Digital Audio Synthesis: Milestone 1

Document Stakeholders (THESE NAMES ARE FOR INFORMATION ONLY)	
Author	**Group**
Approver	**Chase Griswold**
Approver	**Fernando Araujo**
Approver	**Vincent Banh**
Approver	**Alex Baret**

PROGRAM / COURSE NUMBER

ECE: Embedded Systems 5/6780

University of Utah

201 Presidents' Cir

Salt Lake City, Utah 84112

BASELINE	DATE
Current	23 MAR 2024

This Page Intentionally Left Blank

TABLE OF CONTENTS

1 [-] MILESTONE 1: DIGITAL AUDIO SYNTHESIS

To-do List for Project:
Explore the utilization of both hardware and software to implement an efficient, robust audio application.

PCB Design:
- See PCB Detailed Development Specification (DDS) Document.

Hardware/Software exploration:
- Implement Timers as a useful and cost-effective means to trigger actions at regular intervals.
- Implement the STM DAC for communicating with a Oscope/speaker by converting digital samples into analog signals.
- Implement DMA for data transfer with minimal processor resources, enabling simultaneous audio streaming and other processing tasks.

1.1 [-] How we will implement our design

1.1.1 [-] Timers

Embedded timers offer a plethora of advanced functionalities, rendering them highly versatile in diverse applications. Embedded timers can trigger events repeatedly through auto-reload mechanisms, facilitate both count up and count down operations, and generate rectangular waveforms, such as Pulse Width Modulation (PWM) signals. These enhanced capabilities empower embedded systems with precise timing control and signal generation, facilitating a wide range of functionalities in applications spanning from industrial automation to consumer electronics.

Timers in embedded systems are driven by the MCU clock, serving as the fundamental timing mechanism for various functionalities such as timers, ADC, DAC, and communication protocols. The clock operates at a fixed frequency, such as 8MHz, which can be divided using prescalers and auto-reload values. These divisions trigger interrupts at specified intervals, enabling actions like DMA transfers and ADC conversions. For instance, to generate audio output at a specific sample rate, we calculate the prescaler (PSC) and auto-reload (ARR) values based on the desired frequency and the clock frequency.

To achieve an output sample rate of 42 kHz with a clock speed of 8 MHz, we can calculate the prescaler (PSC) and auto-reload (ARR) values using the formula:

$$\text{freq}_{\text{timer}} = \frac{\text{freq}_{\text{clock}}}{\text{PSC} \times (\text{ARR}+1)}$$

Given:

- $\text{freq}_{\text{timer}} = 42,000\,\text{Hz}$
- $\text{freq}_{\text{clock}} = 8\,\text{MHz} = 8,000,000\,\text{Hz}$

We can rearrange the formula to solve for PSC and ARR:

$$\text{PSC} = \frac{\text{freq}_{\text{clock}}}{\text{freq}_{\text{timer}} \times (\text{ARR}+1)}$$

Let's start by selecting an arbitrary value for ARR (auto-reload) and then calculate PSC:

Let's choose ARR = 199.

$$\text{PSC} = \frac{8{,}000{,}000}{42{,}000 \times (199+1)}$$
$$\text{PSC} = \frac{8{,}000{,}000}{42{,}000 \times 200}$$
$$\text{PSC} = \frac{8{,}000{,}000}{8{,}400{,}000}$$
$$\text{PSC} \approx 0.9524$$

As PSC must be an integer value, we round it up to the nearest integer:

PSC = 1

Now that we have PSC = 1, we can calculate ARR using the same formula:

$$\text{ARR} = \frac{\text{freq}_{\text{clock}}}{\text{freq}_{\text{timer}} \times \text{PSC}} - 1$$
$$\text{ARR} = \frac{8{,}000{,}000}{42{,}000 \times 1} - 1$$
$$\text{ARR} = \frac{8{,}000{,}000}{42{,}000} - 1$$
$$\text{ARR} \approx 190.48 - 1$$
$$\text{ARR} \approx 189.48$$

Again, as ARR must be an integer value, we round it to the nearest integer:

ARR = 189

Therefore, for an output sample rate of 42 kHz and a clock speed of 8 MHz, the recommended values for the prescaler (PSC) and auto-reload (ARR) registers would be PSC = 1 and ARR = 189.

A Nyquist sampling rate of 42 kHz should be suitable for our audio application due to its ability to accurately capture the frequency range of human hearing. The Nyquist theorem states that to accurately represent a signal, the sampling rate must be at least twice the highest frequency present in the signal. Since the upper limit of human hearing is generally considered to be around 20 kHz, a sampling rate of 42 kHz comfortably exceeds this threshold, allowing for accurate reproduction of audio signals without introducing aliasing or distortion. Additionally, using a sampling rate higher than the Nyquist rate ensures that the audio signal can be accurately reconstructed during playback, providing high-fidelity audio reproduction. Therefore, a Nyquist sampling rate of 42 kHz is well-suited for audio applications, providing a balance between fidelity and efficiency.

1.1.2 [-] DAC

Digital-to-analog converters (DACs) play a crucial role in bridging the gap between digital and analog domains, facilitating the conversion of binary data into electrical voltages or sound waves. In contrast to analog-to-digital converters (ADCs), which convert continuous analog signals into discrete digital values, DACs perform the inverse operation, aiming to recreate a continuous analog waveform from discrete digital samples. The output waveform's fidelity and shape are determined by the DAC implementation, with simpler DACs producing output at discrete voltage levels and more sophisticated ones employing interpolation techniques to approximate intermediate values. Despite potential inaccuracies in the interpolated values, these methods contribute to smoother output waveforms. On the STM32 platform, DAC functionality accommodates samples quantized to either 8 or 12 bits, offering flexibility in digital-to-analog conversion.

We will setup the DAC for 12-bit conversion, and expound upon what we learned in Lab 6 to implement our design.

1.1.3 [-] DMA

Direct Memory Access (DMA) is a major asset for embedded programmers. DMA proves invaluable by freeing up processor resources for other critical tasks. This includes activities like preparing subsequent buffers or rendering the graphical user interface. DMA facilitates diverse data transfers, spanning from memory-to-peripheral, peripheral-to-memory, to inter-peripheral or inter-memory transfers. For our project, we will focus on memory-to-peripheral transfers, particularly concerning DAC.

We will implement DMA to facilitate efficient data transfer from memory to the DAC. This setup will allow us to stream audio data from memory to the DAC without the need for continuous CPU intervention, thereby freeing up processing resources for other tasks.

To achieve this, we will configure the DMA controller to transfer audio samples stored in memory to the DAC peripheral. The DMA controller will handle the transfer process independently of the CPU, utilizing its own internal registers and logic to move data between memory and the DAC.

We will specify the source and destination addresses for the DMA transfer, indicating where the audio samples are stored in memory and the DAC's data register where the samples should be written. Additionally, we will configure the DMA transfer mode to double buffer (See below source text) to determine how the data transfer should be performed.

Further Reading: Baeldung: How Do DMA Controllers Work?

The following is from above source:

A critical concern arises post-buffering of a second round of data: how do we ascertain the completion of the first data transfer round? This dilemma prompts an evaluation of two synchronization methods: polling and interrupts. Polling involves blocking and awaiting the completion of the first round before initiating the next, while interrupts trigger an interrupt signal upon transfer completion, enabling the initiation of subsequent rounds within the interrupt handler. Despite their distinct advantages and considerations, both methods must ensure buffer integrity and prevent overwrites during concurrent DMA operations and buffer replenishment.

Double buffering is superior to single buffering when setting up DMA because it prevents the risk of overwriting data while it is being transferred. With single buffering, there is a possibility of data corruption or loss if new data is written to the buffer before the previous data transfer is complete. In contrast, double buffering allows one buffer to be filled with data while the other buffer is being transferred, ensuring continuous and uninterrupted data transfer without the risk of corruption or loss. This approach enhances the reliability and efficiency of DMA operations, particularly in scenarios where data integrity is critical.

2 [-] NOTES

Additional explorations (Outside the scope of this project, but this stuff came up in the reading for embedded audio applications).

- Experiment with generating stereo audio, exploring the possibility of stereo output for Channel 2.
- Explore streaming via UART with DMA support.
- Investigate the use of SIMD instructions to buffer multiple samples simultaneously.

SIMD stands for "Single Instruction, Multiple Data." It is a type of parallel computing architecture that performs the same operation on multiple data points simultaneously. SIMD instructions enable a processor to execute a single instruction across multiple data elements, allowing for efficient parallel processing of data. This technology is commonly used in multimedia applications, scientific computing, and other tasks that involve processing large amounts of data in parallel. In SIMD architecture, multiple data elements are processed in parallel using a single instruction, thereby increasing computational throughput and efficiency.