Techniki Mikroprocesorowe

Projekt 1 - Picoblaze
Edytor tekstu na wyświetlaczu LCD

Wydział Fizyki i Informatyki Stosowanej

Adrian Wysocki
nr indeksu: 290538

16 grudnia 2020

1 Wstęp

Celem tego projektu było napisanie programu na mikroprocesor PicoBlaze, który ma działać jako edytor tekstu na wyświetlaczu LDC (wyświetlacz wbudowany w FPGA, na którym 'zaimplementowany' jest procesor). Ponieważ projekt realizowałem samodzielnie, początkowy zakres wymagań został zredukowany z uzgodnieniem prowadzącego zajęcia [3]. Lista wszystkich wymagań projektowych znajduje się w sekcji 'Założenia projektowe'.

2 Założenia projektowe

- klawiatura podłączona przez port PS/2
- podstawowa obsługa przycisków - litery, liczby, podstawowe znaki
- przejście do drugiego wiersza, po zapisaniu ostatniego znaku pierwszego wiersza
- obsługa strzałek - przemieszczanie kursora
- obsługa przycisku **enter** - przejście kursora do drugiego wiersza wyświetlacza lcd
- obsługa przycisku **backspace** - usunięcie znaku znajdującego się przed kursorem

3 Sprzęt

- klawiatura z zestawem instrukcji 2 [1]
- płytka FPGA Spartan-3A/3AN Starter Kit [2] z wgranym bitstreamem mikroprocesora PicoBlaze

4 Dokumentacja

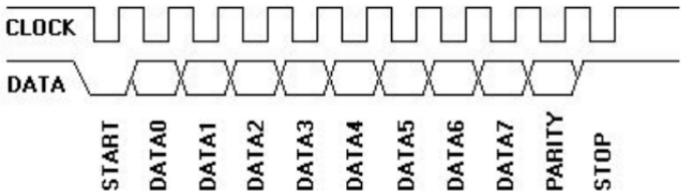
4.1 Architektura

4.1.1 Obsługa interfejsu PS/2

Informacje od klawiatury otrzymywane są poprzez interfejs PS/2. Złącze tego interfejsu posiada 3 linie:

- zegar
- dane
- uziemienie

Interfejs obsługiwany jest poprzez przerwania od opadającego zbocza zegarowego. Po każdym wywołaniu przerwania zapisywany jest kolejny bit danych przychodzących z klawiatury i zapisywany jest on do pamięci RAM. Gdy cała ramka danych zostanie odebrana (znak lub jego część), otrzymany bajt danych zostaje wystawiony w odpowiednie miejsce pamięci RAM, co jest sygnałem dla wątku głównego procesora, że kolejny znak jest gotowy do obsłużenia.



Rysunek 1: Ramka danych PS/2

4.1.2 Obsługa otrzymanych znaków

Wątek główny procesora zajmuje się obsługą otrzymanych od klawiatury bajtów danych znaków - tych, które wystawione zostaną do obsługi przez funkcję obsługującą przerwania od klawiatury.

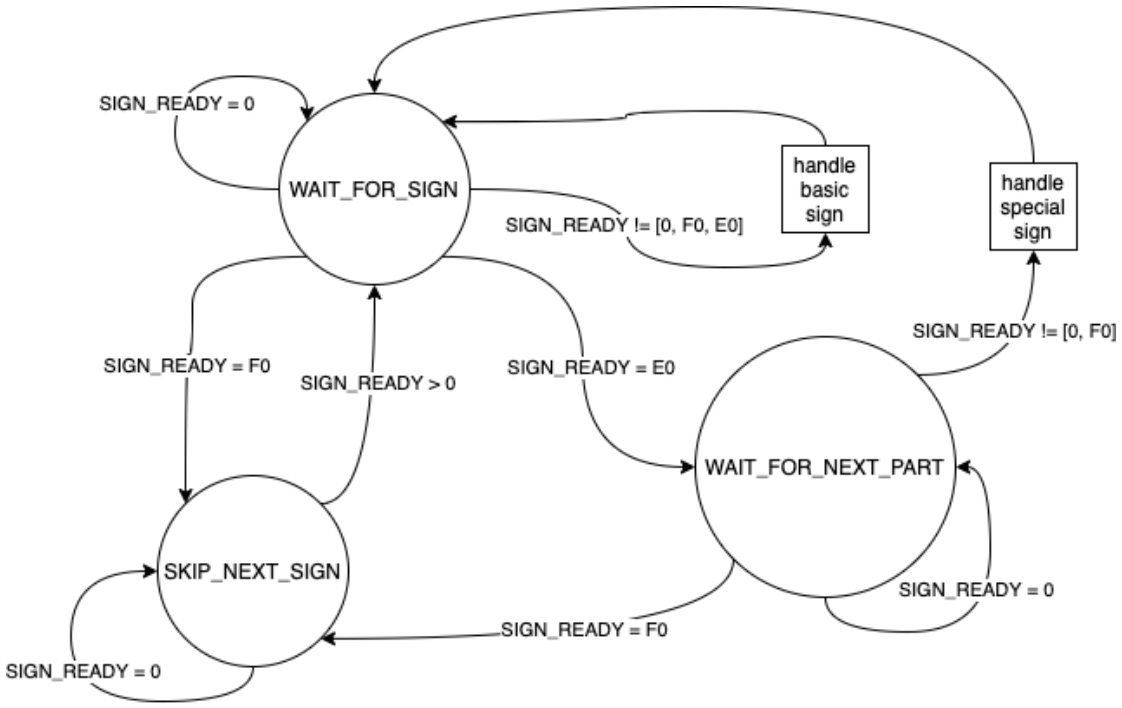
Spoglądając na zestaw instrukcji klawiatury 2 [1], widać że możemy otrzymywać następujące kombinacje bajtów:

- naciśnięcie zwykłego przycisku (np. litera 'a') - klawiatura wyśle jeden bajt - 1C
- puszczenie zwykłego przycisku (np. litera 'a') - klawiatura wyśle dwa bajty - F0 oraz 1C
- naciśnięcie specjalnego przycisku (np. strzałka w lewo) - klawiatura wyśle dwa bajty - E0 oraz 6B
- puszczenie specjalnego przycisku (np. strzałka w lewo) - klawiatura wyśle trzy bajty - E0, F0 oraz 6B

Jak widać poza informacją o jaki przycisk chodzi możemy również otrzymać informację, o tym, że przycisk jest puszczany i o tym, że jest to przycisk specjalny.

Realizując projekt zdecydowałem się na kompletne ignorowanie momentu kiedy przycisk jest puszczany. Dzięki takiemu rozwiązaniu chcąc wypisać dowolny znak na wyświetlaczu procesor otrzymując dwie informacje - wciśnięcie oraz puszczenie przycisku - przetwarza tylko jedną z nich (wciśnięcie) i wypisuje znak jeden raz.

Architektura części odpowiedzialnej za obsługę otrzymanych bajtów zrealizowana została jako maszyna stanów. Jej architektura przedstawiona została na poniższym rysunku:



Rysunek 2: Maszyna stanów aplikacji

4.2 Implementacja

4.2.1 Konfiguracja

Przed wejściem do funkcji main wykonywana jest następująca konfiguracja:

- wyzerowanie wszystkich rejestrów (po resecie płytki nie są one zerowane)
- włączenie przerwań (instrukcja EINT)
- włączenie przerwań od PS/2
- konfiguracja PS/2 - ustawienie typu przerwań na zbocze, ustawienie przerwań na zboczu opadającym, ustawienie przerwań od linii zegara
- inicjalizacja wyświetlacza
- ustawienie kursora w lewym górnym rogu wyświetlacza
- ustawienie stanu początkowego na WAIT_FOR_SIGN

Poniższy listing przedstawia wykonanie powyższych czynności w kodzie programu:

```
1 CALL    reset_regs
2 EINT
3 LOAD    s0, 0b00000010
4 OUT     s0, int_mask
5 CALL    configure_ps2
6
7 CALL    init_display
8
```

```
 9  CALL    push_reg
10  LOAD    s0, 0x80
11  _PUSH   s0
12  CALL    set_cursor
13  CALL    pop_reg
14
15  LOAD    s0, WAIT_FOR_SIGN_STATE
16  STORE   s0, STATE
```

4.2.2 Obsługa przerwań

Każda przychodząca ramka danych od klawiatury posiada 11 bitów:

- 1 bit startu
- 8 bitów danych
- 1 bit parzystości
- 1 bit stopu

W procedurze obsługi przerwań każdy liczony jest każdy bit (licznik znajduje się pod adresem 255). Gdy obsługiwany jest jeden z bitów: 1, 10 lub 11 - jest on po prostu ignorowany. Pozostałe bity (od 2 do 9) to bity danych i zapisywane są w pamięci ram (pod adresem 254). Z każdym kolejnym odebranym bitem pobierane są z pamięci RAM dotychczas otrzymane bity, przesuwane o jeden w lewo, a w pozycji LSB zapisywany jest nowo odebrany bit. Dzięki takiej procedurze, po otrzymaniu 8 bitów danych, pierwszy odebrany bit znajduje się na pozycji MSB, drugi odebrany na MSB-1 itd do ostatniego odebranego, który znajduje się na pozycji LSB.

Po odebraniu 8 bitów danych, znak znajdujący się pod adresem 254, przepisywany jest do adresu 0. Adres ten monitorowany jest cały czas przez wątek procesora, więc po chwili zostaje przez niego odczytany i wyzerowany. Gdy program zajmuje się obsługą otrzymanego znaku, funkcja obsługująca przerwanie może odbierać w tym samym czasie kolejny znak i gdy będzie on gotowy wystawić go pod adres 0.

Poniższy listing przedstawia kod odpowiedzialny za wykonanie wyżej wymienionych czynności:

```
 1
 2  interrupt_handler:
 3      CALL    push_reg
 4      FETCH   s0, PS2_COUNTER
 5      ADD     s0, 1
 6      STORE   s0, PS2_COUNTER
 7
 8      COMP    s0, 1
 9      JUMP    Z, skip_int
10      COMP    s0, 10
11      JUMP    Z, skip_int
12      COMP    s0, 11
13      JUMP    Z, skip_int_fin
14
15      IN      s2, ps2
16      SR0     s2
17      RR      s2
18
```

```
19      FETCH   s1, SIGN
20      SR0     s1
21      OR      s1, s2
22      STORE   s1, SIGN
23
24      JUMP    skip_int
25
26  skip_int_fin:
27      FETCH   s1, SIGN
28      STORE   s1, SIGN_READY
29
30      LOAD    sE, 0
31      STORE   sE, SIGN
32      STORE   sE, PS2_COUNTER
33
34  skip_int:
35      LOAD    sD, 0
36      OUT     sD, int_status
37      CALL    pop_reg
38      RETI
39
40  .CSEG   0x3FF
41  JUMP    interrupt_handler
```

4.2.3 Mapowanie znaków

Mapowanie znaków z zestawu instrukcji klawiatury na ASCII zapisane zostało w pamięci RAM i zaczyna się od adresu 50. Mapowanie wygląda następująco:

```
1  .DSEG 0, 50
2
3  SIGNS_BEGIN:
4      .DB
5      0x1C, 'a',
6      0x32, 'b',
7      0x21, 'c',
8      ...
9      0x25, '4',
10     0x2E, '5',
11     0x36, '6',
12     ...
13     0x41, ',',
14     0x49, '.',
15     0x4A, '/',
16     0x66, 254, ; BACKSPACE
17     0x5A, 255  ; ENTER
18 SIGNS_END:
19
20 .CSEG
```

Jak widać są to następujące po sobie kod znaku, który wysyła klawiatura oraz znak ASCII, który należy obsłużyć (np. wysłać na wyświetlacz).

4.2.4 Pętla główna programu

W pętli głównej, program sprawdza co znajduje się pod adresem 0 RAM - czyli tam, gdzie zapisywany jest znak pochodzący od klawiatury. Jeżeli pobrany znak to 0 (czyli

z klawiatury nie przyszło nic nowego), to program nie robi nic. Jeżeli jest on różny od 0 (stała SIGN_READY), to wykonywany jest skok do obsługi otrzymanego znaku w aktualnym stanie. Pętla main przedstawiona została na poniższym listingu:

```
1  main:
2      FETCH    s0, SIGN_READY
3      COMP     s0, 0
4      JUMP     Z, skip
5
6      LOAD     s1, 0
7      STORE    s1, SIGN_READY
8
9      FETCH    s1, STATE
10
11     COMP     s1, WAIT_FOR_SIGN_STATE
12     JUMP     Z, handle_wait_for_sign_state
13
14     COMP     s1, SKIP_NEXT_SIGN_STATE
15     JUMP     Z, handle_skip_next_sign_state
16
17     COMP     s1, WAIT_FOR_NEXT_PART_STATE
18     JUMP     Z, handle_wait_for_next_part_state
19 skip:
20     JUMP     main
```

4.2.5 Stan WAIT_FOR_SIGN

Gdy odebrany został znak i program znajduje się w tym stanie, to sprawdzone zostaną następujące warunki:

- jeżeli odebrany znak to F0 - stan zostanie zmieniony na SKIP_NEXT_SIGN
- jeżeli odebrany znak to E0 - stan zostanie zmieniony na WAIT_FOR_NEXT_PART

W przypadku, gdy nie został spełniony żaden z powyższych warunków, odebrany znak zostanie jak wciśnięcie podstawowego przycisku (np. litery, liczby, lub znaku), czyli:

- jeżeli otrzymano **enter** - kursor ustawiony zostanie w lewym dolnym rogu
- jeżeli otrzymano **backspace** - wykonane zostaną trzy czynności: przesuń kursor w lewo, wpisz pusty znak (spację), przesuń kursor w lewo
- w pozostałych przypadkach nastąpi po prostu wypisanie znaku na wyświetlacz

Poniższy listing przedstawia kod odpowiedzialny za wykonanie powyższych czynności:

```
1
2  handle_wait_for_sign_state:
3      COMP     s0, 0xF0
4      JUMP     Z, handle_f0_received
5
6      COMP     s0, 0xE0
7      JUMP     Z, handle_e0_received
8
9      JUMP handle_sing
10
11 handle_f0_received:
```

```
12      LOAD    s1, SKIP_NEXT_SIGN_STATE
13      STORE   s1, STATE
14      JUMP    skip
15
16  handle_e0_received:
17      LOAD    s1, WAIT_FOR_NEXT_PART_STATE
18      STORE   s1, STATE
19      JUMP    skip
20
21  handle_sing:
22      ...
```

4.2.6 Stan SKIP_NEXT_SIGN

Jest to specjalny stan, w który program przechodzi po otrzymaniu od klawiatury bajtu 0xF0. Oznacza to, że puszczony został przycisk, którego kod zostanie przesłany jako następny. Jak wspomniane zostało wcześniej, puszczanie klawiszy jest kompletnie ignorowane, więc w tym stanie po odegraniu znaku od klawiatury, program go zignoruje i przejdzie do stanu oczekiwania na nowe znaki z klawiatury (WAIT_FOR_SIGN).

Poniższy listing przedstawia obsługę tego stanu:

```
1  handle_skip_next_sign_state:
2      LOAD    s1, WAIT_FOR_SIGN_STATE
3      STORE   s1, STATE
4      JUMP    skip
```

4.2.7 Stan WAIT_FOR_NEXT_PART

Program przechodzi do tego stanu po otrzymaniu od klawiatury bajtu 0xE0. Oznacza to, że kolejny bajt, który wyśle klawiatura:

- jest klawiszem specjalnym
- jest puszczeniem przycisku (bajt F0)

Jeżeli jest to puszczenie przycisku to program przechodzi do stanu SKIP_NEXT_SIGN. W innym przypadku sprawdza, o który znak specjalny chodzi. W przypadku tego projektu, ze znaków specjalnych obsługiwane są tylko strzałki. Gdy program w tym stanie otrzyma bajt danych, sprawdzi czy pasuje on z kodem, któregoś z przycisków dla strzałek i przejdzie od odpowiedniej funkcji, która wykona przesunięcie kursora.

Poniższy listing przedstawia obsługę tego stanu:

```
1  handle_wait_for_next_part_state:
2      COMP    s0, 0xF0
3      JUMP    Z, handle_f0_received
4      JUMP    handle_special
5
6  handle_special:
7      COMP    s0, 0x6B ; left arrow
8      JUMP    Z, handle_left
9
10     COMP    s0, 0x74 ; right arrow
11     JUMP    Z, handle_right
12
13     COMP    s0, 0x75 ; up arrow
```

```
14      JUMP    Z, handle_up
15
16      COMP    s0, 0x72 ; down arrow
17      JUMP    Z, handle_down
18
19  handle_special_before_fin:
20      LOAD    s1, WAIT_FOR_SIGN_STATE
21      STORE   s1, STATE
22      JUMP    skip
```

Literatura

[1] AT Keyboard Scan Codes (Set 2). https://webdocs.cs.ualberta.ca/~amaral/courses/329/labs/scancodes.html.

[2] Spartan-3A/3AN Starter Kit Manual. *https://www.gta.ufrj.br/ensino/EEL480/spartan3/ug334.pdf*.

[3] Krzysztof Świentek. http://www.fis.agh.edu.pl/koidc/staff.php?worker=6&type=2.