|  |  |
| --- | --- |
| Note again the addition of two callbacks:  HAL\_ADC\_ConvHalfCpltCallback() and HAL\_ADC\_ConvCpltCallback(). The first is called whenever the ADC (and DMA) fills up half the buffer. The second function is called whenever the second half the buffer is filled. You can use this to create an easy double (or “ping pong”) buffer. Do something with the data in half the buffer while the second half is being filled by the DMA.  Run the program in the debugger. Add a breakpoint in one of the HAL\_ADC callbacks and let the program stop there. Hover your mouse over the adc\_buf variable to peek inside the buffer. You should see it filled with ADC conversion values.  We’re not doing anything with this data, as this is just an example to show how to fill up a buffer with DAC values without using the CPU, which should be idling in our while(1) loop! Note that both halves of the buffer get filled, as the DMA clock does not stop when we stop the CPU with the debugger, so it will just keep working to put values into the buffer.  If you connect an oscilloscope to D13, you should see the line toggling each time one of the callback functions runs.  As you can see, it takes about 472 μs to fill up one half of the buffer. That means in a little under a millisecond, we can fill up the entire buffer--4096 samples--without using the CPU! This comes out to be around 4 Msamples/sec.  If you take a look at the ADC clocks in STM32CubeIDE, you can see that they are set to 64 MHz and asynchronous from the system clock. If you connect the ADC clock to an 80 MHz source, you can get that sampling rate up even a little higher. Additionally, we are using 12-bit conversions. You can sacrifice some resolution for faster sample and convert times. | Notez à nouveau l'ajout de deux rappels :  HAL\_ADC\_ConvHalfCpltCallback() et HAL\_ADC\_ConvCpltCallback(). Le premier est appelé chaque fois que l'ADC (et le DMA) remplit la moitié du tampon. La deuxième fonction est appelée chaque fois que la seconde moitié du tampon est remplie. Vous pouvez l'utiliser pour créer facilement un tampon double (ou "ping pong"). Faites quelque chose avec les données dans la moitié du tampon pendant que la seconde moitié est remplie par le DMA.  Exécutez le programme dans le débogueur. Ajoutez un point d'arrêt dans l'un des rappels HAL\_ADC et laissez le programme s'arrêter là. Passez votre souris sur la variable adc\_buf pour jeter un coup d'œil à l'intérieur du tampon. Vous devriez le voir rempli de valeurs de conversion ADC.  Nous ne faisons rien avec ces données, car ce n'est qu'un exemple pour montrer comment remplir un tampon avec des valeurs DAC sans utiliser le CPU, qui devrait être inactif dans notre boucle while(1) ! Notez que les deux moitiés du tampon sont remplies, car l'horloge DMA ne s'arrête pas lorsque nous arrêtons le processeur avec le débogueur, il continuera donc à fonctionner pour mettre des valeurs dans le tampon.  Si vous connectez un oscilloscope à D13, vous devriez voir la ligne basculer à chaque fois que l'une des fonctions de rappel s'exécute.  Comme vous pouvez le voir, il faut environ 472 μs pour remplir la moitié du tampon. Cela signifie qu'en un peu moins d'une milliseconde, nous pouvons remplir tout le tampon -- 4096 échantillons -- sans utiliser le CPU ! Cela revient à environ 4 Méchantillons/sec.  Si vous regardez les horloges ADC dans STM32CubeIDE, vous pouvez voir qu'elles sont réglées sur 64 MHz et asynchrones par rapport à l'horloge système. Si vous connectez l'horloge ADC à une source de 80 MHz, vous pouvez augmenter encore un peu ce taux d'échantillonnage. De plus, nous utilisons des conversions 12 bits. Vous pouvez sacrifier une partie de la résolution pour des temps d'échantillonnage et de conversion plus rapides. |

Lien : [Getting Started with STM32 - Working with ADC and DMA (digikey.fr)](https://www.digikey.fr/en/maker/projects/getting-started-with-stm32-working-with-adc-and-dma/f5009db3a3ed4370acaf545a3370c30c?_ga=2.99241560.772568024.1684842890-1667449458.1683033503)