PCF8563

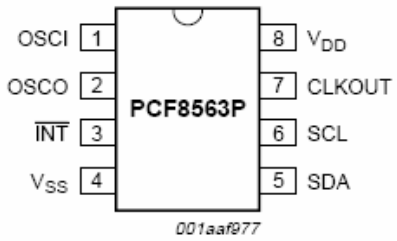
Real time clock/calendar

Rev. 01 — 15 February 2010 Application note by Nando Uitterlinden

1. Algemene omschrijving

De PCF8563 is een CMOS real time klok/kalender die geoptimaliseerd is voor laag vermogen verbruik. Het IC bevat ook een programmeerbare klok uitgang, interrupt uitgang en een laagspanning detectie. Alle adressen en data worden serieel verzonden over een “two-line bidirectional I2C-bus”. De maximale bus snelheid is 400 kbit/s. Het interne “word address register” wordt automatisch opgehoogd na elke gelezen en geschreven byte.

2. Pinning informatie



Pin configuration DIP8

Table 2. Pin description

Symbol	Pin		Description
	DIP8, SO8, TSSOP8	HVSON10	
OSCI	1	1	oscillator input
OSCO	2	2	oscillator output
n.c	-	3	not connected
$\overline{INT}$	3	4	interrupt output (open-drain; active LOW)
V_{SS}	4	5	ground
SDA	5	6	serial data input and output
SCL	6	7	serial clock input
CLKOUT	7	8	clock output, open-drain
V_{DD}	8	9	positive supply voltage
n.c	-	10	not connected

3. Adres informatie

Voordat er data verzonden wordt over de I2C bus moet het apparaat dat moet reageren eerst geadresseerd worden. Het adresseren vindt altijd plaats in de eerste byte na de start procedure. De PCF8563 gedraagt zich als slave receiver of slave transmitter. Om deze reden is het kloksignaal SCL enkel een ingang, maar het data signaal SDA is een bidirectionele lijn. Het adres van de PCF8563 is hieronder weer gegeven.

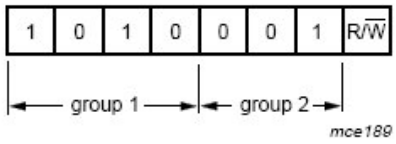


Fig 13. Slave address

8 bit Address for writing is 0xA2, 8 bit Address for reading is 0xA3.
7 bit Address is 0x51.

4. Register informatie

De verschillende soorten tijd informatie, zoals seconden, minuten en uren worden op afzonderlijke plaatsen in het PCF8563 register opgeslagen. Om tijd informatie uit de PCF8563 uit te lezen moet de master aangeven welk register gewenst wordt. De master geeft dit aan door eerst het register nummer naar de slave te schrijven en vervolgens van de slave te lezen. In de tabel hieronder is de indeling van het register weergegeven.

Table 3. Formatted registers overview

Bit positions labelled as x are not implemented. Bit positions labelled with 0 should always be written with logic 0; if read they could be either logic 0 or logic 1.

Address	Register name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
00h	control/status 1	TEST1	0	STOP	0	TESTC	0	0	0
01h	control/status 2	0	0	0	TI/TP	AF	TF	AIE	TIE
02h	seconds	VL	<seconds 00 to 59 coded in BCD>						
03h	minutes	x	<minutes 00 to 59 coded in BCD>						
04h	hours	x	x	<hours 00 to 23 coded in BCD>					
05h	days	x	x	<days 01 to 31 coded in BCD>					
06h	weekdays	x	x	x	x	x	<weekdays 0 to 6>		
07h	months/century	C	x	x	<months 01 to 12 coded in BCD>				
08h	years	<years 00 to 99 coded in BCD>							
09h	minute alarm	AE	<minute alarm 00 to 59 coded in BCD>						
0Ah	hour alarm	AE	x	<hour alarm 00 to 23 coded in BCD>					
0Bh	day alarm	AE	x	<day alarm 01 to 31 coded in BCD>					
0Ch	weekday alarm	AE	x	x	x	x	<weekday alarm 0 to 6>		
0Dh	CLKOUT control	FE	x	x	x	x	x	FD1	FD0
0Eh	timer control	AT	x	x	x	x	x	TD1	TD0
0Fh	timer	<timer countdown value>							

5. Interpretatie en lezen van de data

In dit voorbeeld gaan we een klok maken die uren, minuten en seconde informatie weergeeft. De registers die uitgelezen moeten worden zijn 0x02 voor secondes, 0x03 voor minuten en 0x04 voor uren. De seconden data is 7 bits groot. Dit is rechts uitgelijnd in een 8 bit register, bit <6:0>. Bit <7> wordt gebruikt om aan te geven of de integriteit van de klok gegarandeerd kan worden, dit zullen wij in dit voorbeeld niet behandelen. Bit <7> moet dus genegeerd worden, dit doen we door een bitmasker over de data te leggen. De uren data is 6 bits groot, hierbij moeten bit <7:6> genegeerd worden.

Voordat we de data uit gaan lezen moeten we eerst weten hoe we de data moeten interpreteren. De meest significante nibble in het register vertegenwoordigt de tientallen, het minst significante nibble vertegenwoordigd de eenheden. Wanneer het seconde register bijvoorbeeld de waarde 0x59h bevat betekend dit 59 seconden en niet de decimale voorstelling 89d. De tientallen kunnen in een variabele gezet worden, daarbij kan door 4 right shifts of delen door 16 de seconde informatie eruit gehaald worden en de waarde in het juiste bereik gezet worden. Hierbij moet wel opgelet worden dat bit <8> genegeerd dient te worden. De secondes kunnen uit de data geïsoleerd worden door een bitmasker te nemen die bit <3:0> overneemt en de rest elimineert. In programmacode ziet dit er als volgt uit in het geval van seconden en minuten:

```
Tientallen = ((0x59 >> 4) & 0x07);
Eenheden = (0x59 & 0x0f);
```

In het geval van uren ziet dit er als volgt uit:

```
Tientallen = ((0x12 >> 4) & 0x03);
Eenheden = (0x12 & 0x0f);
```

Een lees sequence van het seconde register ziet er als volgt uit, de master genereert een start conditie, hij schrijft het 7 bits adres met daarachter een 0, dit betekent dat er een schijfactie door de master volgt. Vervolgens schrijft het nummer van het register dat hij wenst uit te lezen. De master genereert een repeated start conditie, en schrijft het 7 bit slave adres met daarachter een 1, dit betekend dat er een leesactie door de master volgt. Hierna genegeerd de master klokpulsen, op deze klokpulsen klokt de slave de waarde van het gevraagde register naar buiten. Indien de master niet meer bytes uit wil lezen moet hij een NAK genereren door de SDA lijn hoog te maken, de transmitter dient nu de SDA lijn hoog te laten waardoor de master af kan sluiten door een stop te genereren.

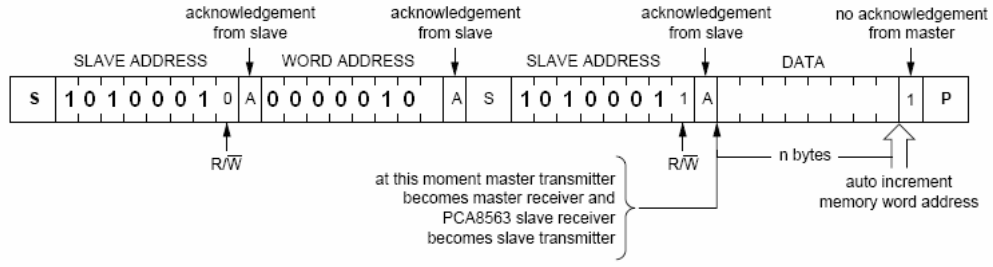


Fig 15. Master reads after setting word address (write word address; read data)

6. Toegepast voorbeeld met ATmega16

```
#define RTC_I2C_ADDR 0x51 //7 bit

int main()
{
      unsigned char sec_one, sec_ten, min_one, min_ten, hour_one,
                        hour_ten, buffer2;
      eeprom_init();

      //rtc run, normal mode
      eeprom_write_byte(RTC_I2C_ADDR, 0x00, 0x00);
      delay_ms(10);

      while(1)
      {
            // read seconds reg.
            eeprom_read_byte(RTC_I2C_ADDR, 0x02, &buffer2);
            delay_ms(10); //rest i2c bus

            sec_ten = ((buffer2 >> 4) & 0x07);
            sec_one = (buffer2 & 0x0f);

            // read minutes reg.
            eeprom_read_byte(RTC_I2C_ADDR, 0x03, &buffer2);
            delay_ms(10); //rest i2c bus

            min_ten = ((buffer2 >> 4) & 0x07);
            min_one = (buffer2 & 0x0f);

            // read hours reg
            eeprom_read_byte(RTC_I2C_ADDR, 0x04, &buffer2);
            delay_ms(10); //rest i2c bus

            hour_ten = ((buffer2 >> 4) & 0x03);
            hour_one = (buffer2 & 0x0f);


            delay_ms(100);
      }


}
```