:=0;
            if slow_voltage>pk then pk:=slow_voltage; //holds the peak
            twi:=pk-slow_voltage;
            INTCON.GIE:=1;
            if twi<0 then twi:=0;
            if twi>target_voltage then action:=idle; //if under the delta peak, exit
          end;
        end;
      2,3:begin //if LiXX charge, wait for end of inhibition
            INTCON.GIE:=0;
            if (action=charge_cc) then if slow_voltage>target_voltage then action:=charge_cv; //if LiXX charge stops the CC charge at max voltage
            if (action=charge_cv) then if twi<ilim then action:=idle; //stops the charge if under final current
            INTCON.GIE:=1;
          end;
        end;
    end;

    tw:=recallmah;
    if (tw>maxcap) then action:=idle;
    if pwm_err=1 then action:=idle;

  until (keys.2=1) or (action=idle);

  action:=idle;
  eeprom_write(uni_mode,mode_idle);
  lcd_df_out(133,@msg_end);
  sec:=0;
  tribeep;
  keys:=0;
  repeat //loop for heatsink cooling, display the cell voltage
    INTCON.GIE:=0;
    twi:=zero_current-slow_current;
    INTCON.GIE:=1;
    if twi<0 then twi:=0;
    twl:=twi;
    displ;
  until (keys.2=1);
  click;
  PORTB.0:=0; //fan OFF

end;

//=====
// Discharge routine
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 11 of 15

```
//=====
procedure discharge;
begin
  cls;
  eeprom_write(uni_mode,mode_discharge);
  PORTB.0:=1;
  prepdis(0);
  case act of
    0:begin
      l:=eeprom_read(profile_point+pcutoff_nicd);
      tw:=1*10;
      lcd_df_out(133,@msg_change3);
      end;
    1:begin
      l:=eeprom_read(profile_point+pcutoff_nimh);
      tw:=1*10;
      lcd_df_out(133,@msg_change4);
      end;
    2:begin
      l:=eeprom_read(profile_point+pcutoff_lipo);
      tw:=2500+(l shl 2);
      lcd_df_out(133,@msg_change5);
      end;
    3:begin
      l:=eeprom_read(profile_point+pcutoff_sla);
      tw:=1500*(l shl 2);
      lcd_df_out(133,@msg_change6);
      end;
  end;

  l:=eeprom_read(profile_point+pcells);
  twl:=1*tw;
  target_voltage:=mvtoadu; //ideal total cutoff voltage in mV

  action:=discharge_cc; //start the control
  sec:=0;
  minu:=0;
  mah:=0;
  keys:=0;
  pwm_err:=0;

  repeat
    INTCON.GIE:=0;
    twi:=slow_current-zero_current;
    INTCON.GIE:=1;
    if twi<0 then twi:=0;
    twl:=twi;
    displ;
    INTCON.GIE:=0;
    if pwm_err=1 then action:=idle;
    if (fast_voltage<target_voltage) then action:=idle; //exit when under cutoff voltage...
    INTCON.GIE:=1;
  until (action=idle) or (keys.2=1);

  action:=idle;
  eeprom_write(uni_mode,mode_idle);
  lcd_df_out(133,@msg_end);
  sec:=0;
  tribeep;
  keys:=0;
  repeat
    INTCON.GIE:=0; //loop for heatsink cooling, display the cell voltage
    twi:=slow_current-zero_current;
    INTCON.GIE:=1;
    if twi<0 then twi:=0;
    twl:=twi;
    displ;
  until (keys.2=1);
  click;
  PORTB.0:=0; //fan OFF
end;

//=====
// PC management for internal parameters -> xx00.AAAA,xx01.BBBB,xx10.CCCC,1x11.DDDD -> write CCCC.DDDD at address AAAA.BBBB
// PC management for internal parameters -> xx00.AAAA,xx01.BBBB,xx10.xxxx,0011.xxxx -> read from address AAAA.BBBB
//=====

procedure pmanage;
begin
  k:=0;
  l:=0;
  m:=0;
  cls;
  lcd_df_out(128,@msg_pcman1); //pc management message
  // lcd_df_out(192,@msg_pcman2);
  keys:=0;
  repeat
    if PIR1.RCIF=1 then //if an UART character has been received
      begin
        if (RCSTA and 6)>0 then //if there is an error
          begin
            j:=RCREG;
            j:=RCREG;
            RCSTA.CREN:=0;
            RCSTA.CREN:=1; //clear the error and flush the buffer
          end
        else
          begin
            eevar:=RCREG; //if no error read the value
            j:=(eevar and 0x30) shr 4; //select the two bit xxAA.DDDD
            case j of
              0:begin
                  if (m=0) then m:=1 else m:=0; // if AA=00 we have an address, first 4 MSB...
                  if (m=1) then
                    begin
                      k:=0;
                      ki:=(eevar and 0x0F) shl 4;
                    end;
                end;
            1:begin

```



# UNIVERSAL CHARGER

## Firmware source code

Revision 1.0

Page 12 of 15

```
        if (m=1) then m:=2 else m:=0;
        if (m=2) then
            begin
                k:=k or (eevar and 0x0F);
            end;
        end;
    2:begin
        if (m=2) then m:=3 else m:=0;
        if (m=3) then
            begin
                l:=0;
                l:=(eevar and 0x0F) shl 4;
            end;
        end;
    3:begin
        if (m=3) then m:=4 else m:=0;
        if (m=4) then
            begin
                l:=l or (eevar and 0x0F);
                if eevar.7=0 then
                    begin
                        l:=eeprom_read(k);
                        TXREG:=l;
                    end
                    //read from EEPROM and send to UART
                else
                    begin
                        eeprom_write(k,l);
                    end
                    //write to EEPROM
                end;
                m:=0;
            end;
        end;
    end;
end;
until (keys<>0);
click;
end;
//repeat until keypressed
```

```
//=====
// Profile display (display of chemistry, number of cells, capacity ...)
//=====
```

```
procedure display_profile;
begin
    case selitem of
    0:begin
        lcd_df_out(128,@msg_change1);
        lcd_df_out(192,@msg_change2);
        l:=eeprom_read(profile_point+pchemistry);
        case l of
        nicd:lcd_df_out(139,@msg_change3);
        nimh:lcd_df_out(139,@msg_change4);
        lipo:lcd_df_out(139,@msg_change5);
        sla:lcd_df_out(139,@msg_change6);
        end;
    end;
    1:begin
        lcd_df_out(128,@msg_change7);
        lcd_df_out(192,@msg_change8);
        l:=eeprom_read(profile_point+pcapacity);
        tw:=l*100;
        wordtostr(tw,tmpword);
        lcd_df_char(138,tmpword[0]);
        lcd_df_char(139,tmpword[1]);
        lcd_df_char(140,tmpword[2]);
        lcd_df_char(141,tmpword[3]);
        lcd_df_char(142,tmpword[4]);
    end;
    2:begin
        lcd_df_out(128,@msg_change9);
        lcd_df_out(192,@msg_change10);
        l:=eeprom_read(profile_point+pnccells);
        bytetostr(l,tmpbyte);
        lcd_df_char(135,tmpbyte[1]);
        lcd_df_char(136,tmpbyte[2]);
    end;
    3:begin
        lcd_df_out(128,@msg_change11);
        lcd_df_out(192,@msg_change12);
        l:=eeprom_read(profile_point+pcharge);
        bytetostr(l,tmpbyte);
        lcd_df_char(136,tmpbyte[0]);
        lcd_df_char(137,tmpbyte[1]);
        lcd_df_char(138,'. ');
        lcd_df_char(139,tmpbyte[2]);
    end;
    4:begin
        lcd_df_out(128,@msg_change13);
        lcd_df_out(192,@msg_change12);
        l:=eeprom_read(profile_point+pdischarge);
        bytetostr(l,tmpbyte);
        lcd_df_char(139,tmpbyte[0]);
        lcd_df_char(140,tmpbyte[1]);
        lcd_df_char(141,'. ');
        lcd_df_char(142,tmpbyte[2]);
    end;
    end;
end;
```

```
//=====
// 8 bit Variable modify
//=====
```

```
procedure modvar(a:byte);
begin
    l:=eeprom_read(eevar);
    if (a=0) then dec(l) else inc(l);
    if l<eelim then l:=eeres;
    eeprom_write(eevar,l);
end;
//read the variable
//modify
//if below EELIM, saturate to EERES
//write back the value
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 13 of 15

```
//=====
// Key managemement during profile selection
//=====

procedure key_profile;
begin
  keys:=0;
  repeat until (keys<>0);
  click;
  if (keys.0=1) then
  begin
    case selitem of
      0:begin
        eevar:=profile_point+pchemistry;
        eelim:=255;
        eeres:=0;
        modvar(0);
      end;
      1:begin
        eevar:=profile_point+pcapacity;
        eelim:=0;
        eeres:=1;
        modvar(0);
      end;
      2:begin
        eevar:=profile_point+pncells;
        eelim:=0;
        eeres:=1;
        modvar(0);
      end;
      3:begin
        eevar:=profile_point+pcharge;
        eelim:=0;
        eeres:=1;
        modvar(0);
      end;
      4:begin
        eevar:=profile_point+pdischarge;
        eelim:=0;
        eeres:=1;
        modvar(0);
      end;
    end;
  end;
  if (keys.1=1) then
  begin
    case selitem of
      0:begin
        eevar:=profile_point+pchemistry;
        eelim:=4;
        eeres:=3;
        modvar(1);
      end;
      1:begin
        eevar:=profile_point+pcapacity;
        eelim:=0;
        eeres:=255;
        modvar(1);
      end;
      2:begin
        eevar:=profile_point+pncells;
        eelim:=20;
        eeres:=19;
        modvar(1);
      end;
      3:begin
        eevar:=profile_point+pcharge;
        eelim:=0;
        eeres:=255;
        modvar(1);
      end;
      4:begin
        eevar:=profile_point+pdischarge;
        eelim:=0;
        eeres:=255;
        modvar(1);
      end;
    end;
  end;
  if (keys.2=1) then
  begin
    inc(selitem);
  end;
end;

//=====
// Battery profile change
//=====

procedure changeprofile;
begin
  selitem:=0;
  repeat
    display_profile;
    key_profile;
  until (keys.2=1) and (selitem=5);
end;

//=====
// Procedure default values
//=====

procedure default_values_pack;
begin
  j:=k*profilelen;
  eeprom_write(j+pchemistry,0);
  eeprom_write(j+pcapacity,30);
  eeprom_write(j+pncells,6);
  eeprom_write(j+pcharge,10);
  eeprom_write(j+pdischarge,40);
  eeprom_write(j+pinhibit,5);

  //Default NiCd (0) -> 0:NiCd, 1:NiMh, 2:LiPo, 3:SLA
  //Default cells capacity (30) -> 30*100=3000mAh
  //Default number of cells (6) -> 6
  //default charge (10) -> 3000*1.0=3A
  //default discharge (40) -> 3000*4.0=12A
  //default deltapeak check inhibition (5) -> 5 minutes
end;
```



# UNIVERSAL CHARGER

## Firmware source code

Revision 1.0

Page 14 of 15

```

eeprom_write(j+pcutoff_nicd,80);
eeprom_write(j+pcutoff_nimh,100);
eeprom_write(j+pcutoff_lipo,125);
eeprom_write(j+pcutoff_sla,125);
eeprom_write(j+pdeltav_nicd,10);
eeprom_write(j+pdeltav_nimh,5);
eeprom_write(j+pmavx_lipo,175);
eeprom_write(j+pmavx_sla,125);
eeprom_write(j+pfinalcurr_lipo,5);
eeprom_write(j+pfinalcurr_sla,5);
eeprom_write(j+ptimeout,120);
end;

//default NiCd cutoff (80) -> 80*10=800mV
//default NiMh cutoff (100) -> 100*10=1000mV
//default LiPo cutoff (125) -> 2500+125*4=3000mV
//default SLA cutoff (125) -> 1500+125*4=2000mV
//default NiCd deltapeak (10) -> 10mV
//default NiMh deltapeak (5) -> 5mV
//default LiPo max voltage (175) -> 3500+175*4=4200mV
//default SLA max voltage (125) -> 2000+125*4=2500mV
//default LiPo final current (5) -> 3000*5/100=150mA
//default SLA final current (5) -> 3000*5/100=150mA
//default cell max charge (120) -> 3000*120/100=3600mAh

procedure default_values_uni;
begin
eeprom_write(uni_profile,0);
eeprom_write(uni_maxcharge,5);
eeprom_write(uni_maxdischarge,20);
eeprom_write(uni_beep,25);
eeprom_write(uni_r5h,183);
eeprom_write(uni_r5l,152);
eeprom_write(uni_r6h,46);
eeprom_write(uni_r6l,224);
eeprom_write(uni_currh,97);
eeprom_write(uni_curr,168);
eeprom_write(uni_mode,mode_idle);
for k:=0 to 15 do eeprom_write(uni_firstline+k,msg_hello1[k]);
for k:=0 to 15 do eeprom_write(uni_secondline+k,msg_hello2[k]);
end;

//Default profile selected (0) -> 1st profile
//Default max charge current (5) -> 5A
//Default max discharge current (20) -> 20A
//default beep tone semiperiod (25) -> 25*10=250us (2000Hz)
//default (183)
//default (152) R5=R5h*256+R5l = 47000 Ohm
//default (46)
//default (224) R6=R6h*256+R6l = 12000 Ohm
//default (97)
//default (168) Curr=Currh*256+Currl= 25000 -> 25000uv/A
//default mode (0) -> idle
//default hello message 1st line -> ".seven-segments."
//default hello message 2nd line -> "www. .com"

//=====
// 16 bit Variable modify
//=====

procedure modvar16(a:byte);
begin
tw:=eeprom_read(eevar);
inc(eevar);
tw:=tw+(eeprom_read(eevar) shl 8);
if (a=0) then tw:=tw+10 else tw:=tw-10;
eeprom_write(eevar,hi(tw));
dec(eevar);
eeprom_write(eevar,lo(tw));
end;

//=====
// Calibration procedure (Volt and Ampere)
//=====

procedure calibration(a:byte);
begin
cls;
calibra:=1;
PORTB.0:=1;
prepdis(2);

tw:=200;
target_current:=atoadu;
if (a=0) then
begin
action:=idle;
eevar:=uni_r6l;
end
else
begin
action:=charge_cc;
eevar:=uni_curr;
end;

keys:=0;
repeat
INTCON.GIE:=0;
tw:=zero_current-slow_current;
INTCON.GIE:=1;
if tw<0 then tw:=0;
tw:=tw;
displ;
if (keys.0=1) then
begin
modvar16(0);
keys:=0;
click;
end;
if (keys.1=1) then
begin
modvar16(1);
keys:=0;
click;
end;
until (keys.2=1);

action:=idle;
click;
calibra:=0;
PORTB.0:=0;
end;

//=====
// Main program, neverending loop
//=====

begin
ADCON0:=0x81;
INTCON:=0x20;
RCSTA:=0x90;
CCPR1L:=0x0;
CCPR2L:=0x0;
T2CON:=0x4;
//PIC hardware related variables init
```



# UNIVERSAL CHARGER

Firmware source code

Revision 1.0

Page 15 of 15

```
CCP1CON:=0xC;
CCP2CON:=0xC;
OPTION_REG:=0x6;
SPBRG:=0x81;
TXSTA:=0xA4;
ADCON1:=0x84;
TRISA:=0x0B;
TRISB:=0xE0;
TRISC:=0xE0;
FR2:=0xFF;
PORTA:=0xFF;
PORTB:=0x0;
PORTC:=0x0;

repeat_k1:=0;
repeat_k2:=0;
repeat_k3:=0;
action:=0;
duty_pwm_charge:=0;
duty_pwm_discharge:=0;
profile_point:=0;
calibra:=0;

//9600 baud @20MHz

//all off
//no calibration at startup

lcd_df_config;
j:=eeprom_read(uni_profile);
if j=255 then
begin
  lcd_df_out(128,@msg_init);
  default_values uni;
  for k:=0 to 11 do default_values_pack;
end
else profile_point:=j*profilelen;
//if the EEPROM is unprogrammed, program for defaults
//writing 255 to the profile selection address the default is reloaded
//else restore the current profile

lcd_df_out_eeprom(128,uni_firstline);
lcd_df_out_eeprom(192,uni_secondline);

beep;

k:=0;
keys:=0;
repeat
  inc(k);
  delays(10);
  until (keys<>0) or (k=255);
  if keys<>0 then click;
  keys:=0;
  //wait for user action or timeout for interrupted action restart

if (k=255) then
begin
  //if no key pressed resume the last operation (if any)
  phase:=eeprom_read(uni_mode);
  //read if there is a suspended action
  if (phase=mode_charge) then
  begin
    //if a precedented action was suspended by a power loss,
    //complete it
    charge;
    //finish the discharge
  end;
  if (phase=mode_discharge) then
  begin
    //finish the discharge
    discharge;
  end;
end;

eeprom_write(uni_mode,mode_idle);
//abort the last action, if necessary

menupos:=0;
//start display from "profile selection"
repeat
  main_menu;
  //display the main menu and check the keys
  case menupos of
    0:selprofile; //battery pack profile selection routine
    1:charge; //charge the battery
    2:discharge; //discharge the battery
    3:changeprofile; //Actual battery pack profile change
    4:pcmanage; //PC control on parameters
    5:calibration(0); //Voltage calibration
    6:calibration(1); //Current calibration
  end;
  //neverending loop
until false;

end.
```