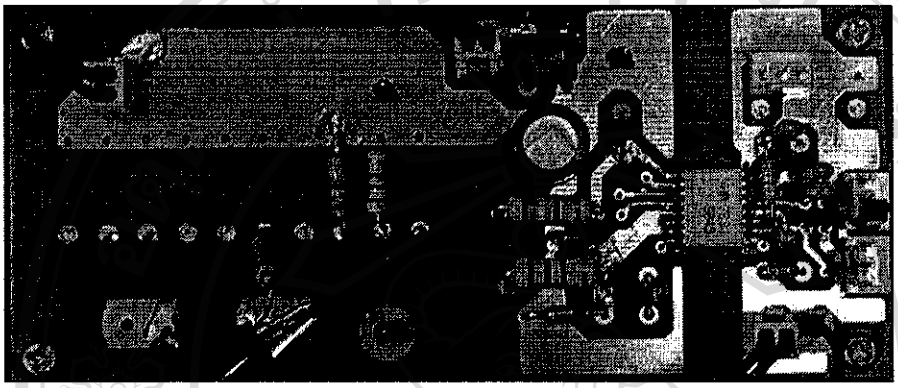
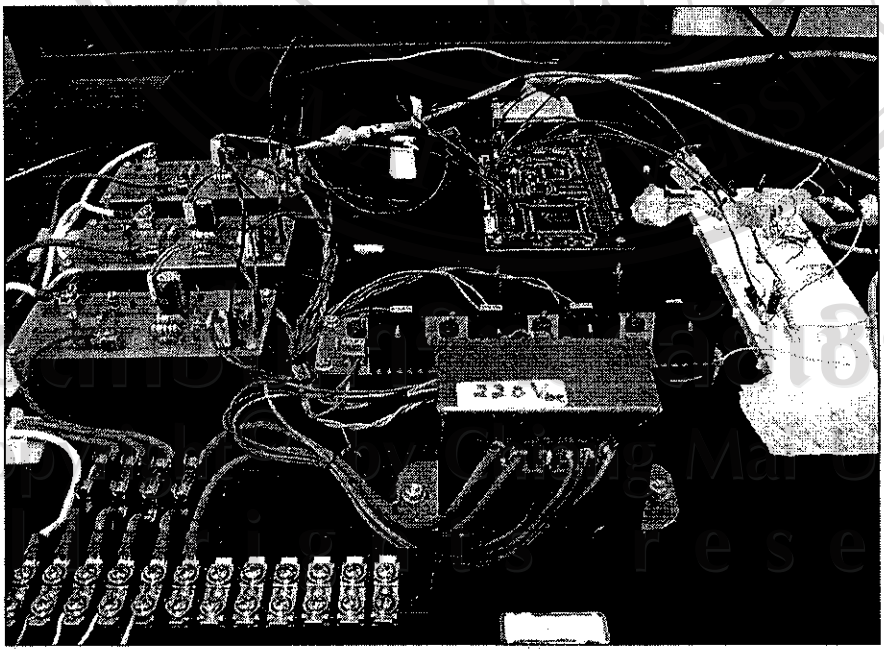
ภาคผนวก

ภาคผนวก ก

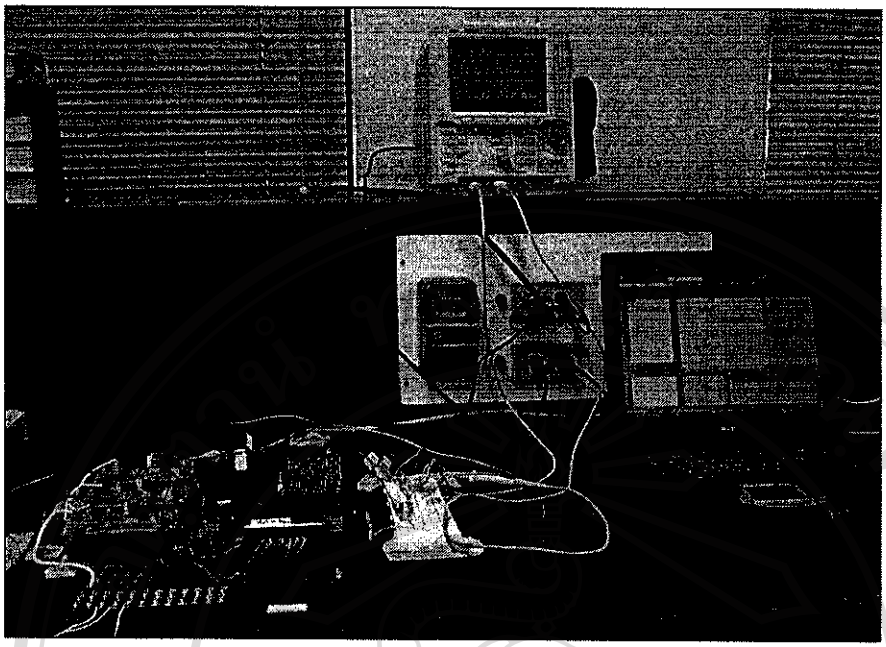
ภาพจริงของต้นแบบอุปกรณ์ตรวจจับแรงดันตกชั่วขณะ



รูปที่ ก.1 ภาพวงจรการใช้งานของ HCPL-788J



รูปที่ ก.2 ภาพของต้นแบบอุปกรณ์ตรวจจับแรงดันตกชั่วขณะชนิดสามเฟส



รูปที่ ก.3 ภาพชุดทดสอบในการปฏิบัติการบนบอร์ดตัวประมวลผลสัญญาณดิจิตอล

ภาคผนวก ข

ข้อมูลจำเพาะของอุปกรณ์

ข.1 ข้อมูลจำเพาะของบอร์ดสำเร็จรูป DSP Controllers รุ่น TMS320F2812 ของบริษัท Texas Instruments

- High-Performance Static CMOS Technoly
  - 150 MHz (6.67 ns Cycle Time)
  - Low-Power (1.8 V Core@135 MHz,1.9 V) Core@150 MHz,3.3 V I/O) Design
- JTAG Boundary Scan Support
- High-Performance 32 Bit CPU (TMS320C28x)
  - 16 x 16 and 32 x 32 MAC Operations
  - 16 x 16 Dual MAC
  - Harvard Bus Architecture
  - Atomic Operations
  - Fast Interrupt Response and Processing
  - Unified Memory Programming Model
  - Code-Efficient (in C/C++ and Assembly)
- On-Chip Memory
  - Flash Devices: Up to 128k x 16 Flash
  - ROM Devices: Up to 128k x 16 ROM
  - 1K x 16 OTP ROM
  - L0 and L1: 2 Blocks of 4K x 16 Each Single-Access RAM (SARAM)
  - H0: 1 Block of 8K x 16 SARAM
  - M0 and M1: 2 Blocks of 1K x 16 Each SARAM
- Boot ROM (4K x 16)
  - With Software oot Modes
  - Standard Math Tables
- 128 Bit Security Key/Lock
  - Protects Flash/ROM/OTP and L0/L1 SARAM
  - Prevents Firmware Reverse Engineering
- Motor Control Peripherals
  - Two Event Managers (EVA,EVB)
  - Compatible to 240xA Devices
- Serial Port Peripherals
  - Serial Peripheral Interface (SPI)
  - Two Serial Communications Interface (SCIs)
  - Enhanced Controller Area Network (eCAN)
  - Multichannel Buffered Serial Port (McBSP)
- 12 Bit ADC , 16 Channals
  - 2 x 8 Channel Input Multiplexer
  - Two Sample and Hold
  - Single/Simultaneuos Conversions
  - Fast Conversion Rate; 80 ns/12.5 MSPS

  Up to 56 General Purpose I/O (GPIO) Pins
- Advanced Emulation Features
  - Analysis and Breakpoint Funtions
  - Real-Time Debug via Hardware
- Development Tools Include
  - ANSI C/C++ Compiler/Assembler/Linker
  - Code Composer Studio
  - DSP/BIOS
  - JTAG Scan Controllers

- External Interface (2812)
  - Up to 1M Total Memory
  - Programmable Wait States
  - Programmable Read/Write Strobe Timing
  - Three Individual Chip Selects
- Clock and System Control
  - Dynamic PLL Ratio Changes Supported
  - On-Chip Dscillator
  - Watchdog Timer Module
- Three External Interrupts
- Peripheral Interrupt Expansion (PIE) Block That Supports 45 Peripheral Interrupts
- Three 32 Bit CPU Timers
- Low-Power Modes and Power Savings
  - IDLE, STANDBY, HALT Modes Supported
  - Disable Individual Peripheral Clocks
- Package Options
  - 179 Ball MicroStar BGA With External Memory Interface (GHH), (ZHH) (2812)
  - 176 Pin Low Profile Quad Flatpack (LQFP) With External Memory Interface (PGF) (2812)
- Temperature Options:
  - A: -40°C to 85°C (GHH, ZHH)
  - S/Q: -40°C to 125°C (GHH, ZHH)

ข.2 ข้อมูลจำเพาะของไอซีเบอร์ HCPL-788J

Isolation Amplifier with Short Circuit and Overload Detection

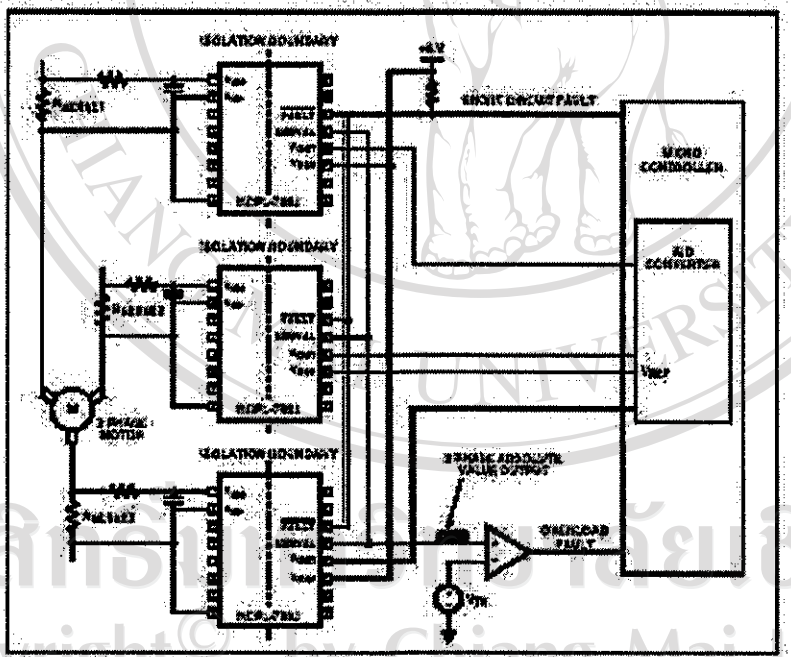
Technical Data

HCPL-788J

Features

- Output Voltage Directly Compatible with A/D Converters (0 V to V_{REF})
- Fast (3 µs) Short Circuit Detection with Transient Fault Rejection
- Absolute Value Signal Output for Overload Detection
- 1 µV/°C Offset Change vs. Temperature
- SO-16 Package
- -40°C to +85°C Operating Temperature Range
- 25 kV/µs Isolation Transient Immunity
- Regulatory Approvals: UL, CSA, IEC/EN/DIN EN 60747-5-2 (891 Vpeak Working Voltage)

Low Cost Three Phase Current Sensing with Short Circuit and Overload Detection



Agilent's Isolation Amplifier with Short Circuit and Overload Detection makes *motor phase current sensing compact, affordable and easy-to-implement* while satisfying worldwide safety and regulatory requirements.

Description

The HCPL-788J isolation amplifier is designed for current sensing in electronic motor drives. In a typical implementation, motor currents flow through an external resistor and the resulting analog voltage drop is sensed by the HCPL-788J. A larger analog output voltage is created on the other side of the HCPL-788J's optical isolation barrier. The output voltage is proportional to the motor current and can be connected directly to a single-supply A/D converter. A digital over-range output ($\overline{FAULT}$) and an analog rectified output (ABSVAL) are also provided. The wire OR-able over-range output ($\overline{FAULT}$) is useful for quick detection of short circuit conditions on any of the motor phases. The wire-OR-able rectified output (ABSVAL), simplifies measurement of motor load since it performs polyphase rectification. Since the common-mode voltage swings several hundred volts in tens of nanoseconds in modern electronic motor drives, the HCPL-788J was designed to ignore very high common-mode transient slew rates (10 kV/μs).

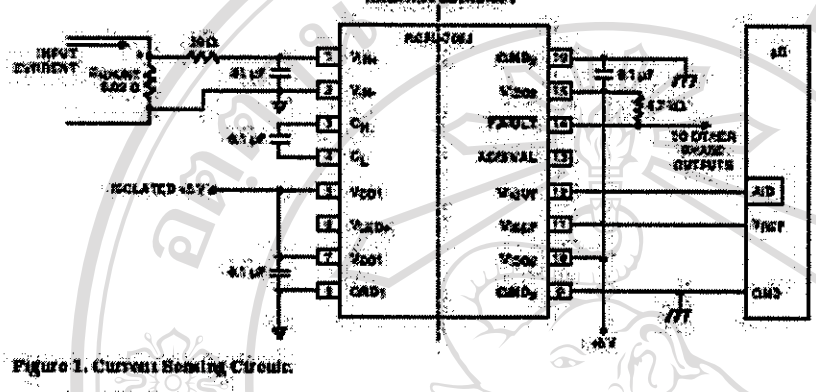


Figure 1. Current Sensing Circuit.

Pin Descriptions

Symbol	Description
V_{IN+}	Positive input voltage (± 200 mV recommended).
V_{IN-}	Negative input voltage (normally connected to GND_1).
C_H	Internal Bias Node. Connections to or between C_H and C_L other than the required 0.1 μF capacitor shown, are not recommended.
C_L	
V_{DD1}	Supply voltage input (4.5 V to 5.5 V).
V_{LED+}	LED anode. This pin must be left unconnected for guaranteed data sheet performance. (For optical coupling testing only.)
V_{DD1}	Supply voltage input (4.5 V to 5.5 V).
GND_1	Ground input.

Symbol	Description
GND_2	Ground input.
V_{DD2}	Supply voltage input (4.5 V to 5.5 V).
$\overline{FAULT}$	Short circuit fault output. $\overline{FAULT}$ changes from a high to low output voltage within 6 μs after V_{IN} exceeds the $\overline{FAULT}$ Detection Threshold. $\overline{FAULT}$ is an open drain output which allows outputs from all the HCPL-788Js in a circuit to be connected together ("wired-OR") forming a single fault signal for interfacing directly to the micro-controller.
ABSVAL	Absolute value of V_{OUT} output. ABSVAL is 0 V when V_{IN}=0 and increases toward V_{REF} as V_{IN} approaches +256 mV or -256 mV. ABSVAL is "wired-OR" able and is used for detecting overloads.
V_{OUT}	Voltage output. Swings from 0 to V_{REF}. The nominal gain is V_{REF}/504 mV.
V_{REF}	Reference voltage input (4.0 V to V_{DD2}). This voltage establishes the full scale output ranges and gains of V_{OUT} and ABSVAL.
V_{DD2}	Supply voltage input (4.5 V to 5.5 V).

ภาคผนวก ค

รายละเอียดซอฟต์แวร์ของระบบ

โปรแกรม Voltage Sag Detection (Main Program)

```
//=============================================================================
// File Name:  VSD.C
//=============================================================================
// Include header files used in the main function

#include "target.h"

#if (DSP_TARGET==F2812)
#include "DSP281x_Device.h"
#endif

#include "IQmathLib.h"
#include "vsd.h"
#include "parameter.h"
#include "build.h"
#include <math.h>

// Prototype statements for functions found within this file.
interrupt void MainISR(void);
#if (DSP_TARGET==F2812)
interrupt void QepISR(void);
#endif

// Global variables used in this system
float32 T = 0.001/ISR_FREQUENCY;
Uint16 IsrTicker = 0;
Uint16 BackTicker = 0;

int16 PwmDacCh1=0;
int16 PwmDacCh2=0;
int16 PwmDacCh3=0;

int16 DlogCh1 = 0;
int16 DlogCh2 = 0;
int16 DlogCh3 = 0;
int16 DlogCh4 = 0;

volatile Uint16 EnableFlag = FALSE;
Uint16 LockRotorFlag = FALSE;

Uint16 SpeedLoopPrescaler = 10;      // Speed loop prescaler
Uint16 SpeedLoopCount = 1;           // Speed loop counter

// Instance a few transform objects
CLARKE clarke1 = CLARKE_DEFAULTS;
PARK park1 = PARK_DEFAULTS;
IPARK ipark1 = IPARK_DEFAULTS;
```

```
PWMGEN pwm1 = PWMGEN_DEFAULTS;
PWMDAC pwmdac1 = PWMDAC_DEFAULTS;
RMPCNTL rc1 = RMPCNTL_DEFAULTS;
RAMPGEN rg1 = RAMPGEN_DEFAULTS;
ILEG2DCBUSMEAS ilg2_vdc1 = ILEG2DCBUSMEAS_DEFAULTS;
DLOG_4CH dlog = DLOG_4CH_DEFAULTS;
void Gpio_select(void);
void main(void)
{

// ***************************************
// Initialization code for DSP_TARGET = F2812
// ***************************************
#if (DSP_TARGET==F2812)

InitSysCtrl();
    EALLOW;
    SysCtrlRegs.HISPCP.all = 0x0000;
    EDIS;

// Disable and clear all CPU interrupts:
      DINT;
      IER = 0x0000;
      IFR = 0x0000;

// Initialize Pie Control Registers To Default State:
// This function is found in the DSP281x_PieCtrl.c file.
      InitPieCtrl();
      InitPieVectTable();

// User specific functions, Reassign vectors (optional), Enable
Interrupts:

// Initialize EVA Timer 1:
// Setup Timer 1 Registers (EV A)
   EvaRegs.GPTCONA.all = 0;
   // Waiting for enable flag set
   while (EnableFlag==FALSE)
    {
      BackTicker++;
    }
// Enable Underflow interrupt bits for GP timer 1
    EvaRegs.EVAIMRA.bit.T1UFINT = 1;
    EvaRegs.EVAIFRA.bit.T1UFINT = 1;
// Enable CAP3 interrupt bits for GP timer 2
    EvaRegs.EVAIMRC.bit.CAP3INT = 1;
    EvaRegs.EVAIFRC.bit.CAP3INT = 1;
      EALLOW;
      PieVectTable.T1UFINT = &MainISR;
      PieVectTable.CAPINT3 = &QepISR;
      EDIS;
// Enable PIE group 2 interrupt 6 for T1UFINT
    PieCtrlRegs.PIEIER2.all = M_INT6;
// Enable PIE group 3 interrupt 7 for CAP3INT
    PieCtrlRegs.PIEIER3.all = M_INT7;
// Enable CPU INT2 for T1UFINT and INT3 for CAP3INT:
      IER |= (M_INT2 | M_INT3);

#endif
```

```
// Initialize PWM module
    pwm1.PeriodMax = SYSTEM_FREQUENCY*1000000*T/2;  // Perscaler X1
(T1), ISR period = T x 1
      pwm1.init(&pwm1);

// Initialize PWMDAC module
    pwmdac1.PeriodMax = (SYSTEM_FREQUENCY*200/(30*2))*5;   // PWMDAC
Frequency = 30 kHz
    pwmdac1.PwmDacInPointer0 = &PwmDacCh1;
    pwmdac1.PwmDacInPointer1 = &PwmDacCh2;
    pwmdac1.PwmDacInPointer2 = &PwmDacCh3;
      pwmdac1.init(&pwmdac1);

// Initialize DATALOG module
    dlog.iptr1 = &DlogCh1;
    dlog.iptr2 = &DlogCh2;
    dlog.iptr3 = &DlogCh3;
    dlog.iptr4 = &DlogCh4;
    dlog.trig_value = 0x1;
    dlog.size = 0x400;
    dlog.prescalar = 1;
    dlog.init(&dlog);

// Initialize QEP module
    qep1.LineEncoder = 2000;
    qep1.MechScaler = _IQ30(0.25/qep1.LineEncoder);
    qep1.PolePairs = P/2;
    qep1.CalibratedAngle = -1250;
    qep1.init(&qep1);

// Initialize the Speed module for QEP based speed calculation
    speed1.K1 = _IQ21(1/(BASE_FREQ*T));
    speed1.K2 = _IQ(1/(1+T*2*PI*30));  // Low-pass cut-off frequency
    speed1.K3 = _IQ(1)-speed1.K2;
    speed1.BaseRpm = 120*(BASE_FREQ/P);

// Initialize enable drive module (FOR DMC1500 ONLY)
    drv1.init(&drv1);

// Initialize ADC module
    ilg2_vdc1.ChSelect = 0x0610;
    ilg2_vdc1.init(&ilg2_vdc1);
// Initialize the RAMPGEN module
    rg1.StepAngleMax = _IQ(BASE_FREQ*T);
// Initialize the RAMPGEN module
    rc1.RampDelayMax = 5;
// Initialize the PID_REG3 module for Id
      pid1_id.Kp = _IQ(0.1);
      pid1_id.Ki = _IQ(T/0.02);
      pid1_id.Kd = _IQ(0/T);
      pid1_id.Kc = _IQ(0.5);
    pid1_id.OutMax = _IQ(0.30);
    pid1_id.OutMin = _IQ(-0.30);
// Initialize the PID_REG3 module for Iq
      pid1_iq.Kp = _IQ(0.1);
      pid1_iq.Ki = _IQ(T/0.02);
      pid1_iq.Kd = _IQ(0/T);
      pid1_iq.Kc = _IQ(0.5);
    pid1_iq.OutMax = _IQ(0.95);
```

```
    pid1_iq.OutMin = _IQ(-0.95);
// Initialize the PID_REG3 module for speed control
    pid1_spd.Kp = _IQ(1);
      pid1_spd.Ki = _IQ(T*SpeedLoopPrescaler/0.3);
      pid1_spd.Kd = _IQ(0/(T*SpeedLoopPrescaler));
      pid1_spd.Kc = _IQ(0.2);
    pid1_spd.OutMax = _IQ(1);
    pid1_spd.OutMin = _IQ(-1);
// Initialize the PID_REG3 module for position control
    pid1_pos.Kp = _IQ(28.2);
      pid1_pos.Ki = _IQ(0);
      pid1_pos.Kd = _IQ(0);
      pid1_pos.Kc = _IQ(0);
    pid1_pos.OutMax = _IQ(1);
    pid1_pos.OutMin = _IQ(-1);
// Enable global Interrupts and higher priority real-time debug
events:
      EINT;
      ERTM;
// IDLE loop. Just sit and loop forever:
      for(;;) BackTicker++;

}
  interrupt void MainISR(void)
{

// Verifying the ISR
    IsrTicker++;

// ***************** LEVEL1 *****************
#if (BUILDLEVEL==LEVEL1)

// ---------------------------------------------------------------
//    Connect inputs of the RMP module and call the Ramp control
// ---------------------------------------------------------------
    rc1.TargetValue = _IQ(SpeedRef);
    rc1.calc(&rc1);

// ---------------------------------------------------------------
//    Connect inputs of the RAMP GEN module and call the Ramp
generator
// ---------------------------------------------------------------
    rg1.Freq = rc1.SetpointValue;
    rg1.calc(&rg1);

// ---------------------------------------------------------------
//    Connect inputs of the INV_PARK module and call the inverse
park transformation
// ---------------------------------------------------------------
    ipark1.Ds = _IQ(VdTesting);
    ipark1.Qs = _IQ(VqTesting);
    ipark1.Angle = rg1.Out;
    ipark1.calc(&ipark1);

// ---------------------------------------------------------------
//    Connect inputs of the SVGEN_DQ module and call the space-
vector gen.
// ---------------------------------------------------------------
      svgen_dq1.Ualpha = ipark1.Alpha;
```

```
      svgen_dq1.Ubeta = ipark1.Beta;
      svgen_dq1.calc(&svgen_dq1);

// ---------------------------------------------------------------------
//    Connect inputs of the PWM_DRV module and call the PWM signal
generation
// ---------------------------------------------------------------------
    pwm1.MfuncC1 = (int16)_IQtoIQ15(svgen_dq1.Ta);
    pwm1.MfuncC2 = (int16)_IQtoIQ15(svgen_dq1.Tb);
    pwm1.MfuncC3 = (int16)_IQtoIQ15(svgen_dq1.Tc);
      pwm1.update(&pwm1);
// ---------------------------------------------------------------------
//    Connect inputs of the PWMDAC module
// ---------------------------------------------------------------------
    PwmDacCh1 = (int16)_IQtoIQ15(svgen_dq1.Ta);
    PwmDacCh2 = (int16)_IQtoIQ15(svgen_dq1.Tb);
    PwmDacCh3 = (int16)_IQtoIQ15(svgen_dq1.Tc);

// ---------------------------------------------------------------------
//    Connect inputs of the DATALOG module
// ---------------------------------------------------------------------
    DlogCh1 = (int16)_IQtoIQ15(svgen_dq1.Ta);
    DlogCh2 = (int16)_IQtoIQ15(svgen_dq1.Tb);
    DlogCh3 = (int16)_IQtoIQ15(svgen_dq1.Tc);
    DlogCh4 = (int16)_IQtoIQ15(svgen_dq1.Ta-svgen_dq1.Tb);

// ---------------------------------------------------------------------
//    Connect inputs of the EN_DRV module and call the
enable/disable PWM signal
// ---------------------------------------------------------------------
    drv1.EnableFlag = EnableFlag;
    drv1.update(&drv1);

#endif // (BUILDLEVEL==LEVEL1)

// ***************** LEVEL2 *****************
#if (BUILDLEVEL==LEVEL2)

// ---------------------------------------------------------------------
//    Connect inputs of the RMP module and call the Ramp control
// ---------------------------------------------------------------------
    rc1.TargetValue = _IQ(SpeedRef);
    rc1.calc(&rc1);

// ---------------------------------------------------------------------
//    Connect inputs of the RAMP GEN module and call the Ramp
generator
// ---------------------------------------------------------------------
    rg1.Freq = rc1.SetpointValue;
    rg1.calc(&rg1);

// ---------------------------------------------------------------------
//    Call the ILEG2_VDC read function.
// ---------------------------------------------------------------------
    ilg2_vdc1.read(&ilg2_vdc1);

// ---------------------------------------------------------------------
//    Connect inputs of the CLARKE module and call the clarke
transformation
```

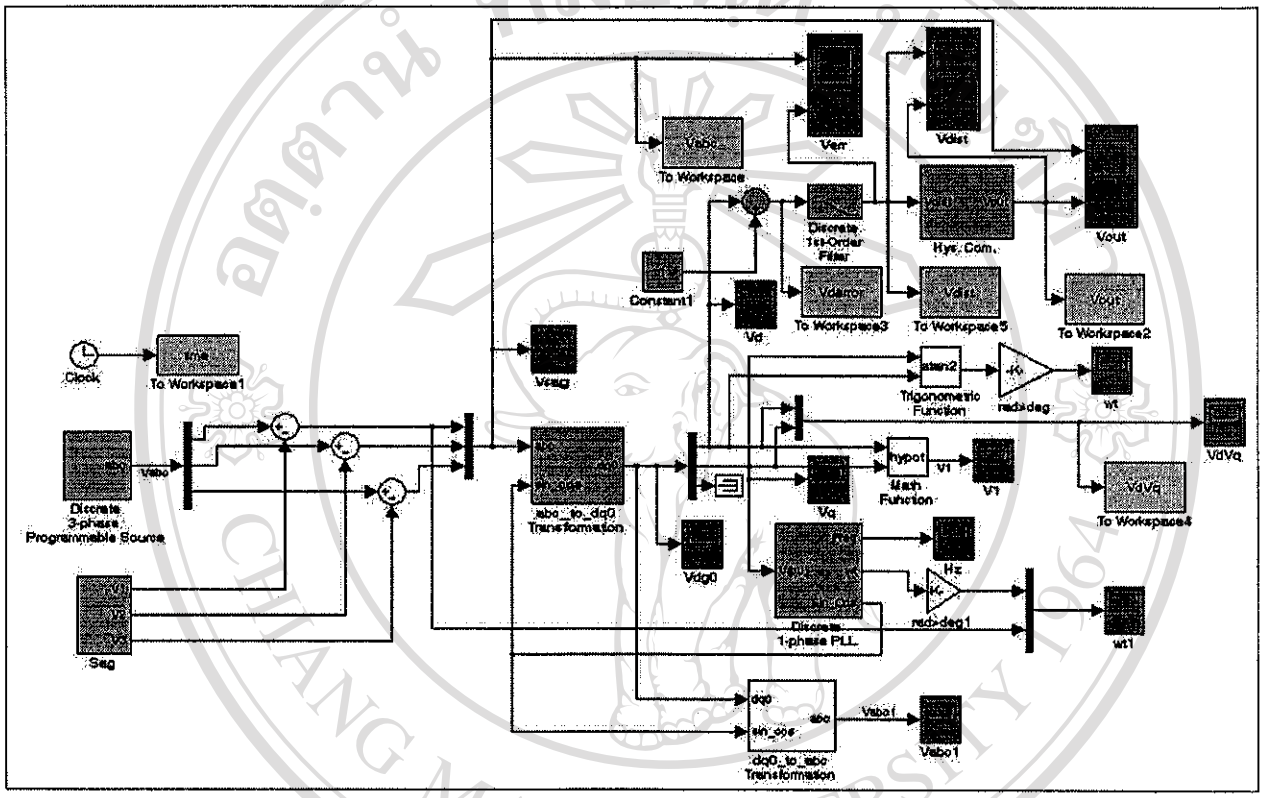
```
// ---------------------------------------------------------------
      clarke1.As = _IQ15toIQ((int32)ilg2_vdc1.ImeasA);
      clarke1.Bs = _IQ15toIQ((int32)ilg2_vdc1.ImeasB);
      clarke1.Cs = _IQ15toIQ((int32)ilg2_vdc1.ImeasC);
      clarke1.calc(&clarke1);

// ---------------------------------------------------------------
//    Connect inputs of the PARK module and call the park
transformation
// ---------------------------------------------------------------
    park1.Alpha = clarke1.Alpha;
    park1.Beta = clarke1.Beta;
    park1.Angle = clarke1.Arctan;
    park1.calc(&park1);
    Gpio_select();          // Setup the GPIO Multiplex Registers
    if ((park1.Dist <= park1.Ref)&&(_IQabs(park1.Dist-
park1.Ref)>park1.band))
      GpioDataRegs.GPADAT.bit.GPIOA0=0;
    else
      GpioDataRegs.GPADAT.bit.GPIOA0=1;

// ---------------------------------------------------------------
//    Connect inputs of the INV_PARK module and call the inverse
park transformation
// ---------------------------------------------------------------
    ipark1.Ds = _IQ(VdTesting);
    ipark1.Qs = _IQ(VqTesting);
    ipark1.Angle = rg1.Out;
    ipark1.calc(&ipark1);

// ---------------------------------------------------------------
//    Connect inputs of the SVGEN_DQ module and call the space-
vector gen.
// ---------------------------------------------------------------
      svgen_dq1.Ualpha = ipark1.Alpha;
      svgen_dq1.Ubeta = ipark1.Beta;
      svgen_dq1.calc(&svgen_dq1);

// ---------------------------------------------------------------
//    Connect inputs of the PWM_DRV module and call the PWM signal
generation

// ---------------------------------------------------------------
    pwm1.MfuncC1 = (int16)_IQtoIQ15(svgen_dq1.Ta);
    pwm1.MfuncC2 = (int16)_IQtoIQ15(svgen_dq1.Tb);
    pwm1.MfuncC3 = (int16)_IQtoIQ15(svgen_dq1.Tc);
      pwm1.update(&pwm1);

// ---------------------------------------------------------------
//    Connect inputs of the PWMDAC module
// ---------------------------------------------------------------
      PwmDacCh1 = (int16)_IQtoIQ15(park1.Dist);
      PwmDacCh2 = (int16)_IQtoIQ15(park1.Qs);
// ---------------------------------------------------------------
//    Connect inputs of the DATALOG module
// ---------------------------------------------------------------
    DlogCh1 = (int16)_IQtoIQ15(park1.Ds);
    DlogCh2 = (int16)_IQtoIQ15(park1.Qs);
    DlogCh3 = (int16)_IQtoIQ15(clarke1.Alpha);
```

```
    DlogCh4 = (int16)_IQtoIQ15(clarke1.Beta);

// ---------------------------------------------------------------
//    Connect inputs of the EN_DRV module and call the
enable/disable PWM signal
// ---------------------------------------------------------------
    drv1.EnableFlag = EnableFlag;
    drv1.update(&drv1);

#endif // (BUILDLEVEL==LEVEL2)

// ---------------------------------------------------------------
//    Call the PWMDAC update function.
// ---------------------------------------------------------------
      pwmdac1.update(&pwmdac1);

// ---------------------------------------------------------------
//    Call the DATALOG update function.
// ---------------------------------------------------------------
    dlog.update(&dlog);

#if (DSP_TARGET==F2812)
// Enable more interrupts from this timer
      EvaRegs.EVAIMRA.bit.T1UFINT = 1;
// Note: To be safe, use a mask value to write to the entire
      EvaRegs.EVAIFRA.all = BIT9;

// Acknowledge interrupt to recieve more interrupts from PIE group 2
      PieCtrlRegs.PIEACK.all |= PIEACK_GROUP2;
#endif

}
#if (DSP_TARGET==F2812)
interrupt void QepISR(void)
{
// ---------------------------------------------------------------
//    Call the QEP_DRV isr function.
// ---------------------------------------------------------------
   qep1.isr(&qep1);
// Enable more interrupts from this timer
      EvaRegs.EVAIMRC.bit.CAP3INT = 1;

// Note: To be safe, use a mask value to write to the entire
      EvaRegs.EVAIFRC.all = BIT2;
// Acknowledge interrupt to recieve more interrupts from PIE group 3
      PieCtrlRegs.PIEACK.all |= PIEACK_GROUP3;
}
#endif
 void Gpio_select(void)
{
    EALLOW;
    GpioMuxRegs.GPAMUX.all = 0x0;   // all GPIO port Pin's to I/O
    GpioMuxRegs.GPBMUX.all = 0x0;
    GpioMuxRegs.GPDMUX.all = 0x0;
    GpioMuxRegs.GPFMUX.all = 0x0;
    GpioMuxRegs.GPEMUX.all = 0x0;
    GpioMuxRegs.GPGMUX.all = 0x0;
    GpioMuxRegs.GPADIR.all = 0x00FF;
    GpioMuxRegs.GPDDIR.all = 0x0;    // GPIO PORT  as input
```

```
    GpioMuxRegs.GPEDIR.all = 0x0;    // GPIO PORT  as input
    GpioMuxRegs.GPFDIR.all = 0x0;    // GPIO PORT  as input
    GpioMuxRegs.GPGDIR.all = 0x0;    // GPIO PORT  as input
    GpioMuxRegs.GPAQUAL.all = 0x0;
    GpioMuxRegs.GPDQUAL.all = 0x0;
    GpioMuxRegs.GPEQUAL.all = 0x0;
    EDIS;
}
```

ภาคผนวก ง

แผนภาพบล็อกในการจำลองการทำงานในโปรแกรม MATLAB/SIMULINK



ประวัติผู้เขียน

ชื่อ	นายพิเชษฐ์ ทานิล
วัน เดือน ปี เกิด	20 ตุลาคม 2520
ประวัติการศึกษา	สำเร็จการศึกษาระดับมัธยมศึกษาตอนต้นที่ โรงเรียนพะเยาพิทยาคม ปีการศึกษา 2535 สำเร็จการศึกษาระดับประกาศนียบัตรวิชาชีพ คณะวิชาไฟฟ้า แผนกช่างไฟฟ้า วิทยาลัยเทคนิคพะเยา ปีการศึกษา 2538 สำเร็จการศึกษาระดับปริญญาตรีครุศาสตรอุตสาหกรรมบัณฑิต (วิศวกรรมไฟฟ้า) สถาบันเทคโนโลยีราชมงคล วิทยาเขตภาคพายัพ ปีการศึกษา 2543