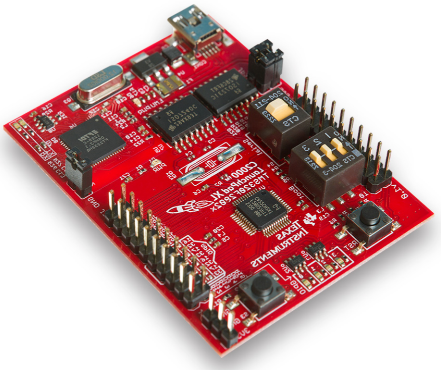
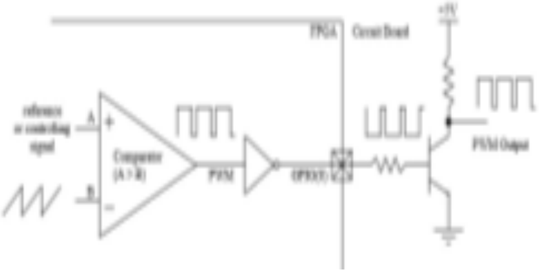
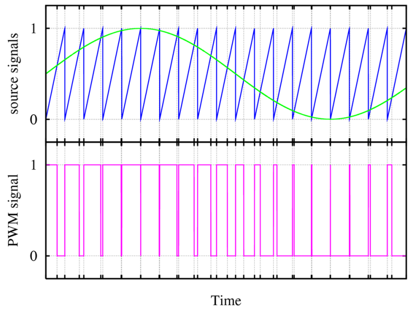
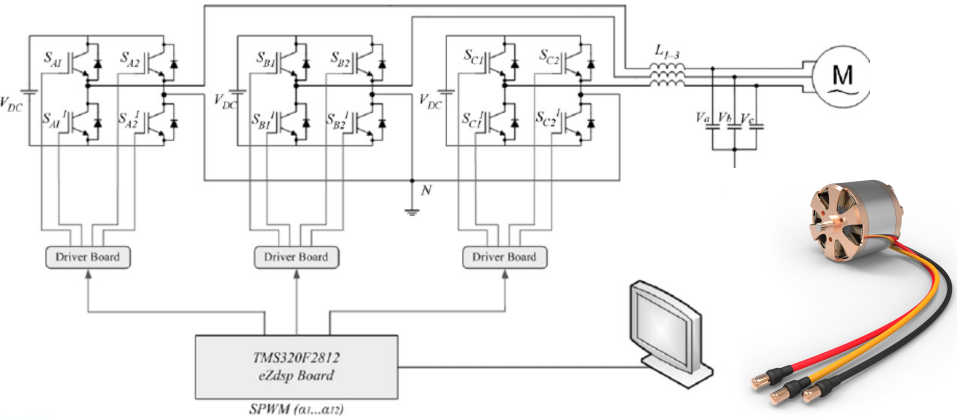
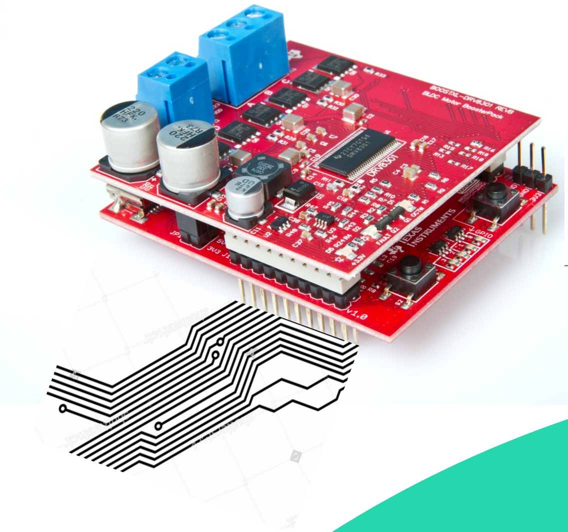










DSP Based Electrical Lab

Gokaraju Rangaraju Institute of Engineering & Technology (Autonomous)

Department of Electrical & Electronics Engineering

DSP Based Electrical Lab

IV Year - I Semester

by

Vinay Kumar A

Associate Professor

DEPARTMENT OF ELECTRICAL & ELECTORNICS ENGINEERING

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING & TECHNOLOGY
Nizampet Road, Bachupally, Kukatpally, Hyderabad-500090
Telangana, India. +91-040- 65864440, 65864441, www.griet.ac.in

GOKARAJU RANGARAJU INSTITUTE OF
ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

DEPARTMENT OF
ELECTRICAL & ELECTORNICS ENGINEERING

CERTIFICATE

*This is to certify that this book is a bonafide record of practical work done in the **DSP based Electrical Lab** in ....semester of ....year during the academic year.........*

Name : ....

Roll No : ....

Date : ....

Internal Examiner *External Examiner*

INDEX

Prog. No	Date	Title of the Program	Marks	Signature
1.		Blinking on-board LED		
2.		Watchdog with CPU Timer interrupts		
3.		Implementing *for* Loop		
4.		Factorial of a number using *for* Loop		
5.		Generation of a Square wave		
6.		Generation of Triangular wave		
7.		Generation of Sine wave		
8.		Acquisition of signal from ADC		
9.		Initializing the Event Manager		
10.		Generation of 1 kHz PWM Pulses at 50% and 75% Duty cycles		
11.		Generation of 5 kHz PWM Pulses at 25% Duty cycle		
12.		Generation of simple PWM pulses at 10KHz		
13.		Generation of ePWM pulses with a dead-band (delay routine)		
14.		An example to run a program in FLASH memory		
15.		Interfacing an external LED		
16.		Generation of SVPWM pulses for an Inverter operation		

Introduction:

A digital signal processor (DSP) is an integrated circuit designed for high-speed data manipulations, and is used in audio, communications, image manipulation, and other data-acquisition and data-control applications. The microprocessors used in personal computers are optimized for tasks involving data movement and inequality testing. The typical applications requiring such capabilities are word processing, database management, spread sheets, etc. When it comes to mathematical computations, the traditional microprocessor is deficient particularly where real-time performance is required. Digital signal processors are microprocessors optimized for basic mathematical calculations such as additions and multiplications.

A DSP system can be defined as an electronic system which can make use of digital signaling processing. Further which is the application of the mathematical operations to represent signals digitally. These signals are represented digitally as sequences of samples. Often, these samples are obtained from physical signals through the ADC and digital signals can be converted back to physical signals through DAC. Digital signal processing enjoys several advantages over analog signal processing. The most significant of these is that DSP systems can accomplish tasks inexpensively that would be difficult or even impossible using analog electronics. Examples of such applications include speech synthesis, speech recognition, and high-speed modems involving error-correction coding. These tasks involve a combination of signal processing and control (e.g., making decisions regarding received bits or received speech) that is extremely difficult to implement using analog techniques.

When we look for the applications DSP processors in electrical engineering, there are many environments where they can be used in controlling circuits such as in Inverter, controlled rectifier, protection systems, reactive power compensation systems like DVR, controlling speeds of motors like BLDC etc.

Types of DSP:

Digital signal processing can be separated into two categories - fixed point and floating point. These designations refer to the format used to store and manipulate numeric representations of data. Fixed-point DSPs are designed to represent and manipulate integers – positive and negative whole numbers – via a minimum of 16 bits, yielding up to 65,536 possible bit patterns (216). Floating-point DSPs represent and manipulate rational numbers via a minimum of 32 bits in a manner like scientific notation, where a number is represented with a mantissa and an exponent (e.g., A x 2B, where 'A' is the mantissa and ‘B’ is the exponent), yielding up to 4,294,967,296 possible bit patterns (232).

The term ‘fixed point’ refers to the corresponding way numbers are represented, with a fixed number of digits after, and sometimes before, the decimal point. With floating-point representation, the placement of the decimal point can ‘float’ relative to the significant digits of the number. For example, a fixed-point representation with a uniform decimal point placement convention can represent the numbers 123.45, 1234.56, 12345.67, etc, whereas a floating-point representation could in addition represent 1.234567, 123456.7, 0.00001234567, 1234567000000000, etc. As such, floating point can support a much wider range of values than fixed point, with the ability to represent very small numbers and very large numbers.

With fixed-point notation, the gaps between adjacent numbers always equal a value of one, whereas in floating-point notation, gaps between adjacent numbers are not uniformly spaced – the gap between any two numbers is approximately ten million times smaller than the value of the numbers (ANSI/IEEE Std. 754 standard format), with large gaps between large numbers and small gaps between small numbers.

Programing Language:

DSPs are programmed in the same languages as other scientific and engineering applications, usually assembly or C. Programs written in assembly can execute faster, while programs written in C are easier to develop and maintain. In traditional applications, such as programs run on personal computers and mainframes, C is almost always the first choice. If assembly is used at all, it is restricted to short subroutines that must run with the utmost speed.

However, DSP programs are different from traditional software tasks in two important respects. First, the programs are usually much shorter, say, one-hundred lines versus ten-thousand lines. Second, the execution speed is often a critical part of the application. This is the reason why many uses a DSP in the first place, for its blinding speed. These two factors motivate many software engineers to switch from C to assembly for programming Digital Signal Processors.

Architecture Overview:

TI Texas Instruments TMS320

Texas Instruments TMS320 is a blanket name for a series of digital signal processors (DSPs) from Texas Instruments. It was introduced on April 8, 1983 through the TMS32010 processor, which was then the fastest DSP on the market. The processor is available in many different variants, some with fixed-point arithmetic and some with floating point arithmetic. The floating point DSP TMS320C3x, which exploits delayed branch logic, has as many as three delay slots. The flexibility of this line of processors has led to it being used not merely as a co-processor for digital signal processing but also as a main CPU.

Newer implementations support standard IEEE **JTAG** control for boundary scan and/or in-circuit debugging. The original TMS32010 and its subsequent variants is an example of a CPU with a modified Harvard architecture, which features separate address spaces for instruction and data memory but the ability to read data values from instruction memory. The **TMS32010** featured a fast multiply-and-accumulate useful in both DSP applications as well as transformations used in computer graphics.

Outline of TMS320 series

- TMS320C1x, the first generation 16-bit fixed-point DSPs. All processors in these series are code-compatible with the TMS32010.
    - TMS32010, the very first processor in the first series introduced in 1983, using external memory.
    - TMS320M10, the same processor but with an internal ROM of 3 KB
    - TMS320C10, TMS320C15 etc.

- TMS320C3x, floating point
  - TMS320VC33
- TMS320C4x, floating point
- TMS320C8x, multiprocessor chip.
  - TMS320C80 MVP (multimedia video processor) has a 32-bit floating-point "master processor" and four 32-bit fixed-point "parallel processors". In many ways, the Cell microprocessor followed this design approach.

C2000 series

C2000 microcontroller family consists of 32-bit microcontrollers with performance integrated peripherals designed for real-time control applications. C2000 consists of 5 sub-families: the newer C28x + ARM Cortex M3 series, C28x Delfino floating-point series, C28x Piccolo series, C28x fixed-point series, and C240x, an older 16-bit line that is no longer recommended for new development. The C2000 series is notable for its high performance set of on-chip control peripherals including PWM, ADC, quadrature encoder modules, and capture modules. The series also contains support for I²C, SPI, serial (SCI), CAN, watchdog, McBSP, external memory interface and GPIO. Due to features like PWM waveform synchronization with the ADC unit, the C2000 line is well suited to many real-time control applications. The C2000 family is used for applications like motor drive and control, industrial automation, solar and other renewable energy, server farms, digital power, power line communications, and lighting. A line of low cost kits is available for key applications including motor control, digital power, solar, and LED lighting.

C5000 Series

- TMS320C54x 16-bit fixed-point DSP, 6 stage pipeline with in-order-execution of opcodes, parallel load/store on arithmetic operations, multiply accumulate and other DSP enhancements. Internal multi-port memory. no cache unit.
- A popular choice for 2G Software defined cellphone radios, particularly GSM, circa late 1990s when many Nokia and Ericsson cellphones made use of the C54x.
- At the time, desire to improve the user interface of cellphones led to the adoption of ARM7 as a general-purpose processor for user interface and control,

off-loading this function from the DSP. This ultimately led to the creation of a dual core ARM7+C54x DSP, which later evolved into the OMAP product line.

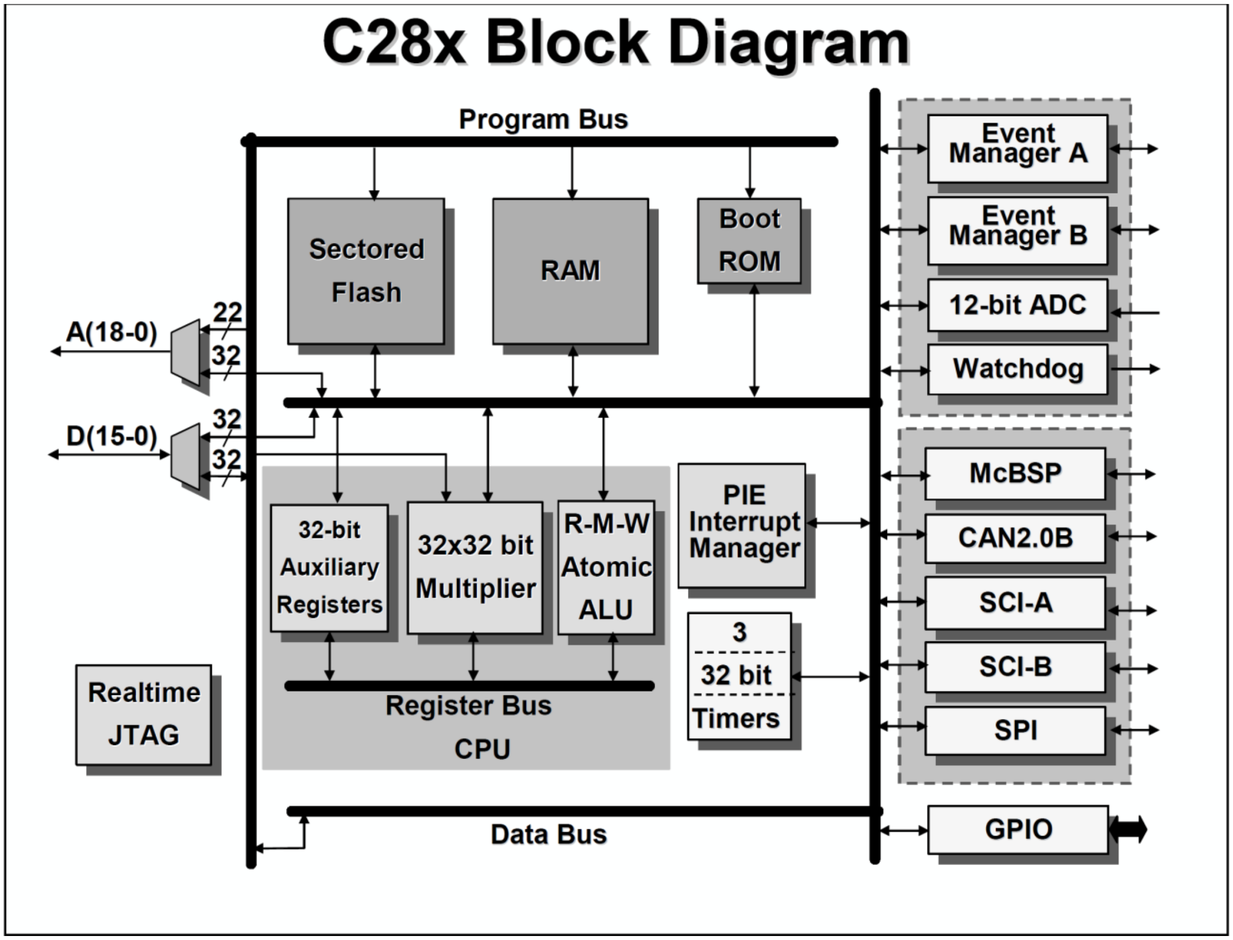
- TMS320C55x generation - fixed-point, runs C54x code but adds more internal parallelism (another ALU, dual MAC, more memory bandwidth) and registers, while supporting much lower power operation.
- Today, most C55x DSPs are sold as discrete chips
- OMAP1 chips combine an ARM9 (ARMv5TEJ) with a C55x series DSP.
- OMAP2420 chips combine an ARM11 (ARMv6) with a C55x series DSP.

C6000 Series

- TMS320 C6000 series, or TMS320C6x: VLIW-based DSPs
  - TMS320C62x fixed-point - 2000 MIPS/1.9 W
  - TMS320C67x floating point - code compatible with TMS320C62x
  - TMS320C64x fixed-point - code compatible with TMS320C62x
  - TMS320C67x+ floating point - architectural update of TMS320C67x
  - TMS320C64x+ fixed-point - major architectural update of TMS320C64x
  - TMS320C674x fixed- and floating point - merger of C64x+ and C67x+
  - TMS320C66x fixed- and floating point - backwards compatible with C674x
- Other parts with C6000 series DSPs include
  - DaVinci chips include one or both of an ARM9 and a C64x+ or C674x DSP
  - OMAP-L13x chips include an ARM9 (ARMv5TEJ) and a C674x fixed and floating point DSP
  - OMAP243x chips combine an ARM11 (ARMv6) with a C64x series DSP
  - OMAP3 chips include an ARM Cortex-A8 (ARMv7) with a C64x+ DSP
  - OMAP4 and OMAP5 chips include an ARM Cortex-A9 or A15 (ARMv7) with a custom C64x+ derivative known as Tesla (or C64T)

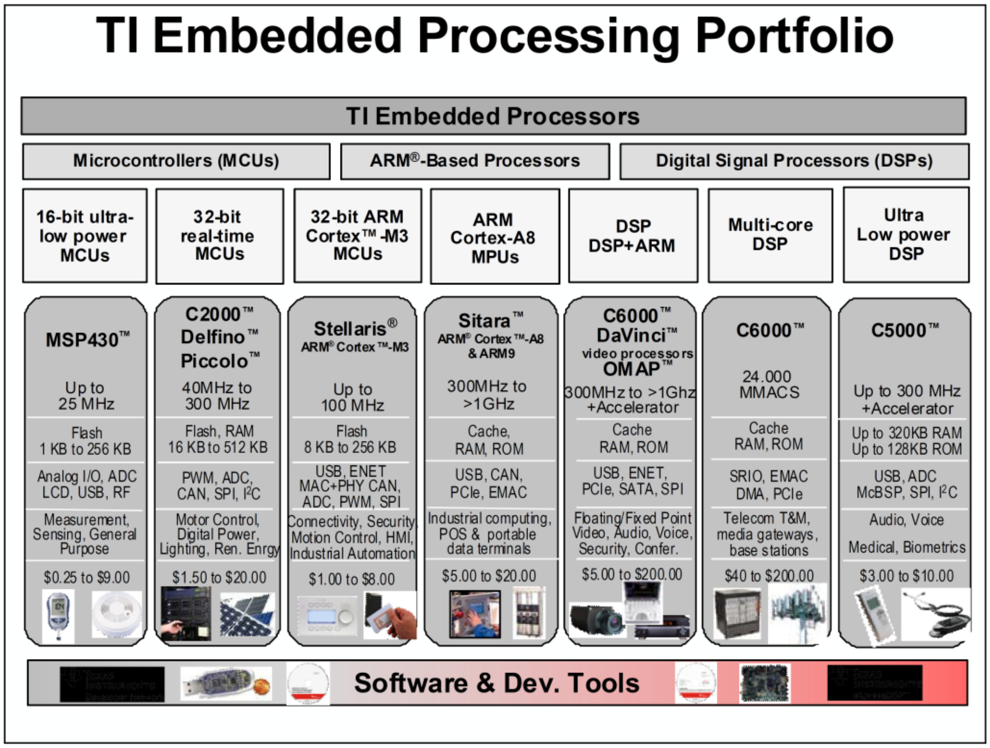
What is the TMS320C28x?

The TMS320C28x is a 32-bit fixed point DSP that specializes in high performance control applications such as, robotics, industrial automation, mass storage devices, lighting, optical networking, power supplies, and other control applications needing a single processor to solve a high-performance application.



The C28x architecture can be divided into 3 functional blocks:

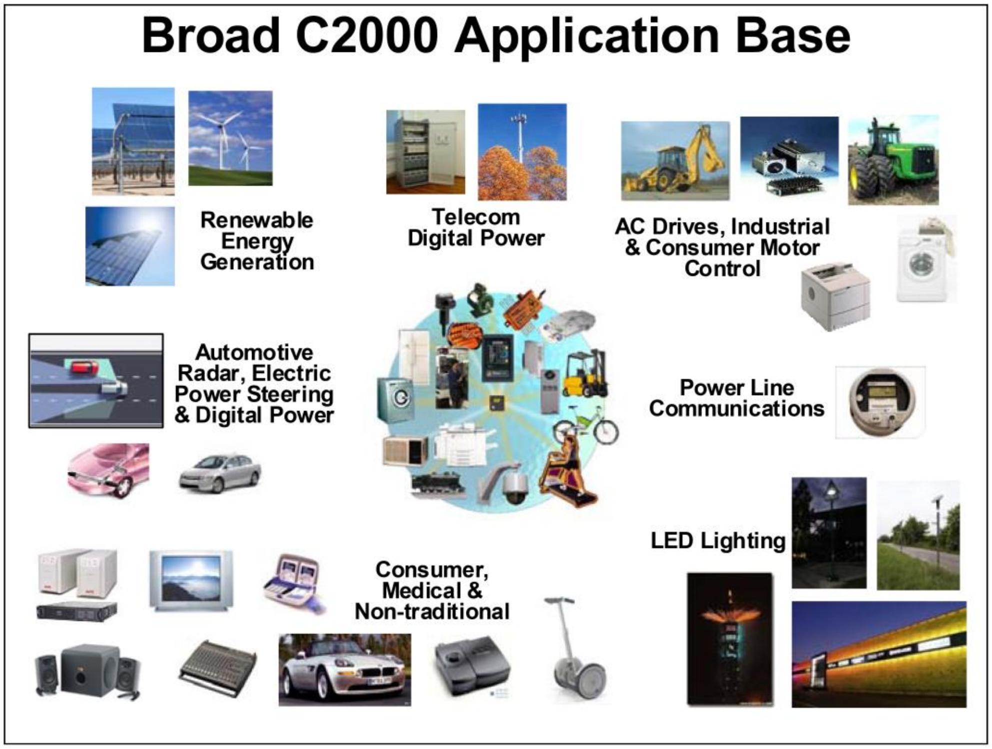
- CPU and busing
- Memory
- Peripherals



TI Embedded Processing Portfolio

TI Embedded Processors						
Microcontrollers (MCUs)		ARM®-Based Processors		Digital Signal Processors (DSPs)		
16-bit ultra-low power MCUs	32-bit real-time MCUs	32-bit ARM Cortex™ -M3 MCUs	ARM Cortex-A8 MPUs	DSP DSP+ARM	Multi-core DSP	Ultra Low power DSP
MSP430™	C2000™ Delfino™ Piccolo™	Stellaris® ARM® Cortex™-M3	Sitara™ ARM® Cortex™-A8 & ARM9	C6000™ DaVinci™ video processors OMAP™	C6000™	C5000™
Up to 25 MHz	40MHz to 300 MHz	Up to 100 MHz	300MHz to >1GHz	300MHz to >1Ghz +Accelerator	24.000 MMACS	Up to 300 MHz +Accelerator
Flash 1 KB to 256 KB	Flash, RAM 16 KB to 512 KB	Flash 8 KB to 256 KB	Cache, RAM, ROM	Cache RAM, ROM	Cache RAM, ROM	Up to 320KB RAM Up to 128KB ROM
Analog I/O, ADC LCD, USB, RF	PWM, ADC, CAN, SPI, I^{2}C	USB, ENET MAC+PHY CAN, ADC, PWM, SPI	USB, CAN, PCIe, EMAC	USB, ENET, PCIe, SATA, SPI	SRIO, EMAC DMA, PCIe	USB, ADC McBSP, SPI, I^{2}C
Measurement, Sensing, General Purpose	Motor Control, Digital Power, Lighting, Ren. Enrgy	Connectivity, Security, Motion Control, HMI, Industrial Automation	Industrial computing, POS & portable data terminals	Floating/Fixed Point Video, Audio, Voice, Security, Confer.	Telecom T&M, media gateways, base stations	Audio, Voice Medical, Biometrics
$0.25 to $9.00	$1.50 to $20.00	$1.00 to $8.00	$5.00 to $20.00	$5.00 to $200.00	$40 to $200.00	$3.00 to $10.00

Software & Dev. Tools



Broad C2000 Application Base

Renewable Energy Generation

Telecom Digital Power

AC Drives, Industrial & Consumer Motor Control

Automotive Radar, Electric Power Steering & Digital Power

Power Line Communications

LED Lighting

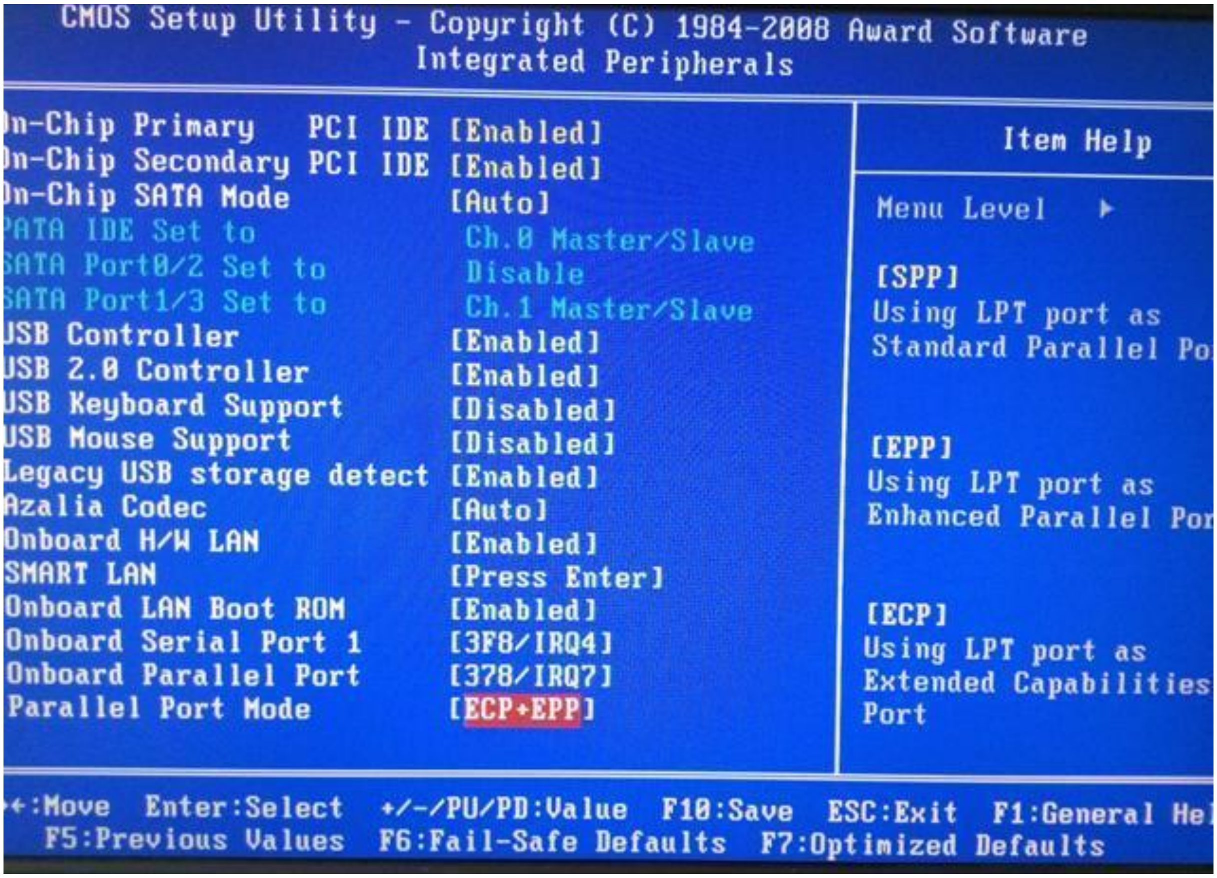
Consumer, Medical & Non-traditional

Hardware and Software Requirements:

1. EzDSPF2812 Kit.
2. Parallel Port cable
3. Power supply
3. Code Composer studio V5 or V6
4. OS-Windows 7

Instructions to configure the Computer Parallel Port

1. Enter into BIOS mode by pressing DEL or F2 Key
2. Go to IO Configuration (Option Differs based on Mother Board Manufacture)
3. Set parallel port address as 0x378 and mode as EPP/ECP
4. Press F10 to Save and Exit.
5. Refer below image for reference



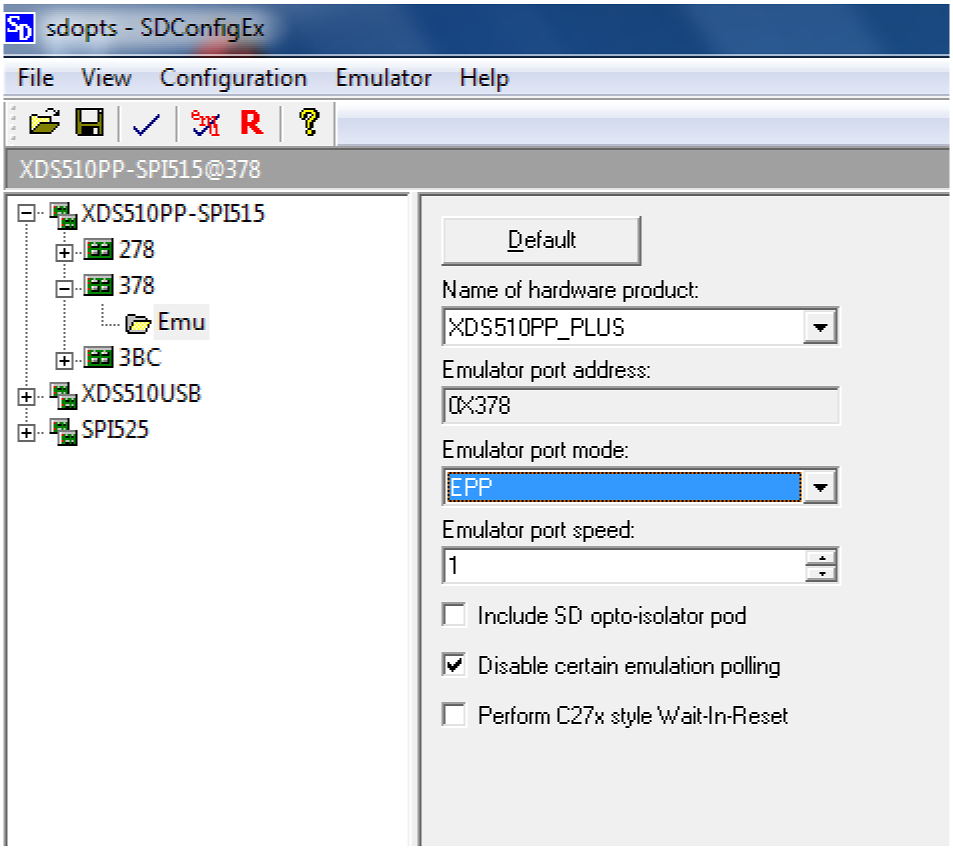
```
   CMOS Setup Utility - Copyright (C) 1984-2008 Award Software
                      Integrated Peripherals

n-Chip Primary   PCI IDE [Enabled]            |     Item Help
n-Chip Secondary PCI IDE [Enabled]            |
n-Chip SATA Mode         [Auto]               | Menu Level  ▸
PATA IDE Set to           Ch.0 Master/Slave   |
SATA Port0/2 Set to       Disable             | [SPP]
SATA Port1/3 Set to       Ch.1 Master/Slave   | Using LPT port as
USB Controller           [Enabled]            | Standard Parallel Po
USB 2.0 Controller       [Enabled]            |
USB Keyboard Support     [Disabled]           |
USB Mouse Support        [Disabled]           | [EPP]
Legacy USB storage detect [Enabled]           | Using LPT port as
Azalia Codec             [Auto]               | Enhanced Parallel Por
Onboard H/W LAN          [Enabled]            |
SMART LAN                [Press Enter]        |
Onboard LAN Boot ROM     [Enabled]            | [ECP]
Onboard Serial Port 1    [3F8/IRQ4]           | Using LPT port as
Onboard Parallel Port    [378/IRQ7]           | Extended Capabilities
Parallel Port Mode       [ECP+EPP]            | Port

↔:Move  Enter:Select  +/-/PU/PD:Value  F10:Save  ESC:Exit  F1:General He
  F5:Previous Values  F6:Fail-Safe Defaults  F7:Optimized Defaults
```

Instructions to install Code Composer Studio V5:

1. Launch the setup from the CCS V5 CD
2. Accept the agreement and NEXT
3. Select the folder to install “default C:\TI” and NEXT
4. Select custom and NEXT
5. Select only C28x 32bit Real time CPU MCU and NEXT
6. In Compiler tools, Select TI C2800 Compiler tools and TI Documentation
7. In device software select both DSP BIOS V5 /SYS BIOS v6
8. Select TI Simulators and NEXT
9. In JTAG Emulator Support select Spectrum digital emulators, TI Emulators(Default), XDS100Emulators and NEXT
10. In CCS Install Options window and NEXT
11. Finally, it will take 20 minutes install the CCS

Instructions to verify the ezDSP’s connection with *sdconfig* :

1. Connect the ezDSP with the Computer with parallel port cable and Power on the ezDSP board
2. Open SdConfigEx v5 from the desktop
3. Double Click XDS510PP-SPI515 and select 378.
4. Double click 378 and select emu and Change the Emulator port mode to EPP as shown below



Name of hardware product: XDS510PP_PLUS

Emulator port address: 0X378

Emulator port mode: EPP

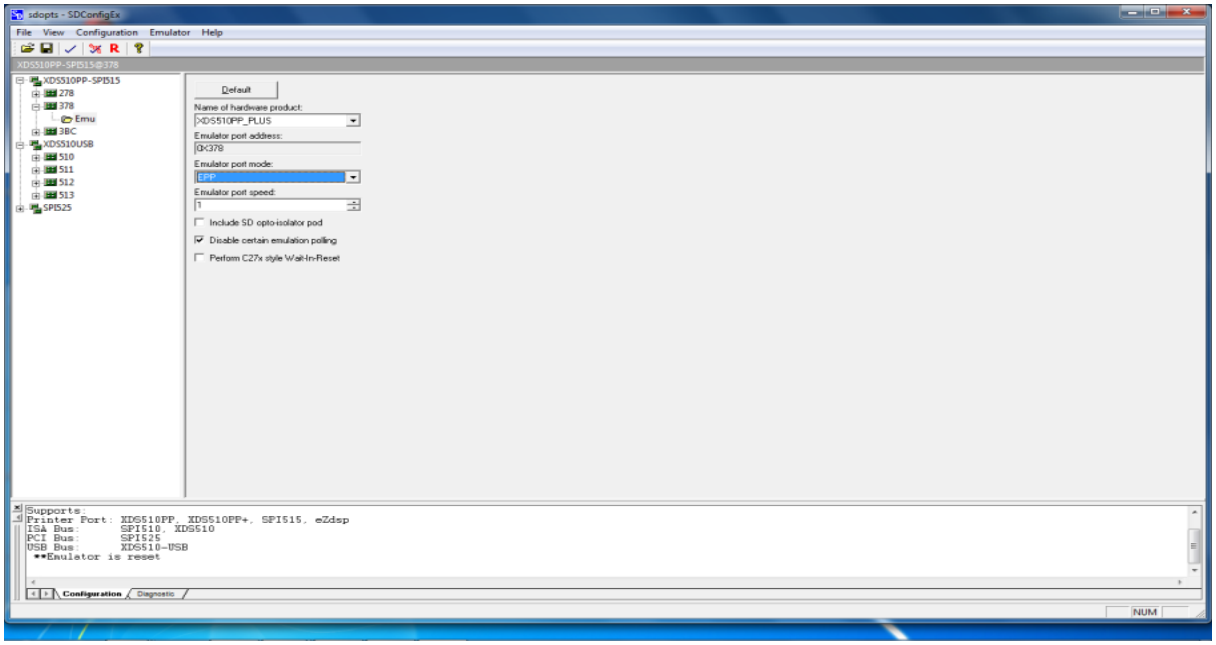
Emulator port speed: 1

☐ Include SD opto-isolator pod

☑ Disable certain emulation polling

☐ Perform C27x style Wait-In-Reset

5. Now Press the R Button or Go to Emulator Menu and Select Reset
6. "Emulator is reset" message will display in the configuration Tab as shown below



Name of hardware product: XDS510PP_PLUS

Emulator port address: 0X378

Emulator port mode: EPP

Emulator port speed: 1

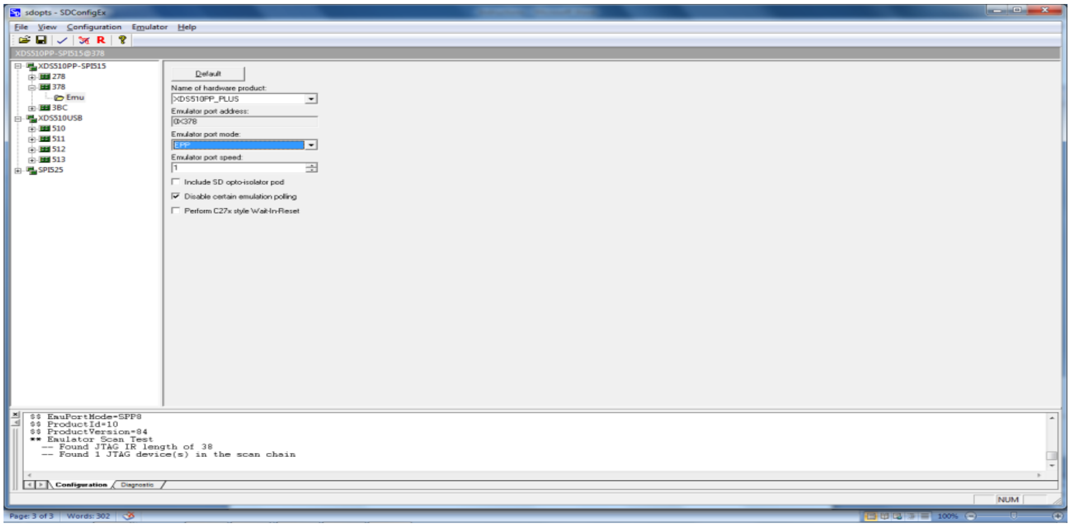
□ Include SD opto-isolator pod

☑ Disable certain emulation polling

□ Perform C27x style Wait-In-Reset

```
Supports:
Printer Port: XDS510PP, XDS510PP+, SPI515, eZdsp
ISA Bus:      SPI510, XDS510
PCI Bus:      SPI525
USB Bus:      XDS510-USB
 **Emulator is reset
```

7. Now Press the EMU with Tick Button or Go to Emulator Menu and Select Test.
8. JTAG IR Length of 38 Message will display in Configuration tab as shown below



Name of hardware product: XDS510PP_PLUS

Emulator port address: 0X378

Emulator port mode: EPP

Emulator port speed: 1

□ Include SD opto-isolator pod

☑ Disable certain emulation polling

□ Perform C27x style Wait-In-Reset

```
$$ EmuPortMode=SPP8
$$ ProductId=10
$$ ProductVersion=84
** Emulator Scan Test
   -- Found JTAG IR length of 38
   -- Found 1 JTAG device(s) in the scan chain
```

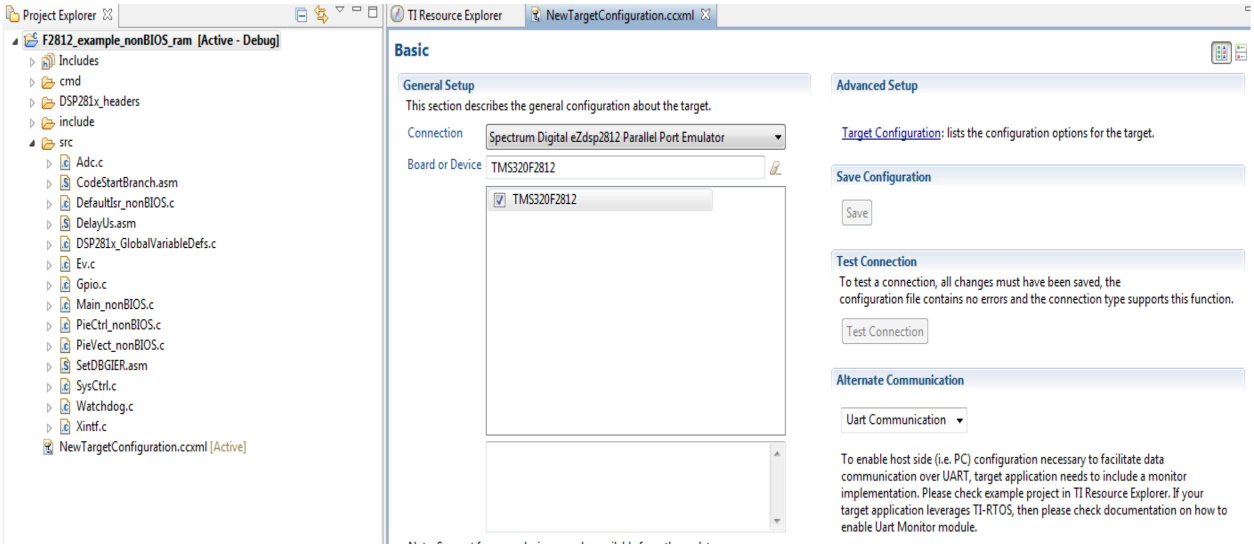
9. If the Emulator rest and JTAG IR length as 38 shows the connection between the system and ezDSP is OK.
10. Now close SD Config.

CCS V6 License Setup.

1. Open CCS V6
2. Go to Help Menu->Code Composer Studio License Information
3. Go to Upgrade Tab-> launch License setup
4. Select Evaluate(90days) or Free License (Onboard and XDS 100 Emulators)
5. Press Finish Button.

Instructions to configure and run sample programs in CCS V6.

1. Open **CCSv6**
2. It will ask for **workspace** location (By default it is user directory) and select **OK**
3. Go to Project Menu-> Import Existing CCS Project
4. Now Select the search directory to F2812_example_nonBIOS_ram and press **finish** button
5. Go to File menu ->New->Target Configuration File and Press **finish** button in the newly opened window
6. Now it will ask you to select the Connection Type and Board Type and save as shown below



```
F2812_example_nonBIOS_ram [Active - Debug]
  Includes
  cmd
  DSP281x_headers
  include
  src
    Adc.c
    CodeStartBranch.asm
    DefaultIsr_nonBIOS.c
    DelayUs.asm
    DSP281x_GlobalVariableDefs.c
    Ev.c
    Gpio.c
    Main_nonBIOS.c
    PieCtrl_nonBIOS.c
    PieVect_nonBIOS.c
    SetDBGIER.asm
    SysCtrl.c
    Watchdog.c
    Xintf.c
  NewTargetConfiguration.ccxml [Active]
```

This section describes the general configuration about the target.

Connection: Spectrum Digital eZdsp2812 Parallel Port Emulator

Board or Device: TMS320F2812

☑ TMS320F2812

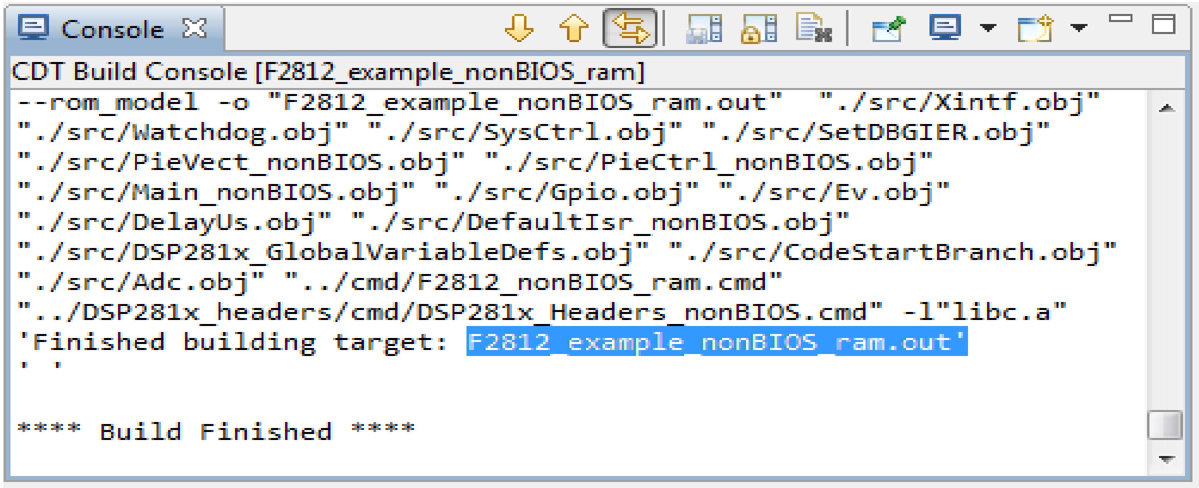
Target Configuration: lists the configuration options for the target.

To test a connection, all changes must have been saved, the configuration file contains no errors and the connection type supports this function.

Uart Communication

To enable host side (i.e. PC) configuration necessary to facilitate data communication over UART, target application needs to include a monitor implementation. Please check example project in TI Resource Explorer. If your target application leverages TI-RTOS, then please check documentation on how to enable Uart Monitor module.

7. Go to Project menu->**Build all**
8. After project built, **.out** file will be generated as shown on Console window



```
CDT Build Console [F2812_example_nonBIOS_ram]
--rom_model -o "F2812_example_nonBIOS_ram.out"  "./src/Xintf.obj"
"./src/Watchdog.obj" "./src/SysCtrl.obj" "./src/SetDBGIER.obj"
"./src/PieVect_nonBIOS.obj" "./src/PieCtrl_nonBIOS.obj"
"./src/Main_nonBIOS.obj" "./src/Gpio.obj" "./src/Ev.obj"
"./src/DelayUs.obj" "./src/DefaultIsr_nonBIOS.obj"
"./src/DSP281x_GlobalVariableDefs.obj" "./src/CodeStartBranch.obj"
"./src/Adc.obj" "../cmd/F2812_nonBIOS_ram.cmd"
"../DSP281x_headers/cmd/DSP281x_Headers_nonBIOS.cmd" -l"libc.a"
'Finished building target: F2812_example_nonBIOS_ram.out'
' '

**** Build Finished ****
```

9. Go to Run Menu -> Select Debug or F11 Key
10. To run program, Go to Run Menu -> Select **Resume** or F8 Key
11. Now the DS2 Led in the ezDSP F2812 will blinking continuously
12. Then Go to Run Menu ->Select suspend then select terminate.

Instructions to Create a New Project in CCS V6

1. Open CCSv6
2. Go to File Menu-> New -> CCS Project
3. Type Project name and other Leave it to default
4. Select Device family as C2000 and variant as 281X Fixed Point and EZDSPf2812
5. Connection as Spectrum Digital ezDSP F2812 Parallel port Emulator
6. Select project templates as empty project and press finish button
7. Now add source files and Cmd by right click the project name in the Project Explorer
8. Follow the Step s 5 to 12 from **Instructions to configure and run sample programs in CCS V6.**

Program No 1: *Blinking on-board LED*

Date:

Objective:

To run a program that blinks the onboard LED

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
// TITLE:   DSP28027 LED Blink Getting Started Program.
#include "DSP28x_Project.h"
interrupt void cpu_timer0_isr(void);
void main(void)
{
  InitSysCtrl();
  DINT;
  InitPieCtrl();
  IER = 0x0000;
  IFR = 0x0000;
  InitPieVectTable();
  EALLOW;
  PieVectTable.TINT0  = &cpu_timer0_isr;
  EDIS;
  InitCpuTimers();
  ConfigCpuTimer(&CpuTimer0,  60, 500000);
  CpuTimer0Regs.TCR.all  = 0x4001;
  EALLOW;
  GpioCtrlRegs.GPBMUX1.bit.GPIO34  = 0;
  GpioCtrlRegs.GPBDIR.bit.GPIO34  = 1;
  EDIS;
  IER  |= M_INT1;
  PieCtrlRegs.PIEIER1.bit.INTx7   = 1;
  EINT;
  ERTM;
  for(;;);
}

interrupt void cpu_timer0_isr(void)
{
  CpuTimer0.InterruptCount++;
  GpioDataRegs.GPBTOGGLE.bit.GPIO34  = 1;
  PieCtrlRegs.PIEACK.all  = PIEACK_GROUP1;
}
//===========================================================================
// No more.
//===========================================================================
```

Result:

Watch Variables:

CpuTimer0.InterruptCount

*Monitor the **GPIO34** LED blink ON and OFF on the TMS320F28027 Launchpad.*

Program No 2: *Watchdog with CPU Timer interrupts*

Date:

Objective:

To run a program that configures the CPU *timer* and *counter*

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
// TITLE:   DSP28027 Device Getting Started with timer and counter

#include "DSP28x_Project.h"

interrupt void cpu_timer0_isr(void);

interrupt void cpu_timer1_isr(void);

interrupt void cpu_timer2_isr(void);

void main(void)

{

  InitSysCtrl();

  DINT;
  InitPieCtrl();

  IER = 0x0000;

  IFR = 0x0000;

  InitPieVectTable();

  EALLOW;

  PieVectTable.TINT0  = &cpu_timer0_isr;

  PieVectTable.TINT1  = &cpu_timer1_isr;

  PieVectTable.TINT2  = &cpu_timer2_isr;

  EDIS;

  InitCpuTimers();

#if (CPU_FRQ_60MHZ)

  ConfigCpuTimer(&CpuTimer0,  60, 1000000);

  ConfigCpuTimer(&CpuTimer1,  60, 1000000);

  ConfigCpuTimer(&CpuTimer2,  60, 1000000);

#endif

#if (CPU_FRQ_50MHZ)

  ConfigCpuTimer(&CpuTimer0,  50, 1000000);

  ConfigCpuTimer(&CpuTimer1,  50, 1000000);

  ConfigCpuTimer(&CpuTimer2,  50, 1000000);

#endif
```

```
#if (CPU_FRQ_40MHZ)
   ConfigCpuTimer(&CpuTimer0,  40, 1000000);
   ConfigCpuTimer(&CpuTimer1,  40, 1000000);
   ConfigCpuTimer(&CpuTimer2,  40, 1000000);
#endif
   CpuTimer0Regs.TCR.all  = 0x4001;
   CpuTimer1Regs.TCR.all  = 0x4001;
   CpuTimer2Regs.TCR.all  = 0x4001;
   IER |= M_INT1;
   IER |= M_INT13;
   IER |= M_INT14;
   PieCtrlRegs.PIEIER1.bit.INTx7  = 1;
   EINT;
   ERTM;
   for(;;);
}
interrupt void cpu_timer0_isr(void)
{
   CpuTimer0.InterruptCount++;
   PieCtrlRegs.PIEACK.all  = PIEACK_GROUP1;
}
```

```
interrupt void cpu_timer1_isr(void)

{

  CpuTimer1.InterruptCount++;

  EDIS;

}

interrupt void cpu_timer2_isr(void)

{

  EALLOW;

  CpuTimer2.InterruptCount++;

  EDIS;

}

//===========================================================================

// No more.

//===========================================================================
```

Result:

Watch Variables:

CpuTimer0.InterruptCount

CpuTimer1.InterruptCount

CpuTimer2.InterruptCount

Observe the timer registers and configuration of CPU Timer0, 1, & 2 and increments a counter each time the timer asserts an interrupt.

Program No 3: *Implementing a for loop*

Date:

Objective:

To run a program to find square of a given number using *for* loop

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
        unsigned int k;
        void main(void)
        {
                unsigned int i;
                while(1)
                {
                        for(i=0; i<100; i++)
                        k=i*i;
                }
        }
//==========================================================================

// No more.

//==========================================================================
```

Result:

Watch variables:

i

k

Observe the variables at each step forward at watchdog and find the square of the given number.

Program No 4: *Factorial of a number using for loop*

Date:

Objective:

To run a program to find factorial of a given number using *for* loop

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
        #include<stdio.h>
        int main()
        {
        int input,i,result=1;
        printf("please input a Integer: ");
        scanf("%d",&input);
        for(i=input;i>0;i--)
              {
              result=result*i;
              }
        printf("the factorial of %d is %d\n",input,result);
        }
//====================================================================================

// No more.

//====================================================================================
```

Result:

Watch variables:

i

$result$

Observe the variables at each step forward at watchdog and find the factorial of the given number.

Program No 5: *Generation of a Square wave*

Date:

Objective:

To write a program to generate a Square wave

Equipment required:

Hardware:

- PC
- TMS320F2812
- Power supply adaptor cable
- DB25 connector printer cable

Software:

- Code composer studio 5.5.0
- Windows 8 OS.

Program:

```
# include<stdio.h>
#include<math.h>
void main()
{
        int *square;
        int i;
        square =(int*)0xC0000000;
        while(1)
        {
            for(i=0;i<50;i++)
            {
                *square++=0x0000FFFF;
            }
            for(i=0;i<50;i++)
            {
                *square++=0x0;
            }
        }
    }
//===========================================================================
// No more.
//===========================================================================
```

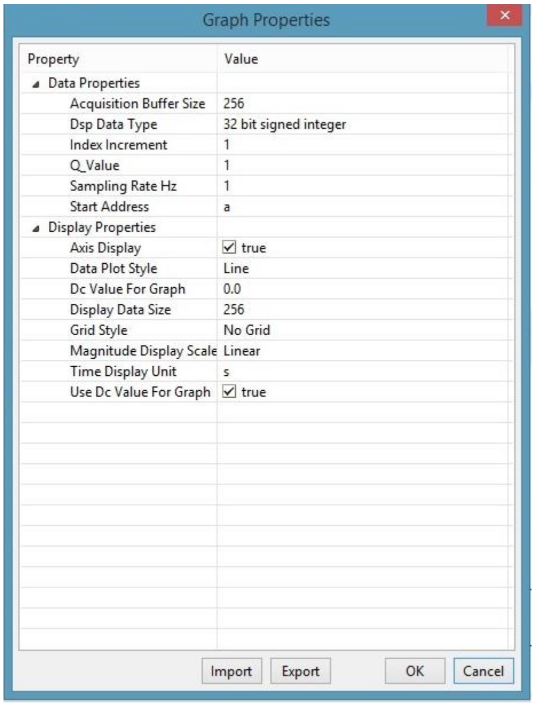
Result:

Watch variables

i

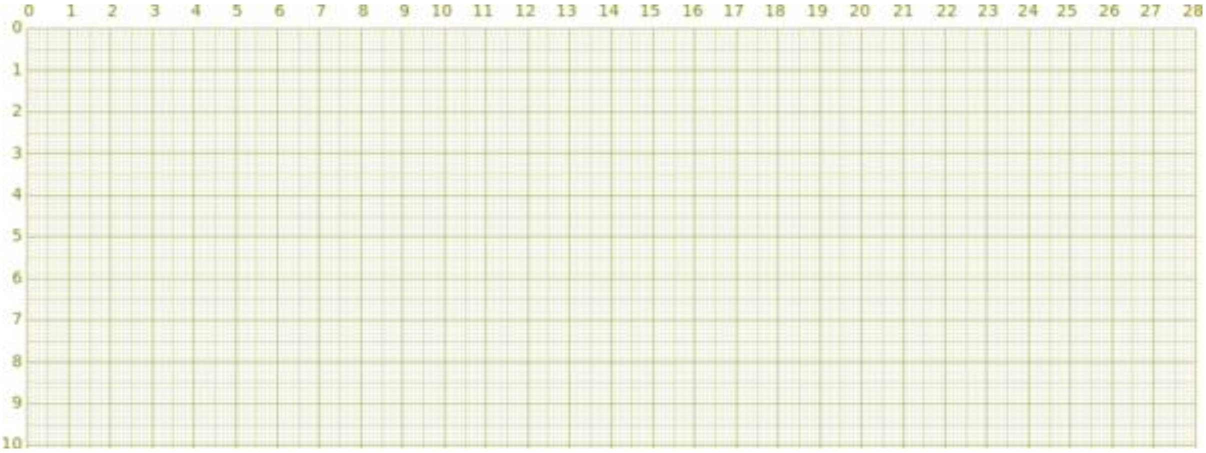
$*square$

View the graph of square wave by entering the following graph properties.



Property	Value
◢ Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	32 bit signed integer
Index Increment	1
Q_Value	1
Sampling Rate Hz	1
Start Address	a
◢ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Dc Value For Graph	0.0
Display Data Size	256
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	s
Use Dc Value For Graph	☑ true

Graph:



Program No 6: *Generation of a Triangular wave*

Date:

Objective:

To write a program to generate a Triangular wave

Equipment required:

Hardware:

- PC
- TMS320F2812
- Power supply adaptor cable
- DB25 connector printer cable

Software:

- Code composer studio 5.5.0
- Windows 8 OS.

Program:

```
//Generation of Triangular wave
#include <stdio.h>
 #include <math.h>
void main()
 {
int *Triangle;
int i=0,j=0;
 Triangle = (int*)0xC0000000;
while(1)
 {               for(i=0;i<50;i++)
     {
  j=j+1;
 *Triangle++  = j;
 }
 for(i=50;i>0;i--)
  {                    j=j-1;
 *Triangle++  = j;
   }
  }
 }
 //===========================================================================
 // No more.
 //===========================================================================
```

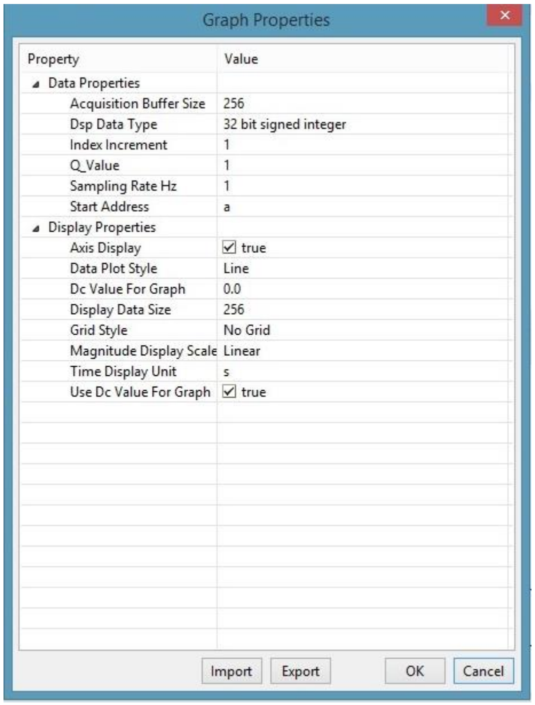
Result:

Watch variables

i

$*triangle$

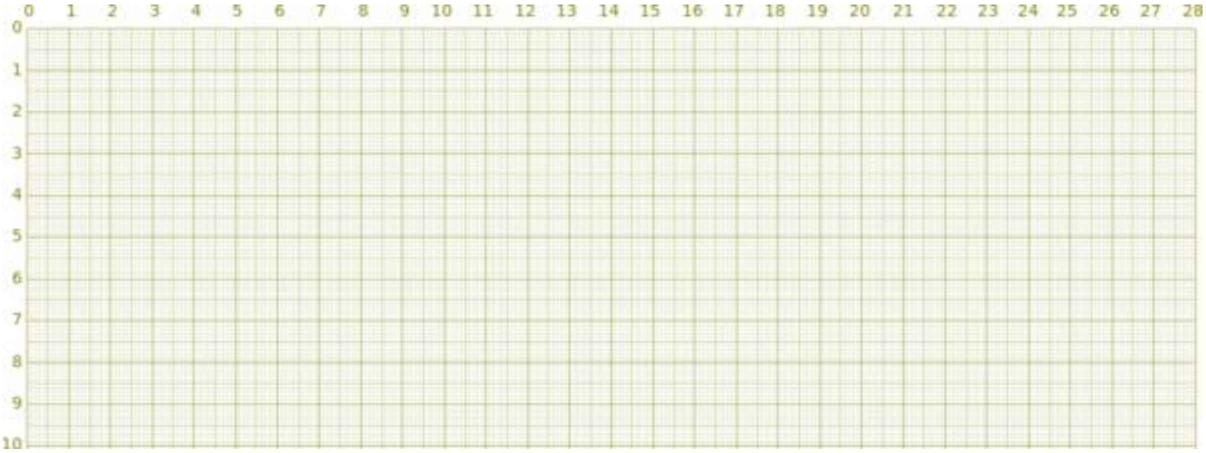
View the graph of triangular wave by entering the following graph properties.



Graph Properties

Property	Value
◢ Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	32 bit signed integer
Index Increment	1
Q_Value	1
Sampling Rate Hz	1
Start Address	a
◢ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Dc Value For Graph	0.0
Display Data Size	256
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	s
Use Dc Value For Graph	☑ true

Graph:



Program No 7: *Generation of a Sine wave*

Date:

Objective:

To write a program to generate a Sine wave

Equipment required:

Hardware:

- PC
- TMS320F2812
- Power supply adaptor cable
- DB25 connector printer cable

Software:

- Code composer studio 5.5.0
- Windows 8 OS.

Program:

```
//Generation of Sine wave
#include<stdio.h>
#include<math.h>
float a[128];
main()
{
int i;
for (i=0;i<128;i++)
{
```

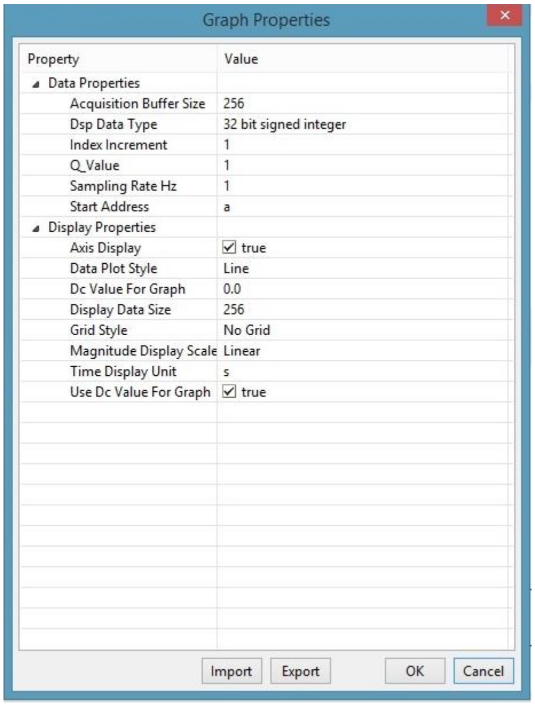
```
a[i]= sin(2*3.14*1000*i/24000);
printf("%f", a[i]);
}
}
//=================================================================
// No more.
//=================================================================
```

Result:

Watch variables

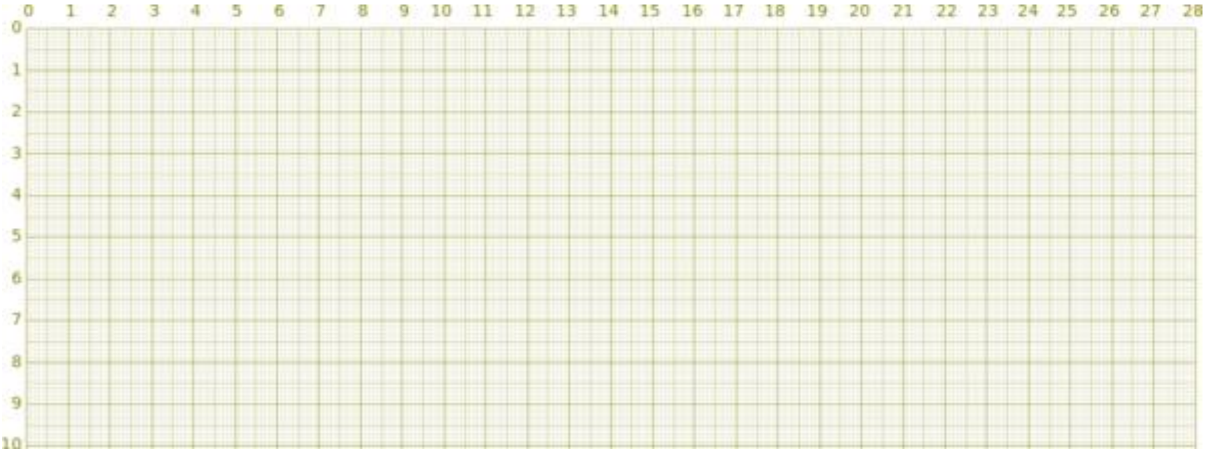
i

View the graph of Sine wave by entering the following graph properties.



Property	Value
◢ Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	32 bit signed integer
Index Increment	1
Q_Value	1
Sampling Rate Hz	1
Start Address	a
◢ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Dc Value For Graph	0.0
Display Data Size	256
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	s
Use Dc Value For Graph	☑ true

Graph:



Program No 8: *Acquisition of signal from ADC*

Date:

Objective:

To write a program to acquire a signal from ADC terminals.

Equipments required:

Hardware:

- PC
- TMS320F2812 eZdsp kit
- Power supply adaptor cable
- DB25 connector printer cable

Software:

- Code composer studio 5.5.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"

interrupt void adc_isr(void);

void Adc_Config(void);

Uint16 LoopCount;

Uint16 ConversionCount;

Uint16 Voltage1[10];

Uint16 Voltage2[10];


main()

{

  InitSysCtrl();

  DINT;

  InitPieCtrl();
```

```
IER = 0x0000;
IFR = 0x0000;
InitPieVectTable();
EALLOW;
PieVectTable.ADCINT1  = &adc_isr;
EDIS;
InitAdc();
PieCtrlRegs.PIEIER1.bit.INTx1  = 1;
IER |= M_INT1;
EINT;
ERTM;
LoopCount = 0;
ConversionCount = 0;

    EALLOW;
    AdcRegs.ADCCTL1.bit.INTPULSEPOS       = 1;
    AdcRegs.INTSEL1N2.bit.INT1E           = 1;
    AdcRegs.INTSEL1N2.bit.INT1CONT        = 0;
    AdcRegs.INTSEL1N2.bit.INT1SEL         = 2;
    AdcRegs.ADCSOC0CTL.bit.CHSEL          = 4;
    AdcRegs.ADCSOC1CTL.bit.CHSEL          = 4;
    AdcRegs.ADCSOC2CTL.bit.CHSEL          = 2;
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL        = 5;
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL        = 5;
    AdcRegs.ADCSOC2CTL.bit.TRIGSEL        = 5;
    AdcRegs.ADCSOC0CTL.bit.ACQPS          = 6;
```

```
        AdcRegs.ADCSOC1CTL.bit.ACQPS          = 6;
        AdcRegs.ADCSOC2CTL.bit.ACQPS          = 6;
        EDIS;

     EPwm1Regs.ETSEL.bit.SOCAEN    = 1;
     EPwm1Regs.ETSEL.bit.SOCASEL   = 4;
     EPwm1Regs.ETPS.bit.SOCAPRD    = 1;
     EPwm1Regs.CMPA.half.CMPA      = 0x0080;
     EPwm1Regs.TBPRD               = 0xFFFF;
     EPwm1Regs.TBCTL.bit.CTRMODE        = 0;

     for(;;)
     {
      LoopCount++;
     }
    }
    interrupt void  adc_isr(void)
    {
     Voltage1[ConversionCount]  = AdcResult.ADCRESULT1;
     Voltage2[ConversionCount]  = AdcResult.ADCRESULT2;
     if(ConversionCount  == 9)
     {
      ConversionCount = 0;
     }
     else ConversionCount++;

     AdcRegs.ADCINTFLGCLR.bit.ADCINT1  = 1;
     PieCtrlRegs.PIEACK.all  = PIEACK_GROUP1;
     return;
    }
//===============================================================

// No more.

//===============================================================
```

Result:

Watch Variables:

Voltage1[10]	*Last 10 ADCRESULT0 values*
Voltage2[10]	*Last 10 ADCRESULT1 values*
ConversionCount	*Current result number 0-9*
LoopCount	*Idle loop counter*

Observe the values from the ADC (GPIO) pins of the processor.

Program No 9: *Initializing the Event Manager*

Date:

Objective:

To write a program to fire an interrupt by initializing an event manager.

Equipment required:

Hardware:

- PC
- TMS320F2812 eZdsp kit
- Power supply adaptor cable
- DB25 connector printer cable

Software:

- Code composer studio 5.5.0
- Windows 8 OS.

Program:

```
#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"
interrupt void eva_timer1_isr(void);
interrupt void eva_timer2_isr(void);
interrupt void evb_timer3_isr(void);
interrupt void evb_timer4_isr(void);
void init_eva_timer1(void);
void init_eva_timer2(void);
void init_evb_timer3(void);
void init_evb_timer4(void);

Uint32  EvaTimer1InterruptCount;
Uint32  EvaTimer2InterruptCount;
Uint32  EvbTimer3InterruptCount;
Uint32  EvbTimer4InterruptCount;
```

```
void main(void)
{

  InitSysCtrl();
  DINT;
  InitPieCtrl();

  IER = 0x0000;
  IFR = 0x0000;

  InitPieVectTable();

  EALLOW;
  PieVectTable.T1PINT  = &eva_timer1_isr;
  PieVectTable.T2PINT  = &eva_timer2_isr;
  PieVectTable.T3PINT  = &evb_timer3_isr;
  PieVectTable.T4PINT  = &evb_timer4_isr;
  EDIS;

  init_eva_timer1();
  init_eva_timer2();
  init_evb_timer3();
  init_evb_timer4();

   EvaTimer1InterruptCount  = 0;
   EvaTimer2InterruptCount  = 0;
   EvbTimer3InterruptCount  = 0;
   EvbTimer4InterruptCount  = 0;

   PieCtrlRegs.PIEIER2.all  = M_INT4;
   PieCtrlRegs.PIEIER3.all  = M_INT1;
  PieCtrlRegs.PIEIER4.all  = M_INT4;
  PieCtrlRegs.PIEIER5.all  = M_INT1;
```

```
    IER |= (M_INT2 | M_INT3 | M_INT4 | M_INT5);

    EINT;
    ERTM;

    for(;;);

}
void init_eva_timer1(void)
{
    EvaRegs.GPTCONA.all                = 0;
    EvaRegs.T1PR                       = 0x0200;
    EvaRegs.T1CMPR                     = 0x0000;
    EvaRegs.EVAIMRA.bit.T1PINT         = 1;
    EvaRegs.EVAIFRA.bit.T1PINT         = 1;
    EvaRegs.T1CNT                      = 0x0000;
    EvaRegs.T1CON.all                  = 0x1742;
    EvaRegs.GPTCONA.bit.T1TOADC  = 2;

}

void init_eva_timer2(void)
{
    EvaRegs.GPTCONA.all                = 0;
    EvaRegs.T2PR                       = 0x0400;
    EvaRegs.T2CMPR                     = 0x0000;
    EvaRegs.EVAIMRB.bit.T2PINT         = 1;
    EvaRegs.EVAIFRB.bit.T2PINT         = 1;
    EvaRegs.T2CNT                      = 0x0000;
    EvaRegs.T2CON.all                  = 0x1742;
    EvaRegs.GPTCONA.bit.T2TOADC  = 2;
}
```

```
void init_evb_timer3(void)
{

    EvbRegs.GPTCONB.all                = 0;

    EvbRegs.T3PR                       = 0x0800;
    EvbRegs.T3CMPR                     = 0x0000;
    EvbRegs.EVBIMRA.bit.T3PINT         = 1;
    EvbRegs.EVBIFRA.bit.T3PINT         = 1;
    EvbRegs.T3CNT                      = 0x0000;
    EvbRegs.T3CON.all                  = 0x1742;
    EvbRegs.GPTCONB.bit.T3TOADC  = 2;
}

void init_evb_timer4(void)
{
    EvbRegs.GPTCONB.all                = 0;
    EvbRegs.T4PR                       = 0x1000;
    EvbRegs.T4CMPR                     = 0x0000;
    EvbRegs.EVBIMRB.bit.T4PINT         = 1;
    EvbRegs.EVBIFRB.bit.T4PINT         = 1;
    EvbRegs.T4CNT                      = 0x0000;
    EvbRegs.T4CON.all                  = 0x1742;
    EvbRegs.GPTCONB.bit.T4TOADC  = 2;
}

interrupt void eva_timer1_isr(void)
{
   EvaTimer1InterruptCount++;
   EvaRegs.EVAIMRA.bit.T1PINT          = 1;
   EvaRegs.EVAIFRA.all                 = BIT7;
   PieCtrlRegs.PIEACK.all              = PIEACK_GROUP2;
}
```

```
interrupt void eva_timer2_isr(void)
{
  EvaTimer2InterruptCount++;
  EvaRegs.EVAIMRB.bit.T2PINT  = 1;
  EvaRegs.EVAIFRB.all  = BIT0;

  PieCtrlRegs.PIEACK.all  = PIEACK_GROUP3;
}

interrupt void evb_timer3_isr(void)
{
  EvbTimer3InterruptCount++;
 EvbRegs.EVBIFRA.all  = BIT7;

PieCtrlRegs.PIEACK.all  = PIEACK_GROUP4;

}

interrupt void evb_timer4_isr(void)
{
   EvbTimer4InterruptCount++;
   EvbRegs.EVBIFRB.all  = BIT0;

   PieCtrlRegs.PIEACK.all  = PIEACK_GROUP5;

}


//===========================================================================
// No more.
//===========================================================================
```

Result:

Watch Variables:

EvaTimer1InterruptCount;

EvaTimer2InterruptCount;

EvbTimer3InterruptCount;

EvbTimer4InterruptCount;

- *Observe that after debugging the program, it sets up EVA Timer 1, EVA Timer 2, EVB Timer 3 and EVB Timer 4 to fire an interrupt on a period overflow.*
- *Also, a count is kept each time each interrupt passes through the interrupt service routine.*
- *EVA Timer 1 has the shortest period while EVB Timer4 has the longest period.*

Program No 10: *Generation of 1 kHz PWM Pulses at 50% and 75% Duty cycles*

Date:

Objective:

To run a program that can generates PWM pulses at 1 kHz for different duty cycles.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable
- CRO

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"
extern void InitSysCtrl(void);
void Gpio_select(void);
void Setup_ePWM1(void);
void main(void)
{
        InitSysCtrl();
        EALLOW;
        SysCtrlRegs.WDCR= 0x00EF;
        EDIS;
        Gpio_select();
        Setup_ePWM1();
        ERTM;
        while(1);
        }

        void Gpio_select(void)
        {
                EALLOW;
                GpioCtrlRegs.GPAMUX1.all                = 0;
                GpioCtrlRegs.GPAMUX1.bit.GPIO0          = 1;
                GpioCtrlRegs.GPAMUX1.bit.GPIO1          = 1;
                GpioCtrlRegs.GPAMUX2.all                = 0;
                GpioCtrlRegs.GPBMUX1.all                = 0;
                GpioCtrlRegs.GPADIR.all                 = 0;
                GpioCtrlRegs.GPBDIR.all                 = 0;
                EDIS;
        }

        void Setup_ePWM1(void)
        {
                EPwm1Regs.TBCTL.bit.CLKDIV              =  0;
                EPwm1Regs.TBCTL.bit.HSPCLKDIV           = 1;
                EPwm1Regs.TBCTL.bit.CTRMODE             = 2;
                EPwm1Regs.AQCTLA.all                    = 0x0060;
                EPwm1Regs.AQCTLB.all                    = 0x0600;
                EPwm1Regs.TBPRD                         = 37500;
                EPwm1Regs.CMPA.half.CMPA                = EPwm1Regs.TBPRD / 2;
                EPwm1Regs.CMPB                          = EPwm1Regs.TBPRD / 2;
        }

//============================================================
// No more.
//============================================================
```

Result:

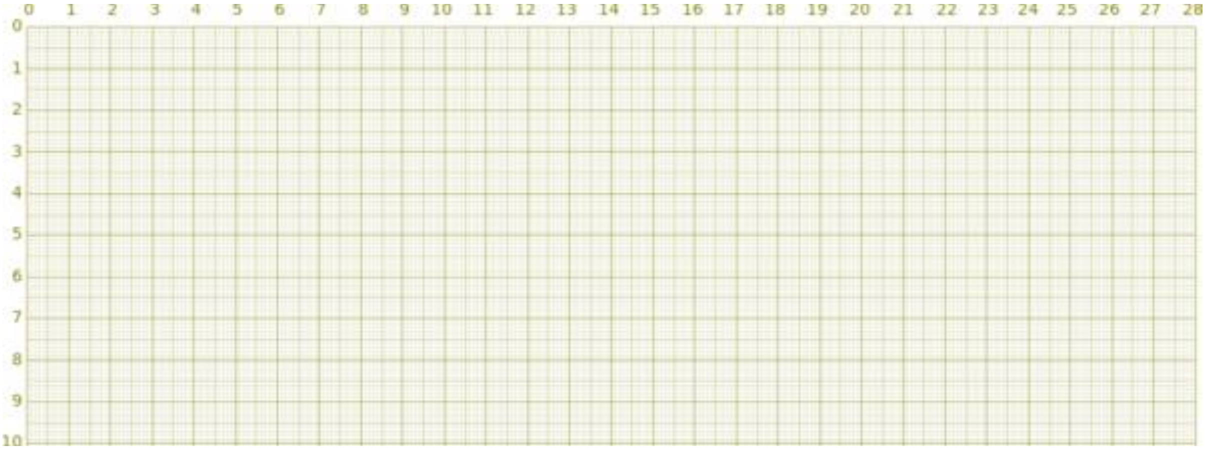
Watch variables

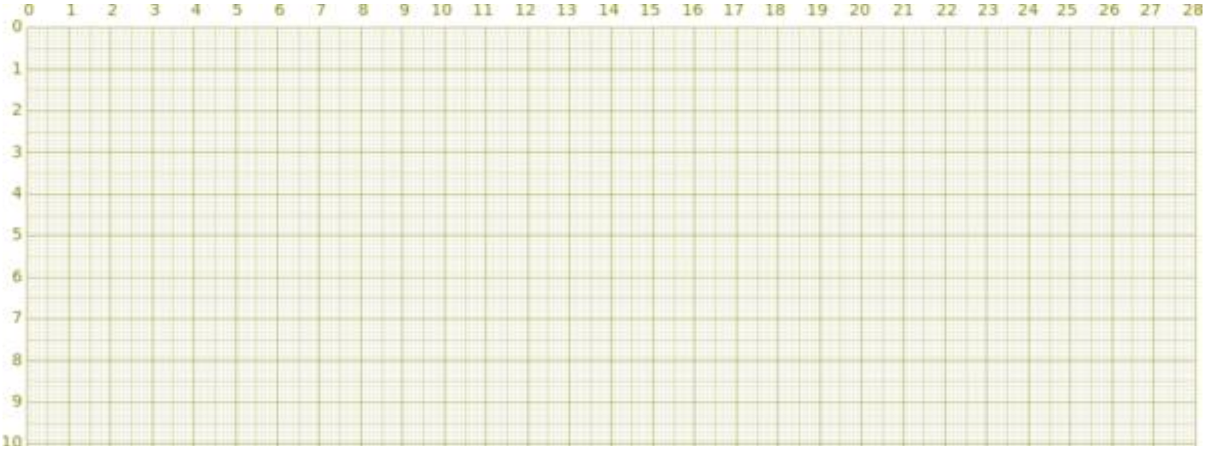
GpioCtrlRegs.GPAMUX1.bit.GPIO0

GpioCtrlRegs.GPAMUX1.bit.GPIO1

By connecting the GPIO 0 and GPIO 1 pins to the CRO, PWM pulses can be observed.

Graphs:





Program No 11: *Generation of 5 kHz PWM Pulses at 25% Duty cycle*

Date:

Objective:

To run a program that can generates PWM pulses at 5 kHz for 25% duty cycles.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable
- CRO

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"
extern void InitSysCtrl(void);
void Gpio_select(void);
void Setup_ePWM1(void);

void main(void)
{

        InitSysCtrl();
        EALLOW;
        SysCtrlRegs.WDCR= 0x00EF;
        EDIS;
        Gpio_select();
```

```
            Setup_ePWM1();
            ERTM;
            while(1);
    }

    void Gpio_select(void)
    {
            EALLOW;
            GpioCtrlRegs.GPAMUX1.all                = 0;
            GpioCtrlRegs.GPAMUX1.bit.GPIO0          = 1;
            GpioCtrlRegs.GPAMUX1.bit.GPIO1          = 1;
            GpioCtrlRegs.GPAMUX2.all                = 0;
            GpioCtrlRegs.GPBMUX1.all                = 0;
            GpioCtrlRegs.GPADIR.all                 = 0;
            GpioCtrlRegs.GPBDIR.all                 = 0;
            EDIS;
    }

    void Setup_ePWM1(void)
    {
            EPwm1Regs.TBCTL.bit.CLKDIV              = 0;
            EPwm1Regs.TBCTL.bit.HSPCLKDIV           = 1;
            EPwm1Regs.TBCTL.bit.CTRMODE             = 2;
            EPwm1Regs.AQCTLA.all                    = 0x0060;
            EPwm1Regs.AQCTLB.all                    = 0x0090;
            EPwm1Regs.TBPRD                         = 750;
            EPwm1Regs.CMPA.half.CMPA                = 1250;
    }

//============================================================
// No more.
//============================================================
```

Result:

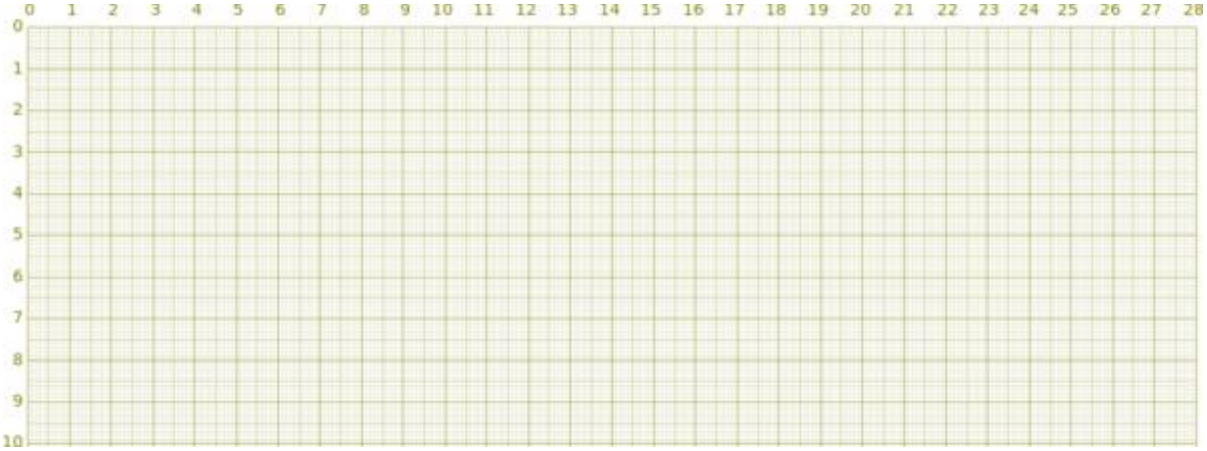
Watch variables

GpioCtrlRegs.GPAMUX1.bit.GPIO0

GpioCtrlRegs.GPAMUX1.bit.GPIO1

By connecting the GPIO 0 and GPIO 1 pins to the CRO, PWM pulses can be observed.

Graph:



Program No 12: *Generation of simple PWM pulses at 10 kHz*

Date:

Objective:

To run a program that can generates PWM pulses at 5 kHz for 25% duty cycles.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable
- CRO

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"
extern void InitSysCtrl(void);

void Gpio_select(void);
void Setup_ePWM1A(void);

void main(void)
{
        InitSysCtrl();
        EALLOW;
        SysCtrlRegs.WDCR= 0x00EF;
        EDIS;
        DINT;
        Gpio_select();
```

```
		Setup_ePWM1A();
		ERTM;
		while(1);
	}

	void Gpio_select(void)
	{
		EALLOW;
		GpioCtrlRegs.GPAMUX1.all		= 0;
		GpioCtrlRegs.GPAMUX1.bit.GPIO0		= 1;
		GpioCtrlRegs.GPAMUX2.all		= 0;
		GpioCtrlRegs.GPBMUX1.all		= 0;
		GpioCtrlRegs.GPADIR.all			= 0;
		GpioCtrlRegs.GPBDIR.all			= 0;
		EDIS;
	}

	void Setup_ePWM1A(void)
	{
		EPwm1Regs.TBCTL.bit.CLKDIV		= 0;
		EPwm1Regs.TBCTL.bit.HSPCLKDIV		= 1;
		EPwm1Regs.TBCTL.bit.CTRMODE		= 2;
		EPwm1Regs.AQCTLA.all			= 0x0006;
		EPwm1Regs.TBPRD				= 1500;
	}

//==============================================================
// No more.
//==============================================================
```

Result:

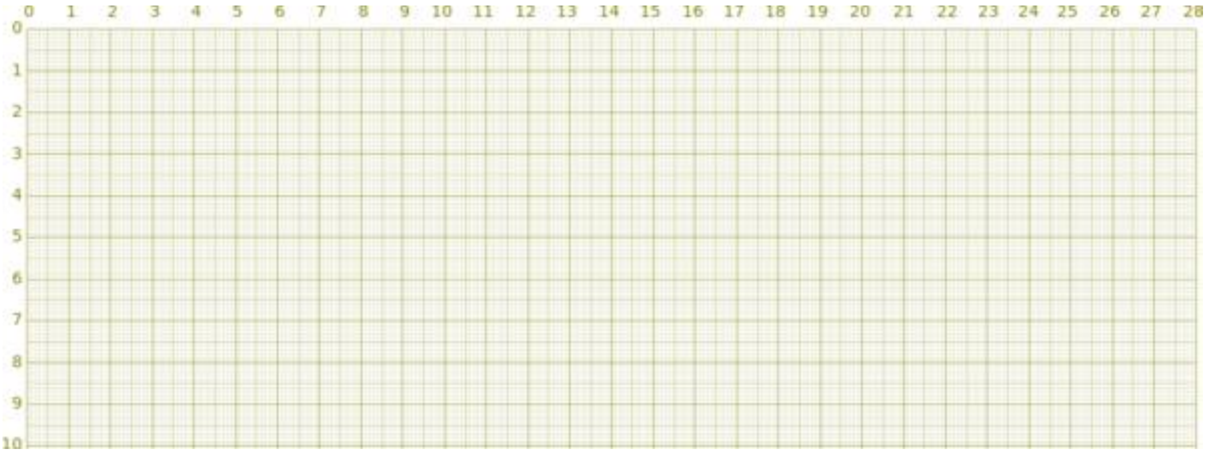
Watch variables

GpioCtrlRegs.GPAMUX1.bit.GPIO0

GpioCtrlRegs.GPAMUX1.bit.GPIO1

By connecting the GPIO 0 and GPIO 1 pins to the CRO, PWM pulses can be observed.

Graph:



Program No 13: *Generation of ePWM pulses with a dead-band (delay routine)*

Date:

Objective:

To run a program that can generates ePWM pulses with a dead region.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable
- CRO

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"
void InitEPwm1Example(void);
interrupt void epwm1_isr(void);

Uint32  EPwm1TimerIntCount;
Uint16  EPwm1_DB_Direction;

#define EPWM1_MAX_DB   0x03FF
#define EPWM1_MIN_DB   0
#define DB_UP   1
#define DB_DOWN  0
```

```
void main(void)
{
   InitSysCtrl();
   InitEPwm1Gpio();
   DINT;
   InitPieCtrl();

   IER = 0x0000;
   IFR = 0x0000;

   InitPieVectTable();

   EALLOW;
   PieVectTable.EPWM1_INT  = &epwm1_isr;

   EDIS;
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC  = 0;
   EDIS;

   InitEPwm1Example();

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC  = 1;
   EDIS;

   EPwm1TimerIntCount  = 0;
   IER  |= M_INT3;
   PieCtrlRegs.PIEIER3.bit.INTx1   = 1;

   EINT;
   ERTM;
```

```
for(;;)
  {
     asm("        NOP");
  }

}

interrupt void epwm1_isr(void)
{
  if(EPwm1_DB_Direction  == DB_UP)
  {
     if(EPwm1Regs.DBFED  < EPWM1_MAX_DB)
     {
       EPwm1Regs.DBFED++;
       EPwm1Regs.DBRED++;
     }
     else
     {
       EPwm1_DB_Direction  = DB_DOWN;
       EPwm1Regs.DBFED--;
       EPwm1Regs.DBRED--;
     }
  }
  else
  {
     if(EPwm1Regs.DBFED  == EPWM1_MIN_DB)
     {
       EPwm1_DB_Direction  = DB_UP;
       EPwm1Regs.DBFED++;
       EPwm1Regs.DBRED++;
     }
     else
     {
      EPwm1Regs.DBFED--;
      EPwm1Regs.DBRED--;
    }
  }
  EPwm1TimerIntCount++;
  EPwm1Regs.ETCLR.bit.INT      = 1;
  PieCtrlRegs.PIEACK.all       = PIEACK_GROUP3;

}

void InitEPwm1Example()
{

  EPwm1Regs.TBPRD              = 6000;
  EPwm1Regs.TBPHS.half.TBPHS   = 0x0000;
  EPwm1Regs.TBCTR              = 0x0000;

  EPwm1Regs.TBCTL.bit.CTRMODE        = TB_COUNT_UPDOWN;
  EPwm1Regs.TBCTL.bit.PHSEN          = TB_DISABLE;
  EPwm1Regs.TBCTL.bit.HSPCLKDIV      = TB_DIV4;
  EPwm1Regs.TBCTL.bit.CLKDIV         = TB_DIV4;

  EPwm1Regs.CMPCTL.bit.SHDWAMODE            = CC_SHADOW;
  EPwm1Regs.CMPCTL.bit.SHDWBMODE            = CC_SHADOW;
  EPwm1Regs.CMPCTL.bit.LOADAMODE            = CC_CTR_ZERO;
  EPwm1Regs.CMPCTL.bit.LOADBMODE            = CC_CTR_ZERO;


  EPwm1Regs.CMPA.half.CMPA     = 3000;
  EPwm1Regs.AQCTLA.bit.CAU     = AQ_SET;
  EPwm1Regs.AQCTLA.bit.CAD     = AQ_CLEAR;
  EPwm1Regs.AQCTLB.bit.CAU     = AQ_CLEAR;
  EPwm1Regs.AQCTLB.bit.CAD     = AQ_SET;
  // Active Low PWMs - Setup Deadband
  EPwm1Regs.DBCTL.bit.OUT_MODE       = DB_FULL_ENABLE;
  EPwm1Regs.DBCTL.bit.POLSEL         = DB_ACTV_LO;
  EPwm1Regs.DBCTL.bit.IN_MODE        = DBA_ALL;
  EPwm1Regs.DBRED                    = EPWM1_MIN_DB;
  EPwm1Regs.DBFED                    = EPWM1_MIN_DB;
  EPwm1_DB_Direction                 = DB_UP;

  // Interrupt where we will change the Deadband
  EPwm1Regs.ETSEL.bit.INTSEL         = ET_CTR_ZERO;
  EPwm1Regs.ETSEL.bit.INTEN          = 1;
  EPwm1Regs.ETPS.bit.INTPRD          = ET_3RD;


}


//===========================================================================
// No more.
//===========================================================================
```

Result:

Watch variables

GpioCtrlRegs.GPAMUX1.bit.GPIO0

GpioCtrlRegs.GPAMUX1.bit.GPIO1

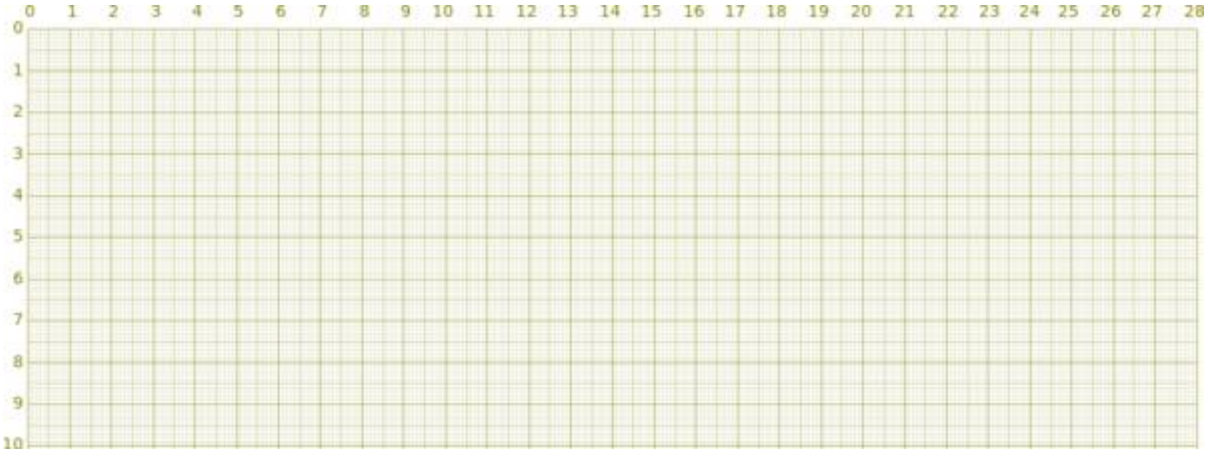
EPwm1Regs.TBCTL.bit.CTRMODE

EPwm1Regs.DBCTL.bit.OUT_MODE

EPwm1Regs.ETSEL.bit.INTSEL

By connecting the GPIO 0 and GPIO 1 pins to the CRO, PWM pulses with dead-band can be observed.

Graph:



Program No 14: *An example to run a program in FLASH memory*

Date:

Objective:

To run a program that can run the program in FLASH memory.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable
- CRO

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"

#define PWM1_INT_ENABLE  1
#define PWM2_INT_ENABLE  1
#define PWM3_INT_ENABLE  1

// Configure the period for each timer
#define PWM1_TIMER_TBPRD   0x1FFF
#define PWM2_TIMER_TBPRD   0x1FFF
#define PWM3_TIMER_TBPRD   0x1FFF

#define DELAY 1000000L
```

```
#pragma CODE_SECTION(EPwm1_timer_isr, "ramfuncs");
#pragma CODE_SECTION(EPwm2_timer_isr, "ramfuncs");

interrupt void EPwm1_timer_isr(void);
interrupt void EPwm2_timer_isr(void);
interrupt void EPwm3_timer_isr(void);
void InitEPwmTimer(void);

Uint32  EPwm1TimerIntCount;
Uint32  EPwm2TimerIntCount;
Uint32  EPwm3TimerIntCount;
Uint32  LoopCount;

extern Uint16 RamfuncsLoadStart;
extern Uint16 RamfuncsLoadEnd;
extern Uint16 RamfuncsRunStart;

void main(void)
{

  InitSysCtrl();

  DINT;

  InitPieCtrl();

  IER = 0x0000;
  IFR = 0x0000;

  InitPieVectTable();

  EALLOW;  // This is needed to write to EALLOW protected registers
  PieVectTable.EPWM1_INT  = &EPwm1_timer_isr;
  PieVectTable.EPWM2_INT  = &EPwm2_timer_isr;
```

```
    PieVectTable.EPWM3_INT  = &EPwm3_timer_isr;
    EDIS;
    InitEPwmTimer();
    EPwm2_timer_isr()
    MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);
    InitFlash();

    EPwm1TimerIntCount = 0;
    EPwm2TimerIntCount = 0;
    EPwm3TimerIntCount = 0;
    LoopCount = 0;

    IER |= M_INT3;

    PieCtrlRegs.PIEIER3.bit.INTx1  = PWM1_INT_ENABLE;
    PieCtrlRegs.PIEIER3.bit.INTx2  = PWM2_INT_ENABLE;
    PieCtrlRegs.PIEIER3.bit.INTx3  = PWM3_INT_ENABLE;

    EINT;   // Enable Global interrupt INTM
    ERTM;   // Enable Global realtime interrupt DBGM

    EALLOW;
    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;
    GpioCtrlRegs.GPBDIR.bit.GPIO34  = 1;
    EDIS;

    for(;;)
    {
       DELAY_US(DELAY);
       LoopCount++;
       GpioDataRegs.GPBTOGGLE.bit.GPIO34  = 1;
    }

  }
```

```
void InitEPwmTimer()
{

  EALLOW;
  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC  = 0;
  EDIS;

  InitEPwm1Gpio();
  InitEPwm2Gpio();
  InitEPwm3Gpio();

  // Setup Sync
  EPwm1Regs.TBCTL.bit.SYNCOSEL  = TB_SYNC_IN;   // Pass through
  EPwm2Regs.TBCTL.bit.SYNCOSEL  = TB_SYNC_IN;   // Pass through
  EPwm3Regs.TBCTL.bit.SYNCOSEL  = TB_SYNC_IN;   // Pass through

  // Allow each timer to be sync'ed

  EPwm1Regs.TBCTL.bit.PHSEN  = TB_ENABLE;
  EPwm2Regs.TBCTL.bit.PHSEN  = TB_ENABLE;
  EPwm3Regs.TBCTL.bit.PHSEN  = TB_ENABLE;

  EPwm1Regs.TBPHS.half.TBPHS  = 100;
  EPwm2Regs.TBPHS.half.TBPHS  = 200;
  EPwm3Regs.TBPHS.half.TBPHS  = 300;

  EPwm1Regs.TBPRD = PWM1_TIMER_TBPRD;
  EPwm1Regs.TBCTL.bit.CTRMODE  = TB_COUNT_UP;     // Count up
  EPwm1Regs.ETSEL.bit.INTSEL  = ET_CTR_ZERO;      // Select INT on Zero event
  EPwm1Regs.ETSEL.bit.INTEN  = PWM1_INT_ENABLE;   // Enable INT
  EPwm1Regs.ETPS.bit.INTPRD  = ET_1ST;           // Generate INT on 1st event
  EPwm2Regs.TBPRD = PWM2_TIMER_TBPRD;
  EPwm2Regs.TBCTL.bit.CTRMODE  = TB_COUNT_UP;      // Count up
  EPwm2Regs.ETSEL.bit.INTSEL  = ET_CTR_ZERO;       // Enable INT on Zero event
  EPwm2Regs.ETSEL.bit.INTEN  = PWM2_INT_ENABLE;    // Enable INT
  EPwm2Regs.ETPS.bit.INTPRD  = ET_2ND;             // Generate INT on 2nd event

  EPwm3Regs.TBPRD = PWM3_TIMER_TBPRD;
  EPwm3Regs.TBCTL.bit.CTRMODE  = TB_COUNT_UP;      // Count up
  EPwm3Regs.ETSEL.bit.INTSEL  = ET_CTR_ZERO;       // Enable INT on Zero event
  EPwm3Regs.ETSEL.bit.INTEN  = PWM3_INT_ENABLE;    // Enable INT
  EPwm3Regs.ETPS.bit.INTPRD  = ET_3RD;             // Generate INT on 3rd event

  EPwm1Regs.CMPA.half.CMPA  = PWM1_TIMER_TBPRD/2;
  EPwm1Regs.AQCTLA.bit.PRD  = AQ_SET;
  EPwm1Regs.AQCTLA.bit.CAU  = AQ_CLEAR;
  EPwm1Regs.AQCTLB.bit.PRD  = AQ_SET;
  EPwm1Regs.AQCTLB.bit.CAU  = AQ_CLEAR;

  EPwm2Regs.CMPA.half.CMPA  = PWM2_TIMER_TBPRD/2;
  EPwm2Regs.AQCTLA.bit.PRD  = AQ_SET;
  EPwm2Regs.AQCTLA.bit.CAU  = AQ_CLEAR;
  EPwm2Regs.AQCTLB.bit.PRD  = AQ_SET;
  EPwm2Regs.AQCTLB.bit.CAU  = AQ_CLEAR;

  EPwm3Regs.CMPA.half.CMPA  = PWM3_TIMER_TBPRD/2;
  EPwm3Regs.AQCTLA.bit.PRD  = AQ_SET;
  EPwm3Regs.AQCTLA.bit.CAU  = AQ_CLEAR;
  EPwm3Regs.AQCTLB.bit.PRD  = AQ_SET;
  EPwm3Regs.AQCTLB.bit.CAU  = AQ_CLEAR;

  EALLOW;
  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC  = 1;         // Start all the timers synced
  EDIS;
}
```

```
interrupt void EPwm1_timer_isr(void)
{
  FlashRegs.FPWR.bit.PWR = FLASH_SLEEP;

  EPwm1TimerIntCount++;
  EPwm1Regs.ETCLR.bit.INT = 1;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

// This ISR MUST be executed from RAM as it will put the Flash into Standby
interrupt void EPwm2_timer_isr(void)
{
  EPwm2TimerIntCount++;
  FlashRegs.FPWR.bit.PWR = FLASH_STANDBY;
  EPwm2Regs.ETCLR.bit.INT = 1;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

interrupt void EPwm3_timer_isr(void)
{
  Uint16 i;
  EPwm3TimerIntCount++;
   for(i = 1; i < 0x01FF; i++) {}
  EPwm3Regs.ETCLR.bit.INT = 1;
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

//=================================================================
// No more.
//=================================================================
```

Result:

Watch variables

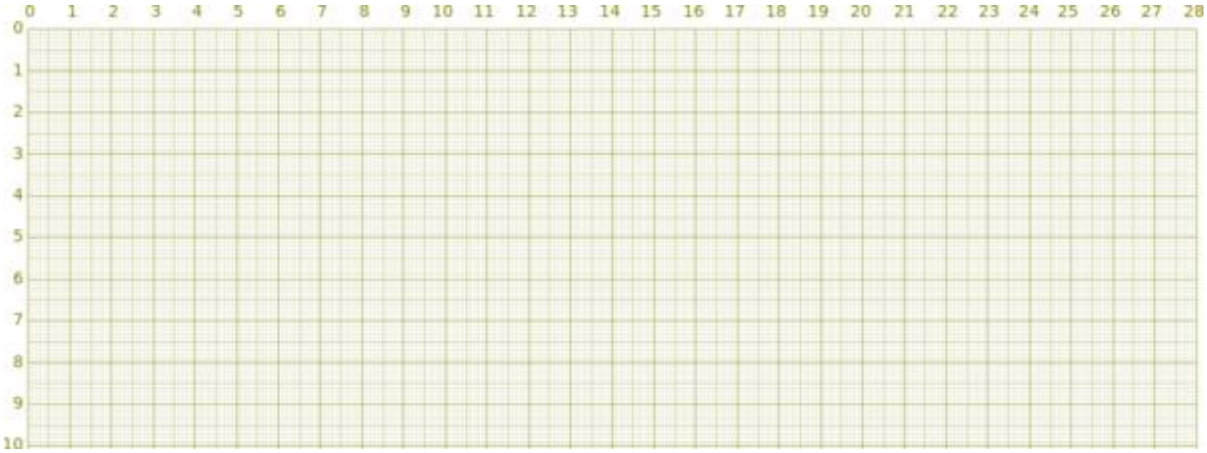
EPwm3TimerIntCount++;

EPwm2TimerIntCount++;

EPwm3TimerIntCount++;

After loading the program in to the Launchpad, by connecting the GPIO pins to the CRO, the output can be seen on the CRO.

Graph:



Program No 15: *Interfacing an LED*

Date:

Objective:

To run a program that can flash the LED with the delay.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP281x_Device.h"
#include <stdio.h>
void Delay_1ms(long);
void main(void)
{
EALLOW;
SysCtrlRegs.WDCR             = 0x0068;
SysCtrlRegs.SCSR             = 0;
SysCtrlRegs.PLLCR.bit.DIV    = 10;
SysCtrlRegs.HISPCP.all       = 0x1;
SysCtrlRegs.LOSPCP.all       = 0x2;
GpioMuxRegs.GPAMUX.all       = 0x0;
GpioMuxRegs.GPBMUX.all       = 0x0;
GpioMuxRegs.GPADIR.all       = 0x0;
GpioMuxRegs.GPBDIR.all       = 0x00FF;
```

```
        EDIS;
        while(1)
                {
                GpioDataRegs.GPBDAT.all  = 0xFF;

                Delay_1ms(1000);

                GpioDataRegs.GPBDAT.all  = 0x0;

                Delay_1ms(1000);
                }
        }
        void Delay_1ms(long  end)
        {
                 long i;
                for (i = 0; i <(9000 * end); i++);
        }



//===========================================================================
// No more.
//===========================================================================
```

Result:

Watch variables

GpioDataRegs.GPADAT.all

GpioDataRegs.GPBDAT.all

Observe the LED flashes with the delay of 1000 ms.

Program No 16: *Generation of SVPWM pulses for an Inverter operation*

Date:

Objective:

To run a program that can generates the SVPWM pulses to gating the Inverter switches.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable
- CRO

Software:

- Code Composer Studio 6.0
- MATLAB/Simulink
- C2000 processor supporting package
- Windows 8 OS.

Program:

**The following program has been generated through MATLAB/Simulink interfacing for the F28027-Launchpad using support package for C2000 processor.*

```
#include "SVPWM_Pulses.h"
#include "rtwtypes.h"
#include "rt_nonfinite.h"
#include "SVPWM_Pulses_private.h"
#include "c2000_main.h"
#include "F2802x_Device.h"
#include "f2802x_examples.h"
```

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

void init_board(void);
void enable_interrupts(void);
extern Uint16 RamfuncsLoadEnd;
void config_schedulerTimer(void);
void disable_interrupts(void);
volatile int IsrOverrun = 0;
static boolean_T OverrunFlag = 0;

void rt_OneStep(void)
{
  if (OverrunFlag++) {
    IsrOverrun = 1;
    OverrunFlag--;
    return;
  }

  asm(" SETC INTM");
  PieCtrlRegs.PIEIER1.all  |= (1 << 6);
  asm(" CLRC INTM");
  SVPWM_Pulses_step();

  /* Get model outputs here */
  asm(" SETC INTM");
  PieCtrlRegs.PIEIER1.all  &= ~(1 << 6);
  asm(" RPT #5 || NOP");
  IFR &= 0xFFFE;
  PieCtrlRegs.PIEACK.all  = 0x1;
  asm(" CLRC INTM");
  OverrunFlag--;
}
```

```
                void main(void)
                {
                  volatile boolean_T  noErr;

                  // Copy InitFlash function code and Flash setup code to RAM

        memcpy(&RamfuncsRunStart,&RamfuncsLoadStart,(Uint32)(&RamfuncsLoadEnd-
                      &RamfuncsLoadStart));

                  // Call Flash Initialization to setup flash waitstates
                  // This function must reside in RAM
                  InitFlash();
                  init_board();
                  rtmSetErrorStatus(SVPWM_Pulses_M, 0);
                  SVPWM_Pulses_initialize();
                  config_schedulerTimer();
                  noErr =
                    rtmGetErrorStatus(SVPWM_Pulses_M) == (NULL);
                  enable_interrupts();
                  while (noErr ) {
                    noErr =
                      rtmGetErrorStatus(SVPWM_Pulses_M) == (NULL);
                  }

                  SVPWM_Pulses_terminate();
                  disable_interrupts();
                }

//============================================================================
// No more.
//============================================================================
```

Result:

Watch variables

GpioDataRegs.GPADAT.all

GpioDataRegs.GPBDAT.all

We can observe the SVPWM waveforms by connecting GPIO pins to the CRO

Graph: