

SW Oscilloscope

Projekt z przedmiotu Systemy Wbudowane

Franciszek Urbański

18 I 2025

Spis treści

1 Cel projektu	1
2 Lista połączeń	1
3 Instrukcja obsługi	2
4 Implementacja	3
4.1 Buffer	3
4.2 LCDControl	4
4.3 Control	4
4.4 ADC	4
4.5 Trigger	5
4.6 main.c	5

1 Cel projektu

Celem projektu było stworzenie podstawowego oscyloskopu zbierającego sygnał z przetwornika ADC, a następnie wyświetlającego przebiegi na ekranie LCD. Oscyloskop miał również umożliwiać odczytanie podstawowych parametrów sygnału (okres, amplituda).

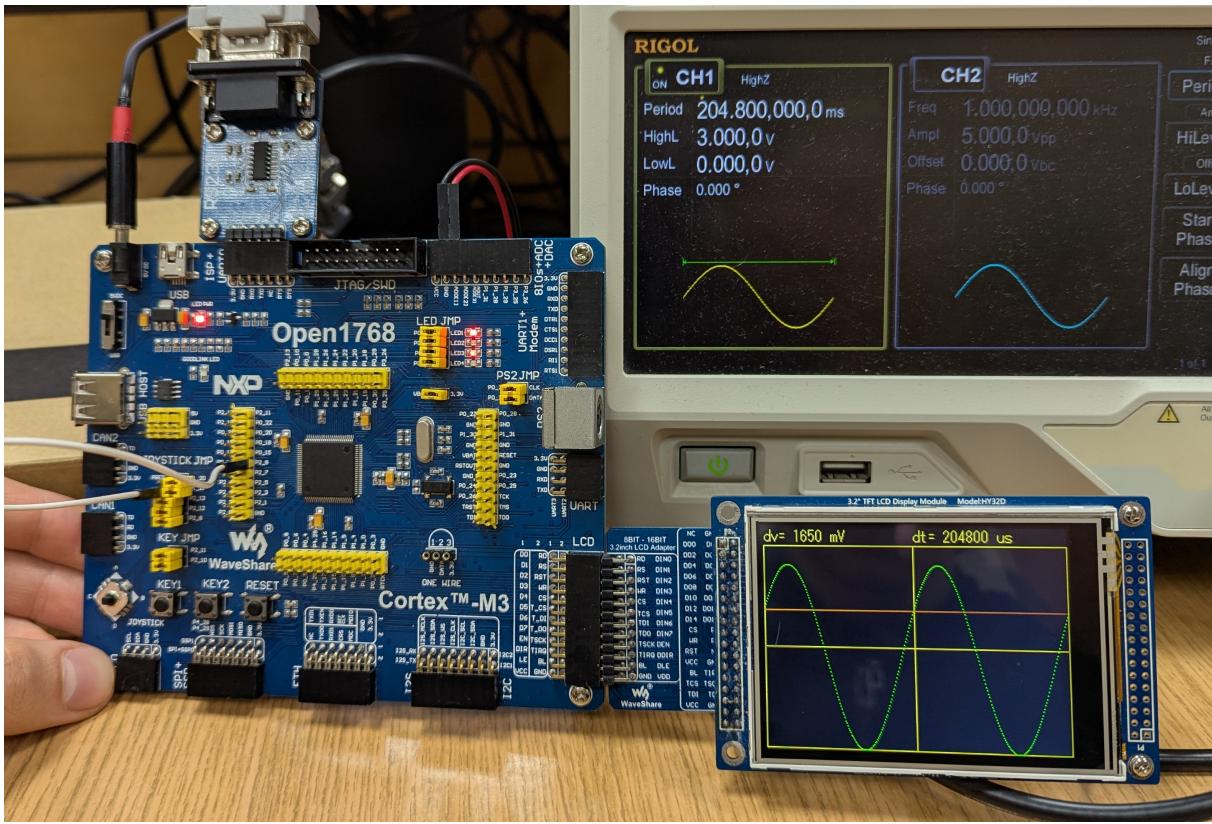
2 Lista połączeń

Projekt wykorzystuje:

- Płytkę Open1768, z kablem zasilającym
- Moduł interfejsu RS232 (opcjonalny)
- Moduł z wyświetlaczem LCD 3.2”
- Dodatkowy kabelek jako zworkę

Moduły te należy podłączyć jak na rys. 1., a zatem:

- Moduł interfejsu RS232 do dedykowanych wyjść UART0 (opcjonalne)
- Ekran LCD do dedykowanych wyjść dla LCD
- Dodatkowy kabelek: należy usunąć zworki D oraz PRESS na wyprowadzeniu JOYSTICK JMP, a następnie połączyć D z najbliższym wyprowadzeniem pinu P2_9. Wyprowadzenie PRESS należy zostawić niepodłączone. Pozostałe przyciski oraz wyprowadzenia joysticka podłączone są w sposób domyślny.
- Wejście oscyloskopu: wejście dodatnie należy wpiąć do wyprowadzenia AD0[1], a ujemne do GND (w grupie wyprowadzeń dedykowanej dla ADC, na prawo od wejścia programatora). Obsługiwany zakres napięć to 0-3.3V



Rysunek 1: Zmontowany oscyloskop

3 Instrukcja obsługi

Aby uruchomić oscyloskop, należy podłączyć wszystko zgodnie z listą w sekcji Lista połączeń. Oscyloskop od razu po uruchomieniu jest gotowy do użytku. Można od razu podać sygnał na jego wejście, jak jest to opisane pod koniec sekcji Lista połączeń. Oscyloskop był testowany i działa poprawnie na sygnałach różnych kształtów, o częstotliwości w zakresie 1 Hz - 2 kHz.

Interfejs graficzny użytkownika na ekranie LCD przedstawiony jest na rys. 2. Jest on bardzo prosty. Na górze opisane są szerokości podziałek napięcia (dv) i czasu (dt), przy czym przez podziałkę mam na myśli odległość między liniami siatki (czyli tutaj pół ekranu). Pod spodem znajduje się panel wyświetlający przebiegi. Linie siatki są żółte, natomiast brązowa linia pozioma to próg triggera. Zielony ślad to oczywiście sygnał podany na oscyloskop.

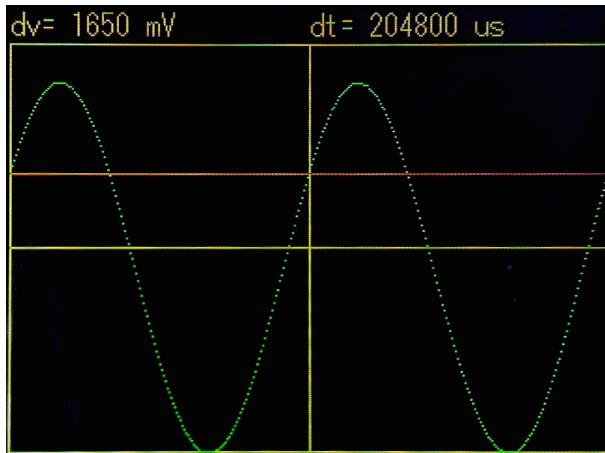
Domyślne (początkowe) wartości ustawień to:

- $dv = 1650 \text{ mV}$ (3300 mV na całą wysokość panelu)
- $dt = 51200 \mu\text{s}$ (102.4 ms na całą szerokość panelu)
- poziom triggera: 1650 mV (w połowie wysokości)

Ustawienia można zmieniać za pomocą joysticka i przycisków KEY1 oraz KEY2. Ruszanie joystickiem w płaszczyźnie pionowej zmienia skalę napięcia ($\pm 300 \text{ mV}$, w górę przybliża widok), natomiast w płaszczyźnie poziomej zmienia skalę czasu ($\cdot 2^{\pm 1}$, w prawo przybliża widok). Skala napięcia zmieniana jest w taki sposób, że napięcie 0V zawsze jest na samym dole. Wciśnięcie przycisku KEY1 obniża próg triggera o 300 mV, zaś KEY2 podnosi go o 300mV.

Opcjonalnym dodatkiem do oscyloskopu jest wyjście UART, które można podpiąć do portu UART0 (jak opisano w sekcji Lista połączeń). Parametry wyjścia to:

- Data bits: 8
- Parity: none
- Stop bits: 1



Rysunek 2: Interfejs graficzny użytkownika na LCD

- Flow control: none
- Baud rate: 115200

Na początek użytkowania oscyloskopu, użytkownik jest witany na wyjściu UART komunikatem Helou. Przy każdej zmianie ustawień wypisywana jest linia w formacie:

`st=[skok czasowy];vm=[maksymalne napięcie];th=[próg triggera];dt=[dt];dv=[dv]`

Dodatkowo, jeśli program skompilowany był ze zdefiniowanym symbolem DEBUG, z każdym wywołaniem triggera wypisywane są wartości: Trig: [poprzednia]; [próg]; [następna]. Znaczenie wypisywanych wartości stanie się jasne w dalszych częściach dokumentacji.

Pomocne w rozwiązywaniu problemów są też odpowiednie diody LED:

1. Zapala się po pomyślnie ukończonej konfiguracji (procedura setup)
2. Nie świeci
3. Zapala się gdy ADC zbiera próbkę, gaśnie gdy jesteśmy w głównym wątku
4. Zapala się gdy rysowany jest sygnał

4 Implementacja

Kod podzielony jest na moduły odpowiadające za poszczególne aspekty funkcjonowania oscyloskopu. Każdy moduł (poza main) składa się z pliku `*.h` oraz `*.c`. Działanie aplikacji najlepiej zrozumieć analizując kolejne moduły.

Kod dostępny jest pod adresem <https://github.com/frun36/sw-oscilloscope>.

4.1 Buffer

Moduł ze strukturą **Buffer**, która wykorzystywana jest jako bufor w oscyloskopie. Składa się z dwóch buforów `uint16_t` o takiej samej pojemności (jeden na nowe wartości, drugi na archiwalne), oraz zmiennej śledzącej obecne zapełnienie bufora na nowe wartości. Deklaruje również dwie funkcje operujące na tym buforze:

- `uint8_t buff_append(Buffer* buff, uint16_t val)` - próbuje wstawić element na koniec bufora. Jeśli się to udało zwraca 1 - jeśli nie (bufor przepelny) zwraca 0.
- `void buff_clear(Buffer* buff)` - kopiuje zawartość bufora na nowe wartości do bufora na wartości archiwalne, czyści bufor na nowe wartości (ustawia rozmiar na 0)

4.2 LCDControl

Moduł grupujący funkcje służące do rysowania po wyświetlaczu LCD. Korzysta on z następujących bibliotek: **Open1768_LCD**, **LCD_ILI9325** oraz **asciiLib** (wykorzystywanych na zajęciach z ekranem LCD). Deklaruje funkcje do wypełniania tła (szybkie, z autoinkrementacją), rysowania punktów, linii poziomych i pionowych, znaków i tekstu. Jest też funkcja do inicjalizacji LCD i przygotowania do działania w oscyloskopie (wywołanie inicjalizujących funkcji bibliotecznych, wypełnienie tła na czarno, narysowanie siatki).

Na szczególną uwagę zasługuje tutaj funkcja **void draw_traces(uint16_t* buff, uint16_t* old, uint32_t size, uint16_t color)**, która rysuje przebiegi. Tutaj wyjaśnia się zastosowanie archiwального bufora w strukturze **Buffer** - przed narysowaniem nowego śladu, stary ślad zamalowywany jest na czarno, co jest szybsze niż przerysowanie całego ekranu na nowo. Dodatkowo w tej metodzie zaimplementowana jest logika zapobiegająca zamalowaniu linii siatki oraz progu triggera - korzystając z metod **is_on_grid** oraz **get_trig_y**, punkty spełniające te kryteria malowane są na kolory odpowiednio **GRID_COLOR** oraz **TRIG_COLOR**.

4.3 Control

Moduł odpowiedzialny za kontrolki oscyloskopu oraz zmieniane przez nie ustawienia. Wykorzystuje on:

- Joystick: A - P2.8, B - P2.12, C - P2.13, D - P2.9 (dodatkowy kabelek)
- Przyciski: KEY1 - P2.11, KEY2 - P2.10
- Na opadającym zboczu odpowiednich pinów z portu 2 (stąd konieczność dodatkowego kabelka) ustawione jest przerwanie EINT3 - jego obsługa robi prosty debouncing (pusta pętla **for** przez 200000 cykli zegara) a następnie sprawdza, co należy zmienić w ustawieniach (zależnie od aktywnej kontroli).
- peryferium UART0 - implementacja wykorzystująca tryb asynchroniczny z biblioteki CMSIS, o parametrach opisanych w sekcji Instrukcja obsługi.

Moduł ten tworzy również globalnąinstancję structa **Control**, przechowującego ustawienia oscyloskopu (**step**, **vmax**, **thresh** - wyjaśnione w sekcji ADC) oraz znane już (obliczone) **dt** i **dv**.

W pliku **Control.h** mamy również zdefiniowaną stałą **SAMPLE_US**, określającą częstotliwość próbkowania oscyloskopu. W wersji ostatecznej jest ona ustawiona na $20 \mu\text{s}$ - jest to najmniejsza wartość, przy której oscyloskop działał bez zauważalnych wyraźnie błędów (np. uciekające piksele na LCD).

4.4 ADC

W module **ADC** znajduje się

- funkcja konfiguruująca timer TIM0 (rejestry z **LPC17xx.h**)
- funkcja konfiguruująca przetwornik ADC0 (rejestry z **LPC17xx.h**)
- obsługa przerwania z ADC0 (serce logiki oscyloskopu)

Timer TIM0 ma wejściowy zegar równy **CCLK** (dla zwiększenia precyzji odmierzania czasu), w taki sposób by generować sygnał dla ADC co **SAMPLE_US** mikrosekund (a zatem musi dwa razy częściej przełączać **MAT0.1**).

ADC wykonuje konwersję na wejściu **ADC0.1**, na każdym narastającym zboczu **MAT0.1** (stąd konieczność 2 razy szybszego timera, opadające zbocze jest ignorowane). Zegar peryferyjny ADC ustawiony jest na **CCLK / 2**, a prescaler w ten sposób, że ostateczny zegar ADC ma 10 MHz (wystarczająco szybki).

Po wykonaniu konwersji, ADC ma skonfigurowane przerwanie (globalnie, nie na konkretnym kanale). W procedurze jego obsługi mieści się cała logika zbierania danych i triggerowania (**ADC.c**, **void ADC_IRQHandler()**).

Procedura ta ma właściwie dwa tryby działania - przed i po spełnieniu warunku triggera (opisanego w sekcji Trigger). Przed spełnieniem warunku triggera procedura (flaga **triggered** nieustawiona):

- Odczytuje wartość z rejestru **ADGDR** (czyszcząc tym samym informację o przerwaniu)

- Zapisuje co n -tą wartość do porównania pod kątem triggera, gdzie $n = 8 \cdot \text{control.step}$ - w zmiennej `prev_mv`
- Do sprawdzenia w funkcji z warunkiem triggera z każdą iteracją przekazywane są wartości: `prev_mv`, `control.thresh` oraz obecny pomiar.

Jeśli po takim sprawdzeniu okaże się, że warunek triggera jest spełniony, ustawiana jest flaga `triggered` i procedura:

- Pomija sprawdzanie warunku triggera
- Każdy m -ty pomiar zapisuje do bufora, gdzie $m = \text{time_step}$ (skalując go do wartości współrzędnej na LCD, przycinając jeśli przekracza współrzędną maksymalną)
- Gdy bufor się przepełni, czyści flagę `triggered`, a ustawia flagę `do_draw` (obsługiwana w głównym wątku, uruchomienie rysowania sygnału)

4.5 Trigger

W module tym testowałem różne triggery (np. `level_trigger` - triggerowanie jeśli sygnał osiągnie określony poziom), ostatecznie używany jest tylko `edge_trigger`.

Trigger ten uruchamia się, gdy sygnał “przebiję” określona wartość progową, tj. gdy poprzednia wartość będzie mniejsza, a następna większa niż próg. Efektywnie więc wykrywa on narastające zbocza przekraczające próg triggera. Jest on skuteczniejszy niż triggerowanie poziomem, ponieważ poziom osiągany jest zarówno na narastających jak i opadających zboczach i triggerowanie poziomem powoduje przeskakujący widok sygnału.

4.6 main.c

W pliku `main.c` mamy dwie funkcje - `void setup()` i `void loop()`.

Funkcja `setup` dokonuje inicjalizacji:

- LED
- LCD
- TIM0
- ADC
- Kontrolek

oraz zapala pierwszą diodę.

W funkcji `loop` zaś mamy:

- obsługa rysowania (`handle_draw` z modułu `ADC`) - funkcja rysuje sygnał, jeśli ustawiona jest flaga `do_draw`
- zgaszenie diody zapalonej w obsłudze przerwania ADC
- `__WFI()` - czekanie na przerwanie, dla oszczędności energii

Jak widać, w głównym wątku oprócz rysowania nie dzieje się zbyt wiele - cała logika oscyloskopu, czyli obsługa odczytu i triggerów, ma miejsce w obsłudze przerwania ADC, a zmiana parametrów oscyloskopu w obsłudze przerwania z GPIO2 (EINT3).