Discrete Fourier Transform algorithm implemented on dsPIC33 µController

Microprocessor Architectures [ELEC-H-473]

Team Members : LIU Yu, YOU Jie, QIN Hao, Mroueh Michael
March 2016
https://github.com/Walkerlikesfish/dspIC33.git

Introduction

Objective

The goal of this project is to implement a Discrete Fourier Transform on a dsPIC33 Microcontroller.

$$X(k) = \sum_{n=0}^{N-1} x_n \,.\, e^{-2i\pi k.n/N}, \qquad k \in \mathbb{N}$$

A particular focus is made on the optimization of the algorithm: computing power and memory allocation efficiency as well as accuracy of final results.

Constraints

- **Number of samples:** *N = 128*: the samples x_n are stored in a circular buffer
- **Inputs:** The samples are 8bit signed integers
- **Expression of the transform core:** The exponentials $e^{-2i\pi.m}$ are stored as fixed point complex numbers and $m = round(k.\frac{n}{N})$ The output samples are complex numbers, with a fixed point representation
- All computation must be performed using fixed point arithmetic
- **Outputs:** The output vector X_k is complex and of size N, with proper precision.

The Simple Algorithm

Basic idea

To calculate the DFT of a given length N(=128), the definition of Fourier Transform is applied here.

$$X_k = \sum_{n=0}^{N-1} x_n \,.\, En\left[k * \frac{n}{N}\right] = \sum_{n=0}^{N-1} x_n * [\cos(m) - jsin(m)]$$

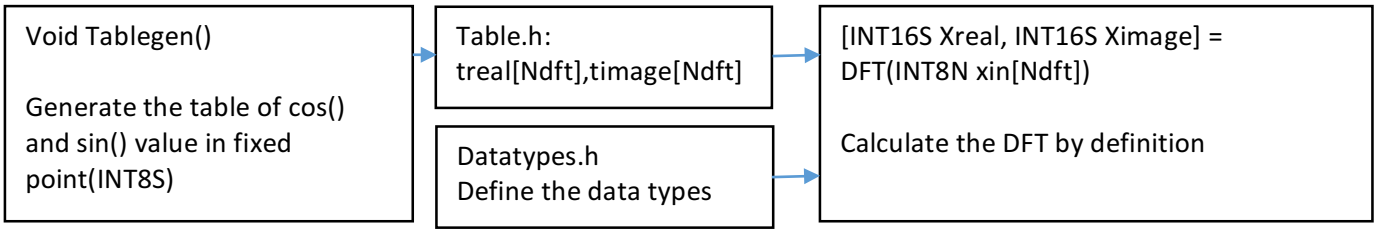
$$X_k^R = \sum_{n=0}^{N-1} x_n * \cos(m)\,; \qquad X_k^I = -\sum_{n=0}^{N-1} x_n * \sin(m);$$

$$k \in [0, \dots, N-1], \qquad m = round[k * \frac{n}{N}]$$

As it is indicated in the constrains, the m is rounded to the nearest integer. If $n \epsilon [0; N-1]$, we could find $m \in [0; 127]$. $X_k^R \; and \; X_k^I$ stands for the real and image part of the coeffitients.
In general, the algorithm applied with fixed point operation is divided into two parts:
1) Prepare the coefficient table for the DFT (offline), say, generate the cos(m) and sin(m) in the equation above; 2) Carry out DFT with only fixed point operation (onboard)
Details of the implemented algorithm will be explained in the assignment.

Assignment

Question 1. ***Split the computation into smaller operations and draw a graph of you computation flow, name the intermediate results.***



Question 2. ***Explain the size and type of the different intermediate results.***

DFT Part

Variable name	Type & size	Explaination
treal	INT8S[] Range:[-128, 127] Size: Ndft	Store the cos(m) coeffients, used to calculate real part txr = (INT16S)treal*(INT16S)xin[j]; //typecast before
timage	INT8S[] Range:[-128, 127] Size: Ndft	Store the sin(m) coeffients, used to calculate the image part txi = -(INT16S)timage*(INT16S)xin[j]; //typecast before
Txr	INT16S [-32768, 32767]	Used to store the current result x(n) *cos(k). (INT8U * INT8S)
txi	INT16S [-32768, 32767]	Used to store the current result x(n)*sin(k). (INT8U * INT8S)
sxr	INT32S	Used to store the sum of real result txr in loop. Sum of INT16S
sxi	INT32S	Used to store the sum of image result txi in the loop. Sum of INT16S
Xreal	INT16S Size: Ndft	The real part of result, get from sxr>>6, where sxr is INT32S, but at most it would be 0x0100. So right shift 6 bits result a 16bits result without overflow risk.
Ximage	INT16S Size: Ndft	Same as Xreal

Table generating Part

Variable name	Type & size	Explaination
m	INT8U: [0, 255]	m = n*k/Ndft, in $e^{-2i\pi.m}$, rounded to integer
tm	INT16U: [0, 10e4]	tm =(2*50*m+128)>>4, 50 = (INT8U) Pi(3.14159)*16
treal	INT8S[] Range:[-128, 127] Size: Ndft	To store the cos coeffients *(treal+i)=(INT8S)(tcos*64);
timage	INT8S[] Range:[-128, 127] Size: Ndft	To store the sin coeffients *(timage+i)=(INT8S)(tsin*64);
Tcos	FP32	To store the result returned from cos() *
Tsin	FP32	To store the result returned from sin()

INT8U: integer 8 bits Unsigned; INT16S: integer 16bits Signed

*1. Here worth noticing, we use two **float point variable** tcos, tsin. we have to mention that because the **table computation** is carried out **offline**, the computation cost both in time and memory should not be restricted too hard, so we use these two variables to store the result returned from function cos(), sin(). However, we could adapt it by writing cos() function on our own, where we could use **series expansion** to calculate an approximation of cos(m), and return a **INT8S**, to save memory and speed.

The decision to use Fixed Point Representation instead of Floating Point Representation is a regular optimization trade-off allowing to save some Data Memory and Computation Power, to the detriment of Precision and Development Time, which is totally relevant in our project.

Question 3. For each operation, justify that no overflow will occur and compute the accuracy.

As it is illustrated, the program finish the DFT, mainly by two parts. First, we calculate the core coefficients, say, the cos(m) and sin(m), and put them into the data memory. Then, we calculate the DFT by its definition and store the result DFT coeffients in the array.

DFT

KEY OPERATION	Analyze
M=I*J/NDFT;	m,I,j,NDft:INT8U, rounded to the nearest number. m:[0, 127], precision: **2^0**, no overflow
TXR = (INT16S)TREAL*(INT16S)XIN[J];	Txr is defined to INT32S, the max is 0x0100, with sum of two INT16S, precision is decided by treal: **2^-6**
SXR = SXR+TXR;	Sum up Txr to get the X(k), Txr is defined INT32S with max 0x0100, so Sxr is defined INT32S, would not overflow. precision: **2^-6**
*(TR+I)=(SXR+16384)>>6;	Round Sxr to the nearest integer. Precision **2^0**

GENERATING TABLE

KEY OPERATION	Analyze
M = I*J/NDFT;	m,I,j,NDft:INT8U, rounded to the nearest number. m:[0, 127], precision: **2^0**
tm=(2*50*m+128)>>4;	50=3.1415*16. Precision: **2^-4**
tcos = cos(double(tm));	Cosine here is calculated by <math.h> function, precision: **2^-126**
*(treal+i)=(signed char)(tcos*64);	Tcos small or equal 1, treal is defined to INT8S, Fix point to **2^(-6)**, no overflow

Question 4. *Assuming the circular buffer filling does not affect your computation, what is the maximum possible frequency for the input sampling? Use the figures you got from basic operations in the labs.*

The input sampling frequency (fsample) should smaller than the speed of the DFT process. Say if the total time for the DFT program process is Tprog, fprog=1/Tprog. fsample<fprog. Because, if the input sampling is too fast, the DFT haven't finished before the input is changed. So the problem comes to the total time of the process (online part exclude the table building part).

The DFT part is constructed by two loops, with Ndft^2 iterations. Each iteration consists: 2x (INT16S)*(INT16S); 2x (INT16S)+(INT16S); 1x(INT8U)*(INT8U); 1x(INT8U)/(INT8U); 1x(INT8U)+(INT8U); 1x(INT8U)mod;

Total **95 instruction cycles**, consuming **1.58 uSecs**, T=1.58e-6 => Tprog = T*Ndft^2 = 0.02 sec; fsample < **50Hz.**