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

Projektbericht zu Digitale Systeme

Programmierung eines Tetrisspiels mittels FPGA

Zwischenbericht DY-Projekt	Eingegangen am: 26.5.2025	Protokollführer: Zhi Hao Tan
Vortragstag 21.5.2025	Testat:	
Dozent: Prof. Dr.-Ing. Robert Fitz		

Inhaltsverzeichnis

Abbildungsverzeichnis

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`.

1 Einführung

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.

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

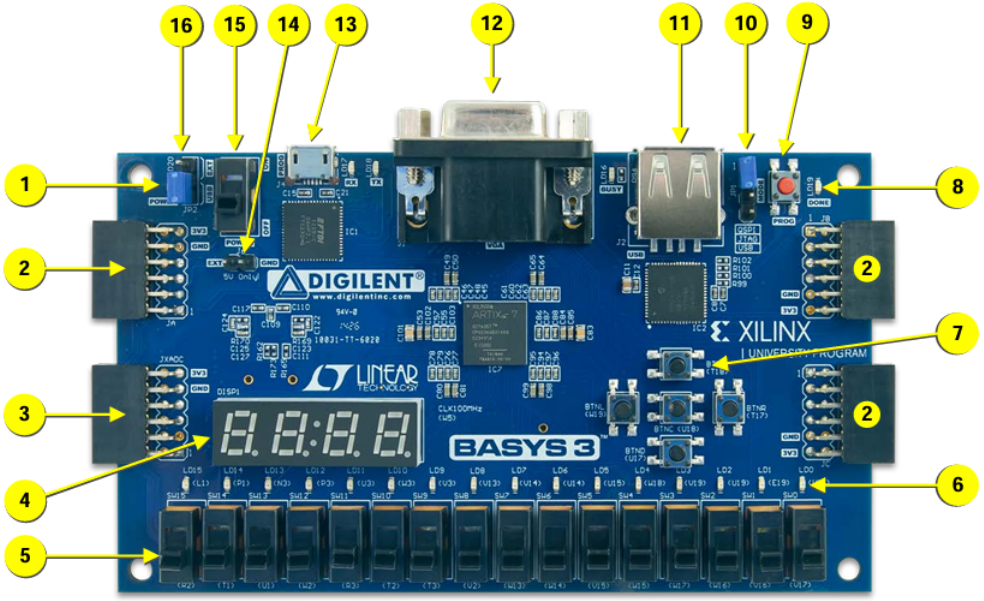

Figure 1. Basys3 FPGA board with callouts.

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 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. Daher kann dieser auf praktisch allen VGA-Monitoren problemlos angezeigt werden. 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:

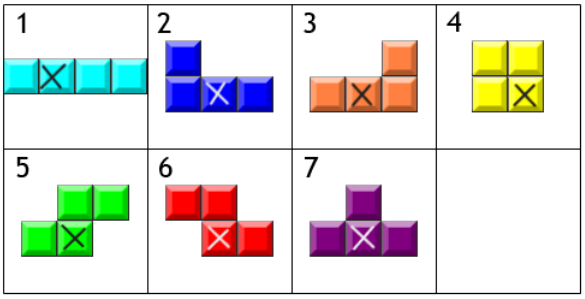

Abbildung 2: Tetrimino-Spawnposition und Drehachse

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 und soll fest sein. Allerdings lässt das Quadrat-Stück sich nicht drehen, weil es achsensymmetrisch ist.

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 Zeit bis zum Ende des Projekts sehr wahrscheinlich 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.

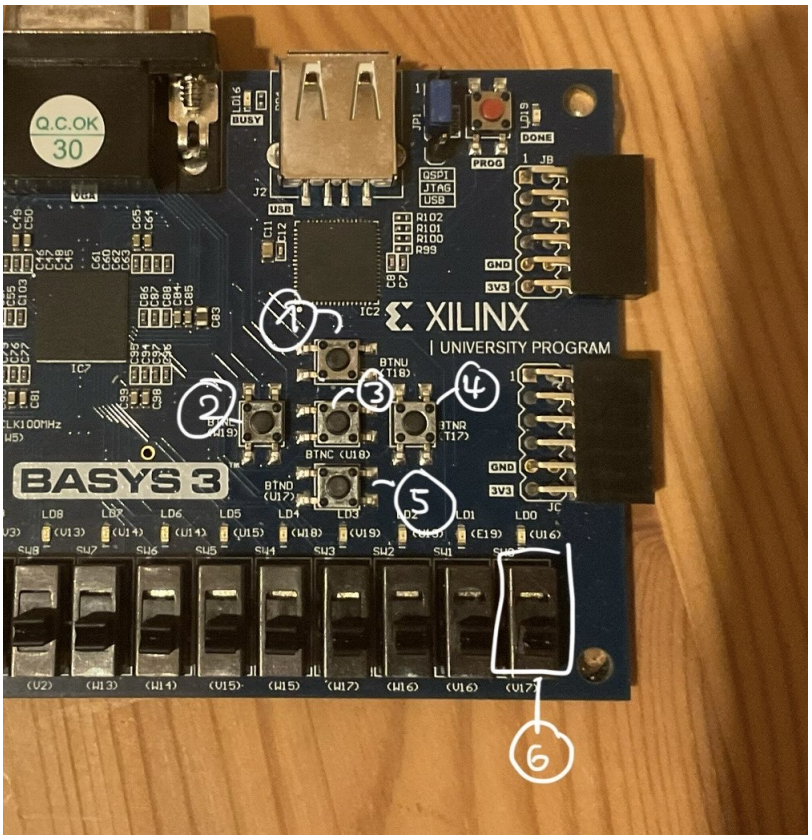

Abbildung 3: Steuerungsoptionen

Die Steuerung:
1. Obere Taste - Tetrimino sofort zum Boden bewegen (schnelles Abwerfen)
2. Linke Taste - Tetrimino um einen Block nach links bewegen
3. Mittlere Taste - Tetrimino 90° um die Drehachse im Uhrzeigersinn rotieren
4. Rechte Taste - Tetrimino um einen Block nach rechts bewegen
5. Unterer Taste - Tetrimino um einem Block nach unten bewegen (langsames Herunterfallen)
6. 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.

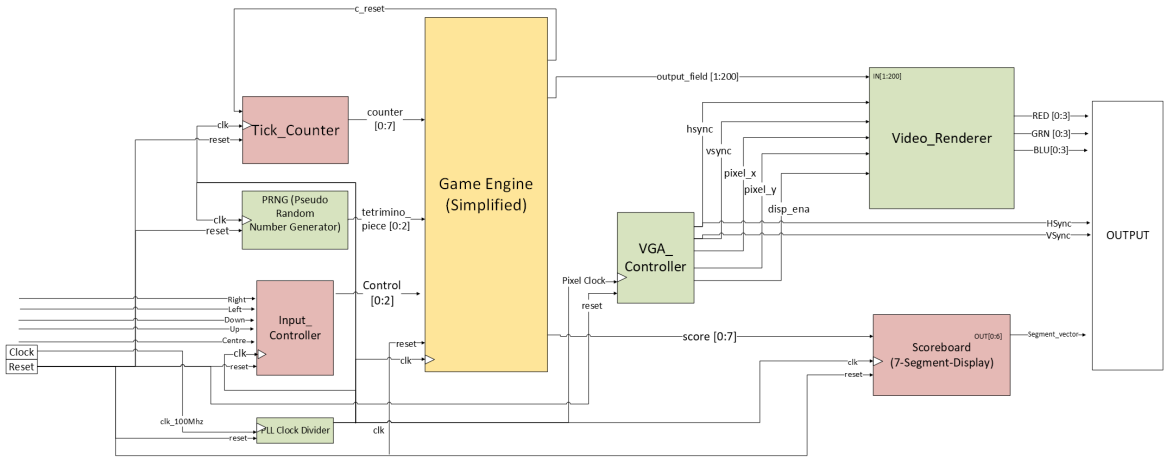

Abbildung 4: Blockschaltbild

Die Farbe zeigt an, wie der aktuelle Stand der Module ist.
Rot : noch nicht angefangen
Gelb : in Arbeit
Grün : (fast) Fertig

Innerhalb vom `Game Engine` ist die Logik etwas komplizierter, wobei hier ungefähr alle Signale zu sehen sind:

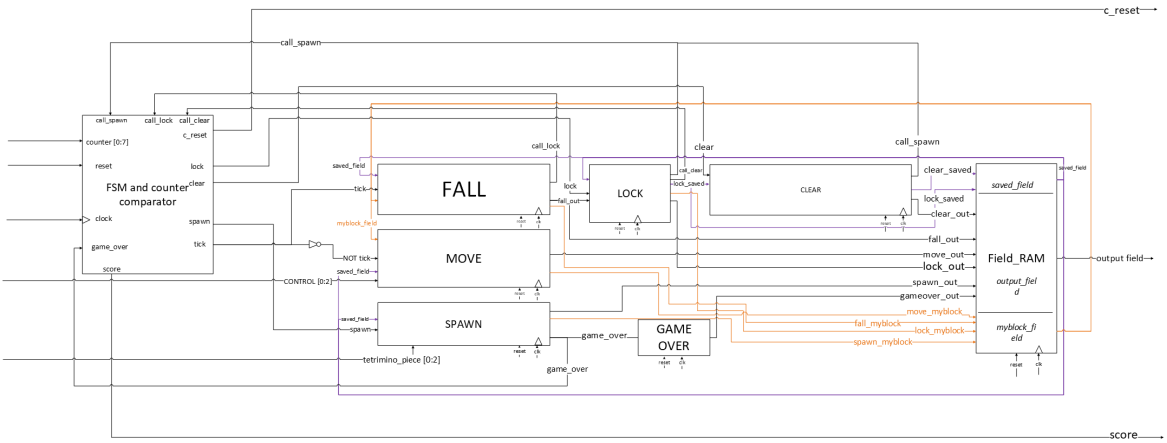

Abbildung 5: Blockschaltbild: `Game Engine`

Die selben Bilder mit einer höheren Auflösung sind im Anhang zu finden.

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 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 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.

3.1.3 Game Engine

In diesem Block liegt der Kern des Spiels, wobei die Zustandsmaschine sich hier befindet.

Das tick_counter wird zunächst kontrolliert, nach dem Erreichen eines definierten Zeitpunktes zurückgesetzt und ein FALL-Signal gesendet. Das FALL-Signal ist komplementär zur Aktivierung des MOVE-Block, und wird zurückgesetzt nach dem Beenden des FALL-Zustandes. Das heißt, FALL und MOVE , sowie andere Sub-Blöcke können nicht gleichzeitig arbeiten. Der SPAWN-Block wird nur nach dem FALL oder

LOCK/CLEAR aktiviert und niemals durch MOVE. SPAWN nimmt den Tetrimino-ID-Vektor und vergleicht ihn mit einer Lookup-Tabelle von Blocktypen, schließlich zeichnet es ihn auf eine myblock_next Zwischenspeicher-Variabel und prüft nach, ob der Schritt zulässig ist. Die Variabel wird nach dem Prozess gelöscht.

Die Zustandsmaschine sorgt hier für die Kontrolle am Ausgang, was im Abschnitt FSM zu finden ist. Allerdings ist hier wichtig, dass alle Zustände unterschiedlich befüllte 10x20 Spielfelder (saved bzw. das hart-gespeichertes Feld; output bzw. das Ausgangsfeld; myblock bzw. das aktuelle Blockfeld) an dem Feld-Arbeitsspeicher liefern, auf die immer bei Zustandsänderung zugegriffen werden kann.

3.1.4 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. 10ms zu beobachten. Dadurch kann das Signal entprellt und ein stabiler Tastendruck erkannt werden.

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

3.1.5 PRNG - Pseudorandom Number Generator

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

Dabei kommt ein Linear Feedback Shift Register (LFSR) 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 27-Bit-Vektor, der mit jedem Takt weitergeschaltet wird. Zusätzlich wird ein 3-Bit-Vektor durch XOR-Verknüpfung bearbeitet und um eine Position nach links verschoben. Das neu eingefügte Bit kommt aus dem laufend aktualisierten 27-Bit-Vektor. Auf diese Weise entsteht eine Bitkombination, die für die Auswahl eines Tetriminos genutzt werden kann.

Da jeder Tetrimino durch einen eindeutigen Identifikator dargestellt wird, kann der resultierende 3-Bit-Vektor direkt einem der sieben Tetrimino-Typen zugeordnet werden. Somit wird bei jedem Spawnen eines neuen Tetriminos eine pseudozufällige, aber auch reproduzierbare Auswahl getroffen.

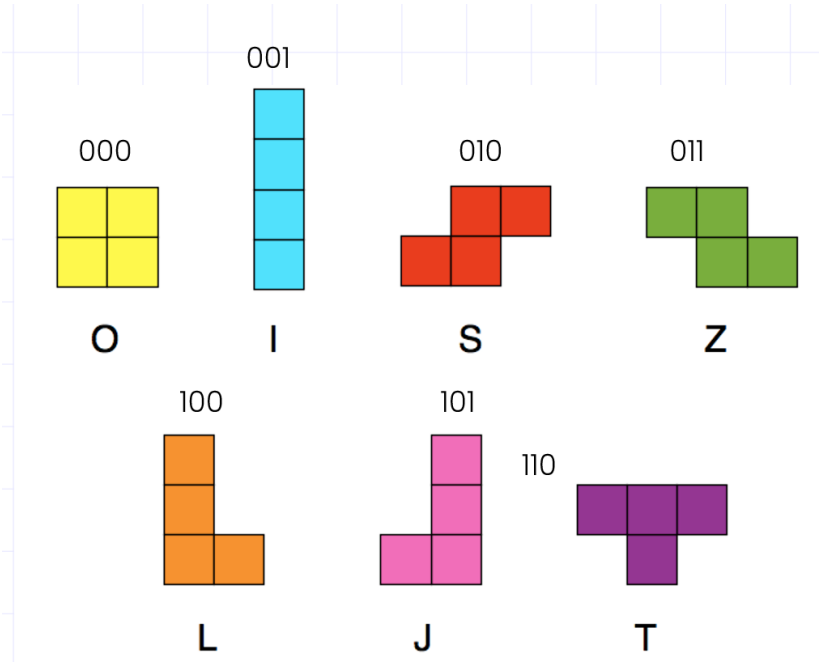

Abbildung 6: Tetrimino Identikfikatorzahl

3.1.6 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. 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

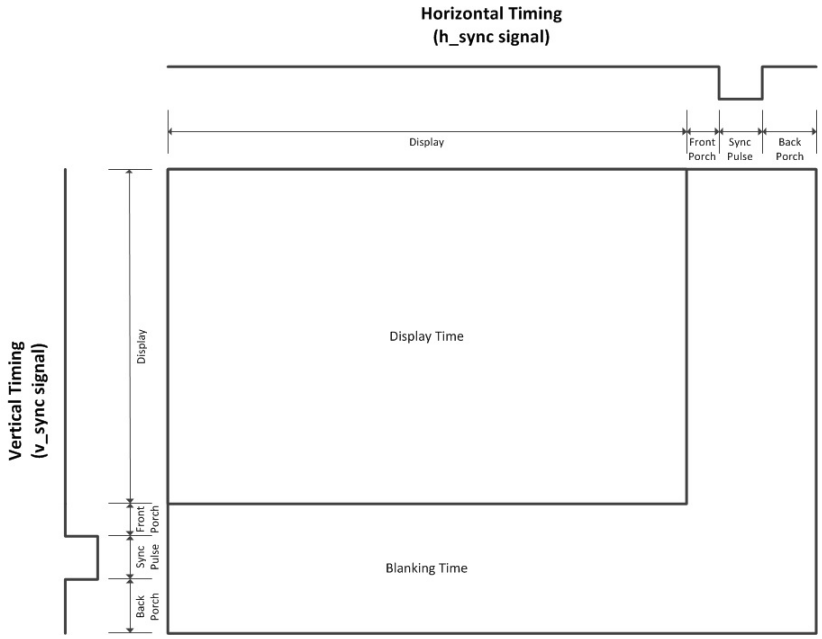

Abbildung 8: Synchronisations- und Pufferzeit

3.1.7 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 Bildschirmko-

ordinaten 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.

3.1.8 Scoreboard

Dies ist der Verantwortliche für die Umsetzung der Punktzahl auf dem eingebauten 7-Segmente-Display. Der Punktestand wird intern als 8-Bit-Wert (0–255) 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.

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.

4 FSM

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.

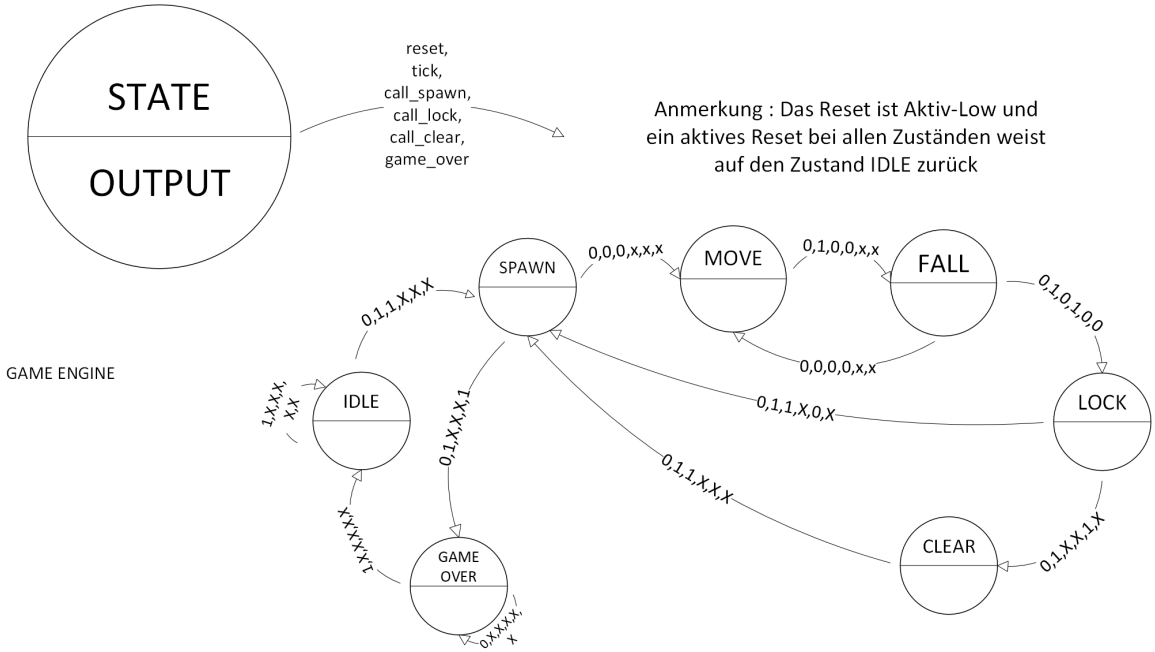

Abbildung 9: Hauptlogik

Jeder Zustand repräsentiert eine spezifische Phase im Spiel, z.B. SPAWN, MOVE, FALL, LOCK, CLEAR oder GAMEOVER. 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 den Operatorvergleich verwendet)

saved_field - gespeichertes Feld

output_field - Ausgabefeld

Zu Beginn des Spiels befindet sich die FSM im SPAWN-Zustand, in dem ein neuer Tetrimino generiert und auf dem Spielfeld platziert wird. Anschließend wechselt die FSM in den MOVE-Zustand, wo auf Benutzereingaben gearbeitet wird (Links/Rechts/Rotation/Abwurf).

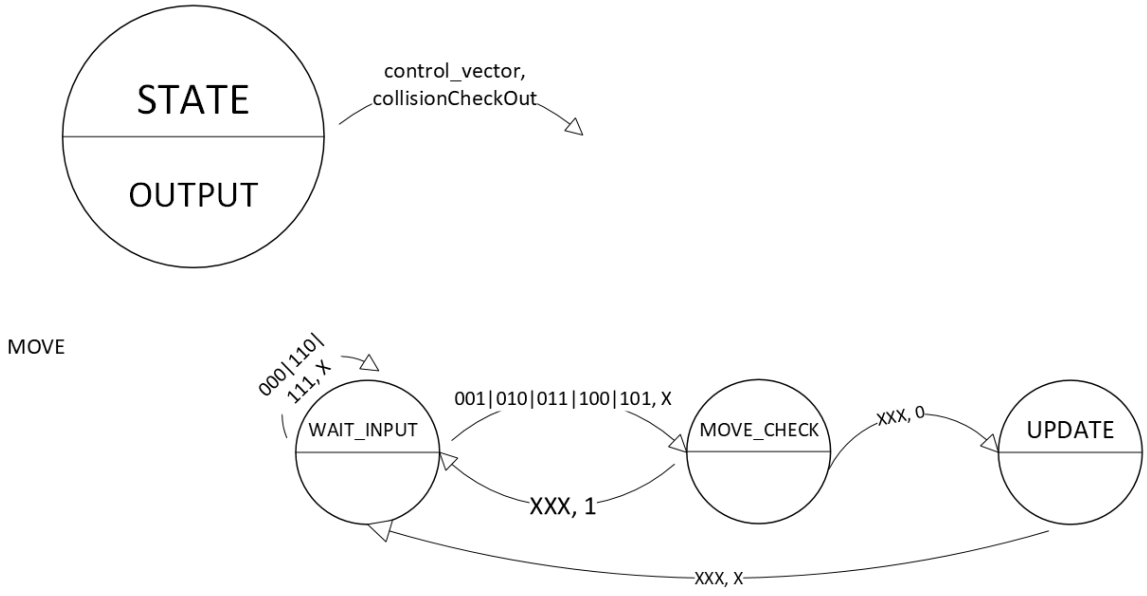

Abbildung 10: MOVE-Logik

Innerhalb des übergeordneten MOVE-Zustands existiert eine separate FSM, die für die Steuerung und Validierung von Spielerbewegungen zuständig ist. Diese FSM verarbeitet Benutzereingaben wie Links-, Rechtsbewegung, Rotation oder Abwerfen des aktiven Tetriminos. Zunächst gelangt sie in einen WAIT-Zustand, in dem die Eingaben abgewartet und zwischengespeichert werden. Sobald eine Eingabe erkannt wurde, prüft die FSM die Gültigkeit, indem das Operator 'OR' zwischen myblock_next und saved_field genutzt wird – das heißt, ob der Tetrimino bei der Bewegung nicht mit anderen Blöcken oder den Spielfeldgrenzen kollidieren würde. Ist die Bewegung erlaubt, wechselt die FSM in den UPDATE-Zustand, in dem die tatsächliche Position des Tetriminos im Spielfeld aktualisiert wird bzw. output_field durch 'AND' Operator den myblock_next und saved_field elementweise verglichen werden. Falls die Bewegung ungültig ist, kehrt sie ohne Änderung zurück in den WAIT-Zustand, und den output_field bleibt unverändert. Eine Anmerkung zu MOVE ist, dass eine Eingabe als ungültig gilt, wenn sich der Drehachseblock beim Drehbefehl am Rand befindet und wenn sich irgendein Block vom Tetrimino beim Links/Rechts schieben am Rand befindet.

Parallel dazu wird durch ein Taktsignal der Übergang in den FALL-Zustand gesteuert, bei dem der Tetrimino automatisch nach unten bewegt wird. Dies entspricht der Gültigkeitsprüfung des MOVE-Zustandes, nur dass es diesmal mit dem `Tick` synchron läuft.

Wenn der Tetrimino keinen Platz mehr hat um weiterzufallen bzw. die Kontrolle "saved_field OR myblock_next"1 liefert, wechselt der Zustand in LOCK, wo der Tetrimino im saved_field fixiert wird. Danach überprüft CLEAR, ob vollständige Linien

existieren, die entfernt werden sollen, und aktualisiert den Punktestand entsprechend. Nach der Entfernung wird das ganze Raster oberhalb von der Reihe um einen Block nach unten verschoben.

Falls keine Kollision beim SPAWN erkannt wird, beginnt der Zyklus erneut mit einem neuen Tetrimino. Sollte jedoch beim SPAWN keine Platzierung mehr möglich sein, bzw. nach der Erzeugung des neuen Blocks in myblock_next durch einen Vergleich mit saved_field fehlgeschlagen hat , wechselt die FSM in den GAMEOVER-Zustand.

Der GAMEOVER-Zustand gibt eine noch nicht festgestellte Anzeige aus und lässt sich nicht verlassen, ohne dass der Spieler den Reset-Switch auslöst.

5 Fortschritt

Das Projekt befindet sich im Mittel des gegebenen Zeitskalars.

5.1 Aktueller Stand

Bisher ist der Großteil des Konzepts für die wesentlichen Bestandteile des Spiels ausgearbeitet. Das eigentliche Programmieren hat nun begonnen, wobei im weiteren Verlauf umfangreiche Tests durchgeführt werden sollen, um die Funktionalität und Integrität des Spiels sicherzustellen. Die VGA-Schnittstelle sowie die Darstellung des 200 Zellen umfassenden Spielfelds funktionieren bereits zuverlässig. Aktuell wird an der `Game Engine` gearbeitet, ebenso wie an essenziellen Modulen wie dem `Input Controller` und dem `Tick Counter`.

5.2 Änderungen

Zu Beginn des Projekts war auch Jason Putrawidjaya Teil des Teams. Da er jedoch nur über begrenzte VHDL-Kenntnisse verfügte und aufgrund anderer Verpflichtungen wenig Zeit für die Projektarbeit aufbringen konnte, verlief der Fortschritt in der Anfangsphase etwas langsamer als geplant. Er beteiligte sich hauptsächlich an der Ausarbeitung der Präsentation, zeigte dort auch Engagement, konnte jedoch am Ende nicht im gewünschten Umfang zum Projekt beitragen. In Absprache mit dem Team entschied er sich daher, das Projekt abzugeben, woraufhin Zhi Hao Tan seine Aufgaben übernommen hat.

Der Ausstieg eines Teammitglieds in einer fortgeschrittenen Projektphase stellt stets eine Herausforderung dar. Insbesondere kann es zu zeitlichen Verzögerungen kommen, da das verbleibende Teammitglied zusätzliche Aufgaben übernehmen muss. In diesem Fall führte die Umstrukturierung zunächst zu einem erhöhten Arbeitsaufwand, insbesondere in der Integration offener Aufgaben. Dennoch konnte das Projekt durch sorgfältige Planung und effektives Zeitmanagement weiterhin zielgerichtet fortgeführt werden.

5.3 Nächste Schritte

Nach Abschluss der Programmierphase sollen intensive Tests und Bugfixing-Maßnahmen durchgeführt werden. Zudem ist die Implementierung zusätzlicher Features geplant, um die Benutzererfahrung weiter zu verbessern.

Dazu gehört unter anderem die farbliche Kodierung der Tetriminos auf dem Display, um die einzelnen Spielsteine leichter unterscheiden zu können. Außerdem soll das Timing zwischen aufeinanderfolgenden Fall-Zuständen reduziert werden, um den Schwierigkeitsgrad zu erhöhen und das Spiel herausfordernder zu gestalten. Ein wei-

teres geplantes Feature ist die Musikausgabe: Ein Buzzer soll verwendet werden, um die bekannte Tetris-Melodie abzuspielen. Die Geschwindigkeit der Musik könnte sich dabei dynamisch an die aktuelle Punktzahl anpassen. Schließlich ist vorgesehen, das Punktestand-Display von der 7-Segment-Anzeige auf den Bildschirm neben das Spielfeld zu verlagern, um die Benutzeroberfläche noch ansprechender und informativer zu gestalten.