Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi
Nigde Omer Halisdemir University Journal of Engineering Sciences

ISSN: 2564-6605

Araştırma / Research

A FULLY EMBEDDED TETRIS GAME APPLICATION IN VHDL

Sadiye Nergis TURAL POLAT[1,*]

[1]*Yıldız Teknik Üniversitesi, Eketronik ve Haberleşme Mühendisliği Bölümü, Davutpaşa Mah. 34220, Esenler/İSTANBUL*

ABSTRACT

In this paper, a Tetris game code which runs on an FPGA development and education board is synthesized. To this end, an original Tetris game code that works solely on the FPGA board is designed in VHDL hardware description language. The novelty of this design lies in the fact that the Tetris code does not use any SRAM modules for storage unlike other embedded Tetris implementations published to date. What is more, the code uses a tile-mapped scheme in which the code groups certain pixels as a tile. Thus, a 1-bit register is enough for the current value of all the pixels that fall in the tile which improves the speed of the code and simplifies the design procedure. The code is written using VHDL hardware description language. Though Altera DE0 development and education board is used for the implementation, thanks to the usage of only the standard VHDL functions, the code can run on any other FPGA boards.

Keywords: FPGA Tetris, embedded system design, Tetris code in VHDL, Game coding in VHDL

VHDL İLE TAM GÖMÜLÜ BİR TETRİS OYUNU GERÇEKLEŞTİRMESİ

ÖZET

Bu çalışmada, bir FPGA geliştirme ve eğitim kiti üzerinde tam gömülü olarak çalışan bir Tetris oyun kodu gerçeklemesi yapılmıştır. Bunun için, FPGA kiti üzerinde bağımsız olarak çalışan, VHDL donanım tanımlama dilinde orijinal bir Tetris oyun kodu tasarlanmıştır. Bu tasarımın özgün yanı, şimdiye kadar bu konuda yapılmış birçok çalışmadan üstün olarak, tasarlanan kodun veri tutma işlemi için herhangi bir SRAM modülü kullanmamasıdır. Kodun yazımında karo-haritalamalı şema denilen, bir grup pikselin tamamını bir karo olarak tanımlayan bir haritalama kullanılmaktadır. Bu şema ile, ilgili karo içindeki piksellerin tamamı için 1-bit bilginin tutulması yeterli olduğundan kod tasarımı oldukça kolaylaşmakta ve aynı zamanda kodun hızlı çalışması sağlanmaktadır. Oyun kodu VHDL donanım tanımlama dilinde yazılmıştır. Sistem olarak Altera DE0 geliştirme ve eğitim kiti kullanılmıştır ancak yazılan kodda yalnız standart VHDL fonksiyonları kullanıldığından kod diğer FPGA kitleri üzerinde de çalışabilmektedir.

Anahtar kelimeler: FPGA Tetris, Gömülü sistem tasarımı, VHDL ile Tetris kodu, VHDL ile oyun kodu

1. INTRODUCTION

Image and video processing technology have had an exponential growth in recent years thus providing us with ample exciting means such as digital cinema and HDTV. Especially, the resolutions have increased dramatically over the last years. Therefore it has become crucial to process large amounts of data in real time. The Field Programmable Gate Array (FPGA) technology supplements a suitable framework for designing and implementing complex systems [1-4]. Thanks to the availability of one-to-one mapping and the spatial and temporal parallelism of the FPGAs, they are often used to implement real-time image processing applications [6-10]. One such application is the game code implementation on FPGAs such as Tetris [11], Connect6 [12-13], Blokus duo [14-16], Trax [17] and Pong [18-19]. In [11], Kuo et al. design a tetris game code using the Spartan 3AN FPGA starter board of Xilinx. The code uses the dual-port block memory (BRAM) units for display and game code controls. For the Connect6 Solver, the VHDL code is implemented on a DE2-115 Development and Education board having the Cyclone IV processor [12]. The code again uses the BRAM units (50.7%) and consumes the 92.6% of the logic elements. The Blokus-Duo solver designed in [14] is implemented on the Arria II GX Development kit of Altera. The code uses approximately 70% of the Combinational ALUTs, 29% of the Total Block Memory Bits and 68% of the DSP blocks. The Trax solver design given in [17] again uses the Arria II GX processor of Altera and consumes 80% of the Combinational

* Sorumlu yazar / Corresponding author, e-posta / e-mail: nergis@yildiz.edu.tr
Geliş / Recieved: 05.02.2019 Kabul / Accepted: 27.12.2019 doi: 10.28948/ngumuh.522790

ALUTs, 63% of the Total Block Memory Bits and 25% of the dedicated logic registers. The Pong game code is implemented on a Nexys 2 Development board which has Spartan 3E FPGA in [18].

The game of Tetris has been likewise very popular in the scientific community since its invention and has been used in a wide range of research areas and applications such as computational intelligence [20-21], linear algebra [22], optimization [23] and cognitive science [24].

In this paper a fully-embedded Tetris game code that runs on an FPGA development board is designed. To this end, all parts of the game code are written in VHDL and synthesized for the dedicated FPGA chip on the DE0 board using Altera Quartus II software. The DE0 Development and Education board is designed in a compact size with all the essential tools. The board is also chosen for its reasonably low price. It is equipped with Altera Cyclone III 3C16 FPGA device, which offers 15,408 Logic Elements, 1.15V~1.25V voltage supply, 516096 total RAM bits. The board provides 346 user I/O pins. The Modelsim program is used for the simulations.

The paper is organized as follows: The second chapter outlines the main Tetris game functions. The VHDL code that achieves the desired functions is given in chapter III and the results are summarized in chapter IV.

2. THE GAME OF TETRIS

Tetris is a puzzle video game originally designed and programmed by Alexey Pajitnov in the Soviet Union. He derived its name from the Greek numerical prefix tetra- (all of the game's pieces, known as Tetrominoes, contain four segments) and tennis, Pajitnov's favorite sport [25].

The playing field of the game is a rectangular vertical shaft called the “well” or “matrix” that consists of 20 rows and 10 columns. A random sequence of tetrominoes (Fig. 1)—shapes composed of four square blocks each—fall down the playing field. The player may repeatedly rotate the shape either clockwise or anticlockwise by 90 degrees, and move it left or right, one cell at a time. The shape falls until it comes to rest on either the bottom of the grid, or above an occupied cell. The cells making up the shape then become occupied, and no further manipulation is allowed. This is referred to as a shape “placement”. After each placement, a new random shape is generated, and the process is repeated. The objective of the game is to manipulate these tetrominoes to create a horizontal line of ten blocks without gaps. When such a line is created, it disappears, and any block above the deleted line is moved down one row. With every ten lines that are cleared, the game enters a new level. As the game progresses, each level causes the tetrominoes to fall faster, and the game ends when the stack of tetrominoes reaches the top of the playing field and no new tetrominoes are able to enter. All of the tetrominoes are capable of single and double clears. I, J, and L are able to clear triples. Only the I tetromino has the capacity to clear four lines simultaneously, and this is referred to as a "tetris" [25].

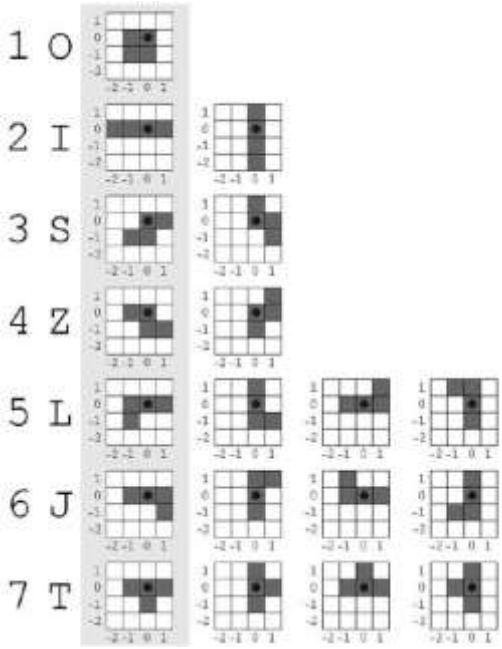

Figure 1. Seven main Tetris game pieces, also known as tetrominoes, and their rotated variations used in the VHDL Code

3. TETRIS GAME CODE IN VHDL

The designed VHDL code that realizes the desired game functions consists of three main parts. The first part generates the necessary clock signals for the game from the on-board 50 MHz clock signal of DE0 Board. The second part constructs the VGA image compatible to VGA Standards for the game display and the third part produces and controls the main Tetris Functions (Fig. 2).

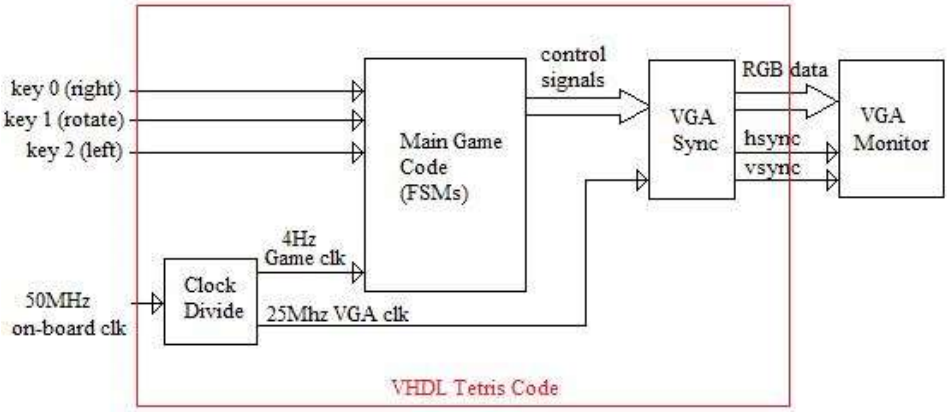

Figure 2. Main block diagram of the VHDL Tetris Code

3.1. System Clock and Game Code Clock

As stated above, the DE0 board has 50 MHz on-board clock. To be able to construct VGA image, it is necessary to have a 25MHz VGA synchronization clock [26]. What is more, to be able to obtain animated image sequences for game play, it is necessary to move the objects in the frame at an observable speed to human eye. After several tries, we set the game clock at 4Hz. This value is set as an initial value; one of the improvements of the code could be adding the speeding up procedure according to the completed full rows (i.e. levels of the game). This clock divide operations is obtained by using appropriate counters in the code.

3.2. VGA Image Construction and Synchronization

VGA (Video Graphics Array) video consists of sequential image frames moving in time. Each frame consists of vertically arranged rows of horizontal pixels. Every image frame is constructed by scanning the pixels from the top left row to the right until the end of line, then repeating the process for the next line until the end of the frame [27].

Standard VGA image has 640x480 pixel resolution and 60Hz refresh rate. These pixels are in the visible range, several more pixels are used for synchronization purposes; therefore the actual pixel size is 800x525. Hence the 25MHz pixel clock is obtained as

$$f_{VGA} = 800x525x60 \cong 25MHz \tag{1}$$

Figure 3 shows horizontal synchronization timing specification and number of pixels used for horizontal and vertical synchronization are used as given in [28]. The proper synchronization signals are obtained by using two counters for horizontal and vertical pixel positions in the code.

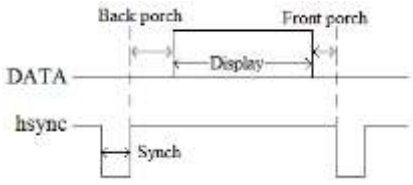

Figure 3. Horizontal timing specification of VGA image [28]

DE0 Board has 4-bit DAC for its VGA output RGB (Red, Green, and Blue) signals, therefore R, G and B signals are constructed as 4-bit signals in the VHDL code.

There is a pixel generation block in the VGA Sync block. The pixel generation block produce the 3-bit RGB signal which depends on the control signals from the main game code and the pixel x and y coordinates for each pixel in the VGA frame. For 640-by-480 VGA resolution, there are about 310k pixels on a screen. This translates to 930k memory bits for a RGB color display. To reduce the memory requirement, one alternative is to use a tile-mapped scheme [27]. In this scheme, a collection of bits are grouped to form a tile and each tile is treated as a display unit. Instead of wasting memory to store a mostly blank screen, the rectangular playing field is divided into 200 smaller square tiles each of which consists of 20×20 pixels (Fig. 4). Since one bit information (occupied tile or non-occupied tile) is sufficient for the state of each tile, it is sufficient to have a 200-bit register (a 20×10 array in VHDL code) instead of 400×200 = 80.000-bit data for each pixel. The current and next states of the playing field are stored in two 20×10 arrays in the code.

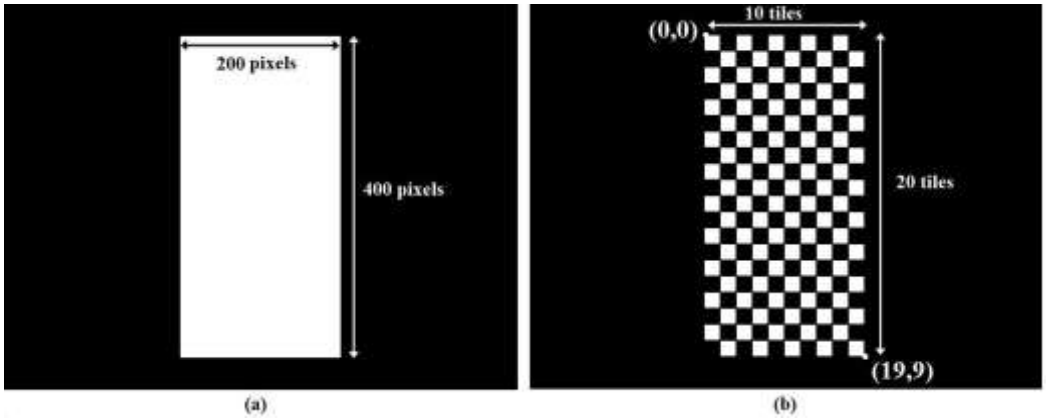

Figure 4. (a) bit-mapped (b) tile-mapped image construction

To control the movement of each Tetris piece, it is necessary to know the current location of the Tetris piece in the playing field. This is obtained by defining a center point (0,0) for each piece (Fig. 5). The target tiles for each move (right, left, rotate or down) are checked before each move to prevent overlapping from the playing field boundaries and to prevent collisions. Obviously, the target tiles change according to the shape and position of every Tetris piece. Then the 4×4 tile block moves to the target area if the target area is available. The 4×4 block definition is also useful when writing the updated playing field info to the 20×10 array at the end of each piece's fall.

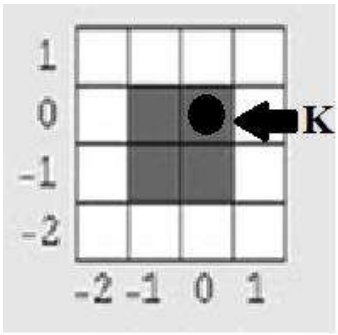

Figure 5. K center point of the Tetris piece

In addition to the playing field, a 4×4 tile field on the left of the playing area is similarly constructed to display the next Tetris piece (Fig. 6). Several scenarios of the game play are presented in Figure 6.

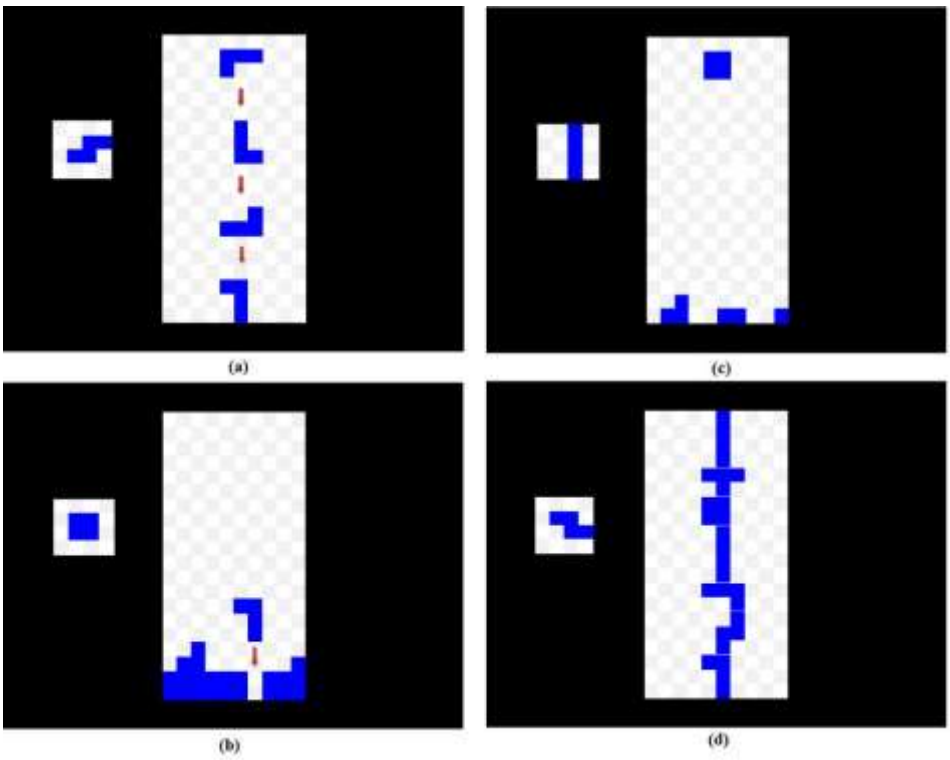

Figure 6. Several moments of the game (a) Move of one tetris piece (b) Just before the completion of a row (c) After completion of a row (d) End of the game

3.3. Illusion of Animation

When an object changes its location gradually in each frame, it creates the illusion of motion and becomes animated. To achieve this, we use registers to store the boundaries of an object and update its value in each VGA frame. In the Tetris game, the falling current shape object is controlled by two pushbuttons and can move to the right (key 0) and left (key 2) and can be rotated in the 90 degrees counterclockwise direction by another pushbutton (key 1) of the DE0 board.

3.4. Main Game Code

The main Finite State Machine of the program consists of eight states (zero, right, left, turn, down, change, minus, hold) and 14 state transitions. State diagram of the main FSM is given in Figure 7.

Zero state is the initial state. The program checks if any of the keys on the DE0 board are pressed by the user and the state transitions are done accordingly. Key 0 controls the right move (so if key 0 is pressed, the next state becomes right), key 1 controls the 90 degrees counterclockwise rotation (turn state) and key 2 controls the left move (left state) of the piece. If none of the keys are pressed, the next state is down. When the appropriate conditions are met (explained below) in every other state or when the reset switch is pressed, the program returns to the zero state.

When none of the buttons are pressed, or when left, right or rotate moves cannot be achieved because of the lack of the appropriate space, the program goes to the down state. If the bottom boundary have not been reached and if the target tiles below the Tetris piece are empty, the Tetris piece moves down one row and the next state becomes zero. Otherwise since the down movement cannot be possible (because of the boundary or collision) the piece keeps its position and the next stage becomes the change state.

Left state is obtained when key 2 is pressed. If the appropriate target tiles are empty and available (available means target tiles are within the field boundaries), the Tetris piece moves one column left of its current position and the next state becomes zero. When the left motion is not available the piece stays where it is and the next state is assumed to be down.

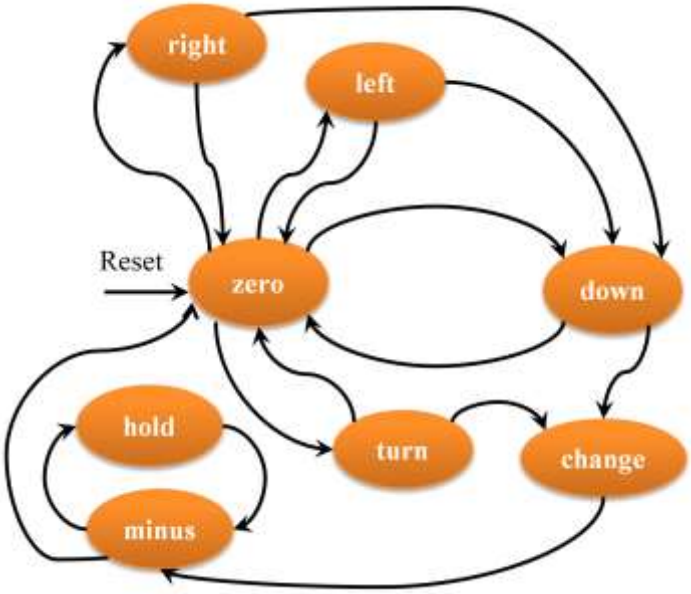

Figure 7. Main FSM State Diagram

Similarly, the right state is reached when the right button (key 0) is pressed. If the right target tiles are empty and available, the Tetris piece moves one column right of its current position and the next state becomes zero. When the right motion is not available the piece stays where it is and the next state is assumed to be down.

The FSM goes to the turn state when key 1 is pressed. If the target tiles empty and available, the Tetris piece rotates 90 degrees counterclockwise and the next state becomes zero. When the rotate is not available the next state becomes change.

Change state is obtained when a collision occurs or when the piece reaches the bottom boundary of the playing field. The piece stays where it is, the 20×10 matrix storing the playing field status is updated accordingly, a new random piece appears at the top of the field and the next state becomes minus to check the status of the playing field for completed rows.

In the Minus state the program checks if there is any completed rows in the playing field. To be able to delete multiple completed rows, the hold state is added. Thus, the program deletes one completed row, then holds this state to finish the computations necessary to evaluate if there is any other completed rows. Two smaller FSMs are used for detecting the full rows, deleting them and updating the relevant matrix values. If there are not any completed rows, the next state becomes zero.

The random retrieving of the next tetromino is achieved by a generated random number in the VHDL code according to [29] and selecting the appropriate tetromino assigned to the generated random number which is all realized in another FSM in the code.

The Flowchart of the Tetris code algorithm is provided in Fig. 8 and the state transitions are provided in Table 1.

The designed Tetris VHDL code is approximately 1300 lines long, the compilation of the code takes around 21 minutes on a P4 C2Duo 3GHz, 4GB RAM PC and the code uses close to 75% of the logic elements, 6% of the total pins and of course none of the total memory bits of the FPGA.

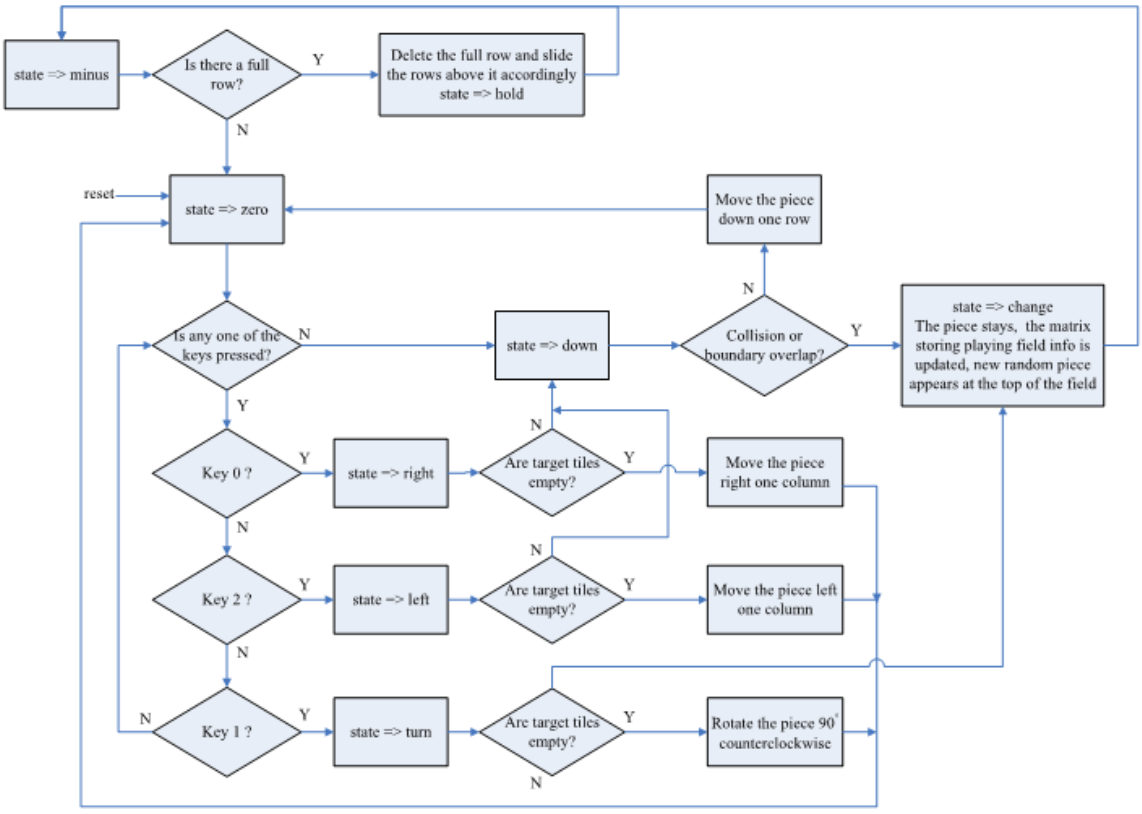

Figure 8. Tetris Code Flowchart

Table 1. State Transitions

Condition	Current State						
	minu	**zero**	**down**	**left**	**righ**	**turn**	**change**
Reset	zero	zero	zero	zero	zero	zero	zero
No keys pressed	—	down	—	—	—	—	—
Key 0	—	right	—	—	—	—	—
Key 1	—	turn	—	—	—	—	—
Key 2	—	left	—	—	—	—	—
Playing field boundaries	—	—	change	down	dow	change	—
Collision	—	—	change	down	dow	change	—
Move available	—	—	zero	zero	zero	zero	minus
Completed row	hold	—	—	—	—	—	—

4. CONCLUSION

In this study, we developed an original Tetris game code that runs entirely on an FPGA chip in VHDL. The code can run on its own without requiring any computer interruption after it is programmed for the first time. This feature provides mobility. The most important aspect of the code is that the code does not use any SRAM modules for storage unlike other implementations in the literature such as [11]. Therefore it is not necessary to write data to a SRAM unit then read from the same address which obviously improves the speed of the code. The utilization of the tile- mapped scheme further improves the speed of the code and simplifies the design procedure. The implementation is done on the Altera DE0 board but since the code is written using standard VHDL functions, it can run on any other FPGA boards. Which as a results, improves the versatility of

the code. Improvements can be added such as the score evaluation and the inclusion of several game levels according to the number of full rows achieved.

REFERENCES

[1] B. Hutchings, and J. Villasenor, "The flexibility of configurable computing," IEEE Signal Processing Magazine, vol. 15, no. 5, pp. 67-84, 1998.
[2] A. Benedetti, and P. Perona, "Real-time 2-D feature detection on a reconfigurable computer," In Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition'98, 1998, pp. 586-593.
[3] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," IEEE Transactions on VLSI Systems, vol. 7, no. 3, pp. 299-308, 1999.
[4] J. Diaz, E. Ros, F. Pelayo, E. M. Ortigosa, and S. Mota, "FPGA-based real-time optical-flow system," IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, no. 2, pp. 274-279, 2006.
[5] A. Downton, and D. Crookes, "Parallel architectures for image processing," Electronics and Communication Engineering Journal, vol. 10, no.3, pp. 139-151, 1998.
[6] D. Crookes, K. Benkrid, A. Bouridane, K. Alotaibi, and A. Benkrid, "Design and implementation of a high level programming environment for FPGA-based image processing," In Proc. IEE Vision, Image and Signal Processing'00, 2000, pp. 377-384.
[7] I. S. Uzun, A. Amira, and A. Bouridane, "FPGA implementations of fast Fourier transforms for real-time signal and image processing," In Proc. IEE Vision, Image and Signal Processing'05, 2005, pp. 283-296.
[8] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. K. Park, M. Kim, and J. W. Jeon, "FPGA Design and Implementation of a Real-Time Stereo Vision System," IEEE Transactions on Circuits and Systems for Video Technology, vol. 20, no. 1, pp. 15-26, 2010.
[9] J. W. Young, J. C. Moyers, and M. Lenox, "FPGA based front-end electronics for a high resolution PET scanner," IEEE Transactions on Nuclear Science, vol. 47, no. 4, pp. 1676-1680, 2000.
[10] I. Andorko, P. M. Corcoran, and P. Bigioi, "FPGA based stereo imaging system with applications in computer gaming," In Proc. International IEEE Consumer Electronics Society's Games Innovations Conference'09, 2009, pp. 239-245.
[11] K. Liu, Y. Yang, and Y. Zhu, "Tetris game design based on the FPGA," In Proc. 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)'12, 2012, pp. 2925-2928.
[12] T. Watanabe, R. Moriwaki, Y. Yamaji, Y Kamikubo. Y. Torigai, Y. Nihira, T. Yoza, Y. Ueno, Y. Aoyama, and M. Watanabe, "An FPGA Connect6 Solver with a two-stage pipelined evaluation," In Proc. International Conference on Field-Programmable Technology (FPT)'11, 2011.
[13] K. Vipin, and S. A. Fahmy, "A threat-based Connect6 implementation on FPGA," In Proc. International Conference on Field-Programmable Technology (FPT)'11, 2011.
[14] T. Fujimori, and M. Watanabe, "Full FPGA game machine," In Proc. IEEE International Conference on Consumer Electronics (ICCE)'16, 2016, pp. 431-432.
[15] T. Yoza, R. Moriwaki, Y. Torigai, Y. Kamikubo, T. Kubota, T. Watanabe, T. Fujimori, H. Ito, M. Seo, K. Akagi, Y. Yamaji, and M. Watanabe, "FPGA Blokus Duo Solver using a massively parallel architecture," In Proc. International Conference on Field-Programmable Technology (FPT)'13, 2013, pp. 494-497.
[16] A. Jahanshahi, M. K. Taram, and N. Eskandari, "Blokus Duo game on FPGA," In Proc. The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS)'13, 2013, pp. 149-152.
[17] T. Fujimori, T. Akabe, Y. Ito, K. Akagi, S.Furukawa, H. Shinba, A. Tanibata, and M. Watanabe "FPGA Trax Solver based on a neural network design," In Proc. International Conference on Field Programmable Technology (FPT)'15, 2015, pp. 260-263.
[18] R. Szabó, and A. Gontean, "Pong game on FPGA with CRT or LCD display and push button controls," In Proc. Federated Conference on Computer Science and Information Systems (FedCSIS)'14, 2014, pp. 729-734.
[19] G. Zhang, and M. Xie, "Design of visual based-FPGA Ping-Pang game with multi-models," In Proc. Second Pacific-Asia Conference on Circuits, Communications and System (PACCS)'10, 2010, pp. 31-34.
[20] Y. Azar, and L. Epstein, "On two dimensional packing," In Proc. Algorithm Theory – SWAT'96, 1996, pp. 321-332.
[21] R. Breukelaar, E. D. Demaine, S. Hohenberger, H. J. Hoogeboom, W. A. Kosters, and D. Liben-Nowell, "Tetris is hard, even to approximate," Journal of Computational Geometry and Applications, vol. 14, pp. 41–68, 2004.
[22] P. G. Casazza, A. Heinecke, K. Kornelson, Y. Wang, and Z. Zhou, "Necessary and sufficient conditions to perform spectral Tetris," Journal of Linear Algebra Applications, vol. 438, pp. 2239-2255, 2013.
[23] L. Langenhoven, W. S. Heerden, and A. P. Engelbrecht, "Swarm Tetris: applying particle swarm optimization to Tetris," In Proc. IEEE Congress on Evolutionary Computation'10, 2010, pp. 1-8.

[24] E. A. Holmes, E. L. James, T. Coode-Bate, C. Deeprose, "Can playing the computer game Tetris reduce the build-up of flashbacks for trauma? A proposal from cognitive science," Journal Plos One, vol. 4, pp. 4153, 2009.
[25] S. Kent, Ultimate History of Video Games, New York: Three Rivers Press, 2001.
[26] Terasic Technologies, Altera DE0 Development and Education Board User Manual, 2009.
[27] P. P. Chu, FPGA Prototyping by VHDL examples, New-Jersey: John Wiley & Sons, 2008.
[28] D. L. Perry, VHDL Programming by Example, New York: McGraw-Hill, 2002.
[29] D. B. Thomas, and W. Luk, "FPGA-optimised uniform random number generators using LUTs and shift registers," In Proc. IEEE International Conference on Field Programmable Logic and Applications'10, 2010, pp. 77-82.