Hochschule für Angewandte Wissenschaften
Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Projektabschlussbericht zu Digitale Systeme

Programmieren eines Tetrisspiels mittels FPGA

Abschlussbericht DY-Projekt	Eingegangen am: 28.06.25	Protokollführer: Zhi Hao Tan
Vortragstag 25.06.2025	Testat:	
Dozent: Prof. Dr.-Ing. Robert Fitz		

Inhaltsverzeichnis

1 Ziel **4**

2 Grundlagen **5**

2.1 Spezifikationen 5

2.2 Tetrimino 5

2.3 Steuerung 6

3 Blockschaltbild **7**

3.1 Blöcke/Module 7

3.1.1 PLL Clock Divider 7

3.1.2 Input Controller 8

3.1.3 PRNG - Pseudorandom Number Generator 8

3.1.4 Tick Counter 9

3.1.5 Game Engine 10

3.1.6 Field RAM 11

3.1.7 VGA Controller 11

3.1.8 Video Renderer 12

3.1.9 Scoreboard 12

3.1.10 Click 13

4 Automatengraph **14**

4.1 GAME ENGINE 14

4.2 CLICK 16

5 Stand **17**

5.1 Features 17

5.2 Herausforderungen 18

6 Reflexion **19**

1 Ziel

Tetris ist ein weltweit bekanntes und beliebtes Puzzle-Videospiel, das die Spieler seit seiner Entwicklung im Jahr 1984 in seinen Bann gezogen hat. Seine einfache, aber süchtig machende Mechanik, bei der es um fallende geometrische Blöcke geht, die so angeordnet werden müssen, dass sie vollständige Linien bilden, macht es zu einem idealen Kandidaten für die Erforschung der Implementierung von Echtzeitsystemen. [1]

Das Spiel ist mit dem Digilent Basys3 FPGA Board [2] realisiert, das von der Hochschule ausgeliehen wurde. Das Basys3 Board besitzt Knöpfe, die als Steuerung vom Spiel verwendet sind, und vor allem ein VGA output, das ein graphisches Display auf einem Bildschirm ermöglicht hat. Die 4 zahlige 7-Segmente-Anzeige wurde auch für die Darstellung der Punktzahl genutzt. Die Stromversorgung erfolgt über ein einfaches Micro-USB-Kabel.

Seit dem Zwischenbericht gab es viele Änderungen, die im Laufe dieses Berichts ausführlich beschrieben werden. Dennoch wurde die beabsichtigte Funktionalität des Projekts erreicht.

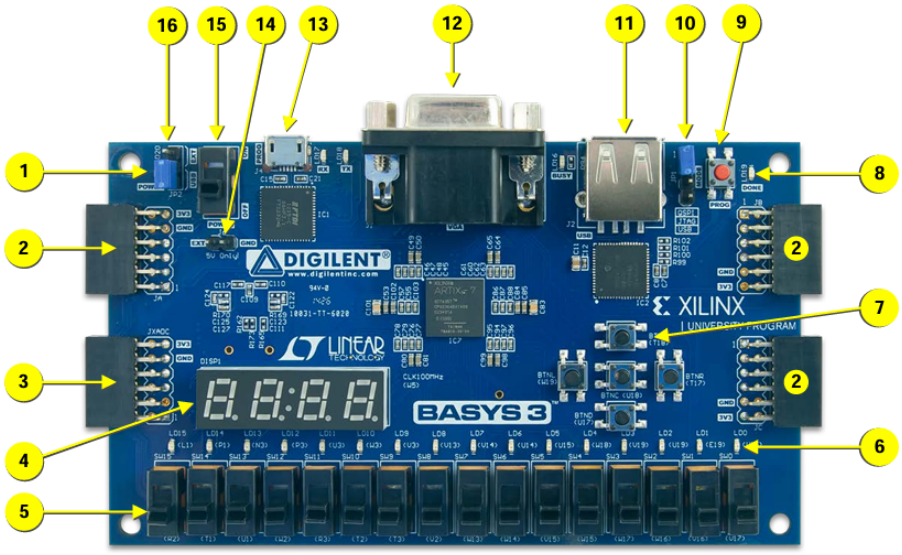

Figure 1. Basys3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Table 1. Basys3 Callouts and component descriptions.

Abbildung 1: Basys3 Board

2 Grundlagen

2.1 Spezifikationen

Ohne einen Weg , das Programmierte visuell darzustellen, wäre es nicht möglich, das Ziel dieses Projekts zu erfüllen. Dafür muss eine Auflösung des Displays und Spielplatzes definiert werden.

Der industrielle Standard für VGA-Output ist hier gebraucht, nämlich eine Auflösung von 640x480 Pixeln, mit einer Bildwiederholfrequenz von 60Hz und einer Pixelfrequenz von 25,175Mhz, welcher auf praktisch allen VGA-Monitoren problemlos angezeigt werden kann. Das Spielfeld muss für den Spieler klar und eindeutig definiert sein, damit er die Veränderungen schnell erkennen und darauf reagieren kann. Dafür ist die Spielfeldgröße von 100x200 Pixeln definiert und dazu auch ein Rand von einem Block groß, der einheitlich 20x20 Pixel hat. Die Spielfeldblöcke haben zusätzlich einen Pufferpixel an den Rändern , damit sie leicht unterschieden werden könnten.

2.2 Tetrimino

Das Tetrimino ist eine Einheit, die aus 4 miteinander verbundenen Blöcke erzeugt ist und als Baustein des Spiels dient. Im Allgemeinen gibt es 7 unterschiedliche Tetriminoblöcke und sie werden am Entstehungspunkt wie folgt erzeugt:

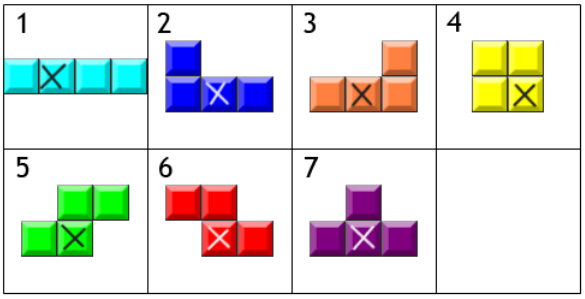

Abbildung 2: Tetrimino-Spawnposition und Drehachse

Das „X“ zeigt an, um welche Achse sie gedreht werden und dient als Referenz für die Ausbringung der Tetriminos. Die Spawnposition der Blöcke ist vordefiniert in der Spielfeld-Position 5x2.

2.3 Steuerung

Obwohl es möglich ist, ein externes und vielleicht einfacher zu bedienendes Gerät wie eine Tastatur oder einen echten Game-Controller zu verwenden, reicht die Zeitrahmen des Projekts nicht, eine Schnittstelle dafür bereitzustellen. Deswegen wird es beschlossen, dass der Spieler die Knöpfe auf dem Basys3 Board bedienen soll, um die Tetriminos zu manövrieren.

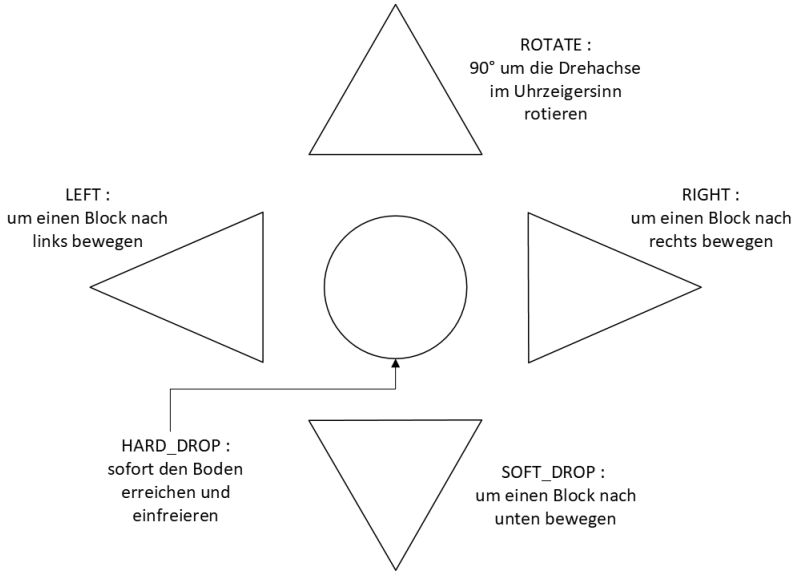

Abbildung 3: Steuerungsoptionen

+ Schalter - Spiel starten (1/HIGH) oder zurücksetzen/resetten (0/LOW)

3 Blockschaltbild

Um einen besseren Überblick des Projekts zu schaffen kommt das Blockschaltbild ins Spiel. Das folgende Blockschaltbild stellt die Hauptkomponenten des Tetris-Systems auf dem FPGA dar und zeigt deren Zusammenspiel. Es bildet die Grundlage für die funktionale und strukturelle Umsetzung der Spielmechanik in der Hardware.

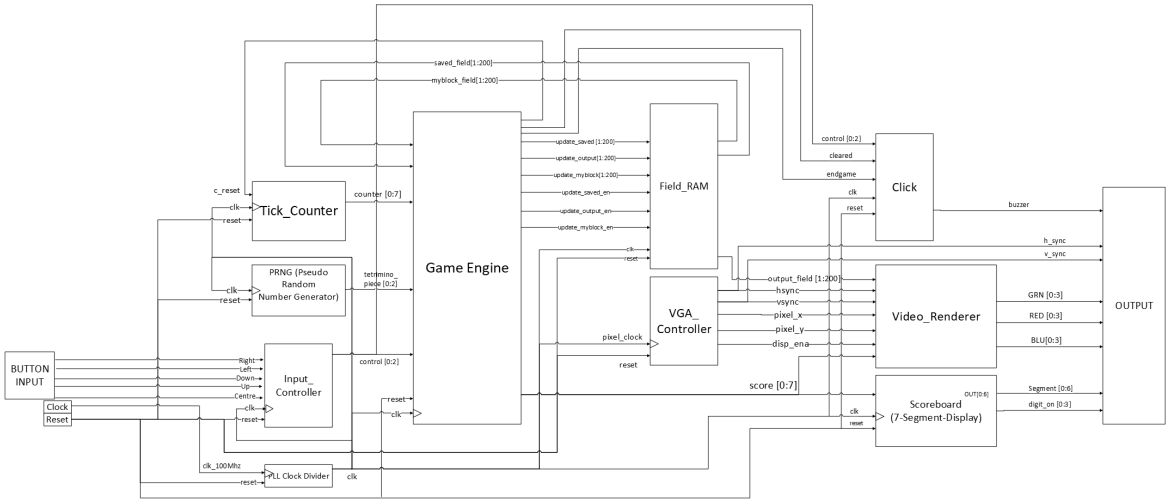

Abbildung 4: Blockschaltbild

Im Vergleich zum Zwischenstand gab es ein paar wichtige Änderungen. Nämlich wurde das `Field_RAM` Module von `Game_Engine` getrennt. Das hat zufolge, das bei der Validierung einfacher zu debuggen ist. Außerdem gibt es ein neues Modul zur Tonausgabe, um das Spiel interaktiver zu gestalten.

3.1 Blöcke/Module

Hier werden die Blöcke des Blockschaltbildes beschrieben. Übrigens gibt es ein Takt- und Reset-Signal am Eingang von allen Blöcken bis auf den `Video Renderer` Block.

3.1.1 PLL Clock Divider

Die VGA Schnittstelle benötigt eine saubere und spezifische Taktfreqeunz , um stabil zu funktionieren. Hierfür ist ein PLL (Phase-Locked-Loop) zu verwenden. Das PLL ist durch das Clocking-Wizard-IP automatisch erstellt, wobei der Nutzer einfach nur die gewünschte Frequenz eingeben soll, um den Takt bereitzustellen. Das gesamtes System verwendet diese Taktfrequenz.

Obwohl das PLL keine exakte Frequenz, die diese VGA-Auflösung bräuchte, erzeugen könnte (Statt 25,175 Mhz gibt das PLL einen Takt von 25,17007Mhz aus), da es ganzzahlige Multiplikatoren und Teiler mit festem Verhältnis verwendet, und 25,175

kein reines Vielfaches oder Teiler von der Quartzfrequenz 100 MHz ist, kommt das Tool dem so nahe wie möglich, und in diesem Fall ist der winzige Frequenzfehler

100-(25,17007/25,175 x 100) = 0,02%

völlig akzeptabel.

3.1.2 Input Controller

Der Spieler kann die Tetriminos über die Tasten auf dem Board steuern – dabei sollen jedoch keine unbeabsichtigten Verzögerungen auftreten. Um Metastabilität zu vermeiden, werden zunächst zwei D-Flipflops zur Synchronisation des Eingangssignals verwendet.

Anschließend wird ein Zwischenspeichersignal genutzt, um das Tastensignal ab dem Zeitpunkt des Tastendrucks für ca. 5ms zu beobachten. Dadurch kann das Signal entprellt und ein stabiler Tastendruck erkannt werden.

Um sicherzustellen, dass ein Tastendruck nur einmal registriert wird, wurde eine Flankenerkennung implementiert. Ohne diese Maßnahme würde die Zustandsmaschine einen langen Tastendruck als mehrere kontinuierliche Eingaben interpretieren, was zu unerwünschten mehrfachen Aktionen führen würde – beispielsweise könnte ein Tetromino mehrfach bewegt oder gedreht werden, obwohl die Taste nur einmal gedrückt wurde. Durch die Erkennung ausschließlich der steigenden Flanke des Tastersignals, also dem Übergang von 'nicht gedrückt' zu 'gedrückt', wird dieses Verhalten unterbunden. Ein gehaltenes Tastendruck hat somit keine weiteren Auswirkungen.

Ein 3-Bit Steuerungsvektor wird generiert und bewegt das aktuelle Tetrimino wie folgt:
000/110/111: nichts
001: nach Links schieben
010: nach Rechts schieben
011: rotieren
100: langsames Herunterfallen
101: schnelles Abwerfen

3.1.3 PRNG - Pseudorandom Number Generator

Die Erzeugung des Tetriminos soll möglichst zufällig und unvorhersehbar sein, um das Spiel interessant und herausfordernd zu gestalten. Genau ist das, das Ziel dieses Moduls.

Dabei kommt ein Linear Feedback Shift Register (LFSR)[4] zum Einsatz. Dieses Prinzip erzeugt eine scheinbar zufällige Bitfolge durch Rückkopplung bestimmter Bits mithilfe von XOR-Verknüpfungen.

In die Implementierung hier besteht das Register aus einem 8-Bit-Vektor, der mit jedem Takt weitergeschaltet wird. Mithilfe eines primitiven Polynoms

$$y = x_8 + x_4 + x_3 + x_2 + x$$

wird ein Feedback-Bit erzeugt, das an der ersten Bitposition des Registers eingefügt wird, während alle anderen Bits um eine Position nach links geschoben werden. Auf diese Weise entsteht bei jedem Takt ein neuer Vektor, der sich nach einer bestimmten Zeitperiode wiederholt.

Anschließend wird der Vektor durch XOR-Verknüpfungen der oberen 6 Bits auf einen 3-Bit-Vektor reduziert, der als pseudozufällige Tetrimino-Kennung verwendet wird.

Im unten gezeigten Bild ist zu erkennen, dass sich die Zustandsfolge in Intervallen von 510 ns wiederholt. Auch wenn dies relativ kurz ist, stellt dies aufgrund der asynchronen Abtastung kein größeres Problem dar. Zudem garantiert das verwendete primitive Polynom, dass alle möglichen Zustände des Registers innerhalb eines Durchlaufs mindestens einmal auftreten.

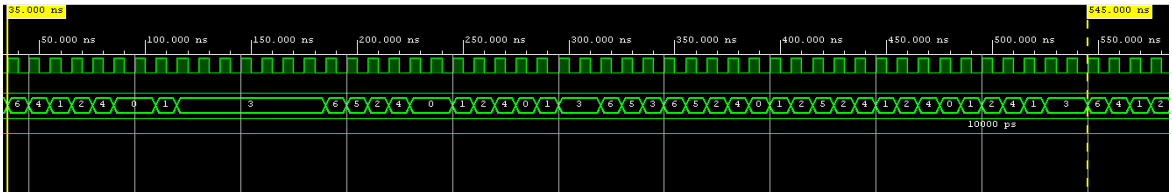

Abbildung 5: Testbench mit dem Zufallszahlgenerator

Da es nur sieben einzigartige Tetriminos gibt, aber der 3-Bit-Ausgang des LFSRs acht mögliche Werte (0 bis 7) ausgeben kann, wird im Fall eines ungültigen Werts (d.h. 7) das zuletzt gültige Ergebnis als aktuelle Ausgabe verwendet. Jeder Tetrimino ist durch eine eindeutige Kennung codiert, sodass der resultierende 3-Bit-Vektor direkt einem der sieben Tetrimino-Typen zugeordnet werden kann. Auf diese Weise wird bei jedem Spawnen eines neuen Tetriminos eine pseudozufällige, aber auch deterministisch reproduzierbare Auswahl getroffen.

3.1.4 Tick Counter

Der Tick-Counter bildet einen zentralen Bestandteil der Spielmechanik, indem er regelmäßige Zeitsignale für zustandsabhängige Abläufe liefert. Er besteht aus einem Clock Divider mit einem internen Zähler, der alle 50ms einen Puls erzeugt. Dieser Puls erhöht einen 8-Bit breiten Zählervektor, der als Grundlage für die Steuerung der Fallgeschwindigkeit bzw. erlaubten Bewegungsdauer der Tetriminos dient. Durch Anpassung der Auswertungslogik des Vectors im `Game Engine` kann die Tick-Periode flexibel an den gewünschten Schwierigkeitsgrad angepasst werden. Zusätzlich wird der Counter über ein Signal `c_reset` nach einem Zustandswechsel zurückgesetzt, um eine konsistente Taktung sicherzustellen.

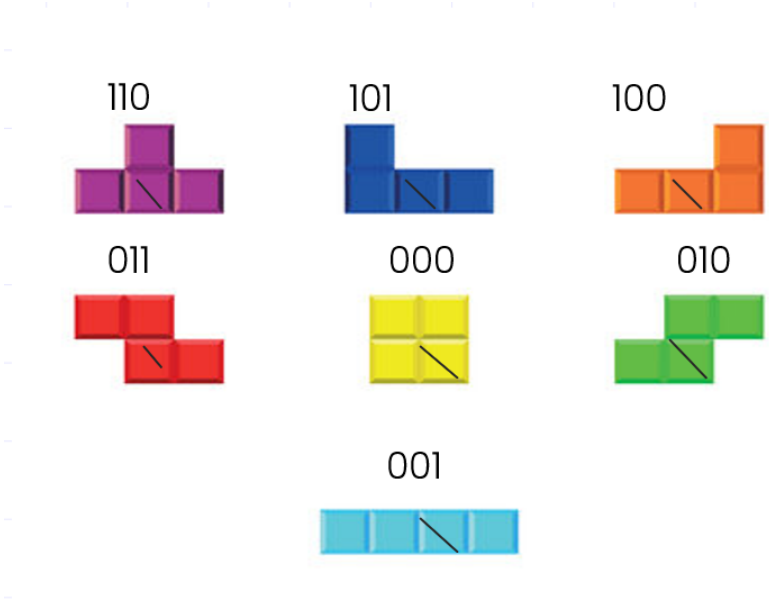

Abbildung 6: Tetrimino Identikfikatorzahl

3.1.5 Game Engine

In diesem Block liegt der Kern des Spiels, wobei die Zustandsmaschine sich hier befindet, die im Abschnitt FSM zu finden ist

Ein dynamisches Tick Signal wird in einem separaten Prozess erzeugt. Hier wird die erlaubte Zeit der Tetrimino-Bewegung zwischen Fallen eines Tetriminos je nach dem Punktezahl verringert, allerdings 50ms pro Punkt. Ab 2 s kann die Zeit hier auf ein Minimum von 500 ms heruntergesetzt werden. Grundlage dafür ist ein Vergleich zwischen diesem Signal und der empfängeten Tick_Counter. Die Punkte sammelt sich unter einem 7-Bit Vektor, das nach jeder Löschung einer Zeile hochgezählt wird.

Innerhalb von diesem Modul gibt es auch wichtige Signale, unter anderem die myblock_next, pivot_x/y, myblock_next und Ton-Signale. myblock_next ist in Prizip ein Geisterfeld, das speichert nur das kommenden Tetrimino und wird nur für Vergleiche mit dem gespeichertem Feld verwendet, um Kollision zu erkennnen und der nächsten Schritt zu bestimmen. pivot_x/y sind Definitionen der Koordinaten für die aktuelle Block-Position. Mit Ton-Signale ist es gemeint, das Kennzeichen eines neulich geleerten Linie, cleared, und wenn das Spiel vorbei ist ,gameover. Sie sind an dem `Click` Modul zu schicken.

Hier ist es wichtig, dass unterschiedlich befüllte 10x20 Spielfelder (saved bzw. das hart-gespeichertes Feld; output bzw. das Ausgangsfeld; myblock bzw. das aktuelle Blockfeld) und die jeweilige Enable-Signale, welches für das Kennzeichen eines Speichrung, an dem Feld-Arbeitsspeicher `Field RAM` geliefert sind, auf die es immer bei Zustandsänderung zugegriffen werden kann.

3.1.6 Field RAM

In diesem Modul wird einfach alle Felder gespeichert, allerdings innerhalb einem getakteten Prozess mit einem Enable-Signal. Andere Modul dürfen jederzeit auf diesen Feldern zugreifen.

3.1.7 VGA Controller

Das VGA-Controller stellt eine zentrale Verbindung zwischen Spiellogik und Benutzeroberfläche dar. Die Ausgabe erfolgt über den VGA-Standard, der eine präzise Ansteuerung von Synchronisation- und Farbsignalen erfordert. Hier werden aber ausschließlich Synchronisation- und aktuelle Pixel-Koordinaten-Signale zugeschickt, wobei die Zuweisung der Farbe durch `Video Renderer` geregelt wird.

Das Controller benötigt also vordefinierte Einstellungswerte für das Display, wobei sie als hier `generic` gespeichert sind. Der Pixeltakt sorgt dafür, die Signalkoordinaten H- und V-Zähler in passenden Zeitintervallen hochgezählt sind (Die sichtbare Bildfläche befindet sich innerhalb eines kleineren Bereichs (640x480), während der restliche Bereich für Synchronisations- und Pufferzeiten reserviert ist.), und dass die H- und V-Sync Signale jeweils am Ende einer vollen Linie und eines vollständig befüllten Bildes rechtzeitig auftaucht.[3] Darüber hinaus wird ein Signal Display Enable übergeben, wenn die Bildschirm Koordinaten sich innerhalb von dem Spielbereich befinden. Das wird bedeutsam während des Färbevorgangs.

VGA Signal 640 x 480 @ 60 Hz Industry standard timing

General timing

Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel freq.	25.175 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

Abbildung 7: VGA Controller Einstellung

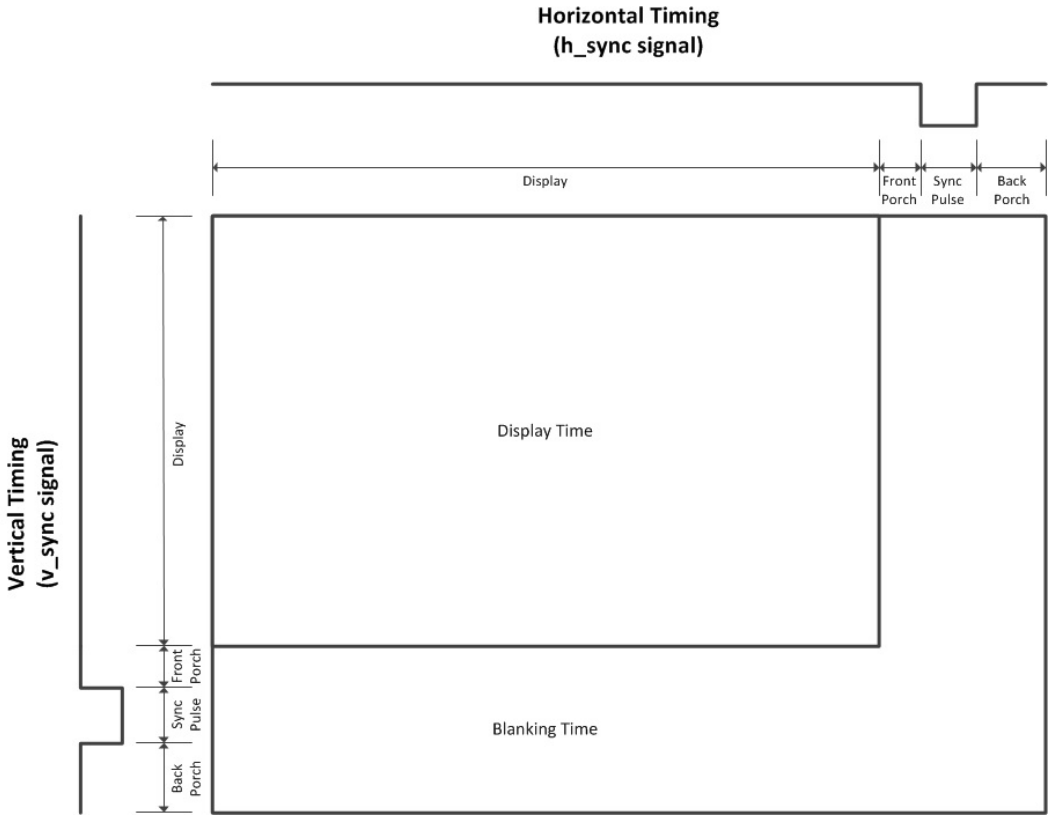

Abbildung 8: Synchronisations- und Pufferzeit

3.1.8 Video Renderer

Diese Schnittstelle verbindet die Ausgabe vom Spiel und die visuelle Anzeige. Mithilfe der wichtigen Werte aus dem `VGA Controller` (H-Sync, V-sync, aktuelle X Koordinate, aktuelle Y Koordinate und das Display-An Indikator) werden die Bildschirmkoordinaten auf logische Blockpositionen im 10×20 Tetris-Feld abgebildet und gefärbt.

Das Spielfeld ist hier aber vordefiniert, das heißt, hier wird nur die aktuelle Block-Information benötigt (entweder eine 1 oder 0) um den Block farblich darzustellen, allerdings auf Schwarz, wo nichts ist, und Weiß, wo es einen Block gibt. Der Rand wird unabhängig von der Eingabe in Grau angezeigt, und lässt sich nicht ändern.

Neu auf dem Display sind nun 3 Ziffern in der linken Ecke, so dass der Spielstand für den Spieler leichter zu erkennen ist. Jedes Ziffer hat eine große von 7x9 Pixeln, das als Konstante in eine Lookup-Tabelle hart kodiert ist.

3.1.9 Scoreboard

Dies ist der Verantwortliche für die Umsetzung der Punktzahl auf dem eingebauten 7-Segmente-Display. Der Punktestand wird intern als 8-Bit-Wert gespeichert und

erhöht sich mit jeder gesetzten oder gelöschten Linie.

Um diesen binären Wert anzuzeigen, wird er zunächst in Decimal konviertiert und in drei dezimale Ziffern (Hunderter, Zehner, Einer) umgewandelt. Da das Basys3 mehrere 7-Segment-Ziffern besitzt, jedoch nur eine gleichzeitig aktivieren kann, wird zeitliches Multiplexing eingesetzt. Dabei wird jede Ziffer einzeln in schneller Folge angezeigt, sodass für das menschliche Auge der Eindruck einer gleichzeitigen Darstellung entsteht. Allerdings wird nur die ganz rechten 3 Ziffern während Betrieb leuchten, weil es nur bis maximal 255 zählen kann.

Die umgerechneten Ziffern werden anschließend durch ein Segment-Encoding (a-g) in die entsprechenden Signale für die 7-Segment-Anzeige übersetzt. Diese Encodierung erfolgt über eine Lookup-Tabelle.

3.1.10 Click

Die Tonausgabe erfolgt über dieses Modul , welches sein eigenes Zustandsautomat bekommen. Es beinhaltet unterschiedliche Frequenzen und Tondauer als Konstante für jeden Zustand, das bei jedem Zustandsänderung für die Festlegung der Rechteckwellen-Periode benutzt, um den Buzzer zu steuern.

Parallel zur Zustandsmaschine läuft ein Zähler, der bei jedem Takt inkrementiert wird. Dieser dient dazu, zu bestimmen, wann ein bestimmter Zustand verlassen werden soll. Grundlage dafür ist ein Vergleich zwischen dem aktuellen Zählerstand und einer vordefinierten Zeitkonstanten.

4 Automatengraph

4.1 GAME ENGINE

Im `Game Engine` wird der gesamte Spielablauf koordiniert, was im wesentlichen durch FSM kontrolliert ist. Das Prinzip der Moore-Maschine ist hier zu verwenden, da der Ausgang von dem jeweiligen Zustand abhängt. Vorauszuschicken ist, dass die Ausgänge in den FSMs hier undefiniert sind, weil die Ausgabe über ein 10x20 Raster erfolgt und hier viel zu kompliziert darzustellen ist.

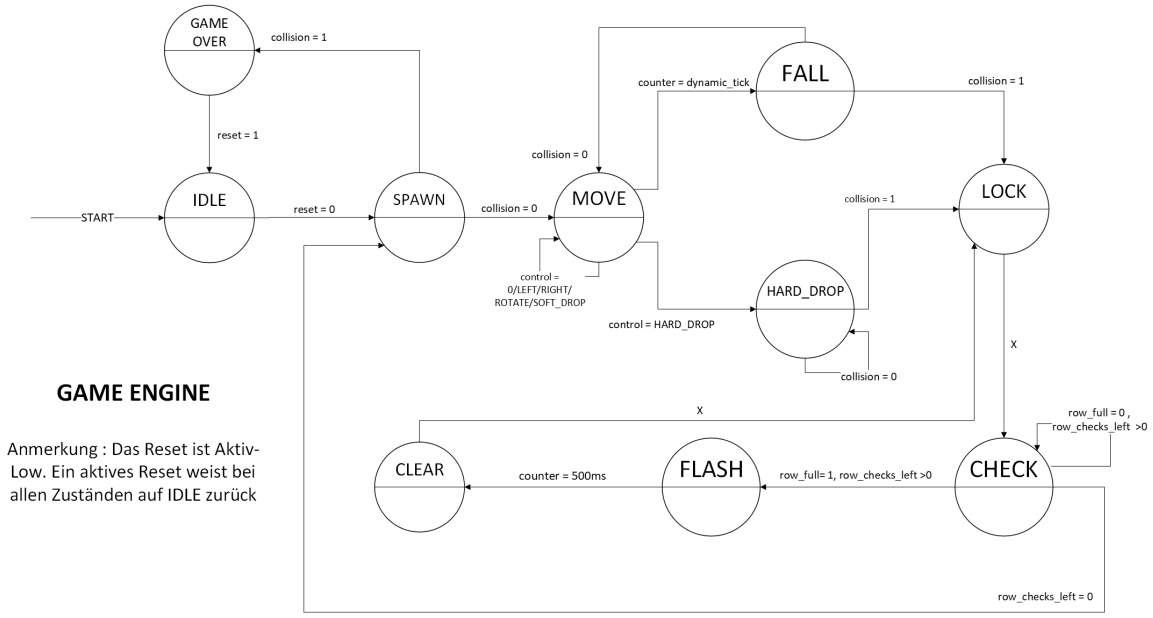

Abbildung 9: Hauptlogik

Jeder Zustand repräsentiert eine spezifische Phase im Spiel. Innerhalb der Zustände sind die 4 Blockfelde im Zwischenspeicher bedeutsam:

myblock - das Feld des gerade kontrollierten Tetriminos

myblock_next - aktualisiertes Positionsfeld des Tetriminos mit Bezug auf den nächsten Zug/Zustand (für Kollisionskontrolle verwendet)

saved_field - gespeichertes Feld

output_field - Ausgabefeld

Zu Beginn des Spiels befindet sich die FSM im SPAWN-Zustand. Dabei wird die Feldposition (5,2) als Referenzpunkt verwendet, um die vier Blöcke eines Tetriminos zu erzeugen. Dieser Referenzpunkt wird gespeichert und für alle späteren Bewegungen des Tetriminos genutzt. Die Koordinaten der vier Blöcke sind für jeden der sieben Blocktypen als horizontale (x) und vertikale (y) Vektoren relativ zum Referenzpunkt vordefiniert.

Anhand dieser Parameter wird ein temporärer Tetrimino im sogenannten Geisterfeld myblock_next erzeugt. Dieses wird mithilfe logischer AND-Operationen mit dem gespeicherten Spielfeld auf Kollision geprüft. Wenn keine Kollision erkannt wird, wird der Geisterblock als aktueller Block in myblock übernommen, mit dem Spielfeld überlagert und an `Output` ausgegeben. Anschließend wechselt der Zustand in MOVE. Dieses Verfahren findet in allen Zuständen außer CLEAR und FLASH Anwendung.

Wenn jedoch bereits beim SPAWN eine Kollision auftritt, wechselt die FSM in den GAMEOVER-Zustand. Hier wird das Ausgabefeld in Intervallen von 500 ms in invertierten Farben dargestellt. Der Zustand kann nur durch Betätigen des Reset-Schalters verlassen werden.

Im MOVE-Zustand werden Benutzereingaben (Links, Rechts, Rotation, schnelles und langsames Abwerfen) überprüft und verarbeitet. Langsames Abwerfen sowie Links- und Rechtsbewegungen verändern lediglich den Referenzpunkt. Bei einer Rotation hingegen wird der Koordinatenvektor des Tetriminos im Uhrzeigersinn mit den folgenden Formeln angepasst:

$$x = y'$$
$$y = -x'$$

Es gibt zwei Möglichkeiten, den MOVE-Zustand zu verlassen:

Zeitbasiert: Wenn der Tick-Zähler die dynamisch nach Punktestand festgelegte Zeitdauer erreicht , wechselt der Zustand in FALL. Das aktuelle Tetrimino wird dann um einen Block nach unten verschoben, sofern keine Kollision auftritt. In diesem Fall wird myblock aktualisiert, und der Zustand wechselt zurück in MOVE. Wenn jedoch eine Kollision erkannt wird, bleibt die Position unverändert, das Tetrimino wird in das Speicherfeld übernommen, und der Zustand wechselt in LOCK.

HARDDROP: Wird diese Eingabe erkannt, verhält sich das System wie im FALL-Zustand, jedoch wird der Tetrimino kontinuierlich um eine Zeile nach unten bewegt, bis keine weitere Bewegung möglich ist. Dann wird die letzte gültige Position gespeichert, und der Zustand wechselt in LOCK.

Im LOCK-Zustand erfolgt lediglich ein Synchronisationstakt zur Aktualisierung des gespeicherten Spielfelds.

Anschließend folgt der CHECK-Zustand, in dem das Spielfeld zeilenweise von unten nach oben überprüft wird, ob eine komplette Zeile vorhanden ist. Wird eine vollständige Zeile erkannt, wird sie markiert und es erfolgt ein Übergang in den

FLASH-Zustand. Hier blinkt die entsprechende Zeile 3 Mal für jeweils 500 ms, um dem Spieler die erzielten Punkte visuell anzuzeigen. Gleichzeitig wird der Punktestand erhöht.

Nach dem Blinken wechselt der Zustand in CLEAR, wo die gefüllte Zeile gelöscht und alle darüberliegenden Zeilen um eine Position nach unten verschoben werden. Da sich dadurch das Spielfeld verändert hat, wird erneut in den LOCK-Zustand gewechselt, um die Synchronisation sicherzustellen. Diese Schleife wiederholt sich, bis alle betroffenen Zeilen verarbeitet wurden.

Schließlich kehrt die FSM zurück in den SPAWN-Zustand, um ein neues Tetrimino zu erzeugen.

4.2 CLICK

Dieser Automat steuert das Tonausgabe-Modul.

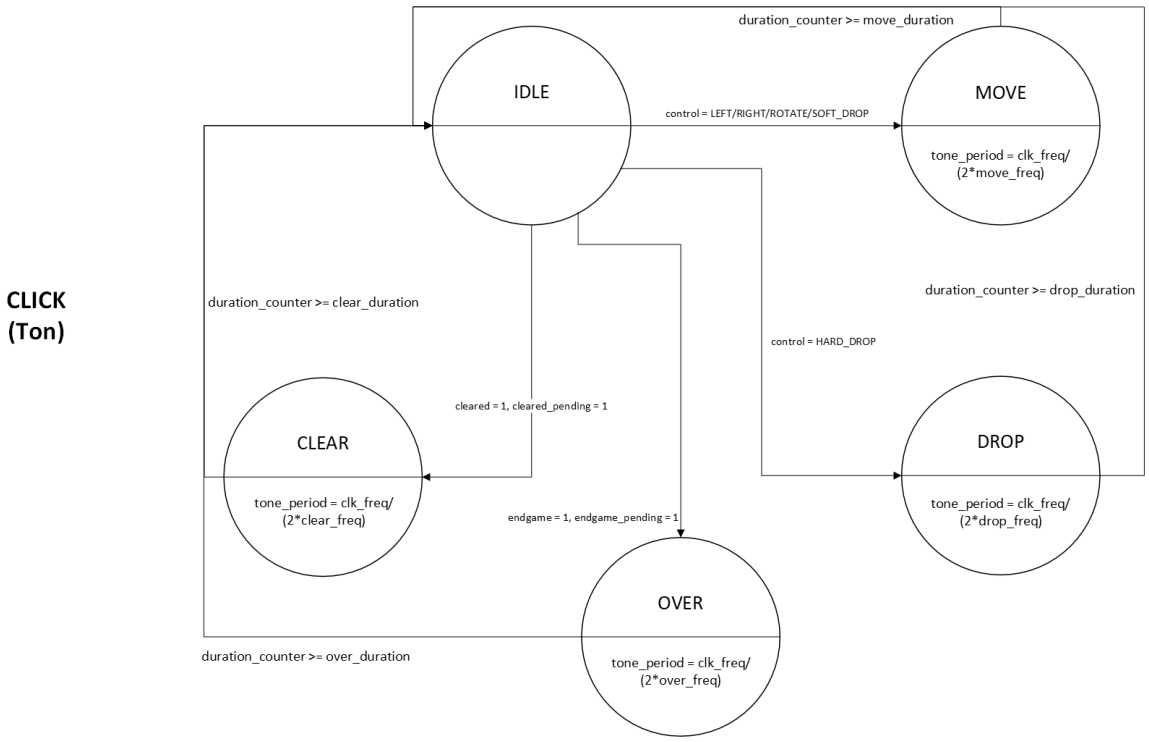

Abbildung 10: Tonausgabelogik

Ausgehend vom Zustand IDLE kann über einen Tastendruck oder durch die Signale CLEAR bzw. GAMEOVER in den entsprechenden Zustand gewechselt werden. In diesen Zuständen wird eine Periodendauer festgelegt, die zur Erzeugung einer Rechteckwelle dient, mit der der Buzzer angesteuert wird.

Nach einer jeweils vordefinierten Zeitdauer wird der aktuelle Zustand automatisch verlassen, und die FSM kehrt zurück in den IDLE-Zustand.

5 Stand

5.1 Features

Im Vergleich zum Zwischenstand konnten die meisten der ursprünglich geplanten Funktionen erfolgreich umgesetzt werden. Das Spiel ist voll funktionsfähig, und alle wesentlichen Komponenten arbeiten wie vorgesehen.

Folgende Funktionen wurden zusätzlich zum Kernstück erfolgreich realisiert:

- Dynamische Fallgeschwindigkeit der Tetriminos, abhängig vom aktuellen Punktestand. Dadurch steigt der Schwierigkeitsgrad im Laufe des Spiels kontinuierlich an.
- Interaktiver Klick-Ton bei Tastendruck, Zeilenlöschung und Game Over über den integrierten Buzzer.
- Punktzahlanzeige auf dem 7-Segment-Display des Basys3-Boards.

Eine Funktion, die bislang nicht umgesetzt wurde, ist die farbkodierte Darstellung der Tetriminos auf dem Display. Eine mögliche Umsetzung wäre jedoch vergleichsweise einfach realisierbar, etwa durch die Einführung eines zusätzlichen 2D-Vektors mit der Größe 10x20, wobei jede Zelle 3 Bit zur Darstellung der Tetrimino-ID enthalten könnte. Diese Identifikatorzahl könnte anschließend farblich codiert und beim Ausgabe berücksichtigt werden.

Ein zusätzliches Modul zur Melodieausgabe wurde programmiert, welches die bekannte Tetrismelodie abspielt. Aufgrund von Zeitmangel konnte dieses Modul jedoch nicht vollständig getestet und integriert werden.

Es besteht auch die Möglichkeit, den nächsten kommenden Tetrimino auf dem Bildschirm anzuzeigen, um dem Spieler eine bessere taktische Planung zu ermöglichen. Die dafür nötigen Datenstrukturen sind bereits vorhanden und könnten relativ einfach eingebunden werden.

Falls die Ressourcen des Boards es zulassen, wäre zudem ein Zwei-Spieler-Modus denkbar. Dafür müsste ein zweiter Controller mit fünf Tasten angeschlossen und die Spiellogik entsprechend erweitert werden. Dieses Feature würde den Funktionsumfang deutlich steigern und das Projekt um einen kompetitiven Aspekt verbessern.

Trotz dieser offenen Erweiterung befindet sich das Projekt in einem stabilen, funktionsfähigen Zustand. Alle Kernfunktionen wurden erfolgreich implementiert, getestet und erfüllen die Anforderungen an ein spielbares Tetris auf dem FPGA.

5.2 Herausforderungen

Im Laufe der Umsetzung des Projekts traten verschiedene technische und konzeptionelle Herausforderungen auf, die ein tieferes Verständnis für die Architektur und VHDL erforderten.

Erstens kam es ziemlich früh an, dass das Projekt beim Syntheseprozess die LUT Ressourcen auf dem Board erreicht hatte. Der Versuch, große Vektoren wie das 200-Zellen-Blockfeld direkt miteinander zu vergleichen, führte schnell an die Grenzen der auf dem Board verfügbaren LUTs. Zur Lösung wurde auf FOR-Schleifen zurückgegriffen, um die Vergleiche sequentiell und ressourcenschonend durchzuführen. Auch auf verschachtelte Schleifen mit "break"wurde verzichtet, da diese in VHDL nicht direkt unterstützt werden und als nicht Synthese-freundlich betrachtet wurde. Stattdessen wurde der Hard-Drop als eigener Zustand in die FSM ausgelagert, um das Verhalten schrittweise und steuerbar umzusetzen.

Zweitens gab es Synchronisationsproblem beim Feldspeichern. Nachdem ein Tetrimino im Zustand FALL auf dem Spielfeld gespeichert wurde, wird er im Zustand CHECK nicht erkannt. Grund dafür war, dass im gleichen Taktzyklus bereits ein Lesevorgang stattfand, bevor die Daten im RAM aktualisiert waren. Die Lösung bestand darin, einen neuen Zustand LOCK einzuführen, der einen zusätzlichen Takt zur Synchronisierung bereitstellt. Dadurch konnte die Datenkonsistenz sichergestellt und das Feld zuverlässig aktualisiert werden.

Die folgende Übersicht zeigt die Ressourcennutzung des finalen Projektstandes. Insgesamt liegt die Auslastung des FPGAs in einem angemessenen Bereich. Weder die Anzahl der genutzten Look-Up Tables (LUTs) noch der Flip-Flops oder sonstiger Komponenten überschreitet kritische Grenzen. Somit bleibt ausreichend Spielraum für eventuelle Erweiterungen oder Optimierungen.

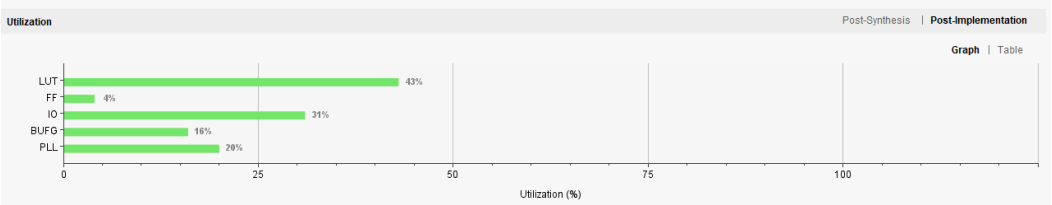

Abbildung 11: Synthese Ergebnis

6 Reflexion

Am Ende des Projekts ist das Spiel voll funktionsfähig und zeigt keine offensichtlichen Fehler. Diese erfolgreiche Umsetzung war nur möglich durch konsequentes Arbeiten, sorgfältige Planung und eine strukturierte Herangehensweise. Besonders deutlich wurde, wie wichtig Disziplin und Ausdauer bei einem komplexen Projekt wie diesem sind – ohne sie wäre das gewünschte Ergebnis kaum erreichbar gewesen.

Obwohl während der Entwicklung zahlreiche Probleme aufgetreten sind, haben gerade diese Herausforderungen zu einem tieferen Verständnis beigetragen. Man lernt mehr durch das Lösen unerwarteter Fehler als durch eine rein fehlerfreie Umsetzung. Jeder Bug zwang dazu, die eigene Herangehensweise zu hinterfragen und die Funktionsweise der FSM und der Spielmechanik besser zu verstehen.

Darüber hinaus konnte ich wichtige Erfahrungen im Strukturieren und Planen vom solchen Projekt sammeln. Insbesondere das Zerlegen in sinnvolle Module, das Festlegen klarer Zuständigkeiten und das frühzeitige Testen einzelner Komponenten haben sich als essenziell bewiesen. Ebenso wurde deutlich, wie bedeutend eine saubere Dokumentation und Versionenkontrolle durch Git ist, sowohl für die eigene Orientierung als auch für eine mögliche spätere Weiterentwicklung oder Übergabe des Projekts.

Zusammenfassend war das Projekt nicht nur eine technische Herausforderung, sondern auch eine wertvolle Lektion in Projektmanagement, Problemlösung und persönlicher Arbeitsweise. Die gesammelten Erfahrungen werden mit Sicherheit auch in zukünftigen Projekten von großem Nutzen sein.

Literatur

[1] Baliika, 'Tetris on FPGA', *GitHub Repository*, [Online]. Available: https://github.com/baliika/fpga-tetris/tree/main.

[2] Digilent Inc., 'BASYS3 FPGA Board', [Online]. Available: https://reference.digilentinc.com https://www.amd.com/content/dam/amd/en/documents/university/aup-boards/XUPBasys3/documentation/Basys3_rm_8_22_2014.pdf.

[3] SECONS Ltd., 'Standard VGA timings', [Online]. Available: http://www.tinyvga.com/vga-timing/640x480@60Hz https://forum.digikey.com/t/vga-controller-vhdl/12794.

[4] Wikipedia contributors, 'Linear-feedback shift register', Wikipedia, The Free Encyclopedia, [Online]. Available: https://en.wikipedia.org/wiki/Linear-feedback_shift_register.

Abbildungsverzeichnis

1 Basys3 Board 4
2 Tetrimino-Spawnposition und Drehachse 5
3 Steuerungsoptionen 6
4 Blockschaltbild 7
5 Testbench mit dem Zufallszahlgenerator 9
6 Tetrimino Identikfikatorzahl 10
7 VGA Controller Einstellung 11
8 Synchronisations- und Pufferzeit 12
9 Hauptlogik 14
10 Tonausgabelogik 16
11 Synthese Ergebnis 18