# *Determining CPU and Memory Requirements for Real-Time Speech Recognition Systems Using the TMS320C3x/C4x*

**Authors: E. Batlle, J.A.R. Fonollosa**

**ESIEE, Paris**

*September 1996*
*SPRA314*

TEXAS INSTRUMENTS

## IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

## TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

**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

# Determining CPU and Memory Requirements for Real-Time Speech Recognition Systems Using the TMS320C3x/C4x

## Abstract

Developing a computer system using real-time speech recognition previously required a workstation using non-specialized CPUs. Limits to the system were imposed by the amount of memory and hardware required.

The Texas Instruments (TI™) TMS320C3x/C4x (’C3x/’C4x) digital signal processor (DSP) is a high performance CMOS 32-bit floating point processor. This project uses the (’C3x/’C4x) DSP to count the cycles consumed by the algorithms involved and to know how many words can be recognized with a predetermined accuracy, allowing you to calculate the hardware required for your recognition system to perform.

This document describes the CPU and memory requirements for DSP real-time speech recognition systems used in consumer applications. Included are sections on signal preprocessing, parameter measurement, vector quantization, veterbi algorithm, and results.

This work was supported by the Spanish government under grant number C95-l022-C05-03. This document was part of the first European DSP Education and Research Conference that took place September 26 and 27, 1996 in Paris. For information on how TI encourages students from around the world to find innovative ways to use DSPs, see TI’s World Wide Web site at www.ti.com.

# Product Support

## Related Documentation

The following list specifies product names, part numbers, and literature numbers of corresponding TI documentation.

- *TMS320C3x User's Guide*, Literature number SPRA031E
- *TMS320C3x C Source Debugger*, Literature number SPRU053B

## 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

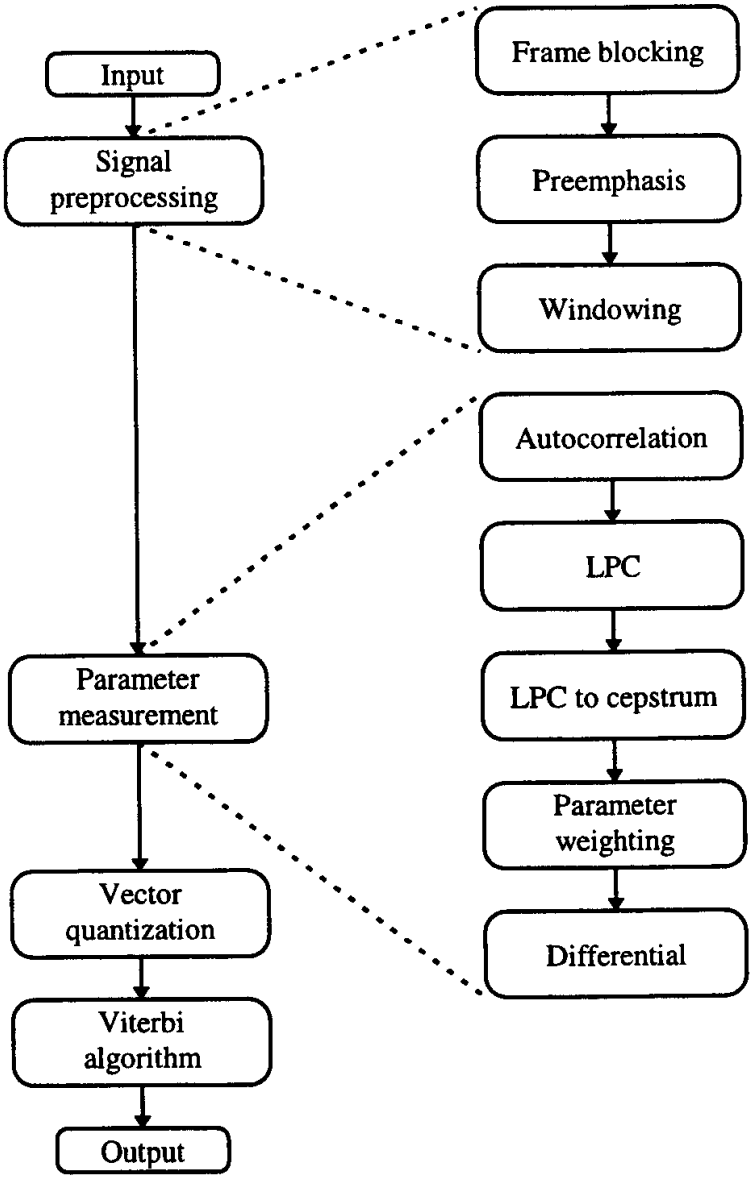
For the past decades, many researchers worked to develop algorithms for speech recognition systems. All these algorithms worked only on workstations and belonged to experimental systems. However, speech recognition systems have spread to consumer applications in recent years, which has led to the development of smaller (and cheaper) systems. In addition, researchers started using DSPs to implement speech recognition systems. One of the most popular DSP families is the TI TMS320. We have developed a system based on the TMS320C3x, a high-performance CMOS 32-bit floating-point DSP.

One point that is very important in the development of a system is to know the limits imposed by the hardware. What we basically do in this paper is use the TMS320C3x/C4x DSP to count the cycles consumed by the algorithms involved in recognition. We thus can know how many words we can recognize with a predetermined accuracy, how much memory we must use, etc.

## Signal Preprocessing

It is useful to preprocess the speech signal to flatten its spectrum and make the system more robust against finite precision effects.

*Figure 1. A Speech Recognition System*



## Frame Blocking

Before entering the recognition system, the input signal is blocked into frames of *FR* samples with an overlap of *WS* samples (Table 1 lists the variables and their meaning). We use *FR*+1 words to store the entire block and its previous sample.

*Table 1. Notation*

| Variable | Meaning |
|---|---|
| FR | frame-length |
| LP | LPC order |
| LC | cepstrum coefs. |
| DI | diff. memory |
| FS | sampling rate |
| WS | window shift |
| CS | size of codebooks |
| VD | vector dimension |
| NS | number of states |
| NO | number of obs. |
| NM | n. of models |
| NT | transitions |

## Preemphasis

To remove the color of the spectrum of the speech signal we apply a first-order high-pass filter.

As we can see in the code below, there is a pipeline conflict and we consume two cycles for each sample.

```
 rptb  end_preem
 subf  r0, *ar0--(1), r1
end_preem:
 stf   r1, *+ar0(1)
 || mpyf r2, *-ar0(1) , r0
```

Where *ar0* points to the last input sample and *r2* is the preemphasis factor.

## Windowing

Signal discontinuities are at the beginning and end of each frame block. We can soften them by windowing each frame. The most used window in speech recognition is the Hamming window. As in the previous section, a pipeline conflict forces us to consume two cycles per sample. The program is

```
rpts   FR-2
mpyf   *ar0--(1), *ar1--(1),  r0
|| stf r0, *+ar0(1)
```

As we make three memory accesses at the same time, a bus conflict occurs and an additional cycle is used.

## Preemphasis and Windowing

As we show in the previous sections, *Preemphasis* and *Windowing*, we can not complete any of the parts without an additional cycle. However, we can perform these two steps jointly to avoid one of the pipeline conflicts.

```
 rptb  end_pre_wind
 subf  r0, *ar0--(1), r2
 mpyf  r2, *ar1--(1), r1
end_pre_wind:
 stf   r1, *+ar0(1)
 || mpyf r3, *-ar0(1), r0
```

With this program we use just $3 \cdot FR$ cycles to get the entire block preemphasised and windowed.

## Parameter Measurement

Once we have preprocessed the speech signal, we can extract the features that the system will use to recognize the words. As is well known, there are many useful parameters that we can consider (mel-cepstrum, LPC, etc.). In our system we use LPC cepstrum parameters because they are not very expensive to calculate and give us an acceptable accuracy.

First of all, we want to know which parts are the most expensive. Previously, we programmed the entire system in C and we used the TI compiler and optimizer. After that, we count how many cycles consume each part. We take common values for a speech recognition system with a sampling rate of 8 kHz, i.e.,

FR=240, LP=8, LC=12, DI=2, FS=8000, WS=120, CS=(128,128,64), VD=(12,12,1), NS=10, NO=3, NM=11, NT=2

Table 2 lists the cycles consumed by each part in the parameter measurement.

*Table 2. Cycles Consumed by the Parameterization*

| Algorithm | Cycles |
|---|---|
| Autocorrelation | 2347 |
| LPC analysis | 959 |
| LPC to cepstrum | 1254 |
| Parameter weighting | 70 |
| Temporal derivative | 403 |

## Autocorrelation Analysis

Table 2 reflects that the autocorrelation is the most expensive algorithm of those involved in the feature extraction. The autocorrelation algorithm is

$$r(i) = \sum_{j=0}^{FR-i=1} \hat{s}(j)\hat{s}(i+j) \quad 0 \le i \le LP \tag{1}$$

We notice that it needs on the order of $LP^2$ cycles to execute.

We programmed the preliminary version of the algorithm in C and used the TI compiler and optimizer. We observed the following assembler code produced by the optimizer:

```
rpts  r1
addf  r0, r2
|| mpyf *ar2++, *ar4++, r0
```

This means that we would not improve the performance of the program by programming it directly in assembler code, because the body of the main loop is already executed with *one-cycle* instructions.

## LPC Analysis

At this step, we use the Levinson-Durbin algorithm to obtain the LPC parameters from the autocorrelation matrix. This algorithm converts the LP+1 autocorrelation coefficients into the LPC parameters using the reflection (PARCOR) coefficients.

$$
\begin{aligned}
&a_{0.0} = 1 \\
&P_0 = r(0) \\
&\Gamma_m = -\frac{1}{P_{m-1}} \sum_{i-0}^{m-1} r(i-m) a_{m-1,i} \\
&a_{m,0} = 1 \\
&a_{m,i} = a_{m-1,i} + \Gamma_m a^{*}_{m\text{-}1,m\text{-}i} \quad 1 \le i < m \\
&a_{m,m} = \Gamma_m \\
&P_m = P_{m-1}\left(1 - |\Gamma_m|^2\right)
\end{aligned}
\qquad (2)
$$

We can improve some of the parts of the output of the TI optimizer, but this will reduce the cost in cycles of the algorithm only a little, because it represents only 1.6 % of the total.

## LPC to Cepstrum

We use the obtained LPC parameters to derive the cepstral parameters. In speech recognition systems the cepstral domain is preferred because convolutional noise turns into additive noise, and because it is easier to separate the vocal tract and the glottal excitation from the received signal.

The algorithm used is

$$
\begin{aligned}
&c_0 = 0 \\
&c_m = a_m + \frac{1}{m} \sum_{k=1}^{m-1} k c_k a_{m\text{-}k} \quad 1 \le m \le LP \\
&c_m = \frac{1}{m} \sum_{k=1}^{m-1} k c_k a_{m\text{-}k} \quad m > LP
\end{aligned}
\qquad (3)
$$

Here the optimizer is not able to solve a pipeline conflict and gives as an output the following code lines.

```
 rptb  ceps1
 float r2, r0
 mpyf  *ar2--, r0
 mpyf  *ar5++, r0
 addf r0, r4
ceps1:
 subi  1, r2
```

When the algorithm represents a small part within the global system (2 %) we can reduce only slightly its cost by programming it in assembler.

## Temporal Derivative

It is well known that we can improve the performance of a speech recognition system by adding temporal derivative information. In our system we approximate the differential by

$$\frac{\partial c_m(t)}{\partial t} = \Delta c_m(t) = \mu \sum_{k=-DI}^{DI} k c_m(t+k)) \qquad (4)$$

Where we can take $\mu = 1$ for simplicity. According to Table 2, this section represents 0.6 % of the global.

## Vector Quantization

In previous sections we obtained all the parameters that we need to recognize the speech signal, but to store all this data we consume a lot of memory. We can reduce the amount of memory required by storing just one value instead of a LC-dimensional vector. To find this value we use a vector quantifier. In our system we store just one value per vector. Doing so, the resulting Markov Models are called discrete (DHMM). Even though we made all the computations for discrete models, is not very complicated to extend them to semi-continuous.

Because the VQ is the most CPU-consuming block in our system, it is important to pay special attention to it.

## Full Search Algorithm

This is the most computationally expensive process but always finds the best match. The algorithm is

$$m^* = \arg \min_{1 \le m \le CS} d(v, y_m) \tag{5}$$

Where $v$ is the feature vector and $y_m$ is the $m$th vector in a CS-vector codebook.

The distance used is $L_1$, that is

$$d(x, y) = \sum_{k=1}^{k} |x_k - y_k| \tag{6}$$

We may use the square of the distance between components (instead of the absolute value).

$$d(x, y) = \sum_{k=1}^{k} (x_k - y_k)^2 \tag{7}$$

With this distance and the corresponding values as previously shown in this section, we obtained the number of cycles listed in Table 3.

*Table 3. Full Search VQ*

| VQ Parameter | Cycles | Percentage (%) |
|---|---|---|
| LPC cepstrum | 17974 | 31.2 |
| LPC ceps. deriv. | 17974 | 31.2 |
| energy deriv. | 1974 | 3.4 |
| TOTAL | 37922 | 65.8 |

## Pre-Computed Energy

The results from Table 3 are not good. We can reduce the cycles used by the VQ by pre-calculating some information. If we decompose the distance used, we can write

$$d(x,y)=\sum_{k=1}^{K}x_k^2+\sum_{k=1}^{K}y_k^2-\sum_{k=1}^{K}2x_k y_k \qquad (8)$$

$\sum_{k-1}^{K}x_k^2$ will always be the same if we do not change the codebook and we can calculate it previously. Doing so this calculus will not consume any cycle during recognition.

Since we use *d*(**x**, **y**) to compare distances we do not need to know its exact value and since $\sum_{k-1}^{K}x_k^2$ is the same for each codebook we do not have to calculate it. From what we said, during recognition we only calculate $\sum_{k=1}^{K}2x_k y_k$, and this can be done with just **K** cycles.

Table 4 shows the cycles used by this algorithm using the pre-calculated values. We wrote the code entirely in C and used the TI optimizer.

*Table 4. Cycles Used by Pre-Calculating the Energy*

| VQ Parameter | Cycles | Percentage (%) |
|---|---|---|
| LPC cepstrum | 16436 | 30.1 |
| LPC ceps deriv | 16436 | 30.1 |
| Energy deriv | 1908 | 3.5 |
| Total | 34780 | 63.7 |

The results in Table 4 are close to those in Table 3 and still poor.

If we analyze the code to compute $\sum_{k=1}^{K}x_k y_k$ produced by the optimizer we see

```
 1di   0, r3
L95:
 mpyf  *ar5++, *ar2++, r0
 subf  r0, r2
 addi  1,  r3
 cmpi  rs, r3
 blo  L95
```

This code consumes more than **K** cycles. We can improve this part by writing it in assembler.

## Assembler Version

In the Pre-Computed Energy section, we calculated the theoretical number of cycles needed by the VQ algorithm and saw that a C implementation of the code does not reach this limit. Now we can try with an assembler version of the code that calculates $\sum_{k=1}^{K} x_k y_k$

```
rpts  r7
mpyf  *ar5++, *ar2++, r0
|| subf r0, r2
```

We can read the cycles consumed by the assembler program in Table 5.

*Table 5. Cycles Used by the Assembler VQ*

| VQ Parameter | Cycles | Percentage (%) |
|---|---|---|
| LPC cepstrum | 5211 | 16.6 |
| LPC ceps. deriv. | 5211 | 16.6 |
| energy deriv. | 1216 | 3 9 |
| TOTAL | 11638 | 37.1 |

This supposes a great reduction of the cycles used even if they do not reach the theoretical limit. This is due to the use of external memory instead of on-chip RAM. Our calculations are for on-chip memory, which means that we can do two memory accesses within a single cycle. Because of the size of the codebooks, we can not store them in the on-chip memory of the TMS320C3x. We need $\sum_{i=1}^{NO} CS_i VD_i$ words to store all the codebooks and we have only 2 Kwords of on-chip RAM. Even if the vector of cepstral parameters is stored in this fast RAM, the codebooks are in the external RAM. That leads us to extra cycles.

## The Viterbi Algorithm

The Viterbi algorithm is the last step in our system. This algorithm determines which word is most likely to be said by the speaker.

Even if it is not the most expensive, it is the most complicated to program. This is the reason why is hard to calculate the number of theoretical cycles consumed by the algorithm. Because of that, we program it in C and use the TI compiler and optimizer. That gives us an approximated number of cycles. In the *Results* section, we can find the cycles consumed by the program for several values of the parameters.

To store the models we need to allocate $NS \cdot NM \cdot \left( NT + \sum_{i=1}^{NC} CS_i \right)$ words in memory.

# Results

In the previous sections we calculated the cycles consumed by the speech recognition system given a determinate set of values of the variables.

Now we can experiment to see how the system is affected by a change in the size of one parameter, i.e., change the order of the LPC parameters, add codewords, etc.

Our first experiment is done with common values. We can take the same values as in the *Parameter Measurement* section.

Table 6 lists the cycles consumed by the first system. We will take these values as a reference to compare with the next experiments.

*Table 6. Common Values*

| Section | Cycles | Percentage (%) |
|---|---|---|
| Preemphasis & windowing | 760 | 2.4 |
| Autocorrelation | 2347 | 7.6 |
| LPC analysis | 959 | 3.1 |
| LPC to cepstrum | 1254 | 4.0 |
| Parameter weighting | 70 | 0.2 |
| Temporal derivative | 403 | 1.3 |
| Vector quantifier | 11638 | 37.1 |
| Viterbi algorithm | 13884 | 44.3 |

As we notice, these results are different from those we expected. If we calculate the theoretical ones we will find a notable difference.[6] This is a result of accessing external memory.

A speech recognition system needs a lot of memory to store codebooks, hidden Markov models, etc. The ideal situation is to store all this information in on-chip RAM, but the TMS320C3x has only 2 Kwords of this type, so we must put all the information in external RAM. All these facts slow our system down. However, it is interesting to know both the number of operations involved in the algorithm (independently of the memory access) and the hardware-dependent implementation.

With this in mind, we can try another experiment to test how the system is affected by a change in the size of one of the codebooks. If we take CS=(256,128,64) we obtain the results shown in Table 7.

*Table 7. Cycles for a Bigger Codebook*

| Section | Cycles | Percentage (%) |
|---|---|---|
| Preemphasis & windowing | 760 | 2.1 |
| Autocorrelation | 2347 | 6.4 |
| LPC analysis | 959 | 2.6 |
| LPC to cepstrum | 1254 | 3.5 |
| Parameter weighting | 70 | 0.2 |
| Temporal derivative | 403 | 1.1 |
| Vector quantifier | 16758 | 46.0 |
| Viterbi algorithm | 13884 | 38.1 |

As we expected, the only part affected is the VQ, which increases the cycles used and also the percentage of the total.

In our following experiment we wish to know the effect of using more words in the recognizer. If we take an entire Markov model for each word, and we make NM=22, we notice that the Viterbi algorithm consumes 27700 cycles, instead of 13884 for NM=11.

In our final experiment we see what happens when we use more cepstral coefficients, i. e., LP=15 and LC=20. Again the most affected part is the VQ, as we see in Table 8, even though the entire system is affected.

*Table 8. Cycles for a Longer Cepstral Vector*

| Section | Cycles | Percentage (%) |
|---|---|---|
| Preemphasis & windowing | 760 | 2.1.9 |
| Autocorrelation | 4090 | 10.3 |
| LPC analysis | 1960 | 4.9 |
| LPC to cepstrum | 2455 | 6.2 |
| Parameter weighting | 94 | 0.2 |
| Temporal derivative | 627 | 1.6 |
| Vector quantifier | 15773 | 39.9 |
| Viterbi algorithm | 13884 | 35.0 |

# References

[1] G. Brassard, et al., "Algorithmique. Conception et analyse", *Masson* 1987

[2] E. Horowitz, et al., *Fundamentals of Computer Algorithms*, Computer Science Press 1989

[3] L. Rabiner, et al., *Fundamentals of Speech Recognition*, Prentice Hall 1993.

[4] *TMS320C3x User's Guide*, Digital Signal Processing Products, Texas Instruments, 1992

[5] "TMS32OC3x C Source Debugger", *Microprocessor Development Systems*, Texas Instruments 1991

[6] E. Batlle, et. al., "Computational Cost and Memory Requirements for Real-Time Speech Recognition Systems", to be published in *JCSPAT* 1996.