EEE3096S Test 3 Memo

28 October 2021

Q1 DAC vs PWM

5 Points

There are many advantages of using a DAC peripheral over just driving a GPIO bit to make PWM output signals (as discussed in Lecture 21, which can avoid the needing to interface to a DAC). Considering that the effective performance of an embedded system may require accurately generated signals of various voltages, provide a brief motivation for why a DAC could be preferable in an embedded system rather than just using PWM.

Sample Answers:

As mentioned in Lecture 21, a major objective of a DAC is to turn a digital value into a proportional voltage (or current; but we can focus on voltage). There are of many varieties of DAC available, but a regular feature of these is that they would be able to provide a voltage that is fairly stable, at least for a while. The drawback of relying on PWM is that if it is driven by the microprocessor using a GPIO, it may be difficult to ensure the pulses are produced consistently, it will also require a lot of processing effort towards toggling the PWM bit, which would be rather wasteful processing. A GPI-driven PWM may likely cause a signal quality that is jittery or fluctuating, it may take longer than a DAC to change voltage levels (e.g. if relying on a capacitor to get charged up).

The main reasons are: DAC gives more consistency, and reduces processing load. Could argue that it improves performance and software quality and consistency (without having to keep toggling a GPIO).

Q2 Benefits of the Flash ADC

3 Points

There are many types of Analog to Digital Converter (ADC) available. The Flash ADC has many advantages over other types of ADC, such as the Successive Approximation ADC. Select which of these is a clear benefit of a Flash ADC over the Successive Approximation ADC:

- The Flash consumes more power
- The Flash provides more digital codes
- The Flash is faster ← this is the main aspect
- The Flash ADCs are usually easier to interface

(interfacing not so relevant an answer because most types of ADC are available to suite various types of interfacing standards).

Q3 ENOB

5 Points

The Effective Number of Bits (ENOB) is an important factor in determining how effective the operation of an ADC is. What is meant by the ENOB of an ADC? For example, If we have an 8-bit ADC that has an ENOB below 8 what does this imply?

Sample Answers:

Effective Number of Bits = ENOB

As mentioned in Lecture 24: The resolution of an ADC is specified by the number of bits, N, that represents the analogue voltage fed into it, as in an N-bits value to represent the analogue input fed into it. In principle giving 2^N signal levels that it can distinguish. The ENOB indicates how many of these N bits effectively contribute towards representing the sampled input.

If an 8-bit had an ENOB below 8 then the voltage levels that it can effectively distinguish is less than 2^8, the lower significant bit for instance may be unstable (e.g. fluctuates due to noise) which would effectively imply there are only really 2^7 voltages that the ADC can distinguish in such a case.

Q4 Analysing performance

12 Points

The following scenario relates to the two sub-questions that follow:

> Consider the block diagram showing that a 10-ADC is connected up to a microcontroller via memory-mapped access.
>
> … see question paper …

Q4.1 Characterizing an ADC offset error

5 Points

What is meant by the offset error of an ADC? Discuss how you might go about characterizing the offset error for this case. You can discuss how you might connect things up, code you might need to write and, if there is an offset error, what you might do to make the provided sampling program work more accurately.

Sample Answers:

The offset error of an ADC is defined as a deviation of the ADC output code transition points that is present across all output codes. So, if an ADC measures a voltage X, if it has an offset error of *y* it return a code that (erroneously) represents an input voltage of X + *y* instead of X.

A method to measure the offset error, at least to get a view on the offset error at low voltages of the range, is to simply feed in the lowest voltage that the ADC can measure, e.g. if it measures from 0V to 5V, can connect it to 0V. Write some code that reads the ADC returned voltages, can read it for a while, not just one sample because that might just return the right value at that moment, would be better to do a min max test of the voltage level while a stable e.g. 0V is fed in for a few ms at least. E.g.:

```
// program to test that 0V gives 0 out:
while (1) {
  ADC_start();          // tell ADC to start a new sample
  while (!ADC_ready()); // wait for ADC to be ready
    x = ADC_read();       // get next sample
    if (x!=0) {
        printf("ADC didn't return 0 at 0V!");
        exit(1);
        }
    delay_ms(10);
    }
```

And see if it returns a code 0 that accurately represents 0V. But it is more thorough to try a range of voltages to see how consistent the offset error is at different voltages.

(note to marker: it is sufficient if the student just discusses testing one voltage, at least lower and not highest voltages as those might supper other problems. Student doesn't need to provide code can just mention it in text).

Q4.2 Speed of sampling ADC

4 Points

As mentioned earlier, the ADC in use does not always respond at the same speed. In our code we have a high-resolution time available via the gettick_us function. Discuss how you might adjust the code to determine how long the ADC took (from calling ADC_start) to the time that the sample is read and put in variable x ready for use.

Sample Answers:

The gettick_us function can be used to time the amount of time it takes between calling ADC_start to asking the ADC to start a new sample, and the time after the sample read from the ADC is written to x. Code to do this would simply be:

```
unsigned long t0;         // ← need at least one var to track time
unsigned long duration;
while (1) {
   t0 = gettick_us();     // get first time stamp  ← added
   ADC_start();           // tell ADC to start a new sample
   while (!ADC_ready()); // wait for ADC to be ready
   x = ADC_read();        // get next sample
   samps[n_samps] = x;    // put sample in array
   duration = gettick_us() - t0; // ← work out the duiration
   n_samps = (n_samps+1)%MAXSAMPS; // increase and wrap counter
   do_processing();       // do processing of samples
   delay_ms(10);          // wait 10ms before starting next sample
   }
```

Note to marker: it is not necessary for the student to provide the full code result, it is sufficient for the student to just explain a solution and suggestion of code needed.

Q4.3 Improving accuracy

3 Points

In the code provided, the do_processing() operation may take longer than other times, in particular as the samps array fills up, it takes longer to complete. Although samples are meant to be taken every 10ms. Discuss how the code might be improved to improve the likelihood of samples being read regularly at 10ms.

Sample Answers:

Simply having the delay(10) is not a very reliable method of sampling precisely every 10ms. If do_processing() and the other operations between the delays take must time (beyond a few 100us) then the accuracy is going to reduce. Every now and then it will take a little longer than previous times to take the next sample, would be a problem of the sampling drifting (imagine if you were to sample 1M samples, with the drifting problem it might end up being a bit under 1M that are actually sampled).

A solution would be using the timer, in a similar was to the answer to 4.2, a time stamp, e.g. t0, can be obtained when one sample is asked for, and then the end of the while loop can wait until the time reaches 10ms after the time stamp t0 has elapsed.

Note to marker: it is OK if the student focuses mainly on showing code improvement, preferably with some comments to aid the explanation; an suitable textual description is sufficient for full marks for this question.