PGS152

8bit MTP Type SuLED with EEPROM

Datasheet

Version 0.01 – Apr. 14, 2023

6F-6, No.1, Sec. 3, Gongdao 5th Rd., Hsinchu City 30069, Taiwan, R.O.C.

TEL: 886-3-572-8688 www.padauk.com.tw

IMPORTANT NOTICE

PADAUK Technology reserves the right to make changes to its products or to terminate production of its products at any time without notice. Customers are strongly recommended to contact PADAUK Technology for the latest information and verify whether the information is correct and complete before placing orders.

PADAUK Technology products are not warranted to be suitable for use in life-support applications or other critical applications. PADAUK Technology assumes no liability for such applications. Critical applications include, but are not limited to, those which may involve potential risks of death, personal injury, fire or severe property damage.

Any programming software provided by PADAUK Technology to customers is of a service and reference nature and does not have any responsibility for software vulnerabilities. PADAUK Technology assumes no responsibility for any issue caused by a customer's product design. Customers should design and verify their products within the ranges guaranteed by PADAUK Technology. In order to minimize the risks in customers' products, customers should design a product with adequate operating safeguards.

Table of content

Revision History

Revision	Date	Description
0.00	2022/10/26	Preliminary version
0.01	2023/04/14	1. Updated "IMPORTANT NOTICE" 2. Amend Section 5.5.4, 5.8, 5.9.4, 5.14.1 3. Amend Fig.2 and Fig.14 4. Other known details bug correct.

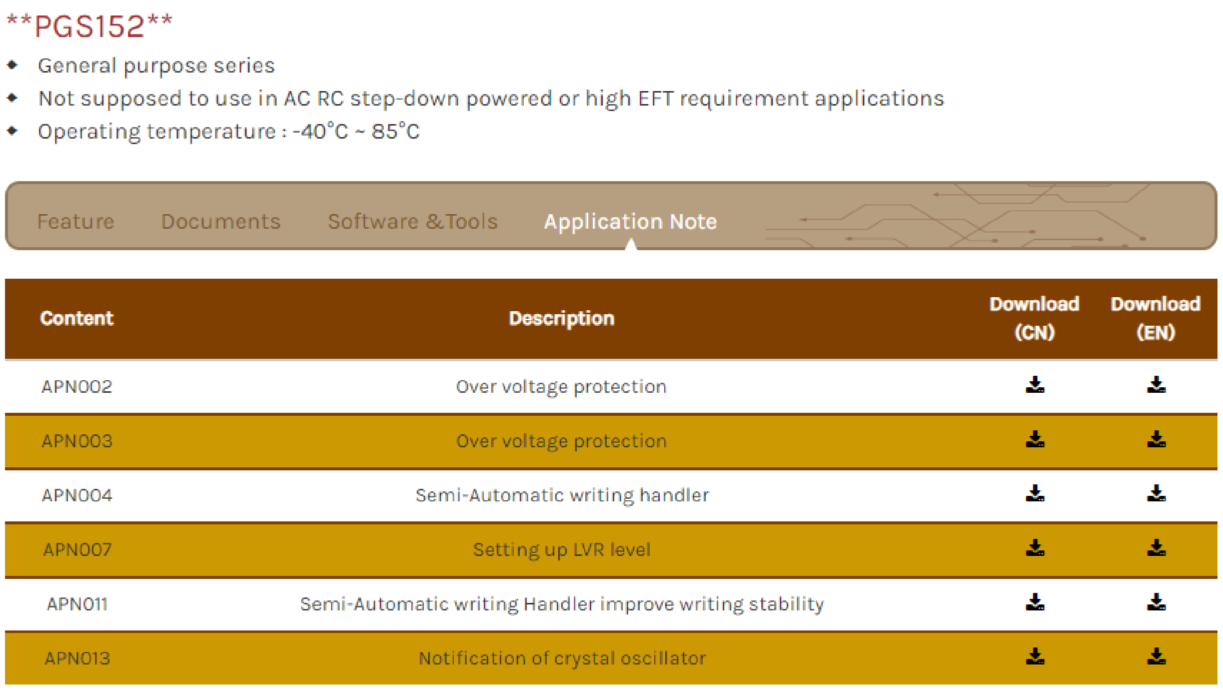
Usage Warning

User must read all application notes of the IC by detail before using it.

Please visit the official website to download and view the latest APN information associated with it.

http://www.padauk.com.tw/en/product/show.aspx?num=171&kw=PGS152

(The following picture are for reference only.)



PGS152

- General purpose series
- Not supposed to use in AC RC step-down powered or high EFT requirement applications
- Operating temperature : -40°C ~ 85°C

Content	Description	Download (CN)	Download (EN)
APN002	Over voltage protection	⭳	⭳
APN003	Over voltage protection	⭳	⭳
APN004	Semi-Automatic writing handler	⭳	⭳
APN007	Setting up LVR level	⭳	⭳
APN011	Semi-Automatic writing Handler improve writing stability	⭳	⭳
APN013	Notification of crystal oscillator	⭳	⭳

1. Features

1.1. Special Features

- ◆ General purpose series
- ◆ Not supposed to use in AC RC step-down powered or high EFT requirement applications.
  PADAUK assumes no liability if such kind of applications can not pass the safety regulation tests.
- ◆ Operating temperature range: -40°C ~ 85°C

1.2. System Features

- ◆ 1.5KW MTP program memory
- ◆ 128 Bytes data SRAM
- ◆ 128 Bytes EEPROM
- ◆ One hardware 16-bit timer
- ◆ One hardware 8-bit timers with 6/7/8-bit PWM generation
- ◆ One set triple 11bit SuLED (Super LED) PWM generators and timers
- ◆ One hardware comparator
- ◆ 8 IO pins with optional pull-high / pull-low resistor
- ◆ Every IO pin can be configured to enable wake-up function
- ◆ Two different IO Driving capability group to meet different application requirements
  Every IO Drive/ Sink Current= 40mA/50mA (Strong) and 5mA/10mA (Normal)
- ◆ Clock sources: IHRC, ILRC & EOSC(XTAL mode)
- ◆ Built-in crystal oscillator capacitance, Disable / 7pF/ 9.5pF / 12.5pF are available
- ◆ For every wake-up enabled IO, two optional wake-up speed are supported: normal and fast
- ◆ Eight levels of LVR: 4.0V, 3.5V, 3.0V, 2.7V, 2.5V, 2.2V, 2.0V and 1.8V
- ◆ Two selectable external interrupt pins by code option
- ◆ Bandgap circuit to provide 1.20V reference voltage

1.3. CPU Features

- ◆ One processing unit operating mode
- ◆ 88 powerful instructions
- ◆ Most instructions are 1T execution cycle
- ◆ Programmable stack pointer to provide adjustable stack level
- ◆ Direct and indirect addressing modes for data access. Data memories are available for use as an index pointer of Indirect addressing mode
- ◆ IO space and memory space are independent

1.4. Ordering/Package Information

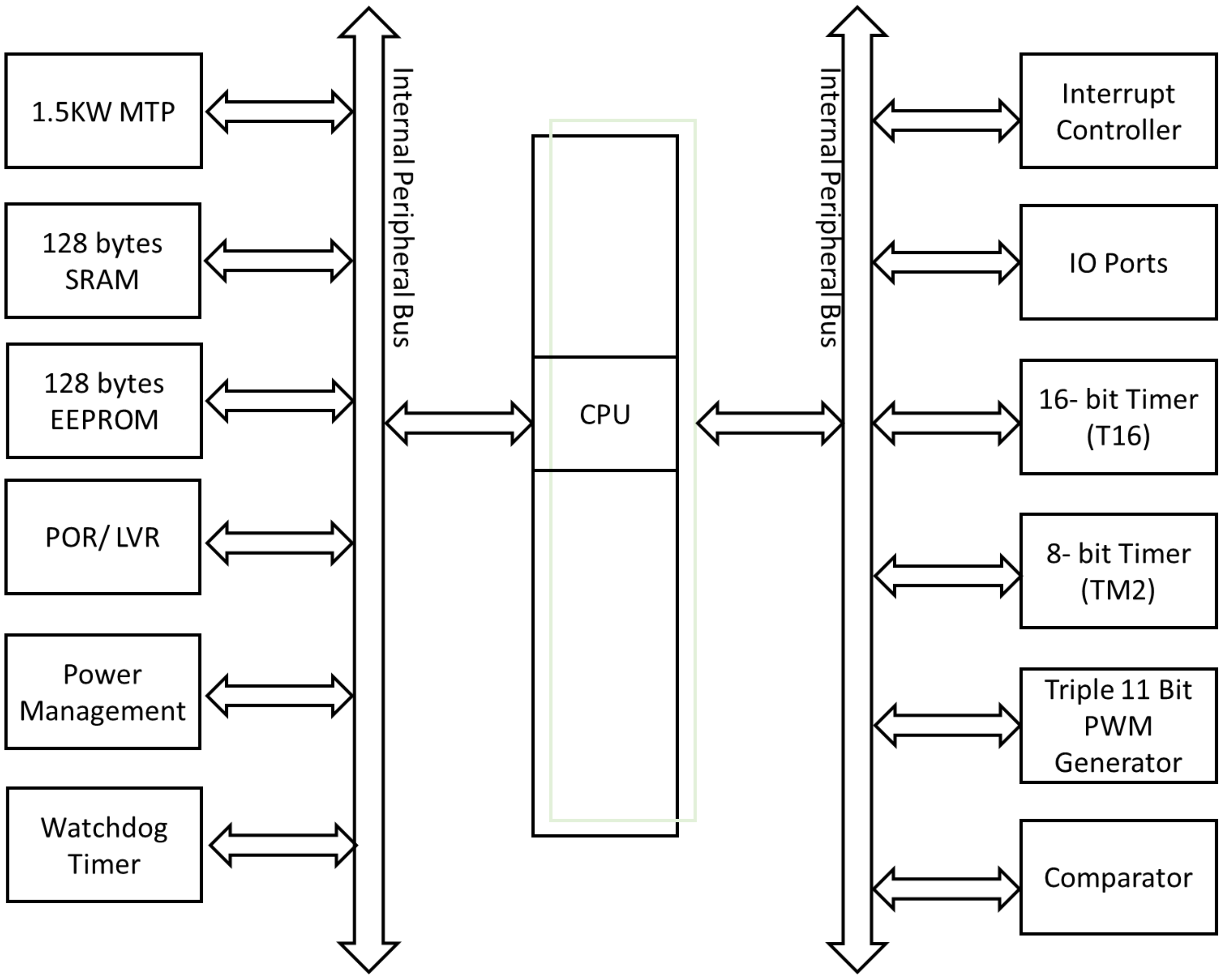
- ◆ PGS152-U06: SOT23-6 (60mil)
- ◆ PGS152-S08: SOP8 (150mil)
- ◆ PGS152-M10: MSOP10 (118mil)
- ◆ PGS152-EY10: ESSOP10 (150mil)

- ● Please refer to the official website file for package size information：”Package information “

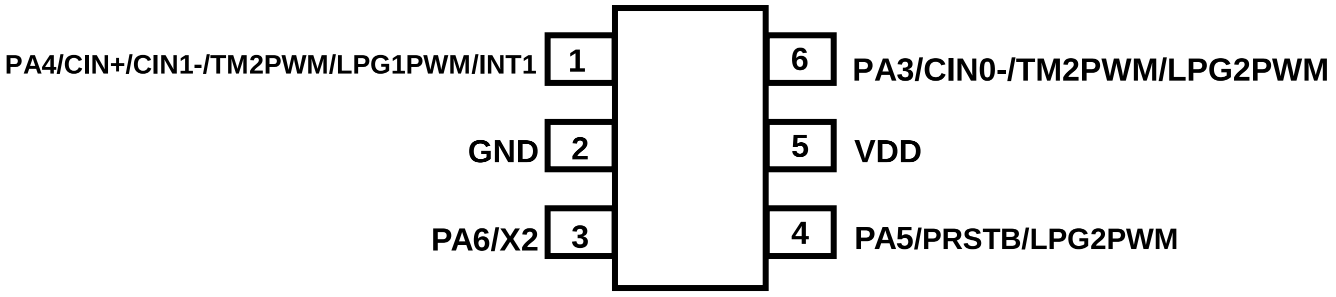
2. General Description and Block Diagram

The PGS152 family is an IO-Type, fully static, MTP-based CMOS 8-bit microcontroller. It employs RISC architecture and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access.

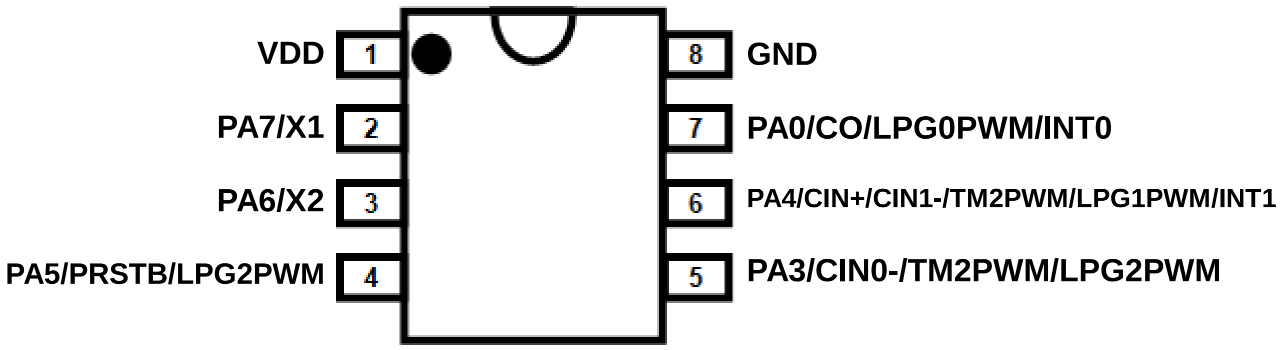
1.5KW bits MTP program memory, 128 bytes EEPROM and 128 bytes data SRAM are inside, one hardware comparator is built inside the chip to compare signal between two pin or with either internal reference voltage $V_{internalR}$ or internal bandgap reference voltage. PGS152 also provides three hardware timers: one 16-bit timer, one 8-bit timer with PWM generation, and one new triple 11-bit timer with SuLED PWM generation (LPWMG0, LPWMG1 & LPWMG2) are included.



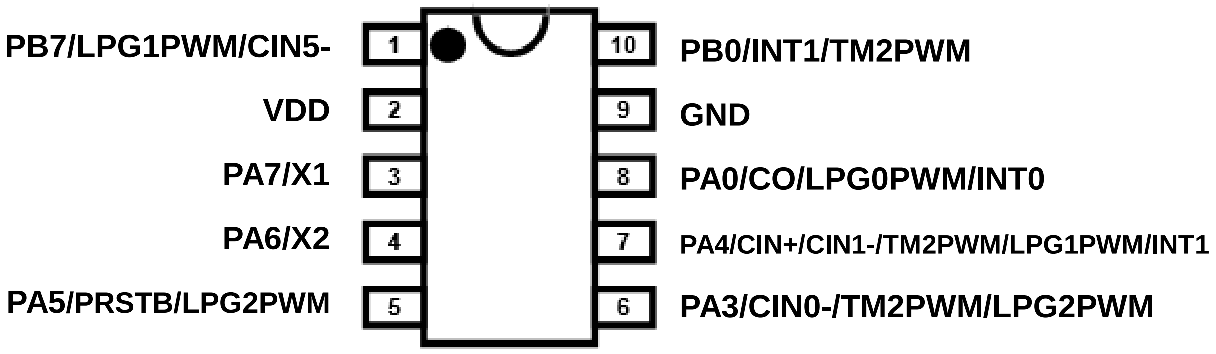
3. Pin Definition and Functional Description



PGS152-U06 (SOT23-6 60mil)



PGS152-S08: SOP8 (150mil)



PGS152-M10: MSOP10 (118mil)
PGS152-EY10: ESSOP10 (150mil)

Pin Name	Pin Type & Buffer Type	Description
PA7 / X1 /	IO ST / CMOS	The functions of this pin can be: (1) Bit 7 of port A. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) X1 is Crystal XIN when crystal oscillator is used. If this pin is used for crystal oscillator, bit 7 of ***padier*** register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 7 of ***padier*** register is "0".
PA6 / X2	IO ST / CMOS	The functions of this pin can be: (1) Bit 6 of port A. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) X2 is Crystal XOUT when crystal oscillator is used. If this pin is used for crystal oscillator, bit 6 of ***padier*** register must be programmed "0" to avoid leakage current. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 6 of ***padier*** register is "0".
PA5 / PRSTB / LPG2PWM	IO ST / CMOS	The functions of this pin can be: (1) Bit 5 of port A. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) Hardware reset. (3) Output of 11-bit PWM generator LPWMG2. This pin can be used to wake-up system during sleep mode; however, wake-up function is also disabled if bit 5 of ***padier*** register is "0". Please put 33Ω resistor in series to have high noise immunity when this pin is in input mode.
PA4 / CIN+ / CIN1- / LPG1PWM / INT1	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 4 of port A. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) Plus input source of comparator. (3) Minus input source 1 of comparator. (4) Output of 11-bit PWM generator LPWMG1. (5) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting When this pin is configured as analog input, please use bit 4 of register ***padier*** to disable the digital input to prevent current leakage. The bit 4 of ***padier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PA3 / CIN0- / TM2PWM / LPG2PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 3 of port A. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) Minus input source 0 of comparator. (3) PWM output from Timer2 (4) Output of 11-bit PWM generator LPWMG2 When this pin is configured as analog input, please use bit 3 of register ***padier*** to disable the digital input to prevent current leakage. The bit 3 of ***padier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled.
PA0 / CO / LPG0PWM / INT0	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port A. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) Output of comparator. (3) Output of 11-bit PWM generator LPWMG0. (4) External interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting The bit 0 of ***padier*** register can be set to "0" to disable wake-up from power-down by toggling this pin.
PB7 / LPG1PWM / CIN5-	IO ST / CMOS	The functions of this pin can be: (1) Bit 7 of port B. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) Output of 11-bit PWM generator LPWMG1. (3) Minus input source 5 of comparator. When this pin is configured as analog input, please use bit 7 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 7 of ***pbdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PB0 / INT1 / TM2PWM	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port B. It can be configured as digital input or two-state output, with pull-high resistor / pull-low resistor. (2) PWM output from Timer2 (3) External interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting When this pin is configured as analog input, please use bit 0 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 0 of ***pbdier*** register can be set to “0” to disable digital input; wake-up from power-down by toggling this pin is also disabled.
VDD	VDD	Positive power
GND	GND	Ground
Notes: **IO**: Input/Output; **ST**: Schmitt Trigger input; **OD**: Open Drain; **Analog**: Analog input pin **CMOS**: CMOS voltage level		

4. Device Characteristics

4.1. AC/DC Device Characteristics

All data are acquired under the conditions of Ta= -40 ^oC ~ 85 ^oC, V_{DD}=5.0V, f_{SYS} =2MHz unless noted.

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
V_{DD}	Operating Voltage	1.8*	5.0	5.5	V	* Subject to LVR tolerance
LVR%	Low Voltage Reset Tolerance	-5		5	%	
f_{SYS}	System clock (CLK)* = IHRC/2 IHRC/4 IHRC/8 ILRC	 0 0 0	 91K	 8M 4M 2M	Hz	 V_{DD} ≧ 3.5V V_{DD} ≧ 2.5V V_{DD} ≧ 1.8V V_{DD} = 5.0V
P_{cycle}	EEPROM Program cycle	1000			Cycles	
P_{read}	EEPROM Read Voltage	1.8			V	
P_{pgm}	EEPROM Erase/Program Voltage	2.5			V	
V_{POR}	Power On Reset Voltage		1.8*		V	* Subject to LVR tolerance
I_{OP}	Operating Current		0.6 120		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=91KHz@5.0V
I_{PD}	Power Down Current (by ***stopsys*** command)		1 0.6		uA uA	f_{SYS}= 0Hz, V_{DD} =5.0V f_{SYS}= 0Hz, V_{DD} =3.3V
I_{PS}	Power Save Current (by ***stopexe*** command)		4		uA	V_{DD} =5.0V; f_{SYS}= ILRC Only ILRC module is enabled.
V_{IL}	Input low voltage for IO lines	0		0.1 V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO lines sink current					
	PA5 ~ 7, PB0, PB7		15 60		mA	V_{DD}=5.0V, V_{OL}=0.5V OPR3[6] = 0 OPR3[6] = 1
	PA4, PA3, PA0		50		mA	V_{DD}=3.0V, V_{OL}=1.0V OPR3[4][2][0] = 1
I_{OH}	IO lines drive current					
	PA5 ~ 7, PB0, PB7		10 45		mA	V_{DD}=5.0V, V_{OH}=4.5V OPR3[6] = 0 OPR3[6] = 1
	PA4, PA3, PA0		50		mA	V_{DD}=3.0V, V_{OH}=2.0V OPR3[4][2][0] = 1
V_{IN}	Input voltage	-0.3		V_{DD} +0.3	V	
$I_{INJ (PIN)}$	Injected current on pin			1	mA	V_{DD} +0.3≧V_{IN}≧ -0.3
R_{PH}	Pull-high Resistance		74		KΩ	V_{DD} =5.0V

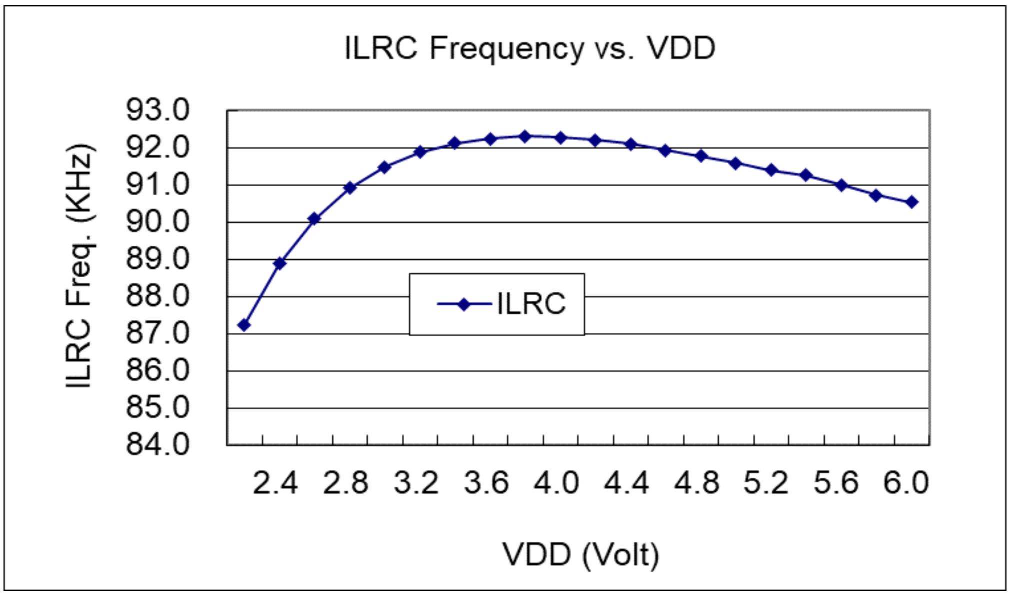
Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25ºC)
R_{PL}	Pull-low Resistance		74		KΩ	V_{DD} =5.0V
V_{BG}	Bandgap Reference Voltage	1.145*	1.20*	1.255*	V	V_{DD} =2.2V ~ 5.5V -40ºC <Ta<85ºC*
f_{IHRC}	Frequency of IHRC after calibration *****	15.76*	16*	16.24*	MHz	25ºC, V_{DD} =2.2V~5.5V
		15.20*	16*	16.80*		V_{DD} =2.2V~5.5V, -40ºC <Ta<85ºC*
		13.60*	16*	18.40*		V_{DD} =1.8V~5.5V, -40ºC <Ta<85ºC
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 5.0V
V_{DR}	RAM data retention voltage*	1.5			V	in stop mode
t_{WDT}	Watchdog timeout period		8k		T_{ILRC}	misc[1:0]=00 (default)
			16k			misc[1:0]=01
			64k			misc[1:0]=10
			256k			misc[1:0]=11
t_{WUP}	Wake-up time period for fast wake-up		45		T_{ILRC}	Where T_{ILRC} is the time period of ILRC
	Wake-up time period for slow wake-up		3000			
t_{SBP}	System boot-up period from power-on for Slow boot-up		33		ms	V_{DD} =5V
	System boot-up period from power-on for Fast boot-up		580		us	
t_{RST}	External reset pulse width	120			us	@ V_{DD} =5V
CPos	Comparator offset*		±10	±20	mV	
CPcm	Comparator input common mode*	0		V_{DD} -1.5	V	
CPspt	Comparator response time**		100	500	ns	Both Rising and Falling
CPmc	Stable time to change comparator mode		2.5	7.5	us	
CPcs	Comparator current consumption		20		uA	V_{DD} = 3.3V

*These parameters are for design reference, not tested for each chip.

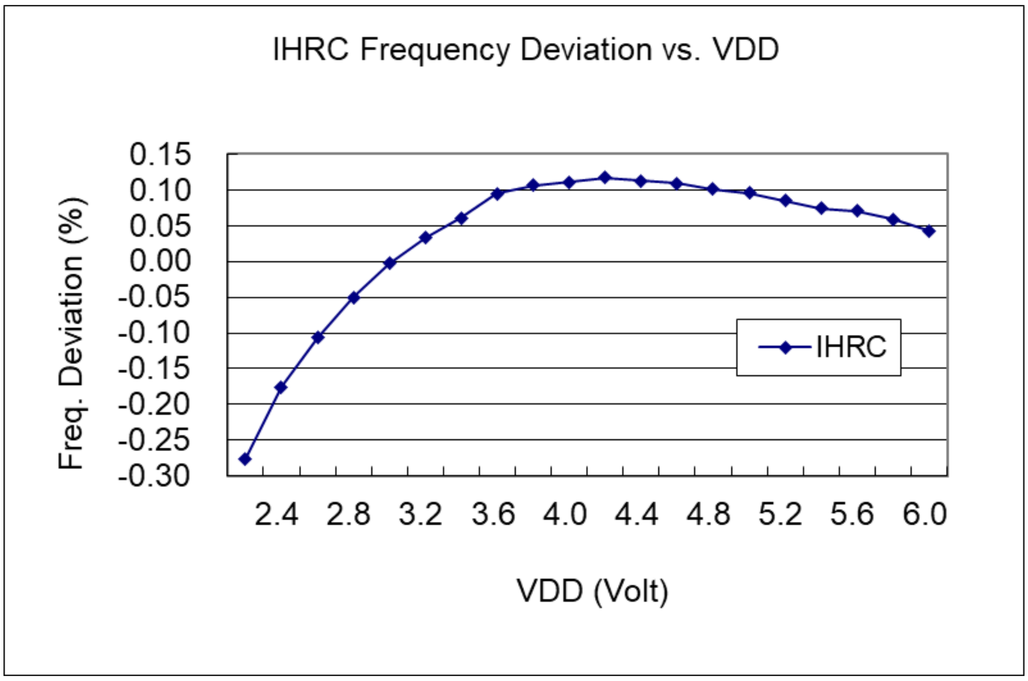
4.2. Absolute Maximum Ratings

● Supply Voltage	1.8V ~ 5.5V (Maximum Rating: 5.5V)
*If V_{DD} is over the maximum rating, it may lead to a permanent damage of IC.	
● Input Voltage ………………………………..	-0.3V ~ V_{DD} + 0.3V
● Operating Temperature ………………………	-40°C ~ 85°C
● Storage Temperature …………………………	-50°C ~ 125°C
● Junction Temperature ………………………..	150°C

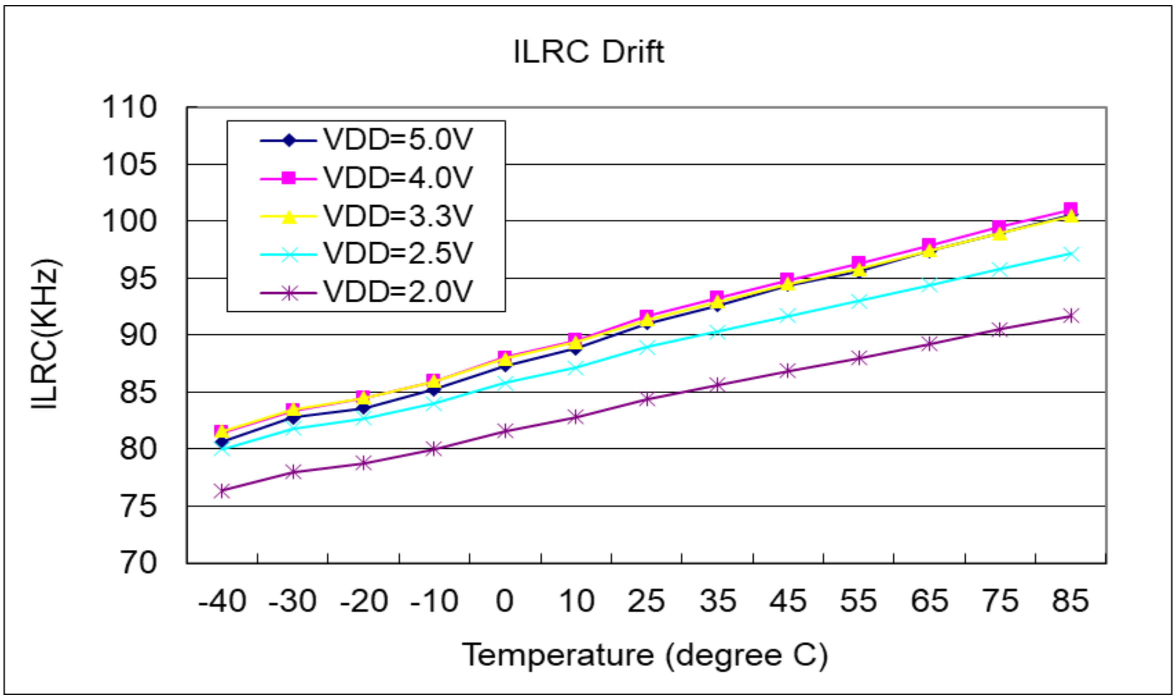
4.3. Typical ILRC frequency vs. VDD



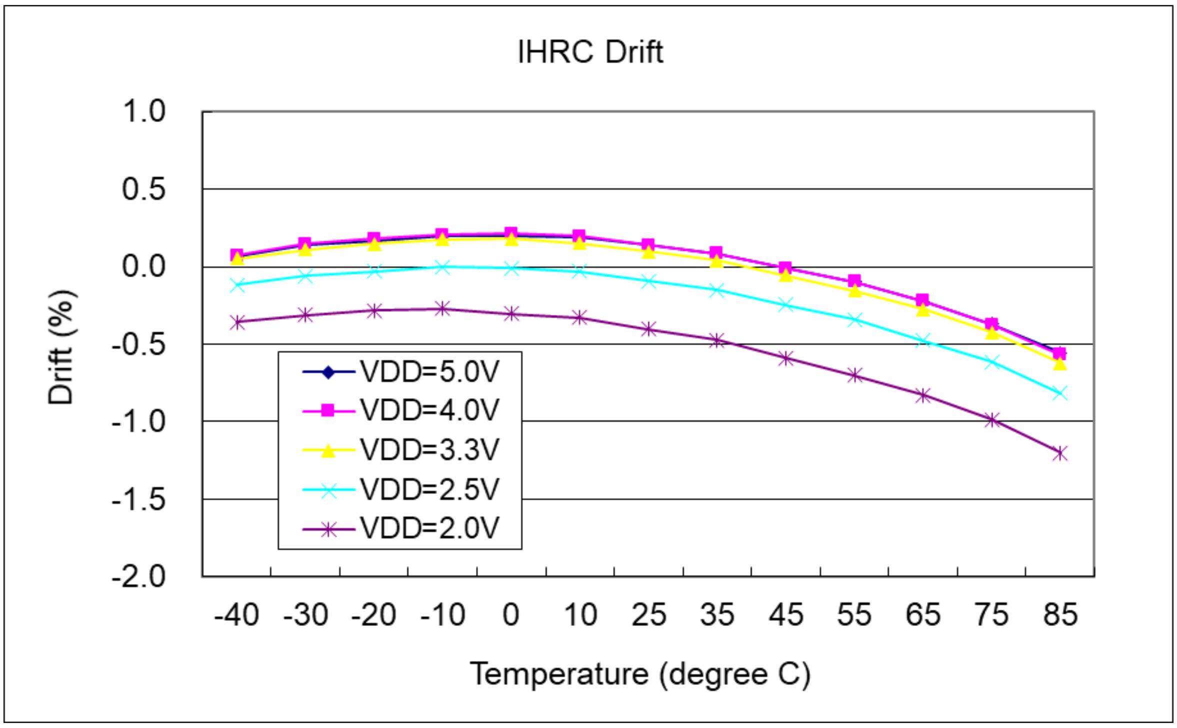
4.4. Typical IHRC frequency deviation vs. VDD (calibrated to 16MHz)



4.5. Typical ILRC Frequency vs. Temperature



4.6. Typical IHRC Frequency vs. Temperature (calibrated to 16MHz)

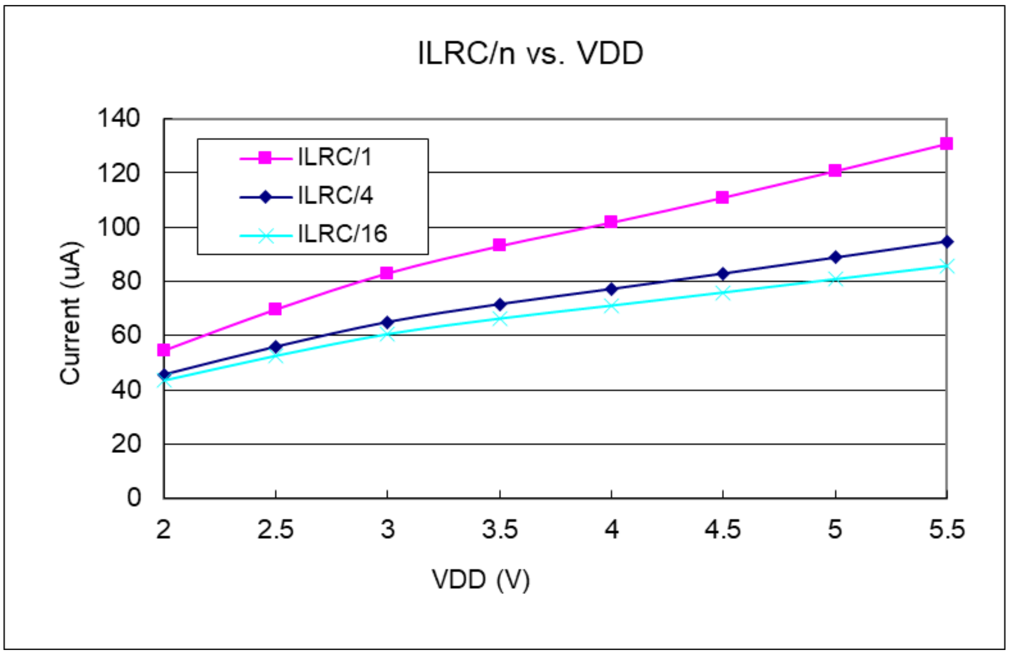


4.7. Typical operating current vs. VDD @ system clock = ILRC/n

Conditions:

ON: Bandgap, LVR, ILRC; **OFF**: IHRC, EOSC, T16, TM2;

IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

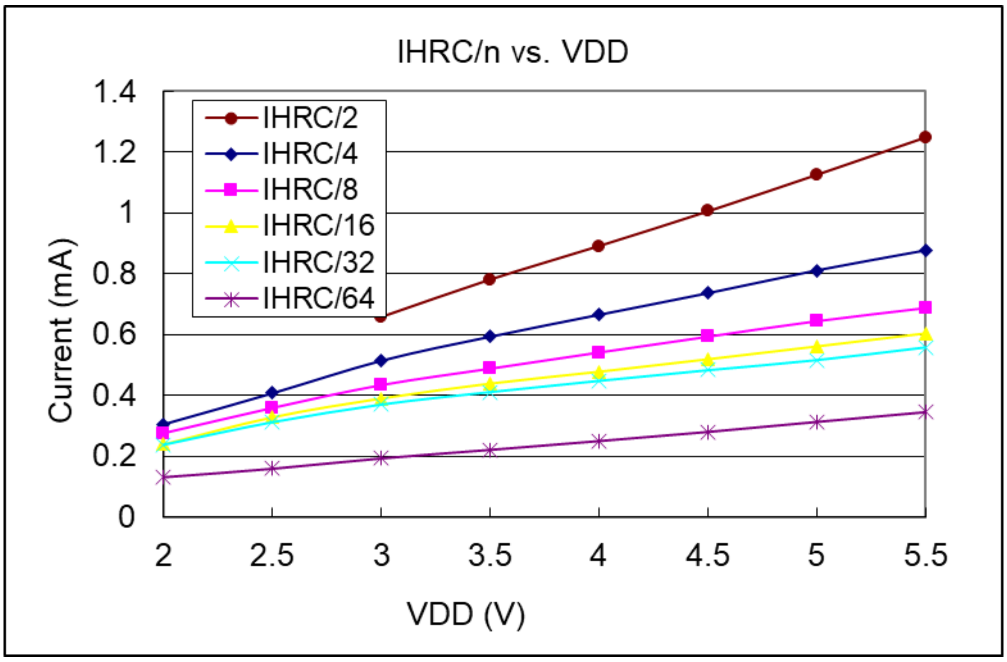


4.8. Typical operating current vs. VDD @ system clock = IHRC/n

Conditions:

ON: Bandgap, LVR, IHRC; **OFF**: ILRC, EOSC, T16, TM2;

IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

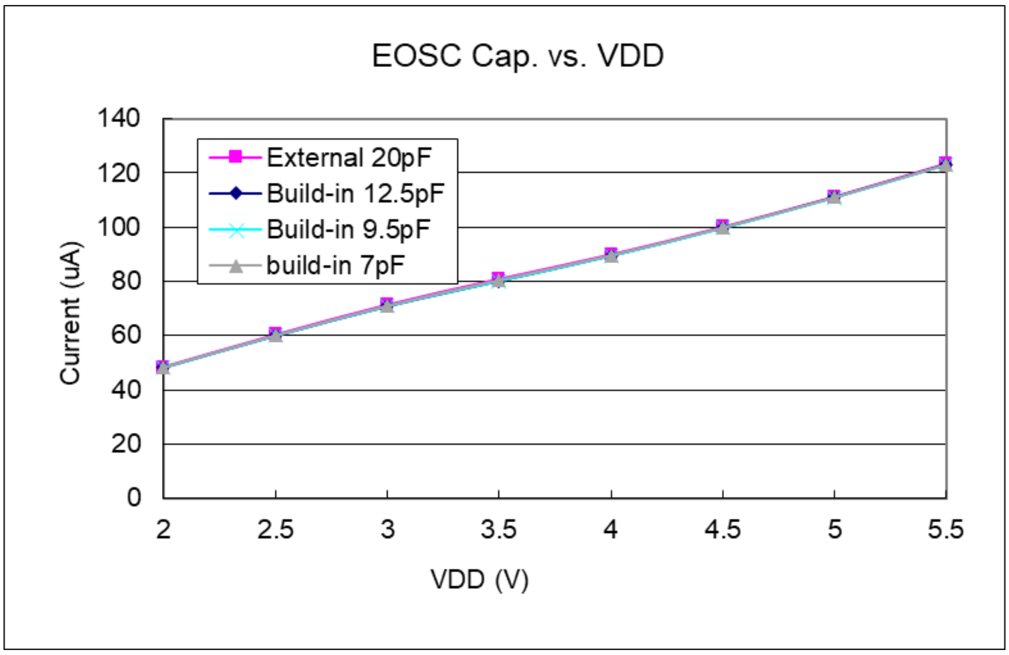


4.9. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n

Conditions:

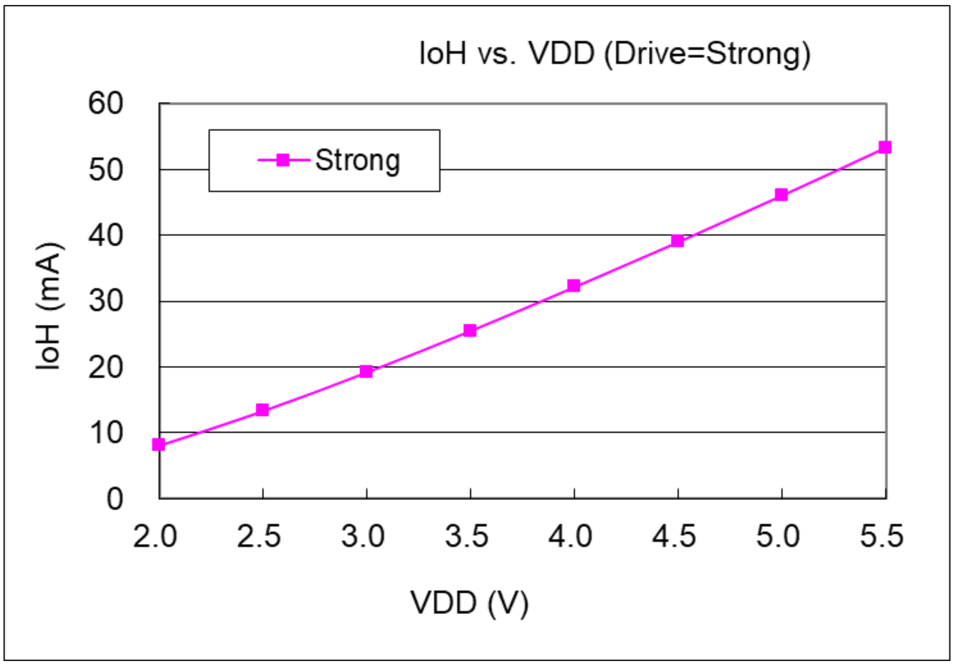
ON: Bandgap, LVR, EOSC; **OFF**: IHRC, ILRC, T16, TM2;

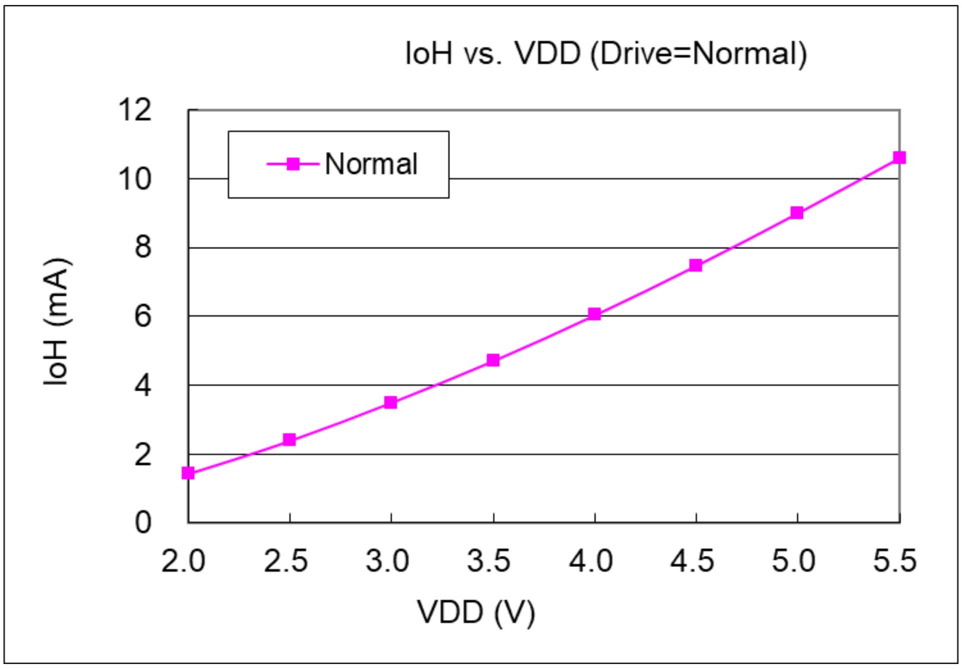
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

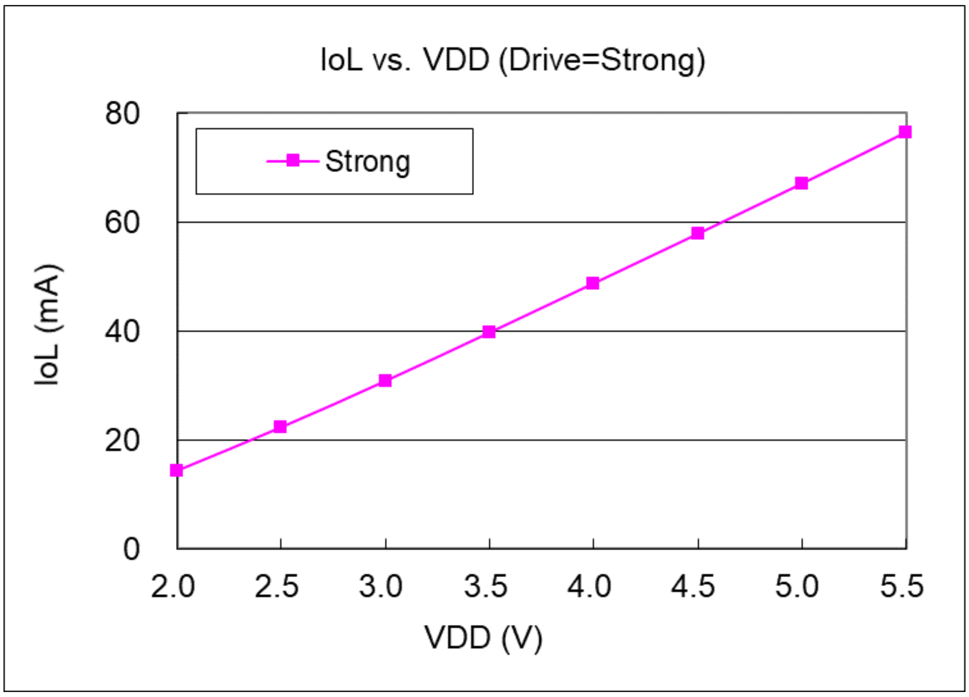


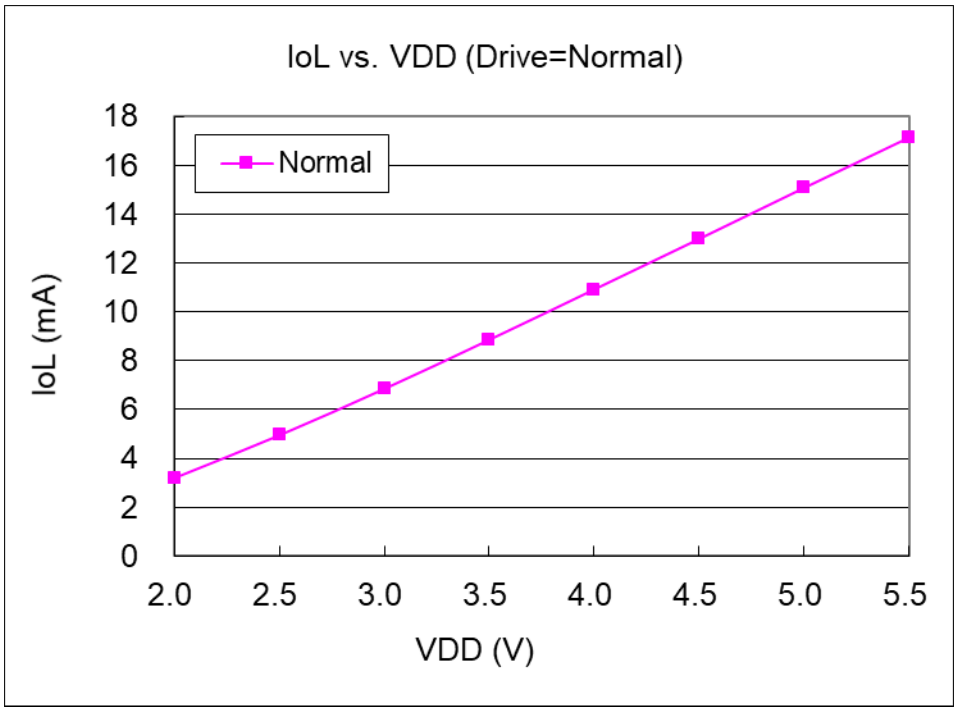
4.10.Typical IO driving current (I_{OH}) and sink current (I_{OL})

(VOH=0.9*VDD, VOL=0.1*VDD)

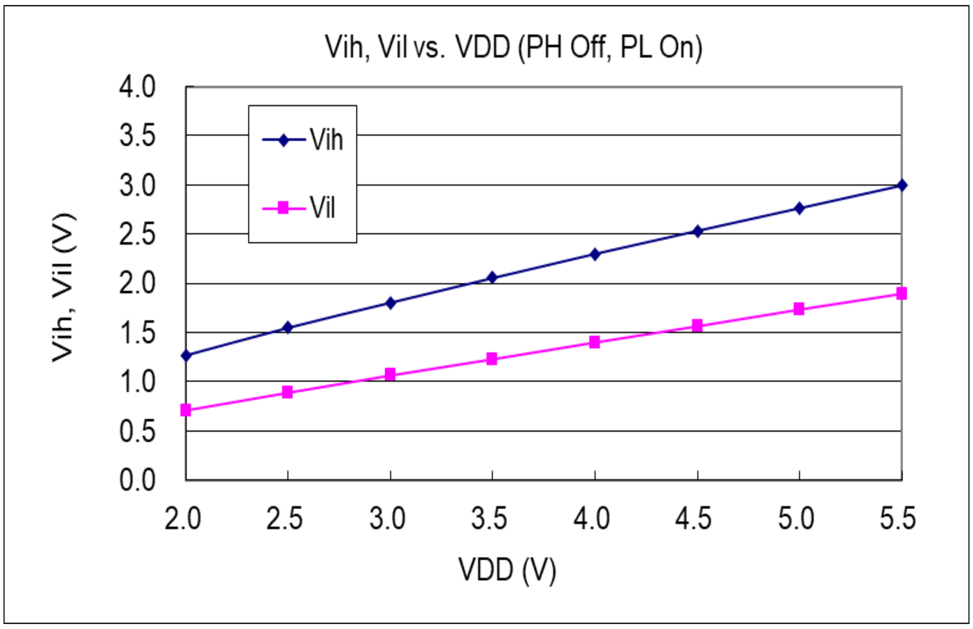



IoH vs. VDD (Drive=Normal)
Normal
IoH (mA)
VDD (V)
0
2
4
6
8
10
12
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

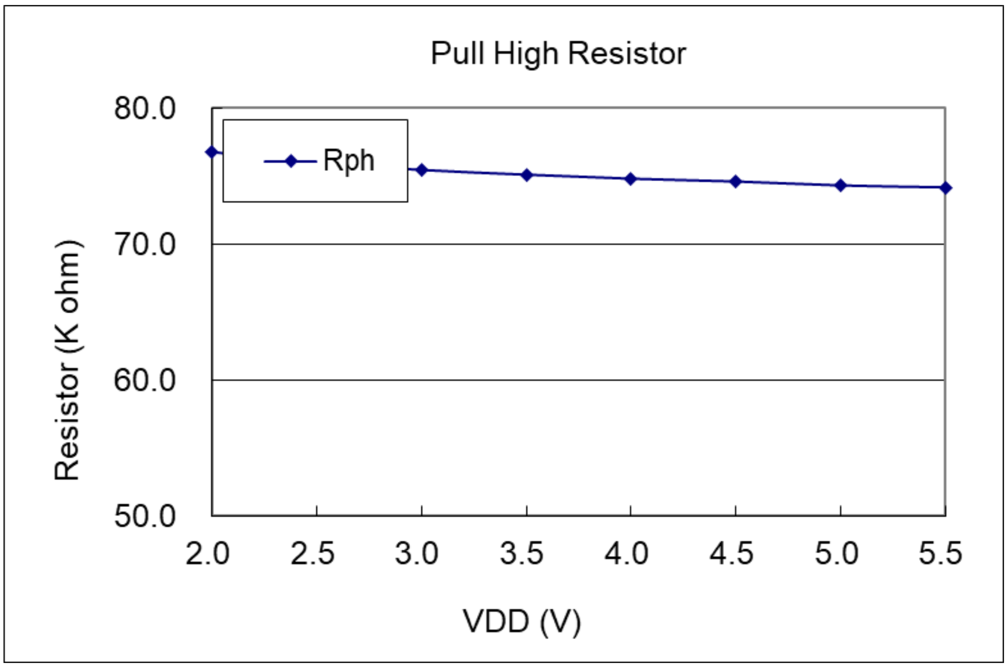

IoL vs. VDD (Drive=Strong)
Strong
IoL (mA)
VDD (V)
0
20
40
60
80
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

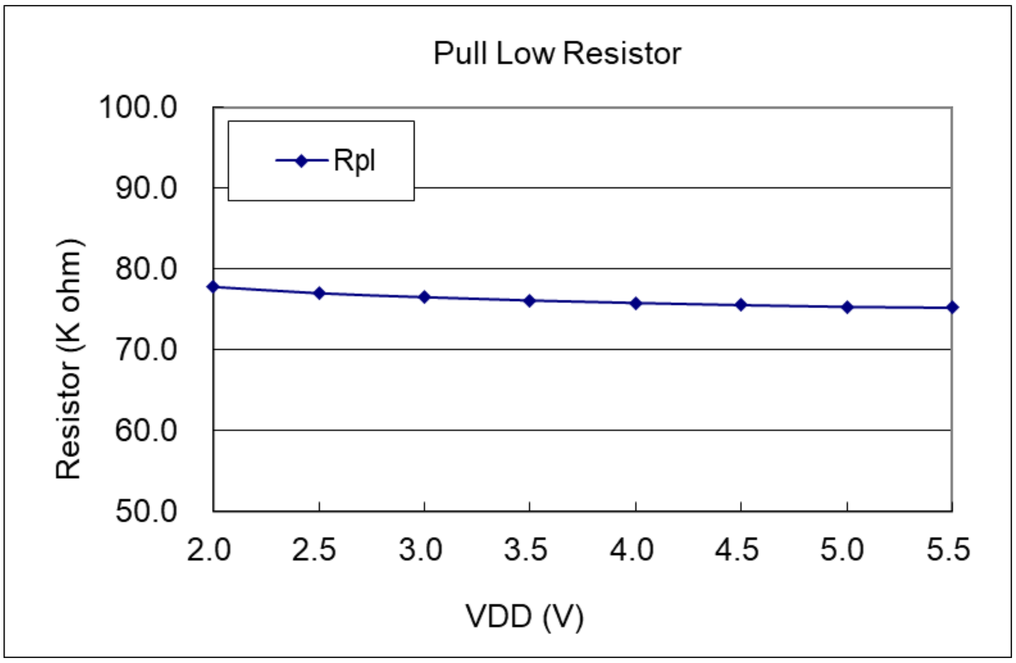


4.11. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})

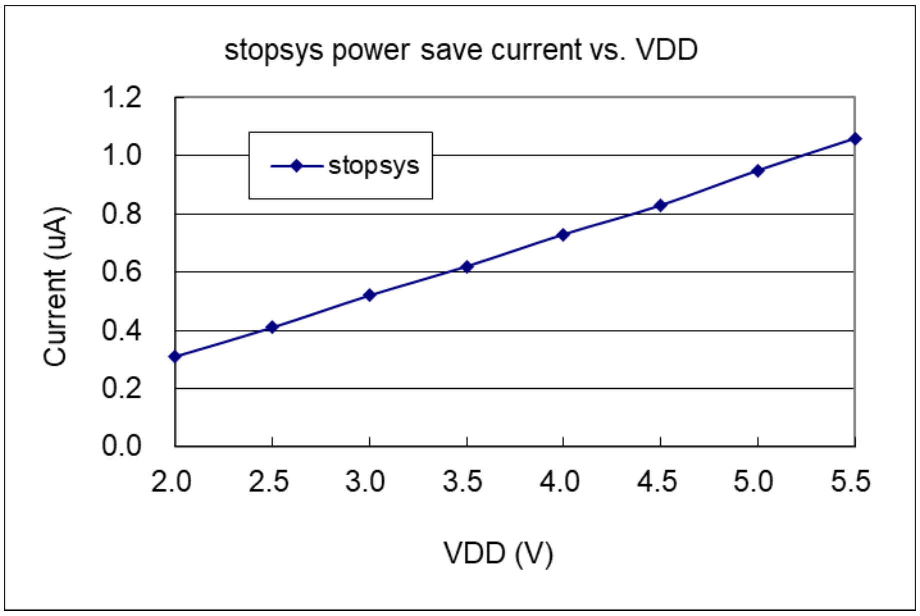


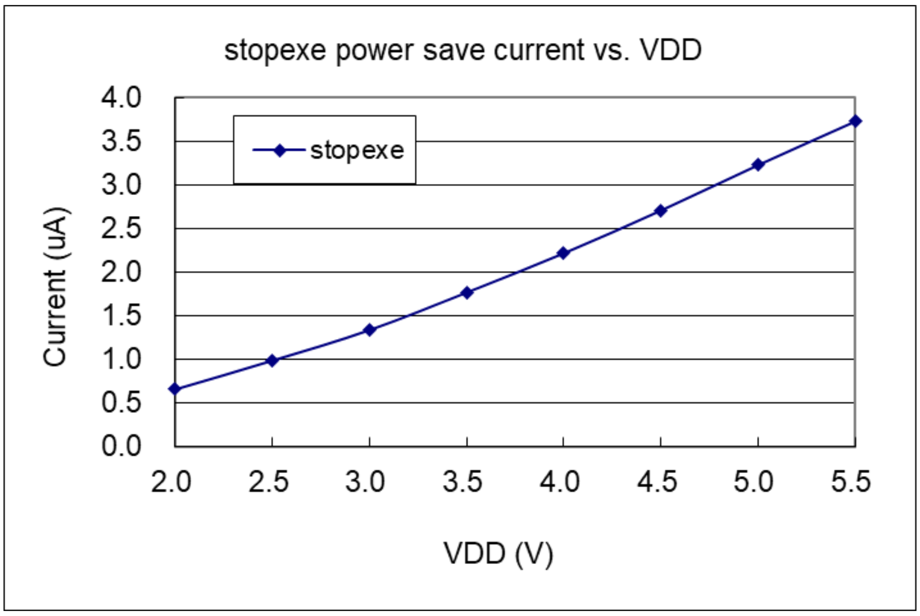
4.12. Typical resistance of IO pull high/low device





4.13. Typical power down current (I_{PD}) and power save current (I_{PS})





5. Functional Description

5.1. Program Memory - MTP

The MTP (One Time Programmable) program memory is used to store the program instructions to be executed. The MTP program memory may contains the data, tables and interrupt entry. After reset, the program will start from 0x000 which is GOTO FPPA0 instruction usually. The interrupt entry is 0x010 if used, the last 32 addresses are reserved for system using, like checksum, serial number, etc. The MTP program memory for PGS152 is 1.5KW that is partitioned as Table 1. The MTP memory from address '0x5E0 to 0x5FF is for system using, address space from0x001 to 0x00F and from 0x011 to 0x4DF are user program spaces.

Address	Function
0x000	GOTO FPPA0 instruction
0x001	User program
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0x4DF	User program
0x5E0	System Using
•	•
0x5FF	System Using

Table 1: Program Memory Organization

5.2. Boot Procedure

POR (Power-On-Reset) is used to reset PGS152 when power up. The boot up time can be optional fast or normal. Time for fast boot-up is about 45 ILRC clock cycles whereas 3000 ILRC clock cycles for normal boot-up. Customer must ensure the stability of supply voltage after power up no matter which option is chosen, the power up sequence is shown in the Fig. 1 and t_{SBP} is the boot up time.

Please noted, during Power-On-Reset, the V_{DD} must go higher than V_{POR} to boot-up the MCU.

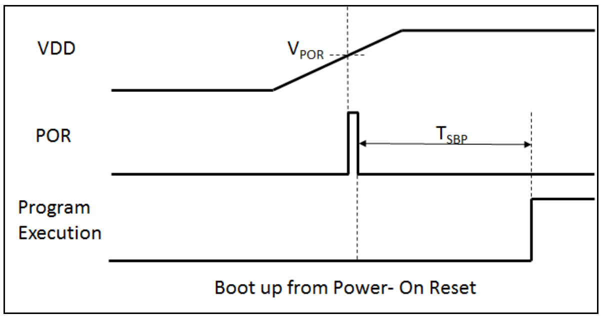
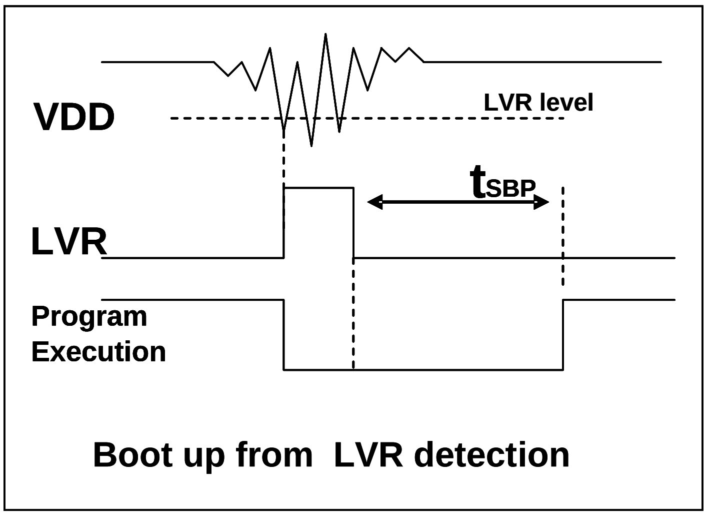
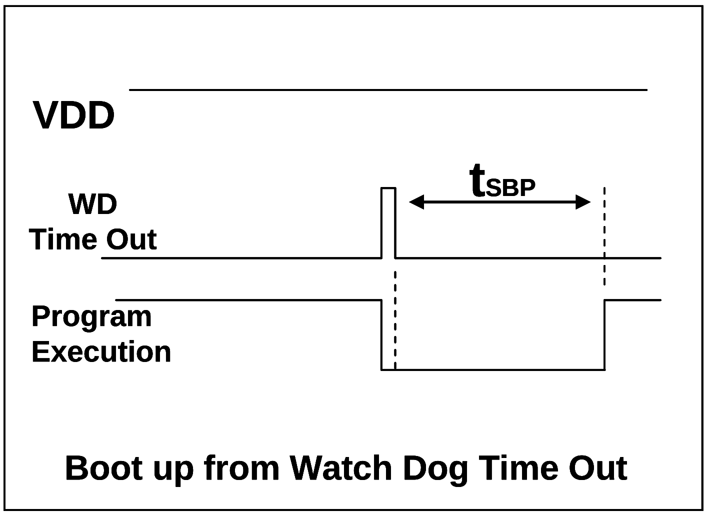
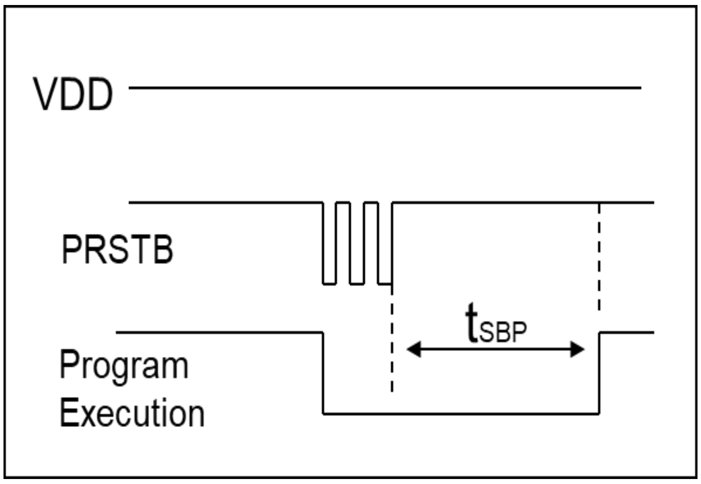


Fig.1: Power-Up Sequence

5.2.1. Timing charts for reset conditions







5.3. Data Memory - SRAM

The access of data memory can be byte or bit operation. Besides data storage, the SRAM data memory is also served as data pointer of indirect access method and the stack memory.

The stack memory is defined in the data memory. The stack pointer is defined in the stack pointer register; the depth of stack memory of each processing unit is defined by the user. The arrangement of stack memory fully flexible and can be dynamically adjusted by the user.

For indirect memory access mechanism, the data memory is used as the data pointer to address the data byte. All the data memory could be the data pointer; it's quite flexible and useful to do the indirect memory access. Since the data width is 8-bit, all the 64 bytes data memory of PGS152 can be accessed by indirect access mechanism.

5.4. Data Memory – EEPROM

EEPROM is also used to store the user data. The different between it and SRAM is EEPROM is non-volatile memory, the data stored in it can still be kept even after the power went off. EEPROM is only accessible by indirect addressing instructions ***STEER*** and ***LDEER***.

Access of the EEPROM is easy. First, user writes the data prepared to program into EEPROM in ***eerl*** registers. Then assign a data pointer for the EEPROM address in SRAM. After the indirect addressing instruction **STEER**, the flag **eermc.6** goes to Busy. By checking this flag, user can monitor whether the EEPROM programming process is done or not. When it is done, user can check the flag **eermc.5** to see if the programming is success. When **eermc.6**=0 but **eermc.5**=1, it means that the writing data timeout, which caused the program failed to be written, so users must be re-written. Alternative, user can just read the EEPROM to verify if the data is correct. When the write/erase data timeout occurs, you can set the ***IHRC_EPM*** register to 0x34, and then perform a write/erase operation again in the enhanced sequence mode. The action is successful, and then change the ***IHRC_EPM*** register setting to 0x3F.

Use the ***LDEER*** instruction to read a byte of data from the EEPROM. Use the ***STEER*** instruction to write a byte data or erase a page (8Bytes) to the EEPROM. Writing a byte is a command to fill 0x5A into **eermc** register before ***STEER*** instruction. Page erase is a command to fill 0xA5 into **eermc** register before the ***STEER*** command. EEPROM data can be overwritten as long as the data bit changes from 1 to 0. If the data bit changes from 0 to 1, you must first erase the page and then write the data.

To read the EEPROM, user issues the indirect addressing instruction ***LDEER***. After a few wait cycles, the data appears in ***eerl*** registers. User can check the Busy flag **eermc.6** to see if the data in ***eerl*** is ready to read.

Notes for EEPROM use:

1. EEPROM must be executed once before EEPROM_Initial macro instructions, recommended execute it after .Adjust_IC macro instruction.

2. IDE provides macro instructions corresponding to EEPROM operations, suggest that user use macro instructions directly to help emulate compatibility. Erase macro instruction **Do_Erase (address)** and write macro instruction **Do_ Program (address)**.

3. IDE provides PGS152 access to EEPROM DemoCode, stored in EE_RW in the IDE's sample project directory. The following routines allow users to refer directly to references. EE_ W's routine writes data directly to the specified address, and the program automatically determines if it can overwrite a write or if it needs to read out data processing, then erases the EEPROM page data and writes it back.

```
//==========================================//
//                  PGS152 EEPROM Byte Write
// Name: EE_W
// Input: adr = eeprom address；data = eeprom data
// Output: non
//==========================================//
void    EE_W (WORD adr, BYTE data)
{
  BYTE        buffer [8];
 @@:
 ldeer  adr;
  .wait0       EERMC.Busy;
  if (EERL == data)   return; //      if data same, end
  #if    _SYS(AT_CHIP)
         A       =       ~ EERL & data;
         if (ZF)
         {       //      can over write
                 EERL =       data;
                 while (1)
                 {
                         Do_Program (adr);
                         .wait0  EERMC.Busy;
                         if (EERMC.Time_Out)      {
                                 IHRC_EPM = 0x34;
                                 continue;      //      Rewritten to right for this? ?
                         }
                         IHRC_EPM = 0x3F;
                         return;
                 }
         }
  #elif  _SYS(AT_ISP_ICE)
         if (EERL == 0xFF)   //      only 0xFF can be written
         {
                 EERL =       data;
                 Do_Program (adr);
                 .wait0  EERMC.Busy;
                 if (EERMC.Time_Out)
                         goto    @B;   //      Not rewritten if checked the same
                 return;
         }
  #else
         A       =       ~ EERL & data;
         if (ZF)
         {       //      can rewritten
                 EERL =       data;
                 while (1)
                 {
                         Do_Program (adr);
                         .wait0  EERMC.Busy;
                         if (! EERMC.Time_Out)     //      Rewritten to right for this? ?
  return;
                 }
```

```
        }
    #endif

    WORD        pnt1    =       adr & 0xFFF8;
    WORD        pnt2    =       buffer;
    BYTE        cnt             =       8;
    do
    {       //      Back up 8 pieces of data
            ldeer   pnt1;
            .wait0  EERMC.Busy;
            *pnt2   =       EERL;
            pnt1$0++;       pnt2$0++;
    } while (--cnt);

    pnt2$0          =       (adr & 7) + buffer;
    *pnt2 =         data;   //      Update Target Data
    //      Erase 8 pieces of data
  @@:   Do_Erase (adr);
    cnt             =       8;
    pnt1$0          =       adr & 0xF8;
    pnt2$0          =       buffer;
    .wait0          EERMC.Busy;
    #if     _SYS(AT_CHIP)
            if (EERMC.Time_Out)
            {
                    IHRC_EPM = 0x34;
                    goto @B;
            }
            IHRC_EPM = 0x3F;
    #else
            if (EERMC.Time_Out)
                    goto @B;
    #endif
    Setup_Program;
    while (1)
    {       //      Write 8 pieces of data
            EERL =          *pnt2;
            Set_Program (pnt1);
            .wait0  EERMC.Busy;

            #if     _SYS(AT_CHIP)
                    if (EERMC.Time_Out)
                    {
                            IHRC_EPM = 0x34;
                            ldeer   pnt1;
                            .wait0  EERMC.Busy;
                            data    =       *pnt2;
                            A               =       ~ EERL & data;
                            if (ZF) continue;
                            goto    @B;
                    }
                    IHRC_EPM = 0x3F;
            #else
                    if (EERMC.Time_Out)
                            continue;
            #endif

            pnt1$0++;       pnt2$0++;
            if (!--cnt)     return;
    }
  }


  //==========================================//
  //                  PGS152 EEPROM Byte Read
  // Name: EE_R
  // Input: adr = eeprom address
  // Output: A = eeprom data
  //==========================================//
  void    EE_R (WORD adr)
  {
    ldeer adr;
    .wait0        EERMC.Busy;
    A      =      EERL;
  }
```

5.5. Oscillator and Clock

There are three oscillator circuits provided by PGS152: external crystal oscillator (EOSC), internal high RC oscillator (IHRC) and internal low RC oscillator (ILRC), and these three oscillators are enabled or disabled by registers eoscr.7, clkmd.4 and clkmd.2 independently. User can choose one of these three oscillators as system clock source and use ***clkmd*** register to target the desired frequency as system clock to meet different applications.

Oscillator Module	Enable/Disable
EOSC	**eoscr**.7
IHRC	**clkmd**.4
ILRC	**clkmd**.2

Table 2: Three oscillation circuits

5.5.1. Internal High RC oscillator and Internal Low RC oscillator

After boot-up, the IHRC and ILRC oscillators are enabled. The frequency of IHRC can be calibrated to eliminate process variation by ***ihrcr*** register; normally it is calibrated to 16MHz. Please refer to the measurement chart for IHRC frequency verse V_{DD} and IHRC frequency verse temperature.

The frequency will vary by process, supply voltage and temperature, please refer to DC specification and do not use for accurate timing application.

5.5.2. Chip calibration

The IHRC frequency and bandgap reference voltage may be different chip by chip due to manufacturing variation, PGS152 provide the IHRC frequency calibration to eliminate this variation, and this function can be selected when compiling user's program and the command will be inserted into user's program automatically. The calibration command is shown as below:

.ADJUST_IC SYSCLK=IHRC/(**p1**), IHRC=(**p2**)MHz, V_{DD}=(**p3**)V;

Where, **p1**=2, 4, 8, 16, 32; In order to provide different system clock.
p2=14 ~ 18; In order to calibrate the chip to different frequency, 16MHz is the usually one.
p3=1.8 ~ 5.5; In order to calibrate the chip under different supply voltage.

5.5.3. IHRC Frequency Calibration and System Clock

During compiling the user program, the options for IHRC calibration and system clock are shown as Table 3:

SYSCLK	CLKMD	IHRCR	Description
○ Set IHRC / 2	= 34h (IHRC / 2)	Calibrated	IHRC calibrated to 16MHz, CLK=8MHz (IHRC/2)
○ Set IHRC / 4	= 14h (IHRC / 4)	Calibrated	IHRC calibrated to 16MHz, CLK=4MHz (IHRC/4)
○ Set IHRC / 8	= 3Ch (IHRC / 8)	Calibrated	IHRC calibrated to 16MHz, CLK=2MHz (IHRC/8)
○ Set IHRC / 16	= 1Ch (IHRC / 16)	Calibrated	IHRC calibrated to 16MHz, CLK=1MHz (IHRC/16)
○ Set IHRC / 32	= 7Ch (IHRC / 32)	Calibrated	IHRC calibrated to 16MHz, CLK=0.5MHz (IHRC/32)
○ Set ILRC	= E4h (ILRC / 1)	Calibrated	IHRC calibrated to 16MHz, CLK=ILRC
○ Disable	No change	No Change	IHRC not calibrated, CLK not changed

Table 3: Options for IHRC Frequency Calibration

Usually, .ADJUST_IC will be the first command after boot up, in order to set the target operating frequency whenever starting the system. The program code for IHRC frequency calibration is executed only one time that occurs in writing the codes into MTP memory; after then, it will not be executed again. If the different option for IHRC calibration is chosen, the system status is also different after boot. The following shows the status of PGS152 for different option:

(1) .ADJUST_IC SYSCLK=IHRC/2, IHRC=16MHz, V_{DD}=5V

After boot up, CLKMD = 0x34：

- IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is enabled
- System CLK = IHRC/2 = 8MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(2) .ADJUST_IC SYSCLK=IHRC/4, IHRC=16MHz, V_{DD}=3.3V

After boot up, CLKMD = 0x14：

- IHRC frequency is calibrated to 16MHz@V_{DD}=3.3V and IHRC module is enabled
- System CLK = IHRC/4 = 4MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(3) .ADJUST_IC SYSCLK=IHRC/8, IHRC=16MHz, V_{DD}=2.5V

After boot up, CLKMD = 0x3C：

- IHRC frequency is calibrated to 16MHz@V_{DD}=2.5V and IHRC module is enabled
- System CLK = IHRC/8 = 2MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(4) .ADJUST_IC SYSCLK=IHRC/16, IHRC=16MHz, V_{DD}=2.5V

After boot up, CLKMD = 0x1C：

- IHRC frequency is calibrated to 16MHz@V_{DD}=2.5V and IHRC module is enabled
- System CLK = IHRC/16 = 1MHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(5) .ADJUST_IC SYSCLK=IHRC/32, IHRC=16MHz, V_{DD}=5V

After boot up, CLKMD = 0x7C：

- IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is enabled
- System CLK = IHRC/32 = 500KHz
- Watchdog timer is disabled, ILRC is enabled, PA5 is in input mode

(6) .ADJUST_IC SYSCLK=ILRC, IHRC=16MHz, V_{DD}=5V

After boot up, CLKMD = 0XE4：

- IHRC frequency is calibrated to 16MHz@V_{DD}=5V and IHRC module is disabled
- System CLK = ILRC
- Watchdog timer is disabled, ILRC is enabled, PA5 is input mode

(7) .ADJUST_IC DISABLE

After boot up, CLKMD is not changed (Do nothing)：

- IHRC is not calibrated
- System CLK = ILRC or IHRC/64 (by Boot-up_Time)
- Watchdog timer is enabled, ILRC is enabled, PA5 is in input mode,

5.5.4. External Crystal Oscillator

If crystal oscillator is used, a crystal is required between X1 and X2. User can choose to use the built-in crystal oscillator capacitor or external resonant capacitor C1/C2 between X1 and X2. Fig.2 shows the hardware connection under this application; the operating frequency of crystal oscillator just can be 32KHz, higher frequency oscillator than 32KHz is NOT supported by PGS152.

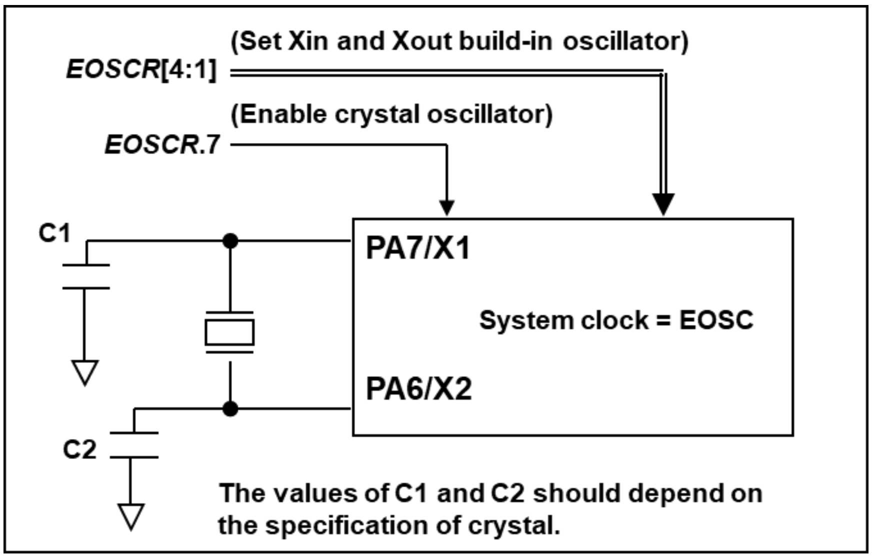

(Set Xin and Xout build-in oscillator)
EOSCR[4:1]
(Enable crystal oscillator)
EOSCR.7
C1
PA7/X1
System clock = EOSC
PA6/X2
C2
The values of C1 and C2 should depend on the specification of crystal.

Fig.2: Connection of crystal oscillator

Besides crystal, the resonant capacitor and options of PGS152 should be fine tuned in *eoscr* (0x0a) register to have good sinusoidal waveform. The *eoscr*.7 is used to enable crystal oscillator module The *eoscr*[4:1] is used to set if the built-in capacitor enable and to set the different driving capacitance to meet the requirement of different build-in capacitor of crystal oscillator:

- *eoscr*[4:3] : It is used to set Xin build-in capacitor for 32KHz crystal oscillator. 00 / 01 / 10 /11: disable/7pF/9.5pF/12.5pF
- *eoscr*[2:1] : It is used to set Xout build-in capacitor for 32KHz crystal oscillator. 00 / 01 / 10 /11: disable/7pF/9.5pF/12.5pF

When using the crystal oscillator, user must pay attention to the stable time of oscillator after enabling it, the stable time of oscillator will depend on frequency, “crystal type”, the resonant capacitor and supply voltage. Before switching the system to the crystal oscillator, user must make sure the oscillator is stable; the reference program is shown as below:

```
void    FPPA0 (void)
{
        . ADJUST_IC   SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V
        $ EOSCR     Enable;             // EOSCR = 0b101_00000;

        $ T16M      EOSC, /1, BIT13;         // while T16.Bit13 0 => 1, Intrq.T16 => 1
                                             // suppose crystal EOSC Is stable
        WORD        count =     0;
        stt16  count;
        Intrq.T16         =     0;
        while(!Intrq.T16) { nop; };          // count from 0x0000 to 0x2000, then trigger INTRQ.T16
        clkmd             =     0xB4;        // switch system clock to EOSC;
        Clkmd.4 = 0;                         // disable IHRC
    ...
}
```

Please notice that the crystal oscillator should be fully turned off before entering the power-down mode, in order to avoid unexpected wake-up event.

5.5.5. System Clock and LVR level

The clock source of system clock comes from EOSC, IHRC and ILRC, the hardware diagram of system clock in the PGS152 is shown as Fig.3.

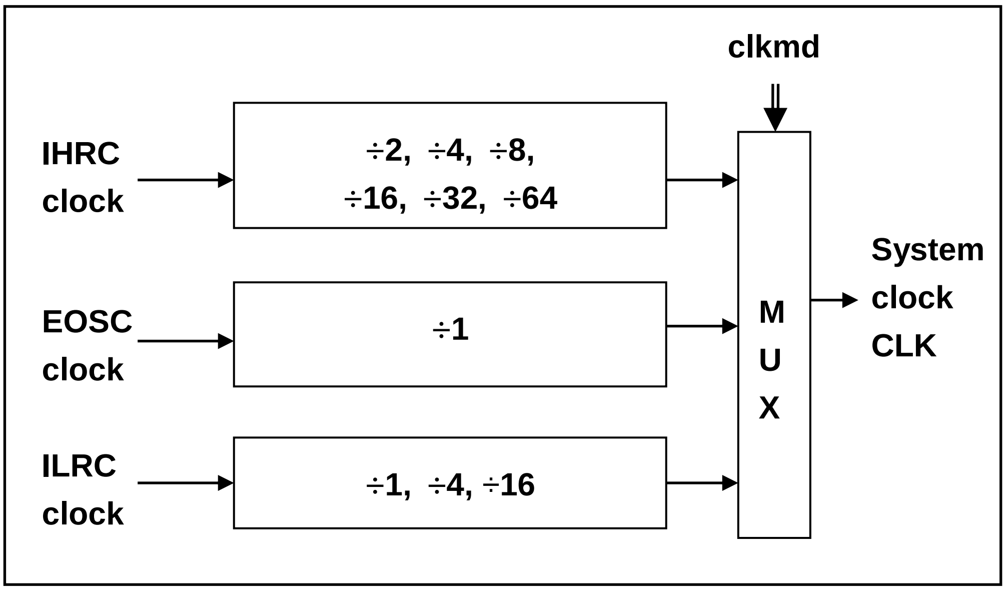


Fig.3: Options of System Clock

User can choose different operating system clock depends on its requirement; the selected operating system clock should be combined with supply voltage and LVR level to make system stable. The LVR level will be selected during compilation. Please refer to Section 4.1.

5.5.6. System Clock Switching

After IHRC calibration, user may want to switch system clock to a new frequency or may switch system clock at any time to optimize the system performance and power consumption. Basically, the system clock of PGS152 can be switched among IHRC, ILRC and EOSC by setting the ***clkmd*** register at any time; system clock will be the new one after writing to ***clkmd*** register immediately. **Please notice that the original clock module can NOT be turned off at the same time as writing command to *clkmd* register.** The examples are shown as below and more information about clock switching, please refer to the “Help” -> “Application Note” -> “IC Introduction” -> “Register Introduction” -> CLKMD”.

Case 1: Switching system clock from ILRC to IHRC/2

```
    …                                //    system clock is ILRC
    CLKMD.4       =    1;            //    turn on IHRC first to improve anti-interference ability
    CLKMD         =    0x34 ;        //    switch to IHRC/2, ILRC CAN NOT be disabled here
    //  CLKMD.2   =    0 ;           //    if need, ILRC CAN be disabled at this time
    …
```

Case 2: Switching system clock from ILRC to EOSC

```
    …                                //    system clock is ILRC
    CLKMD         =    0xA6 ;        //    switch to IHRC, ILRC CAN NOT be disabled here
    CLKMD.2       =    0 ;           //    ILRC CAN be disabled at this time
    …
```

Case 3: Switching system clock from IHRC/2 to ILRC

```
    …                                //    system clock is IHRC/2
    CLKMD         =    0xF4 ;        //    switch to ILRC, IHRC CAN NOT be disabled here
    CLKMD.4       =    0 ;           //    IHRC CAN be disabled at this time
    …
```

Case 4: Switching system clock from IHRC/2 to EOSC

```
    …                                //    system clock is IHRC/2
    CLKMD         =    0XB0 ;        //    switch to EOSC, IHRC CAN NOT be disabled here
    CLKMD.4       =    0 ;           //    IHRC CAN be disabled at this time
    …
```

Case 5: Switching system clock from IHRC/2 to IHRC/4

```
    …                                //    system clock is IHRC/2, ILRC is enabled here
    CLKMD         =    0X14 ;        //    switch to IHRC/4
    …
```

Case 6: System may hang if it is to switch clock and turn off original oscillator at the same time

```
    …                                //    system clock is ILRC
    CLKMD         =    0x30 ;        //    CAN NOT switch clock from ILRC to IHRC/2 and
                                           turn off ILRC oscillator at the same time
```

5.6. Comparator

One hardware comparator is built inside the PGS152; Fig.4 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{\text{internal R}}$ or internal bandgap reference voltage. The two signals to be compared, one is the plus input and the other one is the minus input. For the minus input of comparator can be PA3, PA4, Internal bandgap 1.20 volt, PB7 or $V_{\text{internal R}}$ selected by bit [3:1] of gpcc register, and the plus input of comparator can be PA4 or $V_{\text{internal R}}$ selected by bit 0 of gpcc register. The output result can be enabled to output to PA0 directly, or sampled by Time2 clock (TM2_CLK) which comes from Timer2 module. The output can be also inversed the polarity by bit 4 of ***gpcc*** register, the comparator output can be used to request interrupt service.

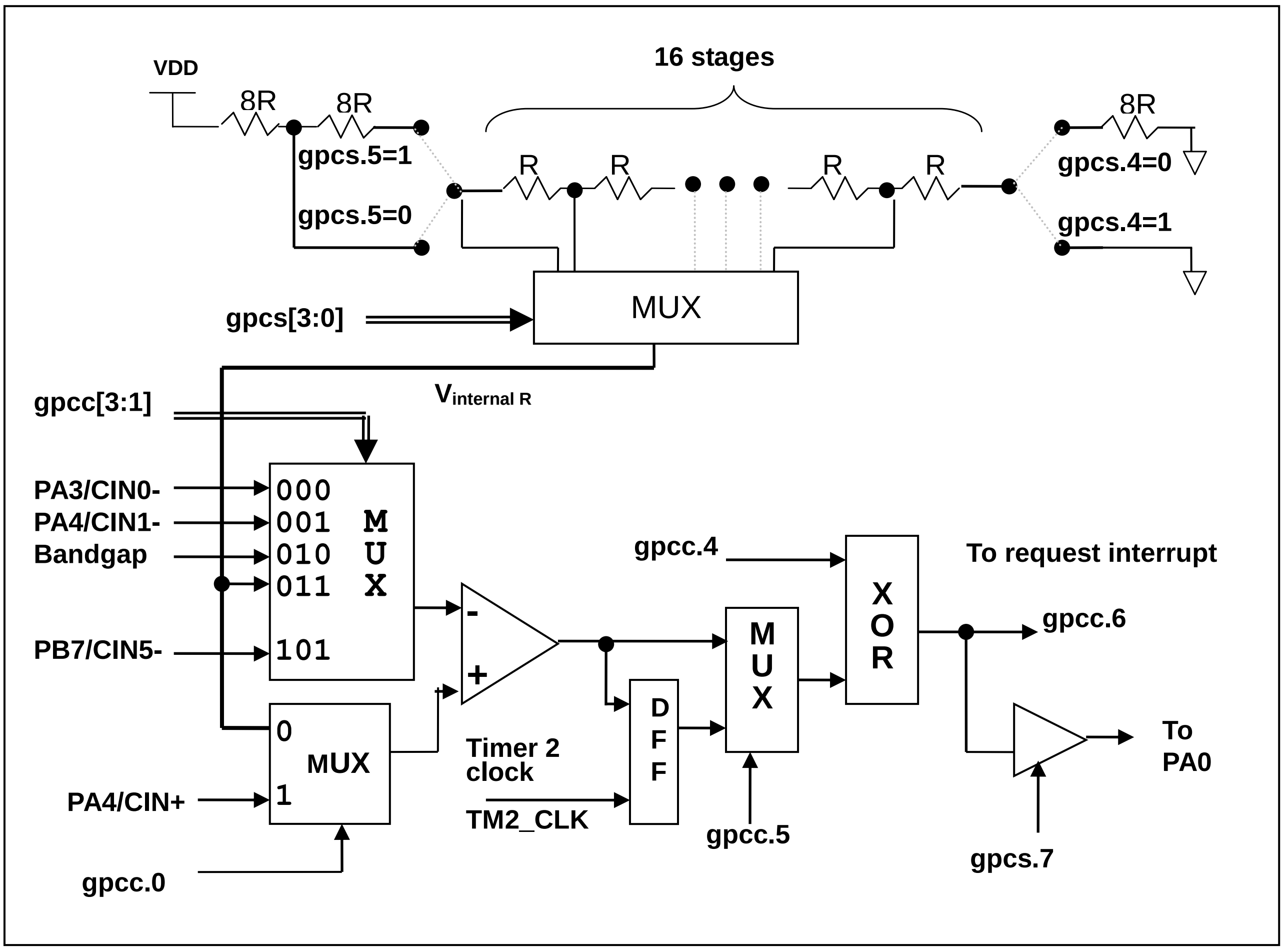
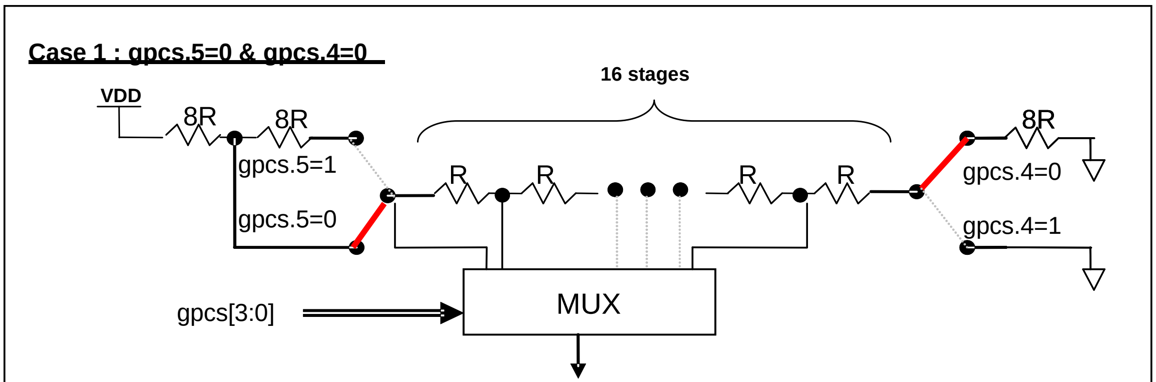


Fig.4: Hardware diagram of comparator

5.6.1 Internal reference voltage ($V_{internal\ R}$)

The internal reference voltage $V_{internal\ R}$ is built by series resistance to provide different level of reference voltage, bit 4 and bit 5 of ***gpcs*** register are used to select the maximum and minimum values of $V_{internal\ R}$ and bit [3:0] of ***gpcs*** register are used to select one of the voltage level which is deivided-by-16 from the defined maximum level to minimum level. Fig.5 to Fig.8 shows four conditions to have different reference voltage $V_{internal\ R}$. By setting the ***gpcs*** register, the internal reference voltage $V_{internal\ R}$ can be ranged from (1/32)*V_{DD} to (3/4)*V_{DD}.

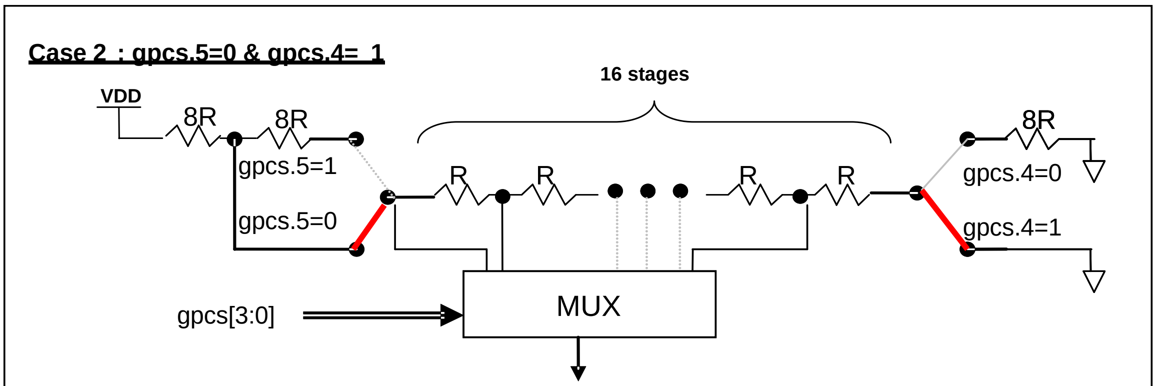


$$V_{internal\ R} = (3/4)\ VDD \sim (1/4)\ VDD + (1/32)\ VDD$$

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{internal\ R} = \frac{1}{4} * VDD + \frac{(n+1)}{32} * VDD, \text{ n = gpcs[3:0] in decimal}$$

Fig.5: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=0

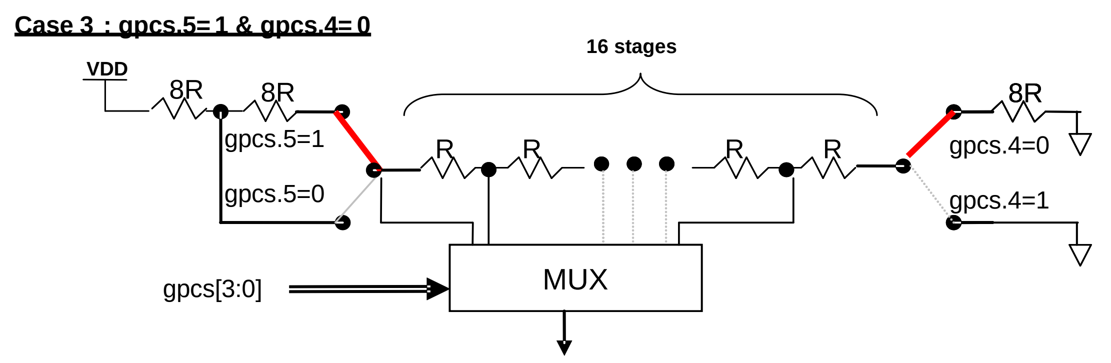


$$V_{internal\ R} = (2/3)\ VDD \sim (1/24)\ VDD$$

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{internal\ R} = \frac{(n+1)}{24} * VDD, \text{ n = gpcs[3:0] in decimal}$$

Fig.6: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=1

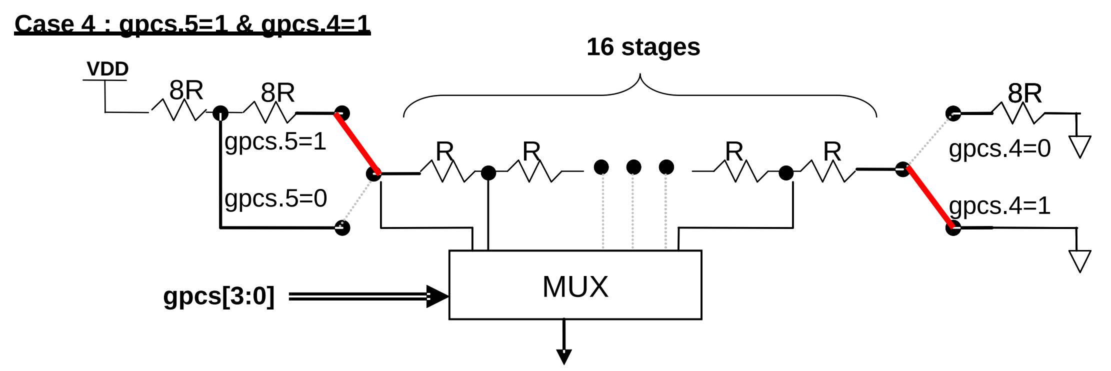


$$V_{\text{internal R}} = (3/5)\ \text{VDD} \sim (1/5)\ \text{VDD} + (1/40)\ \text{VDD}$$

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{\text{internal R}} = \frac{1}{5} * \text{VDD} + \frac{(n+1)}{40} * \text{VDD},\ n = \text{gpcs[3:0] in decimal}$$

Fig.7: $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=0



$$V_{\text{internal R}} = (1/2)\ \text{VDD} \sim (1/32)\ \text{VDD}$$

@ gpcs[3:0] = 1111 ~ gpcs[3:0] = 0000

$$V_{\text{internal R}} = \frac{(n+1)}{32} * \text{VDD},\ n = \text{gpcs[3:0] in decimal}$$

Fig.8: $V_{\text{internal R}}$ hardware connection if gpcs.5=1 and gpcs.4=1

5.6.2 Using the comparator

Case 1:

Choosing PA3 as minus input and $V_{\text{internal R}}$ with (18/32)*V_{DD} voltage level as plus input. $V_{\text{internal R}}$ is configured as the above Figure "gpcs[5:4] = 2b'00" and gpcs [3:0] = 4b'1001 (n=9) to have $V_{\text{internal R}}$ = (1/4)*V_{DD} + [(9+1)/32]*V_{DD} = [(9+9)/32]*V_{DD} = (18/32)*V_{DD}.

```
gpcs   = 0b0_0_00_1001;         // Vinternal R = VDD*(18/32)
gpcc   = 0b1_0_0_0_000_0;       // enable comp, - input: PA3, + input: Vinternal R
padier = 0bxxxx_0_xxx;          // disable PA3 digital input to prevent leakage current
```

or

```
$ GPCS      VDD*18/32;
$ GPCC      Enable, N_PA3, P_R;     // - input: N_xx， + input: P_R(Vinternal R)
PADIER = 0bxxxx_0_xxx;
```

Case 2:

Choosing $V_{\text{internal R}}$ as minus input with (22/40)*V_{DD} voltage level and PA4 as plus input, the comparator result will be inversed and then output to PA0. $V_{\text{internal R}}$ is configured as the above Figure "gpcs[5:4] = 2b'10" and gpcs [3:0] = 4b'1101 (n=13) to have $V_{\text{internal R}}$ = (1/5)*V_{DD} + [(13+1)/40]*V_{DD} = [(13+9)/40]*V_{DD} = (22/40)*V_{DD}.

```
gpcs   = 0b1_0_10_1101;         // output to PA0, Vinternal R = VDD*(22/40)
gpcc   = 0b1_0_0_1_011_1;       // Inverse output, - input: Vinternal R, + input: PA4
padier = 0bxxx_0_xxxx;          // disable PA4 digital input to prevent leakage current
```

or

```
$ GPCS      Output, VDD*22/40;
$ GPCC      Enable, Inverse, N_R, P_PA4;   // - input: N_R(Vinternal R)， + input: P_xx
PADIER = 0bxxx_0_xxxx;
```

Note: When selecting output to PA0 output, GPCS will affect the PA3 output function in ICE. Though the IC is fine, be careful to avoid this error during emulation.

5.6.3 Using the comparator and bandgap 1.20V

The internal bandgap module can provide 1.20 volt, it can measure the external supply voltage level. The bandgap 1.20 volt is selected as minus input of comparator and $V_{internal\ R}$ is selected as plus input, the supply voltage of $V_{internal\ R}$ is V_{DD}, the V_{DD} voltage level can be detected by adjusting the voltage level of $V_{internal\ R}$ to compare with bandgap. If N (gpcs[3:0] in decimal) is the number to let $V_{internal\ R}$ closest to bandgap 1.20 volt, the supply voltage V_{DD} can be calculated by using the following equations:

For using Case 1: V_{DD} = [32 / (N+9)] * 1.20 volt ;

For using Case 2: V_{DD} = [24 / (N+1)] * 1.20 volt ;

For using Case 3: V_{DD} = [40 / (N+9)] * 1.20 volt ;

For using Case 4: V_{DD}= [32 / (N+1)] * 1.20 volt ;

Case 1:

```
$ GPCS VDD*12/40;              //    4.0V * 12/40 = 1.2V
$ GPCC Enable, BANDGAP, P_R;   //   - input: BANDGAP, + input: P_R(Vinternal R)
....
if   (GPC_Out)                  //  or GPCC.6
{                              //    when VDD > 4V
}
else
{                              //    when VDD < 4V
}
```

5.7. 16-bit Timer (Timer16)

A 16-bit hardware timer (Timer16) is implemented in the PGS152, the clock sources of Timer16 may come from system clock (CLK), clock of external crystal oscillator (EOSC), internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), PA4 and PA0, a multiplex is used to select clock output for the clock source. Before sending clock to the counter16, a pre-scaling logic with divided-by-1, 4, 16, and 64 is used for wide range counting. The 16-bit counter performs up-counting operation only, the counter initial values can be stored from memory by *stt16* instruction and the counting values can be loaded to memory by *ldt16* instruction. A selector is used to select the interrupt condition of Timer16, whenever overflow occurs, the Timer16 interrupt can be triggered. The hardware diagram of Timer16 is shown as Fig.9. The interrupt source of Timer16 comes from one of bit 8 to 15 of 16-bit counter, and the interrupt type can be rising edge trigger or falling edge trigger which is specified in the bit 5 of ***integs*** register (address 0x0C).

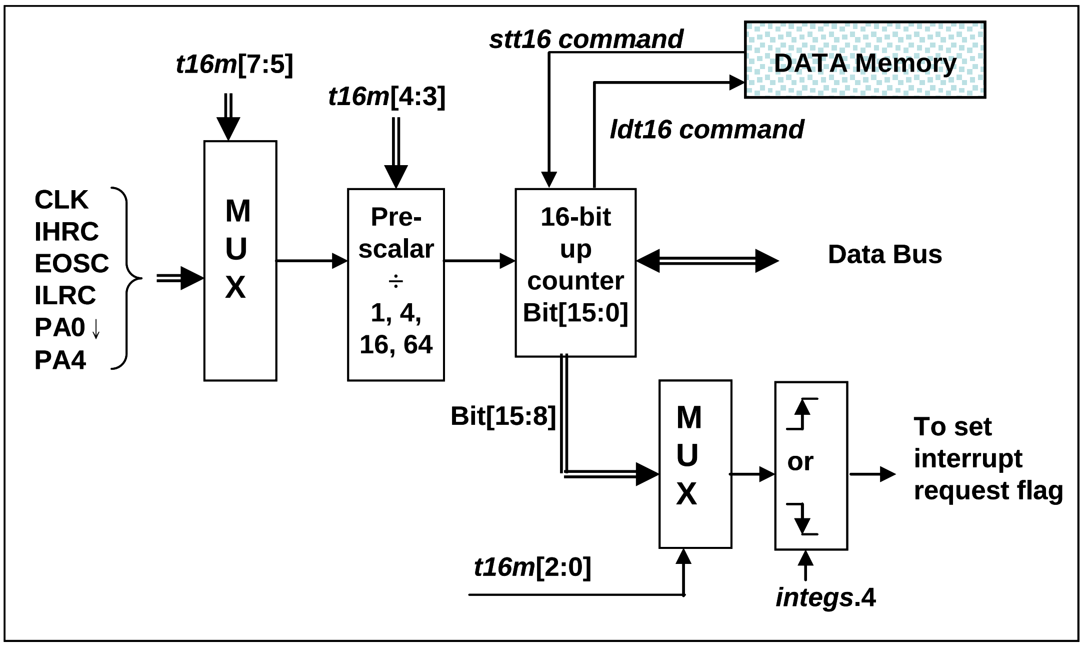


Fig.9: Hardware diagram of Timer16

When using the Timer16, the syntax for Timer16 has been defined in the .INC file. There are three parameters to define the Timer16; 1st parameter is used to define the clock source of Timer16, 2nd parameter is used to define the pre-scalar and the last one is to define the interrupt source. The detail description is shown as below:

```
T16M        IO_RW           0x06
      $ 7~5: STOP, SYSCLK, X, PA4_F, IHRC, EOSC, ILRC, PA0_F    // 1st par.
      $ 4~3: /1, /4, /16, /64                                   // 2nd par.
      $ 2~0: BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15   // 3rd par.
```

User can define the parameters of T16M based on system requirement, some examples are shown below and more examples please refer to “Help → Application Note → IC Introduction → Register Introduction → T16M” in IDE utility.

$ T16M SYSCLK, /64, BIT15;
// choose (SYSCLK/64) as clock source, every 2^16 clock to set INTRQ.2=1
// if using System Clock = IHRC / 2 = 8 MHz
// SYSCLK/64 = 8 MHz/64 = 125KHz, about every 512 mS to generate INTRQ.2=1

$ T16M EOSC, /1, BIT13;
// choose (EOSC/1) as clock source, every 2^14 clocks to generate INTRQ.2=1
// if EOSC=32768 Hz, 32768 Hz/(2^14) = 2Hz, every 0.5S to generate INTRQ.2=1

$ T16M PA0_F, /1, BIT8;
// choose PA0 as clock source, every 2^9 to generate INTRQ.2=1
// receiving every 512 times PA0 to generate INTRQ.2=1

$ T16M STOP;
// stop Timer16 counting

If Timer16 is operated at free running, the frequency of interrupt can be described as below:

$$F_{INTRQ\_T16M} = F_{clock\ source} \div P \div 2^{n+1}$$

Where, F is the frequency of selected clock source to Timer16;
P is the selection of t16m [4:3]; (1, 4, 16, 64)
N is the nth bit selected to request interrupt service, for example: n=10 if bit 10 is selected.

5.8. 8-bit Timer (Timer2) with PWM generation

An 8-bit hardware timer (Timer2) with PWM generation is implemented in the PGS152. Please refer to Fig.10 shown the hardware diagram of Timer2, the clock sources of Timer2 may come from system clock, internal high RC oscillator (IHRC), internal low RC oscillator (ILRC), external crystal oscillator (EOSC), PA0, PA4, PB0 and comparator. Bit [7:4] of register tm2c are used to select the clock of Timer2. If IHRC is selected for Timer2 clock source, the clock sent to Timer2 will keep running when using ICE in halt state. The output of Timer2 can be sent to pin PA3 or PB0, depending on bit [3:2] of tm2c register. A clock pre-scaling module is provided with divided-by- 1, 4, 16, and 64 options, controlled by bit [6:5] of tm2s register; one scaling module with divided-by-1~31 is also provided and controlled by bit [4:0] of tm2s register. In conjunction of pre-scaling function and scaling function, the frequency of Timer2 clock (TM2_CLK) can be wide range and flexible.

The Timer2 counter performs 8-bit up-counting operation only; the counter values can be set or read back by tm2ct register. The 8-bit counter will be clear to zero automatically when its values reach for upper bound register in period mode. The upper bound register is used to define the period of timer or duty of PWM. There are two operating modes for Timer2: period mode and PWM mode; period mode is used to generate periodical output waveform or interrupt event; PWM mode is used to generate PWM output waveform with optional 6-bit, 7-bit or 8-bit PWM resolution, Fig.11 shows the timing diagram of Timer2 for both period mode and PWM mode.

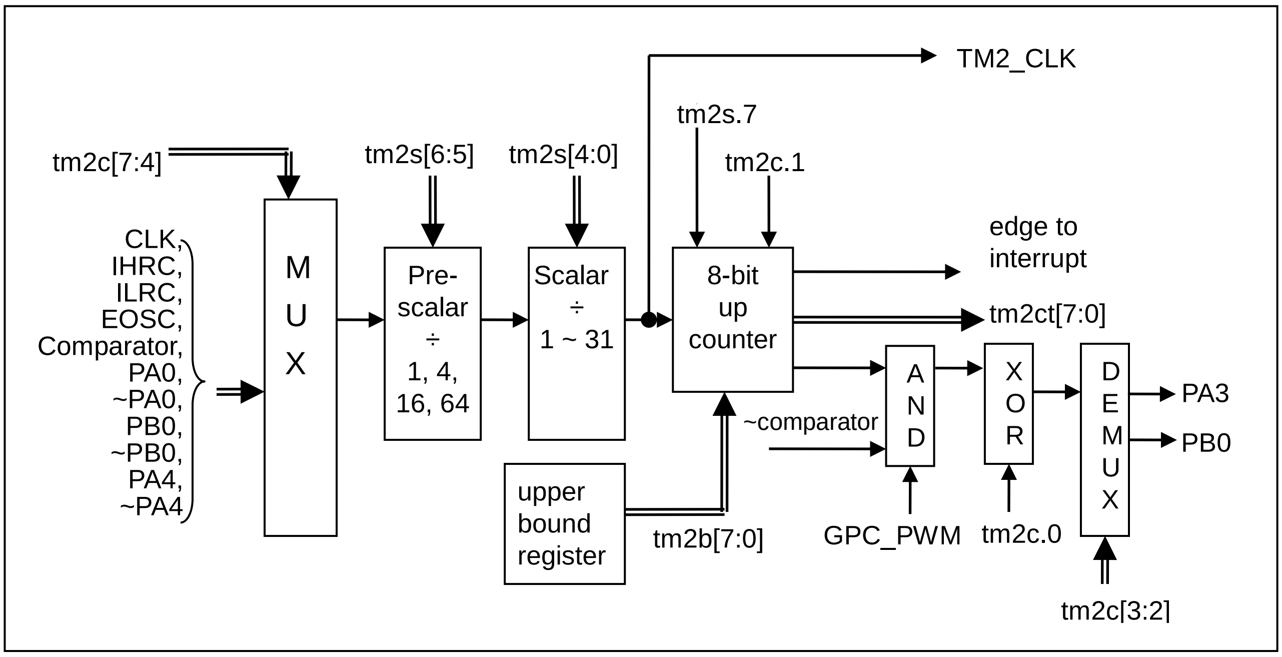
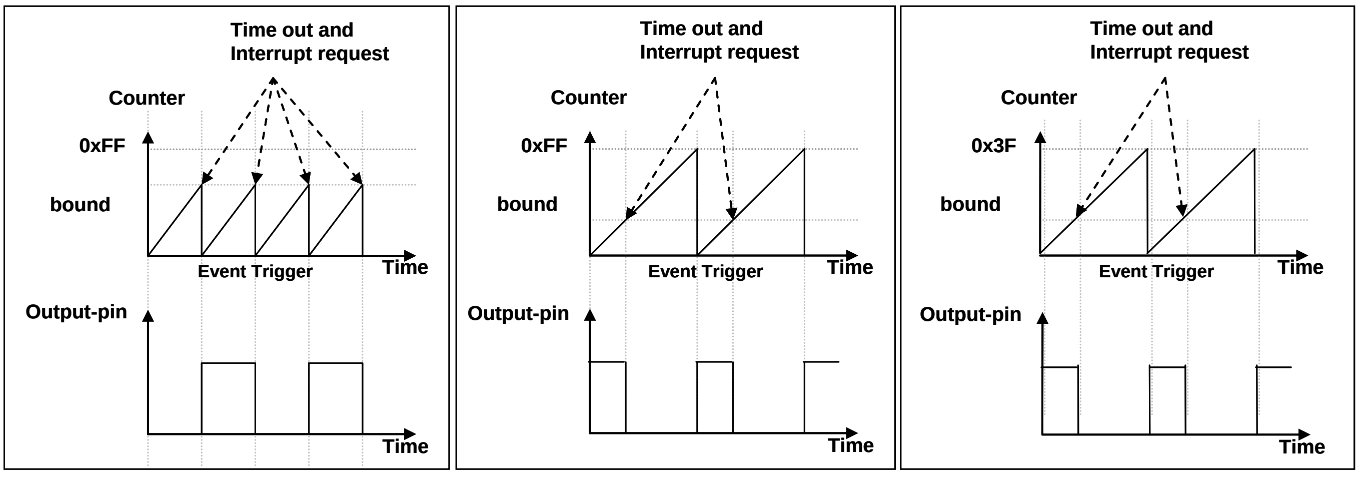


Fig.10: Timer2 hardware diagram



Mode 0 – Period Mode **Mode 1 – 8-bit PWM Mode** **Mode 1 – 6-bit PWM Mode**

Fig.11: Timing diagram of Timer2 in period mode and PWM mode (tm2c.1=1)

A Code Option GPC_PWM is for the applications which need the generated PWM waveform to be controlled by the comparator result. If the Code Option GPC_PWM is selected, the PWM output stops while the comparator output is 1 and then the PWM output turns on while the comparator output goes back to 0, as shown in Fig. 12.

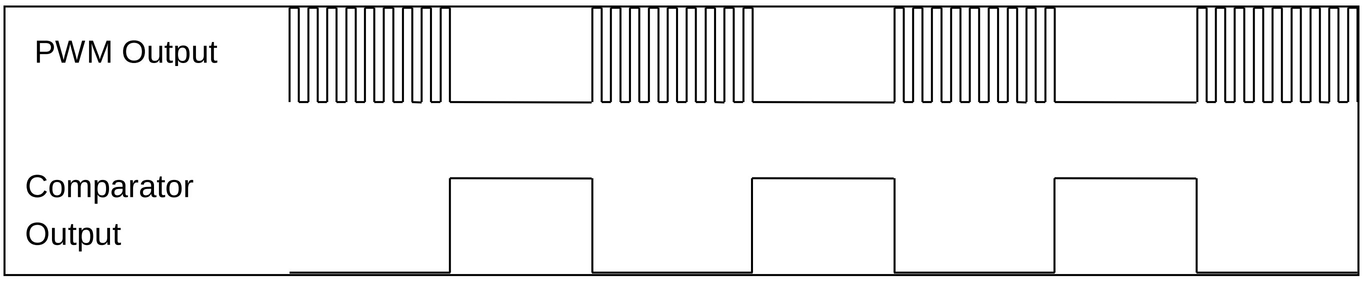


Fig.12：Comparator controls the output of PWM waveform

5.8.1. Using the Timer2 to generate periodical waveform

If periodical mode is selected, the duty cycle of output is always 50%; its frequency can be summarized as below:

Frequency of Output = Y ÷ [2 × (K+1) × S1 × (S2+1)]

Where, Y = tm2c[7:4] : frequency of selected clock source
K = tm2b[7:0] : bound register in decimal
S1 = tm2s[6:5] : pre-scalar (S1=1, 4, 16, 64)
S2 = tm2s[4:0] : scalar register in decimal (S2=0 ~ 31)

Example 1:

tm2c = 0b0001_1000, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0000_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ [2 × (127+1) × 1 × (0+1)] = 31.25KHz

Example 2:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0111_1111, K=127

tm2s[7:0] = 0b0111_11111, S1=64 , S2 = 31

➔ frequency of output = 8MHz ÷ (2 ✕ (127+1) ✕ 64 ✕ (31+1)) =15.25Hz

Example 3:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_1111, K=15

tm2s = 0b0000_00000, S1=1, S2=0

➔ frequency of output = 8MHz ÷ (2 ✕ (15+1) ✕ 1 ✕ (0+1)) = 250KHz

Example 4:

tm2c = 0b0001_1000, Y=8MHz

tm2b = 0b0000_0001, K=1

tm2s = 0b0000_00000, S1=1, S2=0

➔ frequency of output = 8MHz ÷ (2 ✕ (1+1) ✕ 1 ✕ (0+1)) =2MHz

The sample program for using the Timer2 to generate periodical waveform from PA3 is shown as below:

```
Void   FPPA0 (void)
{
       . ADJUST_IC    SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
       …
       tm2ct = 0x00;
       tm2b = 0x7f;
       tm2s = 0b0_00_00001;          //     8-bit PWM, pre-scalar = 1, scalar = 2
       tm2c = 0b0001_10_0_0;         //     system clock, output=PA3, period mode
       while(1)
       {
              nop;
       }
}
```

5.8.2. Using the Timer2 to generate 8-bit PWM waveform

If 8-bit PWM mode is selected, it should set ***tm2c***[1]=1 and ***tm2s***[7]=0, the frequency and duty cycle of output waveform can be summarized as below:

Frequency of Output = Y ÷ [256 × S1 × (S2+1)]

Duty of Output = [(K+1) ÷ 256]×100%

Where, Y = tm2c[7:4] : frequency of selected clock source

K = tm2b[7:0] : bound register in decimal

S1= tm2s[6:5] : pre-scalar (S1=1, 4, 16, 64)

S2 = tm2s[4:0] : scalar register in decimal (S2=0 ~ 31)

<u>Example 1:</u>

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0000_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 ✕ 1 ✕ (0+1)) = 31.25KHz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

<u>Example 2:</u>

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0111_1111, K=127
tm2s = 0b0111_11111, S1=64, S2=31
➔ frequency of output = 8MHz ÷ (256 ✕ 64 ✕ (31+1)) = 15.25Hz
➔ duty of output = [(127+1) ÷ 256] × 100% = 50%

<u>Example 3:</u>

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b1111_1111, K=255
tm2s = 0b0000_00000, S1=1, S2=0
➔ PWM output keep high
➔ duty of output = [(255+1) ÷ 256] × 100% = 100%

<u>Example 4:</u>

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_1001, K = 9
tm2s = 0b0000_00000, S1=1, S2=0
➔ frequency of output = 8MHz ÷ (256 ✕ 1 ✕ (0+1)) = 31.25KHz
➔ duty of output = [(9+1) ÷ 256] × 100% = 3.9%

The sample program for using the Timer2 to generate PWM waveform from PA3 is shown as below:

```
void    FPPA0 (void)
{
    .ADJUST_IC      SYSCLK=IHRC/2, IHRC=16MHz, VDD=5V
    wdreset;
    tm2ct = 0x00;
    tm2b = 0x7f;
    tm2s = 0b0_00_00001;            //      8-bit PWM, pre-scalar = 1, scalar = 2
    tm2c = 0b0001_10_1_0;           //      system clock, output=PA3, PWM mode
    while(1)
    {
        nop;
    }
}
```

5.8.3. Using the Timer2 to generate 6-bit PWM waveform

If 6-bit PWM mode is selected, it should set ***tm2c***[1]=1 and ***tm2s***[7]=1, the frequency and duty cycle of output waveform can be summarized as below:

$$\textbf{Frequency of Output} = Y \div [64 \times S1 \times (S2+1)]$$
$$\textbf{Duty of Output} = [(K+1) \div 64] \times 100\%$$

Where,	tm2c[7:4] = Y : frequency of selected clock source
	tm2b[7:0] = K : bound register in decimal
	tm2s[6:5] = S1 : pre-scalar (S1=1, 4, 16, 64)
	tm2s[4:0] = S2 : scalar register in decimal (S2=0 ~ 31)

Users can set Timer2 to be 7-bit PWM mode instead of 6-bit mode by using ***TM2_Bit*** code option. At that time, the calculation factors of the above equations become 128 instead of 64.

Example 1:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0001_1111, K=31

tm2s = 0b1000_00000, S1=1, S2=0

➔ frequency of output = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz

➔ duty = [(31+1) ÷ 64] × 100% = 50%

Example 2:

tm2c = 0b0001_1010, Y=8MHz

tm2b = 0b0001_1111, K=31

tm2s = 0b1111_11111, S1=64, S2=31

➔ frequency of output = 8MHz ÷ (64 × 64 × (31+1)) = 61.03 Hz

➔ duty of output = [(31+1) ÷ 64] × 100% = 50%

Example 3:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0011_1111, K=63
tm2s = 0b1000_00000, S1=1, S2=0
➔ PWM output keep high
➔ duty of output = [(63+1) ÷ 64] × 100% = 100%

Example 4:

tm2c = 0b0001_1010, Y=8MHz
tm2b = 0b0000_0000, K=0
tm2s = 0b1000_00000, S1=1, S2=0
➔ frequency = 8MHz ÷ (64 × 1 × (0+1)) = 125KHz
➔ duty = [(0+1) ÷ 64] × 100% =1.5%

5.9. 11-bit PWM Generators

One set of triple 11-bit SuLED (Super LED) hardware PWM generator is implemented in the PGS152. It consists of three PWM generators (LPWMG0, LPWMG1 & LPWMG2). Their individual outputs are listed as below:

- LPWMG0 – PA0
- LPWMG1 – PA4
- LPWMG2 – PA3, PA5

Note: 5S-I-S01/2(B) doesn't support the function of the set of 11-bit SuLED hardware PWM generators.

5.9.1. PWM Waveform

A PWM output waveform (Fig.13) has a time-base (T_{Period} = Time of Period) and a time with output high level (Duty Cycle). The frequency of the PWM output is the inverse of the period ($f_{PWM} = 1/T_{Period}$).

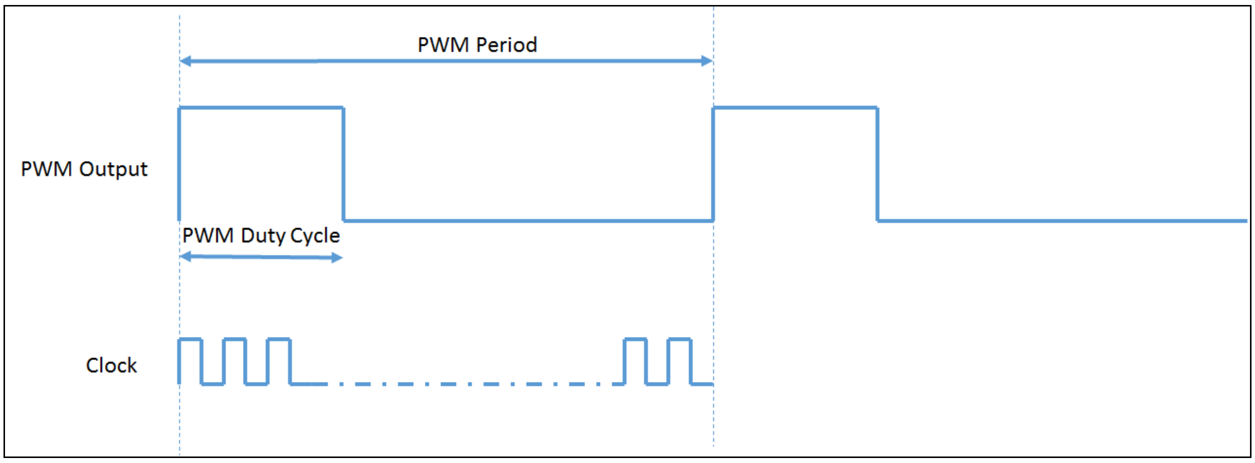


Fig.13: PWM Output Waveform

5.9.2. Hardware Diagram

Fig.14 shows the hardware diagram of the whole set of SuLED 11-bit hardware PWM generators. Those three PWM generators use a common Up-Counter and clock source selector to create the time base, and so the start points (the rising edge) of the PWM cycle are synchronized. The clock source can be IHRC or system clock. The PWM signal output pins that can be selected via ***lpwmgxc[3:1]*** register selection. The PWM output Enable are control by GPC, that can be selected via ***lpwmgxc.7***. The period of PWM waveform is defined by the common PWM upper bound high and low registers, and the duty cycle of individual PWM waveform is defined by the individual set in the PWM duty high and low registers.

The additional OR and XOR logic of LPWMG0 channel is used to create the complementary switching waveforms with dead zone control. Selecting code option GPC_TMx_LPWM can also control the generated PWM waveform by the comparator result.

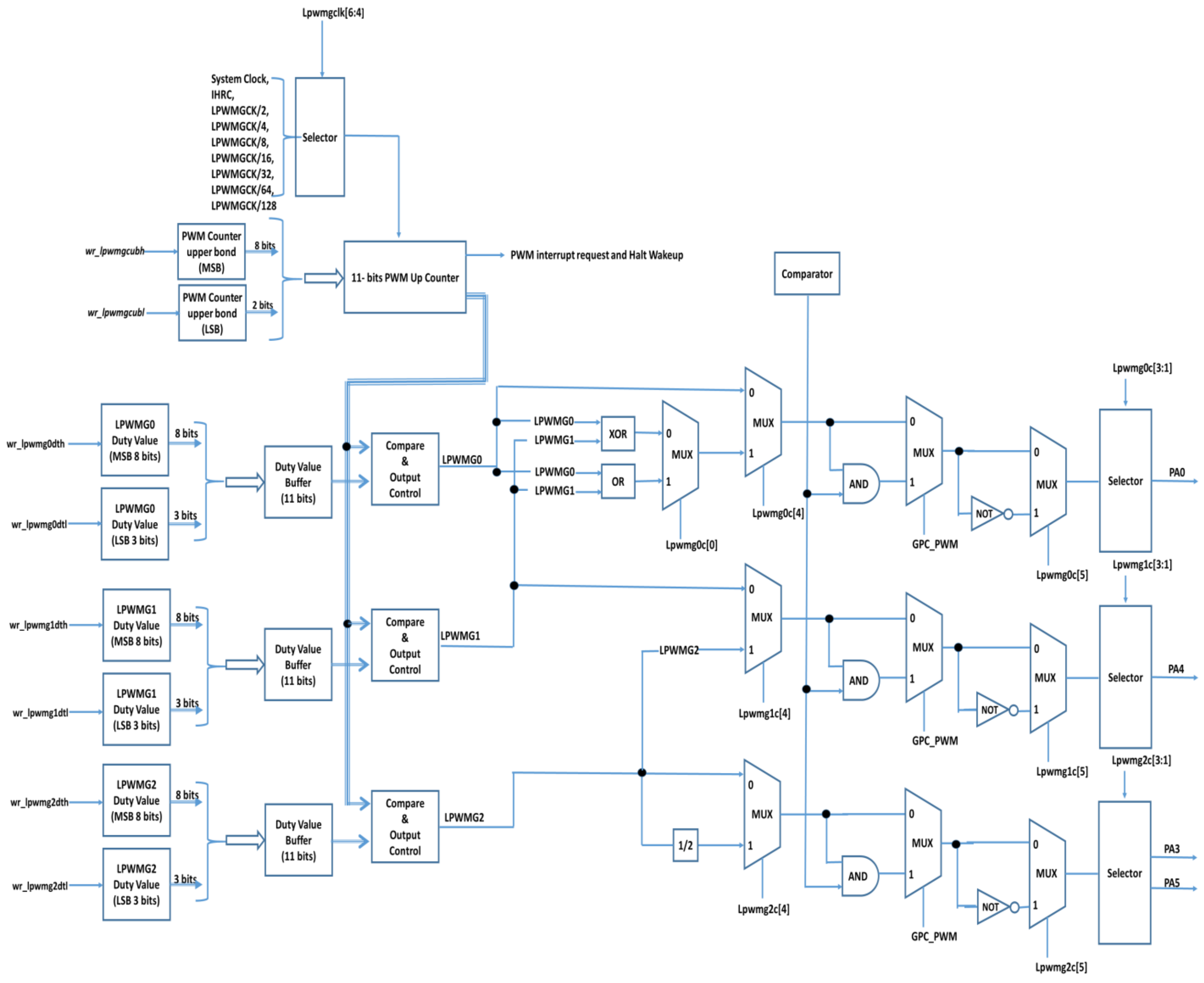

Lpwmgclk[6:4]
System Clock, IHRC, LPWMGCK/2, LPWMGCK/4, LPWMGCK/8, LPWMGCK/16, LPWMGCK/32, LPWMGCK/64, LPWMGCK/128
Selector
wr_lpwmgcubh
PWM Counter upper bond (MSB)
8 bits
wr_lpwmgcubl
PWM Counter upper bond (LSB)
2 bits
11- bits PWM Up Counter
PWM interrupt request and Halt Wakeup
Comparator
wr_lpwmg0dth
LPWMG0 Duty Value (MSB 8 bits)
wr_lpwmg0dtl
LPWMG0 Duty Value (LSB 3 bits)
3 bits
Duty Value Buffer (11 bits)
Compare & Output Control
LPWMG0
LPWMG1
XOR
OR
MUX
Lpwmg0c[0]
Lpwmg0c[4]
AND
GPC_PWM
NOT
Lpwmg0c[5]
Lpwmg0c[3:1]
PA0
wr_lpwmg1dth
LPWMG1 Duty Value (MSB 8 bits)
wr_lpwmg1dtl
LPWMG1 Duty Value (LSB 3 bits)
LPWMG2
Lpwmg1c[4]
Lpwmg1c[5]
Lpwmg1c[3:1]
PA4
wr_lpwmg2dth
LPWMG2 Duty Value (MSB 8 bits)
wr_lpwmg2dtl
LPWMG2 Duty Value (LSB 3 bits)
1/2
Lpwmg2c[4]
Lpwmg2c[5]
Lpwmg2c[3:1]
PA3
PA5

Fig.14: Hardware diagram of whole set of triple SuLED 11-bit PWM generators

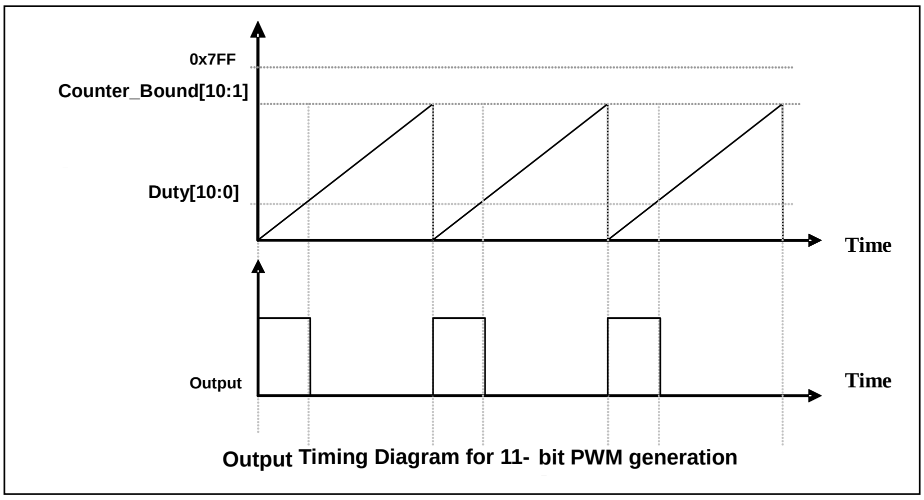


Fig.15: Output Timing Diagram of 11-bit PWM Generator

5.9.3. Equations for 11-bit PWM Generator

$$\textbf{PWM Frequency } F_{PWM} = F_{\text{clock source}} \div [\, P \times (CB10\_1 + 1)\,]$$

$$\textbf{PWM Duty(in time)} = (1 / F_{PWM}) \times (\, DB10\_1 + DB0 \times 0.5 + 0.5) \div (CB10\_1 + 1)$$

$$\textbf{PWM Duty(in percentage)} = (\, DB10\_1 + DB0 \times 0.5 + 0.5) \div (CB10\_1 + 1) \times 100\%$$

Where,

P = *LPWMGCLK* [6:4]; pre-scalar **P**=1,2,4,8,16,32,64,128

DB10_1 = Duty_Bound[10:1] = {*LPWMGxDTH[7:0], LPWMGxDTL[7:6]*}, duty bound

DB0 = Duty_Bound[0] = *LPWMGxDTL[5]*

CB10_1 = Counter_Bound[10:1] = {*LPWMGCUBH[7:0], LPWMGCUBL[7:6]*}, counter bound

5.9.4. PWM Waveforms with Complementary Dead Zones

Based on the specific 11 bit PWM architecture of PGS152, here we employ PWM2 output and PWM0 inverse output after PWM0 xor PWM1 to generate two PWM waveforms with complementary dead zones.

Example program is as follows:

```
#define   dead_zone        10         //    dead time = 10% * (1/PWM_Frequency) us
#define   PWM_Pulse        50         //    set 50% as PWM duty cycle

#define   PWM_Pulse_1      35         //    set 35% as PWM duty cycle
#define   PWM_Pulse_2      60         //    set 60% as PWM duty cycle
#define   switch_time      400*2      //    adjusting switch time
// Note: To avoid noise, switch_time must be a multiple of PWM period. In this example PWM period = 400us,
// so switch_time = 400*2 us.
```

```
void  FPPA0 (void)
{
.ADJUST_IC        SYSCLK=IHRC/16, IHRC=16MHz, VDD=5V;
//******* Generating fixed duty cycle waveform   ************************
//------    Set the counter upper bound and duty cycle -----------------
LPWMG0DTL      =       0x00;
LPWMG0DTH      =       PWM_Pulse + dead_zone;

LPWMG1DTL      =       0x00;
LPWMG1DTH      =       dead_zone;          // After LPWMG0 xor LPWMG, PWM duty cycle=PWM_Pulse%

LPWMG2DTL        =     0x00;
LPWMG2DTH      =       PWM_Pulse + dead_zone*2;

LPWMGCUBL      =       0x00;
LPWMGCUBH      =       100;
//---- Configure clock and pre-scalar ------------------
$ LPWMGCLK      Enable, /1, sysclk;
//------- Output control ----------------------------
$ LPWMG0C       Inverse,PWM_Gen,PA0,gen_xor;     //     After LPWMG0 xor LPWMG,
                                                 //     output the inversed waveform through PA0
$ LPWMG1C       LPWMG1,disable;        //     disable LPWMG1 output
$ LPWMG2C      PA3;                    //     output LPWMG2 waveform through PA3

while(1)
{
      //******** Switching duty cycle ********************************
      // To avoid the possible instant disappearance of dead zone, user should comply with the following
      // instruction sequence.
      // When increase the duty cycle:   50%/60%   →   35%
      LPWMG0DTL       =       0x00;
      LPWMG0DTH       =       PWM_Pulse_1 + dead_zone;
      LPWMG2DTL       =       0x00;
      LPWMG2DTH       =       PWM_Pulse_1 + dead_zone*2;
      .delay      switch_time
      // When decrease the duty cycle:   35% → 60%
      LPWMG2DTL       =       0x00;
      LPWMG2DTH       =       PWM_Pulse_2 + dead_zone*2;
      LPWMG0DTL       =       0x00;
      LPWMG0DTH       =       PWM_Pulse_2 + dead_zone;
      .delay      switch_time
}
}
```

The following figures show the waveforms at different condition.

1. The PWM waveform in a fixed-duty cycle:

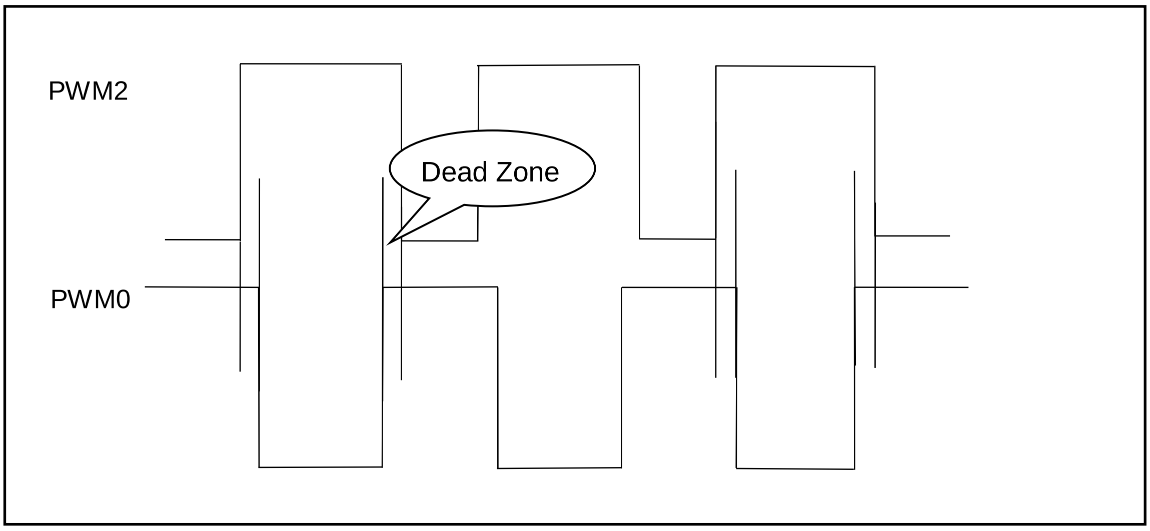


Fig.16: Complementary PWM waveform with dead zones

2. PWM waveform when switching two duty cycles:

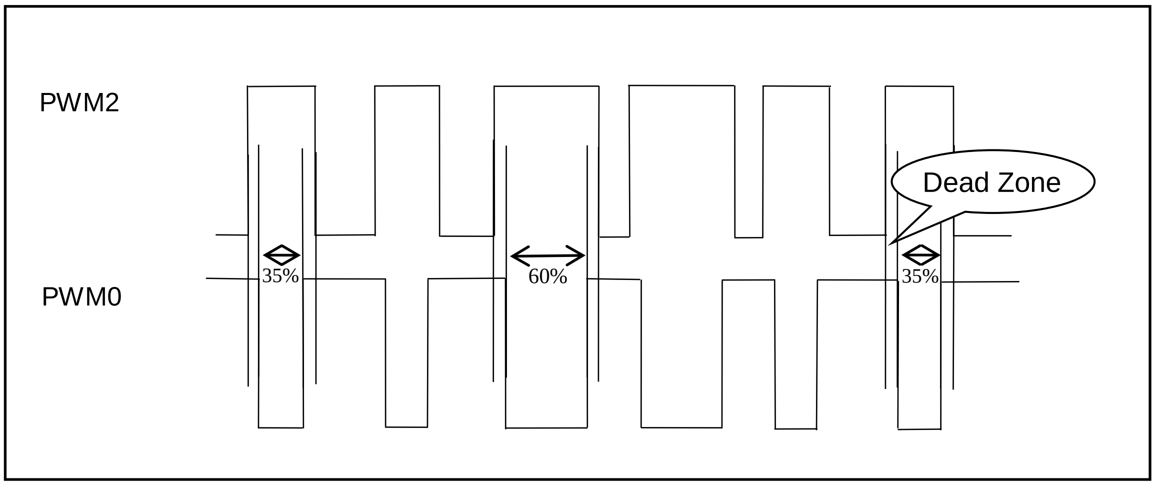


Fig.17: Complementary PWM waveform with dead zones

User can find that above example only provides dead zone where PWM are both in high. If need dead zone where PWM are both in low, you can realize it by resetting each control register's Inverse like:

$ LPWMG0C PWM_Gen,PA0,gen_xor;

$ LPWMG2C Inverse, PA3;.

5.10. WatchDog Timer

The watchdog timer (WDT) is a counter with clock coming from ILRC. WDT can be cleared by power-on-reset or by command ***wdreset*** at any time. There are four different timeout periods of watchdog timer to be chosen by setting the ***misc*** register, it is:

- 8k ILRC clocks period if register misc[1:0]=00 (default)
- 16k ILRC clocks period if register misc[1:0]=01
- 64k ILRC clocks period if register misc[1:0]=10
- 256k ILRC clocks period if register misc[1:0]=11

The frequency of ILRC may drift a lot due to the variation of manufacture, supply voltage and temperature; user should reserve guard band for save operation. Besides, the watchdog period will also be shorter than expected after Reset or Wakeup events. It is suggested to clear WDT by wdreset command after these events to ensure enough clock periods before WDT timeout.

When WDT is timeout, PGS152 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.18.

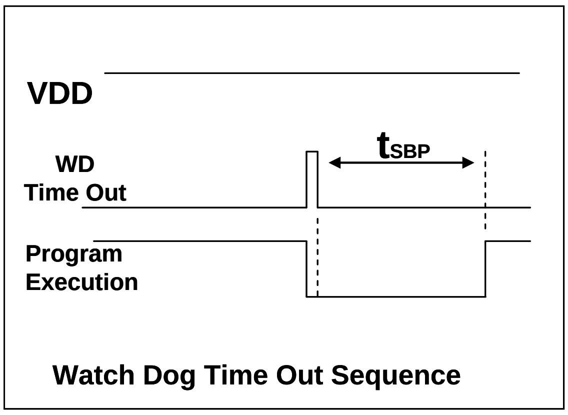


Fig.18: Sequence of Watch Dog Time Out

5.11. Interrupt

There are 7 interrupt lines for PGS152:

- External interrupt PA0
- External interrupt PB0
- Timer16 interrupt
- EEPROM Write Done (EEW_Done) interrupt
- GPC interrupt
- LPWMG interrupt
- Timer2 interrupt

Every interrupt request line has its own corresponding interrupt control bit to enable or disable it; the hardware diagram of interrupt function is shown as Fig.19. All the interrupt request flags are set by hardware and cleared by writing ***intrq*** register. When the request flags are set, it can be rising edge, falling edge or both, depending on the setting of register ***integs***. All the interrupt request lines are also controlled by ***engint*** instruction (enable global interrupt) to enable interrupt operation and ***disgint*** instruction (disable global interrupt) to disable it.

The stack memory for interrupt is shared with data memory and its address is specified by stack register ***sp.*** Since the program counter is 16 bits width, the bit 0 of stack register ***sp*** should be kept 0. Moreover, user can use ***pushaf*** / ***popaf*** instructions to store or restore the values of ***ACC*** and ***flag*** register ***to*** / ***from*** stack memory. Since the stack memory is shared with data memory, the stack position and level are arranged by the compiler in Mini-C project. When defining the stack level in ASM project, users should arrange their locations carefully to prevent address conflicts.

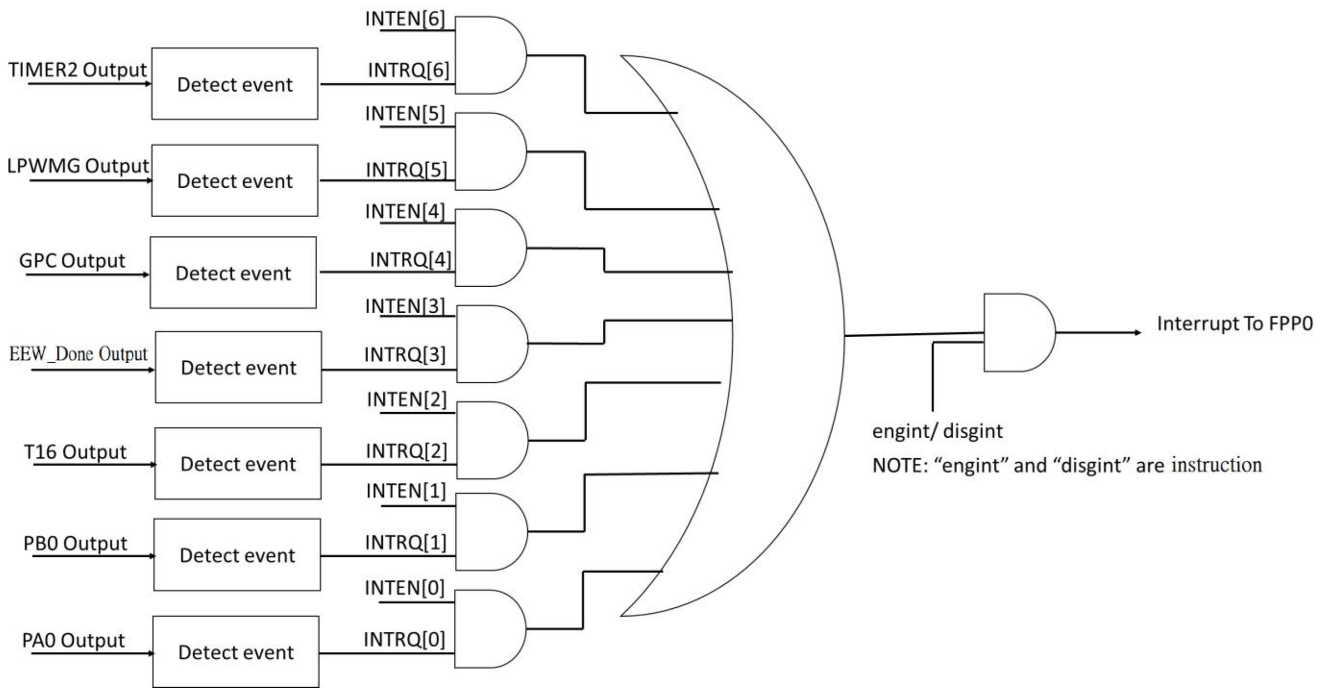


Fig.19: Hardware diagram of interrupt controller

Once the interrupt occurs, its operation will be:

- The program counter will be stored automatically to the stack memory specified by register ***sp.***
- New ***sp*** will be updated to ***sp+2.***
- Global interrupt will be disabled automatically.
- The next instruction will be fetched from address 0x010.

During the interrupt service routine, the interrupt source can be determined by reading the ***intrq*** register.
Note: Even if INTEN=0, INTRQ will be still triggered by the interrupt source.

After finishing the interrupt service routine and issuing the ***reti*** instruction to return back, its operation will be:

- The program counter will be restored automatically from the stack memory specified by register sp.
- New ***sp*** will be updated to ***sp-2***.
- Global interrupt will be enabled automatically.
- The next instruction will be the original one before interrupt.

User must reserve enough stack memory for interrupt, two bytes stack memory for one level interrupt and four bytes for two levels interrupt. And so on, two bytes stack memory is for ***pushaf***. For interrupt operation, the following sample program shows how to handle the interrupt, noticing that it needs four bytes stack memory to handle one level interrupt and ***pushaf***.

```
    void            FPPA0 (void)
    {
        ...
        $   INTEN   PA0;        // INTEN =1; interrupt request when PA0 level changed
        INTRQ   =   0;          // clear INTRQ
        ENGINT                  // global interrupt enable
        ...
        DISGINT                 // global interrupt disable
        ...
    }

  void      Interrupt   (void)          // interrupt service routine
  {
     PUSHAF                             // store ALU and FLAG register

        // If INTEN.PA0 will be opened and closed dynamically,
        // user can judge whether INTEN.PA0 =1 or not.
        // Example:   If (INTEN.PA0 && INTRQ.PA0)   {…}

        // If INTEN.PA0 is always enable,
        // user can omit the INTEN.PA0 judgement to speed up interrupt service routine.

     If (INTRQ.PA0)
     {                                  // Here for PA0 interrupt service routine
```

```
                    INTRQ.PA0 = 0;      // Delete corresponding bit (take PA0 for example)
                ...
        }
          ...
          // X : INTRQ = 0;           // It is not recommended to use INTRQ = 0 to clear all at the end of the
                                      // interrupt service routine.
                                      // It may accidentally clear out the interrupts that have just occurred
                                      // and are not yet processed.
        POPAF                         // restore ALU and FLAG register
        }
```

5.12. Power-Save and Power-Down

There are three operational modes defined by hardware: ON mode, Power-Save mode and Power-Down modes. ON mode is the state of normal operation with all functions ON, Power-Save mode ("***stopexe***") is the state to reduce operating current and CPU keeps ready to continue, Power-Down mode ("***stopsys***") is used to save power deeply. Therefore, Power-Save mode is used in the system which needs low operating power with wake-up occasionally and Power-Down mode is used in the system which needs power down deeply with seldom wake-up. Table 4 shows the differences in oscillator modules between Power-Save mode ("***stopexe***") and Power-Down mode ("***stopsys***").

Differences in oscillator modules between STOPSYS and STOPEXE			
	IHRC	ILRC	EOSC
STOPSYS	Stop	Stop	Stop
STOPEXE	No Change	No Change	No Change

Table 4: Differences in oscillator modules between STOPSYS and STOPEXE

5.12.1. Power-Save mode ("*stopexe*")

Using "***stopexe***" instruction to enter the Power-Save mode, only system clock is disabled, remaining all the oscillator modules active. For CPU, it stops executing; however, for Timer16, counter keep counting if its clock source is not the system clock. The wake-up sources for "stopexe" can be IO-toggle or Timer16 counts to set values when the clock source of Timer16 is IHRC or ILRC modules, or wake-up by comparator when setting GPCC.7=1 and GPCS.6=1 to enable the comparator wake-up function at the same time. Wake-up from input pins can be considered as a continuation of normal execution, the detail information for Power-Save mode shows below:

- IHRC and EOSC oscillator modules: No change, keep active if it was enabled.
- ILRC oscillator modules: must remain enabled, need to start with ILRC when be wakening up.
- System clock: Disable, therefore, CPU stops execution.
- MTP memory is turned off.
- Timer counter: Stop counting if its clock source is system clock or the corresponding oscillator module is disabled; otherwise, it keeps counting. (The Timer contains TM16, TM2, LPWMG0, LPWMG1, LPWMG2.)
- Wake-up sources:
  a. IO toggle wake-up: IO toggling in digital input mode (*PAC* bit is 1 and *PADIER* bit is 1)
  b. Timer wake-up: If the clock source of Timer is not the SYSCLK, the system will be awakened when the Timer counter reaches the set value.

c. Comparator wake-up: It need setting *GPCC*.7=1 and *GPCS*.6=1 to enable the comparator wake-up function at the same time.

An example shows how to use Timer16 to wake-up from "***stopexe***":

```
$ T16M      ILRC, /1, BIT8                 // Timer16 setting
…
WORD        count =     0;
STT16       count;
stopexe;
…
```

The initial counting value of Timer16 is zero and the system will be woken up after the Timer16 counts 256 ILRC clocks.

5.12.2. Power-Down mode ("*stopsys*")

Power-Down mode is the state of deeply power-saving with turning off all the oscillator modules. By using the "*stopsys*" instruction, this chip will be put on Power-Down mode directly. It is recommend to set GPCC.7=0 to disable the comparator before the command "stopsys". The following shows the internal status of PGS152 detail when "***stopsys***" command is issued:

- All the oscillator modules are turned off.
- MTP memory is turned off.
- The contents of SRAM and registers remain unchanged.
- Wake-up sources: IO toggle in digital mode (PADIER bit is 1)

Wake-up from input pins can be considered as a continuation of normal execution. To minimize power consumption, all the I/O pins should be carefully manipulated before entering power-down mode. The reference sample program for power down is shown as below:

```
CLKMD       =     0xF4;        //    Change clock from IHRC to ILRC
CLKMD.4     =     0;           //    disable IHRC
…
while (1)
{
            STOPSYS;           //    enter power-down
            if   (…)   break;  //    if wakeup happen and check OK, then return to high speed,
                               //    else stay in power-down mode again
}
CLKMD       =     0x34;        //    Change clock from ILRC to IHRC/2
```

5.12.3. Wake-up

After entering the Power-Down or Power-Save modes, the PGS152 can be resumed to normal operation by toggling IO pins. Wake-up from timer are available for Power-Save mode ONLY. Table 5 shows the differences in wake-up sources between STOPSYS and STOPEXE.

Differences in wake-up sources between STOPSYS and STOPEXE		
	IO Toggle	Timer wake-up
STOPSYS	Yes	No
STOPEXE	Yes	Yes

Table 5: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PGS152, registers ***padier*** should be properly set to enable the wake-up function for every corresponding pin. The time for normal wake-up is about 3000 ILRC clocks counting from wake-up event; fast wake-up can be selected to reduce the wake-up time by ***misc*** register, and the time for fast wake-up is about 45 ILRC clocks from IO toggling.

Suspend mode	Wake-up mode	Wake-up time (t_{WUP}) from IO toggle
STOPEXE suspend or STOPSYS suspend	Fast wake-up	45 * $T_{ILRC,}$ Where T_{ILRC} is the time period of ILRC
STOPEXE suspend or STOPSYS suspend	Normal wake-up	3000 * T_{ILRC}, Where T_{ILRC} is the clock period of ILRC

Table 6: Differences in wake-up time between Fast/Normal wake-up

Please notice that when Code Option is set to Fast boot-up, no matter which wake-up mode is selected in misc.5, the wake-up mode will be forced to be FAST. If Normal boot-up is selected, the wake-up mode is determined by misc.5.

5.13. IO Pins

All the pins can be independently set into two states output or input by configuring the data registers (***pa, pb***) , control registers (***pac, pbc***) and pull-high/pull-low resistor (***paph/papl, pbph/pbpl***). All these pins have Schmitt-trigger input buffer and output driver with CMOS level. When it is set to output low, the pull- high resistor is turned off automatically. If user wants to read the pin state, please notice that it should be set to input mode before reading the data port; if user reads the data port when it is set to output mode, the reading data comes from data register, NOT from IO pad. As an example, Table 7 shows the configuration table of bit 0 of port A. The hardware diagram of IO buffer is also shown as Fig.20.

pa.0	***pac.0***	***papl.0***	***paph.0***	**Description**
X	0	0	0	Input without pull- high/pull-low resistor
X	0	0	1	Input with pull- high resistor, without pull-low resistor
X	0	1	0	Input with pull-low resistor, without pull- high resistor
0	1	0	X	Output low without pull-low resistor
0	1	1	X	Output low with pull-low resistor
1	1	X	0	Output high without pull- high resistor
1	1	X	1	Output high with pull- high resistor

Table 7: PA0 Configuration Table

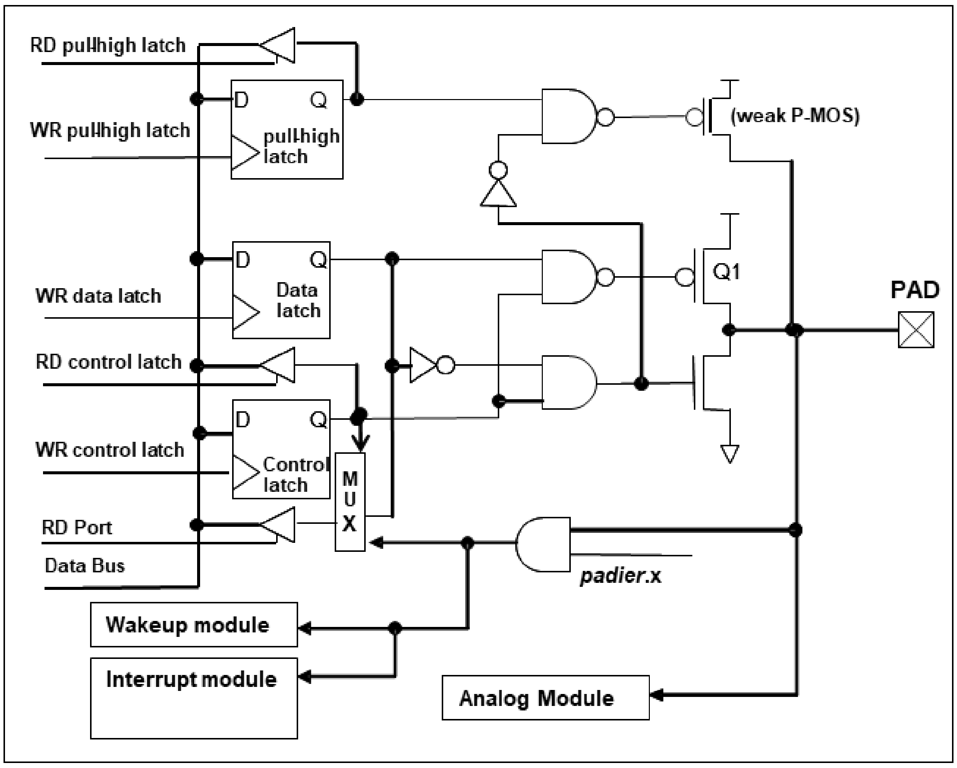


Fig.20: Hardware diagram of IO buffer

All the IO pins have the same structure. The corresponding bits in registers ***padier or pbdier*** should be set to low to prevent leakage current for those pins are selected to be analog function. When PGS152 is put in power-down or power-save mode, every pin can be used to wake-up system by toggling its state. Therefore, those pins needed to wake-up system must be set to input mode and set the corresponding bits of registers ***padier*** to high***.*** The same reason, ***padier***.0 should be set high when PA0 is used as external interrupt pin.

5.14. Reset, LVR and LVD

5.14.1. Reset

There are many causes to reset the PGS152, once reset is asserted, most of all the registers in PGS152 will be set to default values, system should be restarted once abnormal cases happen, or by jumping program counter to address 0x00.

After power-up and LVR reset, the SRAM data will be kept when VDD>V_{DR} (SRAM data retention voltage). However, if SRAM is cleared after power-on again, the data cannot be kept. And, the data memory is in an uncertain state when VDD<V_{DR}.

The content will be kept when reset comes from PRSTB pin or WDT timeout.

5.14.2. LVR reset

By code option, there are 8 different levels of LVR from 1.8V to 4.0V for reset. Usually, user selects LVR reset level to be in conjunction with operating frequency and supply voltage.

5.14.3. LVD

By code option, user can use lvdc[7:2] select different levels of LVD from 1.85V to 5V, LVD can provide more accurate voltage for user confirm the voltage level.

6. IO Registers

6.1. ACC Status Flag Register (*flag*), IO address = 0x00

Bit	Reset	R/W	Description
7 - 4	-	-	Reserved. Please do not use.
3	0	R/W	OV (Overflow Flag). This bit is set to be 1 whenever the sign operation is overflow.
2	0	R/W	AC (Auxiliary Carry Flag). There are two conditions to set this bit, the first one is carry out of low nibble in addition operation and the other one is borrow from the high nibble into low nibble in subtraction operation.
1	0	R/W	C (Carry Flag). There are two conditions to set this bit, the first one is carry out in addition operation, and the other one is borrow in subtraction operation. Carry is also affected by shift with carry instruction.
0	0	R/W	Z (Zero Flag). This bit will be set when the result of arithmetic or logic operation is zero; Otherwise, it is cleared.

6.2. Stack Pointer Register (*sp*), IO address = 0x02

Bit	Reset	R/W	Description
7 - 0	-	R/W	Stack Pointer Register. Read out the current stack pointer, or write to change the stack pointer.

6.3. Clock Mode Register (*clkmd*), IO address = 0x03

Bit	Reset	R/W	Description	
7 - 5	111	R/W	System clock (CLK) selection:	
			Type 0, clkmd[3]=0	Type 1, clkmd[3]=1
			000: IHRC÷4 001: IHRC÷2 010: reserved 011: EOSC÷4 100: EOSC÷2 101: EOSC 110: ILRC÷4 111: ILRC (default)	000: IHRC÷16 001: IHRC÷8 010: ILRC÷16 (ICE does NOT Support.) 011: IHRC÷32 100: IHRC÷64 101: EOSC÷8 11x: reserved
4	1	R/W	Internal High RC Enable. 0 / 1: disable / enable	
3	0	R/W	Clock Type Select. This bit is used to select the clock type in bit [7:5]. 0 / 1: Type 0 / Type 1	
2	1	R/W	Internal Low RC Enable. 0 / 1: disable / enable If ILRC is disabled, watchdog timer is also disabled.	
1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable	
0	0	R/W	Pin PA5/PRSTB function. 0 / 1: PA5 / PRSTB	

6.4. Interrupt Enable Register (*inten*), IO address = 0x04

Bit	Reset	R/W	Description
7	0	R/W	Reserved
6	0	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable
5	0	R/W	Enable interrupt from LPWMG. 0 / 1: disable / enable
4	0	R/W	Enable interrupt from comparator. 0 / 1: disable / enable
3	0	R/W	Enable interrupt from EEPROM EEW_Done. 0 / 1: disable / enable.
2	0	R/W	Enable interrupt from Timer16 overflow. 0 / 1: disable / enable
1	0	R/W	Enable interrupt from PB0. 0 / 1: disable / enable.
0	0	R/W	Enable interrupt from PA0. 0 / 1: disable / enable

6.5. Interrupt Request Register (*intrq*), IO address = 0x05

Bit	Reset	R/W	Description
7	-	R/W	Reserved
6	-	R/W	Interrupt Request from Timer2, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
5	-	R/W	Interrupt Request from LPWM, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
4	-	R/W	Interrupt Request from comparator, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
3	-	R/W	Interrupt Request from EEPROM EEW_Done, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
2	-	R/W	Interrupt Request from Timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	R/W	Interrupt Request from pin PB0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0, this bit is set by hardware and cleared by software. 0 / 1: No Request / request

6.6. Timer16 mode Register (*t16m*), IO address = 0x06

Bit	Reset	R/W	Description
7 - 5	000	R/W	Timer16 Clock source selection. 000: disable 001: CLK (system clock) 010: reserved 011: PA4 falling edge (from external pin) 100: IHRC 101: EOSC 110: ILRC 111: PA0 falling edge (from external pin)
4 - 3	00	R/W	Timer16 clock pre-divider. 00: ÷1 01: ÷4 10: ÷16 11: ÷64
2 - 0	000	R/W	Interrupt source selection. Interrupt event happens when the selected bit status is changed. 0 : bit 8 of Timer16 1 : bit 9 of Timer16 2 : bit 10 of Timer16 3 : bit 11 of Timer16 4 : bit 12 of Timer16 5: bit 13 of Timer16 6: bit 14 of Timer16 7: bit 15 of Timer16

6.7. MISC Register (*misc*), IO address = 0x08

Bit	Reset	R/W	Description
7 - 6	-	-	Reserved.
5	0	WO	Enable fast Wake up. Fast wake-up is NOT supported when EOSC is enabled. 0: Normal wake up. The wake-up time is 3000 ILRC clocks (Not for fast boot-up) 1: Fast wake up. The wake-up time is 45 ILRC clocks.
4 - 3	-	-	Reserved.
2	0	WO	Disable LVR function. 0 / 1 : Enable / Disable
1 - 0	00	WO	Watch dog time out period 00: 8k ILRC clock period 01: 16k ILRC clock period 10: 64k ILRC clock period 11: 256k ILRC clock period

6.8. External Oscillator setting Register (*eoscr*), IO address = 0x0a

Bit	Reset	R/W	Description
7	0	WO	Enable external crystal oscillator. 0 / 1 : Disable / Enable
6 - 5	-	-	Reserved
4 - 3	00	WO	Set Xin Build-in Capacitor for 32KHz Crystal. 00 / 01 / 10 / 11：Disable / 7pF / 9.5pF / 12.5pF
2 - 1	00	WO	Set Xout Build-in Capacitor for 32KHz Crystal 00 / 01 / 10 / 11：Disable / 7pF / 9.5pF / 12.5pF.
0	0	WO	Power-down the Bandgap and LVR/LVD hardware modules. 0 / 1: normal / power-down. Note: If bandgap be disabled, there will only ILRC/T16/TM2 and I/O function can be used.

6.9. Interrupt Edge Select Register (*integs*), IO address = 0x0c

Bit	Reset	R/W	Description
7 - 5	-	-	Reserved.
4	0	WO	Timer16 edge selection. 0 : rising edge of the selected bit to trigger interrupt 1 : falling edge of the selected bit to trigger interrupt
3 - 2	00	WO	PB0 edge selection. 00 : both rising edge and falling edge to trigger interrupt 01 : rising edge to trigger interrupt 10 : falling edge to trigger interrupt 11 : reserved.
1 - 0	00	WO	PA0 edge selection. 00: both rising edge and falling edge of the selected bit to trigger interrupt 01: rising edge of the selected bit to trigger interrupt 10: falling edge of the selected bit to trigger interrupt 11: reserved.

6.10. Port A Digital Input Enable Register (*padier*), IO address = 0x0d

Bit	Reset	R/W	Description
7	0	WO	Enable PA7 digital input and wake-up event. 1 / 0: enable / disable. This bit should be set to low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA7 is analog input and can NOT be used to wake-up the system.
6	0	WO	Enable PA6 digital input and wake-up event. 1 / 0: enable / disable. This bit should be set to low to prevent leakage current when external crystal oscillator is used. If this bit is set to low, PA6 is analog input and can NOT be used to wake-up the system.
5	0	WO	Enable PA5 digital input and wake-up event. 1 / 0: enable / disable. If this bit is set to low, PA5 is analog input and can NOT be used to wake-up the system.
4 - 3	00	WO	Enable PA4-PA3 digital input and wake-up event. 1 / 0: enable / disable. This bit should be set to low when PA4 is assigned as comparator input to prevent leakage current. If these bit are set to low, PA4-PA3 are analog input and can NOT be used to wake-up the system.
2 - 1	00	WO	Reserved
0	0	WO	Enable PA0 digital input, wake-up event and interrupt request. 1 / 0: enable / disable. This bit can be set to low to disable wake-up from PA0 toggling and interrupt request from this pin. If this bit is set to low, PA0 is analog input and can NOT be used to wake-up the system, interrupt from this pin is also disabled.

6.11.Port B Digital Input Enable Register (*pbdier*), IO address = 0x0e

Bit	Reset	R/W	Description
7	0	WO	Enable PB7 digital input, wake up event and interrupt request. 1 / 0 : enable / disable. When this bit is “0”, the PB7 wake up function and interrupt request from this pin are disabled.
6 - 1	0	WO	Reserved
0	0	WO	Enable PB0 digital input, wake up event and interrupt request. 1 / 0 : enable / disable. When this bit is “0”, the PB0 wake up function and interrupt request from this pin are disabled.

6.12.Port A Data Register (*pa*), IO address = 0x10

Bit	Reset	R/W	Description
7 - 3	0x00	R/W	Data register bit 7 - 3 for Port A.
2 - 1	-	-	Reserved
0	0	R/W	Data register bit 0 for Port A.

6.13. Port A Control Register (*pac*), IO address = 0x11

Bit	Reset	R/W	Description
7 - 3	0x00	R/W	Port A control registers for bit 7 - 3. This register is used to define input mode or output mode for each corresponding pin of port A. 0 / 1: input / output
2 - 1	-	-	Reserved
0	0	R/W	Port A control registers for bit 0. This register is used to define input mode or output mode for each corresponding pin of port A. 0 / 1: input / output

6.14. Port A Pull-High Register (*paph*), IO address = 0x12

Bit	Reset	R/W	Description
7 - 3	0x00	R/W	Port A pull-high register for bit 7 - 3. This register is used to enable the internal pull-high device on each corresponding pin of port A and this pull high function is active only for input mode. 0 / 1 : disable / enable
2 - 1	-	-	Reserved
0	0	R/W	Port A pull-high register for bit 0. This register is used to enable the internal pull-high device on each corresponding pin of port A and this pull high function is active only for input mode. 0 / 1 : disable / enable

6.15. Port A Pull-Low Register (*papl*), IO address = 0x13

Bit	Reset	R/W	Description
7 - 3	0x00	R/W	Port A pull-low register for bit 7 - 3. This register is used to enable the internal pull-low device on each corresponding pin of port A and this pull low function is active only for input mode. 0 / 1 : disable / enable
2 - 1	-	-	Reserved
0	0	R/W	Port A pull-low register for bit 0. This register is used to enable the internal pull-low device on each corresponding pin of port A and this pull low function is active only for input mode. 0 / 1 : disable / enable

6.16. Port B Data Register (*pb*), IO address = 0x14

Bit	Reset		R/W	Description
7	0		R/W	Data register bit 7 for Port B.
6 - 1	-		-	
0	0		R/W	Data register bit 0 for Port B.

6.17. Port B Control Register (*pbc*), IO address = 0x15

Bit	Reset	R/W	Description
7	0	R/W	Port B control registers for bit 7. This register is used to define input mode or output mode for each corresponding pin of port B. 0 / 1: input / output.
6 - 1	-	-	
0	0	R/W	Port B control registers for bit 0. This register is used to define input mode or output mode for each corresponding pin of port B. 0 / 1: input / output.

6.18. Port B Pull-High Registers (*pbph*), IO address = 0x16

Bit	Reset	R/W	Description
7	0	R/W	Port B pull-high registers for bit 7. This register is used to enable the internal pull-high device on each corresponding pin of port B. 0 / 1 : disable / enable
6 - 1	-	-	
0	0	R/W	Port B pull-high registers for bit 0. This register is used to enable the internal pull-high device on each corresponding pin of port B. 0 / 1 : disable / enable

6.19. Port B Pull-Low Registers (*pbpl*), IO address = 0x17

Bit	Reset	R/W	Description
7	0	R/W	Port B pull-low registers for bit 7. This register is used to enable the internal pull-low device on each corresponding pin of port B. 0 / 1 : disable / enable
6 - 1	-	-	
0	0	R/W	Port B pull-low registers for bit 0. This register is used to enable the internal pull-low device on each corresponding pin of port B. 0 / 1 : disable / enable

6.20. Comparator Control Register (*gpcc*), IO address = 0x18

Bit	Reset	R/W	Description
7	0	R/W	Enable comparator. 0 / 1 : disable / enable When this bit is set to enable, please also set the corresponding analog input pins to be digital disable to prevent IO leakage.
6	-	RO	Comparator result of comparator. 0: plus input < minus input 1: plus input > minus input
5	0	R/W	Select whether the comparator result output will be sampled by TM2_CLK? 0: result output NOT sampled by TM2_CLK 1: result output sampled by TM2_CLK
4	0	R/W	Inverse the polarity of result output of comparator. 0: polarity is NOT inversed. 1: polarity is inversed.
3 - 1	000	R/W	Selection the minus input (-) of comparator. 000 : PA3 001 : PA4 010 : Internal 1.20 volt bandgap reference voltage 011 : $V_{\text{internal R}}$ 100 : reserved 101 : PB7 11X: reserved
0	0	R/W	Selection the plus input (+) of comparator. 0 : $V_{\text{internal R}}$ 1 : PA4

6.21. Comparator Selection Register (*gpcs*), IO address = 0x19

Bit	Reset	R/W	Description
7	0	WO	Comparator output enable (to PA0). 0 / 1 : disable / enable (Please avoid this situation: GPCS will affect the PA3 output function when selecting output to PA0 output in ICE.)
6	0	WO	Wakeup by comparator enable. (The comparator wakeup effectively when gpcc.6 electrical level changed) 0 / 1 : disable / enable
5	0	WO	Selection of high range of comparator.
4	0	WO	Selection of low range of comparator.
3 - 0	0000	WO	Selection the voltage level of comparator. 0000 (lowest) ~ 1111 (highest)

6.22. Timer2 Control Register (*tm2c*), IO address = 0x1c

Bit	Reset	R/W	Description
7 - 4	0000	R/W	Timer2 clock selection. 0000 : disable 0001 : CLK (system clock) 0010 : IHRC or IHRC *2 (by code option TM2_source) (ICE doesn't support IHRC *2.) 0011 : EOSC 0100 : ILRC 0101 : comparator output 011x : reserved 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) **Notice:** In ICE mode and IHRC is selected for Timer2 clock, the clock sent to Timer2 does NOT be stopped, Timer2 will keep counting when ICE is in halt state.
3 - 2	00	R/W	Timer2 output selection. 00 : disable 01 : Reserved 10 : PA3 11 : PB0
1	0	R/W	TM2 Mode 0: Period Mode 1: PWM Mode
0	0	R/W	Inverse the polarity of result output of TM2. 0: polarity is NOT inversed. 1: polarity is inversed.

6.23. Timer2 Scalar Register (*tm2s*), IO address = 0x1e

Bit	Reset	R/W	Description
7	0	WO	PWM resolution selection. 0 : 8-bit 1 : 6-bit or 7-bit (by code option TM2_bit) (ICE doesn't support 7-bit.)
6 - 5	00	WO	Timer2 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4 - 0	00000	WO	Timer2 clock scalar.

6.24. Timer2 Counter Register (*tm2ct*), IO address = 0x1d

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Bit [7:0] of Timer2 counter register.

6.25. Timer2 Bound Register (*tm2b*), IO address = 0x09

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Timer2 bound register.

6.26. Low Voltage Detect Control Register (*lvdc*), IO address = 0x1f

Bit	Reset	R/W	Description
7 - 2	000000	WO	Set LVD level: in the range of 1.85~5V, increment by 0.05V
1	-	-	Reserved
0	0	RO	The detect result between LVD & VDD 0: VDD > LVD level 1: VDD < LVD level

6.27. LPWMG0 control Register (*lpwmg0c*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	GPC control lpwmg0 output. 0 / 1: Disable / Enable
6	-	RO	Output status of LPWMG0 generator.
5	0	R/W	Lpwmg0 output. 0 / 1: Buffered / Inverted
4	0	R/W	LPWMG0 output selection. 0: LPWMG0 Output 1: LPWMG0 XOR LPWMG1 or LPWMG0 OR LPWMG1 (by lpwmg0c.0)
3 - 1	000	R/W	LPWMG0 Output Port Selection 000: LPWMG0 Output Disable 001: Reserved 010: Reserved 011: LPWMG0 Output to PA0 1xx: Reserved
0	0	R/W	LPWMG0 output pre- selection. 0: LPWMG0 XOR LPWMG1 1: LPWMG0 OR LPWMG1

6.28. LPWMG Clock Register (*lpwmgclk*), IO address = 0x21

Bit	Reset	R/W	Description
7	0	WO	LPWMG Disable/ Enable 0: LPWMG Disable 1: LPWMG Enable
6 - 4	000	WO	LPWMG clock pre-scalar. 000: ÷1 001: ÷2 010: ÷4 011: ÷8 100: ÷16 101: ÷32 110: ÷64 111: ÷128
3 - 1	-	-	Reserved
0	0	WO	LPWMG clock source selection 0: System Clock 1: IHRC or IHRC*2 (by code option PWM_Source)

6.29. LPWMG0 Duty Value High Register (*lpwmg0dth*), IO address = 0x22

Bit	Reset	R/W	Description
7 - 0	-	WO	Bit[10:3] of LPWMG0 Duty.

6.30. LPWMG0 Duty Value Low Register (*lpwmg0dtl*), IO address = 0x23

Bit	Reset	R/W	Description
7 - 5	-	WO	Bit[2:0] of LPWMG0 Duty.
4 - 0	-	-	Reserved

Note: It's necessary to write LPWMG0 Duty_Value Low Register before writing LPWMG0 Duty_Value High Register.

6.31. LPWMG Counter Upper Bound High Register (*lpwmgcubh*), IO address = 0x24

Bit	Reset	R/W	Description
7 - 0	-	WO	Bit[10:3] of LPWMG Counter Bound.

6.32. LPWMG Counter Upper Bound Low Register (*lpwmgcubl*), IO address = 0x25

Bit	Reset	R/W	Description
7 - 6	-	WO	Bit[2:1] of LPWMG Counter Bound.
5 - 0	-	-	Reserved

6.33. LPWMG1 control Register (*lpwmg1c*), IO address = 0x26

Bit	Reset	R/W	Description
7	0	R/W	GPC control lpwmg1 output. 0 / 1: Disable / Enable
6	-	RO	Output status of LPWMG1 generator
5	0	R/W	Lpwmg1 output. 0 / 1: Buffered / Inverted
4	0	R/W	LPWMG1 output selection: 0: LPWMG1 1: LPWMG2
3 - 1	000	R/W	LPWMG1 Output Port Selection: 000: LPWMG1 Output Disable 001: Reserved 010: Reserved 011: LPWMG0 Output to PA4 100: LPWMG0 Output to PB7 1xx: Reserved
0	-	R/W	Reserved

6.34. LPWMG1 Duty Value High Register (*lpwmg1dth*), IO address = 0x28

Bit	Reset	R/W	Description
7 - 0	-	WO	Bit[10:3] of LPWMG1 Duty

6.35. LPWMG1 Duty Value Low Register (*lpwmg1dtl*), IO address = 0x29

Bit	Reset	R/W	Description
7 - 5	-	WO	Bit[2:0] of LPWMG1 Duty.
4 - 0	-	-	Reserved

Note: It's necessary to write LPWMG1 Duty_Value Low Register before writing LPWMG1 Duty_Value High Register.

6.36. LPWMG2 control Register (*lpwmg2c*), IO address = 0x2C

Bit	Reset	R/W	Description
7	0	R/W	GPC control lpwmg2 output. 0 / 1: Disable / Enable
6	-	RO	Output status of LPWMG2 generator.
5	0	R/W	LPWMG2 output. 0 / 1: Buffered / Inverted
4	0	R/W	LPWMG2 output selection: 0: LPWMG2 1: LPWMG2 ÷2
3 - 1	000	R/W	LPWMG2 Output Port Selection: 000: LPWMG2 Output Disable 001: Reserved 010: Reserved 011: LPWMG2 Output to PA3 100: Reserved 101: LPWMG2 Output to PA5 1xx: Reserved
0	-	R/W	Reserved

6.37. LPWMG2 Duty Value High Register (*lpwmg2dth*), IO address = 0x2E

Bit	Reset	R/W	Description
7 - 0	-	WO	Bit[10:3] of LPWMG2 Duty

6.38. LPWMG2 Duty Value Low Register (*lpwmg2dtl*), IO address = 0x2F

Bit	Reset	R/W	Description
7 - 5	-	WO	Bit[2:0] of LPWMG2 Duty
4 - 0	-	-	Reserved

Note: It's necessary to write LPWMG2 Duty_Value Low Register before writing LPWMG2 Duty_Value High Register.

6.39. EEPROM Data Register (*eerl*), IO address = 0x30

Bit	Reset	R/W	Description
7 - 0	-	R/W	EEPROM low byte Data register.

6.40. EEPROM Control Registers (*eermc*), IO address = 0x31

Bit	Reset	R/W	Description
7	-	-	Reserved.
6	0	RO	EEPROM Read or Write = Done / Busy (0 / 1)
5	0	RO	EEPROM Write result = Write or Erase complete/ Write or Erase Time-out (0 / 1)
4 – 0	-	-	Reserved.
7 - 0	0x00	WO	5A: Write 0x5A before each STEER to Enable EEPROM Byte Program A5: Write 0xA5 before each STEER to Enable EEPROM Page Erase, 8 Bytes on one page

6.41. Option Register3 (*opr3*), IO address = 0x3B

Bit	Reset	R/W	Description
7	-	-	Reserved
6	0	WO	0: PA5 ~ 7, PB0, PB7 IOH / IOL current = 5 / 10 mA 1: PA5 ~ 7, PB0, PB7 IOH / IOL current = 40 / 50 mA
5	-	-	Reserved
4	0	WO	0: PA4 IOH / IOL current = 5 / 10 mA 1: PA4 IOH / IOL current = 40 / 50 mA
3	-	-	Reserved
2	0	WO	0: PA3 IOH / IOL current = 5 / 10 mA 1: PA3 IOH / IOL current = 40 / 50 mA
1	-	-	Reserved
0	0	WO	0: PA0 IOH / IOL current = 5 / 10 mA 1: PA0 IOH / IOL current = 40 / 50 mA

Note: users can select the driving and sinking current capabilities of PA4/PA3/PA0 as they needed. The option for those there pins are independent and do not interfere with each others.

6.42. EEPROM IHRC Register (*ihrc_epm*), IO address = 0x3C

Bit	Reset	R/W	Description
7	0	WO	Reserved，write 0
6	0	-	Reserved
5 - 0	0x3F	WO	EEPROM_IHRC 0x3F: Erase/write EEPROM in default sequence mode 0x34: Erase/write EEPROM in enhanced sequence mode (switch to enhanced sequence mode when EERMC .Time_Out)

7. Instructions

Symbol	Description
ACC	Accumulator (Abbreviation of accumulator)
a	Accumulator (symbol of accumulator in program)
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
—	Subtraction
～	NOT (logical complement, 1's complement)
〒	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
M.n	Only addressed in 0~0x3F (0~63) is allowed
IO.n	Only addressed in 0~0x3F (0~63) is allowed

7.1. Data Transfer Instructions

mov a, I	Move immediate data into ACC.
	Example: *mov* a, 0x0f;
	Result: a ← 0fh;
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

mov M, a	Move data from ACC into memory Example: *mov* MEM, a; Result: MEM ← a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
mov a, M	Move data from memory into ACC Example: *mov* a, MEM ; Result: a ← MEM; Flag Z is set when MEM is zero. Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
mov a, IO	Move data from IO into ACC Example: *mov* a, pa ; Result: a ← pa; Flag Z is set when pa is zero. Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
mov IO, a	Move data from ACC into IO Example: *mov* pa, a; Result: pa ← a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

steer index	Store the byte (word) data in IO register(s) eerl (and eerh) to EEPROM by using index as EEPROM address. Start the EEPROM programming progress.
	Example: *steer* index;
	Result: EEPROM[index] ← {(eerh,)eerl};
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	word point = 0;
	eerl = 0x12;
	eerh = 0x34;
	steer point;

ldeer index	Load the byte (word) data in EEPROM to IO register(s) eerl (and eerh) by using index as EEPROM address. Start the EEPROM reading progress.
	Example: ldeer index;
	Result: {(eerh,)eerl} ← EEPROM[index];
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	word point = 0;
	ldeer point;

ldt16 word	Move 16-bit counting values in Timer16 to memory in word. Example: *ldt16* word; Result: word ← 16-bit timer Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --

	```
	word      T16val ;          // declare a RAM word
	…
	*clear*     lb@ T16val ;      // clear T16val (LSB)
	*clear*     hb@ T16val ;     // clear T16val (MSB)
	*stt16*     T16val ;         // initial T16 with 0
	…
	*set1*      t16m.5 ;         // enable Timer16
	…
	*set0*      t16m.5 ;         // disable Timer 16
	*ldt16*     T16val ;         // save the T16 counting value to T16val
	….
	```
	-- 	

stt16 word	Store 16-bit data from memory in word to Timer16.
	Example: *stt16* word;
	Result: 16-bit timer ←word
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	Application Example:
	-- 	
	```
	word      T16val ;          // declare a RAM word
	…
	*mov*       a, 0x34 ;
	*mov*       lb@ T16val , a ;   // move 0x34 to T16val (LSB)
	*mov*       a, 0x12 ;
	*mov*       hb@ T16val , a ;   // move 0x12 to T16val (MSB)
	*stt16*     T16val ;            // initial T16 with 0x1234
	…
	```
	-- 	

idxm a, index	Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction.
	Example: *idxm* a, index;
	Result: a ← [index], where index is declared by word.
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	-- 	
	```
	word      RAMIndex ;           // declare a RAM pointer
	…
	*mov*       a, 0x5B ;            // assign pointer to an address (LSB)
	*mov*       lb@RAMIndex, a ;     // save pointer to RAM (LSB)
	*mov*       a, 0x00 ;            // assign 0x00 to an address (MSB), should be 0
	*mov*       hb@RAMIndex, a ;     // save pointer to RAM (MSB)
	…
	*idxm*      a, RAMIndex ;        // mov memory data in address 0x5B to ACC
	```
	-- 	

Idxm index, a	Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction.
	Example: *idxm* index, a;
	Result: [index] ← a; where index is declared by word.
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	-- 	
	word RAMIndex ; // declare a RAM pointer
	…
	mov a, 0x5B ; // assign pointer to an address (LSB)
	mov lb@RAMIndex, a ; // save pointer to RAM (LSB)
	mov a, 0x00 ; // assign 0x00 to an address (MSB), should be 0
	mov hb@RAMIndex, a ; // save pointer to RAM (MSB)
	…
	mov a, 0xA5 ;
	idxm RAMIndex, a ; // mov 0xA5 to memory in address 0x5B
	-- 	

xch M	Exchange data between ACC and memory Example: *xch* MEM ; Result: MEM ← a , a ← MEM Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

pushaf	Move the *ACC* and *flag* register to memory that address specified in the stack pointer.
	Example: *pushaf*;
	Result: [sp] ← {flag, ACC};
	sp ← sp + 2 ;
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	-- 	
	.romadr 0x10 ; // ISR entry address
	pushaf ; // put ACC and flag into stack memory
	… // ISR program
	… // ISR program
	popaf ; // restore ACC and flag from stack memory
	reti ;
	-- 	

popaf	Restore *ACC* and *flag* from the memory which address is specified in the stack pointer. Example: *popaf*; Result: sp ← sp - 2 ; {Flag, ACC} ← [sp] ; Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

7.2. Arithmetic Operation Instructions

add a, I	Add immediate data with ACC, then put result into ACC Example: *add* a, 0x0f ; Result: a ← a + 0fh Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
add a, M	Add data in memory with ACC, then put result into ACC Example: *add* a, MEM ; Result: a ← a + MEM Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
add M, a	Add data in memory with ACC, then put result into memory Example: *add* MEM, a; Result: MEM ← a + MEM Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc a, M	Add data in memory with ACC and carry bit, then put result into ACC Example: *addc* a, MEM ; Result: a ← a + MEM + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc M, a	Add data in memory with ACC and carry bit, then put result into memory Example: *addc* MEM, a ; Result: MEM ← a + MEM + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc a	Add carry with ACC, then put result into ACC Example: *addc* a ; Result: a ← a + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc M	Add carry with memory, then put result into memory Example: *addc* MEM ; Result: MEM ← MEM + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
nadd a, M	Add negative logic (2's complement) of ACC with memory Example: *nadd a, MEM ;* Result: a ← 〒a + MEM Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
nadd M, a	Add negative logic (2's complement) of memory with ACC Example: *nadd MEM, a ;* Result: MEM ← 〒MEM + a Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
sub a, I	Subtraction immediate data from ACC, then put result into ACC. Example: *sub* a, 0x0f; Result: a ← a - 0fh (a + [2's complement of 0fh]) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
sub a, M	Subtraction data in memory from ACC, then put result into ACC Example: *sub* a, MEM ; Result: a ← a - MEM (a + [2's complement of M]) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV

sub M, a	Subtraction data in ACC from memory, then put result into memory Example: *sub* MEM, a; Result: MEM ← MEM - a (MEM + [2's complement of a]) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
subc a, M	Subtraction data in memory and carry from ACC, then put result into ACC Example: *subc* a, MEM; Result: a ← a – MEM - C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
subc M, a	Subtraction ACC and carry bit from memory, then put result into memory Example: *subc* MEM, a ; Result: MEM ← MEM – a - C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
subc a	Subtraction carry from ACC, then put result into ACC Example: *subc* a; Result: a ← a - C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
subc M	Subtraction carry from the content of memory, then put result into memory Example: *subc* MEM; Result: MEM ← MEM - C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
inc M	Increment the content of memory Example: *inc* MEM ; Result: MEM ← MEM + 1 Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
dec M	Decrement the content of memory Example: *dec* MEM; Result: MEM ← MEM - 1 Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
clear M	Clear the content of memory Example: *clear* MEM ; Result: MEM ← 0 Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

7.3. Shift Operation Instructions

sr a	Shift right of ACC, shift 0 to bit 7 Example: *sr* a ; Result: a (0,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
src a	Shift right of ACC with carry bit 7 to flag Example: *src* a ; Result: a (c,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
sr M	Shift right the content of memory, shift 0 to bit 7 Example: *sr* MEM ; Result: MEM(0,b7,b6,b5,b4,b3,b2,b1) ← MEM(b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
src M	Shift right of memory with carry bit 7 to flag Example: *src* MEM ; Result: MEM(c,b7,b6,b5,b4,b3,b2,b1) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
sl a	Shift left of ACC shift 0 to bit 0 Example: *sl* a ; Result: a (b6,b5,b4,b3,b2,b1,b0,0) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a (b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
slc a	Shift left of ACC with carry bit 0 to flag Example: *slc* a ; Result: a (b6,b5,b4,b3,b2,b1,b0,c) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
sl M	Shift left of memory, shift 0 to bit 0 Example: *sl* MEM ; Result: MEM (b6,b5,b4,b3,b2,b1,b0,0) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
slc M	Shift left of memory with carry bit 0 to flag Example: *slc* MEM ; Result: MEM (b6,b5,b4,b3,b2,b1,b0,C) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM (b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
swap a	Swap the high nibble and low nibble of ACC Example: *swap* a ; Result: a (b3,b2,b1,b0,b7,b6,b5,b4) ← a (b7,b6,b5,b4,b3,b2,b1,b0) Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7.4. Logic Operation Instructions

and a, I	Perform logic AND on ACC and immediate data, then put result into ACC Example: *and* a, 0x0f ; Result: a ← a & 0fh Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
and a, M	Perform logic AND on ACC and memory, then put result into ACC Example: *and* a, RAM10 ; Result: a ← a & RAM10 Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
and M, a	Perform logic AND on ACC and memory, then put result into memory Example: *and* MEM, a ; Result: MEM ← a & MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
or a, I	Perform logic OR on ACC and immediate data, then put result into ACC Example: *or* a, 0x0f ; Result: a ← a | 0fh Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
or a, M	Perform logic OR on ACC and memory, then put result into ACC Example: *or* a, MEM ; Result: a ← a | MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
or M, a	Perform logic OR on ACC and memory, then put result into memory Example: *or* MEM, a ; Result: MEM ← a | MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
xor a, I	Perform logic XOR on ACC and immediate data, then put result into ACC Example: *xor* a, 0x0f ; Result: a ← a ^ 0fh Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV

xor IO, a	Perform logic XOR on ACC and IO register, then put result into IO register
	Example: *xor* *pa, a* **;**
	Result: pa ← a ^ pa ; // pa is the data register of port A
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

xor a, M	Perform logic XOR on ACC and memory, then put result into ACC Example: *xor* a, MEM ; Result: a ← a ^ RAM10 Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
xor M, a	Perform logic XOR on ACC and memory, then put result into memory Example: *xor* MEM, a ; Result: MEM ← a ^ MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV

not a	Perform 1's complement (logical complement) of ACC
	Example: *not* a ;
	Result: a ← ～a
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	Application Example:
	-- 	
	mov a, 0x38 ; // ACC=0X38
	not a ; // ACC=0XC7
	-- 	

not M	Perform 1's complement (logical complement) of memory
	Example: *not* MEM ;
	Result: MEM ← ～MEM
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	-- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	not mem ; // mem = 0xC7
	-- 	

neg a	Perform 2's complement of ACC
	Example: *neg* a;
	Result: a ← ∓a
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	-- 	
	mov a, 0x38 ; // ACC=0X38
	neg a ; // ACC=0XC8
	-- 	

neg M	Perform 2's complement of memory
	Example: *neg* MEM;
	Result: MEM ← ∓MEM
	Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
	
	Application Example:
	-- 	
	mov a, 0x38 ;
	mov mem, a ; // mem = 0x38
	not mem ; // mem = 0xC8
	-- 	

comp a, M	Compare ACC with the content of memory
	Example: *comp a, MEM;*
	Result: Flag will be changed by regarding as (a - MEM)
	Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
	Application Example:
	-- 	
	mov a, 0x38 ;
	mov mem, a ;
	comp a, mem ; // Z flag is set as 1
	mov a, 0x42 ;
	mov mem, a ;
	mov a, 0x38 ;
	comp a, mem ; // C flag is set as 1
	-- 	

comp M, a	Compare ACC with the content of memory Example: comp MEM, a; Result: Flag will be changed by regarding as (MEM - a) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

7.5. Bit Operation Instructions

set0 IO.n	Set bit n of IO port to low Example: *set0* pa.5 ; Result: set bit 5 of port A to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 IO.n	Set bit n of IO port to high Example: *set1* pa.5 ; Result: set bit 5 of port B to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

swapc IO.n	Swap the bit n of IO port with carry bit
	Example: swapc IO.0;
	Result: C ← IO.0 , IO.0 ← C
	When IO.0 is a port to output pin, carry C will be sent to IO.0;
	When IO.0 is a port from input pin, IO.0 will be sent to carry C;
	Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
	Application Example1 (serial output) :
	-- 	
	...
	set1 pac.0 ; // set PA.0 as output
	...
	set0 flag.1 ; // C=0
	swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=0
	set1 flag.1 ; // C=1
	swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=1
	...
	-- 	
	Application Example2 (serial input) :
	-- 	
	...
	set0 pac.0 ; // set PA.0 as input
	...
	swapc pa.0 ; // read PA.0 to C (bit operation)
	src a ; // shift C to bit 7 of ACC
	swapc pa.0 ; // read PA.0 to C (bit operation)
	src a ; // shift new C to bit 7, old C
	...
	-- 	

set0 M.n	Set bit n of memory to low Example: set0 MEM.5 ; Result: set bit 5 of MEM to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 M.n	Set bit n of memory to high Example: *set1* MEM.5 ; Result: set bit 5 of MEM to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7.6. Conditional Operation Instructions

ceqsn a, I	Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as (a ← a – I) Example: *ceqsn* a, 0x55 ; *inc* MEM ; *goto* error ; Result: If a=0x55, then “goto error”; otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
ceqsn a, M	Compare ACC with memory and skip next instruction if both are equal. Flag will be changed like as (a ← a - M) Example: *ceqsn* a, MEM; Result: If a=MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, M	Compare ACC with memory and skip next instruction if both are not equal. Flag will be changed like as (a ← a - M) Example: *cneqsn* *a, MEM;* Result: If a≠MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
cneqsn a, I	Compare ACC with immediate data and skip next instruction if both are no equal. Flag will be changed like as (a ← a - I) Example: *cneqsn* *a,0x55 ;* *inc* *MEM ;* *goto* *error ;* Result: If a≠0x55, then “goto error”; Otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
t0sn IO.n	Check IO bit and skip next instruction if it’s low Example: *t0sn* pa.5; Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn IO.n	Check IO bit and skip next instruction if it’s high Example: *t1sn* pa.5 ; Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t0sn M.n	Check memory bit and skip next instruction if it’s low Example: *t0sn* MEM.5 ; Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn M.n	Check memory bit and skip next instruction if it’s high EX: *t1sn* MEM.5 ; Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
izsn a	Increment ACC and skip next instruction if ACC is zero Example: *izsn* a; Result: a ← a + 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

dzsn a	Decrement ACC and skip next instruction if ACC is zero Example: dzsn a; Result: A ← A - 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
izsn M	Increment memory and skip next instruction if memory is zero Example: izsn MEM; Result: MEM ← MEM + 1, skip next instruction if MEM= 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dzsn M	Decrement memory and skip next instruction if memory is zero Example: dzsn MEM; Result: MEM ← MEM - 1, skip next instruction if MEM = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

7.7. System control Instructions

call label	Function call, address can be full range address space
	Example: *call* function1;
	Result: [sp] ← pc + 1
	pc ← function1
	sp ← sp + 2
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

goto label	Go to specific address which can be full range address space Example: *goto* error; Result: Go to error and execute program. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret I	Place immediate data to ACC, then return Example: *ret* 0x55; Result: A ← 55h ret ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret	Return to program which had function call Example: *ret;* Result: sp ← sp - 2 pc ← [sp] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reti	Return to program from interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: *reti*; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nop	No operation Example: *nop*; Result: nothing changed Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
wdreset	Reset Watchdog timer. Example: *wdreset* ; Result: Reset Watchdog timer. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

pcadd a	Next program counter is current program counter plus ACC.
	Example: *pcadd a*;
	Result: pc ← pc + a
	Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
	Application Example:
	--- 	
	…
	mov a, 0x02 ;
	pcadd a ; // PC <- PC+2
	goto err1 ;
	goto correct ; // jump here
	goto err2 ;
	goto err3 ;
	…
	correct: // jump here
	…
	--- 	

engint	Enable global interrupt enable Example: *engint*; Result: Interrupt request can be sent to FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
disgint	Disable global interrupt enable Example: *disgint* ; Result: Interrupt request is blocked from CPU Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
stopsys	System halt. Example: *stopsys*; Result: Stop the system clocks and halt the system Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
stopexe	CPU halt. The oscillator module is still active to output clock, however, system clock is disabled to save power. Example: *stopexe*; Result: Stop the system clocks and keep oscillator modules active. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reset	Reset the whole chip, its operation will be same as hardware reset. Example: *reset*; Result: Reset the whole chip. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7.8. Summary of Instructions Execution Cycle

2T		*goto, call, idxm, pcadd, ret, reti*
2T	Condition is fulfilled	*ceqsn*, cneqsn,*t0sn, t1sn, dzsn, izsn*
1T	Condition is not fulfilled	
1T		Others

7.9. Summary of affected flags by Instructions

Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*ldt16* word	-	-	-	-
stt16 word	-	-	-	-	*idxm* a, index	-	-	-	-	*idxm* index, a	-	-	-	-
xch M	-	-	-	-	*pushaf*	-	-	-	-	*popaf*	Y	Y	Y	Y
add a, I	Y	Y	Y	Y	*add* a, M	Y	Y	Y	Y	*add* M, a	Y	Y	Y	Y
addc a, M	Y	Y	Y	Y	*addc* M, a	Y	Y	Y	Y	*addc* a	Y	Y	Y	Y
addc M	Y	Y	Y	Y	*sub* a, I	Y	Y	Y	Y	*sub* a, M	Y	Y	Y	Y
sub M, a	Y	Y	Y	Y	*subc* a, M	Y	Y	Y	Y	*subc* M, a	Y	Y	Y	Y
subc a	Y	Y	Y	Y	*subc* M	Y	Y	Y	Y	*inc* M	Y	Y	Y	Y
dec M	Y	Y	Y	Y	*clear* M	-	-	-	-	*sr* a	-	Y	-	-
src a	-	Y	-	-	*sr* M	-	Y	-	-	*src* M	-	Y	-	-
sl a	-	Y	-	-	*slc* a	-	Y	-	-	*sl* M	-	Y	-	-
slc M	-	Y	-	-	*swap* a	-	-	-	-	*and* a, I	Y	-	-	-
and a, M	Y	-	-	-	*and* M, a	Y	-	-	-	*or* a, I	Y	-	-	-
or a, M	Y	-	-	-	*or* M, a	Y	-	-	-	*xor* a, I	Y	-	-	-
xor IO, a	-	-	-	-	*xor* a, M	Y	-	-	-	*xor* M, a	Y	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-	*neg* a	Y	-	-	-
neg M	Y	-	-	-	*set0* IO.n	-	-	-	-	*set1* IO.n	-	-	-	-
set0 M.n	-	-	-	-	*set1* M.n	-	-	-	-	*ceqsn* a, I	Y	Y	Y	Y
ceqsn a, M	Y	Y	Y	Y	*t0sn* IO.n	-	-	-	-	*t1sn* IO.n	-	-	-	-
t0sn M.n	-	-	-	-	*t1sn* M.n	-	-	-	-	*izsn* a	Y	Y	Y	Y
dzsn a	Y	Y	Y	Y	*izsn* M	Y	Y	Y	Y	*dzsn* M	Y	Y	Y	Y
call label	-	-	-	-	*goto* label	-	-	-	-	*ret* I	-	-	-	-
ret	-	-	-	-	*reti*	-	-	-	-	*nop*	-	-	-	-
pcadd a	-	-	-	-	*engint*	-	-	-	-	*disgint*	-	-	-	-
stopsys	-	-	-	-	*stopexe*	-	-	-	-	*reset*	-	-	-	-
wdreset	-	-	-	-	*nadd M, a*	Y	Y	Y	Y	*cneqsn a, I*	Y	Y	Y	Y
cneqsn a, M	Y	Y	Y	Y	*comp a, M*	Y	Y	Y	Y	*nadd a, M*	Y	Y	Y	Y
comp M, a	Y	Y	Y	Y	*swapc IO.n*	-	Y	-	-	*steer* index	-	-	-	-
ldeer index	-	-	-	-										

7.10. BIT definition

Bit access of RAM is only available for address from 0x00 to 0x3F.

8. Code Options

Option	Selection	Description
Security	Enable	MTP content is protected and program cannot be read back
	Disable	MTP content is not protected so program can be read back
LVR	4.0V	Select LVR = 4.0V
	3.5V	Select LVR = 3.5V
	3.0V	Select LVR = 3.0V
	2.7V	Select LVR = 2.7V
	2.5V	Select LVR = 2.5V
	2.2V	Select LVR = 2.2V
	2.0V	Select LVR = 2.0V
	1.8V	Select LVR = 1.8V
Boot-up_Time	Slow	Please refer to t_{WUP} and t_{SBP} in Section 4.1
	Fast	Please refer to t_{WUP} and t_{SBP} in Section 4.1
GPC_TM2_LPWM	Disable	Comparator does not control all PWM outputs
	Enable	Comparator controls all PWM outputs (ICE does NOT Support.)
LPWM_Source	16MHZ	When Lpwmgclk.0= 1, LPWMG clock source = IHRC = 16MHZ
	32MHZ	When Lpwmgclk.0= 1, LPWMG clock source = IHRC*2 = 32MHZ (ICE does NOT Support.)
TM2_Source	16MHZ	When tm2c[7:4]= 0010, TM2 clock source = IHRC = 16MHZ
	32MHZ	When tm2c[7:4]= 0010, TM2 clock source = IHRC*2 = 32MHZ (ICE does NOT Support.)
TM2_Bit	6 Bit	When tm2s.7=1, TM2 PWM resolution is 6 Bit
	7 Bit	When tm2s.7=1, TM2 PWM resolution is 7 Bit (ICE does NOT Support.)
Comparator_Edge	All_Edge	The comparator will trigger an interrupt on the rising edge or falling edge
	Rising_Edge	The comparator will trigger an interrupt on the rising edge
	Falling_Edge	The comparator will trigger an interrupt on the falling edge

9. Special Notes

This chapter is to remind user who use PGS152 series IC in order to avoid frequent errors upon operation.

9.1. Using IC

9.1.1. IO pin usage and setting

(1) IO pin is set to be digital input

◆ When IO is set as digital input, the level of Vih and Vil would changes with the voltage and temperature. Please follow the minimum value of Vih and the maximum value of Vil.

◆ The value of internal pull high resistor would also changes with the voltage, temperature and pin voltage. It is not the fixed value.

(2) IO pin as digital input and enable wakeup function

◆ Configure IO pin as input

◆ Set corresponding bit to “1” in PXDIER

◆ For those IO pins of PA that are not used, PADIER[1:2] should be set low in order to prevent them from leakage.

(3) PA5 is set to be PRSTB input pin

◆ Configure PA5 as input

◆ Set CLKMD.0=1 to enable PA5 as PRSTB input pin

(4) PA5 is set to be input pin and to connect with a push button or a switch by a long wire

◆ Needs to put a >33Ω resistor in between PA5 and the long wire

◆ Avoid using PA5 as input in such application.

(5) PA7 and PA6 as external crystal oscillator

◆ Configure PA7 and PA6 as input

◆ Disable PA7 and PA6 internal pull- high resistor

◆ Configure PADIER register to set PA6 and PA7 as analog input

◆ Ensure EOSC working well before switching from IHRC or ILRC to EOSC.

Note: Please read the PMC-APN013 carefully. According to PMC-APN013, the crystal oscillator should be used reasonably. If the following situations happen to cause IC start-up slowly or non-startup, PADAUK Technology is not responsible for this: the quality of the user's crystal oscillator is not good, the usage conditions are unreasonable, the PCB cleaner leakage current, or the PCB layouts are unreasonable.

9.1.2. Interrupt

(1) When using the interrupt function, the procedure should be:

Step1: Set INTEN register, enable the interrupt control bit

Step2: Clear INTRQ register

Step3: In the main program, using ENGINT to enable CPU interrupt function

Step4: Wait for interrupt. When interrupt occurs, enter to Interrupt Service Routine

Step5: After the Interrupt Service Routine being executed, return to the main program

*Use DISGINT in the main program to disable all interrupts

*When interrupt service routine starts, use PUSHAF instruction to save ALU and FLAG register. POPAF instruction is to restore ALU and FLAG register before RETI as below:

```
void Interrupt (void)    // Once the interrupt occurs, jump to interrupt service routine
{                        // enter DISGINT status automatically, no more interrupt is
        accepted
        PUSHAF;
        …
        POPAF;
}   // RETI will be added automatically. After RETI being executed, ENGINT status
will be restored
```

(2) INTEN and INTRQ have no initial values. Please set required value before enabling interrupt function.

9.1.3. System clock switching

System clock can be switched by CLKMD register. **Please notice that, NEVER switch the system clock and turn off the original clock source at the same time.** For example: When switching from clock A to clock B, please switch to clock B first; and after that turn off the clock A oscillator through CLKMD.

◆ Example : Switch system clock from ILRC to IHRC/2

```
CLKMD   =   0x36;        // switch to IHRC, ILRC can not be disabled here
CLKMD.2 =   0;           // ILRC can be disabled at this time
```

◆ **ERROR:** Switch ILRC to IHRC and turn off ILRC simultaneously

```
CLKMD   =   0x50;        // MCU will hang
```

9.1.4. Watchdog

Watchdog will be inactive once ILRC is disabled.

9.1.5. TIMER time out

When select $ INTEGS BIT_R (default value) and T16M counter BIT8 to generate interrupt, if T16M counts from 0, the first interrupt will occur when the counter reaches to 0x100 (BIT8 from 0 to 1) and the second interrupt will occur when the counter reaches 0x300 (BIT8 from 0 to 1). Therefore, selecting BIT8 as 1 to generate interrupt means that the interrupt occurs every 512 counts. Please notice that if T16M counter is restarted, the next interrupt will occur once Bit8 turns from 0 to 1.

If select $ INTEGS BIT_F(BIT triggers from 1 to 0) and T16M counter BIT8 to generate interrupt, the T16M counter changes to an interrupt every 0x200/0x400/0x600/. Please pay attention to two differences with setting INTEGS methods.

9.1.6. IHRC

(1) The IHRC frequency calibration is performed when IC is programmed by the writer.

(2) Because the characteristic of the Epoxy Molding Compound (EMC) would some degrees affects the IHRC frequency (either for package or COB), if the calibration is done before molding process, the actual IHRC frequency after molding may be deviated or becomes out of spec. Normally , the frequency is getting slower a bit.

(3) It usually happens in COB package or Quick Turnover Programming (QTP). And PADAUK would not take any responsibility for this situation.

(4) Users can make some compensatory adjustments according to their own experiences. For example, users can set IHRC frequency to be 0.5% ~ 1% higher and aim to get better re-targeting after molding.

9.1.7. LVR

LVR level selection is done at compile time. User must select LVR based on the system working frequency and power supply voltage to make the MCU work stably.

The following are Suggestions for setting operating frequency, power supply voltage and LVR level:

SYSCLK	VDD	LVR
2MHz	≧ 1.8V	≧ 1.8V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.5V	≧ 3.5V

Table 8: LVR setting for reference

(1) The setting of LVR (1.8V ~ 4.0V) will be valid just after successful power-on process.
(2) User can set MISC.2 as “1” to disable LVR. However, V_{DD} must be kept as exceeding the lowest working voltage of chip; Otherwise IC may work abnormally.
(3) The LVR function will be invalid when IC in stopexe or stopsys mode.

9.1.8. Programming Writing

- Please use 5S-P-003(x) to program. 3S-P-002 or older versions do not support programming PGS152.
- There are 5 pins for using the writer to program:, PA3, PA5, PA6, VDD and GND
- Special notes about voltage and current while Multi-Chip-Package(MCP) or On-Board Programming
  1. PA5 (V_{PP}) may be higher than 5.0V.
  2. V_{DD} may be higher than 5.0V, and its maximum current may reach about 20mA.
  3. All other signal pins level (except GND) are the same as V_{DD}.
  4. User should confirm when using this product in MCP or On-Board Programming, the peripheral circuit or components will not be destroyed or limit the above voltages.

- **Important Cautions:**
  1. You MUST follow the instructions on APN004 and APN011 for programming IC on the handler.
  2. Connecting a 0.01uF capacitor between VDD and GND at the handler port to the IC is always good for suppressing disturbance. But please DO NOT connect with > 0.01uF capacitor, otherwise, programming mode may be fail.

Please select the following program mode according to the actual situation.

Normal Programming Mode

Jumper connection: Please follow the instruction inside the writer software to connect the jumper and refer the file of Writer “5S-P-003 UM to learn JP7 jumper method when using 5S-P-003(x) to program.

.

On-Board Writing Mode

On-Board Writing is known as the situation that the IC have to be programmed when the IC itself and other peripheral circuits and devices have already been mounted on the PCB. The wires of 5S-P-003(x) are used for On-Board Writing: ICPCK(PA3), ICPDA(PA6), VDD, GND and ICVPP(PA5). They are used to connect PA3, PA6, VDD, GND and PA5 of the IC correspondingly.

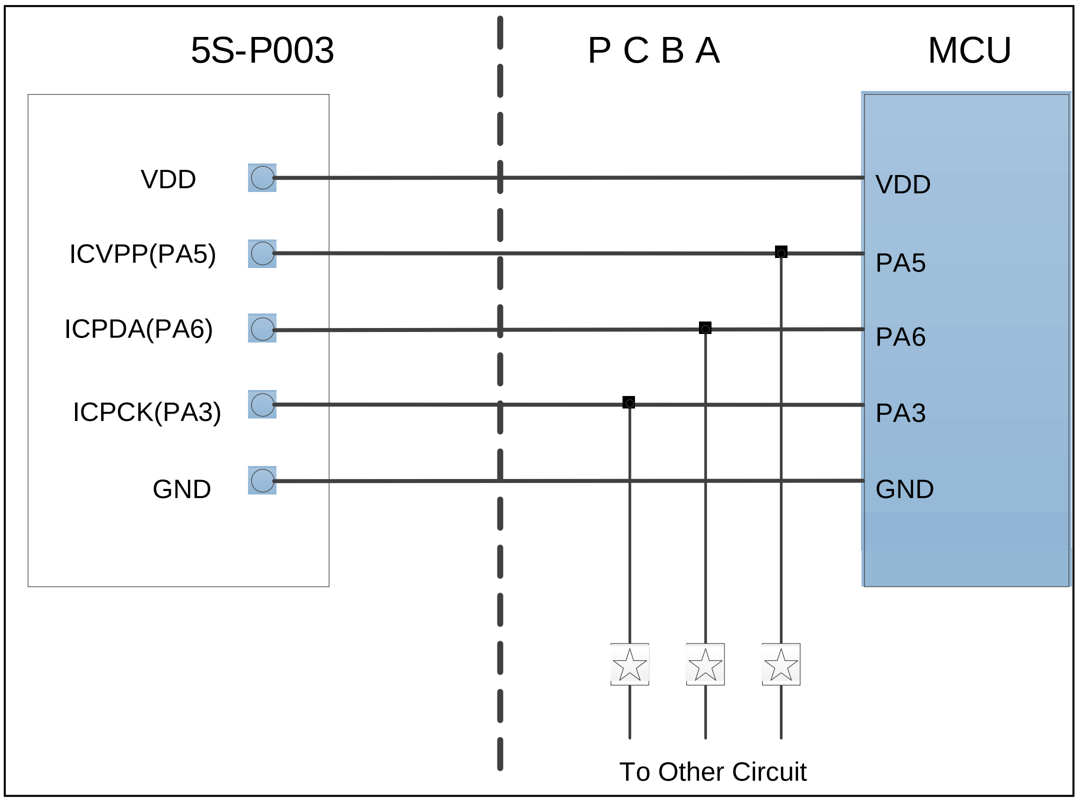


Fig. 21: Schematic Diagram of On-Board Wiring

The symbol ☆ on Fig. 21 can be either resistors or capacitors. They are used to isolate the programming signal wires from the peripheral circuit. It should be≧ 10KΩ for resistance while ≦ 220pF for capacitance.

Notice:

- In general, the limited-voltage programming mode is used in On-board Writing, Please refers to the 13.2 for more detail about limited-voltage programming mode.
- Any zener diode ≦ 5.0V, or any circuitry which clam the 5.0V to be created SHOULD NOT be connected between VDD and GND of the PCB.
- Any capacitor ≧ 500uF SHOULD NOT be connected between VDD and GND of the PCB.
- In general, the writing signal pins PA4, PA5 and PA6 SHOULD NOT be considered as strong output pins.

9.3. Using ICE

5S-I-S01/2 (B) does not support the emulation of PGS152 1-FPPA MCU, please use 6S-M-001 to emulate PGS152 single-core MCU. The following are the notes for using 6S-M-001 to emulate PGS152:

- 6S-M-001 emulates PGS152, not support ***Tm2C.gpcrs***, ***PA4*** function.
- 6S-M-001 emulates PGS152, not support EOSCR built-in capacitor.
- 6S-M-001 emulates PGS152, not support *LVDC*, *OPR3* register.
- When emulating PWM waveforms, it is recommended that users view the waveforms while the program is running. When the emulator pauses or steps, the waveforms may not conform to reality.
- The ILRC frequency of the 6S-M-001 emulator is different from the actual IC and is uncalibrated, with a frequency range of about 40K~80KHz.
- When using 6S-M-001 simulation, it is not recommended to change tm2ct to affect the interruption period of timer2, because it does not conform to actual IC cycle.
- Please compile using IDE version after 0.95C1.

.