MICROCONTROLERE – Lucrarea de laborator 3

Scopul lucrarii:

- descrierea modului de lucru cu mediul integrat de dezvoltare a programelor, VisualDSP++, specific microcontrolerelor de prelucrare a semnalelor (DSP) Analog Devices
- descrierea arhitecturii microcontrolerelui ADSP2181
- realizarea unui program, in limbaj C, cu programarea codecului AD1847
- prezentarea placii de evaluare cu microcontroler ADSP2181, EZ-Kit Lite
- modul de executie cu simulare (depanare) si pe placa reala

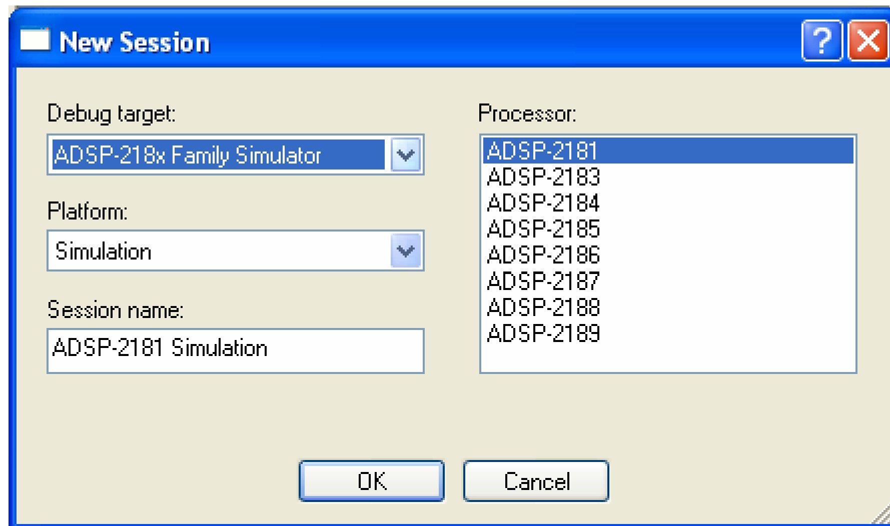
Desfasurarea lucrarii

1. Se va studia arhitectura microcontrolerului ADSP2181
2. Se va studia modul de operare al programului Visual DSP++
3. Se va studia arhitectura placii de evaluare Ez Kit Lite ADSP2181
4. Se va studia proiectul *TalkThrough* care realizeaza programarea ADSP2181 si a codecului AD1847. Se va modifica acest proiect astfel incit sa se realizeze urmatoarele aplicatii (a-d):
    a) inverseaza polaritatea semnalului de la intrare
    b) amplifica emnalul de la intrare cu un factor de 0.5 sau 2.
    c) daca semnalul de la intrare este semnal sinusoidal, realizeaza o redresare mono alternanta
    d) citeste semnalele de intrare de pe canalele *Right* si *Left* si genereaza la iesire pe canalul *Right* suma celor doua semnale si pe canalul *Left* diferenta celor doua intrari.
5. Se va studia si rula proiectul care genereaza 3 forme de unda (*Ez_lite_f_gen_c*).

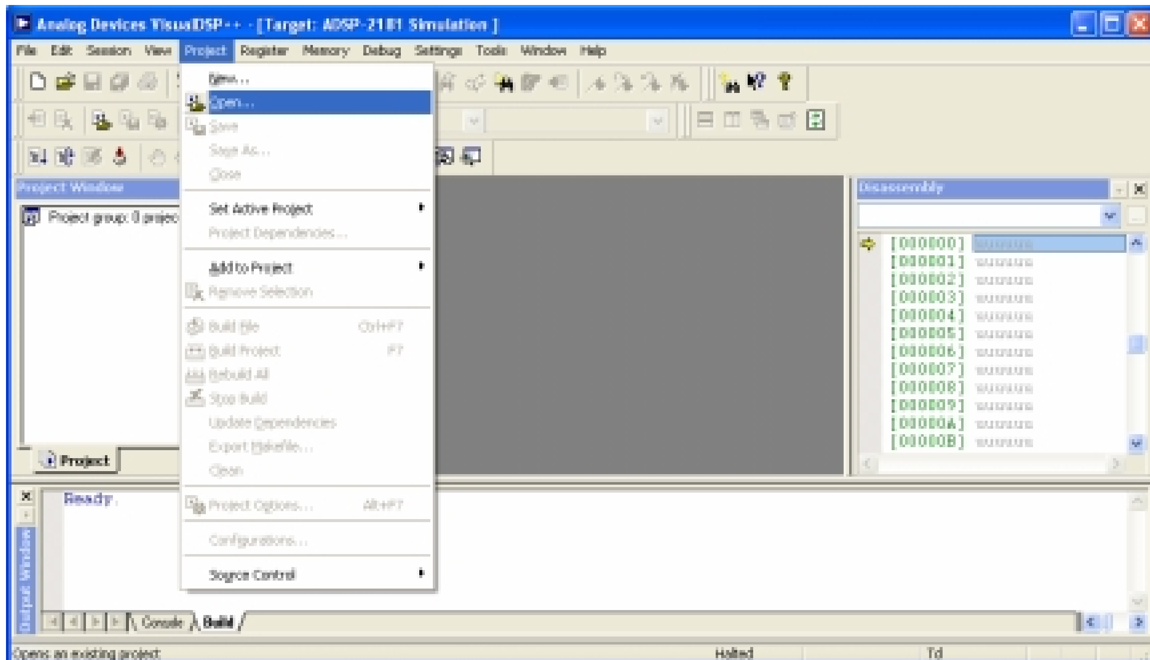
Toate programele vor fi simulate cu ajutorul VisualDSP++ si rulate in timp real pe placa EZ Kit Lite.

DESCRIEREA FUNCTIONARII PROGRAMULUI VISUAL DSP++

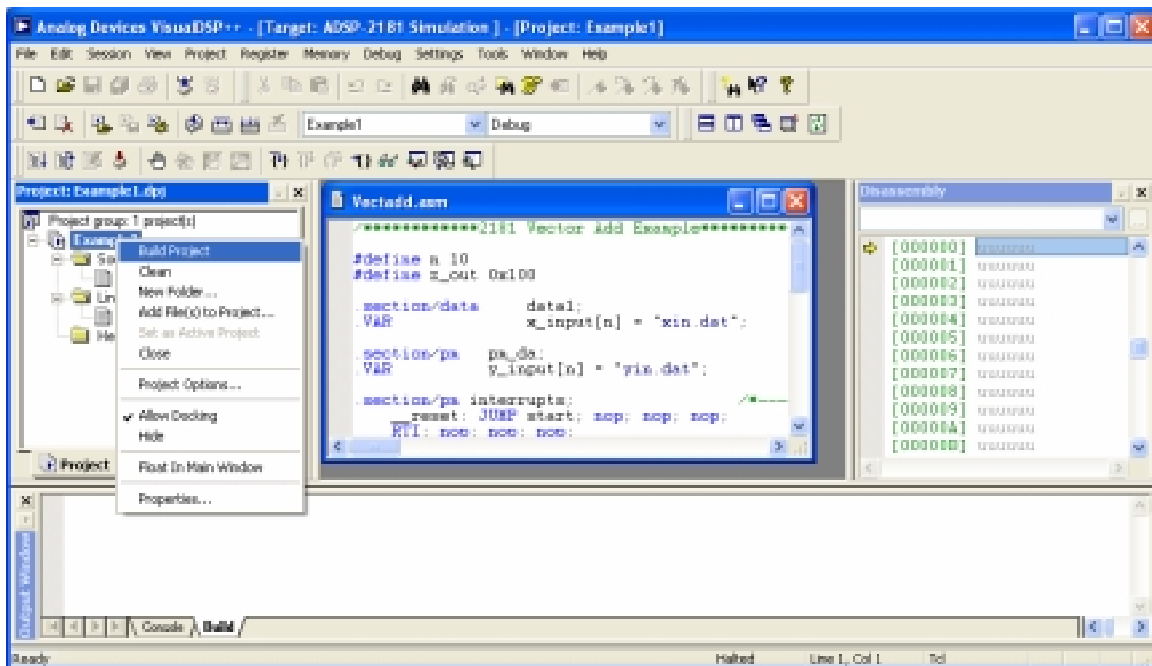
1. Se lanseaza in executie mediul integrat VisualDSP++ ( )
2. Se alege sesiunea de simulare cu processor ADSP2181



3. Se deschide un proiect cu comanda *Project->Open*



4. Compilarea proiectului se efectueaza astfel: in proiectul deschis se selecteaza numele acestuia, se da click dreapta si se allege comanda *Build* :



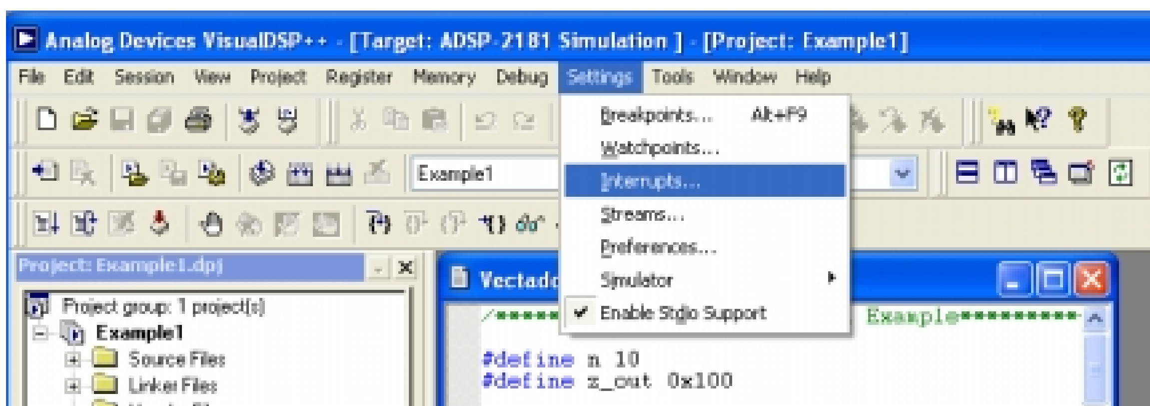
5. Daca apar erori la compilare se corecteaza programele sursa si se repeata pasul 4.

6. Daca nu au aparut erori atunci se poate realiza simularea executiei programului. In continuare se vor urmari urmatoarele aspecte:

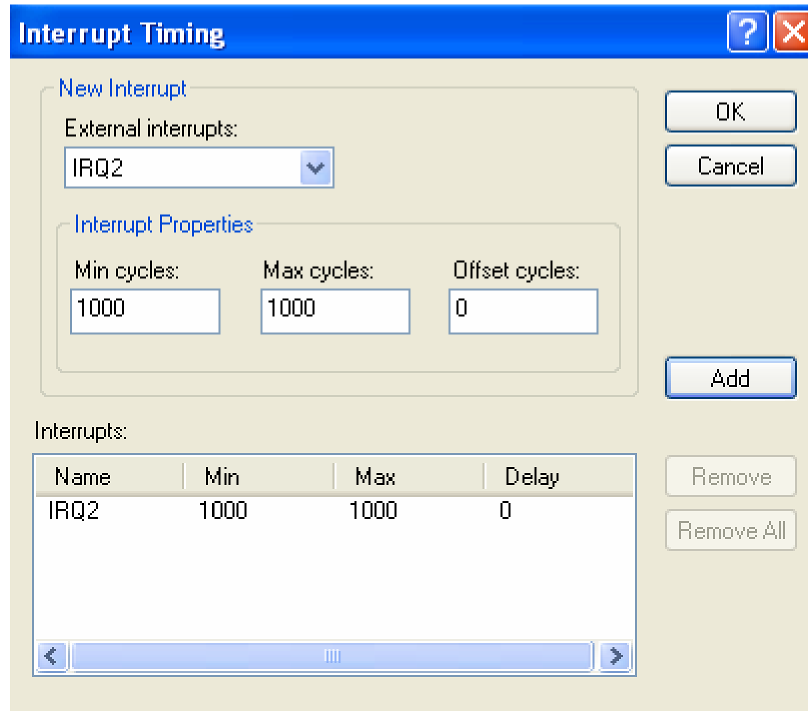
    - simularea intreruperilor
    - simularea porturilor de intrare si de iesire
    - puncte de intrerupere a programului (*breakpoint*)
    - executie pas cu pas sau executie pina la primul punct de intrerupere
    - vizualizarea rezultatelor
    - afisarea grafica a rezultatelor

Simularea intreruperilor

Se realizeaza prin comanda *Settings -> Interrupts*



In fereastra de dialog deschisa se alege nivelul de intrerupere dorit si modul in care se vor simula intreuperile:

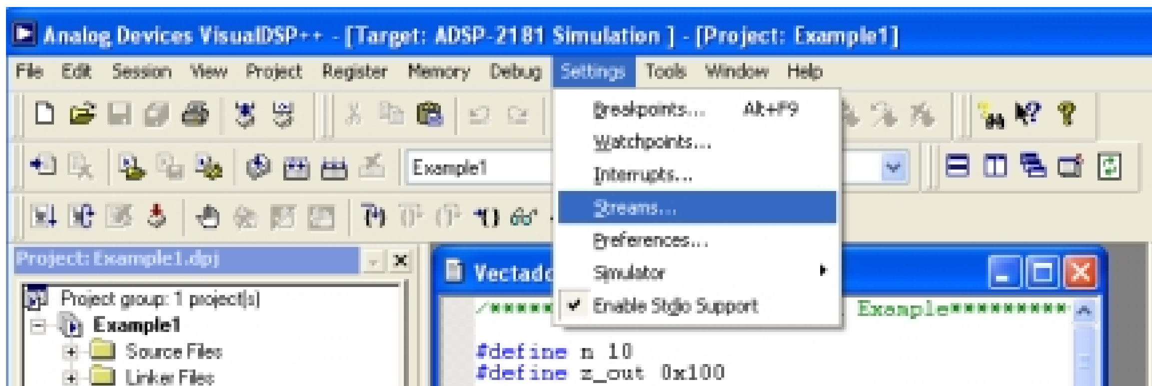


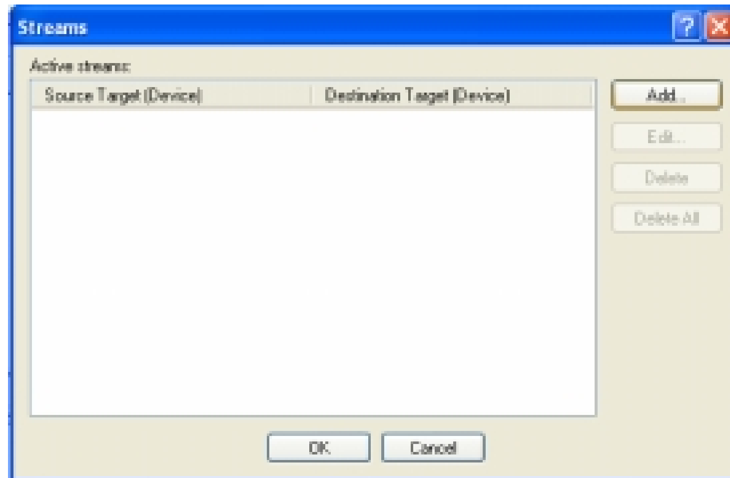
Intreruperile vor fi generate la intervale de timp (exprimate in ciclii de procesor) aleatoare in intervalul *(Min cycles, Max cycles*) cu intirziere initiala de *Offset* cicii. Daca *Min cycles= Max cycles* atunci intreruperile vor fi periodice cu perioada Min cycle.
Comanda se finalizeaza cu *Add* si *OK*.

Simularea porturilor de intrare si de iesire

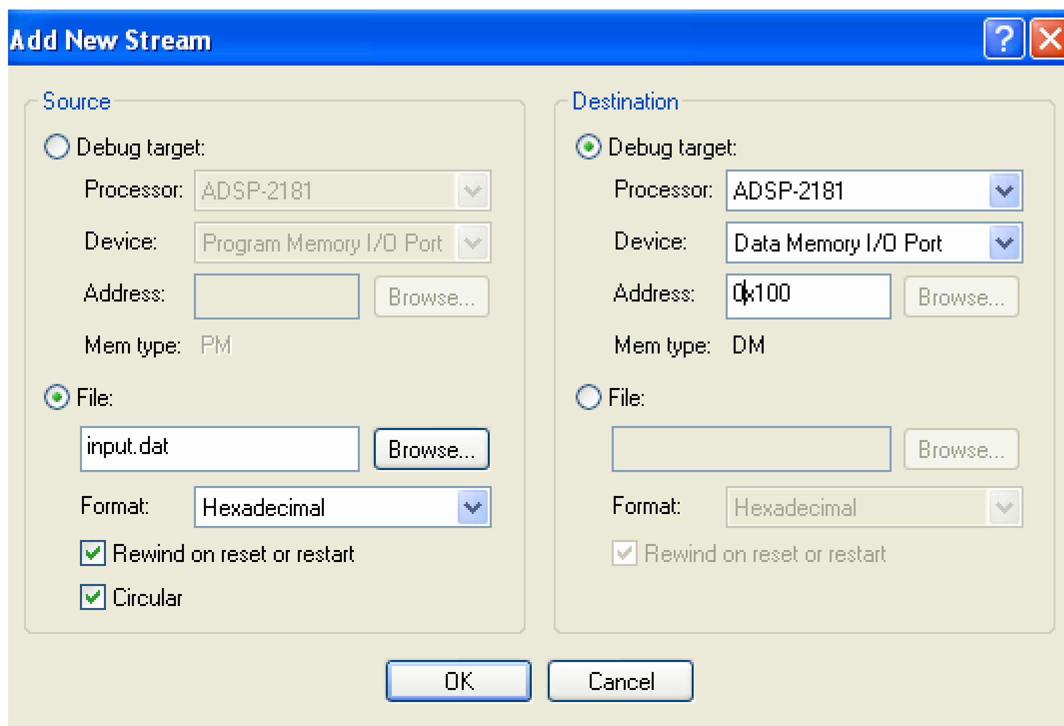
Se pot asocia fisiere unui port de intrare sau de iesire (de exemplu mapate in memoria de date) astfel:

Se da comanda *Settings -> Streams,* apoi *Add*



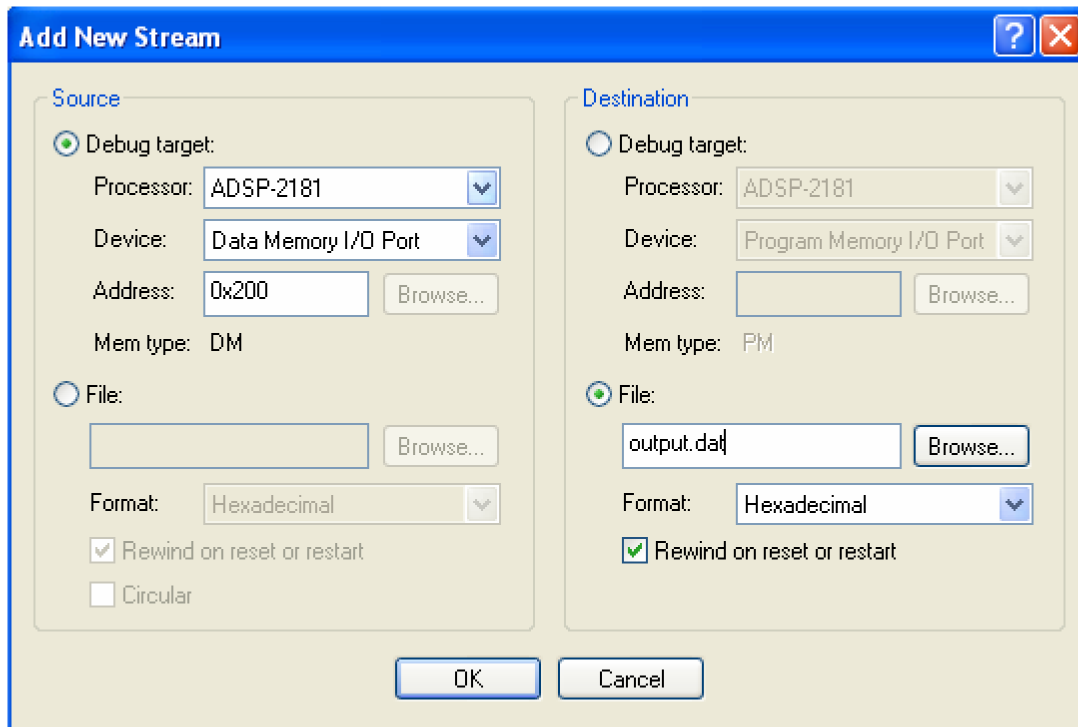


Pentru simularea unui port de intrare cu adresa in *memoria de date 0x0100* si asociat cu fisierul *input.dat* se inscriu urmatoarele informatii in fereastra de dialog deschisa cu comanda anterioara:



Comanda se finalizeaza cu *OK*

Pentru simularea unui port de iesire cu adresa in *memoria de date 0x0200* si asociat cu fisierul *output.dat* se inscriu urmatoarele informatii in fereastra de dialog:



Comanda se finalizeaza cu *OK*

Observatie: Pentru fiecare port de intrare iesire este recomnadabil asa se reia procedura de la inceput cu comanda *Settings->Streams*.

Puncte de intrerupere a programului (*breakpoint*)

Pe linia de program dorita de apasa tasta **F9** (comutare breakpoint)

Executie pas cu pas sau executie pina la primul punct de intrerupere

Se efectueaza cu tastele **F10**, **F11** respectiv **F5**.

Vizualizarea rezultatelor

Se dau comenzile *Memory* (pentru vizualizarea memoriei), *Register* (pentru vizualizarea registrelor din unitatile functionale ale ADSP) din bara de meniu principala. Cu *click dreapta* se deschide un sub-meniu care permite vizualizarea in diferite formate a informatiilor. Ferestrele deschise se pot aranja pe ecran daca se deselecteaza, in acest sub-meniu, optiunea *Allow docking*.

Pentru memorie

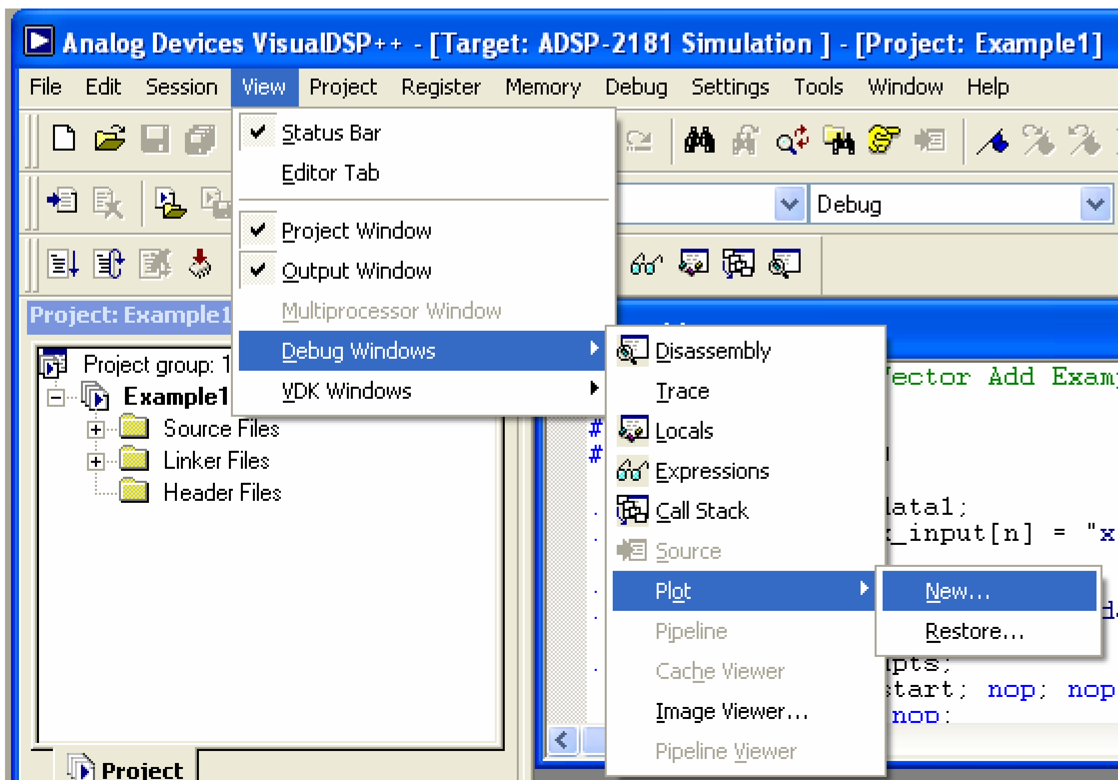
Go To... Ctrl+G
Dump...
Fill...
✔ Address Bar
View Source
✔ Allow Docking
Close
Float In Main Window

sau pentru registre

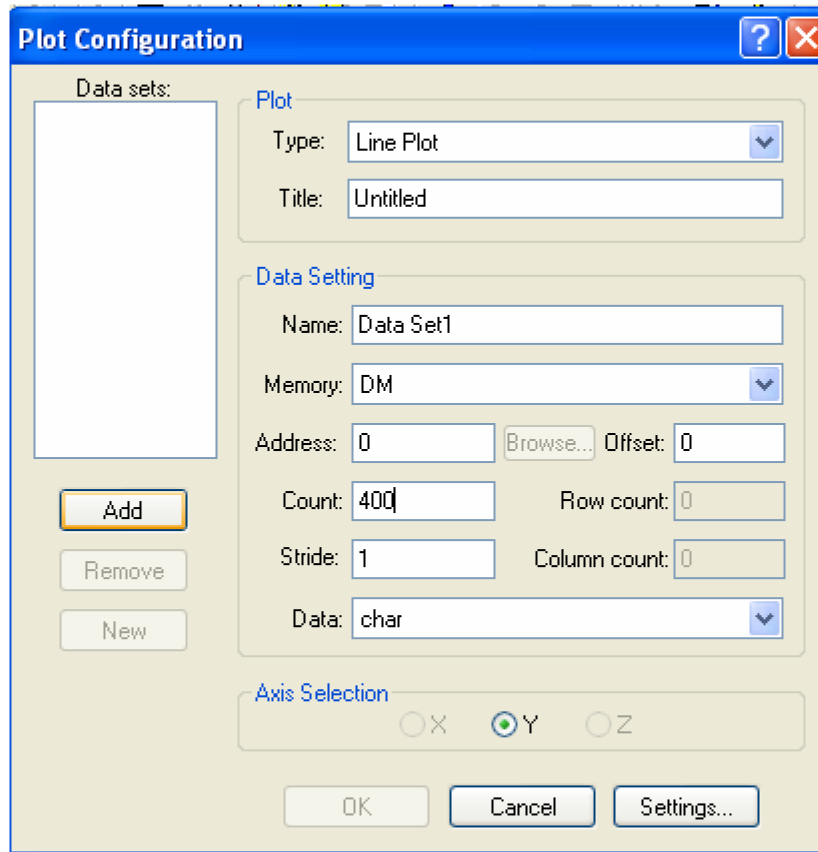
✔ Hexadecimal
Unsigned Integer
Signed Integer
Octal
Fractional
Character
Binary
✔ Allow Docking
Close
Float In Main Window

Afisarea grafica a rezultatelor

Programul Visual DSP++ permite afisarea grafica a datelor.
Se da comanda *View-> Debug Windows -> Plot -> New*:

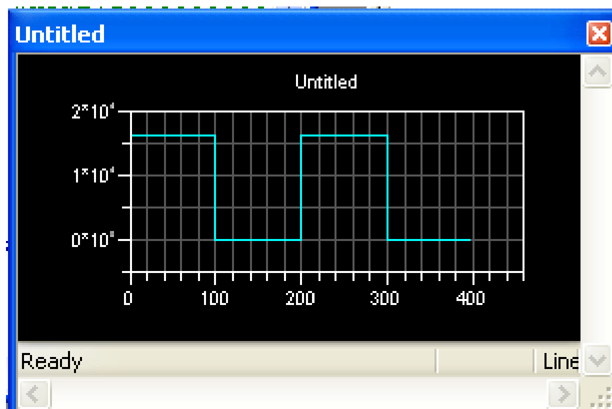


In fereastra de dialog deschisa se aleg adresa initiala si lungimea zonei de memorie care trebuie afisata grafic:



Se doreste afisarea zonei de memorie de date de la adresa *0x0000* si lungimea *400* de locatii. Comanda se finalizeaza cu *Add* si *OK*.

Rezultatul afisarii grafice este:
Cu *click dreapta* apare un sub-meniu care permite vizualizarea avansata a graficului.



Data Cursor
Reset Zoom

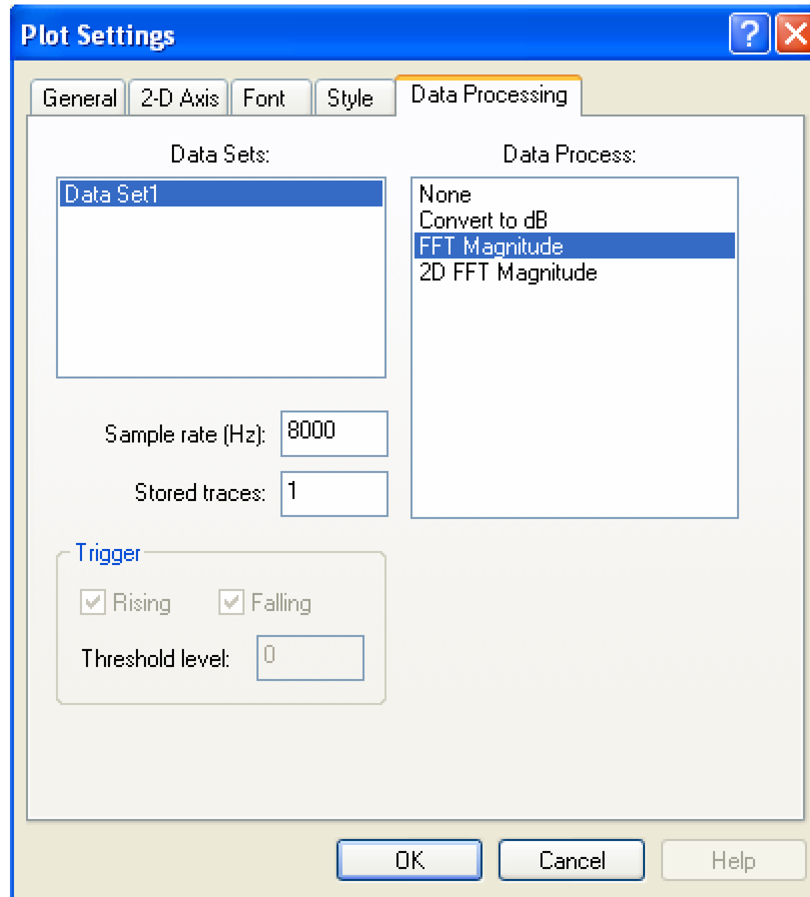
Configure...
Modify Settings...
Save Settings...
Export...

Auto Refresh
Auto Refresh Settings...

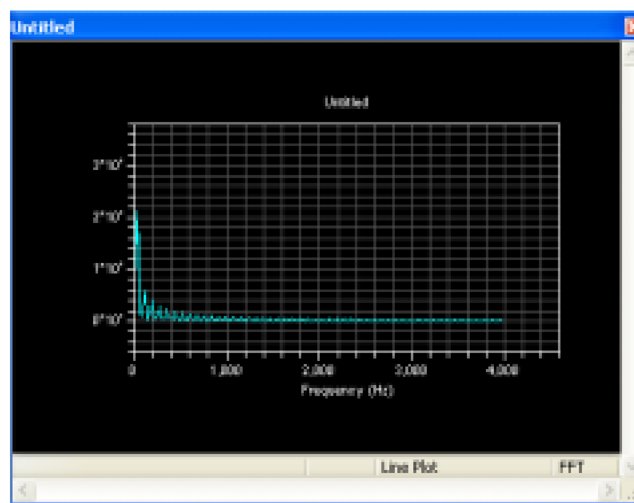
Allow Docking
Close

Float In Main Window

Se poate efectua o transformare Fourier cu comanda: *Modify Settings* -> *Data Processing* -> *FFT Magnitude* -> *OK* (se alege si o frecventa de esantionare adecvata).



Rezultatul afisarii devine:



Exemplu general de program

In continuare se va ilustra modul de programare, in limbajul C, pentru microcontrolerul ADSP2181 si codecul AD1847.

Exemplul de program implementeaza urmatoarea organigrama generala:

Initializari microcontroler ADSP2181
(mod de lucru, porturi seriale, intreruperi SPORT TX, SPORT RX)
Initializare codec AD1847
iTxIsrCounter=0
iCodecInitFinished = 0

(Initializari specifice aplicatiei)
Validare intreruperi

Asteapta intreruperi

Rutina de servire a intreruperilor SPORT TX :
Daca *iCodecInitFinished = 0* **atunci**
{
Transmite cuvint de programare pentru AD1847
Incrementeaza contor *iTxIsrCounter*
Daca *iTxIsrCounter* = Numarul de cuvinte de programare necesare **atunci** *iCodecInitFinished* = 1
}
Revenire

Rutina de servire a intreruperilor SPORT RX

Citeste canalul de receptie Left al AD1847
Citeste canalul de receptie Right al AD1847
Prelucreaza datele citite
Scrie canalul de transmisie Left al AD1847
Scrie canalul de transmisie Right al AD1847
Revenire

Programul complet este prezentat in continuare:

MAIN.C

```
#include "talkthrough.h"

int iLeftChannelIn, iRightChannelIn;                    // input data
int iLeftChannelOut, iRightChannelOut;                  // ouput data
int iCodecRegs[SIZE_OF_CODEC_REGS] =                    // array for codec registers
{
        // names are defined in "talkthrough.h"
                        DATA_FORMAT                     | CLOR | MCE | 0x57,
                        INTERFACE_CONFIGURATION         | CLOR | MCE | 0x09,
                        MISCELLANEOUS_INFORMATION       | CLOR | MCE | 0x40,
                        LEFT_INPUT_CONTROL              | CLOR | MCE | 0x00,
                        RIGHT_INPUT_CONTROL             | CLOR | MCE | 0x00,
                        LEFT_AUX_1_INPUT_CONTROL        | CLOR | MCE | 0x80,
                        RIGHT_AUX_1_INPUT_CONTROL       | CLOR | MCE | 0x80,
                        LEFT_AUX_2_INPUT_CONTROL        | CLOR | MCE | 0x80,
                        RIGHT_AUX_2_INPUT_CONTROL       | CLOR | MCE | 0x80,
                        LEFT_DAC_CONTROL                | CLOR | MCE | 0x00,
                        RIGHT_DAC_CONTROL               | CLOR | MCE | 0x00,
                        PIN_CONTROL                     | CLOR | MCE | 0x00,
                        DIGITAL_MIX_CONTROL             | CLOR | MCE | 0x00
};

// variables for internal use
asm(".global _iSport0RxBuffer;");           // buffers for sport0 are declared in assembler,
asm(".global _iSport0TxBuffer;");           // so they are placed on circular boundaries
asm(".var/circ _iSport0RxBuffer[3];");      // using ".var/circ"
asm(".var/circ _iSport0TxBuffer[3];");
volatile int iTxIsrCounter = 0, iCodecInitFinished = 0;

void main(void)
{
        InitSport0();
        InitInterrupts();
        InitCodec();

        while(1);
}
```

INIT_SPORT.C

```
#include "talkthrough.h"

void InitSport0(void)
{
        // pointers to memory mapped registers
        int *pSport0_Autobuf_Ctrl = &(* (int *) Sport0_Autobuf_Ctrl);
        int *pSport0_Rfsdiv = &(* (int *) Sport0_Rfsdiv);
        int *pSport0_Sclkdiv = &(* (int *) Sport0_Sclkdiv);
        int *pSport0_Ctrl_Reg = &(* (int *) Sport0_Ctrl_Reg);
        int *pSport0_Tx_Words0 = &(* (int *) Sport0_Tx_Words0);
        int *pSport0_Tx_Words1 = &(* (int *) Sport0_Tx_Words1);
```

```
	int *pSport0_Rx_Words0 = &(* (int *) Sport0_Rx_Words0);
	int *pSport0_Rx_Words1 = &(* (int *) Sport0_Rx_Words1);
	int *pSys_Ctrl_Reg = &(* (int *) Sys_Ctrl_Reg);

	// initialise sport0
	*pSport0_Autobuf_Ctrl = 0x0623;		// i2, i3 and m0 are used for autobuffering
	*pSport0_Rfsdiv = 0x0000;		// external frame sync
	*pSport0_Sclkdiv = 0x0000;		// external clock
	*pSport0_Ctrl_Reg = 0x860f;		// 16 bits/word, 32 words/frame, 1 bit mfd
	*pSport0_Tx_Words0 = 0x0007;		// transmit in slots 0, 1, 2, 16, 17, 18
	*pSport0_Tx_Words1 = 0x0007;
	*pSport0_Rx_Words0 = 0x0007;		// receive in slots 0, 1, 2, 16, 17, 18
	*pSport0_Rx_Words1 = 0x0007;
	*pSys_Ctrl_Reg = 0x1000;		// enable sport0

	// initialise dag1 registers
	// use i2/l2 for receiver

	circ_setup(2, (int *) iSport0RxBuffer, SIZE_OF_SPORT0_BUFFERS);
	// use i3/l3 for transmitter

	circ_setup(3, (int *) iSport0TxBuffer, SIZE_OF_SPORT0_BUFFERS);
	asm("m0 = 0x0001;");			// initialise modifier used for autobuffering
}
```

INIT_INTERRUPTS.C

```
#include "talkthrough.h"

void InitInterrupts(void)
{
	sysreg_write(sysreg_ICNTL, 0x0000);		// disable nested interrupts
	interruptf(SIGSPORT0XMIT, Sport0TxIsr);		// assign isr to interrupt vector
	interruptf(SIGSPORT0RECV, Sport0RxIsr);		// assign isr to interrupt vector
}


void Sport0RxIsr(int sig)
{
	iSport0TxBuffer[1] = iLeftChannelOut;	// write processed values to output buffers
	iSport0TxBuffer[2] = iRightChannelOut;
	iLeftChannelIn = iSport0RxBuffer[1];	// read in received values
	iRightChannelIn = iSport0RxBuffer[2];

	ProcessData();				// call function that contains user code
}

void Sport0TxIsr(int sig)
{
	if(!iCodecInitFinished)
	{
		iSport0TxBuffer[0] = iCodecRegs[iTxIsrCounter];
		iTxIsrCounter++;
		if(iTxIsrCounter == SIZE_OF_CODEC_REGS)
		{
			iSport0TxBuffer[0] = 0x0000;
```

```
                              iCodecInitFinished = 1;
                    }
          }
}
```

INIT_CODEC.C

```
#include "talkthrough.h"


void InitCodec(void)
{
          asm("ax0 = dm(i3, m0);");          // start data transfer by writing the first
          asm("tx0 = ax0;");                 // value of iCodecRegs into transmit register

          // wait for initialisation of codec
          while(!iCodecInitFinished);        // wait until all values in iCodecRegs have been sent

          // (flag is set in sport0 transmit isr)
          while(!(iSport0RxBuffer[0] & ACI));          // wait until autocalibration is in progress
          while(iSport0RxBuffer[0] & ACI);             // wait until autocalibration is finished

          // disable transmit interrupt
          sysreg_bit_clr(sysreg_IMASK, 0x0040);
}
```

TALKTHROUGH.H

```
#include <def2181.h>
#include <signal.h>
#include <sysreg.h>
#include <circ.h>

void InitInterrupts(void);
void InitSport0(void);
void InitCodec(void);
void Sport0RxIsr(int sig);
void Sport0TxIsr(int sig);

extern int iLeftChannelIn, iRightChannelIn, iLeftChannelOut, iRightChannelOut;
volatile extern int iSport0RxBuffer[], iSport0TxBuffer[];
extern int iCodecRegs[];
volatile extern int iTxIsrCounter, iCodecInitFinished;

// names for codec registers, used for iCodecRegs[]
#define LEFT_INPUT_CONTROL                      0x0000
#define RIGHT_INPUT_CONTROL                     0x0100
#define LEFT_AUX_1_INPUT_CONTROL                0x0200
#define RIGHT_AUX_1_INPUT_CONTROL               0x0300
#define LEFT_AUX_2_INPUT_CONTROL                0x0400
#define RIGHT_AUX_2_INPUT_CONTROL               0x0500
#define LEFT_DAC_CONTROL                        0x0600
#define RIGHT_DAC_CONTROL                       0x0700
#define DATA_FORMAT                             0x0800
#define INTERFACE_CONFIGURATION                 0x0900
#define PIN_CONTROL                             0x0a00
```

```
#define MISCELLANEOUS_INFORMATION           0x0c00
#define DIGITAL_MIX_CONTROL                 0x0d00

// bit definitions for ad1847
#define CLOR                        0x8000        // clear overrange
#define MCE                         0x4000        // mode change enable
#define ACI                         0x0002        // autocalibrate in progress

// some buffer sizes
#define SIZE_OF_CODEC_REGS          13            // size of array iCodecRegs
#define SIZE_OF_SPORT0_BUFFERS  3                 // size of buffers for sport0
```

PROCESS_DATA.C

```
#include "talkthrough.h"

void ProcessData(void)
{
        iLeftChannelOut = iLeftChannelIn;
        iRightChannelOut = iRightChannelIn;
}
```

Se remarca faptul ca programul are sectiuni scrise in limbaj de asamblare; acest lucru este necesar deoarece trebuie definite cele doua buffere circulare pentru operarea cu codecul AD1847, iar limbajul C nu poate defini in mod corespunzator registrele index necesare adresarii circulare.

Sint rezervate registrele index I2 si I3 si registrul M0 pentru receptia si transmisia codecului; acest lucru trebuie transmis ca o comanda in Visual DSP (pentru compilator *"-reserve i2,i3,m0"* si pentru linker *"-MD__RESERVE_AUTOBUFFER_REGS__"*).

Buferele circulare sint definite in limbaj de asamblare: *asm(".var/circ _iSport0RxBuffer[3];")*; si *asm(".var/circ _iSport0TxBuffer[3];");*

Cu macrodefinitia *circ_setup(numar_reg_index, pointer_buffer, dim_buffer)* se seteaza in mod corespunzator adresarea circulara, ca in exemplul *circ_setup(2, (int *) iSport0RxBuffer, SIZE_OF_SPORT0_BUFFERS);*

Pointerii catre registrele de control ale SPORT sint definite cu instructiunea: *int *pSport0_Autobuf_Ctrl = &(* (int *) Sport0_Autobuf_Ctrl);*

Intreruperile sint setate si validate cu instructiunea *interruptf(NIVEL_INT, Rutina_de_Servire);* ca in exemplul *interruptf(SIGSPORT0RECV, Sport0RxIsr);*

Este necesar un fisier de descriere a arhitecturii *(*.ldf*) care defineste blocurile de memorie si alocarea sectiunilor de program in aceste blocuri.

Exemplu de aplicatie: generator de functii

Se va prezenta un exemplu de aplicatie, in limbajul C, care implementeaza un generator de semnale periodice cu 3 tipuri de forma de unda: sinusoidal, dreptunghiular si triunghiular.

Se vor nota: f_d – frecventa semnalului dorit (generat)

f_s – frecventa de esantionare

$n = \frac{f_s}{f_d}$ - numarul de esantioane pe o perioada

$j \in \{ 0, 1, 2 , \ldots, n\text{-}1 \}$ – indexul esantionului curent pentru semnalele dreptunghiular si triunghiular

x – variabila pentru calculul semnalului sinusoidal

k – parametrul care controleaza forma semnalului dreptunghiular

k_1 – parametrul care controleaza forma semnalului triunghiular

A – amplitudinea semnalelor

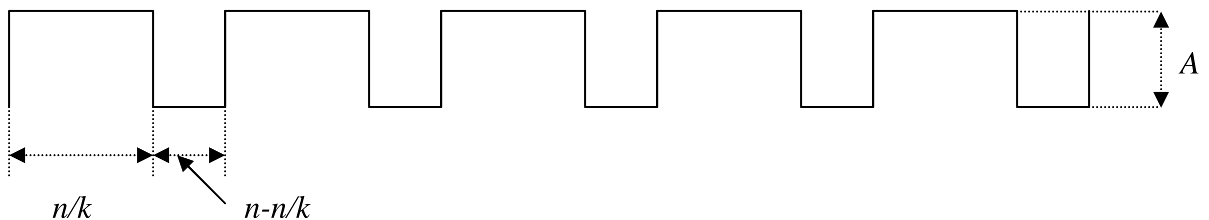
Generarea semnalului sinusoidal

Forma de unda sinusoidala este: $y(j) = \sin(\frac{2\pi f_d}{f_s} j), j \in N$

Daca initial $x = 0$ si la fiecare pas $x = x + \frac{2\pi f_d}{f_s}$, atunci semnalul sinusoidal este calculat ca $y = \sin(x)$ la fiecare iteratie.

Generarea semnalului derptunghiular

Se considera ca primele $\frac{n}{k}$ esantioane din perioada semnalului vor avea valoarea A, iar restul de $n - \frac{n}{k}$ esantioane vor avea valoarea 0.



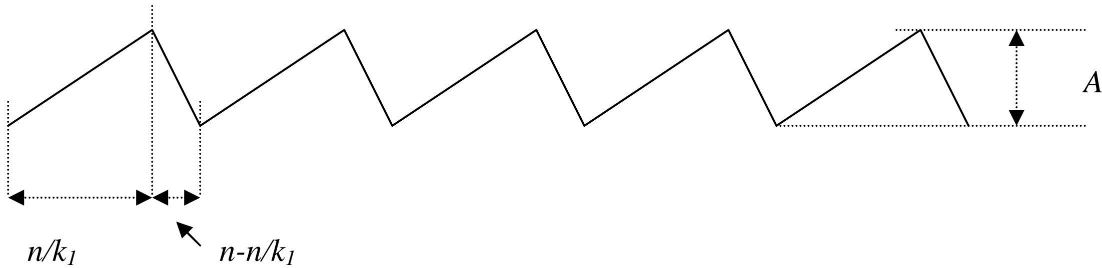
Conditia care trebuie testata in program este $j \le \frac{n}{k}$ sau $j \le \frac{f_s}{k f_d}$ adica $k f_d j \le f_s$

Forma de unda este data de

$$y(j) = \begin{cases} A & daca \quad k f_d j \le fs \\ 0 & in \quad rest \end{cases}$$

Generarea semnalului triunghiular

Se considera ca primele $\frac{n}{k_1}$ esantioane din perioada semnalului reprezinta palierul crescator de la valoarea 0 la valoarea *A*, iar restul de $n-\frac{n}{k_1}$ esantioane reprezinta palierul descrescator de la valoarea *A* la valoarea 0.



Forma de unda este data de formula:

$$y(j)=\begin{cases}\frac{k_1Aj}{n}, & daca \quad j\le\frac{n}{k_1}\\ \frac{k1A}{k1-1}(1-\frac{j}{n}), & in \quad rest\end{cases}$$

sau

$$y(j)=\begin{cases}\frac{k_1f_dAj}{f_s}, & daca \quad k_1f_dj\le f_s\\ \frac{k1A}{k1-1}(1-\frac{f_dj}{f_s}), & in \quad rest\end{cases}$$

S-a introdus o variabila de stare, *state*, care precizeaza ce forma de unda va fi generata: *state* = 0 – semnal sinusoidal, *state* = 1 – semnal dreptunghiular si *state* = 2 – semnal triunghiular.

In programul prezentat anterior se va modifica procedura *ProcessData* astfel:

```
PROCESS_DATA // pentru generatorul de functii

void ProcessData(void)
{

switch (state)
{
        case 0:
                {
                y=A*sinf(x);
                y1=FRACT_TO_INT(y);
                x=x+2*PI*fd/fs;
                if (x>=2*PI) x=0.0;
                break;
                }

        case 1:
                {
                if (k*fd*j<=fs) y=0; else y=A;
                j++;
                if (fd*j>=fs) j=0;
                y1=FRACT_TO_INT(y);
                break;
                }
        case 2:
                {
                if (k1*fd*j<=fs) y=k1*A*j*(fd/fs);
                else y=(k1*A)/(k1-1)*(1-j*(fd/fs));
                j++;
                if (fd*j>=fs) j=0;
                y1=FRACT_TO_INT(y);
                break;
                }
}
        iLeftChannelOut = y1;
        iRightChannelOut = y1;
}
```

Fisierul *Talkthrough.h* va fi completat cu urmatoarele definitii:

```
#include <math.h>
#include <fract.h>

#define PI 3.14159265
#define fd 500.0
#define fs 8000.0
#define A 0.999969
#define k         4.0
#define k1        4.0
```

Datorita faptului ca microcontrolerul ADSP2181 lucreaza in virgula fixa (format 1.15 sau fractionar) este necesara o conversie la aceasta reprezentare cu functia *FRACT_TO_INT*.

Pentru simplificarea simularii intreruperile utilizate sint generate pe nivelul *SIGIRQ2*.

Pentru executia in timp real pe placa EZ_LITE ADSP2181 se introduce un nivel suplimentar de intrerupere, *SIGIRQE* asociat butonului IRQE. Rutina de servire a acestei intreruperi modifica in mod succesiv, la fiecare actionare, valorea variabilei *state*:

```
interruptf(SIGIRQE, state_SW);

void state_SW(int sig)
{
	state=(state+1)%3;
	j=0;
	x=0.0;
	asm("toggle fl1;");
}
```

PLACA DE EVALUARE EZ-KIT Lite ADSP2181

Placa de evaluare EZ-KIT Lite ADSP2181 are urmatoare arhitectura (figura 1):

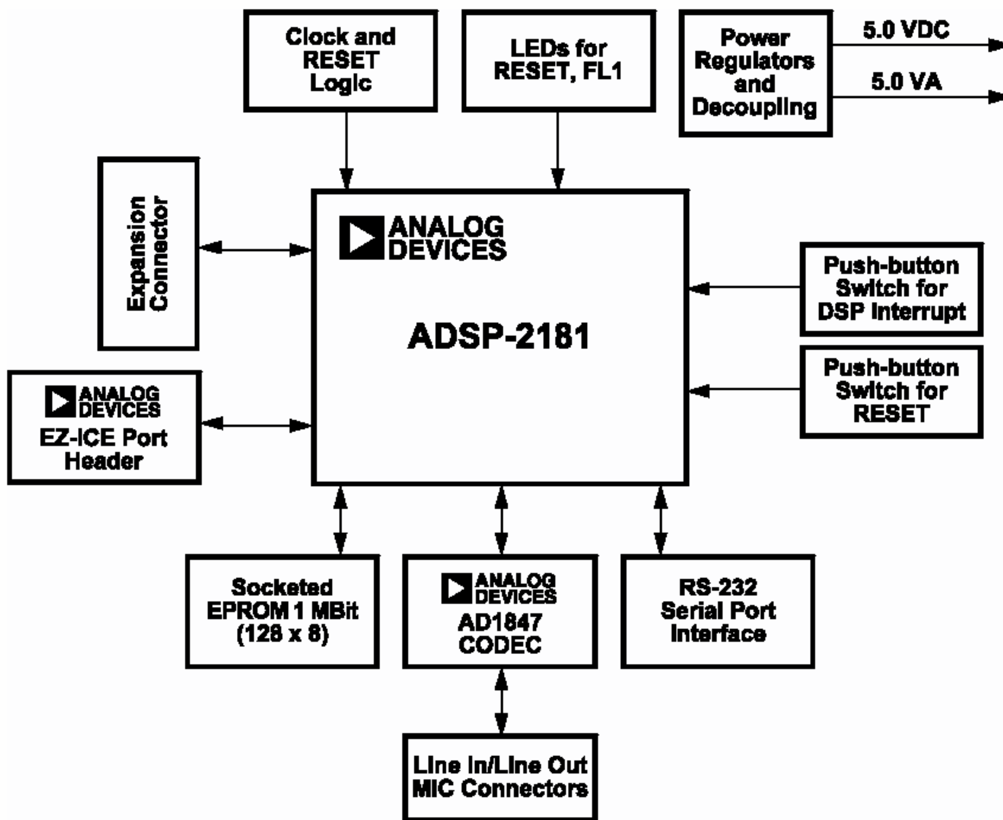


Figura 1. Ahhitectura placii EZ-KIT Lite ADSP2181

Aceasta placa de evaluare are in componenta urmatoarele blocuri:

1. Microcontrolerul ADSP2181 (specializat in prelucrari de semnale – Digital Signal Processing DSP)
2. Interfata de intrare iesire analogica (codecul AD1847, filtre *antialiasing* la intrare, filtre de netezire la iesire, amplificatoare de intrare)
3. Memoria pentru incarcarea programelor (boot memory) de tip EPROM
4. Elemente de interfata cu utilizatorul (butoane , LED-uri)

Figura 2 indica modul de realizare proctica a placii EZ KIT Lite ADSP2181.

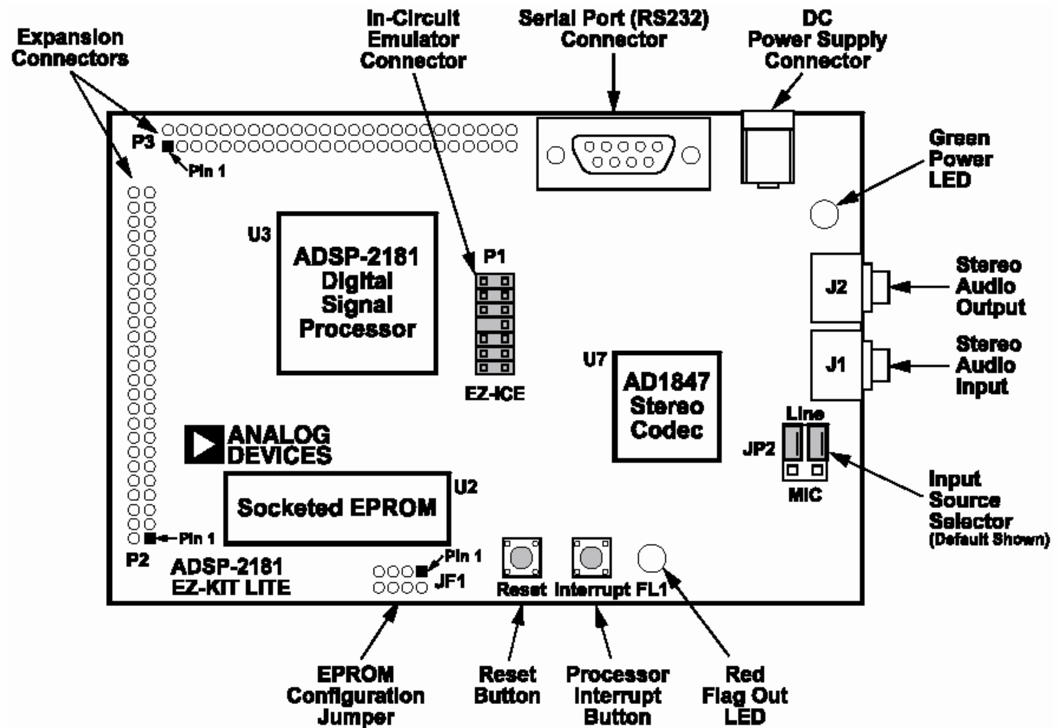


Figura 2. Realizarea proctica a placii EZ KIT Lite ADSP2181

Arhitectura microcontrolerului ADSP2181 este ilustrata in figura 3.

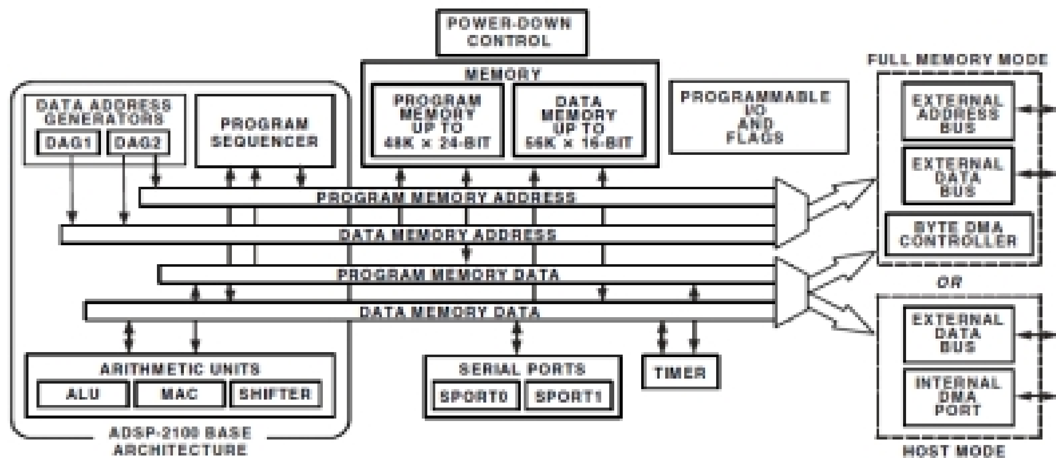


Figura 3. Arhitectura microcontrolerului ADSP2181

Microcontrolerul ADSP2181 are urmatoarele caracteristici:

- arhitectura Harvard modificata cu posibilitatea comunicarii intre busul de date al memoriei de program PMD si busul de date al memoriei de date DMD
- 3 unitati de calcul specializate (ALU, MAC, SHIFTER) ce pot opera in paralel
- unitatile de calcul pot fi interconectate intre ele pe un bus suplimentar de rezultate, R
- 2 unitati de adresare (Data Address Generator – DAG) se pot opera simultan pe cele doua busuri de adrese ale memoriei de program PMA si ale memoriei de date DMA
- memorie interna de date si de program
- timer
- 2 porturi seriale SPORT0 si SPORT1
- porturi paralele de comunicatie cu dispozitive externe (Byte Direct Memory Access – BDMA si Internal DMA - IDMA)
- posibilitatatea conectarii unor memorii externe

Structura unitatilor aritmetice ALU, Multiply and Accumulate (MAC) si SHIFTER, ale microcontrolerului ADSP2181 este ilustrata in figurile 4, 5 si 6.

Portul serial SPORT din microcontrolerul ASP2181 are urmatoarele caracteristici:

- este un port serial cu 5 semnale specifice (ceas serial – SCLK, semnale de sincronizare la receptie – RFS si la transmisie – TFS, semnale seriale de date receptionate - DR sau transmise – DT)
- ceasul serial, semnalele de sincronizare si lungimea caracterului sint programabile
- la fiecare caracter receptionat sau transmis se genereaza o intrerupere
- portul poate functiona cu *autobuffering*: intreruperile se genereaza la receptionarea sau transmiterea unui numar prestabilit de caractere
- portul poate functiona in modul *multicanal* (folosind un singur semnal de sincronizare se pot receptiona sau transmite pe aceeasi linie seriala mai multe caractere)

Modurile *autobuffering* si *multicanal* pot fi utilizate simultan astfel:

- se definesc doua buffere de memorie cu adresare circulara de dimensiune N (numarul de canale dorite); un buffer de receptie *Rxbuff* si un buffer de transmisie *TxBuff*.
- pentru adesarea celor doua buffere circulare se aloca doua grupuri de registre (index, modificator, lungime) care vor fi actualizate si testate in mod automat de portul serial. Intreruperile vor fi generate la umplerea sau golirea acestor buffere circulare.

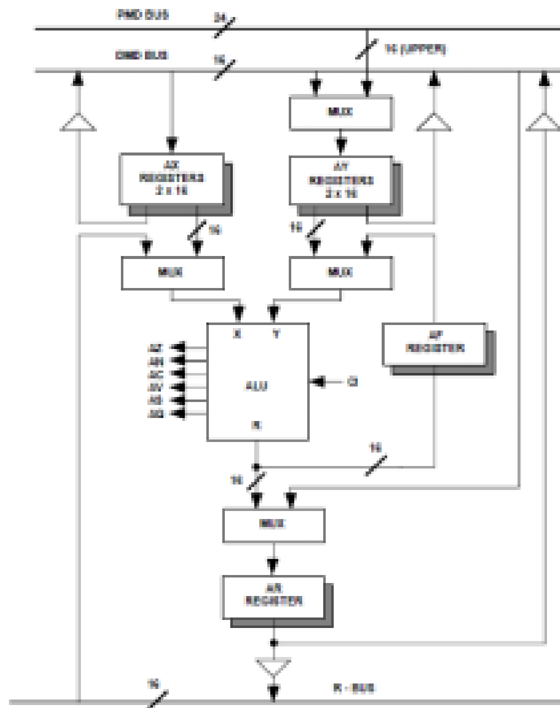


Figura 3. Unitatea aritmetica ALU – ADSP2181

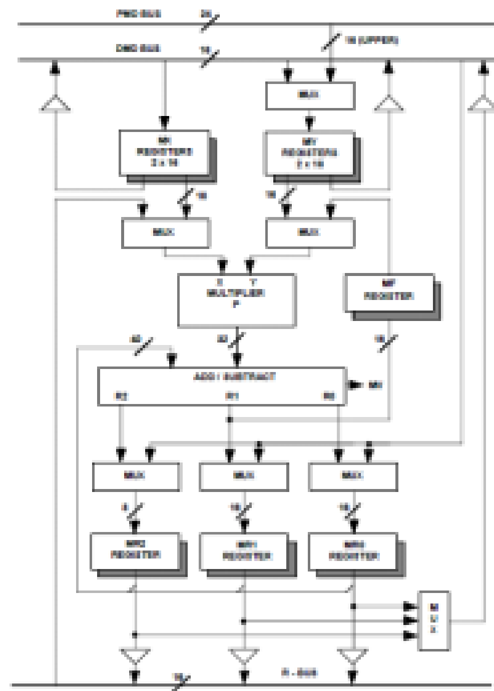


Figura 4. Unitatea de inmultire cu acumulare MAC – ADSP2181

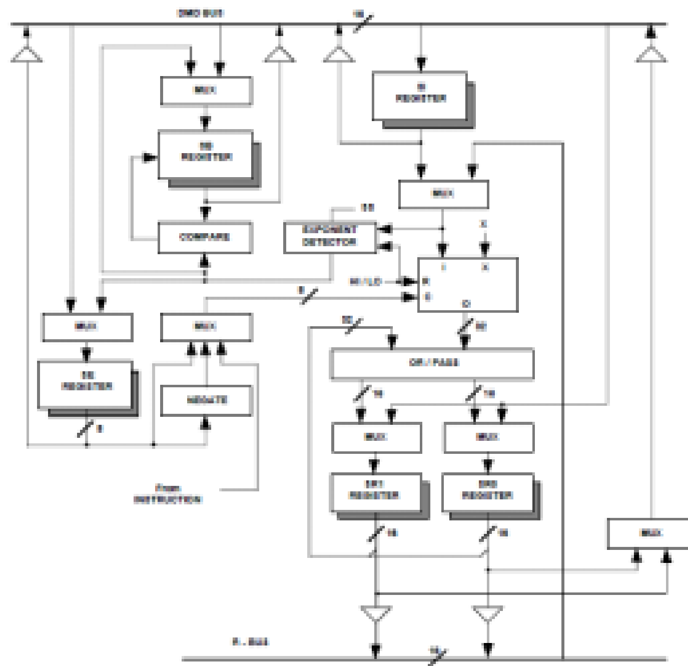


Figura 5. Unitatea de deplasare SHIFTER– ADSP2181

Figura 7 ilustreaza functionarea portului serial SPORT in modul *autobuffering, multicanal*.

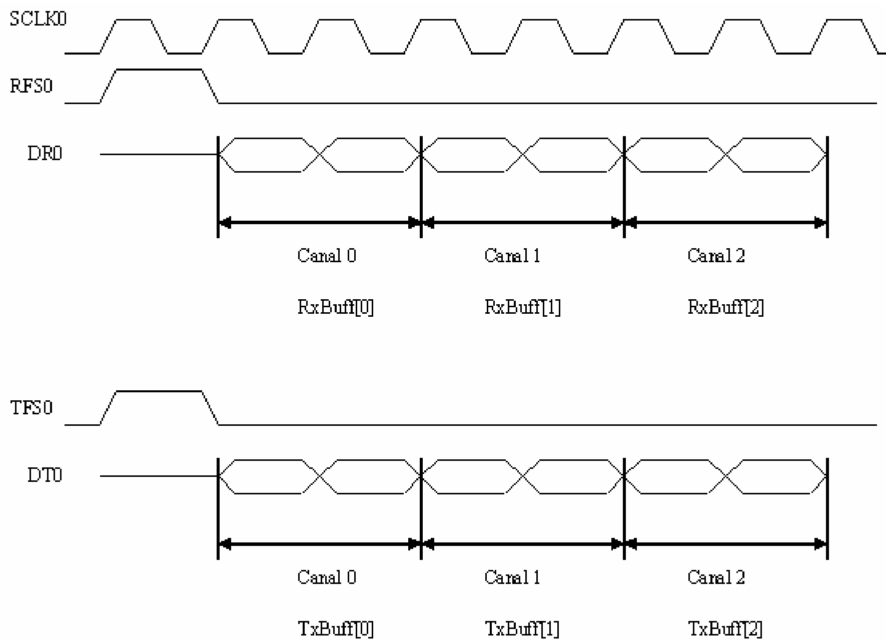


Figura 7. Modul *autobuffering* si *multicanal* (SPORT0)

Conectarea in sistem a microcontrolerului ADSP2181 este prezentata in figura 8.

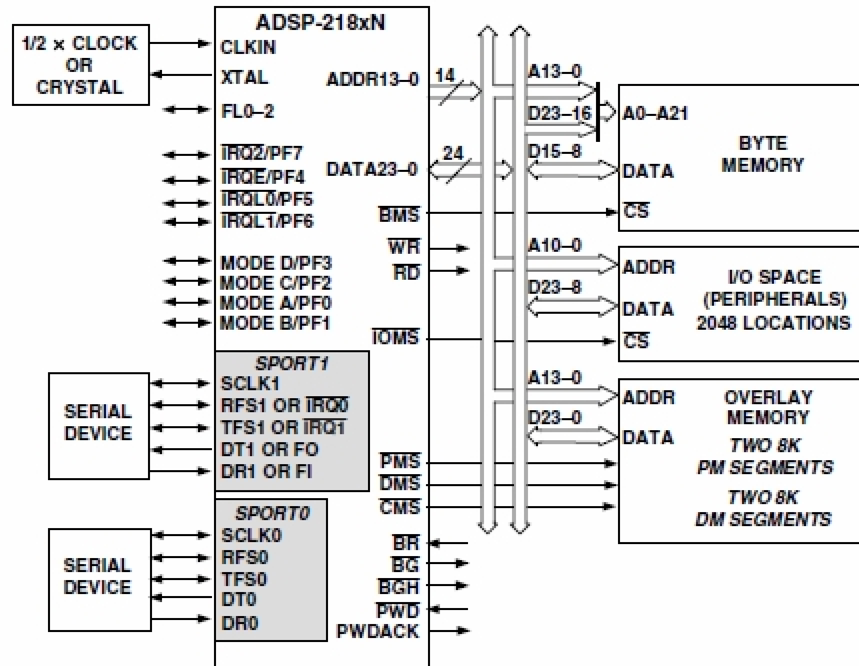


Figura 8. Conectarea microcontrolerului ADSP2181

Placa de evaluare EZ KIT Lite 2181 prelucreaza semnale analogice. Conversia analog digitala si digital analogic este realizata de codecul AD1847, a carei structura este ilustrata in figura 9.

Codecul AD1847 este interconectat cu microcontrolerul ADSP2181 pe un port serial (SPORT0). Comunicatia intre AD1847 si ADSP2181 presupune programarea portului serial in modul *mulicanal* cu *autobuffering* ca in figura 10.

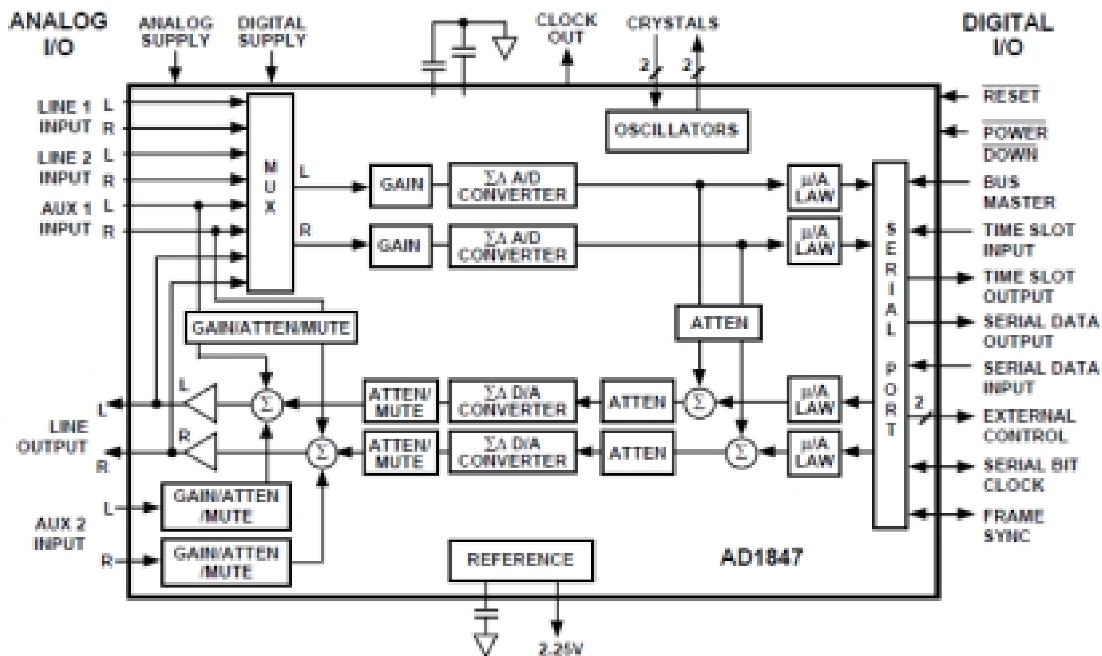


Figura 9. Structura codecului AD1847

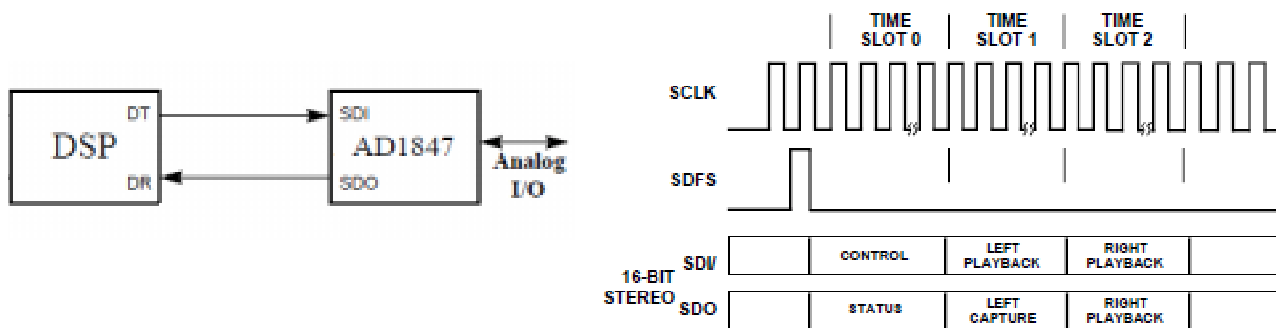


Figura 10. Connectarea AD1847 – ADSP2181

Exista 3 canale la transmisie (comanda, date-stinga, date -dreapta) si 3 canale la receptie (stare, date- stinga si date- dreapta).

Codecul va genera semnalele de sincronizare necesare (ceas serial SCLK si semnalele RFS , TFS).

Informatia de date va fi receptionata in locatiile RxBuff[1] si RxBuff[2], iar informatia transmisa va fi scrisa in locatiile TxBuff[1] si TxBuff[2]. Acest lucru se va efectua in rutina de intreruperi generate de portul serial SPORT0-Rx.

Programarea codecului AD1847 se realizeaza prin transmiterea in canalul de comanda a 13 cuvinte de programare. Structura unui cuvint de programare este ilustrata in figura 11. Structura cuvintului de stare este prezentata in figura 12.

Data 15	Data 14	Data 13	Data 12	Data 11	Data 10	Data 9	Data 8
CLOR	MCE	RREQ	res	IA3	IA2	IA1	IA0
Data 7	**Data 6**	**Data 5**	**Data 4**	**Data 3**	**Data 2**	**Data 1**	**Data 0**
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0

CLOR (Clearoverrange) – daca este “1” atunci se sterg biti overrange din registrul de stare

MCE (Mode Change Enable) – daca e “1” se permite modificarea registrelor de comanda

RREQ(Read Request) – “1” reprezinta o cerere de citire a registrului ce comanda indexat de bitii IA3-0

IA3-IA0 – (Index address) index pentru registrul de comanda; Exista 13 registre de comanda care stabilesc rata de esantionare, amplificarea/atenuarea pe fiecare canal, modul de lucru al codecului (16/32 canale, 1 fir sau 2 fire).

DATA7-DATA0 – datele inscrise in registrul de comanda

Figura 11. Structura unui cuvint de programare pentru AD1847

Data 15	Data 14	Data 13	Data 12	Data 11	Data 10	Data 9	Data 8
res	res	RREQ	res	ID3	ID2	ID1	ID0
Data 7	**Data 6**	**Data 5**	**Data 4**	**Data 3**	**Data 2**	**Data 1**	**Data 0**
res	res	ORR1	ORR0	ORL1	ORL0	ACI	INIT

RREQ (Read Request) – copie a bitului RREQ din registrul de comanda

ID3-ID0 (AD1847 Identifier)

ORR1-0 (Overrange Right Detect), ORL1-0 (Overrange Left Detect) – detecteaza situatiile in care semnalul de intrare pe canalele dreapta sau stinga sunt in afara gamei dinamice a codecului.

ACI (Autocalibrate in progress) – “1” semnifica autocalibrare in desfasurare, “0” indica faptul ca autocalibrarea s-a incheiat.
Acest bit trebuie sa aiba o variatie “0”->”1”->”0”

INIT – “1” indica faptul ca circuitul AD1847 este in faza de initializare

Figura 12. Structura cuvintului de stare pentru AD1847

Secventa de programare necesara codecului AD1847 este ilustrata in figura 13.

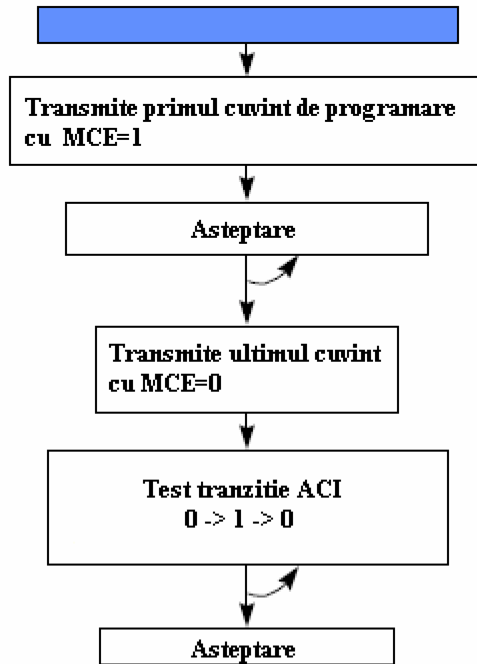


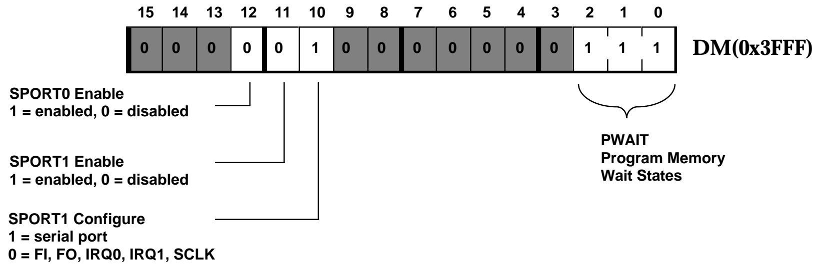
Figura 13. Programarea AD1847

Pentru implementarea organigramei din figura 13 se utilizeaza intreruperile generate de portul serial SPORT0 – Tx. In rutina de servire a acestor intreruperi se se transmit pe rand toate cuvintele de programare. Buclele de astepatre si testul de autocalibrare se vor realiza in programul principal.

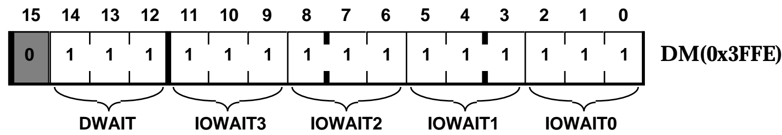
Control/Status Registers

Control register default bit values at reset are as shown; if no value is shown, the bit is undefined after reset. Reserved bits are shown on a gray field—these bits should always be written with zeros.

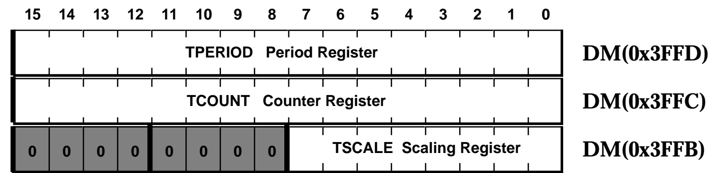
System Control Register



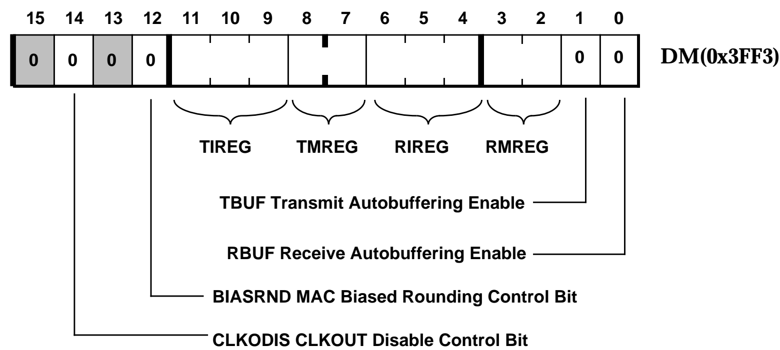
Data Memory Waitstate Register



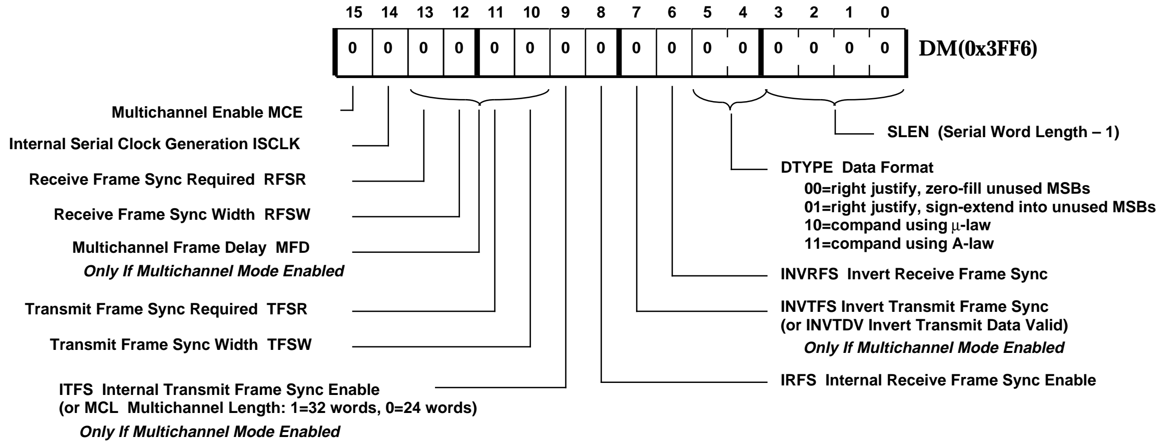
Timer Registers



SPORT0 Autobuffer Control



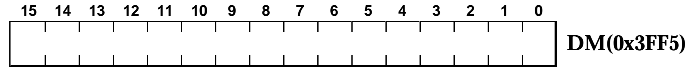
SPORT0 Control Register



$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 * (\text{SCLK frequency})} - 1$$

SPORT0 SCLKDIV

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

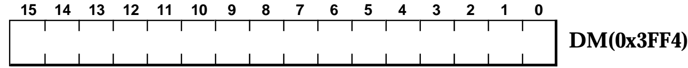


DM(0x3FF5)

$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

SPORT0 RFSDIV

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

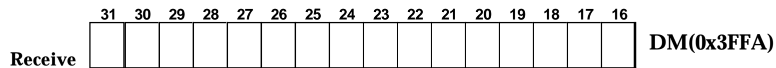


DM(0x3FF4)

SPORT0 Multichannel Word Enables

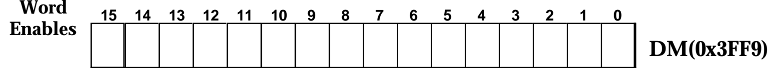
Receive Word Enables

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16



DM(0x3FFA)

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

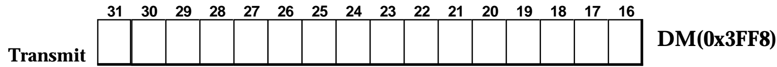


DM(0x3FF9)

1=Channel Enabled
0=Channel Ignored

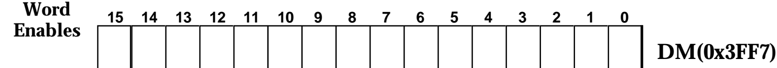
Transmit Word Enables

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16



DM(0x3FF8)

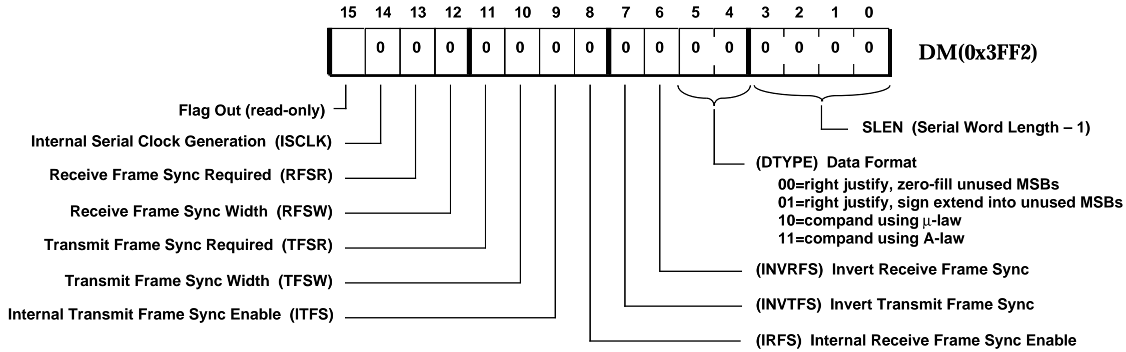
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



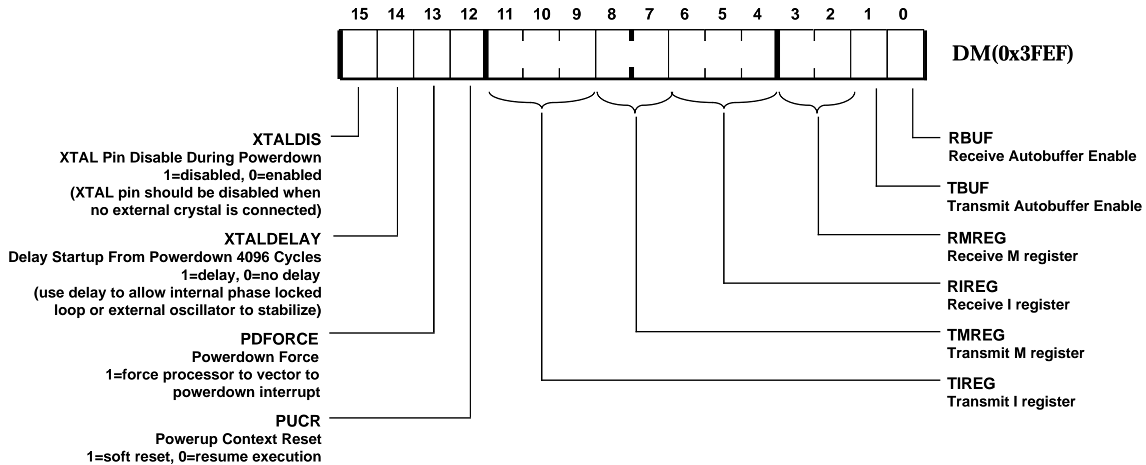
DM(0x3FF7)

Control/Status Registers

SPORT1 Control Register

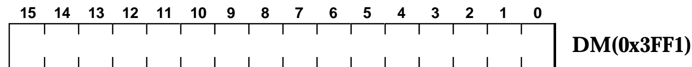


SPORT1 Autobuffer Control



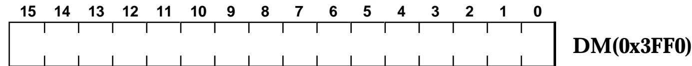
$$SCLKDIV = \frac{\text{CLKOUT frequency}}{2 * (\text{SCLK frequency})} - 1$$

SPORT1 SCLKDIV

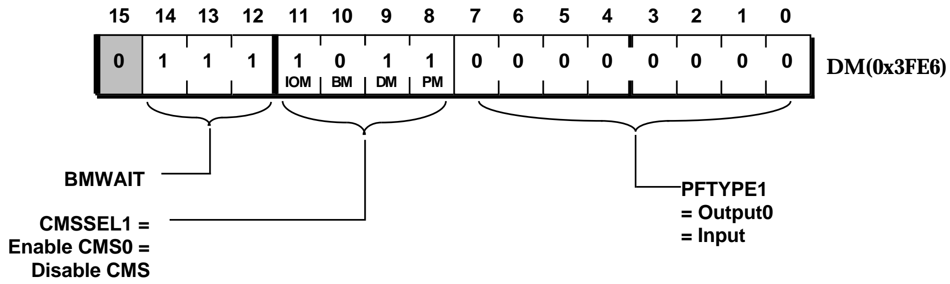


$$RFSDIV = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

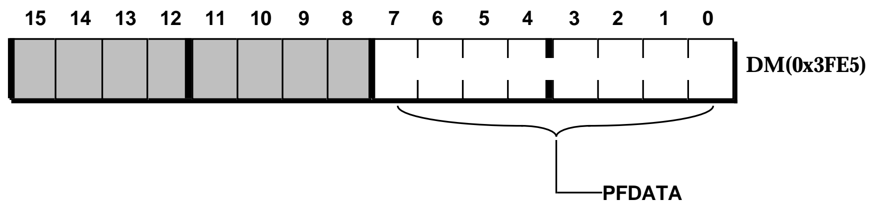
SPORT1 RFSDIV



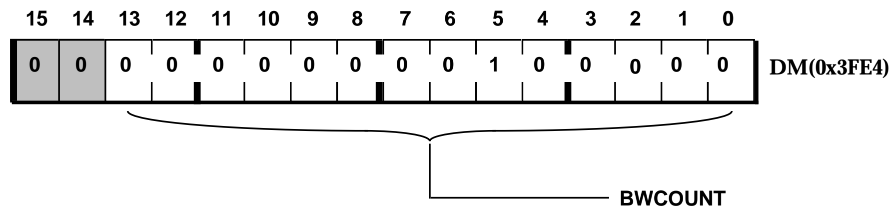
Programmable Flag & Composite Select Control



Programmable Flag Data

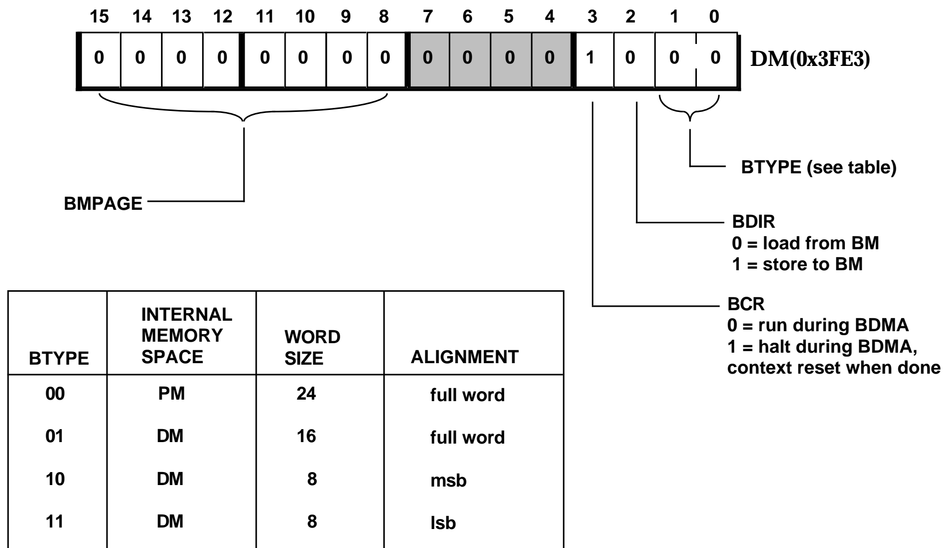


BDMA Word Count



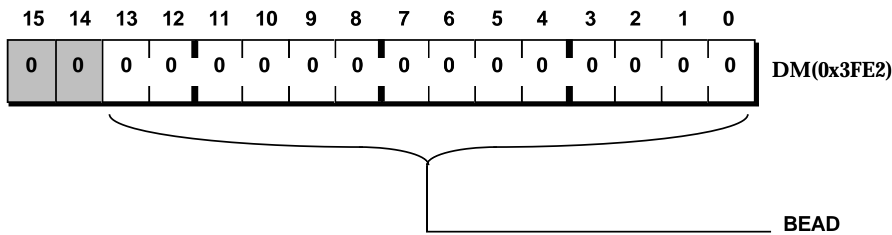
Control/Status Registers

BDMA Control

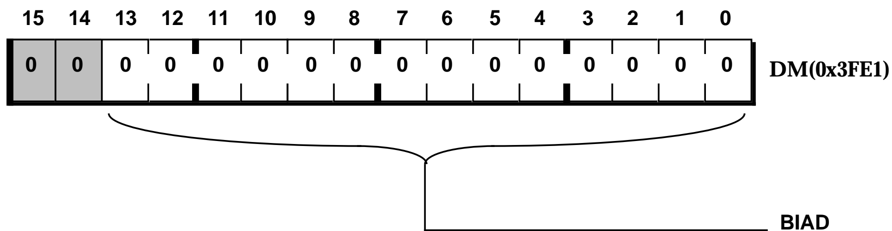


BTYPE	INTERNAL MEMORY SPACE	WORD SIZE	ALIGNMENT
00	PM	24	full word
01	DM	16	full word
10	DM	8	msb
11	DM	8	lsb

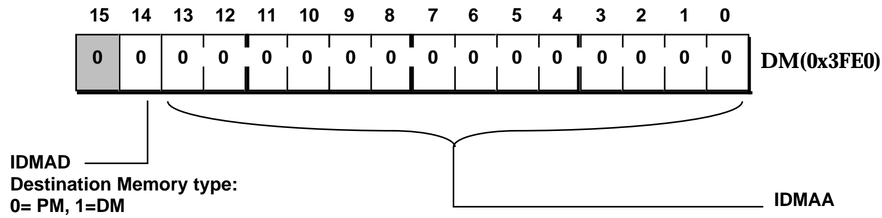
BDMA External Address



BDMA Internal Address



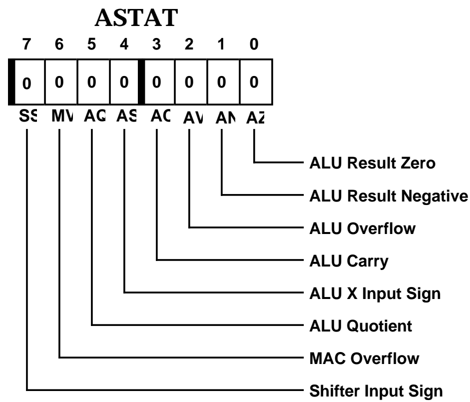
Programmable Flag & Composite Select Control



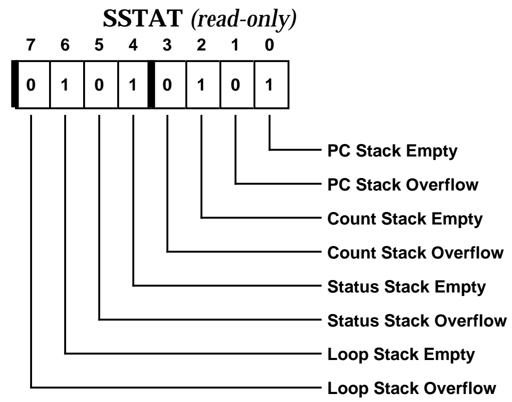
Status Registers

(Non-Memory-Mapped)

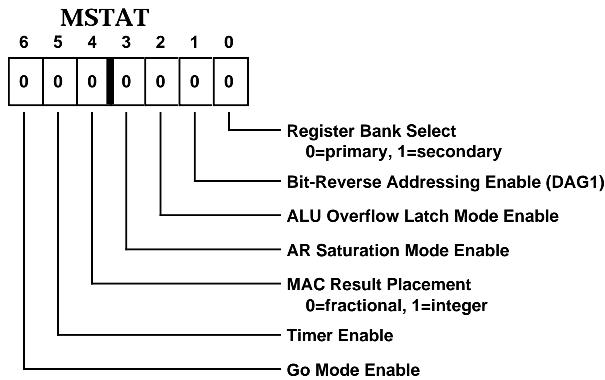
ASTAT



SSTAT *(read-only)*



MSTAT



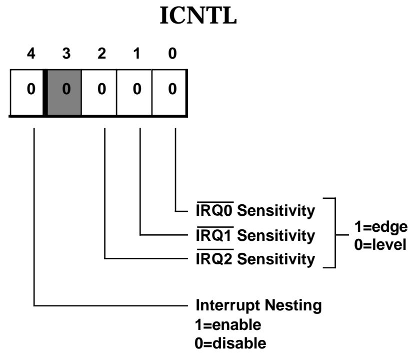
Mode Names for Mode Control Instruction

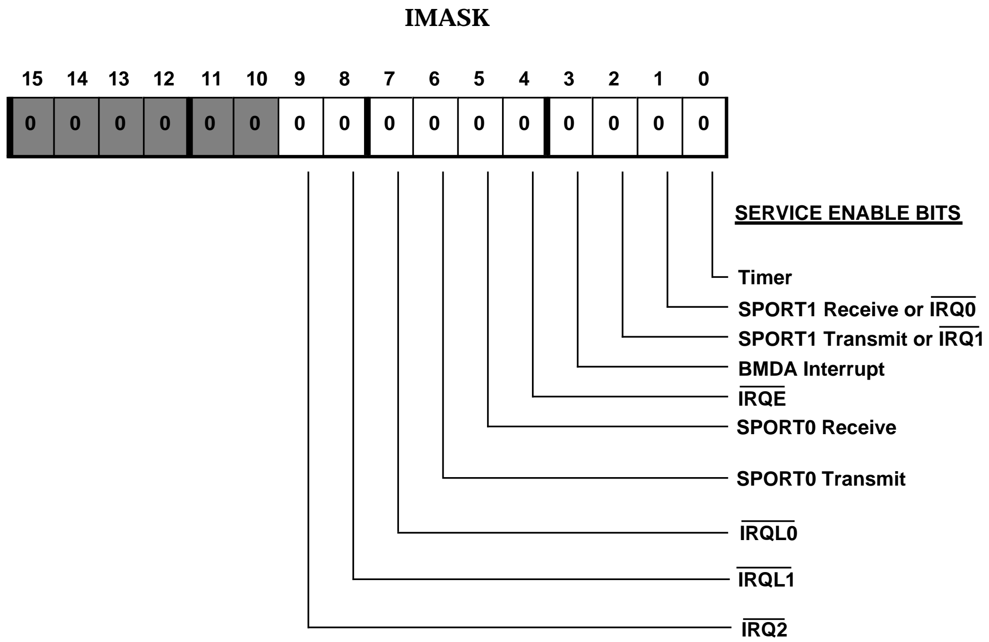
(see Miscellaneous Instructions on page 21)

Bit	*Mode Name*	
0	SEC_REG	Secondary register set
1	BIT_REV	Bit-reverse addressing in DAG1
2	AV_LATCH	ALU overflow (AV) status latch
3	AR_SAT	AR register saturation
4	M_MODE	MAC result placement mode
5	TIMER	Timer enable
6	G_MODE	Go mode enable
7	INTS	Interrupt enable

Interrupt Registers

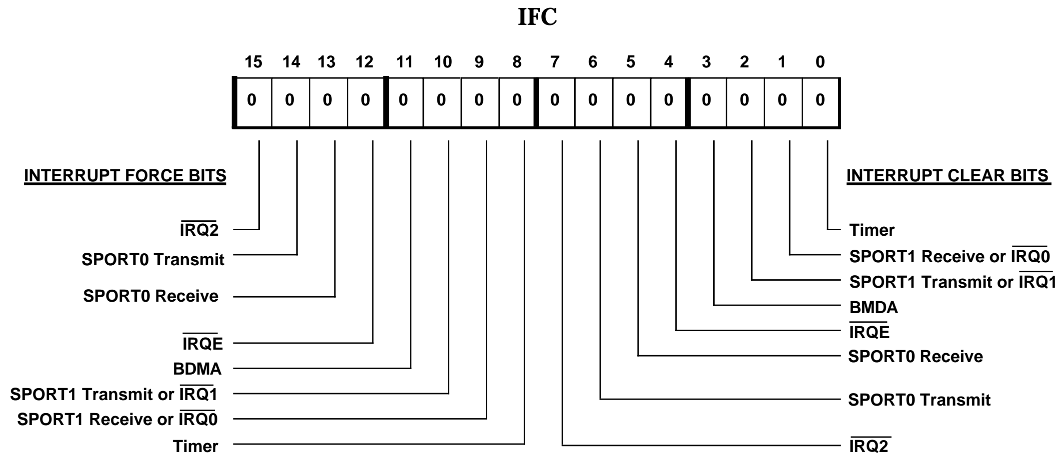
(Non-Memory-Mapped)





Interrupt Registers

(Non-Memory-Mapped)



Memory Maps

Data Memory

Data Memory	*Address*
32 Memory mapped registers	0x3FFF 0x3FE0
Internal 8160 words	0x3FDF 0x2000
8K Internal (DMOVLAY=0) or External 8K (DMOVLAY=1,2)	0x1FFF 0x0000

Program Memory

Program Memory	*Address*
8K Internal (PMOVLAY = 0, MMAP = 0) or **External 8K** (PMOVLAY = 1 or 2, MMAP = 0)	0x3FFF 0x2000
8K Internal	0x1FFF 0x0000

MMAP = 0

Interrupt Vector Tables

ADSP-2181

Interrupt Source	*Interrupt Vector Address (Hex)*	
Reset (or Power up with PUCR=1)	0000	*(highest priority)*
Power Down (non-maskable)	002C	
IRQ2	0004	
IRQL1	0008	
IRQL0	000C	
SPORT0 Transmit	0010	
SPORT0 Receive	0014	
IRQE	0018	
BDMA Interrupt	001C	
SPORT1 Transmit or IRQ1	0020	
SPORT1 Receive or IRQ0	0024	
Timer	0028	*(lowest priority)*

Control/Status Registers

Symbolic names for the memory-mapped control and status registers are provided in four files included with the development software. The symbols are defined as constants equal to the register addresses, and can be used for direct addressing. To use these symbols, include the appropriate file in the your source code files with the assembler's .INCLUDE directive:

Filename	*Include Directive To Use:*
DEF2181.H	.INCLUDE <DEF2181.H>;

Control/Status Register	*Data Memory Address*	*Assembly Code Symbol*
System Control Register	0x3FFF	Sys_Ctrl_Reg
Data Memory Wait State Control Register	0x3FFE	Dm_Wait_Reg
Timer Period	0x3FFD	Tperiod_Reg
Timer Count	0x3FFC	Tcount_Reg
Timer Scaling Factor	0x3FFB	Tscale_Reg
SPORT0 Multichannel Receive	0x3FFA	Sport0_Rx_Words1
Word Enable Register (32-bit)	0x3FF9	Sport0_Rx_Words0
SPORT0 Multichannel Transmit	0x3FF8	Sport0_Tx_Words1
Word Enable Register (32-bit)	0x3FF7	Sport0_Tx_Words0
SPORT0 Control Register	0x3FF6	Sport0_Ctrl_Reg
SPORT0 Serial Clock Divide Modulus	0x3FF5	Sport0_Sclkdiv
SPORT0 Rcv Frame Sync Divide Modulus	0x3FF4	Sport0_Rfsdiv
SPORT0 Autobuffer Control Register	0x3FF3	Sport0_Autobuf_Ctrl
SPORT1 Control Register	0x3FF2	Sport1_Ctrl_Reg
SPORT1 Serial Clock Divide Modulus	0x3FF1	Sport1_Sclkdiv
SPORT1 Rcv Frame Sync Divide Modulus	0x3FF0	Sport1_Rfsdiv
SPORT1 Autobuffer Control Register	0x3FEF	Sport1_Autobuf_Ctrl
Programmable Flag & Composite Select	0x3FE6	Prog_Flag_Comp_Sel_Ctrl
Programmable Flag Data	0x3FE5	Prog_Flag_Data
BDMA Word Count	0x3FE4	BDMA_Word_Count
BDMA Control	0x3FE3	BDMA_Control
BDMA External Address	0x3FE2	BDMA_External_Adress
BDMA Internal Address	0x3FE1	BDMA_Internal_Address
IDMA Control	0x3FE0	IDMA_Control

ADSP-2181 Registers