Development of Software for Switched Power Supply Module with DSP Toolkit

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Dharmendra Savaliya
(14MECE05)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

MAY 2016

Development of Software for Switched Power Supply Module with DSP Toolkit

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Dharmendra Savaliya
(14MECE05)

Under the guidance of

External Project Guide:

Mr. Rasesh Dave (Eng.-SD)
Power Supply Group,
ITER-India, IPR, Block A,
Sangath Skyz, Bhat-Motera Road,
Koteshwar, Ahmedabad.

Internal Project Guide:

Prof. Amit Degada
Assistant Prof., EC Branch,
EE Depart., Institute of Tech.,
Nirma University,
Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

MAY 2016

Declaration

This is to certify that

i). The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

ii). Due acknowledgment has been made in the text to all other material used.

- Dharmendra Savaliya

14MECE05

Certificate

This is to certify that the Major Project entitled **"Development of Software for Switched Power Supply Module with DSP Toolkit"** submitted by **Dharmendra N. Savaliya (14MECE05)**, towards the fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmadabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof.Amit Degada
Internal Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. D.K.Kothari
Section Head (EC)

Dr. P.N.Tekwani
HoD(EE),Director-IT

Certificate

This is to certify that the Major Project entitled **"Development of Software for Switched Power Supply Module with DSP Toolkit "** submitted by **Dharmendra N. Savaliya (14MECE05)**, towards the fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached to the level of required for being accepted for examination.

Date: Place: Ahmedabad

Project Guide:	**Deputy Proj. Man.:**	**Project Manager:**
Mr. R.J. Dave	**Mr. N.P. Singh**	**Mr. U.K. Baruah**
Engineer-SD,	Engineer-SF,	Engineer-H,
Power Supply Group,	Power Supply Group,	Power Supply Group,
ITER-India, IPR.	ITER-India, IPR.	ITER-India, IPR.

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. D. K. Kothari**, Head of Electrical Engineering Department, and **Dr. N. P. Gajjar**, PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Prof. Amit Degada**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to my Project Guide **Mr. Rasesh J. Dave**(ITER-india) for his cordial support and providing valuable information regarding the project and guidance, which helped me in completing this task through various stages. I would also thank to **Mr. N. P. Singh (Deputy Project Manager)** and **Mr. Ujjwal K. Baruah (Project Manager)** Power Supply Group(ITER-India) for always giving good suggestions and help to solve to complete my project in better way.

Lastly, I thank almighty, my parents and friends for their constant encouragement, without which this assignment would not be possible.

- Dharmendra Nandlalbhai Savaliya

14MECE05

Abstract

HVPS (High Voltage power supply) can be used in industrial application such as RF (Radio Frequency) heating. Design & implementation of on-board control for HVPS (high voltage power supply) is done with switched power supply module. Designed High voltage power supply regulation is critical and requires high speed switching electronic system.

A DSP(Digital Signal Processor) software development to achieve on board control, protection and communication functionality of the single Switched Power Supply (SPS) module is proposed. Universal Asynchronous Receive/Transmit (UART) based communication for the SPS module allows transition from one state to another state as per defined sequence. On board controller manages the controlled reproduction and routing of gate pulse for SPS with settable frequency and variable duty cycle. A control interface is designed to communicate with Test Jig via UART bus protocol. A Test Jig (with another DSP) is also envisaged to demonstrate required functionality. Real time Parameters of SPS Module i.e. Heat sink temperature, DC link voltage (Analog to Digital Converter(ADC) based) & present state of the module should be monitored. Real time Pulse width modulation regeneration with delay in terms of nano seconds achieved with XINT(External interrupt).

Operating principles and protection considerations of SPS module are achieved by on-board controller software development of high voltage power supply. Complete system with Test Jig and one Field application board is tested and implemented to verify feasibility of the proposed scheme.

Contents

List of Figures

List of Tables

Abbreviation Notation

ADC ..... Analog to Digital Convertor
CAN ..... Controller Area Network
ECAP ..... Enhanced Capture Module
EPWM ..... Enhanced Pulse Width Modulation
EQEP ..... Enhanced Quadrature Encoder Pulse
FOC ..... Fiber Optical Cable
GUI ..... Graphical User Interface
HVPS ..... High Voltage Power Supply
HES ..... Hall Effect Sensor
I2C ..... Inter-Integrated Circuit
IGBT ..... Insulated Gate Bipolar Transistor
IPR ..... Institute for Plasma Research
ITER ..... International Thermonuclear Experimental Reactor
JTAG ..... Joint Test Action Group
LIN ..... Local Interconnect Network
PCB ..... Printed Circuit Board
PIE ..... Peripheral Interrupt Expansion
PLL ..... Phase Locked Loop
SCI ..... Serial Communication Interface
SPI ..... Serial Peripheral Interface
SPS ..... Switched Power Supply
UART ..... Universal Asynchronous Receiver/Transmitter
VFC ..... Voltage to Frequency Control
XDS ..... Cross Document Sharing
XRS ..... External Reset

Chapter 1

Introduction

1.1 Motivation

High Voltage Power Supply (HVPS) are mainly configured with multi-secondary transformers, insulated gate bipolar transistor (IGBT) based switched Power Supply Modules and controllers. For switched power supply modules, onboard intelligence is mandatory requirement that controls the operation, observes self-protection and communicates with HVPS main controller. This thesis report mainly covers the programming part of DSP based controller of switched power supply module. Testing of DSP program is performed on evaluation board & available as developed Hardware.

This project uses a DSP (TI TMS320F28335 Evaluation Kit) software programming to achieve onboard control, protection and communication functionality of the single SPS (Switched Power Supply) module. UART based communication for the SPS module allows transition from one state to another state as per defined sequence. On board controller manages the controlled reproduction or routing of gate pulse for SPS with settable frequency and duty cycle. Real time Parameters of SPS Module viz. Heat sink temperature, DC link voltage & present state of the module should be monitored. A Test Jig (with another DSP) is also used to demonstrate

required functionality.

1.2 Problem Statement

A typical chopper controller involves switching control, reproduction of switching pulses for IGBTs, besides temperature monitoring, Over Voltage, Under Voltage protection. It involves UART communication for state transitions.

JTAG debugger of controller with XDS cable allows real time debugging in Texas instrument's code composer studio (CCS) debugger. For development of chopper controller, DSP using C programming language with inbuilt PWM, SCI(UART) and ADC module is selected.

1.3 Overview

HVPS's Chopper control card can be configured with different ways in this project, it is based on DSP F28335 Delfino Controller & DSP controller is used to perform switching operation, monitoring & Circuit protection.

In this project development of software to perform & validate SPS module controller is proposed. To achieve desired operations & functionality check of SPS control card is done by Test Jig controller.

Multi-Converter HVPS System contains multiple chopper module. Each chopper module contain it's own DSP based controller.

In figure 1.1 it shows component of chopper like contactor, diode rectifier & IGBT Switch. This Component can be controlled by DSP controller. DC (Direct Current) voltage across balancing resistor, current across IGBT switch & temperature of heat

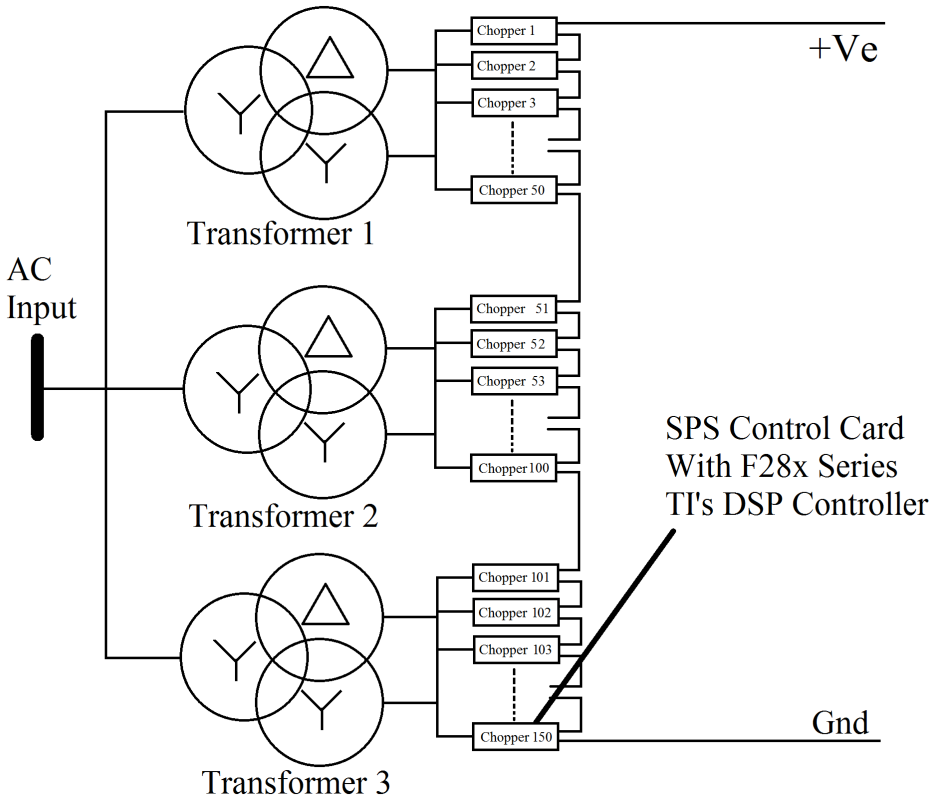


Figure 1.1: Multi-Converter HVPS System

sink at IGBT can be monitor by chopper control card. Developed Software for chopper control card contain multiple task to handle different operation i.e. Parameter passing, DC link voltage monitoring & Current monitoring.

Generated gate pulse is given to IGBT Switch A as shown in 1.2 for varying output voltage with duty cycle. When IGBT switch A is turned off capacitor bank should be discharge to avoid circuit damage, so IGBT switch B is used for fast discharging.

Operation of SPS module control will be divided into different task (state) as shown in 1.3. The sequences of operation are Parameter passing, DC link up, Stand by, Ready & Power on. In Parameter passing Test jig will pass all data range of frequency, temperature, IGBT, under voltage, over voltage & over current. If data

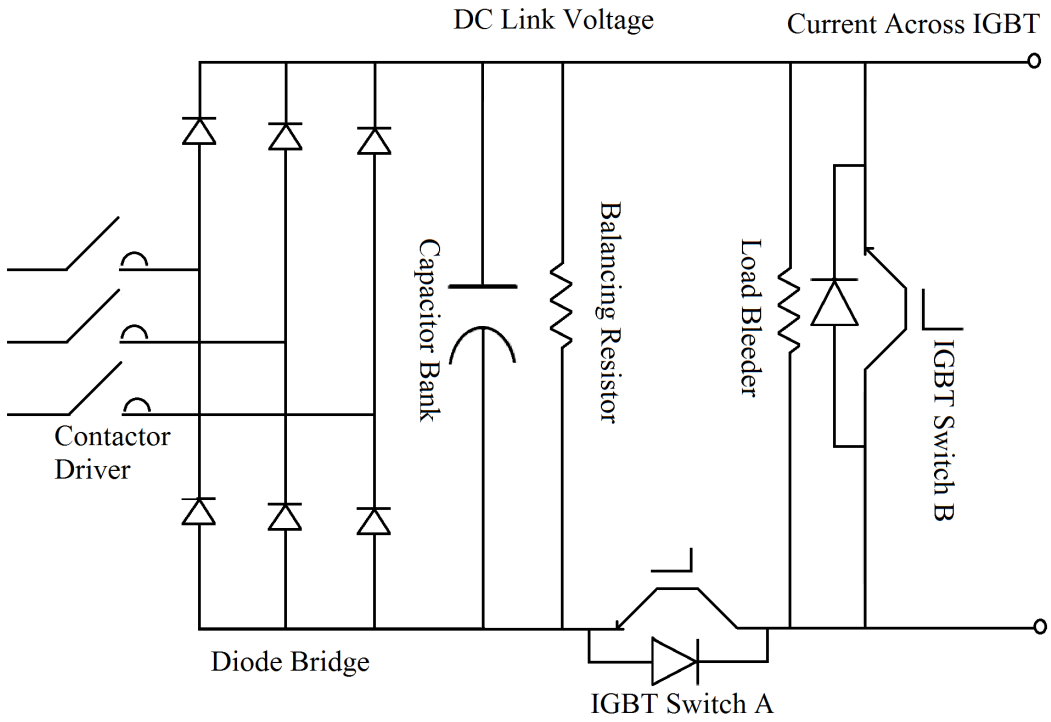


Figure 1.2: SPS Control card circuit

received by control card are ok then ACK be transmitted & achieved state will be DC link up.

In DC link up voltage across capacitor bank is built by soft charging via resistor & it will be measured, if voltage is in range then after 9 sec next state stand by will be achieved. In standby state the control card will directly connected to the power supply & voltage will be measured by sampling, if voltage is not in range error will be generated by control card. When Test jig transmit command for next state SPS module will come into Ready state.

In Ready state if all data checking will be without error then IGBT will be enabled. Then power on command will be transmitted by Test jig & SPS module will start receiving PWM. If received duty cycle is more than 90% SPS module will

generate error.

As shown in figure 1.3 on power ON DSP go into infinite loop, first checking all condition for required parameters are in range. During communication in DC link or Standby state loss occurs than it will generate error. If in state Standby, Ready & Power on SPS receive OFF Pass command it will go back to previous state.

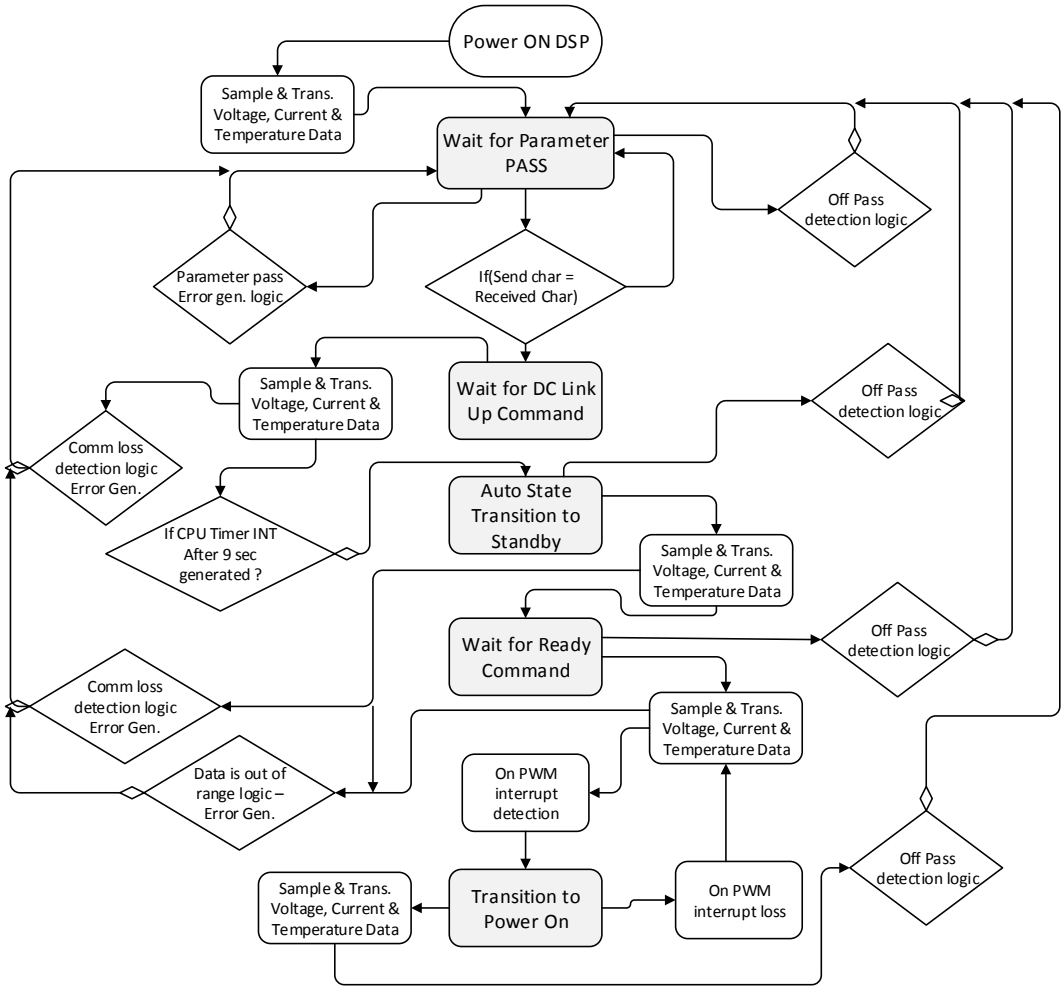


Figure 1.3: Control flow of program

1.4 Project Block Diagram

Control card is connected with external peripheral like IGBT (Insulated Gate Bipo-

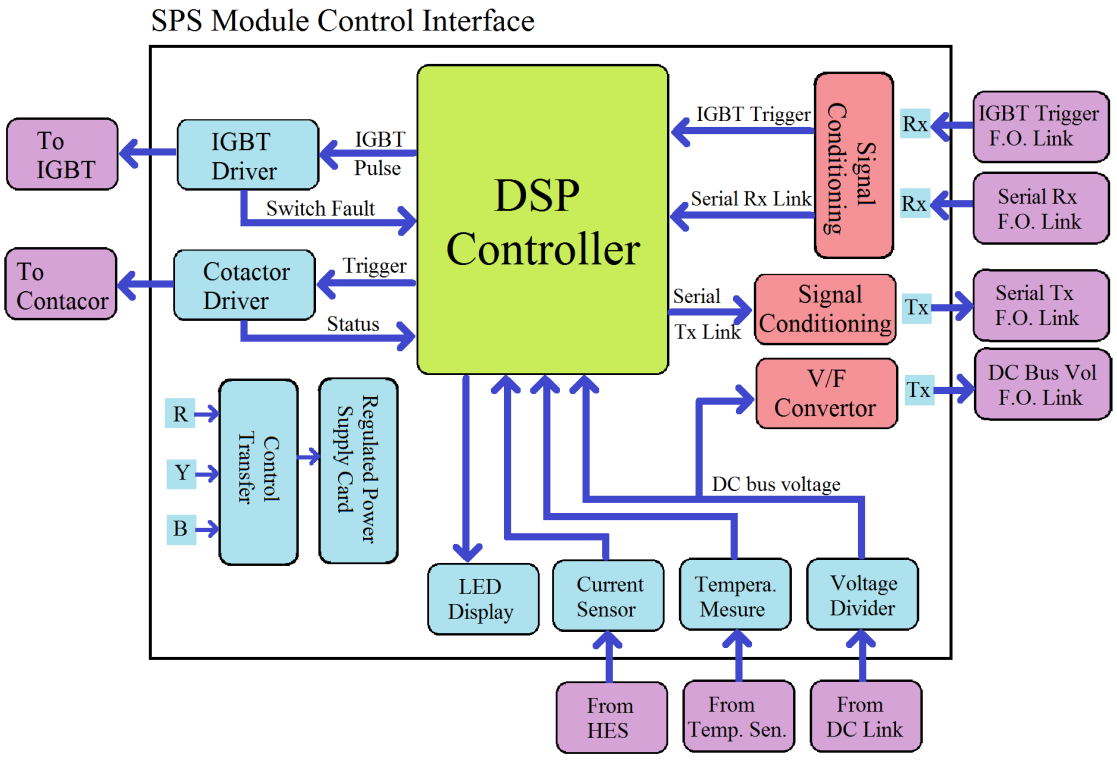


Figure 1.4: Project Block diagram

lar Transistor) driver, Contactor Driver, LED Display, Current Sensor, Temperature sensor & Voltage divide. Communication of controller with Test Jig Controller Board is done by SCI Fiber Optic cable(FOC) link as shown in figure 1.4.

- IGBT driver is useful for switching IGBT operation.
- Contactor driver is useful to operate contactor.
- LED Display useful for displaying of State of SPS module & Error code in SPS module.
- Hall Effect Sensor(HES) is useful for current measurement.

1.4.1 Customized UART Communication Protocol

Communication is done by Serial communication interface(SCI) as SCI is useful for longer distance & main advantage of SCI is Asynchronous communication.

Table 1.1 shows packet structure we have used.Bit 0 of Packet 1 & Packet 2 Contain Health bit. Bit 1 of Packet 1 & Packet 2 Contain Packet ID for packet 1 it is 0 & for Packet 2 it is 1.

In packet 1, bit 2,3 & 4 indicate Status of Chopper module & bit 5,6 & 7 indicates MSB Data of our 9bit DC link value. Last six bits of Packet 2 contain 6 bit MSB of our full DC link value.

Table 1.1: UART Data Frame for Packet

Packet 1 :

Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
Health Bit	Packet ID	Status Bit 1	Status Bit 2	Status Bit 3	Data Bit 1	Data Bit 2	Data Bit 3

Packet 2 :

Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
Health Bit	Packet ID	Data Bit 4	Data Bit 5	Data Bit 6	Data Bit 7	Data Bit 8	Data Bit 9

Test Jig Controller have to perform more number of UART communication Simultaneously. So, Test Jig can handle number of SPS control card By choosing proper parallel Processor. In this project, it considers parameters like frequency monitoring, IGBT Switching, temperature monitoring, under voltage control, over voltage control & over current control.

Table 1.2: Packet Data

Parameter	**Range**	**Original Value**	**Decimal value**
Start Code	-	-	555(HEX)
Frequency	1,2.5,5,10 kHz	1	301
Temperature	50-80 C	70	370
IGBT Switch	0/1	0	300
Under Voltage	600-650 V	650	325
Over voltage	800-860 V	860	430
Over Current	70-285 A	285	285

1.4.2 Task Handling

Project contain different task to perform sequentially. As shown in figure 7.1 Delfino board is Test Jig controller & Piccolo Board is Chopper control Card. Different Task that will be performed by Test Jig controller is as below :

- State 0: Parameter Passing
- State 1: DC link Up
- State 2: Standby
- State 3: Ready
- State 4: Power ON
- Default State: OFF Pass

Test Jig transmit values in first task as parameter passing,if received value is not in range then DSP will set default value & transmit to Test Jig. ADC module will produce result by sampling voltage value & that will be transmitted to Test Jig controller in DC Link up Task.

During DC Link up, DSP controller will monitor DC Link Data & Transmits for

9 sec and automatically switch to stand by state if threshold is achieved. Standby task will transmit ADC sampled value simultaneously as received by ADC Result register. UART transmission done at 100Khz baud rate between Test Jig and SPS Controller. Ready command will be given by Test Jig controller, after receiving this task command, Control card will monitor GPIO Status. LED Display in figure 1.4 will be handle by GPIO Module. Hall Effect sensor will measure current & give value to Control Card. Temperature sensor will measure heat sink temperature. Across Capacitor DC link voltage will be measured by ADC using voltage divider output.

On receiving PWM Pulses from Test Jig control card module will start capturing interrupt with external interrupt module. Duty cycle PWM can be vary from Test Jig during run time. Output voltage at SPS module will be according to ON time period in generated PWM. When Test Jig gives Off Pass command SPS module will come to one step low task as shown in figure 1.3.

1.5 Error Packing & Decoding

If during run time error occurs then health bit in table 1.2 is 1. when error is generated one packet with data shown in 1.3 transmitted to the Test Jig. Packet 1 :

There are two fault levels, level 0 and level 1, as shown in table 1.3. On level 1 fault modules need to go through power on-off cycle.

Table 1.3: Error bits & Fault level

Sr no.	Data Bit	Fault	Fault level
1	Bit 0	Over Current Fault	Level 0
2	Bit 1	Over Voltage Fault	Level 0
3	Bit 2	Over Temperature Fault	Level 0
4	Bit 3	Under Voltage Fault	Level 1
5	Bit 4	Charging Fault	Level 1
6	Bit 5	IGBT short Fault	Level 0
7	Bit 6	WDT fault	Level 1
8	Bit 7	Loss of Communication	Level 0
9	Bit 8	Reserved	Level 0

1.6 Hardware & Software used

C2000 family TI's DSP is used for hardware, Which contains On-board emulation Program and debug your C2000 Launchpad by simply plugging it in to your computer with the included mini USB cable. Support for USB & serial communication is also included. Extend your C2000 Piccolo Launchpad with application focused Booster-packs. All 40 pins on the Launchpad allow for easy access to all the peripherals on the F28027 device.

Why to use TI's DSP?

Pre-programmed C2000 Piccolo F28x MCU Built in isolated XDS100 JTAG Emulator enables real-time in-system programming and debugging via USB CPU reset button and programmable push button enables development on any Piccolo F28x device free unrestricted version of Code Composer Studio integrated development environment (IDE) v6 free download of control SUITE software with examples, libraries, application software and more low Price approximately $17.00.

There are wide range of variety in DSP memory, No of peripherals in C2000 family varies, in this project Ti's DSP used for Compatibility with all MCU. For this project F28027 Piccolo, F28335 Delfino & F2811 Series Controller is being used for

Test Jig & SPS Control card software development.

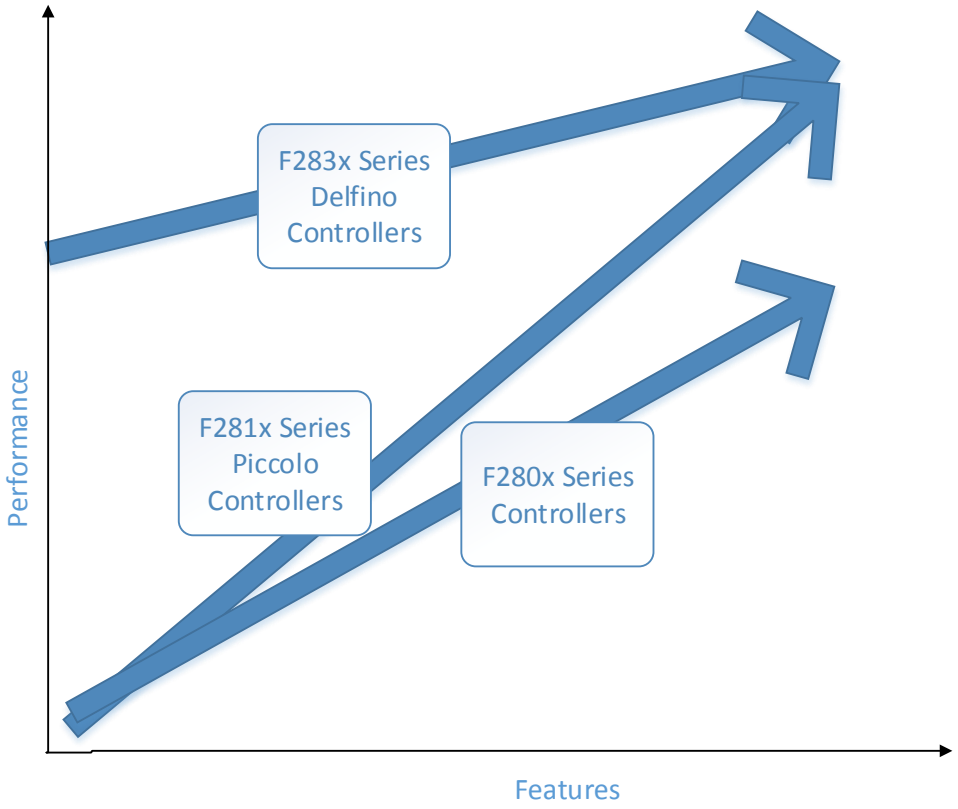


Figure 1.5: C2000 Family TI's DSPs [5]

1.7 Thesis Organization

The rest of the thesis organized as follows.

Chapter 2 This chapter includes basic introduction & Literature Survey of F28x family. Reset circuit, low power mode and other features of TI's DSP is also discussed.

Chapter 3 This chapter deals with different modules of Piccolo launchpad that

we have used in our software development. In this chapter guide is given how we can configure module & use it into our Application.

Chapter 4 This chapter deals with development of C Program. During development of C Program, programmer needs to divide code functionality wise, because it makes easier to edit code & detect the part of code which is not working.

Chapter 5 In this chapter communication with TI Community & answers are included. When programming C software with multiple modules their are possibilities of error generation, TI community provides solution by expert members.

Chapter 7 In this chapter photos of result captured during project work is included. Photos of setup & photos of CCS debug window of project work is also included.

Chapter 8 This chapter describes future scope & the work summary of project that is done.

Chapter A During the beginning of development how to use CCS with launchpad is described in this chapter. Other chapter includes how to configure CCS for launchpad, how to run code & how to debug program and view variable, graph & some features of CCS.

Chapter B This chapter deals with GUI development on GUI Composer provided by CCS. We also learn how to develop GUI for our application using JAVA Script. How can we use pre-processing & post-processing function to see our desire value from range.

Chapter 2

Literature Survey

2.1 Research paper

1 Mandar Bhalekar & Umashankar, **"Development Of a Research Platform for Power Electronic Converter Modeling in Real Time F28335 Digital Simulation Applications using eZDSP"**, IEEE Transaction paper on Power Electronics published in ICICES, Dec 2014

This paper includes requirement of full or partial validation of real time simulation applications.a 3 phase AC-DC-AC converter topology has been used with dc link, diode rectifier and IGBT inverter with inductive load.Open Loop model of AC-DC-AC realization & Design presented in this paper. For PWM generation Delfino F28335 DSP controller is used. The simulation is carried out in MATLAB & SIMULINK and results of simulation and actual model are compared.The proposed PWM algorithm has been validated.

This paper presents the design & analysis of a digitally controlled 3-phase PWM inverter to develop more practical & theoretical knowledge of control applications. The basics of Hardware installations & software optimization for proposed system

have been presented in brief.The simulation results are verified practically using TMS320F28335 DSP at High switching frequency. The very close similarity between actual hardware results of output voltage & simulation wave forms shows the efficiency, accuracy of AC-DC-AC PWM converter.

2 Liran Katzir & Yakir Loewenstern **"Implementation of a High Voltage Power Supply With The Matlab/Simulink Embedded Coder"**, IEEE Transaction paper on Power Electronics published in Convention of Electrical and Electronics Engineers in Israel, Jun 2014

High voltage power supply with a voltage multiplier can be developed by using a DSP-based controller.DSP controller can give the output voltage back to the PWM signals of the power stage.By using the MATLAB & Simulink embedded coder design time is greatly reduced. The design with a DSP controller, does not need additional components and is simple compared to more traditional analog control techniques. Practical results using the Texas Instruments (TI) DSP TMS320F28335 Micro-controller show a good real-time behavior. For example, the output of a 12-link multiplier with a rise time lower than 5mS and a low ripple of under 0.1% is actively corrected by the digital feedback loop.

High voltage power supply, using a voltage multiplier controlled by a DSP-based controller was presented. The closed feedback loop was implemented by sampling the output, processing in the controller and correcting by manipulating the PWM phase-shift signal. The design was done using using the MATLAB/Simulink embedded coder, greatly reducing the design time. The design is relatively simple and can be implemented using a low cost controller. Experimental results of the high voltage power supply using the DSP TMS320F28335. Microcontroller from TI show a a good convergence and real time behavior with a large reduction of

output ripple.

3 Michael G. Giesselmann & William J. Carey **"100-kV High Voltage Power Supply With Bipolar Voltage Output and Adaptive Digital Control"**, IEEE Transaction paper on Plasma Science published in OCT 2014

This paper presents a 100-kV high frequency transformer/ rectifier package, which is capable of a dual output polarity operation. An H-Bridge inverter drives the primary of the high voltage (HV) transformer at a frequency of 20 kHz. The inverter is driven by a Microchip dsPIC33F digital signal controller using peak current mode control with adaptive slope compensation. The HV-tank has two HV-coax output cables with a grounded shield on each cable. If the center conductor of the coax cable designated as negative output is grounded, positive voltage is obtained from the coax cable designated as positive output and vice versa. This paper provides design details and experimental results from tests of the entire system.

2.2 Architecture of C2000 Family

This architecture is common for all C2000 family devices. This is multi-bus architecture & also known as harvard architecture. Program bus & Data bus connected as shown in figure 2.1 by different bus which enhance performance of device. In figure 2.1 upper left corner includes memory like RAM, Boot ROM & Sectored Flash. In lower left diagram we can see execution section which includes Auxiliary Register, Multiplier, Atomic ALU(Arithmetic Logic Unit) & CLA(Control Low Accelerator) is independent unit which have own sets of buses. PIE(Peripheral interrupt Expansion) manager & 3 CPU timer is also available in architecture.

At upper right diagram consist control Unit & lower right diagram consist commu-

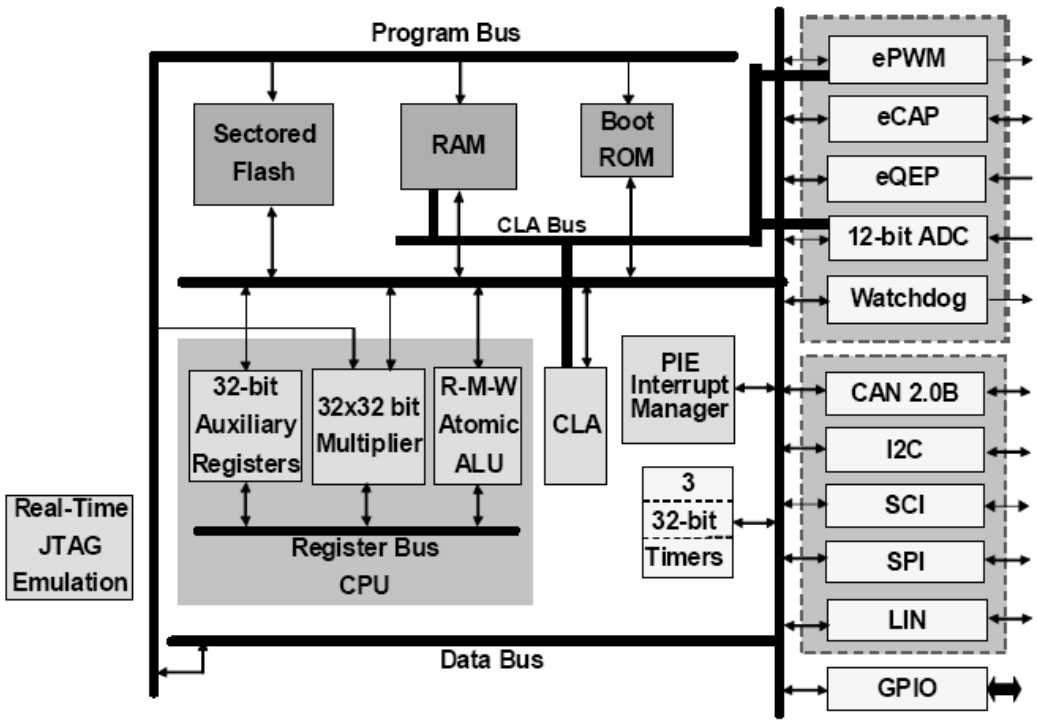


Figure 2.1: F2802x Architecture [4]

nication unit. Control Unit includes peripheral like ePWM(Enhanced pulse width modulation), eCAP(enhanced capture Capture module), eQEP(Enhanced Quadrature Encoder Pulse), 12bit ADC & Watchdog. Communication Unit includes peripheral like A CAN(controller area network), I2C(Inter-Integrated Circuit), SCI(Serial Communications Interface), SPI(Serial Peripheral Interface), LIN(Local Interconnect Network) & GPIO(General purpose input output).

2.3 Introduction to Piccolo Launchpad

The C2000 Piccolo Launchpad is an evaluation platform that allows the user to practice real-time control programming on the C2000 Piccolo micro controllers.

The Launchpad is based on the Piccolo TMS320F28027 with features such as 12bit ADC, 8PWM channels, I2C, SPI, UART, and 64KB of on board flash memory, etc.

It utilizes an integrated isolated Cross Document Sharing(XDS)100 Joint Test Action Group(JTAG) emulator for easy programming and debugging. In addition, it also has 40 Printed Circuit Board(PCB) pins that are available for easy access to the pin of the F28027 microcontroller and programmable/reset button.

2.3.1 Low Power Modes

Table 2.1: Low Power Modes [7]

Low Power Mode	**CPU clock**	**Peripheral Clock**	**Watchdog Clock**	**PLL/OSC**
Normal Run	ON	ON	ON	ON
IDLE	OFF	ON	ON	ON
Standby	OFF	OFF	ON	ON
HALT	OFF	OFF	OFF	OFF

Piccolo Launchpad support different low power modes in each power mode their are distinct configuration for clocks. In normal mode all peripheral like CPU clock, Peripheral clock, Watchdog clock & PLL clock is ON which consumes more power then Other configuration. Ideal mode save power more then normal run by turning CPU clock off. If peripheral clock is turned off in idle mode it comes into standby mode. Switching off all clock module it saves all power & come into halt Mode.

Table 2.2: Exit Low Power Modes [7]

Exit Low Power Mode	**RESET**	**GPIO A**	**WD INT**	**Any INT**
IDLE	YES	YES	YES	YES
Standby	YES	YES	YES	NO
HALT	YES	YES	NO	NO

When code is in any LPM(Low Power Modes) of Launchpad to exit from that mode needs to give instruction to launch pad peripheral. Reset, GPIO, Watch-dog

interrupt or any interrupt can be used to exit Idle mode. To exit from Standby mode Reset, GPIO or Watch-dog interrupt can be used. To exit from halt mode Reset or GPIO can be used.

2.3.2 Reset Circuit

Reset is process to reboot system to get back default configuration. Piccolo launchpad allows different ways to reset it. Hardware logic of Launchpad for reset. If during initialization Missing clock detect it will automatically reset software. Watchdog Timer, Power on Reset, Brown out reset Or External reset(XRS) Pin Active reset to F28x device can be also configured.

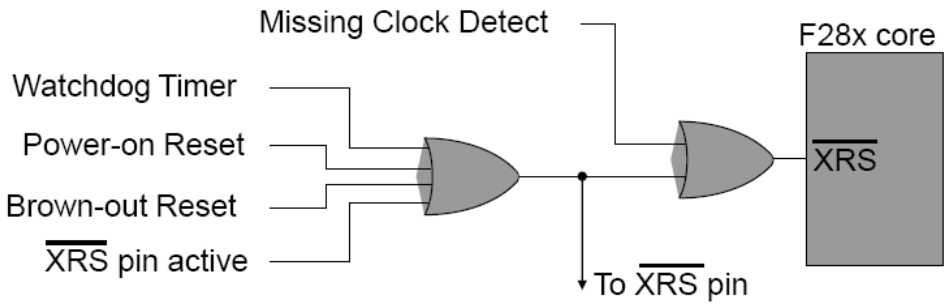


Figure 2.2: Reset Circuit [6]

2.4 Different programming Style

1 Direct Register Programming

As the name suggest, the Direct Register Access Model (DRAM)writes value directly to the individual peripheral registers. All of the peripheral registers are defined in the corresponding header file for specific device. It is therefore important for the user to include such header file along with the developed program. An example of DRAM is demonstrated as follow:

AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;

AdcRegs.ADCINTFLGCLR.bit.ADCINT2 = 1;

This particular code clears the ADC interrupt flag for ADC interrupt 1 and 2.

2 Software Programming

The Software Driver Model (SDM) utilize an API(application programming interface) provided by the peripheral driver library. In which can be used by applications to control peripherals. The driver header file needs to be included in this case to utilize SDM and a handle to specific peripheral needs to be initialized. An example of SDM is shown as follow:

$myPwm1 = PWM\_init((void*)PWM\_ePWM1\_BASE_ADDR, sizeof(PWM\_Obj))$;
$myPwm2 = PWM\_init((void*)PWM\_ePWM2\_BASE_ADDR, sizeof(PWM\_Obj))$;

2.5 Summary

This chapter includes basic introduction, IEEE Transaction paper on power electronic & Literature Survey of F28x family. Reset circuit, low power mode and other features of TI's DSP is also discussed.

Chapter 3

Modules of F28x MCU used in project

3.1 GPIO Module

GPIO module consist three GPIO port. GPIO A, GPIO B & Analog Port. A single GPIO pin is multiplexed Up to three independent level. This three level are MUX selection, Direction Selection & port Configuration. Port A consists of GPIO0 to GPIO31, port B consists of GPIO32 to GPIO38 & The analog port consists of AIO0 to AIO15 as shown in figure 3.1.

In figure 3.2 MUX Control bits are available. To use Port as Gpio by default value is given as 00. To use it for different peripheral like Epwm, ADC, SCI or more PORT MUX bit for appropriate pin can be configured. After MUX selection to configure GPIO for either as input or as output by GPxDIR. By using GPxDAT GPIO can be ON, OFF or Toggle Gpio as Configuration.

Example 3.1 :
Write A Program of C2000(TMS320F28027) Launchpad Evaluation Kit to GPIO toggling on GPIO port 0,1,2 and 3 to blink LED.

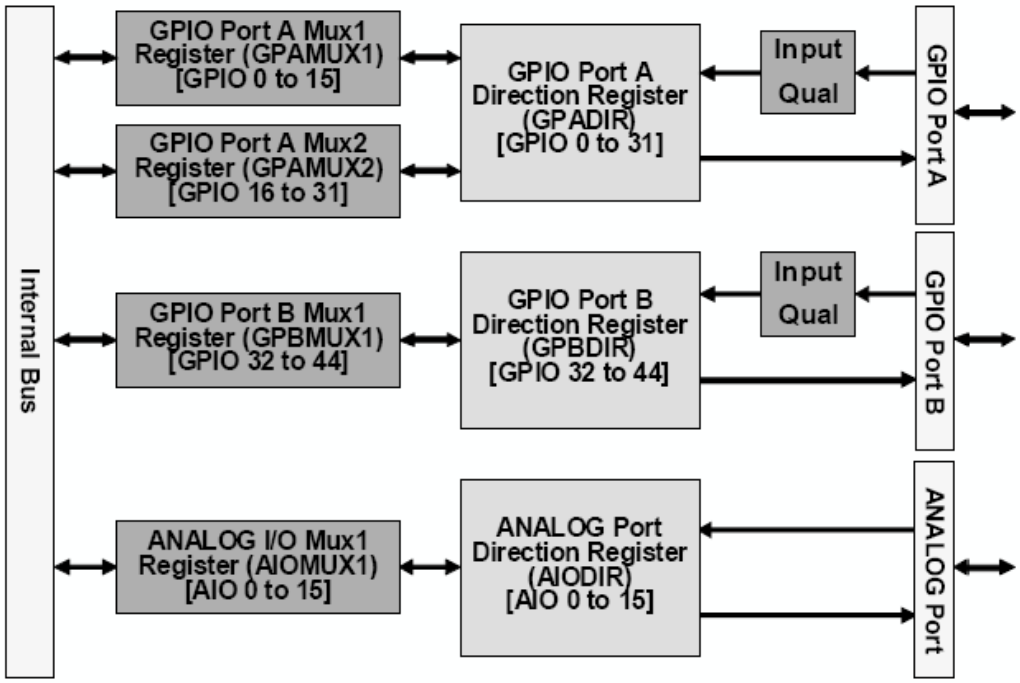


Figure 3.1: GPIO Module [6]

```
EALLOW;
SysCtrlRegs.WDCR = 0x0068;
//Write same for GPIO 1, 2 & 3
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0; // 0=GPIO, 1=EPWM1A, 2=Resv, 3=Resv
GpioCtrlRegs.GPADIR.bit.GPIO0 = 1; // 1=OUTput, 0=INput
EDIS;
//Write same for GPIO 1, 2 & 3
GpioDataRegs.GPASET.bit.GPIO0 =1;
while(1)
{
//Write same for GPIO 1, 2 & 3
GpioDataRegs.GPATOGGLE.bit.GPIO0 = 1;
DELAY_US(1000000);
}
```

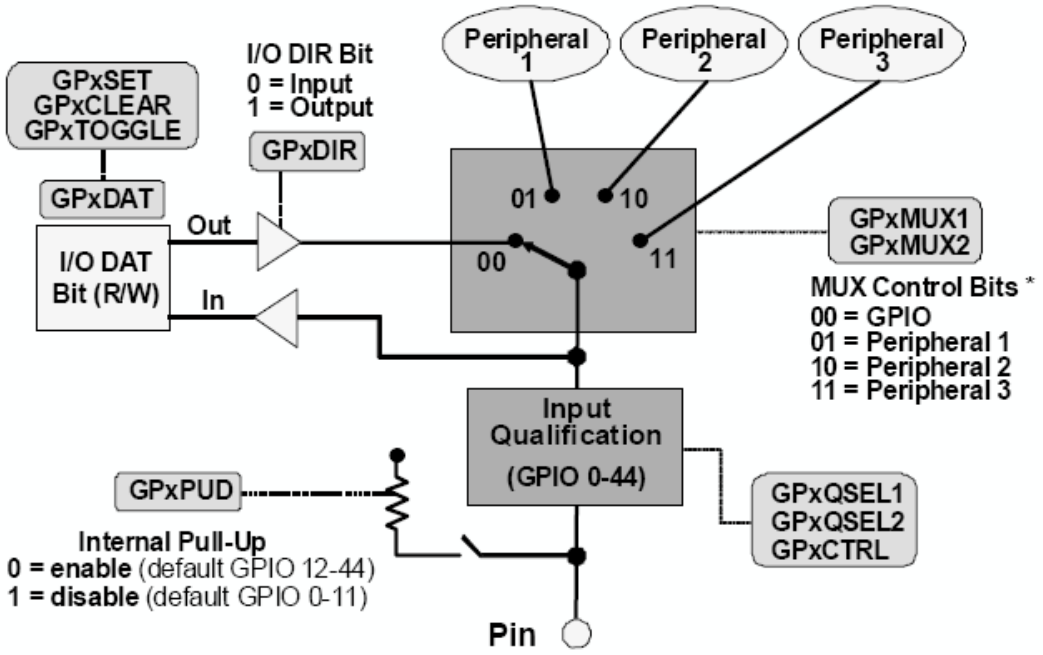


Figure 3.2: GPIO Mux Selection [6]

GUIDE : Now available register to access are EALLOW Protected, so to get access write EALLOW at beginning of code.

After that SysCtrlRegs.WDCR = 0x0068; command to disable watchdog, so it may not run at debug time.
Which indicate bit 3,5 and 6 are high. Means WDCHK =101 it must ALWAYS write 1,0,1 to these bits whenever a write to this register is performed unless the intent is to reset the device via software. Then WDDIS= 1 indicate watch dog Disable.

Now to access GPIO for it first GpioCtrlRegs in which select GPAMUX1 and set GPIO 0 and 2 as general purpose GPIO to be accessed. Then to give direction as Output to get LED Blink on that port GpioCtrlRegs.GPADIR.bit.G- PIO0 = 1; By using this given direction as output can be set.
The use infinite while Loop for LED blink on GPIO port 0 and 2.
Here we have set toggle bit as 1 of GPIO 2 then some delay then set toggle bit as 1

of GPIO 0 then delay.

NOTE :

1 Delay_US function will not perform if watchdog is not disabled.

2 After that initialize all GPIO and turn it off by GpioDataRegs.GPASET.bit.GPIO0 =1; at starting so it GPIO will not blink during the run.

3.2 CPU Timer Module

Piccolo launchpad have 3 CPU timer to perform timing operations. To Acknowledge Timer 0 use TINT0 using PIE table. CPU timer 1 & 2 can be acknowledge by TINT1 & TINT2 or XINT13 & XINT14. To use CPU timer 1 & 2 to acknowledge TINT0 is necessary.

Example 3.2 :

Write A Program of C2000(TMS320F28027) Launchpad Evaluation Kit to generate interrupt using CPU timer 0 and 1.

```
//First Include Header files. __interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
void main(void)
{
InitSysCtrl();
DINT; // Disable CPU interrupts and clear all CPU interrupt flags:
InitPieCtrl();
IER = 0x0000;
IFR = 0x0000;
InitPieVectTable();
EALLOW; // This is needed to write to EALLOW protected registers
PieVectTable.TINT0 = &cpu_timer0_isr;
```

```
PieVectTable.TINT1 = &cpu_timer1_isr;
EDIS; // This is needed to disable write to EALLOW protected registers
InitCpuTimers(); // For this example, only initialize the Cpu Timers
// Configure CPU-Timer 0 and 1 to interrupt every second:
// 60MHz CPU Freq, 1 second Period (in uSeconds)
ConfigCpuTimer(&CpuTimer0, 60, 1000000);
ConfigCpuTimer(&CpuTimer1, 60, 1000000);
CpuTimer0Regs.TCR.all = 0x4001;
// Use write-only instruction to set TSS bit = 0
CpuTimer1Regs.TCR.all = 0x4001;
// Use write-only instruction to set TSS bit = 0
IER |= M_INT1;
IER |= M_INT13;
// Enable TINT0 in the PIE: Group 1 interrupt 7
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
EINT; // Enable Global interrupt INTM
ERTM; // Enable Global real time interrupt DBGM
for(;;);
}

__interrupt void cpu_timer0_isr(void)
{
CpuTimer0.InterruptCount++;
// Acknowledge this interrupt to receive more interrupts from group 1
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
__interrupt void cpu_timer1_isr(void)
{
EALLOW;
```

CpuTimer1.InterruptCount++; // The CPU acknowledges the interrupt.

EDIS;

}

GUIDE :

Now For Programming CPU timer the steps to follow :

Step 1: Initialize system control.

Step 2: Initialize Gpio.

Step 3: Disable all interrupts and Initialize PIE vector table.

Step 4: Initialize Device Peripheral.

Step 5: Use code to enable specific interrupt.

Step 6: Idle loop forever.

Now, here cpu_timer0_isr(); is function declaration.

InitPieCtrl();

This is a function that is provided into TIs header file. Use this function as it is. The purpose of this function is to clear all pending PIE(Peripheral Interrupt Expansion)-Interrupts and to disable all PIE interrupt lines. This is a useful step when we would like to initialize the PIE-unit. Function InitPieCtrl() is defined in the source code file DSP281x_PieCtrl.c; Inside main, direct after the function call InitPieCtrl(); add the function call to:

InitPieVectTable();

This TI-function will initialize the PIE-memory to an initial state. It uses a predefined interrupt table PieVectTableInit() defined in source code file DSP281x_PieVect.c and copies this table to the global variable PieVectTable defined in DSP281x_GlobalVariableDefs.c. Variable PieVectTable is linked to the physical memory of the PIE area. To be able to use InitPieVectTable should be added two more source files to our project:

Source Code file DSP281x_DefaultIsr.c will add a lot of interrupt service routines to our project. When you open and inspect this file you will find that all ISRs consist of an endless for-loop and a specific assembler instruction ESTOP0. This instruction behaves like a software breakpoint. This is a security measure. Remember, at this point we have disabled all PIE interrupts. During run the program can't see an interrupt request. If for some reason like a power supply glitch, noise interference or just a software bug, the DSP calls an interrupt service routine then this event can be catch by the ESTOP0 break.

InitCpuTimers();

This function will set the core Timer0 to a known state and it will stop this timer.

ConfigCpuTimer(&CpuTimer0, 60, 1000000);

The 60 in configuration indicate frequency in MHz and 1000000 is in uS means it is delay of 1 Second.

PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

Before starting timer0 needs to enable its interrupt masks.

To enable CPU timer interrupt set bit 7 of PIEIER1 to 1.

EINT; and ERTM; Enable interrupts globally

CpuTimer0Regs.TCR.bit.TSS = 0;

Now to start the timer 0, the bit TSS inside register TCR will do this task. After the end of main to add our new interrupt service routine cpu_timer0_isr.

1 Increment the interrupt counter CpuTimer0.InterruptCount. This gives number of global information about how often this 1 Sec task was called.

2 Acknowledge the interrupt service as last line before return. This step is necessary to re-enable the next timer0 interrupt service. It is done by: PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

for(;;); Use for loop forever

3.3 Watchdog Module

Watch dog is a timer which detect Program malfunctions & reboot our system for Recovery after predefined time out period.

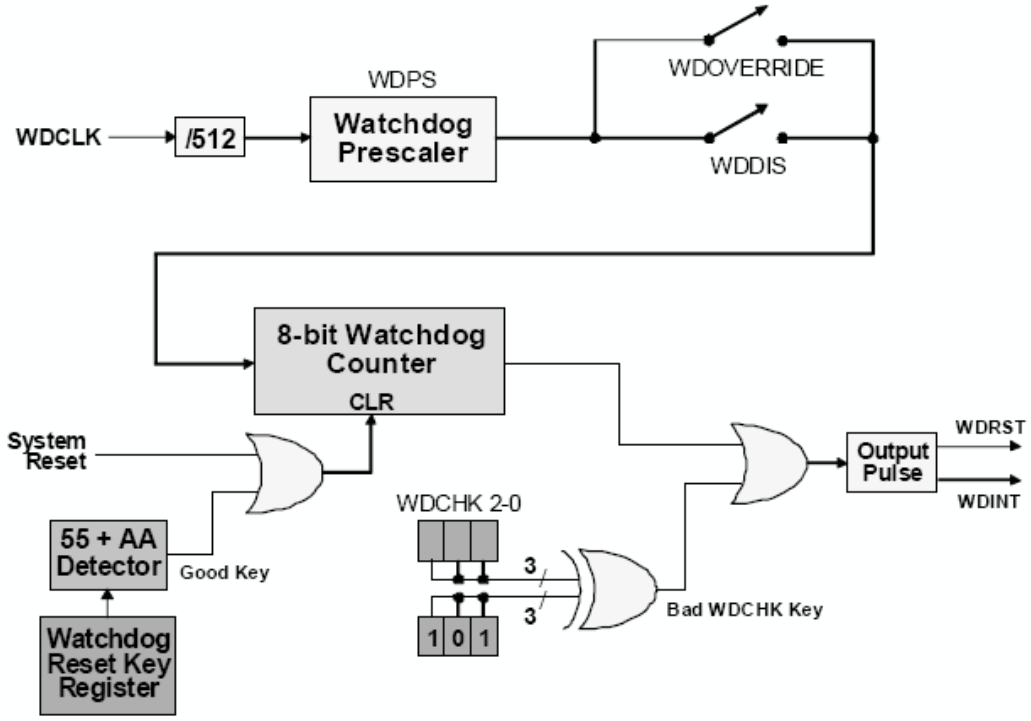


Figure 3.3: Watchdog Select[6]

Watch dog can be configured as shown in figure 3.3. When Good Key $55 + AA$ detect before prescale counter generate timeout then the reset interrupt will not be generated & Counter will be reloaded. But if Other then this key will detect then after timeout **WDINT** will be generated. WDCHK bits from 2-0 will check always during the program if other then 101 WDCHK will be detected then it will Generate **WDRST** signal.

Example 3.3 :

#include "DSP28x_Project.h" // Device Headerfile and Examples Include File

```
interrupt void wakeint_isr(void);
Uint32 WakeCount=0,n=0;
Uint32 LoopCount=0;

void main(void)
{
// Common Flow of System Initialization
EALLOW; // This is needed to write to EALLOW protected registers
PieVectTable.WAKEINT = &wakeint_isr;
SysCtrlRegs.SCSR = BIT1;
EDIS;

PieCtrlRegs.PIECTRL.bit.ENPIE = 1; // Enable the PIE block
PieCtrlRegs.PIEIER1.bit.INTx8 = 1; // Enable PIE Gropu 1 INT8
IER |= M_INT1; // Enable CPU INT1
EINT; // Enable Global Interrupts

ServiceDog();
EALLOW;
SysCtrlRegs.WDCR = 0x0028; // SET timer period
EDIS;

for(;;) // loop forever
{
LoopCount++;
// DELAY_US(13107); // 13.11ms interrupt Period for WDCR = 28
DELAY_US(13108);
ServiceDog();
}
```

```
}

interrupt void wakeint_isr(void)
{
WakeCount++;
// Acknowledge this interrupt to get more from group 1
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

GUIDE : First to Initialize System peripheral like PLL(Phase Locked Loop), WatchDog, enable peripheral clocks functions for this program located in the DSP2802x_SysCtrl.c file.
Clear all interrupts and initialize PIE vector table & disable CPU interrupts. Initialize PIE control registers to their default state. This function is found in the DSP2802x_PieCtrl.c file.

Disable CPU interrupts and clear all CPU interrupt then initialize the PIE vector table with pointers to the shell Interrupt Service Routines (ISR). This is useful for debug purposes. The shell ISR routines are found in DSP2802x_DefaultIsr.c. This function is found in DSP2802x_PieVect.c.

Interrupts that are used in this example are re-mapped to ISR functions found within this file. Connect the watchdog to the WAKEINT interrupt of the PIE Write to the whole SCSR register to avoid clearing WDOVERRIDE bit.

Enable WAKEINT in the PIE: Group 1 interrupt 8 & Enable INT1 which is connected to WAKEINT.

- Reset the watchdog counter, Enable the watchdog.

- Un-comment Service Dog to just loop here.
- Comment Service Dog to take the WAKEINT instead.

Insert all local Interrupt Service Routines (ISRs) and functions here. If local ISRs are used, reassign vector addresses in vector table.

3.4 Serial Communication Interface(SCI) Module

SCI module of Piccolo Launchpad is useful for serial communication between controller. Port as SCI can be configured by proper MUX bits selection.

SCI module is a serial I/O port that permits Asynchronous communication between the C28x and other peripheral devices. The SCI transmit and receive registers are both double-buffered to prevent data collisions and allow for efficient CPU usage. In addition, the C28x SCI is a full duplex interface which provides for simultaneous data transmit and receive. Parity checking and data formatting is also designed to be done by the port hardware, further reducing software overhead.

SCI is receiving LSPCLK means if CPU Clock is 60MHz then we can use SCICLK up to (60/4)MHz = 15MHz.

Example 3.4 :
Write A Program of C2000(TMS320F28027) Launchpad Evaluation Kit to generate Array and transmit to SCIA port with 9600 Baud rate, 1 start bit, 8 data bit, 1 stop bit with no parity and Receive it with External LOOP Back.
#include "DSP28x_Project.h" // Device Header file and Examples Include File
void scia_echoback_init(void);
void scia_fifo_init(void);
#define CPU_FREQ 60E6

```
#define LSPCLK_FREQ CPU_FREQ/4
#define SCI_BAUD 96E2 // 9600 baudrate
#define SCI_BRR (LSPCLK_FREQ/(SCI_BAUD*8))-1
Uint16 ReceivedChar[10]= {0,0,0,0,0,0,0,0,0,0},i;
Uint16 SendChar[10]={0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A};
void main(void)
{
InitSysCtrl();
InitSciaGpio(); // initialize GPIO Port 28 & 29
DINT;
InitPieCtrl();
IER = 0x0000;
IFR = 0x0000;
InitPieVectTable();
scia_fifo_init(); // Initialize the SCI FIFO
scia_echoback_init(); // Initalize SCI for echoback
for(;;){
for(i = 0; i < 10; i++)
{
SciaRegs.SCITXBUF =SendChar[i]; // Transmitter Code
while(SciaRegs.SCIFFRX.bit.RXFFST !=1) { } // Receiver Code
// wait for XRDY =1 for empty state
ReceivedChar[i] = SciaRegs.SCIRXBUF.all;
}
}
}

void scia_echoback_init()
{
```

```
SciaRegs.SCICCR.all =0x0007; // 1 stop bit, No loopback, No parity,8 char bits,
// async mode, idle-line protocol
SciaRegs.SCICTL1.all =0x0003; // enable TX, RX, internal SCICLK,
// Disable RX ERR, SLEEP, TXWAKE
SciaRegs.SCICTL2.all =0x0000;
SciaRegs.SCICTL2.bit.TXINTENA =0;
SciaRegs.SCICTL2.bit.RXBKINTENA =0;

#if (CPU_FRQ_60MHZ)
SciaRegs.SCIHBAUD =0x0000; // 9600 baud @LSPCLK = 15MHz (60 MHz SYSCLK).
SciaRegs.SCILBAUD =SCI_BRR;
#endif
SciaRegs.SCICCR.bit.LOOPBKENA =0; // Disable loop back
SciaRegs.SCICTL1.all =0x0023; // Relinquish SCI from Reset
}

void scia_fifo_init()
{
SciaRegs.SCIFFTX.all=0xE040;
SciaRegs.SCIFFRX.all=0x2044;
SciaRegs.SCIFFCT.all=0x00;
}
```

GUIDE : First include header file & declare function scia_echoback_init & scia_fifo_init. Now define CPU frequency as 60MHz,so LSPCLK is 15MHz. Define baud rate as 9600 & last function to calculate SCI BRR. After that initialize variable. Add common flow of system initialization. Write code to initialize SCI Fifo & SCI configuration.

The SCI for External loopback is Configured. So, connect GPIO port 28 & 29. Now in the infinite "for loop" start to transmit data From Tx pin & by using external loopback receive same data at Rx Pin.

SCI FIFO configuration is done by function scia_echoback_init & scia_fifo_init. In scia_echoba- ck_init function Frame structure ,enable Tx, Rx & defined sci low baud rate value is defined. First disabled internal loopback. Then in function scia_fifo_init Tx & Rx fifo and given delay between data frame zero is configured. To monitor transmitted and received data add variable SendChar & ReceivedChar into Expression window of CCS Debug mode.

3.5 Enhanced Pulse width modulation(EPWM) Module

Enhanced pulse width modulation(EPWM) represents signal as sequence of pulses. PWM have fixed frequency & fix pulse amplitude. When using power switching device such as transistor for desire current or power supply, it is difficult to operate this device in proportional region & easy to control in saturation region.

EPWM is digital signal & is to generate with MCU, it can control properly power switching device. Generated PWM output is also available to the GPIO pins. EPWM module can generate INT for PIE block as well as Start of conversion(SOC) signal for ADC.

Clock prescaler convert SYSCLKOUT signal into timer based clock as shown in figure 3.4. The 16-bit time base counter can generate symmetric & Asymmetric waveform using 3 different mode count up,count down & count up-down mode. Period register shows maximum count value. Compare logic use two compare register

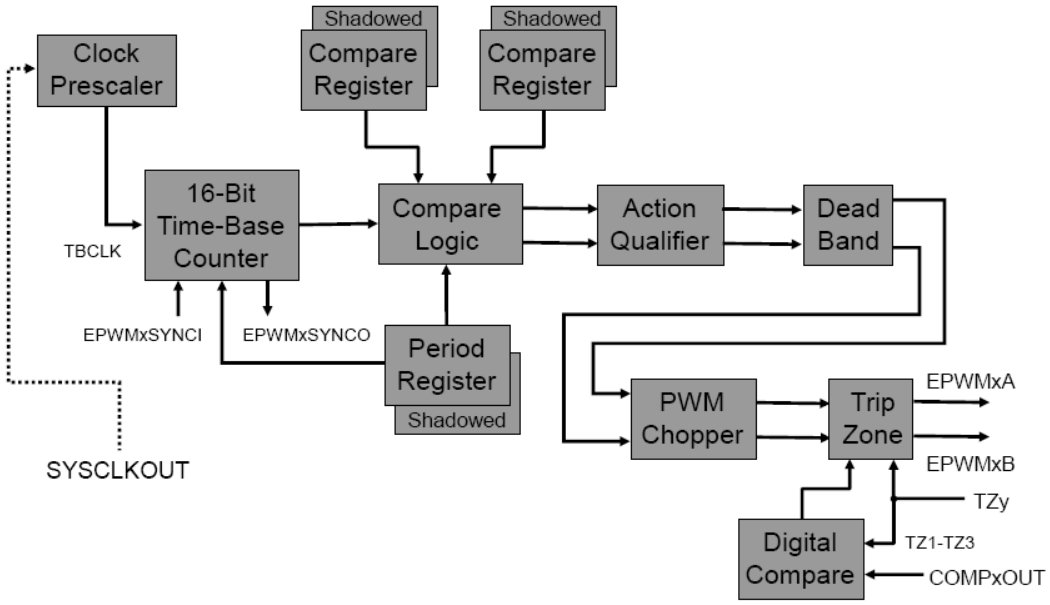


Figure 3.4: EPWM Module[6]

to compare signal from time base counter. Action qualifier are use to generate action using input from Compare logic. Dead band add delay in switching gate signal & prevent short circuit. PWM chopper use high frequency to modulate PWM waveform Trip zone & Digital compare provide protection mechanism to protect Output Pins from over voltage & over current.

Example 3.5 :

```
#include "Lab.h" // Main include file
#define PWM_HALF_PERIOD 1000 // period/2 for 15 kHz symmetric PWM
#define PWM_DUTY_CYCLE 500 // 50% duty cycle
int x;


void main(void)
{
// Do common system initialize here.
InitEPwm(); // Initialize the EPwm (FILE: EPwm.c)
```

```
asm(" EALLOW"); // Enable EALLOW protected register access
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0; // 0=GPIO 1=EPWM1A
GpioCtrlRegs.GPADIR.bit.GPIO1 = 0; // 0=input 1=output
asm(" EDIS"); // Enable EALLOW protected register access
asm(" CLRC INTM, DBGM"); // Enable global interrupts and realtime debug

while(1) // endless loop - wait for an interrupt
{
// J6.1 GPIO0 EPWM1A to J6.2 GPIO1
x = GpioDataRegs.GPADAT.bit.GPIO1; // GPIO1 pin
}
}
```

The InitEpwm function written in EPWM.c file which include EPWM Configuration is as below

```
void InitEPwm(void)
{
asm(" EALLOW"); // Enable EALLOW protected register access
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
asm(" EDIS"); // Disable EALLOW protected register access

//Configure ePWM1 for 15 kHz symmetric PWM on EPWM1A pin
EPwm1Regs.TBCTL.bit.CTRMODE = 0x3; // Disable the timer

EPwm1Regs.TBCTL.all = 0xC033; // Configure timer control register
// bit 5-4 11: SYNCOSEL, 11 = sync-out disabled
// bit 1-0 11: CTRMODE, 11 = timer stopped (disabled)
```

```
EPwm1Regs.TBCTR = 0x0000; // Clear timer counter
EPwm1Regs.TBPRD = PWM_HALF_PERIOD; // Set timer period
EPwm1Regs.TBPHS.half.TBPHS = 0x0000; // Set timer phase
EPwm1Regs.CMPA.half.CMPA = PWM_DUTY_CYCLE; // Set PWM duty cycle

EPwm1Regs.CMPCTL.all = 0x0002; // Compare control register
// bit 1-0 10: LOADAMODE, 10 = load on PRD match

EPwm1Regs.AQCTLA.all = 0x0060; // Action-qualifier control register A
// bit 7-6 01: CAD, 01 = clear
// bit 5-4 10: CAU, 10 = set
EPwm1Regs.AQSFRC.all = 0x0000; // Action-qualifier s/w force register
EPwm1Regs.AQCSFRC.all = 0x0000; // Action-qualifier continuous s/w force reg-
ister
EPwm1Regs.DBCTL.bit.OUT_MODE = 0; // Deadband disabled
EPwm1Regs.PCCTL.bit.CHPEN = 0; // PWM chopper unit disabled
EPwm1Regs.TZDCSEL.all = 0x0000;
// All trip zone and DC compare actions disabled
EPwm1Regs.TBCTL.bit.CTRMODE = 0x2;
// Enable the timer in count up/down mode

asm(" EALLOW"); // Enable EALLOW protected register access
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
// TBCLK to ePWM modules enabled
asm(" EDIS"); // Disable EALLOW protected register access
} // end InitEPwm()
```

GUIDE : In this example 15Khz symmetric waveform is generated using EPWM1A. In the main.c file of example first initialized all peripheral of launchpad with our

epwm Configuration, then take sample by using GPIO. After initEPWM set GPIO1 as input pin. Sampling rate of GPIO is low so this can't give accurate output. Connect Pin J6.1 to J6.2 to see output on GPIO1 port on debug window.

In initEWM function first disable timer, then set timer period & duty cycle. CMPCTL bit for load configured on period match.

3.6 Analog to Digital Converter(ADC) Module

ADC module is 12 bit analog to digital converter. ADC module have 16 input channel & 16 result register as shown in figure 3.5. SOC Configuration select trigger source, channel to convert & acquisition prescale window size that can be triggered by Software, CPU timer, EPWM Or using external signal. ADCINT1 & ADCINT2 is fade back to continue conversion. ADC can be operated in Sequential Or Simultaneous mode. ADC Clocking is done by CPU Clock at 60MHz.

3.6.1 ADC Mode

1 ADC Sequential mode is used while sampling using only channel A. Total sample time is 13 cycles. By using 60MHz CPU clock maximum sampling rate of 4.62 mega samples per second(MSPS) as calculated {60MHz/(13cycles/sample)} = 4.62 MSPS can be achieved.

2 ADC Simultaneous mode is used while sampling by using both channel A & Channel B simultaneously. Total sample time is 26 cycles for 2 sample. By using 60MHz CPU Clock maximum sampling of 4.62 MSPS as {60MHz/(13cycles/sample)} =

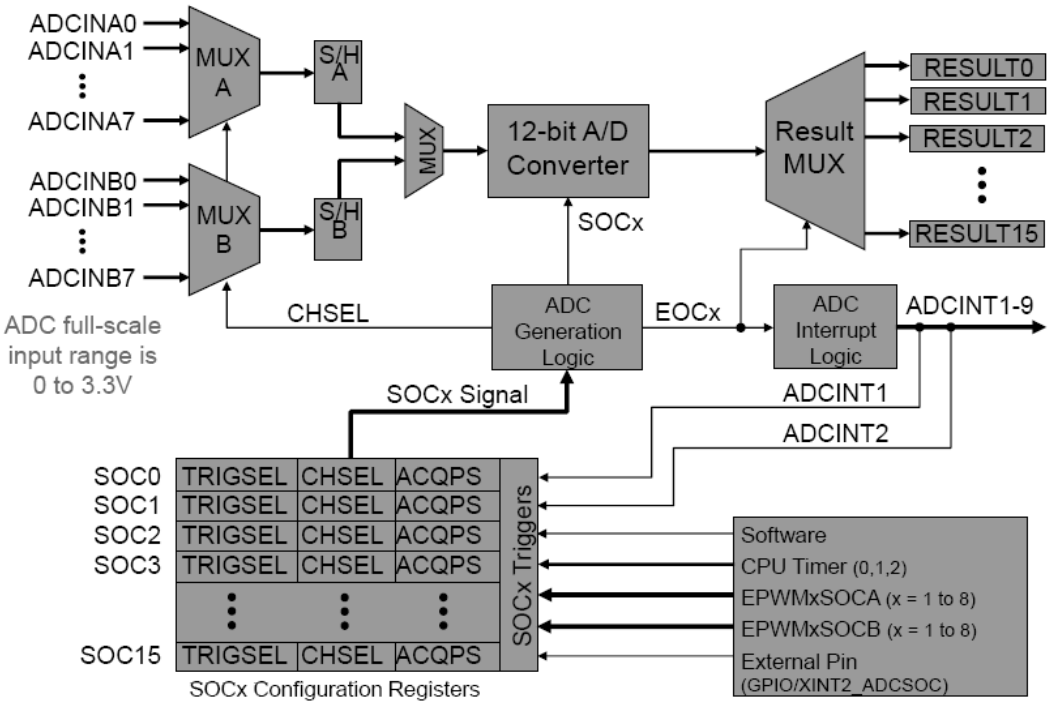


Figure 3.5: ADC Module [6]

4.62 MSPS can be achieved.

Sequential mode can be trigger by AdcRegs.ADCINTSOCSELx & Simultaneous mode can be trigger by AdcRegs.ADCSAMPLEMODE register.

Example 3.6 :

Write a program to generate waveform using EWM1A at GPIO0 as shown in figure 3.8 with period 15Khz and sample it by using ADCINA0 by triggering it with EPWM2A at 50KHz.

In this example to generate waveform to be sample by using EPWM1A and for Triggering ADC use EPWM2A. ADC Result will be copied by CPU to Data memory when ADC ISR is in execution. In this example to take AdcBuf[ADC_BUF_LEN] variable is required which will continuous print our data memory in to Array with

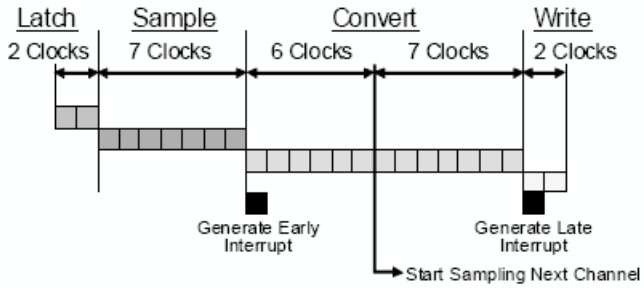


Figure 3.6: ADC Sequential Mode[4]

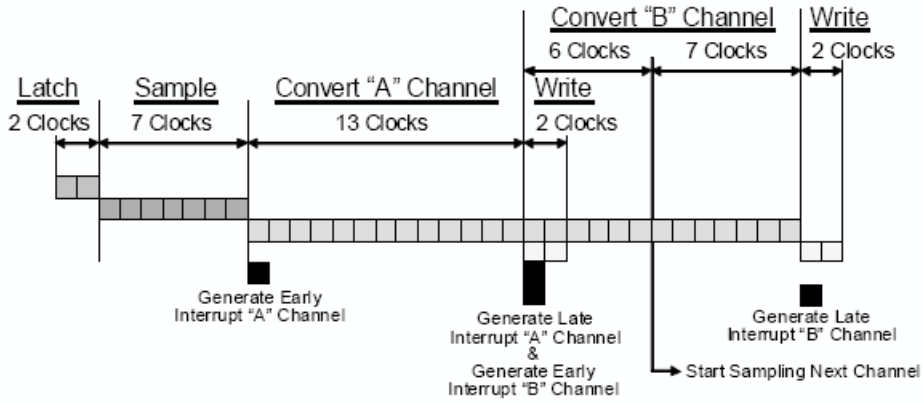


Figure 3.7: ADC Simultaneous Mode [4]

50 array element.

Main() function includes system initialization first, after that initialize EPWM1A with GPIO0 for sample wave and for triggering ADC initialize EPWM2A.

```
#include "Lab.h" // Main include file
#define PWM_HALF_PERIOD 1000 // period/2 for 15 kHz symmetric PWM
#define PWM_DUTY_CYCLE 500 // 50% duty cycle
Uint16 AdcBuf[ADC_BUF_LEN]; // ADC buffer allocation
Uint16 DEBUG_TOGGLE = 1; // Used for realtime mode investigation test
Uint16 adcsamp,x;

void main(void)
```

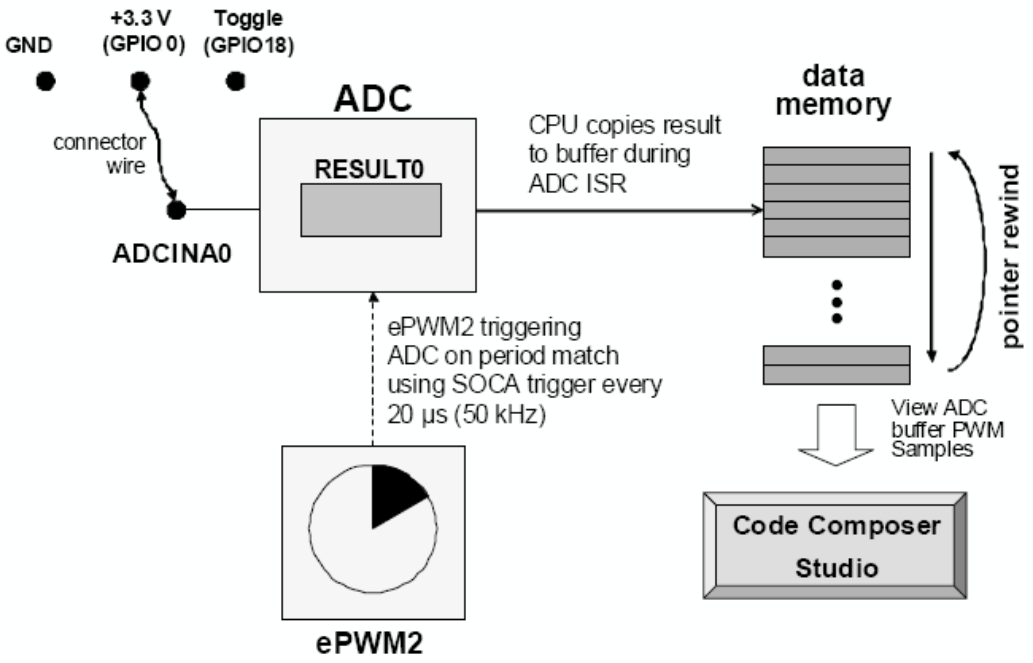


Figure 3.8: ADC Example [7]

```
{
// Do common system initialize here.
InitEPwm(); // Initialize the EPwm (FILE: EPwm.c)
InitAdc(); // Initialize the EPwm (FILE: Adc.c)

asm(" EALLOW"); // Enable EALLOW protected register access
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0; // 0=GPIO 1=EPWM1A
GpioCtrlRegs.GPADIR.bit.GPIO1 = 0; // 0=input 1=output
asm(" EDIS"); // Enable EALLOW protected register access
asm(" CLRC INTM, DBGM"); // Enable global interrupts and realtime debug

while(1) // endless loop - wait for an interrupt
  {
  // J6.1 GPIO0 EPWM1A to J5.6 ADCINA0
```

```
  adcsamp = AdcResult.ADCRESULT0; // GPIO1 pin
  }
}
```

The InitAdc function written in Adc.c file which include ADC configuration is as below

```
void InitAdc(void)
{
asm(" EALLOW"); // Enable EALLOW protected register access
//— Reset the ADC module
AdcRegs.ADCCTL1.bit.RESET = 1; // Reset the ADC
// Must wait 2 ADCCLK periods for the reset to take effect.
asm(" NOP");
asm(" NOP");

//— Power-up and configure the ADC
AdcRegs.ADCCTL1.all = 0x00E4; // Power-up reference and main ADC
DelayUs(1000); // Wait 1 ms after power-up before using the ADC
//— SOC0 configuration
AdcRegs.ADCSAMPLEMODE.bit.SIMULEN0 = 0; // SOC0 in single sample mode
(vs. simultaneous mode)
AdcRegs.ADCSOC0CTL.bit.TRIGSEL = 7; // Trigger using ePWM2-ADCSOCA
AdcRegs.ADCSOC0CTL.bit.CHSEL = 0; // Convert channel ADCINA0 (ch0)
AdcRegs.ADCSOC0CTL.bit.ACQPS = 6; // Acquisition window set to (6+1)=7
cycles
AdcRegs.ADCINTSOCSEL1.bit.SOC0 = 0; // No ADCINT triggers SOC0. TRIGSEL
field determines trigger.
AdcRegs.SOCPRICTL.bit.SOCPRIORITY = 0; // All SOCs handled in round-
```

robin mode

//— ADCINT1 configuration
AdcRegs.INTSEL1N2.bit.INT1CONT = 1; // ADCINT1 pulses regardless of ADCINT1 flag state
AdcRegs.INTSEL1N2.bit.INT1E = 1; // Enable ADCINT1
AdcRegs.INTSEL1N2.bit.INT1SEL = 0; // EOC0 triggers ADCINT1
PieCtrlRegs.PIEIER1.bit.INTx1 = 1; // Enable ADCINT1 in PIE group 1
IER |= 0x0001; // Enable INT1 in IER to enable PIE group
AdcRegs.ADCCTL1.bit.ADCENABLE = 1; // Enable the ADC
asm(" EDIS"); // Disable EALLOW protected register access
} // end InitAdc()

GUIDE : Configuration of EPWM2 to trigger the ADC at a 50 kHz sampling rate in function initEPWM() of Epwm.c file. Configure ePWM1 for 15 kHz symmetric PWM on EPWM1A pin.

3.7 Event capture(ECAP) Module

ECAP module of F28x series is used to get PWM duty cycle with time stamps. Using time stamps to get PWM ON period & PWM Off period from that duty cycle can be calculated. This calculation require times in micro second to get PWM duty value.

Example 3.7 :
void InitECapture(void);
void main(void)
{
InitECap1Gpio();

```
InitECap();
EALLOW; // This is needed to write to EALLOW protected registers
PieVectTable.ECAP1_INT = &ecap1_isr;
EDIS; // This is needed to disable write to EALLOW protected register
for(;;);
}
```

```
__interrupt void ecap1_isr(void)
{
ECap1PassCount++;
ECap1Regs.ECCLR.bit.CEVT4 = 1;
ECap1Regs.ECCLR.bit.INT = 1;
ECap1Regs.ECCTL2.bit.REARM = 1;
TSt1 = ECap1Regs.CAP1; // Fetch Time-Stamp captured at t1
TSt2 = ECap1Regs.CAP2; // Fetch Time-Stamp captured at t2
TSt3 = ECap1Regs.CAP3; // Fetch Time-Stamp captured at t3
TSt4 = ECap1Regs.CAP4; // Fetch Time-Stamp captured at t4
Period1 = TSt3-TSt1; // Calculate 1st period
DutyOnTime1 = TSt2-TSt1; // Calculate On time
DutyOffTime1 = TSt3-TSt2; // Calculate Off time
pcycle = (float)(DutyOnTime1/Period1); // Calculate Duty Cycle
// Acknowledge this interrupt to receive more interrupts from group 4
PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;
}
```

3.8 External Interrupt(XINT) Module

External interrupt in XINT module is used for capturing real time trip as input. XINT can be configured for both kind of trip detection either high to low & low to high. Use XINT to set ISR in real time at trip detection. XINT take times in nano second to capture PWM & regenerate it.

Example 3.8 :

```
__interrupt void xint1_isr(void);
__interrupt void xint2_isr(void);
Uint32 Xint1Count;
Uint32 Xint2Count;
void main(void)
{
InitGpio();
InitPieVectTable();
EnableInterrupts();
EALLOW; // This is needed to write to EALLOW protected registers
PieVectTable.XINT1 = &xint1_isr;
PieVectTable.XINT2 = &xint2_isr;
EDIS; // This is needed to disable write to EALLOW protected registers
PieCtrlRegs.PIEIER1.bit.INTx4 = 1; // Enable PIE Gropu 1 INT4
PieCtrlRegs.PIEIER1.bit.INTx5 = 1; // Enable PIE Gropu 1 INT5
EINT; // Enable Global Interrupts
ERTM; // Enable Global realtime interrupt DBGM
EALLOW;
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0; // GPIO
GpioCtrlRegs.GPADIR.bit.GPIO0 = 0; // input
GpioCtrlRegs.GPAQSEL1.bit.GPIO0 = 0; // Xint1
```

```
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 0; // GPIO
GpioCtrlRegs.GPADIR.bit.GPIO1 = 0; // input
GpioCtrlRegs.GPAQSEL1.bit.GPIO1 = 0; // Xint2

// GPIO0 is XINT1, GPIO1 is XINT2
GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 0; // Xint1 is GPIO0
GpioIntRegs.GPIOXINT2SEL.bit.GPIOSEL = 1; // XINT2 is GPIO1
EDIS;

// Configure XINT1
XIntruptRegs.XINT1CR.bit.POLARITY = 1; // Rising edge interrupt
XIntruptRegs.XINT2CR.bit.POLARITY = 0; // Falling edge interrupt

// Enable XINT1 and XINT2
XIntruptRegs.XINT1CR.bit.ENABLE = 1; // Enable Xint1
XIntruptRegs.XINT2CR.bit.ENABLE = 1; // Enable XINT2

for(;;);
}

__interrupt void xint1_isr(void) // Rising edge interrupt
{
Xint1Count++;
// ISR routine section
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
__interrupt void xint2_isr(void) // falling edge interrupt
{
```

```
Xint2Count++;
// ISR routine section
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

3.9 Summary

This chapter deals with different modules of Piccolo launchpad that we have used in our software development. In this chapter guide is given how we can configure module & use it into our Application.

Chapter 4

Splitting C Program

4.1 Header file

When writing large programs, dividing it into modules is reliable to understand. The program can be separated source files. main() can be in one file, main.c say, the others will contain functions. Our own library of functions by writing a suite of subroutines in one (or more) modules can be created. In fact modules can be shared among many programs by simply including the modules at compilation. There are many advantages to this approach, the modules will divide into common groups of functions. Compilation by compiler will be for each module is separately then it will link compiled modules by linker.

Header files

A modular approach adoption will used naturally, while keeping variable definitions, function prototypes etc. with each module. However what if several modules need to share such definitions?

It is best to centralize the definitions in one file and share this file among the modules. Such a file is usually called a header file. Convention states that these files have a .h suffix.

E.g.: Task_1.h

$\#ifndef PROGRAM\_Task\_1\_H\_$

$\#define PROGRAM\_Task\_1\_H\_$

extern int Task1(int a);

#endif /* PROGRAM_Task_1_H_ */

In example header files are included into Program.c main file by using ifndef definition. Then extern is useful to get external code into our program.c main source file. The function Task1() written in header file will be called from other source file.

4.2 Source file

Now as described in above section needs to call function from other file. To define source file & call it's function from header file discussed below.

E.g. : Task.c

#ifndef Task_1_

#define Task_1_

#ifdef __cplusplus

extern "C"

#endif

#include "DSP28x_Project.h"

extern unsigned int val,Task;

#ifndef Task1_INCLUDED

#define Task1_INCLUDED

int Task1(int a)

{

// Set your condition for jump out to mainloop4out

```
GpioDataRegs.GPASET.bit.GPIO0 =1;
for(;;)
{
GpioDataRegs.GPATOGGLE.bit.GPIO0 =1;
DELAY_US(200000);
if(Task !=1){
goto mainloop1out;
}
}
mainloop1out :
val = Task;
return val;
}
#endif
#ifdef __cplusplus
}
#endif /* extern "C" */
#endif /* NEW_C_ */
```

The source file which includes called function from Task1.h header learned in above section. After including function to header to define this function as global using extern.
Then other header file to it should be included. Don't forget to put one blank line at the end creating source or header file that may generate error.

4.3 Passing argument and Returning value

Now as given in the example section have definition of function named Task1 as shown below.

E.g.:

```
int Task1(int a)
{
.
.
return xyz;
}
```

The argument is into bracket as integer a which will give us value of written variable into bracket in main.c file. At the end of code definition returns xyz which return value to the function & the return type is defined at starting as integer so, it will return integer value.

4.4 Advantages of Using Several Files

Advantages of Using Several Files The main advantages of spreading a program across several files are:

Teams of programmers can work on programs. Each programmer works on a different file.

An object oriented style can be used. Each file defines a particular type of object as a data type and operations on that object as functions. The implementation of the object can be kept private from the rest of the program. This makes for well structured programs which are easy to maintain.

Files can contain all functions from a related group. For Example all matrix operations. These can then be accessed like a function library.

Well implemented objects or function definitions can be reused in other programs,

reducing development time.

In very large programs each major function can occupy a file to itself. Any lower level functions used to implement them can be kept in the same file. Then programmers who call the major function need not be distracted by all the lower level work.

4.5 Summary

This Chapter deals with development of large C Program. When we are developing large C Program we should divide our code functionality wise, because it makes easier to edit code & detect the part of code which is not working.

Chapter 5

Issues Faced During Development

5.1 Insufficient Memory

POST : In program almost PRAMH0 section of .text section is used now it needs

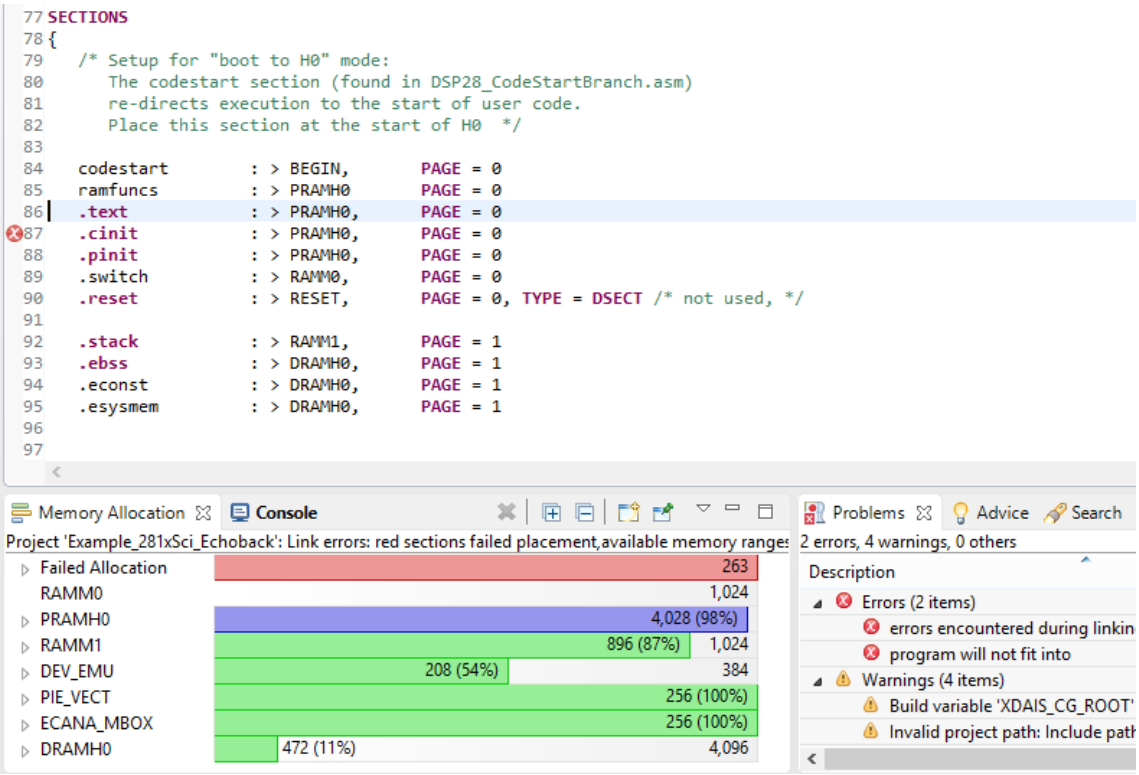


Figure 5.1: Insufficient memory error [6]

to extend memory.Is it possible to extend memory?

ANS : For getting more .text section we need FLASH memory.

5.2 Floating Number Error

POST : As shown in figure we can not get floating number, How can we expand it upto 4 floating digits? & Error:identifier not found status

(x)= Variables | Expressions | Registers

Expression	Type	Value	Address
(x)= status	unknown	Error: identifier not found: status	
(x)= MSB	unknown	Error: identifier not found: MSB	
(x)= LSB	unknown	Error: identifier not found: LSB	
(x)= Range	float	0.0 (Decimal)	0x00000008@Data
(x)= Voltage1	unknown	Error: identifier not found: Voltage1	
(x)= Voltage2	unknown	Error: identifier not found: Voltage2	
(x)= ConversionCount	unknown	Error: identifier not found: Conversi...	
AdcBuf	unsigned int[50]	0x0000000A@Data	0x0000000A@Data
(x)= adcsamp	unsigned int	0	0x00000002@Data
(x)= value	unsigned int	301	0x00000003@Data
(x)= Range	float	0.0 (Decimal)	0x00000008@Data
(x)= temp	float	0.0	0x00000006@Data
Add new expression			

Figure 5.2: Floating point Error and identifier not found[6]

ANS : Problem we faced is even though adcsamp & ans is in float, we are getting the division of adcsamp to Range_b as integer.

The other Error identifier not found status is due to variable is not defined as global, so value will not appear during run time.

```
while(1)                          // endless loop - wait for an interrupt
{
    adcsamp = AdcResult.ADCRESULT0;
    ans = (float)(adcsamp/Range_b);
    temp = ans*Range_a;
    value = temp;
}
//end of main()
```

Figure 5.3: Solved Floating error[6]

5.3 CMD File Error

POST : while changing the built in value we getting error that "Program will not fit into available memory".

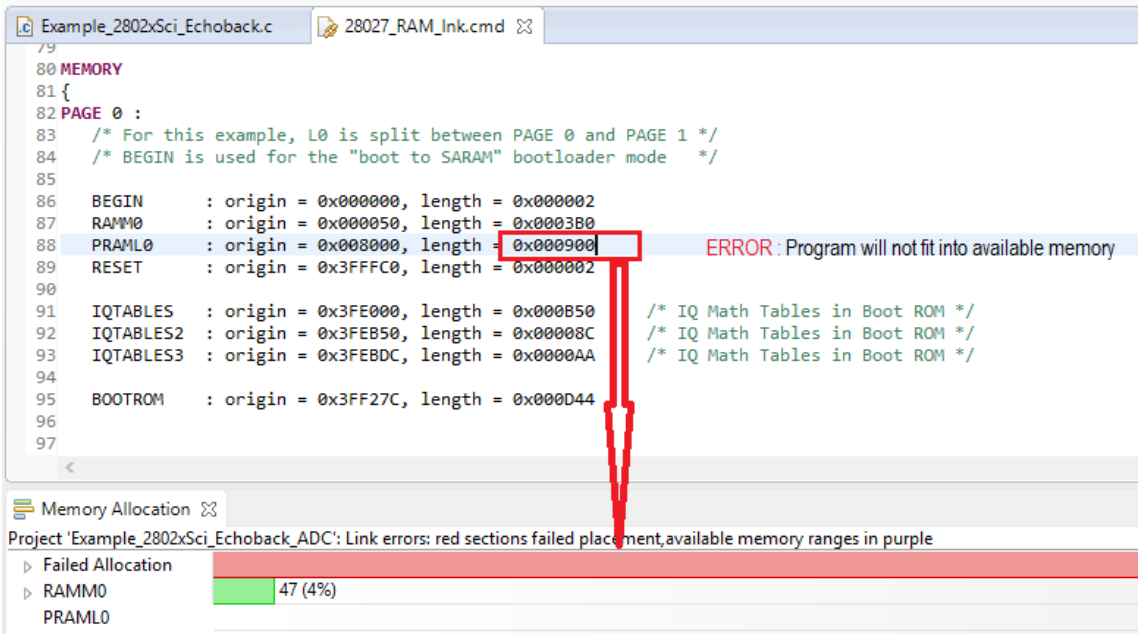


Figure 5.4: Program will not fit into memory [6]

ANS : In this figure 5.2 .text section require more memory, as we have 4K memory for L0 available we can extend it.

But after extending it from 900 to B00 value we get other error in reading variable

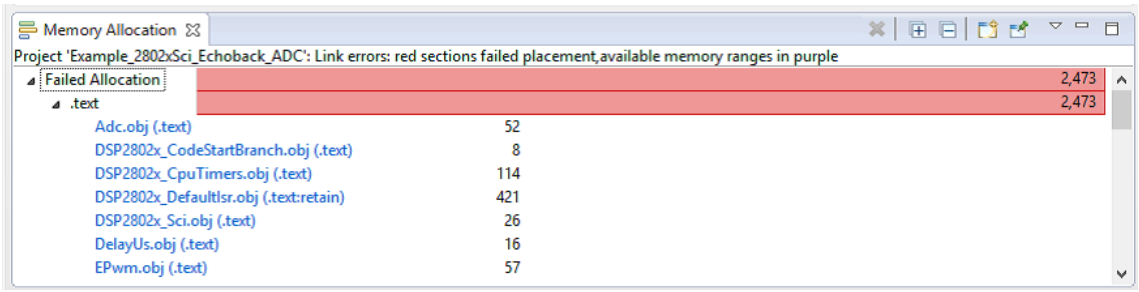


Figure 5.5: text section require more memory [6]

"Memory map is not readable"

& we had solved that error by changing the starting location of Next RAM ,because

that was Overlapping & error was occurring.

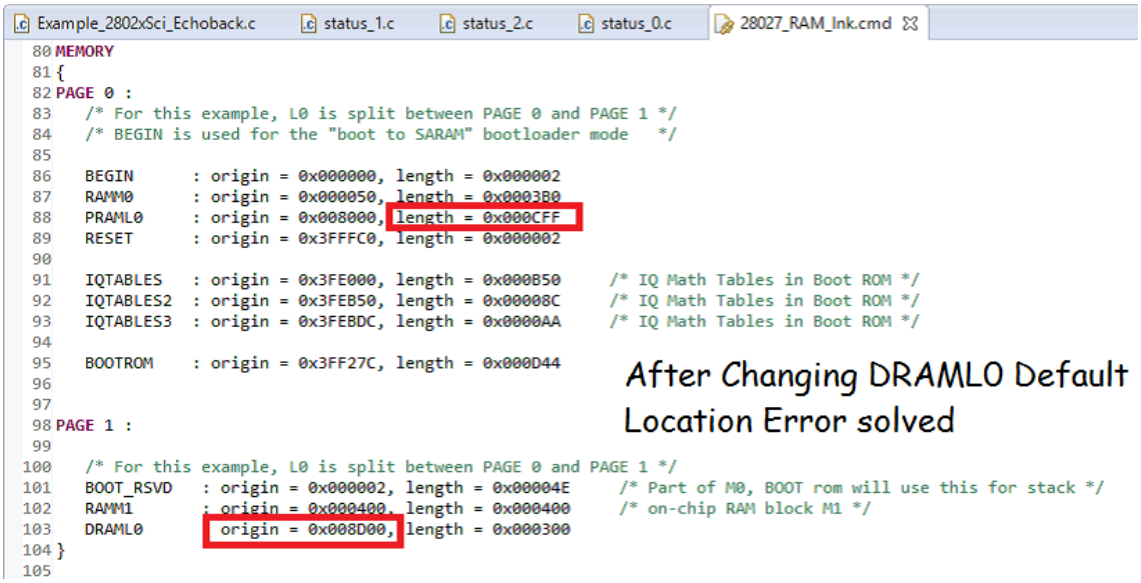


Figure 5.6: Solved Cmd Error[6]

5.4 ADC Stopped working

POST : Program structure is like in figure 5.6. We are initializing Adc as global to code at main function than in first task for loop we are using ADC result from adcresult0 register. But after complete task 1 "for loop" when we come return to MAIN function "adcresult0" register stop to showing sampling value in task 2 "for loop" why?

ANS : In task 1 we have not initialized any peripheral, but at start of task 2 at code start we have OFF all LED, we were using same GPIO port for ADC Triggering by EPWM1A & LED control handling. So, as we execute the following code at start of task 2 the GPIO0 was changed from EPWM to LED GPIO out as in code below and that is why adcresult register was stop to showing result & ADC was Stopped.

GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0; // 0=GPIO, 1=EPWM1A

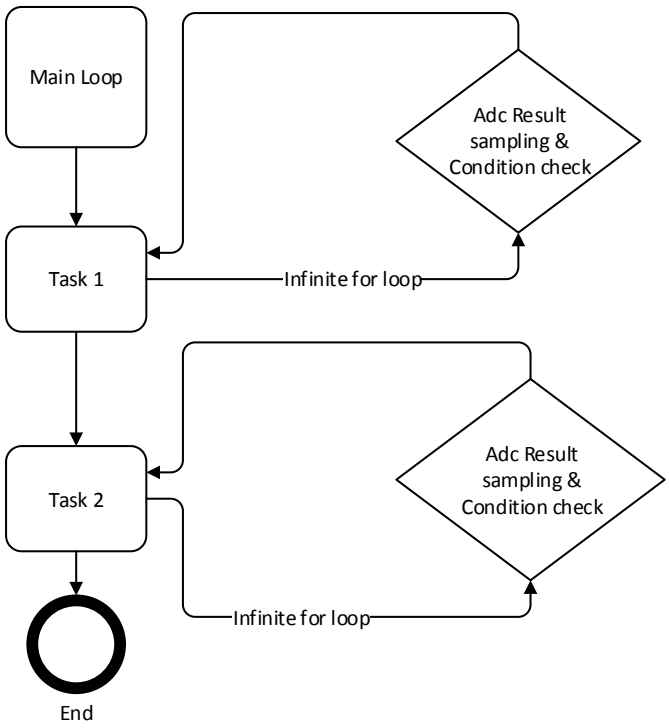


Figure 5.7: Program Structure for ADC [6]

GpioCtrlRegs.GPADIR.bit.GPIO0 = 1; // 1=OUTput, 0=INput
So, Using the EPWM4A which is not going to used for other purpose GPIO.

5.5 Infinite While Loop

POST : During SCI transmission using F28027 we have written following line which actually WAIT for XRDY=1 & program got stuck into infinite while loop
while(SciaRegs.SCIFFRX.bit.RXFFST != 1)
But During run it stop execution with " SciaRegs.SCIFFRX.bit.RXFFST =4 " & Loop was stuck due to FIFO is not clear.

ANS : Their was problem in controlSUITE examples. so we have to change it

to "while(SciaRegs.SCIFFRX.bit.RXFFST == 0) " to clear FIFO.
It might possible that there were 2 or more characters in the FIFO and the example was halting in the while loop and unable to read from the FIFO.

5.6 Summary

In This Chapter we have included error as the post that we had posted on TI Community & answered by us. We have different module available with different pin Configuration. so during Programming large software with many modules their are possibility of error generation.

Chapter 6

Developed GUI

6.1 GUI developed for Test Jig

Development of GUI as shown in figure 6.1 for F28335 Delfino Controller as Test Jig controller & in figure 6.2 F28027 Piccolo Controller as Chopper control card is done with GUI Composer.

Delfino Controller as Test Jig will control SPS control card with F28x series MCU. So, GUI of Test Jig will show value transmitted by delfino controller and set it. We are monitoring the values of Frequency, IGBT, Temperature, Under Voltage, Over voltage & Over current value. When their will be any data change, that will be transmitted by SPS control card as Error Packet.

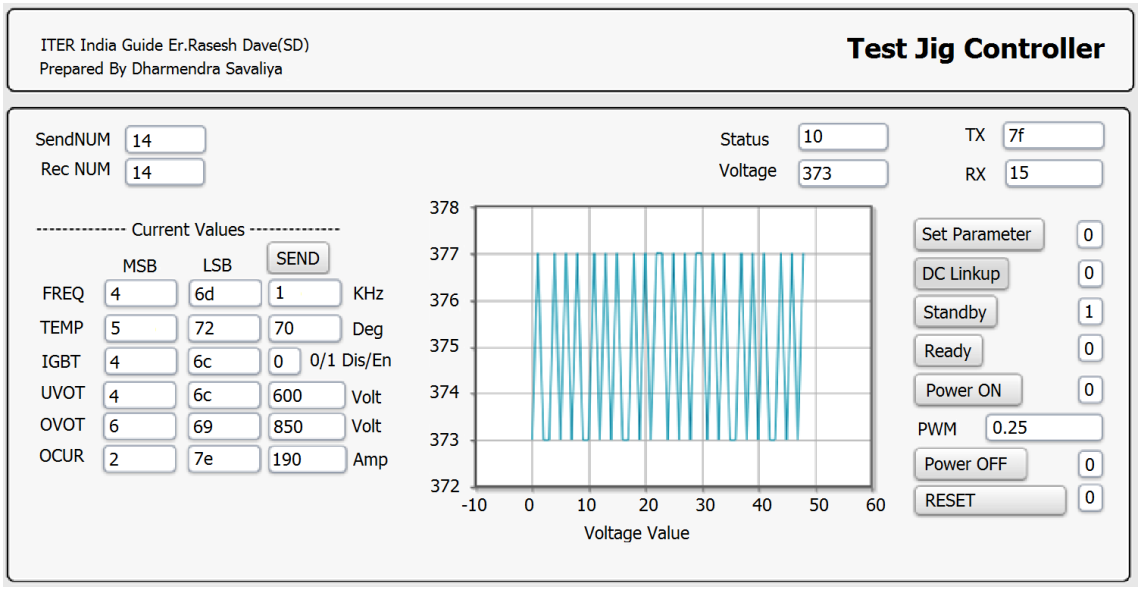


Figure 6.1: F28335 Test Jig Controller GUI

6.2 GUI developed for SPS Control Card

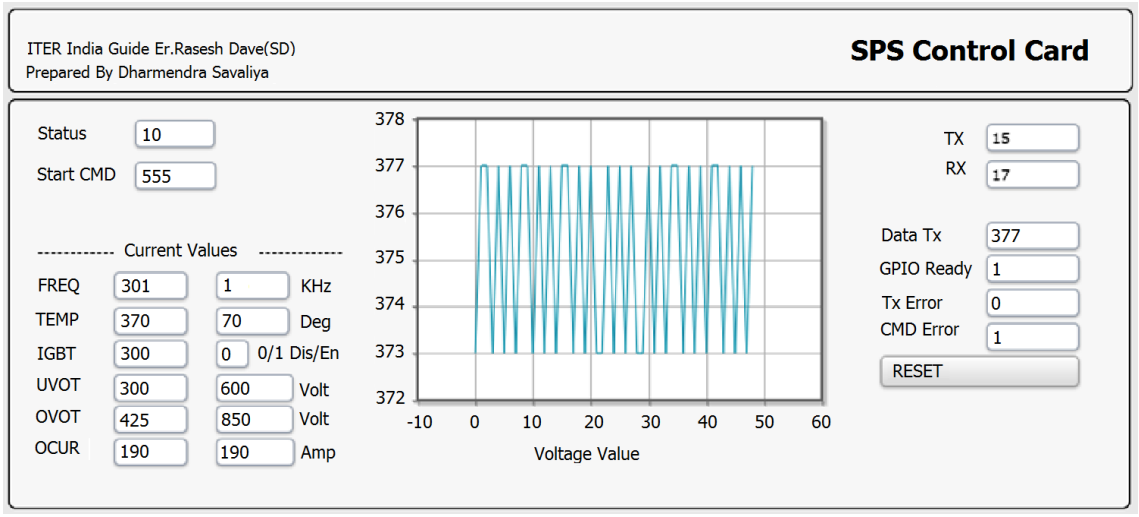


Figure 6.2: F28027 SPS Control Card GUI

6.3 Summary

This Chapter deals with GUI development on GUI Composer provided by CCS. We can learn how to develop GUI for our application using JAVA Script. How can we use Pre-processing & Post-processing function to see our desire value from range.

Chapter 7

Setup and Result

7.1 Experimental Outcomes

Development purpose SPS control module is tested with different available resistors instead of Thermistor. Digital count in Table 7.1 shows value of voltage received at ADC. As temperature increases value of voltage decreases.

Table 7.1: Temperature measurements

Sr no.	Resistor	Digital Count	Voltage	Temperature
1	3.3K	1774	1.2996	71
2	5K	2152	1.5766	60
3	10K	2765	2.0256	44
4	20K	3230	2.3663	24

During development voltage value is sampled with ADC available in DSP. ADC module support range of 0 to 3.3v, so voltage divider is used to give input voltage to ADC. Using the voltage divider voltage range from 0 to 1000v converted to range 0 to 3.3v as shown in Table 7.2.

Table 7.2: Voltage measurements

Sr no.	Input Voltage	Digital Count	Voltage
1	2.96	4048	850
2	2.53	3448	724
3	2.09	2858	600

7.2 System setup for development

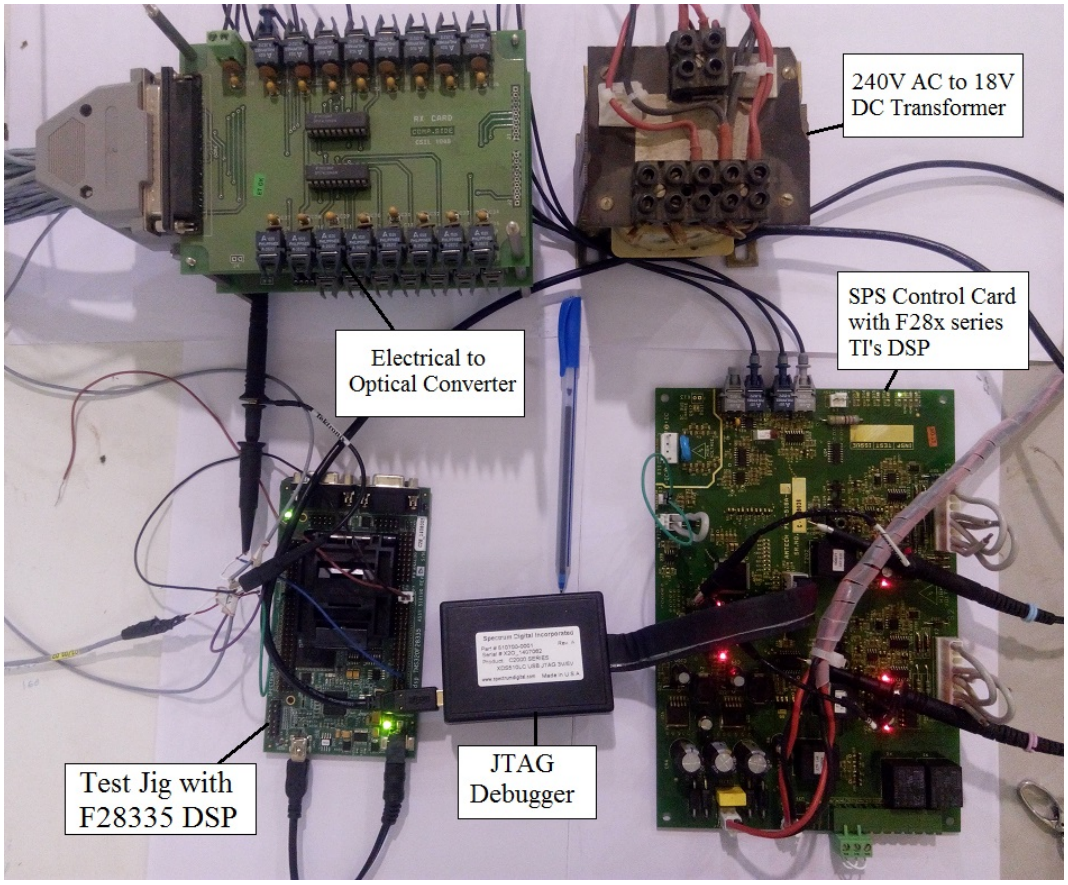


Figure 7.1: Dummy test setup with Test Jig & SPS control card

Development of Test Jig & SPS control card is simultaneously. Setup photo includes controller communication of F28335(Delfino) & F28027(Piccolo) Controller. The blue color output is data transmitted by Test Jig controller & Response by Piccolo Controller is in Yellow Color as shown in Figure 7.2.

7.3 Results

Generated PWM from Test Jig will be captured using ECAP module from F28x series controller. ECAP takes time stamps during capture after that for duty cycle calculation we have to do some operation on data which takes times. In fig 7.2 we can see generated PWM of Test jig & SPS module, there is 180us delay that is not tolerable.

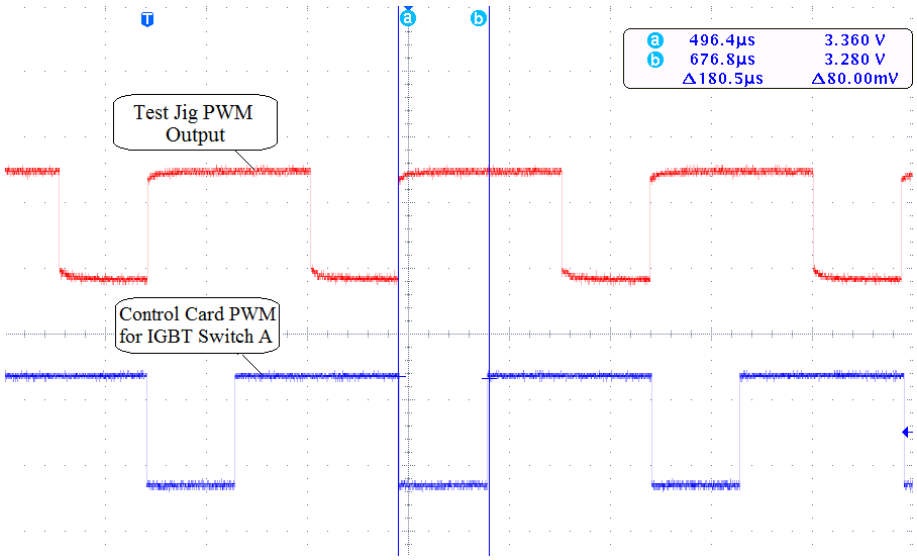


Figure 7.2: EPWM Regeneration from F28x series DSP with ECAP

To reduce delay generated PWM from test jig will be captured using external interrupt module from F28x series controller. External interrupt generate PWM by setting and erasing GPIO as soon as interrupt received that gives real time response. For conditional checking of PWM duty cycle we start CPU timer as interrupt received. In fig 7.3 we can see generated PWM of Test jig & SPS module, there is 222ns delay that is sufficient & real time.

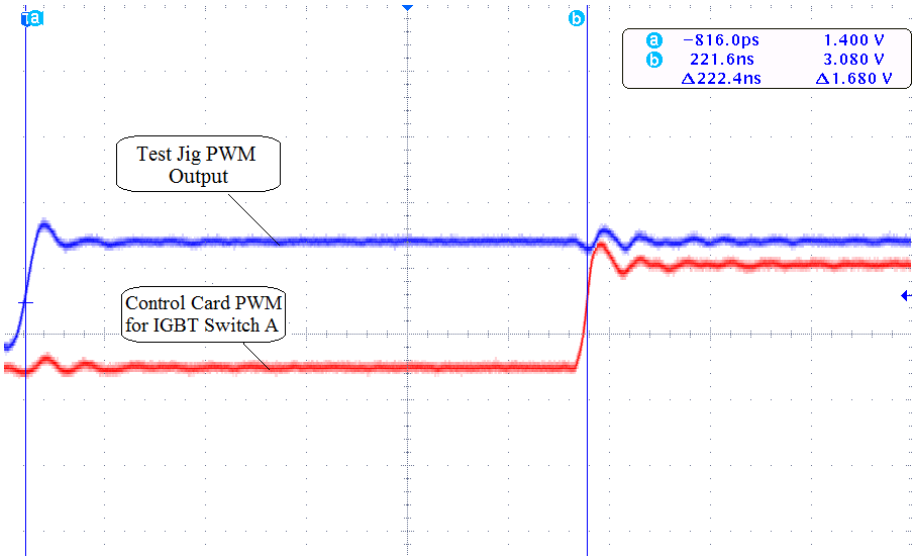


Figure 7.3: EPWM Regeneration from F28x series DSP with XINT

During real time PWM capturing, there is possibility of generation of high frequency or undesirable signal that can damage circuit. high frequency between 1ms as shown in fig 7.4 can be detected by timer interrupt and terminate our program.

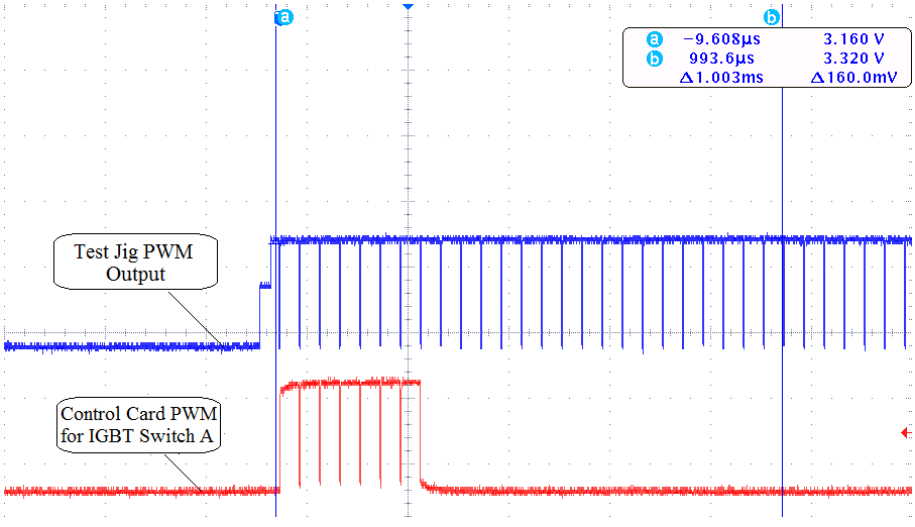


Figure 7.4: high frequency detection by F28x series DSP

During the real time operation when IGBT switch A is off as shown in figure 7.5,

capacitor bank should be discharged for circuit protection. Discharge of capacitor bank will be done by IGBT switch B. When IGBT switch is off we will switch on IGBT switch B for small 3us for discharging capacitor bank.

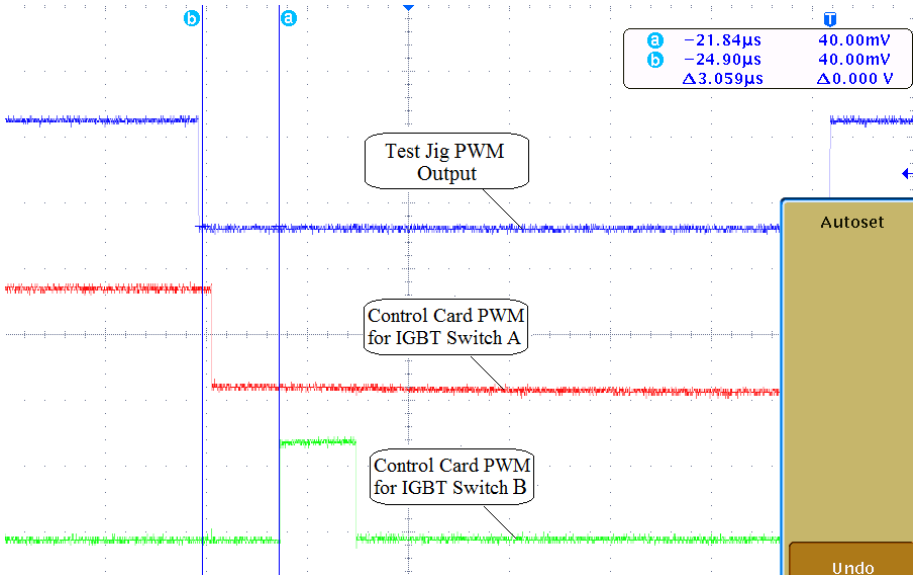
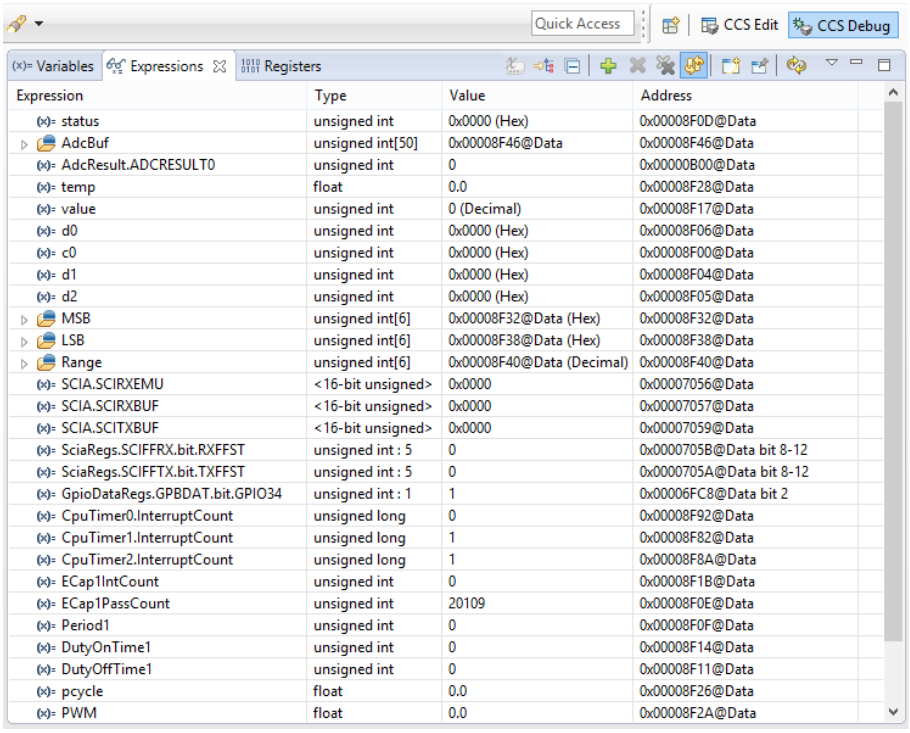


Figure 7.5: EPWM generation for both IGBTs

7.4 CCS Debug Window

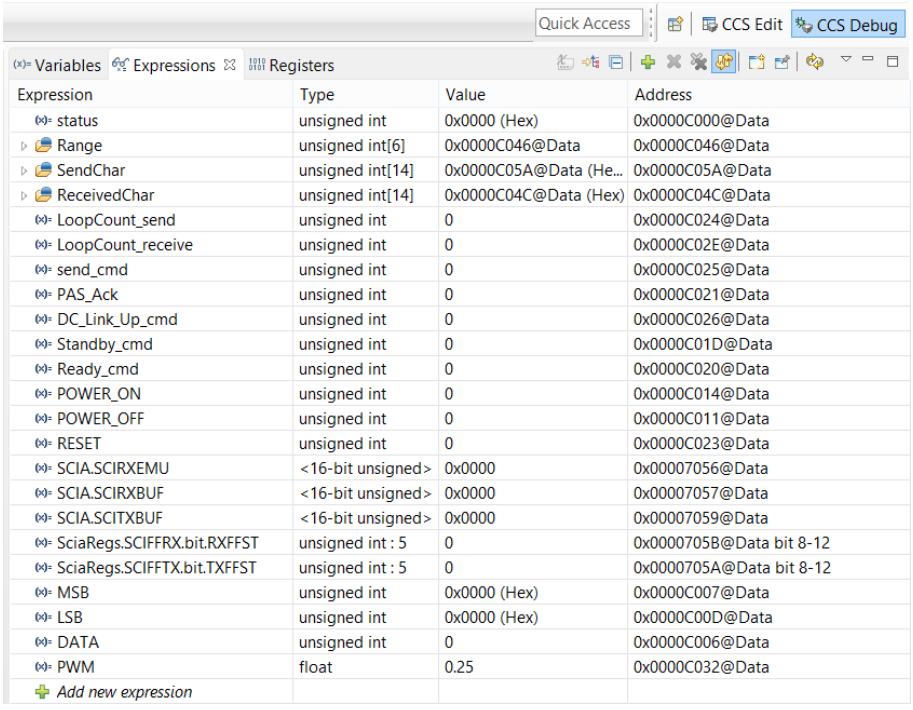
1 Piccolo - SPS Control Card Figure 7.6 is CCS Debug window of F28027 piccolo controller. Expression tab is useful to shows continuous value of global variable. Status, AdcBuf, Adc value, d1,d2 & range are included in Expression for continuous monitoring.

2 Delfino - Test Jig Controller CCS Debug Window for Delfino Test Jig controller as in figure 7.7 includes different value received from piccolo controller. Different task can be operated by this Debug window.



Expression	Type	Value	Address
status	unsigned int	0x0000 (Hex)	0x00008F0D@Data
AdcBuf	unsigned int[50]	0x00008F46@Data	0x00008F46@Data
AdcResult.ADCRESULT0	unsigned int	0	0x00000B00@Data
temp	float	0.0	0x00008F28@Data
value	unsigned int	0 (Decimal)	0x00008F17@Data
d0	unsigned int	0x0000 (Hex)	0x00008F06@Data
c0	unsigned int	0x0000 (Hex)	0x00008F00@Data
d1	unsigned int	0x0000 (Hex)	0x00008F04@Data
d2	unsigned int	0x0000 (Hex)	0x00008F05@Data
MSB	unsigned int[6]	0x00008F32@Data (Hex)	0x00008F32@Data
LSB	unsigned int[6]	0x00008F38@Data (Hex)	0x00008F38@Data
Range	unsigned int[6]	0x00008F40@Data (Decimal)	0x00008F40@Data
SCIA.SCIRXEMU	<16-bit unsigned>	0x0000	0x00007056@Data
SCIA.SCIRXBUF	<16-bit unsigned>	0x0000	0x00007057@Data
SCIA.SCITXBUF	<16-bit unsigned>	0x0000	0x00007059@Data
SciaRegs.SCIFFRX.bit.RXFFST	unsigned int : 5	0	0x0000705B@Data bit 8-12
SciaRegs.SCIFFTX.bit.TXFFST	unsigned int : 5	0	0x0000705A@Data bit 8-12
GpioDataRegs.GPBDAT.bit.GPIO34	unsigned int : 1	1	0x00006FC8@Data bit 2
CpuTimer0.InterruptCount	unsigned long	0	0x00008F92@Data
CpuTimer1.InterruptCount	unsigned long	1	0x00008F82@Data
CpuTimer2.InterruptCount	unsigned long	1	0x00008F8A@Data
ECap1IntCount	unsigned int	0	0x00008F1B@Data
ECap1PassCount	unsigned int	20109	0x00008F0E@Data
Period1	unsigned int	0	0x00008F0F@Data
DutyOnTime1	unsigned int	0	0x00008F14@Data
DutyOffTime1	unsigned int	0	0x00008F11@Data
pcycle	float	0.0	0x00008F26@Data
PWM	float	0.0	0x00008F2A@Data

Figure 7.6: CCS debug window for SPS Control card



Expression	Type	Value	Address
status	unsigned int	0x0000 (Hex)	0x0000C000@Data
Range	unsigned int[6]	0x0000C046@Data	0x0000C046@Data
SendChar	unsigned int[14]	0x0000C05A@Data (He...	0x0000C05A@Data
ReceivedChar	unsigned int[14]	0x0000C04C@Data (Hex)	0x0000C04C@Data
LoopCount_send	unsigned int	0	0x0000C024@Data
LoopCount_receive	unsigned int	0	0x0000C02E@Data
send_cmd	unsigned int	0	0x0000C025@Data
PAS_Ack	unsigned int	0	0x0000C021@Data
DC_Link_Up_cmd	unsigned int	0	0x0000C026@Data
Standby_cmd	unsigned int	0	0x0000C01D@Data
Ready_cmd	unsigned int	0	0x0000C020@Data
POWER_ON	unsigned int	0	0x0000C014@Data
POWER_OFF	unsigned int	0	0x0000C011@Data
RESET	unsigned int	0	0x0000C023@Data
SCIA.SCIRXEMU	<16-bit unsigned>	0x0000	0x00007056@Data
SCIA.SCIRXBUF	<16-bit unsigned>	0x0000	0x00007057@Data
SCIA.SCITXBUF	<16-bit unsigned>	0x0000	0x00007059@Data
SciaRegs.SCIFFRX.bit.RXFFST	unsigned int : 5	0	0x0000705B@Data bit 8-12
SciaRegs.SCIFFTX.bit.TXFFST	unsigned int : 5	0	0x0000705A@Data bit 8-12
MSB	unsigned int	0x0000 (Hex)	0x0000C007@Data
LSB	unsigned int	0x0000 (Hex)	0x0000C00D@Data
DATA	unsigned int	0	0x0000C006@Data
PWM	float	0.25	0x0000C032@Data
Add new expression			

Figure 7.7: CCS debug window for Test Jig

7.5 Summary

this chapter presents the outcome/results of work. Photos of setup & photos of CCS debug window for our project work has been included.

Chapter 8

Conclusion and Future Scope

8.1 Conclusion

Development of Software for Chopper Control Card is presented in this report. Texas instruments provide C2000 Family provide low power & low cost controller. F28335 Delfino controller is used in chopper control card development. This controller provide temperature sensor, CPU timer, watchdog, EPWM, ADC, XINT & SCI features,which is most suitable for chopper control card development. Developed Software allows state transition using UART Communication. Code composer studio provide ControlSUITE which includes header files, Source files & libraries for all Texas instrument's Devices. JTAG Debugger Of Controller with XDS cable will allow to real time debug in CCS debugger.

SPS Module can be controlled by DSP with accurate precision. Fast switching in IGBT can be done by XINT. DC link voltage & IGBT current can be monitor by ADC module for circuit protection. Using On-board intelligence, state of control card can be monitored from Test Jig controller. Developed software for the chopper control is tested with Field application module and Test Jig.

8.2 Future Scope

Developed DSP software can be tested for multiple chopper modules in an integrated manner. Further rigorous testing of software in field is essential to demonstrate required performance.

Appendix A

Code Composer Studio guide

A.1 Structure of files

CCS Project Contains C Source, Assembly, Linker command file, DSP Configuration & Libraries. CCS Project also require to configure Build Option & Build configuration. Different CCS File structures & how to configure project is described in below.

CMD File: Command Linker file includes two main section as Program memory & Data memory. In CMD file Initialized Sections are declared as Flash or Program memory while Uninitialized Sections are declared as RAM or Data Memory.

Example.cmd

{

PAGE 0: /* Program Memory */

FLASH: origin = 0x3E8000, length = 0x10000

PAGE 1: /* Data Memory */

M0SARAM: origin = 0x000000, length = 0x400

M1SARAM: origin = 0x000400, length = 0x400

}

In the CMD file above we have given program memory as page number 0 & Data

memory as page number 1.

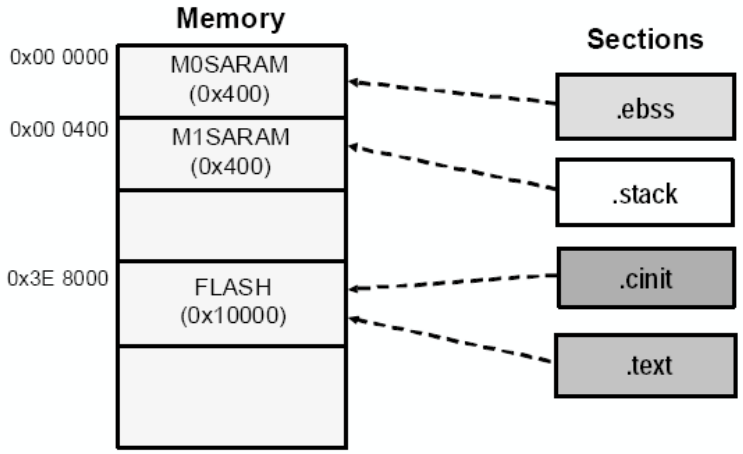


Figure A.1: Section in memory [6]

In Figure A.1 we can see .ebss & .stack is initialized with RAM while .cinit & .text Sections are initialized with FLASH in memory. In table A.1 initialized section should be written into FLASH Or Program memory & Uninitialized sections should be written into RAM or Data memory.

Table A.1: Compiler Section Names [7]

Sections	**Name**	**Description**	**Link Location**
Initialized Sections	.text	Code	FLASH
	.cinit	Initialize Value	FLASH
	.econst	Constant	FLASH
	.switch	Table for Switch statement	FLASH
	.pinit	Global Constructor	FLASH
Uninitialized Sections	.ebss	Global & Static Variable	RAM
	.stack	Stack Space	RAM
	.esysmem	Memory for malloc function	RAM

A.2 Creating New Project

CCS Provide Control Suite software which includes libraries, header file & peripheral source file.Control suite also include Texas instrumentation Compiler. To Create new CCS Project we have to follow steps given below :

Step 1 First from Project menu select create New CCS Project.After that select target as "2802x piccolo" for F28027 Launchpad & "TMS320F28027".For this development board Connection available as "Texas instrumentation XDS 100v1 USB Debug Probe".After that give project name & select compiler version. Then Click on finish.

Step 2 Now to add source file right Click on project and Click on "Add Files".Then Go to location " C:\ti\controlSUITE\device_support\f2802x\v127\DSP280-2x_common\source" and add CodeStartBranch.asm, DefaultISR.c, SysCtrl.c, PieCtrl.c, PieVect.c & necessary peripheral files.

Step 3 After that add file "C:\ti\controlSUITE\device_support\f2802x\v127\DSP-2802x_headers\source\DSP2802x_GlobalVariableDefs.c" & "C:\ti\controlSUITE\device_support\f2802x\v127\DSP2802x_headers\cmd\DSP2802x_Headers_nonBIOS.cmd".

Step 4 Then to add .CCXML file Right Click on Project "NEW" then "Target Configuration File" & add File by Choosing appropriate Target Board.

Step 5 Now to configure project properties Right Click on Project "Properties" then go to Build− >C2000 Compiler And add Source File locations.

Step 6 After that to add linker location Right Click on Project "Properties" then go to Build− >C2000 Linker And add Source File Search paths.

Step 7 That write basic program into main.c File then build, debug & Resume to Run Program on target board.

A.3 Build and Debug Project

When we build, debug & run the program on our target board the procedure take place is shown in figure A.3.Code is written into editor. Now .cmd file added as DSP/BIOS configuration. Compiler covert Source code into machine code or .asm. After that Assembler Translates assembly language statements into target machine code.Then Linker will link .asm file with DSP/BIOS Libraries,Run time Libraries & generate .out file. This generated .out file will be given to Debugger.

Debugger can debug this file Using simulator(SIM), Development Support Kit(DSK), Evaluation Module(EVM) Or Cross document Sharing(XDS) to DSP Board.

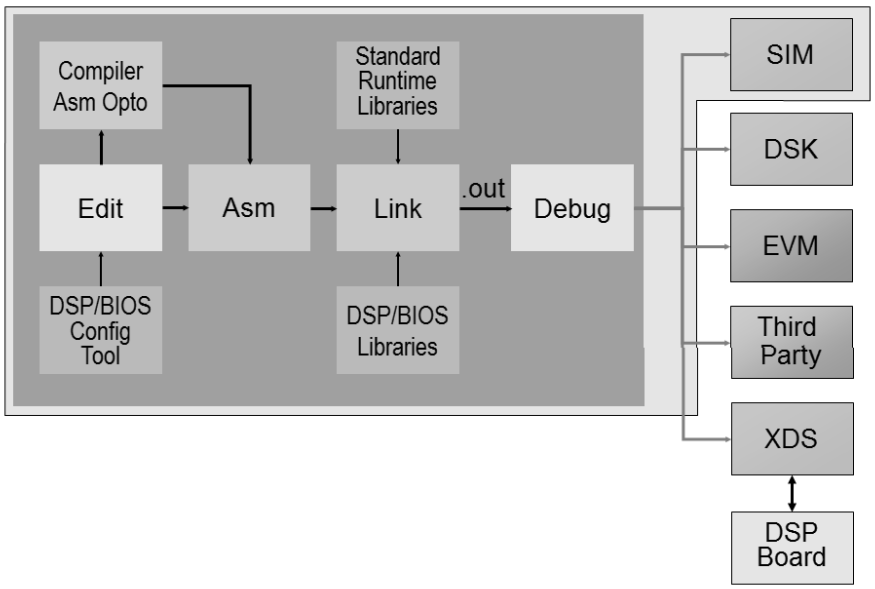


Figure A.2: CCS Work Flow

A.4 Flash Programming

Industrial product with DSP have in built flash program which runs on Power on. In code composer studio for flash programming, we need to change F28x_RAM.cmd

file to F28x.cmd file. we need to first configure our device from project property as shown in figure A.1.

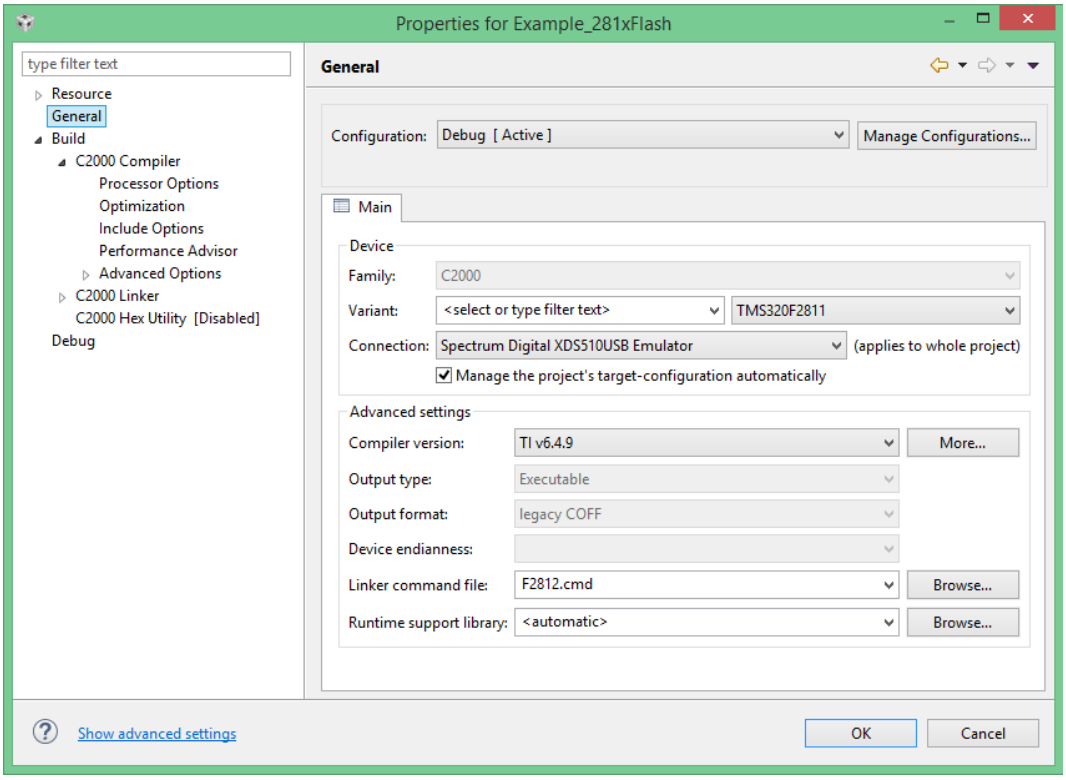


Figure A.3: Property setup for flash programming

Functions that will be run from RAM need to be assigned to a different section. This section will then be mapped using the linker cmd file.

#pragma CODE_SECTION(eva_timer1_isr, "ramfuncs");

#pragma CODE_SECTION(eva_timer2_isr, "ramfuncs");

#pragma CODE_SECTION(evb_timer3_isr, "ramfuncs");

// Copy time critical code and Flash setup code to RAM

// Memcopy should be added before initsysctrl function. MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);

// Call Flash Initialization to setup flash waitstates

// This function must reside in RAM

```
InitFlash();

MEMORY
{
PAGE 0: /* Program Memory */
/* Memory (RAM/FLASH/OTP) blocks can be moved to PAGE1 for data allocation */
FLASHD : origin = 0x3EC000, length = 0x004000 /* on-chip FLASH */
FLASHC : origin = 0x3F0000, length = 0x004000 /* on-chip FLASH */
FLASHA : origin = 0x3F6000, length = 0x001F80 /* on-chip FLASH */
CSM_RSVD : origin = 0x3F7F80, length = 0x000076 /* Part of FLASHA. Program with all 0x0000 when CSM is in use. */
BEGIN : origin = 0x3F7FF6, length = 0x000002 /* Part of FLASHA. Used for "boot to Flash" bootloader mode. */
CSM_PWL : origin = 0x3F7FF8, length = 0x000008 /* Part of FLASHA. CSM password locations in FLASHA */
ROM : origin = 0x3FF000, length = 0x000FC0 /* Boot ROM available if MP/MCn=0 */
RESET : origin = 0x3FFFC0, length = 0x000002 /* part of boot ROM (MP/MCn=0) or XINTF zone 7 (MP/MCn=1) */
VECTORS : origin = 0x3FFFC2, length = 0x00003E /* part of boot ROM (MP/MCn=0) or XINTF zone 7 (MP/MCn=1) */

PAGE 1 : /* Data Memory */
/* Memory (RAM/FLASH/OTP) blocks can be moved to PAGE0 for program allocation */
/* Registers remain on PAGE1 */
RAMM0 : origin = 0x000000, length = 0x000400 /* on-chip RAM block M0 */
RAMM1 : origin = 0x000400, length = 0x000400 /* on-chip RAM block M1 */
```

```
RAML1 : origin = 0x009000, length = 0x001000 /* on-chip RAM block L1 */
FLASHB : origin = 0x3F4000, length = 0x002000 /* on-chip FLASH */
RAMH0 : origin = 0x3F8000, length = 0x002000 /* on-chip RAM block H0 */
}

SECTIONS
{
/* Allocate program areas: */
.cinit : > FLASHA PAGE = 0 // From FLASHA
.pinit : > FLASHA, PAGE = 0 // From FLASHA
.text : > FLASHA PAGE = 0 // From FLASHA
codestart : > BEGIN PAGE = 0
ramfuncs : LOAD = FLASHD, // From FLASHD
        RUN = RAML0,
        LOAD_START(_RamfuncsLoadStart),
        LOAD_END(_RamfuncsLoadEnd),
        RUN_START(_RamfuncsRunStart),
    LOAD_SIZE(_RamfuncsLoadSize),
    PAGE = 0
/* Allocate uninitalized data sections: */
.stack : > RAMM0 PAGE = 1
.ebss : > RAML1 PAGE = 1
.esysmem : > RAMH0 PAGE = 1 // From RAMH0
/* Initalized sections go in Flash */
/* For SDFlash to program these, they must be allocated to page 0 */
.econst : > FLASHA PAGE = 0
.switch : > FLASHA PAGE = 0
.reset : > RESET, PAGE = 0, TYPE = DSECT
vectors : > VECTORS PAGE = 0, TYPE = DSECT
```

}

F2812.cmd file define the memory block start/length for the F2812 PAGE 0 will be used to organize program sections & PAGE 1 will be used to organize data sections. Allocate sections to memory blocks. Codestart user defined section in DSP28_CodeStartBranch.asm used to redirect code execution when booting to flash ramfuncs user defined section to store functions that will be copied from Flash into RAM.

.reset is a standard section used by the compiler. It contains the the address of the start of _c_int00 for C Code. When using the boot ROM this section and the CPU vector table is not needed.Thus the default type is set to DSECT.

A.5 Summary

During the beginning of development how to use CCS with launchpad is described in this chapter. How to configure CCS for launchpad, how to run code & how to debug program and view variable, graph & some features of CCS is shown in chapter.

Appendix B

GUI Development

B.1 GUI Composer

Code Composer Studio Provide GUI composer to develop graphical user interface. To develop GUI is simple using drag & drop button, text box & all other stuff. Back-end programming for pre & post processing function as shown in figure B.1 is done in JAVA Script. To run GUI in real time script file needs to be added in project.

In this project values coming from Host is in decimal so we have defined Pre-processing function into script file to convert it into Hex. After Showing value to GUI we have to do as that was so as Post processing function we have to add reverse function. We have developed GUI as shown in figure 6.1 for F28335 Delfino Controller as Test Jig controller & in figure 6.2 F28027 Piccolo Controller as Chopper control.

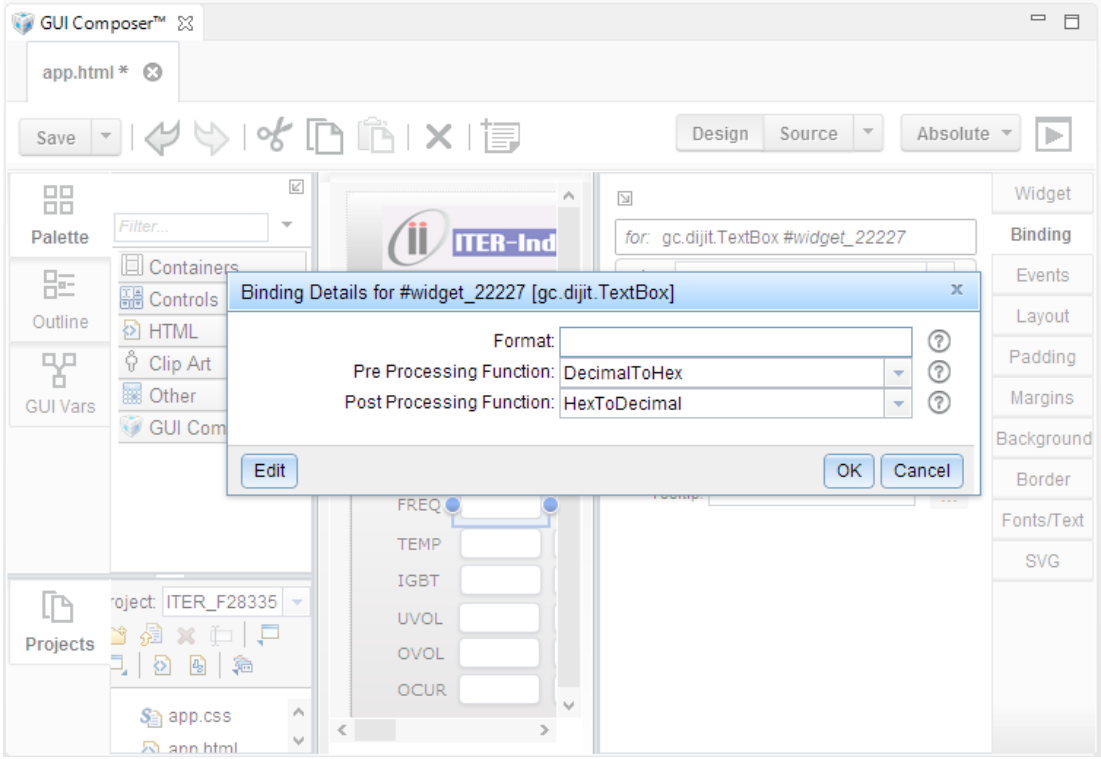


Figure B.1: GUI Composer

References

[1] Mandar Bhalekar & Umashankar, "Development Of a Research Platform for Power Electronic Converter Modeling in Real Time F28335 Digital Simulation Applications using eZDSP", IEEE Transaction paper on Power Electronics published in ICICES, Dec 2014

[2] Liran Katzir & Yakir Loewenstern "Implementation of a High Voltage Power Supply With The Matlab/Simulink Embedded Coder", IEEE Transaction paper on Power Electronics published in Convention of Electrical and Electronics Engineers in Israel, Jun 2014

[3] "Michael G. Giesselmann & William J. Carey "100-kV High Voltage Power Supply With Bipolar Voltage Output and Adaptive Digital Control", IEEE Transaction paper on Plasma Science published in OCT 2014

[4] "TMS320F28027 PICCOLO Documents by TI", [Online]. Available: $http://www.ti.com/product/tms320f28027?qgpn = tms320f28027$ Last Updated: DEC 2013

[5] "C2000 Launchpad Tutor For Beginners ", [Online]. Available: $http://azimsgarage.blogspot.in/p/blog - page.html$, Last Updated: Nov 2014

[6] Texas Instruments"MCU Design Days 2014 for C2000 ", [Online]. Available: $https://www.ti.com$ Last Updated: JUN 2014

[7] Texas instrumentation , *sprt547 C2000 Piccolo Workshop.pdf*, $http://www.ti.com$, JAN 2015

[8] Texas instrumentation , *TMS320F28027 PICCOLO Documents by TI*, $http://www.ti.com/product/tms320f28027?qgpn = tms320f28027$, JUL 2013

[9] Mandar Bhalekar and Umashankar S *A Development of a Research Platform for Power Electronic Converter Modeling in Real Time F28335 Digital Simulation Applications using eZDSP*, ICCPCT-2013

[10] "sprufn3d TMS320F2802x Piccolo System Control and Interrupts.pdf ", [Online]. Available: $https://www.ti.com$ Last Updated: JAN 2015