**Hardware Design and Lab**

**Final Project**

**Slime Jump**

**Team 28**

**111060013 EECS 26' 劉祐廷**

**111060002 EECS 26' 李侑霖**

**Catalog**

**1. FPGA Question: Slot Machine………………………….……P3**

**2. FPGA Question: Chip2chip……………………....………….P8**

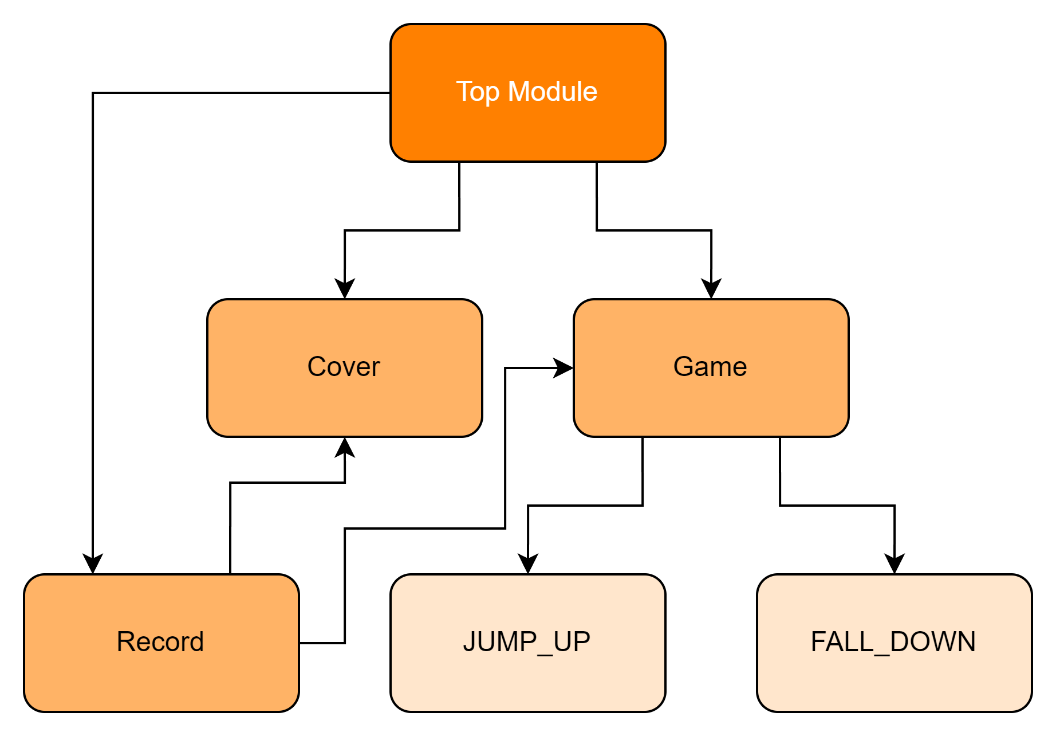
**3. FPGA Question: Car………………………………...…….P9**

**4. What We Have Learned…………………………………P12**

1. **遊戲介紹**

此遊戲靈感來自於手遊《Doodle Jump》，遊戲主角為Slime，玩家須透過鍵盤操控Slime左右移動，讓Slime能踏著floor慢慢往上跳，直到分數到達100分。floor位置是偽隨機生成的，並且跳每跳5次floor可以獲得一次二段跳的機會，由空白鍵觸發且不可疊加。遊戲畫面每下移一次即獲得1分，遊戲會記錄上一局的分數以及每場遊戲的最高分數，並顯示在cover畫面。

1. **遊戲架構**

****

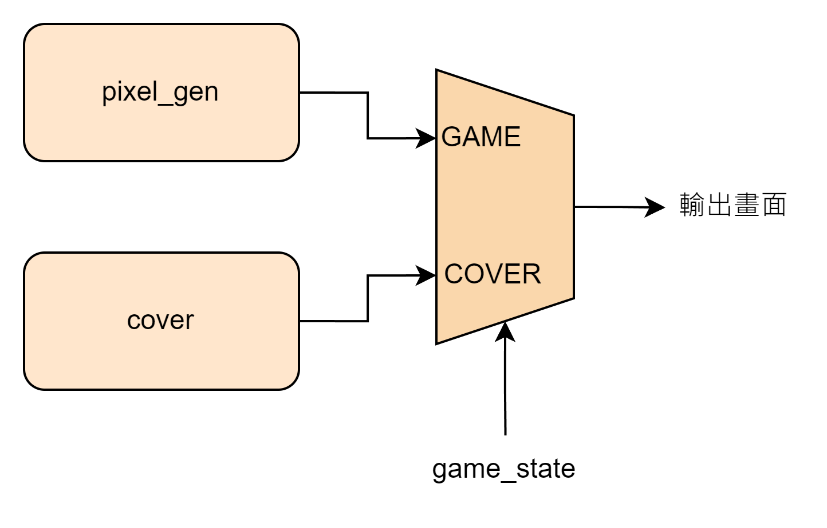
首先我們先將遊戲分為Cover與Game兩種不同的狀態，分別對應到目前應該顯示起始畫面或是目前應該顯示遊戲畫面。在遊戲畫面(Game)的狀態下，依照遊戲主角Slime當前的狀態分為JUMP\_UP與FALL\_DOWN兩種狀態，分別代表Slime目前處於向上跳躍或是向下墜落的狀態。

1. **Module**

|  |  |
| --- | --- |
| module | 功能 |
| Top | 負責場景的選擇(Game或是Cover)與鍵盤輸入的判定，並彙整所有module處理連接到終端設備上的輸入輸出。 |
| cover | 負責生成遊戲開始畫面(Cover)，將畫面各個pixel該顯示的資訊透過Top module接線至vga\_controller。 |
| pixel\_gen | 負責生成遊戲畫面(Game)，處理從slime\_move module與floor module傳進來的遊戲資訊，轉換成由遊戲畫面各個pixel的資訊，透過Top module接線至vga\_controller。 |
| slime\_move | 負責遊戲各種功能的實現，包括Slime的狀態、位置，遊戲分數，Slime與floor碰撞判定，將遊戲資訊處理好，透過Top module接線至pixel\_gen module生成畫面。 |
| floor | 負責生成floor的位置，透過Top module接線至slime\_move module與pixel\_gen module，已進行碰撞判定與生成floor的畫面。 |
| record | 用來記錄上一局遊戲分數與最高分數，在畫面的狀態從Game切到Cover時會去抓取遊戲內的分數，並判斷是否需要修改最高分數。 |
| clk\_div | 負責生成vga\_controller module需要使用的clock。 |
| clk\_vga | 負責生成slime\_move module需要使用的clock。 |
| clk\_floor | 負責生成floor module需要使用的clock。 |
| vga\_controller | 將處理好的pixel轉換成VGA的訊號輸出。 |
| KeyboardDecoder | 解析鍵盤的輸入訊號。 |
| audio | 負責生成Slime彈跳的音效。 |

1. **遊戲功能**

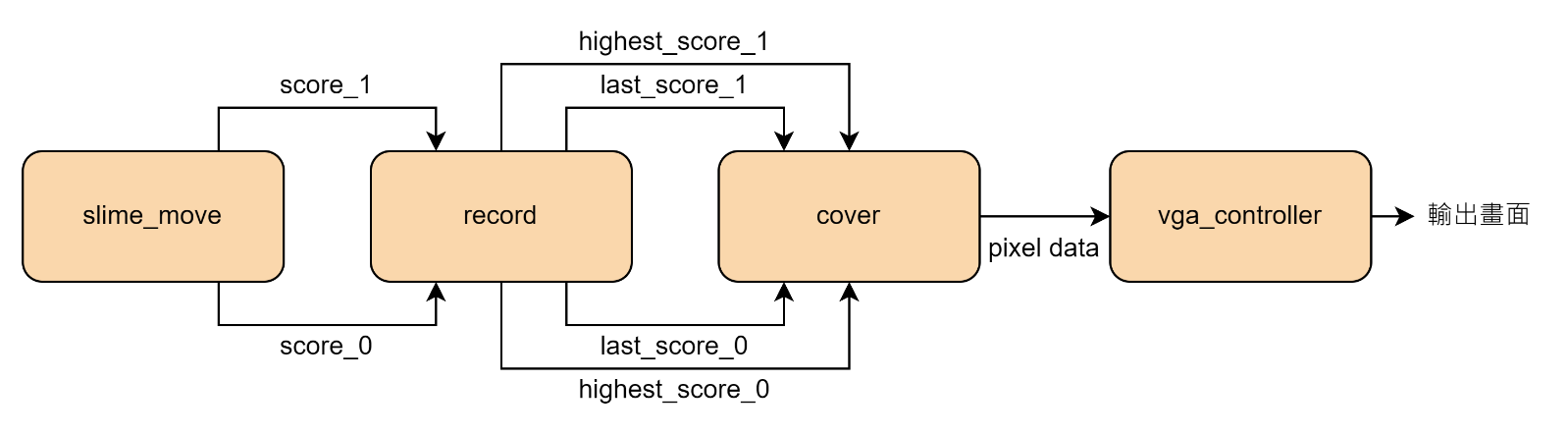
* **遊戲開始畫面與遊戲畫面的切換**

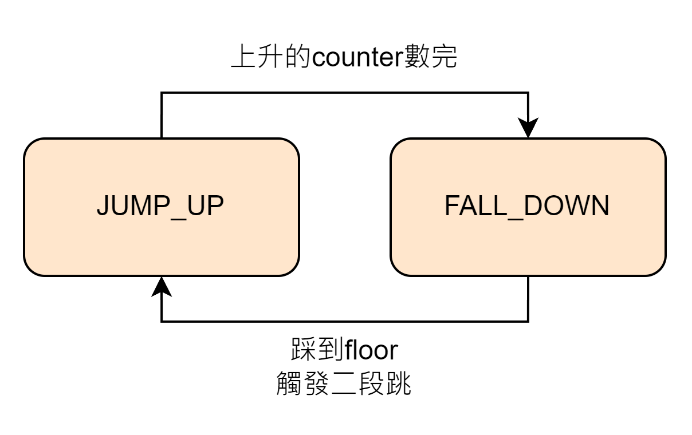


我們是透過用mux去選擇輸出畫面，來達成遊戲場景切換的功能，在遊戲開始畫面時按下Enter，game\_state會被設成GAME，進入遊戲畫面，而當Slime死亡後，game\_state會再被切回COVER，回到遊戲開始畫面。在測試的時候有遇到在遊戲開始畫面時碰觸Key A與Key D後，上局分數被改變的bug，後來發現原因是因為以我們的設計來說，遊戲畫面與起始畫面是同時運行的，因此在遊戲開始畫面時也能操控Slime移動，而去動到上一局的分數，後來我們夠過在鍵盤判定的地方多加了一個以game\_state當條件的mux，使Key A與Key D在遊戲開始畫面時失效以解決這個問題。

* **紀錄上局分數與最高分數**

上局分數與最高分數是由record module在game\_state從GAME切換到COVER時，去抓從slime\_move接出來的分數(score\_1, score\_0)，經過比較後處理成highest\_score\_1, highest\_score\_0, last\_score\_1, last\_score\_0並接入cover module，編號1的wire代表十位數，編號0的代表百位數，經由decoder將這四個數字轉換成對應的pixel資訊，再經由vga\_controller將畫面輸出。



* **Slime跳躍與落下**
  + **定義State**

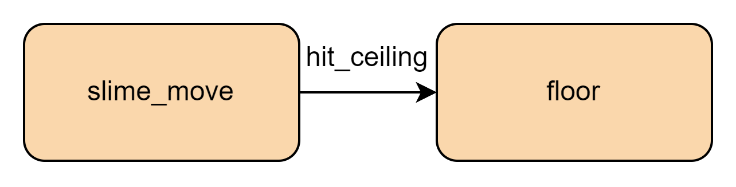
在slime\_move module內，我們依照Slime的狀況將遊戲分為兩個state，分別為JUMP\_UP與FALL\_DOWN，在JUMP\_UP時，Slime是可以穿越板子的，在FALL\_DOWN時才會判定使否有踩到板子。

* + **模擬重力加速度**

當Slime處於JUMP\_UP的狀態時，Slime會以一個初速度向上跳躍，並逐漸變慢，直到某固定高度時，state會變成FALL\_DOWN，此時Slime會向下加速移動，為了避免在下落時穿越floor的問題，我們採取的作法是將跳躍的時間分成三段，第一段時每1個clock cycle移動一個bit，第二段每4個clock cycle移動一個bit，第三段每8個clock cycle移動一個bit，以此盡可能的模擬出重力加速度的效果。

* + **視野畫面上移**

為了避免Slime向上跳出螢幕範圍的問題，我們設定當Slime落下時踩到的板子位於螢幕的上半部的話，要將slime\_move module接到floor module的hit\_ceiling設為1'b1直到Slime的狀態再次切回JUMP\_UP，當hit\_ceiling為1'b1時，floor module會按照Slime跳躍的運動方式將所有的floor向下移動。



* + **Slime二段跳**
* **Slime左右移動**
* **Slime與floor的碰撞判定**
* **floor生成與移動**

1. **遊戲畫面**