

# VHDL Praktikum

## Anwendung Elektronischer Systeme

### Yet Another Signal Generator

Durchführung:	A. Schmocker, T. Lang
Abgabedatum:	19.06.2016
Modul:	BTE5024 Anwendung Elektronischer Systeme
Betreuer:	Thorsten Mähne
Projektseite:	<a href="https://github.com/id101010/vhdl-yasg">https://github.com/id101010/vhdl-yasg</a>

# Inhalt

- [1 Einführung](#)
  - [1.1 Projektplan](#)
- [2 Komponenten und Blockschema](#)
  - [2.1 Aufbau des Gesamtsystems](#)
  - [2.2 Controller](#)
    - [Übersicht](#)
    - [Implementierung](#)
    - [Test](#)
  - [2.3 DDS](#)
    - [Übersicht](#)
    - [Implementierung](#)
    - [Test](#)
  - [2.4 LCD Driver](#)
    - [Übersicht](#)
    - [Implementierung](#)
    - [Testbench](#)
  - [2.5 Rotary Decoder](#)
    - [Übersicht](#)
    - [Implementierung](#)
    - [Test](#)
  - [2.6 SPI Driver](#)
    - [Übersicht](#)
    - [Implementierung](#)
    - [Test](#)
- [3 Statistiken](#)
- [4 Fazit](#)

## 1 Einführung

Im Rahmen des Moduls BTE5024 “Anwendungen Elektronischer Systeme” soll ein Funktionsgenerator entwickelt werden, der mittels Direct Digital Synthesis (DDS) ein Signal einer wählbaren Frequenz und Form (Sinus, Rechteck, Dreieck, Sägezahn) erzeugt und über einen 12-Bit Digital-Analog-Umwandler (DAC) ausgibt. Die Frequenz des Ausgangssignals soll über einen Drehknopf und Taster einstellbar sein und auf einem LCD angezeigt werden. Das digitale System soll in synthetisierbaren VHDL beschrieben und auf dem Xilinx Spartan-3AN Evaluationboard implementiert sowie getestet werden.

## 1.1 Projektplan

Wir haben uns beim Projekt folgende Meilensteine gesetzt:

<b>Datum</b>	<b>Meilenstein</b>
09.05.2016	Grobplan, Konzept fertig
16.05.2015	Modularisierung, Schnittstellendefinition abgeschlossen
30.05.2016	Entwicklung der einzelnen Module abgeschlossen
03.06.2016	Erster Test auf Zielhardware durchgeführt
15.06.2016	Alle Module fertiggestellt, inkl Testbenches
19.06.2016	Alle Fehler behoben und Dokumentation fertiggestellt

## 2 Komponenten und Blockschema

### 2.1 Aufbau des Gesamtsystems

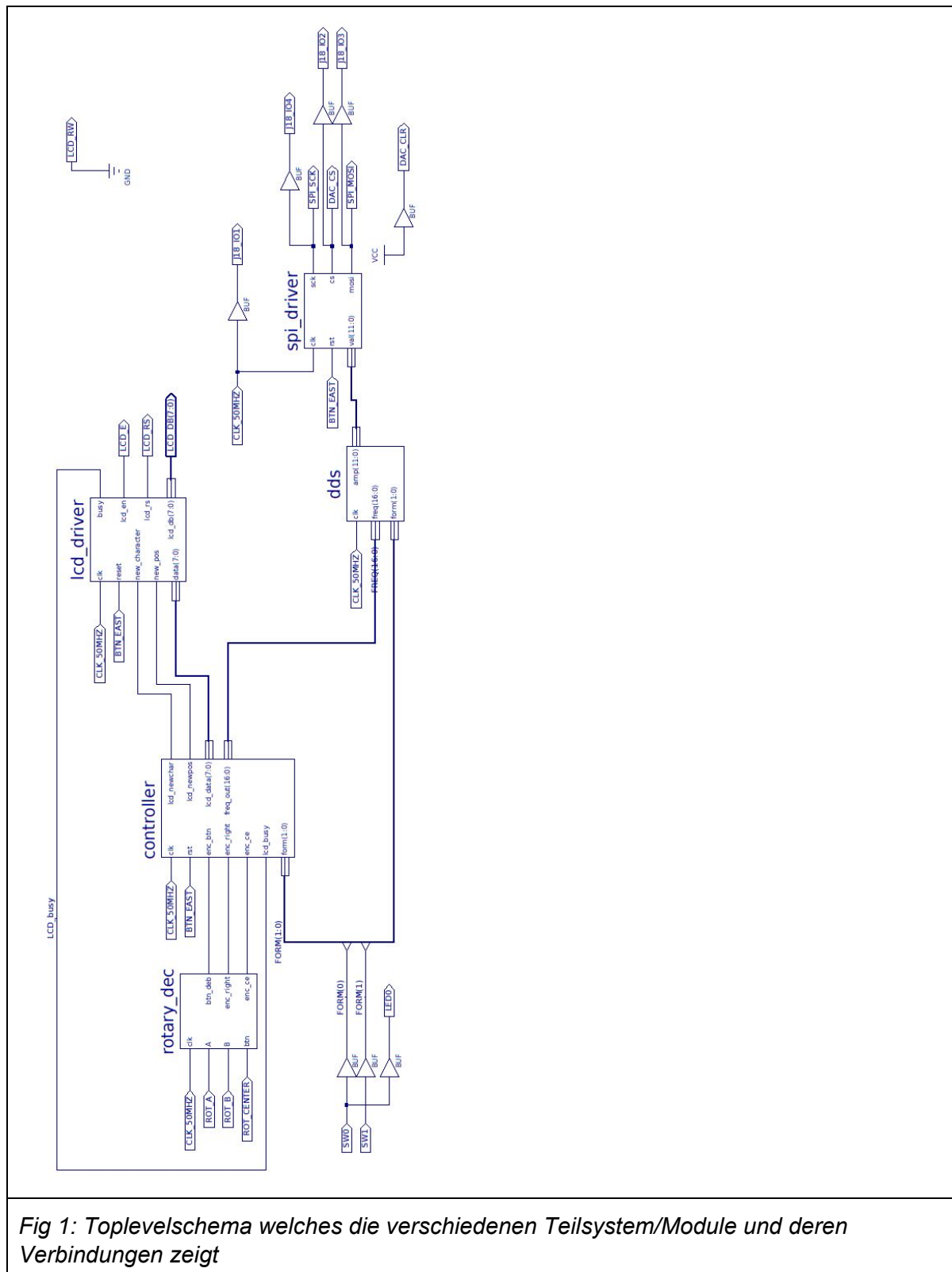


Fig 1: Toplevelschema welches die verschiedenen Teilsystem/Module und deren Verbindungen zeigt

Das Gesamtsystem besteht aus einzelnen Teilsystemen, welche alle mit dem Systemclock von 50MHz getaktet sind. Die Benutzerführung ist mittels LCD realisiert. Die Signalausgabe basiert auf dem 12bit Digital-to-Analog-Converter, des Spartan Boards welcher mittels SPI angesprochen wird. Die Bedienung basiert auf dem Drehgeber, zwei DIP-Switches und einem Taster (Reset), welche alle auf dem Spartan 3AN Board installiert sind.

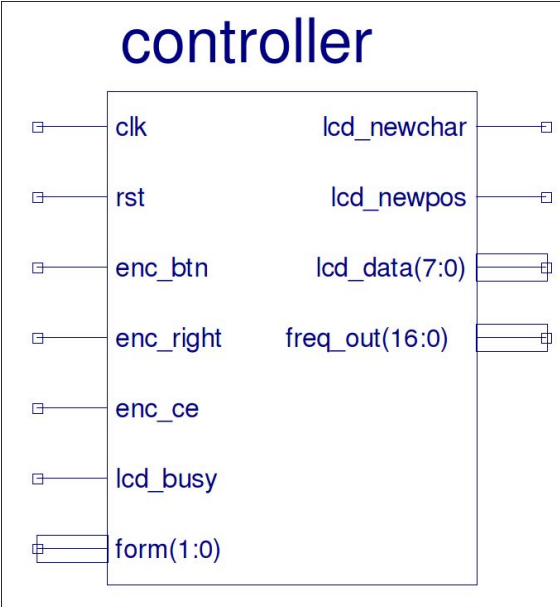
Die Signale des Drehgebers werden vom Modul *rotary\_dec* ausgelesen und in ein entsprechendes, entprelltes Signal konvertiert welches dann dem *Controller*-Modul übergeben wird. Der Drehgeber wird zum Einstellen der Frequenz verwendet. Die DIP-Switches verwenden wir zur Auswahl der Signalform, gemäss untenstehender Tabelle.

DIP2	DIP1	Modus
0	0	Signalform Rechteck
0	1	Signalform Sägezahn
1	0	Signalform Dreieck
1	1	Signalform Sinus

Das *Controller*-Modul ist zuständig für die gesamte Benutzerführung und Berechnung der gewünschten Frequenz. Das *DDS*-Modul setzt die gewünschte Frequenz in eine Amplitude um, bevor diese dann vom *spi\_driver* ausgegeben wird.

## 2.2 Controller

### Übersicht

 <p>The diagram shows a block labeled 'controller'. On the left side, there are input ports: 'clk', 'rst', 'enc_btn', 'enc_right', 'enc_ce', 'lcd_busy', and 'form(1:0)'. On the right side, there are output ports: 'lcd_newchar', 'lcd_newpos', 'lcd_data(7:0)', and 'freq_out(16:0)'.</p>	<p>Das Controller Modul ist für die Benutzerinteraktion (Ein- und Ausgabe) zuständig und vernetzt alle anderen Module.</p> <p>Die Ein- und Ausgänge des Moduls werden mit Treiber-Modulen (Encoder, LCD, SPI) und dem DDS Modul verbunden. Das Modul sendet z.B. die darzustellenden Zeichen an den LCD-Treiber, welcher diese dann zur Darstellung bringt.</p>
<p>Fig 2: Schemasympol des Controllers</p>	<p><u>Ein und Ausgänge:</u></p> <p><b>clk:</b> Clock Eingang</p> <p><b>rst:</b> High-aktiver, asynchroner Reset</p> <p><b>enc_btn:</b> Eingang für entprellten Push-Button zur Wahl der Dezimalstelle</p> <p><b>enc_right:</b> Eingang des Decoders: 1=rechtsdrehung, 0 = linksdrehung</p> <p><b>enc_ce:</b> Eingang des Decoders: Clock-Enable für das obere Signal</p> <p><b>lcd_busy:</b> Lcd-Feedback: 1 = Lcd/Treiber ist zurzeit beschäftigt und darf nicht verwendet werden</p> <p><b>form:</b> Eingang zur Auswahl der Signalform. Beschreibung siehe Kap 2.1</p> <p><b>lcd_newchar:</b> LCD Ausgang: 1 wenn ein neues Zeichen an <i>lcd_data</i> bereitliegt.</p> <p><b>lcd_newpos:</b> LCD Ausgang: 1 wenn eine neue Position/Adresse an <i>lcd_data</i> bereitliegt.</p> <p><b>lcd_data:</b> Entweder ein neues Zeichen oder eine neue Zeichen-Position</p> <p><b>freq_out:</b> Gewünschte Frequenz in Hz</p> <p><u>Generische Parameter:</u></p> <p><b>freq_res:</b> Breite des Frequenz ausgangs in Bits</p>

## Implementierung

Das Modul ist als State Machine implementiert. Dabei gibt es eine Besonderheit (ist auch beim Display Treiber so ähnlich vorhanden): Die State-Machine besitzt einen State *s\_wait* aus dem sie in einen "vom Aufrufer bestimmten" State zurückkehrt. In diesem Fall wird im *s\_wait* State auf das *lcd\_busy* Signal gewartet, und anschliessend wird der nächste State auf den Wert des Register *ret\_state* (für return) gesetzt.

Einige States werden nur einmalig aufgerufen, um statische Strings aufs Display zu schreiben. Die States *s\_form\_cont* bzw *s\_freq\_cont* werden bei jeder Änderung der Signalform bzw Signalfrequenz aufgerufen.

Da das Display kein deaktivieren des Auto-Inkrementes ermöglicht, wird nach dem Senden eines neuen Zeichens normalerweise noch der Cursor zurückgesetzt.

Im folgenden finden sich einige Tests welche die Funktionalität des Moduls veranschaulichen.

## Test

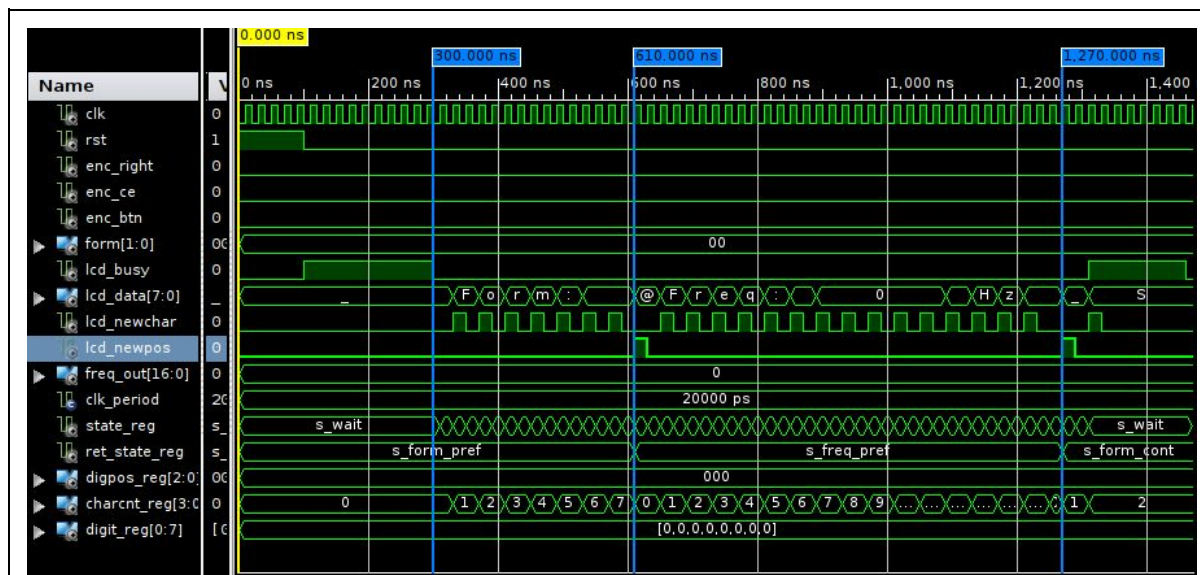


Fig 3: Es ist zu erkennen dass das Modul solange im *S\_WAIT* State bleibt bis das *lcd\_busy* Signal auf 0 geht (Marker 1, blau). Anschliessend folgt das Rausschreiben der statischen Texte, welche auf *lcd\_data* lesbar sind.

Weiter sieht man bei Marker 2 das eine neue Position übermittelt wird (ASCII Interpretation auf *lcd\_data* sinnlos), bevor weitere statische Texte übermittelt werden. Ab Marker 3 wird nochmals ein Text übermittelt, aber hierbei wird ein "langsameres *lcd\_busy*" simuliert. Es ist zu erkennen dass die State-Machine auch hierbei korrekt im State *s\_wait* bleibt.

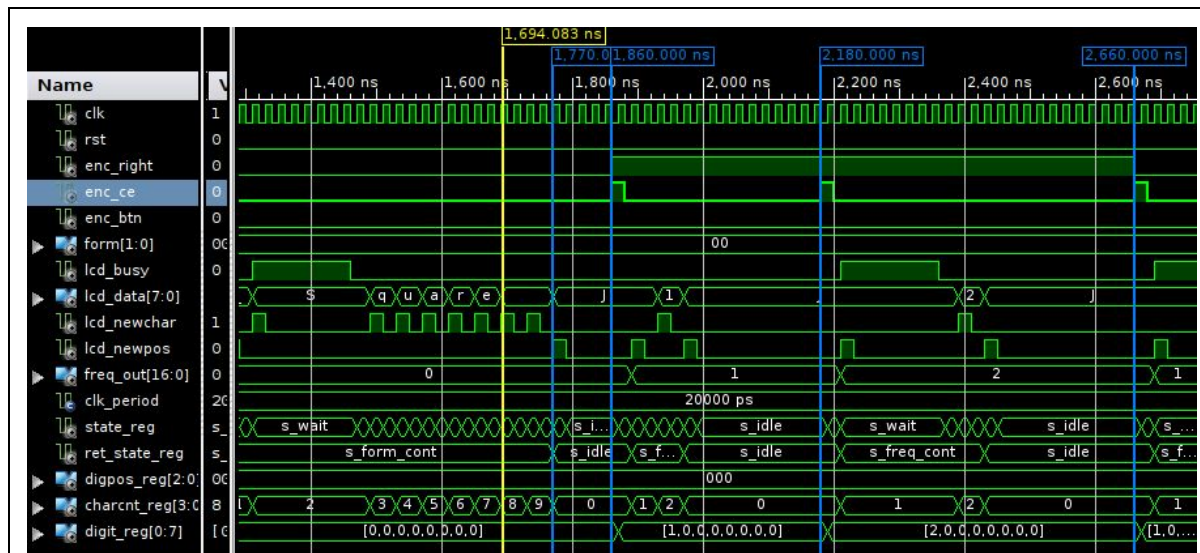


Fig 4: Am Anfang des Bildes ist noch das Ende der statischen Texte zu erkennen ("Square"). Anschliessend wechselt die State-Machine zum ersten Mal in den *S\_IDLE* State (kurz nach Marker 1, blau). Vorher wird noch die aktuelle Cursor-Position gesetzt (Marker 1), auf die 1. Dezimalstelle. Die Ausgangsfrequenz *freq\_out* ist noch auf 0 Hz, ebenso die einzelnen Dezimalstellen in *digit\_reg*. Bei Marker 2 wird anschliessend eine Rechtsdrehung des Inkremental-Gebers simuliert, woraufhin sich *digit\_reg* und *freq\_out* erhöhen. Bei Marker 3 wird nochmals eine Rechtsdrehung simuliert, jetzt aber mit einem "langsamen *lcd\_busy*". Bei Marker 4 wird schliesslich eine Linksdrehung simuliert woraufhin sich die Ausgangsfrequenz von 2 Hz zurück auf 1 Hz ändert.

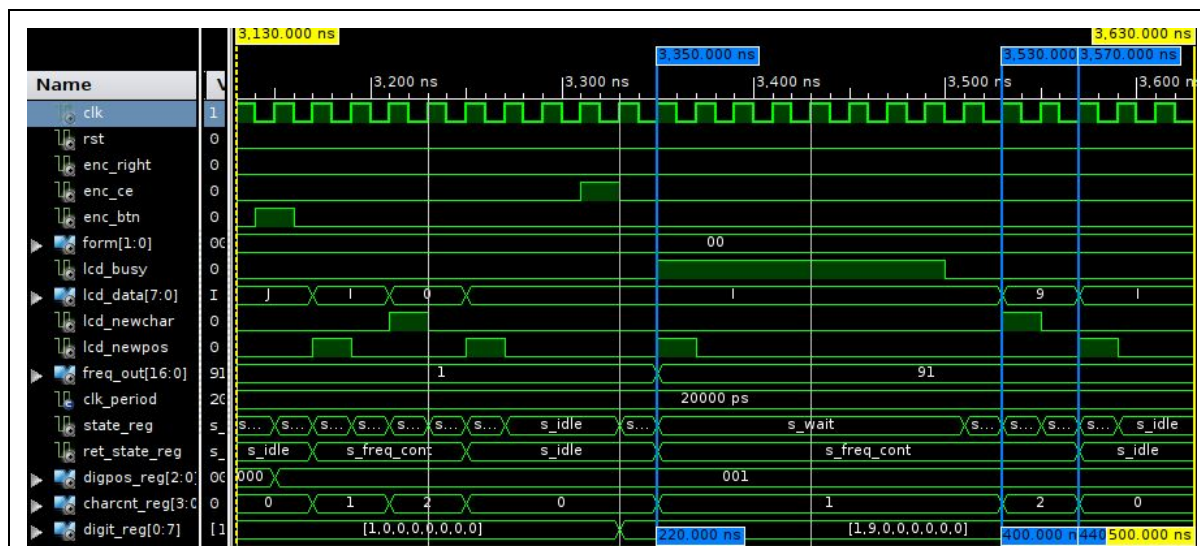


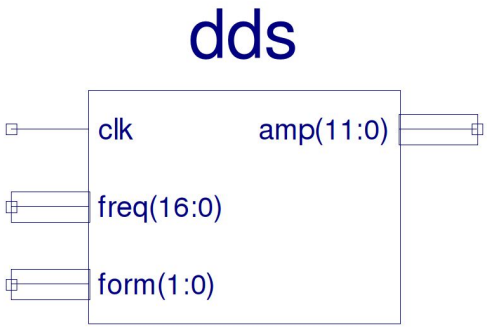
Fig 5: Zu Beginn des Bildes wird erstmalig ein Tastendruck auf den Inkrementalgeber simuliert (*enc\_btn*). Anschliessend ist zu beobachten wie die aktive Dezimalstelle



(*digpos\_reg*) verändert wird, die Cursor-Position angepasst wird, und die Zahl '0' erneut geschrieben wird. Bei Marker 1 (blau) wird schliesslich eine weitere Linksdrehung simuliert, woraufhin erstmal im State *S\_WAIT* verweilt wird. Bei Marker 2 wird dann schliesslich die 2. Dezimalstelle erneut aufs Display geschrieben. Abschliessend wird bei Marker 3 die Cursorposition wieder korrigiert und den Idle State zurückgekehrt.

## 2.3 DDS

### Übersicht

	<p>Das DDS-Modul implementiert "Direct Digital Synthesis" für 4 verschiedene Signalformen (Rechteck, Sägezahn, Dreieck, Sinus). Die gewünschte Signal-Form und Frequenz kann mithilfe der Eingänge eingestellt werden. Am Ausgang wird dann die Amplitude des entsprechenden Signals ausgegeben.</p>
<p>Fig 6: Schemasympol des DDS-Moduls</p>	<p><u>Ein und Ausgänge:</u>  <b>clk:</b> Clock Eingang  <b>freq:</b> Gewünschte Frequenz in Hz  <b>form:</b> Eingang zur Auswahl der Signalform. Beschreibung siehe Kap 2.1  <b>amp:</b> Aktuelle Amplitude des gewählten Signals</p> <p><u>Generische Parameter:</u>  <b>clk_freq:</b> Frequenz des Clocksignals in Hz  <b>freq_res:</b> Breite des Frequenz ausgangs in Bits  <b>adc_res:</b> Amplituden-Auflösung in Bits  <b>acc_res:</b> Auflösung des Phasen-Akkumulators in Bits  <b>phase_res:</b> Effektive Phasen-Auflösung für die Sinus-Signalform (Lookup Table)</p>

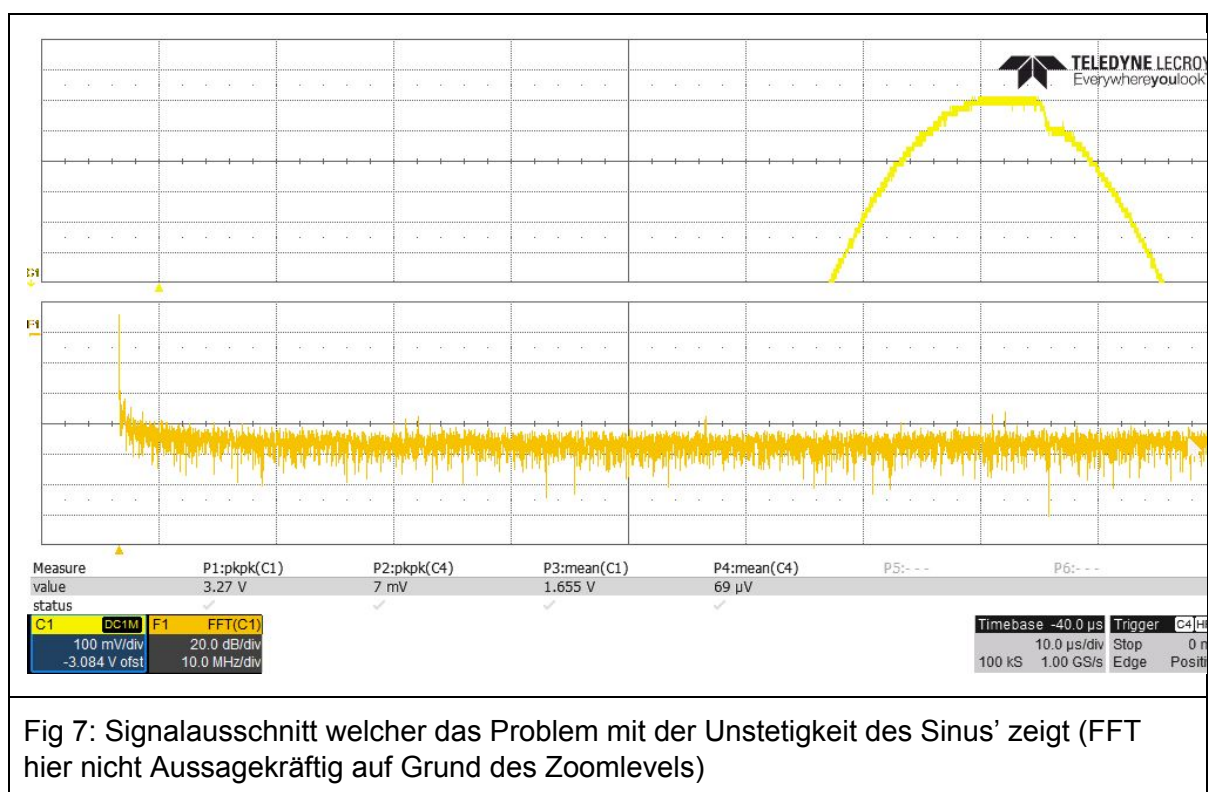
## Implementierung

Die Implementierung wurde anhand der Ideen aus dem Dokument *Fundamentals of Direct Digital Synthesis (DDS)*

(<http://www.analog.com/media/en/training-seminars/tutorials/MT-085.pdf>) durchgeführt.

Die Sinus Signalform wurde im ROM abgelegt, während die restlichen Signalformen zur Laufzeit berechnet werden. Alle Signalformen besitzen einen DC-Anteil von 50% (also in unserem Falle  $2^{11}$ ).

Die Sinus-Signalform wurde einmal komplett im ROM abgelegt und testweise auch nur als  $\frac{1}{4}$  Periode. In beiden Fällen konnten wir jedoch eine Unstetigkeit erkennen, wie das folgende Bild verdeutlicht.



Leider konnten wir das Problem noch nicht weiter eingrenzen. Gerade wenn die Signalform als komplettes abgelegt ist, dürften solche "Übergangseffekte" theoretisch nicht auftreten. Abgesehen von diesem Problem funktioniert das DDS-Modul aber einwandfrei, wie die nachfolgenden Bilder zeigen.

## Test

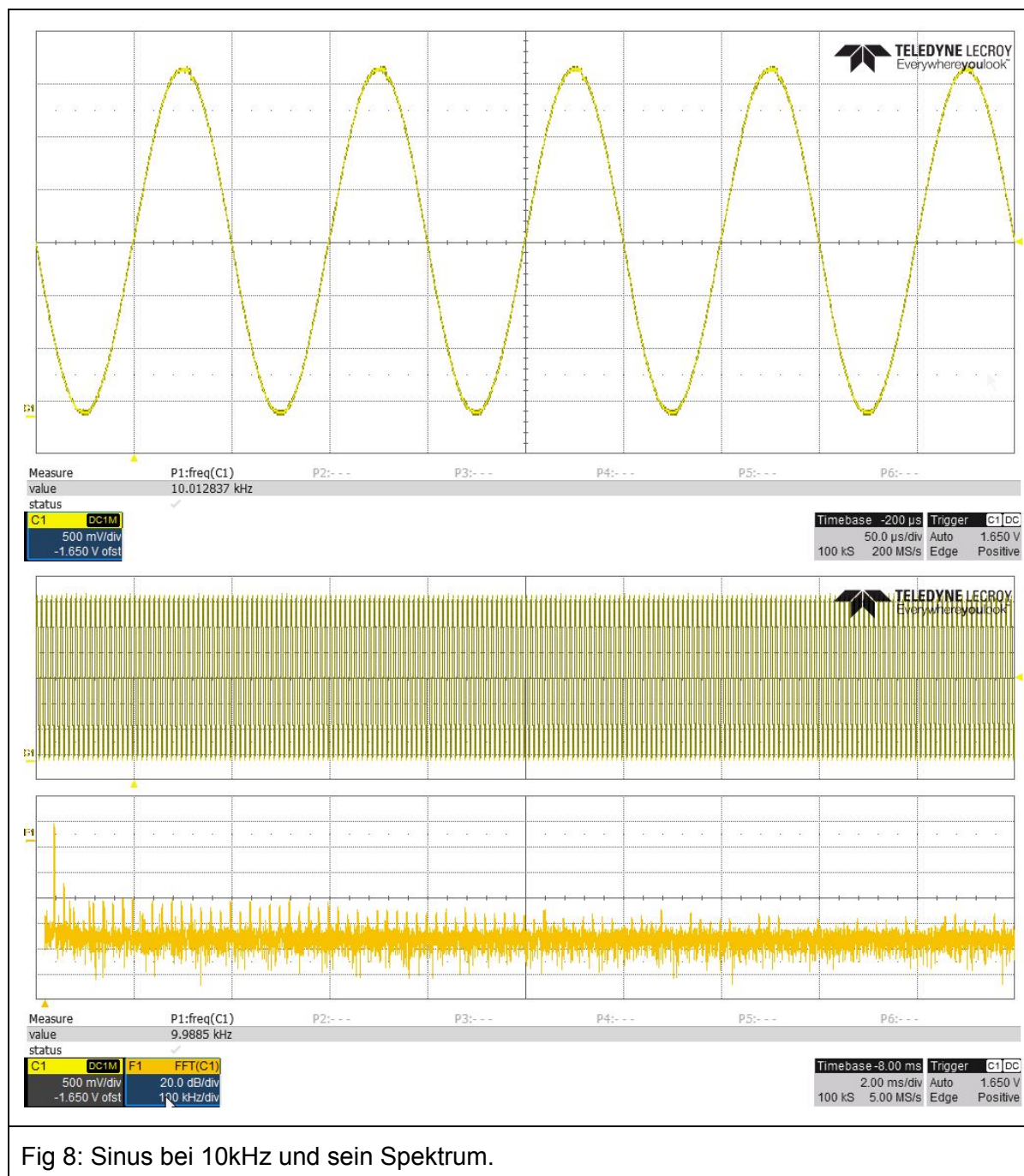


Fig 8: Sinus bei 10kHz und sein Spektrum.

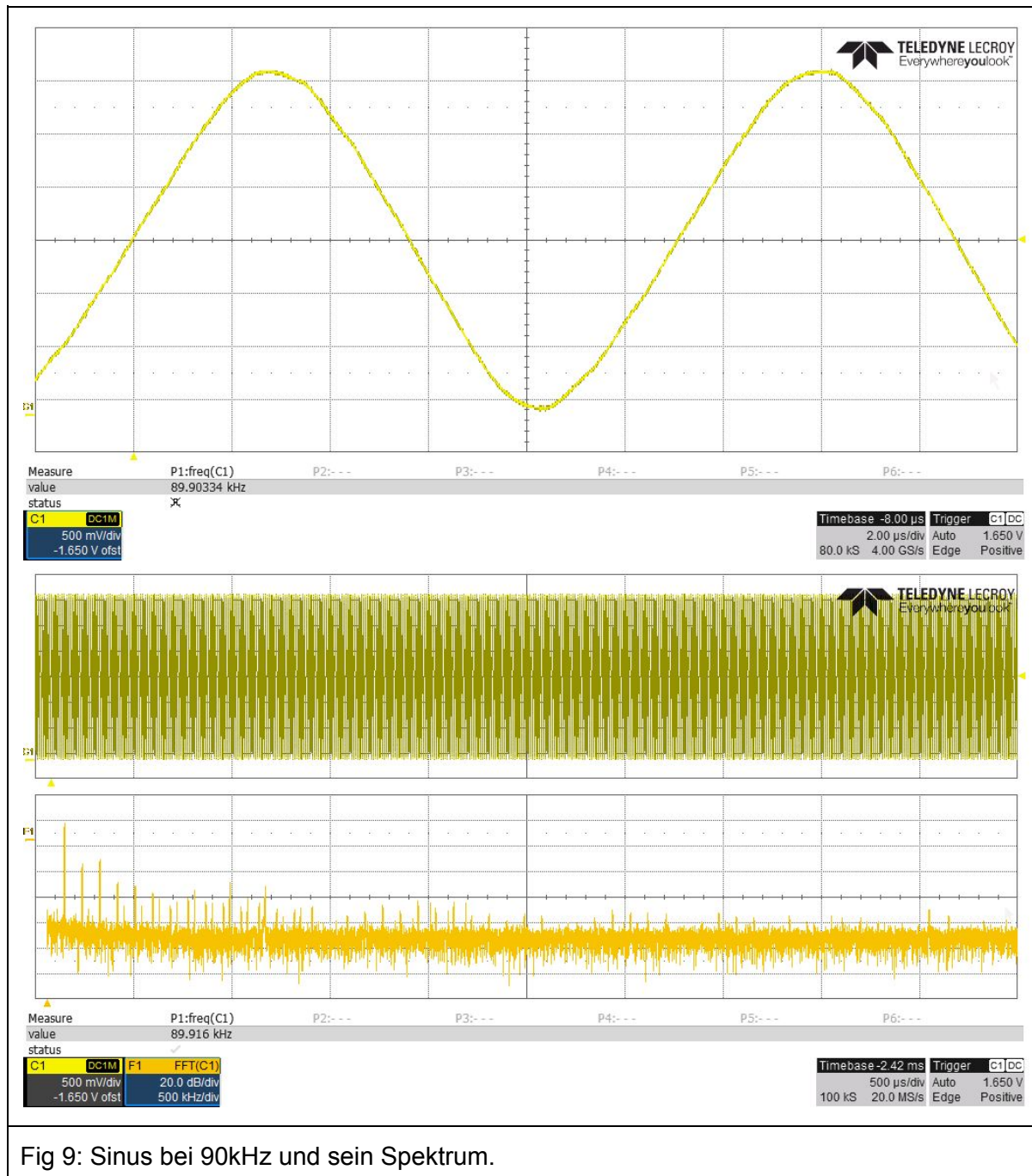


Fig 9: Sinus bei 90kHz und sein Spektrum.



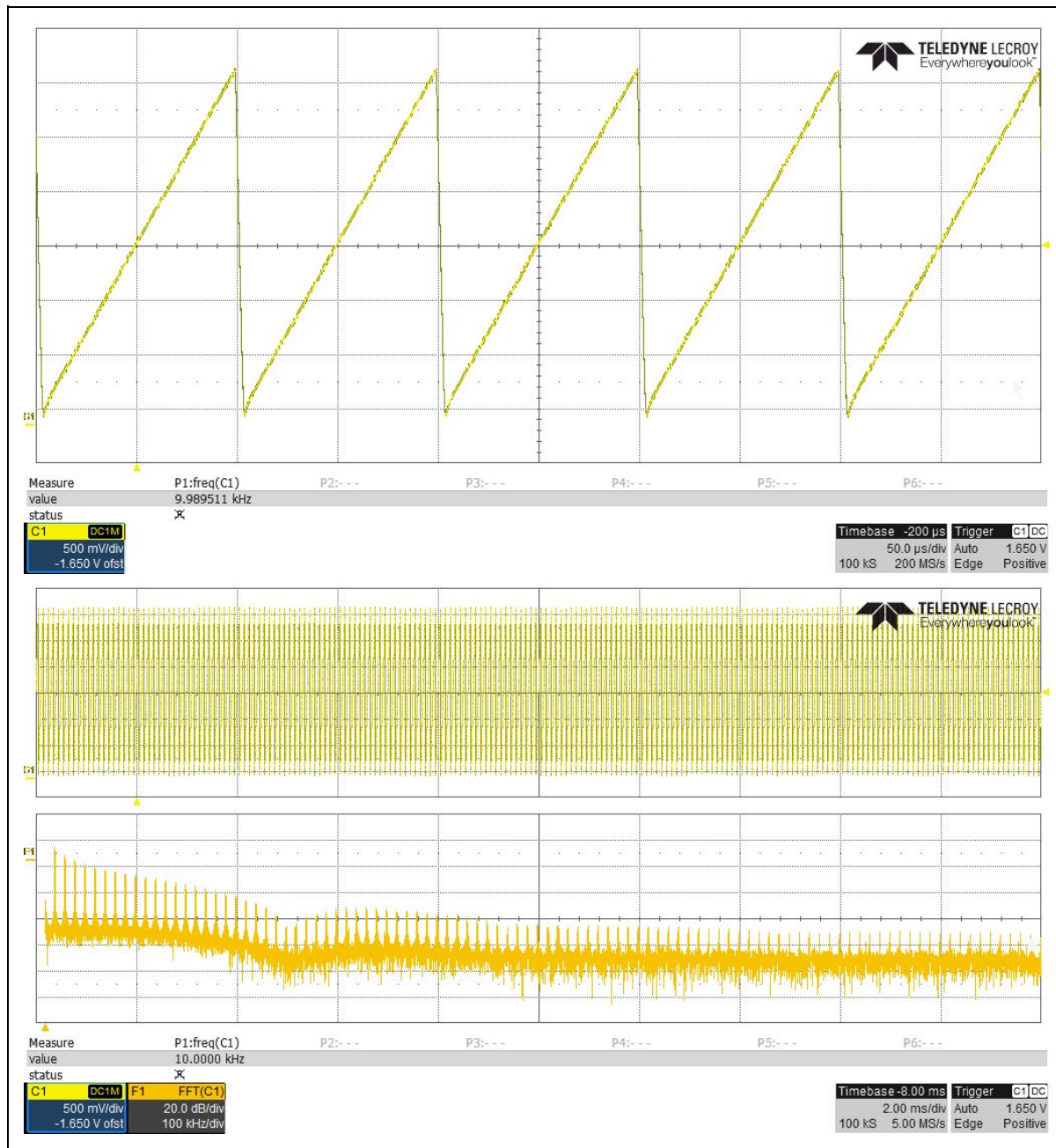


Fig 10: Sägezahn bei 10kHz und sein Spektrum. Zu erkennen ist hier schon die nicht ideale Flankensteilheit.

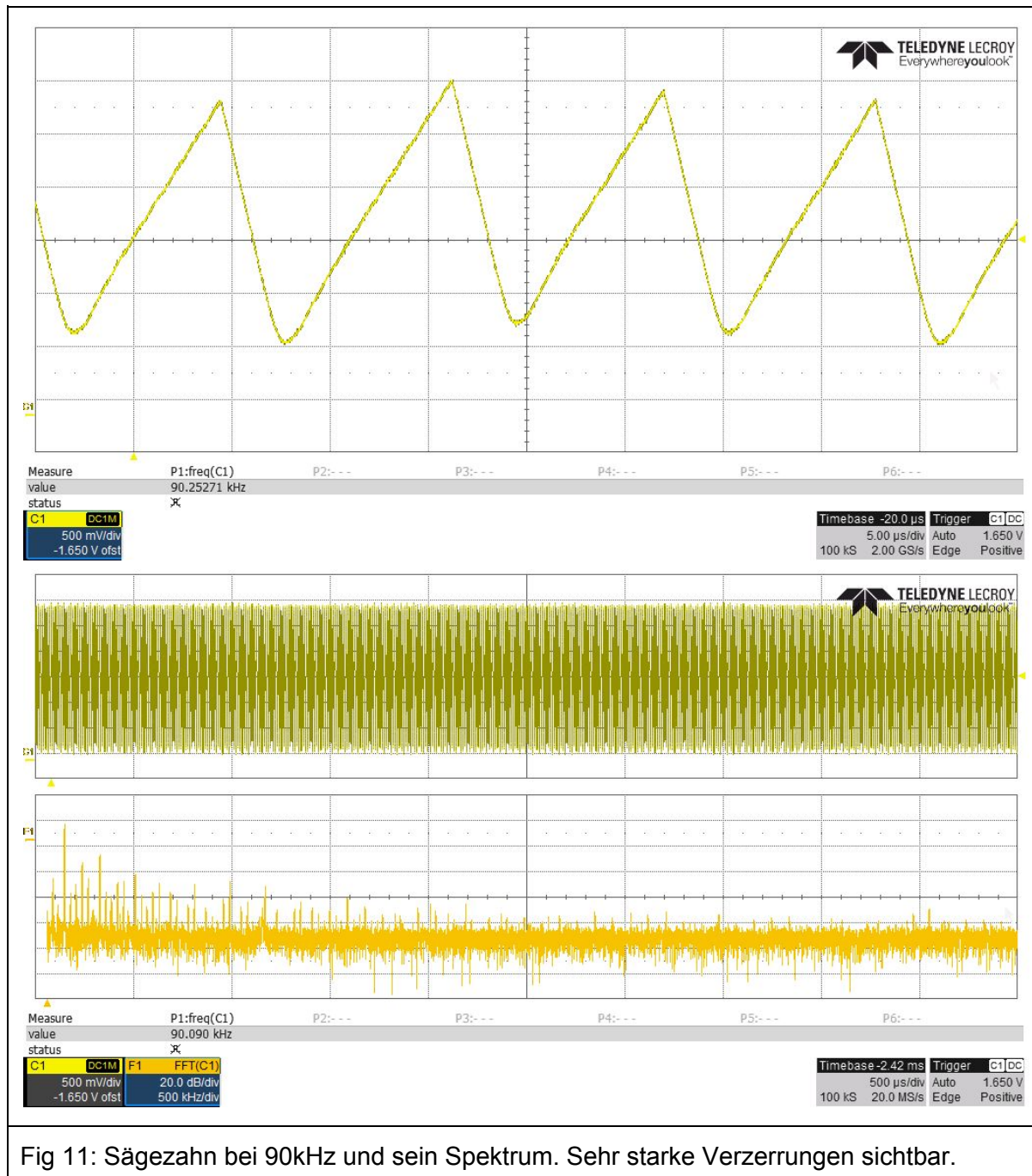


Fig 11: Sägezahn bei 90kHz und sein Spektrum. Sehr starke Verzerrungen sichtbar.

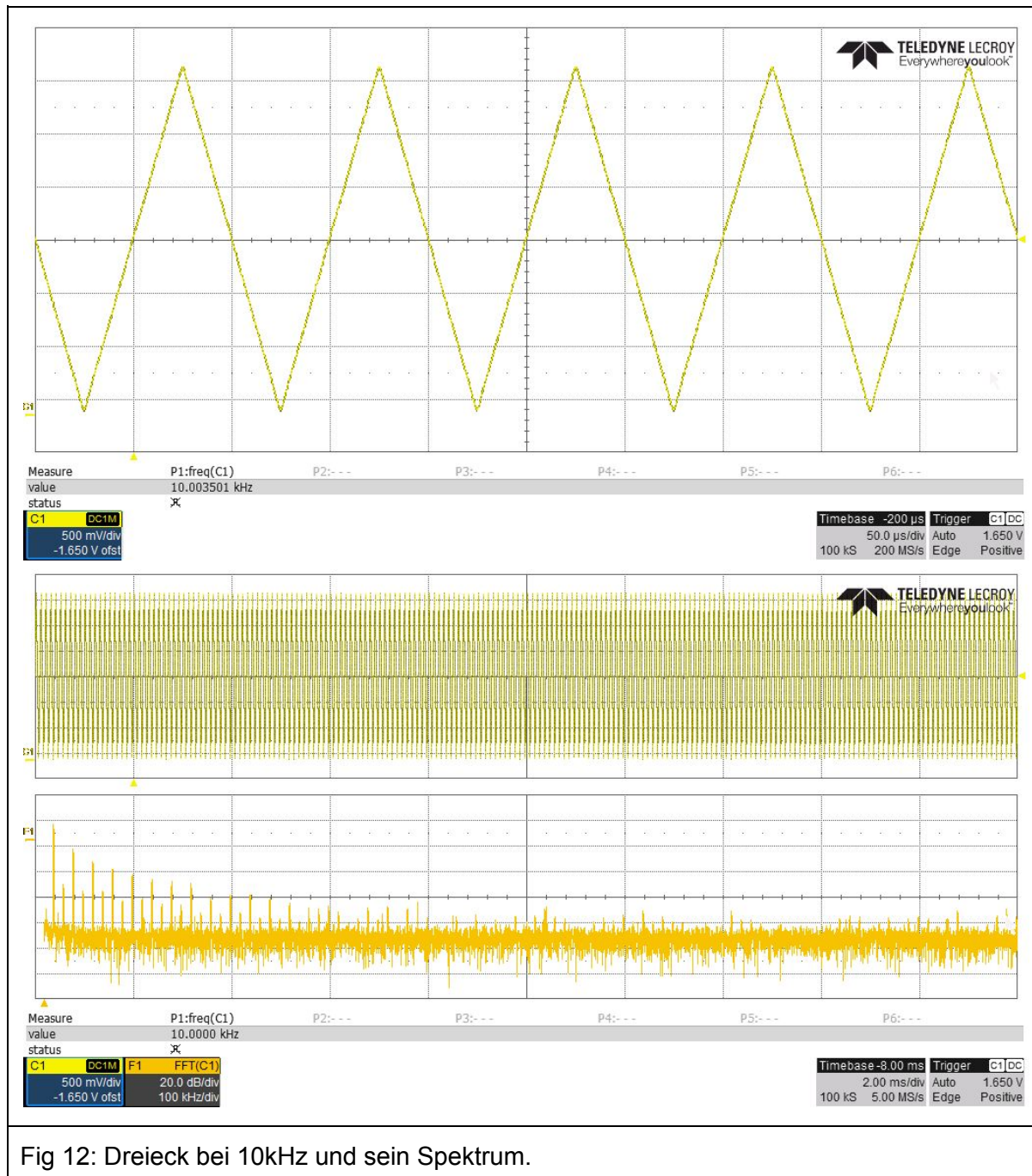


Fig 12: Dreieck bei 10kHz und sein Spektrum.



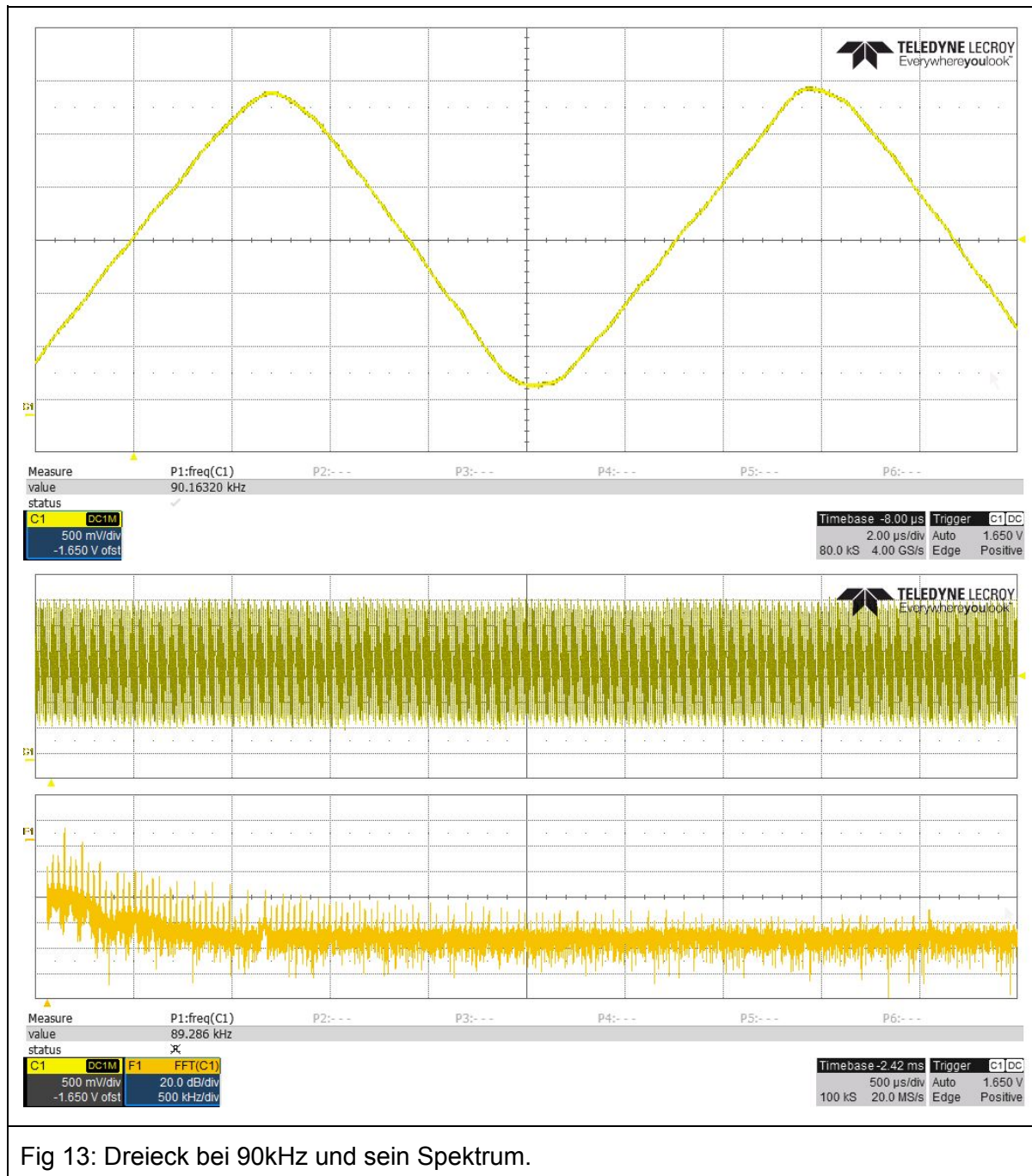


Fig 13: Dreieck bei 90kHz und sein Spektrum.

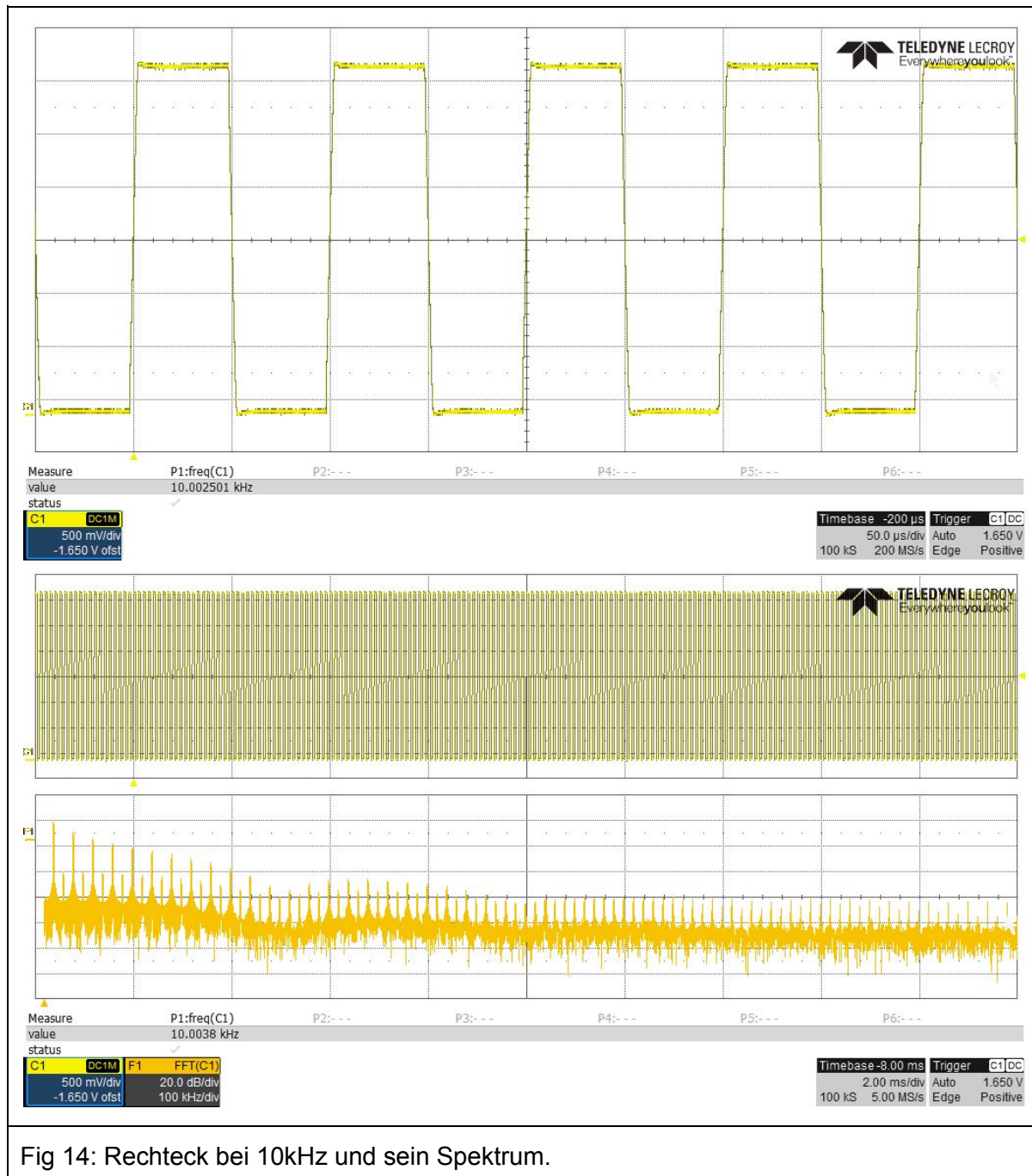


Fig 14: Rechteck bei 10kHz und sein Spektrum.

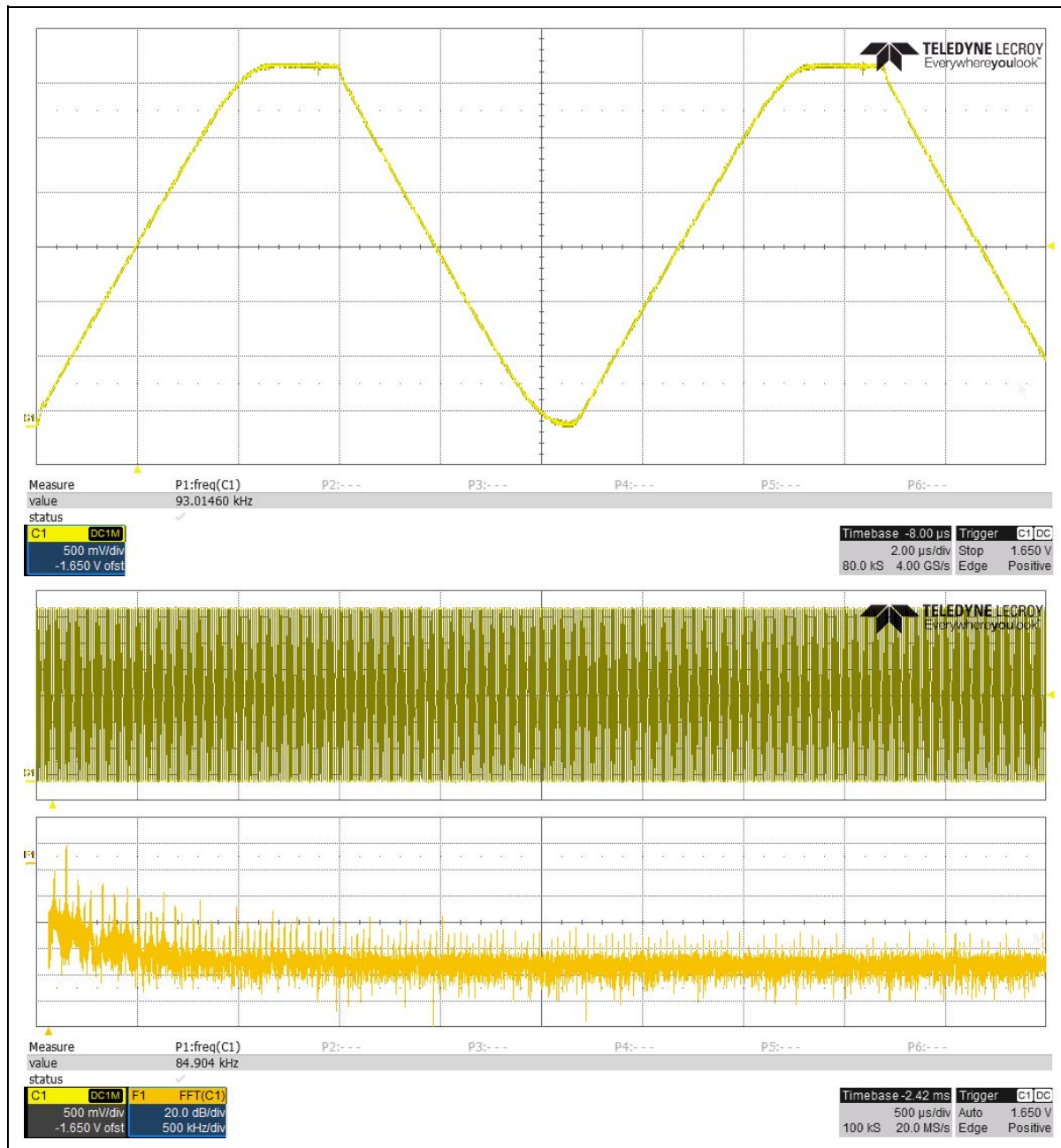
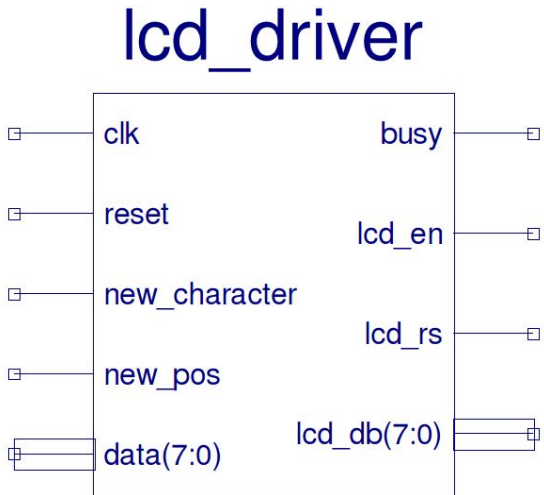


Fig 15: Rechteck bei 90kHz und sein Spektrum. Die Signalform ist stark verzerrt. Begrenzend wirkt wieder die Flankensteilheit des DACs.

## 2.4 LCD Driver

### Übersicht

	<p>Das LCD Modul ist für die Initialisierung und Ansteuerung dieses Controllers zuständig. Das auf dem Spartan Board verbaute Display verfügt über ein ST7066U Controller.</p>
<p>Fig 16: Schemasympol des LCD-Divers</p>	<p><u>Ein und Ausgänge:</u>  <b>clk:</b> Clock Eingang  <b>reset:</b> High aktiver, asynchroner reset  <b>new_character:</b> Dieser Eingang teilt dem Modul mit, dass ein neues Ascii Zeichen vom Datenbus gelesen werden kann.  <b>new_pos:</b> Dieser Eingang teilt dem Modul mit, dass eine neue Cursor Position vom Datenbus gelesen werden kann.  <b>data:</b> 8Bit Datenvektor auf welchem entweder ein Ascii Zeichen oder eine Cursorposition angelegt werden kann.  <b>busy:</b> Ausgang welcher anzeigt, dass das Modul gerade mit Senden beschäftigt ist.  <b>lcd_en:</b> lcd enable Leitung  <b>lcd_rs:</b> lcd register select Leitung  <b>lcd_db:</b> lcd Datenleitungen</p> <p><u>Generische Parameter:</u>  <b>NBITS:</b> Bitbreite der intern verwendeten Zählerbausteine.  <b>clk_freq:</b> Clockfrequenz des Inputclocks, zur Berechnung der Zählerwerte für die Pausenlängen.  <b>wait_init:</b> Länge der Startup Pause. Gemäss Datenblatt 40ms.  <b>wait_between:</b> Länge der Pause zwischen den einzelnen Einstellungen während der</p>



	Initialisierung des Displaycontrollers. Gemäss Datenblatt 37us. <b>wait_pause:</b> Länge der Pause für die Display Clear Anweisung. Gemäss Datenblatt 1.52ms.
--	---

## Implementierung

Das gesamte Modul ist als State Machine realisiert. Die Initialisierungssequenz wurde gemäss Datenblatt implementiert. Das Modul nimmt jeweils ein Ascii Zeichen oder eine Cursorposition an und schickt diese Daten ans LCD, sobald dieses initialisiert wurde.

Gesteuert wird das Modul über die Eingänge *new\_character* und *new\_pos*, sowie den Datenvektor *data*. Mittels den Eingängen *new\_character* und *new\_pos* kann dem lcd\_driver mitgeteilt werden ob auf dem Datenbus ein Ascii Zeichen oder eine Cursorposition anliegt. Während des Sendens ist der Ausgang *busy* gesetzt um dem vorgeschalteten Funktionsblock mitzuteilen, dass gerade ein Sendevorgang abläuft. Der Reset kann mittels Setzen des *reset* Eingangs asynchron ausgeführt werden und führt dazu, dass sich das Display neu initialisiert.

## Testbench

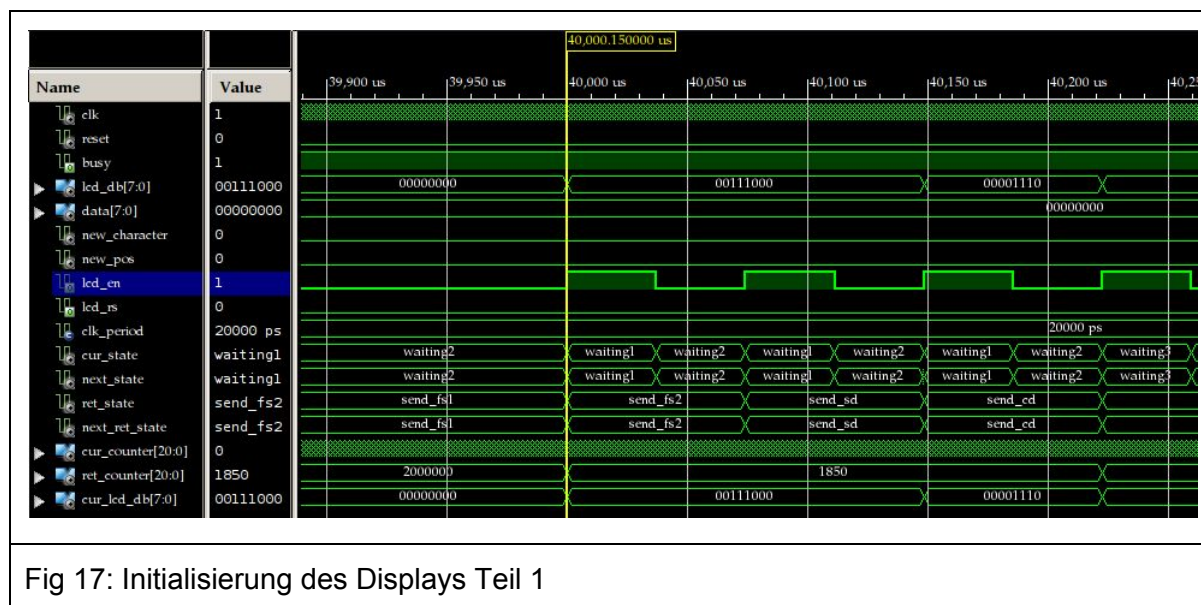


Fig 17: Initialisierung des Displays Teil 1

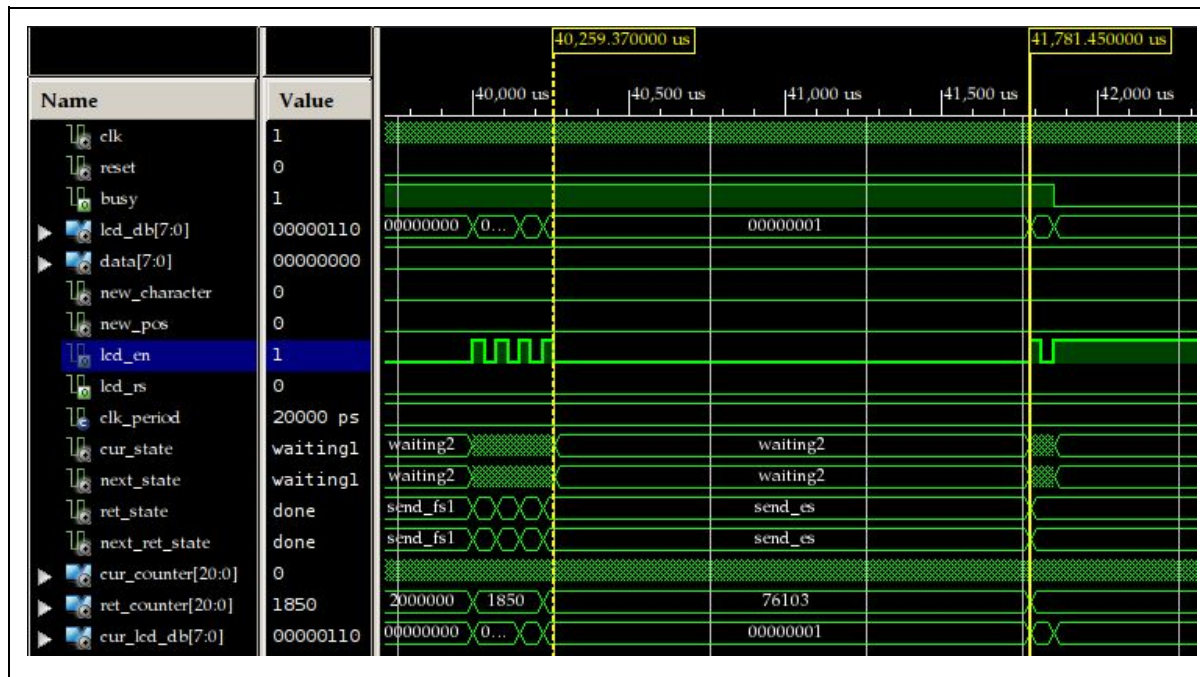


Fig 18: Initialisierung des Displays Teil 2

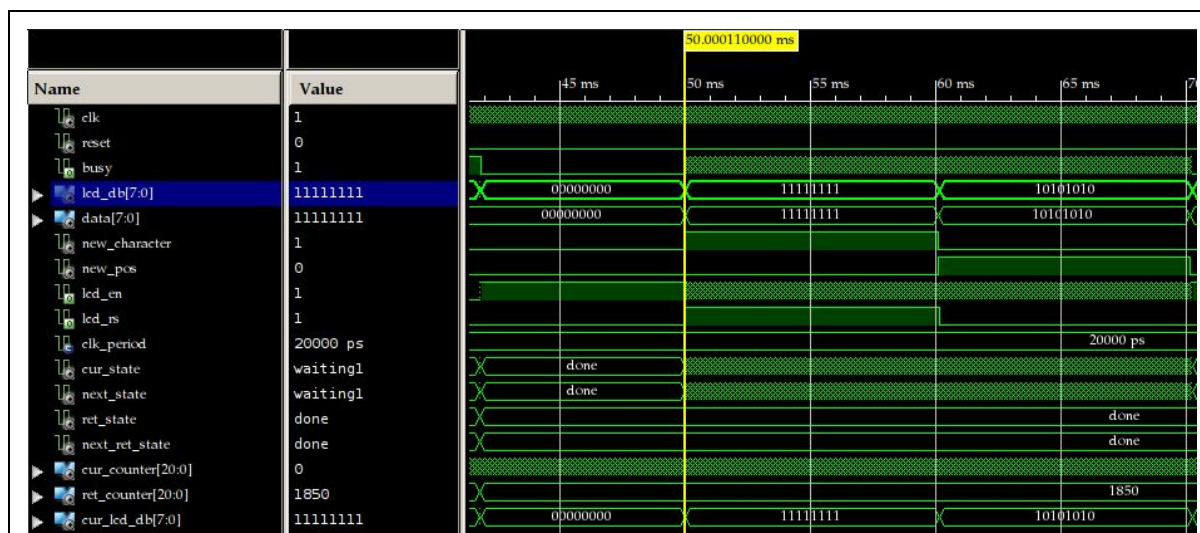
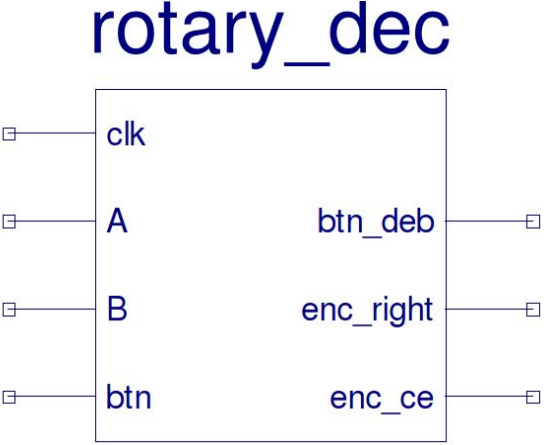


Fig 19: Senden eines Zeichens und setzen des Cursors

## 2.5 Rotary Decoder

### Übersicht

	<p>Das <b>rotary_dec</b> Modul wertet das Signal des Drehgebers aus, welches aus zwei um 90° Phasenverschobenen Rechtecksignalen A und B besteht. Anhand der Phase lässt sich die Drehrichtung bestimmen.</p>
<p>Fig 20: Schemasympol des Rotary-Decoders</p>	<p><u>Ein und Ausgänge:</u>  <b>clk:</b> Clock Eingang  <b>A:</b> A-Signal des Drehgebers (nicht entprellt)  <b>B:</b> B-Signal des Drehgebers (nicht entprellt)  <b>btn:</b> Push-Button Signal des Drehgebers (nicht entprellt)  <b>btn_deb:</b> Entprelltes Buttonsignal  <b>enc_right:</b> Drehrichtungsausgang  <b>enc_ce:</b> Clock Enable Ausgang des Moduls</p> <p><u>Generische Parameter:</u>  <b>clk_freq:</b> Frequenz des Clocksignals in Hz  <b>debounce_time:</b> Zeit während dessen die Eingänge entprellt werden sollen</p>

## Implementierung

Unsere Implementierung des rotary\_dec Moduls ist angelehnt an die VHDL Implementierung von mikrocontroller.net


([http://www.mikrocontroller.net/articles/Drehgeber#Beispielcode\\_in\\_VHDL](http://www.mikrocontroller.net/articles/Drehgeber#Beispielcode_in_VHDL)). Die Idee mit der Dekodierung der Ausgangssignale mittels LUT haben wir von dort übernommen.

Anschliessend haben wir das Modul aber um eine eigene Entprellung erweitert. Die Entprellung der Signal A,B, und Btn funktioniert so dass eine Werteänderung nur alle 10ms akzeptiert wird. Dies haben wir mithilfe eines Zählers (pro Signal) realisiert.



## 2.6 SPI Driver

### Übersicht

	<p>Das SPI-Modul ist für die Ansteuerung des DAC-Chips LTC2624 zuständig. Die angelegte Amplitude (<i>val</i>) wird so oft und schnell wie möglich via SPI an den DAC gesendet.</p>
<p>Fig 21: Schemasympol des SPI-Teibers</p>	<p><u>Ein und Ausgänge:</u>  <b>clk:</b> Clock Eingang  <b>rst:</b> High-aktiver, asynchroner Reset  <b>val:</b> Amplitude die ausgegeben werden soll  <b>sck:</b> SPI clock  <b>cs:</b> SPI chip-select  <b>mosi:</b> SPI data</p> <p><u>Generische Parameter:</u>  <b>clk_freq:</b> Frequenz des Clocksignals in Hz  <b>adc_res:</b> Amplituden-Auflösung in Bits</p>

### Implementierung

Das SPI Modul ist implementiert als State-Machine mit 2 States: *S\_IDLE* und *S\_WORK*. Beim wechsel von *S\_IDLE* to *S\_WORK* wird jeweils der aktuelle Wert *val* in ein Schieberegister übernommen. Anschliessend wird Bit für Bit übertragen bis alle 24 Bits gesendet wurden.

## Test

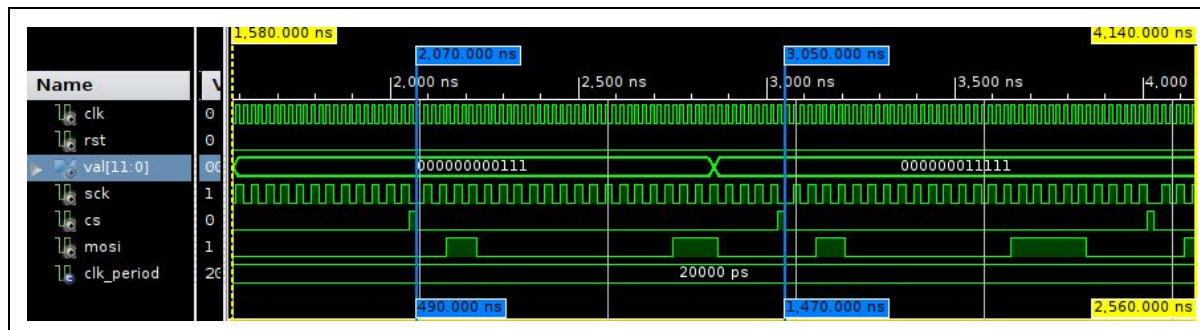


Fig. 22: In der Testbench sind 2 ganze Übertragungen (Nutzdaten: 0x007 und 0x01F) zu sehen (startend bei den blauen Markern). Nebst den Nutzdaten (12 bit) werden jeweils auch noch Steuerdaten (8 bit) gesendet.

Es ist zu erkennen dass die Daten jeweils eine Clock-Periode bevor SCK auf High geht, angelegt werden. Ausserdem ist zu erkennen dass ein Wechsel der Eingangsdaten (val) erst bei der nächsten Übertragung übernommen wird.

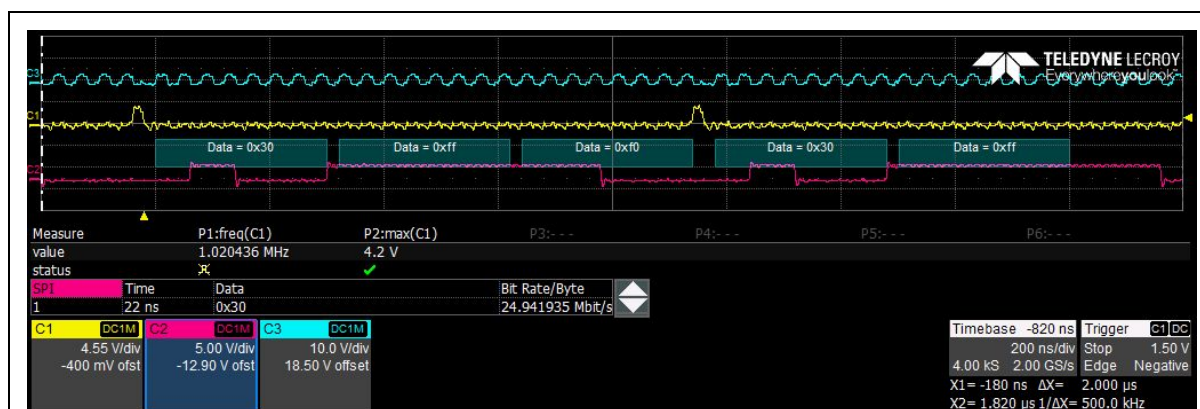


Fig. 23: Aufzeichnung einer Übertragung (0xFFFF) mittels Oszilloskop. C3 (türkis) SCK, C1 (gelb) CS, C2 (violett) MOSI

### 3 Statistiken

<b>Beschreibung</b>	<b>Verwendet</b>	<b>Verfügbar</b>	<b>Auslastung</b>
FlipFlops	266	11776	2%
4-input LUTs (als Logik)	800	11776	6%
4-input LUTs (als verbinder)	105	11776	<1%
Slices	507	5888	8%
IOs	27	372	7%
BUFGMUXs	1	24	4%
MULT18X18SIOs	5	20	25%

Maximale Logiklevel	45
Anzahl Pfade	462217
Minimale Periodendauer (Post-PAR)	7.091ns
Maximale Clock Frequenz	58.510MHz

Gemäss obiger Statistiken (Resources Summary) erkennt man, dass das FPGA nicht stark ausgelastet wird. Eventuell könnte man die Bitbreiten einiger Zähler noch etwas senken. An einigen Stellen könnte man zudem die Bitbreiten automatisch mithilfe des 2er-Logarithmus berechnen lassen (wie wir dass an anderen Stellen auch schon tun), was die Leserlichkeit und die einfache Adaptierfähigkeit des Codes erhöht. Zudem könnte man die Logiktiefe senken (und damit auch die Maximale Taktfrequenz erhöhen) indem man kombinatorische/mathematische Ausdrücke aufteilt in eine Pipeline - dies würde aber wiederum die Anzahl der verwendeten FlipFlops erhöhen.

## 4 Fazit

Das Projekt war für uns sehr lehrreich. Wir konnten alle Vorgaben erfüllen und haben dadurch auch sehr viel Neues gelernt. Durch den konsequenten Einsatz des Versionierungssystems GIT konnten wir die Arbeit gut aufteilen und somit sehr strukturiert arbeiten.

Die Ansteuerung des ST7066U Display-Controllers in VHDL war lehrreich aber auch sehr aufwändig (zumindest aufwändiger als auf einem Mikrokontroller). In der State-Machine des Display-Treibers fielen zwischen den einzelnen Zuständen teils unterschiedliche Wartezeiten an, was in VHDL eher mühsam umzusetzen war. Das Prinzip hinter der Direct Digital Synthesis war uns vorher gänzlich unbekannt und daher sehr spannend. Es war sehr interessant dieses System praxisbezogen anwenden zu können. Natürlich könnte man noch mehr Zeit investieren in eine intuitivere Bedienung, aber dass war ja nicht der Fokus des Projekts.

Unser Fazit für das Projekt fällt durchaus positiv aus und wir sind mit der Arbeit zufrieden.