STM32:

```
#include "main.h"

ADC_HandleTypeDef hadc1;
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();

  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // Motor PWM
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3); // Buzzer PWM
  HAL_ADC_Start(&hadc1);

  // Threshold
  const uint16_t THRESH_LOW = 1500;
```

```
const uint16_t THRESH_MID = 3000;


while (1)
{
   HAL_ADC_Start(&hadc1);
   HAL_ADC_PollForConversion(&hadc1, 10);
   uint32_t adc_val = HAL_ADC_GetValue(&hadc1);

   if (adc_val < THRESH_LOW)
   {
      // Motor 10% duty cycle
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 6553); // 10% dari 65535
      // Buzzer bunyi frekuensi rendah (sekitar 1kHz)
      __HAL_TIM_SET_AUTORELOAD(&htim2, 15999); // Untuk 1kHz
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 8000); // 50% duty cycle
   }
   else if (adc_val > THRESH_MID)
   {
      // Motor 90% duty cycle
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 58981); // 90% dari 65535
      // Buzzer mati
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);
   }
   else
   {
      // Jika di antara 1500–3000, motor dan buzzer mati
      __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);
```

```
      }

      HAL_Delay(10);
    }
}

// ==== Berikut bagian inisialisasi (tidak banyak berubah) ====

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

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
      Error_Handler();
    }

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
                                | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
```

```
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;

  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
```

```
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;

  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_TIM1_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 0;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
```

```
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
   Error_Handler();
}

sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;

if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
   Error_Handler();
}

sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;

if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
   Error_Handler();
}

sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
```

```
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;

  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_TIM_MspPostInit(&htim1);
}

static void MX_TIM2_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 65535;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;

  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
```

```
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;

  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;

  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_TIM_MspPostInit(&htim2);
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    // PB0 sebagai input (tidak dipakai di sini tapi diinisialisasi)
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

void Error_Handler(void)
{
    __disable_irq();
    while (1)
    {
    }
}
```

RASBERRY:

```
from machine import I2C, Pin, UART
import utime
from pico_i2c_lcd import I2cLcd
# Inisialisasi UART
uart = UART(0, baudrate=9600, tx=Pin(0), rx=Pin(1))
```

```
# Inisialisasi LCD I2C
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=400000)
I2C_ADDR = 0x27 # Ganti dengan alamat LCD Anda
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
# Tunggu LCD siap
utime.sleep_ms(100)
lcd.clear()
lcd.putstr("Menunggu input...")
def process_uart_data(data):
 try:
 decoded = data.decode('utf-8').strip()
 lcd.clear()
 if decoded == "MERAH":
 lcd.putstr("Warna: Merah")
 elif decoded == "HIJAU":

lcd.putstr("Warna: Hijau")
 elif decoded == "BIRU":
 lcd.putstr("Warna: Biru")
 else:
 lcd.putstr(f"Data: {decoded}")
 except Exception as e:
 lcd.clear()
 lcd.putstr(f"Error: {str(e)}")
while True:
 if uart.any():
```

```
data = uart.readline()
if data:
process_uart_data(data)
utime.sleep_ms(100) # Beri sedikit jeda
```