Projekt DCF77

Ron Buntschu, Nicolas Takagawa

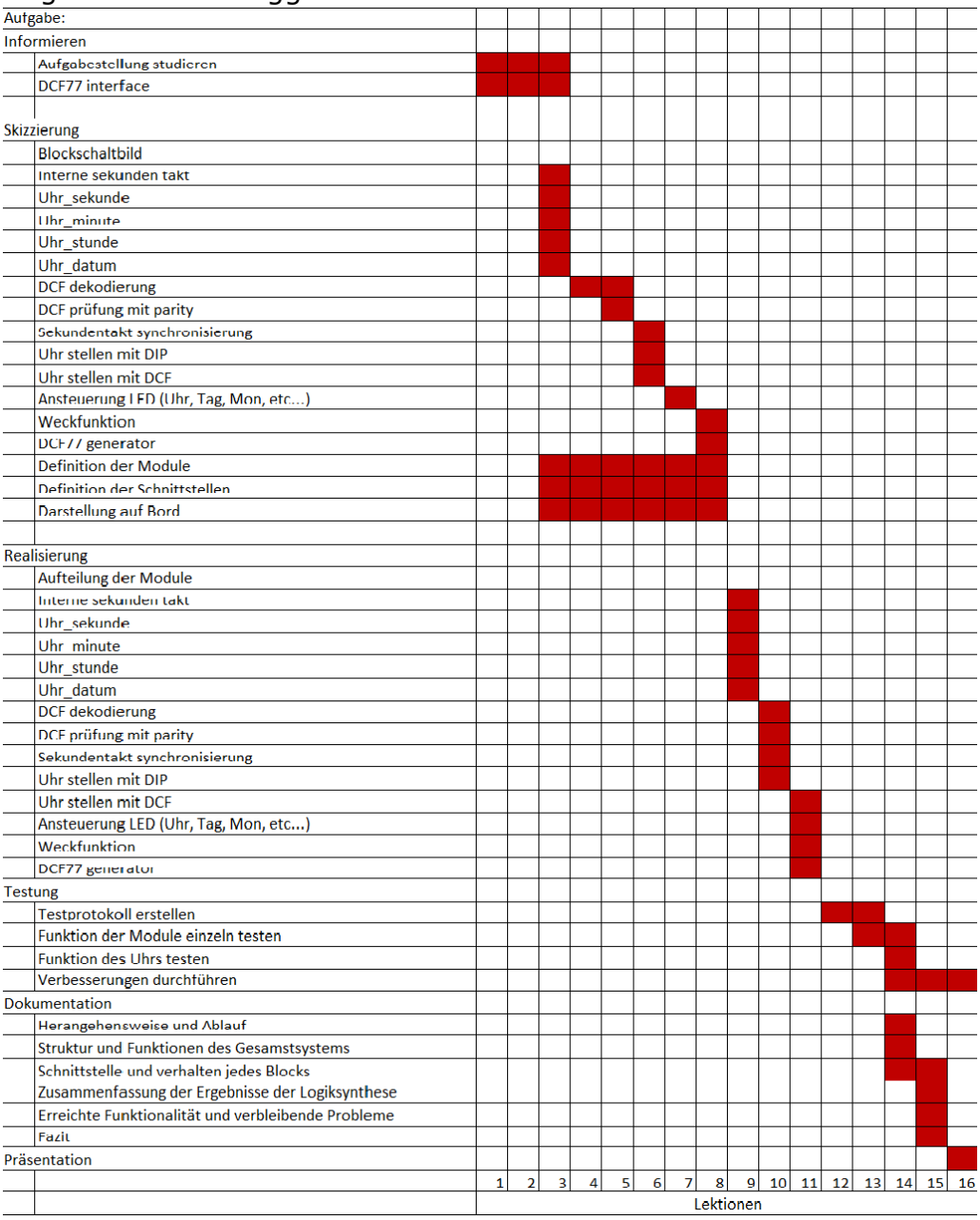
Projektname: Digitalprojekt DCF77
Studierende: Ron Buntschu und Nicolas Takagawa
Dozent: Torsten Mähne
Institution: Berner Fachhochschule Technik und Informatik
Studiengang: Elektro- und Telekommunikation
Klasse: E1d
Semester: Zwei

Inhaltsverzeichnis

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.

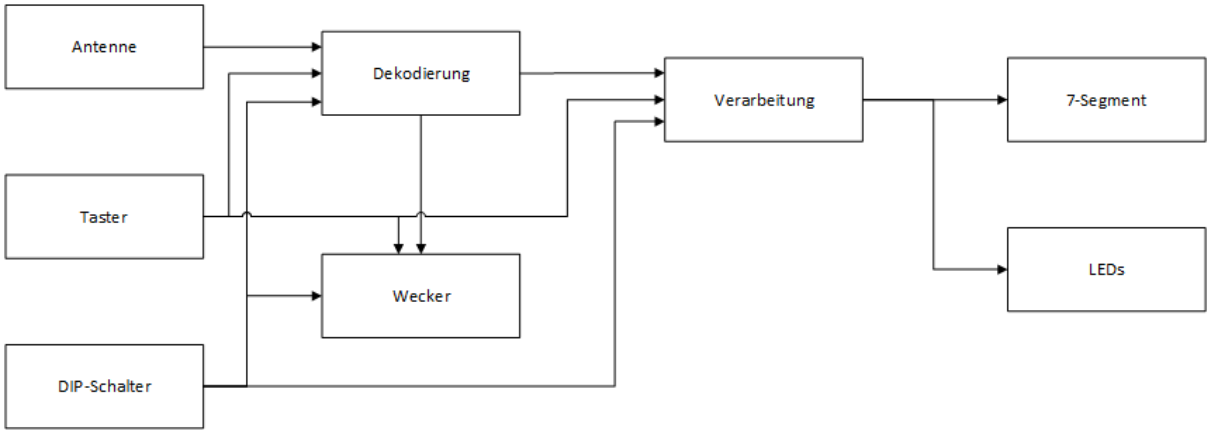


Aufteilung

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.

Gesamtsystem und Bedienungsanleitung



Bedienungsanleitung

Nach laden des Programms:	Nach 2 Minuten sollte sich bei gutem Antennenempfang die Zeit synchronisiert haben und Stunden-Minuten werden angezeigt.
Alle Dip-Switch off:	Taste 1 (von links) unter LED: Anzeige Minuten-Sekunden
	Taste 2 unter LED: Anzeige Tag-Monat
	Taste 3 unter LED: Anzeige Jahr
Dip-Switch links:	1. Schalter: Wecker Ein-Aus
	2. Schalter: Wecker Stellen --> Wecker mit Tasten 1-4 unter LED Stellbar, durch anhaltenden Druck
	3.-7. Schalter: Bits Manuelles Stellen der Stunden; Schalter 3= MSB

Dip-Switch rechts:	1.-6. Schalter: Bits Manuelles Stellen der Minuten; Schalter 1= MSB
	7.Schalter: Asynchroner Reset Anzeige
5.Taste unter LED:	Wenn Antennensignal Offline: Laden der Manuell eingestellten Zeit

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, sodass ich diese verarbeiten und auf dem Display ausgeben kann.
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.

Schnittstelle Hauptschaltung:

Input:	Takt_Wecker_Stellen:	Takt blinken der Segmente beim Wecker stellen.
	Taste_Gecko_Min_Sek:	Taste_gecko_1
	Taste_Gecko_Tag_Monat:	Taste_gecko_2
	Taste_Gecko_Jahr:	Taste_gecko_3
	Schalter_Gecko_Wecker_Stellen:	Wecker_Ein_und_Std Schalter Nr.2
	Schalter_Gecko_Wecker_Ein_Aus:	Wecker_Ein_und_Std Schalter Nr.1
	Stundenbits:	Stunden von Funkuhr; 5bit
	Wecker_Stundenbits:	Std_wecker_bits von Wecker_stellen; 5bit

	Minutenbits:	Minuten von Funkuhr; 6bit
	Wecker_ Minutenbits:	min_wecker_bits von Wecker_stellen; 6bit
	Sekundenbits:	Sekunden von Funkuhr; 6bit
	Tagbits:	Tag von Funkuhr; 5bit
	Monatbits:	Monat von Funkuhr; 4bit
	Jahrbits:	Jahr von Funkuhr; 7bit
Output:	Weckfunktion_zu_LED:	1bit
	Min_Sek_zu_LED:	1bit
	Tag_Mon_zu_LED:	1bit
	Jahr_zu_LED:	1bit
	7-Segmente:	alle 7-Segmentanzeigen
	Reset_w_seg1:	Reset auf segmente von Wecker_stellen bei unmöglichen Zahlen
	Reset_w_seg2:	Reset auf segmente von Wecker_stellen bei unmöglichen Zahlen
	Reset_w_seg3_4:	Reset auf segmente von Wecker_stellen bei unmöglichen Zahlen

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.

Schnittstelle Hauptschaltung:

Input:	Takt_blinken:	Taktfrequenz bei Weckfunktion
	Weckfunktion:	Weckfunktion_zu_LED(von 7Seg)
	Eingangstaste_Min_Sek:	Min_Sek_zu_LED(von 7Seg)
	Eingangstaste_Tag_Monat:	Tag_Mon_zu_LED(von 7Seg)
	Eingangstaste_Jahr:	Jahr_zu_LED(von 7Seg)
Output:	LED_1 bis 120:	Auf alle LED des Geckoboards

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. Diese Frequenz ist nicht ganz genau, da ich den 9bit counter

nicht bis 500, sondern seine maximalen 511, zählen lasse und die 2kHz Frequenz auch nur gerundet ist. Ich nutze diese Frequenz zum Blinken der 7-Segmente beim Wecker stellen (und das Blinken der LED bei Weckfunktion).
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.

Schnittstellen Wecker_stellen:

Input:	Takt_2kHz:	Taktfrequenz auf Flip-Flops
	Schalter_Gecko_Weckerstellen:	Wecker_Ein_und_Std Schalter Nr.2
	Taster_Gecko_1:	Taste_gecko_1
	Taster_Gecko_2:	Taste_gecko_2
	Taster_Gecko_3:	Taste_gecko_3
	Taster_Gecko_4:	Taste_gecko_4
	Reset_Seg1:	Reset_w_seg1 von 7_Seg_Anzeige
	Reset_Seg2:	Reset_w_seg2 von 7_Seg_Anzeige
	Reset_Seg3_4:	Reset_w_seg3_4 von 7_Seg_Anzeige
	Reset_Asynchron:	minuten_und_reset Schalter Nr.7
Output:	Std_wecker_bits:	nach 7_Seg_Anzeige; 5bit
	Min_wecker_bits:	nach 7_Seg_Anzeige; 6bit

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.

Funktionstest

Um die Funktionalität zu garantieren, habe ich verschiedene Tests durchgeführt. als ich den LED Teil fertiggestellt hatte, schaute ich, ob die Buchstaben auf dem Gecko richtig dargestellt werden und ob das Blinken für den Wecker funktioniert. Später habe ich das gesamte Programm, ohne Nicolas Teil getestet. Da achtete ich mich besonders darauf, ob das blinken der 7-Segmente beim Weckerstellen und allgemein das stellen des Weckers funktioniert. Ausserdem durften keine unmöglichen Uhrzeiten eingestellt werden.
Der Letzte Test war dann einfach, ob alle Funktionen mit dem Teil von Nicolas ebenfalls funktionieren. Dies war der Fall.

Erkenntnisse und Fehler

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. Dieses Programm besteht aus sechs Teilen:
DCF77 Parallelisierung: Das DCF77-Signal wird eingelesen und in ein 59bit Schieberegister, für die weitere Verwendung der Daten, eingeschoben.

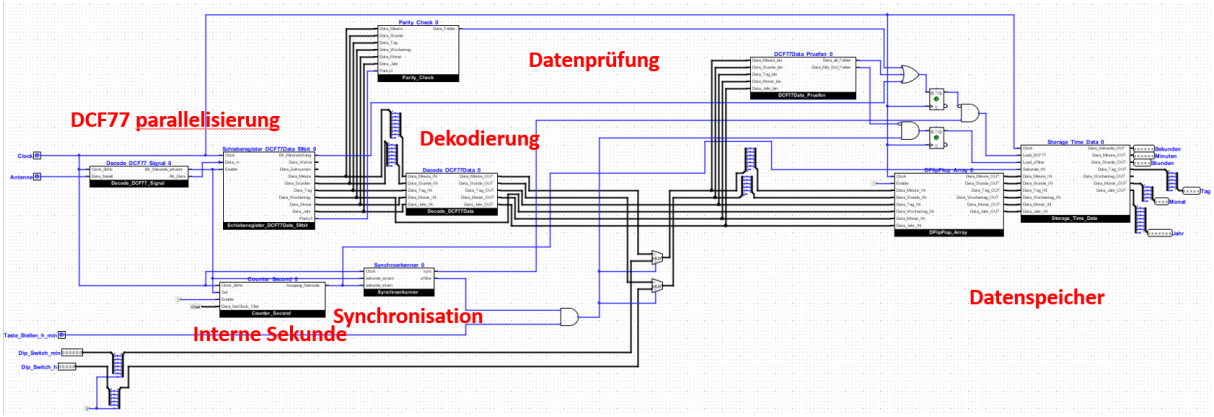
Dekodierung: Die eingelesenen Daten werden ins Binärsystem umcodiert.

Datenprüfung: Die Daten werden auf Fehler geprüft. Die Prüfung umfasst die Paritätsprüfung und die sogenannte Plausibilitätsprüfung.
Nur wenn keine Fehler gefunden werden, soll die Uhr synchronisiert werden.

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

Synchronisation: Überprüft ob es bei dem DCF77 gerade einen Minutenübergang gibt. Auch wird erkannt, ob das Gerät offline von dem 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 werden in diesem Teil die Sekunden, Minuten und die Stundendaten inkrementiert.



Schnittstelle:

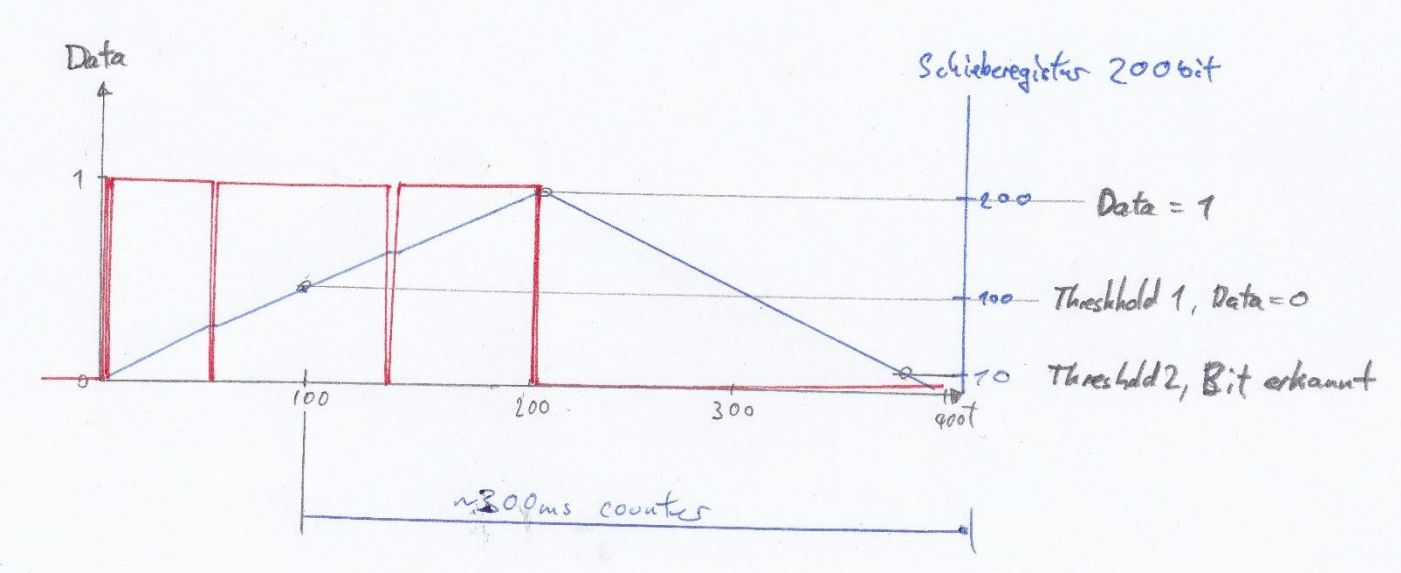
Input:	Clock:	Taktsignal
	Antenne:	DCF77-Signal-Eingang
	Taste_Stellen_h_min:	Taste zum manuellen Stellen von Zeitdaten (nur Minute und Stunde)
	Dip_Switch_min:	Minutendaten-Eingang zu den manuellen Stellen; 6bit
	Dip_Switch_h:	Stundendaten-Eingang zu den manuellen Stellen; 5bit
Output:	Sekunden:	Sekundendaten; 6bit
	Minuten:	Minutendaten; 6bit
	Stunden:	Stundendaten; 5bit
	Tag:	Kalendertagdaten; 5bit
	Monat:	Monatsdaten; 4bit
	Jahr:	Jahrdaten (zweistellig); 7bit

1. Teil: DCF77 Parallelisierung

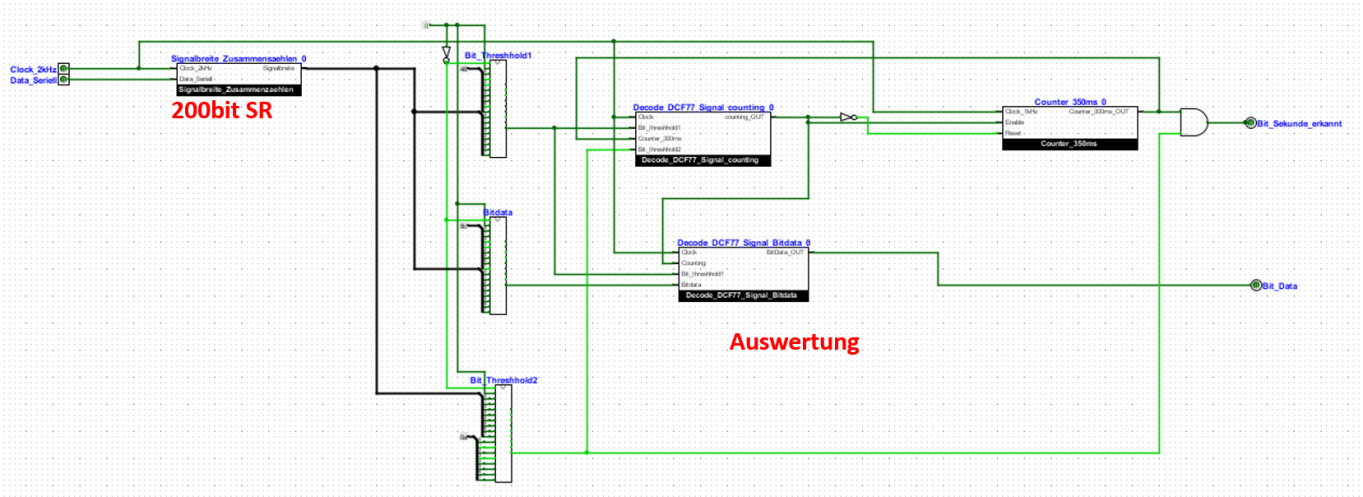
Decode_DCF77_Signal:

Die Erkennung des DCF77-Signals wurde mit einem 200bit Schieberegister realisiert, welches nach jeder Millisekunde den momentanen Wert der Antennenleitung reinschiebt. Die Werte in dem Schieberegister werden Bitweise

zusammengezählt und die Summe benutzt um Aussagen über DCF77-Daten zu machen:



Der Vorteil bei dieser Methode ist, dass sie robuster gegenüber schlechten Signalen ist.
Wird die erste Threshhold „Threshhold1" überschritten, so heisst es, dass es sich möglicherweise um einen Signalzustand „0" handelt und der Daten-Ausgang wird auf Null geschaltet. Gleichzeitig wird ein Counter gestartet, der etwa auf 300ms Zählt.
Wird in dem Schieberegister einen Wert von etwa 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 dem Zeitpunkt wo der Counter fertiggezählt hat, der Wert für Threshhold2 unterschritten wurde. Wenn das nicht der Fall ist, wird davon ausgegangen, dass es sich um fehlerhaftes Signal handelt.



Da in der Realität die Signalbreite immer von den idealen Breite abweicht, wurden die Werte 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 DCF77-Daten erkannt wurden.
	Bit_Data:	Den Wert der DCF77-Daten.

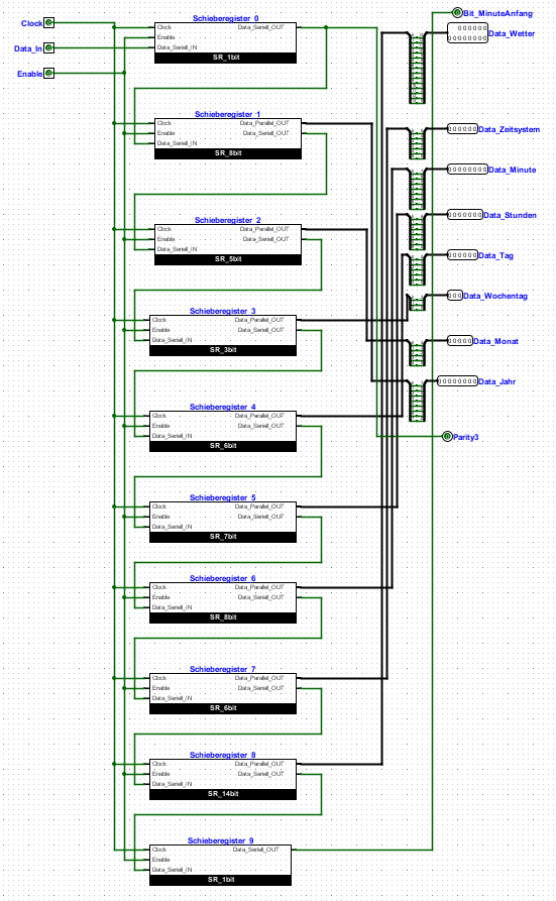
Signalbreite_Zusammenzaehlen:
Besteht aus einem 200bit Schieberegister und Addiernetzwerk, das die einzelnen Bitwerte zusammenzählt.
Jede 1ms wird ein Wert geladen.

Decode_DCF77_Signal_counting: Hier wird entschieden, wann der Counter loszählen soll.
Nach einer Beendung des Counters muss gewartet werden, bis die Unterschreitung der Threshhold2 erfolgt ist.

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 zum nächsten Anlauf.

Counter_350ms: Zählt mittels 9bit-Counter bis 350ms und gibt dann einen „1" für einen Takt aus.

Schieberegister_DCF77Data_59bit:
Die von Decode_DCF77_Signal ausgegebenen seriellen Daten werden in 59bit Schieberegister hineingeschoben und aufgeteilt.
Die Daten sind nun parallel verfügbar und können ausgewertet werden.



Schnittstelle:
Input:

Clock:	Systemtakteingang
Data_In:	serielle Dateieneingang
Enable:	Aktivierung von SR

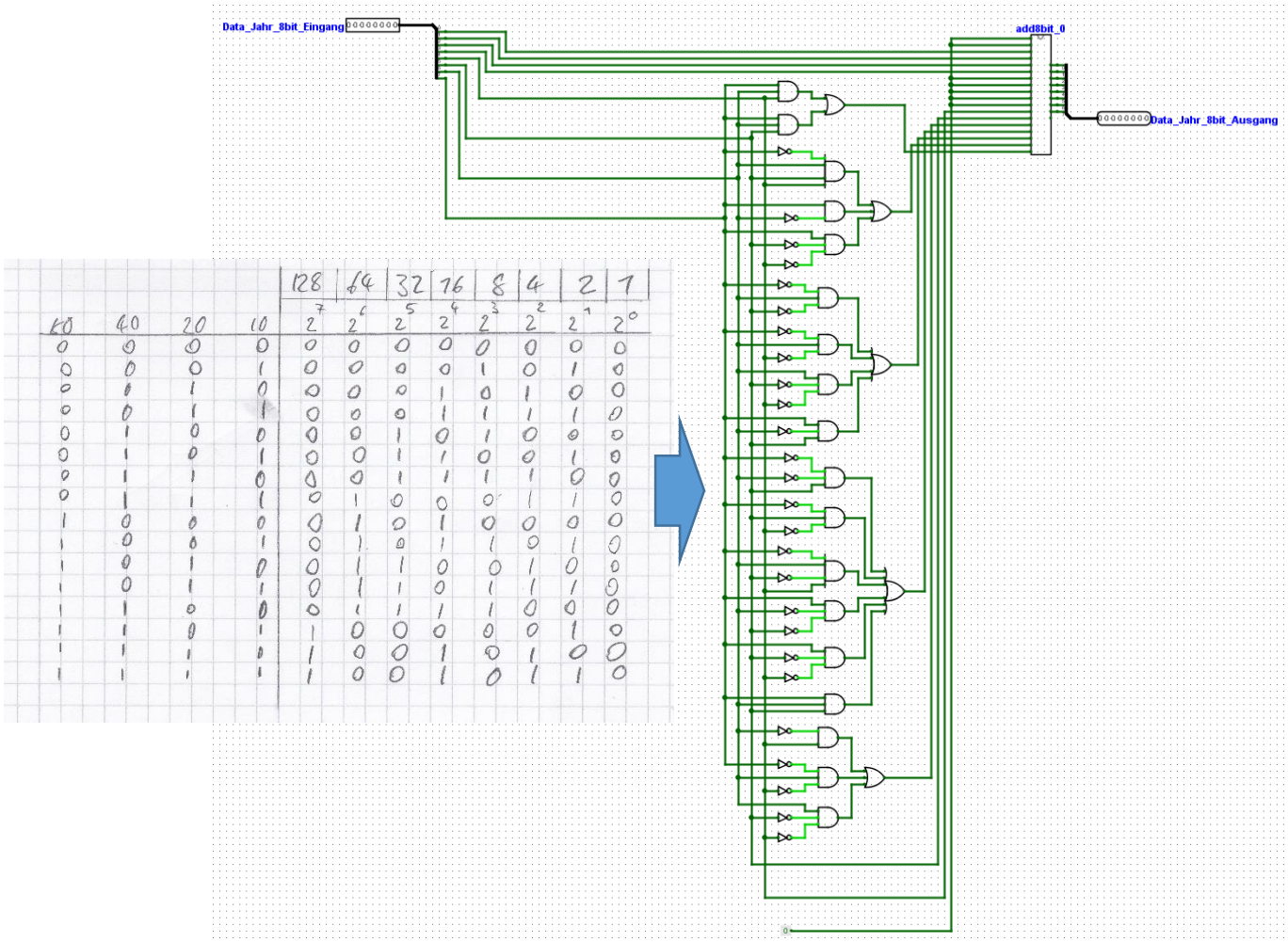
Output:

Bit_MinuteAnfang:	erstes Bit des Datenpackets
Data_Wetter:	Wetterdaten; 14bit
Data_Zeitsystem:	zusätzliche Zeitdaten; 6bit
Data_Minute:	Minutendaten mit Parity; 8bit
Data_Stunden:	Stundendaten mit Parity; 7bit
Data_Tag:	Kalendertagdaten; 6bit
Data_Wochentag:	Wochentagdaten; 3bit
Data_Monat:	Monatsdaten; 5bit
Data_Jahr:	Jahresdaten; 8bit
Parity3:	Paritätsbit von Data_Tag, Data_Wochentag, Data_Monat und Data_Jahr

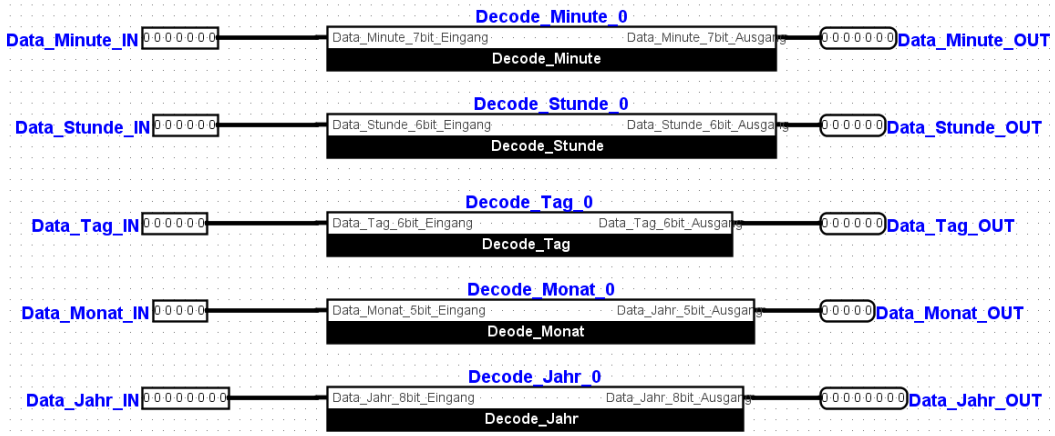
2. Teil: Dekodierung

Decode_DCF77Data:
Die empfangenen Daten sind in BCD-Format kodiert.
Für die Dekodierung wurde der höherwertige BCD-Wert (Zehnerwertig) in Binär umgewandelt und mit den unveränderten niedrigeren BCD-Werten addiert.

Beispiel für Decode_Jahr:



Decode_DCF77Data Schaltung:



Interface:

Input: Data_Minute_IN: Minutendaten ohne Paritätsbit; 7bit 15

	Data_Stunde_IN:	Stundendaten ohne Paritätsbit; 6bit
	Data_Tag_IN:	Kalendertagdaten; 6bit
	Data_Monat_IN:	Monatsdaten; 5bit
	Data_Jahr_IN:	Jahresdaten; 8bit
Output:	Data_Minute_OUT:	dekodierte Minutendaten; 7bit
	Data_Stunde_OUT:	dekodierte Stundendaten; 6bit
	Data_Tag_OUT:	dekodierte Kalendertagdaten; 6bit
	Data_Monat_OUT:	dekodierte Monatsdaten; 5bit
	Data_Jahr_OUT:	dekodierte Jahresdaten; 8bit

Decode_Minute: Dekodierer für die Minutendaten.

Decode_Stunde: Dekodierer für die Stundendaten.

Decode_Tag: Dekodierer für die Stundendaten.

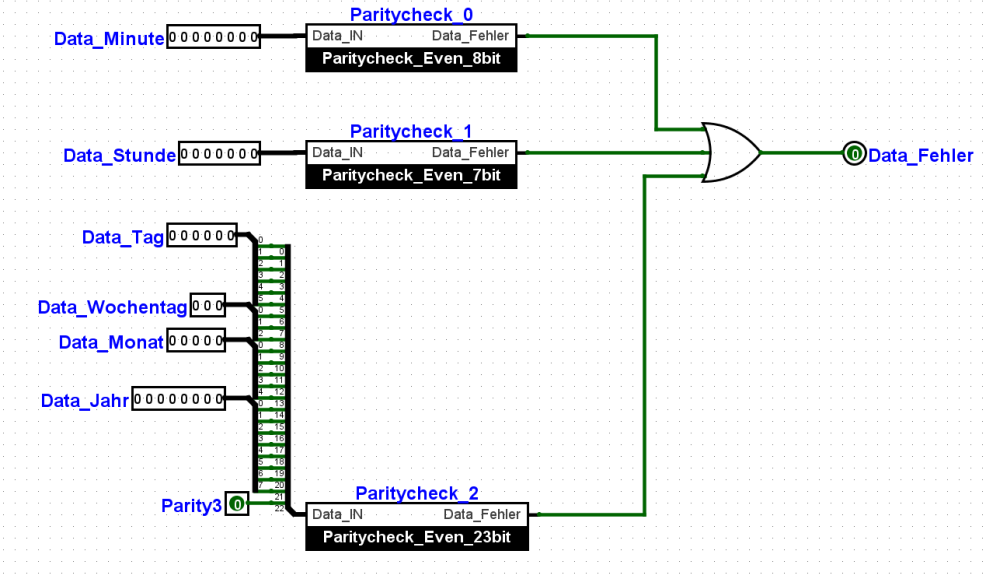
Decode_Monat: Dekodierer für die Monatsdaten.

Decode_Jahr: Dekodierer für die Jahresdaten.

3. Teil: Datenprüfung

Parity_Check:

Kontrolliert die Daten auf Richtigkeit mittels Paritätsbit.
Bei den Minuten und Stundendaten hat jeder ein separates Paritätsbit, jedoch bei den Tag, Wochentag, Monats und Jahresdaten ein gemeinsames.
Bei den DCF77 handelt es sich um Even-Parity. Das heisst, jedes Bit der Daten mit dem Paritybit zusammengezählt, addiert sich auf eine gerade Zahl.
Die Kontrolle wurde mit einer Verkettung von XOR-Gattern realisiert.
Wird ein Fehler erkannt, so wird die Synchronisierung blockiert.



Interface:

Input:	Data_Minute:	Minutendaten mit Parity; 8bit
	Data_Stunde:	Stundentaten mit Parity; 7bit
	Data_Tag:	Kalendertagdaten; 6bit
	Data_Wochentag:	Wochentagdaten; 3bit
	Data_Monat:	Monatsdaten; 5bit
	Data_Jahr:	Jahresdaten; 8bit
	Parity3:	Paritätsbit für Data_Tag, Data_Wochentag, Data_Monat und Data_Jahr
Output:	Data_Fehler:	Wird auf "1" gesetzt wenn die Parität nicht übereinstimmt.

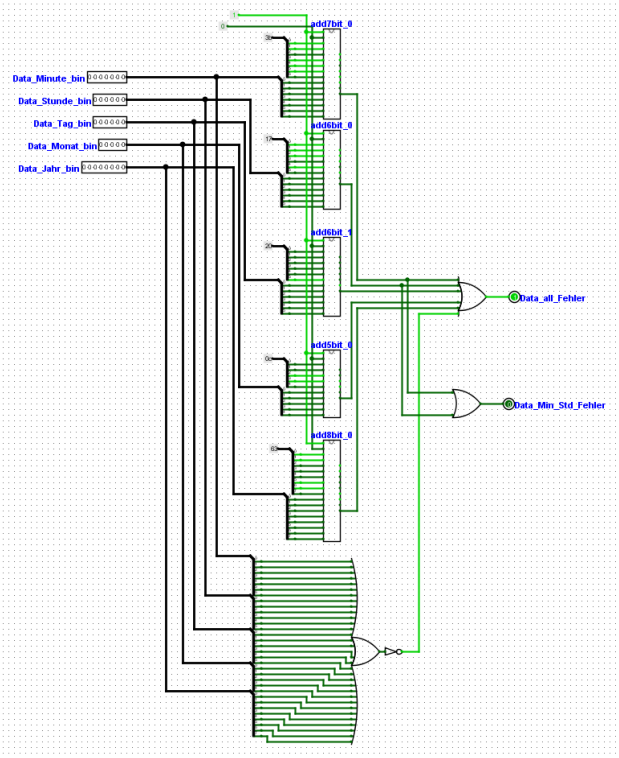
Paritycheck_Even_8bit: Paritycheckblock für 8bit mit Even-Parität

Paritycheck_Even_7bit: Paritycheckblock für 7bit mit Even-Parität

Paritycheck_Even_23bit: Paritycheckblock für 23bit mit Even-Parität

DCF77Data_Pruefen:
Da Paritätskontrolle für eine Fehlererkennung nicht ausreichend ist, werden die empfangenen Daten auf Plausibilität geprüft.
Beispielsweise kann es nicht sein, dass die Minutendaten einen Wert von 78 Minuten enthält.
Da bei einem schlechten Signal sein kann, dass nur Nullen empfangen werden und dies durch Paritätskontrolle nicht erkannt werden konnte, wird hier auch auf den kompletten Null-Empfang geprüft.

Wird ein Fehler erkannt, so wird die Synchronisation blockiert.



Interface:

Input:	Data_Minute_bin:	Minutendaten ohne Paritätsbit; 7bit
	Data_Stunde_bin:	Stundendaten ohne Paritätsbit; 6bit
	Data_Tag_bin:	Kalendertagdaten; 6bit
	Data_Monat_bin:	Monatsdaten; 5bit
	Data_Jahr_bin:	Jahresdaten; 8bit

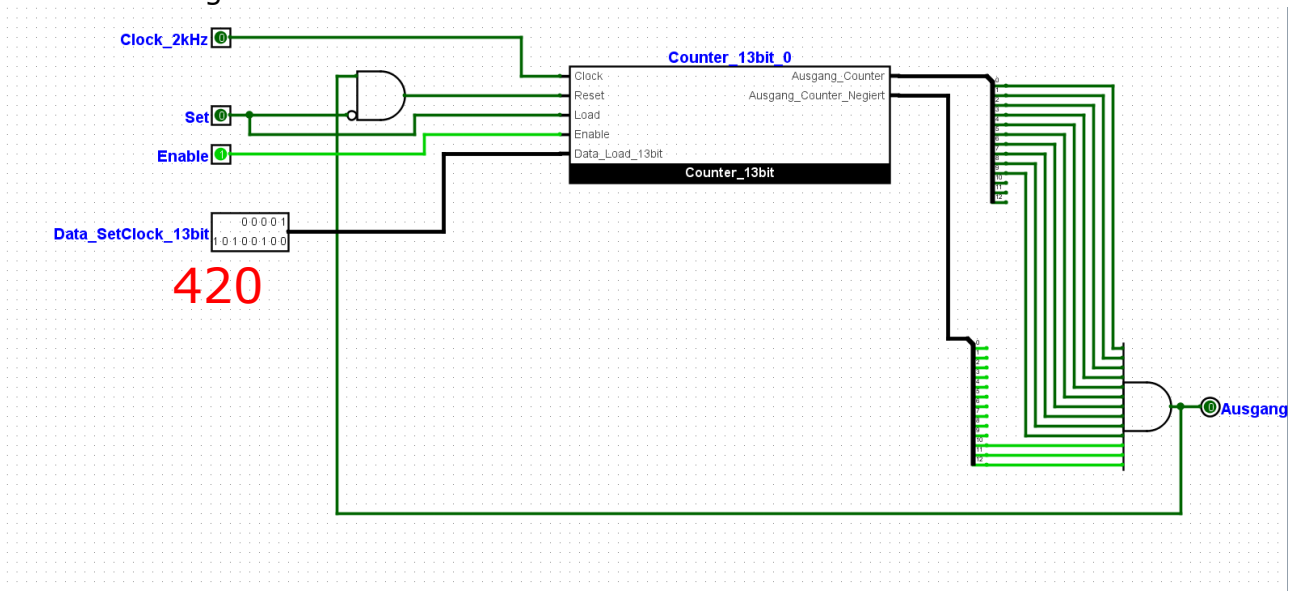
Output:	Data_all_Fehler:	Plausibilitätskontrolle-Bit für DCF77-Daten. Wird auf „1“ geschaltet, wenn die Daten fehlerbehaftet sind.
	Data_Min_Std_Fehler:	Plausibilitätskontrolle-Bit für Offline-Uhrstellung. Es wird nur die Minuten und die Stunden Daten geprüft.

Die Input-Daten in diesem Block müssen alle zuerst dekodiert werden.

4. Teil: Interne Sekunde

Counter_Second:

Aus dem System-Ereignis-Takt von 2kHz d.h. einen Realtakt von 1kHz wird nun einen Sekundentakt erstellt.
Den Wert 1kHz ist jedoch ein gerundeter Wert und soll in Wirklichkeit 1024Hz sein.
Da wir allerdings nach den Test herausgefunden haben, dass der Takt näher zu 1023Hz ist, zählt dieser Block bis 1023.
Dieser Block enthält einen 13bit-Counter, da wir am Anfang mit einen Realtakt von 2kHz ausgegangen sind.
Für die Sekundensynchronisierung ist es möglich den momentanen Zählerstand zu modifizieren.
Für die exakte Sekundensynchronisierung wird hier bei einer Synchronisierung der Wert 420 geladen.



Interface:

Input:	Clock_2kHz:	Systemtakteingang
	Set:	Lädt den Wert für die Zählerstandmodifizierung.
	Enable:	Aktiviert den Zähler.
	Data_SetClock_13bit:	Den Wert, die für den

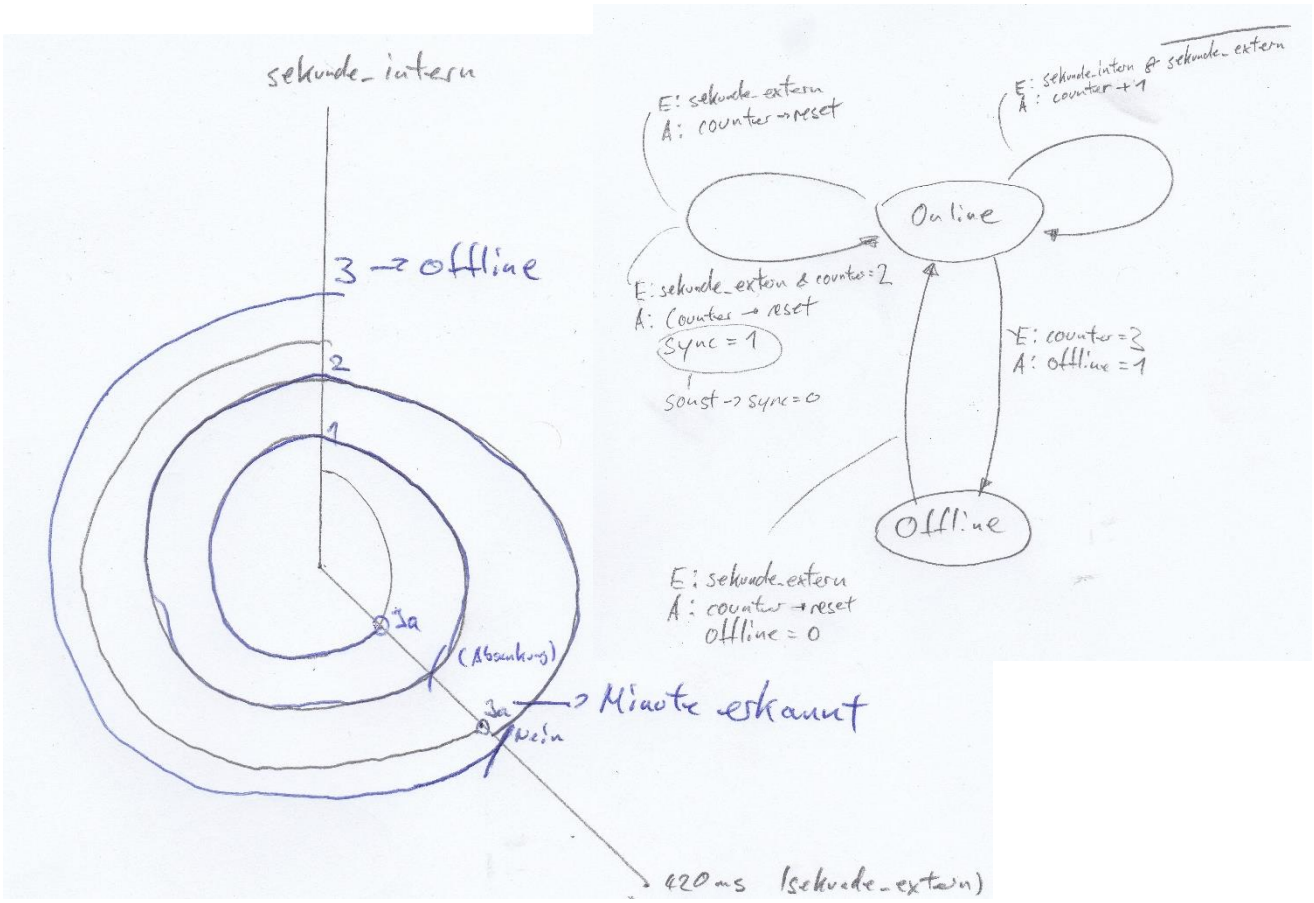
		Zählerstandmodifizierung geladen wird; 13bit
Output:	Ausgang:	Wird bei einer Sekune für einen Takt auf „1“ gestellt.

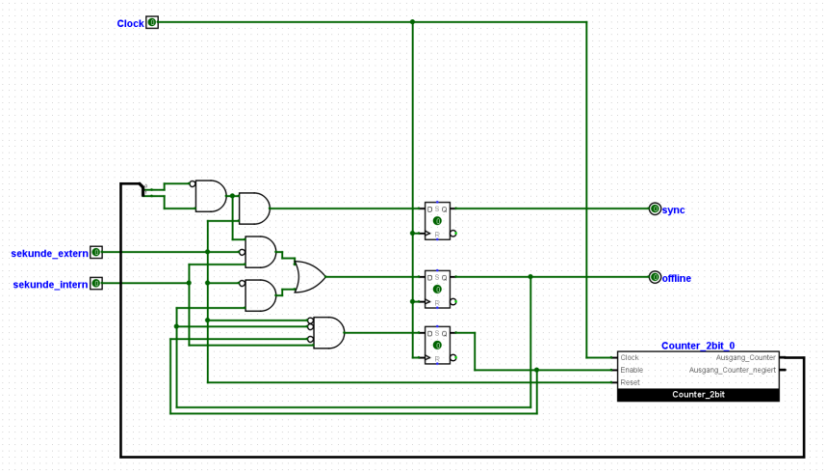
Counter_13bit: 13bit Zähler:

5. Teil: Synchronisation

Synchroerkenner:

Am Ende der Minute wird für eine Sekunde das Signal komplett heruntergesenkt. Um die Uhr erfolgreich synchronisieren zu können muss man dies erkennen. Wird jedoch über eine Periode von einer Sekunde die externe Sekunde nicht erkannt, so gibt man bekannt, dass der Funkuhr offline ist.





Interface:

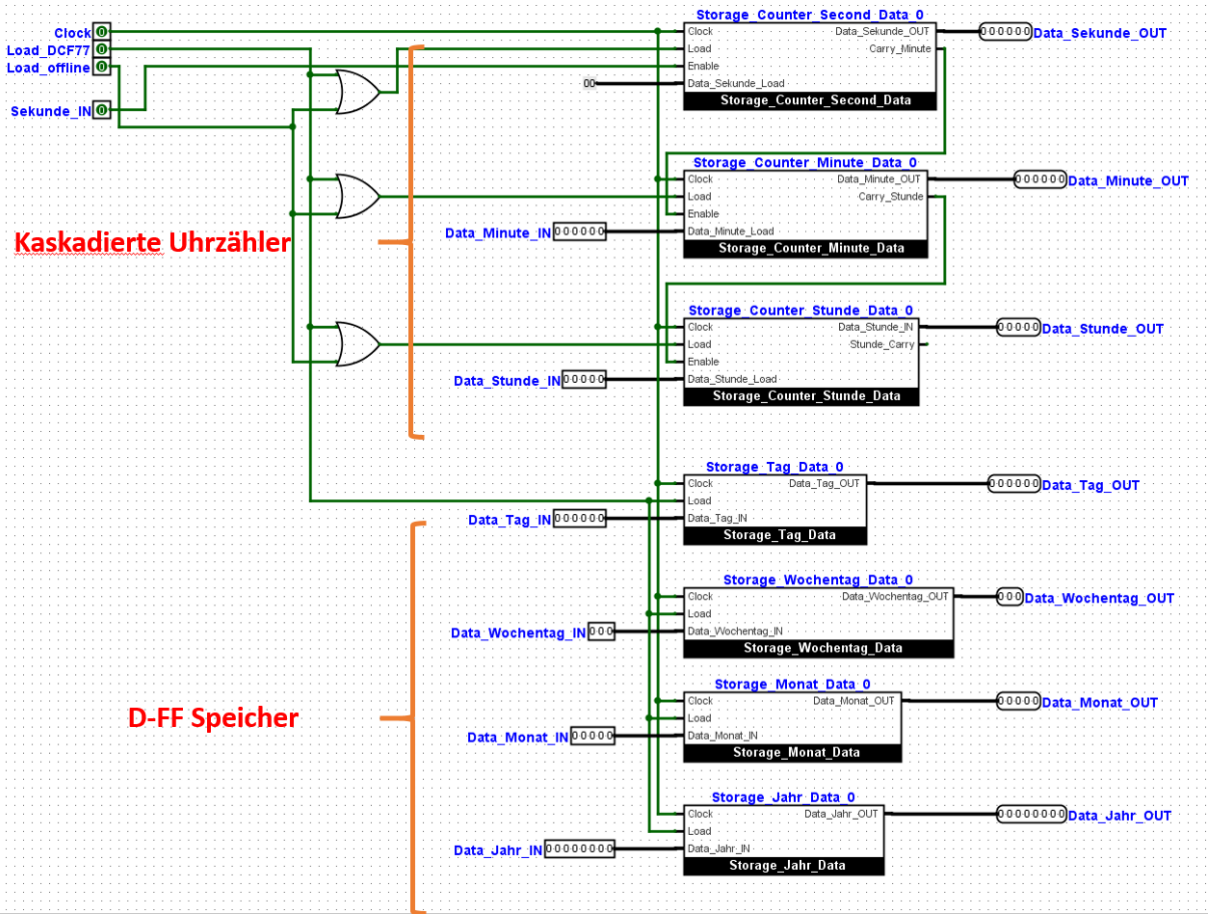
Input:	sekunde_extern:	Eingang für externe Sekunde
	sekunde_intern:	Eingang für interne Sekunde
Output:	sync:	Wird auf „1" gesetzt wenn die Ende der Minute erkannt wird.
	Offline:	Wird auf „1" gesetzt wenn das Gerät offline ist.

Counter_2bit: Zwei-Bit Counter.

6. Teil: Datenspeicher

Storage_Time_Data

Dieser Block ist der Datenspeicher wo die Zeitdaten gespeichert werden. Gleichzeitig findet in diesem Block die Inkrementierung der Sekunden, Minuten und Stunden statt.
Es muss beachtet werden, dass der Carry des Zählers nur für einen Takt auf „1" geschaltet ist, da sonst die Werte mit Systemtakt inkrementiert werden.
Die Daten für den Tag, Wochentag, Monat und Jahr werden nicht inkrementiert und hängen von DCF77-Daten ab.
Bei jeder Synchronisation wird der Sekundenzähler auf Null gesetzt.
Um zu verhindern, dass bei einer manuellen Ladung der Minuten und Stundenzähler nicht gültige Daten für Tag, Wochentag, Monat und Jahr geladen werden, wurde ein zusätzlicher Eingang für Offline-Ladung hinzugefügt.



Interface:

Input:	Clock:	Systemtakteingang
	Load_DCF77:	Die DCF77-Daten werden in Speicher geladen.
	Load_offline:	Nur Minuten und Stundendaten werden geladen.
	Sekunde_IN:	Eingang für das Sekundensignal
	Data_Minute_IN:	Dekodierte Minutendaten; 6bit
	Data_Stunde_IN:	Dekodierte Stundendaten; 5bit
	Data_Tag_IN:	Dekodierte Kalendertagdaten; 6bit
	Data_Wochentag_IN:	Dekodierte Wochentagdaten; 3bit
	Data_Monat_IN:	Dekodierte Monatsdaten; 5bit
	Data_Jahr_IN:	Dekodierte Jahresdaten; 8bit
Output:	Data_Sekunde_OUT:	Ausgang Sekundendaten; 6bit
	Data_Minute_OUT:	Ausgang Minutendaten; 6bit
	Data_Stunde_OUT:	Ausgang Stundendaten; 5bit
	Data_Tag_OUT:	Ausgang Kalendertagdaten; 6bit
	Data_Wochentag_OUT:	Ausgang Wochentagdaten; 3bit
	Data_Monat_OUT:	Ausgang Monatsdaten; 5bit
	Data_Jahr_OUT:	Ausgang Jahresdaten; 8bit

Storage_Counter_Second_Data: Speicherblock für Sekundendaten. Gleichzeitig auch Sekundenzähler.

Storage_Counter_Minute_Data: Speicherblock für Minutendaten. Gleichzeitig auch Minutenzähler.

Storage_Counter_Stunde_Data: Speicherblock für Stundendaten. Gleichzeitig auch Stundenzähler.

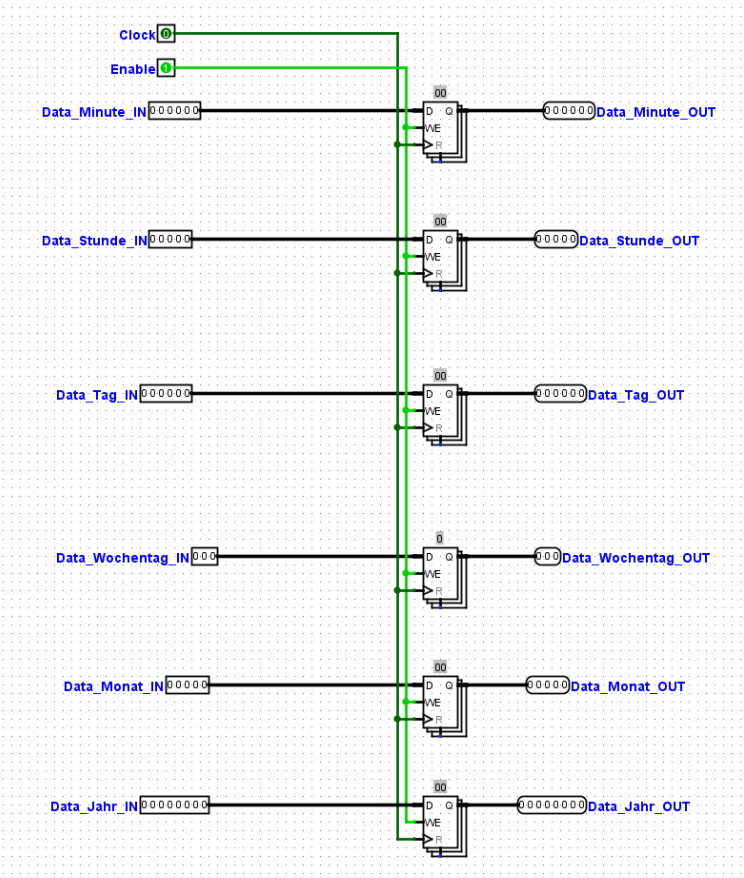
Storage_Tag_Data: Speicherblock für Kalendertagdaten.

Storage_Wochentag_Data: Speicherblock für Wochentagdaten.

Storage_Monat_Data: Speicherblock für Monatsdaten.

Storage_Jahr_Data: Speicherblock für Jahresdaten.

DFlipFlop_Array
Da der 5. Teil meines Systems D-FlipFlops verwendet, müssen alle mit Synchronisation relatierten Leitungen auch mit D-FlipFlops versehen werden, da sonst das Signal für die Synchronisation zu spät ankommt.
In diesem Block werden die Zeitdaten für die Kompatibilität mit Synchronisation in einem vorstufigen D-FF Speicher gespeichert
Die Leitungen für Fehlerprüfung werden extern separat mit D-FF versehen.



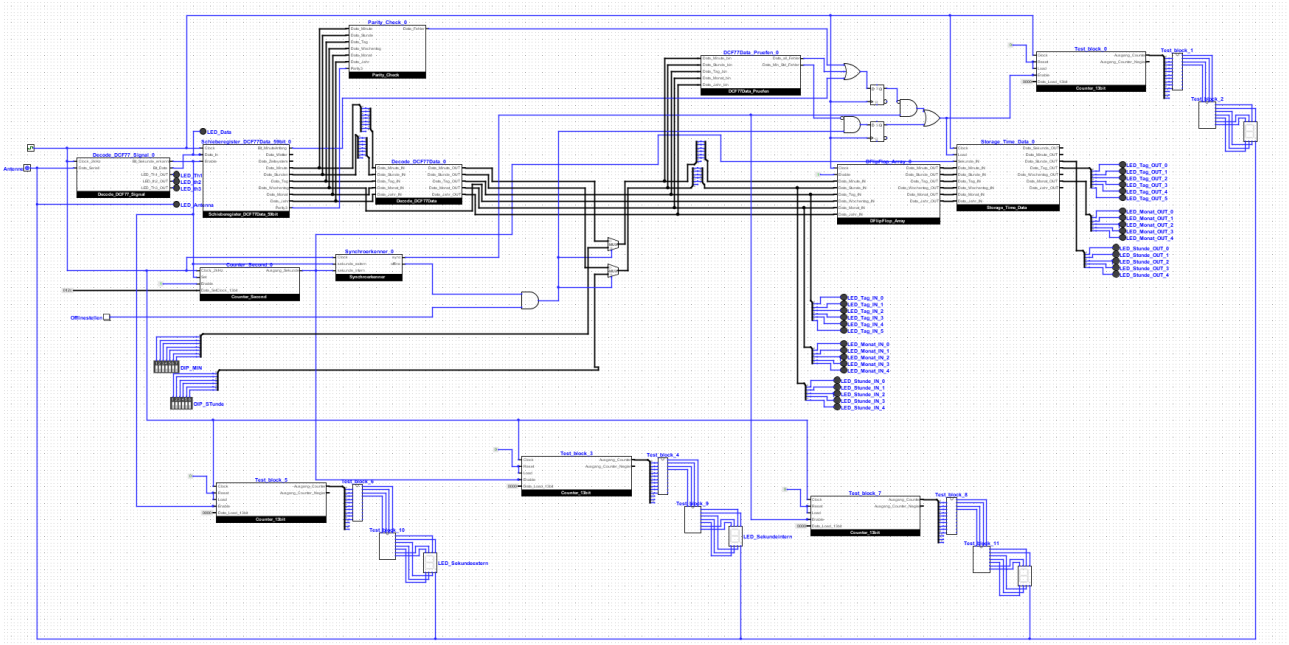
Interface:

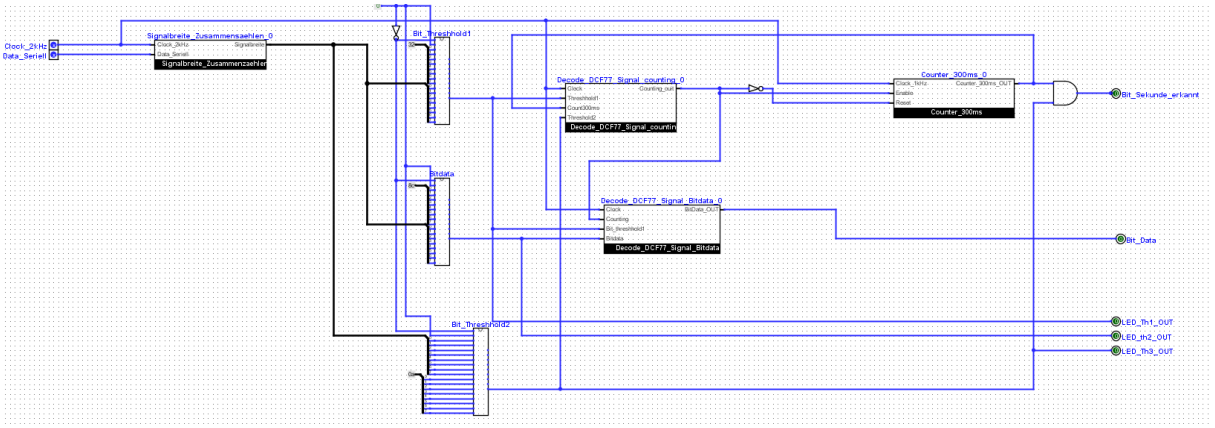
Input:	Clock:	Systemtakteingang
	Enable:	Aktiviert die D-FlipFlops
	Data_Minute_IN:	Dekodierte Minutendaten; 6bit
	Data_Stunde_IN:	Dekodierte Stundendaten; 5bit
	Data_Tag_IN:	Dekodierte Kalendertagdaten; 6bit
	Data_Wochentag_IN:	Dekodierte Wochentagdaten; 3bit
	Data_Monat_IN:	Dekodierte Monatsdaten; 5bit
	Data_Jahr_IN:	Dekodierte Jahresdaten; 8bit
Output:	Data_Sekunde_OUT:	Ausgang Sekundendaten; 6bit
	Data_Minute_OUT:	Ausgang Minutendaten; 6bit
	Data_Stunde_OUT:	Ausgang Stundendaten; 5bit
	Data_Tag_OUT:	Ausgang Kalendertagdaten; 6bit
	Data_Wochentag_OUT:	Ausgang Wochentagdaten; 3bit
	Data_Monat_OUT:	Ausgang Monatsdaten; 5bit
	Data_Jahr_OUT:	Ausgang Jahresdaten; 8bit

Funktiontests

Funkuhr:

Da es etwas schwierig ist, die Funktionen der DCF77 relatierenden Blöcke zu simulieren wurde die Prüfung mittels zusätzlichen LEDs und 7-Segmentanzeige durchgeführt:





Mit den LED’s werden den Antennensignal, dessen Verarbeitung, Daten vor Synchronisierung und Daten nach Synchronisierung ersichtlich.
Mit 7-Segmentanzeigen werden die Änderungen an Leitungen: Externer Sekundentakt, endgültiger Sekundentakt, Synchronisierungsbefehl und endgültige Synchronisation ersichtlich.

Ergebnis: Bei guten Empfang funktioniert alles wie vorgesehen.
Bei einem grösseren Einfluss von Störsignalen wird der Sekundentakt merklich gestört und ab und zu kommt es auch vor, dass fehlerhafte Daten geladen werden.

Verbleibende Probleme

Sekundentakt:
Unter Einfluss von grösseren Störsignalen wird der Sekundentakt merklich gestört.
Die Abstände zwischen den einkommenden Signalen müssen überprüft werden.
Eine leichtere Lösung wäre die interne Sekunde nur dann zu synchronisieren, wenn das Ende der Minute erkannt wurde anstatt nach jeder Erkennung der externen Sekunde.

Fehlerhafte Daten :
Bei gestörten Signalen erkennen ab und zu die Datenprüfungsblöcke die fehlerhaften Daten nicht.
Ein besseres System für das Prüfen von Daten wäre, die zu synchronisierenden Daten mit den vorherigen zu vergleichen.

Fazit

Während der Arbeit am Projekt lernten wir sehr viel. Wir lernten das Programm Logisim kennen, wie wir gewisse Grundschaltungen aufbauen müssen und konnten uns 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.

Die Arbeit mit Logisim hat grundsätzlich gut funktioniert. Das Programm weist zwar einige Fehler auf, aber zur Ausführung dieses Projektes war es ausreichend. Es ist optisch übersichtlich und gut strukturiert. Für repetitive Arbeiten ist es allerdings etwas mühsam. Auch die Simulation welche nicht immer funktioniert und angezeigte Fehler, welche sich meist durch einen Neustart beheben lassen sind etwas schade.

Die Zusammenarbeit in unserem Team hat gut funktioniert, wurde möglichst fair aufgeteilt und die Arbeit konnte daher souverän ausgeführt werden. wir können uns vorstellen auch in Zukunft Projekte zusammen zu realisieren.