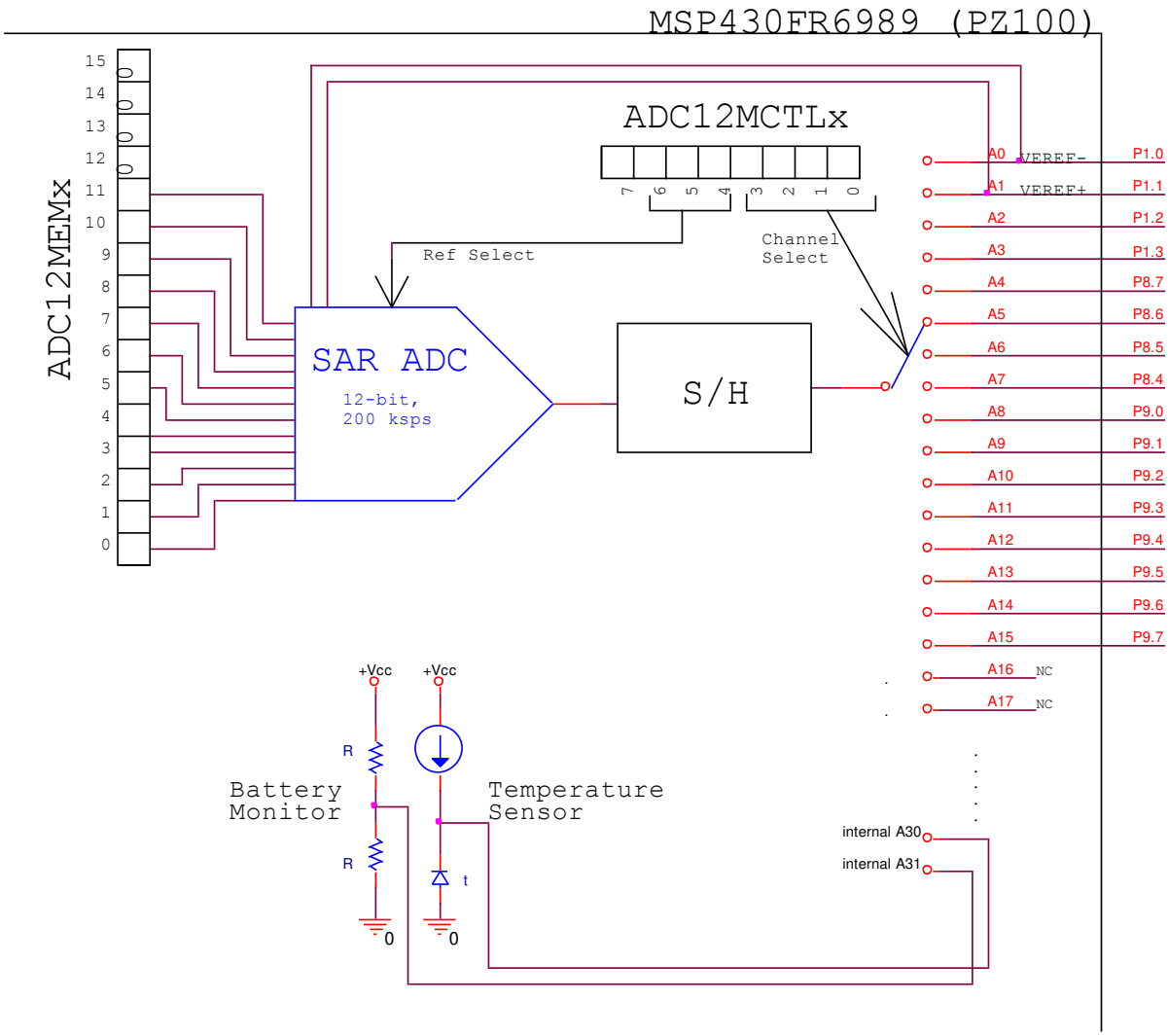
Лабораторно упражнение №7
“Аналогово-цифров преобразувател на MSP430FR6989 и комуникационен модул UART (RS-232)”

7.1. Въведение. Микроконтролерът MSP430FR6989 притежава вграден аналогово-цифров SAR преобразувател с 12-битова разредност, бързодействие 200 ksps и 16 входни канала (**фиг. 7.1**).



Фиг. 7.1

Потребителят може да измерва напрежение с до 16 извода на микроконтролера. Други 2 канала са вътрешни и са за измерване температурата на чипа (с помощта на интегриран диод), за измерване на захранващото напрежение (с помощта на интегриран резисторен делител) и останалите не се използват. При включване на външни еталонни източници за горен и долен праг на измерването потребителят ще може да измерва напрежение само на 14 извода.

Преди даден извод да бъде използван от АЦП-то, трябва да се конфигурира *мултиплексорът*, превключващ извода измежду различните периферни модули (всеки извод може да бъде използван от много периферни модули, но в даден

момент само един е включен на него). Това става чрез регистър PxSEL0 и PxSEL1.

АЦП може да работи в *4 режима*, задавани от битовете ADC12CONSEQx на регистъра ADC12CTL1:

- един канал, едно измерване;
- много канали, по едно измерване на всеки;
- един канал, много измервания;
- много канали, много измервания на всеки.

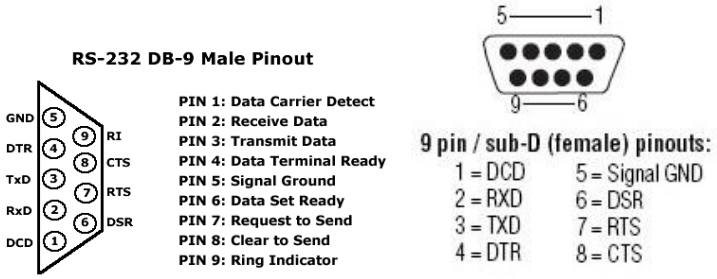
За всеки един канал има по един MCTL регистър, който указва източниците на еталонно напрежение за конкретния канал, както и дали той е първи или последен при многоканално измерване.

АЦП-то може да използва *еталонния източник* на напрежение, вграден в микроконтролера. Неговата стойност може да се избере от напреженията: 1.2 V, 2.0V, 2.5 V. Това може да стане, като се установи бит 0 от REFCTL0 в единица и с REFVSEL битовете се избере една от трите стойности. Да се разгледа MSP430FR6989 User's Guide-а за желаната комбинация на REFVSEL.

Тактовата честота на модула е един от източниците SMCLK, MCLK, ACLK, MODCLK и може да се избере чрез битовете ADC12SSELx от регистъра ADC12CTL1.

Тази честота допълнително може да бъде *разделена* 1 ÷ 512 пъти чрез ADC12DIVx битове от регистъра ADC12CTL1 и ADC12PDIVx битове от регистъра ADC12CTL2.

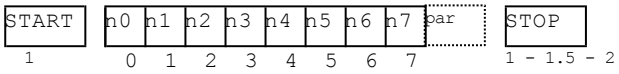
В настоящото упражнение се използва RS232 интерфейс, наречен още UART, за осъществяване на връзка между микроконтролера и персонален компютър. Това е напрежителен, сериен, асинхронен интерфейс. На **фиг. 4.2** са показани най-често използваните куплунги с 9 извода (има и с 25 извода) - мъжки и женски DB9.



Фиг. 7.2

При повечето приложения микроконтролерите използват само три проводника от този куплунг – това са TxD (извод 3) за предаване, RxD (извод 2) за приемане и маса (извод 5). Логическите нива се представят с отрицателни и положителни напрежения за разлика от TTL нивата, където са само положителни. Логическата 0 (наричана Space) е дефинирана като ≥ +5 V, а логическата 1 (наричана Mark) като ≤ -5 V. Максималните нива на напреженията трябва да са в границите ± 15 V, но чиповете, използващи този интерфейс трябва да са способни да издържат до ±25 V. Приемниците трябва да регистрират и затихнали сигнали с амплитуда не по-малка от +3V и не по-голяма от -3 V (т.е. имат минимален 5 – 3 = 2 V запас за шумоустойчивост).

Предаването на данни се осъществява байт по байт, всеки от който съдържа един стартов бит, даннови битове, бит проверка за грешки и един или повече стопови бита (**фиг. 7.3**). Изпращането на данните се извършва от LSB към MSB. Повечето инженери са свикнали да представят байтовете в MSB към LSB формат, затова може да се каже, че при наблюдение на осцилоскопа данните от UART изглеждат обърнати. Когато не се предават данни линиите се държат в логическа 1.



Фиг. 7.3

Битът „*проверка за грешки*" може да бъде 4 вида:

- проверка по четност – битът се установява в 1 или 0, така че сумата от всички единици в изпращания байт да е четно число;
- проверка по нечетност – битът се установява в 1 или 0, така че сумата от всички единици в изпращания байт да е нечетно число;
- проверка с единица (mark parity) – битът е винаги единица;
- проверка с нула (space parity) – битът е винаги нула.

Пример: ако се изпраща числото 00010010 с проверка по четност, деветият бит ще е нула. Аналогично при проверката по нечетност числото 0011 0011 ще има единица за деветия бит.

Стоповите битове може да са :

- един;
- един и половина;
- два.

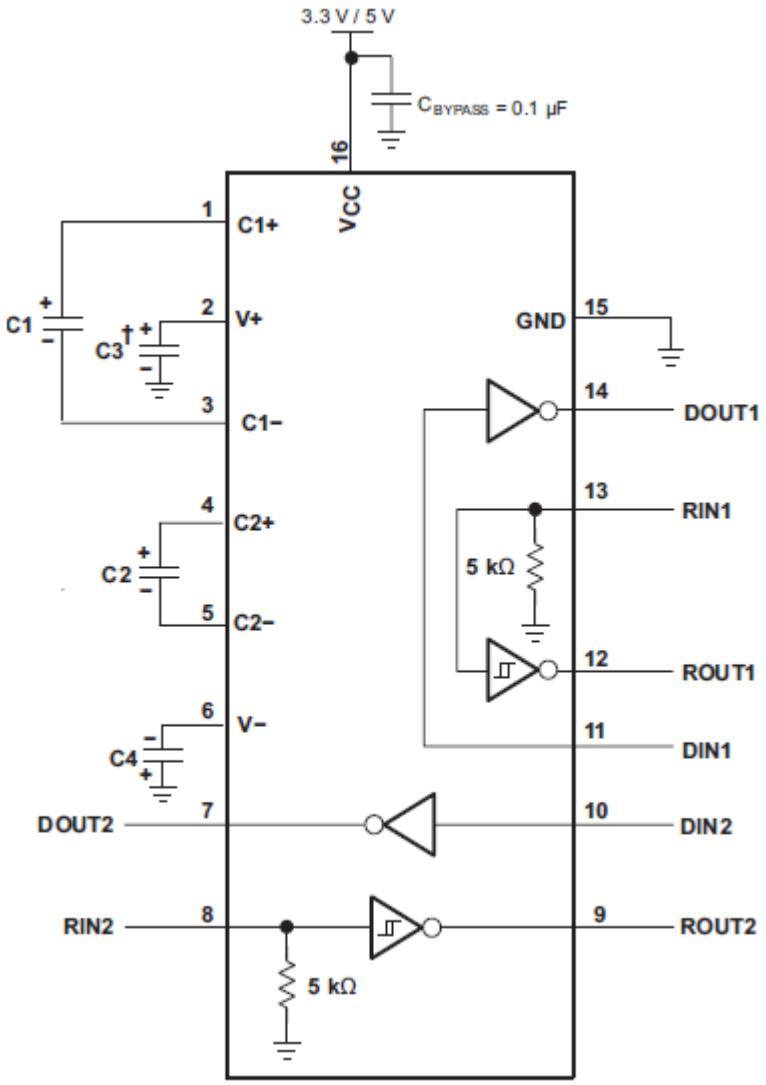
Един и половина и два стоп бита се използват при по-бавни приемащи устройства, които имат нужда от малко по-дълъг (във времето) стоп бит, за да обработят приетите данни.

Данновите битове може да са 5 ÷ 8. Изпращащото устройство предава данните на всеки свой нарастващ фронт на вътрешния му генератор, а приемащото устройство регистрира тези данни на всеки падащ фронт на вътрешния му генератор. Синхронизацията на вътрешните генератори на приемника и предавателя се осъществява по спадащия фронт на старт бита (т.е. при първия преход -15 V→ +15 V). Така интерфейсът не се нуждае от допълнителен проводник за тактов сигнал и затова наричаме UART-а асинхронен интерфейс – разчита се на собствените генератори, вградени в предавателя и в приемника.

Скоростта на предаване се дава в бодове и 1 бод = 1 бит информация. В практиката се използват стандартизирани скорости, някои от които – 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200. По-високи скорости на обмен се осъществяват с USB-UART емулатори. Максималната дължина на кабела между две устройства е 10 m.

Понеже микроконтролерите се захранват с напрежения до 5 V, то те не могат да изработят със сигнали с амплитуда ± 5V ÷ ±15 V. Също така не биха могли да приемат такива сигнали без да се повредят. Затова се използват специални транслаторни схеми, които изработват необходимите сигнали и за интерфейса, и за микроконтролера. Една примерна такава ИС е показана на **фиг.**

7.4. Използван е MAX3232, а на схемата е дадена и вътрешната му структура. Инверторите са всъщност специални и осигуряват транслацията на нивата. Използват се charge-pump постояннотокови преобразуватели за повишаване на положителните и изработване на отрицателните напрежения. Тези преобразуватели използват кондензаторите C1, C2, C3 и C4.



Фиг. 7.4

7.2 Задачи за изпълнение.

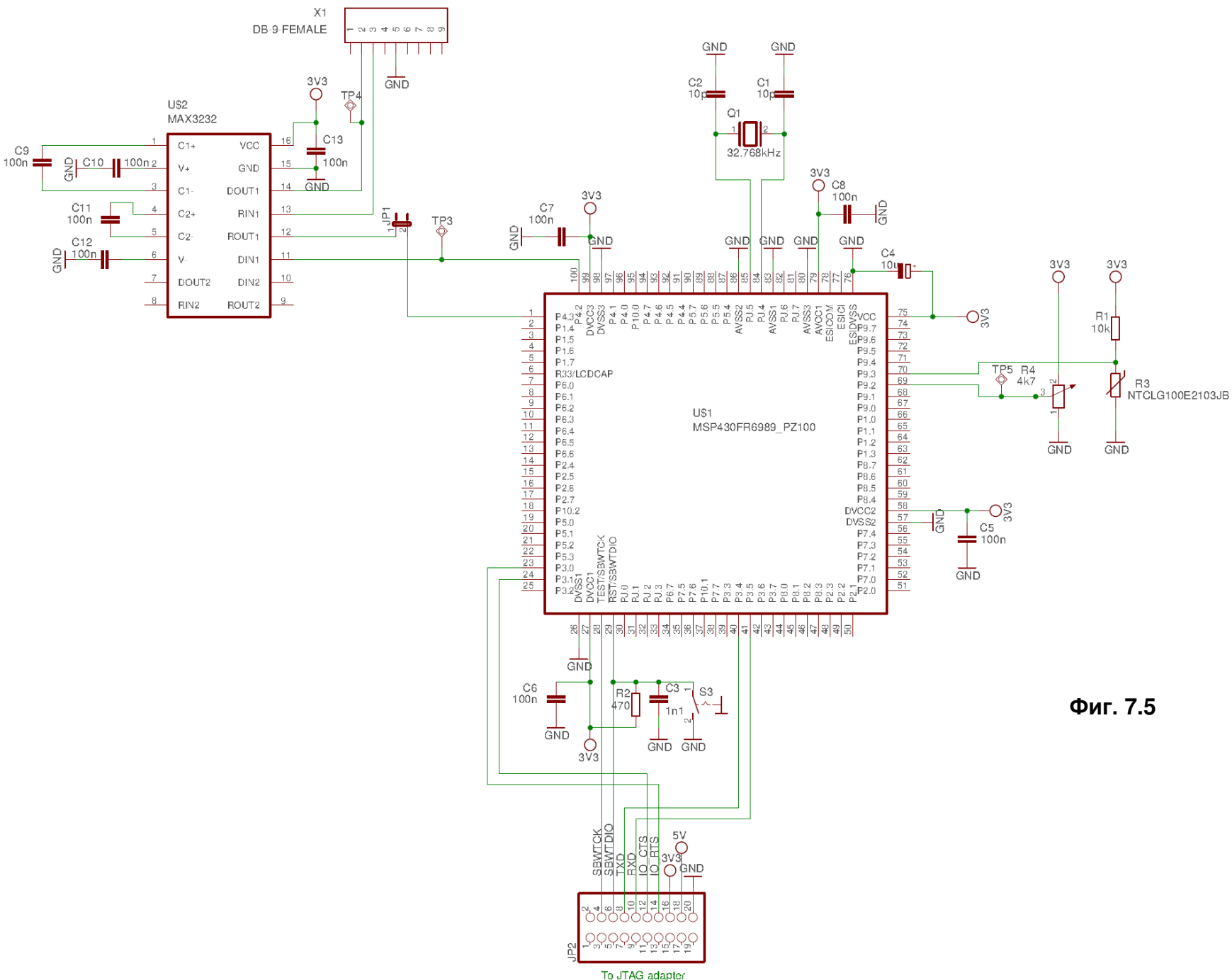
7.2.1. Да се създаде нов C проект на Desktop-a. Да се напише програма, която изпраща „Hello, World!" съобщение по UART (RS232) интерфейса.

За целта трябва да се копират файловете uart.c, uart.h, uartstdio.c и uartstdio.h (предоставени от ръководителя на упражнението) в директорията на проекта. След това в IAR →десен бутон върху top-level директорията → Add Files →указва се пътя до току-що добавените файлове → Open. В main.c се включва хедърния файл uartstdio.h. Използвайки API функциите от този файл реализирайте програмата си. За да видите резултата стартирайте терминална програма (Hyper Terminal, Tera Term, Real Term и т.н.) и настройте връзка **9600 бода**, **8 даннови бита**, **без проверка по четност**, **1 стопов бит**. Честотата на микропроцесора се настройва от библиотеката на 16 MHz.

»RS232 може да бъде виртуален и емулиран от USB дебъгера на демо платката. В <uartstdio.h> библиотеката се използва реалният (физическият) интерфейс.

7.2.2. Да се създаде нов C проект на Desktop-a в отделна директория. Да се свърже потенциометър към демо платката (**фиг.7.5**). Да се напише програма, която измерва напрежението в средната точка на потенциометъра. Да се наблюдава резултата от измерването посредством JTAG дебъгера. Ако за променливата ADCResult пише, че е <unavailable>, изключете оптимизациите на компилатора от Project →Options → Category: C/C++ Compiler → tab Optimizations → Level: сложете радио бутона на None. Изберете Rebuild на проекта.

7.3.3. Да се създаде нов C проект на Desktop-a в отделна директория и да се добави UARTStdio, както беше направено в задача 7.2.1. Да се напише програма, която измерва напреженията от потенциометъра, от термистора и от вграденият резисторен делител на MSP430R6989 (който е свързан към захранване, маса и има коефициент на предаване 0,5). Данните да се изпращат от микроконтролера към компютъра по RS232. Нека данните от измерванията на трите канала се представят в необработен вид като цели числа в интервала 0 ÷ 4095.


X1
DB-9 FEMALE
US2
MAX3232
C1+
V+
C1-
C2+
C2-
V-
DOUT2
RIN2
VCC
GND
DOUT1
RIN1
ROUT1
DIN1
DIN2
ROUT2
C9
100n
C10
100n
C11
100n
C12
100n
C13
100n
3V3
GND
TP4
JP1
TP3
C7
100n
C2
10p
C1
10p
Q1
32.768kHz
C8
100n
C4
10u
US1
MSP430FR6989_PZ100
C5
100n
C6
100n
R2
470
C3
1n1
S3
TP5
R4
4k7
R1
10k
R3
NTCLG100E2103JB
SBWTCK
SBWTDIO
TXD
RXD
IO_CTS
IO_RTS
5V
JP2
To JTAG adapter

Фиг. 7.5

Задача 7.2.1

```
#include "io430.h"
??? //Включи хедърния файл на UARTStdio, който сте добавили към проекта

void main( void )
{
  WDTCTL = WDTPW | WDTHOLD;
  PM5CTL0 &= ~LOCKLPM5;

  ???//Инициализирай UARTStdio функцията → отвори хедърния файл и виж прототипите на
       //достъпните функции

  while(1)
  {
   ??? //Извикай printf за MSP430 с произволен стринг → отвори хедърния файл и виж
         //прототипите на достъпните функции, извикайте printf( ) функцията, както бихте я
         //извикали в конзолна С програма на вашия PC

    __delay_cycles(16000000);
  }
}
```

Задача 7.2.2

```
#include <stdint.h>
#include "io430.h"

void initADC( )
{
   //Избери функцията на извод P9.2 като вход на АЦП (A10)
   P9SEL0 ???= ???;
   P9SEL1 ???= ???;

   ADC12CTL0 &= ~ADC12ENC;     //Забрани преобразуването, ENC = 0
   ADC12CTL0 |= ADC12ON;       //Включи АЦП
   ADC12CTL0 |= ADC12SHT1_15; //Избери Sample-and-hold време = 512 такта на АЦП-то за
                               //канали 8 - 23
   ADC12CTL1 |= ADC12SSEL_2;   //Избери MCLK за тактов сигнал
   ADC12CTL1 |= ADC12SHP;      //Използвай sampling таймера
   ADC12CTL1 |= ADC12CONSEQ_0;  //Избери режим "един канал - едно измерване"
   ADC12CTL1 |= ADC12PDIV_0;  //Раздели тактовия сигнал на 1
   ADC12CTL1 |= ADC12DIV_1;      //Раздели тактовия сигнал на 2 (от втория делител)
   ADC12CTL1 &= ~ADC12ISSH;     //Не инвертирай sample-and-hold сигнала
   ADC12CTL1 |= ADC12SHS_0;     //Използвай ADC12SC бита като сигнал за начало на sample-
                               //and-hold
   ADC12CTL2 &= ~ADC12DF;       //Формат на резултата - прав код
   ADC12CTL2 &= ~ADC12RES_3;     //Занули битовете за разредността
   ADC12CTL2 |= ???;                 //Избери 12-битово преобразуване, Userguide стр. 880
//-------------------------MCTL конфигурация-------------------------------------------------------------------
   ADC12MCTL10 &= ~ADC12DIF;       //Избери single-ended преобразуване
   ADC12MCTL10 |= ADC12VRSEL_0; //Използвай Vdd за горен праг, Vss - за долен.
   ADC12MCTL10 |= ADC12INCH_10;  //Избери A10 за входен канал
   ADC12CTL3 |= ADC12CSTARTADD_10; //Избери регистър ADC12MEM10 да е приемник на
                                   //резултата от преобразуването
   ADC12MCTL10 |= ADC12EOS;              //Канал A10 е крайният канал за преобразуване
//-------------------------------------------------------------------------------------------------------------

   ADC12CTL0 |= ADC12ENC;    //Установи ENC в 1
}
```

```
void main( void )
{
  //Тук ще се съхранява резултата от измерването
  volatile uint16_t ADCResult;

  WDTCTL = WDTPW + WDTHOLD;
   PM5CTL0 &= ~LOCKLPM5;

  //Инициализирай АЦП-то
  initADC();

  while(1){
    ADC12CTL0 |= ???;     // Започни измерване, Userguide →стр. 877

    while(ADC12CTL1 & ???){ }  // Изчакай, докато приключи измерването, Userguide →стр. 878
    ADCResult =  ADC12MEM10; //Запиши резултата в  променливата ADCResult, сложете тук точка
                                  //на прекъсване
   }
}
```

Задача 7.2.3

```
#include <stdint.h>
#include "io430.h"
#include "uartstdio.h"

void initADC( )
{
    //Избери функцията на извод P9.2 като вход на АЦП (A10)
    P9SEL0 ???= ???;
    P9SEL1 ???= ???;

    //Избери функцията на извод P9.3 като вход на АЦП (A11)
    P9SEL0 ???= ???;
    P9SEL1 ???= ???;

    ADC12CTL0 &= ~ADC12ENC;     //Забрани преобразуването, ENC = 0
    ADC12CTL0 |= ADC12ON;       //Включи АЦП
    ADC12CTL0 |= ADC12SHT1_15; //Избери Sample-and-hold време = 512 такта на АЦП-то за
                          //канали 8 - 23
    ADC12CTL0 |= ADC12MSC;      //Автоматично ресетиране на SH таймера след всяко
                          //преобразуване
    ADC12CTL1 |= ADC12SSEL_2;   //Избери MCLK за тактов сигнал
    ADC12CTL1 |= ADC12SHP;      //Използвай sampling таймера
    ADC12CTL1 |= ADC12CONSEQ_1;  //Избери режим "много канали, по едно измерване на всеки"
    ADC12CTL1 |= ADC12PDIV_0;  //Раздели тактовия сигнал на 1
    ADC12CTL1 |= ADC12DIV_1;       //Раздели тактовия сигнал на 2 (от втория делител)
    ADC12CTL1 &= ~ADC12ISSH;     //Не инвертирай sample-and-hold сигнала
    ADC12CTL1 |= ADC12SHS_0;     //Използвай ADC12SC бита като сигнал за начало на sample-
                                 //and-hold
    ADC12CTL2 &= ~ADC12DF;       //Формат на резултата - прав код
    ADC12CTL2 &= ~ADC12RES_3;     //Занули битовете за разредността
    ADC12CTL2 |= ???;      //Избери 12-битово преобразуване, Userguide стр. 880
//-------------------------MCTL конфигурация-------------------------------------------------------------

    ADC12CTL3 |= ADC12CSTARTADD_10; //Избери регистър ADC12MEM10 да е приемник на
                                          //резултата от първото преобразуване

    //Конфигурирай измерването на потенциометъра
    ADC12MCTL10 &= ~ADC12DIF;       //Избери single-ended преобразуване
```

```
    ADC12MCTL10 |= ADC12VRSEL_0; //Използвай Vdd за горен праг, Vss - за долен.
    ADC12MCTL10 |= ADC12INCH_10;  //Избери A10 за входен канал
    ADC12MCTL10 &= ~ADC12EOS;        //Канал A10 не е крайният канал за преобразуване

    //Конфигурирай измерването на NTC резистора (термистора)
    while(REFCTL0 & REFGENBUSY);  //Провери дали REFGENBUSY е 1, ако ДА - изчакай
    REFCTL0 |=( ??? | REFON);              //Задай 2.5 V | Включи вътрешния еталон
    ADC12MCTL11 &= ~ADC12DIF;     //Избери single-ended преобразуване
    ADC12MCTL11 |= ???;  //Използвай Vref(буфериран) за горен праг, Vss - за долен.
    ADC12MCTL11 |= ADC12INCH_11;  //Избери A30 за входен канал
    ADC12MCTL11 &= ~ADC12EOS;     //Канал A30 не е крайният канал за преобразуване

    //Конфигурирай измерването на вътрешния резисторен делител (A31)
    ADC12CTL3 |= ADC12BATMAP;      //Избери вътрешния източник за канал A31
    ADC12MCTL31 &= ~ADC12DIF;       //Избери single-ended преобразуване
    ADC12MCTL31 |= ADC12VRSEL_0;    //Използвай Vdd за горен праг, Vss - за долен.
    ADC12MCTL31 |= ADC12INCH_31;    //Избери A31 за входен канал
    ADC12MCTL31 |= ADC12EOS;        //Канал A31 е крайният канал за преобразуване
//-------------------------------------------------------------------------------------------------------------------------------

    ADC12CTL0 |= ADC12ENC;    //Установи ENC в 1
}

void main( void )
{
  unsigned long adc_samples[3]; //Тук ще се съхраняват резултатите от измерванията

  WDTCTL = WDTPW | WDTHOLD;
  PM5CTL0 &= ~LOCKLPM5;

  initADC(); //Инициализирай АЦП-то
  UARTStdioInit(); //Инициализирай UART модула и UARTprintf функцията

  while(1){
     ADC12CTL0 |= ???; // Започни измерване, Userguide стр. 877
     while(ADC12CTL1 & ???){ }

     adc_samples[0] =  ADC12MEM10;
     adc_samples[1] =  ADC12MEM11;
     adc_samples[2] =  ADC12MEM31;

     UARTprintf("Pot: %04d Tempr: %04d Vdiv: %04d \r", adc_samples[0], adc_samples[1],
adc_samples[2]);

    __delay_cycles(1600000);
  }
}
```

Лабораторно упражнение №8

"Безжичен LAN (WLAN) модул CC3100 и MSP430FR6989. Създаване на Web сървър."

8.1. Въведение. Към демо платката MSP-EXP430FR6989 може да се свърже комуникационна демо платка SimpleLink Wi-Fi CC3100BOOST (произведена от фирмата Texas Instruments), позволяваща реализиране на безжична LAN мрежа (или още Wireless LAN, WLAN, но да не се бърка с WAN) по стандарта IEEE 802.11 b/g.Такъв вид мрежа се поддържа от безжичните домашни рутери, което позволява да се създаде вградена система на базата на MSP430FR6989 с връзка към Интернет (**фиг. 8.1**). Ядрото на SimpleLink платката е интегралната схема CC3100 на фирмата Texas Instruments. Към него трябва да се свърже единствено външна антена и няколко кондензатора. Този вид чипове често се наричат мрежови процесори и служат за добавяне на Интернет свързаност към съществуващи вградени системи със собствен микроконтролер. От друга страна чипът CC3200 включва освен мрежови процесор, също и микроконтролер с ARM Cortex-M4 микропроцесор и периферия, т.е. може да се разглежда като по-сложен микроконтролер. Връзката MSP430FR6989←→CC3100 се осъществява чрез SPI интерфейс. SimpleLink използва SMD антена (отбелязана е като E1), която реализира безжичния интерфейс CC3100←→Wifi рутер. Максималната скорост на Интернет връзката е 16 Mbps (на носеща честота 2.4 GHz), което е много над възможностите на MSP430FR6989 (макс. тактова честота 16 MHz) и реално обменът на данни става на по-ниска скорост.

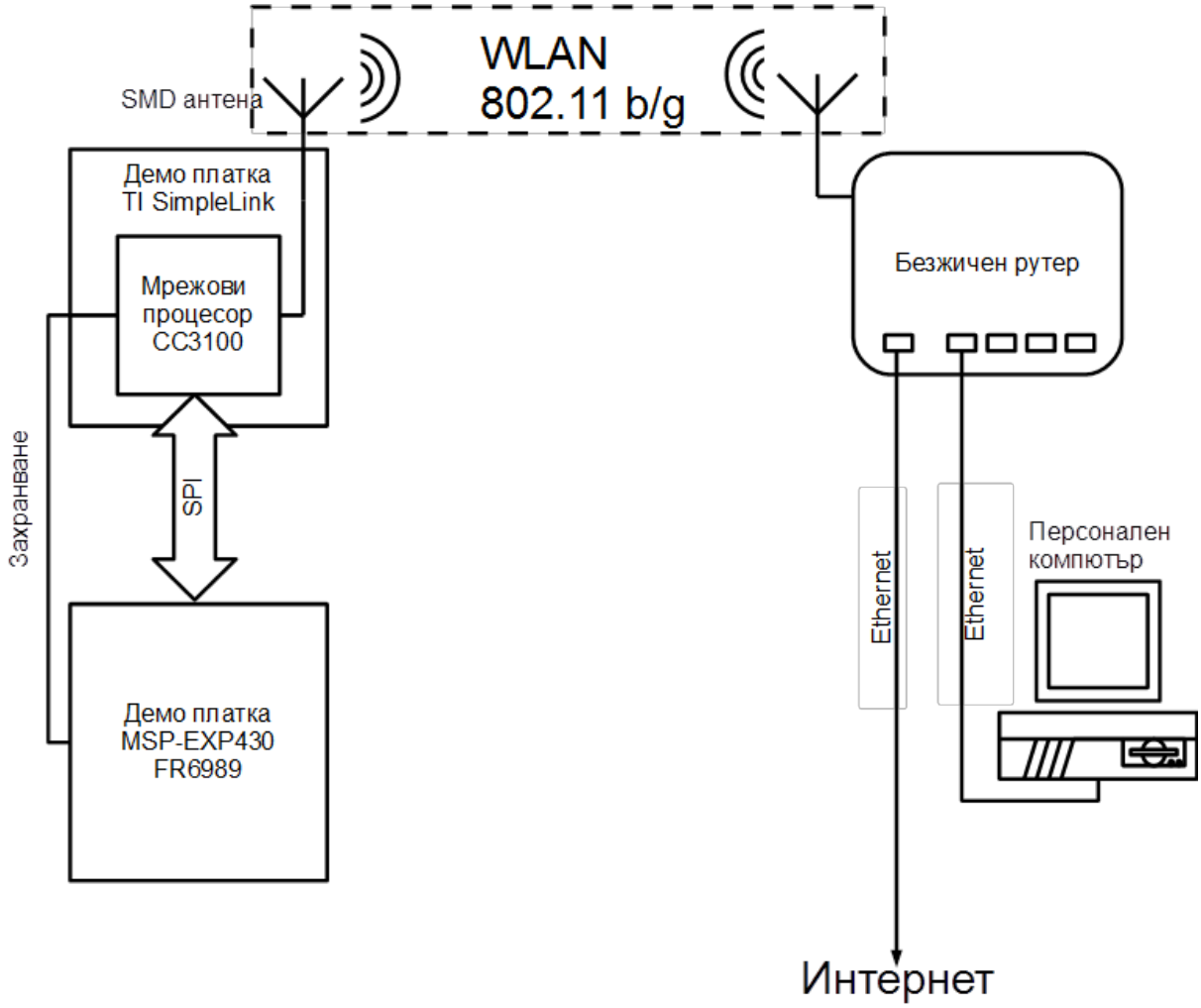
8.2. Основни понятия за Интернет. Някои от по-важните термини за Интернет комуникацията са:

- **LAN** (Local Area Network) - мрежа, обхващаща различен брой компютри, разположени в област не по-голяма от 10 km. Броят на компютрите варира много и зависи от мрежовата структура и използваните кабели. LAN може да се нарече мрежа от три компютъра в три съседни блока, както и 1000 компютъра в един квартал.

- **WLAN** (Wireless Local Area Network) - класическа LAN мрежа, но използваща ефира за преносна среда. Комуникацията се осъществява в определен радиоканал.

- **WAN** (Wide Area Network) – международни мрежи, обхващащи голяма географска област. Наи-известният пример е Интернет. Друг пример са корпоративните мрежи (Интранет) на фирми, имащи клонове по цял свят и които са си изградили WAN за спомагане дейността на фирмата. Такива мрежи осигуряват обмен от около 1 ÷ 6Mbit/s, което е значително по-малко от скоростите, използвани в LAN.

- **Gateway** (на български се използва термина „шлюз“) - активни мрежови устройства, наречени още рутери (router), които представляват компютри или специализирани вградени системи, конфигурирани да управляват маршрута на данните. За да се осъществи връзка от една LAN в друга LAN трябва да се премине през gateway.

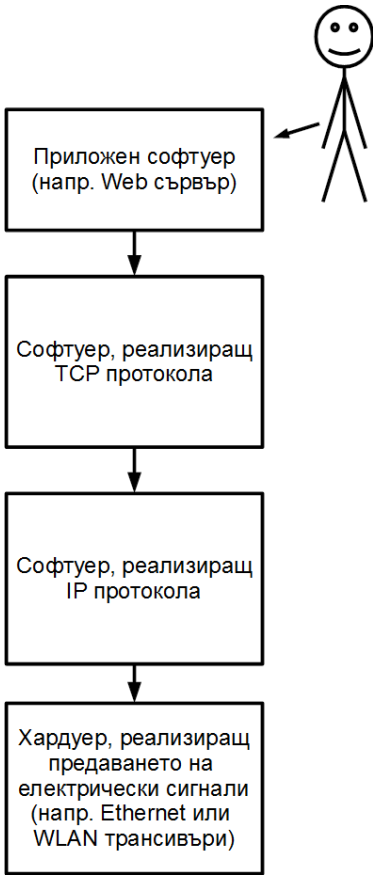


Фиг. 8.1

- **IP адрес** – уникално 32-битово цяло число, присвоено на дадено устройство в Интернет. Такъв адрес е необходим при предаването на информация между две и повече устройства. Изобразява се в десетичен вид, като всеки байт е разделен с точки (например 192.168.1.10, 10.1.1.23, 81.150.230.33 и други). Тези адреси използват версия на IP протокола номер 4 (IPv4).

С 32-битово число може да се адресират до 2^{32} = 4 294 967 296 устройства. Поради широката употреба на световната мрежа Интернет този брой се оказва недостатъчен, затова съществуват и 128-битови адреси (IPv6).

- **IP протокол** (Internet Protocol) – стандартизиран механизъм за предаване на информация в световната мрежа Интернет. Той дефинира комуникация, която се осъществява чрез разделяне на данните на пакети в подателя и събирането им в получателя.Всеки пакет освен полезните данни съдържа и IP адреса на подателя/получателя, заедно с различни служебни данни. При този протокол доставката на информация не се гарантира, т.е. някои от пакетите могат да бъдат изгубени.
- **DHCP протокол** (Dynamic Host Configuration Protocol) – аналогичен на IP протокола, но разликата е, че при първоначалното включване на устройството параметрите на IP протокола (IP адрес, Mask, Gateway и др.) се получават автоматично от DHCP сървър. Това означава, че потребителят ще получи автоматично, без негова намеса, всички необходими настройки, за да се свърже към Интернет. Характерното за този протокол е, че при всяко стартиране на компютъра си, потребителят получава IP адрес, който може да не съвпада с IP адреса от предишна сесия (предишното включване) на компютъра. Нещо повече – този IP адрес може да се промени по време на настоящата сесия.
- **TCP протокол** (Transmission Control Protocol) – протоколът осигурява пренасянето на непрекъсната поредица от данни, като за разлика от повечето Интернет протоколи, работи с изграждане на връзка и гарантиране на доставката на данните. Софтуерът, реализиращ този протокол е на най-високото ниво на абстракция в TCP/IP комуникацията (**фиг. 8.2**). Гарантиране предаването на данните означава, че ако някой пакет се изгуби от IP протокола, TCP слоя ще детектира това и ще го изпрати наново.



Фиг. 8.2

8.3. Връзка клиент-сървър. Връзката между две устройства в Интернет се основава на принципа клиент-сървър, при която:

- **клиент** се нарича компютърна програма, изискваща създаването на връзка и използване на някакъв компютърен ресурс;

-**сървърно приложение**е компютърна програма, която предоставя даден ресурс на една или повече клиентски програми. Сървърното приложение „слуша“ за идващи заявки от клиенти, обслужва ги и продължава да „слуша“ за следващи клиенти.

Типичен пример за комуникация клиент-сървър е качването на файл в Google Drive. Тук Интернет браузърът (Mozilla, Opera, Google Chrome) се нарича клиент, а Google Drive – сървърно приложение. Предоставеният ресурс е дисково пространство.

За да може клиент да се свърже към сървър е необходимо да се знаят поне два параметъра на сървъра:

- IP адрес;
- Порт, на който сървърното приложение очаква заявки от клиентите.

Под порт се разбира софтуерен механизъм за връзка, а не физически порт (куплунг). Софтуерните портове са необходими, за да се реализира достъп до сървъра от много потребители към много сървърни програми, а физически конекторът за връзка може да е само един. Благодарение на

портовете един и същ сървър може да се използва например за уеб-страница и дисково пространство за документи едновременно. По подразбиране уеб-страниците се обслужват на порт 80, т.е. когато напишем в браузъра www.mysite.com, всъщност ще се преведе като www.mysite.com:80. С двуеточие се указва номера на порта. Друг пример е клиентската програма, осигуряваща достъп до файловата система на сървър – SSH. Връзката на SSH се осъществява винаги на порт 22. Валидни числа за портове са 0 ÷ 65535.

От гледна точка на програмиста, връзката клиент-сървър се осъществява посредством т.нар. сокети (socket), които наподобяват файлове. Сокетът е крайна точка за комуникация (endpoint) в една компютърна мрежа. За да започне комуникацията, потребителската програма трябва да отвори сокет, след което да извърши всички операции по комуникацията с помощта на указател към този сокет (точно както се използват файлови указатели) и накрая да затвори сокета. Този вид достъп наподобява файловия достъп (отваряне, четене/запис, затваряне).

В настоящото упражнение се използва библиотека на Texas Instruments, която предлага множество API функции за работа със сокети. Най-често използваните от тях са показани в **Таблица 8.1**.

Име на функцията	**Кратко описание**
sl_Socket( )	Създава крайна точка за комуникация, сокет, и връща указател към нея.
sl_Bind( )	Присвоява на сокета статичния IP адрес на мрежовата карта. Ако тя използва DHCP, като адрес на функцията трябва да се подаде нула. Присвоява на сокета порт, на който ще се осъществяват връзките.
sl_Listen( )	Използва се от сървърни програми. Отваря се присвоеният порт на присвоения IP адрес. От този момент нататък системата очаква („слуша“) за пристигащи връзки от клиенти.
sl_Accept( )	Приема връзка с клиент. Тази функция връща указател към сокета на клиента (с негова помощ може да разберем например IP адреса на клиента). Блокираща функция.
sl_Connect( )	Използва се от клиентски програми. На тази функция се предава IP адреса на сървъра и неговия порт, към който ще се свързваме.

sl_Recv( )	Връща приетите байтове от сървъра/клиента. Блокираща функция.
sl_Send( )	Изпраща определен брой байтове към сървъра/клиента.

Таблица 8.1

8.4. Развойна среда Code Composer Studio. Фирмата Texas Instruments разработва своя развойна среда за микроконтролерите си, въпреки че много от тях могат да се програмират със софтуер с отворен код (от компилатори до програми за зареждане във флаш паметта). При инсталацията на средата потребителят може да избере комерсиалните компилатори на TI (със същото име) и/или отворени такива (GCC). След това при създаването на нов проект може да се избере един от двата.

CCS е базирана на отворената развойна среда Eclipse. Визуално двете среди си приличат много. Вляво на екрана е разположено дървото на проекта, централно - текстов редактор, долу – терминал на компилатора. При дебъгване на програма допълнително се отварят прозорци за дисасембли на програмата и организацията на стека. Преминаването от редактиращ в дебъгващ изглед (perspective) става с два бутона горе, вдясно на екрана.

8.5. Задачи за изпълнение.

8.5.1. Да се разучи създаването на TCP/IP връзка. За целта се стартират два прозореца на програмата Sock, намираща се в MSHT_LAB/tools. В първия прозорец се избира:

- Socket type → Server
- В полето Server Name → 127.0.0.1
- В полето Server Port → 80 (пример, може и друго число)
- Натиска се бутон Listen.

Във втория прозорец се избира:

- Socket type →Client
- В полето Server Name → 127.0.0.1
- В полето Server Port → 80 (пример, може и друго число)
- Натиска се бутон Connect
- В полето Message се въвежда произволно текстово съобщение
- Натиска се бутон Send.

Ако всичко е минало наред, в прозореца на сървъра трябва да се покаже същото съобщение.

Пояснение: с този експеримент тествахме TCP/IP комуникацията по „виртуалната LAN карта“ на персоналния компютър. Linux и Windows

операционните системи поддържат такъв виртуален интерфейс, чийто виртуален IP адрес е 127.0.0.1 и се нарича localhost.

Допълнителен експеримент: научете IP адреса на вашия компютър, като въведете в терминал (за Windows 7 → Startmenu → Search → cmd) командата ipconfig. Попитайте ваши колеги за тяхното IP и опитайте да изпратите текстово съобщение до техния компютър.

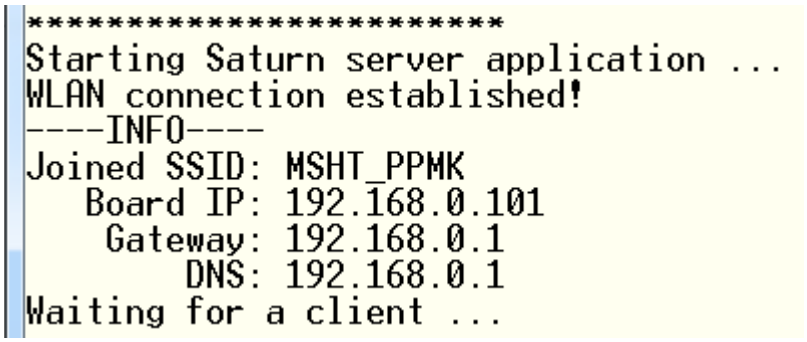
8.5.2. Да се тества работата на MSP430FR6989 и CC3000 с помощта на демо програмата CC3000_FRAM_Sensor_Application. За целта се стартира средата Code Composer Studio на Texas Instruments. От главното меню се избира File → Switch workspace → Other → Browse →указва се пътя до работното поле (workspace) CC3000_workspace → OK → OK. Вляво на средата, в дървото с директории, се намира директорията на проекта SensorApplication. Щраква се два пъти върху нея, за да се покажат C файловете. От тях се щраква два пъти върху main.c, където се намира main( ) функцията на проекта. Избира се Project → Build Project. Компилацията на проекта трябва да премине без грешки. След това се зарежда фърмуерът в микроконтролера чрез Run → Debug. В CCS ще се отвори Debug сесия, което е нормално. Изпълнението на програмата ще спре в началото на main( ).

Следва да се отвори терминална програма за RS232 интерфейс (Hyper Terminal, TeraTerm, Real Term, Conect или др.) с 9600 бода, 8-N-1 и COMx, където x е номера на виртуалния COM порт на компютъра, към който е свързана MSP430FR6989 платката.

Когато се направи това, обратно в CCS се натиска бутон Resume или клавиш F8 от клавиатурата. Сървърът използва безжична LAN връзка, затова преди да стартираме клиента трябва да свържем MSP430FR6989 към безжичен рутер. Ако връзката е успешно осъществена, червеният светодиод LED1 ще свети постоянно.

!Важно→Wi-Fi мрежата трябва да използва WPA2 (криптиран) протокол.

В рамките на няколко секунди трябва да излязат данните на връзката в терминала, **фиг. 8.3**.



Фиг. 8.3

Фърмуерът на MSP430FR6989 реализира сървър, към който ще се свързва клиентът (компютърната програма) CC3000_FRAM_Sensor_Application.

Клиента CC3000_FRAM_Sensor_Application се намира в MSHT_LAB/tools/CC3000_FRAM_Sensor_Application. За да се стартира трябва да се влезе в директорията със същото име и да се щракне два пъти върху batch скрипта:

run_gui.bat

което ще покаже GUI програма, разработена от Texas Instruments за тест на Wi-Fi връзката.

В първия прозорец, който ще се покаже (с име Configure Connection Type) се натиска бутона Manual, който ще отвори нов прозорец (с име Network Interface Selection), в който трябва да изберем LAN картата на компютъра, която го свързва към рутера. Така ще реализираме сценария, показан на **фиг. 8.1**. Натиска се OK.

Ако всичко е минало без грешки, ще се отвори прозорец със слънчева система и планетата Сатурн (**фиг. 8.4**). Позицията на пръстените на Сатурн всъщност се контролира от потенциометъра на демо платката. Температурата се измерва от NTC, разположен също на демо платката. Цветът на Сатурн се изменя в червено или синьо, в зависимост от околната температура. Светодиод LED2 (мигащ) показва изпращането на съобщение към клиента (всяко съобщение съдържа X, Y, Z кординати, стойносттаVdd на демо платката и околната температура).

8.5.3. Да се реализира изпращане на низ от символи от демо платката с MSP430FR6989 към програмата Sock. За целта да се отвори файлът main.c от проекта sendstr в работното поле, отворено по-рано (CC3100_workspace). Да се разучи С кода в този файл. Под коментара:

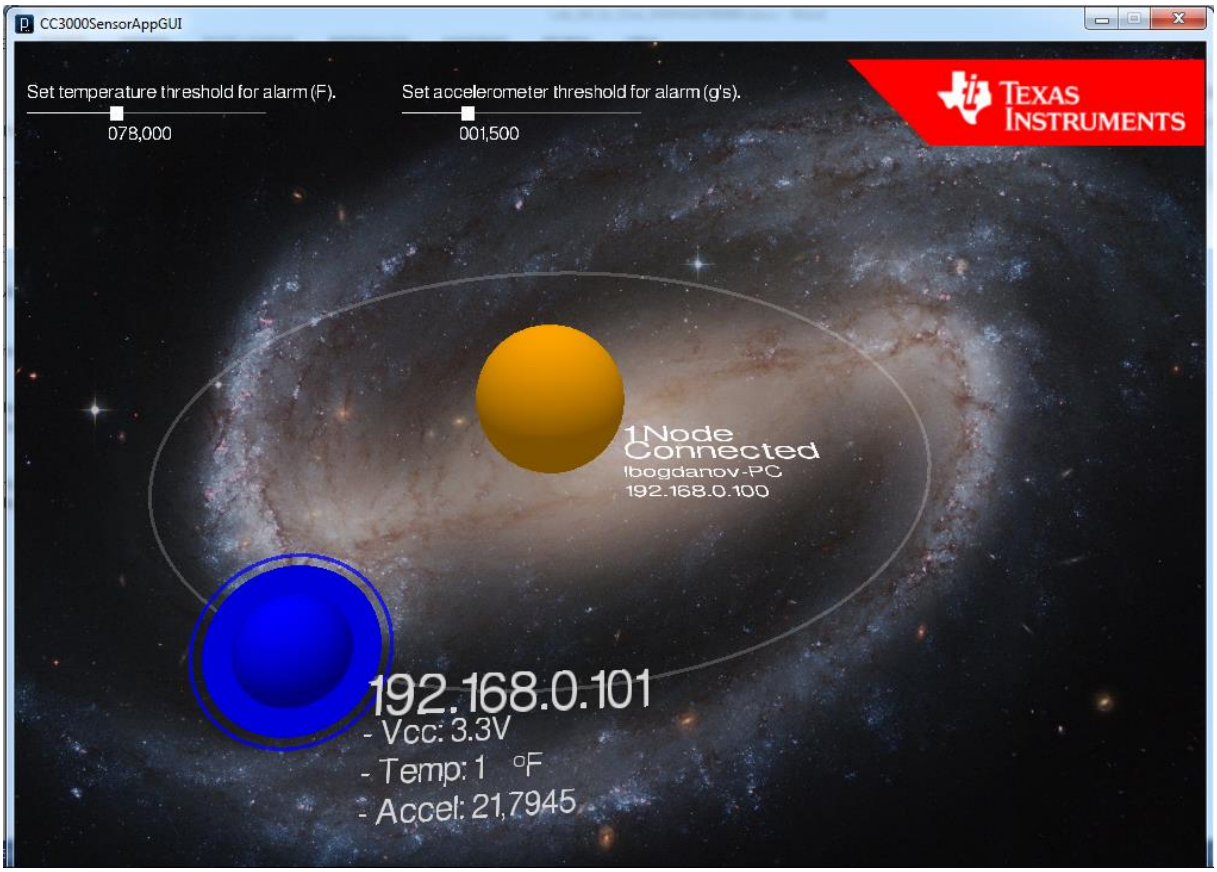
//Send string in TCP/IP terminal

да се извика функцията send( ) с необходимите параметри. Описание на функциите от библиотеката за CC3100 може да бъде намерено в MSHT_LAB/Documents/simplelink_doxygen_api/programmers_guide.html → след което се избира Modules → Socket_api →функция sl_Send( ).

След това се стартира програмата Sock в режим server и се въвежда IP-то на компютъра, на който върви то. За порт се избира числото, което е записано в макросът #define TCP_PORT от sendstr проекта (виж main.c). Натиска се бутона Listen.

Да се компилира и зареди фърмуерът sendstr. Ако свързването с рутера е минало успешно в терминала на COM порта трябва да се видят съобщенията от **фиг. 8.5**. Малко след това в сървърното приложение трябва

да се види низът Hello, World! Изпратете низ към MSP430FR6989 и вие от полето Message: на програмата Sock.



Фиг. 8.4

```
**************************
Starting sendstr client application ...
WLAN connection established!
----INFO----
Joined SSID: MSHT_PPMK
   Board IP: 192.168.0.101
    Gateway: 192.168.0.1
        DNS: 192.168.0.1
Establishing connection with TCP server...
```

Фиг. 8.5

8.5.4. Да се реализира уеб сървър, съдържащ елементарна уеб страница, която изобразява текущите стойности на АЦП от измерването на потенциометъра, терморезистора и делителя на напрежение. Проектът се казва websrv. Да се разучи файлът html_sensors.h. Забележете как в код на C се вгражда HTML – чрез низ от символи (в случая char *index_sens_html). За да е по-прегледен кодът, низът е разположен на няколко реда и в няколко променливи в хедърния файл. Към HTML кода са добавени символи за нов

ред \n и преди кавичките е сложен символът \ (което е изискване на C-низовете). Тоест HTML кодът трябва да претърпи известна трансформация, преди да бъде зареден в контролер, на който няма файлова система. В практиката се използват програми, които генерират хедърен (.h) C файл от HTML (.html) файл.

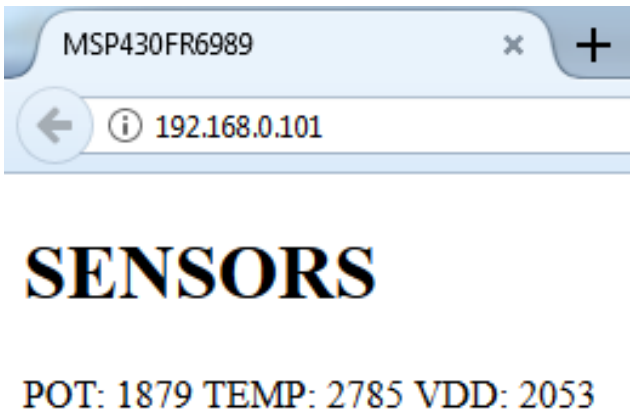
Да се извика функция от sensors.c, която попълва масива data_sens_html_dynamic с валиден HTML код, съдържащ стойностите на АЦП като низ. Това трябва да стане под коментара:

//Invoke the sensor reading function here

След това да се изпрати информацията, поместена в масива data_sens_html_dynamic, подобно на предходното упражнение – с функцията sl_Send( ), под коментара:

//Send the sensor data in HTML format here

Да се компилира и зареди програмата. Да се отвори уеб-браузър и в полето за име на сайт да се напише IP-то на демо платката. Натиснете Enter. Трябва да се появи уеб-страницата, показана на **фиг. 8.6**.



Фиг. 8.6

Пояснение: HTML кодът на програмата е следния:

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
 <HEAD>
    <TITLE>
      MSP430FR6989
    </TITLE>
 </HEAD>
 <BODY>
     <meta http-equiv=\"refresh\" content=\"2\" />
     <H1>SENSORS</H1>
     <P>POT: 0000 TEMP: 0000 VDD: 0000</P>
 </BODY>
</HTML>
```

Задача 8.5.3.

```
#include "simplelink.h"
#include "sl_common.h"
#include "simple_link_func.h"
#include "uartstdio.h"
#include "leds.h"

#define SSID_NAME       "MSHT_PPMK"          // Access point name to connect to.
#define SEC_TYPE        SL_SEC_TYPE_WPA_WPA2    // Security type of the Access piont
#define PASSKEY         "arm-cortex"          // Password in case of secure AP
#define PASSKEY_LEN     pal_Strlen(PASSKEY)     // Password length in case of secure AP
#define TCP_PORT        3333

int main(int argc, char** argv)
{
   SlSockAddrIn_t sock_addr;
   SlSecParams_t secParams = {0};
   _u32 server_ip_addr[4] = {192, 168, 0, 100};
   _u32 server_ip_converted = 0;
   _u8 *tcp_send_msg = "Hello, World!";
   _u8 tcp_recv_msg[128];
   _i16 tcp_recv_msg_size;
   _i16 client_socket = 0;

   stopWDT();
   initClk();
   CLI_Configure();
   init_leds();
   UARTprintf("\n\r************************\n\r");
   UARTprintf("Starting sendstr client application ...\n\r");

   configureSimpleLinkToDefaultState();

   sl_Start(0, 0, 0);

   secParams.Key = (_i8 *)PASSKEY;
   secParams.KeyLen = pal_Strlen(PASSKEY);
```

```
    secParams.Type = SEC_TYPE;
    sl_WlanConnect((_i8 *)SSID_NAME, pal_Strlen(SSID_NAME), 0, &secParams, 0);

    while((!IS_CONNECTED(g_Status)) || (!IS_IP_ACQUIRED(g_Status)))
    { _SlNonOsMainLoopTask(); }

    set_led(LED_RED, 1);

    UARTprintf("Establishing connection with TCP server... ");

    server_ip_converted = (server_ip_addr[0]<<24) | (server_ip_addr[1]<<16) | (server_ip_addr[2]<<8) |
(server_ip_addr[3]);
    sock_addr.sin_family = SL_AF_INET;
    sock_addr.sin_port = sl_Htons((_u16)TCP_PORT);
    sock_addr.sin_addr.s_addr = sl_Htonl(server_ip_converted);

    client_socket = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);

    sl_Connect(client_socket, ( SlSockAddr_t *)&sock_addr, sizeof(SlSockAddrIn_t));

    set_led(LED_GREEN, 1);

    UARTprintf("done!\n\r");

    //Send string in TCP/IP terminal
    ???

    while(IS_CONNECTED(g_Status)){
        tcp_recv_msg_size = sl_Recv(client_socket, tcp_recv_msg, 128, 0);

        if(tcp_recv_msg_size <= 0){
            break;
        }

        tcp_recv_msg[tcp_recv_msg_size] = '\0';

        UARTprintf("MSG from server: %s\n\r", tcp_recv_msg);

        blink_led(LED_GREEN);
    }

    sl_Close(client_socket);

    sl_WlanDisconnect();

    sl_Stop(SL_STOP_TIMEOUT);

    set_led(LED_RED, 0);
    set_led(LED_GREEN, 0);

    while(1){
        __bis_SR_register(LPM0_bits + GIE); //Enter sleep mode
    }
}
```

Задача 8.5.4.

```
#include "html_sensors.h"
#include "simplelink.h"
#include "sl_common.h"
#include "simple_link_func.h"
#include "uartstdio.h"
#include "leds.h"
#include "html.h"
#include "sensors.h"

#define SSID_NAME       "MSHT_PPMK"          // Access point name to connect to.
#define SEC_TYPE        SL_SEC_TYPE_WPA_WPA2    // Security type of the Access piont
#define PASSKEY         "arm-cortex"           // Password in case of secure AP
#define PASSKEY_LEN     pal_Strlen(PASSKEY)     // Password length in case of secure AP
#define MAX_MSG_SIZE    20
#define TCP_PORT        80

int main(int argc, char** argv)
{
    uint8_t data_sens_html_dynamic[40];
    SlSecParams_t secParams = {0};
    SlSockAddrIn_t server_sock_addr;
    _u16 server_sock_addr_size = sizeof(SlSockAddrIn_t);
    SlSockAddrIn_t client_sock_addr;
    _u16 client_sock_addr_size = sizeof(SlSockAddrIn_t);
    _i16 server_socket = 0;
    _i16 client_socket = -1;
    char recv_buff[255];
    unsigned long html_size;

    stopWDT();
    initClk();
    CLI_Configure();
    init_leds();
    init_sensors();
    UARTprintf("\n\r************************\n\r");
    UARTprintf("Starting web server application ...\n\r");

    configureSimpleLinkToDefaultState();

    sl_Start(0, 0, 0);

    sl_NetAppStop(SL_NET_APP_HTTP_SERVER_ID); //Stop internal SimpleLink web page on port 80

    secParams.Key = (_i8 *)PASSKEY;
    secParams.KeyLen = pal_Strlen(PASSKEY);
    secParams.Type = SEC_TYPE;
    sl_WlanConnect((_i8 *)SSID_NAME, pal_Strlen(SSID_NAME), 0, &secParams, 0);

    while((!IS_CONNECTED(g_Status)) || (!IS_IP_ACQUIRED(g_Status)))
    { _SlNonOsMainLoopTask(); }

    set_led(LED_RED, 1);

    server_sock_addr.sin_family = SL_AF_INET;
    server_sock_addr.sin_port = sl_Htons((_u16)TCP_PORT);
    server_sock_addr.sin_addr.s_addr = 0;

    server_socket = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, SL_IPPROTO_TCP);
```

```
    sl_Bind(server_socket, (SlSockAddr_t *)&server_sock_addr, server_sock_addr_size);

    sl_Listen(server_socket, 1);

    while(1){

        while(client_socket < 0){
            client_socket = sl_Accept(server_socket, ( struct SlSockAddr_t *)&client_sock_addr,
(SlSocklen_t*)&client_sock_addr_size);
            _SlNonOsMainLoopTask();
        }

        //Wait for HTTP GET request
        sl_Recv(client_socket, recv_buff, 255, 0);
        UARTprintf("MSG from client: %s \n\r", recv_buff);

        //Static Hello, WORLD web page---------
        //unsigned char *html_msg = "<HTML><HEAD></HEAD><BODY>Hello,
WORLD!</BODY></HTML>";
        //unsigned char *html_msg = "HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n<HTML>\n<BODY>\n\t<H1>Hello, WORLD!</H1>\n</BODY>\n</HTML>\n";
        //sl_Send(client_socket, html_msg, strlen(html_msg), 0);
        //--------------------------------------

        //Invoke the sensor reading function here
        ???

        html_size = strlen(index_sens_html);
        sl_Send(client_socket, index_sens_html, html_size, 0);

        //Send the sensor data in HTML format here
        ???

        html_size = strlen(close_sens_html);
        sl_Send(client_socket, close_sens_html, html_size, 0);

        sl_Close(client_socket);
        client_socket = -1;

        _SlNonOsMainLoopTask();

        blink_led(LED_GREEN);
    }
}
```