S12ZVC256 BASIC TRAINING
Session Objectives

• By the end of this session, you will be able to:

  – Know motor basic, advantage and disadvantage of the different types of motors.

  – Identify the modules integrated in the S12ZVM for BLDC and PMSM motor drive applications

  – Know the MTRCKTSBNZVM128 motor control kit based on the MagniV S12ZVM microcontroller

  – Create an application using CodeWarrior IDE to control the speed of a BLDC motor based on a potentiometer reading

  – Use FreeMaster for non-intrusive debugging
Agenda

• Motor overview
• S12ZVM microcontroller introduction
• Hardware & Software Development Tools
  • Lab #1: Starting a CodeWarrior project for S12ZVM
  • Lab #2: The FreeMaster interface and the ADC
  • A BLDC motor control application
  • Lab #3: Advanced Math and Motor Control Library
  • Lab #4: The PMF, PTU, and GDU
  • Lab #5: Driving a BLDC motor based on Hall sensors
• Q & A
BLDC MOTOR BASICS

• OVERVIEW
Electric Motor Classification

**ELECTRIC MOTORS**

**AC**
- Asynchronous
  - Induction
    - Split phase
    - Shaded pole
    - Capac. run
    - Capac. start
    - Squirrel Cage
    - Wound Rotor
  - Sinusoidal
    - Permanent Magnet
  - Brushless
    - Surface PM
    - Interior PM
  - Wound Field (Excitation winding)

**Synchronous**
- Permanent Magnet

**Variable Reluctance**
- SR
- Stepper
- Separately excited
  - Wound
  - Permanent Magnet
- Series field
- Shunt field (parallel)
- Series universal

**DC**
Electric Motor Classification

**ELECTRIC MOTORS**

- **AC**
  - Asynchronous
    - Induction
      - Split phase
      - Shaded pole
      - Capac. run
      - Capac. start
      - Squirrel Cage
      - Wound Rotor
    - Sinusoidal
      - Permanent Magnet
      - Surface PM
      - Interior PM
    - Brushless
      - Wound Field (Excitation winding)
- **Synchronous**
  - Permanent Magnet
- **Variable Reluctance**
  - Reluctance
  - SR
  - Stepper
    - Separately excited
      - Wound
      - Permanent Magnet
    - Series field
    - Shunt field (parallel)
    - Series universal
S12ZVM Motor targets

• **DC Motors**
  - Two or more permanent magnets in stator
  - Rotor windings connected to mechanical commutator

• **BLDC Motors**
  - PM in rotor, 3-phase conductors in stator
  - Trapezoidal back-EMF

• **Permanent Magnet Synchronous Motors**
  - Similar to BLDC in construction
  - Sinusoidal back-EMF
PM Machines – Trapezoidal vs. Sinusoidal

• The characteristic “Trapezoidal” or “Sinusoidal” is linked with the shape of the Back-EMF of the Permanent Magnet motor.
  - “Sinusoidal” means Synchronous (PMSM) motors
  - “Trapezoidal” means Brushless DC (BLDC) motors

• BLDC motor control (6-step control)
  - Only 2 of the 3 stator phases are excited at any time
  - 1 unexcited phase used as sensor (sensorless control)

• Synchronous motor (Field-oriented control)
  - All 3 phases are persistently excited at any time
Six Step Commutation

Six Step BLDC Motor Control
- Voltage applied on two phases only
- It creates 6 flux vectors
- Phases are powered based on rotor position
- The process is called Commutation
BLDC Six-Step Commutation Principle

- Stator field is generated between 60° to 120° relative to the rotor field to get maximal torque and energy efficiency.
Magnetic Field Distribution in PM Motors

Sinusoidal” or “Sinewave” machine means Synchronous (PMSM)
The characteristic “Trapezoidal” or “Sinusoidal” is linked with the shape of Back EMF of PM motor.

**Magnetic Flux Density**
Shape of the flux density depends on the magnetization of the PM (radial, parallel) and their displacement

**Magnetic Flux Linkage**

**Phase Back EMFs**
Back EMF depends on the shape of the linkage flux.
Basic PM Motor Features and Comparison

**Brushless D.C. Motor**
- BLDC motor
  - 3-phase machine with PM on the rotor
  - Rotor position sensing required for rotor flux position
  - High torque per frame size
  - Synchronous operation
  - Good high speed performance (no brush losses)
  - **High torque ripple**

**Permanent Magnet Synchronous Motor**
- PMSM motor
  - 3-phase machine with PM on the rotor
  - Rotor position sensing required for rotor flux position
  - High torque per frame size
  - Synchronous operation
  - Good high speed performance (no brush losses)
  - **Low torque ripple**
Torque Ripple of PM Motors

**Brushless D.C. Motor**
- Trapezoidal Back-EMF
- Six-Step commutation control
- 2 of the 3 stator phases are excited at any time

**Permanent magnet synchronous motor**
- Sinusoidal Back-EMF (ideal case)
- Field Oriented Control
- All 3 phases persistently excited at any time

![Graphs showing phase currents and back EMF for both motor types](image)

**Resulting torque**
- Torque components of each motor phase
PM Motors in Automotive - Example

**Brushless D.C. Motor**

**Fuel/liquid pumps with BLDC**

*Application requirements:*
- High speed operation
- Simple sensorless control
- Low cost control solution
- Higher efficiency than DC motor

**Permanent magnet synchronous motor**

**Power steering with PMSM**

*Application requirements:*
- High speed operation
- Smooth torque operation
- Suppressed vibration and acoustic noise
MAGNIV S12ZVM
Device Overview
S12ZVM - Single Chip Solution for Motor Control

Discrete Solution

- VREG (8pin)
- MCU or DSC (48pin)
- Gate Driver (48pin)
- LIN phy (8pin)
- Op-amps

Optimize system efficiency

Optimize system cost

S12ZVM Solution:
- ~ 50 fewer solder joints
- ~ 2 to 3 cm2 PCB space
## Operating Voltage Ranges

### Without Boost

<table>
<thead>
<tr>
<th>Vsup</th>
<th>MCU</th>
<th>GDU</th>
</tr>
</thead>
<tbody>
<tr>
<td>20V…40V</td>
<td>Full</td>
<td>Disabled</td>
</tr>
<tr>
<td>7V…20V</td>
<td>Full</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vgs &gt; Vsup - 2*Vbe (5V min)</td>
</tr>
<tr>
<td>6V .. 7V</td>
<td>Full</td>
<td>Disabled</td>
</tr>
<tr>
<td>3.5V .. 6V</td>
<td>Full</td>
<td>Iddx = 25mA max if no external PNP</td>
</tr>
<tr>
<td>&lt;3.5V</td>
<td>Reset</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

### With Boost

<table>
<thead>
<tr>
<th>Vsup</th>
<th>MCU</th>
<th>GDU</th>
</tr>
</thead>
<tbody>
<tr>
<td>20V... 40V</td>
<td>Full</td>
<td>Disabled</td>
</tr>
<tr>
<td>9.5V...20V</td>
<td>Full</td>
<td>Boost OFF for Vsup &gt; 11V Vgs = 9.6V</td>
</tr>
<tr>
<td>6V...9.5V</td>
<td>Full</td>
<td>Boost ON Vgs &gt;9V</td>
</tr>
<tr>
<td>3.5V .. 6V</td>
<td>Full</td>
<td>Boost ON Vgs &gt;9V</td>
</tr>
<tr>
<td>&lt;3.5V</td>
<td>Reset</td>
<td>Disabled</td>
</tr>
</tbody>
</table>
S12ZVML Application Schematic
DEVELOPMENT TOOLS

Hardware + Software
FreeMASTER – Run Time Debugging Tool

- User-friendly tool for real-time debug monitor and data visualization
  - Completely non-intrusive monitoring of variables on a running system
  - Display multiple variables changing over time on an oscilloscope-like display, or view the data in text form
  - Communicates with an on-target driver via USB, BDM, CAN, UART

http://www.freescale.com/freemaster
Auto Math and Motor Control Library Contents

**MLIB**
- Absolute Value, Negative Value
  - MLIB_Abs, MLIB_Abs_sat
- Add/Subtract Functions
  - MLIB_Add, MLIB_AddSat
- Multiply/Divide/Add-multiply Functions
  - MLIB_Mul, MLIB_MulSat
- Shifting
  - MLIB_ShL, MLIB_ShLSat
- Normalisation, Round Functions
  - MLIB_Norm, MLIB_Round
- Conversion Functions
  - MLIB_ConvertPU, MLIB_Convert

**GFLIB**
- Trigonometric Functions
  - GFLIB_Sin, GFLIB_Cos, GFLIB_Tan
- Limitation Functions
  - GFLIB_Limit, GFLIB_VectorLimit, GFLIB_LowerLimit, GFLIB_UpperLimit
- PI Controller Functions
  - GFLIB_ControllerP, GFLIB_ControllerPip
- Interpolation
  - GFLIB_Lut1D, GFLIB_Lut2D
- Hysteresis Function
  - GFLIB_Hyst
- Signal Integration Function
  - GFLIB_IntegratorTR
- Sign Function
  - GFLIB_Sgn
- Signal Ramp Function
  - GFLIB_Ramp
- Square Root Function
  - GFLIB_Sqrt

**GDFLIB**
- Finite Impulse Filter
  - GDFLIB_FilterIR, GDFLIB_FilterR1
- Limitation Functions
  - GDFLIB_Limit, GDFLIB_VectorLimit, GDFLIB_LowerLimit, GDFLIB_UpperLimit
- PI Controller Functions
  - GDFLIB_ControllerP, GDFLIB_ControllerPip
- Interpolation
  - GDFLIB_Lut1D, GFLIB_Lut2D
- Hysteresis Function
  - GDFLIB_Hyst
- Signal Integration Function
  - GDFLIB_IntegratorTR
- Sign Function
  - GDFLIB_Sgn
- Signal Ramp Function
  - GDFLIB_Ramp
- Square Root Function
  - GDFLIB_Sqrt

**GMCLIB**
- Clark Transformation
  - GMCLIB_Cla
- Park Transformation
  - GMCLIB_Park
- Duty Cycle Calculation
  - GMCLIB_Sv
- Elimination of DC Ripples
  - GMCLIB_ElimDC
- Decoupling of PMSM Motors
  - GMCLIB_DecouplingPMSM
- Angle Tracking Observer
  - GMCLIB_Clark

**ACLIB/AMCLIB**
- Angle Tracking Observer
  - GMCLIB_Clark
- Tracking Observer
  - GMCLIB_Park
- PMSM BEMF Observer in Alpha/Beta
  - GMCLIB_Park
- PMSM BEMF Observer in D/Q
  - GMCLIB_Sv
- Content To Be Defined

**Delivery Content**
- Matlab/Simulink Bit Accurate Models
- User Manuals
- Header files
- Compiled Library File
- License File (to be accepted at install time)
HANDS-ON SESSION #1

STARTING AN S12ZVM PROJECT IN CODEWARRIOR
Hands-on session #1 – Objectives

• In this session you will:
  
  – Import an existing project into CodeWarrior for MCU 10.6
  
  – Configure the programming/debugging interface in CodeWarrior
  
  – Run a simple program on the S12ZVM EVB
  
  – Watch variables in the debugging interface
Hardware Setup

• Connect the USB cable from the OSBDM interface, J14, to the computer. This is a single interface for:
  - Programming the MCU via the BDM and
  - Communicating with FreeMaster via SCI port.

• Connect the 12V power supply to the VBAT input, J31
Hardware Setup
Initial Configuration

• Software:
  - From the Start menu select **Code Warrior MCU 10.6**
  - Select a workspace and click OK
Launching the Workbench in CodeWarrior

• From the main page, select “Go to Workbench”

The workbench will open with the CodeWarrior Projects view enabled
Import a project

1. Open Lab1 folder. Select the .project file and drag and drop in the Code Warrior space.
The project window

- The example project source file can be found under “Sources”, and it is called “main.c”
Import project into CodeWarrior

• This simple project will:
  – Configure the CPMU to 25MHz core clock / 12.5 MHz bus clock
  – Configure channel 0 of the TIM timer module to a 1millisecond time base
  – Configure channel 1 of the TIM timer module as an XORed input of the hall sensor interface
  – Configure the SCI to 9600 bps for future utilization
  – Toggle LED 1 every 1 millisecond based on TIM channel 0
  – Toggle LED 2 with every edge of the Hall sensor inputs
Example project – header files

```c
#include <hidef.h>    /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "S12ZVMEVB.h" /* EVB definitions */

/* Global variables to store each Hall sensor input */
unsigned char hall_pattern = 0;
unsigned char hall_a_input = 0;
unsigned char hall_b_input = 0;
unsigned char hall_c_input = 0;
```
Example project – Main()

```c
void main(void) {
    initCPMU();  /* configuration for 25MHz core clock */
    initGPIO();  /* configure pins for LED and Hall interface on board */
    initSCI();   /* initialize SCI port 1 at 9600bps (for future use) */
    initTIM();   /* initialize timer channels */

    EnableInterrupts;

    for(;;) {
        __RESET_WATCHDOG();  /* feeds the dog */

        hall_a_input = 0x01 & (hall_pattern>>0);
        hall_b_input = 0x01 & (hall_pattern>>1);
        hall_c_input = 0x01 & (hall_pattern>>2);

        } /* loop forever */
    } /* please make sure that you never leave main */
```
Application Block Diagram

FreeMASTER control

SCI

CPMU

PTU

RAM/NVM memory
PTU trigger lists

Trigger value

Trigger 0

GDU

PWM0
PWM1

3-phase Inverter

ADC 0

RAM/NVM memory
ADC command lists

RAM memory
ADC result lists

Update new:
- PTU delay
- PWM duty-cycle
Set Global LDOK

ADC 0 Done ISR

Global Load OK

re'load

delay

duty-cycle

PTU delay

PWM duty-cycle

Set Global LDOK

PTU delay

PWM duty-cycle

Set Global LDOK

PTU delay

PWM duty-cycle

Set Global LDOK

PTU delay

PWM duty-cycle

Set Global LDOK
Application Block Diagram - CPMU

- **FreeMASTER control**
  - SCI

- CPMU
  - Clock
  - Reset
  - External pin PAD4
  - RAM/NVM memory
    - ADC command lists
  - RAM/NVM memory
    - PTU trigger lists

- PTU
  - Trigger value
  - Trigger 0
  - ADC Done ISR
  - RAM/NVM memory
    - ADC result lists

- RAM/NVM memory
  - PTU trigger lists

- GDU
  - PWM0
  - PWM1
  - Global Load OK
  - reload

- PMF
  - PWM0
  - PWM1
  - duty-cycle

- TIM
  - update new:
    - PTU delay
    - PWM duty-cycle
    - Set Global LDOK

- 3-phase Inverter
CPMU - Clock, Reset and Power Management

- Configuration S/W routine:
  - Set external 4MHz oscillator as clock source
  - Core clock set to 25 MHz
  - Bus clock set to 12.5 MHz
  - Wait for stable PLL operation
  - Clear fail-monitor flags
Example project – CPMU

```c
void initCPMU(void) {

    while (GDUF_GLVLSF) /* Wait for stable supply after power up */
    GDUF_GLVLSF = 1;

    /* Settings for 25MHz/12.5MHz core/bus clocks, out of 4MHz ext. osc. */
    CPMUREFDIV_REFDIV = 3; /* fREF = 4MHz / (3 + 1) = 1MHz */
    CPMUREFDIV_REFFREQ = 0; /* 0 if fREF is from 0 to 1MHz */
    CPMUSYNR_SYNDIV = 24; /* fVCO = fREF * 2 * (24 + 1) = 50MHz */
    CPMUSYNR_VCOFRQ = 1; /* 1 if fVCO is from 48 to 80MHz */
    CPMUPOSTDIV_POSTDIV = 1; /* fPLL = fVCO (1 + 1) = 25MHz */

    CPMUOSC_OSCIC = 1; /* Enable external oscillator */

    while (CPMUFLAG_UPOSC == 0) {} /* Wait for external oscillator */
    while (CPMUFLAG_LOCK == 0) {} /* Wait for PLL to lock */

    CPMUFLAG = 0x60; /* Clear PORF and LVRF flags */
}
```
PIM – Port Integration Module

• The PIM establishes the interface between the peripheral modules and the device I/O pins

• Routing options:
  – SPI0 to alternative pins
  – SCI1 to alternative pins
  – Various SCI0-LINPHY0 routing options
  – PWM channels to GDU and/or pins
  – TIM0 routing to ACLK, RXD0 or RXD1
  – 3 pin input mux to one TIM0 IC channel (logically XORed)
Application Block Diagram – PIM & SCI

FreeMASTER control

SCI

CPMU

RAM/NVM memory
ADC command lists

External pin
PAD4

RAM/NVM memory
ADC result lists

PTU

Trigger value

Global Load OK

3-phase Inverter

RAM/NVM memory
PTU trigger lists

Trigger

PTU delay
PWM duty-cycle
Set Global LDOK

ADC 0

RAM/NVM memory
ADC command lists

ADC 0 Done
ISR

ADCU

PWM0
PWM1

 PWM0
PWM1

PMF

Global Load OK

ADC 0

Done
 ISR

TIM

3-phase Inverter

delay
duty-cycle

reload
Example project – GPIO (PIM) & SCI

```c
void initGPIO(void)
{
    MODRR0 = MODRR0_CONFIG;  /* Serial Port routing*/
    DDRS = DDRS_CONFIG;       /* User LEDs */
    DDRP = DDRP_CONFIG;       /* EVDD output */

    HALL_SUPPLY = 1;         /* Turn on the hall sensor supply (EVDD) */
    LED1 = 0;               /* Turn off LEDs by default */
    LED2 = 0;               /* Turn off LEDs by default */

    MODRR2 = MODRR2_CONFIG;  /* Serial Port routing*/
}

void initSCI(void)
{
    SCI1BD = SCI_BAUDRATE;   /* Set target baud rate = fbus / SCIBD */
    SCI1CR2_TE = 1;          /* Enable Transmitter */
    SCI1CR2_RE = 1;          /* Enable Receiver */
}
```
Application Block Diagram – TIM

FreeMASTER control

SCI

CPMU

Clock
Reset

External pin PAD4

RAM/NVM memory
ADC command lists

RAM/NVM memory
ADC result lists

PTU

Trigger value

RAM/NVM memory
PTU trigger lists

Trigger 0

GDU

PWM0
PWM1

PMF

PTU delay
PWM duty-cycle
Set Global LDOK

ADC 0

ADC 0 Done ISR

ADC 0

ADC 0

3-phase Inverter

Global Load OK

reload
delay
duty-cycle

TIM

Update new:
- PTU delay
- PWM duty-cycle
Set Global LDOK

RAM memory
ADC result lists

ADC 0

ADC 0

PWM0
PWM1

PWM0
PWM1

ADC command lists

ADC result lists

FreeMASTER control

Global Load OK
void initTIM(void)
{
    TIM0TIOS_IOS0 = 1;    /* Channel 0 configured as an output compare */
    TIM0TCTL2 = 0;        /* No action on pin for channel 0 OC event */

    TIM0TIOS_IOS1 = 0;    /* Channel 1 configured as an input capture */
    TIM0TCTL4 = 0xC;      /* Input Capture on both edges of channel 1 */

    TIM0TIE_C0I = 1;      /* Channel 0 interrupt enabled */
    TIM0TIE_C1I = 1;      /* Channel 1 interrupt enabled */

    TIM0TC0 = TIM_CH0_PERIOD;
    TIM0TSCR2_PR = TIM_PRESCALER;

    TIM0TSCR1_TEN = 1;    /* Enable Timer counter */
}
Example project – TIM Interrupt Service Routines

```c
interrupt VectorNumber_Vtim0ch0 void TIM_CH0_ISR(void)
{
    LED1 = ~LED1;         /* Toggle LED 1 */
    TIM0TC0 = TIM0TCNT + TIM_CH0_PERIOD; /* Set new output compare value */
    TIM0TFLG1 = TIM0TFLG1_C0F_MASK;  /* Clear the flag */
}
```

```c
interrupt VectorNumber_Vtim0ch1 void TIM_CH1_ISR(void)
{
    LED2 = ~LED2;         /* Toggle LED 2 */
    hall_pattern = 0x07 & (PTIT >> 1); /* Capture Hall sensor pattern */
    TIM0TFLG1 = TIM0TFLG1_C1F_MASK;  /* Clear the flag */
}
```
Build the project

1. Right click on Lab1 project and select "Clean Project"

2. Click on the “Build All” button

- Click on the “Build All” button

- Right click on Lab1 project and select "Clean Project"
Start the Debugger

1. Click on the drop-down menu at the bug icon

2. Select “Debug Configurations…”
Debug Configurations

- The following image shows the settings of the debugger in the “CodeWarrior Download” configuration
Debug Configurations

1. Click on Debugger Tab and change the Refresh while running to .2
Starting the debugger

- From the Debug Configurations interface, click on Debug
  - Or, from the workspace, click on the Bug icon.

- CodeWarrior will compile the project and program the device.

- You can start the execution of the code with the provided buttons:
Running the code

• When the code is running, you will see that LED1 is ON and slightly dimmed (1ms ON / 1 ms OFF)
• Rotate the motor manually and see how LED2 toggles at different positions of the rotor
• Stopping the execution at any time will show you the current values of the variables being “watched”
What we have learned until now?

• Import existing project into Eclipse IDE workbench
• Setup Eclipse IDE to compile the source code and program it to a target
• Connect with debugging interface to target, and run the programmed device
HANDS-ON SESSION #2

USING FREEMASTER INTERFACE
Hands-on session #2 – Objectives

• In this session you will:

  - Add a routine to initialize the ADC and read the potentiometer value
  
  - Import the FreeMaster Serial Communication Driver into a CW project
  
  - Start a FreeMaster project and visualize the variable data as text
  
  - Add a scope view in freemaster to visualize the variable data
Lab session #2 – Details

• Import the 2nd lab session “S12ZVM_Lab2”

• This project has two additions:
  
  − FreeMaster serial interface driver that “reports” the values of the project variables to FreeMaster, using the serial port SCI1
  
  − The ADC is configured to cyclically read the potentiometer value
Application Block Diagram – ADC

- FreeMASTER control
- SCI
- CPMU: Clock, Reset
- External pin PAD4
- RAM/NVM memory ADC command lists
- RAM/NVM memory ADC result lists
- RAM memory ADC result lists
- PTU
  - Trigger value
  - Trigger 0
  - ADC 0
    - ADC 0 Done
    - ISR
    - Update new:
      - PTU delay
      - PWM duty-cycle
      - Set Global LDOK
    - Global Load OK
    - reload
- RAM/NVM memory PTU trigger lists
- 3-phase Inverter
- GDU
- PMF
- TIM
- PWM0 PWM1
- PWM0 PWM1
- delay
duty-cycle
reload

58  CONFIDENTIAL AND PROPRIETARY
ADC – Analog-to-Digital Converter

- Programmers model with List Based Architecture for conversion command and result value organization
- 8-, 10-, or 12-bit selectable resolution
- Channel select control for n external analog input channels
- Eight additional internal channels
- Programmable sample time
- Providing a sample buffer amplifier for channel sampling (improved performance in view to influence of channel input path resistance versus conversion accuracy)
- Left/right justified result data
- Individual selectable VRH_0/1 and VRL_0/1 Ref. inputs
- Special conversions for selected VRH_0/1, VRL_0/1, (VRL_0/1 + VRH_0/1) / 2
- Providing 15 conversion interrupts with flexible interrupt organization per conversion result
- One dedicated interrupt for “End Of List” type commands
- Provides conversion sequence abort
- The Command Sequence List and Result Value List are implemented in double buffered manner (two lists in parallel for each function)
- Conversion Command loading possible from System RAM or NVM
ADC – Analog-to-Digital Converter

- Configuration S/W routine:
  - Set ADC clock to 6.25 MHz
  - Set output format to 12-bit resolution, right justified data
  - Set ADC in “Trigger” mode
  - Set pointer to ADC Command Sequence Lists
  - Set pointer to ADC Result Value Lists
  - Enable End of list ISR
  - Enable ADC error ISRs
ADC Configuration

• The following global variables have been defined to store the Command List and Results List:

```c
volatile char ADC0CommandList[4][4] @0x001000 = {
    {0xC1,0xD4,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
};
volatile unsigned short ADC0ResultList[4] @0x001020 = {0, 0, 0, 0};
```

• In this case, there is a single list with a single command for ADC0. The command specifies:

```
{ 0xC1, 0xD4, 0x00, 0x00 }
```

- Reserved
- Sample time (0 = 4 ADC clock cycles)
- Channel and reference Voltage select (D4 = AN4, VRH_1/VRL_1)
- Conversion command & Interrupt select (C0 = End of list, interrupt enable for converted channel)
ADC Initialization

```c
void initADC(void)
{
    // Dual access mode
    ADC0CTL_0_ACC_CFG = 3;
    // Trigger mode
    ADC0CTL_0_MOD_CFG = 1;
    // ADC clock = fbus / (2x(ADC0TIM + 1)) [0.25 - 8MHz]
    ADC0TIM = 0;
    // Right justified result data
    ADC0FMT_DJM = 1;
    // 12-bit resolution
    ADC0FMT_SRES = 4;

    // ADC0 Command & Result Base Pointers
    ADC0CBP = ADC0CommandList;
    ADC0RBP = ADC0ResultList;
    ADC0CROFF1 = 0;

    // Enable End-of-list interrupt
    ADC0CONIE_1_EOL_IE = 1;
    // Enable all errors interrupts
    ADC0EIE = 0xEE;
    // Enable ADC0
    ADC0CTL_0_ADC_EN = 1;

    // Issue a restart event
    ADC0FLWCTL_RSTA = 1;
    while (1 == ADC0FLWCTL_RSTA); // Wait until restart flag is cleared
}
```
Adding instructions to trigger ADC

- In this project we will trigger the ADC by software, using Trigger Mode: conversion flow is controlled only by triggers; restart is automatic when EOL conversion is complete.

- There is a new global variable to store the potentiometer value

```c
unsigned short pot_value = 0;
```

- The following lines in the infinite for loop will execute a Trigger every time the first conversion flag (defined in the ADC command) is clear:

```c
if (1 == ADC0CONIF_CON_IF){
   ADC0CONIF_CON_IF = 1; /* Clear flag */
   ADC0FLWCTL_TRIG = 1; /* Trigger next conversion */
}
```
ADC Interrupt Service Routine

- The potentiometer value is obtained in the ADC conversion complete interrupt routine

```c
interrupt VectorNumber_Vadc0conv_compl void ADC0done_ISR(void)
{
    pot_value = ADC0ResultList[0]; /* Update Adc Result */
    ADC0CONIF = 1; /* Clear ISR flag */
}
```
FreeMaster Communication Driver

• Go to [www.freescale.com/freemaster](http://www.freescale.com/freemaster)
  
  - Go to the “downloads” tab and look for “FreeMASTER Communication Driver”
  
  - In the CodeWarrior project window, paste the FreeMASTER folder into the “Project_Headers” folder of your project
  
  - Once the package is installed, there are several options to interface with the target device, using CAN, SCI, or JTAG
  
  For additional information, refer to Freescale’s Application Note AN4752
Using the FreeMaster Serial Driver

• At the top of your project, we have included the freemaster header file:
  #include "freemaster.h"

• The “main” routine now includes a FreeMaster initialization (must be always after the comms initialization; in this case, the SCI):
  FMSTR_Init();

• The infinite for loop now includes a function that continuously sends the variable values to FreeMaster
  FMSTR_Poll();
Start FreeMASTER Interface

• From the Start Menu in Windows, go to
  - Start > All Programs > FreeMaster 2.0

• The FreeMASTER tool will start
  - ignore all the warnings and error messages, they are most probably caused by incorrectly assigned serial port)
FreeMaster – Configuring the Serial Port

• On the menu bar, go to Project > Options

• Select the correct COMM port, with a speed setting of 9600 (this is the value we used in the SCI initialization)
FreeMaster – Loading the MAP file

• From the options window, go to tab “MAP Files”

• Select the default symbol file:
  – Click on “…” and browse to the location where the ELF file is stored (C:\BLDC_workshop\S12ZVM_Lab2\FLASH)
  – Select the file “S12ZVM_Lab2.elf”

• Select the file format:
  – Binary ELF with DWARF1 or DWARF2 dbg format

• Click OK
FreeMaster – Adding Variables

1. Right click on the table and -> Watch Properties

2. Click on Watch and New
3. Select the desire variable on the Address: list

- Add pot_value
- TIM0TCNT
- Hall_a_input
- Hall_b_input
- Hall_c_input
Adding variables to the Watch List

• Right click into “watch” area and select “Watch Properties”
• Switch to tab “Watch” in Project Block Properties
• Select the variables to watch and click on “Add”
Adding a Scope

- Right-click on New Project and select the option “Create Scope”
- Define a name for the scope
- Change Period to 10ms, and Buffer to 700 points per subset
Setup a variable in the scope

1. Select the first unassigned variable slot
2. Select the variable `pot_value` from the dropdown list
3. With BLOCK 0 selected, click on “Assign vars to block”
4. Set the Y-block left axis min value to 0, max value to 5000.
FreeMASTER Interface

- Start/Stop serial communication with target
- Scope selector
- Visualization of variable in scope
- Variable in watch window
FreeMASTER Interface

- In the FreeMASTER interface for “Empty Project” variable time is watched. This variable is also added to scope interface in order to be monitored in graphical representation.
Running the code

• When the code is running, you will see that LED1 is ON and slightly dimmed (1ms ON / 1 ms OFF)
• Rotate the motor manually and see how LED2 toggles at different positions of the rotor
• The value of each hall sensor line can be visualized in the watch list or in the scope view in real time, without interrupting the code execution
• The ADC converted value from the potentiometer is available in another scope view
What we have learned until now?

- How to initialize the ADC
- How to add the FreeMaster serial driver to a project
- How to setup serial interface in FreeMaster
- Adding variables to the “watch” window in FreeMaster
- Adding variables to a scope view in FreeMaster
HANDS-ON SESSION #3

MATH AND MOTOR CONTROL LIB
Hands-on session #3 – Objectives

• In this session you will:

  – Incorporate the Math and Motor Control Library into a CW project

  – Add a function to generate a sinusoidal waveform
Setting up the Math and Motor Control Library

• From the CW menu bar, go to Project > Properties
  - Go to “C/C++ General” > “Paths and Symbols”
  - In tab “Includes” click “Add…” then look for the following path in the file system
    “C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.0\include”
  - Switch to tab “Libraries”
    ▪ Add the following paths under: ”Search User Paths”:
      "C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.0\lib\cw10x\MC9S12ZVM_AMMCLIB_v1.0.0.UC.a “
Setting up the Math and Motor Control Library

- Go to “C/C++ Build” > “Settings”
- Look for the item “Access Paths” under S12Z Compiler
  ▪ Add the following paths under: ”Search User Paths”:
    "C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\include"
    "C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.2\lib\cw10x"

- Look for the item “General” under S12Z Compiler
  ▪ Add the following text in the “Other Flags” field:
    “-DMCLIB_DEFAULT_IMPLEMENTATION=MCLIB_DEFAULT_IMPLEMENTATION_F16”
Adding variables for a sine wave generator

• At the top of your project, include the following libraries:

```c
#include "mlib.h"
#include "gflib.h"
#include "SWLIBS_Config.h"
```

• In the global variables section, add the following lines:

```c
volatile tFrac16 angle1 = 0;
volatile tFrac16 ampl1 = 0;
volatile tFrac16 sin1 = 0;
```

• In the Timer 0 Channel 0 ISR, add the following lines:

```c
angle1 += 70;
amp11 = (tFrac16) (8* pot_value);
sin1 = MLIB_Mul(GFLIB_Sin(angle1), amp11);
```
Visualizing the sine wave in FreeMaster

- Run the FreeMaster application
  - Move the potentiometer and see how the amplitude of the sine wave is affected.
What we have learned until now?

• Adding the Math and Motor Control Library into a project in CodeWarrior.

• Calling a sine function from the math library
HANDS-ON SESSION #4

PMF / PTU / ADC SYNC
Hands-on session #4 – Objectives

• In this session you will:

  - Generate edge-aligned 20KHz PWM signal on phase A
  - Enable the PTU to synchronize ADC conversion trigger with PWM
  - Measure the potentiometer and phase A voltages
  - Adjust the ADC sampling point with the potentiometer
Application Block Diagram – PMF

FreeMASTER control

SCI

CPMU

RAM/NVM memory
ADC command lists

External pin
PAD4

RAM/NVM memory
ADC result lists

3-phase Inverter

PTU

RAM/NVM memory
PTU trigger lists

Trigger value

Trigger 0

Updated new:
- PTU delay
- PWM duty-cycle
Set Global LDK

ADC 0

ADC 0 Done
ISR

RAM/NVM memory
ADC command lists

External pin
PAD4

RAM memory
ADC result lists

GDU

PWM0
PWM1

reload

delay

duty-cycle

Global Load OK

PMF

TIM

ADC 0
done

Reset

Clock
PMF – Pulse Width Modulator

- Three complementary PWM signal pairs, or six independent PWM signals
- Three 15-bit counters based on core clock
- Features of complementary channel operation:
  - Deadtime insertion
  - Separate top and bottom pulse width correction via current status inputs or software
  - Asymmetric PWM output in center-aligned mode (phase shift)
  - Double switching
  - Separate top and bottom polarity control
- Edge-aligned or center-aligned PWM signals
- Half-cycle reload capability
- Integral reload rates from 1 to 16
- Link to timer output compare input for 6-step BLDC commutation support with optional counter restart
- Reload overrun interrupt
- Individual software-controlled PWM output
- Programmable fault protection
PMF – Pulse Width Modulator

- **Configuration S/W routine:**
  - Set PMF clock to core clock (25MHz)
  - Set PWM A to edge-aligned PWM
  - Set output PWM frequency to 20 kHz
  - Set dead time to 0.5 us
  - Disable PWM outputs 2 to 5 (phase B and C)
  - Enable PWM reload event/ISR
PMF – Pulse Width Modulator

```c
void initPMF(void)
{
    PMFCFG0_EDGEA = 1;
    PMFCFG2_REV0 = 1;
    PMFCFG2_REV1 = 0;
    PMFFQCA = 0;
    PMFMODA = PWM_MODULO;
    PMFDTMA = 10;
    PMFCFG2 |= 0x3c;
    PMFENCA_LDOKA = 1;
    PMFENCA_PWMENA = 1;
    PMFENCA_PMRLEA = 1;
    PMFENCA_GLDOKA = 1;
}
```

- Edge-aligned PWM mode
- Enable generation of PWM reload event
- Configure PWM frequency and dead time settings
- Mask PWM outputs 2 to 5 and apply settings with local Load OK
- Enable PWM generator A
  - Enable Reload Interrupt A
  - Switch to Global Load OK for future updates
PTU – Programmable Trigger Unit

- one 16-bit counter as time base
- two independent trigger generators (TG)
- up to 32 trigger events per trigger generator
- Global Load OK support, to guarantee coherent update of all control loop modules
- trigger values stored inside the global memory map
PTU – Programmable Trigger Unit

• Configuration S/W routine:
  - Set PTU trigger list addresses
  - Enable trigger generator 0
  - Enable Trigger Done ISR
PTU – Programmable Trigger Unit

```c
volatile short PTUTriggerEventList[2][3] @0x1030 = {{0x0100,0x0000,0x0000},{0x0000,0x0000,0x0000}};

void initPTU(void)
{
    PTUIEL_TG0DIE = 1;    /* Enable Trigger Generator 0 Done Interrupt */

    /* Map start address & offset for 2nd (currently not used) list
    PTUPTR = PTUTriggerEventList;
    TG0L1IDX = 0;          /* Same as TG0L0IDX */

    PTUE_TG0EN = 1;        /* Enable Trigger Generator 0 */
}

interrupt VectorNumber_Vptutg0dn void PTUTrigger0Done_ISR(void)
{
    LED2 = 1;             /* debug pin ON */
    PTUIFL = (1 << 0);    /* Clear flag */
    LED2 = 0;             /* debug pin OFF */
}
```
Application Block Diagram – ADC

- **FreeMASTER control**
- **3-phase Inverter**
- **PTU**
  - Trigger value
  - Trigger 0
  - ADC 0 Done ISR
  - RAM/NVM memory PTU trigger lists

- **ADC 0**
  - RAM/NVM memory ADC command lists
  - RAM memory ADC result lists
  - Update new:
    - PTU delay
    - PWM duty-cycle
    - Set Global LDOK

- **GDU**
  - PWM0 PWM1
  - Global Load OK

- **PMF**
  - PWM0 PWM1
  - reload
  - delay
  - duty-cycle

- **SCI**
  - Clock
  - Reset

- **CPMU**
  - External pin PAD4

- **RAM/NVM memory**

- **RAM memory**

- **SCI**

- **PWM0**

- **PWM1**

- **TIM**
ADC – Analog-to-Digital Converter

- Configuration S/W routine:
  - Changed to “Restart” Mode for more accurate timing of the triggers based on PTU trigger generator.
  - Changed to left alignment to use the Multiply & Saturate function to scale converted ADC values to PWM duty cycle.
ADC Configuration

- The following changes were done to the Command List:

```c
volatile char ADC0CommandList[4][4] @0x001000 = {
    {0x00,0xD4,0x00,0x00},  //changed from end of list to normal
    {0xC0,0xDA,0x00,0x00},  //added channel from GDU Phase MUX */
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
};
```

- In this case, there are no software commands to restart or trigger the ADC. The flow control is handled entirely by the PMF and PTU.
ADC Initialization

```c
void initADC(void){

    ADC0CTL_0_ACC_CFG = 3; /* Dual access mode */
    ADC0CTL_0_MOD_CFG = 0; /* Restart mode */
    ADC0TIM = 0; /* ADC clock = fbus / (2x(ADC0TIM + 1)) [0.25 - 8MHz] */

    ADC0FMT_DJM = 0; /* Left justified result data */
    ADC0FMT_SRES = 4; /* 12-bit resolution */

    /* ADC0 Command & Result Base Pointers */
    ADC0CBP = ADC0CommandList;
    ADC0RBP = ADC0ResultList;
    ADC0CROFF1 = 0;

    ADC0CONIE_1_EOL_IE = 1; /* Enable End-of-list interrupt */
    ADC0EIE = 0xEE; /* Enable all errors interrupts */
    ADC0CTL_0_ADC_EN = 1; /* Enable ADC0 */

    ADC0FLWCTL_RSTA = 1; /* Issue a restart event */
    while (1 == ADC0FLWCTL_RSTA); /* Wait until restart flag is cleared */
}
```
Application Block Diagram – GDU

- **FreeMASTER control**
  - SCI

- **CPMU**
  - Clock
  - Reset
  - External pin PAD4

- **PTU**
  - Trigger value
  - Trigger 0
  - Global Load OK
  - RAM/NVM memory
    - PTU trigger lists

- **ADC 0**
  - RAM/NVM memory
    - ADC command lists
  - RAM memory
    - ADC result lists
  - Update new:
    - PTU delay
    - PWM duty-cycle
    - Set Global LDOK

- **GDU**

- **PMF**
  - PWM 0
  - PWM 1
  - reload
  - delay
  - duty-cycle

- **3-phase Inverter**

- **TIM**

- **RAM/NVM memory**
  - ADC command lists

- **Lessar**
  - Control
  - CPMU

- **Clock**
  - Reset

- **External pin PAD4**

- **RAM memory**
  - ADC result lists

- **Update new**
  - PTU delay
  - PWM duty-cycle
  - Set Global LDOK
GDU - Gate Drive Unit

- 11V voltage regulator for FET pre-drivers
- Boost converter option for low supply voltage condition
- 3-phase bridge FET pre-drivers
- Bootstrap circuit for high-side FET pre-drivers
- Charge pump for static high-side driver operation
- Phase voltage measurement with internal ADC
- Two low-side current measurement amplifiers for DC phase current measurement
- Phase comparators for BEMF zero crossing detection in sensorless BLDC applications
- Voltage measurement on DC-Link voltage with internal ADC
- Desaturation comparator for high-side drivers and low-side drivers protection
- Undervoltage detection on FET pre-driver supply pin VLS
- Two overcurrent comparators with programmable voltage threshold
- Overvoltage detection on 3-phase bridge supply HD pin
GDU - Gate Drive Unit

• Configuration S/W routine:
  - Clear error flags
  - Configure and enable charge pump
  - Set blanking time to 1.1 us
  - Set desaturation level to 1.35V
  - Enable FET pre-driver
  - Clear desaturation error flags
  - Route Phase A from the Phase voltage Multiplexer to the ADC
GDU Initialization

```c
void initGDU(void){
    GDUE_GCPE = 1; /* Enable charge pump */
    GDUF = 0xFF;  /* Clear High & Low Voltage Supply flags */
    GDUCLK2_GCPCD = 2; /* Set GDU charge pump clock divider to fbuc / 32 */
    GDUCTR = 0x09; /* blanking time 14/12.5M = 1.1us, HD OV threshold = 20V */
    GDUDSLVL = 0x77; /* Desaturation level (1.35 V) */
    GDUE_GFDE = 1; /* Enable FET pre-driver */
    GDUDSE = 0x77; /* Clear Desaturation Error Flags */
    GDUPHMUX = 0x01; /* Route Phase 0 to ADC multiplexer */
}
```
Back-EMF Voltage Measurement

- Back-EMF voltage cannot be measured within all the active PWM pulse as there is switching noise and resonance transient at the beginning of the PWM pulse.

Resonance transient on Back-EMF voltage depends on motor and power stage parameters.

- Time of Back-EMF voltage sample point is used to calculate exact time of the zero-cross.

Measured Back-EMF voltage

Back-EMF voltage unpowered phase
PWM powered phase
ADC Conversion Complete ISR

```c
interrupt VectorNumber_Vadc0conv_compl void ADC0done_ISR(void)
{
    LED2 = 1;    /* debug pin ON */
    AdcResult = ADC0ResultList[0];
    phase_voltage = ADC0ResultList[1];

    /* scale ADC result value to PWM modulo value */
    delay = MLIB_MulSat(PWM_MODULO, ADC0ResultList[0]>>1);
    delay = delay>>2;    /* take 1/4 of the calculated value */
    if (delay < minDelay) delay = minDelay;

    PTUTriggerEventList[0][0] = delay;
    PTUC PTULDOK = 1;

    ADC0CONIF = 1;    /* Clear flag */
    LED2 = 0;        /* debug pin OFF */
}
```
PMF Reload & PTU Trigger Done ISR

```c
interrupt VectorNumber_Vpmfra void PMFreloadA_ISR(void) {
    volatile char tmp;

    LED1 = 1; /* debug pin ON */
    tmp = PMFFQCA;
    PMFFQCA = PMFFQCA_PWMRFA_MASK; /* Clear flag */
    LED1 = 0; /* debug pin OFF */
}

interrupt VectorNumber_Vptutg0dn void PTTTrigger0Done_ISR(void) {
    LED2 = 1; /* debug pin ON */
    PTUIFL = (1 << 0); /* Clear flag */
    LED2 = 0; /* debug pin OFF */
}
```
Visualizing the data in FreeMaster

- Run the FreeMaster application
  - Move the potentiometer and find the threshold at which the phase voltage returns 0.
What we have learned until now?

• How to configure the PMF to output a 20KHz PWM signal

• How to configure the PTU and define a list of triggers

• How to enable the control loop events for synchronization of ADC trigger events based on PMF and PTU

• How to configure the GDU and route a phase voltage to ADC
HANDS-ON SESSION #5

6-STEP COMMUTATION
6-step commutation

• The PMF initialization now enables all three phases in edge-aligned PWM operation

• The FET outputs are configured in complementary mode

• The commutation event is enabled, and the appropriate phases will be configured in preparation for the next commutation event indicated by the TIM output compare channel 0

• The potentiometer value directly sets the PWM duty cycle
  - CAUTION! No speed control implemented
BLDC Motor Commutation Sequence

• The following table shows the Hall sensor patterns corresponding to each Commutation vector in the motor part of the S12ZVM motor control kit.
Commutation Sector Constant Declaration

- The previous values are used to define the masks for the PWM outputs, and their sequence depending of Clockwise or Counter-clockwise operation

```c
const unsigned char MaskVal[7] = {0x34, 0x1C, 0x13, 0x31, 0x0D, 0x07, 0x3F};
const unsigned char OutCtl[7] = {0x0C, 0x30, 0x30, 0x03, 0x03, 0x0C, 0x00};

const unsigned char BLDCPatternBasedOnHall[2][7] = {
    {0, 0, 2, 1, 4, 5, 3},
    {0, 3, 5, 4, 1, 2, 0}
};
```
Applying the new masks

• Inside the TIM channel 1 ISR (occurring upon every edge of any of the Hall Sensor signals), the hall pattern is evaluated and the commutation sector is determined.

• The PWM mask and the SW controlled outputs are configured, and a commutation event is forced on TIM output compare channel 0

```c
hall_pattern = 0x07 & (PTIT >> 1);
cmtSector = BLDCPatternBasedOnHall[rotDir][hall_pattern];
PMFCFG2 = 0x40 + MaskVal[cmtSector];
PMFOUTC_OUTCTL = OutCtl[cmtSector];
TIM0CFORC_FOC0 = 1;
```
Running the project

• **CAUTION**: move the potentiometer all the way clockwise, to set the minimum duty cycle

• When the code is executed, it will immediately start spinning the motor. The speed is set with the potentiometer.
  
  - There is no speed control, so please avoid any drastic changes to the potentiometer settings to make it easy for the motor to adjust to its new speed.
REFERENCES
References

• Freescale Automotive Motor Control Development Solutions:
  http://www.freescale.com/webapp/sps/site/overview.jsp?code=AUTOMCDEVKITS

• S12ZVM Motor Control Kit for BLDC applications
  http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MTRCKTSBNZVM128

• S12ZVM family web site

• S12ZVM Hardware Design Guidelines: