Disclaimer: This document was part of the DSP Solution Challenge 1995 European Team Papers. It may have been written by someone whose native language is not English. TI assumes no liability for the quality of writing and/or the accuracy of the information contained herein.

Searching for the Best Quadriphase Codes with the TMS320C50 DSK

Authors: D. Goupy, M. Dromigny, P. Lizaraburu, B. Cornillault, L. Fourdan

EFRIE, France

December 1995
SPRA300

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

Tables

Searching for the Best Quadriphase Codes with the TMS320C50 DSK

Abstract

This application report presents a study concerning the search for the best quadriphase complex sequences to use in digital pulse compression (DPC) radar sub-systems.

Presently, only an exhaustive search is able to insure the discovery of best codes. This type of research requires a system with high computational power to scan a very important number of sequences.

The Texas Instruments (TI™) TMS320C50 Digital Signal Processor (DSP) Starter Kit (DSK) is used for this purpose. The programs for this study were coded in ANSI C and translated into Assembly language by using TI software.

A link between the DSK and a standard PC, loaded with Microsoft Windows/DOS, is used to save the results in files on the hard disk.

The speed improvement achieved by using the TI TMS320C50 DSK was more than a factor of 10 better as compared to a search using compiled PASCAL or C programs.

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

We are four French students in a leading engineering school in Paris. Ecole Française d'Electronique et d'Informatique offers an engineering diploma in electronics and computer science. We are all in the electronics program.

Other groups of students in the same school are also preparing for the Texas Instruments DSP Solutions Challenge in the domain of pulse Doppler radar. Our project deals with the pulse compression subsystem.

During our studies in electronics and computer science, we have been particularly interested in the Texas Instruments DSP international contest. We anticipate taking other radar courses in the future and preparing for those courses by reading available books and papers. Radar technology is new and complex but very interesting. We were able to complete our project with the help of one of our teachers who is a specialist in radar technology.

We hope that this report will be helpful to anyone who is interested in digital signal processing.

Fundamentals of Digital Pulse Compression

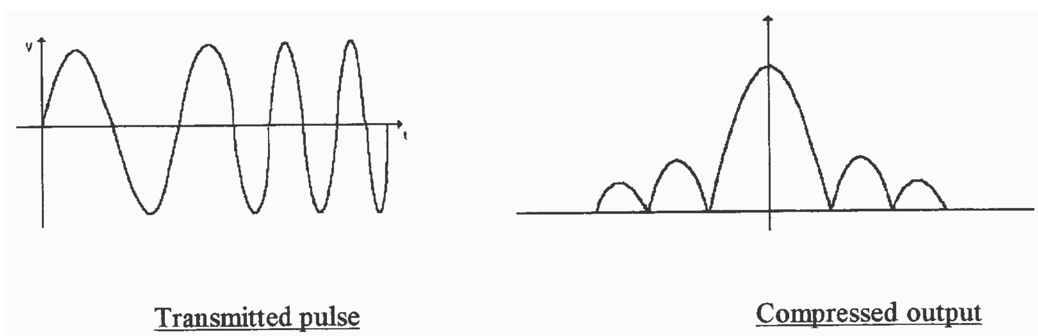
Pulse compression is a technique used in pulsed coherent radar to increase the energy transmitted per pulse without sacrificing the range resolution. This technique consists of raising the time-bandwidth product of the pulse by a particular coding of the frequency or phase in each pulse. The sidelobes must be lowered to avoid hiding other targets.

Several methods are employed to realize the phase modulation of the pulse and the coherent processing. A modern system is based on a discrete phase coding of the transmitting carrier during the pulse, associated with a coherent digital signal processing that is a complex correlator. There are two principal methods for pulse compression:

- Analog devices using dispersive delay lines
- Digital correlators

Coherent signal processing requires that both the magnitude and phase of signal samples be processed. (as shown in Figure 1)

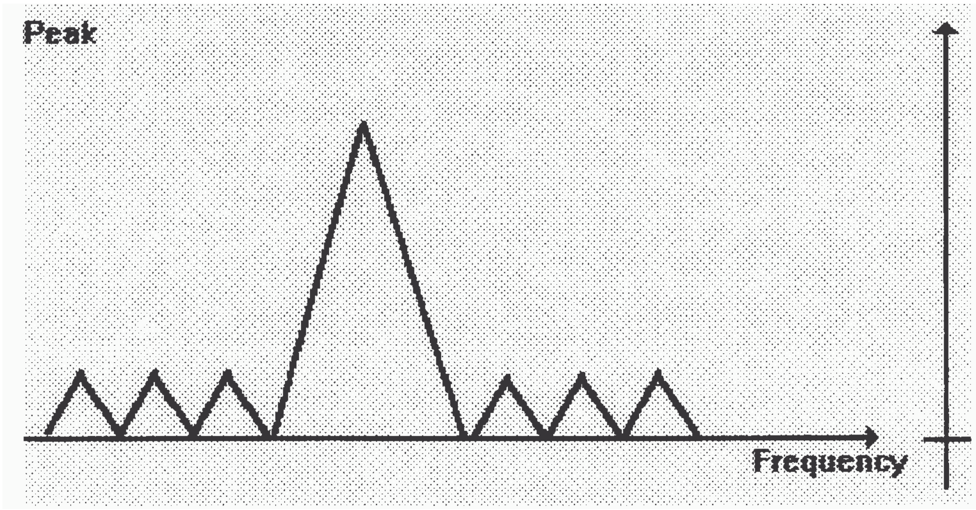
Figure 1. Coherent Signal Processing Components



The various types of modulation used in pulse compression are called codes. Some of the best-known discrete codes are included in the generic term *polyphase codes*. *Biphase* (binary) or *four-phase* (quaternary) codes are simple to use because each considers only one or two real components, utilizing only two values, +1 or -1.

The quality of a code is determined when evaluating the Aperiodic correlation function (ACF) between the transmitted code and the matched model (complex coefficients of the FIR correlator) as illustrated by Figure 2.

Figure 2. Code Quality Diagram



Good codes present minimum peak sidelobes and/or minimum overall effective sidelobes (rms level).

Several theorems have been written for the research of polyphase codes but the mathematical results are essentially negatives. There are as yet no theoretical results about this code that reveals something about their structure. The only way to find all the best polyphase codes is to test them all one by one with a performant calculator.

The search for good sequences is of great interest since there is currently no means of finding best codes mathematically.

Exhaustive search (among the collection of all possible codes) is fundamental but suffers from the combinatorial number explosion. The number of distinct codes is 2^L or 4^L (where L is the sequence length) depending on whether the base is binary or quaternary, respectively.

The primary purpose of this project is to use the TMS320C50 DSP Starter Kit from Texas Instruments to increase the speed of the code search, thereby allowing the processing of longer sequence lengths.

Aperiodic Correlation Function (ACF) And Criteria for Good Sequences

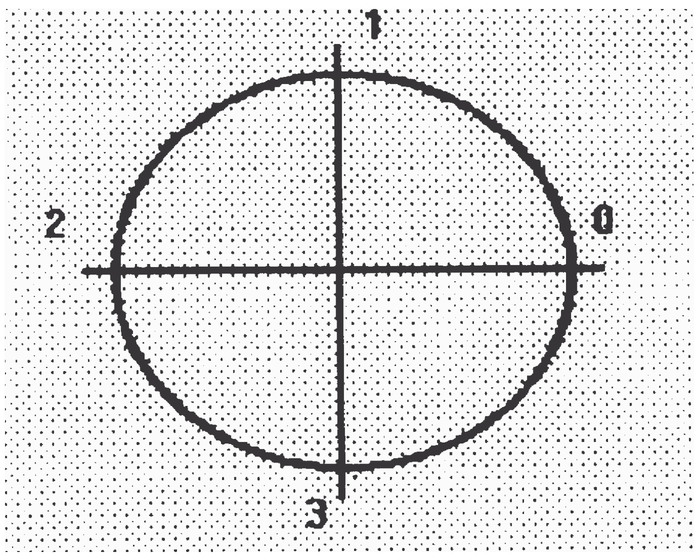
Important Formulas

The purpose of this project is to find the best quadriphase codes.

A digital quadriphase code corresponds to a sequence of *N* complex numbers defined by the set $\{1, \sqrt{-1}, -1, -\sqrt{-1}\}$. *N* is the *code length*. This code is called the *reference code*.

This base is illustrated by Figure 3.

Figure 3 Code Length



The angle of the complex number corresponds to the phase used for the transmitted carrier.

Table 1. Symbol/Phase Angle Definition

Symbol	0	1	2	3
Phase degree	0	90	180	270

A digital correlator is managed at the radar DSP level, calculating the correlation between the return code and the matched model code. The output function (ACF) is given by the following formula.

Table 2. ACF Definition

ACF:	$R(k) = \sum_{i=0}^{N-1} C_i \cdot C^*_{i+K}$	$k \in [-(N-1), N-1]$
C^* represents the complex conjugate of C		

R(0) is the main lobe. The other values correspond to the range sidelobes.

Criteria for Good Codes:

- Peak Sidelobe Level (11as to be minimum):

  Max SLL = Max(R(i))

- RMS Sidelobe Level (has to be minimum):

  $$\text{RMS} = \sqrt{\frac{1}{\text{N -1}} \sum_{1}^{N-1} \text{R}_k^2}$$

- Merit Factor

  $F = N^2 / [2(N-1)\ RMS^2]$

Calculating Method

Shift the Code:

Each sidelobe is associated with one delay. There are *N-1* sidelobes, for a given N length sequence.

In this paragraph, the symbols are manipulated under the modulo 4 complex algebra (complex roots along the unit circle)

The following example uses the quadriphase sequence {0 2 1 3 3 2 }. Five different delays correspond to the five sidelobes levels for a code length of 6.

Table 3. Delay Reference Codes

Reference code	**0**	**3**	**1**	**3**	**2**	**2**
1st delay		0	3	1	3	2
2nd delay			0	3	1	3
3rd delay				0	3	1
4th delay					0	3
5th delay						0

Calculating the Phase Difference

The product of two complex conjugate symbols is implemented by the phase difference (modulo 4 operator).

Take for example the third sidelobe:

Table 4. Third Sidelobe

Reference code	**0**	**3**	**1**	**3**	**2**	**2**
3rd delay				0	3	1
Phase Difference P (Symbol)				3	3	1
Complex Product				-i	-i	i

Note: Where $i = \sqrt{-1}$

Summing All Complex Products (Phase Differences) For One Sidelobe (One Delay)

Each sidelobe is the magnitude of the sum of complex numbers P.

3rd sidelobe' 5 level = abs ((-i)+(-i)+i) = 1

Sum = -1

Lobe value = 1

Calculation of the Other Sidelobes:

Table 5. First Sidelobe

Reference code	**0**	**3**	**1**	**3**	**2**	**2**
1^{st} delay		0	3	1	3	2
Complex product		-I	-1	-1	-i	1

Note: Lobe value: $\sqrt{5}$

Table 6. Second Sidelobe

Reference code	**0**	**3**	**1**	**3**	**2**	**2**
2nd delay			0	3	1	3
Complex product			i	1	i	-1

Note: Lobe value: $\sqrt{2}$

Table 7. Fourth Sidelobe

Reference code	**0**	**3**	**1**	**3**	**2**	**2**
4th delay					0	3
Complex product					-1	-i

Note: Lobe value: $\sqrt{2}$

Table 8. Fifth Sidelobe

Reference code	0	3	1	3	2	2
5^{th} delay				0	3	1
Complex product						-1

Note: Lobe value: $\sqrt{1}$

$$\text{RMS} = \sqrt{[(5+2+1+2+1)/5]} = 1.4832$$

A good code can be transformed into $4B^2$ (64 for the quaternary base) "brother" codes, which have the same ACF in magnitude. The transformations are:

- Time reversal
- Complex conjugate
- In-place rotation of symbols
- Addition of a phase ramp (slope 1,2, or 3 per delay)

Algorithms

Algorithms are described as pseudo-code. Programs in C language are available on diskette.

Main Part

```
Main()
begin
     init_complex_multiply_table  /* step 1*/
     init_X_code                  /* step 2*/
     while (X_code[l]=0) do       /* condition 1*/
     begin
          next_code()
          correlation_function ()
     end
end
```

This is the main part of the program. Principal steps are explained hereafter:

Condition 1 means that the scanning search is stopped when a code beginning with a couple {01} is reached. The algorithm guarantees that at least one code of the equivalent group (“brother” codes) is tested.

Init_complex_multiply_table   (Step 1)

```
Init_complex_multiply_table
     Real[0] = 0
     Real[l] = -l
     Real[2] = 0
     Real[3] = 1
     Real[4] = 0
     Real[5] = -l
     Real[6] = 0
     Real[7] = 0

     Imag[0] = 0
     Imag[1] = 1
     Imag[2] = 0
     Imag[3] = -l
     Imag[4] = 0
     Imag[5] = 1
     Imag[6] = 0
     Imag[7] = -1
```

Phase differences between symbols are in the range { -3, -2, -1, 0, +1, +2, +3} shifted (add 4) into the set { 0, 1, 2, 3, 4, 5, 6, 7}. Arrays Real[ ] and Imag[ ] are defined directly at this step (see Step 1).

Init_X_code() (Step 2)

```
Init_X_code
int loop
begin
     for loop = 0 to loop = longC-l do
     begin
          X_code[loop] = 0
          loop++
     end
end
```

This part initializes the starting code with {0000…0}.

Next_code (Step 3)

```
Next_code()
int compt

begin
     for (compt = longC-1) to (compt = 0) do
     begin
          if(X_code[compt] ≠ 3
          then
          begin
               X_code[compt]++
               breaks
          end
          else
          begin
               X_code[compt] = 0
          end
          compt--
     end
end
```

This routine implements the modulo 4 incrementation. Quaternary symbols are {0, 1, 2, 3). As a result, 3 plus 1 gives 0 and a carry to the left rank.

Correlation_function

The correlation function is the most important part of the program. It corresponds to the test of each code.

```
Correlation function()

$1
array lobe[2] [longC];
int phase_difference;
int rms =0;
int val_lobe;

lobe[0][0] = 0;
lobe[1][0] = 0;

for (dec = 1) to (dec = longC-1) do
     $11
     lobe[0][dec] =0
     lobe[1][dec] =0


     for (compt = dec) to (compt =longC-l) do
          $111
          phase_difference = 4 + X_code[compt -
dec] - X_code[compt]

          lobe[0][dec] = lobe[0][dec] +
Re[phase_difference]
          lobe[1][dec] = lobe[0][dec] +
Im[phase_difference]

         compt++
         111$


     val_lobe = lobe[0][dec]^2 + lobe[l][dec]^2


     if(val_lobe > max_lobe)
          $112
          sortie = 1
          break
          112$
     rms = rms + val_lobe

     if((sortie = 0) & (rms < max_rms))
          $113
          send start word to DSP
          for (compt = 0) to (compt = longC-l) do
               $1131
```

```
                  send X_code[compt] to the DSP
                  send X_code[compt+l) to the DSP
                  send X_code[compt+2] to the DSP
                  send X_code[compt+3) to the DSP
                  send rms to the DSP
                  send stop word to the DSP
                  1131$
            compt++
            113$

      dec++
      11$
1$
```

First Part: Lobe Value Process:

```
for (dec = 1) to (dec = longC -1) do
     $11
     lobe[0][dec] = 0
     lobe[1][dec] = 0

     for (compt = dec) to (compt = longC-1) do
          $111
          phase_difference = 4 + X_code[compt -
dec] – X_code[compt]

          lobe[0][dec] = lobe[0][dec] + Re
[phase_difference]
          lobe[1][dec] = lobe[0][dec] + Im
[phase_difference]

          compt++

          111$

     val_lobe = lobe[0][dec]^2 + lobe[1][dec]^2
```

This part of the program corresponds to the calculation part of this report.

First of all, lobe [0] [dec] and lobe [1] [dec] are initialized. They correspond to real part and imaginary part of the calculated sidelobe.

The *phase_difference* variable corresponds to the difference in phase between the original code and the translated code. As explained earlier, 4 is added to each *phase_difference* value so that the phase difference is positive.

All the *phase_difference* values are added to obtain the sidelobe value.

Second Part: RMS Value Process:

```
if(val_lobe > max_lobe)
     $112
     sortie = 1
     break
     112$
rms = rms + val_lobe
```

To stop the test as early as possible, each sidelobe value is compared with a maximum sidelobe. In this program, the sidelobe maximum length is 4. If the sidelobe is larger than this maximum sidelobe value, the code is considered an unusable code and is discarded. The program then exits this part (sortie =1) and tests the next code.

If the sidelobe is small enough, it is added to the rms value.

Third Part: Correct Codes Sent

```
if((sortie = 0) & (rms < max_rms))
     $113
     send start word to DSP
     for (compt =0) to (compt = longC-1) do
          $1131
          send X_code[compt] to the DSP
          send X_code[compt+1] to the DSP
          send X_code[compt+2] to the DSP
          send X_code[compt+3] to the DSP
          send rms to the DSP
          send stop word to the DSP
          1131$
     compt++
     113$
```

If the code has no large sidelobe and the rms value is smaller than the maximum, the code is sent to the DSP. The maximum RMS value is 2.

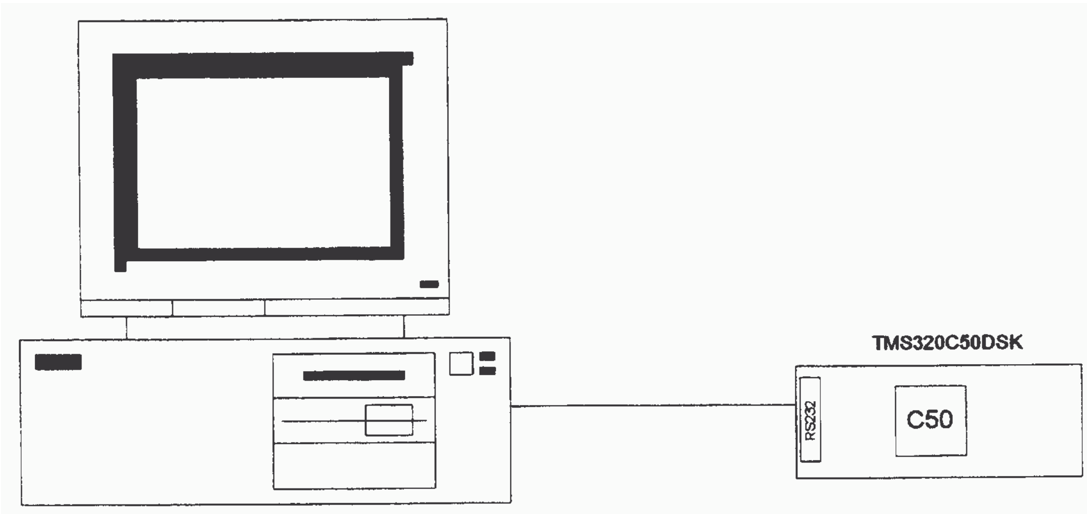
rms is not exactly the RMS value defined in this part, because the DSP microprocessor only works on integer numbers and not on real numbers.

That means that it cannot calculate $\sqrt{\frac{1}{\mathrm{N}-1}\sum_{1}^{N-1}\mathrm{R}_k^2}$. Therefore, rms is defined as $\sum_{1}^{N-1}\mathrm{R}_k^2$ only. The DSP processor sends the code to the computer, which calculates the real RMS value.

PC Communication

Hardware Consideration:

Figure 4. Hardware Diagram



To allow more time for the DSP to select good codes, we used a standard PC to store the codes found by the C50 and to calculate their RMS value. Unfortunately, the PC was waiting most of the time for the DSP to send it a good code.

To overcome this problem, we were forced to consider the architecture of the TMS320C50 DSK. From the schematic diagram, it is easily understood that we would have to use the standard PC RS232 link and the TDM serial port to transfer the good codes from the TI TMS320C50 DSK to the standard PC. That means that we would have to configure the TDM to be able to communicate properly with the PC. This would involve handling the interrupt protocol between the DSP and the PC so that the PC could signal the DSP that the information had been received.

Communication Protocol:

The first step is to download the program into the DSP memory. The DSK debugger is used to download the program. Then the PC program is started, which wakes up the DSP. The next step is to wait for the completion of the entire research. During this phase, the DSP transmits the good codes to the PC each time a good code is encountered by the DSP. The PC responds with the interrupt protocol (interrupt 2) and then calculates the RMS and sorts the new good code into the result table.

Software Performances

Characteristics

Maximum Length for the Search of Quadriphase Codes:

The maximum code length processed by this program is 40. This value is large enough when considering the time duration of the search (more than one year).

Codes Saving:

To avoid loosing information due to an interruption of main power, or any other breakdown, the program periodically saves the codes, via the RS232 serial link, by sending them to the PC.

400 Best Codes:

The software stores the 400 best codes found. When a better code is discovered, the 400th code is discarded.

Speed

This project, utilizing the TI TMS320C50 DSK for the search, is compared with a similar compiled program. The comparison comes from an extrapolation with a sequence length of 25.

Ranking	Processor	Time Duration for search
1st	TIDSKTMS320C5O	1year
2nd	Pentium75	12 years
3rd	486 DX2 66	25 years

In conclusion, the TI ’C50 DSP processor is more than 10 times faster than a standard PC.

Summary

This challenge was of great interest to us. First of all, we learned a lot about radar fundamentals and Digital Pulse Compression. Moreover, it was an interesting experience of teamwork.

The Texas Instruments DSP Solutions Challenge gave us the opportunity to discover TI DSPs using one of the low cost development boards, the TMS320C50 DSP Starter Kit. The TI development software (Assembler, Linker, Debugger, Simulator) was convenient to use.

This challenge has shown us the benefits of coupling a Texas Instruments DSP with a standard PC. The TI DSP extends the power and flexibility of the system beyond the capabilities of a standard PC.

This project has been an enlightening experience for all of us.