République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar
École Nationale d'Ingénieurs de Tunis

Département Génie Electrique

Mini projet microcontrôleurs avancés

Réalisé par :

Abdelkerim EL BANI

Fedi MEZGHANI

Classe : 2AGE1

Encadré par :

Khaled JELASSI

Année universitaire 2024/2025

Résumé

Notre projet consiste à développer un programme en C embarqué qui sera chargé sur la carte électronique programmable STM32F411VE et permettra de générer trois signaux MLI à rapports cycliques sinusoïdaux tout en réalisant un décalage de $2\pi 3$.

Trois étapes sont nécessaires pour réaliser ce projet :

- Générer trois signaux MLI.
- Varier les rapports cycliques sinusoïdalement.
- Effectuer un déphasage de $2\pi/3$ entre les trois sigaux.

Mots clés : Microcontrôleur, STM32F407 ,Programmation embarquée, C embarqué,Génération de signaux

Table des matières

Table des figures

Chapitre 1

Elaboration du projet

1.1 Requis théoriques

1.1.1 Signal MLI ou PWM

Le signal PWM (MLI, Modulation de largeur d'impulsion) est un signal de fréquence constante et de rapport cyclique variable.

1.1.2 Utilisation du signal MLI

La génération d'un signal par PWM est particulièrement avantageux du point de vue de la consommation de la commande. C'est aussi un signal facile à générer par les micro-contrôleurs .

1.1.3 Valeur moyenne d'un signal MLI

La variation de la valeur du rapport cyclique engendre la variation de la valeur moyenne du signal

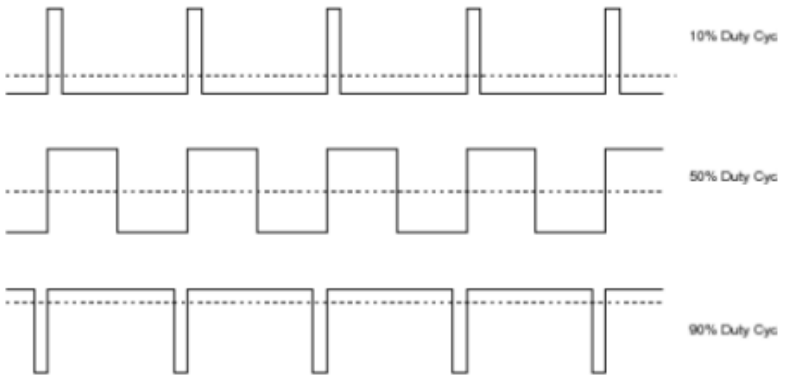

FIGURE 1.1 – Signal MLI

En retirant la fréquence porteuse F=1/T avec un filtre passe bas, il reste la valeur moyenne du signal.

Le fait de varier le rapport cyclique sinusoïdalement engendre une valeur moyenne sinusoïdale.

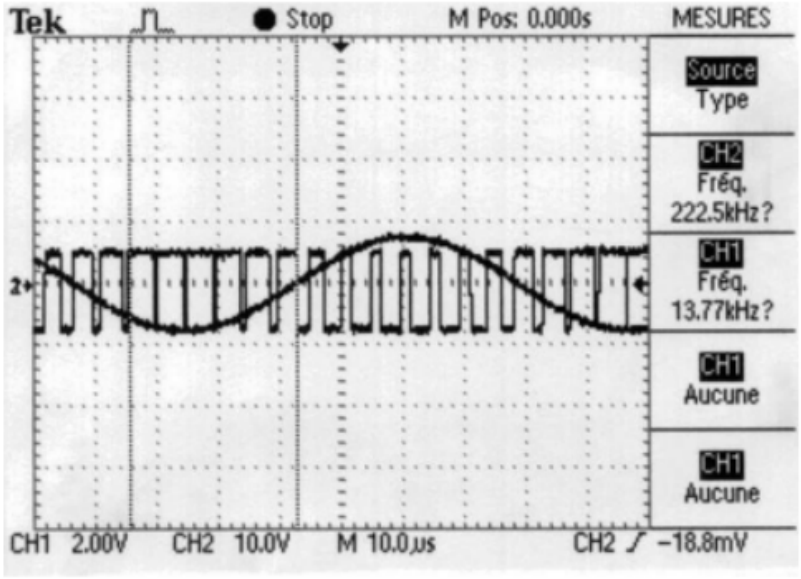

FIGURE 1.2 – Valeur moyenne sinusoïdale

1.2 Les TIMERs

1.2.1 Architecture et composition d'un TIMER

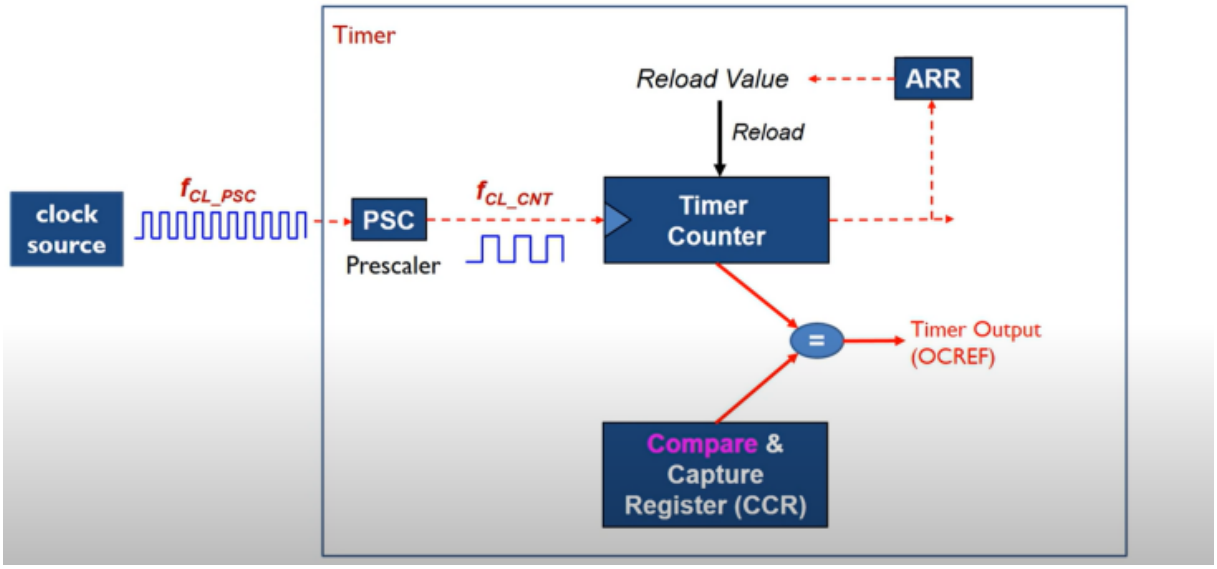

FIGURE 1.3 – Architecture du timer

1.2.2 Fonctionnement en mode MLI

-Le signal de l'horloge (clock source) entre au Prescaler
-Le Prescaler modifie et diminue le signal de l'horloge
-Le Timer sera en mode compteur
-Ce signal est comparé à une valeur choisie CCR
-Un signal modulé est généré OCREF

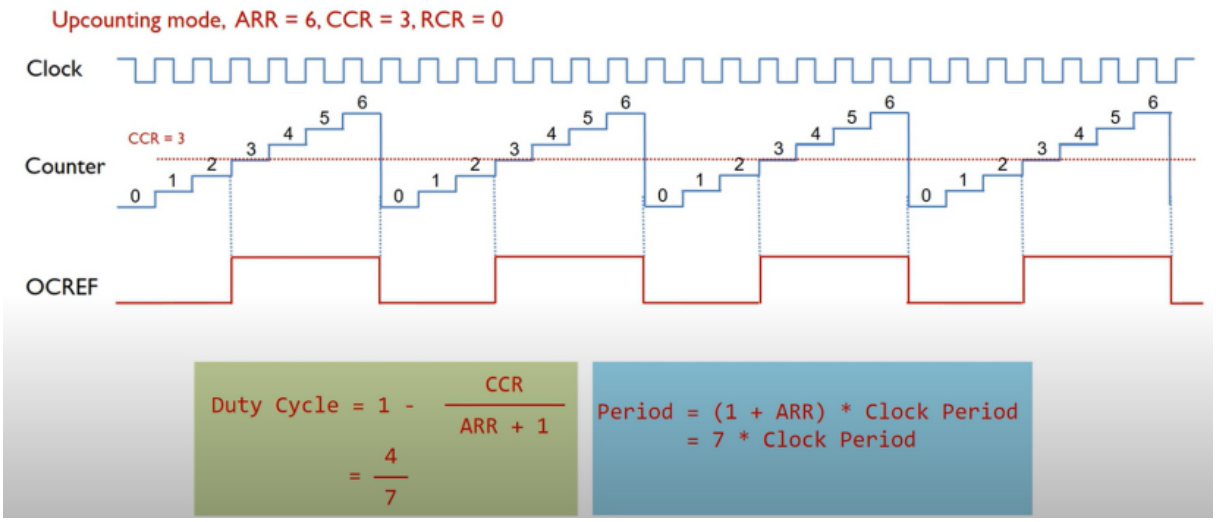

FIGURE 1.4 – Duty cycle and Period

OCREF :
-LOW si counter< CCR
-HIGH si counter >= CRR

1.2.3 Les formules

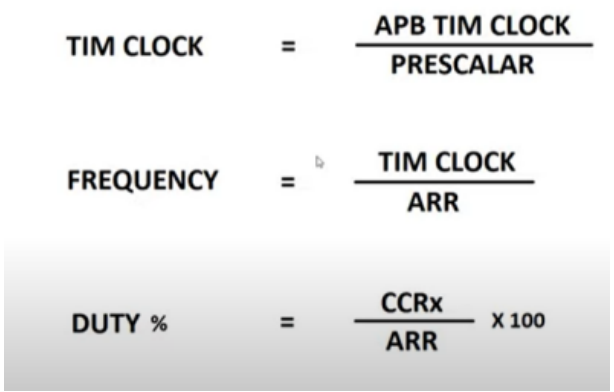

FIGURE 1.5 – Formules

Chapitre 2

Fontionnement generale

2.1 Fonctionnalités principales

Géneration du signal MLI

- Trois temporisateurs (TIM1, TIM2 et TIM4) sont configurés pour générer des signaux PWM sur différents canaux.

```
void TIM1_Init(void) {
    TIM_OC_InitTypeDef sConfigOC = {0};

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 16;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 1000;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    HAL_TIM_PWM_Init(&htim1);

    sConfigOC.OCMode = TIM_OCMODE_PWM1;
    sConfigOC.Pulse = 0;  // Initial duty cycle
    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1);
}
```

FIGURE 2.1 – Configuration du timer 1

On a initialisé le prescaler à 16 et la période à 1KHz pour obtenir une fréquence du signal de sortie égale à 5KHz. (L'horloge dans notre system fonctionne a 84 MHz)

$$f_{\text{output}} = \frac{f_{\text{timer}}}{\text{Prescaler} \times \text{Period}} \approx 5\,\text{kHz}$$

- Les signaux PWM sont utilisés pour piloter des broches de sortie, avec un cycle de service ajusté dynamiquement en fonction d'une entrée analogique (ADC) et d'une modulation sinusoïdale.
- Les signaux présentent un déphasage obtenu en définissant des valeurs initiales du compteur différentes pour chaque temporisateur.

Conversion analogique-numérique(ADC)

- L'ADC est configuré pour lire un potentiomètre connecté au canal 0.

```
void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
```

FIGURE 2.2 – Configuration du ADC 1

- La valeur lue par l'ADC est utilisée pour calculer un pourcentage, qui représente le cycle utile du signal PWM.

```
while (1) {

    // ADC conversion
    HAL_ADC_Start(&hadc1);
    if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {

        // Read ADC value
        adcValue = HAL_ADC_GetValue(&hadc1);

        percentage = (adcValue * 100) / 4095;
```

FIGURE 2.3 – Conversion ADC et mapping du pourcentage

Écran LCD

- Un écran LCD 16x2 caractères est utilisé pour afficher le pourcentage actuel du cycle utile. Pour cela, on a ultilisé la bibliotheque ”lcd16x2” qui offre des fonctions prédéfinis comme l'affichage des chaines de caractères.
- L'écran LCD est initialisé et utilisé pour afficher des animations de démarrage, des détails sur le projet et des mises à jour en direct du cycle utile.

```
lcd16x2_clear();
lcd16x2_setCursor(0, 2);
lcd16x2_printf("Duty Cycle:");
lcd16x2_setCursor(1, 6);
lcd16x2_printf("%u%%", percentage);
```

FIGURE 2.4 – Affichage du rapport cyclique sur le LCD

Animation de démarrage

- Au démarrage, l'écran LCD affiche une séquence d'animations présentant les crédits et les détails du projet.

```
// Startup Animation
for (int i = 0; i < 16; i++)
{
  lcd16x2_setCursor(0, i);
  lcd16x2_printf("\xFF");
  lcd16x2_setCursor(1, i);
  lcd16x2_printf("\xFF");
  HAL_Delay(50);
}

lcd16x2_clear();
lcd16x2_setCursor(0, 3);
lcd16x2_printf("Project by:");
HAL_Delay(1000);

lcd16x2_clear();
lcd16x2_setCursor(0, 3);
lcd16x2_printf("Abdelkerim");
lcd16x2_setCursor(1, 5);
lcd16x2_printf("El Bani");
HAL_Delay(1000);

lcd16x2_clear();
lcd16x2_setCursor(0, 8);
lcd16x2_printf("&");
lcd16x2_setCursor(1, 2);
lcd16x2_printf("Fedi Mezghani");
HAL_Delay(1000);
```

FIGURE 2.5 – Affichage d'une animation et des détails

2.2 Fonctionnement

Initialisation

- L'horloge système est configurée pour des performances optimales.

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    // Enable the power controller clock
    __HAL_RCC_PWR_CLK_ENABLE();

    // Set voltage scaling to optimize power consumption
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

    // Configure the HSE Oscillator and PLL
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON; // Use external oscillator
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON; // Enable PLL
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE; // Set HSE as PLL source
    RCC_OscInitStruct.PLL.PLLM = 8;  // HSE / PLLM = 1 MHz
    RCC_OscInitStruct.PLL.PLLN = 336; // 1 MHz * PLLN = 336 MHz
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4; // 336 MHz / PLLP = 84 MHz (System Clock)
    RCC_OscInitStruct.PLL.PLLQ = 7; // For USB (48 MHz = 336 MHz / 7)
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    // Configure the AHB and APB buses
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK |
                                  RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK; // Use PLL as system clock
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1; // AHB = 84 MHz
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2; // APB1 = 42 MHz
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1; // APB2 = 84 MHz
```

FIGURE 2.6 – Configuration de l'horloge du systeme

- Les pins GPIO sont configurées pour la sortie MLI et le contrôle de l'écran LCD.
- Les temporisateurs sont initialisés pour générer des signaux MLI avec une fréquence déterminée par la période du temporisateur.

La boucle principlae

Dans la boucle principale :
- La fonction HAL-ADC-Start(hadc1) initialise une conversion analogique-numérique.
- La fonction HAL-ADC-PollForConversion attend que la conversion soit terminée ou que le délai soit dépassé.
- La fonction HAL-ADC-GetValue récupère la valeur numérique issue de la conversion (entre 0 et 4095 en résolution 12 bits).
- la valeur calculée du pourcentage est affiché sur l'écran LCD.
- Le pourcentage est transformé en une valeur pour le PWM, proportionnelle à la période du timer.

- Les valeurs calculées sont appliquées aux registres de comparaison des timers via –HAL-TIM-SET-COMPARE pour ajuster le cycle de travail des signaux PWM.

```
while (1) {

    // ADC conversion
    HAL_ADC_Start(&hadc1);
    if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {

        // Read ADC value
        adcValue = HAL_ADC_GetValue(&hadc1);

        percentage = (adcValue * 100) / 4095;

        lcd16x2_clear();
        lcd16x2_setCursor(0, 2);
        lcd16x2_printf("Duty Cycle:");
        lcd16x2_setCursor(1, 6);
        lcd16x2_printf("%u%%", percentage);

        dutyCycle = (uint16_t)((percentage * htim1.Init.Period) / 100);

        // Set the duty cycle for PWM pins
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, dutyCycle);
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, dutyCycle);
        __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, dutyCycle);

        HAL_Delay(100);
    }
}
```

FIGURE 2.7 – Boucle principle du system