논리회로 및 설계 (월)

텀프로젝트 최종보고서

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

소속 : 정보컴퓨터공학부

학번 : 202155543, 202055553

이름 : 김태윤, 서준

1. **프로젝트 주제**

추억의 크롬 공룡 게임을 모티브로 하여 FPGA 보드에서 새로운 방식으로 게임을 구현하는 것을 목표로 합니다.

Text LCD, 키패드, 8 Array 7-Segment, PIEZO, Full Color LED 다양한 하드웨어 장치와의 상호작용을 통해 몰입감 있는 사용자 경험을 제공합니다.

Text LCD : 상태(대기, 진행, 게임 오버, 게임 클리어)와 블록 잔여량 표시

KeyPad : 게임 시작, 재시작 및 게임 도중 사용자의 동작 입력

PIEZO : 상황별 사운드 출력

Full Color LED : 상태별 색 변화 (초록 = 진행 및 클리어, 빨강 = 게임 오버)

7-Segment : 블럭들을 표현하여 실제 게임 진행 상황 출력

1. **게임 진행 및 시나리오**
2. **대기 화면 및 시작**

초기 상태에서는 Text LCD에서 대기 화면이 표시됩니다. 이때, 사용자는 키패드의 \* 키를 눌러서 게임을 시작할 수 있습니다. 게임 시작과 동시에, PIEZO를 통해 간단한 스타트 사운드가 재생될 수 있습니다.

1. 게임 진행 및 블록 피하기

게임이 시작되면, Text LCD의 두 번째 줄에 ‘남은 블록 수 (Block Remain)’가 표시됩니다. 남은 블록 수는 1초 간격으로 10 -> 9 -> 8 … 식으로 감소하여, 이는 FPGA 내부 로직에 의해 자동으로 카운팅됩니다. 사용자는 주어진 키패드를 통해 충돌을 피하며 게임을 진행합니다. 이때, Full Color LED는 초록색으로 점등하여 ‘진행 중’임을 시각적으로 표현합니다. 추가적으로 게임 상황에 맞춰 PIEZO를 제어하여, 긴장감 있거나 경쾌한 효과음을 낼 수 있도록 구현하였습니다.

1. 게임 오버

만약 블록끼리 충돌한다면 게임 오버 화면이 나타납니다. Text LCD에 ‘GAME OVER”가 표시되며, LED는 빨간색으로 점등되어 실패를 시각적으로 알립니다. 동시에 PIEZO를 통해 게임 오버 사운드를 재생하였습니다. 이후, 사용자는 \* 키를 눌러 게임을 재시작할 수 있습니다.

1. 게임 클리어

블록을 모두 피한다면 “Celar! Congratz!”가 LCD에 표시됩니다. LED는 초록색으로 바뀌고 PIEZO로 클리어했다는 사운드가 재생됩니다.

1. **주요 과제 수행**

* 서준

**1. 게임 로직(Game Logic)**

**1.1 전체적인 구조**

**1) 상태 전환(State Machine)**

• \*\*플립플롭(Flip-Flop)\*\*과 \*\*조합논리(Combinational Logic)\*\*를 통해, 게임을 여러 상태(예: 초기화 Init → 대기 Idle → 플레이 Play → 종료 End)로 구분하고 전환합니다.

• 이때, **Trigger** 신호(예: 버튼 눌림, 시간 초과 등)가 들어오면 해당 신호를 토대로 상태 전환 논리를 결정합니다.

• 예: Trigger = 1일 때 Idle → Play, Trigger = 0이 일정 시간 유지되면 Play → End 등.

• 소프트웨어적으로는 switch-case 문이나 if-else 블록으로 구현하지만, 하드웨어(논리회로) 관점에서는 플립플롭 입력에 **조합논리**(AND, OR, NOT 등)를 연결하고, 이를 통해 다음 상태를 결정합니다.

**2) 업데이트(메인 루프)**

게임 로직을 반복적으로 수행하는 핵심 루프입니다. 각 반복(또는 클록 주기)마다 다음을 처리합니다.

1. **입력 처리**

• 스위치, 버튼, **피에조 센서** 등으로부터 입력을 받아 현재 상태에 맞게 해석합니다.

• 여기서 **KEY → agd(캐릭터 좌표 이동용)** 부분이 중요한 역할을 합니다.

• KEY 입력(좌우 이동, 점프 등)을 받아서 캐릭터가 위치한 좌표(agd)에 전달해주는 구조입니다.

• 하드웨어적으로는 이 KEY 신호를 디지털화한 뒤, **MUX**를 통해 캐릭터 좌표(agd) 업데이트 경로를 선택하거나, **Shift Register**를 통해 일정 방향으로 좌표를 변화시키는 방식으로 구현할 수 있습니다.

2. **로직 업데이트**

• 캐릭터 이동, 점수 계산, 충돌 체크(아래 소개할 Obstacle 배열 기반) 등을 수행합니다.

• 장애물(Obstacle)을 정의한 배열을 두고, 이를 **Shift Register**나 **8-Counter**와 연동하여 게임 스테이지를 스크롤(왼쪽에서 오른쪽, 혹은 일정 방향)시키는 방식으로 구현할 수 있습니다.

• 예: 8-Counter로 일정 시간마다 카운트를 증가시키고, 그 값에 따라 Obstacle 배열의 인덱스를 이동(Shift)시켜 화면상 장애물 위치가 바뀌도록 구현.

• Shift Register를 사용해도 유사하게, 일정 클록마다 배열의 데이터를 한 칸씩 이동하면서 장애물이 움직이는 것처럼 보이게 할 수 있습니다.

3. **렌더/출력**

• LED, 7세그먼트, LCD, 혹은 LED 매트릭스 등에 현재 상태(점수, 캐릭터 위치 등)를 표시합니다.

• **Decoder**(디코더)를 사용해 여러 라인을 선택하거나, 7세그먼트를 구동할 때 2진 → 7세그 신호로 변환해줄 수 있습니다.

• MUX도 출력단에서 여러 신호를 선택·출력하는 데 활용될 수 있습니다(예: 디버깅 정보 vs 실제 게임 정보 등).

**1.2 각 요소별 상세 사용 사례**

1. **Mux(멀티플렉서)**

• 입력 선택용으로 사용합니다.

• 예: 캐릭터 좌표(agd)에 전달되는 값이 “KEY 입력에 따른 좌표 변화”인지, “기본 위치” 혹은 “게임 오버 상태에서의 초기화 좌표”인지 등을 결정할 때, Mux를 통해 여러 신호 중 하나를 선택해 agd에 공급할 수 있습니다.

2. **Trigger(트리거 신호)**

• 버튼을 누르는 순간 발생하는 신호나, 특정 조건(시간 초과, 점수 달성 등)이 만족되었을 때 생성되는 이벤트 신호를 말합니다.

• 이 신호가 1이 되면 상태 머신에서 Idle → Play 등으로 전환하도록 설계합니다.

3. **KEY → agd(캐릭터 좌표)**

• 사용자가 누른 KEY를 기반으로 캐릭터 좌표(agd)를 업데이트합니다.

• KEY 값(예: 왼쪽 이동, 오른쪽 이동, 점프 등)을 해석하여 agd.x++, agd.x–, agd.y++ 등의 연산을 수행합니다.

• 하드웨어적으로는 KEY 입력이 레지스터나 조합논리로 전달되어, “현재 좌표 + 1” 혹은 “현재 좌표 - 1” 등의 결과가 Mux를 통해 선택된 뒤 플립플롭(좌표 레지스터)에 저장됩니다.

4. **Decoder(디코더)**

• 디스플레이 단에서 2진 값을 7세그먼트나 LED 열/행 선택 신호로 변환할 때 사용합니다.

• 또는 현재 상태(Init, Idle, Play, End 등)에 따라 LED 점등 패턴을 달리하기 위해, 상태 레지스터의 2진 출력을 디코딩하는 방식으로도 활용할 수 있습니다.

5. **Obstacle 배열 + Shift Register**

• 게임 맵의 장애물 정보를 1차원/2차원 배열로 가지고 있다가, 일정 속도로 스크롤시키는 방식으로 구현합니다.

• **Shift Register**를 사용해 매 클록마다 배열의 데이터를 한 칸씩 이동시키거나, 또는 인덱스만 이동시켜서 위치가 변하는 것처럼 표현합니다.

• 예: obstacle[0..7]에 각 비트 패턴(장애물의 높이, 모양 등)을 저장해두고, Shift Register에 연결해 매 클록마다 왼쪽/오른쪽으로 이동.

6. **8-Counter**

• 8비트 카운터 혹은 3비트 카운터(0~7) 등을 사용하여 정해진 주기에 맞춰 카운트값을 증가시킵니다.

• 이 카운트값을 이용해 **Obstacle 배열**의 인덱스를 변경하거나, **게임 속도**를 제어할 수 있습니다.

• 예: 카운트가 8이 될 때마다 장애물을 한 칸씩 이동시키거나, 일정 카운트 이상에서 Trigger 신호를 발생시키도록 할 수도 있습니다.

**1.3** **구현 흐름 예시**

1. **초기화(Init State)**

• 8-Counter를 리셋하고, Shift Register에 장애물 패턴을 초기값으로 로드합니다.

• 캐릭터 좌표(agd)는 Mux를 통해 “초기 위치”를 선택하도록 설정합니다.

• 디스플레이/LED는 Decoder를 통해 ‘READY’ 또는 초기 점수를 표시.

2. **대기(Idle State)**

• Trigger(버튼, 피에조, KEY 입력 등)를 감지할 때까지 대기.

• Trigger가 1이 되면(예: KEY를 눌러 ‘시작’ 신호 발생) 상태 머신이 다음 상태(Play)로 전환.

3. **플레이(Play State)**

• **KEY → agd** 로 캐릭터 좌표가 변경되며, 장애물(Obstacle 배열)은 **8-Counter**와 **Shift Register**로 일정 주기마다 스크롤됩니다.

• 충돌 체크, 점수 업데이트 등의 로직 수행. (충돌 로직은 bounding box 혹은 특정 인덱스 비교로 처리)

• 만약 특정 조건(충돌 발생, 시간 초과 등)이 만족되면 Trigger가 발생하여 End 상태로 전환.

4. **종료(End State)**

• 최종 점수 표시, LED 점멸 등.

• 다시 한 번 KEY 입력(Trigger)이 들어오면 Init으로 돌아가 재시작할 수 있도록 상태 전환 로직을 구성.

**2. 충돌(Collision) 로직**

**2.1 충돌의 정의 및 구현**

Avoid It게임에서 캐릭터와 장애물 간 충돌이란 COM1 영역에서 상단 / 중단 / 하단의 값이 겹치는 지를 기준으로 정의했습니다. 예시를 들면 캐릭터가 하단에 있다가 장애물이 하단으로 오는 것을 피하지 못한 경우, 하단에 장애물이 날라왔고 중단에 있던 캐릭터가 장애물이 있는 타이밍에 아래로 내려가는 등의 행동을 했을 때를 의미합니다.

실제 회로에서 장애물과 캐릭터의 좌표는 각각의 레지스터에서 비교 연산(비교기, Comparator)을 수행하고, AND 게이트 등으로 조합하여 “충돌 여부”를 뽑아냅니다.

**2.2 충돌 후 처리**

충돌이 감지되면 게임 오버 시그널을 발생시킵니다. 회로적으로는 충돌 신호(Collision Signal)가 1이 되면, 상태 레지스터(State FF)가 ‘End’ 상태로 전환되도록 제어 회로(Combinational Logic)에서 설계했습니다.

이 플래그 정보는 game\_event와 연계되어 LED, LCD, 피에조 출력을 유발하여 충돌 여부를 유저가 알 수 있게 합니다.

**3. 피에조(Piezo) 활용**

피에조 센서는 압력이나 진동을 전기적 신호로 변환하는 소자로, 프로젝트에서 게임의 박진감을 더해주거나 **이벤트를 알리는** 등으로 사용할 수 있습니다.

**3.1 피에조 기본 원리**

• 충격, 진동, 압력 등이 가해지면 피에조 센서 양 끝단에 전압이 발생.

• 발생 전압은 매우 짧고, 크기도 불규칙적이므로 **회로적 안정화**(필터링)가 필요.

• 일반적으로 \*\*커패시터(Capacitor)\*\*와 **저항(Resistor)**, 혹은 **연산 증폭기(Op-Amp)** 등을 통해 원하는 범위의 신호로 정규화(Conditioning).

**3.2 디지털 신호 변환(피에조→MCU/논리회로)**

**1Khz를 Clk으로 사용했고 이에 맞게 분주를 넣어서 적당한 음역대를 나누었습니다. 게임의 상황에 따라 음역대를 오가면서 유저로 하여금 게임 몰입도를 증가시켰습니다.**

* 김태윤

1. **Text LCD 구현 과정**

우선Text LCD를 구동하기 위해 내부 명령어(Function Set, Display On/Off, Entry Mode 등)를 숙지하고, 이를 순차적으로 전송하는 초기화 FSM을 설계하였습니다. 이 과정에서는 1kHz 클록을 기준으로 약 1ms 간격으로 명령을 보내도록 구성하였으며, Verilog 상에서 ST\_INIT1, ST\_INIT2, ST\_INIT3 등의 상태를 정의하여 단계별로 LCD 레지스터를 세팅하도록 했습니다.

TEXT LCD에 문자열(예: “PLAYING…”, “GAME OVER” 등)을 표시하기 위해, 각 ASCII 코드를 한 글자씩 전송하는 로직을 추가하였습니다. 이를 위해 ST\_LINE1, ST\_LINE2 상태를 두고, 내부 카운터(cnt)를 이용해 원하는 문자열 길이만큼 반복 출력하였습니다. 블록 수와 같은 동적 정보는 Verilog에서 실시간 연산을 통해 숫자를 ASCII로 변환(예: digit + 8'd48)하고, 이를 LCD 데이터로 전송하는 방식을 사용하였습니다.

게임 진행 중 남은 블록 수(block\_remain)가 매초 감소할 때 LCD도 즉시 업데이트되어야 합니다. 초기에는 S\_CONTINUE 상태에서 상태가 변하지 않는 한 LCD가 재출력되지 않아 숫자가 바뀌지 않는 문제가 발생하였습니다. 이를 해결하기 위해, ST\_IDLE 상태에서 1초마다(cnt == 999) 다시 ST\_LINE1으로 돌아가 화면을 재출력하도록 FSM을 보완하였습니다. 이로써 “Block Remain : 10 → 9 → 8 …” 등의 수치 변동이 자연스럽게 LCD에 반영될 수 있었습니다.

초기화 루틴이 예상대로 동작하는지, 그리고 1초 주기로 줄어드는 블록 수가 LCD에 원활하게 표시되는지를 실제 보드와 Waveform 분석을 통해 확인하였습니다. 결과적으로 게임 상태(대기, 진행, 오버, 클리어)마다 해당 메시지를 올바르게 표시할 수 있었으며, 남은 블록 수가 1초 간격으로 정확히 감소·출력되는 것을 확인하였습니다.

1. **Full color led 구현**

보드에 장착된 풀컬러 LED는 R/G/B 세 채널을 개별적으로 제어해야 합니다. 본 프로젝트에서는 Red와 Green 신호만 주로 사용하여 게임 상황을 직관적으로 표시하고, Blue 채널은 향후 확장 가능성을 고려해 .xdc 파일에 매핑만 하였습니다. Verilog로 작성된 top 모듈에서 led\_red와 led\_green 신호를 4개의 풀컬러 LED(R/G 핀)에 동일하게 할당하여, 하나의 제어 신호만으로 모든 LED가 같은 색을 띠도록 구성하였습니다.

게임 FSM(game\_event 모듈)에서 상태가 바뀔 때마다 led\_red 또는 led\_green을 활성화하여 LED 색을 전환하도록 설계하였습니다. 구체적으로, 대기 상태(S\_MAIN)와 게임 오버(S\_OVER) 시에는 빨간색(led\_red=1)을, 진행(S\_CONTINUE) 및 클리어(S\_CLEAR) 시에는 초록색(led\_green=1)을 켜도록 했습니다. 이를 통해 사용자에게 게임의 진행 상황을 단번에 전달할 수 있게 되었습니다.

초기 테스트에서 LED가 점등되지 않는 오류가 나타난 경우, .xdc 파일의 핀 배치와 IOSTANDARD(LVCMOS33) 설정을 재확인하여 문제를 해결하였습니다. 이후에는 정상적으로 LED가 상태 변화에 맞춰 빨간색 또는 초록색으로 전환되었고, 필요에 따라 Blue 채널을 활용해 추가 색상을 낼 수 있는 확장성을 확보하였습니다.

빨강, 초록만으로도 게임 진행 여부와 성공, 실패를 직관적으로 전달하기에 충분했으며, 상황에 따라 깜빡이거나 밝기를 조절(PWM)하는 방식으로도 재미 요소를 추가할 수 있습니다. 또한, 파란색 채널을 조합하면 더 다양한 색상 표현도 가능하므로, 추후 LED를 활용한 비주얼 이펙트 확장에 용이하였습니다.

1. **Schematic 일부분**

텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명[[1]](#footnote-1)

도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

자동 생성된 설명[[2]](#footnote-2)

1. 최상위 모듈 avoid\_it 스케마틱 [↑](#footnote-ref-1)
2. Game\_instance의 전체 스케마틱, 주로 mux와 shift\_register가 기능한다. [↑](#footnote-ref-2)