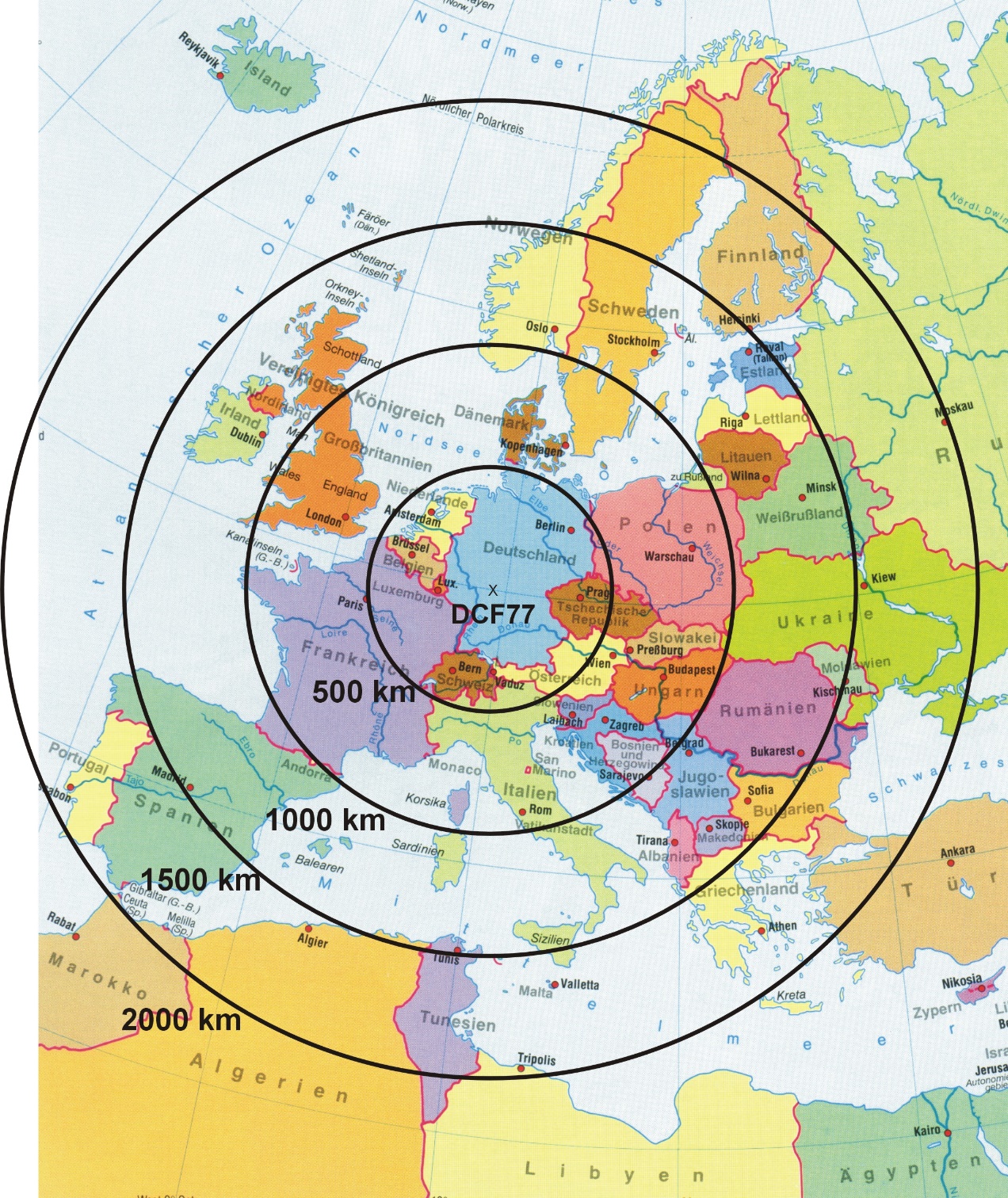
**Projekt DCF77**

**Ron Buntschu, Nicolas Takagawa**



**Projektname:** Digitalprojekt DCF77

**Studierende:** Ron Buntschu und Nicolas Takagawa

**Dozent:** Torsten **Mähne**

**Institution:** Berner Fachhochschule Technik und Informatik

**Studiengang:** Elektro- und Telekommunikation

**Klasse:** E1d

**Semester:** Zwei

Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc448839111)

[**Arbeitsvorbereitung und Aufteilung** 2](#_Toc448839112)

[**Arbeitsplanung** 2](#_Toc448839113)

[**Aufteilung** 3](#_Toc448839114)

[**Teil 1: Anzeige und Wecker (Ron Buntschu)** 4](#_Toc448839115)

[**Anzeige 7-Segment** 4](#_Toc448839116)

[**LED\_Anzeige** 6](#_Toc448839117)

[**LED\_und\_7Seg\_Anzeige\_Final** 6](#_Toc448839118)

[**Erkenntnisse und Fehler** 7](#_Toc448839119)

[**Teil 2: Decodierung (Nicolas Takagawa)** 8](#_Toc448839120)

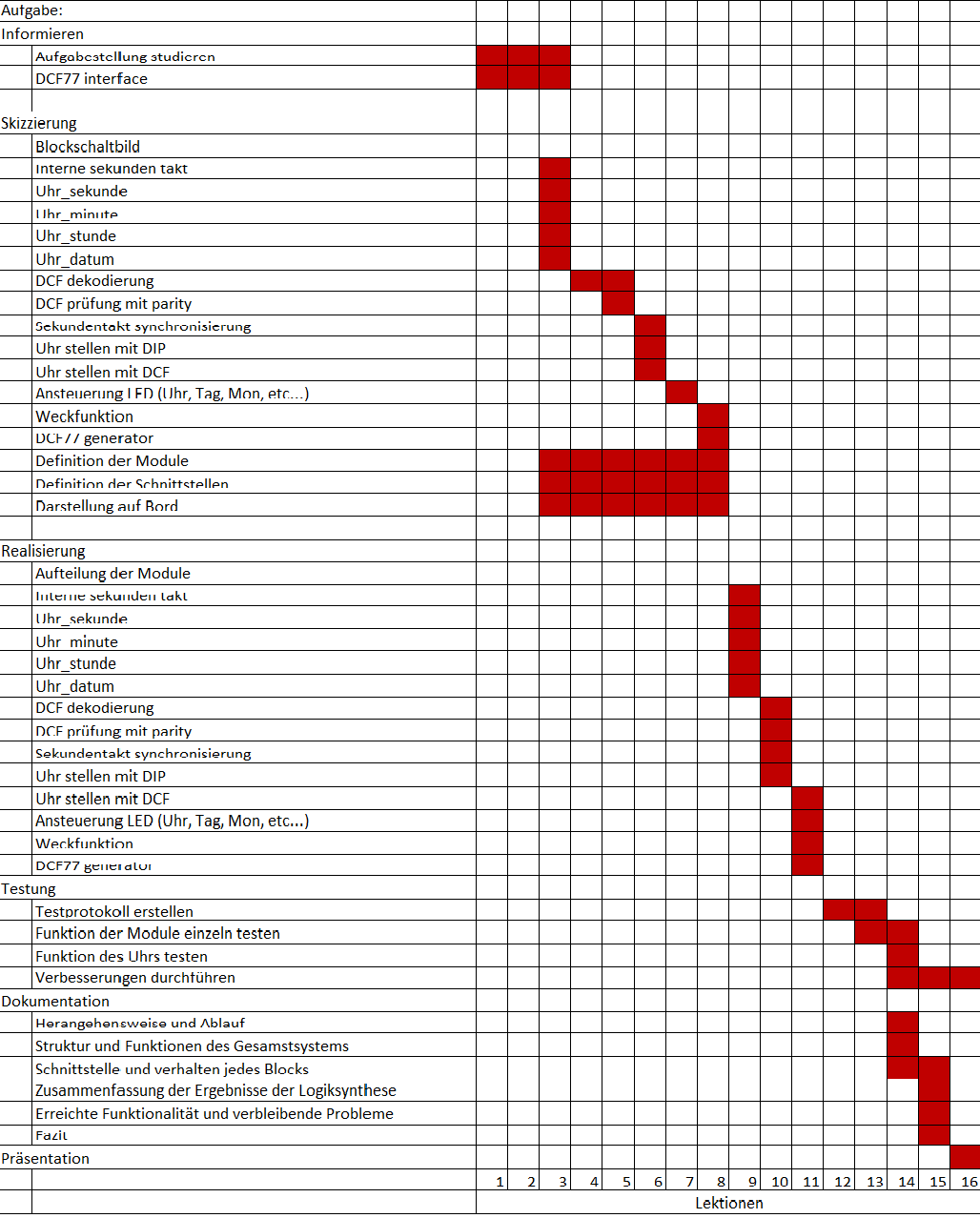
# **Arbeitsvorbereitung und Aufteilung**

## **Arbeitsplanung**

Um uns in das Projekt einarbeiten zu können, studierten wir zu Beginn den Auftrag, sowie die Informationen zur DCF77.

Bereits kurz nach dem Erhalt des Auftrags machten wir uns an die Zeitplanung und überlegten uns, in welche Teile wir die Gestellte Aufgabe Zerlegen können.

Die Zeitplanung an sich war etwas schwierig, da wir Logisim noch wenig kannten und daher nicht genau wussten, wie Zeitintensiv die einzelnen Teilaufgaben sind. Den Zeitplan einzuhalten war aber auch nicht immens wichtig, solange das Projekt rechtzeitig fertig wird. Wir haben während dem Arbeiten gemerkt welchen Aufwand wir in etwa betreiben müssen, sodass wir das Projekt zum Abgabetermin fertiggestellt haben.



## 

## **Aufteilung**

Als nächsten Schritt überlegten wir uns, welche Teilaufgaben unterteilt werden können und wieviel sie Aufwand sie jeweils beanspruchen.

Wir entschieden uns für eine Unterteilung zwischen Decodierung und Darstellung. Zur Darstellung soll ausserdem noch die Programmierung einer Weckfunktion dazu kommen.

Die Decodierung soll zur Aufgabe von Nicolas Takagawa, gelernter Elektroniker, werden, da Elektroniker sich im Bereich der Dekodierung besser auskennen. Die Darstellung wird zur Aufgaben von Ron Buntschu.

# **Teil 1: Anzeige und Wecker (Ron Buntschu)**

## **Anzeige 7-Segment**

Als erstes Logisim Projekt habe ich damit begonnen, eine funktionierende 7-Segmentanzeige zu Programmieren. Nicolas hat mir gesagt, dass er von seiner Seite her die Bits für Stunden, Minuten und so weiter bringen wird, ich dann diese Verarbeiten und auf dem Display anzeigen soll.   
wir entschieden Gemeinsam, dass standardmässig die Stunden und Minuten angezeigt werden sollen. Je nachdem, welche Taste dann gedrückt wird, sollen Minuten-Sekunden, Tag-Monat, oder das Jahr angezeigt werden.  
Zu Beginn des Projekts war ich noch sehr unerfahren mit Logisim, dies ist der Grund, weshalb ich bei den ersten Blöcken noch nicht mit Bussen arbeitete. Ich finde aber man soll in einem Projekt auch die Fortschritte des Programmierers sehen. Deshalb habe ich nicht nochmal alles korrigiert, die Funktionalität war schliesslich vorhanden.

**Multiplex\_Std\_Tag:** Als erstes machte ich verschiedene Multiplexer, um mittels Tastendruck zwischen den Verschiedenen Bits zu unterscheiden und jeweils die Zusammengehörigen auf den 7-SegmentAnzeigen auszugeben. Dieser Block schaltet zwischen Stunden und dem Datumteil Tag um.

**Multiplex\_min\_Monat:** Dieser Subblock sorgt für die Umschaltung zwischen Minuten und dem Monat.

**Multiplex\_Jahr\_Sek:** Der Multiplex\_Jahr\_Sek schaltet zwischen Jahr und Sekunden um.

**Verknuepfung\_min\_mon\_jahr:** Ein Multiplexer für die Einzelnen 7-Segmente.

**Testing7Seg:** Dieser Block dient zur Simulation des Programms. Es ist auch der Block welcher später in ein Gesamtprogramm eingefügt werden kann.

**BCD2\_7Seg:** Diesen Teil, haben ich vom Digitalpraktikum 1 übernommen. Er wandelt BCD zu 7-Segment um, sodass die Anzeigen Angeschlossen werden können.

**BIN2\_BCD:** Auch dieser Block stammt aus dem Praktikum. Er wandelt ein Binäres Signal in BCD um.

**BCD\_Teil:** Ein Teil des Blocks BIN2\_BCD.

**Umschaltung\_min:** Dieser Block sorgt für die Umschaltung auf den 7-Segmentanzeigen. So werden die Minuten, welche normalerweise auf den rechten Anzeigen ausgegeben werden, während des Drückens von der Taste Min\_Sek, auf der linken Seite angegeben.

**Multiplex\_Sek:** Subblock im Teil „Multiplex\_Jahr\_Sek“.

**Tastenfehler:** Damit keine Fehler entstehen, beim Druck von mehreren Tasten Gleichzeitig, werden dank diesem Block einfach normal die Stunden und Minuten angezeigt.

**Weckfunktion\_Zeitvergleich:** Diesen Teil haben ich später, als ich mich mit dem Wecker befasste hinzugefügt. Er vergleicht die Bits der aktuellen Stunden und Minuten mit denjenigen, welche beim stellen des Weckers eingestellt werden.

**Blinken\_Wecker:** Falls der Wecker gestellt wird, sollen die 7-Segmentanzeigen blinken, damit man merkt, dass nicht die aktuelle Stunde und Minute angezeigt wird.

**Reset\_Wk\_Seg1, Reset\_Wk\_S3u4, Reset\_Wk\_Seg2:**

Diese Blöcke habe ich noch als letztes, als ich den Wecker schon beinahe fertiggestellt hatte, eingefügt. Sie haben normalerweise keine Funktion. Wenn jedoch aus irgendeinem Grund, beim Stellen des Weckers, Werte angezeigt werden, welche nicht sein dürfen, gibt es einen Reset auf die Blöcke zum Stellen des Weckers.

## **LED\_Anzeige**

Ein Zeitaufwendiger Teil des Projekts war die LED Anzeige. Meine Vorstellung war, dass mit den LEDs geschriebene Buchstaben, passend zu den aktuell angezeigten Werten, erscheinen. Für die Stunden ein „h“, die Minuten ein „m“, Sekunden ein „s“, Tage ein „D“, Monate ein „M“ und für die Jahre ein „Y“. Ich schrieb mir also auf ein Blatt Papier für welche Buchstaben, welche LED leuchten müssen. Ich schrieb dann eine Liste, welch LEDs nicht benötigt werden. Dann schaute ich welche doppelt oder dreifach vorkommen. Die doppelt und dreifach vorkommenden LED habe ich mit OR-Gattern verbunden.  
In Hinsicht auf die Weckfunktion wollte ich, dass wenn der Wecker aktiv ist, die gesamte Anzeige, mit einem Takt, negativ und wieder normal leuchtet. Daher habe ich am Ende die LED, mit einer XOR Verknüpfung, mit dem Takt und der Weckfunktion verbunden.

**h\_und\_m:** Benötigte LED zum Schreiben von h und m.

**m\_und\_s:** Benötigte LED zum Schreiben von m und s.

**D\_und\_M:** Benötigte LED zum Schreiben von D und M.

**Y:** Benötigte LED zum Schreiben von Y.

**Part\_LED\_7Seg\_Buntr1:** Auch dieser Block diente wieder dem Testen, diesmal für die LED-Anzeige. Ich habe auch die bereits fertige 7-Segmentanzeige in den Block integriert. So konnte ich ein erstes Mal sehen, ob sie auch gemeinsam funktionierten.

## **LED\_und\_7Seg\_Anzeige\_Final**

Diese Datei dient der Zusammenführung von LED und 7-Segment, sowie Ergänzungen, um den Wecker zu stellen und das 2kHz Taktsignal auf die benötigten Takte zu reduzieren.

Die Funktionen des Programms auf dem Gecko-Board sind die folgenden.  
Wenn alle Dip-switches ausgeschaltet sind wird im Normalfall die Zeit in Stunden und Minuten angezeigt. dann kann man mit 3 Tasten unter den LEDs, von links nach rechts, einmal Minuten und Sekunden, Tag und Monate und das Jahr, mit den jeweils dazugehörigen Buchstaben auf den LEDs, anzeigen. Mit dem ersten Dip-switch kann der Wecker eingeschaltet werden. wenn aktuelle und eingestellte Zeit übereinstimmen, blinken sämtliche LED. Der zweite Schalter dient zum stellen des Weckers. Wenn er ein ist, kann man, mit den ersten 4 Tasten unter den LEDs, die Ziffern auf den 7-Segmenten verstellen. Der 5. Taster unter den LED (von links) macht, dass die Stunden und Minuten nicht mehr von der Antenne empfangen und ausgegeben werden, sondern die Uhrzeit mit den fortfolgenden Schaltern manuell eingestellt werden kann. Nach dem letzten Schalter kommt noch ein Dip-switch, welcher einen asynchronen reset auf sämtliche Flip-Flops der Anzeige durchführt.

**kHz\_4Hz:** Dieser Block reduziert die Frequenz von 2kHz auf eine kleinere Frequenz von 4Hz oder genau 3.91Hz. Diese seltsame Frequenz kommt daher, dass ich den 9bit counter nicht bis 500, sondern seine maximalen 511, zählen lasse. Das hat einerseits den Grund, dass es gerade einfacher war und keine Rolle spielt. Andererseits nutze ich diese Frequenz zum Blinken der 7-Segmente beim Wecker stellen (und das Blinken der LED bei Weckfunktion). Da ich das Stellen des Weckers mit einem 2Hz Signal realisiere, wollte ich nicht genau die doppelte Frequenz für das Blinken.

Das eingefügte Flip-Flop setzt sich wenn der Counter durchgezählt hat und

nach einem weiteren durchzählen geht es wieder auf 0.

**Wecker\_stellen:** Diese Funktion kommt zum Einsatz, wenn der Schalter zum stellen des Weckers eingeschaltet ist. Damit die Zeit möglichst rasch eingestellt werden kann, habe ich eine Art gewählt, wie man jede 7-Segmentanzeige einzeln Stellen kann. Die Zahlen werden nun auf jeder Anzeige bei einem andauernden Tastendruck hochgezählt. Das knifflige dabei waren die Stunden. Ich musste die Resets so vorsehen, dass niemals eine höhere Zahl als die 23 angezeigt wird. Wenn also die Zehner der Stunde bereits auf 2 gestellt sind, darf man die Einer nicht mehr höher als auf 3 Stellen können. Wenn hingegen zuerst die Einer auf 4 oder höher gestellt wurden, dürfen die Zehner nicht mehr höher als auf 1 gehen.

Mit dem counter9bit wird der Wert der einzelnen Segmente gezählt. Bei jedem Impuls von kHz\_2Hz wird, je nach gedrückter Taste, eine 1 oder eine 10 addiert.

**kHz\_2Hz:** Hier wird das 2kHz Signal gewandelt, in ein Signal, welches immer wenn der counter auf tausend gezählt hat, einen kurzen (2kHz) Impuls generiert.

**counter9bit:** Diesen Block habe ich von Nicolas übernommen und für meine Verwendung modifiziert. Ich habe ihn so geändert, dass ich 10 oder 1 addieren kann. Ausserdem habe ich die Flip-Flops in ein Gatter, mit Enable Eingang, zusammengefasst. Zusätzlich habe ich, bei allen Flip-Flops in meiner Schaltung, einen asynchronen Reset eingebaut.

**counter13bit:** Wie bereits beim counter9bit habe ich auch diesen Teil von Nicola übernommen und abgeändert.

## **Erkenntnisse und Fehler**

Während der Arbeit am Projekt lernte ich sehr viel. Ich lernte das Programm Logisim kennen, wie ich gewisse Grundschaltungen aufbauen muss und konnte mich auch nochmals intensiv mit den Logischen Gattern auseinandersetzen. Die Theorie zu sämtlichen Logischen Schaltungen wird, bei einem solchen Projekt, viel klarer und man weiss danach auch für nächste arbeiten besser, was man wie realisieren kann.   
Ich konnte feststellen, was clock-gateing bedeutet und welche Auswirkungen es zur Folge hat. Als ich bereits dachte mein Programm sei fertig und das seltsame blinken der LEDs sei, weil mein Gecko4education kaputt ist, erfuhr ich, dass der Grund dafür eigentlich ein gated-clock war. Ich führte das Taktsignal durch Verknüpfungen und erst danach auf die Flip-Flops. Die führte zu Verzögerungen und einem unregelmässigen Blinken. Ich konnte mit einigen Überlegungen das Programm verändern und es war toll nun zu sehen, wie das Programm nun sauber funktionierte.

# **Teil 2: Decodierung (Nicolas Takagawa)**

In diesem Teil wurde das Funkuhrsystem realisiert. Dieser Teil besteht aus sechs Teilen:

**DCF77 parallelisierung:** Das DCF77-Signal wird eingelesen und in 59bit Schieberegister, für das weitere Verwendung der Daten, eingeschoben.

**Dekodierung:** Die eingelesenen Daten wird in Binär umkodiert

**Datenprüfung:** Die Daten werden auf Fehler überprüft. Die Prüfung umfasst den Paritätsprüfung und die sogenannte Plausibilitätsprüfung.

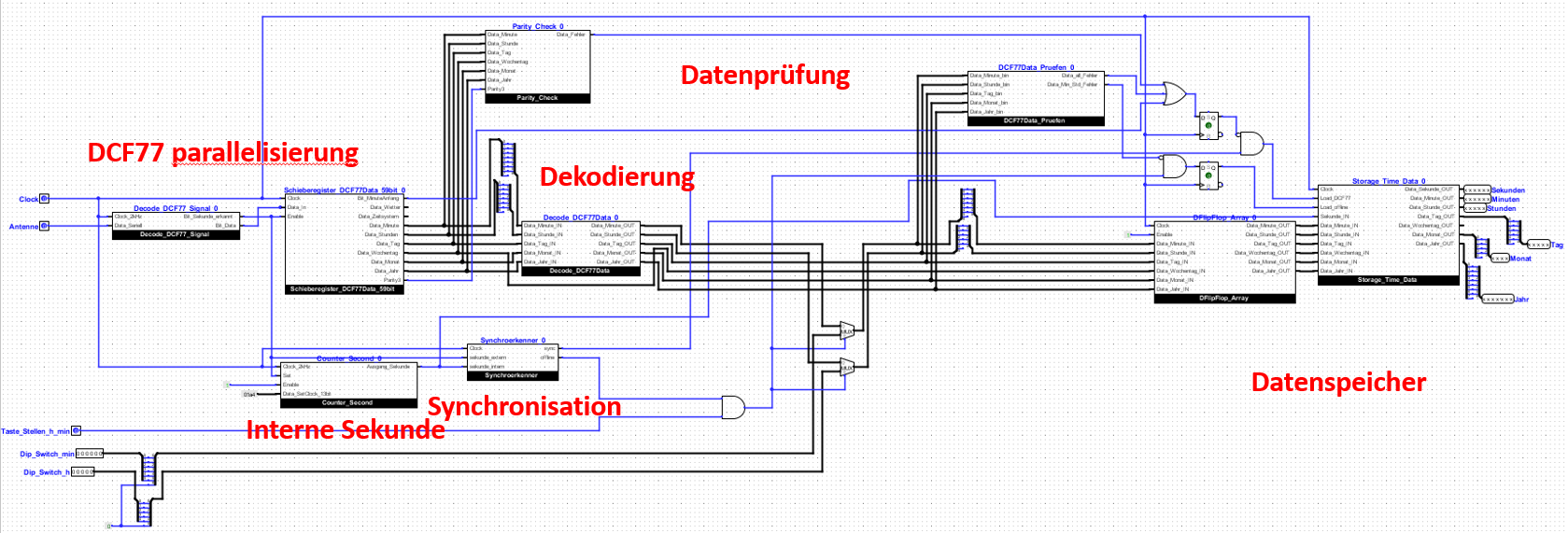
Nur wenn keine Fehler aufgefunden werden kann, soll die Uhr synchronisiert werden.

**Interne Sekunde:** Offline-Sekundentaktgeber. Sie wird mit dem DCF77-Sekundensignal synchronisiert.

**Synchronisation:** Überprüft ob es sich bei den DCF77 einen Minutenübergang handelt. Auch wird es erkannt, ob das Gerät offline von den DCF77 ist.

Wenn das Gerät offline ist, soll es möglich sein, das Uhr manuell mittels DIP-Schalter zu stellen

**Datenspeicher:** Die Gültigen Zeitdaten werden hier gespeichert und ausgegeben. Gleichzeitig wird in diesem Teil die Sekunde, Minute und die Stundendaten inkrementiert.



**Schnittstelle:**

Input:

Clock: Taktsignal

Antenne: DCF77-Signal-Eingang

Taste\_Stellen\_h\_min: Taste zu den manuellen Stellen von Zeitdaten (nur Minute und Stunde)

Dip\_Switch\_min: Minutendaten-Eingang zu den manuellen Stellen; 6bit

Dip\_Switch\_h: Stundendaten-Eingang zu den manuellen Stellen; 5bit

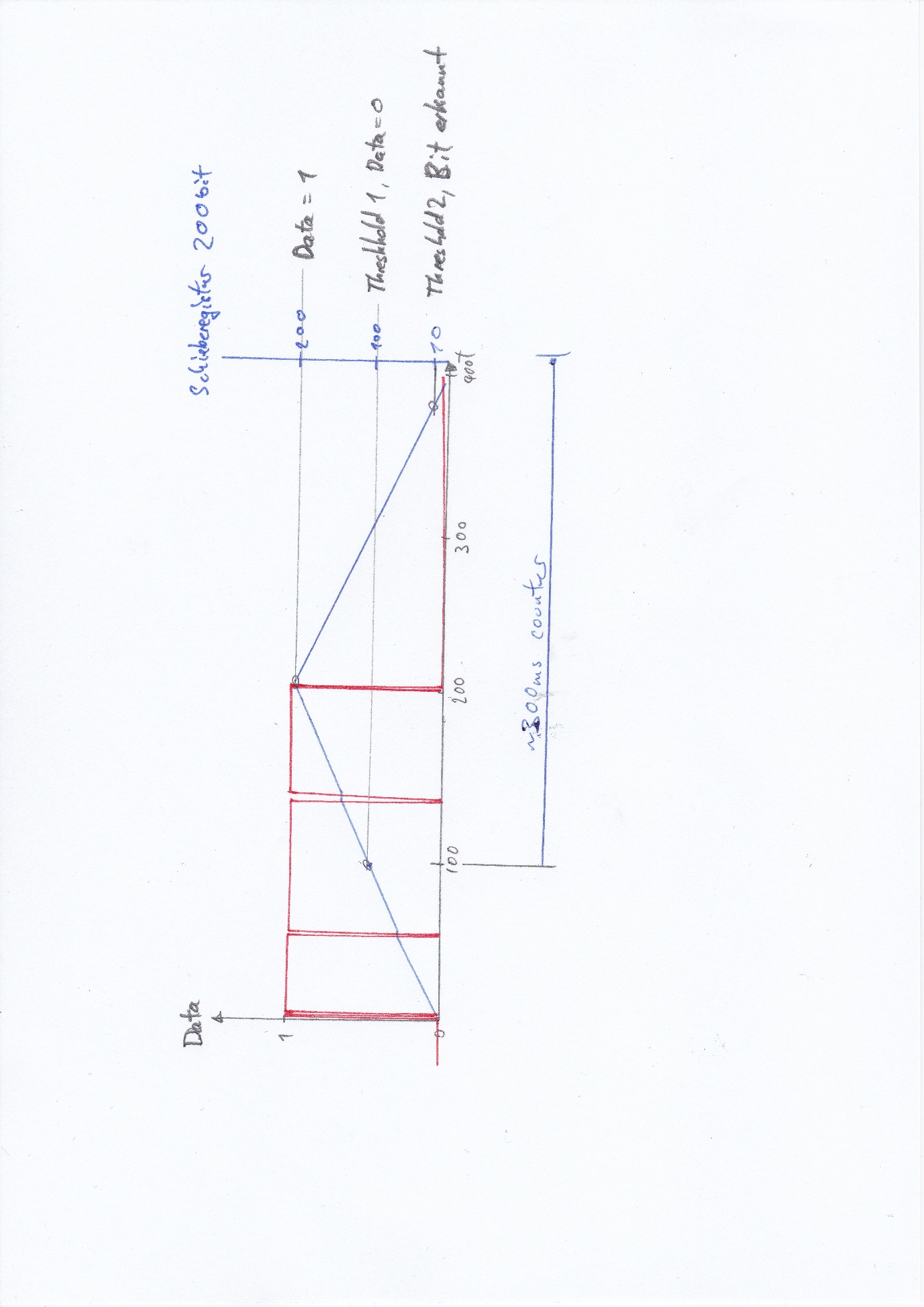
Output:

Sekunden: Sekundendaten; 6bit

Minuten: Minutendaten; 6bit

Stunden: Stundendaten; 5bit

Tag: Kalendertagdaten; 5bit

Monat: Monatsdaten; 4bit

Jahr: Jahrdaten (zweistellig); 7bit

## **Teil: DCF77 parallelisierung**

**Decode\_DCF77\_Signal:**

Die Erkennung des DCF77-Signals wurde mit einer 200bit Schieberegister realisiert, die nach jeder 1ms den momentanen Wert der Antennenleitung reinschiebt.

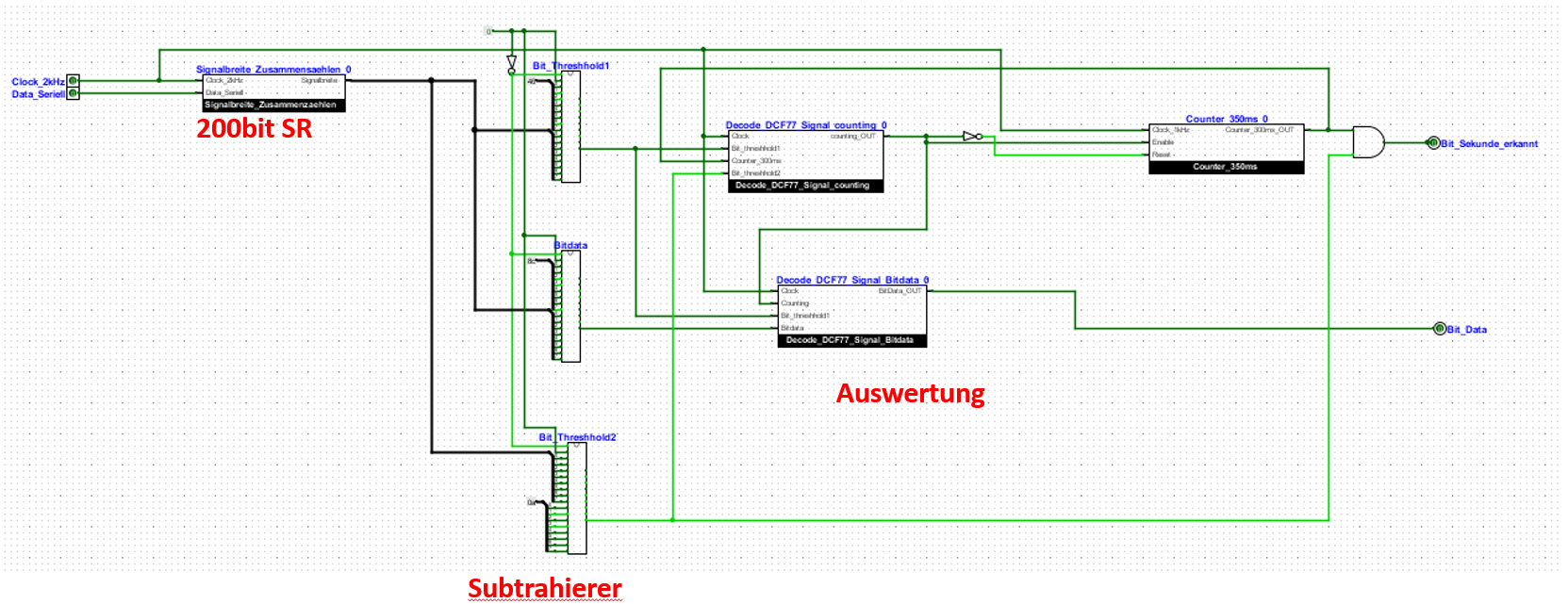
Die Werten in dem Schieberegister wird Bitweise zusammengezählt und die Summe benutzt um Aussagen über DCF77-Daten zu machen:

Der Vorteil bei diesem Methode ist, dass robuster gegen schlechtere Signalen sind.

Wird die erste Threshhold „Threshhold1“ überschritten, so heisst es, dass es möglicherweise um einen Signal „0“ handelt und der Daten-Ausgang wird auf Null geschaltet. Gleichzeitig wird ein Counter gestartet, die etwa auf 300ms Zählt.

Wird in Schieberegister einen Wert von ungefähr 200 erreicht, so heisst es, dass es sich möglicherweise um einen Signal mit den Wert „1“ handelt.

Diese erkannten Daten werden erst mit Bit\_Sekunde\_erkannt freigegeben, wenn bei den Zeitpunkt wo der Counter fertiggezählt hat, den Wert für Threshhold2 unterschritten wurde. Wenn das nicht der Fall ist, wird herausgegangen, dass es sich um fehlerhaftes Signal handelt.

****

Da in der Realität die Signalbreite immer von den idealen Breite abweicht wurde die Werten für Threshholds und Counter angepasst:  
Threshhold1: 70

Bitdata: 140

Threshold2: 10

Counter: 350ms

Interface:  
Input:

Clock\_2kHz: Systemtakteingang

Data\_Seriell: Antennensignaleingang

Output:

Bit\_Sekunde\_erkannt: Gibt „1“ für einen Takt, wenn eine DCF77-Daten erkannt wurde.

Bit\_Data: Den Wert der DCF77-Daten.

**Signalbreite\_Zusammenzaehlen:**

Besteht aus 200bit Schieberegister und Addiernetzwerk, die die einzelnen Bitwerte zusammenzählt.  
Jede 1ms wird einen Wert geladen.

**Decode\_DCF77\_Signal\_counting:** Hier wird entschieden, wann der Counter losgezählt werden soll.

Nach einer beendung der Counter muss gewartet werden, bis die Unterschreitung der Threshhold2 erfolgt wird.

**Signal\_Bitdata:** Gibt den Wert der DCF77-Data aus.

Wenn der Threshhold1 erreicht wird, wird er auf „0“ gesetzt.

Dieser Block behält den Wert bis auf den nächsten anlauf.

**Counter\_350ms:** Zählt mittels 9bit-Counter bis 350ms und gibt dann einen „1“ für einen Takt aus.