Multidiszciplináris tudományok, 10. kötet (2020), 4. sz. pp. 132-141 **https://doi.org/10.35925/j.multi.2020.4.17**

A CODE COMPOSER STUDIO BEMUTATÁSA

Varga Attila Károly
egyetemi docens, Miskolci Egyetem, Automatizálási és Infokommunikációs Intézet
3515 Miskolc, Miskolc-Egyetemváros, e-mail: varga.attila@uni-miskolc.hu

Absztrakt

A jelprocesszoros alkalmazások megvalósításának első lépése az algoritmus funkcionális működésének tesztelése. Ha meggyőződtünk róla, hogy teljes mértékben betölti az előírt feladatot, következhet a második lépés, a megvalósítás. Az esetek túlnyomó többségében ez magas szintű, C/C++ nyelven történik. Ebben a fejlesztési fázisban optimalizálást még nem végzünk. Ha a kód teljesen hibátlanul működik, és megfelelő sebességgel fut, nincs szükség további finomításra, a fejlesztéssel készen vagyunk. A Code Composer Studio korszerű, felhasználóbarát integrált fejlesztői környezet. Egyszerűsíti és felgyorsítja a beágyazott jelfeldolgozó alkalmazások fejlesztési folyamatát. Eszközöket biztosít a programok konfigurálásához, felépítéséhez, nyomkövetéséhez és analizálásához, továbbá magában foglalja a programfejlesztő és optimalizáló eszközöket is. Jelen cikk keretein belül a Code Composer Studio ezen sokrétű funkcionalitását kívánom bemutatni.

Kulcsszavak: *jelprocesszor, jelfeldolgozás, C/C++, Code Composer Studio, programfejlesztés, programoptimalizálás*

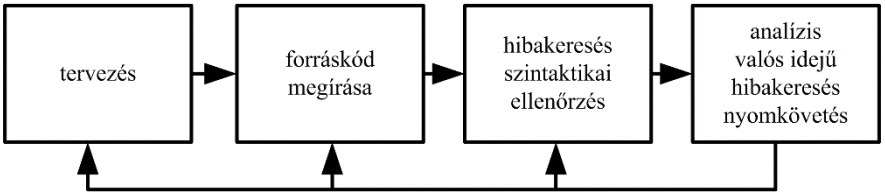
Abstract

The first step in implementing signal processor applications is to test the functional operation of the algorithm. Once we are convinced that it fully fulfills the required task, the second step, implementation, can follow. In the vast majority of cases, this is done in a high-level C/C++ language. We are not yet optimizing at this stage of development. If the code works completely flawlessly and runs at the right speed, no further refinement is needed, we're done with the development. Code Composer Studio is a state-of-the-art, user-friendly integrated development environment. Simplifies and accelerates the development process for embedded signal processing applications. It provides tools for configuring, building, tracking, and analyzing programs, and includes program development and optimization tools. With this publication, I want to introduce this versatile functionality of Code Composer Studio.

Keywords: *signal processor, signal processing, C/C++, Code Composer Studio, software development, program optimization*

1. A Code Composer Studio felépítése és funkciói

A Code Composer Studio (CCS) egy integrált fejlesztői környezet (IDE), amely támogatja a TI mikrokontroller és beágyazott processzorok portfólióját. A CCS egy olyan eszköztárat tartalmaz, amelyet a beágyazott alkalmazások fejlesztésére és hibakeresésére használnak. Támogatja az alkalmazásfejlesztés minden fázisát: a forráskód megírását, a hibakeresést és a valós idejű nyomkövetést. Kódgeneráló eszközöket tartalmaz és valós idejű analizáló képességekkel is rendelkezik. Az 1. ábrán a programfejlesztés lépései láthatók. [1]



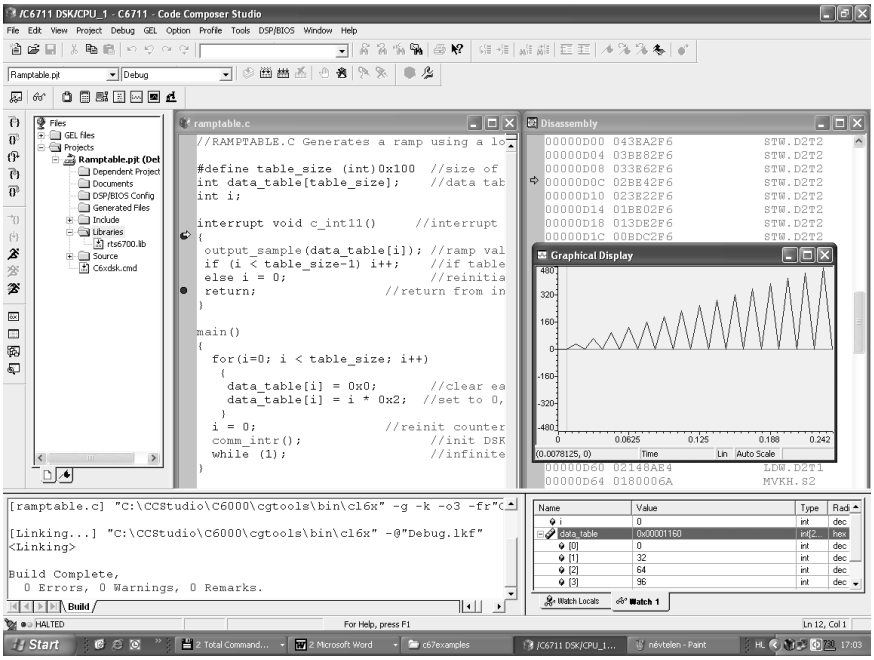
1. ábra. *A programfejlesztés folyamata*

A Code Composer Studio a következő komponensekből tevődik össze (ld. 2. ábra):

- TMS320C6000 kódgeneráló eszközök,
- Code Composer Studio IDE integrált fejlesztői környezet,
- DSP/BIOS plug-in-ek és API,
- RTDX plug-in, host interface, és API.

Az integrált szerkesztőprogram a következő tevékenységeket támogatja:

- kulcsszavak, kommentek, sztringek kiemelése más színnel,
- C blokkok megjelölése zárójellel vagy kapcsos zárójellel, azonos vagy következő zárójel vagy kapcsos zárójel megkeresése,
- behúzás csökkentése és növelése, tabulátorok beállítása,
- keresés és csere egy vagy akár több fájlban is, következő és előző kulcsszó keresése, gyors keresés,
- több művelet visszavonása és visszaállítása,
- szövegkörnyezet-érzékeny súgó.



2. ábra. *Alkalmazásfejlesztés Code Composer Studio alatt*

A CCS alatt az alkalmazás létrehozása projekt keretében történik. Az alkalmazásokat a projektfájlokkal építjük fel. Ezek a fájlok C vagy assembly forrásfájlok, objekt fájlok, könyvtárak, linker parancsfájlok és include fájlok. Menüből beállíthatók a kívánt fordítási és szerkesztési opciók. [2] [3] [4]

A CCS a következő hibaelhárítási funkciókat nyújtja: [5] [6]

- töréspontok beállítása számos léptetés opcióval,
- ablakok automatikus frissítése a töréspontoknál,
- változók figyelése,
- memória és regiszterek figyelése, módosítása,
- szubrutinhívási verem figyelése,
- Próbapont eszközökkel adatok és memória pillanatfelvételek (Snapshot) gyűjtése,
- jelek megjelenése grafikus formában,
- programfutási statisztikák készítése,
- Disassembly és C utasítások vizsgálata futás közben.

2. DSP/BIOS Plug-in-ok

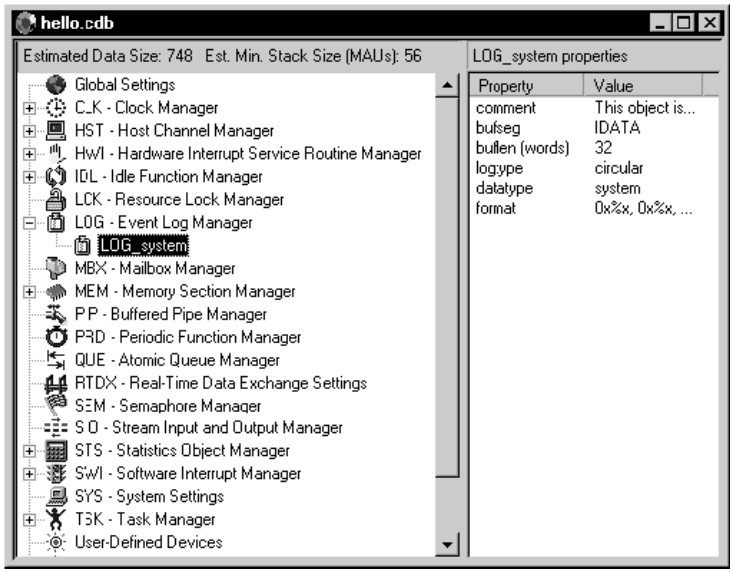
A programanalízis fázisában a hagyományos nyomkövetési funkciók nem használhatók olyan hibák kiszűrésére, amelyek az időzítések miatt jelentkezhetnek. A DSP/BIOS plug-in-ok biztosítanak ilyen valós idejű analízist. A DSP alkalmazások vizuális megjelenítésére, követésére és monitorozására használhatók, miközben a valós idejű teljesítményre csak minimális hatással vannak. A DSP/BIOS API a következő valós idejű analízis lehetőségeket biztosítja:

- *Programkövetés:* események megjelenítése és a program végrehajtása során a vezérlés folyamatának kijelzése.
- *Teljesítőképesség vizsgálatok:* az erőforrások használatát jellemző statisztikák nyomon követése, mint pl. a processzor leterheltsége vagy a folyamatok időzítése.
- *File folyamok kezelése:* célrezidens I/O objektumok host fájlokhoz való rendelését teszi lehetővé.

A DSP/BIOS egy prioritás-alapú ütemezővel is rendelkezik, amit az alkalmazások használhatnak. Ez az ütemező a funkciók periodikus végrehajtását és a prioritásos programszálak végrehajtását támogatja Létrehozhatók konfigurációs fájlok, amelyek a DSP/BIOS alkalmazói program interfész (API) által használt objektumokat definiálnak és leegyszerűsítik a memória térkép és a hardvermegszakítás kezelést. A konfigurációs fájlnak (ld. 3. ábra) kettős szerepe van:

- Globális futási paraméterek állíthatók be a segítségével.
- Vizuális szerkesztőként szolgál run-time objektumok létrehozásához és tulajdonságainak beállításához, amelyeket az alkalmazások DSP/BIOS API hívásai használnak. Ezek az objektumok tartalmazhatnak szoftveres megszakításokat, I/O pipe-okat és eseménynaplókat.

Eltérően a hagyományos nyomkövetéstől, amely a futó programtól elkülönül, a DSP/BIOS lehetőségeinek használatához a felhasználói programot össze kell szerkeszteni néhány DSP/BIOS API modullal. Ahhoz, hogy egy program DSP/BIOS modult használhasson, a DSP/BIOS objekt modult egy konfigurációs fájlban definiálni kell, external objektumként kell deklarálni és a DSP/BIOS API funkciót a forrásprogramban meg kell hívni. Mindegyik modul rendelkezik saját C header vagy assembly makro fájlokkal, amit *include*-al be kell illeszteni a programba. [6] [7] [8]



3. ábra. *Konfigurációk beállítása Code Composer Studio alatt*

Néhány fontosabb DSP/BIOS API modul:

C62. digitális jelprocesszor specifikus funkciók, megszakítások kezelésére,
CLK. időzítő kezelése,
HST. host input/output modulok kezelése,
HWI. hardver megszakítások kezelése,
IDL. üresjárati funkciók kezelése,
LCK. lock (zár) modul az osztott globális erőforrásokat kezeli,
MEM. memória szegmensek specifikációja,
RTDX. valós idejű adatcsere (Real-T/ime Data Exchange),
SWI. szoftvermegszakítás-modul,
SYS. általános célú rendszerfunkciók, mint a program felfüggesztése és üzenet kiírása.

3. Hardver emuláció és valós idejű adatcsere

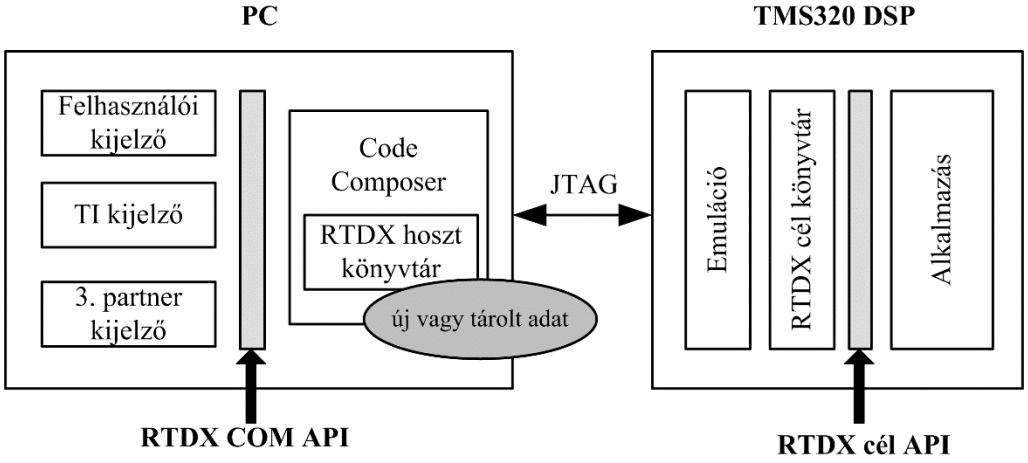
A Texas Instruments digitális jelprocesszor áramkörök tartalmazzák azokat az emulációhoz szükséges eszközöket, amely lehetővé teszi a CCS számára, hogy ellenőrizze a program végrehajtását és valós időben felügyelje a program aktivitását. A kommunikáció egy továbbfejlesztett JTAG csatlakozón keresztül történik. Egy host oldali emulátor interfész, (TI XDS510), alkotja a JTAG kapcsolat host oldalát. A chip-en lévő emulációs hardver pedig a következő funkciókat biztosítja:

- a jelprocesszor indítását, leállítását és resetelését,
- program vagy adat letöltését a jelprocesszorba,
- a jelprocesszor regisztereinek és memóriájának vizsgálatát,
- hardver utasítás vagy adatfüggő töréspontok biztosítását,
- különböző számlálást, beleértve a ciklus pontosságú profilingot is,
- valós idejű adatcserét (RTDX) a host és a jelprocesszor között.

Az RTDX valós idejű, folyamatos betekintést nyújt a processzor működésébe. Lehetővé teszi a rendszerfejlesztők számára adatok átvitelét a host számítógép és a DSP eszközök között anélkül, hogy le

kellene állítani az alkalmazást. Az adatok vizualizálhatók és analizálhatók a host gépen, ami jelentősen lerövidíti a fejlesztési időt. Az RTDX a célprocesszorban és a host gépen lévő komponensekből áll. Egy kis RDTX könyvtár fut a célprocesszoron. A felhasználó alkalmazói programja adatátvitelek céljából funkciókat hív e könyvtár API moduljából. A könyvtár a chipbe épített emulációs hardvert használja adatok mozgatására a host és a jelprocesszor JTAG interfészén keresztül. Az adatátvitel valós időben történik, mialatt a DSP alkalmazás fut.

A host gépen az RTDX könyvtár együttműködik a Code Composer Studio-val. A megjelenítő és analizáló eszközök az RTDX szolgáltatással egy könnyen kezelhető COM API modulon kommunikálhatnak, ha adatokat igényelnek a jelprocesszortól, vagy adatokat akarnak küldeni a jelprocesszor részére. Lehetőség van más szoftver megjelenítő csomagok használatára is mint a National Instruments' LabVIEW, vagy a Microsoft Excel. Az RTDX képes az adatokat valós időben rögzíteni és visszajátszani utólagos analízis céljából. [6] [7] [8]



4. ábra. *Code Composer Studio RTDX*

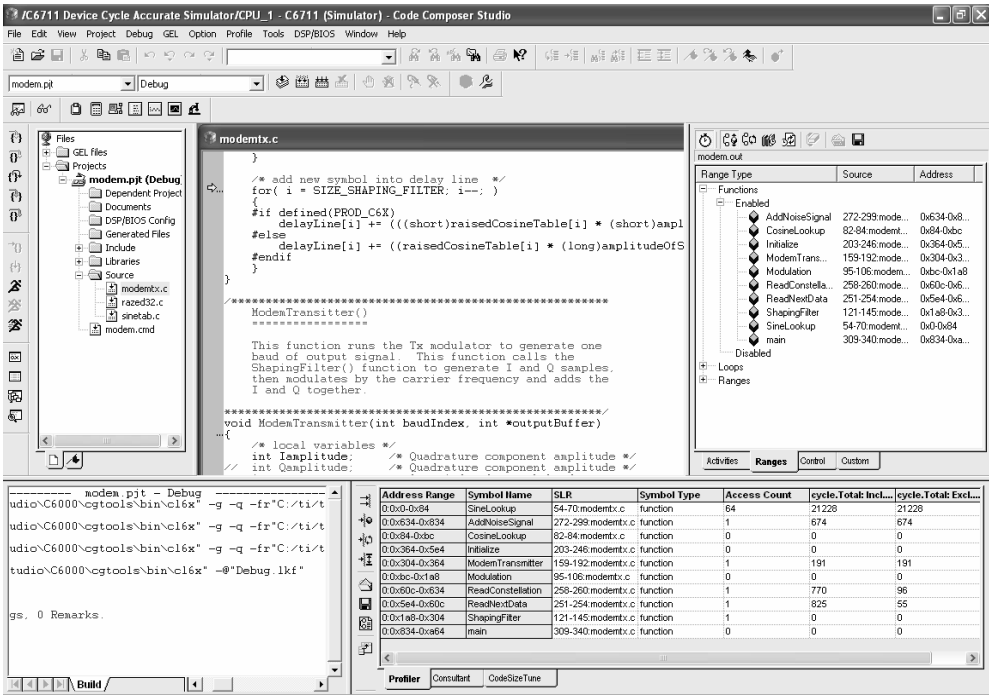
4. Code Composer Studio fájlok és változók

A Code Composer Studio az alábbi fájlokat és kiterjesztéseket használja:

- project.mak: a CCS által használt projektfájl,
- program.c: C program forrásfájlok,
- program.asm: assembly program forrásfájlok,
- filelname.h: C programok, DSP/BIOS API modulok header fájljai,
- filename.lib: könyvtárfájlok,
- project.cmd: szerkesztő program parancsfájlok,
- program.obj*:* forrásfájlokból lefordított objekt fájlok,
- program.out: teljesen lefordított és összeszerkesztett futtatható program,
- project.wks: munkaterület, amely a CCS által használt környezeti változók beállításairól tárol információt,
- program.cdb: A CCS-en belül létrehozott konfigurációs adatbázisfájl azokhoz az alkalmazásokhoz, amelyek a DSP/BIOS API modulokat használják. A konfigurációs fájl elmentésekor a következő fájlok generálódnak: programcfg.cmd szerkesztőprogram parancsfájl és programcfg.h62 header fájl. [7] [8]

5. A programfutás-elemzés (Profiling)

A programvégrehajtási idő vizsgálatához (Profiling) a CCS statisztikákat gyűjt a függvények végrehajtásáról. Miután a programot lefordítottuk, összeszerkesztettük és letöltöttük, be kell állítani a futáselemzőt a profile setup menüből.



5. ábra. *Programfutás-elemzés (profiling)*

Engedélyezni kell a profiling funkciót, ki kell választani a CPU ciklust, majd a Ranges menüpont segítségével kiválasztjuk az elemezni kívánt programrészt. Ezután a Profiler viewer ablakot megnyitjuk, amelyben a programfutásról gyűjtött információk jelennek meg. Ezután elindítjuk a programot. Kis idő múlva (egy perc elegendő), miután leállítjuk a programfutást, a Profiler viewer ablakban megjelennek a programfutás adatai, ahogyan az 5. ábrán is láthatjuk.

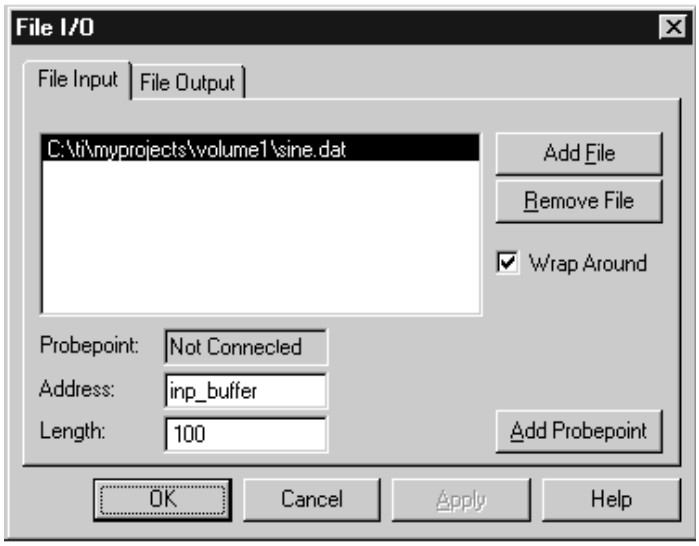
A futáselemzést elvégezhetjük program funkciókra, szubrutinokra, sor és cím tartományokra is.
A CCS átmenetileg leállítja a felhasználói programot a profile ponton. Emiatt az alkalmazás nem fut valós időben. Valós idejű monitorozás az RTDX használatával lehetséges.[7] [8] [9]

6. Algoritmusok és adatok tesztelése host fájl segítségével

Az algoritmusok működését tesztelhetjük a host gépen tárolt fájlok segítségével is. Használhatunk próbapontot, grafikai megjelenítést, animációt és GEL fájlokat.

A próbapont hasznos eszköz az algoritmusfejlesztési fázisban. A következő módokon használhatók:

- A host PC-n tárolt fájlt a célhardver input fájlként használhatja az algoritmus tesztelése céljából.
- Adatokat vihetünk át a célhardverből a host gépbe, analízis céljára, vagy grafikus ablak frissítésére.



6. ábra. *File IO dialógus ablak*

A próbapontok hasonlóan működnek a töréspontokhoz abból a szempontból, hogy mindkettő leállítja a jelprocesszort az akció idejére. Emellett különbségek is vannak a két módszer között. A próbapont leállítja az alkalmazói programot, végrehajt egy akciót, majd folytatja a program futását. A töréspont a végrehajtás idejére leállítja a CPU-t, amíg manuálisan nem folytatjuk a programot, és minden nyitva lévő ablakot felfrissít. A próbapont automatikusan engedélyezi az adat ki- vagy bemeneteket, a töréspont viszont nem. A próbapont használatához a forrásfájlban a kurzorral a dataIO() funkciót tartalmazó sorra kell lépni és a Toggle Probe Point gombbal ki kell jelölni a próbapontot. A File→File I/O. menüben File Inputot kell kiválasztani és meg kell adni a bemeneti fájl nevét. A fájl IO dialóg ablak, a 6. ábrán látható.

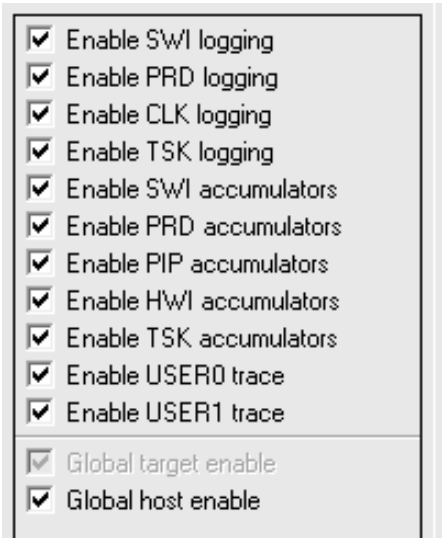
A Code Composer Studio sokoldalú grafikus megjelenítési lehetőséggel is rendelkezik, az adatok az idő és a frekvenciatartományban is megjeleníthetők. A grafikus megjelenítést a View menüből kezdeményezhetjük: View → Graph → Time /Frequency.

A Graph Property (grafikus tulajdonságok) Dialógusban meg kell adni a grafika nevét, kezdő címét, puffer hosszát, a kijelzett adat nagyságát, a skálázás módját (Auto vagy maximum Y érték). [8] [9]

7. A processzor teljesítőképességének vizsgálata

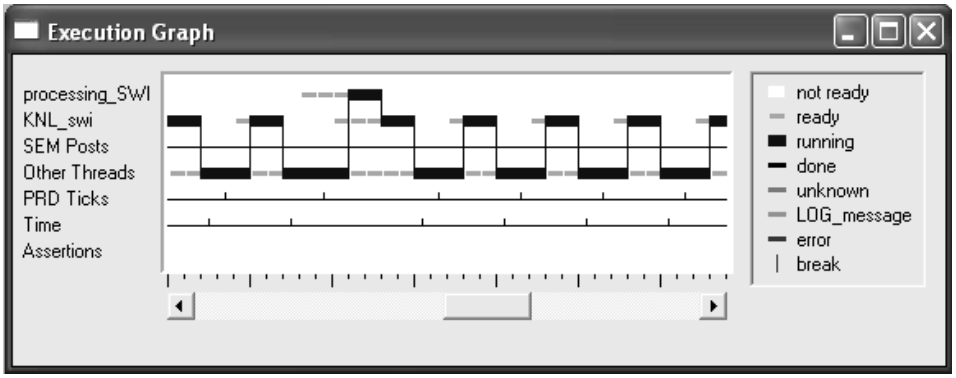
A processzor viselkedésének vizsgálatára több eszköz áll rendelkezésre. Ilyenek a DSP/BIOS modul lehetőségei, mint a futási grafikon (Execution graph), Valós idejű analízis kontrol panel (RTA Control Panel), Statisztika megjelenítő (Statistics View), CLK, SWI, STS, és TRC modulok.

Próbapontok használatával vagy bemeneti és kimeneti fájlok grafikus megjelenítésével az algoritmust teszteltük. A fejlesztés későbbi fázisában azt vizsgáljuk, hogy a programszálak futása lehetővé teszi-e a valós idejű (Real Time) működést. A vizsgálathoz el kell indítani a debuggert. Debug→Go Main parancs hatására a program a main funkció első programsorára ugrik. Ekkor a Tools→DSP/BIOS→RTA Control paranccsal kiválasztjuk az RTA vezérlő panelt.



7. ábra. *RTA Control Panel*

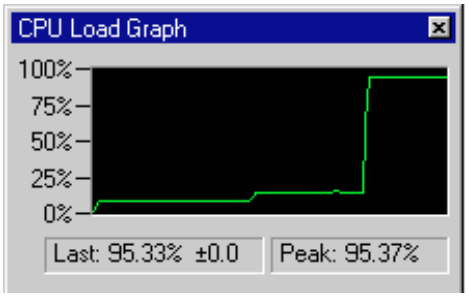
Az enable SWI logging, az enable CLK logging és global host enable engedélyezések után válasszuk a tools menüből a Tools_DSP/BIOS_Execution Graph pontot. A CCS ablak alján megjelenik a végrehajtási grafikon (execution graph). Egy ilyen grafikon látható a 8. ábrán.



8. ábra. *Végrehajtási grafikon*

A végrehajtási grafikonból kiderül, hogy a program képes-e valós idejű módban futni. Azonban ez attól is függ, mennyire van a processzor leterhelve. A vizsgálatok alatt a processzor terhelését szimulálhatjuk is különböző fokú terhelések beállításával, például a programban lévő valamelyik funkció ciklusszámának növelésével. Ez GEL fájl használatával valósítható meg egyszerűen.

A CPU Load grafika és a végrehajtási grafikon alapján megállapítható, hogy a program milyen CPU terhelés mellett képes valós időben futni.[7] [8]



9. ábra. *CPU terhelés grafikon*

8. Összefoglalás

Jelen publikációban bemutattam a Code Composer Studio kínálta lehetőségeket. Az Integrált Fejlesztő Környezete (IDE) felhasználói programok szerkesztését, módosítását, nyomkövetését teszi lehetővé. A szerkesztő szolgáltatása C és assembly forráskódok szerkesztését teszi lehetővé. A Code Composer Studio egy integrált fejlesztői környezet a Texas Instruments mikrovezérlőinek és digitális jelprocesszorainak programozásához, valamint hibakereséséhez. Tartalmaz egy optimalizáló C/C++ fordítót, forráskód-szerkesztőt, projektépítési környezetet, hibakeresőt, profilozót és még sok más funkciót. Az intuitív IDE egyetlen felhasználói felületet biztosít, amely végigvezeti a fejlesztőt az alkalmazásfejlesztési folyamat minden lépésén. A vezérlő, Simulink beágyazott programozóval való programozásához szükséges illesztőprogramokat is tartalmazza. A harmadik partner szoftverszállítók szintén fejlesztenek olyan programokat, amelyek kiegészítik a Code Composer Studio szolgáltatásait, ezek a különböző feladatú Third-Party Plug-in-ok nagy választékban állnak rendelkezésre. A megszokott eszközök és interfészek lehetővé teszik a felhasználók számára, hogy minden eddiginél gyorsabban kezdjék meg a munkát. A Code Composer Studio gyakorlatilag ötvözi az Eclipse szoftver keretrendszer előnyeit és a TI fejlett beágyazott hibakeresési képességeit, ami egyedülálló, funkciókban gazdag fejlesztői környezetet eredményez a beágyazott fejlesztők számára.[9] [10]

9. Köszönetnyilvánítás

A cikkben ismertetett kutatómunka az EFOP-3.6.1-16-2016-00011 jelű *Fiatalodó és Megújuló Egyetem – Innovatív Tudásváros – a Miskolci Egyetem intelligens szakosodást szolgáló intézményi fejlesztése* projekt részeként – a Széchenyi 2020 keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

[1] https://code-composer-studio.software.informer.com.
[2] Dahnoun, N.: *Digital Signalprocessing implementation using the TMS320C6000™ DSP Platform*, Pentice Hall 2000. ISBN 0201-61916-4.
[3] Chassing, R.: *DSP Applications using C and the TMS320C6x DSK*, John Wiley Sons, Inc.
[4] TMS320C6000 Assembly Language Tools User's Guide; Literature Number: SPRU186G, January 2000.

[5] TMS320C6000 CPU and Instruction Set Reference Guide; Literature Number: SPRU189E January 2000.
[6] TMS320C6000 Programmer's Guide; Literature Number: SPRU198D March 2000.
[7] TMS320C6000 Code Composer Studio Tutorial; Literature Number: SPRU301C February 2000.
[8] Code Composer Studio User's Guide; Literature Number: SPRU328B February 2000.
[9] TMS320C6000 Optimizing Compiler User's Guide; Literature Number: SPRU187 March 2000.
[10] https://www.ti.com/tool/CCSTUDIO.