<KissGoodbye> Final Project Report

(Documentation has two parts in this project. One is this final report, including details about our design idea, the block diagram and descriptions of our Verilog modules. The other is a README file, mainly about the rules of game and some corner cases.)

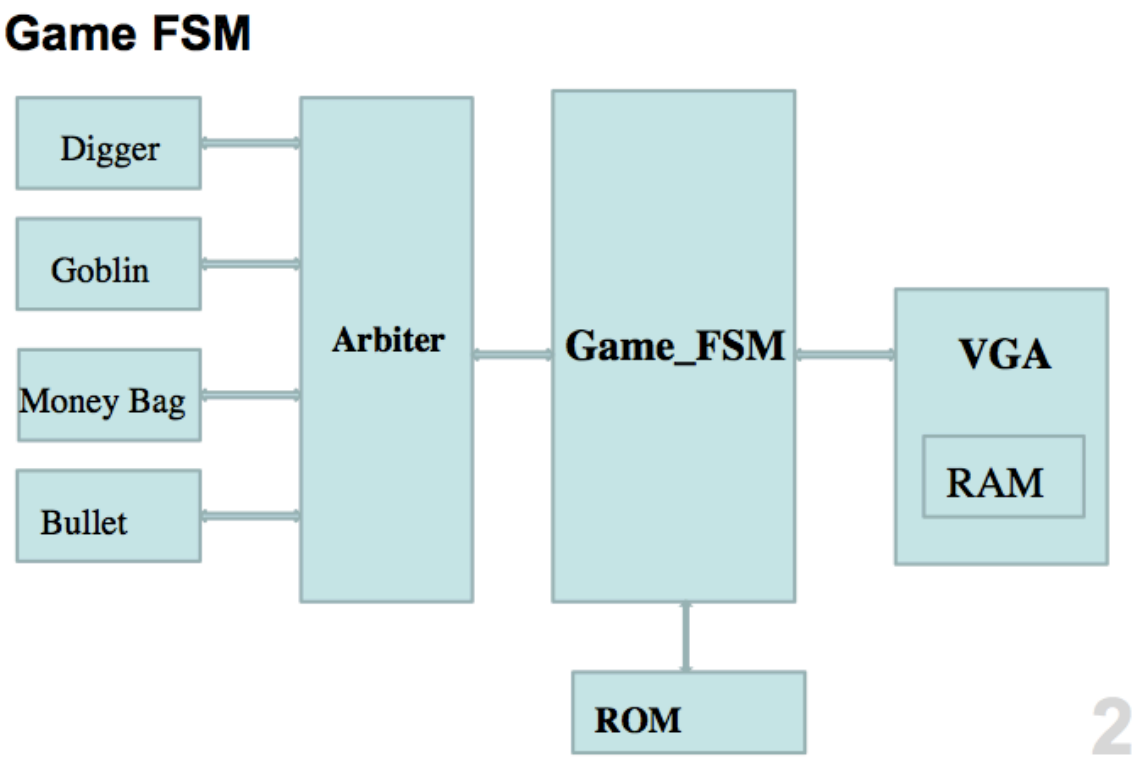
- Team Name: Kiss Goddbye
- Team Member: Yenai Ma, Siwan Yang, Yinan Liu and Xiangchao Huang.

- Short Description of the project:
  Our project is the implementation of a video game called "Digger Game" on FPGA Board Nexys 4. This is an old game created in 1983 by Windmill Software and it is an object-oriented design. There are multiple objects, such as the digger, the bullet shoot by the digger only, the goblins and the moneybags. Our group has already finished basic functionalities of this game but there are still some minor bugs, which are described in the ReadMe.txt file. Brief Description of basic functionalities:
  a. The digger controlled by the keyboard (single player)
  b. Several (3) goblins chasing after the digger (AI)
  c. Collecting diamonds to get score
  d. Collecting the remainder to get score after money bag crush
  e. To kill the goblin by shooting
  f. Using one money bag to kill chasing goblins
  g. Level 1 & Level 2(our game has two levels)

- Design Idea (the top-level design is dramatically significant)

There are many objects shown on the map, such as digger, goblins, diamonds and moneybags. Except diamonds, all the other objects need read and write memory. However, the number of memory ports on FPGA is limited. Therefore, an arbiter is designed to deal with different type requests from different objects. All the objects except diamonds are connected to the arbiter. Before doing any actions, they will all send requests to the arbiter. The arbiter picks up only one request from all the requests and send to game FSM in round-robin fashion. Game FSM handles the request from the objects one by one. Meanwhile, it communicates with the VGA RAM as well.

- Block Diagram



- Description of Main Modules:
  (Logic)
  Request Protocol: Before objects doing any action, they need to send requests, and then "req" signal is asserted. "req" keeps high until ACK/NACK is asserted.

  req: _____⌈_____⌉_____

  ACK/NACK: _____⌈⌉_____

  Status:(16-bit)

Exit	x	y	Direction	Type
(2)	(4)	(4)	(2)	(4)

  All object modules are following the protocol above and has similar interface.

  a. Digger:
  Properties: collecting diamond; collecting the remainder of moneybags; shooting the goblins; digging a hole on the map; surviving from falling moneybags
  Interface Description: Digger module handles the keyboard input and "req" signal is triggered by the input of keyboard. After ACK/NACK, "req" is de-asserted. Meanwhile, req_type and req_content are both sent to the arbiter as well as its own status.

  b. Goblins (AI):
  Properties: chasing after the digger; killing digger; being killed by falling

moneybags or bullets, rebirth after being killed (always 3 goblins running on the screen)

Interface Description: The moving decision is based on the previous direction and map data of 4 neighbor nodes.

Preference:

1. Previous direction available && Reduce distance
2. Turning available && Reduce distance
3. Previous direction available
4. Turning available
5. Return (when the goblin meets the boundary)

c. Moneybags:

Properties: falling down if the block right below is empty; killing the goblins; changing to the remainder if dropping down more than one layers; remain the same if dropping down only one layer; cannot be passed through or collected by goblins

Interface Description: Moneybag Module "req" signal always keeps high unless ACK/NACK comes. After ACK/NACK, "req" remains high again. Meanwhile, req_type and req_content are both sent to the arbiter as well as its own status.

d. Bullet:

Properties: being shot by the digger only; killing the goblins; not able to be shot until the previous one disappears

Interface Description: For there is a speed limitation of bullet, only counter counts to certain number, "req" is asserted. The rest is same as Moneybags Module.

e. Score System:

Properties: killing one goblin – 15; collecting one diamond – 5; collecting the remainder of moneybag – 20

Interface Description: add score based different types (diamond, the remainder of moneybag and goblin)

f. Arbiter: it picks up only one request from all the requests and send to game FSM in round-robin fashion. And it returns the ACK/NACK and the wr signal from the game FSM back to corresponding module.

g. FSM: it handles all the requests from the arbiter (main logic module). More details are shown in Verilog Code. The game FSM has these main states: INIT, HANDLING_REQ, GAME_OVER or GAME_WIN. When the rst is asserted, the state will stay in the INIT state until the reset session finishes.

During the reset session, the game FSM will copy the map in the ROM into VGA ram. After the reset_done signal is asserted, the state will goes to HANDLING_REQ state. Now the game FSM will listen to all the request signals from all the objects. When the score reaches 300, the game_win signal will be asserted. And the level number will go up by one. If the digger is eaten by the goblin, the state will go to the GAME_OVER state. Under the HANDLING_REQ state, there are several sub states. The first state is WAITING_REQ. When the game FSM is under this state, it will listen to all the request inputs from objects. Whenever there is a request come, it will enter READING_REQ_CONTENT. Since the request is always moving one object to another place, the collision test is required to make sure whether this object will hit something. For example, if the bullet hits one goblin, it needs to remove the goblin and bullet. So after the READING_REQ_CONTENT state, it will enter the WRITING_ORIG_POS. In the bullet-hit-goblin case, the bullet will be removed in this state. After that, the state machine will enter the WRITING_REQ_POS, which corresponds to removing goblin in this case. After that, it will return to WAITING_REQ state.

(Display)

a. VGA:

Properties: the display module of our game with one RAM included. The full screen is divided into 15x10 blocks in our design. Accordingly, the VGA ram depth is 150 as well. Each block is represented by one 4-bit value in the RAM. The VGA will keep display the content in the VGA RAM to the screen. our team logo ”Kissgoodbye” also showing up on the screen; besides as well as a score system

Interface Description: The main interface has two parts. The first part is the interface between VGA and game FSM. The game FSM will update VGA RAM accordingly after handling the requests from the objects via this interface. The second part is the interface from the VGA module to all the VGA output pins on the board.

- Difficulties:
  a. So many objects need Memory Read & Write (Sequential logic is hard)
  b. The type of requests from different objects is different
  c. The action varies from one object to another
  d. Object collision detection is complicated.
  e. Position information need to be kept all the time
  f. Many cases result in goblins being killed, digger being killed or goblin rebirth