Osnove mikroprocesorske elektronike

Priprava 11: A/D pretvornik

V sklopu te vaje bomo oživeli "joystick". V pripravi boste *poskrbeli za inicializacijo vseh perifernih enot*, s pomočjo katerih bomo zelo elegantno izvajali *avtomatske periodične meritve* pozicije "joysticka". Za uvod v pripravo si poglejmo, kako smo tokrat zastavili rešitev na strojnem nivoju.

Pristop s pomočjo DMA enote

Do sedaj ste se tekom vaj spoznali z dvema pristopoma, kako lahko *obdelujemo tok podatkov*:

1. *pristop poizvedovanja* (angl. *polling mode*)

   Tak pristop je smiseln za podatkovne tokove, ki so počasni in časovno nekritični ("polling" pristop smo na primer uporabili za branje stanja tipkovnice).

2. *pristop s pomočjo prekinitev* (angl. *interrupt mode*)

   Tak pristop je elegantnejši, saj omogoča, da se na tok podatkov *odzivamo ob dogodkih, ki prožijo prekinitve*. Uporaba prekinitev omogoči, da funkcije za obdelavo podatkovnega toka implementiramo v t. i. "non-blocking" načinu, kar pa močno zmanjša izgube procesorskega časa. Tak pristop smo uporabili pri sprejemanju podatkov preko serijskega vmesnika USART.

V sklopu te vaje pa si bomo ogledali še tretji kompleksnejši pristop:

3. *pristop s pomočjo DMA enote* (angl. *DMA mode*)

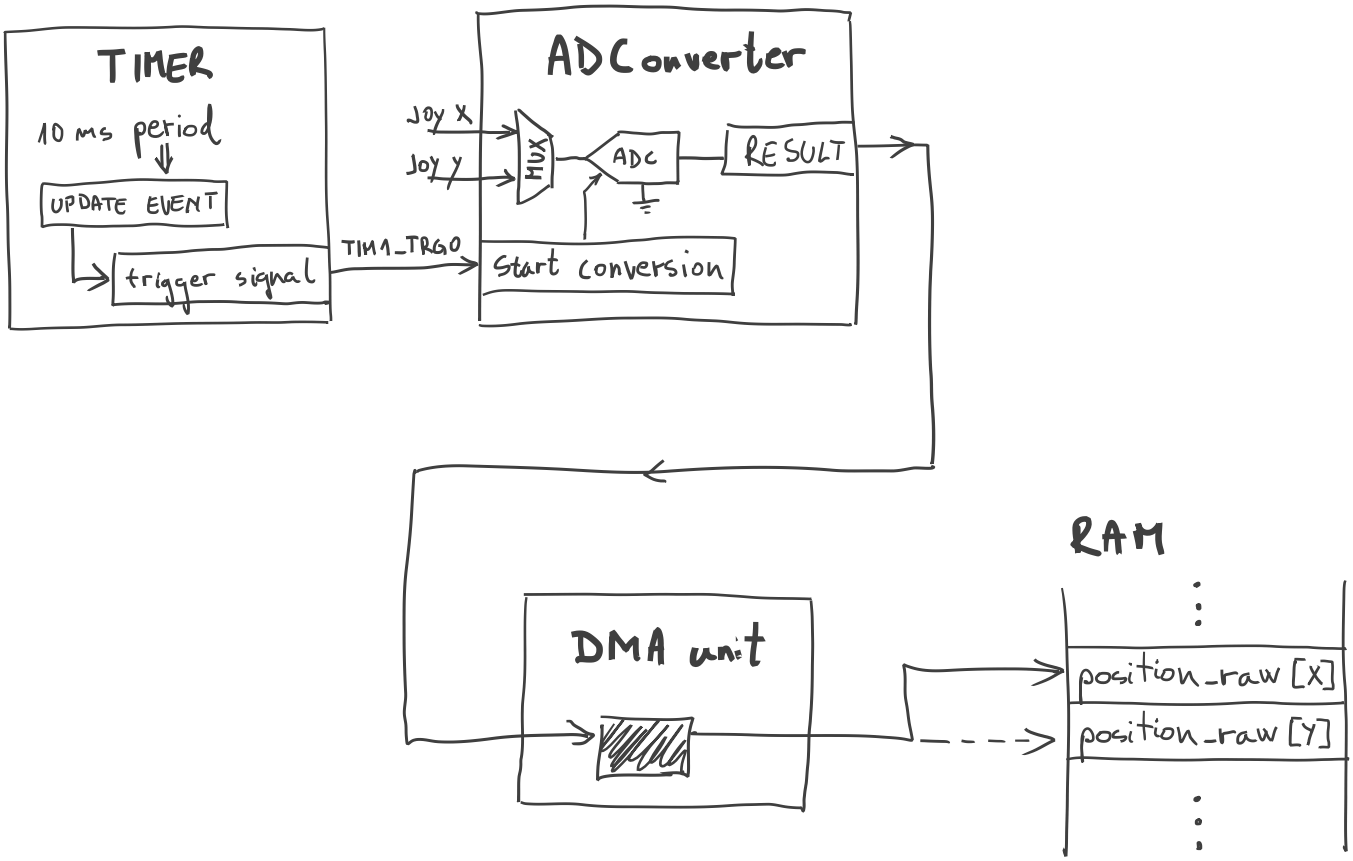
   DMA enota (angl. *Direct Memory Access unit*) je periferna enota, ki *omogoča prenos podatkov med periferno enoto* (npr. USART vmesnikom, A/D pretvornikom ipd.) *in sistemskim pomnilnikom* (tj. RAM-om), *ne da bi pri tem motila izvajanje centralne procesne enote*.

V primeru naše vaje bo DMA enota avtomatsko prenašala rezultate A/D pretvorbe (tj. meritve pozicije "joysticka") v spremenljivko v sistemskem spominu. Vaša programerska naloga bo, da v sklopu funkcij sistemskega nivoja te "*surove podatke*" (angl. *raw measurements*) preračunate v smiselne vrednosti, ki jih bo kasneje končna aplikacija (npr. igrica) lahko uporabila.

Uporaba dogodkov za generiranje notranjih prožilnih signalov

Pri tej vaji pa boste spoznali še en pomemben *mehanizem*, ki omogoči, da periferne enote delujejo *popolnoma samostojno brez motenja izvajanja centralne procesne enote*. Uporabili bomo namreč idejo *dogodka* (angl. event): *dogodek ob prelivu časovnika* bomo uporabili tako, da bo ta dogodek generiral *prožitveni signal* (angl. trigger signal), ki bo sprožil začetek A/D pretvorbe. Tako bomo s pomočjo časovnika izvajali avtomatske periodične meritve pozicije "joysticka", ne da bi pri tem motili izvajanje procesne enote.

Idejo za oživitev "joysticka" na strojnem nivoju si lahko ogledate na spodnji skici.



Še poudarek: pri tej vaji bomo za delo s strojno opremo *uporabljali knjižnice HAL* (in ne LL).

Priprava novega projekta

1. ***S pomočjo orodja*** `STM Cube Project Copy` ***ustvarite nov projekt z imenom***

   `VAJA_11-ADC`

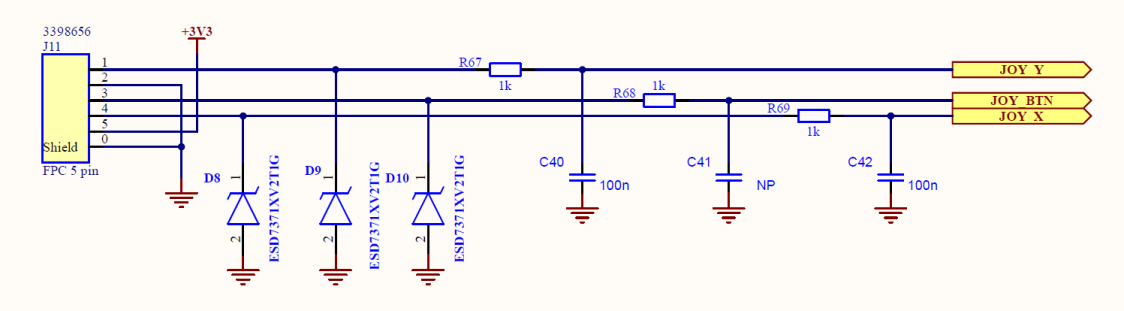
   na podlagi projekta, kjer ste rešili prejšnjo vajo. Projekt nato uvozite v "STM Cube IDE".

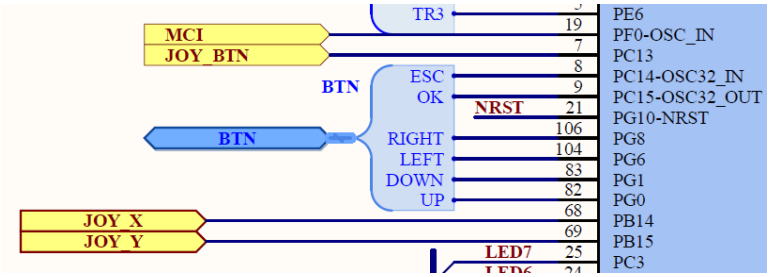
Priprava pinov mikrokrmilnika

2. ***Preučite, kateri pini mikrokrmilnika so uporabljeni za priklop "joysticka".***

   Od "joysticka" so na mikrokrmilnik pripeljani trije signali. Signala `JOY_X` in `JOY_Y` sta namenjena merjenju pozicije "joysticka". S pomočjo teh dveh *analognih* signalov bomo merili odklon X oziroma Y *osi "joysticka"* (angl. joystick axis). Tretji signal `JOY_BTN` pa je *digitalni* signal, ki pa nosi informacijo o pritisku na "joystick", ki se obnaša kot tipka, če na njega pritisnemo z vrha.

   Pomagajte si s spodnjimi izseki iz električne sheme Miškota.





3. ***V orodju CubeMX poskrbite za ustrezno nastavitev <u>funkcije</u> pinov, kamor so pripeljani trije signali "joysticka".***

   Pri tem morate poskrbeti za naslednje stvari:

   a) pin signala `JOY_BTN` nastavite kot GPIO *digitalni vhod* z vklopljenim notranjim zgornjim uporom ("pull-up resistor"),

   b) pina signalov `JOY_X` in `JOY_Y` nastavite kot *analogna vhoda* A/D pretvornika ADC4.

Priprava časovnika za proženje A/D pretvorbe

Mikrokrmilnik STM32G4 namreč omogoča, da *dogodki časovnika* (angl. timer events) kot je na primer *dogodek preliva* ("update event"), sprožijo začetek A/D pretvorbe, *ne da bi pri tem kakorkoli motili izvajanje centralne procesne enote* (tj. izvajanje programa). Poglejte izsek iz dokumentacije spodaj.

Analog-to-digital converters (ADC) **RM0440**

21.2 ADC main features

- Start-of-conversion can be initiated:
  - By software for both regular and injected conversions
  - By hardware triggers with configurable polarity (internal timers events or GPIO input events) for both regular and injected conversions

Izkaže se, da za proženje A/D pretvornika `ADC4` lahko uporabimo časovnik `TIM1` in njegov *interni signal* `TIM1_TRGO` ("trigger output"). Glejte izsek spodaj.

Analog-to-digital converters (ADC) **RM0440**

Table 165. ADC3/4/5 - External triggers for regular channels (continued)

Name	Source	Type	EXTSEL[4:0]
adc_ext_trg8	TIM8_TRGO2	Internal signal from on-chip timers	01000
adc_ext_trg9	TIM1_TRGO	Internal signal from on-chip timers	01001
adc_ext_trg10	TIM1_TRGO2	Internal signal from on-chip timers	01010

4. ***V orodju CubeMX poskrbite za ustrezne nastavitve časovnika*** `TIM1`***, ki ga bomo uporabili za periodično proženje A/D pretvornika*** `ADC4`**.**

   Vaša naloga je torej, da poskrbite za sledeče nastavitve časovnika `TIM1`:

   a) za vir ure časovnika izberite notranjo uro (angl. internal clock).

   b) *modul* časovnika naj bo 10 milisekund (angl. counter period). Tako bomo zajemali informacijo o legi "joysticka" vsakih 10 milisekund, kar se zdi dovolj pogosto.

   c) V sekciji "Trigger Output (TRGO) Parameters" določite, da se signal `TIM_TRGO` generira ob *dogodku preliva* ("update event"). Poglejte spodaj.

   Tako bo časovnik `TIM1` generiral *prožilni signal* ("trigger signal"), ki bo uporabljen za sprožitev A/D pretvorbe.

Sedaj je časovnik `TIM1` pripravljen, da se ga uporabi za proženje A/D pretvorbe časovnika `ADC4`.

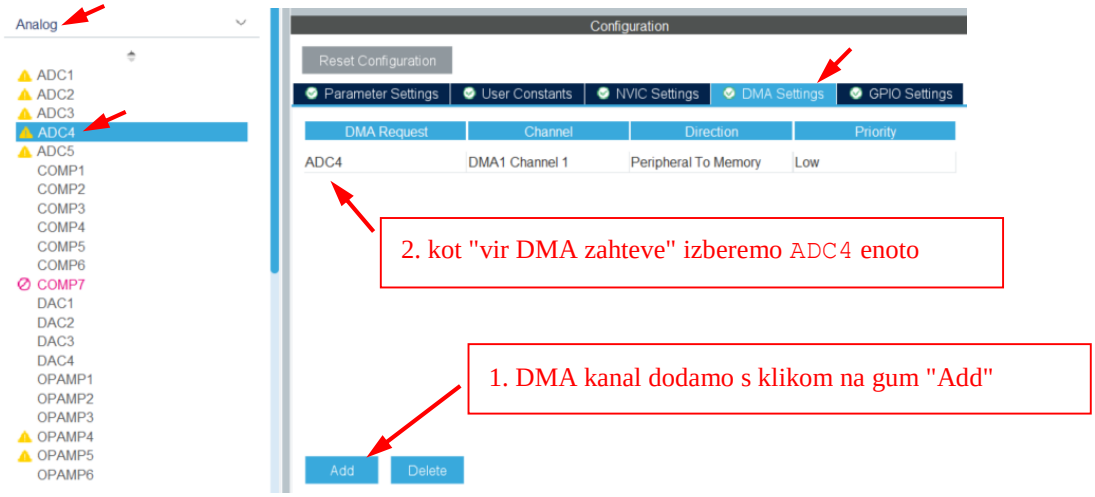
Priprava "Direct Memory Access" enote

Za shranjevanje rezultatov A/D pretvorbe bomo uporabili posebno periferno enoto, ki se s tujko imenuje "*Direct Memory Access*" *enota* (DMA enota). Ta enota bo poskrbela, da se ob končani A/D pretvorbi *rezultati A/D pretvorbe prenesejo iz podatkovnega registra A/D pretvornika v točno določeno lokacijo v RAM spominu mikrokrmilnika*. Ta *prenos podatkov se zgodi tako, da se pri tem ne moti izvajanja centralne procesne enote* (tj. izvajanja programa).

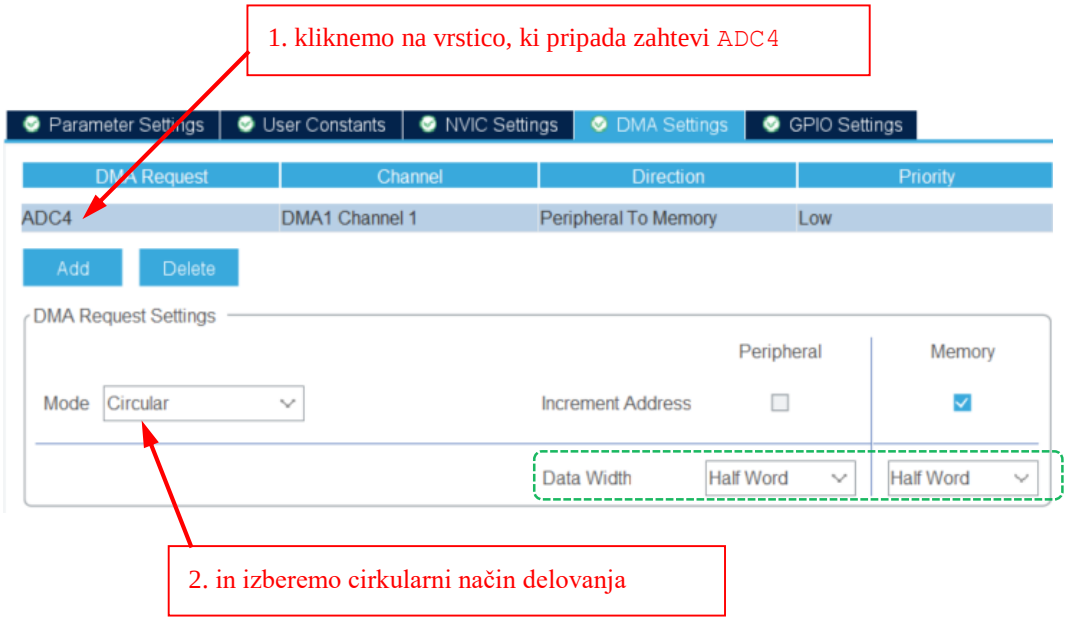
5. ***V orodju CubeMX poskrbite za ustrezne nastavitve DMA enote, ki bo poskrbela za avtomatsko shranjevanje rezultatov A/D pretvorbe*** `ADC4`**.**

   Nastavitve DMA kanala (angl. DMA channel) za prenos podatkov A/D pretvornika se ureja znotraj nastavitev, ki se tičejo A/D pretvornika v zavihku "DMA Settings" (poglejte spodaj).

   Najprej bomo A/D pretvorniku dodelili DMA kanal. Sledite korakom, ki so nakazani na spodnji sliki.



Sedaj pa je potrebno še nastaviti, v katerem načinu naj deluje DMA enota v primeru A/D pretvornika `ADC4`. Sledite korakom s spodnje slike.



Cirkularni način delovanja DMA enote zagotovi, da bo DMA enota po prvem prenosu podatkov iz registra A/D pretvornika v RAM spomin ob naslednji zahtevi za prenos *celoten postopek ponovila* na popolnoma enak način.

Mimogrede lahko opazite, da bo DMA enota prenašala podatke širine "half word" (zeleni del na sliki), torej podatke široke 16 bitov. To je smiselno, saj je rezultat A/D pretvorbe dolg 12 bitov (uporabljamo 12-bitni A/D pretvornik).

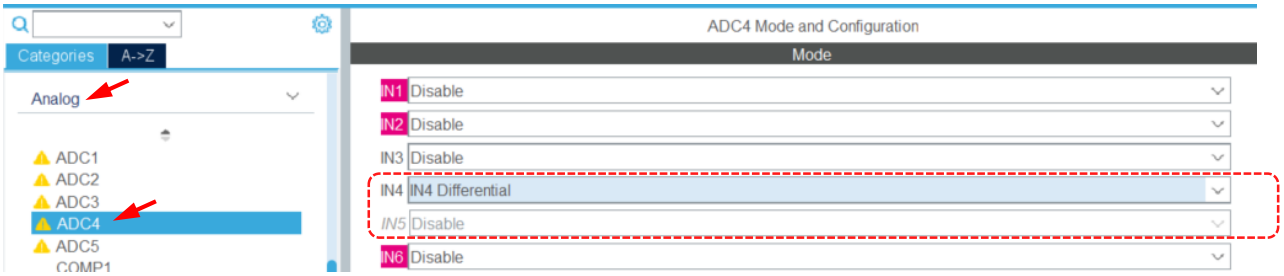
Sedaj je pripravljena tudi DMA enota za delo z A/D pretvornikom `ADC4`. V naslednjem koraku pa bomo nastavili še delovanje pretvornika `ADC4`.

Priprava A/D pretvornika

6. ***V orodju CubeMX poskrbite za ustrezne nastavitve parametrov A/D pretvornika*** `ADC4`**.**

   Nastavitve parametrov A/D pretvornikov se ureja znotraj rubrike "Pinout & Configuration → Analog".

   Najprej je potrebno *pravilno nastaviti vrsto analognih vhodov*: ali gre za diferencialni vhod (angl. *differential input*) ali za "enojni vhod glede na maso" (angl. ground-referenced *single-ended input*). V našem primeru sta oba signala "joysticka" `JOY_X` in `JOY_Y` "enojna" in referirana proti masi (tj. GND potencialu v vezju). Nastavite torej vrsto obeh vhodov kot "single-ended".



   Sedaj pa sledijo nastavitve A/D pretvornika (rubrika "Parameter Settings"). Poskrbite, da nastavite sledeče parametre:

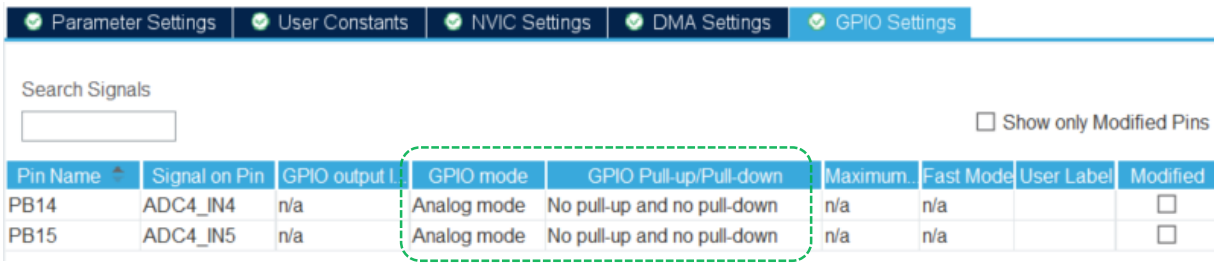
   1) število pretvorb ("Number Of Conversion") = 2, saj želimo zajemati dva signala "joysticka".
   2) Omogočite "Scan Conversion Mode", kar pomeni, da bomo ob sprožitvi pretvorbe pretvorili *oba* vhodna signala *le enkrat*.
   3) Omogočite "DMA *Continuous* Request", kar zagotovi, da A/D pretvornik lahko uporablja DMA enoto *periodično* (in ne le enkrat).
   4) Nastavite vir proženja A/D pretvorbe "External Trigger Conversion Source" na "Timer 1 Trigger Out event". Tako poskrbimo, da bo časovnik `TIM1` s svojim internim signalom `TIM1_TRGO` periodično prožil A/D pretvorbe.

5) V sekciji "Rank 1" nastavite "Channel = 4" (za pretvorbo 4. vhodnega kanala A/D pretvornika).
6) V sekciji "Rank 1" nastavite "Sample time = 640.5 Cycles" (najdaljša pretvorba). Najdaljši čas A/D pretvorbe zagotovi največjo vhodno impedanco analognega vhoda A/D pretvornika.

7) V sekciji "Rank 2" nastavite "Channel = 5" (za pretvorbo 5. vhodnega kanala A/D pretvornika).
8) V sekciji "Rank 2" nastavite "Sample time = 640.5 Cycles" (najdaljša pretvorba).

Preverite zgornje nastavitve s pomočjo slike spodaj.

Mimogrede, v zavihku "GPIO Settings" lahko vidite, kako se bo nastavila funkcija pinov, ki so uporabljeni kot analogni vhodi. Poglejte spodaj.



Pin Name	Signal on Pin	GPIO output l	GPIO mode	GPIO Pull-up/Pull-down	Maximum..	Fast Mode	User Label	Modified
PB14	ADC4_IN4	n/a	Analog mode	No pull-up and no pull-down	n/a	n/a		☐
PB15	ADC4_IN5	n/a	Analog mode	No pull-up and no pull-down	n/a	n/a		☐

Priprava – rešitev "hrošča" v CubeMX generirani kodi

7. ***Shranite nastavitve v orodju CubeMX in potrdite avtomatsko generiranje nove kode.***

   Podobno kot pri prejšnjih vajah, ste s tem korakom poskrbeli, da se bo dodala nizko-nivojska programska koda za inicializacijo strojne opreme, ki jo bomo uporabljali za implementacijo sistemskega modula za delo z "joystickom":

   - inicializacija A/D pretvornika `ADC4`,
   - inicializacija DMA enote, ki bo prenašal rezultate A/D pretvorbe v RAM,
   - inicializacija časovnika `TIM1`, ki bo periodično prožil A/D pretvorbe.

   Poglejte izsek spodaj.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
MX_TIM6_Init();
MX_TIM4_Init();
MX_FMC_Init();
MX_SPI1_Init();
MX_ADC4_Init();
MX_DMA_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
```

   Vendar se izkaže, da je v tej avtomatsko generirani programski kodi težava. Izkaže se, da ***orodje CubeMX inicializira periferne enote v napačnem vrstnem redu!*** Poglejte opis težave na uradnem STM forumu spodaj.

STM32 MCUs (Archived) — tinkerer42 (Customer) asked a question.
October 21, 2019 at 8:06 PM

[BUG REPORT] DMA and ADC initialization order changed in STM32F4 HAL v1.24.1, causing incorrect ADC operation

I upgraded my CubeIDE to v1.1 and it upgraded the integrated CubeMX as well to v5.4, and also the HAL firmware version. re-generating the CubeMX code resulted in a few seemingly minor changes - among them, the order of these calls has changed in main():

```
MX_DMA_Init();
MX_ADC1_Init();
```

the correct order is as above (DMA initialized first), and this is what the previous CubeMX generated. However, in the latest version, the order of these two calls was switched over, **resulting in the ADC not working correctly** (in DMA mode). I can confirm that simply switching the order of these two lines back to what it was in the previous firmware causes the ADC to work correctly with the DMA again. It makes sense too, since in the MX_ADC1_Init() function there are calls to DMA related stuff. Surely if the DMA is re-initialized after this DMA related setup done in MX_ADC1_Init(), then it overrides that setup.

Da bi A/D pretvornik pravilno deloval v navezi z DMA enoto, je *potrebno zagotoviti, da se DMA enota inicializira še preden se inicializira A/D pretvornik*. In kot vidite, to v zgornji kodi ne drži!

To težavo bomo rešili v naslednjem koraku.

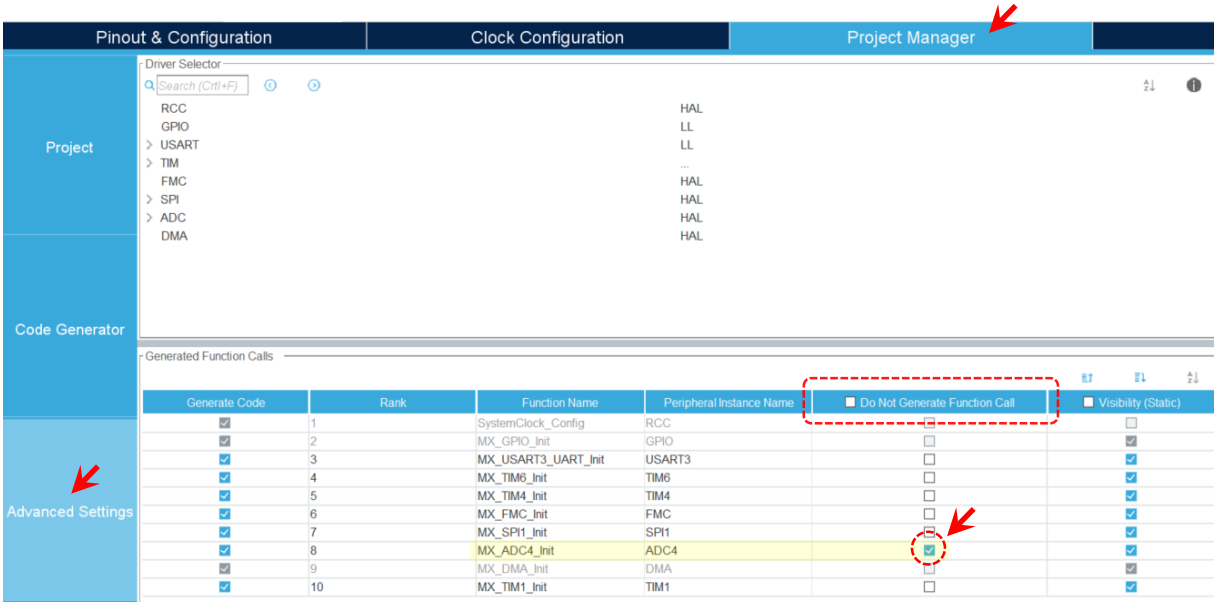
8. ***Odpravite težavo z napačnim vrstnim redom inicializacije perifernih enot.***

Kako lahko popravimo vrstni red inicializacije perifernih enot, če se ta koda zgenerira avtomatsko?

Ideja: kaj če bi *za klic inicializacijske funkcije* `MX_ADC4_Init()` *poskrbeli mi* sami in tega ne bi prepustili avtomatsko generirani kodi? Težavo bomo torej rešili tako, da bomo mi sami poskrbeli, da se inicializacija A/D pretvornika izvede šele po inicializaciji DMA enote.

To bomo storili tako, da

a) ***v orodju CubeMX najprej preprečimo, da bi avtomatsko generirana koda vsebovala tudi klic inicializacijske funkcije*** `MX_ADC4_Init()`. Poglejte sliko spodaj.



b) ***Shranite nastavitve v orodju CubeMX in potrdite avtomatsko generiranje nove kode.***

Sedaj preverite, če sedaj avtomatsko generirana koda res ne vsebuje klica za inicializacijo A/D pretvornika `MX_ADC4_Init()`. Sedaj bi morala avtomatska koda znotraj funkcije `main()` izgledati nekako takole:

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
MX_TIM6_Init();
MX_TIM4_Init();
MX_FMC_Init();
MX_SPI1_Init();
MX_DMA_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
```

V naslednjem koraku je torej le še potrebno ročno dodati inicializacijo A/D pretvornika.

c) ***Dodajte klic funkcije za inicializacijo A/D pretvornika*** `MX_ADC4_Init()` ***pod avtomatsko generirano kodo***.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
MX_TIM6_Init();
MX_TIM4_Init();
MX_FMC_Init();
MX_SPI1_Init();
MX_DMA_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */

MX_ADC4_Init();   // ADC4 inicializiramo ročno po inicializaciji DMA enote!
```

Pazite, da klic dodate znotraj t. i. "comment guards-ov", torej znotraj sekcije

```
/* USER CODE BEGIN */

/* USER CODE END  */
```

Zaključni komentar

Tako ste sedaj poskrbeli, da se bo vsa potrebna strojna oprema pravilno inicializirala.

Dela z inicializacijo strojne opreme je bilo tokrat več, saj za implementacijo "joystick" modula uporabljamo zahtevnejšo *strojno rešitev z uporabo DMA enote*. Po drugi strani pa tak pristop *poenostavi programiranje na sistemskem nivoju*, saj se bodo sedaj *s pomočjo DMA enote meritve pozicije "joysticka" avtomatsko znašle v spremenljivki "handle" strukutre "joysticka".*