# Modularized, Reconfigurable and Bidirectional Charging Infrastructure for Electric Vehicles with Silicon Carbide Power Electronics (MoReSiC)

## Deliverable D3.3 (Month 27)

**Title: "Laboratory prototype of the multiport DC/DC converter"**

**Authors:** Dimosthenis Peftitsis, Kaushik Naresh Kumar
Norwegian University of Science and Technology

**June 2024**

## Executive summary

This deliverable D3.3 presents the design and hardware implementation of the laboratory prototype of the selected multiport isolated DC/DC power electronic converter in the MoReSiC project. Based on the theoretical and simulation assessment of several multiport DC/DC power converter configurations employing, the best-performing system has been chosen for practical design and implementation in the Power Electronics Lab at NTNU. The selected topology consists of two-level full-bridge converters in the primary side and the two secondary sides of the converter, which was implemented using Silicon Carbide Metal Oxide Semiconductor Field-Effect Transistors. This deliverable presents the procedural steps and crucial system components towards the hardware design of the multiport and isolate DC/DC power converter for EV charging. Exemplary experimental measurements of the electrical performance, power losses and efficiency are also presented and analysed.

## Table of Contents

***1. Selected Multiport DC/DC converter configuration ..... 3***

***2. Prototype Design and System Components ..... 4***

***3. Exemplary Experimental Measurements ..... 15***

***4. Conclusions ..... 16***

***5. Appendix ..... 17***

## 1. Selected Multiport DC/DC converter configuration

Based on a thorough theoretical and simulation study, the best performing multiport isolated DC/DC converter has been identified. This is the three-port converter consisting of a full-bridge (FB) circuit on the primary side, which is supplied by 750 V from the local DC bus of the charging station; and two FB circuits connected on the secondary side that can supply an output voltage of 400 V each. The galvanic isolation of the three-port converter is achieved by means of a high-frequency multiport transformer (T/F). A block diagram of the three-port DC/DC converter is shown in Fig. 1 and the schematic diagram of the FB circuit is illustrated in Fig. 2. More specifically, the FBs on the secondary side are fed through the two secondary isolated transformer windings and their output stages can be connected either in series or parallel. The specific connection mode depends by the power and voltage supply requirements to the EV's battery. For example, if an EV with an 800 V battery is to be charged, the outputs of the two secondary FBs are connected in series, while higher power requirements enabling fast charging imposes the need for parallel connecting the output stages of the two secondary FB circuits. Of course, a third option also emerges: the charging of two EVs simultaneously with 400 V, where each one is connected across one of the two secondary FB circuits. The various configurations can be achieved by controlling the switches S1, S2 and S3 dynamically. The detailed evaluation is presented in Deliverable D3.2.

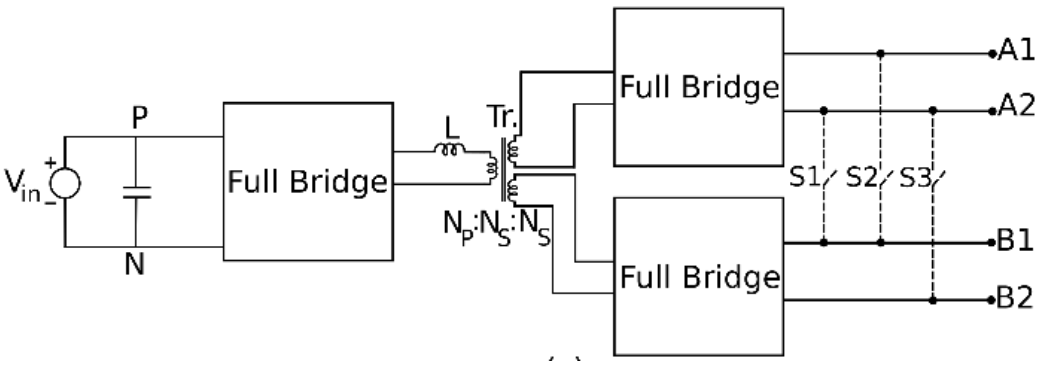



Fig. 1: Block diagram of the three-port FB-FB DAB.

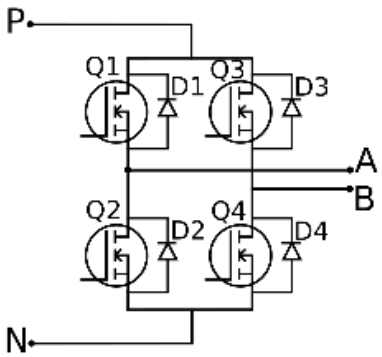



Fig. 2: Schematic diagram of the full bridge circuit used as the fundamental building block for the multiport converter

## 2. Prototype Design and System Components

The design parameters of the three-port isolated DC/DC converters are classified and summarized in the Tables I-VI below.

**Table I: Three-port converter design parameters**

| Parameter | Value | Unit |
|---|---|---|
| Input voltage, $V_{in}$ | 750 | V |
| Output voltage on each FB, $V_{out}$ | 400 | V |
| Rated power, $P$ | 10 | kW |
| Switching frequency, $f_{sw}$ | 100 | kHz |
| Inductor, $L$ | 67.89 | μH |
| Maximum junction temperature, $T_j$ | 100 | ºC |

**Table II: SiC Power MOSFETs Parameters – Primary full-bridge**

| Parameter | Value | Unit |
|---|---|---|
| Model number | NTH4L040N120SC1 | - |
| Breakdown voltage | 1200 | V |
| Rated current at 25ºC / 100ºC | 58 / 41 | A |
| Package | TO-247-4 | - |
| Typical/Maximum On-state resistance at $T_j$ = 25ºC | 40 / 56 | mΩ |
| Typical/Maximum On-state resistance at $T_j$ = 175ºC | 70 / 100 | mΩ |
| Internal gate resistor | 2.4 | Ω |

**Table III: SiC Power MOSFETs Parameters – Secondary full-bridge**

| **Parameter** | **Value** | **Unit** |
|---|---|---|
| Model number | C3M0015065K | - |
| Breakdown voltage | 650 | V |
| Rated current at 25ºC / 100ºC | 120 / 96 | A |
| Package | TO-247-4 | - |
| Typical On-state resistance at $T_j$ = 25ºC | 15 | mΩ |
| Typical On-state resistance at $T_j$ = 175ºC | 20 | mΩ |
| Internal gate resistor | 1.5 | Ω |

**Table IV: T/F design parameters**

| **Parameter** | **Value** | **Unit** |
|---|---|---|
| Primary to Secondary1 Turns ratio | 19:10 | - |
| Primary to Secondary2 Turns ratio | 19:10 | - |
| Input voltage/output voltages (both the secondaries) | 750V/400V | V |
| Magnetization inductance | 2.4 | mH |
| Primary / Sec.1 / Sec.2 Leakage inductance at 100kHz | 5.92 / 4.19 / 4 | μH |
| Rated power | 10 | kW |
| Operating frequency | 100 | kHz |

**Table V: Gate driver design parameters**

| Component | Value | Unit |
|---|---|---|
| Gate driver IC | NCV57000 | - |
| Positive gate driver voltage (Primary MOSFET/ Secondary MOSFET) | +20 / +15 | V |
| Negative gate driver voltage (Primary MOSFET/ Secondary MOSFET) | -5 / -5 | V |
| Peak driver current | 7.8 | A |
| External Turn-on / Turn-off gate resistor | 22 / 10 | Ω |

**Table VI: Other components**

| Components | Type/model | Characteristics |
|---|---|---|
| Heatsink | LA6/150/24V aluminium extruded heat sink from Fischer elektronik. | $R_{th}$ = 0.175 $^{o}$C /W |
| Microcontroller | F28379D LaunchPad™ development kit for C2000™ Delfino™ MCU | 4x 20-pin headers/connectors, 200 MHz dual C28xCPUs and dual CLAs, 1 MB Flash, 16-bit or 12-bit ADCs, comparators, 12-bit DACs, 24 Pulse Width Modulator (PWM) channels with enhanced features , etc. |

Figures 3-14 illustrates the schematics developed at NTNU for the printed circuit boards (PCBs) designs of the converter. These include both the main power circuit, as well as the various auxiliary and control circuits.

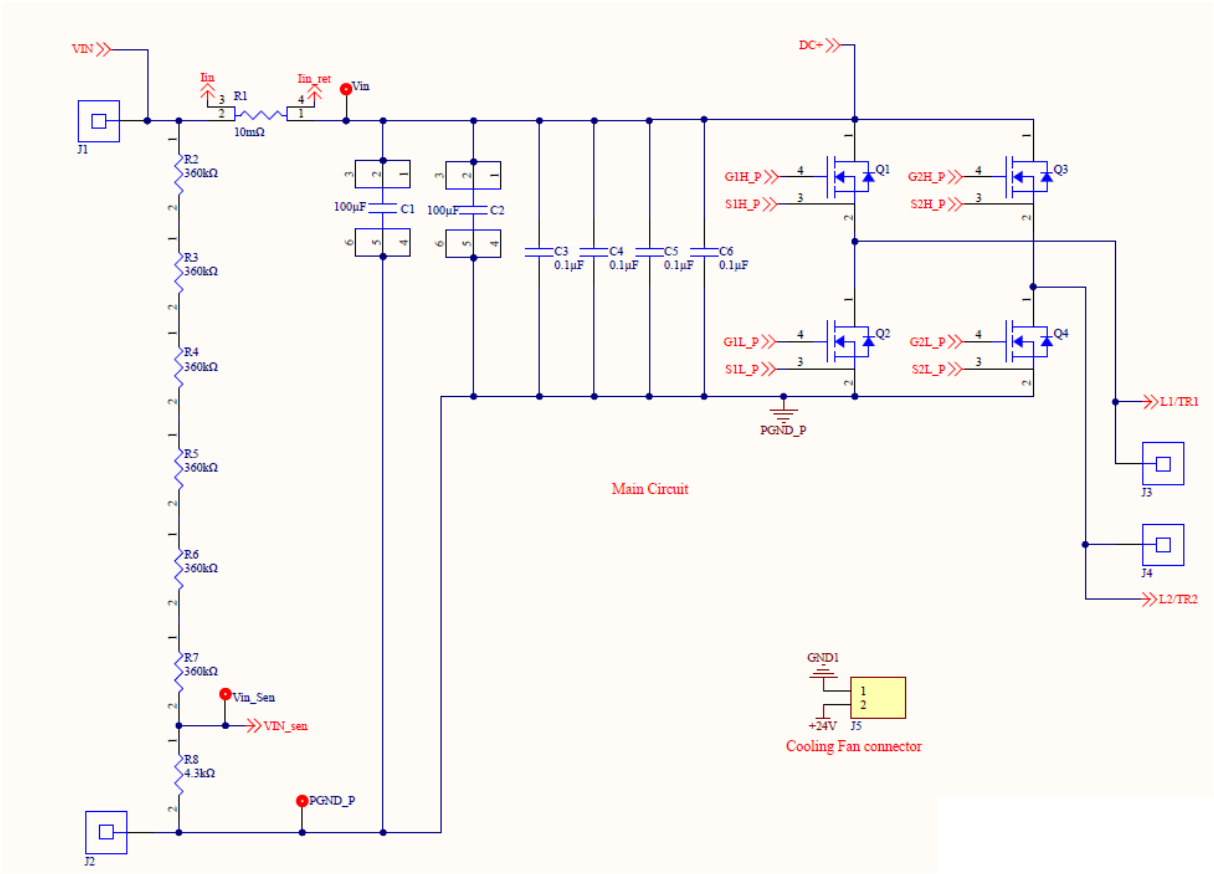



Fig. 3: Schematic diagram for the PCB design of the primary full-bridge main power circuit.

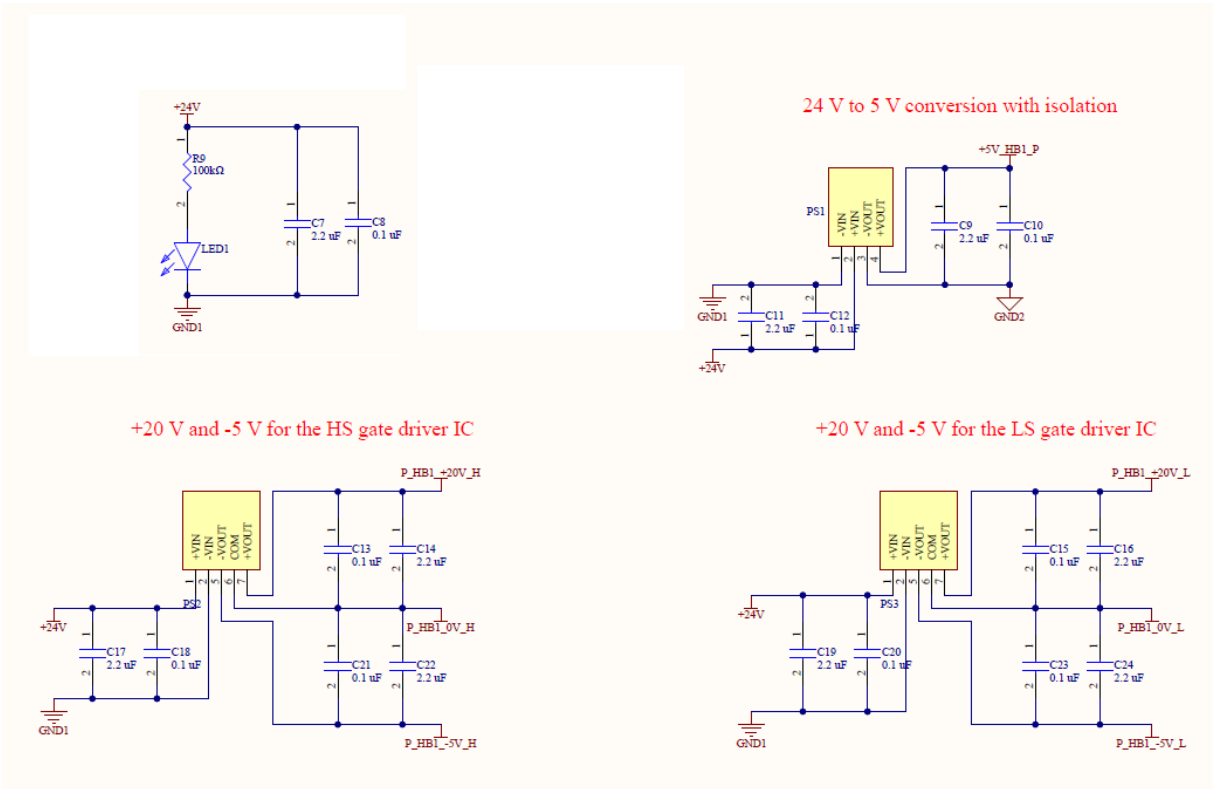



Fig. 4: Schematic diagram for the PCB design of the gate drive circuit power supplies for one half-bridge circuit.

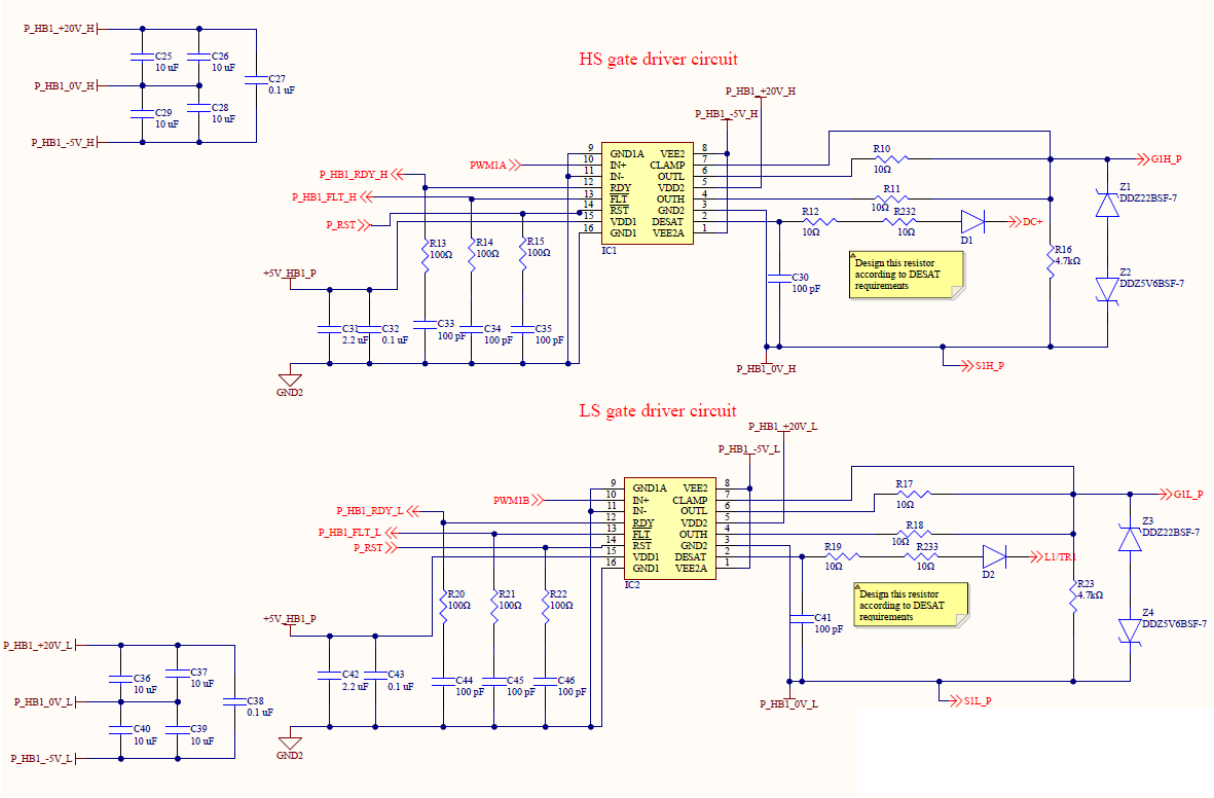



Fig. 5: Schematic diagram for the PCB design of the gate drive circuit suitable for one half-bridge circuit.

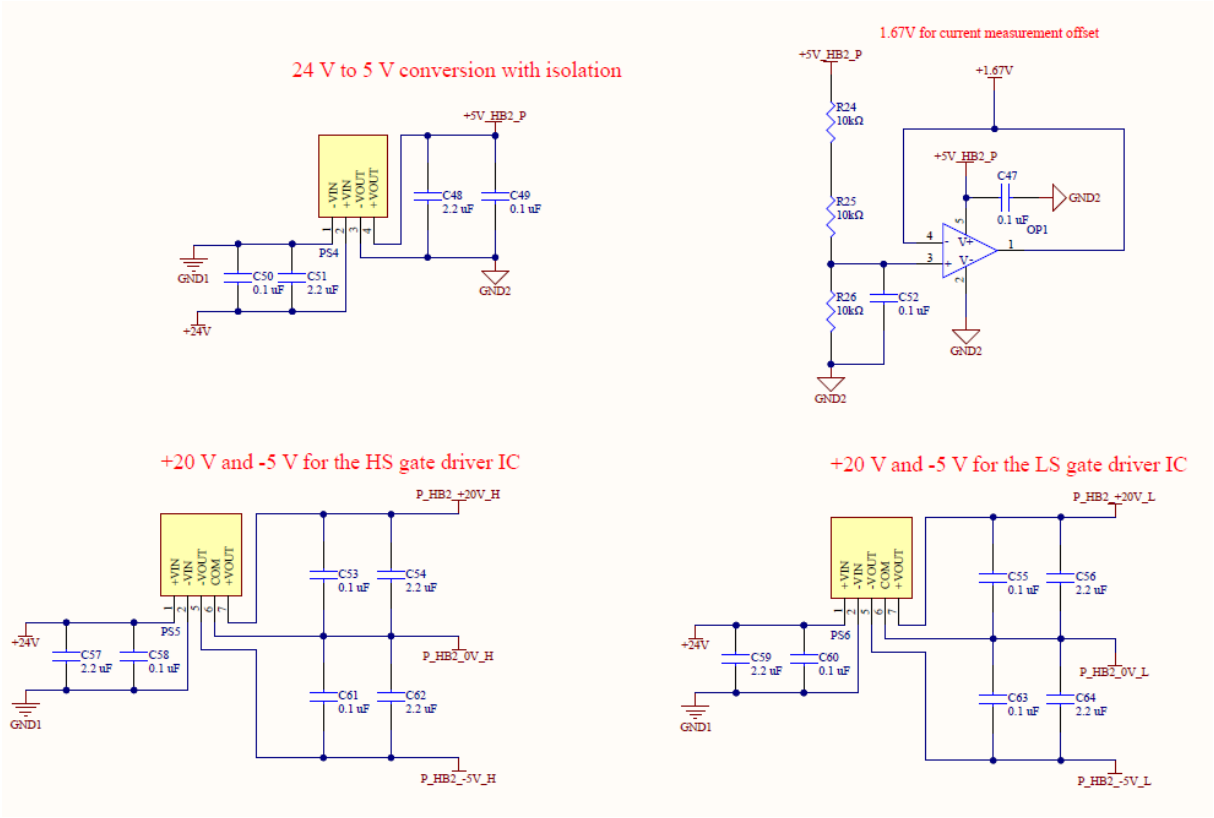



Fig. 6: Schematic diagram for the PCB design of the gate drive circuit power supplies for the second half-bridge circuit.

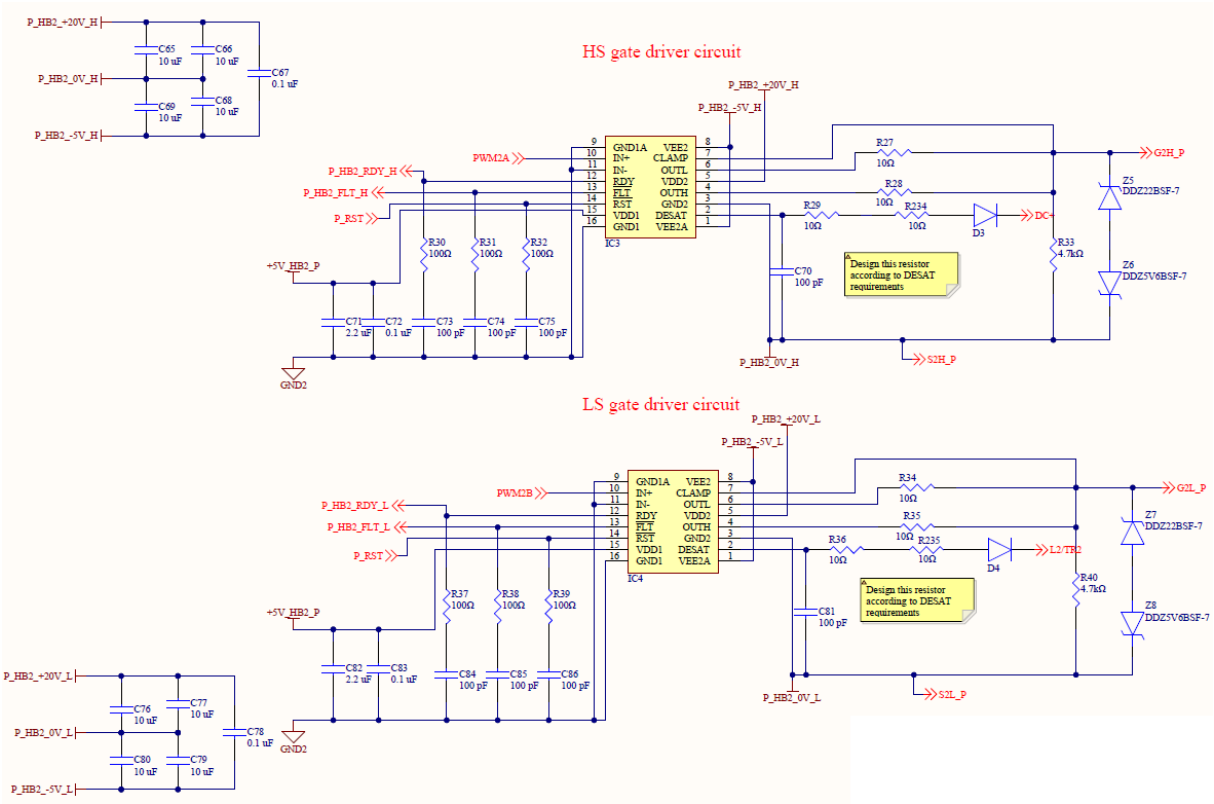



Fig. 7: Schematic diagram for the PCB design of the gate drive circuit suitable for the second half-bridge circuit.

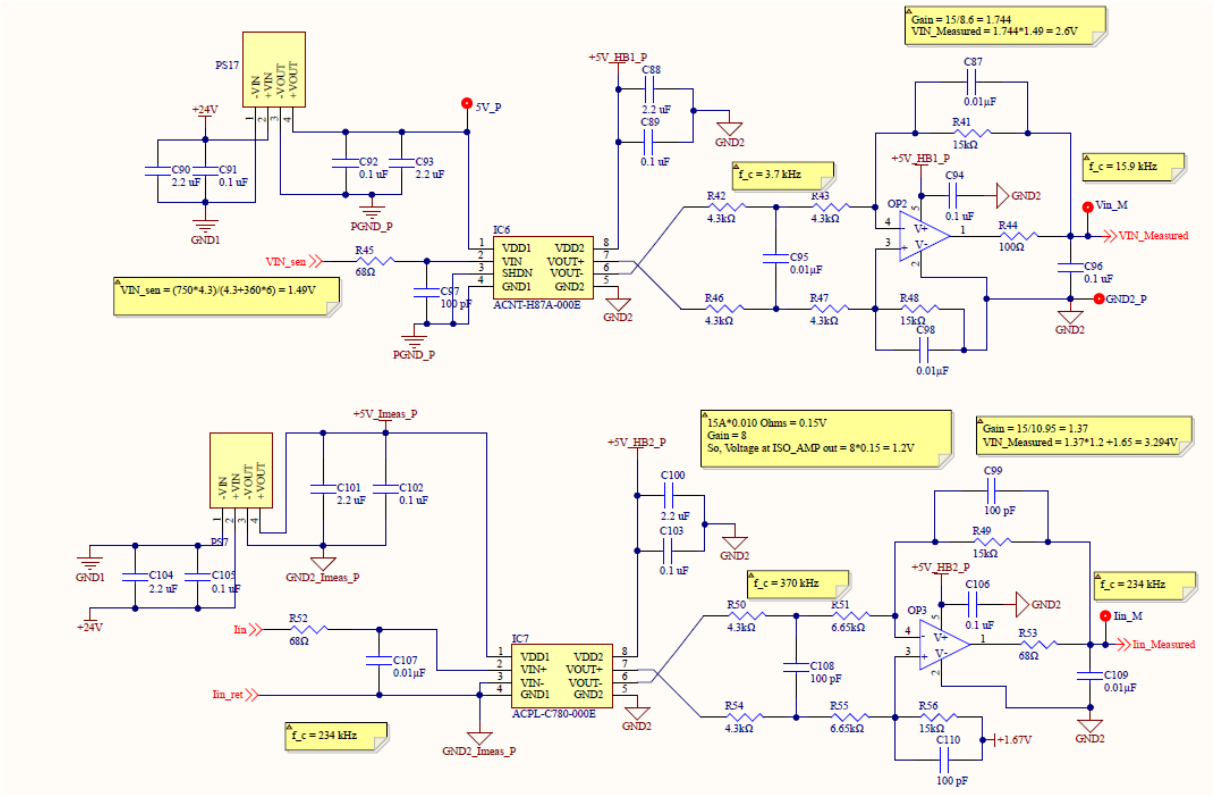



Fig. 8: Schematic diagram for the PCB design of the input DC voltage and current measurement circuits.

The secondary full-bridge circuit design is similar to that of the primary full-bridge circuit with minor variations (positive and negative gate driver voltages are +15V/-5V, measurement circuit is designed according to the DC voltage and current ratings of the secondary full-bridge). Therefore, the schematics for the secondary full-bridge are not included here.

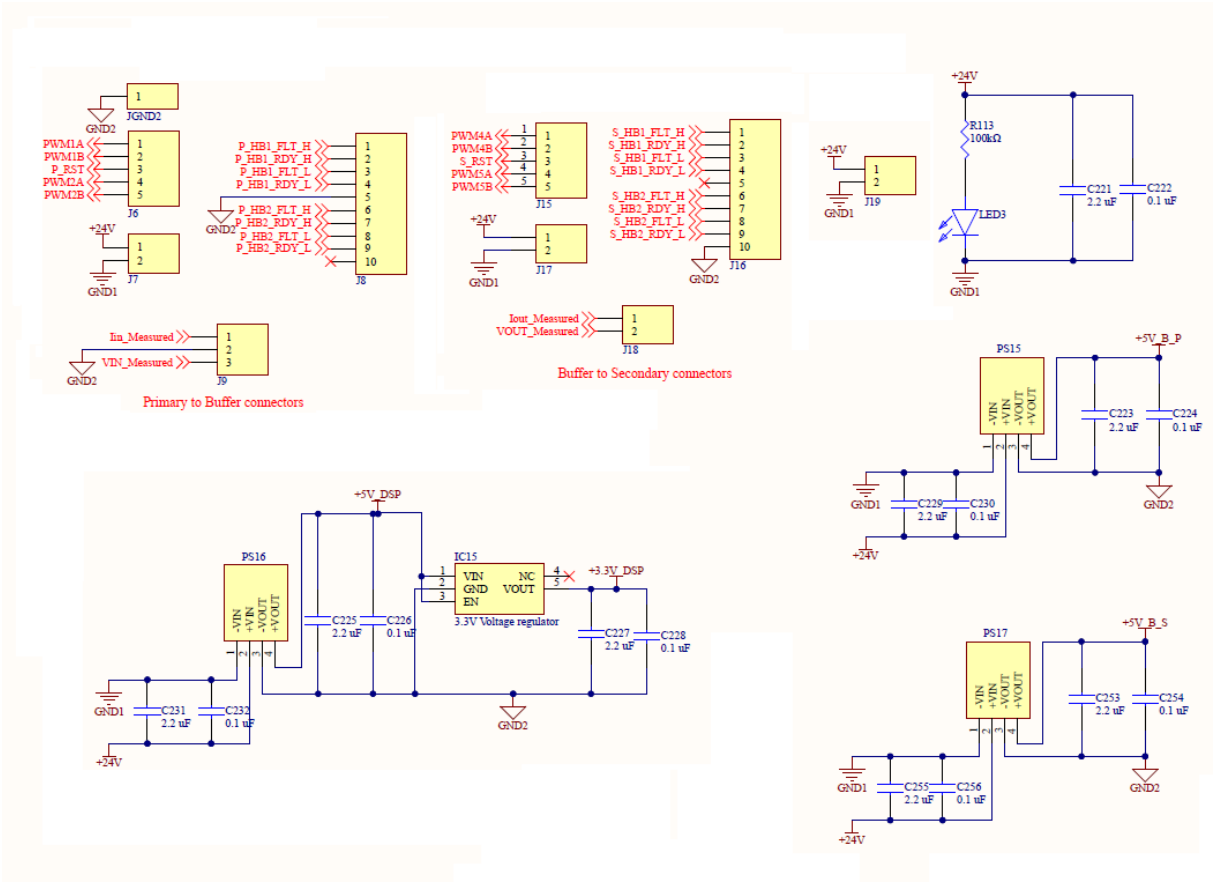



Fig. 9: Schematic diagram for the signal connectors on the primary and secondary full-bridge PCBs and power supply circuits on the signal conditioning/buffer PCB.

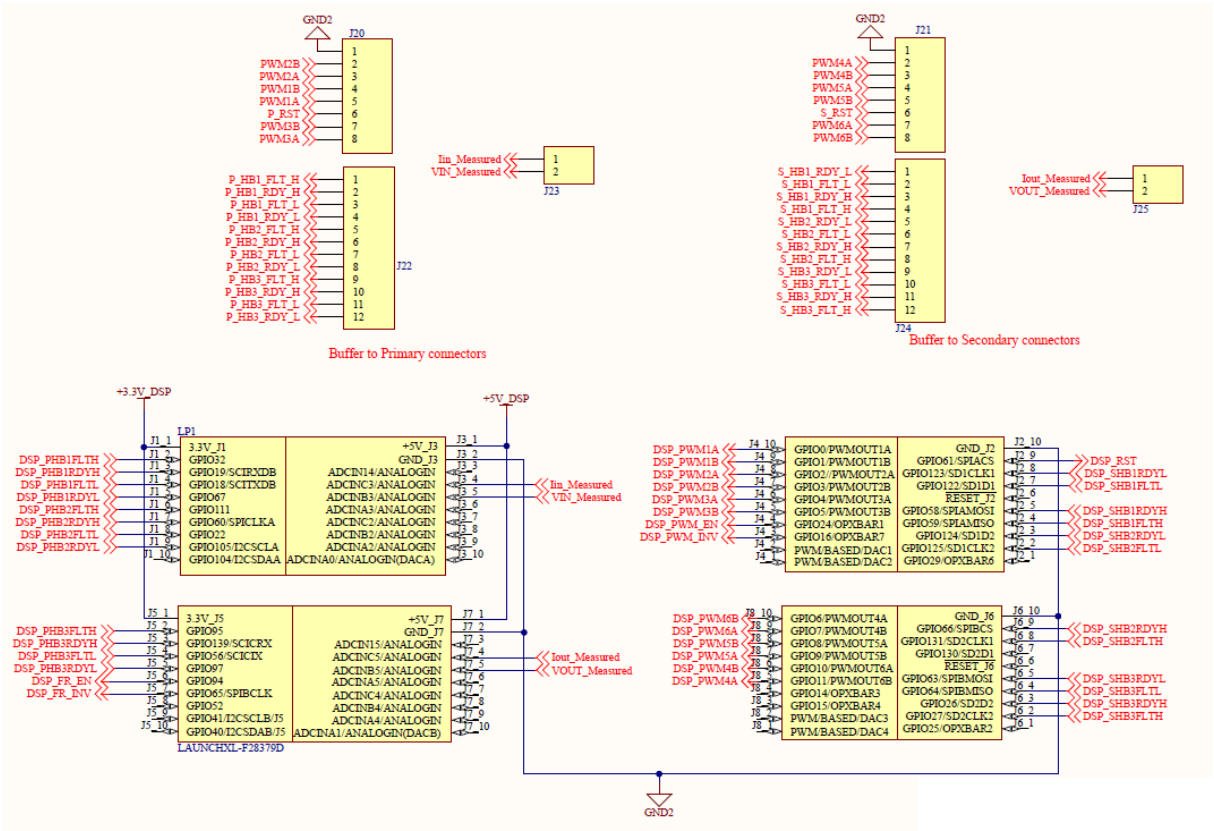



Fig. 10: Schematic diagram for the signal connectors and the F28379D Launchpad connections on the signal conditioning/buffer PCB.

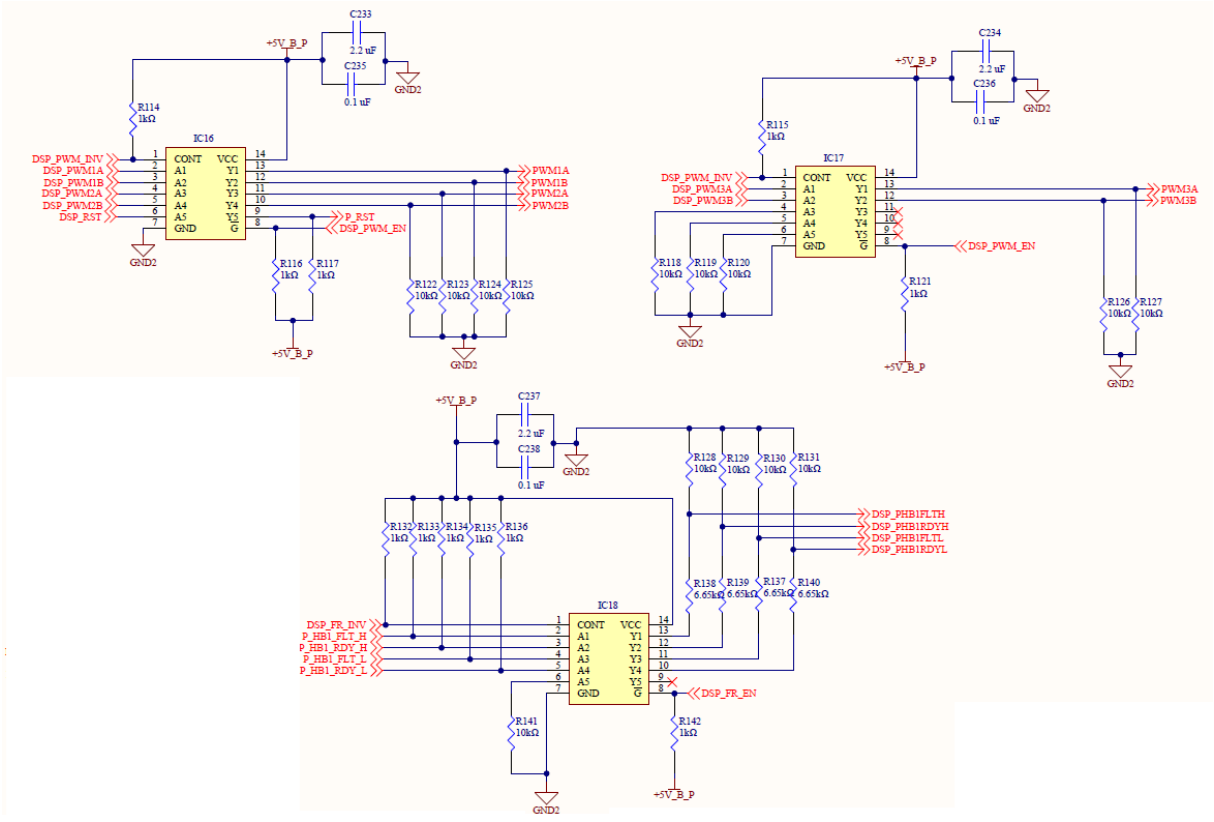


Fig. 11: Schematic diagram for the signal buffer circuits of 6 PWM signals and gate driver status signals from one primary half-bridge on the signal conditioning/buffer PCB.

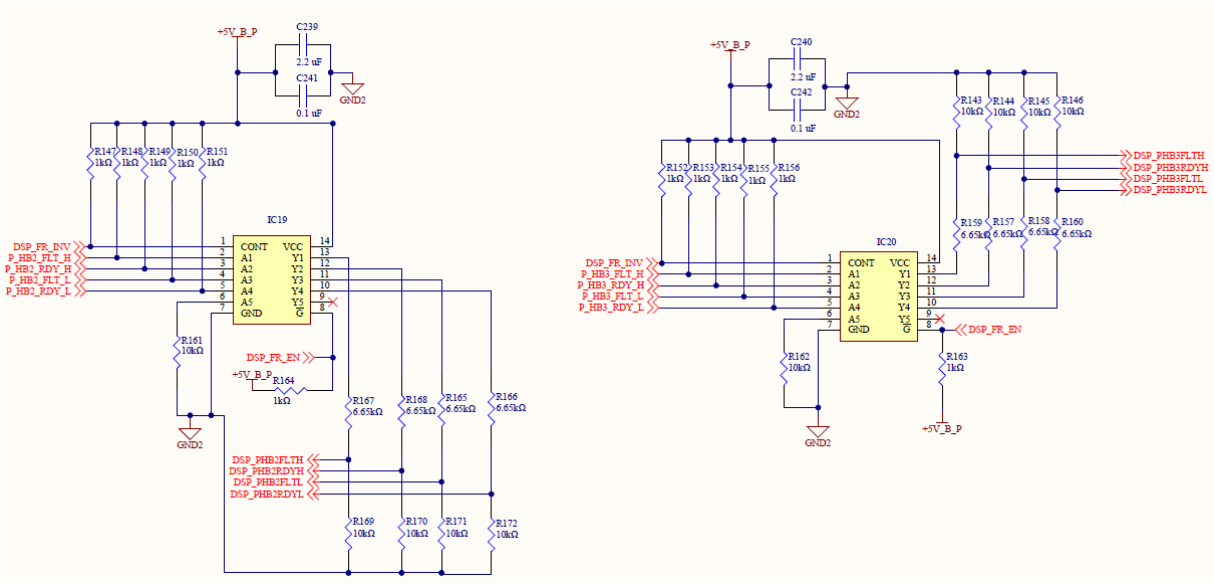



Fig. 12: Schematic diagram for the gate driver status signals from another primary half-bridge and one secondary-1 half-bridge on the signal conditioning/buffer PCB.

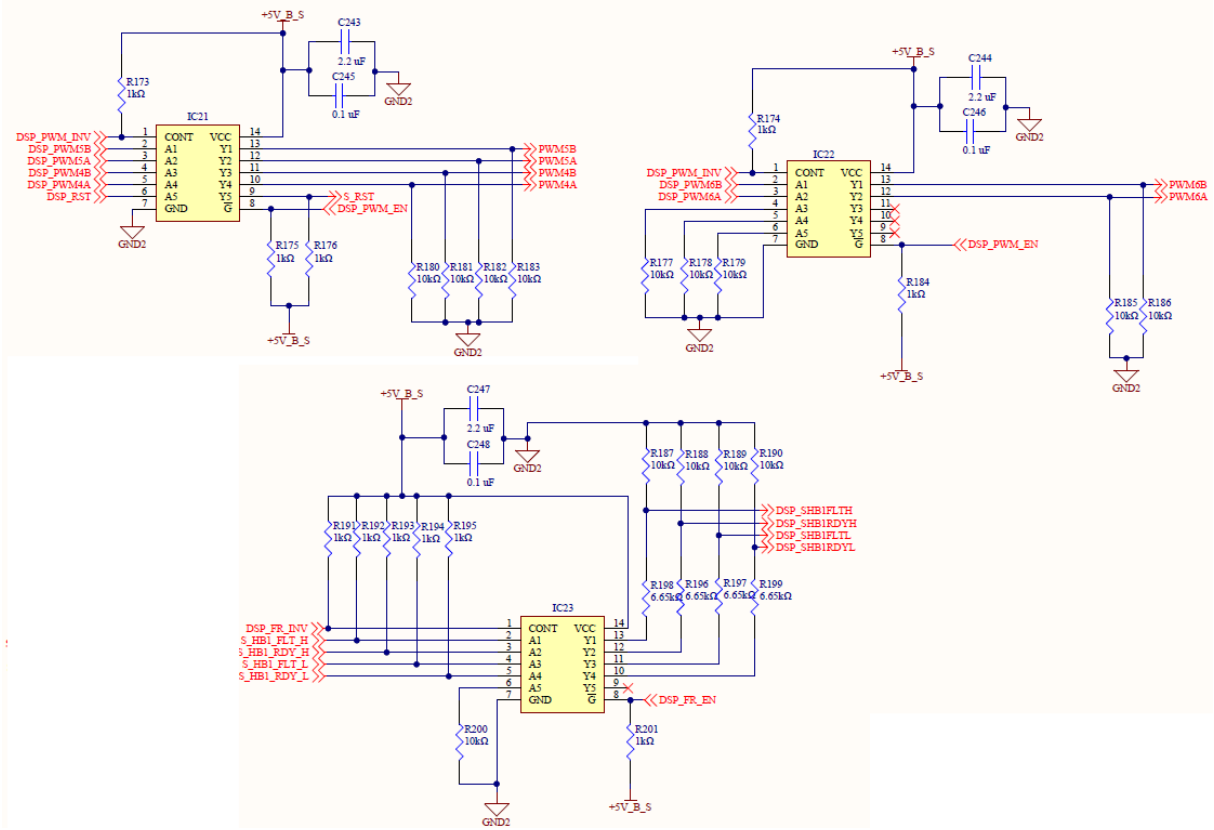



Fig. 13: Schematic diagram for the signal buffer circuits of 6 PWM signals and gate driver status signals from another secondary-1 half-bridge on the signal conditioning/buffer PCB.

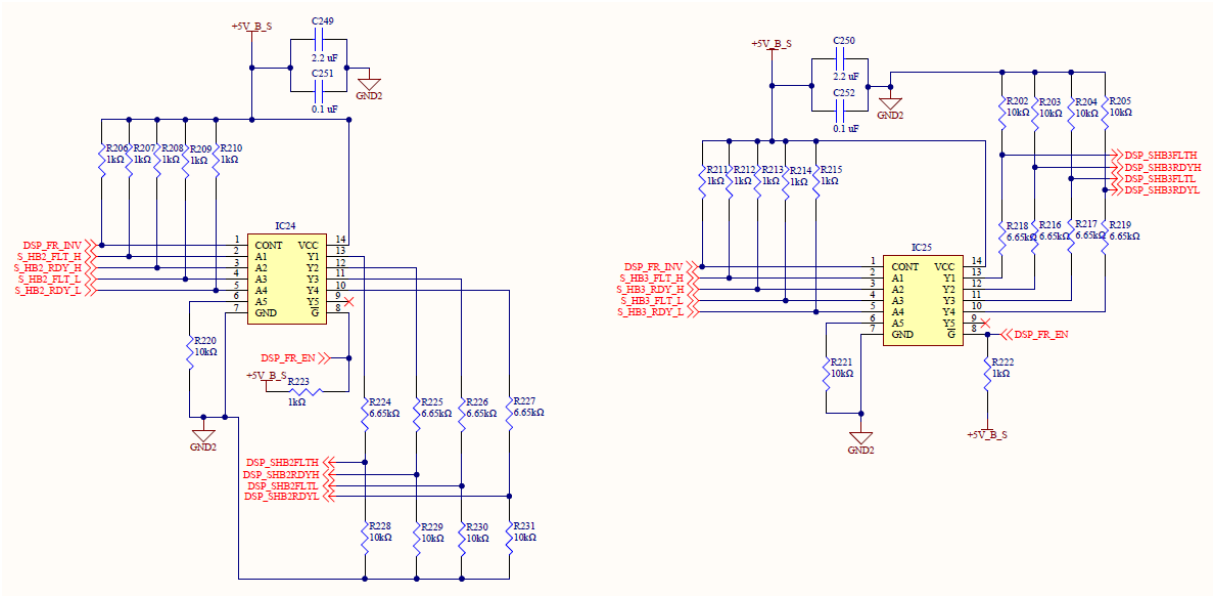
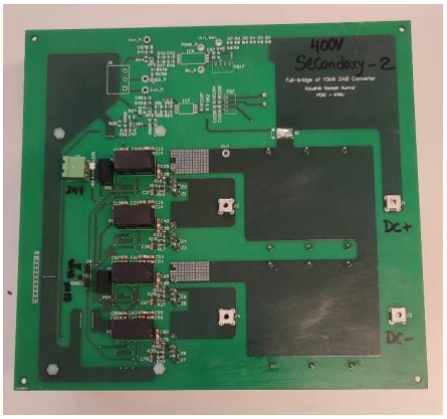
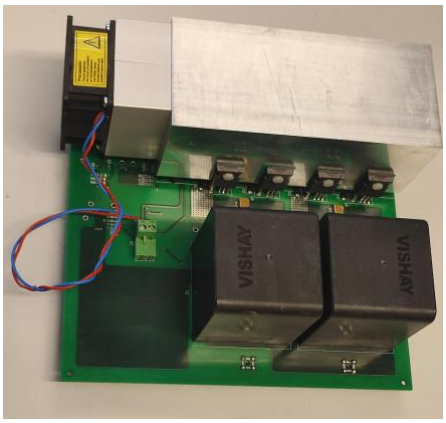



Fig. 14: Schematic diagram for the gate driver status signals from two secondary-2 half-bridges on the signal conditioning/buffer PCB.

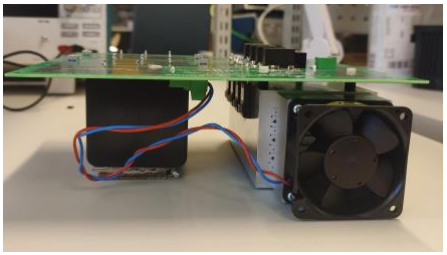
Photos of the components:



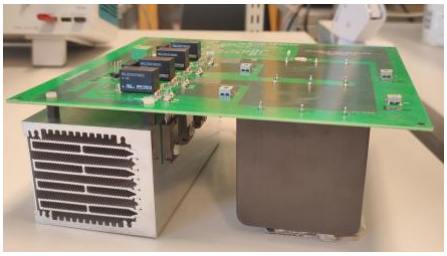

Sub-module: Top view




Sub-module: Bottom view



Sub-module: Side view 1

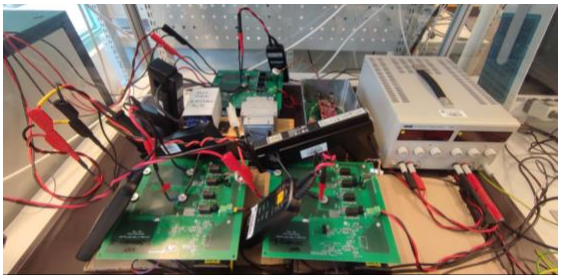


Sub-module: Side view 2

Signal conditioning/buffer board with F28379D Launchpad


Inductor


Three-port Transformer



Three-port DAB setup 2

Fig. 15: Photos of the various components comprising the three-port isolated DC/DC converter laboratory prototype.

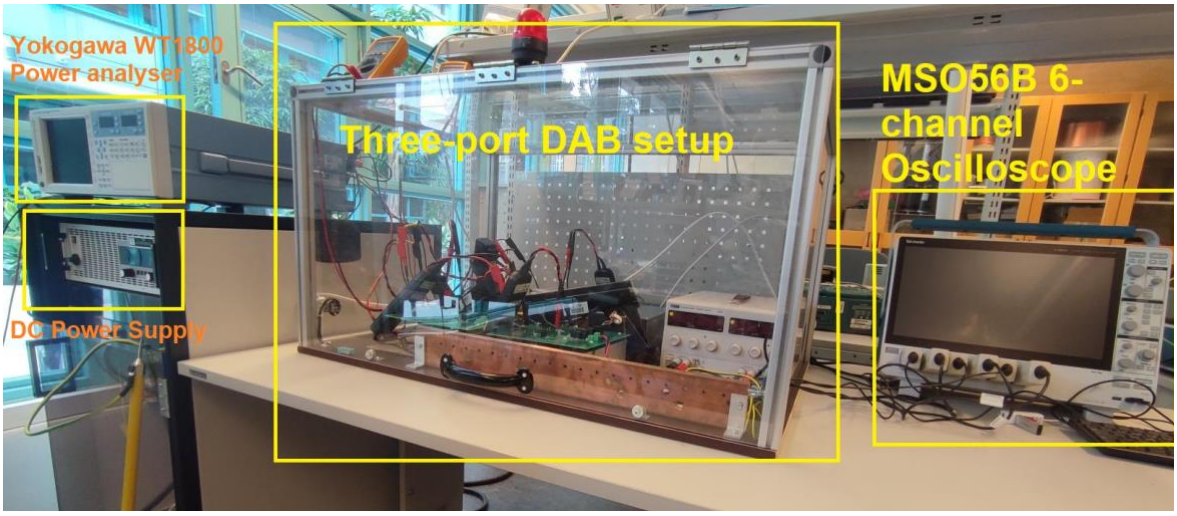



Fig. 16: Photo of the three-port DAB laboratory setup 2

## 3. *Exemplary Experimental Measurements*

A set of electrical measurements in the three-port isolated DC/DC power converter have been performed at various power levels. Figure 17 shows exemplary electrical measurements of the voltages and currents across the output of the primary full-bridge circuit and across the inputs of the two secondary full bridges. The operating conditions for Fig. 17 are Vin = 750V, Vout = 400V, Power = 5kW and with parallel-connected secondaries. Figure 18 shows measurements of the input and output power, as well as power losses and efficiency at 5 kW of load.

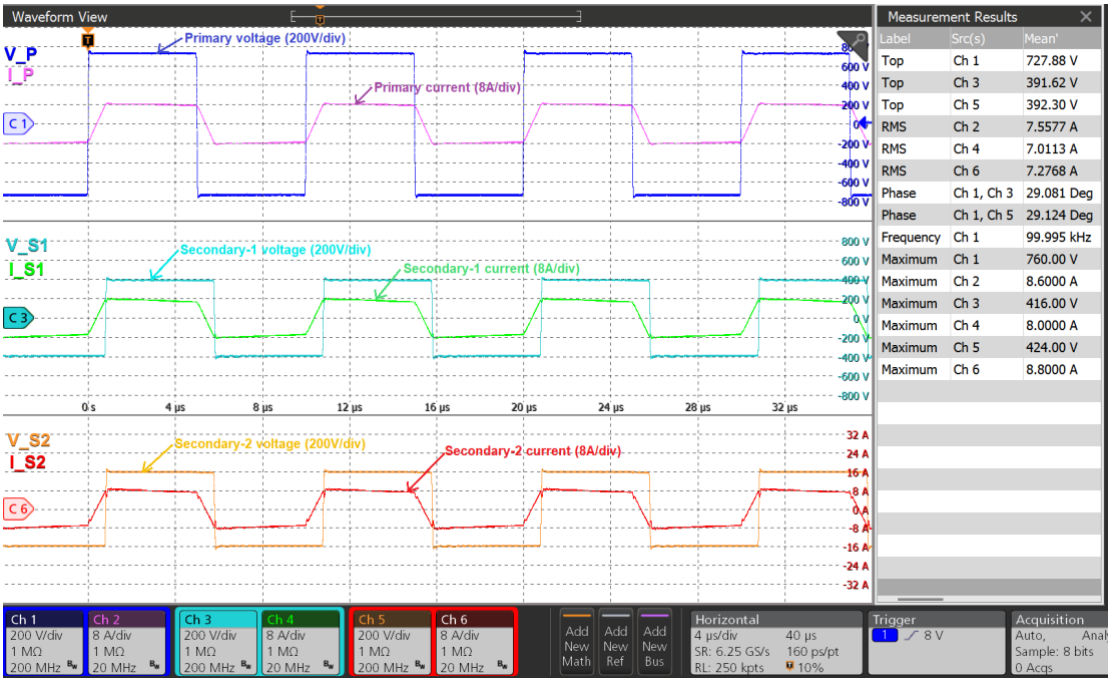



Fig. 17: Exemplary electrical measurements of the three-port isolated DC/DC converter using the MSO56B Tektronix 6-channel Oscilloscope.

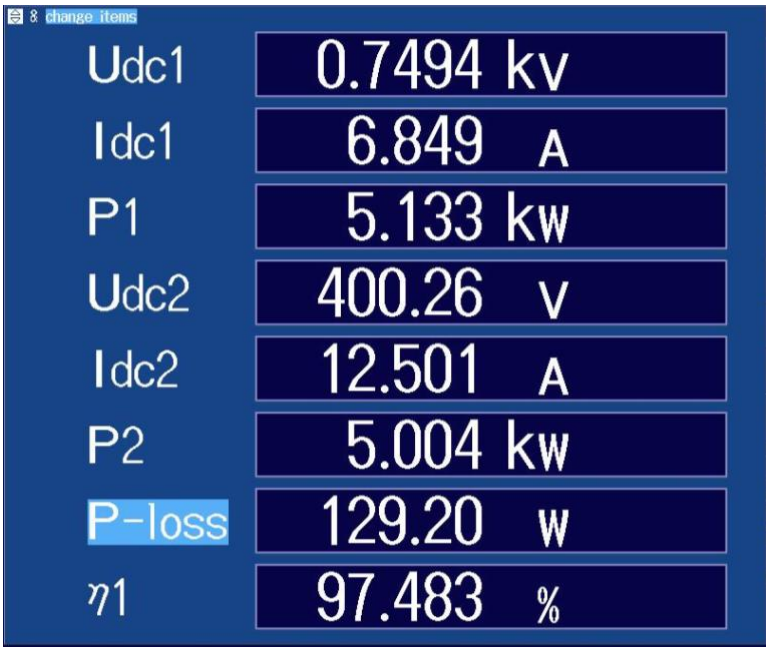



Fig. 18: Power analyser measurements for the input and output power, voltage and currents, as well as power losses and efficiency using the Yokogawa WT1800 power.

With measurements at various power levels, the efficiency and power losses at these power levels are plotted in Figs. 19 and 20, respectively.

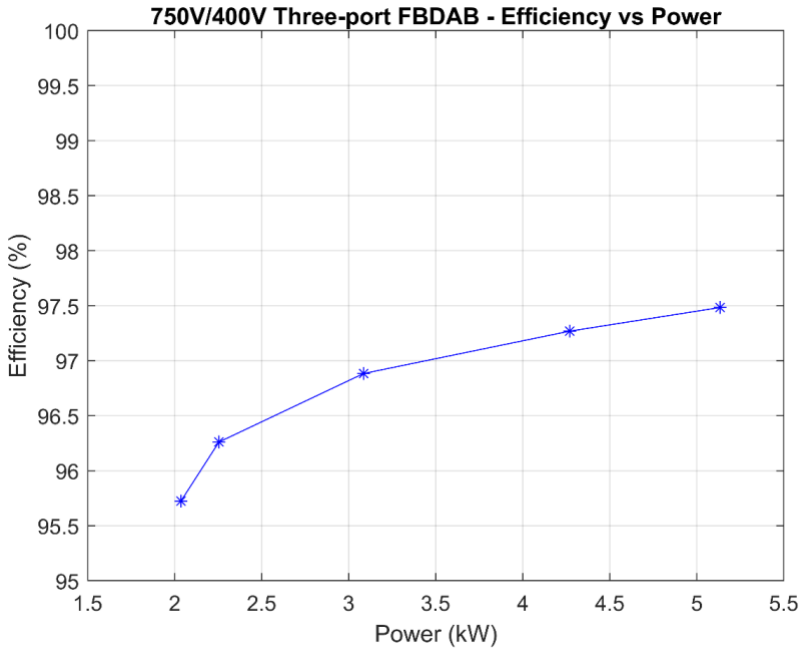



Fig. 19: Total converter efficiency at various power levels based on measurements.

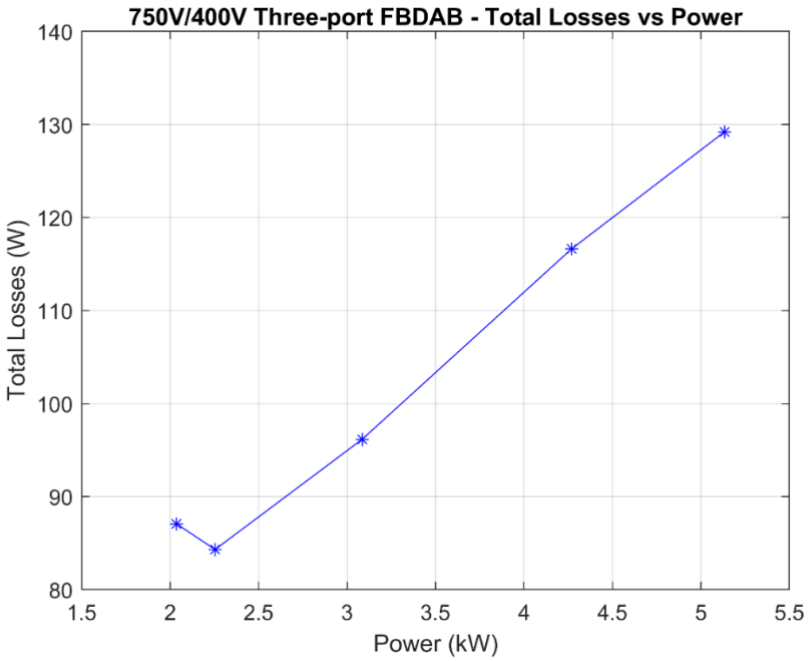



Fig. 20: Total converter losses at various power levels based on measurements.

## 4. Conclusions

In this report the-hardware design of the three-port isolated DC/DC converter was presented in detail. The various design parameters have been listed, while the schematics used for the design of the PCBs were also presented. Moreover, characterization of the magnetic components (i.e.,

inductor and three-port T/F) are shown in order to give a better overview of their required characteristics for this specific converter design. Exemplary electrical measurements at the nominal conditions of voltage and at 5 kW are included in this deliverable and demonstrate the expected correct operation of the converter. Finally, the software developed for the digital microcontroller that controls the operation of the converter is appended in this deliverable.

## 5. Appendix

Figures 21 – 27 illustrate the inductor and transformer parameters characterized using Keysight's E4990A impedance analyser.

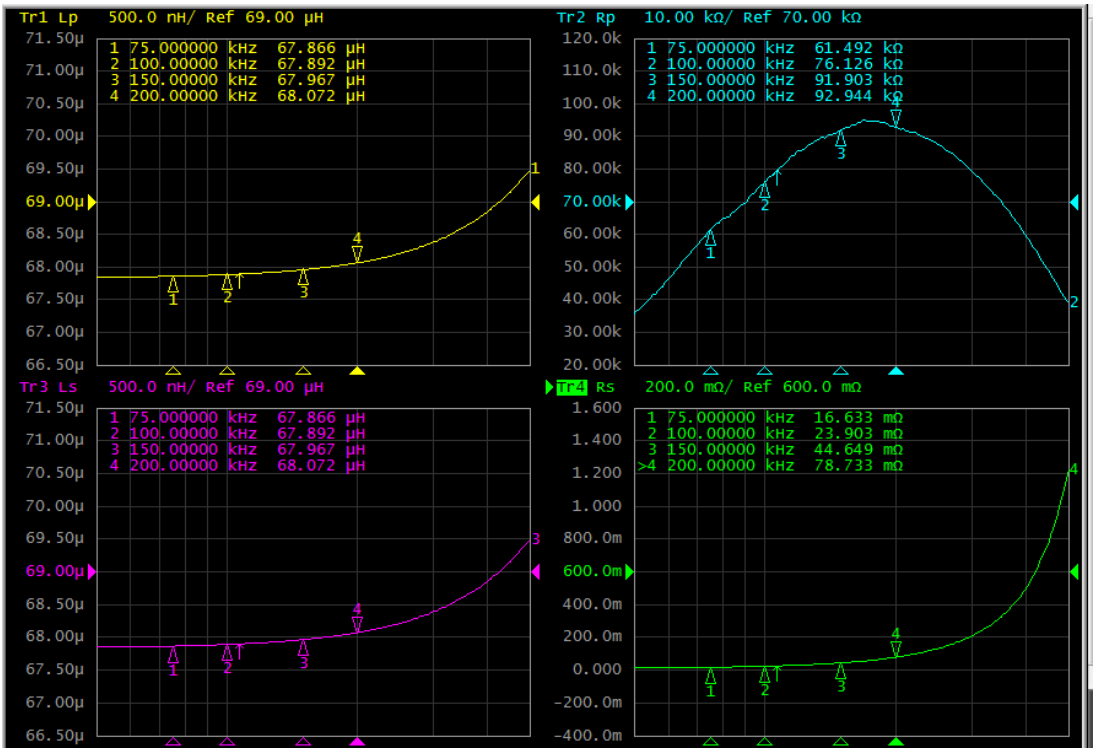



Fig. 21: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the inductor, measured at various frequencies.

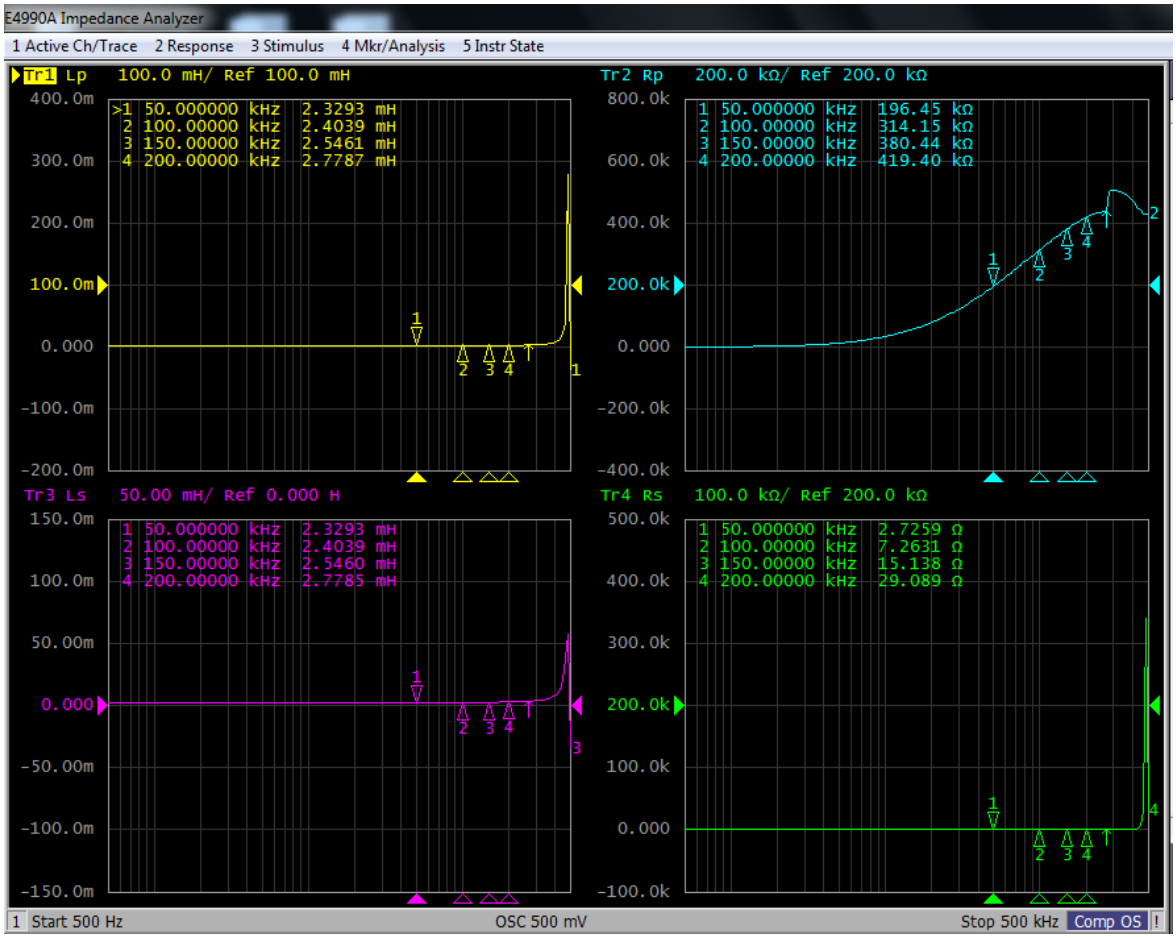



Fig. 22: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the transformer primary winding, measured from the open circuit test.

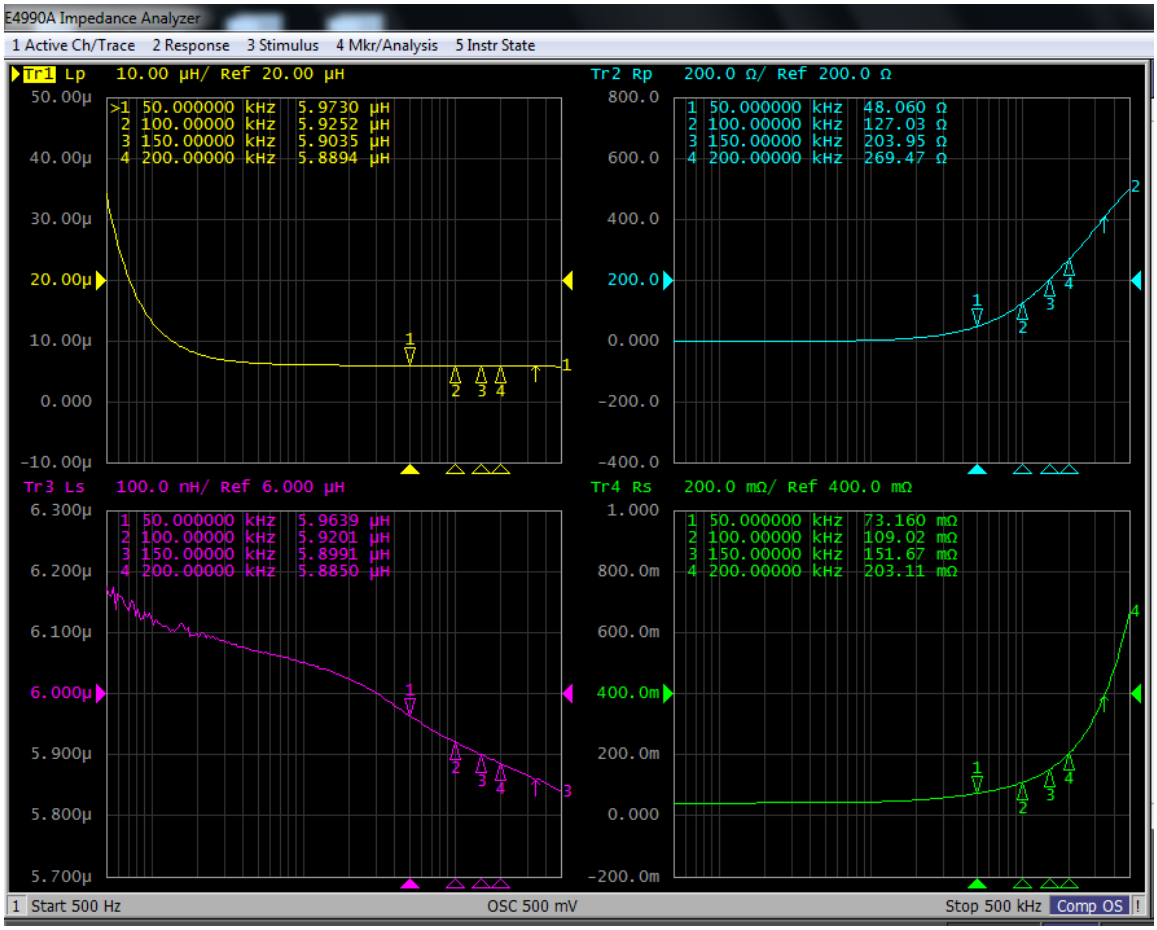



Fig. 23: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the transformer primary winding, measured from the short circuit test.

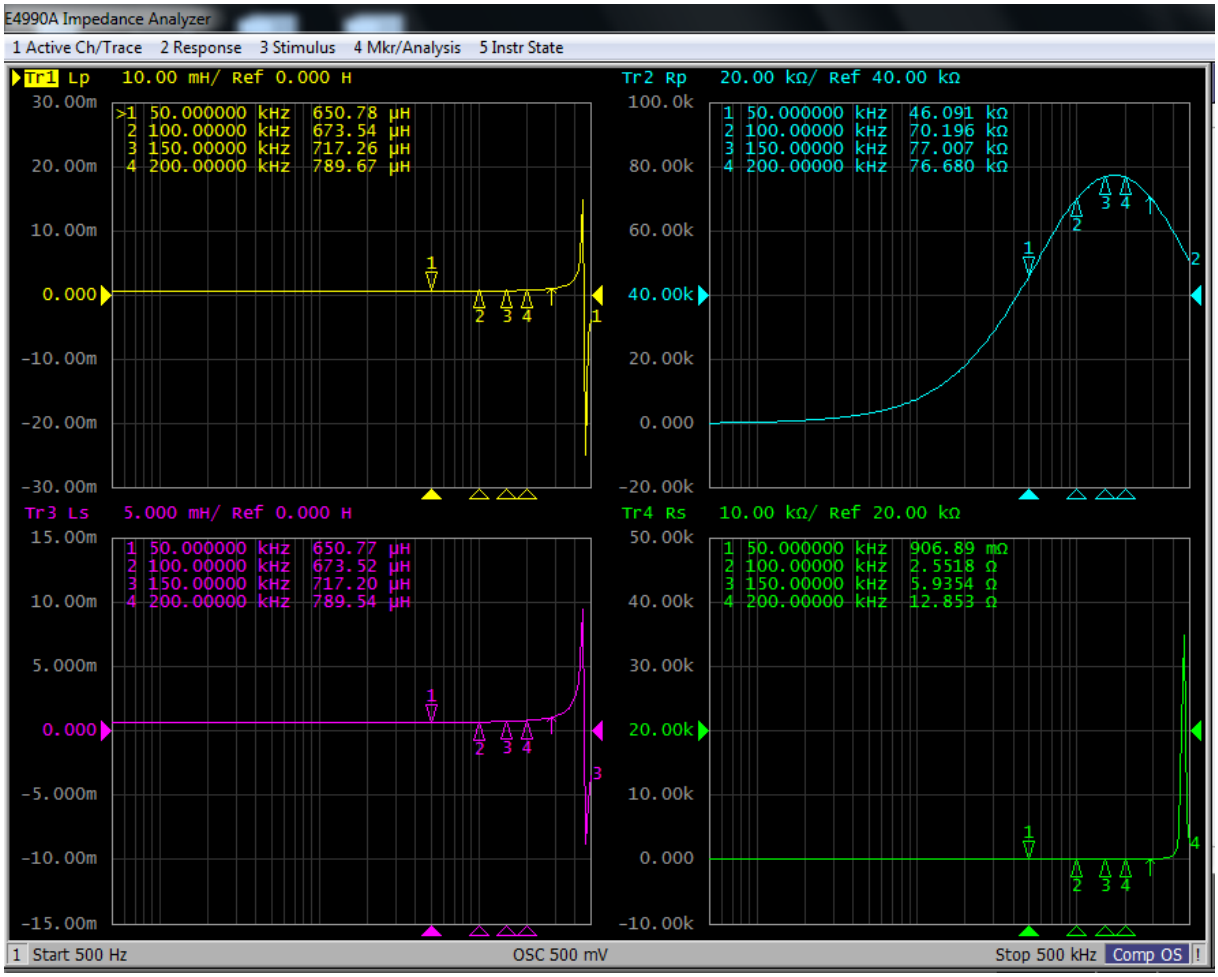



Fig. 24: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the transformer secondary winding-1, measured from the open circuit test.

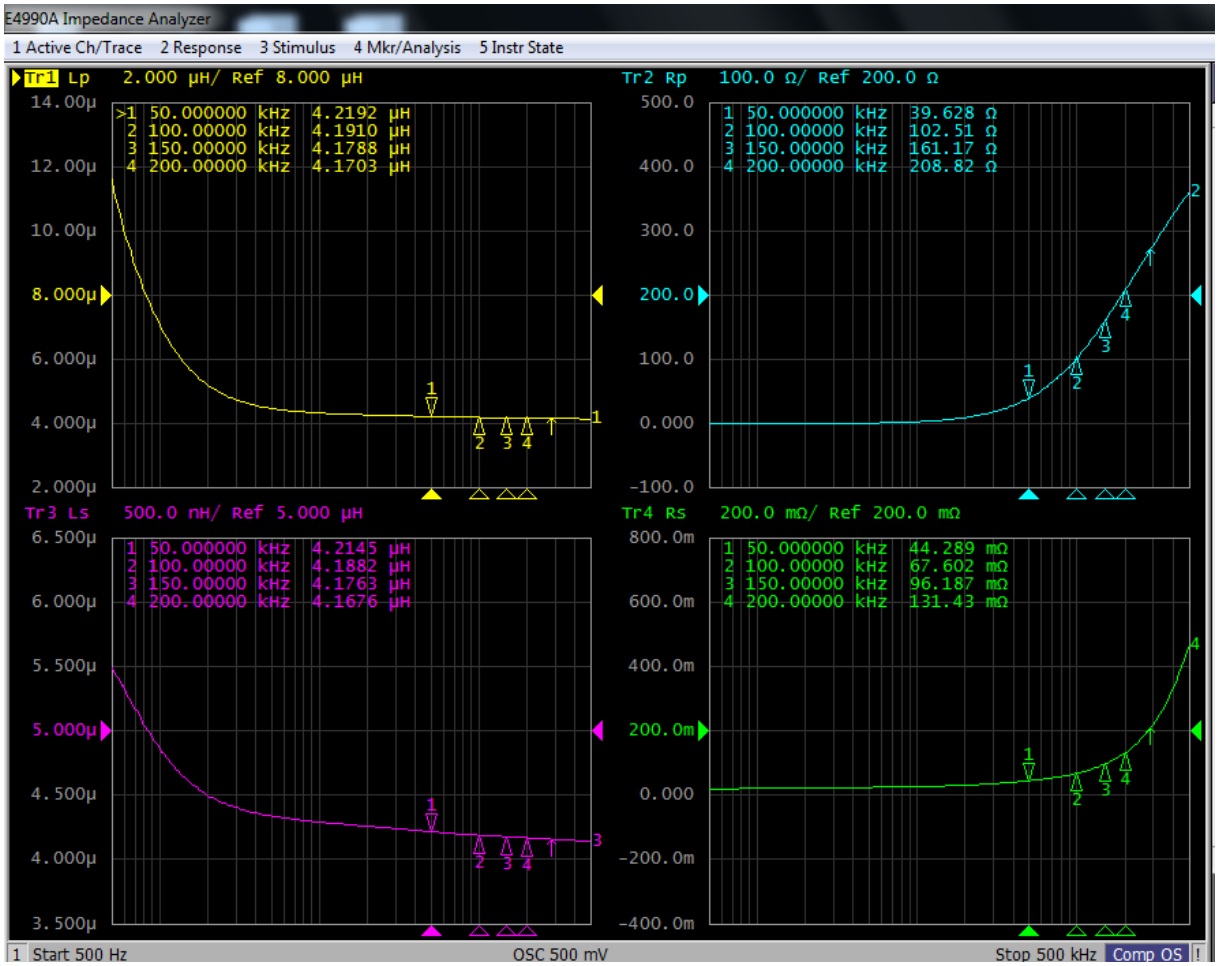



Fig. 25: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the transformer secondary winding-1, measured from the short circuit test.

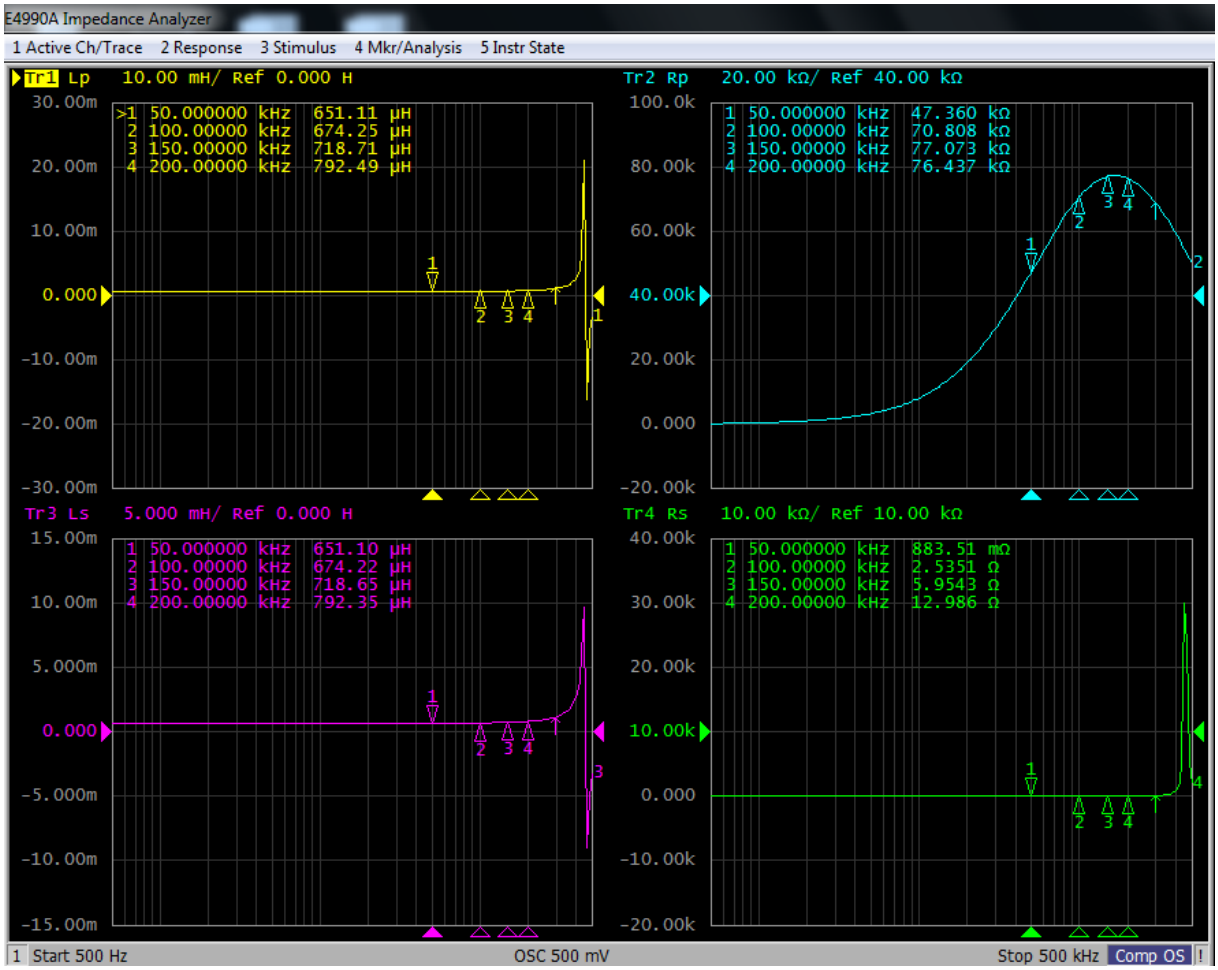



Fig. 26: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the transformer secondary winding-2, measured from the open circuit test.

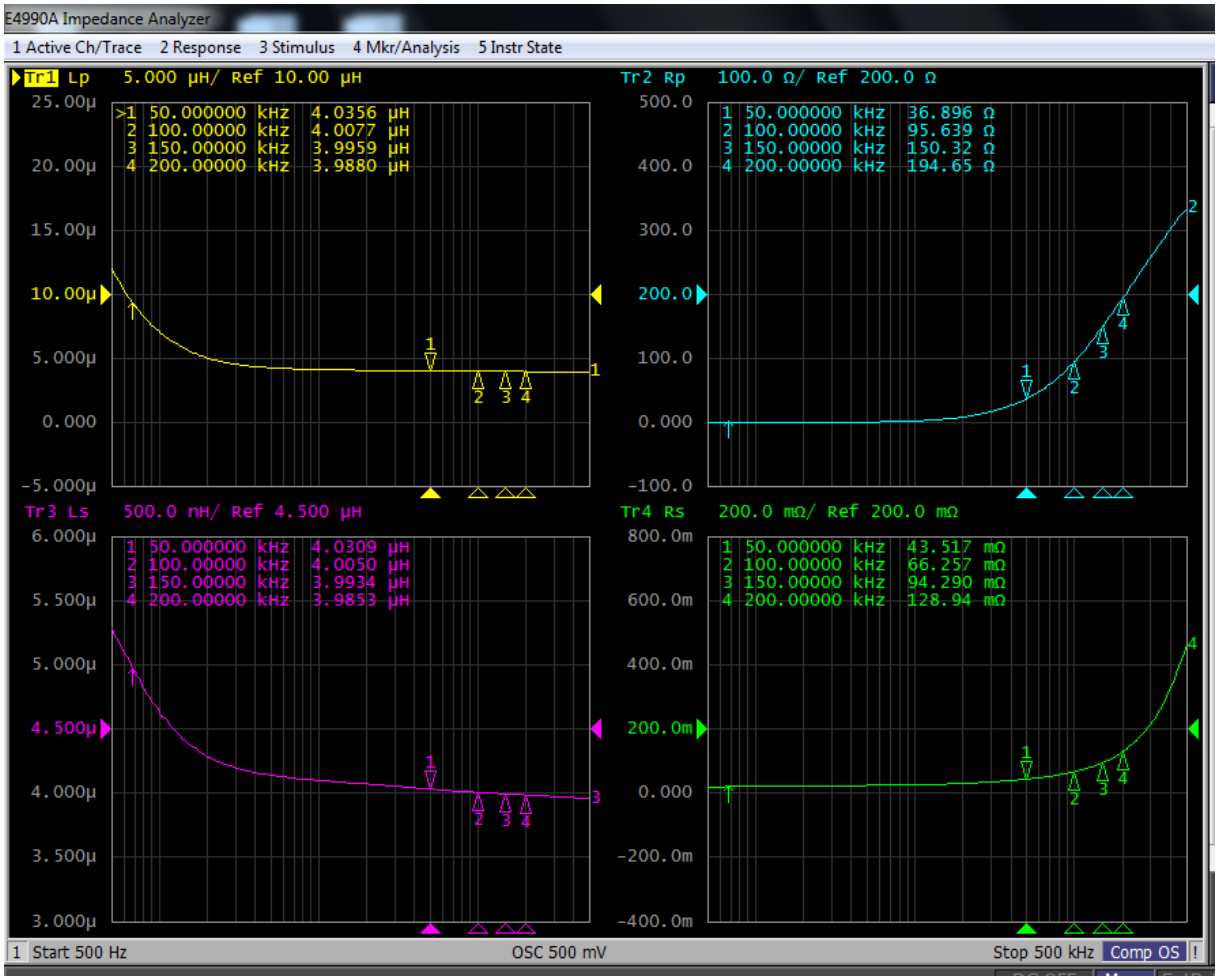



Fig. 27: Parallel inductance ($L_p$), parallel resistance ($R_p$), series inductance ($L_s$), and series resistance ($R_s$) values of the transformer secondary winding-2, measured from the short circuit test.

The microcontroller code for the F28379D Launchpad was developed using the code composer studio (CCS) version 10.4.0. The entire code was organised into separate files for the ease of understanding and faster debugging. The various code files are given below:

**Main_file_TPDAB.c:**

```
/*
 Author      : Kaushik Naresh Kumar
 Organization: Norwegian University of Science and Technology (NTNU)
 Email       : kaushik.n.kumar@ntnu.no

 Description:
To test the Three-port Full-bridge DAB converter:
1) with option to input the required phase-shift value (Facilitated by SCI).
2) with option to obtain voltage and current measurements from the converter for closed-loop control (facilitated by ADC).
*/

//
// Included Files
//
#include "F28x_Project.h"
#include "PWM.h"
#include "SCI.h"
#include "ADC.h"


void main(void)
{

    uint16_t ReceivedChar;
    unsigned char *msg;

//
// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the F2837xD_SysCtrl.c file.
//
   InitSysCtrl();

//
// Step 2. Initialize GPIO:
// This example function is found in the F2837xD_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
//
   InitGpio();


//
// Step 3. Clear all __interrupts and initialize PIE vector table:
//
   DINT;

//
// Initialize the PIE control registers to their default state.
// The default state is all PIE __interrupts disabled and flags
// are cleared.
// This function is found in the F2837xD_PieCtrl.c file.
//
   InitPieCtrl();
```

```
54
55 //
56 // Disable CPU __interrupts and clear all CPU __interrupt flags:
57 //
58    IER = 0x0000;
59    IFR = 0x0000;
60
61 //
62 // Initialize the PIE vector table with pointers to the shell Interrupt
63 // Service Routines (ISR).
64 // This will populate the entire table, even if the __interrupt
65 // is not used in this example.  This is useful for debug purposes.
66 // The shell ISR routines are found in F2837xD_DefaultIsr.c.
67 // This function is found in F2837xD_PieVect.c.
68 //
69    InitPieVectTable();
70
71 //
72 // Enable global Interrupts and higher priority real-time debug events:
73 //
74    EINT;   // Enable Global __interrupt INTM
75    ERTM;   // Enable Global realtime __interrupt DBGM
76
77
78 //
79 // Setup SCI
80 //
81    configureSCI();
82
83    initSCIAFIFO();                        // Initialize the SCI FIFO
84    initSCIAEchoback();                    // Initialize SCI for echoback
85
86    msg = "\r\n\nThree-port full-bridge DAB converter test!!!\0";
87    transmitSCIAMessage(msg);
88
89    msg = "\r\n\nFollowing phase shifts can be set between the primary and the two secondary H-bridges:\0";
90    transmitSCIAMessage(msg);
91
92    msg = "\r\n\nType 2, 3, 4 or 5 to set phase shift = 11.5deg, 16deg, 22deg or 28deg respectively!\n\0";
93    transmitSCIAMessage(msg);
94
95    msg = "\r\nType 6, 7, 8 or 9 to set phase shift = 36deg, 40deg, 44deg or 48deg respectively!\n\0";
96    transmitSCIAMessage(msg);
97
98    msg = "\r\n\nEnter a character to set the phase shift: (Any other character: PS = 10deg)\0";
99    transmitSCIAMessage(msg);
100
101   //
102   // Wait for character
103   //
104          while(SciaRegs.SCIFFRX.bit.RXFFST == 0)
105          {
```

```
106
107         } // wait for empty state
108
109         //
110         // Get character
111         //
112         ReceivedChar = SciaRegs.SCIRXBUF.all;
113
114         //
115         // Echo character back
116         //
117         msg = "\r\nYou sent: \0";
118         transmitSCIAMessage(msg);
119         transmitSCIAChar(ReceivedChar);
120         switch (ReceivedChar)
121         {
122             case 50:                        //ASCII value for character '2' is 50
123             {
124                 EPWM1_PS = 13;
125                 EPWM1_PS_2 = 11;
126                 transmitSCIAMessage("\r\nPhase shift set to 11.5deg! \0");
127                 break;
128             }
129             case 51:                        //ASCII value for character '3' is 51
130             {
131                 EPWM1_PS = 19;
132                 EPWM1_PS_2 = 18;
133                 transmitSCIAMessage("\r\nPhase shift set to 16deg! \0");
134                 break;
135             }
136             case 52:                        //ASCII value for character '4' is 52
137             {
138                 EPWM1_PS = 28;
139                 EPWM1_PS_2 = 28;
140                 transmitSCIAMessage("\r\nPhase shift set to 22deg!! \0");
141                 break;
142             }
143             case 53:                        //ASCII value for character '5' is 53
144             {
145                 EPWM1_PS = 36;
146                 EPWM1_PS_2 = 36;
147                 transmitSCIAMessage("\r\nPhase shift set to 28deg!! \0");
148                 break;
149             }
150             case 54:                        //ASCII value for character '6' is 54
151             {
152                 EPWM1_PS = 47;
153                 EPWM1_PS_2 = 45;
154                 transmitSCIAMessage("\r\nPhase shift set to 36deg \0");
155                 break;
156             }
157             case 55:                        //ASCII value for character '7' is 55
```

```
158             {
159                 EPWM1_PS = 53;
160                 EPWM1_PS_2 = 51;
161                 transmitSCIAMessage("\r\nPhase shift set to 40deg \0");
162                 break;
163             }
164             case 56:                    //ASCII value for character '8' is 56
165             {
166                 EPWM1_PS = 58;
167                 EPWM1_PS_2 = 56;
168                 transmitSCIAMessage("\r\nPhase shift set to 44deg \0");
169                 break;
170             }
171             case 57:                    //ASCII value for character '9' is 57
172             {
173                 EPWM1_PS = 64;
174                 EPWM1_PS_2 = 62;
175                 transmitSCIAMessage("\r\nPhase shift set to 48deg \0");
176                 break;
177             }
178             default:
179             {
180                 EPWM1_PS = 11;
181                 EPWM1_PS_2 = 9;
182                 transmitSCIAMessage("\r\nPhase shift set to 10deg \0");
183                 break;
184             }
185         }
186
187   configurePWM();
188   ConfigureADC();
189   SetupADCSoftware();
190
191   msg = "\r\n\nEnter 1 to enable PWM or any other character to disable PWM\n\0";
192   transmitSCIAMessage(msg);
193
194 //
195 // Infinite loop with option to enable or disable PWM
196 //
197   for(;;)
198   {
199
200       msg = "\r\n\n Now, enter a character to enable/disable PWM: \0";
201       transmitSCIAMessage(msg);
202
203       //
204       // Wait for character
205       //
206       while(SciaRegs.SCIFFRX.bit.RXFFST == 0)
207       {
208
209       } // wait for empty state
```

```
210
211         //
212         // Get character
213         //
214         ReceivedChar = SciaRegs.SCIRXBUF.all;
215
216         //
217         // Echo character back
218         //
219         msg = "\r\nYou sent: \0";
220         transmitSCIAMessage(msg);
221         transmitSCIAChar(ReceivedChar);
222         if (ReceivedChar == 49)                         //ASCII value for character '1' is 49
223         {
224             enable = 1;
225             transmitSCIAMessage("\n PWM enabled \0");
226         }
227         else
228         {
229             enable = 0;
230             transmitSCIAMessage("\n PWM disabled \0");
231         }
232
233     }
234
235 }
236
237 //
238 // EPWM1 ISR
239 //
240 __interrupt void epwm1_isr(void)
241 {
242     if (enable == 1)
243     {
244        enablePWM();
245     }
246     else
247     {
248       disablePWM();
249     }
250
251     StoreADCvalues();
252     //
253     // Clear INT flag for this timer
254     //
255     EPwm1Regs.ETCLR.bit.INT = 1;
256
257     //
258     // Acknowledge this interrupt to receive more interrupts from group 3
259     //
260     PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
261
262  }
263 //
264 // End of file
265 //
266
```

## PWM.h:

```c
/*
Author      : Kaushik Naresh Kumar
Organization: Norwegian University of Science and Technology (NTNU)
Email       : kaushik.n.kumar@ntnu.no

Description:
Contains PWM related functions
*/

//
// Globals
//
uint16_t enable = 0;
uint16_t EPWM1_PS = 11;     // Default Phase Shift between primary and secondary 1 = 10 deg. [(10/360)*(2*TBPRD)-3]
uint16_t EPWM1_PS_2 = 9;   // Default Phase Shift between primary and secondary 2 = 10 deg. [(10/360)*(2*TBPRD)-3]


//
// Defines
//

//To achieve 50% duty cycle
#define EPWM1_TBPRD 250  // TBPRD = (1/2)*(1/(HSPCLKDIV*CLKDIV))*(f_ePWMCLK)/(f_PWM);  f_ePWMCLK = 50MHz, f_sw = 100kHz; SYSCLK = 100MHz (predefined _LAUNCHXL_F28379D not included -> SYSCLK/2 = 100MHz)
#define EPWM1_CMPA 125

//For Dead time = 200ns
#define EPWM1_RED 10     // 1 ePWMCLK cycle = 1/50Mhz = 20ns --> For 200ns -> RED=FED=10
#define EPWM1_FED 10

//
//Function prototypes
//
void configurePWM(void);
void enablePWM(void);
void disablePWM(void);
__interrupt void epwm1_isr(void);
void InitEPwm1(void);
void InitEPwm2(void);
void InitEPwm6(void);
void InitEPwm5(void);
void InitEPwm3(void);
void InitEPwm4(void);


void configurePWM(void)
{

   // Include before initialize GPIO as ePWM

      CpuSysRegs.PCLKCR2.bit.EPWM1 = 1; //enable ePWM1 module clock
      CpuSysRegs.PCLKCR2.bit.EPWM2 = 1; //enable ePWM2 module clock

      CpuSysRegs.PCLKCR2.bit.EPWM6 = 1; //enable ePWM6 module clock
```

```
54        CpuSysRegs.PCLKCR2.bit.EPWM5 = 1; //enable ePWM5 module clock
55
56        CpuSysRegs.PCLKCR2.bit.EPWM3 = 1; //enable ePWM3 module clock
57        CpuSysRegs.PCLKCR2.bit.EPWM4 = 1; //enable ePWM4 module clock
58
59    //
60    //Enable ePWMs
61    //
62
63        EALLOW;
64
65        GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1; // Disable pull-up on GPIO0 (EPWM1A)
66        GpioCtrlRegs.GPAPUD.bit.GPIO1 = 1; // Disable pull-up on GPIO1 (EPWM1B)
67
68        GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1; // Disable pull-up on GPIO2 (EPWM2A)
69        GpioCtrlRegs.GPAPUD.bit.GPIO3 = 1; // Disable pull-up on GPIO3 (EPWM2B)
70
71        GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1; // Disable pull-up on GPIO11 (EPWM6B)[EPWM4A in PCB]
72        GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1; // Disable pull-up on GPIO10 (EPWM6A)[EPWM4B in PCB]
73
74        GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1; // Disable pull-up on GPIO9 (EPWM5B)[EPWM5A in PCB]
75        GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1; // Disable pull-up on GPIO8 (EPWM5A)[EPWM5B in PCB]
76
77        GpioCtrlRegs.GPAPUD.bit.GPIO4 = 1; // Disable pull-up on GPIO4 (EPWM3A)
78        GpioCtrlRegs.GPAPUD.bit.GPIO5 = 1; // Disable pull-up on GPIO5 (EPWM3B)
79
80        GpioCtrlRegs.GPAPUD.bit.GPIO7 = 1; // Disable pull-up on GPIO7 (EPWM4B)[EPWM6A in PCB]
81        GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1; // Disable pull-up on GPIO6 (EPWM4A)[EPWM6B in PCB]
82
83        GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // Configure GPIO0 as EPWM1A
84        GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1; // Configure GPIO1 as EPWM1B
85
86        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1; // Configure GPIO2 as EPWM2A
87        GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1; // Configure GPIO3 as EPWM2B
88
89        GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1; // Configure GPIO11 as (EPWM6B)[EPWM4A in PCB]
90        GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1; // Configure GPIO10 as (EPWM6A)[EPWM4B in PCB]
91
92        GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1; // Configure GPIO9 as (EPWM5B)[EPWM5A in PCB]
93        GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1; // Configure GPIO8 as (EPWM5A)[EPWM5B in PCB]
94
95        GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1; // Configure GPIO4 as (EPWM3A)
96        GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1; // Configure GPIO5 as (EPWM3B)
97
98        GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1; // Configure GPIO7 as (EPWM4B)[EPWM6A in PCB]
99        GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1; // Configure GPIO6 as (EPWM4A)[EPWM6B in PCB]
100
101       //
102       //Configure DSP_PWM_EN, DSP_PWM_INV pins, DSP_FR_EN and DSP_FR_INV pins
103       //
104
```

```
105       GpioCtrlRegs.GPAPUD.bit.GPIO24 = 0;  // Enable pull-up on GPIO24 (Used as DSP_PWM_EN in signal buffer board).
106       GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 0; // Set as GPIO.
107       GpioCtrlRegs.GPADIR.bit.GPIO24 = 1;  // Configure as output.
108       GpioDataRegs.GPASET.bit.GPIO24 = 1;  // It should be set low to enable PWM signal output from buffer IC. Default should be high.
109
110       GpioCtrlRegs.GPAPUD.bit.GPIO16 = 0;  // Enable pull-up on GPIO16 (Used as DSP_PWM_INV in signal buffer board).
111       GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0; // Set as GPIO.
112       GpioCtrlRegs.GPADIR.bit.GPIO16 = 1;  // Configure as output.
113       GpioDataRegs.GPASET.bit.GPIO16 = 1;  // Always high! When set low and DSP_PWM_EN = low -> inverts PWM output from the buffer IC.
114
115       GpioCtrlRegs.GPCPUD.bit.GPIO94 = 0;  // Enable pull-up on GPIO94 (Used as DSP_FR_EN in signal buffer board).
116       GpioCtrlRegs.GPCMUX2.bit.GPIO94 = 0; // Set as GPIO.
117       GpioCtrlRegs.GPCDIR.bit.GPIO94 = 1;  // Configure as output.
118       GpioDataRegs.GPCSET.bit.GPIO94 = 1;  // It should be set low to enable FLT/RDY signals. Default should be high.
119
120       GpioCtrlRegs.GPCPUD.bit.GPIO65 = 0;  // Enable pull-up on GPIO65 (Used as DSP_FR_INV in signal buffer board).
121       GpioCtrlRegs.GPCMUX1.bit.GPIO65 = 0; // Set as GPIO.
122       GpioCtrlRegs.GPCDIR.bit.GPIO65 = 1;  // Configure as output.
123       GpioDataRegs.GPCSET.bit.GPIO65 = 1;  // Always high! When set low and DSP_FR_EN = low -> inverts FLT/RDY signals from the buffer IC.
124
125       EDIS;
126
127       //Interrupts that are used in this example are re-mapped to
128             //ISR functions found within this file.
129             //
130                EALLOW;  // This is needed to write to EALLOW protected registers
131                PieVectTable.EPWM1_INT = &epwm1_isr;
132                EDIS;    // This is needed to disable write to EALLOW protected registers
133
134             //
135             // Step 4. Initialize the Device Peripheral. This function can be
136             //         found in F2837xD_CpuTimers.c
137             //
138
139                EALLOW;
140                CpuSysRegs.PCLKCR0.bit.TBCLKSYNC =0;
141                EDIS;
142
143                InitEPwm1();
144                InitEPwm2();
145                InitEPwm6();
146                InitEPwm5();
147                InitEPwm3();
148                InitEPwm4();
149
150                EALLOW;
151                CpuSysRegs.PCLKCR0.bit.TBCLKSYNC =1;
152                EDIS;
153
154                disablePWM();
155
```

```
156                //
157             // Enable CPU INT3 which is connected to EPWM1-3 INT:
158             //
159                IER |= M_INT3;
160
161             //
162             // Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
163             //
164                PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
165
166 }
167
168
169 void enablePWM(void)
170 {
171     GpioDataRegs.GPACLEAR.bit.GPIO24 = 1;   //Set DSP_PWM_EN low to enable PWM signals
172     GpioDataRegs.GPCCLEAR.bit.GPIO94 = 1;   //Set DSP_FR_EN low to enable FLT/RDY signals
173     EALLOW;
174     EPwm1Regs.TZCLR.bit.OST = 1;
175     EPwm2Regs.TZCLR.bit.OST = 1;
176     EPwm6Regs.TZCLR.bit.OST = 1;
177     EPwm5Regs.TZCLR.bit.OST = 1;
178     EPwm3Regs.TZCLR.bit.OST = 1;
179     EPwm4Regs.TZCLR.bit.OST = 1;
180     EDIS;
181 }
182
183
184 void disablePWM(void)
185 {
186     GpioDataRegs.GPASET.bit.GPIO24 = 1;     //Set DSP_PWM_EN high to disable PWM signals
187     GpioDataRegs.GPCSET.bit.GPIO94 = 1;     //Set DSP_FR_EN high to disable FLT/RDY signals
188     EALLOW;
189     EPwm1Regs.TZFRC.bit.OST = 1;
190     EPwm2Regs.TZFRC.bit.OST = 1;
191     EPwm6Regs.TZFRC.bit.OST = 1;
192     EPwm5Regs.TZFRC.bit.OST = 1;
193     EPwm3Regs.TZFRC.bit.OST = 1;
194     EPwm4Regs.TZFRC.bit.OST = 1;
195     EDIS;
196 }
197
198
199 //
200 // Initialize EPWM1 configuration
201 //
202 void InitEPwm1(void)
203 {
204     EPwm1Regs.TBPRD = EPWM1_TBPRD;                // Set timer period (100 kHz)
205     EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;           // Phase is 0
206     EPwm1Regs.TBCTR = 0x0000;                     // Clear counter
```

```
207
208     //
209     // Setup TBCLK
210     //
211     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up down
212     EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading -> Master
213
214     EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;    //Synchronization output at counter = 0
215     EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
216     EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
217
218     EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
219     EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
220     EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
221     EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
222
223     //
224     // Setup compare
225     //
226     EPwm1Regs.CMPA.bit.CMPA = EPWM1_CMPA;
227
228     //
229     // Set actions
230     //
231     EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // CTR = CMPA (up direction)-> Set PWM1A to Zero
232     EPwm1Regs.AQCTLA.bit.CAD = AQ_SET;             // CTR = CMPA (down direction)-> Set PWM1A to one
233
234
235     //
236     // Active High Complimentary PWMs - Setup Deadband
237     //
238     EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
239     EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
240     EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
241     EPwm1Regs.DBRED.bit.DBRED = EPWM1_RED;
242     EPwm1Regs.DBFED.bit.DBFED = EPWM1_FED;
243
244     //
245     // Interrupt
246     //
247     EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;    // Select INT on Zero event
248     EPwm1Regs.ETSEL.bit.INTEN = 1;               // Enable INT
249     EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;          // Generate INT on 1st event
250
251     //
252     //ADC trigger
253     //
254     EPwm1Regs.ETSEL.bit.SOCAEN = 1;             //Enable Epwm1 ADC SOCA
255 //    EPwm1Regs.ETSEL.bit.SOCASEL = 2;            //Enable event time-base counter equal to period (TBCTR = TBPRD)
256     EPwm1Regs.ETSEL.bit.SOCASEL = 4;            //Enable event time-base counter equal to period (TBCTR = CMPA)
257     EPwm1Regs.ETPS.bit.SOCAPRD = 1;             // Generate pulse on 1st event
```

```
    //
    // TripZone
    //
    EALLOW;
    EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
    EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
    EDIS;
}


//
// Initialize EPWM2 configuration
//
void InitEPwm2(void)
{

    EPwm2Regs.TBPRD = EPWM1_TBPRD;               // Set timer period (100 kHz)
    EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;          // Phase is 0
    EPwm2Regs.TBCTR = 0x0000;                    // Clear counter

    //
    // Setup TBCLK
    //
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up down
    EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading -> no phase difference from ePWM1

    EPwm2Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;


    //
    // Setup compare
    //
    EPwm2Regs.CMPA.bit.CMPA = EPWM1_CMPA;

    //
    // Set actions
    //
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;           // CTR = CMPA (up direction)-> Set PWM2A to one
    EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;         // CTR = CMPA (down direction)-> Set PWM2A to zero


    //
    // Active High Complimentary PWMs - Setup Deadband
    //
    EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
    EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
    EPwm2Regs.DBRED.bit.DBRED = EPWM1_RED;
    EPwm2Regs.DBFED.bit.DBFED = EPWM1_FED;
```

```
309
310    //
311    // TripZone
312    //
313    EALLOW;
314    EPwm2Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
315    EPwm2Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
316    EDIS;
317 }
318
319 //
320 // Initialize EPWM6 configuration
321 //
322 void InitEPwm6(void)
323 {
324     EPwm6Regs.TBPRD = EPWM1_TBPRD;                // Set timer period (100 kHz)
325     EPwm6Regs.TBPHS.bit.TBPHS = EPWM1_PS;         // Defined PS value
326     EPwm6Regs.TBCTR = 0x0000;                     // Clear counter
327
328     //
329     // Setup TBCLK
330     //
331     EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up down
332     EPwm6Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading
333
334
335     EALLOW;
336     SyncSocRegs.SYNCSELECT.bit.EPWM4SYNCIN = 0;    // Get sync input from ePWM1
337     EDIS;
338
339     EPwm6Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
340     EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
341     EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;
342
343     //
344     // Setup compare
345     //
346     EPwm6Regs.CMPA.bit.CMPA = EPWM1_CMPA;
347
348     //
349     // Set actions
350     //
351     EPwm6Regs.AQCTLA.bit.CAU = AQ_SET;            // CTR = CMPA (up direction)-> Set PWM6A to one
352     EPwm6Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // CTR = CMPA (down direction)-> Set PWM6A to zero
353
354     //
355     // Active High Complimentary PWMs - Setup Deadband
356     //
357     EPwm6Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
358     EPwm6Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
359     EPwm6Regs.DBCTL.bit.IN_MODE = DBA_ALL;
```

```
    EPwm6Regs.DBRED.bit.DBRED = EPWM1_RED;
    EPwm6Regs.DBFED.bit.DBFED = EPWM1_FED;

   //
   // TripZone
   //
   EALLOW;
   EPwm6Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
   EPwm6Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
   EDIS;

}


//
// Initialize EPWM5 configuration
//
void InitEPwm5(void)
{
    EPwm5Regs.TBPRD = EPWM1_TBPRD;                // Set timer period (100 kHz)
    EPwm5Regs.TBPHS.bit.TBPHS = EPWM1_PS;         // Defined PS value
    EPwm5Regs.TBCTR = 0x0000;                     // Clear counter

    //
    // Setup TBCLK
    //
    EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up down
    EPwm5Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading


    EALLOW;
    SyncSocRegs.SYNCSELECT.bit.EPWM4SYNCIN = 0;    // Get sync input from ePWM1
    EDIS;

    EPwm5Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
    EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    //
    // Setup compare
    //
    EPwm5Regs.CMPA.bit.CMPA = EPWM1_CMPA;

    //
    // Set actions
    //
    EPwm5Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // CTR = CMPA (down direction)-> Set PWM5A to zero
    EPwm5Regs.AQCTLA.bit.CAD = AQ_SET;             // CTR = CMPA (down direction)-> Set PWM5A to one


    //
```

```
411     // Active High Complimentary PWMs - Setup Deadband
412     //
413     EPwm5Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
414     EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
415     EPwm5Regs.DBCTL.bit.IN_MODE = DBA_ALL;
416     EPwm5Regs.DBRED.bit.DBRED = EPWM1_RED;
417     EPwm5Regs.DBFED.bit.DBFED = EPWM1_FED;
418
419     //
420     // TripZone
421     //
422     EALLOW;
423     EPwm5Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
424     EPwm5Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
425
426     EDIS;
427 }
428
429 //
430 // Initialize EPWM3 configuration
431 //
432 void InitEPwm3(void)
433 {
434     EPwm3Regs.TBPRD = EPWM1_TBPRD;                // Set timer period (100 kHz)
435     EPwm3Regs.TBPHS.bit.TBPHS = EPWM1_PS_2;         // Defined PS value
436     EPwm3Regs.TBCTR = 0x0000;                     // Clear counter
437
438     //
439     // Setup TBCLK
440     //
441     EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up down
442     EPwm3Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading
443
444
445     EPwm3Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
446     EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
447     EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;
448
449     //
450     // Setup compare
451     //
452     EPwm3Regs.CMPA.bit.CMPA = EPWM1_CMPA;
453
454     //
455     // Set actions
456     //
457     EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR;            // CTR = CMPA (up direction)-> Set PWM3A to one
458     EPwm3Regs.AQCTLA.bit.CAD = AQ_SET;          // CTR = CMPA (down direction)-> Set PWM3A to zero
459
460     //
461     // Active High Complimentary PWMs - Setup Deadband
```

```
    //
    EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
    EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
    EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
    EPwm3Regs.DBRED.bit.DBRED = EPWM1_RED;
    EPwm3Regs.DBFED.bit.DBFED = EPWM1_FED;

   //
   // TripZone
   //
   EALLOW;
   EPwm3Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
   EPwm3Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
   EDIS;

}


//
// Initialize EPWM4 configuration
//
void InitEPwm4(void)
{
    EPwm4Regs.TBPRD = EPWM1_TBPRD;                // Set timer period (100 kHz)
    EPwm4Regs.TBPHS.bit.TBPHS = EPWM1_PS_2;         // Defined PS value
    EPwm4Regs.TBCTR = 0x0000;                     // Clear counter

    //
    // Setup TBCLK
    //
    EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up down
    EPwm4Regs.TBCTL.bit.PHSEN = TB_ENABLE;         // Enable phase loading


    EALLOW;
    SyncSocRegs.SYNCSELECT.bit.EPWM4SYNCIN = 0;    // Get sync input from ePWM1
    EDIS;

    EPwm4Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_IN;
    EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
    EPwm4Regs.TBCTL.bit.CLKDIV = TB_DIV1;

    //
    // Setup compare
    //
    EPwm4Regs.CMPA.bit.CMPA = EPWM1_CMPA;

    //
    // Set actions
    //
    EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;           // CTR = CMPA (down direction)-> Set PWM4A to one
```

```
513     EPwm4Regs.AQCTLA.bit.CAD = AQ_SET;             // CTR = CMPA (down direction)-> Set PWM4A to zero
514
515
516     //
517     // Active High Complimentary PWMs - Setup Deadband
518     //
519     EPwm4Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
520     EPwm4Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;
521     EPwm4Regs.DBCTL.bit.IN_MODE = DBA_ALL;
522     EPwm4Regs.DBRED.bit.DBRED = EPWM1_RED;
523     EPwm4Regs.DBFED.bit.DBFED = EPWM1_FED;
524
525     //
526     // TripZone
527     //
528     EALLOW;
529     EPwm4Regs.TZCTL.bit.TZA = TZ_FORCE_LO;
530     EPwm4Regs.TZCTL.bit.TZB = TZ_FORCE_LO;
531
532     EDIS;
533 }
534
535 //
536 // End of file
537 //
538
```

**SCI.h:**

```
/*
 Author      : Kaushik Naresh Kumar
 Organization: Norwegian University of Science and Technology (NTNU)
 Email       : kaushik.n.kumar@ntnu.no

 Description:
Contains SCI related functions
*/


//
//Function prototypes
//

void configureSCI(void);
void initSCIAEchoback(void);
void transmitSCIAChar(uint16_t a);
void transmitSCIAMessage(unsigned char * msg);
void initSCIAFIFO(void);



void configureSCI(void)
{

    //
    // Configure the pins for the SCI-A port.
    // GPIO_SetupPinMux() - Sets the GPxMUX1/2 and GPyMUX1/2 register bits
    // GPIO_SetupPinOptions() - Sets the direction and configuration of GPIOs
    // These functions are found in the F28X7x_Gpio.c file.
    //
    GPIO_SetupPinMux(43, GPIO_MUX_CPU1, 15);              //SCIRX pin setup
    GPIO_SetupPinOptions(43, GPIO_INPUT, GPIO_PUSHPULL);  //SCIRX pin setup
    GPIO_SetupPinMux(42, GPIO_MUX_CPU1, 15);              //SCITX pin setup
    GPIO_SetupPinOptions(42, GPIO_OUTPUT, GPIO_ASYNC);    //SCITX pin setup
}


//
//  initSCIAEchoback - Initialize SCI-A for echoback
//
void initSCIAEchoback(void)
{
    //
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function
    //
    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 = 0x0003;
```

```
    SciaRegs.SCICTL2.bit.TXINTENA = 1;
    SciaRegs.SCICTL2.bit.RXBKINTENA = 1;

    //
    // SCIA at 9600 baud
    // @LSPCLK = 25 MHz (100 MHz SYSCLK) HBAUD = 0x01  and LBAUD = 0x44.
    //
    SciaRegs.SCIHBAUD.all = 0x0001;
    SciaRegs.SCILBAUD.all = 0x0044;

    SciaRegs.SCICTL1.all = 0x0023;          // Relinquish SCI from Reset
}

//
// transmitSCIAChar - Transmit a character from the SCI
//
void transmitSCIAChar(uint16_t a)
{
    while (SciaRegs.SCIFFTX.bit.TXFFST != 0)
    {

    }
    SciaRegs.SCITXBUF.all = a;
}

//
// transmitSCIAMessage - Transmit message via SCIA
//
void transmitSCIAMessage(unsigned char * msg)
{
    int i;
    i = 0;
    while(msg[i] != '\0')
    {
        transmitSCIAChar(msg[i]);
        i++;
    }
}

//
// initSCIAFIFO - Initialize the SCI FIFO
//
void initSCIAFIFO(void)
{
    SciaRegs.SCIFFTX.all = 0xE040;
    SciaRegs.SCIFFRX.all = 0x2044;
    SciaRegs.SCIFFCT.all = 0x0;
}

//
// End of file
//

```

**ADC.h:**

```
/*
 Author      : Kaushik Naresh Kumar
 Organization: Norwegian University of Science and Technology (NTNU)
 Email       : kaushik.n.kumar@ntnu.no

 Description:
Contains ADC related functions
*/

//Globals
Uint16 Vout_ADC;        //ADC converted value between 0-4095
Uint16 Iout_ADC;        //ADC converted value between 0-4095
float Vout;             //Analog value calculated as (Vout_ADC*3.3)/4096
float Iout;             //Analog value calculated as (Iout_ADC*3.3)/4096
float Vout1, Iout1;
//Uint16 DACBout = 2483;  //To generate 2V at DACoutB pin 70

//
// Function Prototypes
//
void ConfigureADC(void);
void SetupADCSoftware(void);
void StoreADCvalues(void);

void ConfigureADC(void)
{
    EALLOW;

    //
    //write configurations
    //
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
    AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

    //
    //Set pulse positions to late
    //
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;

    //
    //power up the ADCs
    //
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;

    //
    //delay for 1ms to allow ADC time to power up
    //
    DELAY_US(1000);

```

```
    EDIS;
}

void SetupADCSoftware(void)
{
    Uint16 acqps;

    //
    // Determine minimum acquisition window (in SYSCLKS) based on resolution
    //
    if(ADC_RESOLUTION_12BIT == AdcbRegs.ADCCTL2.bit.RESOLUTION)
    {
        acqps = 14; //75ns
    }
    else //resolution is 16-bit
    {
        acqps = 63; //320ns
    }

    //
    //Select the channels to convert and end of conversion flag
    //
    EALLOW;
    //ADCB
    AdcbRegs.ADCSOC1CTL.bit.CHSEL = 5;  //SOC1 will convert pin B5 -> Vout
    AdcbRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is acqps + 1 SYSCLK cycles
    AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = 5;//Trigger on ePWM1 SOCA
    AdcbRegs.ADCINTSEL1N2.bit.INT1E = 0;   //Disable INT1 flag

    //ADCC
    AdccRegs.ADCSOC1CTL.bit.CHSEL = 5;  //SOC1 will convert pin C5 -> Iout
    AdccRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is acqps + 1 SYSCLK cycles
    AdccRegs.ADCSOC1CTL.bit.TRIGSEL = 5;//Trigger on ePWM1 SOCA
    AdccRegs.ADCINTSEL1N2.bit.INT1E = 0;   //Disable INT1 flag
    EDIS;

//    Enable DAC outputs
//    EALLOW;
//    DacaRegs.DACOUTEN.bit.DACOUTEN = 1;//enable dacA output-->uses ADCINA0
//    DacaRegs.DACCTL.bit.LOADMODE = 0;//load on next sysclk
//    DacaRegs.DACCTL.bit.DACREFSEL = 1;//use ADC VREF as reference voltage
//    EDIS;

    EALLOW;
    DacbRegs.DACOUTEN.bit.DACOUTEN = 1;//enable dacB output-->uses ADCINA1
    DacbRegs.DACCTL.bit.LOADMODE = 0;//load on next sysclk
    DacbRegs.DACCTL.bit.DACREFSEL = 1;//use ADC VREF as reference voltage
    EDIS;

//    DacbRegs.DACVALS.bit.DACVALS = DACBout; //Set DACoutB i.e. pin 70 to 2V (2483*3.3)/4096

}
```

```
106
107 void StoreADCvalues(void)
108 {
109
110 Vout_ADC = AdcbResultRegs.ADCRESULT1;           //ADC result from pin 65 (ADCIN B5)
111 Vout = (Vout_ADC*3.0)/4096;                     //Equivalent analog value
112 Vout1 = (Vout/3.0)*400;
113
114 //DacbRegs.DACVALS.bit.DACVALS = Vout_ADC;
115
116 Iout_ADC = AdccResultRegs.ADCRESULT1;           //ADC result from pin 64 (ADCIN C5)
117 Iout = (Iout_ADC*3.0)/4096;                     //Equivalent analog value
118 Iout1 = ((Iout-1.67)/3.0)*30;
119
120 DacbRegs.DACVALS.bit.DACVALS = Iout_ADC;
121 }
122
123
124 //
125 // End of file
126 //
127
```