Politechnika Wrocławska

Sterowniki Robotów

Lab 04 - ADC, DAC i DMA

Wydział i kierunek	W12N, Automatyka i Robotyka
Autor i numery albumów	Michał Markuzel 275417
Termin zajęć	Środa 15:15-16:55
Kod grupy zajeciowej	W12AIR-SI0102G
Prowadzący	dr inż. Wojciech Domski

1 Wstęp

Niniejsze sprawozdanie przedstawia wyniki laboratorium dotyczącego obsługi przetworników analogowo-cyfrowych (ADC) oraz cyfrowo-analogowych (DAC) w mikrokontrolerze STM32.

W pierwszym zadaniu zbadano działanie przetwornika ADC, w drugim zadaniu połączono przetworniki ADC i DAC, a w zadaniu domowym zaimplementowano przetwornik DAC bazujący na architekturze drabinki rezystorowej R-2R.

2 Podstawy teoretyczne

2.1 Przetwornik analogowo-cyfrowy (ADC)

Przetwornik analogowo-cyfrowy jest układem elektronicznym, który zamienia sygnał analogowy (ciągły) na odpowiadający mu sygnał cyfrowy (dyskretny). W mikrokontrolerach STM32 stosowane są przetworniki o rozdzielczości 12 bitów, co daje 4096 możliwych poziomów sygnału.

Zasada działania przetwornika ADC w STM32 polega na próbkowaniu napięcia wejściowego i porównywaniu go z wewnętrznym napięciem referencyjnym (najczęściej 3.3V). Wynik konwersji jest wartością całkowitą z zakresu 0-4095, gdzie 0 odpowiada 0V, a 4095 odpowiada napięciu referencyjnemu 3.3V. Wartość napięcia można obliczyć według wzoru:

$V_{measured} = \frac{ADC\_value \cdot V_{ref}}{4095}$

2.2 Przetwornik cyfrowo-analogowy (DAC)

Przetwornik cyfrowo-analogowy wykonuje operację odwrotną do ADC - zamienia sygnał cyfrowy na odpowiadający mu sygnał analogowy. W mikrokontrolerach STM32 wbudowane przetworniki DAC również mają rozdzielczość 12 bitów.

Wartość cyfrową dla DAC można obliczyć na podstawie żądanego napięcia według wzoru:

$DAC\_value = \frac{V_{desired} \cdot 4095}{V_{ref}}$

2.3 Drabinka rezystorowa R-2R

Drabinka rezystorowa R-2R jest prostym układem elektronicznym, który może służyć jako przetwornik DAC. Składa się z rezystorów o dwóch wartościach: R oraz 2R, połączonych w charakterystyczny sposób. Każdy bit sygnału cyfrowego steruje odpowiednim węzłem drabinki, a napięcie wyjściowe jest sumą ważoną poszczególnych bitów.

W przypadku 4-bitowej drabinki R-2R (jak w zadaniu domowym), wartość napięcia wyjściowego można obliczyć jako:

$V_{out} = V_{ref} \cdot \frac{b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0}{2^n}$

gdzie b_i to wartość i-tego bitu (0 lub 1), a n to liczba bitów (w tym przypadku 4).

3 Zadanie 1 - ADC

3.1 Cel

Celem zadania było zaprogramowanie mikrokontrolera STM32 do cyklicznego pomiaru napięcia na pinie ADC4 (PC4) i wyświetlanie wyników na porcie szeregowym.

3.2 Opis realizacji

Do realizacji zadania wykorzystano przetwornik ADC1 mikrokontrolera, który został skonfigurowany do pracy w trybie przerwaniowym. Po każdej konwersji wywoływana jest funkcja callback **HAL_ADC_ConvCpltCallback()**, która odczytuje zmierzoną wartość i ustawia flagę informującą o zakończeniu pomiaru.

W głównej pętli programu, gdy flaga jest ustawiona, obliczane jest napięcie na podstawie surowej wartości ADC, a następnie wyniki są wysyłane przez UART w formacie:

```
nr_indeksu;surowy_odczyt_z_ADC;wartosc_w_woltach
```

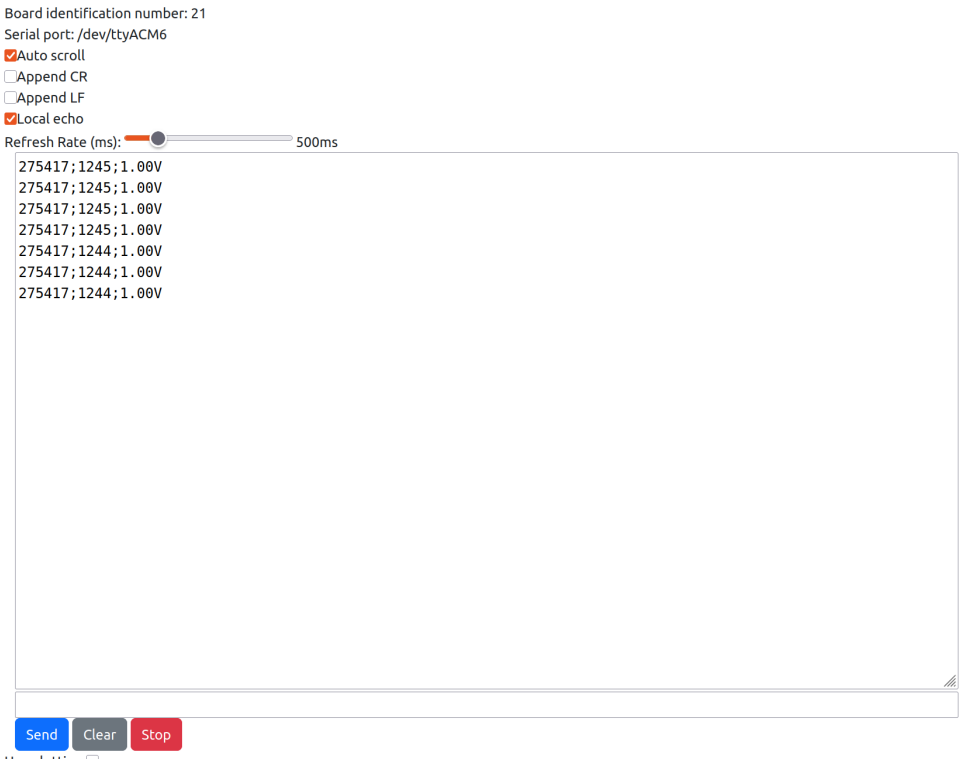
Po wysłaniu wyników, flaga jest zerowana i inicjowana jest kolejna konwersja ADC.

3.3 Kod programu

```
/* USER CODE BEGIN PV */
uint16_t adc_value = 0;
uint8_t adc_flag = 0;
/* USER CODE END PV */

// [...]

/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len)
{
  HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, HAL_MAX_DELAY);
  return len;
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_value = HAL_ADC_GetValue(hadc);
    adc_flag = 1;
  }
}
/* USER CODE END 0 */

// [...]

/* USER CODE BEGIN 2 */
  HAL_ADC_Start_IT(&hadc1);
/* USER CODE END 2 */

// [...]

/* USER CODE BEGIN WHILE */
  while (1)
  {

```

```
37      if (adc_flag == 1)
38      {
39        float voltage = (float)adc_value * (3.3f / 4095.0f);
40        char voltage_str[10];
41        snprintf(voltage_str, sizeof(voltage_str), "%.2f", voltage);
42        printf("275417;%d;%sV\r\n", adc_value, voltage_str);
43        adc_flag = 0;
44
45        HAL_ADC_Start_IT(&hadc1);
46      }
47
48      HAL_Delay(1000);
49
50      /* USER CODE END WHILE */
51
52  // [...]
```

3.4 Wynik



Rysunek 1: Output zadania 1 w serial port na remotelab

3.5 Wnioski

Realizacja zadania pozwoliła na zapoznanie się z podstawami obsługi przetwornika ADC w mikrokontrolerze STM32. Zastosowanie trybu przerwaniowego umożliwia efektywne wykorzystanie czasu procesora, gdyż nie ma potrzeby aktywnego oczekiwania na zakończenie konwersji.

Z wyników pomiarów widać, że wartości surowe ADC mieszczą się w zakresie 0-4095, co odpowiada napięciom od 0V do 3.3V.

4 Zadanie 2 - DAC

4.1 Cel

Celem zadania było zaprogramowanie mikrokontrolera STM32 do cyklicznego generowania określonych napięć (0.5V, 1.5V, 1.7V, 2.5V, 3.3V) na wyjściu DAC1 (PA4), a następnie wykonywanie pomiaru tego napięcia za pomocą ADC1 (PA6) i wyświetlanie wyników na porcie szeregowym.

4.2 Opis realizacji

W realizacji zadania wykorzystano zarówno przetwornik DAC1, jak i ADC1 mikrokontrolera. DAC został skonfigurowany do pracy w trybie normalnym, a ADC w trybie przerwaniowym.

Program cyklicznie zmienia wartość napięcia na wyjściu DAC zgodnie z zadaną sekwencją. Po wystawieniu napięcia następuje 1-sekundowe opóźnienie, a następnie wykonywany jest pomiar za pomocą ADC. Wyniki (zarówno surowe, jak i przeliczone na napięcie) są wysyłane przez UART w formacie:

```
nr_indeksu;surowa_warto  _DAC;napi cie_DAC;surowy_odczyt_z_ADC;
    wartosc_w_woltach_z_ADC
```

Do konwersji napięcia na wartość surową dla DAC oraz odwrotnie wykorzystano odpowiednie funkcje pomocnicze.

4.3 Konfiguracja

W konfiguracji sprzętowej wykorzystano:

- DAC1 na pinie PA4 - do generowania napięcia
- ADC1 na pinie PA6 - do pomiaru napięcia
- UART2 - do komunikacji z komputerem

DAC został skonfigurowany do pracy w 12-bitowym, co daje pełną rozdzielczość 4096 poziomów wyjściowych.

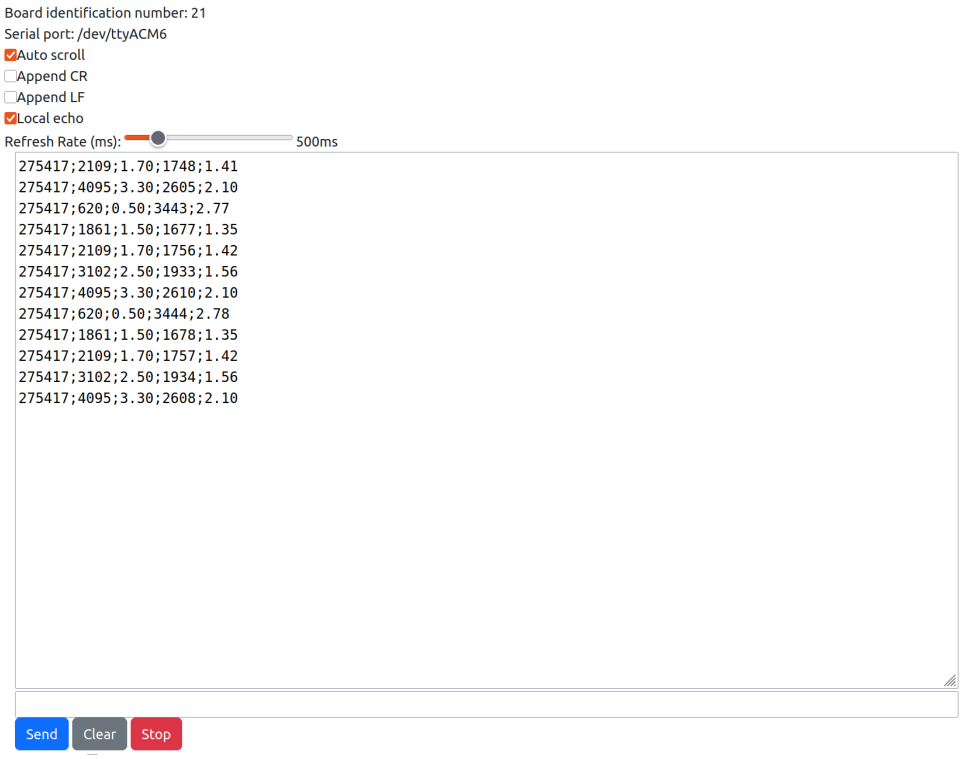
4.4 Kod programu

```
/* USER CODE BEGIN PV */
uint16_t adc_value = 0;
uint8_t adc_flag = 0;
uint16_t dac_value = 0;
uint8_t voltage_index = 0;

// Zadane napi cia w woltach
const float target_voltages[] = {0.5f, 1.5f, 1.7f, 2.5f, 3.3f};
const uint8_t num_voltages = sizeof(target_voltages) / sizeof(target_voltages
    [0]);
/* USER CODE END PV */

// [...]

/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len)
{
```

```
  HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, HAL_MAX_DELAY);
  return len;
}

// Konwersja napi cia na warto    surow  dla DAC
uint16_t voltage_to_dac(float voltage)
{
  return (uint16_t)((voltage * 4095.0f) / 3.3f);
}

// Konwersja warto ci surowej DAC na napi cie
float dac_to_voltage(uint16_t dac_val)
{
  return (float)dac_val * (3.3f / 4095.0f);
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_value = HAL_ADC_GetValue(hadc);
    adc_flag = 1;
  }
}
/* USER CODE END 0 */

// [...]

  /* USER CODE BEGIN 2 */
  dac_value = voltage_to_dac(target_voltages[0]);
  HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dac_value);
  HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);

  HAL_ADC_Start_IT(&hadc1);
  /* USER CODE END 2 */

// [...]

/* USER CODE BEGIN WHILE */
  while (1)
  {

    if (adc_flag == 1)
    {
      // Obliczanie napi cia
      float dac_voltage = target_voltages[voltage_index];
      float adc_voltage = (float)adc_value * (3.3f / 4095.0f);

      // Format warto ci float
      char voltage_str[20];
      snprintf(voltage_str, sizeof(voltage_str), "%.2f", dac_voltage);
      char adc_voltage_str[20];
      snprintf(adc_voltage_str, sizeof(adc_voltage_str), "%.2f", adc_voltage);

      // Format: nr indeksu;surow  warto   DAC;DAC wyra one w Voltach;surowa
     warto   ADC;pomiar ADC wyra ony w Voltach
      printf("275417;%d;%s;%d;%s\r\n", dac_value, voltage_str, adc_value,
    adc_voltage_str);

      // Nast pna warto   napi cia
      voltage_index = (voltage_index + 1) % num_voltages;
      dac_value = voltage_to_dac(target_voltages[voltage_index]);
      HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dac_value);
```

```
78
79          adc_flag = 0;
80          HAL_ADC_Start_IT(&hadc1);
81        }
82
83        HAL_Delay(1000);
84
85        /* USER CODE END WHILE */
86
87  // [...]
```

4.5 Wynik



Rysunek 2: Output zadania 2 w serial port na remotelab

4.6 Wnioski

Realizacja zadania pozwoliła na kompleksowe zapoznanie się z obsługą przetworników ADC i DAC w mikrokontrolerze STM32 oraz na praktyczne zastosowanie wiedzy o konwersji sygnałów analogowych na cyfrowe i odwrotnie.

Z analizy otrzymanych wyników można zaobserwować pewne różnice między napięciem zadanym (DAC) a zmierzonym (ADC). Różnice te mogą wynikać z kilku czynników:

- Niedokładności przetworników DAC i ADC
- Szumów w układzie pomiarowym
- Spadków napięcia na połączeniach między wyjściem DAC a wejściem ADC
- Błędów zaokrągleń w obliczeniach

5 Zadanie Domowe

5.1 Cel

Celem zadania domowego było zaprogramowanie mikrokontrolera STM32 do realizacji przetwornika DAC bazującego na architekturze drabinki rezystorowej R-2R. Program miał generować malejącą sekwencję wartości od 15 do 0, a następnie powtarzać ją. Co 1 sekundę wykonywany był pomiar za pomocą ADC6, a wyniki były wyświetlane na porcie szeregowym.

5.2 Opis realizacji

W realizacji zadania domowego nie korzystano z wbudowanego przetwornika DAC, lecz zaimplementowano przetwornik DAC za pomocą zewnętrznej drabinki rezystorowej R-2R. Do sterowania drabinką wykorzystano 4 piny GPIO (PB2, PB4, PB5, PB13), które odpowiadały 4 bitom wartości cyfrowej (od 0 do 15).

Program cyklicznie zmienia wartość cyfrową od 15 do 0, a następnie wraca do 15. Po wystawieniu wartości na pinach GPIO, wykonywany jest pomiar napięcia za pomocą ADC. Wyniki są wysyłane przez UART w formacie:

```
nr_indeksu;surowa_warto  _DAC;wartosc_DAC_w_woltach;surowy_pomiar_ADC;
    napiecie_ADC_w_woltach
```

5.3 Konfiguracja

W konfiguracji sprzętowej wykorzystano:

- 4 piny GPIO (PB2, PB4, PB5, PB13) - do sterowania drabinką R-2R
- ADC1 na kanale 6 - do pomiaru napięcia wyjściowego z drabinki R-2R

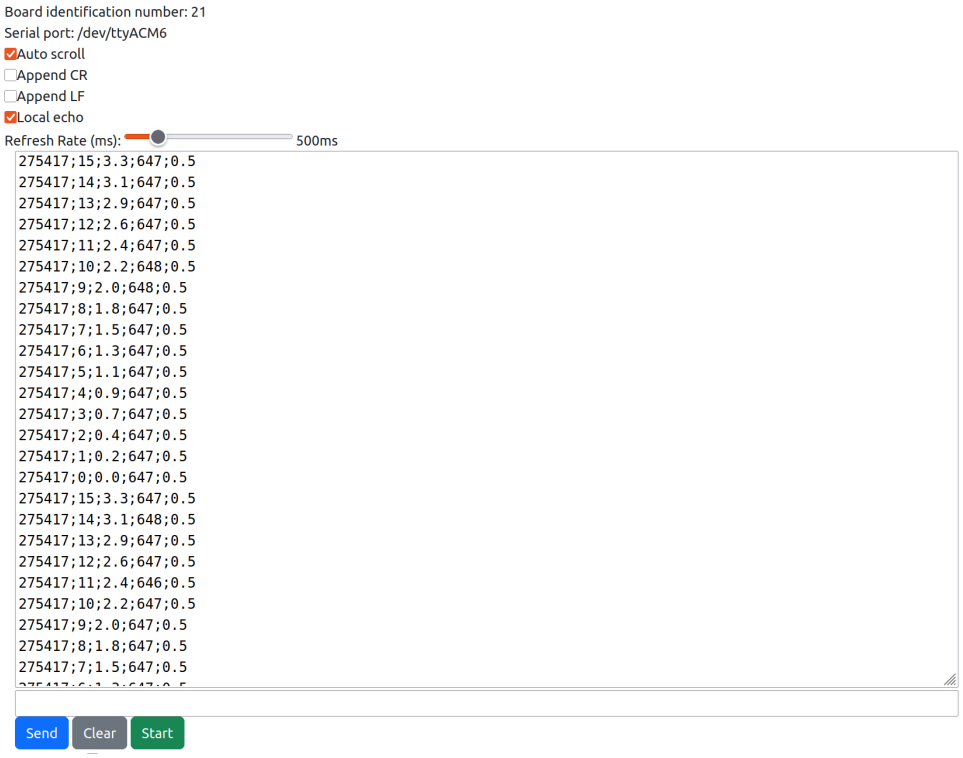
Zewnętrzna drabinka R-2R zbudowana jest z rezystorów o wartościach R i 2R, połączonych w charakterystyczny sposób tworzący przetwornik DAC o rozdzielczości 4 bity (16 poziomów napięcia).

5.4 Kod programu

```c
/* USER CODE BEGIN PV */
uint8_t dac_value = 15;                // Startowa warto    dla DAC (od 15 do 0)
uint16_t adc_value = 0;                // Pomiar z ADC
volatile uint8_t adc_flag = 0;         // Flaga zako czenia konwersji ADC
const uint32_t INDEX_NUMBER = 275417; // Nr indeksu
/* USER CODE END PV */

// [...]

/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len)
{
  HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, HAL_MAX_DELAY);
  return len;
}

// Callback po zako czeniu konwersji ADC
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
```

```
  if (hadc->Instance == ADC1)
  {
    adc_value = HAL_ADC_GetValue(hadc);
    adc_flag = 1;
  }
}
/* USER CODE END 0 */

// [...]

/* USER CODE BEGIN 2 */
  HAL_ADC_Start_IT(&hadc1);
/* USER CODE END 2 */

// [...]

  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if (adc_flag == 1)
    {
      // Obliczenie napi
      float dac_voltage = (float)dac_value * (3.3f / 15.0f);   // Skalowanie
    0-15 do 0-3.3V
      float adc_voltage = (float)adc_value * (3.3f / 4095.0f); // Skalowanie
    0-4095 do 0-3.3V

      // Wypisanie sformatowanych danych
      printf("%lu;%d;%.1f;%d;%.1f\r\n",
             INDEX_NUMBER, dac_value, dac_voltage, adc_value, adc_voltage);

      // Zmniejszenie warto ci DAC i powr t do 15 gdy osi gnie 0
      if (dac_value == 0)
        dac_value = 15;
      else
        dac_value--;

      // Ustawienie pin w GPIO dla 4-bitowej warto ci DAC
      // Ustawienie odpowiednich pin w PB2, PB4, PB5, PB13
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, (dac_value & 0x01) ? GPIO_PIN_SET :
    GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, (dac_value & 0x02) ? GPIO_PIN_SET :
    GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, (dac_value & 0x04) ? GPIO_PIN_SET :
    GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, (dac_value & 0x08) ? GPIO_PIN_SET :
    GPIO_PIN_RESET);

      // Reset flagi i uruchomienie kolejnej konwersji ADC
      adc_flag = 0;
      HAL_ADC_Start_IT(&hadc1);
    }

    // Odczekanie 1 sekundy mi dzy pomiarami
    HAL_Delay(1000);

    /* USER CODE END WHILE */

// [...]
```

5.5 Wynik



Rysunek 3: Output zadania domowego w serial port na remotelab

5.6 Wnioski

Realizacja zadania domowego pozwoliła na praktyczne zapoznanie się z alternatywną metodą implementacji przetwornika DAC, jaką jest drabinka rezystorowa R-2R.

6 Podsumowanie

Laboratorium pozwoliło na praktyczne zapoznanie się z obsługą przetworników analogowo-cyfrowych i cyfrowo-analogowych w mikrokontrolerze STM32. Zadania numer 1 i 2 zostały zrealizowane, a uzyskane wyniki potwierdzają poprawność implementacji.