HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Christhard Vorberg

Empfang und Dekodierung des phasenmodulierten Signals des DCF77-Zeitzeichensenders

Mittweida, 2014

Fakultät Elektro- und Informationstechnik

DIPLOMARBEIT

Empfang und Dekodierung des phasenmodulierten Signals des DCF77-Zeitzeichensenders

Autor:
Herr

Christhard Vorberg

Studiengang:
Informationstechnik

Seminargruppe:
KI09w1

Erstprüfer:
Prof. Dr.Ing. Thomas Beierlein

Zweitprüfer:
Dipl.Ing. Gerd Bucher

Einreichung:
Mittweida, 30.6.2014

Verteidigung/Bewertung:
Mittweida, 2014

Faculty of Electrical Engineering and Information Technology

Diploma FH THESIS

Receive und Decode the phase modulated signal of the DCF77-Time-Signal-Station

author:
Mr.

Christhard Vorberg

course of studies:
Information Technology

seminar group:
KI09w1

first examiner:
Prof. Dr.Ing. Thomas Beierlein

second examiner:
Dipl.Ing. Gerd Bucher

submission:
Mittweida, 30.6.2014

defence/ evaluation:
Mittweida, 2014

Bibliografische Beschreibung:

Christhard, Vorberg:
Empfang und Dekodierung des phasenmodulierten Signals des DCF77-Zeitzeichensenders - 2014 - 13, 41, 31.
Mittweida, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik, Diplomarbeit, 2014

Referat:

Es wird eine Hard- und Softwarelösung zum Empfang des deutschen Zeitzeichensenders DCF77 vorgeschlagen, welche auf dem phasenmodulierten Signal des DCF77 beruht. Die Demodulation erfolgt dabei in der Software mittels SDR-Prinzipien. Zur Decodierung auf Bit-Ebene wird eine Kreuzkorrelation mit einer Pseudozufallsfolge vorgeschlagen. Es werden Anwendungen des hier vorgestellten Prinzips diskutiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

SDR	**Software Defined Radio – Empfänger oder Sender von elektromagnetischer Wellen. Die wesentlichsten Funktionen werden in der Software realisiert.**
PZF	**Pseudozufallsfolge – Zufälliges Signal gebildet durch einen Algorithmus. Meist durch ein rückgekoppeltes Schieberegister erzeugt.**
DCF77	**Rufzeichen des deutschen Zeitzeichen- und Normalfrequenzsenders.**
GPS	**Global Positioning System – Amerikanisches System zur Navigation. Es sendet sehr genaue Zeitmarken aus.**
Eagle	**Einfach Anzuwendender Graphischer Layout-Editor.**
LTspice	**Linear Technologie Simulation Program with Integrated Circuit Emphasis**
PLL	**phase-locked-loop, phasenverriegelte Schleife, Phasenregelschleife**
FLL	**frequency locked loop, frequenzverriegelte Schleife**
OPV	**Operationsverstärker**
MAC	**multiply accumulate operation**

1 Der DCF77 und seine Modulationsarten

"Falls ich in 5 Minuten nicht zurück sein sollte,
warten Sie einfach ein bisschen länger."
Ace Ventura in „Ace Ventura - Ein tierischer Detektiv“

1.1 Der deutsche Zeitzeichensender DCF77

1.1.1 Historisches und Gesetzliches

Am 12. März 1893 unterschieb der deutsche Kaiser Wilhelm II das Gesetzblatt Nr.7 (Anlage Teil 1). Darin stand der wichtige Satz: „Die gesetzliche Zeit in Deutschland ist die mittlere Sonnenzeit des fünfzehnten Längengrades östlich von Greenwich“. Nach in Inkrafttreten des Gesetzes am 1.4.1893, hatte Deutschland das erste Mal in der Geschichte eine einheitliche Zeit.

Dieses Gesetz galt bis zum 1.8.1978. Ab diesem Tag galt das „Gesetz über die Zeitbestimmung“ (ZeitG). Im § 1 Absatz 2 steht: „Die gesetzliche Zeit ist die mitteleuropäische Zeit. Diese ist bestimmt durch die koordinierte Weltzeit unter Hinzufügung einer Stunde.“ Weiterhin steht im §2 dieses Gesetzes „Die gesetzliche Zeit wird von der Physikalisch-Technischen Bundesanstalt dargestellt und verbreitet.“

Am 12. Juli 2008 ging der Wortlaut des ZeitG im „Gesetz über die Einheiten im Messwesen und die Zeitbestimmung“ (EinhZeitG) auf. Im §6 des EinhZeitG heißt es: “Die Physikalisch-Technische Bundesanstalt hat (…) gesetzliche Zeit darzustellen und zu verbreiten“.

Die z. Z. gültige Definition [8] der Sekunde als SI-Einheit liefert auch den Ansatz zur Darstellung der Zeit bei der Physikalisch-Technische Bundesanstalt (PTB) mit Sitz in Braunschweig: Eine Sekunde entspricht dem 9 192 631 770-fachen der Periodendauer beim Übergang zwischen den beiden Hyperfeinstrukturniveaus des Grundzustandes von Atomen des Cäsium-Isotops ^{133}Cs entsprechenden Strahlung. Im PTB stehen nun mehrere sog. Atomuhren, welche genau diese Strahlung erzeugen und messen. Daraus können durch Frequenzteilung auch Sekundenimpulse gewonnen werden. Diese hochgenauen Atomuhren werden dann mit anderen Atomuhren aus anderen Ländern verglichen. Es entsteht die sogenannte Internationale Atomzeit (TAI). Da die Erdumdrehung aber Schwankungen unterworfen ist, würde diese sehr genaue Zeit TAI zu Unterschieden beim Sonnenaufgang führen. Deshalb wird die TAI unter Einfügung oder Weglassen von Schaltsekunden zur Coordinated Universal Time (UTC). Diese Zeit wiederum bildet die

Grundlage für die Mitteleuropäische Zeit bei Addition einer Stunde bzw. für die mitteleuropäische Sommerzeit bei Addition von zwei Stunden.

Die Physikalisch-Technische Bundesanstalt (PTB) kommt ihrer Pflicht, die gesetzliche Zeit zu verbreiten, mittels dreier verschiedener Dienste nach:

1. Über das öffentliche Telefonnetz,
   unter der Rufnummer 0531 512038 kann mittels V.22-Modem bei 1200 Baud die Zeit abgefragt werden.

2. Über das Internet,
   die Adressen ptbtime1.ptb.de, ptbtime2.ptb.de und ptbtime3.ptb.de ermöglichen es die gesetzliche Zeit per NTP und UDP zu ermitteln.

3. Über Funkwellen,
   der in Mainflingen stehende Langwellensender DCF77 überträgt dabei nicht nur die gesetzliche Zeit, sondern dient zur Ausstrahlung einer Normalfrequenz.

Die Arbeit beschäftigt sich nun im Wesentlichen mit dem Dienst der unter 3. genannt ist, denn die Übertragung der Zeit mittels Funkwellen benötigt das geringste technische Equipment. Des weiteren sind keine Kabelsysteme oder andere Dienste nötig.

1.1.2 Die Sendetechnik des DCF77

Abbildung 1 Sendefunkstelle Mainflingen (aus [1])

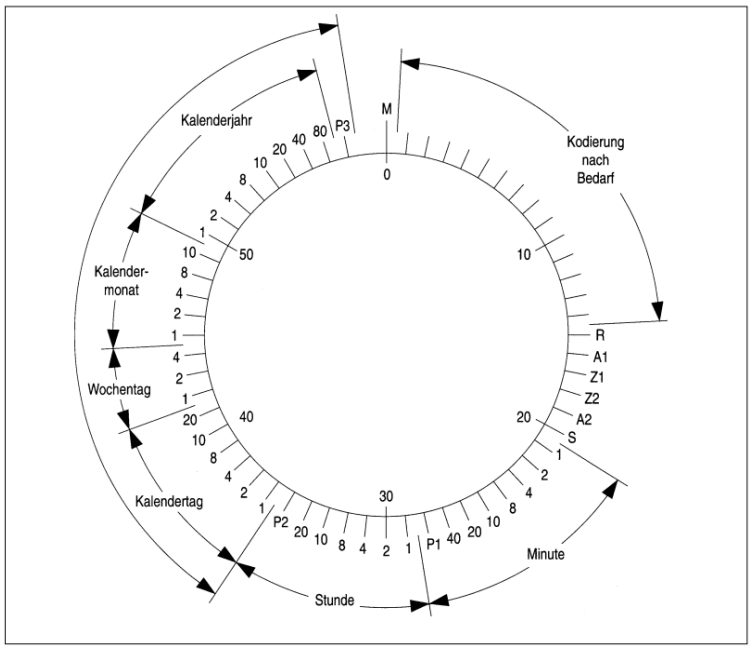
Die in Abbildung 1 zu sehende Antenne dient zur Abstrahlung des Signals des DCF77. Diese Antenne ist auf 150 m hohen Masten montiert. Die Masten stehen natürlich senkrecht. Durch das verwendete Weitwinkelobjektiv kommt es zu der scheinbaren Schrägstellung. Diese Antenne hat Rundstrahleigenschaften und zusammen mit dem 50-kW-Halbleitersender eine geschätzte isotrope Strahlungsleistung von ca. 30-35 kW [1].

Abbildung 2 DCF77 Halbleitersender (aus [9])

1.2 Die Amplitudenmodulation des DCF77

Zur Informationsübertragung mittels elektromagnetischer Strahlung müssen die Parameter dieser Strahlung verändert (moduliert) werden. Es stehen 3 Parameter zu Verfügung: die Amplitude, die Frequenz und die Phase. Da der Sender DCF77 auch als sog. Normalfrequenzsender dient, verbietet sich die Veränderung der Frequenz (FM). Tatsächlich wird die Frequenz der DCF77 auf genau 77500 Hz mittels einer am Standort befindlichen Cäsium-Atomuhr konstant gehalten. Es wird eine Frequenzabweichung von ca. $2 \cdot 10^{-12}$ erreicht [2]. Einen Sender in der Amplitude zu modulieren ist bei geringem Aufwand leicht möglich. Auch ist der Empfang amplitudenmodulierter Signale leicht zu realisieren. Deshalb wird für einfache Funkuhren dieses Modulationsverfahren angewendet.

Wie wird die Amplitudenmodulation (AM) zur Informationsübertragung beim DFC77 benutzt? Grundsätzlich werden Impulse im Sekundenabstand übertragen. Dabei wird nach genau 77500 Schwingungen des Trägers der Pegel des Sendesignals auf 15 % abgesenkt. Die Absenkung wird für genau 7750 oder 15500 Perioden des Trägers aufrechterhalten. Es ergibt sich also Absenkung von 0,1 s oder 0,2 s Länge. Der Abstand von Beginn Absenkung bis zur nächsten Absenkung ist konstant 1 Sekunde. Bei jeder vollen Minute wird die Absenkung weggelassen. Die neue Minute (Sekunde 0) startet mit der ersten Absenkung der Trägeramplitude. Es werden also 59 (0-58) Absenkungen pro Minute verwendet. Die jeweils gültige Zeit wird als 1 Bit pro Sekunde übertragen. Dabei entspricht die Absenkung um 0,1 s einer logischen Null. Die Absenkung um 0,2 s entspricht einer logischen Eins. Über eine Minute können also 59 Bit übertragen werden. Die Zeit und das Datum werden als BCD-Zahlen übermittelt. Weiterhin werden verschlüsselte Wetterinformationen (Bit 0 – Bit 14) und Prüfsummen sowie Statusbits übermittelt. Nachfolgendes Kodierschema aus [2] verdeutlicht dieses.



Legende:
M: Minutenmarke 0,1 s

R: Rufbit

A1: Ankündigung Sommer / Winterzeitumschaltung

A2: Ankündigung Schaltsekunde

Z1 Z2: Zonenzeitbits (MEZ)

S: Startbit der Zeitübertragung 0,2 s

P1 P2 P3: Prüfbits

Abbildung 3 Kodierschema (aus [2])

1.3 Die Phasenmodulation des DCF77

Eine Phasenmodulation zu realisieren ist sowohl im Sender als auch im Empfänger viel schwieriger als die Modulation der Amplitude. Es müssen genaue Zeitbedingungen eingehalten werden. Besonders im Empfänger ist ein hoher Aufwand nötig, da Zeitunterschiede von Bruchteilen einer Trägerschwingungsdauer detektiert werden müssen. Dieser Aufwand kann sich aber lohnen, weil die Zeitimpulse um Größenordnungen genauer übertragen werden können. In [2] wird von zeitlichen Schwankungen der Impulse von weniger als 6,5 µs gesprochen. Des Weiteren ist ein phasenmoduliertes Signal störsicherer gegen Impulsstörungen, da diese die kompletten Sekundenimpulse überdecken können. Das ist besonders bedenklich, da die Sekundenimpulse bei abgesenktem Träger übertragen werden. Viele Störungen sind Impulsstörungen. Besonders Schaltnetzteile, LED-Leuchtmittel und Dimmer senden eine Störstrahlung im Bereich der vom DCF77 benutzten Frequenz von 77500 Hz aus. Ältere Röhrenfernseher mit 50-Hz-Technik können den DCF77 auch stark stören, da die 4. Oberwelle der Zeilenfrequenz bei 78125 Hz liegt. Bei der Arbeit zu [11] wurden entsprechende Beobachtungen gemacht.

Das Kodierschema aus Abschnitt 1.2 Abbildung 3 gilt für die phasenmodulierte Übertragung uneingeschränkt mit der Ausnahme, dass keine Wetterinformation übertragen wird. In den Sekunden 0 – 9 werden logische Einsen übertragen. Diese dienen zur Detektion des Minutenbeginns. Die Phasenmodulation dient zu Übertragung der jeweiligen logischen Bits 0 oder 1 in jeder Sekunde. Zusätzlich kann der Sekundenbeginn genauer bestimmt werden. Diese genaue und störsichere Bestimmung des Sekundenbeginns stellt die eigentliche Leistung des PM-Signals des DCF77 dar. Die Übertragung der Zeitinformation könnte auch über die AM-Signale erfolgen. Die Zeitinformation wird aber sinnvollerweise auch aus der Phasenmodulationsinformation gewonnen, da eine höhere Störsicherheit gegeben ist.

Die Phasenmodulation des DCF77 kennt 3 Zustände des Trägers [3]:

1. Träger nicht moduliert
   Vom Sekundenbeginn bis 200ms ist der Träger in der Phase nicht moduliert

2. Trägerschwingung um +13° verschoben
   Es wird das Signal „low“ übertragen (nicht zu verwechseln mit logisch „0“)

3. Trägerschwingung um -13° verschoben
   Es wird das Signal „high“ übertragen (nicht zu verwechseln mit logisch „1“)

Die hier genannten Signale „low“ und „high“ werden in jeder Sekunde genau 256mal übertragen. Es ergeben sich also 512 Bits pro Sekundenimpuls. Die Verteilung der einzelnen „low“ – und „high“ Bits gehorcht einer Pseudozufallsfolge (PZF). Die Generierung der PZF wird im folgenden Abschnitt beschrieben. Eines dieser Bits ist genau 77500 Hz / 120 Sekunden lang. Es ergibt sich eine Bitfrequenz von 645,833 Hz entsprechend ca. 1,55 ms.

Da 512 Zufallsbits übertragen werden, ergibt sich daraus eine Übertragungszeit von ca. 0,79277 Sekunden. Wird die PZF nicht negiert übertragen, stellt das eine logische „0“ dar. Bei einer negierten PZF wurde eine logische „1“ übertragen. Diese logischen Binärwerte entsprechen den logischen Werten der Amplitudenmodulation und können einer Auswertung zur Gewinnung der Zeitinformation zugeführt werden. Nachfolgende Zeichnung aus [3] verdeutlicht den Zusammenhang:

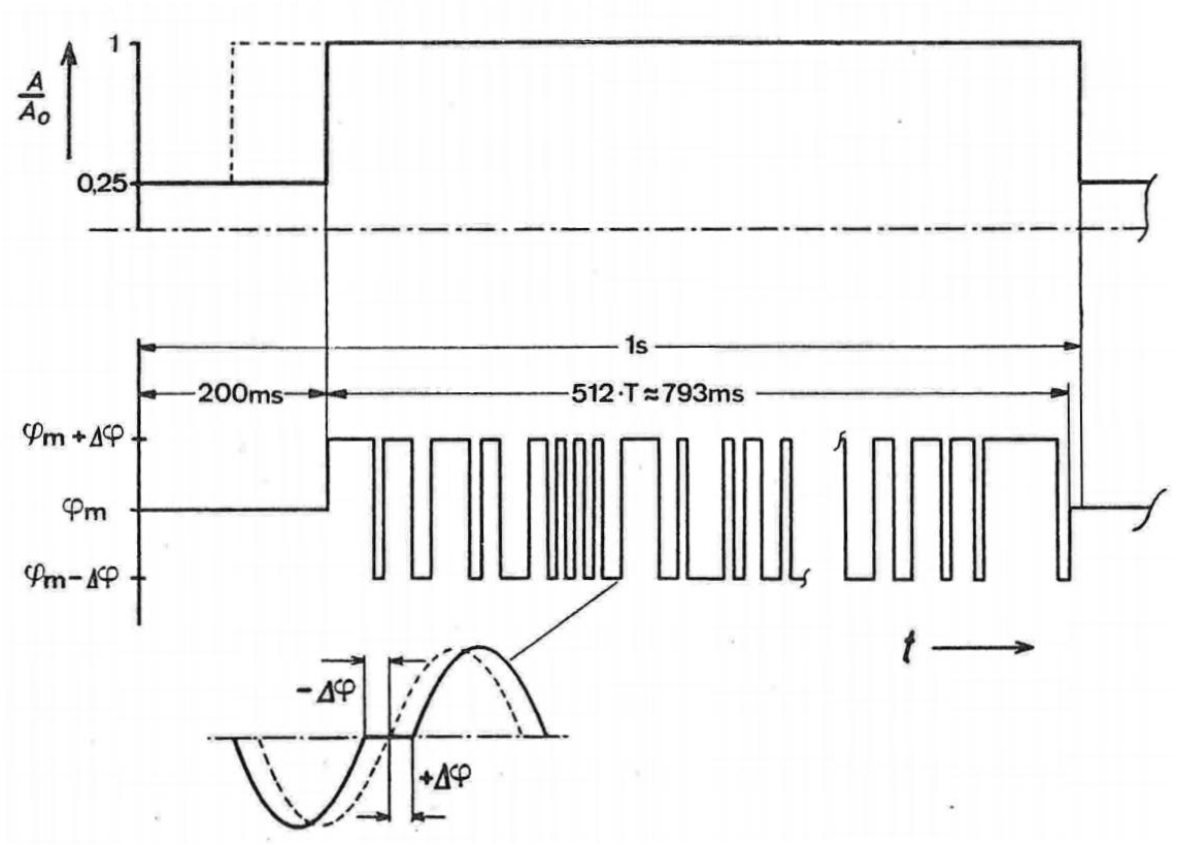


Abbildung 4 Modulation PZF (aus [3])

Zusammen mit dem Minutenimpuls, bestehend aus 10mal logisch „1“, kann über das Kodierschema der Abbildung 3 die Zeitinformation gewonnen werden.

Warum wird aber solch ein großer Aufwand betrieben um nur 1 Bit zu übertragen? Es wird jedes Bit um den Faktor 512 vergrößert. Man spricht von Bitspreizung. Der Vorteil liegt in der höheren Störsicherheit. Von den übertragenen 512 Bit müssen nur ein Bruchteil korrekt detektiert werden, um trotzdem eine logische „0“ oder „1“ zu erkennen. Mit diesem Verfahren ist es möglich Daten sicher zu übertragen, obwohl das Trägersignal im Rauchen verborgen ist. Da die PZF bekannt ist, muss das empfangene Signal nur mittels Kreuzkorrelation bewertet werden. Ergibt die Kreuzkorrelation einen betragsmäßig hohen Wert, so ist ein logisches Signal erkannt worden. Hat dieser Wert ein positives Vorzeichen, wurde eine logische „0“ übertragen. Bei negativem Vorzeichen wurde eine „1“ übertragen.

Um jedoch den genauen Sekundenbeginn zu erkennen, ist weiterer Aufwand nötig: Damit die Kreuzkorrelation einen Maximalwert liefert, muss die Kreuzkorrelation an einem bestimmten Zeitpunkt beginnen. Da dieser Zeitpunkt nicht bekannt ist, muss dieser gesucht werden. Bei gestörtem Amplitudensignal (dieses könnte den ungefähren Zeitpunkt +200 ms angeben), muss über die gesamte Sekunde in Schritten von höchstens 1 Bit-Zeit (1 / 77500 Hz * 120 = 1,55ms) gesucht werden. Ein Suchzyklus würde aus 645 (77500 Hz / 120) Einzelsuchen bestehen. Es könnte also sein, dass man 10 min warten müsste, um den korrekten Sekundenbeginn zu bestimmen. Einen Ausweg bietet der Einsatz von mehreren parallelen Korrelationsvorgängen. Diese würden jeweils im Abstand von ca. 1,5 ms gestartet. Beim Einsatz von 667 Stück (1 s / 1,5 ms) würde der Sekundenbeginn schon nach einer Sekunde feststehen und es müsste keine zyklische Suche angewendet werden.

Das nachfolgende Bild illustriert das Amplituden- und Phasensignal als Wasserfalldiagramm. Die PZF ist als Rauschen zu sehen. Sichtbar sind auch die 200-ms-Lücken im Rauschsignal (PZF) zu jedem Sekundenbeginn. Das Minutenende (59. Sekunde) ist als durchgehender Träger etwa in der Mitte des Bildes zu sehen. Der nachfolgende Sekundenimpuls ist der Beginn der neuen Minute. Das Diagramm wurde mittels IQ-Mischempfänger bei 78125 Hz Mischfrequenz erzeugt.

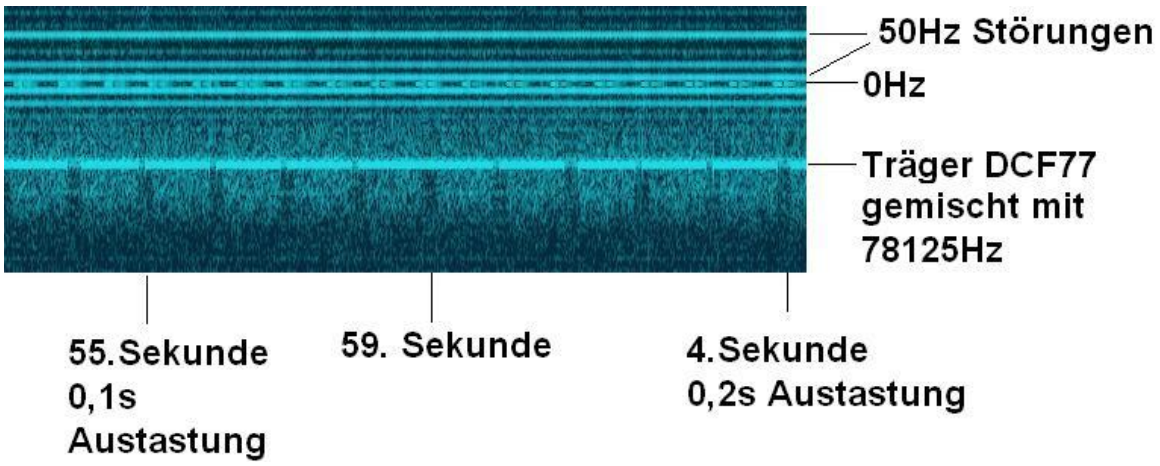


Abbildung 5 PZF-Signal

1.3.1 Generierung der Pseudozufallsfolge (PZF)

Im vorherigen Abschnitt wurde oft über die PZF des DCF77 gesprochen. Leider ist diese nicht veröffentlicht. In [3] ist aber diese Zeichnung angegeben:

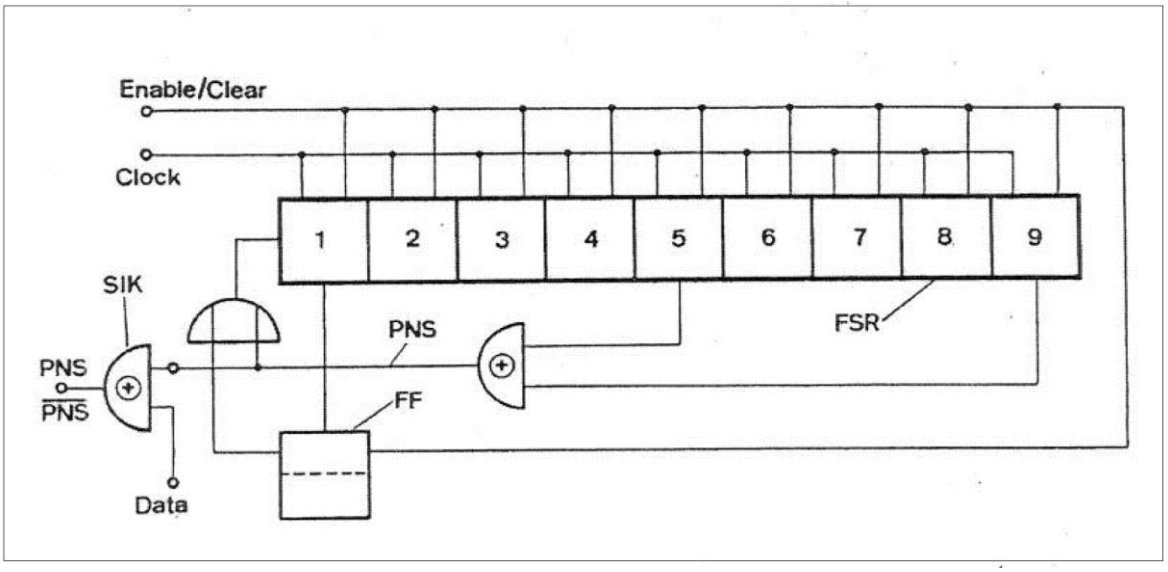


Abbildung 6 PZF-Generator (aus [3])

Mit Hilfe dieser Zeichnung konnte im Rahmen einer Belegarbeit [11] ein C-Programm entwickelt werden, welches die PZF erzeugt. Dieses Programm und die erzeugte PZF sind in Anlage Teil 2 hinterlegt. Die PZF muss nur einmal generiert werden, darum liefert das Programm ANSI-C-Code zur Einbindung als header-Datei. Die vom Programm generierten Kommentare dienen zur Überprüfung, da das Programm genau 256mal „0“ und 256mal „1“ liefern muss.

2 Vorhandene Lösungen

„Tsä...kein Wunder, dass die Elektronik versagt hat.
Hier steht MADE IN JAPAN."
Dr. Emmett Brown in „Zurück in die Zukunft III“

2.1 Vorteile des phasenmodulierten Signals des DCF77

Wie im vorhergehenden Kapitel erwähnt, ist der Empfang eines phasenmodulierten Signals wesentlich schwieriger zu realisieren als der Empfang eines amplitudenmodulierten Signals. Um die Phasenlage eines Signals bewerten zu können, ist ein dem Trägersignal synchrones Referenzsignal oder entsprechende Zeitmarken notwendig. Weiterhin ist die bereits erwähnte Kreuzkorrelation vorzunehmen. Insgesamt ist der Aufwand beachtlich.

Warum sollte man also das Phasensignal auswerten und sich nicht auf das Amplitudensignal verlassen? Es sind vor allem 2 Gründe dies zu tun: Störsicherheit und Genauigkeit.

2.1.1 Störsicherheit

Viele elektrische Geräte erzeugen elektromagnetische Schwingungen, welche im genutzten Frequenzbereich liegen. Amplituden modulierte Signale werden vollständig überdeckt. Vor allem Schaltnetzteile, LED-Leuchtmittel, aber auch Telekommunikationsleitungen (DSL, ISDN) verbreiten einen dichten Störnebel. In der letzten Zeit werden auch sogenannte Powerline-Kommunikationseinrichtungen betrieben. Diese verschärfen die Störsituation weiter, da diese Einrichtungen hochfrequente Signale über das normale Energienetz übertragen. Ein starkes Rauschen auf den verwendeten Frequenzbereichen ist die Folge, weil die verwendeten Leitungen dafür gar nicht geeignet sind und es zu einer starken Abstrahlung auf Grund des Antenneneffekts kommt.

Das Datentelegramm des DCF77 wird mit 1Bit/Sekunde übertragen. Eine vollständige Übertragung dauert ca. 1Minute. Ist in dieser Zeit nur ein Bit falsch empfangen, so muss das Datentelegramm zu mindestens teilweise verworfen werden. Die verwendeten Prüfbits (Parität) erlauben zwar eine Erkennung von Ein-Bit-Fehlern, Zwei-Bit-Fehler können unerkannt bleiben. Eine Korrektur ist auch nicht möglich. In [11] wird, um keine falschen Zeiten zu dekodieren, das Datentelegramm zweimal hintereinander empfangen. Die Folge ist eine Verlängerung der Zeit bis zur ersten Synchronisation. Unter optimalen Bedingungen wird so eine Synchronisation nach min. 2 Minuten und max. 3 Minuten erreicht. Leider sind die Bedingungen nur selten so gut. Neben der Anfälligkeit für Störimpulse haben auch die tageszeitlichen- und jährlichen Schwankungen der Übertragungsbedingen Ein-

fluss auf die Störsicherheit. Funkuhren werden aufgrund der schwierigen Empfangsbedingungen selten synchronisiert. In der Zwischenzeit laufen diese Uhren nur quarzgenau. Sollte die Synchronisation jedoch einmal am Tag erfolgen, ist die Genauigkeit bei Weitem ausreichend. Diese Synchronisation erfolgt dann meist in den Abend- und Nachtstunden, da aufgrund des Fehlens der atmosphärischen D-Schicht ein besserer Empfang der Raumwelle möglich ist. Diese Beobachtungen wurden bei der Realisierung und dem Betrieb von [11] gemacht.

2.1.2 Genauigkeit

Die Genauigkeit des amplitudenmodulierten Signals ist aufgrund der Störimpulse und dem Übertragungsverfahren recht begrenzt. In [11] wird die steigende Flanke des 1. Sekundenimpulses einer neuen synchronisierten Minute als Referenzzeitpunkt verwendet. Kurzzeitige Störungen von unter 100 ms werden nicht erkannt und bewirken Schwankungen der Uhrzeit. Es wurden Schwankungen von bis zu 50 ms beobachtet. Einen Ausweg würde die Bewertung und Mitteilung aller Zeitpunkte der steigenden Flanken der Sekundenimpulse bieten. Trotzdem muss man sagen, dass eine Schwankung von 50 ms natürlich völlig ausreichend für normale Uhren ist. Für die Steuerung von technischen Einrichtungen oder für Spezialzwecke, wie synchrone Blinklichter, ist das jedoch nicht ausreichend.

Da in einem Empfänger zur Demodulation des Phasensignals die Trägerfrequenz erzeugt wird, ist es möglich diese auch auszugeben. Die Trägerfrequenz des DCF77 wird mit den am Standort befindlichen Atomuhren sehr konstant gehalten (10^{-12}). Der Sender dient als Normalfrequenzsender. Somit ist es möglich diese Genauigkeit auch im Empfänger zu erhalten.

2.2 Bekannte Realisierungen zum Empfang

Im vorigen Abschnitt wurde schon von dem hohen Aufwand zum Empfang phasenmodulierter Signale gesprochen. Deshalb sind nur wenige Lösungen bekannt, die dieses tun. Da bei großem Aufwand der Empfang von Zeitmarken über GPS wesentlich genauer ist, wird meist dann das GPS benutzt. Es sind deshalb nur zwei industrielle Anwendungen bekannt, die dieses Signal auswerten. Des weiteren sind auch experimentelle Lösungen publiziert. In den nachfolgenden Abschnitten werden diese vorgestellt.

2.2.1 PZF600

Abbildung 7 PCF600 (aus [12])

Die Firma Meinberg stellt den PZF600 als Eurokarte zum Einsatz in ihren 19“-Gehäusen her. [12]

2.2.2 PZF180pex

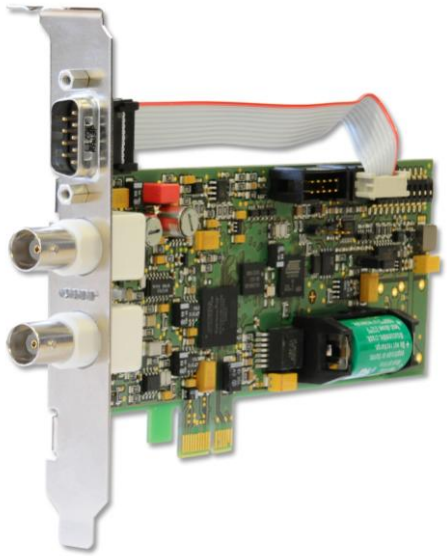


Abbildung 8 PSF180pex (aus [13])

Für den Einsatz im PC-Umfeld ist diese Karte der Firma Meinberg gedacht [13].

2.2.3 DCF77 phase modulation receiver

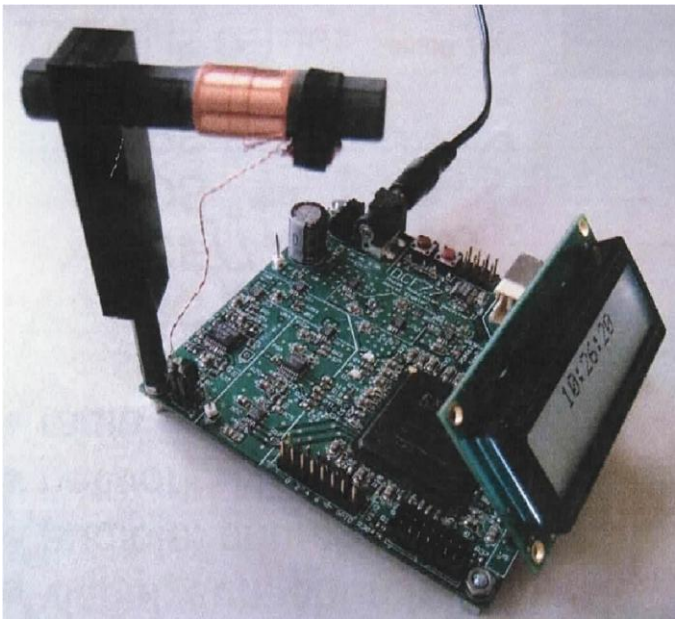


Abbildung 9 FPGA-DCF77-Empfänger (aus [14])

Daniel Engeler beschreibt den Empfang des DCF77-Signals mittels FPGA [14].

2.2.4 Ultra-präziser DSP-basierter DCF77-Zeitsignal-Empfänger

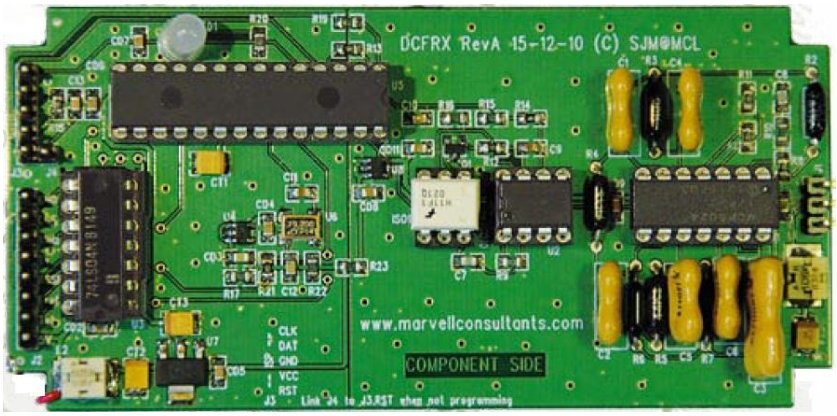


Abbildung 10 DCP-basierter Empfänger (aus [15])

In der Zeitschrift „Elektor 01/2012“ [15] wird ein Empfänger-Bausatz vorgestellt. Dieser basiert auf einem DSP-Mikrocontroller dsPIC33 der Firma Microchip. Dieser Bausatz dient zur Gewinnung eines 10Hz-Referenzsignals. Die Ermittlung der amtlichen Zeit steht hier nicht im Mittelpunkt.

2.3 Schlussfolgerungen für eine Realisierung

Alle im vorherigen Abschnitt genannten Lösungen sind für Uhren sehr aufwendig. Nur die Lösung aus 2.2.4 verspricht das Phasensignal mit relativ wenig Aufwand zu dekodieren. Aber auch dort ist ein spezieller Mikrocontroller mit DSP-Eigenschaften nötig. Für den Einbau in genau gehende und störsichere Funkuhren existiert keine preiswerte Lösung, die das Phasensignal auswertet. Es müssen folgende Anforderungen für eine solche Lösung erfüllt sein:

1. Die Genauigkeit muss besser sein als in den vorhandenen Lösungen, welche das Amplitudensignal auswerten. Im Verlauf der Projektbearbeitung wird die erreichbare Genauigkeit festgelegt, da es sicher einen Kompromiss zum Aufwand geben muss.

2. Die Störsicherheit muss besser als bei vorhandenen amplitudensignalgestützten Funkuhren sein.

3. Der Aufwand sollte geringer sein als Lösungen, welche das GPS-Signal benutzen. Insbesondere sog. „GPS-Mäuse“ sind sehr preiswert erhältlich.

Als weitere Abgrenzungsmerkmale gegenüber GPS-Lösungen sollten eine geringe Leistungsaufnahme und ein weitgehend ungestörter Empfang in Gebäuden, ohne die bei GPS notwendige freie Sicht zu den Satelliten, genannt sein. Einfache „GPS-Mäuse“ liefern auch nur ein serielles Ausgangssignal über USB oder RS232. Für genaue Zeitangaben im Bereich von wenigen Millisekunden ist dieses Format nicht geeignet, da aufgrund der Eigenschaften der seriellen Übertragung es zu Laufzeitschwankungen kommt. Abhilfe schafft ein spezieller 1-Sekundenausgang (1s), dieser ist bei einfachen (billigen) GPS-Geräten jedoch nicht vorhanden.

Aus den genannten Anforderungen ergeben sich folgende Sollkriterien für eine Realisierung:

1. Einsatz eines preiswerten und trotzdem leistungsstarken Mikrokontrollers bei gleichzeitigem geringem Stromverbrauch.

2. Die zu schaffende Lösung sollte für verschiedene Anwendungen anpassbar sein, um eine Amortisation des hohen Entwicklungsaufwands zu erreichen.

3. Verwendung möglichst kostenloser Entwicklungswerkzeuge um die Kosten gering zu halten.

4. Verwendung von Entwicklungswerkzeugen und Prozessoren, bei denen bereits Erfahrungen vorliegen, um den Zeitaufwand für eine Einarbeitung gering zu halten.

5. Möglichst viele Funktionen des Empfängers in der Software unterbringen, um die Kosten gering zu halten.

3 Struktur eines SDR zum Empfang des DCF77

"Ich liebe diesen Plan. Ich bin begeistert daran teil zu haben.
Lasst es uns tun!"
Dr. Peter Venkman in „Ghostbusters“

3.1 Möglichkeiten zur Realisierung des SDR

Aus dem letzten Abschnitt wird klar, dass das zu realisierende Projekt sich am Aufwand (Kosten), der Genauigkeit und der Störsicherheit messen muss. Um dieses zu erreichen, sind konventionelle Empfängerkonzepte wie z. B. das Superheterodyn-Prinzip nicht nutzbar. Der Aufwand wäre zu groß. Weiterhin verbietet sich die Nutzung eines analogen Mischers aufgrund der möglichen Frequenz- und Phaseninstabilität. Eine digitale Verarbeitung der empfangenen Signale möglichst nahe am Eingang verspricht die besten Ergebnisse in Bezug auf den Aufwand.

Soll ein beliebiges analoges Signal mittels Digitaltechnik ausgewertet werden, so ist eine Analog-Digital-Wandlung nötig. Diese Wandlung hat zwei wichtige Qualitätsmerkmale:

1. Abtastrate
   Die Abtastrate (engl. Samplerate) bestimmt, wie oft die Amplitude des Signals pro Zeiteinheit abgetastet wird. Diese Abtastrate wird in Samples per Second (sps) oder Kilosamples per Second (ksps) angegeben.

2. Genauigkeit
   Als Genauigkeit wird die Auflösung des Amplitudenwertes einer Abtastung bezeichnet. Diese Auflösung oder auch Abtasttiefe wird in Bit angegeben.

Beide Merkmale bestimmen, wie genau das analoge Signal im digitalen Bereich zu Verfügung steht. Besonders die Abtastrate hat großen Einfluss auf die zu erwartende Datenmenge und damit auf den für die Auswertung notwendigen Rechenaufwand. Für eine korrekte Auswertung ist die doppelte Nutzfrequenz als Samplefrequenz notwendig. Diese Notwendigkeit wird im Nyquist-Shannon-Abtasttheorem erklärt. Die Genauigkeit (Abtasttiefe) beeinflusst auch die zu erwartende Datenmenge. Da mit jedem zusätzlichen Bit sich die Anzahl der Abtaststufen verdoppelt, haben schon geringe Änderungen großen Einfluss auf die Genauigkeit. Die Anzahl der möglichen Abtaststufen bestimmt den Dynamikbereich der AD-Wandlung. Also den Bereich zwischen dem kleinsten auswertbaren Signal und der Übersteuerung des Empfängers.

In den nächsten Abschnitten soll erläutert werden, welche Möglichkeiten es gibt, die Datenrate und damit den Rechenaufwand gering zu halten.

3.1.1 SDR mit IQ-Mischer

Im vorherigen Abschnitt wurde schon erwähnt, dass die Abtastfrequenz min. 2mal der höchsten Nutzfrequenz entsprechen muss. Da es sonst zur Spiegelung des Eingangssignals an der Samplefrequenz kommt und dieses Spiegelsignal nicht mehr aus dem Datenstrom getrennt werden kann. Um die anfallende Datenmenge zu reduzieren, kann man vor der Digital-Analog-Wandlung das Eingangssignal mittels eines Mischers auf eine niedrigere Frequenz bringen. Verwendet man nicht nur einen Mischer, sondern 2 Mischer und betreibt diese mit um 90° verschobener Frequenz erhalt man einen sog. IQ-Mischer. Diese Mischer haben den Vorteil etwaige Störungen durch Spiegelungen an der Mischfrequenz gut zu unterdrücken. Dieses Prinzip ist weit verbreitet und wurde auch für die ersten Versuche mit dem Empfang des DCF77 verwendet.

Abbildung 11 IQ-SDR

Die Abbildung 11 stellt in Bildmitte den modifizierten „LW-IQ-SDR“ des „Funkamateurs“ dar. Rechts ist eine aktive Antenne mit Ferritstab zu sehen. Links befindet sich eine USB-Soundkarte zur AD-Wandlung. Nähere Beschreibung in [11]. Leider hat das IQ-Prinzip auch Nachteile. Folgende Übersicht stellt die Vor- und Nachteile gegenüber:

Vorteil IQ-Prinzip	**Nachteil IQ-Prinzip**
Reduziert Datenmenge je nach Mischfrequenz bis auf wenige kByte/s	Erhöhter Aufwand (Mischer nötig)
bewehrtes Prinzip	Oszillator für Mischfrequenz nötig
Gute Spiegelfrequenzunterdrückung	2 synchrone AD-Wandler nötig

Die Nachteile des IQ-Prinzips vermeidet das Prinzip der Direktabtastung. Im nachfolgendem Abschnitt wird ein direktabtastender SDR beschrieben.

3.1.2 Direktabtastender SDR

Um den erhöhten Aufwand eines Mischers am Eingang eines SDR zu entgehen, ist ein direkt abtastender Empfänger nötig. Dieser Empfänger wird das Eingangssignal schaltungstechnisch so nah wie möglich an der Antenne in ein Digitalsignal umwandeln. Das Problem ist, dass je nach Frequenz sehr viele Daten anfallen. Da die Frequenz für den DCF77 bei 77500 Hz liegt, ist das Problem für leistungsstarke Mikrocontroller beherrschbar. Ob ein 8-Bit-Mikrokontroller das anfallende Datenvolumen bewerkstelligen kann, wird die weitere Projektbearbeitung ergeben. Wenn von einer 12 Bit tiefen Umwandlung ausgegangen wird und die Abtastrate min. 2 mal der Signalfrequenz entspricht, dann ist von min. 77500 Hz * 2 *1,5Byte = 232,5kByte/s auszugehen. Diese Datenmenge ist für einen Mikrocontroller mit einer max. RAM-Größe von 8kByte problematisch.

Durch verschiedene Maßnahmen muss versucht werden diese Datenrate zu verringern. Auch ist es fraglich, ob eine Abtasttiefe von 12 Bit notwendig ist. Durch geringen schaltungstechnischen Aufwand lässt sich die Abtasttiefe auf 8 Bit verringern. Dadurch wird sich die Datenrate auf 77500 * 2 * 1Byte = 155 kByte/s vermindern. Weiterhin muss in den internen Algorithmen des Mikrocontrollers mit nur 1 Byte gearbeitet werden. Dieses bringt den Vorteil, dass die für die Signalverarbeitung wichtige MAC-Operation (a=a+b*c) sehr schnell ausgeführt werden kann, da der 1 Byte-Multiplikationsbefehl von der Hardware direkt ausgeführt wird. Der sich dadurch einengende Dynamikbereich muss durch Hardwaremaßnahmen kompensiert werden.

Trotz der Probleme mit der großen Datenrate soll versucht werden einen SDR ohne Hardwaremischer aufzubauen. Da das eigentliche Nutzsignal nur wenige Kilohertz Bandbreite beansprucht, scheint das möglich zu sein.

3.2 Aufbau eines direktabtastenden SDR

Ein direktabtastender SDR benötigt wenig Hardware. Die meisten Operationen werden in der Software ausgeführt. Nur ein Filter für die Einhaltung des Nyquist-Shannon-Abtasttheorems ist nötig. Weiterhin werden verschiedene Anpassstufen für die Kopplung zwischen Antenne und Mikrocontroller nötig. Da die Abtasttiefe auf 1Byte beschränkt bleiben soll, ist noch ein in der Verstärkung regelbarer Vorverstärker nötig um eine Übersteuerung des Empfängers bei wechselnden Empfangsbedingungen zu vermeiden. Folgende Blockdarstellung zeigt den Entwurf:

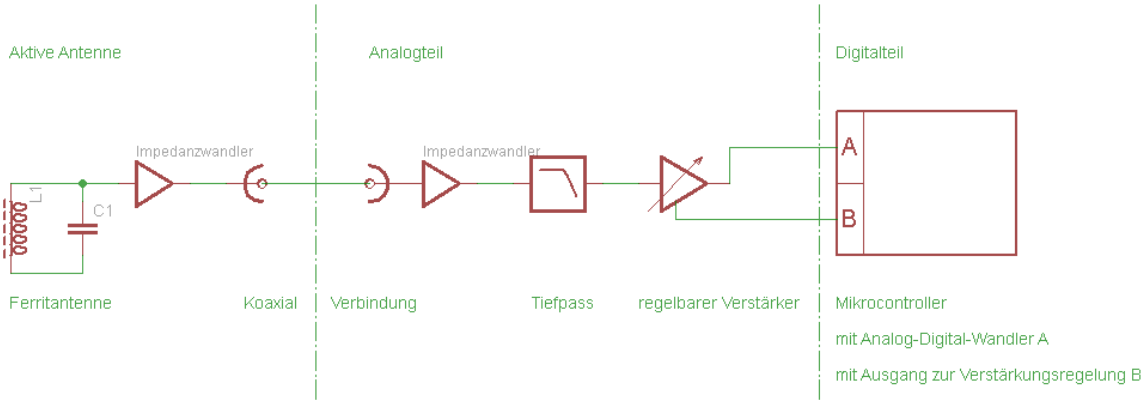


Abbildung 12 Blockbild direktabtastender SDR

Die abgesetzte Antenne ist für eine kontrollierte Empfangsbedingung nötig. Viele Störungen gehen vom Empfänger selbst aus und beeinflussen das Eingangssignal. Da das hier vorgestellte Prinzip dem eines Geradeausempfängers entspricht, kommt es leicht zu einer Schwingneigung, wenn Teile des verstärkten Signals auf den Eingang zurückkoppeln.

3.2.1 Direktabtastender SDR mit Unterabtastung

Selbst bei einer Abtasttiefe von 1 Byte ergeben sich min. 155 kByte/s Datenrate, welche der Mikrocontroller bewältigen muss. Wie könnte man diese enorme Datenrate verringern? Folgender Ansatz zeigt eine Lösung auf: Die hohe Datenrate ergibt sich aus der Anwendung des Nyquist-Shannon-Abtasttheorems. Dieses Theorem stimmt aber nur bei einem Basisbandsignal. Also einem Signal von 0 Hz bis zur größten auftretenden Frequenz. Genaugenommen muss die Abtastrate aber nicht der doppelten Eingangsfrequenz, sondern nur der doppelten Bandbreite des Eingangssignals entsprechen, da keine Signale vorhanden sind, die das Nutzsignal stören können. Man kann den Tiefpass aus Abbildung 12 durch einen Bandpass ersetzten und die Abtastrate und damit die Datenrate drastisch verringern. Natürlich steigt der Aufwand in der Hardware, aber durch die Verwendung von Operationsverstärkern ist das überschaubar.

Folgende Abbildung zeigt das Blockbild:

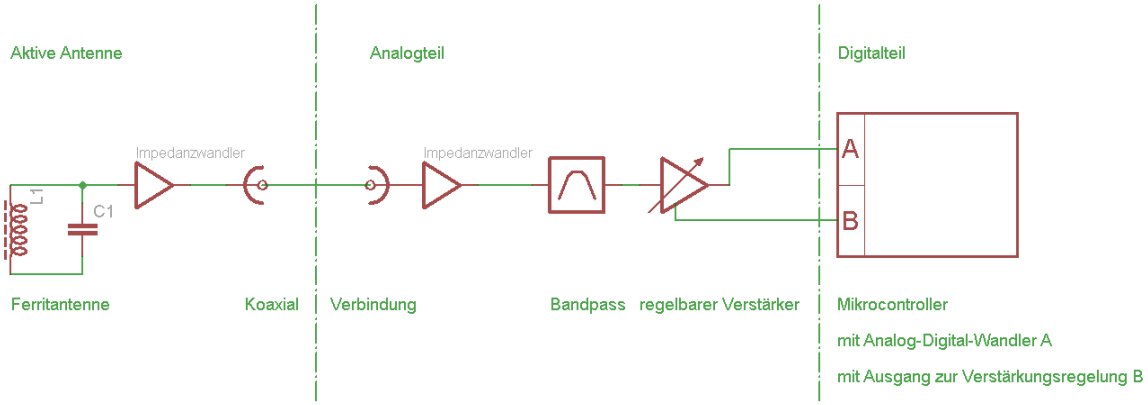


Abbildung 13 Blockbild direktabtastender SDR mit Bandpass

Ein Teil der Aufgaben der Software, nämlich die Selektion der Nutzfrequenz, wird hier der Hardware übergeben. Da die als Schwingkreis realisierte Antenne bereits eine Vorselektion übernimmt, ist die Selektion schon teilweise in der Hardware realisiert. Im Verlauf der Projektbearbeitung wird ein solcher Empfänger konstruiert und aufgebaut.

3.2.2 Der Dynamikbereich eines SDR

Viele in der Literatur beschriebenen SDR kommen ohne den regelbaren Verstärker aus. Diese besitzen aber auch eine Abtasttiefe von 16 Bit. Dieses ergibt aus 16 * 6,02 einen Dynamikumfang von ca. 96dB. Da mit 8 Bit (48 dB) ausgekommen werden soll, sind einige Überlegungen zur Empfangssituation eines Langwellenempfängers nötig. Folgende Bedingungen sind gegeben:

1. Die Nutzfrequenz ist 77500 Hz und es ergibt sich damit eine Wellenlänge von 300000km/s / 77500 1/s = 3,87 km. Damit ist eine Dipolantenne mit Wellenlänge/2 illusorisch. Der Einsatz einer Ferritantenne ist notwendig.

2. Der Empfänger soll in ganz Europa einsetzbar sein.

3. Aus 2. ergibt sich, dass kleinste Eingangssignale noch verarbeitet werden müssen. Die untere Grenze stellt das Rauschen der Antenne dar. Dieses Rauschen gehorcht in etwa einer 1/f-Abhängigkeit. Bei 77500 Hz ist dieses Rauschen bei Weitem größer als die Rauschanteile des Empfängers.

4. In der Nähe des Senders Mainflingen darf es nicht zu Übersteuerungen kommen.

Daraus ergibt sich, das um einen optimalen Empfang zu gewährleisten, das atmosphärische 1/f-Rauschen so weit verstärkt werden muss, dass der anzusteuernde AD-Wander in seinen niederwertigen Bits Daten liefert. Da das Nutzsignal ein Rauschsignal (PZF) darstellt, kann es möglich sein, dass darin dekodierbare Daten enthalten sind. Durch die ge-

ringe angestrebte Auflösung von 8 Bit ergibt sich ein Dynamikumfang von 48 dB. Das bedeutet, es ist nur ein Verhältnis von kleinster zu größter Eingangsspannung von 1:256 möglich. Um Bedingung 4 zu realisieren, wird eine Reglung der Eingangsempfindlichkeit vorgeschlagen. Vor allem für die Auswertung des Amplitudensignals ist das nötig. Eine konkrete Realisierung des hier vorgestellten Prinzips wird im nachfolgenden Kapitel beschrieben.

4 Hardware des SDR

„Mehr Input, Stephanie, mehr Input“

Nummer 5 in „Nummer 5 lebt“

4.1 Der Analogteil des DCF77-SDR

4.1.1 Aufgaben des analogen Teils eines SDR

In einem direktabtastenden SDR muss der analoge Teil des Empfängers mindestens 2 Aufgaben erfüllen:

1. Anpassung der Antenne oder des Eingangs an die digitale Hardware.
2. Filterung des Eingangssignals zur Einhaltung des Nyquist-Shannon-Abtasttheorems

Die Aufgabenstellung für das DCF77-SDR-Projekt beinhaltet zusätzlich auch noch die möglichst preiswerte und energieschonende Realisierung. Aus diesem Grund kommt wie im vorherigen Kapitel beschrieben noch die:

3. Regelung zum Ausgleich der Empfangsfeldstärkenschwankungen

hinzu. Dieses wird nötig um mit 8 bit Abtasttiefe auszukommen.

Weiterhin müssen bei der Entwicklung der analogen Hardware folgende Probleme beachte werden:

A. Störstrahlungen des digitalen Teils des Empfängers auf den Eingang müssen verhindert werden.
B. Rückkopplungen des analogen Ausgangssignals auf den Eingang würde zu Schwingneigung führen.
C. Es darf keine frequenzabhängigen Laufzeitunterschiede des Eingangssignals innerhalb der Bandbreite des Nutzsignals geben, da diese die Demodulation des Nutzsignals erschweren würden.

4.1.2 Realisierung der analogen Hardware

Durch die im Projekt [11] gewonnenen Erfahrungen wird die abgesetzte Antenne als Lösung des Problems (A) beibehalten. Diese Antenne ist eine aktive Antenne. Der aktive Teil der Antenne ist als Kollektorstufe ausgeführt und dient zur Anpassung des hochohmigen Ferritstabschwingkreises an die 50 Ohm des Kabels zum eigentlichen Empfänger. Der Arbeitswiderstand dieser Stufe befindet sich dabei im Empfänger. Dadurch wird eine Fernspeisung über ein Koaxialkabel möglich. Das Prinzip dieser Antenne geht auf [17] zurück. Durch diese Antenne wird auch ein Teil der Aufgabe (1) und (2) gelöst.

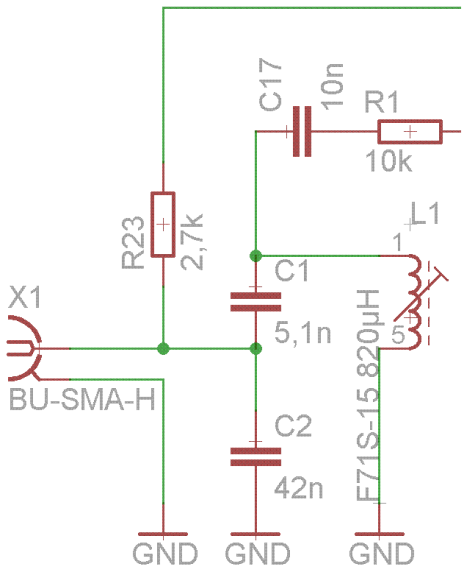


Abbildung 14 Eingangsstufe

Der 50-Ohm-Eingang des eigentlichen Empfängers ist mittels kapazitiven Spannungsteilers an einen Schwingkreis angekoppelt. Dadurch wird eine Spannungserhöhung von ca. 16 dB erreicht. Dieses soll die Empfindlichkeit auf Einstreuungen verringern. Zur Überprüfung der Eingangsstufe wurde diese mit LTspice simuliert:

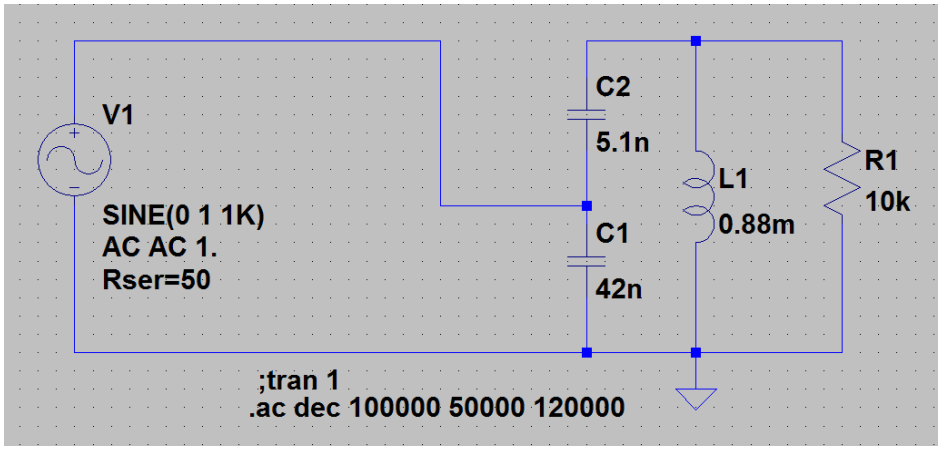


Abbildung 15 Kapazitive Ankopplung

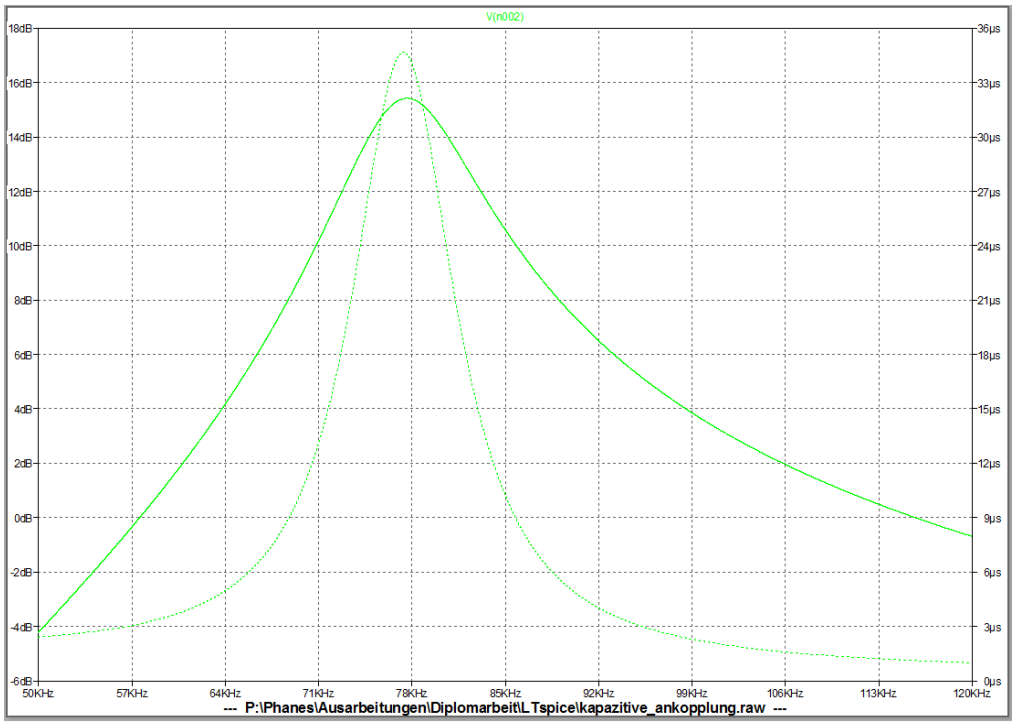


Abbildung 16 Kapazitive Ankopplung - Diagramm

Weiterhin besteht die Schaltung aus 3 Filterstufen, welche als Bandfilter ausgeführt sind. Diese Filter lösen zusammen mit dem Eingangsschwingkreis der Ferritantenne und dem Bandfilter bestehend aus C1, C2 und L1 die Aufgabe der Filterung (2). Dass drei Filter benötigt werden liegt an zwei Gründen:

1. Alle Frequenzen, welche nicht durch das Nyquist-Shannon-Abtasttheorem abgedeckt werden, müssen bei 8-bit-Abtastung um min. 48 dB gedämpft werden. Ein einzelner Filter wäre dabei überfordert.

2. Um die geforderte Bandbreite bei möglichst steilen Filterkurven zu erreichen, erschien der Einsatz sinnvoll.

Die Mittenfrequenzen der Filter wurden auf 77500 Hz, 76860 Hz und 78150 Hz festgelegt. Diese Splittung soll der Bandbreite des Eingangssignals Rechnung tragen. Dieses ermöglicht wahlweise die Realisierung der AD-Wandlung als Unterabtastung oder als Abtastung mit der min. 2 fachen Nutzfrequenz. Falls die Software die volle Abtastrate bewältigen kann, werden in einem weiteren Entwurf die Bandpassfilter durch Tiefpassfilter ersetzt. Die Filter wurden mit OPVs realisiert. Beispielhaft wird in Abbildung 17 der Filter für die mittlere Mittenfrequenz dargestellt.

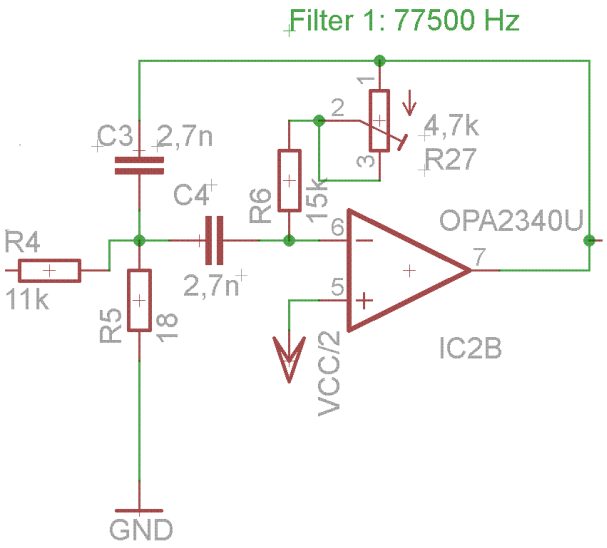


Abbildung 17 Filter 77500 Hz

Zur Berechnung der Filter wurde das Programm „Aktivfilter 2.2“ eingesetzt. Nach der Berechnung erfolgte eine Simulation mit LTspice. Dabei kam ein anderer OPV zum Einsatz, da im ersten Entwurf benutzte Type OPA2340 nicht das benötigte Bandbreiteverstärkungsprodukt lieferte.

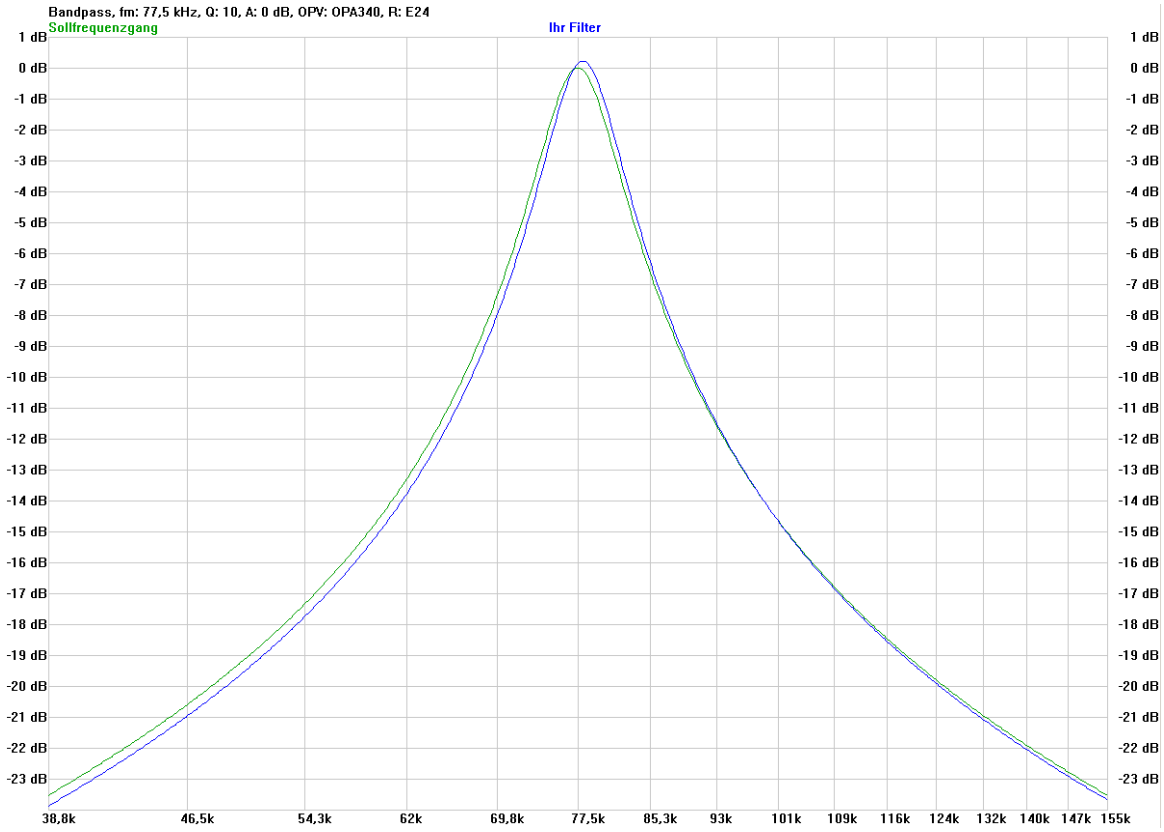


Abbildung 18 Filter 1 77,5kHz

In den Aufgaben des analogen Teils des SDR wurde auch die Regelung zum Ausgleich der Feldstärkenschwankungen genannt (3). Dieses wird ebenfalls durch 3 OPVs realisiert, welche eine mittels Analogschalter umschaltbare Gegenkopplung besitzen.

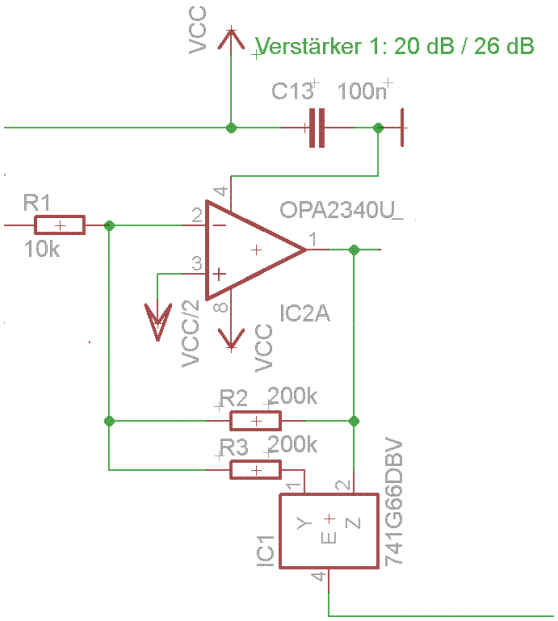


Abbildung 19 Regelbarer Verstärker 1

Durch 3 vom steuernden Mikrocontroller ausgehende Leitungen können 8 verschiedene Verstärkungen eingestellt werden. Im ersten Entwurf sind Gesamtverstärkungen zwischen 66 dB – 108 dB in einem 6 dB Abstand möglich. Ob diese hohe Verstärkung und die feine Abstufung nötig sind, wird im weiteren Projektablauf entschieden.

Zur Anpassung (Aufgabe 1) an den Mikrocontroller ist eine weitere Stufe nötig, da bei hohen Abtastfrequenzen der Eingangswiderstand der AD-Wandlers sinkt. Zusammen mit einem OPV, welcher die halbe Betriebsspannung als Bezugspotential bildet, ergibt sich ein Aufwand von 8 OPVs. Diese werden als Doppel-OPV eingesetzt. 4-fach OPVs zu benutzen erschien aus Stabilitätsgründen nicht vorteilhaft.

In Anlage 3 ist die realisierte Gesamtschaltung hinterlegt.

In Anlage 4 sind die Filterschaltungen als LTspice-Ausdruck und die Simulationen der Frequenzgänge der einzelnen Filter sowie der simulierte Gesamtfrequenzgang hinterlegt.

4.2 Der digitale Teil des DCF77-SDR

4.2.1 Anforderungen an den digitalen Teil des DCF77 SDR

Da in einem SDR viele Funktionen eines Empfängers in der Software realisiert werden, ist es unabdingbar, dass die zugrunde liegende Rechnerhardware leistungsfähig genug ist. Es muss ein Kompromiss zwischen Leistungsfähigkeit, Energieverbrauch, Verfügbarkeit und Vorhandensein von preiswerten Softwarewerkzeugen geschlossen werden. Auch sind die Kosten und die Zeit für die Einarbeitung zu berücksichtigen.

Zur Reduktion der Kosten für das Projekt wurde entschieden, dass die Digitalhardware für mehrere andere Anwendungen einsetzbar sein sollte. Dadurch wird eine höhere Stückzahl an Leiterplatten erreicht, welche die Kosten pro Einzelplatine verringern. Aus diesem Grund sind mehrere serielle Schnittstellen, ein SD-Slot und ein USB-Anschluss zur preiswerten Stromversorgung nötig. Diese Leiterplatte wird z. B. als Platine für Lötübungen in Lehrgängen der Erwachsenenqualifikation verwendet. Dem ist es auch geschuldet, dass als Hauptanzeige eine 7-Segment-LED-Anzeige verwendet wird.

Für die eigentliche Realisierung der DCF77-Uhr mit Phasensignalauswertung sind folgende Anforderungen gegeben:

1. Preiswerter, leicht verfügbarer Mikrocontroller mit kleiner Stromaufnahme und geringem Einarbeitungsaufwand bei leichter Verfügbarkeit der Softwarewerkzeuge.

2. Der Mikrocontroller sollte einen AD-Wandler besitzen, welcher Abtastraten bis zum 2 - 4 fachen der Nutzfrequenz ermöglicht (> 200 kHz).

3. Da die Nutzfrequenz von 77500 Hz einen „krummen“ Wert darstellt, ist eine konfigurierbare Taktaufbereitung im Mikrocontroller wünschenswert. Es wird dadurch kein spezieller Quarz benötigt.

4. Mehrere leistungsstarke Ausgänge des Mikrocontrollers sollten vorhanden sein, um LEDs direkt treiben zu können.

5. Der Mikrocontroller sollte mehrere Timer/Counter besitzen.

6. Die Leiterplatte sollte in ein Standardgehäuse passen.

7. Die verwendeten Bauelemente müssen ohne spezielle Lötverfahren montierbar sein.

4.2.2 Bauelemente-Auswahl

Aufgrund langjähriger Erfahrungen mit den AVR-Mikrocontrollern der Firma Atmel war die Entscheidung zur Auswahl des Mikrocontrollers relativ einfach. Zusätzlich kommt hinzu,

dass die neueren Typen der XMEGA(A)-Serie verschiedene Eigenschaften besitzen, welche man bei anderen Controllern vergeblich sucht. Als Eigenschaften seien genannt:

- Hohe CPU-Taktfrequenz von max. 32 MHz
- Mehrere 16 Bit-Timer/Counter
- Leistungsfähige AD-Wandler mit bis zu 12-bit-Wandlungstiefe bei einer Samplefrequenz von bis zu 1 MHz
- 4 DMA-Kanäle zur Datenübertragung unter Umgehung der CPU
- Event-System zur Hardware gestützten Ereignisverwaltung

Auf Grund dieser Eigenschaften wurde der:

ATXMEGA128A4

ausgewählt. Besonders die letzten beiden genannten Eigenschaften machen es möglich hohe Abtastfrequenzen zu realisieren und die dabei anfallenden Daten, ohne CPU-Belastung in den Speicher zu transportieren. Bei 8 Bit-Abtasttiefe und der Nutzfrequenz von 77500 Hz entsteht eine Datenrate von min. 155 kByte/s.

Im vorherigen Abschnitt wurde schon angedeutet, dass die Erzeugung von 77500 Hz mit Standardquarzen nicht leicht ist. Der genannte Mikrocontroller unterstützt eine flexible Taktkonfiguration. Folgende Überlegungen wurden zur Taktversorgung angestellt:

- Der Mikrocontroller hat eine max. CPU-Frequenz von 32 MHz.
- 310000 Hz / 4 = 77500 Hz.
- Ein externer Quarz darf eine Frequenz von 0,4 – 16 MHz haben.
- Der Mikrocontroller besitzt eine PLL-Schaltung, welche den Quarztakt um den Faktor 2-31 erhöhen kann.

Daraus ergibt sich folgende Lösung. Einsatz seines:

4MHz-Quarzes

an den Pins XTAL1 und XTAL2. Durch die PLL-Schaltung lässt sich der Takt mit 31 multiplizieren. Es ergibt sich eine Haupttaktfrequenz von 124 MHz. Diese wird durch 4 geteilt und dann der CPU zugeleitet. Die Timer, der AD-Wandler und andere Baugruppen arbeiten dann auch mit 31 MHz. Verschiedene Baugruppen arbeiten sogar mit 124 oder 61 MHz. Eine Abtastung des Nutzsignals mit ganzzahligen Teilen der Nutzfrequenz wird dadurch möglich. Dieses macht eine Fensterung der vom AD-Wandler gelieferten Werte mit

Rechteckfenstern ohne störende Mischprodukte möglich. Als Nachteil des Taktes von 31 MHz ist eine verschenkte Rechengeschwindigkeit von ca. 3% zu nennen.

Für die Demonstration der dekodierten Zeit wird ein 4-stelliges-7-Segment-LED-Display vorgesehen. Nur die gemeinsamen Anoden werden über externe Transistoren getrieben. Die Kathoden werden direkt durch den Mikrocontroller angesteuert. Eigene Versuche haben eine mehr als ausreichende Helligkeit ergeben.

Der USB-Anschluss dient nur zur Stromversorgung. Deshalb wurde aus Preisgründen nicht die Standardbeschaltung eines USB-Anschlusses verwendet. Trotzdem ist kann man für Laborzwecke einen USB-Treiber im Mikrocontroller implementieren. Die dafür notwendigen Mikrocontroller-Pins sind am USB-Konnektor angeschlossen.

Die Taster sollen zur Bedienung der DCF77-Demontrationsuhr dienen. Es ist an eine Umschaltung des Datumsformats und eine Helligkeitsregelung gedacht.

Die Schaltung sowie das daraus resultierende Board wurden mit Eagle 6.4 gezeichnet. In Anlage 5 sind die Zeichnungen dazu hinterlegt. In Abbildung 22 ist die realisierte Schaltung abgebildet.

5 Software des SDR

"Haben sie die Schlüssel vom Patentamt geklaut?"

Harry Stamper in "Armageddon - Das Jüngste Gericht"

5.1 Grundlagen und Aufgaben der Software

5.1.1 Verwendete Software- und Hardwaretools

Für den verwendeten Mikrocontroller gibt es mehrere verschiedene Softwareentwicklungstools. Als freie Software stehen mehrere Versionen des AVR-Studios zur Verfügung. In diesem Projekt wurde die Version 4.19 zusammen mit dem WinAVR-V20100110-Compiler verwendet. Als Programmieradapter kam der AVR-ISP-MKII zum Einsatz. Da bei diesem Projekt große Datenmengen anfallen, ist es nötig ein Tool zur schnellen grafischen Ausgabe einzusetzen. Die Lösung dafür ist das Projekt „JAVA-Oszi“ [18]. Dieses Tool zeigt serielle übertragene Daten grafisch in mehreren Charts an.

Für den reinen Entwicklungs- und Debugbetrieb wird als Plattform das Board „Xmega Webserver Board V1.0“ der Firma Köple Engineering verwendet. Zusätzlich wird noch ein Pegelwandler für die RS232-Schnittstelle verwendet.

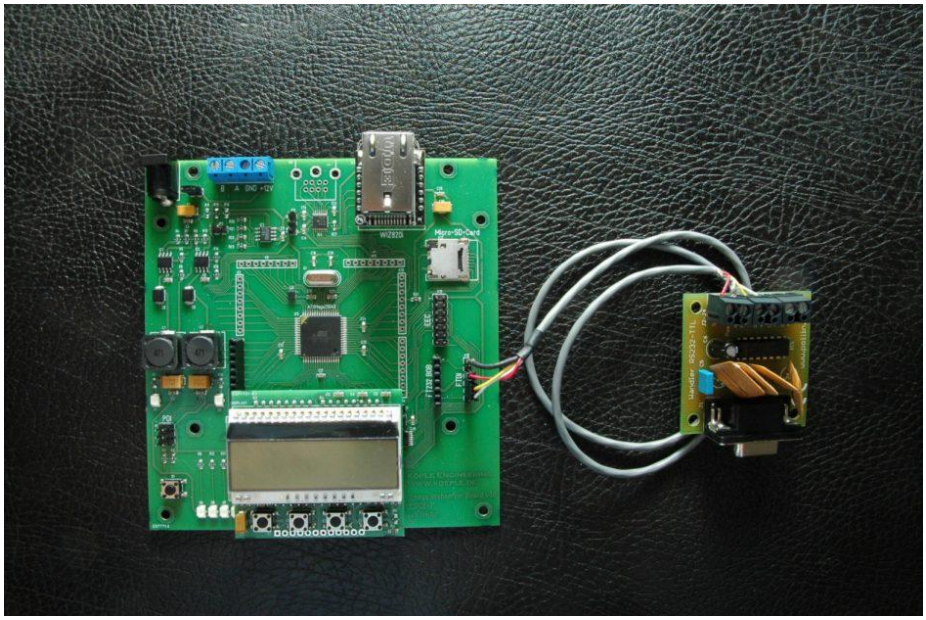


Abbildung 20 Xmega Webserver Board

5.1.2 Aufgaben der Software

Die zu entwickelnde Software für den Mikrocontroller entscheidet maßgeblich über die Funktion des Projektes. Da bei Projektbeginn noch keine Erfahrungen im Bereich Signalverarbeitung mittels Mikrocontroller bestand, wurde die Analoghardware (konkret die Filter) so gestaltet, dass auch mit geringeren Abtastraten gearbeitet werden kann. Trotzdem wurde versucht die Software so zu gestallten, dass eine Abtastrate von min. 155 kHz erreicht werden kann. Bei dieser Abtastfrequenz fällt alle 6,45 µs ein neuer Bytewert an. Da der Controller mit 31 MHz arbeitet, hat der Controller genau 200 Takte Zeit das Byte zu verarbeiten. Wird mit höheren Abtastraten gearbeitet (z. B. 310 kHz) verschärft sich die Probleme noch weiter. Der Hauptaufwand bei der Softwareentwicklung wird also die Realisierung der schnellen AD-Wandlung mit nachfolgender Signalverarbeitung sein:

1. Aufgabe: Schnelle AD-Wandlung mit wenig Rechenaufwand durch die CPU.

Trotz des hohen Aufwands im Analogteil (3 OPV's) zur Signalfilterung ist eine weitere Filterung des Eingangsdatenstroms notwendig. In [2] ist eine Bandbreite des DCF77-Signals von min. 1292 Hz genannt. Um Störsignale auszublenden, sollte dieser Wert zu mindestens in dieser Größenordnung erreicht werden. Weiterhin ist die hohe Datenrate von bis zu 310 kByte/s nicht mit einer 32 MHz CPU zu beherrschen. Diese Datenrate muss reduziert werden:

2. Aufgabe: digitale Filterung, sowie Dezimierung der Datenrate.

Nach der Filterung liegt das Signal dezimiert vor. Trotzdem sind die eigentlichen Nutzdaten noch nicht gewonnen. Es ist die Demodulation des Signals vorzunehmen. Sowohl der Amplituden- als auch der Phasenanteil muss dabei berücksichtigt werden:

3. Aufgabe: Amplituden und Phasendemodulation.

Nach der Demodulation kann der Phasenanteil den Korrelatoren zugeführt werden. Dort werden die Daten mit der PZF verglichen. Pro Sekunde liefern die Korrelatoren 1 Bit. Der Amplitudenanteil dient zur Steuerung der regelbaren Verstärker:

4. Aufgabe: Kreuzkorrelation mit PZF-Signalfolge.

Liefern die Korrelatoren 10mal eine logische 1, dann wurde ein Minutenbeginn erkannt. Bitnummer 10 kann dann als erstes Bit dem eigentlichen Zeitdekoder übergeben werden:

5. Aufgabe: Decodierung der von den Korrelatoren gelieferten Bits zur Generierung der aktuellen Zeit.

Weiterhin sind folgende Aufgaben noch zu erledigen:

6. Aufgabe: Anzeige der aktuellen Zeit.

7. Aufgabe: Initialisierung der Mikrocontrollerhardware sowie der angeschlossenen Baugruppen.

5.2 Realisierung

5.2.1 Struktur der Software

Schon bei Projektbeginn wurde klar, dass die schnelle AD-Wandlung und die digitale Signalverarbeitung die schwierigsten Probleme des gesamten Projekts sind. Um im späteren Projektverlauf mit Tiefpassfiltern und ohne Unterabtastung auskommen zu können, ist eine Abtastrate von min. 155 kHz (2 fache Nutzfrequenz) anzustreben. Es wurde, um gewisse Reserven zu haben, eine Abtastrate von 310 KHz (3 fache Nutzfrequenz) gewählt. Das bedeutet, es fallen bei 8 Bit Auflösung 310 KByte/s an Daten an. Anders ausgedrückt: alle 3,22 µs muss ein Byte verarbeitet werden. Mit herkömmlichen Mikrocontrollern ist das nicht zu bewerkstelligen. Allerdings bieten die Mikrocontroller der ATXMEGA-Serie eine spezielle Hardware an, die auch solche Datenraten bewältigen können. Diese Hardware ist der interne DMA-Controller und das Event-System. Zusammen mit den programmierbaren Zeitgebern ist die 1. Aufgabe wie folgt lösbar:

1. Der Timer1 wird auf die Zielabtastfrequenz eingestellt. Da das System mit einem Takt von 31 MHz arbeitet, muss der Timer1 alle 100 Takte ein Ereignis (kein Interrupt) auslösen.

2. Dieses Ereignis 1 startet den AD-Wandler. Nach ca. 2,5 µs Umsetzzeit wird der AD-Wandler seinerseits ein Ereignis 2 auslösen.

3. Dieses Ereignis 2 wird der Kanal0 des DMA-Controllers entgegen nehmen und das entstandene Byte in einem Buffer1 übertragen. Gleichzeitig werden die Zieladresse des Buffers1 und ein Bytezähler inkrementiert. Die Übertragung geschieht ohne Benutzung der CPU.

4. Nach einer noch festzulegenden Anzahl (Fenstergröße) dieser Timer1/AD-Wandler/DMA-Operationen ist ein Datenblock eingelesen. Zu diesem Zeitpunkt wird der DMA-Kanal0 umgeschalten auf DMA-Kanal1, dabei wird dann ein Buffer2 verwendet. Weiterhin wird ein Interrupt ausgelöst.

5. Die CPU kann jetzt den Buffer1 bearbeiten, da die weiteren Daten über DMA-Kanal1 und Buffer2 aufgenommen werden.

6. Beide DMA-Kanäle und Buffer1 und Buffer2 werden abwechselnd benutzt.

Diese dargestellte Datenaquise ermöglicht es, die volle Umsetzrate der AD-Wandler bis hinauf in den MHz-Bereich auszunutzen. Natürlich müssen die anfallenden Daten auch bearbeitet werden. Die 2. Aufgabe muss das realisieren. Dabei sind zwei Teilaufgaben zu bearbeiten: Filterung und Dezimierung der Datenrate. Für diese kombinierte 2. Aufgabe bietet sich ein Verfahren an, welches beide Teilaufgaben löst: Es wird in einem ersten Schritt eine diskrete Fouriertransformation verwendet (DFT). Dabei wird aber nicht der ganze Frequenzbereich bearbeitet, sondern nur eine Spektralkomponente. Diese bildet jeweils einen Bufferinhalt als 2 Werte im Frequenzbereich ab. Der Buffer stellt sozusagen das Fenster (Windows) der DFT dar. In einem zweiten Schritt wird die Spektralkomponente wieder in den Zeitbereich zurücktransformiert. Dabei wird Folgendes erreicht:

1. Filterung der Eingangsdaten auf einen Bandbreitenwert von Abtastfrequenz durch Fenstergröße

2. Dezimierung der Datenrate auf Datenrate mal 2 durch Fenstergröße

Diese Art von Verarbeitung ist als Görtzel-Algorithmus [19] bekannt.

Nach dem Görtzel-Algorithmus liegen die Daten dezimiert und als I (in phase) und Q (quadrature) Anteil vor. Um den Phasen und Amplituden-Anteil zu berechnen, sind zwei Operationen nötig: Phase = arctan(Q/I) und Amplitude = sqrt(I^2 + Q^2). Dieses soll mittels eines Cordic-Algorithmus [21] geschehen. Da der die Datenaquise startende Timer0 (1. Aufgabe) nur mit der Genauigkeit des CPU-Taktes läuft, wird die Phase, auch ohne Modulation, sich stetig verändern. Es muss also zu einer Kompensation des Frequenzfehlers des CPU-Taktes kommen. Dabei sind zwei Möglichkeiten angedacht:

1. Einsatz eines spannungsgesteuerten Oszillators, um in einer Art PLL eine Frequenzkonstanz zu erreichen.

2. Durch laufendes Umprogrammieren der Timer1-Zeitkonstanten eine Art Frequenzregelschleife (FLL) zu erzeugen.

Nach dem Cordic-Algorithmus liegen die Phase und die Amplitude des Nutzsignals vor. Von dem Start der Datenaquise bis zum Cordic- Algorithmus bildet sich eine Ereigniskette, welche nicht durchbrochen werden darf. Die Rechenzeit der einzelnen Algorithmen muss sich danach richten. Besonders der Cordic-Algorithmus ist dabei schwierig zu realisieren. Alle nachfolgenden Aufgaben 4, 5 und 6 können und müssen asynchron zu den Aufgaben 1, 2 und 3 ablaufen. Dazu dient ein weiterer Timer. Timer0 wird z. B. alle 100µs einen Interrupt auslösen. Daraus wird die interne Zeit (6. Aufgabe) gebildet. Weiterhin startet dieser Interrupt auch jeweils einen Korrelator (4. Aufgabe). Wie viele Korrelatoren gestartet werden, muss im Laufe der Projektbearbeitung noch ermittelt werden. Optimal wären 10000. Leider würde das die Rechenzeit und vor allem den RAM des Mikrocontrollers bei Weitem überlasten. Mit H des Amplitudensignals ist der Sekundenbeginn grob feststellbar. Nach genau 200 ms startet die PZF und die Korrelatoren liefern sinnvolle Werte. Jeweils am Sekundenbeginn wird das für die 5. Aufgabe ermittelte Bit in ein 50 Bit

großes Schieberegister eingeschoben. Am Minutenende wird daraus die aktuelle Zeit ermittelt und in einen Anzeigepuffer kopiert.

Die 7. Aufgabe wird nur nach dem Reset ausgeführt.

5.2.2 Interrupt- und Zeitregime

Der eingesetzte Mikrocontroller hat mehrere Aufgaben quasi gleichzeitig zu erfüllen. In Betriebssystemen wird das durch einen Scheduler erreicht. Leider lassen die beengten Verhältnisse in einem Mikrocontroller nicht immer ein solches Betriebssystem zu. Auch wird vom Betriebssystem viel Rechenleistung verbraucht. Deshalb wird ohne Betriebssystem gearbeitet. Um trotzdem mehrere Aufgaben gleichzeitig erfüllen zu können, ist das Interrupt- und Eventsystem intensiv zu nutzen. Folgende Interrupts und Ereignisse werden verwendet:

DMA-Interrupt:

- Der Takt der CPU wird auf 31 MHz festgelegt
- Ereignis 1 wird ausgelöst vom Timer1. Dieses Ereignis startet den AD-Wandler 310000 mal pro Sekunde.
- Ereignis 2 wird max. 2,5 µs nach AD-Wandlerstart (Umsetzzeit) ausgelöst.
- Ereignis 2 startet die Übertragung eines Bytes in den Hauptspeicher.
- Nach 200 Übertragungen (Fenstergröße und Dezimierung) wird ein mittelpriorisierter Interrupt ausgelöst.
- Dieser Interrupt wird alle ca. 20000 Takte ausgeführt. Kleine Schwankungen liegen an der unterschiedlichen Umsetzzeit des AD-Wandlers. Es ergibt sich eine Interruptfrequenz von 1,55 KHz.

Zeit-Interrupt:

- Alle 100 µs wird der Zeitinterrupt ausgeführt. Er hat die Aufgabe die interne Zeit des Mikrocontrollers zu takten. Es ergibt sich also eine kleinste darstellbare Zeit von 0,1 ms. Weiterhin wird dieser Interrupt zum Starten der Korrelatoren und dem Multiplexen einer LED-Anzeige verwendet.
- Da dieser Interrupt höchst priorisiert ist, muss er möglichst schnell abgearbeitet werden.
- Alle 3100 Takte wird dieser Interrupt ausgelöst. Eine Interrupt-Frequenz von 10 kHz ist die Folge. Da dieser Interrupt zwischen 6 und 7 mal pro DMA-Interrupt ausgelöst wird, ist auf die Kürze dieser Routine zu achten. In den meisten Inter-

ruptfällen wird wenig zu tun sein. Dazu gehört das Inkrementieren eines Millisekunden-Zählers und das Aktualisieren der Ausgabepuffer der LED-Anzeige sowie die Aktualisierung eines Korrelators. Im Durchschnitt wird mit 250 Takten gerechnet. Daraus resultiert, dass von den verfügbaren 20000 Takten des DMA-Interrupts 7 mal 250 = 1750 abzuziehen sind.

Aus den vorhergehenden Betrachtungen ergibt sich folgende Zeitbedingung: Es stehen für die Filterung/Dezimierung und dem Cordic-Algorithmus ca. 18250 Takte entsprechend ca. 0,588 ms zur Verfügung. Für den Görtzel-Algorithmus sind 4 Multiplikationen und 4 Additionen pro Abtastwert angegeben. Da eine Multiplikation/Addition 2 Takte dauert und für Kopieraufgaben noch mal 8 Takte einzuplanen sind, ergibt sich bei einer Fenstergröße von 200 eine Taktanzahl von 16 mal 200 = 3200 Takte. Dieser Wert sollte aber großzügig bewertet werden, da im Hintergrund die DMA-Operationen den Datenbus und den RAM stark belasten. Leider konnten noch keine Erfahrungen über diese Busbelastung gemacht werden. Trotzdem stimmt die Überschlagsrechnung von 3200 zu 18250 Takten optimistisch das geforderte Zeitregime einzuhalten.

5.2.3 Realisierung der Software

Für die Programmentwicklung wird das AVR-Studio 4.19 und als Toolchain WinAVR V20100110 eingesetzt. In der Anlage 6 ist der vollständige Quellcode des bisher realisierten Projektes aufgeführt. Im Projekt fehlen noch die 3. und 4 Aufgabe (Phasendemodulation und Kreuzkorrelation). Die 5. und 6. Aufgabe (Dekodierung und LED-Anzeige) sind bereits als Beispielcode realisiert und befinden sich in der Anlage 7. Alle verwendeten Programmbibliotheken sind aus den Originaldokumenten zu den jeweiligen Mikrocontrollern entnommen [20] und hier nicht aufgeführt.

Zur Erläuterung der Software hier einige Bemerkungen:

Anlage 6:

1. Ausnutzung der Compilereigenheiten

```
typedef struct
{
     uint16_t      zehntausendstel;
     uint8_t sekunden;
     uint8_t minuten;
     uint8_t stunden;
     uint8_t tage;
     uint8_t woche;
     uint8_t monat;
     uint8_t jahr;
} time_int_t;

volatile time_int_t time;
volatile time_int_t* p_time=&time;
```

Durch diese Konstruktion wird eine globale Variable erzeugt (time). Der Zugriff auf diese Variable erfolgt über das Indexregister des Mikrocontrollers. Es wurde dadurch ein schnellerer Zugriff erreicht und Assemblercode vermieden.

2. Schnelles Debuggen unter Ausnutzung des printf( )-Befehls

```
int uputc(char c,FILE *s){
	do{
	}while(!USART_IsTXDataRegisterEmpty(&USARTPort));
	USART_PutChar(&USARTPort, c );
	return 0;
}
int ugetche(FILE *stream){
	uint16_t timeout = 1000;
	uint16_t c = 0;
	do{
		timeout--;
	}while(!USART_IsRXComplete(&USARTPort) && timeout!=0);
	c = USART_GetChar(&USARTPort);
	c=65;
	return c;
}
```

Durch Aufruf von:

```
fdevopen(uputc,ugetche);
```

im Hauptprogramm wird die Standardaus- und Eingabe auf die oben genannten Funktionen geleitet. Dadurch können die Befehle printf und scanf verwendet werden. Die Ausgabe erfolgt auf die serielle Schnittstelle des Mikrocontrollers. Diese ist auf 921600 Bit/s eingestellt. Das ermöglicht eine sehr schnelle Ausgabe großer Datenmengen.

3. Debuggen über das Programm „JAVA-Oszi“:

```
printf("0");
printf("r%02x",dma_buff1[i]);
printf("\n");
```

Die erste Zeile wählt den Chart aus. Es sind 10 Charts möglich. Das kleine „r“ in der zweiten Zeile steht für die Farbe Rot (b=Blau, g=Grün usw.), der Bytewert wird dann hexadezimal übertragen (%02X). Die dritte Zeile schließt den Datensatz ab. Es wird im Chart ein roter Punkt gezeichnet. Die Position in Y-Richtung bestimmt der hexadezimale Wert. In X-Richtung wird einfach nur inkrementiert.

4. Ausgabe über LCD-Display:

Für die Programmentwicklung und das Debuggen wird das „Xmega Webserver Board V1.0“ verwendet. Es besitzt ein LCD-Display, welches mit SPI-angesteuert wird. Deshalb sind die Funktionen

```
void SPI_Init();
void Display_Init();
```

implementiert.

Anlage 7

1. ifdef HTWM_Edition

Dieser Compilerschalter für dieses Projekt wird nicht verwendet. Er ist für eine alternative Hardware vorgesehen.

2. Temperatursensoren

Es ist möglich über die vorhandene i^2C-Schnittstelle einen Temperatursensor anzuschließen. Dieser ist für zukünftige Entwicklungen vorgesehen (Temperaturregelung Quarz).

6 Aufbau, Inbetriebnahme und Probleme

"Ich bin Mr. Wolf. Sie sind 30 Minuten entfernt?
Ich bin in 10 Minuten da!"

Mr.Wolf in „Pulp Fiction“

6.1 Aufbau

Schon bei den ersten Überlegungen wurde klar, dass sowohl der analoge, als auch der digitale Teil nur mittels SMD-Technik realisiert werden kann. Ein Probeaufbau auf Experimentierplatinen schied aus Abschirmungsgründen ebenfalls aus. Es musste also eine professionelle Leiterplatte schon für den Prototyp hergestellt werden. Aus Kostengründen ist der Hersteller „Eurocircuits GmbH“ ausgewählt worden. Der Hersteller bietet weiterhin ein ansprechendes Webfrontend für die Auftragsbearbeitung und Auftragskontrolle an, welches einen schnellen und fehlerarmen Auftragsverlauf verspricht. Als Datenformat zur Übergabe der Produktionsdaten ist die „Eagle“ brd-Datei direkt verwendet worden.

Da zur Montage kein Bestückungsautomat oder Infrarotofen zu Verfügung stand, erfolgte die Montage der Bauelemente mit einem einfachen Lötwerkzeug sowie 0,35 mm Lötzinn. Als zusätzliches Flussmittel diente „Stanniol EF350“. Sowohl zeitlich als auch qualitätsmäßig wurden die Erwartungen an die Herstellung des Prototyps bei Weitem übertroffen. Das nachfolgende Bild soll das demonstrieren:

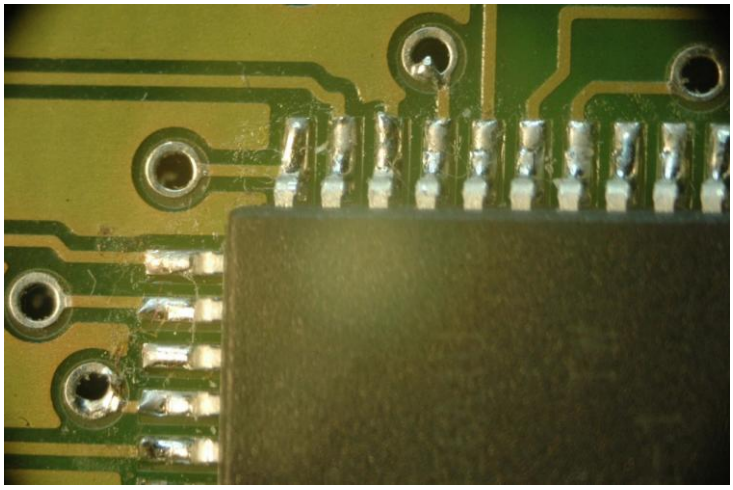


Abbildung 21 SMD-Löten

Die sichtbaren Flussmittelreste können auf der Platine verbleiben. Der Einsatz eines Auflichtmikroskops ist aber unabdingbar. Es wurde der triokulare Typ „Bresser Advance ICD 10x-160x“ eingesetzt.

6.2 Inbetriebnahme

Die Inbetriebnahme des Digitalteils gestaltete sich problemlos. Nach dem Programmieren des Flashspeichers des Mikrocontrollers mit einer Software, welche nur das Amplitudensignal auswertet und dem Anschluss einer Ferritantenne der Firma „Pollin“ konnte nach ca. 5 min eine Synchronisation erreicht werden. Dabei war es nicht notwendig, dass die Antenne genau in Richtung Westen ausgerichtet war. Nur das Empfangsminimum der Ferritantenne ist sehr ausgeprägt. Bei einem Abstand von kleiner 10 cm zum Controller oder zum Schaltnetzteil wurde keine Synchronisation erreicht.

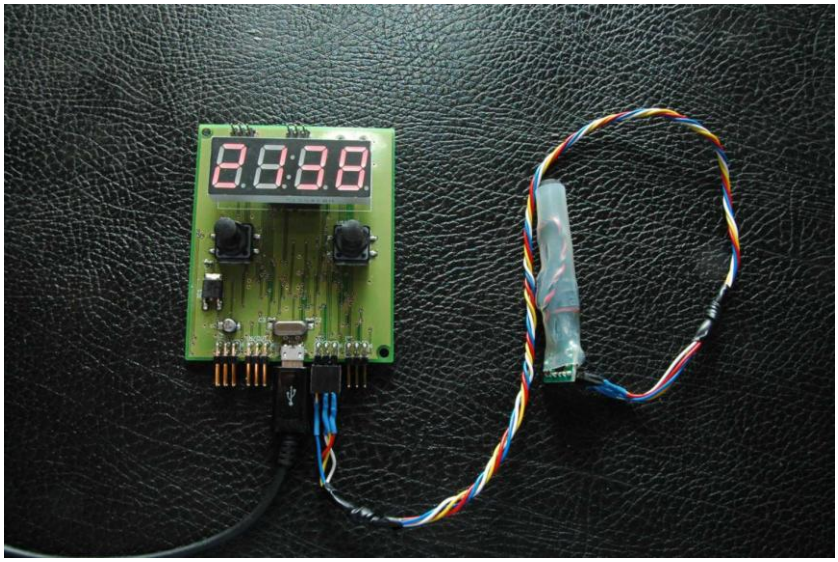


Abbildung 22 Digitalteil mit AM-Antenne

Das Analogteil könnte nur im eingeschobenen Zustand bei voller Verstärkung betrieben werde. Ohne abschirmendes Gehäuse kommt es sofort zum Schwingen auf etwa der Nennfrequenz. Dieses Verhalten war erwartet worden (Kapitel 3.2).

Abbildung 23 Analogteil teilweise eingeschoben

6.3 Probleme

Bei der Auswahl der Operationsverstärker wurde der Fehler gemacht, nicht das Verstärkungsbandbreiteprodukt zu beachten. Der preiswerte OPV OPA2340 ist für 77500 Hz bei 32 dB Verstärkung im Verstärker 2 nicht geeignet. Es wurde stattdessen der RailtoRail-Typ LM6142A mit einem typischen Verstärkungsbandbreiteprodukt von 17 MHz verwendet.

Schwierigkeiten gab es weiterhin beim Einsatz von sehr preiswerten Steckernetzteilen. Diese erzeugen eine starke Störstrahlung, welche eine Synchronisation verhindert.

Die größten Probleme gab es bei der Entwicklung der Software. Es wird noch an der Phasendemodulation gearbeitet. Da diese die Voraussetzung für die Korrelatoren ist, fehlen auch diese. Es ist noch keine befriedigende Lösung für die Nachsteuerung der Samplefrequenz gefunden. Zwischenzeitlich wurde auch der Einsatz eines spannungsgesteuerten Oszillators vorbereitet um die Korrelatoren testen zu können. Da die Realisierung in das Projekt SNTP-Server [11] eingebunden ist, wird auch nach Abschluss dieser Arbeit an der Software weiterentwickelt.

7 Ausblick

“Tue es, oder tue es nicht. Es gibt kein Versuchen.“

Meister Yoda in „Das Imperium schlägt zurück“

7.1 Aufgaben bis zum Projektabschluss

Leider fehlen für einen kompletten Abschluss des Projektes noch verschiedene Programmteile:

1. Überarbeitung und Testung des Görtzelfilters. Die noch vorhandenen real-Werte müssen durch integer-Werte ersetzt werden. Dazu ist es notwendig, Erfahrungen mit der Festkommabibliothek des AVR-Compilers zu sammeln.

2. Realisierung der Phasendemodulation. Die Phasendemodulation benötigt ebenfalls die Festkommabibliothek. Weiterhin steht die Lösung für das Nachregeln der Abtastfrequenz noch aus.

3. Programmieren und Testen der Korrelatoren. Der Algorithmus für einen einzelnen Korrelator ist zwar relativ simpel, die mögliche Anzahl der Korrelatoren ist aber von der noch zur Verfügung stehenden Rechenzeit abhängig. Ohne konkrete Werte der Rechenzeit für die Phasendemodulation ist für die Korrelatorenanzahl keine Zahl zu nennen.

7.2 Realisierung als kompaktes Modul

Durch den Einsatz von einfachen Tiefpassfiltern und einer wirksamen Abschirmung könnten bei Anwendung von Mikrocontrollern der ATXmega-Serie „E“ sehr kleine Module aufgebaut werden. Die Ausgabe der Zeit würde dann über eine serielle Schnittstelle erfolgen. Denkbar ist auch eine Ausgabe der Zeit als simuliertes DCF77-Protokoll. Dadurch könnte dieses Modul als Ersatz für die Module der Firmen „Conrad“, „Reichelt“ oder „Pollin“ dienen.

7.3 Einsatz als SNTP-Server

Vor allem im Verlauf der Projektbearbeitung von [11] konnten viele Erfahrungen bei der Kopplung von ATXmega-Controllern mit Ethernet/TCP/IP-Komponenten gemacht werden.

Da die als Entwicklungsplattform verwendete Platine bereits solche Komponenten enthält, ist eine Realisierung des SNTP-Servers relativ einfach.

7.4 Einsatz als NTP-Server mittels preiswerter Rechnermodule

Es sind sehr preiswerte Mikrocontroller-Module auf dem Markt verfügbar. Als Beispiel sei der „Rasberry Pi“ genannt. Diese Module verwenden schnelle Mikroprozessoren der „ARM“-Architektur. Für diese Module sind komplette Linux-Distributionen verfügbar. Durch den Einsatz eines externen auf die Schnittstelle steckbaren AD-Wandlers mit SPI-Protokoll und Tiefpassfiltern ist es möglich, das hier beschriebene Prinzip zu verwenden. Als Software würde die originale Version des NTP-Servers verwendet. Zur Kopplung ist dann ein entsprechender Treiber für die Hardware und die Demodulation/Korrelation zu schreiben.

Literaturverzeichnis

[1] http://www.ptb.de/cms/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/standort-des-senders.html, 18.3.2010

[2] D. Piester, P. Hetzel, A. Bauch: Zeit- und Normalfrequenzverbreitung mit DCF77 [pdf]5); PTB-Mitteilungen 114, Heft 4, 345-368 (2004)

[3] P. Hetzel: Zeitübertragung auf Langwelle durch amplitudenmodulierte Zeitsignale und pseudozufällige Umtastung der Trägerphase [pdf]. Dissertation, Universität Stuttgart (1987)

[4] D. Piester, A. Bauch, J. Becker, A. Hoppmann: *Time and frequency broadcast with DCF77 [pdf]*; Proc. 43rd Annual Precise Time and Time Interval (PTTI) Systems and Applications Meeting, 14-17 Nov 2011, Long Beach, California, USA, pp. 185-196 (2012)

[5] D. Piester, A. Bauch, J. Becker, T. Polewka, M. Rost, D. Sibold, E. Staliuniene: *PTB's Time and Frequency Activities in 2006: New DCF77 Electronics, New NTP Servers, and Calibration Activities [pdf];* Proc. 38th Annual Precise Time and Time Interval (PTTI) Systems and Applications Meeting, Reston, Virginia, USA, 5-7 Dec 2006, 37-47 (2007)

[6] P. Hetzel: Der Langwellensender DCF77 auf 77,5 kHz: 40 Jahre Zeitsignale und Normalfrequenz, 25 Jahre kodierte Zeitinformation [pdf]5); PTB-Mitteilungen 109, Heft 1, 11-18 (1999)

[7] A. Bauch: Zeitmessung in der PTB[pdf]) PTB-Mitteilungen 122, 23-36 (2012)

[8] *Das Internationale Einheitensystem (SI)*. Deutsche Übersetzung der BIPM-Broschüre „Le Système international d'unités/The International System of Units (8e édition, 2006)". In: *PTB-Mitteilungen*. 117, Nr. 2, 2007 (übersetzt von Cécile Charvieux)

[9] *http://www.ptb.de/cms/de/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/sendereinrichtungen.html, 15.3.2010*

[10] *http://www.ptb.de/cms/de/fachabteilungen/abt4/fb-44/ag-442/verbreitung-der-gesetzlichen-zeit/dcf77/sendereinrichtungen.html, 15.3.2010*

[11] *Ch.Vorberg, Studie zur Realisierung eines Stratum-1- NTP-Servers, Hauptpraktikumsarbeit Hochschule Mittweida 20.5.2013*

[12] *http://www.meinberg.de/german/products/3he-dcf77-korrelations-empfaenger.htm, 29.6.2011*

[13] *http://www.meinberg.de/german/products/pci-express-dcf77-uhr.htm, 18.7.2012*

[14] *D.Engeler, Performance Analysis and Receiver Architectures of DCF77 Radio-Controlled Clocks (PDF)*

[15] *Elektor Heft 1/2012, DSP-basierter DCF77-Zeitsignal-Empfänger*

[16] *Claude Elwood Shannon: Communication in the Presence of Noise, January 1949*

[17] *N.Graubner, W. Traving., DCF77-gesteuertes Frequenznormal mit Funkuhr und Sternenzeit, Funkamateur 2/2009 Seite 153*

[18] *Christian Vorberg, JAVA-Oszi, Grafische Anzeige serieller Daten, Hormersdorf im Januar 2013*

[19] *Gerald Goertzel: An Algorithm for the Evaluation of Finite Trigonometric Series. In: American Math. Monthly. Vol 65. 1958, S. 34-35.*

[20] *http://www.atmel.com/products/microcontrollers/avr/default.aspx*

[21] *Jack E. Volder: The CORDIC Trigonometric Computing Technique. In: IRE Transactions on Electronic Computers. September 1959*

Anlagenverzeichnis

Anlagen, Teil 1 Gesetzblatt

Reichs-Gesetzblatt.

№ 7.

(Nr. 2075.) Gesetz, betreffend die Einführung einer einheitlichen Zeitbestimmung. Vom 12. März 1893.

Wir Wilhelm, von Gottes Gnaden Deutscher Kaiser, König von Preußen ꝛc.

verordnen im Namen des Reichs, nach erfolgter Zustimmung des Bundesraths und des Reichstags, was folgt:

Die gesetzliche Zeit in Deutschland ist die mittlere Sonnenzeit des fünfzehnten Längengrades östlich von Greenwich.

Dieses Gesetz tritt mit dem Zeitpunkt in Kraft, in welchem nach der im vorhergehenden Absatz festgesetzten Zeitbestimmung der 1. April 1893 beginnt.

Urkundlich unter Unserer Höchsteigenhändigen Unterschrift und beigedrucktem Kaiserlichen Insiegel.

Gegeben Berlin Schloß, den 12. März 1893.

(L. S.) Wilhelm.

Graf von Caprivi.

Herausgegeben im Reichsamt des Innern.

Berlin, gedruckt in der Reichsdruckerei.

Reichs-Gesetzbl. 1893. 16

Ausgegeben zu Berlin den 16. März 1893.

Anlagen, Teil 2 Pseudozufallsfolge

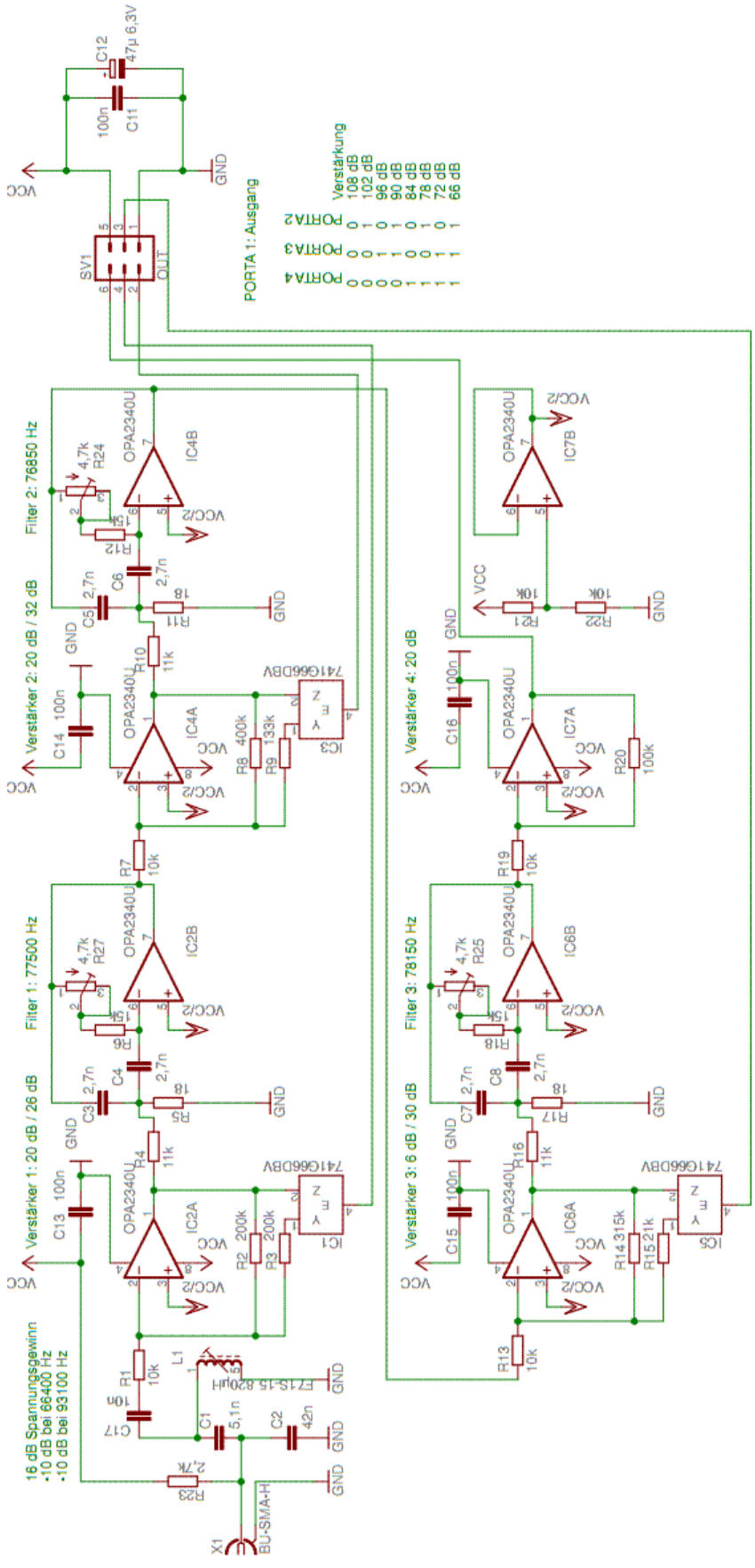
Programm zur Erzeugung der PZF:

```
#include <stdio.h>
int main(void){
	int bit1,bit5,bit9,sr,i;
	int a[512];
	int nullen=0;
	int einsen=0;
	bit1=1;sr=0;
	for(i=0;i<512;i++) {
		sr=sr << 1;
		sr=sr | bit1;
		sr=sr & 0x1ff;
		if ((sr & 0x10)>0) bit5=1; else bit5=0;
		if ((sr & 0x100)>0) bit9=1; else bit9=0;
		bit1=bit5^bit9;
		if (bit1==0) {
			a[i]=0;
			nullen++;
		} else {
			a[i]=1;
			einsen++;
		}
	}	printf("const char noise[] PROGMEM = { ");
		for(i=0;i<512;i++) {
			if (i>0) printf(",");
			if (((i)%32)==0) printf("\n");
				printf("%1d",a[i]);	}
	printf("};\n");
	printf("/* Anzahl Einsen: %d ",einsen);
	printf("Anzahl Nullen: %d*/\n",nullen);
	return 0;
}
```

Pseudozufallsfolge (PZF) des DCF77

```
const char noise[] PROGMEM = {
0,0,0,0,1,0,0,0,1,1,0,0,0,0,1,0,0,1,1,1,0,0,1,0,1,0,1,0,1,1,0,0,
0,0,1,1,0,1,1,1,1,0,1,0,0,1,1,0,1,1,1,0,0,1,0,0,0,1,0,1,0,0,0,0,
1,0,1,0,1,1,0,1,0,0,1,1,1,1,1,1,0,1,1,0,0,1,0,0,1,0,0,1,0,1,1,0,
1,1,1,1,1,1,0,0,1,0,0,1,1,0,1,0,1,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,
1,1,0,0,0,1,1,0,0,1,0,1,0,0,0,1,1,0,1,0,0,1,0,1,1,1,1,1,1,1,0,1,
0,0,0,1,0,1,1,0,0,0,1,1,1,0,1,0,1,1,0,0,1,0,1,1,0,0,1,1,1,1,0,0,
0,1,1,1,1,1,0,1,1,1,0,1,0,0,0,0,0,1,1,0,1,0,1,1,0,1,1,0,1,1,1,0,
1,1,0,0,0,0,0,1,0,1,1,0,1,0,1,1,1,1,1,0,1,0,1,0,1,0,1,0,0,0,0,0,
0,1,0,1,0,0,1,0,1,0,1,1,1,1,0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,0,1,
1,1,0,0,1,1,1,0,1,0,0,1,0,0,1,1,1,1,0,1,0,1,1,1,0,1,0,1,0,0,0,1,
0,0,1,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,1,0,1,1,1,1,0,1,1,0,1,1,0,0,
1,1,0,1,0,0,0,0,1,1,1,0,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,0,0,0,
0,0,1,1,1,1,0,1,1,1,1,1,0,0,0,1,0,1,1,1,0,0,1,1,0,0,1,0,0,0,0,0,
1,0,0,1,0,1,0,0,1,1,1,0,1,1,0,1,0,0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,
1,1,0,1,1,0,0,0,1,0,1,0,1,0,0,1,0,0,0,1,1,1,0,0,0,1,1,0,1,1,0,1,
0,1,0,1,1,1,0,0,0,1,0,0,1,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0};
/* Anzahl Einsen: 256 Anzahl Nullen: 256*/
```

Anlagen, Teil 3 Analogteil - Eagle



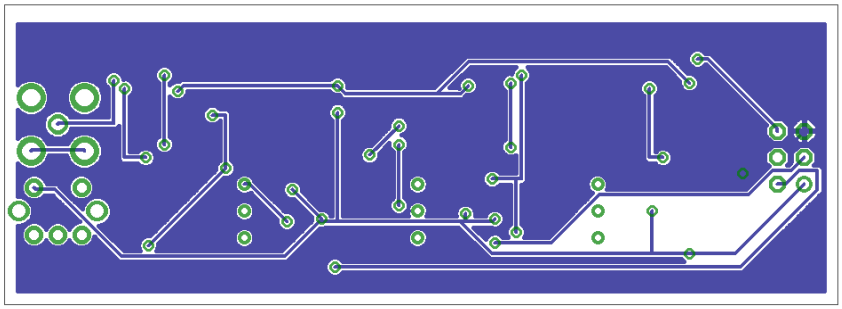


Abbildung 24 Analog-Teil-Bottom

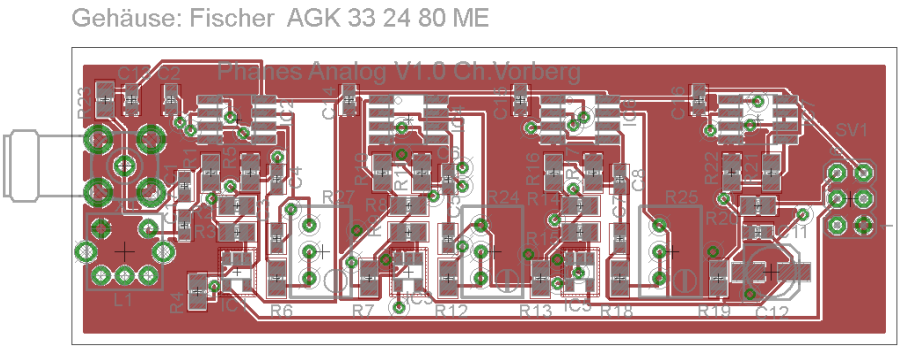
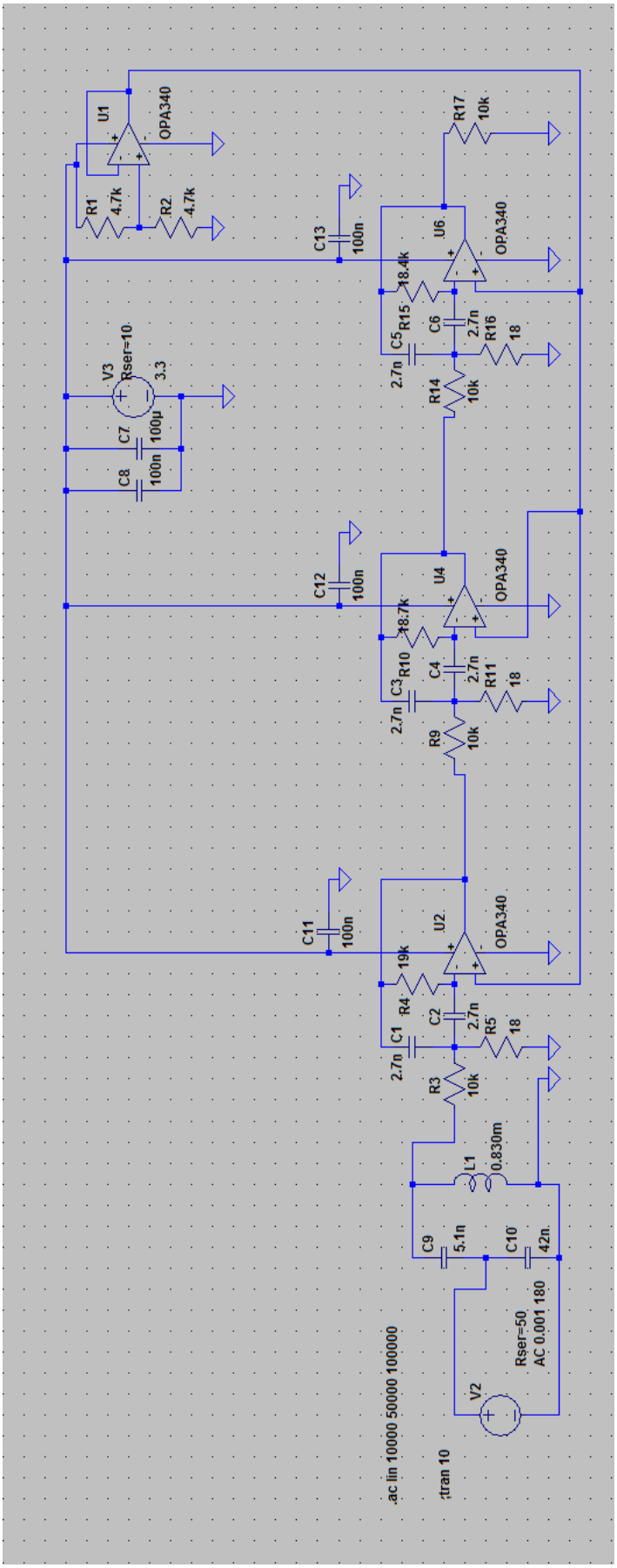
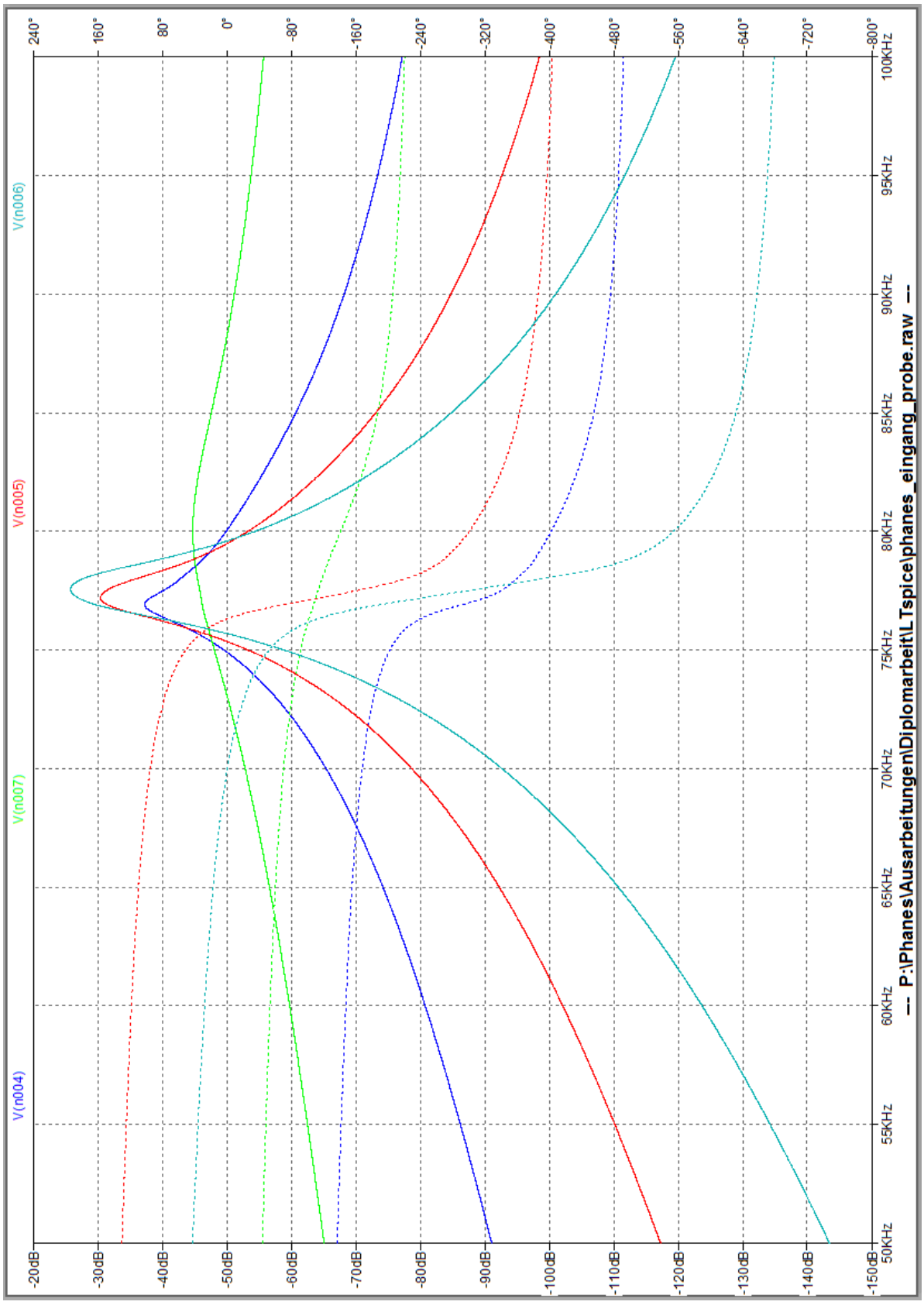


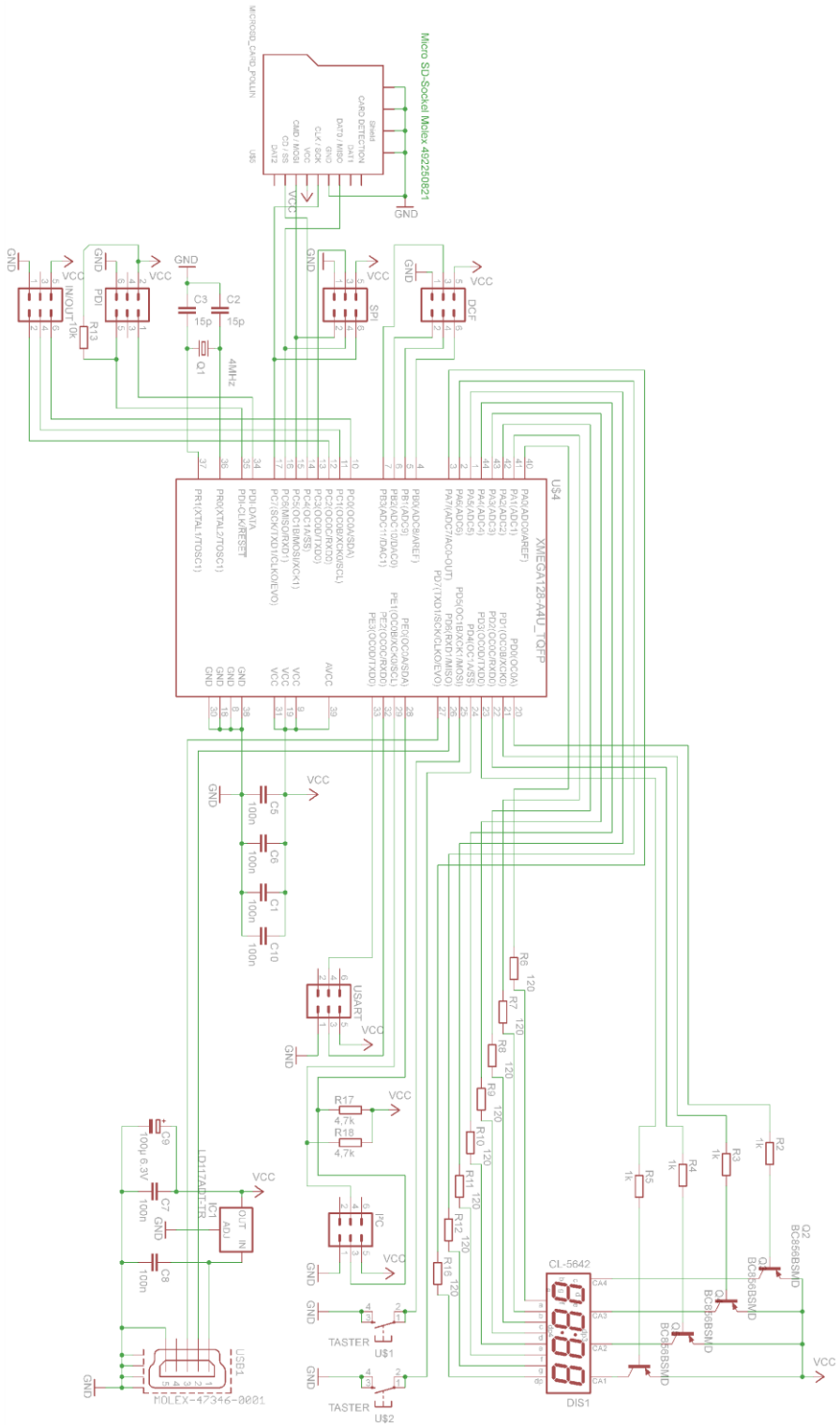
Abbildung 25 Analog-Teil-Top

Anlagen, Teil 4 Analogteil - LTspice




V(n004)
V(n007)
V(n005)
V(n006)
-- P:\Phanes\Ausarbeitungen\Diplomarbeit\LTspice\phanes_eingang_probe.raw --
240°
160°
80°
0°
-80°
-160°
-240°
-320°
-400°
-480°
-560°
-640°
-720°
-800°
-20dB
-30dB
-40dB
-50dB
-60dB
-70dB
-80dB
-90dB
-100dB
-110dB
-120dB
-130dB
-140dB
-150dB
50KHz
55KHz
60KHz
65KHz
70KHz
75KHz
80KHz
85KHz
90KHz
95KHz
100KHz

Anlagen, Teil 5 Digitalteil



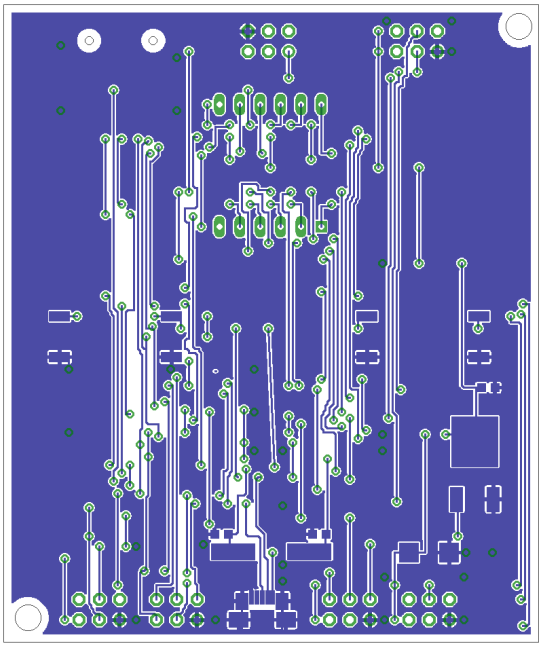


Abbildung 26 Digital-PCB-Bottom

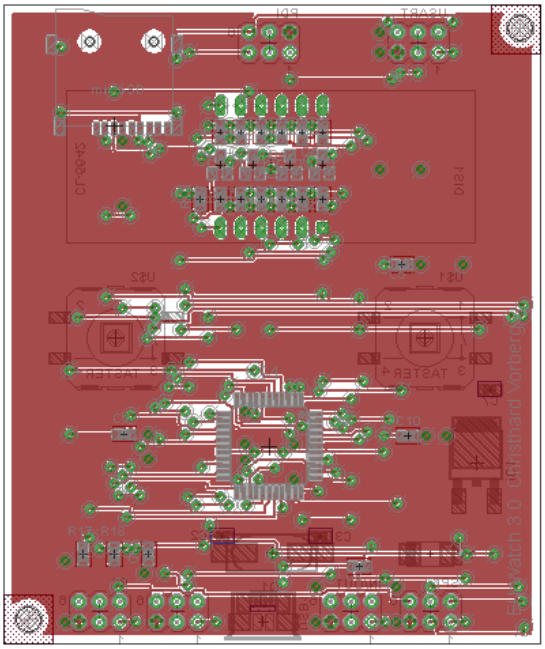


Abbildung 27 Digital-PCB-Top

Anlagen, Teil 6 Hauptprogramm Phanes

```
/*
 * phanes.c
 *
 * Created: 06.06.2014 11:24:10
 *  Author: christ
 */

#define BOARD USER_BOARD

#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <string.h>
#include <math.h>
#include <inttypes.h>
#include "types.h"
#include "asf.h"
#include "board.h"
#include "conf_board.h"
#include "clksys_driver.h"
#include "avr_compiler.h"
#include "usart_driver.h"
#include "event_system_driver.h"

#define USARTPort USARTD0
#define cbi(port,bit) (port) &= ~(1 << (bit))
#define sbi(port,bit) (port) |= (1 << (bit))
#define DECIMIERUNG 100
#define DECIMIERUNG_LENGTH DECIMIERUNG+1
#define FK 77500                    // Gesuchte Frequenz
#define FS FK*3                    // Abtastfrequenz
#define N DECIMIERUNG          // Größe des dma-Buffers

volatile int16 millisekunden=0;
volatile uint32 sekunden=0;
volatile uint32 dmazaehler1=0;
volatile uint32 dmazaehler2=0;

volatile uint8 dma_buff1[DECIMIERUNG_LENGTH];
volatile uint8 dma_buff2[DECIMIERUNG_LENGTH];

void goerzel(volatile uint8*);
void TCC0_OVF_vect(void) __attribute__((interrupt));
void DMA_CH0_vect(void) __attribute__((interrupt));
void DMA_CH1_vect(void) __attribute__((interrupt));
void Clock_Init();
void Timer_Init();
void Ports_Init();
void SPI_Init();
void Display_Init();
void USART_Init();
void ADU_Init();
void Display_Write_Byte(uint8);
void Display_Write_Byte_Dezi(uint8);
void Display_Write_Byte_Control(uint8);
typedef struct
{
      uint16_t       zehntausendstel;
```

```
	uint8_t sekunden;
	uint8_t minuten;
	uint8_t stunden;
	uint8_t tage;
	uint8_t woche;
	uint8_t monat;
	uint8_t jahr;
} time_int_t;

volatile time_int_t time;
volatile time_int_t* p_time=&time;

int uputc(char c,FILE *s){
	do{
	}while(!USART_IsTXDataRegisterEmpty(&USARTPort));
	USART_PutChar(&USARTPort, c );
	return 0;
}
int ugetche(FILE *stream){
	uint16_t timeout = 1000;
	uint16_t c = 0;
	do{
		timeout--;
	}while(!USART_IsRXComplete(&USARTPort) && timeout!=0);
	c = USART_GetChar(&USARTPort);
	c=65;
	return c;
}

int main(void){
	Clock_Init();
	_delay_ms(100);			//Dem Diplay Zeit lassen
	Timer_Init();
	Ports_Init();
	SPI_Init();
	Display_Init();
	USART_Init();
	ADU_Init();
	p_time->zehntausendstel=0;
	p_time->sekunden=0;
	p_time->minuten=0;
	p_time->stunden=0;
	p_time->tage=1;
	p_time->woche=0;
	p_time->monat=1;
	p_time->jahr=0;
	sei();
	PORTA.OUT=0x0f;
	uint8 sek_temp=0;
	fdevopen(uputc,ugetche);
	while(1)
	{
		if (p_time->sekunden!=sek_temp) {
			sek_temp=p_time->sekunden;
			Display_Write_Byte_Control(0x02);
			Display_Write_Byte(0x20);
			Display_Write_Byte_Dezi(p_time->stunden);
			Display_Write_Byte(0x3a);
			Display_Write_Byte_Dezi(p_time->minuten);
			Display_Write_Byte(0x3a);
			Display_Write_Byte_Dezi(p_time->sekunden);
			Display_Write_Byte(0x20);
			Display_Write_Byte_Dezi(p_time->tage);
```

```
                Display_Write_Byte(0x2e);
                Display_Write_Byte_Dezi(p_time->monat);
      //            printf("0%d\n\r",p_time->sekunden);
            //_delay_ms(1);
      /*      do{
                }while(!USART_IsTXDataRegisterEmpty(&USARTPort));
                USART_PutChar(&USARTPort, p_time->sekunden+0x20 );
            }*/
//
printf("0n%02x%02x%02x%02x%02x%02x\n",dma_buff1[0],dma_buff1[1],dma_buff1[2],dma_buff
1[3],dma_buff1[4]);
/*                int i;
                printf("0");
                for (i=0;i<DECIMIERUNG;i++)     printf("r%02x",dma_buff1[i]);
                printf("\n");
                printf("1");
                for (i=0;i<DECIMIERUNG;i++)     printf("b%02x",dma_buff2[i]);
                printf("\n");
*/

            }
      }
}
void TCC0_OVF_vect(void) {     //wird alle 1ms ausgeführt
      p_time->zehntausendstel++;

      //      PORTD.OUTTGL|=PIN3_bm;
                                                //Realisierung der internen Uhr
      if (p_time->zehntausendstel>9999) {
            p_time->sekunden++;     //if ((p_time->sekunden%2)==0)
            PORTA.OUT=p_time->sekunden;
            p_time->zehntausendstel=0;
            if (p_time->sekunden>59) {
                  p_time->minuten++;p_time->sekunden=0;
                  //if (dcf_timeout<MINUTEN_PRO_TAG) dcf_timeout++;   //zählt ohne dcfsync
hoch bis Max 1 Tag
                  if (p_time->minuten>59) {
                        p_time->stunden++;
                        p_time->minuten=0;
                        if (p_time->stunden>23) {
                              p_time->stunden=0;
                              p_time->tage++;
                              p_time->woche++;
                              if (p_time->woche>7) p_time->woche=1;
                              if (((p_time->monat==1) && (p_time->tage>31))
                                          ||((p_time->monat==2) && (p_time->tage>28) &&
((p_time->jahr%4)!=0))
                                          ||((p_time->monat==2) && (p_time->tage>29) &&
((p_time->jahr%4)==0))
                                          ||((p_time->monat==3) && (p_time->tage>31))
                                          ||((p_time->monat==4) && (p_time->tage>30))
                                          ||((p_time->monat==5) && (p_time->tage>31))
                                          ||((p_time->monat==6) && (p_time->tage>30))
                                          ||((p_time->monat==7) && (p_time->tage>31))
                                          ||((p_time->monat==8) && (p_time->tage>31))
                                          ||((p_time->monat==9) && (p_time->tage>30))
                                          ||((p_time->monat==10) && (p_time->tage>31))
                                          ||((p_time->monat==11) && (p_time->tage>30))
                                          ||((p_time->monat==12) && (p_time->tage>31)))
                                    {
                                          p_time->monat++;
                                          p_time->tage=1;
                                    }
```

```
                        if (p_time->monat>12)
                        {
                              p_time->jahr++;
                              p_time->monat=1;
                        }
                  }
            }
      }
      //PortA.OUT=p_time->sekunden;

}
void goerzel(volatile uint8* x){
#define K FK*N/FS
#define A  27  //  -1.52 *64
#define wkn  42 //  0.649 *64
//#define A  -1.52
//#define wkn  0.649

#define pi 3.14
int v1=0;
int v2=0;
int v3=0;
int real;
int imag;
//float ampli;
//float phase;
      v1=0;
      v2=0;
      uint8 i;
//     dmazaehler1++;
//     x[DECIMIERUNG-1]=0;                        //Letztes Element muss 0 sein
      for (i=0;i<DECIMIERUNG-1;i++) {
            v1=x[i]+A*v2-v3;
            v3=v2;
            v2=v1;
      }
      v1=v1/64;
      v2=v2/64;
      v3=v3/64;
      real=v2-A/2*v3;
      imag=v3*wkn;
      int ausgabe;
//     ampli=sqrt(real*real+imag*imag);
//     phase=atan(imag/real);
//     if (real<0) phase=phase-pi/2; else phase=phase+pi/2;
      ausgabe=real/256;
//     printf("0r%d\n",ausgabe);
//     ausgabe=phase;
//     printf("0b%02x\n",ausgabe);
}

void DMA_CH0_vect(void) {
            sbi(DMA.CH0.CTRLB,4);
            dmazaehler1++;
            int i;
                  printf("0");
            //      for (i=0;i<DECIMIERUNG;i++)
                  printf("r%02x",dma_buff1[i]);
                  printf("\n");

            goerzel(dma_buff1);
}
```

```
void DMA_CH1_vect(void) {
		sbi(DMA.CH1.CTRLB,4);
		dmazaehler2++;
		int i;
			printf("0");
		//	for (i=0;i<DECIMIERUNG;i++)
			printf("b%02x",dma_buff2[i]);
			printf("\n");

		goerzel(dma_buff2);
}

void Clock_Init(){
	CLKSYS_XOSC_Config( OSC_FRQRANGE_2TO9_gc,false,OSC_XOSCSEL_XTAL_256CLK_gc
); //4MHz
	CLKSYS_Enable( OSC_XOSCEN_bm );
	do {} while ( CLKSYS_IsReady( OSC_XOSCRDY_bm ) == 0 );
	CLKSYS_Main_ClockSource_Select( CLK_SCLKSEL_XOSC_gc );

// Configure the PLL
	CLKSYS_PLL_Config( OSC_PLLSRC_XOSC_gc, 31);
// -> PLL running at 124 MHz
	CLKSYS_Enable(OSC_PLLEN_bm);
	CLKSYS_Prescalers_Config( 0, CLK_PSBCDIV_2_2_gc );
// Vorteiler A=1 B=2 C=2  -> CPU running at 31 MHz
	do {} while ( CLKSYS_IsReady( OSC_PLLRDY_bm ) == 0 );
	CLKSYS_Main_ClockSource_Select( CLK_SCLKSEL_PLL_gc );
	CLKSYS_Disable(OSC_RC32MEN_bm );
	CLKSYS_Disable(OSC_RC2MEN_bm);
};
void Timer_Init(){
	PMIC_CTRL = 0x06; // Alle interrupts enable ausser Low
	TCC0_CTRLA = 1; // Vorteiler 1
	TCC0_PER = 3100;  // Interrupt alle 100µs
	TCC0_INTCTRLA=3;	// Interrupt High-Level an für Overflow
};
void Ports_Init(){
	PORTA.DIR=0x0f;		//PA0-3 Ausgang (LED) PA4-7Eingang (Taster)
}
void SPI_Init(){
		//Übernommen aus dem original Atmeldokumentation
		/* Init SS pin as output with wired AND and pull-up. */
	PORTB.DIRSET = PIN7_bm;
//	PORTB.PIN7CTRL = PORT_OPC_WIREDANDPULL_gc;
	/* Set SS output to high. (No slave addressed). */
	PORTB.OUTSET = PIN7_bm;
	/* Instantiate pointer to ssPort. */
	PORTD.DIRSET = PIN7_bm;		//spi_clk auf Ausgang
	PORTD.DIRSET = PIN5_bm;		//spi_mosi auf Ausgang
	PORTD.DIR = 0xB0;  // configure MOSI, SS, CLK as outputs on PORTD
	 /* Initialize SPI master on port D. */
   SPID.CTRL=0b01010011;
//	PORTD.OUT |= (1<<5) | (1<<4);
 }
void Display_Init(){
	char init_string[] ={0x39,0x15,0x55,0x6e,0x72,0x38,0x0c,0x01,0x06,0x02};
	uint8 i;
	// R/W Pin config
	PORTE.DIRSET = PIN2_bm;
//	PORTE.PIN2CTRL = PORT_OPC_WIREDANDPULL_gc;
	PORTE.DIRSET = PIN2_bm;
//	PORTB.PIN7CTRL = PORT_OPC_WIREDANDPULL_gc;
	PORTE.OUTCLR = PIN2_bm;
```

```
	PORTB.OUTCLR = PIN7_bm;
	for (i=0;i<sizeof(init_string);i++) {
		SPID.DATA = init_string[i];
		while(!(SPID.STATUS & (1<<7)));
	}
	_delay_ms(100);
}

void Display_Write_Byte(uint8 data){
	PORTB.OUTCLR = PIN7_bm;
	PORTE.OUTSET = PIN2_bm;
	SPID.DATA = data;
	while(!(SPID.STATUS & (1<<7)));
}
void Display_Write_Byte_Control(uint8 data){
	PORTB.OUTCLR = PIN7_bm;
	PORTE.OUTCLR = PIN2_bm;
	SPID.DATA = data;
	while(!(SPID.STATUS & (1<<7)));
}

void Display_Write_Byte_Dezi(uint8 data){
	Display_Write_Byte(data/10+0x30);
	Display_Write_Byte(data%10+0x30);
}
void USART_Init(){
	/* PIN7 (TXD1) as output.		*/
	PORTD.DIRSET = PIN3_bm;
	/* PD2 (RXD1) as input.		*/
	PORTD.DIRCLR = PIN2_bm;
		/* USARTC0, 8 Data bits, No Parity, 1 Stop bit. */
	USART_Format_Set(&USARTPort, USART_CHSIZE_8BIT_gc, USART_PMODE_DISABLED_gc,
false);
	USART_Baudrate_Set(&USARTPort, 150 , -7);  //921600 Bit/s bei 32Mhz
	/* Enable both RX and TX. */
	USART_Rx_Enable(&USARTPort);
	USART_Tx_Enable(&USARTPort);
	fdevopen(uputc,ugetche);    //für printf
}
void ADU_Init(){
/*------------------------------------------- ADU-Timer Initialisierung ----------------------------
--------------*/
	TCC1_CTRLA = 0x1; // Vorteiler 1
	TCC1_PER = 159;  // /160 alle 5µs
 /*------------------------------------------- Event & ADU Initialisierung ---------------------------
---------------*/
	EVSYS_SetEventSource( 0, EVSYS_CHMUX_TCC1_OVF_gc );    //Event7 von Timer-C1-
Overflow
	ADCB.EVCTRL = (uint8_t) ADC_SWEEP_0_gc | ADC_EVSEL_0123_gc |
ADC_EVACT_SWEEP_gc;
	ADCB.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN5_gc;
	ADCB.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
	PORTB.DIRSET= PIN1_bm;
	PORTB.DIRCLR= PIN5_bm;
	ADCB.CTRLB = ( ADCB.CTRLB & ~ADC_RESOLUTION_gm ) |
ADC_RESOLUTION_12BIT_gc;
	ADCB.CTRLB = ( ADCB.CTRLB & ~( ADC_CONMODE_bm | ADC_FREERUN_bm ) );
   ADCB.REFCTRL = ADC_REFSEL_INT1V_gc | 0x02;
   ADCB.PRESCALER = ADC_PRESCALER_DIV8_gc;

 /*------------------------------------------- DMA Initialisierung ------------------------------
--------------*/
	DMA.CTRL =		0b00000100;			//Double Puffer canel 0 & 1 an
```

```
	DMA.CH0.CTRLA = 0b00100100;
	DMA.CH0.CTRLB = 0b00000011;                    //Interrupt Middle Priority
	DMA.CH0.ADDRCTRL = 0b10001101;  //Quelle kein reload Destination reload nach Trans-
action increment
	DMA.CH0.TRIGSRC = 0x20;              // ADCB CH0
	DMA.CH0.TRFCNTH = 0;
	DMA.CH0.TRFCNTL = DECIMIERUNG;
	DMA.CH0.TRFCNTH = 0;
	DMA.CH0.DESTADDR0 = 0xff & ((uint16_t)  (&dma_buff1[0]));
	DMA.CH0.DESTADDR1 = 0xff & ((uint16_t) (&dma_buff1[0]) >> 8) ;
	DMA.CH0.DESTADDR2 = 0;
	DMA.CH0.SRCADDR0 =  0x50;//0xff & ((uint16_t)(&ADCB.CH0.RESL)); //0xff & ((unsigned
int)  (&ADCB.CH0.RESL));
	DMA.CH0.SRCADDR1 =  0x02;//0xff & (((uint16_t)(&ADCB.CH0.RESL))>> 8);//0xff &
((unsigned int) (&ADCB.CH0.RESL) >> 8) ;
	DMA.CH0.SRCADDR2 = 0;

	DMA.CH1.CTRLA = 0b00100100;
	DMA.CH1.CTRLB = 0b00000011;                    //Interrupt Middle Priority
	DMA.CH1.ADDRCTRL = 0b10001101;  //Quelle kein reload Destination reload nach Trans-
action increment
	DMA.CH1.TRIGSRC = 0x20;              // ADCB CH0
	DMA.CH1.TRFCNTH = 0;
	DMA.CH1.TRFCNTL = DECIMIERUNG;
	DMA.CH1.TRFCNTH =0;
	DMA.CH1.DESTADDR0 = 0xff & ((uint16_t)  (&dma_buff2[0]));
	DMA.CH1.DESTADDR1 = 0xff & ((uint16_t) (&dma_buff2[0]) >> 8) ;
	DMA.CH1.DESTADDR2 = 0;
	DMA.CH1.SRCADDR0 =  0x50;//0xff & ((uint16_t)(&ADCB.CH0.RESL)); //0xff & ((unsigned
int)  (&ADCB.CH0.RESL));
	DMA.CH1.SRCADDR1 =  0x02;//0xff & (((uint16_t)(&ADCB.CH0.RESL))>> 8);//0xff &
((unsigned int) (&ADCB.CH0.RESL) >> 8) ;
	DMA.CH1.SRCADDR2 = 0;
	DMA.CH0.CTRLA |= DMA_CH_ENABLE_bm; //Nur Kanal 0 wird aktiviert, da Double Puffer
	DMA.CTRL |= DMA_ENABLE_bm;
 /*---------------------------------------------- Hauptprogramm ----------------------------------
------------------*/
	PMIC_CTRL = 0x07;
	printf("Goerzel\n");
	ADCB.CTRLA |= ADC_ENABLE_bm;
};
```

Anlagen, Teil 7 Hauptprogramm Phanes-LED

```
// DCF77 Uhrenmodul EU-Watch Copyright 2013/14 by Christhard Vorberg alle Rechte verbleiben beim Autor
// jegliche nicht autorisierte Verbreitung oder Veränderung bedürfen der Zustimmung des Autors.
// Dieses Programm ist nur für Schulungszwecke vorgesehen.
// Es wird für die korrekte Funktion keine Garantie übernommen.
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/twi.h>
//#include "types.h"
#include "lm75.h"
#include <avr/pgmspace.h>
#include <avr/eeprom.h>
#define F_CPU 8000000UL                    // 8 MHz
#define DCF77_UPDATE 7200            //wieviele Sekunden ohne Sync ohne Benachichtigung
#define WANN_DATUM     3                   //in welche Sekunde wird das Datum angezeigt
#define WANN_WO_DATUM  3
#define WANN_TEMP        4          //Ab welcher Sekunde werden die Temperaturen dargestellt.
#define MAX_TEMPSENSORS 1
#define PT_EEPROM_TEMP_KORR (uint8_t *) 0
//Programm braucht weniger Speicher
// #define LESS_CODE_PROGRAM
//Quellcode für die HTWM-Uhr von Christian
//#define HTWM_EDITION
                                           //Variable für die interne Uhr läuft mit 1ms Genauigkeit
volatile uint16 tausendstel=0;
volatile uint8        zehntausendstel=0;
volatile int8  dimmer=9;
volatile uint8 sekunden=0;
volatile uint8 minuten=0;
volatile uint8 stunden=0;
volatile uint8 tage=1;
volatile uint8  woche=1;
volatile uint8 monat=1;
volatile uint8 jahr=0;
                                           //Variablen für die DCF77 Dekodierung
volatile uint8 dcfcheck;             //Paritätsvaiable
volatile uint8 dcfstate;            //Akt. DCF-Sekunden
volatile uint8 dcf_min;
volatile uint8 letzte_min;
volatile uint8 dcf_std;
volatile uint8 letzte_std;
volatile uint8 letzte_woche;
volatile uint8 dcf_tag=0;
volatile uint8 dcf_woch;
volatile uint8 dcf_monat=1;
volatile uint8 dcf_jahr;
volatile uint8 dcf_status;          //Bit15-Bit20 des DCF-Telegramms (Sommerzeit, Rufbit, Startbit, Schaltsekunde)
volatile uint8 dcfok=0;                    //0=letzte Minute nicht sysnchron 1=letzte Minute hat sysnchronisiert
volatile uint8 dcf77ct;                   //Zeit zwischen den DCF-Impulsen in 10ms z.B.80=800ms
volatile uint8 dcf_debug;           //Zwischenspeicher für die dcf77ct zur Anzeige
```

```
volatile uint8 stelle=0;            // zählt von 1-4
volatile uint8 segmente[]={0,0,0,0};   //7-Segmentbelegung
volatile uint8 dp[]={0,0,0,0};              //bei 5642B   nur dp[2]  belegt
volatile uint8 start=0;                              //0 = Start, 1= normal 2= Woochentag +
Datum
volatile uint8 key_1=0;
volatile uint8 key_2=0;
union tempera {
      int16 temperatur;
      struct{
            uint8 temp1;
            uint8 temp2;
      };
};
union tempera tempera;
int32 temp32, temp32_1;
static uint8 temp_buff[3];
volatile uint16 dcf_timeout=DCF77_UPDATE;
void dcf(void);
void dcf_stoerung(void);
void dcf_scan(unsigned char pit);
void dcf_reset(void);
void dcf_fehler(void);
void zeit(void);
void startanzeige(void);
void datum(void);
void wo_datum(void);
void tempanzeige(uint8);
void sek_anzeige(void);
void debug_anzeige(void);
int siebensegment(int);
uint16 wochentag(uint8);

int main(void){
      uint8 i;
      uint8 merk_sekunde = 0;
      int8  temp_korr[MAX_TEMPSENSORS];
      uint8 menue = 0;
#ifdef HTWM_EDITION
      //Initialisieren der Ports
      DDRB= 0b00000011;                  //Portb = eingang 1,6,7 + ausgang 0-1 + ?
      DDRC= 0b00001111;                  //PortC 0-3  Ausgänge für Stellen D1-D4
      DDRD= 0xff;                         //PortD ausgang A-DP
      PORTD=0;                            //7Seg-LEDs aus
      PORTB |=4;
      TCCR1A=0b00000000;
      TCCR1B=0b00001011;                  //wgm12 (ctc) Vorteiler 64
      OCR1A=124;
      TIMSK1 |=(1<<OCIE1A);         // Timer Overflow Interrupts zulassen
#else
      //Initialisieren der Ports
      DDRC= 0b00111111;                  //PortC 0-3  Ausgänge für Stellen 4-5 TWI
      DDRD=0xff;
      PORTD=0xFF;                           //7Seg-LEDs aus
      DDRB=0b00000000;                   //PortB Eingabe
      PORTB|=0b00000110;                  //Pullup für Taster ein
      TCCR1A=0b00000000;
      TCCR1B=0b00001010;                  //wgm12 (ctc) Vorteiler 8
      OCR1A=99;                            //8MHz/8/100=10000Hz   // 124 atmega8 Fusebit
DIV8 disable!!!
//    TCCR1B=0b00001011;                  //wgm12 (ctc) Vorteiler 64
//    OCR1A=124;                            //8MHz/64/125=1000Hz   // 124 atmega8 Fusebit
DIV8 disable!!!
```

```
#if defined (__AVR_ATmega8__)
      TIMSK|=(1<<OCIE1A);           // Timer Overflow Interrupts zulassen   //TIMSK atmega8
#else
      TIMSK1|=(1<<OCIE1A);          // Timer Overflow Interrupts zulassen   //TIMSK atmega8
#endif

#endif
      sei();                              // Interrupts ein
      twi_init();
      //1.Sensor Adresse 0
      temp_buff[0]=0b10010000;       // 12bit einschalten
      temp_buff[1]=0b00000001;
      temp_buff[2]=0b01100000;
      TW_Start_Transceiver_With_Data(temp_buff,3);
      temp_buff[0]=0b10010000;       // Adress-Pointer zurückstellen
      temp_buff[1]=0b00000000;
      temp_buff[2]=0b00000000;
      TW_Start_Transceiver_With_Data(temp_buff,3);
/*//2.Sensor Adresse 1
      temp_buff[0]=0b10010010;       // 12bit einschalten
      temp_buff[1]=0b00000001;
      temp_buff[2]=0b01100000;
//     TW_Start_Transceiver_With_Data(temp_buff,3);
      temp_buff[0]=0b10010010;             // Adress-Pointer zurückstellen
      temp_buff[1]=0b00000000;
      temp_buff[2]=0b00000000;
//     TW_Start_Transceiver_With_Data(temp_buff,3); */
                                                  //Temperaturkorrektur aus EEPROM holen
      for (i=0;i<MAX_TEMPSENSORS;i++)
temp_korr[i]=eeprom_read_byte(PT_EEPROM_TEMP_KORR+menue-1);
      //Hauptschleife
      while (1){
            set_sleep_mode(SLEEP_MODE_IDLE);
       sleep_mode();                 // in den Schlafmodus wechseln
            if (key_1==1) {
                  if (menue==0)  start++; else temp_korr[menue-1]++;
                  if (start>7) start=1;
      }
            if (key_2==1) {
                  if (menue==0)  dimmer--;       else temp_korr[menue-1]--;
                  if (dimmer<1) dimmer=9;
            }
            if (key_1==2) {
                  menue++;
                  if (menue>MAX_TEMPSENSORS) menue=1;

                  temp_korr[menue-1]=eeprom_read_byte(PT_EEPROM_TEMP_KORR+menue-1);
            }
            if (key_2==2) {
                  if (menue!=0) eeprom_write_byte(PT_EEPROM_TEMP_KORR+menue-1,
temp_korr[menue-1]);

                  menue=0;
            }
            key_1=0;key_2=0;
            if (menue==0) {
                  switch (start) {
                        case 0: if (((sekunden%10)==WANN_TEMP ) &&
(TW_Get_State_Info()==0xF8)) tempanzeige(1); else  startanzeige(); break;
//                       case 0: startanzeige(); break;
                        case 1: switch(sekunden%10) {
                                    case WANN_DATUM:                  datum();break;
                                    case WANN_TEMP:  if (TW_Get_State_Info()==0xF8)
```

```
tempanzeige(1); else  zeit();break;
                                    default:                zeit();
                                }
                                break;
                        case 2: switch(sekunden%10) {
                                    case WANN_WO_DATUM:         wo_datum(); break;
                                    case WANN_TEMP: if (TW_Get_State_Info()==0xF8)
tempanzeige(1); else  zeit();break;
                                    default:                zeit();
                                }
                                break;
                        case 3: sek_anzeige();
                                break;
                        case 4: tempanzeige(1);
                                break;
                        case 5: datum();
                                break;
                        case 6: wo_datum();
                                break;
                        case 7: debug_anzeige();
                                break;
                        default:startanzeige(); break;
                    }
                } else {
                    if (((tausendstel/100)%2)==0) {
                        segmente[0]=siebensegment(temp_korr[menue-1] % 16);
                        segmente[1]=siebensegment(temp_korr[menue-1] / 16);
                    } else {
                        segmente[0]=255;
                        segmente[1]=255;
                    //
                    }
                    segmente[2]=255;
                    segmente[3]=siebensegment(menue);
                }
                if (merk_sekunde!=sekunden){
                    merk_sekunde=sekunden;
                    if ((sekunden%10)==WANN_TEMP-1) {
                                temp_buff[0]=0b10010001;
                                temp_buff[1]=0;
                                temp_buff[2]=0;
                                TW_Start_Transceiver_With_Data(temp_buff,3);
                    }
                    if ((sekunden%10)==WANN_TEMP) {
                                TW_Get_Data_From_Transceiver(temp_buff,3);
                                tempera.temp1=temp_buff[2];
                                tempera.temp2=temp_buff[1];
                                tempera.temperatur/=16;
                                if ((temp_korr[0]&0b10000000)==0)
                                    tempera.temperatur+=temp_korr[0];
                                else
                                    tempera.temperatur-=255-temp_korr[0];
                                temp32=tempera.temperatur;
                                temp32*=THERM_DECIMAL_STEPS_12BIT;
                                temp32/=1000;
                    }
                }
            }
}
void datum(void){
     segmente[0]=siebensegment(monat % 10);
     segmente[1]=siebensegment(monat / 10);
     segmente[2]=siebensegment(tage % 10);
```

```
	segmente[3]=siebensegment(tage / 10);
}
void zeit(void){
	segmente[0]=siebensegment(minuten % 10);
	segmente[1]=siebensegment(minuten / 10);
	segmente[2]=siebensegment(stunden % 10);
	segmente[3]=siebensegment(stunden / 10);
}
void sek_anzeige(void){
	segmente[0]=siebensegment(sekunden % 10);
	segmente[1]=siebensegment(sekunden / 10);
	segmente[2]=siebensegment(minuten % 10);
	segmente[3]=siebensegment(minuten / 10);
}
void debug_anzeige(void){
	segmente[0]=siebensegment(dcf_debug % 16);
	segmente[1]=siebensegment(dcf_debug / 16);
	segmente[2]=siebensegment(dcfstate % 10);
	segmente[3]=siebensegment(dcfstate / 10);
}
void startanzeige(void){
if (tausendstel<500) {
//	segmente[0]=siebensegment(temp_buff[0] % 16);
//	segmente[1]=siebensegment(temp_buff[0] / 16);
//	segmente[2]=siebensegment(temp_buff[1] % 16);
//	segmente[3]=siebensegment(temp_buff[1] / 16);
//	segmente[0]=siebensegment(tempera.temp1 % 16);
//	segmente[1]=siebensegment(tempera.temp1/ 16);
//	segmente[2]=siebensegment(tempera.temp2 % 16);
//	segmente[3]=siebensegment(tempera.temp2 / 16);
//  segmente[0]=0xff;
//	segmente[1]=siebensegment(temp32 % 10);
//	segmente[2]=siebensegment(temp32 % 100 /10);
//	segmente[3]=siebensegment(temp32 % 1000 /100);
//	segmente[2]=siebensegment(tempo % 16);
//	segmente[3]=siebensegment(tempo / 16);
//	segmente[2]=siebensegment(TW_Get_State_Info() % 16);
//	segmente[3]=siebensegment(TW_Get_State_Info() / 16);
		segmente[0]=siebensegment(sekunden % 10);
		segmente[1]=siebensegment(sekunden / 10);
		segmente[2]=siebensegment(minuten % 10);
		segmente[3]=siebensegment(minuten / 10);
	} else {
		segmente[0]=0b01111111;
		segmente[1]=0b01111111;
		segmente[2]=0b01111111;
		segmente[3]=0b01111111;
	}
}
void tempanzeige(uint8 i){
	if (temp32<0){
		segmente[0]=0b00111111;
		segmente[1]=siebensegment(~(temp32-1) % 10);
		segmente[2]=siebensegment(~(temp32-1) % 100 /10);
		segmente[3]=siebensegment(~(temp32-1 )% 1000 /100);

	} else {
		segmente[0]=0xff;
		segmente[1]=siebensegment(temp32 % 10);
		segmente[2]=siebensegment(temp32 % 100 /10);
		segmente[3]=siebensegment(temp32 % 1000 /100);
	}
}
```

```
void wo_datum(void){
	segmente[0]=siebensegment(tage % 10);
	segmente[1]=siebensegment(tage / 10);
#ifndef LESS_CODE_PROGRAM
	segmente[2]=wochentag(woche) % 256;
	segmente[3]=wochentag(woche) / 256;
#else
	segmente[2]=siebensegment(jahr) % 10;
	segmente[3]=siebensegment(jahr) / 10;

#endif
}

ISR( TIMER1_COMPA_vect)
{
	if (zehntausendstel>dimmer) {
#ifdef HTWM_EDITION
		PORTB |= 0b00000011;//alle LED aus
		PORTC |= 0b00001111;
#else
		PORTC &=0xf0;		//alle LED aus
#endif
	}
	zehntausendstel++;
	if (zehntausendstel>9) {
	zehntausendstel=0;
	tausendstel++;
	//multiplexen
	stelle++;
#ifdef HTWM_EDITION
	PORTB |= 0b00000011;//alle LED aus
	PORTC |= 0b00001111;
#else
	PORTC &=0xf0;		//alle LED aus
#endif

#ifdef HTWM_EDITION
	if (stelle>6) stelle=1;
	if (dp[stelle-1]==0) PORTD=segmente[stelle-1] &= 0b01111111;
			else PORTD=segmente[stelle-1] |= 0b10000000;
	switch (stelle) {
		case 1: PORTC &=0b11110111;break;
		case 2: PORTC &=0b11111011;break;
		case 3: PORTC &=0b11111101;break;
		case 4: PORTC &=0b11111110;break;
		case 5: PORTB &=0b11111101;break;
		case 6: PORTB &=0b11111110;break;
		default: PORTC |=0x0f;
				PORTB |= 0b00000011;//alle LED aus
	}
#else
	if (stelle>4) stelle=1;
	if (dp[stelle-1]==0) PORTD=segmente[stelle-1] |= 0x80;
			else PORTD=segmente[stelle-1] &= 0b01111111;
	switch (stelle) {
		case 1: PORTC |=8;break;
		case 2: PORTC |=4;break;
		case 3: PORTC |=2;break;
		case 4: PORTC |=1;break;
		default: PORTC &=0xf0;
	}
#endif
	//Realisierung der internen Uhr
```

```
    if (tausendstel>999) {
        sekunden++;
        if (dcf_timeout<DCF77_UPDATE) dcf_timeout++;   //zählt ohne dcfsync hoch
        tausendstel=0;
        if (sekunden>59) {minuten++;sekunden=0;
            if (minuten>59) {stunden++;minuten=0;
                if (stunden>23) {stunden=0;tage++;woche++;if (woche>7) woche=1;

#ifdef LESS_CODE_PROGRAM
                                if (monat < 8)
                                {
                                    if (  ((monat==2) && (tage>28) && ((jahr%4)!=0))
                                    || ((monat==2) && (tage>29) && ((jahr%4)==0))
                                    || ((monat % 2 == 0) && (tage > 30))
                                    || ((monat % 2 != 0) && (tage > 31))) {
                                            jahr++;
                                            monat=1;
                                        }
                                }
                                else if ((monat % 2 != 0 && tage > 30)
                                        || (monat % 2 == 0 && tage > 31)) {
                                            jahr++;
                                            monat=1;
                                        }
#else
                                if (((monat==1) && (tage>31))
                                    ||((monat==2) && (tage>28) && ((jahr%4)!=0))
                                    ||((monat==2) && (tage>29) && ((jahr%4)==0))
                                    ||((monat==3) && (tage>31))
                                    ||((monat==4) && (tage>30))
                                    ||((monat==5) && (tage>31))
                                    ||((monat==6) && (tage>30))
                                    ||((monat==7) && (tage>31))
                                    ||((monat==8) && (tage>31))
                                    ||((monat==9) && (tage>30))
                                    ||((monat==10) && (tage>31))
                                    ||((monat==11) && (tage>30))
                                    ||((monat==12) && (tage>31))) {
                                    monat++;
                                    tage=1;
                                    }
#endif
                                    if (monat>12)
                                    {
                                        jahr++;
                                        monat=1;
                                    }
                    }
                }
        }
    }
                                        //Abfrage des DCF-Impulses , Realisierung Blinken
   if ((tausendstel%10)==0) {                       //alle 10ms
#ifdef HTWM_EDITION
        if ((PINB & 0b00000100)==0) {
#else
        if ((PINB & 0b00000001)==0) {
#endif
            if(dcf_timeout>=DCF77_UPDATE) { dp[2]=0; }          ///LED ein blinkt wie
DCF-Impuls bei Signalsuche
            dcf77ct++;
        }
        else {
```

```
                if (dcf_timeout>=DCF77_UPDATE) { dp[2]=1; } ///LED aus blinkt bei Signalsuche
                if (dcf77ct>0)  dcf();
            }
            if (dcf_timeout<DCF77_UPDATE) {if ((sekunden%2)==0) dp[2]=1; else dp[2]=0;}
        }
                                                //Tastenabfrage alle 10ms
        if ((tausendstel%10)==1) {      //alle 10ms, aber um 1ms versetzt wegen Rechenlastver-
teilung
            static uint16 key_count1=0;
            static uint16 key_count2=0;
            if ((PINB & 0b00000100)==0){
                key_count1++;
            } else {
                if (key_count1>1) {
                    key_1=1;                                        //1=normale
Betätigung
                    if (key_count1>100)    key_1=2;                 //2=lange Betätigung
                }
                key_count1=0;
            }
            if ((PINB & 0b00000010)==0){
                key_count2++;
            } else {
                if (key_count2>1) {
                    key_2=1;                                        //1=normale
Betätigung
                    if (key_count2>100)    key_2=2;                 //2=lange Betätigung
                }
                key_count2=0;
            }
        }
}
}
int siebensegment(int s){
    int i=0;
    if (s==16) i=0;
        else {
          s=s & 0b1111;
          switch (s)
        {
#ifdef HTWM_EDITION
                case 0:i=~0b11000000;break;                 //0b00111111
                case 1:i=~0b11111001;break;                 //0b00000011
                case 2:i=~0b10100100;break;                 //0b01101101
                case 3:i=~0b10110000;break;                 //0b01100111
                case 4:i=~0b10011001;break;                 //0b01010011
                case 5:i=~0b10010010;break;                 //0b01110110
                case 6:i=~0b10000010;break;                 //0b01111110
                case 7:i=~0b11111000;break;                 //0b00100011
                case 8:i=~0b10000000;break;                 //0b01111111
                case 9:i=~0b10010000;break;                 //0b01110111
                case 10:i=~0b10001000;break;              //0b01111011
                case 11:i=~0b10000011;break;              //0b01011110
                case 12:i=~0b10000110;break;              //0b00111100
                case 13:i=~0b10100001;break;              //0b01001111
                case 14:i=~0b10000110;break;              //0b01111100
                case 15:i=~0b10001110;break;              //0b01111000
#else
                case 0:i=0b11000000;break;                  //0b00111111
                case 1:i=0b11111001;break;                  //0b00000011
                case 2:i=0b10100100;break;                  //0b01101101
                case 3:i=0b10110000;break;                  //0b01100111
                case 4:i=0b10011001;break;                  //0b01010011
```

```
                case 5:i=0b10010010;break;                  //0b01110110
                case 6:i=0b10000010;break;                  //0b01111110
                case 7:i=0b11111000;break;                  //0b00100011
                case 8:i=0b10000000;break;                  //0b01111111
                case 9:i=0b10010000;break;                  //0b01110111
                case 10:i=0b10001000;break;                 //0b01111011
                case 11:i=0b10000011;break;                 //0b01011110
                case 12:i=0b10000110;break;                 //0b00111100
                case 13:i=0b10100001;break;                 //0b01001111
                case 14:i=0b10000110;break;                 //0b01111100
                case 15:i=0b10001110;break;                 //0b01111000
#endif
        }
        }
        return i;
}

uint16 wochentag(uint8 wt){
        uint16 i=0;
        wt=wt & 0b0111;
        switch  (wt) {
                case 0: i= 0; break;
                case 1: i= 0b1100100010100011; break;
                case 2: i= 0b1010000111101111; break;
                case 3: i= 0b1100100011101111; break;
                case 4: i= 0b1010000110100011; break;
                case 5: i= 0b1000111010101111; break;
                case 6: i= 0b1001001010001000; break;
                case 7: i= 0b1001001010100011; break;
        }
        return i;
}

void dcf(void){

void dcf_reset(void){
                dcfcheck=0;dcfstate=0;dcf77ct=0;dcf_status=0;dcf_min=0;
                dcf_std=0;dcf_tag=0;dcf_woch=0;dcf_monat=0;dcf_jahr=0;
}
void dcf_fehler(void){
                dcfok=0;
                dcf_reset();
}



void dcf_scan(unsigned char pit){
#ifdef LESS_CODE_PROGRAM
        char nocheck = 0;

#endif
        dcfstate++;
        if (pit==1){
                switch (dcfstate-1) {


#ifndef LESS_CODE_PROGRAM
                case 15:        dcf_status+=1;dcfcheck++;break;         //R Rufbit
                case 16:        dcf_status+=2;dcfcheck++;break;         //A1 Ankündigung
MEZ<->MESZ
                case 17:        dcf_status+=4;dcfcheck++;break;         //Z1 0 bei MEZ 1
bei MESZ
                case 18:        dcf_status+=8;dcfcheck++;break;         //Z2 1 bei MEZ 0
```

```
bei MESZ
                case 19:      dcf_status+=0x10;dcfcheck++;break;          //A2
Schaltsekunde
                case 20:      dcf_status+=0x20;dcfcheck++;break;          //Startbit -- muss
gesetzt sein
                case 21:      dcf_min+=1;dcfcheck++;break;
                case 22:      dcf_min+=2;dcfcheck++;break;
                case 23:      dcf_min+=4;dcfcheck++;break;
                case 24:      dcf_min+=8;dcfcheck++;break;
                case 25:      dcf_min+=10;dcfcheck++;break;
                case 26:      dcf_min+=20;dcfcheck++;break;
                case 27:      dcf_min+=40;dcfcheck++;break;
                case 28:      dcfcheck++;    if((dcfcheck%2)!= 0) dcf_fehler();break;
                case 29:      dcf_std+=1;dcfcheck++;break;
                case 30:      dcf_std+=2;dcfcheck++;break;
                case 31:      dcf_std+=4;dcfcheck++;break;
                case 32:      dcf_std+=8;dcfcheck++;break;
                case 33:      dcf_std+=10;dcfcheck++;break;
                case 34:      dcf_std+=20;dcfcheck++;break;
                case 35:      dcfcheck++;if((dcfcheck%2)!= 0) dcf_fehler();break;
                case 36:      dcf_tag+=1;dcfcheck++;break;
                case 37:      dcf_tag+=2;dcfcheck++;break;
                case 38:      dcf_tag+=4;dcfcheck++;break;
                case 39:      dcf_tag+=8;dcfcheck++;break;
                case 40:      dcf_tag+=10;dcfcheck++;break;
                case 41:      dcf_tag+=20;dcfcheck++;break;
                case 42:      dcf_woch+=1;dcfcheck++;break;
                case 43:      dcf_woch+=2;dcfcheck++;break;
                case 44:      dcf_woch+=4;dcfcheck++;break;
                case 45:      dcf_monat+=1;dcfcheck++;break;
                case 46:      dcf_monat+=2;dcfcheck++;break;
                case 47:      dcf_monat+=4;dcfcheck++;break;
                case 48:      dcf_monat+=8;dcfcheck++;break;
                case 49:      dcf_monat+=10;dcfcheck++;break;
                case 50:      dcf_jahr+=1;dcfcheck++;break;
                case 51:      dcf_jahr+=2;dcfcheck++;break;
                case 52:      dcf_jahr+=4;dcfcheck++;break;
                case 53:      dcf_jahr+=8;dcfcheck++;break;
                case 54:      dcf_jahr+=10;dcfcheck++;break;
                case 55:      dcf_jahr+=20;dcfcheck++;break;
                case 56:      dcf_jahr+=40;dcfcheck++;break;
                case 57:      dcf_jahr+=80;dcfcheck++;break;
                case 58:      dcfcheck++;if((dcfcheck%2)!= 0) dcf_fehler();break;
                default:{}
        }
#else
                case 15:      dcf_status+=1;break;              //R Rufbit
                case 16:      dcf_status+=2;break;              //A1 Ankündigung MEZ<-
>MESZ
                case 17:      dcf_status+=4;break;              //Z1 0 bei MEZ 1 bei MESZ
                case 18:      dcf_status+=8;break;              //Z2 1 bei MEZ 0 bei MESZ
                case 19:      dcf_status+=0x10;break;       //A2 Schaltsekunde
                case 20:      dcf_status+=0x20;break;       //Startbit -- muss gesetzt sein
                case 21:      dcf_min+=1;break;
                case 22:      dcf_min+=2;break;
                case 23:      dcf_min+=4;break;
                case 24:      dcf_min+=8;break;
                case 25:      dcf_min+=10;break;
                case 26:      dcf_min+=20;break;
                case 27:      dcf_min+=40;break;
                case 28:      nocheck = 1; if(((dcfcheck+1)%2)!= 0) dcf_fehler();break;
                case 29:      dcf_std+=1;break;
                case 30:      dcf_std+=2;break;
```

```
                case 31:      dcf_std+=4;break;
                case 32:      dcf_std+=8;break;
                case 33:      dcf_std+=10;break;
                case 34:      dcf_std+=20;break;
                case 35:      nocheck = 1; if(((dcfcheck+1)%2)!= 0) dcf_fehler();break;
                case 36:      dcf_tag+=1;break;
                case 37:      dcf_tag+=2;break;
                case 38:      dcf_tag+=4;break;
                case 39:      dcf_tag+=8;break;
                case 40:      dcf_tag+=10;break;
                case 41:      dcf_tag+=20;break;
                case 42:      dcf_woch+=1;break;
                case 43:      dcf_woch+=2;break;
                case 44:      dcf_woch+=4;break;
                case 45:      dcf_monat+=1;break;
                case 46:      dcf_monat+=2;break;
                case 47:      dcf_monat+=4;break;
                case 48:      dcf_monat+=8;break;
                case 49:      dcf_monat+=10;break;
                case 50:      dcf_jahr+=1;break;
                case 51:      dcf_jahr+=2;break;
                case 52:      dcf_jahr+=4;break;
                case 53:      dcf_jahr+=8;break;
                case 54:      dcf_jahr+=10;break;
                case 55:      dcf_jahr+=20;break;
                case 56:      dcf_jahr+=40;break;
                case 57:      dcf_jahr+=80;break;
                case 58:      nocheck = 1; if(((dcfcheck+1)%2)!= 0) dcf_fehler();break;
                default:{}
          }

          if (nocheck == 0)
                dcfcheck++;
#endif
     } else {
          switch (dcfstate-1) {
                case 28:      if((dcfcheck%2)!= 0) dcf_fehler();break;                //P1
                case 35:      if((dcfcheck%2)!= 0) dcf_fehler();break;                //P2
                case 58:      if((dcfcheck%2)!= 0) dcf_fehler();break;                //P3
          default:{}
          }
     }
     if (dcf77ct>149) {
          if
((dcf_min<60)&&(dcf_std<24)&&(dcf_tag<32)&&(dcf_monat<13)&&(dcf_jahr>8)&&dcf_status
>0x20) {
                if ((dcf_min==(letzte_min+1)&&(dcf_std==letzte_std)) ||
                     ((dcf_min==0) && (letzte_min==59) && (dcf_std==letzte_std+1)) ||
                          ((dcf_min==0) && (letzte_min==59) && (letzte_std==23) &&
(dcf_woch==letzte_woche+1)) ||
                               ((dcf_min==0) && (letzte_min==59) && (letzte_std==23) &&
(dcf_woch==1) && (letzte_woche==7))){
                     tausendstel=0;
                     dcfok=1;
                     sekunden=0;
                     minuten=dcf_min;
                     stunden=dcf_std;
                     tage=dcf_tag;
                     woche=dcf_woch;
                     monat=dcf_monat;
                     jahr=dcf_jahr;
                     if (start==0) start=1;
                     dcf_timeout=0;
```

```
                } else dcfok=0;
                letzte_min=dcf_min;
                letzte_std=dcf_std;
                letzte_woche=dcf_woch;
                dcf_reset();
      } else dcf_fehler();
     }
}

void dcf_stoerung(void){
//leer - Impulse unter 70ms werden ignoriert
}
           dcf_debug=dcf77ct;
           if (dcf77ct>209) dcf_fehler();
           if ((dcf77ct>187) && (dcf77ct<210)) dcf_scan(0);            //tritt nur beim minute-
nimpuls auf; für Sek 58 (P3)
//           if ((dcf77ct>192) && (dcf77ct<190)) dcf_fehler();
           if ((dcf77ct>149) && (dcf77ct<188)) dcf_scan(1);            //tritt nur beim minute-
nimpuls auf; für P3
           if ((dcf77ct>96) && (dcf77ct<150))  dcf_fehler();
           if ((dcf77ct>86) && (dcf77ct<97))      dcf_scan(0);
           if ((dcf77ct>85) && (dcf77ct<87))      dcf_fehler();
           if ((dcf77ct>70) && (dcf77ct<86))      dcf_scan(1);
           if ((dcf77ct>6) && (dcf77ct<71))       dcf_fehler();
           if (dcf77ct<7)                              dcf_stoerung(); else dcf77ct=0;
}
```

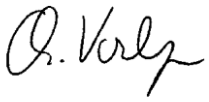
Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Zwönitz, den 29.06.2014



Christhard Vorberg