Hochschule
Augsburg University of
Applied Sciences

Hardwaresysteme
Kürprojekt

Fakultät für
Informatik

Studienrichtung
M.Sc. Informatik

Mathias Schoppe
Entwicklung eines ICs - Der Schaltkreisentwurf

Betreuer: Prof. Dr.-Ing. Gundolf Kiefer
Abgabe der Arbeit am: 09.07.2023

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon +49 821 5586-3450
Fax +49 821 5586-3499

Verfasser der Ausarbeitung:
Mathias Schoppe
Matr. Nr.: 2165952
M. Sc. Informatik
Teamkollegen: Sascha Binkert
und Timo Winklbauer

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In dieser Ausarbeitung liegt der Fokus auf der Entwicklung eines eigenen integrierten Schaltkreises (IC) mithilfe des Tiny-Tapeout-Projekts[1]. Der Prozess des IC-Designs ist äußerst komplex und erfordert spezialisierte Kenntnisse sowie teure Ausrüstung und Ressourcen. Das Tiny-Tapeout-Projekt hat es geschafft, ein besonderes Konzept zu entwickeln, dass selbst Anfängern ermöglicht, ihre eigenen IC-Designs zu realisieren. Es bietet eine kostengünstige Möglichkeit, ein eigenen IC fertigen zu lassen und eröffnet somit neue Möglichkeiten für kreative Ideen und individuelle Schaltungen. Das Ziel der Arbeit besteht darin, ein eigenes IC-Design für ein Reaktionsspiel zu entwerfen und zu simulieren. Hierbei wird darauf geachtet, dass maximal 500 Logikgatter und ausschließlich die Ein- und Ausgänge des Tiny-Tapeout Evaluationsboards für die Umsetzung benötigt werden. Durch die Realisierung dieses Projekts sollen grundlegende Kenntnisse im IC-Design erlangt und praktische Erfahrungen in der Entwicklung von elektronischen Schaltungen gesammelt werden. Dieses Projekt wird in drei seperaten Ausarbeitungen unterteilt. Der Prozess des IC-Designs, verschiedene Tools von Tiny-Tapeout und der Schaltkreisentwurf des Reaktionsspiels werden jeweils von Timo Winklbauer, Sascha Binkert und Mathias Schoppe abgedeckt. Dieser Teil der Ausarbeitung behandelt Aspekte, welche für das Vorhaben der konkreten Umsetzung zu beachten sind und bezieht sich dabei auf den Entwurf des Reaktionsspiels.

2 Grundlagen

Das nachfolgende Kapitel befasst sich mit grundlegenden Themen, welche für die Umsetzung der Arbeit unabdingbar sind. Die Themenbereiche werden zusammengefasst und so erläutert, dass auch Lesende ohne Vorerfahrung im Bereich der Hardwareentwicklung alle Aspekte der Arbeit nachvollziehen können. Es wird ausschließlich auf Grundlagen, welche relevant für die konkrete Umsetzung des Projekts sind, eingegangen.

2.1 Linear rückgekoppeltes Schieberegister

Ein linear rückgekoppeltes Schieberegister (engl. linear feedback shift register (LFSR)) ist eine Schaltung, die aus einer Reihe von Flip-Flops (digitale Speicherelemente) besteht, die in einer Kette miteinander verbunden sind. Die Rückkopplung erfolgt, indem das Ausgangssignal eines Flip-Flops mit dem Eingang eines vorherigen Flip-Flops verbunden wird. Dadurch bildet sich ein geschlossener Rückkopplungs-Pfad.[2]
Die Anzahl der Flip-Flops in einem Schieberegister bestimmt die Anzahl der Speicherplätze oder Zustände, die es halten kann. Ein 4-Bit-Schieberegister kann beispielsweise 16 verschiedene Zustände einnehmen, während ein 8-Bit-Schieberegister 256 verschiedene Zustände einnehmen kann.

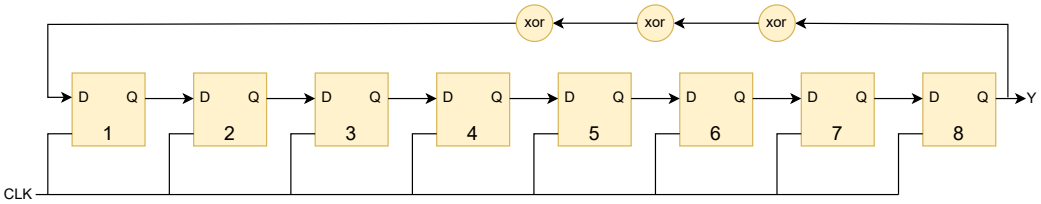


Abbildung 1: Abbildung eines Fibonacci-LFSR, angelehnt an [3]

In Abbildung 1 ist ein 8-Bit Fibonacci-LFSR dargestellt. Dabei repräsentiert CLK den Takteingang und Y den Ausgang des LFSR. Bei jedem Taktimpuls wird der gespeicherte Wert der Flip-Flops eine Stelle weiter geschoben. Bei jedem Taktimpuls wird das Rückkopplungs-Bit in das erste Flip-Flop des Schieberegisters eingespeist, alle anderen Bits werden zum nächsten Flip-Flop geschoben. Dies bedeutet, dass das Ausgangssignal des ersten Flip-Flops als Eingangssignal des zweiten Flip-Flops dient, das Ausgangssignal des zweiten Flip-Flops als Eingangssignal des dritten Flip-Flops und so weiter.

Das Fibonacci-LFSR zeichnet sich durch das in Gleichung 1 dargestellte primitive Generatorpolynom aus. Dies ermöglicht dem Schieberegister die Generierung von Pseudozufallszahlen.

$$pf(x) = x^8 + x^6 + x^5 + x^4 + 1 \tag{1}$$

2.2 Hardware Timer

Der Hardware Timer ist eine Schaltungskomponente, die verwendet wird, um Zeit zu messen oder Zeit basierte Ereignisse zu erzeugen. Er besteht aus einem internen Zähler, der bei jedem Taktimpuls inkrementiert oder dekrementiert wird und so die Grundlage für die Zeiterfassung bildet. Der Hardware Timer wird durch Steuersignale gesteuert, welche das starten, stoppen und zurücksetzen ermöglichen. Diese Steuersignale können entweder von einer externen Steuerung oder durch interne Logik generiert werden.
Damit durch den Hardware Timer eine bestimmte Zeit in Sekunden überbrückt wird, betrachten wir folgende Zusammenhänge. Die Taktfrequenz f gibt an, wie viele Taktzyklen pro Sekunde auftreten. Um eine bestimmte Zeit t in Sekunden zu warten, muss die Anzahl der Takte berechnet werden, die während dieser Zeit vergehen. Dazu wird in Gleichung 2 die Zeit t mit der Taktfrequenz f multipliziert.

$$\text{Anzahl der Takte} = t * f \tag{2}$$

Es ist wichtig zu beachten, dass diese Berechnung auf der idealisierten Annahme basiert, dass die Taktfrequenz und der Takt stabil und präzise sind. In der Praxis können jedoch Ungenauigkeiten auftreten, die die Zeitmessung beeinflussen.

3 Schaltkreis Entwurf

In diesem Kapitel wird die Entstehung des IC-Entwurfs unseres Reaktionsspiels erläutert. Durch die Simulation des Designs werden wir die Funktionalität des Reaktionsspiels überprüfen und abschließend ein Layout erstellen, das für die Fertigung des ICs im Rahmen des Tiny-Tapeout-Projekts verwendet werden kann.

3.1 Das Reaktionsspiel

Um den Schaltkreis genau definieren zu können, muss vorab die exakte Funktionsweise des Reaktionsspiels definiert werden. Das Spiel wird ausgelegt für zwei Spieler. Jeder Spieler hat zwei Knöpfe als Eingabemöglichkeit und es gibt einen zusätzlichen Startknopf. Die insgesamt fünf Knöpfe werden als Eingangssignale in den von uns entwickelten IC übermittelt. Als Interface für die Kommunikation mit dem Benutzer wird die auf dem Evaluationsboard von Tiny-Tapeout verbaute 7-Segment-Anzeige verwendet.[4] Dadurch ergibt sich das in Abbildung 2 dargestellte Design für das Reaktionsspiel.

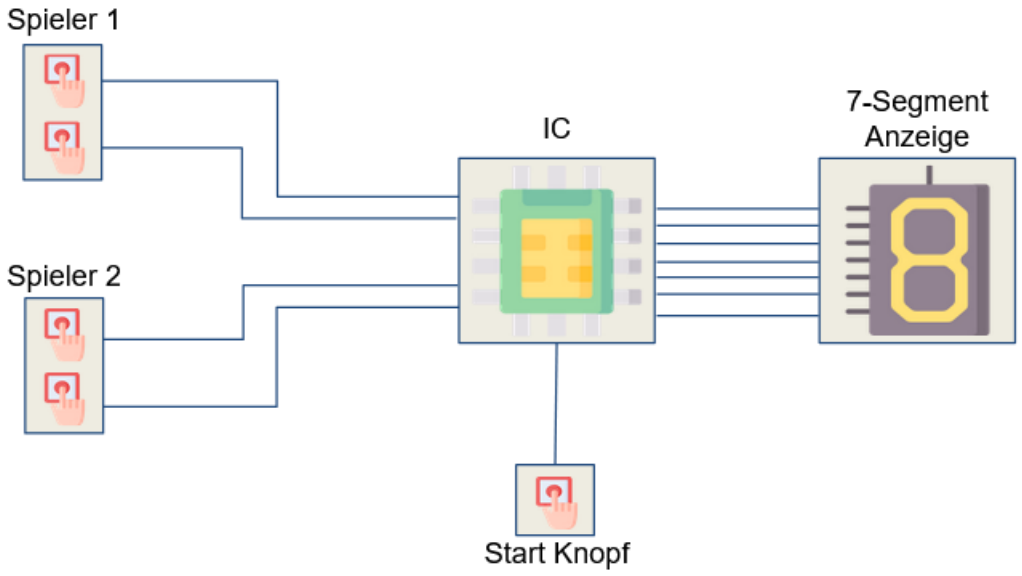


Abbildung 2: Design des Reaktionsspiels

Die Leuchtdioden der 7-Segment-Anzeige werden für die Visualisierung drei verschiedener Zustände verwendet, wie in Tabelle 1 dargestellt.

Zustand	Beschreibung
Blinken	Wenn ein Spiel gestartet wird, blinken alle Segmente der Anzeige innerhalb von drei Sekunden drei mal auf
Zufälliges Aufleuchten	Nachdem eine pseudo zufällige Wartezeit verstrichen ist, leuchtet entweder das linke (bzw. obere) oder rechte (bzw. untere) Segment auf.
Gewinner Anzeigen	Je nachdem, ob ein Spieler zu früh, den richtigen Knopf oder den falschen Knopf drückt wird der Sieger ermittelt. Gewinnt Spieler 1, dann leuchtet die linke (bzw. obere) Hälfte der 7-Segment-Anzeige auf. Gewinnt Spieler 2, dann leuchtet die rechte (bzw. untere) Hälfte der 7-Segment-Anzeige auf.

Tabelle 1: Erklärung der Zustände der 7-Segment-Anzeige

Anhand der in Tabelle 1 definierten Zustände, lässt sich der Spielablauf für unser Reaktionsspiel ableiten. Das Spiel durchläuft die folgenden Schritte:

1. Warten bis der Start-Knopf betätigt wird
2. Sobald das Spiel Startet, blinkt die 7-Segment-Anzeige drei mal auf
3. Wenn das blinken beendet ist, warte eine (pseudo-)zufällige Zeit
4. Nach Ablauf der Zufallszeit, leuchtet das linke oder rechte Segment auf
5. Warte auf Knopfdruck der Spieler
6. Sieger wird ermittelt und die entsprechenden Segmente in der 7-Segment-Anzeige leuchten auf

3.2 Register-Transfer-Ebene

Nachdem die grundlegenden Funktionen des ICs im vorherigen Kapitel definiert wurden, kann der Entwurf auf Register-Transfer-Ebene beginnen. Der Prozess des Entwurfs durchläuft mehrere Schritte. Zuerst wird der Algorithmus anhand einer einfachen Beschreibungssprache definiert. Danach werden die Zustände des Algorithmus in einem Zustandsübergangsdiagramm modelliert, sodass die Überführung in eine Hardwarebeschreibungssprache (hier **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage (VHDL)) im nächsten Schritt vereinfacht wird. Zuletzt wird das Programm synthetisiert und simuliert. Auf den Entwurf des Algorithmus wird in dieser Stelle nicht tiefer eingegangen, da die Funktionsweise des Schaltkreises in den Schritten des Zustandsübergangsdiagramms und der Überführung in VHDL klar wird.

3.2.1 Zufallsgenerierung

Für die Zufallsgenerierung wird, wie im Kapitel 2.1 eingeführt, ein 8-bit linear rückgekoppeltes Schieberegister verwendet. Als Bereich für die zufällige Wartezeit haben wir eine Zeit zwischen 0,5 und 5 Sekunden definiert. Nach Formel 2 kann abhängig von der Taktfrequenz des Evaluationsboards die Anzahl der Takte ausgerechnet werden, welche mindestens bzw. maximal gewartet werden dürfen um in diesem Bereich zu bleiben. Die Taktfrequenz des Evaluationsboards ist einstellbar.[4] Wir haben für unser Projekt eine Taktfrequenz von 25 Kilohertz festgelegt. Daher ergibt sich für die zufällige Wartezeit von 0,5 Sekunden folgende Taktanzahl:

$$cnt_{min} = 25.000 * 0.5 = 12.500$$

Die maximale Anzahl der Takte lässt sich wie folgt berechnen:

$$cnt_{max} = 25.000 * 5 = 125.000$$

Für den Zähler werden also 17-Bit benötigt, um den Wertebereich bis 125.000 darstellen zu können. Damit mindestens eine halbe Sekunde Wartezeit entsteht, wird der Zähler mit der Bitfolge $00011000100000000 (12.544)$ initialisiert, was in etwa der halben Sekunde entspricht. Für die zufällige Wartezeit werden die drei niedrigsten Bits des LFSR für die drei höherwertigsten bits des Zählers verwendet, wie in Abbildung 3 dargestellt ist.

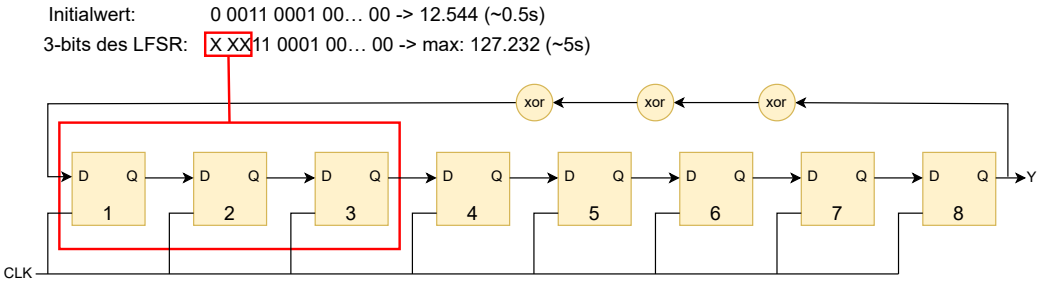


Abbildung 3: Wertevergabe des Zählers für die zufällige Wartezeit

Somit ergeben sich die in Tabelle 2 dargestellten möglichen Kombinationen für die Wartezeit.

Bitfolge	Wartezeit in Sekunden
0011 0001 00... 0	0,50 Sekunden
0 0111 0001 00... 0	1,16 Sekunden
0 1011 0001 00... 0	1,81 Sekunden
0 1111 0001 00... 0	2,47 Sekunden
1 0011 0001 00... 0	3,12 Sekunden
1 0111 0001 00... 0	3,78 Sekunden
1 1011 0001 00... 0	4,43 Sekunden
1 1111 0001 00... 0	5,09 Sekunden

Tabelle 2: Alle möglichen Bitfolgen des Zählers für die zufällige Wartezeit

Für die Selektion, ob das linke oder rechte Segment der 7-Segment-Anzeige aufleuchten soll, wird das höchstwertige Bit des LFSR extrahiert. Der Wert 0 bedeutet, dass das rechte Segment aufleuchten soll und der Wert 1, dass das linke Segment aufleuchten soll.

3.2.2 Zustandsübergrangsdiagramm

In Abbildung 4 ist das aus dem Algorithmus überführte Zustandsübergangsdiagramm dargestellt. Das Diagramm modelliert die folgenden Zustände:

- **State_Reset:** In diesem Startzustand wird auf das drücken des Start-Knopfs gewartet. Sobald dieser gedrückt wurde, wird der Zustand verlassen
- **State_blink3Times:** Der Zustand fasst den Ablauf des dreifachen aufblinken der Segmentanzeige zusammen. Ist das Blinken beendet, wird die zufällige Wartezeit initialisiert
- **State_waitRndInit:** Setzt den Zähler des Timers auf einen zufälligen Wert, sodass nach Formel 2 eine Zeit zwischen 0.5-5 Sekunden gewartet wird
- **State_waitRnd:** Dekrementiert den Zähler bei jedem Clock-Zyklus. Währenddessen wird geprüft, ob ein Spieler einen seiner Knöpfe zu früh betätigt. Falls ja, wird direkt der Gewinner angezeigt. Sonst leuchtet nach Ablauf der Zeit entweder das linke oder rechte Segment auf
- **State_setRightLED:** Das rechte Segment leuchtet und es wird auf Knopfdruck von einem der Spieler gewartet, sodass der Gewinner ermittelt werden kann
- **State_setLeftLED:** Das linke Segment leuchtet und es wird auf Knopfdruck von einem der Spieler gewartet, sodass der Gewinner ermittelt werden kann

- **State_p1Won:** Spieler 1 hat das Spiel gewonnen, das bedeutet die linke Hälfte der 7-Segment-Anzeige leuchtet auf. Anschließend wird auf ein erneutes drücken des Start-Knopfes gewartet
- **State_p2Won:** Spieler 2 hat das Spiel gewonnen, das bedeutet die rechte Hälfte der 7-Segment-Anzeige leuchtet auf. Anschließend wird auf ein erneutes drücken des Start-Knopfes gewartet

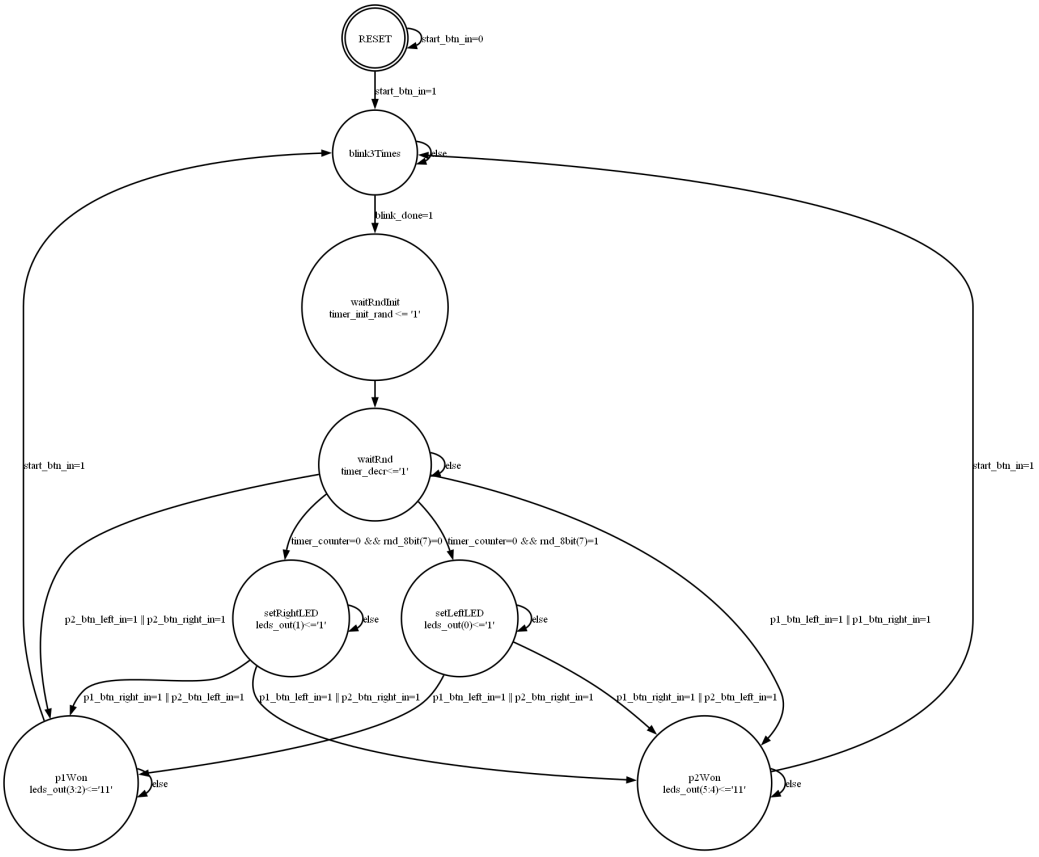


Abbildung 4: Zustandsübergangsdiagramm des ICs

Dieser Ablauf stellt exakt die Funktionsweise unseres Algorithmus dar und wurde anschließend von uns in VHDL-Code übersetzt. Die Signalbenennung muss, damit die Abgabe bei Tiny-Tapeout stattfinden kann, an das vorgegebene Interface wie folgt angepasst werden.

```
ENTITY ALGORITHMUS IS
  PORT(
    clk : IN STD_LOGIC; -- 25 kHz
    rst_n: IN STD_LOGIC;
    uio_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    uio_oe: out STD_LOGIC_VECTOR (7 DOWNTO 0);
    uo_out : out STD_LOGIC_VECTOR(7 DOWNTO 0);
    ena : in STD_LOGIC
  );
END ALGORITHMUS;
```

Die Signale uio_{oe}, uio_{in} und uo_{out} haben folgende Bedeutung:

- uio_{oe}: Dieses Signal konfiguriert die bidirektionale Input/Output Schnittstelle des Evaluationsboards. (0 = Input, 1 = Output)
- uio_{in}: Steht für das Input Signal in die 8-Pin-bidirektionale-Input/Output Schnittstelle des Evaluationsboards, diese werden für unsere fünf Spiel-Knöpfe als Inputs verwendet
- uo_{out}: Steht für das Output Signal der 8-Pin-Output Schnittstelle des Evaluationsboards und ist mit der 7-Segment-Anzeige verbunden

3.2.3 Simulation

Die Simulation unseres Schaltkreises findet in zwei Komponenten statt:

1. **Testbench**: Um die Korrektheit des Algorithmus zu validieren, wird eine VHDL-Testbench geschrieben
2. **Field Programmable Gate Array (FPGA)**: Als weitere Absicherung, wird der Algorithmus so angepasst, dass er auf einem FPGA laufend gemacht und das Spiel vorab getestet werden kann

Nachdem die Testbench erfolgreich elaboriert und simuliert wurde, können wir diese mit dem Tool GTKWave[5] betrachten. In Abbildung 5 ist der Verlauf der Ein- und Ausgänge unseres ICs dargestellt.

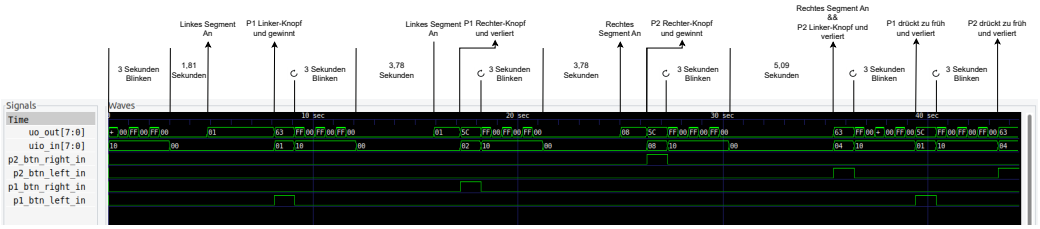


Abbildung 5: Signalverläufe der Algorithmus Testbench

Für die Simulation mit einem FPGA wird das Zybo Zynq-7000 ARM/FPGA SoC Trainer Board [6] verwendet. Der Code für den IC musste an die Schnittstellen des Boards angepasst werden. Das Reaktionsspiel auf dem FPGA wurde im Abschluss-Kolloquium vorgeführt. Leider sind hierbei keine Bildaufnahmen dokumentiert worden.

3.3 Gatter-Ebene, Verdrahtung und Platzierung

Auf Gatter-Ebene und für die Verdrahtung und Platzierung werden die Aufgaben von Tiny-Tapout abgenommen. Damit dies möglich ist, muss ein Github Projekt angelegt werden, welches sich an einer Vorlage orientiert. In der Projekt-Vorlage wurden Github-Actions angelegt, zur Validierung des Schaltkreises. Wenn alle Tests bestehen, resultiert eine gds-2 Datei[7]. Da Tiny-Tapeout ausschließlich Verilog als Hardwarebeschreibungssprache akzeptiert, mussten unsere VHDL-Dateien mithilfe des Tools yosys[8] zu gültigem Verilog Code konvertiert werden. In unserem aktuellen Stand, wurde der Verilog Algorithmus von den Github-Actions akzeptiert, wie in Abbildung 6 zu sehen ist. Vor der Einreichung bei Tiny-Tapeout, muss der konvertierte Code für den Algorithmus erneut auf Richtigkeit überprüft werden.

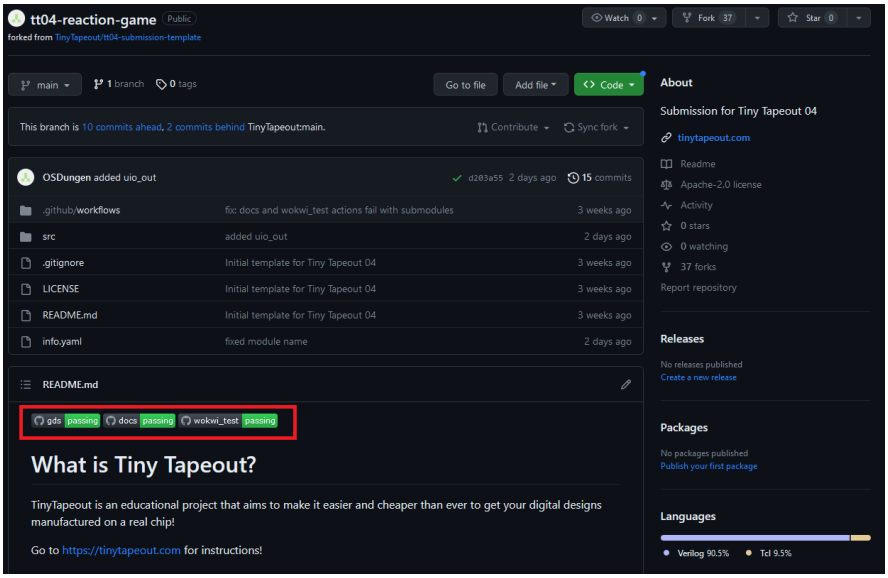
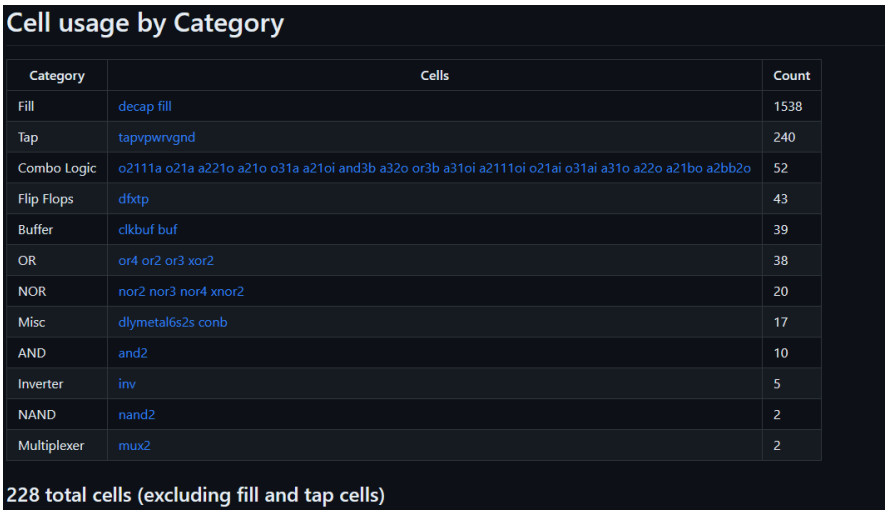


Abbildung 6: Tiny-Tapeout04 Github Repository

Darüber hinaus ermöglicht GitHub den Zugriff auf eine Statistik, die zusammenfasst, wie häufig bestimmte Standardzellen in unserem Schaltkreis vorkommen. Wie aus Abbildung 7 entnommen werden kann, wird für unsere Schaltung 228 Zellen benötigt. Für den Schaltkreis werden Combo Logic, Flip Flops, Buffer, Or, And, Nand, Inverter und Multiplexer verwendet. Interessant wäre eine Analyse der Misc(dt. "Verschiedenes") Zellen, da diese Latches oder ähnliches beinhalten könnten, welche es im Hardwaredesign zu vermeiden gilt.



Cell usage by Category

Category	Cells	Count
Fill	decap fill	1538
Tap	tapvpwrvgnd	240
Combo Logic	o2111a o21a a221o a21o o31a a21oi and3b a32o or3b a31oi a2111oi o21ai o31ai a31o a22o a21bo a2bb2o	52
Flip Flops	dfxtp	43
Buffer	clkbuf buf	39
OR	or4 or2 or3 xor2	38
NOR	nor2 nor3 nor4 xnor2	20
Misc	dlymetal6s2s conb	17
AND	and2	10
Inverter	inv	5
NAND	nand2	2
Multiplexer	mux2	2

228 total cells (excluding fill and tap cells)

Abbildung 7: Zellübersicht nach Synthese des Algorithmus

4 Zusammenfassung und Ausblick

Das IC-Design für das Reaktionsspiel wurde erfolgreich implementiert und auf einem FPGA getestet. Das Spiel funktionierte wie erwartet, und die Spieler konnten ihre Reaktionszeit messen, indem sie den richtigen Knopf zur richtigen Zeit drückten. Die LEDs zeigten die Spielzustände korrekt an, einschließlich des Blinkens, der zufälligen aufleuchtenden LED und der Anzeige des Gewinners. Beim Testen mit einer VHDL Testbench wurden verschiedene Szenarien durchgespielt, um sicherzustellen, dass das Spiel korrekt reagiert und die gewünschten Ergebnisse liefert. Das IC-Design erwies sich als robust und zuverlässig in Bezug auf die Spiellogik.

Das IC-Design erfüllt alle definierten Anforderungen. Es verwendet die vorgegebenen Eingänge und Ausgänge des Tiny-Tapeout-Evaluationsboards und beschränkt sich auf maximal 500 Logikgatter. Der Entwurf soll im Rahmen des nächsten Rollouts eingereicht werden, sodass der IC entstehen kann. Allerdings muss der von VHDL zu Verilog konvertierte Algorithmus zuvor noch getestet werden.

Der Entwurf des ICs hat zu wertvoller Erfahrung im Bereich der Hardwareentwicklung geführt. Es konnten Kenntnisse und Fähigkeiten im IC-Design erlangt werden. Die Berücksichtigung verschiedener Aspekte, wie die Auswahl geeigneter Logikgatter und das Routing von Signalen hat geholfen, wertvolle Erkenntnisse zu erlangen.

5 Abkürzungsverzeichnis

IC integrierter Schaltkreis

LFSR linear rückgekoppeltes Schieberegister

VHDL **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage

FPGA Field Programmable Gate Array

6 Literaturverzeichnis

Literatur

[1] Tiny Tapeout. *Tiny Tapeout Homepage*. online verfügbar unter `https://tinytapeout.com` zuletzt aufgerufen am 02.07.2023. 2023.

[2] Dirk Fox. „Linear Rückgekoppelte Schieberegister“. In: *Datenschutz und Datensicherheit-DuD* 32.5 (2008), S. 351.

[3] Wikipedia. *Linear rückgekoppelte Schieberegister*. online verfügbar unter `https://upload.wikimedia.org/wikipedia/commons/thumb/e/e4/LFSR-Fibonacci.svg/650px-LFSR-Fibonacci.svg.png` zuletzt aufgerufen am 03.07.2023. 2023.

[4] Timo Winklbauer. „Entwicklung eines ICs - Tiny Tapeout“. In: (2023), S. 10.

[5] Udi Finkelstein. *GTKWave Homepage*. online verfügbar unter `https://gtkwave.sourceforge.net/` zuletzt aufgerufen am 07.07.2023. 2023.

[6] Digilent. *Zybo Zynq-7000 Reference*. online verfügbar unter `https://digilent.com/reference/programmable-logic/zybo/start` zuletzt aufgerufen am 07.07.2023. 2023.

[7] Sascha Binkert. „Entwicklung eines ICs - Vorstellung von verschiedenen Tools“. In: (2023), S. 1–3.

[8] Claire Xenia Wolf. *Yosys Open SYnthesis Suite Github*. online verfügbar unter `https://github.com/YosysHQ/yosys` zuletzt aufgerufen am 07.07.2023. 2023.