c++ code to output PWM for acceleration and deceleration.

To generate a PWM signal for acceleration and deceleration in C++, we can use hardware-specific libraries that interface with hardware timers and GPIO (General Purpose Input/Output) pins. The specific implementation can vary depending on the platform (e.g., Arduino, Raspberry Pi). Below, I will show two examples:

1. **Arduino**: For microcontrollers like Arduino.
2. **Raspberry Pi**: Using the wiringPi library.

**Example 1: Arduino**

**Setup:**

1. Arduino IDE installed.
2. An Arduino board (e.g., Arduino Uno).

**Code:**

const int pwmPin = 9; // PWM output pin on Arduino

int pwmValue = 0; // Initial PWM value

void setup() {

pinMode(pwmPin, OUTPUT);

}

void loop() {

// Acceleration phase

for (pwmValue = 0; pwmValue <= 255; pwmValue++) {

analogWrite(pwmPin, pwmValue);

delay(10); // Adjust delay for acceleration rate

}

// Full speed

delay(1000); // Run at full speed for 1 second

// Deceleration phase

for (pwmValue = 255; pwmValue >= 0; pwmValue--) {

analogWrite(pwmPin, pwmValue);

delay(10); // Adjust delay for deceleration rate

}

// Stop

delay(1000); // Stop for 1 second

}

C++

Copy code

**Example 2: Raspberry Pi Using wiringPi Library**

**Setup:**

1. Raspberry Pi running Raspbian.
2. wiringPi library installed.

**Installation of wiringPi:**

sudo apt-get update

sudo apt-get install wiringpi

Shell

Copy code

**Code:**

#include <wiringPi.h>

const int pwmPin = 1; // GPIO18 (Physical pin 12) in PWM Mode

const int pwmRange = 1024; // Range for PWM

void setup() {

wiringPiSetup();

pinMode(pwmPin, PWM\_OUTPUT);

pwmSetMode(PWM\_MODE\_MS);

pwmSetRange(pwmRange);

pwmSetClock(384); // Adjust to set the PWM frequency

}

void loop() {

// Acceleration phase

for (int pwmValue = 0; pwmValue <= pwmRange; pwmValue++) {

pwmWrite(pwmPin, pwmValue);

delay(10); // Adjust delay for acceleration rate

}

// Full speed

delay(1000); // Run at full speed for 1 second

// Deceleration phase

for (int pwmValue = pwmRange; pwmValue >= 0; pwmValue--) {

pwmWrite(pwmPin, pwmValue);

delay(10); // Adjust delay for deceleration rate

}

// Stop

delay(1000); // Stop for 1 second

}

int main() {

setup();

while (1) {

loop();

}

return 0;

}

C++

Copy code

**Summary:**

* **Arduino**: Uses the analogWrite function to output PWM for the specified pin. The delay function is used to control the rate of acceleration and deceleration.
* **Raspberry Pi**: Uses the wiringPi library to control PWM output. This requires setting up the PWM mode, range, and clock to appropriately manage the PWM signal.

Let me know which platform you are working on, or if you have specific requirements so I can better tailor the solution to your needs!

Copy

Good response

Bad response

on stm32m103

To generate PWM signals for acceleration and deceleration on an STM32 microcontroller (such as the STM32F103), we typically use the HAL (Hardware Abstraction Layer) library provided by STMicroelectronics. The steps involve configuring the PWM output using a timer peripheral and then updating the PWM duty cycle in the code.

Here is an example using STM32CubeMX and HAL library in STM32CubeIDE.

**Step-by-Step Procedure**

**1. Configure PWM with STM32CubeMX**

1. **Open STM32CubeMX** and create a new project for the STM32F103 microcontroller.
2. **Configure the clock settings** as needed.
3. **Enable Timers** (e.g., TIM2) for PWM output:
   * Select TIM2 and configure one channel (e.g., Channel 1) in PWM mode.
   * Choose an appropriate pin for PWM output (e.g., PA0 for TIM2\_CH1).
4. **Set Timer Frequency** and **PWM properties**:
   * Configure the timer's prescaler and counter period to achieve the desired PWM frequency.
   * The counter period determines the PWM resolution.
5. **Generate the code** using STM32CubeMX.

**2. Write the Code in STM32CubeIDE**

The generated code will include the necessary initialization routines, and you'll need to add the logic for accelerating and decelerating the PWM signal.

Here's an example of how the code might look:

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private variables ---------------------------------------------------------\*/

TIM\_HandleTypeDef htim2;

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_TIM2\_Init(void);

/\* User Code \*/

void setPWM(uint16\_t pwmValue) {

\_\_HAL\_TIM\_SET\_COMPARE(&htim2, TIM\_CHANNEL\_1, pwmValue);

}

int main(void) {

/\* Reset of all peripherals, Initialize the Flash interface and the Systick. \*/

HAL\_Init();

/\* Configure the system clock \*/

SystemClock\_Config();

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_TIM2\_Init();

/\* Start PWM \*/

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_1);

uint16\_t pwmValue = 0;

uint16\_t maxPWMValue = \_\_HAL\_TIM\_GET\_AUTORELOAD(&htim2);

while (1) {

// Acceleration phase

for (pwmValue = 0; pwmValue <= maxPWMValue; pwmValue++) {

setPWM(pwmValue);

HAL\_Delay(10); // Adjust delay for acceleration rate

}

// Full speed

HAL\_Delay(1000); // Run at full speed for 1 second

// Deceleration phase

for (pwmValue = maxPWMValue; pwmValue > 0; pwmValue--) {

setPWM(pwmValue);

HAL\_Delay(10); // Adjust delay for deceleration rate

}

// Stop

HAL\_Delay(1000); // Stop for 1 second

}

}

static void MX\_TIM2\_Init(void)

{

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

htim2.Instance = TIM2;

htim2.Init.Prescaler = 72 - 1; // Adjust prescaler as needed

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 1000 - 1; // Adjust period for PWM resolution

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

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, &sConfig

C

Copy code

Copy

Good response

Bad response

continue

Certainly! Here is the continuation of the code within the MX\_TIM2\_Init function and the supporting functions for the STM32 microcontroller:

if (HAL\_TIM\_PWM\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

HAL\_TIM\_MspPostInit(&htim2);

}

static void MX\_GPIO\_Init(void)

{

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\*Configure GPIO pin : PA0 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0;

GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.HSEPredivValue = RCC\_HSE\_PREDIV\_DIV1;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL9;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief This function handles any errors that occur during initialization.

\* @retval None

\*/

void Error\_Handler(void)

{

while(1)

{

// Stay here to indicate an error (could also add custom error handling here)

}

}

C

Copy code

**Explanation & Final Notes**

* **TIM2 Configuration**:
  + **Prescaler**: Divides the timer's input clock to slow it down. Adjust this for your specific needs.
  + **Counter Period**: Defines the maximum count value. Adjust to set the PWM resolution.
  + **Clock Division**: Sets the clock division.
  + **Output Compare (OC) Mode**: Sets PWM mode (PWM1 in this case).
* **GPIO Initialization**:
  + Configures the GPIO pin for the Timer's PWM output.
* **System Clock Configuration**:
  + Sets up the system clock source and configuration for the microcontroller.
* **Error Handler**:
  + Infinite loop to indicate error state; customize as needed for your error handling.

**Usage**

* **Set PWM Duty Cycle**: The setPWM function is used to update the PWM value.
* **Control Loop**: In the main function, the PWM value is varied to produce acceleration and deceleration.

**Summary**

This code sets up a PWM signal on an STM32F103 and varies the duty cycle to emulate acceleration and deceleration. Make sure to adjust the TIM2 prescaler, counter period, and delay values to fit your specific requirements. If you need more detailed adjustments or have further specific questions, feel free to ask!