TR-AC-0059

003

Implementation of Adaptive Controller for ESPAR Antenna and Experiments for Performance Verification

Eddy Taillefer　程 俊　大平 孝

2001. 9.26

ATR環境適応通信研究所

Contents

List of Figures

List of Tables

Glossary and Key words

ABF : Aerial Beam Forming

API : Application Programming Interface

BPSK : Binary Phase Shift Keying

C : Programming language

$\mathbb{C}$: Set of complex numbers

CMA : Constant Modulus Algorithm

DBF : Digital Beam Forming

DSP : Digital Signal Processor

emf : electro-mechanic force

ESPAR : Electronicaly Steerable Passive Array Radiator

Frontend : Graphical interface of a software

IEEE : Institute of Electrical and Electronics Engineers

IEICE : Institute of Electronics, Information and Communication Engineers

ISM : Industrial, Scientific and Medical

LMS : Least Mean Square

MBF : Microwave Beam Forming

OS : Operating System

$\mathbb{R}$: Set of real numbers

RF : Radio Frequency

SDMA : Space Division Multiple Access

Key words : ESPAR antenna, adaptive beamforming, convergence, reactance, interference, gradient

INTRODUCTION

Nowadays, wireless communication systems are rapidly progressing, bringing more and more functionalities to users. Also the number of users is constantly increasing, and communication systems have to face to more constraints of bandwidth, data rates, and specifically in mobile wireless systems, the power consumption and mobile terminals size. The objective of researchers and engineers is to permanently reduce the cost and size of components without decreasing the quality of signal transmission.

ACR-ATR has proposed a Wireless Ad-hoc Community Network (WACNet) concept, in order to use in ad-hoc communities, which are formed with an unspecified number of people temporally connected with a common purpose. In this new concept, a new kind of antenna is proposed, the ESPAR antenna, a reactively controlled directive array that enables full azimuth directivity. This antenna has low cost, low power consumption and high power level.

Optimisation of the ESPAR antenna reactance is still a hard task; even though different ways have been tested, like Hamiltonian algorithm, CMA algorithm or genetic algorithm. A technique for adaptively controlling the load reactance's on the passive radiators, thus forming both beam and nulls is presented[?], for the first time, for the ESPAR antenna.

In this paper, the purpose is the sector beam forming of the ESPAR antenna[?]. Our work was to find an algorithm for this purpose and made experiments to conform this aim in practical use of the antenna. Actually this purpose has been succefully reached.
After presenting the WACNet concept, the ESPAR antenna features will be shown in details. Then an algorithm[?] for ESPAR antenna will be shwon, also, the modifications we made on this algorithm in order to fit to our purpose will be explained.
Finally, to show how ESPAR antenna can form beam, and trace the antenna sector pattern, simulations and experiments of this new algorithm will be presented.

Chapter 1

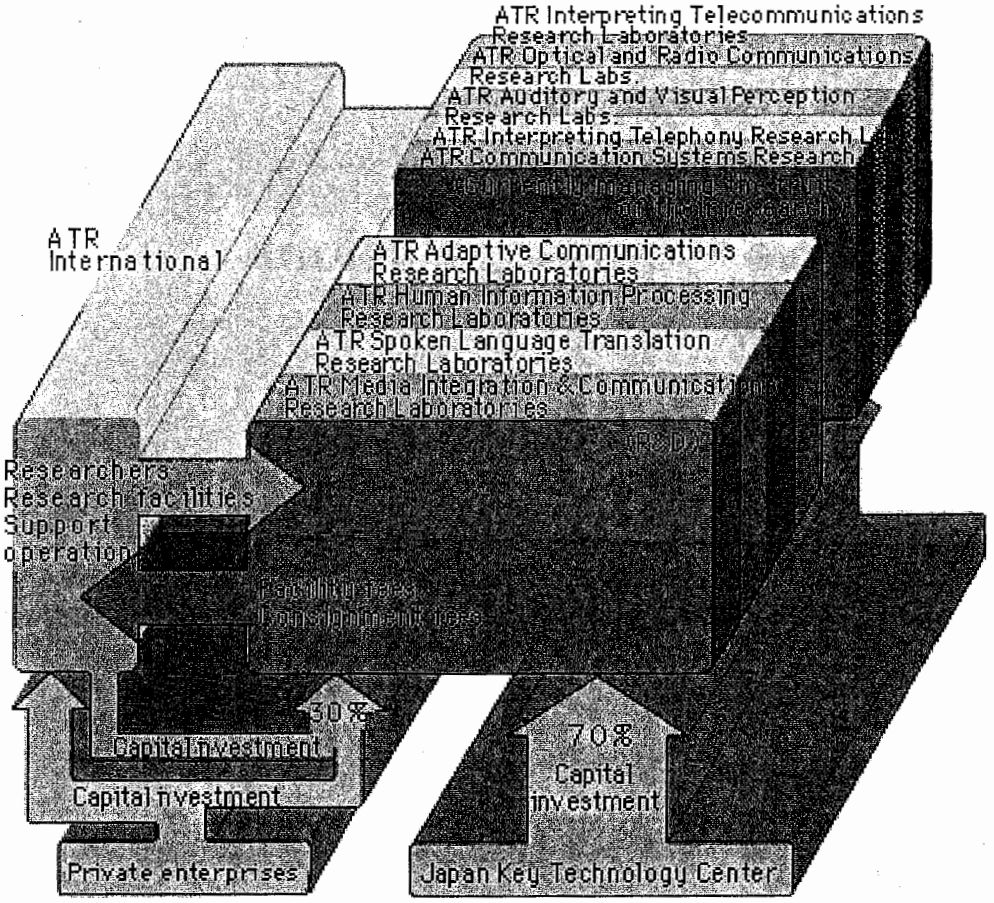
ATR Adaptive Communications Research Lab.

1.1 Introduction to ATR

In the information society of the 21st century, great volumes of diverse information will be disseminated through various media. This vast, varied information exchange will require advanced systems for accurate and efficient processing of information. Such systems must be easy to handle or operate . They must also be freely accessible and readily integrated into our day-to-day lives. These are vital aspects of this important research:

- New multi-media communication systems
- Cross-language global communication systems
- Ideal human-machine interfaces

- Developing fundamental technologies for highly adaptive communication systems



The ATR Group was established in March 1986 with support from various sections of industry, academia and government to serve as a major center of basic and creative telecommunications R&D. Its basic policy is to promote research collaboration among researchers both at home and abroad. The ATR Group comprises nine corporations: R&D corporations, four corporations designed to manage research fruits, and an umbrella corporation, ATR International. While the former four are now conducting research, the latter four have completed their R&D programs and are now managing the fruits of their research to promote their propagation and wider application. The four R&D corporations' activities are funded by the Japan Key Technology Center (70%) and 136 private enterprises (30%).

CAPITAL : ATR International : 22.0325 billion YEN (Invested by 136 Companies)

1.2 ACR Laboratories

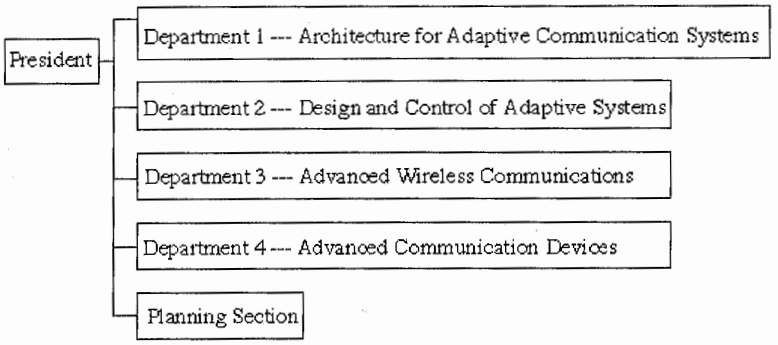
1.2.1 Organisation

Established on March 27, 1996

Research Period : March 1996 to February 2003

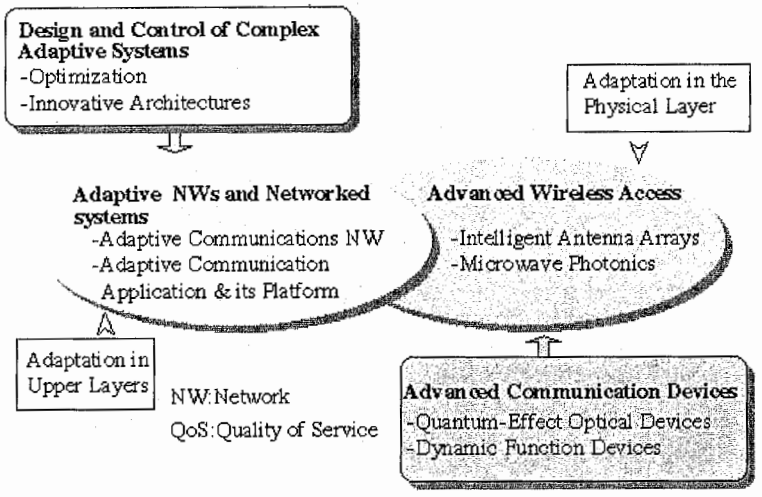
Total Budget for 7 years : ¥11,900 million

Organization

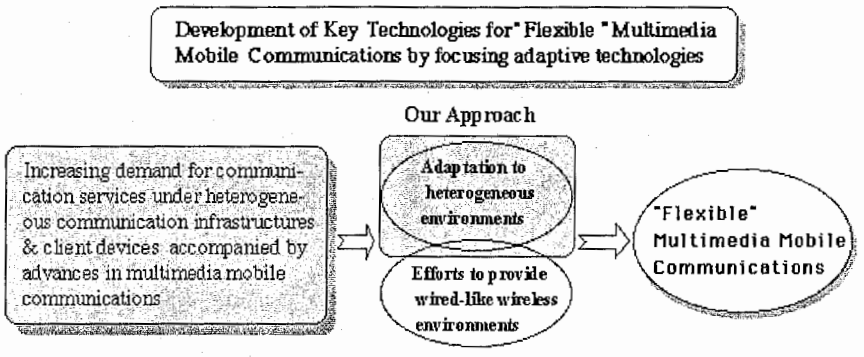


1.2.2 Research domains

-Adaptive Technologies for Multimedia Wireless Communications-



1.2.3 Main purpose project



Typical NW variation

NW	Bandwidth
LAN	10-100M
ISDN	128k
PHS	32k

Physical variation among clients

Terminal	Memory	Screen size	Bits/pixel
PC	64MB	1024x768	16b/p
laptop	16MB	800x600	8b/p
PDA	2MB	320x200	2b/p

Data rate in IMT-2000

Mobility	Date rate (Ws)
Fixed	2M
Movable	384k
Fast	144k

Chapter 2

Adaptive antenna

In this chapter, the WACNet concept will be presented, in order to situate the ESPAR antenna in this new concept. After, the ESPAR antenna structure and modelisation will be introduced. In the same time, it will be shown how the algorithm has been modified to fit to our purpose : *form the antenna sector beam.*

2.1 Wireless Ad-Hoc Community Network

The concept of Wireless Ad-Hoc Community Network(WACNet)[?] has recently became a hot topic. Such network can be considered as a means of communications among portable user terminals that temporally meet, and, where distance and time come close yet easy connection to a network infrastructure is not possible.
For example, Bluetooth[?] has been developed to connect between mobile handy phones, headsets, PCs, and other devices using the ISM[1] band (2.4GHz). Bluetooth has great possibilities to enable wireless ad-hoc networks.
The concept of WACNet is proposed in order to have more scalability, more speeds, and low output power level, that makes it very attractive for mobile network use.

2.1.1 Concept and features of the WACNet

The WACNet consists of portable user terminals and has no infrastructure such as hosts, base stations, switches, and hubs. It is therefore expected to be applicable anywhere; not only in fixed space such as a office company

[1]Industrial, Scientific and Medical

but also in a moving space such as a bus or train. The WACNet has scalability, i.e., the ability to accept a large number of users terminals because of autonomous segmentation and routing.
The key technologies developed to achieve the WACNet are: routing schemes, Space Division Multiple Access (SDMA), user terminals, microwave signal processing, and Electronically Steerable Passive Array Radiator (ESPAR) antennas.
The ESPAR antenna is the main subject of this report and will be explain more in details.

Routing schemes

Each user terminal carries out WACNet routing autonomously. This mean that each user terminal has switching, relaying functions and routes made by the user terminals through the use of these functions voluntarily. In these routing schemes, dynamic topologies such as the number of terminals and the relative locations and velocities among terminals must be considered.

Space Division Multiple Access

To achieve scalability, the adoption of dynamic network segmentation is considered. With dynamic network segmentation, a network is divided into a number of segments consisting of some terminals. One channel is able to be used repeatedly in the different segments, so the SDMA can enable a limited network resources to be effectively utilized. Packets from one terminal to terminals in the same segment are delivered directly and packets to terminals in different segments are delivered via relay terminals. A segmentation algorithm has been proposed that employs the value of vector's dot product to perform segmentation[?].

User terminals

To make the WACNet popular it is necessary to miniaturize and reduce the cost and weight of the user terminals. One of the important reductions to do is to reduce the battery weight. Accordingly, if an adaptive antenna can be implement in the user terminal, a large among of battery power could be saved. Indeed, the adaptive antenna has a higher gain than an omnidirectional antenna, and can therefore reduce the RF power necessary to transmit packets. The adaptive antenna can also suppress co-channel interference.

Microwave signal processing

Adaptive antennas have a great advantage in saving RF power. However, the present adaptive antennas are DBF(*cf. figure 2.1*) antennas, which need the same number of RF high-power or low-noise amplifiers, frequency converters, and D/A or A/D converters.
Therefore, the implementation of adaptive antennas in user terminals had been said to be impractical. This is now possible with the "Microwave signal processing" concept using RF beam forming(*cf. figure 2.2 on the next page*) instead of DBF.
A new beam forming architecture has been proposed to achieve the ultimate reductions in the size, weight, power dissipation, and fabrication cost. The operating principle of this architecture is based upon electromagnetic coupling among array elements. It is called ABF(*cf. figure 2.3 on the following page*) because signal combining is carried out in space, not in circuits. Since this array requires only one RF port to feed, the RF circuit scale is drastically reduced compared with other configurations.

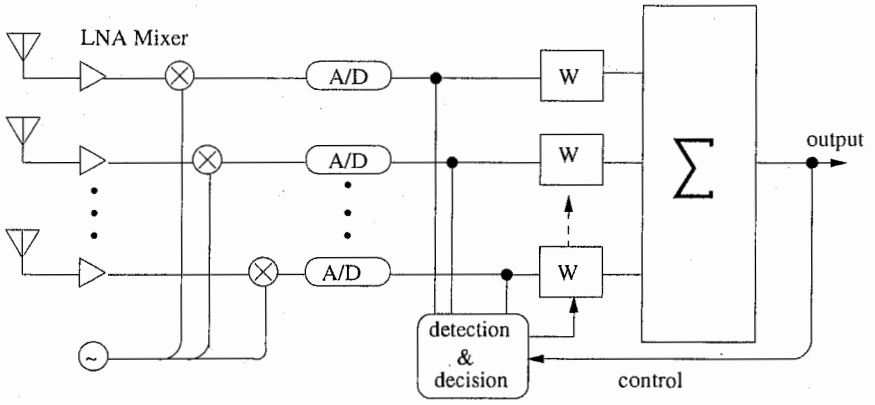


FIGURE 2.1: DBF: Digital Beam forming

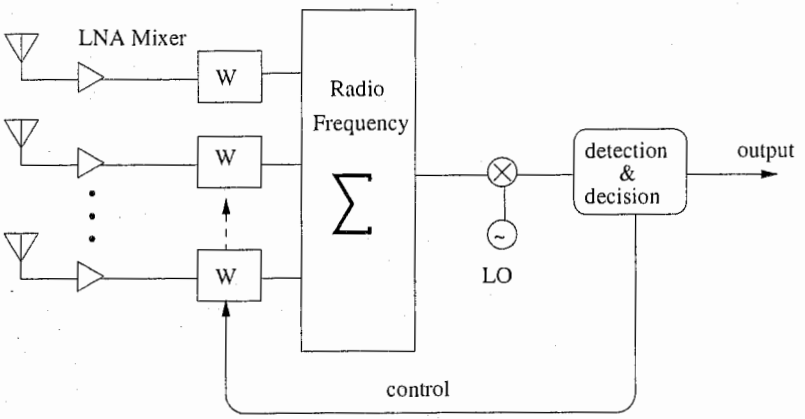


FIGURE 2.2: MBF: Microwave Beam forming

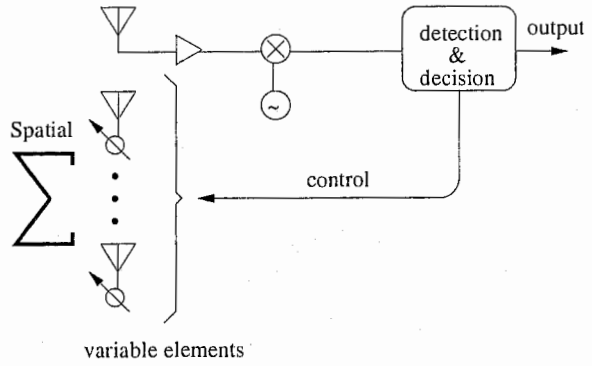


FIGURE 2.3: ABF: Aerial Beam forming

2.2 ESPAR antenna

As a practical hardware realization of the ABF concept, the Electronically Steerable Passive Array Radiator antenna(*cf. figure 2.3 on the preceding page*) has been proposed. The ESPAR antenna[?] consists of $(M+1)$ elements, with $M = 6$ in our configuration. This antenna is a reactively controlled array with only an active radiator, the 0-th element, connecting to the receiver (*figure 2.4*). The M others elements are passives, and the pattern is forming by changing the values of the reactances of these passive radiators.

Compared with classical array antennas, ESPAR antenna has low hardware complexity, low cost, low power consumption, etc. The ESPAR antenna, does not need RF-amplifiers, bandpass filters, A/D converters for each element like in conventional adaptive array[?, ?, ?]. This specific antenna seems to be good candidate for mobile applications and wireless computer network.

In this configuration, only the active radiator signal can be measured, and the 6 others element signals can be not observed. Therefore, to better the pattern, the reactances can be adjusted only with one signal feedback information.

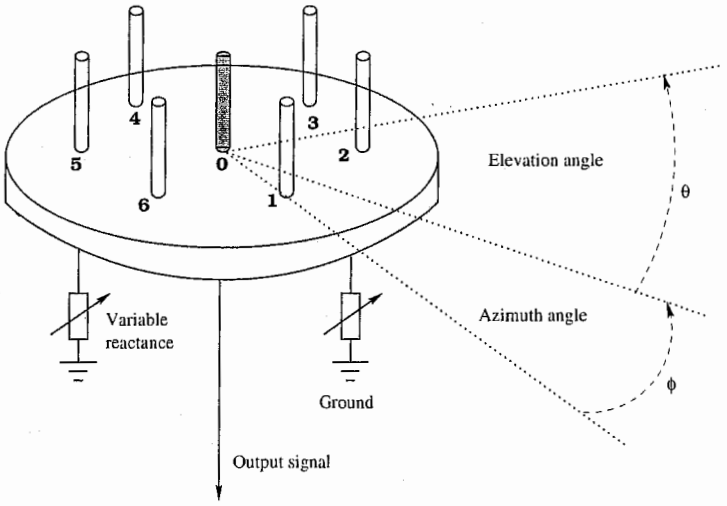


FIGURE 2.4: ESPAR antenna

2.2.1 Formulation of the ESPAR antenna

In this section, only the receive-mode formulation of ESPAR antenna will be tackle.

Notations

First, the notations used for all equations and calculus have to be explained. A scalar or complex value is noted: x, x_n, X, etc. A vector or matrix is noted: $\underline{x}$, $\underline{X}$, etc.

$$\text{Example :} \quad \underline{X} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix}$$

The notation $\underline{X}^T$ is the transpose of the vector or matrix $\underline{X}$.
The constant j is defined by $j \in \mathbb{C}$ and $j^2 = -1$.
$\underline{I_n}$ represents the identity matrix of order n.

Formulation

The $(M+1)$-element ESPAR antenna is a circular array antenna, with the active radiator located in the centre of a circular ground plane. The remaining M elements are passive radiators surrounding the active radiator symmetrically. Each of these M radiators is terminated by a variable reactance.

These reactances will be noted: $\underline{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_m \\ \vdots \\ x_M \end{pmatrix}$

where the variable vector, $\underline{x}$ is called reactance vector, and x_m is the value of the m-th element reactance. In practical uses the reactance x_m must be constrained in certain ranges.

Now, the output of the ESPAR antenna that is function of the M reactances will be formulated. Note that the output of the ESPAR antenna will be consider as the input of the antenna controller.

The RF signal impinging on the antenna is denoted $\underline{s}(t)$ with, $\underline{s}(t) = \begin{pmatrix} s_0(t) \\ \vdots \\ s_m(t) \\ \vdots \\ s_M(t) \end{pmatrix}$

where $s_m(t)$ is the RF signal impinging on the m-th element.

The RF currents on the elements are not independent but mutually coupled with each other, and depend on the value of the reactances. The RF current vector is noted $\underline{i}(t)$, and $\underline{i}(t)^T = [i_0(t), \ldots, i_m(t), \ldots, i_M(t)]$, with the component $i_m(t)$ appearing on the m-th element. Afterward, $\underline{i}(t)$ will be noted $\underline{i}$.
Then, the single-port RF output $y(t)$ of the antenna is given by :

$$y(t) = \underline{i}^T \underline{s}(t) \tag{2.1}$$

In practical uses, the noise must be take into account, the formulation becomes :

$$y(t) = \underline{i}^T \underline{s}(t) + \underline{n}(t) \tag{2.2}$$

where $n(t)$ is an additive white Gaussian noise. But in the following formulation, the noise is not considered.

Now, let give a representation of the current vector as a function of the reactance vector. By using the theorem of reciprocity (1)[?], it can be assumed that the ESPAR antenna transmitting and receiving patterns are the same.

THEOREM 1 (RECIPROCITY THEOREM FOR ANTENNAS)
If an emf[2] *is applied to the terminals of an antenna A and the current measured at the terminals of another antenna B, then an equal current (in both amplitude phase) will be obtained at the terminals of antenna A if the same emf is applied to the terminals of antenna B.*

In transmit-mode (*resp.* receive-mode), the output impedance (*resp.* input impedance) of the central element is not influenced by the mutual coupling of elements. Therefore, the RF voltage of the central element, can be imposed as function of the current appearing on this element :

$$v_0 = V_s - Z_0 i_0 \tag{2.3}$$

[2]electro-mechanic force

where Z_0 is the output impedance, and V_s is the internal source RF voltage.

Then, the RF voltage current can be explained as a function of $\underline{i}$:

$$\underline{v} = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_m \\ \vdots \\ v_M \end{pmatrix} = \begin{pmatrix} V_s - Z_0 i_0 \\ -jx_1 i_1 \\ \vdots \\ -jx_m i_m \\ \vdots \\ -jx_M i_M \end{pmatrix} = V_s \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \underline{X} \cdot \underline{i}. \tag{2.4}$$

where, $\underline{X} = diag[Z_0, jx_1, \ldots, jx_M]$ is called the reactance matrix, and, $v_m = -jx_m i_m$ is the RF voltage imposed on the reactance x_m.

In addition, the RF current $\underline{i}$ and RF voltage $\underline{v}$ have the relationship :

$$\underline{i} = \underline{Y} \cdot \underline{v} \tag{2.5}$$

where $\underline{Y}$ is the admittance matrix, with

$$\underline{Y} = \begin{pmatrix} y_{00} & \cdots & y_{0M} \\ & \ddots & \\ \vdots & y_{kl} & \vdots \\ & & \ddots \\ y_{M0} & \cdots & y_{MM} \end{pmatrix}$$

y_{kl} represents the mutual admittance between the elements k and l ($(k,l) \in (\llbracket 0, M \rrbracket)^2$).

By replacing the voltage in (2.5) by its expression in (2.4), the current becomes :

$$\underline{i} = (\underline{I_{M+1}} + \underline{X} \cdot \underline{Y})^{-1} y_0 \tag{2.6}$$

where y_0 is the first column of $\underline{Y}$.

The number of components[3] of $\underline{Y}$ can be reduced. First, According to the theorem of reciprocity (THEOREM *1*),

$$\forall (k,l) \in (\llbracket 0, M \rrbracket)^2 \quad y_{kl} = y_{lk} \tag{2.7}$$

In addition, the cyclic symmetry of the elements of the ESPAR antenna implies some symmetries in sub-parts of matrix $\underline{Y}$. Finaly, the admittance

[3]unknow coefficients in the matrix

matrix $\underline{Y}$ are determined by only 6 components of the mutual admittances[?], and can be write as follow :

$$\underline{Y} = \begin{pmatrix} y_{00} & y_{10} & & \cdots & & & y_{10} \\ y_{10} & y_{11} & y_{21} & y_{31} & y_{41} & y_{31} & y_{21} \\ & y_{21} & \ddots & \ddots & \ddots & \ddots & y_{31} \\ \vdots & y_{31} & \ddots & \ddots & \ddots & \ddots & y_{41} \\ & y_{41} & \ddots & \ddots & \ddots & \ddots & y_{31} \\ & y_{31} & \ddots & \ddots & \ddots & \ddots & y_{21} \\ y_{10} & y_{21} & y_{31} & y_{41} & y_{31} & y_{21} & y_{11} \end{pmatrix} \tag{2.8}$$

The values of the six components (y_{00},y_{10},y_{11},y_{21},y_{31},y_{41}), depend on the physical structure of the antenna and are constant.

The ESPAR antenna formulation can be summarized, using equation 2.1 and equation 2.7, by :

$$y(t) = ((\underline{I_{M+1}} + \underline{X} \cdot \underline{Y})^{-1} y_0)^T \underline{s}(t) \tag{2.9}$$

Unlike in conventional adaptive array, the signal $\underline{s}(t)$ impinging on the elements of the ESPAR antenna is not measurable. Only the single-port output $\underline{y}(t)$ of the active radiator can be measured.
To control the reactances ($\underline{X}$), $y(t)$ is used as feedback. But $y(t)$ is a non-linear function of $\underline{x}(t)$. Thus, direct application of most of the algorithms of the conventional adaptive array does not apply.
It is interesting to propose a model of the received signal $\underline{s}(t)$, before focusing on the adaptive algorithm.

2.2.2 Signal model

In this section a receive signal model is proposed. In this model there are a total of $Q + 1$ signal. This model can be expressed as :

$$\underline{s}(t) = \sum_{q=0}^{Q} \underline{a}(\theta_q) u_q(t) \tag{2.10}$$

where $\underline{s}(t)$ is a $M + 1$ vector representing the signals impinging on each element. Then, $\underline{a}(\theta)$ is the steering vector of the ESPAR antenna and defined by

$$\underline{a}(\theta) = \begin{pmatrix} 1 \\ e^{j\frac{\pi}{2}\cos(\theta-\phi_1)} \\ e^{j\frac{\pi}{2}\cos(\theta-\phi_2)} \\ \vdots \\ e^{j\frac{\pi}{2}\cos(\theta-\phi_M)} \end{pmatrix}$$

$\phi_m = \frac{2\pi}{M}(m-1)$ $m \in [\![0, M]\!]$, is the position of the m-th element relative to an arbitrary axis.
The angle θ is the DOA relative to the same axis

And, $u_q(t)$[4] represents a signal with DOA[5] θ_q. In this case $\underline{s}(t)$ is supposed to be a superposition of $Q+1$ signals.

2.2.3 Adaptive algorithm

There are several approaches to make adaptive control of the ESPAR antenna. The main problem is the calculation and optimisation of the ESPAR reactance with time constraints. Thus the adaptive algorithm used should converge rapidly. Several algorithms are candidate for ESPAR antenna control. In this section, some algorithm will be briefly enounced, then the algorithm chose to made the experiments will be explain.

Adaptive methods

All of these methods aim at optimising a function or a relation. The problem is to find the six reactances corresponding to the best directivity. Due to the no linearity of the modelling ESPAR antenna output signal (2.9), there are several local minimum.

A first method is to use the CMA[6]. This method is based on the constant modulus of the transmitted signal, thus a error can be generate without using a reference signal, indeed the purpose is to minimize this error. But this method can not distinguish which is the desired signal, only find an optimum weight vector of ESPAR antenna which will give a constant modulus output.

A second method is to use Genetic Algorithms, in order to find the best reactances values. This kind of algorithms use evolution and selection processes, which can be divided in : selection, crossover and mutation. GA usually follow these steps initialisation, selection, mutation and crossover. In the case of the ESPAR antenna the main purpose of using a genetic algorithm

[4] $q = 0, 1, \ldots, Q$
[5] Direction Of Arrival
[6] Constant Modulus Algorithm

is to reduce the set of possible reactances, in a set of reactances including the optimum reactances.

A other method is to use a steepest gradient-based algorithm[?], this algorithm is not the optimum, should an optimum algorithm exists. This method is explain in the following.

Gradient-based algorithm

In this section a gradient-based adaptive algorithm for ESPAR antenna will be described.
In the conventional steepest gradient algorithm, the function to be minimize or maximize is the cost function, here it will be called J_n.
A gradient vector of J_n is defined as:

$$\nabla_{J_n} \triangleq \frac{\partial J_n}{\partial \underline{x}} \triangleq \begin{pmatrix} \frac{\partial J_n}{\partial x_1} \\ \frac{\partial J_n}{\partial x_2} \\ \vdots \\ \frac{\partial J_n}{\partial x_n} \end{pmatrix} \tag{2.11}$$

where $J_n = J(x_1, x_2, \ldots, x_n)$, $\frac{\partial J_n}{\partial x_n}$ denotes the derivative with respect to $\underline{x}$.
To avoid any confusion, it is important to note that in this definition ∇_{J_n} is a vector and not a scalar.

With the steepest gradient algorithm we proceed as follows:

- $\underline{x} = \underline{x_{init}}$ this initial value is taken arbitrarily or is given by theory considerations.
- **for** (n=1; n<=N_{conv}; n++)
  {
  $\nabla_{J_n} \leftarrow \frac{\partial J_n}{\partial \underline{x}}$
  update $\underline{x}$ by making a change in direction of ∇_{J_n} :
  $\underline{x}(n+1) = \underline{x}(n) \pm \mu \nabla_{J_n}$
  }

Here, n denotes the time, N_{conv} is the number of iterations which assures the convergence and $\mu \in \mathbb{R}^+$ controls the convergence speed.

An estimate value of the gradient vector ∇_{J_n} (*cf. equation 2.11*) may be obtained by using *finite-difference approximations of derivatives* (*cf. Appendix 1*).
Here, the first-order partial derivative $\frac{\partial J_n}{\partial x_i}$ can be use. The formulation will be:

$$\frac{\partial J_n}{\partial x_m} \approx \frac{Js_n(x_m + \Delta x_m) - Js_n(x_m)}{\Delta x_m} \tag{2.12}$$

where with this special notation, $Js_n(x_m + \Delta x_m) = J_n(x_1, x_2, \ldots, x_m + \Delta x_m, \ldots, x_M)$ and $Js_n(x_m) = J_n(x_1, x_2, \ldots, x_m, \ldots, x_M)$, and $m = 1, 2, \ldots, M$. Δx_m corresponds to the perturbation size.

By using this estimate value of the gradient vector, the gradient vector can be write like this :

$$\nabla_{J_n} \triangleq \frac{\partial J_n}{\partial \underline{x}} \approx \begin{pmatrix} \frac{J_n(x_1+Deltax_1,x_2,\ldots,x_M)}{\Delta x_1} \\ \frac{J_n(x_1,x_2+\Delta x_2,\ldots,x_M)}{\Delta x_2} \\ \vdots \\ \frac{J_n(x_1,x_2,\ldots,x_m+\Delta x_m,\ldots,x_M)}{\Delta x_m} \\ \vdots \\ \frac{J_n(x_1,x_2,\ldots,x_M+\Delta x_M)}{\Delta x_M} \end{pmatrix} - \begin{pmatrix} \frac{J_n(x_1,x_2,\ldots,x_M)}{\Delta x_1} \\ \frac{J_n(x_1,x_2,\ldots,x_M)}{\Delta x_2} \\ \vdots \\ \frac{J_n(x_1,x_2,\ldots,x_M)}{\Delta x_m} \\ \vdots \\ \frac{J_n(x_1,x_2,\ldots,x_M)}{\Delta x_M} \end{pmatrix} \tag{2.13}$$

If each component of gradient vector, has considered having the same perturbation size, *i.e* $\Delta x_1 = \Delta x_2 = \ldots = \Delta x_M = \Delta x$, the equation 2.13 can be write :

$$\nabla_{J_n} \approx \underline{J_n} - \underline{J_n}^{(0)} \tag{2.14}$$

with $\underline{J_n}$ equal to the part before minus of equation 2.13 and $\underline{J_n}^{(0)}$ equal to the part after minus of equation 2.13.

In some cases, it is useful to consider some normalizations of the gradient vector. Here only 2 cases of normalization have been considered and proposed, of course there are several approaches of normalization.

1. normalization of the cost function J_n. According to the case in the equation 2.14, the normalization can be written as follow :

$$(\nabla_{J_n})_{normalize} = \frac{\underline{J_n} - \underline{J_n}^{(0)}}{\|\underline{J_n}\|_2} \tag{2.15}$$

with

$$\|\underline{J_n}\|_2 = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(J_n(x_1, x_2, \ldots, x_i + \Delta x, \ldots, x_M))^2}$$

2. normalization of the gradient vector ∇_{J_n}

$$(\nabla_{J_n})_{normalize} = \frac{\partial J_n}{\partial \underline{x}} \times \frac{1}{\|\frac{\partial J_n}{\partial \underline{x}}\|_2} \tag{2.16}$$

with

$$\|\frac{\partial J_n}{\partial \underline{x}}\|_2 = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(\frac{\partial J_n}{\partial x_i})^2}$$

Now let apply the steepest gradient algorithm to the adaptive control of the reactances of ESPAR antenna. First, it is important to remind, the presence of an intractable matrix inverse in the representation of $y(t)$ (*cf. eq. 2.9*) and the fact that the signal vector impinging on the passive elements of the antenna can not be observed.
In the case of ESPAR antenna, the only observable signal is $y(t)$, the cost function will be chosen according to this information.
For each purpose of using the ESPAR antenna, a suitable cost function will be adopted.

In this paper two cost functions for two different purposes will be shown. The first purpose[?], makes the ESPAR antenna form a beam in the direction of desired signal, and form null(s) in direction(s) of interference signal(s) according to the degrees of freedom of the ESPAR antenna. In this case a reference signal $r(t)$ is used. This signal is known to both the transmitter and receiver.
In the gradient-based adaptive algorithm, a *cross-correlation coefficient* is use as cost function. By introducing the parameter P as the dimension of $\underline{r}(n)$ and $\underline{y}(n)$ which is the vector of time samples of $r(t)$ and $y(t)$, the *cross-correlation coefficient* of $\underline{r}(n)$ and $\underline{y}(n)$ is defined as:

$$\nabla_{J_n} = \rho_n = \frac{|\underline{y}^H(n)\underline{r}(n)|}{\sqrt{\underline{y}^H(n)\underline{y}(n)}\sqrt{\underline{r}^H(n)\underline{r}(n)}} \tag{2.17}$$

where, $\underline{A}^H$ denotes the complex conjugate transpose of $\underline{A}$.

The second purpose, is to make ESPAR antenna form a beam in the direction of a desired signal. This means that only the desired signal impinges on the elements. In this case the cost function used, is the power[7] of

[7]or inverse power

the ouput signal $y(t)$. Using the same parameter P than above, the expression of the cost function is:

$$\nabla_{J_n} = Power(y(t)) = \frac{1}{P}\sum_{i=1}^{P}|y_i(n)|^2 \tag{2.18}$$

with $\underline{y}(n) = [y_1(n), y_2(n), \ldots, y_P(n)]^T$

In this report, the main purpose is to form the antenna sector pattern. Thus the cost function use is the power. In the next chapter, we will show simulation of the algorithm using *power* as *cost function*, also experiments conditions will be described.

Chapter 3

Adaptive control of ESPAR antenna

In the previous chapter, the ESPAR antenna and an adaptive algorithm have been introduced. Now let consider adaptive control of the ESPAR antenna. In the first hand, with MatLab software some simulations will be made to compute the sector beam pattern, also some results of these simulations will be shown. in the other hand, two experiments conditions will be described. The experiments aim at forming the sector beam pattern.

3.1 Simulation with MatLab

3.1.1 The purpose

The purpose of this work was to confirm that the algorithm could work. The matlab simulation was based on the steepest gradient algorithm described in the previous chapter. This algorithm has been slightly change in order to fit with this purpose :
To show that the ESPAR antenna can adaptively steer beam in one direction for one signal. And also, to trace the ESPAR antenna sector pattern.
This means that from an initial set of reactances, the algorithm should find a set of reactances which maximize the power in the sense of the steepest gradient method. The *convergenge* is assumed when the power goes towards a maximum value, i.e. the reactances become constant.

Firstly, the algorithm has been write in order to form beam with *power* as cost function criterion, and *inverse power* as cost function criterion. In

the first case, it aimed at forming a beam in the DoA[1] of the signal whereas in the second case it aimed at forming a null in the DoA of signal.
At the following, only the beam forming has been considered, although the programs made allow to test the algorithm generically.

The matlab simulation consist of several programs to do the simulation in these steps :

Step 1 Input the parameters :

- **M** : number of passive elements
- **N** : number of iterations
- **P** : size of blocks iterations for each reactances (*cf. equation 2.17 on page 19 and 2.18 on page 20 and figure 3.1 on the facing page*)
- μ : step size parameter control
- Δx : derivative step size according to equation 2.12 on page 18
- $x1, \ldots, x6$: initial reactances values

Step 2 Generate test signal :

- desired signal
- interferences signal (not use)
- noise

Step 3 Compute the adaptive algorithm to get the final reactances

Step 4 Draw graphs (convergence curve, antenna pattern, reactances variation)

According to figure 3.1 on the facing page, the desired signal vector will have $N \times (M+1) \times P$ values, that means a total of $N \times (M+1) \times P$ symbols required for N iterations. The algorithm proposed for the adaptive reactances search is :

input M,μ,**vector** Δx

create X **dimension** M
create ∇J_n**dimension** M

[1]Direction of Arrival

```
create cost dimension M

let n=1
initialize X[0],...,X[M-1]
calculate cost_0=cost function

while (n≤N)
{
    for (m=0;m<M;m++)
    {
        X[m] = X[m] + Δx[m]
        calculate cost[m]=cost function
        ∇J_n[m]=cost[m]-cost_0
        X[m] = X[m] - Δx[m]
    }
    for (m=0;m<M;m++)
    {
        X[m] = X[m] + μ*∇J_n[m]
        constraint X[m] in range [minΩ; maxΩ]
    }
}
```

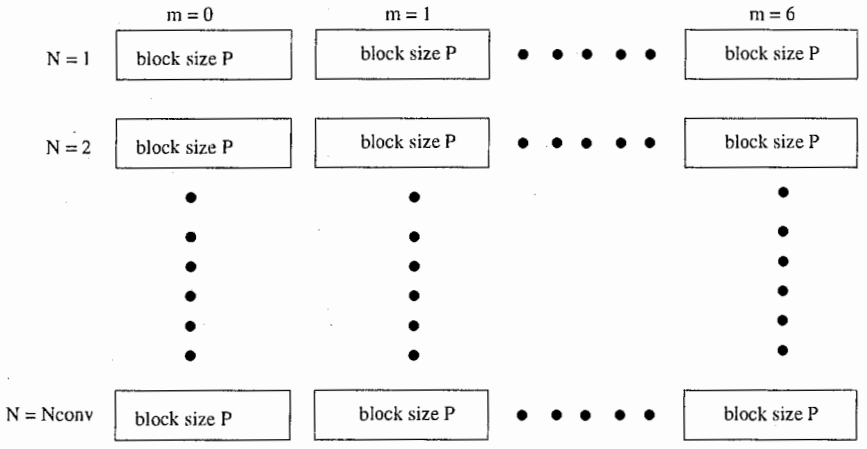


FIGURE 3.1: Iteration block diagram

3.1.2 Simulation results

Using the algorithm explained before, the sector pattern of the antenna has been computed. The sector pattern means, for one impinging signal, one beam will be formed in each direction of the sector angle[2]. Here, the reactances values will be compute only for 12 angles with 30° of spacing out.

It is important to locate here the simulation condition. The desired signal is simulated by a BPSK signal vector, the noise is not considered, and the range of reactances are constrained in $x \in [-300\Omega; 300\Omega]$. The initial reactances values is set to zero in the MatLab simulation

The reactances values for the 12 angles can be see in the table 3.1. These results have been obtained for these following initials parameters : $\mathbf{M} = 6$ (*i.e 6 elements*), $\mathbf{N} = 500$, $\mathbf{P} = 100$, $\mu = 2$, $\Delta x = 20$.

Due to the symmetry of the ESPAR antenna structure, in the data results,*table 3.1 on the next page*, there is a symmetry for the reactances around 0Ω. In the figure 3.3 on page 27 the antenna pattern has been traced for 0°,30°,60°,90°, the others values can be deduced by symmetry.
The table 3.2 on page 26 and the curve 3.2 on page 27 show the behavior of the algorithm. In the table 3.2 on page 26, the reactances do not really change after 500 iterations. In the *convergence* curve it is also possible to see that the power go towards a constant value. This means the algorithm converges, this method can be used to adaptively find the reactances.

It the table 3.1 it can be see that for most of the time the reactances values reach the extremities of the constrained range. A modification of the algorithm (section 3.1.1 on page 22) is proposed. This modification will be use during the experiments. It consists in adding a constraint in the adaptive look up like this :

input M,μ,**vector** Δx

create X **dimension** M
create ∇J_n**dimension** M
create cost **dimension** M

let n=1
initialize X[0],...,X[M-1]
calculate cost$_0$=*cost function*

[2]from 0° to 360°

	Angles of desired signal					
	0°	30°	60°	90°	120°	150°
x_1	-246.0	-300	202.3	154.1	90.1	39.4
x_2	202.3	-300	-246.0	-300	202.3	154.1
x_3	90.1	154.1	202.3	-300	-246.0	-300
x_4	-2.4	39.4	90.1	154.1	202.3	-300
x_5	90.1	39.4	-2.4	39.4	90.1	154.1
x_6	202.3	154.1	90.1	39.4	-2.4	39.4
	Angles of desired signal					
	180°	210°	240°	270°	300°	330°
x_1	-2.4	39.4	90.1	154.1	202.3	-300
x_2	90.1	39.4	-2.4	39.4	90.1	154.1
x_3	202.3	154.1	90.1	39.4	-2.4	39.4
x_4	-246.0	-300	202.3	154.1	90.1	39.4
x_5	202.3	-300	-246.0	-300	202.3	154.1
x_6	90.1	154.1	202.3	-300	-246.0	-300

TABLE 3.1: Matlab simulation results, reactances values for sector pattern

```
while (n≤N)
{
    for (m=0;m<M;m++)
    {
        if (X[m] + Δx_m > max)
        {
          X[m] = X[m] - Δx[m]
          calculate cost[m]=cost function
          ∇J_n[m]=-(cost[m]-cost_0)
          X[m] = X[m] + Δx[m]
        }
        else
        {
          X[m] = X[m] + Δx[m]
          calculate cost[m]=cost function
          ∇J_n[m]=cost[m]-cost_0
          X[m] = X[m] - Δx[m]
        }
    }
```

	Number of iterations		
	500	1000	2000
x_1	-246.0	-251.4	252.6
x_2	202.3	261.4	298.5
x_3	90.1	112.5	124.7
x_4	-2.4	-5.7	-7.5
x_5	90.1	112.5	124.7
x_6	202.3	261.4	298.5

TABLE 3.2: Matlab simulation results, reactances versus the number of iterations

```
    for (m=0;m<M;m++)
    {
        X[m] = X[m] + μ*∇J_n[m]
        constraint X[m] in range [minΩ; maxΩ]
    }
}
```

This modification, will change the behavior of the reactances when an extremity of the constraint reactance range is reached. In the case of the ESPAR antenna the optimum reactances are not unique. By going all the time in the same direction for each reactance an optimum solution can be forgotten. If the ESPAR antenna solution set is considered as a 6 dimensions set[3], from one point to one solution[4], the path until this solution does not follow all the time the same direction for all the axes.

[3] 6 axes

[4] extremum

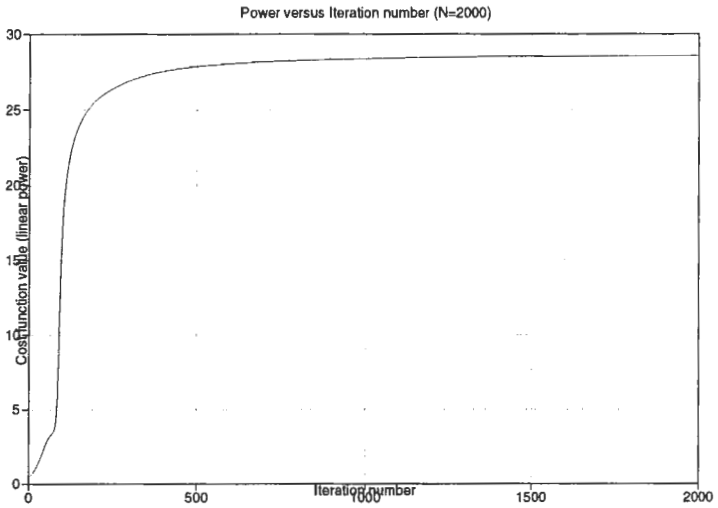


FIGURE 3.2: Matlab simulation, power as cost function versus number of iterations

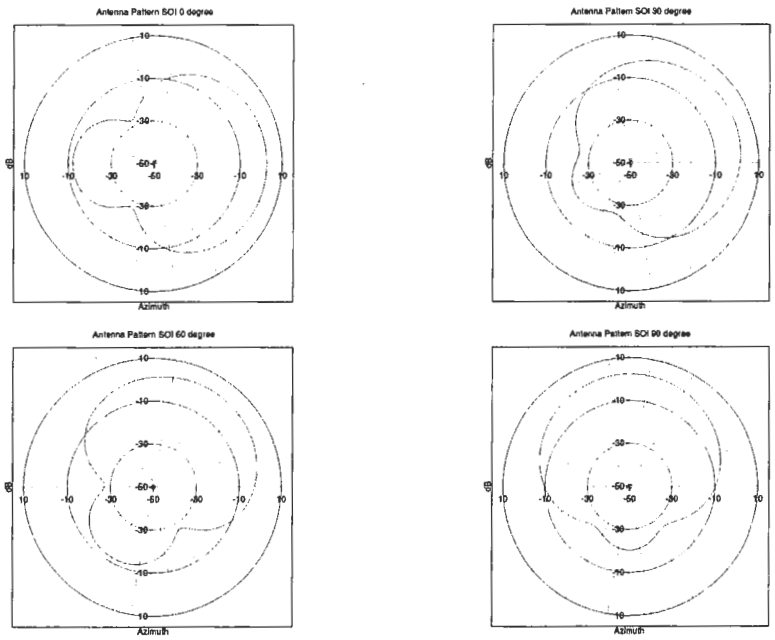


FIGURE 3.3: Matlab simulation, ESPAR antenna patterns : first quadrant (polar)

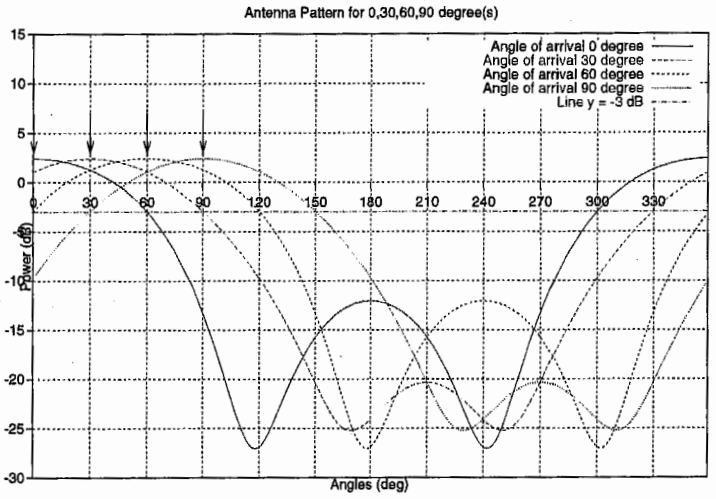


FIGURE 3.4: Matlab simulation, ESPAR antenna patterns : first quadrant (cartesian)

3.2 Adaptive control of ESPAR antenna using DSP (solution 1)

In this part an adaptive control of ESPAR antenna using DSP controller, will be presented. The DSP solution for adaptive control, allows to have real time experiments and allows to test many algorithms. In this part of the work, the purpose was to experimentally form the sector beam pattern, for 12 angles between 0° and 180°.
In the experiment case, the reactance values cannot be directly set. Thus, in this section the term "*reactance(s)*" refers to the "*voltage(s)*", which can be adjusted.

3.2.1 Experiment conditions

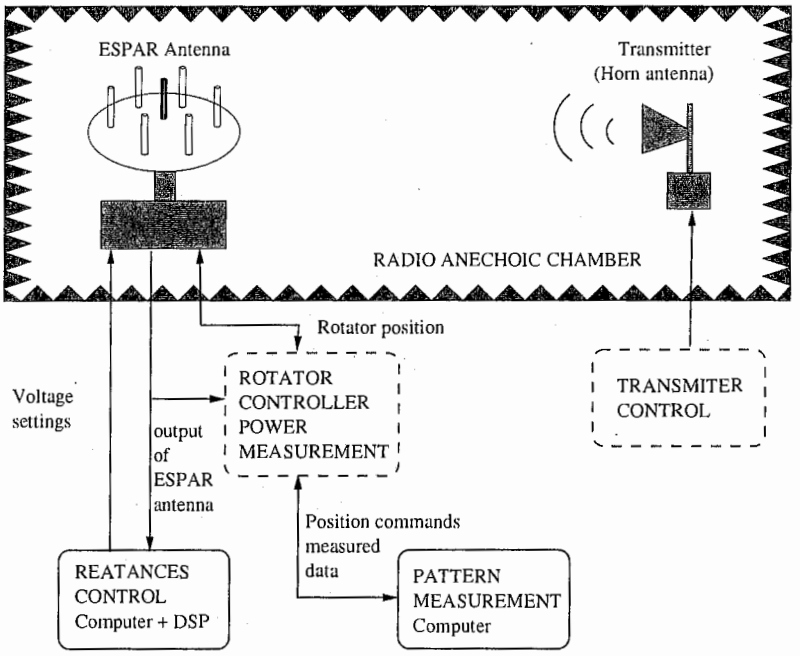


FIGURE 3.5: Experiment condition scheme (DSP solution)

Hardware conditions

FIGURE 3.6: ESPAR antenna

The ESPAR antenna used (*cf. figure 3.6*), is composed of 6 passive monopoles around a central active monopole. The central monopole is coupled with the 6 others elements. The reactances of the passive monopoles are adjusted by directly adding some loads at their extremities or indirectly changing the voltage values at their extremities.
The reactance values for each element can not be directly set, a control voltage has been made for this. Thus, there is a relationship between the voltage set and the reactance value. This relation depends on the components use to make the varactances, and the coupling reactance between the elements. The voltage value varies between -0.5 Volt and 20 Volt. In fact a D/A[5] is used to set the voltage value, this D/A is coded with 12 bits and then can accept values between -2048 and 2047, this range is preferred from 0 to 4095 because of symmetry. It can be noticed that with this range of values, a simple search algorithm should deal with $(2^{12})^6 = 2^{72} \approx 4.7 \times 10^{21}$ possibilities of reactances, it is obvious that a better method must be used for real time system. Finally the relationship between voltage and reactance is :

$$x = -0.0217v - 49.21 \tag{3.1}$$

where x is the reactance and v the voltage, this relation assures that $x \in [-93.63\Omega; -4.77\Omega]$ with $v \in [\![-2048; 2048]\!]$

[5]Digital to Analog converter

Note that this relation is shift by an offset, that prevents the voltage from having a symmetry around 0 as it is shown in the simulation results (*cf. table 3.1 on page 25*).

The experiments has been made in an anechoic room (*cf. figure 3.5 on page 29*). The antenna was placed on a rotating platform control by an independent system. To control the reactance values, a Computer adding with a DSP board has been used. The DSP board and computer features are as follow :

	Device features	
	Computer	DSP Board
Manufacturer	none	Spectrum Signal Processing
CPU(s)	Pentium III	TMS320C6701 (x 2)
CPU Manufacturer	Intel	Texas Instrument
CPU(s) speed	800MHz	167 MHz
Memory	256 Mo	1M-Bit on each CPU*

The DSP board used has some specific features, for instance in addition to the DSP CPU memory* the board has some dedicated memories (*figure 3.7 on the following page*). It is connect to the computer motherboard by PCI bus. Also, a local PCI bus on Daytona connects the processor nodes, system control operations, and the PMC site together. The DEC 21153 chip provides the PCI-to-PCI bridge from this local PCI bus to the external host PCI bus. Spectrum's Hurricane chip provides a high performance bridge between the PCI bus and DSP processors. Each processing node is connected to the local PCI bus through a Hurricane chip. A Hurricane chip is also used as the interface between the 'C6x Host Port Interface bus' and test bus controller system control operations, and the local PCI bus.
The others features are listed here:

- Full-width PCI board consists of two C6701 floating-point processors processing nodes[6].
- 128K x 32-bit of synchronous SRAM per processing node
- 4M x 32-bit of SDRAM per processing node
- 8K x 32-bit of dual-port RAM supporting low-latency data exchange between processing nodes

[6]node A and node B

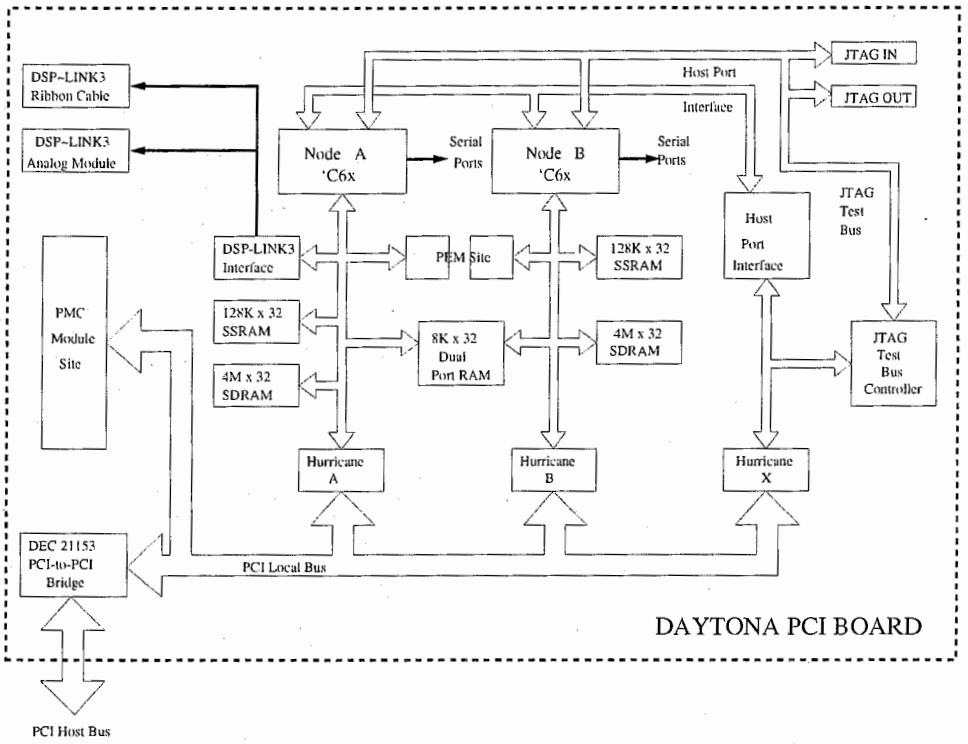


FIGURE 3.7: Daytona Block Diagram

- Host PCI interface using the DEC21153 PCI-to-PCI bridge chip
- 132 Mbytes/s PMC module site. A PMC site is accessible from both the local PCI bus and host PCI bus. The PMC site allows a PMC (PCI Mezzanine Connector) card to be installed in order to add I/O, memory, or other capabilities to the Daytona.
- Convenient paged memory access to the 'C6x address space through the 'C6x Host Port Interface (HPI)
- External and on-board JTAG debugging support
- PEM (Processor Expansion Module) sites
- DSP LINK3 I/O interface, provided for on-board and external I/O boards and, controlled via processing node A.

Software conditions

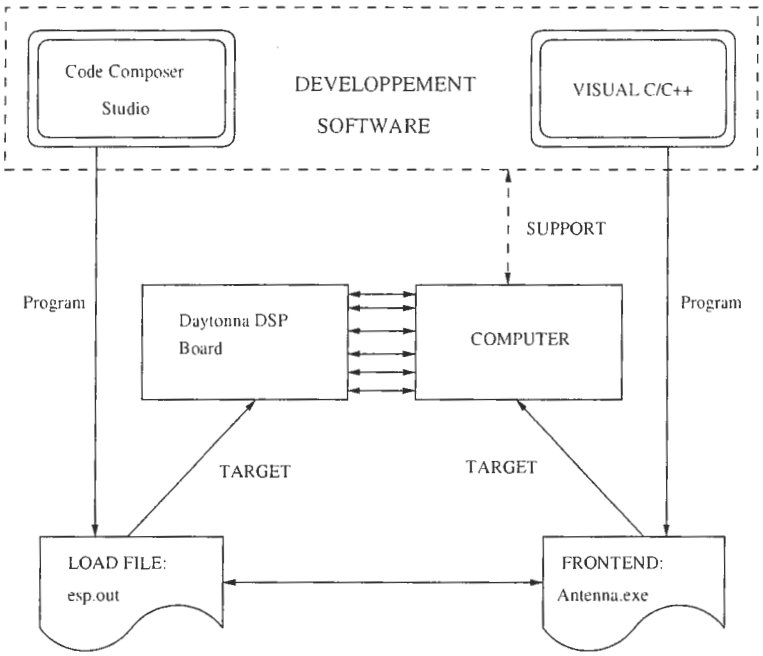


FIGURE 3.8: Software organisation

The ESPAR controller programs(cf. figure 3.10 on page 37) have been made by Mr. Kiyoshi Kobayashi of NTT-ATC[7]. The program scheme and the front-end screenshot can be seen respectively in the figure 3.8 and figure 3.10 on page 37.
In this part the purpose was to study by *reverse engineering* the software source code, in order to make the antenna form beam.
Indeed, the purpose of the software made by NTT was to experiment the adaptive algorithm that makes the ESPAR antenna steer its beam and nulls automatically, presented in section 2.2.3 on page 17 and paper [?].

Our purpose, was to modify these programs in order to make the ESPAR antenna form one beam in the condition presented in scheme 3.5 on page 29. The main problem was to show that ESPAR antenna can form one beam to an incoming signal, and then for all direction of the sector. Actually the ESPAR antenna has been not ever test in multi-signals conditions. The experiments protocol and theory necessary to make experiments in multi-path

[7]Nippon Telegraph and Telephone - Advanced Technology Corporation

and multi-signals condition, are actually studied by Mr. Atsuya Andō.

First, let describe the program scheme (*3.8 on the preceding page and 3.9*).

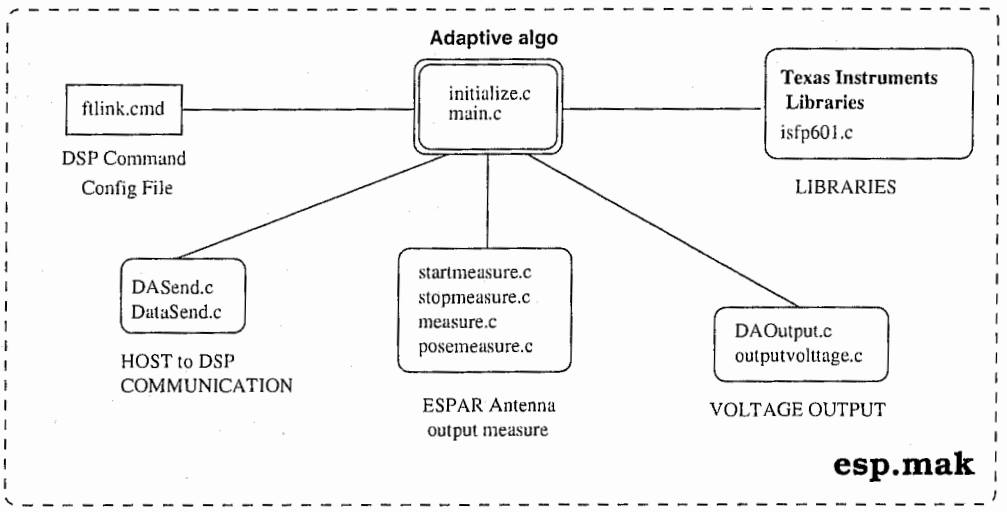


FIGURE 3.9: DSP programs block diagram (project file : esp.mak)

The program code has been split in two parts: one part, made in C, to be use on the DSP and the other, made in C++, to be use on the *host* (PC). When "*antenna.exe*" is executed, it loads the file "*esp.out*" in the DSP memory. Then, a communication between the *host* and the *DSP board* allows to control parameters and get values, ESPAR antenna measure output signal, *cost function* values, reactances values, ...
The main DSP program is a finite state machine, which waits for the *host* commands, get the ESPAR antenna Q/I[8] output signals, put them in a FIFO[9], and adaptively compute the *voltages* values.
For instance, for 1 iteration, and 1 elements change, it has to write in the FIFO, read the value of the top of the FIFO, compute the new voltages, and make a D/A output. So the D/A converter is used to set the voltages and the A/D converter is used to get the I and Q channels values.

To modify the program (*figure 3.8 on the preceding page*), only the DSP part (*figure 3.9*) has to be modified. The modification made was to change the computing of the cost function in the DSP algorithm part.

[8]Quadrature and In phase
[9]First Input, First Output

The two files to change was "*measure.c*" and "*main.c*". In the original code, the cost function used is the same presented in the equation 2.17 on page 19, so the section program is like this :

```
/* ------------------------------------------------------------------- */
for(i = 0; i < length; i++){

        /* get Q and I values of ESPAR output signal */
        y_data[i].q = -((float)((buffer_y[i] & 0xffff) >> 4) - 2048);
        y_data[i].i = -((float)((buffer_y[i] & 0xfff00000) >> 20) - 2048);

        /* get Q and I values of reference signal */
        r_data[i].q = (float)((buffer_r[i] & 0xffff) >> 4);
        r_data[i].i = (float)(buffer_r[i] >> 20);

}

/* calculate cost function */
for (p = 0; p < length; p++){

        a += (y_data[p].i * r_data[p].i + y_data[p].q * r_data[p].q);
        b += (y_data[p].i * r_data[p].q - y_data[p].q * r_data[p].i);
        y += (y_data[p].i * y_data[p].i + y_data[p].q * y_data[p].q);
        z += (r_data[p].i * r_data[p].i + r_data[p].q * r_data[p].q);

}

sqrt_x = sqrtf((a * a) + (b * b)) / datalen;
sqrt_y = sqrtf(y / datalen);
sqrt_z = sqrtf(z / datalen);

mecorr = (sqrt_x / (sqrt_y * sqrt_z));

/* send cost function */
sendMeasure->coeffData = mecorr;
/* ------------------------------------------------------------------- */
```

To fit to our purpose, i.e use the cost function in equation 2.18, the section program has been modified like this :

```
/* ------------------------------------------------------------------- */
```

```
for(i = 0; i < length; i++){
        /* get Q and I values of ESPAR output signal */
        y_data[i].q = -((float)((buffer_y[i] & 0xffff) >> 4) - 2048);
        y_data[i].i = -((float)((buffer_y[i] & 0xfff00000) >> 20) - 2048);

}

/* calculate cost function */
for (p = 0; p < length; p++){

        a += (y_data[p].i * y_data[p].i + y_data[p].q * y_data[p].q);

}

mecorr = a/((float)(param.sample<<SHIFT_COEF));
sendMeasure->coeffData = mecorr;
/* ------------------------------------------------------------------------ */
```

In the modification, we introduced a new parameter: SHIFT_COEF. At the beginning, the purpose of this parameter was to limit the cost function range value. Because compared in the *cross correlation* as cost function case, the value of the cost function in the case of *power* is very high, so the extrema values of the voltages[10] are always reach. So the algorithm does not work as expected. .
After, we considered a normalization (*cf. equations 2.15 on page 18 and equation 2.16 on page 19*) of the cost function in order to obtain better result. In this case, the significance of the SHIFT_COEF parameter is to proportionately shift the values in an other range. Also this range is not important, the most important was the behavior[11] of the algorithm.
Some Experiment results, can be found in chapter 4 on page 43.

[10] -2048 and 2047
[11] convergence, speed of convergence,...

FIGURE 3.10: Frontend of the DSP adaptive ESPAR controller software

3.3 Adaptive control of ESPAR antenna (solution 2)

The ESPAR antenna is very attractive for industrial. ATR works in collaboration with several companies, in particular, on the ESPAR antenna project, ATR works with Antenna Gigen corporation. Antenna Gigen products the ESPAR Antenna, so they work in collaboration with ACR's researchers. The purpose of the work is the same :
To make the ESPAR antenna steer beam for several angles and for one signal. And also, to trace the ESPAR antenna sector pattern in order to confirm the behavior of the Antenna.

To do this, an homogeneous and similar work condition is needed in both part, ATR and Antenna Gigen. ATR had to find the algorithms, and to make the ESPAR controller program and software.

3.3.1 Experiment conditions

The experiments has been made also in an anechoic room (*cf. figure 3.11 on the facing page*). The antenna was placed on a rotating platform control by an independent system.
A Computer adding with a GPIB board and a D/A board is used to control, the voltages settings and power measures, with this solution we can also control the rotation of the antenna with the same computer.
A RF[12] Network Analyzer is used to measure the power. In fact the RF-NA can provide the ratio value between two of its ports, it can also generate continuous wave at different frequencies.

Hardware conditions

The Hardware used is described as follow:

Computer : Intel Pentium III 500MHz, 256 Mo of RAM

GPIB Board : National Instruments 183617G-01. The GPIB[13] allows communicating with devices supporting the American standard IEEE488.1, and also, for some new board the American standard IEEE488.2. The board can control up to 16 devices in the same time. This board it is used to control the Network Analyzer and the antenna rotator.

[12]Radio Frequency
[13]General Purpose Interface Bus

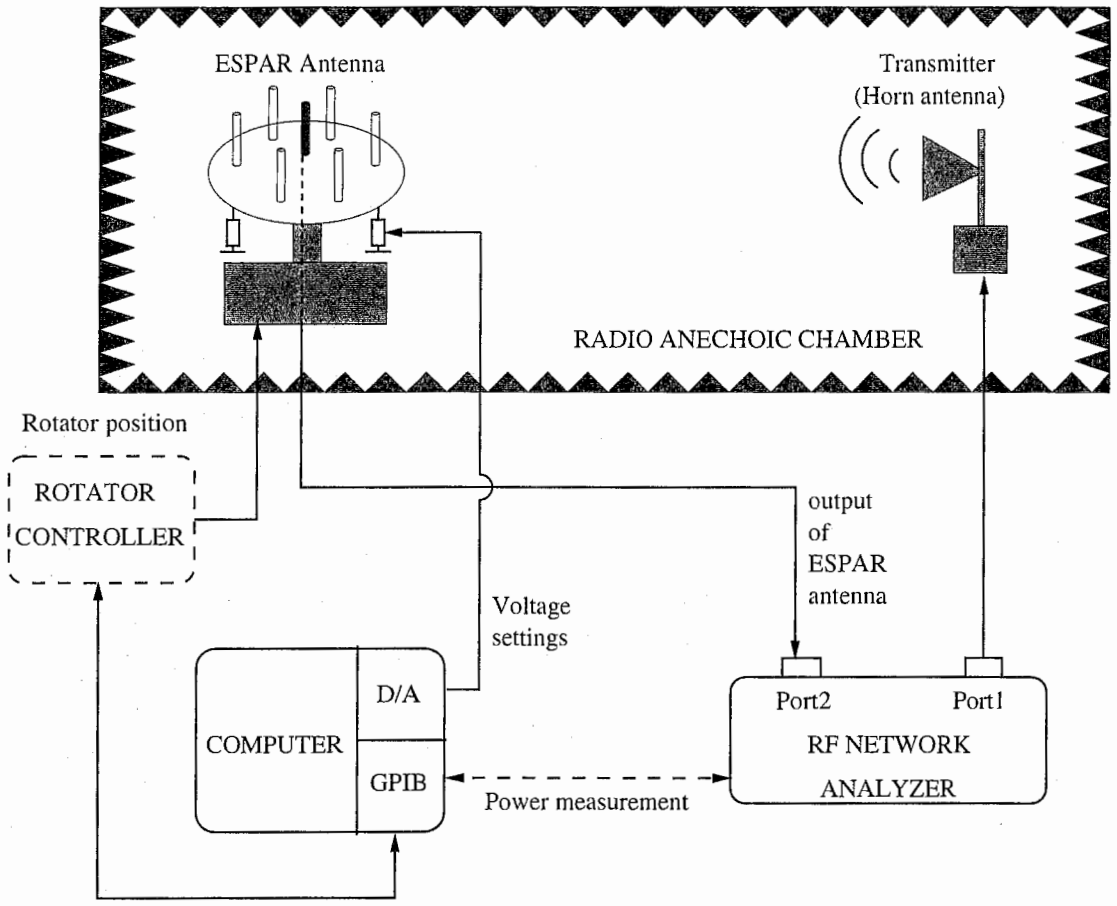


FIGURE 3.11: Experiment condition scheme (2^{nd} solution)

D/A Board : CONTEC DA12-8L(PC). It is a digital to analog converter. It features is : 8 channels, 12 bits converter, 3 modes of voltages output ($\pm$10V,$\pm$5V and 0V$\sim$10V). The ESPAR antenna voltages varies between 0.5V and 20V, so a DC converter is added at the output of the DA board used in 0V$\sim$10V mode.

Network Analyzer : Agilent E8358A. Allows measuring the ratio between the ESPAR antenna output and the transmitter, so the power of received signal.

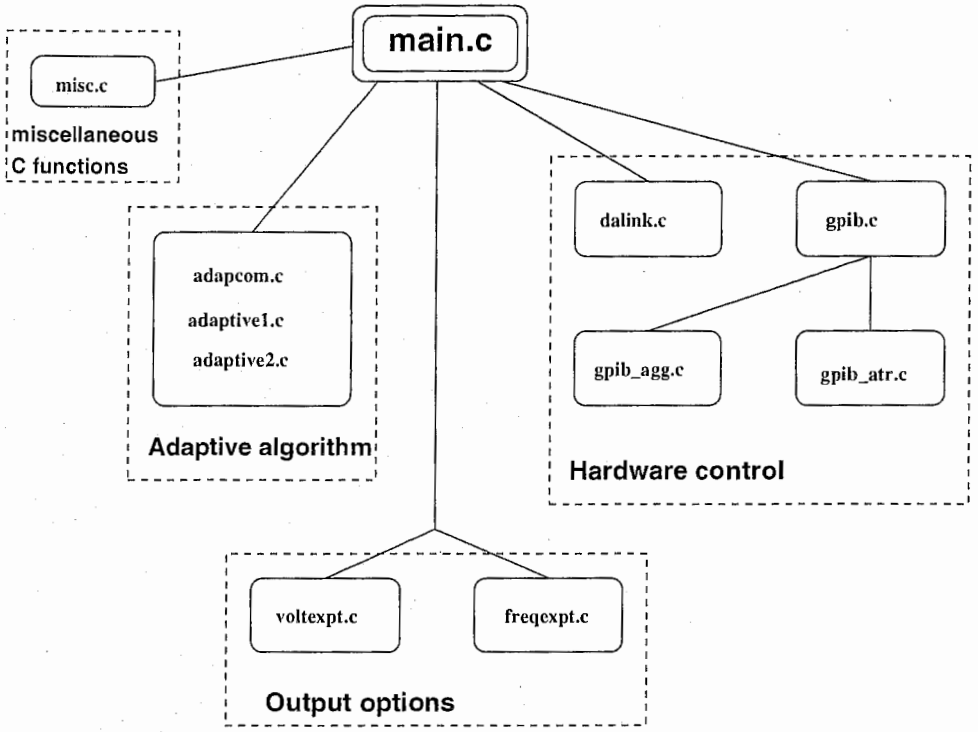


FIGURE 3.12: Software block diagram

Software conditions

To achieve the software control, several API is used. The GPIB board came with API drivers and libraries for several programming languages. A specific API made by CONTEC was used to drive the D/A board, this API is a common and generic API, so the program made is assumed to work on almost all CONTEC D/A board.
The programming language is the C with Microsoft Visual Studio on Microsoft Windows 98 OS. The IEEE488.1 is a simple ASCII[14] communication protocols, a string of commands is sent to the target device, and the target device sent back some data in same format.
The programs scheme made are described in figure 3.12. The executable file is a DOS command to be used as UNIX/Linux like commands. We made this program to works at both, Antenna Gigen environment and ATR environment. Indeed, Antenna Gigen and ATR does not use the same Network

[14]7 bits American format

Analyzer, as the NA used depend of the commands, the specific "GPIB control" part has been split in two parts.
The "main.c" program only consists in analyzing the input commands parameter, initializing the devices, running functions, closing the devices.

In this solution, to synchronize the different boards, a delay time is used between the modifications of the voltages and the measure of the power. Indeed, the voltages modification takes time before having "measurable" effect. Also that depends on the computer and OS features. So, this delay is a parametric delay.
In the next chapter, some results for the both experiment condition, will be presented

Chapter 4

Experiment results and discussion

In the previous chapter, the experiments condition and configuration have been explained, now, some experiments[?] results will be shown and comments. This chapter will be composed of three parts. In the two first parts, some experiment condition and results will be presented and commented for the "*solution*[1] *1*" and the "*solution*[2] *2*". Then a general comment will be made upon these experimental results.

4.1 Experimental results using DSP

The main object of these experiments was to make the antenna form beam, and find the better parameters[3] which allow to obtain the better results. These fundamental experiments aimed at having some data about the ESPAR antenna behaviour.

The experiment shows here is done with normalize cost function, as presented in equation 2.16 on page 19. Also the initial parameters were as follow:

- **P** = 240
- $\Delta x = 40$
- $\mu = 300$
- **Initial voltages** $v_1 = v_2 = \ldots = v_6 = -1000$

[1]Using a DSP
[2]Network analyser and D/A converter
[3]P,N,μ,Δx

In the graphs shown in figure 4.1 on the next page, the beam forming can be observed with the iteration increasing. These graphs show that the algorithm goes towards a beam forming value of the reactances. Also that shows us, that in practical use the number of iterations is very important. Thus, the first pattern (N=10) shows that the power output for the 0^o degree direction is very low, until (N=1000) where the power output is higher, but not the highest. In this experiment, the difference of gain power is about 7dB.
Also by seeing the convergence curve in figure 4.2 on page 46, it can be seen that the curve increases but does not reach a constant range.
So two things can be said. First the difference power, for the desired angle, between the first iteration and the last one, depends of the initial reactances. And secondly the control of the N parameter will depends of the power of the input signal.

It was also interesting to compare the two different normalizations presented in equations 2.15 on page 18 and 2.16 on page 19. In the curve in figure 4.3 on page 46 the normalisation is the version of equation 2.16 on page 19, and, the initial parameters are the same.
In this curve the power level grows up faster, and seems to converge in mean towards a constant value. This means that the normalization of the cost function can change the behaviour of the convergence speed.
In this case the instability of the curve is obvious, but it is according to the theory[?]. Thus, a solution could be to use a variable μ parameter which will depend of n, i.e $\mu = \mu(n)$, as Pr. Shishkov proposes in [?]. The new formulation of μ will be :

$$\mu(n) = \frac{\mu}{1 + \frac{n}{\tau}} \tag{4.1}$$

where τ and μ are constant values.
So, when n will increase, i.e. goes towards convergence value, $\mu(n)$ will vary in the way to decrease the convergence step, so decrease the instability.

All these results and remarks have been use in the following experiments.

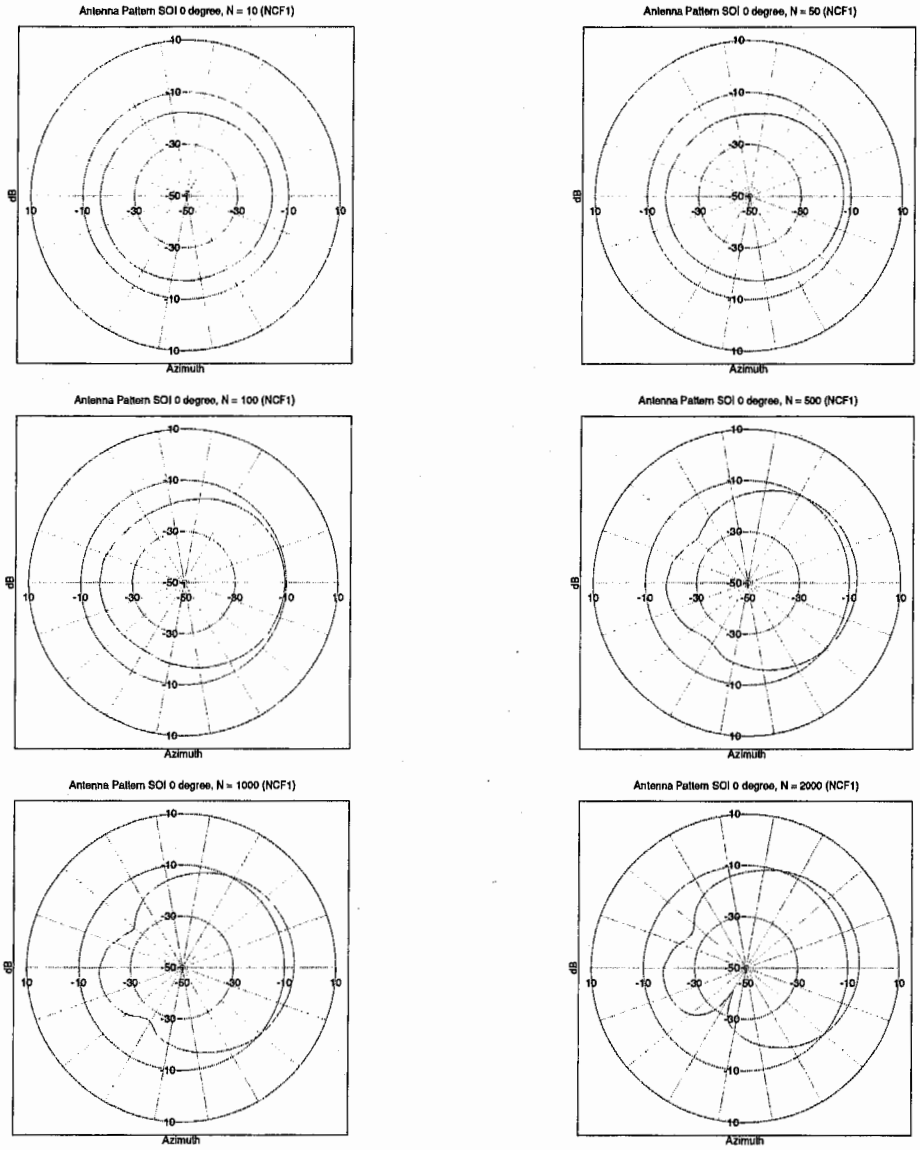


FIGURE 4.1: Experiment, ESPAR antenna patterns : beam forming for N = 10, 50, 100, 500, 1000, 2000 (with normalization)

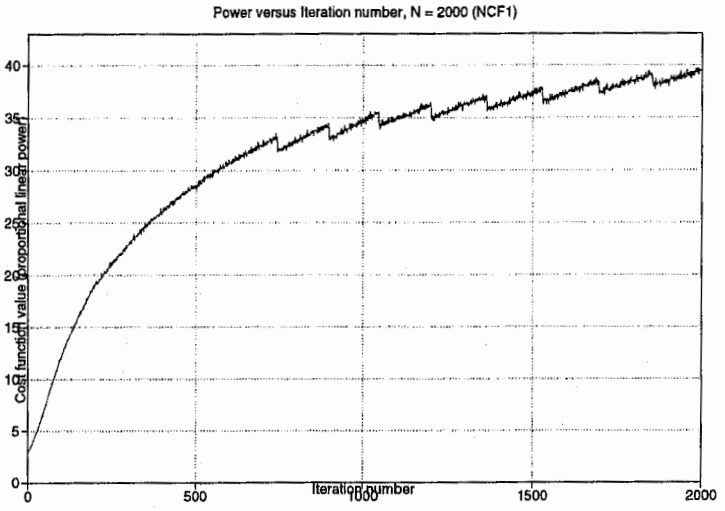


FIGURE 4.2: Experiment, power as cost function versus number of iterations

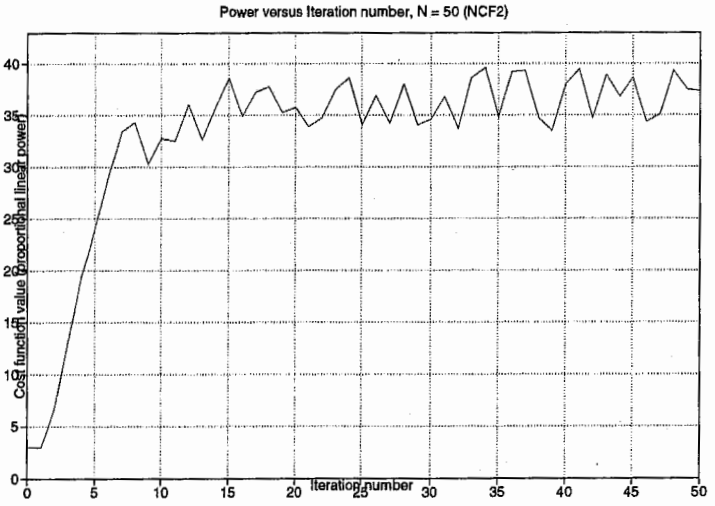


FIGURE 4.3: Experiment, power as cost function versus number of iterations

4.2 Experimental results for 2^{nd} solution

In these experiment, the power is directly measured whereas in DSP case the power is calculated using the Q and I signals of the output signal of the antenna. The results expected should be most closed to the simulation results.
The experiments took place at Antenna GIGEN Corporation[4], in their anechoic experimental room. Many experiments have been made, in order to inspect the influence of the frequency, the initial voltage values, the parameters[5], the variation off one or all of the voltages. All these experiment results are presented in Mr. Jun Cheng experiment paper for WACNet. Obviously in our paper only some results will be shown.

The patterns in figure 4.4 on the next page represents the beam forming for 0°,30° and 60°. The value of μ is high because the linear power level is low (*see figure 4.6 on page 50 and figure 4.5 on page 49.* It can be noticed that the pattern for 60° can be obtained by symetry of the pattern of 0°.
Also, the figure 4.6 on page 50 and figure 4.5 on page 49 show the convergence curve of the cost function and the voltages, for 0° and 30°.

[4] Tōkyō
[5] μ,Δx

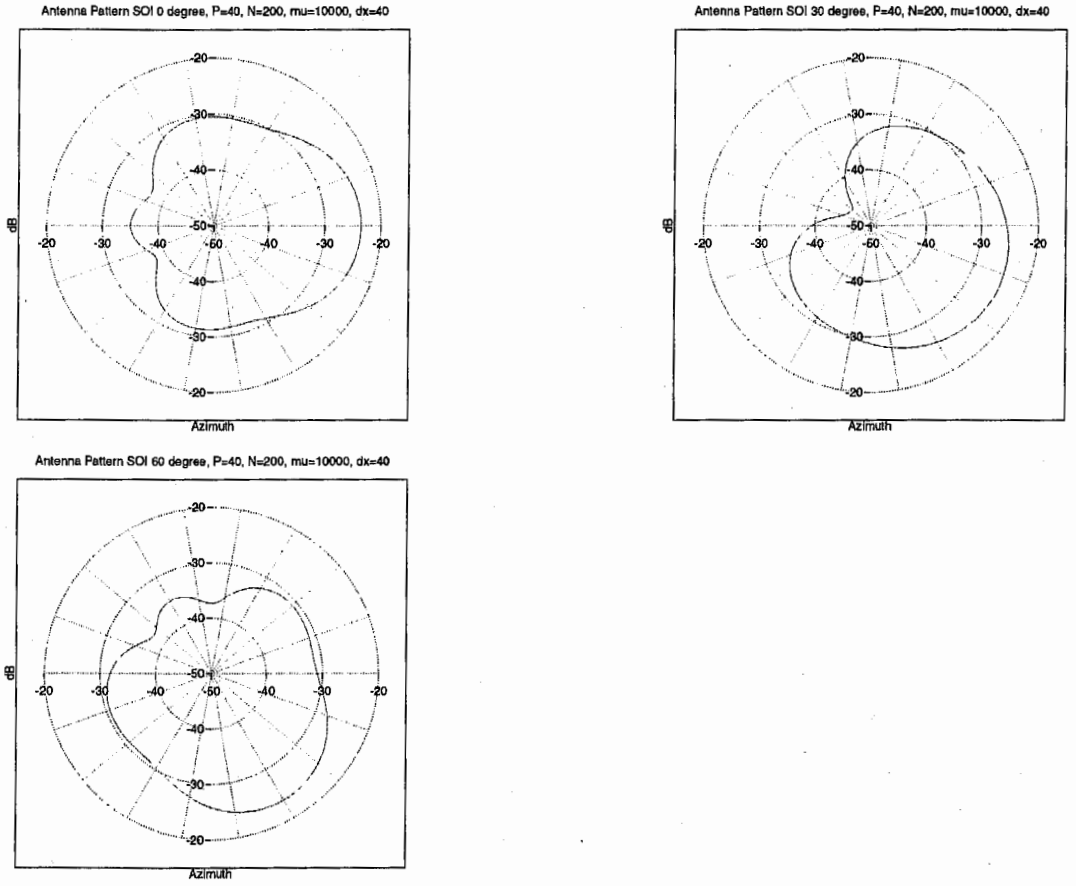


FIGURE 4.4: Experiment, ESPAR antenna patterns : beam forming for SOI 0, 30 and 60 degree

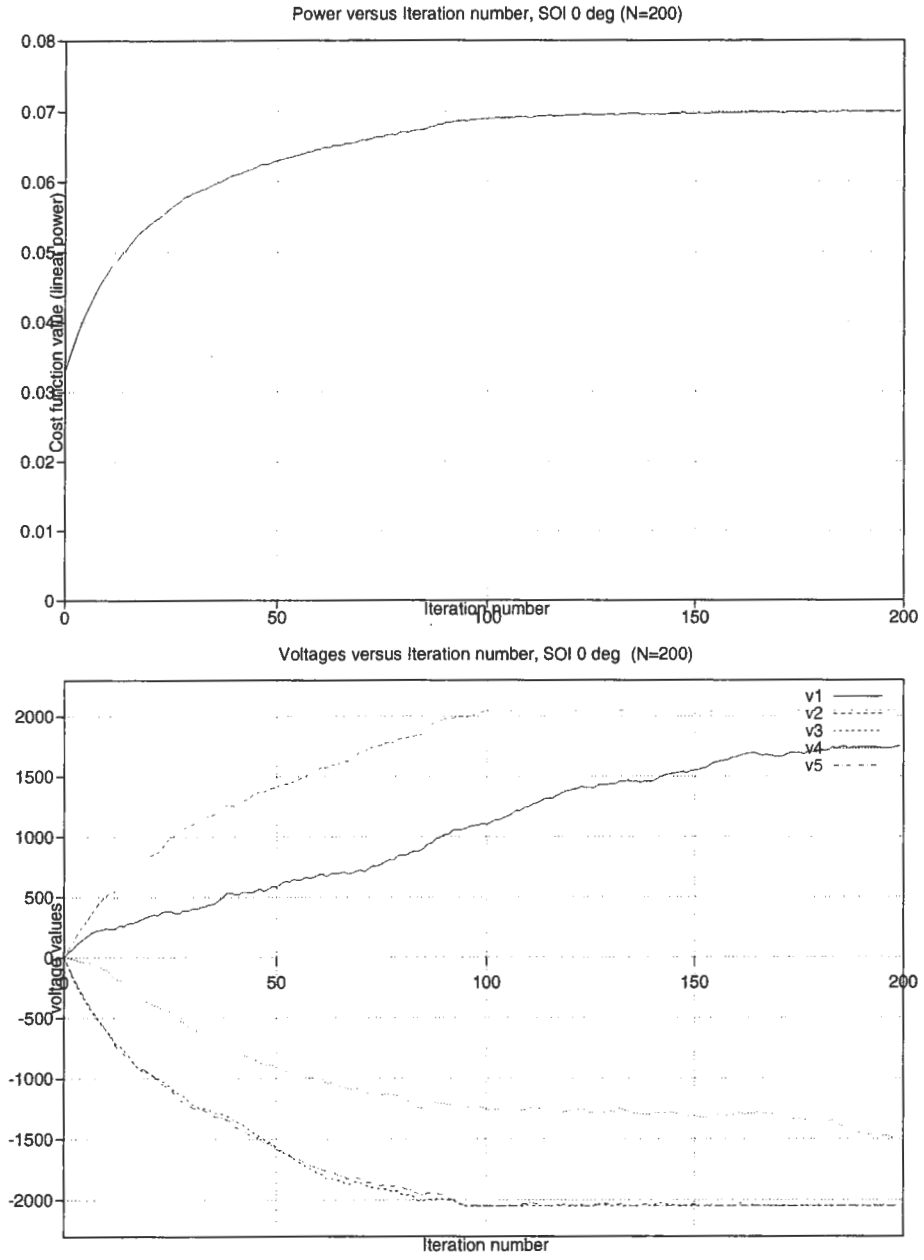


FIGURE 4.5: Experiment, convergence curve, for cost function and voltages, SOI = 0 degree

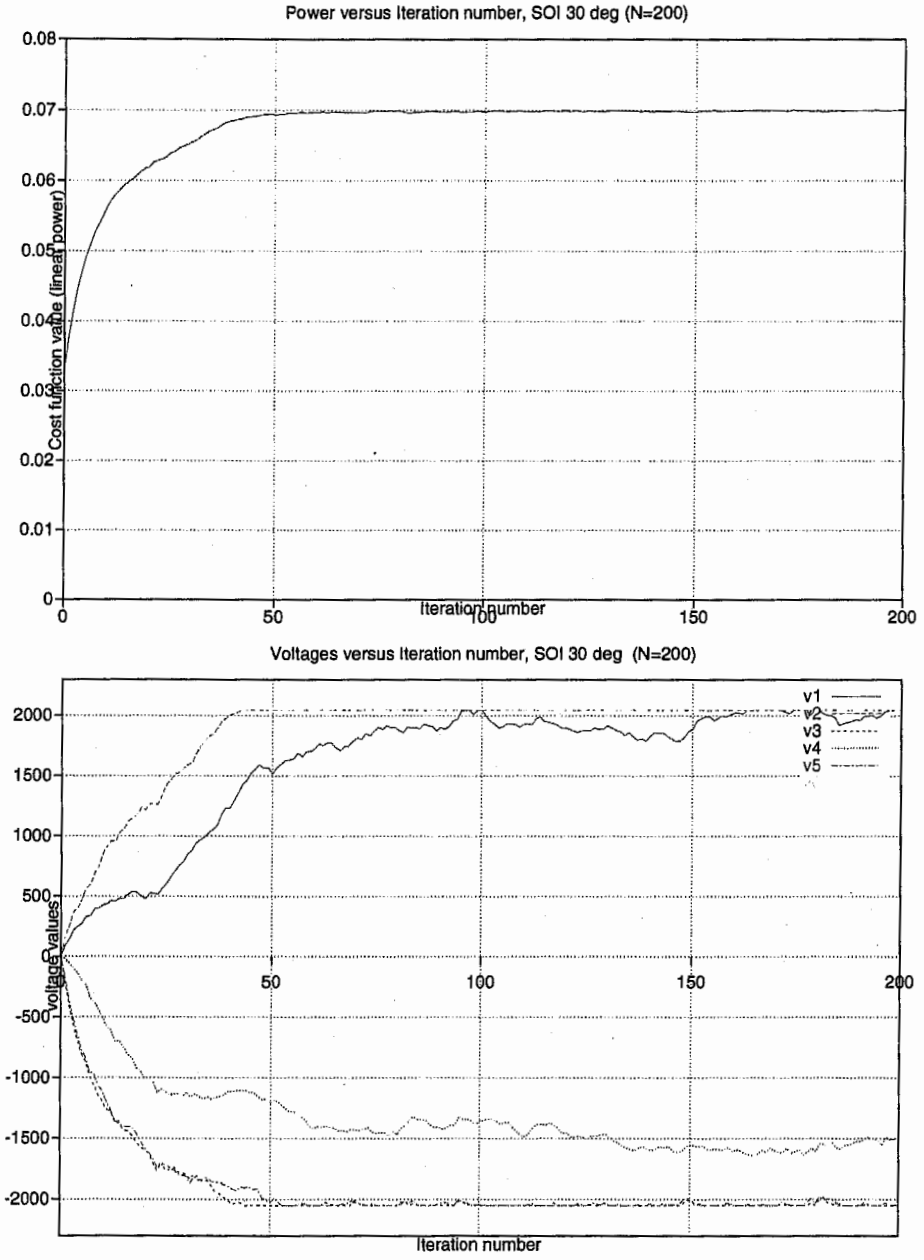


FIGURE 4.6: Experiment, convergence curve, for cost function and voltages, SOI = 30 degree

4.3 Conclusion of this part

To summarize we can say that ESPAR antenna can form beam in all directions. Adaptively find the voltages allowed making useful experiments and concluded that the antenna can steer beam, i.e can work. The ESPAR antenna sector beam pattern can be form in the two experiment conditions, as it can be seen on the results. Then, with the adaptive algorithm we made, we can also inspect the behavior of the ESPAR antenna in several frequencies, and voltages values.

CONCLUSION

This paper proposes a way to make the ESPAR antenna adaptively form beam, and to obtain the antenna sector beam pattern[?]. The ESPAR antenna is a new kind of smart and low cost antenna, which allows many perspectives.

Actually find an algorithm that allows ESPAR antenna to steer beam in direction of signal and nulls in direction of interferences, is again a hot topic. The main issue is that ESPAR antenna is not a conventionnal array antenna. Theories about array antenna can not be directly apply. At ATR-ACR we try to find the better algorithm for this problem, which fit to the WACNet exigencies.

The gradient-based algorithm[?], is the first algorithm proposed by ATR-ACR for the ESPAR antenna adaptive control. Our present work was to modify this algorithm in order to adaptively form beam and then obtain the sector beam pattern. The simulations made with MatLab, and the several experiments made in the anechoic room, show that we can adaptively make the ESPAR antenna form beam.

With our present work we can also inspect the behaviour of the ESPAR antenna in many condition, and find the better parameters like initial voltages, gradient step size, etc., to improve the beam forming gain power. Also this work can be extend to other algorithms.

POWERED BY
A TUX IN LATEX

ACKNOWLEDGEMENT

I would like to express my gratitude to Mr. Jun Cheng, who was an very good supervisor, Mr. Takashi Ōhira, head of Department 3, who was an excellent head supervisor, and Dr. Bokuji Komiyama, president of ACR for their helpful encouragements and their interests to my work.

Even he does not work now at ACR, I would like to thank Mr. Yukihiro Kamiya who introduced me to ACR laboratories.

I would like to thank professor Blagovest Shiskov for his help at the beginning and during my intership, and all the ACR department 3 team, Mr. Masaya Hashiguchi, Mr. Kyōichi Iigusa, Ms Qing Han, Mr. Atsuya Andō, Mr. Akifumi Hirata, Mr. Kehu Yang, and Mr. Tomoshige Furuhi for their interesting collaboration and discussions.

I would like to express my many thanks to Ms. Naoko Nakaya, Ms. Mami Hamba, Ms. Natsuko Harada, Ms. Sachiko Kudō and Ms Mieko Tanaka for their helpful contribution in my Japanese language study and their kind personnality.

I also would like to thank the ATR soccer team and all of their members for the good "football" playing moments. I would like to thank the SHIEN group and specialy Ms. Kay Mikami, Ms. Makiko Tatsumi and Ms. Maki Miura. And I finally express my gratitude to Mr. Yōiti Kado, Mr. Keizo Inagaki, Ms. Hu Wei Wei, Senior Vaccaro *Don* Pablo, and all ATR-ACR staff.

O my god ! I almost forgot to thank the genius Robbert Schulb for his wonderfull help.

Bibliography

[1] The official bluetooth sig website. http://www.bluetooth.com/.

[2] Jun Cheng, Yukihiro Kamiya, and Takashi Ōhira. Adaptive beamforming of espar antenna based on steepest gradient algorithm. *IEICE Trans. Com.*, E84-B(7):1790–1800, July 2001.

[3] duflot. Cours sur les antennes. http://www-mo.enst-bretagne.fr/~duflot/courstel/antennes/anten0_f.html.

[4] Koichi Gyoda, Yuichiro Ohno, Yōito Kado, and Takashi Ōhira. Wacnet: Wireless ad-hoc community network. *IEEE Inter. Conf. on Industrial Electronics, Control and Instrumentation*, pages 1153–1158, October 22-28 2000. Nagoya Japan.

[5] Masaya Hashiguchi, Jun Cheng, Kyōichi Iigusa, Eddy Taillefer, Akifumi Hirata, and Takashi Ōhira. Basic experiments for adaptive beamforming of the espar antenna. *Proceedings of the 2001 communications society conference of IEICE*, B-1-65:71–76, September 18 2001. in Japanese.

[6] J.E. Hudson. *Adaptive Array Principles.* IEE electromagnetic wave series 11, 1981.

[7] McGraw-Hill. *Antennas.* ELECTRICAL AND ELECTRONIC ENGINEERING SERIES, 1950.

[8] Edmond Nicolau and Dragoş Zaharia. *Adaptive Arrays.* ELSEVIER, 1989.

[9] T. Ōhira, K. Gyoda, A. Akiyama, and K. Yang. Equivalent weight vector and array factor formulation for espar antennas. *Technical Report of IEICE*, AP2000-44,SAT2000-41,NW2000-41, July 2000. in Japanese.

[10] Yuichiro Ohmo, Takashi Ōhira, and Koichi Gyoda. A segmentation scheme for sdma ad-hoc networks. *IEICE Technical Report*, IN99-77:1–6, Nov 1999.

[11] Blagovest Shiskov and Takashi Ōhira. Adaptive beamforming of espar antenna based on stochastic approximation theory. ATR Adaptive Communications Research Laboratories, 2001.

APPENDIX 1

1.4 Finite-difference approximations of derivatives

Let consider a $\mathcal{C}^2(\mathbb{R})$ function $f : \mathbb{R} \to \mathbb{R}, x \longmapsto f(x)$. For the approximation of the first derivatives, $\frac{\partial f}{\partial x}$ finite-difference approximations can be used. The finite-differences are calculated from a number of functions evaluations of particular $\vec{x}$ values. Foward (*figure* 1.7) or central(*figure* 1.8) approximation can be used.

1.4.1 Foward approximation

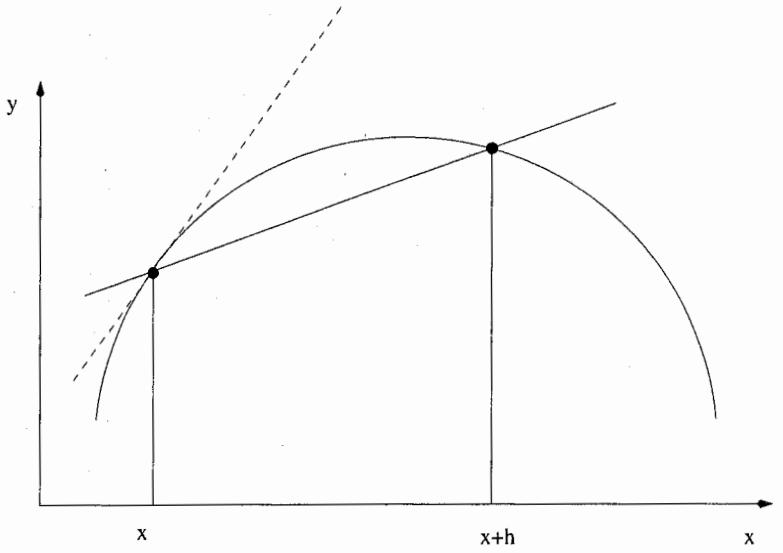


FIGURE 1.7: First order partial derivative

The Taylor-series approximation of the twice continuously differentiable

univariate[6] function $f(x)$ is :

$$f(x+h) = f(x) + hf'(x) + O(h^2) \tag{1.2}$$

$O(h^2)$ represents a term of order 2 or higher. The forward-difference formula

$$\varphi_F(f,h) = \frac{f(x+h) - f(x)}{h} = f'(x) + O(h) \tag{1.3}$$

$\varphi_F(f,h)$ is the approximation of the first derivative of $f(x)$ using the step length h. Similarly, the backward-difference approximation can be calculated by

$$\varphi_B(f,h) = \frac{f(x) - f(x-h)}{h}. \tag{1.4}$$

The truncation error of both, the forward and backward-differences is

$$\frac{1}{2} \cdot h \cdot f''(x + \theta \cdot h) \tag{1.5}$$

where $0 \leq \theta \leq 1$

1.4.2 Central approximation

For higher accuracy the central-difference (*cf. figure* 1.8) approximation can be used.
Central-differences are derived from the Taylor expansion of second order

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3) \tag{1.6}$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3). \tag{1.7}$$

From these equations the central-difference formula

$$\varphi_C(f,h) = \frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2) \tag{1.8}$$

is derived.

In this approximation the truncation error is of second order

$$\frac{1}{6} \cdot h^2 \cdot f^{(3)}(x + \theta \cdot h), \tag{1.9}$$

[6]Here the function $f(x)$ is assumed to be a univariate function. For multivariate functions finite differences are performed for each dimension of $\vec{x}$.

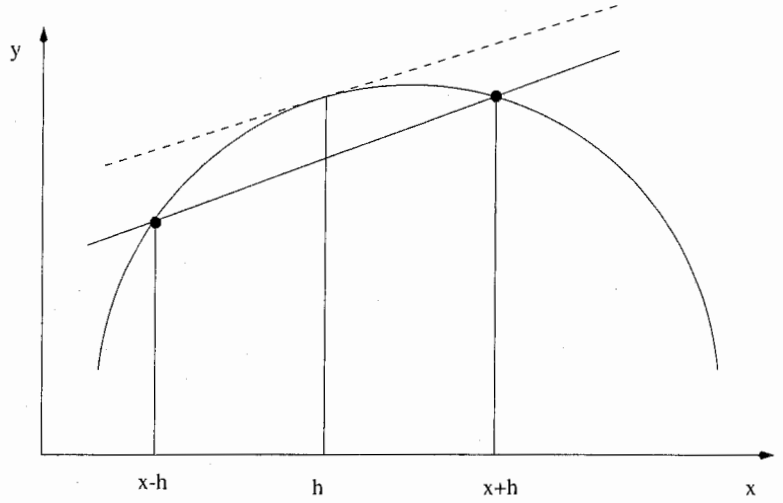


FIGURE 1.8: Second order partial derivative

but two function evaluations around $\vec{x}$ are required.

When the target function has n independent parameters $s \cdot n$ function evaluations are required using a central-difference approximation. If a single evaluation takes several minutes to hours the forward-difference formula is preferable to reduce the overall calculation time. To keep the error of the approximation in an acceptable range the step size is adapted for each input parameter, depending on the values of the previous iteration.

Appendix A : MatLab source codes

essai1.m

program name : essai1.m
path: source/matlab/reactance_control/essai1.m
comments : Main program

```
function [grang,cost_func]=essai1(N,P,mu,delta,DoAs,SOI,type)

% test of the adaptive algo
%
% essai1(N,P,mu,delta,DoAs,SOI,type)
% DoAs      liste of DoA
% type      cost function type (see adaptivealgo CostFuncType)
% SOI       signal of interest (one among DoAs)

disp(['N     -> ' num2str(N)]);
disp(['P     -> ' num2str(P)]);
disp(['mu    -> ' num2str(mu)]);
disp(['delta -> ' num2str(delta)]);
disp(['DOAs  -> ' num2str(DoAs)]);
disp(['SOI   -> ' num2str(SOI)]);
disp(['type  -> ' type]);

%clear all;

cwd = pwd;
cd(tempdir);
pack
cd(cwd)
```

```
M = 7;

%SOI = 4;
Zrange=[-300 300];

tot=N*P*M;        % a total of samples for training


NoSig=length(DoAs);


% symbol sequences for desired and interference signals

% prepare for desired signal (sindex=SOI)
desiredsig=[];
for n=1:N
   sigrefblock=(1-2*randint(P,1,2));
   for mele=1:M
       desiredsig=[desiredsig; sigrefblock];   % repeat M blocks
   end
end

sig(:,SOI)=4*desiredsig;   % symbol sequence of desired signal
clear desiredsig;

% prepare for interference signal
for sindex=1:NoSig
  if sindex~=SOI
      sig(:,sindex)=(1-2*randint(tot,1,2));
  end
end

% preparing the noise
% noise=(randn(1,tot)+randn(1,tot)*j);
noise=zeros(1,tot);

%  signals imping on elements
Xin=zeros(M,length(sig(:,SOI)));    % NUM*Fs/Fd samples for training
for sindex=1:NoSig
   steervector=ESteer(DoAs(sindex),M);
   Xin = Xin + steervector.'*sig(:,sindex).'; % Response to a direct path from angle theta.
end
```

```
clear steervector;
clear sindex;

% calcul (steepest gradient)
% preparing reference signal
r = sig(:,SOI).'*exp(j*deg2rad(randint(1,1,360)));
deltaZ = ones(M,1)*delta;

% type = 'power';
[w,x,cost_func]=adaptivealgo(Xin,r,noise,mu,deltaZ,P,Zrange,type);
disp(['reactances for ', type,' -> ']);
disp(num2str(x));

% drawing graphs and curves, so the results

NoFig = 1;
hold off;

figure(30);
subplot(2,2,1);

for i=1:N
%     divers(i-1) = real(cost_func(i))-real(cost_func(i-1));
    divers(i) = mean(real(cost_func(1:i)));
end
% divers(N) = divers(N-1);

plot(1:N,real(cost_func),'',1:N,divers,'r-');
plot(1:N,real(cost_func));

%axis([0 N 0.01 1]);
xlabel('Number of Iterations');
ylabel(type);
title([type ' vs Iterations']);

minTheta = 0;
maxTheta = 360;
theta = deg2rad(minTheta:maxTheta-1);

val = ones(1,maxTheta-minTheta);
for index=2:M
    val=[val;exp(j*(pi/2.0)*cos(theta-(index-2)*(2*pi/(M-1))))];
end
```

```
range = minTheta:maxTheta-1;                    % range for DoAs

gain = 20*log10(abs(w.'*val));

grang = [range.',gain.'];

subplot(2,2,2);
plot(range,gain,'',DoAs,gain(DoAs+1),'r.--');
title(['DOA:  ',num2str(DoAs), '       SOI: ',num2str(SOI),'     Cost Function:    ',type]);
ylabel('Response (dB)');

% ad_gain = min(gain);
% if ad_gain < 0
%     ad_gain = -ad_gain;
% else
%     ad_gain = 0;
% end

% polar(deg2rad(range),gain+ad_gain);
% title(['DOA:  ',num2str(DoAs), '       SOI: ',num2str(SOI),'     Cost Function:    ',type]
% ylabel('Response (dB)');

subplot(2,2,3);
polar_ploth(deg2rad(range),gain,min(gain),2,'m-', max(gain));
title(['DOA:  ',num2str(DoAs), '       SOI: ',num2str(SOI),'     Cost Function:    ',type]);
title(['DOA:  ',num2str(DoAs),'     Cost Function:    ',type]);
% ylabel('Response (dB)');
```

adaptivealgo.m

program name : adaptivealgo.m
path: source/matlab/reactance_control/reactance_control/adaptivealgo.m
comments : Adpative algorithm function

```
function [w,x,cost_func]=adaptivealgo(Xin,r,noise,mu,deltaZ,P,Zrange,CostFuncType)
% gradient-based algorithm for ESPAR antenna
%
% adaptivealgo(Xin,r,noise,mu,deltaZ,P,Zrange,CostFuncType)
%
% input:    Xin      [M,N*P]-matrix an antenna input signal
%           r        [1,N*P]-matrix a desired array output
%           deltaZ   [M,1] pertubation size for each individual reactance
%           P        same as in the Cheng paper
%           Zrange   range of reactances
%           CostFuncType :
%                           rho -> normalize cross correlation between r and y
%                           1-rho2 -> 1 minus rho ^ 2
%                           pwower -> (1/P)*(y*y')
%                           power inverse -> ...
%
% output:   w        M-vector adaptive weight
%           x        M-vector reactance
%

[M,val] = size(Xin);
N = val/(P*M);
clear val;

x0 = -j*50; % x0 is always set to zero (or -j50)

% Init x(1)
x = zeros(M,1);
x(1) = x0;

deltax = diag(deltaZ);
deltax(1,1) = 0;

% calcul of the current vector
w = rea2cur(x);
```

```
% main loop steepest gradient algorithm
for n=1:N

    tmp = Xin(:,((n-1)*P*M+1):n*P*M);
    ref = r(((n-1)*P*M+1):n*P*M);
    nse = noise(((n-1)*P*M+1):n*P*M);

    % measure y(n) = Wnc^H * Xin + noise

    y=w.'*tmp(:,1:P)+nse(1:P);

    % cost function calculate
    switch CostFuncType
        case 'rho'
            cost_func(n) = abs(ref(1:P)*y')/(sqrt(ref(1:P)*ref(1:P)')*sqrt(y*y'));
            cost0 = cost_func(n)*ones(1,M);
        case 'power'
            cost_func(n) = (y*y')/P;
            cost0 = cost_func(n)*ones(1,M);
        case 'power inverse'
            cost_func(n) = P/(y*y');
            cost0 = cost_func(n)*ones(1,M);
        case '1-rho2'
            val = abs(ref(1:P)*y')/(sqrt(y*y')*sqrt(ref(1:P)*ref(1:P)'));
            cost_func(n) = 1.0-val*val;
            %cost_func(n) = 1 - abs(ref(1:P)*y')^2/((y*y')*(ref(1:P)*ref(1:P)'));
            cost0 = cost_func(n)*ones(1,M);
        otherwise
            error('bad cost function type');
            return;
    end
    cost0(1) = 0.0;

    % inner loop, M blocs calcul
    for i=2:M
       % Xm=Xm+deltaXm

       dx = x+deltax(:,i);

       for p=2:M
          if dx(p) > Zrange(2)
             dx(p) = Zrange(2);
```

```
            end
            if dx(p) < Zrange(1)
               dx(p) = Zrange(1);
            end
        end

        wI = rea2cur(dx);

        tmpI = tmp(:,(i-1)*P+1:i*P);
        refI = ref(:,(i-1)*P+1:i*P);
        nseI = nse(:,(i-1)*P+1:i*P);

         yI(i,:) = wI.'*tmpI+nseI;

         switch CostFuncType
             case 'rho'
                 cost(i) = abs(refI*yI(i,:)')/(sqrt(yI(i,:)*yI(i,:)')*sqrt(refI*refI'));
             case 'power'
                 cost(i) = (yI(i,:)*yI(i,:)')/P;
             case 'power inverse'
                 cost(i) = P/(yI(i,:)*yI(i,:)');
             case '1-rho2'
                 val = abs(refI*yI(i,:)')/(sqrt(yI(i,:)*yI(i,:)')*sqrt(refI*refI'));
                 cost(i) = 1.0-val*val;
                 %cost(i) = 1 - abs(refI*yI(i,:)')^2/((yI(i,:)*yI(i,:)')*(refI*refI'));
             otherwise
                 error('bad cost function type');
             return;
         end
    end

    % gradient normalize
%     x = x  + (mu/norm(cost-cost0,2))*(cost-cost0).';

    % not normalize
    x = x  + mu*(cost-cost0).';

    x(1) = x0;

    for i=2:M
       if x(i) > Zrange(2)
          x(i) = Zrange(2);
       end
```

```
        if x(i) < Zrange(1)
           x(i) = Zrange(1);
        end
    end

    w=rea2cur(x);
end
```

rea2cur.m

program name : rea2cur.m
path: source/matlab/reactance_control/rea2cur.m
comments : Reactance to current

```
function cur=rea2cur(rea)
%
% function cur=rea2cur(rea)
% Given reactance vector, calulate the current vector
%
% input:  rea, reactance vector
% output: cur, current vector
%
% the follwing data is from Dr. T. Ohira
%y00 = [0.00860035, -0.0315844]  --- 中央素子入力アドミタンス
%y10 = [-0.00372642, 0.0072319]   --- 中央周辺結合アドミタンス
%y11 = [0.00962295, -0.01656835]   --- 周辺素子入力アドミタンス
%y21 = [-0.000377459, 0.0117867]   --- 隣接素子結合アドミタンス
%y31 = [0.00002720885, -0.0063736]   --- 次隣接素子結合アドミタンス
%y41 = [0.001779525, 0.002208335]   --- 対向素子結合アドミタンス

% set the admittances
VS=100;

m=7;
%y00 = 0.00860035-j*0.0315844;  %--- 中央素?q 入力アドミタンス
y00 = 0.0008616-j*0.0120795;
%y10 = -0.00372642+j*0.0072319;   %--- 中央??Ó結?KK A ドミタンス
y10 = -0.0006963+j*0.0036462;
%y11 = 0.00962295-j*0.01656835;  % --- ??Ó素?q 入力アドミタンス
y11 = 0.0044216-j*0.0071600;
%y21 = -0.000377459+j*0.0117867;  % --- 隣?Ú素?q 結?KK A ドミタンス
y21 = 0.0009721+j*0.0047851;
%y31 = 0.00002720885-j*0.0063736;  % --- ?泓Œ?Ú素?q 結?KK A ドミタン
ス
y31 = -0.0005376-j*0.0011297;
%y41 = 0.001779525+j*0.002208335;   %--- 対向素?q 結?KK A ドミタンス
y41 = 0.0001701-j*0.0002950;
```

```
% set the admittance matrix
yy=[y00 y10 y10 y10 y10 y10 y10;
    y10 y11 y21 y31 y41 y31 y21;
    y10 y21 y11 y21 y31 y41 y31;
    y10 y31 y21 y11 y21 y31 y41;
    y10 y41 y31 y21 y11 y21 y31;
    y10 y31 y41 y31 y21 y11 y21;
    y10 y21 y31 y41 y31 y21 y11];

% set the admittance vector
    yvector=[y00 y10 y10 y10 y10 y10 y10];

% calulate the current vector
    cur=VS*inv(eye(m)+j*yy*diag(rea))*yvector.';
```

ESteer.m

program name : ESteer.m
path: source/matlab/reactance_control/ESteer.m
comments :

```
function steervector=ESteer(theta,M)
%
%function ESteer(theta,M)
% Forming a steer vector with respect to the DOA theta.
% input: theta, Doa of signal
%         m,    number of elements of antenna
% output: steer vector

derad=pi/180.0;

%ESPAR Antenna
steervector(1)=1.0;
for index=2:M
steervector(index)=exp(j*pi/2.0*cos(theta*derad-(index-2)*(2*pi/(M-1))));
end
```

rad2deg.m

program name : rad2deg.m
path: source/matlab/reactance_control/rad2deg.m
comments : Function to convert from radian to degree

```
function ret=rad2deg(in_rad)
 % convert radian angle to degree angle

 if in_deg >=0 & in_deg <= (2*pi)
     ret = 360*(in_rad/(2*pi));
 else
     disp('do not deal with values different from [0,2*pi]');
 end
```

deg2rad.m

program name : deg2rad.m
path: source/matlab/reactance_control/deg2rad.m
comments : Function to convert from degree to radian

```
function ret=deg2rad(in_deg)
 % convert degree angle to radian angle

 if in_deg >=0 & in_deg <= 360
     ret = 2*pi*(in_deg/(360));
 else if in_deg >=-180 & in_deg <0
         ret = 2*pi*((in_deg+360)/360);
     else
         error('only [-180,0[U[0,360]')
     end

 end
```

polar_ploth.m

program name : polar_ploth.m
path: source/matlab/reactance_control/polar_ploth.m
comments : Function to trace polar sector (Made by R. Schulb)

```
function polar_ploth(theta, r, centre_value, flag, S, max_mag)
%POLAR_PLOT     polar plot of power values with centre value defined in
%               typical H field orientation
%
%polar_plot(THETA, R, C, FIELD, flag, S, MAX_MAG) will plot the power values
%defined by their angle THETA and magnitude R in a polar form around a centre value of
%C. S is the line format you wish to use and is the same as the formats for the
%standard plot function
%
%MAX_MAG is the maximum magnitude of r that you want to set the axis to.
%e.g. if you are plotting multile plots you will want to set the maximum r axis
%to the maximum of all plots. If only plotting one plot then simply enter
%
%           polar_ploth(THETA, R, CENTRE_VALUE, FLAG, S, max(r))
%
%FLAG = 0 - Data plotted on new figure and polar axes drawn
%FLAG = 1 - Data plotted on new figure without polar axes
%FLAG = 2 - Data plotted on current figure and polar axes added
%FLAG = 3 - Data plotted on current figure without adding polar axes
%
%Also, any modifications to the axis plotting/representation can be adjusted in the first
%few lines of the polar_ploth function m file. The modifications are: Number of rings,
%the angle were the dB markings are displayed, the number of angle marks (radial lines)
%and axis color.
%
%The FLAG is useful for plotting multiple plots if you want to use a legend. For example,
%to plot 3 plots with a legend of the graphs use the following FLAG order: 1, 3, 2.
%Make sure the axis are plotted in the last trace so they will not appear in the 'legend'


%some setup parameters
number_of_rings = 4;    %number of rings shown in plot
dB_mark_angle = 45;     %angle where the dB markings are
no_angle_marks = 12;    %12 gives 30 degree spaceing
line_color = 'k';       %color of axis - k is black
```

```
%creat figure if necessary
if flag == 0 | flag == 1
   figure
end

hold on

%make sure input is not bogus
if max_mag < centre_value
   text(-0.7*max_mag,0.1,'Centre value is larger than maximum input value');
   break;
end

if max_mag == centre_value
   text(-0.7*max_mag,0.1,'Centre value is same as maximum input value');
   break;
end

%nomalize max_mag to centre value
max_mag = max_mag - centre_value;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%normalize the r input value. Shift all values up by the centre value
%amount. If the CV is negative, then all values will be shifted up,
%but if the CV is positive, then all values will be shifted down.
r = r-centre_value;
for i = 1:1:length(r)
   if r(i)<0
      r(i) = 0;
   end
   if r(i)>max_mag
      r(i) = max_mag;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plot the values
x = r.*cos(theta);
y = r.*sin(theta);
plot(x,y,S);

%if the flag is set to plotting on current figure, then quit out
```

```
%without drawing axis and values
if flag == 1 | flag == 3
   hold off;
   break;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Draw the polar axis

%Draw the concentric circles
phi = 0:0.01:2*pi;
rings = max_mag;
for i = 1:1:number_of_rings
   rings = [rings (number_of_rings-i)*max_mag/number_of_rings];
end

for i = 1:1:length(rings)
   x = rings(i)*cos(phi);
   y = rings(i)*sin(phi);
   plot(x,y,strcat(':',line_color))
end

%Draw the angle markers
phi = (0:360/no_angle_marks:359)*pi/180;
for i = 1:1:length(phi)
   x = max_mag*cos(phi(i));
   y = max_mag*sin(phi(i));
   plot([0 x],[0 y],strcat(':',line_color))
end

%Draw on text labeling the angles
phi = 0:360/no_angle_marks:359;
for i = 1:1:length(phi)
   x = max_mag*1.15*cos(phi(i)*pi/180);
   y = max_mag*1.15*sin(phi(i)*pi/180);
   text(x,y,num2str(phi(i)))
end
axis([-max_mag*1.2 max_mag*1.2 -max_mag*1.2 max_mag*1.2])

%Draw on the text labeling the magnitude
angle_of_mark = dB_mark_angle*pi/180;
for i = 1:1:length(rings)
   x = (rings(i)+0.05*max_mag)*cos(angle_of_mark);
```

```
    y = (rings(i)+0.05*max_mag)*sin(angle_of_mark);
    text(x,y,num2str(rings(i)+centre_value))
end

axis off

hold off
```

Appendix B : DSP source codes

eddy.h

program name : eddy.h
path: source/dsp/src_modif/eddy.h
comments : Header

```
/*
   ----------------------------------------
   Some defines added by Taillefer Eddy in order to
   check the use of power and power inverse as
   cost function
*/

/* x(n+1) = x(n) + mu * COST_FUNCTION */

#define POWER 1         /* for beam */
#define POWINV 2         /* for null   */
#define POWMINUS 3         /* for null */
#define POWINVMINUS 4         /* for beam */

#define COST_FUNCTION POWER

#define NORMALIZE 1 /* normalization of the cost function
   Jk = (Ek-E0)/sqrt(sum(Ei-E0,i=1:M)/M)
   0 -> normalize is not used
   1 -> normalize is used

*/
```

```
#define NORM_VERSION 1 /* normalize version, 1-2 are possible values
  1-> Jk = (Ek-E0)/sqrt(sum(Ei,i=1:M)/M)
  2-> Jk = (Ek-E0)/sqrt(sum(Ei-E0,i=1:M)/M)

keep to 1
*/

#define SHIFT_COEF 13 /* 13 -> 14 -> 0 -> 16 */

#define OFFSET_I (float)(-30.0) /* dc offset in I channel */
#define OFFSET_Q (float)(10.0) /* dc offset in Q channel */
/*
   ----------------------------------------
*/
```

macro.h

program name : macro.h
path: source/dsp/src_modif/macro.h
comments : Header

```
struct IQ {
float           i; /* I-ch */
float           q; /* Q-ch */
};

/*extern float    correlation(const struct IQ *, const struct IQ *);
  */
```

ftutil.h

program name : ftutil.h
path: source/dsp/src_modif/ftutil.h
comments : Header

```
/*SDOC---------------------------------------------------------------------------
Spectrum Signal Processing Inc. Copyright 1998

$Workfile:: ftutil.h                                                          $

$Modtime:: 1/12/99 9:29p                                                      $

$Revision:: 5                                                                 $

$Archive:: /MAC/dy62_152.v10/dev/test/hwtest/hpiread/ftutil.h                 $

Contents:

This file contains several utility functions used in the FT example code.

-------------------------------------------------------------------------EDOC*/
#ifndef SSP_FTUTIL_H
#  define SSP_FTUTIL_H

#include "ftc6x.h"

RESULT C6x_FTUTIL_FlashLed( UINT32 Led, UINT32 Duration );
#define C6X_FLASHLED_OK      (0x00800000)
#define C6X_FLASHLED_ERROR   (0x00100000)

RESULT C6x_FTUTIL_GetPCIDevNumber( UINT32 *SSRAM_DevNum_Reg );

UINT32 C6x_FTUTIL_TimerStart( void );
UINT32 C6x_FTUTIL_TimerRead( void );

#endif

/* End of $Workfile:: ftutil.h                                                $
*****************************************************************************/
```

command.h

program name : command.h
path: source/dsp/src_modif/command.h
comments : Header

```
#include "ftc6x.h"

struct IQ {
float           i; /* I-ch */
float           q; /* Q-ch */
};


#define CMD_DSP_INIT 0x00
#define CMD_DSP_START 0x10
#define CMD_DSP_STOP 0x18
#define CMD_DSP_RESUME 0x20
#define CMD_DSP_POSE 0x28

#define CMD_VALID 0x80
#define CMD_INIT_RES 0x80000000
#define CMD_DATA_SEND 0x80000010

#define C6X_DMA_FLAGS  0x01000050
#define C6X_DMA_PEM_FLAGS  0x01000040

typedef struct tagCMD_FORMAT{
UINT32 cmdFlag;
UINT32 cmdId;
UINT32 cmdWords;
UINT32 cmdData[4];
} CMD_FORMAT;

typedef struct tagCMD_FORMAT_SEND{
UINT32 cmdFlag;
UINT32 cmdId;
UINT32 cmdWords;
UINT32 cmdData[1];
} CMD_FORMAT_SEND;
```

```
typedef struct tagINIT_PARAM{
UINT32 savedData;
UINT32 sample;
UINT32 updateCount;
float step;
INT32 DeltaX;
/* UINT32 DeltaX; */
UINT32 element;
UINT32 guardTime;
UINT32 daSetup;
INT32 voltage[8];
} INIT_PARAM;

typedef struct tagMEASURED_DATA{
UINT32 frameNo;
UINT32 elementNo;
UINT32 coeffId;
UINT32 coeffWords;
float coeffData;
UINT32 voltId;
UINT32 voltWords;
INT32 voltData[8];
UINT32 inputId;
UINT32 inputWords;
INT32 inputData[2];
} MEASURED_DATA;
```

esppem.h

program name : esppem.h
path: source/dsp/src_modif/esppem.h
comments : Header

```
#define PEM_FIFO_RESET ((volatile UINT32 *)0x03000000) /* Reset FIFO
*/
#define PEM_FIFO_START ((volatile UINT32 *)0x03000004) /* Start FIFO
*/
#define PEM_FIFO_ENABLE ((volatile UINT32 *)0x03000008) /* Enable FIFO
*/
#define PEM_STATUS ((volatile UINT32 *)0x0300000c) /* Status(Sampling, Calculation,
 DA Setup) */
#define PEM_PARAM_SAMPLE ((volatile UINT32 *)0x03000010) /* Sample Count
*/
#define PEM_PARAM_GUARD ((volatile UINT32 *)0x03000014) /* Guard Time
*/
#define PEM_PARAM_SETUP ((volatile UINT32 *)0x03000018) /* DA Setup Time
*/
#define PEM_COUNT ((volatile UINT32 *)0x0300001c) /* 2MHz Counter */
#define PEM_AD_READ ((volatile UINT32 *)0x03040000) /* AD Read Command
*/
#define PEM_REF_READ ((volatile UINT32 *)0x03080000) /* Reference Read Command
*/

/******* D/A Output Register (Lower 8 bit is valid.) *******/
#define PEM_DA1_LOW ((volatile UINT32 *)0x030c0000) /* DA Channel 1 Lower 8 bit
*/
#define PEM_DA1_HIGH ((volatile UINT32 *)0x030c0004) /* DA Channel 1 Higher 8 bit
*/
#define PEM_DA2_LOW ((volatile UINT32 *)0x030c0008) /* DA Channel 2 Lower 8 bit
*/
#define PEM_DA2_HIGH ((volatile UINT32 *)0x030c000c) /* DA Channel 2 Higher 8 bit
*/
#define PEM_DA3_LOW ((volatile UINT32 *)0x03100000) /* DA Channel 3 Lower 8 bit
*/
#define PEM_DA3_HIGH ((volatile UINT32 *)0x03100004) /* DA Channel 3 Higher 8 bit
*/
#define PEM_DA4_LOW ((volatile UINT32 *)0x03100008) /* DA Channel 4 Lower 8 bit
*/
```

```
#define PEM_DA4_HIGH ((volatile UINT32 *)0x0310000c) /* DA Channel 4 Higher 8 bit
*/
#define PEM_DA5_LOW ((volatile UINT32 *)0x03140000) /* DA Channel 5 Lower 8 bit
*/
#define PEM_DA5_HIGH ((volatile UINT32 *)0x03140004) /* DA Channel 5 Higher 8 bit
*/
#define PEM_DA6_LOW ((volatile UINT32 *)0x03140008) /* DA Channel 6 Lower 8 bit
*/
#define PEM_DA6_HIGH ((volatile UINT32 *)0x0314000c) /* DA Channel 6 Higher 8 bit
*/
#define PEM_DA7_LOW ((volatile UINT32 *)0x03180000) /* DA Channel 7 Lower 8 bit
*/
#define PEM_DA7_HIGH ((volatile UINT32 *)0x03180004) /* DA Channel 7 Higher 8 bit
*/
#define PEM_DA8_LOW ((volatile UINT32 *)0x03180008) /* DA Channel 8 Lower 8 bit
*/
#define PEM_DA8_HIGH ((volatile UINT32 *)0x0318000c) /* DA Channel 8 Higher 8 bit
*/

#define PEM_DA_OUTPUT ((volatile UINT32 *)0x031c0000) /* DA Output Command
*/

#define PEM_CLEAR ((volatile UINT32 *)0x031c0008) /* DA Output Command
*/
```

main.c

program name : main.c
path: source/dsp/src_modif/main.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSPプログラム                                */
/*                                                                           */
/* 内容 : メインモジュール                                                   */
/*                                                                           */
/*****************************************************************************/


#include "ftc6x.h" /* DSPボード用ヘッダ */
#include "command.h" /* エスパアンテナDSPプログラム用ヘッダ */
#include "esppem.h" /* A/D D/Aボード I/F用ヘッダ */
#include "mathf.h"
#include "eddy.h"

#define STAT_IDLE 0 /* アイドル状態 */
#define STAT_READY 1 /* 計測開始待ち状態 */
#define STAT_START 2 /* 計測中 */

#define VOLT_MAX 2047    /* maximun voltage */
#define VOLT_MIN -2048

volatile CMD_FORMAT *cmdToDsp; /* HOSTからDSPへのコマンド */
volatile CMD_FORMAT *cmdToHost; /* DSPからHOSTへのコマンド */
UINT32 pciHostMemAddr; /* HOSTバッファのPCI物理アドレス */
UINT32 pciHostMemSize; /* HOSTバッファのサイズ */
INIT_PARAM param; /* 設定パラメータフォーマット */
MEASURED_DATA *sendMeasure; /* データ転送コマンドフォーマット */
UINT32 fifoDetected;
```

```
UINT32 poseFlag; /* 一時停止フラグ */

extern void Initialize(void); /* 初期化 */
extern void LoadParam(void); /* パラメータ設定 */
extern void OutputVoltage(void);/* 制御電圧初期設定 */
extern void DAOutput(int []); /* 制御電圧設定 */
extern void DAsend(int []); /* 制御電圧データ転送 */
extern void Response(RESULT); /* 初期化終了コマンド */
extern float Measure(int , int);
void interrupt IntPem();

main()
{
RESULT rv; /* DSP ボードサポートライブラリの返り値 */
UINT32 status = STAT_IDLE; /* DSP ステート */
UINT32 *read = (UINT32 *)0x80008000;

int m, n, k;
int i;

float corr[9]; /* 相関係数 */
float DeltaRho[9];
#if (NORMALIZE == 1)
float ValueTmp = 0;
float DivCoef = 0;
#endif
int Xn[9]; /* 制御電圧 */
/* int Xtmp;  */
int mem; /* 摂動前の制御電圧 */

m = 0; /* エレメント数初期化 */
n = 1; /* フレーム数初期化 */

fifoDetected = FALSE;
poseFlag = FALSE;

for(k = 0; k < 9; k++) {
 DeltaRho[k]= 0;
}


/******* Daytona の初期化 *******/
/* C6201 を初期化 */
```

```
rv = C6x_OpenC6x(NO_FLAGS);
while(rv != 0);

/* Hurricaneを初期化 */
rv = C6x_OpenHurricane(NO_FLAGS);
while(rv != 0);

/* PEMのアクセスタイミングを設定 */
*C6X_EMIF_CE3CR_PTR = FT_EMIF_CE3_REG_VAL;

/* DSPへのコマンドを受けるアドレスを設定 */
cmdToDsp  = (CMD_FORMAT *)0x00400000;

/* HOSTへのコマンドを受けるアドレスを設定 */
cmdToHost = (CMD_FORMAT *)0x00410000;


/******* コマンド検出 *******/
while(1){
/* コマンド検出 */
if(CMD_VALID == cmdToDsp->cmdFlag){
/* 状態遷移 */
switch(status){
/* 初期化待ち */
case STAT_IDLE:
/* 初期化コマンド検出 */
if(CMD_DSP_INIT == cmdToDsp->cmdId){
status = STAT_READY; /* 状態 -> 計測開始待ち */
Initialize(); /* 初期化 */
LoadParam(); /* パラメータをロード */
DataSend(); /* データSSRAM書き込み */

for(k = 0; k < 8; k++) {
 Xn[k]= 0;
}

for(k = 0; k < param.element; k++) {
 Xn[k]= param.voltage[k];
}

OutputVoltage(); /* 制御電圧をDACに出力 */
Response(rv); /* 初期化完了をHostに応答 */
}
```

```
break;

/* 計測開始待ち */
case STAT_READY:
/* 初期化コマンド検出 */
if(CMD_DSP_INIT == cmdToDsp->cmdId){
Initialize(); /* 初期化 */
LoadParam(); /* パラメータをロード */
DataSend(); /* データ転送 */

for(k = 0; k < param.element; k++) {
 Xn[k]= param.voltage[k];
}

OutputVoltage(); /* 制御電圧を DAC に出力 */
Response(rv); /* 初期化完了を Host に応答 */
}
/* 計測開始コマンド検出 */
else if(CMD_DSP_START == cmdToDsp->cmdId){
status = STAT_START; /* 状態 -> 計測中 */
StartMeasure(); /* 計測開始指令 */
m = 0; /* エレメント数初期化 */
n = 1; /* フレーム数初期化 */
}

break;

/* 計測中 */
case STAT_START:
/* 計測停止コマンド検出 */
if(CMD_DSP_STOP == cmdToDsp->cmdId){
status = STAT_READY; /* 状態 -> 計測開始待ち */
StopMeasure(); /* 計測停止指令 */
}
/* 計測一時停止コマンド検出 */
else if(CMD_DSP_POSE == cmdToDsp->cmdId && !poseFlag){
/*PoseMeasure();*/ /* 計測一時停止指令 */
poseFlag = TRUE; /* 一時停止フラグをセット */
}
/* 計測再開コマンド検出 */
else if(CMD_DSP_RESUME == cmdToDsp->cmdId && poseFlag){
/*ResumeMeasure();*/ /* 計測再開指令 */
```

```
poseFlag = FALSE; /* 一時停止フラグをリセット */
}

break;

/* 状態不定 */
default:
status = STAT_IDLE; /* 状態 -> 初期化待ち */
break;
}

/* コマンド無効 */
cmdToDsp->cmdFlag = 0;
}

/* FIFO からの割り込み検出、および計測中状態 */
if(fifoDetected && (STAT_START == status)){
if(n == 1 && m == 0){
for(i = 0; i < 2; i++) {
*read = *PEM_AD_READ;
}
*PEM_PARAM_SAMPLE = param.sample - 1; /* サンプル数設定 */
}

if(n <= param.updateCount){
if(m < param.element){
if(m != 0) {
Xn[m-1] = mem; /* 摂動前の制御電圧に戻す */
}
DAsend(Xn); /* 制御電圧を SSRAM に出力 */
corr[m] = Measure(n, m); /* 相関係数演算 */
mem = Xn[m]; /* 摂動前制御電圧保存 */
Xn[m] = param.DeltaX + Xn[m]; /* 摂動 */

/* if (Xtmp > VOLT_MAX)
  Xn[m] = param.DeltaX - Xn[m];
else
  Xn[m] = Xtmp;    */


DAOutput(Xn); /* 制御電圧を DAC に出力 */

m++; /* エレメント数インクリメント */
```

```
}
else {
Xn[m-1] = mem; /* 摂動前の制御電圧に戻す */
DAsend(Xn); /* 制御電圧を SSRAM に出力 */
corr[m] = Measure(n, m); /* 相関係数演算 */

/* ------------------------------------------------------------ */
#if (NORMALIZE == 1)

ValueTmp = 0;
for (k = 1;k <= param.element; k++) {
/* 相関係数差分の算出 */
DeltaRho[k] = corr[k] - corr[0];
/* 制御電圧の更新 */

#if (NORM_VERSION == 1)
/* first version */
ValueTmp = ValueTmp + corr[k]*corr[k];
#elif (NORM_VERSION == 2)
/* second version */
ValueTmp = ValueTmp + DeltaRho[k]*DeltaRho[k];
#else
       /* default version */
       ValueTmp = ValueTmp + corr[k]*corr[k];
#endif
}
DivCoef = sqrtf(ValueTmp/((float)param.element));

for (k = 1;k <= param.element; k++) {
#if((COST_FUNCTION == POWMINUS) || (COST_FUNCTION == POWINVMINUS))
Xn[k-1] = Xn[k-1] - (int)(param.step * (DeltaRho[k]/DivCoef));
#else
Xn[k-1] = Xn[k-1] + (int)(param.step * (DeltaRho[k]/DivCoef));
#endif

  }
/* ------------------------------------------------------------ */
#else

for(k = 1; k <= param.element; k++) {
/* 相関係数差分の算出 */
DeltaRho[k] = corr[k] - corr[0];
/* 制御電圧の更新 */
```

```
#if((COST_FUNCTION == POWMINUS) || (COST_FUNCTION == POWINVMINUS))
Xn[k-1] = Xn[k-1] - (int)(param.step * DeltaRho[k]);
#else
Xn[k-1] = Xn[k-1] + (int)(param.step * DeltaRho[k]);
#endif
}
#endif
/* ------------------------------------------------------------ */

    for (k = 0; k <= param.element; k++) {
     if (Xn[k] > VOLT_MAX)
     Xn[k] = VOLT_MAX;
     else if (Xn[k] < VOLT_MIN)
     Xn[k] = VOLT_MIN;
    }

DAOutput(Xn); /* 制御電圧をDACに出力 */
n++; /* フレーム数インクリメント */
m = 0; /* エレメント数リセット */
}
}

/* 計測完了 */
if(n == (param.updateCount + 1)){
status = STAT_READY; /* 状態 -> 計測開始待ち */
n = 0; /* フレーム数リセット */
*PEM_FIFO_START = 0x00; /* A/DD/Aボードの FIFOデータ取り込み停止 */
for(i = 0; i < 2; i++);
}
fifoDetected = FALSE;
}
}
}

void interrupt IntPem()
{
while((*PEM_STATUS & 0x3) == 0x01);

  fifoDetected = TRUE;
}
```

measure.c

program name : measure.c
path: source/dsp/src_modif/measure.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 相関係数演算モジュール                                            */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "command.h"
#include "esppem.h"
#include "eddy.h"
#include <mathf.h>

struct IQ       y_data[1024]; /*  data */
struct IQ       r_data[1024]; /*  data */

extern CMD_FORMAT *cmdToHost;
extern INIT_PARAM param;
extern UINT32 pciHostMemAddr;
extern MEASURED_DATA *sendMeasure;
extern UINT32 poseFlag; /* 一時停止フラグ */

int buffer_y[1024];
int buffer_r[1024];

#define MAX_INTERNAL_BUF 1024

struct {
```

```
unsigned int input_data : 1;
} flag;

float Measure(int n, int m)
{

int i;
int p;
float mecorr = 0;
RESULT rv;
UINT32 *cmdBufY;
UINT32 *cmdBufR;
UINT32 length = param.sample;
UINT32 datalen = param.sample;
UINT32 DataLength;

float a = 0;
float b = 0;
/* float c = 0;
float d = 0;
*/ float y = 0;
float z = 0;
/* float x = 0;
*/
float sqrt_x;
float sqrt_y;
float sqrt_z;

#if((COST_FUNCTION == POWINV) || (COST_FUNCTION == POWINVMINUS))
float intermd; /* for use of power inverse */
#endif

sendMeasure = (MEASURED_DATA *)&cmdToHost->cmdData[1];
cmdBufY = (UINT32 *)sendMeasure->inputData; /* SSRAMのポインタ */
cmdBufR = (UINT32 *)&sendMeasure->inputData[param.sample]; /* SSRAM
のポインタ */

flag.input_data = (param.savedData & 0x04) >> 2;

if(flag.input_data == 1) {
DataLength = param.sample * 2 + 2;
}
else if(flag.input_data == 0) {
```

```
DataLength = 0;
}

sendMeasure->frameNo = n; /* フレームNo. */
sendMeasure->elementNo = m; /* エレメントNo. */


/* only y measurement */

while(MAX_INTERNAL_BUF < length){
/* PEM(SDRAM)から内部RAMへデータを転送  */
rv = C6x_SetUpC6xDma((UINT32 *)buffer_y, (UINT32 *)PEM_AD_READ, MAX_INTERNAL_BUF,
C6X_DMA_PEM_FLAGS);
while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);
/* 内部RAMからSSRAMへデータを転送 */
rv = C6x_SetUpC6xDma(cmdBufY, (UINT32 *)buffer_y, MAX_INTERNAL_BUF,
C6X_DMA_FLAGS);
while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);

/* PEM(SDRAM)から内部RAMへデータを転送  */
/* rv = C6x_SetUpC6xDma((UINT32 *)buffer_r, (UINT32 *)PEM_REF_READ, MAX_INTERNAL_BUF,
 C6X_DMA_PEM_FLAGS);
  while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);
*/ /* 内部RAMからSSRAMへデータを転送 */
/* rv = C6x_SetUpC6xDma(cmdBufR, (UINT32 *)buffer_r, MAX_INTERNAL_BUF,
C6X_DMA_FLAGS);
while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);
*/
/* SSRAMのバッファポインタを更新 */
cmdBufY += MAX_INTERNAL_BUF;
cmdBufR += MAX_INTERNAL_BUF;

/* データ長を更新 */
length -= MAX_INTERNAL_BUF;

for(i = 0; i < MAX_INTERNAL_BUF; i++){

y_data[i].q = -((float)((buffer_y[i] & 0xffff) >> 4) - 2048) - OFFSET_I;
y_data[i].i = -((float)((buffer_y[i] & 0xfff00000) >> 20) - 2048) - OFFSET_Q;

/* r_data[i].q = (float)((buffer_r[i] & 0xffff) >> 4);
r_data[i].i = (float)(buffer_r[i] >> 20);
*/
```

```
}

for (p = 0; p < MAX_INTERNAL_BUF; p++){

a += (y_data[p].i * y_data[p].i + y_data[p].q * y_data[p].q);
/* b += (y_data[p].i * r_data[p].q - y_data[p].q * r_data[p].i);   */
/* y += (y_data[p].i * y_data[p].i + y_data[p].q * y_data[p].q);   */
/* z += (r_data[p].i * r_data[p].i + r_data[p].q * r_data[p].q);   */

}
}

if(length > 0){

/* PEM(SDRAM) から内部 RAM へデータを転送 */
rv = C6x_SetUpC6xDma((UINT32 *)buffer_y, (UINT32 *)PEM_AD_READ, length,
 C6X_DMA_PEM_FLAGS);
while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);
/* 内部 RAM から SSRAM へデータを転送 */
rv = C6x_SetUpC6xDma(cmdBufY, (UINT32 *)buffer_y, length,
 C6X_DMA_FLAGS);
while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);

/* PEM(SDRAM) から内部 RAM へデータを転送 */
/* rv = C6x_SetUpC6xDma((UINT32 *)buffer_r, (UINT32 *)PEM_REF_READ,
 length, C6X_DMA_PEM_FLAGS);
  while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);
*/ /* 内部 RAM から SSRAM へデータを転送 */
/* rv = C6x_SetUpC6xDma(cmdBufR, (UINT32 *)buffer_r,
 length, C6X_DMA_FLAGS);
while(*C6X_DMA_CH0_PRIMCR_PTR & C6X_DMA_CH_PRIMCR_STATUS_MASK);
*/
}

for(i = 0; i < length; i++){

y_data[i].q = -((float)((buffer_y[i] & 0xffff) >> 4) - 2048) - OFFSET_I;
y_data[i].i = -((float)((buffer_y[i] & 0xfff00000) >> 20) - 2048) - OFFSET_Q;

/* r_data[i].q = (float)((buffer_r[i] & 0xffff) >> 4);
r_data[i].i = (float)(buffer_r[i] >> 20);
*/
}
```

```
for (p = 0; p < length; p++){

a += (y_data[p].i * y_data[p].i + y_data[p].q * y_data[p].q);
/* b += (y_data[p].i * r_data[p].q - y_data[p].q * r_data[p].i);    */
/* y += (y_data[p].i * y_data[p].i + y_data[p].q * y_data[p].q);    */
/* z += (r_data[p].i * r_data[p].i + r_data[p].q * r_data[p].q);    */

}

/*
c = a * a;
d = b * b;
x = c + d;

sqrt_x = sqrtf(x) / param.sample;
sqrt_y = sqrtf(y / param.sample);
sqrt_z = sqrtf(z / param.sample);

sqrt_x = sqrtf((a * a) + (b * b)) / datalen;
sqrt_y = sqrtf(y / datalen);
sqrt_z = sqrtf(z / datalen);

mecorr = (sqrt_x / (sqrt_y * sqrt_z));
*/

#if((COST_FUNCTION == POWINV) || (COST_FUNCTION == POWINVMINUS))
/*  use of power inverse */
intermd = a/((float)(1<<SHIFT_COEF));
    mecorr = param.sample/intermd;

#elif((COST_FUNCTION == POWER) || (COST_FUNCTION == POWMINUS))
    mecorr = a/((float)(param.sample<<SHIFT_COEF));

    /* by default use of power */
#else
    mecorr = a/((float)(param.sample<<SHIFT_COEF));
#endif

    sendMeasure->coeffData = mecorr;

  cmdToHost->cmdFlag    = 0x00;
cmdToHost->cmdId      = CMD_DATA_SEND;
```

```
cmdToHost->cmdWords   = DataLength + 16;
cmdToHost->cmdData[0] = 0x0;

rv = C6x_SetUpHurricaneDma(
(UINT32 *)pciHostMemAddr,
(UINT32 *)((UINT32)cmdToHost & 0x003fffff),
DataLength + 4 + 16,
FT_C6X_SETUPHURCDMA_DIR_LOC_TO_PCI
| FT_C6X_SETUPHURCDMA_PCI_MEM_CYCLE
| FT_C6X_SETUPHURCDMA_BLOCK_UNTIL_DONE );

  cmdToHost->cmdFlag    = CMD_VALID;

while(poseFlag == TRUE);

rv = C6x_SetUpHurricaneDma(
(UINT32 *)pciHostMemAddr,
(UINT32 *)((UINT32)cmdToHost & 0x003fffff),
1,
FT_C6X_SETUPHURCDMA_DIR_LOC_TO_PCI
| FT_C6X_SETUPHURCDMA_PCI_MEM_CYCLE
| FT_C6X_SETUPHURCDMA_BLOCK_UNTIL_DONE );

    return(mecorr);

}
```

outputvoltage.c

program name : outputvoltage.c
path: source/dsp/src_modif/outputvoltage.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 制御電圧設定モジュール                                            */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "command.h"
#include "esppem.h"

#define WAIT_COUNT 1000

extern INIT_PARAM param;

void OutputVoltage(void)
{

INT32 *voltage = param.voltage;
UINT32 output;
UINT32 i;

output = voltage[0] + 2048;
*PEM_DA2_LOW = output;
*PEM_DA2_HIGH = (output >> 8);

output = voltage[1] + 2048;
*PEM_DA1_HIGH = (output >> 8);
```

```
*PEM_DA1_LOW = output;

output = voltage[2] + 2048;
*PEM_DA3_LOW = output;
*PEM_DA3_HIGH = (output >> 8);

output = voltage[3] + 2048;
*PEM_DA4_LOW = output;
*PEM_DA4_HIGH = (output >> 8);

output = voltage[4] + 2048;
*PEM_DA5_LOW = output;
*PEM_DA5_HIGH = (output >> 8);

output = voltage[5] + 2048;
*PEM_DA6_LOW = output;
*PEM_DA6_HIGH = (output >> 8);

/* output = voltage[6] + 2048;*/ /* 2001.06.21 */
*PEM_DA7_LOW = output;
*PEM_DA7_HIGH = (output >> 8);

/* output = voltage[7] + 2048;*/ /* 2001.06.21 */
*PEM_DA8_LOW = output;
*PEM_DA8_HIGH = (output >> 8);

*PEM_DA_OUTPUT = 0x0;
for(i = 0; i < WAIT_COUNT; i++);
}
```

posemeasure.c

program name : posemeasure.c
path: source/dsp/src_modif/posemeasure.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 計測再開/一時停止モジュール                                       */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "esppem.h" /* A/D D/A ボード I/F 用ヘッダ */

void PoseMeasure(void)
{

/* RESULT rv;
rv = C6x_ControlDisableC6xInt(C6201_IxR_EXT_INT5);
while(rv != 0);
*/
int i;

*PEM_FIFO_START = 0x00;
for(i = 0; i < 2; i++);

}

void ResumeMeasure(void)
{
/* RESULT rv;
```

```
rv = C6x_ControlEnableC6xInt(C6201_IxR_EXT_INT5);
while(rv != 0);
*/
int i;

*PEM_FIFO_START = 0x01;
for(i = 0; i < 2; i++);

}
```

initialize.c

program name : initialize.c
path: source/dsp/src_modif/initialize.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 初期化、パラメータ設定モジュール                                  */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "command.h"
#include "esppem.h"

extern CMD_FORMAT *cmdToDsp;
extern UINT32 pciHostMemAddr;
extern UINT32 pciHostMemSize;
extern INIT_PARAM param;

void Initialize()
{

int i;

*PEM_FIFO_RESET = 0x01; /* FIFO リセット */
for(i = 0; i < 200; i++);

}

void LoadParam (void)
{
```

```
pciHostMemAddr = cmdToDsp->cmdData[0]; /* PCI 物理アドレス設定 */
pciHostMemSize = cmdToDsp->cmdData[1]; /* PCI メモリサイズ設定 */
param = *((INIT_PARAM *)&cmdToDsp->cmdData[2]); /* 各パラメータ設定 */

/* PEM の初期化 */
*PEM_PARAM_GUARD = param.guardTime - 1; /* ガードタイム設定 */
*PEM_PARAM_SETUP = param.daSetup - 1; /* D/A Setup タイム設定*/

}
```

response.c

program name : response.c
path: source/dsp/src_modif/response.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 初期化に対する応答コマンド送信モジュール          */
/*                                                                           */
/*****************************************************************************/

#include "command.h"
#include "ftc6x.h"

extern CMD_FORMAT *cmdToHost;
extern UINT32 pciHostMemAddr;

void Response(RESULT Status)
{
RESULT rv;

cmdToHost->cmdFlag    = CMD_VALID;
cmdToHost->cmdId      = CMD_INIT_RES;
cmdToHost->cmdWords   = 1;
cmdToHost->cmdData[0] = Status;


/* HOST への応答コマンドを DMA 転送する */
rv = C6x_SetUpHurricaneDma(
(UINT32 *)pciHostMemAddr,  /* PCI 物理アドレス */
(UINT32 *)((UINT32)cmdToHost & 0x003fffff),  /* SSRAMのローカルアド
レス */
```

```
4,/* 転送ワード数 */
FT_C6X_SETUPHURCDMA_DIR_LOC_TO_PCI  /* 転送方向 DSP to HOST */
| FT_C6X_SETUPHURCDMA_PCI_MEM_CYCLE /* サイクルタイプ */
| FT_C6X_SETUPHURCDMA_BLOCK_UNTIL_DONE ); /* DMA 転送が完了するまで
WAIT する */

while(rv != 0);

}
```

startmeasure.c

program name : startmeasure.c
path: source/dsp/src_modif/startmeasure.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 計測開始モジュール                                                */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "command.h"
#include "esppem.h"

extern CMD_FORMAT *cmdToDsp;
extern INIT_PARAM param;

void StartMeasure(void)
{

RESULT rv; /* DSP ボードサポートライブラリの返り値 */
int i;

param = *((INIT_PARAM *)&cmdToDsp->cmdData[2]); /* 各パラメータ設定 */

/* FIFO リセット */
*PEM_FIFO_RESET = 0x01;
for(i = 0; i < 2; i++);

/* PEM からの割り込みをイネーブル */
rv = C6x_ControlEnableIntSrc(FT_INT_EINT5, FT_INTSRC_PEM_INT1_EINT5);
```

```
while(rv != 0);

rv = C6x_ControlEnableC6xInt(C6X_CSR_GIE | C6201_IxR_EXT_INT5);
while(rv != 0);

*PEM_PARAM_SAMPLE = param.sample + 1; /* サンプル数設定 */

/* FIFO取り込み開始 */
*PEM_FIFO_START = 0x01;
for(i = 0; i < 2; i++);



}
```

stopmeasure.c

program name : stopmeasure.c
path: source/dsp/src_modif/stopmeasure.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 計測停止モジュール                                                */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "esppem.h"

void StopMeasure(void)
{

RESULT rv; /* DSP ボードサポートライブラリの返り値 */
int i;

  rv = C6x_ControlDisableIntSrc(FT_INT_EINT5,  (FT_PEM_CTRL_PIE1_5 | FT_PEM_CTRL_PIE2_5) );
while(rv != 0);

rv =C6x_ControlDisableC6xInt(C6201_IxR_EXT_INT5);
while(rv != 0);

*PEM_FIFO_START = 0x00; /* FIFO データ取り込み停止 */
for(i = 0; i < 2; i++);

}
```

DAOutput.c

program name : DAOutput.c
path: source/dsp/src_modif/DAOutput.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                              エスパアンテナ制御装置                       */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                                  DSP プログラム                           */
/*                                                                           */
/* 内容 ： 制御電圧設定モジュール                                            */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "esppem.h"

#define WAIT 1
void DAOutput(int Xn[])
{

UINT32 output;
UINT32 i;

output = Xn[0] + 2048;
*PEM_DA1_HIGH = (output >> 8);
*PEM_DA1_LOW = output;
for(i = 0; i < WAIT; i++);

output = Xn[1] + 2048;
*PEM_DA2_HIGH = (output >> 8);
*PEM_DA2_LOW = output;

output = Xn[2] + 2048;
*PEM_DA3_HIGH = (output >> 8);
```

```
*PEM_DA3_LOW = output;
for(i = 0; i < WAIT; i++);

output = Xn[3] + 2048;
*PEM_DA4_HIGH = (output >> 8);
*PEM_DA4_LOW = output;

output = Xn[4] + 2048;
*PEM_DA5_HIGH = (output >> 8);
*PEM_DA5_LOW = output;
for(i = 0; i < WAIT; i++);

output = Xn[5] + 2048;
*PEM_DA6_HIGH = (output >> 8);
*PEM_DA6_LOW = output;

/* output = Xn[6] + 2048;*/ /* 2001.06.21 */
*PEM_DA7_HIGH = (output >> 8);
*PEM_DA7_LOW = output;
for(i = 0; i < WAIT; i++);

/* output = Xn[7] + 2048;*/ /* 2001.06.21 */
*PEM_DA8_HIGH = (output >> 8);
*PEM_DA8_LOW = output;

}
```

DAsend.c

program name : DAsend.c
path: source/dsp/src_modif/DAsend.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 ： 制御電圧値転送モジュール                                          */
/*                                                                           */
/*****************************************************************************/

#include "ftc6x.h"
#include "command.h"

extern CMD_FORMAT *cmdToHost;
extern INIT_PARAM param;
extern MEASURED_DATA *sendMeasure;

void DAsend(int Xn[])
{

int k;

sendMeasure = (MEASURED_DATA *)&cmdToHost->cmdData[1];

for(k = 0; k < param.element; k++) {
sendMeasure->voltData[k] = Xn[k];
}

if(param.element < 8)
for(k = param.element; k < 8; k++) {
sendMeasure->voltData[k] = 0;
```

```
}
}
```

DataSend.c

program name : DataSend.c
path: source/dsp/src_modif/DataSend.c
comments :

```
/*****************************************************************************/
/*                                                                           */
/*                          エスパアンテナ制御装置                           */
/*                                                                           */
/*****************************************************************************/

/*****************************************************************************/
/*                                                                           */
/*                              DSP プログラム                               */
/*                                                                           */
/* 内容 : HOST へのデータ転送モジュール                                      */
/*                                                                           */
/*****************************************************************************/

#include "command.h"
#include "ftc6x.h"

extern CMD_FORMAT *cmdToHost;
extern INIT_PARAM param;
extern MEASURED_DATA *sendMeasure;


struct {
unsigned int sel_data : 1;
unsigned int sel_vol  : 1;
unsigned int sel_corr : 1;
} flags;


void DataSend(void)
{

sendMeasure = (MEASURED_DATA *)&cmdToHost->cmdData[1];

flags.sel_corr = param.savedData;
```

```
flags.sel_vol  = (param.savedData & 0x02) >> 1;
flags.sel_data = (param.savedData & 0x04) >> 2;

sendMeasure->coeffId = 1; /* 相関係数識別 ID */
sendMeasure->coeffWords = 1; /* 相関係数データワード数 */

sendMeasure->voltId = 2; /* 制御電圧識別 ID */
sendMeasure->voltWords = 8; /* 制御電圧データワード数 */

sendMeasure->inputId = 3; /* 入力信号識別 ID */

if(flags.sel_data == 1) { /* 入力信号データワード数 */
sendMeasure->inputWords = param.sample * 2;
}
else if(flags.sel_data == 0) {
sendMeasure->inputWords = 0;
}

}
```

Appendix C : ESPAR control source code (solution 2)

comhead.h

program name : comhead.h
path: source/antennaGG2/comhead.h
comments : General declarations header

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#ifndef COMHEAD_H
#define COMHEAD_H

#define ATR_VER 1 /* ATR Network Analyser */
#define AGG_VER 2 /* Antenna GIGEN Network Analyser */

/* ------------------------------------------------------------------------------ *
/* CHOOSE HERE THE VERSION YOU WANT TO USE */
#define NA_VERSION AGG_VER

/* ------------------------------------------------------------------------------ *
#if (NA_VERSION==AGG_VER)
#define TIMESOVERMSEC 2000 /* waiting time settings for Antenna Gigen */
#else /* waiting time settings for ATR */
#define TIMESOVERMSEC 3000 /* old 1420-3000 : depends of the PC clock */
#endif
/* ------------------------------------------------------------------------------ *

/* pause define */
```

```
#define pause {char c;fflush(stdin);printf("Press ENTER key ...\n");c = getchar();}

/* debugging define */
#if 0
#define DB(A) {A;}
#define DEBUG_OUT stdout
#define DB_pause pause
#else
#define DB(A) {A;}
#define DB_pause {}
#define DEBUG_OUT _debug_file
#endif

/* ------------------------------------------------------------------------------ */
/* TEMPORISATION A miliseconds */
#if 1
#define Idling(A) {int Ts=TIMESOVERMSEC,i,ii;for(i=1;i<A;i++) for(ii=1;ii<Ts;ii++);}
#else
#define Idling(A) {}
#endif
/* ---------------------------------------------------------------- */

float *String2FloatTable (char *str);
void recopy_f2f (int n,float *f1,float *f2);

#endif
```

main.c

program name : main.c
path: source/antennaGG2/main.c
comments : Main program

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

#include <windows.h>
#include "decl-32.h"
#include "apiaio.h"

#include "comhead.h"
#include "gpib.h"
#include "dalink.h"
#include "adaptive.h"
#include "mfunc.h"

/* external global variables declarations */
extern WORD Ch_Def[];
extern WORD Output_Range[];
extern FILE *_debug_file;

/* GLOBAL variables declarations */
int OPT_OUTPUT = 0;
int OPT_ADPALGO = 0;
int OPT_VOLTEXP = 0;
int OPT_FREQEXP = 0;
int OPT_OMNIP  = 0;

#define XOR(A,B) ( ((A) && (!B)) || ((!A) && (B)) )

#define CHECK_OPT XOR(OPT_OMNIP==1,XOR(OPT_OUTPUT==1,XOR(OPT_ADPALGO==1,XOR(OPT_FREQEXP=

int main (int argc,char *argv[])
```

```
{

char file_name[256]= {'\0'};

/* D/A main variables */
HANDLE hdrv;
WORD grpno = (WORD)GRPNO;
DWORD dwret;

/* D/A output variables */
WORD  outbuf[NBVOLTAGE] = {0};
AOUT ipaout;

/* GPIB variable(s)*/
int Dev;

/* -------- USER VARIABLES ------------- */
struct param param;
FILE *fp1 = NULL;

/* ------------------------------------ */

/* ------------------------------------------------------------------------
   --------- ARGUMENTS CHECK
   ------------------------------------------------------------------------ */

if ( argc > 1 )
{
while ( --argc )
{
if ( !strncmp(argv[argc],"--help",6) || !strncmp(argv[argc],"-h",2) )
{
printf("Use : %s [--output|--adpalgo|--voltexp|--freqexp|--omni] [--file=<file name>]\n",
argv[0]);
printf("\t --output    : simple D/A output value\n");
printf("\t --adpalgo   : use adaptive algorithm\n");
printf("\t --voltexp   : fix 5 voltages and measure power for 1 variable voltage\n");
printf("\t --freqexp   : fix all voltages and measure power for variable frequence\n");
printf("\t --omni      : measure power for all voltage variation between an input range\n")
printf("\t <file name> : name of data file, for '--voltexp', '--freqexp', '--omni'
and '--adpalgo'\n");
printf("\n");
exit(1);
```

```
}
else if ( !strncmp(argv[argc],"--file=",7) )
sscanf (argv[argc],"--file=%s",file_name);

else if ( !strncmp(argv[argc],"--output",8) )
OPT_OUTPUT = 1;

else if ( !strncmp(argv[argc],"--adpalgo",9) )
OPT_ADPALGO = 1;

else if ( !strncmp(argv[argc],"--voltexp",9) )
OPT_VOLTEXP = 1;

else if ( !strncmp(argv[argc],"--freqexp",9) )
OPT_FREQEXP = 1;

else if ( !strncmp(argv[argc],"--omni",6) )
OPT_OMNIP = 1;

else
{
fprintf(stderr,"%s : Unknown option %s \n",argv[0],argv[argc]);
fprintf(stderr,"Try `%s --help' for more information \n",argv[0]);
exit(1);
};
};
}
else
{
fprintf(stderr,"%s : not enough arguments \n",argv[0]);
fprintf(stderr,"Try `%s --help' for more information \n",argv[0]);
exit(1);
};

/* check the options */
if (!CHECK_OPT)
{
fprintf(stderr,"%s : can not use several options in the same time \n",argv[0]);
exit(1);
}

/* check the options dependances */
if ((OPT_ADPALGO==1 || OPT_VOLTEXP==1 || OPT_FREQEXP==1 || OPT_OMNIP==1)
```

```
&& strlen(file_name)==0)
{
fprintf(stderr,"%s : missing file name (option '--file=') \n",argv[0]);
fprintf(stderr,"Try '%s --help' for more information \n",argv[0]);
exit(1);
};


/* ------------------------------------------------------------------------
   --------- INIT
   ------------------------------------------------------------------------ */
printf("--------------- Initializing --------------------\n");

/* create GPIB debug file output */
_debug_file = fopen("gpib_debug_file.txt","w");
if (_debug_file == NULL)
{
fprintf(stderr,"Error : can not create %s\n","gpib_debug_file.txt");
exit(1);
};

/* Initialize GPIB */
Dev = NA_gpib_init();
// Dev = gpib_init_prompt();

/* Open D/A board */
    dwret = 1;
/* looking for group number */
while ((dwret != 0) && grpno<=16)
{
dwret = AioOpen(&hdrv,(WORD)DRVNO,++grpno);
printf("D/A board (group number:%d) [0x%xh] .... %s \n",grpno,dwret,(dwret==0)?"OK":"Error"
};
if (dwret != 0)
{
printf("Error : can not open D/A board (errno:0x%xh)\n",dwret);
exit(1);
}

/* set the output range */
AioRange(hdrv,Ch_Def,Output_Range,NBVOLTAGE);

/* settings for output commands */
```

```
ipaout.OutBuf = outbuf;
ipaout.OutChNo = Ch_Def;
ipaout.Channels = NBVOLTAGE;

/* output 0v in all channel set*/
{
float volt[MAXCHANNEL] = {0.0};
ConvertVolt2Binary(ipaout.Channels,volt,outbuf);
}; /* after outbf is 0v binary values */
dwret = AioOut(hdrv,&ipaout);
if (dwret!=0)
printf("Error in: AioOut(hdrv,&ipaout) [0x%xh]\n",dwret);
else
printf("Init output succeded ...\n");

printf("--------------- End initialize ------------------\n\n");

/* ------------------------------------------------------------------------ */

/* ------------------------------------------------------------------------
   --------- MAIN
   ------------------------------------------------------------------------ */

if (OPT_OUTPUT==1)
{
/* case of simple D/A output */
char ch[2] = "Y";

while(ch[0]=='Y' || ch[0]=='y')
{
Simple_Output_Test(hdrv);
printf("Do you want to 'output' again ? (y/n):");
scanf("%s",ch);
};

}
else if (OPT_ADPALGO==1)
{
/* prompting params */
param = PromptValues ();
// param = DefaultParamValues ();
ipaout.Channels = (WORD)param.element;
```

```
/* set init of GPIB */
#if (NA_VERSION==ATR_VER)
NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_WIN,GPIB_TRACE,param.P,
(float)GPIB_FREQ,GPIB_POWL,GPIB_S_MEAS);
#else
AGG_NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_POWL,AGG_NA_POINT,3000,1,
(float)GPIB_FREQ,0,GPIB_S_MEAS);
#endif

/* adaptive algo */
fp1 = fopen(file_name,"w");

Print_Param_Values (fp1,param);
AdptiveAlgo_ver2(fp1,Dev,GPIB_WIN_CHANNEL,hdrv,&ipaout,param);
fclose(fp1);
}

else if (OPT_VOLTEXP==1)
{
/* set init of GPIB */
#if (NA_VERSION==ATR_VER)
NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_WIN,GPIB_TRACE,param.P,
(float)GPIB_FREQ,GPIB_POWL,GPIB_S_MEAS);
#else
AGG_NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_POWL,AGG_NA_POINT,3000,1,
(float)GPIB_FREQ,0,GPIB_S_MEAS);
#endif
fp1 = fopen(file_name,"w");
Voltage_Power_Measure (Dev,GPIB_WIN_CHANNEL,hdrv,fp1);
fclose(fp1);

}

else if (OPT_OMNIP==1)
{
/* set init of GPIB */
#if (NA_VERSION==ATR_VER)
NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_WIN,GPIB_TRACE,param.P,
(float)GPIB_FREQ,GPIB_POWL,GPIB_S_MEAS);
#else
AGG_NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_POWL,AGG_NA_POINT,3000,1,
(float)GPIB_FREQ,0,GPIB_S_MEAS);
#endif
```

```
fp1 = fopen(file_name,"w");
Voltage_Power_Measure_all (Dev,GPIB_WIN_CHANNEL,hdrv,fp1);
fclose(fp1);

}

else if (OPT_FREQEXP==1)
{
/* set init of GPIB */
#if (NA_VERSION==ATR_VER)
NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_WIN,GPIB_TRACE,param.P,
(float)GPIB_FREQ,GPIB_POWL,GPIB_S_MEAS);
#else
AGG_NA_GPIBfreq_init(Dev,GPIB_WIN_CHANNEL,GPIB_POWL,AGG_NA_POINT,3000,1,
(float)GPIB_FREQ,0,GPIB_S_MEAS);
#endif
fp1 = fopen(file_name,"w");
Frequence_Power_Measure (Dev,GPIB_WIN_CHANNEL,hdrv,fp1,(float)GPIB_FREQ);
fclose(fp1);

}


/* ------------------------------------------------------------------------ */

/* ------------------------------------------------------------------------
   --------- CLOSE
   ------------------------------------------------------------------------ */
fclose(_debug_file);
gpib_close (Dev);
dwret = AioClose(hdrv);
if (dwret != 0)
printf("Error : can not close D/A board (errno:0x%xh)\n",dwret);
exit(1);
/* ------------------------------------------------------------------------ */

}
```

adaptive.h

program name : adaptive.h
path: source/antennaGG2/adaptive.h
comments : Header for the adaptive algorithm function declarations

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "apiaio.h"
#include "dalink.h"

#ifndef ADAPTIVE_H
#define ADAPTIVE_H

struct param {
int N; /* nb of iterations */
int P;
float mu;
float tau;
int deltaX;
int element; /* number of elements */
int initVolt[MAXCHANNEL];
};

struct param PromptValues (void);
struct param DefaultParamValues (void);

float MeasurePower (int Dev,int ch);

int AdptiveAlgo (FILE *out_file,int Dev,int ch,HANDLE hdrv,
AOUT *ipaout,struct param param);
int AdptiveAlgo_ver1 (FILE *out_file,int Dev,int ch,HANDLE hdrv,
AOUT *ipaout,struct param param);
int AdptiveAlgo_ver2 (FILE *out_file,int Dev,int ch,HANDLE hdrv,
AOUT *ipaout,struct param param);
```

```
void DAsend (int iterNb,int nb,int *Xn);
void Print_Param_Values (FILE *fp,struct param p);
void DAOutput (HANDLE hdrv,AOUT *ipaout,int *values,int nb);
void Write_Voltage (FILE *out_file,int num,int nb,int *Xn,float corr0);

#endif
```

adapcom.c

program name : adapcom.c
path: source/antennaGG2/adapcom.c
comments : Genaral functions for the adaptive part

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "comhead.h"
#include "adaptive.h"
#include "dalink.h"
#include "gpib.h"

float MeasurePower(int Dev,int ch)
{
struct StructSMeasure pinfo;
float ret = 0.0;

#if (NA_VERSION==ATR_VER)
pinfo = Get_S_Measure(Dev,ch,"S21");
#else
pinfo = AAG_Get_S_Measure(Dev);
#endif

return (pinfo.magMean);
}

struct param DefaultParamValues (void)
{
struct param ret;

ret.deltaX = 40;
ret.element = 6;
ret.mu = 300;
ret.N = 10;
```

```
ret.P = 2;

ret.initVolt[0] = 1500;
ret.initVolt[1] = 1500;
ret.initVolt[2] = -1500;
ret.initVolt[3] = -1500;
ret.initVolt[4] = -1500;
ret.initVolt[5] = 1500;
ret.initVolt[6] = VOLT_MIN;
ret.initVolt[7] = VOLT_MIN;

return (ret);
}

struct param PromptValues (void)
{
struct param ret;
int k;
extern WORD Ch_Def[];

fflush(stdin);
ret.element = 6;
ret.P = 3;
// printf("Number of elements .... :");
// scanf("%d",&ret.element);
printf("Number of iterations .. :");
scanf("%d",&ret.N);
// printf("Length of P param ..... :");
// scanf("%d",&ret.P);
printf("Delta X ............... :");
scanf("%d",&ret.deltaX);
// printf("tau ................... :");
// scanf("%f",&ret.tau);
printf("Mu param .............. :");
scanf("%f",&ret.mu);

/* prompting init voltages */
for (k=0;k<ret.element;k++)
{
ret.initVolt[k] = VOLT_MAX+1;

while ( !(ret.initVolt[k]<=VOLT_MAX && ret.initVolt[k]>=VOLT_MIN) )
{
```

```
printf("Initial Voltage on Element%d (%d:%d)[ch:%d] <= ",k+1,VOLT_MIN,VOLT_MAX,Ch_Def[k]);
scanf("%d",&ret.initVolt[k]);
};
};

for (k=ret.element;k<MAXCHANNEL;k++)
ret.initVolt[k] = VOLT_MIN;

return (ret);
}

void Print_Param_Values (FILE *fp,struct param p)
{
if (fp != NULL)
{
fprintf(fp,"#M\t=%d;\n#N\t=%d;\n#P\t=%d;\n",p.element,p.N,p.P);
fprintf(fp,"#deltaX\t=%d;\n#mu\t=%4.2f\n#tau\t=%4.2f\n",p.deltaX,p.mu,p.tau);
fprintf(fp,"#VOLT_MIN:%d; VOLT_MAX:%d;\n\n",VOLT_MIN,VOLT_MAX);
}
}

void DAOutput(HANDLE hdrv,AOUT *ipaout,int *values,int nb)
{
int dwret,k;

for (k=0;k<nb;k++)
ipaout->OutBuf[k] = (WORD)(values[k]-VOLT_MIN);

dwret = AioOut(hdrv,ipaout);
if (dwret!=0)
printf("Error in: DAOutput [0x%xh]\n",dwret);
/* tempo for voltage affectation */
Idling(300);
}

void DAsend(int iterNb,int nb,int *Xn)
{
int i;

printf("[%d] ",iterNb);
for(i=0;i<nb;i++)
printf("%d, ",Xn[i]);
printf("\n");
```

```
}

void Write_Voltage (FILE *out_file,int num,int nb,int *Xn,float corr0)
{
int k;

fprintf(out_file,"%d ,",num);
fprintf(out_file,"%3.2f ,",20.0*log10(corr0));
fprintf(out_file,"%f ,",corr0);
for (k=0;k<nb-1;k++)
fprintf(out_file,"%d ,",Xn[k]);
fprintf(out_file,"%d \n",Xn[k]);
fflush(out_file);
}
```

adaptive1.c

program name : adaptive1.c
path: source/antennaGG2/adaptive1.c
comments : Adaptive algorithm functions

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "comhead.h"
#include "adaptive.h"
#include "dalink.h"
#include "gpib.h"


int AdptiveAlgo_ver1(FILE *out_file,int Dev,int ch,HANDLE hdrv,
AOUT *ipaout,struct param param)
{
float corr0 = 0.0;
float corr[NBVOLTAGE] = {0.0};
float DeltaRho[NBVOLTAGE] = {0.0};
float ValueTmp = 0.0;
float DivCoef = 1.0;

int Xn[NBVOLTAGE];
int Xtmp;
int sign;
int n,k,m;
int mem;

/* write header results in file */
fprintf(out_file,"#indice ,cost function (dB) ,cost function(linear) ,");

for (k=0;k<param.element-1;k++)
fprintf(out_file,"v%d ,",k);
fprintf(out_file,"v%d \n",k);
```

```
/* set init voltages */
for (k=0;k<param.element;k++)
Xn[k] = param.initVolt[k];

for (n=1;n<=param.N;n++)
{
DAOutput(hdrv,ipaout,Xn,param.element);
corr0 = MeasurePower(Dev,ch);

/* write file */
Write_Voltage(out_file,n,param.element,Xn,corr0);
/* print voltage values */
DAsend(n,param.element,Xn);

ValueTmp = 0.0;
for (m=0;m<param.element;m++)
{
Xtmp = param.deltaX + Xn[m];
sign = +1;
mem = Xn[m];

if (Xtmp >= VOLT_MAX)
{
Xn[m] = Xn[m] - param.deltaX;
sign = -1;
}
else
Xn[m] = Xtmp;

DAOutput(hdrv,ipaout,Xn,param.element);
corr[m] = MeasurePower(Dev,ch);
Xn[m] = mem;

if (sign > 0)
DeltaRho[m] = corr[m] - corr0;
else
DeltaRho[m] = -(corr[m] - corr0);

/* for normalization */
ValueTmp = ValueTmp + corr[m]*corr[m];
};
```

```
/* compute of new Xn */
DivCoef = (float)sqrt(ValueTmp/((float)param.element));
for (k=0;k<param.element;k++)
{
Xn[k] = Xn[k] + (int)(param.mu * (DeltaRho[k]/DivCoef));

if (Xn[k] > VOLT_MAX)
Xn[k] = VOLT_MAX;
else if (Xn[k] < VOLT_MIN)
Xn[k] = VOLT_MIN;
};

};

return (1);

}

int AdptiveAlgo(FILE *out_file,int Dev,int ch,HANDLE hdrv,AOUT *ipaout,struct param param)
{
float corr[NBVOLTAGE+1] = {0.0};
float DeltaRho[NBVOLTAGE+1] = {0.0};
int sign[NBVOLTAGE] = {1};

int Xn[NBVOLTAGE],
Xtmp;

float ValueTmp = 0.0;
float DivCoef = 0.0;

int n,m,k;
int mem;

n = 1;
m = 0;

/* write header results in file */
fprintf(out_file,"#indice ,cost function (dB) ,cost function(linear) ,");

for (k=0;k<param.element-1;k++)
fprintf(out_file,"v%d ,",k);
fprintf(out_file,"v%d \n",k);
```

```
/* set init voltages */
for (k=0;k<param.element;k++)
Xn[k] = param.initVolt[k];
/* output initial voltages */
DAOutput(hdrv,ipaout,Xn,param.element);

while (n <= param.N)
{
if(n <= param.N)
{
if(m < param.element)
{
if(m != 0)
{
Xn[m-1] = mem;
};

corr[m] = MeasurePower(Dev,ch);

mem = Xn[m];
Xtmp = param.deltaX + Xn[m];

if (Xtmp >= VOLT_MAX)
{
Xtmp = Xn[m] - param.deltaX;
if (m != 0)
sign[m-1] = -1;
}
else if (m != 0)
sign[m-1] = 1;

Xn[m] = Xtmp;

DAOutput(hdrv,ipaout,Xn,param.element);

m++;
}
else
{
Xn[m-1] = mem;
corr[m] = MeasurePower(Dev,ch);

/* normalization */
```

```
ValueTmp = 0.0;
for (k = 1;k <= param.element; k++)
{
if (sign[k-1] > 0)
DeltaRho[k] = corr[k] - corr[0];
else
DeltaRho[k] = -(corr[k] - corr[0]);

ValueTmp = ValueTmp + corr[k]*corr[k];
};

DivCoef = (float)sqrt(ValueTmp/((float)param.element));

for (k = 1;k <= param.element; k++)
{
Xn[k-1] = Xn[k-1] + (int)(param.mu * (DeltaRho[k]/DivCoef));
};

for (k = 0; k < param.element; k++)
{
if (Xn[k] > VOLT_MAX)
Xn[k] = VOLT_MAX;
else if (Xn[k] < VOLT_MIN)
Xn[k] = VOLT_MIN;
};

DAOutput(hdrv,ipaout,Xn,param.element);

/* write file */
Write_Voltage(out_file,n,param.element,Xn,corr[0]);
// DAsend(n,param.element,Xn);
Write_Voltage(stdout,n,param.element,Xn,corr[0]);

n++;
m = 0;
};

};
};

return (1);
}
```

adaptive2.c

program name : adaptive2.c
path: source/antennaGG2/adaptive2.c
comments : Adaptive algorithm functions

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "comhead.h"
#include "adaptive.h"
#include "dalink.h"
#include "gpib.h"


int AdptiveAlgo_ver2(FILE *out_file,int Dev,int ch,HANDLE hdrv,
AOUT *ipaout,struct param param)
{
float corr0 = 0.0;
float corr[NBVOLTAGE] = {0.0};
float DeltaRho[NBVOLTAGE] = {0.0};
float ValueTmp = 0.0;
float DivCoef = 1.0;
float mu;

int Xn[NBVOLTAGE];
int XT[NBVOLTAGE];

int n,k,m;
int mm;

/* write header results in file */
fprintf(out_file,"indice ,cost function (dB) ,cost function(linear) ,");

for (k=0;k<param.element-1;k++)
fprintf(out_file,"v%d ,",k);
```

```
fprintf(out_file,"v%d \n",k);

/* set init voltages */
for (k=0;k<param.element;k++)
Xn[k] = param.initVolt[k];

DAOutput(hdrv,ipaout,Xn,param.element);  /* only for the first iteration */
Idling(600);  /* waiting at the first iteration */

for (n=1;n<=param.N;n++)
{

DAOutput(hdrv,ipaout,Xn,param.element);
corr0 = MeasurePower(Dev,ch);

/* write file */
Write_Voltage(out_file,n,param.element,Xn,corr0);
/* print voltage values */
// DAsend(n,param.element,Xn);
Write_Voltage(stdout,n,param.element,Xn,corr0);


// ValueTmp = 0.0;
for (m=0;m<param.element;m++)
{

for (mm=0; mm<param.element;mm++)
XT[mm]=Xn[mm];

if (param.deltaX + XT[m] < VOLT_MAX) {
XT[m] = param.deltaX + XT[m];
DAOutput(hdrv,ipaout,XT,param.element);
corr[m] = MeasurePower(Dev,ch);
DeltaRho[m] = corr[m] - corr0;
} else  {
XT[m] = XT[m] - param.deltaX;
DAOutput(hdrv,ipaout,XT,param.element);
corr[m] = MeasurePower(Dev,ch);
DeltaRho[m] = -(corr[m] - corr0);
}


/* for normalization */
```

```
// ValueTmp = ValueTmp + corr[m]*corr[m];
};

/* compute of new Xn */
// DivCoef = (float)sqrt(ValueTmp/((float)param.element));



// mu= param.mu / ((float)1.0 + ((float)n)/param.tau);
// printf("%d -- %f\n",n,mu);
mu= param.mu;
for (k=0;k<param.element;k++)
{
// Xn[k] = Xn[k] + (int)(mu * (DeltaRho[k]/DivCoef));
Xn[k] = Xn[k] + (int)(mu * (DeltaRho[k]/corr0));

if (Xn[k] > VOLT_MAX)
Xn[k] = VOLT_MAX;
else if (Xn[k] < VOLT_MIN)
Xn[k] = VOLT_MIN;
};

};

return (1);

}
```

adaptive2.c

program name : adaptive2.c
path: source/antennaGG2/adaptive2.c
comments : Adaptive algorithm functions

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "comhead.h"
#include "adaptive.h"
#include "dalink.h"
#include "gpib.h"


int AdptiveAlgo_ver2(FILE *out_file,int Dev,int ch,HANDLE hdrv,
AOUT *ipaout,struct param param)
{
float corr0 = 0.0;
float corr[NBVOLTAGE] = {0.0};
float DeltaRho[NBVOLTAGE] = {0.0};
float ValueTmp = 0.0;
float DivCoef = 1.0;
float mu;

int Xn[NBVOLTAGE];
int XT[NBVOLTAGE];

int n,k,m;
int mm;

/* write header results in file */
fprintf(out_file,"indice ,cost function (dB) ,cost function(linear) ,");

for (k=0;k<param.element-1;k++)
fprintf(out_file,"v%d ,",k);
```

```
fprintf(out_file,"v%d \n",k);

/* set init voltages */
for (k=0;k<param.element;k++)
Xn[k] = param.initVolt[k];

DAOutput(hdrv,ipaout,Xn,param.element);  /* only for the first iteration */
Idling(600);  /* waiting at the first iteration */

for (n=1;n<=param.N;n++)
{

DAOutput(hdrv,ipaout,Xn,param.element);
corr0 = MeasurePower(Dev,ch);

/* write file */
Write_Voltage(out_file,n,param.element,Xn,corr0);
/* print voltage values */
// DAsend(n,param.element,Xn);
Write_Voltage(stdout,n,param.element,Xn,corr0);


// ValueTmp = 0.0;
for (m=0;m<param.element;m++)
{

for (mm=0; mm<param.element;mm++)
XT[mm]=Xn[mm];

if (param.deltaX + XT[m] < VOLT_MAX) {
XT[m] = param.deltaX + XT[m];
DAOutput(hdrv,ipaout,XT,param.element);
corr[m] = MeasurePower(Dev,ch);
DeltaRho[m] = corr[m] - corr0;
} else  {
XT[m] = XT[m] - param.deltaX;
DAOutput(hdrv,ipaout,XT,param.element);
corr[m] = MeasurePower(Dev,ch);
DeltaRho[m] = -(corr[m] - corr0);
}


/* for normalization */
```

```
// ValueTmp = ValueTmp + corr[m]*corr[m];
};

/* compute of new Xn */
// DivCoef = (float)sqrt(ValueTmp/((float)param.element));



// mu= param.mu / ((float)1.0 + ((float)n)/param.tau);
// printf("%d -- %f\n",n,mu);
mu= param.mu;
for (k=0;k<param.element;k++)
{
// Xn[k] = Xn[k] + (int)(mu * (DeltaRho[k]/DivCoef));
Xn[k] = Xn[k] + (int)(mu * (DeltaRho[k]/corr0));

if (Xn[k] > VOLT_MAX)
Xn[k] = VOLT_MAX;
else if (Xn[k] < VOLT_MIN)
Xn[k] = VOLT_MIN;
};

};

return (1);

}
```

dalink.h

program name : dalink.h
path: source/antennaGG2/dalink.h
comments : D/A board header

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include "apiaio.h"


#ifndef DALINK_H
#define DALINK_H

/* ------------------------------------------------------------ */

/* defines & declarations for the D/A board */
#define DRVNO 4 /* 4  -> DA12 8L(PC); 26 -> DA12-16(PCI) */
#define GRPNO 0

#define MAXCHANNEL 8 /* depend of the D/A board model */

#define NBVOLTAGE 6 /* corresponding to the number of used CHANNELs (max=MAXCHANNEL) */
#define NBBITS 12 /* nombre de bits de codage (ex: 12bits => 0 - 4095) */
#define GAINMAX 1 /* maximun gain (see board specification) */

#define VOLT_MIN -(1<<(NBBITS-1))
#define VOLT_MAX (1<<(NBBITS-1))-1

/* ------------------------------------------------------------ */

WORD *ConvertVolt2Binary (int nbVal,float *volt,WORD *binary);
int Simple_Output_Test (HANDLE hdrv);

#endif
```

dalink.c

program name : dalink.c
path: source/antennaGG2/dalink.c
comments : D/A board functions

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>

#include "comhead.h"
#include "apiaio.h"
#include "dalink.h"

/* ------------------------------------------------------------ */
/* which voltage matching with the output channel */
#if (NA_VERSION==ATR_VER)
WORD Ch_Def[NBVOLTAGE] = {4, 3, 2, 1, 0, 5};  // ATR cable
#else
WORD Ch_Def[NBVOLTAGE] = {0, 1, 2, 3, 4, 5};  // Antenna Gigen cable
#endif

/* output range
0 : -10V <-> +10V
1 : -5V <-> +5V
2 : 0V <-> +10V */
WORD Output_Range[NBVOLTAGE] = {2, 2, 2, 2, 2, 2};
/* ------------------------------------------------------------ */


WORD *ConvertVolt2Binary(int nbVal,float *volt,WORD *binary)
{
float RangeMin,RangeMax;
int i;
WORD *ret;

if (volt!=NULL && binary!=NULL)
{
```

```
for(i=0;i<nbVal && i<NBVOLTAGE;i++)
{
switch (Output_Range[i])
{
case (0) : RangeMin = -10.0; RangeMax = 10.0; break;
case (1) : RangeMin = -5.0; RangeMax = 5.0; break;
case (2) : RangeMin = 0.0; RangeMax = 10.0; break;
};
binary[i] = VIToBinary(RangeMin,RangeMax,GAINMAX,NBBITS,volt[i]);
};
ret = binary;
};

return (ret);

}

int Simple_Output_Test (HANDLE hdrv)
{
WORD  outbuf[MAXCHANNEL] = {0};
WORD outchno[MAXCHANNEL] = {0, 1, 2, 3, 4, 5, 6, 7};
float volt[MAXCHANNEL] = {0.0};
AOUT ipaout;
int dwret,k,
outr = 2,
    mode = 0;
float RangeMin,RangeMax;

    ipaout.Channels = 1;
ipaout.OutBuf = outbuf;
ipaout.OutChNo = outchno;

printf(" ---- Simple D/A output voltage on channel(s) ----\n");

printf("How many channels do you want ........ :");
scanf("%d",&ipaout.Channels);
if (ipaout.Channels>MAXCHANNEL)
ipaout.Channels = MAXCHANNEL;

printf("Range for D/A board output\n");
printf("(0) -10V to 10V\n");
printf("(1)  -5V to  5V\n");
printf("(2)   0V to 10V\n");
```

```
printf("Output range for these channels (0-2). :");
scanf("%d",&outr);
if (0<outr || outr>2)
outr = 2;
switch (outr)
{
case (0) : RangeMin = -10.0; RangeMax = 10.0; break;
case (1) : RangeMin = -5.0; RangeMax = 5.0; break;
case (2) : RangeMin = 0.0; RangeMax = 10.0; break;
};

/* set the output range */
for (k=0;k<ipaout.Channels;k++)
outbuf[k] = outr;
AioRange(hdrv,outchno,outbuf,ipaout.Channels);

printf("Mode of input (0->volt; 1->integer) .. :");
scanf("%d",&mode);
if (mode!=0 && mode!=1)
mode = 0;

for (k=0;k<ipaout.Channels;k++)
{
if (mode==1)
{
int val;

val = VOLT_MAX+1;
while ( !(val<=VOLT_MAX && val>=VOLT_MIN) )
{
printf("Initial Voltage(%d:%d)[ch:%d] <= ",VOLT_MIN,VOLT_MAX,outchno[k]);
scanf("%d",&val);
};
outbuf[k] = (WORD)(val - VOLT_MIN);
}
else
{   volt[k] = RangeMax+(float)1.0;
while ( !(volt[k]<=RangeMax && volt[k]>=RangeMin) )
{
printf("Initial Voltage(%3.1fV;%3.1fV)[ch:%d] <= ",RangeMin,RangeMax,outchno[k]);
scanf("%f",&volt[k]);
};
```

```
}
};

if (mode==0)
ConvertVolt2Binary(ipaout.Channels,volt,outbuf);

dwret = AioOut(hdrv,&ipaout);
if (dwret!=0)
printf("Error in: AioOut(hdrv,&ipaout) [0x%xh]\n",dwret);
else
printf("Output succeded ...\n");
pause;

return (dwret);
}
```

gpib.h

program name : gpib.h
path: source/antennaGG2/gpib.h
comments : GPIB functions header

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#ifndef GPIB_H
#define GPIB_H

#include <windows.h>
#include "decl-32.h"


/* ------------------------------------------------------------------------------ */
/* declararions */
#define MAX_MEAS 1024

/*  decarations GPIB */
#define ARRAYSIZE 32768     // Size of read buffer
#define TMPM ARRAYSIZE

#define BDINDEX 0     // Board Index
#define PRIMARY_ADDR_OF_DMM 16     // Primary address of device
#define NO_SECONDARY_ADDR 0     // Secondary address of device
#define TIMEOUT T10s  // Timeout value = 10 seconds
#define EOTMODE 1     // Enable the END message
#define EOSMODE 0     // Disable the EOS mode

/* ------------------------------------------------------------------------------ */
/* main strcuture */
struct StructSMeasure {
int nbMea;
float mag[MAX_MEAS];
float phase[MAX_MEAS];
float magMean;
float phaseMean;
};
```

```
/* ------------------------------------------------------------------------------ *
/* init settings for the GPIB-PNA */

#define GPIB_WIN_CHANNEL 1
#define GPIB_WIN 1
#define GPIB_TRACE 1
#define GPIB_FREQ 2.484 /* freq in GHz */
// #define GPIB_POWL -10 /* power level in dBm (-90; +20) */
#define GPIB_POWL 0 /* power level in dBm (-90; +20) */
#define GPIB_S_MEAS "S21"

/* only for ATR version */
#define MEAS_PHASE 0 /* 1:measure the phase, 0:doesnot measure */

/* only for AGG version */
#define AGG_NA_POINT 3

/* ------------------------------------------------------------------------------ *
/* GPIB commun functions  */

int NA_gpib_init ( void );
int gpib_close (int Dev);
int gpib_init_prompt (void);

int GPIBCrd (int Dev,char *str,long size);
int GPIBCwrt (int Dev,char *str);
void GPIBCleanup (int Dev, char* ErrorMsg);

/* ------------------------------------------------------------------------------ *
/* ATR Network analyser (Agilent E8358A) */

int Create_S_Measure (int Dev,int channel,char *name_meas);
int Delete_S_Measure (int Dev,int channel,char *name_meas);

int NA_GPIBparam_init (int Dev,int channel,int num_win,long start_freq,
long stop_freq,long power_level,int nb_pt);
int NA_GPIBfreq_init (int Dev,int channel,int num_win,int num_trace,
int nb_pt,float freq,int power_level,char *name_meas);

struct StructSMeasure Get_S_Measure (int Dev,int channel,char *name_meas);

/* ------------------------------------------------------------------------------ *
```

```
/* Antenna GIGEN Network analyser (???) */

int AGG_NA_GPIBfreq_init (int Dev,int channel,int power_level,int nb_pt,int bandwith,
int averf,float freq,int ref,char *name_meas);
struct StructSMeasure AAG_Get_S_Measure (int Dev);

#endif
```

gpib.c

program name : gpib.c
path: source/antennaGG2/gpib.c
comments : GPIB commom functions

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>

#include "comhead.h"

/*
 *  Include the WINDOWS.H and DECL-32.H files. The standard Windows
 *  header file, WINDOWS.H, contains definitions used by DECL-32.H and
 *  DECL-32.H contains prototypes for the GPIB routines and constants.
 */
#include <windows.h>
#include "decl-32.h"
#include "gpib.h"

char ErrorMnemonic[21][5] = {"EDVR", "ECIC", "ENOL", "EADR", "EARG",
                             "ESAC", "EABO", "ENEB", "EDMA", "",
                             "EOIP", "ECAP", "EFSO", "", "EBUS",
                             "ESTB", "ESRQ", "", "", "", "ETAB"};

FILE *_debug_file;

/* ------------------------------------------------------------- */
int gpib_init_prompt (void)
{
int  Dev;
int bdindex,padmm,nsadr;

printf("---------- GPIB open settings -----------------\n");
printf("Board Index:");
```

```
scanf("%d",&bdindex);
printf("Primary address of device:");
scanf("%d",&padmm);
printf("Secondary address of device:");
scanf("%d",&nsadr);
printf("-----------------------------------------------\n");
    Dev = ibdev(bdindex, padmm, nsadr,
                TIMEOUT, EOTMODE, EOSMODE);
    if (ibsta & ERR)
    {
      printf("Unable to open device\nibsta = 0x%x iberr = %d\n",
              ibsta, iberr);
      exit(1);
    }

    ibclr (Dev);
    if (ibsta & ERR)
    {
       GPIBCleanup(Dev, "Unable to clear device");
       exit(1);
    }

return (Dev);
}

/* ------------------------------------------------------------- */
int NA_gpib_init( void )
{
int  Dev;

    Dev = ibdev(BDINDEX, PRIMARY_ADDR_OF_DMM, NO_SECONDARY_ADDR,
                TIMEOUT, EOTMODE, EOSMODE);
    if (ibsta & ERR)
    {
      printf("Unable to open device\nibsta = 0x%x iberr = %d\n",
              ibsta, iberr);
      exit(1);
    }

    ibclr (Dev);
    if (ibsta & ERR)
    {
       GPIBCleanup(Dev, "Unable to clear device");
```

```
        exit(1);
    }

return (Dev);
}

/* ------------------------------------------------------------ */
int gpib_close (int Dev)
{

/* Take device local
Idling(800);
ibloc(Dev);
if (ibsta & ERR)
GPIBCleanup(Dev,"in gpib_close can not do ibloc\n");
*/
    ibonl(Dev, 0);

    return 0;
}

/* ------------------------------------------------------------ */
void GPIBCleanup(int Dev, char* ErrorMsg)
{
    printf("Error : %s\nibsta = 0x%x iberr = %d (%s)\n",
            ErrorMsg, ibsta, iberr, ErrorMnemonic[iberr]);
    if (Dev != -1)
    {
       printf("Cleanup: Taking device off-line\n");
       ibonl (Dev, 0);
    }
}

/* ------------------------------------------------------------ */
int GPIBCwrt(int Dev,char *str)
{
  long sze = (long)strlen(str);
  int ret;

  ret = ibwrt(Dev,str,sze);
  if (ibsta & ERR)
  {
  GPIBCleanup(Dev,str);
```

```
        exit(1);
    };

    DB(fprintf(DEBUG_OUT,"debug GPIB write %s [=>%d]\n",str,ret);
    fflush(DEBUG_OUT);
    DB_pause;);

    return (ret);
}

/* ------------------------------------------------------------- */
int GPIBCrd(int Dev,char *str,long size)
{
    int ret;
    char *ptr = str;

    ret = ibrd(Dev,str,size);
    if (ibsta & ERR)
    {
    GPIBCleanup(Dev,str);
    exit(1);
    }

    if (ptr!=NULL)
    while(*ptr!='\n') ptr++;
     *(ptr+1) = '\0';

    DB(fprintf(DEBUG_OUT,"debug GPIB read %s [=>%d]\n",str,ret);
    fflush(DEBUG_OUT);
    DB_pause;
    );

    return (ret);
}
```

gpib_agg.c

program name : gpib_agg.c
path: source/antennaGG2/gpib_agg.c
comments : GPIB functions for Antenna Gigen Network Analazer

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>

#include "comhead.h"
#include <windows.h>
#include "decl-32.h"
#include "gpib.h"

/*

  This version is for Antenna GIGEN network analyser

*/


/* ---------------------------------------------------------------
Get the S magnitude measure
   ---------------------------------------------------------------- */

struct StructSMeasure AAG_Get_S_Measure(int Dev)
{
char tmp[51] = {'\0'};
struct StructSMeasure ret;
int i;

/*Set Linear Magnitude */
// GPIBCwrt(Dev,"LINM;");
// GPIBCwrt(Dev,"OPC?;WAIT;");
```

```
/* Set ASCII format for data measurement */
// GPIBCwrt(Dev,"FORM4;");

/* Read data */
GPIBCwrt(Dev,"OUTPFORM;");
Idling(300);

/* Get the Log Mag */
ret.magMean = (float)0.0;
for (i=0;i<AGG_NA_POINT;i++)
{
GPIBCrd(Dev,tmp,50);
tmp[24] = '\0';
sscanf(tmp,"%f",&ret.mag[i]);
ret.magMean += ret.mag[i];
Idling(30);
};
ret.nbMea = AGG_NA_POINT;
ret.magMean = ret.magMean/AGG_NA_POINT;

return (ret);
}

/* ----------------------------------------------------------------
Network Analyser settings
   ---------------------------------------------------------------- */

int AGG_NA_GPIBfreq_init(int Dev,int channel,int power_level,int nb_pt,
int bandwith,int averf,float freq,int ref,char *name_meas)
{
char tmp[TMPM];

/* Reset the Network Analyser */
GPIBCwrt(Dev,"RST;");
// GPIBCwrt(Dev,"OPC?;PRES;");

/* Authorize beep warning */
GPIBCwrt(Dev,"BEEPWARNON;");

/*  Set channel to show measure */
sprintf(tmp, "CHAN%d",channel);
GPIBCwrt(Dev,tmp);
```

```
/* define S measure */
sprintf(tmp, "%s;",name_meas);
GPIBCwrt(Dev,tmp);

/* Set Log Mag format to set ref line value */
GPIBCwrt(Dev,"LOGM;");
GPIBCwrt(Dev,"OPC?;WAIT;");
Idling(300);

/* reference line value (LOGM +-500dB,LINM +-500) */
sprintf(tmp,"REFV%ddB;",ref);
GPIBCwrt(Dev,tmp);

/* select number of point  */
sprintf(tmp, "POIN%d;",nb_pt);
GPIBCwrt(Dev,tmp);

/* IF bandwidth */
sprintf(tmp, "IFBW%dHz;",bandwith);
GPIBCwrt(Dev,tmp);

/* Set frequency */
sprintf(tmp, "CWFREQ%2.6fGHz;",freq);
GPIBCwrt(Dev,tmp);
    Idling(30);

/* Set Power Level */
sprintf(tmp, "POWE%d;",power_level);
GPIBCwrt(Dev,tmp);
    Idling(30);

/* ------------- Add  */
/*Set Linear Magnitude */
GPIBCwrt(Dev,"LINM;");
GPIBCwrt(Dev,"OPC?;WAIT;");

/* Set ASCII format for data measurement */
GPIBCwrt(Dev,"FORM4;");
/* ------------- */

/* averaging settings */
GPIBCwrt(Dev,"AVEROFF;");
Idling(30);
```

```
// sprintf(tmp, "AVERFACT%d;",averf);
// GPIBCwrt(Dev,tmp);

/* Wait until network finished all settings */
GPIBCwrt(Dev,"OPC?;WAIT;");

return (1);
}
```

gpib_atr.c

program name : gpib_atr.c
path: source/antennaGG2/gpib_atr.c
comments : GPIB functions for ATR Network Analazer

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <conio.h>

#include "comhead.h"
#include <windows.h>
#include "decl-32.h"
#include "gpib.h"


/*

  This version is for ATR network analyser

*/

/* ------------------------------------------------------------- */
int Create_S_Measure(int Dev,int channel,char *name_meas)
{
char tmp[TMPM] = {'\0'};
   int ret = -1;

if (name_meas != NULL)
{
sprintf(tmp,"CALCULATE%d:PARAMETER:DEFINE 'Perso_%s',%s",channel,name_meas,name_meas);
ret = GPIBCwrt(Dev,tmp);
Idling(500);
};

return (ret);
```

```
}
/* ------------------------------------------------------------ */

int Delete_S_Measure(int Dev,int channel,char *name_meas)
{
char tmp[TMPM] = {'\0'};
   int ret = -1;

if (name_meas != NULL)
{
sprintf(tmp,"CALCULATE%d:PARAMETER:DELETE 'Perso_%s',%s",channel,name_meas,name_meas);
ret = GPIBCwrt(Dev,tmp);
Idling(500);
};

return (ret);

}

/* ------------------------------------------------------------ */
struct StructSMeasure Get_S_Measure(int Dev,int channel,char *name_meas)
{
char tmp[TMPM] = {'\0'};
struct StructSMeasure ret;
float *ftmp;
int k;

/* Set the Log Mag
sprintf(tmp,"CALCULATE%d:FORMAT MLOGARITHMIC",channel);*/
/*if MLINEAR */
sprintf(tmp,"CALCULATE%d:FORMAT MLINEAR",channel);
GPIBCwrt(Dev,tmp);

/* select S measurement */
sprintf(tmp,"CALCULATE%d:PARAMETER:SELECT 'Perso_%s'",channel,name_meas);
GPIBCwrt(Dev,tmp);

/* Get the Log Mag */
sprintf(tmp,"CALCULATE%d:DATA? FDATA",channel);
    GPIBCwrt(Dev,tmp);
Idling(300);
GPIBCrd(Dev,tmp,TMPM-1);
ftmp = String2FloatTable(tmp);
```

```
ret.nbMea = (int)ftmp[0];
recopy_f2f(ret.nbMea,&ftmp[1],ret.mag);

ret.magMean = (float)0.0;
for (k=0;k<ret.nbMea;k++)
ret.magMean += ret.mag[k];

ret.magMean = ret.magMean/ret.nbMea;

/* NOTE: we can avoid this part (calcul of phase) if it is note necessary */

#if MEAS_PHASE
/* Set the phase */
sprintf(tmp,"CALCULATE%d:FORMAT PHASE",channel);
GPIBCwrt(Dev,tmp);

/* select S measurement */
sprintf(tmp,"CALCULATE%d:PARAMETER:SELECT 'Perso_%s'",channel,name_meas);
GPIBCwrt(Dev,tmp);

/* Get the Phase */
sprintf(tmp,"CALCULATE%d:DATA? FDATA",channel);
    GPIBCwrt(Dev,tmp);
Idling(300);
GPIBCrd(Dev,tmp,TMPM-1);
ftmp = String2FloatTable(tmp);
ret.nbMea = (int)ftmp[0];
recopy_f2f(ret.nbMea,&ftmp[1],ret.phase);
#endif

return (ret);
}
/* ------------------------------------------------------------ */
/* !!!!!!   IMPORTANT  freq in GHz   !!!!!!!*/
int NA_GPIBfreq_init(int Dev,int channel,int num_win,int num_trace,int nb_pt,
float freq,int power_level,char *name_meas)
{
char tmp[TMPM];
int retv;

/* f...preset */
GPIBCwrt(Dev,"SYSTEM:FPRESET");
```

```
/* window settings */
/* check if the window already exist */
sprintf(tmp, "DISPLAY:WINDOW%d:STATE?",num_win);
GPIBCwrt(Dev,tmp);
GPIBCrd(Dev,tmp,TMPM-1);
sscanf(tmp,"%d",&retv);
if (retv != 1)
{
sprintf(tmp, "DISPLAY:WINDOW%d:STATE ON",num_win);
GPIBCwrt(Dev,tmp);
};

/* define S measure */
Create_S_Measure(Dev,channel,name_meas);
sprintf(tmp, "DISPLAY:WINDOW%d:TRACE%d:FEED 'Perso_%s'",num_win,num_trace,name_meas);
GPIBCwrt(Dev,tmp);

/* Set ASCII format for data measurement */
GPIBCwrt(Dev,"FORMAT:DATA ASCII");

/* initialize trigger */
/* automatic sweep */
sprintf(tmp,"INITIATE%d:CONTINUOUS ON; *OPC?",channel);
GPIBCwrt(Dev,tmp);
GPIBCrd(Dev,tmp,TMPM-1);

/* All measurements in the channel made per trigger */
    if (nb_pt > 1)
{
sprintf(tmp, "SENSE%d:SWEEP:TRIGGER:POINT OFF",channel);
GPIBCwrt(Dev,tmp);
sprintf(tmp, "SENSE%d:SWEEP:POINTS %d",channel,nb_pt);
GPIBCwrt(Dev,tmp);
}
else
sprintf(tmp,"SENSE%d:SWEEP:TRIGGER:POINT ON",channel);

/* select continue wave and frequency value */
sprintf(tmp, "SENSE%d:SWEEP:TYPE CW",channel);
GPIBCwrt(Dev,tmp);

sprintf(tmp, "SENSE%d:FREQUENCY:CW  %2.6f GHz",channel,freq);
GPIBCwrt(Dev,tmp);
```

```
/* power level */
sprintf(tmp, "SOURCE%d:POWER1 %d",channel,power_level);
GPIBCwrt(Dev,tmp);

return (1);
}

/* ------------------------------------------------------------ */
/* this function is not used */
int NA_GPIBparam_init(int Dev,int channel,int num_win,long start_freq,
long stop_freq,long power_level,int nb_pt)
{
char tmp[TMPM];
int retv;

/* preset */
GPIBCwrt(Dev,"SYSTem:FPRESET");

/* window settings */
/* check if the window already exist */
sprintf(tmp, "DISPLAY:WINDOW%d:STATE?",num_win);
GPIBCwrt(Dev,tmp);
GPIBCrd(Dev,tmp,TMPM-1);
sscanf(tmp,"%d",&retv);
if (retv != 1)
{
sprintf(tmp, "DISPLAY:WINDOW%d:STATE ON",num_win);
GPIBCwrt(Dev,tmp);
};

GPIBCwrt(Dev,"TRIGger:SEQuence:SCOPe ALL");  //One trigger will trigger all channels

/* GPIBCwrt(Dev,"DISPlay:WINDow1:TRACe1:FEED 'My_S11'"); */

sprintf(tmp,"INITiate%d:CONTinuous OFF; *OPC?",channel);
GPIBCwrt(Dev,tmp);           //set MANUAL SWEEP and wait for completion
GPIBCrd(Dev,tmp,TMPM-1);     //when read, then completion

sprintf(tmp, "SENSe%d:SWEep:TRIGger:POINt OFF",channel);
GPIBCwrt(Dev,tmp);            //set all points to be read per trigger event
sprintf(tmp, "SENSe%d:SWEep:POINts %d",channel,nb_pt);
GPIBCwrt(Dev,tmp);
```

```
sprintf(tmp, "SENSe%d:FREQuency:STARt %g",channel,start_freq);
GPIBCwrt(Dev,tmp);

sprintf(tmp, "SENSe%d:FREQuency:STOP %g",channel,stop_freq);
GPIBCwrt(Dev,tmp);

sprintf(tmp, "SOURce%d:POWer %g",channel,power_level);
GPIBCwrt(Dev,tmp);

return (1);
}
```

freqexpt.c

program name : freqexpt.c
path: source/antennaGG2/freqexpt.c
comments : Function(s) for inspecting the effect of frequency

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "Apiaio.h"
#include "comhead.h"
#include "adaptive.h"
#include "dalink.h"
#include "gpib.h"
#include "mfunc.h"

int Frequence_Power_Measure (int Dev,int ch,HANDLE hdrv,FILE *fp,float freq_center)
{
WORD  outbuf[NBVOLTAGE] = {0};
AOUT ipaout;

int k,dwret;
float lpow,rangeMin,rangeMax,step,ind;

extern WORD Ch_Def[];

    ipaout.Channels = NBVOLTAGE;
ipaout.OutBuf = outbuf;
ipaout.OutChNo = Ch_Def;

fflush(stdin);
printf("Frequence range minimun (MHz) . :");
scanf("%f",&rangeMin);
printf("Frequence range maximun (MHz) . :");
scanf("%f",&rangeMax);
```

```
printf("Step (MHz) ..................... :");
scanf("%f",&step);

printf("---- Set the voltages ----\n");
/* Set the initial voltages */
for (k=0;k<ipaout.Channels;k++)
{
int val;

val = VOLT_MAX+1;
while ( !(val<=VOLT_MAX && val>=VOLT_MIN) )
{
printf("Initial Voltage on Element%d (%d:%d)[ch:%d] <= ",k+1,VOLT_MIN,VOLT_MAX,Ch_Def[k]);
scanf("%d",&val);

};
outbuf[k] = (WORD)(val - VOLT_MIN);
fprintf(fp,"#Voltage element%d =\t%d;\n",k+1,val);
};

fprintf(fp,"#central frequence : %2.6f;\n",freq_center);
fprintf(fp,"#frequence range   : %5.6f,%5.6f;\n",rangeMin,rangeMax);
fprintf(fp,"#frequence step    : %5.6f;\n",step);
fprintf(fp,"\n#Freq offset (MHz)\tpower (dB)\tpower (linear)\n");

/* output the voltages */
dwret = AioOut(hdrv,&ipaout);
if (dwret!=0)
printf("Error in: DAOutput [0x%xh]\n",dwret);
Idling(500);

/* Measure the power */
for (ind=rangeMin;ind<=rangeMax;ind+=step)
{
Set_frequence (Dev,ch,freq_center,ind);
lpow = MeasurePower(Dev,ch);

fprintf(fp,"%5.6f\t%f\t%2.6f\n",ind,20.0*log10(lpow),lpow);
printf("%5.6f\t%f\t%2.6f\n",ind,20.0*log10(lpow),lpow);

}

return (1);
```

```
}



/*
IMPORTANT !!!!!!!!!!!!!!!!!!!!!!
IN THIS FUNCTION
FREQ_CENTER IS IN  **** GHz *****
FREQ_OFFSET IS IN  **** MHz *****
*/

int Set_frequence (int Dev,int ch,float freq_center,float freq_offset)
{

float freq;
char tmp[TMPM];
int ret;

freq = freq_center + (freq_offset/(float)1000.0);
if (freq<0)
{
printf("Mega giga fatal Error : negative frequence value %f\n",freq);
exit(1);
}

/* Set frequency */
#if (NA_VERSION==ATR_VER)
sprintf(tmp, "SENSE%d:FREQUENCY:CW  %2.6f GHz",ch,freq);
#else
sprintf(tmp, "CWFREQ%2.6fGHz;",freq);
#endif

ret = GPIBCwrt(Dev,tmp);
    Idling(500);

return (ret);
}
```

voltexpt.c

program name : voltexpt.c
path: source/antennaGG2/voltexpt.c
comments : Function(s) for inspecting the effect of voltage(s)

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

/*
    Last modification:
    7 September 2001, add of option --omni
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "Apiaio.h"
#include "comhead.h"
#include "adaptive.h"
#include "dalink.h"
#include "gpib.h"
#include "mfunc.h"

int Voltage_Power_Measure (int Dev,int ch,HANDLE hdrv,FILE *fp)
{
WORD  outbuf[NBVOLTAGE] = {0};
AOUT ipaout;

int eltNotSet,step,rangeMin,rangeMax,k,eltVolt;
float lpow;

extern WORD Ch_Def[];

ipaout.Channels = NBVOLTAGE;
ipaout.OutBuf = outbuf;
ipaout.OutChNo = Ch_Def;

fflush(stdin);
```

```
eltNotSet = 7;
while (eltNotSet>6 || eltNotSet<1)
{
printf("Element choose (1:%d) .. :",ipaout.Channels);
scanf("%d",&eltNotSet);
}

step = 0;
while (step<1 || step>1000)
{
printf("Step (1:1000) ......... :");
scanf("%d",&step);
}

rangeMin = VOLT_MIN-1;
rangeMax = VOLT_MAX+1;
while (rangeMin>rangeMax || rangeMin<VOLT_MIN || rangeMax>VOLT_MAX)
{
printf("Range minimun (%d:%d) . :",VOLT_MIN,VOLT_MAX);
scanf("%d",&rangeMin);
printf("Range maximun (%d:%d) . :",VOLT_MIN,VOLT_MAX);
scanf("%d",&rangeMax);
}

/* Set the initial voltages */
for (k=0;k<ipaout.Channels;k++)
{
int val;

if (k!=(eltNotSet-1))
{
val = VOLT_MAX+1;
while ( !(val<=VOLT_MAX && val>=VOLT_MIN) )
{
printf("Initial Voltage on Element%d (%d:%d)[ch:%d] <= ",
k+1,VOLT_MIN,VOLT_MAX,Ch_Def[k]);
scanf("%d",&val);

};
outbuf[k] = (WORD)(val - VOLT_MIN);
fprintf(fp,"#Voltage element%d =\t%d;\n",k+1,val);
}
```

```
else
fprintf(fp,"#Voltage element%d =\tvariable;\n",eltNotSet);
};
fprintf(fp,"\n#Voltage range : %d;%d;\n",rangeMin,rangeMax);
fprintf(fp,"#step          : %d;\n",step);
fprintf(fp,"\n#Element%d\tpower (dB)\tpower (linear)\n",eltNotSet-1);

/* Measure the power */
for (eltVolt=rangeMin;eltVolt<=rangeMax;eltVolt+=step)
{
int dwret;

outbuf[eltNotSet-1] = (WORD)(eltVolt - VOLT_MIN);
dwret = AioOut(hdrv,&ipaout);
if (dwret!=0)
printf("Error in: DAOutput [0x%xh]\n",dwret);
Idling(300);

lpow = MeasurePower(Dev,ch);

fprintf(fp,"%d\t%f\t%2.6f\n",eltVolt,20.0*log10(lpow),lpow);
printf("%d\t%f\t%2.6f\n",eltVolt,20.0*log10(lpow),lpow);

}

return (1);
}


int Voltage_Power_Measure_all (int Dev,int ch,HANDLE hdrv,FILE *fp)
{
WORD  outbuf[NBVOLTAGE] = {0};
AOUT ipaout;

int step,rangeMin,rangeMax,k,eltVolt;
float lpow;

extern WORD Ch_Def[];

ipaout.Channels = NBVOLTAGE;
ipaout.OutBuf = outbuf;
ipaout.OutChNo = Ch_Def;
```

```
fflush(stdin);

rangeMin = VOLT_MIN-1;
rangeMax = VOLT_MAX+1;

while (rangeMin>rangeMax || rangeMin<VOLT_MIN || rangeMax>VOLT_MAX)
{
printf("Range minimun (%d:%d) . :",VOLT_MIN,VOLT_MAX);
scanf("%d",&rangeMin);
printf("Range maximun (%d:%d) . :",VOLT_MIN,VOLT_MAX);
scanf("%d",&rangeMax);
}

step = 0;
while (step<1 || step>1000)
{
printf("Step (1:1000) ......... :");
scanf("%d",&step);
}

fprintf(fp,"\n#Voltage range : %d;%d;\n",rangeMin,rangeMax);
fprintf(fp,"#step           : %d;\n",step);

/* Measure the power */
for (eltVolt=rangeMin;eltVolt<=rangeMax;eltVolt+=step)
{
int dwret;
for (k=0;k<ipaout.Channels;k++)
{
outbuf[k] = (WORD)(eltVolt - VOLT_MIN);
};

dwret = AioOut(hdrv,&ipaout);
if (dwret!=0)
printf("Error in: DAOutput [0x%xh]\n",dwret);
Idling(300);

lpow = MeasurePower(Dev,ch);

fprintf(fp,"%d\t%f\t%2.6f\n",eltVolt,20.0*log10(lpow),lpow);
printf("%d\t%f\t%2.6f\n",eltVolt,20.0*log10(lpow),lpow);

}
```

```
return (1);
}
```

mfunc.h

program name : mfunc.h
path: source/antennaGG2/mfunc.h
comments : Miscellaneous function declarations

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <windows.h>
#include <stdio.h>
#include <math.h>

#include "apiaio.h"
#include "dalink.h"

#ifndef MFUNC_H
#define MFUNC_H

int Voltage_Power_Measure (int Dev,int ch,HANDLE hdrv,FILE *fp);
int Set_frequence (int Dev,int ch,float freq_center,float freq_offset);
int Frequence_Power_Measure (int Dev,int ch,HANDLE hdrv,FILE *fp,float freq_center);
int Voltage_Power_Measure_all (int Dev,int ch,HANDLE hdrv,FILE *fp);

#endif
```

misc.c

program name : misc.c
path: source/antennaGG2/misc.c
comments : Miscellaneous functions

```
/*
    ESPAR control (C) Taillefer Eddy ATR-ACR 2001
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <windows.h>
#include <math.h>

#include "gpib.h"
#include "comhead.h"

/* ---------------------------------------------------------------- */
void recopy_f2f (int n,float *f1,float *f2)
{
int i;

for (i=0;i<n;i++)
f2[i] = f1[i];
}

/* ---------------------------------------------------------------- */
/* convert a  string of float value in a float table
first element is the table size
ex : "1.0,2.5E+02,0.003" -> {4; 1.0; 2.5E+02; 0.003}
*/

float *String2FloatTable(char *str)
{
/* this define creates some limitations */
static float ret[MAX_MEAS] = {(float)0.0};

if (str != NULL)
{
```

```
char *ptr;
int cpt,i;

/* count number of values */
ptr=str;
cpt=0;
while((*ptr!='\0') && *ptr!='\n')
{
if (*ptr++==',')
cpt++;
};

cpt = (cpt+1) % MAX_MEAS;
ret[0] = (float)cpt; /* array size = cpt + 1 */

/* get the values */
ptr=str;
for (i=1;i<cpt+1;i++)
{
sscanf(ptr,"%f",&ret[i]);
/* go to   next */
while((*ptr!='\0') && *ptr++!=',');
};
};

return (ret);
}
```