18WSC054-Electronic System Design With FPGAs.

Kenneth Darling B522608

***Abstract* – This paper presents the implementation and modification of the chess clock designed and simulated in Task1. This includes verification of functionality in simulation (using Questasim) and on the target hardware.**

I. Introduction

A Chess clock is a timer used in games of tournament chess where each player only has a set amount of time to complete all their moves. A Chess clock consists of two adjacent timers and two start/stop buttons. Stopping one timer will start the other, so one timer is always running but the two will never run simultaneously. A player loses the game if their clock hits zero before the game is finished.

II. Problem definition

The task for this coursework was to take the chess clock designed in task 1 and implement it onto hardware. In addition to the specification given in task 1 a "wrong mate" input has been added. This is used to indicate the player has incorrectly called check mate. This causes the incorrect player to lose the game. The requirements set out in task 1 have been modified to meet the updated specification.

CCZ-001 The value for the chess clock counters will be set using the *VALUE* signal and loaded when the load switch is on.

CCZ-002 A value will only be loaded if the chess clock is in the reset state.

CCZ-003 Both timers will be set with the same initial value.

CCZ-004 Only one timer will decrement at a time, the two timers will not run simultaneously.

CCZ-005 Once a timer has hit zero a win signal is set high and both the timers will stop.

CCZ-006 Pressing both players Stop buttons simultaneously will pause both timers.

CCZ-007 Pressing the wrong mate button will cause the current player to win the game.

III. Chess Clock Modifications

An additional clock input has been added to the chess clock module to control decrementing the selected player's counter. The 1Hz signal produced by the clock divider is used to decrement the current players timer once per second. The 1Hz signal from the clock divider was initially used as the clock signal for the chess clock module. As updating the state of the finite state machine within the chess clock is a clocked process, this caused the state to only update once per second. During initial testing input from a player could be missed if the button was not held for at least a second. To prevent this the second clock input was added enabling the logic of the chess clock to still use the 100MHz clock of the Zedboard. [1] *CURRENT_STATE* outputs the current state of the chess clock and this is shown on the LEDs of the Zedboard. A *WRONG_MATE* input has been added to indicate a player has falsely called check mate. As a player calls check mate after ending their turn pressing this button causes the current player to win the game. An updated state diagram is shown in Fig. 1.

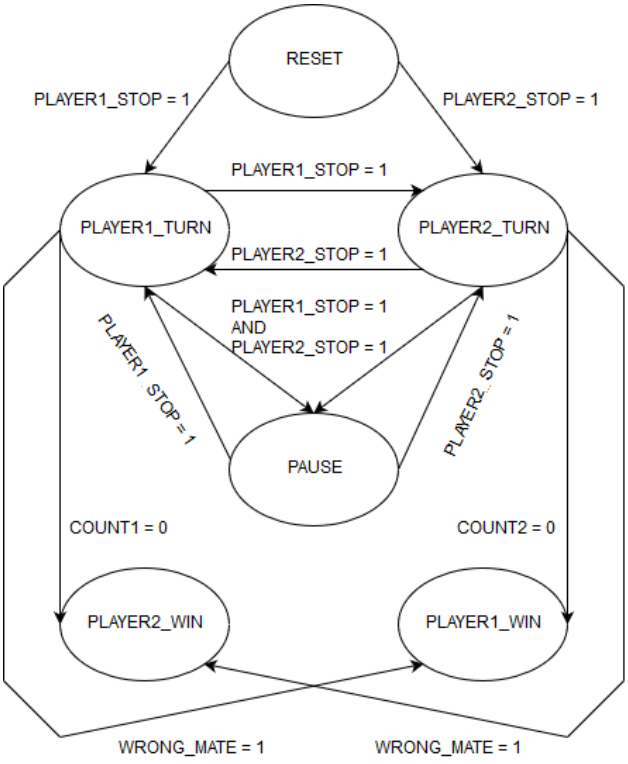

Fig. 1. FSM diagram for the chess clock.

IV. Clock Divider

A Clock Divider, or frequency divider, is a circuit that divides an input clock to produce an output clock with a lower frequency. [2] A Clock divider consists of a counter which increments on the rising or falling edge of the input clock. Once the counter has reached its required value the output is inverted. This produces an output clock with a lower frequency than the input clock. The value the counter counts too can be adjusted to produce clocks of different frequencies. A generic is used so this value can be set when a clock divider entity is instantiated. To produce the 1Hz clock needed for decrementing the players timers a value of 50,000,000 was used as the base clock of the Zedboard is 100Mhz. [1] The clock dividers counter is reset when switching players. This was done so the remaining time taken during a player's turn less than a second does not affect the next player. Without resetting the counter, if player 1's turn took 5.5 seconds player 2's counter would decrement by 1 after only 0.5 seconds.

V. Zedboard chess clock module

The Zedboard chess clock module is used to map the hardware inputs and outputs of the Zedboard development board to the chess clock module. The *DEBOUNCE_PROCESS* is used to manage the inputs from the on-board buttons. Switch bounce is caused when the contacts of a switch rapidly connect and disconnect when a switch is initially pressed. This can cause multiple erroneous button pressed to be generated. Debounce aims to negate these by adding a delay to when the input is read so the switch contacts are stable. [3] The process checks that a button has been held for 0.1 seconds and then sets the corresponding control signal high for a clock cycle. The *HOLD* signal is then set high until the button is released which prevents another signal being sent until the button has been released and reapplied. This is used to prevent the chess clock skipping the first turn of the game or when resuming from the *PAUSE STATE* as the same button is used to start and stop the first turn. *LED_PROCESS* sets the LEDs used to show which player has won the game and the current state of the chess clock. Switch SW1 swaps the LED mode to show the current players remaining time. This is represented as two binary coded decimals. Due to limitations of the Zedboard only having 8 LEDs only the last two digits are shown. When the chess clock is in *RESET_STATE* the value of *VALUE* is shown on the LEDs and when in *PAUSE_STATE* the value shown alternates between the two players timers. Switches *SW7, SW6, SW5* are used to set *VALUE. SW0* loads the starting value to each players timer.

TABLE 1
LED USAGE IN STATE MODE

LD7	Reset
LD6	Pause
LD5	Player1 Turn
LD4	Player2 Turn
LD3, LD2	Not used
LD1	Player1 Win
LD0	Player2 Win

TABLE 2
USER INPUT

BTNL	Player 1 Stop
BTNR	Player 2 Stop
BTNU	Wrong Mate
BTND	Reset
SW7	Set time 10s
SW6	Set time 60s
SW5	Set time 120s
SW1	Switch between current state and current players remaining time
SW0	Load selected time

VI. Simulation and RTL Validation

The test bench produced from task 1 has been modified to include the wrong mate scenario, pause functionality was already included in the original test bench. Only the chess clock module has been tested in simulation as both the clock divider and button debounce used within the Zedboard chess clock module would take too long to simulate. A TCL script is used to compile the required components and then run the testbench. TCL is a scripting language used in conjunction with Questasim to run simulations. [4] This also enables simulations to be run from the terminal. As per the specification a signal force from the TCL script is used to set the wrong mate button high and to set both players buttons high to pause the state machine. Checks for both cases have been added to the test bench and these checks will fail if the TCL commands are not executed. There are three test cases within the test bench. Testcase 1 is unchanged from the first task and simulates a game running until Player 2 runs out of time. Pause is also tested in this testcase. Testcase 2 test the wrong mate functionality. Additionally, the wrong mate signal is forced using the TCL script. Testcase 3 tests the pause functionality with both players buttons set high using the TCL script. A test within the test bench checks that the current state of the chess clock is in the pause state.

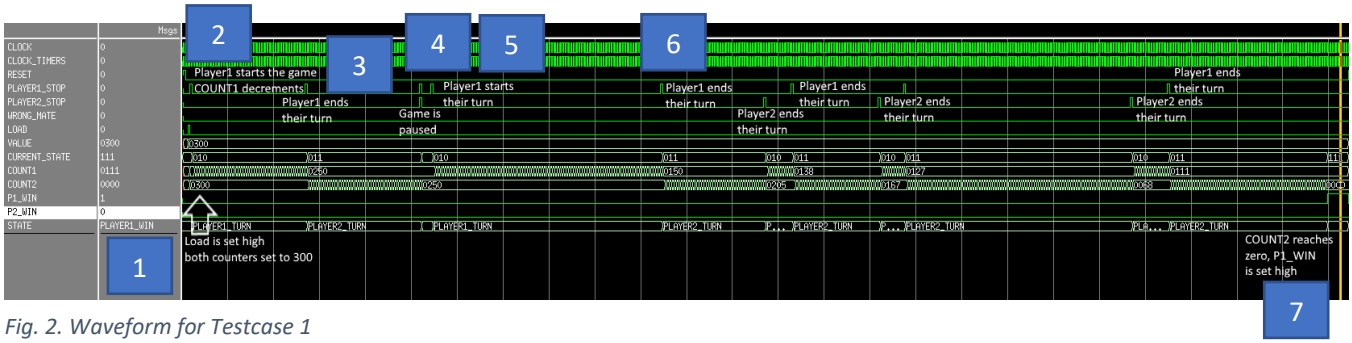

Fig. 2. Waveform for Testcase 1

An annotated waveform for testcase 1 is shown in Fig. 2. The chess clock is reset and the value for both players counter is set to 300 shown at point 1. *PLAYER1_STOP* is set high starting player 1's turn shown at point 2. This turn lasts 50 clock cycles. At the end of this turn *PLAYER1_STOP* is set high again to transition to player 2's turn shown at point 3. At point 4 both *PLAYER1_STOP* and *PLAYER2_STOP* are set high moving the chess clock to the *PAUSE* state. Both timers stop for 5 clock cycles. *PLAYER1_STOP* is then set high to start player 1's turn after the pause shown at point 5. This turn ends at point 6. The players continue to alternate turns until point 7 when player 2's counter reaches zero. The chess clock then transitions to *PLAYER1_WIN* state and *P1_WIN* is set high by the chess clock. Player 1's counter stops.

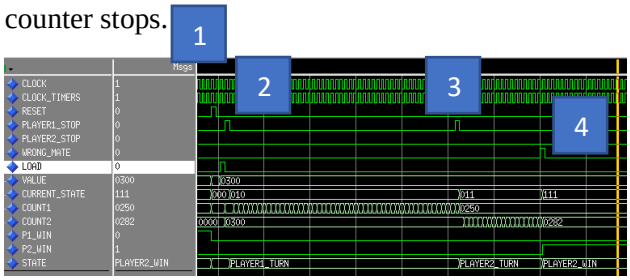

Fig. 3. Waveform for Testcase 2 (Wrong Mate)

Fig. 3. shows the waveform for testcase 2. The game starts the same as in testcase 1 with *LOAD* set high and *VALUE* is set to 300. *PLAYER1_STOP* is set high starting player 1's turn shown at point 2. This turn lasts 50 clock cycles. At the end of this turn *PLAYER1_STOP* is set high again to transition to player 2's turn shown at point 3. During player 2's turn *WRONG_MATE* is set high. This causes the chess clock to move to the *PLAYER2_WIN* state shown at point 4.

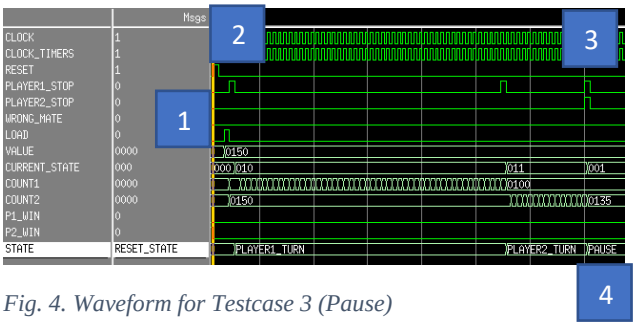

Fig. 4. Waveform for Testcase 3 (Pause)

Fig. 4. shows the waveform for Testcase 3. *VALUE* is set to 150 shown at point 1. *PLAYER1_STOP* is set high starting player 1's turn shown at point 2. This turn lasts 50 clock cycles. Player2's turn then lasts for 15 clock cycles. At this time the TCL force of both plays stop buttons is triggered and the game is paused, this is shown at point 3. The test bench then tests if the chess clock is in the *PAUSE* state. This test is done using the *CURRENT_STATE* signal. In the pause state this is set to "001". Point 4 shows that the chess clock is in the pause state.

```
elkgd@el-linux:~/projects/chess_clock
# Loading ieee.numeric_std(body)
# Loading work.tb_chess(testbench)#1
# Loading work.chess_clock(rtl)#1
# Loading work.four_digit_counter(rtl)#1
# Compilation is over
# ** Note: 1st turn test passed
#    Time: 1085 ns  Iteration: 0  Instance: /tb_chess
# ** Note: 2nd turn test passed
#    Time: 2535 ns  Iteration: 0  Instance: /tb_chess
# ** Note: 3rd turn test passed
#    Time: 3035 ns  Iteration: 0  Instance: /tb_chess
# ** Note: 4th turn test passed
#    Time: 4135 ns  Iteration: 0  Instance: /tb_chess
# ** Note: 5th turn test passed
#    Time: 4985 ns  Iteration: 0  Instance: /tb_chess
# ** Note: Player 1 win test passed
#    Time: 5085 ns  Iteration: 0  Instance: /tb_chess
# ** Note: Player 2 win, Wrong mate test passed
#    Time: 9215 ns  Iteration: 0  Instance: /tb_chess
# ** Note: TCL pause passed
#    Time: 10245 ns  Iteration: 0  Instance: /tb_chess
# ** Error: All tests passed
#    Time: 10245 ns  Iteration: 0  Instance: /tb_chess
VSIM 2>
```

Fig. 5. Test output in console

Fig. 5. shows the output of the testbench in the console. This is achieved using the *-c* flag when launching Questasim. This enables a developer to quickly test changes in the code without waiting for Questasim to launch its GUI (Graphical user interface). This is especially useful when accessing the Linux server from a remote location.

VII. Requirements traceability

Fig. 2. shows that the initial value for both players counters set equal to *VALUE* when *LOAD* is high. This satisfies requirement CCZ-001 and CCZ-003.

Requirement CCZ-002 has been tested with the *LOAD_TEST* procedure included within the testbench. The waveform for this is shown in Fig. 6. *LOAD* is set high at point 1 but the value of *COUNT1* and *COUNT2* do not change.

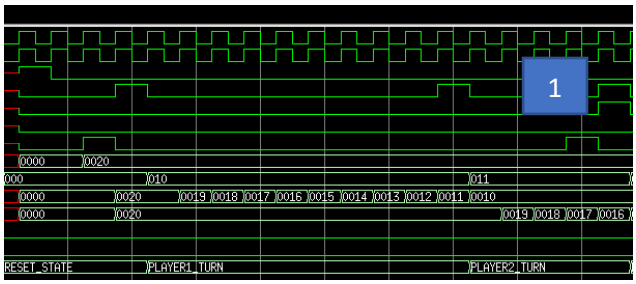

Fig. 6. Waveform for load test

The waveform shown in Fig. 2. shows that only one counter decrements at a time, meeting requirement CCZ-004.

Once Player 2's counter reaches zero player 1's win output is set high shown at point 7 in Fig. 2. meeting requirement CCZ-005. This output is mapped to an LED in the Zedboard module.

Point 4 in Fig. 2. shows that when both players buttons are pressed simultaneously the chess clock moves to the *PAUSE* state and both players timers are paused. This meets requirement CCZ-006.

In Fig. 3. the wrong mate input is set high during player2's turn. The chess clock then moves into the *PLAYER2_WIN* state meeting requirement CCZ-007.

VIII. Conclusion

The Zedboard can function as a basic chess clock showing when a player has exhausted their time. Due to the limitations of the Zedboard hardware there is not an appropriate way to show the remaining time of each players timer. There is an LCD screen on the development board which could be used but this would require more time that this assignment allowed. The Zedboard also features Pmod connectors which could be used to connect a seven segment display to show each players available time. Some rules of chess allow for a player to gain time one their turn is completed. Using the Bronstein method of timing a player gains up to 5 seconds back on completing their turn (but not more time that they started that turn with). [5] Another method is Fisher timing where each turn a player gains 30 seconds and if they complete their turn in less that 30 seconds they can gain time. [5] These timing modes could be added to the chess clock with the onboard switches used to set the timing mode.

References

[1] AVNET, "ZedBoard," 1 January 2014. [Online]. Available: http://zedboard.org/product/zedboard. [Accessed 20 April 2019].

[2] S. Akthar, "VHDL Code for Clock Divider (Frequency Divider)," ALL ABOUT FPGA, 29 June 2014. [Online]. Available: https://allaboutfpga.com/vhdl-code-for-clock-divider/. [Accessed 20 April 2019].

[3] J. Christoffersen, "Switch Bounce and How to Deal with It," All about circuits, 03 September 2015. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/. [Accessed 14 May 2019].

[4] https://www.doulos.com/knowhow/tcltk/tutorial/, "What is Tcl?," Doulos, [Online]. Available: https://www.doulos.com/knowhow/tcltk/tutorial/. [Accessed 14 May 2019].

[5] C. Bird, "Chess Clocks… Understanding Time," Chess Tournament SErvices, 14 December 2011. [Online]. Available: https://chesstournamentservices.com/2011/12/chess-clocks-understanding-time/. [Accessed 14 May 2019].