

0. BEVEZETÉS

A “Jelfeldolgozó rendszerek tervezése” c. tantárgy célkitűzése, hogy a hallgatók megszerezzék azokat az ismereteket, amelyek segítségével képesek lesznek **megvalósítani** jelfeldolgozó processzoron (DSP-n) digitális jelfeldolgozási feladatokat.

A megvalósítás sokirányú ismeretet tételez fel:

- ismerni kell az elméleti alapokat,
- meg kell tudni tervezni az egyes rész rendszereket,
- ezeket tudni kell integrálni egy DSP programban és
- ismerni kell azt a fejlesztői környezetet, ahol a megvalósítás történik.

E tárgyat megelőzték a “Diszkrét idejű jelek és rendszerek analízise” és a “ Digitális jelfeldolgozás eszközei” c. tantárgyak, melyekben megismerkedtek - egyrészt a téma matematikai apparátusával, elméleti hátterével, - másrészt a TMS 320 5402 tip. jelfeldolgozó processzorral és annak programozásával.

A “ Jelfeldolgozó rendszerek tervezése” c. tantárgy a fenti folyamatból a tervezési lépést helyezi előtérbe. Alapvetően algoritmusokkal fogunk foglalkozni, amelyek segítségével a kiinduló specifikációs adatokból meghatározzuk azokat a *paramétereket*, melyek a megvalósításban szerepet játszanak. Ugyanakkor foglalkozni fogunk a másodlagos effektusokkal is, amelyek a megvalósítás *minőségét* befolyásolják. A példaként választott megvalósítandó rendszerek javarészt a híradástechnika területéről lesznek megválasztva. Véleményünk szerint az itt megszerzett tapasztalatok más területen is hasznosak lesznek.

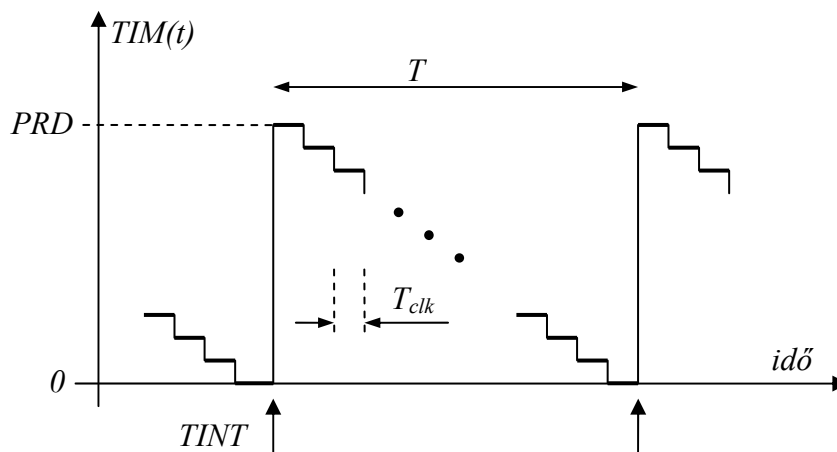
Ezzel a tárggyal párhuzamosan fut a “Laboratórium I.” c. tantárgy, melynek keretében az itt tanultak gyakorlatban történő kipróbálása lesz a feladat. A fentebb vázolt tevékenység-sor ezzel válik teljessé.

0.1. A valós idejű programok felépítése

Napjainkban a digitális jelfeldolgozási feladatokat döntő többségben digitális jelfeldolgozó processzorokkal (DSP) oldják meg. Ezen processzorok architektúrája illeszkedik legjobban a valós időben végrehajtandó feladatok gyors elvégzéséhez.

A processzor programja valójában egy végtelen ciklusban futó program, amit minden mintavételi periódusban végre kell hajtani. Az elvégzendő utasítások száma (a program hossza) nem lehet nagyobb mint mintavételi periódusidő (T) és processzor-órajel periódus idejének hányadosa (T_{clk}).

A diszkrét idejű jelfeldolgozás egyik alap feladata a mintavételi időpontok kijelölése. A diszkrét időpontok meghatározását a DSP-k a beépített TIMER segítségével oldják meg. A processzor timer regiszterének (TIM) tartalma minden gépi ciklusban automatikusan 1-el csökken, és amikor a regiszter tartalma zérussá válik, egy interrupt (TINT) generálódik. Ezzel párhuzamosan a periódus regiszter (PRD) tartalma áttöltődik a TIM regiszterbe. A diszkrét időpontok kijelölését ez az interrupt fogja végezni.



0.1.ábra A mintavételi idő beállítása a PRD regiszterrel

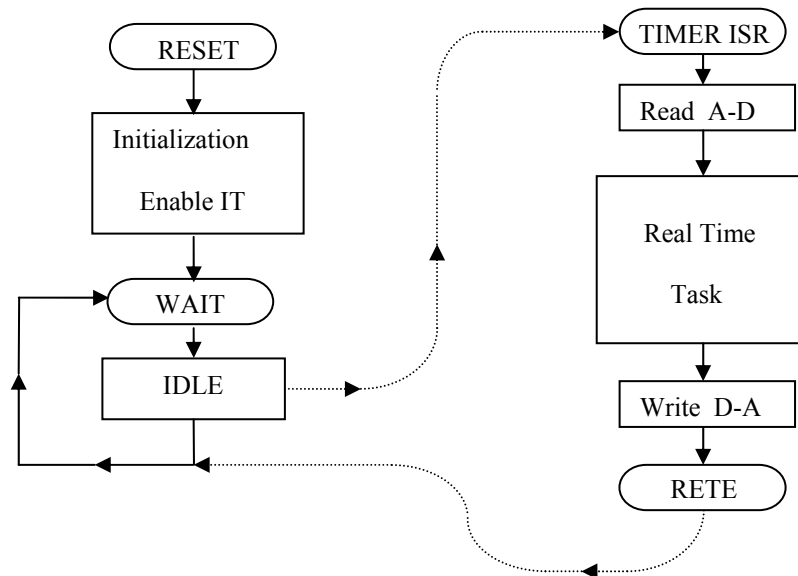
Adott f_s mintavételi frekvencia mellett a PRD regiszter tartalmát a következő formulával határozzuk meg:

$$PRD = \text{round}\left(\frac{f_{clk}}{f_s}\right) - 1 = \text{round}\left(\frac{T}{T_{clk}}\right) - 1$$

Nagyságrendi tájékozódás végett, ha $f_s = 10$ kHz és $f_{clk} = 100$ MHz a PRD 9999-re adódik. A valós idejű program végrehajtásához szükséges gépi ciklusok számának ebben az esetben tehát kisebbnek kell lennie tízezernél.

Tegyük fel, hogy olyan ügyesek vagyunk, hogy a megoldandó feladatot sokkal kevesebb utasítással megoldjuk, mint a fenti korlát. Döntenünk kell arról, hogy mit csináljon a processzor a fennmaradó időben? A szokásos döntés az, hogy a processzor álljon meg a feladat elvégzése után. A megállást az IDLE utasítással lehet kiváltani, melyből egy engedélyezett hardver interrupt megjelenése tudja tovább lendíteni a processzort. Az interrupt hatására az IDLE utasítás *utáni* program memória cím automatikusan elmentődik a *stack*-re és a vezérlés az interrupt vektor tábla adott interrupthoz tartozó címére adódik át. Ezen a címen egy ugró utasítás helyezhető el. A DSP programot úgy célszerű elkészíteni, hogy innen

az ugrás a *valós idejű feladatot* megvalósító program memória területre történjen. Ezt a kód részt hívjuk *Interrupt Service Routine*-nak (ISR). Az ISR utolsó utasítása a return (RETE) utasítás, melynek hatására a *stack*-ről a visszatérési cím betöltődik a *program counter*-be (PC), ahonnan a processzor folytatni fogja a futását. Mivel az összes valós idejű feladatot az ISR-ben már elvégeztük, itt nincs más teendők, mint az IDLE utasításra ugrani.



0.2. ábra A valós idejű program végrehajtásának szervezése

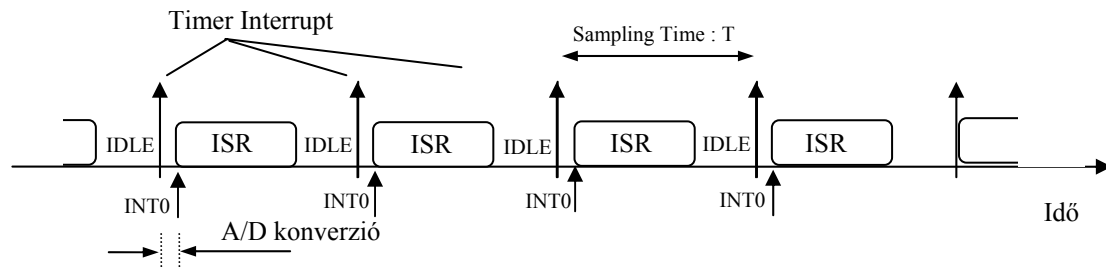
A fenti folyamatábrán folytonos vonalak jelzik a vezérlés átadásnak a programban utasítások formájában is megjelenő folyamatát, míg a szaggatott vonal szerinti vezérlés átadások a szervezésnek köszönhetően fognak bekövetkezni.

A fenti ISR-ben feltüntettük az analóg-digitál konverterről történő olvasás és a digitál-analóg konverterre való kiírás lépését. Mint tudjuk, a konverzióknak szigorúan periódikusan (*jitter*-mentesen) kell bekövetkeznie. A konverziók indítását (*trigger*-elését) szoftverből és hardverből is elvégezhetjük.

Az A-D és D-A konverterek konverzió start lábaira adott jelek két forrásból jöhetnek: vagy a programban elhelyezett utasítás hatására (tipikusan egy perifériára történő írás vagy olvasás, az írt vagy olvasott adat érdektelen) vagy egy processzor jel (TOUT, amit a TINT: timer interrupt vált ki) kezdődik el a konverzió.

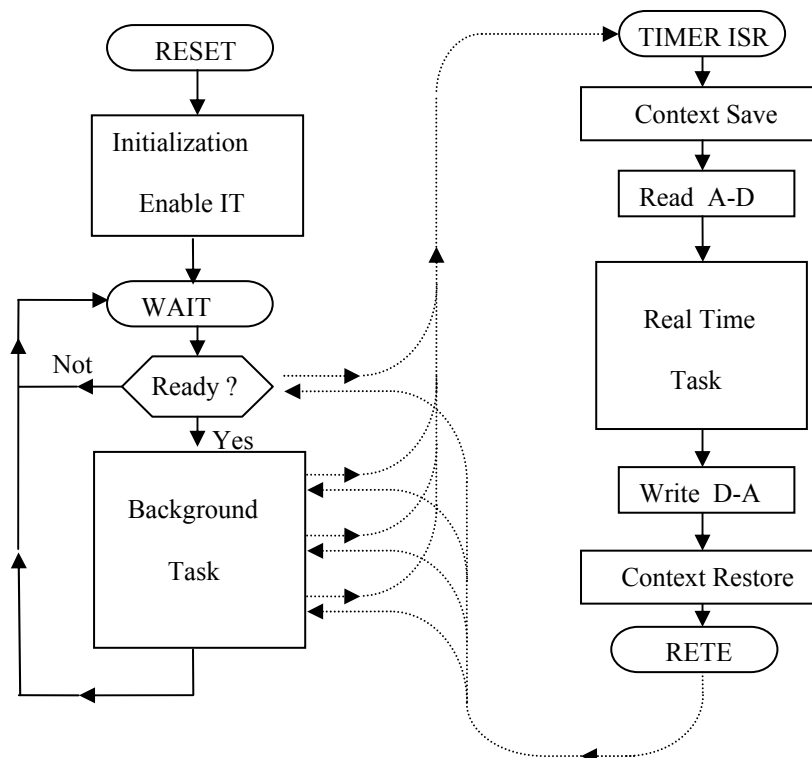
Az első esetben akkor járunk el helyesen, ha az ISR első utasításai közé helyezzük el az A-D-ről az olvasást, majd utána az új minta vételére szolgáló trigger jel kibocsátását. (A beolvasott adat természetesen az előző triggerelési időponthoz tartozik.) A D-A konverter indítását is itt célszerű elvégezni, mert az ISR elején lévő utasítások végrehajtásának ideje a TINT megjelenéséhez még szigorúan kötve vannak. Hibás az az eljárás, amikor a D-A-t akkor indítjuk, amikor a kimeneti minta kiszámításával elkészültünk. Gondoljunk arra, hogy az ISR-ben a vezérlés (bizonyos feltételektől, pld.a jeltől függően) más és más program utakat jelöl ki, melyek végrehajtási ideje különböző. Szoftver triggerelésnél tehát a helyes eljárás az, hogy az eredményt, ha elkészült, kiírjuk az átalakító átmeneti tárolójába (bufferébe), de az indítást a még fix időpontban végezzük el.

A másik eljárás az lehet, hogy a rendszert úgy konfiguráljuk, hogy a konverziók indítását a hardver végezze el (TOUT). Ekkor persze nem kezdődhet timer IT azzal, hogy beolvassuk az A-D-ről az adatot, mivel az még nem készült el. Viszont eljárhatunk úgy, hogy az A-D konverzió végét jelző INT0 interruptot használjuk fel az ISR végrehajtására. Ilyenkor csak az INT0-t engedélyezzük. (A TINT generálódik, csak nincs engedélyezve, de ettől még a TOUT jel létrejön.)



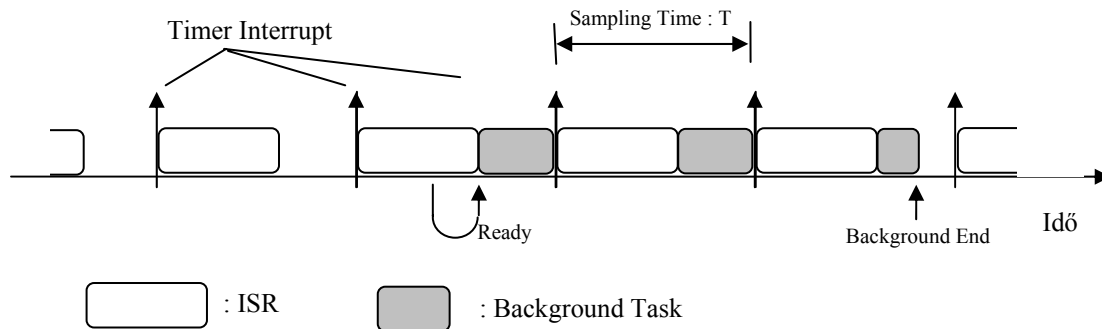
0.3. ábra A folyamatok idő diagramja hardver triggerelés esetén

Egy másik tipikus feladat lehet az, amikor a valós idejű feladat mellett van egy másik feladat (*background task*, *háttér feladat*) is, amelyet nem mintavételi idő periodicitással kell végrehajtani. Gondoljunk például egy FFT alapú spektrum analízátor megvalósításra, melynél első lépésben a jel elő-feldolgozása után össze kell gyűjteni adott számú mintát (real time task), majd ha ez megtörtént, el lehet kezdeni az FFT végrehajtását (background task). Ilyenkor nem célszerű leállítani az adatgyűjtést, mert mintákat veszítenénk el és az analízátor lelassulna.



0.4. ábra A feladat végrehajtásának folyamata

Tehát most nem állítjuk le a processzort, hanem egy várakozási hurokban köröztetjük a vezérlést (lásd.1.4.ábra). Ha a TINT bekövetkezik, a vezérlés az ISR-re adódik át, majd onnan ugyanebbe a hurokba kerül vissza. A *Ready* szemafor eredetileg úgy van beállítva, hogy a vezérlés a hurokban marad. Az N -ik ISR végrehajtás után (pld. összegyűlt N adat) az ISR-ben a *Ready flag*-et átállítjuk úgy, hogy az ISR-ből a hurokba visszatérve onnan ki tud lépni és el tudja kezdeni a background task-ot. Az elkövetkező néhány megszakítás a feladat végrehajtása során fog bekövetkezni. A vezérlés a visszatéréskor a background task-ba kerül mindaddig, míg az el nem készül. A végén a *Ready flag*-et visszaállítjuk a várakozási hurokban maradásnak megfelelő értékre.



0.5. ábra Az idő megosztása a valós és nem valós idejű feladatok között

Ez a példa az előző feladathoz képest annyiban más, hogy a processzor erőforrásait most két feladat között kell megosztani. Itt elsősorban a processzor regisztereinek tartalmának megőrzésére gondolunk. A background task végrehajtása egyszer csak megszakad és egy másik feladat kezdődik, amelyik ugyanazon regisztereket akarja használni. Annak érdekében, hogy ne következzen be adat veszteség, a valós idejű feladatot ezért kezdjük most úgy, hogy a regiszterek tartalmát elmentjük a *stack*-re (*context save*), majd a visszatérés előtt, visszaállítjuk(*context restore*) a regiszter tartalmakat. A háttér feladatba visszatérve a közbejött megszakítás így nem okoz hibát.

Nézzük meg ezek után azt a kérdést, hogy a mi megvalósítani kívánt feladatunk hogyan fog elindulni.

A jelfeldolgozási feladat megvalósítását úgy képzeljük el, hogy elkészítjük a DSP programot, majd azt betöltjük a DSP külső (external) program memóriájába. A processzort a betöltés alatt RESET állapotban tartjuk. Ezután a RESET állapotot feloldjuk, és a processzor elkezd futni.

A RESET alatt a processzor utasítás számlálója az interrupt vektor tábla reset címére (FF80h) mutat, így az első utasítást erről a címről hajtja végre. Ennek egy feltétel nélküli ugró utasításnak kell lennie, melynek hatására a tényleges program kezdetre adódik át a vezérlés. A tényleges program most a mi egyetlen kis feladatunkat jelenti. Tehát nem számíthatunk a BIOS és az erre épülő rendszer programok támogatására. Minden feladat ami a rendszer indításához kell, az ránk (az általunk írt programra) hárul.

A feladatok egy része az adott hardver környezetben állandó, így elég azt egyszer "belőni", így ez a program részlet a korábbi munkákból átvehető.(lásd minta programok státusz regiszter kezelése).

A TMS320 5402-es processzor viszonylag nagy ON-CHIP memóriával rendelkezik (4000h word), ami megfelelő konfigurálás után egyaránt lehet program és adat memória is. Ennek az elérhető sebesség szempontjából van jelentősége, mivel a külső memóriából való utasítás lehívás illetve adat beolvasás több gépi ciklust igényel, mint a belső memóriából

történő végrehajtáskor. A Harvard struktúra csak a belső memóriából történő futás esetén áll fenn.

Ezért általános elv lehet az, hogy az inicializálás (ami az indítás után csak egyszer fut) külső memóriából fusson, míg a további részek a belső memóriából kerüljenek végrehajtásra. Az inicializálásnak tehát legyen része a külső program memóriából a belső memóriába való átmásolás. Ugyan ezt vonatkozik a program működéséhez szükséges adatokra is.

Tipikusan az inicializálás utolsó fázisában húzzuk fel a timer-t, rendelkezünk az interrupt engedélyezésről, majd átadjuk a vezérlést az ON-CHIP memóriára, várva az első interrupt megjelenését. (Lásd minta program.)

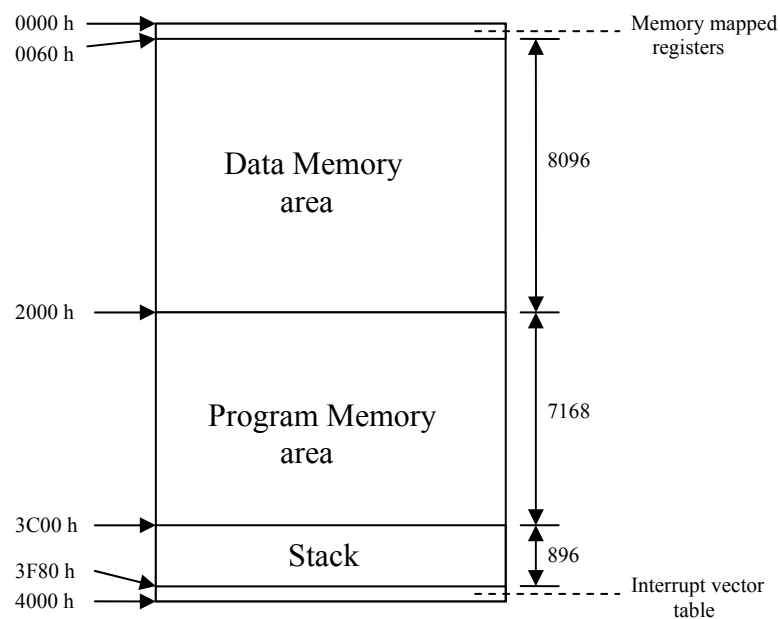
0.2. Néhány javasolt konvenció

Minden kezdet nehéz. Ezen próbálunk meg segíteni azzal, hogy néhány döntést hozunk most, amit persze nem kötelező betartani (elképzeltető olyan feladat ahol nem is lehet), de amelyek a kezdet nehézségein átsegítenek.

Az első elv legyen az, hogy egyetlen bináris file-t hozunk létre, ami tartalmazza a programot és a kiinduló adatokat is. Mivel a külső program memória nagy (64 kWord) ez legyen egy ilyen méretű file, aminek nagy része persze üres, de ezzel elkerüljük a több részletből álló eset bonyodalmait. Ezt a file-t fogjuk a külső program memória 0000h címére betölteni.

Döntsünk ezek után a belső (ON-CHIP) memória felhasználásáról. Rendelkezésre áll 16 kWord (4000h) terület, melynek címtartománya a 0000h-3FFFh területre terjed ki. Ez mind az adat, mind a program memória használat esetére érvényes.

Ebből a területből legyen a 0000h-1FFFh terület az adatoké, a 2000h-3BFFh a valós idejű programé, a stack legyen a 3C00h-3F7Fh területen míg a belső interrupt vektor tábla a 3F80h-3FFFh memória tartományon helyezkedjen el.



0.6. ábra Az ON-CHIP memória felosztása

Mint tudjuk az 5402-es processzor adatmemóriájának első 96 pozícióján a memóriába ágyazott regiszterek helyezkednek el (0000h-0060h).

A program memória itt azt jelenti, hogy a végrehajtandó kód helyezkedik el ezen a területen. Miután az inicializálás alatt bemásoltuk a külső memóriából a programot a belsőbe, majd átadtuk a vezérlést erre a területre, ezt a területet a vezérlés (jó esetben) nem hagyja el. (rossz esetben a vezérlés olyan helyre fut ahol esetleg nincs is kód.)

Bizonyos utasítások az operandusai szintaxisuk szerint program memóriabeli konstansok. Ezeket a konstansokat azonban elhelyezhetjük az adat memóriának szánt területre is (nem bontva meg a program terület egységét), ha a státus regiszter megfelelően van beállítva. (Lásd minta program).

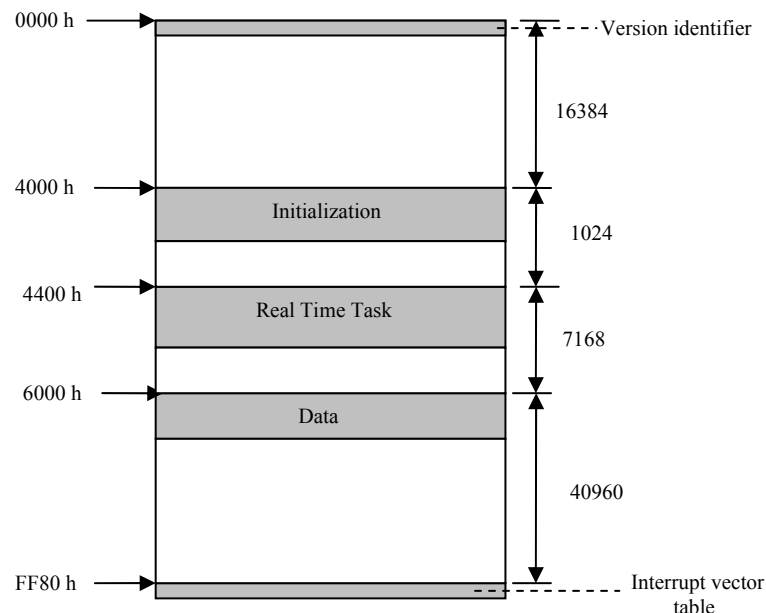
Mint fentebb említettük, egyetlen bináris file-t akarunk létrehozni, amelyben a különböző program részek mellett az adataink is be lesznek szerkesztve. Hogy ebbe az "x.ram" kiterjesztésű file-ba mi hova kerüljön az a *linker command file*-ban dől el.

Az interrupt vektor kezdetének az FF80h címen kell kötelezően lennie. Ha az inicializálás elején már úgy konfiguráljuk a processzort, hogy az ON-CHIP memóriát program memóriának is engedélyezzük, akkor a belső memória a külső program memória első 16 kWord (0000h-3FFFh) tartományát elfedi, így abba nem helyezhetünk el kódot. Így a programot (és az adatokat) csak e tartomány fölé szerkesszünk!

A 0000h címre a dátumot és a program verzió számát érdemes beszerkeszteni, hogy betöltéskor lássuk, hogy a betöltés megtörtént.

Az egyes program részeket érdemes külön szekciókra (section) bontani, így azok elhelyezkedéséről külön rendelkezhetünk, megkönnyítve ezzel a bemásolás feladatát. Hasonló okokból célszerű az adatokat is külön szekciónként kezelni, vagy külön-külön *y.asm*, *z.asm* forrás file-okból fordíttatni.

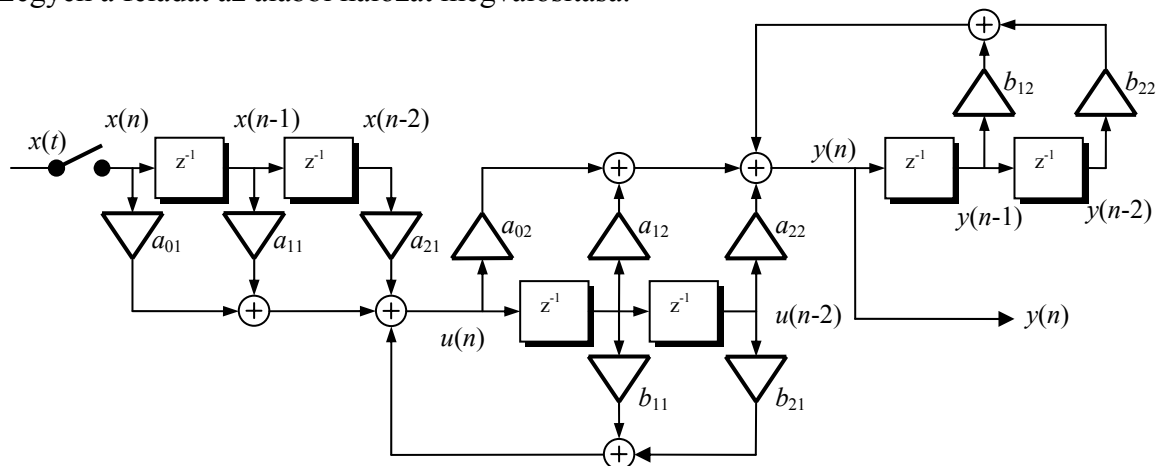
Az alábbi ábrán egy lehetséges memória térképet mutatunk



0.7. ábra Az külső program memória felosztása

0.3. Egy példa

Legyen a feladat az alábbi hálózat megvalósítása:



0.8. ábra A megvalósítandó hálózat

Az együtthatók értékei:

$$\begin{aligned} a_{01} &= 0.1286; & a_{11} &= 0.2153; & a_{21} &= 0.1286; & b_{11} &= 0.8349; & b_{21} &= -0.3077; \\ a_{02} &= 0.4360; & a_{12} &= 0.3218; & a_{22} &= 0.4360; & b_{12} &= 0.4949; & b_{22} &= -0.7586; \end{aligned}$$

A hálózatot leíró differencia egyenletek:

$$u(n) = a_{01}x(n) + a_{11}x(n-1) + a_{21}x(n-2) + b_{11}u(n-1) + b_{21}u(n-2)$$

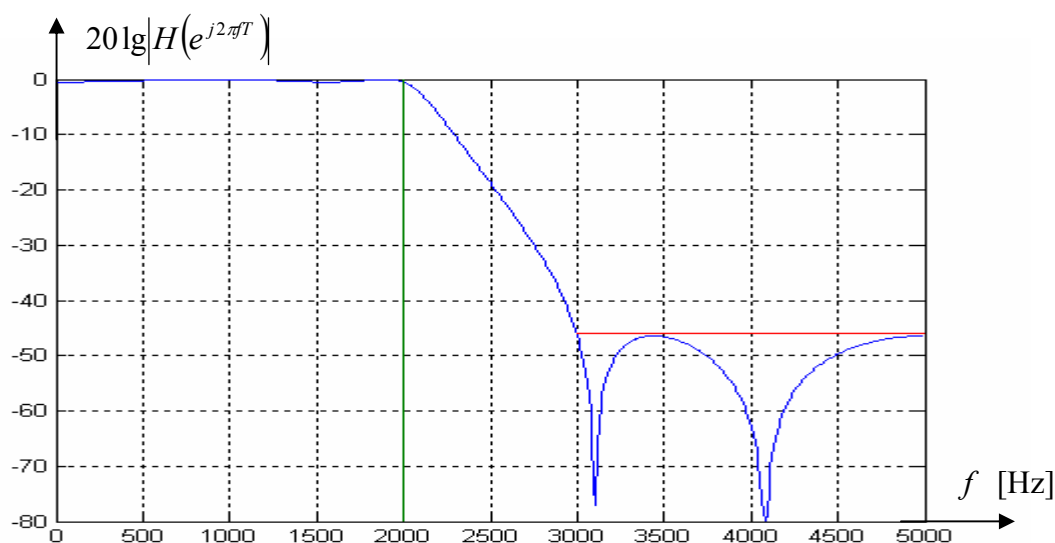
$$y(n) = a_{02}u(n) + a_{12}u(n-1) + a_{22}u(n-2) + b_{12}y(n-1) + b_{22}y(n-2)$$

A késleltetések sorrendben:

$$y(n-2) = y(n-1); \quad y(n-1) = y(n); \quad u(n-2) = u(n-1); \quad u(n-1) = u(n); \quad x(n-2) = x(n-1); \quad x(n-1) = x(n);$$

A szűrő specifikációja:

Mintavételi frekvencia:	10 kHz
Áteresztő tartomány:	0-2 kHz ingadozás ≤ 0.5 dB
Záró tartomány:	3-5 kHz elnyomás ≥ 46 dB



0.9. ábra A szűrő amplitúdó karakterisztikája

A fejlesztői környezet:

C:/.....

 /DSPLAB

 /TOOLS

 asm500.exe (Assembler)

 lnk500.exe (Linker)

 hex500.exe (Hex konv)

 t002RAM.exe (T00 to .ram)

 pcdsp5m.exe (monitor)

 /ELSO

 also.asm (DSP source)

 intvec.asm (int. vector source)

 filter.asm (filter coeff.)

 version.asm (verzió)

 utils.equ (deklarációk)

 linker.cmd (linker command file)

 alcb.bat (fordítást vezérlő batch file)

 clear.bat (fordítás után takarít)

A fordítás elindítása az ELSO könyvtárból : **alcb** also also

A letöltendő program: **also.ram** (**size:** 131016)

A letöltés a pcdsp5m.exe programmal.

Az **elso.asm** file tartalma:

```
*****
*      elso.asm      *
*      4th order IIR filter      *
*****

        .width      80
        .mmregs
        .global      STARTUP
        .global      TIMISR
        .ref          COEFF
        .include      utils.equ
*****
*      Initialization      *
*****
        .text
STARTUP      .set      $
LD           #0,      DP           ; set data page pointer to 0
SSBX        INTM      ; interrupts globally disabled
RSBX        OVM       ; set normal overflow mode
STM         #7492h,   SWWSR      ; Set wait states: 2 and 7(I/O)
STM         #3FE0h,   PMST      ; DARAM set to program space, too
STM         #3F7Fh,   SP       ; Init stack pointer
                                   ; IT vectors relocated to 3F7Fh
*****
*      Zeroing On-Chip DARAM      *
*****
STM         #60h,     AR1         ; Zeroing : 0060h - 3FFFh
RPTZ        A,        #3F9Fh     ; Length: 4000h-60h-1
STL         A,        *AR1+
*****
* Copy ".intvec" section to ON_CHIP      *
*****
STM         #3F80h,   AR1         ; Relocated address of IT Vector Table
RPT         #7Fh      ; Length= 80h
MVPD        #0FF80h,*AR1+        ; Copy Table to End of DARAM (3F80h)
*****
* Copy ".filter" section to ON_CHIP      *
*****
STM         #380h,    AR1         ; Relocated address of COEFF Table
RPT         #9        ; Length= 10
MVPD        #6000h,*AR1+        ; Copy filter Coeff.
*****
* Copy ".real" section to ON_CHIP      *
*****
STM         #2000h,   AR1         ; Onchip WAIT address
RPT         #(real_end - real_start - 1)
MVPD        #real_start,*AR1+
*****
*      INIT CONSTANTS ON PAGE 6      *
*****
LD          #6,       DP
*
ST          #300h,     MIT1        ;Address of VAR1
ST          #301h,     MIT2        ;Address of VAR2
ST          #1,        ONE
ST          #8000h,    H8000       ;Mask for D/A
*****
*      Config INT0 Interrupt      *
*****
CONFIG PORTR PA6,      KAMU        ; config ADC trigger: TOUT
        PORTR PA7,      KAMU      ; config DAC update : TOUT
*
LD          #0,        DP          ; Data Page = 0
ST          #9999,     PRD         ; Fclk=100 MHz
ST          #9999,     TIM         ; Fs = 10 kHz
ANDM        #0FFF0h,TCR          ; Clear TDDR bits
ORM         #0FFFFh,IFR          ; Clear pending ITs
STM         #01h,      IMR         ; Turn on INT0
RSBX        INTM      ; Enable IT
*
*
B           WAIT           ; Jump to OnChip
```

```

*****
* Real Time Task (Runing On_Chip) *
*****
        .sect      ".real"
        .label     real_start
WAIT    IDLE      1                ;Waiting for INTO
        B          WAIT            ;Return from TIMISR,
                                   ; goto WAIT
*****
* Timer Interrupt Service Routine *
*****
TIMISR   .set      $
*        CONTEXT_SAVE                ;Not used
        LD         TIM,    A
        STL        A,      TIM1    ;TIM1=Start TIM
*
        LD         #6,      DP
        PORTR      PA0,     XBE    ; read new sample from A/D
        STM        #COEFF, AR3    ;AR3-->A01
        STM        #310h,  AR4    ;AR4-->X0
        STM        #315h,  AR5    ;AR5-->U2
        LD         XBE,     A
        STL        A,      *AR4   ;X0=XBE
*
        LD         ONE,14,  A      ;A=Rounding bit
        MAC        *AR3+,*AR4+,A   ;A=A+A01*X0
        MAC        *AR3+,*AR4+,A   ;A=A+A11*X1
        MAC        *AR3+,*AR4+,A   ;A=A+A21*X2
        MAC        *AR3+,*AR5-,A   ;A=A+B21*U2
        MAC        *AR3+,*AR5-,A   ;A=A+B11*U1
        STH        A,1,  *AR5     ;U0=A
*
        STM        #318h,  AR4     ;AR4-->Y2
        LD         ONE,14,  A      ;A=Rounding bit
        MAC        *AR3+,*AR5+,A   ;A=A+A02*U0
        MAC        *AR3+,*AR5+,A   ;A=A+A12*U1
        MAC        *AR3+,*AR5, A   ;A=A+A22*U2
        MAC        *AR3+,*AR4-,A   ;A=A+B22*Y2
        MAC        *AR3+,*AR4-,A   ;A=A+B12*Y1
        STH        A,1,  *AR4     ;Y0=A
*
        STH        A,1,    YKI    ;Save Output
*
        STM        #317h,  AR4     ;AR4-->Y1
        RPT        #7
        DELAY      *AR4-          ;U0,Y0 are overwritten ;don't care!
*****
* Write Output *
*****
WR_OUT   MVDK      MIT1,   AR6    ;
        MVDK      MIT2,   AR7
        LD        *AR6,    A
        LD        *AR7,    B
        STL       A,      VAL1   ;Write Global Mem.
        STL       B,      VAL2   ;Write Global Mem.
*
        XOR       H8000,  A      ;Change Sign bit
        XOR       H8000,  B      ;Change Sign bit
        STL       A,      TEMP1
        STL       B,      TEMP2
        PORTW     TEMP1,  PA1     ; Write output Buffer
        PORTW     TEMP2,  PA2     ; Write output Buffer
*****
* Time consumption *
*****
        LD        #0,      DP
        LD        TIM1,    B
        LD        TIM,     A
        STL       A,      TIM2    ;TIM2= End TIM
        SUB       TIM2,    B
        STL       B,      TIM3    ;TIM3= Time consumption
*        CONTEXT_RESTORE        ;Not used
*
        RETE                      ;Return and Enable IT
        .label    real_end
        .end

```

Az **intvec.asm** file tartalma:

```
.sect ".intvec"
.ref STARTUP          ; initialization entry point
.ref TIMISR           ; timer ISR entry point
.align 0x80           ; must be aligned on page boundary

reset:  B STARTUP      ; address of first instruction
        NOP
        NOP
NMI:    RETE           ; enable interrupts and return
        NOP
        NOP
        NOP
sint17  .space 4*16    ; software interrupts
sint18  .space 4*16
sint19  .space 4*16
sint20  .space 4*16
sint21  .space 4*16
sint22  .space 4*16
sint23  .space 4*16
sint24  .space 4*16
sint25  .space 4*16
sint26  .space 4*16
sint27  .space 4*16
sint28  .space 4*16
sint29  .space 4*16
sint30  .space 4*16
int0:   BD    TIMISR   ; The only one active IT
        NOP          ; int0: ADC ready
        NOP          ; ADC trigger: TOUT signal
int1:   RETE
        NOP
        NOP
        NOP
int2:   RETE
        NOP
        NOP
        NOP
tint:   RETE
        NOP
        NOP
        NOP
rint0:  RETE
        NOP
        NOP
        NOP
xint0:  RETE
        NOP
        NOP
        NOP
        NOP
rint1:  RETE
        NOP
        NOP
        NOP
        NOP
xint1:  RETE
        NOP
        NOP
        NOP
        NOP
int3:   RETE
        NOP
        NOP
        NOP
        NOP
.end
```

A **filter.asm** file tartalma:

```
.sect ".filter"
.global COEFF
COEFF .set $
*
* #1 Second order section
.word 4214 ;A01 format: 16 TC Q15
.word 7056 ;A11 format: 16 TC Q15
.word 4214 ;A21 format: 16 TC Q15
.word -10083 ;B21 format: 16 TC Q15
.word 27359 ;B11 format: 16 TC Q15
*
* #2 Second order section
.word 14286 ;A02 format: 16 TC Q15
.word 10544 ;A12 format: 16 TC Q15
.word 14286 ;A22 format: 16 TC Q15
.word -24859 ;B22 format: 16 TC Q15
.word 16217 ;B12 format: 16 TC Q15
*
.end
```

A **version.asm** file tartalma:

```
.sect ".version"
*
.word 2006h ;year
.word 02h ;month
.word 05h ;day
.word 1 ;program number
*
.end
```

A **utils.equ** file tartalma:

```
***** Page 6 *****
XBE .set 00h
YKI .set 01h
*
X0 .set 10h
X1 .set 11h
X2 .set 12h
U0 .set 13h
U1 .set 14h
U2 .set 15h
Y0 .set 16h
Y1 .set 17h
Y2 .set 18h
*
TEMP1 .set 60h
TEMP2 .set 61h
TEMP3 .set 62h
TEMP4 .set 63h
*
MIT1 .set 70h ; Address of VAR1 for DA1
MIT2 .set 71h ; Address of VAR2 for DA2
VAL1 .set 72h
VAL2 .set 73h
*
ONE .set 7Dh
H8000 .set 7Eh
KAMU .set 7Fh
*
***** Page 0 *****
TIM1 .set 7Dh ; Start TIM
TIM2 .set 7Eh ; End TIM
TIM3 .set 7Fh ; Delta TIM
*
```

* hiánypotlo szimbolumok

```
PA0    .equ 0
PA1    .equ 1
PA2    .equ 2
PA3    .equ 3
PA4    .equ 4
PA5    .equ 5
PA6    .equ 6
PA7    .equ 7
```

* Makrok

CONTEXT_SAVE .macro

```
    PSHM AG      ;Save 8 guard bits of accA
    PSHM AH      ;Save upper 16 bits of accA
    PSHM AL      ;Save lower 16 bits of accA
    PSHM BG      ;Save upper 16 bits of accB
    PSHM BH      ;Save upper 16 bits of accB
    PSHM BL      ;Save lower 16 bits of accB
    PSHM TRN     ;Save transition register
    PSHM T       ;Save temporary register
    PSHM AR7     ;Save AR7
    PSHM AR6     ;Save AR6
    PSHM AR5     ;Save AR5
    PSHM AR4     ;Save AR4
    PSHM AR3     ;Save AR3
    PSHM AR2     ;Save AR2
    PSHM AR1     ;Save AR1
    PSHM AR0     ;Save AR0
    PSHM ST0     ;Save ST0
    PSHM ST1     ;Save ST1
    PSHM BK      ;Save circular size register
    PSHM IMR     ;Save interrupt mask register
    PSHM BRC     ;Save block repeat counter
    PSHM REA     ;Save block repeat end address
    PSHM RSA     ;Save block repeat start address
    PSHM PMST    ;Save PMST register
    .endm
```

CONTEXT_RESTORE .macro

```
    POPM PMST    ;Restore PMST register
    POPM RSA     ;Restore block repeat start address
    POPM REA     ;Restore block repeat end address
    POPM BRC     ;Restore block repeat counter
    POPM IMR     ;Restore interrupt mask register
    POPM BK      ;Restore circular size register
    POPM ST1     ;Restore ST1
    POPM ST0     ;Restore ST0
    POPM AR0     ;Restore AR0
    POPM AR1     ;Restore AR1
    POPM AR2     ;Restore AR2
    POPM AR3     ;Restore AR3
    POPM AR4     ;Restore AR4
    POPM AR5     ;Restore AR5
    POPM AR6     ;Restore AR6
    POPM AR7     ;Restore AR7
    POPM T       ;Restore temporary register
    POPM TRN     ;Restore transition register
    POPM BL      ;Restore lower 16 bits of accB
    POPM BH      ;Restore upper 16 bits of accB
    POPM BG      ;Restore 8 guard bits of accB
    POPM AL      ;Restore lower 16 bits of accA
    POPM AH      ;Restore upper 16 bits of accA
    POPM AG      ;Restore 8 guard bits of accA
    .endm
```

A linker.cmd file tartalma:

```
/******  
* LINKER COMMAND FILE FOR PCDSP5  
*  
* Usage: lnk500 <obj files...> -o <out file> -m <map file> lcf.cmd  
* cl500 <src files...> -z -o <out file> -m <map file> lcf.cmd  
*****/  
  
intvec.obj  
filter.obj  
version.obj  
  
MEMORY  
{  
PAGE 0: /* Program Space */  
    PROG0:      o=0000h l=4      /* Start of Program memory */  
    EXT_STUP:    o=4000h l=0400h /* External Program memory */  
    EXT_REAL:    o=4400h l=1C00h /* External Program memory */  
    INT_REAL:    o=2000h l=1C00h /* OnChip Prog.Mem. */  
    V_EXT:       o=0FF80h l=80h  /* External IT vector table */  
    V_INT:       o=3F80h l=80h  /* Internal IT Vector Table */  
    EXT_FILT:    o=6000h l=10    /* Ext.Prog.Mem.filter coeff*/  
    INT_FILT:    o=380h l=10    /* Int.Data space for coeff */  
  
PAGE 1: /* Data Space */  
    MMRS:        o=0000h l=0060h /* Memory-Mapped Registers */  
    SPAD:         o=0060h l=0020h /* Scratch-Pad RAM */  
    DARAM:        o=0080h l=1F80h /* DATA space for variables */  
    STACK:        o=3C00h l=0380h /* Init SP=3F7Fh */  
}  
SECTIONS  
{  
.text:      load = EXT_STUP PAGE 0  
.version:   load = PROG0 PAGE 0  
.real:      load = EXT_REAL PAGE 0, run = INT_REAL PAGE 0 /* main program */  
.intvec:    load = V_EXT PAGE 0, run = V_INT PAGE 0 /* IT vector table */  
.filter:    load = EXT_FILT PAGE 0, run = INT_FILT PAGE 0 /* Filter coeff */  
.data:      load = SPAD PAGE 1 /* temporal storage area */  
}
```

Az alcb.bat file tartalma:

```
@echo off  
if "%2" == "" goto :Help  
  
::Set TOOLS_DIR=.\tools\  
::Set ASM=asm500 -l  
::Set LINK=lnk500 -m %1.map  
::Set LCF=linker.cmd  
::Set INTVECT=%2  
  
:::vectors  
  
if exist error.log del error.log  
echo"BUILDER BATCH FILE STARTED:  
echo"  
  
if exist %1.obj del %1.obj  
if exist %1.lst del %1.lst  
echo"ASM500: assembling %1.asm, creating %1.obj & %1.lst files ...  
echo"  
..\tools\asm500 %1.asm -l >>error.log  
if errorlevel 1 goto err1:  
:main.obj done  
  
if exist intvec.obj goto vectdone  
echo"ASM500: assembling intvec.asm, creating intvec.obj & intvec.lst files ...  
echo"  
..\tools\asm500 intvec.asm -l >>error.log  
if errorlevel 1 goto err1:  
:vectdone
```

```
if exist version.obj goto versiondone
echo"ASM500: assembling version.asm, creating version.obj & version.lst files ..."
echo"
..\tools\asm500 version.asm -l >>error.log
if errorlevel 1 goto err1:
:versiondone

if exist filter.obj goto filterdone
echo"ASM500: assembling filter.asm, creating filter.obj & filter.lst files ..."
echo"
..\tools\asm500 filter.asm -l >>error.log
if errorlevel 1 goto err1:
:filterdone

echo"LNK500 - using linker.cmd:
echo"linking %1,obj, intvec.obj & words.obj, & twidle.obj, creating %2.out & %2.map files ...
echo"
..\tools\lnk500 linker.cmd %1.obj -o %2.out -m %2.map >>error.log
if errorlevel 1 goto err1:
:COFF out file done

echo"HEX500: converting %2.out, creating %2.hex file ..."
echo"
..\TOOLS\hex500 -fill 0 -t %2.out >>error.log
if errorlevel 1 goto err1:
:HEX conversion done

echo"T002RAM: converting %2.hex, creating:
..\TOOLS\t002ram %2.t00 >>error.log
:downloadable file created

echo"
echo"***** WOW, SUCCESS ! *****"
goto end:

:err1
echo"
echo"!!!!!!! SHIT, SOMETHING IS WRONG !!!!!!!"
echo"
echo"Look in the ERROR.LOG file!
goto end:

:Help
@echo off
echo"
echo"This batch builds DSP program for TMS320c5402 DSP
echo"
echo"Syntax: alcb INPUT OUTPUT
echo"
echo"where INPUT is a filename1 [.asm program file]
echo" OUTPUT is a filename2 [.ram downloadable file]
echo"
echo"Example: alcb munka1 m1
echo"
echo"Note: To have success, the following EXE files must exist in
echo" ..\TOOLS directory: ASM500, LNK500, HEX500, T002RAM
:end
```

A clear.bat file tartalma:

```
@echo off

if exist *.obj del *.obj
if exist *.lst del *.lst
if exist *.map del *.map
if exist *.out del *.out
if exist *.t00 del *.t00
if exist *.t10 del *.t10
if exist *.obj del *.obj
if exist *.lll del *.lll
if exist *.log del *.log
if exist *.bak del *.bak
if exist *.* del *.*
if exist *.swp del *.swp
```