Praktikum
Schaltungsdesign mit FPGA

Luis David Anchia Saenz
Matrikelnummer: 1754783

1 Projekt Faltungscodierer

1.1 Encoder mit Schieberegister

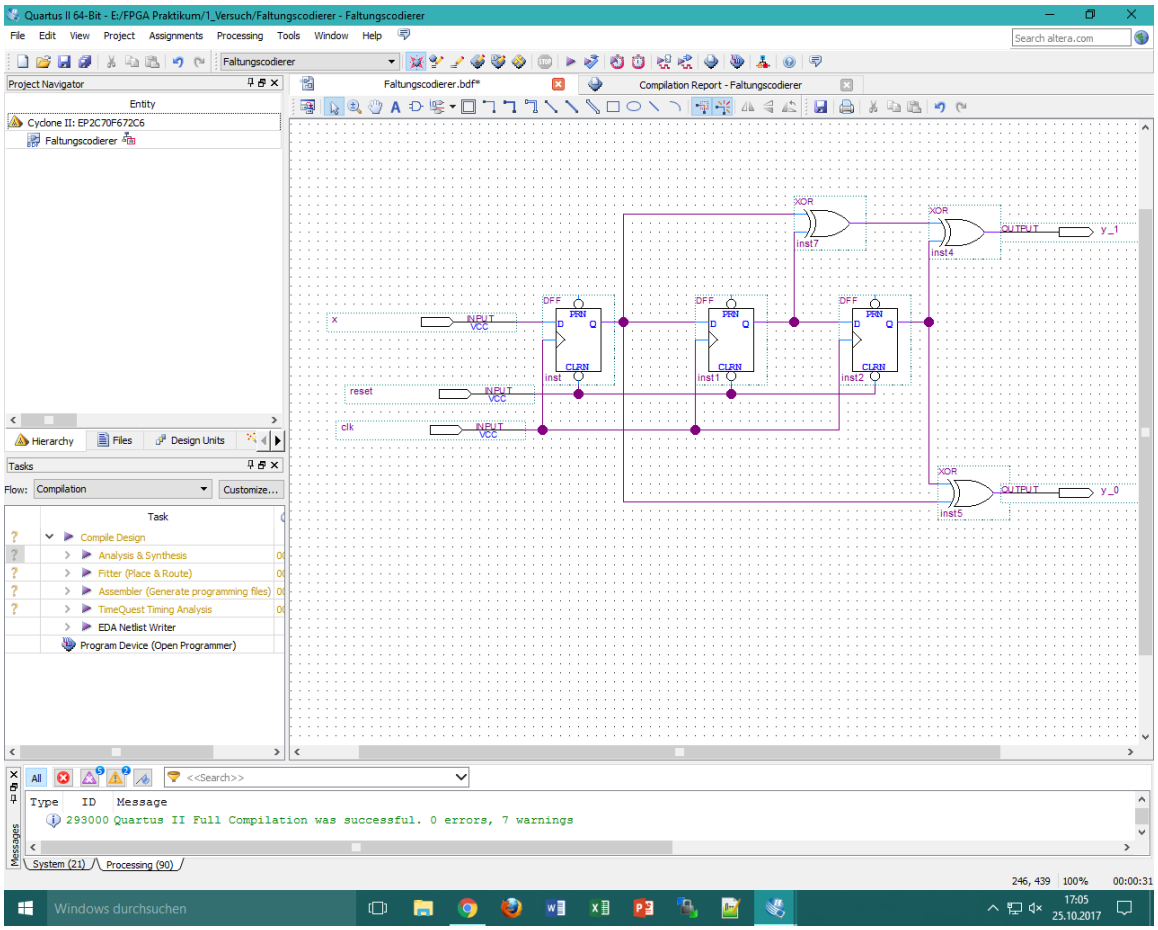

Abbildung 1-1 Blockschaltung des Encoders mit Schieberegister

Abbildung 1-1 zeigt die Blockschaltung eines Faltungscodierers mit einer Gedächtnistiefe m = 2 und den Generatorpolynomen $g^{(0)} = 5_8 = 101_b$ und $g^{(1)} = 7_8 = 111_b$. Die Polynome werden durch die Schaltung der Verschiedene Flip-Flop Ausgänge implementiert. In diesem Fall, das Polynom $g^{(0)}$ würde unten im Blockschaltbild und das Polynom g(1) würde oben implementiert. Obwohl die Gedächtnistiefe m = 2 ist, sind 3 Flip-Flops notwendig. Das Flip-Flop links wird benutzt, um den Eingangswert für eine Periode des Clocksignals konstant zu halten, da diesen Wert von den XOR-Glieder bearbeitet wird und sollte sich inzwischen nicht ändern.

Insgesamt wurden 6 Glieder benutzt, Ein- und Ausgänge ausgenommen. Diese sind die 3 Flip-Flops und die 3 XOR-Glieder. Man könnte aber die Anzahl auf fünf Glieder verkleinern, wenn man ein XOR-Glied entfernen würde. In diesem Fall würde man das XOR unten behalten und die zwei obere durch ein einziges XOR ersetzen. Da die Werte der Eingang und des letzten Flip-Flops

schon unten verarbeitet werden, braucht man nur dieses Ergebnis mit dem Wert am Ausgang des zweiten Flip-Flops mit einem XOR zusammen zu bearbeiten.

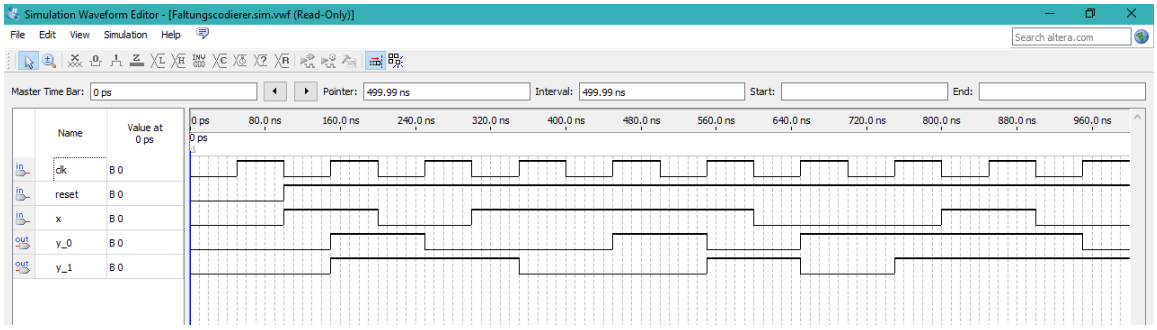

Abbildung 1-2: Funktionale Simulation der Blockschaltung

Um die Funktionalität des Faltungscodierers zu überprüfen sind die Simulationen in Abbildungen 1-2 und 1-3 erstellt worden. Am Eingang x wurde die Bitfolge $B9_h = 10111001_b$, MSB zuerst, gefolgt von Nullen gegeben. Das Ergebnis kann man aus der funktionalen Simulation korrekt lesen, wenn y_1 als MSB und y_0 al LSB interpretiert werden. Das Ergebnis in Abbildung 1-3 zeigt kurze Fehler in der Ausgangswerte der Schaltung. Man erkennt Spikes, da wo sich ein Wert kurzfristig auf 1 stellt, obwohl es 0 sein sollte. Ähnliche Glitches findet man auch, wenn ein Wert kurz den Wert 0 nimmt und dann wieder auf 1 zurückkehrt. Dies kann man erklären mit der Schaltung. Die XORs bearbeiten seine Eingangswerte praktisch kontinuierlich. Dies ist ein Problem für das XOR oben rechts, weil es einen falschen Wert am Ausgang stellt, wenn die Werte im Schieberegister sich schieben, aber das XOR vorher noch nicht sein Ausgang gewechselt hat. Dann stellt es den korrekten Wert am Ausgang, wenn das XOR bevor sein Ausgang aktualisiert hat.

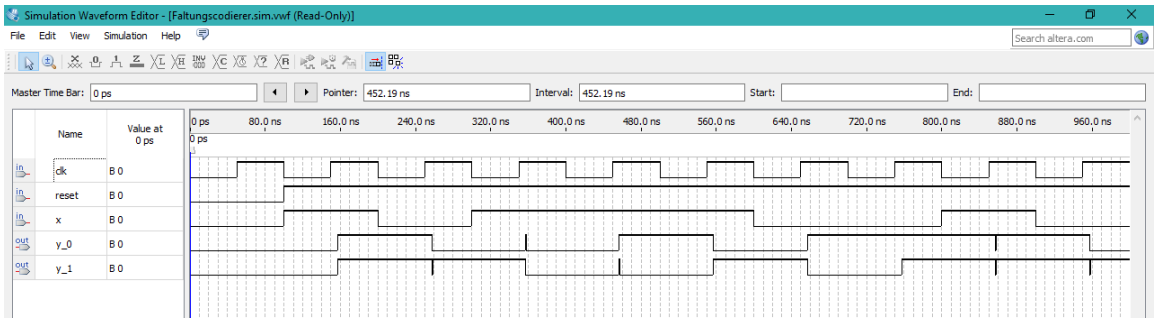

Abbildung 1-3: Timing Simulation der Blockschaltung

1.2 Encoder mit VHDL

Ein Zustandsautomat würde mit VHDL erstellt, der die gleiche Funktion wie der Blockschaltung in Abbildung 1-1 hat. Zum Vergleich steht das Ergebnis der Simulation in Abbildung 1-4 vor.

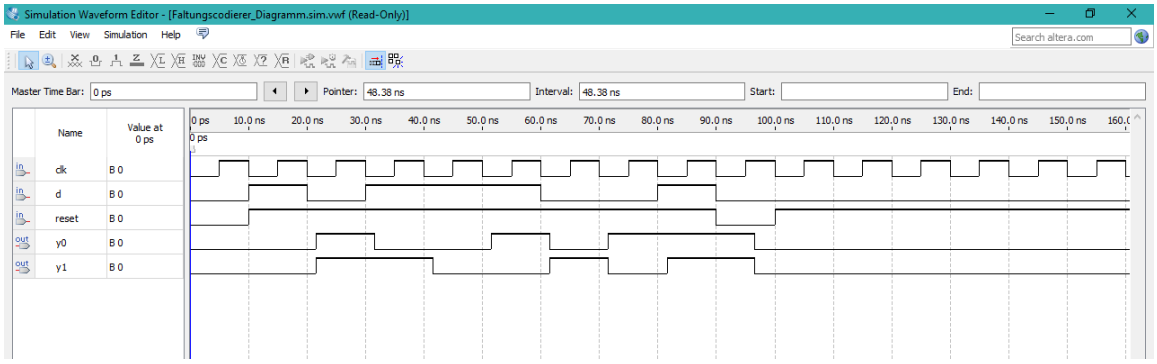

Abbildung 1-4: Timing Simulation des Zustandsautomats mit VHDL

Wenn man in das Ergebnis zoomt (Abbildung 1-5), erkennt man, dass die Ausgänge nicht gleichzeitig ihre Werte annehmen. Mit der Platzierung durch den Compiler ist y_0 schneller als y_1. Dies passiert wegen der Platzierung der Elemente auf dem FPGA. Die Signallaufzeiten können unterschiedlich sein und diese Zeit wird größer, je weiter voneinander entfernt die Komponenten sind. Dies zeigt Abbildung 1-6. Da wurden die Gatter, die den Ausgang y_0 bestimmen, sehr weit weg von dem Rest platziert. Man erkennt sogar, dass jetzt y_0 langsamer ist als y_1.

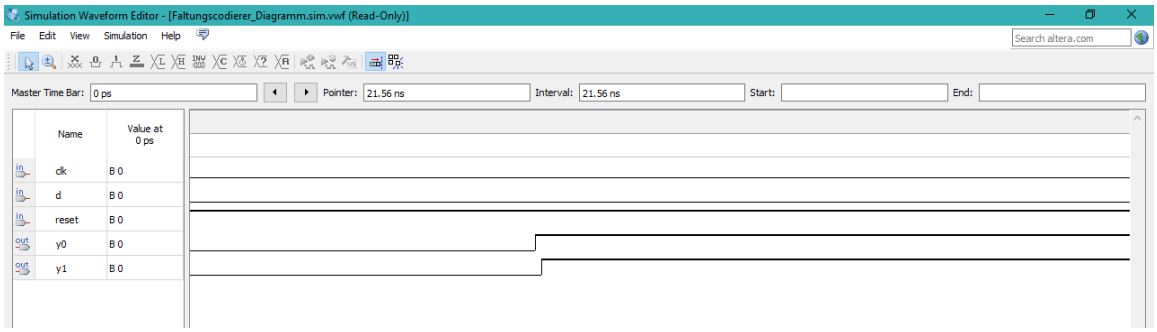

Abbildung 1-5: Gleiches Ergebnis wie in Abb. 1-4. Ein kleinerer Bereich wird gezeigt.

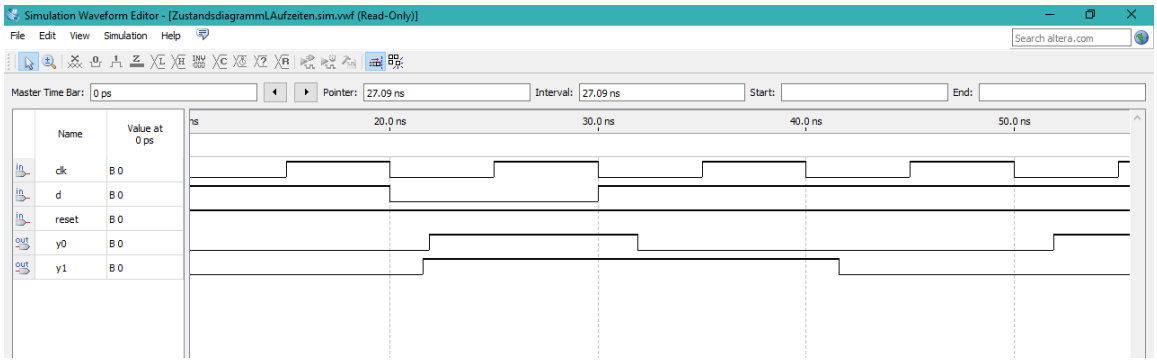

Abbildung 1-6: Ergebnis nachdem die Logik für y_0 weit weg von dem Rest platziert wurde

1.3 Encoder mit AHDL

Hier würde auch ein Zustandsautomat entworfen, aber dieses Mal mit AHDL. Das Ergebnis der Timing Simulation kann man in Abbildung 1-7 sehen. In diesem Fall nehmen beide Ausgänge ihre Werte gleichzeitig an (Abb. 1-8), da jeder Ausgang jeweils von einem D-Flip-Flop synchronisiert wird. Später wird es klar, dass trotzdem kurze Fehler stattfinden.

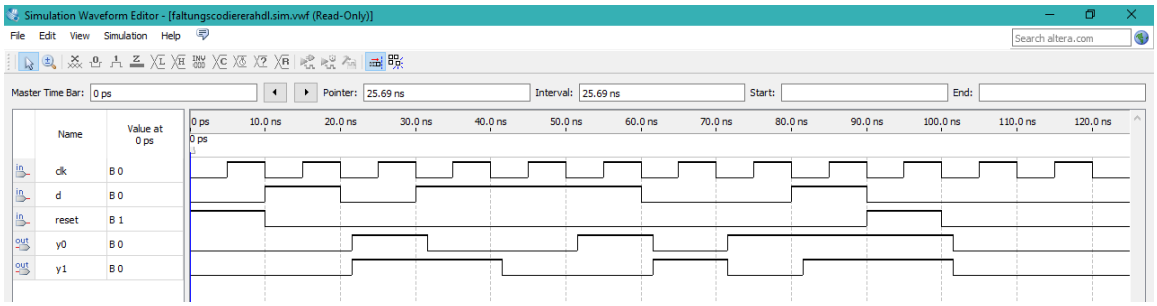

Abbildung 1-7: Ergebnis der Timing Simulation mit AHDL

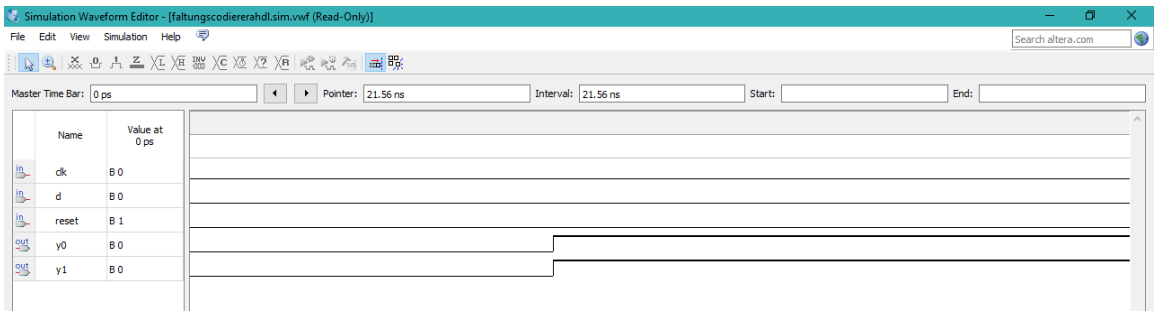

Abbildung 1-8: Kleinerer Bereich des Ergebnisses mit AHDL

1.4 Vergleich der Entwürfe

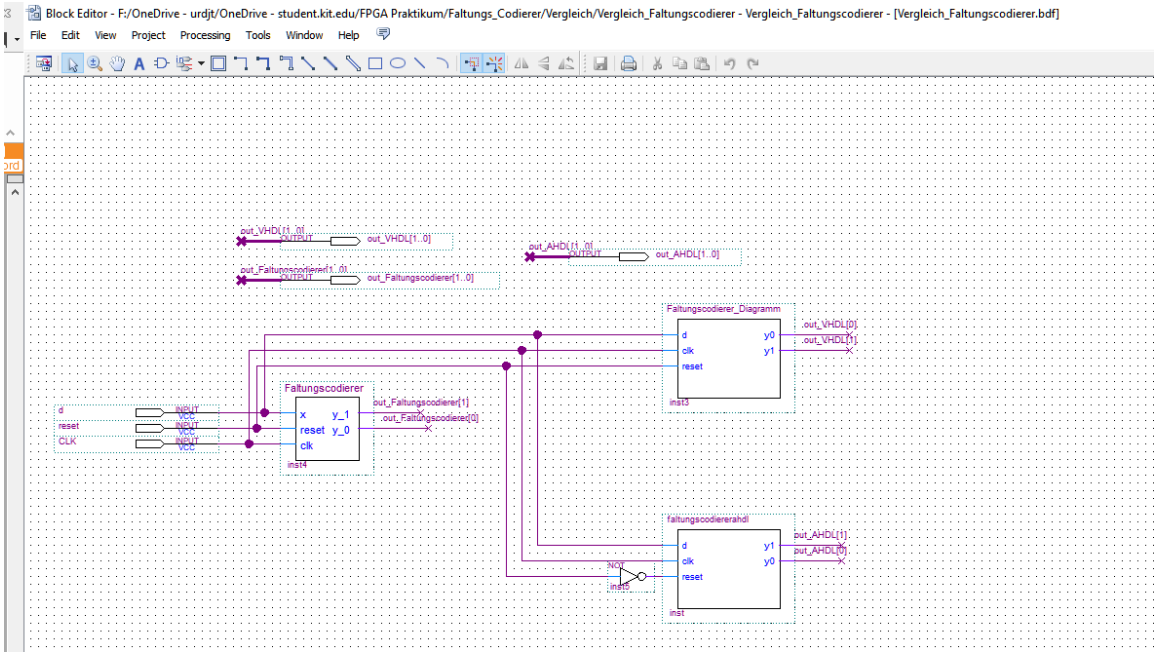

Abbildung 1-9: Blockschaltung mit den drei Alternativen

Alle drei Faltungscodierer sind auf eine einzige Schematic (Abb. 1-9) hinzugefügt und mit der gleichen Bitfolge wie immer stimuliert.

Abbildung 1-0-10: Ergebnis der Timing Simulation für aller 3 Entwürfe

Hier wird es wieder klar, dass der Faltungscodierer als Blockdiagramm (Signal out_faltungscodierer) die größte Anzahl an Fehlern erzeugt. Man erkennt auch, dass die Designs mit VHDL und AHDL erzeugen Fehler nur, wenn beide Bits am Ausgang sich ändern, während der als Blockschaltung entworfene Codierer praktisch bei jeder Änderung ein Fehler erzeugt. Das Design mit VHDL erzeugt weniger Fehler, trotzdem dauern diese Fehler relativ lange, im Vergleich mit dem Entwurf mit AHDL.

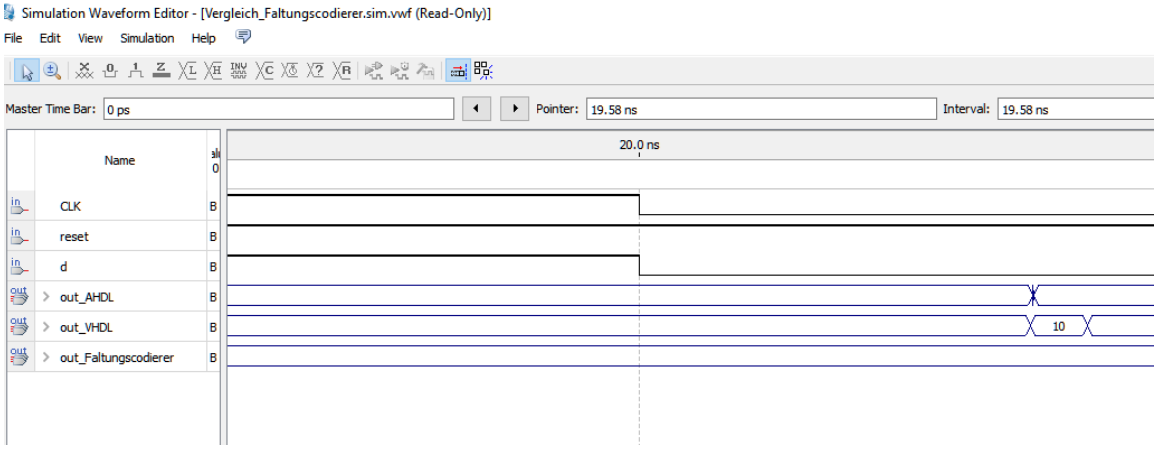

Abbildung 1-11: Vergleich der Dauer von Fehlern

2 Projekt Lauflicht

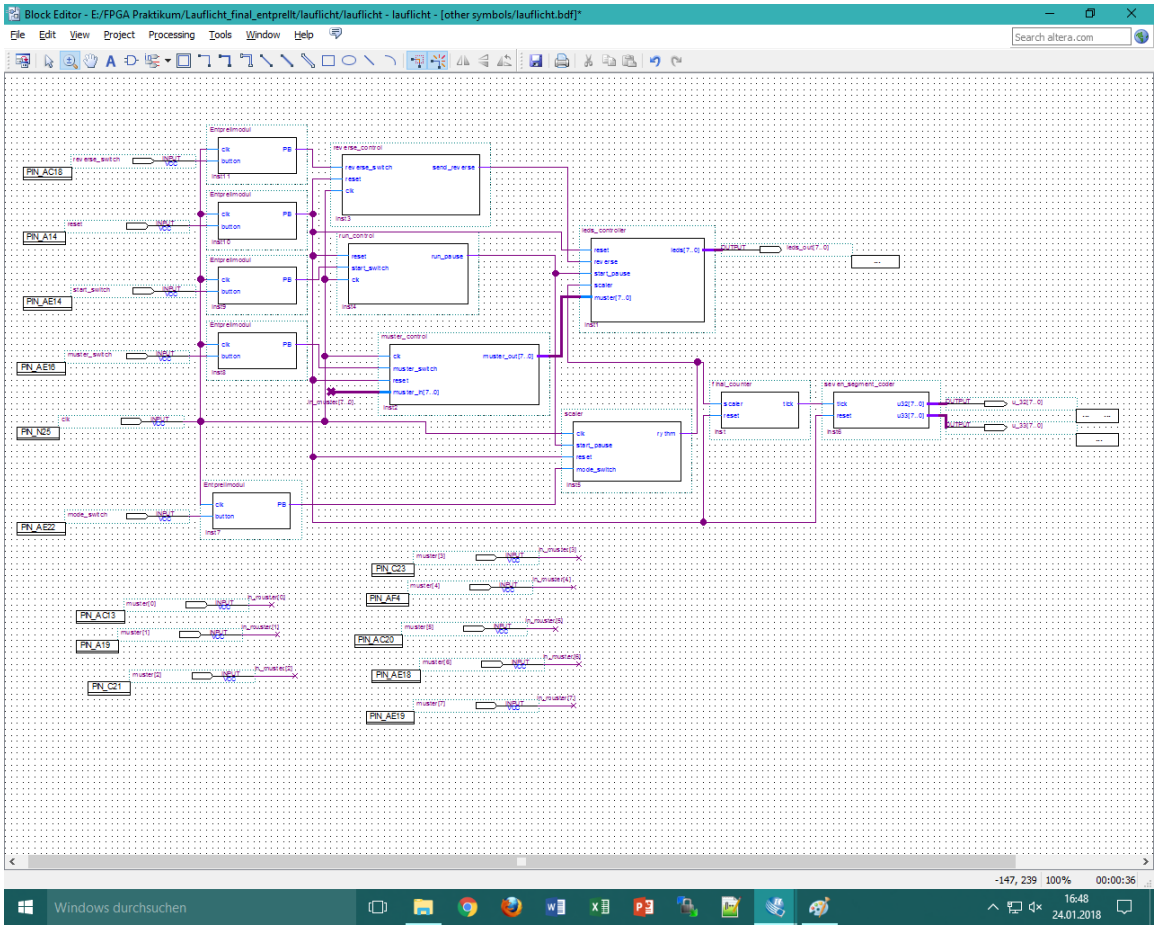

Abbildung 2-1: Blockschaltbild der Schaltung für das Lauflicht

Die gesamte Schaltung ist als die Zusammenschaltung von Zustandsautomaten, Frequenzteilern, Siebensegment Coder und die gegebene Entprellmodule entwickelt worden. Alle Module sind mit VHDL geschrieben worden und einzeln auf ihre korrekte Funktion untersucht. Nachdem alle Probleme korrigiert wurden, sind alle Die Module in der Enddatei hinzugefügt worden und diese wurde ohne die Entprellmodule getestet. Zuletzt sind die Entprellmodule hinzugefügt worden und das Board programmiert. Eine Beschreibung jedes Modul folgt:

- **reverse_control:** Es ist ein Zustandsautomat, der die Richtung des Lauflichtes bestimmt. Wenn der REVERSE_SWITCH Knopf gedrückt wird wechselt der Automat zwischen seine zwei Zustände (links → 0 am Ausgang, rechts → 1 am Ausgang). Dies wird dann von dem leds_controller Modul gelesen.
- **run_control:** Auch ein Zustandsautomat, der sein Zustand wechselt, wenn der START_SWITCH Knopf gedrückt wird (Pause → 0 am Ausgang, Laufen → 1 am Ausgang).

- **muster_control:** Dieser Automat liefert ein 8 Bit langer Signal mit dem Anfangszustand der LEDs an das leds_controller Modul. Default ist 11111110_b (nur ein LED leuchtet), aber dies ändert sich, wenn der MUSTER_SWITCH gedrückt wird. Wenn das Passiert, dann wechselt der Automat sein Zustand und liest die Werte die von dem DIP-Schalter geliefert wird und leitet diese Werte weiter an leds_controller.
- **scaler:** Das Scaler ist ein Frequenzteiler. Es besteht aus einer Mischung aus Zustandsautomat und Zähler. Der Zähler zählt hoch bis einer bestimmten Zahl erreicht wird und dann fängt wieder von vorne an. Dieser Zahl wird durch den aktuellen Zustand bestimmt, der Zustand kann mit dem MODE_SWIITCH Schalter verändert werden. Für den Wert am Ausgang sind zwei Werte des Zählers wichtig 0 und die Hälfte der maximalen Zahl in dem aktuellen Zustand. Wenn der Zähler bei null ist, wechselt der Ausgang auf 0 und, wenn der Zähler bei der Hälfte ist, wechselt der Ausgang auf 1.
- **final_counter:** Zählt von 0 bis 7, einmal pro Takt des Signals aus dem Scaler. Jedes Mal , dass Null vorliegt, wechselt der Ausgang auf 1. Wenn 5 vorliegt, dann wechselt der Ausgang auf 0. Am Ende ist dieses Modul ein zweiter Frequenzteiler. Das Ausgangssignal wird dem seven_segment_coder Modul gegeben.
- **seven_segment_coder:** Dieses Modul zählt von 0 bis 99 einmal pro Takt des Signals aus final_counter, d.h. jedes Mal, dass das Licht ein Durchlauf macht. Der Wert des Zählers wird dann in ganzen Vielfachen von 10 zerlegt und dieser Wert wird dann codiert und durch den Ausgang u_32 an eine Siebensegmentanzeige weitergeleitet. Das was übriggeblieben ist, wird auch codiert und durch den Ausgang u_33 an eine andere Anzeige weitergeleitet.
- **leds_controller:** Dieses Modul ist ein Zustandsautomat. Jeder Zustand entspricht eine Stelle auf der Reihe von LEDs, dies wird durch den Ausgang an die LEDs weitergeleitet. Wenn der Default Muster 11111110 vorliegt, Läuft der Automat hin und her. Wenn ein Muster aus dem DIP-Schalter vorliegt, dann läuft er in eine einzelne Richtung in einer Loop. In diesem zweiten Modus, kann die Richtung verändert werden (Signal aus reverse_control). Außerdem, egal welches Muster vorliegt, ist es möglich, mit Hilfe der Signale aus run_control und scaler, das Lauflicht zu pausieren und die Geschwindigkeit der Bewegung zu verändern.

3 Projekt digitale Filter

Für die digitalen Filter war es nötig im Festkommadarstellung und mit dem Zweierkomplement zu arbeiten, dazu noch ein Paar Anmerkungen.

Dezimal	Binär: Zweierkomplement + Festkommadarstelung	Entspricht
0,25	0,01	1_b
0,5	0,10	2_b
-0,25	1,11	7_b
-0,5	1,10	6_b

Tabelle 1: Zweier Kompliment Darstellung von den vorkommenden Dezimalzahlen

Alle Filter bekommen die Werte zur Bearbeitung aus einem ADC. Diese ADC liefert ein Wert, der aus 14 Bit besteht. Durch die Multiplikationen mit 3 Bit lange Faktoren entsteht am Ende ein 17 Bit langer Wert. Da die Faktoren so interpretiert werden, als 2 von der 3 Bit wären Nachkommastellen, muss man die 2 LSB aus dem Ergebnis auch als Nachkommastellen interpretieren. Diese zwei Bit können dann entfernt werden, d.h. man kann Werte unterhalb 4_b für die Überprüfung der Impulsantwort nicht benutzen. Der DAC am Ende braucht ein 14 Bit Wert für die Umwandlung, d.h. ein Bit muss noch entfernt werden. Dazu entfernt man das MSB. Am Ende hat man ein ganzzahliger, 14 Bit langer Wert zum Weiterleiten. Dieses Vorgehen ist bei allen Filtern anwesend.

3.1 Tiefpass

Für den Tiefpass mit globalen Summierer sind D-Flip-Flops als Verzögerungsglieder benutzt worden, wie bei einem Schieberegister. Die Multiplikationen und der Akkumulator würden mit Hilfe der Megafunction *altmult_add* gemacht (Abb. 3-2). Die gegebene Konstanten sind so wie in Tabelle 1 dargestellt eingegeben. Der Logikverbrauch kann aus Abb. 3-1 abgelesen werden, insgesamt sind 69 Elemente notwendig.

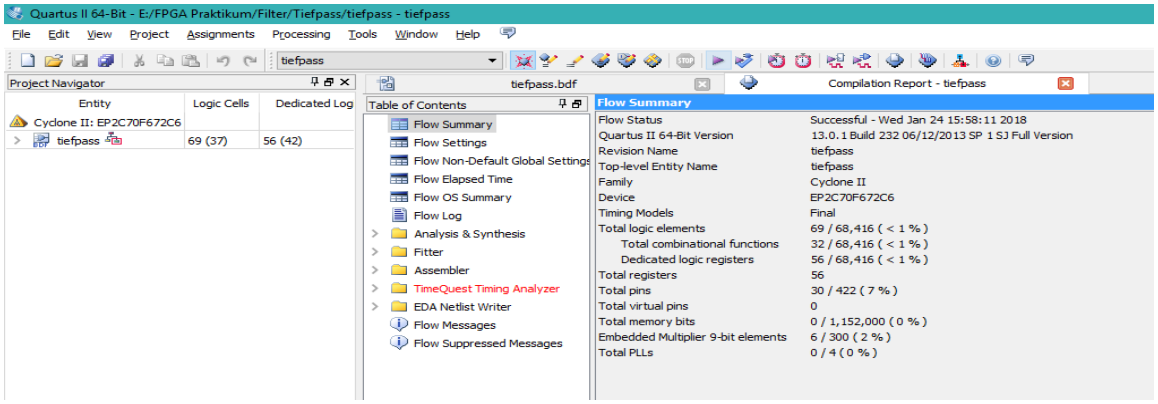

Abbildung 3-1: Logikverbrauch des Tiefpasses

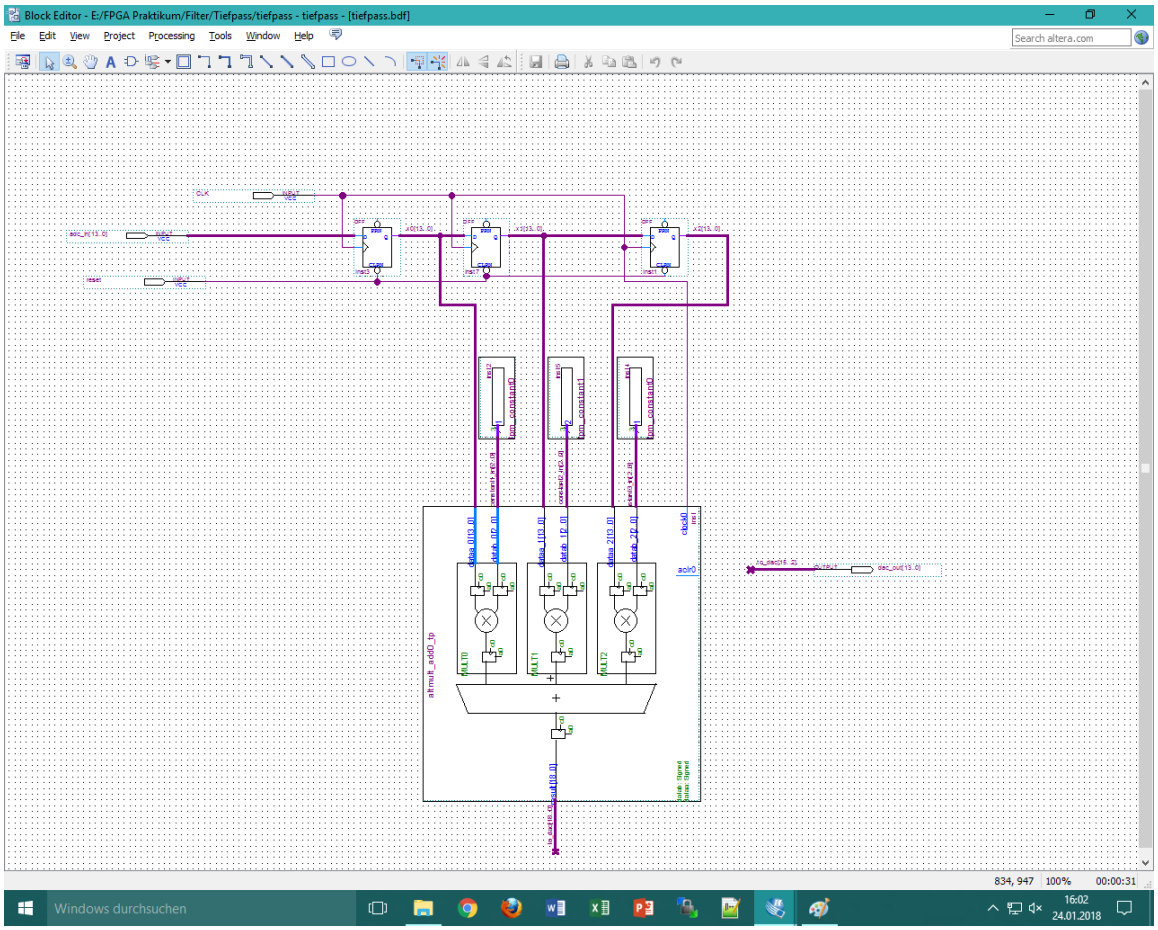

Abbildung 3-2:Digitales Tiefpassfilter mit den Koeffizienten 0.25, 0.5 und 0.25

Die maximale Arbeitsfrequenz liegt bei 199MHz (Abb. 3-3), deutlich über die Clock Frequenz des Entwicklungsboards, falls man diese als Abtastfrequenz benutzen wollte.

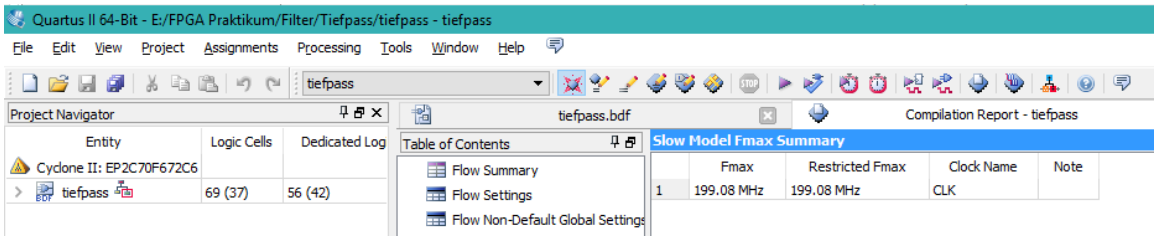

Abbildung 3-3: Maximale Arbeitsfrequenz des Tiefpasses

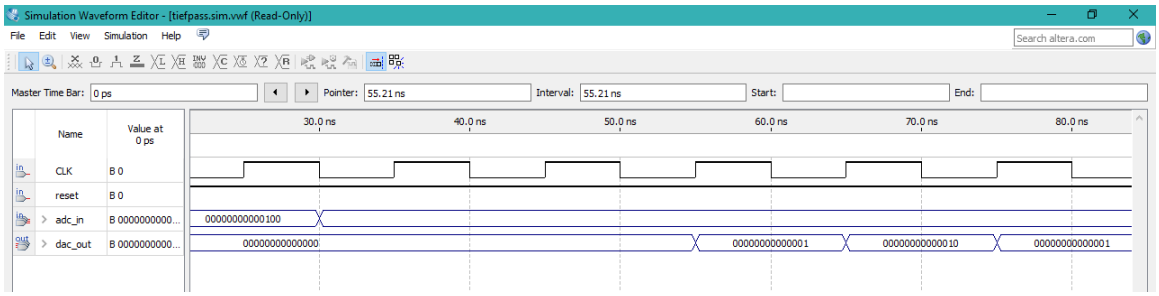

Abbildung 3-4: Impulsantwort des Tiefpasses

Wenn man die Impulsantwort dieses Filters für den Impuls 4_b berechnet, dann kommt man auf [1_b; 2_b; 1_b]. Man sieht dieses Resultat entspricht das Ergebnis der Simulation. Außerdem wurde die Sprungantwort Überprüft. Dazu ist als Sprung den Wert 1_b benutzt worden. In der Berechnung hat das System 3 Zyklen, um den Eingang stabil zu folgen, aber in der Simulation hat es 2 Zyklen mehr gedauert, wahrscheinlich wegen der Megafunktion.

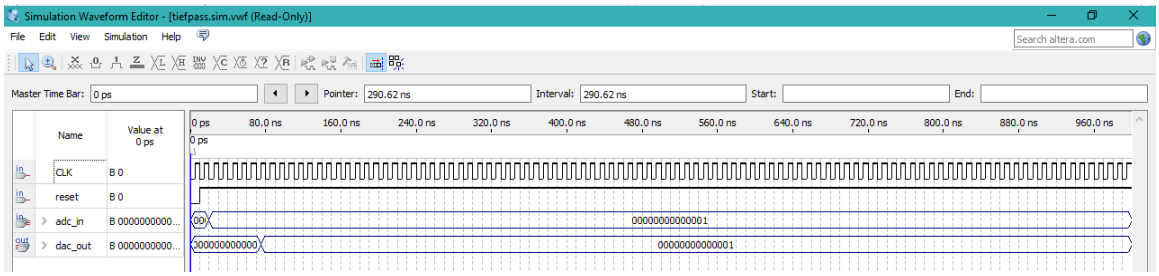

Abbildung 3-5: Sprungantwort des Tiefpasses

3.2 Hochpass

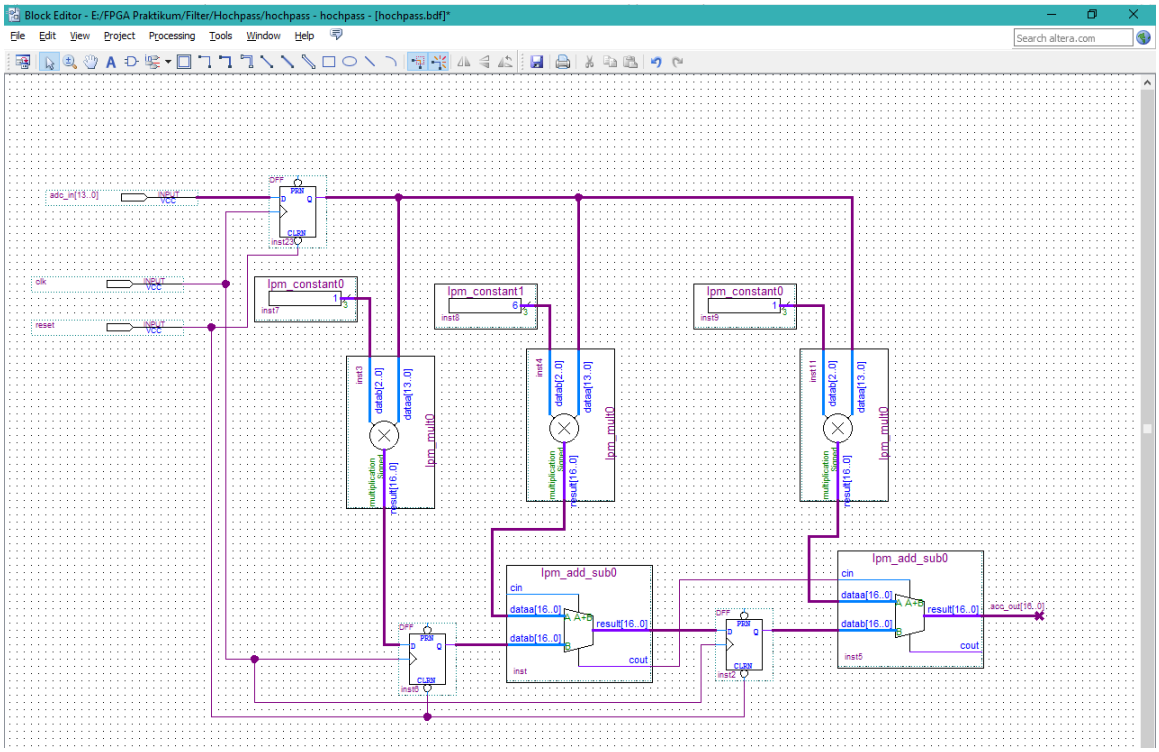

Abbildung 3-6: Digitales Hochpassfilter mit den Koeffizienten 0.25, -0.5 und 0.25

Der Hochpass mit verteilten Summierern steht im Abb. 3-6. Es ist offensichtlich, dass die Megafunctions in diesem Fall nicht die gleichen sind wie beim Tiefpass, weil die Flip-Flops sich zwischen den Multiplikatoren und den Summierern befinden.

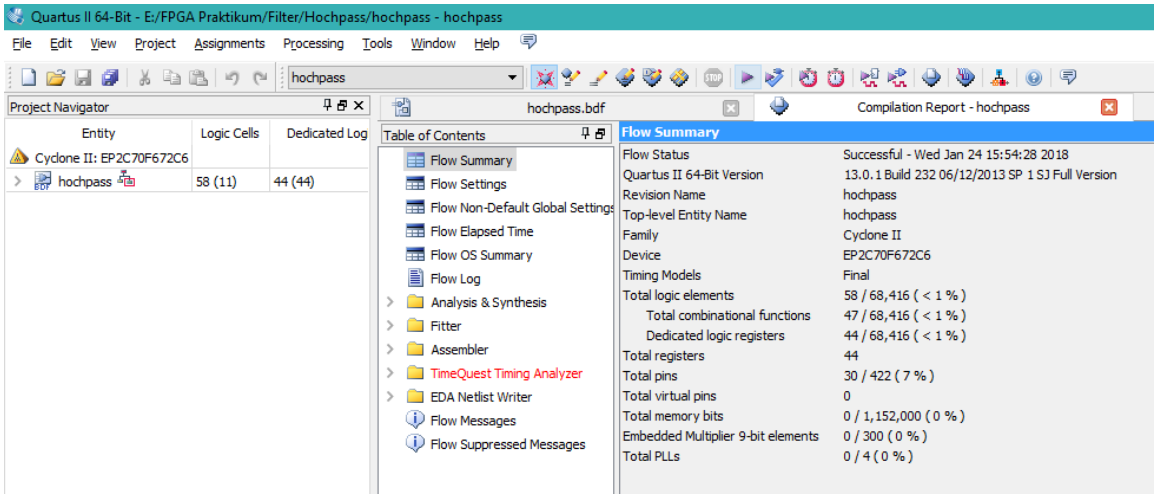

Abbildung 3-7: Logikverbrauch des Hochpasses

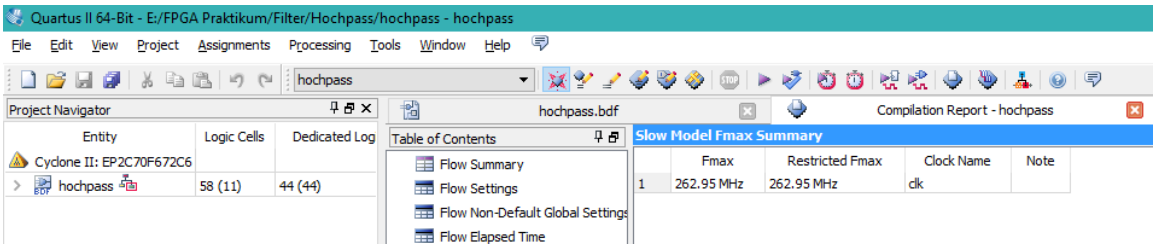

Abbildung 3-8: Maximale Arbeitsfrequenz des Hochpasses

Aus Abb. 3-7 liest man, dass der Logikverbrauch für dieses Hochpass 58 Logikelemente beträgt. Es sind also 11 Logikelemente weniger als das Tiefpass benötigt. Eigentlich sollten diese beide Werte ungefähr gleich sein, da beide Schaltungen die gleichen Elemente haben. Der Unterschied liegt hier auch an den benutzten Megafunctions. Die einfache Addierer und Multiplikatoren im Hochpass brauchen weniger Logikelemente als der große, getaktete Megafunction im Tiefpass.

Die maximale Arbeitsfrequenz liegt in diesem Fall bei ~263MHz (Abb. 3-8). Man sieht, dass es schneller als das Tiefpass ist. Dies kann liegen an die kleinere Anzahl an Komponenten im Hochpass, die einen Takt brauchen. Hier kann wieder die große Megafunction schuld sein. Insgesamt scheint das Hochpass besser zu funktionieren.

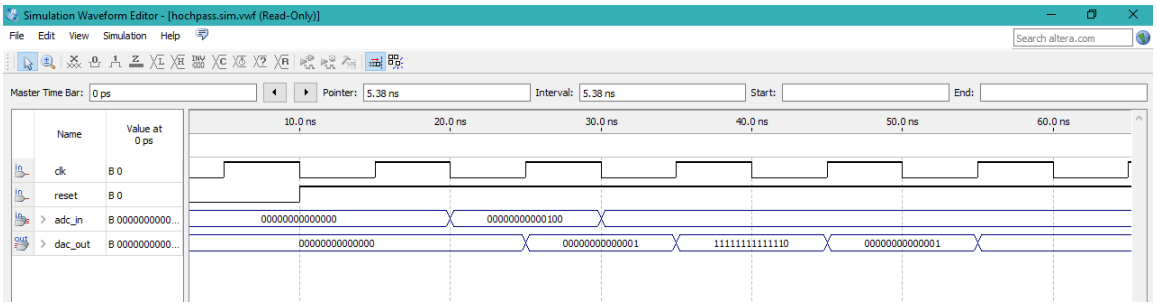

Abbildung 3-9: Impulsantwort des Hochpasses für den Impuls 4_b

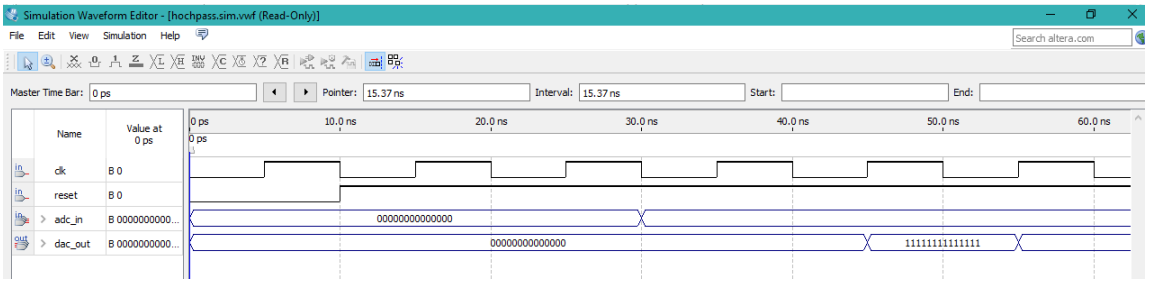

Abbildung 3-10: Sprungantwort des Hochpasses. Nach den Wert mit 14 „1“ kommen wieder nur Nullen vor.

3.3 Bandpass

Das Bandpass wurde mit VHDL entworfen. Die Kompilierte Schaltung braucht nur 47 Logikelemente und kann bei maximal 420MHz arbeiten, wenn die Einschränkungen der I/O Wechselrate beachtet werden. Sonst könnte die Schaltung bis ungefähr 905MHz

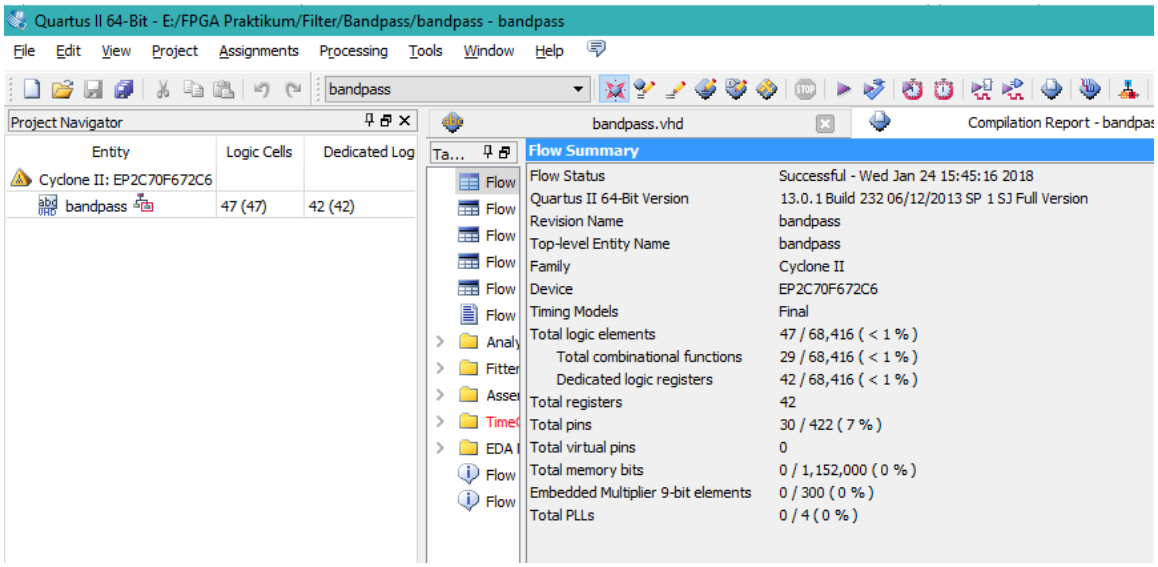

Abbildung 3-11: Logikverbrauch des Bandpasses

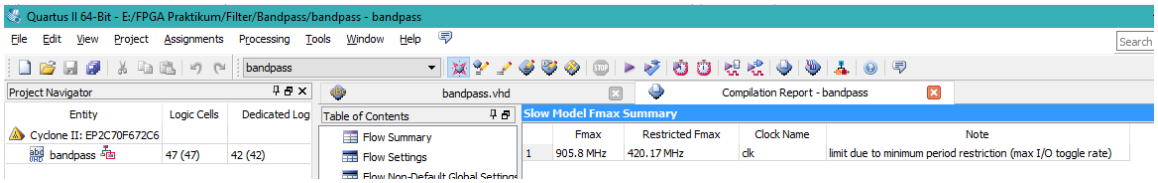

Abbildung 3-12: Maximale Arbeitsfrequenz des Bandpasses

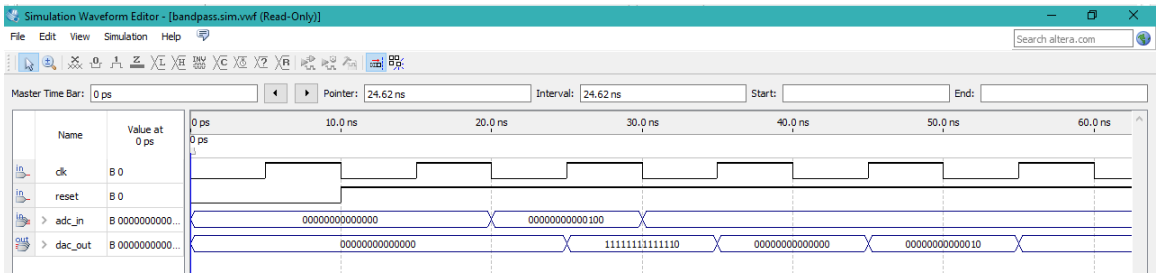

Abbildung 3-13: Impulsantwort des Bandpasses für den Impuls 4_b

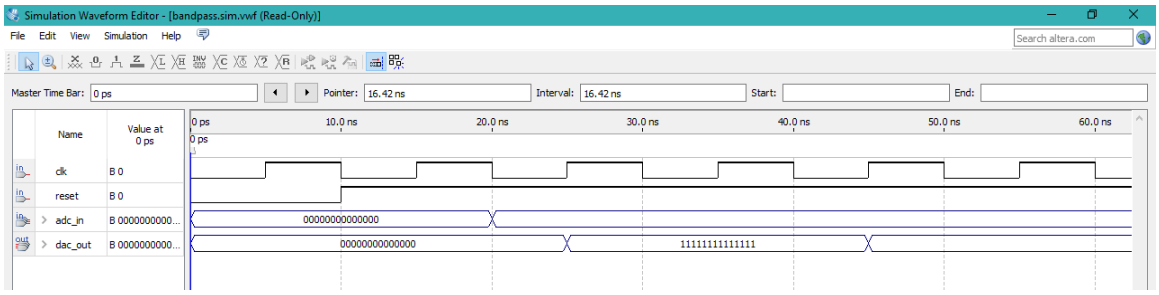

Abbildung 3-14: Sprungantwort des Bandpasses

3.4 Bandsperre

Unten stehen die Ergebnisse für die Bandsperre. Der Logikverbrauch beträgt 43 Logikelemente, 4 weniger als das Bandpass. Die maximale Arbeitsfrequenz würde 919MHz betragen, wäre es nicht begrenzt durch die I/O. Die Bandsperre kann also fast 15MHz schneller arbeiten als das Bandpass. Es scheint vorteilhafter eine Bandsperre zu benutzen.

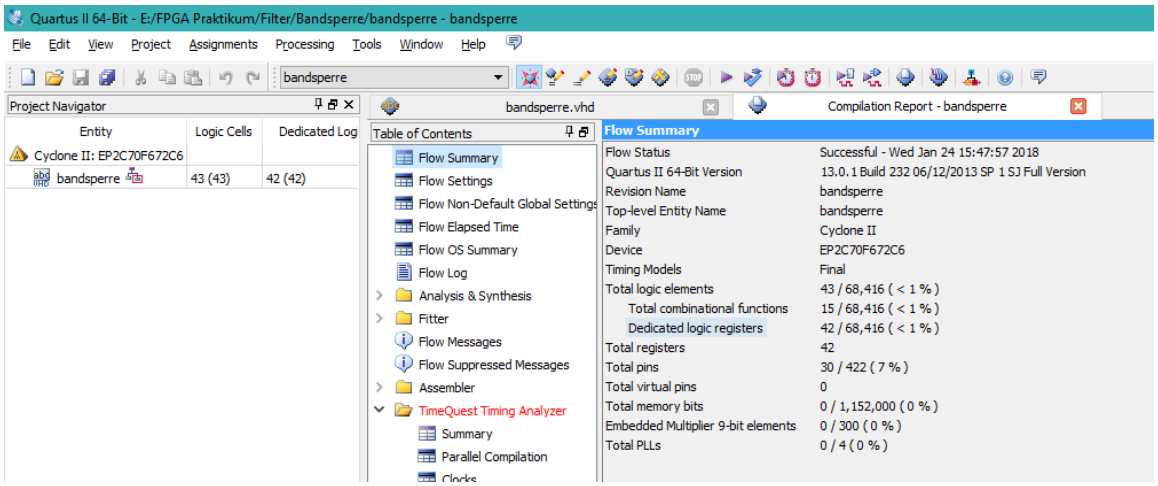

Abbildung 3-15: Logikverbrauch der Bandsperre

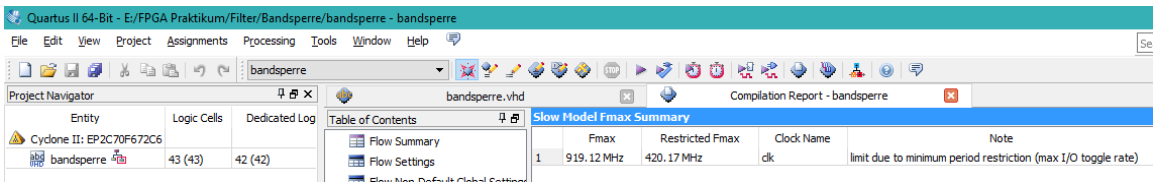

Abbildung 3-16: Maximale Arbeitsfrequenz der Bandsperre

Impuls- und Sprungantwort stehen unten zum späteren Vergleich.

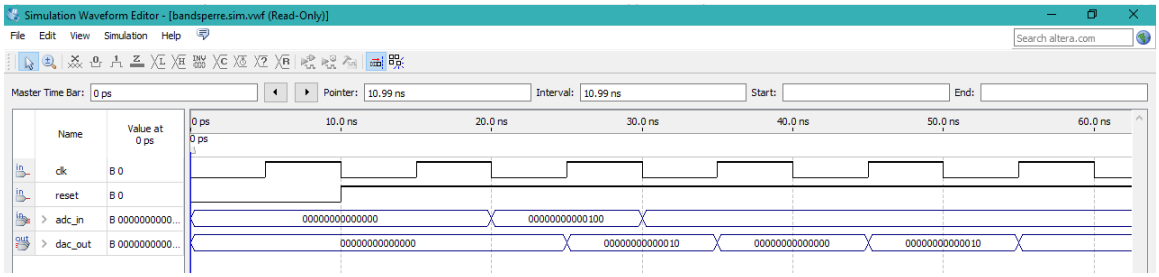

Abbildung 3-17: Impulsantwort der Bandsperre für den Impuls 4_b

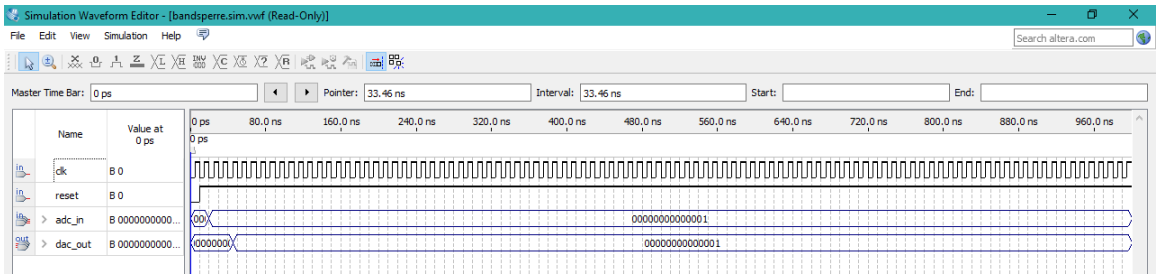

Abbildung 3-18: Sprungantwort der Bandsperre

3.5 Projekt mit allen Filtern

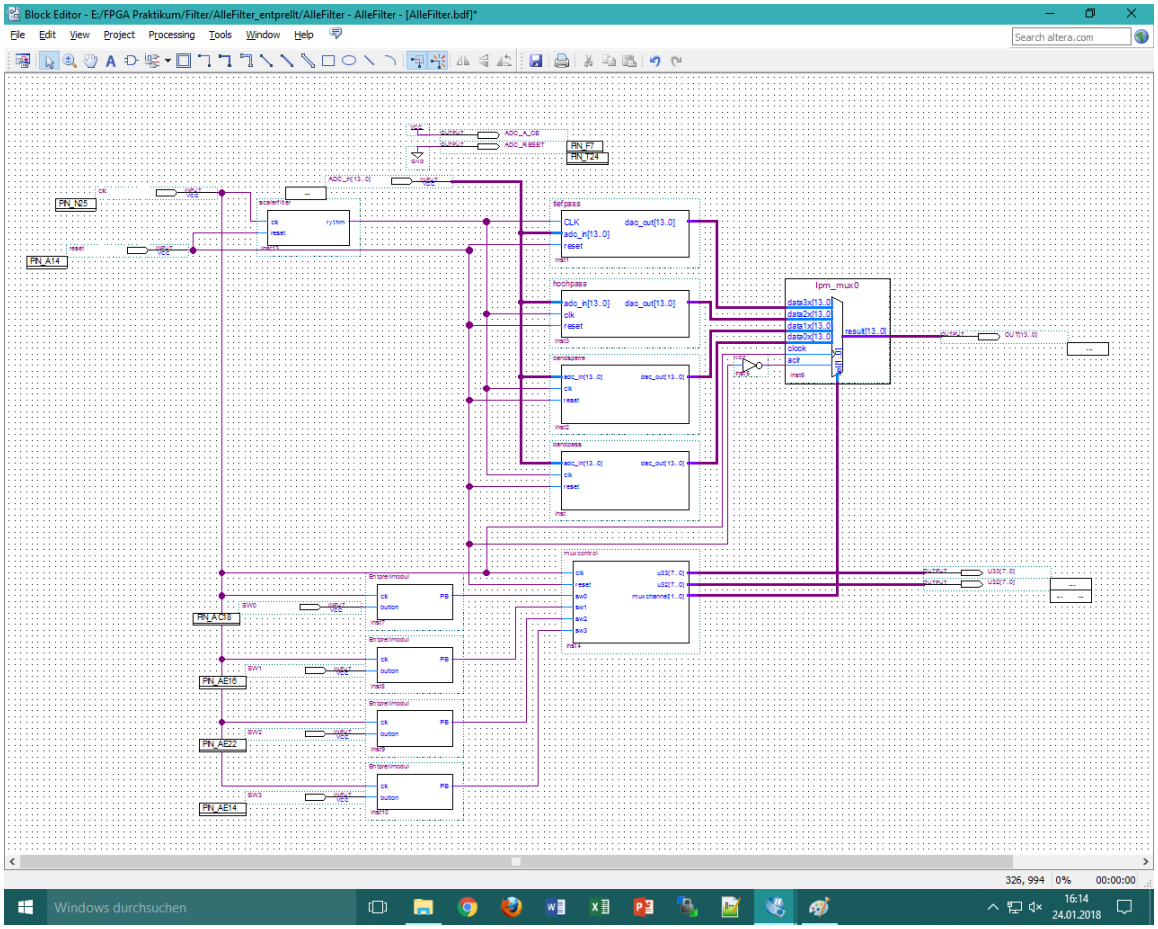

Abbildung 3-19: Schaltung mit allen Filtern

Alle Filter wurden in eine einzelne Schaltung hinzugefügt, um es später auf dem Entwicklungsboard zu programmieren. Außerdem sind vier weitere Module nötig gewesen. Zuerst steht ein Frequenzteiler oben links. Dieser erzeugt aus dem 100MHz Clock Signal ein neues Signal mit der Frequenz 5 MHz. Dieses langsamere Signal wird als Abtastfrequenz der Filtern benutzt. Dazu wurde auch noch ein Multiplexer hinzugefügt, um den gewünschtem Ausgang aus zu wählen. Diese Multiplexer kann nicht von alleine arbeiten, deswegen braucht man das muxcontrol Modul. Dieses Modul besteht aus einem Zustandsautomat und aus einem Siebensegmentcoder. Der Ausgang muxchannel ist 2 Bit weit und durch den Wert von diesem Ausgang, wird ein Filter gewählt. Dieser Ausgang entspricht den aktuellen Zustand, in dem der Automat sich befindet (Tabelle 2). Je nach Zustand werden verschiedene Werte über u32 und u33 weitergeleitet, so dass man weiß welche Filter gewählt wurde (bp: Bandpass, bs: Bandsperre, hp: Hochpass, tp: Tiefpass).

Switch	Zustand	Mux Ausgang – Anzeige Ausgang
0	00	00 - bp
1	01	01 - bs
2	10	10 – hp
3	11	11 - tp

Tabelle 2: Muxcontrol Modul - Funktion

Um die Funktionalität zu überprüfen, wurde der Impuls 4_b den Filtern gegeben und der Antwort mit den vorherigen Ergebnisse verglichen. Dazu stehen die entrsprechende Bereiche unten vergrößert.

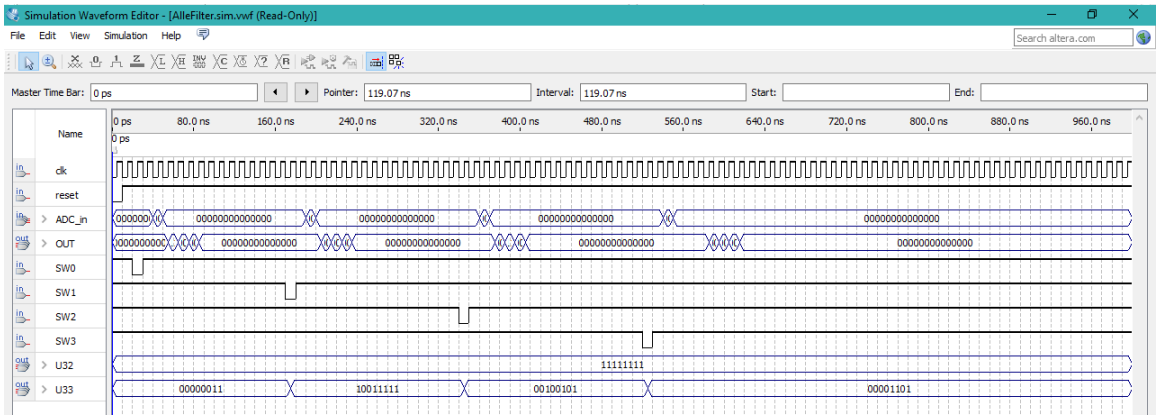

Abbildung 3-20: Simulationsvorgehensweise

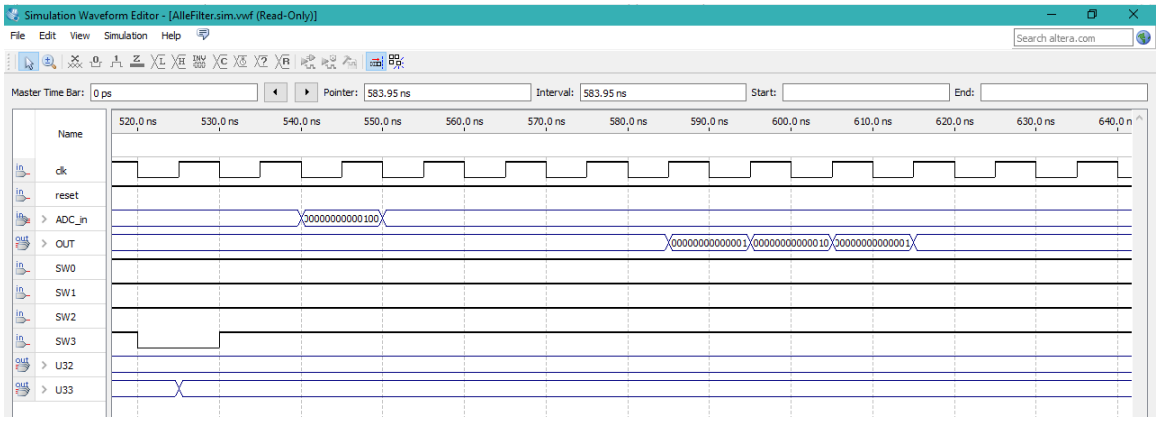

Abbildung 3-21: Simulationsabschnitt - Tiefpass

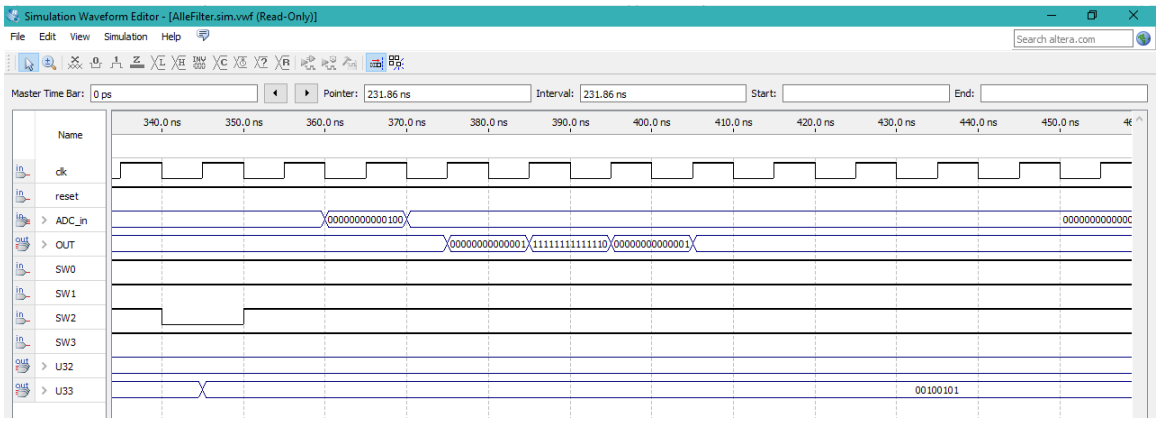

Abbildung 3-22: Simulationsabschnitt - Hochpass

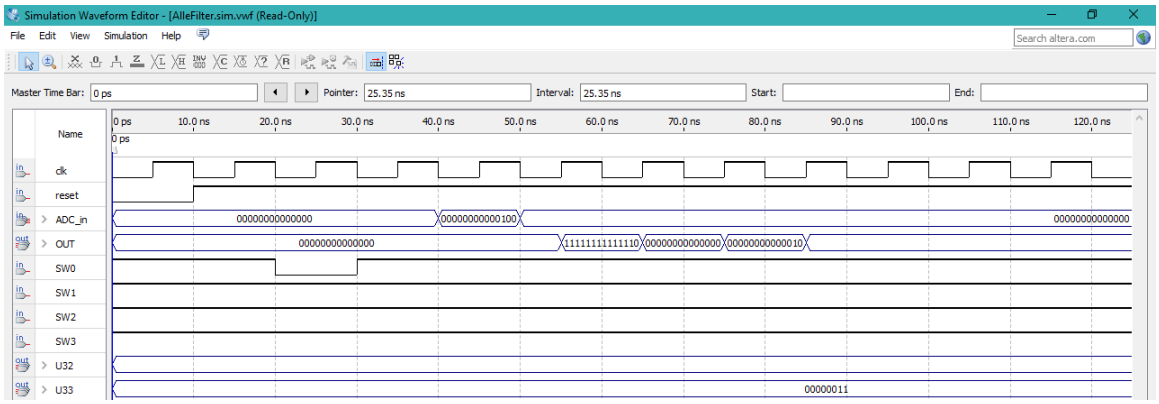

Abbildung 3-23: Simulationsabschnitt - Bandpass

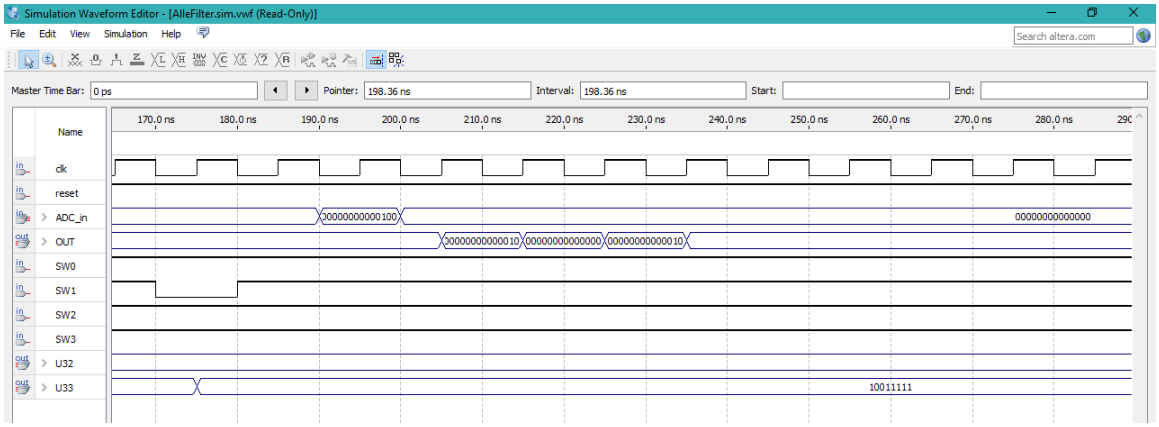

Abbildung 3-24: Simulationsabschnitt - Bandsperre

Die Ergebnisse sind genau gleich wie die, die bevor erzeugt wurden.

Die gesamte Schaltung braucht 160 Logikelemente (Abb. 3-25), wenn nicht entprellt. Nachdem die vier Entprellmodulen für die Taster zugefügt wurden, ist die Anzahl an Logikelementen stark gestiegen. In dem entprellten Fall sind 1193 Logikelementen nötig (Abb. 3-26).

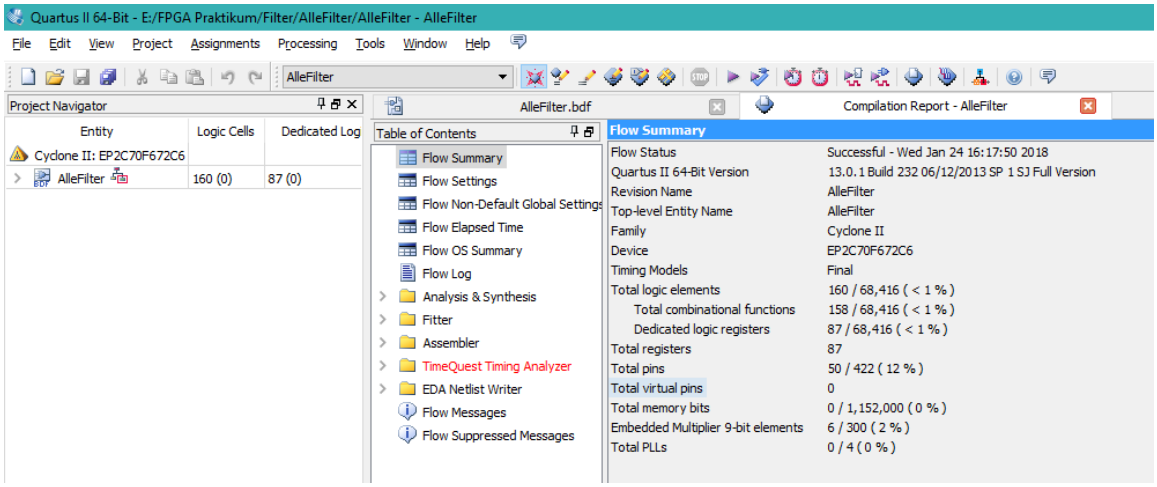

Abbildung 3-25: Logikverbrauch - Nicht entprellte Schaltung

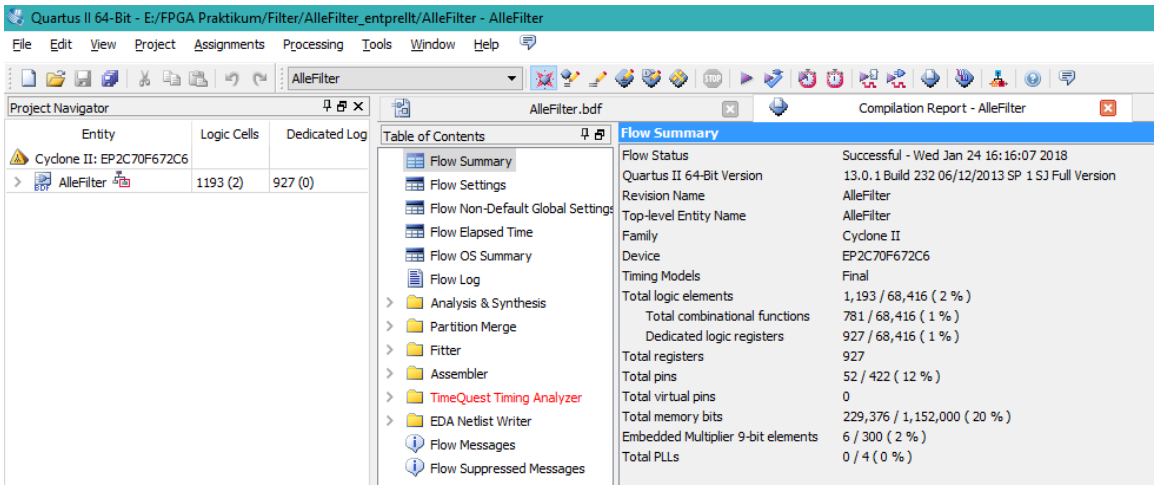

Abbildung 3-26: Logikverbrauch - Entprellte Schaltung

Die maximale Arbeitsfrequenz konnte nur für die nicht entprellte Schaltung bestimmte werden. Die maximale Frequenz beträgt 162.68MHz. Dieser Wert ist sogar kleiner als die maximale Frequenz des Tiefpasses. Es ist sehr wahrscheinlich, dass das getaktete Multiplexer die Frequenz beschränkt, weil alle andere Komponenten eigentlich mit viel höhere Frequenzen arbeiten können. Für die entprellte Schaltung hat Quartus II die maximale Frequenz aus unbekannte Gründen nicht gezeigt.

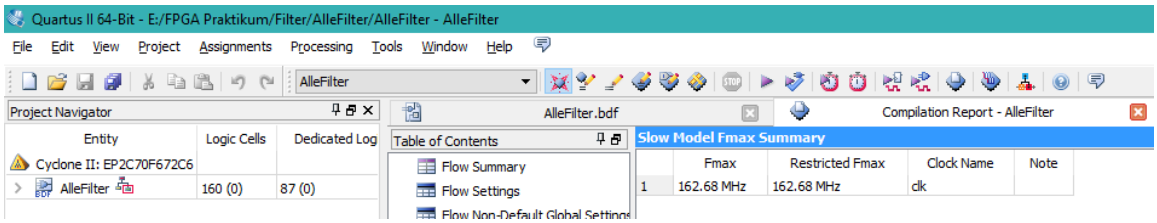

Abbildung 3-27: Maximale Arbeitsfrequenz - Nicht entprellte Schaltung

3.6 Programmierung des Cyclone II Bausteins

Bei allen Filtern wurde der Frequenzgenerator angeschlossen und von ganz tiefen Frequenzen bis 2MHz langsam umgestellt. Der Verlauf bei einige Frequenzen, die den erwarteten Verlauf des entsprechenden Filters sind in den Abbildungen gezeigt. Man erkennt, dass alle Filter den korrekten Verhalten zeigen.

In einige Bilder sieht man eine Art Oszillation, die sich mit dem Ausgangswerte „überlappt“. Dies kommt zu stande, wegen der gewählten Abtastfrequenz. Da die 5MHz nicht mit ein ganzes Vielfaches der Signalfrequenz übereinstimmt, „verschieben sich die Abtastpunkte im nächsten Periode des Signals und das passiert immer weiter. Eventuell werden ungefähr die gleiche Werte abgetastet als die, die eine bestimmte Anzahl von Perioden bevor abgetastet wurden. Diese „Quasi-Periodizität“ erzeugt das was in z.B. Abb. 3-33 oder in Abb. 3-41 zu sehen ist.

3.6.1 Tiefpass

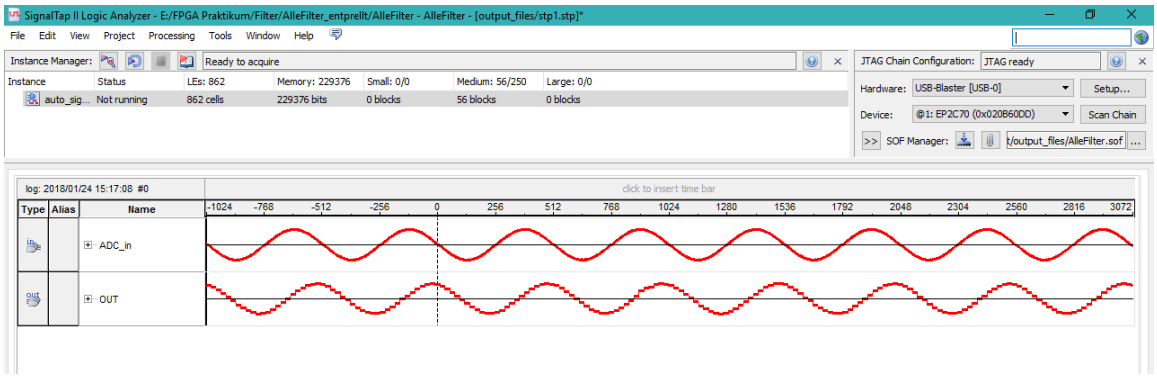

Abbildung 3-28: Tiefpass – f = 200kHz

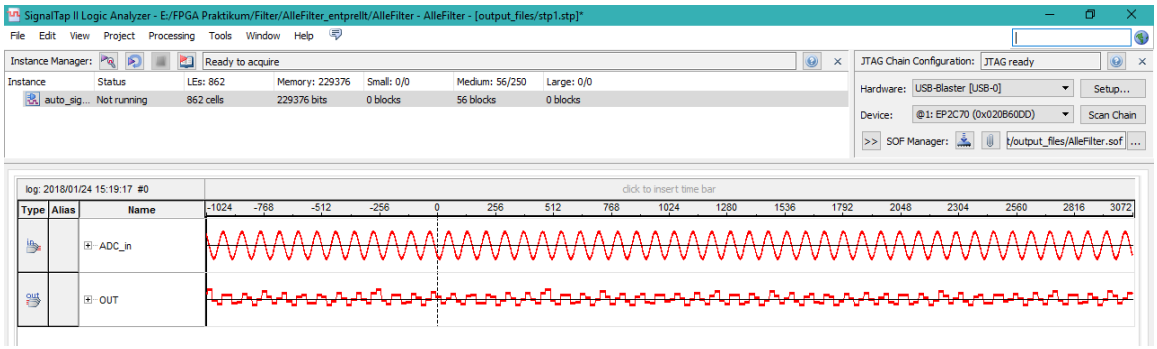

Abbildung 3-29: Tiefpass - f = 1,2MHz

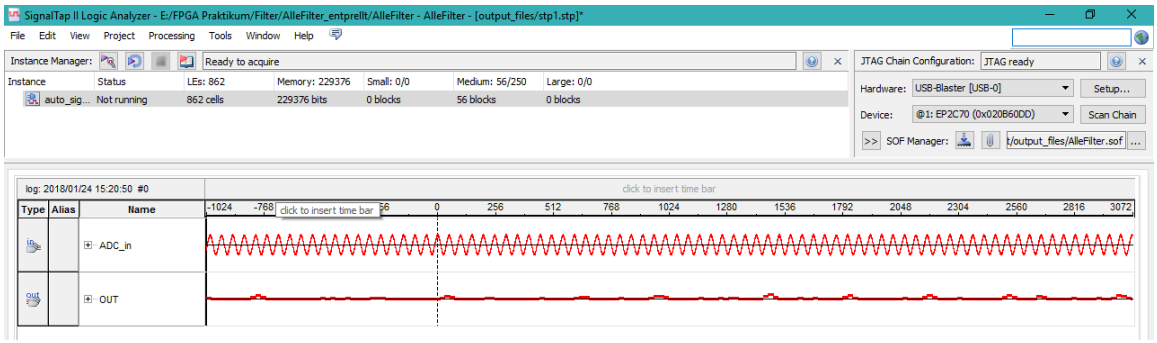

Abbildung 3-30: Tiefpass - f = 2MHz

3.6.2 Hochpass

\

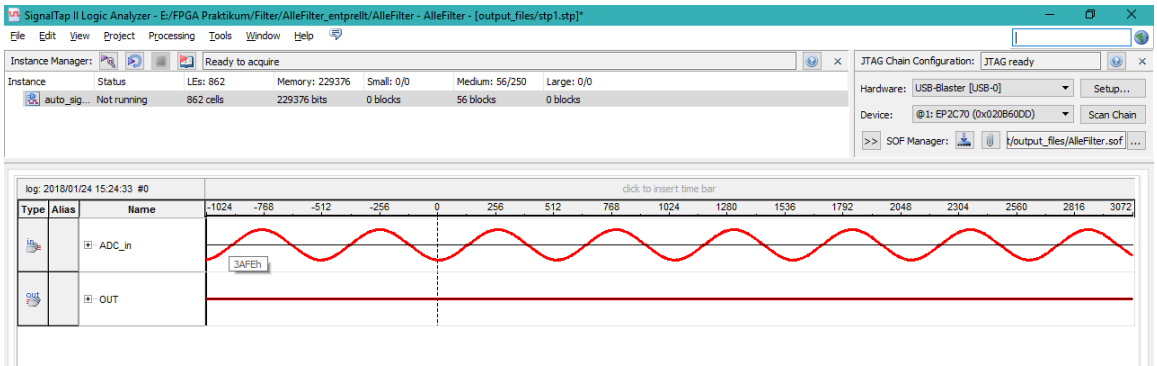

Abbildung 3-31: Hochpass - f = 200kHz

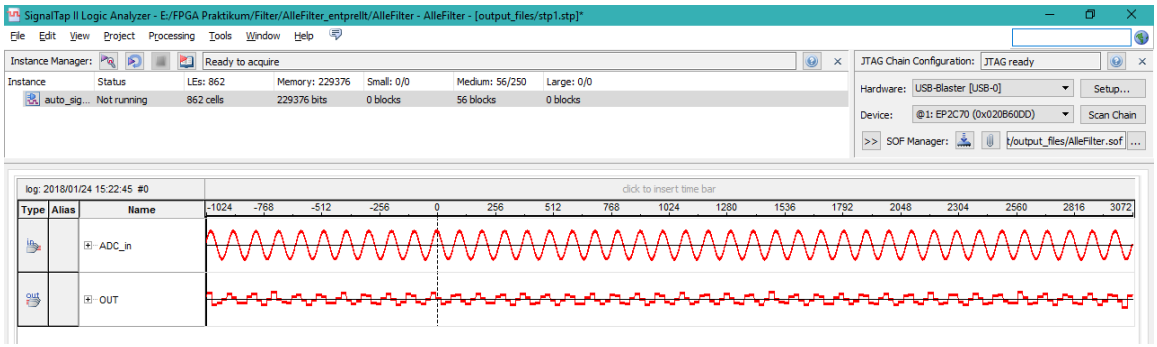

Abbildung 3-32: Hochpass - f = 1MHz

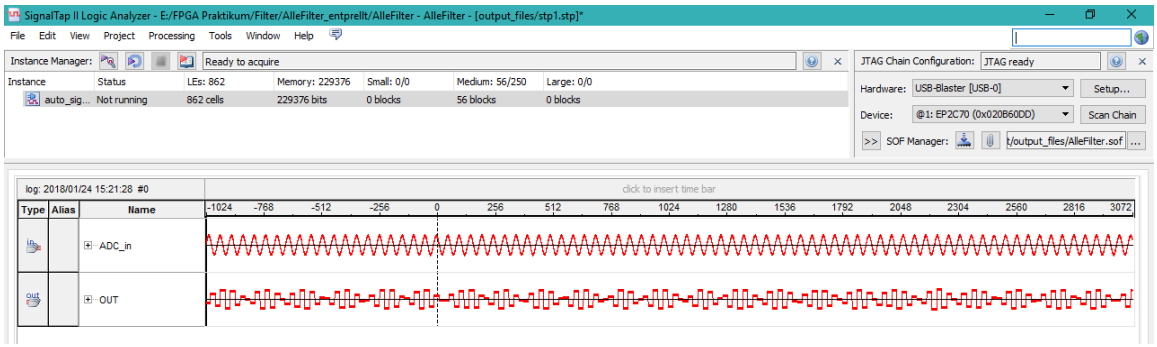

Abbildung 3-33: Hochpass - f = 2MHz

3.6.3 Bandpass

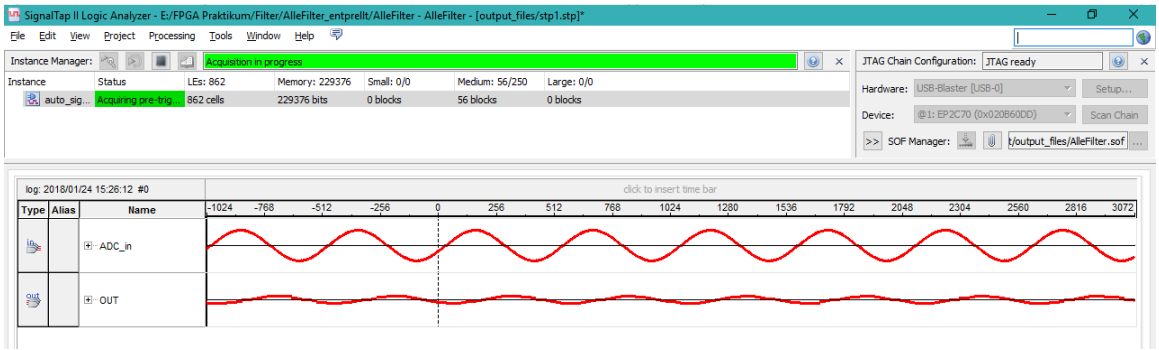

Abbildung 3-34: Bandpass - f = 200kHz

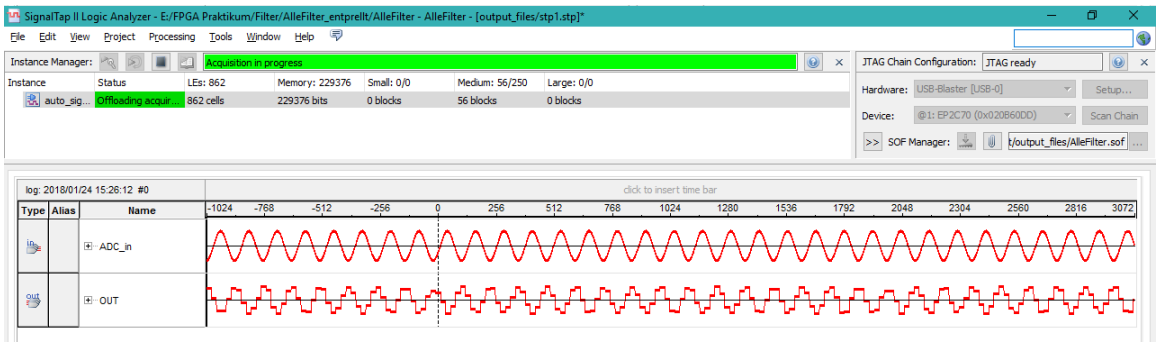

Abbildung 3-35: Bandpass - f = 800kHz

\

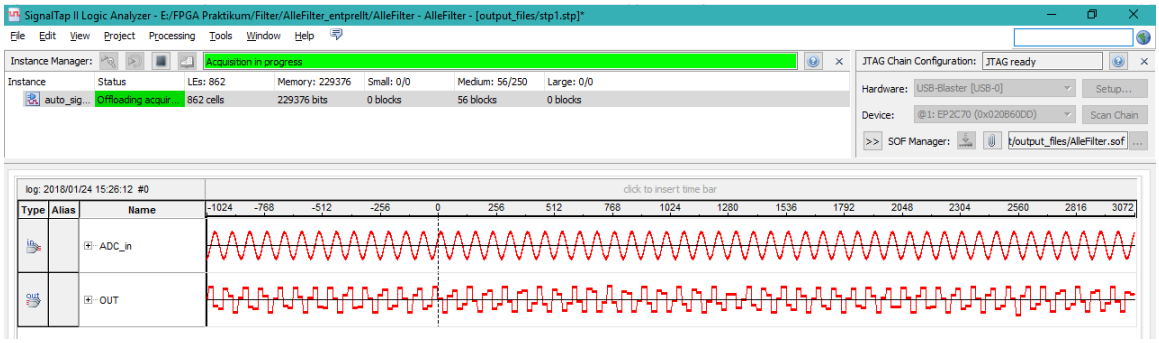

Abbildung 3-36: Bandpass - f = 1,1MHz

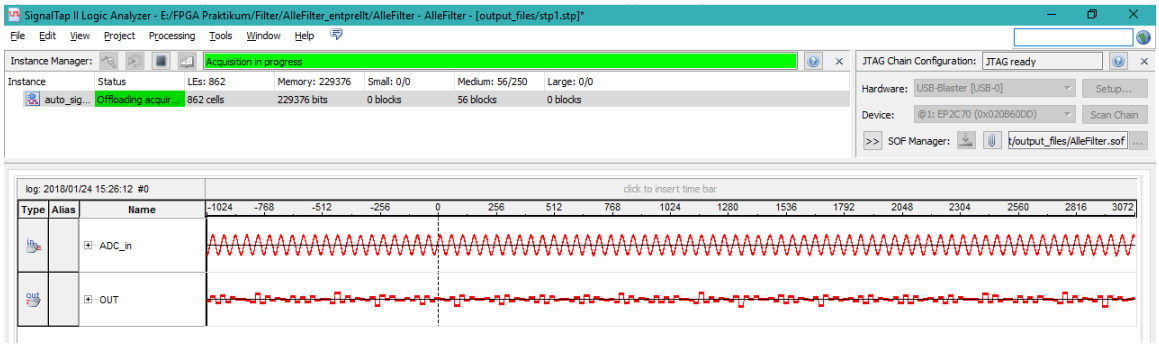

Abbildung 3-37: Bandpass - f = 2MHz

3.6.4 Bandsperre

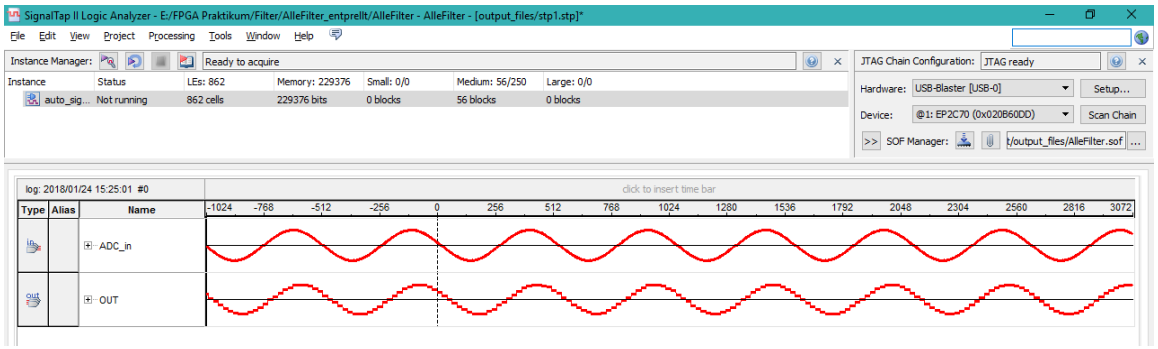

Abbildung 3-38: Bandsperre - f = 200kHz

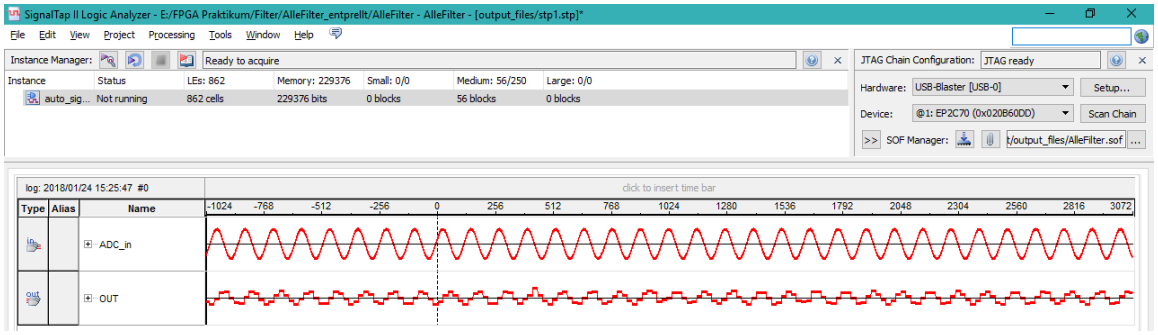

Abbildung 3-39: Bandsperre – 800kHz

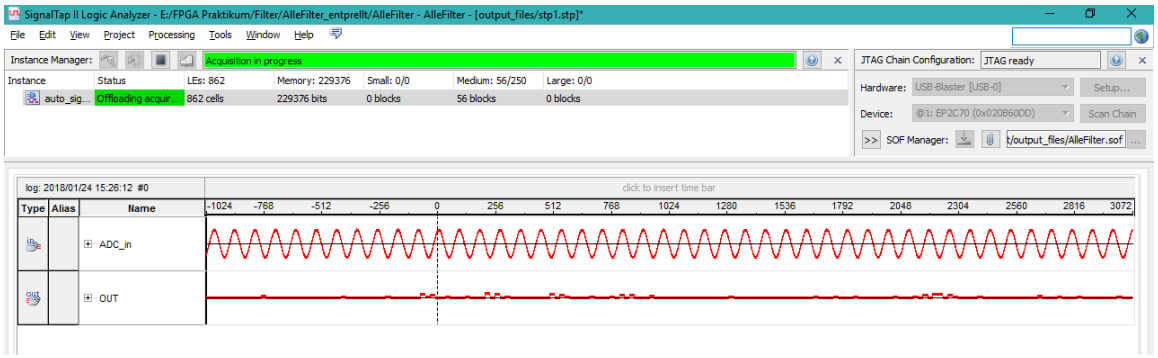

Abbildung 3-40: Bandsperre - f = 1,1MHz

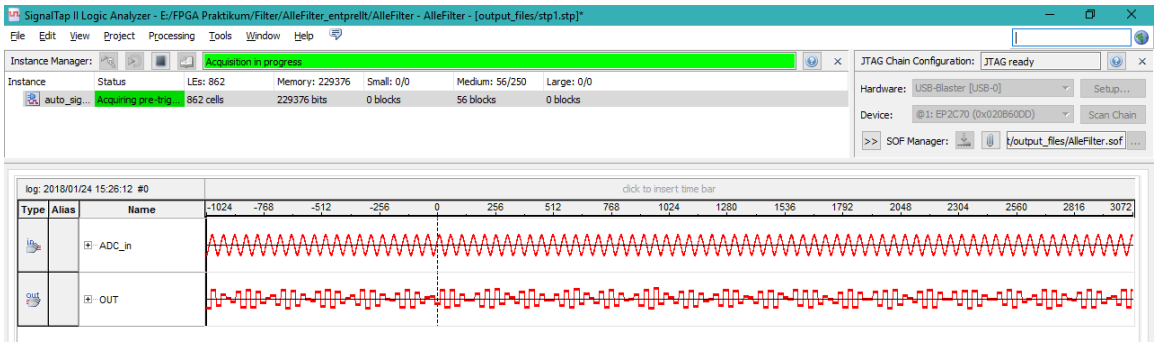

Abbildung 3-41: Bandsperre - f = 2MHz