Laborator 2
Comunicaţia serială între placa Nucleo şi PC

Secţiuni lucrare:

Echipamente necesare lucrării:

- Placă de dezvoltare NUCLEO F103RB
- Cablu mini USB

CMSIS – Cortex Microcontroller Software Interface Standard

După cum sugerează și numele, CMSIS este un strat de abstractizare software sau un set de biblioteci standardizate pentru familia de procesoare Cortex M. În figura 1 se poate vedea că CMSIS este compus în mare parte din funcții low level care apelează diverse blocuri hardware ale procesorului. În același timp oferă un API (Application Programming Interface) standardizat pentru diverse aplicații embedded și pentru sistemele de operare în timp real, oferindu-le o modalitate de a accesa modulele hardware ale procesorului și perifericele cipului.

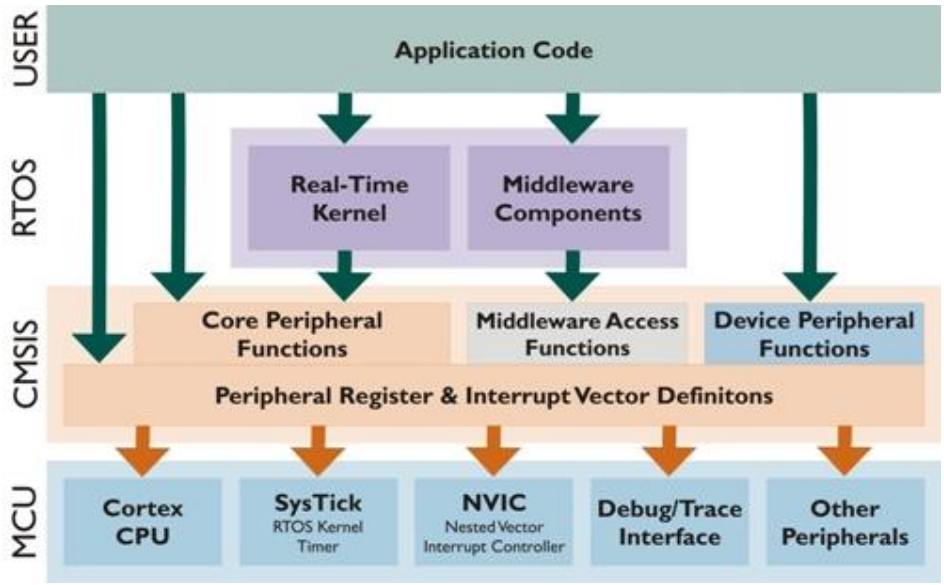


Figura 1. Plasarea CMSIS pe o arhitectură de tip ARM

CMSIS oferă dezvoltatorilor posibilitatea de a scrie bucăți de cod reutilizabil pentru orice procesor din familia ARM Cortex-M. Cu alte cuvinte, un program scris pentru un Cortex M0 de la STMicroelectronics poate să fie portat cu modificări foarte mici (aproape zero) pe un Cortex M3 de la Texas Instruments.

Mediul CMSIS oferă programatorilor un set de funcții și un set de tipuri de date care sunt comune pentru orice familie de ARM Cortex M. Un asemenea tip de data este GPIO_InitTypeDef. Variabilele GPIO (General-purpose input/output) se ocupă cu setarea pinilor microcontrolerului.

Pentru a controla dispozitivele sau pentru a citi o valoare de intrare trebuie ca pinii GPIO să fie setaţi corespunzător. În continuare se va prezenta un model de configurare a acestora.

- se declară o structură GPIO_InitTypeDef
- se porneşte semnalul de ceas pentru magistrala pe care se află perifericul folosit
- se setează câmpurile structurii astfel:
  - GPIO_Pin – specifică pinul GPIO care este configurat (de exemplu GPIO_Pin_5)
  - GPIO_Speed – setează viteza pentru pinul specificat. Poate avea valorile:

    GPIO_Speed_10MHz;

    GPIO_Speed_2MHz;

    GPIO_Speed_50MHz;
  - GPIO_Mode – specifică modul de funcţionare al pinului; el poate fi de intrare sau ieşire şi se va configura din program.

Figura 2 arată structura de bază a unui pin. Fiecare pin poate să fie atât digital, cât și analogic, de intrare sau de ieșire. Cu alte cuvinte, funcția pinului este aleasă de programator.

Figure 13. Basic structure of a standard I/O port bit

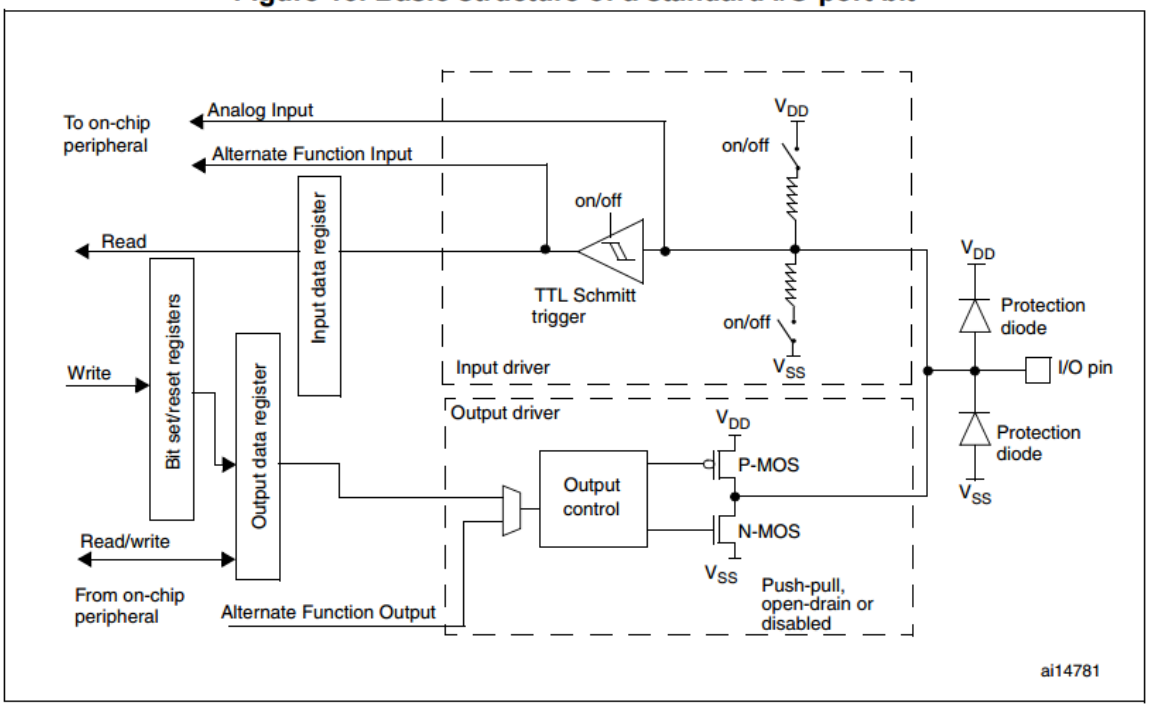


Figura 2. Structura de bază a unui pin (**Referance manual pag 159 v16**)

Un pin poate fi configurat, după cum urmează:

- **GPIO_Mode_AIN** - Se alege atunci când pin-ul respectiv va fi folosit pentru intrări analogice.
- **GPIO_Mode_IN_FLOATING** - Input floating: pinul este lăsat floating (până nu este tras la un nivel de tensiune din exterior), are o impedanţă mare, nu are rezistentă internă de pull up/pull down (se foloseşte o rezistenţă externă de pull up/down)
- **GPIO_Mode_IPD** - Input pull-down: Pinul ete configurat ca input, având activată o rezistenţă de pull-down;
- **GPIO_Mode_IPU** - Input pull-up: Pin-ul ete configurat ca input, având activată o rezistentă de pull-up;
- **GPIO_Mode_Out_OD** - General purpose output - Open-drain: pin de output care are dezactivat tranzistorul P-MOS, tranzistorul N-MOS are sursa la masa, iar drena nu este conectata. Cele mai multe aplicaţii, inclusiv magistrala I2C folosesc rezistenţe externe de pull-up.
- **GPIO_Mode_Out_PP** - General purpose output - Push-pull: se folosesc ambii tranzistori, fiecare activaţi în functie de nivelul de tensiune dorit. Tranzistorul de sus (P-MOS) va fi deschis atunci când la ieşire trebuie să fie „1 logic" respectiv tranzistorul N-MOS va fi deschis când la ieşire trebuie să fie „0 logic".
- **GPIO_Mode_AF_OD** - Alternate Function output - Open-drain: pe lângă funcţia lor de bază (care este în mod implicit după resetare), pinii mai pot fi configuraţi pentru a avea şi alte funcţii, printre care I2C, SPI, PWM, USART etc.
- **GPIO_Mode_AF_PP** - Alternate Function output - Push-pull

- se iniţializează structura conform parametrilor specificaţi:

GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct), unde primul parametru, GPIOx, specifică portul, iar celălalt parametru este pointer la structura care conţine informaţiile de configurare. Pentru a modifica valoarea unuia sau mai multor pini, se vor folosi regiştrii BSRR (Port Bit Set/Reset register) şi BRR (Port Bit Reset register).

Semnalul de ceas (Clock)

Semnalul de clock sau semnalul de ceas, este folosit de toate perifericele din familia STM32. Fiecare modul foloseşte un semnal de ceas independent pentru a respecta condițiile de low power (un consum redus de energie). Prin urmare, semnalul de ceas este o componentă importantă din arhitectura procesorului și trebuie studiată în detaliu. Microcontrolerul STM32F013 are 5 surse de clock:

- **HIS** – High Speed Internal este generat de un oscilator intern RC de 8MHz și poate să fie folosit direct ca și ceas de sistem sau să fie împărțit la doi si să fie folosit ca intrare pentru PPL. Avantajul HIS este că oferă un semnal de ceas cu costuri foarte mici. Ca și principal dezavantaj, frecvența lui poate să varieze în funcție de temperatura ambientală. Pentru procese care necesită un semnal de clock precis, o etapă de calibrare este necesară.
- **HSE** – High Speed External poate fi generat din două surse: cristal/resonator ceramic extern sau o sursă externă de ceas.
- **LSI** – Low Speed Internal este generat de un oscilator RC, low power, care este folosit pentru modul de Stop și Standby ale watchdog-ului si pentru modulul de Auto-Wakeup. Frecvența clock-ului este de aproximativ 40 KHz.
- **LSE** – Low Speed External este un semnal generat de un cristal cu o frecvență de 32.768 KHz sau de un oscilator ceramic. Este folosit pentru a genera un semnal de ceas foarte precis pentru perifericiele de timp real (RTC).
- **PLL** – Phase Locked Loop poate să aibă ca intrări fie semnalul de ceas provenit de la HIS fie semnalul provenit de la un oscilator extern HSE. Fiecare PLL trebuie configurat înainte să fie pornit (trebuie setată sursa semnalului de ceas și factorii de multiplicare sau de divizare).

Câteva circuite de scalare (prescalers) permit configurarea frecvenței AHB, a magistralei de viteză mare APB2 și a celei de viteză mai scăzuta APB1. Frecvența maximă pentru AHB și APB2 este de 72 MHz, iar pentru APB1 este de 36 MHz. Toate semnalele de ceas pentru periferice sunt derivate din semnalul de ceas principal (System Clock sau SYSCLK), cu excepția memoriei flash care tot timpul este conectată la HIS, USB, I2S și Ethernet.

Atât semnalele HSI, HSE și PPL pot fi folosite ca semnal de clock pentru întregul sistem, pe când LSI și LSE sunt semnale mai lente ce nu pot fi folosite pentru semnalul de ceas al întregului sistem. Arhitectura semnalelor de ceas se poate vedea în figura 3. Se observă că există mai multe semnale de ceas pentru nucleul Cortex și pentru periferice. De ce? Dacă nu este specificat un semnal de ceas pentru SYSCLK, se va alege HIS ca semnal de ceas pentru întregul sistem. În cazul în care se dorește setarea unui alt semnal de ceas, aceasta trebuie făcută înainte de începerea programului.

Un prim motiv este dat de viteza acestor semnale. Semnalele de ceas cu frecvență mare sunt folosite pentru dispozitive cu frecvență mare, precum CPU, iar semnalele cu viteză scăzută sunt folosite pentru dispozitive mai lente, cum ar fi perifericele. Cel de al doilea motiv este dat de caracteristica de low power a microcontrolerului.

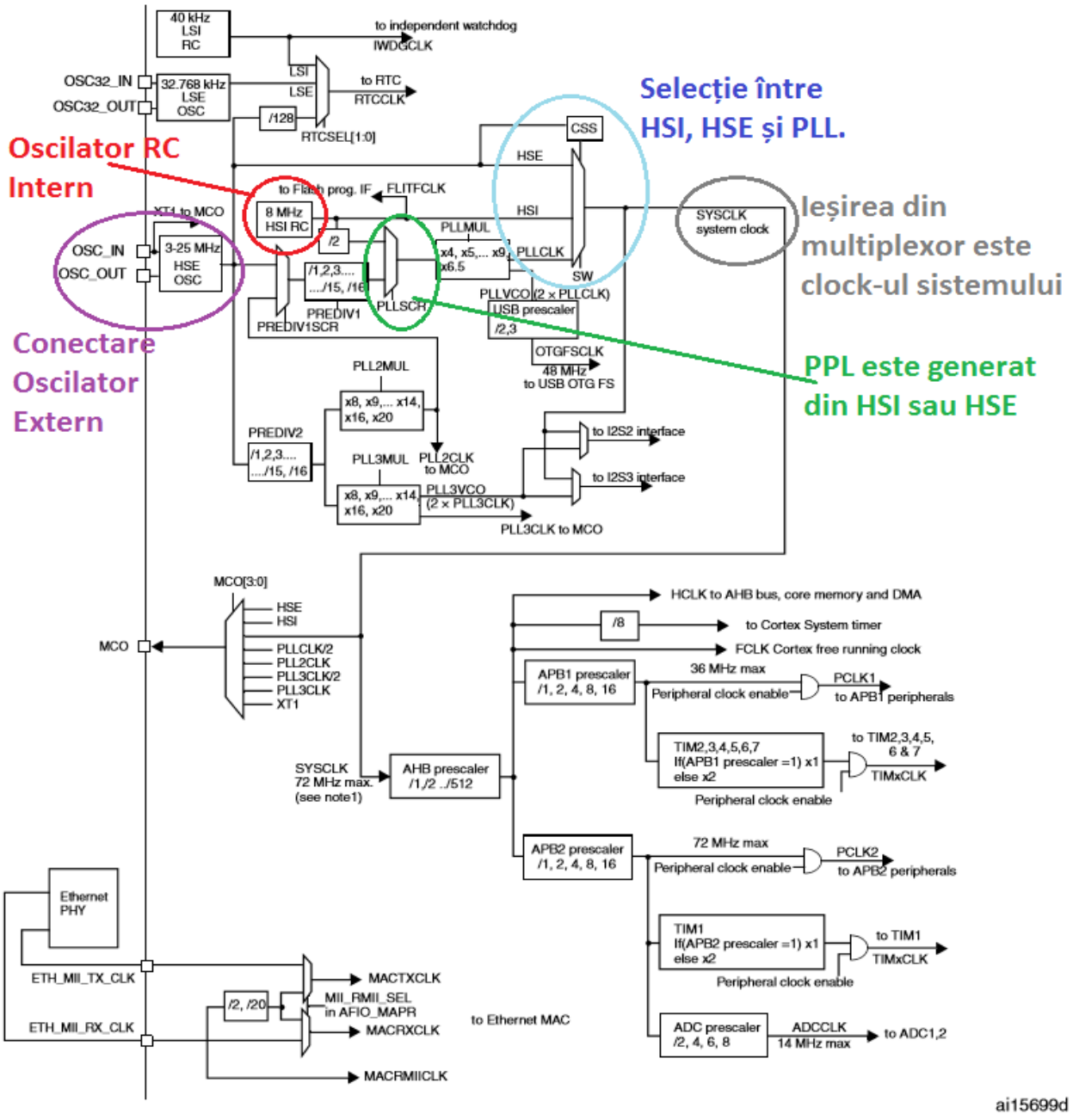


Figura 3. Arhitectura semnalului de ceas (STM32F1xx Referance Manual Rev 16, pag 125)

Atenție!

Toate perifericele microcontrolerului sunt inițial oprite, pentru a reduce consumul de energie. În momentul în care dorim să lucrăm cu un anumit periferic (pin, magistrală de date,

comunicație usart etc.), trebuie să se activeze clock-ul pentru acesta. În programul din laboratorul curent veţi vedea câteva exemple.

USART

În cele ce urmează, vom explora modalități în care două dispozitive pot comunica. Vom lua ca exemplu un dispozitiv A și un dispozitiv B, dispozitive ce vor comunica prin intermediul unui canal. O primă problemă este ridicată de ***sincronizare***. Sincronizarea, în termeni simpli, este un proces în cadrul căruia se realizează o conexiune prin care se pot trimite date între două dispozitive caracterizate de diverse frecvenţe. Pot exista situaţii în care dispozitivele operează la aceeaşi frecvenţă sau situaţii în care un dispozitiv este mai rapid decât celălalt.

Cele două dispozitive trebuie să fie capabile să își transmită unul altuia informația în toate situațiile prezentate mai sus.

A și B pot fi calculatoare, precum un PC sau un microcontroler, sau periferice, asemenea unui ecran LCD. Există mai multe metode prin care două dispozitive pot să se sincronizeze.

1. Cea mai simplă metodă se numește "**Blind-Cycle Synchronization**". Să presupunem că dispozitivul A este mult mai rapid decât dispozitivul B. În acest caz, dispozitivul A va trimite un mesaj către B și va aştepta un timp predefinit (Δt) până va trimite următorul mesaj. Timpul (Δt) este de obicei setat de frecvența dispozitivului B, parametru ce se poate găsi în datasheet-ul dispozitivului. Aceasta metodă se folosește în momentul în care dispozitivul B nu este foarte complex. Cu toate că este lentă, va funcţiona tot timpul dacă viteza de comunicare este fixată conform datasheet-ului dispozitivului B.
2. Cea de-a doua metodă poartă numele de "**Busy Waiting**". Dacă dispozitivul B este mai avansat și capabil să semnaleze prin intermediul unui *flag* dispozitivului A că lucrează la primirea semnalului, atunci A trebuie să aștepte B să termine de procesat înainte să se trimită următoarea informație. Prin urmare, A o să verifice tot timpul *flag*-ul lui B, iar

când acesta a terminat de procesat, se va trimite urmatoarea informație. Această metodă este una mai bună, dar presupune ca dispozitivul B să fie capabil să indice dispozitivului A faptul că procesează sau că a terminat de procesat, lucru pe care nu toate dispozitivele îl pot face.

3. Cea de a treia metodă se realizează prin **întreruperi**. Aceasa pornește de la metoda 2 și presupune că dispozitivul B este capabil să genereze un *flag*. Acest *flag* va genera în dispozitivul A un eveniment hardware, anume o întrerupere, ce va lansa în execuție o funcție de tratare a întreruperii (ISR- Interupt Service Routine). Cu alte cuvinte, tranziția *flag*-ului din B de la 1 la 0 va genera un eveniment hardware, ce pornește la rândul său o secvență de cod scrisă special de programator. Scopul acesteia este de a trimite următorul mesaj către B, care tocmai a terminat de procesat informația recepționată anterior. Cu alte cuvinte, tranziția *flag*-ului din B de la 1 la 0 va genera un eveniment hardware, care la randul lui va lansa în execuție o funcție care va trimite următorul mesaj dispozitivului B pentru că acesta tocmai ce a terminat de procesat mesajul anterior. Această metodă este mai avansată față de anterioarele, dar presupune că dispozitivul A este capabil să gestioneze întreruperi externe şi că dispozitivul B este capabil să îi comunice dispozitivului A starea *flag*-ului.
4. Metoda **Periodic Pooling** verifică periodic statusul dispozitivelor. Poate folosi ceasul pentru a genera periodic întreruperi în care să verifice dacă dispozitivele au setat *flag*-ul care semnalizează ca poate avea loc transferul de date.
5. Controlerul de **DMA (Direct Memory access)** poate citi sau scrie în memorie date de intrare, independent de programul principal. În cazul UART se pot transfera datele din/în memorie fără ca procesorul să se ocupe de fiecare cuvant, în aşa fel făcându-se transferul datelor foarte rapid între memorie şi periferice, fară intervenția CPU.

În acest laborator, se va folosi a doua metodă, **Busy Waiting**.

Transmiterea datelor prin USART

Modulul de USART de pe microcontrolerul STM32F1xx permite un control ridicat asupra datelor transmise. Se poate opta pentru transmiterea unui cuvânt de 8 sau 9 biți, pentru un bit de paritate pentru verificarea datelor trimise și se poate alege numărul de biți pentru semnalul de stop.

Comunicația începe prin a trage linia de la o tensiune de 3.3 V la 0 V pentru un timp T. În acest timp, se va porni semnaul de ceas, pentru citirea fiecărui bit. Acest timp, va seta Baud Rate. Baud Rate reprezintă numărul de biți ce vor fi transmiși într-o secundă. Valorile întâlnite mai des sunt 4800, 9600, 19200, 115200 și altele. Secvența biților ce vor fi transmiși se poate vedea în figura 4.

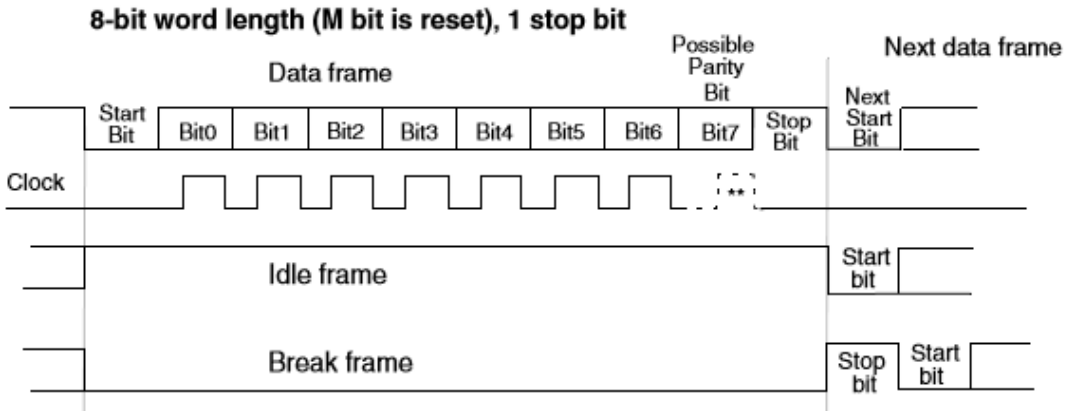


Figura 4. Lungimea cuvântului de 8 biți. (**Word lenght programming,referance manual Rev 16 pag. 795**).

Interfațare între interfața USB a calculatorului și placa Nucleo

Comunicarea cu microcontrolerul plăcii Nucleo se face prin intermediul debugger-ului (figura 5). Microcontrolerul de pe acest debuger conține un program care poate să realizeze o legătură între protocolul USB și protocolul USART. În cazul în care există doar microcontrolerul și nu există și un debugger, comunicația între microcontroler și un PC se poate realiza prin intermediul unui convertor USB-to-Serial sau FTDI. Placa Nucleo poate comunica cu PC-ul prin intermediul unui driver care trebuie instalat pe calculator (link-ul de unde poate fi descărcat se găsește în laboratorul 1). De asemenea, pentru a vizualiza mesajele transmise prin conexiunea serială, pe PC trebuie să existe un terminal. Există o multitudine de terminale pe piață și sunteți încurajați să folosiți terminalul cu care sunteți obișnuiți, dar exemplele din acest laborator și din

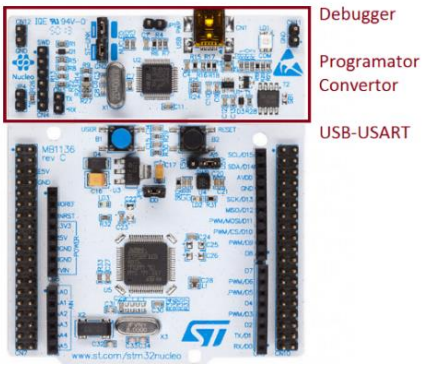


Figura 5. Debugger Nucleo

cele ce urmează o să fie dezvoltate în jurul terminalului Hterm. Acesta poate fi descărcat de la această adresă: http://www.der-hammer.info/terminal/ .

Program

Exemplu de program de comunicaţie serială

```
void initUsart()
{
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART2, &USART_InitStructure);
  USART_Cmd(USART2, ENABLE);
}
```

Pentru a putea iniția comunicația USART, primul pas care trebuie făcut este pornirea ceasului de pe magistrala de date unde se află și USART-ul pe care dorim să îl folosim. În cazul nostru, trebuie să localizăm USART2 în arhitectura sistemului. După cum se poate vedea în figura 6, USART2 se află pe APB1 (Advanced Peripheral Bus).

Se va folosi structura oferită de bibliotecile CMSIS pentru a face setarea comunicației USART. În acest sens, s-a creat o variabilă de tipul *USART_InitTypeDef* numită USART_InitStructure. Aceasta conține toate câmpurile relevante comunicației seriale. Primul câmp este USART_BaudRate = 115200. Aici se setează viteza la care dispozitivul va transmite datele. Al doilea câmp, USART_WordLenght = USART_WordLength_8b, setează câmpul de date la 8 biți. Modulul USART de pe familia de microcontrolere STM32F1xx permite transmiterea datelor formate din 8 sau 9 biți. Al treilea câmp, USART_StopBits = USART_StopBits_1, este dat de numărul de biți care indică terminarea trasmiterii datelor. În acest caz se va folosi doar 1 bit, dar pot să existe și 2 biți. Cel de al patru-lea câmp este USART_Parity = USART_Parity_No, însă modulul de USART poate să insereze la sfârșitul comunicației un bit de paritate.

Următorul câmp, USART_HardwareFlowControl= USART_HardwareFlowControl_None se referă la standardul RS232. Conectorul de RS232 are doi pini RTS (Request to Send) și CTS

(Clear to Send). Acești doi pini anunță fiecare dispozitiv de starea celuilalt. În cazul de față nu se va folosi o mufă RS232, prin urmare nu este nevoie de acest mod. Următoarea linie USART_Mode = USART_Mode_Rx | USART_Mode_Tx setează modulul USART să fie capabil să primească informații dar şi să le trimită, prin pinii de RX - receive și TX – transmit.

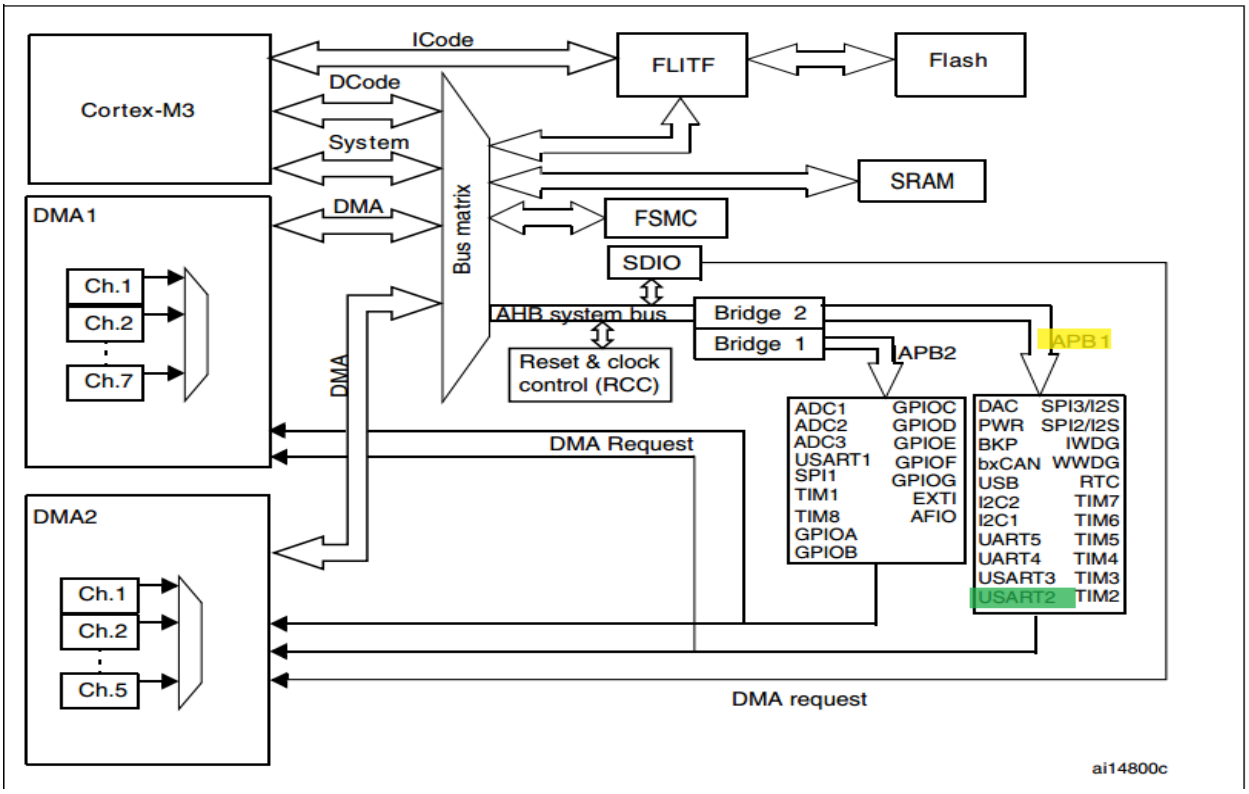
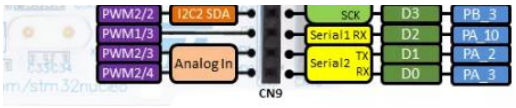


Figura 6. Arhitectura Sistemului (Referance Manual, pag. 46)

```
void initGPIO()
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```



Funcţia *initGPIO()* configurează pinii de RX şi TX de pe USART2, după modelul prezentat la începutul lucrării. Se observă ca RX este pe pinul PA_3 iar TX pe pinul PA_2.

În cele ce urmează se va folosi metoda Busy Waiting pentru a realiza comunicația serială între microcontroler și PC. Prima funcție folosită este cea care trimite un string.

```
void sendString(char a[])
{
   int i,n;
   n=strlen(a);
  for (i=0;i<n;i++)
 {
    while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
    USART_SendData(USART2, a[i]);
  }
}
```

Funcția primește ca parametru un vector de caractere și trimite fiecare caracter în parte pe serial. Cu ajutorul funcției strlen(a) se află câte caractere are vectorul *a*, iar apoi se face parcurgerea vectorului trimițându-se fiecare caracter pe serial. Linia de cod while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET); așteaptă ca dispozitivul B, să termine de procesat informația trimisă anterior până să se trimită un nou caracter. Cu *USART_SendData()* se realizează efectiv trimiterea fiecărui caracter din vector.

```
void sendChar(char a)
{
   while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
    USART_SendData(USART2, a);
}
```

Funcția de mai sus, trimite doar un singur caracter. Este asemănătoare cu *sendString()*, doar că nu mai necesită bucla *for* pentru parcurgerea vectorului.

```
char readChar()
{
   char chr;
   while(USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == RESET);
   {
      chr = USART_ReceiveData(USART2);
```

```
    }
  return chr;
}
```

Funcția *readChar()* așteaptă primirea unui caracter pe serială, și îl returnează.

Proiect X şi 0

Exemplul următor îşi propune să implementeze jocul X și 0 pe placa de dezvoltare Nucleo F103. Cei doi jucători vor fi reprezentați de utilizator și de placa Nucleo. Programul va implementa următoarele funcții:

- **Afișare mesaj întâmpinare** cu regulile jocului: pozițiile de pe tabla de joc vor fi notate de la 1 la 9, utilizatorul va alege primul mutarea și va juca cu X.
- **Preluare alegere utilizator** – utilizatorul va indica o poziție de la 1 la 9 cu alegerea dorită. Se va verifica în mod automat dacă alegerea este validă.
- **Alegere Nucleo** – programul va genera aleatoriu o alegere validă.
- **Verificare victorie sau remiză** – programul va verifica după fiecare alegere condițiile de victorie sau de remiză. Dacă este îndeplinită una dintre condiții, se va afișa un mesaj corespunzător și jocul va fi reinițializat.
- **Afișare tablă de joc** – după fiecare alegere completă (X și 0) se va afișa tabla de joc cu alegerile deja efectuate. Dacă după mutarea utilizatorului se va constata îndeplinirea condiției de victorie, tabla de joc va fi afișată imediat.

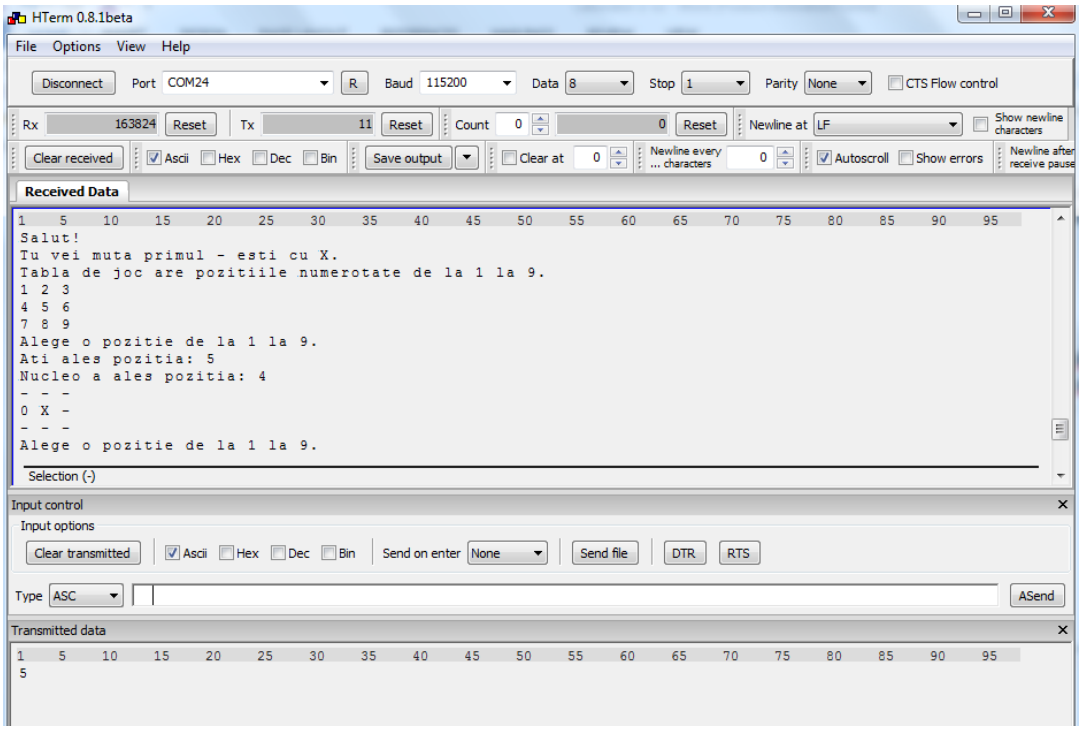


Figura 7. Fereastra de monitorizare a comunicaţiei seriale

Pentru a trimite alegerea făcută către plăcuță se va trece un număr de la 1 la 9 și se apăsa Enter. În mod automat, programul va genera o alegere aleatorie și va afișa tabla de joc cu alegerile făcute, invitând utilizatorul să facă următoarea mutare. Dacă utilizatorul indică o poziție invalidă, va fi obligat să aleagă din nou.

În momentul în care s-a detectat îndeplinirea condiției de victorie (sau remiză), jocul se oprește, programul indică câștigătorul și, după o pauză de 5 secunde, jocul se reia.

Programul va conține în partea de declarații inițializarea vectorului care va memora starea tablei de joc.

```
char tabla[]={'-','-','-','-','-','-','-','-','-'};
```

Funcția principală a programului va efectua următoarele sarcini:

- Afișare mesaj întâmpinare,
- Apelare funcție preluare mutare utilizator,
- Verificare condiție victorie sau remiză.

- Reluare ciclu până la apariția condiției de victorie sau remiză.
- După apariția condiției de victorie sau remiză, introducere întârziere 5 secunde, ”ștergere” ecran și reinițializare vector tablă de joc.

```
int main()
{
		uint8_t i;
		initGPIO();
		initTimer();
		initUsart();
		while(1)
		{
				sendString("Salut!\n");
				sendString("Tu vei muta primul - esti cu X.\n");
				sendString("Tabla de joc are pozitiile numerotate de la 1 la 9.\n");
				sendString("1 2 3\n");
				sendString("4 5 6\n");
				sendString("7 8 9\n");

				do
				{
					mutareX();
					if (victorie()) break;
					mutare0();
					afisare();
				}while(!victorie());

				Delay(5000);

				for (i=0; i<100; i++)
						sendString("\n");
				for (i=0; i<9; i++)
					tabla[i]='-';
		}
}
```

Funcția de victorie are rolul de a verifica după fiecare mutare (automată sau a utilizatorului) condițiile de victorie sau remiză. Dacă este validată una din condițiile de victorie sau remiză, va afișat un mesaj corespunzător și funcția va întoarce valoarea true.

```
_Bool victorie()
{
 char winner = '-';
	uint8_t i;
```

```
 if (tabla[0]==tabla[1] && tabla[1]==tabla[2]) winner=tabla[0]; // linie 1
 if (tabla[3]==tabla[4] && tabla[4]==tabla[5]) winner=tabla[3]; // linie 2
 if (tabla[6]==tabla[7] && tabla[7]==tabla[8]) winner=tabla[6]; // linie 3
 if (tabla[0]==tabla[3] && tabla[3]==tabla[6]) winner=tabla[0]; // coloana 1
 if (tabla[1]==tabla[4] && tabla[4]==tabla[7]) winner=tabla[1]; // coloana 2
 if (tabla[2]==tabla[5] && tabla[5]==tabla[8]) winner=tabla[2]; // coloana 3
 if (tabla[0]==tabla[4] && tabla[4]==tabla[8]) winner=tabla[0]; // diagonala principala
 if (tabla[2]==tabla[4] && tabla[4]==tabla[6]) winner=tabla[2]; // diagonala secundara

 if (winner =='-')
          {
  winner = '*';
  for (i=0; i<9; i++)
    if (tabla[i]=='-')
          winner='-';
 }

 if (winner=='-')
          return 0;

 else
 {
   if (winner == 'X')
   {
     afisare();
     sendString("Ati castigat! Felicitari!\n");
   }
   if (winner == '0')
          sendString("Nucleo a castigat! Mai incercati!\n");

   if (winner == '*')
          sendString("Remiza! Plictisitor!\n");

   return 1;
  }
}
```

Funcția de afișare este apelată din funcția principală și din funcția victorie() în momentul în care s-a validat câștigarea utilizatorului. Funcția face afișarea vectorului cu tabla de joc.

```
void afisare()
{
          uint8_t i;

   for (i=0; i<9; i++)
   {
     sendChar(tabla[i]);
     sendString(" ");
                    if (i==2 || i==5 || i==8)
                              sendString("\n");
   }
}
```

Cele două funcții de alegere a unei noi poziții (mutareX – utilizator, mutare0 – Nucleo) preiau sau generează alegerea, o validează și afișează această alegere.

```
void mutare0()
{
 int i=-1;
 while (i==-1)
          {
  i = rand()%9;
  if (tabla[i]=='-')
                    {
     tabla[i]='0';
     sendString("Nucleo a ales pozitia: ");
     sendChar(i+1+'0');
     sendString("\n");
     break;
  }
  else i=-1;
 }
}
void mutareX()
{
  _Bool valid = 0;
  uint8_t m0 = 0;
  while (!valid)
  {
   sendString("Alege o pozitie de la 1 la 9.\n");
   while (m0<'1' || m0>'9')
  {
    m0 = readChar();
    Delay(1000);
   }

   sendString("Ati ales pozitia: ");
   sendChar(m0);
    m0 = m0 - 48;
   sendString("\n");

   if (tabla[m0-1]!='-')
          sendString("Pozitie eronata!\n");
          else
          {
              tabla[m0-1]='X';
              valid = 1;
            }
  }
}
```