ZynqLab06

학과	전기정보공학과
학번	20101418
이름	장민진
과목명	SoC 설계 입문
제출일자	2024/11/12

목차

I. 개요 및 설계 구조

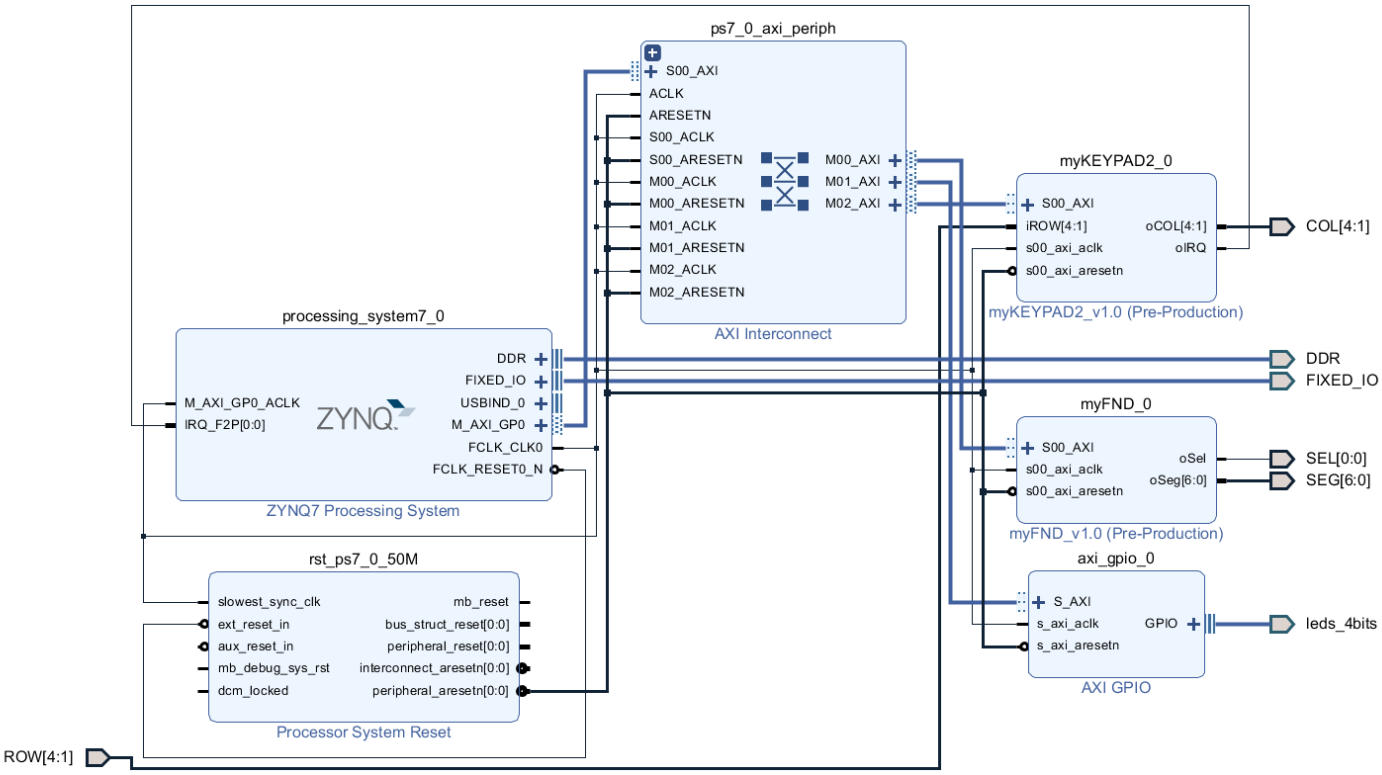


Figure 1 . Block Design

Figure 1은 최종 Block Design으로, 사진과 같이 GPIO LEDs_4bits와 Custom IP로 구성한 FND와 Keypad를 추가하였다.

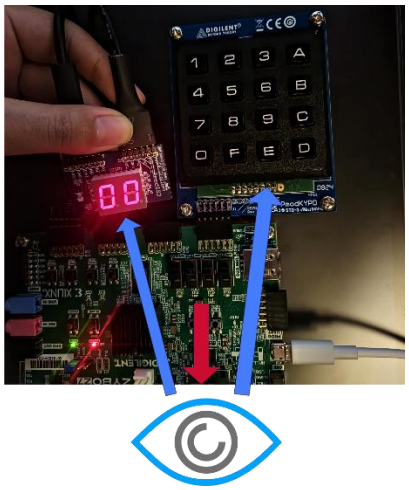


Figure 2 . 관찰 방향

Figure 2에서 볼 수 있듯이 Pmod Keypad와 4개의 LED 방향, Seven Segment를 한 방향에서 보기 쉽게 하기 위해서, SDK에서 GPIO 블록에 Write 하기 전 oKEYNUM의 비트를 반전한 값을 LED에 Write하여 출력하도록 설계하였다.

Keypad 설계

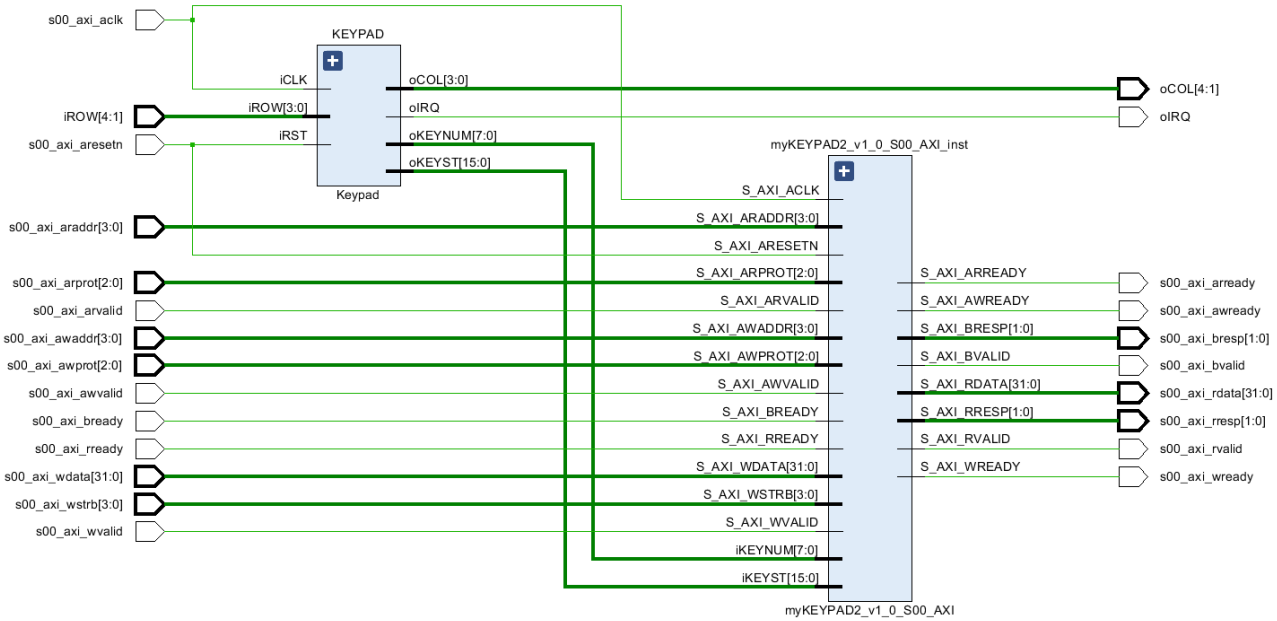


Figure 3 . myKeypad2 IP

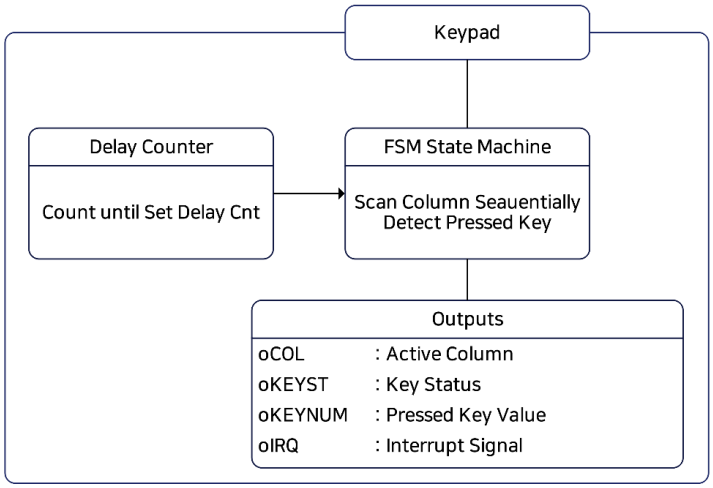


Figure 4 . Keypad Structure

Custom IP myKEYPAD2는 oCOL[4:1]을 FSM 구조에 따라 순차적으로 출력하고 iROW[4:1]을 입력으로 받아 눌린 키를 감지하여 oKEYST와 oKEYNUM으로 출력하게 된다.

Figure 3에서 볼 수 있듯이 KEYPAD 모듈에서 oKEYNUM과 oKEYST가 출력되어 myKEYPAD2의 입력으로 연결되며, 내부 레지스터 slv_reg0에 SW 검증을 위한 oKEYST가 저장되고, slv_reg1에는 oKEYNUM이 저장된다.

Interrupt 신호인 oIRQ는 키패드가 눌렸을 때(iROW != 4'b1111) High로 활성화된다.

Delay Counter 로직은 parameter로 선언한 DELAY 변수에 값을 설정하여 열이 전환되는 속도를 조절할 수 있다.

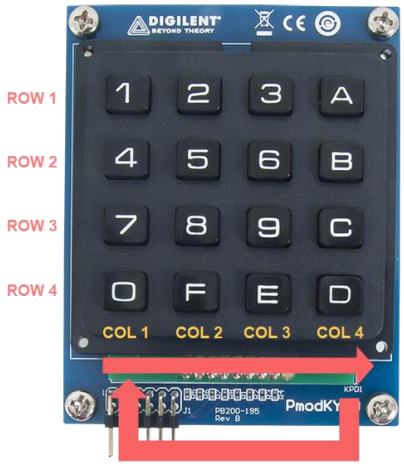


Figure 5 . Keypad 구조

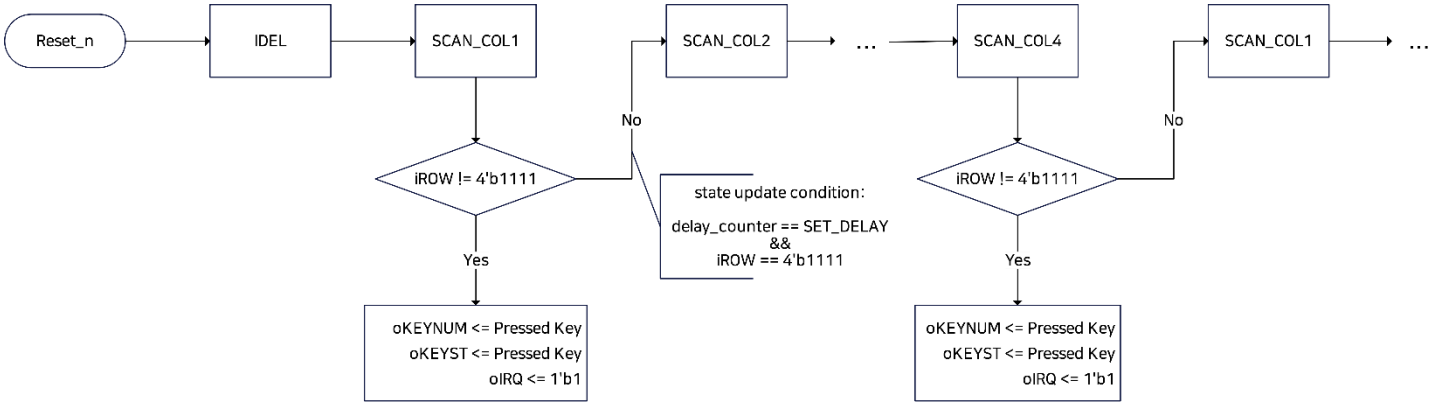


Figure 6 . Keypad FSM State Diagram

Figure 6는 KEYPAD 모듈의 내부 FSM State Diagram을 나타낸 블록 선도이다.

Reset_n으로 리셋 신호를 받으면 FSM은 IDLE 상태로 초기화되며, 내부에서 사용하는 레지스터들을 초기화하는 작업을 마친 후 next state를 SCAN_COL1로 전환한다.

SCAN_COL1에서는 oCOL = 4'b1110을 출력하여 첫 번째 열을 활성화하여 iROW를 읽어오고,
특정 키가 눌려서 iROW가 4'b1111이 아니라면, 해당하는 키의 값(1, 4, 7, 0 중 눌린 키 값)을 읽어오게 된다.

그리고 키가 눌린 경우에는 oIRQ를 High로 활성화하게 된다.

여기서 current state를 next state로 업데이트 시키는 조건은 iROW != 4'b1111 이면서 (아무 키가 눌리지 않은 상태)
설정한 Delay Count를 충족했을 때이다. 이를 통해, 키를 눌렀다 뗄 때까지 state를 전환하지 않음으로써
키가 눌린 도중에는 state가 변경되어 키를 잘못 인식하는 경우를 방지할 수 있도록 FSM을 설계하였다.

따라서 위 다이어그램과 같이 SCAN_COL4에서 다시 SCAN_COL1로 반복되며, 열을 순차적으로 활성화하며 행을 읽어와 눌린 키가 무엇인지 Decoding 하는 과정을 수행한다.

Debouncing 로직

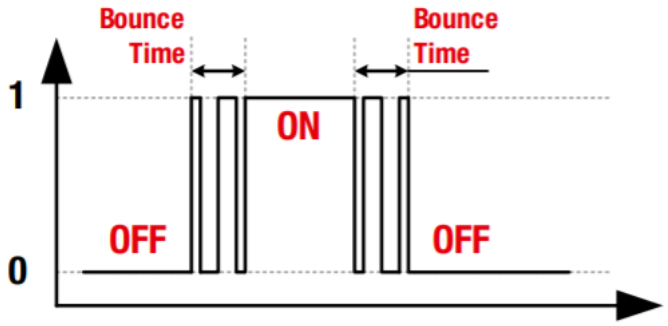


Figure 7 . 바운싱 현상 (출처: Digikey TechForum)

SDK에서 보드 검증을 통해 인터럽트를 수행하는 과정에서 바운싱 현상이 발생하는 것을 확인하였다.

바운싱(bouncing)은 스위치의 상태가 변하는 순간 기계적인 진동에 의해서 매우 짧은 시간 내에 접점이 붙었다 떨어지는 것을 반복하는 현상이다. SDK에서 특정 키들을 눌렀다 떼는 순간에 주로 발생하였다.

과제의 중간 결과물 중 SW에서 처리할 때는 값이 이전과 같다면 출력하지 않게 했던 SDK 코드가 있었으나,
값을 비교하여 처리하게 된다면, 같은 키를 손으로 입력하고자 하여도 입력되지 않게 된다.

예를 들어 1번 키를 두 번 누르고자 하였을 때, 첫 번째 출력만 가능하고 두 번째 출력이 불가능하다.

이를 해결하기 위해 Keypad의 HW 로직에 debounce_counter를 추가하였다.

사람이 손으로 누르는 것이 아닌 순간적인 바운싱에 의한 출력을 필터링하기 위해
카운터를 통해서 너무 짧은 시간으로 활성화되는 oIRQ는 출력하지 않도록 하는 로직을 추가하였다.

그 결과 SDK에서 확인할 수 있었던 바운싱으로 인한 다수의 인터럽트 출력 문제를 성공적으로 해결하였다.

II. Simulation Waveform

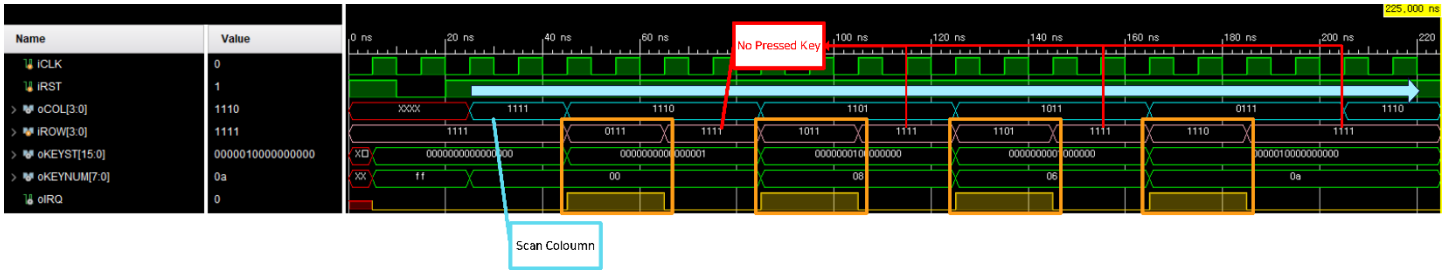


Figure 8 . Behavioral Simulation

Figure 8은 Behavioral Simulation의 결과로, 목표 설계 조건대로 동작하는 모습을 확인할 수 있다.

시뮬레이션 파형을 편리하게 확인하기 위해 DELAY 변수는 Testbench에서 작은 값으로 설정하여 진행하였다.

빨간색으로 표시한 파형을 확인해보면, Row가 1111으로 입력되고 있는 경우는 아무 키도 눌리지 않은 상태로, oIRQ가 Low로 나타나는 모습을 확인할 수 있다.

그리고 하늘색으로 표시한 oCOL은 최초 IDLE 상태인 1111에서 시작하여 SCAN_COL1~SCAN_COL4의 4가지 state를 순차적으로 반복하여 하나의 열씩 성공적으로 활성화하고 있음을 확인할 수 있다.

또한 주황색으로 나타난 키가 눌리고 있는 상황에서는 oIRQ가 활성화되고,
State가 업데이트되지 않는 모습을 확인할 수 있다.

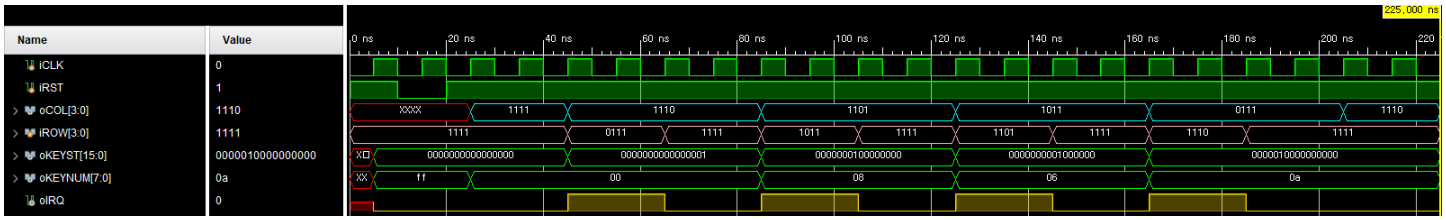


Figure 9 . Post-Implementation Functional Simulation

Post-Implementation Simulation 또한 Behavioral Simulation에서 수행했던 바와 같이 동작함을 확인하였다.
따라서 HW에서도 원활히 동작할 것이라고 판단하였다.

III. Board Test 결과

SDK 결과

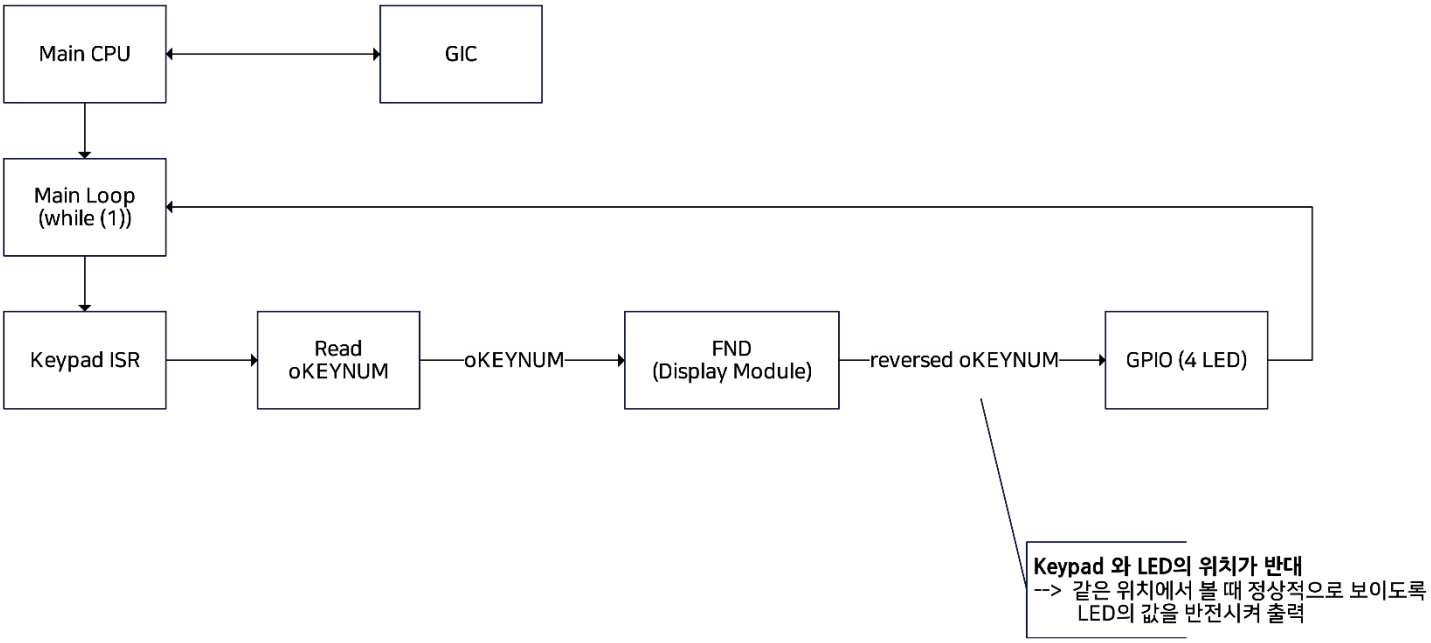


Figure 10 . SDK 동작 구조

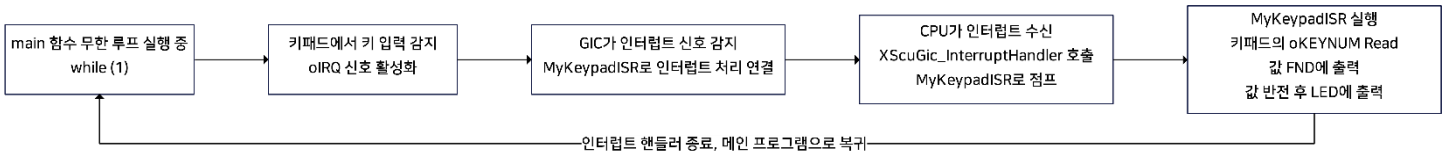


Figure 11 . Interrupt 동작 구조

먼저 BaseAddr + 4 위치에 저장되어 있는 oKEYNUM 값(HW)을 포인터로 선언한다.

main 함수에서 while(1)로 무한 루프를 실행하여, 실행을 유지한다.

그 과정에서 키패드에서 키 입력이 감지되면, oIRQ 신호가 활성화되며 Keypad ISR를 활성화된다.

Keypad ISR에서는 oKEYNUM 값을 Read하여 FND에서 출력하게 된다.

개요에서 말했듯이 LED는 관찰자의 입장에서 나머지 두 모듈과 반대이기 때문에

oKEYNUM 비트를 반전한 값을 LED에 출력한다.

또한 동적구동 단계의 SDK에서는 같은 키를 두 번 누르는 것은 감지하지 못하지만,

HW로 인터럽트를 구현한 최종 코드에서는 같은 키를 연속으로 눌렀을 때도 감지할 수 있도록 설계하였다.

Figure 12 . 동적구동 SDK

과제 Keypad 동적 구동의 SDK 결과물은 Figure 12과 같다

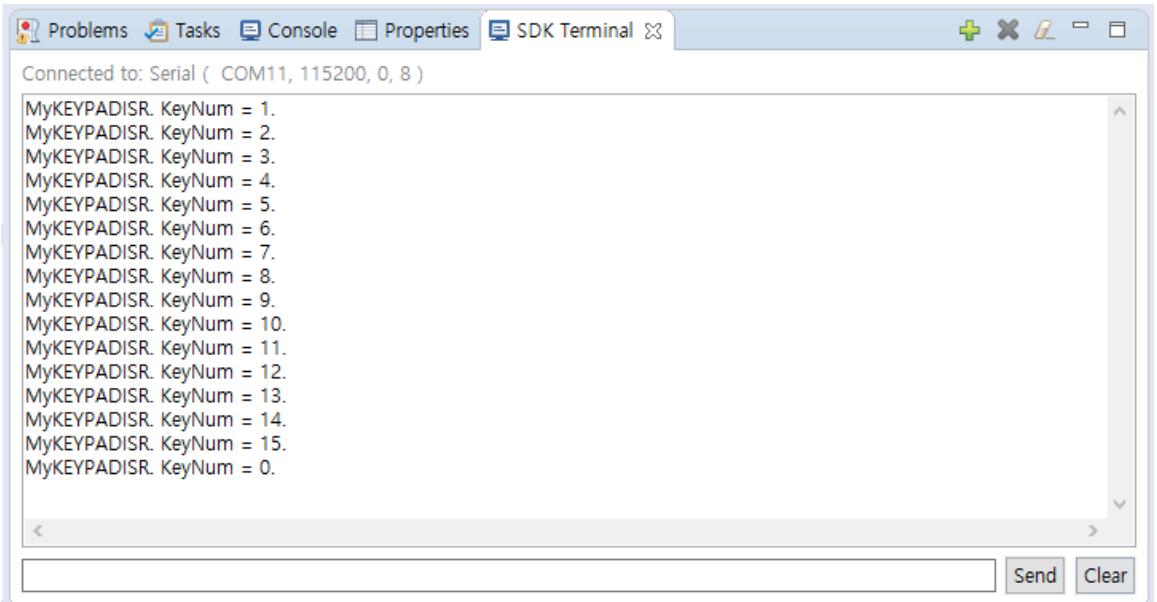
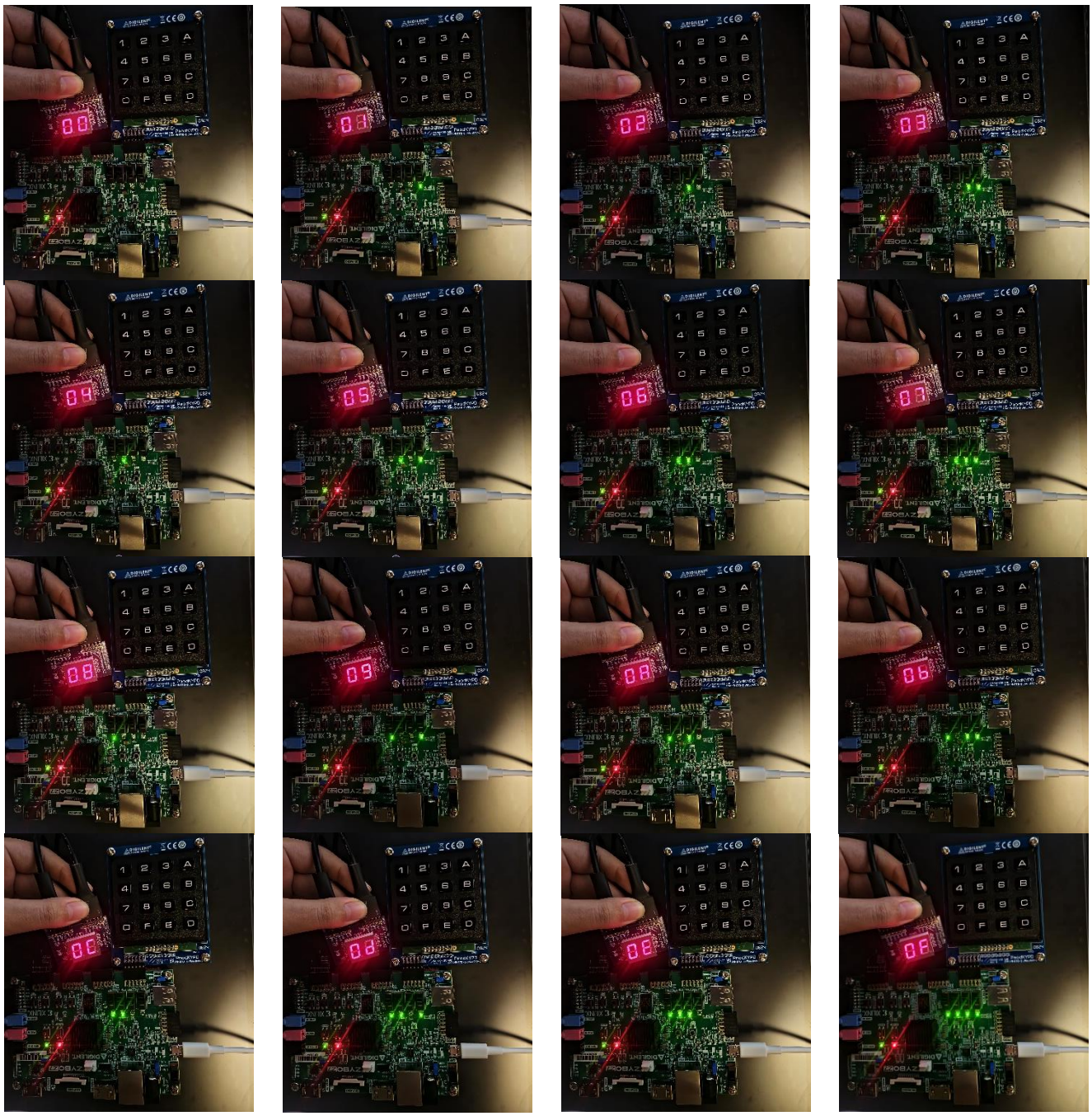


Figure 13 . SDK Terminal 출력 결과 (Interrupt)

HW의 출력 결과인 oKEYNUM이 정상적으로 출력되는 모습을 관찰할 수 있다.

최종 Board 구동 사진



Board 테스트에서도 FND와 LED가 누른 값과 일치하게 출력되고 있는 모습을 확인할 수 있다.

FND는 이전 과제에서 사용했던 IP를 그대로 사용하였다. (Hex 출력 mode 사용)

IV. 결론

이번 실험을 통해, 인터럽트의 장점을 체험해 볼 수 있었다.
main 함수에서 Delay를 기다리는 과정에서 출력 결과에 대한 응답도 느려졌던 지난 SDK 실험 환경들에 비해
즉각적으로 빠른 응답을 받을 수 있다는 장점을 직접 활용해볼 수 있는 좋은 경험을 얻을 수 있었다.
Custom IP인 Keypad에서 인터럽트 신호의 발생 여부를 결정하는 기능을 추가하여 인터럽트를 enable하고,
GIC를 통하여 인터럽트 신호의 트리거 모드와 priority를 설정하고 적용하여
최종적으로 CPU 코어가 인터럽트 신호를 전달받아 해당 인터럽트 처리 함수를 실행하는 SW 코드를 경험해 볼 수 있었다.
또한, 지난 과제에서 제작했던 Custom IP를 그대로 Re-use하여 기능 추가를 빠르고 안전하게 수행할 수 있었다.
IP를 Re-use 하여 설계 시간을 단축시키는 것이 어떠한 의미인지 직접 체감해볼 수 있었다.
Keypad의 HW를 구현하는 과정에서 Reference Manual을 참조하여 목표하는 동작대로 구동되도록
설계하는 과정도 좋은 경험이었다고 생각한다.
마지막으로, 스위치의 바운싱 현상을 하드웨어적으로 디바운싱하여 해결한 것도 성공적으로 마무리할 수 있었다.