HAW HAMBURG

BACHELORTHESIS
Fabian Kappelmann

Konzeption und Umsetzung eines Demonstrators für einen Bellcore-Angriff

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Fabian Kappelmann

Konzeption und Umsetzung eines Demonstrators für einen Bellcore-Angriff

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Elektro- und Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Heike Neumann
Zweitgutachter: Prof. Dr. Annabella Rauscher-Scheibe

Eingereicht am: 26.03.2020

Fabian Kappelmann

Thema der Arbeit

Konzeption und Umsetzung eines Demonstrators für einen Bellcore-Angriff

Stichworte

RSA, RSA-CRT, Bellcore Angriff, Spannungsversorgungs-Glitch, Fehlerinjektion, Mikrocontroller, Hardwaresicherheit

Kurzzusammenfassung

Diese Thesis beschreibt die Entwicklung und Umsetzung eines Demonstrators für einen Bellcore Angriff. Dafür wird ein Mikroconroller auf die Anfälligkeit eines Spannungsversorgungs-Glitches untersucht und anschließend auf diesen Grundlagen ein Demonstrator entwickelt.

Fabian Kappelmann

Title of Thesis

Design and implementation of an demonstrator for a Bellcore attack

Keywords

RSA, RSA-CRT, Bellcore attack, voltage fault injection, power glitch, fault injection, microcontroller, hardware security

Abstract

This thesis describes the design and implementation of an demonstrator for a Bellcore attack. First of all, a microcontroller was examined for vulnerabilities of a voltage fault injection. After that the demonstrator based on the results was created.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

CCS Code Composer Studio.

CPU Central Processing Unit.

CRT Chinesischer Restsatz.

D-FF Delay-Flip-Flop.

DAVE Digital Application Virtual Engineer.

DPA Differential Power Analysis.

DUT Device Under Test.

EMF Elektromagnetisches Feld.

GGT größter gemeinsamer Teiler.

GPIO General Purpose Input/Output.

GPTM General-Purpose Timer.

IDE integrierte Entwicklungsumgebung.

LSB Least Significant Bit.

MOSFET Metall-Oxid-Halbleiter-Feldeffekttransistor.

MSB Most Significant Bit.

RS-FF Reset-Set-Flip-Flop.

RSA Rivest-Shamir-Adleman.

SFM Straightforward Method.

SOC Sytem-on-a-Chip.

SPA Simple Power Analysis.

UART Universal Asynchronous Receiver Transmitter.

USB Universal Serial Bus.

V-FI Voltage Fault Injection.

VDDC Core-Spannungsversorgung.

VDDP Pad Spannungsversorgung.

VSS Massepotenzial des Boards.

1 Einleitung

Die Digitalisierung schreitet immer weiter voran und ist in der heutigen Zeit nicht mehr nur Bestandteil der Industrie. Schon jetzt haben es viele intelligente, mit dem Internet verknüpfte Geräte in den Haushalt privater Personen geschafft. Sei es das Handy, welches man sich aus dem heutigen Alltag nicht mehr wegdenken kann, oder der intelligente Kühlschrank, welcher über eine Internetverbindung verfügt. Diese Anbindung an das Internet ermöglicht es dem Kühlschrank, quasi zu jedem Zeitpunkt in nahezu Echtzeit mit anderen Endgeräten zu kommunizieren. So eröffnen sich neue Möglichkeiten für eine smarte Nutzung. Ein Beispiel dafür ist, dass ein Kühlschrank zu jeder Zeit weiß, welche Lebensmittel in welcher Menge verfügbar sind. Dies führt dazu, dass der Kühlschrank Produkte selbstständig nachbestellen kann.
Neben den vielen neuen Möglichkeiten, die durch diesen Fortschritt entstehen, treten jedoch auch Probleme auf:

(1) Wie können diese Endgeräte miteinander kommunizieren, ohne das ein drittes Gerät die Kommunikation abhört?

(2) Wie kann sichergestellt werden, dass die Nachricht nicht verändert wird?

(3) Wie kann zweifelsfrei bewiesen werden, wer der Absender der gesendeten Nachricht ist?

Während sich die Frage (1) mit der Vertraulichkeit der Kommunikation beschäftigt und die Frage (2) das Thema der Integrität einer Nachricht hinterfragt, befasst sich die Frage (3) mit der Authentizität der Nachricht.

Ein in der Praxis häufig verwendeter Krypto-Algorithmus ist der Rivest-Shamir-Adleman (RSA) [RSA78]. Der zugrundeliegende Algorithmus ist recht einfach und ermöglicht es eine Nachricht zu Verschlüsseln, Entschlüssen und einer Nachricht eine digitale Signatur

hinzuzufügen. Das Ziel dieser Thesis ist es, eine mögliche Implementierung des Signaturverfahrens des RSA anzugreifen. Die dabei verwendete Implementierung ist der Chinesischer Restsatz (CRT). Der CRT wird auf dem Infineon-Entwicklungsboard (XMC4700 Relax Kit Lite Board [Rel16]) programmiert. Anschließend wird ein Bellcore Angriff durchgeführt. Zur Veranschaulichung der Attacke wird ein Demoprojekt erstellt.

Diese Thesis startet mit den allgemeinen Grundlagen, gefolgt von Erklärung des RSA (Algorithmus und Implementierung). Außerdem befinden sich im Grundlagenkapitel die verschiedenen Arten von Attacken, um den RSA zu brechen, und der Bellcore-Angriff. Im mittleren Teil werden der Versuchsaufbau und das verwendete Equipment näher erklärt. Außerdem werden hier bestimmte Vorversuche durchgeführt, wie z. B. die genaue Bestimmung der Glitchdauer und der Verzögerungszeit. Zuletzt werden der Bellcore-Angriff durchgeführt und die Ergebnisse dargestellt.

2 Grundlagen

2.1 Delay-Flipflop

In der Digitaltechnik wird zum Speichern von Informationen ein sogenanntes Delay-Flip-Flop (D-FF) verwendet. Dieses verfügt über einen Informationseingang D, einen Takteingang C sowie über zwei Ausgänge Q und $\widetilde{Q}$. Wechselt das Taktsignal von 0 auf 1 (steigende Flanke), wird der Informationseingang D abgetastet. Das abgetastete Signal bzw. der Wert des abgetasteten Signales (0 oder 1) wird an den Ausgang Q weitergeleitet und für $\widetilde{Q}$ invertiert. Die Ausgänge behalten, solange ihren Wert, bis die nächste steigende Taktflanke am D-FF anliegt. Da in den meisten Fällen die Information am Informationseingang D anliegt, bevor die steigende Flanke des Taktsignals erscheint, ergibt sich hier eine Verzögerungszeit t_d zwischen dem Informationseingang und den Ausgängen des D-FF. Ein D-FF kann durch die folgende Wahrheitstabelle dargestellt werden:

Tabelle 2.1: Wahrheitstabelle für ein D-FF (modifiziert nach [Sca09])

C	D	Q_{n-1}[1]	Q[2]	$\widetilde{Q}$[3]
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

[1] Vorheriger Ausgangswerte, Startet mit dem Wert 0
[2] Ausgangswert: $Q = \overline{C}Q + CD$
[3] Ausgangswert: $\widetilde{Q} = \overline{Q}$

In der Wahrheitstabelle (Tabelle 2.1) lassen sich drei Hauptzustände erkennen:

- Der vorherige Q-Zustand wird immer dann gespeichert, wenn das Taktsignal 0 ist (Speichen).
- Der Q-Ausgang wird gesetzt, wenn das Taktsignal und das Datensignal 1 sind (Setzen).
- Der Q-Ausgang wird zurückgesetzt, wenn das Taktsignal 1 ist und das Datensignal 0 ist (Zurücksetzen).

Aus diesen drei Zuständen lässt sich eine NAND-Schaltung für ein taktzustandsgetriggertes D-FF entwerfen [Sca09]:

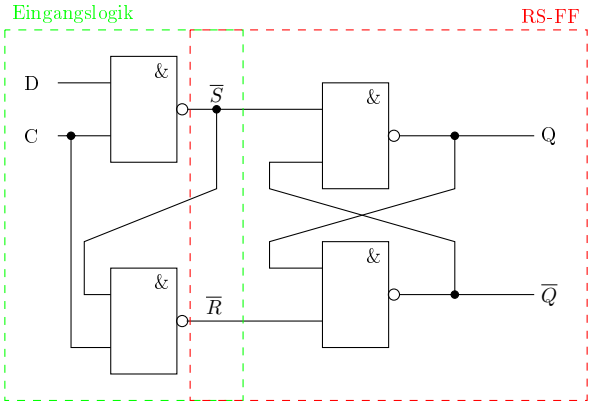


Abbildung 2.1: D-FF als NAND-Schaltung (modifiziert nach [Sca09])

Es ist zu sehen, dass die Schaltung aus einer Eingangslogik (grün) und einem Reset-Set-Flip-Flop (RS-FF) besteht (vgl. Abbildung 2.1). Die Eingangslogik legt fest, welchen der drei Zustände das RS-FF annehmen soll.

2.2 Assembler

Der Assembler ist ein Programm, welches den geschriebenen Programmcode in Maschinensprache übersetzt. Er stellt somit eine Schnittstelle zwischen Programmiersprache und Maschinensprache dar. Meist ist es so, dass der Hochsprachencompiler zunächst den Programmcode in Assemblersprache übersetzt und dieser vom Assembler in Maschinensprache umgewandelt wird (siehe Abbildung 2.2). Anders als ein Hochsprachencompiler

orientiert sich der Assembler nicht an der menschlichen Sprache, sondern an der für den Chip verwendeten Architektur. Dies bedeutet, dass ein Assembler an die verwendete Architektur gebunden ist und sich dadurch Assembler unterschiedlicher Architekturen unterscheiden können. In allgemeiner Form sieht ein Assemblerbefehl folgendermaßen aus:

Befehl Ziel [,Quelle][4]

Der Assemblerbefehl besteht aus einem Befehl, wie z. B. „mov“ und einem Ziel, welches den neuen Speicherort angibt. Optional kann der Befehl auch eine Quelle enthalten. Das Ziel und die Quelle werden durch ein Komma getrennt.

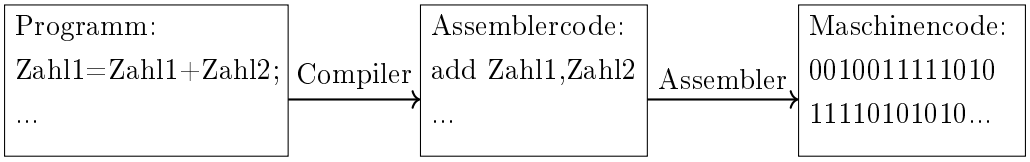


Abbildung 2.2: Übersicht über Compilerschritte

Beispiele für Assemblerbefehle sind:

- **mov Zahl1, Zahl2** Der mov-Befehl verschiebt die *Zahl*2 in die *Zahl*1.
- **add Zahl1, Zahl2** Der add-Befehl addiert die beiden Zahlen und speichert das Ergebnis anschließend in *Zahl*1.
- **sub Zahl1, Zahl2** Der sub-Befehl subtrahiert die beiden Zahlen und speichert das Ergebnis anschließend in *Zahl*1.

Hierbei stehen *Zahl*1 und *Zahl*2 nicht direkt für Zahlen, sondern für die Adressen an denen die Zahlen gespeichert sind.

[4] [] kennzeichnet eine optionale Angabe

2.3 RSA-Algorithmus

2.3.1 Allgemein

Die in diesem Kapitel beschrieben Informationen sind sinngemäß aus [RSA78] übernommen. Der RSA gehört zu den asymmetrischen Verschlüsselungsalgorithmen. Das bedeutet, dass er ein sogenanntes „public Key"-Verfahren verwendet. Bei einem „public-Key"-Verfahren besitzt jeder Teilnehmer einen privaten Schlüssel, einen öffentlichen Schlüssel und private Parameter. Der öffentliche Schlüssel wird frei zugänglich (z. B. auf einem öffentlichen Server) abgelegt. Im Gegensatz dazu muss der private Schlüssel gegen unbefugten Zugriff/Verwendung geschützt werden. Dieser private Schlüssel darf nur dem einem Teilnehmer bekannt sein. Dieser Teilnehmer darf den privaten Schlüssel niemals preisgeben. Ist dies doch passiert, ist die gesamte Kommunikation von diesem Teilnehmer nicht mehr gesichert.

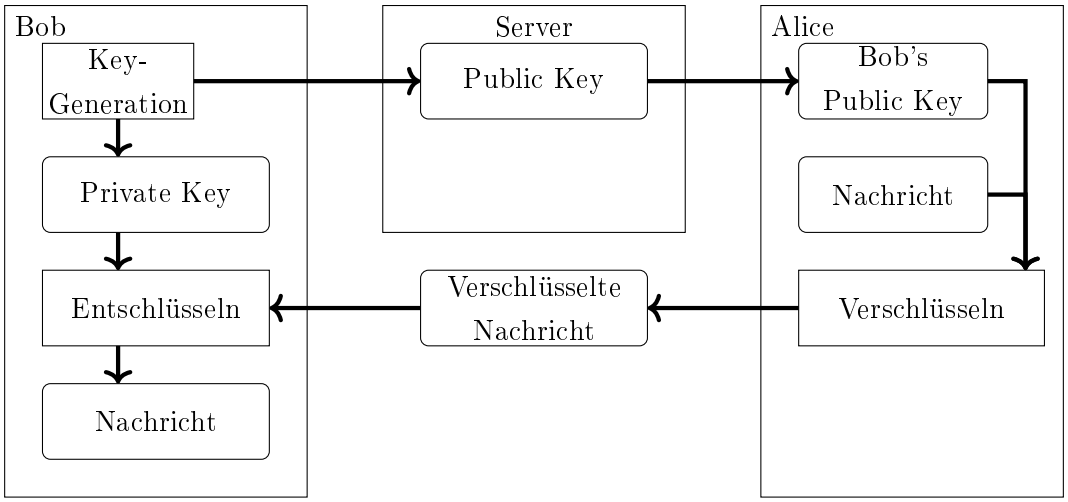


Abbildung 2.3: Schema: RSA für die Verschlüsselung

Die in Abbildung 2.3 genannten Namen entsprechen denen in der Kryptographie üblich verwendeten Namen. *Bob* steht für den Empfänger der Nachricht, *Alice* ist die Senderin.

Um die Kommunikation zu sichern, müssen zunächst zwei große[5] Primzahlen p und q gewählt werden. Diese beiden Primzahlen sind die privaten Parameter und müssen geheim

[5] Größenordnung >1024 bit

gehalten werden.
Die Multiplikation der geheimen Parameter ergibt den ersten Teil des öffentlichen Schlüssels N:

$$N = p \cdot q. \tag{2.1}$$

Obwohl N öffentlich gemacht wird, sind die privaten Parameter geheim. Das liegt daran, dass es mathematisch sehr aufwendig ist N zu faktorisieren, solange die beiden Faktoren p und q groß genug sind. Es gibt bis heute keinen effizienten Algorithmus für dieses Problem.
Zur Bestimmung des zweiten Teils des öffentlichen Schlüssels wird die eulersche ϕ-Funktion von N benötigt:

$$\phi(N) = (p-1) \cdot (q-1) \tag{2.2}$$

Mittels $\phi(N)$ wird e, der zweiter Teil des öffentlichen Schlüssels, gewählt. Dabei muss e eine teilerfremde Zahl zu $\phi(N)$ sein. Zur Prüfung, ob zwei Zahlen teilerfremd sind, wird der größter gemeinsamer Teiler (GGT) verwendet:

$$ggT(e, \phi(N)) = 1 \tag{2.3}$$

Der private Schlüssel d entspricht dem multiplikativen Inversen von e modulo $\phi(N)$. Daraus ergibt sich:

$$e \cdot d \equiv 1 \pmod{(\phi(N))} \tag{2.4}$$

Nach dieser Berechnung sind alle Schlüsselteile bekannt:

- öffentlicher Schlüssel: e, N
- privater Schlüssel: d
- geheime Parameter: p, q.

Verschlüsselung

Das Prinzip der Verschlüsselung basiert darauf, dass es eine Funktion gibt, die in eine Richtung („Verschlüsseln“) einfach zu berechnen ist. Jedoch ist die andere Richtung („Ent-

schlüsseln“) ohne bestimmtes Wissen („privater Schlüssel“) mathematisch sehr aufwendig und dadurch nicht praktikabel. Diese Funktion $E(x)$ ermöglicht es die Nachricht M in den Geheimtext C umzuwandeln. Hierzu wendet sie den öffentlichen Schlüssel auf die Nachricht an:

$$E(M) = C \tag{2.5}$$

Wendet man die Gleichung 2.5 auf den RSA an, ergibt sich

$$M^e \bmod N = C. \tag{2.6}$$

Entschlüsselung

Die Entschlüsselung ($D(x)$) funktioniert mittels des privaten Schlüssels des Empfängers. Dieser wird auf den Geheimtext C angewendet und liefert die Nachricht M:

$$D(C) = M \tag{2.7}$$

Konkret bedeutet das für den RSA:

$$C^d \bmod N = M. \tag{2.8}$$

Zusammenfassung

Zum Verschlüsseln wird die Nachricht mit dem öffentlichen Schlüssel des Empfängers verschlüsselt ($E(M)$). Damit der Empfänger die Nachricht lesen kann, muss dieser seinen privaten Schlüssel auf den Geheimtext anwenden ($D(E(M))$). Für eine verschlüsselte Nachricht zwischen zwei Personen gilt folgende Gleichung:

$$M = D(E(M)) \tag{2.9}$$

Durch das Einsetzen von Gleichung 2.6 und Gleichung 2.8 in Gleichung 2.9 erhält man:

$$\begin{aligned} M &= (M^e)^d \bmod N \\ M &= M^{ed} \bmod N. \end{aligned} \tag{2.10}$$

Aus Gleichung 2.4 ist bekannt, dass $e \cdot d \equiv 1 \pmod{\phi(N)}$ ist. Daraus ergibt sich:

$$\begin{aligned} M &= M^1 \bmod N \\ \Rightarrow M &= M \end{aligned} \tag{2.11}$$

Die Gleichung 2.11 gilt nur für $M < N$.

2.3.2 Signatur

Eine verschlüsselte Kommunikation ist nicht zwangsläufig sicher. So kann es z.B. sein, dass sich ein Teilnehmer der Kommunikation als ein anderer ausgibt (Authentifizierung) oder eine dritte Person die Nachricht modifiziert (Integrität). Dies passiert bspw. wenn ein Angreifer aktiv in die Kommunikation eingreift.
Für das Schutzziel der Authentizität bedeutet das, dass der Angreifer aktiv mit dem Namen einer anderen Person kommunizieren kann. Die Echtheit des Namen kann in einer verschlüsselten Kommunikation ohne Signatur vom Kommunikationspartner nicht überprüft werden.
Bei der Integrität der Nachricht besteht dasselbe Problem. Der aktive Angreifer kann eine Nachricht, innerhalb einer verschlüsselten Kommunikation, abfangen und verändern ohne das der Empfänger der Nachricht überprüfen kann, ob die Nachricht verändert wurde.

Um diese Probleme zu lösen gibt es die Signatur. Die Signatur macht es möglich den Absender einer Nachricht zu verifizieren und gleichzeitig die Integrität der Nachricht zu bestätigen. Hierfür wird die zusendende Nachricht mit dem private Key d des Senders verschlüsselt. Diese verschlüsselte Nachricht (Signatur) wird mit der Nachricht an den Empfänger übertragen. Da die Signatur aus der Nachricht generiert wird, ist es nahezu unmöglich die Nachricht abzuändern, ohne das die angehängte Signatur ungültig wird. Außerdem ist eine dritte Person nicht in der Lage eine Signatur einer anderen Person zu fälschen, solange der private Schlüssel nicht komprimiert wurde.

Für die Signatur S werden zunächst die aus Verschlüsselung und Entschlüsselung bekannten Parameter benötigt (öffentlicher Schlüsse, privater Schlüssel usw.).
Die Signatur setzt sich aus der Nachricht M und dem privaten Schlüssel d des Senders zusammen. Zum Signieren wird die Entschlüsselungsfunktion $D(x)$ auf die Nachricht M

angewendet:

$$S = D(M)$$
$$S = M^d \bmod N. \tag{2.12}$$

Da nur der Sender der Nachricht seinen privaten Schlüssel haben sollte, ist sichergestellt, dass die Nachricht auch wirklich von ihm stammt. Auch die Überprüfung der Signatur für den Empfänger ist recht einfach. Er wendet die Verschlüsselungsfunktion $E(x)$ auf die empfangene Signatur an. Stimmen dabei Signatur und Absender überein, ist die ursprüngliche Nachricht wieder sichtbar.

$$M = E(S)$$
$$M = S^e \bmod N. \tag{2.13}$$

2.3.3 Implementierung

Der RSA kann auf unterschiedliche Weisen implementiert werden. In dieser Thesis werden nur der Straightforward Method (SFM)-RSA und der CRT-RSA betrachtet (siehe [Gir06]).

Straightforward-Variante

Der SFM-RSA kann sowohl für Verschlüsselung, Entschlüsselung als auch für die Signatur und Verifikation verwendet werden. In dieser Thesis wird die „Square and Multiply"-Implementierung betrachtet. Hierbei wird der Exponent vom Most Significant Bit (MSB) zum Least Significant Bit (LSB) durchlaufen.

Algorithmus 1 : Square and Multiply (Binary exponentiation: Left-to-right Variante) [Sma16]

Input : Exponent e, Nachricht M, Anzahl an Bits im Exponenten t, Modulo N
Output : Verschlüsselte Nachricht C

```
1 C ← 1
2 for i ← t to 0 do
3 |   C ← (C · C) mod N
4 |   if e_i = 1 then
5 |   |   C ← (C · M) mod N
6 |   end
7 end
```

CRT

Der CRT hat gegenüber dem SFM-RSA den Vorteil, dass er weniger zeitintensiv ist. Dies ist möglich, indem er den Exponenten d für die Berechnung durch zwei Zahlen (d_p, d_q) mit der halben Bitlänge ersetzt:

$$d_p = d \bmod (p-1) \tag{2.14}$$

$$d_q = d \bmod (q-1). \tag{2.15}$$

Da die beiden Zahlen d_p und d_q von dem privaten Schlüssel d und den geheimen Parametern p, q abhängen, müssen diese auch bekannt sein. Dadurch ist der CRT auf die Entschlüsselung und Signatur von Nachrichten beschränkt. Außerdem werden die beiden multiplikativen inversen Elemente (u, v) von p mod q und q mod p benötigt. Diese lassen sich durch das Lösen der Gleichung:

$$1 = u \cdot p + v \cdot q \tag{2.16}$$

bestimmen. Die Bestimmung der Lösung ist mittels erweiterten euklidischen Algorithmus möglich.

Algorithmus 2 : CRT-Implementierung einer Signatur

Input : Nachricht M, privater Schlüssel d, geheime Parameter p, q

Output : Signatur S

1 $d_p \leftarrow d \bmod (p-1)$

2 $d_q \leftarrow d \bmod (q-1)$

3 $(u, v) \leftarrow 1 = u \cdot p + v \cdot q$

4 $sig_p \leftarrow M^{d_p} \bmod p$

5 $sig_q \leftarrow M^{d_q} \bmod q$

6 $S \leftarrow u \cdot p \cdot sig_q + v \cdot q \cdot sig_p \bmod N$

Die ersten drei Zeilen in Algorithmus 2 können aus der Funktion ausgelagert werden, da sie, sofern sich der private Schlüssel nicht ändert, immer gleich sind und somit zur Schlüsselgenerierung gehören. Die Zeilen 4 und 5 können durch den SFM-RSA berechnet werden.

2.4 Angriffe auf Kryptosysteme

2.4.1 Klassifizierung von Angriffen

Es gibt viele Arten und Möglichkeiten, wie ein Angriff auf ein Kryptosystem durchgeführt werden kann. Um die Angriffe besser unterscheiden zu können, gibt es verschiedene Klassen. Diese Einstufungsmöglichkeiten sind nachfolgend erklärt. Angriffe auf Kryptosysteme lassen sich nach Oliver Kömmerling und Markus G. Kuhn [KK99] in vier Arten klassifizieren:

- **Microprobing-Angriffe** können direkt auf den Mikrocontroller/Chip zugreifen und somit den integrierten Schaltkreis beobachten, manipulieren oder stören.
- **Software-Angriffe** greifen die Implementierung auf dem Mikrocontroller an, wie z. B. das Datenübertragungsprotokoll.
- **Abhörangriffe** sind Angriffe, bei denen die Kommunikation vom/mit dem Mikrocontroller abgehört wird. Hierzu gehören bspw. auch die elektromagnetische Abstrahlung des Mikrocontrollers während der Abarbeitung seiner Aufgaben.

- **Fehlergenerierungsangriffe** nutzen eine abnormale Betriebsbedingung des Mikrocontroller aus. Hierzu gehören zum Beispiel das Erhöhen/Verringern der Taktgeschwindigkeit.

Diese Klassifizierung wurde von Sergei P. Skorobogatov [Sko05] um eine weitere Klasse erweitert:

- **Reverse Engineering** wird verwendet, um die innere Struktur eines integrierten Schaltkreises/Mikrocontrollers auszuwerten, die Funktionsweise zu verstehen und die damit verbundenen Schwachstellen zu identifizieren.

Neben den bereits genannten Klassen, um einen Angriff auf ein Kryptosystem zu klassifizieren, ist ebenfalls die Einordnung in zwei Gruppen üblich [Sko05]:

- **Invasive Angriffe** sind Angriffe, bei welchen der Chip bspw. Schicht für Schicht abgetragen wird und die für den Angreifer relevanten Layer/Schichten im Chip freigelegt werden („Reverse Engineering“). Auch das „Microprobing“ kann zu den invasiven Angriffen gehören, wenn durch dieses der Chip nach dem Angriff nicht mehr brauchbar ist. Ein invasiver Angriff ist meist mit viel Arbeitsaufwand verbunden und benötigt spezielles Werkzeug, da z. B. das Abtragen von Schichten eines Chips nicht ohne Weiteres möglich ist.

- **Nicht-invasive Angriffe** charakterisieren sich dadurch, dass von außen kaum oder gar nicht in den Chip eingegriffen wird. Unter diese Angriffen fallen die anderen drei genannten Klassen von Angriffen („Software-Angriffe“, „Abhörangriffe“ und „Fehlergenerierungsangriffe“). Es wird nicht zwangsläufig spezielles Equipment benötigt. Außerdem ist der Chip in den meisten Fällen nach dem Angriff noch nutzbar.

Es ist zu erkennen, dass sich die beiden Gruppen dahingehend unterscheiden, wie stark sie das System zerstören und welcher Arbeitsaufwand benötigt wird. Aber auch die Reproduzierbarkeit wird in diesen beiden Gruppen unterschieden. Während bei einem invasiven Angriff die Reproduzierbarkeit aufgrund der Zerstörung meist mit einem gleich großen Arbeitsaufwand verbunden ist, wird bei einem nicht-invasiven Angriffen nur eine erfolgreiche Durchführung des Angriffs benötigt. Anschließend kann dieser Angriff beliebig oft an diesem Kryptosystem wiederholt werden.
Ein Problem bei der Unterscheidung von Angriffen in invasiv oder nicht-invasiv ist, dass z. B. bei einem Fehlergenerierungsangriff durchaus Labor-Equipment benötigt wird (Zugriff auf den richtigen Bereich im Chip) und somit diese Angriffe nicht eindeutig einer

Klasse zugeordnet werden können. Aus diesem Grund wurde eine weitere Gruppen eingeführt:

- **Halb-invasive Angriffe [Sko05]** sind Angriffe, welche nicht klar einer der beiden genannten Gruppen („invasive Angriffe“ und „Nicht-invasive Angriffe“) zugeordnet werden können. Das ist der Fall, wenn der Angriff einerseits über einen hohen Arbeitsaufwand verfügt, welcher für einen invasiven Angriff typisch ist. Anderseits ist er jedoch genauso einfach reproduzierbar, wie ein nicht-invasiver Angriff. Meist geht ein „halb-invasiver Angriff“ mit dem Entkapseln des Chip einher.

2.4.2 Seitenkanalangriffe

Die Seitenkanalangriffe gehören zu der Gruppe der „nicht-invasiven“ Angriffe. Sie greifen ihre Informationen durch Größen wie die Laufzeit, den Stromkonsum oder die elektromagnetische Abstrahlung ab. Die durch einen Seitenkanalangriff gewonnen Informationen können für weitere Angriffe genutzt werden, da sie z. B. Aufschluss darüber geben, wann eine bestimmte, stromintensive Berechnung im Programm ausgeführt wird. Es gibt verschiedene Arten des Seitenkanalangriffs.

Eine ist die **Simple Power Analysis (SPA)**. Bei einer SPA wird der Stromverbrauch des Mikrocontrollers beobachtet. Dieser variiert je nach ausgeführter Instruktion. Somit können bestimmte Muster eines Kryptoalgorithmus im Stromprofil gefunden werden. Diese Muster ermöglichen es den privaten Schüssel aus dem Stromprofil zu extrahieren. Es lassen sich bspw. bei dem SFM-RSA Muster für die verschiedenen Bit-Werte erkennen (siehe Abbildung 2.4). Bei dieser Implementierungsvariante wird immer das Quadrieren ausgeführt. Die anschließende Multiplikation wird nur dann ausgeführt, wenn das entsprechende Bit im Exponenten (Schlüssel) 1 ist. Das bedeutet, dass nur bei einer 1 an der entsprechenden Stelle im Exponenten weiterer Strom benötigt wird. Auch dauert der Rechenschritt bei einer 1 länger, da sowohl das Quadrieren als auch die Multiplikation durchgeführt werden müssen. Dies lässt sich im Stromprofil wiederfinden [KJJ99]. Die Implementierungsmöglichkeiten des RSA sind im Unterabschnitt 2.3.3 näher erklärt.

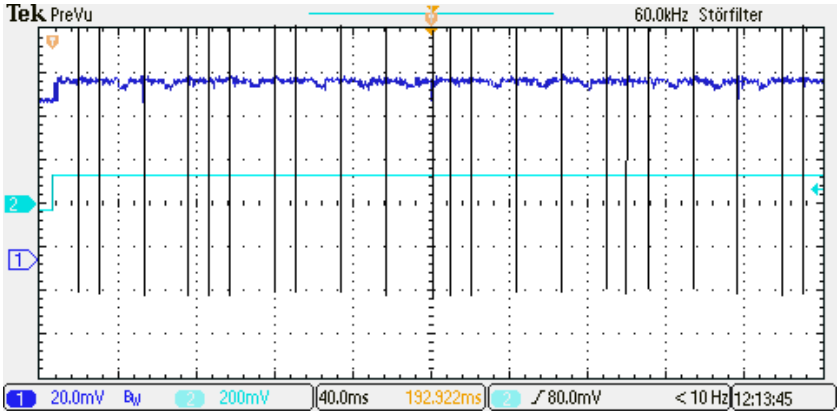


Abbildung 2.4: SPA für RSA mit der Schlüssellänge 1024 und der Nachricht "Hallo“ [blau=Stromprofil, hellblau=Triggersignal]

Eine andere Möglichkeit ist die **Differential Power Analysis (DPA)**. Diese ist in [KJJ99] beschrieben.

2.4.3 Fehlerinjektion

Bei einer Fehlerinjektion wird ein Fehler von außen in das Kryptosystem injiziert. Die Ziele einer Fehlerinjektion können z. B. bestimmte Register in der Central Processing Unit (CPU) oder Speicherbereiche sein. Hierbei lässt sich zwischen einer **Hardwarefehlerinjektion mit Kontakt**, einer **Hardwarefehlerinjektion ohne Kontakt** und einer **Softwarefehlerinjektion** unterscheiden [HTI97].

Die **Hardwarefehlerinjektion mit Kontakt** ist eine Injektion, die physischen Kontakt mit dem Zielsystem hat. Hierzu gehören z. B. der Power-Glitch oder der Clock-Glitch. Diese Art der Fehlerinjektion werden in dem Kapitel Glitch-Angriff näher erklärt.

Die **Hardwarefehlerinjektion ohne Kontakt** beschreibt Fehler, die ohne direkten Kontakt mit dem Zielsystem hergeführt werden. Sie können z. B. durch Größen wie die Temperatur (Temperatur-Angriff), elektromagnetische Felder (EMF-Angriff) oder durch Licht (Optischer Angriff) realisiert werden.

Die **Softwarefehlerinjektion** beschreibt den Vorgang, wenn in einem Programm ein Schadcode injiziert wird. Dieser führt dann zu Fehlern im eigentlichen Zielprogramm.

Beispiele für die verschiedenen Fehlerinjektionen können in [BBKN12] nachgelesen werden.

2.4.4 Glitch-Angriff

Der Glitch-Angriff gehört im Allgemeinen zur Gruppe der „halb-invasiven Angriffe“, kann aber auch den „nicht-invasiven Angriffe“ zugeordnet werden. Dies wird z. B. daran unterschieden, ob der Chip für den Angriff entkapselt werden muss oder nicht. Unterschieden wird bei Glitch-Angriffen zwischen drei Arten [KK99][BECN+06]:

1. Elektromagnetisches Feld (EMF)-Glitch
2. Clock-Glitch
3. Power-Glitch.

Ein Glitch-Angriff nutzt die Schwachstellen in der Mikrocontroller-Architektur oder der Software aus. Hierbei wird versucht, mittels einer Störung, einen oder mehreren Flipflops des Mikrocontroller in einen falschen Zustand zu bringen. Dies ist beispielsweise möglich, wenn der Glitch dazu führt, dass der Mikrocontroller eine Anweisung im Programm überspringt oder durch eine andere ersetzt. Eine weitere Möglichkeit besteht darin, die Daten auf dem Weg vom Register zum Speicher zu stören. In beiden Fällen kommt es zu einem fehlerhaften Ergebnis/Zustand im Speicher.

Clock-Glitch

Ein Clock-Glitch variiert die Taktgeschwindigkeit des Mikrocontrollers. Er erhöht diese für eine kurze Zeit von einem oder mehreren halben Takten. Da der Takt erhöht ist, tastet der D-FF das Eingangssignal „zu früh“ ab und es liegt noch kein gültiges Signal am Eingang an (vgl. Abbildung 2.5). Dieser nicht gültige Zustand am Informationseingang des D-FF hat zur Folge, dass die Instruktion nicht richtig oder gar nicht ausgeführt wird. Nach dem Glitch arbeitet der Prozessor wieder mit seiner normalen Taktgeschwindigkeit weiter. Dieser Angriff kann mit einer zuvor durchgeführten SPA kombiniert werden und an sicherheitsrelevanten Stellen zu Fehlern in der Berechnung oder zum Überspringen von wichtigen Funktionen führen.

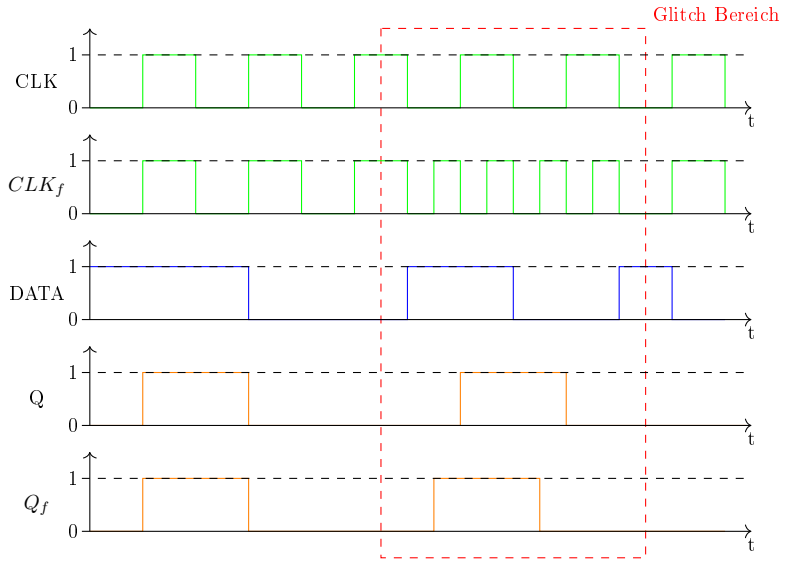


Abbildung 2.5: Impulsdiagramm: Auswirkung eines Clock-Glitches auf ein D-FF

In der Abbildung 2.5 sind ein normales Clock-Signal (CLK), ein fehlerhaftes Clock-Signal (CLK_f), das Datensignal (DATA), der Flipflopausgang mit dem normalen Clock-Signales (Q) und der Ausgang mit dem fehlerhaften Clock-Signales (Q_f) dargestellt. Beim Vergleichen der beiden Ausgänge (Q und Q_f) fällt auf, dass sich die Signale im Glitch Bereich unterscheiden.

Power-Glitch

Der Power-Glitch, auch Voltage Fault Injection (V-FI) genannt, greift die Versorgungsspannung des Prozessors an. Hierbei kann die Spannung kurzzeitig erhöht oder verringert werden. Die Variation in der Spannungsversorgung hat die Auswirkung, dass bestimmte Instruktionen nicht mehr korrekt oder gar nicht mehr ausgeführt werden. Des Weiteren ist eine konstante, stabile Spannungsversorgung für den Speicher wichtig. Wird dieser nicht konstant mit Spannung versorgt, kann dies dazu führen, dass er unter Umständen nicht mehr richtig arbeitet und es zu Fehlern beim Lesen oder Schreiben im Speicher kommt. Die genauen Auswirkungen von positiven bzw. negativen Power-Glitchen können in [ZDCR14] nachgelesen werden. Eine Möglichkeit einen Power-Glitch durchzuführen ist, die Core-Spannungsversorgung (VDDC) für einen kurzen Zeitraum komplett zu unterbrechen [BFP19].

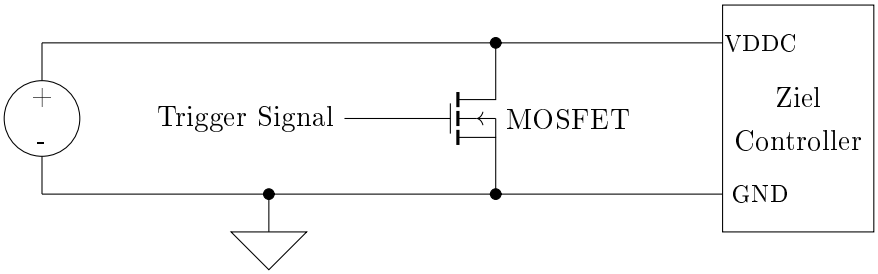


Abbildung 2.6: Beispielaufbau Power-Glitch mit MOSFET (modifiziert [BFP19])

In der Abbildung 2.6 ist zu sehen, dass die Versorgungsspannung des Prozessorkerns (VDDC) für die Dauer des Triggersignals durch den Metall-Oxid-Halbleiter-Feldeffekttransistor (MOSFET) auf Masse gezogen wird. Dies ermöglicht es, den Prozessorkern für eine kurze Zeitspanne nicht oder nur mit einer zu geringen Spannung zu versorgen.

Bei einem Power-Glitch sind die Dauer und der Zeitpunkt des Glitches von enormer Wichtigkeit. Ist die Dauer zu lang kann es dazu kommen, dass der Prozessorkern nicht mehr arbeitet oder sich selbst neu startet. Ist die Dauer zu kurz kann es passieren, dass der Prozessorkern nicht auf den Glitch reagiert und normal weiterarbeitet.

EMF-Glitch

Ein EMF-Glitch benutzt ein elektromagnetische Feld, um eine Störung im Chip zu erzeugen. Bei einem solchen Glitch wird z. B. die Schwellspannung eines Transistors, durch ein erzeugtes elektromagnetisches Feld, kurzzeitig überschritten und dieser zum „Schalten“ gebracht [KK99]. Ein möglicher Aufbau und die Analyse eines solchen Angriffes sind in [SSAQ02] beschrieben.

2.4.5 Auswirkungen des Glitch-Angriffs

Das Ziel von Glitch-Angriffen ist, sicherheitsrelevante Instruktionen in der Software zu umgehen. Hierfür können z. B. fehlerhafte Werte für die Instruktion dienen (z. B. Clock-Glitch). Die nicht korrekten Werte können bei einer möglichen if-Abfrage dazu führen, dass diese fehlerhaft ausfällt. Dasselbe Verhalten gilt auch für einen Power-Glitch, da bei diesem der Prozessor unterversorgt wird und nicht korrekt arbeitet. Das kann zu fehlerhaften if-Abfragen und/oder Instruktionen führen (siehe Abbildung 2.7).

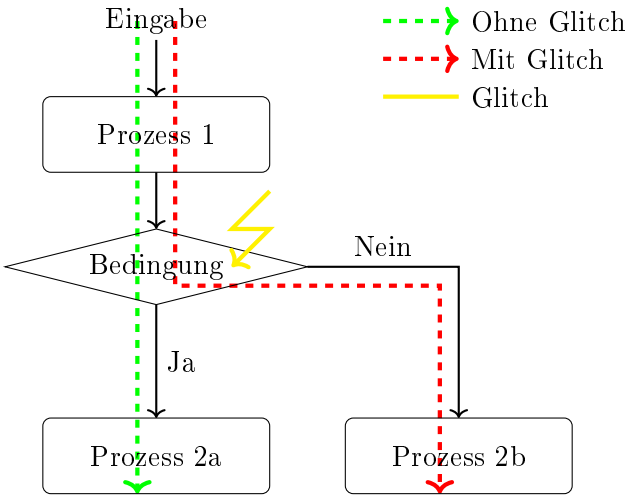


Abbildung 2.7: Auswirkung eines Glitches auf eine if-Abfrage

Eine solche fehlerhafte Bedingung kann Fehler in der Berechnung verursachen. Dies ist die Grundlage für den Bellcore-Angriff (siehe Abschnitt 2.5). Um die Auswirkungen besser nachvollziehen zu können, folgt ein Beispiel einer Add-Instruktion mit und ohne Glitch.

Tabelle 2.2: Assemblerbeispiel: ADD-Befehl ohne Glitch

Nummer	Instruktion	ax[Binär]	bx[Binär]	cx[Binär]
1.	mov ax, 10	1010	-	-
2.	mov bx, 45	1010	101101	-
3.	add ax,bx	110111	101101	-
4.	mov cx, ax	110111	101101	110111

In Tabelle 2.2 sind die Assembler-Instruktionen für einen ADD-Befehl ohne Glitch zu sehen. Es ist zu erkennen, dass die beiden Zahlen *ax* und *bx* im 3. Schritt addiert und im 4. Schritt in *cx* geschrieben werden. Wenn während des 3. Schrittes ein Glitch injiziert wird (rot hinterlegte Instruktionen), sieht der ADD-Befehl aus wie in Tabelle 2.3. Hierbei wird angenommen, dass der Glitch das Überspringen der eigentlichen ADD-Instruktion bewirkt (siehe. Schritt 3 und 4).

Tabelle 2.3: Assemblerbeispiel: ADD-Befehl mit Glitch

Nummer	Instruktion	ax[Binär]	bx[Binär]	cx[Binär]
1.	mov ax, 10	1010	-	-
2.	mov bx, 45	1010	101101	-
3.	add ax, bx	1010	101101	-
4.	mov cx, ax	1010	101101	1010

2.5 Bellcore

Der Bellcore-Angriff greift die CRT-Implementierung des RSA an. Die Grundlagen für diesen Angriff wurden in [BDL96] und [BDL01] veröffentlicht. [BDL01] zeigt auf, dass man mit zwei Signaturen (einer richtigen Signatur (S) und einer fehlerhaften Signatur ($\hat{S}$)) einen der beiden geheimen Parameter bestimmen kann. Anschließend ist es durch eine einfache Division von N und dem einem geheimen Parameter möglich den anderen geheimen Parameter zu bestimmen.

Sei S eine gültige, durch den CRT-RSA berechnete Signatur für die Nachricht M. Wie bereits bekannt, besteht S aus zwei Teilen, im folgenden als S_1 und S_2 bezeichnet:

$$S = S_1 + S_2 \tag{2.17}$$

Des Weiteren wird eine fehlerhafte Signatur für dieselbe Nachricht M benötigt. Dies ist der Fall, wenn während der Berechnung von S_1 und/oder S_2 ein Fehler passiert:

$$\hat{S} = \hat{S_1} + \hat{S_2}. \tag{2.18}$$

Vorausgesetzt der Fehler passiert nur während einer der beiden Teilberechnung $\hat{S_1}$ oder $\hat{S_2}$, so gilt für $\hat{S}$:

$$\hat{S_q} = S_1 + \hat{S_2} \tag{2.19}$$

oder

$$\hat{S_p} = \hat{S_1} + S_2. \tag{2.20}$$

Tritt in einer der beiden Fälle (Gleichung 2.19 oder Gleichung 2.20) ein, lässt sich einer der beiden geheimen Parameter durch den GGT von N und $S - \hat{S}$ bestimmen (siehe Gleichung 2.21).

$$q = ggT(S - \hat{S_q}, N)$$
$$\text{oder}$$
$$p = ggT(S - \hat{S_p}, N) \tag{2.21}$$

Durch $\frac{N}{p} = q$ oder $\frac{N}{q} = p$ lässt sich der andere geheime Parameter berechnen. Die in Gleichung 2.21 gezeigte Berechnung ist möglich, da gilt das

$$S = u \cdot p \cdot sig_q + v \cdot q \cdot sig_p \bmod N$$

ist. Die Berechnung $S - \hat{S}$ ergibt für den Fall, dass die Berechnung von sig_q bei der fehlerhaften Signatur gestört wurde:

$$S - \hat{S} = (u \cdot p \cdot sig_q + v \cdot q \cdot sig_p \bmod N) - (u \cdot p \cdot \hat{sig_q} + v \cdot q \cdot sig_p \bmod N) \tag{2.22}$$

Durch das Umsortieren der Terme ergibt sich:

$$S - \hat{S} = ((u \cdot p \cdot sig_q) - (u \cdot p \cdot \hat{sig_q}) + (v \cdot q \cdot sig_p) - (v \cdot q \cdot sig_p)) \bmod N$$

Nach dem Kürzen sieht die Gleichung folgendermaßen aus:

$$\begin{aligned} S - \hat{S} &= ((u \cdot p \cdot sig_q) - (u \cdot p \cdot \hat{sig_q})) \bmod N \\ &= (u \cdot p \cdot (sig_q - \hat{sig_q})) \bmod N \end{aligned}$$

In der Gleichung 2.23 befindet sich der geheime Parameter p. Dieser ist Teil der Zahl N. Da $N = p \cdot q$ gilt, ergibt sich:

$$ggT((u \cdot p \cdot (sig_q - \hat{sig_q})) \bmod N, N) = p \tag{2.23}$$

Die Auswirkungen eines Bellcore-Angriffs auf RSA sind in [BDL01] zusammengefasst. In [Len96] wird angemerkt, dass für einen Bellcore-Angriff eine fehlerhafte Signatur ausreicht. Vorausgesetzt wird wieder, dass man die Nachricht M kennt. Da $M = S^e \bmod N$

gilt, lässt sich $M \bmod \mathrm{N} = S^e$ auf Gleichung 2.21 anwenden:

$$q = ggT((S^e \bmod N) - (\hat{S}^e \bmod N), N) \tag{2.24}$$

$$=> q = ggT(M - \hat{S}^e, N) \tag{2.25}$$

Dies gilt analog für den geheimen Parameter p.

3 Setup

3.1 Tiva TM4C1294NCPDT

Zur Generierung des Triggersignales wird ein Board aus der Tiva C-Serie verwendet. Das TM4C1294NCPDT [Ins] verfügt über einen ARM Cortex-M4 Chip. Des Weiteren ist das Tiva Board als Sytem-on-a-Chip (SOC) realisiert. Dadurch können viele verschiedene Anwendungen mit einem Chip umzusetzen werden. Das Triggersignal wird durch einen der acht General-Purpose Timer (GPTM) umgesetzt. Da für den Glitch ein Signal im ns-Bereich benötigt wird, muss der Mikrocontroller in der Lage sein, eine ausreichend hohe Taktfrequenz zur Verfügung zu stellen. Mit maximal 120 MHz als interne Taktquelle, liegt der TM4C1294NCPDT bei ungefähr 8,33 ns pro Taktzyklus:

$$t = \frac{1}{\text{max. Taktfrequenz}} = \frac{1}{120\,\text{MHz}} \approx \underline{\underline{8{,}33 \cdot 10^{-9}\,\text{s}}} \tag{3.1}$$

Das TM4C1294NCPDT-Board ist mit sieben Kommunikationsschnittstellen ausgestattet, darunter sind z. B. Ethernet, Universal Serial Bus (USB) und auch acht Universal Asynchronous Receiver Transmitter (UART)-Schnittstellen. Außerdem besitzt das Board 15 physikalische General Purpose Input/Output (GPIO)-Blöcke, welche entweder als Ein- oder Ausgang parametriert werden können oder Spezialfunktion übernehmen. Die möglichen Spezialfunktion sind in [Ins] aufgeführt.

Die für den Angriff relevanten Daten des TM4C1294NCPDT sind in Tabelle 3.1 aufgeführt.

Tabelle 3.1: Übersicht über die relevanten Daten des TM4C1294NCPDT [Ins]

Funktion	Beschreibung
Taktgeschwindigkeit	max. 120 MHz
Flash Speicher	1024 kB
UART	8 UART-Module
GPTM	8 16/32-bit GPTM-Blöcke
GPIO	15 physikalische GPIO Blöcke

3.1.1 Timer

Das TM4C1294NCPDT Board verfügt über acht konfigurierbare GPTM. Die acht GPTM können in verschiedenen Zählmodi verwendet werden. Für diese Thesis wird nur der „Counting-Down"-Modus und der „One-Shot-Modus" verwendet. Diese beiden Modi funktionieren so, dass der Timer von einer bestimmten Zahl runterzählt und nach dem Erreichen der unteren Grenze nicht automatisch erneut startet. Außerdem setzt der Timer nach dem Erreichen der unteren Grenze im Flag Register das Flag „TATORIS". Dieses Flag kann im Programm abgefragt und weiterverwendet werden (vgl. [Ins]).

Umrechnung Zeiteinheit in Systemtakte

Für die Umrechnung der Zeiteinheit in die Anzahl der benötigten Systemtakte ist folgende Gleichung zuständig:

$$\begin{aligned} \text{Anzahl an Takten} &= \text{gewünschte Zeit} \cdot \text{Zeit für einen Takt} \\ &= \text{gewünschte Zeit} \cdot \frac{1}{\text{Systemtaktfrequenz}} \end{aligned} \tag{3.2}$$

3.2 Infineon Board (XMC 4700 Relax Kit Lite)

Das Infineon Board „XMC 4700 Relax Kit Lite“ wird als sogenanntes Device Under Test (DUT) verwendet. Der verwendete Chip XMC 4700-F144K2048 gehört zu der Familie der XMC 4000 Reihe von Infineon. Diese Boards verfügen über eine 32-Bit-ARM-Cortex M4 CPU, welche einen 2048 kB großen Flashspeicher und einen 352 kB Datenspeicher besitzt.

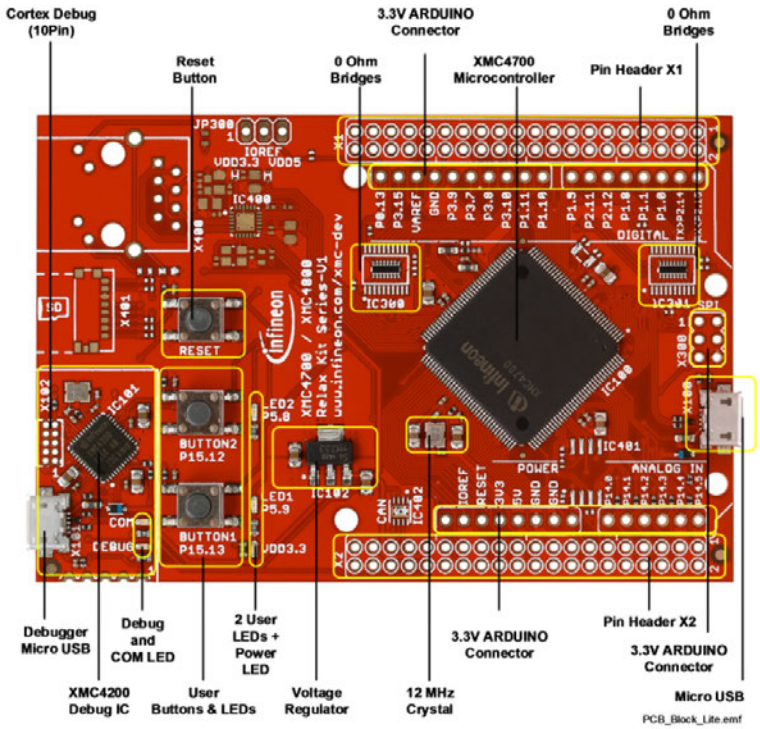


Abbildung 3.1: Übersicht: XMC 4700 Relax Kit Lite [Rel16]

Die für den Chip benötigte Spannung von min. 3,13 V bis max. 3,63 V wird mittels eines Spannungsreglers generiert. Der Spannungsregler versorgt das Board mit konstanten 3,3 V [Rel16]. Die VDDC wird innerhalb des XMC4700 Chips auf 1,3 V reduziert. Außerdem befinden sich, wie in Abbildung 3.2 zu erkennen, an den Versorgungspins jeweils 100 nF Kondensatoren gegen das Massepotenzial des Boards (VSS). Ergänzend hierzu sind an der Pad Spannungsversorgung (VDDP) und VDDC jeweils 10 µF parallel zu den 100 nF geschaltet.

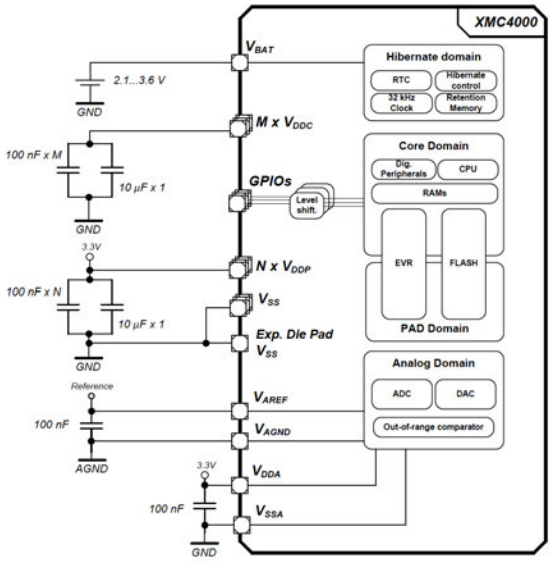


Abbildung 3.2: Spannungsversorgungsschema [XMC18]

Da die VDDC Spannung angegriffen wird, ist es notwendig, vor den Versuchen die Kondensatoren an den VDDC Pins zu entfernen (siehe Abbildung 3.3). Die gekennzeichneten Kondensatoren könnten dem injizierten Spannungsabfall entgegenwirken und somit den Angriff undurchführbar machen. An der Stelle der Kondensatoren befindet sich später im Versuch der MOSFET, welcher die intern generierte VDDC-Spannung mit dem VSS-Potenzial verbindet (vgl. Abbildung 2.6 im Grundlagenkapitel). Der MOSFET muss nach der Entfernung der Kondensatoren an einem der VDDC-Ausgängen (19, 61, 90 und 125) des Chips fest gelötet werden.

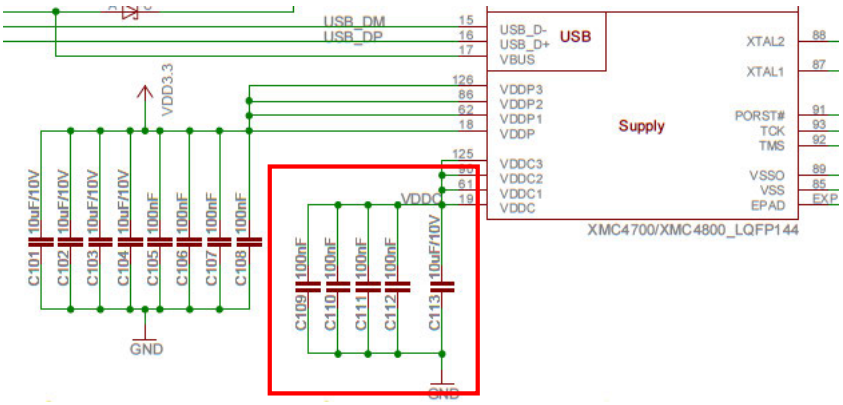


Abbildung 3.3: Schaltplan: Zu entfernende Kondensatoren (rot) [Auszug aus [Rel16]]

Des Weiteren ermöglicht der XMC 4700 Chip mittels der USB-Schnittstelle auf dem Relax Kit Lite eine UART Kommunikation [XMC18]. Auch verfügt das Board über programmierbare GPIO Pins. Diese Pins können bspw. als Trigger für das Glitchsignal benutzt werden.

3.2.1 Reset

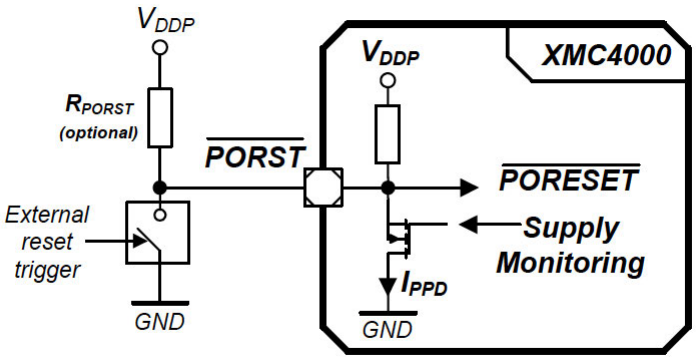


Abbildung 3.4: Schema: Reset XMC4700 [XMC18]

Die Informationen dieses Abschnittes sind sinngemäß aus [Rel16] übernommen. Neben der Möglichkeit, dass sich der Mikrocontroller selbstständig bei bestimmten Fehlern zurücksetzt (Watchdog), kann das DUT über fünf weitere Wege manuell zurückgesetzt werden (vgl. Abbildung 3.4):

1. Über den auf dem Board befindlicher Reset-Knopf
2. Über die auf dem Board befindliche Debug-Schnittstelle (IC101)
3. Über die externe Debug-Schnittstelle (X102)
4. Über die Arduino Versorgungsleiste (X302, Reset)
5. Über die Pinleiste X2 (X2.32, #RST)

Wie in Abbildung 3.4 zu sehen, besitzt der XMC4700 einen bidirektionalen Pin. An diesem Pin liegt während des normalen, nicht fehlerhaften Ablauf des Programms ein logisches High-Signal an ($\overline{PORST}$). Liegt ein Fehler vor, welcher intern detektiert werden kann (z. B. bei der Spannungsversorgung), wird das $\overline{PORST}$-Signal auf logisch Low gezogen (MOSFET im XMC4700 schaltet). Sollte es zu einem Fehler kommen, der nicht

intern erkannt wird, kann das DUT, über die fünf oben genannten Wege, extern zurückgesetzt werden (siehe Abbildung 3.4 „External reset trigger“).

Als externes Signal kann beispielsweise, wie oben erwähnt, der Reset-Knopf auf dem Board dienen (siehe Abbildung 3.5). Es ist zu erkennen, dass mittels eines Pullup-Widerstands das $RESET\#$ Signal auf dem logischen High-Wert gehalten wird. Schaltet der Reset-Knopf ergibt sich ein direkter Weg von der Versorgungsspannung VDD nach GND über $SW102$. Somit liegt das Massepotenzial an $RESET\#$ an, welches dem logischen Low-Wert entspricht.

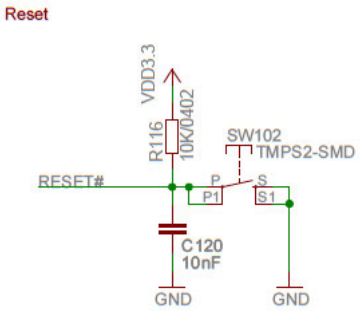


Abbildung 3.5: Reset-Signal am XMC4700 Relax Kit Lite (Reset Knopf)[Rel16]

3.2.2 Einstellbare Taktzeiten

Die integrierte Entwicklungsumgebung (IDE) von Infineon bietet die Möglichkeit die Taktfrequenz softwaretechnisch anzupassen. Das Einstellen der Taktzeit ermöglicht es zu beeinflussen, wie schnell das XMC4700 Relax Kit Lite arbeitet. Diese Arbeitsgeschwindigkeit wirkt sich direkt auf die Dauer für bestimmte Instruktionen aus, wie z. B. eine if-Abfrage. Bei einer langsamen Taktfrequenz liegt die benötigte Zeit pro Instruktion höher als bei einer schnellen Taktfrequenz. Die eingestellte Taktzeit wirkt sich direkt auf die benötigte Berechnungsdauer der RSA aus. Zum Einstellen der Taktzeit werden die in Abbildung 3.6 gezeigten „Clock divider“ verwendet.

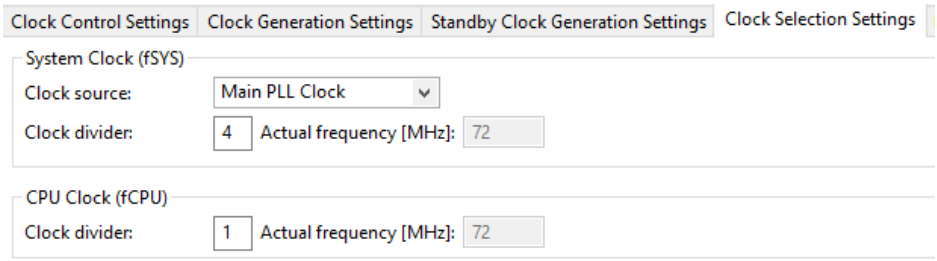


Abbildung 3.6: Ausschnitt: Einstellungsmöglichkeiten für die Taktfrequenzen (XMC4700)

In der Abbildung ist zu sehen, dass die Systemtaktfrequenz (f_{SYS}) bestimmt wird, indem die eingestellte Taktquelle (288 MHz) durch vier geteilt wird. Anschließend lässt sich aus der f_{SYS} die CPU-Taktfrequenz (f_{CPU}) berechnen. Hierzu wird die Systemtaktfrequenz durch den „Clock divider“ der CPU Clock geteilt. Der „Clock divider“ für die Systemtaktfrequenz muss zwischen 1 bis 256 liegen und der für die CPU-Taktfrequenz muss 1 oder 2 sein. Die „Clock divider“ können nur ganze Zahlen annehmen.
Aus diesen Eigenschaften ergeben sich verschiedene Taktfrequenzen (f_{SYS} und f_{CPU}). Diese sind in der nachstehenden Tabelle festgehalten:

Tabelle 3.2: XMC4700: Mögliche Taktfrequenzen für eine Taktquelle mit 288 MHz (f_{SYS} und f_{CPU})[Auszug]

Taktquelle	Systemtaktteiler	CPU-Taktteiler	f_{SYS}	f_{CPU}
288 MHz	1	1	288 MHz	288 MHz
288 MHz	1	2	288 MHz	144 MHz
288 MHz	2	1	144 MHz	144 MHz
288 MHz	2	2	144 MHz	72 MHz
288 MHz	3	1	96 MHz	96 MHz
288 MHz	3	2	96 MHz	48 MHz
288 MHz	4	1	72 MHz	72 MHz
288 MHz	4	2	72 MHz	36 MHz
288 MHz	5	1	57,6 MHz	57,6 MHz
288 MHz	5	2	57,6 MHz	28,8 MHz
...	...	...	...	....

3.3 MOSFET

Der MOSFET wird verwendet, um die kurzzeitige Reduzierung oder Unterbrechung der VDDC herzustellen. Hierzu wird dieser, wie in Abschnitt 2.4.4 gezeigt, mit der VDDC-Spannung und dem VSS-Potential verbunden. Als Trigger am Gate Eingang des MOSFET dient ein extern generiertes Signal.
Bei dem in der Thesis verwendeten MOSFET handelt es sich um den IRLM2502pbf. Aus dem Datenblatt lassen sich die folgenden Schaltzeiten entnehmen [Rec14]:

Tabelle 3.3: Schaltzeiten des IRLM2502pbf [Rec14]

Parameter	Kürzel	Typ. Wert in ns
Einschaltverzögerung	$t_{d(on)}$	7,5
Anstiegszeit	t_{rise}	10
Ausschaltverzögerung	$t_{d(off)}$	54
Abfallzeit	t_{fall}	26

Bedingungen: $V_{DD} = 10\,\text{V}$, $I_D = 1{,}0\,\text{A}$, $R_G = 6\,\Omega$ und R_D[1]$= 10\,\Omega$

Die Einschaltzeit des MOSFET setzt sich aus der Einschaltverzögerung $t_{d(on)}$ und der Anstiegszeit t_{rise} zusammen. Die Ausschaltzeit setzt sich ebenso aus zwei Komponenten zusammen ($t_{d(off)}$ und t_{fall}). Die Tabelle 3.3 liefert eine Einschaltzeit $t_{MOSFETEIN} \approx 17{,}5\,\text{ns}$ und eine Ausschaltzeit $t_{MOSFETAUS} \approx 80\,\text{ns}$.
Neben den Zeiten des MOSFET ist die Gate-Schwellspannung wichtig. Diese gibt an, ab welcher Spannung der MOSFET schaltet. Die minimale Gate-Schwellspannung (V_{GS}) beträgt 0,6 V und die maximale 1,2 V (vgl. [Rec14]).

3.4 Trigger-Signal

Das Trigger-Signal bestimmt maßgeblich die Dauer des Glitches. Der Bereich, in welchem der Glitch erfolgreich durchgeführt werden kann, befindet sich bei ca. drei Takten des DUT. Dieser Bereich wird in Unterabschnitt 4.2.1 genauer bestimmt. Somit muss der verwendete MOSFET auch in diesem Zeitbereich sicher schalten können. Aus [XMC18]

[1] Für Pulsweiten $\leq 300\,\text{ns}$; duty cycle $\leq 2\%$

lässt sich eine maximale Systemfrequenz $f_{sys} = 144\,\text{MHz}$ entnehmen. Daraus lässt sich die minimale Systemtaktzeit bestimmen:

$$\text{Takt} = \frac{1}{f_{sys}} = \frac{1}{144 \cdot 10^6 \text{Hz}} = \underline{\underline{6{,}944\,\text{ns}}} \tag{3.3}$$

Aus Gleichung 3.1 ersichtlich ist, dass das Tiva-Board über eine minimale Systemtaktzeit von 8,33 ns verfügt. Da 6,944 echt kleiner 8,33 ist, ist es bei einer eingestellten Taktfrequenz von 144 MHz des DUT nicht möglich den Glitch auf einen Takt genau zu timen. Da bei einem Glitch aber meist mehrere Instruktionen angegriffen werden, stellt dies kein großes Problem dar und kann somit in dieser Thesis vernachlässigt werden.

4 Vorbereitung

4.1 Tiva Board

4.1.1 Bestimmung der Ein- und Ausschaltzeiten eines Pins

Das Timing ist bei einem Power-Glitch von großer Bedeutung. Aus diesem Grund müssen bestimmte Faktoren, wie die Ein-, Ausschaltzeit und eine mögliche Verzögerungszeit, bestimmt und berücksichtigt werden.
Das Vorgehen zur Bestimmung der Zeiten ist in Abbildung 4.1 zusammengefasst. Zunächst wird die Taktgeschwindigkeit des Mikrocontrollers auf 120 MHz gestellt. Anschließend werden der entsprechende Port und der Pin am Board konfiguriert. Nach der Konfiguration des Tiva Boards werden 20 Mal die folgenden Programmschritte ausgeführt:

1 Setzen des Pins (A7)

2 Warten bis logischer High-Pegel anliegt

3 Rücksetzen des Pins (A7)

4 Warten bis logischer Low-Pegel anliegt.

Zur Überprüfung der Pegel wird ein weiterer Pin als Eingang konfiguriert (K2). Dieser wird im Programm solange abgefragt, bis er einer logischen 1 entspricht (Polling). Der Programmcode befindet sich im Anhang unter Unterabschnitt A.1.2.

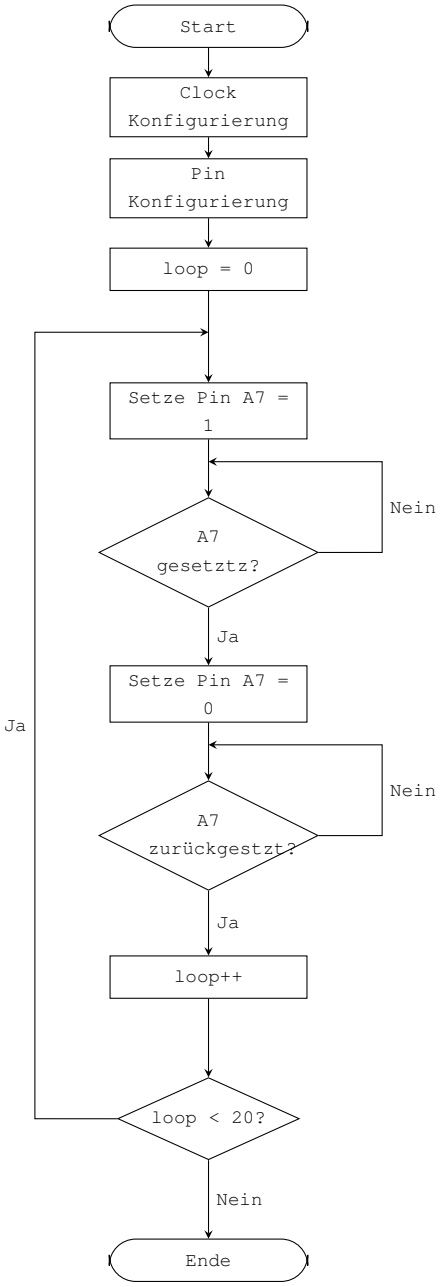


Abbildung 4.1: Programmablaufplan: Bestimmung der Ein- und Ausschaltzeit des A7 Pins (Tiva Board)

Die Einschaltzeit der Pins t_{rise} und die Ausschaltzeit des Pins t_{fall} werden mittels PicoScope bestimmt. Hierzu wird das Signal am Pin A7 oszillographiert und die Messung „Zykluszeit“ hinzugefügt. Diese zeigt die minimale und maximale Zykluszeit an und be-

rechnet den Mittelwert und die Standardabweichung der Messdaten. Aus den gemessenen Daten werden t_{rise} und t_{fall} ermittelt. Die Abbildung 4.2 zeigt, dass sich die benötigte Gesamtdauer für ein 1-Signal der Dauer t_{Ein} annähernd durch $t_{gesamt} \approx t_{rise} + t_{Ein} + t_{fall}$ berechnen lässt.

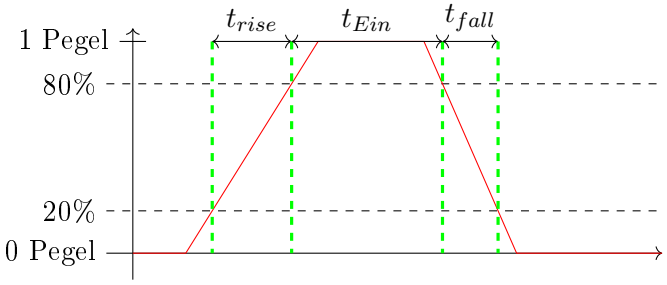


Abbildung 4.2: Beispiel: Rise Time t_{rise} und Fall Time t_{fall}

Da für den Glitch eine möglichst geringe Schaltdauer der Triggerpins benötigt wird, wird die Auswirkung verschiedener Optimierungseinstellungen der verwendeten IDE überprüft. Diese Einstellungen beschleunigen z. B. den Programmablauf indem sie überflüssige Anweisungen löschen und den Code optimal anordnen. Ein schneller Programmablauf führt zu kürzeren Schaltzeiten des Triggerpins.

Die IDE Code Composer Studio (CCS) bietet die Möglichkeit den Programmcode nach bestimmten Präferenzen zu übersetzen. Zwei dieser Einstellmöglichkeiten sind die Programmgröße und die Programmgeschwindigkeit (vgl. Abbildung 4.3).

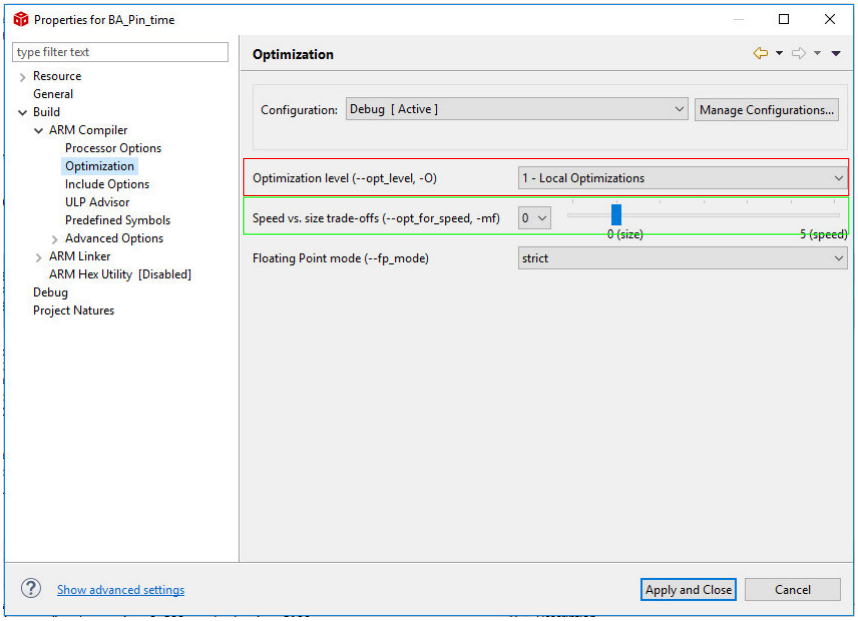


Abbildung 4.3: Übersicht: Einstellungen Optimierung

Bei der rot umrahmten Einstellung lässt sich ein Optimierungslevel (off-4) festlegen. Das Level 0 sagt aus, dass nur das Register optimiert wird, während Level 4 bedeutet, dass das ganze Programm überarbeitet und verbessert wird. Mit „Speed vs. size“ (grüner Kasten) wird eingestellt, ob bei der Optimierung mehr auf den Speicherbedarf oder auf die Programmgeschwindigkeit geachtet werden soll.

Um den für die Anwendung besten Build zu finden, liefert das CCS einen „Optimierungsassitenten“ (siehe Abbildung 4.4).

Abbildung 4.4: CCS: Beispiel Optimierungslevel vs. Code Size

Die Programmgröße spielt bei dem Glitch nur eine sekundäre Rolle. Primär ist die Programmgeschwindigkeit wichtig. Aus diesem Grund wird der Optimierungsassistent fest auf die Geschwindigkeitsoptimierung eingestellt Anschließend werden die verschiedenen Optimierungslevel ausprobiert und geprüft, welche Konfigurierung in Frage kommt. Hierzu werden der Optimierungsassistent und das PicoScope verwendet. Aus den Abbildungen in dem Unterabschnitt A.4.1 ist ersichtlich, dass drei der fünf Optimierungslevel in Frage kommen. Die Level 2, 3 und 4 liefern eine Zykluszeit von 183,3 ns. Dieses Ergebnis wird auch durch die Betrachtung eines einzelnen Ein- und Ausschaltvorgangs bestätigt (vgl. Abbildung 4.5).

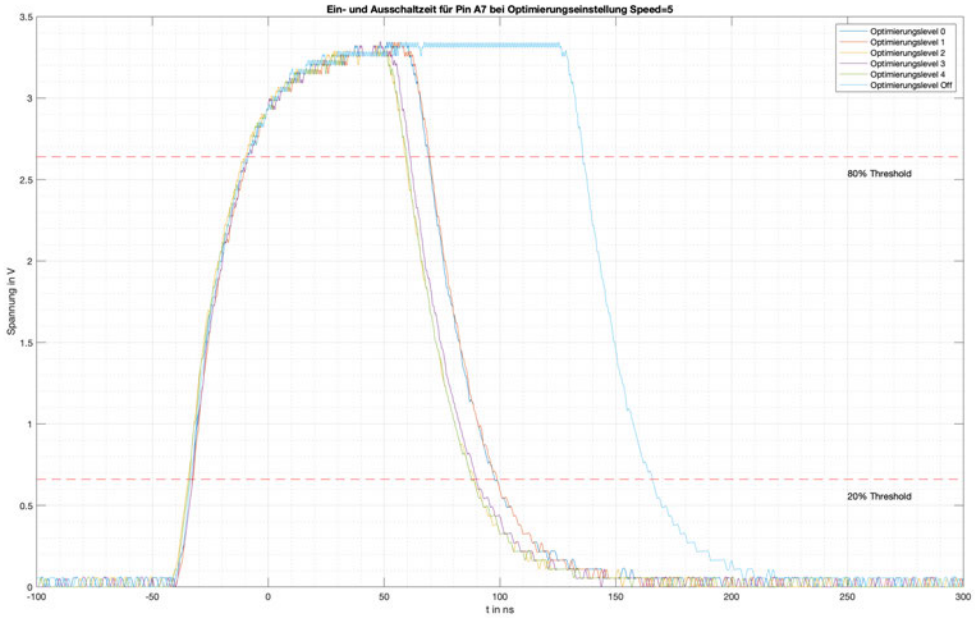


Abbildung 4.5: Ein- und Ausschaltzeit Pin A7 (Einzel)

Bei der Auswertung der Abbildung 4.5 ist zu erkennen, dass das Optimierungslevel Off (hell Blau) die längste Zeit benötigt. Das Optimierungslevel 0 (dunkel Blau) und 1 (Orange) liegen im Mittelfeld und am schnellsten sind die Optimierungslevel 2 (Gelb), 3 (Violett) und 4 (Grün). Das selbe Ergebnis liefern auch die, vom Picoscope gemessenen Zykluszeiten (siehe Unterabschnitt A.4.1). Mittels Optimierungsassistent lässt sich die beste Konfigurierung für die drei Level (2, 3 und 4) bestimmen. Da die Geschwindigkeit bei den drei in Frage kommenden Optimierungsleveln relativ gleich ist und bei dem Le-

vel 3 die Programmgröße am kleinsten ist, wird dieses Level für die weitere Verwendung ausgewählt.

Abbildung 4.6: Optimierungsassistent: maximale Programmgeschwindigkeit und Optimierungslevel 3

In Abbildung 4.6 ist das Ergebnis des Optimierungsassitenten für das Optimierungslevel 3 zu sehen. Auf der Y-Achse sind die verschiedenen Programmgeschwindigkeiten (0-5) aufgetragen und auf der X-Achse die Programmgröße. Für die Auswertung ist nur die Programmgröße bei der Programmgeschwindigkeit 5 relevant, da bei dieser Geschwindigkeit das Programm für die maximale Geschwindigkeit optimiert wird.

Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	31,2 ns	31,2 ns	31,2 ns	31,2 ns	0 s		Gesamte Spur
C	Zykluszeit	183,3 ns	183,3 ns	183,3 ns	183,3 ns	0 s		Gesamte Spur
C	Anstiegszeit [80/20%]	26,8 ns	26,8 ns	26,8 ns	26,8 ns	0 s		Gesamte Spur

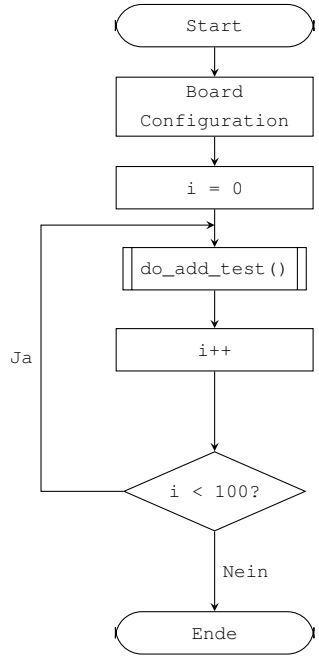
Abbildung 4.7: GPIO: Ein- und Ausschaltzeit für Optimierungslevel 3

Bei diesem Optimierungslevel beträgt die Einschaltzeit $t_{GPIOEIN} \approx 26{,}8\,\text{ns}$ und die Ausschaltzeit $t_{GPIOAUS} \approx 31{,}2\,\text{ns}$ (vgl. Abbildung 4.7).

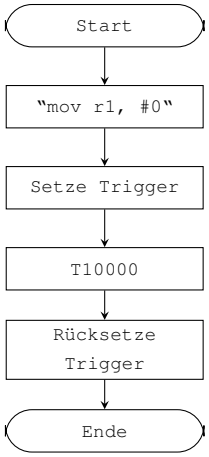
4.2 XMC4700 Relax Kit Lite

4.2.1 Bestimmung der Dauer einer Instruktion

Damit die Glitchdauer die richtige Länge hat, muss zunächst die Dauer einer Instruktion bestimmt werden. Hierzu wird ein Testprogramm[1] verwendet, welches ein Trigger-Signal setzt, dann eine zuvor festgelegte Anzahl an Additions-Instruktionen ausführt und anschließend den Trigger wieder zurücksetzt (siehe Abbildung 4.8b). Dieses Programm wird für drei unterschiedliche Anzahlen von Instruktionen ausgeführt (100, 1000 und 10 000). Da es bei der Ausführung der Instruktionen zu Abweichungen der Berechnungszeit durch externe und interne Einflüsse kommen kann, wird jede Anzahl an Instruktionen 100 Mal ausgeführt (vgl. Abbildung 4.8a).



(a) Programmablaufplan zur Bestimmung der Instruktionsdauer



(b) Programmablaufplan: do_add_test() für 10 000 Instruktionen[1]

Der Programmcode befindet sich im Anhang im Unterabschnitt A.1.3. Die Taktgeschwindigkeit ist bei der IDE „Digital Application Virtual Engineer (DAVE)“ standardmäßig auf 144 MHz eingestellt. Dies entspricht der max. Taktgeschwindigkeit des XMC4700 Relax Kit Lite.

[1] Testprogramm eines NXP-Experten

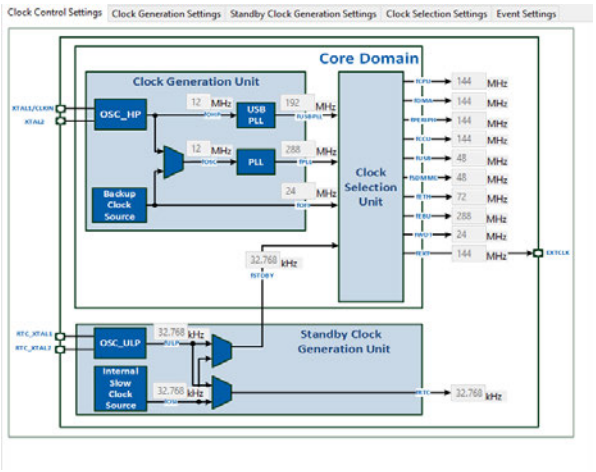


Abbildung 4.9: Übersicht über die verschiedenen Taktfrequenzen des Mikrocontrollers [Standard-Einstellung]

In der Abbildung 4.9 sind die verschiedenen Taktgeschwindigkeiten des XMC4700 zu sehen. In der oberen linken Ecke befindet sich die „Clock Generation Unit“. Diese ist für die Taktgenerierung zuständig. Die hier generierten Frequenzen werden in der „Clock Selection Unit“ verringert und bestimmten Bereichen, wie z. B. der CPU-Taktfrequenz f_{CPU} zugeordnet.

Die Taktfrequenz spielt für die Bestimmung der benötigten Takte für eine Instruktion keine Rolle. Aus diesem Grund wird für den Versuch die Standardeinstellung nicht geändert und die 144 MHz Taktfrequenz beibehalten. Als Instruktion wird „add r1, r1, #1“ genutzt (Additionsinstruktion). Alle wichtigen Daten sind in Tabelle 4.1 festgehalten.

Tabelle 4.1: Eckdaten für den Versuch: Instruktionsdauer

Eigenschaft	Wert
Taktgeschwindigkeit	144 MHz
Instruktion	Addition
Anzahl an Instruktionen	100, 500, 750, 1000 2000, 3450, 10 000, und 30 000

Während des Versuchs wird der Trigger-Pin oszillografiert und mittels PicoScope die Dauer vom Setzen bis zum Rücksetzen eines Pins gemessen.

Beispielhaft werden die Versuche mit 100, 1000 und 10 000-Additionen näher betrachtet:

100-Instruktionen

In Abbildung 4.10 ist die Messung für 100 Instruktionen zu sehen. Am unteren Rand befindet sich die Laufzeitmessung („Höhe Impulsbreite“). Die Höhe der Impulsbreite gibt an, wie lange das Signal auf einem bestimmten Level gehalten wird. Für den Versuch ist dieses Level das logische High-Signal.

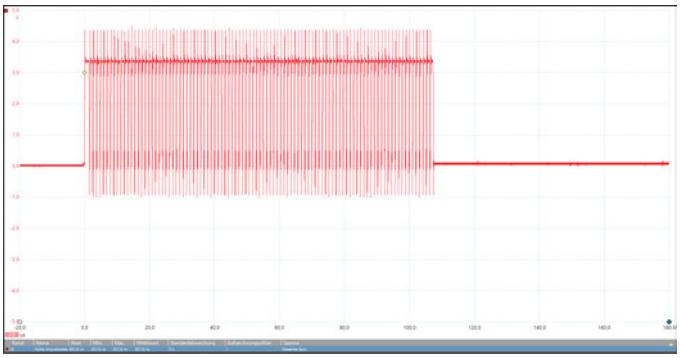


Abbildung 4.10: Laufzeitmessung für 100-Instruktionen

Die nach 100 Versuchen berechnete Impulsbreit/Lautzeit beträgt 831,8 ns.
Teilt man diese Zeit durch die Anzahl an Instruktionen ergibt sich die Zeit für eine Instruktion:

$$\frac{832{,}8\,\text{ns}}{100} = 8{,}328 \cdot 10^{-9}\text{s} = 8{,}328\,\text{ns}. \tag{4.1}$$

1000-Instruktionen

Die Messung für 1000 Instruktionen ist in Abbildung 4.11 dargestellt. Auch hier befindet sich die Messung der Laufzeit am unteren Rand der Abbildung.

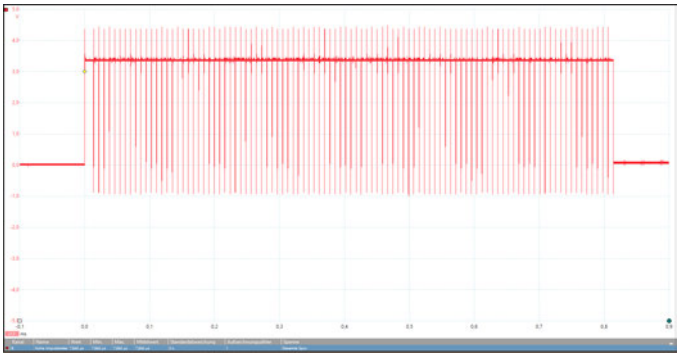


Abbildung 4.11: Laufzeitmessung für 1000-Instruktionen

Die gemessene Impulsbreite beträgt in diesem Fall 7,844 µs.
Die Dauer für eine Instruktion beträgt:

$$\frac{7{,}844\,\mu s}{1000} = 7{,}844 \cdot 10^{-9} s = 7{,}844\,ns. \tag{4.2}$$

10 000-Instruktionen

Für 10 000-Instruktionen sieht der oszillografierte Zeitverlauf folgendermaßen aus:

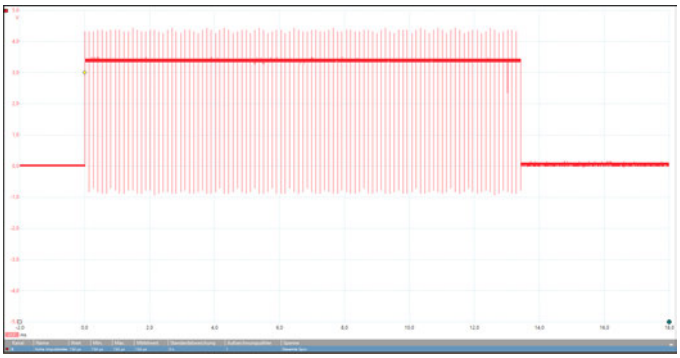


Abbildung 4.12: Laufzeitmessung für 10 000-Instruktionen

Die Impulsbreite beträgt 134 µs.
Bei 10 000-Instruktionen ergibt sich die Dauer einer Instruktion von 13,4 ns (siehe Gleichung 4.3).

$$\frac{134\,\mu s}{10\,000} = 13{,}4 \cdot 10^{-9} s \tag{4.3}$$

Aus den gezeigten Versuchen und den Versuchen für die restlichen Durchläufe mit 500, 750, 2000, 3450 und 30 000 Additionen lässt sich folgender Zusammenhang zwischen Laufzeit und Anzahl an Instruktionen aufstellen (vgl. Tabelle 4.2 und Abbildung 4.13).

Tabelle 4.2: Laufzeit und Anzahl an Instruktionen

Anzahl an Instruktionen	Dauer einer Instruktion in [ns]	Anzahl Systemtakte
100	8,318	1,1978
500	7,978	1,2488
750	7,892	1,1364
1000	7,844	1,1295
2000	9,155	1,3183
3450	13,449	1,9367
10 000	13,400	1,9296
30 000	13,387	1,9277

Bei den 30 000 Additionen tritt beim Übersetzen des Programmcodes ein interner Fehler des Compilers auf. Das Programm lässt sich aber auf dem Mikrocontroller ausführen und es lassen sich Messungen durchführen.

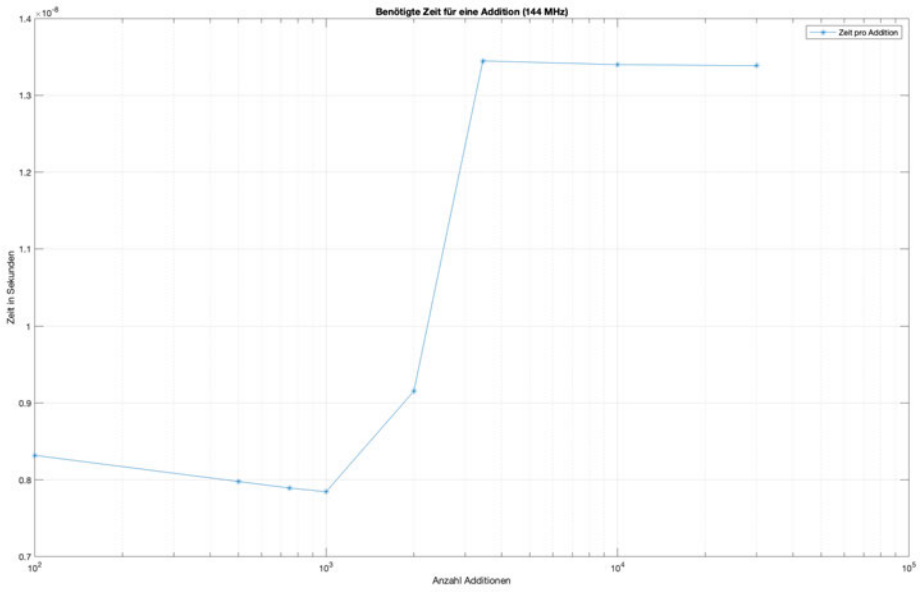


Abbildung 4.13: Zeit pro Addition für 100, 500, 750, 1000, 2000, 3450, 10 000 und 30 000 Additionen (144 MHz)

In Abbildung 4.13 und Tabelle 4.2 ist zu erkennen, dass für eine niedrige Anzahl an Additionen (100 bis 1000) die Dauer gemittelt bei $8{,}0080 \cdot 10^{-9}$ s und die Anzahl von Systemtakten bei 1,1531 liegen. Bei ca. 2000 Additionen steigt die benötigte Zeit für eine Instruktion sprunghaft auf $13{,}449 \cdot 10^{-6}$ s. Dies entspricht ca. 1,9 Systemtakten bei der zuvor festgelegten Taktgeschwindigkeit von 144 MHz.

Für eine andere Taktgeschwindigkeit, z. B. 72 MHz ergibt sich der selbe zeitliche Verlauf, nur die benötigte Zeit für die Additionen ist größer (ca. doppelt so groß). Die Ergebnisse für 72 MHz sind in Abbildung 4.14 abgebildet.

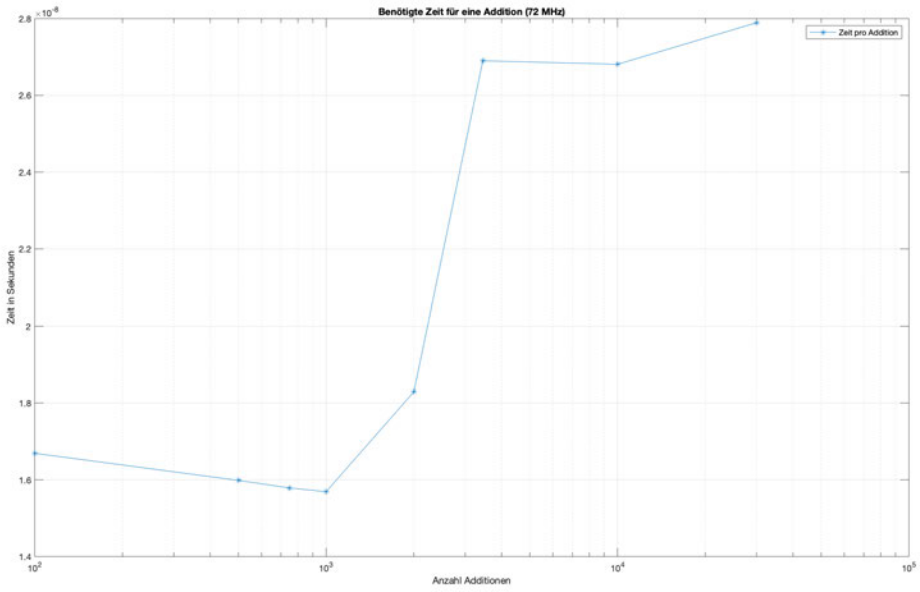


Abbildung 4.14: Zeit pro Addition für 100, 500, 750, 1000, 2000, 3450, 10 000 und 30 000 Additionen (72 MHz)

4.2.2 Fazit

Für den Aufwand des RSA wird angenommen, dass er über der Grenze von 2000-Additionsanweisungen liegt. Dieser ist schwer zu vergleichen, da beim RSA überwiegend Operationen wie potenzieren, multiplizieren oder die Modulo-Rechnung benötigt werden. Diese sind viel rechenintensiver als eine einfache Addition. Für die weiteren Versuche wird angenommen, dass eine Instruktion ca. 1,9-Systemtakte benötigt.

4.3 Berechnung der theoretischen Zeiten

In diesem Kapitel werden die verschiedenen, benötigten Zeiten berechnet (Einschaltzeit des Glitches $t_{on,glitch}$, Ausschaltzeit des Glitches $t_{off,glitch}$, Ansteuerungszeit für den Trigger-Pin t_{pin}, Ansteuerungszeit des MOSFETs t_{mosfet} und die Glitchdauer t_{glitch}). Diese Zeiten werden benötigt um die genaue Einschaltdauer des Triggerpins zu bestimmen. Die Zeit t_{glitch} gibt an, wie lange der MOSFET im leitenden Zustand bleiben soll. Aus dieser Angabe lässt sich berechnen, wie lange der MOSFET „arbeitet“ (t_{mosfet}). Die

Zeit, in welcher der Glitch wirkt, liegt somit zwischen den Zeiten t_{glitch} und t_{mosfet}. Die Zeit t_{pin} gibt an, wie lange der Triggerpin für eine Zeit t_{mosfet} eingeschaltet werden muss. Die genauere Zusammensetzung dieser Zeiten ist in diesem Kapitel beschrieben.

4.3.1 Einschaltzeit des Glitch

Die Einschaltzeit des Glitches $t_{on,glitch}$ wird durch zwei Parameter bestimmt: der Einschaltdauer des GPIO Pins und der Einschaltzeit des MOSFET. Die Einschaltzeit des GPIO beträgt 26,8 ns (vgl. Unterabschnitt 4.1.1) und die des MOSFET ist 17,5 ns.
Es ergibt sich eine Gesamteinschaltzeit von

$$t_{on,glitch} = 26{,}8 \cdot 10^{-9}\,\mathrm{s} + 17{,}5 \cdot 10^{-9}\,\mathrm{s} = 44{,}3 \cdot 10^{-9}\,\mathrm{s} = \underline{\underline{44{,}3\,\mathrm{ns}}}. \tag{4.4}$$

Die $t_{on,glitch}$-Dauer beschreibt die benötigte Zeit vom Zeitpunkt an, ab dem im Programm der Tiggerpin für den Glitch auf High gesetzt wird, bis zu dem Zeitpunkt, ab dem der MOSFET voll durchgeschaltet ist. Hierbei ist zu beachten, dass bei der Berechnung bestimmte Faktoren wie die Gate-Schwellspannung $V_{GS(th)}$ des MOSFET nicht berücksichtigt werden. Die Gate-Schwellspannung für die Bestimmung der Pin-Einschaltzeit liegt bei 80 % von 3,3 V. Laut Datenblatt schaltet der MOSFET spätestens bei einer Schwellspannung von 1,2 V (vgl. [Rec14]). Diese Abweichung der Schwellspannung wirkt sich auf Ein- und Ausschaltzeiten des GPIO aus. Die Auswirkungen sind in Abbildung 4.15 zu sehen.

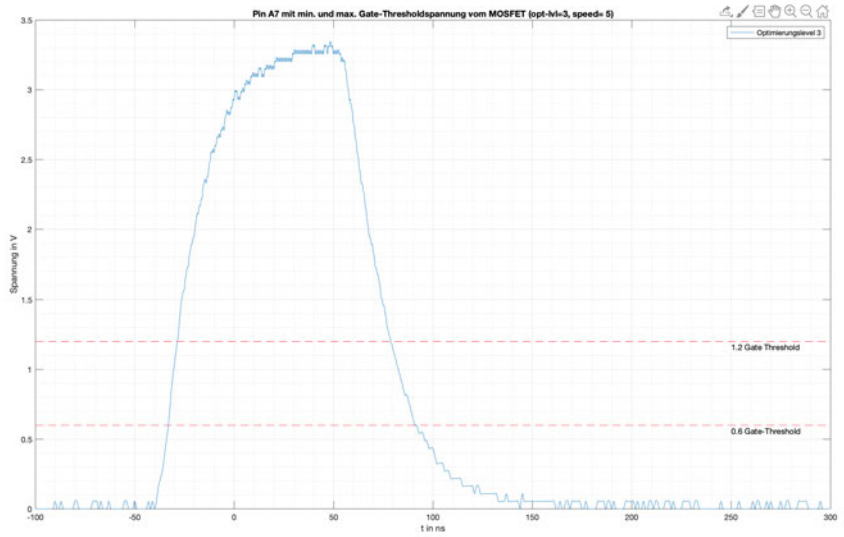


Abbildung 4.15: Ein- und Ausschaltzeit für einen Pin mit 0,6 V und 1,2 V $V_{GS(th)}$ [opt.lvl = 4, speed = 5]

0,6 V Gate-Schwellspannung

Aus der Abbildung 4.15 resultiert eine Einschaltzeit des Pins $t_{on,GPIO}$ für eine Gate-Schwellspannung von 0,6 V mit

$$|t_{on,GPIO}| = |(-40\,\text{ns}) - (-33{,}25\,\text{ns})| = \underline{\underline{6{,}75\,\text{ns}}}. \tag{4.5}$$

Für 0,6 V $V_{GS(th)}$ liegt $t_{on,glitch}$ bei

$$t_{on,gltich} = 6{,}75\,\text{ns} + 17{,}5\,\text{ns} = \underline{\underline{24{,}25\,\text{ns}}}. \tag{4.6}$$

1,2 V Gate-Schwellspannung

Die $t_{on,GPIO}$-Zeit für eine Gate-Schwellspannung von 1,2 V liegt bei ca.

$$|t_{on,GPIO}| = |(-40\,\text{ns}) - (-28{,}75\,\text{ns})| = \underline{\underline{11{,}25\,\text{ns}}}. \tag{4.7}$$

Daraus resultiert eine $t_{on,glitch}$ von

$$t_{on,glitch} = 11{,}25\,\text{ns} + 17{,}5\,\text{ns} = \underline{\underline{28{,}75\,\text{ns}}}. \tag{4.8}$$

Die verschiedenen Einschaltzeiten sind in Tabelle 4.3 festgehalten.

Tabelle 4.3: Einschaltzeit des Glitch $t_{on,glitch}$

Beschreibung	Wert in [ns]
Allgemein	44,3
0,6 V-$V_{GS(th)}$	24,25
1,2 V-$V_{GS(th)}$	28,75

4.3.2 Ausschaltzeit des Glitch

Für die Bestimmung der Ausschaltzeit wird angenommen, dass die Spannung am Pin die 3,3 V während des Einschaltvorgangs erreicht. Ist dies nicht der Fall, kann nicht pauschal eine Ausschaltzeit bestimmt werden, da nicht bekannt ist, von welchen Spannungsniveau ausgeschaltet wird. Zur Bestimmung der Zeiten wird die Abbildung 4.15 verwendet.

Allgemein

Die allgemeinen Ausschaltzeiten können dem Unterabschnitt 4.1.1 und Tabelle 3.3 entnommen werden. Daraus ergibt sich eine Ausschaltzeit $t_{off,glitch}$ von

$$t_{off,glitch} = 31{,}2 \cdot 10^{-9}\,\text{s} + 80 \cdot 10^{-9}\,\text{s} = 111{,}2 \cdot 10^{-9}\,\text{s} = \underline{\underline{111{,}2\,\text{ns}}}. \tag{4.9}$$

0,6 V Gate-Schwellpannung

In Abbildung 4.15 ist zu erkennen, dass das Signal bei ca. 53 ns beginnt abzufallen. Das 0,6 V-Level erreicht es bei 90,75 ns. Es ergibt sich $t_{off,GPIO} = 90{,}75\,\text{ns} - 53\,\text{ns} = 32{,}75\,\text{ns}$.

Ersetzt man in Gleichung 4.9 die 31,2 ns für den allgemeinen Fall durch die gerade berechneten 32,75 ns, beträgt die $t_{off,glitch}$-Zeit:

$$t_{off,glitch} = 32{,}75\,\text{ns} + 80\,\text{ns} = \underline{\underline{112{,}75\,\text{ns}}}. \tag{4.10}$$

1,2 V Gate-Schwellspannung

Ähnlich wie, bei der 0,6 V Gate-Schwellspannung-Spannung beginnt das Signal bei ca. 90,75 ns zu fallen. Anders als beim vorherigen Spannungsniveau, erreicht der Pin die 1,2 V-Grenze bei 78,75 ns (vgl.Abbildung 4.15). Somit beträgt die $t_{off,GPIO}$ für diesen Fall 12 ns.

Eingesetzt in Gleichung 4.9 berechnet sich die folgende Ausschaltdauer (siehe Gleichung 4.11).

$$t_{off,glitch} = 12\,\text{ns} + 80\,\text{ns} = \underline{\underline{92\,\text{ns}}} \tag{4.11}$$

Die Ausschaltzeiten sind zur besseren Übersicht in der nachfolgenden Tabelle festgehalten.

Tabelle 4.4: Ausschaltzeit des Glitches $t_{off,glitch}$

Beschreibung	Wert in [ns]
Allgemein	111,2
0,6 V-$V_{GS(th)}$	112,75
1,2 V-$V_{GS(th)}$	92

4.3.3 Ansteuerungszeit des Triggerpins

Das Tiva Board liefert das Triggersignal für den MOSFET. Somit ist das Tiva Board von essentieller Wichtigkeit für die Dauer des Glitches. Um die Zeit zu bestimmen, in welcher der Trigger Pin eingeschaltet sein muss, werden die Einschaltzeiten aus Tabelle 4.3 benötigt. Die Ausschaltzeiten aus Tabelle 4.4 werden hierbei nicht benötigt, da sie erst wirken, wenn im Programm das Trigger-Signal ausgeschaltet wurde (vlg. Abbildung 4.16).

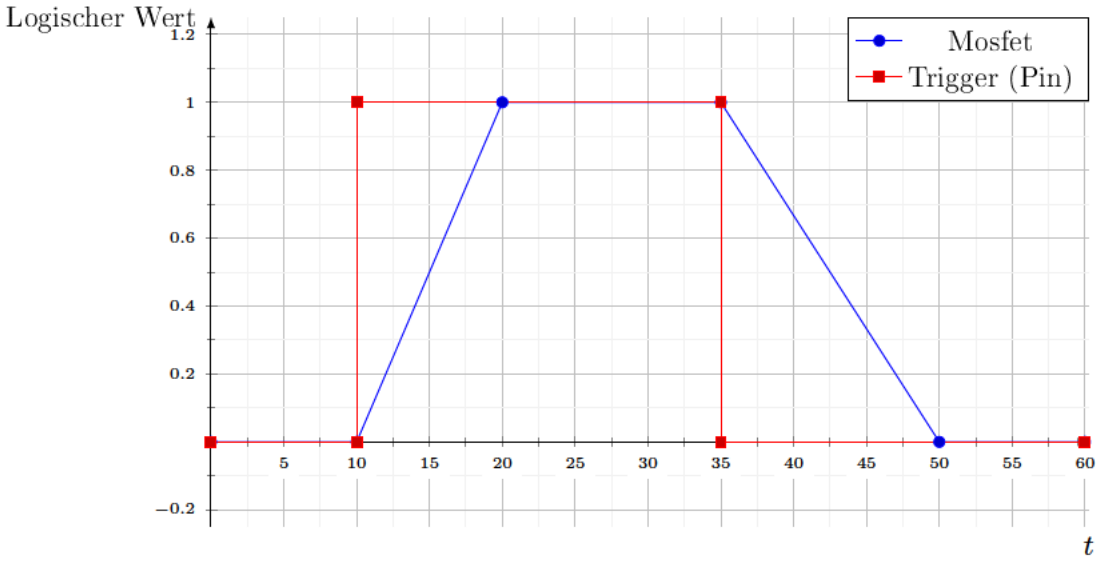


Abbildung 4.16: Skizze: Einschaltdauer Pin am Tiva Board

In dem in Abbildung 4.16 gezeigten Kontext bedeutet der logische Wert 1 für den MOSFET, dass dieser geschaltet ist.
Die theoretische Dauer, für die der Triggerpin auf dem logischen 1 Wert sein muss, berechnet sich aus $t_{on,GPIO}$, $t_{on,MOSFET}$ und der gewünschten Glitchdauer t_{glitch}. Die Rechnung ist in allgemeiner Form in Gleichung 4.12 zu sehen.

$$\begin{aligned} t_{pin} &= t_{on,GPIO} + t_{on,MOSFET} + t_{glitch} \\ &= t_{on,glitch} + t_{glitch} \end{aligned} \tag{4.12}$$

Mittels dieser Gleichung lassen sich die drei Fälle (Allgemein, 0,6 V Gate-Schwellspannung und 1,2 V Gate-Schwellspannung) bestimmen (siehe Tabelle 4.5).

Tabelle 4.5: Trigger-Pin: Ein-Dauer für eine Glitchdauer von 100 ns

Beschreibung	Wert in [ns]
Allgemein	144,3
0,6 V Gate-Schwellspannung	124,25
1,2 V Gate-Schwellpannung	128,75

4.3.4 MOSFET-Zeit

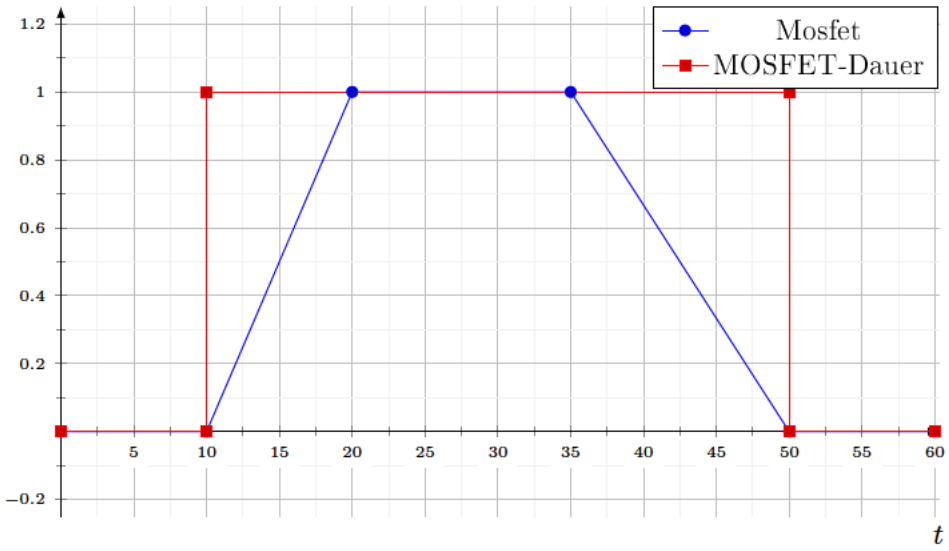


Abbildung 4.17: Skizze: MOSFET-Dauer

In Abbildung 4.17 ist die benötigte Zeit für das Einschalten, das Ausschalten und die Zeit, in welcher der MOSFET voll leitend ist, zu erkennen (vgl. rote Linie). Diese MOSFET-Dauer t_{mosfet} gibt an, wie lange der MOSFET geschaltet ist. Diese Zeit ist wichtig, da in diesem Bereich der Glitch wirken kann. Ob dies der Fall ist, hängt von der verbleibenden VDDC-Spannung ab. Die MOSFET-Dauer lässt sich durch die Addition der Einschaltzeit $t_{on,glitch}$, der Zeit t_{glitch} und der Ausschaltzeit $t_{off,glitch}$ berechnen. In allgemeiner Form berechnet sich die Zeit folgendermaßen:

$$t_{mosfet} = t_{on,glitch} + t_{glitch} + t_{off,glitch} = \underline{\underline{44{,}3\,\text{ns} + t_{glitch} + 111{,}2\,\text{ns}}} \tag{4.13}$$

Für die drei Fälle und $t_{glitch} = 100\,\text{ns}$ ergeben sich folgende Werte:

Tabelle 4.6: MOSFET: Ein-Dauer für eine Glitchdauer von $100\,\text{ns}$

Beschreibung	Wert in [ns]
Allgemein	255,5
0,6 V Gate-Thresholdspannung	237
1,2 V Gate-Thresholdspannung	220,75

4.3.5 Theoretische Glitchzeit

Die theoretische Glitchzeit beginnt, anders als die Trigger-Pin Zeit, erst nachdem der MOSFET geschaltet hat. Sie kennzeichnet die Zeit, in welcher der MOSFET „voll“ geöffnet ist. Für die Glitchzeit wird nur diese Zeit betrachtet, da in diesem Zeitraum die Auswirkungen des Glitches auf das DUT maximal sind.

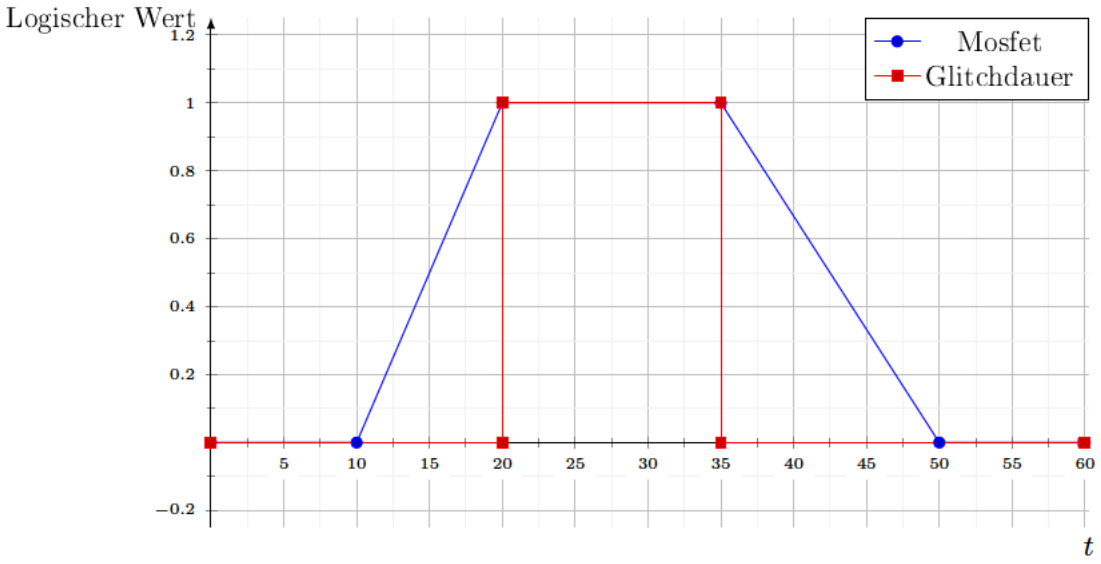


Abbildung 4.18: Skizze: Glitchdauer

Betrachtet man die Abbildung 4.18 fällt auf, dass sich die Glitchdauer t_{glitch} sowohl auf die MOSFET-Dauer t_{mosfet}, als auch auf die Trigger-Pin-Dauer t_{pin} auswirkt (vgl. Gleichung 4.12 und Gleichung 4.13).
Wichtig ist, dass die Glitchzeit t_{glitch} keinesfalls die genaue Dauer des Glitches wiedergibt. Sie gibt die Dauer an, in welcher der MOSFET voll leitend ist. Diese Dauer kann im Gegensatz zu den anderen Zeiten durch einen GPTM gesteuert werden. Die tatsächliche Glitchzeit liegt zwischen der Zeit t_{glitch} und der Dauer, in welcher der MOSFET „arbeitet“ (t_{mosfet}).

4.3.6 Überprüfung der theoretisch berechneten Zeiten

Zur Überprüfung der in Unterabschnitt 4.3.3 berechneten $t_{on,GPIO}$ Zeit wird der GPIO Pin A7 des Tiva Boards mit dem MOSFET verbunden. Als Source-Spannung dient der 5 V-Pin am Tiva Board (vgl. Abbildung 4.19).

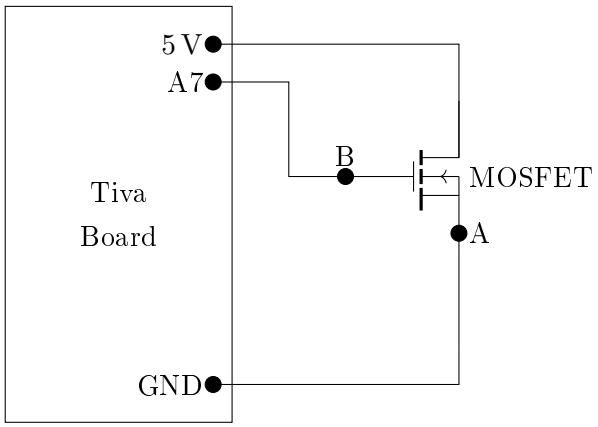


Abbildung 4.19: Aufbau zur Überprüfung der theoretischen $t_{on,GPIO}$-Zeit

An dem Punkt A wird die Spannung hinter dem MOSFET gemessen. Ist dieser nicht geschaltet, liegt dort das Masse-Potential an, sonst nicht. Der Punkt B ist für das Messen des Trigger-Signals. Für alle Versuche wird der Grundaufbau aus Abbildung 4.19 verwendet und durch ein weiteren Messpunkt, welcher das Ausschalten des Trigger-Pins anzeigt, erweitert.

Glitchdauer

Für den Versuch wird die Glitchzeit $t_{glitch} = 200\,\text{ns}$ gewählt.

MOSFET-Dauer

Die MOSFET-Dauer t_{mosfet} lässt sich durch die angegebene Glitchdauer bestimmen. Hierzu wird t_{gltich} in die Gleichung 4.13 eingesetzt:

$$t_{mosfet} = t_{on,glitch} + 200\,\text{ns} + t_{off,glitch}.$$

Für den allgemeinen Fall ergibt sich eine MOSFET-Dauer von:

$$\begin{aligned} t_{mosfet} &= 44{,}3\,\text{ns} + 200\,\text{ns} + 111{,}2\,\text{ns} \\ &= \underline{\underline{355{,}5\,\text{ns}}} \end{aligned}$$

Triggerpin Zeit

Aus $t_{glitch} = 200\,\text{ns}$ und der Gleichung 4.12 resultiert eine Triggerpin-Zeit

$$\begin{aligned} t_{Pin} &= t_{on,Glitch} + t_{glitch} = 44{,}3\,\text{ns} + 200\,\text{ns} \\ &= \underline{\underline{244{,}3\,\text{ns}}}. \end{aligned} \tag{4.14}$$

Ergebnisse

Die Messungen der Zeiten ergeben die folgenden Ergebnisse. Für $t_{glitch} = 200\,\text{ns}$ ist die Messung der Triggerpin-Zeit in Abbildung 4.20 zu sehen. In den nachfolgenden Abbildungen ist das gelbe Signal die Spannung, welche am MOSFET anliegt, das blaue Signal die Spannung, welche am MOSFET-Gate anliegt und das rote Signal ein Hilfspin. Dieser Hilfspin zeigt an, wann der Triggerpin schaltet.

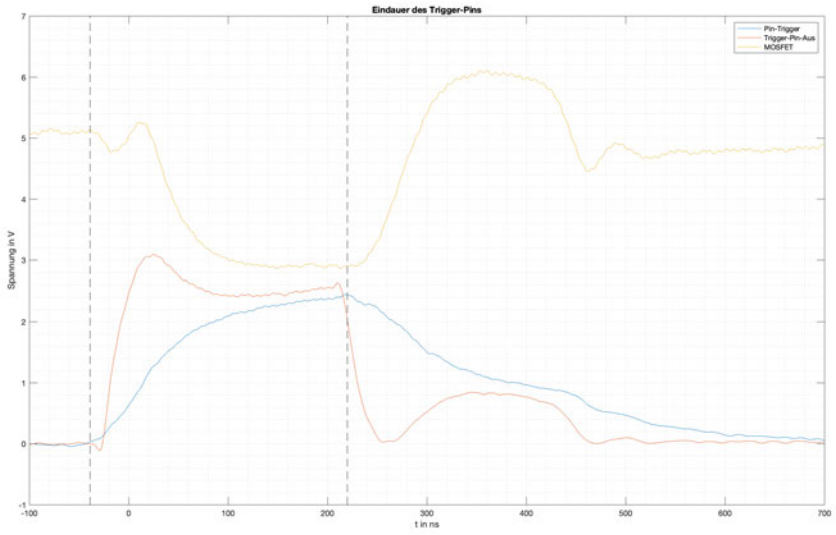


Abbildung 4.20: Messung der Triggerpin-Zeit für eine Glitchdauer von 200 ns

Der linke Marker (t_1) in der Abbildung 4.20 kennzeichnet den Einschaltmoment des Triggerpins. Dieser liegt in der Messung bei ca. $t_1 = -38{,}85\,\text{ns}$. Der zweite Marker ($t_2$) zeigt den Zeitpunkt, an dem der Triggerpin ausgeschaltet wird. In Abbildung 4.20 ist $t_2 = 219{,}8\,\text{ns}$. Die Differenz der beiden Marker beschreibt die Triggerpin-Zeit t_{pin}. Daraus folgt

$$\begin{aligned} t_{pin} &= t_2 - t_1 = 219{,}8\,\text{ns} - (-38{,}85\,\text{ns}) \\ &= \underline{\underline{258{,}65\,\text{ns}}}. \end{aligned} \tag{4.15}$$

Der theoretisch berechnete Wert liegt bei 244,3 ns. Es ergibt sich eine Abweichung von 13,85 ns. Diese ist durch ungenaues Messen und dem ungenauen Ablesen in dem kleinen Zeitbereich zu erklären.

Der zeitliche Spannungsverlauf für den MOSFET ist in Abbildung 4.21 zu sehen.

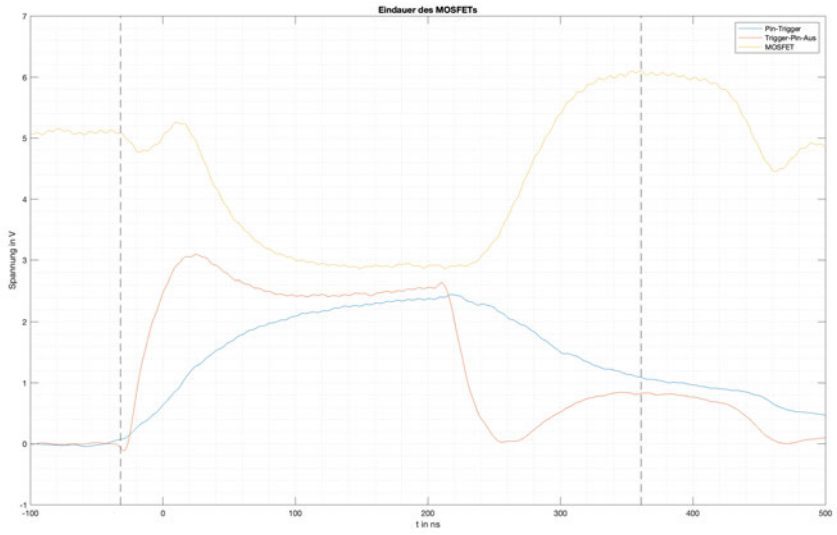


Abbildung 4.21: Messung der MOSFET-Dauer für 200 ns

Die MOSFET-Dauer startet mit dem Beginn (linke Markierung) des steigenden Trigger-Signals (blau) und endet, wenn die MOSFET-Spannung ihren Höhepunkt erreicht hat (rechte Markierung).

Mittels dieser beiden Marker lässt sich die MOSFET-Dauer berechnen.

$$\begin{aligned} t_{mosfet} &= t_2 - t_1 = 360{,}7\,\text{ns} - -32{,}12\,\text{ns} \\ &= \underline{\underline{392{,}82\,\text{ns}}}. \end{aligned} \tag{4.16}$$

Die Abweichung zum theoretisch berechneten Wert liegt bei 37,3200 ns. Diese ist auf der einen Seite durch das ungenaue Messen und Ablesen im ns-Bereich zu erklären. Auch werden in der theoretischen Berechnung bestimmte Faktoren, wie z.B. der Überschwinger oder die tatsächliche Schwellspannung, nicht betrachtet. Wenn der Wert verwendet wird, bei welchem die MOSFET-Spannung das 5 V-Niveau wieder erreicht, ergibt sich ein Wert, der näher an dem theoretisch berechneten Wert liegt (288,8 ns − (−32,12 ns) = 320,92 ns).

Auch die gewählte Glitchdauer t_{glitch} ist in der Messung wiederzufinden (siehe. Abbildung 4.22).

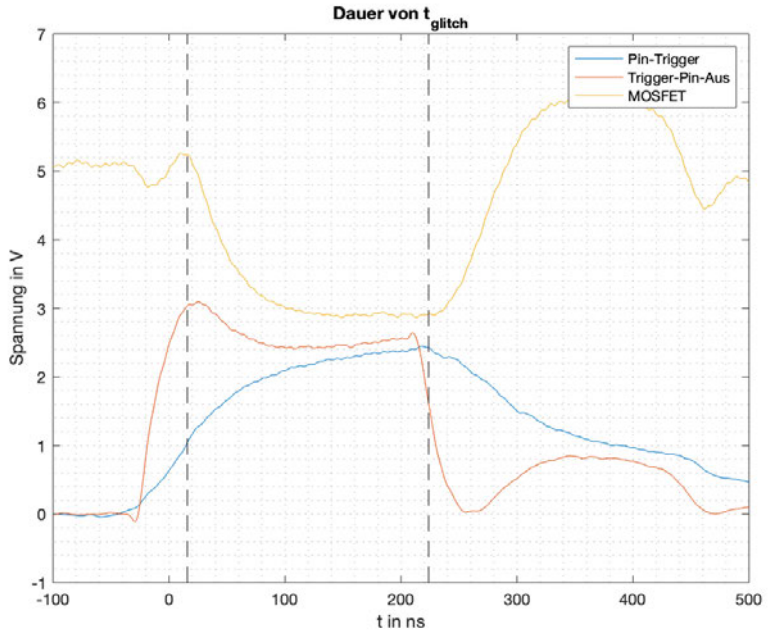


Abbildung 4.22: Messung der Glitchdauer für 200 ns

Die Glitchdauer charakterisiert sich dadurch, dass sie die Ein- und Ausschaltzeiten vernachlässigt. Die linke Markierung zeigt den Zeitpunkt, ab dem der MOSFET geschaltet ist, während die rechte Markierung den Zeitpunkt markiert an dem der Trigger-Pin ab-

geschaltet wird. Aus diesen beiden Markierungen ergibt sich, für diese Messung, eine Glitchzeit von:

$$t_{glitch} = t_2 - t_1 = 223{,}8\,\text{ns} - 15{,}79\,\text{ns} = \underline{\underline{208{,}01\,\text{ns}}}. \tag{4.17}$$

Dieser Wert entspricht fast dem theoretisch gewählten Wert von 200 ns. Die Abweichung ist mit den selben Begründungen wie in der beiden Auswertungen zuvor zu erklären.

4.3.7 Anzahl an Instruktionen während der MOSFET-Schaltzeit

Wie aus dem Abschnitt 3.3 bekannt beträgt die reine MOSFET-Schaltzeit:

$$t_{mosfet} = 17{,}5\,\text{ns} + 80\,\text{ns} = \underline{\underline{97{,}5\,\text{ns}}} \approx 10{,}26\,\text{MHz}. \tag{4.18}$$

Durch den Vergleich der MOSFET-Schaltzeit, den in Tabelle 3.2 angegeben f_{CPU} Frequenzen und den aus Unterabschnitt 4.2.1 gewonnen Informationen lässt sich bestimmen, wie viele Instruktionen auf dem XMC4700 Relax Kit Lite während der MOSFET-Schaltzeit ausgeführt werden. Diese Anzahl spiegelt die kleinste Anzahl der Instruktionen wieder, die theoretisch übersprungen werden kann.
In der nachfolgenden Auflistung (Tabelle 4.7) wird nur die Schaltzeit des MOSFET betrachtet. Das bedeutet, dass mögliche Verlängerungen durch eine längere Triggerdauer nicht mit berücksichtigt werden. Für die maximale CPU-Taktfrequenz von 288 MHz werden während der Schaltvorgänge des MOSFET 14,04 Instruktionen ausgeführt. Bei einer eingestellten CPU-Taktfrequenz zwischen 20,57 bis 19,2MHz entspricht die Schaltzeit des MOSFET genau der Dauer für eine Instruktion.

Tabelle 4.7: Übersicht: CPU-Taktfrequenzen, MOSFET-Schaltzeit und Anzahl an Instruktionen (Durchgeführte Instruktionen > 3450, Taktquelle = 288 MHz, 1,92 Takte pro Instruktion)

CPU-Taktfrequenz in [MHz]	in [ns]	Zeit pro Instruktion	MOSFET-Schaltzeit	Anzahl der Instruktionen
288 MHz	3,47 ns	6,66 ns	97,5 ns	14,625
144 MHz	6,944 ns	13,33 ns	97,5 ns	7,313
96 MHz	10,41 ns	20 ns	97,5 ns	4,875
72 MHz	13,88 ns	26,66 ns	97,5 ns	3,656
57,6 MHz	17,36 ns	33,33 ns	97,5 ns	2,925
48 MHz	20,83 ns	40 ns	97,5 ns	2,438
41,14 MHz	24,30 ns	46,66 ns	97,5 ns	2,089
36 MHz	27,77 ns	53,33 ns	97,5 ns	1,828
32 MHz	31,25 ns	60 ns	97,5 ns	1,625
28,8 MHz	34,72 ns	66,66 ns	97,5 ns	1,463
26,18 MHz	38,19 ns	73,33 ns	97,5 ns	1,330
24 MHz	41,66 ns	80 ns	97,5 ns	1,219
22,15 MHz	45,14 ns	86,66 ns	97,5 ns	1,125
20,57 MHz	48,61 ns	93,33 ns	97,5 ns	1,045
19,2 MHz	52,08 ns	100 ns	97,5 ns	0,975
...	...	...	...	...

Bei der Bestimmung der CPU-Taktfrequenz ist zu beachten, dass sie nur gleich oder die Hälfte der Systemtaktfrequenz sein kann. Für bestimmte Anwendungen wie Ethernet oder UART sind jedoch bestimmte Taktfrequenzbänder einzuhalten.

5 Durchführung

5.1 Parametersuche

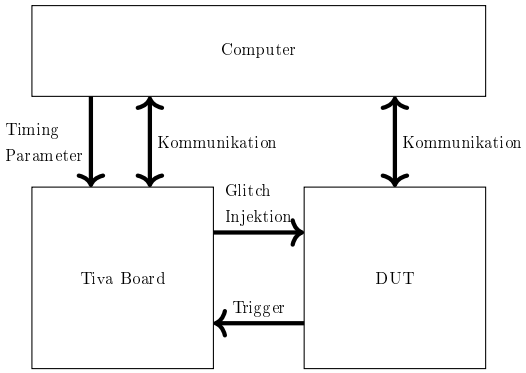


Abbildung 5.1: Schematische Übersicht: Parametersuche Aufbau

5.1.1 Aufbau

In Abbildung 5.1 ist der schemenhafte Aufbau der Parametersuche zu sehen. Ein Computer dient als übergeordneter „Master“. Er kommuniziert sowohl mit dem Trigger-Erzeuger (Tiva Board), als auch mit dem DUT (XMC4700 Relax Kit Lite).

Der „Master“ übernimmt, während der Parametersuche, die folgenden Aufgaben:

1 Parameter an den Trigger-Erzeuger senden,

2 Glitch-Durchlauf starten,

3 Ergebnisse auswerten.

Programmablauf

Der Programmablauf aus der Sicht des Computers besteht aus der Festlegung der Parameterbereiche, dem Aufbau der Kommunikation mit dem Trigger-Erzeuger und dem DUT, der Auswertung der Glitchergebnisse und dem Durchlauf der Parametersuche.
Der Parameterbereich umfasst die Größen: Glitchdauer, Verzögerungszeit und Anzahl an Durchführung pro Parametersatz.
Ein schemenhafter Programmablaufplan befindet sich im Anhang (siehe Abbildung A.1).

Kommunikation

Für die Kommunikation zwischen den verschiedenen Geräten stehen die in der Tabelle 5.1 gezeigten Befehle zur Verfügung. Mittels dieser lässt sich der Ablauf der Parametersuche steuern.

Tabelle 5.1: Übersicht: Kommunikationsbefehle für die Parametersuche

Befehl	Bedeutung
/P:para1:para2:	Glitchparameter [max. 20 Zeichen] para1 = Glitchdauer para2 = Glitchverzögerungszeit
/S	Startbefehl
/R	Rücksetzbefehl
/N	Keine Auswirkung des Glitches
/F	Auswirkung des Glitches
/W	Verbindung prüfen

Auf dem XMC4700 Relax Kit Lite wird die aus dem Unterabschnitt 4.2.1 bekannte „do_add_test“-Funktionen verwendet. Anders als aus dem Kapitel bekannt, wird die Anzahl an Add-Instruktionen fest auf 10 000 gestellt. Außerdem wird der bekannte Programmablauf auf dem XMC4700 Relax Lite Kit um die UART-Kommunikation und einen Watchdog-Handler ergänzt.
Die IDE DAVE ermöglicht es diese Anwendungen als „Applikation“ hinzuzufügen und legt automatisch den benötigten Code und die benötigten Imports an (siehe Abbildung 5.2).

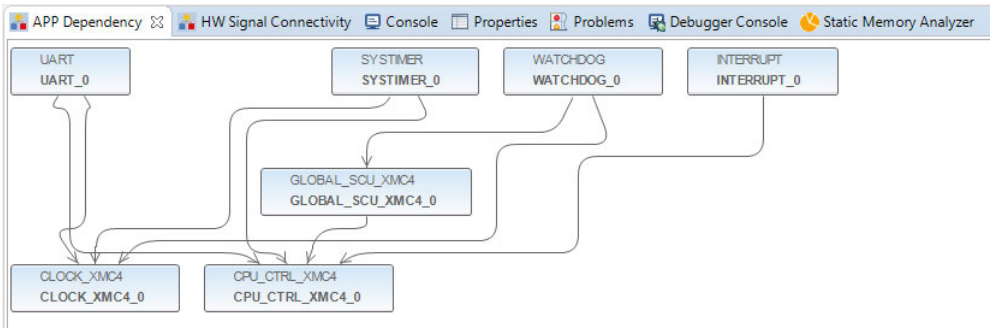


Abbildung 5.2: Parametersuche: Dave Applionsübersicht

Der Programmablaufplan befindet sich im Anhang (vgl. Abbildung A.2). Die Main des Programms ist als Auszug in Listing 5.1.1 zu sehen.

```
...
int main(void){
        ...
        while(1U){
                if(uartRxCounterOld != uartRxCounter){
                        if(text[i-1] == '\n'){
                                if(((char)text[1]) == 'S')
                                        do_add_test();
                                for(i = 0; i < MAXZEICHEN;i++)
                                        text[i] = '\0';
                                i = 0;
                                uartRxCounterOld = uartRxCounter;
                        }
                }
        }
}
...
```

Die Zeilen 9 bis 11 sind für das Zurücksetzen der Nachricht zuständig. Dies ist notwendig, da die nachfolgende Nachricht kürzer sein könnte. Würde die Nachricht nicht mit neutralen Elementen gefüllt werden, würde die neue Nachricht nur einen Teil überschreiben und der Rest der alten Nachricht würde in der Variable gespeichert bleiben. Dies kann zu Fehlern in der Auswertung der Nachricht führen.
Durch die Zuweisung „uartRxCounterOld = uartRxCounter“ wird dem Programm realisiert, dass die neue Nachricht abgearbeitet wurde. Die Variable uartRxCounter wird im Empfangs-Interrupt gesetzt, sobald ein neues Zeichen einer Nachricht empfangen wird.
Die Kommunikation allgemein wird, wie bereits erwähnt, mittels UART realisiert. Das Protokoll legt die Baudrate, die Anzahl der Paketgröße in Bit, die Parität und die Anzahl an Stopp-Bits fest.

5.1.2 Kommunikation zwischen den Geräten

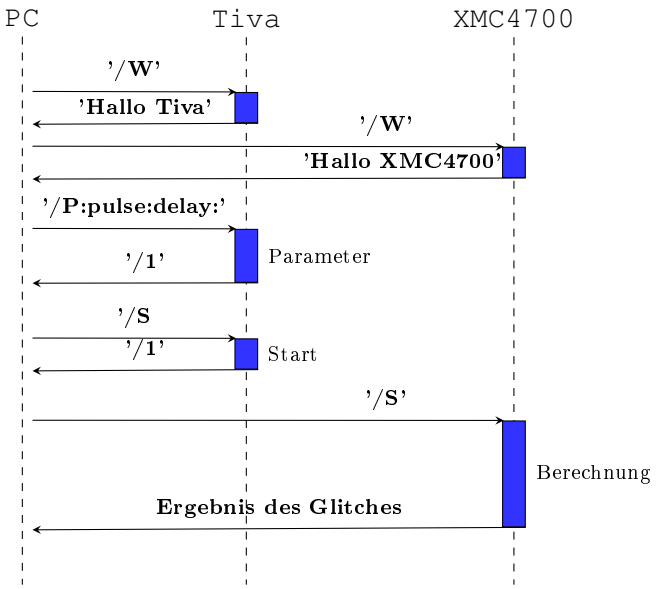


Abbildung 5.3: Übersicht: Erfolgreiche Kommunikation zwischen dem PC, Tiva Board und dem XMC4700 Relax Kit Lite (1 Berechnung)

In Abbildung 5.3 ist die Kommunikation zwischen den verschiedenen Geräten im zeitlichen Verlauf dargestellt. Nach dem der PC die Verbindungen für die UART-Kommunikation, zu den beiden anderen Geräten aufgebaut hat, prüft das Programm, ob diese beiden Gerät auch antworten. Anschließend werden die Glitchparameter vom PC an das Tiva

Board übermittelt und vom Tiva Board bestätigt. Zuletzt wird an beide Geräte (Tiva und XMC4700) der Startbefehl geschickt.
Der letzte Schritt wird im Parametersuchprogramm beliebig oft wiederholt. Ist die Untersuchung eines Parametersatzes fertig, wird der nächste Parametersatz übertragen und getestet. Dies geschieht solange, wie es einen noch nicht geprüften Parametersatz gibt.

5.1.3 Versuch

Die Parametersuche wird für drei verschiedene Taktfrequenzen (f_{CPU}) durchgeführt. Hierbei werden die maximale einstellbare Taktfrequenz(144 MHz) und zwei weitere Taktfrequenzen verwendet. Die beiden anderen Taktfrequenzen werden so gewählt, dass ihre Zykluszeiten relativ nah an der Dauer von 3 Instruktionen liegen (vgl. Tabelle 3.2). Diese beiden Taktfrequenzen sind 72 und 48 MHz. Die Anzahl an durchgeführten Additionen wird für diesen Versuch auf 10 000 festgelegt und das Tiva Board wird, wie in den vorherigen Kapiteln beschrieben, mit seiner maximalen Taktfrequenz von 120 MHz konfiguriert. Die Bedingungen für diesen Versuch sind in der nachfolgenden Tabelle festgehalten:

Tabelle 5.2: Übersicht: Bedingungen für die Parametersuche

Bezeichnung	Wert
Tiva Taktfrequenz	120 MHz
Tiva Tigger Pin	A7
Infineon Taktfrequenz	144, 72 und 48 MHz
Anzahl der Add-Befehle	10 000
Parametersatz Durchläufe	100
MOSFET	IRLML2502Pdbf
Messgerät	PicoScope

Die genau Bestimmung der Glitchdauer ist sehr zeitaufwendig, da meist nicht klar ist, in welchem Bereich überhaupt zu suchen ist. Durch Rücksprache mit einem Experten von NXP lässt sich die Glitchdauer bei 144 MHz auf einen Bereich um 300 ns eingrenzen. Dieser Wert entspricht Erfahrungswerten und wird nicht weiter hinterfragt.

Versuch mit Taktfrequenz f_{CPU} =144 MHz

Zunächst muss die Anzahl an Takten, die das Tiva Board den Pin einschalten muss, bestimmt werden. Die Glitchdauer für die eingestellte CPU-Taktfrequenz beträgt ca. 300 ns. Wie aus der Gleichung 3.1 aus dem Abschnitt 3.1 bekannt ist, beträgt die Dauer eines Taktes ($t_{takt,tiva}$) bei der maximalen Taktfrequenz von 120 MHz 8,33 ns. Der mögliche Systemtaktbereich für eine Glitchdauer von 300 ns berechnet sich aus der Zeit, in welcher der MOSFET voll leitend ist (t_{glitch}) und der Zeit, in welcher der MOSFET angesteuert wird. ($t_{on,mosfet}$). Die untere Grenze für die t_{glitch}-Zeit ergibt sich durch das Gleichsetzen der Glitchdauer mit der t_{mosfet}-Zeit:

$$300\,\text{ns} = t_{mosfet} = 155{,}5\,\text{ns} + t_{glitch} = 144{,}5\,\text{ns} \tag{5.1}$$

Teilt man diese Zeit durch die Dauer eines Systemtaktes, ergibt sich die Anzahl an benötigten Systemtakten:

$$S_{anzahl,min} = \frac{144{,}5\,\text{ns}}{8{,}33\,\text{ns}} = \underline{\underline{17{,}35}} \tag{5.2}$$

Die obere Grenze ergibt sich unter der Annahme, dass der Glitch nur dann wirkt wenn der MOSFET voll leitend ist:

$$\begin{aligned} S_{anzahl,max} &= \frac{t_{glitch}}{t_{takt,tiva}} = \frac{300\,\text{ns}}{8{,}33\,\text{ns}} \\ &= \underline{\underline{36{,}014}}. \end{aligned} \tag{5.3}$$

Auswertung
Für den Versuch liegt der Bereich für die t_{glitch}-Dauer bei 0 bis 58-Systemtakten. Die Verzögerungszeit liegt bei 50, 60 und 70 Systemtakten. Sie gibt an, wie lange das Programm den Glitch verzögern soll nachdem das Triggersignal gekommen ist.

Der Versuchsaufbau liefert die folgenden Ergebnisse für eine CPU-Taktfrequenz von 144 MHz (siehe Abbildung 5.4). In dem gezeigten Bild wird nach t_{glitch} unterschieden, das heißt, dass die Versuche mit unterschiedlich Verzögerungszeiten, aber mit dem selben t_{glitch} aufaddiert werden.

Die Ergebnisse lassen sich in drei Bereiche unterteilen.

Der erste Bereich geht von 0 bis 18-Systemtakten. In diesem Abschnitt hat der injizierte Glitch keinerlei Auswirkungen auf die 10 000 Additionen ds Testprogramms. Die Rate der erfolgreich durchgeführten Durchläufe liegt in diesem Bereich bei 300 von 300.

Im zweiten Abschnitt (20 bis 22) ist die Glitchdauer lang genug, um sich auf die Additionen auszuwirken. Aber in diesem Bereich liegt die Anzahl der benötigten Neustarts[1] unterhalb der erfolgreichen Durchläufe. Die Fehlerwahrscheinlichkeit liegt in diesem Bereich bei ca. 2,55 %.

Der dritte Abschnitt geht von 25 bis 30-Systemtakten. In diesem Abschnitt ist der Mikrocontroller nicht mehr in der Lage die Additionen ohne Fehler auszuführen. Die Zurücksetzrate liegt hier bei ca. 75 % oder mehr. Auch die Fehlerwahrscheinlichkeit steigt im letzten Abschnitt weiter an und hat bei 30-Systemtakten einen Wert von 10 %.

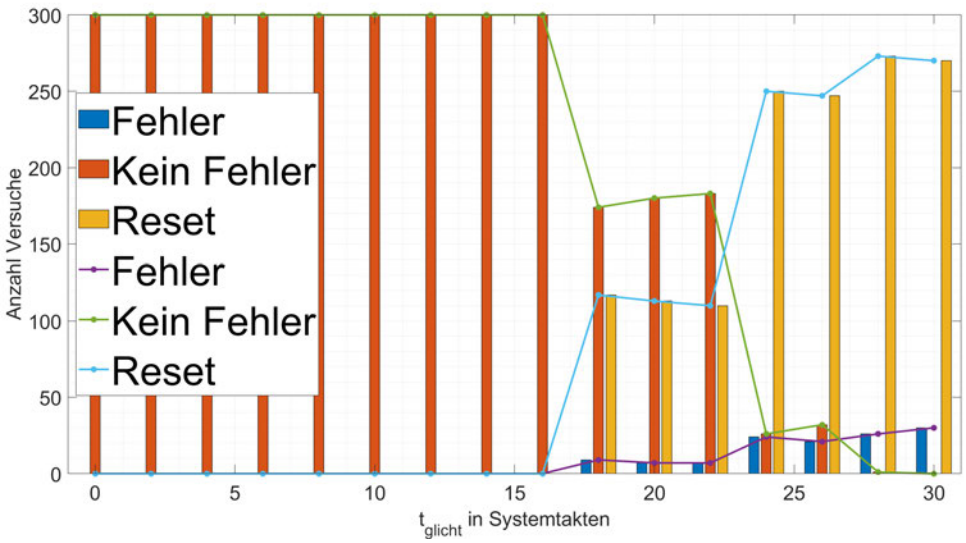


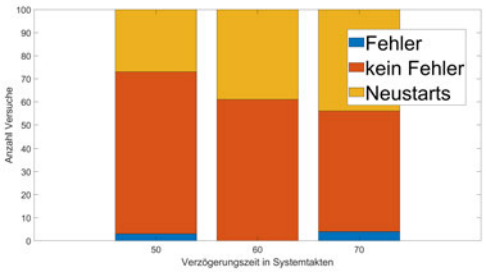
Abbildung 5.4: Ergebnisse der Parametersuche für 144 MHz

Außerdem ist in der Abbildung 5.4 zu erkennen, dass der zuvor genannte Erwartungswert von 300 ns (17-Systemtakte) den Bereich kennzeichnet, ab welchem der Glitch Auswirkungen auf die Testfunktion zeigt (in der Abbildung bei 18-Systemtakten).

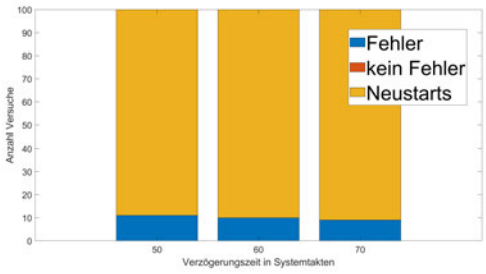
Für den Glitchangriff sind somit nur die beiden hinteren Abschnitte interessant. Aus

[1] Ein Neustart ist nötig, wenn sich der XMC4700 Relax Kit Lite während den Instruktionen aufhängt.

diesem Grund werden zwei Glitchzeiten t_{glitch} aus diesen beiden Abschnitten näher betrachtet. Aus dem Abschnitt von 20 bis 22-Systemtakten ist es die Glitchdauer von 22-Systemtakten (siehe Abbildung 5.5a). Im anderen Bereich wird die Glitchdauer von 30-Systemtakten genauer untersucht (siehe Abbildung 5.5b). Die Auswahlkriterien für diese beiden Zeiten sind die Fehlerwahrscheinlichkeit und das Verhältnis von fehlerfreien Durchgängen zu benötigten Neustarts.



(a) Detailansicht: Parametersuche für $t_{glitch} = 22$



(b) Detailansicht: Parametersuche für $t_{glitch} = 30$

Abbildung 5.5: Detailansicht: Parametersuche für $t_{glitch} = 22$ und 30 Systemtakten bei $f_{CPU} = 144\,\text{MHz}$

Es ist zu erkennen, dass bei der Glitchdauer von 22-Systemtakten die Verzögerungszeit eine Auswirkung auf die Fehlerwahrscheinlichkeit hat.
In Abbildung 5.5a weisen die beiden Verzögerungszeiten von 50 und 70 Fehler in der Berechnung auf. Dies ist bei einer Verzögerungsdauer von 60-Systemtakten nicht der Fall. Außerdem ist erkennbar, dass die Anzahl an fehlerlosen Durchläufen mit zunehmender Verzögerungszeit abnimmt[2].
Bei der Detailbetrachtung der Glitchdauer von 30-Systemtakten fällt auf, dass die Fehlerwahrscheinlichkeit über die verschiedenen Verzögerungszeiten relativ konstant bleibt. Außerdem kommt es bei dieser Glitchdauer zu keinem fehlerfreien Durchlauf des Testprogramms (vgl. Abbildung 5.5b).
Die Betrachtung eines Durchgangs für t_{glitch} von 20-Systemtakten zeigt, dass die Dauer der verminderten VDDC bei ca. 320 ns liegt (vgl. Abbildung 5.6 rote Linien).

[2]Keine Aussage sondern eine Annahme. Die Betrachtung von 3 Verzögerungszeiten ist nicht aussagekräftig genug.

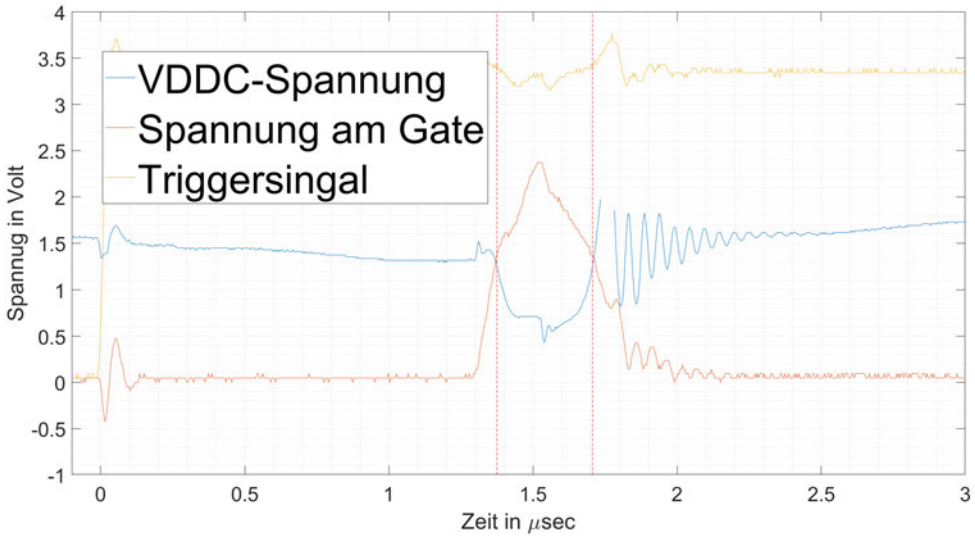


Abbildung 5.6: Auswirkung eines Versuchs mit t_{glitch} = 20-Systemtakten

Versuch mit Taktfrequenz f_{CPU} =72 MHz

Für diese CPU-Taktfrequenz liegen keine Erwartungswerte vor. Somit wird das Programm auf eine maximale Glitchdauer t_{glitch} von 200-Systemtakten gestellt und um die Abbruchbedingung, dass es nicht weniger als 50 % erfolgreiche Durchläufe pro Versuch geben darf.

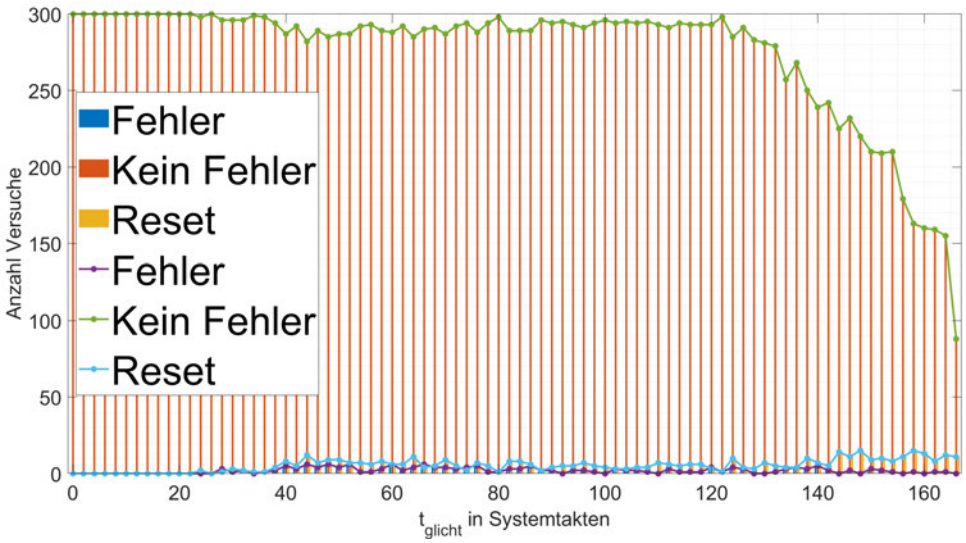


Abbildung 5.7: Ergebnisse der Parametersuche für 72 MHz

In der Abbildung 5.7 ist, wie in dem Versuch für 144 MHz, der Bereich 0 bis 20 ohne Auswirkung des Glitches. Ab 20-Systemtakten steigen die Werte für die Fehler und die Neustarts leicht an und bleiben relativ konstant auf diesem Niveau. Bei ca. 140-Systemtaken fällt die Fehlerwahrscheinlichkeit auf nahezu 0 % und die Anzahl an Neustarts steigt leicht.
Anders als in der Abbildung 5.4 liegt die Anzahl der Neustarts unterhalb der Anzahl der fehlerfreien Durchläufe. Der Abfall der fehlerlosen Durchgänge des Testprogramms lässt sich mit internen Schutzmaßnahmen des XMC4700 Relax Kite Lite erklären. Diese Überwachen bspw. die Stromzufuhr. Liegt diese für eine zu lange Zeit über oder unter dem normalen Wert, geht der Mikrocontroller in einen Schutzzustand, z.B. Neustart.
Die erste Anzahl an Systemtakten für t_{glitch}, bei welcher der Glitch die höchste Fehlerwahrscheinlichkeit erreicht liegt bei 44. Dies entspricht etwa dem 2-fachen des Wertes aus dem Versuch für 144 MHz.

Versuch mit Taktfrequenz f_{CPU} =48 MHz

Für diesen Versuch liegen keine Erfahrungswerte für die Glitchzeit t_{glitch} vor. Aus diesem Grund wird die Bedingung eingeführt, dass es nicht mehr als 90 % Neustarts oder andere Fehler geben darf.

Auswertung

Der Versuch liefert die folgenden Ergebnisse:

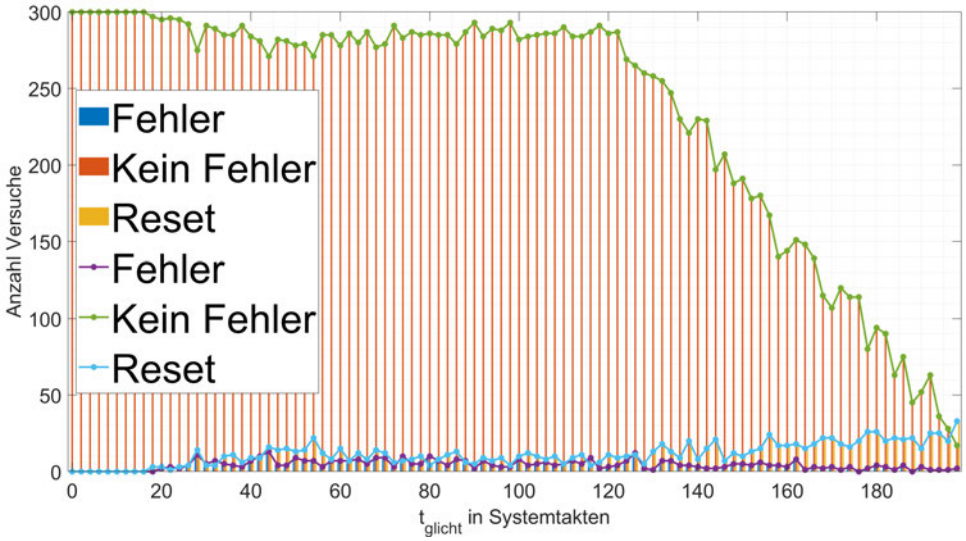


Abbildung 5.8: Ergebnisse der Parametersuche für 48 MHz

Auch bei dieser Taktfrequenz passieren im vorderen Systemtaktbereich (0 bis 20) für die Glitchdauer t_{glitch} keine Fehler. Ab 20-System takten steigt sowohl die Fehlerrate, als auch die Neustartrate an. Die Fehlerrate erreicht bei 44-Systemtakten ihr Maximum. Außerdem ist zu sehen, dass ab einer bestimmten Glitchdauer (hier bei ca. 120-Systemtakten) zu einem rapiden Verringerung der fehlerlosen Durchgänge kommt. In demselben Zeitraum steigen jedoch nicht die Raten für einen Fehler oder einen Neustart an. Der Fall der Anzahl von fehlerlosen Durchgängen ist damit zu erklären, dass das Infineon Relax Lite Kit über eigenes Überwachungsequipment verfügt, so kann z.B. der Stromregler überwacht werden. Sollte bei der Überwachung ein fehlerhafter Zustand entdeckt werden geht das Board in einen Fehlerzustand. Dieser Fehlerzustand wird in dem Versuch nicht berücksichtigt und führt zu einem Rückgang der fehlerfreien Durchgänge ohne ein Anstieg der anderen beiden Werte.

Fazit: Parametersuche

Während der benutzbare Bereich, in welchem der Glitch zu seinem Fehler in der Berechnung führt und die Neustartrate nicht zu groß ist, bei den CPU-Frequenzen 72 MHz

und 48 MHz relativ groß ist (ca. 100-Systemtakte) ist er bei einer CPU-Frequenz von 144 MHz nur 6-Systemtakte groß. Für den folgenden Versuch des Glicht-Angriffs eignet sich nach der Betrachtung der Ergebnisse eine Glitchdauer t_{glitch} von 20-Systemtakten für eine CPU-Taktfrequenz von 144 MHz am besten. Bei den anderen beiden Frequenzen liegt die maximale Fehlerwahrscheinlichkeit bei einer Glitchdauer von 44-Systemtakten. Der nachfolgende Glitch-Angriff wird auf die maximale CPU-Taktfrequenz von 144 MHz angewendet.

5.2 Glitch-Angriff

5.2.1 Aufbau

Der Aufbau des Glitch-Angriff ist derselbe wie bei der Parametersuche (siehe Abbildung 5.1). Auf dem PC läuft bei diesem Versuch nicht mehr die Parametersuche, sondern der Glitch wird mit einem festen Parametersatz ausgeführt. Dieser Parametersatz ist in der Parametersuche bestimmt worden. Außerdem wird das Programm auf dem XMC4700 Relax Kit Lite um die Funktionalität der Signatur mittels RSA ergänzt.

Bestimmung der RSA Komponenten

Die Bestimmung der RSA Komponenten, wie z.B. die beiden geheimen Parameter p un q geschieht mittels eines Onlinetools [Gen20]. Auf der Seite kann die Schlüssellänge eingestellt werden und anschießend werden automatisch die Zahlen p, q, n und d in Abhängigkeit von dem gewählten e berechnet/ausgesucht. Für eine Schlüssellänge von 256 bit sehen die Parameter folgendermaßen aus:

Tabelle 5.3: Parameterübersicht: 256 bit-RSA

Parameter	1024 bit in [hex]
p	0xf63d785fbbef6398e91018b19c33c6c5
q	0x8a9ac3d1e77f8cd7a1453799de97669d
n	0x85520038be7273983f9398ec2260648613a125f4f6bfafaae1ad10cf812664d1
e	0x10001
d	0x48f05efcac7d136ef5417c059c9d1e0146093393506eb9e0624eb91dc2187f91

Die Werte für 512 und 1024 bit können im Anhang nachgelesen werden.
Diese Werte werden fest in das Programm einprogrammiert und können nicht zur Laufzeit geändert werden. Dies ermöglicht es Leistung vom DUT einzusparen, weil hierdurch bestimmte Werte nicht während der Laufzeit berechnet werden müssen (z.B. der Teil des öffentlichen Schlüssels n).
Auch werden, durch die Verwendung des Online-Tools [Gen20], mögliche Fehlerquellen, wie z.B. die Auswahl der Primzahlen, eliminiert.

5.2.2 Signatur mittels RSA auf dem XMC4700 Relax Kit Lite

Für den RSA werden Zahlen der Größenordnung 256 bis 1024-bit benötigt. Damit der Mikrocontroller so große Zahlen speichern und verwenden kann, wird die Flint-Bibliothek verwendet. Diese ist speziell für größere Zahlen ausgelegt (Langzahlarithmetik). Außerdem liefert diese, für die Signatur hilfreiche, Funktionen wie z. B. $xgcd\_l$, welche den GGT und die multiplikativen Inversen der Modulorechnung berechnet.

Ein Ausschnitt aus der CRT-Implementierung ist nachfolgend in Listing 5.1 dargestellt. Es ist in dem Programmcode zu sehen, dass die Berechnung des CRT in drei Abschnitte eingeteilt ist. Dies ermöglicht es nur einen bestimmten Teil der CRT-Berechnung anzugreifen. Das ist eine Grundvoraussetzung für den Bellcore Angriff (vgl. Abschnitt 2.5).

Listing 5.1: Ausschnitt aus Chinesischer Restsatz in C

```
0         ...
          //second part
2         dec_l(prime2);
          mod_l(exponent, prime2, exp_prime2);
4         inc_l(prime2);
          if (sign_v < 0) sub_l(divisor, v, v);
6         squareAndMultiply(base, &exp_prime2, prime2, &result_prime2);
          ZEROCLINT_K(exp_prime2);
8
          //first part + second part
10        mul_l(prime1, &result_prime2, &result_prime2);
          mul_l(prime2, &result_prime1, &result_prime1);
12
          mmul_l(u, &result_prime2, &result_prime2, divisor);
14        mmul_l(v, &result_prime1, &result_prime1, divisor);

16        madd_l(&result_prime1, &result_prime2, result, divisor);
  }
```

Der weitere Programmcode befindet im Anhang.

5.2.3 Ergänzung Kommunikation für den Glitch-Angriff

Die bereits vorhandene Kommunikation aus Tabelle 5.1 muss um die Möglichkeit einer Signatur erweitert werden. Hierzu sendet der PC eine festgelegte Nachricht an das DUT und wartet auf die Antwort. Die Tabelle der Kommunikationsbefehle sieht nach der Erweiterung folgendermaßen aus:

Tabelle 5.4: Erweiterung: Kommunikationsbefehle

Befehl	Bedeutung
/P:para1:para2:	Glitchparameter [max. Zeichen: 20]
	para1 = Glitchdauer
	para2 = Glitchverzögerungszeit
/S	Startbefehl
/R	Rücksetzbefehl
/R:para1:	Signatur berechnen von para1
	para1 = Nachricht
	[max. Zeichen: Schlüssellänge+10]
/W	Verbindung prüfen
/B:para1:para2:	Bellcore ausführen
	para1 = fehlerhafte Signatur
	para2 = öffentlicher Schlüssel

5.2.4 Verifikation der RSA-Implementierung

Die Verifikation der RSA-Implementierung beschränkt sich auf die Berechnung der Signatur. Hierzu werden zunächst für die drei Schlüssellängen (256, 512 und 1024 bit) jeweils drei pseudo-zufällige Nachrichten generiert (siehe Unterabschnitt A.1.5). Hierbei handelt es sich jeweils um eine Nachricht mit einem Zeichen, eine Nachricht mit der Schlüssellänge (256, 512 und 1024) und eine Nachricht mit einer Länge zwischen den beiden Grenzen.

Diese Nachrichten werden auf dem Infineon XMC4700 Relax Kit Lite signiert und die berechnete Signatur mit der am Computer berechneten Signatur verglichen. Die am Computer berechnete Signatur wird mittels [Cry20] ermittelt, hierzu müssen die Nachrichten zunächst noch in die hexadezimale Darstellung überführt werden. Die für den RSA notwendigen Komponenten (p, q usw.) sind aus dem Abschnitt 5.2.1 übernommen.

Für die Schlüssellänge 256 bit ergeben sich die folgenden Signaturen:

Tabelle 5.5: Vergleich der Signaturen für die Schlüssellänge 256 bit

Gerät	Zeichen der Nachricht			Signatur
	Min.	Zufällig	Max.	
Computer	X			2B79EC709ADCEE9E3D885F 4269917C75148B1BD2EF03E 6942E074DAC075954E8
Relax Kit Lite	X			2b79ec709adcee9e3d885f42699 17c75148b1bd2ef03e6942e074 dac075954e8
Computer		X		601F138D6D0010B16D16BF 7D703419DE448145C85AD4 88D5FFA3CDEA8CDCFDB3
Relax Kit Lite		X		601f138d6d0010b16d16bf7d703 419de448145c85ad488d5ffa3c dea8cdcfdb3
Computer			X	6D920C44451D3DF6CD0578 1386F0B019D6A02636C8E2 A2939B894CBE5820136A
Relax Kit Lite			X	6d920c44451d3df6cd05781386f 0b019d6a02636c8e2a2939b894 cbe5820136a

Es ist zu erkennen, dass die Signaturen vom PC und dem XMC4700 Relax Kit Lite bei allen versuchten Nachrichtenlängen gleich sind. Somit ist bewiesen, dass das Infineon Board die Signatur richtig berechnet. Die Nachrichten für die anderen Schlüssellänge sind im Anhang(siehe Anhang Unterabschnitt A.3.1).

5.2.5 Durchführung eines Glitch-Angriffs

Der Glitch wird für die maximal einstellbare CPU-Taktfrequenz von 144 MHz durchgeführt. Hierbei beträgt die Glitchdauer t_{glitch} 20-Systemtakte.
Zur besseren Bestimmung des Zeitpunkts, an welchem der Glitch wirken muss, wird weiterhin der Triggerpin verwendet. Wenn im Programm der erste Teil der Signatur berechnet wird, gibt der Triggerpin den Startimpulse für den Glitch.

Für die Signierung wird die CRT-Implementierung verwendet, da diese mittels Bellcore angegriffen werden kann.

Tabelle 5.6: Bedingungen: Glicht Angriff

Parameter	Wert
CPU-Taktfrequenz	144 MHz
t_{glitch}	20-Systemtakte
Durchgeführte Versuche	100

Die Nachrichten für die einzelnen Schlüssellänge (256, 512 und 1024) werden per Pseudozufall generiert. Die Generierung folgt den folgenden Bedingungen:

- Die Nachricht sollte keine Leerzeichen enthalten.
- Die Nachricht darf kein „:“ beinhalten, da dies als Trennzeichen bei der Kommunikation dient.
- Die Nachricht darf keine vordefinierten Zeichenfolgen wie „\n“ beinhalten.

Diese Einschränkungen sind nicht allgemeingültig. Sie beziehen sich nur auf den Versuch und sind z.B. durch die verwendeten IDEs bedingt.

Schlüssellänge 256 bit

Die Signatur wird auf dem XMC4700 Relax Kit Lite mittels dem RSA mit der Schlüssellänge 256 bit erzeugt. Aus den genannten Bedingungen ergibt sich die Nachricht M:

$$M = \text{Z29Qb3SY;\{nUdY?o‘}$$

Schlüssellänge 512 bit

Bei diesem Versuch wird ein 512 bit-RSA verwendet. Die Erzeugung der Nachricht M folgt den bekannten Bedingungen. Es ergibt sich die Nachricht M:

$$M = \text{Eul3I\_pEtWL1z6XwA}$$

Schlüssellänge 1024 bit

Die Ergebnisse des Bellcore Angriffs auf ein RSA mit der Schlüssellänge 1024 bit befinden sich im Anhang. Diese werden aus Gründen der Übersichtlichkeit an dieser Stelle nicht näher erläutert.

5.2.6 Fehlerhafte Signaturen

Schlüssellänge 256 bit

Die nicht fehlerhaft Signatur der Nachricht M ist:
„0x31f2cfcff462e45f0aba648089978788fcb022c7f651718b95974e897c1b02aa“

In der Tabelle 5.7 sind die fehlerhaften Signaturen festgehalten, welche während des Glitch Angriffs aufgetreten sind.

Tabelle 5.7: Fehlerhafte Signaturen für einen 256 bit-RSA

Nummer	fehlerhaft Signatur in [hex]
1	700bd0e30d3532d489b161b6201e7e014c4a49c776e4b1401968d5e79fc7e11c
2	1cc3484b53caae572a058a1099aef8b00a0a223a9fec6ca0041343b1d78a0184

Die fehlerhaften Signaturen können sich von Versuch zu Versuch unterscheiden. Dies liegt daran, dass nicht immer dieselben Versuchsbedingungen vorliegen. Auch ist in Tabelle 5.7 zu erkennen, dass es bei dem selben Versuch zu zwei unterschiedlichen, fehlerhaften Signaturen kommt. Das ist der Fall, weil der injizierte Glitch die Berechnung zum Beispiel nicht genau an derselben Stelle stört (kleine bauteilbedingte Abweichung in der Verzögerungszeit und/oder Glitchdauer).

Schlüssellänge 512 bit

Die fehlerfreie Signatur für einen RSA mit der Schlüssellänge 512 bit lautet:
„0x5b55e671d9d2f980b056b9d1a7ae9d2d385383ea16e4682b86a0a645a53e40bf5b6f750913
ffe1b28bee2922d5e2f8b1d0284cf69bbb7a544ca4056181ead483“

Während des Glitch Angriffs kommt es zu fehlerhaften Berechnungen der Signatur. Diese fehlerhaften Signaturen sind in Tabelle 5.8 zu sehen.

Tabelle 5.8: Fehlerhafte Signaturen für einen 512 bit-RSA

Nummer	fehlerhafte Signatur in [hex]
1	5d6a990a917fd8a2bfa2fa7cb96af329eb2375c0fc6d8e330c59f157e2a3ff6b2b d78808773c18cac0caf944681fea9295c84b8330e5a8f13f82a68c77cbceb1
2	642bcff9997c6f1c7095b00dfb6ded033ff75e2dc43f666055c8a06054470d3053 4912fcca5c38cef4f8429135e07eca11735fddfb70f8e959d6949855cfb5b3
3	54f3ddb2ef5610f7b3933c3285a37349d5ee6939f638c5e2c0a15d6d8a0d29c823 f5468f21fa7a4db99fb776a0642d696f6947b6282da9a0733f892c99585078
4	179c5218219c9152acda4fcc614df1ddf1f6b508c72e87c2d085600579294b4c3b 9e4369deb5024046e6332e81e98e2c28dc28977b8f014a241018c32cff57fd
5	fdf2faa883f60e6d4d4532eb80be24f09e0fa3d12e50dfeb496db8e0d71149d139 b050690fb02442eb4b83fafbe2d0127d8125f884fe853b195ba43b596e3d0
6	875f8be4446ea3be7322943b1a6b7dd672598abfe4251bc8aea352f948460c1471 daec199b7349835b1dff1a7d4d1f1aeca05f63c0ca0fb31fcca31412cc24ea

Auswertung

Die durch den Glitch verursachten, fehlerhaften Signaturen werden einem Bellcore-Angriff unterzogen. Dieser liefert die beiden geheimen Parameter p und q für den Fall, dass der Glitch die Berechnung der Signatur richtig gestört hat.
Für den Bellcore-Angriff ist auf einem weiteren XMC4700 Relax Kit Lite (unmodifiziert) derselbe RSA implementiert. Das Programm ist um die Möglichkeit, einen Bellcore-Angriff durchzuführen erweitert worden (siehe Unterabschnitt A.2.2 oder Unterabschnitt A.1.6). Der hierbei verwendete Bellcore-Angriff nutzt eine fehlerhafte Signatur $\hat{Sig}$ und die original Nachricht M.
Der Ablauf des Bellcore Angriffs ist folgender:

1 Senden der Nachricht an das XMC4700 Relax Kit Lite.

2 Senden der fehlerhaften Signatur und des öffentlichen Schlüssels an das XMC4700 Relax Kit Lite.

3 Anwendung des Bellcore Angriffs auf dem XMC4700 Relax Lite Kit.

4 Berechnung des neuen privaten Schlüssels aus den neuen p und q.

Anschließend, an diese vier Schritte, wird die Signatur überprüft. Hierzu wird die Nachricht M fünf Mal signiert und mit der original Signatur verglichen. Sind die Signaturen identisch war der Bellcore Angriff erfolgreich.
Für die in Tabelle 5.7, Tabelle 5.8 und im Anhang gezeigten fehlerhaften Signaturen liefert der Bellcore-Angriff folgende Ergebnisse:

Tabelle 5.9: Ergebnisse Bellcore-Angriff für 256 bit- und 512 bit-RSA

Schlüssellänge	Signatur Nr.	Bellcore-Angriff erf.	Bellcore-Angriff nicht erf.
256	1	X	
256	2	X	
256	3[1]		X
512	1	X	
512	2	X	
512	3	X	
512	4	X	
512	5	X	
512	6	X	
512	7[1]		X
1024	1	X	
1024	2	X	
1024	3	X	
1024	4	X	
1024	5	X	
1024	6[1]		X

In der Tabelle 5.9 ist zu erkennen, dass alle im Versuch, durch den Glitch verursachten, fehlerhaften Signaturen für einen Bellcore-Angriff und der damit verbundenen Bestimmung des privaten Schlüssels geeignet sind. Nur die mit der 1 gekennzeichneten Signaturen konnten nicht für einen Bellcore-Angriff verwendet werden. Dies liegt daran, dass sie nicht durch einen Fehler in der Berechnung der Signatur entstanden sind. Sie wurden nachträglich per Hand modifiziert.

[1] Per Hand erzeugte fehlerhafte Signaturen (Tauschen eines Zeichens)

Beispielhaft wurden die, für die Schlüssellänge 256 bit, durch den Bellcore Angriff berechneten Parameter p und q ausgegeben. Hierbei ergaben sich die Parameter:

Tabelle 5.10: Geheime Parameter durch Bellcore Angriff

Parameter	Wert
p	f63d785fbbef6398e91018b19c33c6c5
q	8a9ac3d1e77f8cd7a1453799de97669d

Diese beiden Parameter sind, bis auf das sie vertauscht sind, identisch mit den originalen Parametern (vgl. Tabelle A.1). Die Tatsache, dass sie vertauscht sind, liegt nur an der Zuordnung im Programm und ist somit zu vernachlässigen.

6 Fazit

Die Ergebnisse zeigen, dass eine Glitchdauer von ca. 320 ns bei einer f_{CPU} von 144 MHz zu einer falschen Teilberechnung der Signatur führt. Diese fehlerhafte Signatur kann von einem Bellcore Angriff genutzt werden, um einen der beiden privaten Parameter des verwendeten RSA zu berechnen. Ist dies geschehen, ist es recht einfach den zweiten privaten Parameter zu bestimmen und damit alle Bestandteile des privaten Schlüssels zu berechnen.
Jedoch zeigen die Ergebnisse auch, dass die Wahrscheinlichkeit eines erfolgreichen Glitches bei dem verwendeten Setup sehr gering ist. Das ist im Prinzip nicht von signifikanten Nachteil, da für den Bellcore Angriff nur eine fehlerhafte Signatur benötigt wird, aber für ein Demoprojekt wäre eine höhere Erfolgswahrscheinlichkeit wünschenswert.
Für den Erfolg des Glitchangriffs ist das Parameterpaar bzw. die tatsächliche Glitchdauer von enormer Wichtigkeit. Während der Thesis hat sich gezeigt, dass die Bestimmung der Parameter der aufwendigste Teil des ganzen Versuchs ist. Es muss ein MOSFET gewählt werden, welcher schnell und präzise genug schalten kann. Außerdem erweist sich die Erzeugung des zeitlich richtigen Triggersignales als technisch schwer umsetzbar. Es wird ein entsprechend passender Generator benötigt, der das Signal in der benötigten Zeit genieren kann. Sollten diese beiden Bedingungen erfüllt sein, ist die Suche der richtigen Parameter noch immer mit einem großen zeitlichen Aufwand verbunden. Ohne ein gewisses Vorwissen, wo der Zeitbereich liegen könnte, kann die Bestimmung der Parameter mehrere Stunden, wenn nicht sogar Tage dauern.
Während der Vorbereitung ist aufgefallen, dass die Auswahl des MOSFET für diesen Versuch sehr wichtig ist. Der zunächst ausprobierte MOSFET „BUZ11“ besaß eine zu große Schaltzeit und zeigte keine erfolgreichen Ergebnisse.
Auch zeigt die Notwendigkeit den Chip zuvor modifizieren zu müssen, dass die meisten Chips heutzutage gegen einen solchen „primitiven“ Angriff geschützt sind. Dies kann in Form vom Hardware, wie bspw. die Kondensatoren oder eine Überwachung des benötigten Stroms, oder in Form von Software, wie die Überwachung bestimmter Speicherbereiche, realisiert werden.

Als Alternative für das verwendete Tiva Board würde sich ein Board mit höherer und genauerer Taktzeit eignen. Eine weitere Alternative wäre ein Funktionsgenerator. Dieser kann das Triggersignal generieren oder wie in [BFP19] direkt als Versorgungsspannung für den Chip dienen.

Ausblick

Die Programme sind momentan aus Zeitgründen noch als einzelne Programme und Skripte vorhanden. Als Ausblick würde sich anbieten, diese in einem Projekt zu vereinen und eine Benutzeroberfläche zu entwerfen. Außerdem könnte ein noch schneller schaltender MOSFET verwendet werden. Dies würde es ermöglichen die Parameter noch präziser zu Bestimmen und die Wahrscheinlichkeit eines erfolgreichen Glitches erhöhen.

Literaturverzeichnis

[BBKN12] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. Proceedings of the IEEE, 100(11):3056–3076, 2012.

[BDL96] Dan Boneh, Richard A. Demillo, and Richard J. Lipton. On the importance of checking computations (extended abstract), 1996.

[BDL01] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of eliminating errors in cryptographic computations. Journal of cryptology, 14(2):101–119, 2001. seen: 06.11.2019.

[BECN+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE, 94(2):370–382, 2006.

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the glitch: Optimizing voltage fault injection attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 199–224, 2019.

[Cry20] Cryptomathic. Rsa calculator. http://extranet.cryptomathic.com/rsacalc/index, mar 2020.

[Gen20] RSA Key Generation. Rsa key generator online. https://www.mobilefish.com/services/rsa_key_generation/rsa_key_generation.php, mar 2020.

[Gir06] Christophe Giraud. An rsa implementation resistant to fault attacks and to simple power analysis. IEEE Transactions on computers, 55(9):1116–1120, 2006.

[HTI97] Mei-Chen Hsueh, Timothy K Tsai, and Ravishankar K Iyer. Fault injection techniques and tools. Computer, 30(4):75–82, 1997.

[Ins] Texas Instruments. Tiva c series tm4c1294ncpdt microcontroller data sheet (rev. b). http://www.ti.com/lit/ds/symlink/tm4c1294ncpdt.pdf. seen: 21.03.2020.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Annual International Cryptology Conference, pages 388–397. Springer, 1999.

[KK99] Oliver Kömmerling and Markus G Kuhn. Design principles for tamper-resistant smartcard processors. Smartcard, 99:9–20, 1999.

[Len96] Arjen K Lenstra. Memo on rsa signature generation in the presence of faults. Technical report, 1996.

[Rec14] International Rectifir. Irlml2502pbf Datasheet. https://www.infineon.com/dgdl/irlml2502pbf.pdf?fileId=5546d462533600a401535668048e2606, 2014. seen:31.1.2020.

[Rel16] Board User Manual XMC4700 XMC4800 Relax Kit Series. https://www.infineon.com/dgdl/Infineon-Board_User_Manual_XMC4700_XMC4800_Relax_Kit_Series-UM-v01_02-EN.pdf?fileId=5546d46250cc1fdf01513f8e052d07fc, Infineon Technologies AG, 81726 Muinch, Germany, 2016. seen: 07.11.2019.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21(2):120–126, February 1978. seen: 05.11.2019.

[Sca09] Gerd Scarbata. Synthese und Analyse digitaler Schaltungen. Walter de Gruyter, 2009.

[Sko05] Sergei Petrovich Skorobogatov. Semi-invasive attacks: a new approach to hardware security analysis. 2005.

[Sma16] Nigel P Smart. Cryptography made simple, volume 481. Springer, 2016.

[SSAQ02] David Samyde, Sergei Skorobogatov, Ross Anderson, and J-J Quisquater. On a new way to read data from memory. In First International IEEE Security in Storage Workshop, 2002. Proceedings., pages 65–69. IEEE, 2002.

[XMC18] XMC4700 / XMC4800 Datasheet. https://www.infineon.com/dgdl/Infineon-XMC4700-XMC4800-DS-v, Infineon Technologies AG, 81726 Muinch, Germany, 2018. seen: 27.01.2020.

[ZDCR14] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Bruno Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pages 130–135. IEEE, 2014.

A Anhang

A.1 Programmcode

A.1.1 RSA-Implementierung

Square And Multiply

Der Programmcode für die „Square and Multiply“-Implementierung befindet sich auf der beigefügten CD.

Chinesischer Restsatz

Der Programmcode für die CRT-Variante befindet sich auf der beigefügten CD.

A.1.2 Bestimmung der Ein- und Ausschaltzeiten eines Pins

Der Programmcode für die Bestimmung der Ein- und Ausschaltzeiten eines Pins befindet sich auf der beigefügten CD.

A.1.3 Bestimmung der Instruktionsdauer

Der Programmcode für die Bestimmung der Instruktionsdauer befindet sich auf der beigefügten CD.

A.1.4 Bestimmung der Glitchzeiten

A.1.5 Generierung der pseudo-zufälligen Nachrichten

Der Programmcode für die Generierung der pseudo-zufälligen Zahlen für die Signatur befindet sich auf der beigefügten CD.

A.1.6 Bellcore Angriff

Glitch Angriff (Tiva)

Der Programmcode für den Glitch Angriff für das Tiva Board befindet sich auf der beigefügten CD.

Glitch Angriff (Infineon)

Der Programmcode für den Glitch Angriff für das Infineon Board befindet sich auf der beigefügten CD.

Glitch Angriff (Python)

Der Programmcode für den Glitch Angriff für den PC befindet sich auf der beigefügten CD.

Schlüsselberechnung (Infineon)

Der Programmcode für die Schlüsselberechnung für das Infineon Board befindet sich auf der beigefügten CD.

Schlüsselberechnung (Python)

Der Programmcode für die Schlüsselberechnung für den PC befindet sich auf der beigefügten CD.

A.1.7 Parametersuche

Parametersuche (Tiva)

Der Programmcode für die Parametersuche auf dem Tiva Board befindet sich auf der beigefügten CD.

Parametersuche (Infineon)

Der Programmcode für die Parametersuche auf dem Infineon Board befindet sich auf der beigefügten CD.

Parametersuche (Python)

Der Programmcode für die Parametersuche auf dem PC befindet sich auf der beigefügten CD.

A.2 Programmablaufpläne

A.2.1 Parametersuche

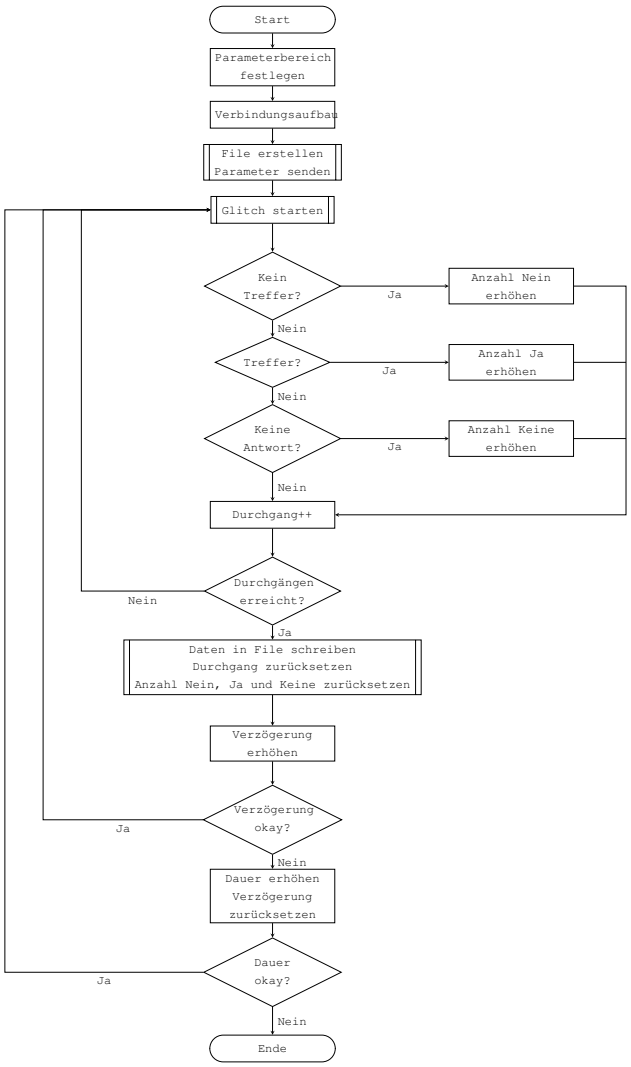
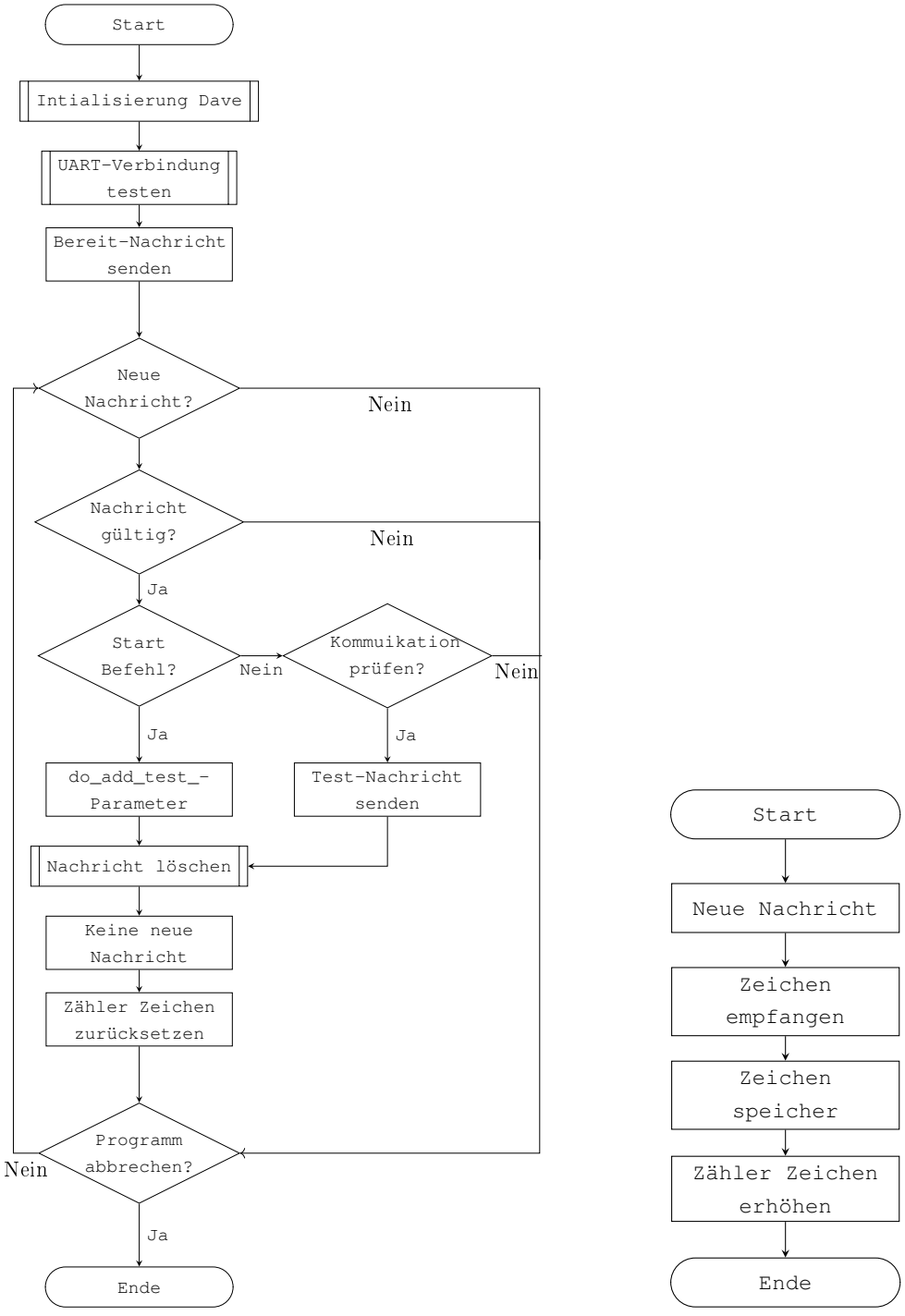
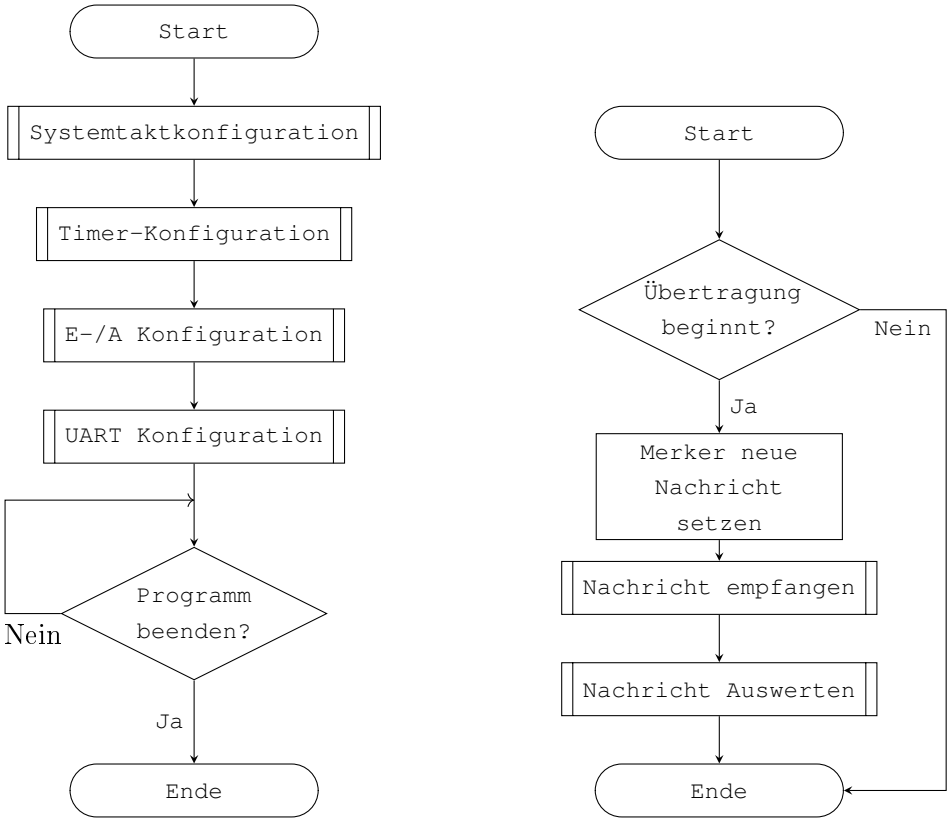


Abbildung A.1: Programmablaufplan: Parametersuche (Computer)



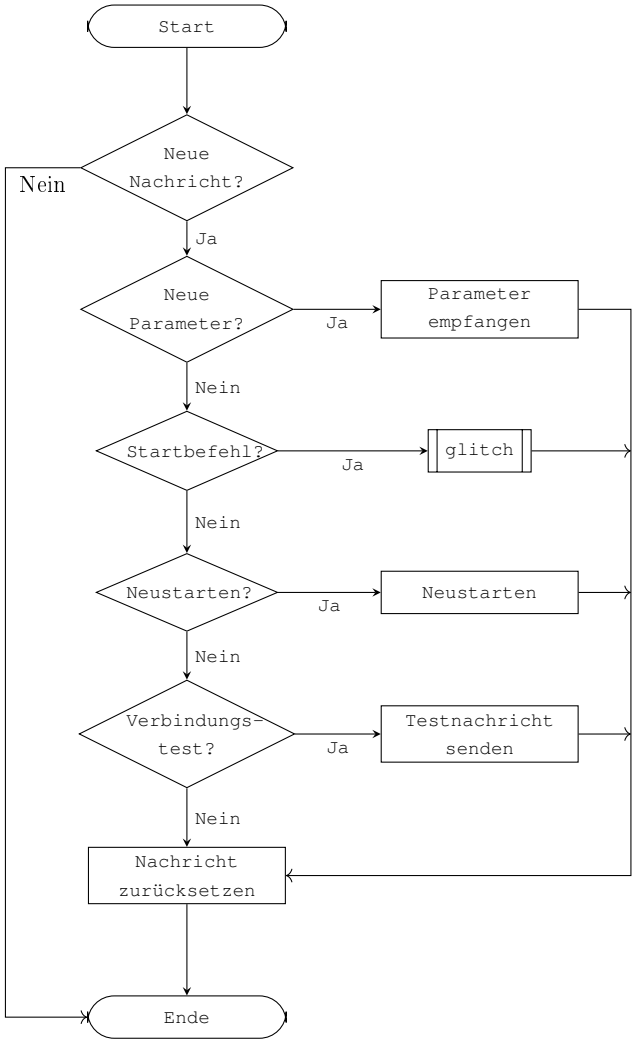
(a) Main (b) Interrupt

Abbildung A.2: Programmablaufplan: Parametersuche (XMC4700 Relax Kit Lite)



(a) Main (b) Interrupt

Abbildung A.3: Programmablaufplan: Parametersuche (Tiva Board)



(a) Auswertung der Kommunikation

Abbildung A.4: Programmablaufplan: Parametersuche (Tiva Board)

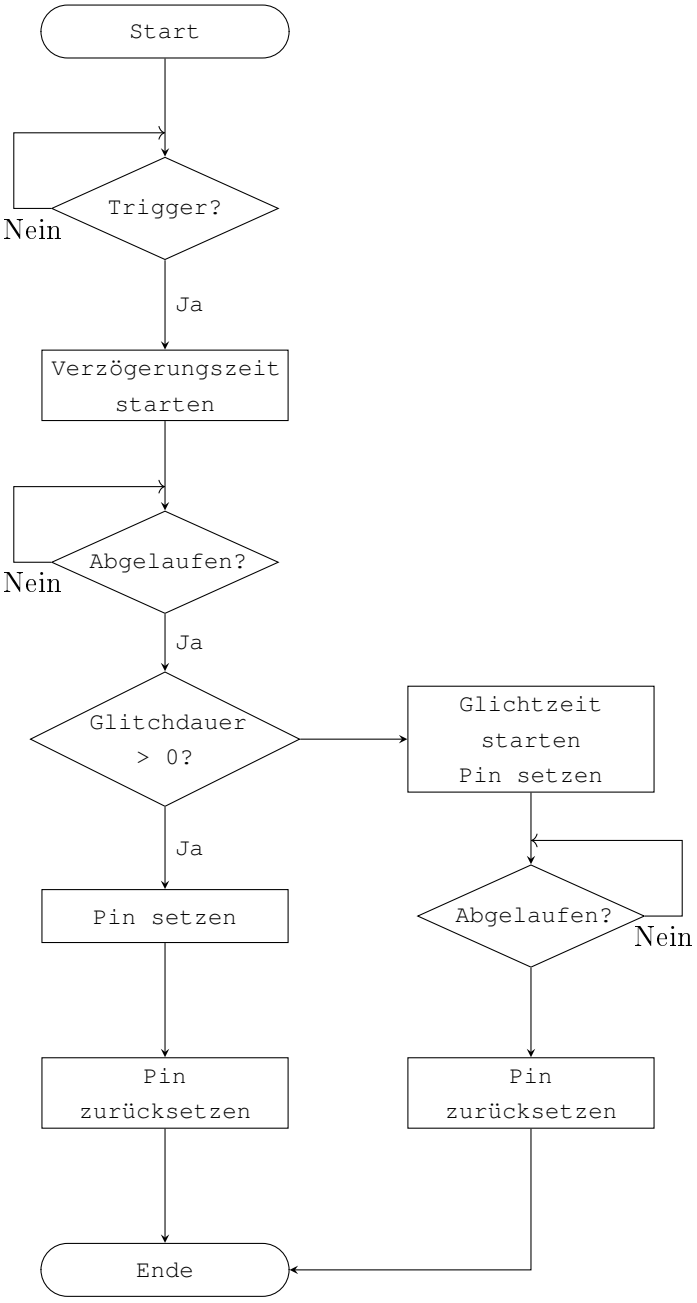


Abbildung A.5: Programmablaufplan: Glitch (Tiva Board)

A.2.2 Bellcore-Angriff

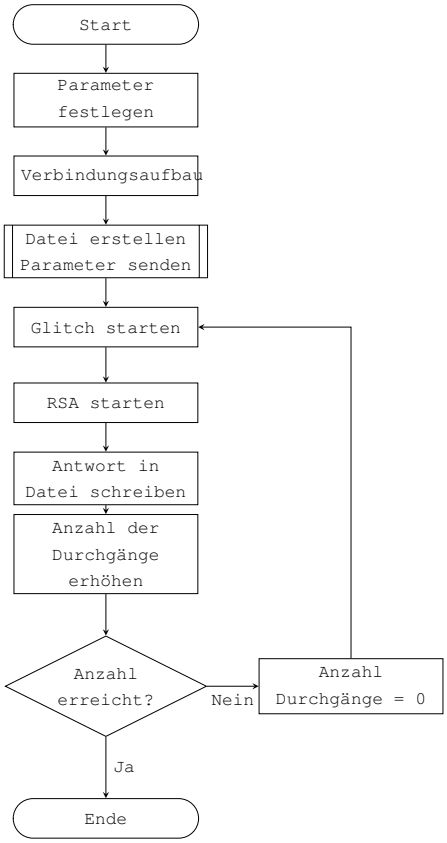
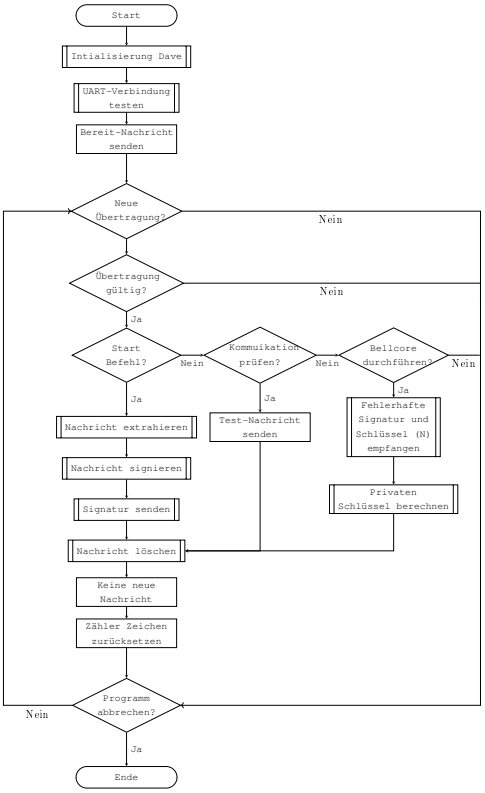


Abbildung A.6: Programmablaufplan: Bellcore Angriff (PC)



(a) Main (Modifiziert)

Abbildung A.7: Programmablaufplan: Bellcore-Angriff (XMC4700 Relax Kit Lite)

Die anderen Programme wurden nicht verändert. Die Programmablaufplane können von der Parametersuche übernommen werden.

A.3 RSA

RSA Parameter

Tabelle A.1: RSA mit 256 bit

Parameter	Wert in hexadezimaler Darstellung
p	f63d785fbbef6398e91018b19c33c6c5
q	8a9ac3d1e77f8cd7a1453799de97669d
e	10001
d	48f05efcac7d136ef5417c059c9d1e0146093393506eb 9e0624eb91dc2187f91
n	85520038be7273983f9398ec2260648613a125f4f6bfa faae1ad10cf812664d1

Tabelle A.2: RSA mit 512 bit

Parameter	Wert in hexadezimaler Darstellung
p	dcc5400b2da92ba16f82c36c3b4861474aa5bcaafda5 59b650d6ceb2e87f6a79
q	a6b556d800c662248d7e1d36b10589247f59d3b23266 4a6ba150acd133a5adbd
e	10001
d	ea36ce670f3ff6421236e8eb4edc45d21e9a4ef32da1 5737401cdf87f13d55407d9c797e53172c790c15c21e de988149150c90d4d6f706f37f2508ccce40a21
n	8fc449e2121fdade73c3ae4a7b78db3e6d989ab6d51e d14b35cbabd7873e8283753349e11a004c6705497ed2 09049a488c52c00da3ee8e7b8045f6527c026055

Tabelle A.3: RSA mit 1024 bit

Parameter	Wert in hexadezimaler Darstellung
p	ee30fc801e4a76e2548a924cc15d607514f0a9b79dc7c 6bda142a7c600006fe754ee15fa1fcc3bf500325a9292 612bb75c30cf69233daaef99e6fc1458fc948f
q	ee2ca7f9c89148dc6ca1b7c1ebac4784116c007276db5 46ec5684563e28d103a6fc2733e056454f0018ed83ef3 195aba728bb74087dcd7882f0027c335c7521b
e	10001
d	1fc73bc7e0b08c89e2483bfb08ff7b607782c5b40e434 e6cbaedebd4ce507e726056bfcebe94c1df953f19ea26 41af9e057ac391d668fd1af0bce500b09c500970b67e0 6e600660388870fd73439f59bb509036348fa298a06dc 8b3f79f8652f34985bf561f73afe565ff03c3ffaa7a94 652e253f3a78d00dcd1043d0a8a48c9
n	dd9b1774df290e80dde6211a8eea0e0dea9f8cb0677e7 74fbd9b6591e689f69945d1b23e63cb0ee1d948993b1d 7811dc825637265a677f88f5276fac0949f2537de56d0 dc2aa2c444c8c58d41cb2c1adb688e9e9e9db04302bca 6b2cf81197386d051a827417cdca8334b64874d51d75d 6b1dda295e5bc9be65f83ac60627915

A.3.1 Nachrichten für die Signatur

Tabelle A.4: Nachrichten für die Schlüssellänge 256 bit

Schlüssellänge	Nachricht in hexadezimaler Darstellung
1	58
Zufall	52436a5b5651457164626a3d62
256	45775862775c496e3d6d42705c45354d5a326e3f35785f4c5 07d59385a4a4541

Tabelle A.5: Nachrichten für die Schlüssellänge 512 bit

Schlüssellänge	Nachricht in hexadezimaler Darstellung
1	58
Zufall	6c376e53625e52
512	6375547378374b565c6a46356d613d375c483e747c715130 3b6e376d3256316a4352717a747d5e436a41537347587674 4b35484e5d514072554b5361505d727d

Tabelle A.6: Nachrichten für die Schlüssellänge 1024 bit

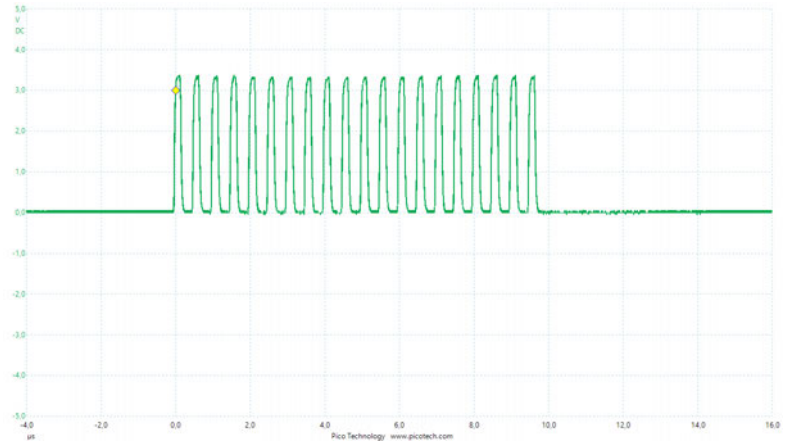
Schlüssellänge	Nachricht in hexadezimaler Darstellung
1	58
Zufall	513855594e667a75375e736d4e5a60787274
1024	6f6e38394161667c34774f473869404a37764672717a62443 1364d6e4d626c6e4f7a5c6d75683b367e724d467b4332476 25c50574738566e69794a473b62576c41607a4236624e4c5 53a32417946306151 5f656e563368505e7e4d415a5e6b4b4 67d62765b5b766e5e4570677a54344e647e307a726641343 a547b5f6c3f6c7e

Alle weiteren Daten befinden sich auf der beigefügten CD.

A.4 Programmoptimierung (Tiva)

A.4.1 Daten für Geschwindigkeit 5 und Optimierungslevel off

(a) Optimierungsassistent: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel Off



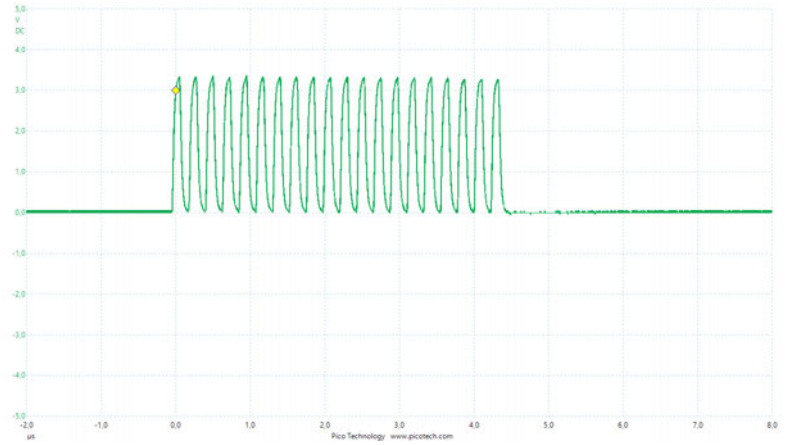
(b) Picoscope: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel Off

Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	33,35 ns	33,35 ns	33,35 ns	33,35 ns	0 s	1	Gesamte Spur
C	Zykluszeit	500 ns	500 ns	500 ns	500 ns	0 s	1	Gesamte Spur
C	Anstiegszeit [80/20%]	29,8 ns	29,8 ns	29,8 ns	29,8 ns	0 s	1	Gesamte Spur

(c) Picoscope: Messungen für Optimierungslevel Off

Abbildung A.8: Datensatz für Optimierungslevel Off

(a) Optimierungsassistent: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 0

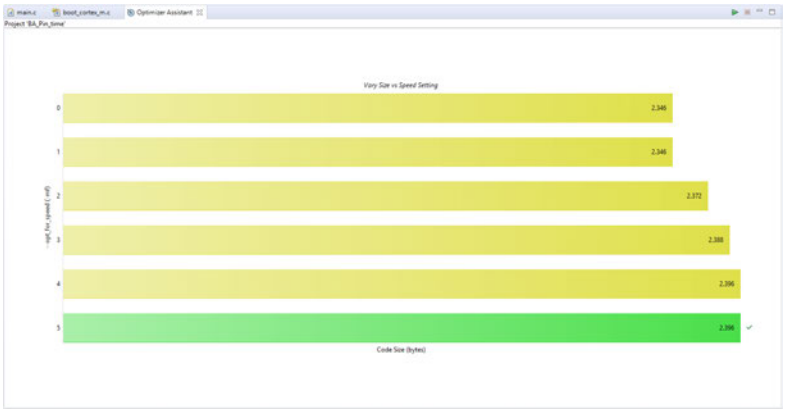


(b) Picoscope: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 0

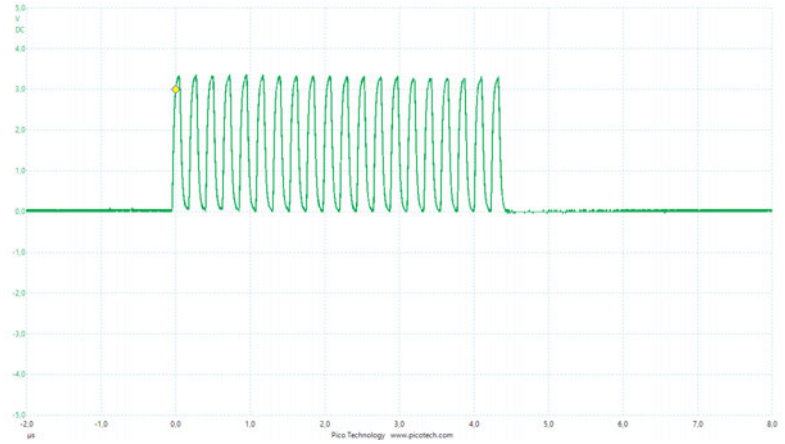
Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	33,45 ns	33,45 ns	33,45 ns	33,45 ns	0 s	1	Gesamte Spur
C	Zykluszeit	225 ns	225 ns	225 ns	225 ns	0 s	1	Gesamte Spur
C	Anstiegszeit [80/20%]	29,5 ns	29,5 ns	29,5 ns	29,5 ns	0 s	1	Gesamte Spur

(c) Picoscope: Messungen für Optimierungslevel 0

Abbildung A.9: Datensatz für Optimierungslevel 0



(a) Optimierungsassistent: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 1



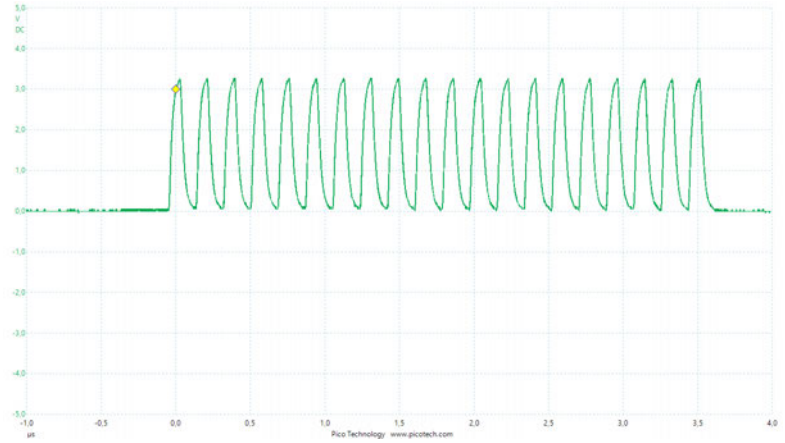
(b) Picoscope: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 1

Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	33.3 ns	33.3 ns	33.3 ns	33.3 ns	0 s	1	Gesamte Spur
C	Zykluszeit	225 ns	225 ns	225 ns	225 ns	0 s	1	Gesamte Spur
C	Anstiegszeit [80/20%]	29.5 ns	29.5 ns	29.5 ns	29.5 ns	0 s	1	Gesamte Spur

(c) Picoscope: Messungen für Optimierungslevel 1

Abbildung A.10: Datensatz für Optimierungslevel 1

(a) Optimierungsassistent: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 2



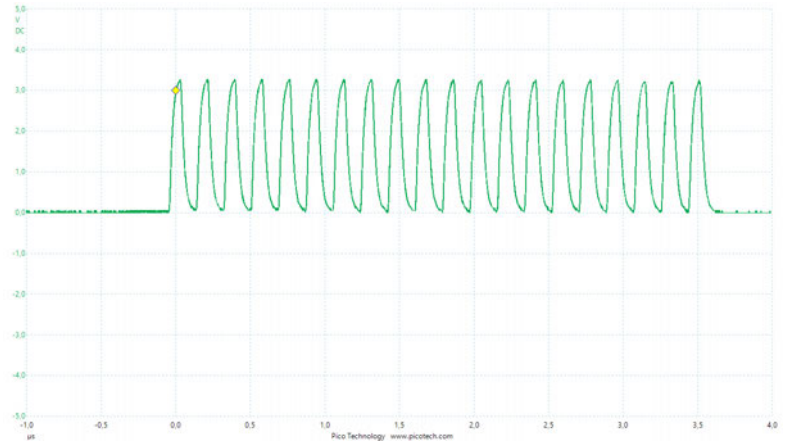
(b) Picoscope: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 2

Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	31,2 ns	31,2 ns	31,2 ns	31,2 ns	0 s		Gesamte Spur
C	Zykluszeit	183,3 ns	183,3 ns	183,3 ns	183,3 ns	0 s		Gesamte Spur
C	Anstiegszeit [80/20%]	26,8 ns	26,8 ns	26,8 ns	26,8 ns	0 s		Gesamte Spur

(c) Picoscope: Messungen für Optimierungslevel 2

Abbildung A.11: Datensatz für Optimierungslevel 2

(a) Optimierungsassistent: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 3

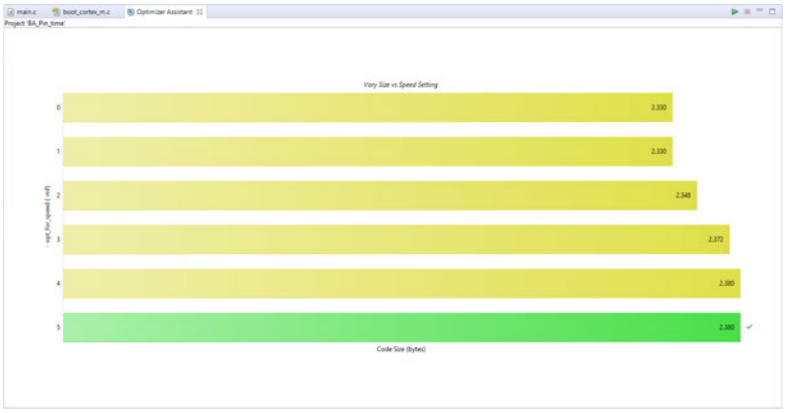


(b) Picoscope: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 3

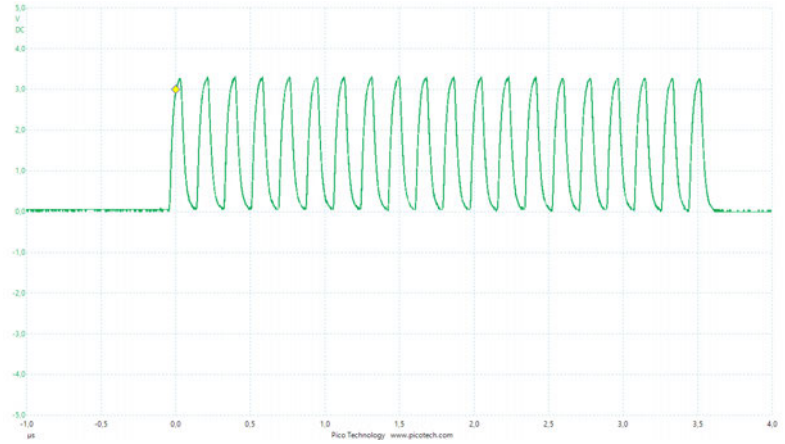
Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	31,2 ns	31,2 ns	31,2 ns	31,2 ns	0 s		Gesamte Spur
C	Zykluszeit	183,3 ns	183,3 ns	183,3 ns	183,3 ns	0 s		Gesamte Spur
C	Anstiegszeit [80/20%]	26,8 ns	26,8 ns	26,8 ns	26,8 ns	0 s		Gesamte Spur

(c) Picoscope: Messungen für Optimierungslevel 3

Abbildung A.12: Datensatz für Optimierungslevel 3



(a) Optimierungsassistent: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 4



(b) Picoscope: Programmgeschwindigkeit vs. Programmgröße für Optimierungslevel 4

Kanal	Name	Wert	Min.	Max.	Mittelwert	Standardabweichung	Aufzeichnungszähler	Spanne
C	Abfallzeit [80/20%]	31,2 ns	31,2 ns	31,2 ns	31,2 ns	0 s	1	Gesamte Spur
C	Zykluszeit	183,3 ns	183,3 ns	183,3 ns	183,3 ns	0 s	1	Gesamte Spur
C	Anstiegszeit [80/20%]	26,8 ns	26,8 ns	26,8 ns	26,8 ns	0 s	1	Gesamte Spur

(c) Picoscope: Messungen für Optimierungslevel 4

Abbildung A.13: Datensatz für Optimierungslevel 4

Glossar

Alice Alice steht stellvertretend für eine, an der Kommunikation beteiligte Person.

Baudrate Anzahl an übertragenden Symbolen pro Sekunde.

Bob Bob steht stellvertretend für eine, an der Kommunikation beteiligte Person.

Build Entwicklungsstufe einer Software.

Chinesischer Restsatz Der Chinesische Restsatz ist ein math. Theorem. Es findet außerdem Verwendung in der CRT-Implementierung des RSA.

Code Composer Studio Code Composer Studio ist eine IDE zum Erstellen von Anwendungen für Texas Instrument Prozessoren.

Flint Bibliothek für Langzahlarithmetik (unteranderem für die Programmiersprache C).

Metall-Oxid-Halbleiter-Feldeffekttransistor Der Metall-Oxid-Halbleiter-Feldeffekttransistor gehört zu den Feldeffekttransistoren mit isoliertem Gate.

Polling Polling ist eine zyklische Abfrage, um den Status von Soft- oder Hardware zu ermitteln.

Pullup-Widerstand Ist ein Widerstand, welcher die Signalleitung mit einem höherwertiges Potenzial verbindet, solange diese nicht aktiv auf das niedriges Potenzial gezogen wird.

Schwellspannung Spannung bei der z. B. ein MOSFET leitend wird..

Square And Multiply Implemtierungsmöglichkeit des RSA.

Watchdog deut. Wachhund, Überwacht eine bestimmte Tätigkeit des Programms und gibt eine Meldung im Fehlerfall. Diese Meldung kann auch das Zurücksetzen beinhalten. Beispiel: Zeitüberwachung für bestimmte Funktionsabläufe.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original