ESDC - Block Diagram
Mastermind 2-Player Game

Contents

1. Introduction

Specs Update

In this laboratory assignment we are not delivering any updated project SPECS, as no changes have been asked by the teacher and also at this point we have not performed any changes in the specifications of our design.

Document Description

The following sections will provide the reader with a detailed description of our project Block Diagram. The document is divided in two main sections. Section 2 gives an overall and detailed explanation of the interaction between all blocks in our digital system. Section 3 describes the behaviour of each individual block by giving an overall description of its functioning and also by focusing on the logic behind each of its pins (I/O).

2. Block Diagram

2.1. Global System State Diagram

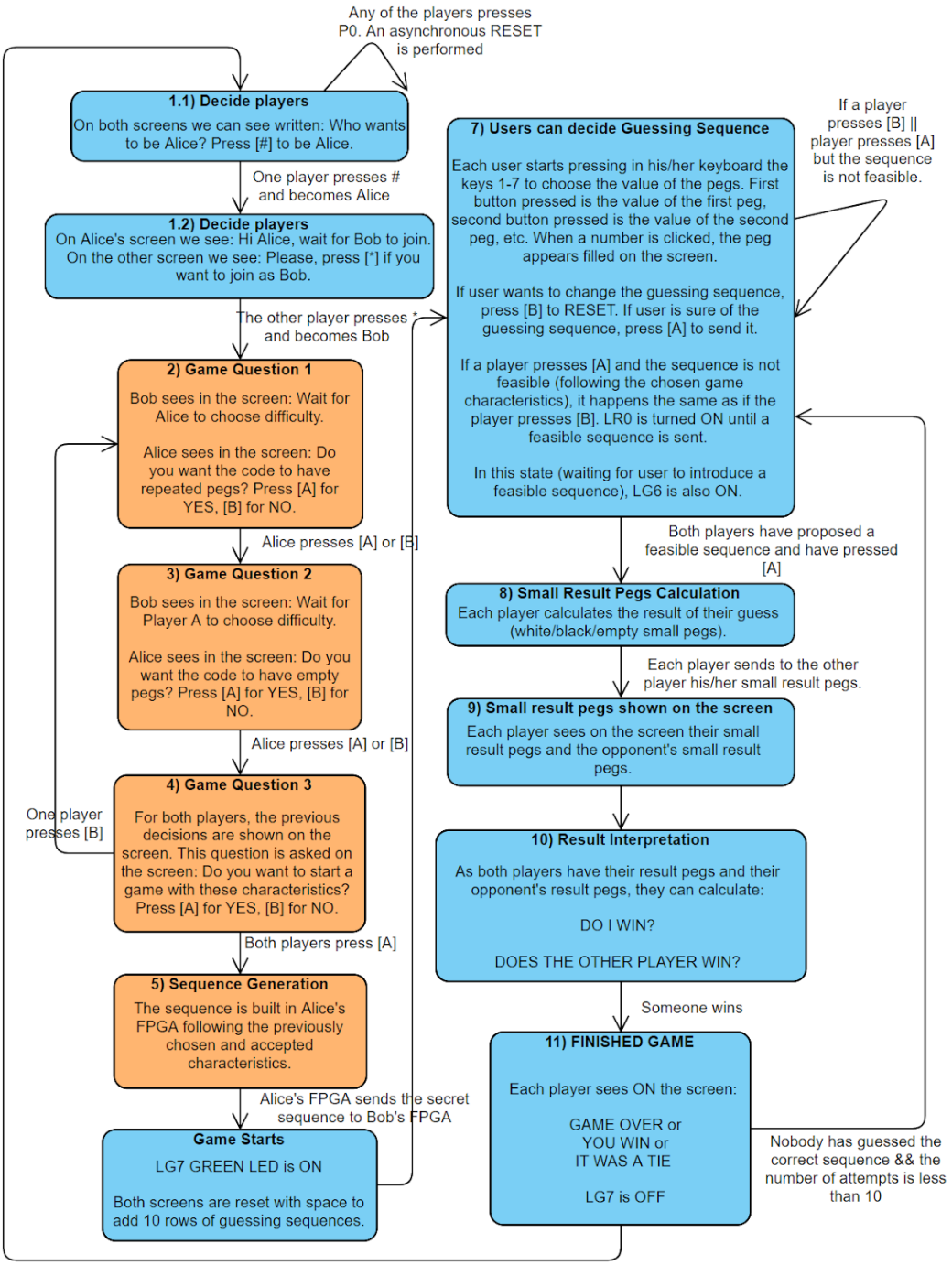

Figure 2.1.1. (Initial) General System Flux Diagram (simplified and not updated with detail).

In **Figure 2.1.1**, we can see a possible Flux Diagram to describe our whole digital system. We have performed it in order to have a good reference to understand the functioning of our system. This diagram has been extracted from the project SPECS, and even though there are little discrepancies, it helps to understand the approach of the game.

2.2. Global System Description

In **Figure 2.2.1**, we can see the full system Block Diagram. However, as it is not possible to properly see all the signals in this image, we have done a more general Block Diagram by hand (see **Figure 2.2.2**) that makes it easier to understand how the whole system is connected. Also, when fully explaining the system, we will add some screenshots of particular parts of the Block Diagram from the *.bdf* file.

Before starting, we would like to highlight that the only moment when the system differs between Alice and Bob is when they are in the Identifier Blocks (and also in some procedures that take place in the Main Block). Basically, stages 2.a) and 2.b) are the main parts when the system flux differs from both characters.

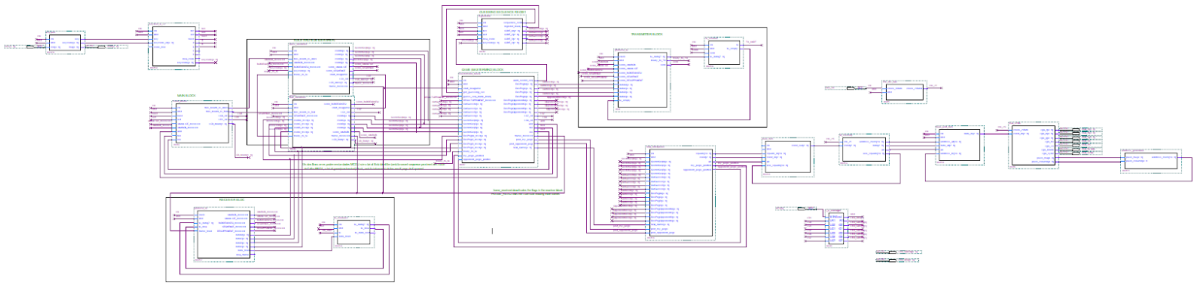

Figure 2.2.1. Full Block Diagram in Quartus Software (*block_diagram.bdf* file)

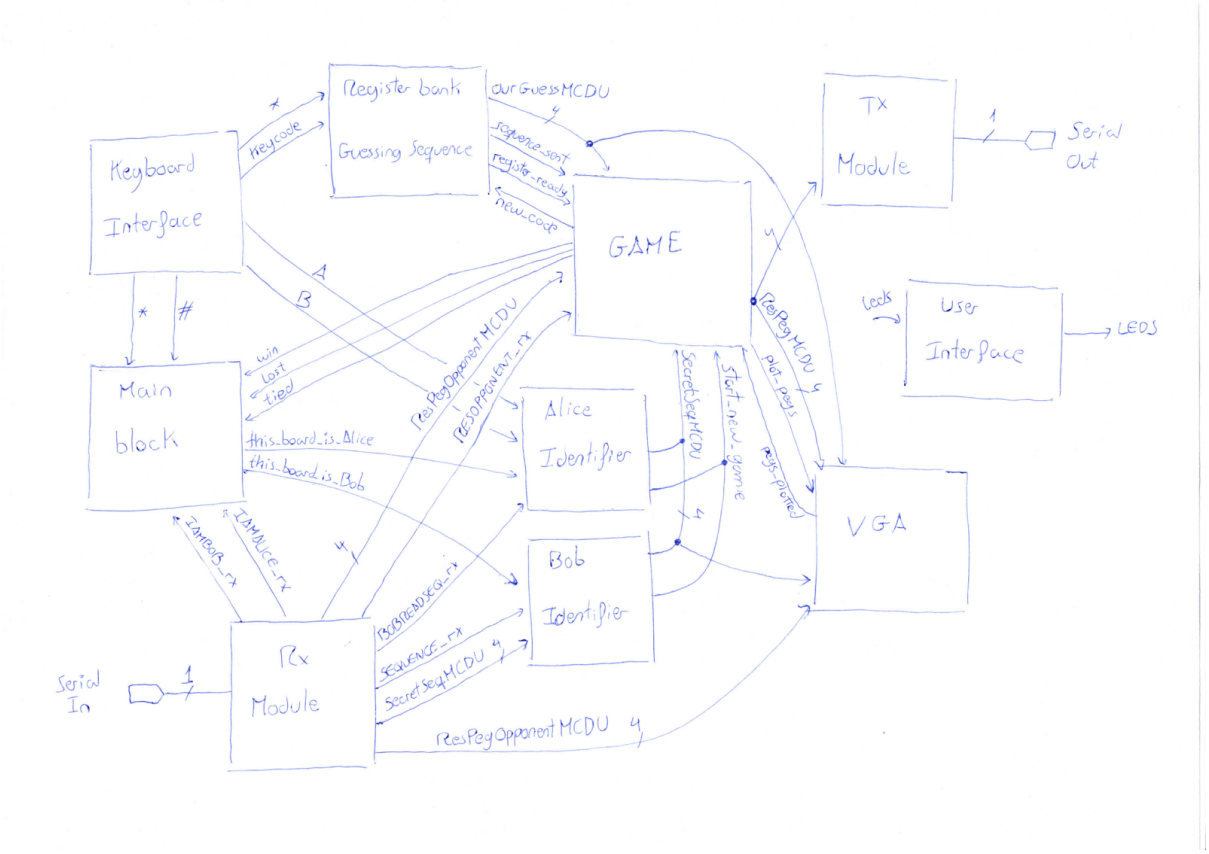

Figure 2.2.2. Summarized Block Diagram (done by hand)

Stage 1: System Init

The main blocks that take part in the System Init stage are the keypad (Keyboard UI) and the Main Block. Also, the VGA interpreter is used because there are some events that will have to be shown on the user screen.

A) **Just to clarify → If a game has already happened before (AFTER STAGE 4 FINISHES):**

If a game has already happened (the user has played Mastermind turns until the game has finished), independently of the game result (won/tied/lost), the user will have to press the # key in order to start again. When this key is pressed, the whole game will be reseted and everything will start again (LEDs will be turned off and the VGA screen will go background color entirely). It is important to highlight that LG6 and LG7 LEDs are turned off in the Game Block when the game has ended (there is no need to ask the Main Block to do it as the Game Block is able to turn off these LEDs).

For example, LG1 and LG2 LEDS are deactivated here, because their function is to specify whether the user is Alice or Bob, but as we are starting a new game the user has still not a character associated to their FPGA.

B) **If the system has just been initiated and we have not played yet:**

When the system is initiated, we see on the screen the following message: "**PRESS # TO BECOME ALICE, PRESS * TO BECOME BOB**". The VGA Interpreter is notified of this need by the value of the 5 bits bus VGA_Game. The Main Block is waiting for the user to press either # or * in order to identify as a character. While the user has not been identified as one of the two characters, the LED LG0 will remain on, but at the moment the user has been identified as Alice or Bob, this LED will turn off. So far, this LED is turned on.

B.1) User wants to identify as Alice:

The user presses the # key because they want to be identified as Alice. We have to check if this is possible.

- **B.1.1) The opponent has already been identified as Alice:** We will know if this is the case because the input IAMALICE is high. If this is the case, first of all, it is very important to remember that the Main Block does not have to deactivate the IAMALICE_received flag from the receiver block by activating the frame_received output because we need this flag activated in the Bob Identifier block (this user will have to be identified as Bob). We have to notify the user that the decision of being Alice is incorrect by turning on the LED LR1. This user, in this game, can only be identified as Bob. ***Start Stage 1 AGAIN***
- **B.1.2) The opponent has not been identified as Alice:** We will now if this is the case because the input IAMALICE is low. If this is the case, the output this_board_is_Alice will turn high, the LED LG0 will turn off (the character identification has finished) and the Main Block will stop being the working block for some time, until the game finishes. ***Move to 2.a): Character Identification as Alice***

B.2) User wants to identify as Bob:

The user presses the * key because they want to be identified as Bob. We have to check if this is possible.

- **B.2.1) The opponent has already been identified as Bob:** We will know if this is the case because the input IAMBOB is high. If this is the case, first of all, it is very important to remember that the Main Block does not have to deactivate the IAMBOB_received flag from the receiver block by activating the frame_received output because we need this flag activated in the Alice Identifier block (this user will have to be identified as Alice). Also, we have to notify the user that the decision of being Bob is incorrect by turning on the LED LR1. This user, in this game, can only be identified as Alice. ***Start Stage 1 AGAIN***
- **B.2.2) The opponent has not been identified as BOB:** We will know if this is the case because the input IAMBOB is low. If this is the case, the output this_board_is_Bob will turn high, the LED LG0 will turn off (the character identification has finished) and the Main Block will stop being the working block for some time, until the game finishes. ***Move to 2.b): Character Identification as Bob***

It is important to highlight that the frame_received output in the Main Block is not necessary because we do not want to deactivate the IAMALICE_received and IAMBOB_received flags because they will be required in the Identifier blocks. However, we have kept the output in order to follow the standard of having a frame_received output in all the blocks that contain flags from the receiver block.

Stage 2.a): Character Identification as Alice

If the system flux is at this point, this means that the user has successfully been identified as Alice. The Alice Identifier Block is activated by its input signal this_board_is_Alice. The first thing that this block does is activate the output send_IAMALICE, in order to notify the opponent that this FPGA has been identified as Alice. Also, the LG1 LED is activated, indicating that this user is Alice. At this point, Alice will have to answer two questions related to the game difficulty. We are still seeing on screen the message "**PRESS # TO BECOME ALICE, PRESS * TO BECOME BOB**". Now the VGA_Game 5 bits bus has to communicate to the VGA Interpreter that Alice has to see a question on the screen (question 1): "**DO YOU WANT THE SECRET SEQUENCE TO HAVE REPEATED PEGS?**". To accept the proposal, the user has to press the A key on the keyboard and to decline it, the user has to press the B key on the keyboard (these are inputs of the Alice Identifier). After the user has answered this question, the VGA_Game bus will notify the VGA Interpreter that Alice has to see question 2 on the screen: "**DO YOU WANT THE SECRET SEQUENCE TO HAVE EMPTY PEGS?**". To accept the proposal, the user has to press the A key on the keyboard and to decline it, the user has to press the B key on the keyboard. When the user has answered both questions, the Alice Identifier has enough information to generate the secret sequence. When the secret sequence is generated, it is put in the MCDU buses of 4 bits with the name of SecretSeq. Now, the identifier will wait for the input IAMBOB_received to be high, which will mean that the opponent user has been correctly identified (this is part of a kind of control of errors). Then, the Alice Identifier will activate the output frame_received in order to deactivate the flag IAMBOB_received in protocol rx. After this, the Alice Identifier will activate the send_SEQUENCE output to ask the transmitter block to send the secret sequence to the

opponent (who needs it, because they are Bob and they need the secret sequence to play). To do this, it is necessary to wait for the ready_to_tx input to be high, meaning that the transmitter bus is empty. Now, the Alice Identifier waits to get the BOBREADSEQ_received flag activated, meaning that Bob has received the sequence properly. When this happens, the frame_received output is activated again, in order to make sure that protocol rx can deactivate this flag. Finally, to finish this stage, the start_newgame output is activated in order to start a new game (this pin is connected as an input of the Game Block). ***Move to 3: Mastermind Game***

As we have previously specified in the project SPECS document, both players have the secret sequence in their FPGA but they cannot access it because it is, as its name indicated, secret. For both cases (Alice and Bob), the secret sequence is stored in their identifier.

Stage 2.b): Character Identification as Bob

If the system flux is at this point, this means that the user has successfully been identified as Bob. The Bob Identifier Block is activated by its input signal this_board_is_Bob. The first thing that this block does is activate the output send_IAMBOB, in order to notify the opponent that this FPGA has been identified as Bob. Also, the LG2 LED is activated, indicating that this user is Bob. At this point, the VGA_Game bus of 5 bits will notify the VGA Interpreter that Bob has to see on the screen: "**PLEASE WAIT BOB**". However, it is also an option that at this point the VGA screen is just in background color. The Bob Identifier block will wait until the input SEQUENCE_received is activated by the protocol rx block. At this point, the block will activate the frame_received output in order to turn the received flag low at the receiver block. Also, the MCDU outputs of the Bob Identifier will get the values of the MCDU outputs that we have at this moment at the protocol rx block, which correspond to the secret sequence values. After this, the output send_BOBREADSEQ will be activated (after checking that the ready_to_tx is high in a waiting state) to notify the transmitter that Bob wants to send the BOBREADSEQ frame to Alice to notify her that he has properly received the sequence. Finally, to finish this stage, the start_newgame output is activated in order to start a new game (this pin is connected as an input of the Game Block). ***Move to 3: Mastermind Game***

Stage 3: Mastermind Game

If the user is at this point, this means that the start_newgame input of the Game (Mastermind) Block has been activated by any of the identifiers. At this point, the user has to notify the VGA Interpreter that a game has started and that therefore the whole screen has to be in background color. This is made by using the output VGA_Game of 5 bits. When the game starts, a series of steps will be repeated in a loop. This loop will end in case someone has guessed the sequence (either Bob or Alice) or in case the number of failed turns is 10 (maximum opportunities given by the game, based on the original classical Mastermind Game). In the following lines we can see the steps in the loop.

(TURN STARTS)

A) **Ask the user for a guessing sequence (VGA interaction)**

The Game Block waits for the Guessing Sequence Register to be ready to use (waiting for the input guess_seq_bank_ready to be high). When the bank is ready, the Game Block output called want_secret_seq is activated and asks the register to store a new guessing sequence (the new_code register input is activated). When this happens, the register stores the value of the keyboard keys pressed by the user in the OurGuess MCDU 4 bits buses. As we have previously explained (this is also explained in the SPECS document and with more detail), each number pressed in the keyboard corresponds to a specific color.

We have to remember that the LED LG6 must be activated while the register is working. This means that when the user is introducing the guessing sequence, this LED will be turned on. Also, the LG7 LED will be turned on during the time between the activation of the start_newgame input and the activation of any of the finish game outputs (won/tied/lost).

The VGA Interpreter is constantly reading and updating on the screen the value of the OurGuess MCDU buses. Therefore, everytime the user presses a key, we see its correspondent color on the screen. When the user is happy with the guessing sequence that they see on the screen, they have to press te * key. At this moment, the sequence_sent output of the register is activated, activating the got_guessing_seq input of the Game Block. Due to this, the Game Block knows that it has access to the guessing sequence entered by the user and that the result pegs can be calculated.

B) **Compute the feedback for the entered guessing sequence (Generate the Result pegs)**

The result pegs of the user are calculated by comparing the value of the secret sequence stored in one of the identifiers (the Alice or the Bob one, depending on whether the user is Alice or Bob) and the value of the guessing sequence currently stored in the Guessing Sequence register. When the pegs are computed, this means that the FPGA has all the necessary information to show on the screen the entered guessing sequence and the result pegs. Therefore, the VGA_Game 5 bits bus and the activation of the plot_my_pegs Game Block output inform the VGA Interpreter that it has to show on the screen the guessing sequence and the result pegs of the user. The VGA Interpreter notifies the Game Block that the guessing sequence and result pegs are being shown on the screen by activating the output my_pegs_plotted, which is connected to the Game Block input with the same name.

It is interesting to highlight the fact that, at this point, the guessing sequence of this turn will be shown on the screen twice because we have in the guessing sequence row and also in the turn row, but it will disappear from the guessing sequence row at the moment we move to the next turn. This is a way to keep track of the previously entered sequences in order to help the user to guess the secret sequence.

C) **Send user result pegs and wait for the result pegs of the opponent**

At this moment, the FPGA has to send to the opponent the result pegs so that the opponent has them. The Game Block makes this possible by activating the send_RESOPPONENT

flag, which tells to the transmitter block that it is necessary to send the result pegs through the transmitter block (this can be done after making sure that the ready_to_tx input is high in a waiting state).

The Game Block enters in a new wait state that waits for the opponent result pegs to arrive at the receiver block. The Game Block is notified when this happens because the flag RESULTOPPONENT_received is activated by the receiver block. When this happens, the Game Block has to deactivate this receiver flag by activating the frame_received output.

D) Show on the screen the opponent result pegs

At this point, the result pegs of the opponent are located in the MCDU 4 bits buses of the receiver. The VGA Interpreter knows that it has to show on the screen the opponent result pegs due to the value of the VGA_Game output of the Game Block and the activation of the plot_opponent_pegs output of the Game Block. The VGA Interpreter gets the value of the result pegs of the opponent and plots it. The VGA Interpreter notifies the Game Block that the opponent result pegs are being shown on the screen by activating the output opponent_pegs_plotted, which is connected to the Game Block input with the same name. When this input is activated in the Game Block, this block has enough information to check whether someone has won and therefore to decide if the game continues or not. If the game finishes, the Game Block will communicate to the Main Block the result of the game with the won/lost/tied outputs, which are inputs of the Main Block. The LED LG7 is turned off because the game has finished.

(TURN FINISHES → Go back to A) in case nobody has guessed the secret sequence OR in case this turn was the 10th one)

Move to 4: Finished Game

Stage 4: Finished Game

If the system is at this point this means that the game has finished, which also means that one of the three finishing signals has been activated (won/lost/tied). By the activation of these signals, we are moving from working in the Game Block to working in the Main Block. The VGA_Game 5 bits bus informs to the VGA Interpreter that the game result has to be shown on the screen. We see a message such as: "**YOU WIN**" or "**GAME OVER**" or "**IT'S A TIE**". Also, on top of the screen we see the value of the correct sequence (the secret sequence), which is still stored in the correct identifier. Now we are at the moment when we want to start a game again, which corresponds to Stage 1.A).

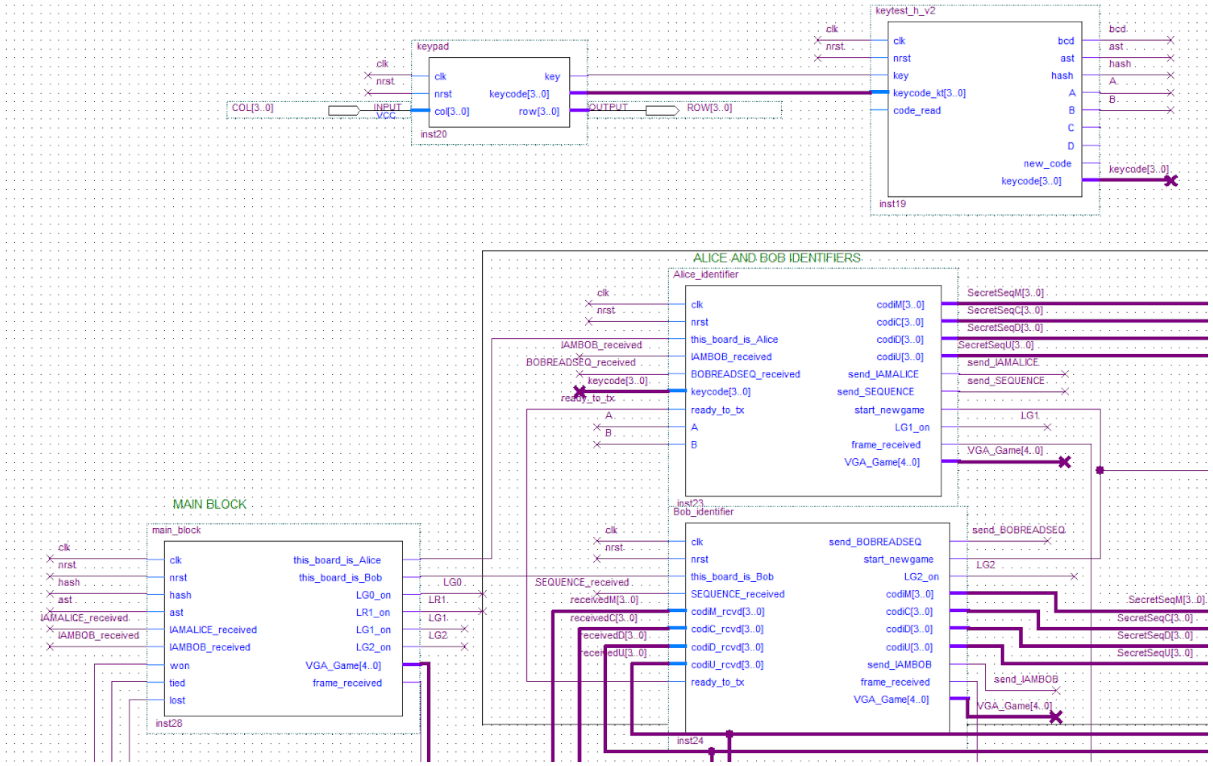

Figure 2.2.3. Main blocks that take part in Stage 1 and Stage 2.

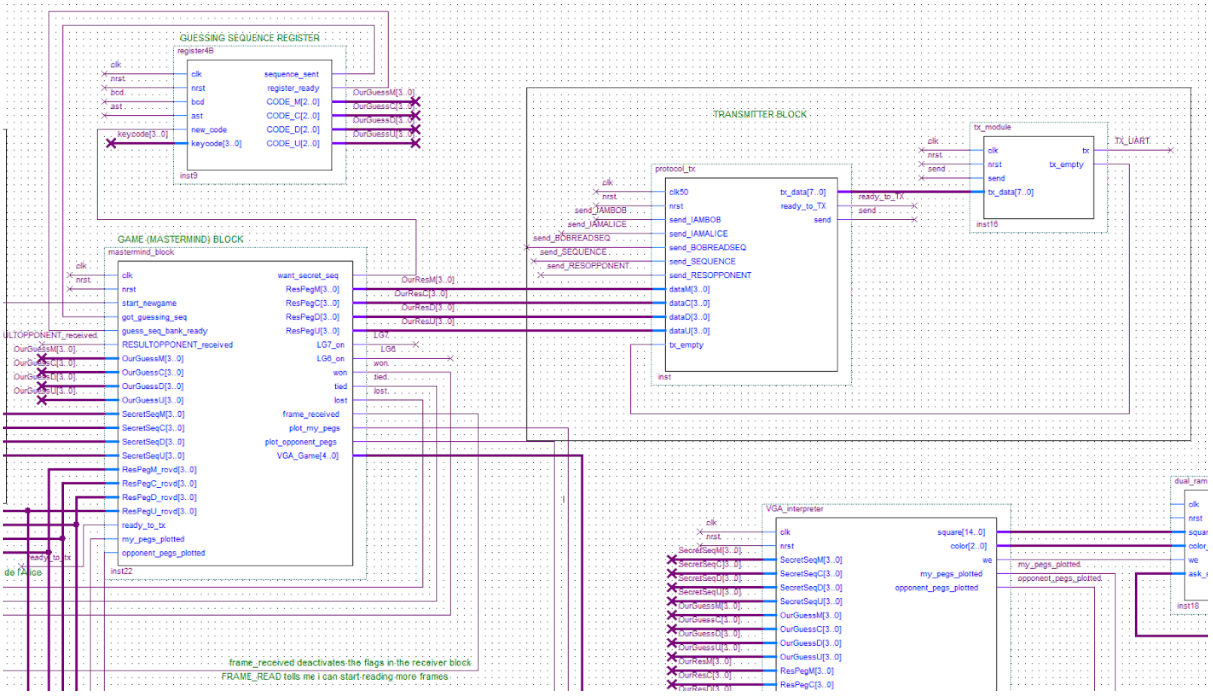

Figure 2.2.4. Main blocks that take part in Stage 3 (such as the guessing sequence register and the game/mastermind block).

3. Description of Individual Blocks

We consider it important to highlight that inputs such as *nrst* and *clk* will not be described in each block due to being reiterated and therefore not necessary information about the system. In **Figure 3.1.**, we see the distribution of the sequence and result pegs. We consider that is useful to include in this document this figure (which comes from the SPECS document) because when explaining some blocks we will refer to this nomenclature (*ABCD* and *1234* pegs).

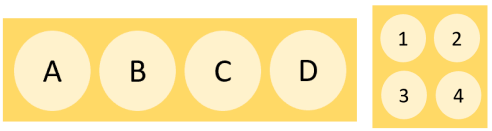

Figure 3.1. Sequence pegs (left) and small result pegs (right). This Figure comes from the project specifications document.

3.1. Keyboard Block

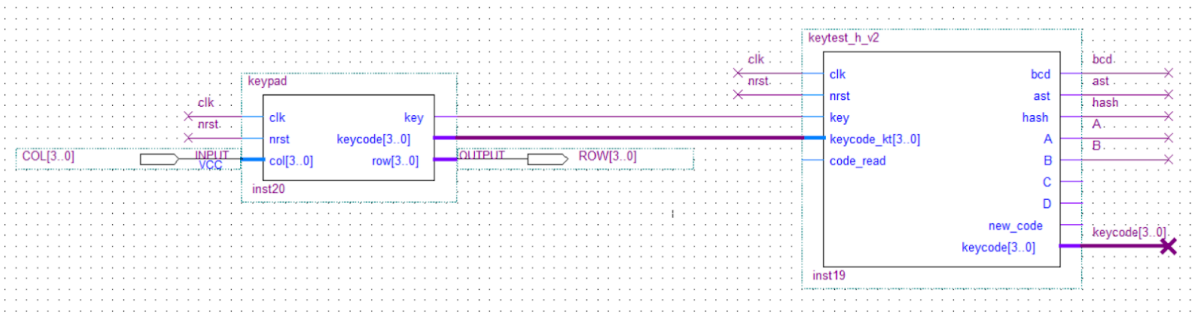

Figure 3.1.1. keypad and keytest blocks (*block_diagram.bdf* file from Quartus Software).

a. keypad & keytest

INPUTS:

- col[3..0] **(4 bits)**: signal to connect to the external keyboard.

OUTPUTS:

- row[3..0] **(4 bits)**: signal to connect to the external keyboard.
- bcd: flag signal that indicates when the user presses a number between 0 and 9 in the keyboard.
- ast: flag signal that indicates when the user presses * in the keyboard.
- hash: flag signal that indicates when the user presses # in the keyboard.
- A, B, C, D: flag signal that indicates when the user presses each letter in the keyboard.
- keycode[3..0] **(4 bits)**: code of the key pressed in hexadecimal. When the user presses any key from 0 to 9, keycode contains the corresponding hexadecimal code. * has de code E and # has de code F.

3.2. Register Bank

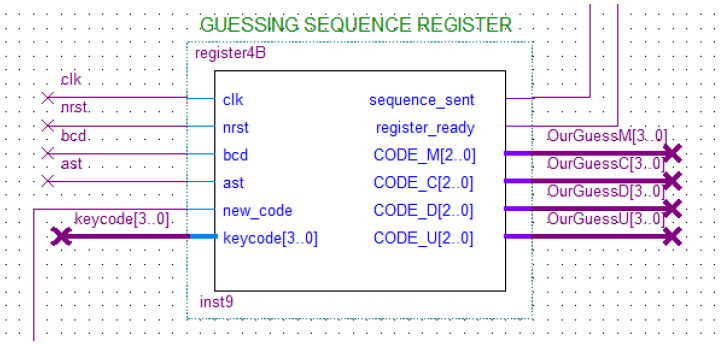

Figure 3.2.1. register bank blocks (*block_diagram.bdf* file from Quartus Software).

This register stores the guessing sequence entered by the user in each turn until the next turn starts or the game ends. In this design, there is only one register because there is only one sequence stored by the user.

INPUTS:

- bcd: flag that is activated when the user maintains a bcd key pressed. Everytime the user presses a bcd key, this flag will be activated and the bcd value (equivalent to one of the colors of the game) will be stored in the MCDU buses. Therefore, the color assigned to the bcd value will appear in the guessing sequence.
- ast: flag that is activated when the user maintains the [*] key pressed. The user must press it to send the guessing sequence that is currently stored in the MCDU buses. This means the user has to press the [*] key when they are sure about the introduced guessing sequence. When this input is activated, the register's output named *sequence_sent* is also activated to notify the Game Block that the sequence has been introduced.
- new_code: this input is activated by the Game Block when the user can introduce the guessing sequence in the current turn. When this input is activated, the register starts 'functioning' and therefore the user is able to introduce the guessing sequence.
- keycode[3..0] **(4 bits bus)**: this 4 bits bus is an output of the Keyboard Block and contains the code of the key that is being pressed by the user. It is useful for our design because it contains the bcd value of the pressed key, and each bcd value corresponds to one color in the sequence.

OUTPUTS:

- sequence_sent: this output is activated when the user has introduced the guessing sequence and pressed [*]. It notifies the Game Block that the turn can continue because the user has already chosen their guessing sequence.
- register_ready: this output is activated when the register bank is ready to store a new sequence. This pin is read by the Game Block, who waits for it to be high and then asks the register for a new guessing sequence.
- CODE_M **(4 bits bus)**: 4 bits bus storing Peg A in the guessing sequence.
- CODE_C **(4 bits bus)**: 4 bits bus storing Peg B in the guessing sequence.
- CODE_D **(4 bits bus)**: 4 bits bus storing Peg C in the guessing sequence.
- CODE_U **(4 bits bus)**: 4 bits bus storing Peg D in the guessing sequence.

3.3. Transmitter Block

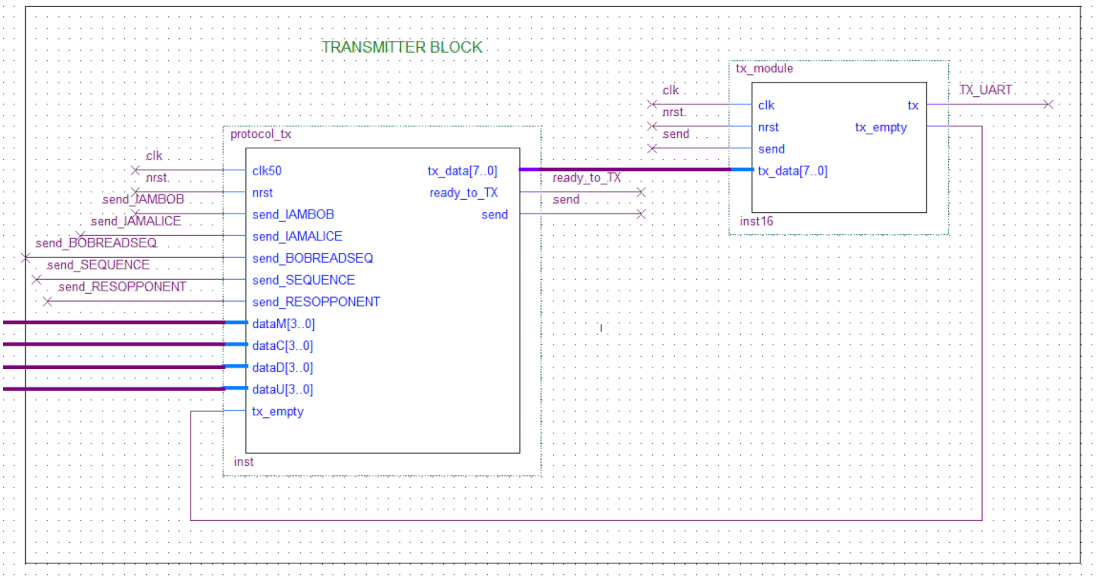

Figure 3.3.1. protocol_tx and tx_module blocks (*block_diagram.bdf* file from Quartus Software).

a. protocol_tx

INPUTS

- send_IAMBOB: Active high. Signal that, when active, it means the frame “IAMBOB” has to be sent.
- send_IAMALICE: Active high. Signal that, when active, it means the frame “IAMALICE” has to be sent.
- send_BOBREADSEQ: Active high. Signal that, when active, it means the frame “BOBREADSEQ” has to be sent.
- send_SEQUENCE: Active high. Signal that, when active, it means the frame “SEQUENCE” has to be sent.
- send_RESOPPONENT: signal that, when active, it means the frame “RESOPPONENT” has to be sent.
- dataM[3..0] **(4 bits)**: Bus of 4 bits where the Peg A or Peg 1 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- dataC[3..0] **(4 bits)**: Bus of 4 bits where the Peg B or Peg 2 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- dataD[3..0] **(4 bits)**: Bus of 4 bits where the Peg C or Peg 3 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- dataU[3..0] **(4 bits)**: Bus of 4 bits where the Peg D or Peg 4 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- tx_empty: When low, it indicates that the UART is currently transmitting data and consequently busy. If high, it indicates that the UART is not transmitting data (i.e. the UART is idle).

OUTPUTS

- tx_data[7..0] **(8 bits)**: Data byte to be transmitted.
- ready_to_TX: Active high. When active it means that the transmitter is not transmitting any frame and is ready to be used.
- send: Synchronous signal, active high. It should be activated one clock period to start a new transmission (awakening signal). This signal can only be set to one when the output TX_EMPTY is high, i.e., when the UART is not currently transmitting any data.

b. tx_module

INPUTS:

- send: Synchronous signal, active high. It should be activated one clock period to start a new transmission (awakening signal). This signal can only be set to one when the output TX_EMPTY is high, i.e., when the UART is not currently transmitting any data.
- tx_data[7..0] **(8 bits)**: Data byte to be transmitted.

OUTPUTS:

- tx: Serial output. The serial transmission protocol is: START BIT (low level) + PAYLOAD (8 data bits, LSB first) + STOP BIT (high Level). While idle (not transmitting), TX is set to high level. The speed of the serial transmission can be configured by setting internal parameters. As a matter of fact, this configuration sets how many clock cycles a transmitted bit is held at the output (see waveform below). The figure shows that each bit transmitted is held for N clock cycles. This implies that for transmitting a byte, it is required (10*Clock period) seconds (10 = 8 payload bits + start bit + stop bit).
- tx_empty: When low, it indicates that the UART is currently transmitting data and consequently busy. If high, it indicates that the UART is not transmitting data (i.e. the UART is idle).

3.4. Receiver Block

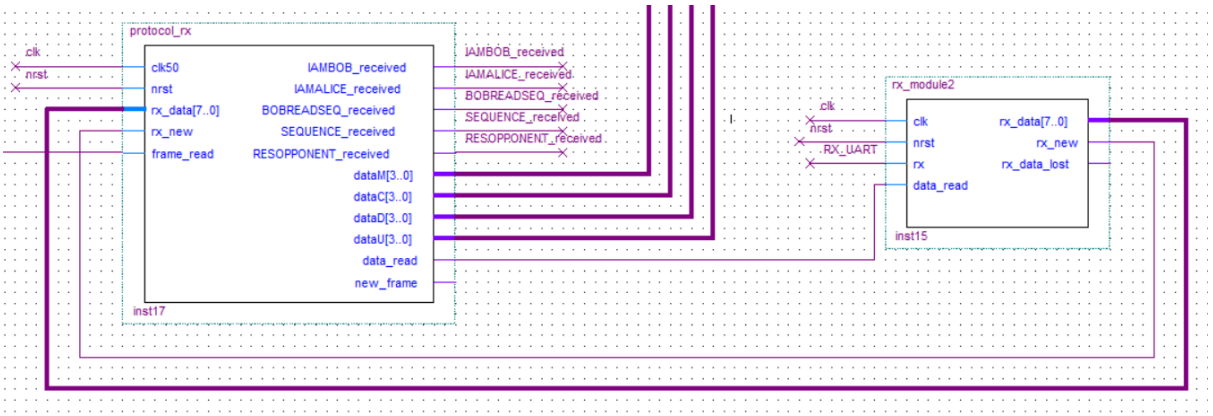

Figure 3.4.1. protocol_rx and rx_module blocks (*block_diagram.bdf* file from Quartus Software).

a. protocol_rx

INPUTS:

- rx_data[7..0] **(8 bits)**: Payload of the last byte received. To be read by another block.
- rx_new: Flag signal (active high). It is active when a new byte is received. This flag signal deactivates when the input DATA_READ is activated for one clock period.
- frame_read: ACK signal that acknowledges that the block “Bob_identifier” has received the frame “SEQUENCE”; “Alice_identifier” has received the frame “IAMBOB” or “BOBREADSEQ”; “mastermind_block” has received the frame “RESULTOPPONENT”; or “main_block” has received the frame “IAMALICE” or “IAMBOB”.

OUTPUTS:

- IAMBOB_received: Active high. When active means that the frame “IAMBOB” has been received. Signal connected to “main_block” and “Alice_identifier”.
- IAMALICE_received: Active high. When active means that the frame “IAMALICE” has been received. Signal connected to “main_block”.
- BOBREADSEQ_received: Active high. When active means that the frame “BOBREADSEQ” has been received. Signal connected to “Alice_identifier”.
- SEQUENCE_received: Active high. When active means that the frame “SEQUENCE” has been received. Signal connected to “Bob_identifier”.
- RESOPPONENT_received: Active high. When active means that the frame “RESOPPONENT” has been received.
- dataM[3..0] **(4 bits)**: Bus of 4 bits where the Peg A or Peg 1 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- dataC[3..0] **(4 bits)**: Bus of 4 bits where the Peg B or Peg 2 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- dataD[3..0] **(4 bits)**: Bus of 4 bits where the Peg C or Peg 3 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- dataU[3..0] **(4 bits)**: Bus of 4 bits where the Peg D or Peg 4 is transmitted, in case the frame to transmit is “SEQUENCE” or “RESOPPONENT”.
- data_read: This input should be activated by the block that reads the information received by this UART, to acknowledge that the last received byte has been read. Synchronous signal, active high. This signal should be active one clock period.

b. rx_module2

INPUTS:

- rx: Serial input. The serial transmission protocol is: START BIT (low level) + PAYLOAD (8 data bits, LSB first) + STOP BIT (High Level). As the transmitter sets the serial line to high level when the transmitter is idle, the receiver activates when detects that the input serial line goes to zero. The speed of the serial transmission can be configured by setting internal parameters. As a matter of fact, this configuration sets how many clock cycles a transmitted bit is supposed to be at the input. The configuration parameter must be the same at both the receiver and transmitter. As in the transmitter, if a bit is supposed to be held at the input for N

clock cycles, once the UART is activated, it takes (N*10 clock period) seconds to finish the reception of a byte.

- data_read: This input should be activated by the block that reads the information received by this UART, to acknowledge that the last received byte has been read. Synchronous signal, active high. This signal should be active one clock period.

OUTPUTS:

- rx_data[7..0] **(8 bits)**: Payload of the last byte received. To be read by another block.
- rx_new: Flag signal (active high). It is active when a new byte is received. This flag signal deactivates when the input DATA_READ is activated for one clock period.

3.5. Alice Identifier Block

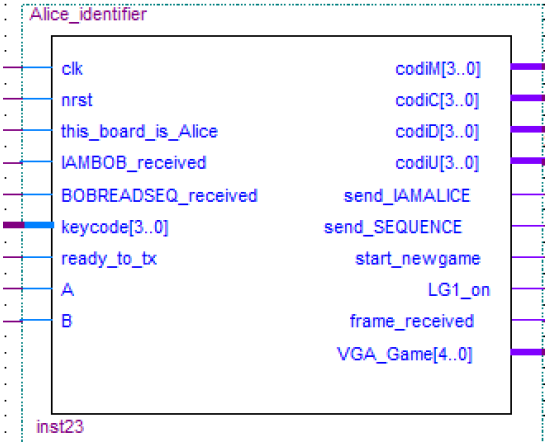

Figure 3.5.1. Alice_identifier block (*block_diagram.bdf* file from Quartus Software).

The aim of this block is to make possible that the user identifies themselves as the character Alice. It is connected to the Main Block and to the Mastermind/Game Block.

INPUTS:

- this_board_is_Alice: this output is activated when the user has been identified as Alice by the Main Block.
- IAMBOB_received: this flag is activated when the frame IAMBOB has been received in the receiver block. This flag has to be deactivated at the receiver by activating the output frame_received.
- BOBREADSEQ_received: this flag is activated when the frame BOBREADSEQ has been received in the receiver block. This flag has to be deactivated at the receiver by activating the output frame_received.
- keycode[3..0]: this input will not be used (most likely) but we leave it here in case we want to identify the code of the keys introduced by the user (when answering the questions asked to Alice).
- ready_to_tx: this input is activated when the transmitter block is ready to send data to the opponent (the tx bus is empty because it is not working right now).

- A: this input is activated when the key [A] is being pressed by the user (this will be used when Alice is answering the questions shown on screen, in order to accept the difficulty options).
- B: this input is activated when the key [B] is being pressed by the user (this will be used when Alice is answering the questions shown on screen, in order to decline the difficulty options).

OUTPUTS:

- codiM **(4 bits bus)**: 4 bits bus storing Peg A in the secret sequence.
- codiC **(4 bits bus)**: 4 bits bus storing Peg B in the secret sequence.
- codiD **(4 bits bus)**: 4 bits bus storing Peg C in the secret sequence.
- codiU **(4 bits bus)**: 4 bits bus storing Peg D in the secret sequence.
- send_IAMALICE: this output is activated when we want to send the frame IAMALICE to the opponent. The Identifier Block will have to wait for the input ready_to_tx to be activated in order to send this frame.
- send_SEQUENCE: this output is activated when we want to send the frame SEQUENCE to the opponent. The Identifier Block will have to wait for the input ready_to_tx to be activated in order to send this frame. The sequence sent will be the one stored in the codiMCDU outputs.
- start_newgame: this output will be activated when a new game has to start. This will be an input of the game block.
- LG1_on: this output is connected to the I/O manager indicating when the LG1 LED must be on. The LG1 must be on while the user is identified as the character Alice. This LED will be turned off in the Main Block when a new game starts.
- frame_received: this output deactivates the flags from the receiver that had the purpose of notifying the Identifier of an event.
- VGA_Game[4..0] **(5 bits bus)**: 5 bits bus that include information that the VGA interpreter will interpret as a hint to know what the VGA must plot on screen at each moment (for example, the questions asked to Alice must be plotted on the screen).

3.6. Bob Identifier Block

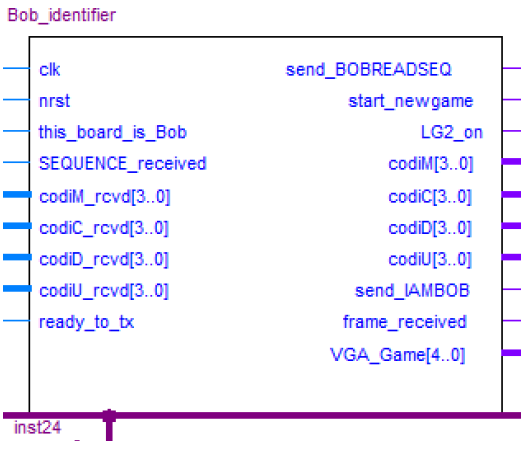

Figure 3.6.1. Bob_identifier block (*block_diagram.bdf* file from Quartus Software).

The aim of this block is to make possible that the user identifies themselves as the character Bob. It is connected to the Main Block and to the Mastermind/Game Block.

INPUTS:

- this_board_is_Bob: this output is activated when the user has been identified as Bob by the Main Block.
- SEQUENCE_received: this flag is activated when the frame SEQUENCE has been received in the receiver block. This flag has to be deactivated at the receiver by activating the output frame_received. The activation of this input is related to having the secret sequence from the opponent (Alice) in the MCDU output buses of the protocol rx block.
- codiM_rcvd **(4 bits bus)**: 4 bits bus storing Peg A in the received secret sequence.
- codiC_rcvd **(4 bits bus)**: 4 bits bus storing Peg B in the received secret sequence.
- codiD_rcvd **(4 bits bus)**: 4 bits bus storing Peg C in the received secret sequence.
- codiU_rcvd **(4 bits bus)**: 4 bits bus storing Peg D in the received secret sequence.
- ready_to_tx: this input is activated when the transmitter block is ready to send data to the opponent (the tx bus is empty because it is not working right now).

OUTPUTS:

We are including as inputs of the identifier block the secret sequence because then this sequence is stored in the block during the whole game (as in the Alice identifier block).

- send_BOBREADSEQ: this output is activated when we want to send the BOBREADSEQ frame to our opponent, meaning that we (Bob) has received the secret sequence from Alice.
- start_newgame: this output will be activated when a new game has to start. This will be an input of the game block.
- LG2_on: this output is connected to the I/O manager indicating when the LG2 LED must be on. The LG2 must be on while the user is identified as the character Bob. This LED will be turned off in the Main Block when a new game starts.
- codiM **(4 bits bus)**: 4 bits bus storing Peg A in the secret sequence.
- codiC **(4 bits bus)**: 4 bits bus storing Peg B in the secret sequence.
- codiD **(4 bits bus)**: 4 bits bus storing Peg C in the secret sequence.
- codiU **(4 bits bus)**: 4 bits bus storing Peg D in the secret sequence.
- send_IAMBOB: this output is activated when we want to send the frame IAMBOB to the opponent. The Identifier Block will have to wait for the input ready_to_tx to be activated in order to send this frame.
- frame_received: this output deactivates the flags from the receiver that had the purpose of notifying the Identifier of an event.
- VGA_Game[4..0] **(5 bits bus)**: 5 bits bus that include information that the VGA interpreter will interpret as a hint to know what the VGA must plot on screen at each moment (for example, the questions asked to Alice must be plotted on the screen).

3.7. Mastermind (Game) Block

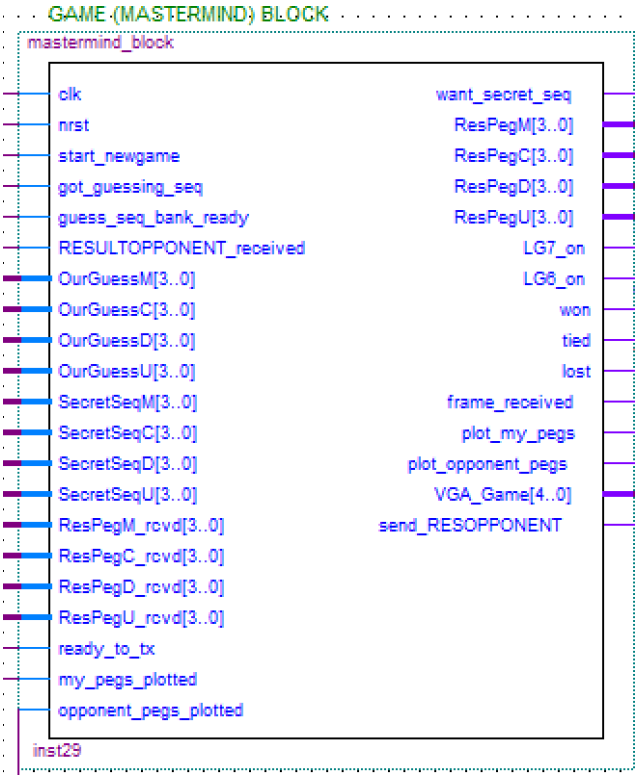

Figure 3.7.1. mastermind_block (*block_diagram.bdf* file from Quartus Software).

This block is one of the main system blocks, as it directs the functioning of the game itself (this is why it is called the Game/Mastermind Block). It is connected to both character identifiers (Alice and Bob), to the guessing sequence register, to the receiver and transmitter blocks and to the VGA interpreter.

INPUTS:

- start_newgame: this input is activated by one of the character identifiers (the one corresponding to the character of the FPGA). It indicated the moment when a new game can start.
- got_guessing_seq: it is activated by the register bank and tells the Game Block that the user has already introduced the guessing sequence of the current run.
- guess_seq_bank_ready: it is activated by the register bank when it is ready to store a new guessing sequence. The Game Block asks the register bank for a new guessing sequence when this input is high (meaning that the bank is free for us).
- RESULTOPPONENT_received: this flag is high when the frame with the result pegs from the opponent has arrived to the receiver block. When this flag is activated, the output frame_received has to be also activated to indicate to the receiver block that the RESULTOPPONENT_received flag can be deactivated.
- OurGuessM **(4 bits bus)**: 4 bits bus storing Peg A in the guessing sequence.
- OurGuessC **(4 bits bus)**: 4 bits bus storing Peg B in the guessing sequence.
- OurGuessD **(4 bits bus)**: 4 bits bus storing Peg C in the guessing sequence.
- OurGuessU **(4 bits bus)**: 4 bits bus storing Peg D in the guessing sequence.
- SecretSeqM **(4 bits bus)**: 4 bits bus storing Peg A in the secret sequence.

- SecretSeqC **(4 bits bus)**: 4 bits bus storing Peg B in the secret sequence.
- SecretSeqD **(4 bits bus)**: 4 bits bus storing Peg C in the secret sequence.
- SecretSeqU **(4 bits bus)**: 4 bits bus storing Peg D in the secret sequence.

We do not need as input the result pegs of the user because they are computed inside the Game Block. However, the opponent result pegs are needed to calculate if the game has finished (someone has won, lost or tied).

- ResPegM_rcvd **(4 bits bus)**: 4 bits bus storing Peg 1 of opponent result pegs.
- ResPegC_rcvd **(4 bits bus)**: 4 bits bus storing Peg 2 of opponent result pegs.
- ResPegD_rcvd **(4 bits bus)**: 4 bits bus storing Peg 3 of opponent result pegs.
- ResPegU_rcvd **(4 bits bus)**: 4 bits bus storing Peg 4 of opponent result pegs.
- ready_to_tx: this input is activated when the transmitter block is ready to send data to the opponent (the tx bus is empty because it is not working right now).
- my_pegs_plotted: this input will turn off the plot_my_pegs output flag, meaning that the pegs (both the guessing sequence and the result ones) have been already plotted on the screen.
- opponent_pegs_plotted: this input will turn off the plot_opponent_pegs flag, meaning that the opponent result pegs have already been plotted on the screen.
- send_RESOPPONENT: this flag is activated when from the Game Block we want to send our result pegs to the opponent.

OUTPUTS:

- want_secret_sequence: this output notifies the register (when it is ready) that the Game Block wants the user to introduce a new guessing sequence.
- ResPegM **(4 bits bus)**: 4 bits bus storing Peg 1 of the user result pegs.
- ResPegC **(4 bits bus)**: 4 bits bus storing Peg 2 of the user result pegs.
- ResPegD **(4 bits bus)**: 4 bits bus storing Peg 3 of the user result pegs.
- ResPegU **(4 bits bus)**: 4 bits bus storing Peg 4 of the user result pegs.

It is not necessary to output the result pegs of the opponent because they are already present in the MCDU outputs of the receiver block (protocol rx).

- LG7_on: this output is connected to the I/O manager indicating when the LG7 LED must be on. The LG7 must be on when the game is in progress (during all the game turns, from the moment when start_newgame is activated to the moment any of the finished game signals - won, tied, lost- is activated).
- LG6_on: this output is connected to the I/O manager indicating when the LG6 LED must be on. The LG6 must be on when the user can introduce a new guessing sequence, which means when the register bank is being used.
- won: this flag indicates to the main block that the game has ended and that the user has won.
- tied: this flag indicates to the main block that the game has ended and that the user has tied with the opponent.
- lost: this flag indicates to the main block that the game has ended and that the user has lost.
- frame_received: this output is activated in order to turn the flag RESULTOPPONENT_received off in the receiver block, notifying the receiver that the

Game Block has already been notified with the fact that the FPGA has received the result pegs from the opponent.

- plot_my_pegs: this flag indicates to the VGA interpreter that the pegs of the user guessing sequence and the result pegs of the user have to be plotted on screen by using VGA. This flag will be turned off when the VGA interpreter activates the input my_pegs_plotted in the Game Block.
- plot_opponent_pegs: this flag indicates to the VGA interpreter that the result pegs of the opponent have to be plotted on screen by using VGA. This flag will be turned off when the VGA interpreter activates the input opponent_pegs_plotted in the Game Block, meaning that the opponent pegs have already been plotted.
- VGA_Game[4..0] **(5 bits bus)**: 5 bits bus that include information that the VGA interpreter will interpret as a hint to know what the VGA must plot on screen at each moment (for example, the current game turn).

3.8. Main Block

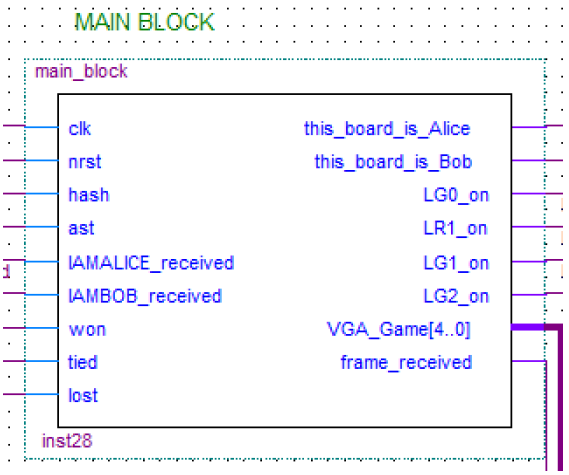

Figure 3.8.1. main_block (*block_diagram.bdf* file from Quartus Software).

INPUTS:

- hash: flag that is activated when the user maintains the [#] key pressed. This indicates the Main Block that the user wants to become Alice (if pressed at the start of the game) or that the user is ready to start a new game (if a game has ended and the result is being shown on screen).
- ast: flag that is activated when the user maintains the [*] key pressed. This indicates the Main Block that the user wants to become Bob (if pressed at the start of the game).
- IAMALICE_received: this flag is activated when the frame IAMALICE has been received in the receiver block. This means that the opponent has been identified as the character Alice, so the user in this FPGA can only be Bob from now on (unless a reset is produced). This flag has to be deactivated at the receiver by activating the output frame_received.
- IAMBOB_received: this flag is activated when the frame IAMBOB has been received in the receiver block. This means that the opponent has been identified as the character BOB, so the user in this FPGA can only be Alice from now on (unless a

reset is produced). This flag has to be deactivated at the receiver by activating the output frame_received.

- won: this flag indicates to the main block that the game has ended and that the user has won.
- tied: this flag indicates to the main block that the game has ended and that the user has tied with the opponent.
- lost: this flag indicates to the main block that the game has ended and that the user has lost.

OUTPUTS:

- this_board_is_Alice: this output is activated when the user has been identified as Alice. It activates the Alice Identifier Block.
- this_board_is_Bob: this output is activated when the user has been identified as Bob. It activates the Bob Identifier Block.
- LG0_on: this output is connected to the I/O manager indicating when the LG0 LED must be on. The LG0 must be on when the user can still choose a character.
- LG1_on: this output is connected to the I/O manager indicating when the LG1 LED must be on. The LG1 must be on when the user is Alice. It is an output of the Main Block only because this LED will have to be deactivated when starting a new game, when the character of the user in the previous game was Alice (if there has been a previous game):
- LG2_on: this output is connected to the I/O manager indicating when the LG2 LED must be on. The LG2 must be on when the user is Bob. It is an output of the Main Block only because this LED will have to be deactivated when starting a new game, when the character of the user in the previous game was Bob (if there has been a previous game):
- LR1_on: this output is connected to the I/O manager indicating when the LR1 LED must be on. The LR1 must be on when there has been an error when identifying the users as a specific character. For example, if an user wants to identify as Alice but the frame IAMALICE has already been received, this is considered as an user error and we have decided that the user should be notified of this incident.
- VGA_Game[4..0] **(5 bits bus)**: 5 bits bus that include information that the VGA interpreter will interpret as a hint to know what the VGA must plot on screen at each moment (for example, characters have to identify themselves and they have to see the two key-press options on the screen).
- frame_received: this output deactivates the flags from the receiver that had the purpose of notifying the Main Block of an event (such as IAMBOB_received or IAMALICE_received.

3.9. Video Graphics Array (VGA)

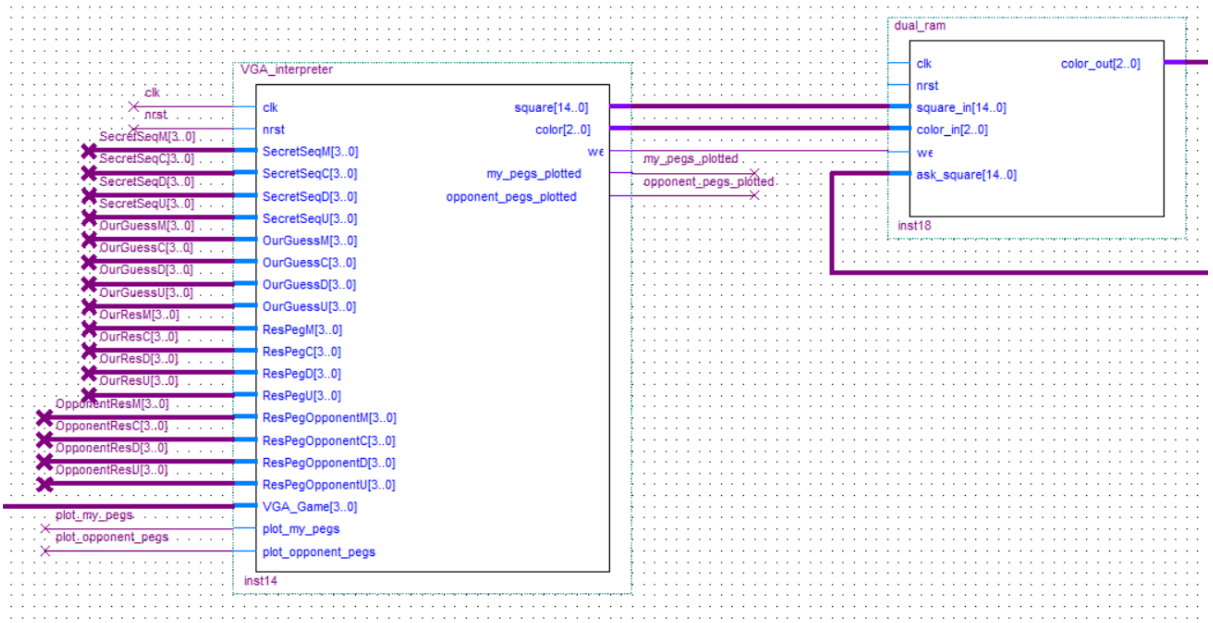

Figure 3.9.1. VGA_interpreter and dual_ram blocks (*block_diagram.bdf* file from Quartus Software).

a. VGA_interpreter

This block takes the secret sequence, guess code, result pegs and opponent result pegs and consecutively sends for each square the corresponding color. This block is aware of the current turn so it knows in which position a given sequence should be plotted. It also displays the initial screen, all the questions, and the game over screen.

INPUTS:

- SecretSeqM[3..0], SecretSeqC[3..0], SecretSeqD[3..0], SecretSeqU[3..0] **(4 bits)**: values of the colors of the 4 pegs of the user code, i.e., the correct code generated by Alice. These signals come from Alice_identifier if the user is Alice or from Bob_identifier if the user is Bob.
- OurGuessM[3..0], OurGuessC[3..0], OurGuessD[3..0], OurGuessU[3..0] **(4 bits)**: values of the colors of the 4 pegs of the current guess code. These signals are the 4 last colors that the user selected. The signals come from the Guessing Sequence Register and are updated each time the user presses a new color when the user has to introduce the guess code.
- ResPegM[3..0], ResPegC[3..0], ResPegD[3..0], ResPegU[3..0] **(4 bits)**: These are the result pegs of the user that come from the Game block.
- ResPegOpponentM[3..0], ResPegOpponentC[3..0], ResPegOpponentD[3..0], ResPegOpponentU[3..0] **(4 bits)**: These are the results pegs of the opponent that is received in the Receiver Block.
- VGA_Game[4..0] **(5 bits)**: This value indicates the number of the current turn. There are 10 turns, plus the turn you see the correct sequence, plus the initial and final

screens where can be seen the difficulty options and if the user is the winner, loser or there is a tie.

- plot_my_pegs: Active high. This signal is active when the user has put the guess code and its result pegs have been calculated by the Game block, so these can be plotted with the VGA. Connected to “Game”
- plot_my_opponent_pegs: Active high. This signal is active when the result pegs of the opponent have been received and can be plotted with the VGA. Connected to “Game”

OUTPUTS:

- square[14..0] **(15 bits)**: This value indicates the square identifier. There are a total of 160x120=19200 squares in the screen. Connected to “dual_ram”.
- color[2..0] **(3 bits)**: This value indicates the color that corresponds to the signal “square[14..0]”. Connected to “dual_ram”.
- we: Active high. If active means that the two signals above are written in “dual_ram”. Connected to “dual_ram”.
- my_pegs_plotted: Flag signal connected to “Game”. Is activated when all the squares corresponding to the guessed code and the result pegs of the users have been written in “dual_ram”.
- opponent_pegs_plotted: Flag signal connected to “Game”. Is activated when all the squares corresponding to the opponent result pegs have been written in “dual_ram”.

b. dual_ram

This block is used to write and read the different colors for each square in a RAM. The block “VGA_interpreter” stores data in this RAM and the block “wr_memory” reads from this RAM by asking the value of a given square.

INPUTS

- square_in[14..0] **(15 bits)**: value of the square from “dual_ram” from “VGA_interpreter”.
- color_in[2..0] **(3 bits)**: color of the square “square_in” from “VGA_interpreter”.
- we: Write Enable from the “VGA_interpreted”. If active saves the color of the square from “VGA_interpreter” to the RAM.
- ask_square[14..0] **(15 bits)**: Signal from “wr_memory” in which is indicated the square that wants to know its color stored in the RAM.

OUTPUTS

- color_out[2..0] **(3 bits)**: Signal connected to “wr_memory”. The signal is the correspondent color from the square that is asked in “ask_square”.

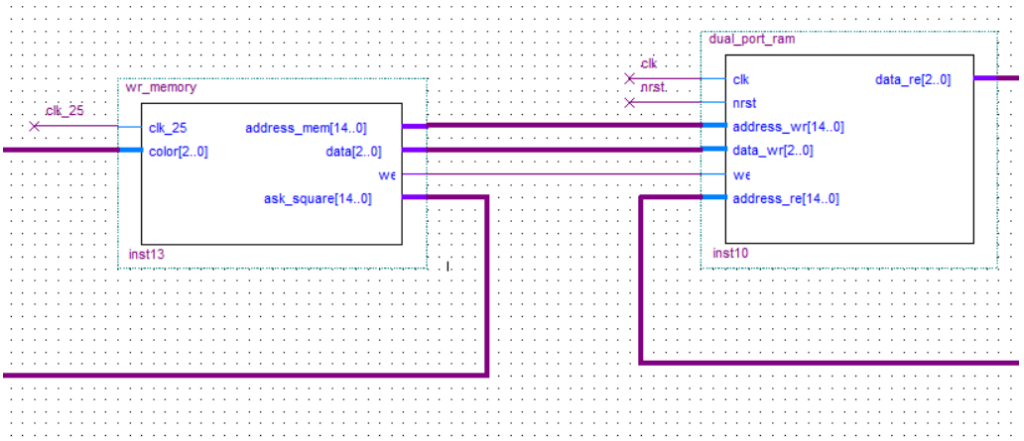

Figure 3.9.2. wr_memory and dual_port_ram blocks (*block_diagram.bdf* file from Quartus Software).

c. wr_memory

This block asks the “dual_ram” the color of a given square and sends this information to the RAM in the block “dual_port_ram” in an address that is understandable to the “VGA_SYNC”. With this address, the position of the pixel in question can be determined.

INPUTS

- color[2..0] **(3 bits)**: color of the square that is asked in “ask_square”. Connected to “dual_ram”.

OUTPUTS

- address_mem[14..0] **(15 bits)**: address that corresponds to the square asked in “ask_square”, is the address in which the block “dual_port_ram” will store the color received in “color” and which is given in “data”
- data[2..0] **(3 bits)**: color that will be stored in the block “dual_port_ram” in the address “address_mem”.
- we: Active high, enables to write in the RAM of “dual_port_ram”.
- ask_square[14..0] **(15 bits)**: here, this block sends the square identifier to “dual_ram” to ask for the correspondent color, which is received in “color”.

d. dual_port_ram

This block is used to write and read in a RAM the different colors for each address of a particular pixel. The block “wr_memory” writes in this RAM and the block “VGA_SYNC” reads.

INPUTS

- address_wr[14..0] **(15 bits)**: address in which the “data_wr” will be written in the RAM. Connected to “wr_memory”.
- data_wr[2..0] **(3 bits)**: value (color) that will be stored in the RAM in “address_wr”.
- we: Write Enable from the block “wr_memory”.

- address_re[14..0] **(15 bits)**: address to read the value stored in the RAM, it's value will be given in "data_re". It comes from the block "address_generator".

OUTPUTS

- data_re[2..0] **(3 bits)**: data readed from the RAM in the address "addres_re". Connected to "VGA_SYNC".

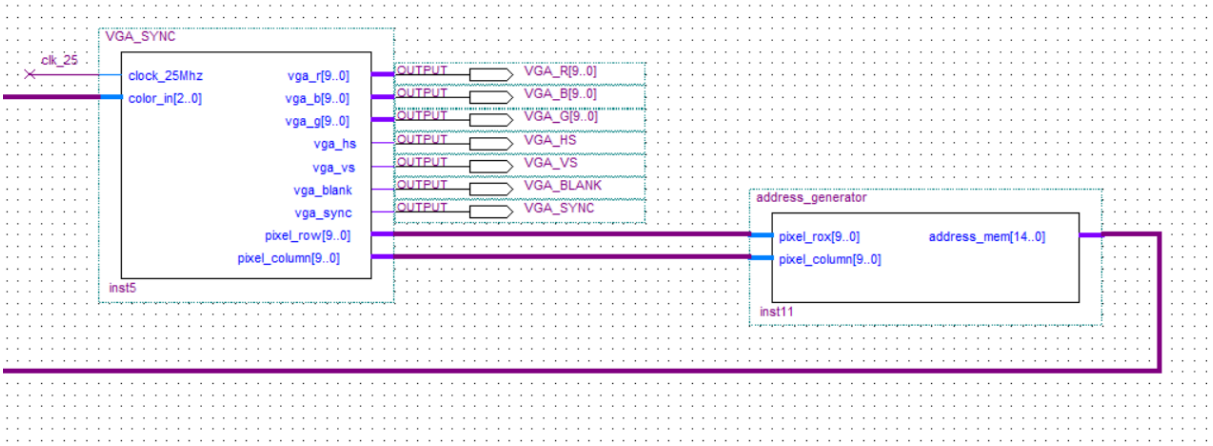

Figure 3.9.3. VGA_SYNC and address_generator blocks (*block_diagram.bdf* file from Quartus Software).

e. VGA_SYNC

This block is the one connected directly to the screen. It displays synchronously the different colors for each pixel in the screen. It informs the block "address_generator" of the next pixel to display, and then the value is received from the block "dual_port_ram".

INPUTS

- color_in[2..0] **(3 bits)**: color that will be shown in the screen in the pixel that was indicated in "pixel_row" and "pixel_column". Connected to "dual_port_ram".

OUTPUTS

- vga_r[9..0], vga_b[9..0], vga_g[9..0] **(10 bits)**: colors displayed in the screen in the correspondent pixel which depends of the time.
- vga_hs: horizontal synchronizer.
- vga_vs: vertical synchronizer.
- vga_blank: when active, all the screen is blank.
- vga_sync:
- pixel_row[9..0] **(10 bits)**: row value of the next pixel to show in the screen.
- pixel_column[9..0] **(10 bits)**: column value of the next pixel to show in the screen.

f. address_generator

This block takes the value of the row and column of a pixel and gives the corresponding address, taking into account the maximum number of bits that the RAM can support.

INPUTS

- pixel_row[9..0] **(10 bits)**: row value of the next pixel to show in the screen.
- pixel_column[9..0] **(10 bits)**: column value of the next pixel to show in the screen.

OUTPUTS

- address_mem[14..0] **(15 bits)**: address that contains the information of the pixel in a given row and column.

10. User Interface (UI)

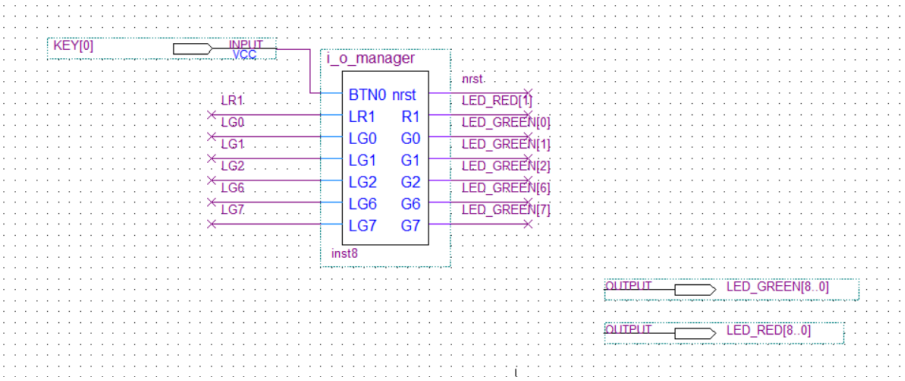

Figure 3.10.1. i_o_manager block (*block_diagram.bdf* file from Quartus Software).

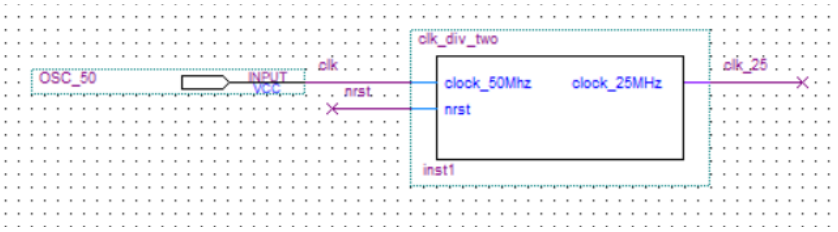

Figure 3.10.2. clk_div_two block (*block_diagram.bdf* file from Quartus Software).

a. i_o_manager

This block is used to connect the signals of the LEDs and the nrst to the board.

INPUTS

- BTN0: nrst is active when the button P0 is pressed
- LR1: Red led, activated from the main block.
- LG0: Green led, activated from the “main” block. It indicates that the user can choose to be Alice or Bob.
- LG1: Green led, activated from “Alice identifier” block. It indicates that the user is Alice.
- LG2: Green led, activated from “Bob identifier” block. It indicates that the user is Bob.
- LG6: Green led, activated from the “game” block. It indicates that the user can enter a sequence.
- LG7: Green led, activated from the “game” block. It indicates that the game is in progress.

OUTPUTS

- nrst: Active low, asynchronous.
- R1.
- G0, G1, G2, G6, G7.

b. clk_div_two

INPUTS

- clock_50Mhz: clock with a frequency of 50 MHz. The one used in almost every block.

OUTPUTS

- clock_25MHz: clock with a frequency of 25 MHz. It is used for the blocks “VGA_SYNC” and “wr_memory”.

Annex: VGA SPECS

Here it will be explained the general ideas of how the game will be displayed on the screen during the game.

The screen has a resolution of 640x480 pixels. The idea is to divide the screen in 20x15 squares (so each square is made of 32x32 pixels), where each square can have a peg plotted. These squares can also be divided in 8x8 subsquares, then we would have 160x120=19200 subsquares (each subsquare is made of 4x4 pixels).

With this we can plot the pegs with a circular form by defining the color for each subsquare.

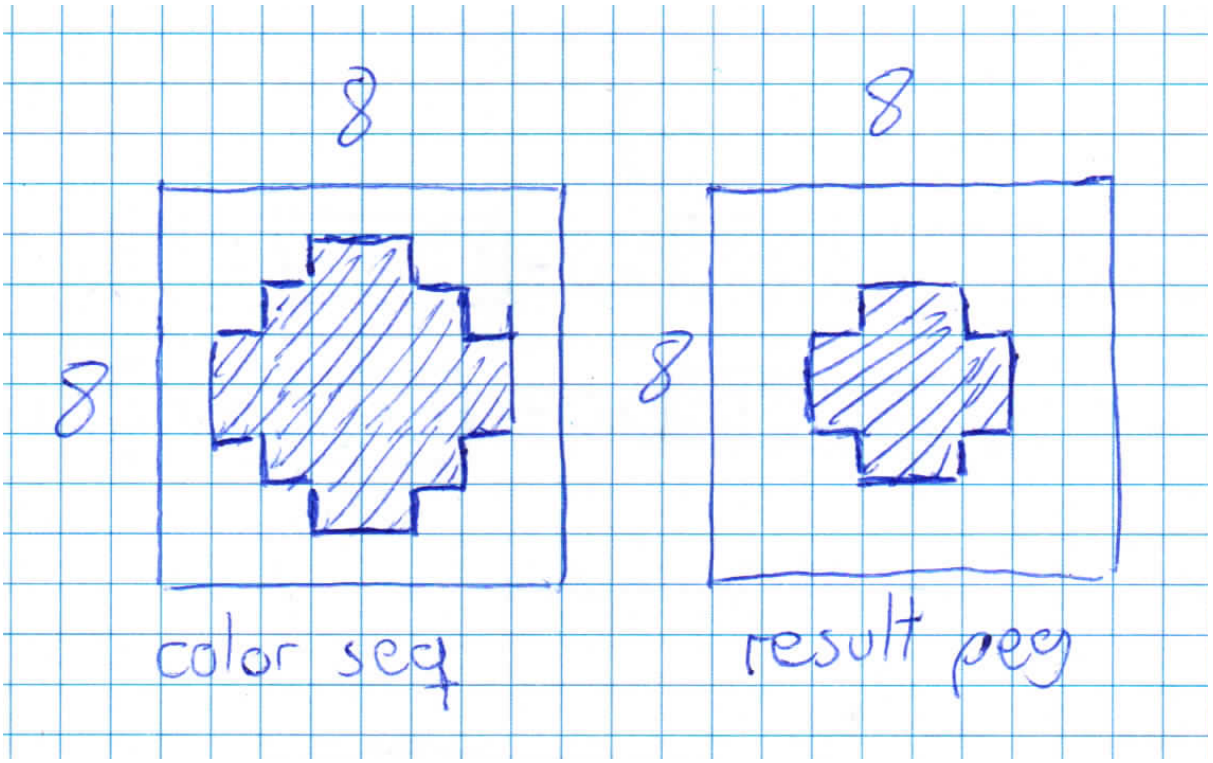

Figure A.1. Pixel representation of the sequence pegs and the result pegs.

The bits necessary to specify the address for the columns are 8, and for the rows 7. Then we have an address of 15 bits, which is in the range of the limit of bits that the RAM supports.

Then, in the screen we would have the following:

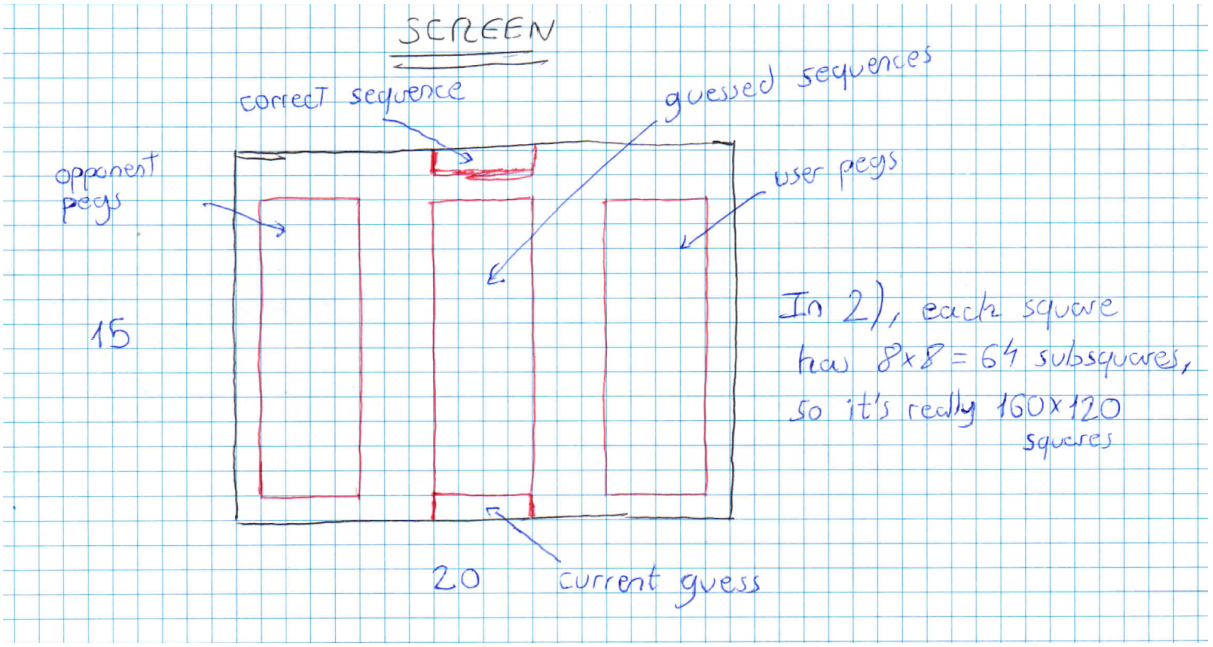

Figure A.2. Representation of the squares that are plotted in the screen.

Where each square of the paper represents a square, so in each square a peg could be inserted.