//////////////////////////////////////////////////////////////////////////////////////////////

// Groundwater level sensor //

// November/December 2015 Frans Mulder //

//////////////////////////////////////////////////////////////////////////////////////////////

#include <EEPROM.h>

#include "Wire.h"

#define RTC\_ADDRESS 0x68 // Refers to DS3233 Real Time Clock module, accessed through I2C protocol

//////////////////////////////////////////////////////////////////////////////////////////////

// Pin Connections //

//////////////////////////////////////////////////////////////////////////////////////////////

#define RedLedPin 6 // Red Led is connected to pin 6 (Port D, bit6)

#define GreenLedPin 7 // Green Led is connected to pin 7 (Port D, bit7)

#define SensorPin 5 // Timer 1 external clock is connected to pin 5 (Port D, bit5)

#define SwitchPin 8 // Push Button, Normally Low, High when pressed, Debounced T = 20ms

#define SensorEnablePin 4 // By making pin 4 high the sensor is activated

//////////////////////////////////////////////////////////////////////////////////////////////

// POWERDOWN MODES //

//////////////////////////////////////////////////////////////////////////////////////////////

#define PDTimerCounter0 0 // Power Down Timer Counter 0

#define PDTimerCounter1 1 // Power Down Timer Counter 1

#define PDTimerCounter2 2 // Power Down Timer Counter 2

#define PDADC 3 // Power Down AD Converter

#define PDUSART 4 // Power Down USART

#define PDSPI 5 // Power Down SPI

#define PDTWI 6 // Power Down TWI (Two Wire Interface)

//////////////////////////////////////////////////////////////////////////////////////////////

// Sensor States //

//////////////////////////////////////////////////////////////////////////////////////////////

#define CHECKSYS 0 // Just after a reset, It will check whether all systems are up

#define INTHEBOX 1 // All systems are ok, probe is attached, waiting for probe to hit water

#define CALIBRATION 2 // The probe is enetered in water and is being calibrated

#define WIFI 3 // Wifi is being initialised. It looks for standard SSID to logOnn

#define INTERNET 4 // Attempting Internet Access

#define REGULAR 5 // Regular mode, Waterlevel is measured periodically and stored in RAM

#define INIT\_ERR 10 // During Initialisation a Subsytem didn't work or other error

#define CALIBRATION\_ERR 20 // Something went wrong in the calibration. Probe has to be taken out of the water

//////////////////////////////////////////////////////////////////////////////////////////////

// LED PATTERNS //

//////////////////////////////////////////////////////////////////////////////////////////////

#define LEDOFF 0 // Led is continuously off

#define LED\_CONT1 1 // Led is continuously on for 1 seconds, then off for 4 seconds

#define LED\_CONT2 2 // Led is continuously on for 2 seconds, then off for 3 seconds

#define LED\_CONT3 3 // Led is continuously on for 3 seconds, then off for 2 seconds

#define LED\_FLASH1S1 4 // Once per second the Led blinks for 25ms

#define LED\_FLASH1S5 5 // Once per 5 seconds the Led blinks for 25ms

#define LED\_FLASH2S5 6 // Twice per 5 seconds the led blinks for 25ms, interval 375ms

#define LED\_FLASH3S5 7 // Three times 5 seconds the led blinks for 25ms,interval 375ms

#define LED\_BLINK1S 8 // Led is one second on and one second off

#define LED\_BLINK2S 9 // Led is two seconds on and two seconds off

#define LED\_BLINK3S 10 // Led is three seconds on and three seconds off

#define LED\_FLASH1M5 11 // One time per 5 minutes the led blinks for 25ms

#define LED\_ON true // Led should be on

#define LED\_OFF false // Led should be off

//////////////////////////////////////////////////////////////////////////////////////////////

// DATA STRUCTURES //

//////////////////////////////////////////////////////////////////////////////////////////////

struct Timestamp {

byte hh, mm, ss, DD, MM, YY;

}; // BCD coded Time & Date info, these are updated in the time routine

struct Logvalue { // The way the logged values are stored

unsigned int Dnumber; // Day number assuming 1-1-1900 = 0

unsigned int Snumber; // Second number assuming 00:00:00 is 0 and 23:59:59 is 86399

int Wlevel;

};

//////////////////////////////////////////////////////////////////////////////////////////////

// GLOBAL VARIABLES //

//////////////////////////////////////////////////////////////////////////////////////////////

byte SensorState = 0;

unsigned long ConstantA=80962000L; // Default values, will be recalculated after calibration

unsigned long ConstantB=767L; // Default values, will be recalculated after calibration

boolean buttonHigh = false;

unsigned int TickCounter=0; // Keeps track of 25ms ticks, there are 2401 ticks in a second

boolean TickChange; // Semafore to indicate a Timer2 interrupt occurred

boolean SecondsOverflow = false; // Is set if one second is elapsed since previous reset of TickCounter

unsigned int daynumber; // 1-1-1900 is 0; 40909 is 1-1-2012, used as time-stamp in data logging.

unsigned secnumber; // time in seconds of the day, there are 86400 seconds in a day, used as times-stamp in data logging.

Timestamp Tstamp; // Multipurpose variable to store a BCD Timestamp in

Logvalue LogBuffer[100]; // The round robin buffer in which measurement values are stored with a timestamp

Logvalue LV; // Temp Logvalue storage

int LogbufferPtr=0; // Pointer that points to the next empty position in the data log buffer

byte RedLedPattern; // Indicator for the pattern the red led should signal

byte RedLedStep; // which step in the pattern of the red led

boolean RedLedOn; // true if Red Led is/should be on

unsigned int NextEventRedLed; // At which TickCount the Red Led should change state

byte GreenLedPattern; // Indicator for the pattern the green led should signal

byte GreenLedStep; // which step in the pattern of the green led

boolean GreenLedOn; // true if Green Led is/should be on

unsigned int NextEventGreenLed; // At which TickCount the Green Led should change state

byte Input\_Buffer[12]; // Input buffer to receive Data from Serial

//////////////////////////////////////////////////////////////////////////////////////////////

// Timer 2 Interrupt vector //

//////////////////////////////////////////////////////////////////////////////////////////////

#define TICK\_HANDLER TIMER2\_COMPA\_vect // Interruptvector for Timer2 Interrupts

//////////////////////////////////////////////////////////////////////////////////////////////

// Pro forma declaration of functions, Actual declarations can be found below the main loop //

//////////////////////////////////////////////////////////////////////////////////////////////

//General

void InitIO();

boolean AlreadyInitialised();

void SetCPUto3MHz();

//Sensor related

void InitCounter1();

void SensorOff();

void SensorOn();

boolean ProbeAttached();

boolean ProbeInWater();

void ClearTimer1();

unsigned long ReadCounter1();

long CaptureFreq();

int WaterLevel(long CF);

//Power management related

void PowerDownItem(int sel);

void PowerUpItem(int sel);

void PowerDownChip();

void GotoSleep();

void NoSleep();

//RTC related

void InitRTC();

byte decToBcd(byte val);

void setRTCtime(byte ss, byte mm, byte hh, byte dd, byte mon, byte yy);

void readRTCtime(byte \*ss, byte \*mm, byte \*hh, byte \*dow, byte \*dd, byte \*mon, byte \*yy);

//Time keeping related

void InitTimer2();

void IncrementSeconds();

void UpdateTime();

//Data handling related

void IncrementBufferPtr();

// Serial Data Input

byte Read\_Data(byte No\_Off\_Bytes); // Read a certain number of bytes from Serial connection

byte ConvertToBCD(byte CC1, byte CC2); // Convert ASCII chracters to BCD

void FlushInputBuffer(); // Flush UART from all pending data

boolean CheckConvert2Time(); // Check if Buffer has a valid Time and puts it in the Time Stamp

boolean CheckConvert2date(); // Check if buffer has a valid date and puts it in the Time Stamp

void ConvertLV2TSTMP(); // Converts the value in LV to a timestamp BCD code in Tstamp

void ConvertTSTMP2LV(); // Converts a timestamp BCD code in Tstamp to a datenumber and sec number LV

// Led Pattern Related

boolean SetLEDStep(int Tctr, unsigned int \*NLE, byte Pttrn, byte \*Lstp, boolean rst);

boolean SwitchPressed(boolean bH);

//////////////////////////////////////////////////////////////////////////////////////////////

// SETUP //

//////////////////////////////////////////////////////////////////////////////////////////////

void setup(){

Serial.begin(115200);

SetCPUto3MHz(); // only necessary when running on a 16MHz system. All delay() will take twice as long

InitIO(); // Set IO-ports in the right mode

InitCounter1(); // Set timer 1 as external counter

InitTimer2(); // Prepare timer 2 as interrupt source for waking up from sleep mode

SensorOff(); // Make sure Sensor is off

Wire.begin(0x00); // Initialise I2C protocol with Arduino as Master with Adress 00

sei(); // Enable interrupts from Timer2 to come through

PowerDownChip(); // All units are powered down untill needed. Only Timer2 remains active for the sleep mode

} // End Setup

//////////////////////////////////////////////////////////////////////////////////////////////

// MAIN LOOP //

//////////////////////////////////////////////////////////////////////////////////////////////

void loop(){

// The loop will only be entered after a Reset or when all is done that should be done in the 25ms cycle

// Any time left is used to go to sleep or in an active form of waiting

// Sleeping is only allowed when no Leds are on because in sleep-mode all IO is inactive

// Check if any Leds should be on

RedLedOn = (SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, false));

Serial.print("Red: ");

Serial.print(NextEventRedLed);

GreenLedOn = (SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, false));

Serial.print(" Green: ");

Serial.println(NextEventGreenLed);

// Turn Leds on or off

if (GreenLedOn) digitalWrite(GreenLedPin, HIGH);

else digitalWrite(GreenLedPin, LOW);

if (RedLedOn) digitalWrite(RedLedPin, HIGH);

else digitalWrite(RedLedPin, LOW);

buttonHigh = (digitalRead(SwitchPin) == 1); // for debugging only

// Ok, if any Leds are on we need to wait until the remainder of the 25ms have elapsed

if (RedLedOn || GreenLedOn) { // it can't go to sleep because the Leds would turn off

TickChange = false; // Reset TickChange flag

while (!TickChange); // and wait unttil the ISR has changed it to true

}

else { // If both Leds are off, it can go to sleep

TickChange = false; // Reset TickChange flag

while (!TickChange); // and wait unttil the ISR has changed it to true

// GotoSleep(); // The ISR will wake the processor up at this point

}

// Serial.print("TickCounter : "); // for debugging only

// Serial.println(TickCounter); // for debugging only

switch (SensorState){

//Dependong on the sensor state it will determine what to

case (CHECKSYS) : // This state is entered after a hardware reset or after downloading the scetch.

if (SwitchPressed(buttonHigh)) {

SensorState = INTHEBOX;

GreenLedPattern = LED\_BLINK1S; // Set Led pattern for Green Led

RedLedPattern = LED\_BLINK1S; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : INTHEBOX= ");

Serial.println(SensorState);

}

break; // End case CHECKSYS:

case (INTHEBOX): // Waiting for Calibration, the sensor will remain in this state until it is put in water for the first time

if (SwitchPressed(buttonHigh)){

SensorState = CALIBRATION;

GreenLedPattern = LED\_FLASH3S5; // Set Led pattern for Green Led

RedLedPattern = LED\_FLASH1S1; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : CALIBRATION= ");

Serial.println(SensorState);

}

break; // End case INTHEBOX:

case (CALIBRATION): // Calibration mode

if (SwitchPressed(buttonHigh)){

SensorState = WIFI;

GreenLedPattern = LED\_CONT1; // Set Led pattern for Green Led

RedLedPattern = LED\_BLINK1S; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : WIFI= ");

Serial.println(SensorState);

}

break; // End case CALIBRATION:

case (WIFI):

if (SwitchPressed(buttonHigh)){

SensorState = INTERNET;

GreenLedPattern = LED\_CONT2; // Set Led pattern for Green Led

RedLedPattern = LED\_BLINK2S; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : INTERNET= ");

Serial.println(SensorState);

}

break; // End case WIFI

case (INTERNET) :

if (SwitchPressed(buttonHigh)){

SensorState = REGULAR;

GreenLedPattern = LED\_FLASH3S5; // Set Led pattern for Green Led

RedLedPattern = LED\_CONT3; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : REGULAR= ");

Serial.println(SensorState);

}

break; // End case INTERNET

case (REGULAR) : // This is the regular mode the sensor will operate in

if (SwitchPressed(buttonHigh)){

SensorState = INIT\_ERR;

GreenLedPattern = LED\_FLASH2S5; // Set Led pattern for Green Led

RedLedPattern = LED\_FLASH1S5; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : INIT\_ERR= ");

Serial.println(SensorState);

}

break; // End case REGULAR

case (INIT\_ERR) :

if (SwitchPressed(buttonHigh)){

SensorState = 20;

GreenLedPattern = LED\_FLASH1S1; // Set Led pattern for Green Led

RedLedPattern = LED\_FLASH1S1; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : CALIBRATION\_ERR= ");

Serial.println(SensorState);

}

break; // End case INIT\_ERR:

case (CALIBRATION\_ERR) :

if (SwitchPressed(buttonHigh)){

SensorState = INTHEBOX;

GreenLedPattern = LED\_FLASH2S5; // Set Led pattern for Green Led

RedLedPattern = LED\_FLASH3S5; // Set Led pattern for Red Led

SetLEDStep(TickCounter, &NextEventRedLed, RedLedPattern, &RedLedStep, true);

SetLEDStep(TickCounter, &NextEventGreenLed, GreenLedPattern, &GreenLedStep, true);

Serial.print("SensorState : INTHEBOX= ");

Serial.println(SensorState);

}

break; // End case CALIBRATION\_ERR

} // End Switch Sensor State

} // End Main loop

//////////////////////////////////////////////////////////////////////////////////////////////

// Actual Declaration of Functions //

//////////////////////////////////////////////////////////////////////////////////////////////

void SetCPUto3MHz(){

CLKPR = 0x80; // Set Clock Prescale Change Enable

CLKPR = 0x01; // Set Clock Prescale Bits to 1 = divide by 2 = 8 MHz

}

// ..oOo.

boolean AlreadyInitialised()

// If the Real Time Clock has already been set, it should not be set again

// This checked whether the first two bytes in EEPROM have a random value or a specific value

// Only the first time this sketch is downloaded to the Arduino the sketch will ask to set the time

// All later (re)-loads of the sketch or whenever the reset button is pressed it will not ask for time and date

// A special reset sketch needs to be run first if this would be required (test purposes)

// the specific pattern in binary is b10101010 and b01010101, Ther is a chnace of 1 in 64.000 that this pattern is present by random.

// in itself this is not a problem, because when the programmer loads the sketch for the first time it will notice that there is no request

// for time and date info.

{

return (EEPROM.read(0)==0xAA)&&(EEPROM.read(1)==0x55);

}// End AlreadyInitialised

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Sensor routines, Transducer generates a frequency depending on waterlevel //

// Timer1 is set as an external counter. //

// If no probe is attached, freq is above 250 kHz //

// If a dry probe is attached, freq is around 100 kHz but depends on length of the probe //

//////////////////////////////////////////////////////////////////////////////////////////////

void InitIO()

// Initialises the IO-ports to the right mode

{

// First set all IO-pins as input with internal Pull-Up activated, to reduce power cnsupmtion.

DDRB = DDRD & B11000000; // Set PortB pins 0-5 in input mode (6 & 7 are in use for X-tal)

PORTB = B00111111 | PIND; // Activate internal Pull-up resistors PortB

DDRC = DDRC & B11000000; // Set PortC pins 0-5 in input mode, pin 6 is uset for hardware reset

PORTC = B00111111 | PIND; // Activate internal Pull-up resistors PortC

DDRD = DDRD & B00000011; // Set PortD pins 2-7 in input mode, Pin 0 & 1 are used for RXT and TXT and we'll leave them alone

PORTD = B11111100 | PIND; // Activate internal Pull-up resistors PortD

pinMode(SwitchPin, INPUT);

digitalWrite(SwitchPin, LOW); // de-activate internal pull-up resistor

pinMode(RedLedPin, OUTPUT);

pinMode(GreenLedPin, OUTPUT);

pinMode(SensorPin, OUTPUT);

pinMode(SensorEnablePin, OUTPUT);

}// End InitIO

// ..oOo.

void InitCounter1()

// Initialises Timer 1 to count external pulses of oscillator on pin5

// There is no Compare Output Match interrupt

{

TCCR1A = 0 ; // No Compare Output Mode, Prepare for CTC

TCCR1B = B00011111; // WGM13 = 1, WGM12 = 1; CS12 = 1, CS11 = 1, CS10 =1

} // End InitTimer1()

// ..oOo.

void SensorOff()

// Switch off sensor, it is powered directly from the IO pin

{

PowerDownItem(PDTimerCounter1); // TimerCounter 1 is not required when sensor is off

digitalWrite(SensorEnablePin, LOW); // Deactivate sensor to save power

} // End SensorOff()

// ..oOo.

void SensorOn()

// Switch on sensor, it is powered directly from the IO pin

{

PowerUpItem(PDTimerCounter0); // delay() uses counter 0

PowerUpItem(PDTimerCounter1); // Power up TimerCounter 1 to measure frequency

digitalWrite(SensorEnablePin, HIGH); // Activate sensor

delay(1); // wait 2ms for sensor to stabelise

PowerDownItem(PDTimerCounter0); // TimerCounter 0 (used for delay() )is not required anymore

} // End SensorOn()

// ..oOo.

void ClearTimer1()

// Make sure TimerCounter1 starts from zero

{

TCNT1 = 0;

}

// ..oOo.

unsigned long ReadTimer1()

{

return TCNT1 ;

}

// ..oOo.

boolean ProbeAttached(){ // Returns true if indeed probe is attached

return (CaptureFreq() < 250000l);

} // End ProbeAttached

// ..oOo.

boolean ProbeInWater(){ // Returns true if indeed probe is in water

return (CaptureFreq() < 125000l);

} // End ProbeInWater

// ..oOo.

long CaptureFreq()

#define COUNTINGPERIOD 100

#define COUNTINGPERIODSPERSEC 10

// measures the number of pulses in the counting period and converts it to a frequency

// Take care figures are consistent

{

long hlp;

PowerUpItem(PDTimerCounter0); // delay() uses TimerCounter 0

SensorOn(); //turn sensor on

ClearTimer1();

delay(COUNTINGPERIOD);

hlp = (long) ReadTimer1()\* COUNTINGPERIODSPERSEC;

SensorOff(); // turn sensor off

PowerDownItem(PDTimerCounter0); // TimerCounter 0 is not required anymore

return hlp;

}

// ..oOo.

int WaterLevel(long CF)

// Returns Waterlevel in 0.1mm based on the formula:

// h = a/f - b; Constants a and b are derived through an off-line calibration of the sensor

//

{

return (int) (ConstantA / CF - ConstantB);

}

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Powerdown routines, Most relevant info from the Atmeg datasheet, //

// PRR : Power reduction Register //

//////////////////////////////////////////////////////////////////////////////////////////////

void PowerDownItem(int sel)

{

switch (sel)

{

case PDADC:

PRR = PRR | B00000001; // set bit 0 in Power Reduction Register

break;

case PDUSART:

PRR = PRR | B00000010; // set bit 1 in Power Reduction Register

break;

case PDSPI:

PRR = PRR | B00000100; // set bit 2 in Power Reduction Register

break;

case PDTimerCounter1:

PRR = PRR | B00001000; // set bit 3 in Power Reduction Register

break;

case PDTimerCounter0:

PRR = PRR | B00100000; // set bit 5 in Power Reduction Register

break;

case PDTimerCounter2:

PRR = PRR | B01000000; // set bit 6 in Power Reduction Register

break;

case PDTWI:

PRR = PRR | B10000000; // set bit 7 in Power Reduction Register

break;

}

}

// ..oOo.

void PowerUpItem(int sel)

{

switch (sel)

{

case PDADC:

PRR = PRR & B11111110; // clear bit 0 in Power Reduction Register

break;

case PDUSART:

PRR = PRR & B11111101; // clear bit 1 in Power Reduction Register

break;

case PDSPI:

PRR = PRR & B11111011; // clear bit 2 in Power Reduction Register

break;

case PDTimerCounter1:

PRR = PRR & B11110111; // clear bit 3 in Power Reduction Register

break;

case PDTimerCounter0:

PRR = PRR & B11011111; // clear bit 5 in Power Reduction Register

break;

case PDTimerCounter2:

PRR = PRR & B10111111; // clear bit 6 in Power Reduction Register

break;

case PDTWI:

PRR = PRR & B01111111; // clear bit 7 in Power Reduction Register

break;

}

}

// ..oOo.

void PowerDownChip()

{

PowerDownItem(PDTimerCounter0);

PowerDownItem(PDTimerCounter1);

//PowerDownItem(PDTimerCounter2);

PowerDownItem(PDADC);

PowerDownItem(PDUSART);

PowerDownItem(PDSPI);

PowerDownItem(PDTWI);

}// End PowerDownChip

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Initialise Timer2 to CTC-mode, Refer to Atmel datasheet //

// Pre-scaler is set to 1024, i.e. counting freq is approx 40 Hz //

// When running on a 8-MHz board ofcourse //

//////////////////////////////////////////////////////////////////////////////////////////////

void InitTimer2(){

cli(); //stop interrupt

TCCR2A = 0;

TCCR2B = 0;

TCCR2A |= (1 << WGM21); // Set Timer2 in CTC mode

TCCR2B |= (1 << CS20); // Set Pre-scaler to 1024

TCCR2B |= (1 << CS21); //

TCCR2B |= (1 << CS22); //

TCNT2 = 0; // initialize counter value to 0

OCR2A = 195; // should be 24.984 milliseconds i.e. 40 wake-ups per second

TIFR2 = 2; //

TIMSK2 |= (1 << OCIE2A); // Enable Timer2 Interupts

//sei(); // allow interrupts

} //end InitTimer2

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Interrupt Service Routine for Timer2 interrupts. The interrupt wakes up the processor //

// and increments the tick counter. If 1 second is elapsed the SecondsOverflow flag is set //

//////////////////////////////////////////////////////////////////////////////////////////////

ISR(TIMER2\_COMPA\_vect){

TCNT2 = 0; // Restart Timer 2 from 0

NoSleep(); // Disable Sleep mode as fast as possible

TickCounter++; // Record that again 24.984 ms have elapsed

TickChange = true; // Record that Timer2 interrupt happened

if (TickCounter%80 == 0){ // There are 40 interrupts in one second has elapsed

SecondsOverflow = true; // The rest of the time keeping is handled separatly in which the flag is cleared

}

if (TickCounter >= 24000L) TickCounter = 0; // five minutes have elapsed

sei(); // allow interrupts

} //end ISR Timer2

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Routines to put the board to sleep //

// Atmel datasheet is the best source for understanding how it works //

//////////////////////////////////////////////////////////////////////////////////////////////

void GotoSleep(){

// Puts Arduino at sleep until it is awakened by the Timer2 Interrupt

sei(); // allow interrupts

SMCR = B00000001; // Enable Sleep mode

SMCR = B00000111; // Select Power save mode

} //end GotoSleep

// ..oOo.

void NoSleep(){

// Disables the sleepmide

SMCR = B00000000; // Disable Sleep mode

} //end NoSleep

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Time Keeping Routines //

//////////////////////////////////////////////////////////////////////////////////////////////

void UpdateTime(){

// The SecondsOverflow flag is set in the Timer2 Interrupt routine when 1 second has elapsed

if (SecondsOverflow) {

SecondsOverflow = false;

secnumber++;

} // End if SecondsOverflow

if (secnumber >= 86400) { // there are 86400 seconds in a day

secnumber = 0; // Start with a new day

daynumber++; // and increment daycounter

} // End if secnumber overflow

} // end Update time

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Real Time Clock Routines //

//////////////////////////////////////////////////////////////////////////////////////////////

void setRTCtime(byte ss, byte mm, byte hh, byte dd, byte mon, byte yy)

{ // BCD coding is assumed for each of the variables

// sets time and date data to DS3231

Wire.beginTransmission(RTC\_ADDRESS);

Wire.write(0); // set next input to start at the seconds register

Wire.write(ss); // seconds 0..59

Wire.write(mm); // minutes 0..59

Wire.write(hh); // hours 0..23

Wire.write(1); // day of week (not used)

Wire.write(dd); // date 1..31

Wire.write(mon); // month 1..12

Wire.write(yy); // year 1..99

Wire.endTransmission();

} // End setRTCtime

// ..oOo.

void readRTCtime(byte \*ss, byte \*mm, byte \*hh, byte \*dow, byte \*dd, byte \*mon, byte \*yy)

{

Wire.beginTransmission(RTC\_ADDRESS);

Wire.write(0); // set DS3231 reg. pointer to 0x00

Wire.endTransmission();

Wire.requestFrom(RTC\_ADDRESS, 7); // get seven bytes from RTC from reg 0x00

\*ss = Wire.read() & 0x7f; // blank off bit 7

\*mm = Wire.read();

\*hh = Wire.read() & 0x3f; // blank off bit 7 & 6

\*dow = Wire.read();

\*dd = Wire.read();

\*mon = Wire.read();

\*yy = Wire.read();

} // End readRTCtime

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Data Handling and Storage //

//////////////////////////////////////////////////////////////////////////////////////////////

void IncrementBufferPtr(){

LogbufferPtr++;

if (LogbufferPtr >= 100) LogbufferPtr =0;

}

//////////////////////////////////////////////////////////////////////////////////////////////

// Serial Data Input Functions //

//////////////////////////////////////////////////////////////////////////////////////////////

byte Read\_Data(byte No\_Off\_Bytes){

byte BufferPtr = 0; // points to the first empty space in the buffer

boolean All\_Received = false;

while ((Serial.available() > 0) && !All\_Received) { // Read 8 bytes or until an End of Line

Input\_Buffer[BufferPtr] = Serial.read(); // read the incoming byte and store it in the buffer

delayMicroseconds(100); // Aparently this delay is necessary to empty the input buffer

All\_Received = ((Input\_Buffer[BufferPtr] == 0x0A) || BufferPtr == No\_Off\_Bytes); // check if we received 8 or an EOL character

if (!All\_Received) BufferPtr++; // if not ready, increment Buffer Pointer to point to the next empty Space

else {

FlushInputBuffer(); // discard rest of input buffer

}

}

return BufferPtr;

}

// ..oOo.

void FlushInputBuffer(){

while (Serial.available()>0) {

Serial.read();

delayMicroseconds(100); // The delay is necessary because the incoming stream is sometimes too slow and the buffer is empty whilst still receiving a byte

}

}

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Date Conversion Functions //

//////////////////////////////////////////////////////////////////////////////////////////////

boolean CheckConvert2date(){

boolean ok = false;

ok = ((Input\_Buffer[2] == 45) && // should be "-"

(Input\_Buffer[5] == 45)); // should also be "-"

if (!ok) return false;

ok = ((Input\_Buffer[0] == 48) || // '0' is allowed

(Input\_Buffer[0] == 49) || // '1' is allowed

(Input\_Buffer[0] == 50) || // '2' is allowed

(Input\_Buffer[0] == 51)); // '3' is allowed

if (!ok) return false;

ok = ((Input\_Buffer[1] >= 48) &&

(Input\_Buffer[1] <= 57)); // '0'.. '9' are allowed

if (!ok) return false;

ok = ((Input\_Buffer[3] == 48) || // '0' is allowed

(Input\_Buffer[3] == 49)); // '1' is allowed

if (!ok) return false;

ok = ((Input\_Buffer[4] >= 48) &&

(Input\_Buffer[4] <= 57)); // '0'.. '9' are allowed

if (!ok) return false;

ok = (Input\_Buffer[6] == 49); // only '1' is allowed

if (!ok) return false;

ok = ((Input\_Buffer[7] == 57) || // '9' is allowed

(Input\_Buffer[7] == 56) || // '8' is allowed

(Input\_Buffer[7] == 55) || // '7' is allowed

(Input\_Buffer[7] == 54) || // '6' is allowed

(Input\_Buffer[7] == 53)); // '5' is allowed

if (!ok) return false;

// Convert to BCD

Tstamp.DD = ConvertToBCD(Input\_Buffer[0],Input\_Buffer[1]);

Tstamp.MM = ConvertToBCD(Input\_Buffer[3],Input\_Buffer[4]);

Tstamp.YY = ConvertToBCD(Input\_Buffer[6],Input\_Buffer[7]);

// Check consistency of dates

ok = (Tstamp.MM <= 0x12); // only 12 months in a year

if (!ok) return false;

if ((Tstamp.MM == 0x02) && (Tstamp.YY != 0x16)) ok = ((Tstamp.DD >= 01) && (Tstamp.DD <= 0x28)); // not a leap year, february can have up to 28 days

if (!ok) return false;

if ((Tstamp.MM == 0x02) && (Tstamp.YY = 0x16)) ok = ((Tstamp.DD >= 01) && (Tstamp.DD <= 0x29)); // 2016 is a leap year, february can have up to 29 days

if (!ok) return false;

if

((Tstamp.MM == 0x01) || // these months can have up to 31 days

(Tstamp.MM == 0x03) ||

(Tstamp.MM == 0x05) ||

(Tstamp.MM == 0x07) ||

(Tstamp.MM == 0x08) ||

(Tstamp.MM == 0x10) ||

(Tstamp.MM == 0x12)) ok = ((Tstamp.DD >= 0x01) && (Tstamp.DD <= 0x31));

if (!ok) return false;

if

((Tstamp.MM == 0x04) || // these months can have up to 30 days

(Tstamp.MM == 0x06) ||

(Tstamp.MM == 0x09) ||

(Tstamp.MM == 0x11)) ok = ((Tstamp.DD >= 0x01) && (Tstamp.DD <= 0x30));

if (!ok) return false;

return ok;

} // End CheckConvert2date

// ..oOo.

boolean CheckConvert2Time(){

boolean ok = false;

ok = (Input\_Buffer[2] == 58); // should be ":"

if (!ok) return false;

ok = ((Input\_Buffer[0] == 48) || // '0' is allowed

(Input\_Buffer[0] == 49) || // '1' is allowed

(Input\_Buffer[0] == 50)); // '2' is allowed

if (!ok) return false;

ok = ((Input\_Buffer[1] >= 48) &&

(Input\_Buffer[1] <= 57)); // '0'.. '9' are allowed

if (!ok) return false;

ok = ((Input\_Buffer[3] >= 48) &&

(Input\_Buffer[3] <= 53)); // '0'.. '5' are allowed

if (!ok) return false;

ok = ((Input\_Buffer[4] >= 48) &&

(Input\_Buffer[4] <= 57)); // '0'.. '9' are allowed

if (!ok) return false;

// Convert to BCD

Tstamp.hh = ConvertToBCD(Input\_Buffer[0],Input\_Buffer[1]);

Tstamp.mm = ConvertToBCD(Input\_Buffer[3],Input\_Buffer[4]);

// Check consistency of dates

ok = (Tstamp.hh <= 0x23); // only 23 full hours in a day

if (!ok) return false;

ok = (Tstamp.mm <= 0x59); // only 59 full minutes in an hour

if (!ok) return false;

return ok;

} // End CheckConvert2Time

// ..oOo.

byte ConvertToBCD(byte CC1, byte CC2){

//Assumes CC1 and CC2 represent a ASCII char 0..9 or A..F

// 0 = 0x30 (48 DEC) up to 9 = 0x39 (57 DEC), A = 0x41 (65 DEC)up to F 0x47 (70 DEC)

byte hlp =0;

if ((CC1 >= 48) && (CC1 <= 57)){

hlp = (CC1 - 48)\*16;

}

if ((CC1 >= 65) && (CC1 <= 70)){

hlp = (CC1 - 55)\*16;

}

if ((CC2 >= 48) && (CC2 <= 57)){

hlp = hlp + (CC2 - 48);

}

if ((CC2 >= 65) && (CC2 <= 70)){

hlp = hlp + (CC2 - 55);

}

} // End ConvertToBCD

// ..oOo.

byte decToBcd(byte val){

return (val/10\*16 + val%10);

}// End decToBcd

// ..oOo.

byte BCDTodec(byte val){

return ((val/16)\*10 + val%16);

}

// ..oOo.

void ConvertLV2TSTMP(){

long LDD;

Tstamp.hh = decToBcd((byte)(LV.Snumber / 3600));

Tstamp.mm = decToBcd((byte)((LV.Snumber % 3600) / 60));

Tstamp.ss = decToBcd((byte)((LV.Snumber % 3600) % 60));

LDD = (long) (LV.Dnumber-36525U); // # days since 01-01-2000 (a leap year)

Tstamp.YY = 00; // first assume it is 1-1-2000

while (LDD > 1461){ // if there are more than 1461 days left a full series of 4 years has passed, one of which is a leap year

Tstamp.YY = Tstamp.YY + 4;

LDD = LDD - 1461;

} // DD <= 1461

if (LDD > 1096) { // The first year of the group is a leap year with 366 days

Tstamp.YY = Tstamp.YY + 3;

LDD = LDD - 1096; // DD <= 365

} // DD <= 1096

if (LDD > 731) { // The first year of the group is a leap year with 366 days

Tstamp.YY = Tstamp.YY + 2;

LDD = LDD - 731;// DD <= 365

} // DD <= 731

if (LDD > 366) { // The first year of the group is a leap year with 366 days

Tstamp.YY = Tstamp.YY + 1;

LDD = LDD - 366;// DD <= 366

} // DD <= 366

// DD is now the day number in the year YY

Tstamp.MM = 1 ; // This is the first month of the year

if ((LDD>31)&&(Tstamp.MM==1)){ // It is not January

LDD = LDD - 31;

Tstamp.MM++; // it might be February

}

if (((Tstamp.YY%4)==0)&&(Tstamp.MM==2)) {// it is a Leap Year

if ((LDD>29)&&(Tstamp.MM==2)){ // It is not February in a Leap Year

LDD = LDD - 29;

Tstamp.MM++; // it might be March

}

}

else { // it is not a Leap Year

if ((LDD>28)&&(Tstamp.MM==2)){ // It is not February in a non Leap Year

LDD = LDD - 28;

Tstamp.MM++; // it might be March

}

}

if ((LDD>31)&&(Tstamp.MM==3)){

LDD = LDD - 31; // It is not March

Tstamp.MM++; // it might be April

}

if ((LDD>30)&&(Tstamp.MM==4)){

LDD = LDD - 30; // It is not April

Tstamp.MM++; // it might be May

}

if ((LDD>31)&&(Tstamp.MM==5)){

LDD = LDD - 31; // It is not May

Tstamp.MM++; // it might be June

}

if ((LDD>30)&&(Tstamp.MM==6)){

LDD = LDD - 30; // It is not June

Tstamp.MM++; // it might be July

}

if ((LDD>31)&&(Tstamp.MM==7)){

LDD = LDD - 31; // It is not July

Tstamp.MM++; // it might be Aug

}

if ((LDD>31)&&(Tstamp.MM==8)){

LDD = LDD - 31; // It is not Aug

Tstamp.MM++; // it might be Sept

}

if ((LDD>30)&&(Tstamp.MM==9)){

LDD = LDD - 30; // It is not Sept

Tstamp.MM++; // it might be Oct

}

if ((LDD>31)&&(Tstamp.MM==10)){

LDD = LDD - 31; // It is not Oct

Tstamp.MM++; // it might be Nov

}

if ((LDD>30)&&(Tstamp.MM==11)){

LDD = LDD - 30; // It is not Nov

Tstamp.MM++; // it must be December

}

Tstamp.DD = decToBcd((byte)(LDD));

Tstamp.MM = decToBcd((byte)(Tstamp.MM));

Tstamp.YY = decToBcd((byte)(Tstamp.YY));

}

// ..oOo.

void ConvertTSTMP2LV(){

LV.Snumber = BCDTodec(Tstamp.ss) + BCDTodec(Tstamp.mm)\*60 + BCDTodec(Tstamp.hh)\*3600;

LV.Dnumber = 36526; // 1-1-2000

LV.Dnumber = LV.Dnumber + (BCDTodec(Tstamp.YY)/4)\*1461; // There are 1461 days in 4 years, one of which is a leap day

LV.Dnumber = LV.Dnumber + (BCDTodec(Tstamp.YY)%4)\*365; // all remaining full years have 365 days

if ((BCDTodec(Tstamp.YY)%4) == 0) LV.Dnumber--;

switch (Tstamp.MM) {

case (0x01): // It's Januari

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD);

break;

case (0x02): // It's Februari

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 ;

break;

case (0x03): // It's March

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x04): // It's April

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x05): // It's May

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x06): // It's June

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x07): // It's July

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31 + 30;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x08): // It's August

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31 + 30 + 31;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x09): // It's September

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x10): // It's October

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x11): // It's November

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

case (0x12): // It's December

LV.Dnumber = LV.Dnumber + BCDTodec(Tstamp.DD) + 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30;

if (BCDTodec(Tstamp.YY)%4 == 0) LV.Dnumber++;

break;

} // end switch on months

} // End ConvertTSTMP2LV

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Led Pattern Manipulation //

//////////////////////////////////////////////////////////////////////////////////////////////

boolean SetLEDStep(int Tctr, unsigned int \*NLE, byte Pttrn, byte \*Lstp, boolean rst){

const PROGMEM int TimePerStep[] ={

0, 23999, // 0, LEDOFF

40, 160, // 2, LED\_CONT1

80, 120, // 4, LED\_CONT2

120, 80, // 6, LED\_CONT3

1, 39, // 8, LED\_FLASH1S1

1, 199, // 10, LED\_FLASH1S5

1, 15, 1, 183, // 12, LED\_FLASH2S5

1, 15, 1, 15, 1, 167, // 16, LED\_FLASH3S5

40, 40, // 22, LED\_BLINK1S

80, 80, // 24, LED\_BLINK2S

120, 120, // 26, LED\_BLINK3S

1, 15, 1, 11983, // 28, LED\_FLASH2M5

2, 2 }; // 32, Not in Use

const PROGMEM byte PatternStart[] ={

0, // 0, LEDOFF

2, // 1, LED\_CONT1

4, // 2, LED\_CONT2

6, // 3, LED\_CONT3

8, // 4, LED\_FLASH1S1

10, // 5, LED\_FLASH1S5

12, // 6, LED\_FLASH2S5

16, // 7, LED\_FLASH3S5

22, // 8, LED\_BLINK1S

24, // 9, LED\_BLINK2S

26, // 10, LED\_BLINK3S

28, // 11, LED\_FLASH2M5

32, // 12, Not in use

};

if (rst){

\*NLE = Tctr + 2;

\*Lstp = PatternStart[Pttrn];

}

if (Tctr == (int) \*NLE){ // It is time to move to the next period so load the parameters for the next period

\*Lstp = \*Lstp + 1; // increment ptr

if (\*Lstp >= PatternStart[Pttrn + 1]){

\*Lstp = PatternStart[Pttrn];

}

\*NLE = (unsigned int) (Tctr + TimePerStep[\*Lstp]); // Set time for next Led event

if (\*NLE >= 24000) \*NLE -= 24000;

} //End If

return \*Lstp % 2 == 0; // if even we switch the led on

} // End SetLEDStep

// ..oOo.

//////////////////////////////////////////////////////////////////////////////////////////////

// Read Switch //

//////////////////////////////////////////////////////////////////////////////////////////////

boolean SwitchPressed(boolean bH){

if (digitalRead(SwitchPin) == 0) return bH; // only returns true if the switch is released after being pressed

else return false;

}