EE-156 Lab 4

Emmett Roberts & Griffin Faecher

10/6/2025

Driving a DAC with FreeRTOS ticks using HAL

Professor Joel Grodstein

Tufts University

By the time we use the Arduino API, we've added substantial overhead to *analogWrite*(). Can you list some of the overhead?

analogWrite() in our lab4_DAC.c file has plenty of overhead. Everytime we call analogWrite we have to check if the DAC has been enabled, since the first time its called it isn't. We called the repetitive initialization functions as shown in the small code snippet below. Other overhead includes us simply checking to see which pins the function is being called with. These repeated checks take extra time every time the function is called, making the code slower in general.

```
if (!DAC1_enabled) {
    MX_DAC1_Init();
    MX_GPIO_Init();
    HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);
}
DAC1_enabled = 1;
```

Figure 1: analogWrite() snippet

Using the FreeRTOS tick to time our sawtooth was nice and simple. However, we were fairly limited in the waveforms we could create. If we tried to create musical notes (and say we wanted them to roughly be sine waves), what are some limits on the note frequencies that we could reasonably build?

Well since the analog wave generation was limited by the timing between interrupt ticks (1ms), I would assume the musical notes would have similar problems. The note frequencies that were higher than 500 Hz would become very distorted/jittery. This gives notes at least 2 samples per tick, which is the upper limit of distortionless output.

If we were to try to construct a sine wave with, lets say, 10 different values, then the highest frequency wave we could create would be 100 hz. This would allow us to play up to a G2 on a piano. However, a 10 value sine wave would be incredibly jittery, meaning the actual output would probably not be consistent. If we used a sine wave with 20 different values (outputs per period), then we could produce a G1, which is a very low note.

Many groups had sawtooths with a fair amount of jitter on the scope. Explain a code scenario in *task_DAC_sawtooth*() that could cause the problem, and how to fix it.

The jitter on our sawtooth was caused by the sharp increases in the value integer that we were outputting through the DAC. The code below shows that we were adding 25 to the output level of the dac every millisecond, meaning that our sawtooth only had 10 distinct voltages that it output.

```
value = (value + 25);
if(value >= 250) {
        value = 0;
}
vTaskDelay(1);
```

Figure 2: task_DAC_sawtooth() snippet

There are two ways to fix this. One is to decrease the frequency of the waveform. If the frequency is decreased, there is more time for the dac to output distinct voltages, thus decreasing the overall jitter of the waveform. The other way is to decrease the maximum (or minimum) voltage of the waveform. If, we lowered the maximum output voltage from 3.3V to a lower value (say 2V), the value counter wouldn't have to count so high, and thus there can be more definition in the waveform.

CODE:

MAIN:

```
void task_sawtooth (void * pvParameters) {
    uint32_t value = 0;
    while(1){
        analogWrite(A3, value);
        analogWrite(A4, value);
        value = (value + 25);
        if(value >= 250) {
                value = 0;
        }
        vTaskDelay(1);
    }
}

//every 10ms 2550 ms = 2.55s
// .01s / 51 = 0.039 vtask delay

#define BLINK_GREEN_DELAY ( 500 / portTICK_PERIOD_MS )
void task_blink_green (void *pvParameters) {
        bool is_on = 0;
        while(1) {
                digitalWrite(D13, is_on);
                is_on = !is_on;
                vTaskDelay(250);
        }
}

int main() {
    //clock_setup_16MHz();      // 16 MHz, AHB and APH1/2 prescale=1x
    clock_setup_80MHz();        // 80 MHz, AHB and APH1/2 prescale=1x

    // The green LED is at Nano D13, or PB3.
    pinMode(D13, "OUTPUT");
    pinMode(A3, "OUTPUT");
    pinMode(A4, "OUTPUT");
    digitalWrite (D13, 0);

    // Create tasks.

    TaskHandle_t task_handle_sawtooth = NULL;
    BaseType_t task_create_OK = xTaskCreate (
        task_sawtooth, "sawtooth gen",
        100, // stack size in words
        NULL, // parameter passed into task, e.g. "(void *) 1"
        1+tskIDLE_PRIORITY, // priority
        &task_handle_sawtooth);
    if (task_create_OK != pdPASS) for ( ;; );

    // Next, writer task #1, that just writes.
    TaskHandle_t task_handle_blink_green = NULL;
    task_create_OK = xTaskCreate (
        task_blink_green, "blink green",
        100, // stack size in words
        NULL, // parameter passed into task, e.g. "(void *) 1"
        tskIDLE_PRIORITY, // priority
        &task_handle_blink_green);
    if (task_create_OK != pdPASS) for ( ;; );

    vTaskStartScheduler();
}
```

DAC:

```c
//////////////////////////////////////////////////////////////////////
// HAL version.
//////////////////////////////////////////////////////////////////////

#ifdef USE_HAL

#include "stm32l4xx_hal.h"
#include "stm32l4xx_hal_dac.h"

/* Private variables ---------------------------------------------------------*/
DAC_HandleTypeDef hdac1;

void Error_Handler(void);

/**
  * @brief DAC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_DAC1_Init(void) {

    __HAL_RCC_DAC1_CLK_ENABLE();

    if (HAL_Init()) Error_Handler();

    /* USER CODE BEGIN DAC1_Init 0 */

    /* USER CODE END DAC1_Init 0 */

    DAC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN DAC1_Init 1 */

    /* USER CODE END DAC1_Init 1 */

    /** DAC Initialization
     */
    hdac1.Instance = DAC1;
    if (HAL_DAC_Init(&hdac1) != HAL_OK)
    {
    Error_Handler();
    }

    /** DAC channel OUT1 config
     */
    sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
    sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
    sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
    sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_DISABLE;
    sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
    if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
    {
    Error_Handler();
    }

    /** DAC channel OUT2 config
     */
    if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_2) != HAL_OK)
    {
    Error_Handler();
    }
    /* USER CODE BEGIN DAC1_Init 2 */

    /* USER CODE END DAC1_Init 2 */

}
```

```c
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* USER CODE BEGIN MX_GPIO_Init_1 */

  /* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);

  /*Configure GPIO pin : PB3 */
  GPIO_InitStruct.Pin = GPIO_PIN_3;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* USER CODE BEGIN MX_GPIO_Init_2 */

  /* USER CODE END MX_GPIO_Init_2 */
}

// This is the Arduino API.
// Most Arduino boards don't have a DAC; so on those boards, this function
// actually does a PWM on a digital GPIO pin. But it's true analog on the few
// Arduinos that have a DAC, and that's what we do too. Also, it defaults to
// 8-bit writes; a few Arduinos support AnalogWriteResolution(12 bits). We
// could do that easily, but I haven't bothered.
// - 'Pin' can only be A3 (PA4, for DAC 1) or A4 (PA5, for DAC 2).
// - 'Value' is in [0,255] to write in [0,3.3V].
void analogWrite (enum Pin pin, uint32_t value) {
    static bool DAC1_enabled=0, DAC2_enabled=0;
    if (pin==A3) {      // DAC #1
        if (!DAC1_enabled) {
            MX_DAC1_Init();
            MX_GPIO_Init();
            HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);
        }
        DAC1_enabled = 1;
        HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_8B_R, value);
    } else if (pin==A4) {   // DAC #2
        if (!DAC2_enabled) {
            MX_DAC1_Init();
            MX_GPIO_Init();
            HAL_DAC_Start(&hdac1, DAC_CHANNEL_2);
        }
        DAC2_enabled = 1;
        HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_2, DAC_ALIGN_8B_R, value);
    } else
        error ("Called analogWrite() on a non-DAC pin");
}
```

```c
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#endif
```