SUSTech CS207 Digital Design Project Piano

A Mini Piano Toy ~~not playable~~

Introduction

In this semester project, we implement a piano toy with learn mode
and music player on the Xilinx(TM) FPGA EGO1 board, and the code
is written in `Verilog`. Inside the piano, we put four songs for you to enjoy,
`Twinkle Twinkle Little Star`, `Haruhikake`, `Ode to Joy` and `Jingle Bell`.
The implementation seems quite simple and straight forward, but it works pretty well.
That's exactly what we need. Anyway, enjoy your time with the toy!

Features

- [x] Mode Switching & Display
- [x] Free Mode
  - [x] Play Notes
  - [x] Three Octaves (C3 - C5)
- [x] Autoplay Mode
  - [x] Play Songs Automatically
  - [x] Index of Song Display
  - [x] Song Switching
  - [x] LED Cue of Notes
  - [x] Gap between notes
  - [x] Pause during Play (*Bonus*)
  - [x] Support Crotchet and Quaver (*Bonus*)
- [x] Learning Mode
  - [x] LED Cue during Learning
  - [x] Evaluation of Learning
  - [x] Multiple User and profile
- [ ] Key Adjustment (*Bonus*)
- [ ] VGA Display (*Bonus*)
- [ ] Song Record & Replay (*Bonus*)
- [ ] Detailed Evalution (*Bonus*)

User Manual

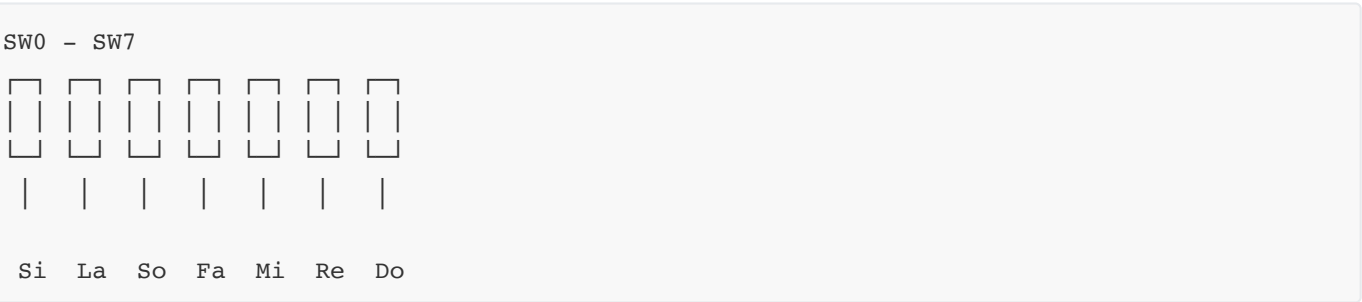
When turning on the board, a welcome message "HELLO" will be displayed.
And then you can start using it by changing the mode first.
The tiny switches of SW8 has two functions. The right most three switches is for changing mode

Mode	Switch	Function
Free	001	Play the piano
Auto	011	Play songs automatically
Learning	111	Play throught the instruction

And the left most two switches is for changing user in learning mode.
When you enter a mode, the name of current mode will be displayed on the segtube.

Free Mode

The switches (SW6 - SW0) is the keys of piano

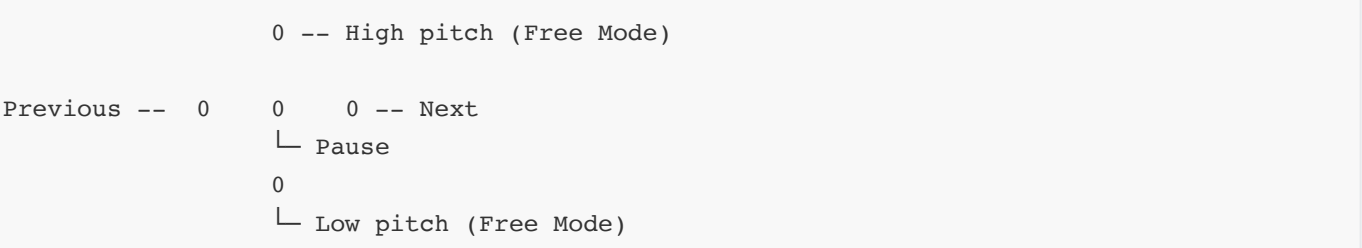


When you turn on one of them, the corresponding note will be played.
Note that more than one keys turned on at the same time will not play any notes.
On the right of the board, the up button(S4) is used to play the higher octaves and the down button is for the lower octaves, e.g., when you turn on the key `Do` while pressing the up button, the piano will play `C5` instead of `C4`.

Auto Mode

When you feel tired of playing by yourself, you can ask the board to play songs automatically by switching into auto mode. The piano has 4 songs built in (if you want add more, feel free to notify me in **Issue**), `Twinkle Twinkle Little Star`, `Haruhikage`, `Ode to Joy` and `Jingle Bell`.

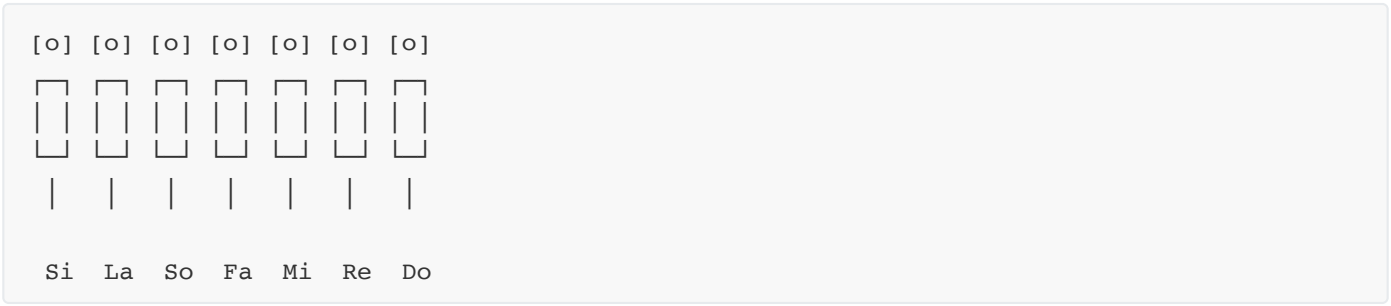
The song is switched by the buttons, left for previous song, right for next song. And the middle button can control the pause of song if you'd like it to pause for a while.
When you select a song, press the pause button and it will start playing.



On the segment tube, you can see clearly the index of current song and above the keys,
LEDs of corresponding note played currently will be lighted.

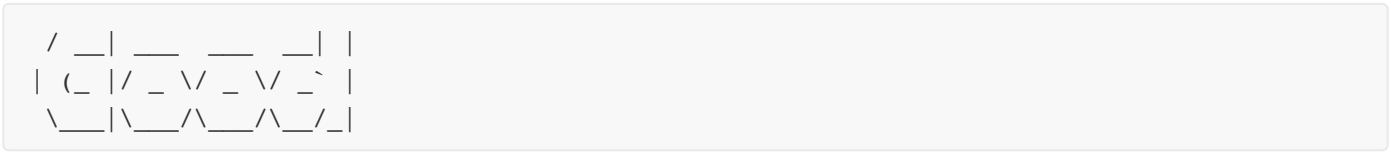
Learning Mode

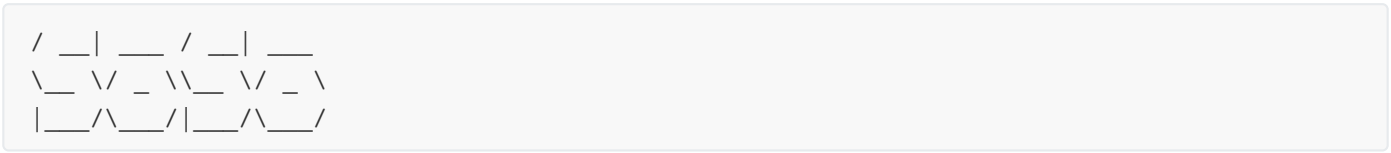
After seeing how songs are played, perhaps you'd love to play them on your own. Similar to the auto mode, but this time, you should play the notes with the help of LED indicators.

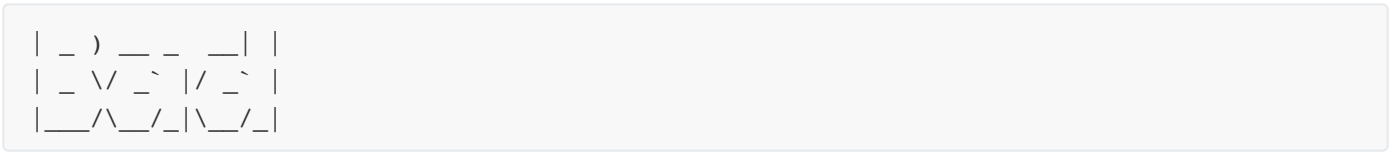


When the LED is on, you must turn on the switch(key) and turn off it when the LED is off AS SOON AS POSSIBLE.
You can change the songs as in the Auto Mode.

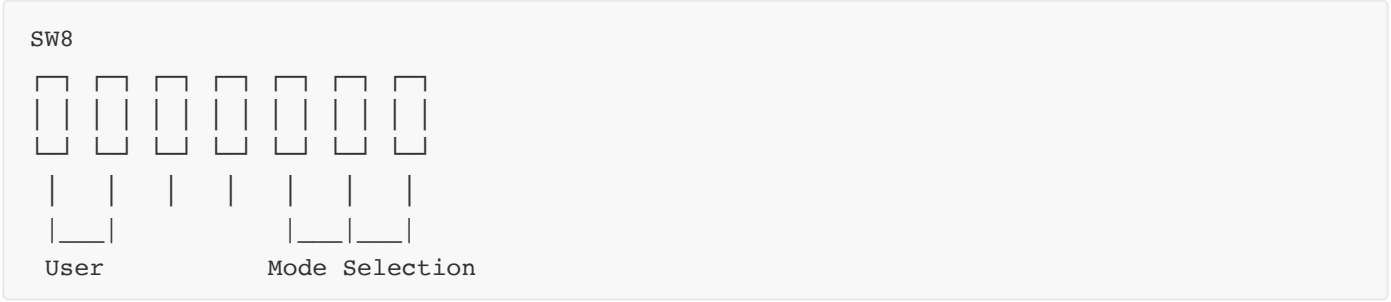
After completely learning a song, you can see the grade of your learning on the segment tube. We offer three levels of grade based on the latency of your pressing.



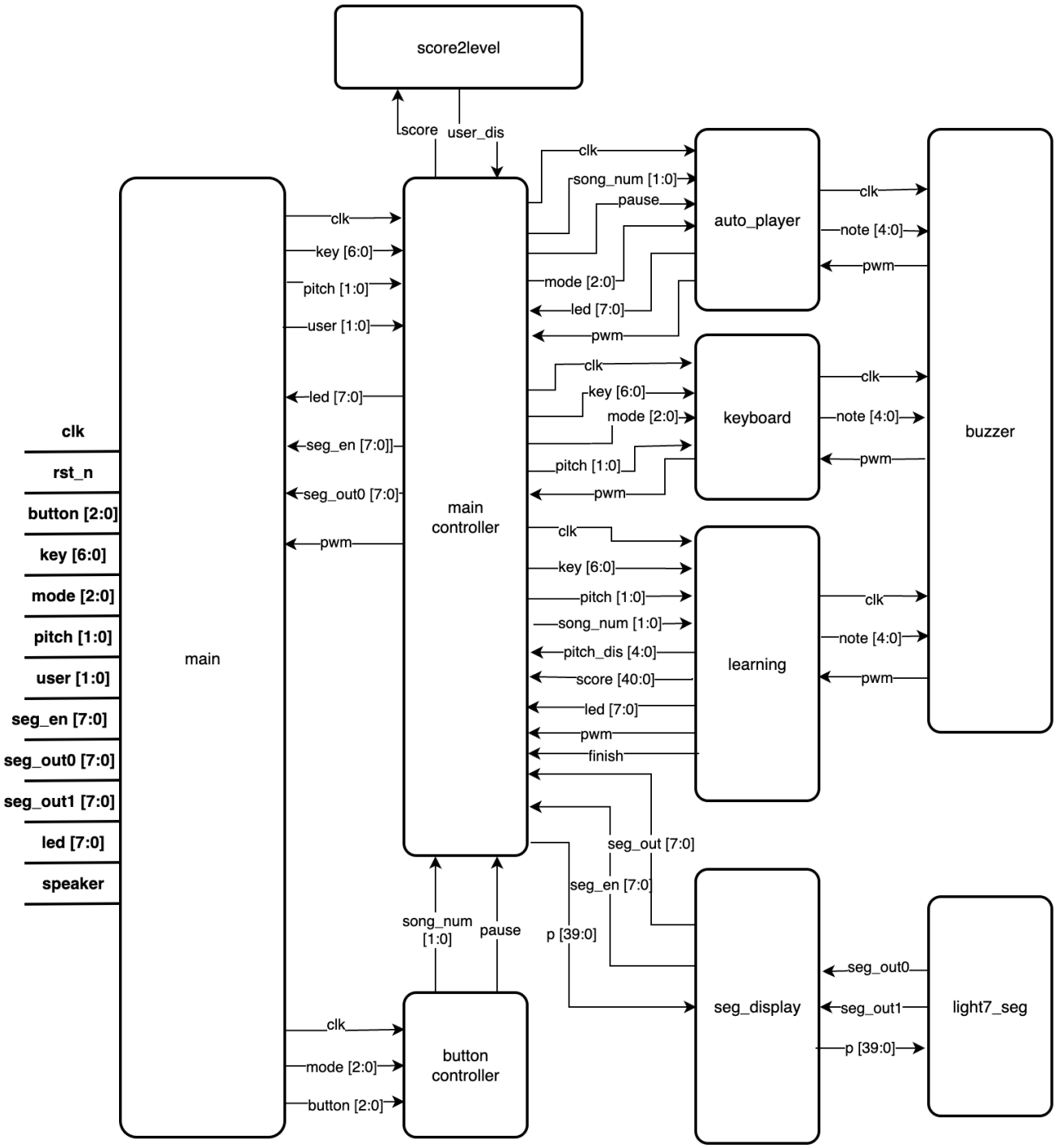




In this mode, you have the first three songs to learn. And the position of the last song is occupied by the user profile, which is used to display your average score. You can change the user by the left most two switch of SW8.



Architecture



Modules

`main_controller` Module:

- **Input:** clk, rst_n, key, pitch, user, mode, song_num, pause
- **Output:** speaker, seg_en, seg_out0, seg_out1, led

- **Description:** Select the signals from the three major submodules and control the information to display for each mode, based on the `mode` input. The three major submodules, i.e. `autoplay`, `keyboard` and `learning` is initialized here and the output is controlled by this module. Meanwhile, the record of scores for each user is stored and the conversion dof score to level is done here.

`button_controller` Module:

- **Input:** clk, mode, button
- **Output:** song_num, pause
- **Description:** This module has two finite state machines, the one to control the current index of song for `autoplay` and `learning` module, and the one to control the pause of song in `autoplay` module. The conversion of states is determined by the mode and button input. The conversion occurs only when the mode is in either of them, and the button is debounced.

`autoplay` Module:

- **Input:** clk, mode, pause, song_num
- **Output:** speaker, led
- **Description:** The main module for Auto Mode. It controls the sequence of playing notes and the tempo in each song accoring to the *song_num*, which is implemented by a state machine whose states are the indices of notes in a song. And the control signal *mode* is for reset to make sure the song is played from head, and the other control signal *pause* is for pause in a song. It will output a pwm signal generated from `buzzer`. This module seems to simulate the user input of `keyboard`.

`keyboard` Module:

- **Input:** clk, mode, key, pitch
- **Output:** speaker
- **Description:** The main module for Free Mode. The note to be played is controlled by user through the input key and pitch. The two inputs together decide the note. The module is powered by a table which converts key and pitch into note encoding in `buzzer`. The module is reset by the mode signal if it does not match the code of Free Mode.

`learning` Module:

- **Input:** clk, mode, key, pitch, song_num
- **Output:** speaker, led, finished, score
- **Description:** The main module for Learning Mode. The module inherits the functions of `keyboard` and `autoplay` and extends with two tasks. The tasks for this module is to output the LED indicators to guide user, to record the score of user. The former task is implemented by adding a output `led` in the state machine. And the latter task is defined based on the rule that score is calculated by the opposite of cumulative latency in a song. The latency is counted when the LED is on but the key is not turned on.

`buzzer` Module:

- **Input:** clk, note

- **Output:** speaker
- **Description:** Generate PWM signal for the input note. It has a table converting the encoded note to the frequency of PWM signal.

`seg_display` Module:

- **Input:** clk, rst_n, p0, p1, p2, p3, p4, p5, p6, p7
- **Output:** seg_en, seg_out0, seg_out1
- **Description:** Generate control signal for 7 segment display based on the input data. The data is encoded by the table from the submodule `light7_seg`

`light7_seg` Module:

- **Input:** seg_in
- **Output:** seg_out
- **Description:** Convert the information encoded by the number to the signals of 7segment. It maintains a character table that might be used in the project. Through this way we can simplified the expression of characters.

`score2level` Module:

- **Input:** score
- **Output:** p0, p1, p2, p3
- **Description:** Calculate the level of the score that the current user get in the learning mode, and convert it into the input that seg_display accepts.

Port Binding

Please refered to piano_constr.xdc

Port	Pin	Description
clk	P17	100MHz Clock
rst_n	P15	reset
speaker	H17	PWM
pwm_ctrl	T1	disable 3.5 mm
key[0]	R1	Do
key[1]	N4	Re
key[2]	M4	Mi
key[3]	R2	Fa
key[4]	P2	So
key[5]	P3	La

key[6]	P4	Si
pitch[0]	R17	low pitch (C3)
pitch[1]	U4	high pitch (C5)
button[0]	V1	Previous Song
button[1]	R15	Pause
button[2]	R11	Next Song
mode[0]	T5	mode input 1
mode[1]	T3	mode input 2
mode[2]	R3	mode input 3
user[0]	U2	user input 1
user[1]	U3	user input 2
led[7]	F6	preserved for debug
led[6]	G4	indicator for Si
led[5]	G3	indicator for La
led[4]	J4	indicator for So
led[3]	H4	indicator for Fa
led[2]	J3	indicator for Mi
led[1]	J2	indicator for Re
led[0]	K2	indicator for Do
seg_en[7]	G2	eighth 7segment enable
seg_en[6]	C2	seventh 7segment enable
seg_en[5]	C1	sixth 7segment enable
seg_en[4]	H1	fifth 7segment enable
seg_en[3]	G1	forth 7segment enable
seg_en[2]	F1	third 7segment enable
seg_en[1]	E1	second 7segment enable
seg_en[0]	G6	first 7segment enable
seg_out0[7]	B4	first four 7segments control a
seg_out0[6]	A4	first four 7segments control b

seg_out0[5]	A3	first four 7segments control c
seg_out0[4]	B1	first four 7segments control d
seg_out0[3]	A1	first four 7segments control e
seg_out0[2]	B3	first four 7segments control f
seg_out0[1]	B2	first four 7segments control g
seg_out0[0]	D5	first four 7segments control h
seg_out1[7]	D4	last four 7segments control a
seg_out1[6]	E3	last four 7segments control b
seg_out1[5]	D3	last four 7segments control c
seg_out1[4]	F4	last four 7segments control d
seg_out1[3]	F3	last four 7segments control e
seg_out1[2]	E2	last four 7segments control f
seg_out1[1]	D2	last four 7segments control g
seg_out1[0]	H2	last four 7segments control h

Implementation of Bonus

Pause: The pause feature is supported by two parts, the finite state machine of it with button debounce in `button_controller` and the holding for paused note in `autoplay`.
The state machine has two states, 1 for **pause** and 0 for **play**.
The inital state is 1 since the song is started when user confirms. When the user pressing the pause button, the state of pause is flipped.

Flexible Tempo: Previously the duration of a note played is fixed. By adding a new argument to indicate the divison of duration, the crotchet and quaver are achieved since the duration of them is half and quarter. Therefore, the state machines of Auto Mode and Learning Mode have two outputs, `note` indicating the note to be played and `note_len` indicating the division of it.
And in the counter for the duration, the termination of counting will be `cnt == gap / note_len` instead of simply `cnt == gap`.

Contributors

- **@Ben Chen** *Task:* Program (50%)
- **@Zhuo Wang** *Task:* Test (50%)

Development Schedule

Task	Start time	Detail	Status	Estimating	Actual
Main Module	2023-12-10	Do the wire planing and architecture design	Finished	3 Days	2 Days
Buzzer Module	2023-12-12	PWM of notes generation	Finished	2 Days	1 Day
Button_controller Module	2023-12-13	Buttons debounce	Finished	2 Days	2 Days
Seg_display Module	2023-12-15	Full 7Seg sisplay	Finished	2 Days	2 Days
Main_controller Module	2023-12-17	Display and PWM signal selector	Finished	2 Days	1 Day
Keyboard Module	2023-12-18	Free Mode and octaves support	Finished	2 Days	2 Days
Autoplay Module	2023-12-20	Auto Mode and flexible tempo	Finished	2 Days	2 Days
Learning Module	2023-12-22	Learning Mode and evaluation rule	Finished	1 Day	1 Day
Add Songs	2023-12-23	Choose songs and encode the song sheet for autoplay and learning	Finished	1 Day	1 Day
Document	2023-12-24	Compose project report (aka this README)	Pending	1 Day	1 Day
Overall Test	2023-12-25	Run the final test, record demo video and prepare for the presentation	Finished	1 Day	Day

Summary

The developemnt procedure is quite challenging and tough, but we finally get there. Hooray! Along the way, it's full of surprise and unexpected failure, furure cannot be predicted. But when we looked back, it's certain that you will get what you've written, and the board is operating the way you program it, but sometimes it's not what you expected. Therefore, we shall test every part, from bottom to the top, to verify that its behaviour looks what we'd like it to be. Also, it's important to apply the existing code, like the code that we were taught during lab session, since it's verified by the instrcutors and almost unlikely to corrupt. As for our product, we would say it's not perfect. We have some ideas about refactoring it and adding more feature to it.

Problems & Solution

- Bug cased by dismatch of bandwidth. Solved by the checking the warning message of synthesis.
- Unfamiliar used of behaviour and sequential logic, which causes some lag in testing.
- Improper design of codes in verilog, which may reduce performance (suggested by warning message).
- Song sheets are hard encoded. (Yet fixed, propose an issue if you have idea).

Improvement

- Some reg are oversized and it's suggested by the warning message of synthesis.
- Notes have fixed LEDs and 7 Segment message, and it's needless to write duplicate assignment inside the state machine.
- Implement the key adjustment feature by adding a module to remap the key to note.
- Implement the display of detailed score. Trapped by the design of rule and display.
- Implement the record of song played by user. Trapped by the load of develop and test.
- Implement the VGA display. Trapped by the lack of knowledge.

Project Topic Design

If we were to design the project topic, we'd like to propose a **Sokoban** game with VGA display.
The topic is based on the thoughts that

- VGA is an essential part of FPGA, also benefitial to the study of computer architecture
- It's challenging and requires complex design
- It requires mountainous submodules which can train the engineering skills
- Getting the game right will lead to a full mark, and extra features should not be counted too much.

The draft requirements would be

Task	Category	Detail
Report (20pt)	Document	Description anything that the users and contributors need to know, e.g., user manual principles and structure.
Basic Function (50pt)	Code	Display the map and character normally. When the user press button, the character should move and push the boxes correctly. The wall behaves correctly.
Standard Function (40pt)	Code	Support multiple maps. It would be great if levels are available.
Bonus (20pt)	Code	1. Multiple users with record 2. Background music and sound effect ...

Installation

1. At first, you should have a `EGO1` board in your hand.
2. Open `Vivado` and create a project. Then import the sources folder into the *design source* and constr folder into *constraints* in the project.
3. Finally, follow the steps in `Vivado`, which are *synthesis*, *implementation* and *generate bitstream*. You should get the bitstream file now.
4. Alternatively, you can use the bitstream provided, which locates at bit and release.

5. Once you get the bitstream, you can connect the board to your computer and click **program device** in the hardware manager panel.

Enjoy it!