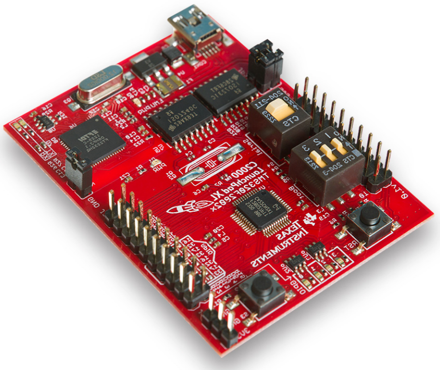
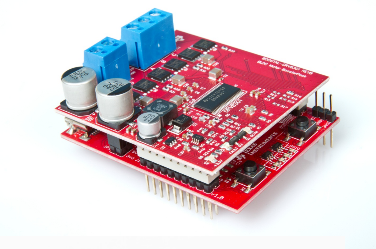
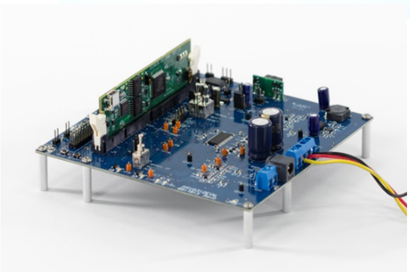
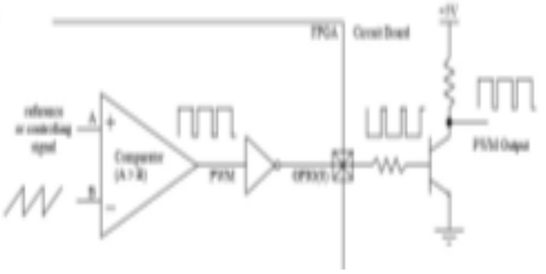
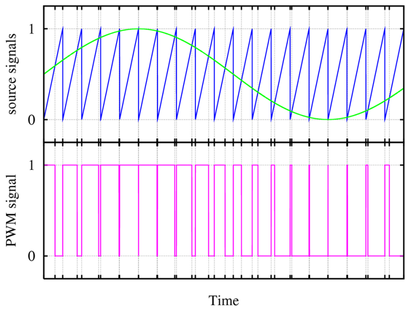
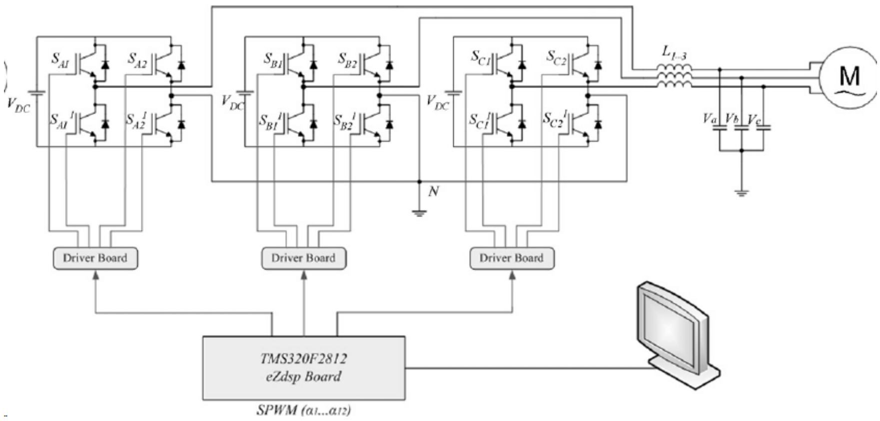












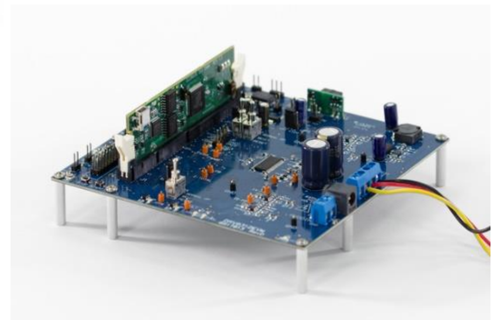
DSP and Microcontroller Lab

M.Tech (PE) I Year - II Semester

Gokaraju Rangaraju Institute of Engineering & Technology

(Autonomous)

Department of Electrical & Electronics Engineering



DSP and Microcontroller Lab

M.Tech (PE) I Year - II Semester

by

Dr. Vinay Kumar Awaar

Associate Professor

DEPARTMENT OF ELECTRICAL & ELECTORNICS ENGINEERING

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING & TECHNOLOGY

Nizampet Road, Bachupally, Kukatpally, Hyderabad-500090

Telangana State, India. +91-040- 7207344440, 7207714441, 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 and Microcontroller 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.		Introduction to Code Composer Studio- An example: Watchdog with CPU Timer interrupts.		
2.		Square of a given number using for loop		
3.		Factorial of a given number using for loop		
4.		Configuring GPIO pins of TMS320F28027 processor for flashing onboard LEDs		
5.		Configuring ADC pins for real time data exchange.		
6.		Generation of gate signals for DC-DC boost converter.		
7.		Generation of gate signals for DC-AC 1-phase full bridge inverter.		
8.		Generation of simple PWM pulses at 10 kHz		
9.		Generation of ePWM pulses with a dead-band (delay routine)		
10.		Generation of gate signals for 3-phase voltage source inverter.		
11.		An example to run a program in FLASH memory		
12.		Speed control of DC motor by interfacing embedded coder with MATLAB Simulink.		
13.		Speed control of BLDC motor with a velocity control mode.		
14.		Speed control of an induction motor with v/f control mode.		
15.		Speed control of PMSM motor FOC control mode.		

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 I^2C, 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.

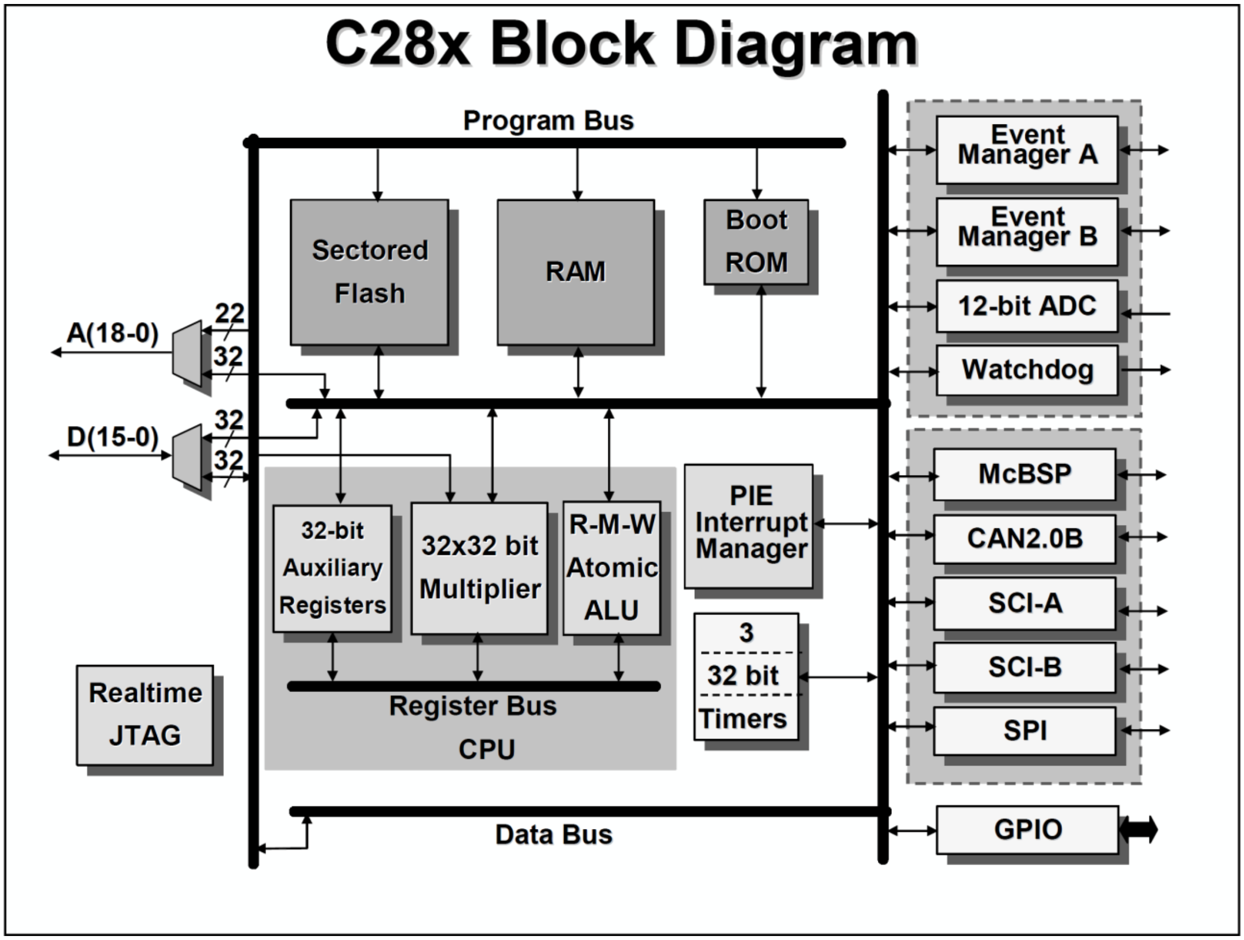
- TMS320C55x generation - fixed-point, runs C54x code but adds more internal parallelism (another ALU, dual MAC, more memory bandwidth) and registers, while supporting much lower power operation.
- Today, most C55x DSPs are sold as discrete chips
- OMAP1 chips combine an ARM9 (ARMv5TEJ) with a C55x series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

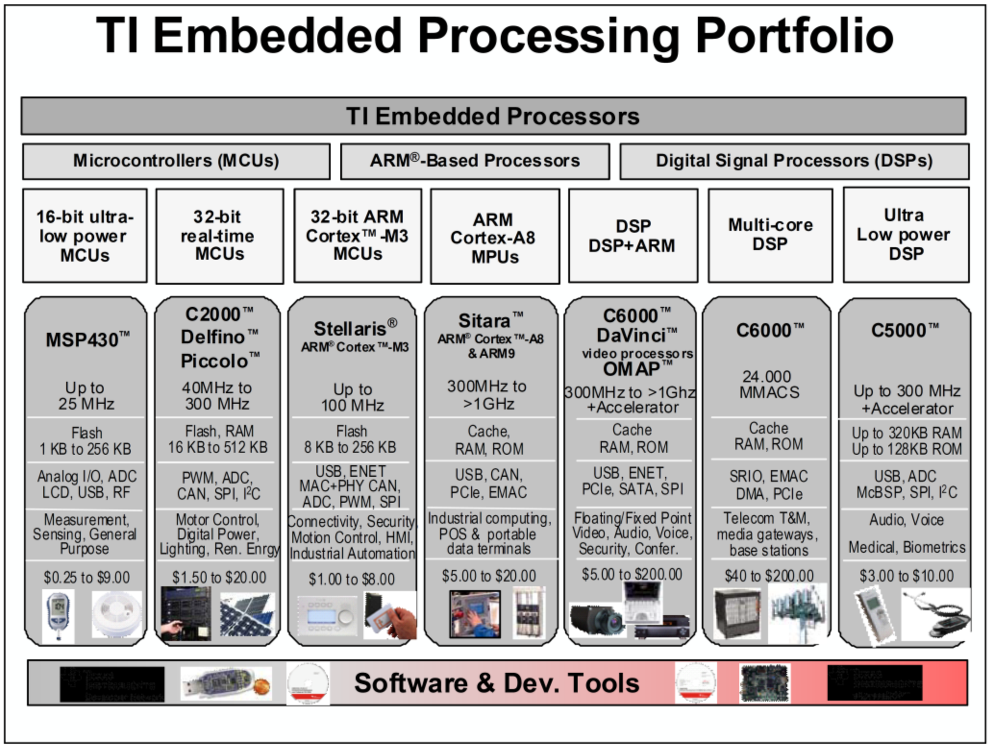
What is the TMS320C28x?

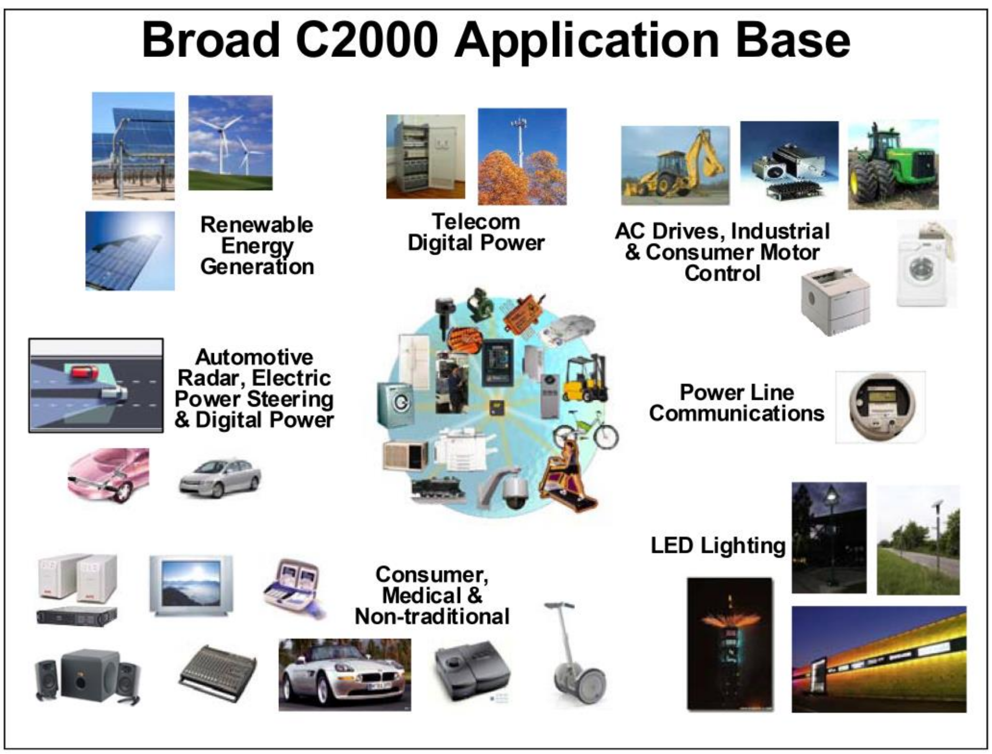
The TMS320C28x is a 32-bit fixed point DSP that specializes in high performance control applications such as, robotics, industrial automation, mass storage devices, lighting, optical networking, power supplies, and other control applications needing a single processor to solve a high-performance application.



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™
Up to 25 MHz
Flash 1 KB to 256 KB
Analog I/O, ADC LCD, USB, RF
Measurement, Sensing, General Purpose
$0.25 to $9.00
C2000™ Delfino™ Piccolo™
40MHz to 300 MHz
Flash, RAM 16 KB to 512 KB
PWM, ADC, CAN, SPI, I²C
Motor Control, Digital Power, Lighting, Ren. Enrgy
$1.50 to $20.00
Stellaris® ARM® Cortex™-M3
Up to 100 MHz
Flash 8 KB to 256 KB
USB, ENET MAC+PHY CAN, ADC, PWM, SPI
Connectivity, Security, Motion Control, HMI, Industrial Automation
$1.00 to $8.00
Sitara™ ARM® Cortex™-A8 & ARM9
300MHz to >1GHz
Cache, RAM, ROM
USB, CAN, PCIe, EMAC
Industrial computing, POS & portable data terminals
$5.00 to $20.00
C6000™ DaVinci™ video processors OMAP™
300MHz to >1Ghz +Accelerator
Cache RAM, ROM
USB, ENET, PCIe, SATA, SPI
Floating/Fixed Point Video, Audio, Voice, Security, Confer.
$5.00 to $200.00
C6000™
24.000 MMACS
Cache RAM, ROM
SRIO, EMAC DMA, PCIe
Telecom T&M, media gateways, base stations
$40 to $200.00
C5000™
Up to 300 MHz +Accelerator
Up to 320KB RAM Up to 128KB ROM
USB, ADC McBSP, SPI, I²C
Audio, Voice
Medical, Biometrics
$3.00 to $10.00
Software & Dev. Tools

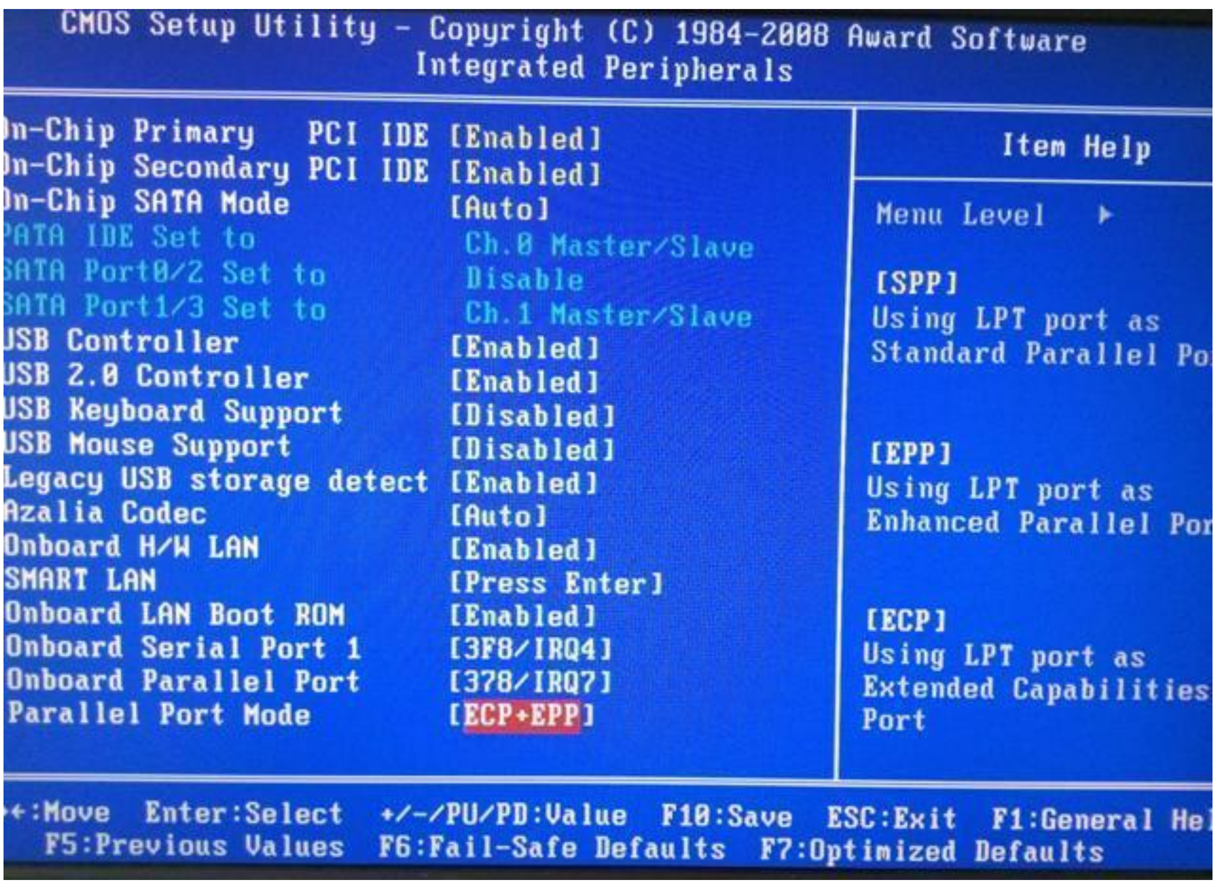

Broad C2000 Application Base
Renewable Energy Generation
Telecom Digital Power
AC Drives, Industrial & Consumer Motor Control
Automotive Radar, Electric Power Steering & Digital Power
Power Line Communications
LED Lighting
Consumer, Medical & Non-traditional

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

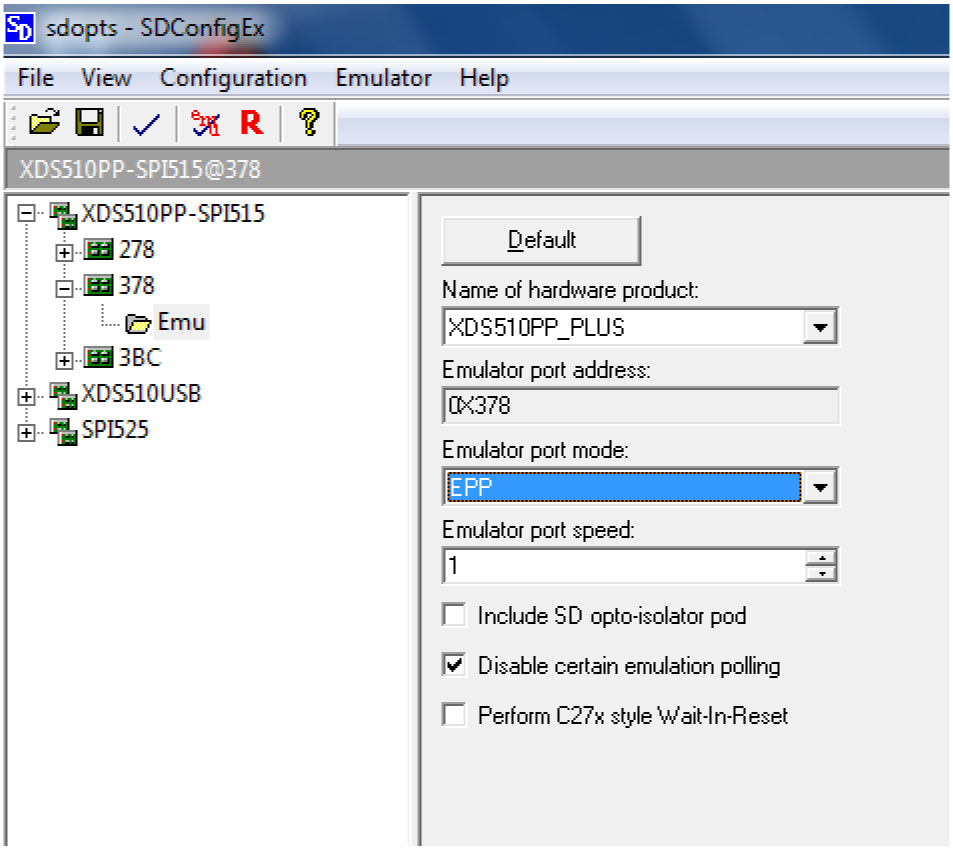


Instructions to install Code Composer Studio V5:

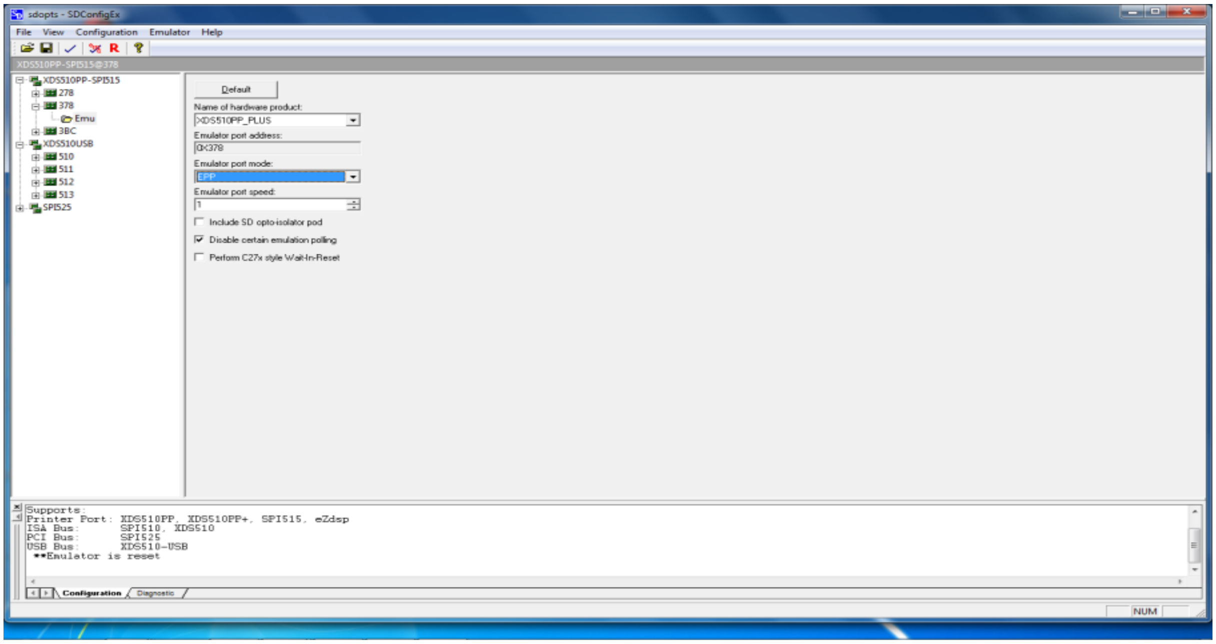
1. Launch the setup from the CCS V5 CD
2. Accept the agreement and NEXT
3. Select the folder to install "default C:\TI" and NEXT
4. Select custom and NEXT
5. Select only C28x 32bit Real time CPU MCU and NEXT
6. In Compiler tools, Select TI C2800 Compiler tools and TI Documentation
7. In device software 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* :

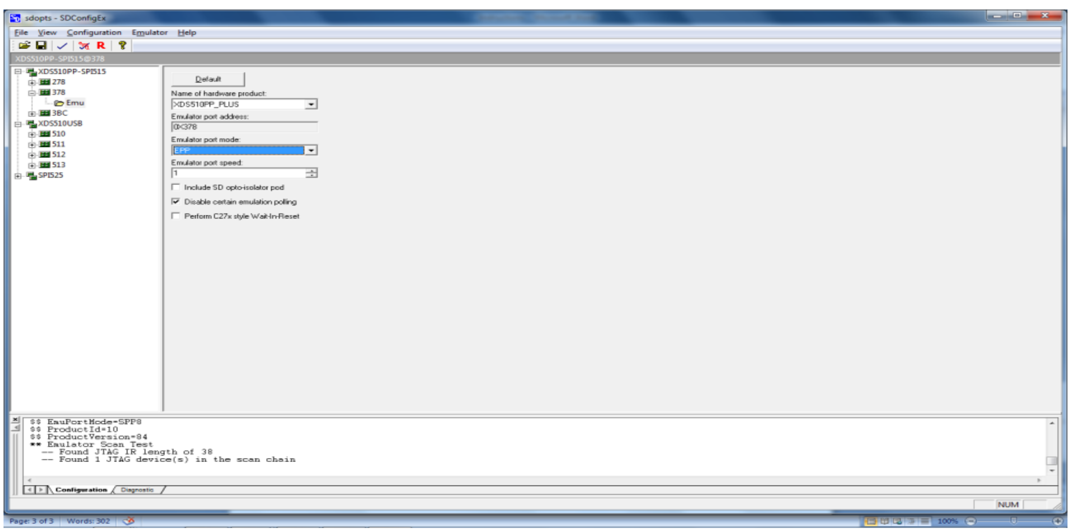
1. Connect the ezDSP with the Computer with parallel port cable and Power on the ezDSP board
2. Open SdConfigEx v5 from the desktop
3. Double Click XDS510PP-SPI515 and select 378.
4. Double click 378 and select emu and Change the Emulator port mode to EPP as shown below



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



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



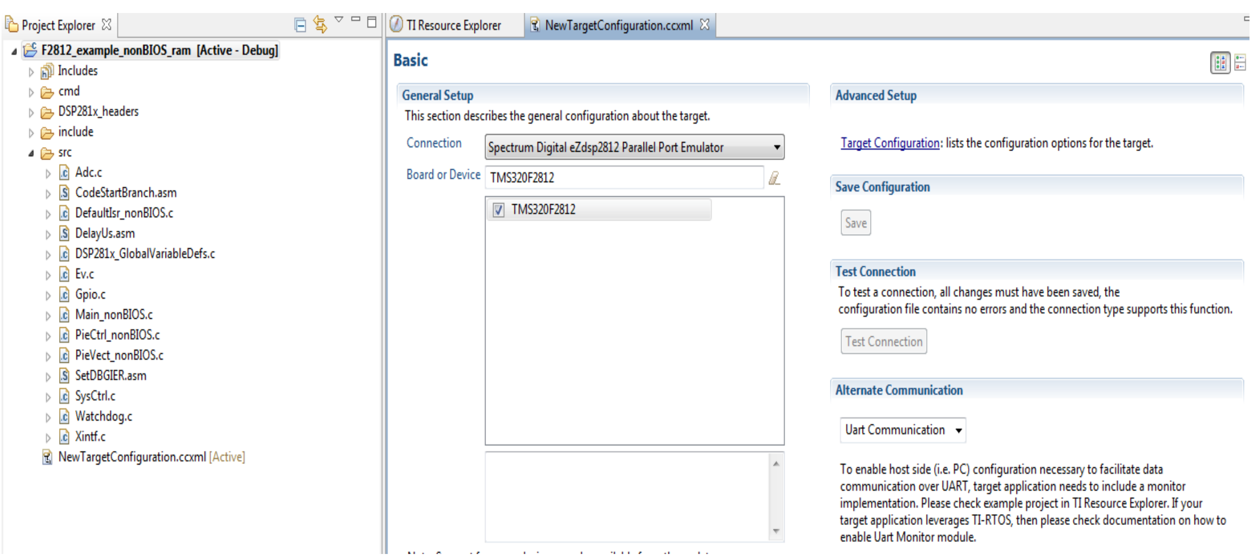
9. If the Emulator rest and JTAG IR length as 38 shows the connection between the system and ezDSP is OK.
10. Now close SD Config.

CCS V6 License Setup.

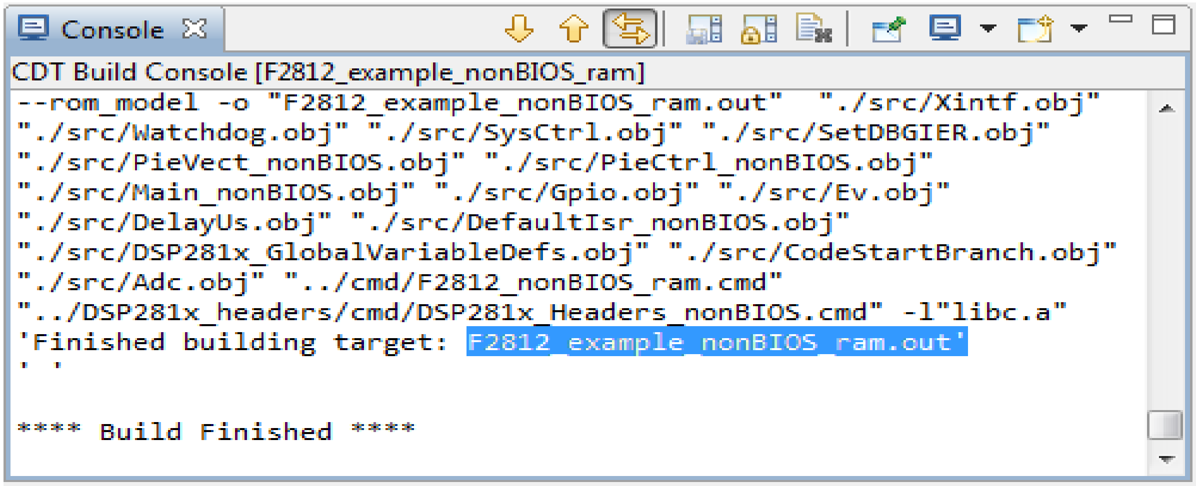
1. Open CCS V6
2. Go to Help Menu->Code Composer Studio License Information
3. Go to Upgrade Tab-> launch License setup
4. Select Evaluate(90days) or Free License (Onboard and XDS 100 Emulators)
5. Press Finish Button.

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



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



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: *Introduction to Code Composer Studio- An example: 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 2: *Square of a given number using 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 3: *Factorial of a given 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 4: *Configuring GPIO pins of TMS320F28027 processor for flashing onboard LEDs*

Date:

Objective:

To run a program that blinks the onboard LEDs.

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
//##########################################################################
#######
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include "f2802x_common/include/adc.h"
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/timer.h"
#include "f2802x_common/include/wdog.h"
```

```
// Prototype statements for functions found within this file.
interrupt void cpu_timer0_isr(void);

uint16_t interruptCount = 0;

ADC_Handle myAdc;
CLK_Handle myClk;
FLASH_Handle myFlash;
GPIO_Handle myGpio;
PIE_Handle myPie;
TIMER_Handle myTimer;

void main(void)
{
   CPU_Handle myCpu;
   PLL_Handle myPll;
   WDOG_Handle myWDog;

   // Initialize all the handles needed for this application
   myAdc = ADC_init((void *)ADC_BASE_ADDR, sizeof(ADC_Obj));
   myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
   myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
   myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));
   myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj));
   myPie = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj));
```

```
    myPll = PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));
    myTimer = TIMER_init((void *)TIMER0_BASE_ADDR, sizeof(TIMER_Obj));
    myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));

    // Perform basic system initialization
    WDOG_disable(myWDog);
    CLK_enableAdcClock(myClk);
    (*Device_cal)();

    //Select the internal oscillator 1 as the clock source
    CLK_setOscSrc(myClk, CLK_OscSrc_Internal);

    // Setup the PLL for x10 /2 which will yield 50Mhz = 10Mhz * 10 / 2
    PLL_setup(myPll, PLL_Multiplier_10, PLL_DivideSelect_ClkIn_by_2);

    // Disable the PIE and all interrupts
    PIE_disable(myPie);
    PIE_disableAllInts(myPie);
    CPU_disableGlobalInts(myCpu);
    CPU_clearIntFlags(myCpu);

    // If running from flash copy RAM only functions to RAM
#ifdef _FLASH
    memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
#endif
```

```
    // Setup a debug vector table and enable the PIE
    PIE_setDebugIntVectorTable(myPie);
    PIE_enable(myPie);

    // Register interrupt handlers in the PIE vector table
    PIE_registerPieIntHandler(myPie, PIE_GroupNumber_1, PIE_SubGroupNumber_7,
(intVec_t)&cpu_timer0_isr);

    // Configure CPU-Timer 0 to interrupt every 500 milliseconds:
    // 60MHz CPU Freq, 50 millisecond Period (in uSeconds)
    //    ConfigCpuTimer(&CpuTimer0, 60, 500000);
    TIMER_stop(myTimer);
    TIMER_setPeriod(myTimer, 50 * 500000);
    TIMER_setPreScaler(myTimer, 0);
    TIMER_reload(myTimer);
    TIMER_setEmulationMode(myTimer,
TIMER_EmulationMode_StopAfterNextDecrement);
    TIMER_enableInt(myTimer);
    TIMER_start(myTimer);

    // Configure GPIO 0-3 as outputs
    GPIO_setMode(myGpio, GPIO_Number_0, GPIO_0_Mode_GeneralPurpose);
    GPIO_setMode(myGpio, GPIO_Number_1, GPIO_0_Mode_GeneralPurpose);
    GPIO_setMode(myGpio, GPIO_Number_2, GPIO_0_Mode_GeneralPurpose);
    GPIO_setMode(myGpio, GPIO_Number_3, GPIO_0_Mode_GeneralPurpose);
```

```
    GPIO_setDirection(myGpio, GPIO_Number_0, GPIO_Direction_Output);
    GPIO_setDirection(myGpio, GPIO_Number_1, GPIO_Direction_Output);
    GPIO_setDirection(myGpio, GPIO_Number_2, GPIO_Direction_Output);
    GPIO_setDirection(myGpio, GPIO_Number_3, GPIO_Direction_Output);

    GPIO_setLow(myGpio, GPIO_Number_0);
    GPIO_setHigh(myGpio, GPIO_Number_1);
    GPIO_setLow(myGpio, GPIO_Number_2);
    GPIO_setHigh(myGpio, GPIO_Number_3);

    // Enable CPU INT1 which is connected to CPU-Timer 0:
    CPU_enableInt(myCpu, CPU_IntNumber_1);

    // Enable TINT0 in the PIE: Group 1 interrupt 7
    PIE_enableTimer0Int(myPie);

    // Enable global Interrupts and higher priority real-time debug events
    CPU_enableGlobalInts(myCpu);
    CPU_enableDebugInt(myCpu);

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

}
```

```
interrupt void cpu_timer0_isr(void)
{
   interruptCount++;

   // Toggle GPIOs
   GPIO_toggle(myGpio, GPIO_Number_0);
   GPIO_toggle(myGpio, GPIO_Number_1);
   GPIO_toggle(myGpio, GPIO_Number_2);
   GPIO_toggle(myGpio, GPIO_Number_3);

   // Acknowledge this interrupt to receive more interrupts from group 1
   PIE_clearInt(myPie, PIE_GroupNumber_1);
}
//===========================================================================
// No more.
//===========================================================================
```

Result:

Watch Variables:

Monitor the ***GPIO0-4 LEDs*** *blink on (for 500 msec) and off (for 500 msec) on the TMS320F28027 Launchpad.*

Program No 5: *Configuring ADC pins for real time data exchange.*

Date:

Objective:

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

Equipment required:

Hardware:

- Laptop
- TMS320F28027 Launchpad
- XDS100v2 USB cable

Software:

- Code composer studio 5.5.0
- Windows 8 OS.

Program:

```
//  Title:  ADC Temperature Sensor Conversion to Degrees Celsius/Degrees Kelvin


#include "DSP28x_Project.h"     // DSP28x Headerfile


#include "f2802x_common/include/adc.h"

#include "f2802x_common/include/clk.h"

#include "f2802x_common/include/flash.h"

#include "f2802x_common/include/gpio.h"

#include "f2802x_common/include/pie.h"

#include "f2802x_common/include/pll.h"

#include "f2802x_common/include/wdog.h"
```

```
#define CONV_WAIT 1L //Micro-seconds to wait for ADC conversion. Longer than
necessary.

int16_t temp; //raw temperature sensor reading

int16_t degC; //temperature in deg. C

int16_t degK; //temperature in deg. K


CLK_Handle myClk;

FLASH_Handle myFlash;

GPIO_Handle myGpio;

PIE_Handle myPie;


void main()

{

   ADC_Handle myAdc;

   CPU_Handle myCpu;

   PLL_Handle myPll;

   WDOG_Handle myWDog;


   // Initialize all the handles needed for this application

   myAdc = ADC_init((void *)ADC_BASE_ADDR, sizeof(ADC_Obj));

   myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));

   myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
```

```
myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));

myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj));

myPie = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj));

myPll = PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));

myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));


// Perform basic system initialization

WDOG_disable(myWDog);

CLK_enableAdcClock(myClk);

(*Device_cal)();


//Select the internal oscillator 1 as the clock source

CLK_setOscSrc(myClk, CLK_OscSrc_Internal);


// Setup the PLL for x10 /2 which will yield 50Mhz = 10Mhz * 10 / 2

PLL_setup(myPll, PLL_Multiplier_10, PLL_DivideSelect_ClkIn_by_2);


// Disable the PIE and all interrupts

PIE_disable(myPie);

PIE_disableAllInts(myPie);

CPU_disableGlobalInts(myCpu);

CPU_clearIntFlags(myCpu);
```

```
    // If running from flash copy RAM only functions to RAM
#ifdef _FLASH
    memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
#endif

    // Initalize GPIO
    // Enable XCLOCKOUT to allow monitoring of oscillator 1
    GPIO_setMode(myGpio, GPIO_Number_18, GPIO_18_Mode_XCLKOUT);
    CLK_setClkOutPreScaler(myClk, CLK_ClkOutPreScaler_SysClkOut_by_1);

    // Setup a debug vector table and enable the PIE
    PIE_setDebugIntVectorTable(myPie);
    PIE_enable(myPie);

// Initialize the ADC
    ADC_enableBandGap(myAdc);
    ADC_enableRefBuffers(myAdc);
    ADC_powerUp(myAdc);
    ADC_enable(myAdc);
    ADC_setVoltRefSrc(myAdc, ADC_VoltageRefSrc_Int);
    ADC_enableTempSensor(myAdc);
```

```
//Connect channel A5 internally to the temperature sensor
   ADC_setSocChanNumber (myAdc, ADC_SocNumber_0, ADC_SocChanNumber_A5);
//Set SOC0 channel select to ADCINA5
   ADC_setSocChanNumber (myAdc, ADC_SocNumber_1, ADC_SocChanNumber_A5);
//Set SOC1 channel select to ADCINA5
   ADC_setSocSampleWindow(myAdc, ADC_SocNumber_0,
ADC_SocSampleWindow_7_cycles);
//Set SOC0 acquisition period to 7 ADCCLK
   ADC_setSocSampleWindow(myAdc, ADC_SocNumber_1,
ADC_SocSampleWindow_7_cycles);
//Set SOC1 acquisition period to 7 ADCCLK
   ADC_setIntSrc(myAdc, ADC_IntNumber_1, ADC_IntSrc_EOC1);
//Connect ADCINT1 to EOC1
   ADC_enableInt(myAdc, ADC_IntNumber_1);                    //Enable ADCINT1

   // Note: two channels have been connected to the temp sensor
    // Set the flash OTP wait-states to minimum. This is important
   // for the performance of the temperature conversion function.
   FLASH_setup(myFlash);
      //Main program loop - continually sample temperature
   for(;;)
   {
```

```
        //Force start of conversion on SOC0 and SOC1
        ADC_forceConversion(myAdc, ADC_SocNumber_0);
        ADC_forceConversion(myAdc, ADC_SocNumber_1);

        //Wait for end of conversion.
        while(ADC_getIntStatus(myAdc, ADC_IntNumber_1) == 0) {
        }

        // Clear ADCINT1
        ADC_clearIntFlag(myAdc, ADC_IntNumber_1);

        // Get temp sensor sample result from SOC1
        temp = ADC_readResult(myAdc, ADC_ResultNumber_1);

        // Convert the raw temperature sensor measurement into temperature
        degC = ADC_getTemperatureC(myAdc, temp);
        degK = ADC_getTemperatureK(myAdc, temp);
    }
}
//==========================================================================
// No more.
//==========================================================================
```

Result:

Watch Variables:

$temp =$

$degC =$

$degK =$

This program shows how to convert a raw ADC temperature sensor reading into deg. C or deg. K.

Program No 6: *Generation of gate signals for DC-DC boost converter.*

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:

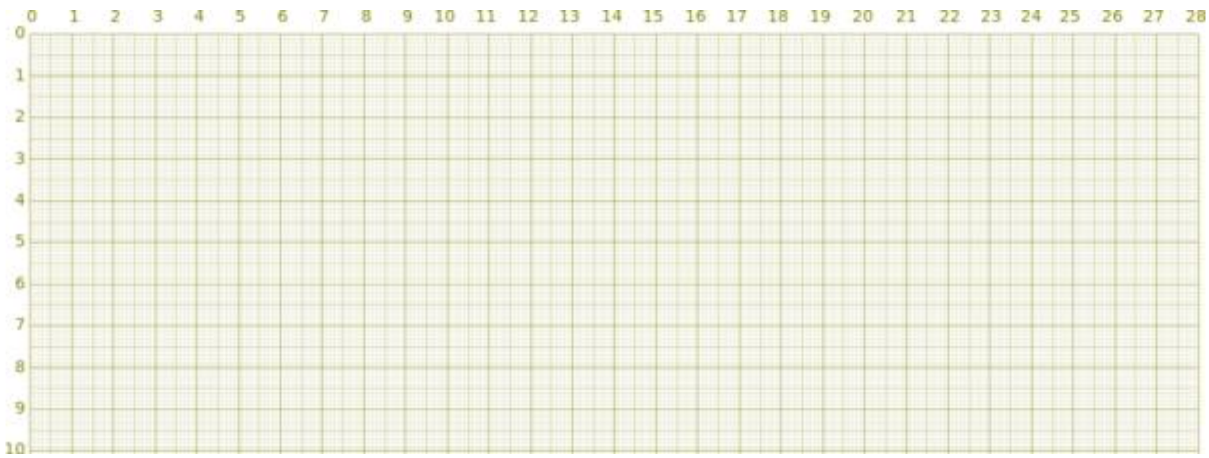
Watch variables

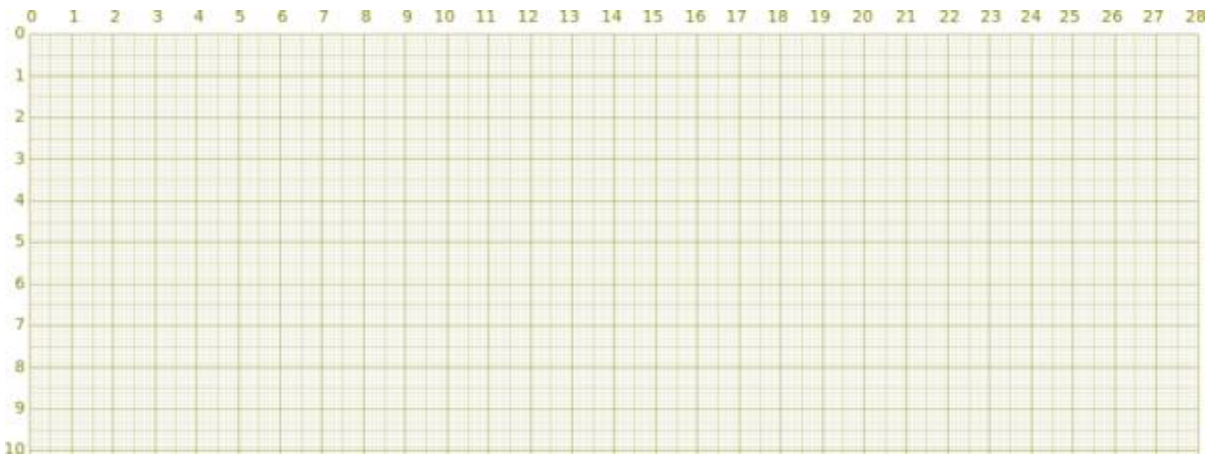
GpioCtrlRegs.GPAMUX1.bit.GPIO0

GpioCtrlRegs.GPAMUX1.bit.GPIO1

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

Graphs:





Program No 7: *Generation of gate signals for DC-AC 1-phase full bridge inverter.*

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:

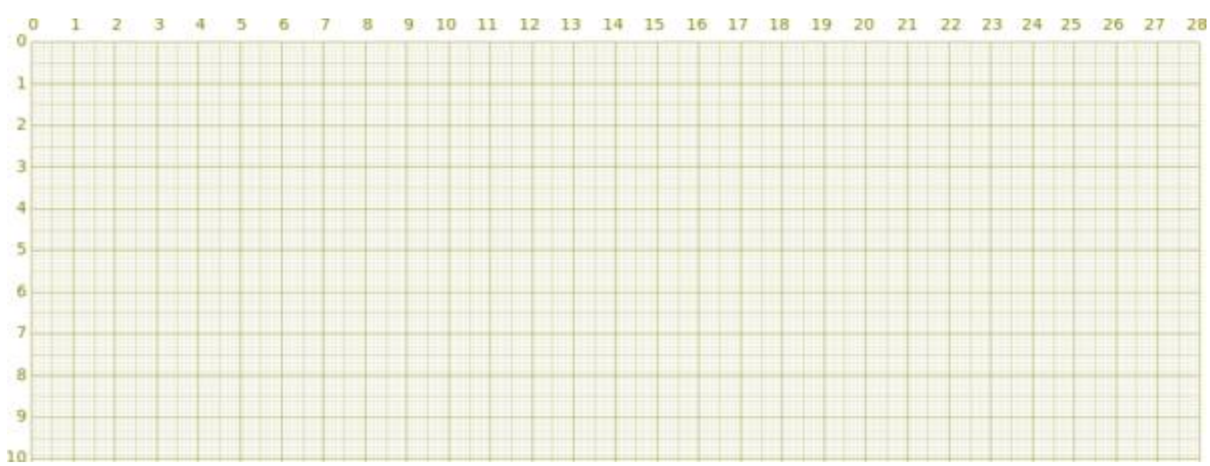
Watch variables

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 8: *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:

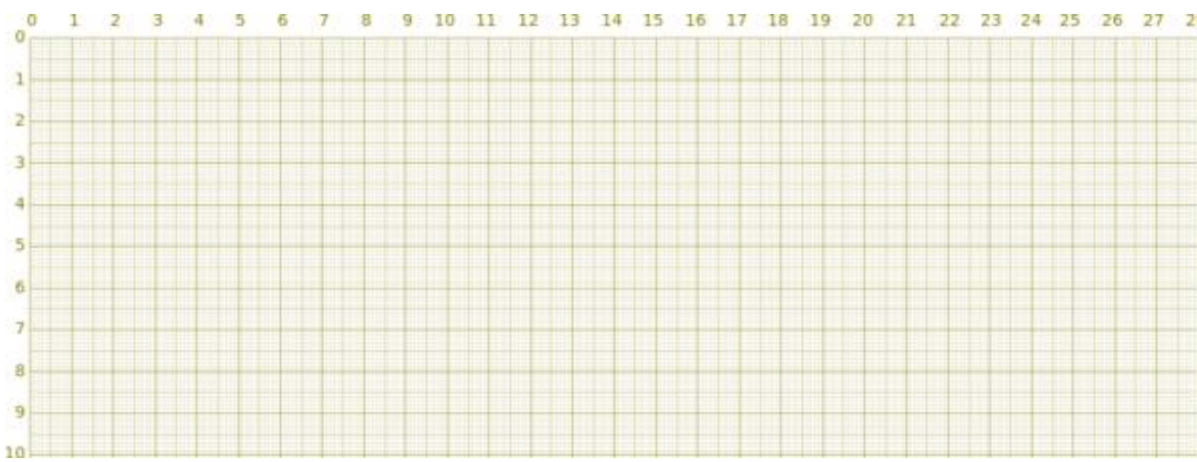
Watch variables

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 9: *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

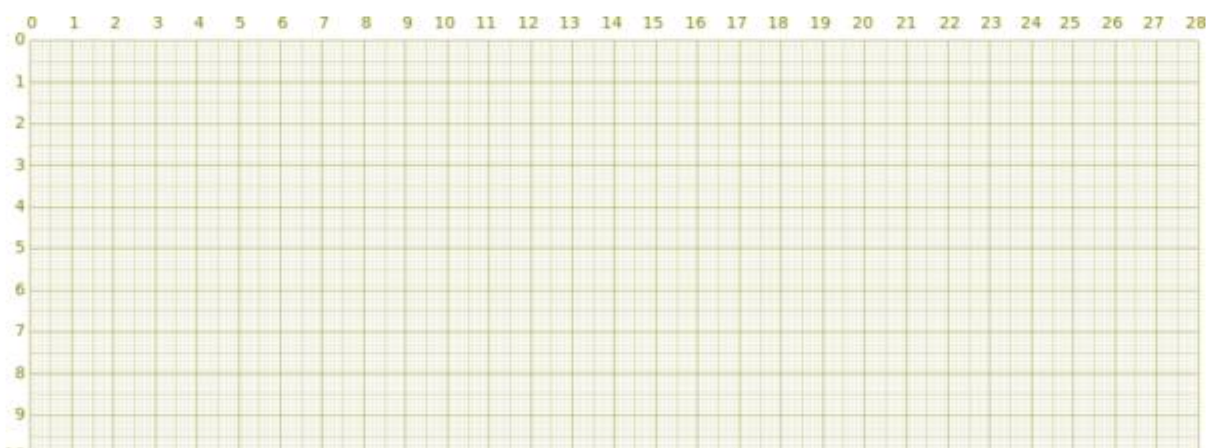
EPwm1Regs.TBCTL.bit.CTRMODE

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 10: *Generation of gate signals for 3-phase voltage source inverter.*

Date:

Objective:

To run a program that can generates the SVPWM pulses to gating the 3-phase 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:

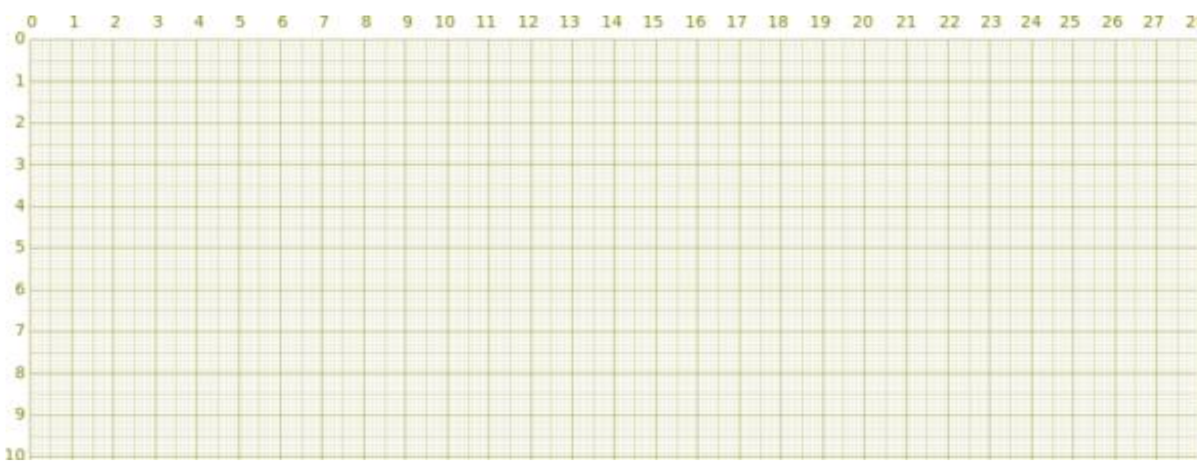
Watch variables

GpioDataRegs.GPADAT.all

GpioDataRegs.GPBDAT.all

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

Graph:



Program No 11: *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

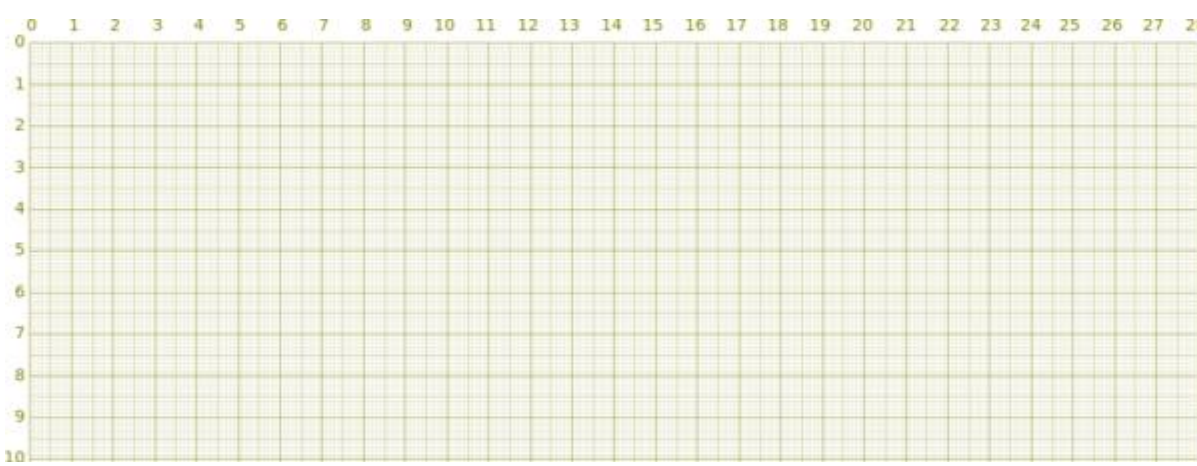
EPwm3TimerIntCount++;

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 12: *Speed control of DC motor by interfacing embedded coder with MATLAB Simulink.*

Date:

Objective:

To run a program that can vary the speed of a DC motor.

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_5ms(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_5ms(5000);

        GpioDataRegs.GPBDAT.all = 0x0;

        Delay_5ms(5000);
        }
    }
    void Delay_5ms(long end)
    {
         long i;
        for (i = 0; i <(10000 * end); i++);
    }



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

Result:

Watch variables

GpioDataRegs.GPADAT.all

GpioDataRegs.GPBDAT.all

Observe the speed of the DC motor varies with the delay of 5000 ms.

Program No 13: *Speed control of BLDC motor with a velocity control mode.*

Date:

Objective:

To run a program that can vary the speed of a BLDC motor.

Equipment required:

Hardware:

- Laptop
- DRV8312 Evaluation Board
- JTAG XDS100V1

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

The proper settings for switches and jumpers.

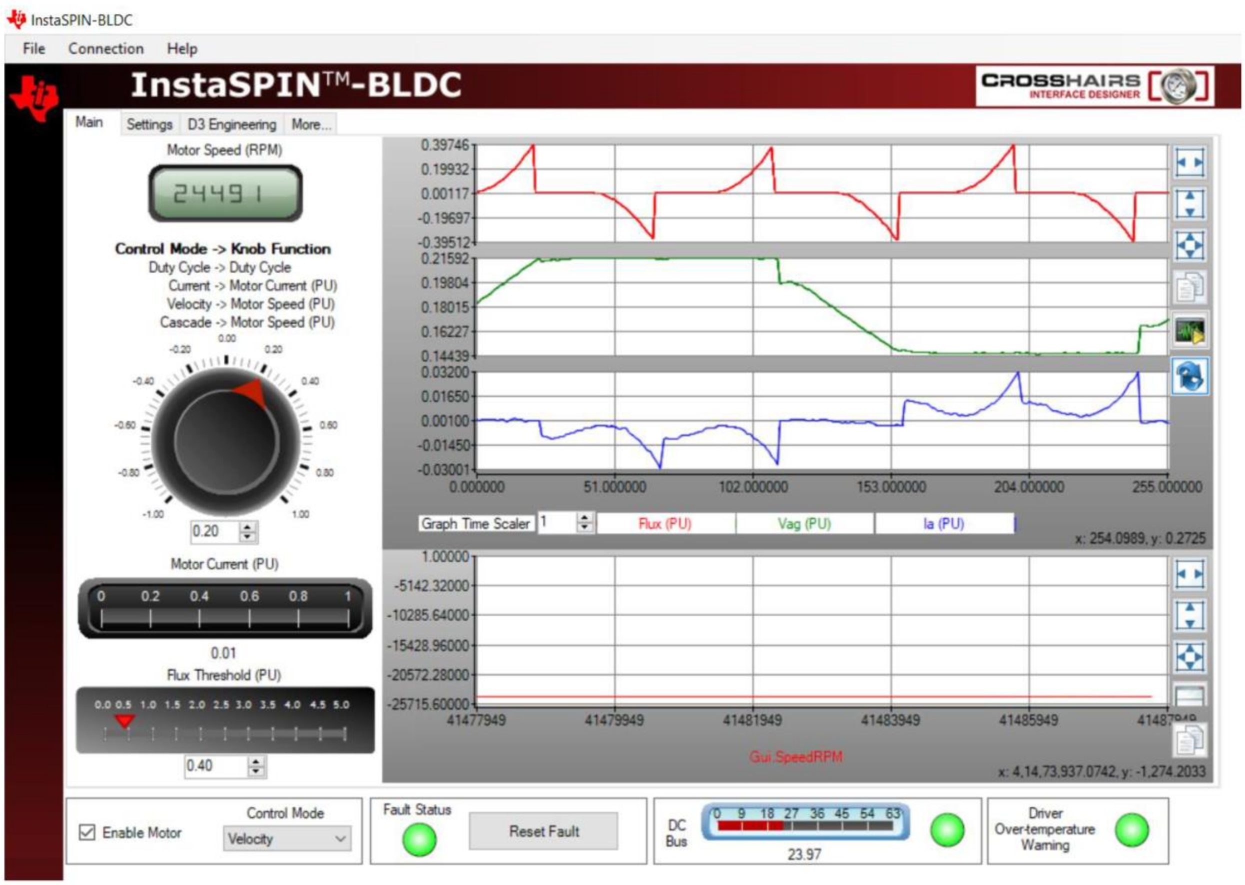
- JP1: VR1 state
- M1: HIGH position
- Motor phase wires MOA,MOB,MOC.
- DC voltage :J9.

Launching GUI application:

1. Connect the JTAG XDS1001V1 from laptop to J1 on control card S3 – ON
2. Click on c drive – TI - control suite – developing kits – drv8312-c2-kit –GUI.
3. The INSTA SPIN-BLDC connects the target and GUI is used to identify the hardware.
4. The three phase BLDC motor is connected to motor terminals of DRV8312.
5. The code is generated using control card by launching the GUI and INSTA Spin BLDC is the technology for generating integrated back Emf.

Velocity control mode:

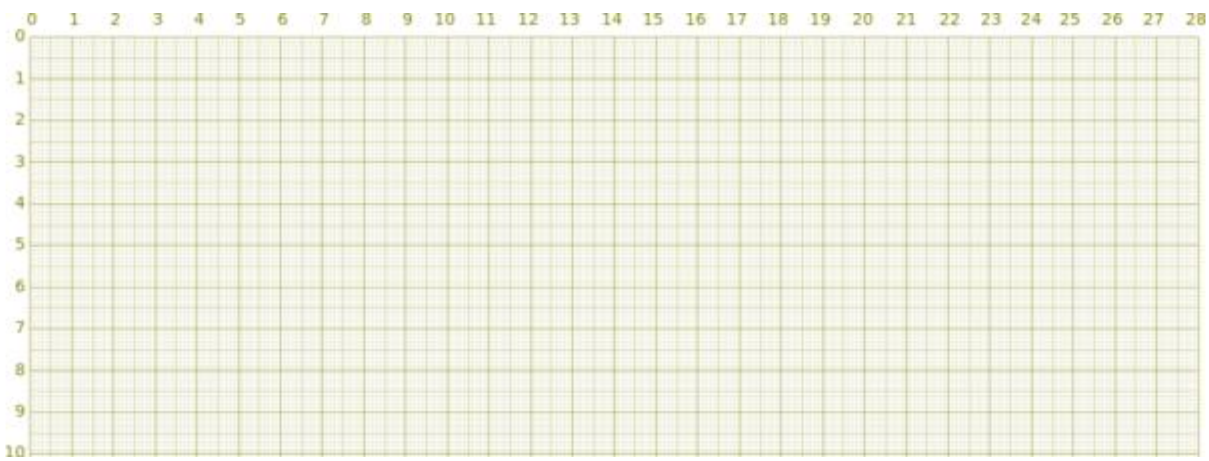
- The velocity control mode is used to control the speed using PI controller.
- The actual speed and desired speed difference is fed as an input to PI controller.
- The PI controls the duty cycle by varying K_p proportional gain and the integral gain K_i is given as PWM pulses to the switches in the inverter. This method controls the voltage amplitude required to maintain the desired speed.
- The velocity control mode is stable at higher speeds, but if the speed is lowered the system tends to unstable.
- If the motor is spinning at a higher speed, there would be many commutation intervals per second and the velocity feedback value will improve many times per second. Whereas, if the motor is rotating at a lower speeds the commutation rate will be low.



Observations:

S.No.	Velocity Mode Value	Speed in r.p.m.

Graph:



Program No 14: *Speed control of an induction motor with v/f control mode.*

Date:

Objective:

To run a program that can vary the speed of a an induction motor.

Equipment required:

Hardware:

- Laptop
- TMS320F28027F Launch Pad
- JTAG XDS100V1

Software:

- Code Composer Studio 6.0
- Windows 8 OS.

Program:

```
#include "DSP28x_Project.h"     // DSP28x Headerfile
#include "f2802x_common/include/adc.h"
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/wdog.h"
#include "f2802x_common/include/sci.h"
CLK_Handle myClk;
FLASH_Handle myFlash;
GPIO_Handle myGpio;
PIE_Handle myPie;
ADC_Handle myAdc;
```

```
int16_t temp; //raw ADC Result Data
void main() {
        ADC_Handle myAdc;
          CPU_Handle myCpu;
          PLL_Handle myPll;
          WDOG_Handle myWDog;

          // Initialize all the handles needed for this application
          myAdc = ADC_init((void *)ADC_BASE_ADDR, sizeof(ADC_Obj));
          myClk = CLK_init((void *)CLK_BASE_ADDR, sizeof(CLK_Obj));
          myCpu = CPU_init((void *)NULL, sizeof(CPU_Obj));
          myFlash = FLASH_init((void *)FLASH_BASE_ADDR, sizeof(FLASH_Obj));
          myGpio = GPIO_init((void *)GPIO_BASE_ADDR, sizeof(GPIO_Obj));
          myPie = PIE_init((void *)PIE_BASE_ADDR, sizeof(PIE_Obj));
          myPll = PLL_init((void *)PLL_BASE_ADDR, sizeof(PLL_Obj));
          myWDog = WDOG_init((void *)WDOG_BASE_ADDR, sizeof(WDOG_Obj));

          // Perform basic system initialization
          WDOG_disable(myWDog);
          CLK_enableAdcClock(myClk);
          (*Device_cal)();

          //Select the internal oscillator 1 as the clock source
          CLK_setOscSrc(myClk, CLK_OscSrc_Internal);

          // Setup the PLL for x12 /2 which will yield 60Mhz = 10Mhz * 12 / 2
          PLL_setup(myPll, PLL_Multiplier_12, PLL_DivideSelect_ClkIn_by_2);

          // Disable the PIE and all interrupts
          PIE_disable(myPie);
```

```
    PIE_disableAllInts(myPie);
    CPU_disableGlobalInts(myCpu);
    CPU_clearIntFlags(myCpu);

    // If running from flash copy RAM only functions to RAM
#ifdef _FLASH
    memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
#endif
DELAY_US(20000);
//    PIE_setDebugIntVectorTable(myPie);
    PIE_enable(myPie);

    // Initialize the ADC
    ADC_enableBandGap(myAdc);
    ADC_enableRefBuffers(myAdc);
    ADC_powerUp(myAdc);
    ADC_enable(myAdc);
    ADC_setVoltRefSrc(myAdc, ADC_VoltageRefSrc_Int);
    ADC_setSocChanNumber (myAdc, ADC_SocNumber_0,
ADC_SocChanNumber_A4);    //Set SOC0 channel select to ADCINA5
    ADC_setSocChanNumber (myAdc, ADC_SocNumber_1,
ADC_SocChanNumber_A4);    //Set SOC1 channel select to ADCINA5
    ADC_setSocSampleWindow(myAdc, ADC_SocNumber_0,
ADC_SocSampleWindow_7_cycles);   //Set SOC0 acquisition period to 7 ADCCLK
    ADC_setSocSampleWindow(myAdc, ADC_SocNumber_1,
ADC_SocSampleWindow_7_cycles);   //Set SOC1 acquisition period to 7 ADCCLK
    ADC_setIntSrc(myAdc, ADC_IntNumber_1, ADC_IntSrc_EOC1);
//Connect ADCINT1 to EOC1
    ADC_enableInt(myAdc, ADC_IntNumber_1);
//Enable ADCINT1
```

```
    FLASH_setup(myFlash);
    GPIO_setMode(myGpio, GPIO_Number_0, GPIO_0_Mode_GeneralPurpose);
    GPIO_setDirection(myGpio, GPIO_Number_0, GPIO_Direction_Output);
    GPIO_setMode(myGpio, GPIO_Number_1, GPIO_1_Mode_GeneralPurpose);
    GPIO_setDirection(myGpio, GPIO_Number_1, GPIO_Direction_Output);
    GPIO_setMode(myGpio, GPIO_Number_2, GPIO_2_Mode_GeneralPurpose);
    GPIO_setDirection(myGpio, GPIO_Number_2, GPIO_Direction_Output);
    GPIO_setMode(myGpio, GPIO_Number_3, GPIO_3_Mode_GeneralPurpose);
    GPIO_setDirection(myGpio, GPIO_Number_3, GPIO_Direction_Output);
    GPIO_setHigh(myGpio, GPIO_Number_0);
    GPIO_setHigh(myGpio, GPIO_Number_1);
    GPIO_setHigh(myGpio, GPIO_Number_2);
    GPIO_setHigh(myGpio, GPIO_Number_3);

    while (1)
    {
ADC_forceConversion(myAdc, ADC_SocNumber_0);
ADC_forceConversion(myAdc, ADC_SocNumber_1);

//Wait for end of conversion.
while(ADC_getIntStatus(myAdc, ADC_IntNumber_1) == 0) {
}

// Clear ADCINT1
ADC_clearIntFlag(myAdc, ADC_IntNumber_1);




// Get temp sensor sample result from SOC1
```

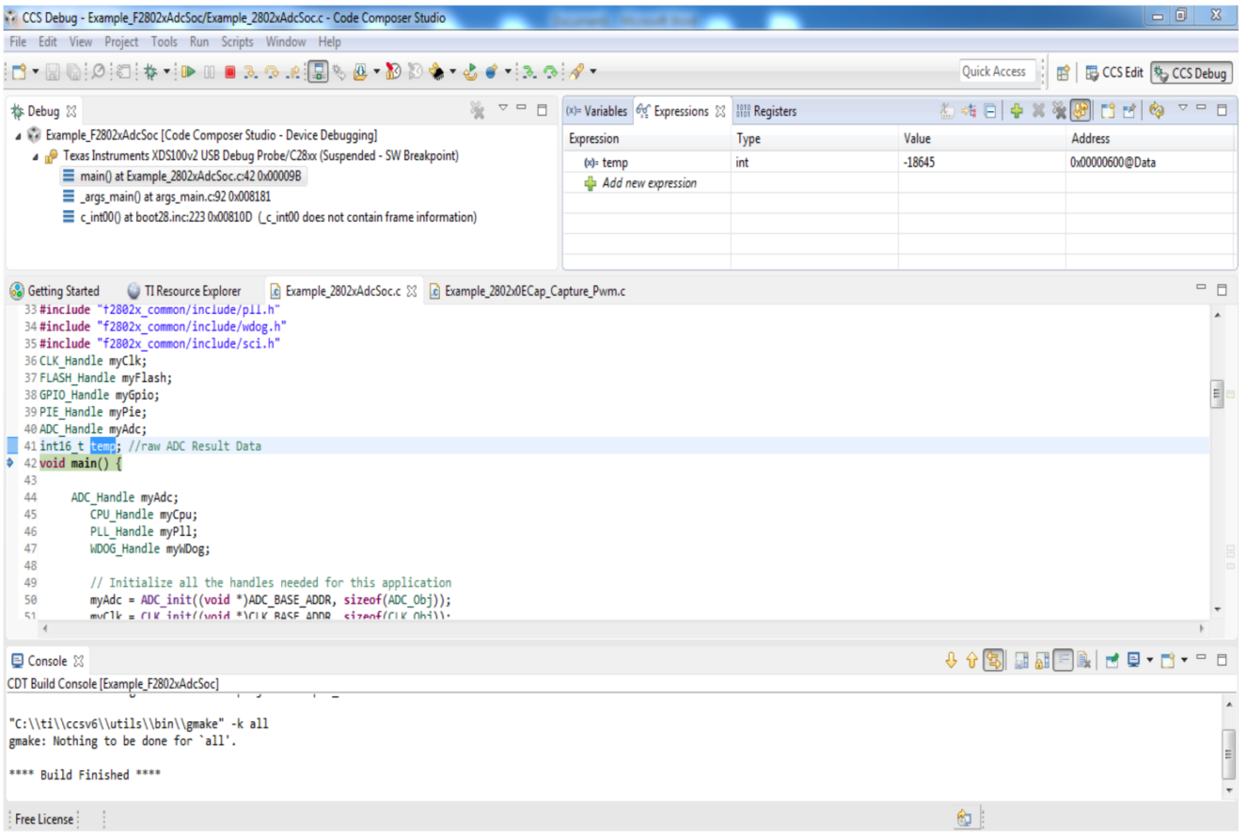
```
        int i;
        int j;
        i=10000;
        j=i-temp;

         GPIO_setHigh(myGpio, GPIO_Number_0);
         GPIO_setHigh(myGpio, GPIO_Number_1);
         DELAY_US(temp);

         GPIO_setLow(myGpio, GPIO_Number_0);
         GPIO_setLow(myGpio, GPIO_Number_1);
         DELAY_US(j);
         GPIO_setHigh(myGpio, GPIO_Number_2);
         GPIO_setHigh(myGpio, GPIO_Number_3);
         DELAY_US(temp);

         GPIO_setLow(myGpio, GPIO_Number_2);
         GPIO_setLow(myGpio, GPIO_Number_3);
         DELAY_US(j);
       }

}
//=============================================================================
// No more.

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

GUI application for voltage control of induction motor:

Steps to be followed to use GUI in code composer studio:

1) Open CCS and import pwm example into it, and copy the required code into that example.

2) Build the example and next debug it as shown below.

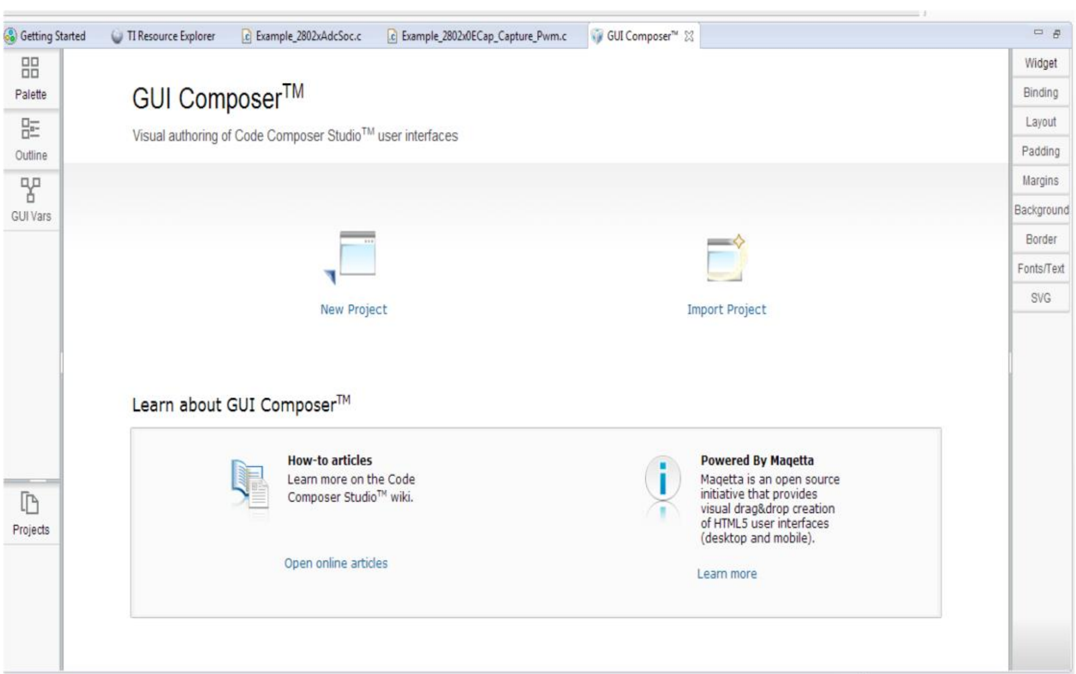


3) Once it is debugged, take an expression of a variable i.e. to vary in GUI. Here, in this example TEMP is the variable which is considered for the varying the DUTY CYCLE. First of all declare this *temp* as a global variable it means at the top of code just like shown below.

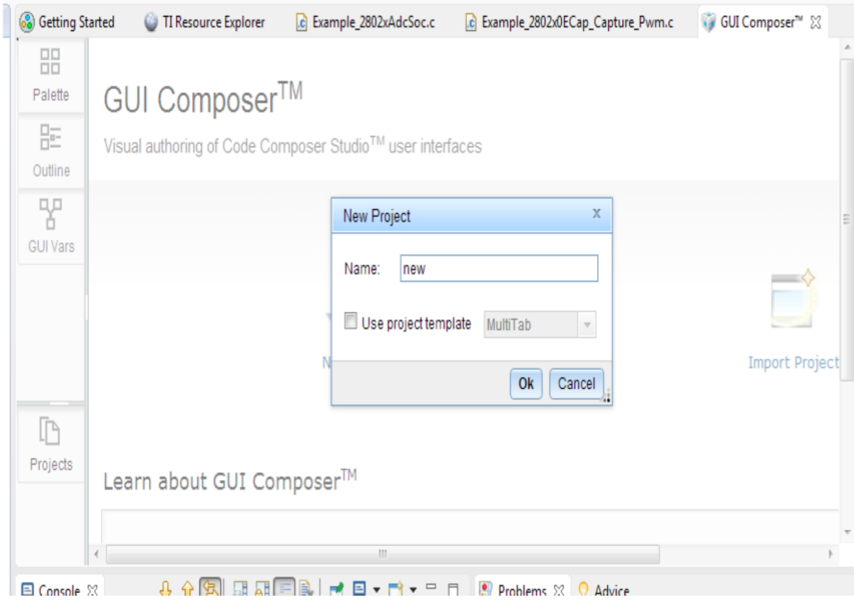
```c

#include "DSP28x_Project.h"     // DSP28x Headerfile
#include "f2802x_common/include/adc.h"
#include "f2802x_common/include/clk.h"
#include "f2802x_common/include/flash.h"
#include "f2802x_common/include/gpio.h"
#include "f2802x_common/include/pie.h"
#include "f2802x_common/include/pll.h"
#include "f2802x_common/include/wdog.h"
#include "f2802x_common/include/sci.h"
CLK_Handle myClk;
FLASH_Handle myFlash;
GPIO_Handle myGpio;
PIE_Handle myPie;
ADC_Handle myAdc;
int16_t temp; //raw ADC Result Data
void main() {

     ADC_Handle myAdc;
        CPU_Handle myCpu;
        PLL_Handle myPll;
        WDOG_Handle myWDog;
```

4) Once we declare *temp* as a global variable, add it to watch expression to see how it varies.

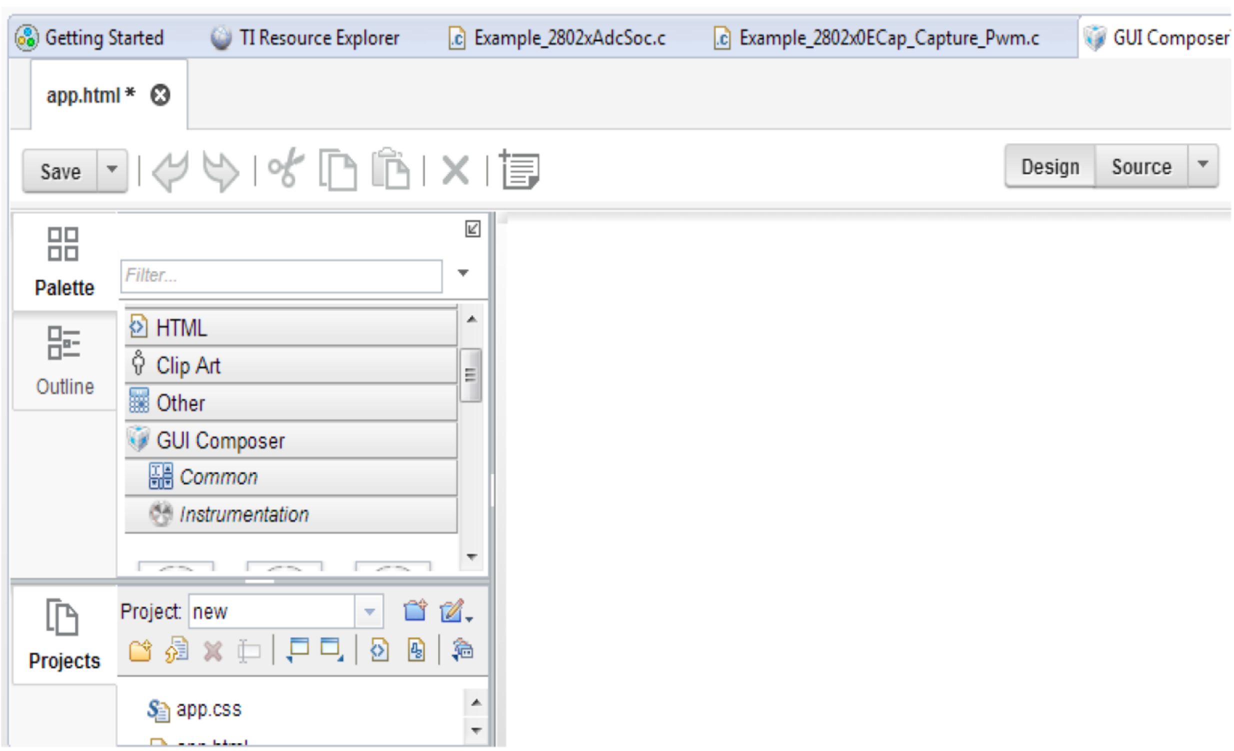
5) Now go to VIEW button and select GUI composer in it, once it is opened it is seen as below,



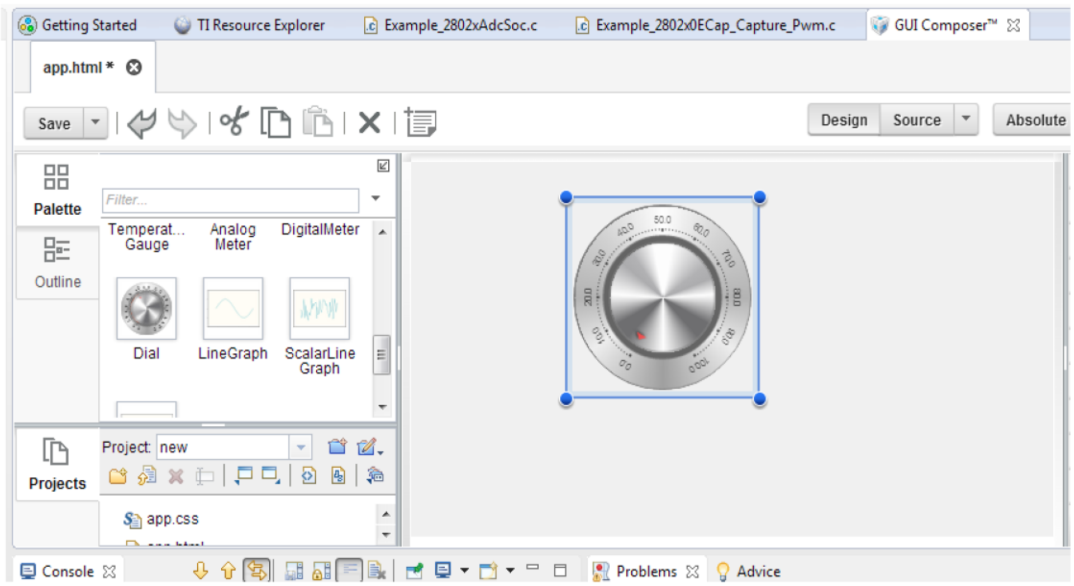
6) Go to new project, then it will ask to assign a name to it; then assign a name. Here in this example it is saved as NEW as a project name.



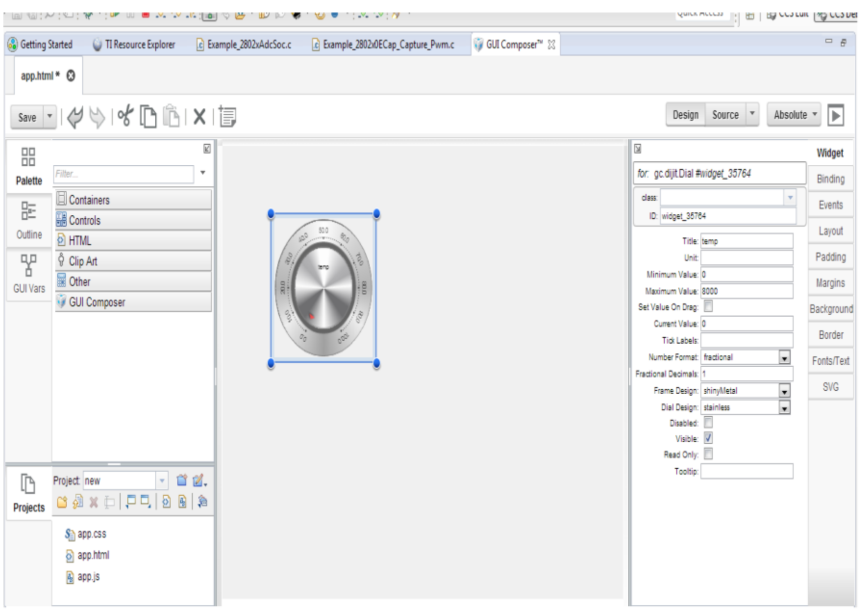
7) We can see palette, outline, and GUI vars on its left side, go to palette and GUI composer in it and instrumentation.



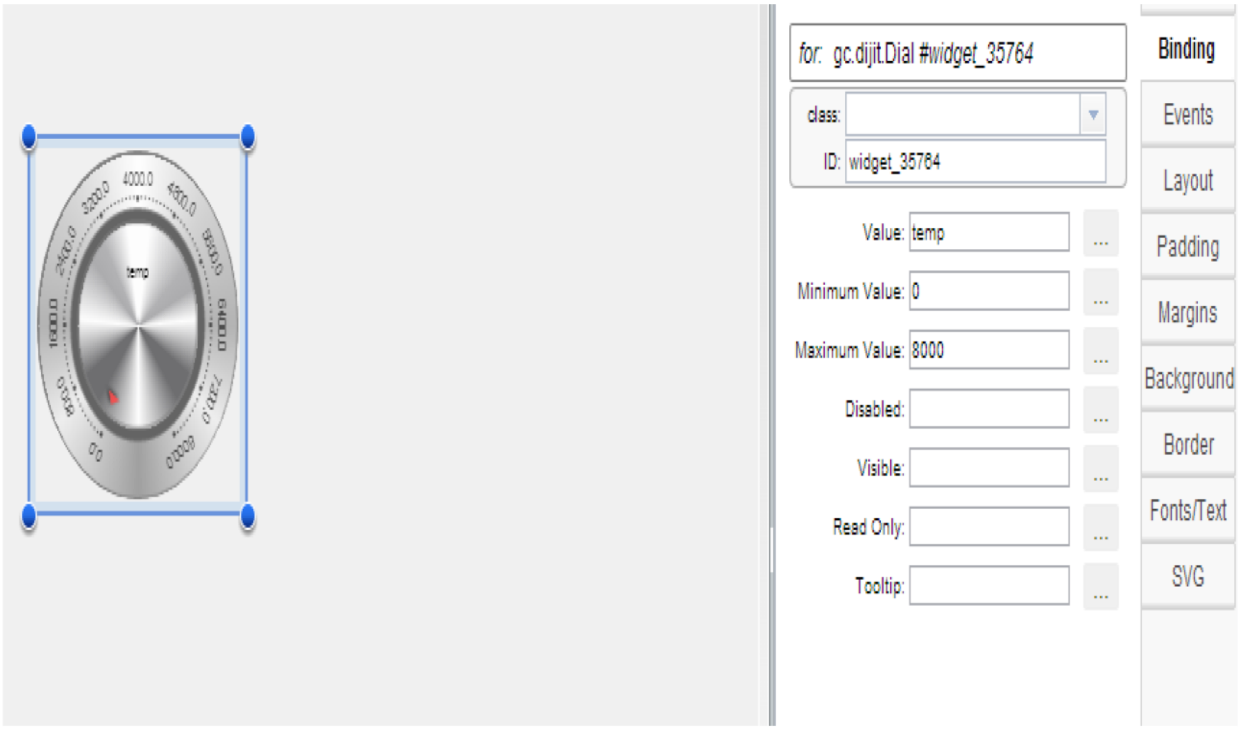
8) Once we open an instrumentation, we can find many dials in it; select the dial shown below by drag and dropping it.



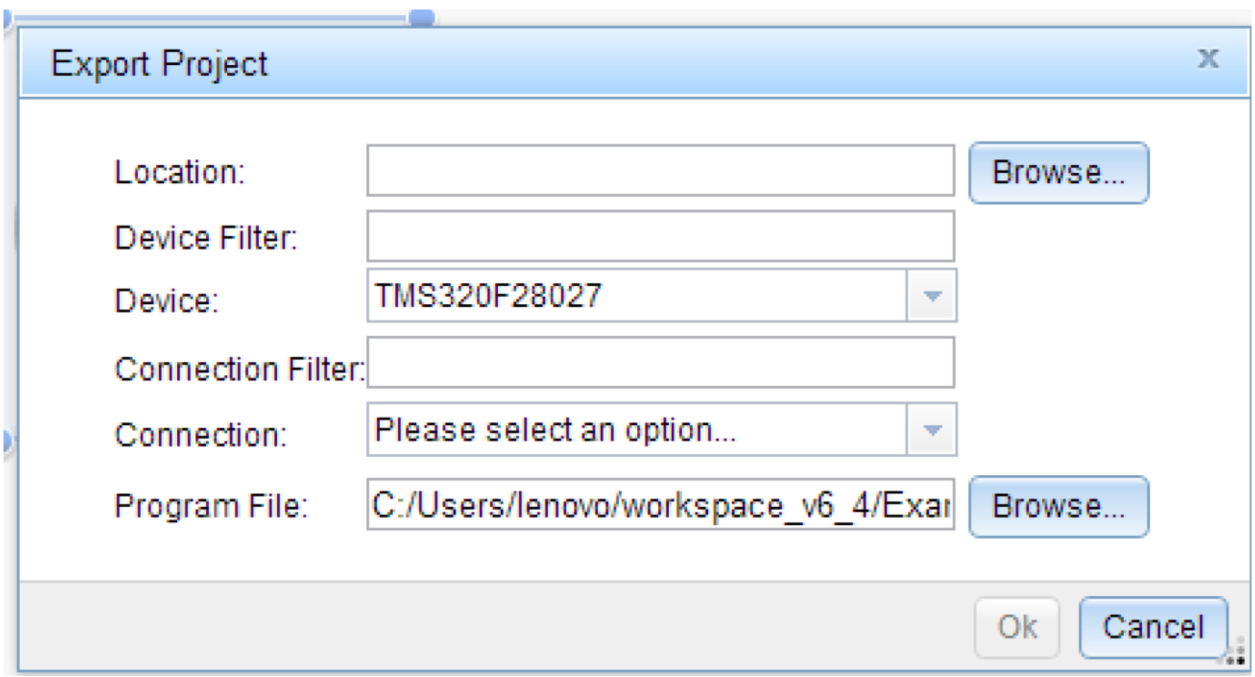
9) Once we select the dial, go to the *widgets* which is on right side, and assign *temp* into its *Title* and we can set the range of dial as shown below. Here we need to change the *Title*, *Minimum value* and *Maximum value.*



10) Then go to *Binding* and change the values as



11) Next export the project to the location



12) First save the folder of the example project at any local disk or desktop.

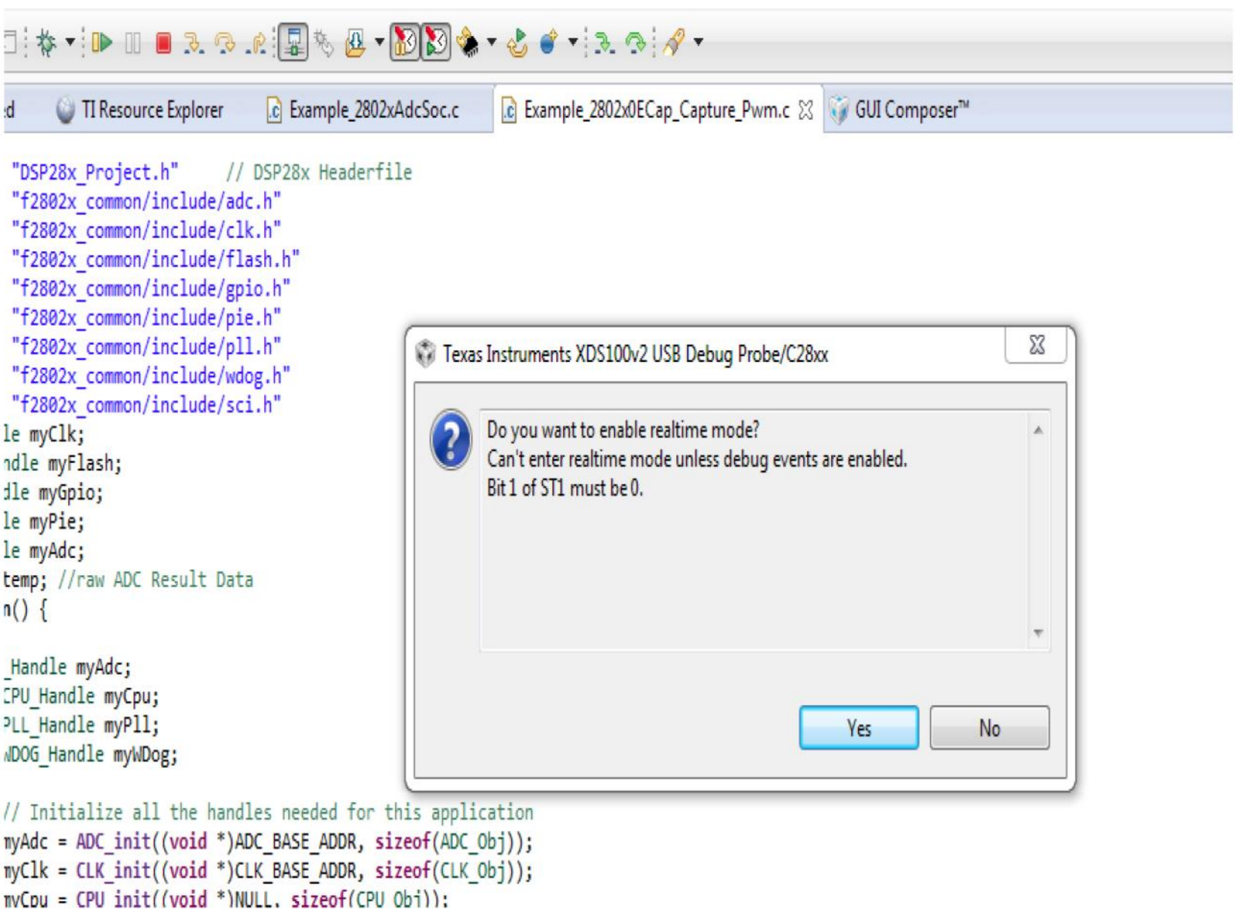
13) We define one location to save this project into it at LOCATION.

14) Select f28027 device

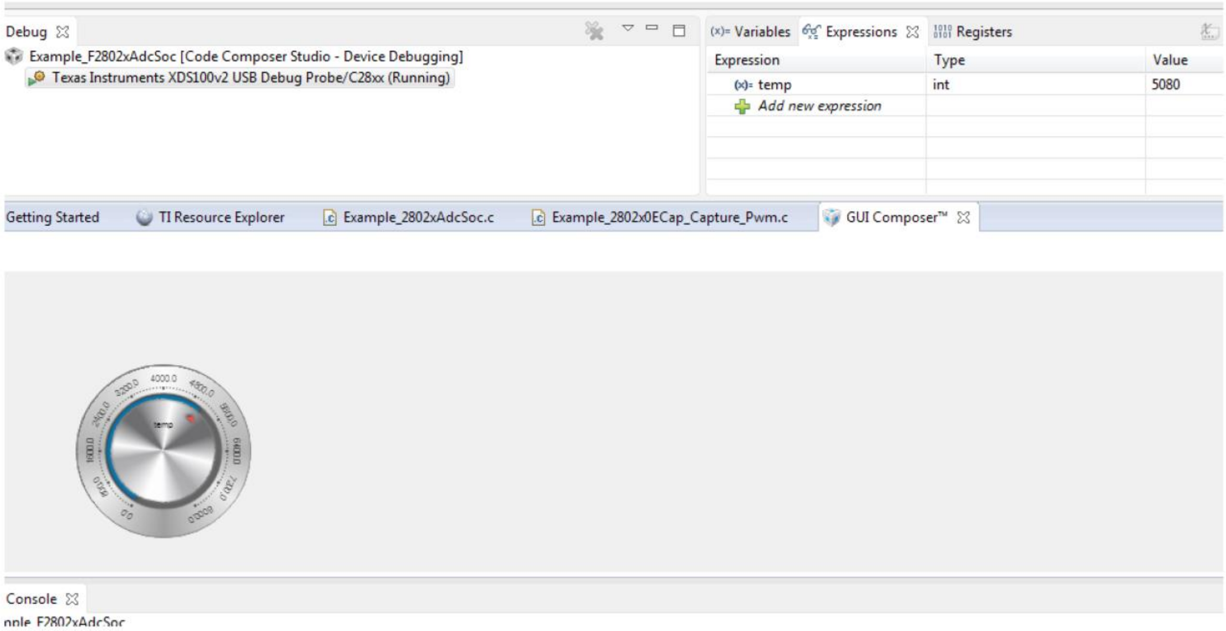
15) Select XDS 100v2 USB connection.

16) And browse the program file which we have saved in particular location; then browse that particular coded (.c) file from location. And click OK.

17) After go to CCS and enable *silicon real time mode* as shown below; it is to be done after build and debug process.



18) After enabling it, just to run the program and go to GUI composer and click PREVIEW MODE on right side shown as arrow. And now we can vary the knob and see the respected changes in watch expression which correspondingly changes the speed of the induction motor.



Program No 15: *Speed control of PMSM motor.*

Date:

Objective:

To study the speed of a PMSM motor.

Equipment required:

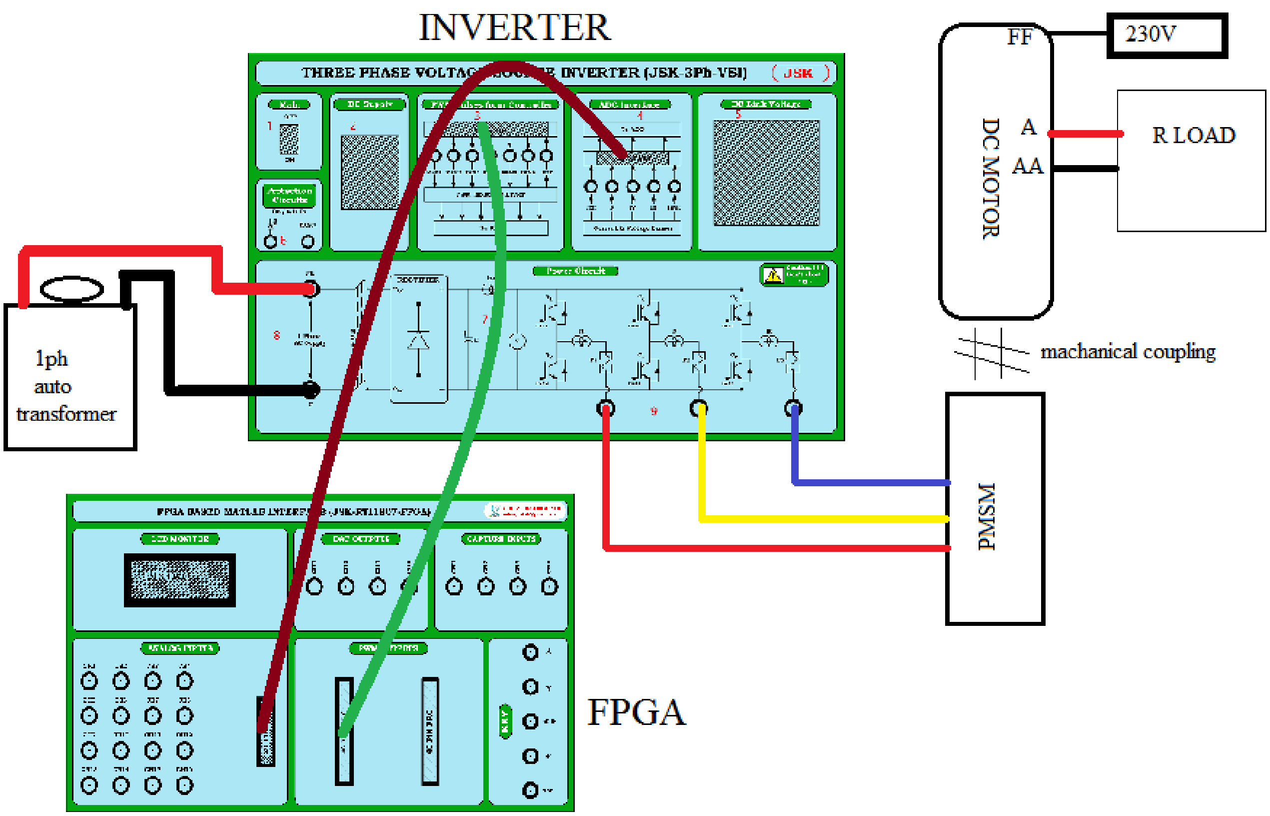
Hardware:

- JSK-3PHVSI-1KW-SP6(Device) module.
- 310V AC PMSM Motor coupled With DC Shunt Motor.
- 1-Ph Auto Transformer.
- Patch chords.
- Power chords.

CONNECTION PROCEDURE

1. Connect the power chord to the AC input of the device module (JSK-3PHVSI-1KW) provided at the back of the module.
2. Connect power chord to FPGA BASED MATLAB INTERFACE (JSK-RTI1807-FPGA) board.
3. Connect 2 pin power chord to dc motor field supply.
4. Connect A, AA terminals of DC SHUNT Motor to Resistive Load.
5. Connect the 230 VAC supply to the AUTO transformer and connect the output of Auto transformer to the terminals Ph, N AC INPUT of the 3φ VSI module.
6. Connect the PMSM Motor R, Y,B terminal to Power module R,Y,B Terminal.
7. Connect the PMSM Motor speed feedback terminal to Power module.
8. Connect FPGA controller 40pin and 20 pin from FPGA controller to 3$ VSI module.
9. Connect the MOTOR body to EB SUPPLY earth.

CONNECTION DIAGRAME



EXPERIMENT PROCEDURE:

1. Verify the connection as per the connection procedure.
2. Switch ON the FPGA BASED MATLAB INTERFACE (JSK-RTI1807-FPGA) board.
3. Switch ON the power ON/OFF switch of the power module (JSK-3PH-VSI).
4. Protection circuit LED is glow mean press the Reset button; if that LED is glow mean we cannot be able to operate the system.
5. connect the output of the transformer to the power module (JSK-3PH-VSI). Switch ON the AUTO transformer; check the auto transformer output is zero; if not, then make it zero.

6. Switch on the power Field (F & FF) supply of DC SHUNT MOTOR.
7. Now LCD of the module displays the following,

JSKLAB INSTRUMENTS

CHENNAI - 100

PMSM MOTOR SPEED
CONTROL

1. REF SPEED:
2. ACTUAL SPEED:

8. Switch on the MCB. Vary the Auto transformer voltage from minimum up to 310V DC of DC LINK voltage.
9. Now, vary the REFERENCE SPEED of the motor. Now, the ACTUAL SPEED also will be maintain in same from Min. to Max. OR Max.to Min., because of its running in closed loop condition.
10. Now, vary the resistive load (across the A & AA terminal of DC SHUNT MOTOR).
11. Now, the current of the PMSM motor will be increase; but the speed will be maintaining constant.
12. All *pwm* pulses & current waveforms can be able to see with respect to ground.
13. Vary the resistive load step by step; that time also the speed will be maintaining in constant.
14. After finishing the experiment, make the zero voltage of AUTO transformer & SWITCH OFF the MCB.
15. Then, SWITCH OFF all systems.

PROTECTION CIRCUIT:

1. During running time any over current (I_{dc}, I_r, I_y, I_b) happen means the protection circuit will be enabling.
2. First make the voltage zero in AUTO transformer; then wait sometime up to the DC LINK VOLTAGE is zero.
3. Then press the reset button of FPGA BASED MATLAB INTERFACE (JSK-RTI1807-FPGA) Board.
4. Now press the RESET button of the module (JSK-3PHVSI-1KW).
5. Don’t see the output voltage waveform without isolation of scope.
6. Don’t short the high voltage & low voltage terminals.

Department of Electrical & Electronics Engineering

Vision of Institute:

To be among the best of the institutions for engineers and technologists with attitudes, skills and knowledge and to become an epicenter of creative solutions .

Mission of Institute:

To achieve and impart quality education with an emphasis on practical skills and social relevance.

Vision of Department:

The Vision of Electrical and Electronics Engineering Department is to become a nationally and internationally leading institution of higher learning, building upon the culture and values of universal science and contemporary education, and a center of research and education generating the knowledge and the technologies which lay the groundwork in shaping the future in Electrical and Electronics Engineering.

Mission of Department:

To provide Technical knowledge and soft skills required to succeed in life, career and help society to achieve self-sufficiency.

Gokaraju Rangaraju Institute of Engineering & Technology (Autonomous)

Nizampet Road, Bachupally, Kukatpally, Hyderabad- 500090
Telangana, India.
Ph: +91-040- 65864440, 65864441. Fax: 040-23040860, info@griet.ac.in