```
'----------------------------------------
'File: eeprom.luna
'----------------------------------------
'LunaAVR v2014 R2.4
'----------------------------------------
'change log
'2015-04-14 de0508, Uwe
'----------------------------------------
'Systemsettings
'----------------------------------------
const F_CPU = 11059200
//avr.Device = atmega328p
avr.Device = atmega168
avr.Clock = F_CPU
avr.Stack = 128

'----------------------------------------
'Macros
'----------------------------------------
#define BV(n) as (1<<(n))
#define NBV(n) as (not BV(n))

#define HZ(f)  as long( f )
#define KHZ(f) as (long(f) *1000)
#define MHZ(f) as (long(f) *1000*1000)
#define GHZ(f) as MHZ((f) *1000)

'----------------------------------------
'Typedefinitionen
'----------------------------------------
struct uint64_t
 long lo32
 long hi32
endstruct

'----------------------------------------
'Modulaktivierung
'----------------------------------------
#define USE_SOFTWARE_I2C as 1

'----------------------------------------
'Library
'----------------------------------------
#include "config.luna"

'----------------------------------------
'Init Ports
'----------------------------------------
LED_POWER.mode = output
LED_POWER = LED_OFF

' Software SPI
#if 0
ADF4350_DATA.mode = output, low
ADF4350_CLK.mode = output, low

U4_ADF4350_LE.mode = output, high
U7_ADF4350_LE.mode = output, high

U4_ADF4350_LD.mode = input, pullup
U7_ADF4350_LD.mode = input, pullup
#endif
```

```
'--------------------------------------
'Init Module
'--------------------------------------
avr.Interrupts.Enable

'--------------------------------------
'Uart0
'--------------------------------------
RXD_PIN.mode = input, pullup
TXD_PIN.mode = output, high

uart.interface = uart0
uart0.Baud = FA_NWT_BAUDRATE
uart0.Rxd.fifo = 16
uart0.Txd.Fifo = 256
uart0.Rxd.Enable
uart0.Txd.Enable

'--------------------------------------
'Software I2C
'--------------------------------------
const I2C_BYTE_AVAILABLE = 65536
const I2C_PAGE_SIZE = 128
const I2C_PAGE_AVAILABLE = word(I2C_BYTE_AVAILABLE
/I2C_PAGE_SIZE)

#if USE_SOFTWARE_I2C
asm
; wordaround
.set SoftTwiInit = 1
endasm

TWI.Init()
if TWI.Test() then
 uart0.print "twi error: after TWI.Init()"
endif
#endif

LED_POWER = LED_ON

'--------------------------------------
'globale Variable
'--------------------------------------
dim i2c_buffer(I2C_PAGE_SIZE-1) as byte
dim n as word

'--------------------------------------
' Mainprogramm
'--------------------------------------
wait 2

n = 0
' es reicht aus nur 16 anstatt I2C_PAGE_AVAILABLE Seiten (page) zu lesen
while (n < 16)
 I2C_Read_Page_as_Byte( n )
 n += 1
wend

halt()
'--------------------------------------
'end
```

```
'--------------------------------------

procedure I2C_Read_Page_Scale( page as word )
dim m as word
dim w16, adr as word
dim f as single

adr = word(page * I2C_PAGE_SIZE)

TWI.Get_Block( adr, avr.i2c_buffer(), I2C_PAGE_SIZE )
if (TWI.twi_error) then
 uart0.print "twi error: TWI.Get_Block()"
endif

m = 0
while (m < I2C_PAGE_SIZE)
if (m and 1) then ' highbyte
 ' Adresse
 uart0.print str( word(adr+m) );";";
 w16.highbyte = avr.i2c_buffer(m)
 f = single(w16) / 65536.0
 uart0.Print format("0.0000", f)
else ' lowbyte
 w16.lowbyte = avr.i2c_buffer(m)
endif

m += 1
wend
endproc

'--------------------------------------

procedure I2C_Read_Page_as_Byte( page as word )
const PRINT_BYTE_PER_LINE = 16

dim m as word
dim b as byte
dim adr as word

adr = word(page * I2C_PAGE_SIZE)
TWI.Get_Block( adr, avr.i2c_buffer(), I2C_PAGE_SIZE )
if (TWI.twi_error) then
 uart0.print "twi error: TWI.Get_Block()"
endif

m = 0
while (m < I2C_PAGE_SIZE)
if ((m and (PRINT_BYTE_PER_LINE-1)) = 0) then
 uart0.print HEX( word(adr+m) );": ";
endif

b = avr.i2c_buffer(m)
//uart0.WriteByte( b )

uart0.Print HEX( b );" ";

m += 1

if ((m and (PRINT_BYTE_PER_LINE-1)) = 0) then //(m mod 16) alle 16 Byte ein
CRLF
 uart0.print ""
endif
```

```
wend
endproc

'--------------------------------------

procedure I2C_Read_Page_as_Word( page as word )
const PRINT_BYTE_PER_LINE = 16

dim m as word
dim w16 as word
dim adr as word

adr = word(page * I2C_PAGE_SIZE)

TWI.Get_Block( adr, avr.i2c_buffer(), I2C_PAGE_SIZE )

if (TWI.twi_error) then
 uart0.print "twi error: TWI.Get_Block()"
endif

m = 0
while (m < I2C_PAGE_SIZE)
if ((m and (PRINT_BYTE_PER_LINE-1)) = 0) then
 uart0.print HEX( word(adr+m) );": ";
endif

if (m and 1) then ' highbyte
 w16.highbyte = avr.i2c_buffer(m)
 uart0.Print HEX( w16 );" ";
else ' lowbyte
 w16.lowbyte = avr.i2c_buffer(m)
endif

m += 1
if ((m and (PRINT_BYTE_PER_LINE-1)) = 0) then //(m mod 16) alle 16 Byte ein
CRLF
 uart0.print ""
endif

wend
endproc

'--------------------------------------
'Klassen laden
'--------------------------------------

#include "class/i2c.luna" //Software TWI - I2C Eeprom
```

```
'---------------------------------------
'File: config.luna
'---------------------------------------

'---------------------------------------
' Port Definitionen
'---------------------------------------
'PortC
'Atmel (H344) 2FC I2C EEprom - AT24C512C
'I2C

#define SW_I2C_SDA as portc.0
#define SW_I2C_SCL as portc.1
#define SW_I2C_SPEED as KHZ(100)
#define SW_I2C_BASE_ADDR as 0b1010000
#define SW_I2C_ADDR as (SW_I2C_BASE_ADDR<<1)
#define SW_I2C_READ_ADDR as (SW_I2C_ADDR +1)
#define SW_I2C_WRITE_ADDR as (SW_I2C_ADDR +0)
```

```
'---------------------------------------
'File: i2c.luna
'---------------------------------------
'change log
'2015-04-04 de0508, Uwe
'---------------------------------------

#if 0
avr.Device = atmega328p
#endif


#library "Library/SoftTwi.interface"

'---------------------------------------
'Klasse TWI
'---------------------------------------
class TWI
'---------------------------------------
'Konstanten
'---------------------------------------

'---------------------------------------
'Variable
'---------------------------------------
dim twi_data8   as byte
dim twi_data16  as word
dim twi_error   as byte

'---------------------------------------
'Init()
'---------------------------------------
procedure Init()
 SoftTwi.PinScl = SW_I2C_SCL
 SoftTwi.PinSda = SW_I2C_SDA
 SoftTwi.Delay  = byte(MHZ(1) /SW_I2C_SPEED) 'us
 SoftTwi.Init
endproc

'---------------------------------------
'Test()
'---------------------------------------
```

```
function Test() as byte
 void Get_Byte( 0 )
 return twi_error
endfunc

'--------------------------------------
'Get_Byte( adr as word ) as byte
'--------------------------------------
function Get_Byte( adr as word ) as byte
 wi_data8 = 0
 twi_error = false

 if softtwi.start(SW_I2C_WRITE_ADDR) then
  softtwi.WriteByte adr.HighByte
  softtwi.WriteByte adr.LowByte
  softtwi.stop
  softtwi.start(SW_I2C_READ_ADDR)
  twi_data8 = softtwi.ReadByteNak
  softtwi.stop
 else
  twi_error = true
 endif

 return twi_data8
endfunc

'--------------------------------------
'Get_Next_Byte() as byte
'--------------------------------------
function Get_Next_Byte() as byte
 twi_data8 = 0
 twi_error = false

 if softtwi.start(SW_I2C_READ_ADDR) then
  twi_data8 = softtwi.ReadByteNak
  softtwi.stop
 else
  twi_error = true
 endif

 return twi_data8
endfunc

'--------------------------------------
'Get_Word( adr as word ) as word
'--------------------------------------
function Get_Word( adr as word ) as word
 twi_data16 = 0
 twi_error = false

 if softtwi.start(SW_I2C_WRITE_ADDR) then
  softtwi.WriteByte adr.HighByte
  softtwi.WriteByte adr.LowByte
  softtwi.stop

  softtwi.start(SW_I2C_READ_ADDR)
  twi_data16.highbyte = softtwi.readbyte
  twi_data16.lowbyte = softtwi.ReadByteNak
  softtwi.stop
 else
  twi_error = true
```

```
 endif

 return twi_data16
endfunc

'--------------------------------------
' Get_Next_Word() as word
'--------------------------------------
function Get_Next_Word() as word
 twi_data16 = 0
 twi_error  = false

 if softtwi.start(SW_I2C_READ_ADDR) then
  twi_data16.highbyte = softtwi.readbyte
  twi_data16.lowbyte = softtwi.ReadByteNak
  softtwi.stop
 else
  twi_error = true
 endif

 return twi_data16
endfunc

'--------------------------------------
' Get_Block( adr as word, byRef buffer() as byte, size as word )
'--------------------------------------
procedure Get_Block( adr as word, byRef buffer() as byte, size as word )
 dim idx as word
 twi_error = false

 if softtwi.start(SW_I2C_WRITE_ADDR) then
  softtwi.WriteByte adr.HighByte
  softtwi.WriteByte adr.LowByte
  softtwi.stop

  softtwi.start(SW_I2C_READ_ADDR)
  idx = 0
  while (idx < (size -1))
   buffer(idx) = softtwi.readbyte
   idx += 1
  wend

  buffer(idx) = softtwi.ReadByteNak
  softtwi.stop
 else
  twi_error = true
 endif
endproc

endclass
```