Laborversuch 3

Thema:

Funkgesteuerte Uhr

Inhaltsverzeichnis

Zusätzlich erforderliche Unterlagen:

- Vorlesungsmanuskript zu Kapitel 1 bis 4 mit Anhang Metroworks CodeWarrior HCS12 Entwicklungsumgebung
- Dokumentationspaket für den HCS12 Mikrocontroller und das Dragon12-Entwicklungsboard
- Ergebnisse des Laborversuchs 2: Interrupt- und Ein-/Ausgabeprogrammierung

1. Überblick

Ziel des Laborversuchs

- Implementierung einer komplexeren Applikation
- Gemischte Programmierung in C und Assembler

In der zweiten Laborübung ist eine Uhr programmiert worden. Die Uhr bestand aus einem Timer, der periodisch im 10ms Raster einen Interrupt erzeugte, aus dem dann über einen Softwarezähler der 1sec-Uhrentakt gewonnen und die Uhrzeit abgeleitet wurde. Leider vergisst diese Uhr beim Abschalten der Spannungsversorgung jedes Mal die Zeit und muss beim nächsten Einschalten mühsam wieder eingestellt werden. Außerdem kommt es bei längerem Betrieb der Uhr wahrscheinlich zu zunehmenden Zeitabweichungen, da der Sekundentakt durch kleine Toleranzen des Quarzgenerators auf dem Dragon12-Entwicklungsboards nicht exakt stimmt.

Das Einstellen der Uhr und das laufende Nachstellen soll nun automatisiert werden. Dazu gibt es in Deutschland einen Langwellensender, der in einem bestimmten Format Zeitzeichen aussendet. Dieses Format nennt man DCF77. Es enthält neben der Uhrzeit und der Angabe, ob es sich bei der Zeit um die normale Mitteleuropäische Zeit (MEZ) oder die Sommerzeit (MESZ) handelt, auch eine Information über das aktuelle Datum, das damit ebenfalls auf dem LCD-Display dargestellt werden kann.

Ablauf des Laborversuchs

Der Laborversuch besteht aus einem Vorbereitungsteil und einem Teil, der erst im Labor durchgeführt wird. Der Vorbereitungsteil enthält eine Reihe von Aufgaben, die **vor dem Laborversuch** bearbeitet werden müssen und in diesem Umdruck mit **Vorbereitungsaufgabe** gekennzeichnet sind. Bei der Vorbereitung werden Assemblerprogramme erstellt und, soweit möglich, mit dem Simulator getestet. Im Labor werden diese und weitere Programme auf das Dragon12-Entwicklungsboard portiert und auf der realen Hardware erprobt.

(1) **Notwendige Vorkenntnisse und Vorbereitungsaufwand**: Für die Vorbereitung dieses Laborversuchs muss ausreichend Zeit eingeplant werden. Die im Rahmen der Laborversuche 1 und 2 erstellten Programme, z.B. für die Umwandlung von Dezimal- und Hexadezimalzahlen in ASCII-Strings und die Ausgabe auf dem LCD-Display des Dragon12-Entwicklungsboards, werden in diesem Laborversuch wiederverwendet.

(2) **Laborunterlagen: Den vollständigen Laborumdruck sowie vorbereitete CodeWarrior-Projekt-Dateien finden Sie auf der Web-Seite zu dieser Vorlesung**. Entwerfen Sie alle Programme zunächst mit Hilfe eines Programmablaufplans (Flussdiagramm), bevor Sie den Programmcode schreiben. Vergessen Sie nicht, Ihre Programme ausführlich zu kommentieren. Bringen Sie zum Laborversuch sämtliche Vorbereitungsunterlagen inklusive der Programmablaufpläne und der CodeWarrior-Projektdateien mit.

Die Vorbereitung sollte in der Laborgruppe gemeinsam so durchgeführt werden, dass jeder Gruppenteilnehmer in der Lage ist, den Laborbetreuern die Lösung zu sämtlichen Vorbereitungsaufgaben selbstständig zu erklären und bei Bedarf sämtliche Programme vorzuführen und zu erläutern.

Bitte führen Sie die Vorbereitung ohne Rückgriff auf Altmeister durch, auch wenn das eine deutliche Zeitersparnis bedeuten würde. In der Klausur werden Sie auch keinen Altmeister haben und die Labors sind in dieser Lehrveranstaltung die allerbeste Prüfungsvorbereitung. Softwareentwicklung in einer beliebigen Programmiersprache kann man nur dann wirklich lernen, wenn man es selbst macht!

Mangelhafte Vorbereitung oder unentschuldigtes Fehlen führt zum Nichtbestehen des Labors. Das nichtbestandene Labor kann dann im folgenden Semester wiederholt werden.

(3) **Labordurchführung:** Während des Laborversuchs benötigen Sie in jedem Fall das Vorlesungsmanuskript sowie die vollständige Dokumentation des HCS12-Mikrocontrollers und seiner Peripheriebausteine sowie des Dragon12-Entwicklungsboards. Bringen Sie diese Unterlagen bitte zum Labor mit. Während des Labors sind die als **Laboraufgabe** gekennzeichneten Fragen zu bearbeiten.

2. DCF77-Signalformat

Der von der Physikalisch-Technischen Bundesanstalt PTB betriebene DCF77-Sender bei Frankfurt strahlt ein digital amplitudenmoduliertes Langwellensignal aus, das von einem einfachen Funkempfänger empfangen und demoduliert werden kann. Das Ausgangssignal des Funkempfängers wird mit einem Digitaleingang unseres Mikrocontrollers verbunden. Diese Rechtecksignalfolge enthält bitweise codiert die Datums- und Uhrzeitinformation und soll von der Software decodiert werden, die Sie in diesem Laborversuch entwickeln werden.

Das Rechtecksignal (Bild 2.1) liegt im Ruhezustand auf H (High) und enthält im Abstand von genau 1sec L-(Low)-Impulse (Sekundenimpulse). Jeder L-Impuls enthält, über seine Länge codiert, ein Bit der Datums- und Uhrzeitinformation. Datum und Uhrzeit werden jede Minute erneut übertragen. Der Beginn einer Minute wird dadurch angezeigt, dass der vorherige Sekundenimpuls, d.h. die 59 Sekunde, fehlt (Minutenmarke).

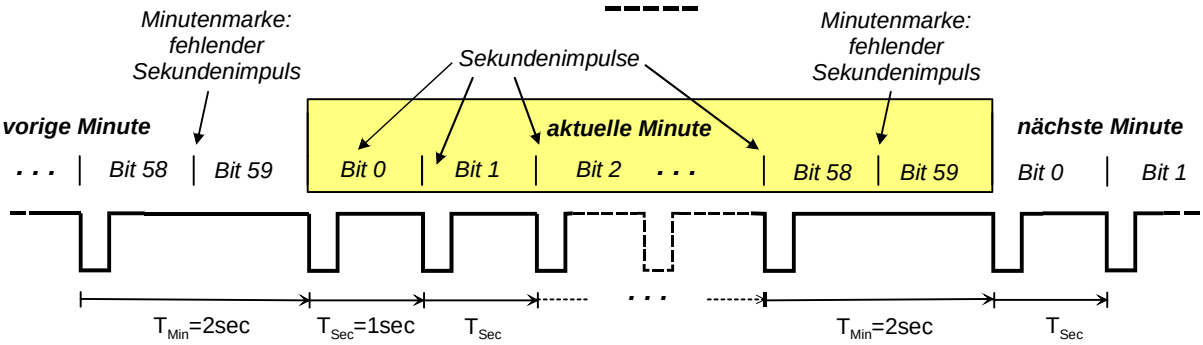

Bild 2.1 DCF77-Format: Sekundenimpulse und Minutenmarken

Die Uhrzeit- und Datumsinformation wird binär in der Dauer der L-Impulse codiert. Ein kurzer Impuls mit etwa 100ms Dauer überträgt ein 0-Bit, ein langer Impuls mit ca. 200ms Dauer überträgt ein 1-Bit (Bild 2.2).

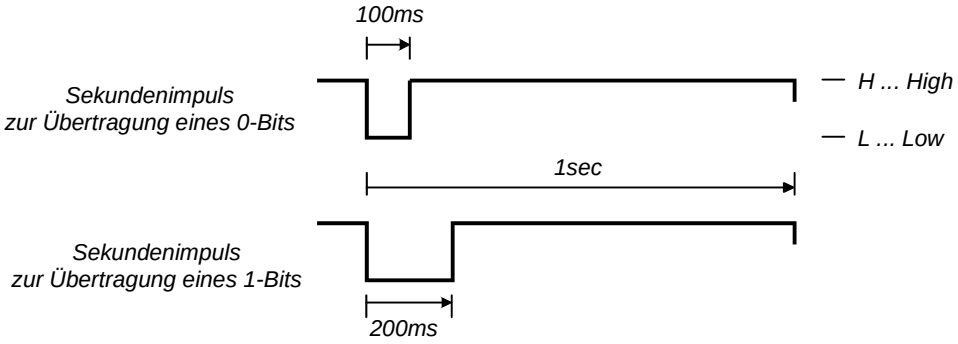

Bild 2.2 Sekundenimpulse zur Übertragung von ‚0' und ‚1'

Die je Minute übertragenen 59 Datenbits enthalten neben dem Datum und der Uhrzeit noch einige Zusatz- und Steuerinformationen (Einzelheiten z.B. in [1]). Für unsere Aufgabenstellung von Bedeutung sind nur die eigentlichen Zeit- und Datumsinformationen (Bild 2.3):

Sekunde bzw. Bit	*Bedeutung*
0 bis 16	Für unsere Anwendung nicht von Bedeutung
17, 18	01_B = Normalzeit MEZ, 10_B = Sommerzeit MESZ
19	Für unsere Anwendung nicht von Bedeutung
20	Immer 1_B
21 bis 27	Minuten (7bit BCD)
28	Paritätsbit für die Bits 21 bis 27 (Even Parity)
29 bis 34	Stunden (6bit BCD)
35	Paritätsbit für die Bits 29 bis 34 (Even Parity)
36 bis 41	Tag (6bit BCD)
42 bis 44	Wochentag (3bit) mit 1=Montag, …, 7=Sonntag
45 bis 49	Monat (5bit BCD)
50 bis 57	Jahr (8bit BCD, nur die letzten beiden Stellen, d.h. ohne Jahrhundert)
58	Paritätsbit für die Bits 36 bis 57 (Even Parity)

Alle Zahlen werden im BCD-Code beginnend beim LSB übertragen. Die Anzahl der Bits ist so gewählt, dass die größte mögliche Zahl gerade noch übertragen werden kann. Bei Monatstagen also die Werte 1 bis 31, wofür bei der BCD-Codierung dann 6bit (2bit für die Zehnerstelle, 4bit für die Einerstelle) ausreichen. Bei der Jahreszahl dagegen sind Werte zwischen 0 und 99 möglich und daher 8bit (je 4bit für die Zehner- und für die Einerstelle) nötig.

Beispiel:
Der 25. Tag eines Monats wird folgendermaßen codiert:

$$25_D = 20 + 5 = \underset{\text{MSB}}{1}\ 0\ .\ 0\ 1\ 0\ \underset{\text{LSB}}{1}\ _{\text{BCD}}$$

Dieser Zahlenwert wird folgendermaßen übertragen:

Bit	36 LSB	37	38	39	40	41 MSB
Stellenwert	1	2	4	8	10	20
Codierung	1	0	1	0	0	1
Impuls	200ms	100ms	200ms	100ms	100ms	200ms

Weiteres Beispiel:
Das Jahr 2007 wird folgendermaßen übertragen:

Bit	50 LSB	51	52	53	54	55	56	57 MSB
Stellenwert	1	2	4	8	10	20	40	80
Codierung	1	1	1	0	0	0	0	0
Impuls	200ms	200ms	200ms	100ms	100ms	100ms	100ms	100ms

Die Bits 17, 18, 20 sowie die Paritätsbits 28, 25 und 58 können zur Überprüfung der empfangenen Daten verwendet werden.

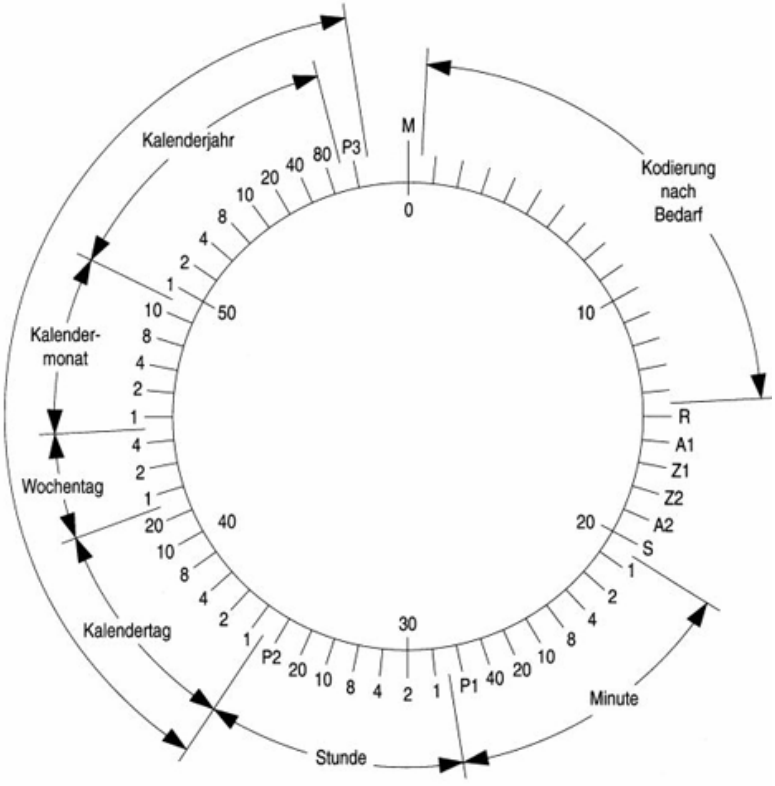

Bild 2.3

Da die Übertragung der vollständigen Information eine ganze Minute dauert, wird stets die Information für die folgende Minute gesendet. So wird z.B. ab 23.59 Uhr bereits die Zeit 00.00 Uhr und das Datum des folgenden Tages übertragen.

Weitere Informationen über DCF77 finden Sie im z.B. Internet unter

[1] Physikalisch-Technische Bundesanstalt: Webseiten zu DCF77
http://www.ptb.de/de/org/4/44/442/dcf77_1.htm

[2] Langwellensender DCF77, Wikipedia, http://de.wikipedia.org/wiki/DCF77

3. Anforderungen an die Funkuhr

Unsere Funkuhr soll folgende Anforderungen erfüllen:

- Auf dem LCD-Display des Dragon12-Entwicklungsboards soll in der oberen Zeile die aktuelle Uhrzeit im Format **`Stunde : Minute : Sekunde`**
  und in der unteren Zeile das Datum **`Tag . Monat . Jahr`**
  ausgegeben werden. Die im Labor 2 vorhandene Temperaturanzeige sowie die Einstellung der Uhr über die Taster SW2 … SW5 entfällt.
- Da der DCF77-Sender gelegentlich zu Wartungszwecken abgeschaltet wird und an manchen Standorten, z.B. in Gebäuden mit einer Stahlbetonkonstruktion, der Funkempfang nicht immer zuverlässig ist, soll das Grundkonzept der Uhr aus dem Labor 2 beibehalten werden, d.h. die eigentliche Uhr wird weiterhin mit dem Takt betrieben, der durch den Timer des HCS12 erzeugt wird. Die DCF77-Information wird lediglich dazu verwendet, die Uhr einmal je Minute auf die empfangene DCF77-Uhrzeit einzustellen und zusätzlich das Datum anzuzeigen. Falls das DCF77-Signal nicht vorhanden ist oder ausfällt, soll die Uhr einfach mit dem Timertakt allein weiterlaufen.
- Der vom Timer-Modul und seiner Treibersoftware erzeugte Takt soll weiterhin dazu verwendet werden, die LED am Port B.0 im Sekundentakt blinken zu lassen.
- Die LED am Port B.1 soll immer dann eingeschaltet werden, wenn das Rechtecksignal Low ist, und ausgeschaltet, wenn es wieder High ist. Wenn ein DCF77-Signal empfangen wird, blinkt diese Leuchtdiode damit ebenfalls im Sekundentakt, setzt aber einmal je Minute bei der Minutenmarke aus.
- Die LED am Port B.3 soll immer eingeschaltet werden, sobald eine vollständige und fehlerfreie DCF77-Datums- und Zeitinformation empfangen wurde, und soll wieder abgeschaltet werden, falls in einer Minute gar keine oder eine fehlerhafte Datums- und Zeitinformation empfangen wurde.
- Die DCF77-Datums- und Zeitinformation soll zumindest rudimentär auf Gültigkeit geprüft werden, bevor die Uhr damit nachgestellt und das Datum angezeigt wird. Sie könnten z.B. überprüfen, ob die Bits 17, 18 und 20 tatsächlich korrekte Werte aufweisen und ob die Zeit- und Datumsangaben plausibel sind, d.h. z.B. Stundenwert im Bereich 0 … 23, Minuten im Bereich 0 … 59. Wenn Sie dabei Fehler feststellen, sollten Sie zu Testzwecken die LED am Port B.2 solange einschalten, bis Sie wieder eine fehlerfreie Botschaft decodiert haben.
- Das Rechtecksignal des DCF77-Funkempfängers wird im Labor an den Port H.0 des HCS12 angeschlossen. Schreiben Sie Ihr Programm aber so, dass ein beliebiger anderer Port verwendet werden kann. Gehen Sie daher davon aus, dass der Port lediglich gelesen werden kann, aber keinen Interrupt auslösen kann, obwohl dies beim Port H möglich wäre. Die Erkennung der negativen und positiven Signalflanken des Rechtecksignals und die Zeitmessung sollen in Software erfolgen. Isolieren Sie die Konfiguration des Ports als Eingang in einer Funktion **`void initPort(void)`** und das Lesen des Ports in einer Funktion **`char readPort()`**, die 0 zurückgibt, wenn das Rechtecksignal Low ist und einen Wert >0, wenn das Signal High ist.

4. Mögliche Softwarearchitektur für die Implementierung

Im CodeWarrior-Projekt **`lab3-Funkuhr-Vorlage.mcp`** finden Sie eine Implementierung einer freilaufenden Uhr, die im Kern der Uhr aus Labor 2 entspricht. Das Hauptprogramm **`main.c`** sowie die Uhrenfunktion in **`clock.c`** sind in C geschrieben, verwenden aber die bekannten, in Assembler geschriebenen LCD- und Timer-Treiber, die für diesen Versuch teilweise modifiziert und vereinfacht wurden.

Die Funktion der freilaufenden Uhr ist im Wesentlichen auf drei Funktionen verteilt (Bild 4.1):

- Die Funktion **`tick10ms()`** ist ein Unterprogramm, das aus der ISR des Timer-Moduls und damit periodisch alle 10ms aufgerufen wird. Da ISR-Funktionen möglichst kurz sein sollen, inkrementiert diese Funktion lediglich den Softwaretimer **`uptime`**, ruft die Funktion **`sampleDCF77Signal()`** auf (siehe unten) und setzt einmal je Sekunde die globale Variable **`clockEvent`**.
- Das Hauptprogramm fragt diese Variable ab und ruft, wenn sie gesetzt ist, d.h. einmal je Sekunde, die Funktion **`processEventsClock()`** auf. Diese Funktion zählt die Sekunden und wandelt sie in die Uhrzeit bestehend aus Stunden, Minuten und Sekunden um, die in globalen Variablen gespeichert werden.
- Anschließend ruft das Hauptprogramm die Funktion **`displayTimeClock()`** auf, die die Uhrzeit mit Hilfe des LCD-Treibers auf dem LCD-Display anzeigt und setzt die globale Variable **`clockEvent`** wieder zurück.

Diese Aufteilung hat den Vorteil, dass die hardwareabhängigen Teile von der eigentlichen Uhrenfunktion isoliert sind und die drei Funktionen sehr gut unabhängig voneinander getestet werden können.

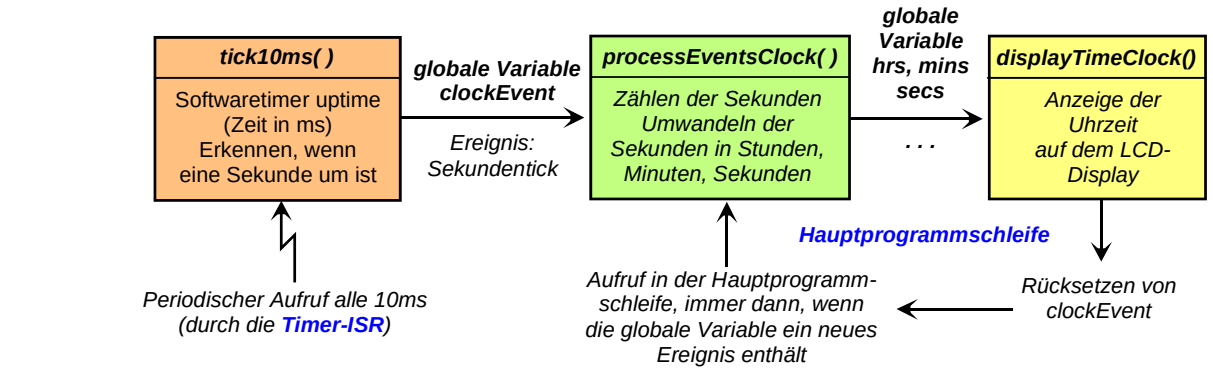

Bild 4.1: Architektur der freilaufenden Uhr

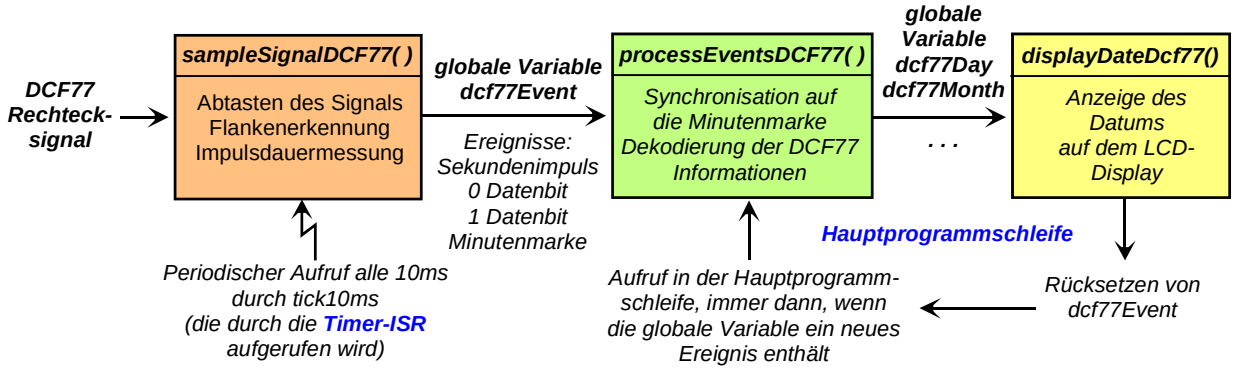

Bild 4.2: Mögliche Architektur der DCF77-Funkuhr-Erweiterung

Dieselbe Architektur eignet sich auch für die DCF77-Funkuhr-Erweiterung (Bild 4.2):

- Das DCF77-Rechtecksignal wird in der Funktion **`sampleDCF77Signal()`** periodisch im Abstand von 10ms abgetastet. Dabei werden durch Vergleich des aktuellen Signals mit dem Wert bei der letzten Abtastung die Signalflanken erkannt und die Zeiten T_{Low} und T_{Pulse} gemessen (Bild 4.3). Die Zeitmessung erfolgt mit Hilfe des Softwaretimers **`uptime`** (siehe oben), dessen aktuellen Wert die Funktion beim Aufruf z.B. als Parameter erhält. Theoretisch könnte auch das Timer-Modul direkt verwendet werden, doch deckt der Softwarezähler mit einer Periodendauer von 65536ms einen wesentlich größeren Zeitbereich ab als der Hardwaretimer TCNT des Timer-Moduls. Die Funktion wertet bei jeder positiven bzw. negativen Signalflanke die jeweils gemessene Zeit als „Ereignis" aus und liefert das Ergebnis als **`dcf77Event`**. Mögliche Ereignisse und ihre Zeitpunkte sind:

Ereignis	*Signalflanke*	*Bedeutung und Bedingung*
NODCF77EVENT	Keine	Signal seit der letzten Abtastung unverändert
VALIDSECOND	Negativ	Gültiger Sekundenimpuls, wenn T_{Pulse}=1s ± 100ms. Das Toleranzfenster ist notwendig, um leichte Schwankungen des DCF77-Signals und der Periode der Timer-ISR zu berücksichtigen.
VALIDMINUTE	Negativ	Gültige Minutenmarke, wenn T_{Pulse}=2s ± 100ms.
VALIDZERO	Positiv	Gültiges 0-Bit, wenn T_{Low}=100ms ± 30ms
VALIDONE	Positiv	Gültiges 1-Bit, wenn T_{Low}=200ms ± 30ms
INVALID	Negativ Positiv	Wenn T_{Pulse} außerhalb des Toleranzfensters liegt. Wenn T_{Low} außerhalb des Toleranzfensters liegt.

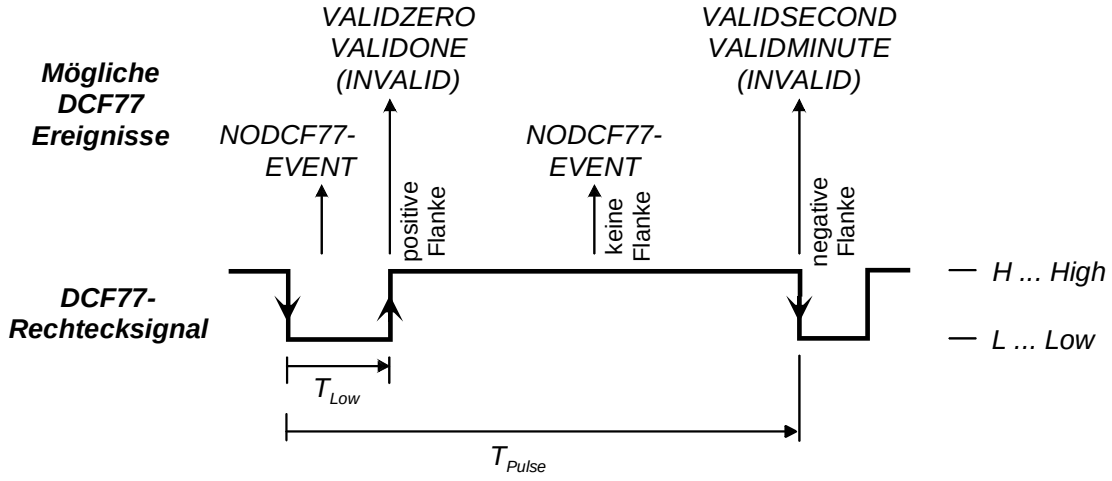

Bild 4.3: Auswertung des DCF77-Rechtecksignals

- Die Funktion **`processDCF77Events()`** wird im Hauptprogramm immer dann aufgerufen, wenn eines der Ereignisse (außer NODCF77EVENT) aufgetreten ist, speichert die empfangenen DCF77-Datenbits und decodiert Zeit und Datum, sobald ein vollständiges Datenpaket angekommen ist (d.h. frühestens nach Bit 57). In diesem Fall wird dann die freilaufende Uhr über die Funktion **`setClock()`** auf die empfangene Zeit gestellt. Diese Funktion kann z.B. nach dem bekannten Konzept eines Zustandsautomaten (Finite State Machine) entworfen werden.
- Anschließend ruft das Hauptprogramm die Funktion **`displayDateDCF77()`** auf, die das Datum mit Hilfe des LCD-Treibers auf dem LCD-Display anzeigt und setzt die globale Variable **`dcf77Event`** wieder zurück.

5. Entwurf der Funkuhr

Die vorgestellte Architektur ist nur eine Möglichkeit, die Aufgabenstellung zu lösen. Sie dürfen für Ihre eigene Lösung ein beliebiges anderes Konzept verwenden, solange Sie die in Abschnitt 3 dargestellten Anforderungen erfüllen. Wir erwarten von Ihnen, dass Sie das Programm für die Funkuhr in Ihrer Laborgruppe selbstständig entwickeln und keinen Programmcode anderer Laborgruppen oder aus anderen Quellen verwenden. Den Programmcode aus der Vorlage **`lab3-Funkuhr-Vorlage.mcp`** und Ihren eigenen Programmcode aus dem Labor 2 dagegen dürfen Sie beliebig verwenden oder modifizieren.

Vorbereitungsaufgabe 5.1

Beschreiben Sie die von Ihnen gewählte Softwarearchitektur durch Bilder ähnlich wie Bild 4.1 und 4.2 im Abschnitt 4 und erstellen Sie detaillierte Flussdiagramme zur Auswertung des DCF77-Rechtecksignals und zur Decodierung der DCF77-Botschaft.

Implementieren Sie die gesamte Software. Das Einlesen des DCF77-Rechtecksignals und die Messung der Impulsdauern (in unserem Vorschlag die Funktionen **`readPort()`** und **`sampleDCF77Signal()`** sollen in Assembler geschrieben werden. Die Funktionen zur Auswertung, Decodierung und Anzeige der DCF77-Daten (bei uns **`processEventsDCF77()`, `displayDateDCF77()`**) dürfen Sie beliebig in C oder Assembler schreiben.

Versuchen Sie, soviel wie möglich von Ihrer Software mit dem Simulator ohne Hardware zu testen, so dass Sie im Labor keine oder nur noch sehr wenig Software schreiben müssen. Testen Sie Funktion für Funktion und schreiben Sie gegebenenfalls kleine Testprogramme, um einzelne Funktionen testen zu können.

Bringen Sie Ihre vollständigen Vorbereitungsunterlagen und die (fast) fertige Software zur Laborübung mit. Dort müssen Sie dem Laborbetreuer Ihren Entwurf anhand der Flussdiagramme erläutern und vorführen, wie Sie die einzelnen Funktionen getestet haben.

Hinweise:

- Als Hilfsfunktionen für den Test enthält die Projektvorlage im Modul **`dcf77Sim.c`** die Funktion **`readPortSim()`**, die das DCF77-Rechtecksignal simuliert. Die Funktion kann beim Testen anstelle der Funktion **`readPort()`** eingesetzt werden. Sie muss periodisch alle 10ms genau einmal aufgerufen werden und liefert einen Logikwert ‚0' oder ‚1' zurück, der dem „aktuellen" Wert eines DCF77-Rechecksignals entspricht.
  Denken Sie aber daran, dass die Simulation deutlich langsamer als in Echtzeit abläuft. Für die Simulation einer vollständigen DCF77-Botschaft, d.h. 1 min in Echtzeit, benötigt der Simulator der HCS12-Entwicklungsumgebung auf einem 1,6GHz Intel Core-Prozessor etwa 3 min bei 100% CPU-Auslastung. Dies liegt daran, dass für 1min Echtzeit immerhin gut 1,4 Milliarden HCS12-CPU-Takte simuliert werden. Damit macht es wenig Sinn, wenn Sie das gesamte Funkuhr-Programm im Trial- und Error-Verfahren mit dem Simulator entwickeln. Sie brauchen eine wohl durchdachte Programmarchitektur und eine klare Teststrategie für die einzelnen Funktionen, sonst sitzen Sie tagelang vor dem Simulator, bevor Ihr Programm einigermaßen funktioniert …
- Enumerationen wie **`enum { NODCF77EVENT, VALIDZERO, …}`** können auch in Assembler verwendet werden. Sie entsprechen beim HCS12 einer einfachen 8bit-Variable, wobei das erste Element der Enumeration den Wert 0, das zweite den Wert 1, das dritte den Wert 3 usw. erhält.

6. Programmtest auf dem Dragon12-Entwicklungsboard

Laboraufgabe 6.1

Während des betreuten Teils der Laborübung integrieren Sie die von Ihnen zuvor geschriebene Software auf der Hardware. Testen müssen Sie nun besonders die Funktionen, die ohne Hardware nicht gut prüfbar waren.

Testen Sie gründlich, ob Ihre Funkuhr wie gewünscht funktioniert.