AGH, Szymon Telega, Michał Wojtasik

Projekt na Systemy Wbudowane, I2C

"Gra w kulkę" przy użyciu akcelerometru "Pmod ACL" oraz wyświetlacza LCD

Dokumentacja

I. Założenia wstępne

Założeniem projektu było stworzenie gry w kulkę, przy pomocy akcelerometru Pmod ACL oraz wyświetlacza. Przesył danych pomiędzy kontrolerem a akcelerometrem został zrealizowany przy użyciu inteferjsu I2C. Użytkownik przy użyciu akcelerometru powinien mieć możliwość sterowania kulką, w celu przejścia "labiryntu" wyrysowanego na ekranie LCD.

II. Akcelerometr Pmod ACL

Akcelerometr ma różne 'tryby' pracy, potrafi mierzyć zarówno zwykłe przyspieszenie na trzech osiach X, Y oraz Z, jak i również rejestrować 'tapnięcie' pojedyncze i podwójne, rejestrować spadek swobodny etc. W naszym przypadku przydatną funkcją jest mierzenie "aktywności" przyspieszenia na osiach, w celu późniejszego sterowania położenia kulki.

Aktywność na tych trzech osiach, potrafi on sczytywać w 4 różnych zakresach:

1. -2g : 2g
2. -4g : 4g
3. -8g : 8g
4. -16g : 16g

Nasz projekt nie wymagał rejestracji większego przyspieszenia niż -1g oraz 1g - założeniem jest iż użytkownik nie będzie machał akcelerometrem, co jest utrudnione połączeniem kablowym z mikrokontrolerem.

Akcelerometr ten aby go uruchomić i odbierać dane trzeba było w naszym przypadku 'zainicjalizować' początkowymi wartościami w pewnych rejestrach. Rejestry tego akcelerometru widoczne są poniżej:

REGISTER MAP

Table 19.

Address					
Hex	Dec	Name	Type	Reset Value	Description
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

W naszym przypadku nadpisania wymagały rejestry Data Format Control (zakres pracy tj. -2g : 2g), ActivityTreshold , Axis enable control for activity, Power-saving features (włączenie akcelerometru).

Odbiór danych

Akcelerometr ten jak widać w zbiorze rejestrów ma po dwa rejestry 8-bitowe danych na każdą oś. Aktywność na danej osi znajduje się na 10 najmłodszych bitach tych rejestrów. Wartość, zapisana jest w notacji U2. Nas interesuje przyspieszenie na osi X oraz Y.

III. Interfejs I2C

Konfiguracja tego interfejsu nie należała do najprostszych, jednak również nie najtrudniejszych. Interfejs ten najpierw został skonfigurowany w main(), a następnie napisane zostały dwie funkcje, jednak która po podaniu jej adresu rejestru slave'a oraz danych, wpisywała te dane do danego rejestru – druga natomiast po podaniu adresu rejestru slave'a, sczytywała z niego dane. Dane odbierane oraz wysyłane były po 1 bajcie.

IV. Realizacja całości

Do zaprogramowania naszej gry, potrzebowaliśmy jakiegoś "czasomierza", jako że chcieliśmy operować na wzorach na drogę, oraz prędkość, oraz aby co sensowny interwał czasowy wyrysowywać położenie kulki od nowa. Skorzystaliśmy z prostej konfiguracji SysTicka, który generował przerwanie co 1 / 60 sekundy.

W handlerze SysTicka, mając zdefiniowane położenie kulki jako zmienne globalne, sczytywaliśmy aktualne przyspieszenie na osiach X oraz Y akcelerometru, a następnie dzięki jego pomocy, modyfikowaliśmy prędkość (która również była zmienną globalną, przy czym na początku jest równa 0). Na prędkość kulki nałożyliśmy pewne ograniczenie odgórne, z praktycznych powodów funkcjonalności gry.

Początkowo zamierzaliśmy prawidłowo przekształcać przyspieszenie, na jednostki prawdziwego przyspieszenia, jednak koniec konców postanowiliśmy tego nie robić, a jedynie doświadczalnie przemnożyć otrzymane przyspieszenie przez pewną stałą, aby funkcjonalność była odpowiednia.

Wyrysowana na wyświetlaczu została czarna plansza, z białym labiryntem, którego kulka nie może dotknąć, oraz z jedną zieloną ścianką, do której musi "dojść". Co każde wywołanie przerwania, czyli co 1/60 sekundy, po obliczeniu przemieszczenia kulki, następuje próba zamalowania poprzedniego miejsca w którym się znajdowała, oraz narysowanie jej w miejscu starym + przemieszczenie. W momencie rysowania kulki w nowym miejscu sprawdzany jest kolor pikseli, aby sprawdzić czy kulka wchodzi na ściankę (która jest biała) lub do mety (która jest zielona). W przypadku wejścia w ściankę gra zaczyna się od nowa, w przypadku dojścia do mety, wyświetlany jest napis "Win Win!"