Hallo allerseits,

hier ist eine vereinfachte, aber universell nutzbare Variante meiner DCF-Uhr entstanden.
Ausgangspunkt war, dass ich für einen Datenlogger eine unkomplizierte Zeitbasis ohne aufwändige Kalibrierung benötigte.

Es bot sich an, die "alte" DCF-Uhr für diesen Zweck umzubauen, überflüssige Funktionen zu entfernen und alles in ein einziges Modul mit eigenem Variablensatz und geregeltem Zugriff von aussen zu verpacken.
Das Ergebnis ist ein Modul, dass lediglich in ein Projekt eingebunden werden muss und auf das über nur einige wenige Funktionen zugegriffen wird.

Die Hardware muss die Uhr sich natürlich mit dem übrigen Projekt teilen:
Benötigt werden zwei Timer und sowie ein interruptfähiger Eingangspin für das DCF-Signal.
Timer0 ist fest belegt. Als zweiter Timer kann zwischen Timer1 und Timer2 gewählt werden
(#define TIMER [TIMER1 | TIMER2]).

~~Das DCF-Signal wird an INT0 und INT1 empfangen (ja, die sind miteinander verbunden !), es ist aber keine große Kunst, das Programm auch mit einem einzigen Pin-Interrupt laufen zu lassen.~~

Das DCF-Signal wird per default am Pin INT0 angeschlossen.
Alternativ kann jeder andere Pin benutzt werden, der einen Pin-Change-Interrupt auslösen kann.
Anzupassen sind dabei die Freigabe des Interrupts, der Interrupt muss definiert sein als "Any logical change on XY generates an interrupt request" und der Aufruf der ISR(INT0_vect) muss an den gewählten Pin_Change angepasst werden.

Zur Kontrolle der Uhr ist eine Kontroll-LED nützlich:
- während des Wartens auf den ersten Minutenbeginn zeigt die LED Dauerlicht.
- Nach der Synchronisation bis zum Erkennen der DCF-Zeit blinkt die LED im Sekundentakt.
- Danach verlischt sie.

Sobald ein Fehler im DCF-Signal registriert wird, zeigt die LED wieder Dauerlicht.
Zum Beginn einer neuen Minute wird die LED wieder ausgeschaltet.
Die Uhr selbst läuft dabei natürlich ungestört (und quarzgesteuert) weiter.

Das Hauptprogramm muss kooperativ sein, darf nicht in Warteschleifen verharren und muss zeitnahe Abfragen auf den Beginn der nächsten Sekunde zulassen.
Wird eine neue Sekunde/Minute erkannt (DCF_New_Second() == TRUE), dann muss innerhalb des Moduls die Zeit fortgeschrieben und mit der DCF-Zeit synchronisiert werden (das macht DCF_Check_Time()).

Die Funktion DCF_New_Second() wurde überarbeitet und liefert am Minutenbeginn nur dann TRUE zurück, wenn die Zeit sich tatsächlich ändern wird - bisher konnten zwei Aufrufe am Minutenbeginn direkt aufeinander folgen.
(Anmerkung: zu Beginn einer neuen Minute melden die XTL-Uhr und auch die DCF-Uhr (sofern sie aktiv ist) einen Minutenbeginn. Das Flag DCF_valid bestimmt, wer von beiden die Zeit synchronisiert.
DCF_New_Second() liefert ein FALSE zurück, wenn die "falsche" Uhr den Minutenbeginn anzeigt.)

In main.c ist dargestellt, wie das Modul benutzt werden kann.

Die implementierten Funktion (siehe DCF_77_mini.h):

```
void     DCF_Init(void);                 // wird nur 1x aufgerufen beim Programmstart
uint8_t  DCF_New_Second(void);           // Liefert TRUE, wenn eine neue Sekunde angebrochen ist
void     DCF_Check_Time(void);           // muss immer aufgerufen werden,
                                         // wenn DCF77_New_Second() == TRUE
uint8_t  DCF_Get_Time(uint8_t idx);      // Liefert die angefragte Zeitkomponente als Wert zurück
void     DCF_Get_TimeStr(uint8_t idx, char * s); // Liefert die angefragte Zeitkomponente als String zurück
uint8_t  DCF_Get_Errors(void);           // Liefert die Anzahl der DCF-Fehler seit Programmstart/
                                         // letztem Aufruf zurück
```

Die Funktion DCF_Get_Errors() liefert die Anzahl der Minuten, in denen Fehler im DCF-Signal erkannt wurden. Bei jedem Aufruf der Funktion wird der Zähler zurückgesetzt.
Die Konstanten für die Zeitkomponenten stehen in "DCF_77_mini.h".

Der Speicherbedarf des Moduls liegt bei etwa 2600 Byte.

Ergänzung vom 10.10.11

Bisher habe ich die DCF-Empfänger immer erfolgreich mit 5V oder 3.3V betrieben. Dabei war das DCF-Signal zeitlich stabil und die Signalflanken waren sauber ohne Nachschwingen.

In einem aktuellen Projekt habe ich nun versucht, mit geringeren Versorgungspannungen zu arbeiten um ggf. einen Batteriebetrieb zu ermöglichen (laut Datenblatt arbeitet der Empfänger ab 1.2 V).
Bei dieser Betriebsspannung reduziert sich die Stromaufnahme tatsächlich auf 70uA.
Allerdings wird das Signal extrem launisch, hat sehr unterschiedliche Pulslängen und schwingt heftig nach.
Der aktuelle Kompromiss ist der Betrieb mit ca. 2.1 Volt, die Pulsdauern sind dabei recht stabil, die Stromaufnahme liegt bei 300uA.
Allerdings schwingt das Signal an der steigenden Flanke trotzalledem nach und löst gelegentlich einen zusätzlichen Interrupt aus, der zum Einlesen zusätzlicher DCF-Bits und damit zu Fehlern bei der Decodierung der Zeit führte.

Um diese Probleme zu erkennen, einzugrenzen und zu beheben ist die Funktion DCF_Get_Timings() eingebaut, die nachfolgende Informationen aus dem DCF-Modul abholt:
- aktuelle Uhrzeit in Minuten und Sekunden (damit der Synchronisationsfortschritt verfolgt werden kann),
- Anzahl der empfangenen DCF-Bits,
- gemessene Länge der Pulsdauer im 10ms,
- gemessene Länge der Taktdauer in 10ms,
- vom Programm generierter Errorcode der laufenden Übertragung,
- Hinweis, ob das Programm das DCF-Signal als gültig interpretiert hat (DCF_valid).

Wenn man diese Daten interpretiert, dann ist schnell erkennbar, ob Empfangsprobleme vorliegen:

1.) Die gemessene Länge von Puls und Takt muss mit den programminternen Voreinstellungen verglichen werden. Ggf. müssen die Vorgaben den Eigenschaften des DCF-Empfängers angepasst werden.
Wenn die Versorgungsspannung des Empfängers verändert wird, kann sich das Timing des Signals auch deutlich verändern.
2.) Immer dann, wenn die Anzahl der DCF-Bits <> der Sekundenzahl ist (Ausnahme: Sekunde 0), dann liegt ein Fehler im Empfang vor:
Ist die Zahl der Bits kleiner, dann passen vermutlich Puls- oder Taktlänge nicht, so dass DCF-Bits überlesen werden.
Ist die Anzahl größer, dann sind einzelne Bits mehrfach empfangen worden.
Ursache ist das Nachschwingen des Signal, dass kurz nacheinander (aber trotzdem im Rahmen der vorgegebenen Toleranzen) zusätzliche Interrupts auslöst.

Letzteres Problem ist softwaremäßig gelöst:
sobald eine erste steigende Flanke erkannt wurde, wird erst nach einer fallenden Flanke - die im Zeitfenster von 100/200ms liegen muss - die nächste Flanke als Abschluss eines Bits registriert.

Ein Tiefpass vor dem INT-Pin ist in jedem Fall empfehlenswert.

10.10.11

Michael S.