```
/*


CELL MODULE FOR ATTINY1624

(c)2019 to 2021 Stuart Pittaway

MODIFIED BY
2024 Nidula Gunawardana

COMPILE THIS CODE USING MICROCHIP STUDIO

LICENSE
Attribution-NonCommercial-ShareAlike 2.0 UK: England & Wales (CC BY-NC-SA 2.0
UK)
https://creativecommons.org/licenses/by-nc-sa/2.0/uk/

* Non-Commercial — You may not use the material for commercial purposes.
* Attribution — You must give appropriate credit, provide a link to the license,
  and indicate if changes were made.
  You may do so in any reasonable manner, but not in any way that suggests the
    licensor endorses you or your use.
* ShareAlike — If you remix, transform, or build upon the material, you must
  distribute your
  contributions under the same license as the original.
* No additional restrictions — You may not apply legal terms or technological
  measures
  that legally restrict others from doing anything the license permits.
*/
/*

*/

#define F_CPU 2000000UL // 2 MHz clock speed

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <avr/eeprom.h>
#include <diybms_tinyAVR2.h>
#include <FastPID.h>
#include <SerialEncoder.h>
#include <packet_processor.h>

#define RX_BUFFER_SIZE 64

uint8_t SerialPacketReceiveBuffer[8 + sizeof(PacketStruct)];

SerialEncoder PacketSerial;

CellModuleConfig Config;

PacketProcessor Processor(&Config);

volatile bool WatchdogTriggered = false;
```

```
volatile uint8_t InterruptCounter = 0;
volatile uint16_t PulsePeriod = 0;
volatile uint16_t OnPulseCount = 0;

void DefaultConfig()
{
  Config.Calibration = 1.0;
  Config.BypassTemperatureSetPoint = 65;
  Config.BypassThresholdmV = 4100;
}

void watchdog()
{
  WatchdogTriggered = true;
  Processor.IncrementWatchdogCounter();
}

void onPacketReceived()
{
  diyBMSHAL::EnableSerial0TX();

  if (Processor.onPacketReceived((PacketStruct *)SerialPacketReceiveBuffer))
  {
    diyBMSHAL::NotificationLedOn();
  }

  PacketSerial.sendBuffer(SerialPacketReceiveBuffer);

  diyBMSHAL::FlushSerial0();
  diyBMSHAL::NotificationLedOff();
}

FastPID PID(5.0, 1.0, 0.1, 3, 8, false);

void ValidateConfiguration()
{
  if (Config.Calibration < 0.8 || Config.Calibration > 10.0)
  {
    Config.Calibration = 1.0;
  }

  if (Config.BypassTemperatureSetPoint > DIYBMS_MODULE_SafetyTemperatureCutoff)
  {
    Config.BypassTemperatureSetPoint = DIYBMS_MODULE_SafetyTemperatureCutoff - 10;
  }
}

void StopBalance()
{
  Processor.WeAreInBypass = false;
  Processor.bypassCountDown = 0;
  Processor.bypassHasJustFinished = 0;
  Processor.PWMSetPoint = 0;
  Processor.SettingsHaveChanged = false;
```

```
  OnPulseCount = 0;
  PulsePeriod = 0;

  diyBMSHAL::StopTimer1();
  diyBMSHAL::DumpLoadOff();
}

void setup()
{
  wdt_disable();
  wdt_reset();

  bool JustPoweredUp = true;

  if ((GPIOR0 & 0x08) == 0x08)
  {
    watchdog();
    JustPoweredUp = false;
  }

  diyBMSHAL::SetPrescaler();
  diyBMSHAL::SetWatchdog8sec();
  diyBMSHAL::ConfigurePorts();

  if (JustPoweredUp)
  {
    diyBMSHAL::PowerOn_Notification_led();
  }
  if (!Settings::ReadConfigFromEEPROM((uint8_t *)&Config, sizeof(Config),
    EEPROM_CONFIG_ADDRESS))
  {
    DefaultConfig();
  }

  ValidateConfiguration();

  PID.setOutputRange(0, 255);

  StopBalance();

  // Set up UART
  UBRR0 = F_CPU / (8 * DIYBMSBAUD) - 1;
  UCSR0A |= (1 << U2X0);
  UCSR0B |= (1 << RXEN0) | (1 << TXEN0);
  UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);

  PacketSerial.begin(&onPacketReceived, sizeof(PacketStruct),
    SerialPacketReceiveBuffer, sizeof(SerialPacketReceiveBuffer));
}

void BalanceTimer()
{
  InterruptCounter++;
  PulsePeriod++;
  if (InterruptCounter == 255)
  {
```

```
    InterruptCounter = 0;
  }
  if (InterruptCounter <= Processor.PWMSetPoint)
  {
    diyBMSHAL::DumpLoadOn();
    OnPulseCount++;
  }
  else
  {
    diyBMSHAL::DumpLoadOff();
  }

  if (PulsePeriod == 1000 && OnPulseCount != 0)
  {
    float CurrentmA = ((float)Processor.CellVoltage() / (float)LOAD_RESISTANCE);

    float milliAmpHours = (CurrentmA * ((float)OnPulseCount / (float)1000.0)) *
      (1.0 / 3600.0);

    Processor.MilliAmpHourBalanceCounter += milliAmpHours;

    OnPulseCount = 0;
    PulsePeriod = 0;
  }
}

ISR(TCA0_OVF_vect)
{
  BalanceTimer();
  TCA0.SINGLE.INTFLAGS = 0x01;
}

inline void identifyModule()
{
  if (Processor.identifyModule > 0)
  {
    diyBMSHAL::NotificationLedOn();
    Processor.identifyModule--;

    if (Processor.identifyModule == 0)
    {
      diyBMSHAL::NotificationLedOff();
    }
  }
}

void loop()
{
  wdt_reset();
  identifyModule();

  if (Processor.SettingsHaveChanged)
  {
    StopBalance();
  }
```

```
  if (WatchdogTriggered)
  {
    diyBMSHAL::double_tap_Notification_led();
    StopBalance();
  }

  diyBMSHAL::TemperatureVoltageOn();

  Processor.TakeAnAnalogueReading(ADC_INTERNAL_TEMP);

  if (Processor.bypassCountDown == 0)
  {
    Processor.TakeAnAnalogueReading(ADC_EXTERNAL_TEMP);

    diyBMSHAL::ReferenceVoltageOn();

#if (SAMPLEAVERAGING == 1)
    Processor.TakeAnAnalogueReading(ADC_CELL_VOLTAGE);
#else
    for (size_t i = 0; i < 5; i++)
    {
      Processor.TakeAnAnalogueReading(ADC_CELL_VOLTAGE);
    }
#endif
  }

  diyBMSHAL::ReferenceVoltageOff();
  diyBMSHAL::TemperatureVoltageOff();

  if (WatchdogTriggered == true && !(UCSR0A & (1 << RXC0)))
  {
  }
  else
  {
    for (size_t i = 0; i < 200; i++)
    {
      PacketSerial.checkInputStream();
      _delay_ms(1);
    }
  }

  int16_t InternalTemp = Processor.InternalTemperature();

  if (InternalTemp > DIYBMS_MODULE_SafetyTemperatureCutoff || InternalTemp >
    (Config.BypassTemperatureSetPoint + 10))
  {
    PID.clear();
    StopBalance();
  }

  if (Processor.BypassCheck() && InternalTemp <
    DIYBMS_MODULE_SafetyTemperatureCutoff)
  {
    if (!Processor.WeAreInBypass)
    {
      Processor.WeAreInBypass = true;
```

```
      Processor.bypassCountDown = 50;
      Processor.bypassHasJustFinished = 0;

      diyBMSHAL::StartTimer1();
      Processor.PWMSetPoint = 0;
    }
  }

  if (Processor.bypassCountDown > 0)
  {
    if (InternalTemp < (Config.BypassTemperatureSetPoint - 6))
    {
      Processor.PWMSetPoint = 0xFF;
    }
    else
    {
      Processor.PWMSetPoint = PID.step(Config.BypassTemperatureSetPoint,
        InternalTemp);

      if (PID.err())
      {
        PID.clear();
        StopBalance();
      }
    }

    Processor.bypassCountDown--;

    if (Processor.bypassCountDown == 0)
    {
      StopBalance();
      Processor.bypassHasJustFinished = 150;
    }
  }

  if (Processor.bypassHasJustFinished > 0)
  {
    Processor.bypassHasJustFinished--;
  }

  WatchdogTriggered = false;

  if (!Processor.WeAreInBypass && Processor.bypassHasJustFinished == 0 && !(UCSR0A
    & (1 << RXC0)))
  {
    PID.clear();
    diyBMSHAL::Sleep();
  }
}

int main(void)
{
  setup();
  while (1)
  {
```

```
    loop();
  }
}
```