

## 1. Laborator

### 1.1. Scopul laboratorului

Informațiile prezentate în cele ce urmează sunt esențiale pentru înțelegerea mecanismelor intime de pornire a diferitelor tipuri de sisteme *embedded*.

Următoarele obiective vor fi atinse în cadrul acestui laborator:

1. Înțelegerea diferitelor mecanisme de *boot*-are a sistemelor *embedded*.
2. Realizarea tuturor conexiunilor și setărilor (*software* și *hardware*) necesare pentru trimiterea și rularea imaginii sistemului de operare (SO).
3. Înțelegerea principiilor și caracteristicilor diferitelor modalități de dezvoltare a aplicațiilor *software* pe calculatorul personal (PC – *personal computer*) și de testare a funcționalității acestuia direct pe sistemul *embedded*.

### 1.2. Introducere

În cadrul laboratorului de „Sisteme *embedded* inteligente” există un număr de 3 sisteme diferite de dezvoltare, de tip *embedded*. Acestea sunt:

1. **OMAP3530 EVM** (*E*valuation *M*odule) – procesorul fiind implementat pe o arhitectură ARM;
2. **BeagleBoard-xM** – procesorul având o arhitectură ARM;
3. **eBox-3310** – procesorul fiind implementat pe o arhitectură x86.



Figura 1.1. Sistemele: (a) OMAP3530 EVM, (b) BeagleBoard-xM și (c) eBox-3310

Pentru a fi capabili să rulăm sistemul de operare (SO) **Windows Embedded Compact™** pe aceste sisteme de dezvoltare, prima dată, trebuie construită imaginea sistemului de operare (SO) – deci, trebuie obținut fișierul **nk.bin** – fișier ce conține această imagine.

Fișierul **nk.bin** conține toate fișierele ce compun SO (*kernel*-ul SO, *driver*-ele, aplicațiile, fișierele de configurare, baza de date *registry* etc.). Obținerea acestui fișier se realizează prin compilarea diferitelor componente din care este compus SO în cadrul mediului de dezvoltare IDE (*Integrated Development Environment*) Microsoft Visual Studio. Pe calculatoarele din laborator veți găsi următorul director care conține un subfolder specific pentru fiecare nouă imagine a SO Windows Embedded Compact dezvoltată:

Pentru cele 3 plăci vor exista următoarele sub-directoare individualizate:

1. **OMAP3530\_EVM** - pentru sistemul de dezvoltare **OMAP3530 EVM**,
2. **BeagleBoard-xM** - pentru placa **BeagleBoard-xM** și
3. **eBox-3310** - pentru sistemul **eBox-3100**.

Încărcați unul din aceste proiecte, funcție de sistemul *embedded* cu care lucrați, în mediul de dezvoltare Microsoft Visual Studio:

1. Lansați mediul de dezvoltare **Microsoft Visual Studio**, dând click pe icoana existentă pe *desktop*.
2. Apoi: **File** → **Open** → **Project/Solution**....
3. Din structura de directoare anterior menționate (specifică fiecărei plăci de dezvoltare utilizate) alegeți fișierul corespunzător cu extensia „.sln”<sup>1</sup> (*solution file*).
4. Selectați componentele necesare funcționării SO în conformitate cu necesitățile proprii aplicații. În proiectele existente acestea sunt deja selectate. Dvs. doar parcurgeți structura ierarhică ce conține componentele SO pentru a vizualiza care din acestea au fost utilizate în construirea imaginii SO.
5. Compilați SO.

Din punct de vedere practic subpunctele 4 și 5 (prezentate anterior) nu vor fi realizate practic în cadrul acestui laborator, în special datorită lipsei informațiilor necesare selectării componentelor optime minime necesare funcționării corecte a sistemului *embedded* (punctul 4) și a timpului foarte mare necesar compilării imaginii SO (punctul 5 – peste 30 de minute pe un sistem cu un procesor Intel Core2 Duo, la 3 GHz, ce rulează SO Windows XP SP3, cu antivirusul invalidat). Aceste două subpuncte subcapitole (4 și 5) vor fi realizate în alt laborator.

În urma compilării diferitelor componente ce intră în compoziția SO, se obține imaginea SO Windows Embedded Compact, deci, se obține fișierul **nk.bin**, precum și un număr de alte fișiere necesare diferitelor etape în cadrul procesului de *boot*-are a SO (precum **MLO**, **XLDR**, **EBOOTSD.nb0** etc.).

Încărcarea fișierului **nk.bin** pe sistemul *embedded* și lansarea în execuție a SO se poate realiza în mai multe modalități: prin *ethernet* (cablu de rețea), conexiune USB, conexiune serială, prin scrierea imaginii în memoria flash a sistemului *embedded*, prin copierea pe un SD card sau USB Flash și lansarea în execuție etc.

Deoarece în cadrul diferitelor laboratoare vom modifica imaginea SO prin adăugarea de noi componente și/sau configurarea altora (configurare realizată prin intermediul înregistrărilor din *registry*) aceasta va implica o dinamică continuă a imaginii SO. Din acest motiv vom încărca imaginea SO la începutul fiecărui laborator, sau de câte ori este necesar, prin intermediul conexiunii *ethernet* sau USB.

### 1.3. Conectarea sistemului *embedded* cu PC-ul

Din punct de vedere practic trebuie să urmărim 2 aspecte:

1. încărcarea și execuția imaginii sistemului de operare și
2. dezvoltarea, încărcarea, execuția și depanarea aplicației care va rula în cadrul SO, anterior încărcat și executat.

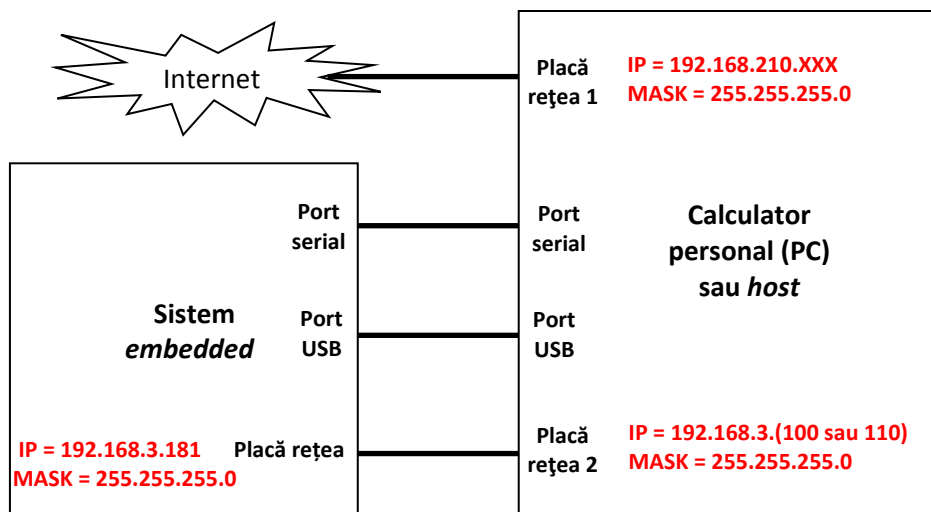
Deoarece imaginea SO va fi trimisă prin *ethernet* sau printr-o conexiune USB (funcție de sistemul *embedded* instalat), în mod minimal una din aceste două conexiuni trebuie să existe. În cazul conexiunii USB ne vom folosi de protocolul **RNDIS** (*Remote Network Driver Interface Specification*) ce furnizează o conexiune virtuală *ethernet* prin intermediul protoalelor de comunicație ce susținute de standardul USB (*Universal Serial Bus*).

Depanarea programelor poate fi realizată în mod similar prin conexiune *ethernet* sau USB. În plus, mai este

---

<sup>1</sup> Fișierul cu extensia „.sln” este un fișier de tip text. Acest fișier înglobează diferite structuri de date necesare organizării proiectelor în Visual Studio. El este similar, din punct de vedere conceptual, fișierelor „.prj” din Borland C sau celor de tipul project workspace („.dsw”) din mediul Visual C++ 6.0.

necesară o conexiune serială pentru configurarea procesului de *boot*-are. În funcție de sistemul *embedded* realizarea conexiunilor se poate realiza în mai multe moduri posibile, vezi **Figura 1.2**. Această diversitate a sistemelor *embedded* (moduri de conectare, arhitecturi, periferice înglobate, tipuri de medii de stocare a datelor etc.) este caracteristică acestui domeniu în care există foarte puține constrângeri, fiecare producător având propria soluție tehnică. Comparând piața sistemelor *embedded* cu cea a calculatoarelor personale (unde arhitectura unui PC este standardizată) putem spune că prima este una de-a dreptul „democratică”, chiar cu tendințe “anarhice”.



**Figura 1.2.** Potențialele conexiuni între sistemul *embedded* și calculatorul personal (*host*)

În **Tabelul 1.1.** sunt prezentate modalitățile de comunicare cu dispozitivele existente în laborator, iar în **Figura 1.2** sunt prezentate pe lângă conexiunile posibile și setările existente pe sistemul *host* pentru plăcile de rețea existente. Se observă din tabel că placa de dezvoltare OMAP3530 EVM oferă cea mai mare flexibilitate posibilă, permițând comunicarea cu ea și încărcarea imaginii SO prin numărul maxim de medii.

**Tabelul 1.1.** Tipul sistemului *embedded* și modalitățile de conectare la sistemul *host*

	Configurarea procesului de <i>boot</i> -are - comunicare cu EBOOT (port sistem)	Încărcare imagine SO (prin/de la)	Încărcare și dezvoltare programe	Parametrii comunicație serială
<b>BeagleBoard-xM</b>	Serială (UART1)	USB/SD	USB	115200 baud, 8 biți de date, fără paritate, 1 bit stop
<b>OMAP3530 EVM</b>	Serială (UART3)	Ethernet/USB/SD/ Flash	USB / Ethernet	
<b>eBox-3310</b>	Serială (UART1 – COM1)	Ethernet/Flash	Ethernet	38400 baud, 8 biți de date, fără paritate, 1 bit stop

#### 1.4. Încărcarea SO – concepte fundamentale

După crearea imaginii SO, următorul pas este *boot*-area acestuia. Prin *boot*-area (*boot* în limba engleză – prescurtarea de la *bootstrapping*) se înțelege procesul prin care, în urma resetării unui sistem, acesta se pregătește (de exemplu își verifică și își inițializează diferitele dispozitive componente) pentru încărcarea unui sistem de operare oarecare.

**Calculatoarele personale** (*desktop*-urile, *laptop*-urile) și nu numai (de ex. serverele) realizează acest proces prin intermediul BIOS-ului care descoperă, verifică și inițializează diferitele subsisteme și, în final, încarcă SO.

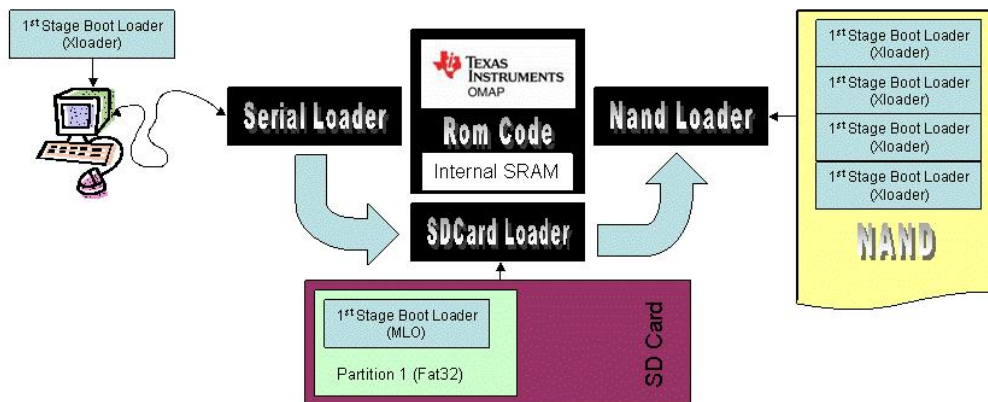
**Sistemele de tip *embedded***, unele dintre ele mai limitate în resurse, realizează procesul de *boot*-are în mai mulți pași. De regulă acest proces implică trei pași. În cazul nostru, a sistemelor OMAP3530 EVM și BeagleBoard-xM, procesul de încărcare a imaginii sistemului de operare este unul în trei pași.

În **primul pas** un mic program *bootloader* ce rezidă în memoria ROM (internă sistemului *embedded*) localizează, încarcă (în RAM) și lansează un program de mici dimensiuni denumit *first-stage bootloader*. Acesta încarcă și execută – deja vorbim de **cel de al doilea pas** – un program mai mare tot de tip *bootloader* (denumit și *second-stage bootloader*). Acest ultim element de tip *bootloader*, în **ultimul pas (al treilea)**, inițializează întreg sistemul *embedded*, încarcă și, în final, lansează sistemul de operare (SO). În continuare vom prezenta în detaliu acești pași într-un mod mai detaliat.

### Primul pas

În cadrul acestei etape, codul existent în ROM-ul, intern SoC-ului, va încerca să *boot*-eze prin intermediul diferitelor periferice existente, precum: serial, SD Card, memoria internă NAND sau de pe un dispozitiv conectat la USB (USB *flash*).

Ordinea în care procesorul va încerca să *boot*-eze este determinată de informația existentă (nivel zero sau unu logic) pe un număr de linii de intrare (de tip GPIO – *General Purpose Input/Output*) cunoscute sub numele de linii de tip SYS\_BOOT.



**Figura 1.3.** Ordinea de *boot*-are a sistemului de dezvoltare OMAP35 EVM

Acest mic program de tip *bootload*-er pe care ROM-ul (plăcii de dezvoltare) caută să-l încarce și să îl execute este denumit **XLDR** (denumirea este pentru SO Windows și vine de la *eXtensive LoadDeR*) sau **x-load** (pentru SO Linux). Programul din ROM-ul intern SoC-ului realizează anterior și o minimă inițializare a dispozitivelor *hardware* de unde încearcă să încarce *bootload*-erul **XLDR** sau **x-load** și ulterior îl încarcă pe acesta la o locație fixă în SRAM-ul intern. Imediat după aceea **XLDR** (sau **x-load**) este lansat în execuție.

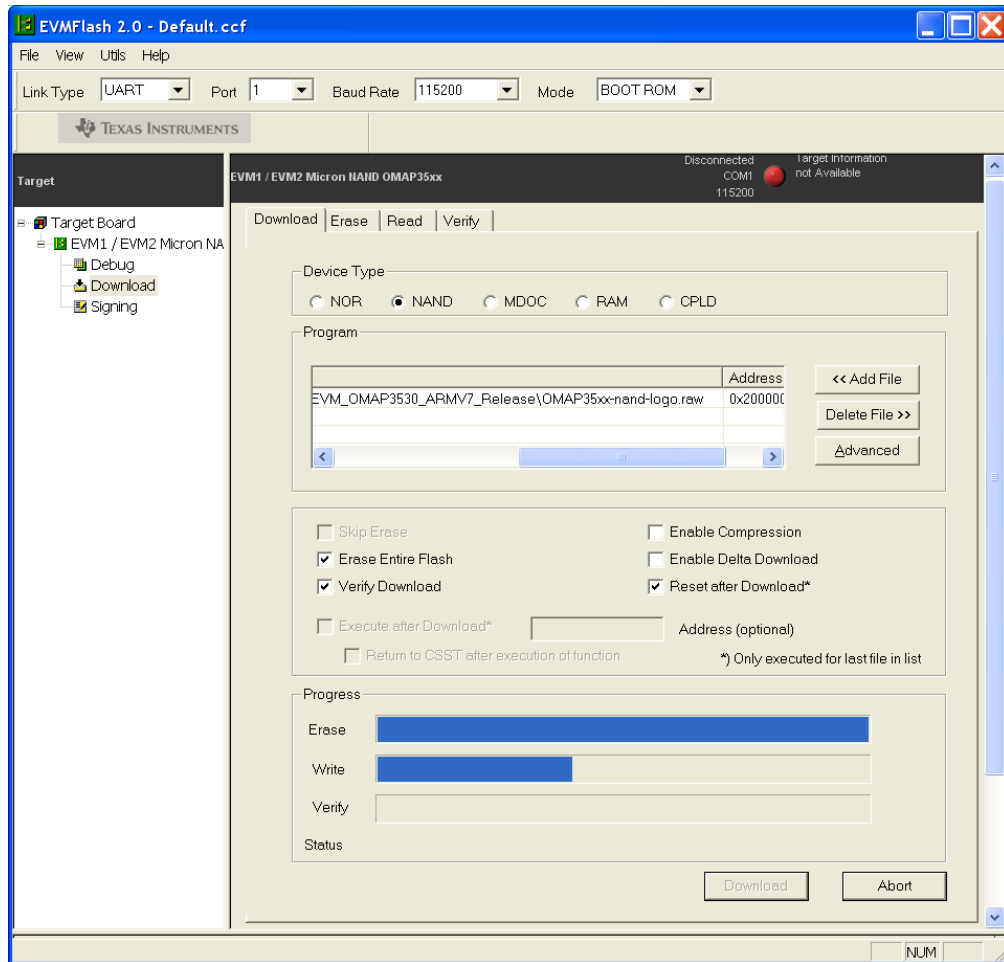
În **Figura 1.3** se prezintă acest prim pas a procesului de *boot*-are în care SoC-ul OMAP caută să *boot*-eze de pe portul serial, apoi de pe SD card și ulterior din memoria *flash* NAND.

În cazul *boot*-ării prin intermediul conexiunii seriale un simplu ID este scris în portul serial, dacă calculatorul *host* (un PC) răspunde corect într-un anumit interval de timp, programul din ROM va încărca în SRAM *bootloader*-ul XLDR predându-i controlul, ulterior încărcării.

În cazul în care în de la portul serial nu se primește nici un răspuns, programul din ROM va căuta să încarce modulul **XLDR** de pe SD card (ordinea de *boot*-are poate fi schimbată funcție de starea liniilor SYS\_BOOT). Dacă pe unul din controlerile de SD card (MMC1 și/sau MMC2 în cazul OMAP3530 EVM) este găsit un SD card, programul din ROM caută în prima partiție, din tabelul partițiilor, un fișier cu numele MLO (fișier ce conține codul binar al *bootloader*-ului **XLDR** sau **x-load** și un antet ce specifică locația de memorie unde va fi încărcat fișierul și lungimea lui). Dacă totul decurge normal *first-stage* *bootloader* este încărcat și preia controlul

sistemului.

În cazul *boot*-ării din memoria flash NAND, aceasta este împărțită în blocuri (sau sectoare). Programul din ROM va încerca să încarce *bootstrap*-erul din primele 4 sectoare, începând cu primul și continuând cu celelalte. În situația în care în unul din acestea este găsită informația corectă, *bootloader*-ul este încărcat și preia controlul sistemului.



**Figura 1.4.** Interfața grafică a programului de scriere a *bootloader*-elor **XLDR** și a **EBOOT** în memoria Flash

*First-stage* bootloader este un fișier de dimensiuni reduse, în special datorită constrângerilor de a fi copiat și executat în memoria SRAM. În urma compilării imaginii SO (ne referim aici la SO Windows Embedded Compact) va rezulta și fișierul **XLDR**, dar numai atunci când configurația în care se realizează compilarea este de tip **Release**. În configurația de tip **Debug** datorită codului intern a diferitelor librării (care este neoptimizat în varianta **Debug**), datorită informațiilor suplimentare de *debug* introduse în codul rezultat lungimea fișierului **XLDR** va fi mai mare decât cea permisă. Din acest motiv fișierul **XLDR** nu este generat în configurația **Debug**. Există variante diferite ale fișierului **XLDR** funcție de dispozitivul prin intermediul căruia se realizează încărcare și prin care se va realiza încărcarea etapei doi din *bootloader*.

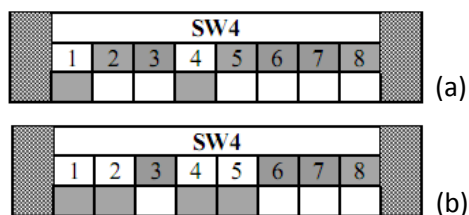
În cazul SO Linux procesul este similar; fișierul **x-load** rezultă și el în urma procesului de compilare.

Atât *first-stage* bootloader cât și *second-stage* bootloader comunică prin intermediul interfeței seriale, trimițând informații despre procesele momentane în desfășurare sau cerând date pentru configurarea diferiților pași (numai *second-stage* bootloader), vezi **Figura 1.6**.

Pentru încărcarea codului din **XLDR** în memoria Flash, fiecare producător oferă propriile programe de scriere în Flash. De exemplu, pentru placa OMAP3530 EVM compania TI furnizează programul EVMFlash, vezi o imagine cu interfața grafică a programului în **Figura 1.4**. Fișierul care se programează în memoria Flash va fi **OMAP35xx-nand-logo.raw**. Acest fișier înglobează atât *first-stage* bootloader (XLDR sau x-load) cât și *second-stage* bootloader (EBOOT).



În momentul în care se programează memoria Flash trebuie avută în vedere geometria acesteia (de exemplu placa TI **OMAP3530 EVM** este construită în două variante constructive, cu două tipuri diferite de memorii), cât și modalitatea prin care se realizează programarea (prin intermediul portului serial sau USB – opțiunea **Link Type** din partea de sus, dreapta a ferestrei din **Figura 1.4**). De asemenea, placa de dezvoltare **OMAP3530 EVM** trebuie să fie configurată pentru a putea fi programată – prin intermediul grupului de comutatoare SW4 – vezi **Figura 1.5**.



**Figura 1.5.** Poziția bancului de comutatoare SW4 pentru permiterea accesului la memoria flash prin intermediul (pozițiile pe alb reprezintă starea ON a comutatorului): (a). Portului UART sau a (b). Portului USB

Anterior programării memoriei *flash*, zonele de memorie în care rezidă *bootloader*-ele XLDR și EBOOT vor fi șterse (se determină geometria memoriei *flash* și se șterg toate blocurile care sunt rezervate – aici se găsesc *bootloader*-ele **XLDR** și **EBOOT**).

### Cel de al doilea pas

Arhitectura *software* ce susține procesul de încărcare implică existența a doi pași, a două programe de tip *bootloader*. Despre primul am discutat la punctul precedent. Al doilea *bootloader* se regăsește sub numele: **EBOOT** (în Windows) și **U-boot** (în SO Linux). Acest *bootloader* este încărcat de *first-stage* bootloader (**XLDR** sau **x-load**).

*First-stage* bootloader (**XLDR** sau **x-load**) este cel care încarcă *second-stage* bootloader și îi predă controlul (îl execută).

O particularitate a acestor două programe de *boot*-are este dată de locul de stocare a lor – atunci când ele se regăsesc în memoria flash. Primele 4 sectoare sunt rezervate întotdeauna pentru **XLDR** sau **x-load** în timp ce începând cu sectorul 5 începe modulul **EBOOT** sau **U-boot**.

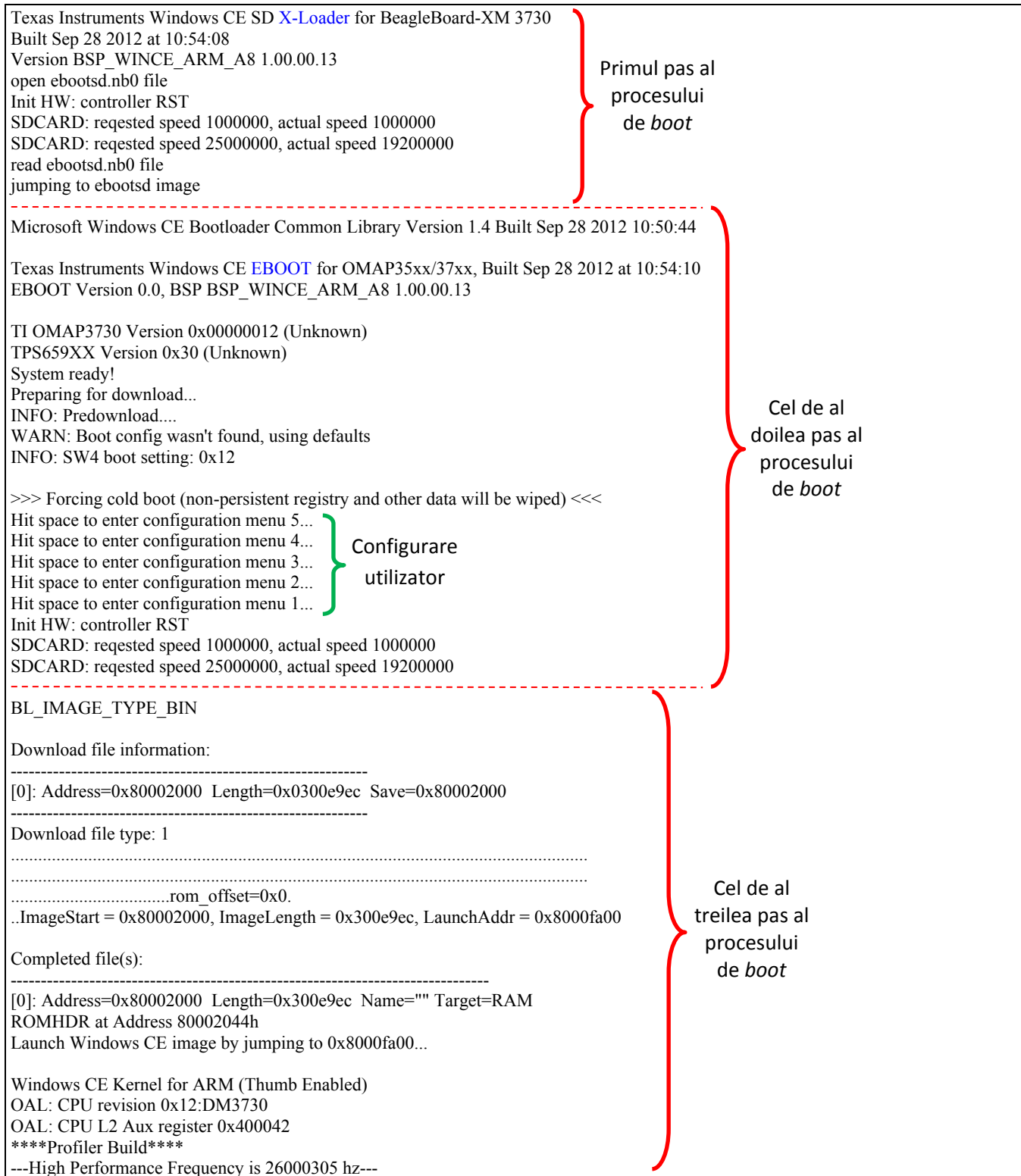
Pentru ca *second-stage* să fie capabil să încarce imaginea SO, obținută în urma compilării, sunt necesare informații suplimentare de comunicare între mediul de dezvoltare și aplicația **EBOOT**. Din acest motiv se poate iniția o comunicație serială cu modulul **EBOOT** pentru configurarea diferiților parametri necesari transferului imaginii SO și/sau pentru configurarea și pregătirea diferitelor dispozitive *hardware* a sistemului *embedded* în vederea susținerii procesului de *boot* (de ex. formatarea memoriei *flash*).

În aplicația de tip terminal prin intermediul căreia comunicăm cu modulul **EBOOT** vom fi avertizați la un anumit moment să apăsăm spațiu pentru a intra în meniul de configurare a modulului **EBOOT**, vezi **Figura 1.6**.

### În cel de al treilea pas

*Second-stage* bootloader (**EBOOT** sau **U-boot**) încarcă imaginea SO (fișierul **nk.bin** în Windows sau **uImage** în Linux – în SO Linux în acest fișier se găsește doar kernel-ul SO !!!) și îl lansează în execuție. Etapele acestui ultim pas pot fi vizualizate în aplicația terminal care afișează datele recepționate serial de la modulul **EBOOT**, **Figura 1.6**.

Având imaginea sistemului de operare, deci fișierului **nk.bin**, în continuare vom prezenta trei modalități de încărcare și executare a acesteia pe diferite sisteme de dezvoltare.



**Figura 1.6.** Informațiile afișate în timpul procesului de boot-are

Transferul acestui fișier se poate realiza prin intermediul mai multor căi: SD-card, USB, *ethernet*, USBFn RNDIS etc. Deoarece utilizarea unui SD-card sau a unui USB necesita un număr mare de operații mecanice (introducerea și scoaterea memoriei SD) s-au ales celelalte metode de boot-are a SO. Astfel, pentru **OMAP3530 EVM** și **eBox-3310** se va utiliza pentru transfer conexiunea *ethernet* în timp ce pentru **BeagleBoard-xM** portul USB (în BSP-ul oferit de producător pentru acest sistem de dezvoltare transferul pe *ethernet* nu este implementat).

## 1.5. Boot-area SO de pe un SD Card

Pașii necesari *boot*-ării SO de pe un SD *card* sunt următorii:

1. Formatați **SD Card**-ul sub orice sistem de operare (SO) utilizând **doar** FAT32 – pentru sistemul de fișiere. Deoarece codul existent în ROM-ul intern SoC-ului este de mici dimensiuni, nu au fost implementate subrutine de accesare pentru toate versiunile posibile de sisteme de fișiere (FAT12, FAT16/FAT16B, FAT32, FAT+, exFAT, NTFS etc.).
2. Configurați cardul SD pentru ca acesta să fie *boot*-abil. Astfel, cu ajutorul unui program ce vă permite să editați valorile din cadrul sectoarelor *card*-ului (de ex. **HxDSetupEN** sau **Roadkil Sector Editor**), scrieți valoarea 0x80 (prin aceasta se marchează MBR drept *boot*-abil) în octetul situat la adresa 0x1BE (se marchează drept *boot*-abilă prima partiție din tabelul partițiilor).
3. Se salvează modificările făcute.
4. Se copie în primul pas doar fișierul MLO pe SD *card* astfel încât acesta să fie primul în structura de fișiere.
5. Se copie ulterior fișierele **EBOOTSD.NB0** și **NK.BIN**. Toate aceste fișiere le veți găsi în directorul *release* urmând calea:

**c:\WINCE700\OSDesign\BeagleBoard-xM\BB-XM\RelDir\BB-XM\_ARMV7\_Release**

Calea anterioară este dată pentru placa **BeagleBoard-xM**, pentru celelalte sisteme de dezvoltare calea către fișierele necesare *boot*-ării se formează în mod asemănător.

6. Se configurează placa astfel încât *boot*-area să aibă loc de pe SD card:
  - a. în cazul plăcii **OMAP3530 EVM** se pun comutatoarele *switch*-ului SW4 în pozițiile 1-3 ON, 4-5 OFF, 6 ON, 7-8 OFF.

SW4 Switch Position							
1	2	3	4	5	6	7	8
ON	ON	ON	OFF	OFF	ON	OFF	OFF

**Figura 1.7.** Pozițiile comutatoarelor SW pentru *boot*-are de pe SD card

- b. în cazul sistemului de dezvoltare **BeagleBoard-xM** acesta va *boot*-a în mod implicit de pe SD *card*, nemaifiind necesare configurări suplimentare
7. Se introduce cardul SD în sistemul *embedded* – **care nu este alimentat în acest moment!**
  8. Se alimentează sistemul *embedded*; acesta va *boot*-a direct din SD *card* (în cazul plăcii OMAP3530 EVM în principal datorită poziției, anterior menționate, a comutatoarelor.

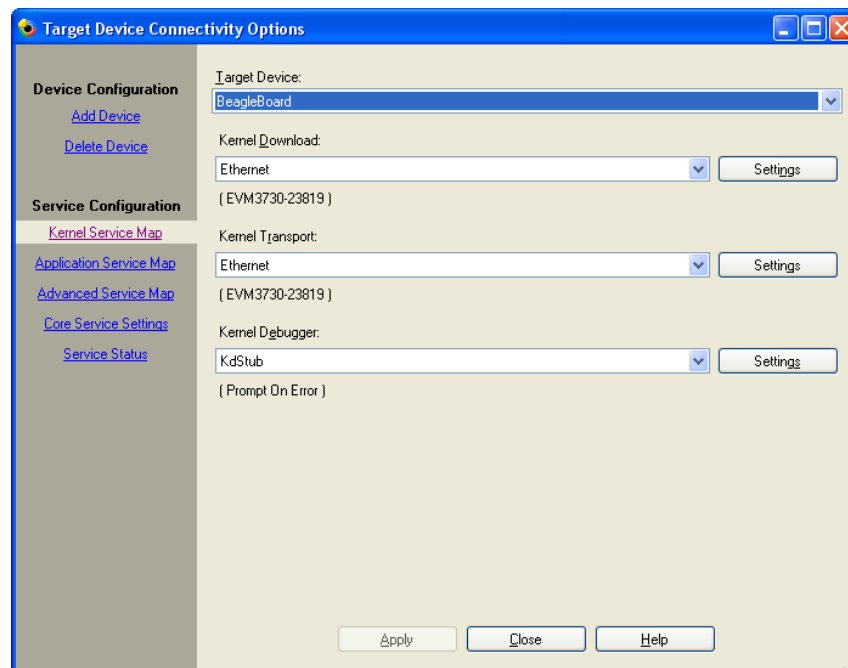


## 1.6. Boot-area SO prin intermediul conexiunii realizate prin USB

Protocolul **RNDIS** (*Remote NDIS*<sup>2</sup>) permite stabilirea unei conexiuni *ethernet* prin intermediul unei conexiuni realizată prin USB. Pentru realizarea unei conexiuni **USB RNDIS** este necesară instalarea unor drivere pe **PC host** și, în plus, trebuie atribuită o adresă statică de IP adaptorului de rețea RNDIS utilizat în comunicarea cu sistemul *embedded*.

Pentru încărcarea și *boot*-area SO prin intermediul mecanismului USB RNDIS trebuie urmați pașii:

1. Conectați un cablu de tipul USB A/B între portul USB al sistemului pe care se dezvoltă imaginea SO (sistemul *host*) și portul USB de tipul OTG (On-The-Go) a sistemului *embedded* (TI **OMAP3530 EVM** sau **BeagleBoard-xM**).
2. Conectați cablul serial între sistemul *embedded* și cel *host*.
3. Porniți aplicația **Tera Term** (aveți un *shortcut* pe *desktop*-ul sistemului *host*), selectând portul “**Com 1**” (portul serial prin intermediu căruia realizați conexiunea, vezi **Figura 1.2**) drept port de comunicare cu sistemul *embedded*. Configurați parametrii comunicației seriale în conformitate cu datele prezentate în **Tabelul 1.1**. (**Setup** → **Serial port** ...), după configurare apăsați OK.
4. Alimentați sistemul *embedded*.
5. În fereastra terminal apăsați SPACE când vă este cerut, pentru a avea posibilitatea setării parametrilor transferului.
6. Alegeți pentru opțiunea “*boot device*”: **USBFn RNDIS** (în meniul “**Select Boot Device**”).
7. Alegeți “*debug device*” **USBFn RNDIS** (în meniul “**Select KITL (Debug) Device**”).



**Figura 1.8.** Configurarea opțiunilor de comunicație pentru dispozitivul *target* (sistemul *embedded*)

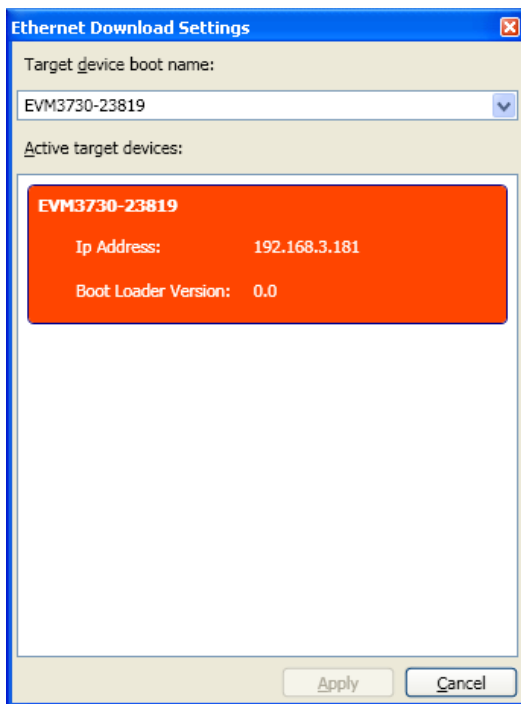
8. În cadrul rubricii “Network Settings”:
  1. Selectați “Disable DHCP” – alegem această opțiune deoarece pe rețea nu există un server de tip DHCP care să ofere o adresă de IP sistemului de dezvoltare;
  2. Puneți adresa de IP statică: **192.168.3.181**;

<sup>2</sup> *Network Driver Interface Specification* – este o specificație caracteristică SO Windows care precizează cum protocoalele de comunicație, precum TCP/IP, și driverele dispozitivelor de rețea trebuie să comunice.

3. Iar masca IP: **255.255.255.0**.

Toate aceste setări vor caracteriza sistemul *embedded* doar în timpul procesului de *boot*. Imediat ce EBOOT își termină rolul – atunci când imaginea SO a fost încărcată și executată –, noile setări ale plăcii de rețea, gestionată de această dată de SO Windows Embedded Compact, ce tocmai a fost încărcat, pot fi altele.

9. Dacă nu ați încărcat deja proiectul cu imaginea SO, realizați pașii 1-5 din cadrul **Subcapitolului 1.2.** prezentați anterior.
10. În cadrul **Platform Builder**, deci în mediul **Visual Studio**, selectați **Target** → **Connectivity Options** și aveți grijă ca următoarele opțiuni să fie selectate (vezi **Figura 1.8.**):
  - a. Kernel Download puneți Ethernet
  - b. Kernel Transport puneți Ethernet
  - c. Kernel Debugger puneți KdSub
11. Apăsăți pe butonul de *Settings* și lăsați deschisă fereastra care a apărut. Fereastra este similară cu cea din **Figura 1.9** dar fără a prezenta existența unui sistem activ în câmpul “**Active target devices**”.



**Figura 1.9.** Selectarea dispozitivului *embedded*, în care vom încărca imaginea SO, din cele disponibile

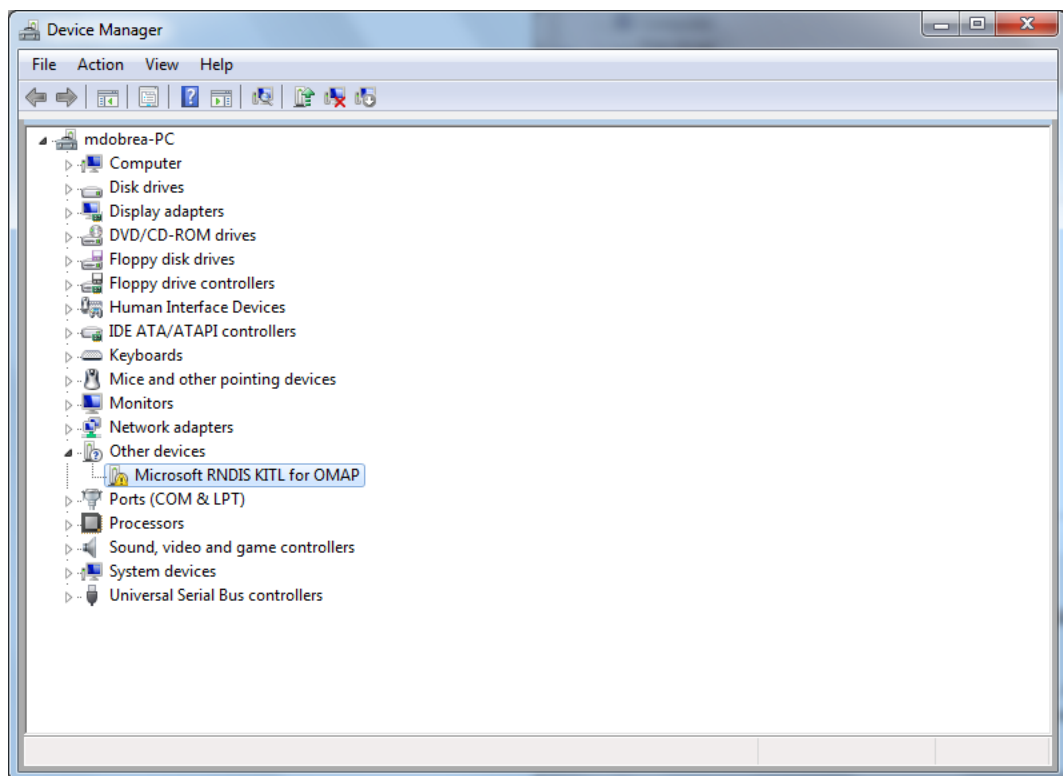
12. În meniul de comunicație cu *bootloader*-ului **EBOOT**, din aplicația terminal, selectați “**Exit and Continue**” pentru începerea procesului de încărcare a imaginii SO.
13. În urma acestei operații trebuie să obțineți inițializarea corectă a RNDIS și să vizualizați, în programul terminal, o secvență de răspuns primită de la sistemul *embedded* similară cu următoarea:

```
Selection: 0
HostMiniInit:: !!
Rndis:: initialization: with addr=480ab000
Rndis:: Address static map to addr=b60ab000
Rndis:: initialization!
Rndis:: PDDInit Success!
Rndis:: Get MAC address 200,902f,b5d
RndisMdd:: PDD's max RX buffer = [8192] bytes.
Rndis:: initialization: Success
```

Ceea ce ne interesează este să obținem mesajul: ***“Rndis:: initialization: Success”*** ceea ce ne confirmă realizarea unei conexiuni corecte între sistemul *embedded* (mai exact a aplicației **EBOOT** care rulează acum pe acesta) și mediul Visual Studio.

Următoarele **trei** etape se aplică doar în situația în care sistemele cu care lucrați nu sunt configurate și nu sunt instalate *driver*-ele necesare.

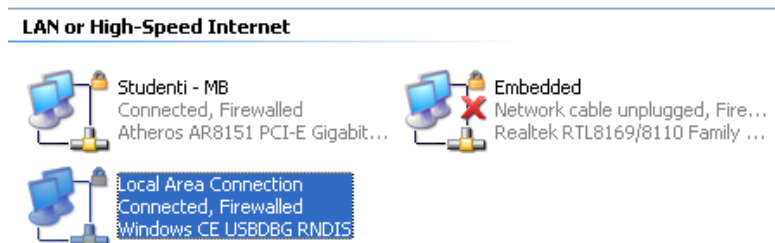
14. Dacă în cadrul ultimului mesaj (din cele prezentate anterior) nu obțineți textul ***“Rndis:: initialization: Success”*** este posibil ca *driver*-ul de pe PC *host* să nu se fi instalat corect. Iar în **Device Manager**-ul PC-ului *host* veți observa existența unui driver de tip RNDIS incorect instalat (marcat cu semnul exclamării), similar cu imaginea prezentată în figura de mai jos. Pentru instalarea corectă a acestui driver urmați pasul imediat următor.



**Figura 1.10.** În **Device Manager** suntem atenționați că nu este instalat sau este instalat incorect *driver*-ul pentru dispozitivul RNDIS

15. Dacă calculatorul vă cere un driver pentru USB RNDIS. Instalați-l din una din următoarele locații:
  - i. Pentru Windows Embedded Compact 7.0:
    - În cazul în care pe calculatorul *host* aveți instalat SO Windows Vista sau Windows 7 locația în care găsiți *driver*-ul este următoarea:  
**C:\Wince700\Platform\Common\Src\Common\Kitldr\Usbdbg\Usbdbgrndismdd\Host\Vista\**
    - Pentru Windows XP:  
**C:\Wince700\Platform\Common\Src\Common\Kitldr\Usbdbg\Usbdbgrndismdd\Host\XP\**
  - ii. Windows CE 6.0 R3:  
**C:\WINCE600\PUBLIC\COMMON\OAK\DRIVERS\ETHDBG\RNDISMIN\HOST**
16. În **Network connections**, fiți siguri că aveți un IP și un *subnet mask* similar cu cel existent pe dispozitivul *embedded* – să aparțină aceleiași clase. În cazul nostru, valorile recomandate sunt IP = **192.168.3.100** și *subnet mask* = **255.255.255.0**.

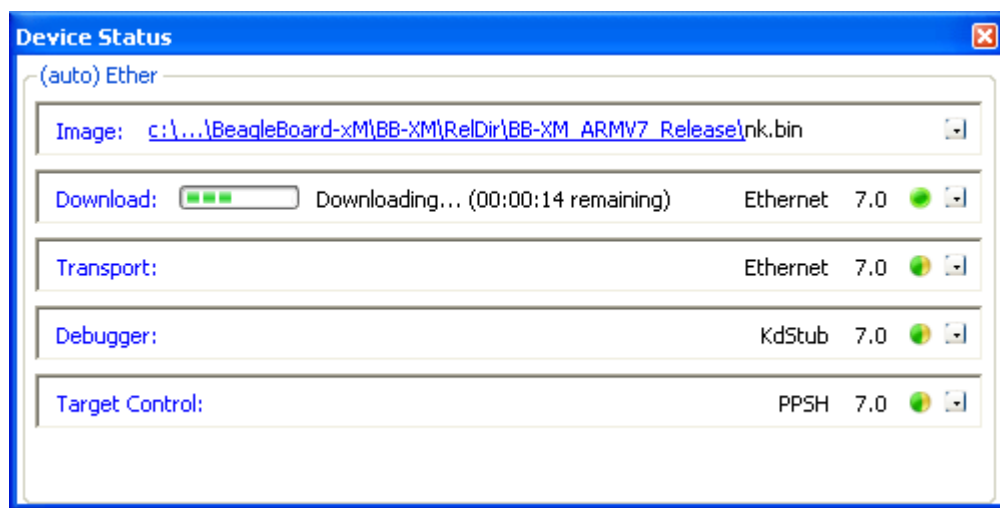
Atenție doar ca aceste setări să le aplicați plăcii virtuale “Windows CE USBDBG RNDIS” și nu altui dispozitiv, vezi **Figura 1.11**.



**Figura 1.11.** Plăcile existente în “Network Connection”

17. Dacă în fereastra terminal (ce prezintă toate mesajele generate prin intermediul liniei seriale în cadrul comunicației cu sistemul *embedded*) ați obținut mesajul “**Rndis:: initialization: Success**”, atunci în fereastra prezentată în **Figura 1.9** va apare cel puțin un dispozitiv activ (doar unul în cazul dvs., deoarece la calculatorul *host* este conectat un singur sistem de dezvoltare), selectați-l pe acesta (el așteaptă imaginea SO) – pentru a putea realiza acest pas în consola serială de configurare trebuie să vedeți că sistemul *embedded* trimite pachete de tipul **BOOTME**. Prin pachetele de date BOOTME sistemul *embedded* informează mediul de dezvoltare ca a terminat de inițializat toate dispozitivele și este gata să primească imaginea SO.
18. Apăsați butonul **Apply**.
19. Selectați opțiunea **Attach to device** (din mediul **Visual Studio**) pentru descărcarea imaginii SO. Imediat se va deschide o fereastră similară cu cea din **Figura 1.12** și încărcarea imaginii SO va începe.

După finalizarea procesului de încărcare, interacționați cu interfața grafică a SO **Windows Embedded Compact** prin intermediul tastaturii, *mous*-ului și a monitorului.



**Figura 1.12.** Fereastra în care este prezentat derularea procesului de încărcare pe sistemul *embedded* a imaginii SO Windows Embedded Compact

## 1.7. Boot-area SO prin intermediul conexiunii ethernet

Pentru încărcarea imaginii prin intermediul conexiunii de rețea, vom utiliza mecanismul de *boot-area* a plăcii de dezvoltare care presupune existența *first-stage* și *second-stage bootloader* în memoria *flash* a sistemului *embedded*.

Procesul de încărcare a imaginii SO prin intermediul conexiunii ethernet este similar la sistemele **OMAP3530 EVM** și **eBox-3310**. În cele ce urmează se vor prezenta pașii pentru sistemul OMAP3530 EVM iar la sfârșit se vor face completările pentru eBox-3310.

Pașii necesari a fi urmați pentru încărcarea prin ethernet a imaginii SO sunt:

1. Conectați sistemul *embedded* la sistemul *host*. Aveți nevoie de 2 conexiuni fizice:
  - a. Un cablu de rețea de tip *crossover*. În mod normal într-o rețea de tip *ethernet* calculatoarele sau orice alt tip de dispozitiv trebuie să se conecteze la un *ruter* care îndeplinește 2 funcții: asignează o adresă de IP, unică în rețea, pentru fiecare dispozitiv ce se conectează și transferă datele de la un sistem către alt sistem – aflat în rețeaua internă, gestionată de *ruter*, sau către orice alt calculator accesibil, conectat în internet.  
Cablul *crossover* este utilizat în conectarea a două dispozitive (calculatoare, sisteme *embedded*, imprimante etc.) direct, fără a necesita prezența unui *ruter* sau a oricărui alt echipament de rețea. Prin conectarea specifică a unor fire (astfel încât ieșirea unui sistem să fie trimisă în intrarea celuilalt) cele două dispozitive pot comunica între ele.
  - b. Un cablu serial. Conexiunea se va realiza între primul port serial al sistemului *host* (Com 1 pe cea mai mare parte a sistemelor sau pe acel port la care este legat cablul serial) și portul specific sistemului *embedded* – vezi **Tabelul 1.1**.
2. Porniți aplicația **Tera Term** (aveți un *shortcut* pe *desktop*-ul sistemului *host*), selectând portul “Com 1” drept port de comunicare cu sistemul *embedded*. Configurați parametrii comunicației seriale în conformitate cu datele prezentate în **Tabelul 1.1**. (**Setup** → **Serial port ...**), după configurare apăsați OK.
3. Alimentați sistemul *embedded*. *Boot-area* se va realiza din memoria flash. Pentru aceasta, anterior memoria flash a fost programată cu *first-stage bootloader* (**XLDR**) și *second-stage bootloader* (**EBOOT**) iar sistemul **OMAP3530 EVM** a fost configurat să *boot-eze* din memoria *flash* – vezi configurația specifică a bancului de comutatoare SW4 prezentată în **Figura 1.13**.

SW4 Switch Position							
1	2	3	4	5	6	7	8
OFF	ON	OFF	ON	OFF	OFF	OFF	OFF

**Figura 1.13.** Poziția comutatoarelor necesară pentru boot-area sistemului din memoria flash

4. În fereastra terminal apăsați SPACE, când vă este cerut, pentru a avea posibilitatea setării parametrilor transferului.
5. Alegeți pentru opțiunea “*boot device*”: LAN9115 MAC (în meniul “**Select Boot Device**”).
6. Alegeți “*debug device*”: LAN9115 MAC (în meniul “**Select KITL (Debug) Device**”).
7. În cadrul rubricii “Network Settings”:
  4. Selectați “**Disable DHCP**”
  5. Puneți adresa de IP static cu următoarele valori: 192.168.3.181
  6. Iar masca IP: 255.255.255.0

Toate aceste setări vor caracteriza sistemul *embedded* doar în timpul procesului de *boot*. Imediat ce EBOOT își termină rolul – atunci când imaginea SO a fost încărcată și executată –, noile setări ale plăcii de rețea, gestionată de această dată de SO **Windows Embedded Compact**, pot fi altele.

8. Dacă nu ați încărcat deja proiectul cu imaginea SO, realizați pașii 1-5 din cadrul **Subcapitolului 1.1** prezentați anterior.

9. În cadrul **Platform Builder**, deci în mediul Visual Studio, selectați **Target → Connectivity Options** și aveți grijă ca următoarele opțiuni să fie selectate (vezi **Figura 1.8.**):
  - a. Kernel Download puneți Ethernet
  - b. Kernel Transport puneți Ethernet
  - c. Kernel Debugger puneți KdSub
10. Apăsați pe butonul de *Settings* și lăsați deschisă fereastra care a apărut. Fereastra este similară cu cea din **Figura 1.9** dar fără a prezenta existența unui sistem activ în câmpul “Active target devices”.
11. Asigurați-vă că în Network connections aveți un IP și un *subnet mask* similar cu cel existent pe dispozitivul embedded – să aparțină aceleiași clase. În cazul nostru, valorile recomandate sunt IP = 192.168.3.100 și subnet mask = 255.255.255.0.  
 Atenție doar ca aceste setări să le aplicați plăcii ce poartă numele “*Embedded*” și nu altui dispozitiv, vezi **Figura 1.11.**
12. În meniul de comunicație cu *bootloader*-ului EBOOT, din aplicația terminal, selectați “**Exit and Continue**” pentru începerea procesului de încărcare a imaginii SO.
13. În urma acestei operații sistemul *embedded* începe trimiterea pachetelor BOOTME prin care își anunță disponibilitatea de a primi imaginea SO.

```

Selection: 0
OALFlashStoreOpen: 2048 blocks, 64 sectors/block
OALFlashStoreOpen: 2048 bytes/sector, 14 reserved blocks
INFO: Boot device uses MAC 00:50:c2:7e:8d:d0
INFO: *** Device Name EVM3530-36304 ***
+EbootSendBootmeAndWaitForTftp
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
.....

```

**Figura 1.14.** Finalizarea corectă a procesului de configurare a sistemului *embedded* OMAP3530 și cererile acestuia către sistemul *host* în vederea primirii imaginii SO

14. În fereastra deschisă alegeți unul din dispozitivele care este conectat și care așteaptă imaginea SO – pentru aceasta în consola serială de configurare trebuie să vedeți că sistemul *embedded* trimite pachete de tipul BOOTME.
15. Apăsați butonul *Apply*.
16. Selectați opțiunea **Attach to device** (din mediul Visual Studio) pentru descărcarea imaginii SO. Imediat se va deschide o fereastră similară cu cea din **Figura 1.12** și încărcarea imaginii SO va începe.
17. După finalizarea procesului de încărcare, vizualizați interfața grafică a SO **Windows Embedded Compact** și interacționați cu ea. Pentru sistemele *embedded* care au ecrane dotate cu *touch screen*, realizați în mod corect etapa de calibrare a ecranului tactil (apăsați cu *stylus* fix în mijlocul cruciulițelor ce vor apare), altfel în momentul în care veți interacționa cu tastatura virtuală (care este de mici dimensiuni), pentru acționarea unei taste, va trebui să apăsați doar în spațiul destinat tastei dorite și nu în spațiul asociat altei taste.

Procesul de *boot*-are a sistemelor *embedded* de tip **eBox** este puțin diferit față de cel al sistemelor ce au procesorul construit pe baza arhitecturii ARM (**OMAP3530 EVM** și **BeagleBoard-xM**). Deoarece procesorul din sistemele **eBox** este de tip x86 procesul de *boot*-are este similar cu al acestora până într-un anumit punct, după care urmează calea comună tuturor sistemelor *embedded* prezentată în cadrul **Subcapitolului 1.4.**



Astfel, la pornirea alimentării, programul care rezidă în ROM-ul sistemului, denumit BIOS, inițializează diferitele dispozitive *hardware* a sistemului *embedded* și pornește procesul de *boot*-are de pe memoria *flash* (de exemplu CF – CompactFlash), urmând pașii:

1. Se execută codul din sectorul MBR (*Master Boot Record*<sup>3</sup>);
2. Se execută fișierele IO.SYS și MSDOS.SYS;
3. Se lansează în execuție interpretorul de comenzi DOS (command.com – *shell*-ul SO DOS);
4. Se execută fișierul *batch* AUTOEXEC.BAT ce oferă utilizatorului un set de opțiuni de încărcare a imaginii SO Windows Embedded Compact:

- |   |
|---|
| <ol style="list-style-type: none"><li>1. Load nk.bin OS image from local storage</li><li>2. Load OS image from development station with DHCP service</li><li>3. Load OS image from host, the local Static IP is 192.168.3.100</li><li>4. Clean Boot (no commands)</li></ol> |
|---|

5. Funcție de opțiunea aleasă se inițializează în mod corespunzător starea unor variabile: NET\_IOBASE, NET\_IRQ, NET\_IP.
6. În final se lansează programul loadcepc.exe cu următoarele argumente:

`loadcepc /C:1 /L:800x600x16 /e:%NET_IOBASE%:%NET_IRQ%:%NET_IP% eboot.bin`

Loadcepc.exe este un *bootloader* poate lansa în execuție imaginea SO (fișierul **nk.bin**) stocat local (pe CF) sau poate lansa în execuție un alt *bootloader* **Eboot.bin** (denumit și *ethernet bootloader*) necesar stabilirii conexiunii cu calculatorul *host* unde imaginea SO a fost dezvoltată.

Selectându-se opțiunea 3, în programul terminal se va observa recepționarea pachetelor de cerere de tip BOOTME.

Pentru încărcarea imaginii SO de pe sistemul *host* se execută pașii de la numărul 14 la 17 prezentați anterior. Evident că anterior pașii 1 (conectarea cablurilor) și 2 (lansarea aplicației **Tera Term**) au fost realizați.

O altă diferență în *boot*-area sistemelor **eBox** față de cele ce au procesor construit pe arhitectură ARM este dată de faptul că programul terminal este utilizat doar în vizualizarea procesului de *boot*-are și nu și în configurarea lui.

---

<sup>3</sup> MBR-ul are o lungime de 512 octeți și este primul sector al mediului de stocare. Conține: tabela partițiilor (ce descrie caracteristicile partițiilor existente pe elementul de stocare), codul *bootstrap* (ce identifică partiția *boot*-abilă și îi dă controlul) și opțional poate include elemente de identificare a mediului de stocare (o semnătură a a acestuia) sau alte elemente de identificare (de exemplu, informații despre momentul când a fost partiționat).

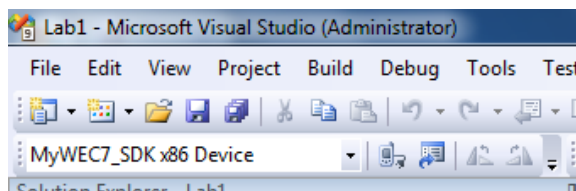
## 1.8. Rularea pe sistemul *embedded* a unei aplicații software

În continuare se vor prezenta pașii necesari a fi îndepliniți în vederea rulării unui program dezvoltat pe sistemul *host* în cadrul dispozitivului *embedded*. Deoarece scopul acestui subcapitol nu este acela de a prezenta modalitatea scrierii unui anumit program, în structura de directoare **D:\Embedded\Lab1** veți găsi un mic program demonstrativ dezvoltat în mediul Microsoft Visual Studio cu ajutorul claselor MFC. Acest program are o interfață grafică minimală (panou, buton de închidere, zonă titlu etc.), un buton și o zonă de afișare a mesajelor.

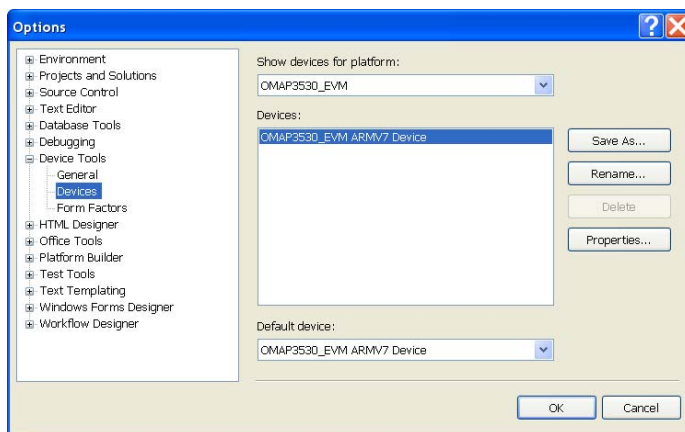
Pentru a rula acest program pe sistemul *embedded* executați pașii:

1. Lansați în execuție mediul **Microsoft Visual Studio** prin intermediul icoanei plaste pe *desktop*-ul calculatorului *host*.
2. Încărcați proiectul programului de test din directorul: **D:\Embedded\Lab1** prin următoarea secvență de comenzi: **File → Open → Project/Solution....** → Selectați din directorul anterior menționat fișierul **Lab1.sln** → **Open**
3. Recompilați și link-editați codul sursă al proiectului pentru a fi siguri că aveți fișierul executabil: **Build → Rebuild Solution** sau apăsați în mod direct combinația de taste **Ctrl+Alt+F7**.
4. Dacă ați primit mesajul „*Rebuild All: 1 succeeded, 0 failed*” (importantă este partea **1 succeeded, 0 failed**) treceți la punctul următor, în caz contrar corectați greșelile din program și generați fișierul executabil.
5. Următoarea acțiune este de conectarea a mediului Microsoft Visual Studio, mediu ce rulează pe dispozitivul *host*, cu dispozitivul *embedded*. Acest pas se poate realiza prin două modalități:
  - a. **Tools → Conect to device ...** → Selectarea dispozitivului prin alegerea corespunzătoare a opțiunilor **Platform și Devices → Connect**;
  - b. Prin selectarea dispozitivului *hardware* conectat la dispozitivul *host*, cu ajutorul câmpului


 , vezi **Figura 1.15**, și apăsarea butonului .

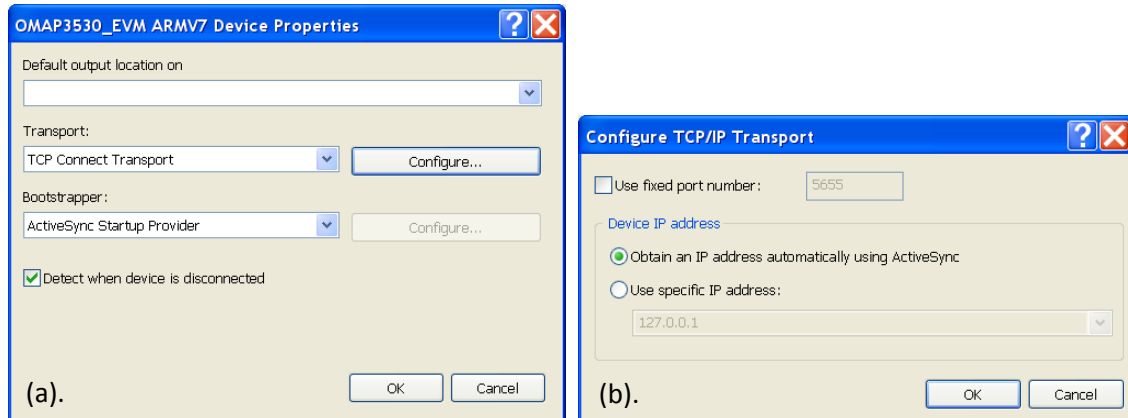


**Figura 1.15.** Opțiunile de conectare la dispozitivul *embedded*



**Figura 1.16.** Fereastra ce vă oferă opțiunile de configurare a conexiunii  
Microsoft Visual Studio → sistem *embedded*

6. Dar înainte de toate trebuie să configurăm această conexiune. Pentru aceasta executați secvența de comenzi:  
**Tools → Options... → maximizați Device Tools → Devices** sau, mai simplu, apăsați butonul  (*Device options*). Fereastra care se va deschide va fi similară cu cea din **Figura 1.16**.
7. Apăsați butonul **Properties ...**, fereastra care va apare va fi similară cu cea din **Figura 1.17(a)**, dacă se apasă butonul **Configure ...** va apare o ultimă fereastră de configurare similară celei din **Figura 1.17(b)**.



**Figura 1.17.** Ferestrele utilizate în configurarea parametrilor comunicării

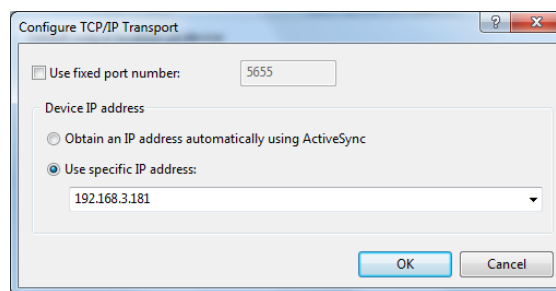
Programul dezvoltat de noi va fi transmis către sistemul embedded prin intermediul conexiunii *ethernet* (protocol TCP). Întotdeauna pentru realizarea unei conexiuni prin intermediul TCP dispozitivele implicate trebuie să aibă o adresă de IP proprie. Sistemul *embedded* este unul neconfigurat, din acest motiv, suplimentar, mai este nevoie de o conexiune USB pentru obținerea adresei de IP prin intermediul componentei **ActiveSync**.

**ActiveSync** este utilizat în sincronizarea informațiilor (fișiere, adrese, contacte, *email*-uri etc.) între un dispozitiv *embedded* și un calculator care rulează Windows XP. Pentru calculatoarele care au instalat SO **Windows Vista**, 7 sau 8 programul similar este **Microsoft Windows Mobile Device Center**. Deci **ActiveSync** sau **Microsoft Windows Mobile Device Center** trebuie instalate anterior executării mediului Microsoft Visual Studio.

Dacă sistemul *embedded* suportă mecanismul de obținere a adresei de IP prin intermediul componentei **ActiveSync** (are un portul necesar – are o comunicație pe USB activă – și toate componentele *software* incluse în imaginea SO **Windows Embedded Compact**) atunci setările prezentate în **Figura 1.17** sunt îndeajuns.

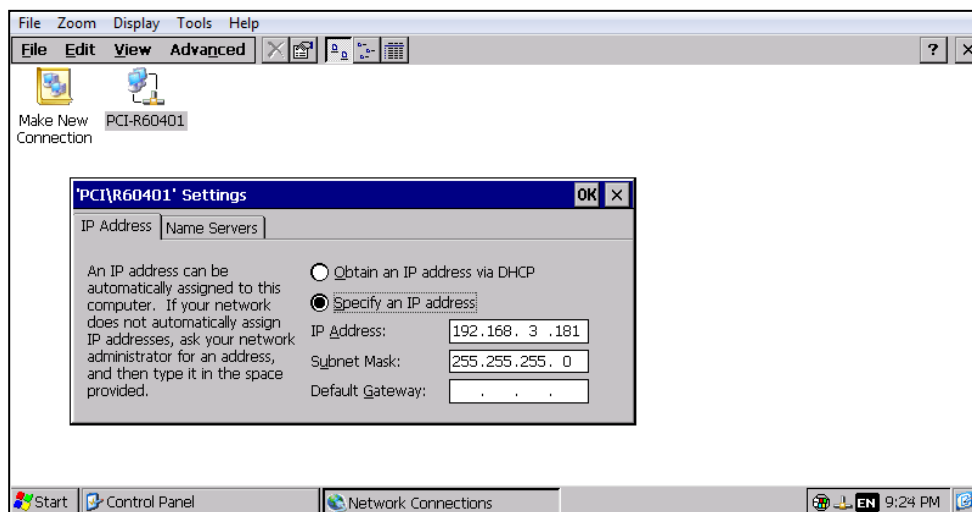
Pentru sistemele *embedded* care nu suportă mecanismul de obținere a adresei de IP prin intermediul componentei **ActiveSync** se execută, în plus, și următorii doi pași (ne referim aici la **eBox-3310**):

8. Se bifează **Use specific IP address** și se trece în câmpul de mai jos adresa IP a sistemului *embedded*, vezi **Figura 1.18**. Se apasă OK.

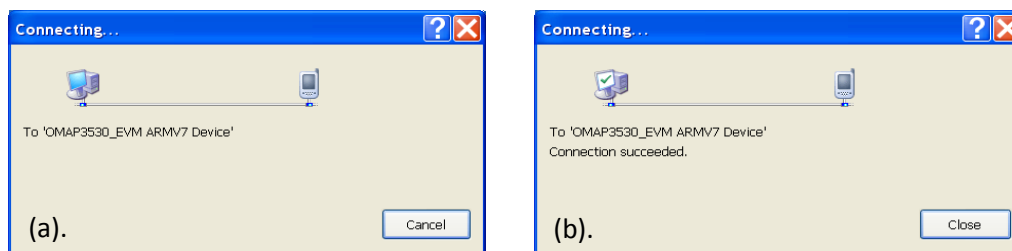


**Figura 1.18.** Specificarea adresei sistemului *embedded*


9. Se configurează adresa de IP și Mask-ul sistemului embedded: **Start → Settings → Control Panel → Network and Dial-up Connections**, selectându-se pentru **eBox-3310** placa PCI-R60401 și configurându-se conform **Figura 1.19**. Se apasă OK.



**Figura 1.19.** Configurarea sistemului embedded



**Figura 1.20.** Realizarea procesului de comunicare între mediu și sistemul *embedded*

10. Se conectează mediul de dezvoltare la sistemul *embedded* – se execută pasul 5. În fereastra care apare, în primul pas se realizează negocierea și inițializarea comunicării (care la sistemele **OMAP3530** și **BeagleBoard-xM** durează mai mult, deci așteptați fără a da alte comenzi mediului), iar în final se afișează realizarea conexiunii. Pentru sistemele *host* pe care rulează Windows XP și deci este implicit instalat **ActiveSync** trebuie să aveți iconița de pe *taskbar* de culoare verde înaintea începerii procesului de trimitere executare a aplicației dvs. Pe sistemul *embedded* – aceasta simbolizează realizarea conexiunii între **ActiveSync** și sistemul *embedded*.
11. Lansați în execuție programul prin una din metodele:
- Debug → Start Debugging;**
  - Apăsați F5 sau
  - Apăsați pe iconița .
12. Interacționați cu interfața grafică a programului ce rulează pe sistemul *embedded*.