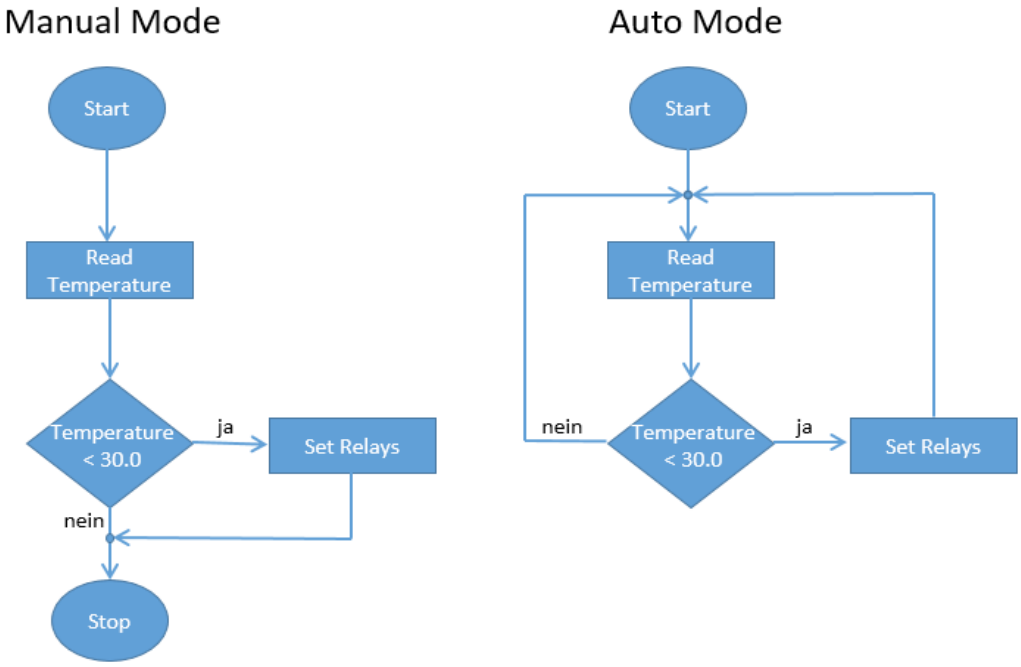
Application Note

Verwendung der PiXtend C-Library "pxdev"



APP-PX-220

Stand 04.04.2017, V1.03

Qube Solutions UG (haftungsbeschränkt)
Arbachtalstr. 6, 72800 Eningen, Germany
http://www.qube-solutions.de/
http://www.pixtend.de

Versionshistorie

Version	Beschreibung	Bearbeiter
1.00	Dokument erstellt	PT
1.01	Dokument aktualisiert & überarbeitet	TG
1.02	Quelltexte überarbeitet und formatiert	TG
1.03	Beschreibung von OutputData.wPwm0/1 korrigiert	TG

Inhaltsverzeichnis

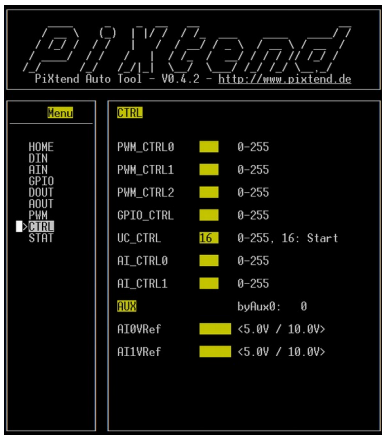
1. Einleitung

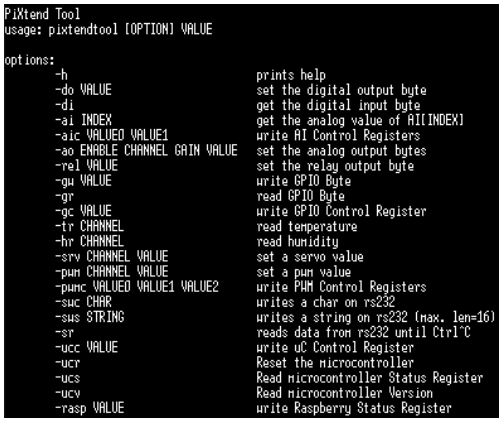
Mit der **PiXtend C-Library** („**pxdev**“) werden Sie in die Lage versetzt eigene Linux-Programme für Ihr PiXtend-Board zu programmieren bzw. die Funktionalitäten in Ihre eigenen Programme zu integrieren.

Sie erhalten **Zugriff auf alle Eingänge und Ausgänge**, die Ihnen mit PiXtend (V1.2 & V1.3) zur Verfügung stehen.

In diesem Dokument wird erläutert, welche Schritte dafür notwendig sind. Außerdem erhalten Sie eine Übersicht zu den Möglichkeiten, die die C-Library bietet. Am Ende dieses Dokuments finden Sie zwei sauber dokumentierte Beispielprogramme die Sie testen oder als Grundlage für eigene Entwicklungen verwenden können.

Vielleicht haben Sie bereits die Programme ***pxauto*** und ***pixtendtool*** ausprobiert. Diese Programme wurden von Qube Solutions mit Hilfe der C-Library erstellt.





Viele weitere Informationen, Tipps und Tricks finden Sie auch in unserem Support-Forum unter: http://www.pixtend.de/forum/

Sollten trotzdem Fragen offenbleiben, so bitten wir Sie zuerst in den FAQ's im Anhang nachzuschauen. Wenn die Frage dort nicht geklärt wird können Sie uns gerne per E-Mail (support@pixtend.de) in Kenntnis setzen.

Die jeweils neusten Versionen aller Dokumente und Software-Komponenten finden Sie im Download-Bereich unserer Homepage: http://www.pixtend.de/downloads/

1.1 Voraussetzungen

Diese Anleitung setzt voraus, dass Sie die C-Library ***pxdev***, bereits installiert haben. Falls Sie dies noch nicht getan haben, können Sie die App-Note *pxdev* – Linux Tools für PiXtend installieren auf unserer Homepage im Download-Bereich herunterladen.

Die **einfachste Möglichkeit** ist der Einsatz unseres SD-Karten Image „**PiXtend Image Linux Tools**“ welches Sie ebenfalls **kostenlos auf unserer Homepage** herunterladen können. Auf diesem Image ist pxdev und die benötigten Tools bereits installiert.

Außerdem sollten Sie sich mit den Application-Notes *„Control- und Status Bytes“ & „Prozessdaten von PiXtend“* vertraut machen.
Grundkenntnisse in der C-Programmierung werden vorausgesetzt. Auch der Umgang mit unterschiedlichen Zahlensystemen (dezimal, binär, hexadezimal) und die Umrechnung zwischen diesen muss bekannt sein.

App-Note *„pxdev – Linux Tools für PiXtend installieren“*

http://www.pixtend.de/files/manuals/AppNote_pxdev_DE.pdf

App-Note *„Control-und Status Bytes“*

http://www.pixtend.de/files/manuals/AppNote_PiXtend_Control_Status_Bytes.pdf

App-Note *„Prozessdaten von PiXtend“*

http://www.pixtend.de/files/manuals/AppNote_PiXtend_Prozessdaten.pdf

Unter https://sourceforge.net/projects/pixtend/ finden Sie immer die neusten Quelltexte der PiXtend-Bibliothek. Wenn Sie unter Linux programmieren, so kann es hilfreich sein an einem PC die Quelltexte darzustellen oder diese auszudrucken.
Damit vermeiden Sie ein ständiges wechseln zwischen Ihrem Programmcode und den Quelltexten von *pxdev*.

Die C-Library *pxdev* und die in diesem Dokument gezeigten Schritte sind sowohl für PiXtend V1.2, als auch für PiXtend V1.3 gültig (soweit nicht explizit anders angegeben).

1.2 Haftungsausschluss

Weder Qube Solutions UG noch 3S-Smart Software Solutions können für etwaige Schäden verantwortlich gemacht werden die unter Umständen durch die Verwendung der zur Verfügung gestellten Software, Hardware, Treiber oder der hier beschriebenen Schritte entstehen können.

1.3 Sicherheitshinweise

PiXtend darf nicht in sicherheitskritischen Systemen eingesetzt werden.

Prüfen Sie vor der Verwendung die Eignung von Raspberry Pi und PiXtend für Ihre Anwendung.

2. Vorbereitung

Wie bereits das Titelbild dieser App-Note andeutet, gibt es zwei verschiedene Arten PiXtend mit Hilfe der C-Library anzusteuern:

- ***Manual Mode***
- ***Auto(matic) Mode***

Der *Manual Mode* ist für einzelne, manuelle Aufrufe gedacht. Bei jedem Funktionsaufruf wird über SPI (*Serial Peripheral Interface*) das entsprechende Gerät (PiXtend-Mikrocontroller oder Digital-Analog-Wandler – DAC) angesprochen und es wird nur eine kleine Datenmenge übertragen. Zum Beispiel wird der Zustand der digitalen Eingänge eingelesen oder die digitalen Ausgänge gesetzt. Der Modus ist sehr einfach zu implementieren, ist aber vergleichsweise langsam. Unser Beispielprogramm *pixtendtool* basiert auf dem *Manual Mode*.

Sollen viele oder gar alle Daten zwischen PiXtend und Raspberry Pi ausgetauscht werden, so bietet es sich an den *Auto Mode* zu verwenden. Damit ist es möglich alle Ein- und Ausgangswerte auf einmal auszutauschen. Die minimale Zykluszeit beträgt 25 ms (100 ms, wenn DHT-Sensoren verwendet werden sollen).
Der *Auto Mode* bietet noch weitere Vorteile, wie einen optionalen Watchdog-Timer und eine Überprüfung auf Übertragungsfehler (CRC-Prüfsumme). Unser Beispielprogramm *pxauto* verwendet den *Auto Mode*, wie der Name schon erahnen lässt.

Zunächst betrachten wir aber die Vorgehensweise, die für beide Arten (*Manual & Auto*) gleichermaßen zutrifft. Wir legen dabei Ordner und Dateien an. Im Kapitel *5. Kompilieren und Ausführen des Programms* machen wir aus unserem Code ein ausführbares Programm.

Wir arbeiten immer im Home-Verzeichnis des User „pi“. Damit vermeiden wir Probleme mit Lese- und Schreibberechtigungen, welche in anderen Ordnern auftreten können.

Nach dem Einloggen per SSH oder nach dem Booten direkt mit angeschlossener Tastatur & Maus am Raspberry Pi, landen Sie immer automatisch in diesem Verzeichnis. Falls Sie das Verzeichnis gewechselt haben sollten, so kehren Sie mit folgendem Befehl zurück:

```
cd /home/pi
```

2.1 Erstellen eines neuen Ordners und eines C-Files

```
pi@raspberrypi: ~ $ mkdir pixCExample
pi@raspberrypi: ~ $ cd pixCExample
```

Nun erstellen wir ein neues C-File *pixCExample* und editieren dieses mit dem Editor „*nano*“:

```
pi@raspberrypi: ~/pixCExample $ touch pixCExample.c
pi@raspberrypi: ~/pixCExample $ nano pixCExample.c
```

Der Editor nano startet und Sie sehen, dass die Datei *pixCExample* noch leer ist.

2.2 Einbinden der Bibliotheken

Nun können wir die benötigte PiXtend-Bibliothek (*pxdev*) in unser C-File einbinden:

```
#include <pixtend.h>
```

Außerdem sollten Sie folgende Standard-C-Bibliotheken einbinden:

```
#include <stdlib.h>
#include <stdio.h>
```

Nun kann mit dem eigentlichen Programmieren begonnen werden.

3. Programmierung

In diesem Abschnitt sollen die verschiedenen C-Funktionen, welche *pxdev* bietet, vorgestellt und genauer erläutert werden. Die Namensgebung der in dieser App-Note verwendeten Übergabeparameter unterscheiden sich teilweise vom Header-File der PiXtend Bibliothek. Dies dient ausschließlich zur besseren Übersichtlichkeit und Verständlichkeit.

Wir beginnen mit den Funktionen des *Manual Mode* und betrachten anschließend, ab Kapitel *3.2 Automatic Mode*, auch den *Auto Mode* für zyklische Datenübertragungen.

3.1 Manual-Mode

Betrachten wir zunächst beispielhaft einige Funktionen des Manual Mode.

3.1.1 SPI konfigurieren / aktivieren

Verwendbare Funktionen:

```
int Spi_Setup(int spi_device)
```

Um die Kommunikation zwischen PiXtend und dem Raspberry Pi über SPI zu starten, muss als Erstes die *Spi_Setup* Funktion mit dem jeweiligen *SPI-Device* aufgerufen werden. Zum Beispiel das SPI-Gerät 0:

```
Spi_Setup(0);
```

Das Gerät „0“ ist auf PiXtend der Mikrocontroller, welcher den Großteil der I/O-Funktionen verwaltet. Dieser wird in der Regel für jedes Programm benötigt.

Sollten auch die analogen Ausgänge verwendet werden, so muss diese Funktion ein zweites mal mit dem Wert „1“ aufgerufen werden.

```
Spi_Setup(1);
```

Es handelt bei *Spi_Setup* hier um eine Funktion, die nur einmal aufgerufen werden muss und auch nur einmal aufgerufen werden sollte (einmal mit Parameter „0“ und ggf. ein weiteres mal mit Parameter „1“). Es macht Sinn die Funktion in einer Initialisierungs-Routine beim Programmstart einmal auszuführen.

Achtung: Ein mehrfaches Aufrufen kann zum Fehler oder dem „Einfrieren“ der SPI-Verbindung führen.

3.1.2 Temperatur auslesen

Verwendbare Funktionen:

```
int Spi_Set_GpioControl(int gpio_mode)
uint16_t Spi_Get_Temp(int dht_id)
```

Es soll ein Temperatur-Sensor am PiXtend-*GPIO0* ausgelesen werden. Dazu rufen wir zunächst die *Gpio_Control* Funktion auf. Mit dieser kann auf das Control-Byte *GPIO_CTRL* zugegriffen werden.

```
Spi_Set_GpioControl(0x10);
```

Mit diesem Aufruf wird das Bit 5 in *GPIO_CTRL* gesetzt. Dieses steht für *DHT0* bzw. die Aktivierung des DHT-Modus für den PiXtend-*GPIO0.*

Statt direkt den Wert als Zahl als Parameter zu verwenden, können Sie natürlich auch eine Variable geeigneten Datentyps einsetzen.

Nun kann, der korrekten Anschluss des Sensors vorausgesetzt, die Temperatur von *DHT0* ausgelesen werden:

```
temp = Spi_Get_Temp(0);
```

Der Integer-Variable *temp* übergeben wir den Rückgabewert der Funktion.

Der Übergabeparameter steht für die jeweilige *DHT_ID*. Es kann ein Wert von 0 bis 3 übergeben werden. In diesem Fall wird eine Null übergeben. Diese steht für *DHT0*.

Um die Temperatur in Grad Celsius zu erhalten, können Sie folgende Zeile verwenden:

```
fTemperature = (float)(temp>>8)*1.0;
```

fTemperature ist vom Datentyp *float*. Mit (float) „casten“ wir die Integer Variable zu einer gleitkomma Variablen.

Mit folgendem Code können wir nun die aktuelle Temperatur mit zwei Nachkommastellen auf der Linux-Konsole ausgegeben werden:

```
printf("Temperatur: %2.f Grad Celsius\n",fTemperature);
```

In einem zyklischen Programm ist darauf zu achten, dass die Temperatur nicht häufiger als alle 2 Sekunden abgerufen werden sollte (gilt für den *Manual Mode*). Es macht keinen Sinn die Funktion "Spi_Get_Temp" schneller aufzurufen, weil die Sensoren nur alle 2 Sekunden einen neuen Wert liefern können. Das ist für eine Temperaturmessung auch in den allermeisten Fällen völlig ausreichend.

3.1.3 Relais schalten

Verwendbare Funktionen:

int Spi_Set_Relays(int relays)

uint8_t Spi_Get_Relays()

Als Nächstes sollen die vier Relais auf PiXtend geschaltet werden.

Hierzu rufen wir folgende Funktion auf:

```
Spi_Set_Relays(0x01);
```

In diesem Beispiel wird das *Relay0* aktiviert. Dies kann man am „klacken“ oder an der jeweiligen grünen LED erkennen.

Sollen alle Relais auf einmal geschaltet werden, kann man folgenden Übergabeparameter verwenden:

```
Spi_Set_Relays(0x0F);
```

Um die Relays wieder abzuschalten, wird dieselbe Funktion mit „0“ aufgerufen.

```
Spi_Set_Relays(0x00);
```

Soll überprüft werden welche Relais momentan durch-geschaltet sind, steht die Funktion *Get_Relays* zur Verfügung.

```
getRelays = Spi_Get_Relays();
```

Die Integer-Variable *getRelays* bekommt von der Funktion einen Rückgabewert „0“, wenn kein Relais aktiviert ist und bis zu „15“, wenn alle Relais aktiv geschaltet sind.

Achtung: Die Funktion *Spi_Get_Relays* steht bei der PiXtend Version 1.2 nicht zur Verfügung.

3.1.4 GPIOs konfigurieren und verwenden

Verwendbare Funktionen:

```
int Spi_Set_GpioControl(int gpioMode);
int Spi_Set_Gpio(int outputValue);
int Spi_Get_Gpio()
```

In diesem Schritt, sollen die vier PiXtend-GPIOs mit *Set_GpioControl(int gpioMode)* konfiguriert werden.

Um alle vier **GPIOs als Ausgänge** zu verwenden, muss auf das Control-Byte *GPIO_CTRL* zugegriffen werden. Dies machen wir mit der Code-Zeile:

```
Spi_Set_GpioControl(0x0F);
```

In *GPIO_CTRL* werden nun die ersten 4 Bits (LSBs) gesetzt.

Durch die Funktion *Set_Gpio(int outputValue),* können die Ausgänge auf einen Wert gesetzt werden.

```
Spi_Set_Gpio(0x0F);
```

Die als Ausgänge konfigurierten GPIOs werden so auf den Wert 1 gesetzt.

Sollen die **GPIOs als Eingänge** konfiguriert werden, muss man die Bits in *GPIO_CTRL* auf den Wert „0“ setzen.

```
Spi_Set_GpioControl(0x00);
```

Mit *Get_Gpio()* lassen sich die, als Eingänge konfigurierten GPIOs, auslesen.

```
getGPIO = Spi_Get_Gpio();
```

Die Funktion gibt den Wert, der als Eingänge konfigurierten GPIOs, als Dezimalwert zurück. Dieser Wert wird der Integer-Variablen *getGPIO* übergeben. Der maximale Wert, der erreicht werden kann, wenn an allen Eingänge eine „1“ anliegt, beträgt „15“ (0b00001111).

Bitte beachten Sie, dass die GPIOs „schwimmende Eingänge“ haben. Das bedeutet, dass sich der Eingangswert entweder „0“ und „1“ aufweisen kann, wenn extern kein definierter Pegel angelegt wurde.

3.1.5 Digitale Ein- & Ausgänge

Verwendbare Funktionen:

```
int Spi_Set_Dout(int value)
uint8_t Spi_Get_Dout()
int Spi_Get_Din()
```

Um die **digitalen Ausgänge** auf PiXtend auf den Wert „1“ zu schalten, wird der nachfolgende Code verwendet:

```
Spi_Set_Dout(0xFF);
```

Da PiXtend sechs digitale Ausgänge besitzt, würde eigentlich der Wert 0x3F (0b00111111) ausreichen, um alle Ausgänge zu setzen. Da die letzten 2 Bits aber keine Verwendung haben, kann auch 0xFF (0b11111111) verwendet werden.

Es besteht auch die Möglichkeit, die digitalen Ausgänge auf ihren Wert zu überprüfen.

Dazu kann diese Funktion genutzt werden:

```
digitalOutputs = Spi_Get_Dout();
```

Diese gibt einen dezimalen Wert von „0“ bis „63“ zurück. Je nachdem, wie viele digitale Outputs auf den Wert 1 gesetzt sind.

Ein Beispiel:
Erhalten Sie einen Rückgabewert von „32“ dezimal, so entspricht das dem folgenden binären Wert: 0b00100000. Im binären System ist hier nun leicht zu erkennen, dass der sechste digitale Ausgang aktiv und alle anderen inaktiv sind. Bitte beachten Sie, dass Ein- und Ausgänge auf PiXtend immer ab „0“ gezählt werden. Der sechste digitale Ausgang entspricht also DO5 (Beschriftung auf PiXtend-Leiterplatte / Edelstahlhaube).

Achtung: Die Funktion Spi_Get_Dout() steht bei der PiXtend Version 1.2 nicht zur Verfügung.

Für das Auslesen der acht **digitalen Eingänge**, gibt es ebenfalls eine Funktion:

```
digitalInput = Spi_Get_Din();
```

Diese Funktion gibt den Wert der acht Eingänge als Dezimalwert zurück.

Wenn an allen Eingängen eine „1“ anliegt, so gibt die Funktion den Wert „255“ (0b11111111) zurück.

3.1.6 Analoge Ein- & Ausgänge

Verwendbare Funktionen:

int Spi_Set_Aout(int channel, uint16_t value)

uint16_t Spi_Get_Ain(int channel)

int Spi_Set_AiControl(ai_cntrl0, ai_cntrl1)

Um die **analogen Ausgänge** anzusprechen, wird diese Funktion aufgerufen:

```
Spi_Set_Aout(1,1023);
```

Die Funktion verlangt als ersten Eingangsparameter den Kanal (0 oder 1) und einen Wert zwischen 0...1023. Die 1023 kommt durch die 10 bit Auflösung des Digital-Analog Wandlers zustande: 2^{10} - 1 = 1023.
Der Wert „0“ entspricht einem Spannungswert am Ausgang von 0 V, „1023“ entspricht dem maximalen Spannungswert von 10 V. Falls Sie einen abweichenden Wert am Ausgang messen sollten, so trimmen Sie die Ausgangsspannung mit Hilfe der beiden Potentiometer auf den gewünschten Wert.

Achtung: Für die analogen Ausgänge muss zuvor die Funktion Spi_Setup(1); aufgerufen werden, falls dies noch nicht geschehen ist.

Die **analogen Eingänge** können mit folgender Zeile ausgelesen werden:

```
analogIn=Spi_Get_Ain(0);
```

Die Funktion verlangt den analogen Kanal (0 - 3), an dem das Eingangssignal anliegt und gibt einen Dezimalwert 0...1023 zurück. Dieser Wert wird in diesem Beispiel der Integer Variablen *analogIn* zugewiesen.

Mit den Control-Bytes *AI_CTRL0* und *AI_CTRL1* können die vier analogen Eingänge konfiguriert werden. Für die Konfiguration rufen wir eine separate Funktion auf:

```
Spi_Set_AiControl(0xFF,0x60);
```

Der erste Parameter steht für *AI_CTRL0*. In diesem Beispiel werden alle analogen Eingänge auf eine Mittelwertbildung über 50 Messungen konfiguriert. Der zweite Parameter steht für *AI_CTRL1* und wird auf einen Vorteiler von 8 (2 MHz) eingestellt.

Über die genauen Auswirkungen und Möglichkeiten der Konfiguration können Sie sich im entsprechenden Dokument Control- & Status-Bytes genauer informieren.

3.1.7 PWM- & Servo-Modus

PWM-Modus

Verwendbare Funktionen:

int Spi_Set_PwmControl(int pwm_ctrl0, int pwm_ctrl1, int pwm_ctrl2)

int Spi_Set_Pwm(int channel, uint16_t value)

PWM0 und *PWM1*, sind für das Erzeugen eines PWM-Signals geeignet. Zuerst einmal muss auf den PWM-Modus umgeschaltet werden.

```
Spi_Set_PwmControl(0x61,0xE8,0x03);
```

Die Funktion verlangt als ersten Parameter das erste PWM-Control-Byte *PWM_CTRL0*.

Hier wird das erste Bit, für den PWM-Mode auf „1“ geschaltet. Bit 6 (*CS0*), Bit 7 (*CS1*) und Bit 8 (*CS2*) stehen für den verwendeten *Prescaler* (deutsch: Vorteiler). In diesem Beispiel wird ein Vorteiler von 64 eingestellt.

Der zweite Parameter steht für *PWM_CTRL1* und der dritte Parameter für *PWM_CTRL2*. Diese sind für die PWM Frequenz/Periodendauer verantwortlich. Jeder Parameter erwartet ein Byte als Übergabewert.
In diesem Beispiel wird also PWM_CTRL1..2 auf dezimal „1000“ gesetzt.

Nachdem *PWM0* und *PWM1* konfiguriert wurden, müssen diese noch gesetzt werden. Dies übernimmt folgende Funktion für uns:

```
Spi_Set_Pwm(0,1000);
Spi_Set_Pwm(1,500);
```

Mit der ersten Funktion wird *PWM0* auf einen Duty Cycle (deutsch: Tastverhältnis) von 100% eingestellt. Bei der zweiten Funktion wird *PWM1* auf einen 50% Duty Cycle eingestellt. Der erste Übergabewert steht für den jeweiligen Kanal *PWM0* oder *PWM1*. Der zweite Parameter ist eine 2 Byte große Integer-Variable und steht für den Duty-Cycle. Der Wert kann im Bereich von 0 bis 65535 liegen (16 bit Wert - WORD).
In unserem Beispiel macht es jedoch keine Sinn einen größeren Wert als 1000 (dez) zu übergeben, da hier bereits der maximale Duty Cycle von 100% erreicht ist.

Servo-Modus

Verwendbare Funktionen:

```
int Spi_Set_PwmControl(int pwm_ctrl0, int pwm_ctrl1, int pwm_ctrl2)
int Spi_Set_Servo(int channel, int value)
```

Der Servo-Modus ist standardmäßig (nach dem Start von PiXtend) aktiv. Es muss also keine Konfiguration erfolgen, außer die PWM-Ausgänge wurden zuvor aktiv in den PWM-Modus umgestellt. In diesem Fall können die Ausgänge mit folgendem Befehl auf den Servo-Modbus zurück-konfiguriert werden:

```
Spi_Set_PwmControl(0,0,0);
```

Für den Servo-Modus, muss das Mode Bit auf den Wert „0“ gesetzt werden. Die restlichen Bits *CS0*, *CS1* und *CS2* spielen hier keine Rolle. Daher schreiben wir der Einfachheit halber alle drei Parameter auf „0“.

Um den Ausschlag eines Servos einzustellen, wird folgende Funktion verwendet:

```
Spi_Set_Servo(1,255);
```

Der erste Übergabeparameter der Funktion bestimmt den jeweiligen Kanal, *PWM0* (0) oder *PWM1* (1). Der zweite Wert ist eine 1 Byte große Integer-Variable bzw. Zahl. Diese steht für die Stellung des Servos. In diesem Beispiel haben wir den Servo 1 (PWM1) auf den maximalen Wert 255 eingestellt. Wir erhalten also den maximalen Ausschlag des Servos. Der minimale Wert und damit minimale Ausschlag ist hier „0“.

Achtung: *PWM0* und *PWM1* können entweder beide im normalen PWM- oder beide im Servo-Modus betrieben werden. Eine „Mischung“ ist nicht vorgesehen.

Hiermit schließen wir die Erläuterungen zum *Manual Mode* ab. Es gibt noch weitere Funktion und weitere Werte die abgefragt oder geschrieben werden können. Wenn Sie diese verwenden möchten, dann empfehlen wir Ihnen einen Blick in Kapitel *1.1 Voraussetzungen* angegebenen App-Notes oder in die Quelltexte der *pxdev* C-Library.

3.2 Automatic Mode

Die *SPI Auto Mode Funktion* erlaubt die zyklische und schnelle Abarbeitung der verschiedenen PiXtend-C-Funktionen. Dabei wird mit Hilfe der zyklischen Redundanzprüfung *(cyclinc redundancy check CRC)* die Übertragung per SPI auf Fehler überprüft. Des Weiteren bietet der Einsatz der Funktion den Vorteil, dass der Watchdog des PiXtend-Mikrocontrollers verwendet werden kann. Bei einem Softwarefehler verhindert dieser einen undefinierten Zustand des Systems. Die Software teilt dem Watchdog in regelmäßigen Abständen mit, dass sie ordnungsgemäß läuft. Meldet sich das laufende Programm nach einer gewissen Zeit nicht zurück, löst der Watchdog einen Reset des Mikrocontrollers aus und das Programm startet neu (definierter Zustand).

Um die Zykluszeit einzustellen und die dauerhafte Übertragung zwischen Raspberry Pi und PiXtend zu gewährleisten, muss beim *Auto Mode* eine Endlosschleife („infinity loop“) implementiert werden. Die Schleife wird abhängig von einer Bedingung oder nach einer bestimmten Zeit erneut aufgerufen. Um die Zykluszeit einzustellen kann beispielsweise eine Verzögerung von 100 ms eingebaut werden. Die kleinste Zykluszeit die ohne Verwendung der DHT Sensoren eingestellt werden darf, ist 25 ms. Sollten jedoch DHT-Sensoren zum Einsatz kommen, muss die Zykluszeit 100 ms oder größer sein, um mögliche Übertragungsfehler auszuschließen.

Vewendbare Funktionen:

int Spi_AutoMode(struct OutputData, struct InputData)

int Spi_AutoModeDAC(struct OutputDataDAC)

Voraussetzungen:

Die Funktion *Spi_Setup(0)* muss für die Kommunikation mit dem Mikrocontroller aufgerufen werden.

Die Funktion *Spi_Setup(1)* muss für die Kommunikation mit dem DAC (für analoge Ausgänge) aufgerufen werden.

Das *Run* Bit des Mikrocontrollers in *Uc_Ctrl* muss auf den Wert „1“ gesetzt sein:

```
OutputData.byUcCtrl = 0x10;
```

Soll zusätzlich auch der Watchdog-Timer verwendet werden, muss die Funktion wie folgt aufgerufen werden:

```
OutputData.byUcCtrl = 0x11;
```

Mit der Funktion *Spi_AutoMode(struct OutputData, struct InputData)* wird ein kompletter Datenaustausch zwischen PiXtend und Raspberry Pi initiiert. Um die Funktion nutzen zu können muss jeweils eine neue Struktur vom Typ *pixtIn* und *pixtOut* angelegt werden.

```
struct pixtIn InputData;
struct pixtOut OutputData;
```

Folgende Variablen stehen in der Struktur zur Verfügung. Die zu übergebenden Daten entsprechen den Daten im *Manual Mode*. Für weiter Infos zu den einzelnen Bits und Bytes verweisen wir auch an dieser Stelle auf die App-Notes *Control- und Status Bytes* und *Prozessdaten von PiXtend*.

```
OutputData.byDigOut;        //Ein Byte für Setzen der digitalen Ausgänge
OutputData.byRelayOut;      //Ein Byte für Setzen der Relais
OutputData.byGpioOut;       //Ein Byte für Setzen der GPIO Outputs
OutputData.wPwm0;           //Zwei Bytes für PWM0 Value
OutputData.wPwm1;           //Zwei Bytes für PMW1 Value
OutputData.byPwm0Ctrl0;     //Erstes Byte für PWM Control
OutputData.byPwm0Ctrl1;     //Zweites Byte für PWM Control
OutputData.byPwm0Ctrl2;     //Drittes Byte für PWM Control
OutputData.byGpioCtrl;      //Ein Byte für GPIO Control
OutputData.byUcCtrl;        //Ein Byte für UC Control
OutputData.byAiCtrl0;       //Ein Byte für Analog In Control 0
OutputData.byAiCtrl1;       //Ein Byte für Analog In Control 1
OutputData.byAux0;          //Ein Byte für Jumperstellung Analog In 0,1
```

Mit *OutputData.byAux0* kann die Jumperstellung der analogen Eingänge AI0 und AI1 auf 5 V oder 10 V gesetzt werden. Bei kleineren Spannungen als 5 V macht es Sinn die 5 V Jumperstellung zu verwenden, da so mit einer höheren Auflösung gemessen werden kann.

Möchte man den AI0 auf 10 V setzen, muss die Variable wie folgt befüllt werden:

```
OutputData.Aux0 = 0x01;
```

Soll AI1 auf eine Jumperstellung von 10 V gesetzt werden, verwendet man:

```
OutputData.Aux0 = 0x02;
```

Sollen beide analogen Eingänge auf 10 V gesetzt werden:

```
OutputData.Aux0 = 0x03;
```

Für die Jumperstellung 5 V, müssen die Bits nicht gesetzt bzw. mit dem Wert „0“ beschrieben werden.

Die **Eingangsdaten von PiXtend** können folgenden Variablen entnommen werden:

```
InputData.byDigIn;          //Ein Byte für die digitalen Eingänge
InputData.byGpioIn          //Ein Byte für die GPIO Eingänge
InputData.wAi0;             //Zwei Byte für analog Eingang 0
InputData.wAi1;             //Zwei Byte für analog Eingang 1
InputData.wAi2;             //Zwei Byte für analog Eingang 2
InputData.wAi3;             //Zwei Byte für analog Eingang 3
InputData.wTemp0            //Zwei Byte für DHT0 Wert Temperatur
InputData.wTemp1            //Zwei Byte für DHT1 Wert Temperatur
InputData.wTemp2            //Zwei Byte für DHT2 Wert Temperatur
InputData.wTemp3            //Zwei Byte für DHT3 Wert Temperatur
InputData.wHumid0           //Zwei Byte für DHT0 Wert Luftfeuchte
InputData.wHumid1           //Zwei Byte für DHT1 Wert Luftfeuchte
InputData.wHumid2           //Zwei Byte für DHT2 Wert Luftfeuchte
InputData.wHumid3           //Zwei Byte für DHT3 Wert Luftfeuchte
InputData.byUcVersionL;     //Ein Byte für Low Byte Mikrocontroller Version
InputData.byUcVersionH;     //Ein Byte für High Byte Mikrocontroller Version
InputData.byUcStatus;       //Ein Byte für RUN, Stop Status PiXtend Mikrocontroller
InputData.rAi0;             //float Variable für Spannung analog Eingang 0
InputData.rAi1;             //float Variable für Spannung analog Eingang 1
InputData.rAi2;             //float Variable für Spannung analog Eingang 2
InputData.rAi3;             //float Variable für Spannung analog Eingang 3
InputData.rTemp0;           //float Variable für Temperatur DHT22 GPIO0
InputData.rTemp1;           //float Variable für Temperatur DHT22 GPIO1
InputData.rTemp2;           //float Variable für Temperatur DHT22 GPIO2
InputData.rTemp3;           //float Variable für Temperatur DHT22 GPIO3
InputData.rHumid0;          //float Variable für Luftfeuchte DHT22 GPIO0
InputData.rHumid1;          //float Variable für Luftfeuchte DHT22 GPIO1
InputData.rHumid2;          //float Variable für Luftfeuchte DHT22 GPIO2
InputData.rHumid3;          //float Variable für Luftfeuchte DHT22 GPIO3
```

Achtung: Die Temperatur- und Luftfeuchteumrechnungen in *rTemp* und *rHumid*, gelten ausschließlich für DHT22 Sensoren. Bei Bedarf muss die Umrechnung der Werte im Programm selbst implementiert werden. Die Umrechnung für DHT11 Sensoren wird im nächsten Update hinzugefügt.

Um die beiden Strukturen der Funktion zu übergeben, wird folgende Zeile verwendet:

```
Spi_AutoMode(&OutputData,&InputData);
```

Die Struktur *OutputData* enthält dabei die Werte für die Control-Bytes und Prozessdaten (Ausgangswerte aus Sicht des Raspberry Pi / PiXtend).

In die Struktur *InputData* werden die Werte der Eingänge während des Programmablaufs geschrieben (vom PiXtend-Mikrocontroller).

Ein kleines Beispiel:
Mit folgendem Code kann die Temperatur über *DHT0* ausgelesen und auf die Konsole ausgegeben werden:

```
dht0Temp=InputData.rTemp0;
printf("Temperatur C: %.2f\n", dht0Temp);
```

Die Funktion *Spi_AutoModeDAC(struct OutputDataDAC)* wird ausschließlich für die Kommunikation mit dem *Digital-Analog-Converter (DAC)* verwendet. Die Funktion verlangt als Übergabeparameter eine Struktur vom Typ *pixtOut*. Diese kann folgendermaßen erstellt werden:

```
struct pixtOutDAC OutputDataDAC
```

Die Struktur muss die Werte für die beiden analogen Ausgänge beinhalten.

```
OutputDataDAC.wAOut0 = 1023; //1023 entspricht dem Maximalwert, 0 Minimalwert
OutputDataDAC.wAOut1 = 1023;
```

Hier werden die analogen Ausgänge auf 100% gesetzt. Mit dem folgenden Aufruf übergeben wir die Struktur *OutputDataDAC* der Funktion.

```
Spi_AutoModeDAC(&OutputDataDAC);
```

4. Beispielprogramme

4.1 Manual Mode

```
//Include the librarys
#include <stdio.h>
#include <pixtend.h>

//Define some variables
int digitalInput = 0;
uint16_t temp = 0;
uint16_t analogInput = 0;
float fTemperature = 0;

int main(void)
{

	printf("Hello World! Hello PiXtend!\n");

	/*SPI CONFIGURATION*/
	//For microcontoller
	Spi_Setup(0);
	//For DAC
	Spi_Setup(1);

	/*CONFIGURATION FOR DHT22 ON GPIO1/DHT1*/
	//Set Bit 5 GPIO_CTRL -> DHT1
	Spi_Set_GpioControl(0x20);

	//Read Temperature -> DHT1
	temp = Spi_Get_Temp(1);

	//Temperature in Celsius
	fTemperature = (float)temp/10.0;
	printf("Temperature [°C]: %.2f\n", fTemperature);

	/*Digital Inputs*/
	//Read Digital Inputs, Value: 0..255
	digitalInput = Spi_Get_Din();
	printf("Digital Inputs [RAW]: %d\n", digitalInput);

	//Set all Relays if at least one input is High
	if(digitalInput > 0)
	{
		printf("Input detected - setting Relays!\n");
		/*RELAYS*/
		//Set Relays 0..3
		Spi_Set_Relays(15);
	}
```

```
    else
    {
        Spi_Set_Relays(0);
    }

    /*Analog Outputs*/
    //Channel 1, Value 1023 (10V)
    Spi_Set_Aout(0,1023);
    //Channel 2, Value 512 (5V)
    Spi_Set_Aout(1,512);

    /*Analog Inputs*/
    //Read Analog Input 0
    analogInput = Spi_Get_Ain(0);
    printf("Analog Input [RAW]: %d\n", analogInput);

    return(0);
}
```

4.2 Automatic Mode

```
//Include the librarys
#include <pixtend.h>
#include <stdio.h>

//Function prototype
int GetPixData();

//Define some variables
float dht1Temp=0.0;
int16_t dht1Hum=0;

//PiXtend input data
struct pixtIn InputData;

//PiXtend output data
struct pixtOut OutputData;

//PiXtend DAC Output Data
struct pixtOutDAC OutputDataDAC;

//Read Temperature and Humidity from DHT1 (DHT22)
int GetPixData()
{
	dht1Temp=InputData.rTemp1;
	dht1Hum=InputData.wHumid1;

	//Only print out valid data
	if(dht1Hum>0)
	{
		//Write out the data on the linux console
		printf("Temperature [°C]: %.2f\n", dht1Temp);
		printf("Humidity [%%]: %.2f\n", (float)dht1Hum/10);
	}
	return 0;
}

int main(void)
{

	//Config SPI
	Spi_Setup(0);
	Spi_Setup(1);

	//Write Data in Structure OutputData
	OutputData.byUcCtrl = 16; //Set the PiXtend-Controller to RUN mode
	OutputData.byGpioCtrl = 0x20; //GPIO1 is used for DHT1 (DHT22)
	OutputDataDAC.wAOut0 = 1023; //pre-load analog outputs
	OutputDataDAC.wAOut1 = 1023;
```

```
    //Auto Mode in infinity loop
    while(1)
    {
        //Do the data transfer!
        Spi_AutoMode(&OutputData,&InputData);
        Spi_AutoModeDAC(&OutputDataDAC);

        //Set all Relays if at least one digital input is High
        if(InputData.byDigIn>0)
        {
            printf("Input detected - setting Relays!\n");
            OutputData.byRelayOut = 15;
        }
        else
        {
            OutputData.byRelayOut = 0;
        }

        //Toggle Analog Outputs
        if(OutputDataDAC.wAOut0 == 1023 || OutputDataDAC.wAOut1 == 1023)
        {
            OutputDataDAC.wAOut0 = 512;
            OutputDataDAC.wAOut1 = 512;
        }
        else
        {
            OutputDataDAC.wAOut0 = 1023;
            OutputDataDAC.wAOut1 = 1023;
        }

        //Read data and print it out
        GetPixData();

        //Take a 250 ms break between the cycles (min. 100 ms)
        delay(250);
    }
    return 0;
}
```

5. Kompilieren und Ausführen des Programms

Um das Programm ausführen zu können, muss es zunächst kompiliert werden. Für das Kompilieren muss folgende Zeile eingegeben werden:

```
pi@raspberrypi ~ $ sudo gcc -Wall -o "pixCExample" "pixCExample.c" -lpixtend -lwiringPi
```

Nun wird *pixCExample.c* kompiliert und in *pixCExample* gespeichert.

Mit dem Befehl:

```
pi@raspberrypi ~ $ sudo ./pixCExample
```

wird das Programm ausgeführt.

Wer *sudo* nicht verwenden möchte, kann der kompilierten Datei *pixCExample* auch die benötigten Rechte geben. Dazu kann beispielsweise folgende Zeile verwendet werden:

```
pi@raspberrypi ~ $ sudo chmod -R 0777 pixCExample
```

Nun kann *sudo*[1] beim Ausführen des C-Programms weggelassen werden.

Für das Beenden eines Programms können Sie einfach die Tastenkombination STRG → C verwenden.

1 *sudo* wird hier überhaupt erst notwendig, weil die SPI-Schnittstelle über die wiringPi-Library (http://wiringpi.com/) angesprochen wird. Dieser Zugriff erfordert nach wie vor Superuser-Rechte.

6. Frequently Asked Questions (FAQ)

Meine Werte vom DHT-Sensor stimmen nicht. Was kann ich tun?
(Antwort gilt nur für den *Manual Mode*)

Bei den DHT Sensoren muss darauf geachtet werden, dass diese nicht zu schnell hintereinander im Programm aufgerufen werden. Wird die Funktion zu oft aufgerufen, kann es zu Übertragungsfehlern der SPI-Schnittstelle kommen. Dies führt zu falschen Messwerten. Am Besten ist es eine Delay Funktion von 200 – 300 ms in das Programm einzubauen. Es könnte auch überprüft werden, ob die Funktion beendet ist und bereit für eine erneute Übertragung ist. Dadurch sollten falsche Messwerte vermieden werden.
Im *Auto Mode* werden bisher nur DHT22 Sensoren unterstützt (Stand: Oktober 2016).

Was muss ich besonders beachten, wenn ich die PiXtend C-Library verwende?

Egal ob im *Manual Mode* oder *Automatic Mode* muss immer als Erstes die SPI-Schnittstelle, mit der Funktion *Spi_Setup(int device)*, konfiguriert werden. Die *Spi_Setup* Funktion darf nur maximal zwei mal (einmal für device 0, einmal für device 1) aufgerufen werden. Daher sollte die Aufrufe in eine Initialisierungsroutine ausgelagert werden, die nur einmal ausgeführt wird (direkt nach dem Start des Programms).

Im Automatic Modus muss zusätzlich das Run-Bit gesetzt werden. Mit der Zuweisung *OutputData.byUcCtrl = 16;* wird dieses Bit gesetzt. Auch diese Funktion muss nur einmal am Anfang des Programms aufgerufen werden, ein zyklischer Aufruf ist nicht notwendig.

Was sind die Vorteile des *Automatic Mode*?

Im *Auto Mode* haben Sie die Möglichkeit die Daten von PiXtend zyklisch abzufragen und zu setzen. Sie brauchen sich hier keine Gedanken über Übertragungszeiten zu machen. Dies erledigt die *Auto Mode* Funktion von selbst. Lediglich die Zykluszeit muss eingestellt werden. Ein CRC-Check ist im *Automatic Mode* enthalten. Dieser prüft automatisch auf mögliche Übertragungsfehler. Ist der Watchdog aktiviert, kann dieser verhindern dass der Mikrocontroller sich in einem undefinierten Zustand befindet. Meldet sich der Controller nach einer bestimmten Zeit nicht zurück, wird ein Reset ausgelöst und das Programm erneut gestartet.

Es kommt immer wieder zu Übertragungsproblemen zwischen PiXtend und Raspberry Pi. Was kann ich tun?

Überprüfen Sie in ihrem Programm, ob Sie die Funktion Spi_Setup(int device) mehrfach oder zyklisch aufrufen. Ist dies der Fall, so ist dies der Grund für die Übertragungsfehler. Sollte gar keine Übertragung statt finden, überprüfen Sie auch, ob der Jumper *SPI_EN* auf dem PiXtend gesteckt ist. Sollten DHT-Sensoren verwendet werden, muss die Zykluszeit größer 100 ms / 10 Hz sein. So werden Messfehler vermieden. Allgemein sollte die Zykluszeit nicht weniger als 25 ms / 40 Hz betragen.