Projekt

Sterowniki robotów

Dokumentacja

Oscyloskop
KD 23MTS

Skład grupy:
Dominik Pluta, 263460
Kamil Winnicki, 263434

Termin: srTN17

Prowadzący:
dr inż. Wojciech Domski

5 czerwca 2024

Spis treści

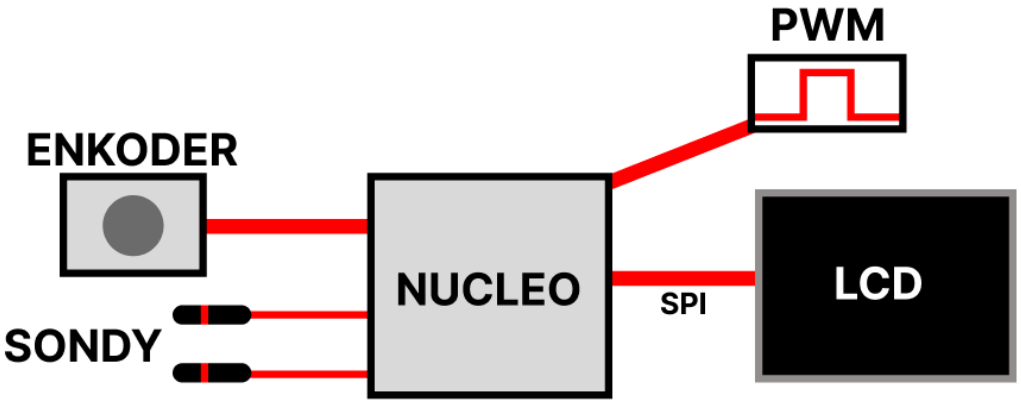
1 Opis projektu

Oscyloskop [2] [3] zbudowany w oparciu o mikrokontroler STM32L474RG oraz dotykowy, 4"wyświetlacz LCD. Stworzone urządzenie będzie posiadało następujące funkcjonalności:

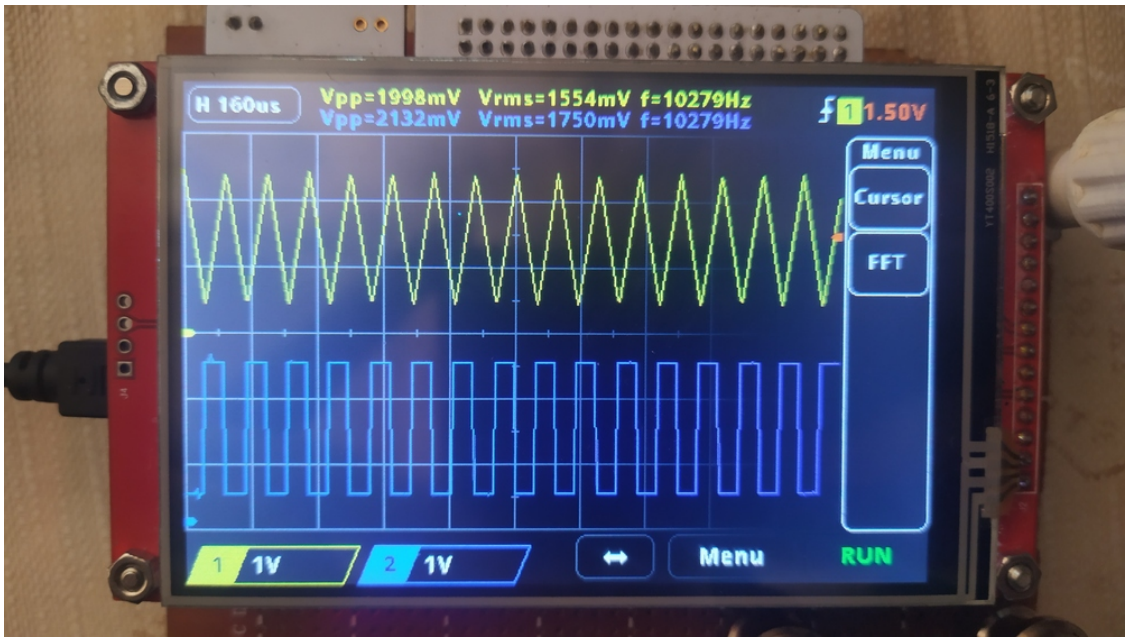
- dwa kanały,
- skalowanie osi X oraz Y,
- przesuwanie wykresu w osi X oraz Y,
- dwa kursory w osi X oraz Y,
- pomiar RMS,
- pomiar peak-to-peak,
- pomiar częstotliwości,
- trigger,
- szybką transformate fouriera,

Do obsługi oscyloskopu przez użytkownika wykorzystany będzie enkoder oraz dotyk w ekranie.

Do testów oscyloskopu wykorzystany będzie sygnał PWM wygenerowany na jednym z pinów mikrokontrolera.



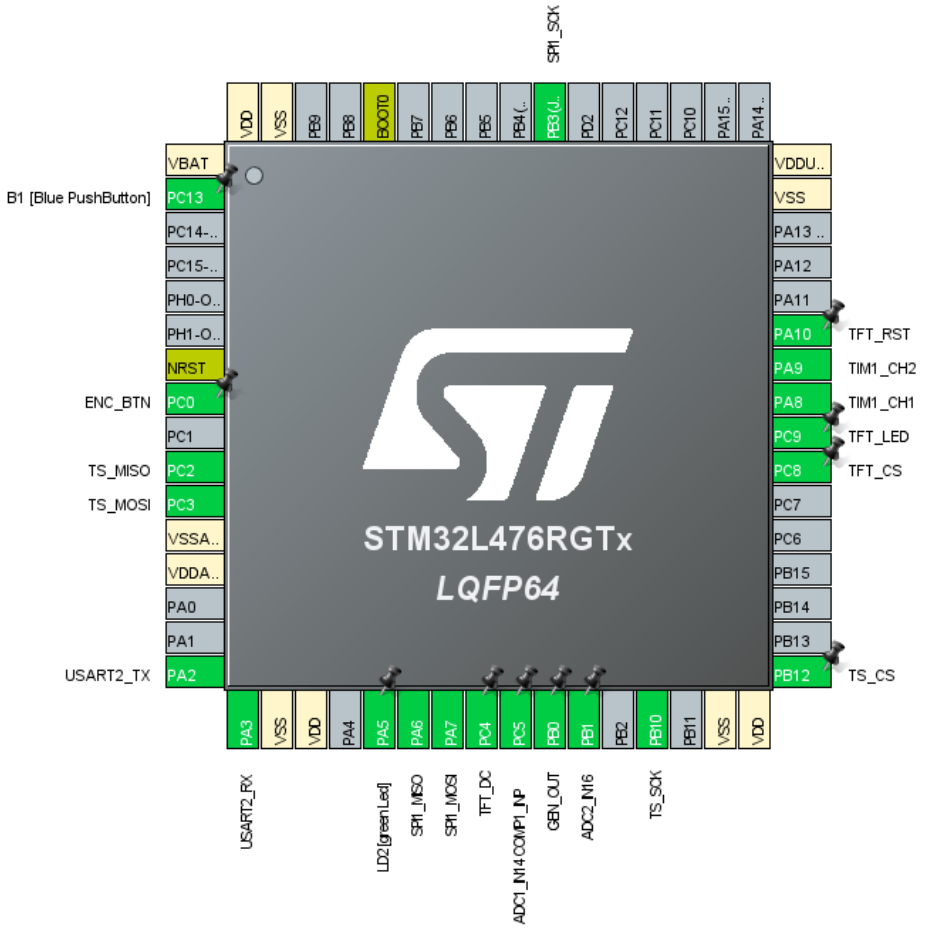
Rysunek 1: Architektura systemu



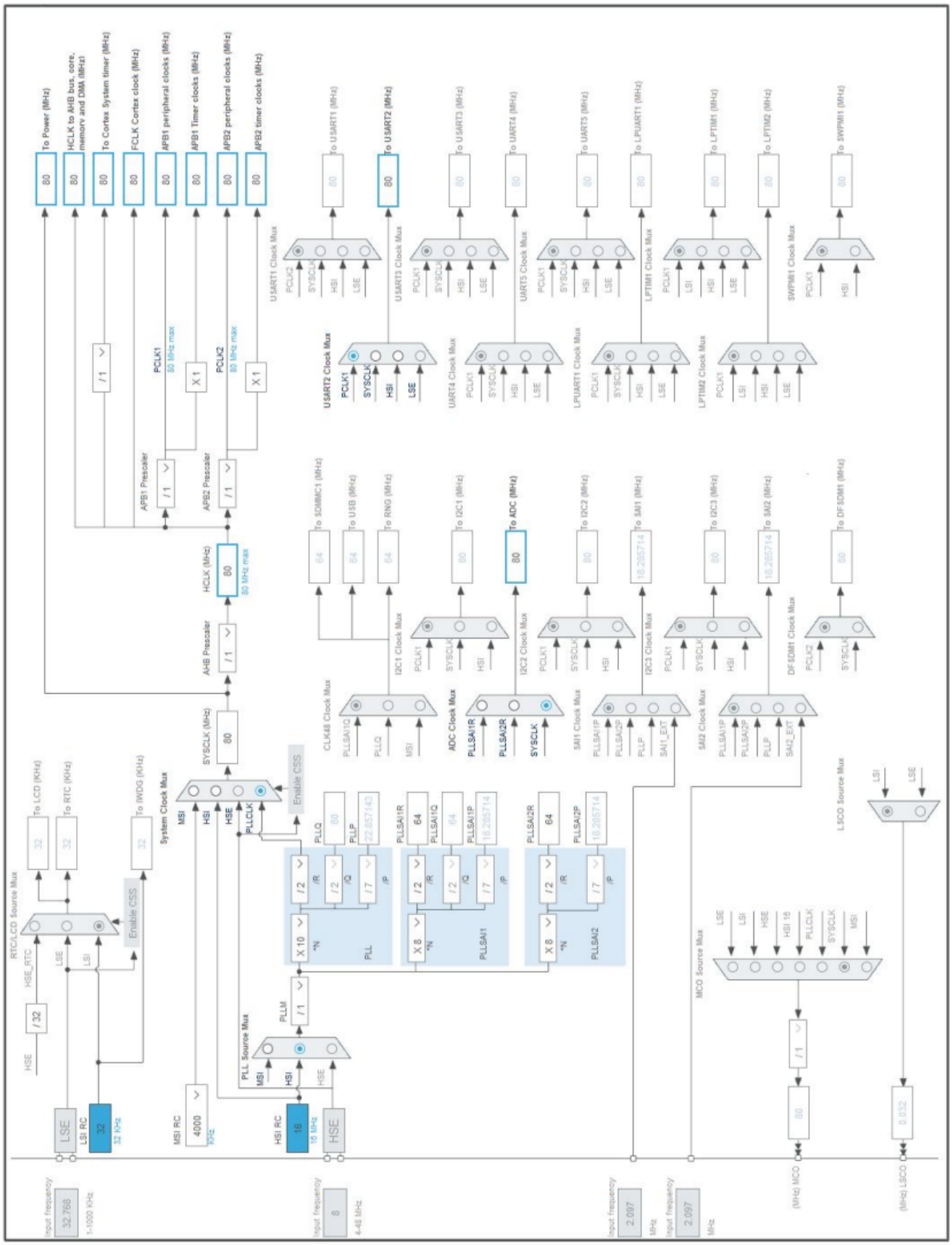
Rysunek 2: Zdjęcie projektu

Rysunek 3: Porównanie zmierzonych sygnałów do zadanych

2 Konfiguracja mikrokontrolera



Rysunek 4: Konfiguracja wyjść mikrokontrolera w programie STM32CubeMX



Rysunek 5: Konfiguracja zegarów mikrokontrolera

2.1 Konfiguracja pinów

Numer pinu	PIN	Tryb pracy	Funkcja/etykieta
2	PC13	GPIO_EXTI13	B1 [Blue PushButton]
8	PC0	GPIO_Input	ENC_BTN
10	PC2	SPI_MISO	TS_MISO
11	PC3	SPI_MOSI	TS_MOSI
16	PA2	USART2_TX	USART2_TX
17	PA3	USART2_RX	USART2_RX
21	PA5	GPIO_Output	LD2 [green Led]
22	PA6	SPI_MISO	SPI1_MISO
23	PA7	SPI_MOSI	SPI1_MOSI
24	PC4	GPIO_Output	TFT_DC
25	PC5	ADC1_IN2, COMP1_NP	CH1 & trigger
26	PB0	GPIO_Output	GEN_OUT
27	PB1	ADC2_IN16	CH2
29	PB10	SPI2_SCK	TS_SCK
33	PB12	GPIO_Output	TS_CS
39	PC8	GPIO_Output	TFT_CS
40	PC9	TIM3_CH4	TFT_LED
41	PA8	TIM1_CH1	TIM1_CH1
42	PA9	TIM1_CH2	TIM1_CH2
43	PA10	GPIO_Output	TFT_RST
55	PB3	SPI1_SCK	SPI2_SCK

Tabela 1: Konfiguracja pinów mikrokontrolera

2.2 SPI1

Konfiguracja peryferium interfejsu SPI, wykorzystywanego do komunikacji z LCD. Wszystkie piny tego peryferium mają ustawione 'Maximum output' speed na 'Very High'.

Parametr	Wartość
Mode	Full-Duplex Master
Frame Format	Motorola
Data Size	8 bit
First bit	MSB first
Prescaler (for Baud Rate)	2 (40 MBits/s)
Clock polarity	Low
Clock phase	1 Edge

Tabela 2: Konfiguracja peryferium SPI

2.3 SPI2

Konfiguracja peryferium interfejsu SPI, wykorzystywanego do komunikacji z panelem dotykowym na LCD. Wszystkie piny tego peryferium mają ustawione 'Maximum output' speed na 'Very High'.

Parametr	Wartość
Mode	Full-Duplex Master
Frame Format	Motorola
Data Size	8 bit
First bit	MSB first
Prescaler (for Baud Rate)	16 (5 MBits/s)
Clock polarity	Low
Clock phase	1 Edge

Tabela 3: Konfiguracja peryferium SPI

2.4 ADC

Wykorzystujemy wbudowane przetworniki ADC w mikrokontrolerze. ADC1 odpowiada za pomiary z kanału 1 oscyloskopu, natomiast ADC2 za pomiary z kanału 2 oscyloskopu. Oba przetworniki zostały skonfigurowane w poniższy sposób.

Parametr	Wartość
Resolution	12-bit
Continous conversion mode	Enabled
DMA Continuous Requests	Enabled
Sampling time	6.5 Cycles

Tabela 4: Konfiguracja ADC

2.5 COMP1

Konfiguracja komparatora wykorzystywaego do implementacji triggera.

Parametr	Wartość
Input [-]	DAC OUT1
Speed/Power Mode	High Speed
Trigger Mode	Rising Edge Interrupt
Hysteresis Level	Low

Tabela 5: Konfiguracja komparatora

2.6 DAC1

Konfiguracja DAC używanego do implementacji triggera.

Parametr	Wartość
OUT1 mode	Connected to on chip-peripherals only
Output Buffer	Disable
Trigger	None

Tabela 6: Konfiguracja DAC

2.7 DMA

Używamy DMA do wysyłania danych do wyświetlacza, oraz zbierania próbek z ADC.

DMA request	Stream	Direction	Priority
SPI1_TX	DMA1_Channel3	Memory To Peripheral	Low
ADC1	DMA1_Channel1	Peripheral To Memory	Low
ADC2	DMA1_Channel2	Peripheral To Memory	Low

Tabela 7: Konfiguracja DMA

2.8 TIM1

Timer 1 został wykorzystany do obsługi enkodera inkrementalnego.

Parametr	Wartość
Prescaler	0
Counter Mode	Up
Counter Period	65535
Internal Clock Division	No Division
Repetition Counter	0
auto-reload preload	Disable
Encoder Mode	Encoder Mode TI1
Polarity	Rising Edge

Tabela 8: Konfiguracja TIM1

2.9 TIM4

Timer 4 został wykorzystany do wyzwalania pomiarów ADC gdy komparator ich nie wyzwala. Został skonfigurowany tak, żeby wyzwalał przerwanie co sekundę:

$$Update\_event = \frac{TIM\_CLK}{(PSC+1)(ARR+1)}Hz \tag{1}$$

TIM_CLK = 80MHz, PSC = 7999, ARR = 9999

$$Update\_event = \frac{80MHz}{(7999+1)*(9999+1)} = 1Hz \tag{2}$$

Parametr	Wartość
Prescaler	7999
Counter Mode	Up
Counter Period	9999
Internal Clock Division	No Division
auto-reload preload	Disable
Trigger Event Selection TRGO	Update Event

Tabela 9: Konfiguracja TIM4

2.10 USART2

Komunikacja przez USART jest wykorzystywana przy kalibracji dotyku wyświetlacza do wysłania m.in. nowych wartości kalibracyjnych.

Parametr	Wartość
Baud Rate	115200 Bits/s
Word Length	8 Bits (including parity)
Parity	None
Stop Bits	1

Tabela 10: Konfiguracja USART

3 Urządzenia zewnętrzne

3.1 Wyświetlacz dotykowy LCD 480x320px

Wyświetlacz oparty jest o sterownik ILI9488 [1]. Dotyk w wyświetlaczu obsługiwany jest przez sterownik XPT2046.

3.2 Enkoder obrotowy KY-040

Enkoder będzie wykorzystywany do takich funkcji jak:

- skalowanie wykresu,
- przesuwanie wykresu,
- ustawianie triggera,
- ustawianie kursorów,

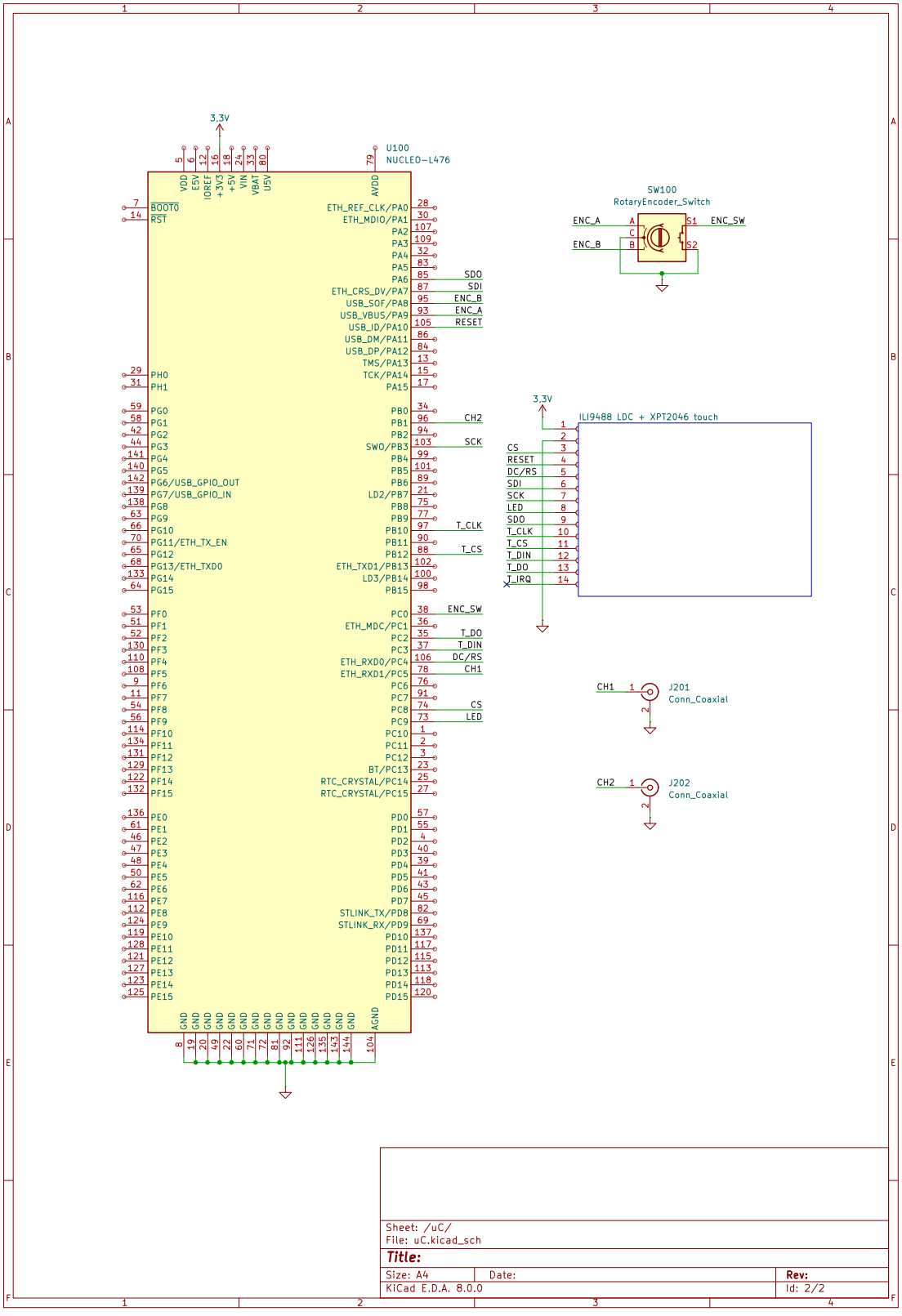
W celu obsługi tego peryferium TIMER 1 został skonfigurowany w trybie: "Combined Channels - Encoder Mode" [4]

Wyjścia DT oraz CLK zostały podpięte do wyjść timera 1 TIM1-CH1 oraz TIM1-CH2 co odpowiada pinom PA8 oraz PA9.

Odczyt impulsów z enkodera odbywa się poprzez odczyt wartości rejestru

$$htim1.Instance->CNT \tag{3}$$

4 Projekt elektroniki



5 Opis działania programu

5.1 Obsługa interfejsu

Obsługa większości funkcji oscyloskopu oparta jest o dotykowy wyświetlacz.
Dostępna funkcjonalność wyświetlacza:

- Po kliknięciu w wartość podzialki czasu na kratke, za pomocą enkodera można zmienić jej wartość,
- Po pierwszym kliknięciu w lewym dolnym rogu w odpowiedni kanał pojawia się menu gdzie po wyborze odpowiedniej opcji (skalowanie albo przesuwanie) można za pomocą enkodera wykonywać odpowiednie operacje.
- Kolejne kliknięcie w dany kanał powoduje wyświetlanie lub zaprzestanie wyświetlania wykresu.
- W sytuacji kliknięcia przycisku Menu, włącza się menu z dostępnymi opcjami: kursory oraz FFT.

5.2 Pomiary sygnałów z użyciem ADC

Jeden piksel na wyświetlaczu odpowiada jednej próbce z ADC, więc częstotliwość próbkowania została dobrana tak, aby przy 10 μs na działkę było zebranych 42 próbki (tyle pikseli ma jedna podziałka), więc powinna ona wynosić 4,2 MHz.

Częstotliwość próbkowania można wyliczyć ze wzoru (zakładając, że Clock Prescaler = ADC_CLOCK_ASYNC_DIV1):

$$f_{sampling} = \frac{\text{ADC\_clock}}{\text{Sampling time} + 12.5}$$

Dobrałem wartości: ADC_clock = 80 MHz, Sampling time = 6.5

$$f_{sampling} = \frac{80\text{ MHz}}{6.5 + 12.5} = 4.210526\text{ MHz}$$

Uzyskana częstotliwość odbiega od założonej, lecz rozbieżność ta jest pomijalnie mała. Jednakże lepszym rozwiązeniem prawdopodonie było by uniezależnienie częstotliwości próbkowania od ilości pikseli na działkę i skalowanie wyresu przez zmianę sposobu rysowania go.

Częstotliwość próbkowania ADC jest zmieniana w zależności od ustawionej podstawy czasu, dzięki czemu uzyskiwane jest skalowanie wykresu w dziedzinie czasu. Do wyboru jest 9 wartości podstawy czasu: 10, 20, 40, 80, 160, 320, 640, 1280 i 2560 μs. Zostało to zrealizowane w następujący sposób.

```
1  switch (oscilloscope.timeBase_us){
2      case 10:
3        hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
4        break;
5      case 20:
6        hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV2;
7        break;
8      ...
9      ...
10     ...
11     case 2560:
12       hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV256;
13       break;
14
15     // Initialize the ADC with the configured settings
16     if (HAL_ADC_Init(&hadc1) != HAL_OK){
17       // Initialization Error
18       Error_Handler();
19     }
20 }
```

Warto zwrócić uwagę, żę powyższy kod zmienia prescaler dla zarówno dla ADC1, jak i dla ADC2.

5.3 Implementacja triggera

Przetwornik DAC1 ustawia napięcie zadziałania triggera następującą funkcją:

```
HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, conv_voltage_to_DAC(
    oscilloscope.triggerLevel_mV/1000.0));
```

Napięcie to jest odbierane na odwracającym wejściu komparatora COMP1 i jest porównywane z napięciem na wejściu nieodwracającym, które jest podłączone na tym samym pinie, co wejście kanału 1 oscyloskopu. Na zboczu narastającym sygnału na kanale 1 oscyloskopu, gdy wartość dorównuje napięciu ustawionemu przez DAC, wyzwalane jest przerwanie startujące zbieranie próbek na obu kanałach.

```
void HAL_COMP_TriggerCallback(COMP_HandleTypeDef *hcomp){
  comparatorTriggeredFlag = 1;
  if(conv_done & done_drawing){
    conv_done = 0;
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*) oscilloscope.ch1.waveform_raw_adc,
        MEMORY_DEPTH);
    HAL_ADC_Start_DMA(&hadc2, (uint32_t*) oscilloscope.ch2.waveform_raw_adc,
        MEMORY_DEPTH);
    ready_to_draw = 1;
    done_drawing = 0;
  }
}
```

Aby zapewnić działanie pomiaru sygnału nie wyzwalającego triggera, został użyty timer generujący przerwanie co sekundę, który startuje ADC, jeśli flaga mówiąca o wyzwoleniu komparatora nie jest ustawiona, oraz gdy oscyloskop nie jest w trybie zatrzymania pomiaru.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if(htim == &htim4){
    if(!comparatorTriggeredFlag && !oscilloscope.stop){
      HAL_ADC_Start_DMA(&hadc1, (uint32_t*) oscilloscope.ch1.waveform_raw_adc ,
          MEMORY_DEPTH);
      HAL_ADC_Start_DMA(&hadc2, (uint32_t*) oscilloscope.ch2.waveform_raw_adc ,
          MEMORY_DEPTH);
      done_drawing = 0;
    }
  }
}
```

5.4 Rysowanie wykresu

Poniższy kod rysuje wykres sygnału na ekranie. Warto zwrócić uwagę na użycie zmiennej `offsetBetweenChannels`, która służy zniwelowaniu przesunięcia fazowego między dwoma kanałami, powstałemu przez nie idealnie jednoczesne startowanie ADC.

```
void draw_waveform(Oscilloscope_channel* ch, uint64_t timeBase_us, int offset, int stop){
  const uint offsetBetweenChannels = 360;
  for(int i = 0; i < CANVA_WIDTH; ++i){
    if(ch->number == 1)
      ch->waveform_display[i] = convertAdcToVoltage(ch->waveform_raw_adc[i+offset+(offsetBetweenChannels/timeBase_us)])*1000;
    else
      ch->waveform_display[i] = convertAdcToVoltage(ch->waveform_raw_adc[i+offset])*1000;
  }
  for(int i = 0; i < 420-1; ++i){
    int x0 = i;
    int x1 = i+1;
    int y0 = CANVA_MIDDLE_V - ch->y_offset - (ch->waveform_display[i]*42)/ch->y_scale_mV;
    int y1 = CANVA_MIDDLE_V - ch->y_offset - (ch->waveform_display[i+1]*42)/ch->y_scale_mV;
          \\ Ensure that the waveform fints on the canva
    if(y0 < 32)
      y0 = 32;
    if(y0 > 284)
      y0 = 284;
    if(y1 < 32)
      y1 = 32;
    if(y1 > 284)
      y1 = 284;
    drawLine(x0, y0, x1, y1, ch->color);
  }
}
```

5.5 Pomiar peak-to-peak

```
uint32_t calculate_peak_to_peak(uint32_t *waveform){
  uint32_t max=0, min;
  min = 50000;
  for(int i = 0; i < CANVA_WIDTH; ++i){
    if(waveform[i]<min){
      min=waveform[i];
    }
    if(waveform[i]>max){
      max=waveform[i];
    }
  }
  return max-min;
}
```

5.6 Pomiar RMS

```
uint32_t calculate_RMS(uint32_t *waveform) {
    uint64_t sum_of_squares = 0;
    for (int i = 0; i < CANVA_WIDTH; ++i) {
        sum_of_squares += waveform[i] * waveform[i];
    }
    float mean_of_squares = (float)sum_of_squares / CANVA_WIDTH;
    float rms = sqrtf(mean_of_squares);
    return rms;
}
```

5.7 Obsługa wyświetlacza

Aby zapewnić akceptowalne działanie wyświetlacza zmodyfikowałem bibliotekę `https://github.com/offpic/ILI9488-DMA-SPI-STM32-4-3.95-INCH-STM32F103-TOUCH`, aby używała bufora do rysowania na wyświetlaczu. Dzięki temu, obraz nie miga przy rysowaniu. Wadą takiego rozwiązania jest bardzo duże zużycie pamięci.

```
/**
 * @brief Buffer storing 4-bit color values for each LCD pixel.
 * Each byte represents color information for two adjacent pixels, utilizing 4
     bits per pixel
 */
static uint8_t image_buffer[ILI9488_TFTWIDTH*ILI9488_TFTHEIGHT/2 ] = {0};

/**
 * @brief Macro to retrieve the color value of a pixel from the image buffer.
 *
 * @param x The x-coordinate of the pixel.
 * @param y The y-coordinate of the pixel.
 * @return The 4-bit color value of the specified pixel.
 */
#define IMG_BUFF_GET(x, y)      (((x)%2)==0)?(image_buffer[((y)*240)+((x)/2)]>>4):(
    image_buffer[((y)*240)+((x)/2)]&0x0F)

/**
 * @brief Macro to set the color value of a pixel in the image buffer.
 *
 * @param x The x-coordinate of the pixel.
 * @param y The y-coordinate of the pixel.
 * @param c The 4-bit color value to set for the pixel.
 */
#define IMG_BUF_SET(x, y, c)   if(((x)%2)==0){image_buffer[((y)*(_width/2))+((x)/2)
    ]&=0x0F;image_buffer[((y)*(_width/2))+((x)/2)]|=((c)<<4);} else{image_buffer
    [((y)*(_width/2))+((x)/2)]&=0xF0;image_buffer[((y)*(_width/2))+((x)/2)]|=(c);}
```

5.8 Kursory

W programie zawarte sa dwa rodzaje kursorów do wykonywania pomiarów:

- kursory poziome w osi Y - do pomiaru amplitudy,
- kursory pionowe w osi X - do pomiaru czasu,

Interfejs kursorów jako otwierane menu składa się z wyboru kanału, wyboru między kursorem w osi X i Y, oraz wyświetlenie wyniku pomiaru. Za pomocą kursorów pionowych dodatkowo można zmierzyć częstotliwość.

Kursory te działają w następujący sposób (Na przykładzie kursorów pionowych):

- Ustawienie danego kursora odbywa się na podstawie odczytu z enkodera gdzie ilość impulsów przekłada się bezpośrednio na pozycje, czyli gdy ilość impulsów równa się 200, kursor rysowany jest

na pozycji 200-tego piksela w danej osi. Wartość ta dodawana jest do zmiennej globalnej, dla każdego z pojedynczych kursorów a następnie rejestr odpowiedzialny za ilość impulsów na enkoderze jest zerowany. Zmiana kursora odbywa się poprzez naciśniecie przycisku na enkoderze. Zaimplementowane jest również ograniczenie przesuwania kursorów aby nie wychodzily one poza kratki na wyswietlaczu.

- Obliczenie czasu lub amplitudy odbywa się za pomocą wzoru:

```
1        time_1 = (time_cursor_1*time_per_grid)/PX_PER_GRID_X;
2        time_2 = (time_cursor_2*time_per_grid)/PX_PER_GRID_X;
3
4        time   = time_2 - time_1;
```

Gdzie:

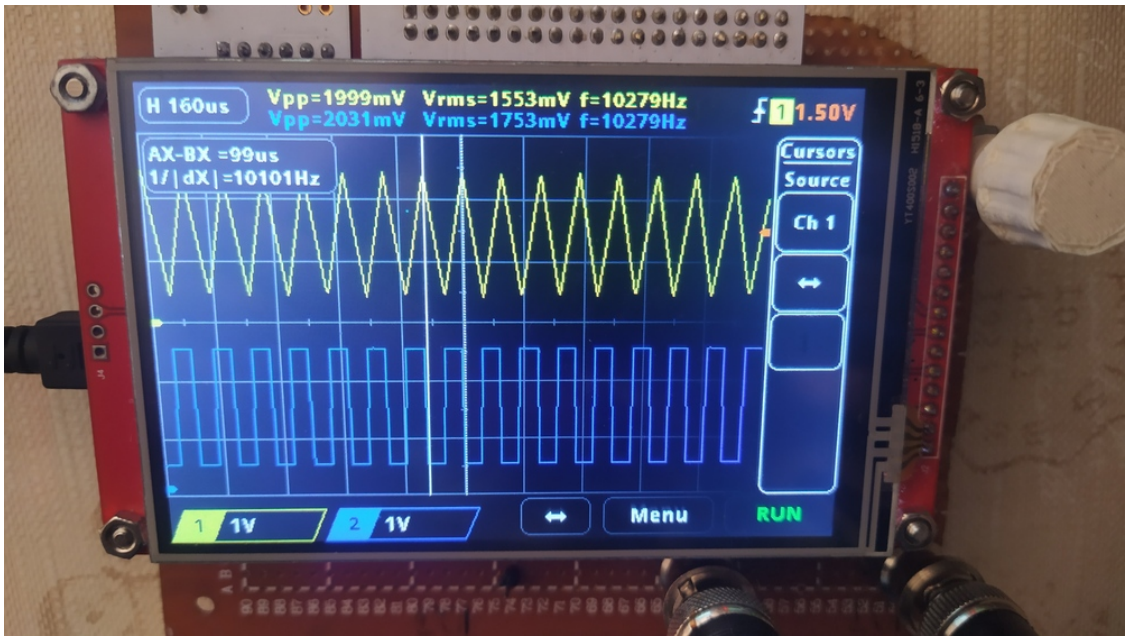
  - time 1 oraz time 2, zmienne w których przechowywany jest czas: "od zera do kursora"
  - time-cursor-1 oraz time-cursor-2, są to zmienne przechowujące pozycje na którym pikselu znajduje się aktualnie kursor,
  - time-per-grid, podziałka czas na kratke,
  - PX-PER-GRID, ilośc pikseli na kratke na siatce wyswietlacza, wyznaczany ze wzoru: rozmiar wyświetlacza(w pikselach) / ilość kratek,

Przykład: kursor 1 = 105, kursor 2 = 210, czas na kratke = 100ms, piksele na kratke = 42,

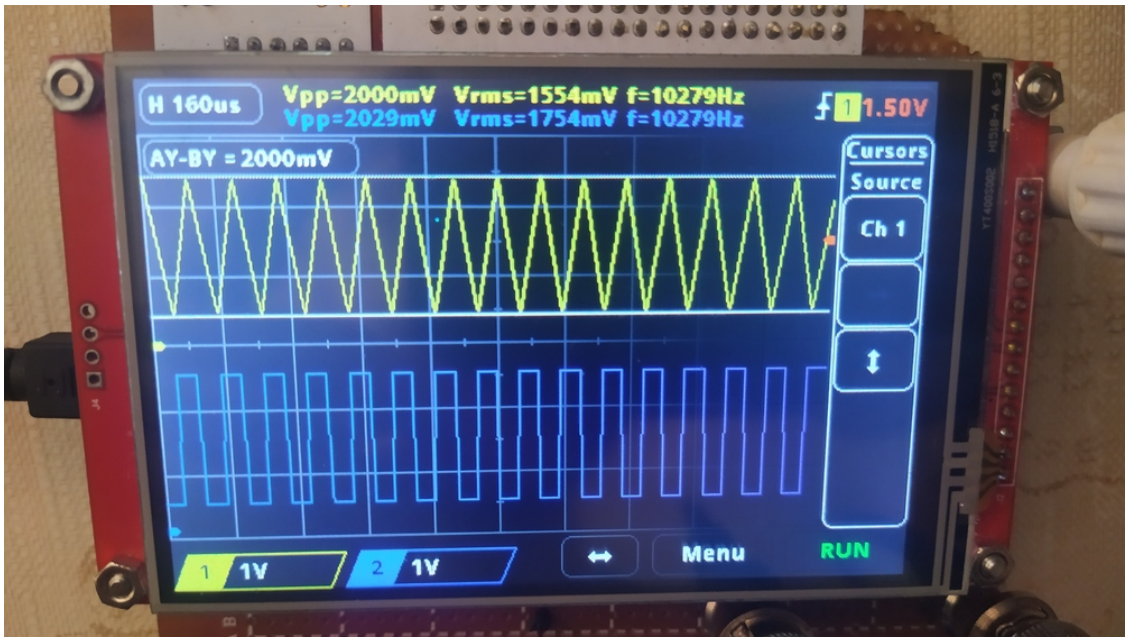
$$time1 = ((105 * 100)/42) = 250 \tag{4}$$

$$time2 = ((210 * 100)/42) = 500 \tag{5}$$

$$time = 500 - 250 = 250ms \tag{6}$$



Rysunek 6: Kursory w osi X



Rysunek 7: Kursory w osi Y

5.9 Szybka transformata Fouriera i pomiar częstotliwości

Pomiar częstotliwości zostal zaimplementowany przy użyciu transformaty FFT. Wyznaczana jest ona na podstawie widma. Szukana częstotliwosć sygnału jest tam gdzie słupek dla danej częstotliwości jest najwyższy (najwyższa amplituda).

Ze względu na pomiar częstotliwośći za pomocą FFT transformata FFT wykonywana jest z każdym przejściem pętli programu dla aktywnych kanałów.

Szybka Transformata Fouriera

- Użytkownik może włączyć FFT i wybrać kanał do analizy.
- Kod przetwarza dane sygnału z wybranego kanału za pomocą FFT, co pozwala na przekształcenie sygnału czasowego na widmo częstotliwościowe.
- Wyniki FFT są wyświetlane na ekranie oscyloskopu jako widmo częstotliwościowe, co pozwala użytkownikowi na analizę zawartości częstotliwościowej sygnału.

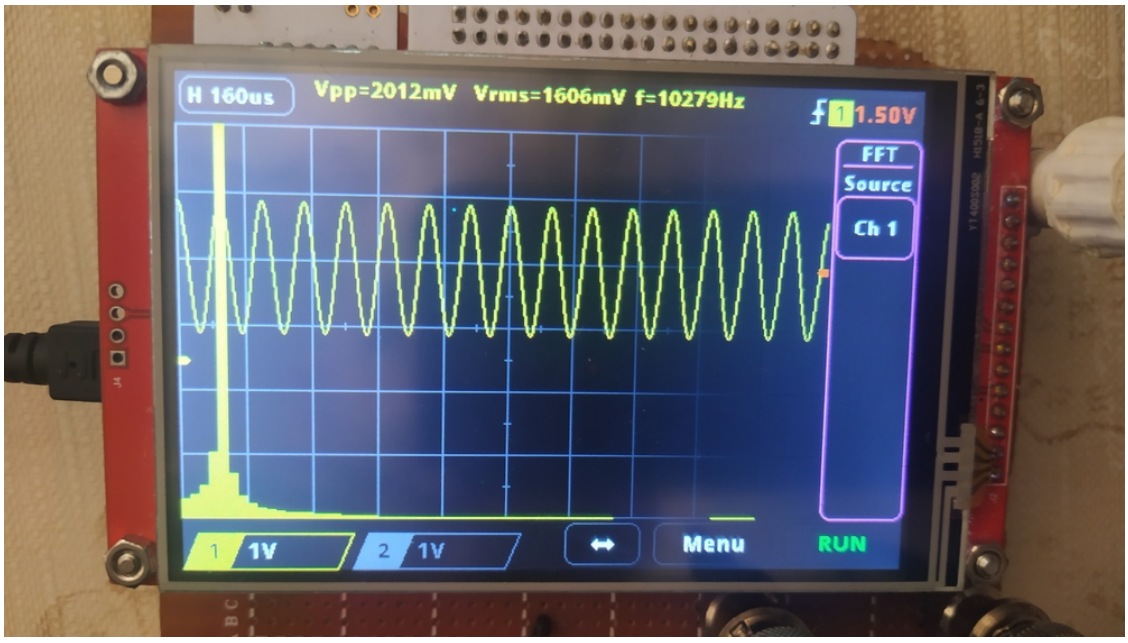
Wykonanie szybkiej transformaty Fouriera podzielone jest na dwie funkcje:
CalculateFFT():

```
1  void calculateFFT(Oscilloscope_channel * ch, uint32_t sampling_frequency){
2
3    Complex X[FFT_SIZE];
4    for (int i = 0; i < FFT_SIZE; i++) {
5      X[i].real = ch->waveform_display[i];
6      X[i].imag = 0;
7    }
8
9    fft(X, FFT_SIZE);
10
11   for (int i = 0; i < FFT_SIZE / 2; i++) {
12     ch->fft_amplitude[i] = sqrt(X[i].real * X[i].real + X[i].imag * X[i].imag);
13     ch->fft_frequency[i] = (i *sampling_frequency) / FFT_SIZE;
14
15   }
16
17   uint32_t amp_max = 0;
18   uint32_t freq = 0;
```

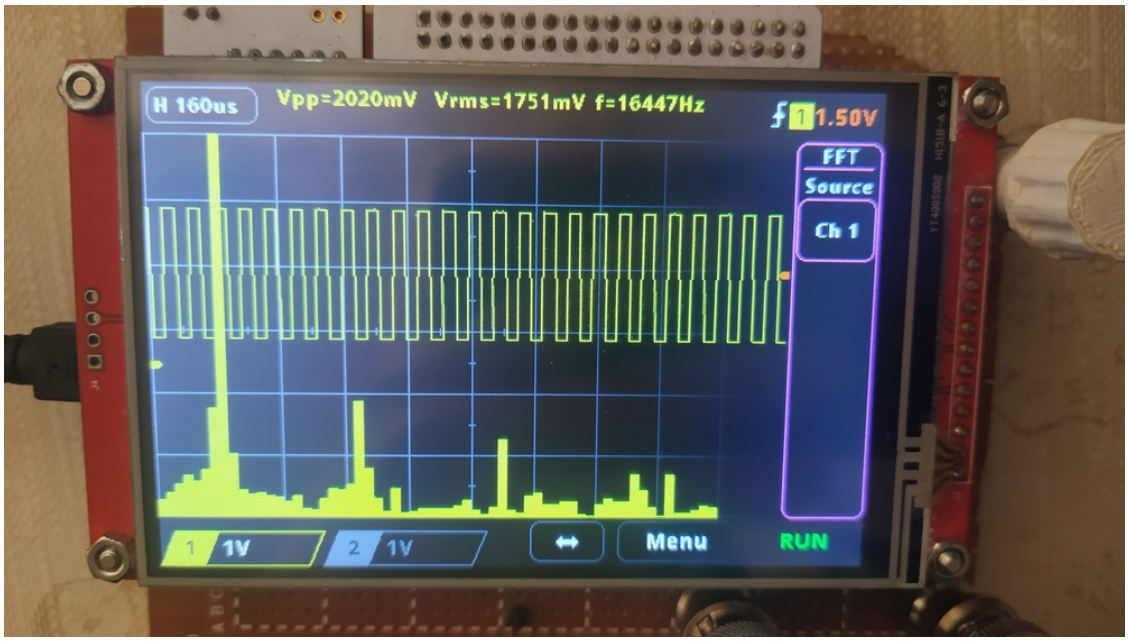
```
19
20    for(int i = 1; i < FFT_SIZE / 2; i++){
21      if(ch->fft_amplitude[i] > amp_max){
22        amp_max = ch->fft_amplitude[i];
23        freq = ch->fft_frequency[i];
24      }
25    }
26
27    ch->channel_frequency = (uint32_t)freq;
28  }
```

oraz **fft():**

```
1     void fft(Complex *X, int N) {
2     if (N <= 1) return;
3
4     Complex *X_even = (Complex *)malloc(N / 2 * sizeof(Complex));
5     Complex *X_odd = (Complex *)malloc(N / 2 * sizeof(Complex));
6     for (int i = 0; i < N / 2; i++) {
7         X_even[i] = X[i * 2];
8         X_odd[i] = X[i * 2 + 1];
9     }
10
11    fft(X_even, N / 2);
12    fft(X_odd, N / 2);
13
14    for (int k = 0; k < N / 2; k++) {
15        double t = -2 * M_PI * k / N;
16        Complex e = {cos(t), sin(t)};
17        Complex temp = {e.real * X_odd[k].real - e.imag * X_odd[k].imag, e.real *
              X_odd[k].imag + e.imag * X_odd[k].real};
18        X[k].real = X_even[k].real + temp.real;
19        X[k].imag = X_even[k].imag + temp.imag;
20        X[k + N / 2].real = X_even[k].real - temp.real;
21        X[k + N / 2].imag = X_even[k].imag - temp.imag;
22    }
23    free(X_even);
24    free(X_odd);
25  }
```



Rysunek 8: FFT dla sygnału sinusoidalnego

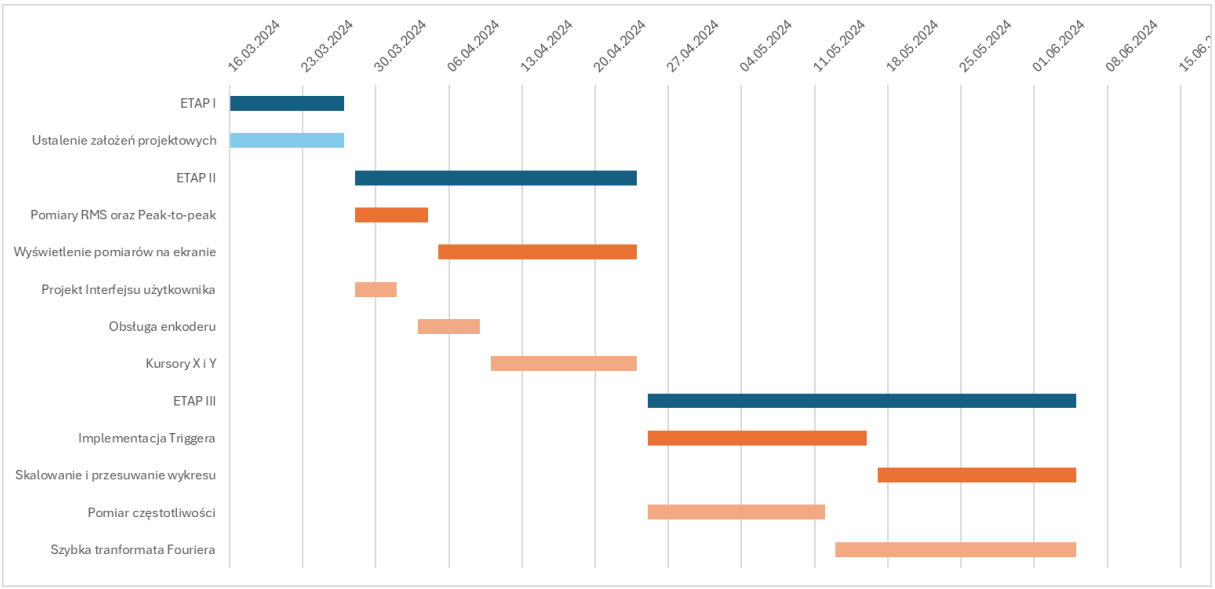


Rysunek 9: FFT dla sygnału prostokątnego

6 Harmonogram pracy

Kamienie milowe:

- uruchomienie wyświetlacza
- działający trigger
- szybka transformata Fouriera



Rysunek 10: Diagram Gantta

6.1 Podział pracy

Dominik Pluta	%	**Kamil Winnicki**	%
Pomiar RMS i peak-to-peak		Projekt interfejsu użytkownika	
Obsługa wyświetlacza		Obsługa Enkodera	
Odczytywanie wartości z ADC		Kursory w osi X	
Wyświetlanie wykresu zmierzonego sygnału		Kursory w osi Y	

Tabela 11: Podział pracy – Etap II

Dominik Pluta	%	**Kamil Winnicki**	%
Implementacja triggera		Pomiar częstotliwości	
Rozciąganie i przesuwanie wykresu w osi X i Y		Szybka transformata Fouriera	

Tabela 12: Podział pracy – Etap III

7 Podsumowanie

Wszystkie założenia zostały zrealizowane i stworzony projekt spełnia funkcję prostego oscyloskopu o zakresie napięć wejściowych od 0V do 3.3V i częstotliwości do 400kHz a wykonywane pomiary są względnie dokładne. Należy jednak wspomnieć że po przekroczeniu około 100kHz działanie oscyloskopu znacznie zwalnia, ponieważ mikrokontroler jest w dużej części czasu zajęty przez obsługę przerwania od komparatora. Można by rozwiązać ten problem wyłączając komparator na pewien czas, lecz zabrakło nam czasu aby dopracować ten aspekt konstrukcji.

W przyszłości oscyloskop mógłby zostać ulepszony o dodatkowe układy elektronicznme które pozwoliły by na zwiększenie zakresu amplitudy mierzonych sygnałów.

Należy zwrócić uwagę, że aktualny kod do obsługi wyświetlacza zajmuje większość pamięci mikrokontrolera, co utrudnia rozbudowę projektu i uniemożwliwiło nam zwiększenie rozdzielczości wykonywanego FFT.

8 Link do repozytorium projektu

`https://github.com/Dominik-Workshop/KD-23MTS`

Literatura

[1] hpinfotech. ILI9488 datasheet.

[2] J. Rydzewski. Oscyloskop elektroniczny. 1982.

[3] J. Rydzewski. Pomiary oscyloskopowe. 1999.

[4] ST. STM32L476R datasheet.