RX and RL78 Series

Rev.1.00

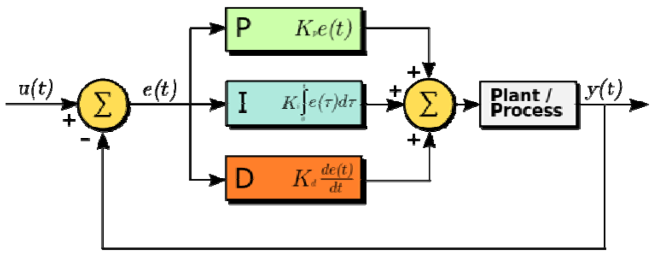
PID Controller Software for RX and RL78

Mar 31, 2013

Introduction

PID – **P**roportional-**I**ntegral-**D**erivative Controller is the most commonly used feedback controller, and it is widely used in industrial control system. A PID controller calculates an “error” value as the difference between process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control intputs.

The PID controller can be applied to any system that requires close-loop control. For example, speed control, temperature control, brightness control, etc.



A PID Controller block diagram

Where:

u(t): Controller output,

K_p:	Proportional gain
K_i:	Integral gain
K_d:	Derivative gain
e:	Error = SP-PV
t:	Time or instantaneous time (the present)
τ:	Variable of integration

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

A PID Algorithm

This application note introduced a standard & optimized PID control algorithm for RX and RL78 MCU Benchmark of the algorithm is also described.

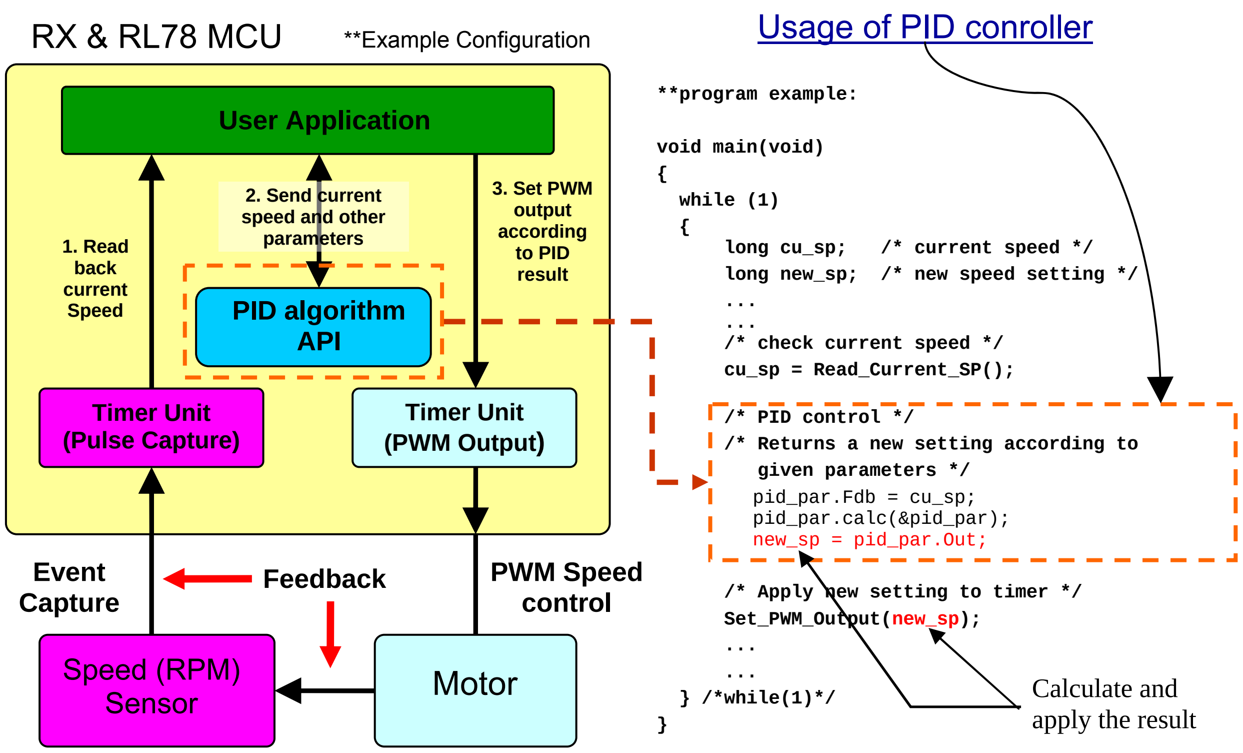
Target Device

RX600, RL78

Contents

1. Configuration

PID Controller is a software control algorithm which being widely adopted in close loop control system. Diagram below shows the software configuration for an application.



```
**program example:

void main(void)
{
  while (1)
  {
      long cu_sp;   /* current speed */
      long new_sp;  /* new speed setting */
      ...
      ...
      /* check current speed */
      cu_sp = Read_Current_SP();

      /* PID control */
      /* Returns a new setting according to
         given parameters */
        pid_par.Fdb = cu_sp;
        pid_par.calc(&pid_par);
        new_sp = pid_par.Out;

      /* Apply new setting to timer */
      Set_PWM_Output(new_sp);
      ...
      ...
  } /*while(1)*/
}
```

A floating point PID controller is demonstrated with RX simulator. For RX, HEW with RX simulator is used for simulation.

A fix point (integer) PID controller is demonstrated with RL78 simulator. For RL78, CubeSuite+ with RL78 Simulator for simulation.

There are two key files for PID controller, PID.h and PID.c:

PID.h:	Consists declaration of PID controller data structure and related function.
PID.c:	Consists 2 functions of PID controller, PID_Init() and PID_Control(). It is for initialization and calculation.

Usage example of PID controller can be found in main.c file.

Detail of PID controller explanation in next chapter.

Note:

The PID controller software provides a standard API and algorithm for RX and RL78 MCU. Actual parameter tuning of PID controller in the application program is needed.

2. PID Controller Data Structure & Standard API

This section describes the PID controller components as follow:

	floating point	fix point
Data Structure	fPID_PAR	PID_PAR
API	fPID_Init()	PID_Init()
	fPID_Control()	PID_Control()

It is easy differentiate floating point operation and fix point operation by ‘f’ at the 1st character. Where ‘f’ means floating point, without ‘f’ is fix point.

In this application note, RX use floating point, and RL78 use fix point to illustrate how to use the PID controller software.

2.1 PID Controller Data Structure

To create a standard API for PID function in RX (floating point) and RL78 (fix point) core MCU, a standard structure named “fPID_PAR”, “PID_PAR” for PID function is created in PID.h file and shown as below. As RX MCU contains hardware FPU, float data type is defined for all variables and operation will be in floating point. For RL78, it use long data type for all variables and operation will be in fix point.

There are mainly three types of data consisted in the Data structure:

1. Input/Output:	Target value, current value and calculated value.
2. Parameter:	Constants for calculation, limit comparison. Need tuning, specially the gains.
3. Variable:	For Calculation.

Floating point PID data structure:

```
typedef struct fPID_PAR {
      _fl Ref;                   // Input: Reference input
      _fl Fdb;                   // Input: Feedback input
      _fl Err;                   // Variable: Error
      _fl Kp;                    // Parameter: Proportional gain
      _fl Up;                    // Variable: Proportional output
      _fl Ui;                    // Variable: Integral output
      _fl Ud;                    // Variable: Derivative output
      _fl OutPreSat;             // Variable: Pre-saturated output
      _fl IntegralMax;           // Parameter : Integral maximum value
      _fl IntegralMin;           // Parameter : Integral minimum value
      _fl IntegralErr;           // Variable: Saturated difference
      _fl OutMax;                // Parameter: Maximum output
      _fl OutMin;                // Parameter: Minimum output
      _fl Out;                   // Output: PID output
      _fl SatErr;                // Variable: Saturated difference
      _fl Ki;                    // Parameter: Integral gain
      _fl Kc;                    // Parameter: Integral correction gain
      _fl Kd;                    // Parameter: Derivative gain
      _fl Up1;                   // History: Previous proportional output
      _fl Dt;                    // Parameter: time constant
      void (*calc)();            // Pointer to calculation function
} fPID_PAR;
```

Fix point PID data structure:

```
typedef struct PID_PAR {
       _l Ref;                   // Input: Reference input
       _l Fdb;                   // Input: Feedback input
       _l Err;                   // Variable: Error
       _l Kp;                    // Parameter: Proportional gain
       _l Up;                    // Variable: Proportional output
       _l Ui;                    // Variable: Integral output
       _l Ud;                    // Variable: Derivative output
       _l OutPreSat;             // Variable: Pre-saturated output
       _l IntegralMax;           // Parameter : Integral maximum value
       _l IntegralMin;           // Parameter : Integral minimum value
       _l IntegralErr;           // Variable: Saturated difference
       _l OutMax;                // Parameter: Maximum output
       _l OutMin;                // Parameter: Minimum output
       _l Out;                   // Output: PID output
       _l SatErr;                // Variable: Saturated difference
       _l Ki;                    // Parameter: Integral gain
       _l Kc;                    // Parameter: Integral correction gain
       _l Kd;                    // Parameter: Derivative gain
       _l Up1;                   // History: Previous proportional output
       _l Dt;                    // Parameter: time constant
       void (*calc)();           // Pointer to calculation function
} PID_PAR;
```

Inside the PID Controller data structure, there are 20 variables defined in either float/long data type plus 1 function pointer which totally occupy 84 bytes. To use this standard PID structure, it must firstly initialize the structure. The following section describes how to initialize the PID structure.

2.2 PID Controller Standard API

There are mainly 2-sets (floating point and fix point) of API for PID Controller:

fPID_Init(), PID_Init():	Data structure initialization
fPID_Control(), PID_Control():	PID controller main calculation routine

2.2.1 PID Controller Initialization – fPID_Init, PID_Init

It is important that to initialize the PID controller before to use it. Improper initialize PID controller resulting damage to the system.

For parameters, there are four tuning constants Kp: Proportional gain, Ki: Integral gain, Kc: Integral correction gain, and Kd: Derivative gain. These gains controls the PID output characteristic, these parameters have to be tuned properly. The default values of the gains are 1. The rest of parameters are limit of PID controller. User has to give the upper/lower limit for the calculated output. It is more for safety consideration.

For variables, as they are for calculation purpose, it should be set to ‘0’ as initialize value. For inputs, ‘Ref’ is the target value, where ‘Fdb’ is the current value. For output, ‘Out’ is the calculated value. Besides, other than the variables, the function pointer is also needed to assign a PID calculation function into it. In the example, the address of the fPID_Control function has been put into the function pointer.

Example below shows the initialization of floating point PID controller:

```
/*************************************************************/
/* Description:     Floating point PID Control for RX       */
/*                  Initialization                          */
/*************************************************************/
void fPID_Init(void)
{
      /* Initial Value */
      pid_par.Ref = Target_dist;       // Input: Reference input
      pid_par.Fdb = 0;                 // Input: Feedback input
      pid_par.Err = 0;                 // Variable: Error
      pid_par.Kp = 1;                  // Parameter: Proportional gain
      pid_par.Up = 0;                  // Variable: Proportional output
      pid_par.Ui = 0;                  // Variable: Integral output
      pid_par.Ud = 0;                  // Variable: Derivative output
      pid_par.OutPreSat = 0;           // Variable: Pre-saturated output
      pid_par.IntegralMax=99999;       // Parameter : Maximum Integral value
      pid_par.IntegralMin=-99999;      // Parameter : Maximum Integral value
      pid_par.IntegralErr = 0;         // Variable: Integral error accumulator
      pid_par.OutMax = 99999;          // Parameter: Maximum output
      pid_par.OutMin = -99999;         // Parameter: Minimum output
      pid_par.Out = 0;                 // Output: PID output
      pid_par.SatErr = 0;              // Variable: Saturated difference
      pid_par.Ki = 1;                  // Parameter: Integral gain
      pid_par.Kc = 1;                  // Parameter: Integral correction gain
      pid_par.Kd = 1;                  // Parameter: Derivative gain
      pid_par.Up1 = 0;                 // History: Previous proportional output
      pid_par.Dt = 0.01;               // time constant
      pid_par.calc = (&fPID_Control);  // Pointer to calculation function
}
```

Example below shows the initialization of fix point PID controller, note that all values should be integers (long data type).

```
/*************************************************************/
/* Description:     Fix point PID Control for RL78          */
/*                  Initialization                          */
/*************************************************************/
void PID_Init(void)
{
       /* Initial Value */
       pid_par.Ref = Target_dist;        // Input: Reference input
       pid_par.Fdb = 0;                  // Input: Feedback input
       pid_par.Err = 0;                  // Variable: Error
       pid_par.Kp = 1;                   // Parameter: Proportional gain
       pid_par.Up = 0;                   // Variable: Proportional output
       pid_par.Ui = 0;                   // Variable: Integral output
       pid_par.Ud = 0;                   // Variable: Derivative output
       pid_par.OutPreSat = 0;            // Variable: Pre-saturated output
       pid_par.IntegralMax=MAX_LIMIT;    // Parameter : Maximum Integral value
       pid_par.IntegralMin=MIN_LIMIT;    // Parameter : Maximum Integral value
       pid_par.IntegralErr = 0;          // Variable: Integral erro accumulator
       pid_par.OutMax = MAX_LIMIT;       // Parameter: Maximum output
       pid_par.OutMin = MIN_LIMIT;       // Parameter: Minimum output
       pid_par.Out = 0;                  // Output: PID output
       pid_par.SatErr = 0;               // Variable: Saturated difference
       pid_par.Ki = 1;                   // Parameter: Integral gain
       pid_par.Kc = 1;                   // Parameter: Integral correction gain
       pid_par.Kd = 1;                   // Parameter: Derivative gain
       pid_par.Up1 = 0;                  // History: Previous proportional output
       pid_par.Dt = 1;                   // time constant (unit)
       pid_par.calc = (&PID_Control);    // Pointer to calculation function
}
```

2.2.2 PID Control– fPID_Control, PID_Control

The theory of PID controller can be easily found on Internet. Basically it is the sum of Proportional terms, Integral terms, and Derivative terms.

Equation of PID control often describes as below:

pid_out = (Pe * Kp) + (Ie * Ki) + (De * Kd) Calculation perform in a fixed period (dt)

Where:

Proportional terms	Pe	Current error value, (Set-point - present value)
	Kp	Proportional gain, tuning parameter
Integral terms	Ie	Sum of instantaneous error, (accumulation of Pe*dt)
	Ki	Integral gain, tuning parameter
Derivative terms	De	Slope of error over time, ((Pe - pervious Pe)/dt)
	Kd	Derivative gain, tuning parameter

And, pseudocode for PID control example is found:

```
/* Pseudocode */
previous_error = 0
integral = 0
start:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
/* Pseudocode end */
```

Based on the pseudocode, a PID Controller function is made as below. Floating point and Fix point PID controller are basically use the same source code, but the variables are in different data types, float vs long. For floating point calculation, fPID_Control() should be used. For fix point calculation, PID_Control should be used.

In addition to the pseudocode, Integral terms MAX/MIN are checked, as well as calculated output MAX/MIN. As mentioned in pervious section, the maximum and minimum values are for safety consideration, which would limited the output in order to protect the system.

User should set this value carefully according to the system specification.

Floating point PID controller:

```
/************************************************************/
/* Description:     Fix point PID Control for RX            */
/************************************************************/
/* Input: address of pid_parameters:                        */
/*         - Constant (float) Kp, Ki, Kd                    */
/*         - Variable (float) Up, Ui, Ud                    */
/*         address of calculated (fix) result               */
/*                additional parameters (TBD)               */
/* Output:    pid->Err:'0' Calculation OK                    */
/*            pid->Err:'-1' Calculation outside limit        */
/************************************************************/
/* Remark: This function should be call periodly            */
/************************************************************/
void fPID_Control(fPID_PAR *pid)
{
       float cal_result;
       float present_err;

       /*** SatErr, Kc not used ***/
       /* present error */
       present_err = pid->Ref - pid->Fdb;

       /* Integral term calculation */
       /* Add Integral terms (Ui * Ki) */
       cal_result= pid->Ki * present_err;

       if (cal_result> pid->IntegralMax)
       {
              cal_result= pid->IntegralMax;
       }
       else if ( cal_result< pid->IntegralMin)
       {
              cal_result= pid->IntegralMin;
       }

       /* Add Proportional terms (Pe * Kp) */
       cal_result += pid->Kp * present_err;

       /* Sum of instantaneous error */
       pid->Ui += present_err * pid->Dt;

       /* Slope of error over time */
       pid->Ud = (present_err - pid->Up1)/(pid->Dt);

       /* Add Derivative terms (De * Kd) */
       cal_result += pid->Kd * pid->Ud;

       pid->Up1 = present_err;

       /* Result check here */
       if (cal_result > pid->OutMax)
       {
              pid->Out = pid->OutMax;
              pid->Err = -1;
       }
       else if (cal_result < pid->OutMin)
       {
              pid->Out = pid->OutMin;
              pid->Err = -1;
       }
       else
       {
              pid->Out = cal_result;
              pid->Err = 0;
       }
}
```

Fix point PID controller:

```
/************************************************************/
/* Description:     Fix point PID Control for RL78           */
/************************************************************/
/* Input: address of pid_parameters:                         */
/*        - Constant (long) Kp, Ki, Kd                       */
/*        - Variable (long) Up, Ui, Ud                       */
/*        address of calculated (fix) result                 */
/*               additional parameters (TBD)                 */
/* Output:    pid->Err:'0' Calculation OK                     */
/*            pid->Err:'-1' Calculation outside limit         */
/************************************************************/
/* Remark: This function should be call periodly             */
/************************************************************/
void PID_Control(PID_PAR *pid)
{
       long cal_result;
       long present_err;

       /*** SatErr, Kc not used ***/
       /* present error */
       present_err = pid->Ref - pid->Fdb;

       /* Integral term calculation */
       /* Add Integral terms (Ui * Ki) */
       cal_result= pid->Ki * present_err;

       if (cal_result> pid->IntegralMax)
       {
              cal_result= pid->IntegralMax;
       }
       else if ( cal_result< pid->IntegralMin)
       {
              cal_result= pid->IntegralMin;
       }

       /* Add Proportional terms (Pe * Kp) */
       cal_result += pid->Kp * present_err;

       /* Sum of instantaneous error */
       pid->Ui += present_err * pid->Dt;

       /* Slope of error over time */
       pid->Ud = (present_err - pid->Up1)/(pid->Dt);

       /* Add Derivative terms (De * Kd) */
       cal_result += pid->Kd * pid->Ud;

       pid->Up1 = present_err;

       /* Result check here */
       if (cal_result > pid->OutMax)
       {
              pid->Out = pid->OutMax;
              pid->Err = -1;
       }
       else if (cal_result < pid->OutMin)
       {
              pid->Out = pid->OutMin;
              pid->Err = -1;
       }
       else
       {
              pid->Out = cal_result;
              pid->Err = 0;
       }
}
```

3. Use of PID Controller

A simple example shows how to use the PID controller. In the example, the PID controller is initialized, with a target speed. For feedback control, current speed is fetched periodically and put into PID controller for calculation. After that, the calculated result will be used as new speed. This process continues, until the current speed reach the target.

```
#define TARGET_SPEED 100

struct PID_PAR pid_par;          /* PID variable */

void main()
{
       ....
       ....
       fPID_Init();
       pid_par.Ref = TARGET_SPEED;
       ....
       ....
       While(1)
       {
             /* Get Current Speed */
             get_speed(&Feedback);
             /* PID */
             pid_par.Fdb = Feedback;
             pid_par.calc(&pid_par);
             Cal_Result = pid_par.Out;

             /* Set speed */
             set_speed(Cal_Result);
       }
}
```

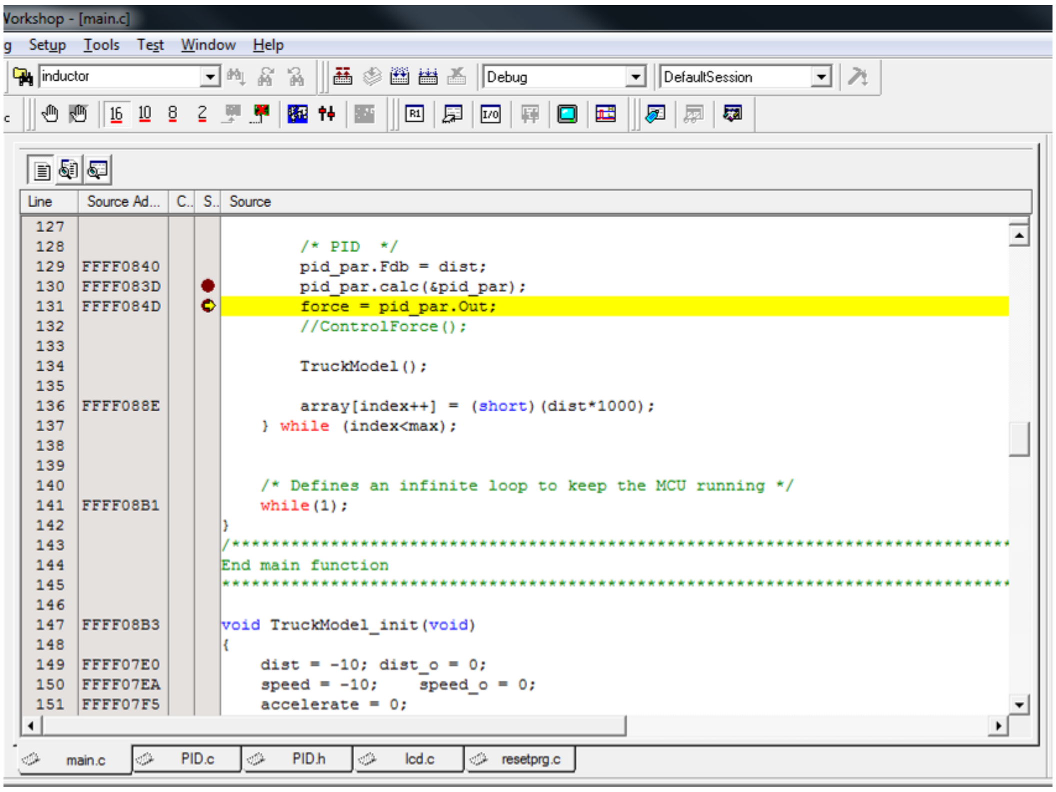
4. PID Controller Benchmark

This section describes the PID controller performance on RX and RL78 respectively. RX is using floating point PID controller for evaluation, and RL78 is using fix point PID controller for evaluation.

4.1 Floating point PID Controller on RX

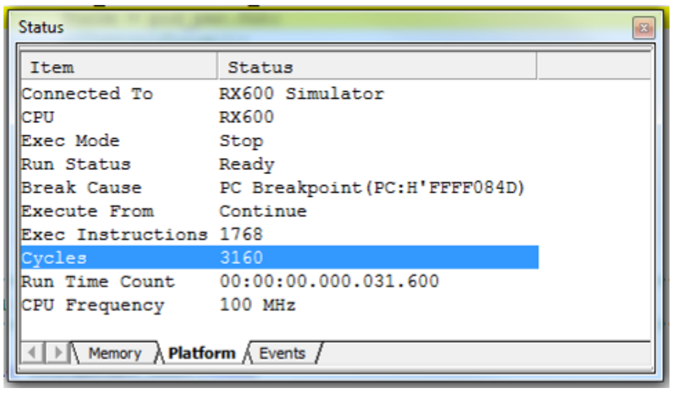
RX600 core is including FPU architecture that supports floating point calculation. In addition, the RX compiler provides options for different optimization levels for execution speed which might be worth to study of how the optimization level is selected affect the calculation speed of the PID controller. So in this section, it is tried to compare the execution cycle of the PID controller using different level of speed optimization options and RX simulator is used for the measurement.

To measure the execute cycle, S/W break point will be set before and after the PID controller,



```
129  FFFF0840        /* PID  */
                     pid_par.Fdb = dist;
130  FFFF083D        pid_par.calc(&pid_par);
131  FFFF084D        force = pid_par.Out;
                     //ControlForce();

                     TruckModel();

136  FFFF088E        array[index++] = (short)(dist*1000);
                 } while (index<max);


                 /* Defines an infinite loop to keep the MCU running */
141  FFFF08B1    while(1);
             }
             /**********************************************************************
             End main function
             ***********************************************************************

147  FFFF08B3 void TruckModel_init(void)
             {
149  FFFF07E0    dist = -10; dist_o = 0;
150  FFFF07EA    speed = -10;     speed_o = 0;
151  FFFF07F5    accelerate = 0;
```

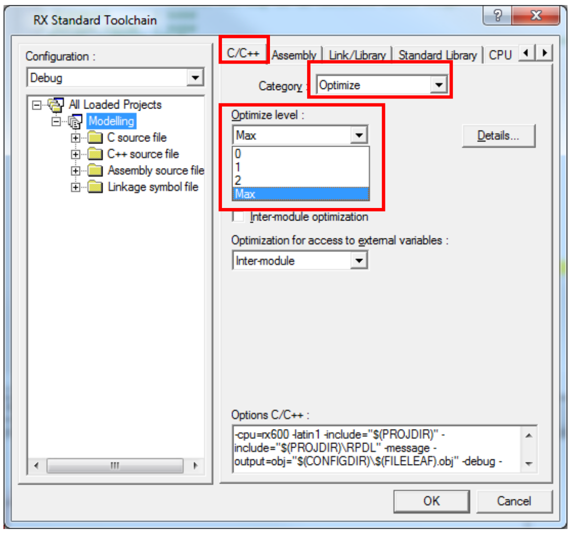
Meanwhile, the execution cycle has been calculated using the cycle information under “Status>Platform” windows.



Item	Status
Connected To	RX600 Simulator
CPU	RX600
Exec Mode	Stop
Run Status	Ready
Break Cause	PC Breakpoint(PC:H'FFFF084D)
Execute From	Continue
Exec Instructions	1768
Cycles	3160
Run Time Count	00:00:00.000.031.600
CPU Frequency	100 MHz

4.1.1 Execution cycle measurement of different optimization levels with FPU enabled

In order to perform the measurement, it is necessary to change the optimization level setting inside the “RX standard toolchain” which is shown as below. There are totally 4 levels which are 0,1,2,Max that can be selected. All these options are evaluated and result is shown on table 3-1.



Configuration : Debug

Category : Optimize

Optimize level : Max
0
1
2
Max

Inter-module optimization

Optimization for access to external variables : Inter-module

Options C/C++ :

```
-cpu=rx600 -latin1 -include="$(PROJDIR)" -
include="$(PROJDIR)\RPDL" -message -
output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -
```

Figure. Optimization setting window

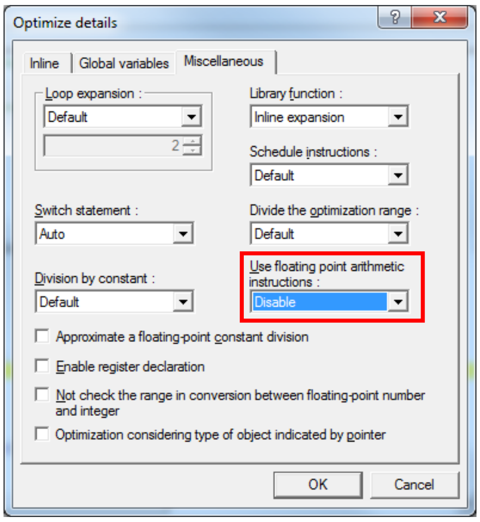
Table 3-1, No of execution cycle count for different optimization level

Compilation with FPU enabled				
Optimization level for speed	0	1	2	MAX
No. execution cycle	168	105	98	98

Remark: Standard Library Optimization for speed = 2 with FPU enabled

4.1.2 Execution cycle measurement of different optimization levels with FPU disabled

Besides, disable the FPU function in C Compiler is also tried for the benchmarking.



Loop expansion : Default
2

Library function : Inline expansion

Schedule instructions : Default

Switch statement : Auto

Divide the optimization range : Default

Division by constant : Default

Use floating point arithmetic instructions : Disable

Approximate a floating-point constant division

Enable register declaration

Not check the range in conversion between floating-point number and integer

Optimization considering type of object indicated by pointer

Compilation with FPU disabled				
Optimization level for speed	0	1	2	MAX
No. execution cycle	769	724	730	594

Remark: Standard Library Optimization for speed = 2 with FPU enabled

In case of 100MHz operation, each execution cycle consumes 10ns. In case of FPU enabled, the PID loop can be completed around 1us. However, if disabled the FPU, it takes at least 6us to complete 1 PID loop no matter optimization level is 0 or max which is 6 times higher than using FPU.

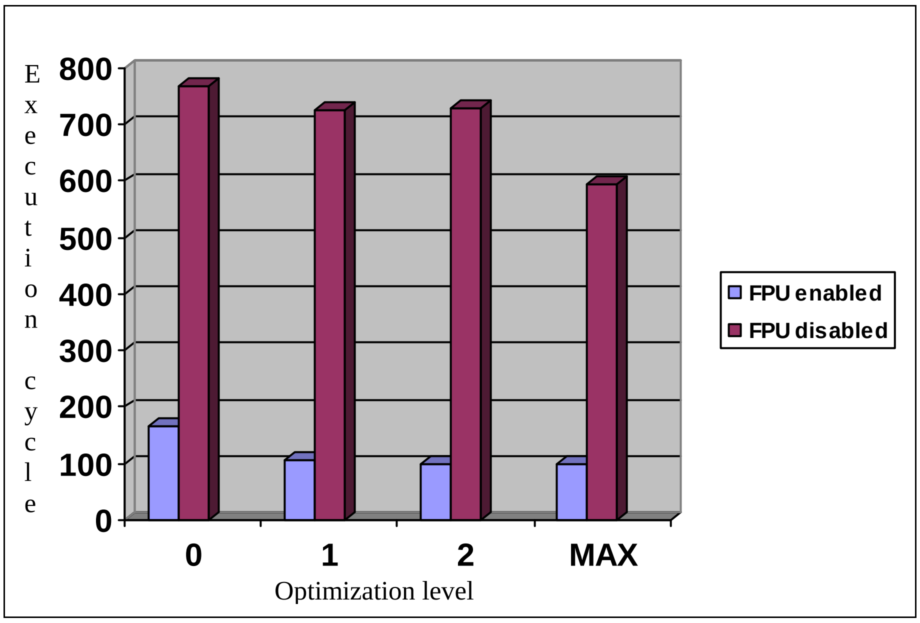
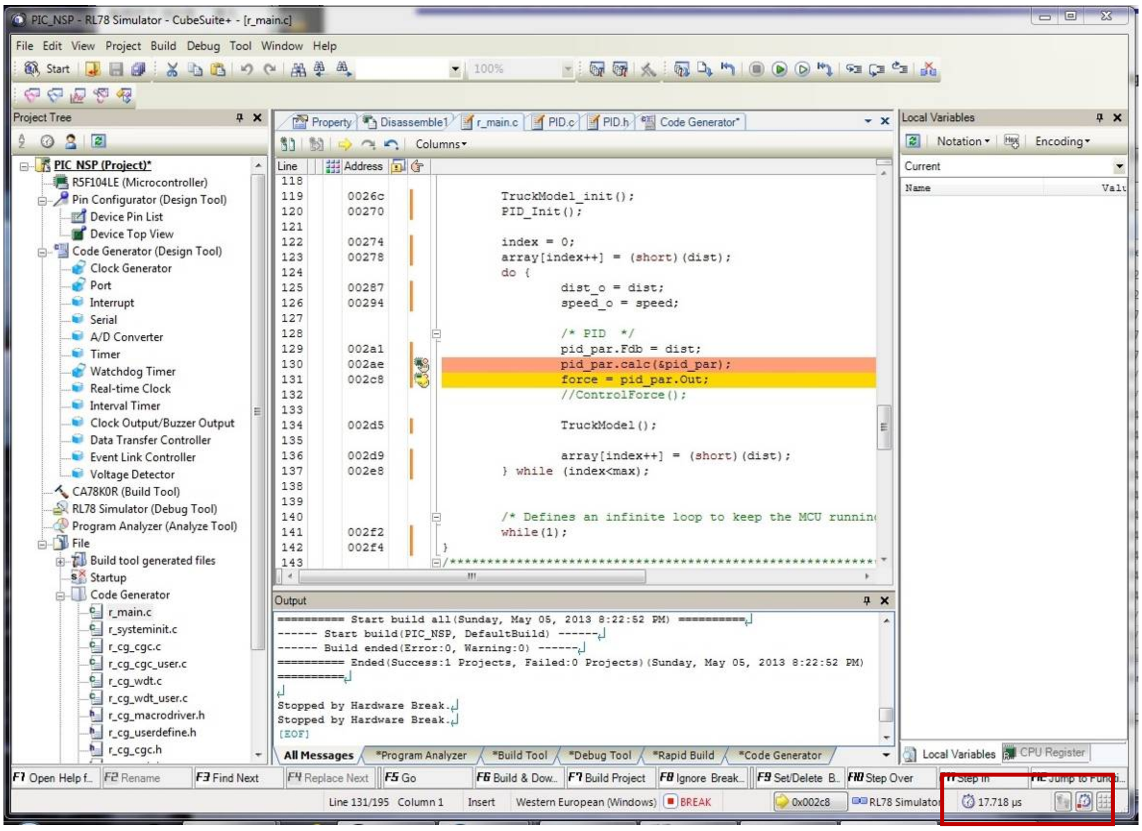


Figure 1 PID execution cycle measurements with different optimization levels and enabled/disabled FPU.

4.2 Fix point PID Controller on RL78

Unlike RX, RL78 has no hardware FPU to support floating point calculation. To enhance the performance, fix point (integer) calculation is used in the PID controller. For optimization, RL78 compiler also provides different types of optimization. In this section, it is tried to compare the execution cycle of the PID controller using different level of optimization options and RL78 simulator is used for the measurement. 32MHz clock is used for simulation.

To measure the execute time, S/W break point will be set before and after the PID controller, meanwhile, the execution time has been measured using timer in status bar of CubeSuite+ (red box below).



```
0026c        TruckModel_init();
00270        PID_Init();

00274        index = 0;
00278        array[index++] = (short)(dist);
             do {
00287                dist_o = dist;
00294                speed_o = speed;

                     /* PID  */
002a1                pid_par.Fdb = dist;
002ae                pid_par.calc(&pid_par);
002c8                force = pid_par.Out;
                     //ControlForce();

002d5                TruckModel();

002d9                array[index++] = (short)(dist);
002e8        } while (index<max);


             /* Defines an infinite loop to keep the MCU running
002f2        while(1);
002f4    }
         /*****************************************************
```

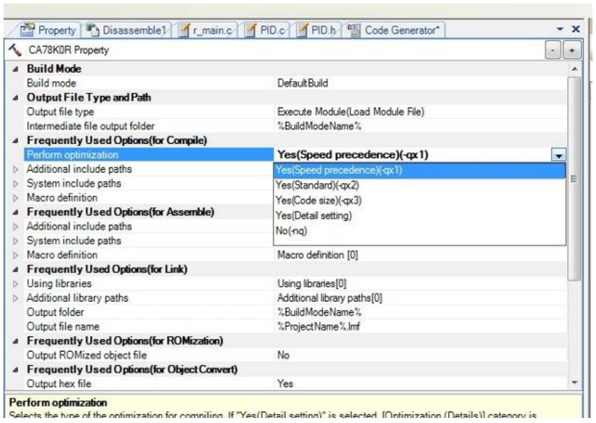
```
========== Start build all(Sunday, May 05, 2013 8:22:52 PM) ==========
------ Start build(PIC_NSP, DefaultBuild) ------
------ Build ended(Error:0, Warning:0) ------
========== Ended(Success:1 Projects, Failed:0 Projects)(Sunday, May 05, 2013 8:22:52 PM)
==========

Stopped by Hardware Break.
Stopped by Hardware Break.
[EOF]
```

Line 131/195 Column 1 | Insert | Western European (Windows) | BREAK | 0x002c8 | RL78 Simulator | 17.718 μs

PIC_NSP - RL78 Simulator - CubeSuite+ - [r_main.c]
File Edit View Project Build Debug Tool Window Help
Project Tree
PIC_NSP (Project)*
R5F104LE (Microcontroller)
Pin Configurator (Design Tool)
Code Generator (Design Tool)
CA78K0R (Build Tool)
RL78 Simulator (Debug Tool)
Program Analyzer (Analyze Tool)
Local Variables
Output
All Messages
F5 Go
F7 Build Project
F10 Step Over

4.2.1 Execution time measurement of different type of optimization

In order to perform the measurement, it is necessary to change the optimization types setting inside the "CA78K0R (Build Tool)" which is shown as below. There are totally 5 types of optimization. In measurement, 4 types were tested – Speed, Standard, Code size and no optimize. These options are evaluated and result is shown on table 4-1.



CA78K0R Property

Build Mode	
Build mode	DefaultBuild
Output File Type and Path	
Output file type	Execute Module(Load Module File)
Intermediate file output folder	%BuildModeName%
Frequently Used Options(for Compile)	
Perform optimization	Yes(Speed precedence)(-qx1)
Additional include paths	
System include paths	
Macro definition	
Frequently Used Options(for Assemble)	
Additional include paths	
System include paths	
Macro definition	Macro definition [0]
Frequently Used Options(for Link)	
Using libraries	Using libraries[0]
Additional library paths	Additional library paths[0]
Output folder	%BuildModeName%
Output file name	%ProjectName%.lmf
Frequently Used Options(for ROMization)	
Output ROMized object file	No
Frequently Used Options(for Object Convert)	
Output hex file	Yes

Yes(Speed precedence)(-qx1)
Yes(Standard)(-qx2)
Yes(Code size)(-qx3)
Yes(Detail setting)
No(-nq)

Perform optimization
Selects the type of the optimization for compiling. If "Yes(Detail setting)" is selected, [Optimization (Details)] category is

Figure. Optimization setting window

Table 4-1, Execution time for different optimization level

RL78 Simulator, 32MHz Clock, 31.25ns/cyc				
Type of Optimization	None (-nq)	Standard (-qx2)	Speed (-qx1)	Code (-qx3)
Measured execution time	17.218us	17.218us	17.218us	19.406us
Code Size of PID_Control()	435 bytes	433 bytes	435 bytes	339 bytes

From the table results above, it is shown that there is no different for optimization type – none, standard and speed. For code size optimization, although the code has 100-bytes less, the time taken also longer to complete the PID calculation.

4.3 Further Optimization

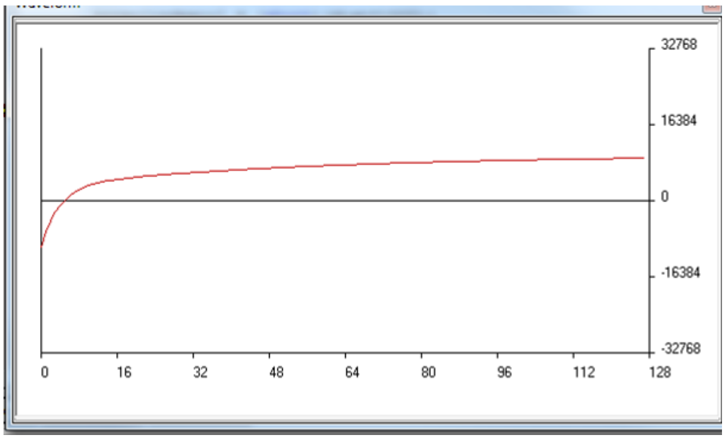
In some application like motor control, which Derivative term is small that can be ignored. In this case, the algorithm can be simplified to: **pid_out = (Pe * Kp) + (Ie * Ki)**. Hence it is known as PI Controller.

5. Evaluation Results of the PID Controller

To evaluate the RX PID Controller, a S/W truck model has been applied in the testing program and the response of the truck distance is calculated and stored into the array. After that, the content of the array can be plotted using waveform window in simulator to confirm the function of the PID Controller and also it can be observed that different setting value of Kp, Ki and Kd of how to influence the response of the system.

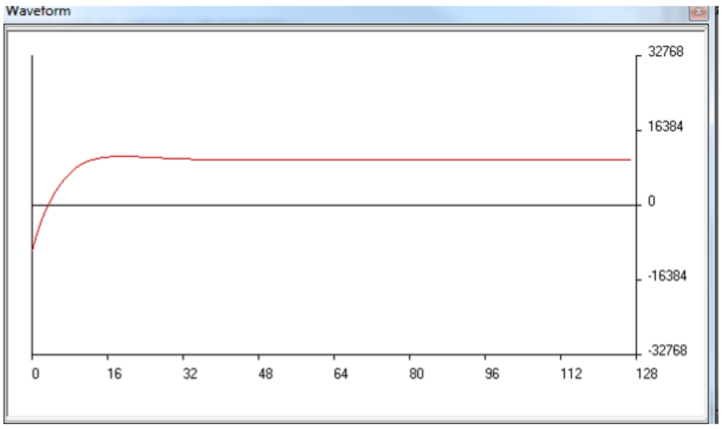
A) Kp=1, Ki=1, Kd=1

In this case, the PID response shows that the output is slowly approach the target setting.



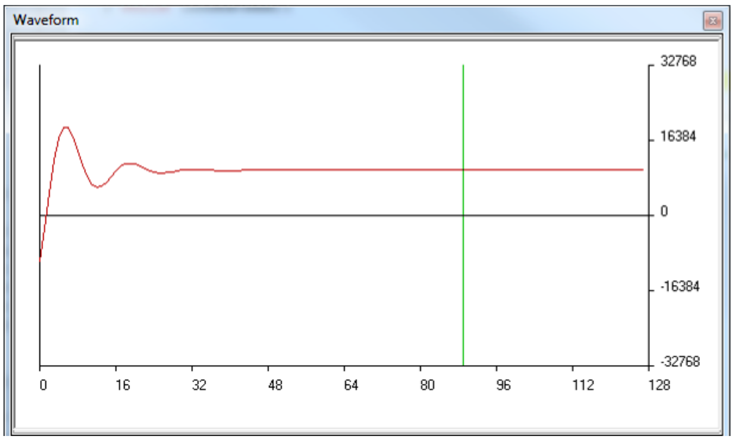
B) Kp = 10, Ki=1, Kd=1

In this case, increase the Proportional gain results speed up the time to reach the target, but slightly overshoot is introduced.



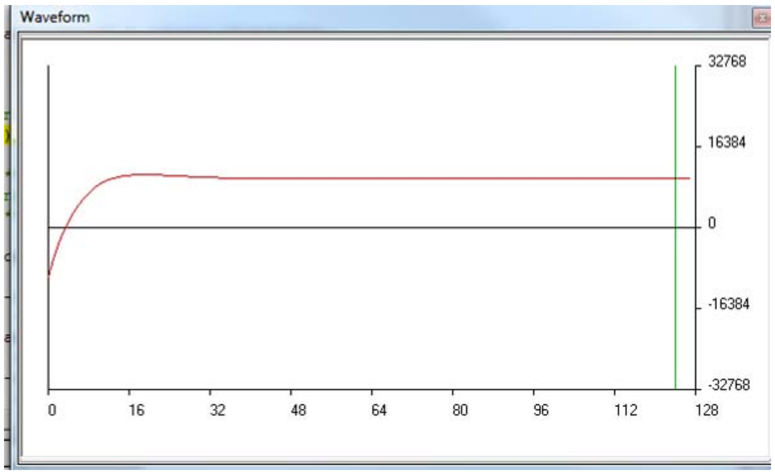
C) Kp=100, Ki=1, Kd=1

In this case, further increase the Proportional gain results further speed up the time to reach the target, but more overshoot is introduced.



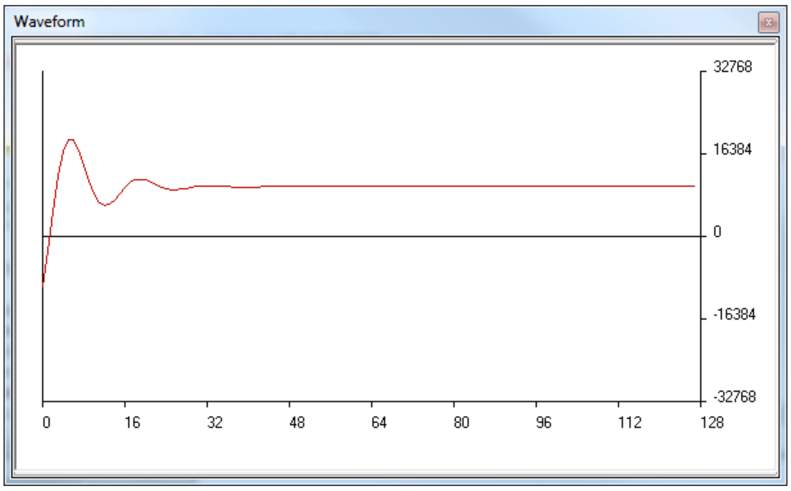
D) Kp=1, Ki=10, Kd=1

In this case, increase the Integral gain results speed up the time to reach the target, but slightly overshoot is introduced.



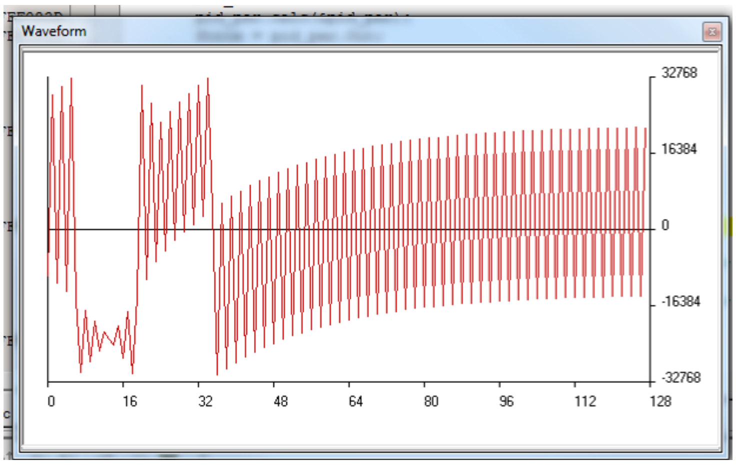
E) Kp=1, Ki=100, Kd=1

In this case, further increase the Integral gain results further speed up the time to reach the target, but more overshoot is introduced.



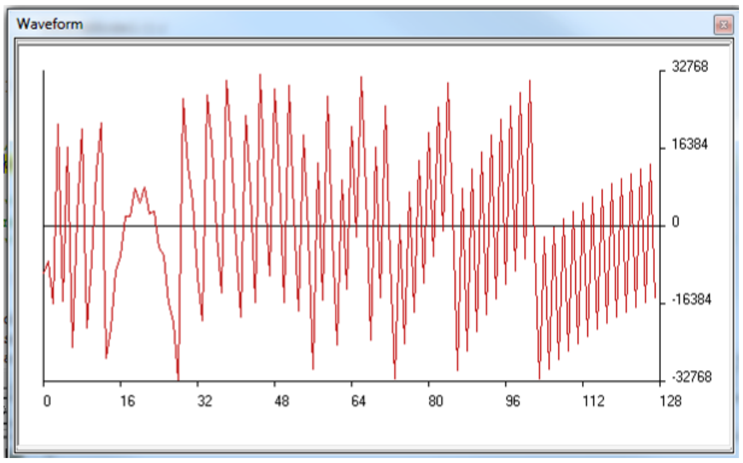
F) Kp=1, Ki=1, Kd=10

In this case, increase the Derivative gain results oscillation, although the output result is toward to the target setting, but it cannot be used.



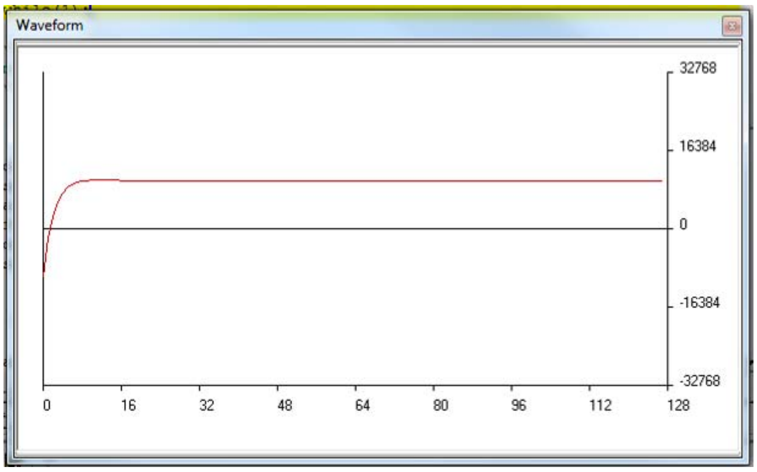
G) Kp=1, Ki=1, Kd=100

In this case, further increase the Derivative gain results more oscillation, basically it cannot be used.



H) Kp=10, Ki=10, Kd=2

In this case, Proportional gain and Integral gain are increase to 10, with the Derivative gain increase to 2. The PID response shows that the setup time has been reduced and overshoot has been also reduced.



5.1 Summary

In conclusion, a well tuned PID controller will improve the system response. A poor tuned PID controller could results damage to the system. So the tuning of PID controller is important for the application software.

Table below shows the effects of increasing a parameter independently:

Parameter	**Rise time**	**Overshoot**	**Settling time**	**Steady-state error**	**Stability**
Kp	Decrease	Increase	Small change	Decrease	Degrade
Ki	Decrease	Increase	Increase	Eliminate	Degrade
Kd	Minor change	Decrease	Decrease	No effect	Improve if Kd small

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

All trademarks and registered trademarks are the property of their respective owners.

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Mar.13.	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2013 Renesas Electronics Corporation. All rights reserved.
Colophon 2.2