Front-End Processing for Monopulse Doppler Radar

Authors: P.H. Dezaux, X. Gilles, S. Marques

EFRIE, France

December 1995
SPRA299

TEXAS INSTRUMENTS

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Front-End Processing for Monopulse Doppler Radar

Abstract

FEPMR is standing for Front-End and Doppler Processing for Monopulse Doppler Radar. It is a PCB at the frontier between the RF and the Digital Circuits for the radar AXIR. There are four channels, corresponding to the four squinted beams of the antenna (Right, Left, Up and Down). Each beam channel is processed in phase (I) and quadrature (Q) real baseband signals.

The eight video analog signals are amplified and then converted by 8 ADC (Analog to Digital Converters). Four Texas Instruments (TI™) TMS320C50 dgital signal processors (DSPs) (one for each beam channel) are operating two major algorithms in cascade.

❑ A flexible FIR (Finite Impulse Response) Digital Filter with complex coefficients is first applied as a **Digital Pulse Compressor** to provide 16 range bins. AXIR transmitter is using a quadriphase sequence (1 up to 31 sub-pulses). Coefficients for the reference code are optimised by the Radar Manager (project C) to reduce the range side lobes.

❑ The second part implements three IIR (Infinite Impulse Response) Digital Filters, using also complex coefficients as a Doppler filter. These coefficients are dynamically computed by the Radar Manager (project C) as a function of the tracked target range and Doppler. An output time decimation is provided in accordance with the coherent and non-coherent integration process.

For each beam, each range cell and each Doppler, the I and Q signals are combined to form magnitudes to be delivered to the TRACKER (project B) via 4 dual-port RAM. At the end of the digital calculations, there are 4 * 16 * 3 bins, corresponding respectively to the 4 beams, the 16 ranges and the 3 Doppler filters. The output rate is fixed to 2 ms, independently of the flexible **Pulse Repetition Interval** (PRI) controlled dynamically by the Radar Manager.

This project is implemented on a Printed Circuit Board (PCB), using four TMS320C50. In a close future, a single TMS320C82 could probably do the same job within the same real time specifications because of its improvements.

This document was an entry in the 1995 DSP Solutions Challenge, an annual contest organized by TI to encourage students from around the world to find innovative ways to use DSPs. For more information on the TI DSP Solutions Challenge, see TI’s World Wide Web site at www.ti.com.

Product Support on the World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Introduction

This project deals with the study of a Front-End Processing for a Monopulse Doppler Radar (FEPMR), in charge of Pulse Compression and Doppler Digital Filtering. **AXIR** (**A**utomatic **X**band **I**nstrumentation **R**adar) is a concept of a low cost radar, specially designed for the TEXAS INSTRUMENTS DSP SOLUTIONS CHALLENGE. The idea was to use the performances of the TMS320C4x and TMS320C5x families to design a performant and intelligent radar processing, made of multiprocessors DSP chips.

The objective product cost is under 80.000 $ for the complete radar, including the pedestal, the antenna, the transmitter, the R.F. Front-End processors and the display. AXIR is basically a Monopulse Doppler Ground Based Radar with many civilian applications:

- Tracking of Meteo sounders.
- Cloud Doppler analysis for Meteo purposes.
- Short range air control for parallel runways or difficult access airports.
- Wind shear and burst detection.
- Trajectory control for sportive aerobatics events or air clubs.
- General low cost instrumentation radar (radar cross section evaluation, tutorial radar for universities, private air-tracking…).

The radar processor has been divided into four separate projects submitted to the TI challenge, respectively in charge of the following tasks:

A. FRONT END & DOPPLER PROCESSING (this project)
B. DIGITAL TRACKER.
C. RADAR MANAGER.
D. TARGET SIMULATOR.

AXIR features modern technology for the other sub-systems. The antenna is a planar Monopulse array of printed patches. The transmitter, the stalo and the R.F. homodyne Front-End are 100 % solid-state. All the circuits and the processors are located at the back of the thin antenna.

The pedestal uses robotics low cost and modern technology. The antenna box is free to rotate 360°, both in azimuth and elevation. A standard PC, under MICROSOFT WINDOWS / MSDOS is used as an operator remote control, graphics display, monitoring and recording. A radio link could be managed between the sensor and the operator desk. This project corresponds to the Printed Circuit Board number 1 (PCB #1).

Three undergraduates students from the Ecole FRancaise d’Electronique et d’Informatique have elaborated the hardware and the software of this project, under the supervision of an advising Professor.

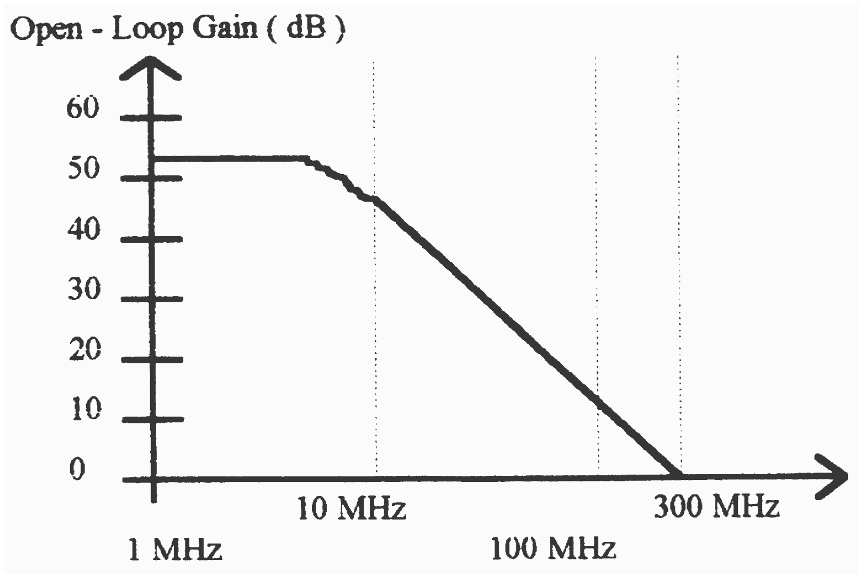
Overall Hardware/Software Description

Video Amplification

There are four channels, corresponding to the four squinted beams of the antenna (Right, Left, Up and Down). Each beam channel is processed in phase (I) and quadrature (Q) real baseband signals.

These eight signals are amplified by eight monolithic operational **video amplifiers** NE 5539 (Philips Semiconductors RF Communications Products). This model was selected for its wide bandwidth and its high slew rate. They provide a true differential input impedance device, with proper external compensation to design operations over a wide range of closed-loop gains. The amplifiers are providing 30 dB gain, a 2 MHz , bandwidth and a low noise $\left(4nV/\sqrt{Hz}\right)$.

Figure 1. Frequency Response of the Amplifier



These signals are sent to the digital converters.

Analog to Digital Conversion

Eight Analog to Digital Converters (Analog Devices AD876) are used to encode the four I & Q video signals. AD 876 is a CMOS 10-bits 20 MSPS sampling A / D converter which has a very good differential nonlinearity (0,5 LSB). By implementing a multistage pipeline architecture with output error correction logic, the AD 876 offers accurate performances and guarantees no missing codes.

The sampling frequency is set to 2 MHz (500 ns). The Radar Manager (project C) delivers the clock (2 Mhz) and the conversion window (signal named PRI).

This signal has to respect some time rules:

- ❑ The number of samples could be 17 to 47 (according to 16 range bins, plus 1 to 31 taps for the Digital Filter.
- ❑ PRI must be low between 8,5 µs (17 samples) and 23,5 µs (47 samples).
- ❑ PRI is periodic (50 µs to 100 µs) according to the repetition used by the radar. This value is computed by the Radar Manager.

An octal buffer and line driver with 3-states outputs (SN 74F244 from Texas Instrument) is used to latch the ADC outputs. This component is interesting because of its fast logic (switching characteristics of typically 4 ns). That means that the outputs can be stopped quickly, whereas the switching of the ADC is approximately 25 ns.

The data are converted 10 bits to 16 bits to fit the digital external data bus by copying out the MSB bit (d9) to the bits d9 up to d15.

Input Interface Between DSP and ADC

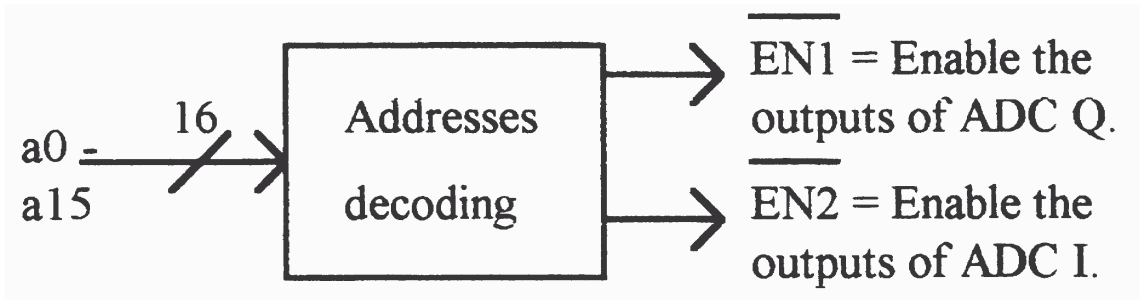
There are four DSP (TMS 320 C50), one per antenna beam, each interfaced with two ADC (refer to diagram Appendices B and C). Numerical Data are available by addressing the ADC as external memory. The addresses are 2C60h (channel Q) and 2C62h (channel I), chosen from the Data Memory Map.

When the PRI is low, the DSP takes in the samples, with the following instructions forming an interruption subroutine declenched by interruption IT1 (low active state during at least 3 consecutive machine cycles):

```
BLDD # 2C60h, AR1 ;AR1 initialised in the main with 0100h
BLDD # 2C62h, AR2 ;AR2 initialised in the main with 0200h
#AR1 ++
# AR2 ++
```

ADC Data reading is illustrated by Figure 2.

Figure 2. Addressing the ADC


a0 -
a15
16
Addresses decoding
$\overline{EN1}$ = Enable the outputs of ADC Q.
$\overline{EN2}$ = Enable the outputs of ADC I.

As the DSP is running with interruption IT1, the signal which will activate the interruption during at least 3 consecutive machine cycles have to be chosen. The input of this interruption is a low active state and as the frequency of new samples is 2 Mhz, a combination between the signal PRI and the 2 Mhz clock, which are synchroned because issued from the same Master Clock (project C), can be used as IT1.

Digital Pulse Compression

General Description

The aims of the Digital Pulse Compression system (DPC) are to reduce the peak transmitted power and the voltage / current ratio, keeping a good range resolution.

The DPC consists of a digital correlation between:

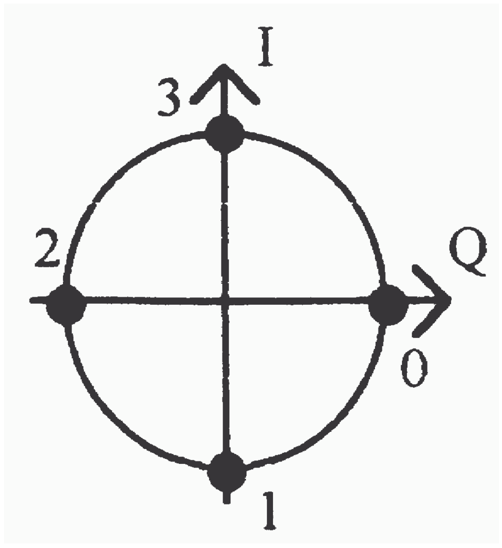
- the received signal from the target, corresponding to a delayed image of the transmitted waveform, modified by the Doppler effect.
- the reference code matched to the transmitter sequence.

This lead to the single pulse ambiguity which is the output amplitude as a 3d surface function of the range delay and the Doppler shift.

The samples used by the DPC are coming from the two ADC which operate on the received signal at a 500 ns period (range resolution of 75 meters). The Digital Correlation is performed on a part of this received video signal, according to the length of the reference code chosen (called Lx code). There are nine different codes used in the DPC system which length may vary from 1 up to 31 coefficients. The number of Digital Correlation outputs is equal to 16 (16 range bins). The number of samples taken from the ADC will be included between 17 (1 + 16) up to 47 (31 + 16).

The only information needed to calculate the DPC is the length of the reference code. This information is given by the Radar Manager (project C), using the serial port communication interface.

These codes are quadriphase codes which the four complex numbers are placed in the complex plane using the following positions:



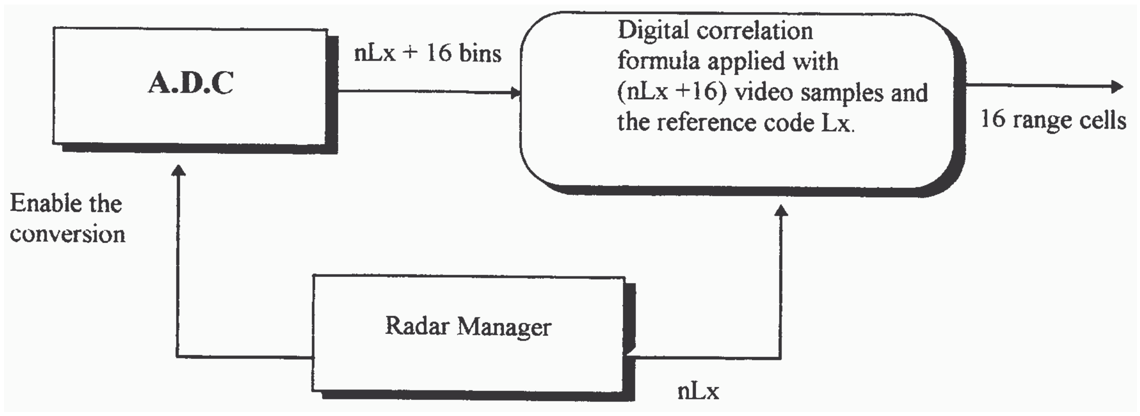
The result of one digital correlation is called a range cell.

List of the nine used reference codes:

Code(1)	:	**0**
Code(3)	:	**002** (BARKER)
Code(5)	:	**00020** (BARKER)
Code(7)	:	**0002202** (BARKER)
Code(11)	:	**00022202202** (BARKER)
Code(13)	:	**0000022002020** (BARKER)
Code(15)	:	**000110331231020** (BARKER)
Code(21)	:	**002222222002202020220**
Code(31)	:	**0012333000301301033212021310320**

Overview of the Calculation

Figure 3. Digital Pulse Compression


A.D.C
nLx + 16 bins
Digital correlation formula applied with (nLx +16) video samples and the reference code Lx.
16 range cells
Enable the conversion
Radar Manager
nLx

Digital Correlation Formula

$$DPC\_Result(k) = \sum_{j=0}^{nLx} Video[j+k].Lx[j] \qquad With\ k: 0 \rightarrow 15$$

DPC_Result (k)	: Output of the range cell number k.
nLx	: length of the reference code.
Video	: Inputs from the ADC (complex video bins).
Lx	: Reference code used.

To obtain the 16 range cells, the correlation is repeated 16 times by moving the variable k from 0 to 15.

Hardware Description and Algorithm

To speed up the DPC process, the nine codes used are stored in each DSP memory . Here is the memory location for each variable used for the DPC algorithm:

Complex video samples :	Real part : 0100h - 0l5Dh	Imaginary part : 0200h - 025Dh
Reference codes	Real part: 0460h - 068Dh	Imaginary part: 0700h - 092Dh
DPC results	Real part : 0300h - O31Fh	Imaginary part : 0320h - 033Fh
Working variables:	0800h - 087Fh	

The only information given by the Radar Manager (project C) is the number of the code (the nLx variable), which represents the correlation length. This code is set every 2 ms via the serial port. An internal interruption is detected each time the reference code has to be changed.

The program of the DPC in C language is given in the Appendix F.

The 16 range cells given by the DPC algorithm are the elementary informations taken from the received signal on a PRI period. They will be associated with the Doppler filtering which will give the information necessary to track the detected target.

After having simulated the algorithm in C language and in comparison with the arrays of calculation, the reason for ordering the data in memory and the importance of the shift implemented by the LTD instruction can be seen. To optimise real time and speed calculation, the for loop can be replaced by a shift of the data in memory (LTD). This instruction loads the T register with the contents of the address, adds the result of the previous multiply to the accumulator and shifts the data to the next higher address in data memory. Associated with MPY (multiplication of the contents of the T register with the contents of the address), the calculations are optimised for real time specifications.

```
ZAC                   ;Zero the accumulator
LTD    VideoQ         ;CoefR . Doppler
MPY    LxQ
LTS    VideoI         ;CoefQ . DopplerQ - CoefI . DopplerI
MPY    LxI
DMOV VideoI
```

These instructions can be repeated nLx times with the RPTB instruction, improving the time for calculations.

Doppler Processing

The Doppler Processing is used to reduce the bandwidth (coherent integration) and to reject fixed echoes. It is applied on the 16 complex range bins calculated by the Pulse Compressor. It corresponds to a complex coefficients **BandPass Filter** (BPF), centered around a specific Doppler frequency.

This filter is used 3 times, with 3 different center frequencies, on the 16 * 2 inputs (real and imaginary parts). So, the outputs contained 16 * 2 * 3 values. These 3 filters are Digital order one filters, based on a relationship between the input sequence DPC_Result (n) and the output sequence Doppler (n).

Calculations

The Z transfer of the three Doppler filters is:

$$H\left(z\right)=\frac{K}{1\text{-}C.z^{-1}}$$

Each filter uses 2 coefficients : K and C.

- K corresponds to a gain and is common to the 3 filters.

- C is the complex pole ($C = A \cdot e^{j.2\Pi.f} = \alpha + j \cdot \beta$).

The calculations are done for 3 Doppler frequencies ($f_{central}$, f_{lower}, f_{upper}) with the same amplitude K.

The 3 filters are calculated with the following recursion:

Doppler (n) = K . DPC_Result (n) + α . Doppler (n – 1)

with: $0 < n < 17$
Doppler (0) = 0
DPC_Result : Inputs from the Digital Pulse Compression.
Doppler : Outputs of the calculation.

The Effective Noise Bandwidth (ENBW) for this recursive filter is given by:

$$ENBW = \frac{1 - \|C\|}{1 + \|C\|}$$ where C is the complex pole.

The calculations are done for the 16 range cells (real and imaginary parts). The results give 6 arrays of 16 elements, loaded between 0400h and 045Fh in the data memory.

The initial conditions are set at the beginning of the tracking phase:

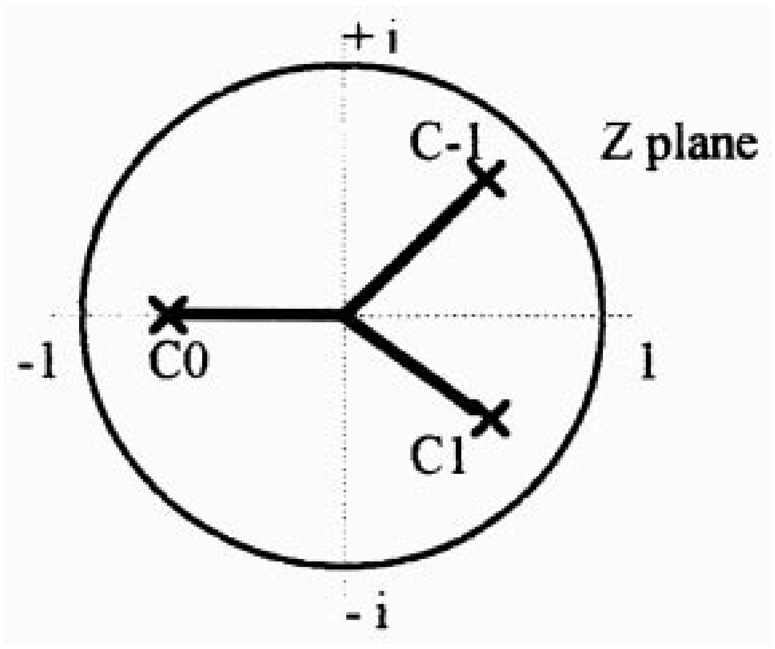
$$ENBW = \frac{1}{3} \Rightarrow \|C\| = \frac{1}{2}$$

$$f_{central} = \frac{1}{2}(C_0)$$

$$f_{lower} = \frac{1}{6}(C_{-1})$$

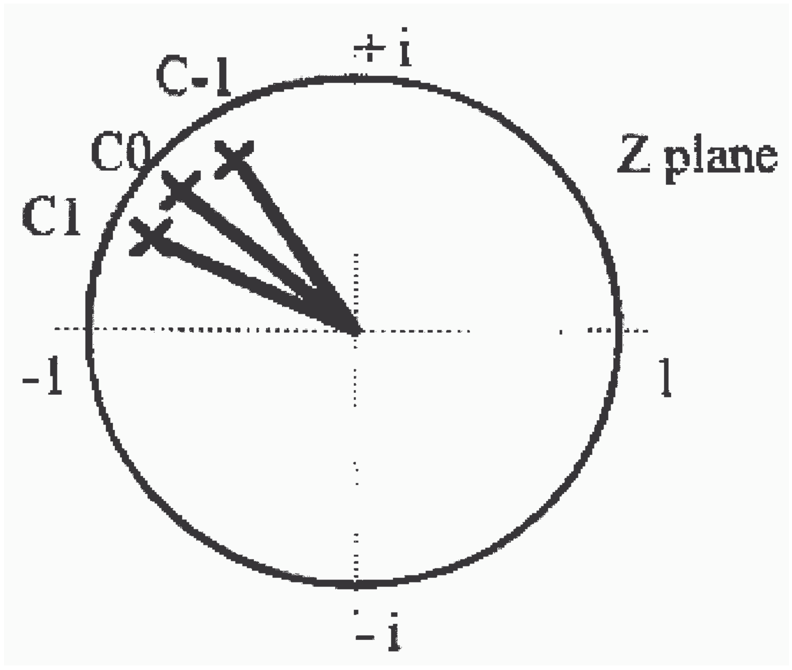
$$f_{upper} = \frac{5}{6}(C_1)$$

Figure 4. Initial Doppler Filters Poles for the Acquisition Phase



These coefficients (K, α = A. cos (2 . Π . f) and β = sin (2. Π. f)) are sent by the Radar Manager via the serial port and are loaded between addresses 0690h and 069Ch. Periodically, coefficients are re-calculated by the Radar Manager in order to match the target Doppler frequency and to reduce the bandwidth.

Figure 5. Poles of 3 Narrow Filters Around the Central Doppler Frequency During the Tracking Phase (Example)



The central frequency is deduced from the trajectory. The radar manager (project C) computes the estimated and smoothed Doppler coefficients.

C Program

The program in C language is specified before writing the ASM program and is given in the Appendix F. It can help to optimise the ASM program by taking advantages (for execution time and data memory) of the TMS C50 features. To maximise the real time calculations, the same instructions than in the DPC program are used : LTD, MPY, DMOV and RPTB.

Approximate Modulus

Magnitudes of previous results are calculated and transmitted to the Digital Tracker.

The following approximate formula is used:

$$M = |I| + |Q| + \frac{1}{2}\left||I| - |Q|\right|$$

ABS is available in the TMS320C50 instruction set. This formula is computed on the 16 range cells * 2 (Q and I parts) * 3 Doppler for each beam. Results are corresponding to 16 range cells * 3 Dopplers * 4 beams.

Output Interface with the Digital Tracker

To share the 16 * 3 * 4 values with the Tracker, four **Dual Port RAM** are used. According to the documentation from Integrated Device Technology (IDT), the Dual Port RAM allows 2 independent devices to have simultaneous read and write access to the same memory. This allows the 2 devices to communicate with each other by passing data through the common memory. So, a dual-port memory has 2 sets of address, data and read / write control signals, each of them access the same set of memory cells. The product 7133 from IDT allows a memory of 2Kword on 16 bits. With a good access time, the only problem about using this chip is about writing into the same cell at the same time. But as this project only writes the data and as the Tracker has only to read the data at a different time, the problem never occurs.

This memory has been declared between addresses 2C00h and 2C5Fh, according to the memory map (external device).

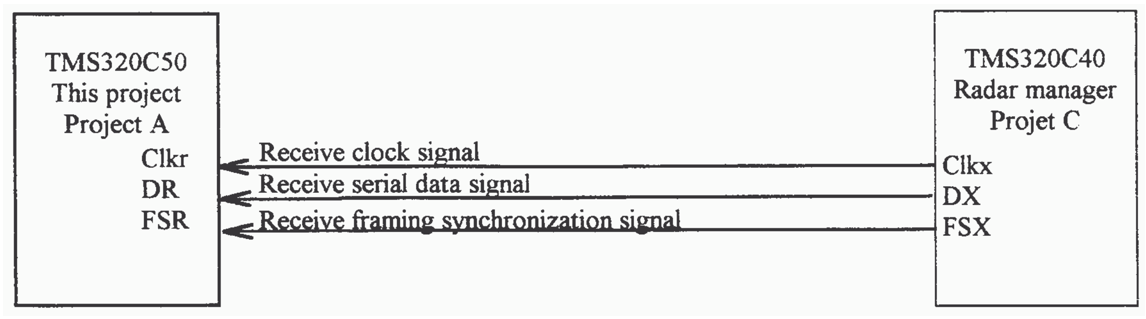
Communication with the Radar Manager

By the use of a serial link between this project and the Radar Manager (project C), a refresh of global variables is proceeded. These global variables are used for the 2 digital filters which have been presented precedently. The maximum operating frequency of the serial port while using internal clocks is the frequency of CLKOUT divided by four (5 Mbit/s at 50 ns ; 7,14 Mbit/s at 35 ns).

Architecture

The signals CLKR (**R**eceive **Cl**ock **S**ignal), DR (**R**eceive serial **D**ata signal), FSR (**R**eceive **F**raming synchronisation **S**ignal) of our DSP's are connected to the Radar Manager' DSP (project C).

Figure 6. Communication via the Serial Port



Configuration

The use of the serial port receiver needs to program three memory registers :

		Hexadecimal address
SPC	**S**erial **P**ort **C**ontrol register	22h
DRR	**D**ata **R**eceive **R**egister	20h
IMR	**I**nterrupt **M**ask **R**egister	4h
IFR	**I**nterrupt **F**lag **R**egister	6h

The register SPC has been programmed this way :

bit I	DLB, value 0	*Digital Loopback mode Bit, not used*
bit 2	FO, value 0	*16 bits transfer*
bit 3	FSM, value 0	*Frame Synch Mode Bit, no frame sync pulse required*
bit 4	MCM, value 0	*Clkx is taken from the Clx pin and not from internal clock*
bit 5	TXM, value 0	*FSX pin is configured as an input*
bit 6&7	XRST, RRST	*O for reset the transmitter and the receiver, 1 for running*
bit 14	SOFT, value 0	*not used*
bit 15	FREE, value 0	*free run (1=stop)*

Other bits can't be written and are unused, except :

bit 10 RRDY *Receive Ready bit, a transition from 0 to 1 indicates that the data can be read.*

When a byte is received, an internal interrupt RINT (serial port Receive INTerrupt) is generated.

This interrupt must not be masked in the Interrupt Mask Register (IMR, dec 4, hex 4) in order to execute procedure *change_global_variables* :

bit 4 (Rint), has to be equal to 1 which enables the corresponding interrupt.

Global Variables Needed for Digital Calculations

The first filter (Digital Pulse Compression) only needs one variable : a number between 1 and 9 which corresponds to the choice of the Lx code (9 different codes).

- ❑ **'L'** : integer 16 bits numero of LX code.

The second filter (Doppler Processing) computes different variables :

- ❑ **'K'** : integer (16 bits) Amplitude coefficient of the filter.
- ❑ **'i', 'c', 's'** :integers (16 bits) Real part (α) for the inferior, central and superior frequencies.
- ❑ **'I' 'C' 'S'** :integers (16 bits) Imaginary part (β) for the inferior ,central and superior frequencies.

These are the only variables that this project needs for the calculations.

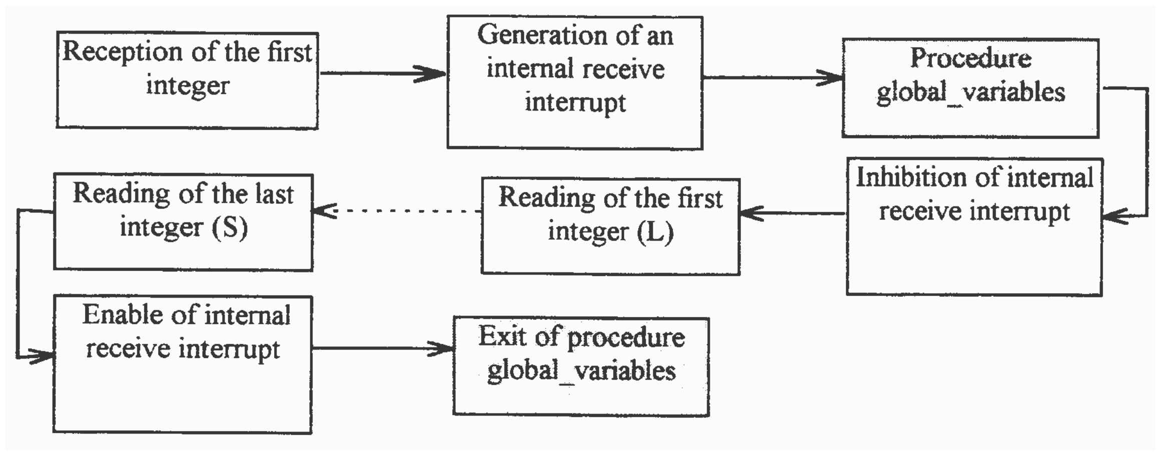
Transmission Protocol

Two different methods have been studied:

- ❑ The Radar Manager (project C) transmits only the variables that need to be changed (for example : K, 1200, i, 1300, C, 2400). The main advantage of this method consists in the fact that the Radar Manager doesn't need to transmit the eight variables. But it needs a lot of cycles at the reception for the tests to recognise the variables.
- ❑ An other method consists to transmit the 8 words (16 bits). When the receive buffer is full, the integer is placed into the corresponding variable. This method has been retained because of its speed : all the variables are sent every 2 ms.

The starting reception is detected by an interrupt signal. This signal is generated when the first integer is received. Then the procedure global_variables is executed.

Figure 7. Reception Diagram


Reception of the first integer
Generation of an internal receive interrupt
Procedure global_variables
Inhibition of internal receive interrupt
Reading of the first integer (L)
Reading of the last integer (S)
Enable of internal receive interrupt
Exit of procedure global_variables

Performances and Conclusion

This section concerns the effects of the 2 filters on filter performances. Indeed, some features of filters include phase characteristics, stability and coefficient quantization effects. An important consideration is the stability of the filter. The Digital Pulse Compression is inherently stable (i.e., a bounded input always produces a bounded output). On the other hand, the Doppler Processing may or may not be stable, depending on the location of the pole of the filter.

Digital filters are designed with the assumption that the filter will be implemented on an infinite precision device. However, as all processors are of finite precision, it is necessary to approximate the ‘ ideal ‘ filter coefficients. This approximation introduces coefficients quantization error. The net result due to imprecise coefficient representations is a deviation of the resultant filter frequency response from the ideal one.

Another problem in implementing a digital filter is the quantization error due to the finite wordlength effect in the hardware. Source of error arising from the use of finite wordlength include the following :

- I / 0 signal quantization (ADC conversion).
- Filter coefficient quantization.
- Correlated roundoff noise.
- Dynamic range constraints.

The advantages of using **digital filters** over their Analog counterparts are:

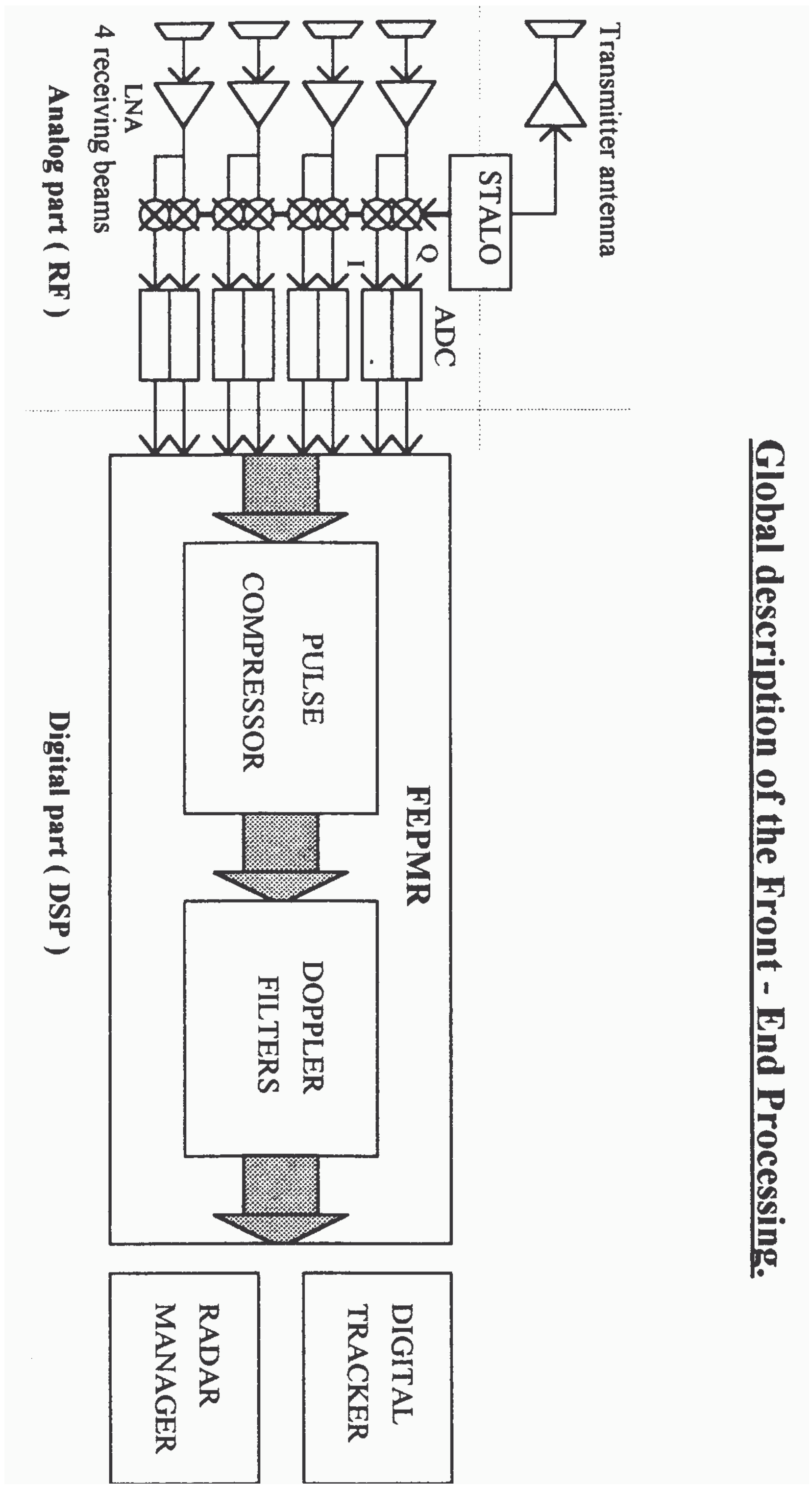
- high reliability.
- high accuracy.
- no effect of component drift on system.
- component tolerance not critical.
- Ease for changing filter parameters.

The 16-bit coefficients and the 32-bit accumulator of the TMS 320 processors help minimise the quantization effects. Special instructions also help to overcome problems in the accumulator. These features, in addition to a powerful instruction set and a fast clock timing, make the TMS320C50 ideal programmable processors for our application.

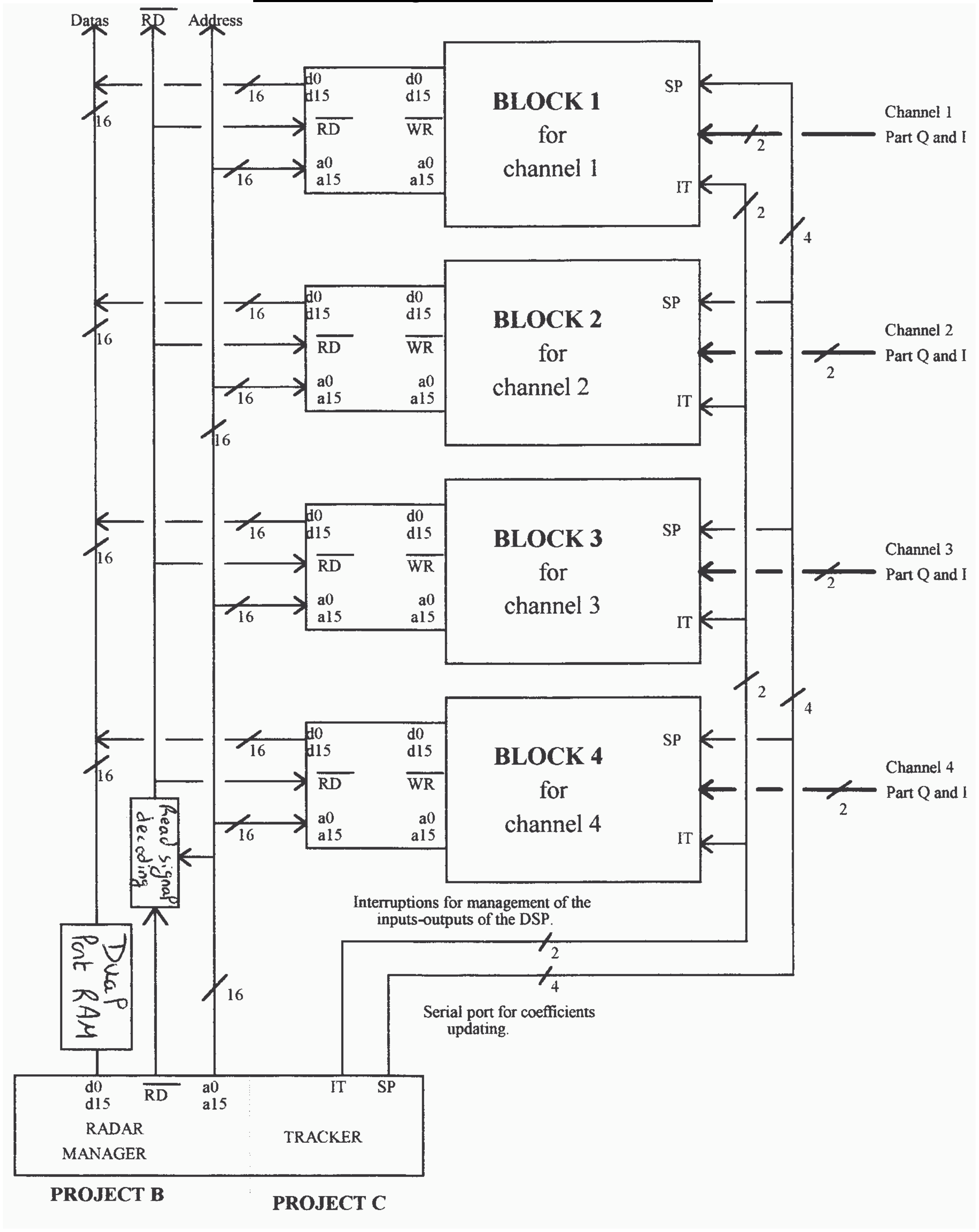
In the worst case of the PRI, there is only 50 µs (shortest period of the PRI) - 23,5 µs (47 samples of 500 ns each) = 26,5 µs for the digital calculations, before the next samples. Indeed, the radar must work in real time, so it can't memorise and calculate later. This is the main reason for using 4 DSP C50 in parallel : the 47 * 4 samples (worst case) can't be loaded and 1 * 4 + 3 * 4 + 1 * 4 = 20 algorithms can't be processed in real time. With 4 DSP, we have only 47 samples and 1 + 3 + 1 = 5 algorithms to process them one after the others and in the same time than the first case. Indeed, the first case needs to compute with 45 MIPS whereas the second one needs only 15 MIPS per DSP. And DSP TMS 320 C50 allows a 35 / 50 ns single-cycle fixed-point instruction execution time (28.6 / 20 MIPS).

This project demonstrates that a typical processing of Monopulse Doppler Radar implemented on only 4 chips, realising 16 range cells (500 ns) * 3 complex Doppler filters * 4 beams is equivalent to 4 * 15 MIPS. It represents 75 % of the time possibilities offered by the 4 DSP TMS320C50.

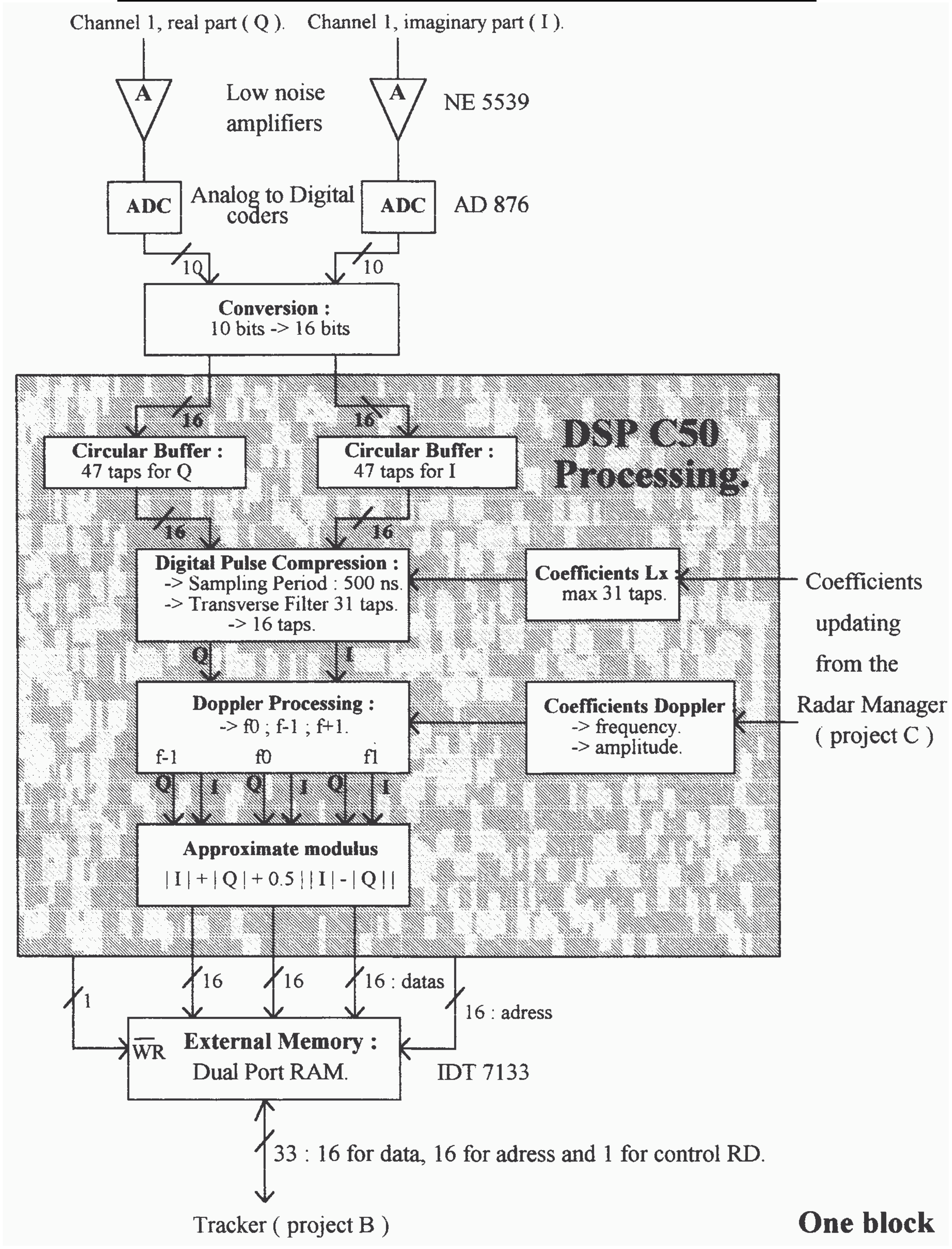
Appendices


Global description of the Front - End Processing.

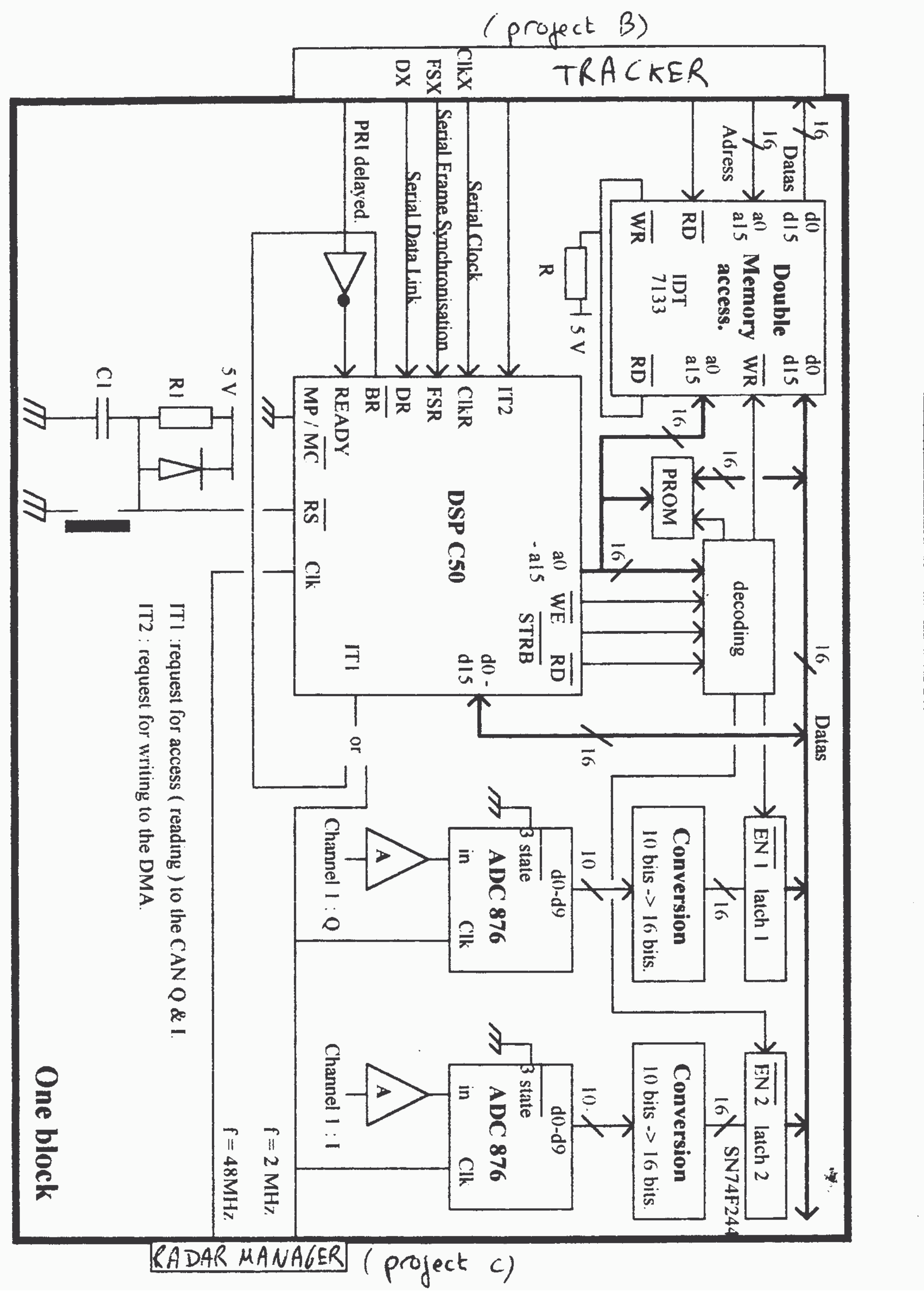
Global Diagram of the 4 Beams



Functional Diagram for One Antenna Channel


Channel 1, real part (Q).
Channel 1, imaginary part (I).
A
Low noise amplifiers
A
NE 5539
ADC
Analog to Digital coders
ADC
AD 876
10
10
Conversion : 10 bits -> 16 bits
16
16
Circular Buffer : 47 taps for Q
Circular Buffer : 47 taps for I
DSP C50 Processing.
16
16
Digital Pulse Compression : -> Sampling Period : 500 ns. -> Transverse Filter 31 taps. -> 16 taps.
Coefficients Lx : max 31 taps.
Coefficients updating from the Radar Manager (project C)
Q
I
Doppler Processing : -> f0 ; f-1 ; f+1.
f-1
f0
f1
Coefficients Doppler -> frequency. -> amplitude.
Q I Q I Q I
Approximate modulus
| I | + | Q | + 0.5 | | I | - | Q | |
1
16
16
16 : datas
16 : adress
WR
External Memory : Dual Port RAM.
IDT 7133
33 : 16 for data, 16 for adress and 1 for control RD.
Tracker (project B)
One block

Electrical diagram for one channel.



Memory Allocation Reserved for the ASM Program Inside the DSP

VideoR	0100	015D	47 values of 16 bits
VideoI	0200	025D	47 values of 16 bits
CompressR	0300	031F	16 values of 16 bits
CompressI	0320	033F	16 values of 16 bits
Doppler0R	0340	035F	16 values of 16 bits
Doppler0I	0360	037F	16 values of 16 bits
Doppler1R	0380	039F	16 values of 16 bits
Doppler1I	03A0	03BF	16 values of 16 bits
Doppler2R	03C0	03DF	16 values of 16 bits
Doppler2I	03E0	03FF	16 values of 16 bits
Working variables	0800	087F	64 variables of 16 bits
Doppler0	0400	041F	16 values of 16 bits
Doppler1	0420	043F	16 values of 16 bits
Doppler2	0440	045F	16 values of 16 bits
LxR	0460	068D	31*9 values of 16 bits
CoefR	0690	0695	3 values of 16 bits
CoefI	00696	069C	3 values of 16 bits
Lx1	0700	092D	31*9 values of 16 bits
Mémoire Double accès	2C00	2C5F	48 values of 16 bits
CAN Q	2C60	2C61	1 value of 16 bits
CAN I	2C62	2C63	1 value of 16 bits

All the addresses are given in hexadecimal and refer to the DATA memory and not to the program memory.

C Program for the Front - End Processing

```
#include <stdio.h>              #include <math.h>   #include <conio.h>
#define nDPC_Result 16

/* Init of the LX coefficients. */
void initLx ( int nlx, int LxQ[], int LxI[] )

{       int i;

        for  ( i=0 ; i < nlx; i++ )
                {LxQ [I] = 0;
                LxI [i] = 0;
                }
        LxQ [0] = 1;

        switch ( nlx )
        { case 3 :      LxQ [1] = 1;    LxQ [2] = -1;
                        break;
        case 5 :        LxQ [1] = 1;    LxQ [2] = 1;    LxQ [3] = -1;   LxQ [4] = 1;
                        break;
        case7 :         LxQ [1] = 1;    LxQ [2] = 1;    LxQ [3] = -1;   LxQ [4] = -1    LxQ [5] = 1;
                        LxQ [6] = -1;
                        break;
        case 11:        LxQ [1] = 1;    LxQ [2] = 1;    LxQ [3] = -1    LxQ [4] = -1;   LxQ [5] = -1;
                        LxQ [6] = 1;    LxQ [7] = -1;   LxQ [8] = -1;   LxQ [9] = 1;    LxQ [10] = -1;
                        break;
        case l3:        LxQ [1] = 1;    LxQ [2] = 1;    LxQ [3] = 1;    LxQ [4] = 1;    LxQ [5] = -1;
                        LxQ [6] = -1;   LxQ [7] = 1;    LxQ [8] = 1;    LxQ [9] = -1;   LxQ [10] = 1;
                        LxQ [11] = -1;  LxQ [12] = 1;
                        break;
        case 15:        LxQ [1] = 1;    LxQ [2] = 1;    LxQ [5] = 1;    LxQ [9] = -1;   LxQ [12] = 1;
                        LxQ [13] = -1;  LxQ [14] = 1;   LxQ [3] = 1;    LxQ [4] = 1;    LxQ [6] = -1;
                        LxQ [7] = -1;   LxQ [8] = 1;    LxQ [10] = -1;  LxQ [11] = 1;
                        break;
        case 2l:        LxQ [1] = 1;    LxQ [2] = -1;   LxQ [3] = -1;   LxQ [4] = -1;   LxQ [5] = -1;
                        LxQ [6] = -1;   LxQ [7] = -1;   LxQ [8] = -1;   LxQ [9] = 1;    LxQ [10] = 1;
                        LxQ [11] = -1;  LxQ [12] = -1;  LxQ [13] = 1;   LxQ [14] = -1;  LxQ [15] = 1;
                        LxQ [16] = -1;  LxQ [17] = 1;   LxQ [18] = -1;  LxQ [19] = -1;  LxQ [20] = 1
                        break;
        case 3l:        LxQ [1] = 1;    LxQ [3] = -1;   LxQ [7] = 1;    LxQ [8] = 1;    LxQ [9] = 1;
                        LxQ [11] = 1;   LxQ [14] = 1;   LxQ [16] = 1;   LxQ [19] = -1;  LxQ [21] = -1;
                        LxQ [22] = 1;   LxQ [23] = -1;  LxQ [27] = 1;   LxQ [29] = -1;  LxQ [30] = 1;
                        LxQ [2] = 1;    LxQ [4] = -1;   LxQ [5] = -1;   LxQ [6] = -1;   LxQ [10] = -1;
                        LxQ [12] = 1;   LxQ [13] = -1;  LxQ [15] = 1;   LxQ [17] = -1;  LxQ [18] = -1;
                        LxQ [20] = 1;   LxQ [24] = 1;   LxQ [25] = -1;  LxQ [26] = 1;   LxQ [28] = -1;

        }
}

/* Init of the DSP inputs : video. */
void initvideo ( int nvideo, int videoQ[ ], int videoI[ ] )
{       int i;

             for ( i =  0 ; i < nvideo ; i++)
                    asm
                    {           MOVE 2C60, @videoQ [i];
                                MOVE 2C62, @videoI [i];
                    }
}

/* Init of the arrays used for the results. */
void initDPC_Result ( int DPC_ResultQ [ ] int DPC_ResultI [ ] )

{         int i;

          for ( i = 0 ; i <nDPC_Result ; i++ )
                    {      DPC_ResultQ [I] = 0;
                           DPC_ResultI [i] = 0;
                    }
}

/* Writing to the screen 2 integer arrays. */
void affiche ( int nvideo, int LxQ[ ], int LxI[ ], int ntxt, char texte[ ] )


{        int i;


         clrscr ( );
         for ( i =0 ; i < ntxt – 1 ; i++)
         {      printf ( "%c", texte [i] );
         }
         printf ( "%c \n", texte [ntxt – 1 ] );
         for ( i = 0 ; i <nvideo ; i++ )
         {      printf ( "%2d : Valeur %4d %4d \n", i, LxQ [i], LxI [i] );
         }
         I = getch ( );
}

/* Calculation of Doppler coefficients from the frequencies and the amplitude delivered by the project C via the serial port.
          => normally designed by the projet C, but here for the simulation. */
void calcul_coef ( float fzero, float fmoins, float fplus, float amp, int coefQ [ ], int coefI [ ] )

{        coefQ [0] = ( int ) amp * cos ( 2 * 180 * fzero );
         coefI [0] = ( int ) amp * sin ( 2 * 180 * fzero );
         coefQ [1] = ( int ) amp * cos ( 2 * 180 * fmoins );
         coefI [1] = ( int ) amp * sin ( 2 * 180 * fmoins );
         coefQ [2] = ( int ) amp * cos ( 2 * 180 * fplus );
         coefI [2] = ( int ) amp * sin ( 2 * 180 * fplus);
}

/* Output to the DMA of the results. */
void ecrire_ext ( int doppler0 [ ], int doppler1 [ ], int doppler2 [ ] )


{         int i;

          for ( i = 0 ; i <nDPC_Result ; i++ )
          asm
          {      MOVE @doppler0 [i], 2C00 + i
           }
          for ( i = 0 ; i <nDPC_Result ; i++ )
          asm
          {      MOVE @doppler1 [i], 2C20 + i;
           }
          for ( i = 0 ; i <nDPC_Result ; i++ )
          asm
```

```
            {       MOVE @doppler2 [i], 2C40 + i;
             }
}

/* Main program. */

int main ( )

{           int nlx, nvideo, i, j;
            int LxQ [31], LxI [31];
            float fzero, fmoins, fplus, amp, K;
            int videoQ [47], videoI [47], DPC_ResultQ [nDPC_Result], DPC_ResultI [nDPC_Result], coefQ [3],
                    coefI [3], doppler0 [nDPC_Result], doppler1 [nDPC_Result], doppler2 [nDPC_Result];
            int doppler0Q [nDPC_Result], doppler0I [nDPC_Result], doppler1Q [nDPC_Result], doppler1I [nDPC_Result],
                    doppler2Q [nDPC_Result], doppler2I [nDPC_Result];

            nlx = 15;               /*Max value=31. */
            nvideo = nlx + nDPC_Result

            /* Init of the nlx divisions of LxQ and LxI. */
            initLx ( nlx, LxQ, LxI);
            affiche ( nlx, LxQ, LxI, 2, "Lx");

            /* Init of the nvideo divisions of videoQ and videoI. */
            initvideo ( nvideo, videoQ, videoI );
            affiche ( nvideo, videoQ, videoI, 8, "video in");

            /* Init to 0 of the nDPC_Result divisions of DPC_ResultQ and DPC_ResultI. */
            initDPC_Result ( DPC_ResultQ, DPC_ResultI );
            affiche ( nDPC_Result, DPC_ResultQ, DPC_ResultI, 8, "DPC_Result" );

            /* Calculation of the Digital Pulse Compression. */
            for ( i = 0 ; i <nDPC_Result ; i++ )
            {       for ( j=0 ; j < nlx ; j++ )
                    {       DPC_ResultQ [i] = DPC_ResultQ [i] + LxQ [j] * videoQ [i+j] - LxI [j] * videoI [i+j];
                            DPC_ResultI [i] = DPC_ResultI [i] + LxQ [j] * videoI [i+j] + LxI [j] * videoQ [i+j];
                    }
             }
            affiche ( nDPC_Result, DPC_ResultQ, DPC_ResultI, 8, "DPC_Result" );

            /* Coefficients from the project C send by the serial port. */
            fzero = 10.235;
            fmoins = 7.59;
            fplus = 11.15;
            amp = 5;  /* These values have been chosen at random for the simulation.*/
            K = 1;
            calcul_coef ( fzero, fmoins, fplus, amp, coefQ, coefI );

            /* Calculation of the Doppler Processing. */
            doppler0Q [0] = K * DPC_ResultQ [0];
            doppler0I [0] = K * DPC_ResultI [0];
            doppler 1 Q [0] = K * DPC_ResultQ [0];
            doppler1I [0] = K * DPC_ResultI [0];
            doppler2Q [0] = K * DPC_ResultQ [0];
            doppler2I [0] = K * DPC_ResultI [0];
            for ( i = 1; i < nDPC_Result ; i++ )
            {       doppleroQ [i] = K*DPC_ResultQ [i] + coefQ [0]*doppler0Q [i-1] - coefI [0]*doppler0I [i-1];
                    doppler0I [i] = K*DPC_ResultI [i] + coefI [0]*doppler0Q [i-1] + coefQ [0]*doppler0I [I-1];
```

```
                doppler1Q [i] = K*DPC_ResultQ [i] + coefQ [1]*doppler1Q [i-1] - coefI [1]*doppler1I [I-1];
                doppler1I [i] = K*DPC_ResultI [i] + coefI [1]*doppler1Q [i-1] + coefQ[1]*doppler1I [I-1];
                doppler2Q [i] = K*DPC_ResultQ [i] + coefQ [2]*doppler2Q [i-1] - coefI [2]*doppler2I [i 1];
                doppler2I [i] = K*DPC_ResultI [i] + coefI [2]*doppler2Q [i-1] + coefQ [2]*doppler2I [I-1];
    }
    affiche ( nDPC_Result, doppler0Q, doppler0I, 7, "doppler0" );
    affiche ( nDPC_Result, doppler1Q, doppler1I, 7, "doppler1" );
    affiche ( NPC_Result, doppler2Q, doppler2I, 7, "doppler2");

    /* Calculation of the approximate modulus. */
    for ( i = 0 ; i < nDPC_Result ; i++ )
    {       doppler0 [I] = fabs ( doppler0I [i]) + fabs ( doppler0Q [I] ) + 0.5 * fabs ( fabs ( doppler0I [i] ) –
                      fabs ( doppler0Q [i] ) );
            doppler1 [i] = fabs ( doppler1I [I] ) + fabs ( doppler1Q [i] ) + 0.5 * fabs ( fabs ( doppler1I [I] ) –
                      fabs ( doppler1Q [i] ) );
            doppler2 [I] = fabs ( doppler2I [I] ) + fabs ( doppler2Q [i] ) = 0.5 * fabs ( fabs ( doppler2I [i] ) –
                      fabs ( doppler2Q [i] ) );
    }

    affiche ( nDPC_Result, doppler0, doppler1, 15, "doppler0 + doppler1" );
    affiche ( nDPC_Result, doppler1, doppler2, 15, "doppler1 + doppler2 " );

    /* Output of the results. */
    ecrire_ext ( doppler0, doppler1, doppler2 );
}
```

C and Asm Programs for the Serial Communication

```
// Constant //
#define nb_word_transfered          8
#define begin_adr                   Ox9OOOh          //9000h->9016h//
#define adr_serial_buffer           32

// Global variables //
L, K, I, c, s, I, C, S            :integer;

// Reception of interruption Rint signales that the first byte is being transmitted //
// Execution of procedure void change_global_variables (void) //

void change_global_variables (void)

// Local variables //
integer *buffer_DRR;
integer *register;

{
        register=04h;                              // adress of the register IMR //
        *register=0h;                              // inhibit interruptions //
        register=022h;                             // adress of the register SPC //
        buffer_DRR = adr_serial_buffer;
        K= *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        L= *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        i = *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        c= *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        s= *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        I= *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        C= *buffer_DRR;
        repeat until (*register & 04 00h == 1)     // wait until bit RRDY==1; an integer is received //
        S= *buffer_DRR;
        register=04h;                              // adress of the register IMR //
        *register = 0l0h;                          // allow RINT interruption //
}
```

First, the vector has to be stored in the RAM:

```
        LAMM     change_global_variables    ;ACC=ISR adress
        BACC                                ;Branch to ISR
```

In the main program there is an initialization part:

```
        SPLK     #0008h, SPC          ; Set SP as CLK; frame sync receive
                                      ; Set TXM=MCM=DLB=FO=0, FSM=1
                                      ; and put SP into reset (XRST=RRST=0)
        SPLK     #00C8h, SPC          ; Take SP out of reset, setup interrupts
```

```
            SPLK        #0FFFFh, IFR             ; Clear IFR
            SPLK        #010h, IMR               ; Turn on RINT
            CLRC        INTM                     ; Enable interrupts
            LAR         AR7, #9000h              ; Setup where to write received data
            CLRC         XF                      ; Signal ready to receive
SELF1       B           SELF 1                   ; Wait for interrupts
```

The interrupt subroutine change_global_variables is:

```
CHANGE_GLOBAL_VARiABLES
            LACL        DRR               ; Load received value
            SACL        *+                ; Write to memory block
TEST1       BIT *,5,SPC                   ; wait transition from 0 to 1 of bit 10 (RRDY) of SPC
            BCND TEST1, NTC               ; TC=0, return to TEST1
NEXT        LACL         DRR              ; Load received value
            SACL         *+               ; Write to memory block
            ….
            RETE                          ; Clears INTM (enable interrupt) and execute RETI,
                                          ; return from interrupt
```

Glossary

ADC	:	Analog to Digital Converter.
AXIR	:	Automatic Xband Instrumentation Radar.
CLKR	:	Receive CLocK
DMA	:	Double Memory Access.
DR	:	Receive serial Data signal
DRR	:	Data Receive Register
DSP	:	Digital Signal Processor.
FIR	:	Finite Impulse Response.
FSR	:	Receive Framing Synchronisation signal
IFR	:	Interrupt Flag Register
IIR	:	Infinite Impulse Response.
IMR	:	Interrupt Mask Register
MSB	:	Most Significant Bit
PRF	:	Pulse Repetition Frequency.
PRI	:	Pulse Repetition Interval.
R.F.	:	Radio Frequency.
SPC	:	Serial Port Control register