Fachhochschule
Dortmund
University of Applied Sciences and Arts

Masterarbeit

Im Masterstudiengang FB Elektrotechnik
Energiesysteme
(Automatisierungs- und Antriebssysteme)

Digital geregelte Frequenzkorrektur eines strahlenharten Relaxations-Oszillators für eine CAN Bittiming Einheit in 65nm CMOS Technologie

"Digitally controlled frequency correction of a radiation-hard relaxation oscillator for a CAN bit timing unit in 65nm CMOS technology"

Vorgelegt von: Reda Bouroumiya
Matr.-Nr.: 7087741

an der Fachhochschule Dortmund

Betreuender Professor: **Prof. Dr.-Ing Michael Karagounis**

Zweitprüfer: **M. Eng Jeremias Kampkötter**

“No idea is really bad, unless we are uncritical.
What is really bad is to have no idea at all.”
— George Pólya

Stichworte

CAN-Protokoll, Bittiming-Logik, Synchronisationsmechanismen, strahlenharter Relaxationsoszillator, 65 nm CMOS-Technologie, Entwurfssoftware Virtuoso von Cadence, gemischte analog/digitale Schaltungen, Analog-Mixed-Signal-Simulation, Entwurf einschleifiger Regelkreise, zeitdiskreter PID-Regler, Verilog Hardware-Beschreibungssprache, Frequenzregelung, Regler-Robustheit.

Keywords

CAN protocol, bit timing logic, synchronization mechanisms, radiation-hard relaxation oscillator, 65 nm CMOS technology, Virtuoso design software from Cadence, mixed analog / digital circuits, analog mixed signal simulation, design of single-loop control loops, time-discrete PID controller, Verilog hardware Descriptive language, frequency control, controller robustness.

Kurzzusammenfassung

Diese Arbeit behandelt den Entwurf und die Implementierung einer Frequenzregelung für den analogen Relaxationsoszillator des im Kontrollsystem des ATLAS-Pixeldetektors eingesetzten CANakari-Controllers des MOPS Chips. Bestehend aus einem Pulszähler, einem PID-Regler, Phasenfehler-Register und einem Control-FSM-Modul, wird das Regelsystem mit dem digital-gesteuerten analogen Oszillator und der Bittiming-Logik verdrahtet. Diese Komponenten können miteinander kommunizieren, Daten austauschen und bilden somit einen geschlossenen Regelkreis. Der Regelalgorithmus beobachtet das eingehende Signal Rx des CAN Busses und verändert die Stellgröße bei entstehender Regelabweichung durch die Detektierung einer fallenden Flanke außerhalb des im CAN Standard definierten Synchronisationssegments, so dass die Taktfrequenz in einem Toleranzintervall stabilisiert wird. Dies gewährleistet, dass es im CAN-Netzwerk nicht zu Synchronisationsfehlern bei der Nachrichtenübertragung kommt. Da es sich um eine gemischte analog/digitale Schaltung handelt, wird das Regelkreis-Verhalten mit Hilfe einer A/MS-Simulationen beurteilt. Die Simulationen dienen einerseits zur Untersuchung wichtiger dynamischer Eigenschaften der Regelstrecke und andererseits zur Beurteilung des Regelkreis-Verhaltens mit den gewählten Regler-Parametern.

Abstract

This thesis deals with the design and implementation of a frequency control for the analog relaxation oscillator of the CANakari controller used in the MOPS chip which is part of control system of the ATLAS pixel detector. The control system, consisting of a pulse counter, a PID controller, phase error control module and a Control FSM module, is wired with the digitally controlled analog oscillator and the bit timing logic. These components can communicate with each other, exchange data and thus form a closed control loop. The control algorithm observes the incoming signal Rx derived from the CAN bus and changes the manipulated variable if there is a control deviation by detecting a falling edge outside the synchronization segment defined in the CAN standard, so that the clock frequency is stabilized within a tolerance interval. This ensures that there are no synchronization errors during message transmission in the CAN network. Since it is a mixed analog / digital circuit, the control loop behavior is assessed with the aid of the A/MS simulations. The simulations are used on the one hand to investigate important dynamic properties of the controlled system and on the other hand to assess the control loop behavior with the selected controller parameters.

Inhaltverzeichnis

Abbildungsverzeichnis

1. Einleitung

Die CMOS[1]-Technologie ist auf dem Gebiet der integrierten digitalen Schaltungen von großer Bedeutung. Leistung wird von CMOS-Schaltungen nur während des Umschaltvorgangs aufgenommen. Aufgrund dessen ist die CMOS-Technologie die erste Wahl bei ICs[2] mit hoher Packungsdichte. Die niedrigen Herstellungskosten und die Möglichkeit sowohl analoge als auch digitale Schaltungen auf demselben Chip zu integrieren, um die Gesamtleistung zu verbessern und/oder die Kosten für die Produktion zu senken, machen die CMOS-Technologie nach wie vor attraktiv.

Die große Mehrzahl moderner ICs sind Mixed-Signal-ICs. Diese leistungsfähigen Schaltungen zeichnen sich durch die Integration von analogen und digitalen Komponenten auf einem Chip aus. Daher hat die Bedeutung von gemischt analog/digitalen Schaltungen in den letzten Jahren stark zugenommen. Die Simulation solcher Schaltungen ist jedoch problematisch, da die Verfahren für Analog- und Digitalschaltungen sehr unterschiedlich sind. Während bei einem Analogsimulator komplexe Differentialgleichungssysteme mit numerischen Verfahren gelöst werden, können sich digitale Simulatoren aufgrund verschiedener Eigenheiten digitaler Schaltungen (Zeit- und Wertdiskrete Signale) auf die Auswertung einfacher Datenstrukturen zu diskreten Zeitpunkten beschränken (Ereignissteuerung). Die Schwierigkeit einer gemischten Simulation (Mixed-Signal-Simulation) liegt insbesondere in der unterschiedlichen Geschwindigkeit, mit der die beiden Schaltungsarten ausgewertet werden können.

Im Rahmen des ATLAS-Pixeldetektor-Projekts wird der aus der Automobilindustrie bekannte CAN[3]-Standard für die Kommunikation zwischen den im Detektorkontrollsystem (MOPS[4] Chip) enthaltenen Komponenten eingesetzt. Da der CAN-Standard keine Übertragung eines Taktsignals vorsieht, ist die Synchronisation der Knoten in einem CAN-Netzwerk sehr wichtig, um sicherzustellen, dass Variationen der Oszillator-Frequenz bei den einzelnen Bus-Teilnehmern nicht zu Fehlern während der Kommunikation führen.

Diese Arbeit verfolgt das Ziel, die regelkonforme Funktionsweise des CANakari CAN-Controller Bittiming-Moduls durch Analog-Mixed-Signal-Simulationen zu verifizieren und ein Regelsystem zur Frequenzregelung, bestehend aus einem digitalen PID-Regler und anderen digitalen Komponenten, zu entwickeln. Das Regelsystem hat die Aufgabe, die Informationen, welche das Bittiming-Modul liefert auszuwerten und die Taktfrequenz des digital gesteuerten Relaxationsoszillators auf einen Sollwert von 10 MHz zu regeln. Denn zur Datenübertragung zwischen dem DCS[5] Computer im Leitstand und dem MOPS Chip im Pixeldetektor wird eine Übertragungsrate von 125 Kbit/s verwendet, die von einem internen Taktsignal mit einer Frequenz 10MHz abgeleitet wird.

Die Modellierung des digitalen Regelsystems erfolgt mittels der Verilog Hardware-Beschreibungsprache. Die Entwurfsumgebung besteht aus dem Bittiming-Modul des CAN

[1] Complementary metal-oxide-semiconductor
[2] integrated circuit
[3] Controller Area Network
[4] Monitoring of Pixel System
[5] Detector control system

Controllers und einem Regelsystem-Modul, welche beide vom Simulator als digitale Komponenten betrachtet werden und einem digital gesteuerten Relaxationsoszillator, welcher den analogen Teil repräsentiert. Daher dient der erste Teil dieses Berichtes als eine Einführung in das CAN Protokoll. Die Einführung beinhaltet auch eine ausführliche Beschreibung der Bittiming Logik, welche für die Synchronisation zuständig ist. Im zweiten Teil dieses Berichtes erfolgt eine Dokumentation des Entwurfs und der Dimensionierung des strahlenharten Relaxationsoszillators in 65nm CMOS-Technologie, welcher als Taktgeber in der Testschaltung eingesetzt wird. Im dritten Teil des Berichts wird die Einrichtung einer Analog/Mixed-Signal Simulation schrittweise veranschaulicht und erläutert. Der vierte und letzte Teil dieses Berichts beinhaltet eine detaillierte Beschreibung der Funktionsweise des Regelsystems, welches für die Regelung der Taktfrequenz des Oszillators verantwortlich ist. Abschließend werden die Simulationsergebnisse im letzten Teil des Berichts grafisch dargestellt und interpretiert.

2. Grundlagen zum CAN Protokoll

Das CAN[6] Protokoll regelt die Kommunikation auf einem seriellen BUS[7], bei dem mehrere gleichberechtigte Komponenten über einen 2-Draht Bus miteinander verbunden werden. Das CAN Protokoll ist als ISO[8] 11898 international standardisiert und umfasst die Layer 1, d.h, die physische Schicht und den Layer 2, d.h., die Datensicherungsschicht im ISO/OSI-Referenzmodell. ISO/OSI steht hierbei für International Standardisation Organisation/Open Systems Interconnect.

Aufgrund der hohen Störsicherheit, der geringen Kosten und der Echtzeitfähigkeit wird CAN auch in der Automatisierungstechnik eingesetzt. Die elektrische Störsicherheit wird unter anderem dadurch erreicht, dass ein Bit auf zwei Leitungen gleichzeitig mit einer gegensinnigen Potenzialänderung abgebildet wird. Eingestreute Störungen wirken auf beide Leitungen in der gleichen Richtung ein. Da die beiden differentiellen Leitungen jedoch immer gegensinnige Pegel haben, bleibt die Differenz der Pegel trotz Störungen erhalten.

Die Leitungen CAN-High und CAN-Low enthalten das invertierte und das nicht invertierte serielle Datensignal. Der Zustand mit zwei unterschiedlichen Pegeln auf CAN-H und CAN-L wird der dominante Zustand genannt und entspricht per CAN Definition einer logischen Null. Der Zustand mit zwei gleichen Pegeln wird als rezessiv bezeichnet und entspricht einer logischen Eins. Legt ein Knoten eine logische Null auf den Bus, überschreibt er möglicherweise den Zustand einer logischen Eins eines anderen Knotens, da die Kopplung der Knoten über die Busleitung eine logische Und-Verknüpfung darstellt.

Die Daten auf dem CAN-Bus werden in der (Non Return to Zero) NRZ- Codierung übertragen. Zur Erhöhung der Störsicherheit wird eine zyklische Redundanzprüfung (CRC) durchgeführt. Ein Bit-Stuffing-Mechanismus wird von CAN verwendet, um bei der Kommunikation zwischen mehreren Busteilnehmern, deren Taktgeneratoren untereinander nicht synchronisiert sind, fehlerhafte Bit-Erkennung zu vermeiden. Dieses Verfahren fügt ein Stuffing-Bit mit invertierter Polung nach fünf gleichpoligen Bits in dem Bitstrom. Somit wird verhindert, dass die Übertragung zwischen Sender und Empfänger auseinanderdriftet. Beim Empfang einer Folge von fünf gleichen Bits entfernen die Empfänger das folgende sechste Bit, damit dieses zusätzliche Bit nicht zu Fehlern in der Nachrichtenübertragung führt und die ursprüngliche Information erhalten bleibt.

2.1. CAN Rahmenformat Felderbeschreibung

Es wird zwischen folgenden CAN-Botschaftsarten bzw. Frames unterschieden:

- Normale Botschaften bzw. Daten-Frames sind CAN-Nachrichten, welche vom Sender an die Empfänger übertragen werden.

[6] Controller Area Network
[7] Binary Unit System
[8] International Standard Organization

- Remote-Botschaften bzw. Remote-Frames ermöglichen einem Teilnehmer des Netzwerks, notwendige Informationen von anderen Teilnehmern anzufordern.
- Error-Frames werden versandt, um anzuzeigen, dass ein Fehler bei der Übertragung festgestellt wurde.
- Overload-Frame, dient als Zwangspause zwischen Daten- und Remote-Frames.

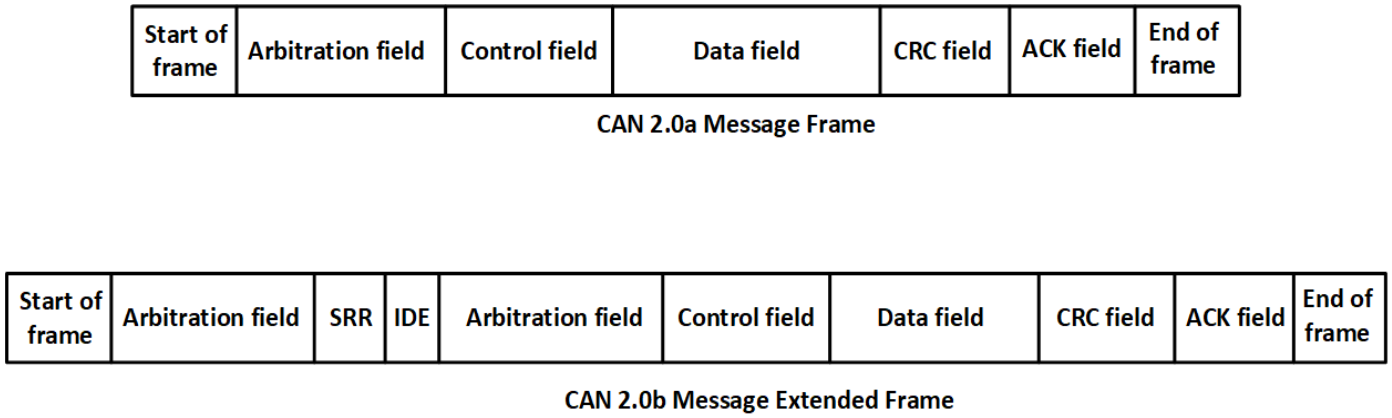

Abbildung 1: Aufbau einer CAN-Botschaft

2.1.1. Datenrahmen

Das CAN Rahmenformat besteht aus sieben Segmenten. Start of Frame (SOF), Arbitrationfeld, Control Feld, Data Feld, Cyclic Redundancy Check (CRC) Feld, Acknowledge und End of Frame (EOF) (siehe Abbildung 1).

Start of Frame

Das SOF definiert den Beginn eines neuen Sendevorgangs. Es besteht aus einem dominanten Bit und wird von allen empfangenden Einheiten des Busses zur Synchronisation mit der sendenden Einheit verwendet. Sollten mehrere Teilnehmer gleichzeitig eine Transmission starten wollen, wird über einen Arbitrationsprozess entschieden, welcher Teilnehmer den Bus belegen kann.

Arbitrationsfeld

Das Arbitrationsfeld ist entscheidend dafür, welche Botschaft an die Empfänger des CAN-Busses übertragen wird. Das Feld besteht aus einem 11 oder 29-Bits Identifier und einem Remote-Transmission (RTR) Bit. Ziel des Arbitrierungsvorgangs ist die Übertragung der Botschaft, welche die höchste Priorität besitzt. Die Botschaft mit dem niedrigsten

numerischen Wert im Identifier-Feld, entspricht der höchsten Priorität, da der Arbitrierungsprozess bitweise vorgeht und die Bits über eine UND-Verknüpfung ausgewertet werden.

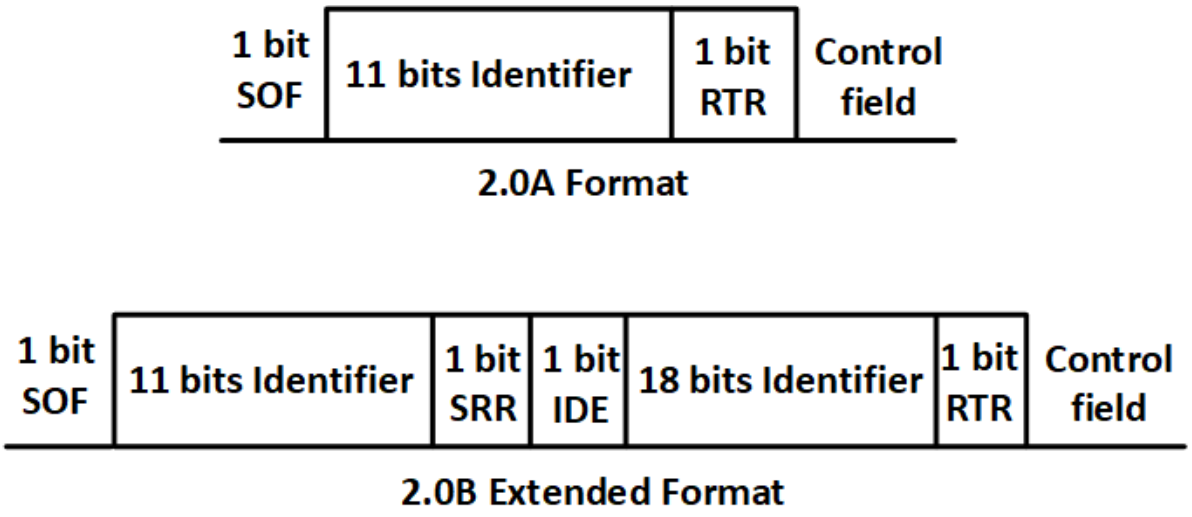

Abbildung 2: Arbitrationsfeld

Das RTR–Bit unterscheidet zwischen Data und Remote-Frame. Bei Data-Frames ist das RTR-Bit **dominant** und bei Remote-Frames **rezessiv**, woraus eine höhere Priorität von Data-Frames folgt.

Control-Feld

Das Control-Feld besteht aus dem Identifier Extension Bit, welches für eine Identifier-Länge von 11-Bit dominant und für einen 29-Bit Identifier rezessiv sein muss. Darauf folgt ein dominantes reserved Bit. Die letzten vier Bits des Control-Feldes werden als Data Length Code bezeichnet und enthalten die Längeninformation des nachfolgenden Data-Feldes.

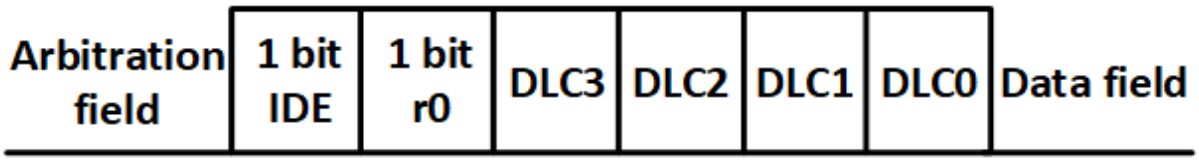

Abbildung 3: Kontroll-Feld

Datenfeld

Das Datenfeld kann aus null bis acht Bytes an Daten bestehen und beinhaltet die zu übertragende Bitfolge.

CRC-Feld

Das 16 Bit breite CRC-Feld kennzeichnet die zyklische Prüfsumme über alle vorangegangene Felder. Das CRC Feld besteht aus einer 15 Bit CRC Summe und aus dem rezessiven CRC Delimiter.

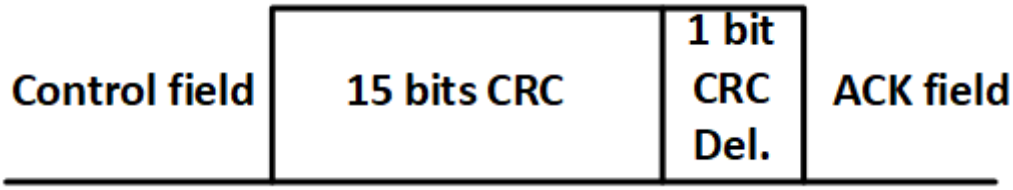

Abbildung 4: CRC-Feld

Acknowledge-Feld

Das Acknowledge-Feld enthält die Rückmeldung der anderen Bus-Teilnehmer, ob die Nachricht erfolgreich empfangen wurde oder nicht. Ein dominantes Acknowledge Bit zeigt dabei einen erfolgreichen Empfang der CAN-Nachricht an.

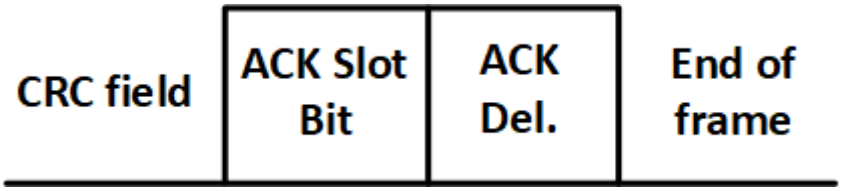

Abbildung 5: Das Acknowledge-Feld

End of Frame

Das End of Frame Feld kennzeichnet das Ende der CAN-Nachricht und besteht aus sieben rezessiven Bits.

2.1.2. Remoterahmen

Durch Remote-Frame kann ein Busteilnehmer einen anderen Busteilnehmer veranlassen, bestimmte Daten zu senden. Der Aufbau des Remote-Frames und Daten-Frames unterscheidet sich insoweit, dass beim Remote-Frame das RTR-Bit rezessiv ist und das Datenfeld vollständig entfällt.

2.1.3. Error-Frame

Wird ein Fehler von einem Busteilnehmer erkannt, werden die anderen Busteilnehmer durch die Sendung des Error-Frames informiert. Der Error-Frame besteht aus zwei Feldern, dem Error Flag und dem Error Delimiter. Es gibt zwei Arten von Error Frames, Active und Passive Error Frames. Im Error Active Modus wird ein Active Error Frame gesendet, welcher aus 6 dominanten Bits besteht, gefolgt vom Error Delimiter 8 rezessiven Bits, während im Error Passive Zustand das Passive Error Frame, aus 6 rezessiven Bits und dem Error Delimiter besteht.

2.1.4. Overload-Frame

Das Overload–Frame beinhaltet zwei Felder, das Overload–Flag und den Overload–Delimiter. Das Overload–Flag besteht aus 6 dominanten Bits, der Overload-Delimiter besteht aus 8 rezessiven Bits. Der Sinn des Frames besteht darin, zu signalisieren, dass der empfangende Teilnehmer Zeit braucht, bevor er die nächste Nachricht senden bzw. empfangen kann.

2.2. CAN-Bus Bitzeit

Alle Teilnehmer des CAN-Busses müssen mit derselben Bitzeiteinheit (Nominal Bit Time) arbeiten, sodass eine einheitliche Übertragungsrate auf dem Bus herrscht. Die Synchronisation der Knoten in einem CAN-Netzwerk ist sehr wichtig, um sicherzustellen, dass Variationen der Oszillator-Frequenz bei den einzelnen Bus-Teilnehmern nicht zu Fehlern während der Kommunikation führen.

CAN–Controller verfügen über die Möglichkeit einer harten (Hard Synchronisation) und weichen (Soft Synchronisation) Synchronisation. Die harte Synchronisation wird beim ersten rezessiven zum dominanten Übergang eines Startbits durchgeführt und markiert den Start einer neuen Botschaft. Die weiche Synchronisation oder die Resynchronisation erfolgt bei jedem Flankenwechsel von rezessiv zu dominant und dient der Verkürzung beziehungsweise Verlängerung der Bitzeiten. Am häufigsten werden Resynchronisationsmechanismen während der Arbitrierungsphase initialisiert. Die Busteilnehmer synchronisieren sich auf den „führenden“ Sender, der als erster zu senden beginnt. Falls der führende Sender nicht die

Arbitrierung gewinnt, müssen sich die Empfänger auf andere CAN Einheiten synchronisieren, die zeitweise die Führung übernehmen und eventuell zum vorherigen Sender nicht gleich synchronisiert sind.

Die Teilnehmer auf dem CAN–Bus sind mit einem individuellen Taktgenerator ausgestattet, was die Anpassung der Bitzeit in jedem CAN-Knoten ermöglicht, sodass eine einheitliche Übertragungsrate herrscht. Jedes Bit wird in eine Anzahl von Grundzeiteinheiten T_q unterteilt, welche als Time–Quantum bezeichnet, und in vier Segmente gruppiert werden. Dabei handelt es sich um das Synchronisationssegment, das Propagation-Segment, das Phase Buffer Segment 1 und das Phase Buffer Segment 2. Die Dauer eines Time–Quantums ist ein ganzzahliges Vielfaches der Oszillatorperiode.

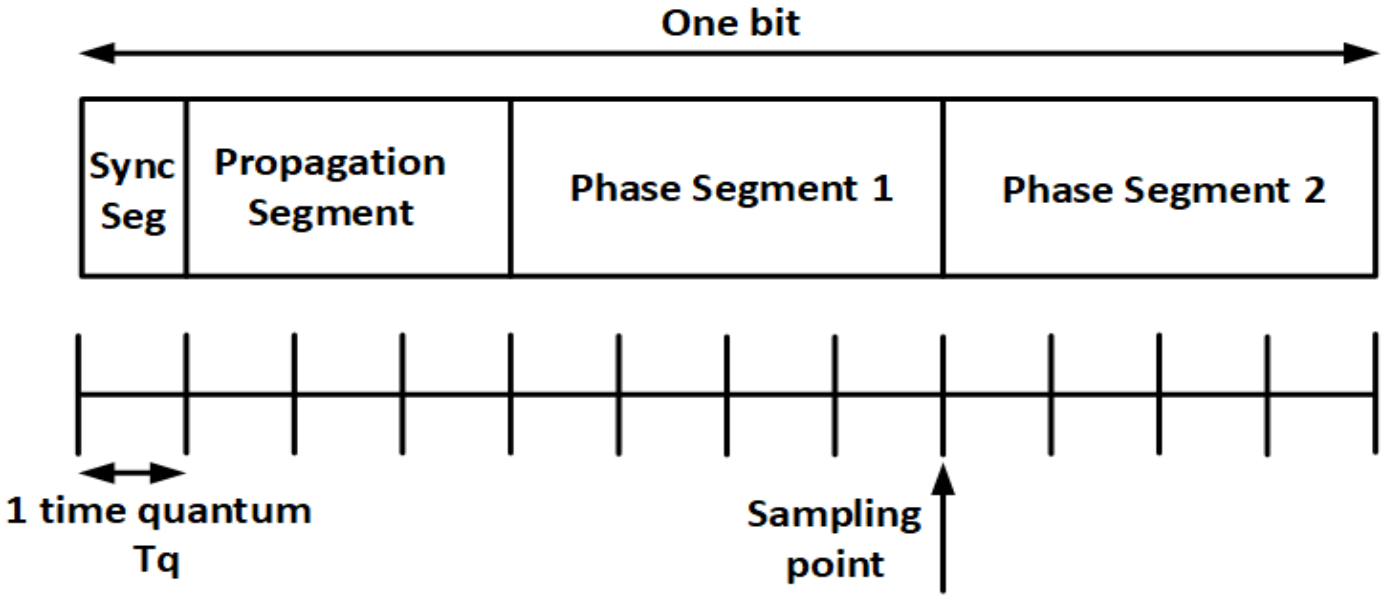

Abbildung 6: Zusammensetzung einer Bitzeiteinheit

Das Synchronisationssegment ist ein Time-Quantum lang und wird zur Synchronisation der Knoten im CAN-Bus verwendet. Innerhalb dieses Segments wird ein Flankenwechsel erwartet.

Das Propagationssegment kompensiert die Übertragungsverzögerung in den Busleitungen.

Phase Buffer Segment 1 und Phase Buffer Segment 2 dienen dazu, die Länge des Bits anzupassen, falls der Flankenwechsel verfrüht oder verspätet auftritt. Der Abstand zwischen einer Flanke und dem Synchronisationssegment wird als Phasenfehler bezeichnet. Im Falle eines positiven Phasenfehlers wird das Phase-Segment verlängert bzw. bei einem negativen Phasenfehler verkürzt. Hierbei gilt als Kriterium, ob der Flankenwechsel nach dem Synchronisationssegment oder vor dem Synchronisationssegment aufgetreten ist.

Die maximale Anzahl an Time-Quanten, um die das Phase–Segment 1 und Phase–Segment 2 verlängert bzw. verkürzt werden dürfen, wird durch den Wert der Synchronisationssprungweite (SJW) bestimmt. Die Synchronisationssprungweite (SJW)

definiert, wie weit der Sample-Point innerhalb der Phase Buffer Segmente verschoben werden darf, um Phasenfehler zu kompensieren. Laut CAN-Spezifikation sollte der SJW-Wert zwischen 1 und 4 einstellbar sein.

Ist der Phasenfehler positiv und kleiner als die maximale Synchronisationssprungweite SJW, wird das Phase Buffer 1 Segment entsprechend dem Phasenfehler verlängert, ansonsten wird das Phase Buffer 1 Segment nur um SJW verlängert.

Ist der Phasenfehler negativ und sein Betrag kleiner als die maximale Synchronisationssprungweite SJW, wird das Phase Buffer 2 Segment entsprechend dem Phasenfehler verkürzt, ansonsten wird das Phase Buffer 2 Segment nur um SJW verkürzt.

Wenn der Betrag jedoch größer als die maximale Synchronisationssprungweite ist, kann der Phasenfehler nicht vollständig kompensiert werden und es bleibt ein Restfehler (Phasenfehler - SJW) erhalten.

Puffer Segmente werden nur zeitweise verlängert bzw. verkürzt. Zu Beginn der nächsten Bitzeit nehmen sie wieder die voreingestellten Werte an.

2.3. Timing Logik Einheit

Die Timing Logik besteht aus einem Zustandsautomaten, der den Bus auf fallende Flanken überwacht, einem Puffer, der den Buswert Rx speichert und zeitverzögert an den Zustandsautomat weiterleitet, zwei Registern TsegReg und SmpldbitReg, die für die Anpassung der Bit-Segmente im Falle einer Resynchronisation und für die Signalisierung, dass das aktuelle Bit auf dem Bus gespeichert werden soll, gedacht sind, einem Zähler, welcher angibt, wie viele Time-Quanten T_q abgelaufen sind und mit dem Prescale_EN getaktet ist, einem arithmetischen Bauteil, das den Zählerstand und die Länge der Bitsegmente verarbeitet und dem Zustandsautomaten Signale und Werte liefert, die er braucht, um bei fallenden Flanken protokollkonforme Synchronisationsentscheidungen zu fällen (siehe Abbildung 7).

Der Timing Logik Einheit werden die Signale tseg1, tseg2, sjw, clock, Prescale_EN, reset, hardsync und der Buswert Rx zugeführt, wobei die Zeitsegmente tseg1, tseg2 und sjw dem Propagationssegment, Phase Buffer 1 Segment, dem Phase Buffer 2 Segment und Synchronisationssprungweite entsprechen. Das clock und Prescale_EN entsprechen dem Takt Generator Signal und dem zugehörigen Vorteiler. Der Wunsch einer Durchführung der harten Synchronisation wird der FSM über das Hardsync Signal übermittelt.

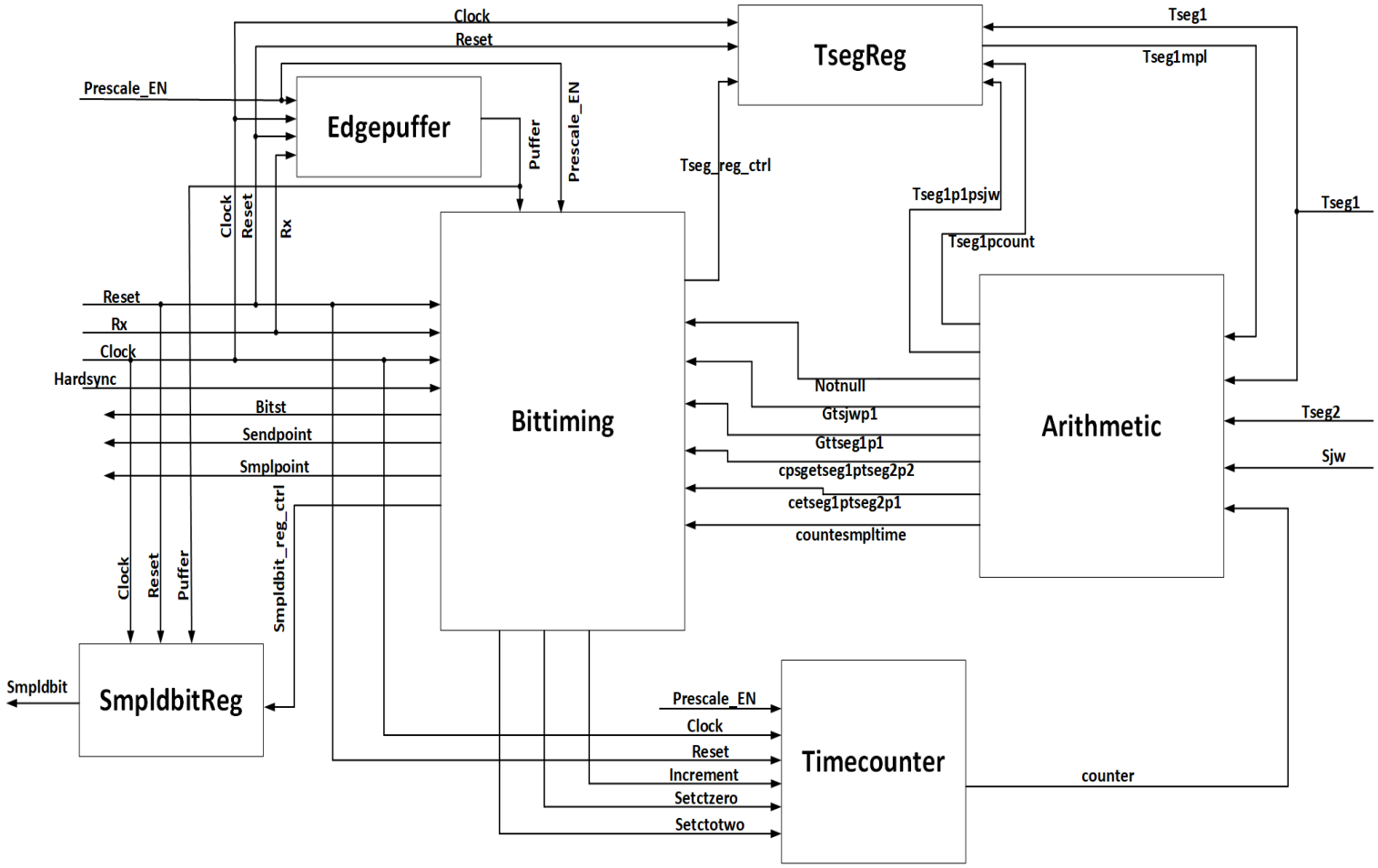

Abbildung 7: Aufbau der Timing Logik Einheit

Die arithmetische Einheit setzt in Abhängigkeit vom Zählerstand einige Signale. Das Signal notnull ist immer dann wahr, wenn der Zählerstand einen von null unterschiedlichen Wert hat. Dadurch kann die Bittiming FSM erkennen, ob eine fallende Flanke außerhalb des Synchronization Segments auf dem Bus detektiert worden ist. Das Signal gtsjwp1 (greater then [sjw plus 1]) wird von der arithmetischen Einheit gesetzt, wenn der Zählerstand größer als der Wert von sjw plus eins ist. Da die Werte von tseg1, tseg2 und sjw aus Platzgründen in die entsprechenden Bereiche des Generalregisters[9] um eins dekrementiert wurden, wird eins zu den Werten in der arithmetischen Einheit addiert.

Erkennt die FSM eine fallende Flanke bei einem Zählerstand der kleiner als die Synchronization Jump Width ist, kann das Zeitsegment tseg1 einfach um den aktuellen Zählerstand vergrößert werden. Ansonsten darf das Zeitsegment nur um den Wert von sjw gedehnt werden.

Der Wert von tseg1pcount wird dem tseg1 im TsegReg-Register, bei einem Zählerstand, welcher kleiner als sjw ist, bzw. tseg1p1psjw bei einem Zählerstand, welcher größer als sjw ist, zugewiesen und über das Signal tseg1mpl (tseg1 manipulated) an die arithmetische Einheit

[9] Enthält Informationen, welche von globaler Wichtigkeit für die Funktion des CAN-Controllers sind.

weitergeleitet. Das TsegReg-Register wird von der FSM über das Signal tseg_reg_ctrl gesteuert.

Das Signal countesmpltime wird wahr, wenn der Wert des Zählers dem Samplezeitpunkt entspricht. Der Samplezeitpunkt ist erreicht, wenn das Ende des tseg1 Segmentes erreicht ist (counter=tseg1+1). Zu diesem Zeitpunkt setzt die FSM das smplpoint Signal und gibt den aktuellen Buspegel an den Medium Access Controller[10] weiter. Ist zuvor auf Grund einer verspäteten fallenden Flanke das tseg1 Segment verlängert worden, wird auch der Samplezeitpunkt um den gleichen Zeitraum verzögert. Das Signal gttseg1p1 wird wahr, wenn der Zählerstand den Samplezeitpunkt überschritten hat und sich somit im Zeitsegment tseg2 befindet. Erkennt die FSM nun eine fallende Flanke, verkürzt sie das Segment tseg2. Befindet sich der aktuelle Zählerstand bei Erkennung der fallenden Flanke sjw oder weniger Zeiteinheiten vom Ende des Zeitsegmentes tseg2 entfernt, kann die FSM die aktuelle Position als Synchronisationssegment interpretieren und somit in den Anfang des tseg1 Segmentes springen. Befindet sich der aktuelle Zählerstand bei Erkennung der fallenden Flanke mehr als sjw Zeiteinheiten vom Ende des Zeitsegmentes tseg2 entfernt, darf sie den Beginn des Zeitsegmentes tseg1 erst einleiten, wenn die entsprechende Distanz erreicht worden ist. Zu diesem Zweck wird das Signal cpsgetseg1ptseg2p2 (counter + sjw greater or equal tseg1 + tseg2 +2) benutzt. Das Signal wird wahr, wenn die Distanz zum Ende des Zeitsegments tseg2 sjw oder weniger Zeiteinheiten beträgt. Das Signal cetseg1ptseg2p1 (=tseg1p1mpl+tseg2) wird gesetzt, wenn der Zählerstand das Ende des Zeitsegments tseg2 entspricht.

Wird das Ende des Zeitsegmentes tseg2 regulär, d.h. ohne den Erhalt einer verfrühten fallenden Flanke erreicht, springt die FSM in der letzten Grundzeiteinheit in einen speziellen Zustand, in dem über das Signal setctzero der Zähler zurückgesetzt wird und durch das Signal sendpoint der Medium Access Controller aufgefordert wird, das nächste Bit auf den Bus zu legen.

[10] Der MAC-Layer stellt den Kern des CAN-Protokolls dar. Er beschreibt CAN-Rahmen, die Arbitrierung sowie die Fehlererkennung und Signalisierung.

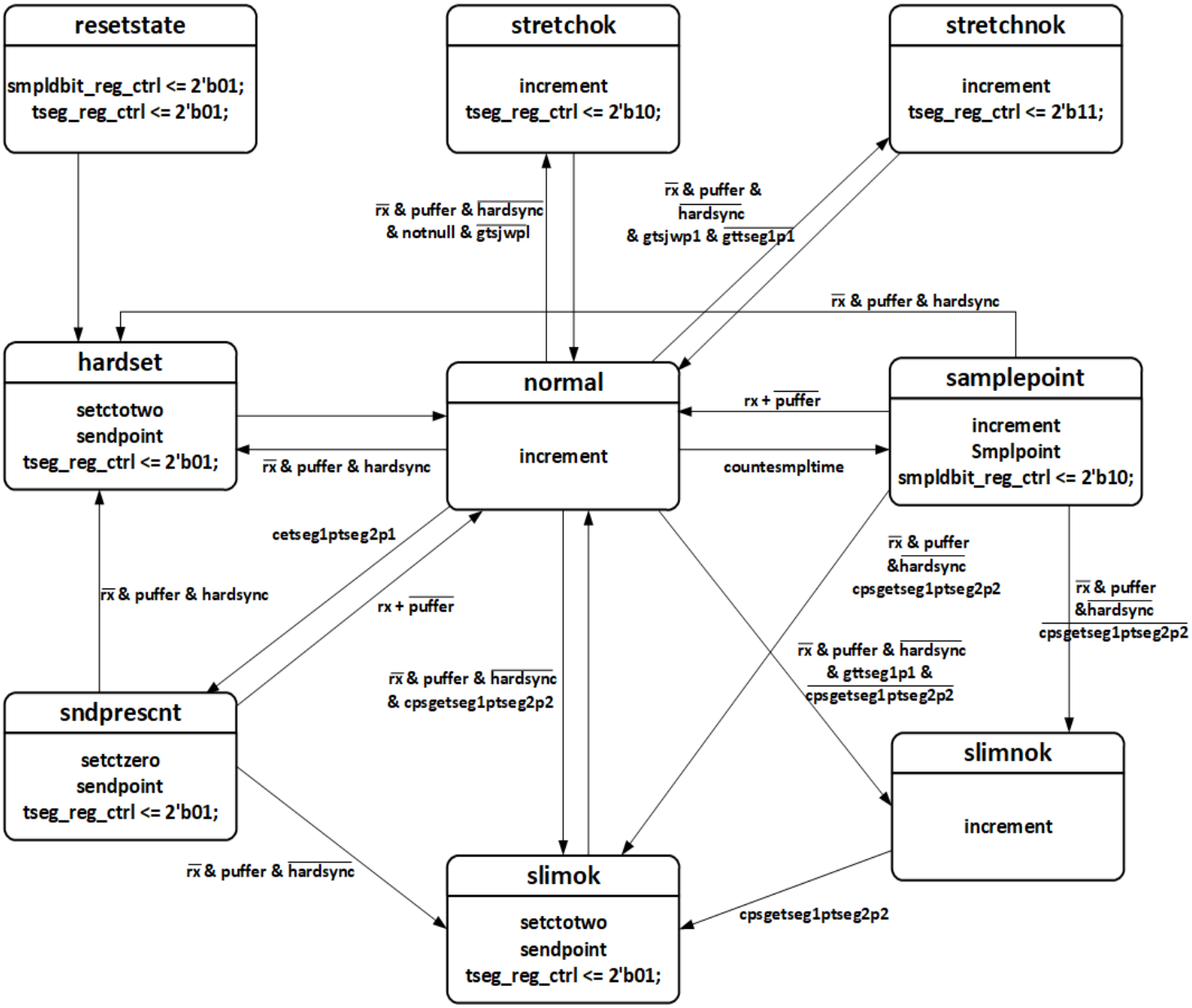

Abbildung 8: Zustandsübergangsdiagramm des Timingzustandsautomats

Das Zustandsübergangsdiagramm mit allen verwendeten Signalen wird in Abbildung 8 dargestellt. Mit einem Reset des Controllers wird das smpldbit Signal gesetzt, in dem der Wert „01“ dem Register smpldbit_reg_ctrl zugewiesen wird. Der Zustandsautomat geht aus dem Zustand resetstate in den Zustand hardset über. In diesem Zustand werden die Signale setctotwo und sendpoint gesetzt. Nachdem der Zählerwert auf zwei gesetzt wurde und der Medium Access Controller das nächste Bit auf den Bus gelegt hat, geht die FSM in den Zustand normal. In diesem Zustand wird zu jedem Takt der Zähler inkrementiert und auf fallende Flanken gewartet. Eine fallende Flanke manifestiert sich dadurch, dass der aktuelle Buswert rx den Wert null hat und der zu Beginn der Grundzeiteinheit in den Puffer gespeicherte Buswert den Wert eins hat (puffer=’1’ and rx=’0’). Wird eine solche fallende Flanke erkannt, während das hardsync Signal aktiviert ist, wird im Zustand hardset die harte Synchronisation durchgeführt. Bei einer harten Synchronisation wird der Grundzeitabschnitt, in dem die

fallende Flanke aufgetreten ist, als Synchronization Segment betrachtet. Das Synchronization Segment wird normalerweise dem Zählerstand null zugeordnet. Die Beeinflussung des Zählerstandes ist jedoch erst eine Takteinheit später im Zustand hardset möglich. Zu diesem Zeitpunkt müsste der Zähler bereits den Wert eins beinhalten. Die FSM setzt aus diesem Grund das Signal setctotwo, das den Zähler zum nächsten Takt auf den Wert zwei setzt. Ist das hardsync Signal nicht gesetzt und tritt eine fallende Flanke auf wird in Abhängigkeit der oben besprochenen Signale das Zeitsegment tseg1 verlängert (stretchok, stretchnok) oder das Zeitsegment tseg2 verkürzt. Tritt die Fehlerflanke während eines Zählerstandes ein, der nicht null (notnull) und kleiner oder gleich der sjw ist (gtsjwp1=0) wird im Zustand stretchok das Zeitsegment tseg1 entsprechend dem Zählerwert verlängert, in dem der binär kodierte Wert „10“ dem Register tseg_reg_ctrl zugewiesen wird. Ist der Zählerstand jedoch größer als der Wert der sjw (gtsjwp1=1) und ist der Samplezeitpunkt noch nicht überschritten (gttseg1p1=0), wird das Zeitsegment tseg1 im Zustand stretchnok um den Wert der **sjw** verlängert, in dem der binär kodierte Wert „11“ dem Register tseg_reg_ctrl zugeordnet wird.

Erreicht der Zählerstand den Samplezeitpunkt, geht die FSM in den Zustand samplepoint über, setzt das smplpoint Signal und leitet den aktuellen Buswert an den Medium Access Controller weiter (smpledbit<=puffer), in dem der binär kodierte Wert „10“ dem Register smpldbit_reg_ctrl zugewiesen wird. Falls eine fallende Flanke (rx=1 und puffer=1) während diesem Zustand auftritt und das hardsync Signal gesetzt ist, springt die FSM in dem Zustand hardset. Ist das hardsync Signal nicht gesetzt und die Distanz zum Ende des Zeitsegments tseg2 kleiner oder gleich sjw (cpsgetseg1ptseg2p2=1), springt die FSM in den Zustand slimok**.** Ansonsten geht die FSM in den Zustand slimnok über**,** was zu einer Verkürzung des Zeitsegments tseg2 führt**.**

Befindet sich der Zählerstand im Zeitsegment tseg2 und ist die Distanz zum Ende des Zeitsegments größer als sjw (gttseg1p1=1 und cpsgetseg1ptseg2p2=0), springt die FSM in den Zustand slimnok**.** In diesem Zustand wird der Zähler inkrementiert, bis die Entfernung zum Ende des Zeitsegments tseg2 kleiner oder gleich sjw ist (cpsgetseg1ptseg2p2=1). Dann geht die FSM in den Zustand slimok über, um die Verkürzung des Zeitsegmentes tseg2 einzuleiten. In diesem Zustand wird dem Zähler der Wert zwei (setctotwo=1) zugewiesen und das Signal sendpoint gesetzt.

Erreicht der Zählerstand die vorletzte Grundzeiteinheit des Zeitsegments tseg2 wird das Signal cetseg1ptseg2p1 von der arithmetischen Einheit aktiviert, so dass die FSM zum nächsten Takt in den Zustand sndprescnt befördert wird. In diesem Zustand wird der Zähler auf null zurückgesetzt(setctzero=1) und der Medium Access Controller wird über das Signal sendpoint aufgefordert, das nächste Bit auf den Bus zu legen. Falls eine fallende Flanke während dieses Zustands auftritt und das hardsync Signal gesetzt ist, springt die FSM in den Zustand hardset. Ist das hardsync Signal nicht gesetzt, springt die FSM in den Zustand slimok**.**

3. Der Relaxationsoszillator in 65nm CMOS Technologie

3.1. Aufbau eines Doppel-Kondensator-Relaxationsoszillator

Ein **Relaxationsoszillator** ist ein elektronischer Signalgenerator, bei dem ein Energiespeicher wie beispielsweise ein Kondensator fortwährend geladen und in Kombination mit einem Transistorschalter wie MOSFETs entladen wird. Dadurch wird eine Kippschwingung (Sägezahnschwingung) erzeugt, welche das Aussehen eines Sägezahns besitzt. Durch zusätzliche Schaltungsteile wie analoge Komparatoren können weitere Schwingungsformen wie z.B. eine Rechteckschwingung abgeleitet werden.

Relaxationsoszillatoren werden wegen ihrer relativ hohen Frequenzgenauigkeit und ihres geringen Energieverbrauchs bevorzugt. Außerdem lassen sie sich kostengünstig produzieren. Sie werden vielfach u.a. zur Spannungs/Frequenzwandlung, zur FM-Modulation und in phasengeregelten Schaltkreisen (PLL-Schaltkreisen) zur Frequenzsynchronisation, zur Taktsynchronisation oder zur FM-Demodulation verwendet.

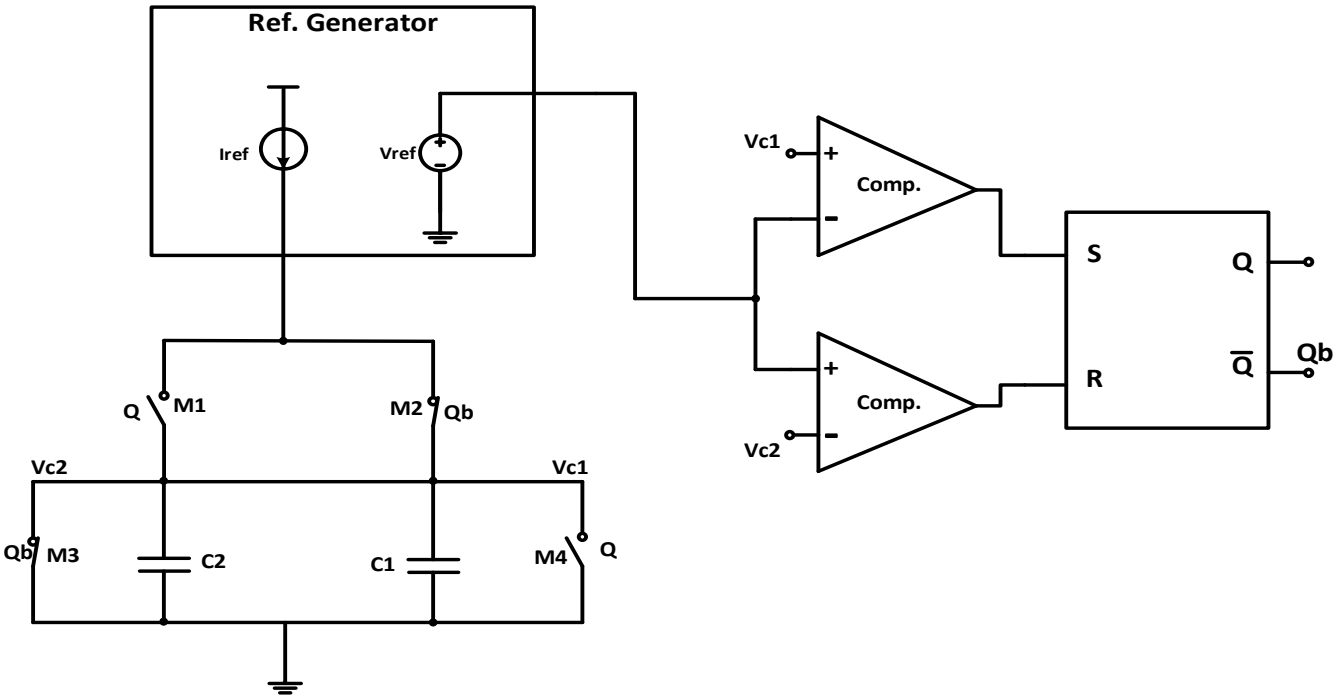

Abbildung 9: Grundaufbau des Doppel-Kondensator-Relaxationsoszillator

Wie in Abbildung 9 zu erkennen ist, setzt sich der Relaxationsoszillator aus mehreren wichtigen Teilen zusammen. Ein Referenzgenerator erzeugt die Referenzspannung V_{REF} und den Referenzstrom I_{REF}. Dieser Strom wird abwechselnd auf die Kondensatoren C1 und C2 geführt, um durch Integration des Stromes Spannungsverläufe zu erzeugen. Außerdem

werden Schalter verwendet, welche zu den Kondensatoren parallel und in Reihe geschaltet sind, um den Stromfluss zu steuern und bei Bedarf die Kondensatoren zu entladen.

Ist der Stromfluss I_{REF} durch den Kondensator C1 bzw. C2 konstant, steigt die Spannung V_{C1}bzw. V_{C2} linear mit der Zeit an. Wird der Kondensator C1 bzw. C2 nicht weiter geladen und zeitgleich

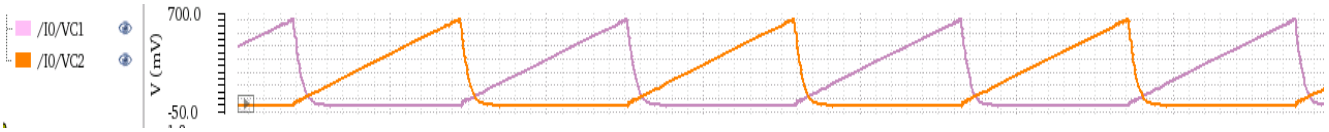

Abbildung 10: Sägezahnspannungen V_{C1} und V_{C2}

der parallelgeschaltete Schalter geschlossen, wird dem Kondensator Energie entnommen und die Kondensatorspannung V_{C1} bzw. V_{C2} sinkt (Abbildung 10). Die Ausgangssignale der beiden Komparatoren, welche die Spannung V_{C1} bzw. V_{C2} mit der Referenzspannung V_{REF} vergleichen, sind mit dem Set- bzw. Reset-Eingang des SR-Latchs verbunden. Die Ausgänge der Komparatoren werden dann dem S-R-Latch zugeführt, um die Lade- und Entladevorgänge für C1 und C2 zu steuern. Die Schalter M1, M2, M3 und M4 sind mit den SR-Latch Ausgangssignalen Q und Qb verbunden.

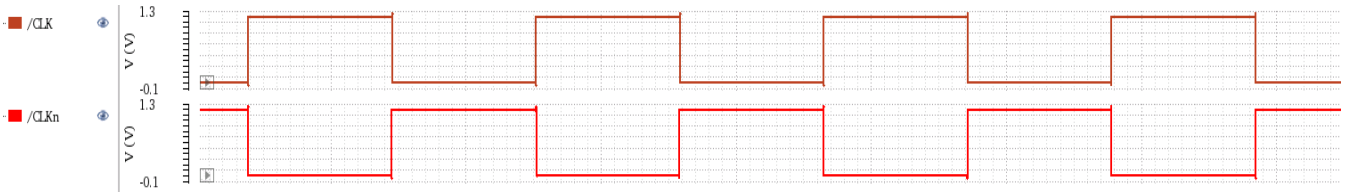

Abbildung 11: Pulssignale Q und Qb

Die Schaltfrequenz des erzeugten Pulssignals Q bzw. Qb (Abbildung 11) kann wie folgt ermittelt werden.

Die auf dem Kondensator gespeicherte Ladung Q kann mit folgender Gleichung beschrieben werden :

$$\boldsymbol{Q = \int I_{REF}\, dt = I_{REF} \int dt = I_{REF} * T} \qquad (3.1.1)$$

Wobei **T** der Periodendauer des Signals entspricht.

Und für den Zusammenhang zwischen Spannung und Ladung am Kondensator gilt:

$$\boldsymbol{Q = C * V_{REF}} \tag{3.1.2}$$

Aus den beiden Gleichungen ergibt sich

$$\boldsymbol{C * V_{REF} = I_{REF} * T} \tag{3.1.3}$$

Für C_1und C_2 gelten die folgenden Zusammenhänge

$$\boldsymbol{T_1 = \frac{C_1 V_{REF}}{I_{REF}}} \text{ und } \boldsymbol{T_2 = \frac{C_2 V_{REF}}{I_{REF}}} \tag{3.1.4}$$

Da $C_1 = C_2 = C$ und $T_1 + T_2 = T_{osc}$ sind, folgt daraus

$$\boldsymbol{T_{osc} = \frac{2CV_{REF}}{I_{REF}}} \tag{3.1.5}$$

Und somit

$$\boldsymbol{f_{osc} = \frac{I_{REF}}{2CV_{REF}}} \tag{3.1.6}$$

daraus kann interpretiert werden, dass die Genauigkeit der Schaltfrequenz $\boldsymbol{f_{osc}}$ von der Genauigkeit des Referenzstromflusses $\boldsymbol{I_{REF}}$ und des Referenzspannungswerts $\boldsymbol{V_{REF}}$ abhängt. Darüber hinaus ist die Schaltfrequenz umgekehrt proportional zu der Gesamtkapazität.

3.2. Erweiterung des Relaxationsoszillators

Durch den im vorherigen Abschnitt erwähnten Zusammenhang (3.1.6) besteht die Möglichkeit, eine Variation der Schaltfrequenz des Oszillators zu realisieren, in dem die Schaltung, welche die Kapazität definiert, konfigurierbar aufgebaut wird.

Von daher wird der Grundaufbau des Doppel-Kondensator-Relaxationsoszillators (siehe Abbildung 9) modifiziert und so erweitert, dass die Kapazität durch zuschaltbare Kondensatoren verändert werden kann (Abbildung 12).

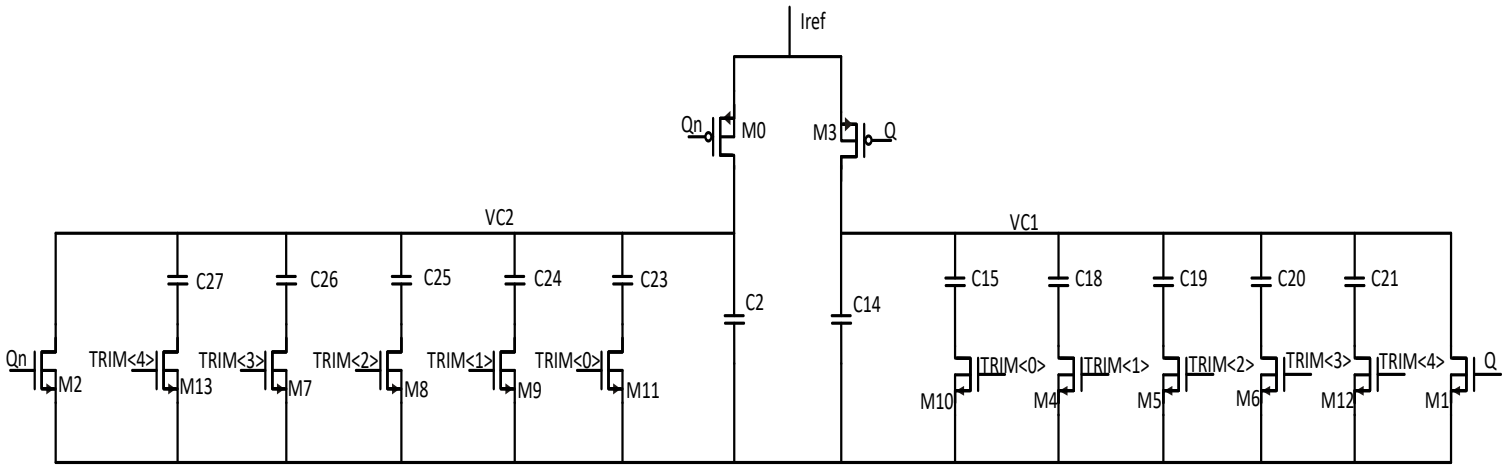

Abbildung 12: steuerbare parallelgeschaltete Kondensatoren

Die schaltbaren Kondensatoren sind wie folgt dimensioniert:

	C_{links}					**C2**	**C2**	C_{rechts}				
Instanzname	C27	C26	C25	C24	C23	C2	C14	C15	C18	C19	C20	C21
Kapazitätswert pro Kondensator	21,44 fF					1,231 pF		21,44 fF				
Multiplier	16	8	4	2	1	1	1	1	2	4	8	16
Gesamtkapazität	343,1fF	171,5fF	85,77fF	42,89fF	21,44fF	1,231pF	1,231pF	21,44fF	42,89fF	85,77fF	171,5fF	343,1fF
Steuersignal	Trim<4>	Trim<3>	Trim<2>	Trim<1>	Trim<0>	Kein Steuersignal		Trim<0>	Trim<1>	Trim<2>	Trim<3>	Trim<4>

Tabelle 1: die binär-gewichtete Kondensatoren

Der Multiplizier-Faktor bezeichnet die Anzahl der parallelgeschalteten Kondensatoren.

Die Schalttransistoren (M13, M7, M8, M9, M11, M10, M4, M5, M6, M12), welche mit den Kondensatoren in Reihe geschaltet sind, werden durch externe Digitalsignale (TRIM<4:0>) gesteuert (Abbildung 12).

Durch das Konzept mit schaltbaren parallelen Kondensatoren wird eine Variation der Gesamtkapazität in einem Intervall von $[2.5pF, 3.8pF]$ ermöglicht, und somit kann die Schaltfrequenz des Oszillators im Intervall $[7.9Mhz, 11{,}21Mhz]$ eingestellt werden, weil sie proportional zu der Gesamtkapazität ist (siehe Gleichung 3.1.6)**.**

3.2.1. Dimensionierung der MOS-Transistoren

Die Kondensatoren in der oben dargestellten Schaltung (Abbildung 12) werden über die PMOS-Transistoren (M0, M3) aufgeladen und über NMOS-Schalter (M1, M2) entladen. Wie in Abbildung 12 veranschaulicht, ist der NMOS-Transistor M1 den Kondensatoren (C27, C26, C25, C24, C23, C2) bzw. M2 (C14, C15, C18, C19, C20, C21) parallelgeschaltet. Die NMOS-Transistoren M1 und M2 wirken dabei wie spannungsgesteuerte Widerstände.

Die Entladungszeit der Kondensatoren ist abhängig von dem Innenwiderstand des MOSFETs und dem Kapazitätswert.

$$\boldsymbol{\tau = R_{DS,on} * C} \tag{3.1.7}$$

Wobei $\boldsymbol{R_{DS,on}}$ dem Innenwiderstand eines NMOS-Transistors und **C** der Kapazität entsprechen.

Der Kanalwiderstand des NMOS-Transistors im ohmschen Bereich lässt sich wie folgt ermitteln:

Der Zusammenhang zwischen I_D, U_{GS} , U_{DS} bei einem NMOS-Transistor, wenn $U_{GS} \geq U_{THN}$ und $U_{DS} \leq U_{DS,sat}(= U_{GS} - U_{THN})$:

$$\boldsymbol{I_D = K_p \frac{W}{L} * ((U_{GS} - U_{THN}) * U_{DS} - \frac{U_{DS}^2}{2})} \tag{3.1.8}$$

Wobei $\boldsymbol{K_p}$ einer technologiespezifischen Größe, **W** der Breite und **L** der Länge des Transistorkanals, $\boldsymbol{U_{THN}}$ der Schwellenspannung, $\boldsymbol{U_{DS}}$ der Drain-Source-Spannung und $\boldsymbol{U_{DS,sat}}$ der Drain-Source-Sättigungsspannung entsprechen.

Da die NMOS-Transistoren im Linearen Bereich ($\boldsymbol{U_{DS} \approx 0}$) betrieben werden, kann der Kanalwiderstand wie folgt beschrieben werden

$$\boldsymbol{R_{DS,on}^{-1} = \frac{dI_D}{dU_{DS}} = K_p \frac{W}{L} * (U_{GS} - U_{THN}) - K_p \frac{W}{L} * U_{DS}} \tag{3.1.9}$$

Daraus folgt,

$$\boldsymbol{R_{DS,on} = \frac{1}{K_p \frac{W}{L} * \left(U_{DS,sat} - U_{DS}\right)}} \tag{3.1.10}$$

Wenn $U_{DS,sat} \gg U_{DS}$, kann die Gleichung 3.1.10 wie folgt vereinfacht werden

$$\boldsymbol{R_{DS,on} \approx \frac{1}{K_p \frac{W}{L} * \left(U_{DS} - U_{THN}\right)}} \tag{3.1.11}$$

Anhand der Gleichung für den Kanalwiderstand des NMOS Transistors kann das $\frac{W}{L}$ Verhältnis so gewählt werden, dass der benötigte Widerstandswert eingestellt wird, um die gewünschte Entladungszeit des Kondensators zu erzielen.

Mit der folgenden Konfiguration werden alle in der Schaltung (Abbildung 12) verwendete NMOS-Transistoren dimensioniert

$$\begin{cases} \boldsymbol{L = 200\ nM} \\ \boldsymbol{W = 1\ \mu M} \end{cases} \tag{3.1.12}$$

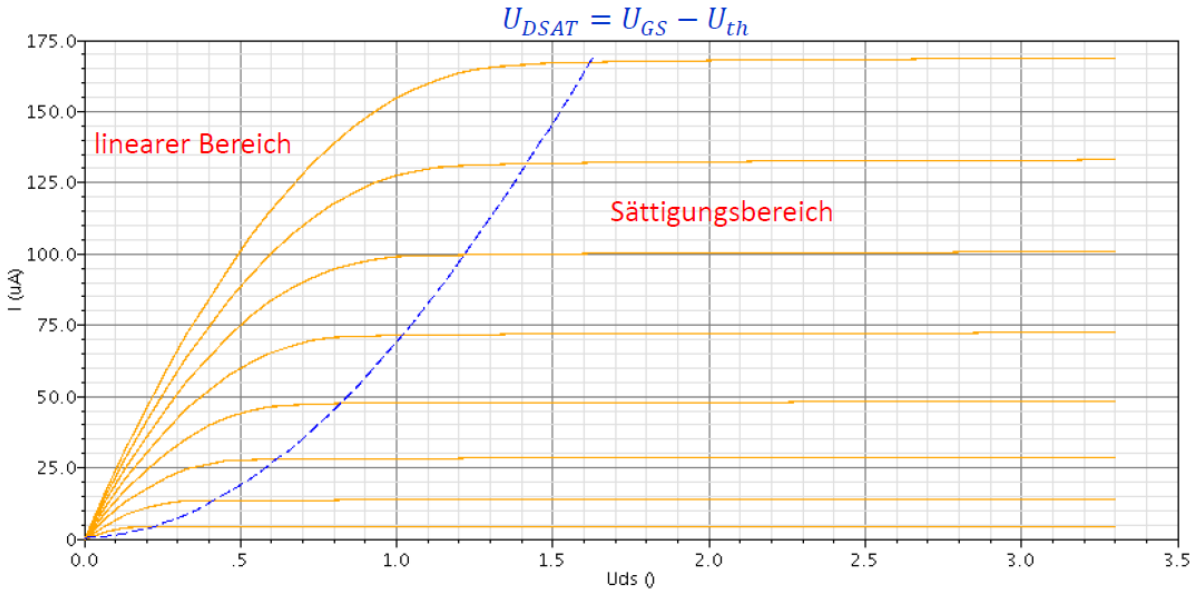

Abbildung 13: Ausgangskennlinien eines Feldeffekttransistors in starker Inversion [5]

Der obere PMOS-Transistor M0 bzw. M3 verbindet die Kondensatoren in der Schaltung mit dem Referenzstrom $\boldsymbol{I_{REF}}$, wenn der Qn- bzw. Q-Ausgang auf „0“ gesetzt ist (Abbildung 12). Zunächst arbeitet M0 bzw. M3 im Sättigungsbereich, bis der Qn- bzw. Q-Ausgang auf „1“ gesetzt ist, sodass der PMOS-Transistor in den linearen Bereich übergeht.

Der Kanalwiderstand des PMOS-Transistors ergibt sich aus den oben aufgeführten Gleichungen für NMOS-Transistoren durch Vertauschen der Indizes der einzelnen Transistorspannungen. $\boldsymbol{U_{GS}}$ und $\boldsymbol{U_{DS}}$ werden folglich bei PMOS-Transistoren durch $\boldsymbol{U_{SG}}$ und $\boldsymbol{U_{SD}}$ ersetzt. Dadurch kann die invertierte Ansteuerung des PMOS-Transistors ohne die Verwendung von Vorzeichen abgebildet werden.

Wie in der Abbildung 13 zu sehen ist, ist der Kanalstrom des MOSFETs einigermaßen konstant, wenn sich der Transistor im Sättigungsbereich befindet.

Wenn $U_{SG} \leq U_{THN}$ und $U_{SD} \geq U_{SD,sat}(= U_{SG} - U_{THP})$, ist der Zusammenhang zwischen $\boldsymbol{I_D}$ und U_{SG} bei einem PMOS Transistor in starker Inversion wie folgt definiert.

$$\boldsymbol{I_D = \frac{K_p}{2}\frac{W}{L} * (U_{SG} - U_{THP})^2 (1 + \lambda(U_{SD} - U_{SD,sat})} \qquad (3.1.13)$$

Wobei $\boldsymbol{\lambda}$ dem Kanallängenmodulationsparameter entspricht.

Unter Vernachlässigung der Kanallängenmodulationsparameter gilt für $\boldsymbol{I_D}$,

$$\boldsymbol{I_D = \frac{K_p}{2}\frac{W}{L} * (U_{SG} - U_{THP})^2} \qquad (3.1.14)$$

Allgemein gilt wegen der geringeren Ladungsträgerbeweglichkeit der Löcher, der die Stromergiebigkeit bestimmende Parameter $K_{p,PMOS} < K_{p,NMOS}$. Der Unterschied kann durch eine größere Kanalbreite W_{PMOS} für den PMOS-Transistor wieder ausgeglichen werden.

Um annähernd gleiche Stromergiebigkeit zwischen dem PMOS-Transistor und dem NMOS-Transistor zu erzielen, werden die PMOS- und NMOS-Transistoren symmetrisch dimensioniert. Dies erfolgt nach folgender Beziehung:

$$\boldsymbol{\left(\frac{W}{L}\right)_{PMOS} = \frac{K_{p,NMOS}}{K_{p,PMOS}}\left(\frac{W}{L}\right)_{NMOS}} \qquad (3.1.15)$$

Wobei

$$\frac{K_{p,NMOS}}{K_{p,PMOS}} \approx 2,5 \tag{3.1.16}$$

Dies ist auch eine Voraussetzung, damit die PMOS- und NMOS-Transistoren den gleichen Kanalwiderstand aufweisen

$$R_{DS,NMOS} = R_{DS,PMOS} \tag{3.1.17}$$

Außerdem muss die Schaltung strahlenhart sein. Dies wird durch die Verwendung von Transistoren mit dünnem Oxid sichergestellt. Desweitern werden keine Transistoren mit minimaler Länge oder Breite verwendet.

Mit der folgenden Wahl der Kanalbreite wird das Ziel erreicht.

$$\begin{cases} L = 200\ nM \\ W = 2,5\ \mu M \end{cases} \tag{3.1.18}$$

3.2.2. Entwurf und Funktionsweise des Pulsgenerators

Die Steuerung der MOS Transistoren M0, M2, M1, M3 erfolgt über den sogenannten Pulsgenerator (siehe Abbildung 14). Die Aufgabe des Pulsgenerators ist die in der Schaltung eingesetzten MOS-Transistoren so anzusteuern, dass die parallelgeschalteten Kondensatoren abwechselnd geladen und entladen werden. Im Folgenden wird die Funktionsweise dieser Struktur erläutert.

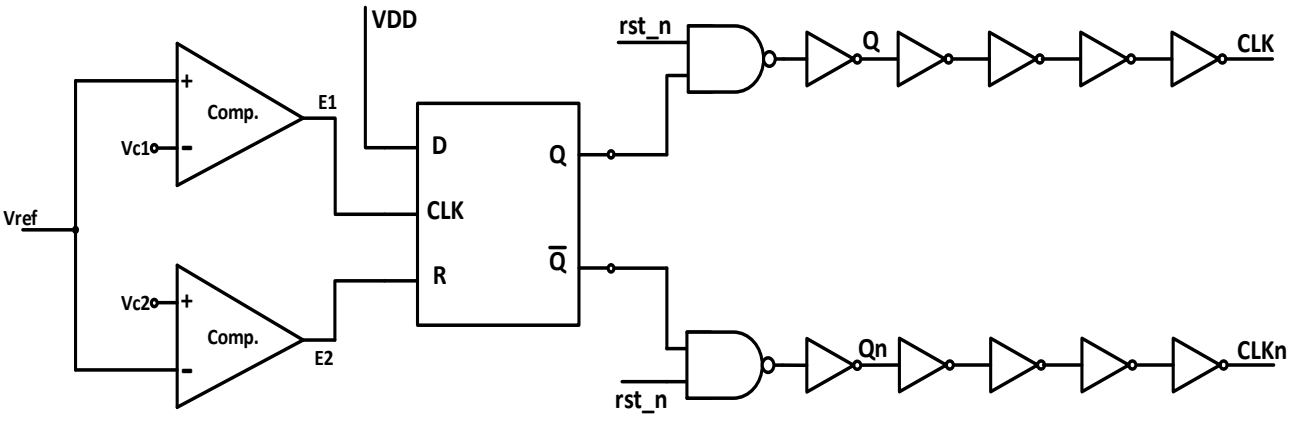

Abbildung 14: Aufbau eines Pulsgenerators

Wie in Abbildung 14 zu sehen ist, werden 2 Komparatoren für die Überwachung der Kondensatorspannungen V_{C1} und V_{C2} verwendet. Die Kondensatorspannung V_{C2} ist an den positiven Eingang des unteren Komparators und die Kondensatorspannung V_{C1} an den negativen Eingang des oberen Komparators angeschlossen. Desweitern ist eine Referenzspannung V_{ref} von 600 mV an den negativen Eingang des unteren Komparators und an den positiven Eingang des oberen Komparators angeschlossen. Der erste Komparator liefert eine logische „1“, wenn die Kondensatorspannung V_{C1} grösser als V_{ref} ist und eine logische „0“, wenn die Kondensatorspannung kleiner als V_{ref} ist. Der zweite Komparator liefert eine logische „0“, wenn die Kondensatorspannung V_{C2} grösser als V_{ref} ist und eine logische „1“ wird ausgegeben, wenn die Kondensatorspannung kleiner als V_{ref} ist (Abbildung 15).

Das D-Flipflop besitzt einen Daten- (D), einen Takt-(C) und einen Reset-Eingang. Q bzw. $\overline{Q}$ bezeichnet das Ausgangssignal bzw. das komplementäre Ausgangssignal. Solange der Takt nicht aktiv ist, wird der aktuelle Zustand gehalten.

Am D-Eingang liegt stetiges High-Signal an, was durch die Verbindung des D-Eingangs mit der Versorgungspannung erzielt wird. Der obere Komparator ist an den Takt-Eingang und der untere Komparator an den Low-aktiven Reset-Eingang des Flip-Flops angeschlossen. Das Ausgangssignal Q bzw. $\overline{Q}$ wird an ein NOR-Gatter mit nachgeschaltetem Inverter geführt. Die NOR-Gatter erhalten am zweiten Eingang das externe Reset-Signal (rst_n).

Die NOR-Funktion ist unter Verwendung der CMOS-Technologie leichter zu implementieren als die OR-Funktion. Daher wird für den Fall, dass eine OR-Funktionalität benötigt wird, dies durch ein NOR-Gatter und einen nachgeschalteten Inverter realisiert.

Die Gate-Elektroden der M3 und M1 Transistoren bzw. M0 und M2 Transistoren sind mit dem OR-Gatter Ausgang Q bzw. Qn verbunden.

Der Q- bzw. Qn-Ausgang wird bei der steigenden Flanke des oberen Komparator-Ausgangssignals auf „1“ bzw. auf „0“ gesetzt. Wenn der durch den unteren Komparator getriebene Reset-Eingang eine fallende Flanke erhält, wird der Q- bzw. Qn-Ausgang auf „0“ bzw. auf „1“ zurückgesetzt.

Während des Zeitraums, bei der die Kondensatorspannung V_{C1} zwischen 0 V und V_{ref} und die Kondensatorspannung V_{C2} bei 0 V liegen, wird vom oberen Komparator eine logische „0“ bzw. dem unteren Komparator eine logische „1“ ausgegeben. Da keine steigende Flanke auf dem CLK Eingang erscheint und das Reset Signal inaktiv ist, bleibt der Flipflop Ausgang Q auf 0 bzw. Qn auf 1. Das führt dazu, dass der Transistor M3 den Referenzstrom leitet bzw. M0 sperrt und der Transistor M1 sperrt bzw. M2 Strom leitet. Somit steigt die Kondensatorspannung V_{C1} linear an, während V_{C2} bei 0 V liegt, bis V_{C1} den Wert V_{ref} erreicht. Bei dieser Spannung gibt der obere Komparator eine logische „1“ aus, wodurch eine steigende Flanke auf dem Takt-Eingang des Flip-Flops entsteht. Mit der steigenden Flanke des Taktsignals übernimmt das Flip-Flop die logische „1“, welche am Dateneingang D anliegt und legt sie auf den Datenausgang Q.

Als Folge wird der Transistor M3 sperrend bzw. M0 leitend und der Transistor M1 leitend bzw. M2 sperrend. Die Kondensatoren, welche sich auf der rechten Hälfte der Schaltung befinden,

entladen sich, während die Kondensatoren, welche sich auf der linken Seite der Schaltung befinden, aufgeladen werden. Das führt dazu, dass die Kondensatorspannung V_{C1} auf nahezu 0V abfällt, wobei die Kondensatorspannung V_{C2} anfängt, linear anzusteigen. Der obere Komparator gibt wieder eine logische „0“ aus. Sobald V_{C2} den Spannungswert V_{ref} überschreitet, gibt der untere Komparator eine logische „0“ aus, wodurch das Reset des FlipFlops ausgelöst wird.

Das D-Flipflop reagiert darauf mit einer logischen „0“ am Ausgang Q bzw. mit einer logischen 1 am Ausgang Qn. Somit wiederholt sich der Vorgang periodisch und es entsteht auf diese Weise ein Rechtecksignal mit einer einstellbaren Schaltfrequenz $\boldsymbol{f_{osc}}$, welches die MOS-Transistoren steuert.

Das Rechtecksignal wird über eine Kette aus vier Invertern im oberen und unteren Signalpfad geführt, welche jeweils eine erhöhte Signalstärke durch die Parallelschaltung von mehreren Invertern besitzen. Durch die serielle Verschaltung der Inverter ergibt sich eine Optimierung der Leistungsaufnahme für die gewünschte Signalstärke, wobei sich auch eine Schaltverzögerung ergibt, die für die Anwendung jedoch unerheblich ist.

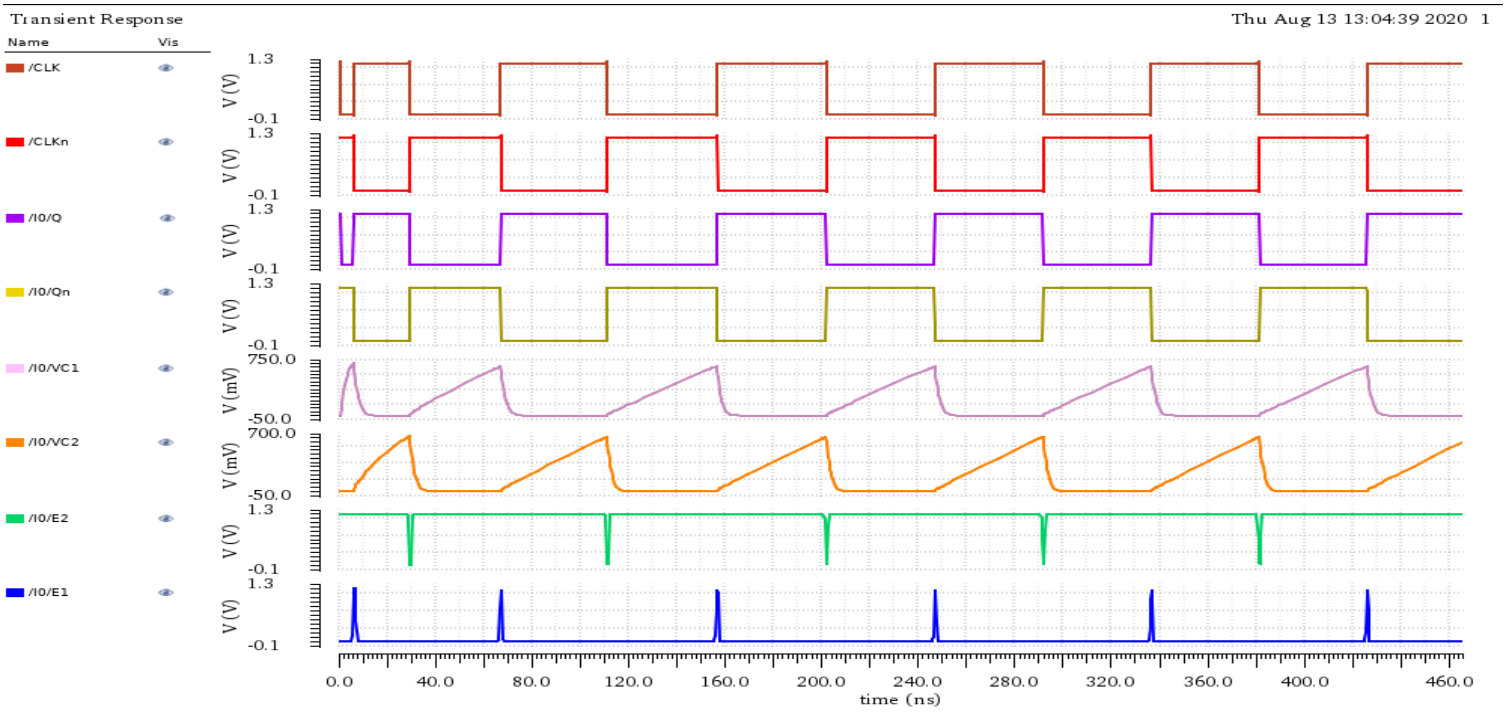

Abbildung 15: wichtige Signale des Relaxationsoszillators

Alle wichtigen Signale sind in der Abbildung 15 dargestellt.

3.2.3. Simulation des Relaxationsoszillators

Im nachfolgenden Experiment wird zuerst die benötigte Gesamtkapazität zur Einstellung einer Oszillatorfrequenz von 10 MHz rechnerisch geschätzt. Dadurch wird festgelegt, welche Steuersignale FTRIM<4:0> aktiviert werden müssen, um die angepeilte Pulsfrequenz einstellen zu können. Anschließend wird das berechnete Ergebnis durch Simulation geprüft. Die Simulation wird mit der Entwurfssoftware Virtuoso von Cadence durchgeführt.

Mit Hilfe von (3.1.6) kann die benötigte Kapazität für eine bestimmte Schaltfrequenz ermittelt werden.

$$\boldsymbol{C_{Ges} = \frac{I_{REF}}{f_{osc} V_{REF}}} \tag{3.1.19}$$

Als vorgegebene Werte zugewiesen sind der Referenzstrom $\boldsymbol{I_{REF} = 20,28\ \mu A}$, die Referenzspannung $\boldsymbol{V_{REF} = 654\ mV}$ und die gewünschte Schaltfrequenz des Oszillators $\boldsymbol{f_{osc} = 10\ MHz}$.

$$\boldsymbol{C_{Ges} = \frac{20,28\ \mu A}{10\ MHz * 654\ mV}} \tag{3.1.20}$$

Somit

$$\boldsymbol{C_{Ges} = 3,101\ pF} \tag{3.1.21}$$

Wie man in der Abbildung 12 und die dazugehörige Tabelle sieht, sind die schaltbaren Kondensatoren, welche sich sowohl in der linken als auch in der rechten Hälfte der Schaltung befinden, identisch dimensioniert. Daraus ergibt sich,

$$\boldsymbol{C_{links} = C_{rechts} = \frac{C_{Ges}}{2} = 1,5505\ pF} \tag{3.1.22}$$

Wobei $\boldsymbol{C_{links}}$ bzw. $\boldsymbol{C_{rechts}}$ der Gesamtkapazität der Kondensatoren, welche sich in der linken bzw. in der rechten Seite der Schaltung befinden, entspricht.

Die Kondensatoren, welche mit den externen Steuersignalen FTRIM<4:0> aktiviert werden, werden wie folgt bestimmt

$$C_{rechts} - C_{14} = C_{Links} - C_2 = 1,5505\ pF - 1,231pF = 319,46\ fF \quad (3.1.23)$$

Der erforderliche Kapazitätswert entspricht dann ungefähr der Summe von C_{23}, C_{24}, C_{25} bzw. C_{18}, C_{19}, C_{20}, da $C_{15} + C_{19} + C_{21} = C_{23} + C_{25} + C_{27} = 300,16\ fF \approx 319,46\ fF$

Die digitalen Steuersignale werden wie folgt aktiviert, um die 10 MHz Pulsfrequenz einzustellen

$$FTRIM<0:4> = \{0,1,1,1,0\}$$

Für die Simulation wird der Relaxationsoszillator in die in Abbildung 16 dargestellte Testbench eingebettet, welche neben einem Referenzgenerator für die Versorgung über idealen Spannungsquellen als konfigurierbare Eingangssignale für die Steuerung der Oszillatorfrequenz verfügt.

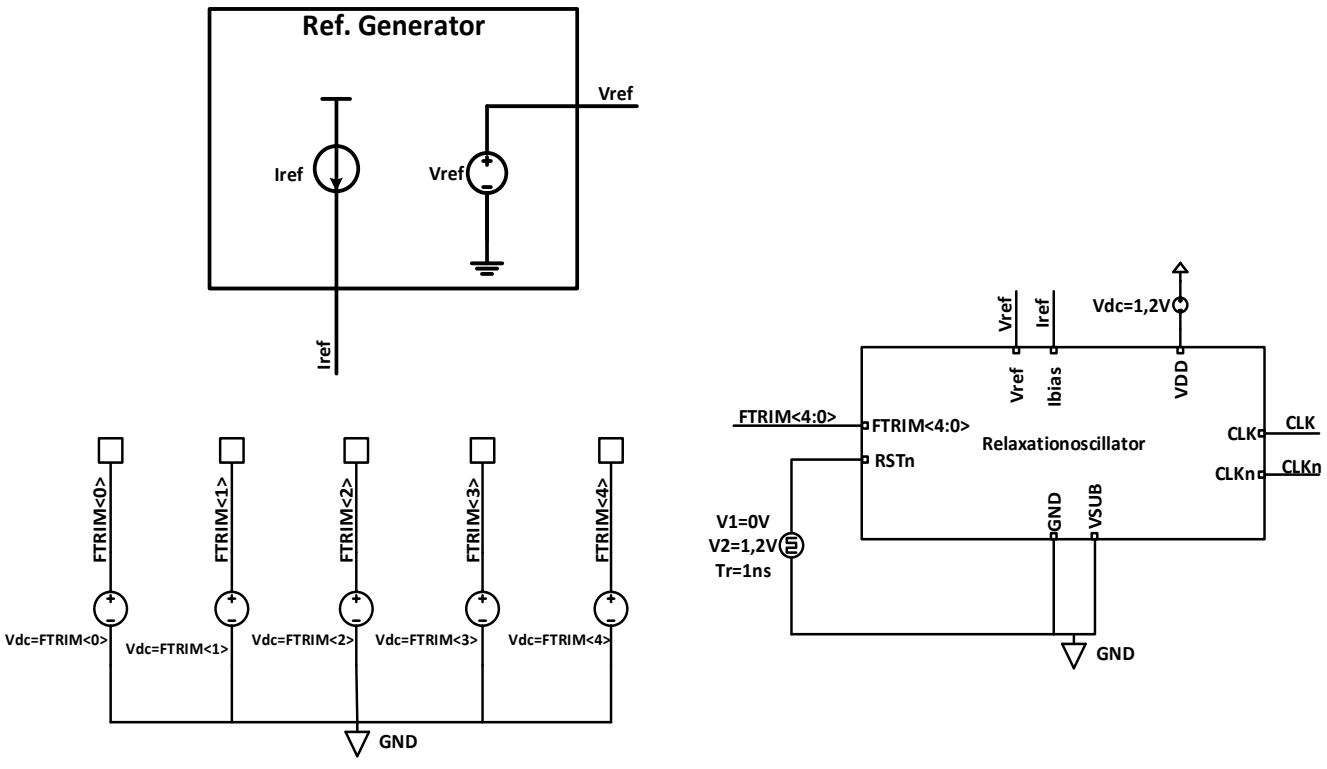

Abbildung 16: Testbench Relaxationsoszillator

In dem „ADE L Test Editor“ (Abbildung 17) wird die Simulation der wichtigen Ausgangsignale des Relaxationsoszillators konfiguriert, nämlich die Eingangs- und Ausgangsvariablen, die Art der Analyse (trans, dc…) und die Simulationszeit.

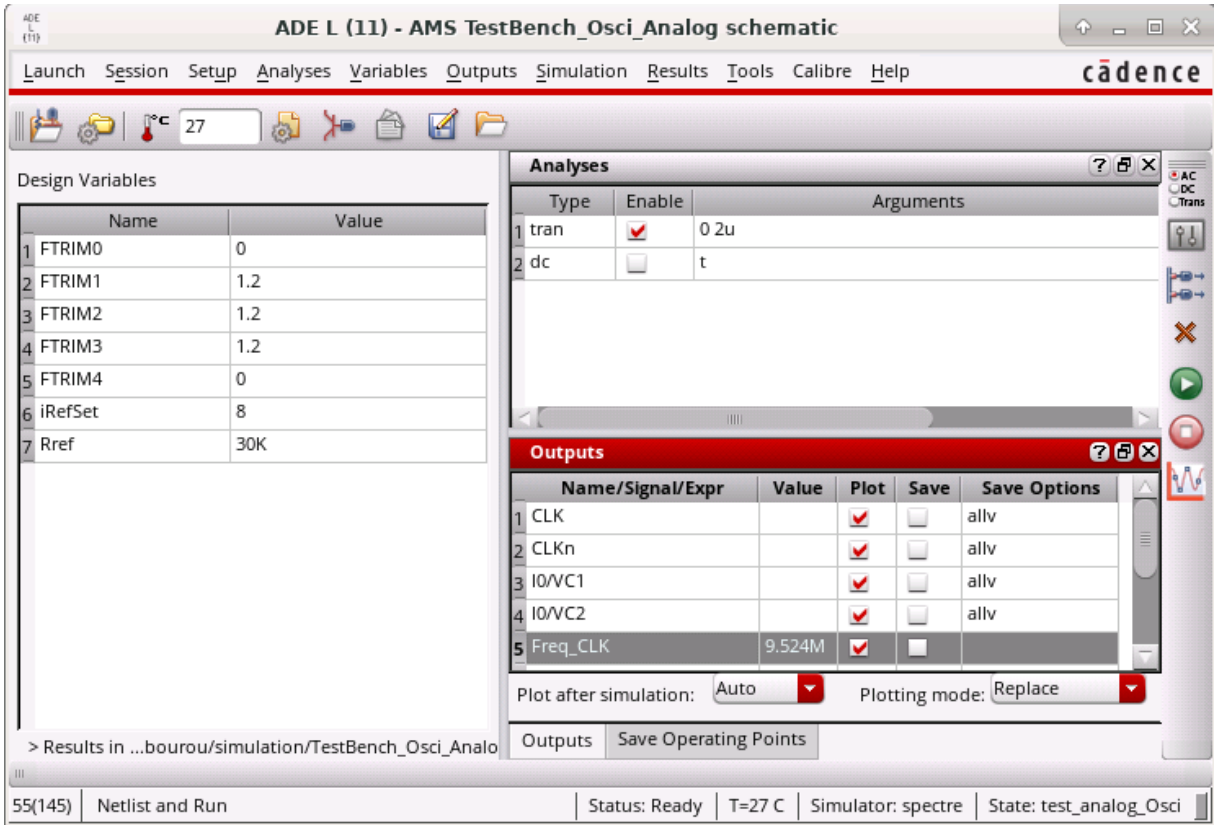

Abbildung 17: Konfigurationsfenster der Simulation in der Software Virtuoso

In der Kategorie „Design Variables“ (Abbildung 17) werden die Steuersignale in der Spalte „Value“ entweder mit „0 V“ (low-signal) oder mit „1,2V“ (high-signal) definiert. Dadurch wird eine Schaltfrequenz für den Oszillator festgelegt.

In der Kategorie „Analyses“ wird die Simulationsart „Transient“ selektiert und die Simulationszeit definiert, um die Spannungsverläufe der Ausgangssignale in einem Diagramm darzustellen.

In der Kategorie „Outputs“ werden die Signale, die man darstellen bzw. simulieren möchte, ausgewählt, in unserem Fall „CLK“, „CLKn“, „VC1“ und „VC2“. Wobei CLK und CLKn die Ausgangssignale des Relaxationsoszillator und VC1 bzw. VC2 die Ausgangsspannung der parallelgeschalteten Kondensatoren, welche sich in der rechten bzw. in der linken Hälfte der Schaltung befinden, entsprechen.

Die Simulation verläuft für eine konstante Temperatur von 27°C.

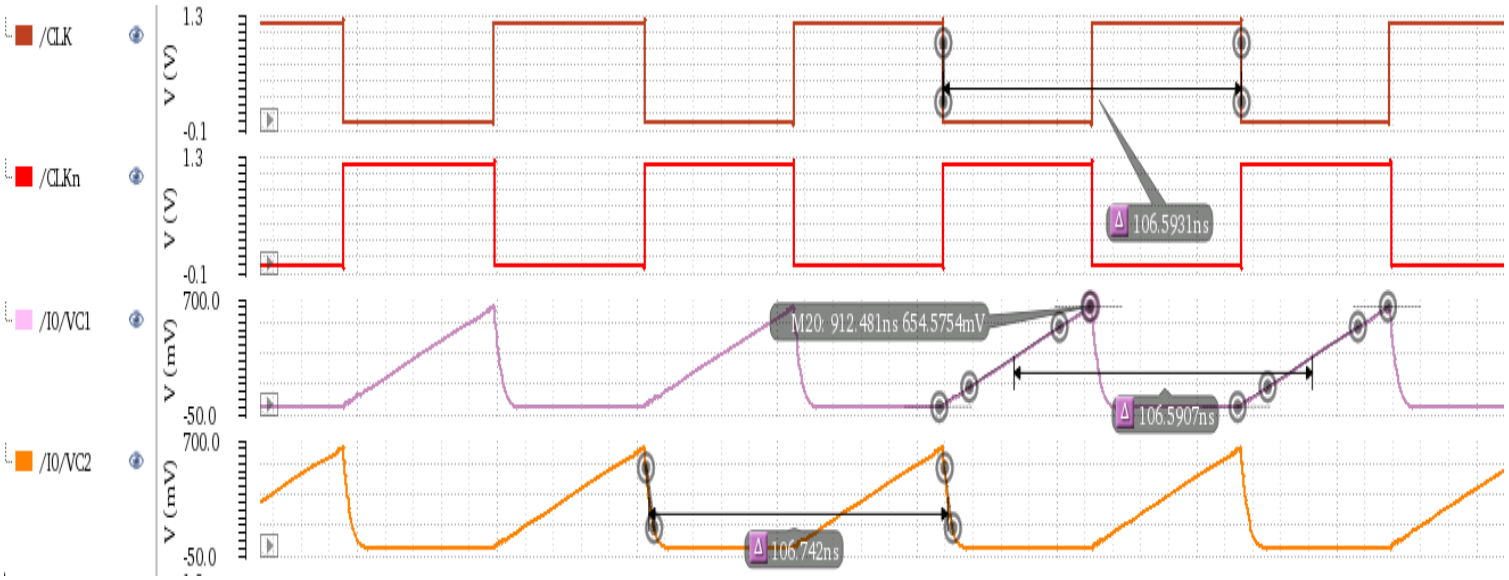

Abbildung 18: CLK-, CLKn-, VC1-, VC2-Signale mit einer Schaltfrequenz von 10 MHz

In der Abbildung 18 sind die Simulationsergebnisse für die Oszillatorschaltung dargestellt. Die simulierte Periodendauer des CLK-Signals liegt ungefähr bei 106,59 ns, welche circa 9,5 MHz entspricht. Da die Sägezahnspannungen $\boldsymbol{V_{C1}}$ und $\boldsymbol{V_{C2}}$ bei der Aufladung über dem Referenzspannung $\boldsymbol{V_{REF}}$ steigen, entstehen Abweichungen von der berechneten Schaltfrequenz bei der Simulation.

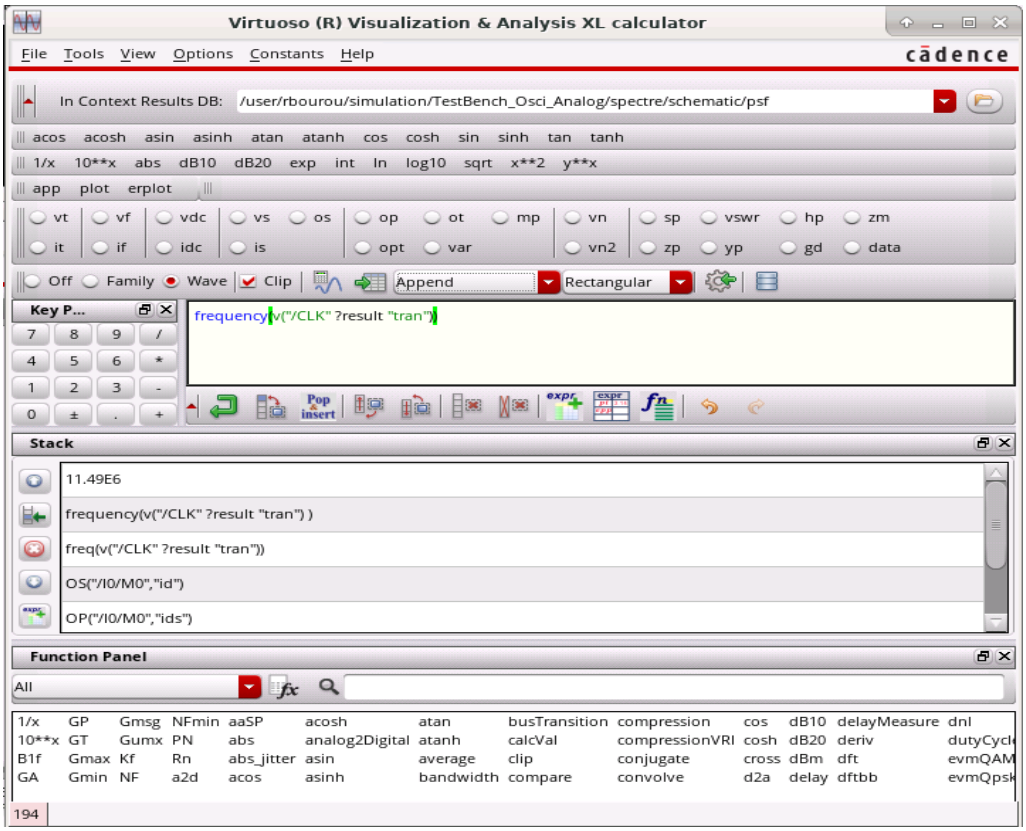

Abbildung 19: Calculatorfenster

Eine weitere Möglichkeit, die Schaltfrequenz des Oszillators zu simulieren, bildet der so genannte Calculator. Dieses Tool lässt sich im ADE-L-Fenster unter Tools > Calculator öffnen. Der Calculator bietet weitreichende Möglichkeiten zur numerischen Analyse und Auswertung der Simulationsergebnisse anhand von speziellen Funktionen und generischen mathematischen Ausdrücken (Abbildung 19).

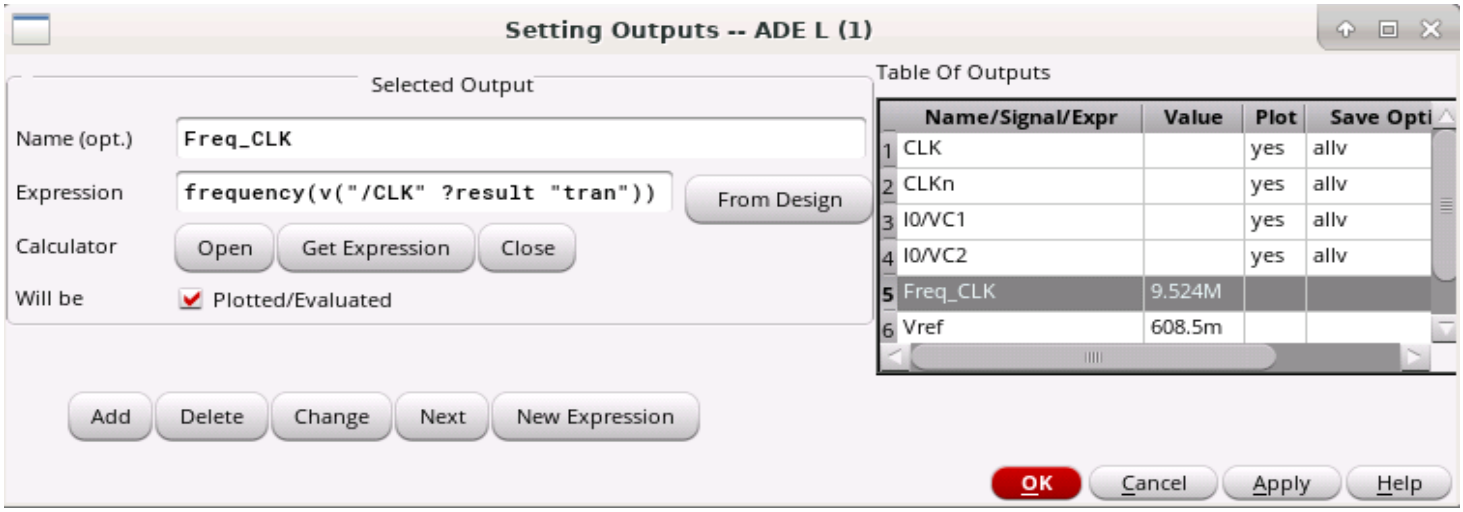

Abbildung 20: Einstellung der Ausgänge im ADE-L Simulationskonfigurator

Mit Hilfe des Calculators wird der folgende Ausdruck „frequency (v ("/CLK"? result "tran"))“ als Ausgangswert im "Outputs" definiert, um aus der Transienten Simulation des CLK-Signals automatisiert die Frequenz zu extrahieren (Abbildung 20).

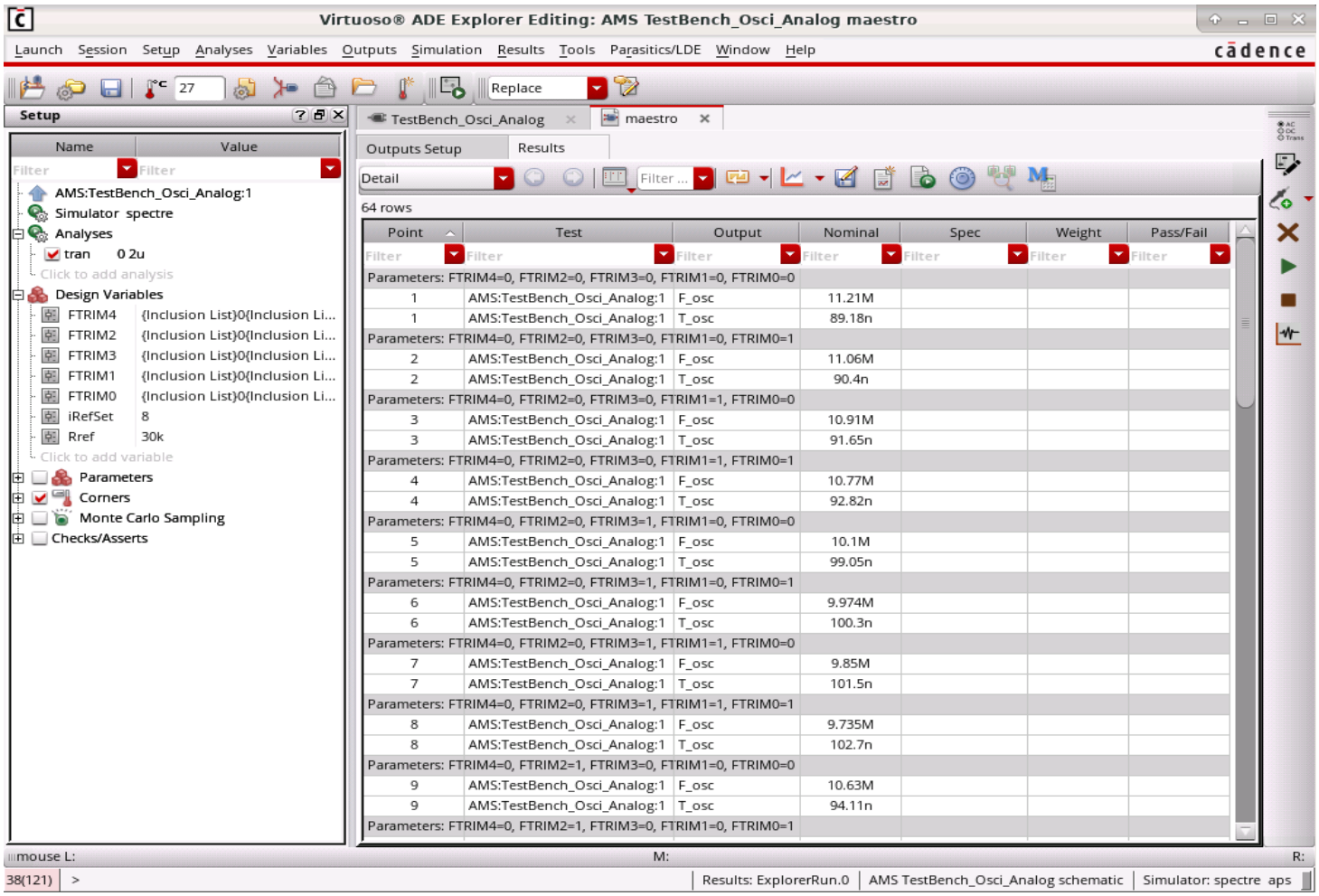

Abbildung 21: Virtuoso ADE Explorer

Im ADE Explorer hat man die Möglichkeit, Sweeps einzurichten. Im Design Variables-Fensterbereich können die im Schematic festgelegten Variablen importiert und einen Wertebereich für diese Variablen definiert werden. Dies erfolgt, in dem man doppelt auf den Wert der Variable klickt. Anschließend öffnet sich das Parametrize-Fenster durch einen weiteren Klick auf die Schaltpläne mit drei Punkten (Abbildung 22). Desweitern wird die Voreinstellung mit einem Klick auf Delete Spec gelöscht und abschließend wird eine neue From/To-Parametrisierung angelegt.

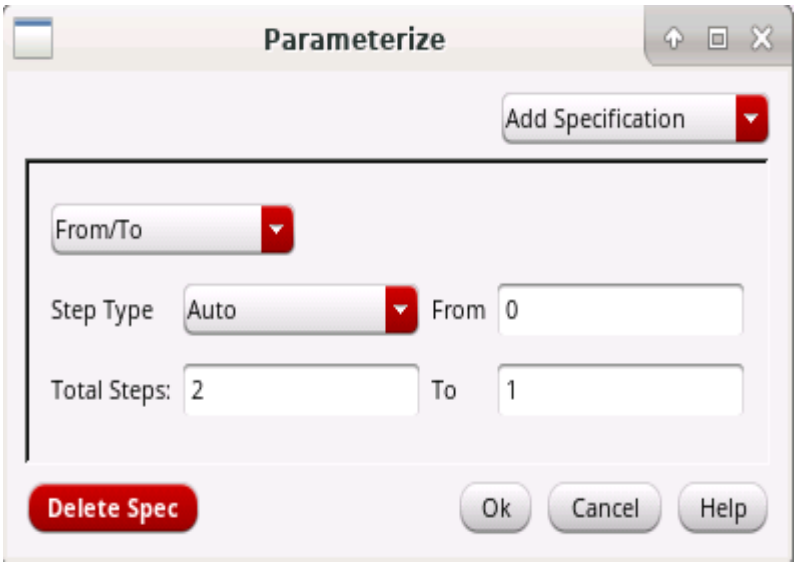

Abbildung 22: Einstellen eines Sweeps mit 2 Steps der Design Variable FTRIM0

Da das FTRIM-Datenwort eine Länge von 5 Bits hat, ist die Anzahl der möglichen Werte $2^{Anzahl\ der\ Bits} = 2^5 = 32$. Das ermöglicht 32 Zustände und damit einen Wertebereich für dezimale Ganzzahlen von 0 bis 31. Es ist dann zu erwarten, dass die ADE-Simulationsumgebung 32 Sweep Points erzeugt und für jeden Sweep Point Schaltfrequenz und Periodendauer des CLK-Signals simuliert werden (Abbildung 21).

Unter dem Reiter Results werden nach Abschluss der Simulation die Ergebnisse des Simulationsdurchlaufes gelistet.

Die nachfolgende Tabelle zeigt alle mögliche Werte der Schaltfrequenz und Periodendauer, welche vom digital gesteuerten Relaxationsoszillator eingestellt werden können, in Abhängigkeit von Steuersignalen FTRIM<4:0>.

FTRIM<4>	FTRIM<3>	FTRIM<2>	FTRIM<1>	FTRIM<0>	Dezimal	T_osc(ns)	Schritt T_osc(ns)	F_osc(MHz)	Schritt F_osc(MHz)
0	0	0	0	0	0	89,19		11,21	
0	0	0	0	1	1	90,4	1,21	11,06	0,15
0	0	0	1	0	2	91,65	1,25	10,91	0,15
0	0	0	1	1	3	92,82	1,17	10,77	0,14
0	0	1	0	0	4	94,11	1,29	10,63	0,14
0	0	1	0	1	5	95,33	1,22	10,49	0,14
0	0	1	1	0	6	96,57	1,24	10,35	0,14
0	0	1	1	1	7	97,75	1,18	10,23	0,12
0	1	0	0	0	8	99,07	1,32	10,09	0,14
0	1	0	0	1	9	100,3	1,23	9,972	0,118
0	1	0	1	0	10	101,5	1,20	9,85	0,122
0	1	0	1	1	11	102,7	1,20	9,734	0,116
0	1	1	0	0	12	104	1,30	9,615	0,119
0	1	1	0	1	13	105,2	1,20	9,506	0,109
0	1	1	1	0	14	106,4	1,20	9,395	0,111
0	1	1	1	1	15	107,7	1,30	9,289	0,106
1	0	0	0	0	16	108,9	1,20	9,18	0,109
1	0	0	0	1	17	110,1	1,20	9,079	0,101
1	0	0	1	0	18	111,4	1,30	8,978	0,101
1	0	0	1	1	19	112,6	1,20	8,883	0,095
1	0	1	0	0	20	113,9	1,30	8,783	0,1
1	0	1	0	1	21	115,1	1,20	8,691	0,092
1	0	1	1	0	22	116,3	1,20	8,598	0,093
1	0	1	1	1	23	117,5	1,20	8,511	0,087
1	1	0	0	0	24	118,8	1,30	8,418	0,093
1	1	0	0	1	25	120	1,20	8,333	0,085
1	1	0	1	0	26	121,2	1,20	8,248	0,085
1	1	0	1	1	27	122,4	1,20	8,167	0,081
1	1	1	0	0	28	123,7	1,30	8,083	0,084
1	1	1	0	1	29	124,9	1,20	8,006	0,077
1	1	1	1	0	30	126,1	1,20	7,928	0,078
1	1	1	1	1	31	127,3	1,20	7,853	0,075

Tabelle 2: Simulationsergebnisse

Der Maximalwert der Schaltfrequenz wird erreicht, wenn der binär kodierte Wert „00000“ dem Register FTRIM<4:0> zugewiesen wird. Hingegen wird der Minimalwert der Schaltfrequenz erreicht, wenn der binär kodierte Wert „11111“ dem Register FTRIM<4:0> zugeordnet wird. Immer wenn sich der Wert für FTRIM<4:0> um 1 erhöht, erhöht sich die Periodendauer um 1.23 ns bzw. vermindert sich die Schaltfrequenz um durchschnittlich 0.1 MHz.

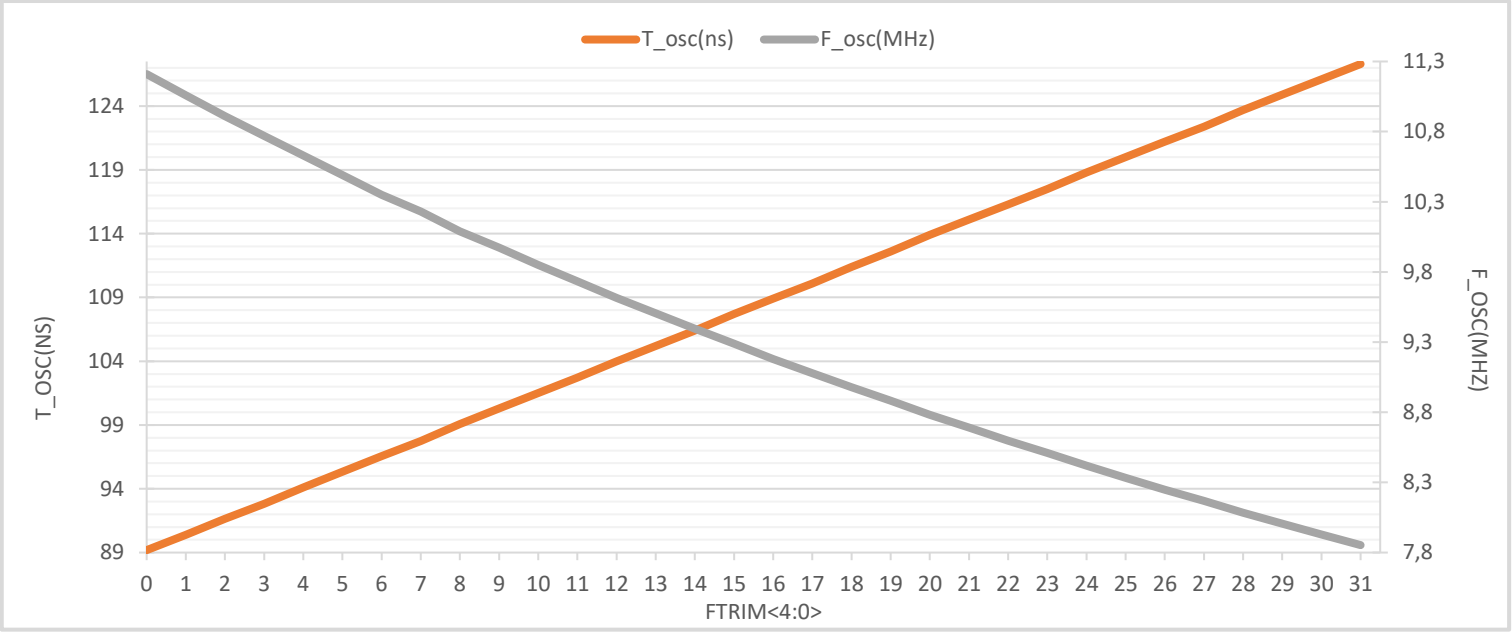

Abbildung 23: Periodendauer und Taktfrequenz des digital gesteuerten Relaxationsoszillators

Wie die Abbildung 23 veranschaulicht, stellt die Taktperiode abhängig von den Steuersignalen FTRIM<4:0> der Graf einer linearen Funktion dar und kann somit wie folgt mathematisch ausgedrückt werden:

$$\boldsymbol{T_{osc}(k) = (1.23 * k + 89.19)[ns]} \qquad (3.1.24)$$

Wobei $k \in \{0,1,2, \dots ,31\}$

Diese lineare Beziehung zwischen der Taktperiode und den Steuersignalen FTRIM<4:0> wird als Grundlage für den Entwurf des Regelsystems genommen, welcher im Kapitel 5 vorgestellt und detailliert beschrieben wird.

4. Analog Mixed Signal Simulation

Wie im zweiten Kapitel bereits erläutert wurde, besteht die Bittiming Logik aus den Mechanismen der harten und weichen Synchronisierung. Der im vorherigen Kapitel vorgestellte Relaxationsoszillator wird als Taktgeber für den CAN-Controller verwendet.

In diesem Kapitel wird eine Verifikation Anhand einer A/MS[11]-Simulation durchgeführt, um sicherzustellen, dass die oben genannten Prozesse regelkonform ausgeführt werden.

Als nächstes werden Die Schritte zur Einrichtung einer A/MS-Simulation in der Software Cadence Virtuoso beschrieben.

4.1. Softwareanforderungen

Der A/MS Designer-Simulator ist ein Mixed-Signal-Simulator, der den Sprachstandard Verilog-AMS und drei analoge Solver, Spectre, APS und UltraSim, unterstützt. Er bietet Designern die Möglichkeit, die Simulation an die Merkmale ihrer Entwürfe anzupassen. Der Spectre-Solver bietet eine bewährte, hochgenaue Simulation und der UltraSim-Solver eine schnelle Simulation, welche sich für Designs mit einer großen Anzahl von Bauelementen wie Transistoren, Widerständen und Kondensatoren eignet. Der APS-Solver bietet die schnellste Simulationsgeschwindigkeit für Designs mit mehr als 50K Elementen in SPICE-Blöcken.

Zur Ausführung der AMS Simulation werden die folgenden Softwarepakete benötigt:

▪ XCELIUM 18.09 (oder höher) für den AMS-Simulator

▪ IC6.1.8 ISR3 (oder höher) für ADE[12] Explorer

Um AMS ausführen zu können, muss die XCELIUM-Software installiert sein und die analogen Simulator-Engines (Spectre, APS und Ultrasim) enthalten. AMS ist ein einzelner ausführbarer Simulator und kein Co-Simulator. Daher werden die Spectre-, APS- und Ultrasim-Solver in den AMS-Simulator integriert und gemeinsam kompiliert.

Die MMSIM / SPECTRE-Installation enthält hingegen die eigenständigen ausführbaren Dateien für Spectre, APS und Ultrasim. Es wird für die analoge Simulation über ADE[4] Explorer Tool verwendet.

- Die MMSIM-Lizenz gilt für AMS, Spectre, APS, SpectreRF und Ultrasim.
- Die **MMSIM / SPECTRE**-Installation wird für die analoge Simulation verwendet.
- Die **XCELIUM**-Installation wird von **AMS** Simulator für die Mixed-Signal-Simulation verwendet.

Da es sich bei Cadence-Software um eine Linux-Anwendung handelt, werden mit der Eingabe des Befehls **module avail** in das Terminal alle auf dem Server installierte Umgebungsmodule

[11] Analog Mixed Signal
[12] Analog Design Environment

dargestellt, um sicherzustellen, dass die benötigten Softwarepakete für die Simulation zur Verfügung stehen (Abbildung 24).

```
[rbourou@hoerni ~]$ cd cadence/tsmc65
[rbourou@hoerni tsmc65]$ module avail

------------------------ /usr/share/Modules/modulefiles ------------------------
dot         module-git  module-info modules     null        use.own

---------------------------- /eda/local/modulefiles ----------------------------
Cadence/2015_2016                       MathWorks/Matlab/R2020a
Cadence/2016_2017                       mem-tools
Cadence/2016_2017_PVS                   Mentor/CALIBRE/2015.4
Cadence/2016_2017_rc_edi                Mentor/CALIBRE/2016.4
Cadence/2017_2018                       Mentor/CALIBRE/2018.4
Cadence/2018_2019                       Mentor/CALIBRE/2018.4.centos7
Cadence/2019_2020                       Mentor/HDL-Designer/2016.1
Cadence/ASSURA/4.14_616                 Mentor/HDL-Designer/2018.1
Cadence/ASSURA/4.15_107                 Mentor/HDL-Designer/2019.4
Cadence/ASSURA/4.15_111                 Mentor/HyperLynx-SI/9.4.2U1
Cadence/ASSURA/4.15.115                 Mentor/LEONARDO/2019a
Cadence/ASSURA/4.16.103                 Mentor/OASYS/2019.1
Cadence/CONFORMAL/18.10_200             Mentor/PRECISION/2019.1.1
Cadence/CONFORMAL/19.10.300             Mentor/QUESTA/2019.4
Cadence/EDI/14.26                       Mentor/QUESTA/questa10.5c-4
Cadence/EDI/14.28                       Mentor/QUESTA/questa10.7c
Cadence/EXT/15.26                       OpenOCD/0.10.0-14.3
Cadence/EXT/17.12                       Sage/7.6
Cadence/EXT/18.12                       Synopsys/2016_2017
Cadence/EXT/19.11                       Synopsys/2017_2018
Cadence/EXT/19.13.000                   Synopsys/2019_2020
Cadence/GENUS/16.12_000                 Synopsys/CUSTOM-COMP/2016.06
Cadence/GENUS/17.11_000                 Synopsys/CUSTOM-COMP/2017.12
Cadence/GENUS/18.10_000                 Synopsys/CUSTOM-COMP/2019.06
Cadence/GENUS/19.11.000                 Synopsys/CUSTOM-SIM/2016.06
Cadence/IC/6.1.6_130                    Synopsys/CUSTOM-SIM/2017.12
Cadence/IC/6.1.7_704                    Synopsys/CUSTOM-SIM/2018.09
Cadence/IC/6.1.7_715                    Synopsys/CUSTOM-SIM/2019.06
Cadence/IC/6.1.8_000                    Synopsys/CUSTOM-WV/2016.06
Cadence/IC/6.1.8.060                    Synopsys/CUSTOM-WV/2017.12
Cadence/INCISIVE/15.10_002              Synopsys/CUSTOM-WV/2018.09
Cadence/INCISIVE/15.20_010              Synopsys/CUSTOM-WV/2019.06
Cadence/INCISIVE/15.20_038              Synopsys/EMBEDIT/2020.06-SP1
Cadence/INCISIVE/15.20_058              Synopsys/FINESIM/2016.06
Cadence/INNOVUS/16.13_000               Synopsys/FINESIM/2017.12
Cadence/INNOVUS/17.11_000               Synopsys/FINESIM/2019.06
Cadence/INNOVUS/18.10_000               Synopsys/FM/2019.12
Cadence/INNOVUS/19.11_000               Synopsys/HSIMPLUS/2016.06
Cadence/INNOVUS/19.11.000               Synopsys/HSIMPLUS/2017.12
Cadence/JLS/19.11.000                   Synopsys/HSIMPLUS/2019.06
Cadence/LIBERATE/14.14                  Synopsys/HSPICE/2016.06
Cadence/LIBERATE/15.14_070              Synopsys/HSPICE/2017.12
Cadence/LIBERATE/16.14_122              Synopsys/HSPICE/2019.06
Cadence/LIBERATE/18.10_293              Synopsys/ICC/2019.12
Cadence/LIBERATE/19.20.100              Synopsys/ICC2/2019.12
Cadence/MMSIM/14.10_765                 Synopsys/ICV/2019.12
Cadence/MMSIM/15.10_627                 Synopsys/installer/51
Cadence/PVS/15.17.000                   Synopsys/LC/2019.12
Cadence/PVS/16.11.000                   Synopsys/MW/2019.12
Cadence/PVS/16.12.000                   Synopsys/PT/2019.12
Cadence/PVS/19.10.000                   Synopsys/SABER/2019.06
Cadence/RC/14.25                        Synopsys/SFPGA/2019.09
Cadence/RC/14.27                        Synopsys/SPYGLASS/2019.06
Cadence/SPECTRE/16.10_510               Synopsys/STARRC/2019.12
Cadence/SPECTRE/18.10_077               Synopsys/SYN/2019.12
Cadence/SPECTRE/19.10.063               Synopsys/VCS/2019.06
Cadence/SSV/16.13                       TowerJazz/ts18is
Cadence/SSV/17.11                       triplication/CheckTMR
Cadence/SSV/18.10.000                   triplication/TMRG/2018
Cadence/SSV/19.11.000                   TSMC/CERN65nm
Cadence/VIPCAT/11.30.051                TSMC/CERN65nm2016_17
Cadence/VIPCAT/11.30.057                TSMC/CERN65nm2017_18
Cadence/XCELIUM/19.03.013               TSMC/CERN65nm2017_18_qrc
Cliosoft/SOS/6.32.p7                    TSMC/CERN65nm2017_18_qrc2
```

Abbildung 24: Umgebungsmodule

Wie man in der Abbildung 24 sieht, sind die benötigten Umgebungspakete verfügbar und stehen zur Anwendung bereit.

4.2. Implementierung der Bittiming Logik in Virtuoso

Im Folgenden wird die Implementierung der Bittiming-Einheit in der Entwurfssoftware Virtuoso schrittweise erläutert und beschrieben.

Als Erstes wird das benötigte Umgebungspacket mit der folgenden Anweisung geladen.

- **module load TSMC/RD53**

Anschließend kann die Entwurfssoftware Virtuoso von Cadence mit der folgenden Befehlseingabe im Terminal gestartet werden

- **virtuoso&**

In Folge öffnet sich die Virtuoso-Konsole. Über das Menü **Tools\Library Manager** kann der Library Manager geöffnet werden. Im Library Manager lassen sich unter anderem Bibliotheken verwalten, in denen Schaltungsentwürfe und Bauelemente organisiert sind.

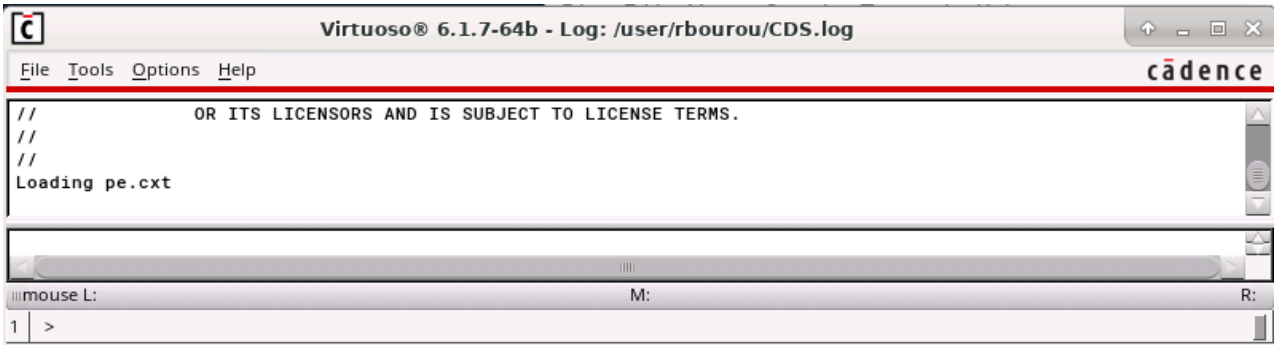

Abbildung 25: Das Virtuoso-Fenster

Als Nächstes wird eine neue Bibliothek erstellt, in welcher der Relaxationsoszillator-Schaltungsentwurf, die CAN-Bittiming-Verilog-Module und eine Testbench für die Simulation angelegt werden.

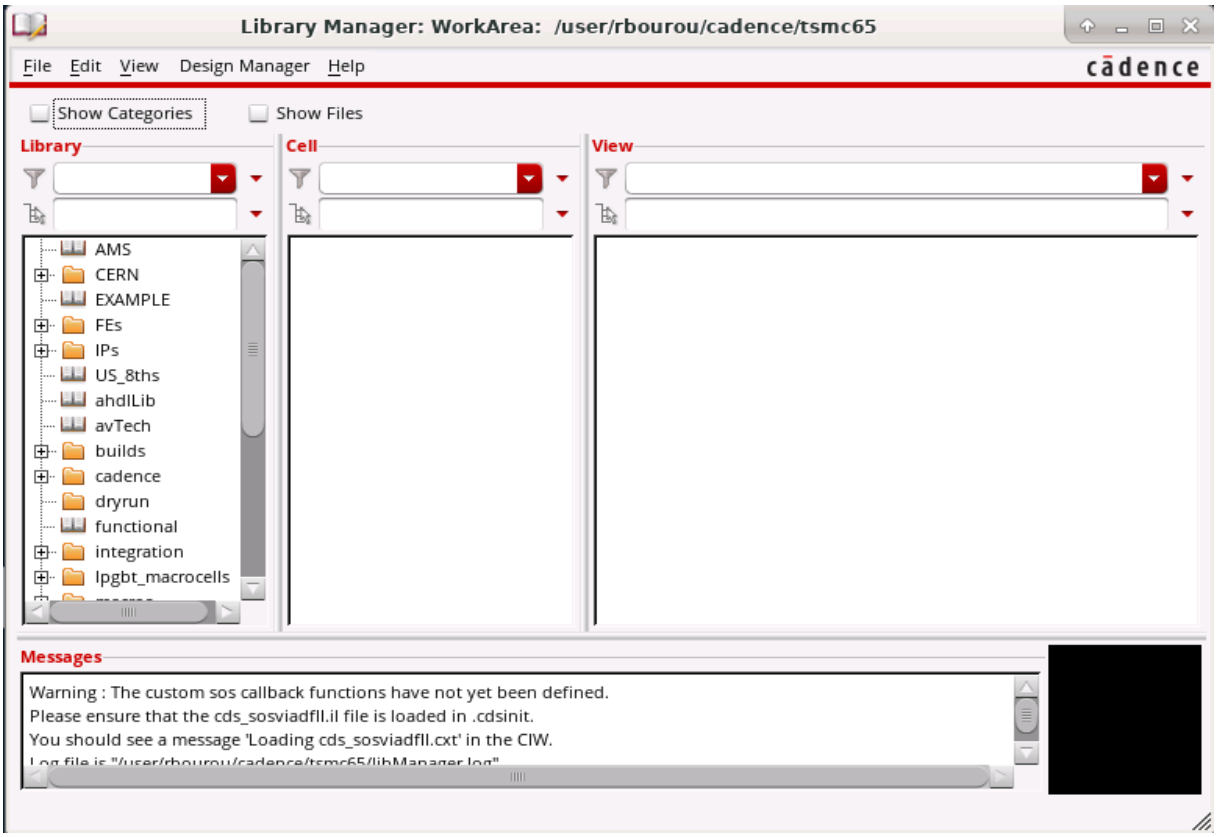

Abbildung 26: Der Library Manager

Im Library Manager kann eine neue Bibliothek unter **File\New\Library** erstellt werden. Bei der Erstellung einer neuen Bibliothek ist zu beachten, dass die Anbindung an die verwendete Technologiebibliothek, in unserem Fall **TSMC 65nm**, festgelegt ist.

Anschließend können die neuen Zellen (Cell View) erstellt werden, welche für die AMS Simulation erforderlich sind. Eine Zelle wird im Library Manager unter **File\New\Cell View** angelegt. Im Dialogfenster New File können verschiedene Vorgaben für das zu öffnende Editor-Fenster getroffen werden (Abbildung 27).

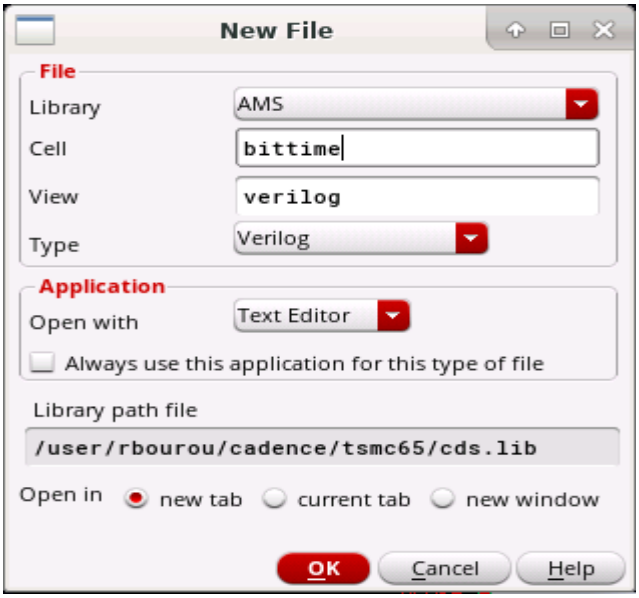

Abbildung 27: Erstellung einer neuen Zelle

In diesem Fenster ist die neu zu erstellende Bibliothek auszuwählen sowie ein Name für die Zelle, die entworfen wird, und die Ansicht (View) einzugeben. Da sich das Verilog Design der Bittiming-Einheit in Module aufteilen lässt, ist für jedes Modul eine Zelle mit der Ansicht „verilog“ zu generieren, wobei als Anwendung „Text Editor“ auszuwählen ist. Durch die Bestätigung der Eingaben mit Ok öffnet sich das Editor-Fenster.

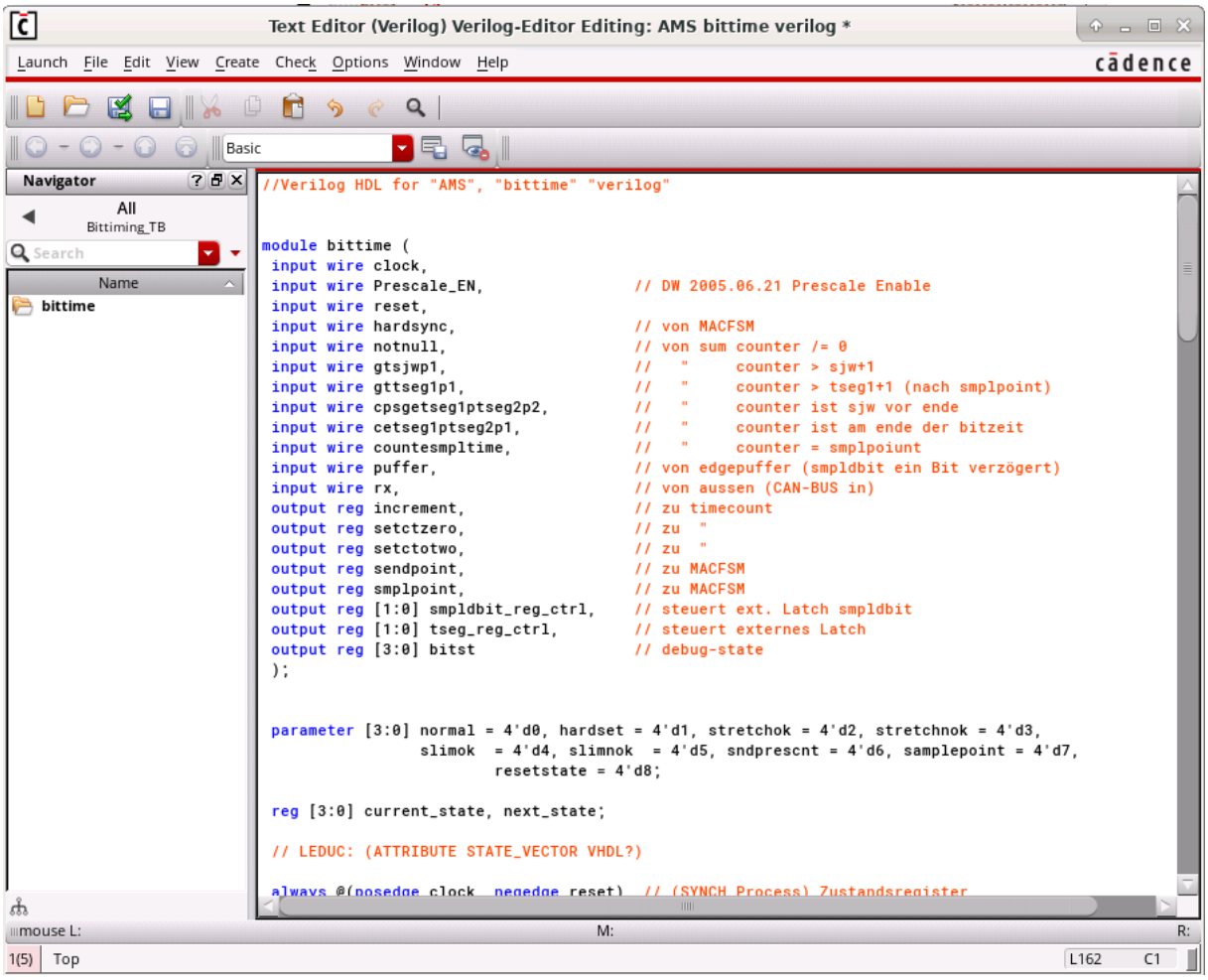

Abbildung 28: Verilog-Editor

Im Editor-Fenster erfolgt die Implementierung der im vorherigen Kapitel beschriebenen FSM Zustandmaschine in **Verilog HDL**[13] (Abbildung 28). Mit einem Klick auf das Save Symbol im oberen Fensterbereich oder über den Menüpunkt **File\Extract** wird eine Datenbank, welche die im Skript befindlichen Instanzen Netze und Pins enthält, aus dem eingegebenen Verilog-Code erzeugt. Sofern die Erstellung der Datenbank erfolgreich ist, erscheint das „**Cellview symbol**“ Fenster (Abbildung 29) mit der Abfrage, ob ein Symbol für die Zelle „**bittime**“ erstellt werden soll.

[13] Hardware Description Language

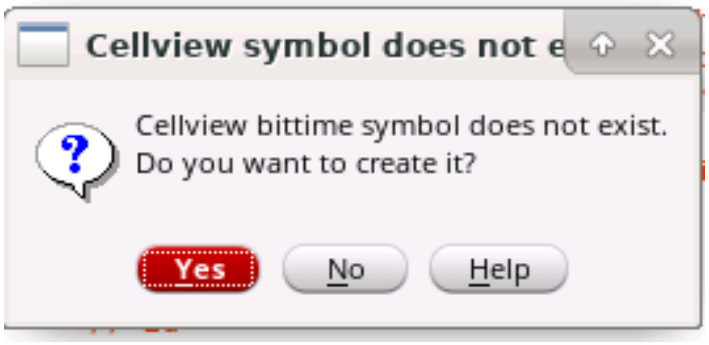

Abbildung 29: Bestätigung der Erstellung eines Symbols

Mit einem Klick auf **Yes** wird das Symbol automatisch generiert und im Symbol Editor zur weiteren Bearbeitung geöffnet (Abbildung 30)

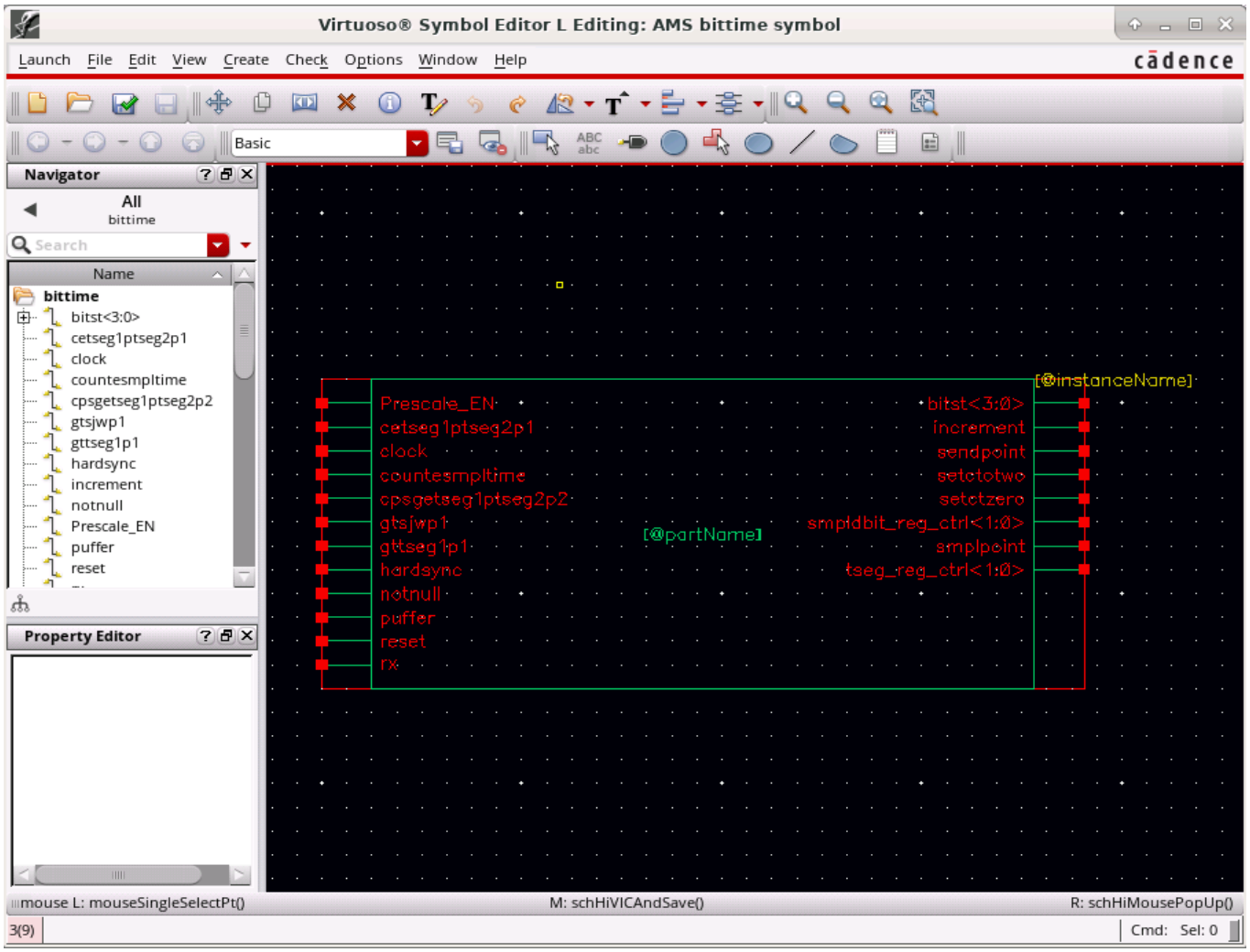

Abbildung 30: Bearbeiten eins neu generierten Symbols

Wie in der Abbildung 30 zu erkennen ist, wird das Symbol mit einer grünen Umrandung und kleinen roten quadratischen Punkten, welche die zuvor definierte Pinbelegung kennzeichnen, dargestellt. Die Textfelder [@InstanceName] und [@PartName] sind Anzeigevariablen, welche den Instanz- bzw. den Zellnamen im Schematic bezeichnen. Abschließend wird das Symbol durch einen Klick auf „**Check and Save**“ gespeichert.

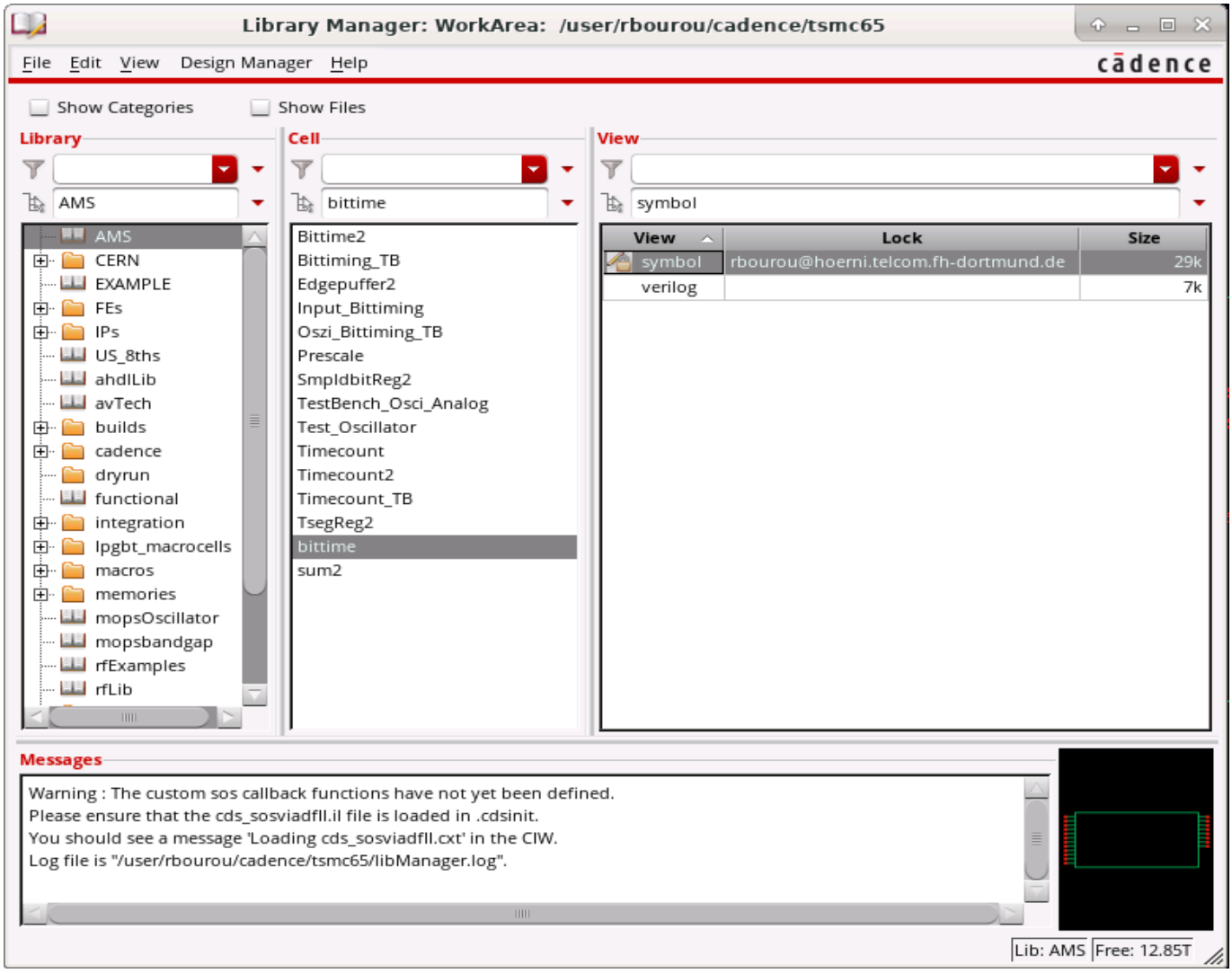

Abbildung 31: das erstellte Symbol im Library Manager

Das erstellte Symbol kann in einem anderen Schematic verwendet werden. In diesem Fall muss über Create Instance im Library Browser in der Bibliothek AMS und der Zelle **bittime** das View Symbol selektiert werden (Abbildung 31).

Auf diese Weise werden die einzelnen Zellen und die zugehörigen Symbole, welche die Bittiming Einheit beschreiben, erstellt und in der Bibliothek **AMS** gespeichert. Danach werden sie in einem neuen Schematic zusammengebracht und verbunden (Abbildung 32).

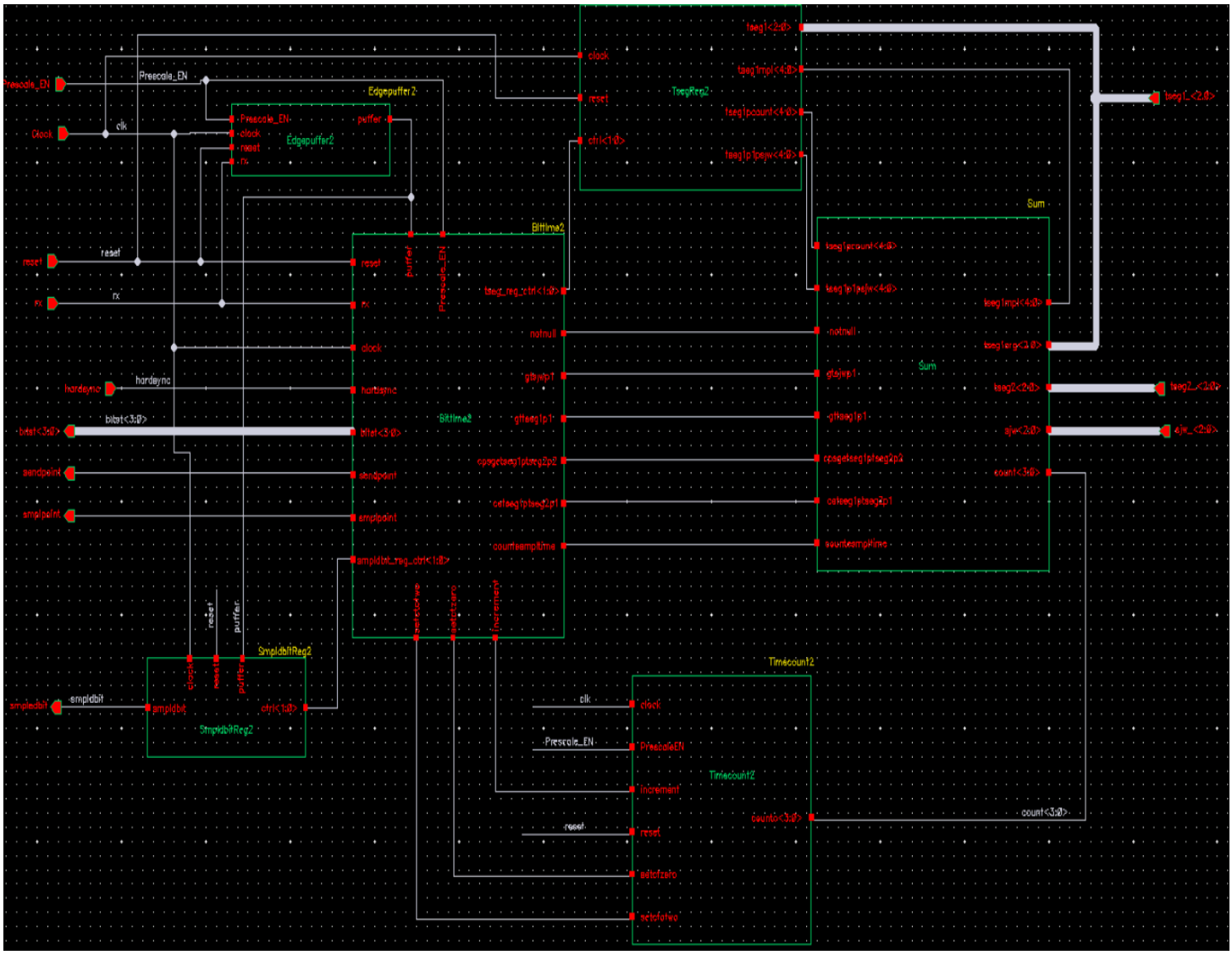

Abbildung 32: Schematische Ansicht der Bittiming Logik

Wie in Abbildung 32 veranschaulicht, setzt sich die Bittiming-Logik-Zelle wiederum aus nachfolgenden Unterzellen zusammen:

- **Bittime** stellt die Statemachine des Bittiming-Moduls dar.
- **TsegReg** stellt das Propagationssegment- und Phase-Buffer-1-Segment-Register des Bitsegments dar.
- **Timecount** stellt den Zähler des Bittiming-Moduls dar.
- **SmpldbitReg** stellt der Smpldbit Register dar.
- **Sum** stellt die Arithmetische Einheit des Bittiming-Moduls dar.
- **Edgepuffer** stellt das Puffer-Signal Modul dar.

Aus der Schematic-Ansicht der Bittiming-Logik (Abbildung 32) wird erneut ein Symbol generiert, welches in der Testbench für die AMS Simulation verwendet wird (Abbildung 33).

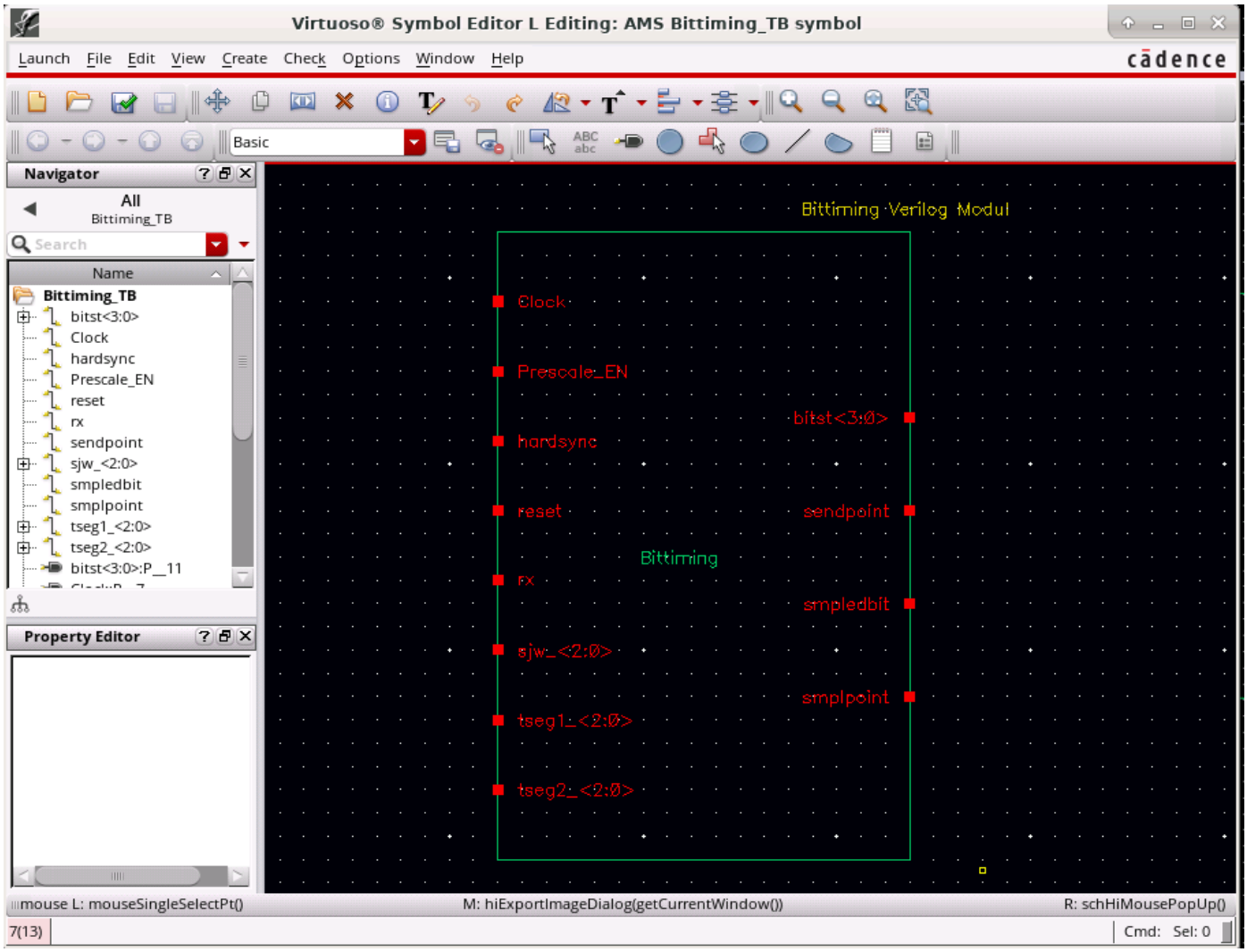

Abbildung 33: Symbol der Bittiming Logik

4.3. Einrichten der AMS Simulation

Nun sind alle notwendigen **Bittiming** Verilog-Module für die Testbench erstellt und es kann mit der Erstellung der Testbench an sich begonnen werden. Dazu wird in der Library **AMS** eine neue Zelle mit dem Namen **Osci_Bittiming_TB** und der View **Schematic** generiert. In diesem Schematic wird die gemischte analog/digitale Schaltung in einem speziellen Schaltplan integriert, um die AMS Simulation durchführen zu können.

Zur Simulation der Bittiming Einheit wird neben der analogen Oszillator- und Referenz-Schaltung aus dem Kapitel 3 (siehe Abbildung 16), welcher als Taktgeber verwendet wird, eine Verilogverhaltensbeschreibung für die Eingangsstimuli entworfen, mit der die Eingangssignale des **Bittiming** Moduls angesteuert werden. Dazu wird eine neue Verilog Zelle mit dem Namen „**Input_Bittiming**“ generiert, deren zugehöriges Symbol in einer Schematic Testschaltung eingefügt und mit den Eingängen des Bittiming Verilog Moduls angeschlossen wird. Außerdem wird ein Verilog-Modul mit dem Namen „**Prescale_EN**“ erstellt, welches die Frequenz des vom

analogen Relaxationsoszillator erzeugten Taktsignals reduziert und an die Bittiming-Einheit über das Signal **Prescale_EN** weiterleitet.

Nachdem alle für die Simulation benötigten Komponenten im Schematic importiert und miteinander verbunden wurden, sieht die gemischte analog/digitale Testschaltung wie folgt aus:

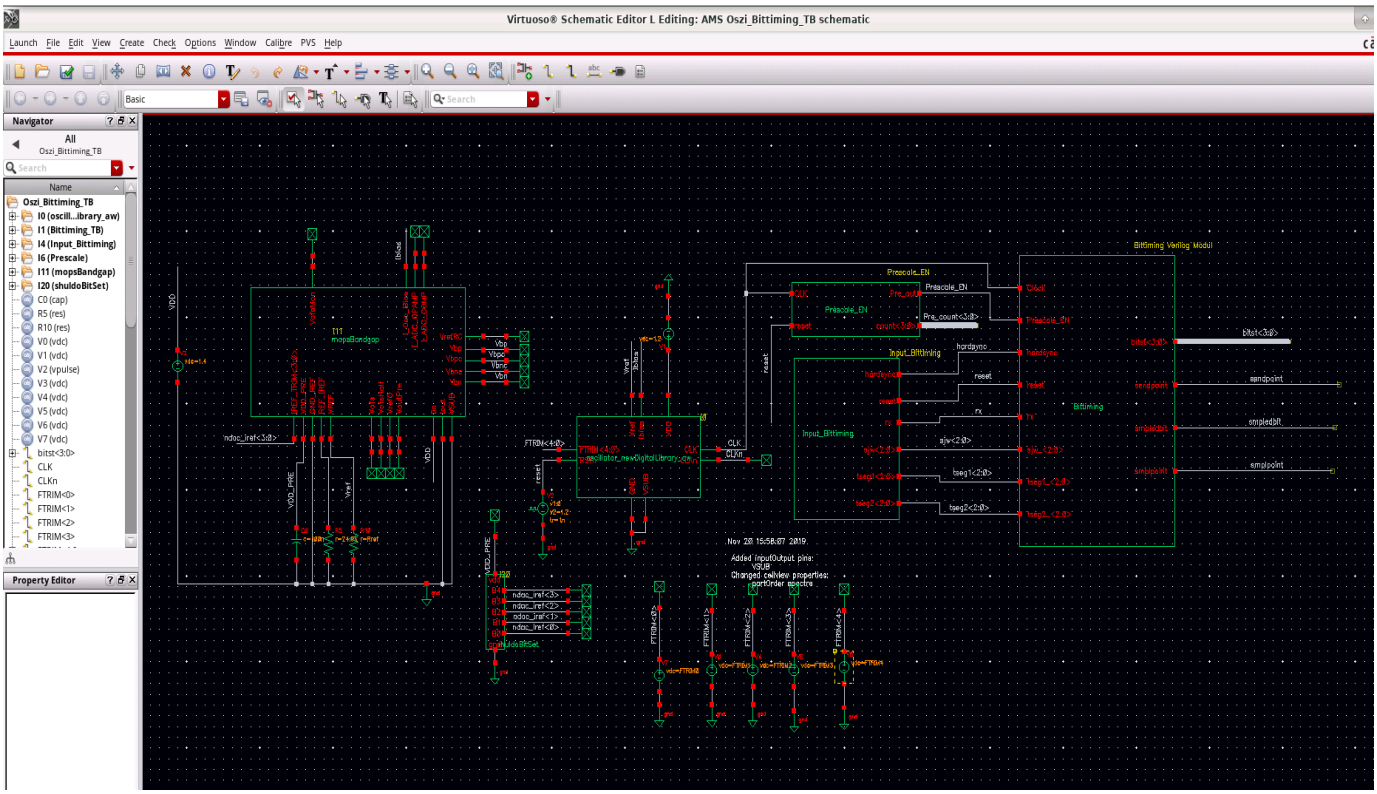

Abbildung 34: die gemischte Analog/Digitale Testschaltung

Im nächsten Schritt muss für die Testbench eine Konfiguration erstellt werden, in der eingestellt wird, welcher View für eine Instance zur Simulation verwendet werden soll. Diese Konfiguration beginnt im Virtuoso Schematic Editor, wo die Testschaltung entworfen ist. Hierzu ist in der Menüleiste **File/New** auszuwählen.

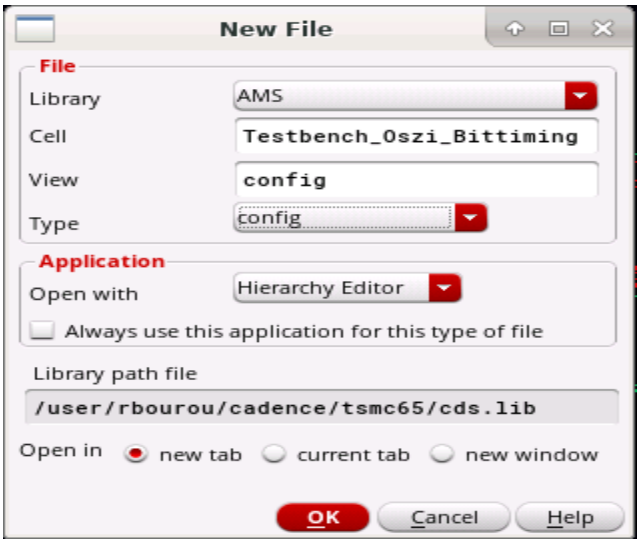

Abbildung 35: Neue Zelle für die Konfigurationsansicht

Im Fenster „New File“ wird sowohl unter **View** als auch unter **Type config** ausgewählt bzw. eingetragen. Da die Erstellung der neuen Konfigurationszelle aus dem Testschaltung-Editor geöffnet wurde, werden die anderen Felder, wie Bibliothek Name usw., automatisch ausgewählt. Nach Bestätigung der Eingaben durch OK öffnet sich das „New Configuration“ Fenster (Abbildung 36).

Abbildung 36: New Configuration Fenster

Mit dem Klick auf dem **Use Template** Button öffnet sich das in Abbildung 37 dargestellte Fenster.

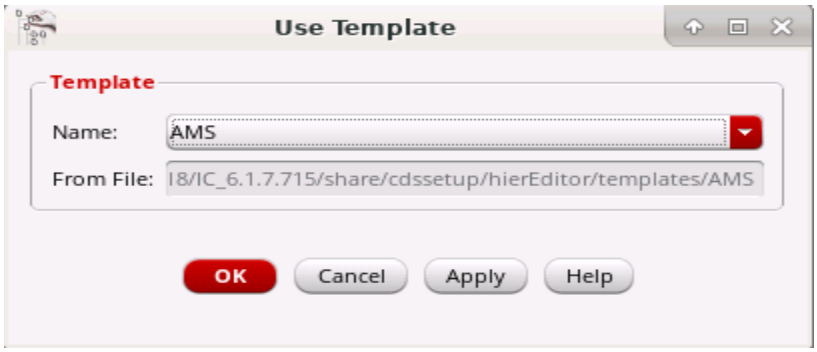

Abbildung 37: Use Template Fenster

Im „**Use Template**“ Fenster wird unter Name **AMS** ausgewählt. Nach Bestätigung der Eingabe durch Ok werden die benötigten Informationen für die Konfiguration im Fenster „**New Configuration**“ automatisch ausgewählt (Abbildung 38). Hierbei ist zu beachten, dass unter View **schematic** ausgewählt ist und der Name der Bibliothek und der zu simulierenden Testschaltung richtig eingetragen sind.

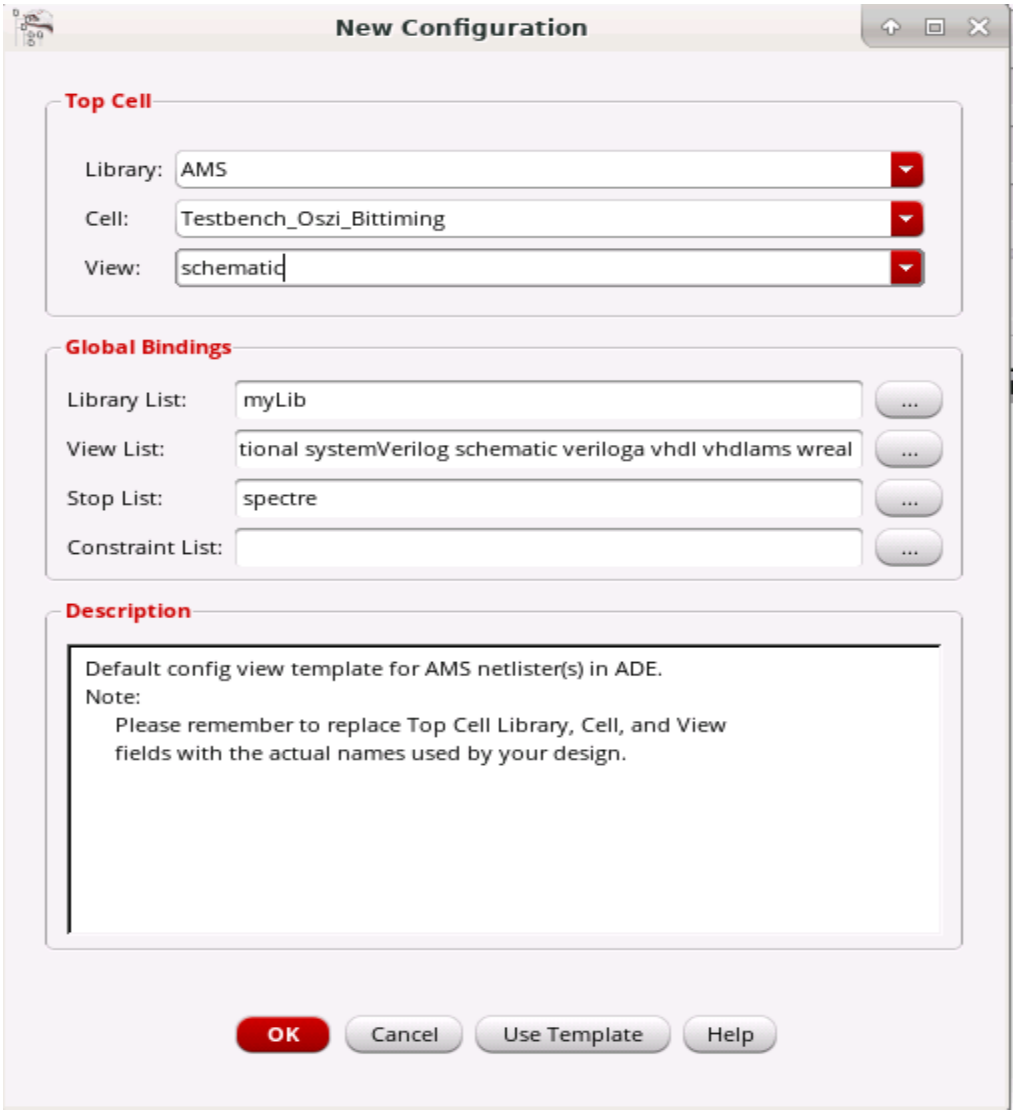

Abbildung 38: New Configuration Fenster

Nach Bestätigung der Eingaben durch ok öffnet sich das **Virtuoso Hierarchie Editor** Fenster (Abbildung 39), welches eine Tabellenansicht entspricht, in der alle verwendeten Zellen aufgeführt sind.

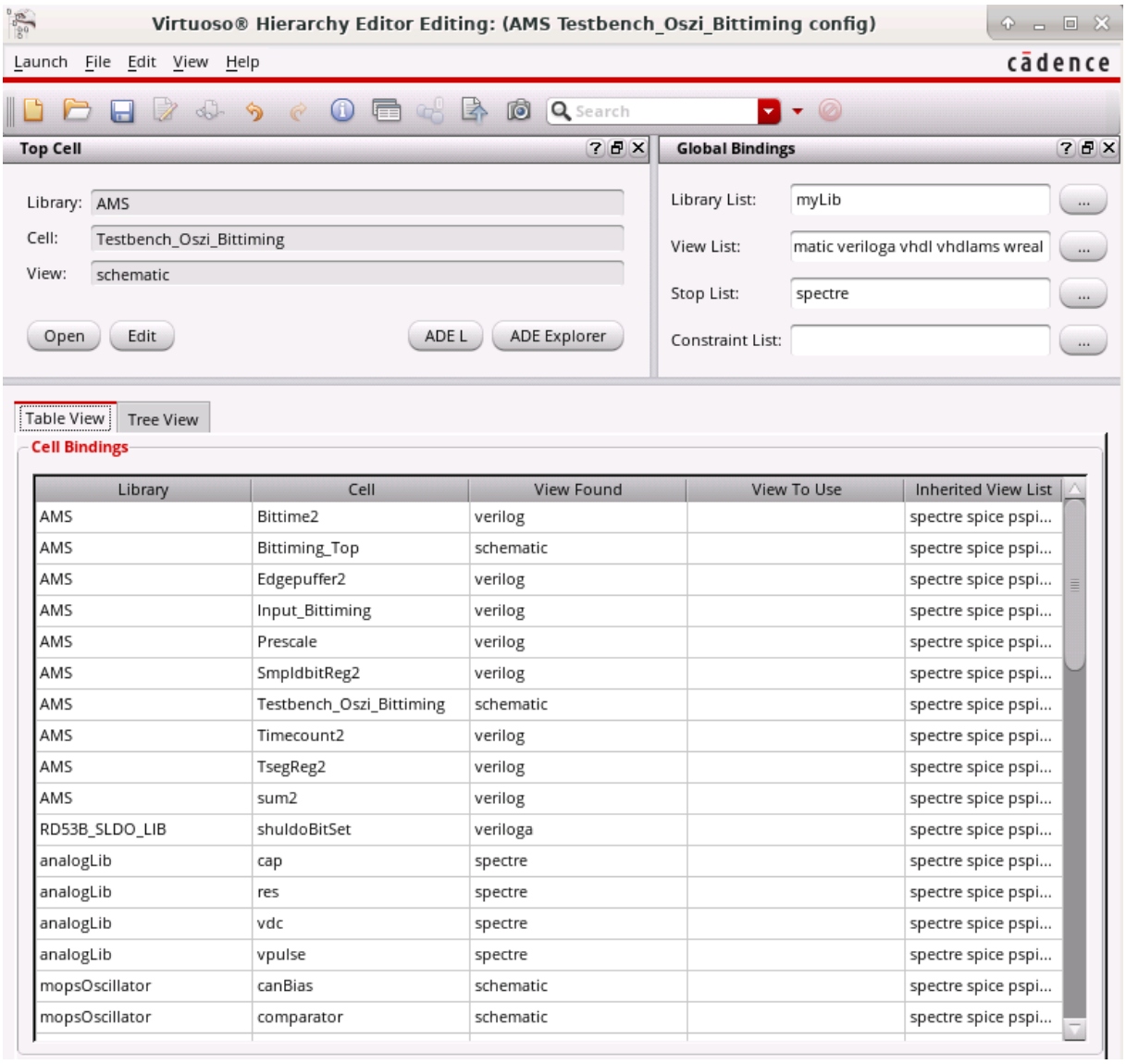

Abbildung 39: Virtuoso Hierarchie Editor

Die Konfigurationsansicht wird vom Netzlister[14] verwendet, um festzulegen, welche View für die in der Testschaltung enthaltenen Bauelementen bei der Erstellung der Simulationsnetzliste verwendet werden soll. Beispielsweise werden in der oben gezeigten Konfigurationsansicht (Abbildung 39) die **Testbench_Oszi_Bittiming** Zelle als Schematic und die **Bittime2** als Verilog Modul definiert.

Der Hierarchie-Editor verfügt über zwei Anzeigemodi:

- **Tabellenansicht**
- **Baumstrukturansicht**

Der Hierarchie-Editor kann zur Baumstrukturansicht umgewandelt werden, indem auf die Registerkarte "**Tree View**" geklickt wird. In beiden Ansichten kann für jede Instanz der gewünschte View für die Simulation ausgewählt werden.

[14] Netzlister beinhaltet die textuelle Beschreibung der Verbindungen zwischen den in der Testbench enthaltenen Komponenten

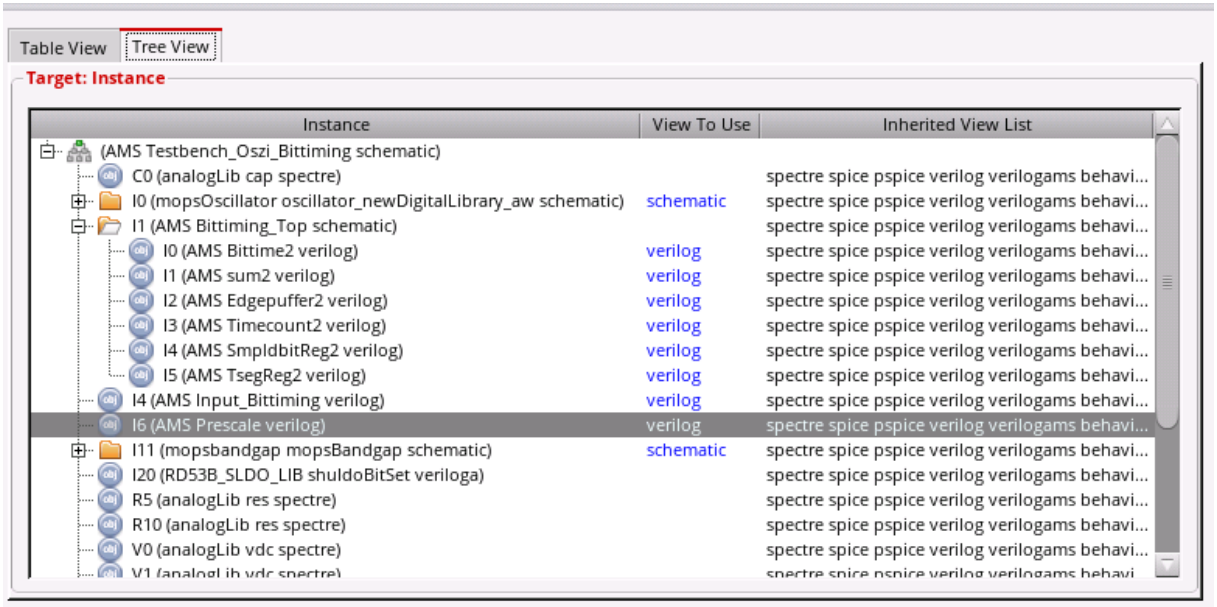

Abbildung 40: Baumansicht

Wie man in der Abbildung 40 sieht, sind die Unterzellen **I0** bis **I5** des **Bittiming**-Moduls als „**verilog**“ View definiert und die Oszillatorzelle wiederum als „**schematic**“. Das heißt, dass die Instanzen **I0** bis **I5** von dem **AMS** Simulator als digitale Komponenten betrachtet werden, während die Oszillatorschaltung den analogen Teil der Testschaltung repräsentiert.

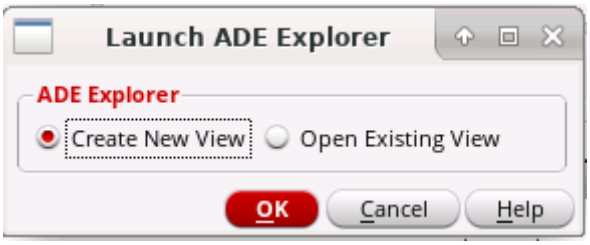

Abbildung 41: Launch ADE Explorer Fenster

Für die **AMS** Simulation wird der **ADE Explorer** verwendet. Dieses Tool lässt sich im Virtuoso Hierarchie Editor mit dem Klick auf **ADE Explorer** Button aufrufen. Wenn **ADE Explorer** das erste Mal gestartet wird, muss eine neue View angelegt werden. Dazu wird die Option „**Create New View**“ im „**Launch ADE Explorer**“ Fenster selektiert und mit **Ok** bestätigt (Abbildung 41).

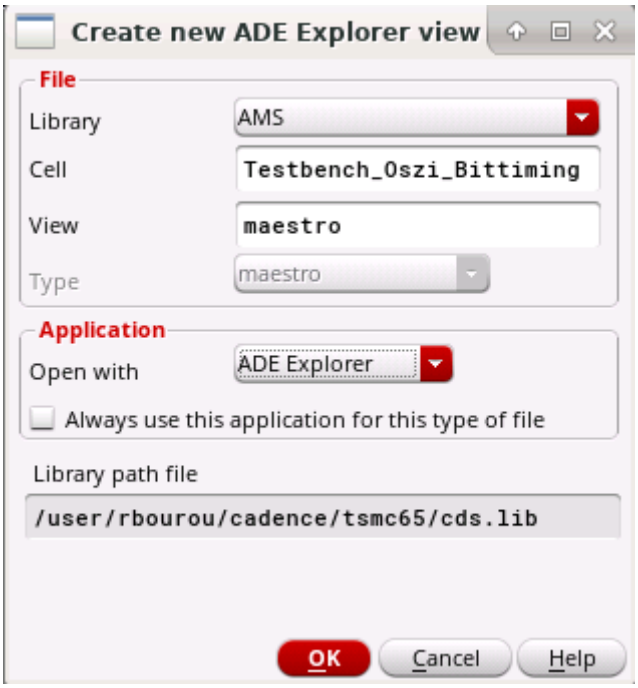

Abbildung 42: Konfiguration einer neuen ADE Explorer View

Nach Bestätigung der Eingaben durch **OK** öffnet sich das ADE Explorer Hauptfenster (Abbildung 43).

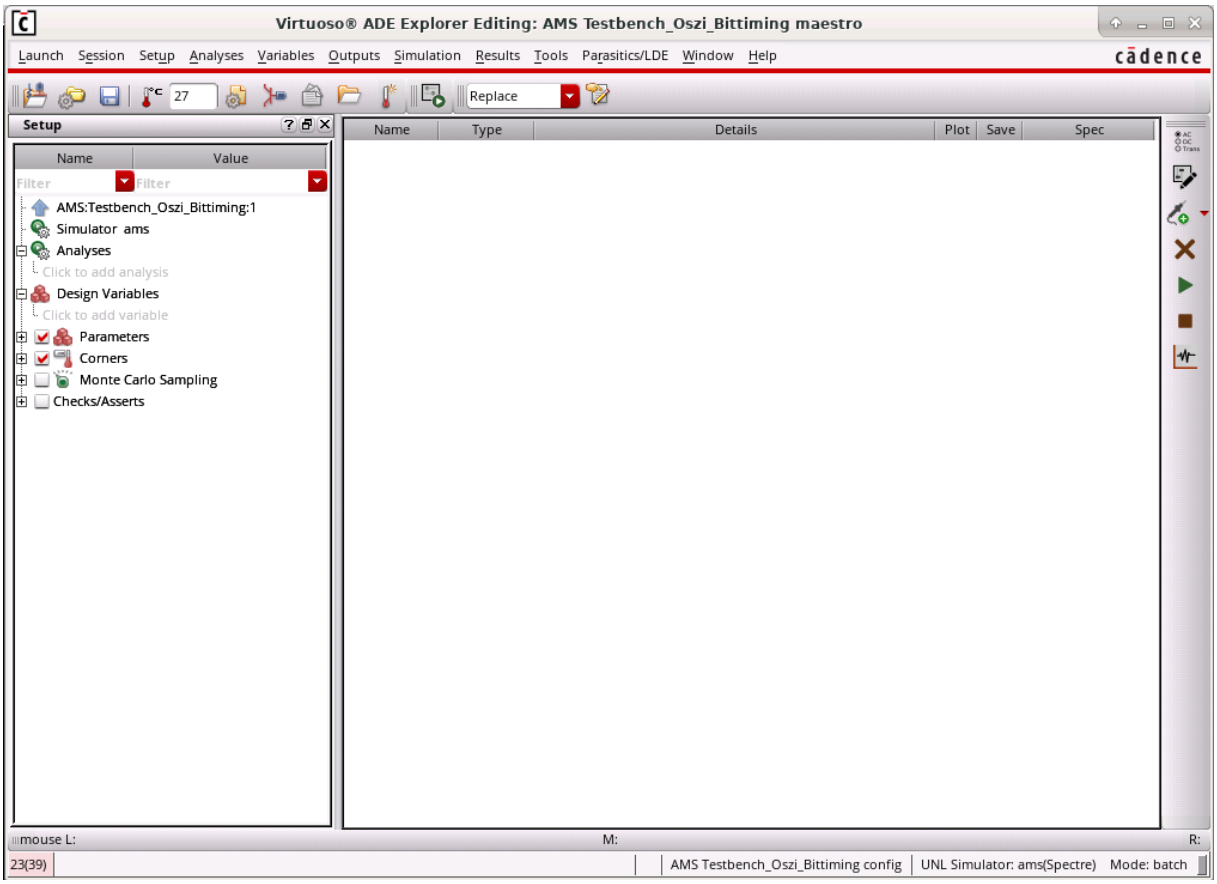

Abbildung 43: Virtuoso ADE Explorer Hauptfenster

In diesem Fenster kann unter dem Menü **Setup/Simulator** überprüft werden, ob der Simulator wie in der Abbildung unten auf **ams** eingestellt ist.

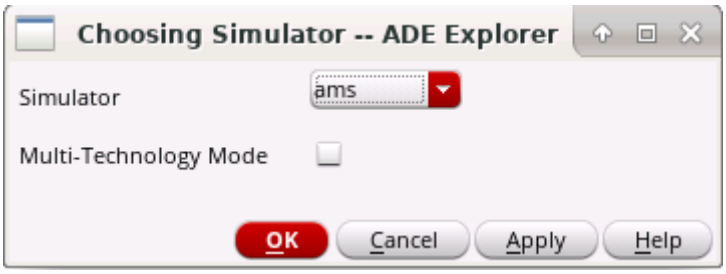

Abbildung 44: Simulator-Einstellung

Für die Verifikation der Mixed-Signal-Schaltungen werden aufgrund der Kombination von Digital und Analogkomponenten transiente Simulationen im Zeitbereich eingesetzt. Hierzu ist in der Menüleiste **Analyses/Choose** auszuwählen und im „**Choosing Analyses**“ Fenster das Feld **tran** zu selektieren. Im Feld Stop Time wird eine Simulationszeit eingetragen, welche in diesem Fall 20 µs beträgt. Anschließend wird das Feld Enabled selektiert und die Simulationseinrichtung mit einem Klick auf OK bestätigt (Abbildung 45).

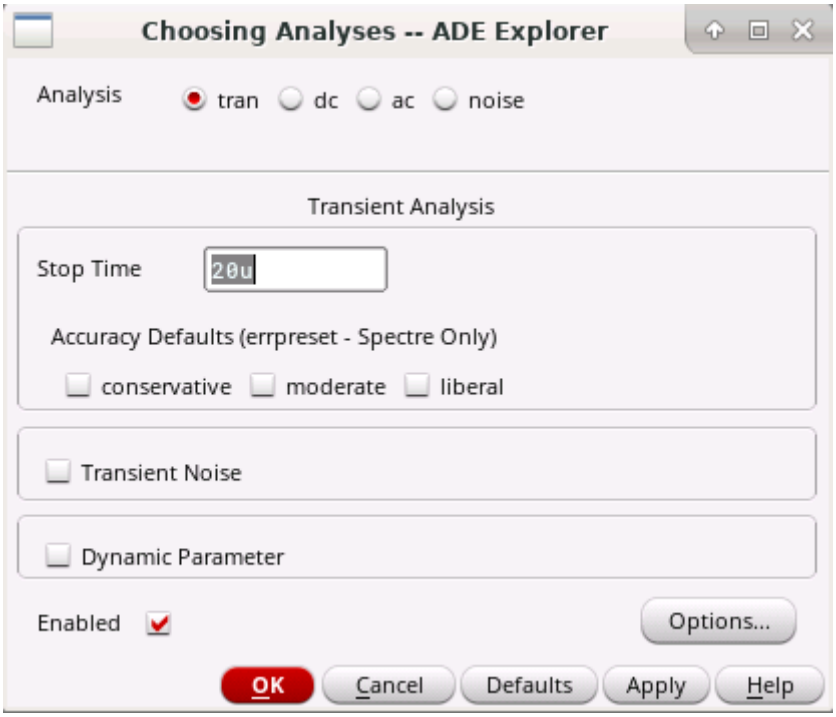

Abbildung 45: Konfiguration einer Transientenanalyse

Festgelegte Variablen im Schematic müssen in **ADE Explorer** importiert werden. Dies Erfolgt über den Bereich **Design Variables** auf der linken Seite des **ADE Explorer** Hauptfensters. Ein Rechtsklick in diesen Bereich öffnet ein Kontextmenü. Durch Auswahl des Menüpunktes **Copy From Cellview** werden alle im Schematic angelegten Variablen importiert. Durch einfaches Anklicken des entsprechenden Feldes in der Spalte Value können den Variablen Werte zugewiesen werden.

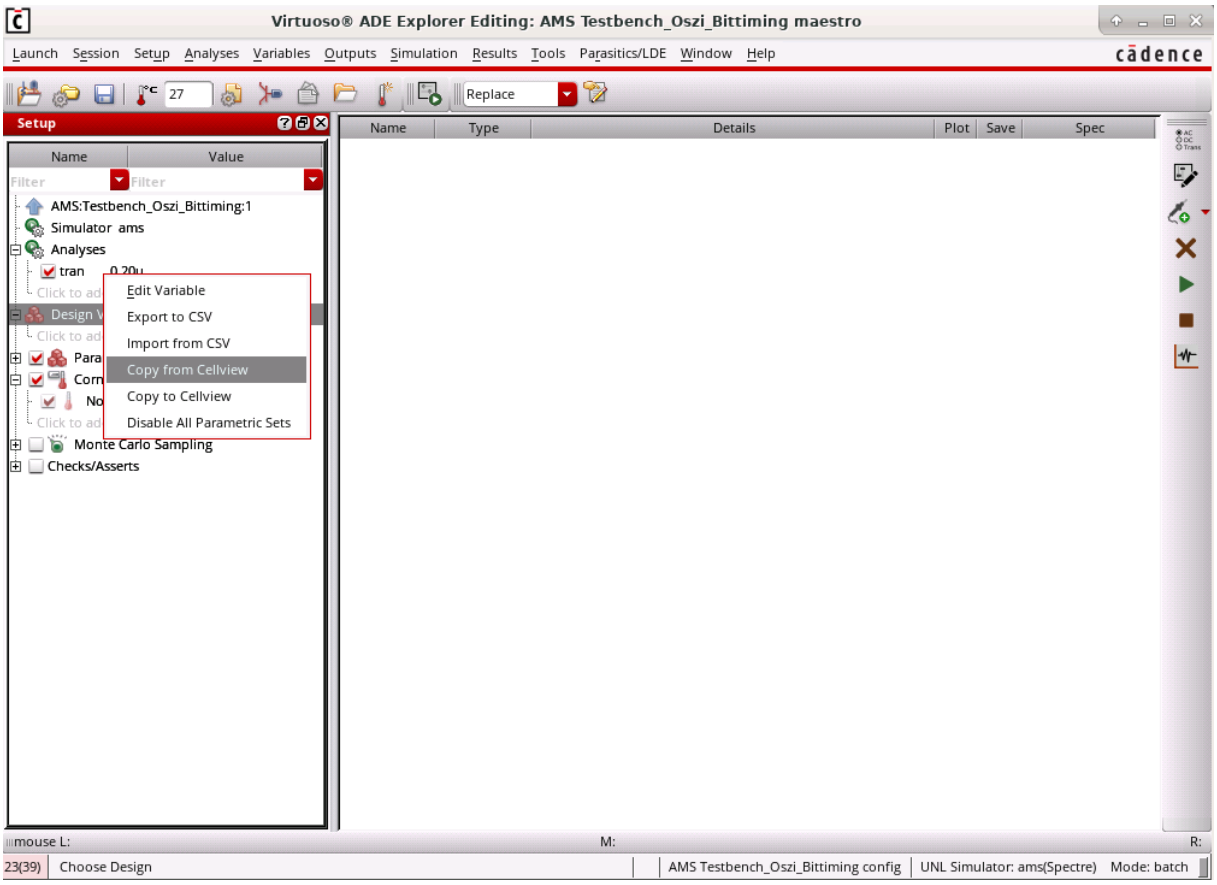

Abbildung 46: Import der Variablen aus der Testschaltung

Nach dem Import sollten die Variablen **FTRIM0**, **FTRIM1**, **FTRIM2**, **FTRIM3**, **FTRIM4**, **iRefSet** und **Rref** im Design Variables-Fensterbereich gelistet sein. Die Variablen werden zunächst auf die folgenden Zahlenwerte gesetzt.

Name	Value
FTRIM0	0
FTRIM1	1.2
FTRIM2	1.2
FTRIM3	1.2
FTRIM4	0
iRefSet	8
Rref	30k

Als nächstes sollen die Verbindungsregeln, welche für die analog/digitale Signalumwandlung zwischen den digitalen und analogen Blöcken verwendet werden, festgelegt werden. Hierzu ist in der Menüleiste **Setup→Connect Rules/IE** auszuwählen (siehe Abbildung 46).

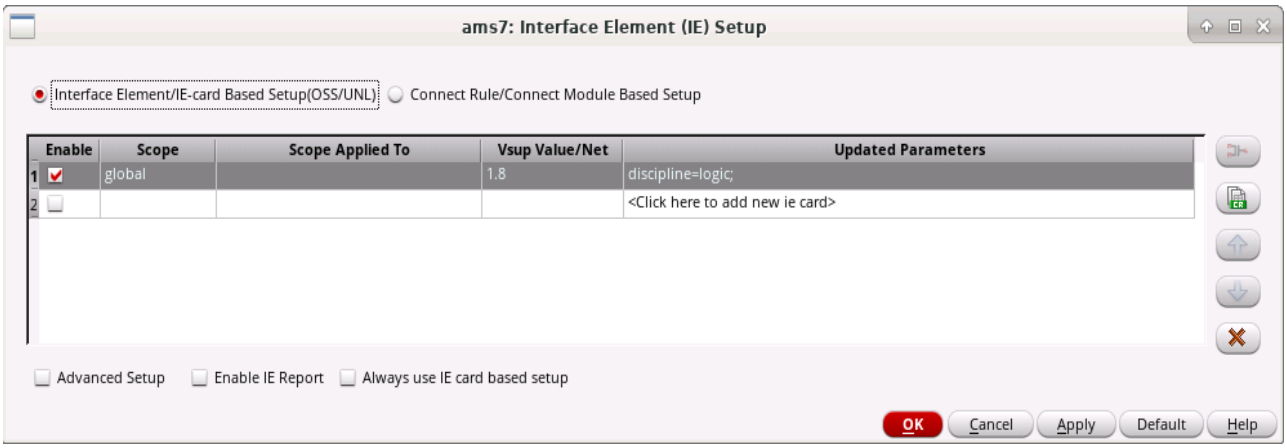

Abbildung 47: Einstellung der Schnittstellenparameter

Die Schnittstellenparameter zwischen dem Analog- und Digital-Teil sind im Hintergrund in einem **Verilog-AMS** Skript definiert und können über dem in Abbildung 47 dargestellten Fenster angepasst werden.

Durch Anklicken des **Vsup Value/Net** Feldes kann der Parameter **Vsup** an den Wert 1,2 V angepasst werden, welcher dem Versorgungsspannungswert für den Analogteil entspricht und als logische „1“ in dem Digitalteil interpretiert wird.

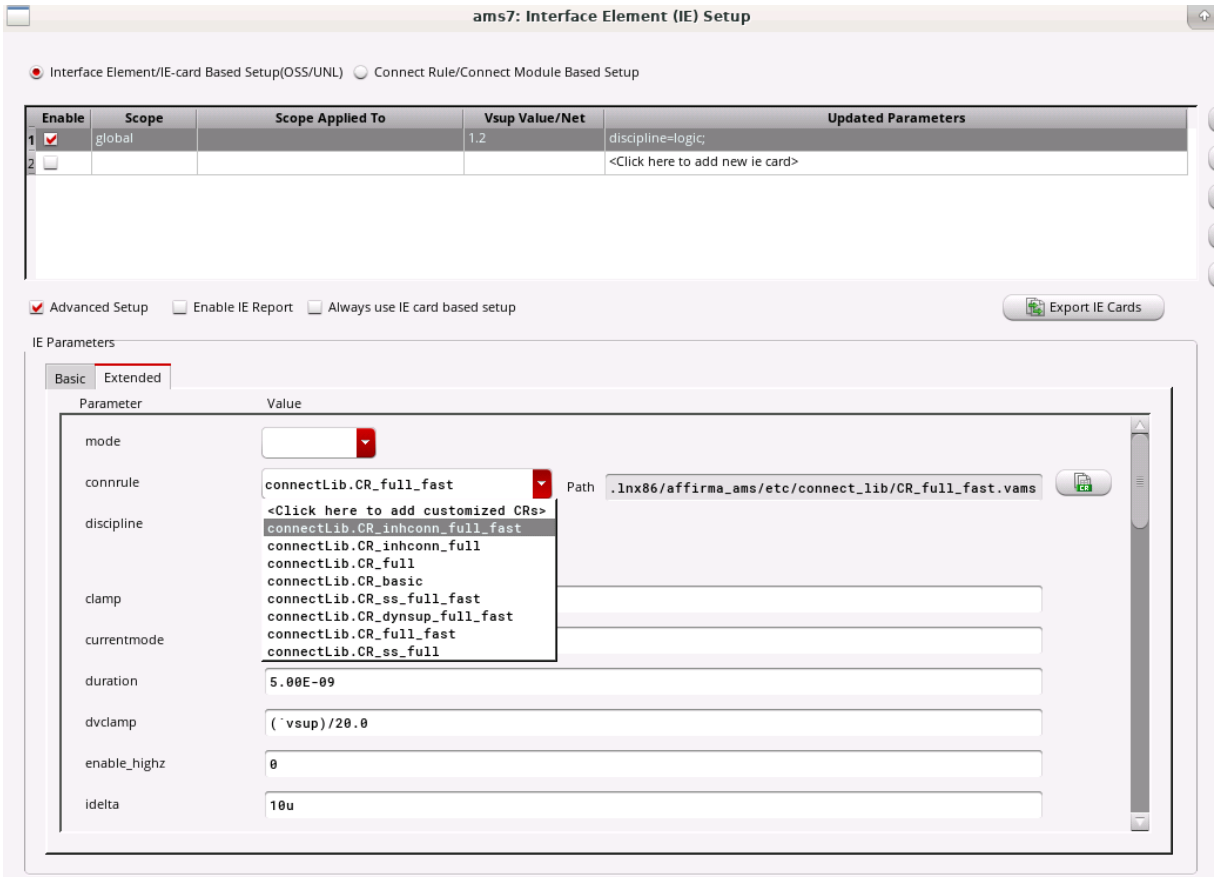

Abbildung 48: Extended Tab der Schnittstellenparameter-Einrichtung

Um weitere Parameter anpassen zu können, muss das Kontrollkästchen **Advanced Setup** aktiviert werden (Abbildung 48). In der Extended Ansicht können alle Schnittstellenparameter eingestellt werden. Beispielweise können unter **connrule** andere Verbindungsregeln gewählt

werden. Es wird empfohlen, die **full_fast** Verbindungsregeln für die **AMS** Simulation auszuwählen. Mit einem Klick auf Ok wird die Anpassung der Schnittstellenparameter abgeschlossen.

Die Simulationsgeschwindigkeit kann durch die gleichzeitige Ausführung mehrerer Simulationen deutlich beschleunigt werden. Hierzu wird unter **Setup→High-Performance/Parasitic Reduction** ausgewählt und die Option **APS** im Bereich **Simulation Performance Mode** selektiert (Abbildung 49).

Abbildung 49: Einstellung der AMS Spectre Performance

Die Anzahl parallellaufender Simulationen wird im Bereich **Multi-Threading** eingestellt. Dort wird die Option **Auto** selektiert, dann wird die Anzahl der vorhandenen CPU-Cores im Rechner automatisch ermittelt und dadurch festgelegt, wie viele Threads parallel ausgeführt werden.

Abschließend werden Ausgangssignale, welche simuliert bzw. dargestellt werden sollen, für die **Tran**-Simulation ausgewählt. Dazu ist im ADE Explorer Hauptfenster in der Menüleiste **Outputs→To Be Plotted→Select on Design** auszuwählen (Abbildung 50).

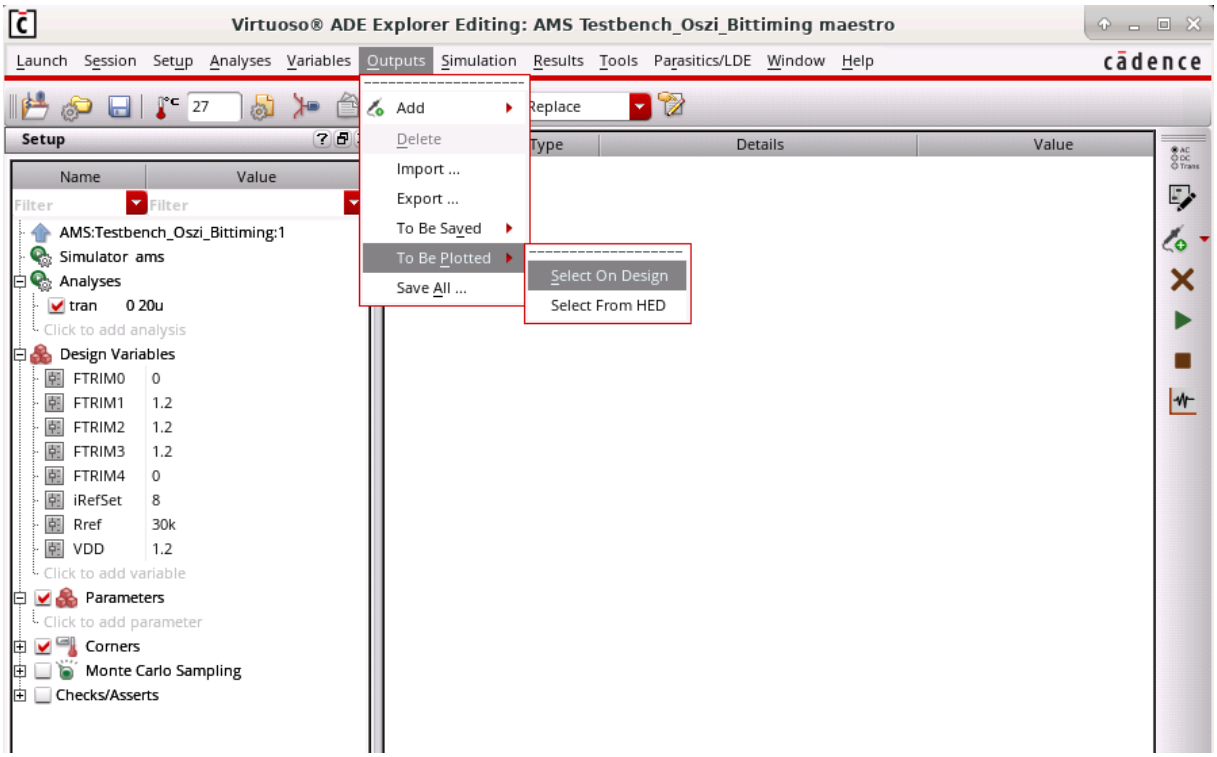

Abbildung 50: Einrichten von Ausgängen

Nun wird in die Schematic-Ansicht gewechselt und per Klick auf die entsprechenden Leitungen die benötigten Signale als Output angelegt. Die selektierten Leitungen werden farblich markiert und die gewählten Stellen werden im Outputs-Bereich des ADE Explorer Fensters gelistet (Abbildung 51).

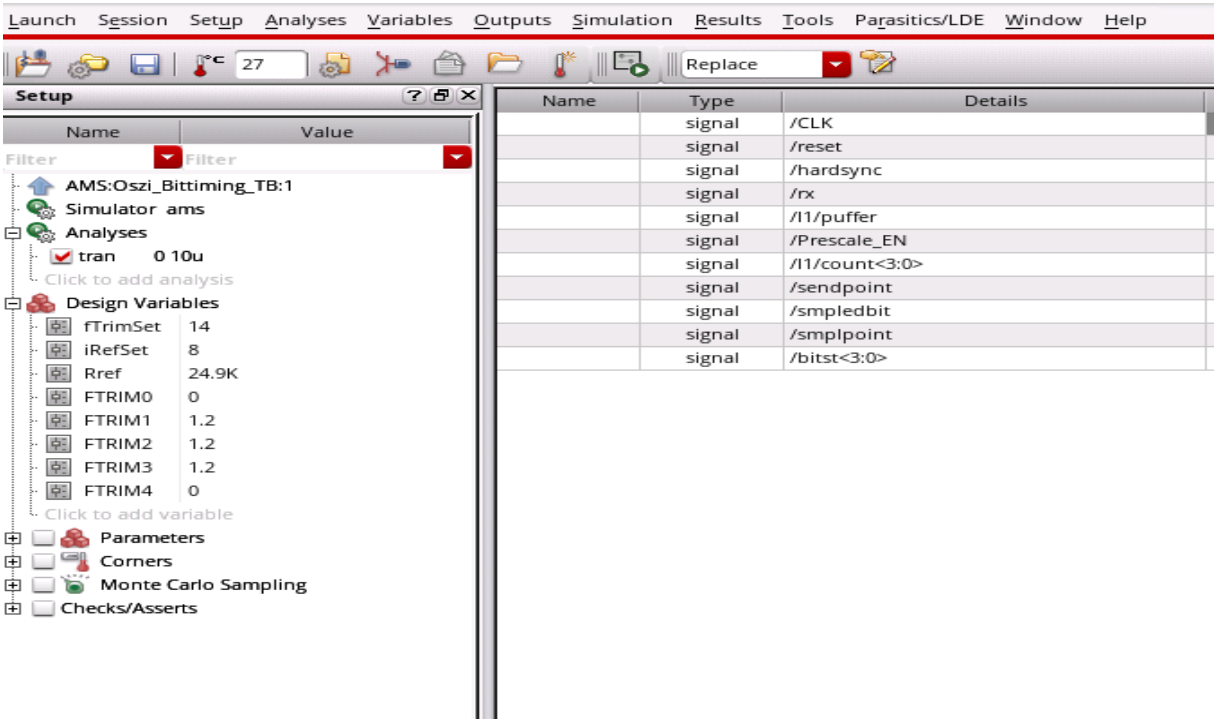

Abbildung 51: die Outputs in dem ADE Explorer Fenster

Nach der Auswahl der Ausgangssignale kann nun die **AMS** Simulation durchgeführt werden. Dazu wird mit einem Klick auf das grüne Symbol **Run** in der rechten Symbolleiste oder über den Menüpunkt **Simulation→Netlist and Run** die Simulation gestartet.

```
// Library - AMS, Cell - Oszi_Bittiming_TB, View - schematic
// LAST TIME SAVED: Oct 20 09:27:20 2020
// NETLIST TIME: Oct 28 23:48:04 2020

`worklib AMS
`view schematic

`timescale 1ns / 1ns
(* cds_ams_schematic *)

module Oszi_Bittiming_TB ( );
wire [3:0] bitst;
wire [2:0] tseg2;
wire [2:0] sjw;
wire [2:0] tseg1;
wire [3:0] Pre_count;
wire [4:0] FTRIM;
wire [3:0] ndac_iref;
wire VDD_PRE;
wire net043;
wire VDD;
wire net029;
wire net041;
wire Vref;
wire net038;
wire net037;
wire Vbn;
wire Vbnc;
wire Vbp;
wire Ibias;
wire Vbpc;
wire net039;
wire net035;
wire net040;
wire net036;
wire net042;
wire reset;
wire CLKn;
wire CLK;
wire net034;
wire Prescale_EN;
```

Abbildung 52: generierte Netzliste aus der Testbench

Hierfür wird automatisiert eine Netzliste aus der **Testschaltung** generiert und dem **AMS** Simulator übergeben. Die erzeugte Verilog-AMS-Netzliste lässt sich im **ADE Explorer** unter **Simulation→Netlist→Display** anzeigen.

4.4. Simulationsergebnisse

Als nächstes wird die Funktionalität des Bittiming-Moduls geprüft. Mit folgenden Werten wird die AMS-Simulation durchgeführt.

Tseg1	**4**
Tseg2	**5**
Sjw	**3**

Wobei **Tseg1** die Felder Propagation-Segment und Phase-Segment 1 umfasst. **Tseg2** entspricht der Länge des Phase-segments 2. **SJW** entspricht dem Wert der Synchronisationssprungweite.

Zur Verifizierung des Bittiming-Moduls werden folgende Signale simuliert.

CLK	Taktsignal aus dem analogen Oszillator
Reset	Setzt das Bittiming Modul zurück
Hardsync	Fördert die Durchführung der Harte Synchronisation
Rx	Das eingehende Signal an das Timing Modul
Puffer	Wird mit Rx Signal verglichen, um eine fallende Flanke zu detektieren
Prescaler_EN	Taktsignal-Frequenzteiler Ausgangssignal, steuert die Zählereinheit
Count<3:0>	Zählerstand der Zeitsegmente
Sendpoint	Bezeichnet das Ende des Bitsegments
Smpledbit	Wird aktiv, wenn das aktuelle Bit auf dem Bus gespeichert werden soll
Smplpoint	Bezeichnet den Samplezeitpunkt
Bitst<3:0>	Zustand der Bittiming-FSM

4.4.1. Harte Synchronisation

Die harte Synchronisation wird zu Beginn der CAN-Botschaft beim ersten rezessiv zu dominant Übergang eines Startbits durchgeführt. Wie in der Abbildung 53 zu sehen ist, geht die Zustandmaschine nach dem Reset des CAN-Controllers und bei dem Flankenwechsel in den Zustand **hardset,** welcher mit 1 bezeichnet ist (Bitst<3:0>=4’d1).

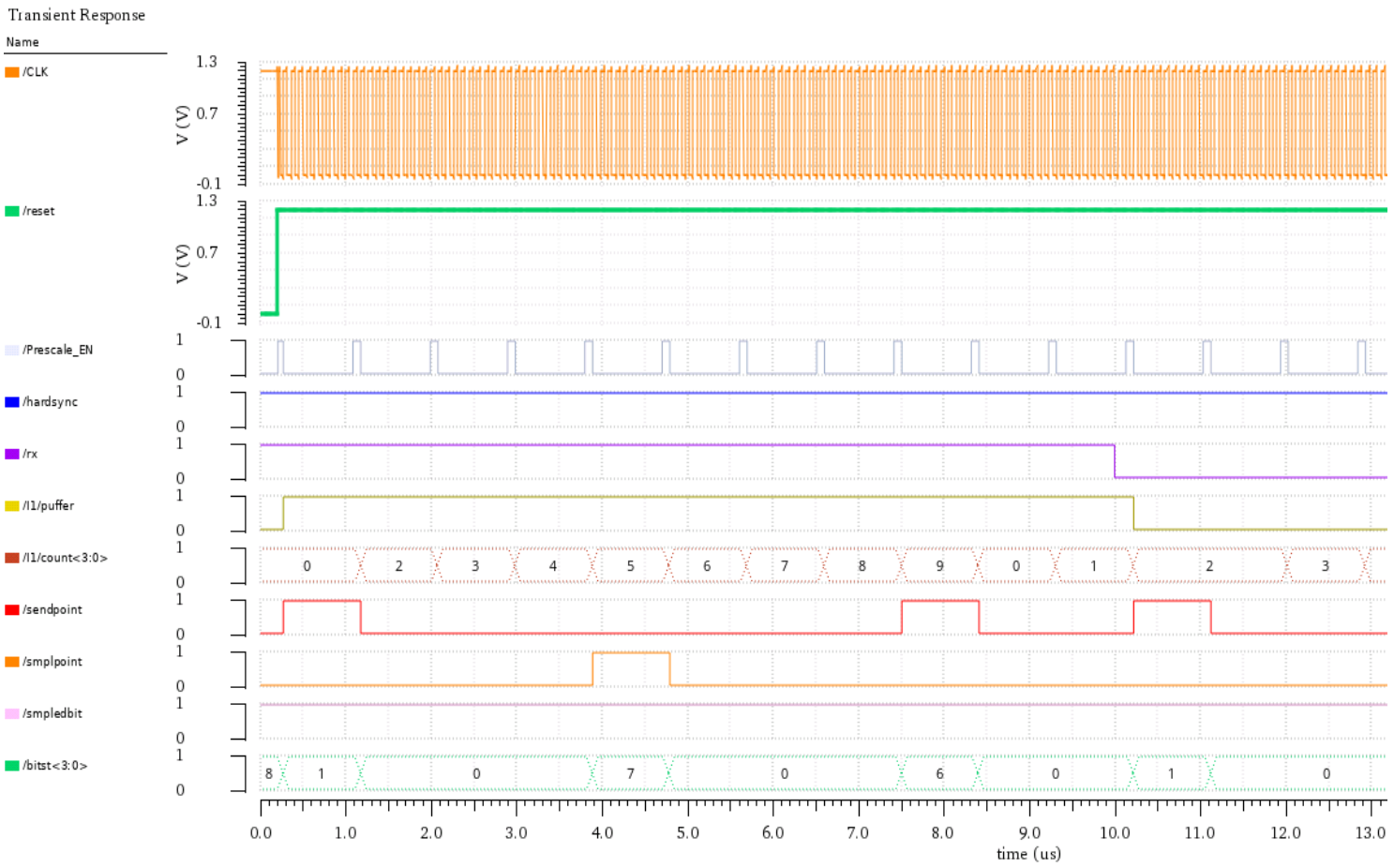

Abbildung 53: Harte Synchronisation des Bittiming Moduls

4.4.2. Resynchronisation

Die Resynchronisation oder weiche Synchronisation wird mit jeder detektierten fallenden Flanke durchgeführt. Falls der Flankenwechsel außerhalb des Synchronisationssegments stattfindet, reagiert die Bittiming-FSM darauf mit einer Verkürzung bzw. Verlängerung der Zeitsegmente.

- **verfrühter Flankenwechsel**

Wie in der Abbildung 54 zu sehen ist, tritt die fallende Flanke innerhalb des Phase-Segments 2 auf. Es handelt sich hier um einen negativen Phasenfehler von eins, da der Flankenwechsel beim Zählerstand neun aufgetreten ist. Die Bittiming FSM reagiert darauf mit einer Verkürzung des Phase-Segments 2 um eins.

Da der Phasenfehler kleiner als der SJW Wert ist, springt die Zustandmaschine in den Zustand **Slimok** (Bitst<3:0>=4'd4). In diesem Zustand wird dem Zählerstand zwei (count<3:0>=4'd2) zugeordnet und das Signal **sendpoint** gesetzt. Somit ist der Phasenfehler kompensiert.

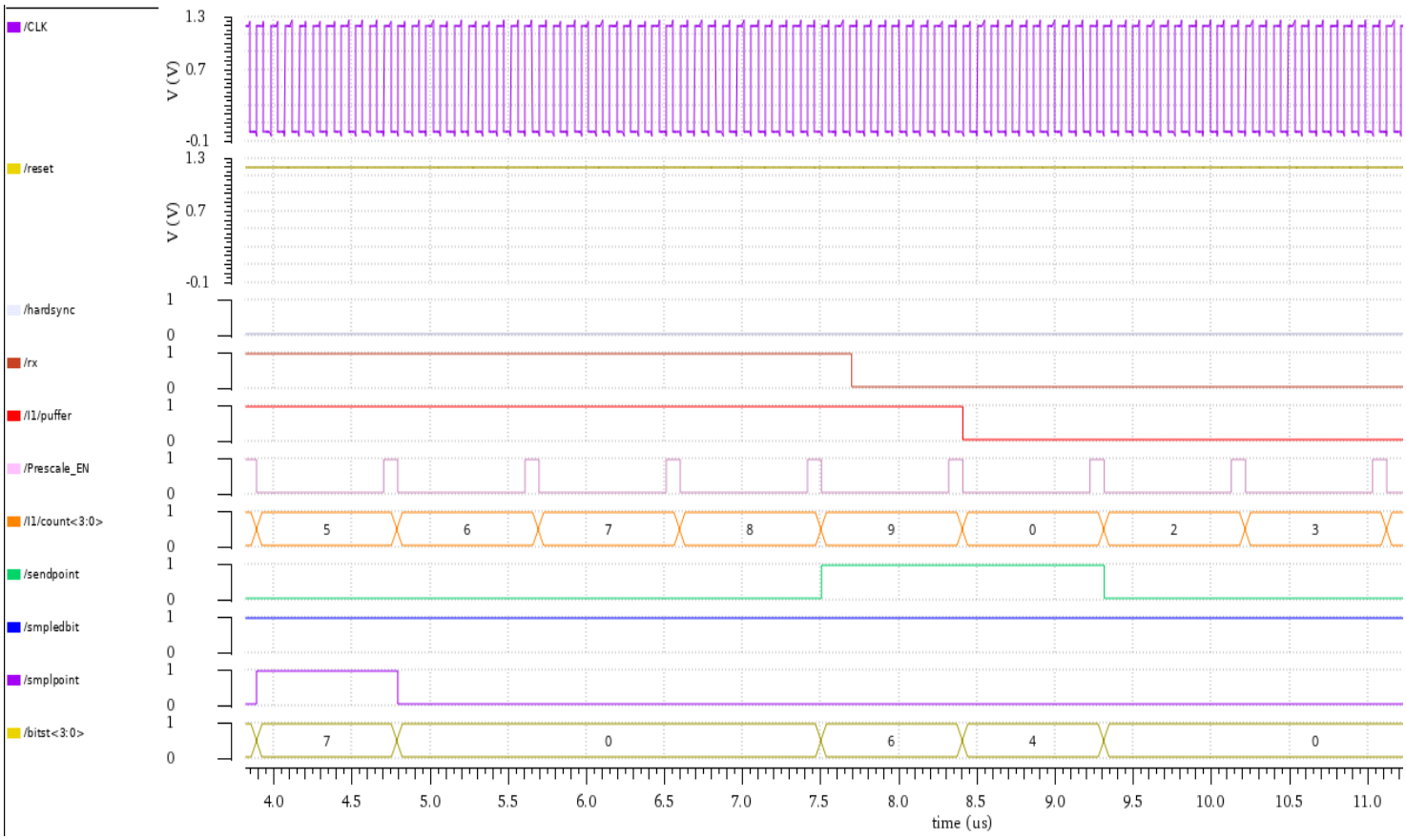

Abbildung 54: verfrühter Flankenwechsel mit Phasenfehler=1

In Abbildung 55 ist ein verfrühter Flankenwechsel mit einem Phasenfehler von fünf dargestellt. Da der Phasenfehler größer ist als den SJW Wert, geht Die Bittiming FSM in den Zustand **Slimnok** (bitst<3:0>=4'd5). Das Phase-Segment 2 wird dann um drei verkürzt, welcher dem SJW Wert und dem maximalen Wert, der kompensiert werden kann, entspricht. Daher überspringt der Zähler (count<3:0>) die Werte neun, null und eins.

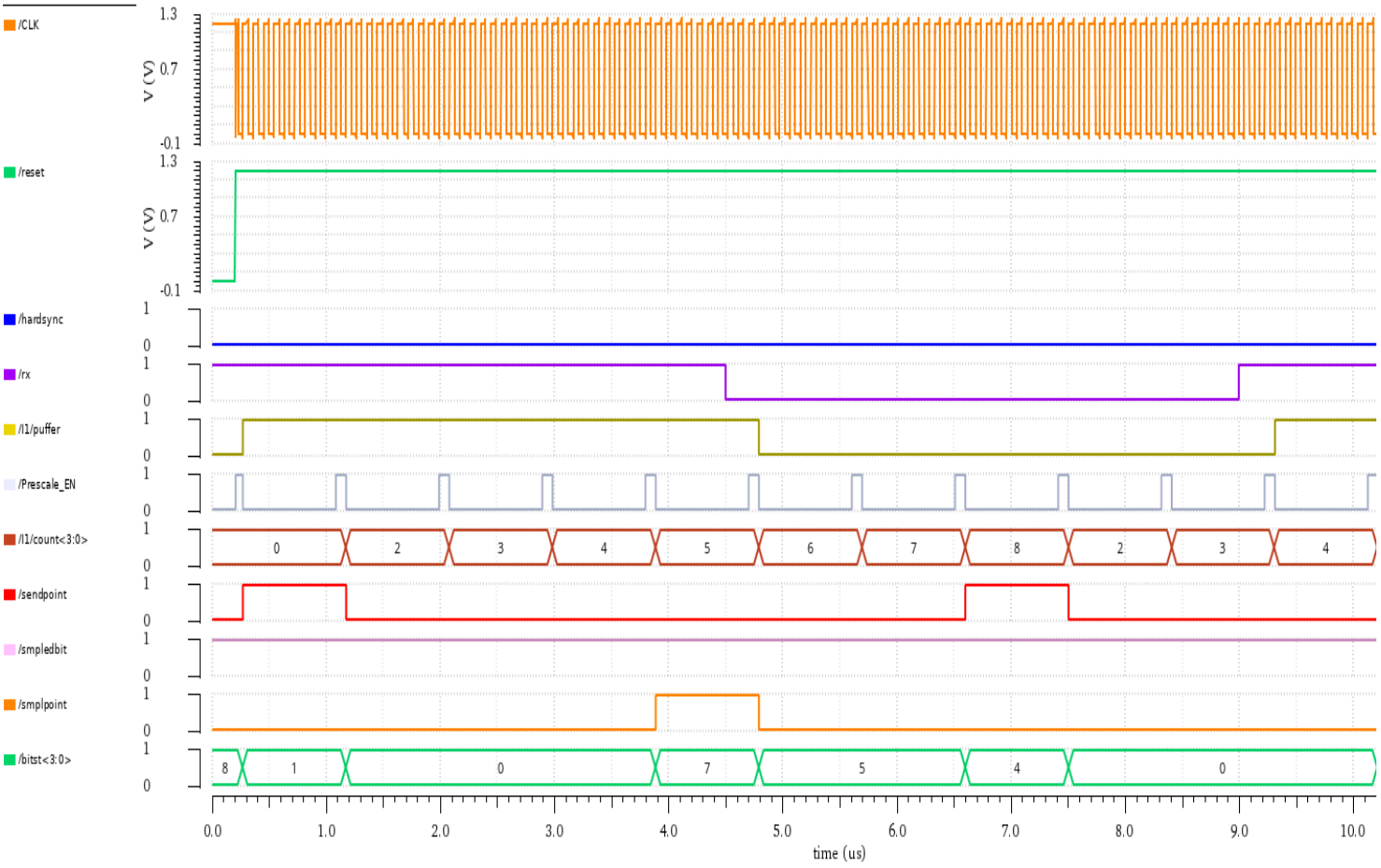

Abbildung 55: verfrühter Flankenwechsel mit Phasenfehler=5

- verspäteter Flankenwechsel

In Abbildung 56 ist eine verspätete fallende Flanke mit einem positiven Phasenfehler von drei dargestellt. Die Bittiming FSM geht in den Zustand **Stretchok** (Bitst<3:0>=4'd2). Der Zähler (count<3:0>) wird um den Wert des Phasenfehlers erweitert. Das Zeitsegment **tseg1** wird um drei verlängert, in dem das Signal **Smplpoint** erst beim Zählerstand acht gesetzt wird.

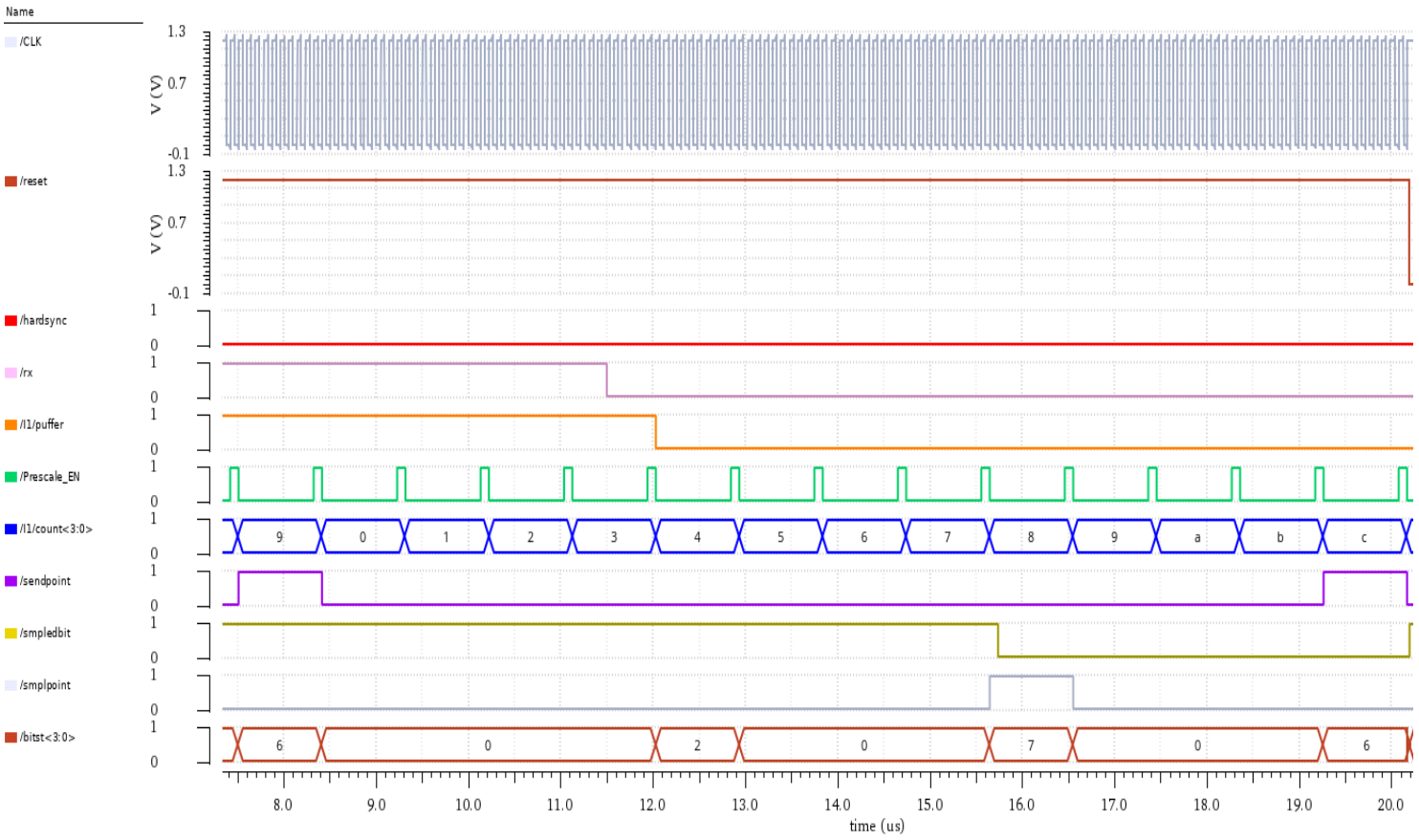

Abbildung 56: verspäteter Flankenwechsel mit Phasenfehler=3

In der Abbildung 57 ist eine verspätete fallende Flanke mit einem positiven Phasenfehler von vier dargestellt. Da der Phasenfehler größer ist als der SJW Wert und der Samplezeitpunkt noch nicht überschritten ist, geht die Bittiming FSM in den Zustand **Stretchnok** (bitst<3:0>=4'd3). Dann wird das Zeitsegment tseg1 um den Wert von **sjw** verlängert, in dem das Signal **Smplpoint** erst beim Zählerstand acht gesetzt wird.

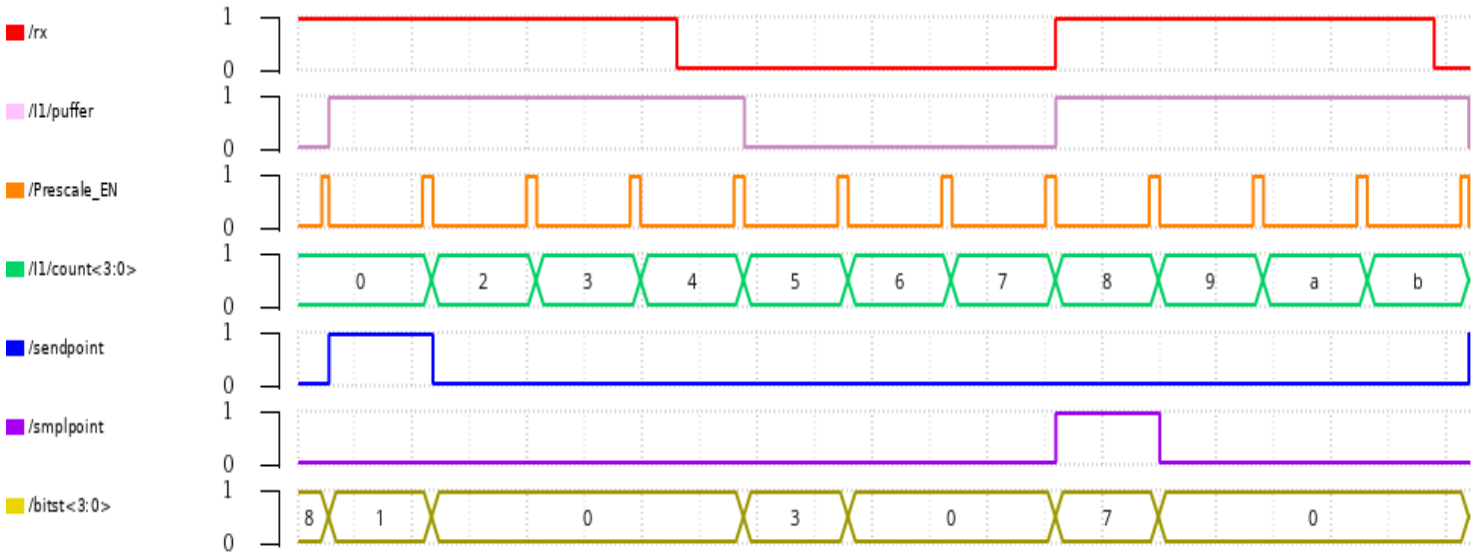

Abbildung 57: verspäteter Flankenwechsel mit Phasenfehler=4

Die oben liegende Simulationsergebnisse belegen, dass die Mechanismen der harten und weichen Synchronisation regelkonform und erfolgreich ausgeführt werden, und somit ist die Verifizierung des Bittiming-Moduls abgeschlossen.

5. Entwurf des Regelsystems

Die im vorherigen Kapitel bereits beschriebene gemischte (analog/digitale) Schaltung (Abbildung 34) wird mit einem Regelsystem erweitert, sodass eine Frequenzregelung des analogen Relaxationsoszillators ermöglicht wird. Dieser dient als Taktgeber für die Bittiming-Einheit des CAN-Controllers.

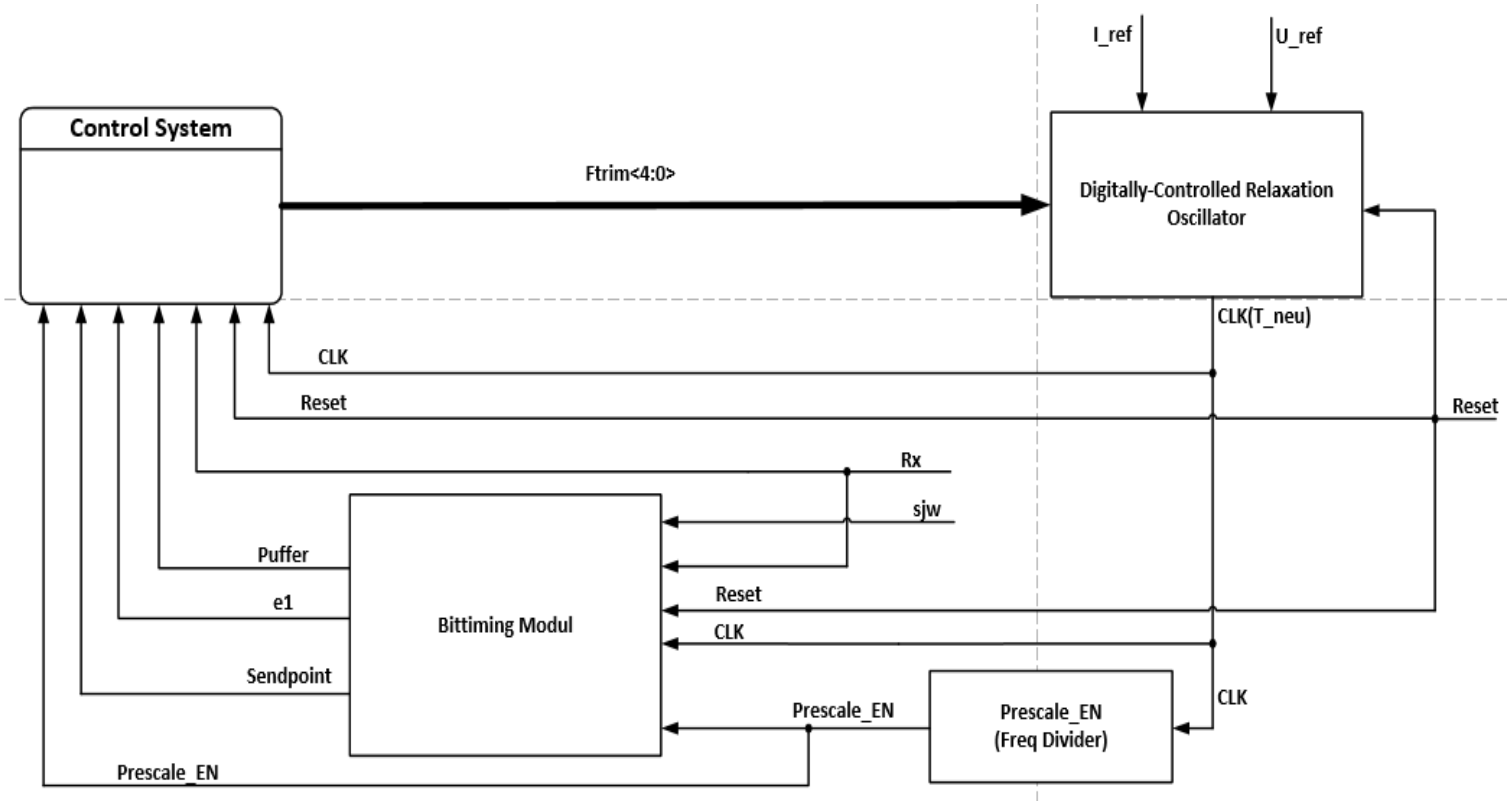

Abbildung 58: Systemübersicht des geschlossenen Regelkreises

Die Abbildung 58 veranschaulicht das High-Level Blockschaltbild, welches eine Gesamtübersicht des geschlossenen Regelkreises aufzeigt. Das zu entwerfende Regelsystem wird in der Gesamtschaltung eingefügt und mit den anderen Komponenten verbunden. Dieses System wird vom Bittiming-Modul mit Informationen versorgt, welche für die Frequenzregelung relevant sind.

Die Hauptaufgabe des Regelsystems besteht darin, bei einer detektierten fallenden Flanke außerhalb des Synchronisationssegments die Taktfrequenz des Oszillators zu erhöhen bzw. zu vermindern, sodass die nächste fallende Flanke innerhalb des Synchronisationssegments auftritt.

Im Folgenden wird die Entwickelung des Regelssystems und die damit verbundene Theorie genauer erläutert.

5.1. Der geschlossene Regelkreis

Der einfache Standardregelkreis, welcher im nachfolgenden Blockschaltbild dargestellt wird (Abbildung 59), besteht aus einem Regler, einer Regelstrecke oder einem Prozess sowie einer negativen Rückkopplung der Regelgröße **y**. Die Störgröße **d** wirkt sich in den meisten Fällen auf den Ausgang der Regelstrecke aus. Jedoch kann sie auch verschiedene Teile der Regelstrecke beeinflussen.

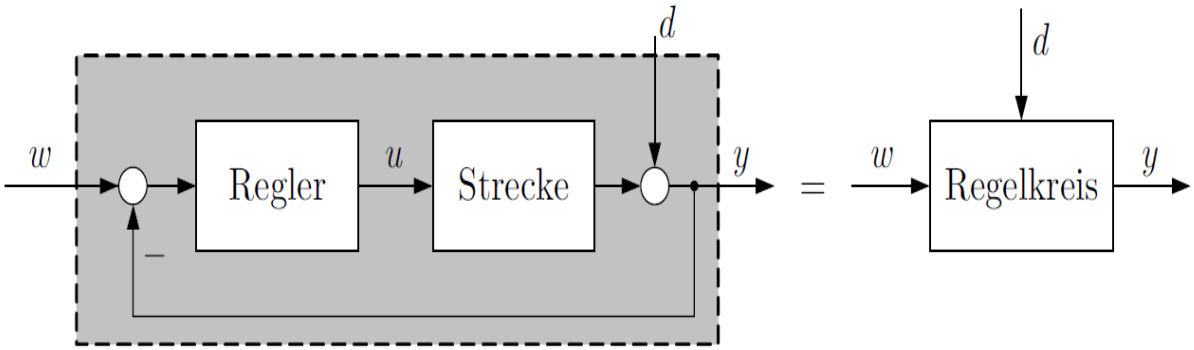

Abbildung 59: Standardregelkreis [7]

Grundprinzip einer einfachen Regelung ist die negative Rückkopplung der Regelgröße **y**. Aus ihr und der Führungsgröße wird durch Subtraktion die Regelabweichung abgebildet, bzw. die Regelgröße **y** wird mit der Führungsgröße (Sollwert) **w** verglichen.

$$\boldsymbol{e = w - y} \tag{3.1.25}$$

Die Regelabweichung **e** wird dem Regler zugeführt, welcher daraus entsprechend der gewünschten Dynamik des Regelkreises eine Stellgröße **u** bildet.

Eine der wichtigsten Anforderungen an den Regelkreis sind Störkompensation und Sollwertfolge. Sie werden erreicht, indem die Regelgröße der Führungsgröße für eine vorgegebene Klasse von Stör- und Führungssignalen asymptotisch folgen soll:

$$\boldsymbol{\lim_{t\to\infty}\big(w(t) - y(t)\big) = 0 \;\rightarrow\; \lim_{t\to\infty} e(t) = 0} \tag{3.1.26}$$

Die Stabilität ist eine besonders relevante Eigenschaft für technische Systeme. Nur wenn das System stabil ist, kann es über eine längere Zeit und unter der Einwirkung von Störungen betrieben werden. Ist es instabil, so muss es durch eine Regelung stabilisiert werden.

Das zu entwickelnde Regelsystem sollte zudem weitere Anforderungen erfüllen:

Der DCS-Computer im CERN-Leitwarte sendet Botschaften an den DCS-Chip im Pixeldetektor über das CAN-Netzwerk und verwendet hierfür eine Übertragungsrate von $125\,Kbit/s$,

welche im MOPS Chip von einem 10 Mhz Takt abgeleitet wird. Aus diesem Grund soll der Regelalgorithmus die aus der Regelabweichung berechnete Stellgröße, in unserem Fall FTRIM<4:0> Control-Bits, auf die Regelstrecke einwirken, sodass der analoge Taktgenerator ein Taktsignal mit einer Schaltfrequenz von circa 10 MHz erzeugt. Dieser regelungstechnische Eingriff ist aufgrund der in Abschnitt 3.2.3 ausgedrückte mathematische Beziehung 3.1.24 realisierbar.

Ein weiterer relevanter Punkt für die Frequenzregelung ist die zulässige Oszillatortolleranz. Wie bereits im Kapitel 2 erwähnt, ist eine Synchronisation durch Verlängerung oder Verkürzung von Phasensegmenten immer nach einer fallenden Flanke möglich. Der „Worst-Case"-Fall ist eine fallende Flanke nach fünf dominanten und fünf rezessiven Bits. In diesem Fall liegen zehn Bits zwischen zwei fallenden Flanken, welche zur Synchronisation geeignet sind. Längere Abstände sind nicht möglich, weil der CAN-Controller über den Bit-Stuffing-Mechanismus verfügt. Wenn eine Korrektur um die Synchronisationssprungweite T_{SJW} nach zehn Bitzeiten T_{bit} noch möglich sein soll, dann kann die Oszillatortolleranz wie folgt ermittelt werden:

$$\boldsymbol{10T_{bit} * df \leq |T_{SJW}|} \tag{3.1.27}$$

Wobei $T_{SJW} = SJW * T_q$ und T_q Time-Quantum entspricht.

Durch Auflösung nach df folgt die Regel:

$$\boldsymbol{df \leq \left|\frac{T_{SJW}}{10T_{bit}}\right|} \tag{3.1.28}$$

Zur Berechnung der Oszillatortoleranzen ist zudem zu berücksichtigen, dass ein Bit in $10\ T_q$ unterteilt ist und $T_{SJW} = 3\ T_q$. Daraus ergibt sich:

$$\boldsymbol{df \leq \left|\frac{3}{100}\right| \leftrightarrow -0,03 \leq df \leq 0,03} \tag{3.1.29}$$

Somit darf die Regelgröße, in unserem Fall Taktfrequenz des Oszillators, nur um $\pm 3\%$ variieren.

Die Regelungsaufgabe besteht im Entwurf und der Realisierung eines Regelungssystems, mit dem der Regelkreis die an ihn oben aufgeführten Anforderungen erfüllt.

Im folgenden Abschnitt wird das High-Level Blockschaltbild, welches eine Gesamtübersicht des Regelsystems enthält, vorgestellt.

5.2. Top-Modul des Regelsystems

Das Regelsystem besteht aus vier hierarchisch strukturierten Komponenten (Abbildung 60), welche über Schnittstellensignale miteinander kommunizieren, einen Zustandsautomaten, der den Bus auf fallende Flanken überwacht, einen Taktzähler, der den vom analogen Oszillator erzeugten Puls zählt, ein Phasenfehler Register, das die Regelabweichung berechnet und dem PID-Regler zugeführt wird. Dieser berechnet daraus die Stellgröße und sendet diese über FTRIM<4:0> Control-Bits an den analogen Relaxationsoszillator.

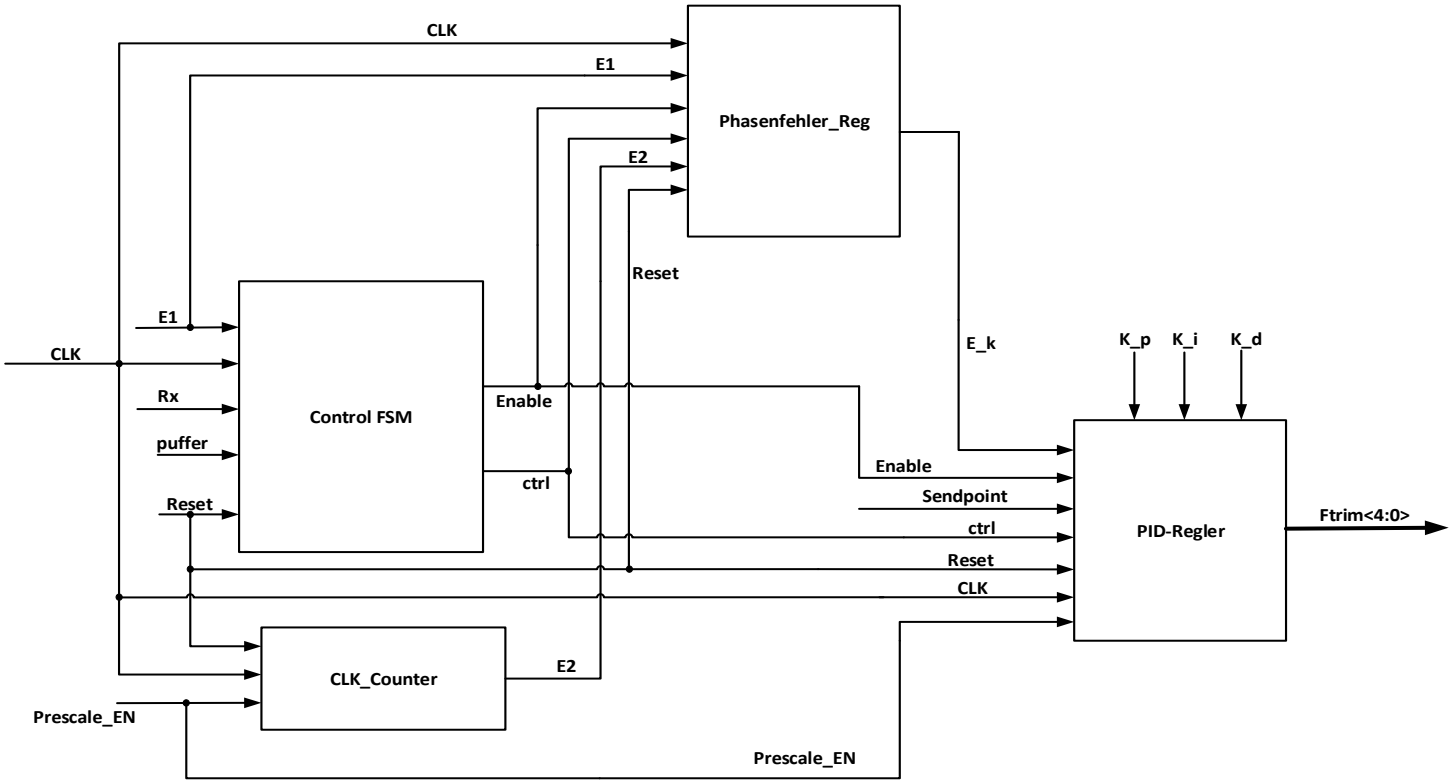

Abbildung 60: Top Entity Des Regelsystem

Dem Regelsystem werden die Signale CLK, Reset, der an das Bittiming-Modul eingehende Buswert Rx, puffer, Prescale_EN, Sendpoint und E1 von der Bittiming-Einheit zugeführt, wobei E1 dem Zählerstand entspricht. Das Clock-Signal wird vom analogen Oszillator geliefert.

Nachdem eine grobe Übersicht des Systems besteht, werden als nächstes die einzelnen Funktionalitäten in Modulen zerlegt und mit verschiedenen Entwurfsmethoden entwickelt. Jedes Modul erfüllt eine Teilaufgabe des kompletten Regelsystems und wird durch lokale Signale mit anderen Modulen verbunden. So ergibt sich ein strukturiertes Gesamtsystem.

5.2.1. Control-FSM-Modul

In der Abbildung 61 wird das Zustandsübergangsdiagramm mit den verwendeten Signalen dargestellt. Bei Start und Reset des CAN-Controllers beginnt der Automat im Zustand Reset. Nachdem die Signale Enable und ctrl in diesem Zustand zurückgesetzt wurden, geht der Zustandsautomat aus dem Zustand Reset in den Zustand Normal über. In diesem Zustand wird auf eine fallende Flanke gewartet und zu jedem Takt den Zähler E2 inkrementiert. Wird eine fallende Flanke außerhalb des Synchronisationssegments erkannt, wird in Abhängigkeit des Bittiming Zählerstand (E1) entschieden, in welchem Zustand die FSM als Nächstes übergeht. Tritt die Fehlerflanke während eines Zählerstandes ein, der zwischen null und fünf liegt, wird im Zustand PosPhasFehler die Taktperiode erhöht, in dem das Signal Enable gesetzt und dem Register ctrl der Wert eins zugewiesen wird. Ist der Zählerstand (E1) jedoch größer oder gleich sechs, dann wird im Zustand NegPhasFehler die Taktperiode vermindert, in dem der Wert zwei dem Register ctrl zugeordnet und das Enable Signal gesetzt wird.

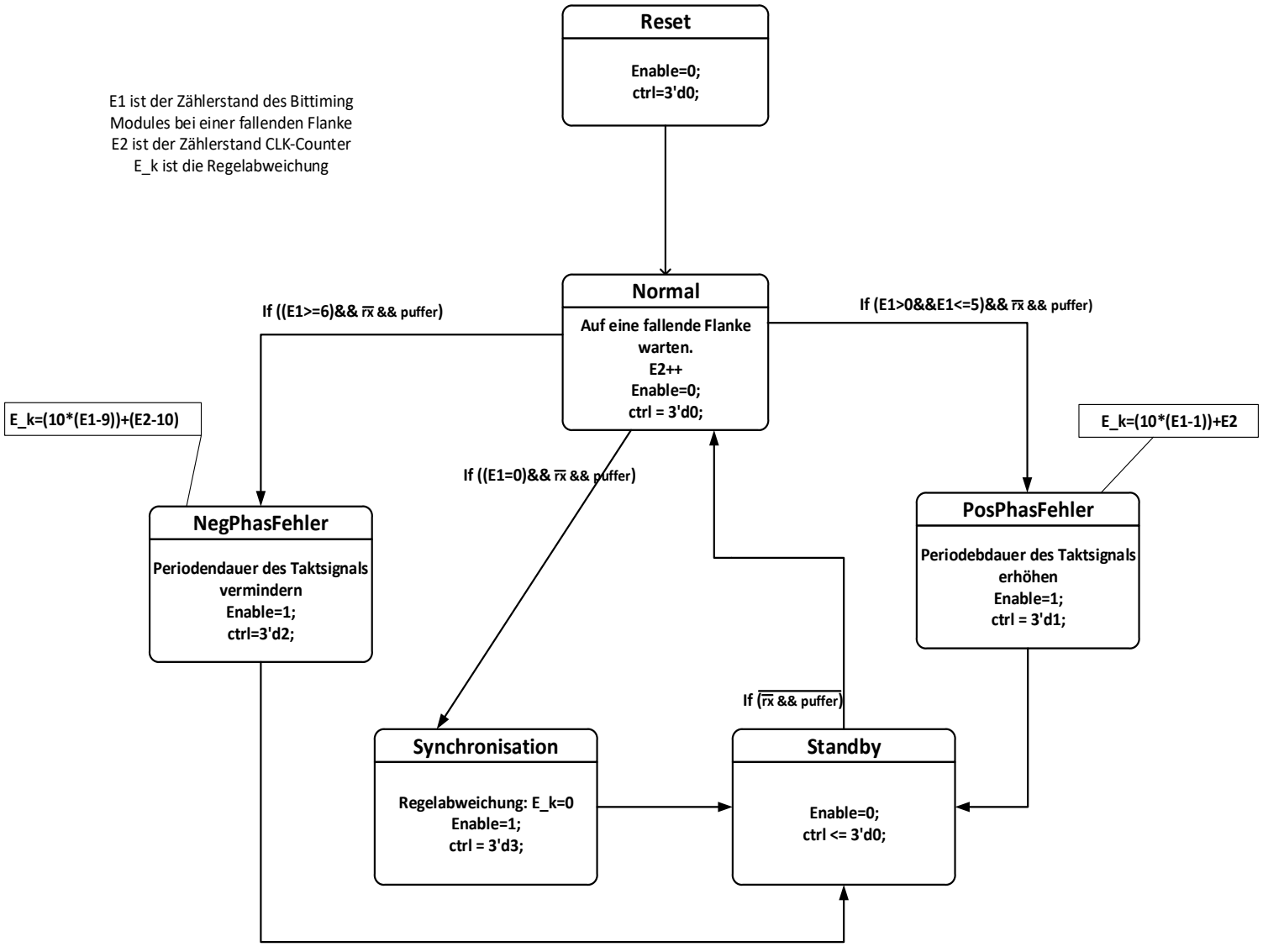

Abbildung 61: Zustandsübergangsdiagramm des Regelungzustandsautomats

Wird eine fallende Flanke detektiert, während der Zähler den Wert null beinhaltet, springt die FSM in den Zustand Synchronisation. In diesem Zustand wird keine Erhöhung bzw. Verminderung der Taktperiode eingeführt, indem das Enable Signal gesetzt und dem ctrl Register der Wert drei zugeordnet wird. Nachdem die Signale in den drei oben genannten Zuständen gesetzt wurden, verzweigt die FSM in den Zustand Standby und wartet bis die Bedingung (Rx=1‘b0 && puffer=1’b1) nicht mehr erfüllt ist, dann springen Sie zurück in den Zustand Normal.

5.2.2. CLK_Counter Modul

Das CLK_Counter-Modul zählt die vom analogen Relaxationsoszillator erzeugten Pulse und bestimmt anhand dieser die entsprechende Position der fallenden Flanke innerhalb eines Time-Quantums. Das Pre_Out-Signal wird vom Prescale-Modul geliefert und entspricht dem Vorteiler des Takt-Generator-Signals. Durch das Pre-out-signal wird das Time-Quantum abgeschlossen, wodurch der Zähler zurückgesetzt wird.

Das Modul leitet den Zählerstand, bei welchem eine fallende Flanke erkannt worden ist, an das PhasenfehlerReg-Modul weiter.

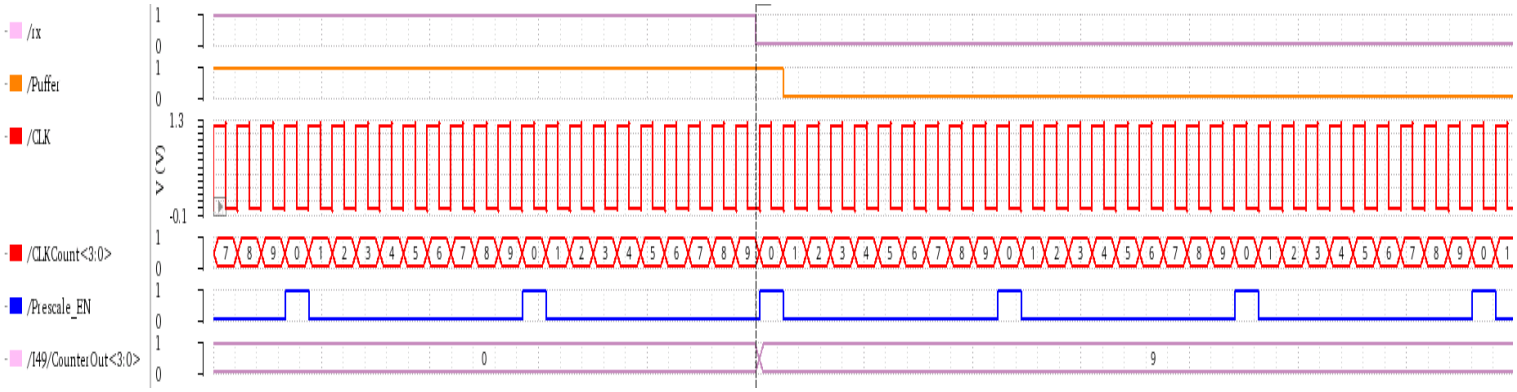

Abbildung 62: Signalverlauf des CLK-Counter-Moduls

In der in Abbildung 62 dargestellten Simulation ist eine fallende Flanke zu erkennen, welche bei einem Taktzählerstand neun aufgetreten ist. Der Zählerwert wird in einem internen Register gespeichert und über den Ausgang CLKCount<3:0> an das PhasenfehlerReg-Modul übermittelt.

5.2.3. PhasenfehlerReg Modul

Das PhasenfehlerReg-Modul dient dazu, die Regelabweichung aus den Werten E1 und E2 zu berechnen. E1 und E2 entsprechen dem Zählerstand der Bittiming-Einheit bzw. dem Taktzähler, bei welchem eine fallende Flanke eingetreten worden ist. Dieses Modul wird über die von FSM erzeugten Signale Enable und ctrl angesteuert. Soweit Enable gesetzt ist, wird die Regelabweichung $E(k)$ in Abhängigkeit des Inhalts ctrl<2:0> ermittelt.

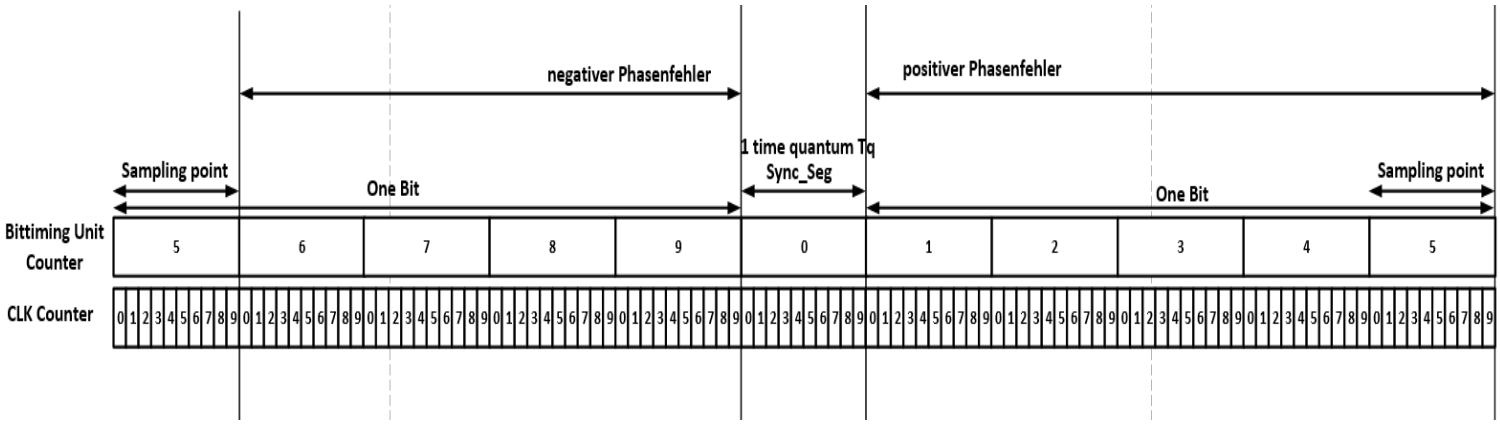

Abbildung 63: Aufteilung ein Bit

Die Regelabweichung gibt Auskunft über die Entfernung der Fehlerflanke vom Synchronisationssegment. Die oben aufgeführte Abbildung 63 veranschaulicht die Aufteilung eines Bits in Zeitsegmente, bzw. in Taktpulse. Als Berechnungsgrundlage hierzu dienen die untenstehenden Gleichungen:

$$\boldsymbol{E\_k} = \begin{cases} \big(\mathbf{10} * (\boldsymbol{E1} - \mathbf{1})\big) + \boldsymbol{E2} & \boldsymbol{falls\ \ ctrl} < \mathbf{2{:}0} >= \mathbf{3'}\boldsymbol{d1} \\ \big(\mathbf{10} * (\boldsymbol{E1} - \mathbf{9})\big) + (\boldsymbol{E2} - \mathbf{10}) & \boldsymbol{falls\ \ ctrl} < \mathbf{2{:}0} >= \mathbf{3'}\boldsymbol{d2} \\ \mathbf{0} & \boldsymbol{falls\ \ ctrl} < \mathbf{2{:}0} >= \mathbf{3'}\boldsymbol{d3} \end{cases} \tag{3.1.30}$$

5.2.4. PID-Regler

Der PID-Regler wird aufgrund seiner Einfachheit in der Industrie immer noch häufig eingesetzt. Es ist nicht erforderlich, ein konkretes mathematisches Modell der Regelstrecke aufzustellen, um den Regler-Entwurf durchzuführen. Der Regler wird lediglich durch die Anpassung der 3 Verstärkungen parametriert, sodass die an den Regelkreis gestellten Forderungen erfüllt werden.

Die Aufgabe eines Reglers besteht darin, bei Abweichungen die Stellgröße $u(t)$ so zu verändern, dass der Betrag der Regelabweichung $e(t)$ sich verringert oder im Idealfall auf null gebracht wird. Anders gesagt, die Regelabweichung soll so schnell wie möglich so klein wie möglich gemacht werden. Der PID-Regler setzt sich aus einem proportionalwirkenden P, einem integralwirkenden I und einem differenzierenden D Anteil zusammen (siehe Abbildung 64).

Der kontinuierliche zeitliche Verlauf der Stellgröße $u(t)$ ergibt sich:

$$\boldsymbol{u(t)} = \boldsymbol{K_R e(t)} + \frac{\boldsymbol{K_R}}{\boldsymbol{T_I}} \int_{\mathbf{0}}^{\boldsymbol{t}} \boldsymbol{e(t)dt} + \boldsymbol{K_R T_D} \frac{\boldsymbol{de(t)}}{\boldsymbol{dt}} \tag{3.1.31}$$

Wobei K_R dem Verstärkungsfaktor, T_I und T_D Zeitkonstanten entsprechen.

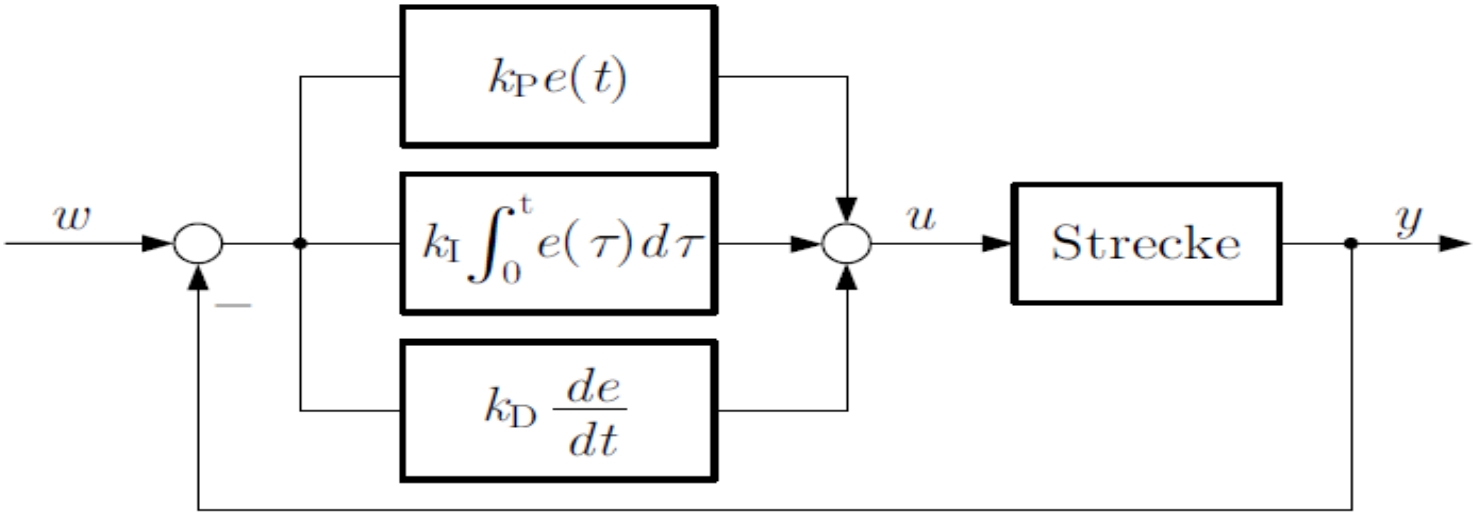

Abbildung 64: Struktureller Aufbau eines PID-Reglers

Die Gleichung 3.1.31 wird Laplace transformiert, um daraus die Übertragungsfunktion des PID-Reglers zu erhalten:

$$\boldsymbol{U(s) = K_R E(s) + \frac{K_R}{(T_I * s)} E(s) + K_R (T_D * s) E(s)} \qquad (3.1.32)$$

Die Parallelschaltung der oben genannten P-, I- und D-Anteil in s-Bereich ergibt:

$$\boldsymbol{G_{PID}(s) = \frac{U(s)}{E(s)} = P + \frac{I}{s} + Ds} \qquad (3.1.33)$$

Wobei $P = K_R$, $I = \frac{K_R}{T_I}$, $D = K_R T_D$

Zur Realisierung des digitalen PID-Reglers muss die zeitkontinuierliche Übertragungsfunktion $\boldsymbol{G_{PID}(s)}$ des analogen Reglers (Gleichung 3.1.33) in eine zeitdiskrete Übertragungsfunktion $\boldsymbol{G_{PID}(z)}$ transformiert werden. Die Transformation von der s-Ebene in die z-Ebene erfordert eine Äquivalenzbeziehung, die den Zusammenhang zwischen s und z darstellt. Für diese Anwendung wird das Euler Rückwärts numerische Verfahren verwendet:

$$\boldsymbol{s = \frac{z - 1}{z T_s}} \qquad (3.1.34)$$

Die Diskretisierung der zeitkontinuierliche Übertragungsfunktion $\boldsymbol{G_{PID}(s)}$ erfolgt durch den Einsatz der oben erwähnten Transformationsbeziehung und ergibt die folgende Übertragungsfunktion:

$$\begin{aligned} \boldsymbol{G_{PID}(z)} &= \boldsymbol{P + \frac{zT_s}{z-1} I + \frac{z-1}{zT_s} D} \\ &= \boldsymbol{P + \frac{z}{z-1} T_s I + \frac{z-1}{z} \frac{D}{T_s}} \end{aligned}$$

$$\boldsymbol{G_{PID}(z) = \frac{\left(P + T_s I + \frac{D}{T_s}\right) z^2 + \left(-P - 2\frac{D}{T_s}\right) z + \frac{D}{T_s}}{z^2 - z}} \qquad (3.1.35)$$

Wobei $\boldsymbol{T_s}$ die Abtastperiode ist.

Die Gleichung 3.1.35 kann wie folgt vereinfacht werden:

$$\boldsymbol{G_{PID}(z) = \frac{(K_P + K_I + K_D)z^2 + (-K_P - 2K_D)z + K_D}{z^2 - z} = \frac{U(z)}{E(z)}} \qquad (3.1.36)$$

Wobei $\boldsymbol{K_P = P}$, $\boldsymbol{K_I = T_S I}$ und $\boldsymbol{K_D = \frac{D}{T_s}}$

Mit Gleichung 3.1.36 folgt durch kreuzweises Multiplizieren der Signalgrößen die Beziehung:

$$\boldsymbol{U(z)(z^2 - z) = \left(K_p + K_i + K_d\right)E(z)z^2 + \left(-K_p - 2K_d\right)E(z)z + K_d E(z)} \qquad (3.1.37)$$

Als nächstes wird die Gleichung mit den Zustandsvariablen $\boldsymbol{z^{-1}}$ und $\boldsymbol{z^{-2}}$ erweitert. Aufgrund der Beziehung zwischen dem Laplace-Operator **s** und dem Z-Operator **z**

$$\boldsymbol{z = e^{sT}} \qquad (3.1.38)$$

Bzw.

$$\boldsymbol{z^{-1} = e^{-sT}} \qquad (3.1.39)$$

kann festgehalten werden, dass eine Multiplikation mit $\boldsymbol{z^{-1}}$ eine Verschiebung um die Abtastzeit in die Vergangenheit beinhaltet und in einem digitalen System einen Zustandsspeicher darstellt, also den Bezug auf die Vergangenheit des Wertes.

$$\boldsymbol{U(z) - U(z)z^{-1} = \left(K_p + K_i + K_d\right)E(z) + \left(-K_p - 2K_d\right)E(z)z^{-1} + K_dE(z)z^{-2}} \quad (3.1.40)$$

Dann ergibt sich daraus die folgende Differenzengleichung für die Stellgröße $U(k)$:

$$\boldsymbol{U(k) = U(k-1) + \left(K_p + K_i + K_d\right)E(k) + \left(-K_p - 2K_d\right)E(k-1) + K_dE(k-2)} \quad (3.1.41)$$

Die Gleichung 3.1.41 enthält die Abtastwerte $\boldsymbol{U(k-1)}$, $\boldsymbol{E(k-1)}$ und $\boldsymbol{E(k-2)}$, welche sie in der Vergangenheit liegenden Werte für die Stellgröße U und die Regelabweichung E darstellen.

Im Folgenden werden die rekursiven Ausdrücke $\boldsymbol{U(n)}$, wobei $\boldsymbol{n \in \{1, 2, 3, \dots, k-1\}}$, aus den vorhergehenden Werten der Stellgröße und der Regelabweichung berechnet:

$$\boldsymbol{U(k-1) = U(k-2) + \left(K_p + K_i + K_d\right)E(k-1) + \left(-K_p - 2K_d\right)E(k-2) + K_dE(k-3)}$$

$$\boldsymbol{U(k-2) = U(k-3) + \left(K_p + K_i + K_d\right)E(k-2) + \left(-K_p - 2K_d\right)E(k-3) + K_dE(k-4)}$$

$$\boldsymbol{U(k-3) = U(k-4) + \left(K_p + K_i + K_d\right)E(k-3) + \left(-K_p - 2K_d\right)E(k-4) + K_dE(k-5)}$$

….

$$\boldsymbol{U(2) = U(1) + \left(K_p + K_i + K_d\right)E(2) + \left(-K_p - 2K_d\right)E(1) + K_dE(0)}$$

$$\boldsymbol{U(1) = U(0) + \left(K_p + K_i + K_d\right)E(1) + \left(-K_p - 2K_d\right)E(0)}$$

Daraus ergibt sich die folgende mathematische Darstellung:

$$\boldsymbol{U(k) = U(0) + \left(K_p + K_i + K_d\right)\sum_{n=0}^{k} E(n) + \left(-K_p - 2K_d\right)\sum_{n=0}^{k-1} E(n) + K_d\sum_{n=0}^{k-2} E(n)} \quad (3.1.42)$$

Als Nächstes werden die Faktoren $\boldsymbol{K_p}$, $\boldsymbol{K_i}$, $\boldsymbol{K_d}$ in der Gleichung 3.1.42 ausgeklammert.

$$\begin{aligned}\boldsymbol{U(k) = U(0) + K_p\left(\sum_{n=0}^{k} E(n) - \sum_{n=0}^{k-1} E(n)\right) + K_i\left(\sum_{n=0}^{k} E(n)\right)} \\ \boldsymbol{+ K_d\left(\sum_{n=0}^{k} E(n) - 2\sum_{n=0}^{k-1} E(n) + \sum_{n=0}^{k-2} E(n)\right)}\end{aligned} \quad (3.1.43)$$

Die Gleichung 3.1.43 kann wie folgt vereinfacht werden:

$$\boldsymbol{U(k) = U(0) + K_p E(k) + K_i \sum_{n=0}^{k} E(n) + K_d (E(k) - E(k-1))} \tag{3.1.44}$$

Somit kann Für die Frequenzregelung folgende Differenzengleichung aufgestellt werden

$$\boldsymbol{U(k) = K_p E(k) + K_i I(k) + K_d D(k)} \tag{3.1.45}$$

Wobei

$$\begin{cases} \boldsymbol{U(0) = 0} \\ \boldsymbol{I(k) = \sum_{n=0}^{k} E(n) = I(k-1) + E(k)} \\ \boldsymbol{D(k) = E(k) - E(k-1)} \end{cases} \tag{3.1.46}$$

Abbildung 65 beschreibt die Struktur der Gleichung 3.1.45. Hierbei ähnlt die zeitdiskrete Darstellung der parallelgeschalteten proportionalwirkenden P, integralwirkenden I und differenzierenden D Glieder den Strukturellen Aufbau des PID-Reglers im zeitkontinuierlichen Bereich (Abbildung 64).

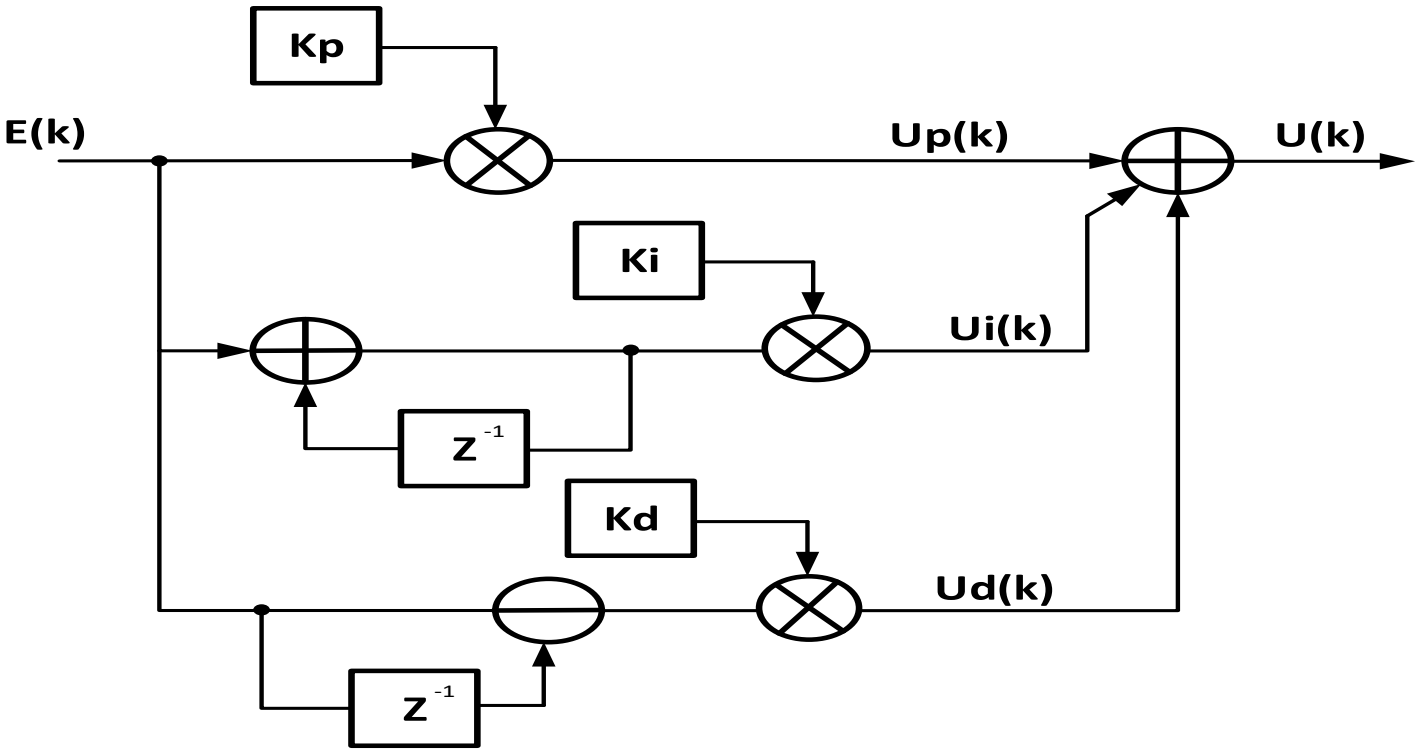

Abbildung 65: Signalflussgraph des PID-Reglers

Für die Realisierung des PID-Reglers wird das Festkomma-Format verwendet. Die Regler-Parameter werden mit einem Vorzeichen-Bit, drei Integer-Bits und Vier Fraktional-Bits implementiert und in der nachfolgenden binären Darstellung gespeichert (siehe Anhang 8.1):

$$\begin{cases} \boldsymbol{K_p = (x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 3, f = 4)} \\ \boldsymbol{K_i = (x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 3, f = 4)} \\ \boldsymbol{K_d = (x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 3, f = 4)} \end{cases} \tag{3.1.47}$$

Die Gleichung 3.1.45 beschreibt den Algorithmus des PID-Reglers im zeitdiskreten Bereich. Für die Verilog-Implementierung der Gleichung 3.1.45 werden ein Addierer und drei Multiplizierer benötigt. Darüber hinaus werden die Funktionen, welche für die Akkumulation $\boldsymbol{I(k)}$ und die Differenz $\boldsymbol{D(k)}$ der Regelabweichung zuständig sind, in einzelnen Komponenten realisiert. So ist es möglich, dass auch einzelne Module für weitere Aufgaben wiederverwendbar sind.

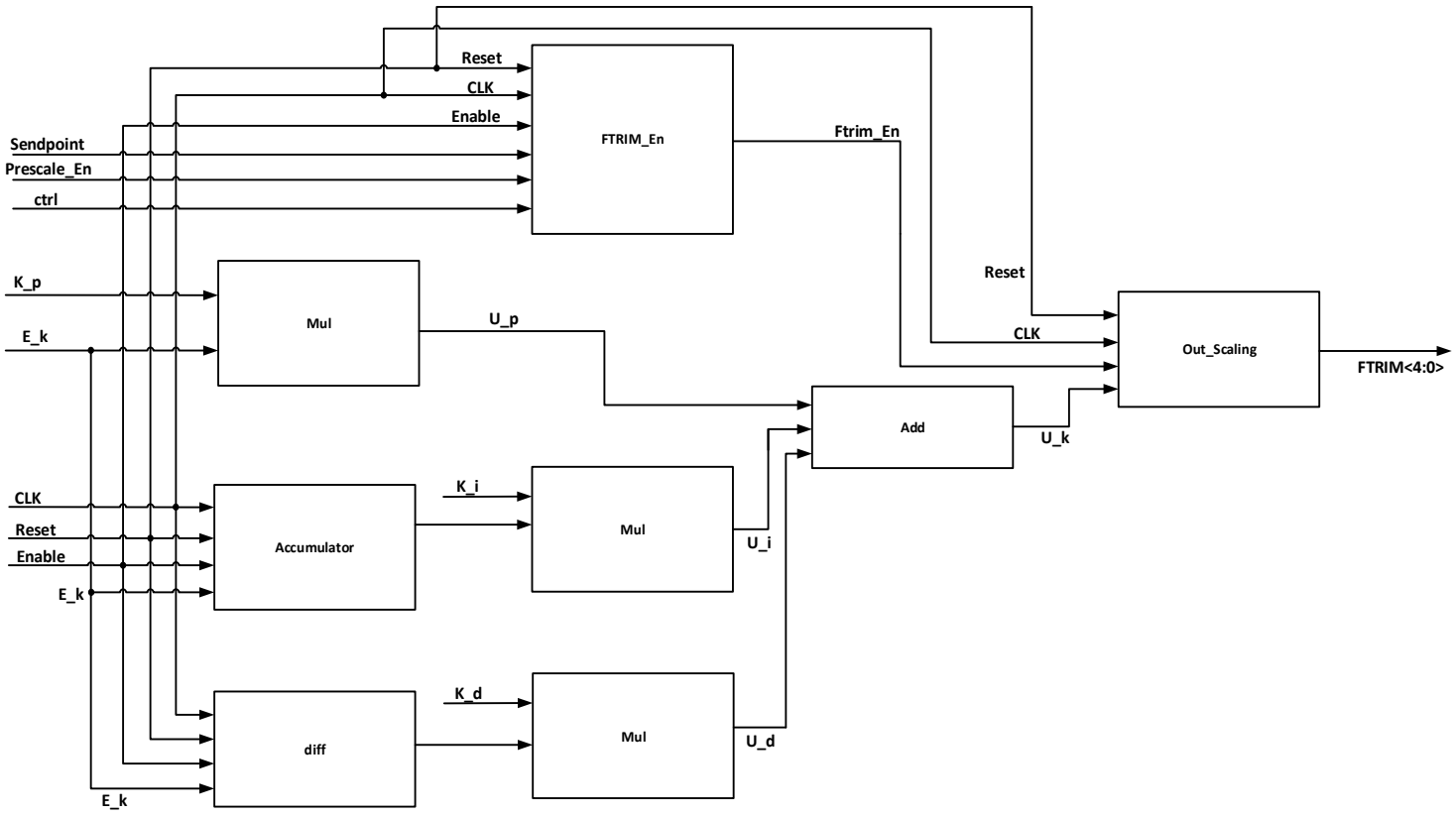

Abbildung 66: Aufbau Top-Level-Modul PID-Regler

Das Akkumulator-Modul stellt $I(k)$ aus der Gleichung 3.1.46 dar und wird durch das Signal Enable aktiviert. Ist Enable gesetzt, wird die aktuelle Regelabweichung $E(k)$ mit den in der Vergangenheit liegenden Abtastwerten zusammenaddiert. Das Additionsergebnis wird in einem internen Register gespeichert und über den Ausgang an das rein kombinatorische Mul-

Modul weitergeleitet. Anschließend wird $I(k)$ mit dem Regler-Parameter K_i multipliziert und über den Ausgang U_i an das Add-Modul übertragen.

Das Diff-Modul subtrahiert den im Delayregister $E(k-1)$ gespeicherten Wert mit dem aktuellen Abtastwert der Regelabweichung $E(k)$. Das Subtraktionsergebnis wird mit dem Regler-Parameter K_d multipliziert und durch das Signal U_d an das Add-Modul geliefert.

Die Multiplikationsergebnisse haben folgendes binäres Festkomma-Format (siehe Anhang 8.1):

$$\begin{cases} \boldsymbol{U_p(k) = (x_{10}, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 10, f = 4)} \\ \boldsymbol{U_i(k) = (x_{10}, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 10, f = 4)} \\ \boldsymbol{U_d(k) = (x_{10}, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 10, f = 4)} \end{cases} \tag{3.1.48}$$

wobei $E(k) = (x_7, \dots, x_0)_2 = (s = 1, i = 7, f = 0)$

Im Add-Modul wird die Regelgröße $U(k)$ des PID-Reglers aus der Addition der berechneten P-, I- und D-Anteile gebildet und dem Out-Scaling-Modul zugeführt.

Das Additionsergebnis aus dem Add-Modul sieht in binären Festkomma-Darstellung wie folgt aus:

$$\boldsymbol{U(k) = (x_{10}, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (s = 1, i = 10, f = 4)} \tag{3.1.49}$$

Im Out-Scaling-Modul werden die von dem Add-Modul übermittelten ersten 5 Vorkomma-Bit von den insgesamt fünfzehn Bit breite Stellgröße U(k) in einem Register gespeichert (reg<=U_k[8:4]) und durch den Ausgang an den analogen Oszillator übertragen. Ist der Integer-Teil von $U(k)$ (U_k[15:4]>31) größer als den Maximalwert von 31, wird der Maximalwert dem Register FTRIM<4:0> zugeordnet. Ist die übermittelte $U(k)$ negativ, wird der Wert null dem Register FTRIM<4:0> zugewiesen. Das Out_Scaling-Modul wird durch das FTRIM_En-Modul gesteuert, um die Freigabe der FTRIM<4:0> Control-Bits einzuleiten.

Das FTRIM_EN-Modul hat die Funktion, die vom PID-Algorithmus berechnete Stellgröße U(k) mit dem Beginn des nächsten Bits den Oszillator zu übermitteln. Die Aktualisierung der FTRIM-Controlbits geschieht erst nach vollständiger Durchführung des Synchronisation-Mechanismus. Die Verlängerung oder Verkürzung des Zeitsegments tseg sind vom Time-Quantum abhängig, welches vor dem Eintritt der fallenden Flanke (außerhalb des Synchronisationssegmentes) vorhanden war. Das Time-Quantum ist wiederum abhängig von der Periodendauer des Taktssignals, welches vom Oszillator erzeugt wird. Die Verkürzung, bzw. Verlängern des Tseg-Zeitsegments geschieht nach der Detektierung der fallenden Flanke, um Phasenfehler zu kompensieren. Von daher führt die sofortige Änderung der FTRIM-Controlbits dazu, dass die Periodendauer des Taktssignals sich erhöht, bzw. sich verringert und hat damit zur Folge, dass sich das Time-Quantum entsprechend verlängert, bzw. verkürzt. Dies kann zu einer Beeinträchtigung des Synchronisationsmechanimus führen - und damit zu einem verbundenen Restfehler. Durch diesen Restfehler wird der PID-Regler unnötig belastet, was

zu Schwingungen der Regelgröße um den Sollwert und somit zu einer Verlangsamung der Verfolgung des Sollwertes führen kann. Was wiederum zur Folge hat, dass der Regler mehr Ausregelzeit benötigt, um die gezielte Schaltfrequenz zu erreichen [T].

Die oben genannte These wird durch die in Kapitel 6.1 veranschaulichten Simulationsergebnisse bewiesen (siehe Abbildung 96).

5.2.5. Implementierung des Regelsystems in Virtuoso

In Abschnitt 4.2 wurde die Implementierung neuer Verilog-Module in der Entwurfssoftware Virtuoso bereits ausführlich behandelt, deshalb folgen hierzu an dieser Stelle keine weiteren Ausführungen.

Zunächst wird eine neue persönliche Bibliothek mit dem Namen Control_System erstellt, in welcher die Regelsystem-Verilog-Module angelegt werden (Abbildung 67).

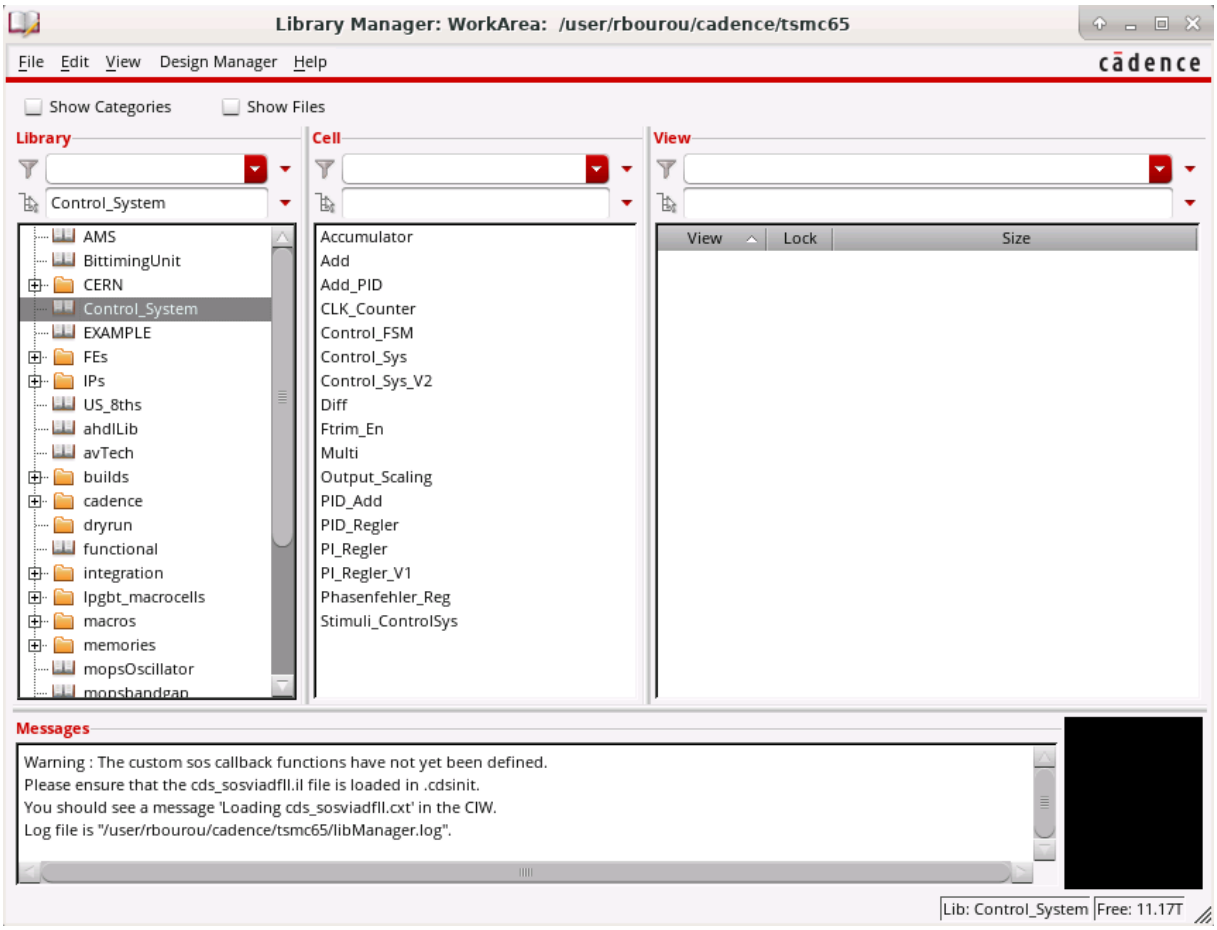

Abbildung 67: Die Control_System Bibliothek in Virtuoso Library Manager

Anschließend werden die einzelnen Module und die dazugehörigen Symbole, die das Regelsystem beschreiben, erstellt und in der Bibliothek **Control_System** abgespeichert.

Die Erzeugung der Verilog-Zellen ist den Ausführungen des Kapitels 4 zu entnehmen.

Darauffolgend werden die einzelnen Komponenten, welche den PID-Regler beschreiben, in Schematic-Editor eingefügt und miteinander verdrahtet, um miteinander zu kommunizieren und Daten auszutauschen (Abbildung 68).

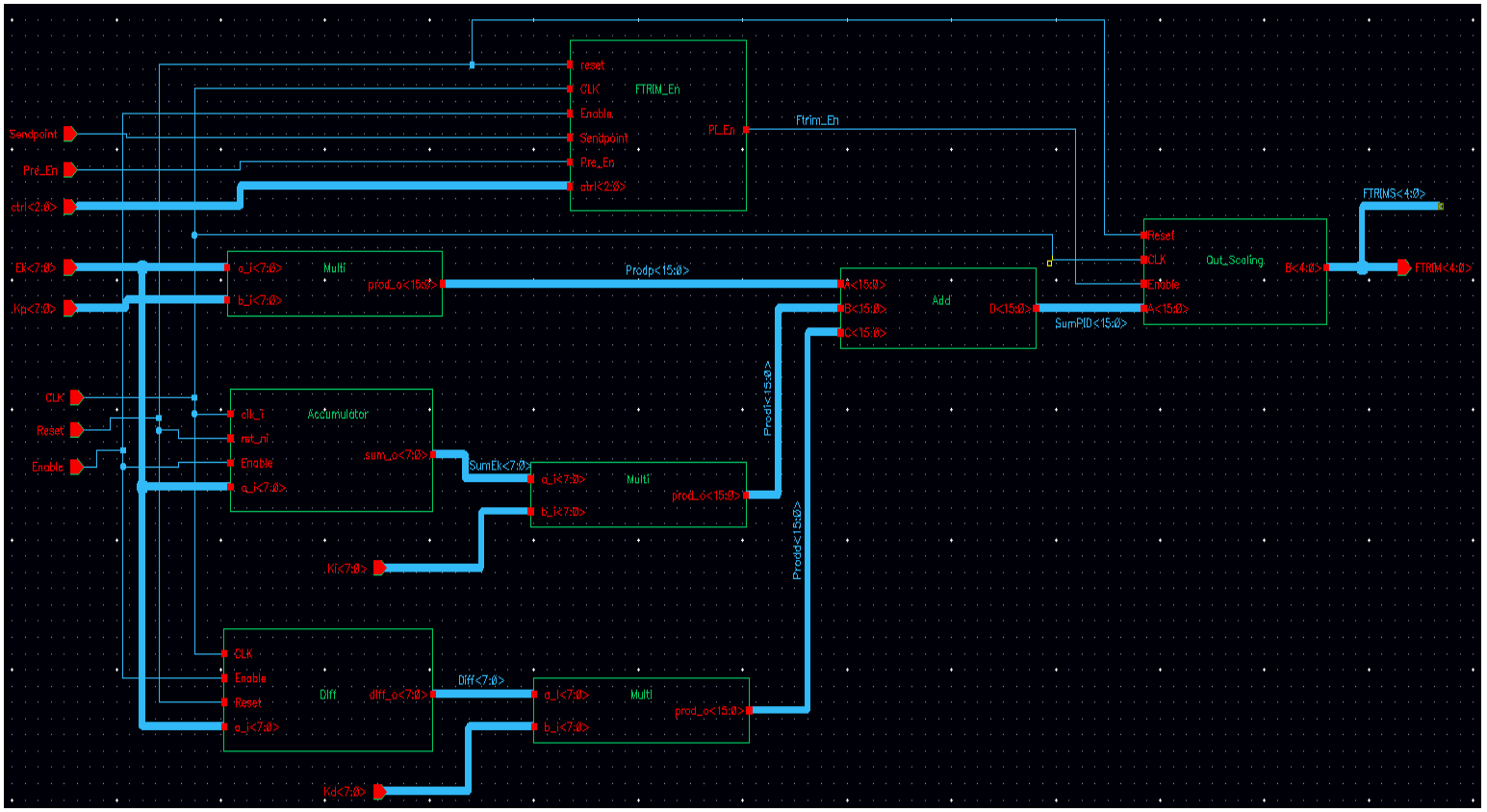

Abbildung 68: Top-Modul-PID-Regler in Virtuoso

Im Anschluss wird aus der Schematic-Ansicht des PID-Reglers ein erneutes Symbol generiert, welches im Top-Modul-Regelsystem verwendet wird (Abbildung 69).

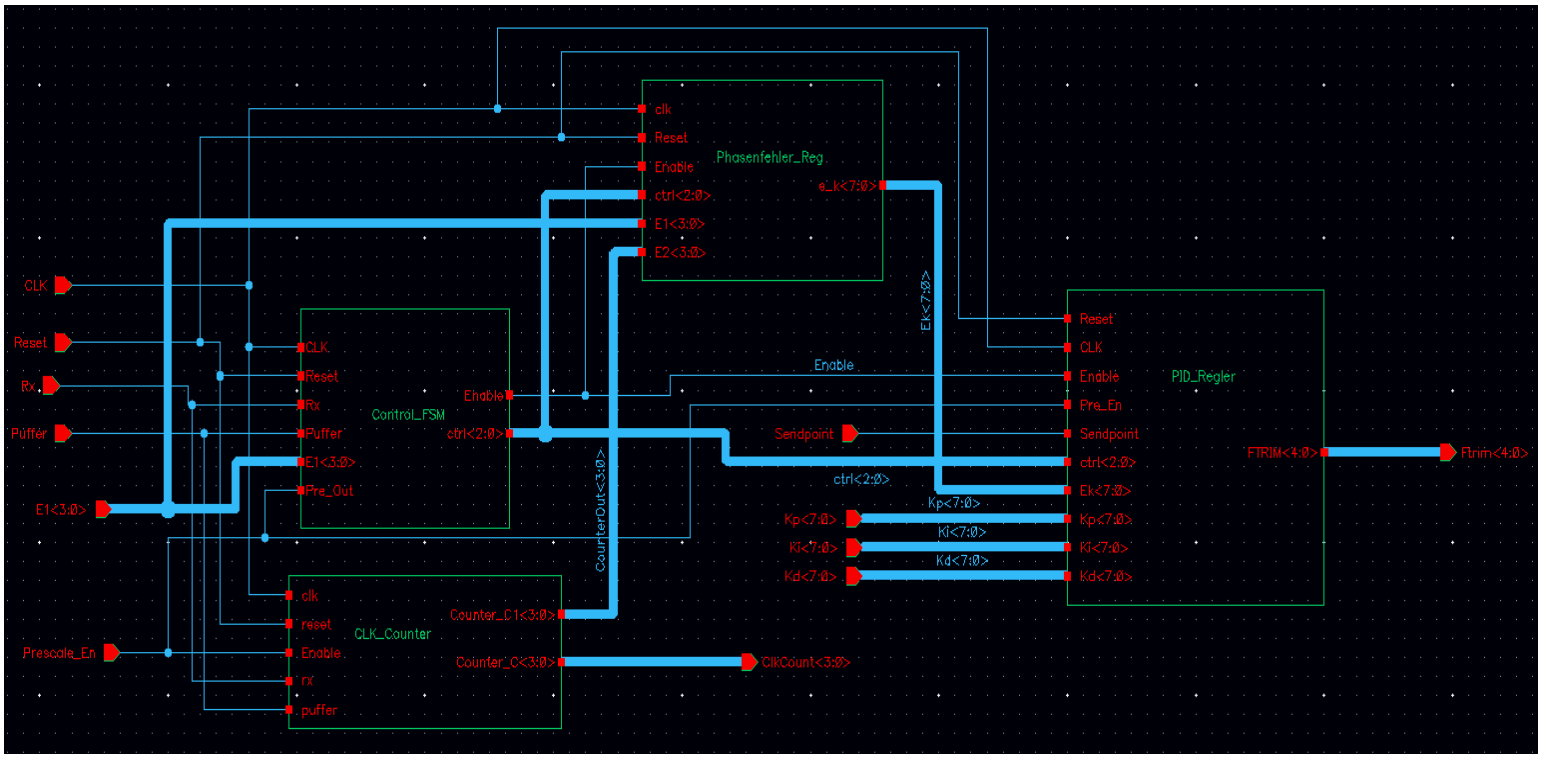

Abbildung 69: Top-Modul-Regelsystem in Virtuoso Schematic Edit-Fenster

Abschließend wird zu guter Letzt ein neues Symbol aus der Schematic-Ansicht des Top-Modul-Regelssystems erzeugt, welches in der Testschaltung im nächsten Kapitel verwendet wird.

6. Simulation des geschlossenen Regelkreises

Viele praktische Regelungsaufgaben sind dadurch gekennzeichnet, dass kein mathematisches Modell der Regelstrecke verfügbar ist und ein Regler gesucht wird, der gewisse Anforderungen zu erfüllen hat. Bei der folgenden Regelungsaufgabe wird so vorgegangen, dass der Regler an die Regelstrecke angeschlossen und mit Hilfe von A/MS-Simulationen nach günstigen Regler-Parametern gesucht wird. Die Simulationen dienen einerseits zur Beobachtung wichtiger dynamischer Eigenschaften der Regelstrecke und andererseits zur Beurteilung des Regelkreisverhaltens mit den gewählten Regler-Parametern. Darüber hinaus wird die Funktionalität des Regelsystems anhand von A/MS-Simulationen verifiziert.

Das Einrichten einer A/MS-Simulation in der Software Cadence Virtuoso ist den Ausführungen des Abschnitts 4.3 zu entnehmen.

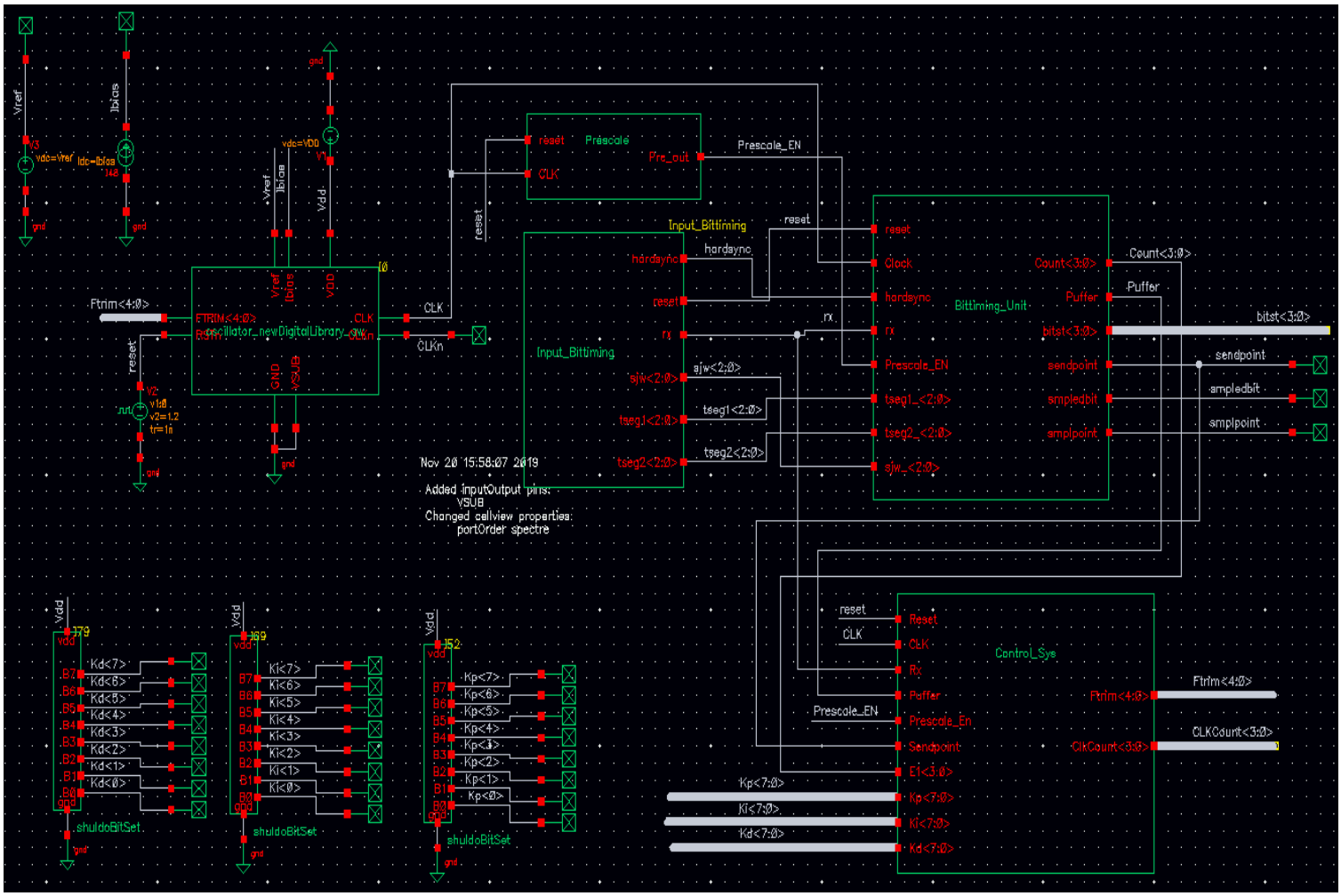

Abbildung 70: Testschaltung für die Simulation des geschlossenen Regelkreises

Die im Abschnitt 4.3 verwendete Testschaltung (Abbildung 34) wird mit dem Regelsystem erweitert und stellt somit den zu simulierenden geschlossenen Regelkreis dar. Um die Dauer

der AMS-Simulationsdurchführung zu minimieren, wird der Referenzgenerator durch ideale Strom- und Spannungsquellen ersetzt. Die Regler-Parameter werden in dem **schuldoBitSet** Verilog-A-Modul von Dezimal zu Binär umgewandelt und über die Signale $Kp < 7:0 >$, $Ki < 7:0 >$ und $Kd < 7:0 >$ an das Regelsystem übermittelt.

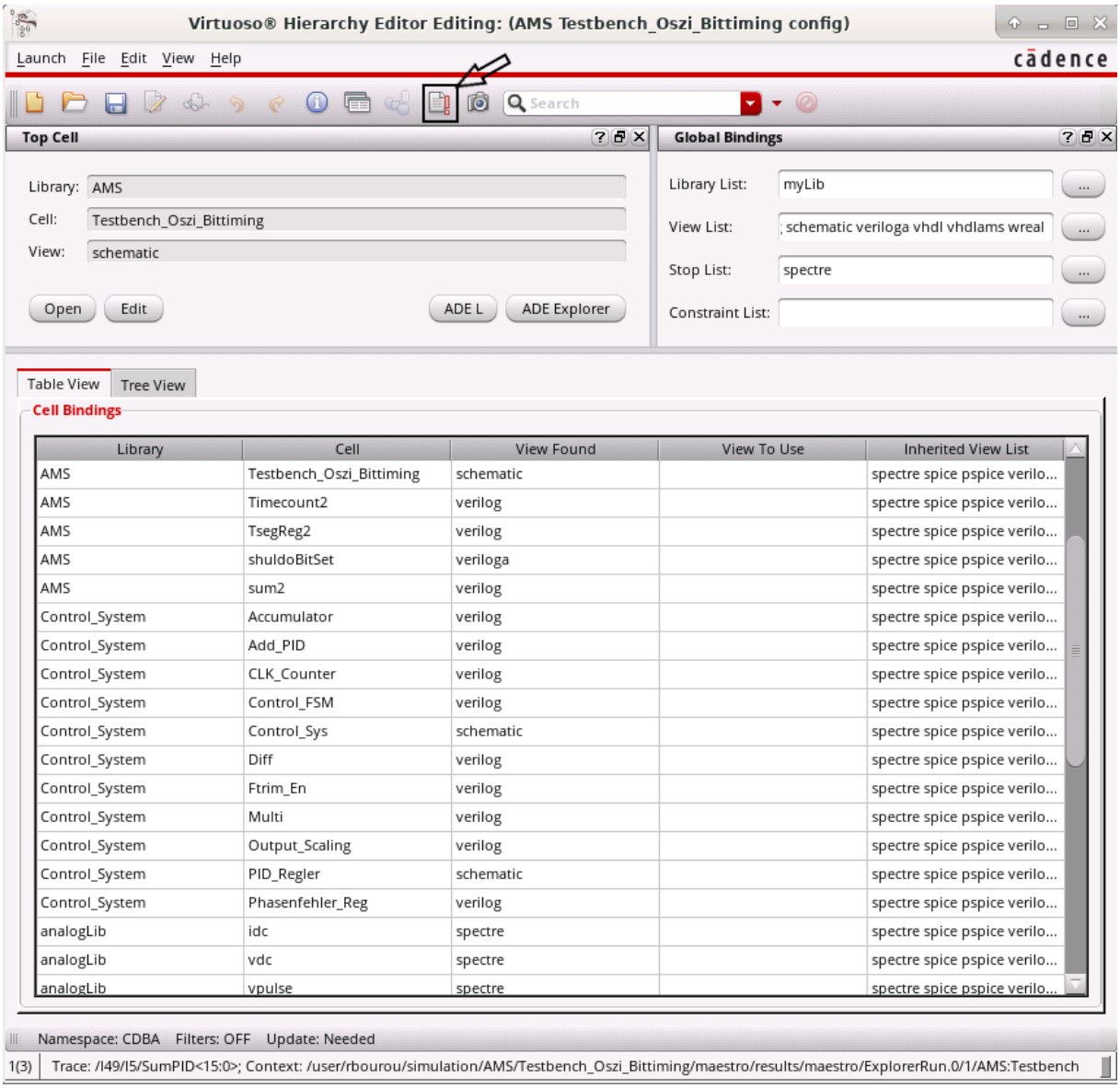

Abbildung 71: Das Hierarchie-Editor-Fenster

Nachdem in der Testschaltung neue Komponenten eingefügt wurden, muss die Konfiguration der AMS-Simulation editiert werden. Hierzu wird der Hierarchie-Editor geöffnet und durch Anklicken in der Menüleiste auf das rote Ausrufezeichen-Symbol „recompute the hierarchy“ werden die Liste der verwendeten Zellen und die Hierarchische Struktur der Top-Level-Schematic für den Netzlister aktualisiert (Abbildung 71).

Für die Durchführung der AMS-Simulation wird, wie bereits in Kapitel 4.3 beschrieben, der **ADE Explorer** verwendet. Dieses Tool lässt sich im Virtuoso Hierarchie-Editor mit dem Klick auf **ADE Explorer** Button aufrufen.

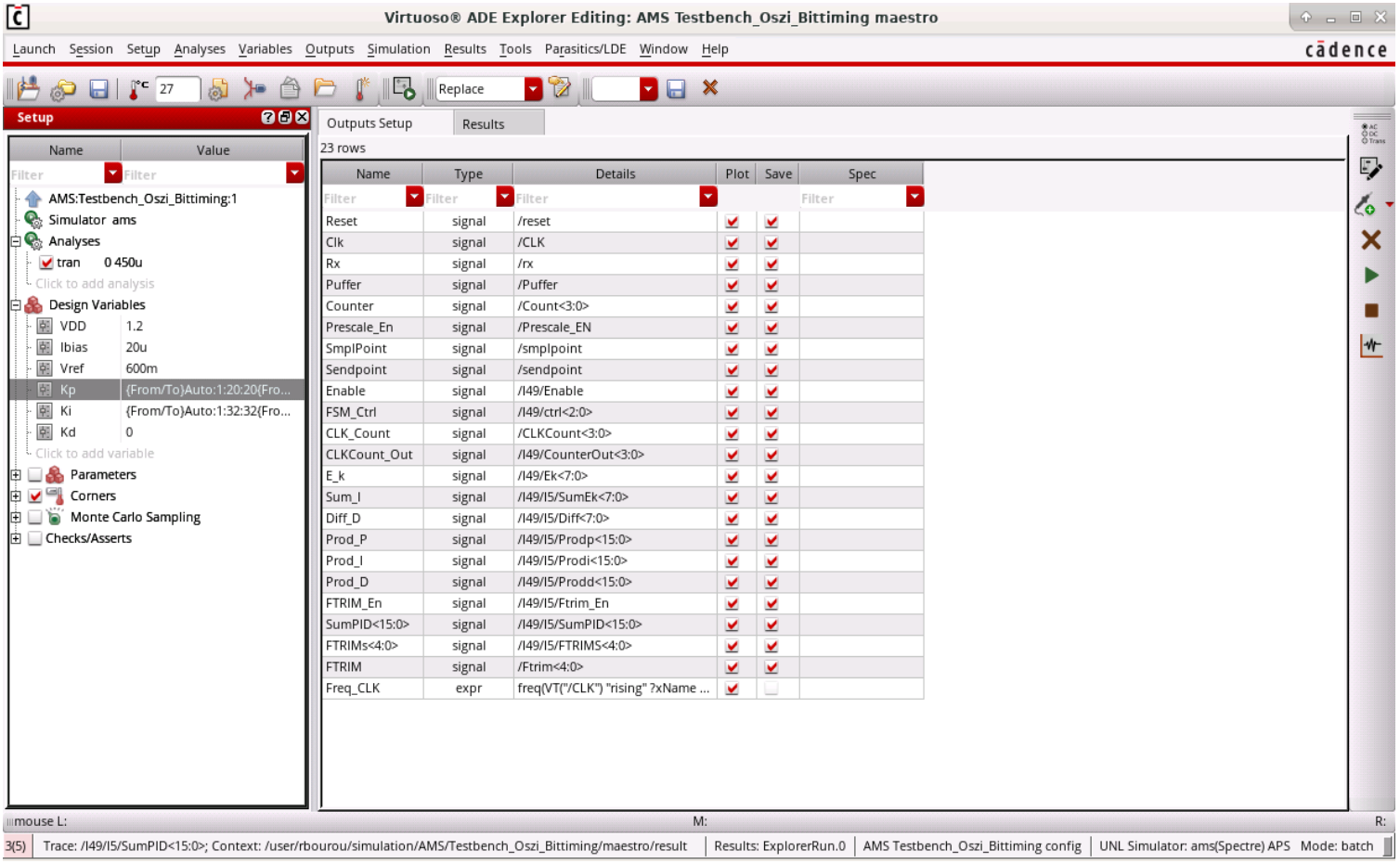

Abbildung 72: Simulationsumgebung ADE Explorer

Für die AMS-Simulation wird eine Transientenanalyse mit einer Simulationszeit von $450\ \mu s$ eingesetzt.

Damit die Regler-Parameter in einem Wertebereich verändert werden kann, werden sie als Variablen eingeführt, die in den Objekt-Eigenschaften des instanziierten **schuldoBitSet** Verilog-A-Moduls eingetragen werden (siehe Abbildung 73). Anschließend werden sie im Bereich Design Variables importiert. Darauffolgend werden Kp und Ki als Sweep-Variablen gewählt. Die Variable Kp bzw. Ki soll im Intervall von 1 bis 20 bzw. von 1 bis 32 mit einem Schritt von 1 durchlaufen.

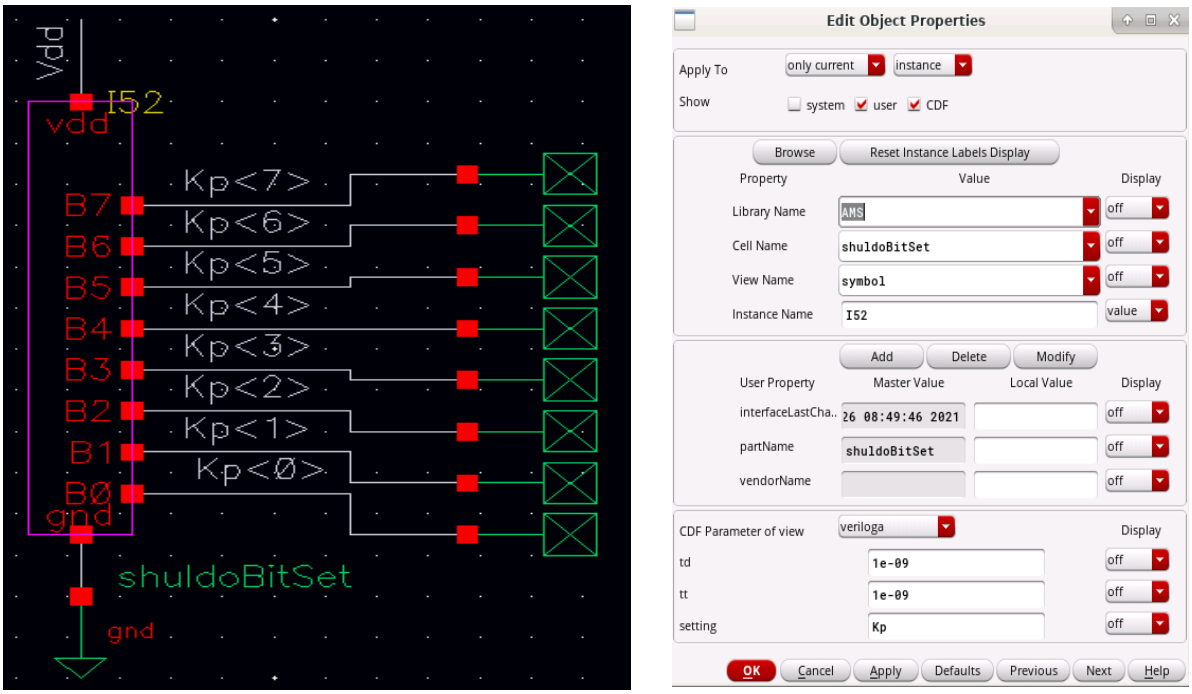

Abbildung 73: Das SchuldoBitSet Veriloga-Modul für den Regler-Parameter Kp

Die anderen Design-Variablen werden auf die folgenden Zahlenwerte gesetzt:

Variable	Wert
V_{DD}	1.2 V
I_{bias}	20 µA
V_{ref}	600 mV
K_d	0

Abschließend werden Ausgangssignale, welche simuliert bzw. dargestellt werden sollen, für die **Tran**-Simulation ausgewählt. Für die Regelgröße **Freq_CLK**, welche der Taktfrequenz des Oszillators entspricht, wird der aus Calculator nachfolgende Ausdruck als Output konfiguriert:

Freq_CLK	expr	freq(VT("/CLK") "rising" ?xName "time" ?mode "auto" ?threshold 0)

Zur Verifizierung des Regelalgorithmus und des geschlossenen Regelkreisverhaltens werden folgende Signale simuliert.

CLK	Taktsignal aus dem analogen Oszillator
Reset	Setzt das Bittiming- und das Regelsystem-Modul zurück
Rx	Das eingehende Signal an das Timing Modul
Puffer	Wird mit Rx Signal verglichen, um eine fallende Flanke zu detektieren

Prescaler_En	Taktsignal-Frequenzteiler Ausgangssignal, steuert die Zählereinheit
Count<3:0>	Zählerstand der Zeitsegmente
Sendpoint	Bezeichnet das Ende des Bitsegments
Enable	Wird gesetzt, wenn eine fallende Flanke auftritt und aktiviert den Regelalgorithmus.
FSM_Ctrl<2:0>	Zustand der Control-FSM
CLK_Count<3:0>	Zählerstand des Taktzählers
E_k<7:0>	Regelabweichung E(k)
Sum_I<7:0>	Akkumulator-Ausgangssignal
Diff<7:0>	Differenz-Ausgangssignal
Prod_P<15:0>	P-Anteil Ausgangssignal
Prod_I<15:0>	I-Anteil-Ausgangssignal
Prod_D<15:0>	D-Anteil-Ausgangssignal
SumPID<15:0>	Stellgröße des PID-Reglers
FTRIM_En	Wird gesetzt, wenn ein neues Bitsegment beginnt, nachdem eine fallende Flanke aufgetreten ist.
FTRIMs<4:0>	Steuersignale des Relaxationsoszillators

Für die A/MS-Simulation des Gesamtsystems wird ein sich periodisch wiederholendes Rechteckpuls-Signal mit einer Pulsfrequenz von $62{,}5\ KHz$ als eingehendes Rx-Signal sowie eine Abtastperiode von $20\ \mu s$ verwendet.

Nach der Auswahl der Ausgangssignale kann nun die AMS-Simulation durchgeführt werden. Dazu wird mit einem Klick auf das grüne Symbol **Run** in der rechten Symbolleiste oder über den Menüpunkt **Simulation→Netlist and Run** die Simulation gestartet.

6.1. Simulationsergebnisse

Nach Abschluss der A/MS-Simulation werden die Resultate des Simulationsdurchlaufes in der Simulationsumgebung unter dem Reiter Results gelistet. Es wird für jeden Sweep-Point die oben aufgeführten Outputs simuliert, was insgesamt 640 durchgeführten Simulationen entspricht.

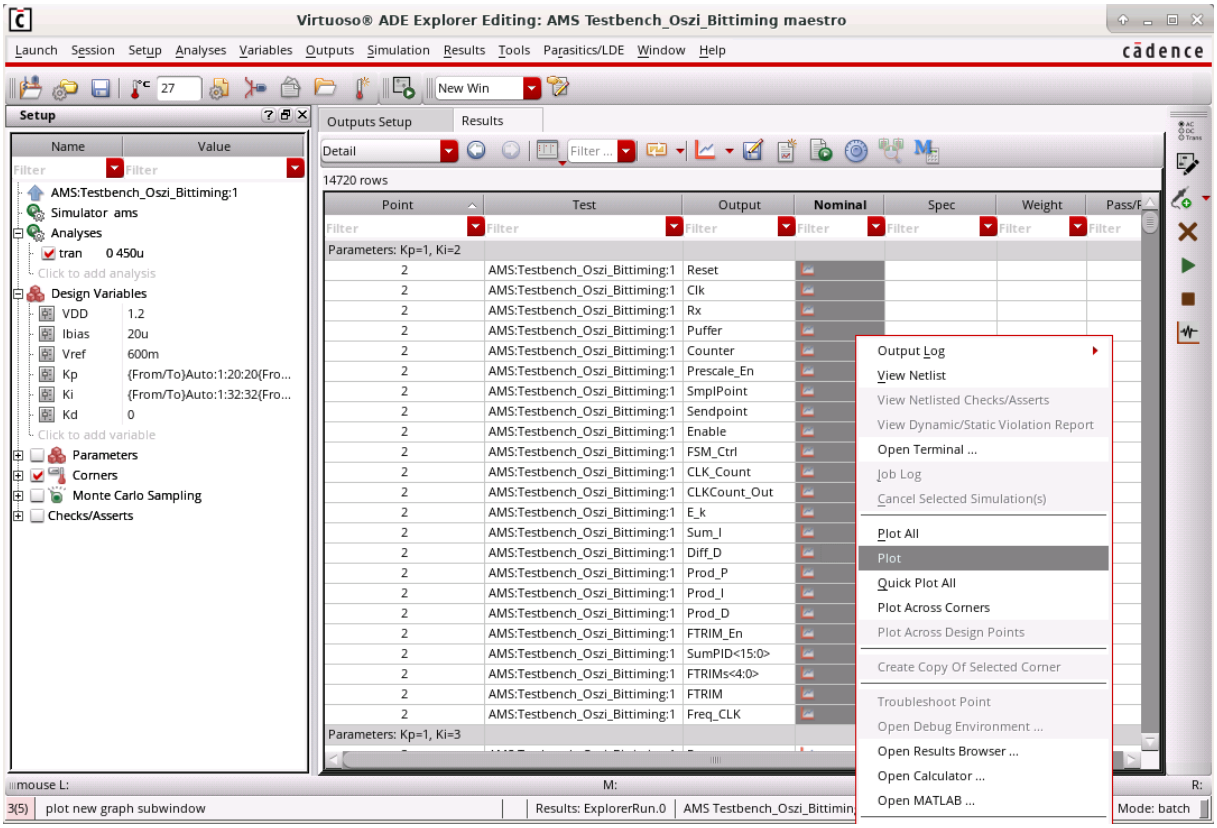

Abbildung 74: Das Results-Fenster

Die oben genannten Ausgänge lassen sich für einen Sweep-Punkt grafisch darstellen, indem im Results-Fenster die simulierten Outputs selektiert, mit der rechten Maustaste darauf geklickt und der Punkt Plot ausgewählt wird.

Als nächstes wird die Funktionalität des Regelalgorithmus bei dem Sweep-Punkt 2, welcher dem Regler-Parameter-Wertepaar $K_p = (0000\ 0001)_2 = (0.0625)_{10}$ und $K_i = (0000\ 0010)_2 = (0.125)_{10}$ entspricht, anhand der Simulationsergebnisse überprüft.

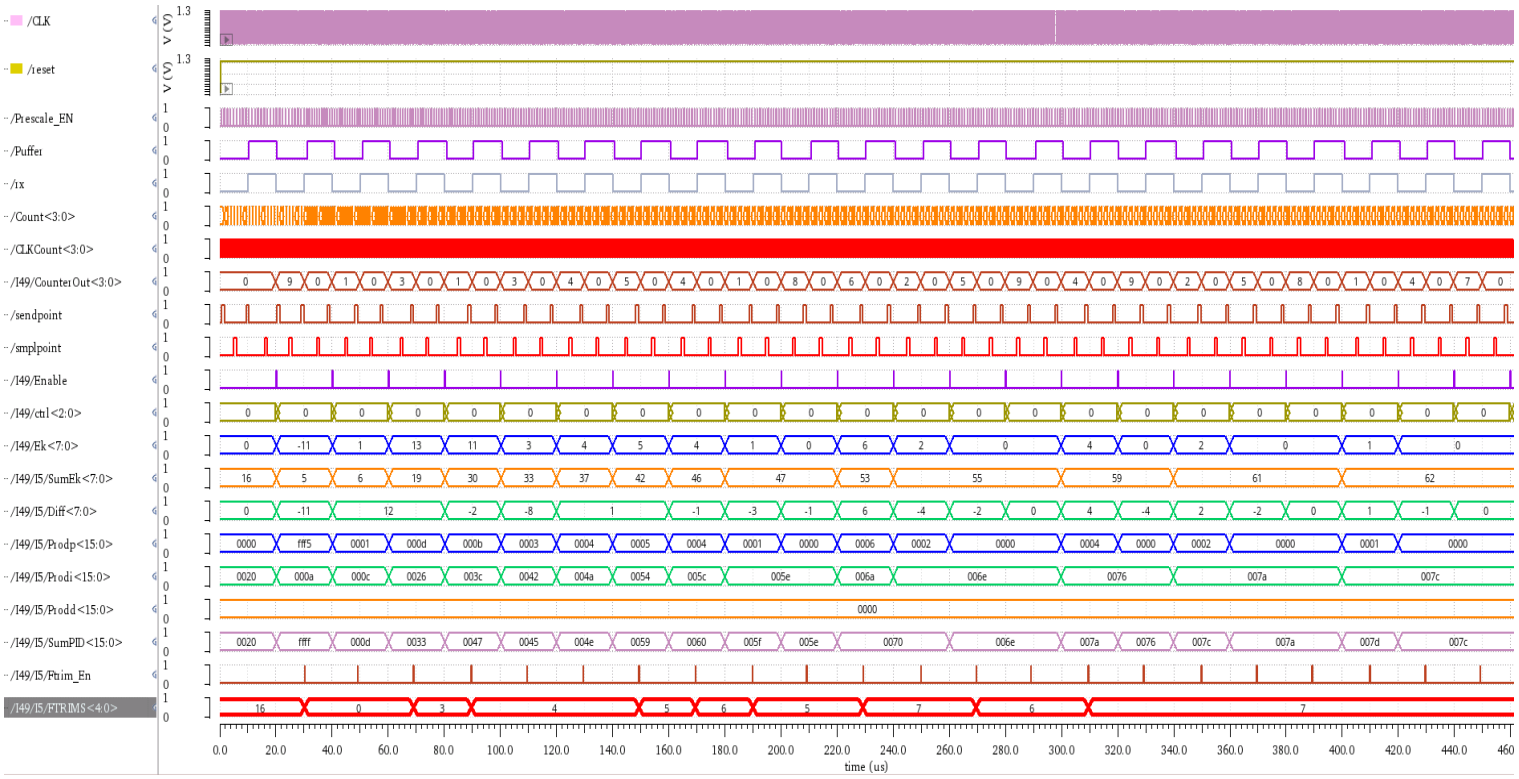

Abbildung 75: Signalverläufe der Ausgänge

Wie man in der Abbildung 76 sieht, handelt es sich um eine verfrühte fallende Flanke, deshalb geht die Control-FSM in den Zustand **NegPhasFehler** (ctrl<2:0>=3'd2) über und setzt das Enable-Signal (Enable=1'b1).

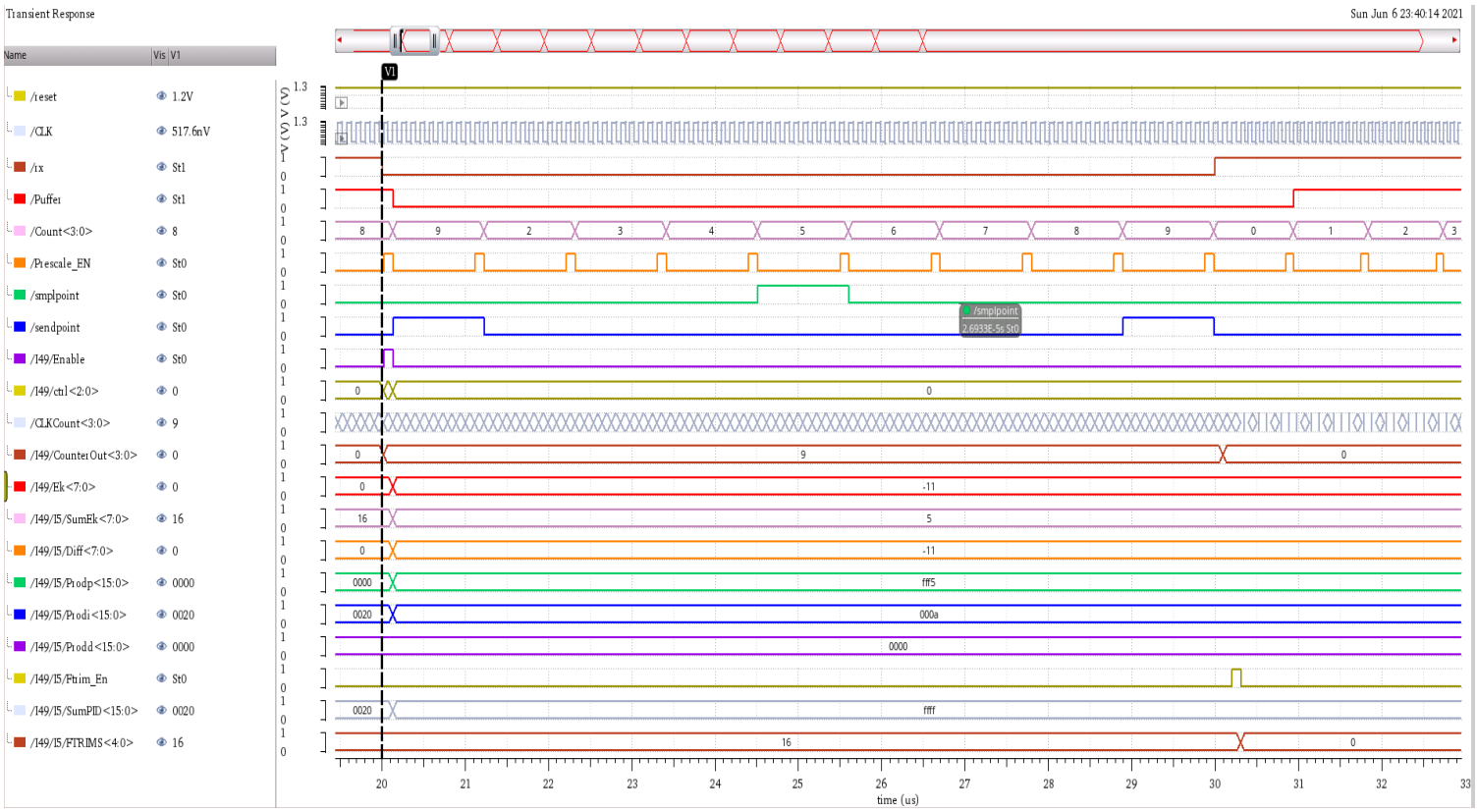

Abbildung 76: Die erste fallende Flanke

Die fallende Flanke ist bei Bittiming-Zählerstand acht und bei Clock-Zählerstand neun aufgetreten. Diese Werte werden über die Ausgänge Count<3:0> und CLKCounterOut<3:0> an das PhasenfehlerReg-Modul weitergeleitet. Daraus wird die Regelabweichung E(k) mit Hilfe von Gleichung 3.1.30 wie folgt berechnet:

$$\begin{cases} \boldsymbol{E(0) = 0} \\ \boldsymbol{E(1) = \big(10 * (E1 - 9)\big) + (E2 - 10) = (10 * (8 - 9)) + (9 - 10) = -11 = Ek < 7:0 >} \end{cases}$$

Der P- und I-Anteil werden mit Hilfe von Gleichung 3.1.45 berechnet und mit den Simulationsergebnissen verglichen.

$$\begin{cases} \boldsymbol{U_p(1) = K_p * E(1) = (0.0625)_{10} * (-11)_{10} = (-0.6875)_{10}} \\ \boldsymbol{U_i(1) = K_i * (I(0) + E(1)) = (0,125)_{10} * ((16)_{10} + (-11)_{10}) = (0.625)_{10}} \end{cases}$$

Daraus ergibt sich:

$$\begin{cases} \boldsymbol{Prodp < 15:0 >= Kp < 7:0 > * Ek < 7:0 >= (fff5)_{16} = (-0.6875)_{10} = U_p(1)} \\ \boldsymbol{Prodi < 15:0 >= Ki < 7:0 > * SumEk < 7:0 >= (000a)_{16} = (0.625)_{10} = U_i(1)} \end{cases}$$

Als nächstes werden Der P- und I-Anteil zusammenaddiert und bilden somit die Stellgröße $U(1)$.

$$\boldsymbol{U(1) = U_p(1) + U_i(1) = (-0.0625)_{10} = (ffff)_{16} = SumPID < 15:0 >}$$

Der Wert null wird dem FTRIM<4:0> Control-Bits Register zugewiesen, da die Stellgröße $U(1)$ im negativen Bereich liegt.

Erst ein Bit später wird das FTRIM_En-Signal gesetzt, was zur Aktualisierung des FTRIM<4:0> Steuersignals führt (Abbildung 76).

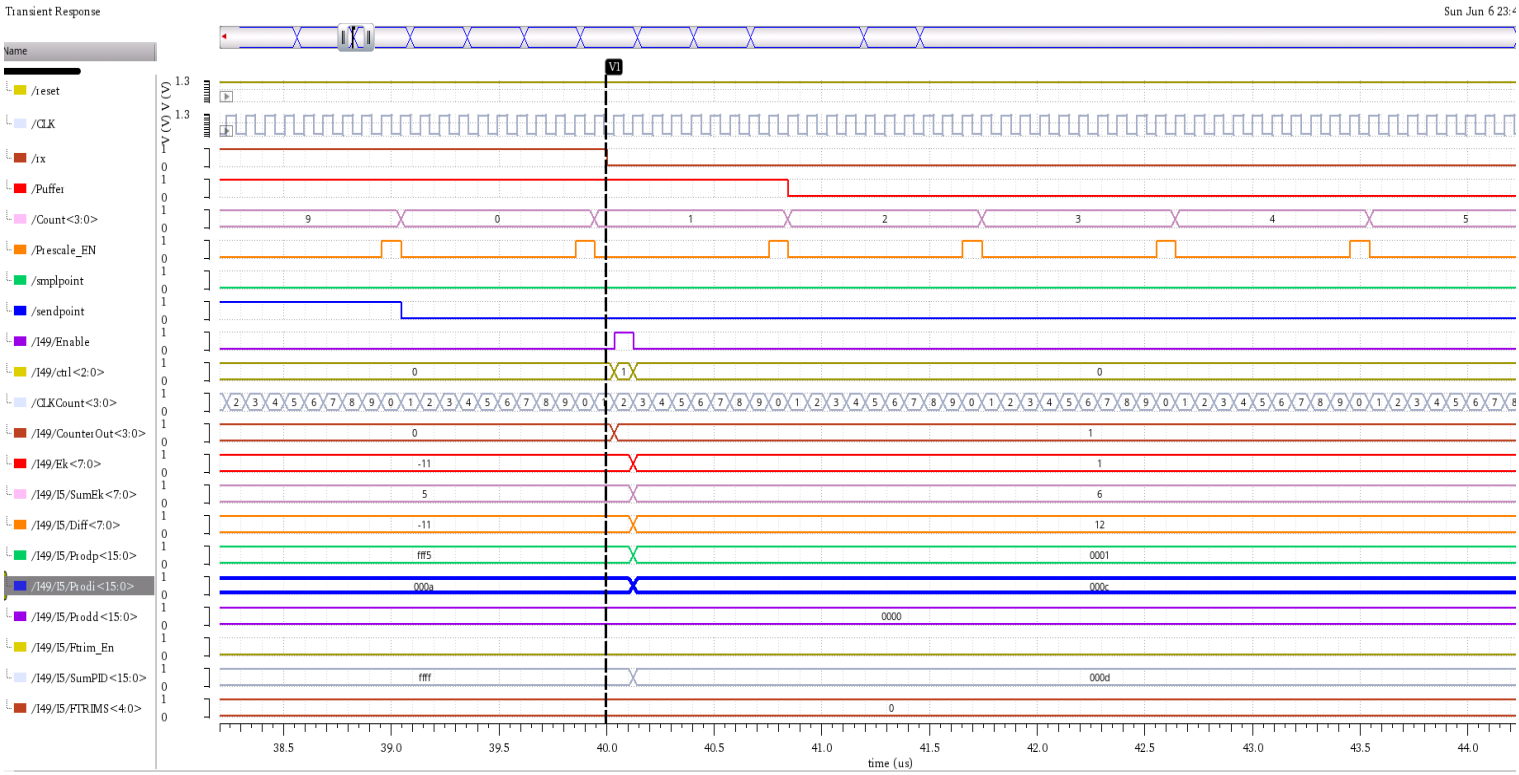

Abbildung 77: Die zweite fallende Flanke

Beim zweiten Flankenwechsel handelt es sich um einen positiven Phasenfehler. Daher springt die Control-FSM in den Zustand **PosPhasFehler**. In diesem Zustand wird der Wert eins dem ctrl-Register zugeordnet und das Enable-Signal wird gesetzt (ctrl<2:0>=3'd1 und Enable=1'b1).

Anschließend werden die Bittiming- und Clock-Counter-Zählerstand erfasst und an das PhasenfehlerReg-Modul übermittelt:

$$\boldsymbol{E1 = 1 = Count < 3:0 > \ und \ E2 = 1 = CLKCounterOut < 3:0 >}$$

Daraus lässt sich die Regelabweichung mit Hilfe von Gleichung 3.1.30 wie folgt bestimmen:

$$\boldsymbol{E(2) = \big(10 * (E1 - 1)\big) + E2 = 1 = Ek < 7:0 >}$$

Der P- und I-Anteil werden mit Hilfe von Gleichung 3.1.45 berechnet und mit den Simulationsergebnissen verglichen.

$$\begin{cases} \boldsymbol{U_p(2) = K_p * E(2) = (0.0625)_{10} * (1)_{10} = (0.0625)_{10}} \\ \boldsymbol{U_i(2) = K_i * (I(1) + E(2)) = (0,125)_{10} * ((5)_{10} + (1)_{10}) = (0.75)_{10}} \end{cases}$$

Daraus ergibt sich:

$$\begin{cases} \boldsymbol{Prodp < 15:0 >= Kp < 7:0 > * Ek < 7:0 >= (0001)_{16} = (0.0625)_{10} = U_p(2)} \\ \boldsymbol{Prodi < 15:0 >= Ki < 7:0 > * SumEk < 7:0 >= (000c)_{16} = (0.75)_{10} = U_i(2)} \end{cases}$$

Anschließend werden Der P- und I-Anteil zusammenaddiert und somit wird aus dem Additionsergebnis die Stellgröße $U(2)$ gebildet.

$$\boldsymbol{U(2) = U_p(2) + U_i(2) = (0.8125)_{10} = (000d)_{16} = SumPID < 15:0 >}$$

Von der insgesamt fünfzehn Bit breiten Stellgröße $SumPID < 15:0 >$ werden die ersten 5 Vorkomma-Bits durch den Ausgang $FTRIM < 4:0 >< = SumPID < 8:4 >$ mit dem Wert null an den analogen Oszillator übertragen.

Die oben dargestellten Simulationsergebnisse des geschlossenen Regelkreises stimmen mit den berechneten Werten überein und dienen als Nachweis für die Funktionalität des gesamten Regelsystems.

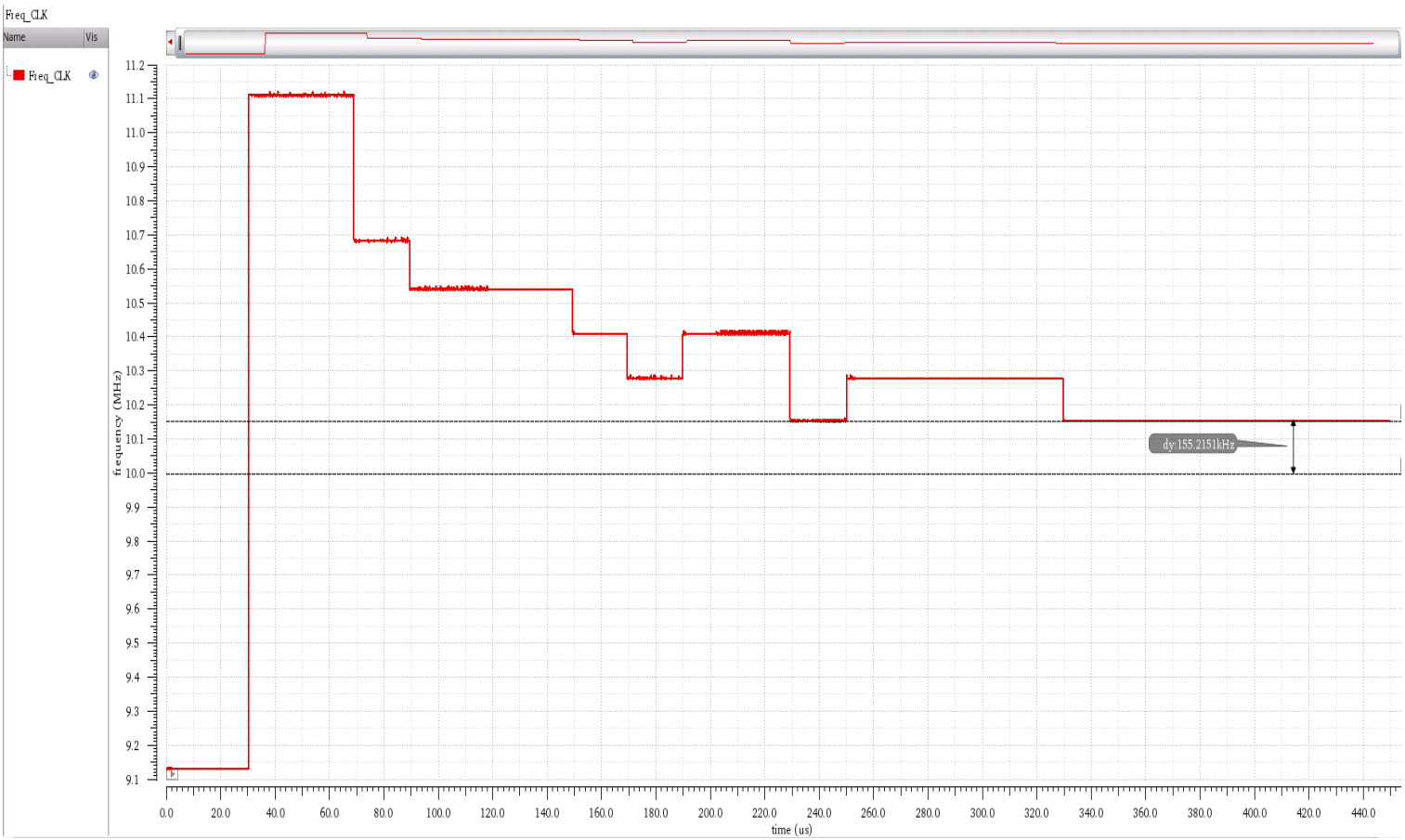

Abbildung 78: Regelgröße-Verlauf bei Kp=0.0625 und Ki=0.125

Die Abbildung 78 veranschaulicht den Verlauf der Regelgröße, konkret die Taktfrequenz des analogen Oszillators. Beim ersten Abtastzeitpunkt wird die Stellgröße neu berechnet, was zu einer drastischen Erhöhung der Regelgröße führt. Danach nimmt die Regelabweichung durch den Regelalgorithmus stetig bis zum Erreichen des stationären Zustands ab. Der Signalverlauf der Regelabweichung in Abbildung 75 zeigt, dass diese vollständig kompensiert wurde, weil $Ek < 7:0 >$ den Wert null im stationären Zustand beinhaltet. Konkret bedeutet dies, dass die detektierten fallenden Flanken ab dieser Abtastzeit innerhalb des Synchronisationssegments aufgetreten sind.

Die Taktfrequenz hat jedoch eine Abweichung von $0{,}155\ MHz$ (siehe Abbildung 78). Dies hat zur Folge, dass der Taktfrequenz-Sollwert mit den Parametern K_p und K_i nicht erreicht werden kann. Dennoch ist die in Abschnitt 3.1.29 erwähnte Oszillatortoleranz-Bedingung erfüllt, weil

$$\boldsymbol{df_{ist} = 0,155\ MHz < 0,3\ MHz = 3\% * f_{soll}}$$

Als nächstes werden weitere Simulationsergebnisse dargestellt und nach passenden Regler-Parametern K_p und K_i gesucht, mit denen die Abweichung vom Sollwert minimiert, im Idealfall vollständig entfernt und ebenfalls schnell wie möglich erreicht werden kann.

Für $K_p = (0000\ 0001)_2 = (0{,}0625)_{10}$; $K_i = (0000\ 0111)_2 = (0{,}4375)_{10}$

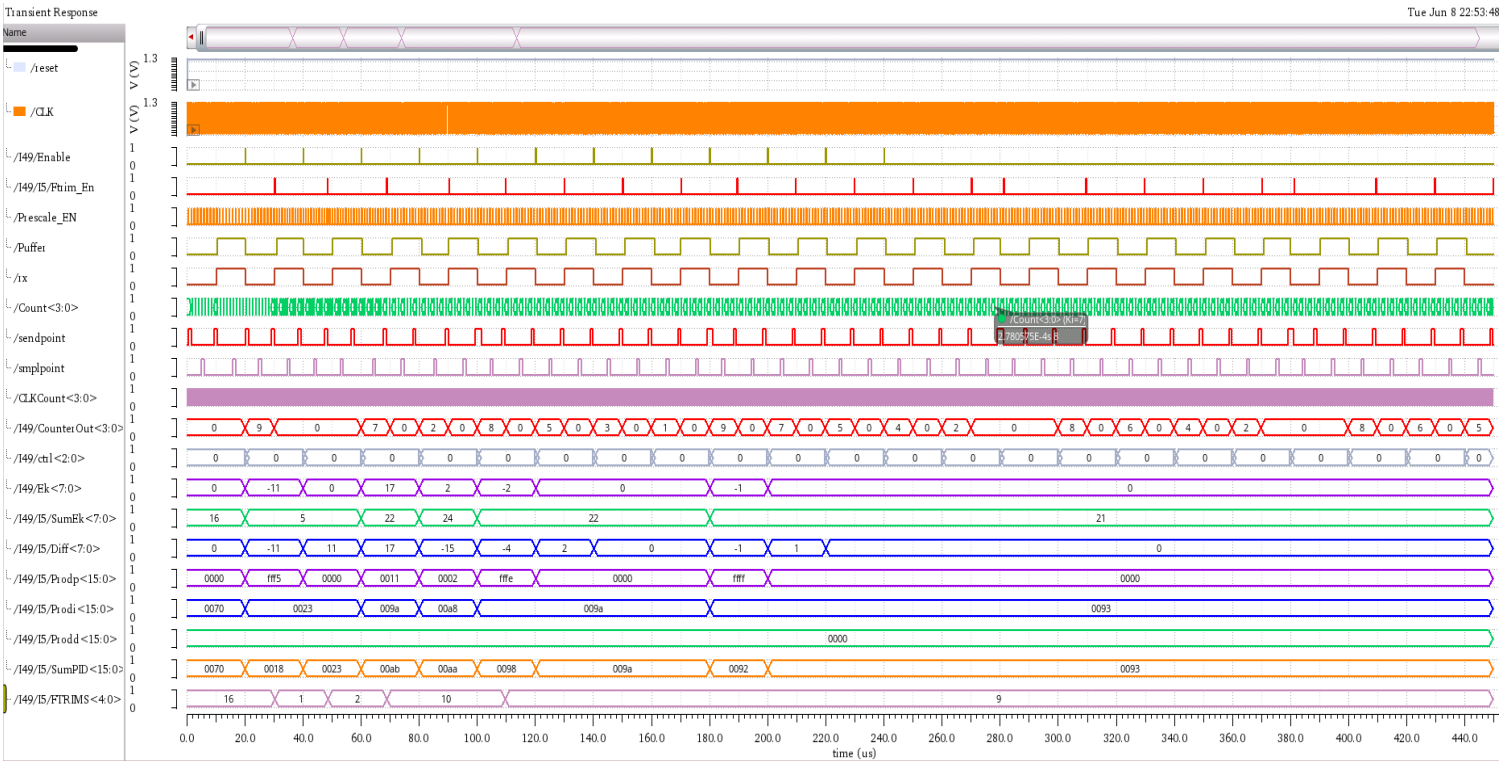

Abbildung 79: Signalverläufe der Testschaltung

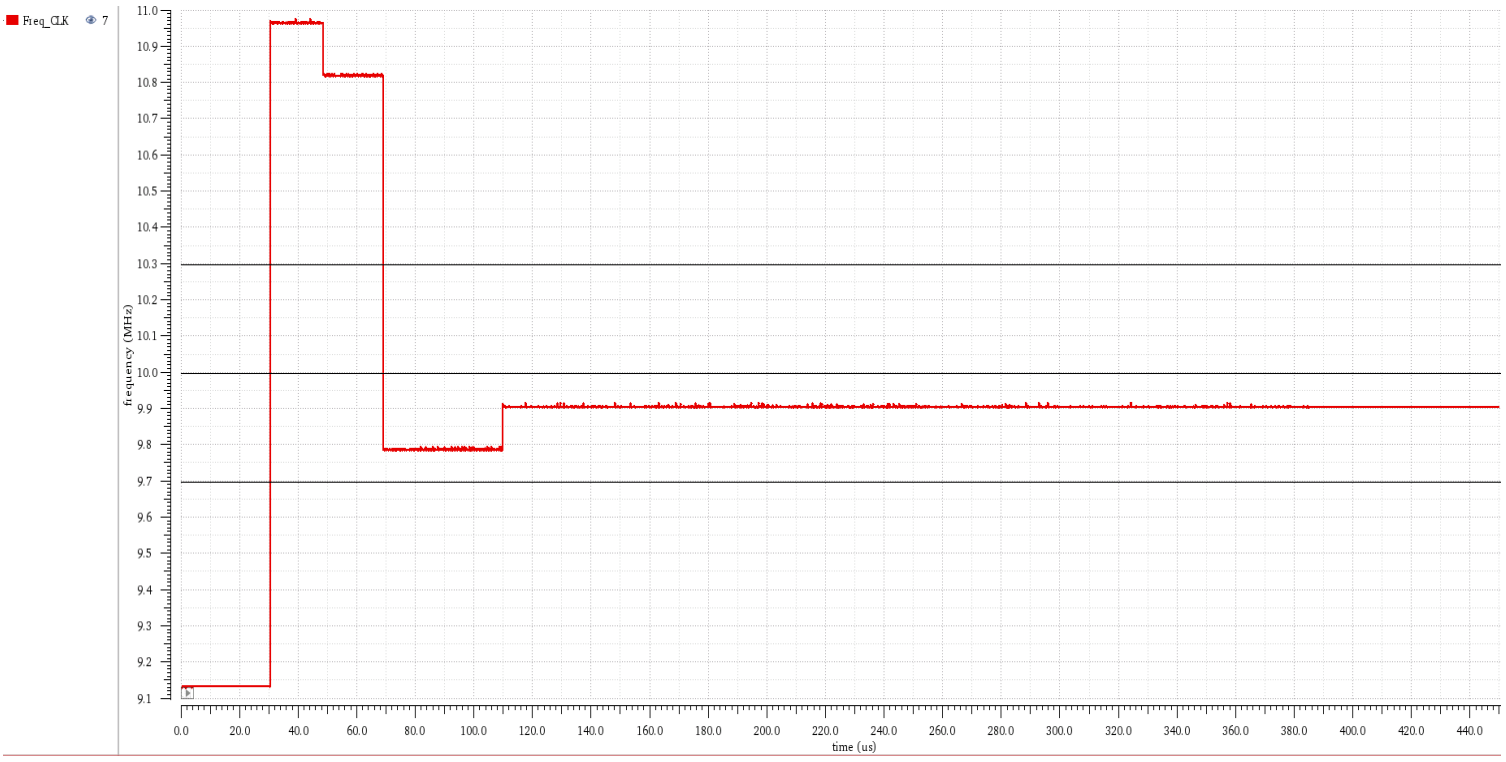

Abbildung 80: Signalverlauf der Regelgröße

Für $K_p = (0000\ 0001)_2 = (0{,}0625)_{10}$; $K_i = (0001\ 1011)_2 = (1{,}6875)_{10}$

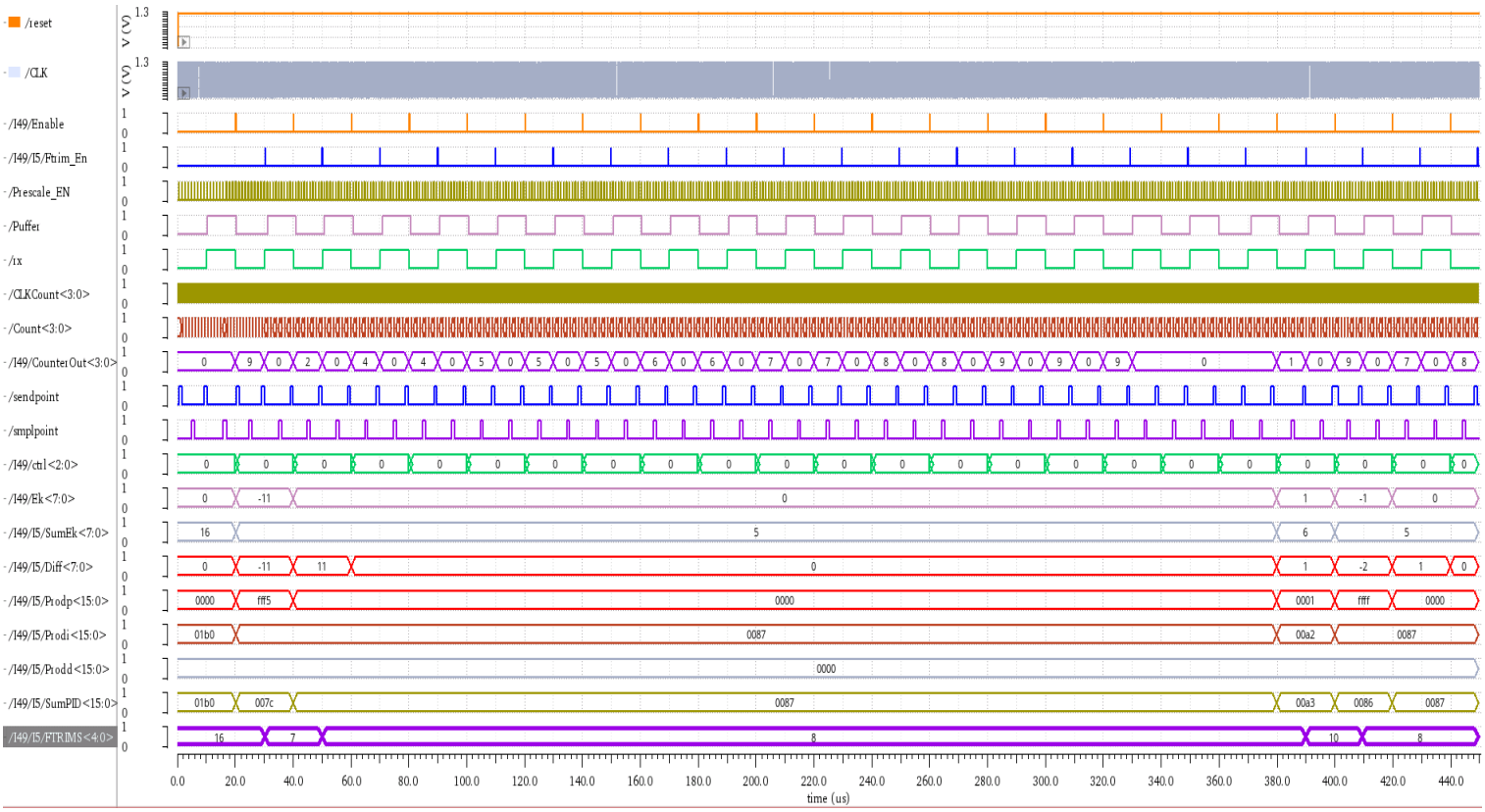

Abbildung 81: Signalverläufe der Testschaltung

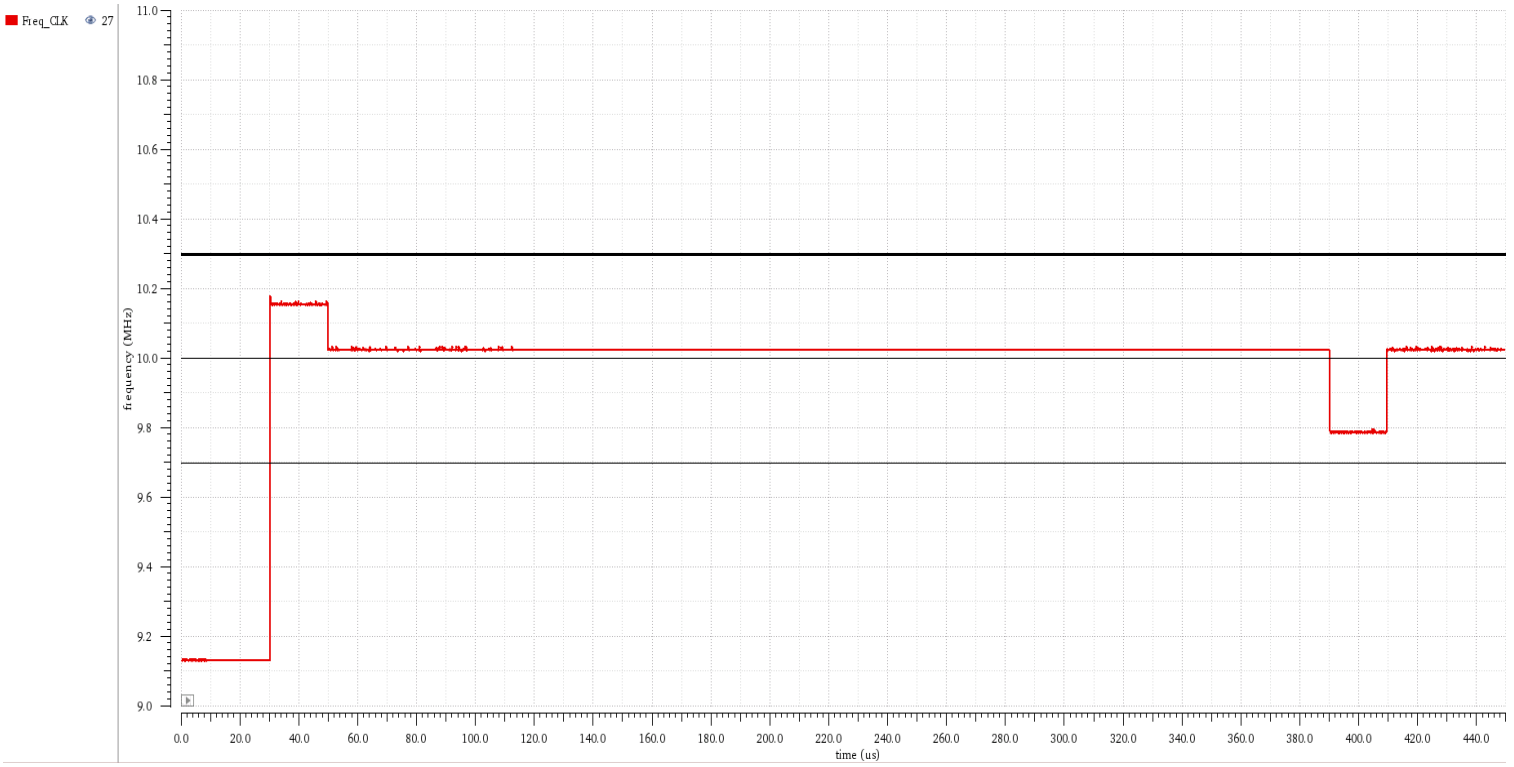

Abbildung 82: Signalverlauf der Regelgröße

Für $K_p = (0000\ 0001)_2 = (0{,}0625)_{10}$; $K_i = (0001\ 1100)_2 = (1{,}75)_{10}$

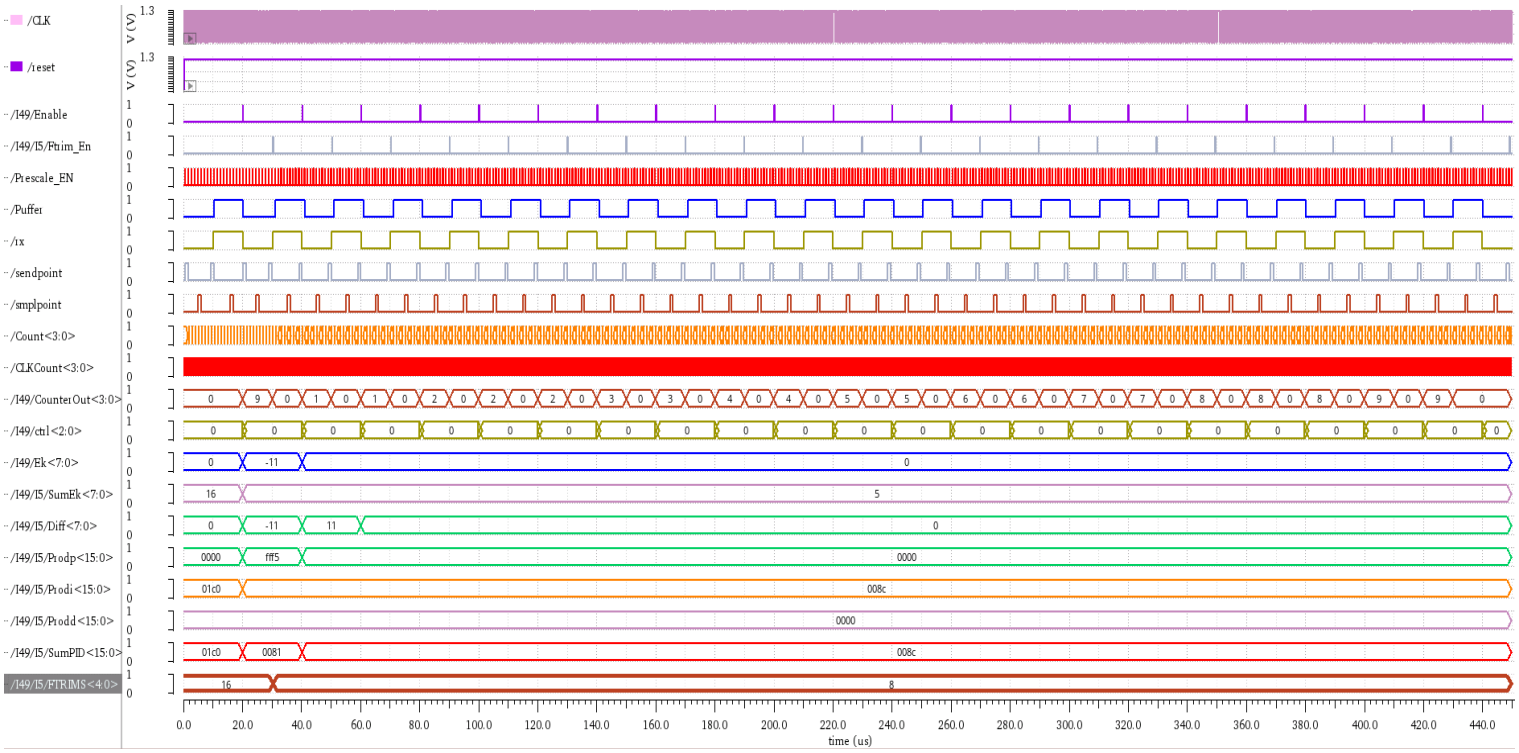

Abbildung 83: Signalverläufe der Testschaltung

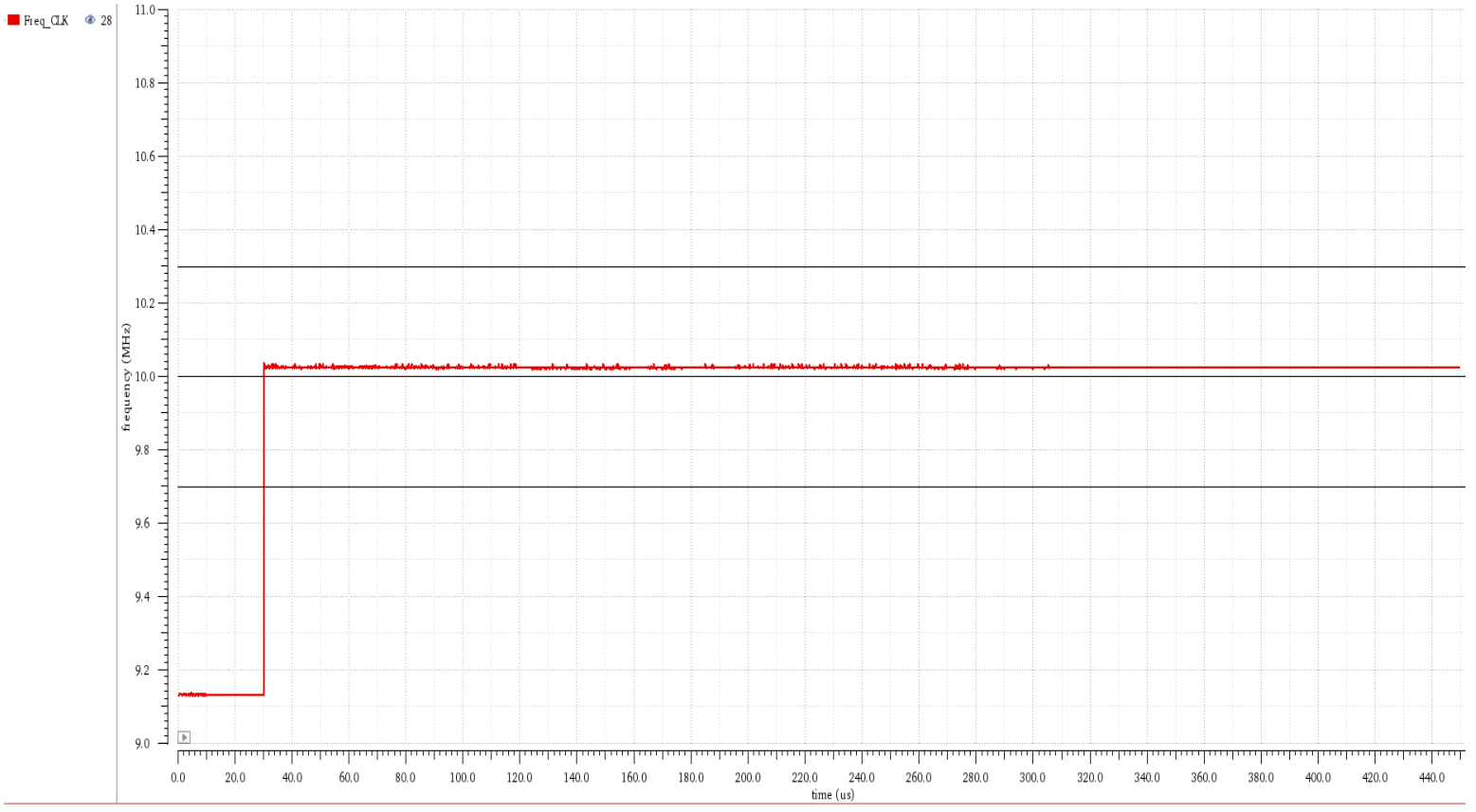

Abbildung 84: Signalverlauf der Regelgröße

Für $K_p = (0000\ 0001)_2 = (0{,}0625)_{10}$; $K_i = (0010\ 0000)_2 = (2)_{10}$

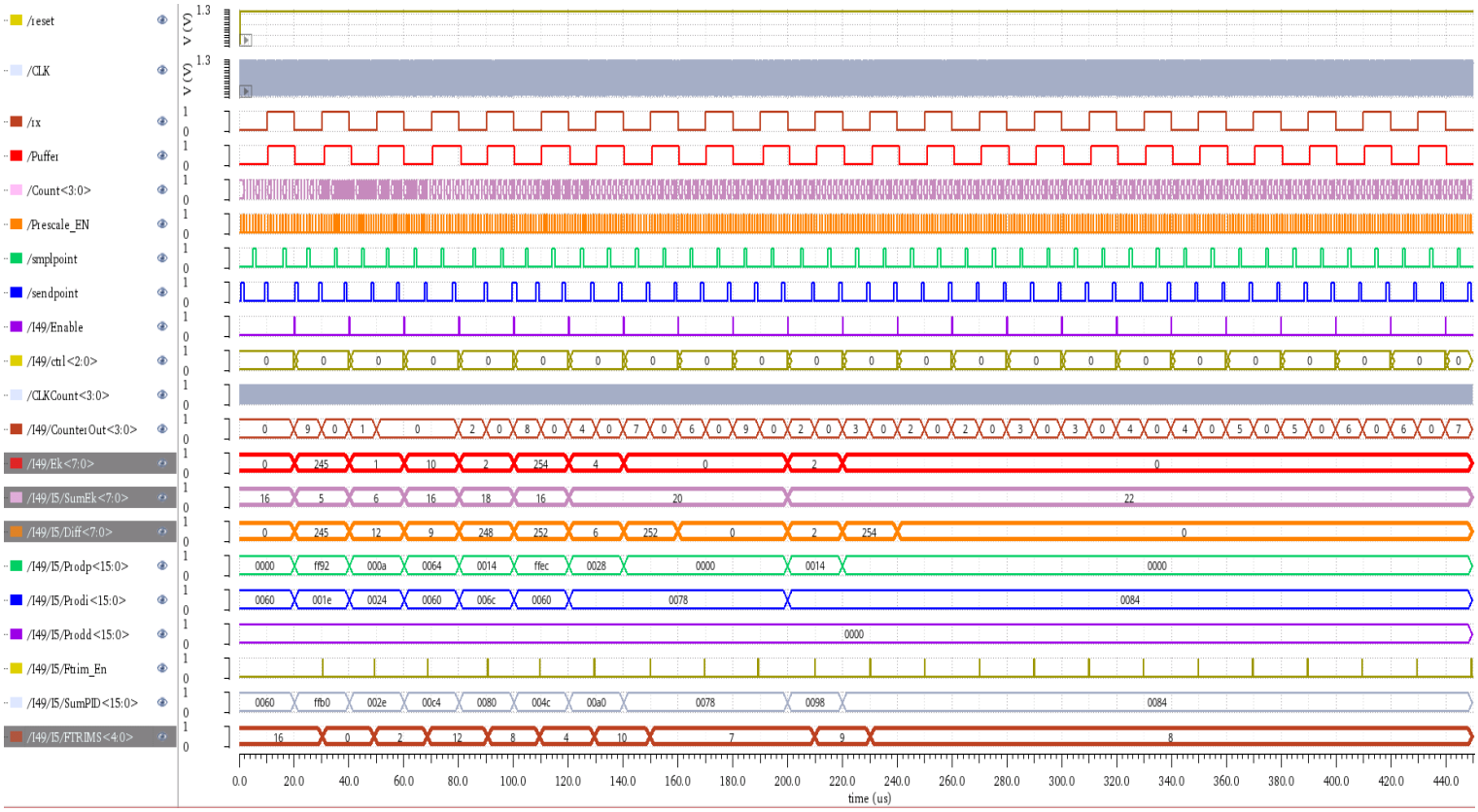

Abbildung 85:Signalverläufe der Testschaltung

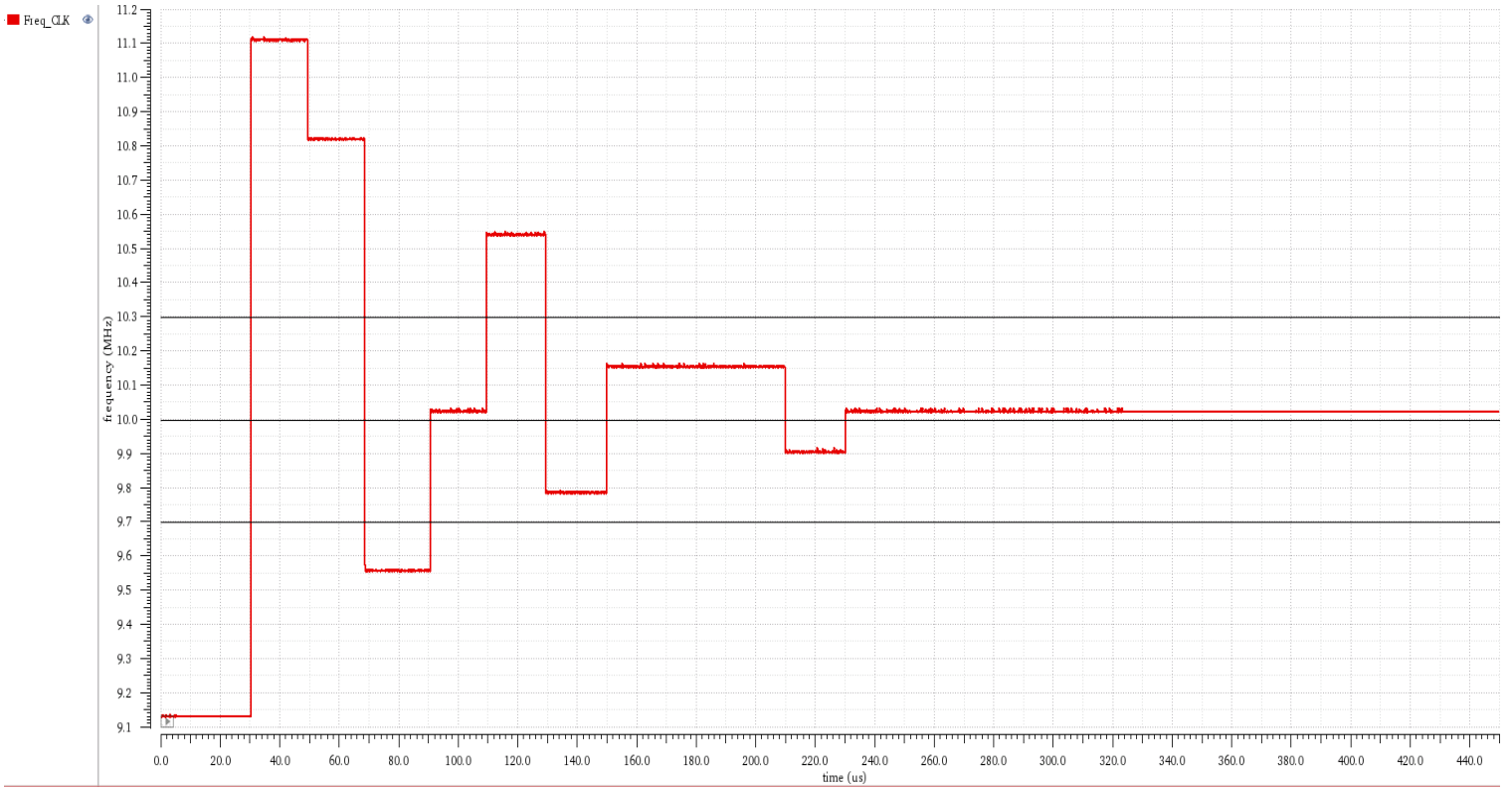

Abbildung 86: Signalverlauf der Regelgröße

Für $K_p = (0000\ 0010)_2 = (0{,}0625)_{10}$; $K_i = (0010\ 0000)_2 = (2)_{10}$

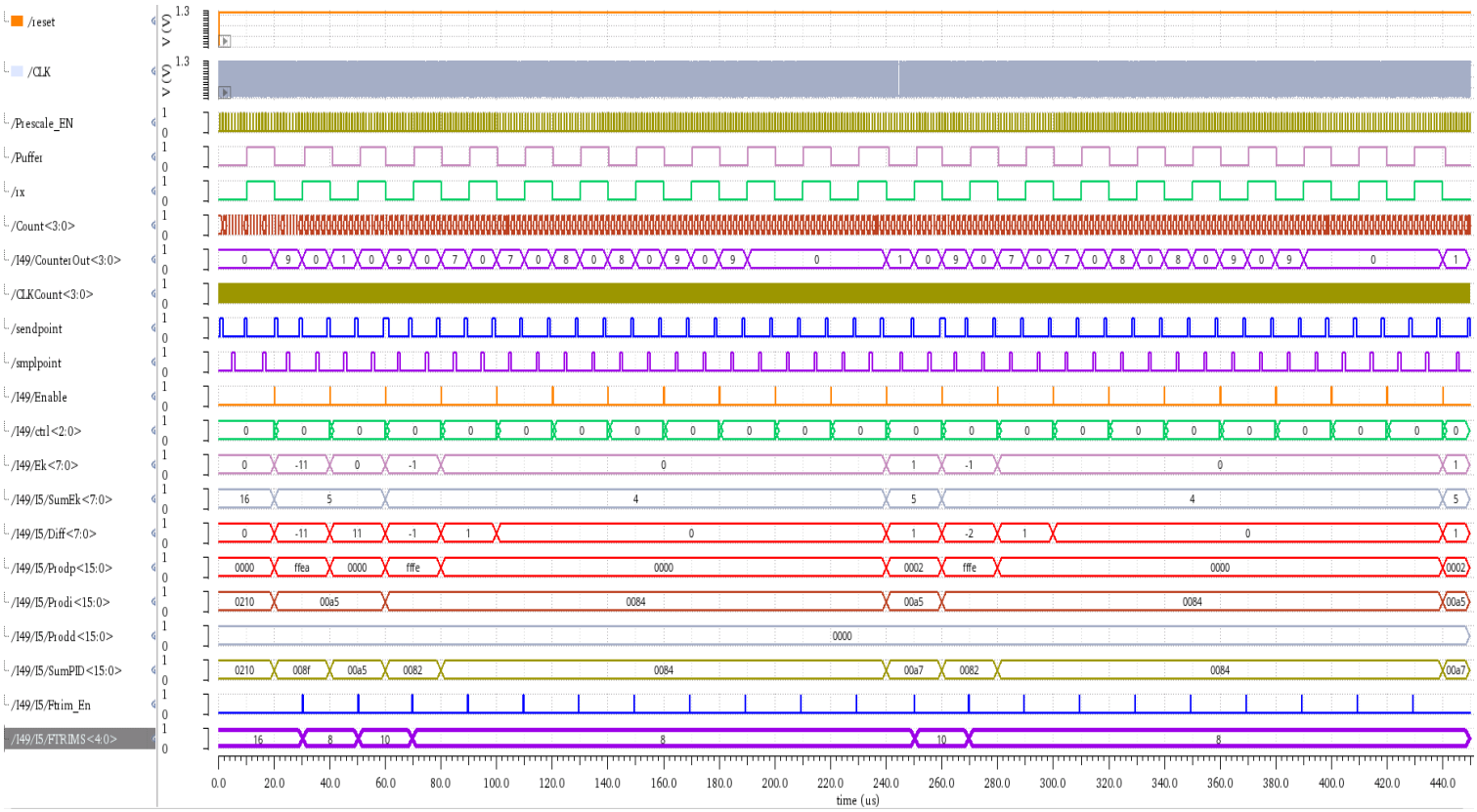

Abbildung 87: Signalverläufe der Testschaltung

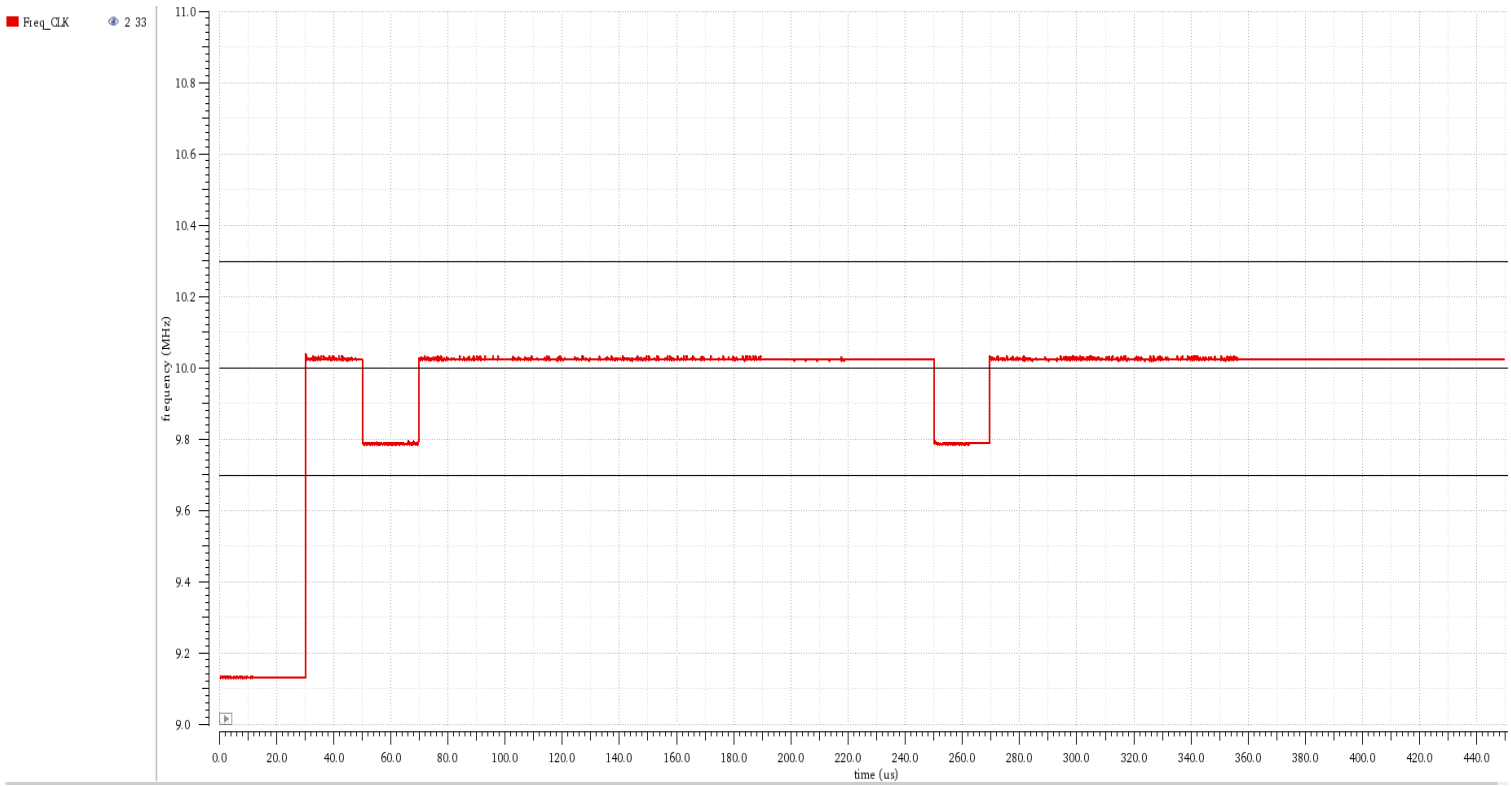

Abbildung 88: Signalverlauf der Regelgröße

Für $K_p = (0000\ 0011)_2 = (0{,}1875)_{10}$; $K_i = (0001\ 1011)_2 = (1{,}6875)_{10}$

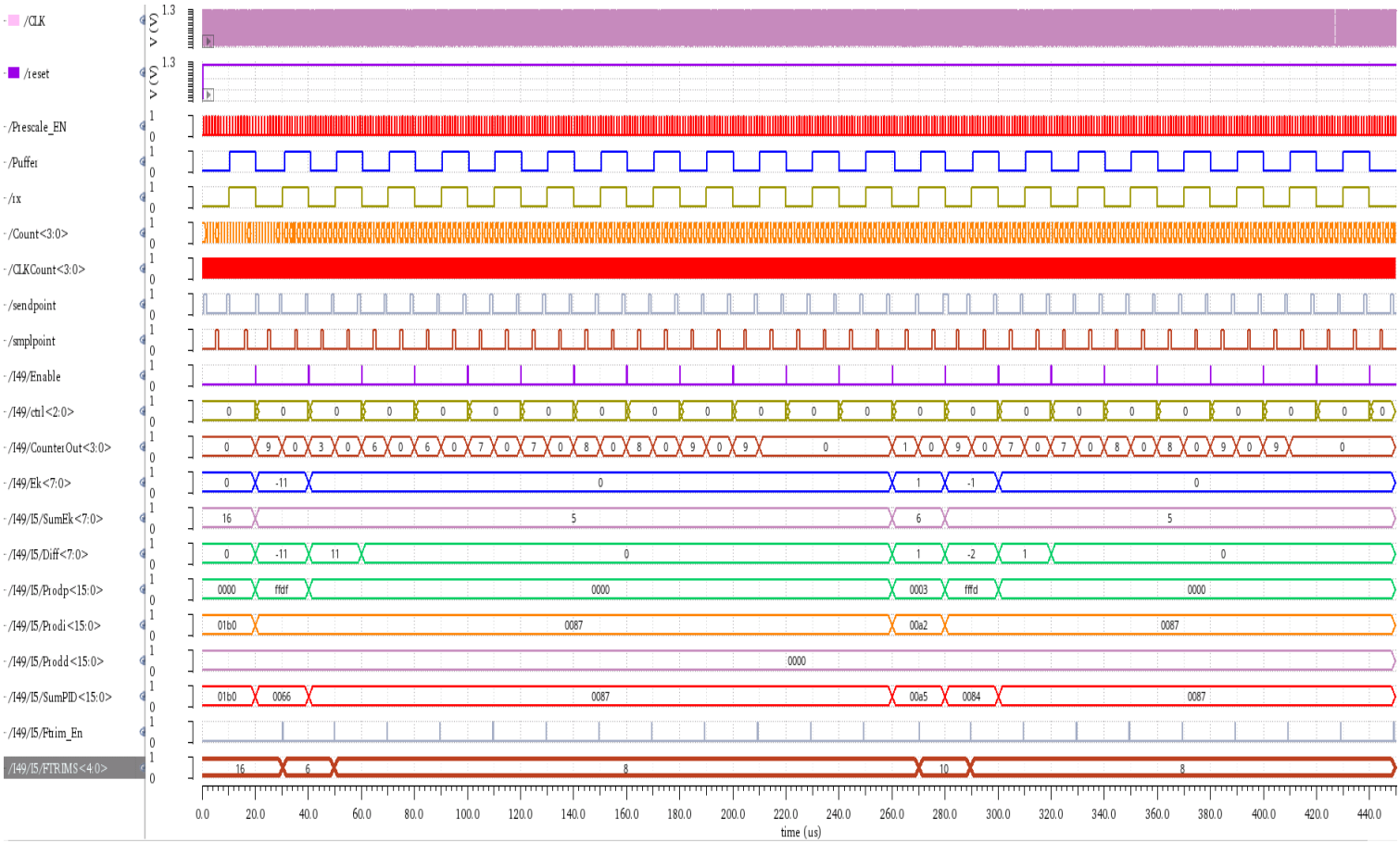

Abbildung 89: Signalverläufe der Testschaltung

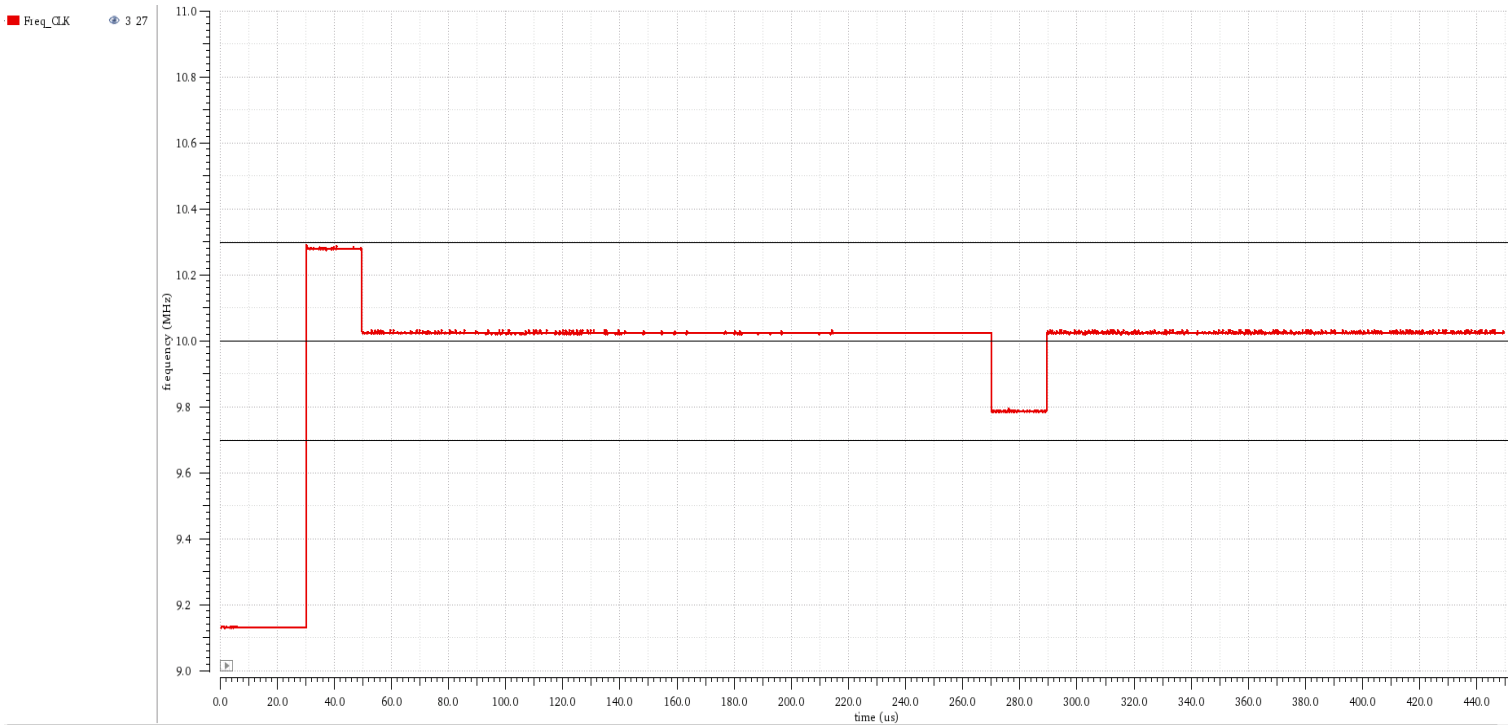

Abbildung 90: Signalverlauf der Regelgröße

Für $K_p = (0000\ 0111)_2 = (0{,}4375)_{10}$; $K_i = (0000\ 1000)_2 = (0{,}5)_{10}$

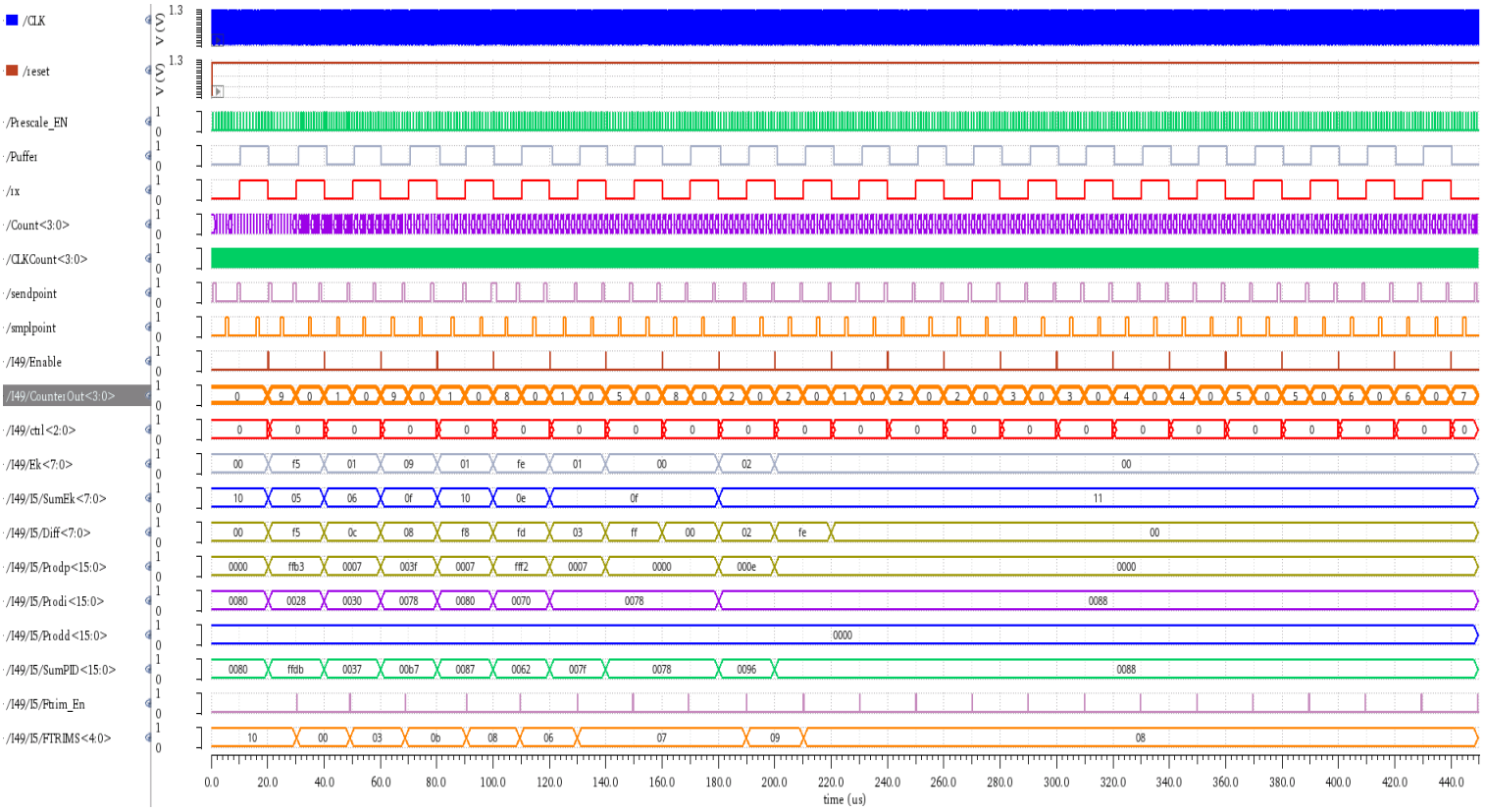

Abbildung 91: Signalverläufe der Testschaltung

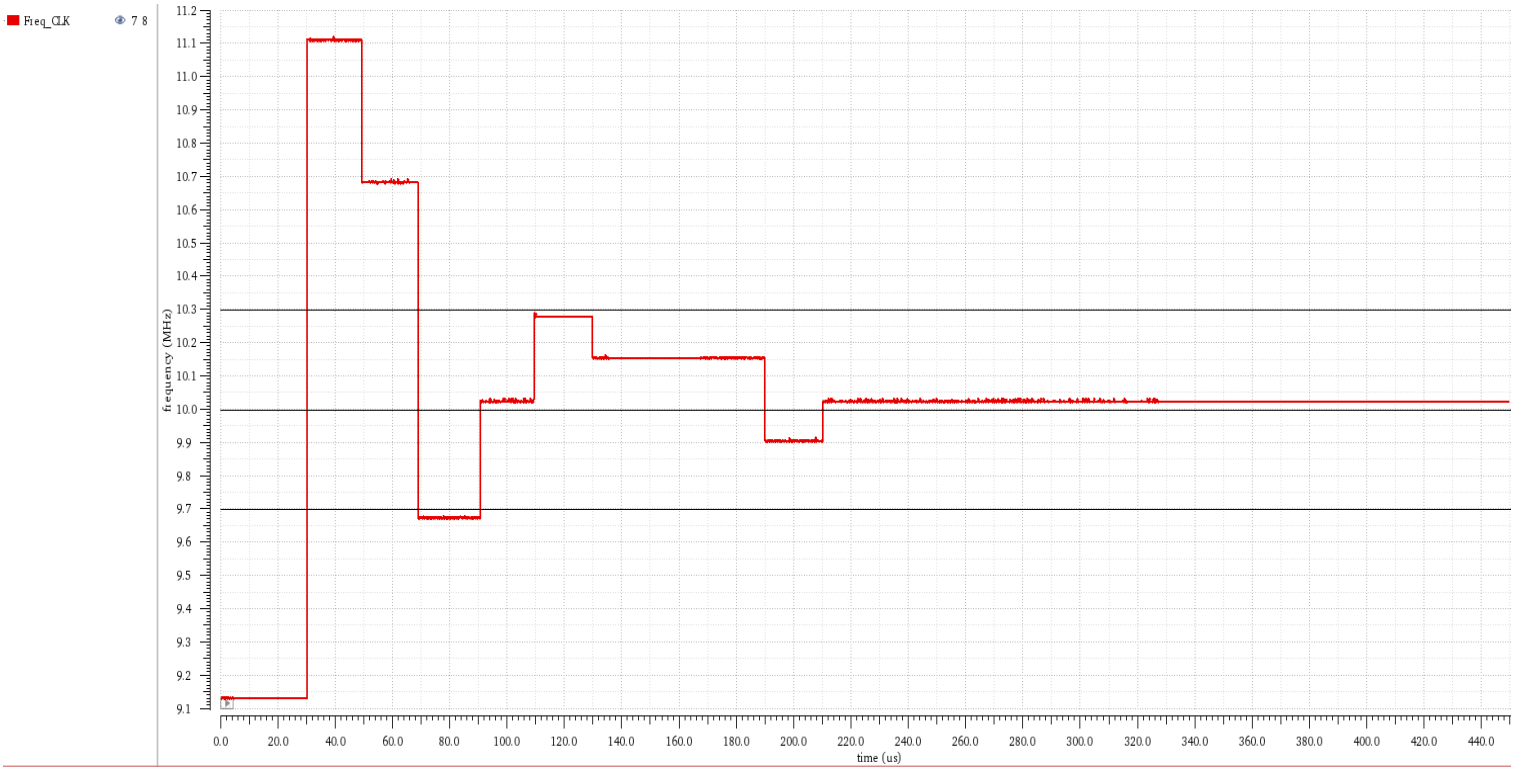

Abbildung 92: Signalverlauf der Regelgröße

Für $K_p = (0000\ 1010)_2 = (0{,}625)_{10}$; $K_i = (0000\ 1001)_2 = (0{,}5625)_{10}$

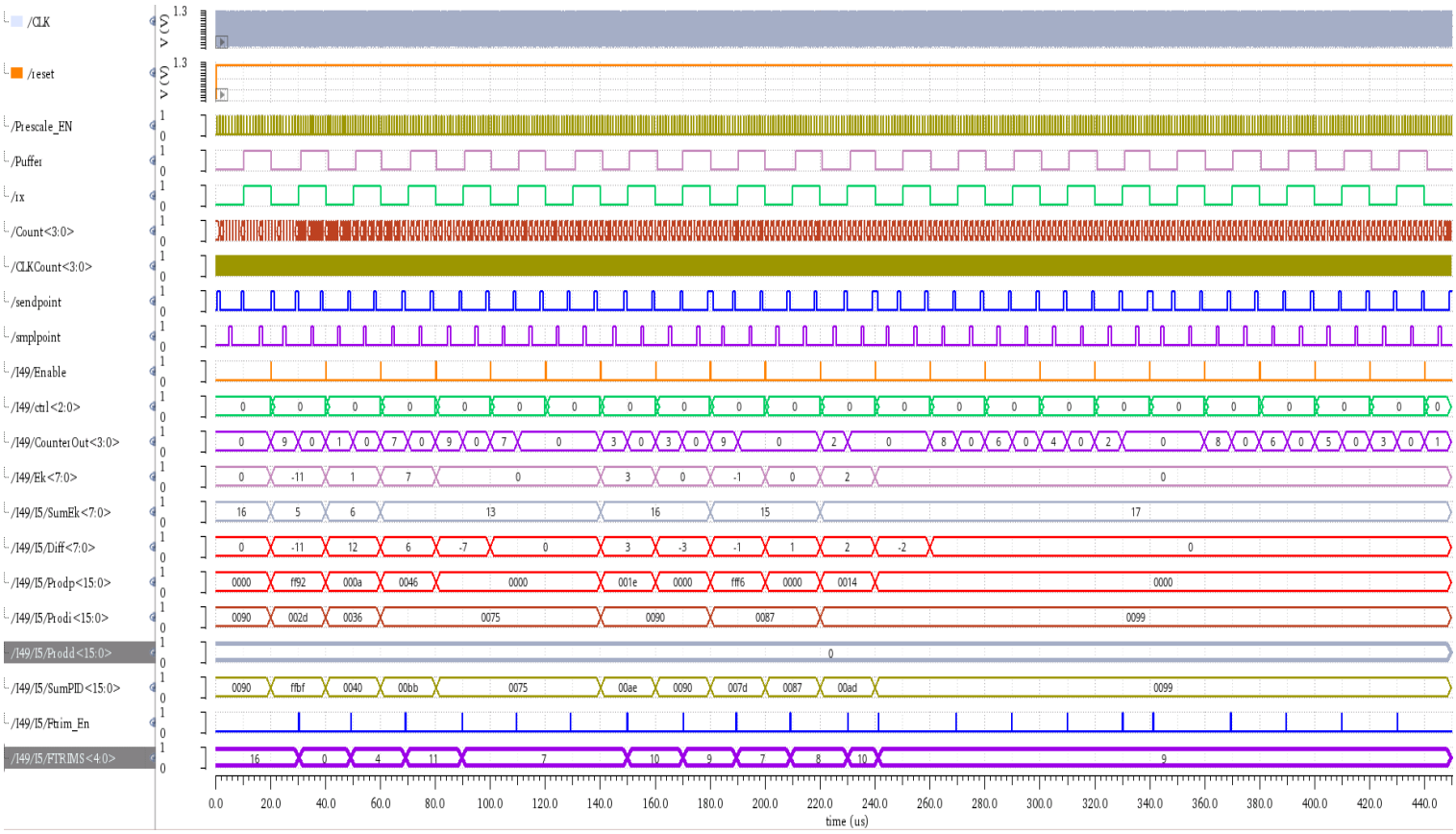

Abbildung 93: Signalverläufe der Testschaltung

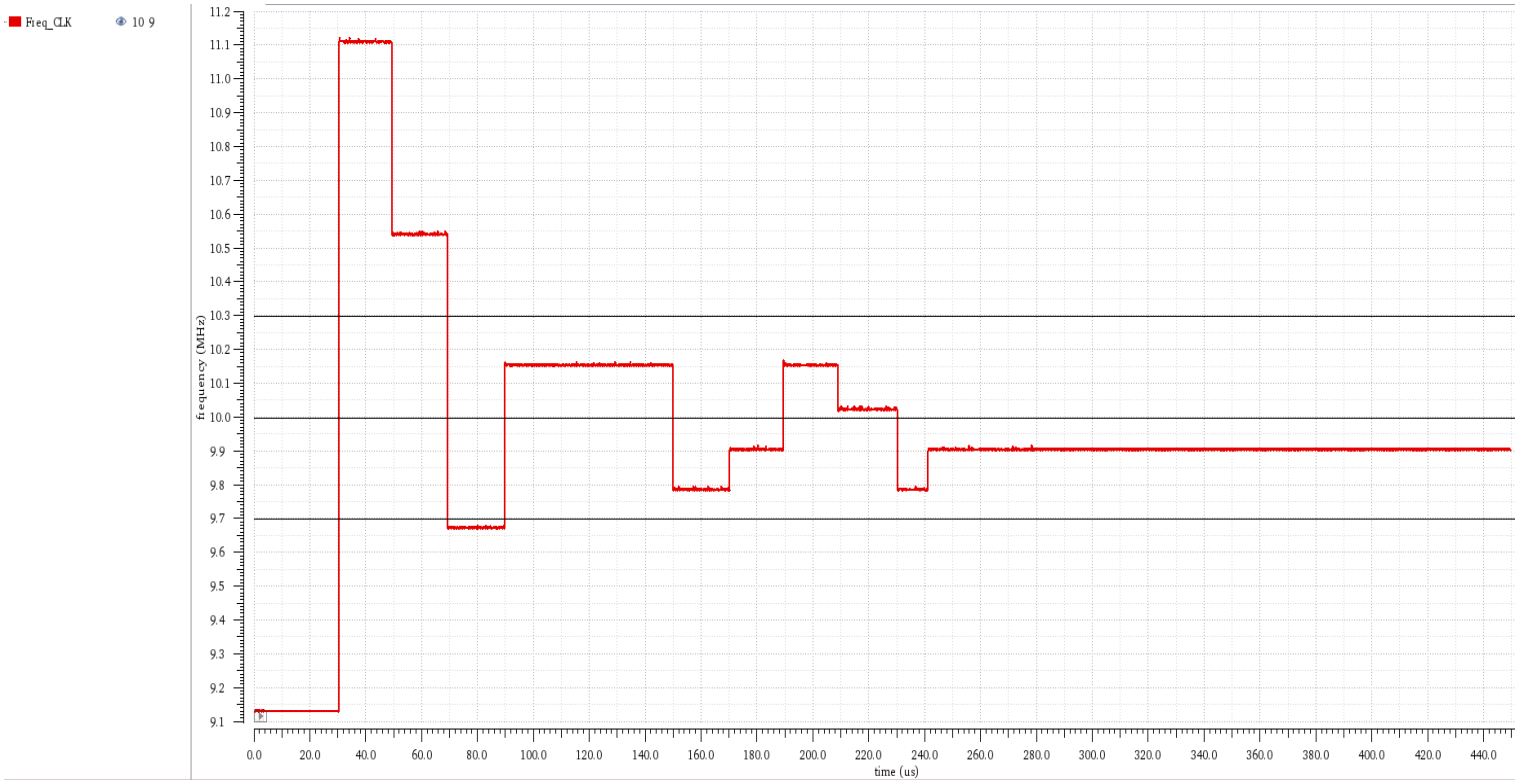

Abbildung 94: Signalverlauf der Regelgröße

In den oben aufgeführten Simulationsergebnisse wurden drei waagerechte Linien bei den Werten $10{,}3MHz$, $10MHz$ und $9{,}7MHz$ platziert, welche den Oszillator-Toleranz-Grenzwerten und dem Taktfrequenz-Sollwert entsprechen.

Diese Simulationsergebnisse ermöglichen die Untersuchung des Regelkreisverhaltens und gewährleisten somit eine schnelle und unkomplizierte Festlegung der Regler-Parameter, mit denen die in Abschnitt 5.1 ursprünglich gegebenen Anforderungen an den geschlossenen Regelkreis erfüllt werden.

Durch Erhöhung des K_i-Parameters konnte das von P-Anteil verursachte Überschwingen ausgeregelt werden. Dies führte zur Verbesserung der Regler-Reaktion auf die Regelabweichung.

Die Simulationsverläufe der Regelgröße für unterschiedliche Regler-Parameter-Wertepaare liegen innerhalb der Oszillator-Toleranz-Spanne im stationären Zustand. Somit ist die Regelungsgenauigkeit als erforderliche Bedingung gegeben.

Mit dem Wertepaar $K_p = (0{,}0625)_{10}$ und $K_i = (1{,}75)_{10}$ konnte die beste Reaktion auf die Regelabweichung erzielt werden. Außerdem konnte die Sollwert-Taktfrequenz von $10\ MHz$ wesentlich schneller erreicht werden.

Im Folgenden wird die im Abschnitt 5.2.4 genannte These hinsichtlich FTRIM-Controlbits-Änderung durch entsprechende Simulationsergebnisse überprüft. Simuliert wird die Taktfrequenz des analogen Oszillators mit dem Wertepaar $K_p = (0{,}0625)_{10}$ und $K_i = (1{,}75)_{10}$.

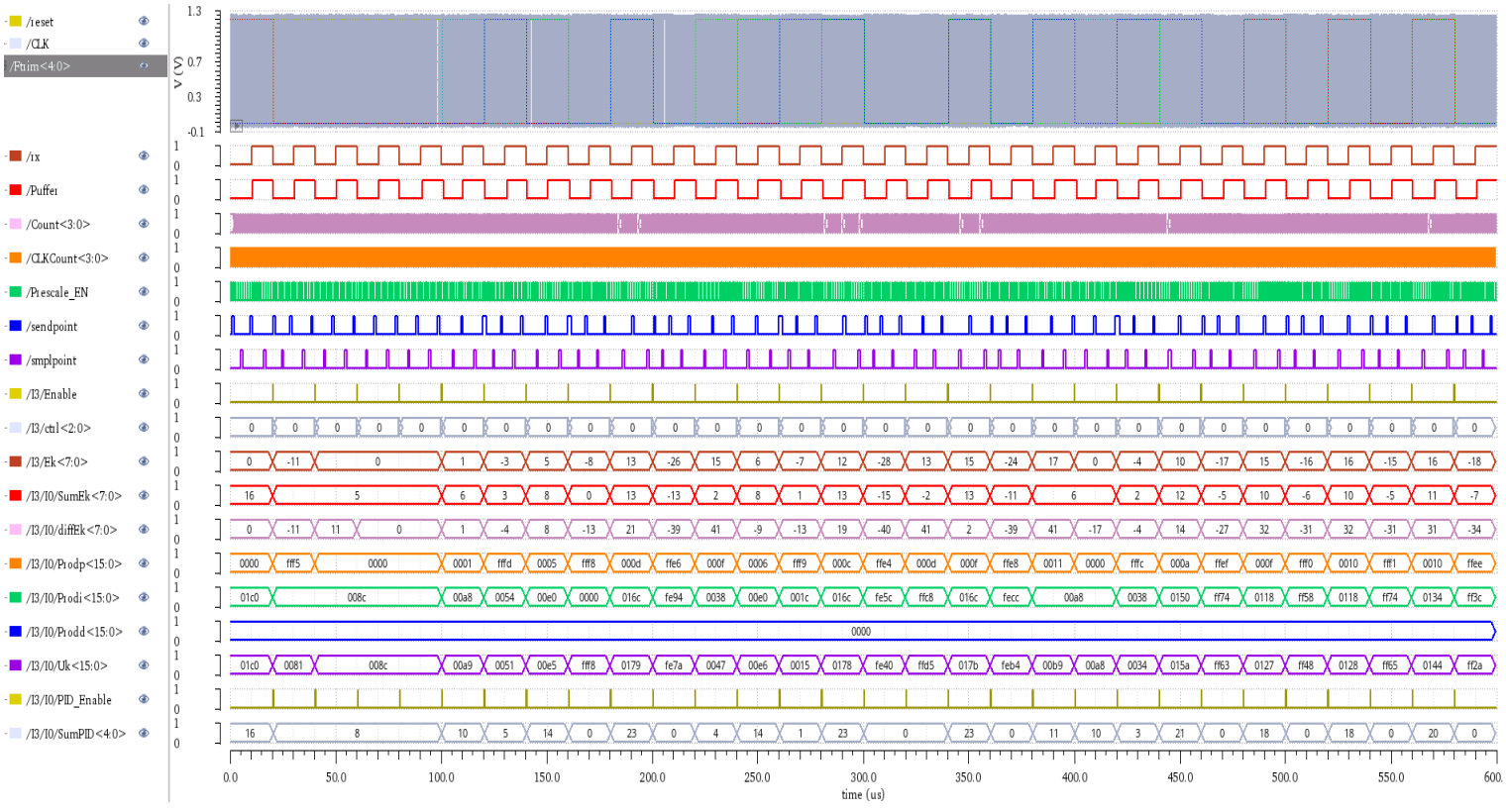

Abbildung 95: Signalverläufe für das Wertepaar $K_p = (0{,}0625)_{10}$ und $K_i = (1{,}75)_{10}$

Wie in Abbildung 95 veranschaulicht, wird der FTRIM-Code abweichend zu den oben aufgeführten Simulationsergebnissen, nicht erst mit Beginn der nächsten Bit geändert (PID_Enable<=1), sondern umgehend nachdem das Enable-Signal gesetzt ist.

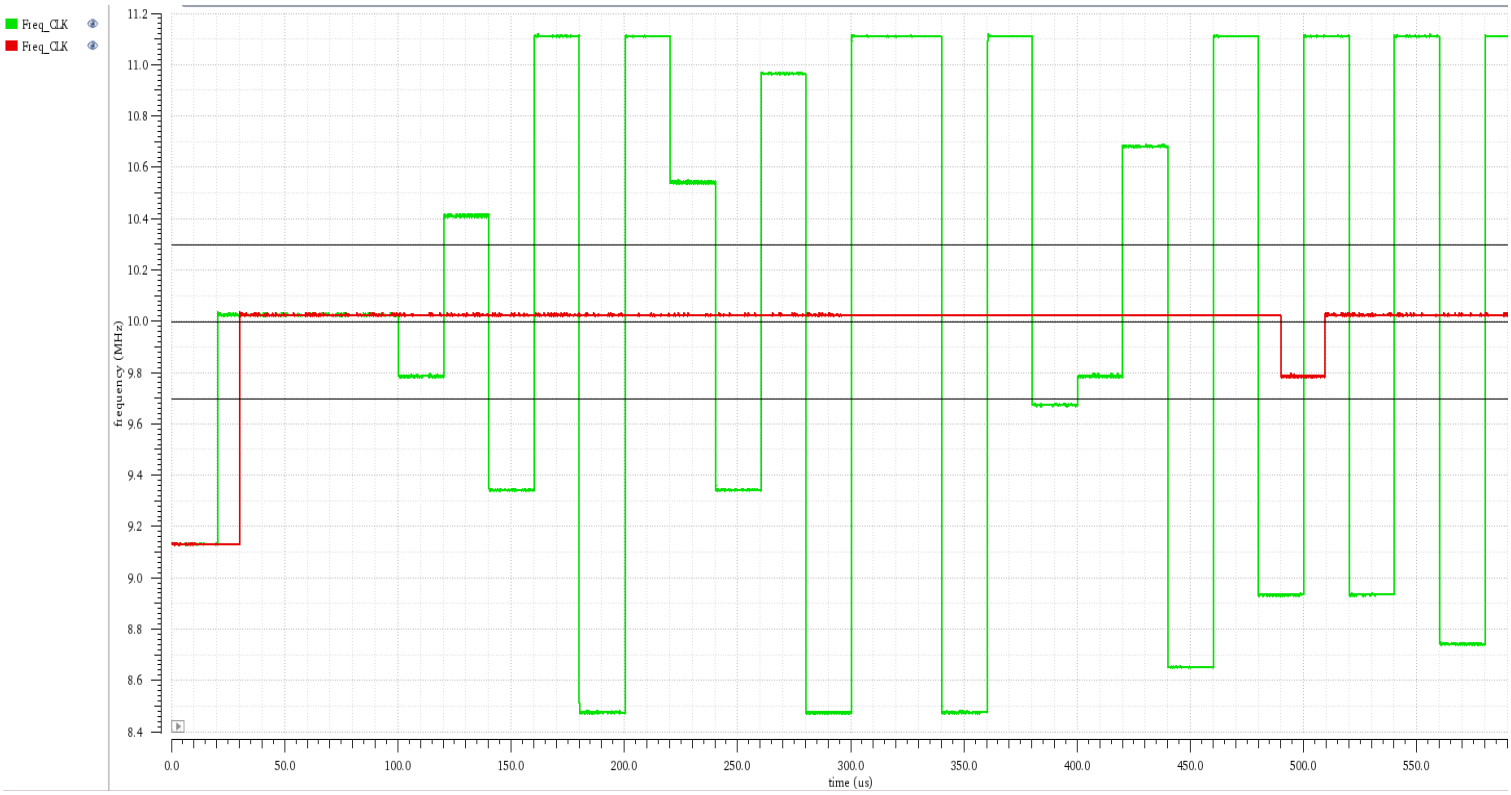

Abbildung 96: Signalverläufe der Regelgröße

Die Abbildung 96 veranschaulicht zwei Verläufe der Regelgröße. Die grüne Kurve zeigt die Reaktion der Regelgröße auf die entstehende Regelabweichung auf, bei welcher die Änderung des FTRIM-Codes nach dem Setzen des Enable-Signals umgehend erfolgt. Die rote Kurve stellt Hingegen das Regelgröße-Verhalten dar, bei welcher der FTRIM-Code erst mit Beginn der nächsten Bit aktualisiert wird. Auf Grundlage dieser Simulationsergebnisse kann abschließend festgestellt werden, dass eine sofortige Änderung des FTRIM-Codes zu starken Schwingungen der Regelgröße um den Sollwert führt und somit ein instabiles Verhalten aufweist. Dies ist eine Bestätigung der im Abschnitt 5.2.4 genannte These [T].

6.2. Regler-Robustheit

Wie in Kapitel 3, Abschnitt 3.1, bereits erwähnt, hängt die Genauigkeit der vom analogen Oszillator erzeugten Taktfrequenz von der Genauigkeit des Referenzstromflusses $\boldsymbol{I_{REF}}$ und des Referenzspannungswerts $\boldsymbol{V_{REF}}$ (siehe Gleichung 3.1.6). Eine Abweichung von diesen vorgesehenen Werten kann im Betriebzustand vorkommen, da sie von der Temperatur, der

Prozessvariation und der absorbierten Strahlungsdosis abhängig sind. Von daher kann die Variation dieser Parameterwerte als Störgröße betrachtet werden.

Diese Störgrößen, welche zwischen Eingang und Ausgang der Regelstrecke auftreten, ändern unerwünscht die Regelgröße. Diese Änderungen können jedoch unter der Voraussetzung einer richtigen Regler-Struktur und Regler-Dimensionierung ausgeregelt werden.

Zur Untersuchung der Robustheit des Reglers wird als nächstes die an die Regelstrecke eingewirkte Störgröße durch das Variieren des Referenzstromflusses $\boldsymbol{I_{REF}}$ nachgebildet. Anschließend wird die Reaktion des Reglers darauf simuliert.

Wie im Abschnitt 3.2.3 bereits erwähnt, kann die vom Oszillator erzeugte Taktfrequenz innerhalb des Intervalls $[7{,}85MHz\ 11{,}21MHz]$ variieren (siehe Tabelle 2). Basierend darauf kann die Störgröße wie folgt in einem Werteintervall definiert werden:

Der maximale Taktfrequenz-Wert $F_{osc} = 11.12\ MHz$ wird bei $I_{ref} = 24.7\ \mu A$ erreicht. Es entspricht einer Taktperiode von $T_{osc} = 89.\ 7\ ns$. Hingegen wird der minimale Wert $F_{osc} = 7.85\ MHz$ Bei $I_{ref} = 17.14\ \mu A$ erreicht. Dies entspricht einer Taktperiode von $T_{osc} = 127.\ 4\ ns$. Somit soll der Stromfluss innerhalb des Werteintervalls $[17.14\mu A\ 24.7\mu A]$ variiert werden. Dazu wird in der AMS-Simulation zunächst ein Sweep für den Parameter Referenzstromflusses $\boldsymbol{I_{REF}}$ eingerichtet.

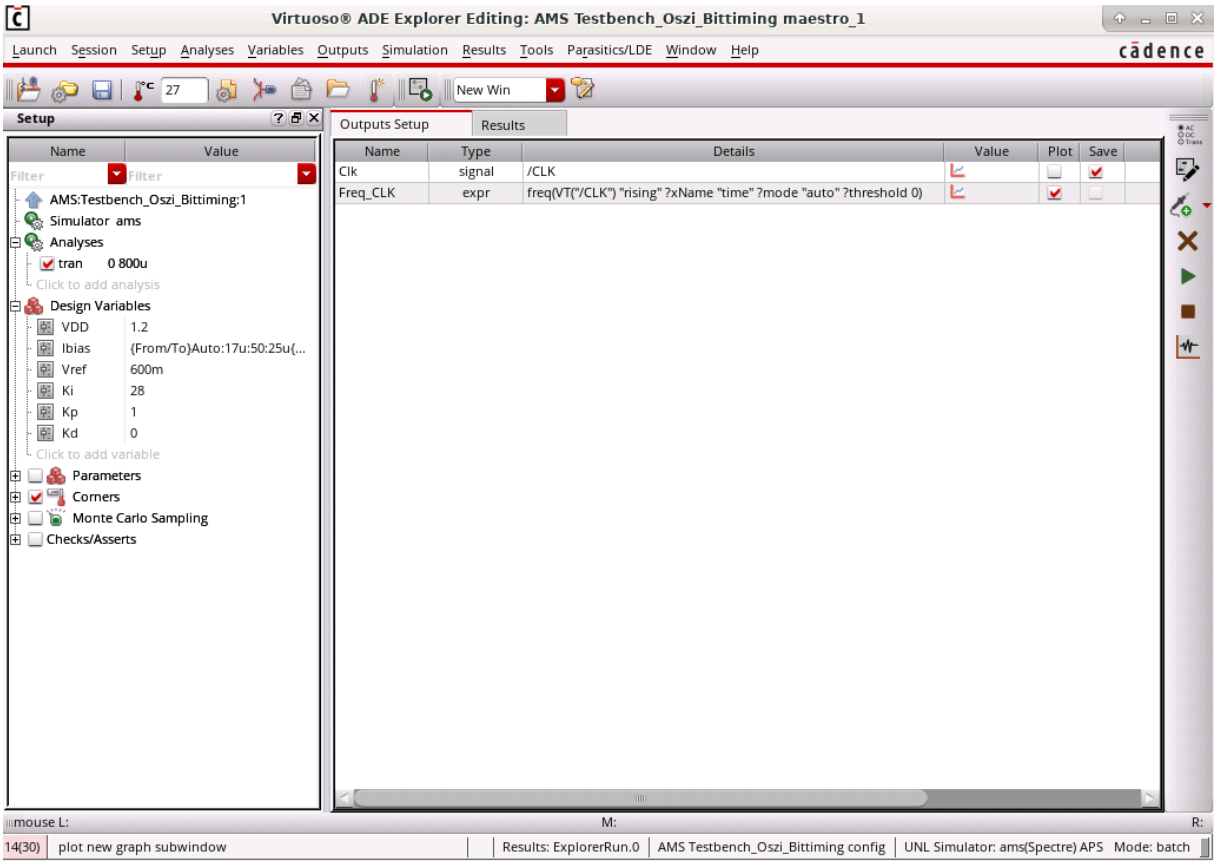

Abbildung 97: Die Simulationsumgebung

Mit dem aus dem vorherigen Abschnitt ausgewählten Regler-Parameter-Wertepaar $K_p = (0000\ 0001)_2 = (0{,}0625)_{10}$ und $K_i = (0001\_1100)_2 = (1{,}75)_{10}$ werden nachfolgende Simulationen durchgeführt. Abschließend wird die Regelgröße **Freq_CLK** als Ausdruck formuliert, der den Wert von Taktfrequenz des Clock-Signals ermittelt, und für die A/MS-Simulation als Output festgelegt.

Nach Ausführung der A/MS-Simulation kann die Robustheit des Reglers anhand der nachfolgenden Simulationsergebnisse überprüft werden.

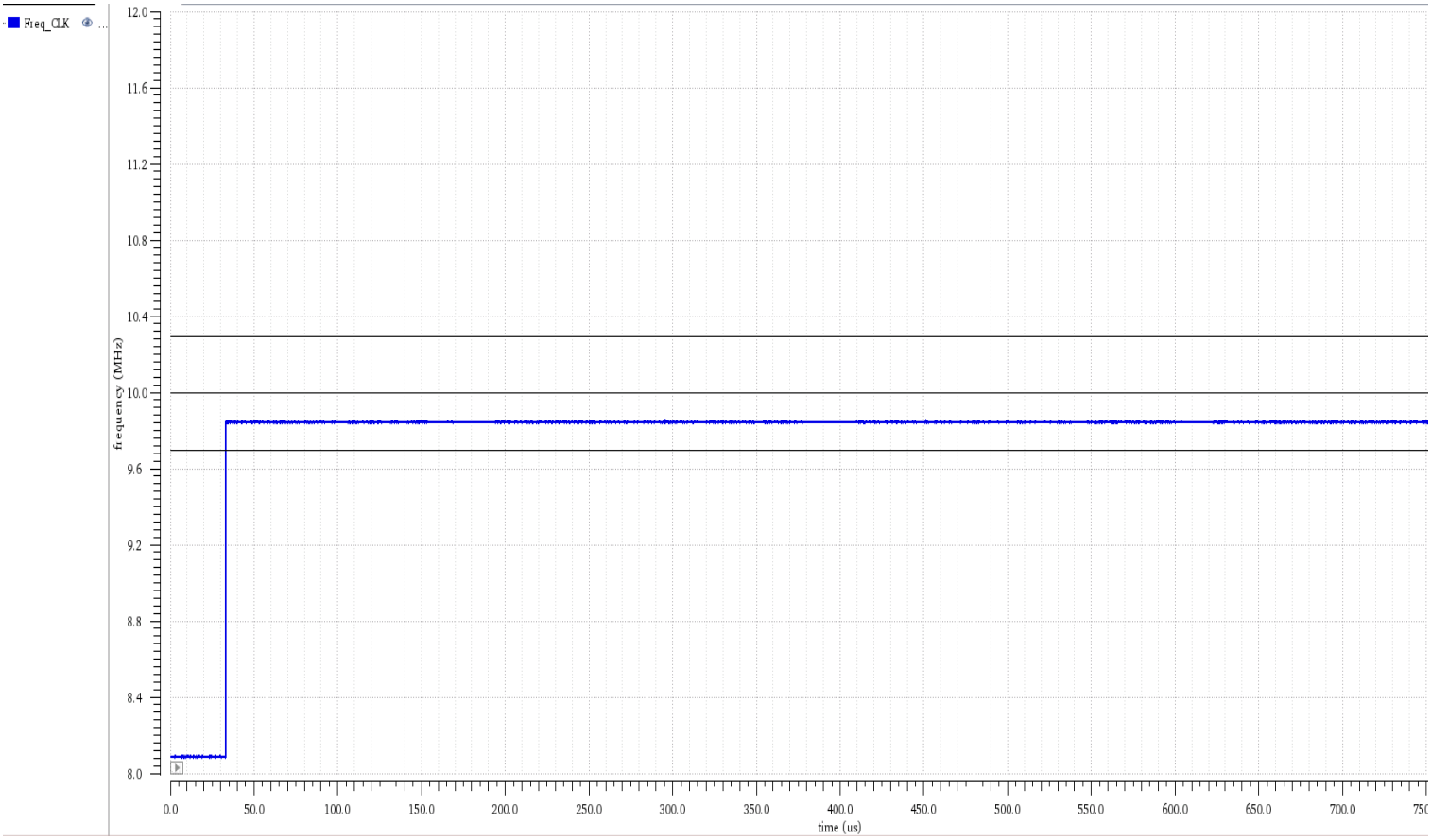

Abbildung 98: Signalverlauf Der Regelgröße für $I_{ref} = 17.65\ \mu A$

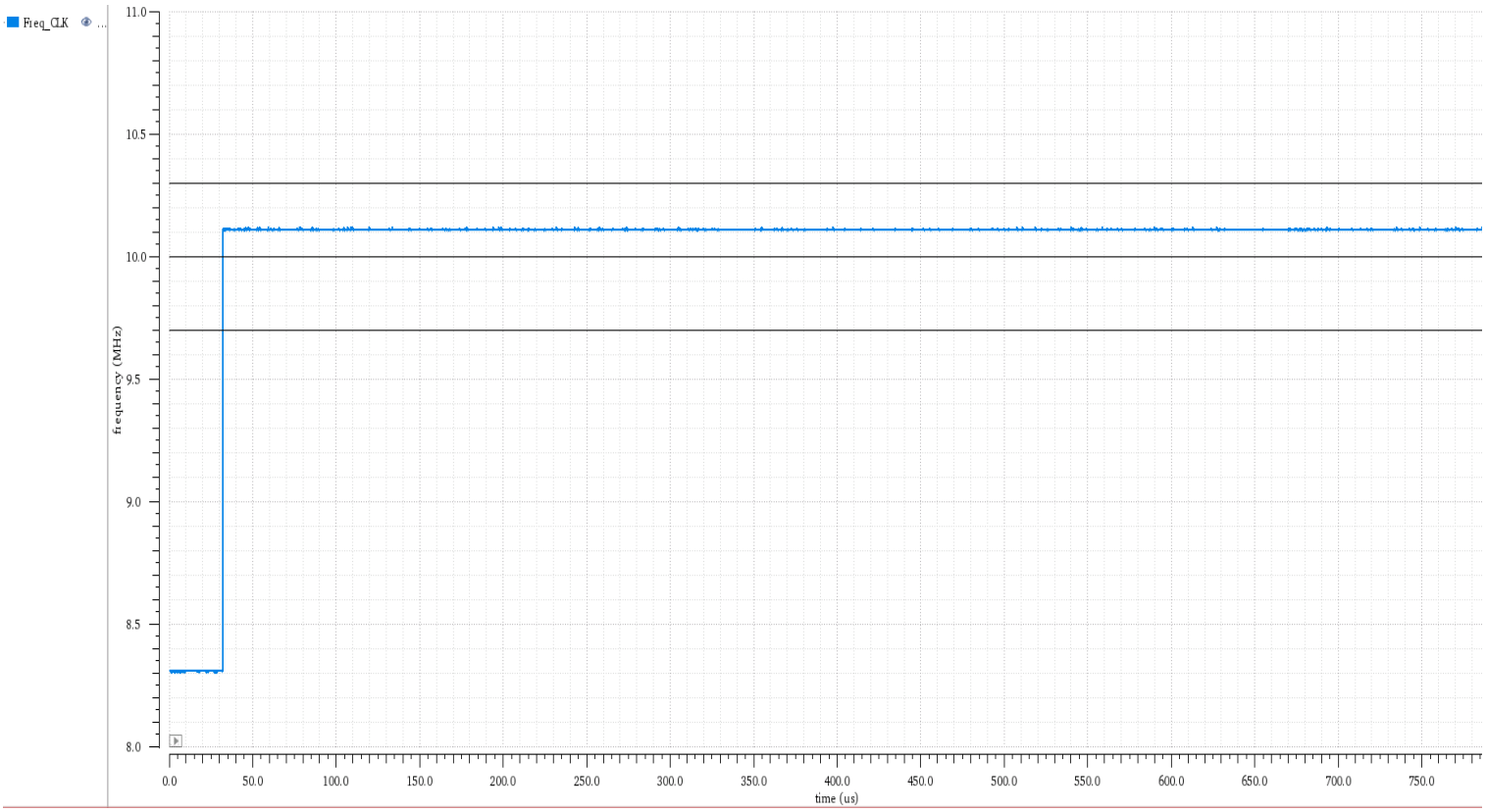

Abbildung 99: Signalverlauf Der Regelgröße für $I_{ref} = 18{,}14\ \mu A$

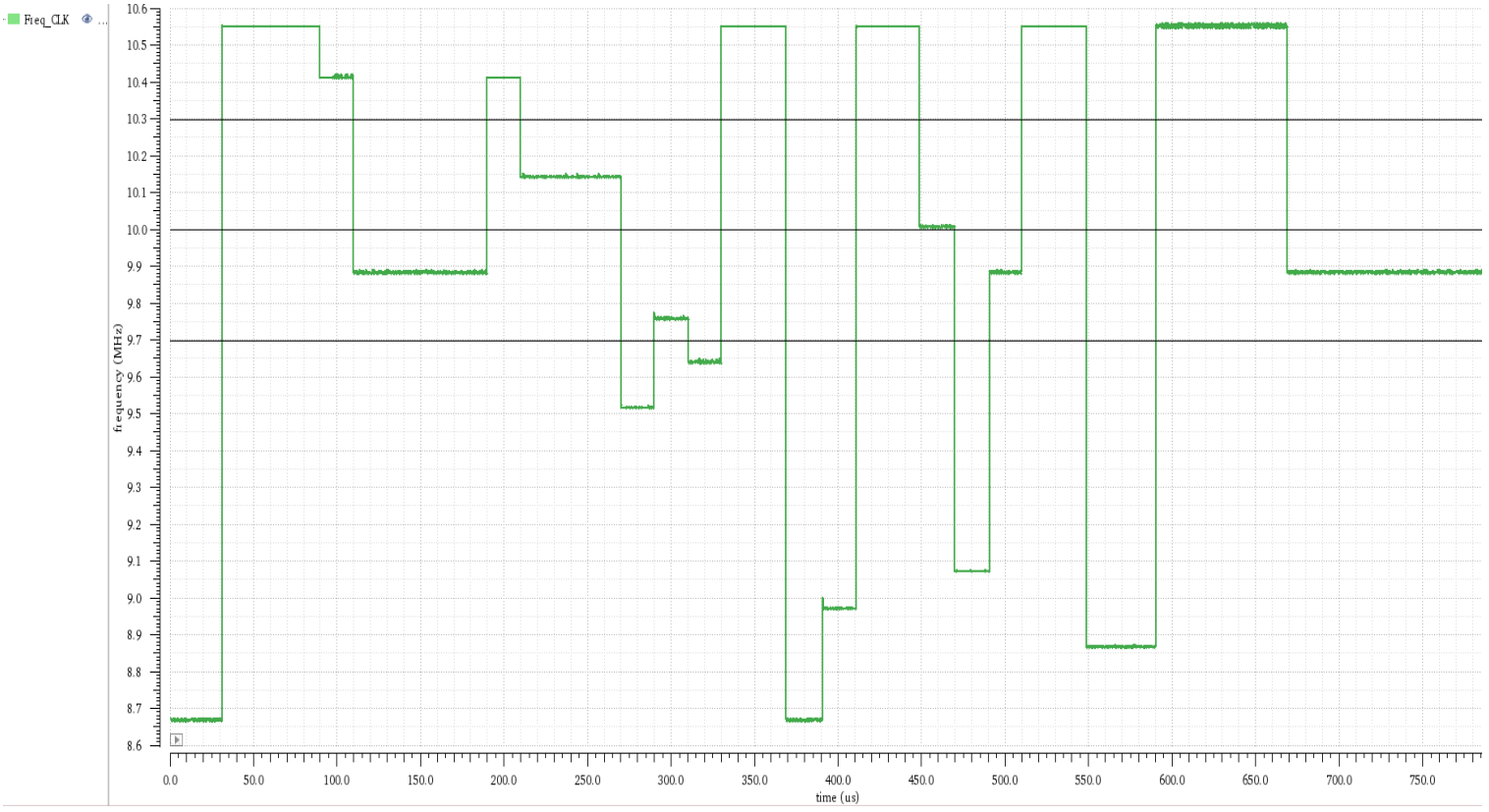

Abbildung 100: Signalverlauf Der Regelgröße für $I_{ref} = 18.95\ \mu A$

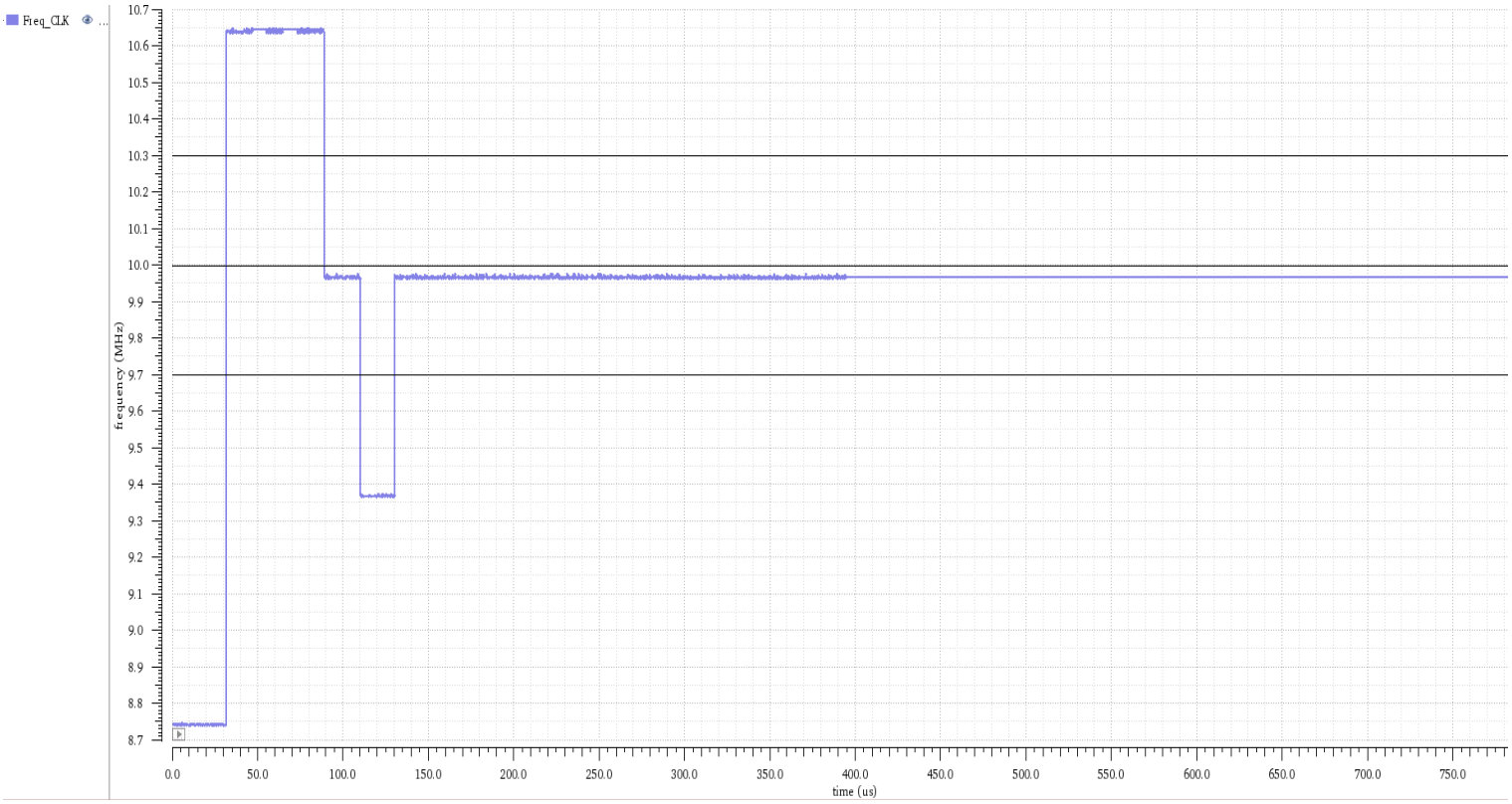

Abbildung 101: Signalverlauf Der Regelgröße für $I_{ref} = 19{,}12\ \mu A$

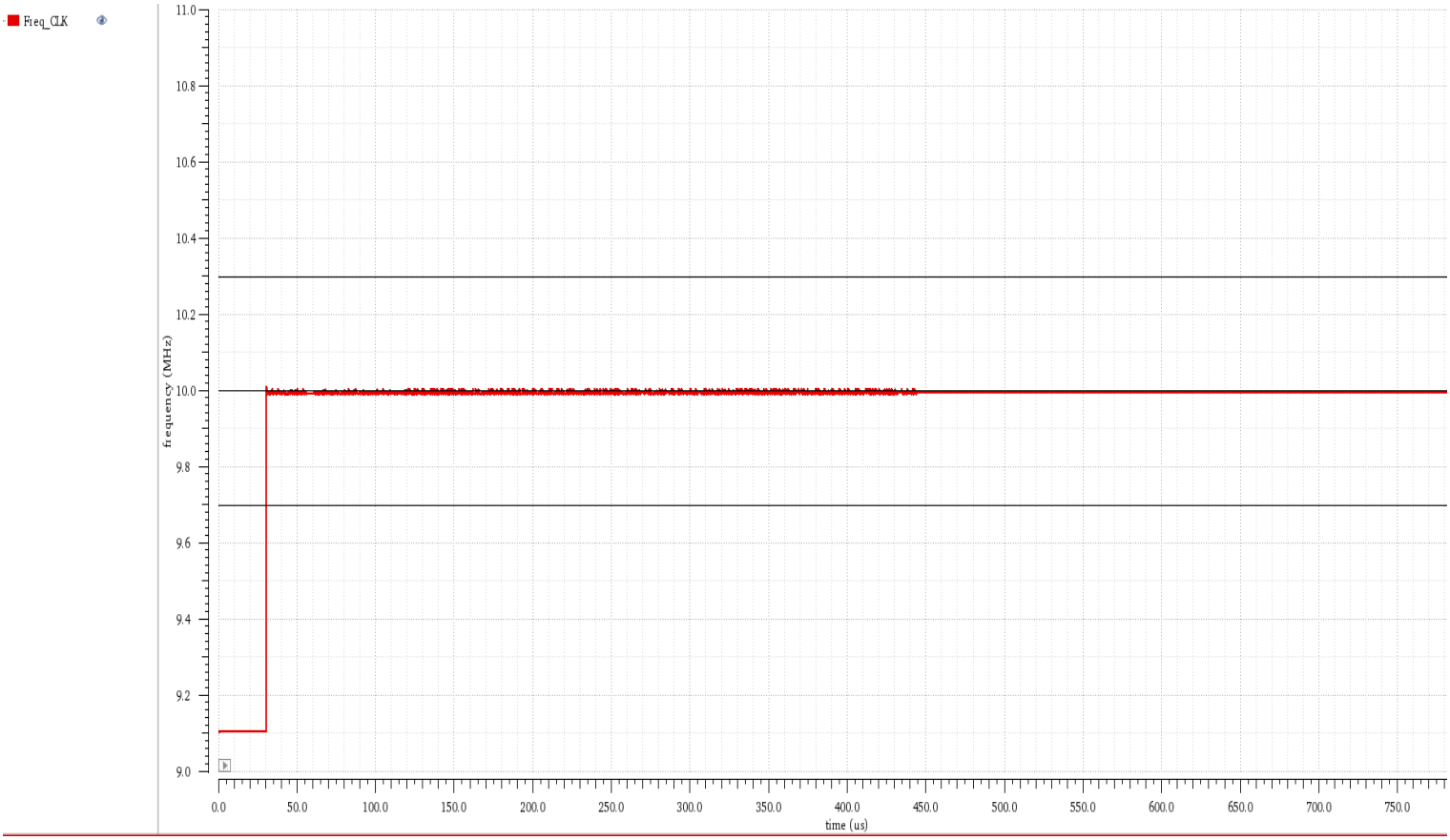

Abbildung 102: Signalverlauf Der Regelgröße für $I_{ref} = 19.94\ \mu A$

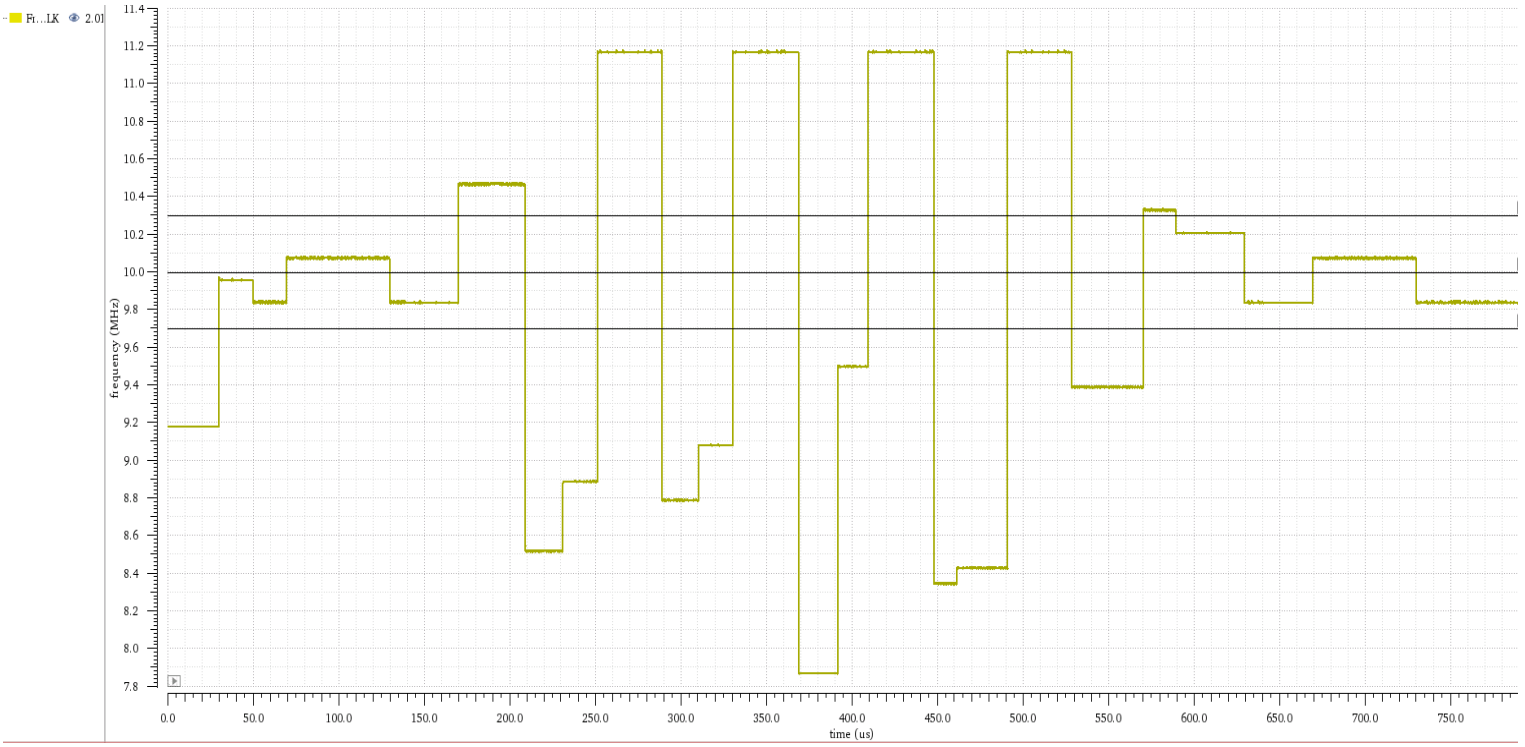

Abbildung 103: Signalverlauf Der Regelgröße für $I_{ref} = 20.11\ \mu A$

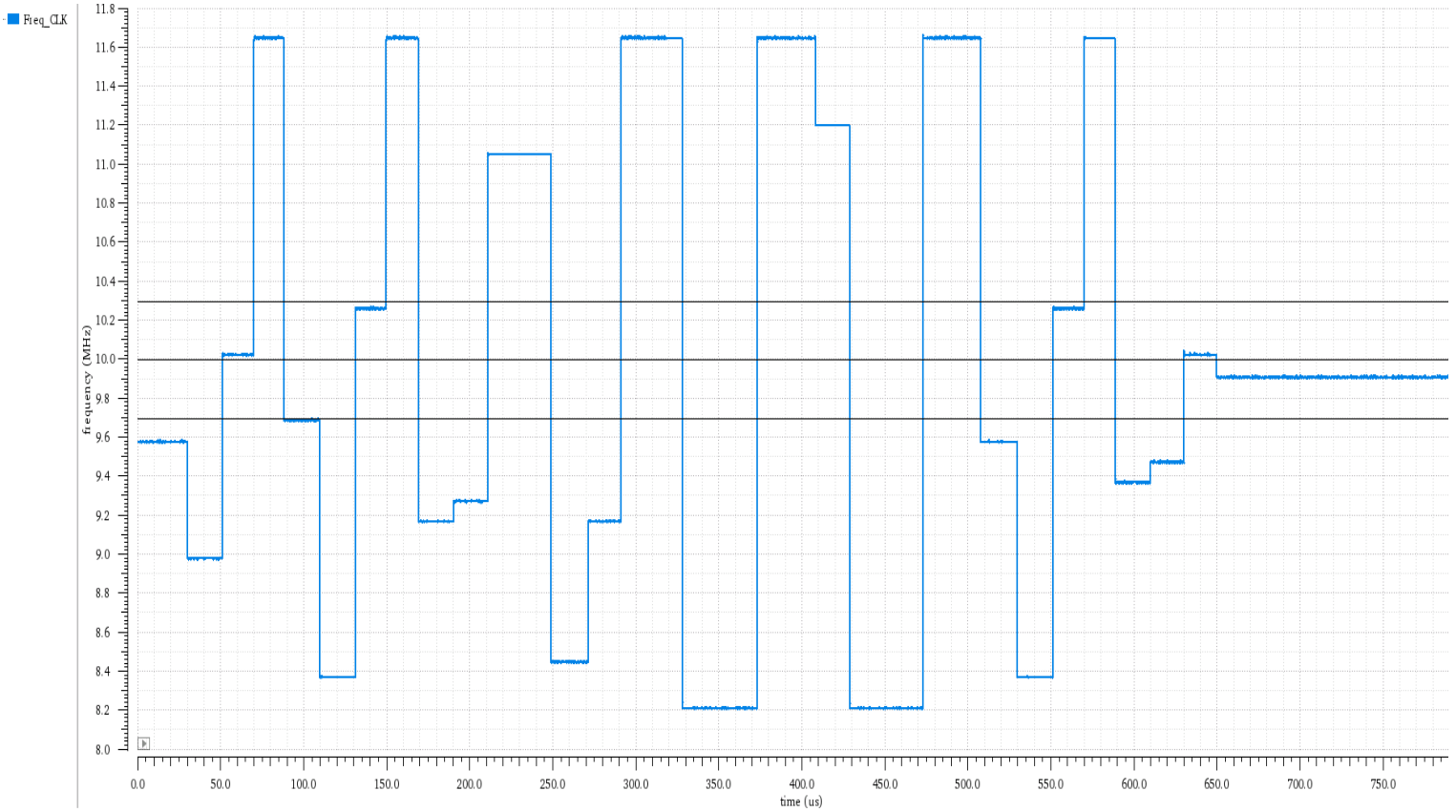

Abbildung 104: Signalverlauf Der Regelgröße für $I_{ref} = 21\ \mu A$

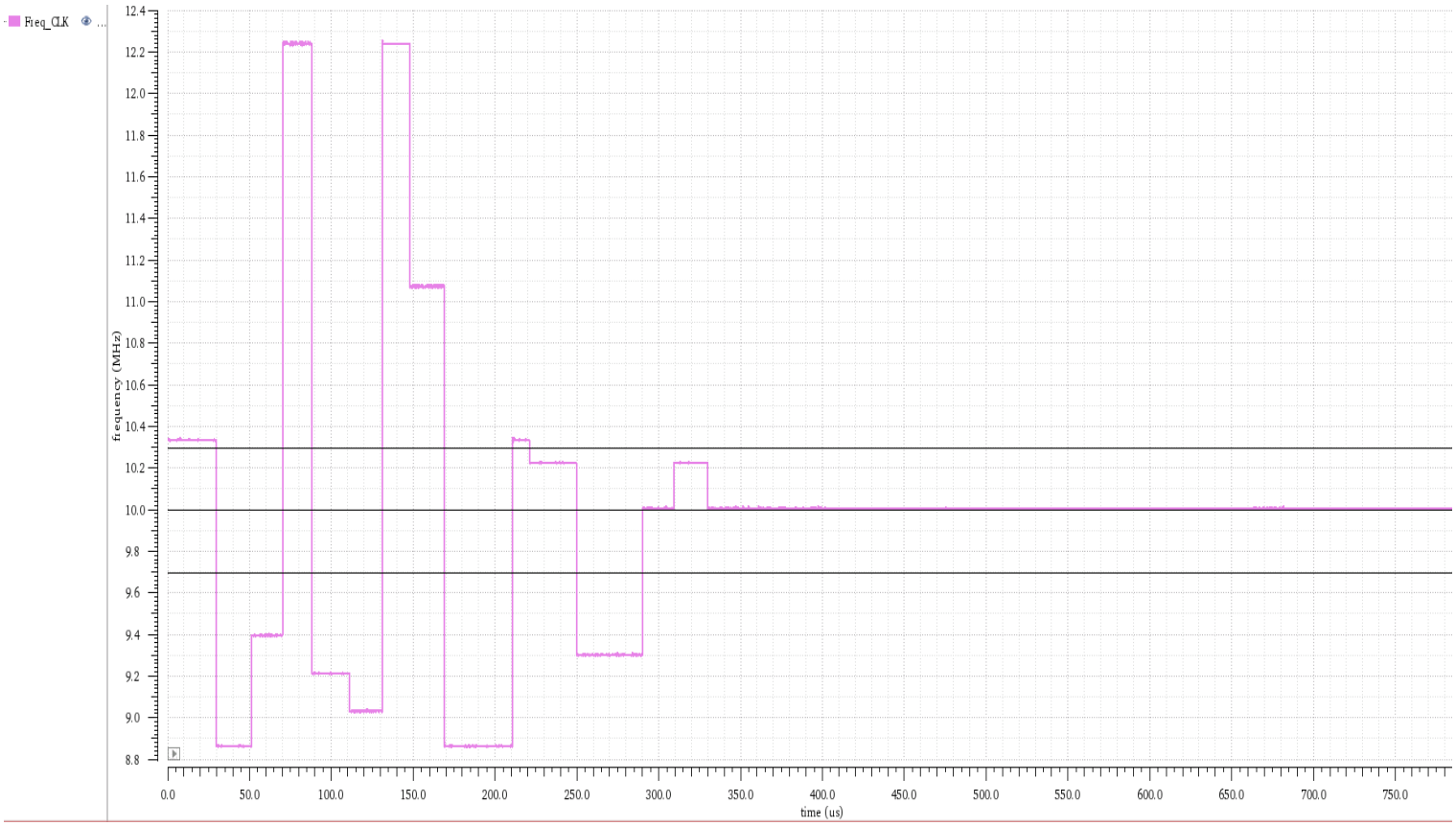

Abbildung 105: Signalverlauf Der Regelgröße für $I_{ref} = 22.71\mu A$

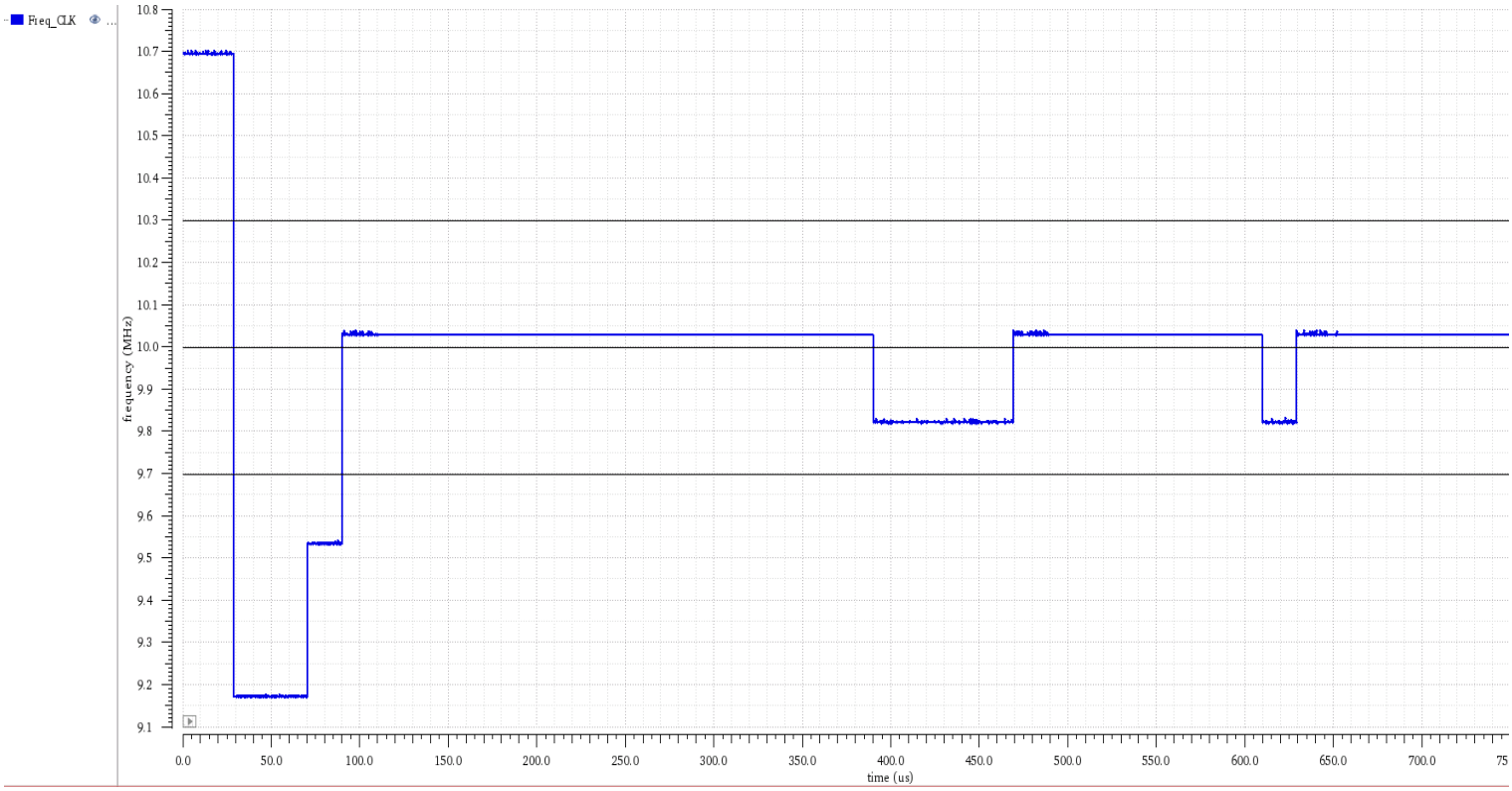

Abbildung 106: Signalverlauf Der Regelgröße für $I_{ref} = 23.53\ \mu A$

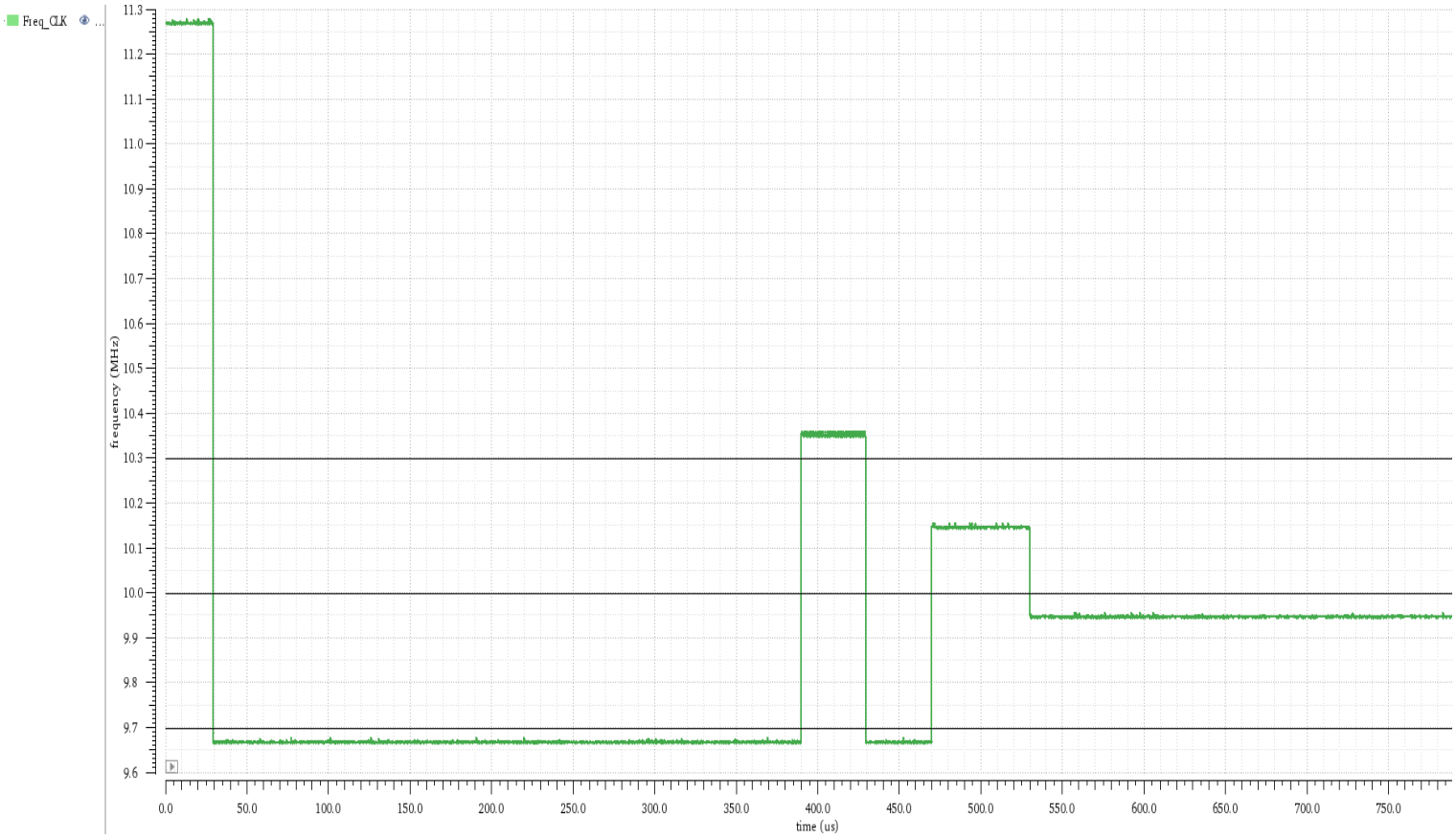

Abbildung 107: Signalverlauf Der Regelgröße für $I_{ref} = 24.7\ \mu A$

Die oben dargestellten Simulationsverläufe der Regelgröße zeigen, dass die durch Variieren des Stromflusses nachgebildete Störgröße vom Regler ausgeregelt werden konnte. Denn die Regelgröße liegt innerhalb der Oszillator-Toleranzspanne $9{,}7\ MHz < F_{osc} < 10{,}3\ MHz$ bei unterschiedlichen Stromfluss-Werten im stationären Zustand.

7. Fazit

Die Aufgabenstellung, eine ausführliche Beschreibung der CAN-Bittiming Logik zu verfassen, eine Dokumentation über den Relaxationsoszillator zu erstellen, die regelkonforme Funktionsweise des CAN-Controller-Bittiming-Moduls mit Hilfe der Analog-Mixed-Signal-Simulation zu verifizieren, ein Regelsystem zu entwerfen und mit der Bittiming-Einheit zu koppeln, um eine Frequenzregelung des analogen Relaxationsoszillator zu realisieren, ist erfüllt worden.

Da im Kapitel 4 die Synchronisationsmechanismen des CAN-Bittiming-Moduls durch A/MS Simulation geprüft werden, haben sich die Inhalte des ersten Kapitels auf die Beschreibung der Bittiming-Logik fokussiert.

Der entwickelte Relaxationsoszillator in 65nm CMOS-Technologie ist auf Grund der Verwendung von Dünn-Gate-Oxid-Transistoren strahlenhart.

Der im zweiten Kapitel vorgestellte Relaxationsoszillator wird in der gemischt analog/digitalen Testschaltung zur Verifikation des CAN-Bittiming-Moduls als Taktgeber verwendet und vom A/MS-Simulator als Analogteil betrachtet.

Die Einrichtung der A/MS-Designer-Simulation wird in Kapitel 4 detailliert erläutert.

Die A/MS-Simulation wurde in der Entwurfssoftware Virtuoso von Cadence durchgeführt. Die entsprechenden Simulationsergebnisse hierzu wurden im letzten Kapitel des Berichts ausführlich dokumentiert. Die im vierten Kapitel dargestellten Simulationsverläufe zeigen, dass die Zustände der FSM regelkonform gewechselt werden, was somit zum Abschluss der Verifizierung des Bittiming-Moduls führt.

Die in Kapitel 3 dargestellte lineare Beziehung 3.1.24 zwischen den Steuersignalen FTRIM<4:0> und der Taktperiode des analogen Relaxationsoszillators ermöglicht einen regelungstechnischen Eingriff, um die Taktfrequenz innerhalb des Toleranzintervalls $[9{,}7MHz\ 10{,}3MHz]$ zu stabilisieren.

Der Entwurf des Regelsystems, welches unter anderem aus einem PID-Regler besteht, die Implementierung des Regelsystems im Gesamtsystem und die damit verbundene Theorie wurden in Kapitel 5 umfassend erläutert.

Die in Kapitel 6 aufgeführten Simulationsergebnisse des geschlossenen Regelkreises ergaben, dass bei passender Wahl der Regler-Parameter die Regelabweichung vollständig beseitigt werden kann und die Soll-Taktfrequenz schnell erreicht wird. Darüber hinaus konnte die Funktionalität des Regelsystems und die Robustheit des Reglers anhand dieser Simulationen überprüft werden.

Aufgrund des umfangreichen und komplexen Aufbaus des analogen Oszillators und des zugehörigen Referenzgenerators nimmt die Durchführung der A/MS-Simulation eine Laufzeit von mehreren Tagen in Anspruch. Daher wurde der analoge Referenzgenerator mit idealen Strom- und Spannungsquellen ersetzt, um die Simulationsdauer zu reduzieren.

8. Anhang

8.1. Festkommazahlen

Es gibt zwei Methoden, um reelle Zahlen in der Verilog Hardware-Beschreibungsprache zu implementieren:

- Festkomma-Darstellung: Das Komma bleibt für alle Zahlen an einer beliebigen, aber fest vorgegebenen Stelle.
- Fließkomma- oder auch Gleitkomma-Darstellung: Das Komma wird so verschoben, dass signifikante Stellen erhalten bleiben [8].

In dieser Arbeit wird die Festkomma-Darstellung verwendet.

Festkomma-Zahlen setzen sich aus folgenden Teilen zusammen:

$$\boldsymbol{z = (s, i, f)} \tag{3.1.50}$$

Wobei s Vorzeichen, i ganzzahliger Anteil und f der gebrochene Anteil entsprechen.

In der Festkomma-Darstellung stellt im Allgemeinen der Bitvektor

$$\boldsymbol{(x_{vk-1}, \dots, x_1, x_0, x_{-1}, \dots, x_{-nk+1}, x_{-nk})_2} \tag{3.1.51}$$

Die Zahl dar.

$$\boldsymbol{z = \sum_{i=-nk}^{vk-1} x_i * 2^i} \tag{3.1.52}$$

Wobei vk Vorkommastellen und nk Nachkommastellen entsprechen.

Das Komma liegt somit rechts der Stelle x_0. Negative Zahlen werden durch ein Bit für das Vorzeichen oder in der Komplementdarstellung codiert. In der Signalverarbeitung wird häufig mit Zahlen kleiner als eins gerechnet [8].

Beispiel: $z = 5{,}5625_{10}, vk = 4\,Bit; nk = 4\,Bit, x = (0101\ 1001)$

Bei der Addition von Kommazahlen muss der Ergebnis-Bitvektor für eine überlauffreie Realisierung eine Bitstelle breiter sein als der größte Summand:

$$\boldsymbol{(x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 + (x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (x_4, \dots, x_0, x_{-1}, \dots, x_{-4})_2}$$

Bei der Multiplikation muss der Ergebnis-Bitvektor so breit gewählt werden, wie die Summe der Vektorbreiten der Summanden:

$$(x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 * (x_3, \dots, x_0, x_{-1}, \dots, x_{-4})_2 = (x_8, \dots, x_0, x_{-1}, \dots, x_{-8})_2$$

8.2. Verilog-Code

Control-FSM-Modul:

```
//Verilog HDL for "Control_System", "Control_FSM" "verilog"

module Control_FSM(
input wire CLK,
input wire Reset,
input wire Rx,
input wire Puffer,
input wire Pre_Out,
input wire [3:0] E1,
output reg [2:0] ctrl,
output reg Enable
);

 parameter [2:0] Normal = 3'd0, ResetZ = 3'd1, NegPhasFehler = 3'd2,
PosPhasFehler = 3'd3, Synchronisation= 3'd4,Standby= 3'd5;
 reg [3:0] current_state, next_state;
 reg [3:0] Status;
 reg [3:0] Count;

 always @(posedge CLK, negedge Reset)  // (SYNCH Process) Zustandsregister
 begin
  if (Reset == 1'b0)
        current_state <= ResetZ;
  else
        current_state <= next_state;
  end

 always @(current_state)  // (steateb_p Process) Debug Ausgangsschaltnetz
  begin
   case (current_state)
     Normal          : Status <= 4'h0;
     ResetZ          : Status <= 4'h1;
     NegPhasFehler   : Status <= 4'h2;
     PosPhasFehler   : Status <= 4'h3;
     Synchronisation : Status <= 4'h4;
     Standby         : Status <= 4'h5;
     default         : Status <= 4'h6;
    endcase
  end

  always @(current_state, Rx, Puffer,E1,Pre_Out)
  begin
    Enable=0;
    case (current_state)
```

```
      ResetZ : begin
        ctrl = 3'd0;
        Enable=0;
        next_state = Normal;
      end
      Normal: begin
         ctrl = 3'd0;
        if (Rx == 1'b0 && Puffer == 1'b1 && E1<5 && E1>0)
            begin
                next_state = PosPhasFehler;
            end
         else if ((Rx == 1'b0 && Puffer == 1'b1 && E1==0)||(Rx == 1'b0 && Puffer
== 1'b1 && E1==9 && Pre_Out==1))
            begin
                next_state = Synchronisation;
            end
         else if (Rx == 1'b0 && Puffer == 1'b1 && E1>=5)
            begin
                next_state = NegPhasFehler;
            end
         else
            next_state = Normal;
      end
      PosPhasFehler: begin
          ctrl = 3'd1;
          Enable=1;
          next_state = Standby;
      end
      NegPhasFehler: begin
          ctrl = 3'd2;
          Enable=1;
          next_state = Standby;
      end
      Synchronisation: begin
          ctrl = 3'd3;
          Enable=1;
          next_state = Standby;
      end
      Standby: begin
          ctrl = 3'd0;
          Enable=0;
          if (Rx==1'b0 && Puffer==1'b1)
                next_state = Standby;
          else
                next_state = Normal;
      end
          default: begin
          ctrl = 3'd0;
          Enable=0;
          next_state = current_state;
      end

     endcase
  end
endmodule
```

CLK-Counter-Modul:

```
//Verilog HDL for "Control_System", "CLK_Counter" "verilog"

module CLK_Counter (
input wire clk,
input wire reset,
input wire puffer,
input wire rx,
input wire Enable,
output [3:0] Counter_C, //Zhler wird inkrementiert bei jedem High Pegel
CLK
output reg [3:0] Counter_C1 // in diesem Reg. wird der Zhlerstand
gespeichert, bei dem eine fallende Flanke aufgetreten ist
    );
reg [3:0] Count;
reg [3:0] k1;

always @(posedge clk,negedge reset)
begin
    if (reset==0)
        begin
            Count<=0;
            Counter_C1<=0;
            k1<=0;
        end
    else
        begin
            Count<=Count+1;
            if (Enable)
                Count<=1;
            if (rx==1)
                begin
                    k1<=Count;
                    Counter_C1<=0;
                end
            if (rx==0 && puffer)
                begin
                    if (k1==9)
                        Counter_C1<=0;
                    else
                        Counter_C1<=k1+1;
                end
           if (Count==4'd9)
                Count<=0;
        end
end
assign Counter_C=Count;
endmodule
```

PhasenfehlerReg-Modul:

```
//Verilog HDL for "Control_System", "Phasenfehler_Reg" "verilog"


module Phasenfehler_Reg (
input wire clk,
input wire Reset,
input wire Enable,
input wire [3:0] E2, //Clk-Counter
input wire [3:0] E1, //Bittiming-Counter
input wire [2:0] ctrl,
output reg signed [7:0] e_k
);

 always @(posedge clk, negedge Reset)
 begin
  if (Reset == 1'b0)
    e_k <= 8'd0;
  else
   begin
     if (Enable==1)
     begin
        case (ctrl) // umschalten
               3'd1 : e_k <= (10*(E1-1))+ E2;  // positive Phasenfehler
               3'd2 : e_k <= (10*(E1-9))+(E2-10);  // negative Phasenfehler
               3'd3 : e_k <= 0;
               default : e_k <= e_k; // halten
        endcase
     end
   end
 end
endmodule
```

PID-Regler-Modul:

Multi-Modul:

```
//Verilog HDL for "Control_System", "Multi" "verilog"


module Multi #(
parameter WidthA = 8,
parameter WidthB = 8
) (
  input wire signed [WidthA - 1:0] a_i,
  input wire signed [WidthB - 1:0] b_i,
  output wire signed [WidthA + WidthB - 1:0] prod_o
);

  assign prod_o = a_i * b_i;

endmodule
```

Add-Modul:

```
//Verilog HDL for "Control_System", "Add_PID" "verilog"


module Add_PID#(
        parameter WidthA=16,
        parameter WidthB=16,
        parameter WidthC=16,
        parameter WidthD=16
)(
        input wire signed [WidthA-1:0] A,
        input wire signed [WidthB-1:0] B,
        input wire signed [WidthC-1:0] C,
        output wire signed [WidthC-1:0] D
);

        assign D=A+B+C;

endmodule
```

Accumulator-Modul:

```
//Verilog HDL for "Control_System", "Accumulator" "verilog"


module Accumulator #(
  parameter WidthA = 8,
  parameter WidthAcc = WidthA  //+4
) (
  input wire clk_i,
  input wire rst_ni,
  input wire Enable,
  input wire signed [WidthA - 1:0] a_i,
  output reg signed [WidthAcc - 1:0] sum_o
);

  localparam Max = (2 ** (WidthAcc - 1)) - 1; //(2^(12-1))-1
  localparam Min = - (2 ** (WidthAcc - 1));   //-(2^(12-1))


  reg signed [WidthAcc - 1:0] acc;
  reg signed [WidthAcc - 1:0] acc_added;


  always @* begin
    acc_added = acc + a_i;

    if (acc > 0 && a_i > 0 && acc_added < 0) begin
      sum_o = Max;
    end else if (acc < 0 && a_i < 0 && acc_added > 0) begin
      sum_o = Min;
    end else begin
```

```
      sum_o = acc_added;
    end
  end

  always @(posedge clk_i, negedge rst_ni) begin
    if (rst_ni == 1'b0) begin
      acc <= 5'b10000;
    end else if (Enable) begin
      acc <= sum_o;
    end
  end
endmodule
```

Diff-Modul:

```
//Verilog HDL for "Control_System", "Diff" "verilog"


module Diff #(
  parameter Width = 8
) (
  input wire CLK,
  input wire Reset,
  input wire Enable,
  input wire signed [Width - 1:0] a_i,
  output wire signed [Width - 1:0] diff_o
);

  reg signed [Width - 1:0] a_delayed;

  always @(posedge CLK, negedge Reset) begin
    if (Reset == 1'b0) begin
      a_delayed <= 0;
    end else if (Enable) begin
      a_delayed <= a_i;
    end
  end

  assign diff_o = a_i - a_delayed;

endmodule
```

FTRIM-En-Modul:

```
//Verilog HDL for "Control_System", "Ftrim_En" "verilog"


module Ftrim_En (
input wire CLK,
input wire reset,
input wire Enable,
input wire Pre_En,
input wire Sendpoint,
input wire [2:0] ctrl,
output reg PI_En
);

reg [3:0] couto;
reg test;
reg SP;
reg [2:0] CT;

always @(posedge CLK, negedge reset) // pos. flanke (clock), asynchroner reset
begin
    if (reset==1'b0)
        begin
                        SP<=0;
            couto<=0;
            test<=0;
            PI_En<=0;
        end
    else
        begin
            if (Enable == 1)
               begin
                     CT<=ctrl;
                     test<=1;
                     couto<=0;
               end
             SP<=Sendpoint;
            if (SP==1 && Sendpoint==0 && test==1)
                couto<=couto+1;
            if ((couto==3'd2 && CT==3'd2)||(couto==3'd1 && CT==3'd1)||(couto==3'd1
&& CT==3'd3))
               begin
                     if (test==1)
                     begin
                         PI_En<=1;
                         test<=0;
                         couto<=0;
                     end
               end
            if (test==0)
               begin
                   PI_En<=0;
               end
        end
end
endmodule
```

Output_Scaling -Modul:

```
//Verilog HDL for "Control_System", "Output_Scaling" "verilog"


module Output_Scaling#(
    parameter WidthA=16,
    parameter WidthB=5
)(
    input wire signed [WidthA-1:0] A,
    input wire Reset,
    input wire CLK,
    input wire Enable,
    output reg [WidthB-1:0] B
    );

    localparam Max = (2**WidthB)-1;

    always @(posedge CLK, negedge Reset)
    begin
        if (Reset==0)
            B<=5'b10000;
        else
            begin
                if (Enable)
                begin
                    if (A>0 && A[15:4]>Max )
                        B <= Max;
                    else if (A<=0)
                        B <= 0;
                    else
                        B <= A[8:4];
                end
            end
    end
endmodule
```

9. Literaturverzeichnis

[1] B. Cimbili, D. W. (09. 02 2017). *A PVT-Tolerant Relaxation Oscillator in 65nm CMOS*. Von IEEE Xplore: https://ieeexplore.ieee.org/document/7848442 abgerufen

[2] Karagounis, M. (2000). *Design eines CAN Controllers mit VHDL und SpecCharts.* Fachhochschule Köln.

[3] Cadence Design Systems. (2020). *Introduction to AMS Designer Simulation, Rapid Adoption Kit (RAK).* Cadence Design Systems. Von https://support.cadence.com abgerufen

[4] Cadence Design Systems. (2019). *Spectre AMS Designer and Xcelium Simulator.* Cadence Design Systems.

[5] Karagounis, M. (Wintersemester 17/18). Computer unterstützter Entwurf in der Mikroelektronik, Kapitel 2: Transistorkennlinien, Folie 5.

[6] Lunze, J. (2008). *Automatisierungstechnik.* Bochum: Oldenbourg Verlag München Wien.

[7] Lunze, J. (Oktober 2019). *Regelungstechnik 2, Mehrgrößensysteme, Digitale Regelung.* Bochum: Springer Vieweg.

[8] Gessler, R. (2020). *Entwicklung Eingebetteter Systeme.* Ravensburg: Springer Vieweg.

[9] BAKER, R. J. (2010). *CMOS CIRCUIT DESIGN, LAYOUT, AND SIMULATION.* John Wiley & Sons, Inc., Hoboken, New Jersey.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass die von mir vorgelegte Prüfungsleistung selbstständig und ohne unzulässige fremde Hilfe erstellt worden ist. Alle verwendeten Quellen sind in der Arbeit so aufgeführt, dass Art und Umfang der Verwendung nachvollziehbar sind.

Dortmund, den 06. Juli 2021