ECOLE CENTRALE DE NANTES

LABORATORY REPORT LAB4

Timer – PWM – Interrupt – Control of a power converter

By:

AROMOSE QUDUS AYINDE

SUBMITTED TO: PROF. MICHAEL HILARET

Program: Embedded Computing

Program Specialization: M1 Electric Vehicles Propulsion and Control.

January 12, 2024.

QUESTION 1: CLASSIFICATION OF THE ARM TIMERS

The STM32F303x6/8 includes advanced control timer, 5 general-purpose timers, basic timer, two watchdog timers and a SysTick timer. The table below compares the features of the advanced control, general purpose and basic timers.

Table 1: Timer Features Comparison

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary outputs
Advanced control	TIM1(1)	16-bit	Up, Down, Up/Down	Any integer between 1 and 65536	Yes	4	Yes
General-purpose	TIM2	32-bit	Up, Down, Up/Down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM3	16-bit	Up, Down, Up/Down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM15	16-bit	Up	Any integer between 1 and 65536	Yes	2	1
General-purpose	TIM16, TIM17	16-bit	Up	Any integer between 1 and 65536	Yes	1	1
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

1. TIM1 can be clocked from the PLL x 2 running at up to 144 MHz when the system clock source is the PLL and AHB or APB2 subsystem clocks are not divided by more than 2 cumulatively.

PART 1 : Control of a buck, boost or buck/boost converter

Question 2 : PWM configuration

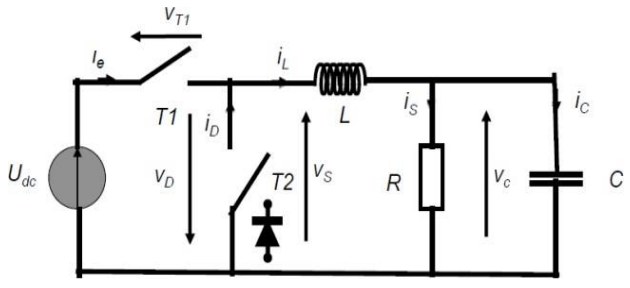

Figure 1: Boost Converter.

Select timer TIM2 and configure one output as a PWM with the following configuration :

- PWM frequency at 10kHz.
- Up-counting configuration (edge-aligned mode)
- Active high
- The counter continue to count at the next overflow
- New duty cycle value is taken into account only when there is an overflow (for the newt PWM period),
- The duty cycle is constant at 25%.

Solution

For this event we selected TIMER 2 CHANNEL 1, PORT PA0 according to the table in Appendix1

The appropriate PSC and ARR was calculated from equation (1)

$$PSC = 0, ARR = 6399$$

$$ARR = \frac{f_{board}}{f(PSC+1)} - 1 = \frac{T}{T_{board}(PSC+1)} - 1 \quad (1)$$

$$T = (PSC + 1)(ARR + 1)T_{board} \quad (2)$$

The code is written in listing 1.0

LISTING 1.0: CODE FOR QUESTION 1

```
#include "main.h"

void setup (){

//1 - pin configuration:

RCC->AHBENR |= RCC_AHBENR_GPIOAEN; // or clockForGpio(port);


//set PA0 as alternative function

GPIOA->MODER &= ~(0x3<<0);//reset bit MODER0[0:1]

GPIOA->MODER |= 0x2; //set PA0 as alternate function


// Set alternate function (see datasheet page 241/1141)

GPIOA->AFR[0] &= ~(0x0F<<0); // reset AFR0[3:0] bits

GPIOA->AFR[0] |= 0x01<<0; // set AFR0[3:0] bits as "0001" = AF1


//2 - timer configuration (use TIM2@10kHz)

RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

__asm("nop");
```

```
//reset peripheral (mandatory!)
RCC->APB1RSTR |= RCC_APB1RSTR_TIM2RST;
RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM2RST;
__asm("nop");
//config timer@10kHz, with 6400 ticks
TIM2->PSC = 0; //prescaler : 64MHz tick@1/64us
TIM2->ARR = 6399; //auto-reload: counts 6400 ticks

//3- PWM configuration
TIM2->CCMR1 &=~(0x03<<0); //Channel 1 as output cc1S
TIM2->CCMR1 &= ~(0x010070); //reset OC1M[3:0]=0000, bit 16,6,5,4
TIM2->CCMR1 |= 6 << 4; //output PWM mode 1 "0110"
TIM2->CCMR1 |= 1<<3;//pre-load register, update during event
TIM2->CR1 &= ~(7<<4); //mode 1 // edge aligned mode plus direction =0
TIM2->CCER |= 1; //enable channel 1
TIM2->CR1 |=1 ; //config reg : enable, 1<<0
TIM2->CCR1 = 1600; //25% of ARR
}

int main()
{
setup();
}
```

The result of listing 1 is shown in figure 1

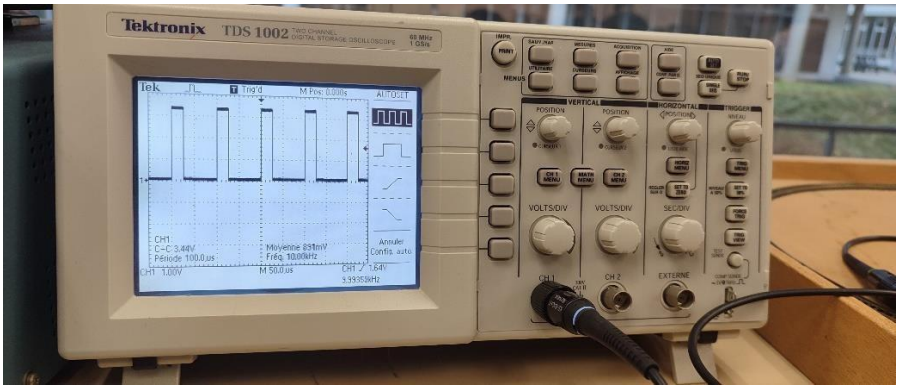

Figure 1: Oscilloscope Result Display.

QUESTION 3 : DUTY-CYCLE VARIATION

Based on Lab2, use the ADC to tune the duty-cycle of the PWM signal. Observe the PWM signal with an oscilloscope or digital analyser.

As instructed by the professor, Question 3 was exempted.

QUESTION 4 a: PWM AND INTERRUPT

Add an interrupt when the timer overflow. Design the code of the Interrupt Service Routine (ISR) so that the duty-cycle fluctuate between 0 to 100% and come back to 0 indefinitely.

LISTING 2.0: PWM AND INTERRUPT

```
void setup (){

RCC->AHBENR |= RCC_AHBENR_GPIOAEN; // or clockForGpio(port);

//set PA0 as alternative function

GPIOA->MODER &= ~(0x3<<0);//reset bit MODER0[0:1]

GPIOA->MODER |= 0x2; ////set PA0 as alternate function

GPIOA->MODER &= ~(0x3<< 2);//reset bit MODER1[0:1]

GPIOA->MODER |=(0x01<<2);//set pinA1 as output

GPIOA->AFR[0] &= ~(0x0F<<0); // reset

GPIOA->AFR[0] |= 0x01<<0; // "0001" = AF1

//2 - timer configuration (use TIM2@10kHz)

RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

__asm("nop");

//reset peripheral (mandatory!)

RCC->APB1RSTR |= RCC_APB1RSTR_TIM2RST;

RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM2RST;

__asm("nop");

//config timer@100kHz, with 6400 ticks
TIM2->PSC = 0; //prescaler : tick@100us
```

```
TIM2->ARR = 6399; //auto-reload: counts 6400 ticks

//3- PWM configuration
TIM2->CCMR1 &=~(0x03<<0); //Channel 1 as output cc1S
TIM2->CCMR1 &= ~(0x010070); //OC1M[3:0]=0000, bit 16,6,5,4
TIM2->CCMR1 |= 6 << 4; //output PWM mode 1 "0110"
TIM2->CCMR1 |= 1<<3;//pre-load register, update during event
TIM2->CR1 &= ~(7<<4); //mode 1 // edge aligned mode plus direction =0
TIM2->CCER |= 1; //enable channel 1
TIM2->CR1 |=1 ; //config reg : enable, 1<<0

TIM2->CCR1 = 1600; //25%
//Enable Interrupt
TIM2->DIER |= TIM_DIER_UIE;//0x01<<0 Update interupt flag when it overflow
NVIC_EnableIRQ(TIM2_IRQn);// TIM2 with priority 28 for the interrupt handler

}


void TIM2_IRQHandler()
{
TIM2->SR &= ~TIM_SR_UIF;//reset the update
inc+=0.00001;//to increment every micro seconds
if(inc>=1)
{
inc=0;//when the increment is up to 100 percent increment stop
}
TIM2->CCR1=(unsigned short int)(ARR_MAX*inc);//To vary the pulse width in channel 1
TIM2->CCR2=(unsigned short int)(ARR_MAX*inc);//To vary the pulse width in channel 2
}


int main()

{

setup();

}
```

QUESTION 4 b: PWM AND INTERRUPT SYNCHRONIZATION

How you can check that the sampling period of the ISR is correct ?

- The sampling period of the ISR can be check by sending interrupt flash signal at the output bit **PA1** in the interrupt handler and it is check on the oscilloscope.

```
GPIOA->ODR |= 1<<1;//setting PA1
for(volatile int i=0; i<20; i++);//set high at least for 10us
GPIOA->ODR &= ~(0x1<<1);//resetting PA1
```

- Verify that the ISR is exactly synchronized with the PWM signal: It is shown in Figure 2 that the interrupt signal at 10kHz.

```
void TIM2_IRQHandler()
{
TIM2->SR &= ~TIM_SR_UIF;//reset the update
GPIOA->ODR |= 1<<1;//setting PA1
for(volatile int i=0; i<20; i++);//set high at least for 10us
GPIOA->ODR &= ~(0x1<<1);//resetting PA1
}

int main()

{

setup();

}
```

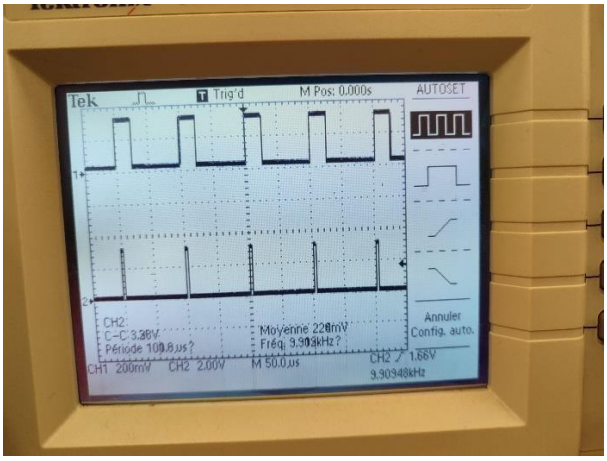

Figure 2: PWM with interrupt signals.

QUESTION 5: CURRENT CONTROLLER

5a. Write the controller equation in continuous-time with the proposed variables below:
IP Controller

The IP Controller is given in equation(3) and it control scheme is given in figure 3.

$$I(t) = K_i \int e(t)\,dt - K_p I(t) \quad (3)$$

5b. Discretize the controller based on a Euler or trapezoidal approximation,

According to the figure 4

Forward approximation: $I_k = I_{k-1} + a_k T_s$ (4)

Backward approximation: $I_k = I_{k-1} + a_{k-1} T_s$ (5)

The trapezoidal approximation: $I_k = I_{k-1} + \frac{T_s}{2}(1 + z^{-1})a_k$ (6)

Using the discretized controller based on trapedoidal approximation the control law is given in(7)

$$u(k) = int\_previous + \frac{T_s}{2}(1 + z^{-1})e(t) - K_p I \quad (7)$$

And the control scheme of the forward discretized system is with anti-windup is shown in Figure 3.

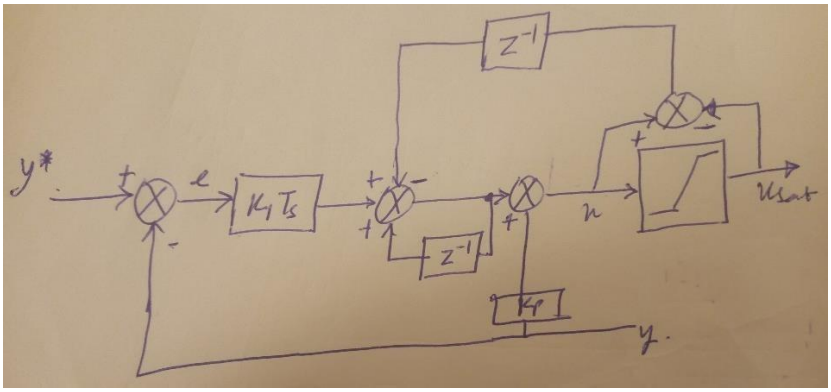

Figure 3. Control scheme of IP Discrete Controller with anti-windup.

The code is implemented in Listing 3.0 in the interrupt handler.

Listing 3.0: Implementation of IP control law

```
#define ARR_MAX 6399
#define _1surduty 0.01
float inc =0;
#define _10over4095 0.00244200244
#define Kp 10
#define Ki 1.0
#define Ts 0.0001
#define Udc 100
#define _1surUdc 0.01
float erreur[2] = {0, 0};
float int_current = 0;
float current_ref = 200;
float current = 0;
float prev_current = 0;
float usat = 0;
float u = 0;
float dc;

void TIM2_IRQHandler()
{
TIM2->SR &= ~TIM_SR_UIF;
// error
erreur[0]=current_ref-current;
//euler ki_ with antiwinding up
int_current+=(Ki*erreur[0]*Ts)-(u-usat);
//reference
```

```
u=int_current-Kp*current;

if(u>=Udc)
{
usat=Udc;
}
else if(u<=0)
{
usat=0;
}
else{
usat=u;
}
dc = usat * _1surUdc;
TIM2->CCR1 = dc * ARR_MAX;
}

int main()

{

setup();

}
```

Question 6 : Active low Reconfigure the CCER register so that the PWM is active low.

```
TIM2->CCER |= (1<<4); //CC1p bit=1 active low
```

When change the polarity of the channel we obtained a mode 2 PWM signal as shown in Figure 4.

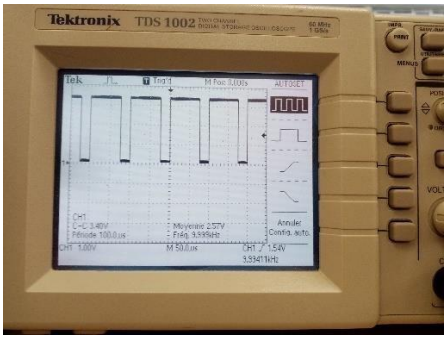

Figure 4: Active low PWM

QUESTION 7 : ACTIVE LOW NEXT RECONFIGURE THE CCMR1 REGISTER SO THAT THE PWM IS ACTIVE LOW (I.E MODE 2). LATER ON, CHANGE ALSO CCER.

```
TIM2->CCMR1 &= ~(0x010070); //OC1M[3:0]=0000, bit 16,6,5,4

TIM2->CCMR1 |= 7 << 4; //output PWM mode 2 "0111"
```

Result give an active high PWM in Figure 6.

```
TIM2->CCER &= ~(1<<4); //CC1p bit=0
```

Figure 7 displays result when CCER bit is invert then it give an active high.

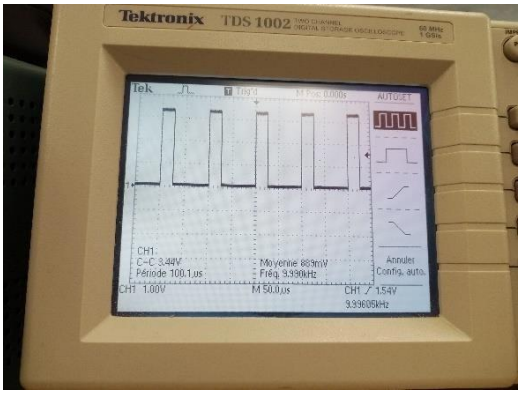

Figure 6: PWM MODE 2 with CC1P=1

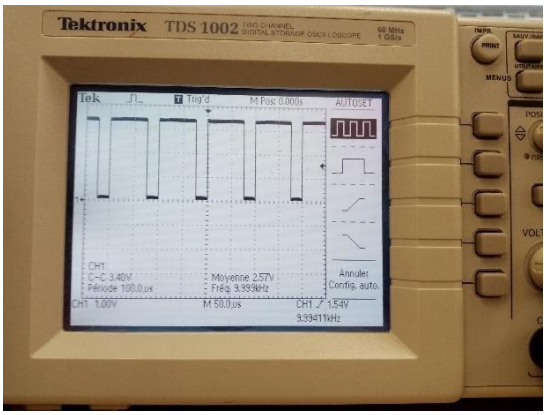

Figure 7: Active low PWM MODE 2 with CC1P=0

Question 8 : PWM with a down-counter Reconfigure the PWM so that the counter is now a down-counter. To obtain the same duty-cycle in the PWM signal as previously, what do you need to change in the code ?

To obtain the same duty-cycle with down counter we change the polarity in the CC1P=1 and the result is shown in Figure 8. And the piece of code is shown below:

```
TIM2->CR1 &= ~(1<<4); //DIR bit4=0
TIM2->CCER |= (1<<4); //CC1p bit=1 active low
```

PART 2: CONTROL OF A FULL-BRIDGE CONVERTER

Question 9

ENABLE	IN1	IN2	Vs	Comments
0	X	X	0	0 at stead state
1	0	0	0	Lower freewheeling
1	0	1	-VDC	Increase of speed
1	1	0	+VDC	Decrease of speed
1	1	1	1	Upper freewheeling

QUESTION 10: BIPOLAR CONTROL

For this case Timer 2, Channel 1 and Channel 2 are configured with their corresponding pins PA0 and PA1 respectively according to the data sheet

```
#define ARR_MAX 6399
void setup (){

RCC->AHBENR |= RCC_AHBENR_GPIOAEN; // enable I/O port A clock;

RCC->AHBENR |= (1<<18); // or allow port B to access the clock

//set PB3 as output for enable signal

GPIOB->MODER &= ~(0x1<<7);//reset bit 7

GPIOB->MODER |= (0x1<<6);//set bit 6

//disable the motor

GPIOB->ODR &= ~(0x1<<3);// reset bit 3 of ODR

//set PA0 as alternative function

GPIOA->MODER &= ~(0x3<<0);

GPIOA->MODER |= 0x2;


//set PA1 as alternative function

GPIOA->MODER &= ~(0x3<<2);

GPIOA->MODER |= (0x2<<2);

//Set the Alternate funtion for PA0 mode 1

GPIOA->AFR[0] &= ~(0x0F<<0); // reset
```

```
GPIOA->AFR[0] |= 0x01<<0; // "0001" = AF1
//Set the Alternate funtion for PA1 mode 2
GPIOA->AFR[0] &= ~(0x0F<<4); // reset
GPIOA->AFR[0] |= 0x01<<4; // "0001" = AF1

//2 - timer configuration (use TIM2@10kHz)
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
__asm("nop");
//reset peripheral (mandatory!)
RCC->APB1RSTR |= RCC_APB1RSTR_TIM2RST;
RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM2RST;
__asm("nop");

//config timer@10kHz, with 6400 ticks
TIM2->PSC = 0; //prescaler : tick@100us
TIM2->ARR = 6399; //auto-reload: counts 6400 ticks

//3- PWM configuration
//Channel 1
TIM2->CCMR1 &=~(0x03<<0); //Channel 1 as output cc1S
TIM2->CCMR1 &= ~(0x010070); //channel 1 as output (~(0x03<<8) to put "00")
TIM2->CCMR1 |= 6 << 4; //output PWM mode 1 "0110"
TIM2->CCMR1 |= 1<<3;//pre-load register, update during event, 1<<3
TIM2->CCER |= 1; //enable channel 1
TIM2->CCR1 = 1600; //25% channel 1

//Channel 2
TIM2->CCMR1 &=~(0x03<<8); //Channel 2 as output cc2S
```

```
TIM2->CCMR1 &= ~(0x01007<<12); //clear the 16 bit regiter of CCMR1

TIM2->CCMR1 |= (0x07 << 12); //output PWM mode 2 "0111"

TIM2->CCMR1 |= (0x1<<11);//pre-load register, update during event, 1<<11

TIM2->CCER |= (1<<4); //enable channel 2

TIM2->CCR2 = 1600; //25% for channel 2


TIM2->CR1 &= ~(7<<4); // edge aligned mode plus direction =0

TIM2->CR1 |=1 ; //config reg : enable, 1<<0


//enable signal is on

GPIOB->ODR |=(0x01<<3);


//Enable Interrupt

TIM2->DIER |= TIM_DIER_UIE;//0x01<<0 Update interupt flag when it overflow

NVIC_EnableIRQ(TIM2_IRQn);// TIM2 with priority 28 for the interrupt handler

}



void TIM2_IRQHandler()
{

TIM2->SR &= ~TIM_SR_UIF;//reset the update

inc+=0.00001;

if(inc>=1)

{

inc=0;
}
TIM2->CCR1=(unsigned short int)(ARR_MAX*inc);
TIM2->CCR2=(unsigned short int)(ARR_MAX*inc);
}


int main()

{

setup();

}
```

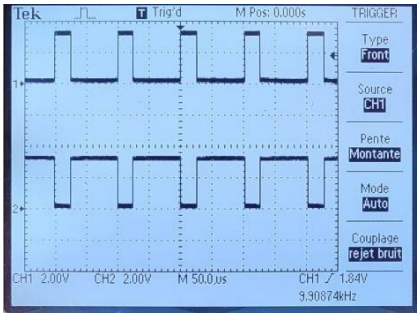

Figure 8: Bipolar Control PWM.

Question 11 : Dead-time with Advanced-Control-Timers

For this Question TIM 1 was selected according to the Alternate Function in Appendix A

20.4.18 TIM1/TIM8/TIM20 break and dead-time register (TIMx_BDTR)

Address offset: 0x44

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	BK2P	BK2E	BK2F[3:0]				BKF[3:0]			
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		DTG[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **DTG[7:0]**: Dead-time generator setup
This bit-field defines the duration of the dead-time inserted between the complementary outputs. DT correspond to this duration.
DTG[7:5]=0xx => DT=DTG[7:0]x t_{dtg} with $t_{dtg}=t_{DTS}$.
DTG[7:5]=10x => DT=(64+DTG[5:0])xt_{dtg} with $T_{dtg}=2xt_{DTS}$.
DTG[7:5]=110 => DT=(32+DTG[4:0])xt_{dtg} with $T_{dtg}=8xt_{DTS}$.
DTG[7:5]=111 => DT=(32+DTG[4:0])xt_{dtg} with $T_{dtg}=16xt_{DTS}$.
Example if T_{DTS}=125ns (8MHz), dead-time possible values are:
0 to 15875 ns by 125 ns steps,
16 us to 31750 ns by 250 ns steps,
32 us to 63us by 1 us steps,
64 us to 126 us by 2 us steps
Note: This bit-field can not be modified as long as LOCK level 1, 2 or 3 has been programmed (LOCK bits in TIMx_BDTR register).

For bit DTG[7:5]=0b10000000 => DT=(64+DTG[5:0])xtdtg with Tdtg=2xtDTS=2x1/64MHz DT=2us.

And the Main Output is enabled TIM1->BDTR|=(1<<15);

The implementation of the code can be found in Listing 5.0

LISTING 5.0: DEAD-TIME WITH ADVANCED-CONTROL-TIMERS

```
void setup (){

// enable I/O port A clock;

RCC->AHBENR |= (1<<17);
```

```
RCC->AHBENR |= (1<<18); // or allow port B to access the clock
__asm("nop");

//set PB3 as output for enable signal
GPIOB->MODER &= ~(0x1<<7);//reset bit 7
GPIOB->MODER |= (0x1<<6);//set bit 6
//disable the motor
GPIOB->ODR &= ~(0x1<<3);// reset bit 3 of ODR
//set PA7 as alternative function
GPIOA->MODER &= ~(0x3<<14);
GPIOA->MODER |= (0x2<<14);
//set PA8 as alternative function
GPIOA->MODER &= ~(0x3<<16);
GPIOA->MODER |= (0x2<<16);

//Set the Alternate function for PA7
GPIOA->AFR[0] &= ~(0x0F<<28); // reset
GPIOA->AFR[0] |= 0x06<<28; // "0110" = AF6

//Set the Alternate funtion for PA8 (GPIOx_AFRH)
GPIOA->AFR[1] &= ~(0x0F<<0); // reset
GPIOA->AFR[1] |= 0x06<<0; // "0110" = AF6

//2 - timer configuration (use TIM1@10kHz)
RCC->APB2ENR |= (0x1<<11);
__asm("nop");

//reset peripheral (mandatory!)
```

```
RCC->APB2RSTR |= (0x1<11);// set Timer 1

RCC->APB2RSTR &= ~(0x1<11);//reset Timer 1

__asm("nop");

//config timer@100kHz, with 100 ticks (duty cycle step at 1%)

TIM1->PSC = 0; //prescaler : tick@10us

TIM1->ARR = 6399; //auto-reload: counts 100 ticks

//3- PWM configuration

TIM1->CR1 &= ~(7<<4); // edge aligned mode plus direction =0, upcounting

TIM1->CR1 |=1 ; //config reg : enable, 1<<0

TIM1->CCR1 = 1600; //25% channel 1

//Channel 1

TIM1->CCMR1 &=~(0x03<<0); //Channel 1 is configured as output

TIM1->CCMR1 &= ~(0x010070); //channel 1 as output (~(0x03<<8) to put "00")

TIM1->CCMR1 |= 6 << 4; //output PWM mode 1 "0110"

TIM1->CCMR1 |= 1<<3;//pre-load register, update during event, 1<<3

TIM1->CCER |= 1; //enable channel 1 with complementary

TIM1->CCER |= (1<<2); //enable channel 1N

//Dead time generation

TIM1->BDTR &= ~(0xFF<<0);

TIM1->BDTR |= (0x80<<0);//2us dead time

//Enable Main Output

TIM1->BDTR|=(1<<15);

//enable signal is on
```

```
GPIOB->ODR |=(0x01<<3);
}


int main()

{

setup();

}
```

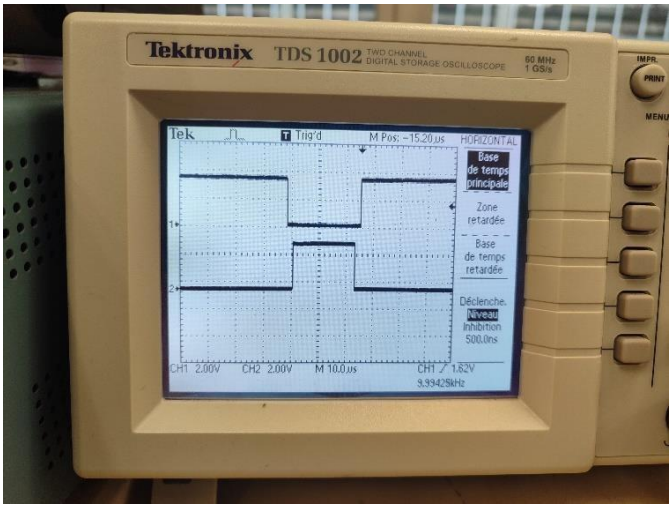

Figure 9: Bipolar Control with Dead-Time using Advanced-Control-Timers

CONCLUSION

The following were achieved:

- **Generation of PWM signal with Timer configuration.**
- **IP controller design in Timer Interrupt Handler.**
- **Generation of Bipolar Control PWM Complementary signals Using TIMER of different channels.**
- **Generation of Bipolar Control PWM Complementary signals with Dead-Time using Advanced-Control-Timers.**

APPENDIX 1

A1. ALTERNATE FUNCTIONS

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS_AF	TIM2/TIM15/ TIM16/TIM17/ EVENT	TIM1/TIM3/ TIM15/ TIM16	TSC	I2C1/TIM1	SPI1/ Infrared	TIM1/ Infrared	USART1/USA RT2/USART3/ GPCOMP6	GPCOMP2/ GPCOMP4/ GPCOMP6	CAN/TIM1/ TIM15	TIM2/TIM3/TI M17	TIM1	TIM1	OPAMP2	-	EVENT
Port A	PA0	-	TIM2_CH1/TI M2_ETR	-	TSC_G1_IO1	-	-	-	USART2_CTS	-	-	-	-	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	-	TSC_G1_IO2	-	-	-	USART2_RTS _DE	-	TIM15_CH1N	-	-	-	-	-	EVENTOUT
	PA2	-	TIM2_CH3	-	TSC_G1_IO3	-	-	-	USART2_TX	COMP2_OUT	TIM15_CH1	-	-	-	-	-	EVENTOUT
	PA3	-	TIM2_CH4	-	TSC_G1_IO4	-	-	-	USART2_RX	-	TIM15_CH2	-	-	-	-	-	EVENTOUT
	PA4	-	-	TIM3_CH2	TSC_G2_IO1	-	SPI1_NSS	-	USART2_CK	-	-	-	-	-	-	-	EVENTOUT
	PA5	-	TIM2_CH1/TI M2_ETR	-	TSC_G2_IO2	-	SPI1_SCK	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA6	-	TIM16_CH1	TIM3_CH1	TSC_G2_IO3	-	SPI1_MISO	TIM1_BKIN	-	-	-	-	-	-	-	-	EVENTOUT
	PA7	-	TIM17_CH1	TIM3_CH2	TSC_G2_IO4	-	SPI1_MOSI	TIM1_CH1N	-	-	-	-	-	-	-	-	EVENTOUT

A2. TIMX CAPTURE/COMPARE MODE REGISTER 1 (TIMX_CCMR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

A3. TIMX CAPTURE/COMPARE MODE REGISTER 2 (TIMX_CCMR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC2M [3]	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OC1M [3]
							Res.								Res.
							rw								rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw