CE-2812, Lab Week 8, YATA (Yet Another Timer Application) – Analog Signal Generation with DMA

1 PURPOSE

The purpose of this lab is to explore another peripheral (DAC) and optionally DMA.

2 PREREQUISITES

- The Nucleo-F446RE board had been mounted onto the Computer Engineering Development Board.

3 ACTIVITIES

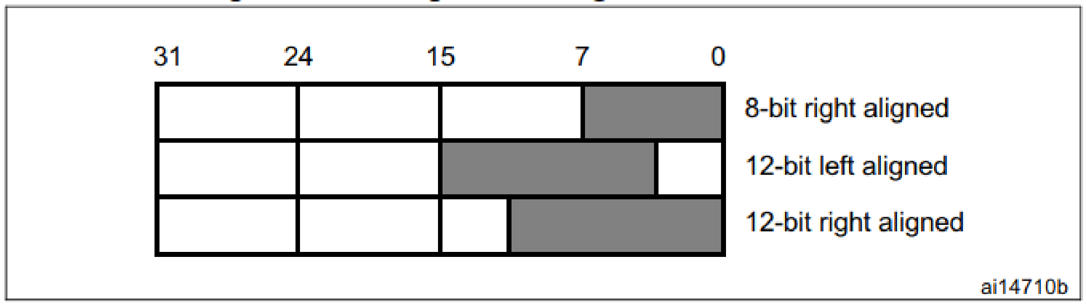
3.1 BACKGROUND

3.1.1 DAC

Our full-featured microcontroller happens to have a two-channel digital to analog converter (DAC). This is a common but certainly not universal peripheral for microcontrollers. In most basic form, it works very much opposite the ADC peripheral – you write a register with a particular value and a proportional analog voltage appears on the output. Of course, for signal generation, the register will need regular updates.

Upon first glance at the reference manual, the DAC appears to be complicated, with no less than 14 registers. It turns out, however, that many of these registers are redundant, and only a few are really needed. The multitude of registers allows several different formats of source data to be handled by hardware instead of by software. The basis of the issue is that the DAC operates from a 12-bit source number. The excerpt below from the reference manual depicts how data written to different registers will be interpreted.

Figure 93. Data registers in single DAC channel mode



Basic configuration is actually quite minimal. The corresponding output pin(s) should be put in analog mode, the clock enabled to the DAC peripheral (in APB1ENR), and the DAC channel(s) enabled in the control register. Moving the source data to actually drive the output voltage can be done as soon as the source data written, or, can be triggered by a timer. These options are also configured in the control register.

Basic timers TIM6 and TIM7 are specifically included for timing DAC updates and do not have capture or compare functions. Several other timers can be used as well.

3.1.2 DMA (Optional)

The last three labs have used the timer/counter peripherals in various configurations. We saw that we can generate a simple square wave to drive the piezo speaker with no intervention, and then saw by using interrupts, we can manage that signal generation in the "background" via the interrupt service routine. This application will be similar in that we will generate a signal. However, instead of managing that signal with an interrupt / ISR scheme (which would be perfectly easy to do…) we will use yet another provision of our microcontroller – direct memory access (DMA).

DMA allows peripherals to directly access memory, independent of the core. Consider the background music lab. When it was time to play the next note, the ISR needed to read the new frequency from the song array (into a general purpose register) and then write that same value to the timer's ARR and/or CCR1 register. What if the timer could simply access the note structure directly at the correct time and not bother the processor with an interrupt request at all? It can, and that is the premise of DMA. Of course, this behavior must be timed, so once again, we will call on a timer/counter to pace this process.

3.2 General Requirements

- Add option to "generate waveform." Ideally, this will be in command form as illustrated below.
- Once invoked, the "generate waveform" function should:
  - Use malloc() to dynamically allocate memory to store one cycle of the desired signal
  - Calculate desired DAC samples and store in allocated memory
  - Configure DAC
  - configure a timer for DAC updates – TIM6 is recommended
  - Configure DMA (optional)
    - If not using DMA, you will need an ISR to service the timer / DAC
  - Start signal generation
- Signal generation should continue indefinitely until halted by another command (much like halting background music)
- You should be able to invoke other menu commands while signal generation continues in background. This should include background music and measuring frequencies as well as the original memory r/w/dump commands.
- Be sure not to reuse any resources used for any other functions currently implemented.

3.3 Command Structure

- The command issued to generate a signal should be a single line command and not a series of prompts.
- At a minimum, you need to support a sine wave. You may wish to optionally support triangle, sawtooth, other?? Sine is available in the C standard library (math.h)
- You need to support a range of frequencies and sampling rates. The suggested minimum range of sine waves is 10 Hz to 10 kHz with a minimum of 10 samples per cycle and supporting at least 100 samples per cycle (or more).
- You may have the generated waveform vary from min to max voltage (writing 0 to 4095 to DAC) or, may optionally have the amplitude set by your command.
- Possible command: **prompt> sine 1000 100** ← generate sine wave, 1000 Hz, with 100 samples per cycle.
- For the command above, you will need to dynamically allocate memory to store 100 samples of one cycle of a sinewave, and then configure the timer to play back those 100 samples so that a 1 kHz sinewave is produced. In other words, you will need to write a sample to the DAC every 10 microseconds (1000 Hz means 1000 cycles per second, 100 samples per cycle means 0.001/100 sample rate or about 100 microseconds).
- There may be some combinations of samples and signal frequency that do not work very well, such as 10 kHz with 1000 samples per cycle. This combination (which exceeds specifications) would require an update every

100 nanoseconds which will not be possible with our processor running at 16 MHz. Our processor can run faster…

4 Deliverables

When completed:

1. Submit to Canvas a **single pdf** printout of your completed source code to Canvas. **Include in a comment block at the top of your code a summary of your experience with this project.**
2. Ask to demo your lab to instructor. You can do this via writing your name on the whiteboard.
   a. If you demo during lab in Week 8, you will earn a 10% bonus on this lab.
   b. If you demo during lab in Week 9, you will be eligible for full credit.

- Demos are ONLY accepted during lab periods. If you are unable to demo by the end of lab in Week 8, you lose the 10% of the assignment attributed to the demo (per syllabus).

- Demos must be ready a reasonable amount of time before the end of the lab period. If you write your name on the board at 9:45 and lab ends at 9:50, and there are five names in front of yours, you will be unlikely to complete your demo by the end of lab and hence lose the bonus or demo points.

4.1 Grading Criteria

For full credit, your solution must:

- Minor errors usually result in a deduction of ~ 3 points (three such errors results in ~ a letter grade reduction)
- Major errors, such as not achieving a requirement, usually result in a deduction of 5 to 10 points.