Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

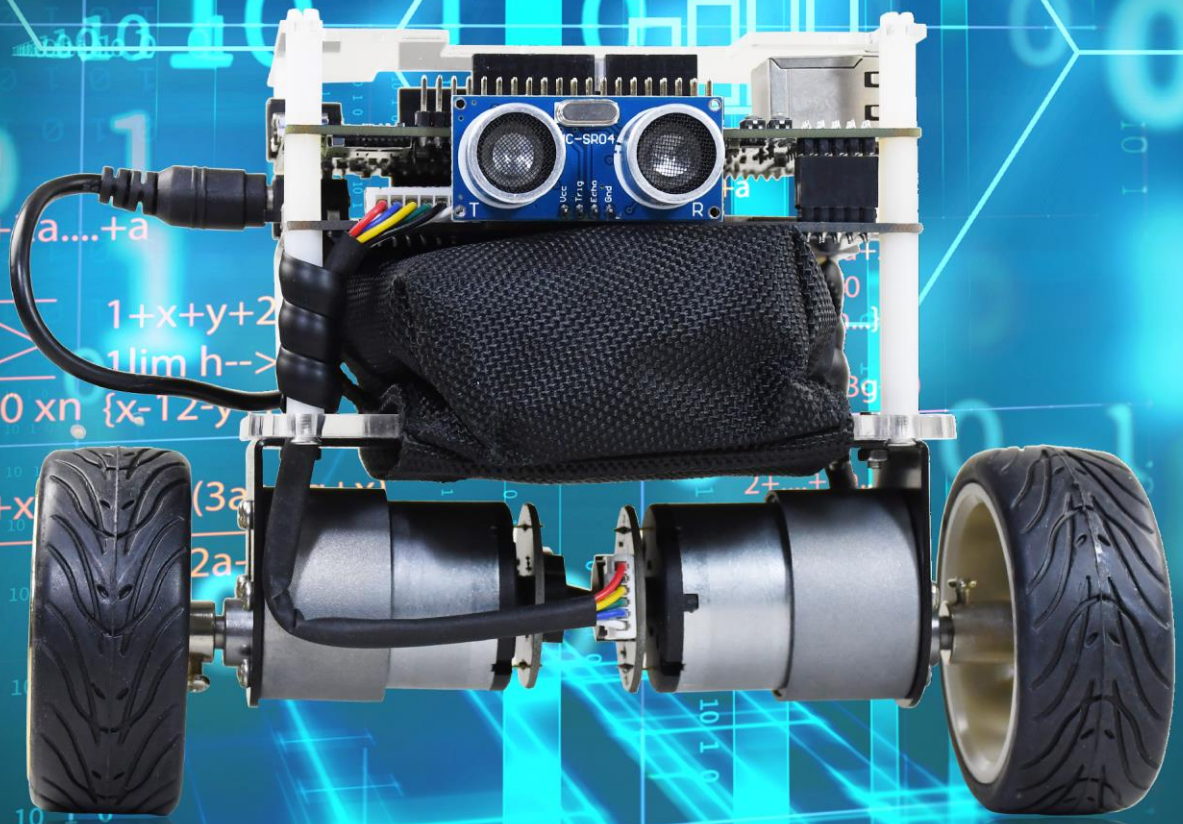
We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Self-Balancing Robot

User Guide



terasIC
www.terasic.com

Copyright © 2003-2018 Terasic Inc. All Rights Reserved.

CONTENTS

Chapter 1

Using the Self-Balancing Robot

1.1 Control the Motor

To keep the robot in vertical balance status, user need to control the rotation of the motor and make sure the accelerated rotation direction is reversed to the robot tilt direction. User need to learn how to control the rotation direction and speed of the motor.

This section describes how to control the motor forward rotation or reverse, also describes how to control the speed of the motor. As general FPGA IOs of DE10-Nano are unable to drive the motor, an extra motor drive chip or circuit is needed to drive the motors, the motor drive chip used on the robot is Toshiba TB6612FNG, which can be used to control two DC motors simultaneously. As shown in **Figure 1- 1**, the control signals--- IN1, IN2, PWM (control signals of motor A and B) and STBY are connected to FPGA, the control signals --O1 and O2 output to the motor. The way to control the rotation direction and speed of the motors is described as below.

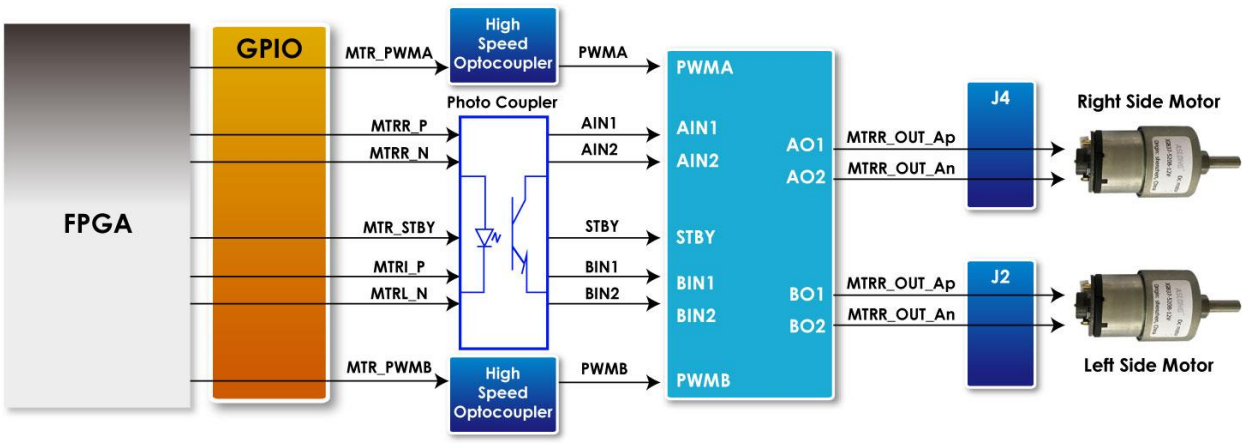


Figure 1- 1 Block Diagram of Motor Driver Control Function

Note: As there are some photo couplers between the FPGA and TB6612FNG, the logic of control

signals output from FPGA should be opposite to the logic described in the TB6612FNG datasheet.

■ Control Rotation Direction

Table 1- 1 lists the TB6612FNG control function.

Table 1- 1 TB6612FNG Control Function

FPGA Control Output				Driver Input				Driver Output		Modes description
MTRX_P	MTRX_N	MTR_PWMX	MTRX_STBY	IN1	IN2	PWM	STBY	O1	O2	--
0	0	1/0	0	1	1	1/0	1	0	0	Short brake
1	0	1	0	0	1	1	1	0	1	CCW
		0	0			0	1	0	0	Short brake
0	1	1	0	1	0	1	1	1	0	CW
		0	0			0	1	0	0	Short brake
1	1	1	0	0	0	1	1	OFF (High Impedance)		Stop
0/1	0/1	1/0	1	1/0	1/0	1/0	0	OFF (High Impedance)		Standby

- Control the logic value for IN1 and IN2 can drive the motor to counterclockwise rotation (IN1 = 0; IN2 = 1) or clockwise rotation (IN1 = 1; IN2 = 0).

- The motor will stop rotation when logics of both the two control signals (IN1 and IN2) are 0.

- STBY is equal to Chip Enable function. The motor will stop and wait for new command when STBY logic is 0.

In summary, user can easily change the motor rotation direction via controlling the IN1 and IN2 logic value.

■ Control Rotation Speed

The motor speed of the motors can be controlled by controlling the Duty Cycle of the PWM signal.

As shown in Figure 1- 2, the motor speed is faster while the PWM signal Duty Cycle is higher (Which means the ratio of the high logic positive pulse duration to the total pulse period is higher).

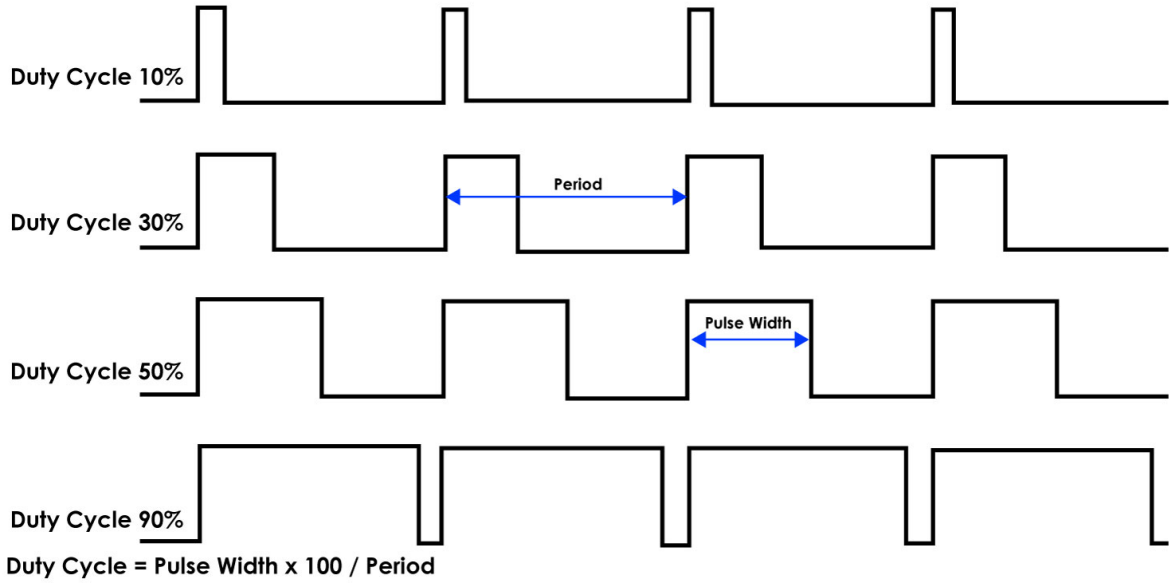


Figure 1- 2 The diagram of different Duty Cycle

The maximum PWM frequency that TB6612FNG provides is 100KHz. For the Self-Balancing Robot, the PWM frequency is set as 7.14KHz.

■ Example Description

Motor control IP TERASIC_DC_MOTOR_PWM.v is provided in the robot demo code. In this demo, it is packed as Qsys component and used to control the right and left motor. User can find the TERASIC_DC_MOTOR_PWM.v file in the robot system CD: Demonstrations\BAL_CAR_Nios_Code\IP\TERASIC_DC_MOTOR_PWM

● IP Symbol

Figure 1- 3 shows the symbol of TERASIC_DC_MOTOR_PWM.v and its block diagram. The main outputs are DC_MOTOR_IN1, DC_MOTOR_IN2 and PWM, others are Avalon interface signals. The DC_MOTOR_IN1 and DC_MOTOR_IN2 are the control signals that can control the motor rotation direction and stopping, which has been described in previous section. PWM control signal is responsible for controlling the motor speed.

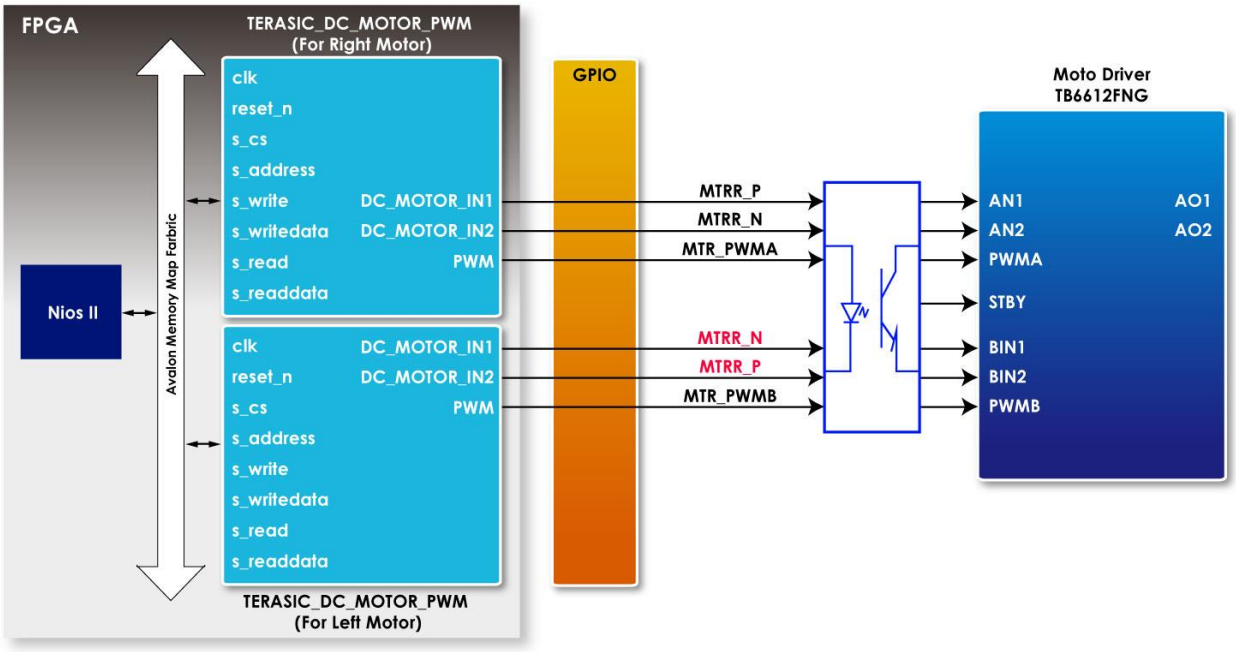


Figure 1- 3 TERASIC_DC_MOTOR_PWM.v symbol and block diagram

Table 1- 2 describes the Register Table of the motor control IP. Base Address 1~0 is the control register of PWM, Base Address 2 is the control register of motor brake control. User can read these registers value through Nios.

Table 1- 2 Register Table for TERASIC_DC_MOTOR_PWM.v IP

Reg Address	Bit Field	Type	Name	Description
Base Addr + 0	31:0	R/W	total_dur	PWM total duration value
Base Addr + 1	31:0	R/W	high_dur	PWM high duration value
Base Addr + 2	31:3	-	Unused	Unused bit
	2	R/W	motor_fast_decay	Motor brake control 1 for fast brake 0 for short brake
	1	R/W	motor_forward	Motor direction control : 1 for forward 0 for backward

	0	R/W	motor_go	Motor enable: 1 for start 0 for stop

IP Code Description

a. Control Rotation Direction

Below is Partial code for controlling the rotation direction:

```
always @(*)
begin
   if (motor_fast_decay)
   begin
      // fast decay
      if (motor_go)
      begin
         if (motor_forward)
            {DC_MOTOR_IN2, DC_MOTOR_IN1,PWM} <= {1'b1, 1'b0,PWM_OUT}; // forward
         else
            {DC_MOTOR_IN2, DC_MOTOR_IN1,PWM} <= {1'b0, 1'b1,PWM_OUT}; // reverse
      end
      else
         {DC_MOTOR_IN2, DC_MOTOR_IN1,PWM} <= {1'b1, 1'b1,1'b0};
   end
   else
   begin
      // slow decay
      if (motor_go)
      begin
         if (motor_forward)
            {DC_MOTOR_IN2, DC_MOTOR_IN1,PWM} <= {1'b1, 1'b0,PWM_OUT}; // forward
         else
            {DC_MOTOR_IN2, DC_MOTOR_IN1,PWM} <= {1'b0, 1'b1,PWM_OUT}; // reverse
      end
      else
```

```
        {DC_MOTOR_IN2, DC_MOTOR_IN1,PWM} <= {1'b0, 1'b0,1'b0};
    end
end
```

The register values of motor control register are related to DC_MOTOR_IN1 and DC_MOTOR_IN2 control signals, then user can control the motor rotation direction.

To drive the motors moving forward, user need set both *motor_go* and *motor_forward* as "1". Then the code ""DC_MOTOR_IN2, DC_MOTOR_IN1, PWM} <= {1'b1, 1'b0, PWM_OUT}; // forward" will be executed.

DC_MOTOR_IN1 outputs logic 0 and DC_MOTOR_IN2 outputs logic 1. During the logic transmit from FPGA to motor driver IC, they will be inverted to 1 and 0 respectively through the photo coupler. IN1 and IN2 of TB6612FNG will receive logic 1 and 0 respectively. As shown in Table 4-1, in this state, the motor will be clockwise rotation to drive the robot moving forward.

User can set *motor_fast_decay* as 1 and *motor_go* as 0 if they need a fast braking. Then, the below code will be executed.
DC_MOTOR_IN2, DC_MOTOR_IN1, PWM} <= {1'b1, 1'b1,1'b0};

Finally, the IN1 and IN2 logic will receive logic 0 and logic 0 respectively. As shown in Table 4-1, the motor is stopped.

The right and left motors on the robot are assembled opposite, so their rotation direction is opposite too. As we use the same IP to control both the right and left motor, the control signals in project top level file (DE10_Nano_Bal.v) are defined opposite, as described in code below:

```
Qsys u0 (
        //clock && reset
    .clk_clk                        (FPGA_CLK2_50),             // clk.clk)
    .reset_reset_n                  (1'b1),                     // reset.reset_n

        //right motor control
    .dc_motor_right_conduit_end_1_pwm           (MTR_PWMA),                                     //
dc_motor_right_conduit_end_1.pwm
    .dc_motor_right_conduit_end_1_motor_in1     (MTRR_P),           //          .motor_in1
    .dc_motor_right_conduit_end_1_motor_in2     (MTRR_N),           //          .motor_in2
```

```
        //left motor control
      .dc_motor_left_conduit_end_1_pwm          (MTR_PWMB),          //
dc_motor_left_conduit_end_1.pwm
      .dc_motor_left_conduit_end_1_motor_in1    (MTRL_N),            //          .motor_in1
      .dc_motor_left_conduit_end_1_motor_in2    (MTRL_P),
```

b. Control Rotation Speed

Below is partial code for controlling the rotation speed:

```
////////////////////////////////////////
// PWM
reg          PWM_OUT;
reg   [31:0]   total_dur;
reg   [31:0]   high_dur;
reg   [31:0]   tick;

always @ (posedge clk or negedge reset_n)
begin
   if (~reset_n)
   begin
      tick <= 1;
   end
   else if (tick >= total_dur)
   begin
      tick <= 1;
   end
   else
      tick <= tick + 1;
end
always @ (posedge clk)
begin
      PWM_OUT <= (tick <= high_dur)?1'b1:1'b0;
end
```

The *tick* register is the main counter, *total_dur* represents the total_dur register described in **Table 4-2**. When the *tick* value equals to *total_dur* setting value, the whole counter will be reset and

recounted. *PWM_OUT* represents one PWM cycle is finished. So, the longer the PWM cycle is, the larger the *total_dur* value will be. In this demo, the *total_dur* register is set to 7000 as default, it is one PWM cycle when the counter counts to 7000. The output PWM frequency is 7.14KHz (50MHz/7000).

As shown in **Figure 1- 4**, the motor speed depends on *high_dur* register. During one PWM cycle, when the *tick* value is less than *high_dur*, the PWM output is 1; otherwise, the PWM output is 0. So, the PWM Duty Cycle depends on the *high_dur*. Therefore, the larger the *high_dur* value is, the Duty Cycle will be larger and rotation speed will be faster.

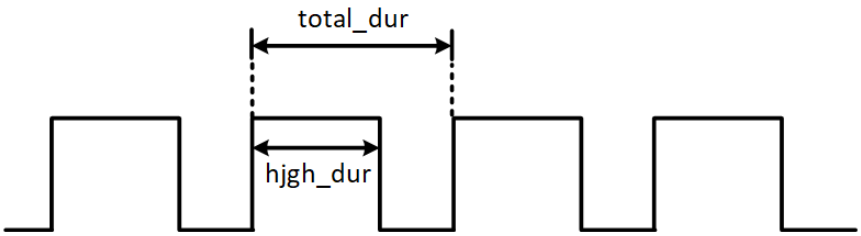


Figure 1- 4 The diagram of relationship between total_dur and high_dur in PWM

1.2 Detect the Motor speed and Direction

Section 1.1 introduces how to control motor speed and direction, this section will introduce how to use the Hall effect sensor and decoder on the motor to detect the motor speed and direction in real time.

■ Detection Principle

Figure 1- 5 shows the appearance of motor, there are two Hall effect sensors and one magnetic Rotor on the motor. When the motor rotates, it will drive the magnetic Rotor to pass through the Hall effect sensors, and then, the Hall effect sensor magnetic force will change and generate Hall effect voltage, a digital circuit will process the Hall effect voltage and output square wave pulse (See **Figure 1- 6**).

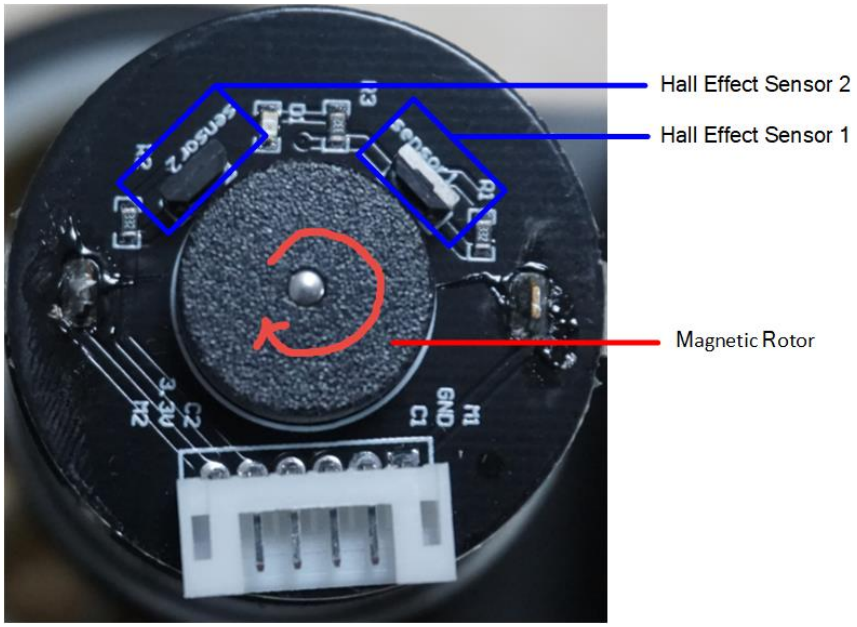


Figure 1- 5 the Hall effect sensor and magnetic Rotor on a motor

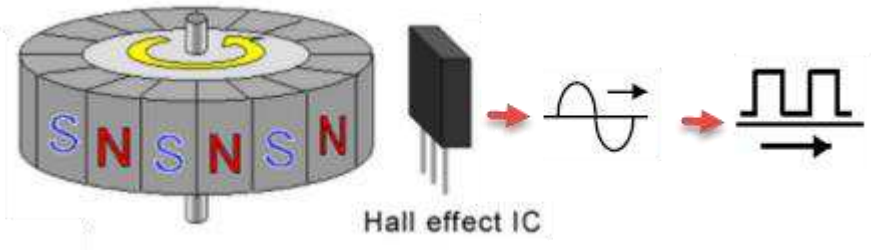


Figure 1- 6 Hall effect sensor and the square wave pulse

The two Hall effect sensors output two waves in different phases (Phase A and Phase B) as the two sensor locations are different (See **Figure 1- 7**). When magnetic Rotor rotates, the first sensed sensor will output wave first, and the other sensor output will delay. That is why the two waves have different phases. So, we can know the motor rotation direction according which sensor wave phase is ahead. **Figure 1- 7** is the motor clockwise rotation status. In addition, we can also calculate the motor speed according to the pulse number. Over a period of time, the faster the motor rotates, the more pulses it generates.

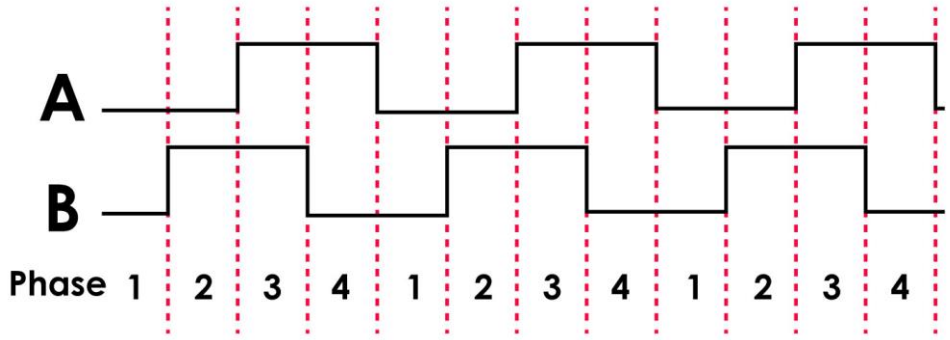


Figure 1- 7 Hall effect decoder output A B Phase waves

Figure 1- 8 shows the motor phase pins connecting to FPGA. We can obtain the motor speed and direction in real time by writing code to just detect the phase and pulse number of the two signals (MTRR_IN_PA, MTRR_IN_PB).

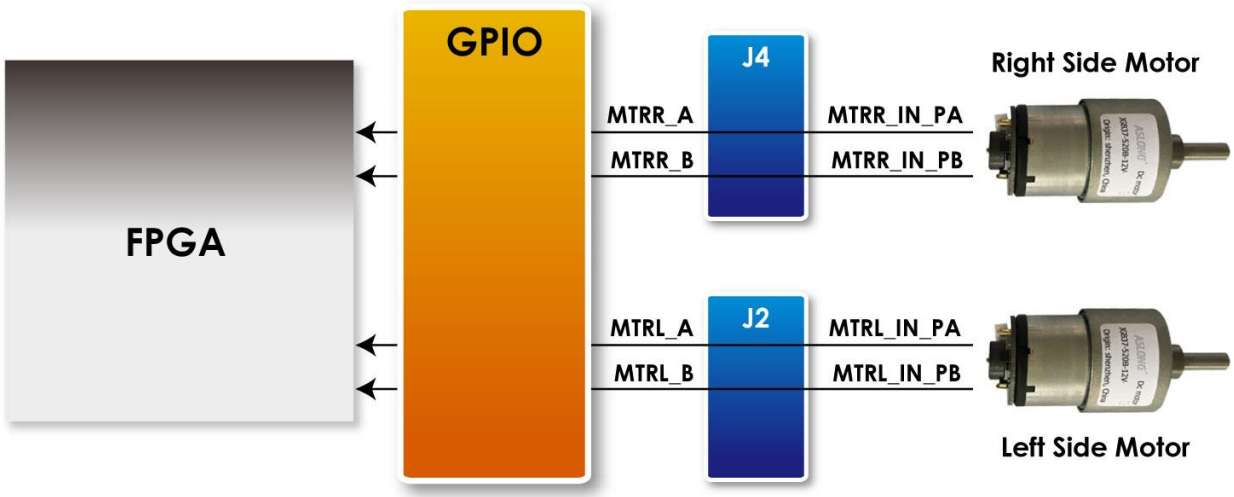


Figure 1- 8 The motor phase pins connect to FPGA

■ Example Description

We do provide a Qsys IP in the Self-Balancing Robot demo for users to obtain the motor speed and direction, the IP can be found in folder:

\Demonstrations\BAL_CAR_Nios_Code\IP\motor_measure\motor_measure.v

● IP Symbol

Figure 1- 9 is the symbol and the system block diagram of motor_measure.v. Here we only draw the detecting diagram of the motor on the right, and the motor on the left also has the same module to detect the speed.

The module phase_AB[1:0] is used to connect the motor to receive the waves from the Hall effect sensor. The module can detect and obtain the motor speed and direction and save the data in the module register. CPU can read the data from Avalon bus.

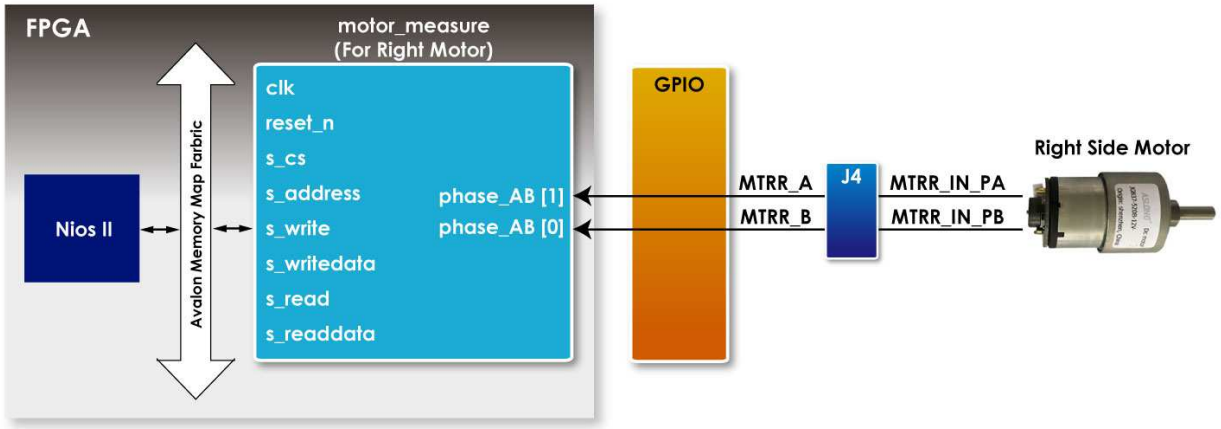


Figure 1- 9 The motor_measure.v module and the system block diagram(for the motor on the right)

● Register Table

Table 1- 3 is the IP register table. The *Counter* register in address "Base Addr+0" will counts the motor pulse number. The system can detect how much the motor rotates according to this register, and the positive number means clockwise direction, the negative number means counterclockwise direction. *Counter* is read only register, the CPU can only read the Counter value. The *Counter* register in address "Base Addr+2" is for write. The *count_en* register in address "Base Addr+1" is for controlling the two *Counter*s. When *count_en* is set to 1, the *Counter* register will start to count.

Table 1- 3 motor_measure.v IP Register

Reg Address	Bit Filed	Type	Name	Description
Base Addr + 0	31:16	RO	Unuse	-

	15:0	RO	Counter (For Read)	Read Counter value for Motor output pules
Base Addr + 1	31:30	RW	Unuse	-
	1	RW	count_en	Enable Motor pulse counter
Base Addr + 2	31:16	WO	Unused	-
	15:0	WO	Couter (For Write)	Set Counter vaule

● IP Code Description

There is a submodule code TERAISC_AB_DECODER.v in motor_measure.v IP, this submodule can detect phase A and phase B signals from the motor, and according the different phases, the submodule can figure out whether the rotation direction is clockwise or counterclockwise. The direction result will output to DO_DIRECT port and the rotation pulse will output to DO_PULSE port. Then these two signal will pass to the motor_measure.v IP.

```
TERAISC_AB_DECODER u_decoder
(
    .DI_SYSCLK(clk),
    .DI_PHASE_A(phase_AB[0]),
    .DI_PHASE_B(phase_AB[1]),
    .DO_PULSE(conter_pulse),
    .DO_DIRECT(direction)
);
```

Please refer to below code list in below, the motor_measure IP has a 16bit *Counter* (initial value is 16'h8000) register, which is enabled only if the "*count_en*" register is set to 1. In the beging When the motor rotates clockwise (direction = 1), The *Counter* register will count from the initial value and increase by the number of pulses returned from the motor; If the motor rotates counterclockwise, the *Counter* register also will decrement with the number of pulses returned by the motor. The system will periodically read the value of this *Counter* register to acquire the current

number of rotation speed of the motor.

```
always @( posedge clk)
begin
    if(s_cs && s_write && s_address==`CNT_WRITE)
            counter<=s_writedata[15:0];
    else if(count_en && conter_pulse )
    begin
        if(direction)
        begin
            if(counter<16'hffff)
                counter<=counter+1;
        end
        else if(!direction)
        begin
             if(counter>0)
                counter<=counter-1;
        end
        else
            counter<=0;
    end
end
```

Users can refer to the Nios version demo file Motor.cpp (which location is \Demonstrations\BAL_CAR_Nios_Code\software\DE10_Nano_bal) for the steps of system reading counter. During the Self-Balancing Robot system initialization process, the *count_en* will be set to 1, after this, the system will read the counter every 10ms. Every time after the reading, the system will set the counter back to the initial value(16'h8000) and wait for the next reading time. The system will use the latest counter value to minus the initial value (16'h8000), the result will be a positive number if the motor direction is clockwise, the result will be a negative number if the motor direction is counterclockwise.

The counter values of the two motors will finally transfer to the balancing PID algorithm.

1.3 Get the tilt angle of Robot

This section describes how to obtain the tilt angle of the Self-Balancing Robot, and how to get an angularity correction to keep the robot balance.

■ How to get the tilt angle of the body

The idealized state of the balance car is to maintain a vertical 90-degree angle to the ground. However, there are only two wheels support the body, which is more likely to lean forward or backward. It actually exists an angle of θ showing as **Figure 1- 10**. Our aim is to read out this angle and feed it back to the balance system for controlling the motor rotating in the opposite direction. In this way, it will make the tilt of the Angle to become the ideal 0 degree as a correction.

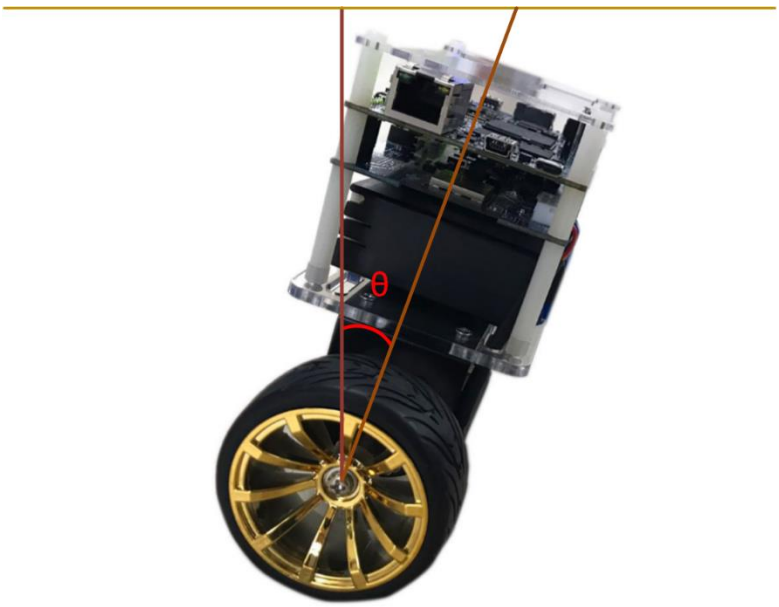


Figure 1- 10 The tilted angle of the balance car

To obtain the tilt angle of the body, the Motion Tracking device MPU-6500 on the robot will be used. The single-chip MPU-6500 integrates a 3-axis accelerometer, and a 3-axis gyroscope. It is able to read out the acceleration of there-axis (/g) from accelerator, and angular speed of three axes (/Sec) from the gyroscope. The balance system will use both of these two sensors to compute the tilt angle.

First of all, we need to obtain the status of the XYZ coordinates of the MPU-6500 in the robot.

From **Figure 1- 11**, It can be seen if the balance of the car is tilted forward or backward, corresponding changes resulting the acceleration of the X-axis and Y-axis. Meanwhile, the angular speed of the Y-axis will also change.

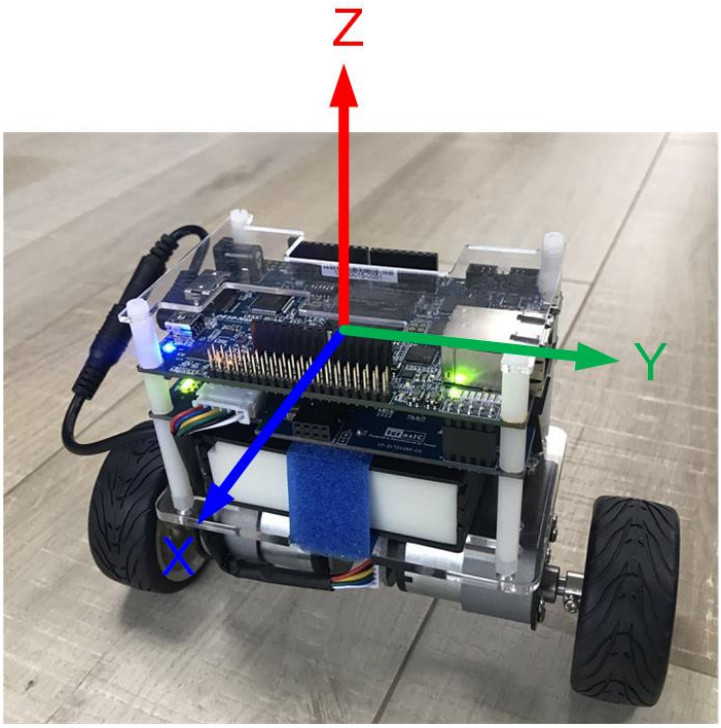


Figure 1- 11 Status of the XYZ coordinates on the mpu-6500

Continue to introduce how to calculate the tilt angle of the balance car from the accelerometer. **Figure 1- 12**, ignore the horizontal acceleration of the car, there is a vertical angle θ when body tilts forward or backward. G is for the acceleration of gravity. Resolve g vector into X and Z directions, gx and gz are coordinate components for X-axis and Z-axis respectively, θ is the tangent angle of gx or gz. Read out the values of gx & gz from the accelerometer in the MPU-6500. Get the degree of angle θ by using the function θ = arctan(gx/gz).

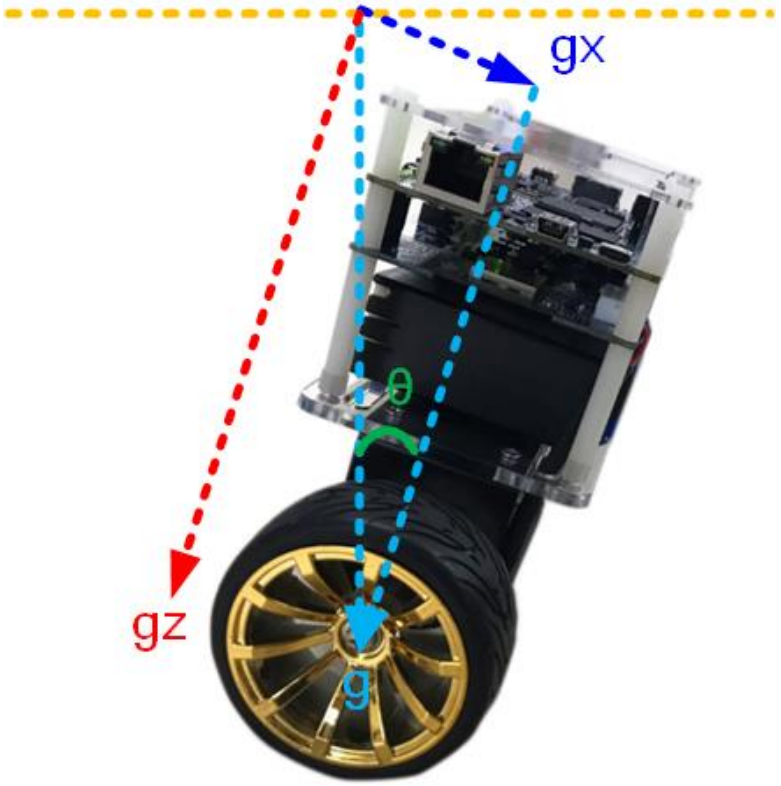


Figure 1- 12 The tilted angle of the self-balance Robot

Besides, it can also get the body tilted angle from Y-axis angular rate by using the gyroscope in the MPU-6500. See below **Figure 1- 13**, when the body tilts, the angular rate changes as well. Obtain the angle in Y-axis via doing the integral calculation on the angular rate.

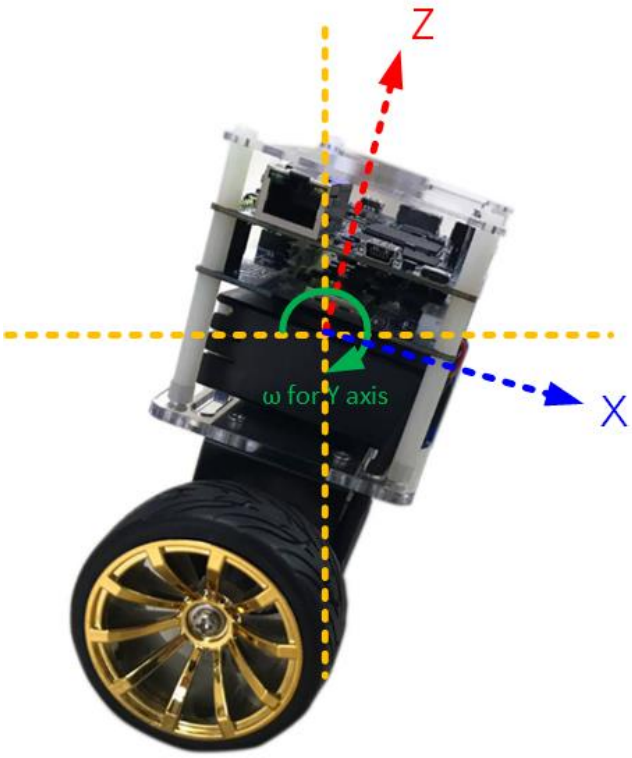


Figure 1- 13 The tilted angle of the self-balance Robot

There is an error in the angles calculated by the above two methods. When the accelerometer reads the angle, the acquired value error will increase when the outside is disturbed. Use the gyroscope's angular speed to integrate the acquired angle, due to the integral calculation will accumulate the error, and increase with time, the error will become bigger and bigger. If you use the angle with larger error to balance the system, it will be very difficult to stabilize the robot. Therefore, so a method on Angle error correction is required.

It is common to use Kalman Filter as a method, which adopts the data fusion of two sensors (gyro and accelerometer) to get a more precise angle.

Figure 1- 14 shows the comparison of the original tilt Angle (blue) and the Angle (orange) by Kalman Filter. From the figure, you can see that the change in angle is large different for unfiltered angle (blue). By working with these data, the balance of the car body could not be stable. However, the varying amplitude for the orange Angle after filtering is significantly reduced.

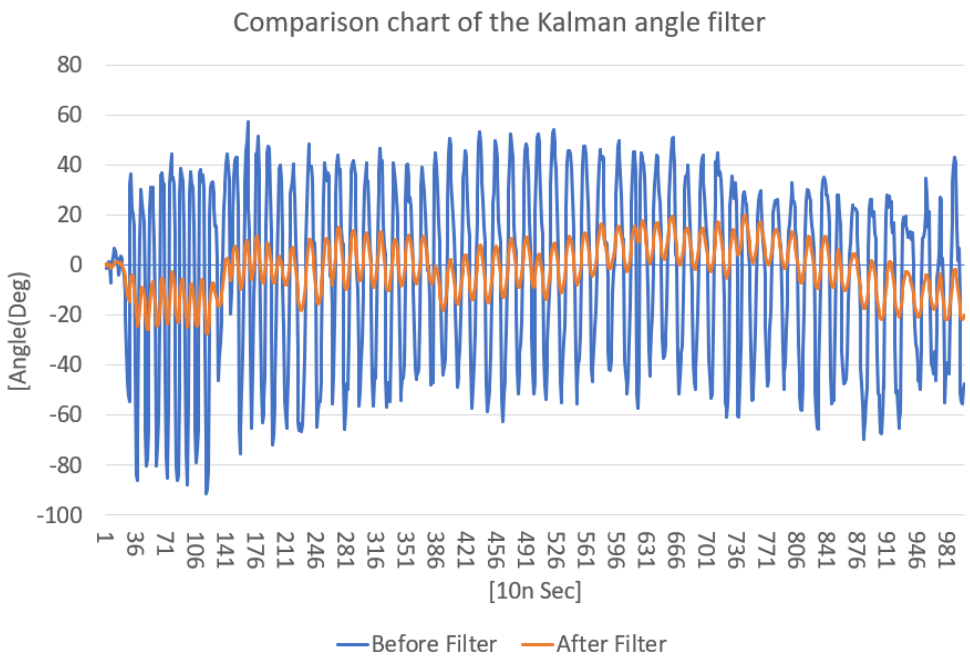


Figure 1- 14 Comparison chart of the Kalman angle filter

■ MPU-6500 Operation

FPGA uses I2C or SPI interface to control the MPU-6500, In our demo, we use the I2C interface to read the register of the MPU-6500 with Salve Address 7'b1011001. The data registers of XYZ axis accelerator and gyroscope locate in the range of 3B(Hex) ~ 48(Hex). It requires the initial operation on starting up. For more details about the MPU-6500 and Register map, please refer to the CD\Datasheet\Sensor\. And refer to the MPU.cpp & MPU.h provided in CD\Demonstrations\BAL_CAR_Nios_Code\software\DE10_Nano_bal\ for control code.

■ Example Descriptions

We provide the Nios II demo for balance car by using the Open-core I2C module in Qsys. This module allows Nios II to access the mpu-6500 through the I2C interface.

The function for obtaining the tilted angle is provided in path: \Demonstrations\BAL_CAR_Nios_Code\software\DE10_Nano_bal\main.cpp, the main codes are as following:

```
/*****************************************************************************
```

```
Function        : Get Angle value (Kalman filter)
parameter       :
return value :
*************************************************************************/
void Get_Angle(void)
 {
                int16_t ax, ay, az, gx, gy, gz;
                mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
                Gyro_Balance=-gy;
                x_angle=atan2(ax,az)*180/PI;
                gy=gy/16.4;
                Angle_Balance=kalman.getAngle(x_angle,-gy);
                Gyro_Turn=gz;
 }
```

First, read out the acceleration of X-, Y-, & Z- Axes and angular rate of gyroscope.

mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

Because the polarity of the angular rate read out is opposite to the actual value, it is necessary to do a negative operation.

Gyro_Balance=-gy;

As described above, the tilted angle is obtained by computing the acceleration of X-axis & Y-axis. Use function atan2() to obtain the angle.

*x_angle=atan2(ax,az)*180/PI;*

As described above again, we can also get the tiled angle from Y- axis argular rate of gyroscope. However, the read-out value needs to divide by precision.

gy=gy/16.4;

Here, 16.4 is the Sensitivity Scale Factor for the gyroscope. Why use this value are introduced in the following descriptions.

The data register of the MPU-6500 is a 16-bit register, as the MSB is sign bit, the output range of the data register is -7FFF~7FFF, same as -32767~32767 in decimal format: See the diagram as below, if the full-scale range of ±2000 degrees/sec is selected, that means -32767 corresponding to -2000(°/s) and 32767 is corresponding to 2000(°/s). While reading out the value of gyroscope as 1000, the corresponding angular rate can be computed as below: 32767/2000 =1000/x; As x = 16.4 (°/s), The corresponding Sensitivity Scale Factor is 16.4 LSB/(°/s) in the manual. It is able to compute the angular rate as the same way if selecting the other ranges.

3.1 Gyroscope Specifications

Typical Operating Circuit of section 4.2, VDD = 1.8V, VDDIO = 1.8V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	3
	FS_SEL=1		±500		°/s	3
	FS_SEL=2		±1000		°/s	3
	FS_SEL=3		±2000		°/s	3
Gyroscope ADC Word Length			16		bits	3
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	3
	FS_SEL=1		65.5		LSB/(°/s)	3
	FS_SEL=2		32.8		LSB/(°/s)	3
	FS_SEL=3		16.4		LSB/(°/s)	3
Sensitivity Scale Factor Tolerance	25°C		±3		%	2
Sensitivity Scale Factor Variation Over Temperature	-40°C to +85°C		±4		%	1
Nonlinearity	Best fit straight line; 25°C		±0.1		%	1
Cross Axis Sensitivity			±2		%	1

Figure 1- 15 MPU-6500 datasheet-Gyroscope Specifications

Finally, send the Angle value obtained from the accelerometer and gyroscope the kalman filter function, to obtain the Angle of the car body with a smaller error.

Angle_Balance=kalman.getAngle(x_angle,-gy);

When dealing with the turning of the body, the system needs to refer to the angular speed of the z axis.

Gyro_Turn=gz;

The parameters above will be provided to system for balance PID(Proportional–Integral–Derivative) controlling to feed back the actual condition of the body to achieve a balanced state.

1.4 Detect Obstacle Distance

This section describes how to detect the obstacle distance in front of the robot by using the Ultrasonic module.

■ Principle

As shown in **Figure 1- 16**, the robot assembled HC-SR04 Ultrasonic module. Besides VCC and GND pin, the module is controlled mainly by TRIG and ECHO signal.

The detection process is described as below:

- To start detecting the distance, input High-level logic signal to the TRIG I/O for at least 10us.
- The Module automatically sends eight 40 kHz and detect where there is pulse signal return.
- The echo port will automatically output a high-level logic when detecting a rebound signal. The duration of the high-level logic is the timing for the ultrasonic wave to be reflected back to the module after the ultrasonic wave is emitted from the module.

Figure 1- 16 Ultrasonic module working diagram

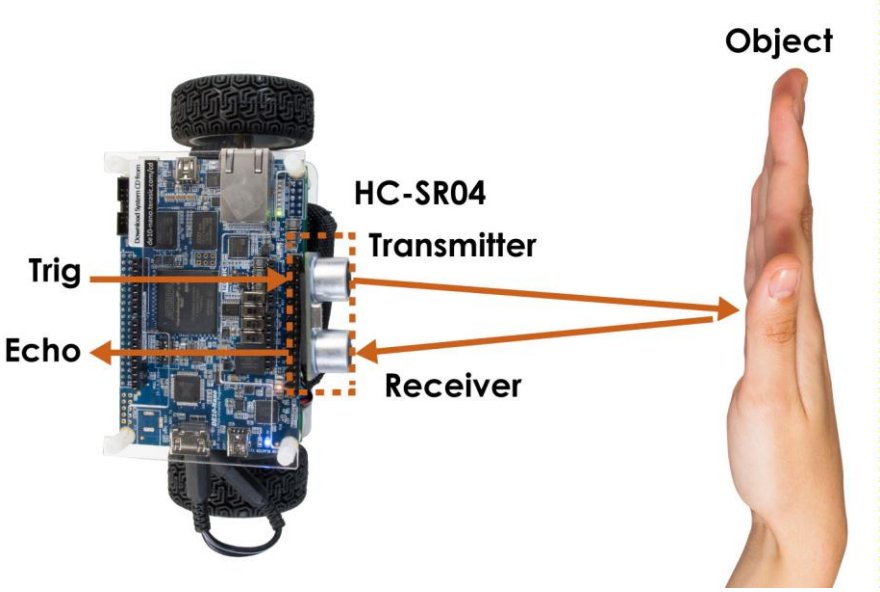


Figure 1- 17 Ultrasonic module working diagram

■ Calculate Distance

Obstacle distance= (high level logic time × Speed of sound (340m/s) / 2

The obstacle distance is calculated by the formula above, please note that the distance unit is meter.

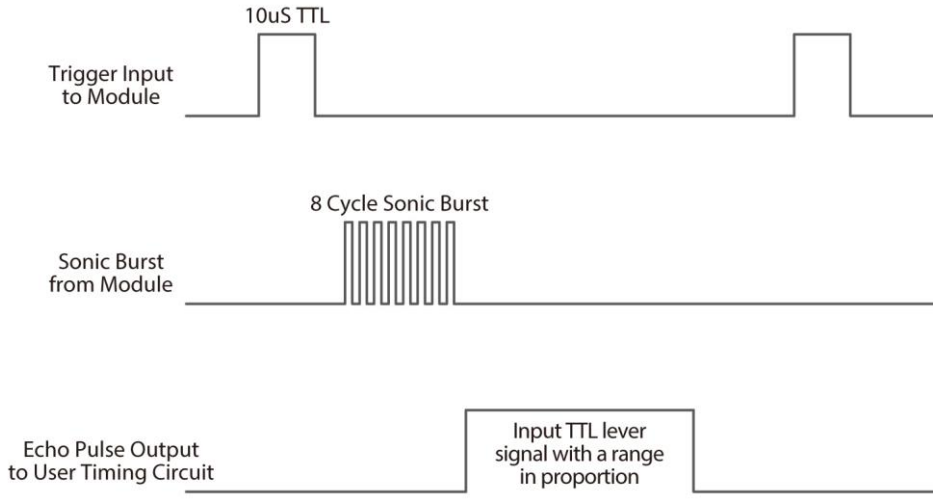


Figure 1- 18 Ultrasonic Module Timing Diagram

■ Demonstration Description

The demo provides Qsys IP which can read the obstacle distance from the Ultrasonic module, it's

located in \Demonstrations\BAL_CAR_Nios_Code\IP\sonic_distance\sonic_distance.v

IP symbol

As shown in Figure 1- 19, the IP controls the TRIG pin, drives the module to start to detect obstacle, monitors whether there is reflection signal on ECHO pin and calculate the duration of the high level logic signal, then CPU will can read the data.

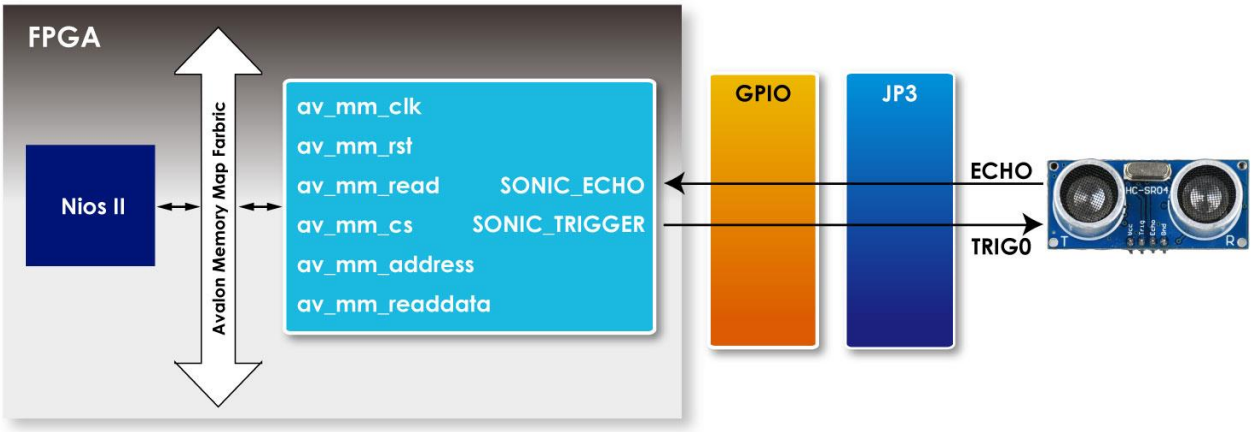


Figure 1- 19 The IP symbol and block diagram

Register Table

Table 1- 4 shows the register table of the IP. measure_value register stores the ultrasonic transmission time that the module detects the obstacle distance each time.

Table 1- 4 Register table of the IP

Reg Address	Bit Filed	Type	Name	Description
Base Addr + 0	31:22	RO	Unuse	--
	21:0	RO	measure_value	Sonic wave propagation time

IP Code Description