

Table of contents

1	FAQ	2
2	I'm looking for	8
3	Tips 'n Tricks for driving a DC motor	12
3.1	Introduction.....	12
3.2	Basic circuits	13
3.2.1	Single high side/low side drive.....	13
3.2.2	H-bridge driving.....	14
3.3	Basic drive software using PWM	15
3.3.1	Functions description	15
3.3.2	Example c-code	16
3.4	Speed sensing.....	17
3.4.1	Functions description	17
3.4.2	Example c-code	19
3.5	PWM synchronised sensing of the motor current.....	21
3.5.1	Functions description	21
3.5.2	Example c-code	22
3.6	Reading in switches.....	28
3.6.1	Single switches polling.....	29
3.6.2	Global Data / Interfaces	30
3.6.3	Example c-code for 2 low side single switches connected to SW0 and SW1	31
4	Disclaimer	34

1 FAQ

Q When I want to connect the emulator to the chip using the Evaluation-board I receive the message "Error : Chip is not plugged. Check connection and try again". What to do?

A Check that the Mini DIN9 cable is plugged in properly.
Check that Power supply is on.
When using E-mlx programmer do following steps:
Start the programmer software '..\Programmer\EMlxMMProg.exe'
Select File\OpenConfig\MLX16... to open the Mlx81100 mmf file installed by Mlx81100Conf_x_x_x.msi
Select 'Tools\Options\Programming' tick checkbox 'Keep Supply between patterns'
Select tab 'Programmer', choose from pull-down menu 'Mini E-mlx' on 'Autodetect'
Push the 'Diagnose' button -> you should receive the message >>> Info : Hardware successfully checked!
Select tab 'Pattern' and go to pattern 'POWER_ON'
Push the 'Execute' button -> in addition to D2 "VBAT ok" D13 "E-MLX running" should light up

Q How do I enable PWM synchronized shunt ADC measurement?

A Prerequisites:
Enable PWMx_ECI interrupt -> PWM(x)_CMP1_ENABLE(p) .
Set register PWMx_CMP[7:0] to your desired value.
Set bit CFG_PWM[3] = 1 to connect IPWM signal to analog companion.

For ADC measurement of driver state signals it is possible to synchronize ADC measurement to PWM signals. By using the IPWM signal coming from MelexCM the ADC conversion will be started. This strobe signal IPWM must be enabled via register bit "IPWMEN", otherwise it will be ignored. Also bit "SOC" must be set to "1" in order to do this cyclic measurement. SW must take care that IPWM time frame is much bigger than sampling time plus conversion time plus reading result time, otherwise the measurement results can not be true. Using "IPWMEN" signal for synchronization makes only sense if a channel with PWM dependency is selected.

Q How to use the EEPROM?

A See NVRAM_TechNote.pdf for detailed description.
Main features:
124bytes reserved for user
emulated EEPROM in flash
EEPROM is automatically stored at power-down in flash and restored at POR

Q I run out of EEPROM memory, how do I extend the EEPROM?

A Declare an additional buffer in RAM:
extern int myarray[128/2] __attribute__((nodp, addr(0xe780)));
Take care of stack and heap.

At Brown out interrupt first save the “normal” EEPROM buffer to Flash:
example c-code for ISR

/*start of ISR*/

```
PSUP_0(); //set highest priority

ENTER_SYSTEM_MODE();

/* disable all power consumers */

SaveNvRam();
```

Next start to copy from myarray into “normal” EEPROM buffer

```
r = (uint16*)&0xE780;
for(w = (uint16*)&0xFC00; w < (uint16*)&0xFC80;)
{
    *w++ = *r++;
}
```

Next save contents of myarray into flash using following function

```
flash_page_program(FLASH_ADD);
```

where FLASH_ADD is the address in flash where contents of myarray is stored, FLASH_ADD is a multiple of 128bytes. See also..\\melexcm_platform_release1_5_2\projects\Examples\FLASH_EX1 for your reference.

/*end of ISR*/

At POR the “normal” EEPROM contents is restored automatically. The user has to take care of restoring the contents of myarray, e.g.:

```
x = (uint16*)&FLASH_ADD;
for(y = (uint16*)&0xE780; y < (uint16*)&0xE800;)
{
    *y++ = *x++;
}
```

NOTE:

User must take care of not exceeding the maximum write cycles of flash(see spec 82001_cust.pdf, chapter 3.2. Flash and NVRAM).

Q How do I use Brown out interrupt for EEPROM save?

A Please refer to following AN AppNote_MLX82001_EEPROM_UserManual.pdf

Q How do I apply the PWM to the FET drivers?

A Example with c-code:

Init the PWM block

```
PWMA_PLL_CLOCK();
```

```
PWMA_INIT(M, N, PER, PLT, PHT, PCMP, ECI, EPI, MODE, SET);
```

Next call function

```
connectPWM_A(int target);
```

from example motor_control.c with target=[LS1,LS2,HS1,HS2].

Q How do I enable hardware interlock delay?

A Clear bit SINGLE in registers LSxFET and HSxFET, the corresponding driver pair HS1/LS1 (HS2/LS2) is **used in push/pull mode** (necessary when applying inverted PWM to the high side FET). Select bits FILT[1:0] according to your needed duration of interlock delay, Configure a mask time for edges after switching, to ignore comparator's output for the selected time. Each output (HS1; HS2; LS1; LS2) has its own mask time selection in single mode.

Q How do I include the LIN loader in my application?

A Please refer to following documents:

1 software application note "**LIN_Loader_Application_Example.pdf**"

->how to include commands and functions, c-sources

2 application note user manual "**LIN_Loader_UserManual.pdf**"

-> how to use the PC tools

3 application note user manual "**MelexCMSWPlatform_UserManual.pdf**"

->how to include the loader libraries during compilation, compile switches

4 application note user manual "**AppNote_M81100_MLX81200_Reflashing_on_module_1_0.pdf**"

->prerequisites, hardware issues, overall description

Q I want to know the reason of wake-up, POR, LIN or Watch dog. Where is stored?

A The wake up source is stored in register 0xFA0E "**LIN&WU CONF**".

NOTE: The bit *LINWU* must be read before starting the MLX4. Starting the MLX4 CPU **MLX4_START()** will clear *LINWU*. The Mlx4 is started in function **power_on()**.

Q Do I need to write my own LIN API?

A No, Melexis delivers a LIN API running on the MLX16 and MLX4 including documentation, validation test reports and examples. Melexis also delivers the LIN firmware running on the MLX4.

Q How do I set a variable at a fixed address in dp RAM?

A Use following construction

```
extern unsigned char myvar __attribute__((dp, addr(0xa)));
```

Q How to enter Sleep mode from application?

A Sample c-code

```
void GotoSleep(void);

void GotoSleep()
{
/*int SW_CFG_Pin( volatile int SW, int PU,int PD, int OD)*/
SW_CFG_Pin ( 7,1,0,0 ); /*SW7 Pull-up enabled, using low side switch as wake
up source*/
SWCONF=0x10; //Pull-down comparator disabled
//Pull-up comparator enabled
/*using Pull-up comp because we need to detect falling edge on SW7*/
//we need not to write lower byte
//lower byte isn't used for Wake-up
PLL_STOP(); //switch off PLL
MLX4_RESET();
MLX16_STOP();
/*STOP is sent to Analog companion that switches off the voltage
regulators*/

do
{
NOP();
}while(1); //endless loop until Sleep and next POR
}
```

Q How to calculate the analog watch dog time?

A See spec MLX81100_Specification.pdf chapter **7.4.8 Watching the API by Analogue Watchdog (WDOG)**. The timing definition parameter has been selected to make it simple to compute a reference between CWD capacitance value and achieved trigger time:
 $tWDt_R [ms] = CWD[nF]$ and
 $tWD_R [ms] = 0.1 * CWD [nF]$
Example:
CWD = 10nF -> $tWDt_R = 10ms$ and $tWD_R = 1ms$
A complete watchdog period will be for the example above: $10ms + 1ms = 11ms$

Minimum Load capacitor at Pin CWD

A minimum capacitor is needed for parts of software like Flash programming, when WDOG can not be acknowledged by CPU. During design-in, a start value of about 10nF is recommended; this value has to be adapted to needs during application development. Please include temperature dependencies as well as leakage currents into calculation.

Q How to control the slew rate or wave shaping for output PWM's to drive the external FET's?

A To shape the signals at the gates of the FETs one could adapt the series resistors in the line HSx/LSx to the gate. Rise/fall time of the signals at the FET gate is determined by Cgate and Rseries.

Use the following calculation:

Rdson: Rdson of the internal pre-drivers, typ. 40Ohm

Rser: series resistor in the line xxOhm

Cgate gate capacitance of the used FET

Trise -> 10%-90%

$$R_{series} = R_{dson} + R_{ser}$$

$$\tau = R * C$$

$$\tau = R_{sries} * C_{gate}$$

$$T_{rise} = 2.2 * \tau$$

Q I'm using a communication interface different to LIN. What things do I need to take care of?

A Please refer to following documents:

1 hardware application note "[Application_Note_MLX81100_PWM_general_1_1.pdf](#)"

2 application note user manual "[AppNote_M81100_MLX81200_Reflashing_on_module_1_0.pdf](#)"

The pin LIN must always be routed to the modules connector.

2 I'm looking for...

Index

A B C D E F G H I J K L M N O P Q R S T
U V W X Y Z

A

- ADC MLX81100_Specification.pdf
- Analog watchdog MLX81100_Specification.pdf
- Application schematic examples Application_Note_MLX81100_example_circuits.pdf

Up

B

- Block diagram MLX81100 MLX81100_Specification.pdf
- Block diagram MLX82001 digital core 82001_cust.pdf
- Brown out interrupt MLX81100_Specification.pdf

Up

C

- Current consumption MLX81100_Specification.pdf

Up

D

- Digital watchdog 82001_cust.pdf

Up

E

EEPROM NVRAM_TechNote.pdf
AppNote_MLX82001_EEPROM_UserManual.pdf

Electrical characteristics analog IC..... MLX81100_Specification.pdf

Electrical characteristics digital IC 82001_cust.pdf

[Up](#)

F

FET output driver (HS1/LS1, HS2/LS2) . MLX81100_Specification.pdf

[Up](#)

G

[Up](#)

H

High voltage IOs MLX81100_Specification.pdf

[Up](#)

I

Interrupts description 82001_cust.pdf

IOs description 82001_cust.pdf

[Up](#)

J

Jump start MLX81100_Specification.pdf

[Up](#)

K

[Up](#)

L

LIN API	MelexCM_LIN_API.pdf MelexCM_Standard_LIN_API.pdf
LIN Loader	LIN_Loader_TechNote.pdf
LIN Loader user manual	LIN_Loader_UserManual.pdf
LIN loader software example	LIN_Loader_Application_Example.pdf
LIN transceiver	MLX81100_Specification.pdf
Load dump interrupt	MLX81100_Specification.pdf

Up

M

Memories description	82001_cust.pdf
MLX16 op code	MLX16X8_DataBook.pdf
MLX4 control	82001_cust.pdf

Up

N

NVRAM	see EEPROM
-------------	------------

Up

O

Up

P

Package information	MLX81100_Specification.pdf
Pin description	MLX81100_Specification.pdf
Power up/down sequence	82001_cust.pdf
PWM blocks description	82001_cust.pdf

Up

Q

[Up](#)

R

Register set analog..... MLX81100_Specification.pdf

Register set digital 82001_cust.pdf

Reset MLX81100_Specification.pdf
82001_cust.pdf

[Up](#)

S

SPI block description 82001_cust.pdf

SW pins description see High voltage IOs

System control bits 82001_cust.pdf

[Up](#)

T

Timer blocks description..... 82001_cust.pdf

[Up](#)

U

UART block description 82001_cust.pdf

[Up](#)

V

[Up](#)

W

Wake up by local sources or LIN..... MLX81100_Specification.pdf

[Up](#)

X

[Up](#)

Y

[Up](#)

Z

[Up](#)

3 Tips 'n Tricks for driving a DC motor

3.1 Introduction

This chapter covers information referring the basic software parts and the basic electronic devices needed to drive a DC motor.

3.2 Basic circuits

3.2.1 Single high side/low side drive

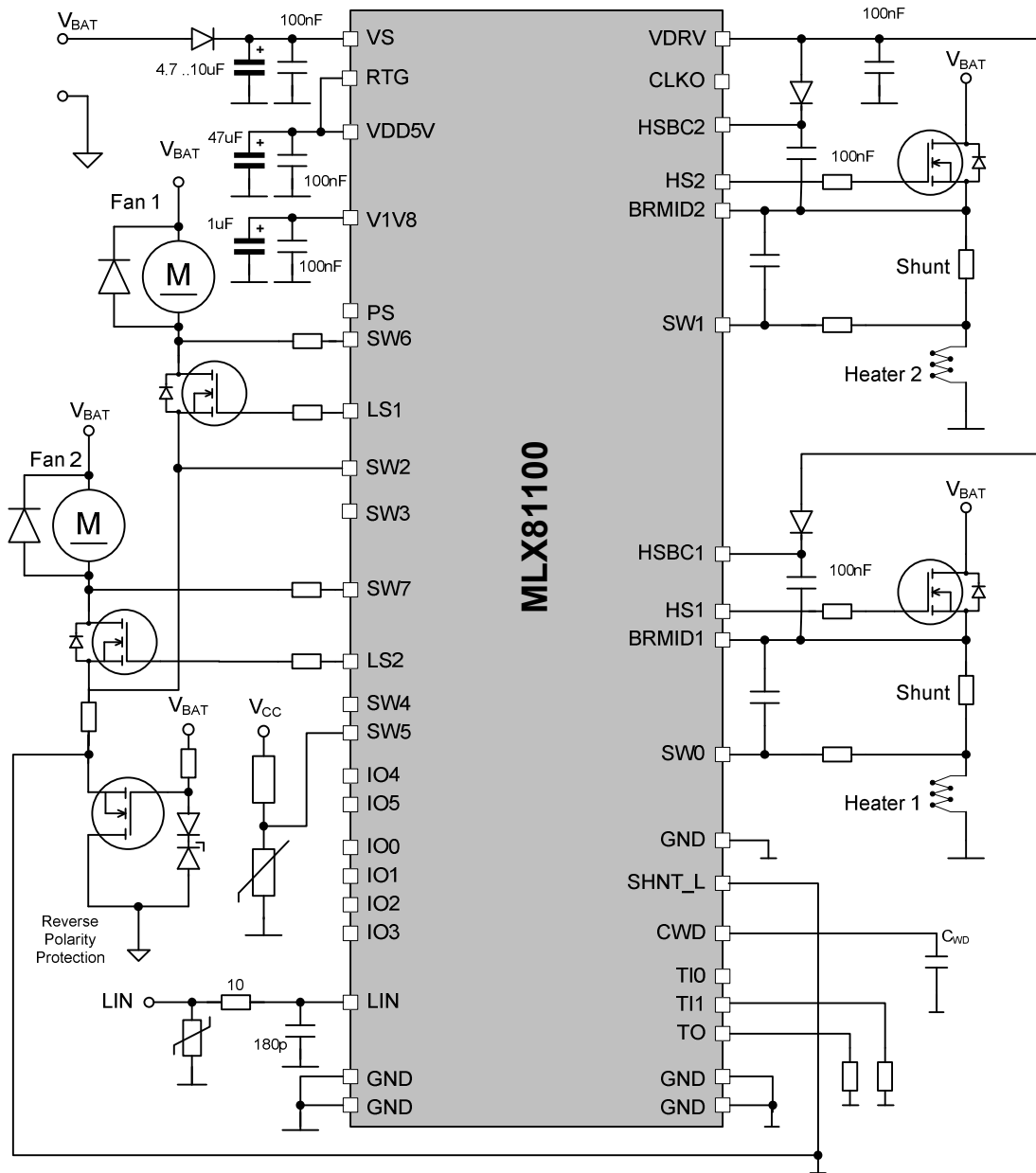


Figure 1: Basic schematic for single FET driving

3.2.2 H-bridge driving

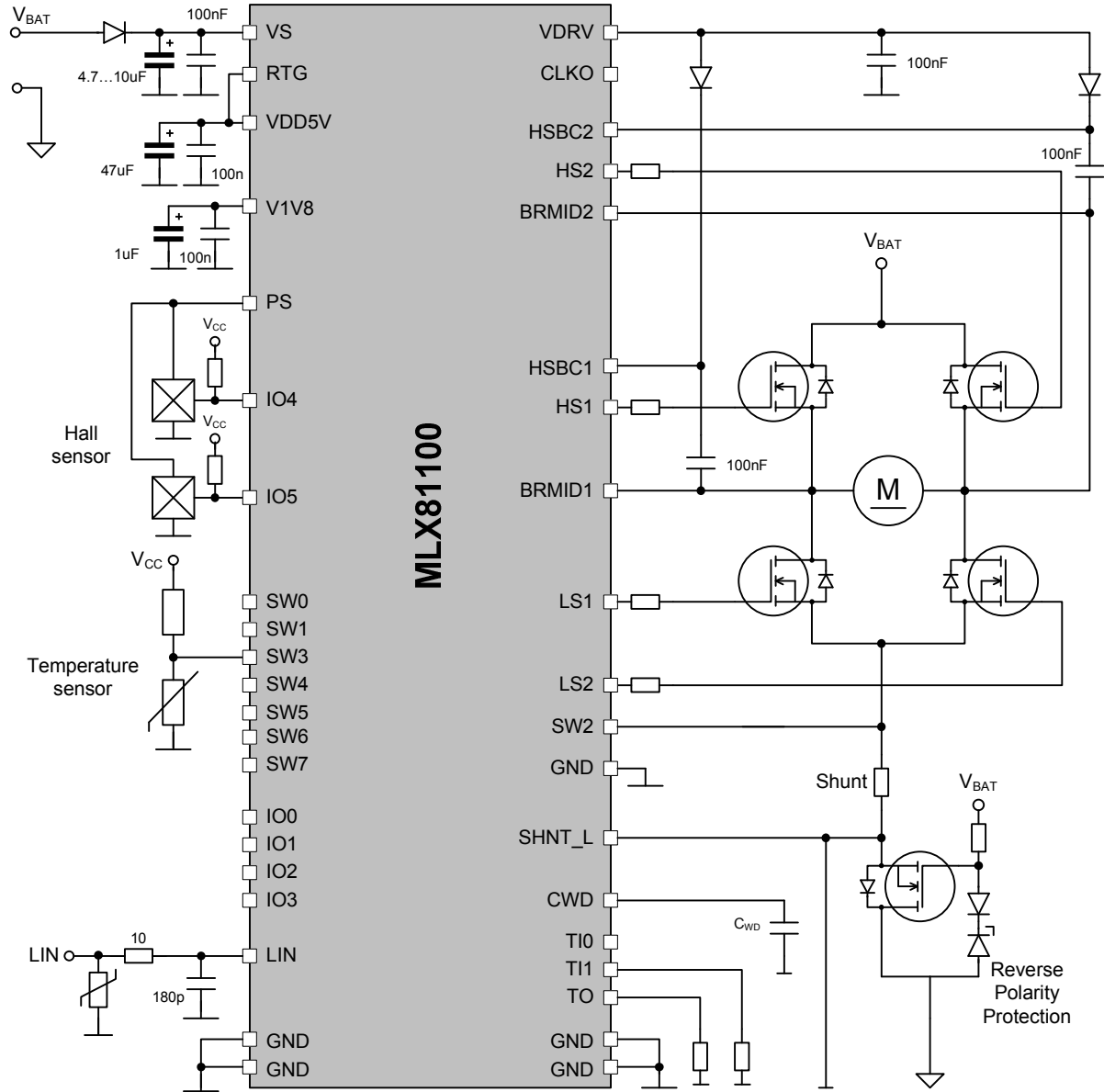


Figure 2: Basic schematic for H bridge driving

3.3 Basic drive software using PWM

The following c-code uses Macros coming with the software platform release for MLX81100 available under Softdist.

3.3.1 Functions description

PWM configuration

Syntax:

```
F1: void configPWM()
```

Description:

The function is used to configure PWM block A in PWM master mode.

Following MACROS are called:

- PWMA_PLL_CLOCK();
- connects the PLL clock to the PWM block
- PWMA_INIT(5,0,0xff,0x00,0x80,0,0,0,0,0);
- enables Timer block A in pulse accu mode
- only rising edges are counted
- edges are sampled with PLL frequency

Bridge configuration

Syntax:

```
F3: void initBridge()
```

Description:

The function is used to configure the H bridge and init all FET drivers.

Following MACROS are called:

- switchVDRV(on);
- switches on the VDRV supply voltage
- switchFET([HS1,HS2,LS1,LS2],off)
- all Fet drivers are switched off

Routing PWM signal to FET

Syntax:

```
F4: void connectPWM_A(LS2)
```

Description:

The function is used to connect the PWMA signal to FET **LS2**.

Switch FET

Syntax:

```
F5: void switchFET(LS2)
```

Description:

The function is used change the output state *[int flag]* of the FET *[int target]*

- with target=[LS1,LS2,HS1,HS2] and flag=[on,off].

The desired direction of the motor movement is determined by the parameters target:

	Direction forward	Direction backward
HS1	100% on	Inverted PWMA
LS1	100% off	PWMA
HS2	Inverted PWMA	100% on
LS2	PWMA	100% off

3.3.2 Example c-code

Supposing to let the motor move forward see example code below:

```
PWMA_PLL_CLOCK();
/*PWMA_INIT(M, N, PER, PLT, PHT, PCMP, ECI, EPI, MODE, SET);*/
PWMA_INIT(5,0,0xff,0x00,0x80,0,0,0,0,0);
InitBridge();
connectPWM_A(LS2);
switchFET(HS1,on);
switchFET(LS1,off);
switchFET(HS2,off);
```

The FET drivers are configured in push-pull mode so that the inverted PWMA is automatically applied to the HS2 FET driver. The interlock delay can be adapted by writing the bits FILT[1:0] according to your needed duration of interlock delay.

3.4 Speed sensing

Following features apply:

- One hall output is connected to IO0
- Timer A is used to count the hall pulses
- Hall pulses are stored in global variable
- Amount of hall pulses in a certain time period equals the speed.

3.4.1 Functions description

IO pins configuration

Syntax:

```
F1: void cfgIOPins ()
```

Description:

The function is used to configure IO0 to be input for Timer A input channel A.

Timer configuration

Syntax:

```
F2: void configTimer()
```

Description:

The function is used to configure Timer block A in Pulse accumulator mode.
Following MACROS are called:

- `TIMERA_OSC_CLOCK();`
- connects the PLL clock to the timer block
- `TIMER_PULSE_INIT(A, TIMER_DIV_1, TIMER_EDGA_RISE);`
- enables Timer block A in pulse accu mode
- only rising edges are counted
- edges are sampled with PLL frequency
- `TIMERA_INT3_ENABLE(3);`
- timer overflow interrupt is enabled
- interrupt priority is set to "3"

Timer interrupt

Syntax:

```
F3: void __interrupt__ TimerA_INT3 ()
```

Description:

The interrupt routine of TimerA is called when a Timer overflow occurred. This means 0xFFFF rising hall pulses were counted.

Global Data / Interfaces

Syntax:

```
D1:volatile uint16 Hall_Count_Rising=0;
```

Description:

This variable represents the accumulated number of rising hall pulses and has to be set global.

3.4.2 Example c-code

```

#include "alib.h"
#include "board.h"

/*****
/* prototypes of functions */
/*****
void cfgIOPins(void); // configure the IO pins of the MelexCM
void configTimer(void); //config Timer

/*****
/* prototypes of interrupt routines */
/*****
void __interrupt__ TimerA_INT3();

volatile uint16 TimAOVRF=0;

volatile uint16 Hall_Count_Rising=0;

//define constants

// To be used for P2M16T port
#define TMRA_A_IO0 0 // Define which IO
#define TMRB_A_IO1 ( 1 << 4 ) // controls the channel
#define TMRC_A_IO2 ( 2 << 8 ) // input 'TMRx_A'
//define functions

#define TIMER_PULSE_INIT(U, DIV, EDGA)
do
{
TCTRL_##U = 1;
TREGA_##U = 0;
TREGB_##U = 0;
TCTRL_##U = 3|TIMER_MODE_PULSE|(DIV)|(EDGA);
}while(0)

/*****
/* configTimer() */
/* */
/* Function that sets TimerA in PULSE ACCU mode */
/* input: */
/* output: */
/* result: TimerA is running in PULSE ACCU mode */
/*****
void configTimer()
{
TIMERA_OSC_CLOCK();

TIMER_PULSE_INIT(A, TIMER_DIV_1, TIMER_EDGA_RISE);

TIMERA_INT3_ENABLE(3);
}

```

```

/*****/
void TimerA_INT3()
{
// TIMER INTERRUPT INT3
//1.disable Int
//2.Inc Hall_Count_Rising
//3.enable Int
TIMERA_INT3_DISABLE(); // 1.Disable Timer interrupts
TimAOVRF++; // 2.Inc TimAOVRF
TIMERA_INT3_ENABLE(3); // 3.Enable Timer interrupts
return;
}

```

```

/*****/
/*the mainloop of the application*/
int main()
{
cfgIOPins();
configTimer();

do
{
Hall_Count_Rising=TCNT_A;
} while (1);

return 1;
}

```

```

/*****/
/*
*/
/* configure the IO pins of the MelexCM */
/*
*/
/*****/
void cfgIOPins()
{
// configure the IO pins of the MelexCM as output

```

```

IO0_IO1_INIT(IO_FUNC_INPUT, IO_OD_CMOS, IO_Z_35, IO_FUNC_INPUT, IO_OD_CMOS, IO_Z_35);

P2M16T|=TMRA_A_IO0;
}

```

3.5 PWM synchronised sensing of the motor current

3.5.1 Functions description

ADC configuration

Syntax:

```
F1: void cfgADC ()
```

Description:

The function is used to configure the ADC for shunt measurement. The pin SW2 is connected to the high side of the Shunt.

Measuring the offset

Syntax:

```
F2: void measureOffset ()
```

Description:

The function is used to measure the offset of the differential op-amp.

ADC reference configuration

Syntax:

```
F3: void ADC_SelectReferenceVoltage(int VREF)
```

Description:

The function is used to select the ADC voltage reference.

ADC input channel configuration

Syntax:

```
F4: void ADC_SetChannel(int Channel)
```

Description:

The function is used to select the ADC channel.

SWpin configuration

Syntax:

```
F5: int SW_CFG_Pin(int SW, int PU, int PD, int OD)
```

Description:

The function is used to configure the SW pins of MLX81100.

ADC measurement

Syntax:

```
F6: unsigned int ADC_READ(void)
```

Description:

The function is used to start one ADC conversion. It returns the ADC value.

3.5.2 Example c-code

```
#include "board.h"

#include "alib.h" //include all libs at once

void cfgADC(void); // configure the ADC
void measureOffset(void);
void ADC_SelectReferenceVoltage(int VREF);
void ADC_SetChannel(int Channel);
unsigned int ADC_READ(void);
extern int SW_CFG_Pin(int SW, int PU, int PD, int OD);

/*****ADC variables*****/
volatile unsigned int ADCValue=0;
volatile unsigned int ADCOffset=0;

/***** CONSTANTS for ADC registers *****/
#define PRY6_ANA_CUST7( 3 << 10 ) //Define the priority forANACUSTI(7)
/*****/

/*****/
/*the mainloop pf the application*/
int main()
{

wdog_hook();

//config the PWM -> PWMA master, PWMB slave, enable PWMA_CMPI, update Threshold simultan
CFG_PWM |= 0x08;// Set bit CFG_PWM[3] = 1 to connect IPWM signal to analog companion.

PWMA_PLL_CLOCK();

PWMA_INIT(5,0,0xff,0x00,0x80,0x70,1,0,0,0);

//Set register PWMx_CMP[7:0] to your desired value

//Enable PWMx_ECI interrupt
PWMA_CMPI_ENABLE(3);

//config the ADC -> ADC is ready for shunt measurement SW2-SHUNT_L, VREF=1V, GAIN=1
cfgADC();
```

```

InitBridge();

//measure the offset of ADC channels
measureOffset();

connectPWM_A(LS2);

switchFET(HS1,on);
switchFET(LS1,off);
switchFET(HS2,off);

ADC_CTRL1 |= (1 << SOC); // SOC=1, starts sample phase of AD conversion

// the mainloop of the firmware
do {

wdog_hook();

} while (1);

return 1;
}

/*****/
void __interrupt__ Analog_INT7(void)
{
if (ADCcount>=Buffersize)
{
ADC_CTRL1 &= ~(1 << SOC); // SOC=0, stop sample phase of AD conversion
// SOC is automatically set when synchronize with IPWM
}
}

ADCValue=ADC_DAT12_W & 0x03FF;//Read ADC-Value and trim to 10Bit

ADCValue-=ADCOffset;

}
/*****/

/* cfgADC() */
/* */
/* Function that sets ADC for shunt measurement */
/* input: */
/* output: */
/* result: ADC is ready for shunt measurement */
/*****/
void cfgADC()
{
SetIOBit(COM_PERMCK);// if Bit in premain.S set ->Register is not writable anymore

SPRDH |= PRY6_ANA_CUST7;

```

```
ADC_SelectReferenceVoltage(VRH1);
SW_CFG_Pin(2,0,0,0);
ADC_SetChannel(ADC_Channel15);
```

```
ADC_CTRL1 |= (1 << GAINDISO) | (1 << IPWM_EN);
// IPWM_EN=1, synchronize SOC with IPWM starts sample phase of AD conversion
// GAINDISO=1, disable gain, gain=1
```

```
}
/*****
*/
/* measureOffset() */
/* */
/* Function that gets the offset of ADC for shunt measurement */
/* input: */
/* output: */
/* result: ADC is ready for shunt measurement */
/*****
void measureOffset()
{
int temp=0;
temp=ADC_CTRL12_W; //save ADC_CTRL word
ADC_CTRL1 &= ~(1 << IPWM_EN); /* IPWM_EN=0, don't synchronize SOC with IPWM starts sample
phase of AD conversion*/
ADC_CTRL2 |= (1 << OFFS); // OFFS=1, measure offset
ADCOffset=ADC_READ();
ADC_CTRL1 |= (1 << IPWM_EN); /* IPWM_EN=0, don't synchronize SOC with IPWM starts sample
phase of AD conversion*/
ADC_CTRL2 &= ~(1 << OFFS); // OFFS=1, measure offset
ADC_CTRL12_W=temp; //restore ADC_CTRL word
}
```

```
// ***** ADC_SetChannel*****
// Description:
// - Set ADC-Channel
//
// parameters: int channel ->(0..15)
// return:
// *****
void ADC_SetChannel(int Channel)
{
int temp;
temp= ADC_CTRL1 & 0xFFF0;//

switch (Channel)
{
case ADC_Channel0: //SW0
case ADC_Channel1: //SW1
case ADC_Channel2: //SW2
case ADC_Channel3: //SW3
case ADC_Channel4: //SW4
case ADC_Channel5: //SW5
case ADC_Channel6: //SW6
case ADC_Channel7: //SW7
case ADC_Channel15: //SW2
temp|= (Channel << ADCSEL0); //Set Channel SW-Pins
//Channel15 is connected to SW2
if (Channel==ADC_Channel15) Channel=ADC_Channel2;
```

```

SW_CFG_Pin(Channel, 0, 0, 0);
//clear Opendrain
//clear PullDown
//clear PullUp
break;
case ADC_Channel8:           //MLX81100
case ADC_Channel9:           //MLX81100
case ADC_Channel10:          //MLX81100
case ADC_Channel11:          //VSS
case ADC_Channel12:          //Temp
case ADC_Channel13:          //MLX81100
case ADC_Channel14:          //MLX81100
temp |= (Channel << ADCSEL0); //Set Channel
break;
default: ;
}
ADC_CTRL1=temp;              //write Selected-Channel to Port
MSEC_DELAY(1);               // settling time experimental value!!!
}

// ***** ADC_Read *****
// Description:
// - Start ADC-Conversion / wait for result
// -
// parameters: none
// return: 10Bit ADC-Value
// *****
unsigned int ADC_READ(void)
{
    unsigned int ReturnValue=0;
    SetIOBit(COM_PERMCK);     // if Bit in premain set ->Register is not writable anymore
    ADC_CTRL1 |= (1 << SOC);  // SOC=1, starts sample phase of AD conversion

    while (!(ADC_CTRL1 & (1<<EOC))) //wait for End of Conversion
    {
        wdog_hook();
    }

    ReturnValue=ADC_DAT12_W & 0x03FF; //Read ADC-Value and trimm to 10Bit
    ClrIOBit(COM_PERMCK);          // if Bit in premain.S set ->Register is not writable anymore
    return ReturnValue;
}

// ***** ADC SelectReferenceVoltage*****
// Description:
// - set Referenz-Voltage for the ADC
//
// parameters: int VREF ->VRH1,VRH2,VRH3
// *****
void ADC_SelectReferenceVoltage(int VREF)
{
    unsigned int temp;
    temp =ADC_CTRL2;
    temp &= ~(1 << VRH1) | (1 << VRH2) | (1 << VRH3);
    switch(VREF)
    {
        case VRH2:
            temp |= (1 << (VRH2));
    }
}

```

```

break;

case VRH3:
temp |= (1 << (VRH3));
break;

case VRH1:
default:
temp |= (1 << (VRH1));
break;

}
ADC_CTRL2 = temp;

}

// ***** SW_CFG_Pin*****
// Description:
// - select SW0..SW7 pin
// - can set/clear Pullup/Pulldown/Open-drain
// parameters: int SW ,PU ,PD ,OD
// return: 0-success, -1 wrong Port
// edit:-disable all unused Bit
//
// *****

int SW_CFG_Pin( volatile int SW, int PU,int PD, int OD)
{

if ((0<SW) && (SW>8))//if not SW=0..7
{
return -1; //cancel -wrong Port
}

else
{
//Pullup
if(PU==0)
SWPU = 0x00; //Clear all Pullup
else
SWPU = ( 1 << (SW)); //Set only 1 Pullup
//Pulldown
if(PD==0)
SWPD = 0x00; //Clear all Pulldown
else
if (PU==0) // Set Pulldown only
SWPD = ( 1 << (SW)); //if Pullup off
// Opendrain
if(OD==0)
{
SWOD = 0x00; //Clear all OpenDrain
}
else
{
SWOD = ( 1 << (SW)); //Set only 1 Opendrain
}
}
}

```

```
return 0; //success  
}
```

3.6 Reading in switches

Following features apply:

- low side switches are connected to the SW pins of MLX81100
- switches are debounced

In the example there are only two single switches polled. The switches have to be checked separately because there is only one comparator for all SW-switches available. The following figure shows the hardware connection of the single switch polling.

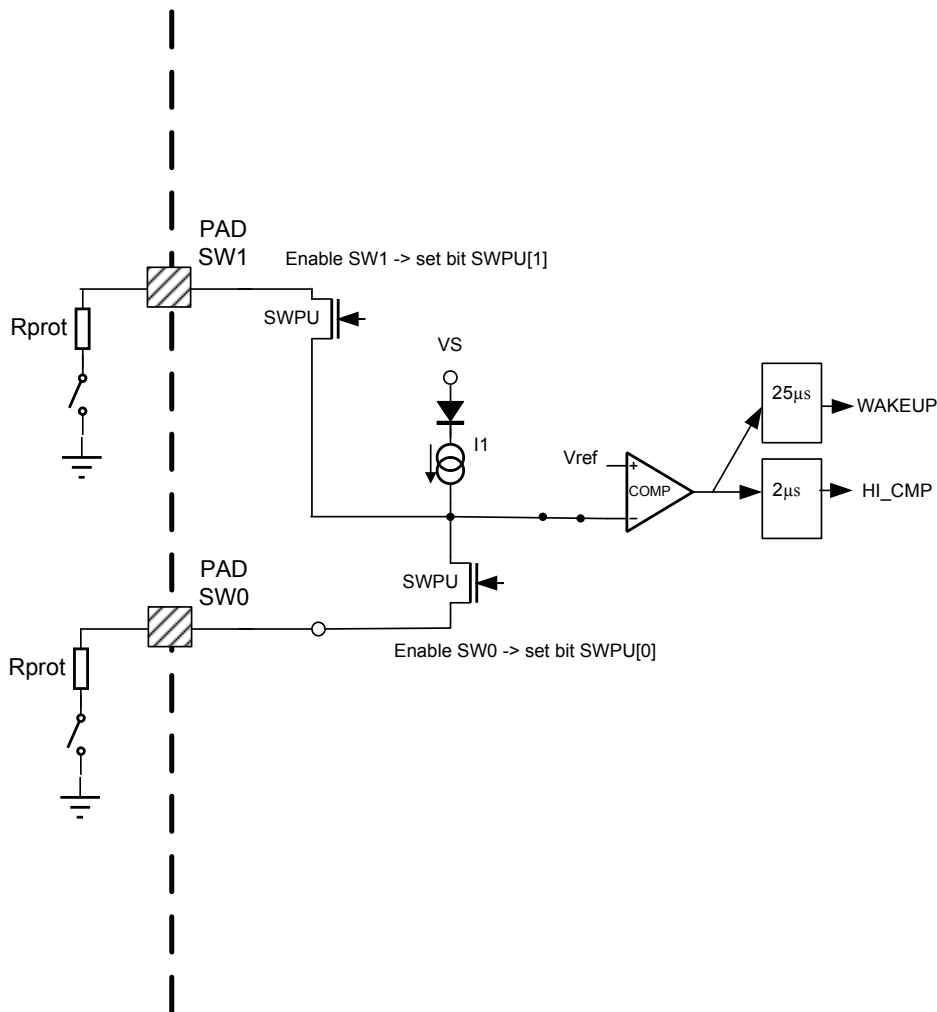


Figure 3: SW principle hardware schematic

3.6.1 Single switches polling

The following figure shows the control flow of the single switch polling.

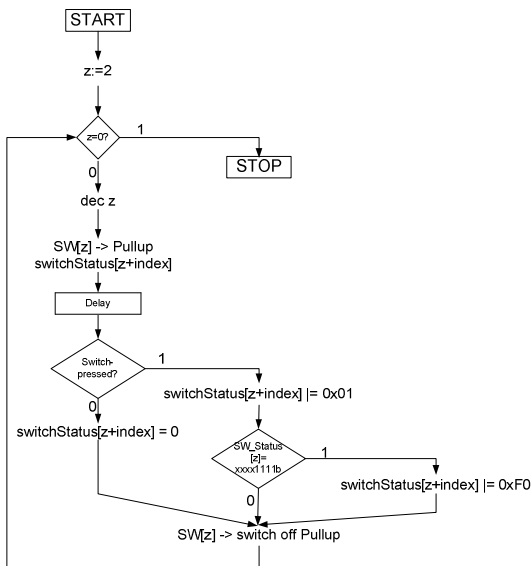


Figure 4: Control Flow - Single Switches

At first pull up current source will be switched on. The switch status variable will be shifted one bit left. Afterwards, the status of the single switch will be checked. Thereafter, the pull up current source of the single switch will be switched off. The second single switch can be checked. The single switches SW0 and SW1 should be used.

3.6.2 Global Data / Interfaces

Every switch gets one byte for its status. The first nibble is used to get the debouncing status and the second one shows the current switch status. There are 2 bytes (2 Single Switches) for the switch status necessary. These are stored global because they could be used by other functions, e.g. for the preparation of the LIN-Frame data.

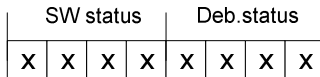


Figure 5: Figure 7: Content of the switch status variable

All possible occupancies of the byte in figure 9 are shown in the following table.

Status	Switch status				Debouncing status			
Switched off	0	0	0	0	0	0	0	0
Switched off	0	0	0	0	0	0	0	1
Switched off	0	0	0	0	0	0	1	0
Switched off	0	0	0	0	0	0	1	1
Switched off	0	0	0	0	0	1	0	0
Switched off	0	0	0	0	0	1	0	1
Switched off	0	0	0	0	0	1	1	0
Switched off	0	0	0	0	0	1	1	1
Switched off	0	0	0	0	0	0	0	0
Switched off	0	0	0	0	1	0	0	1
Switched off	0	0	0	0	1	0	1	0
Switched off	0	0	0	0	1	0	1	1
Switched off	0	0	0	0	1	1	0	0
Switched off	0	0	0	0	1	1	0	1
Switched off	0	0	0	0	1	1	1	0
Switched on	1	1	1	1	1	1	1	1

Table 1: Storage occupancy of the switch-status

It's easier to handle these variables as an array. The size of the array is 2 * 1 byte.

Syntax:

```
D1: uint8 switchStatus [2];
```

Description:

This array represents the status of all switches and has to be set global. The storage occupancy of every byte is shown in table 1. In the following table is depicted the assignment of every byte.

Array	Switch status				Debouncing Status			
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
switchStatus [1]	Single Switch 2 (SW1)							
switchStatus [0]	Single Switch 1 (SW0)							

Table 2: Assignment of switchStatus

3.6.3 Example c-code for 2 low side single switches connected to SW0 and SW1

```
//-includes-----
#include "board.h" //get io customer ports MLX81100
#include "switches.h" //function and constants definitions
#include "alib.h"

//define constants
#define SINGLE_SW 2 //number of single switches

//current stabilization delay
#define Delay_10us __asm__ volatile ("djnz x,." : : "x" (12));

//define functions
void pollSwitches(void); //poll switch status
#define SWPU0_set 1
#define SWPU1_set 2
#define SWPU2_set 3
#define SWPU3_set 4
#define SWPU4_set 5
#define SWPU5_set 6
#define SWPU6_set 7
#define SWPU7_set 8

//-define-global-variables-----

//switch status and debouncing status
uint8 switchStatus[SINGLE_SW];

//-function-pollSwitches()-----
//-low side switches-----
//poll switches to get their status
void pollSwitches(void)
{
    //define local variables
    uint8 count_single_sw=0; //count single switches
    uint8 index=0; //index for switchStatus

    //poll single switches
    for(count_single_sw=0;count_single_sw<SINGLE_SW;count_single_sw++)
    {
        switch(count_single_sw) //enable switch
        {
            case 0: SWPU |= SWPU0_set;break; //first single switch
            case 1: SWPU |= SWPU1_set;break; //second single switch
            default:break;
        }

        Delay_10us(); //wait 10us for current stabilization
        wdog_hook();
        //shift status var. one bit left
        switchStatus[index] <<= 1;
    }
}
```

```
if((SWCMP & HI_CMP) != HI_CMP) //get status of single switch
{
    //set first debouncing bit to 1
    switchStatus[index] |= 1;

    //check debouncing status of single switch
    if ((switchStatus[index] & 0x0F) == 0x0F)
    {
        //set single switch status to on
        switchStatus[index] |= 0xF0;
    }
}
else
{
    switchStatus[index] = 0; //reset switch status
}

switch(count_single_sw) //disable switch
{
    case 0: SWPU &= ~SWPU0_set;break; //first single switch
    case 1: SWPU &= ~SWPU1_set;break; //second single switch
    default:break;
}

index++; //set new index
}
```

History record

Rev.	No.	Change	Date
1.0	1	Creation	10/08/08

References

[1]	Specification digital core MLX82001	82001_cust.pdf
[2]	Specification analog companion MLX81100	MLX81100_specification.pdf
[3]	AN example schematics	Application_Note_MLX81100_example_circuits.pdf
[4]	EEPROM TechNote	NVRAM_TechNote.pdf
[5]	EEPROM user manual	AppNote_MLX82001_EEPROM_UserManual.pdf
[6]	LIN API description	MelexCM_LIN_API.pdf
[7]	Standard LIN API description	MelexCM_Standard_LIN_API.pdf
[8]	LIN Loader	LIN_Loader_TechNote.pdf
[9]	LIN Loader user manual	LIN_Loader_UserManual.pdf
[10]	LIN loader software example	LIN_Loader_Application_Example.pdf
[11]	MLX82001 machine code description	MLX16X8_DataBook.pdf

4 Disclaimer

Devices sold by Melexis are covered by the warranty and patent indemnification provisions appearing in its Term of Sale. Melexis makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. Melexis reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with Melexis for current information. This product is intended for use in normal commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by Melexis for each application.

The information furnished by Melexis is believed to be correct and accurate. However, Melexis shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interrupt of business or indirect, special incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of Melexis' rendering of technical or other services.

© 2007 Melexis NV. All rights reserved.

For the latest version of this document, go to our website at:

www.melexis.com

Or for additional information contact Melexis Direct:

Europe and Japan:
Phone: +32 1367 0495
E-mail: sales_europe@melexis.com

All other locations:
Phone: +1 603 223 2362
E-mail: sales_usa@melexis.com

ISO/TS 16949 and ISO14001 Certified