Laborator 7

ADC și DMA

Secțiuni lucrare:

ADC – Analog to Digital Converter (Convertor Analog Numeric)

Rezoluția ADC-ului

Rezoluția ADC-ui este unul dintre elementele cheie care determină cât de precis se va realiza conversia. Diferența de rezoluție se poate observa în figura de mai jos. Pe scurt, cu cât un ADC are o rezoluție mai bună, cu atât el va genera un răspuns mai apropiat de realitate. ADC-ul de pe microcontrolerul STM32F103 oferă programatorului posibilitatea de a alege rezoluția, sau numărul de biți pe care acesta va stoca valoarea covertită. ADC-ul poate să stocheze valoarea converititată pe un număr maxim de 12 biți (cu valori care variază între 0 și 4095), dar poate să stocheze valoarea covertită și pe mai puțini biți precum pe 10 biți (cu valori între 0 și 1023) , 8 biți (cu valori între 0 și 255) și chiar și 6 biți (cu valori între 0 și 63).

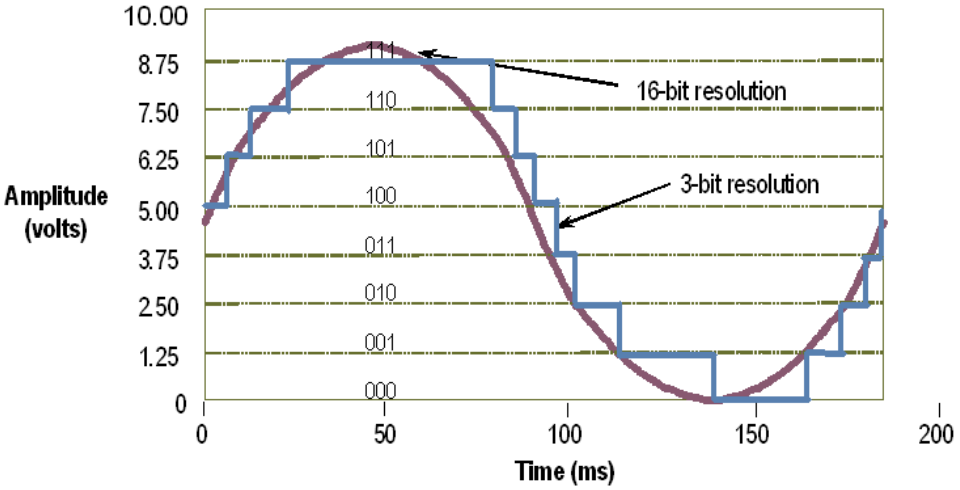


Figura 1. Rezoluție conversie ADC

Timpul de Conversie și timpul de sampling

Timpul de sampling este timpul necesar pentru a încărca toate condensatoarele necesare pentru eșantionare din interiorul ADC-ului. Această perioadă de eșantionare trebuie să fie suficient de mare pentru ca sursa care urmează să fie măsurată să încarce toți condensatorii și să îi mențină la acel nivel. Prin urmare, alegerea timpului de eșantionare va depinde de curentul pe care sursa măsurată poate să îl ofere. Cu cât curentul generat de sursă crește cu atât timpul de eșantionare poate să scadă. Modulul de ADC din componența microcontrolerului STM32F103 oferă 8 perioade de eșantionare predefinite (1.5, 7.5, 13.5, 28.5, 41.5, 55.5, 71.5, 239.5 cicli de ceas). Durata unui ciclu de ceas, depinde de frecvența modului ADC.

ADC NucleoF103

Convertorul Analog Numeric de pe microcontrolerul STM32F103 este unul dintre cele mai puternice convertoare din interiorul unui microcontroler, avand o frecvență maximă de sample de 1 Mhz. Microntrolerul are inclus două ADC-uri similare, ADC 1 cu 18 canale (canalul 16 si 17 sunt folosite pentru senzorul de temperatură intern și pentru tensiunea de referință a ADC-ului), iar ADC2 cu 16 canale. Fiecare ADC are patru moduri în care acesta poate să funcționeze: singly, continually, scanned și discontinously. Rezultatele obținute în urma conversiei pot fi transferate în memorie folosint DMA. Modulul ADC încorporat în microcontrolerul STM32F103 oferă o gamă largă de opțiuni programatorului.

Microcontrolerul STM32F103 încorporat pe placa Nucleo, rulează la o frecvență de 72Mhz. Semnalul de ceas pentru ADC este generat de un Prescaler. Imediat după inițializare, frecvența bus-ului rulează la aceeași frecvență ca și cea a sistemului. Conform datasheet-ului, clock-ul ADC-ului trebuie să fie undeva între 600khz și 14Mhz, prin urmare este nevoie de un prescaler de cel puțin 6 ca frecvența de ceas să fie în acest interval. Imediat după ce ADC-ul a fost configurat, acesta poate să fie pornit prin modificarea bitului ADON din registrul CR2. După o perioadă de stabilizare, ADC-ul poate să înceapă conversia. Conform manualului, perioada de stabilizare nu o să depășească 1us.

ADC-ul are o funcție de auto-calibrare, pentru a reduce erorile generate de variația condensatoarelor interne. Manualul sugerează ca ADC-ul să se auto-calibreze o dată imediat după pornirea acestuia.

Rezultatul conversiei (rezultat pe 12 biți) o să fie stocat într-o variabilă pe 16 biți. Acest rezultat poate să fie aliniat la stânga, sau la dreapta în funcție de necesitățile programatorului. Biți nefolosiți vor fi setați pe 0.

Programare ADC

ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;

Există aproximativ 10 moduri de setare a ADC-ului. ADC-ul 1 poate să lucreze sincronizat cu ADC-ul 2 pentru a realiza măsurători mai complexe care necesită măsurare în același timp a două unități, precum tensiune și curent pentru a calcula puterea. Modul independet, setează ADC1 independet față de ADC 2. Pentru mai multe informații puteți accesa link-ul)

ADC_InitStructure.ADC_ScanConvMode = ENABLE;

Se setează acest câmp ENABLE în momentul în care se dorește coversia informațiilor disponibile pe mai multe canale ale aceluiași ADC.

ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;

Acest câmp este setat enable, în momentul în care se dorește conversia continuă a datelor. Cu alte cuvinte, după ce ADC-ul a terminat de convertit un bloc de date, el va reîncepe conversia fără a avea nevoie de intervenția procesorului.

ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;

În acest câmp se specifică sursa de la care vine trigger-ul extern. Acesta poate să vine de la un Timer sau de la o întrerupere externă. Pentru exemplul din laborator nu se vor folosi surse de Trigger externe.

ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;

ADC-ul de pe STM32F103 este un ADC pe 12 biți. Datele pot fi stocate într-un registru pe 16 biți. În acest caz, ADC-ul oferă programatorului posibilitatea de a alinia datele la stânga sau la dreapta, completând restul de 4 biți cu zero.

ADC_InitStructure.ADC_NbrOfChannel = 3;

În acest câmp se setează numărul de canale ale ADC-ului folosite.

DMA

În multe proiecte cu microcontrolere, există o nevoie de a transfera datele dintr-o zonă în alta. De exemplu, se pot lua informațiile dintr-un periferic precum ADC și ulterior stocate în memoria flash/RAM. În cazul unei comunicații SPI, este nevoie de luarea datelor din memorie și punerea acestora pe bus-ul de SPI. Dacă aceste operații de mutare a datelor dintr-o zonă în alta sunt efectuate de procesor, acesta pierde timp de procesare. Pentru a evita folosirea procesorului în trasferul de date, majoritatea microcontrolerelor moderne au o unitate DMA (Direct Memory Access). DMA-ul transferă date între diverse zone de memorie fară intervenția procesorului, acesta fiind liber să se ocupe de operații mai importante.

Microcontrolerele de desitate mică STM32, precum cel folosit în acest laborator au o unitate DMA cu 7 canale, pe când cele cu densitate mai mare dispun de două unități DMA având în total 12 canale independente (DMA2 dispune de 5 canale). DMA-ul poate să automatizeze transferul de informație între două zone de memorie (memory to memory sau m2m), și de asemenea poate să realizeze operații de transfer de date între periferice și memorie sau invers memorie-periferic. Canalele DMA-ului pot să aibă patru nivele de propritate: very high, high, medium și low. În cazul în care două canale cu aceeși prioritate doresc să acceseze un periferic sau o zonă de memorie, canalul cu numărul cel mai mic primește accesul. Canalul de DMA

poate să fie configurat ca să mute informația într-un buffer circular. Cu alte cuvinte, are mecanisme interne, hardware, care îi permit să se deplaseze printr-un vector și cand ajunge la un anumit index, să se reseteze și să revină la prima poziție. Prin urmare, DMA-ul este o soluție ideală pentru orice trasfer de date de la un periferic către memorie.

Un alt lucru important, este faptul că DMA-ul accesează bus-ul doar când trimite date. Acest lucru este posibil datorită faptului că operația de calculare a adresei, trimisul de ACK și request-ul sunt făcute în timp ce un canal al DMA-ului folosește magistrala de date. Cu alte cuvinte, cand un canal termină de transferat datele, cel de al doilea canal este pregatit să trimită datele imediat.

Programarea DMA-ului

Fiecare canal al DMA-ului poate să fie configurat folosind patru regiștrii: memory address, peripheral address, number of data și configuration. De asemenea, toate canalele mai au doi regiștrii dedicați: DMA interrupt status register și interrupt flag clear register. O dată ce DMA-ul a fost setat, acesta poate să incrementeze singur zonele de memorie, fară să întrerupă funcționalitatea procesorului. Canalele DMA pot să genereze trei întreruperi: transfer finished, half-finished și transfer error.

DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;

Adresa de memorie de unde se citesc datele de la periferic.

DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)ADCConvertedValue;

Adresa de memorie unde se găsește variabila în care se dorește stocarea datelor. În exemplul de mai sus ADCCovertedValue este un vector, prin umare nu este necesar operatorul &.

DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;

Există două posibile inițializări:

- DMA_DIR_PeripheralDST – Acest define va fi folosit în cazul în care se dorește scrierea în regiștrii perifericului
- DMA_DIR_PeripheralSRC – Acest define va fi folosit în cazul în care se dorește citirea din regiștrii unui periferic.

DMA_InitStructure.DMA_BufferSize = 7;

Reprezintă numărul de elemente care vor fi citite/scrise. Cu alte cuvinte, în cazul exemplului de mai sus, se vor copia 7 elemente de dimensiune specificată mai jos.

DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;

Există două posibile inițializări:

- DMA_PeripheralInc_Disable – DMA-ul nu o să treacă la următoarea adresă, o să citească mereu de la adresa specificată în primul câmp (DMA_PeripheralBaseAddr)
- DMA_PeripheralInc_Enable – DMA-ul o să incrementeze adresa de început la fiecare ciclu, cu dimensiunea specificată mai jos.

DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;

- DMA_MemoryInc_Enable – La fiecare transfer, DMA-ul va incrementa zona de memorie în care va stoca informația
- DMA_MemoryInc_Disable – DMA-ul nu va incrementa zona de memorie în care stochează informația, aceasta fiind rescrisă.

DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;

Dimensiunea datelor preluate de la periferic. Există trei posibile inițializări:

- DMA_PeripheralDataSize_Byte – 8 biți
- DMA_PeripheralDataSize_HalfWord – 16 biți
- DMA_PeripheralDataSize_Word – 32 biți

DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;

Setează dimensiunea datelor preluate din memorie. Există trei posibile inițializări:

- DMA_MemoryDataSize_Byte – 8 biți
- DMA_MemoryDataSize_HalfWord– 16 biți
- DMA_MemoryDataSize_Word– 32 biți

DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;

Există două posibile inițializări:

- DMA_Mode_Circular – Modul circular este folosit în momentul în care se dorește salvarea datelor într-o coadă sau vector circulară. Când se atinge limita setată în câmpul de mai sus (DMA_BufferSize) aceasta se resetează și revine la adresa de început (DMA_MemoryBaseAddr). De asemenea, după ce s-a terminat trasferul unui bloc de date, trasferul de date este reluat automat fără a necesita reinițializarea DMA-ului.
- DMA_Mode_Normal – Se incrementează automat până se atinge limita de date dată de DMA_BufferSize. Când această limită este atinsă, DMA-ul nu se mai resetează. Daca se dorește copierea a unui bloc de date nou, DMA-ul trebuie reinițializat.

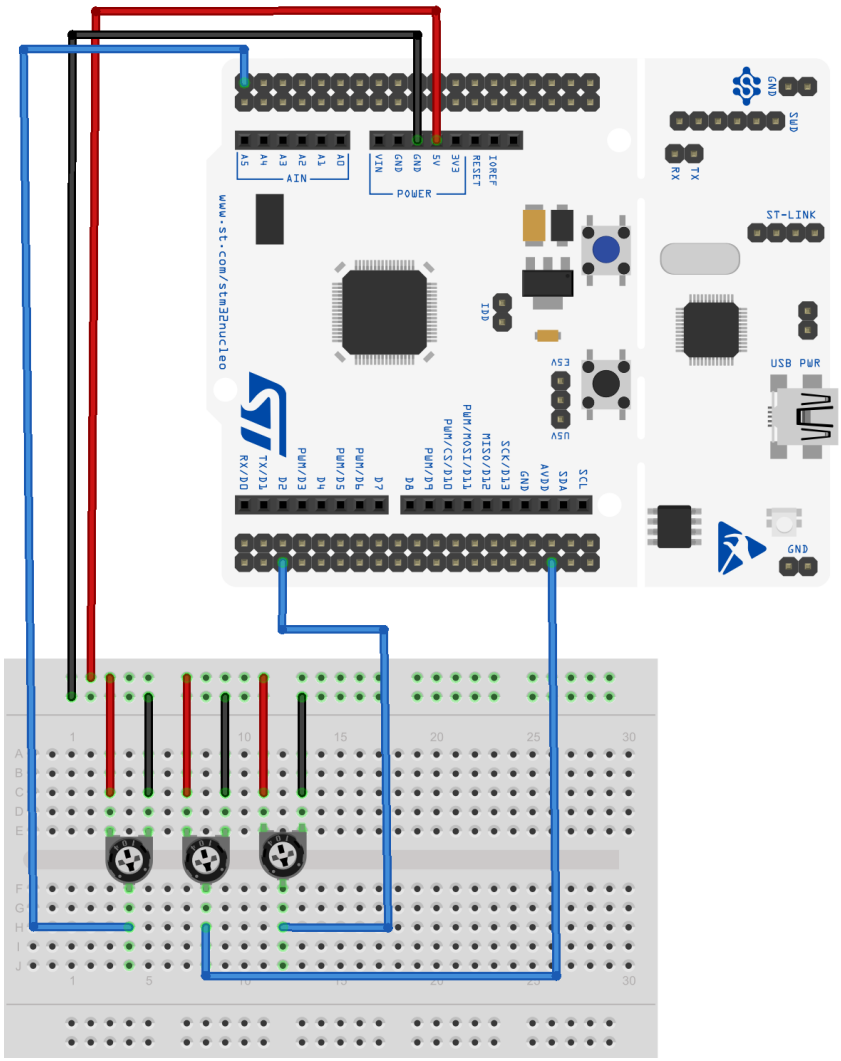
DMA_InitStructure.DMA_Priority = DMA_Priority_High;

În acest câmp se setează una din cele patru priorități ale canalului DMA.

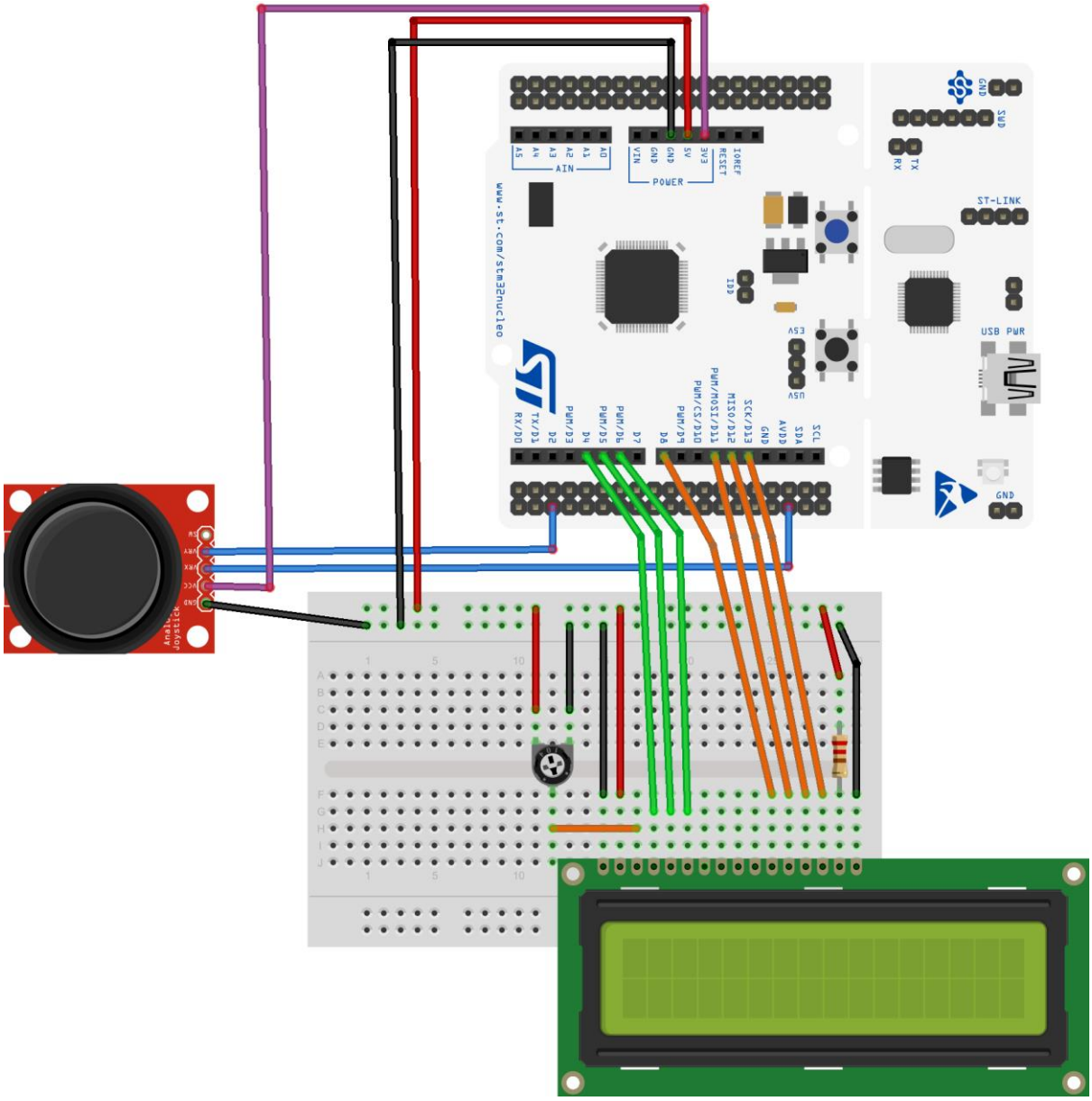
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

Acest câmp se setează pe Enable, în momentul în care se dorește trasferul de date în memorie. Cu alte cuvinte, dacă se dorește copierea unui vector în alt vector cu DMA, acest câmp trebuie să fie Enable.

Schema de interconectare 1



Schema de interconectare 2



Surse:

http://www.ni.com/white-paper/4806/en/

https://developer.mbed.org/users/borislav/notebook/serial-port-plotter/