DIGITAL SIGNAL PROCESSORSING LAB
LAB REPORT – 10

NAME: YETURI VENKATESH
ROLL No.: 224102324

CONVOLUTION AND CORRELATION OF TWO SIGNALS

INTRODUCTION:

Real-time digital signal processing applications can be developed on the chip with the Texas Instruments TMS320C6713 and TMS320C6416 Digital Signal Processing Starter Kits.

Digital signal processor : --- 1. Floating point , 2. Fixed point

Each device comprises a compact circuit board with a TMS320C6713 floating-point digital signal processor or a TMS320C6416 fixed-point digital signal processor, as well as a TLV320AIC23 analogue interface circuit (codec), and it connects to a host PC via a USB port.

DSK SUPPORT TOOLS:

1. A Texas Instruments DSP starter kit (DSK). The DSK package includes:
   a) Code Composer Studio (CCS), which gives users access to the required application runtime environments. The C compiled code, assembler, linker, debugging tools, and other toolkits are all included in CCS's integrated software development environment (IDE).
   b) A circuit board (the TMS320C6713 DSK) containing a digital signal processor and a 16 – bit stereo codec for analog signal input and output.
   c) A universal synchronous bus (USB) cable that connects the DSK board to a PC.
   d) A +5 V universal power supply for the DSK board.

2. A PC: The DSK board connects to the USB port of the PC through the USB cable included with the DSK package.

3. An oscilloscope, spectrum analyser, signal generator, headphones, microphone, and speakers.

C6713 and C6416 DSK BOARDs:

The DSK boards, which measure approximately 5 × 8 inches, include either a 225 - MHz C6713 floating point digital signal processor or a 1 - GHz C6416 fixed - point digital signal processor and a 16 - bit stereo codec TLV320AIC23 (AIC23) for analog input and output.

The DSK boards each include 16 MB (megabytes) of synchronous dynamic RAM (SDRAM) and 512 kB (kilobytes) of flash memory.

Four connectors on the boards provide analog input and output: MIC IN for microphone input, LINE IN for line input, LINE OUT for line output, and HEADPHONE for a headphone output (multiplexed with line output).

The status of four user DIP switches on the DSK board can be read from within a program running on the DSP and provide the user with a feedback control interface.

The states of four LEDs on the DSK board can be controlled from within a program running on the DSP. Also onboard the DSKs are voltage regulators that provide 1.26 V for the DSP cores and 3.3 V for their memory and peripherals.

TMS320C6713 Digital Signal Processor:

The TMS320C6713 (C6713) is based on the very - long - instruction - word (VLIW) architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle.

Features of the C6713 include:

264 kB of internal memory (8 kB as L1P and L1D Cache and 256 kB as L2 memory shared between program and data space), eight functional or execution units composed of six ALUs and two multiplier units, a 32 - bit address bus to address 4 GB (gigabytes), and two sets of 32 - bit general – purpose registers.

TMS 320 C 6416 Digital Signal Processor:

The TMS320C6416 (C6416) is based on the VELOCITI advanced very - long - instruction word (VLIW) architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle.

Features of the C6416 include:

YETURI VENKATESH
224102324

1056 kB of internal memory (32 kB as L1P and L1D cache and 1024 kB as L2 memory shared between program and data space), eight functional or execution units composed of six ALUs and two multiplier units, a 32 - bit address bus to address 4 GB (gigabytes), and two sets of 32 - bit general - purpose registers.

CODE COMPOSER STUDIO:

Code Composer Studio (CCS) provides an integrated development environment (IDE) for real - time digital signal processing applications based on the C programming language. It incorporates a C compiler, an assembler, and a linker. It has graphical capabilities and supports real - time debugging.

STEPS FOLLOWED IN CODE COMPOSER STUDIO FOR LED BLINKING:

After ensuring that the hardware requirements are met, we must configure the environment (for BSP- Board Support Library and CSP- Chip Support Library) such that the kit can perform a specific function.

Here, I've implemented LED blinking and convolution and correlation operations.

The necessary setup steps in order to accomplish this are as described in the following:

1. Download BSL and CSL library.
2. Make a copy of this file in the installation directory of “ti” app.
3. Create a workspace where we will save our current working files.
4. Open Code Composer Studio (CCS) Integrated Development Environment (IDE).
5. Create a new CSS project with project name as “Convolution_Correlation”.
6. Set family device as C6000 and Variant as C641X Performance Value DSP for DSK6416.
7. Choose connection type as Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator.
8. Choose Project Templates and click Finish.
9. Create led.c file to generate convolution and correlation and LED blinking (explained below after steps).

YETURI VENKATESH
224102324

10. Perform Build Settings:
    a. Including header files of BSL and CSL in the project.
    b. Defining DSP Chip symbol under predefined symbols.
    c. Adding BSL and CSL Libraries to the Linker.
    d. Link the BSL and CSL to C6000 Linker.
11. Build and Run the project.

Led.c file to execute Convolution and correlation:

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

#define SIGNAL_LENGTH 10 // length of signals

void convolution(float signal1[], float signal2[], float result[]) {
   int i, j;
   for (i = 0; i < SIGNAL_LENGTH; i++) {
      result[i] = 0;
      for (j = 0; j < SIGNAL_LENGTH; j++) {
         if (i >= j) {
            result[i] += signal1[i-j] * signal2[j];
         }
      }
   }
}

void correlation(float signal1[], float signal2[], float result[]) {
   int i, j;
   for (i = 0; i < SIGNAL_LENGTH; i++) {
      result[i] = 0;
      for (j = 0; j < SIGNAL_LENGTH; j++) {
         if (i + j < SIGNAL_LENGTH) {
            result[i] += signal1[j] * signal2[i+j];
         }
      }
   }
}
```

YETURI VENKATESH
224102324

```
int main() {
    float signal1[SIGNAL_LENGTH] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    float signal2[SIGNAL_LENGTH] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    float result[SIGNAL_LENGTH];

    // perform convolution
    convolution(signal1, signal2, result);

    // print convolution result
    printf("Convolution Result:\n");
    for (int i = 0; i < SIGNAL_LENGTH; i++) {
        printf("%f ", result[i]);
    }
    printf("\n");

    // perform correlation
    correlation(signal1, signal2, result);

    // print correlation result
    printf("Correlation Result:\n");
    for (int i = 0; i < SIGNAL_LENGTH; i++) {
        printf("%f ", result[i]);
    }
    printf("\n");

    return 0;
}
```

YETURI VENKATESH
224102324