PGS134

8bit MCU with 12-bit ADC & EEPROM

Datasheet

Version 0.03 – Jun. 8, 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
0.02	2023/03/10	1. Updated "IMPORTANT NOTICE" 2. Amend Section 5.4, 5.16, 6.15 3. Updated Code Options (Chapter 8) 4. Updated the instructions in Sections 9.1.1, 9.1.8 5. Other known details bug correct.
0.03	2023/06/08	1. Updated P_{cycle}、f_{IHRC} in Sections 4.1 2. Updated Code Options (Chapter 8) 3. Updated the instructions in Sections 9.1.1, 9.1.8 4. Updated Other precautions 5. 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=168&kw=PGS134

(The following picture are for reference only.)

PGS134

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

Feature Documents Software &Tools Application Note

Content	Description	Download (CN)	Download (EN)
APN001	Output impedance of ADC analog signal source		
APN002	Over voltage protection		
APN003	Over voltage protection		
APN004	Semi-Automatic writing handler		
APN005	Effects of over voltage input to ADC		
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

- 4K words MTP
- 512 bytes EEPROM
- 256 bytes SRAM(128 * 16 configuration)
- One hardware 16-bit timer
- Two hardware 8-bit timers with PWM generation
- Three hardware 11-bit PWM generators (PWMG0, PWMG1 & PWMG2)
- One hardware comparator
- Bandgap circuit to provide 1.20V reference voltage
- Up to 14-channel 12-bit resolution ADC with one channel comes from internal Bandgap reference voltage or 0.25*V_{DD}
- ADC reference high voltage: external input, internal VDD, Bandgap 1.2V、1.6V、2.0V、2.4V、3.0V and 4.0V
- One 1T 8x8 hardware multiplier
- Max. 22 IO pins with optional pull-high and pull-low resistor
- Two different IO Driving capability group to meet different application requirements
  (1) PB4, PB7 Drive/ Sink Current= 30mA/35mA (Strong) and 13mA/17mA (Normal)
  (2) Other IOs (except PA5) Drive/ Sink Current = 11mA/(13 or 20) mA
- Every IO pin can be configured to enable wake-up function
- Built-in VDD/2 LCD bias voltage generator to provide maximum 4x17 dots LCD display
- Clock sources: IHRC, ILRC and EOSC (XTAL)
- For every wake-up enabled IO, two optional wake-up speed are supported: normal and fast
- Eight levels of LVR reset: 4.0V, 3.5V, 3.0V, 2.7V, 2.5V, 2.2V, 2.0V, 1.8V
- Two selectable external interrupt pins by code option

1.3. CPU Features

- 8bit high performance RISC CPU
- 98 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

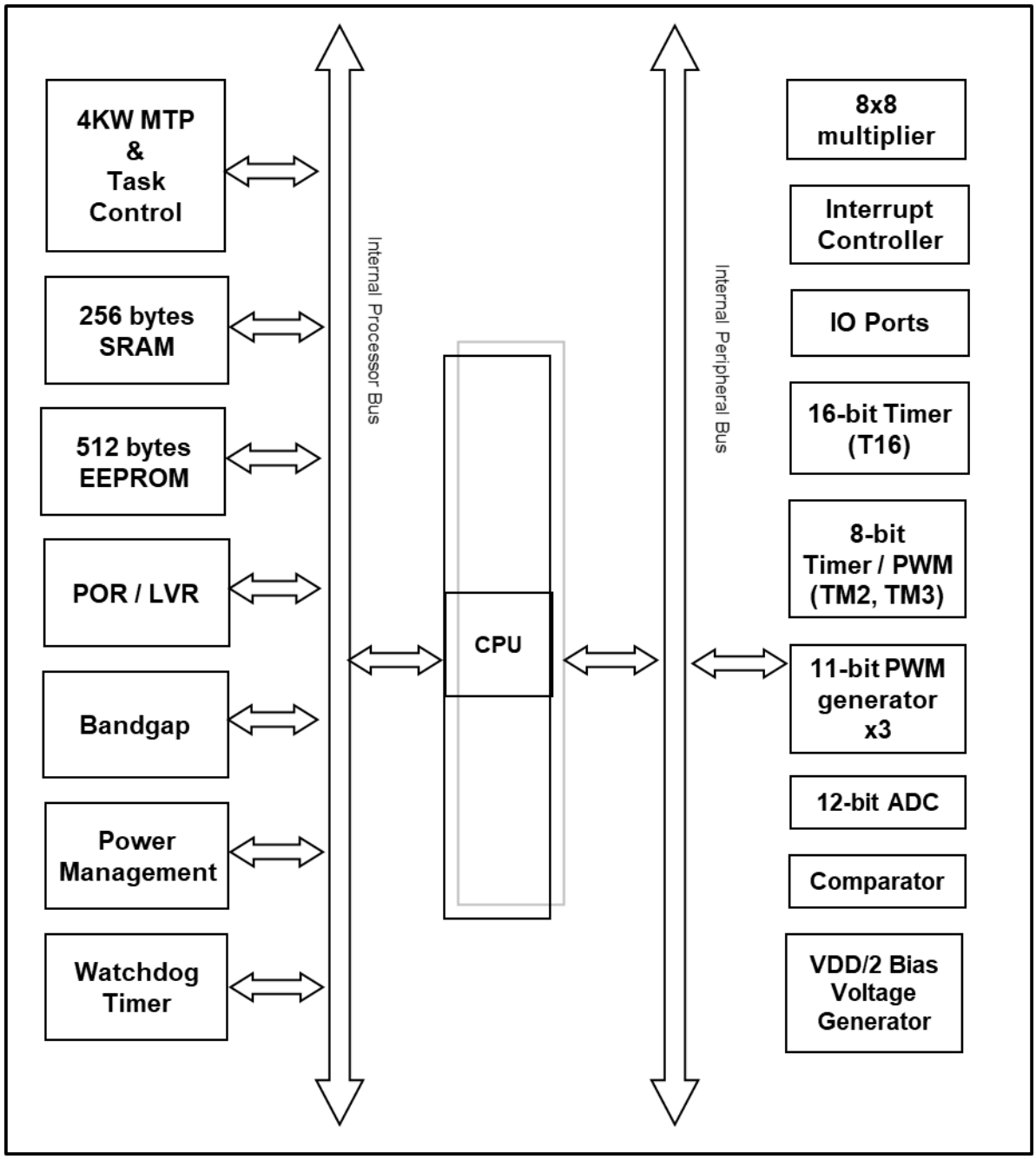
- ✧ PGS134-S08: SOP8 (150mil)
- ✧ PGS134-M10: MSOP10 (118mil)
- ✧ PGS134-S14: SOP14 (150mil)
- ✧ PGS134-S16A: SOP16A (150mil)
- ✧ PGS134-S16B: SOP16B (150mil)
- ✧ PGS134-S20: SOP20 (300mil)
- ✧ PGS134-H20: HTSOP20 (150mil)
- ✧ PGS134-S24: SOP24 (300mil)
- ✧ PGS134-Y24: SSOP24 (150mil)
- ✧ PGS134-4N10: DFN3*3-10P (0.5pitch)
- ✧ PGS134-2J16A: QFN4*4-16P (0.65pitch)
- ✧ PGS134-1J16A: QFN3*3-16P (0.5pitch)
- ✧ PGS134-2J24: QFN4*4-24P (0.5pitch)

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

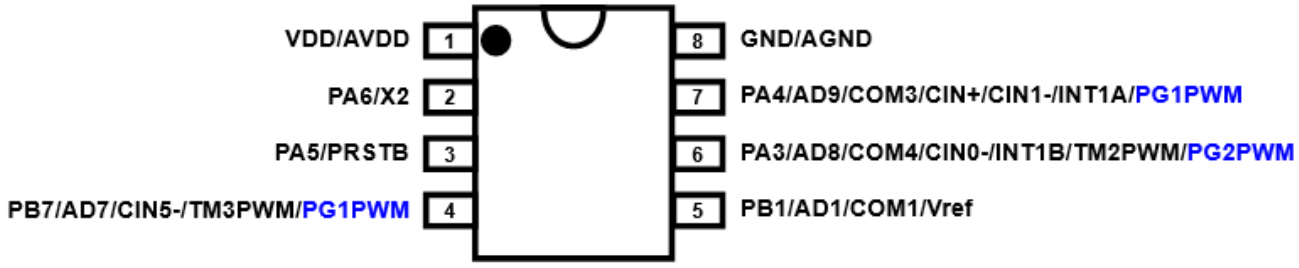
2. General Description and Block Diagram

The PGS134 family is an MTP-based CMOS 8-bit microcontroller with 12bit ADC. 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.

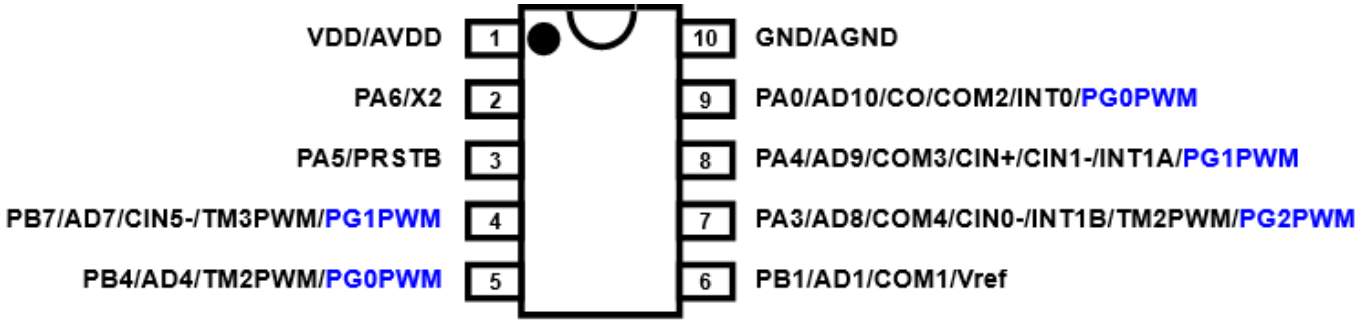
4KW MTP program memory, 512 bytes EEPROM and 256 bytes data SRAM are inside. One up to 14 channels 12-bit ADC is built inside the chip with multiple reference voltage sources selectable. PGS134 also provides six hardware timers: one is 16-bit timer, two are 8-bit timers with PWM generation, and three hardware 11-bit timers with PWM generation are also included. PGS134 also supports one hardware comparator and VDD/2 LCD bias voltage generator for LCD display application.



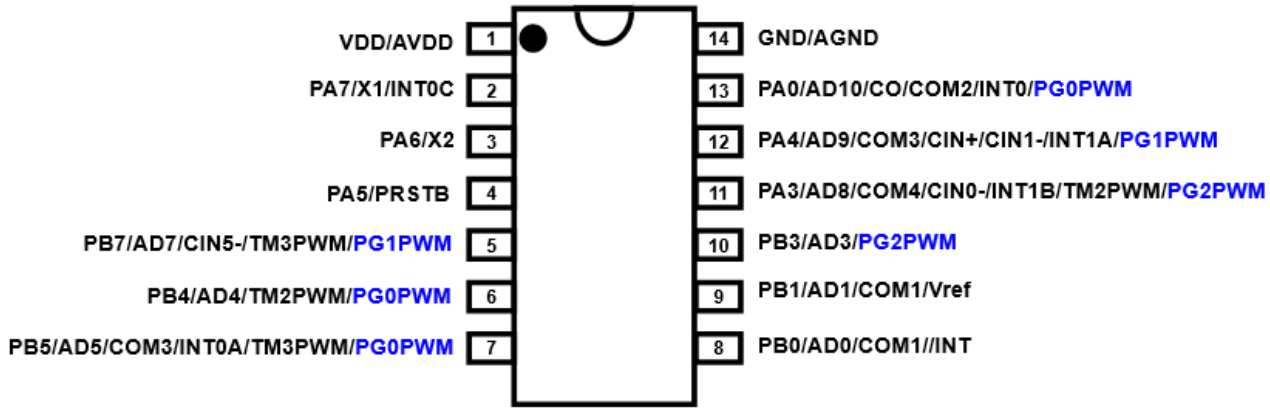
3. Pin Assignment and Description



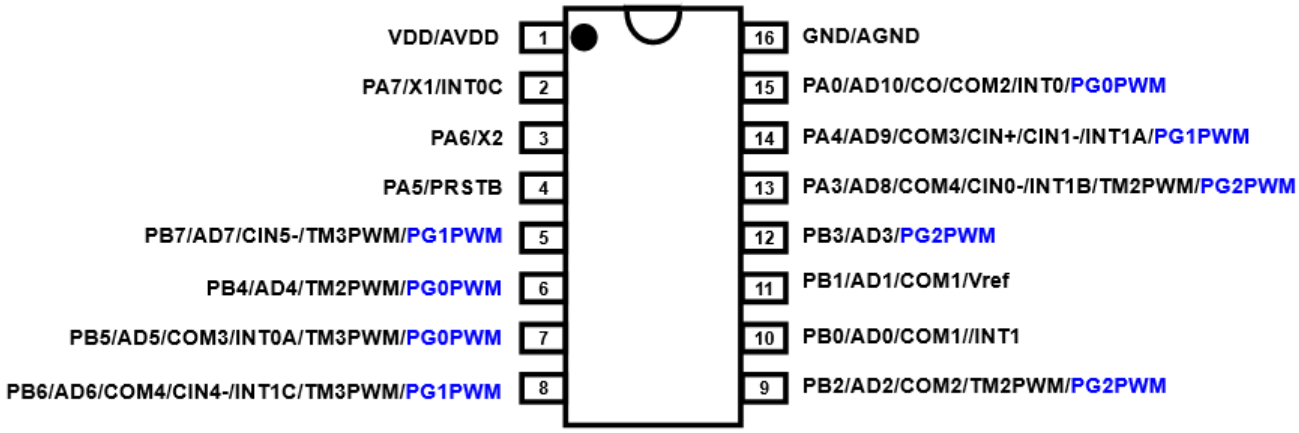
PGS134-S08 (SOP8-150mil)



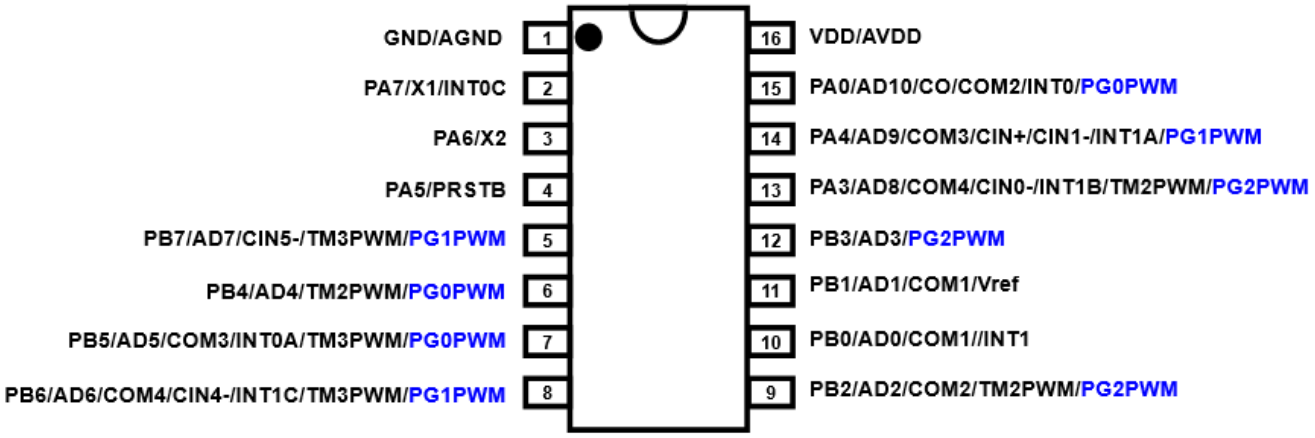
PGS134-M10 (MSOP10-118mil)



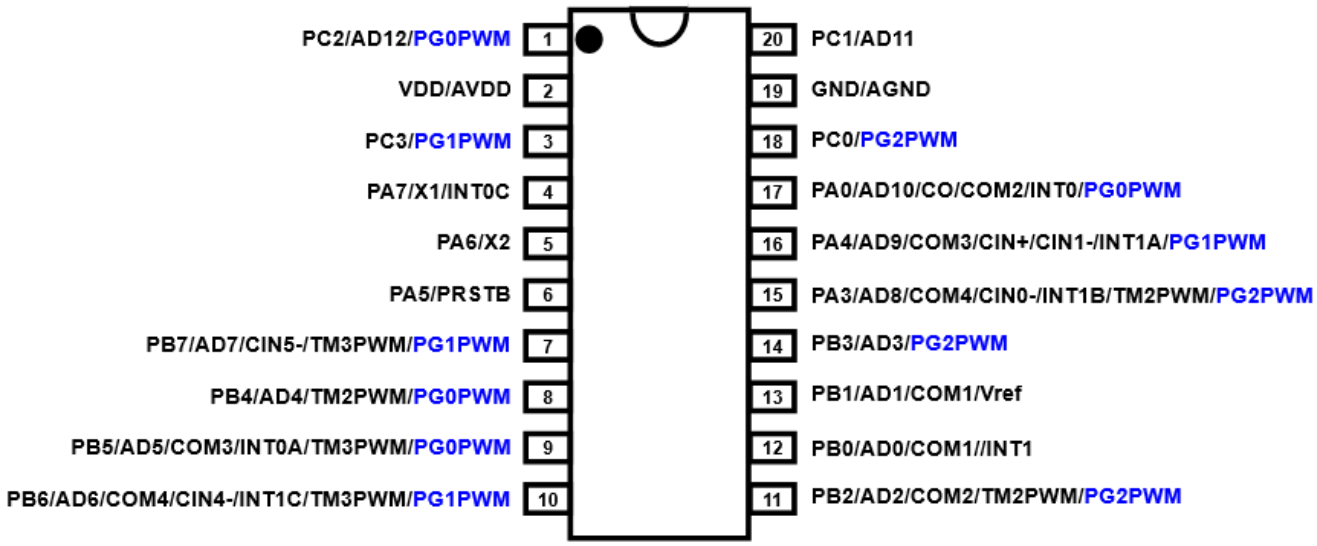
PGS134-S14 (SOP14-150mil)



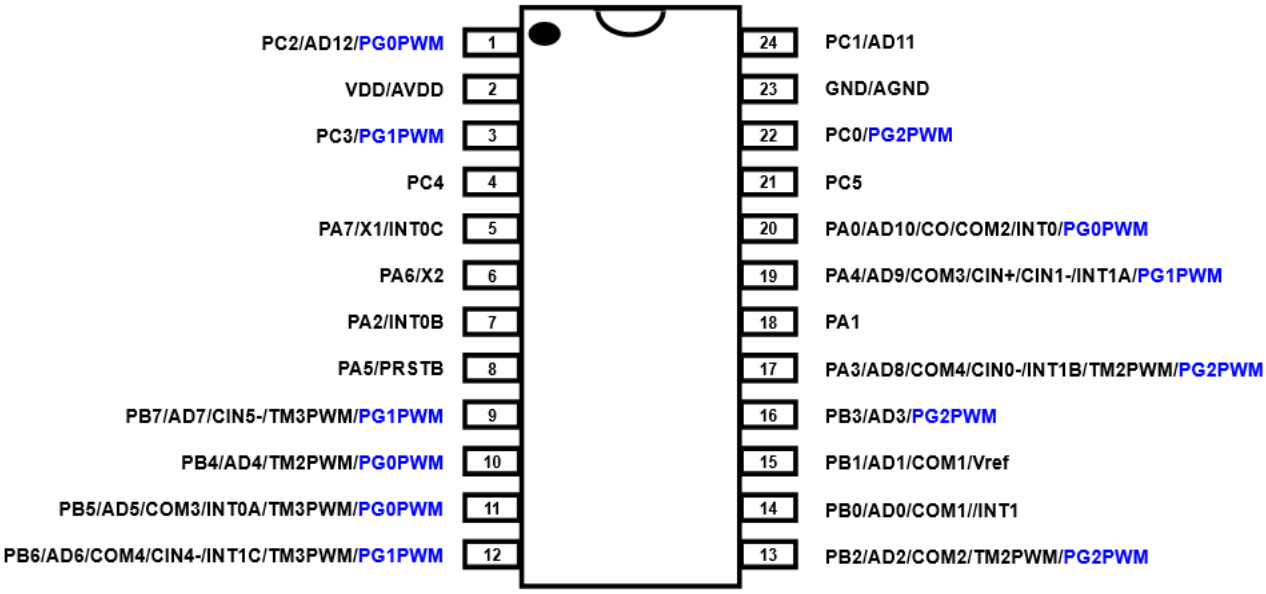
PGS134-S16A (SOP16A-150mil)



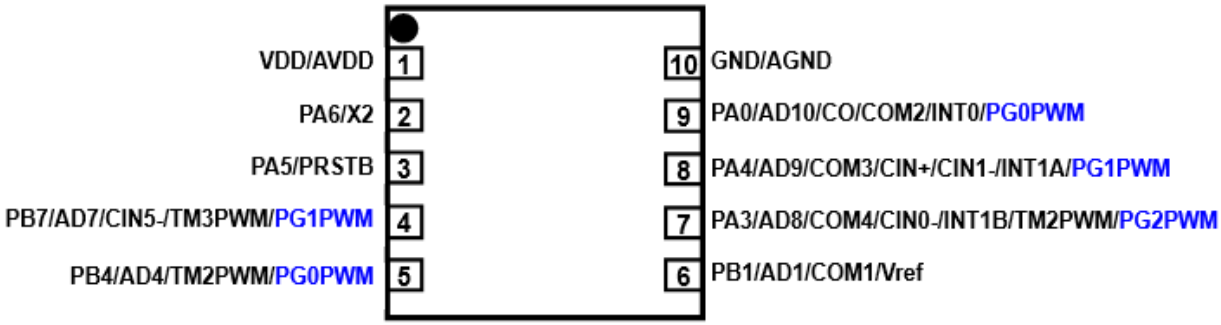
PGS134-S16B (SOP16B-150mil)



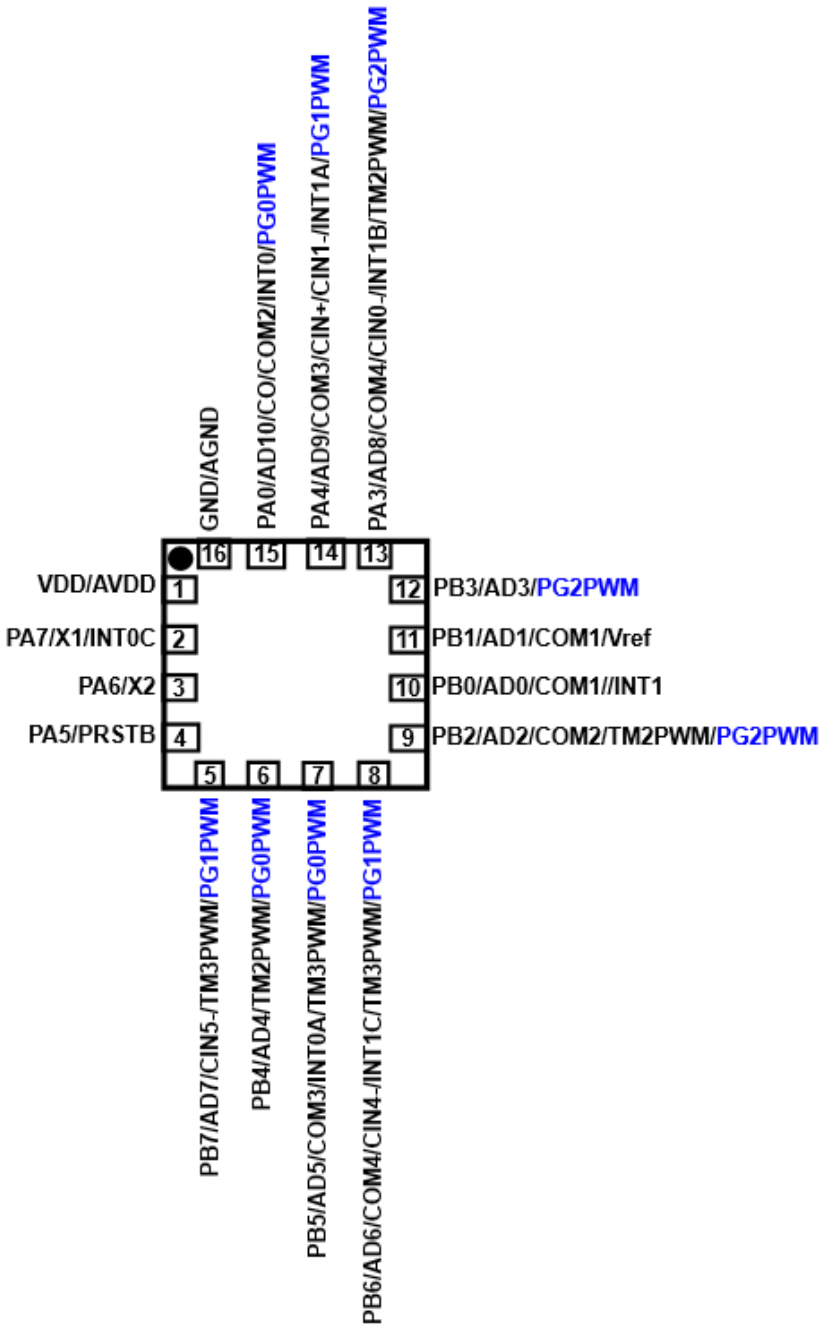
PGS134-S20 (SOP20-300mil)
PGS134-H20 (HTSOP20-150mil)



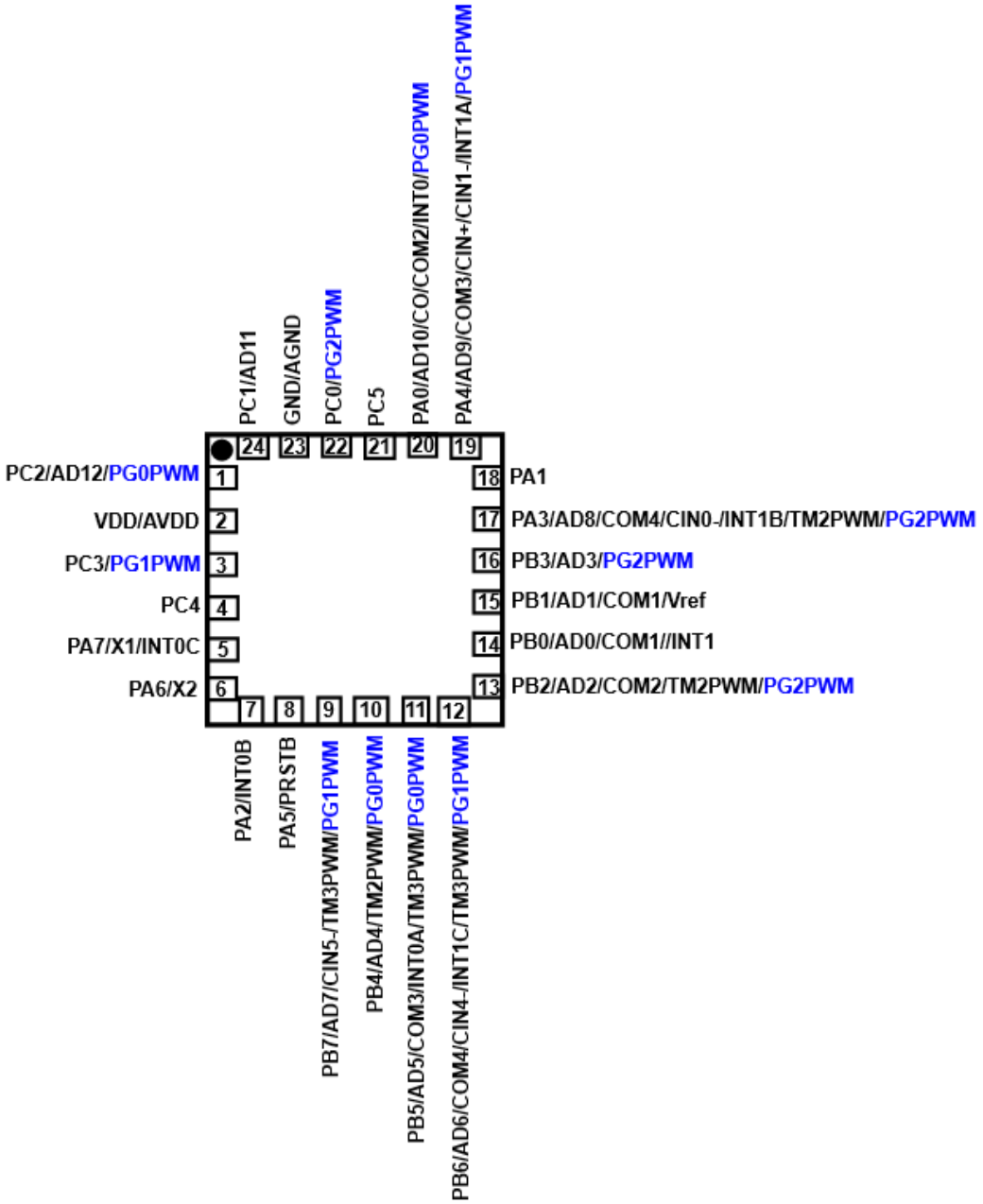
PGS134-S24 (SOP24-300mil)
PGS134-Y24 (SSOP24-150mil)



PGS134-4N10 (DFN3*3-10P-0.5pitch)



PGS134-2J16A (QFN4*4-16P-0.65pitch)
PGS134-1J16A (QFN3*3-16P-0.5pitch)



PGS134-2J24 (QFN4*4-24P-0.5pitch)

Pin Name	Pin Type & Buffer Type	Description
PA7 / X1 / INT0C	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 and pull-low resistor. (2) X1 is Crystal XIN(X1) when crystal oscillator is used. (3) External interrupt line 0C. It can be used as an external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting 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 and pull-low resistor. (2) X2 is Crystal XOUT(X2) 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	IO (OD) 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 and pull-low resistor. (2) Hardware reset. 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".
PA4 / AD9 / COM3 / CIN+ / CIN1- /INT1A / PG1PWM	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 and pull-low resistor by software independently (2) Channel 9 of ADC analog input (3) COM3 to provide (1/2 V_{DD}) for LCD display (4) Plus input source of comparator. (5) Minus input source 1 of comparator. (6) External interrupt line 1A. It can be used as an external interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting (7) Output of 11-bit PWM generator PWMG1 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 function by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PA3 / AD8 / COM4 / CIN0- / INT1B / TM2PWM / PG2PWM	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 and pull-low resistor independently by software (2) Channel 8 of ADC analog input (3) COM4 to provide (1/2 V_{DD}) for LCD display (4) Minus input source 0 of comparator. (5) External interrupt line 1B. It can be used as an external interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting (6) PWM output from Timer2 (7) Output of 11-bit PWM generator PWMG2 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 function by toggling this pin is also disabled.
PA2 /INT0B	IO ST / CMOS	The functions of this pin can be: (1) Bit 2 of port A. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) External interrupt line 0B. It can be used as an external interrupt line 0. 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 2 of register ***padier*** to disable the digital input to prevent current leakage. The bit 2 of ***padier*** register can be set to “0” to disable digital input; wake-up function by toggling this pin is also disabled.
PA1	IO ST / CMOS	The functions of this pin can be: (1) Bit 1 of port A. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software The bit 1 of ***padier*** register can be set to “0” to disable wake-up function by toggling this pin.
PA0 / AD10 / COM2 / CO / INT0 / PG0PWM	IO ST / CMOS / Analog	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 and pull-low resistor independently by software (2) Channel 10 of ADC analog input (3) COM2 to provide (1/2 V_{DD}) for LCD display (4) Output of comparator. (5) External interrupt line 0. It can be used as an external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting (6) Output of 11-bit PWM generator PWMG0. When this pin is configured as analog input, please use bit 0 of register ***padier*** to disable the digital input to prevent current leakage. The bit 0 of ***padier*** register can be set to “0” to disable digital input; wake-up function by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PB7 / AD7 / CIN5- / TM3PWM / PG1PWM	IO ST / CMOS / Analog	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 and pull-low resistor independently by software (2) Channel 7 of ADC analog input (3) Minus input source 5 of comparator. (4) PWM output from Timer3 (5) Output of 11-bit PWM generator PWMG1 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 function by toggling this pin is also disabled.
PB6 / AD6 / COM4 / CIN4- / INT1C / TM3PWM / PG1PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 6 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 6 of ADC analog input (3) COM4 to provide (1/2 V_{DD}) for LCD display (4) Minus input source 4 of comparator. (5) External interrupt line 1C. It can be used as an external interrupt line 1. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting. (6) PWM output from Timer3 (7) Output of 11-bit PWM generator PWMG1 When this pin is configured as analog input, please use bit 6 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 6 of ***pbdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
PB5 / AD5 / COM3 / INT0A / TM3PWM / PG0PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 5 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 5 of ADC analog input (3) COM3 to provide (1/2 V_{DD}) for LCD display (4) External interrupt line 0A. It can be used as an external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service and configurable by register setting (5) PWM output from Timer3 (6) Output of 11-bit PWM generator PWMG0 When this pin is configured as analog input, please use bit 5 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 5 of ***pbdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PB4 / AD4 / TM2PWM / PG0PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 4 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 4 of ADC analog input (3) PWM output from Timer2 (4) Output of 11-bit PWM generator PWMG0 When this pin is configured as analog input, please use bit 4 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 4 of ***pbdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
PB3 / AD3 / PG2PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 3 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 3 of ADC analog input (3) Output of 11-bit PWM generator PWMG2 When this pin is configured as analog input, please use bit 3 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 3 of ***pbdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
PB2 / AD2 / COM2 / TM2PWM / PG2PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 2 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 2 of ADC analog input (3) COM2 to provide (1/2 V_{DD}) for LCD display (4) PWM output from Timer2 (5) Output of 11-bit PWM generator PWMG2. When this pin is configured as analog input, please use ***pbdier.2*** to disable the digital input to prevent current leakage. The ***pbdier.2*** can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
PB1 / AD1 / COM1 / Vref	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 1 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 1 of ADC analog input (3) COM1 to provide (1/2 V_{DD}) for LCD display (4) External reference high voltage for ADC. When this pin is configured as analog input, please use bit 1 of register ***pbdier*** to disable the digital input to prevent current leakage. The bit 1 of ***pbdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PB0 / AD0 / COM1 / INT1	IO ST / CMOS / Analog	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 and pull-low resistor independently by software (2) Channel 0 of ADC analog input (3) COM1 to provide (1/2 V_{DD}) for LCD display (4) External interrupt line 1. It can be used as an 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 function by toggling this pin is also disabled.
PC5	IO ST / CMOS	The function of this pin can be: (1) Bit 5 of port C. It can be configured as digital input, two-state output with pull-high and pull-low resistor independently by software The bit 5 of ***pcdier*** register can be set to "0" to disable digital input; wake-up from power-down by toggling this pin is also disabled.
PC4	IO ST / CMOS	The function of this pin can be: (1) Bit 4 of port C. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software The bit 4 of ***pcdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
PC3 / PG1PWM	IO ST / CMOS	The functions of this pin can be: (1) Bit 3 of port B. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Output of 11-bit PWM generator PWMG1. The bit 3 of ***pcdier*** register can be set to "0"to disable digital input; wake-up function by toggling this pin is also disabled.
PC2 / AD12 / PG0PWM	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 2 of port C. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 12 of ADC analog input (3) Output of 11-bit PWM generator PWMG0 When this pin is configured as analog input, please use bit 2 of register ***pcdier*** to disable the digital input to prevent current leakage. The bit 2 of ***pcdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
PC1 / AD11	IO ST / CMOS / Analog	The functions of this pin can be: (1) Bit 1 of port C. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Channel 11 of ADC analog input When this pin is configured as analog input, please use bit 2 of register ***pcdier*** to disable the digital input to prevent current leakage. The bit 2 of ***pcdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.

Pin Name	Pin Type & Buffer Type	Description
PC0 / PG2PWM	IO ST / CMOS	The functions of this pin can be: (1) Bit 0 of port C. It can be configured as digital input or two-state output, with pull-high and pull-low resistor independently by software (2) Output of 11-bit PWM generator PWMG2 The bit 0 of ***pcdier*** register can be set to "0" to disable digital input; wake-up function by toggling this pin is also disabled.
VDD/ AVDD	VDD/ AVDD	VDD: Digital positive power AVDD: Analog positive power VDD is the IC power supply while AVDD is the ADC power supply. AVDD and VDD are double bonding internally and they have the same external pin.
GND / AGND	GND / AGND	GND: Digital negative power AGND: Analog negative power GND is the IC ground pin while AGND is the ADC ground pin. AGND and GND are double bonding internally and they have the same external pin.
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 V_{DD}=5.0V, f_{SYS} =2MHz unless noted.

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25°C)
V_{DD}	Operating Voltage	2.2#	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	 82K	 8M 4M 2M	Hz	 V_{DD} ≧ 3.5V V_{DD} ≧ 2.5V V_{DD} ≧ 2.2V V_{DD} = 5.0V
P_{cycle}	EEPROM Program cycle	100K			Cycle	
P_{read}	EEPROM Read Voltage	2.2 #		V_{DD}	V	# V_{DD} voltage
P_{pgm}	EEPROM Erase/Program Voltage	2.2 #		V_{DD}	V	# V_{DD} voltage
V_{POR}	Power On Reset Voltage		2.0*			* Subject to LVR tolerance
I_{OP}	Operating Current		0.6 80		mA uA	f_{SYS}=IHRC/16=1MIPS@5.0V f_{SYS}=ILRC=82KHz@5.0V
I_{PD}	Power Down Current (by ***stopsys*** command)		0.9 0.5		uA uA	f_{SYS}= 0Hz,V_{DD}=5.0V f_{SYS}= 0Hz,V_{DD}=3.3V
I_{PS}	Power Save Current (by ***stopexe*** command)		4.0		uA	V_{DD} =5.0V; f_{SYS}= ILRC Only ILRC module is enabled.
V_{IL}	Input low voltage for IO lines	0		0.2 V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO lines Sink current (PB4/PB7 current can be switched through *PB4_PB7_Drive* in the Code Option)					
	PB4, PB7 (Normal) PB4, PB7 (Strong) PA0-4, PB2, PB5-6 PA5-7, PB0-1, PB3, PC0-5		24 40 25 15		mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO lines Drive current					
	PB4, PB7 (Normal) PB4, PB7 (Strong) Other IOs		17 32 14		mA	V_{DD}=5.0V, V_{OH}=4.5V
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		62		KΩ	V_{DD} =5.0V
V_{BG}	Bandgap Reference Voltage	1.145*	1.20*	1.255*	V	V_{DD} =2.2V ~ 5.5V -40°C <Ta<85°C*
f_{IHRC}	Frequency of IHRC after calibration *	15.76*	16*	16.24*	MHz	25°C, V_{DD} =2.2V~5.5V
		15.20*	16*	16.80*		V_{DD} =2.2V~5.5V, -40°C <Ta<85°C*
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 5.0V
V_{ADC}	ADC working voltage	2.2		V_{DD}	V	

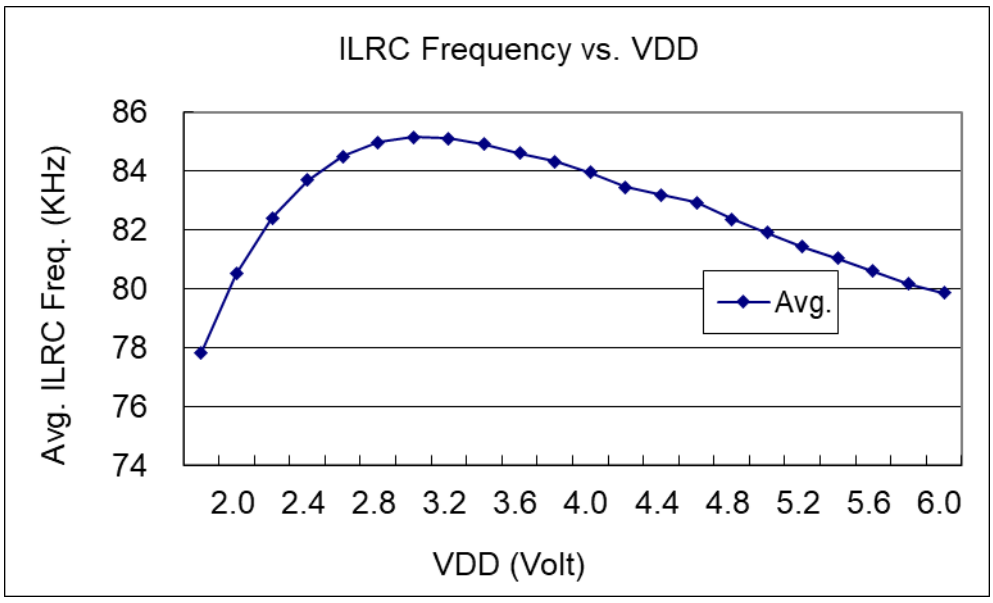
Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25ºC)
V_{AD}	AD Input Voltage	0		V_{DD}	V	
ADrs	ADC resolution			12	bit	
ADcs	ADC current consumption		1.1 1		mA	@5V @3V
ADclk	ADC clock period		2		us	2.2V ~ 5.5V
t_{ADCONV}	ADC conversion time (t_{ADCLK} is the period of the selected AD conversion clock)		16		t_{ADCLK}	12-bit resolution
AD DNL	ADC Differential Non-Linearity		±2*		LSB	
AD INL	ADC Integral Non-Linearity		±4*		LSB	
ADos	ADC offset		2*		mV	@ V_{DD} =3V
V_{REFH}	ADC reference high voltage 4V 3V 2V	 3.90* 2.93* 1.95*	 4* 3* 2*	 4.10* 3.07* 2.05*	V	@ V_{DD} =5V, 25 ºC
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 (fast)		45		T_{ILRC}	Where T_{ILRC} is the time period of ILRC
	Wake-up time period (normal)		3000			
t_{SBP}	System boot-up period from power-on for Normal boot-up		28		ms	V_{DD} =5V
	System boot-up period from power-on for Fast boot-up		620		us	V_{DD} =5V
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.

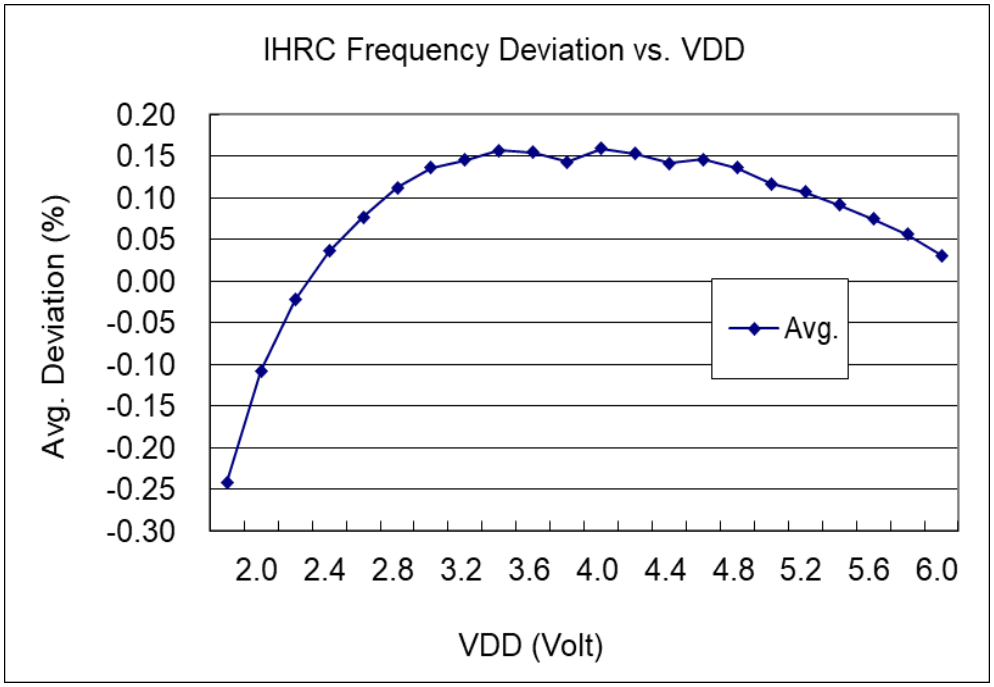
4.2. Absolute Maximum Ratings

- Supply Voltage ……..... 2.2V ~ 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
- Junction Temperature …… 150°C
- Storage Temperature …… -50°C ~ 125°C

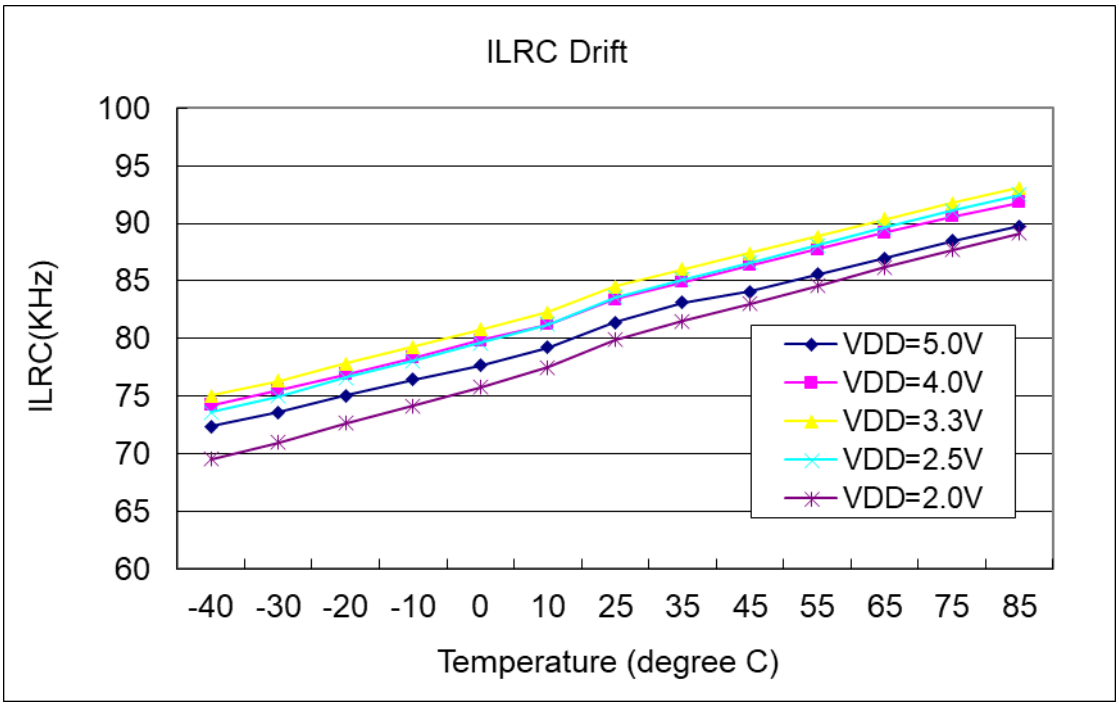
4.3. Typical ILRC frequency vs. VDD



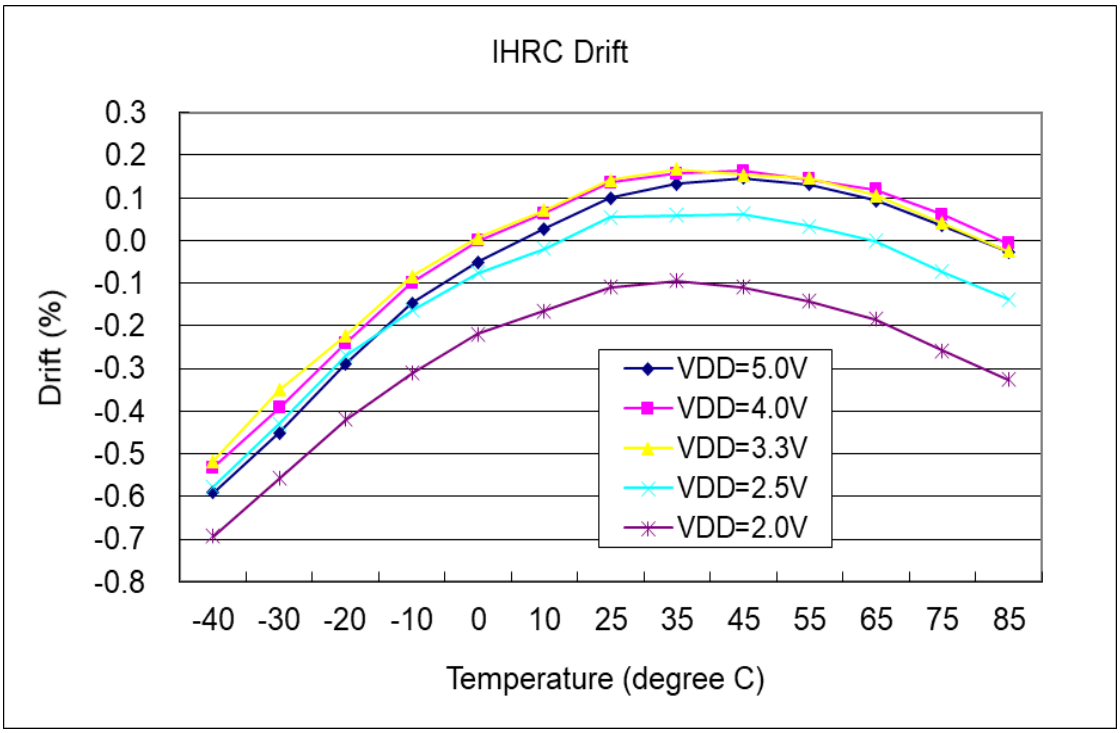
4.4. Typical IHRC frequency deviation vs. VDD (calibrated to 16MHz)



4.5. Typical ILRC Frequency vs. Temperature



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

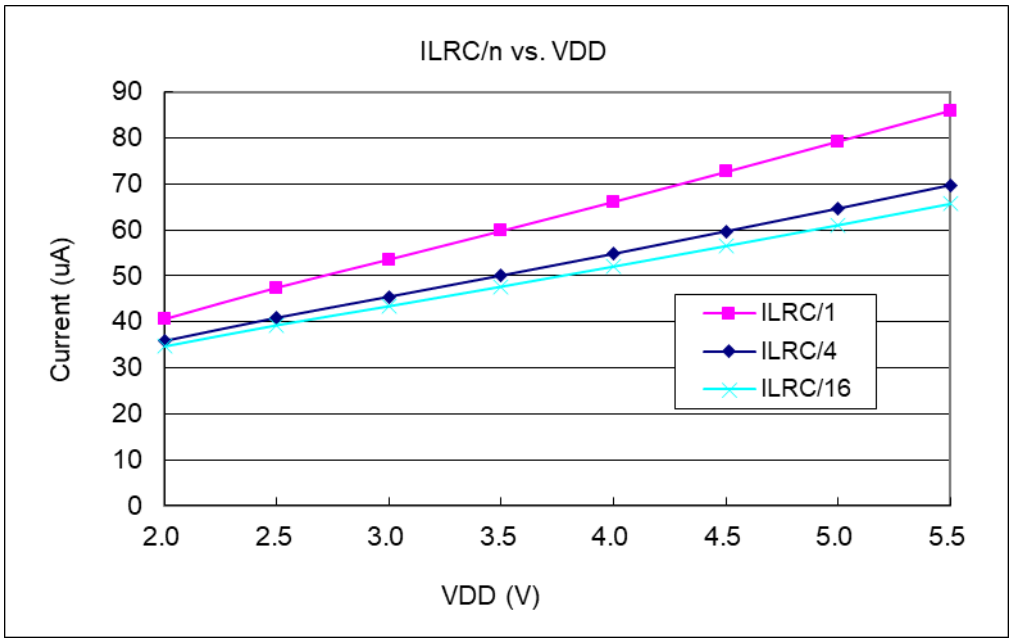


4.7. Typical operating current vs. VDD @ system clock = ILRC/n

Conditions:

ON: ILRC, Bandgap, LVR; **OFF**: IHRC, EOSC, T16, TM2, TM3, ADC modules;

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

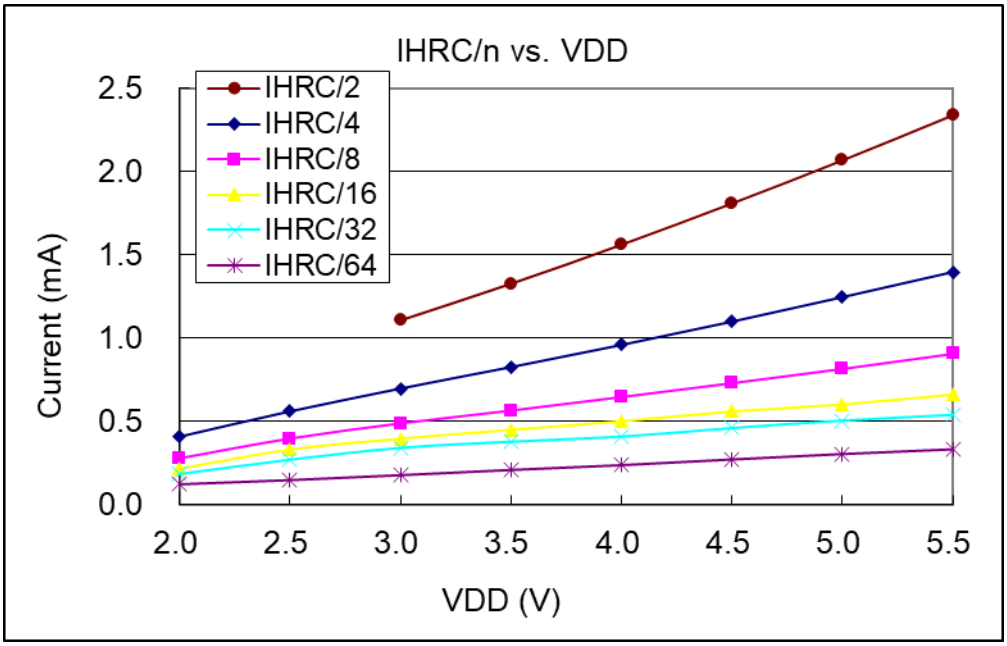


4.8. Typical operating current vs. VDD @ system clock = IHRC/n

Conditions:

ON: IHRC, Bandgap, LVR; **OFF**: ILRC, EOSC, T16, TM2, TM3, ADC modules;

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



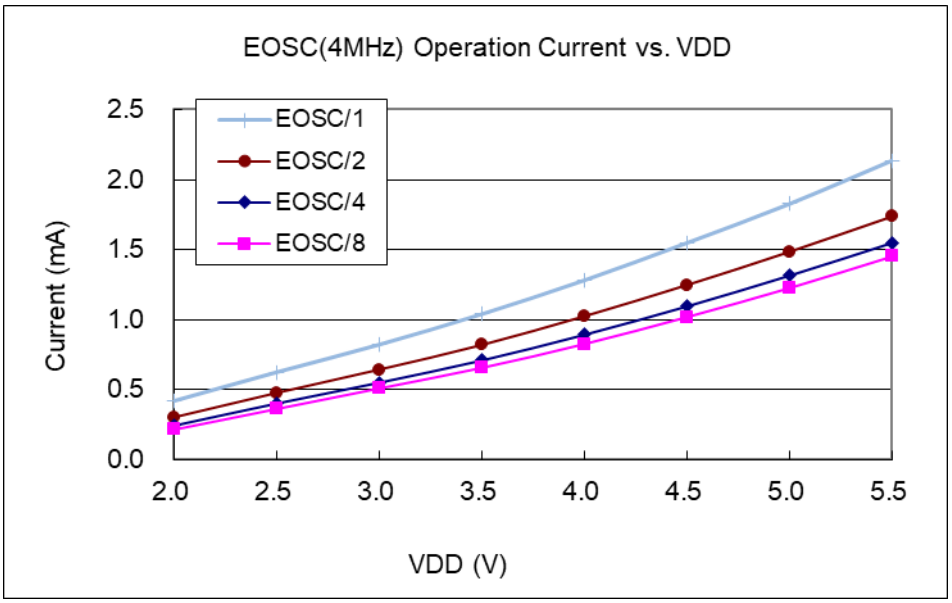
4.9. Typical operating current vs. VDD @ system clock = 4MHz EOSC / n

Conditions:

ON: EOSC [6,5] = [1,1], Bandgap, LVR;

OFF: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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



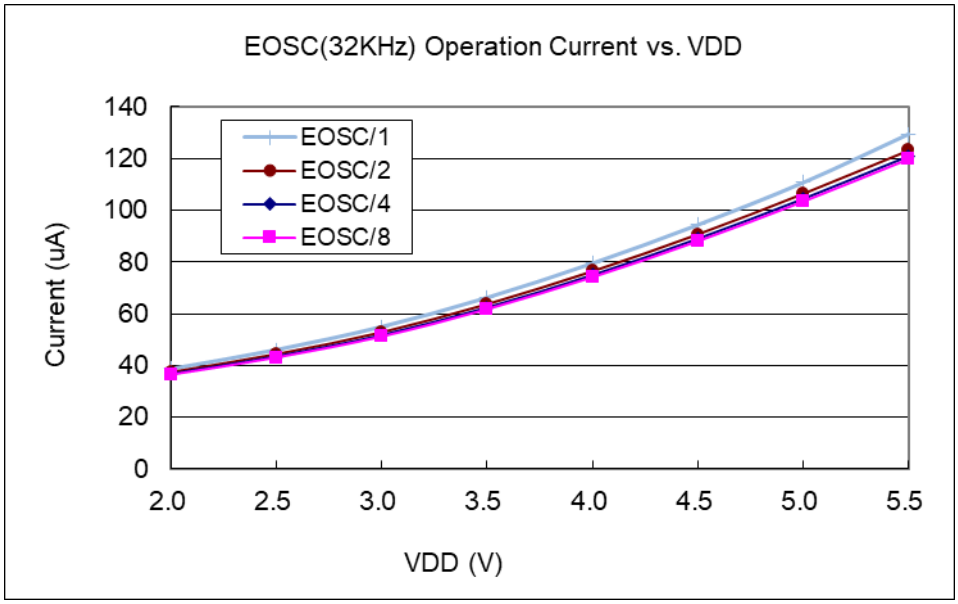
4.10. Typical operating current vs. VDD @ system clock = 32KHz EOSC / n

Conditions:

ON: EOSC[6,5] = [0,1], Bandgap, LVR;

OFF: IHRC, ILRC, T16, TM2, TM3, ADC modules;

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



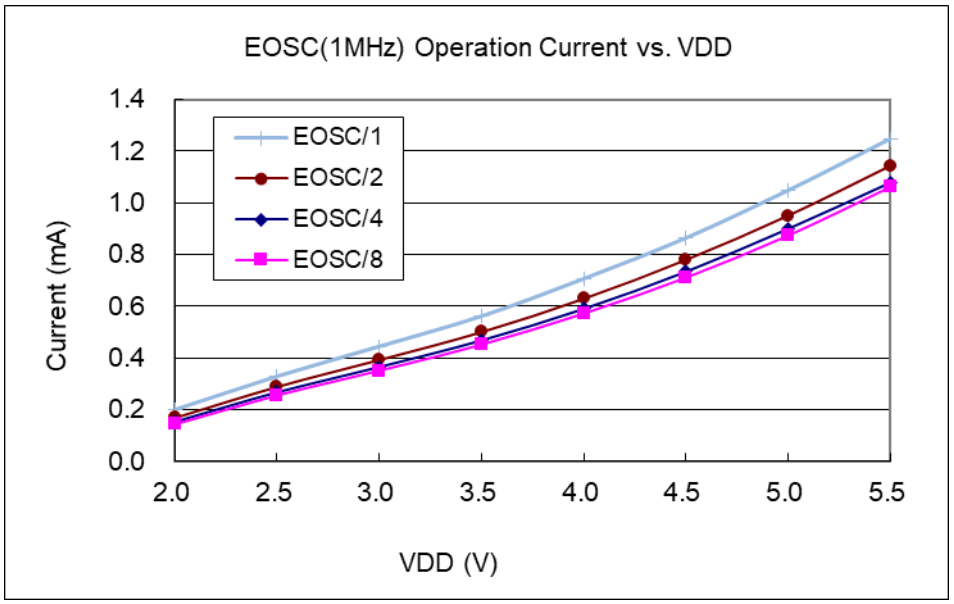
4.11. Typical operating current vs. VDD @ system clock = 1MHz EOSC / n

Conditions:

ON: EOSC[6,5] = [1,0], Bandgap, LVR;

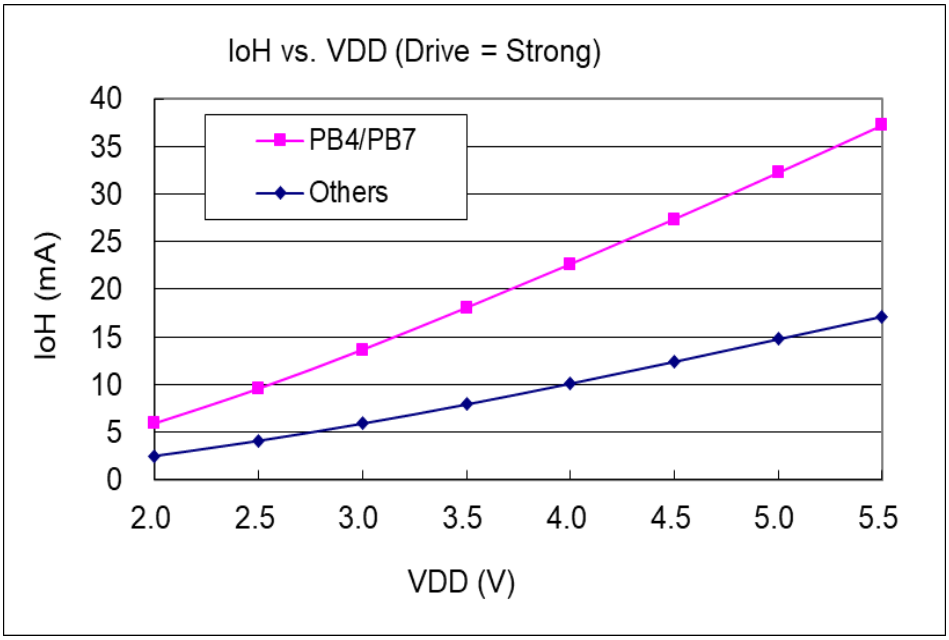
OFF: IHRC, ILRC, T16, TM2, TM3, ADC modules;

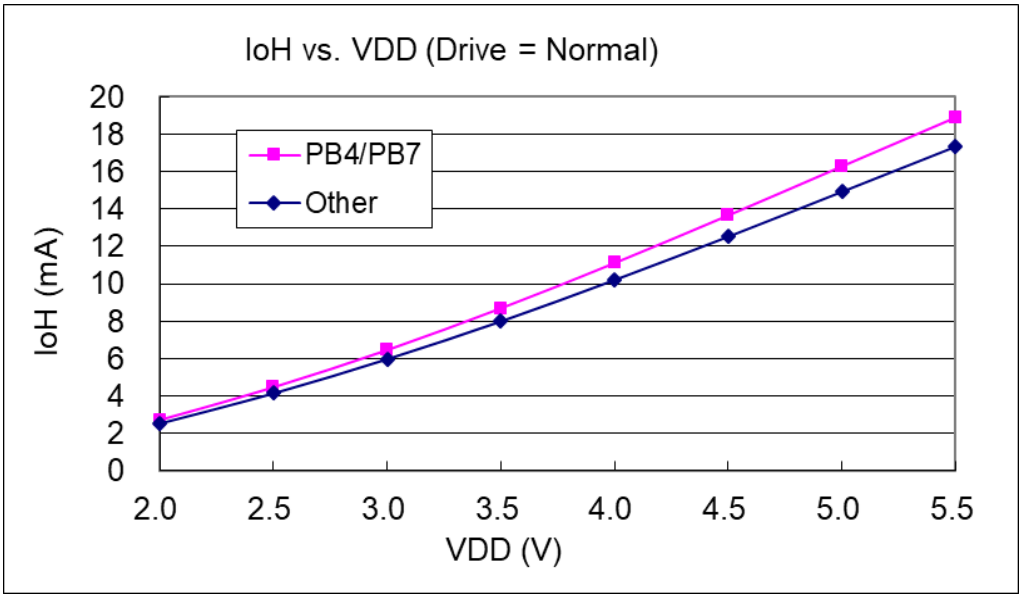
IO: PA0:0.5Hz output toggle and no loading, **others**: input and no floating

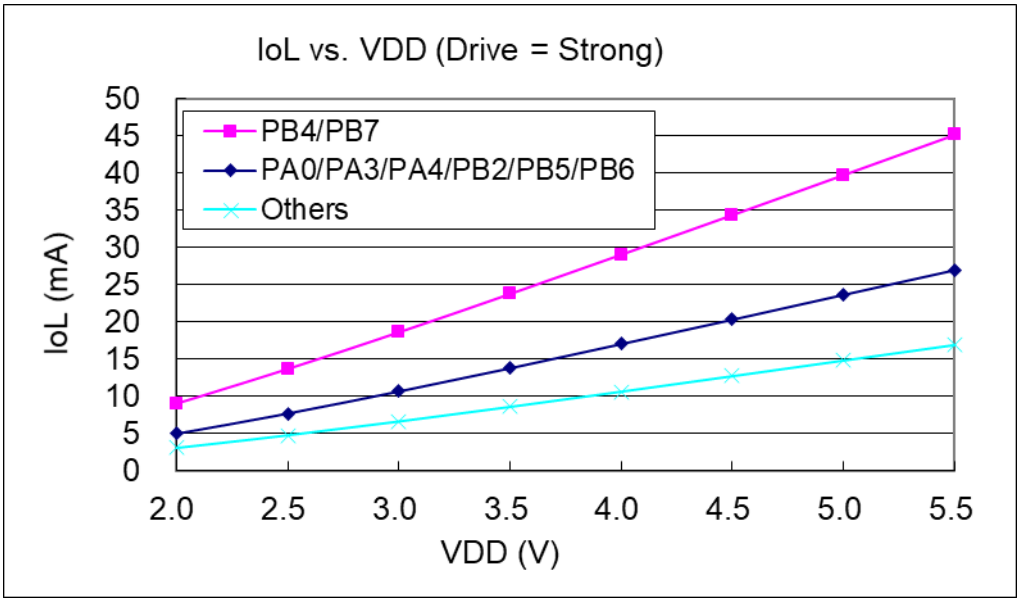


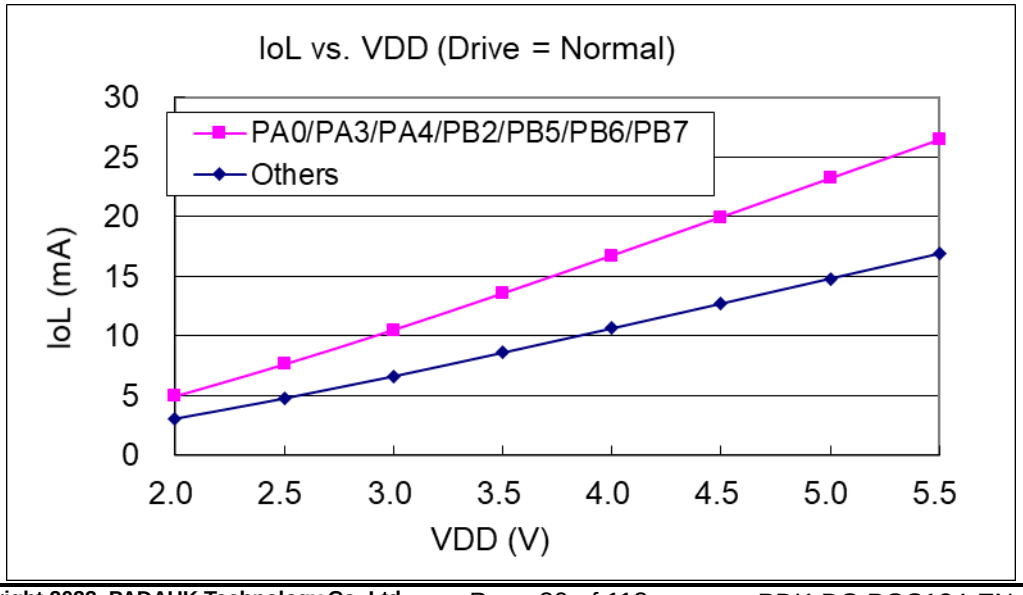
4.12. Typical IO driving current (I_{OH}) and sink current (I_{OL})

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

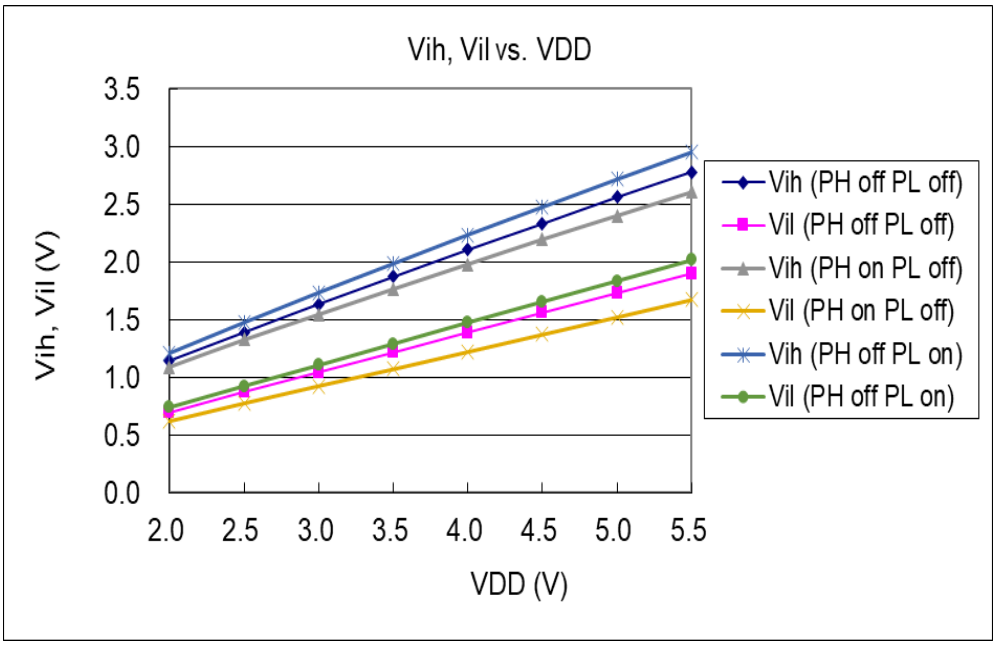




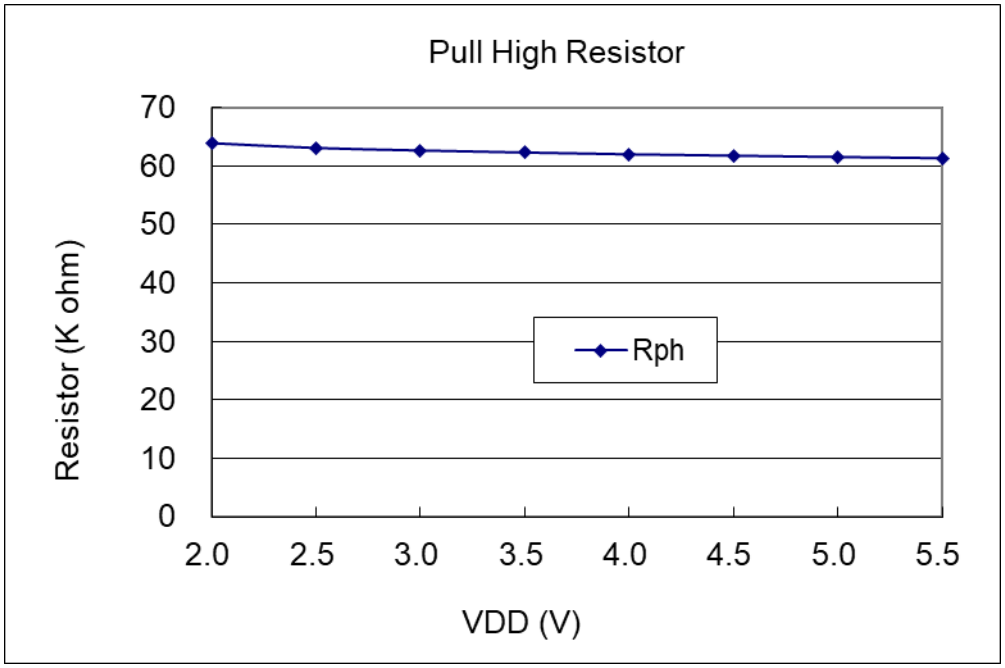




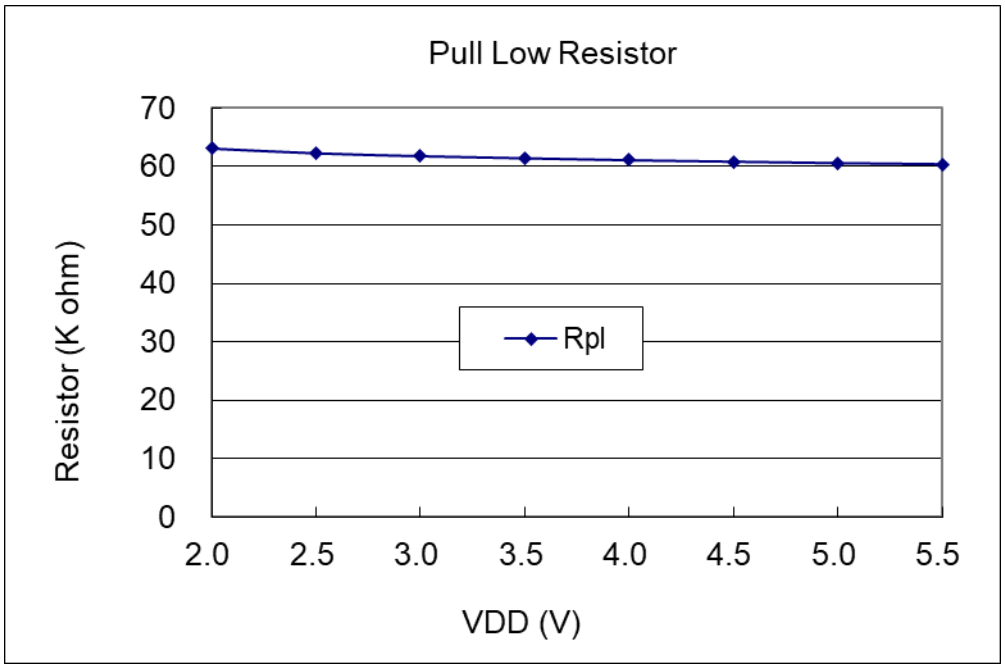
4.13. Typical IO input high/low threshold voltage (V_{IH}/V_{IL})



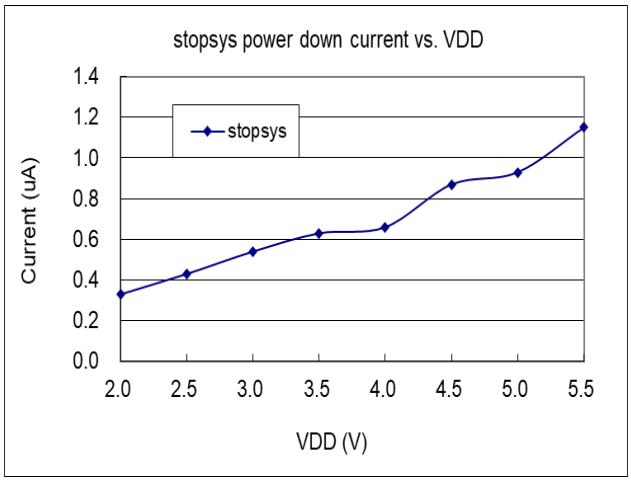
4.14. Typical resistance of IO pull high device

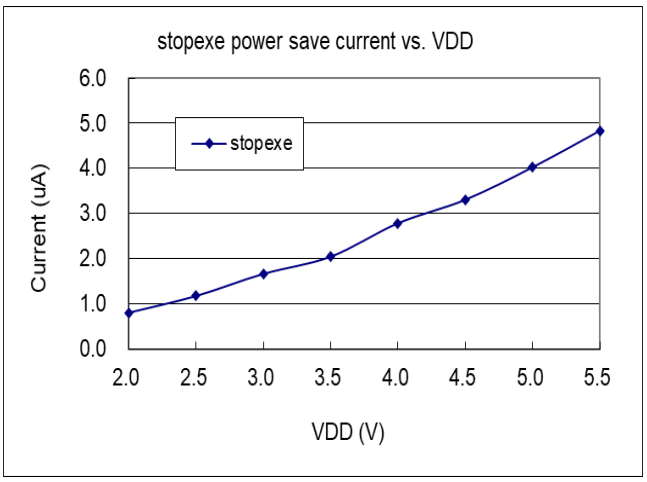


4.15. Typical resistance of IO pull low device



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





5. Functional Description

5.1. Program Memory - MTP

The MTP (Multiple 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 the initial address 0x000 which is GOTO FPPA0 instruction usually. The interrupt entry is 0x10 if used, the last 8 addresses are reserved for system using, like checksum, serial number, etc. The MTP program memory for PGS134 is 4KW that is partitioned as Table 1. The MTP memory from address 0XFF0 to 0xFFF is for system using, address space from 0x001 to 0x00F and from 0x011 to 0XFE7are user program spaces.

Address	Function
0x000	GOTO FPPA0 instruction
0x001	User program
•	•
0x00F	User program
0x010	Interrupt entry address
0x011	User program
•	•
0xFEF	User program
0XFF0	System Using
•	•
0xFFF	System Using

Table 1: Program Memory Organization

5.2. Boot Procedure

POR (Power-On-Reset) is used to reset PGS134 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.

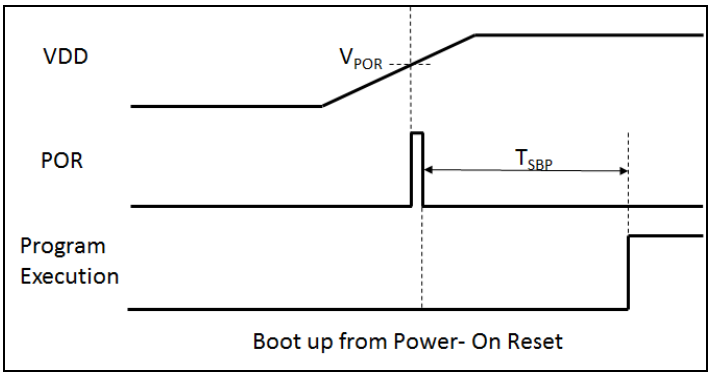
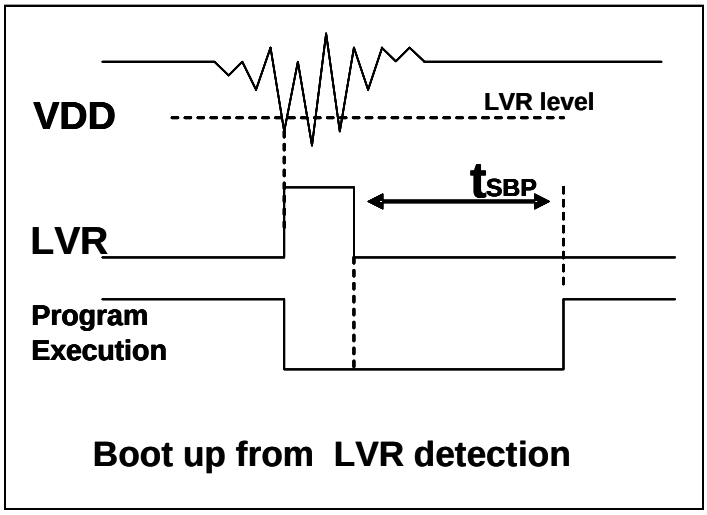
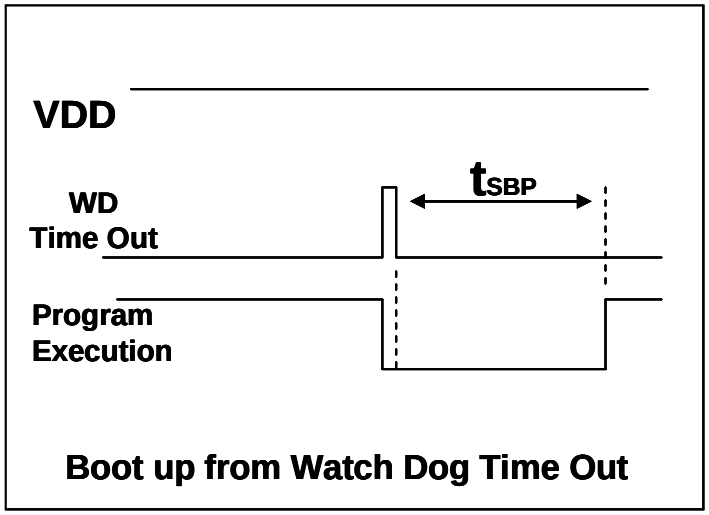


Fig.1: Power-Up Sequence

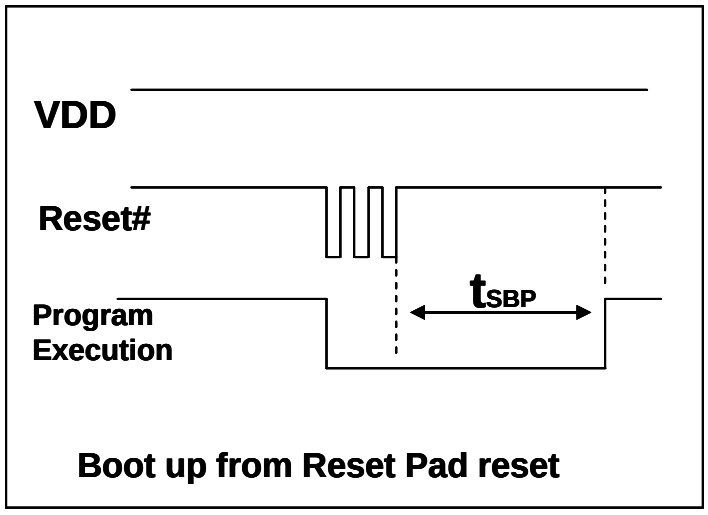
5.2.1. Timing charts for reset conditions



Boot up from LVR detection



Boot up from Watch Dog Time Out



Boot up from Reset Pad reset

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 256 bytes data memory of PGS134 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 read even after the power went off. EEPROM is only accessible by indirect addressing instructions **STEER** and **LDEER**.

Example 1: EEPROM write byte

(1) Define EEPROM addressing variable

```
word    E2Adr_Index =       0;
```

(2) EERMC writes 0x5A enables Byte Program (Support automatic erase)

```
EERMC       =       0x5A;
```

(3) *EERL* assigns the data to be written

```
EERL  =       10;
```

(4) Use *STEEL* to write data to EEPROM (address: E2Adr_Index (0)).

```
STEER       E2Adr_Index;
```

(5) Wait for write finished

```
while(EERMC.Busy)       NOP;
```

Note: The value of the addressing variable E2Adr_Index is modified by itself to achieve writing data to different Byte addresses of EEPROM

Example 2： EEPROM read byte

(1) Define EEPROM addressing variable

```
word    E2Adr_Index =       0;
```

(2) *EERMC* writes 0x5A enables Byte Read

```
EERMC       =       0x5A;
```

(3) Use *LDEER* to read data from EEPROM (address: E2Adr_Index (0)).

```
LDEER       E2Adr_Index;
```

(4) Read data from *eerl* after two instruction cycle

```
NOP;  NOP; variable = eerl;
```

Note: The *eerl* register will hold this data until the next write or erase

Example 3：EEPROM full erase

(1) Define EEPROM addressing variable

word E2Adr_Index = 0;

(2) *EERMC* writes 0xA5 enables Chip Erase

EERMC = 0xA5;

(3) Use *STEER* to erase all data in EEPROM

STEER E2Adr_Index;

(4) Wait for erase finished

While(EERMC.Busy) NOP;

Note: 1. Chip Erase can erase all data from EEPROM and cannot be paged.

2. After data erasure, all data in the EEPROM is 0xFF.

5.5. Oscillator and clock

There are three oscillator circuits provided by PGS134: 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 of ILRC 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, PGS134 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=2.5 ~ 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 PGS134 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 in input mode

(7) .ADJUST_IC DISABLE

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

- IHRC is not calibrated and IHRC module is disabled
- System CLK = ILRC or IHRC/64
- 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 or resonator is required between X1 and X2. Fig.2 shows the hardware connection under this application; the range of operating frequency of crystal oscillator can be from 32 KHz to 4MHz, depending on the crystal placed on; higher frequency oscillator than 4MHz is NOT supported.

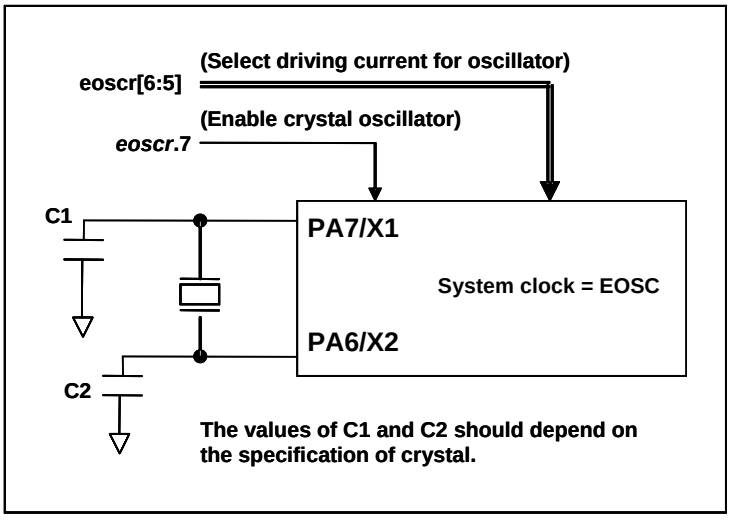


Fig.2: Connection of crystal oscillator

Besides crystal, external capacitor and options of PGS134 should be fine tuned in *eoscr* (0x0a) register to have good sinusoidal waveform. The *eoscr*.7 is used to enable crystal oscillator module, *eoscr*.6 and *eoscr*.5 are used to set the different driving current to meet the requirement of different frequency of crystal oscillator:

- *eoscr*.[6:5]=01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator
- *eoscr*.[6:5]=10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator
- *eoscr*.[6:5]=11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator

Table 4 shows the recommended values of C1 and C2 for different crystal oscillator; the measured start-up time under its corresponding conditions is also shown. Since the crystal or resonator had its own characteristic, the capacitors and start-up time may be slightly different for different type of crystal or resonator, please refer to its specification for proper values of C1 and C2.

Frequency	C1	C2	Measured Start-up time	Conditions
4MHz	4.7pF	4.7pF	6ms	(*eoscr*[6:5]=11, *misc*.6=0)
1MHz	10pF	10pF	11ms	(*eoscr*[6:5]=10, *misc*.6=0)
32KHz	22pF	22pF	450ms	(*eoscr*[6:5]=01, *misc*.6=0)

Table 4: Recommend values of C1 and C2 for crystal and resonator oscillators

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” external 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, 4MHz;          // EOSCR = 0b110_00000;

$   T16M  EOSC, /1, BIT13;         // T16 receive 2^14=16384 clocks of crystal EOSC,
                                   // Intrq.T16 =>1, crystal EOSC Is stable
    WORD    count =    0;
    stt16  count;
    Intrq.T16   =    0;
    do
    {   nop; }while(!Intrq.T16);   // count from 0x0000 to 0x2000, then set 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 wakeup 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 PGS134 is shown as Fig.3.

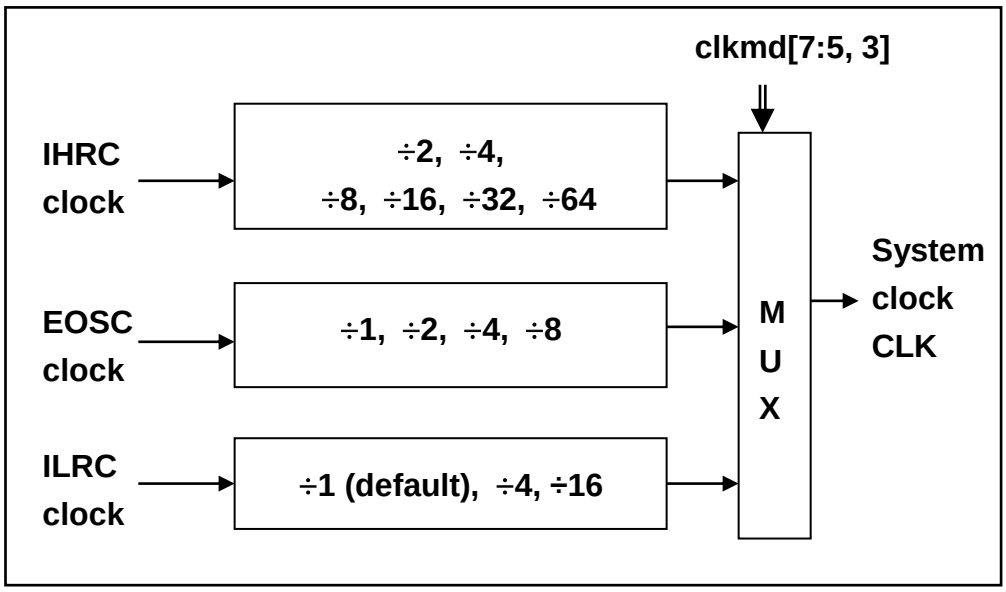


Fig.3: Options of System Clock

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, and the lowest LVR levels can be chosen for different operating frequencies. 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 PGS134 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 PGS134; Fig.4 shows its hardware diagram. It can compare signals between two pins or with either internal reference voltage $V_{internal\ R}$ 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, PB6, PB7 or $V_{internal\ R}$ selected by bit [3:1] of gpcc register, and the plus input of comparator can be PA4 or $V_{internal\ R}$ 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.

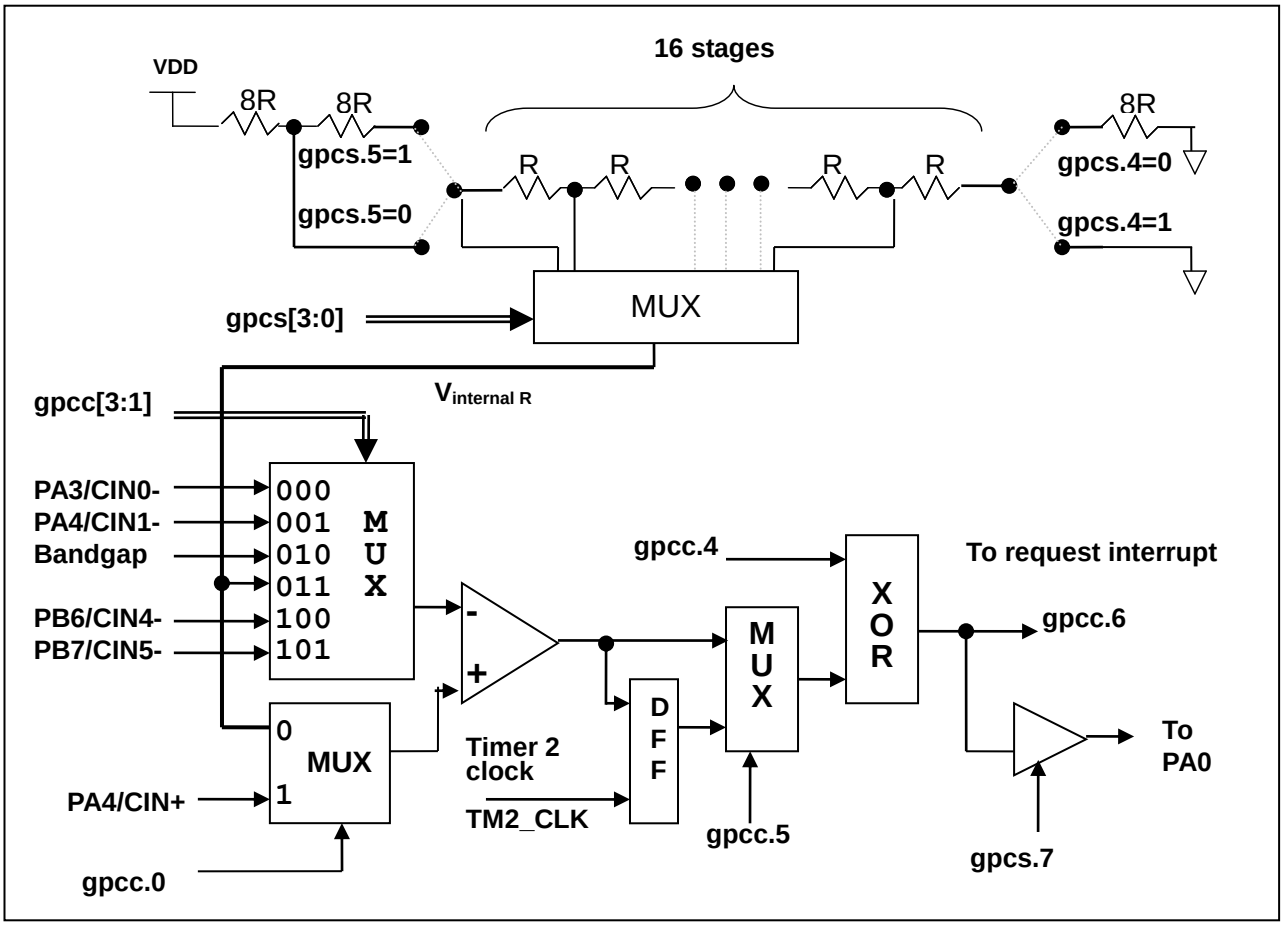


Fig.4: Hardware diagram of comparator

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}.

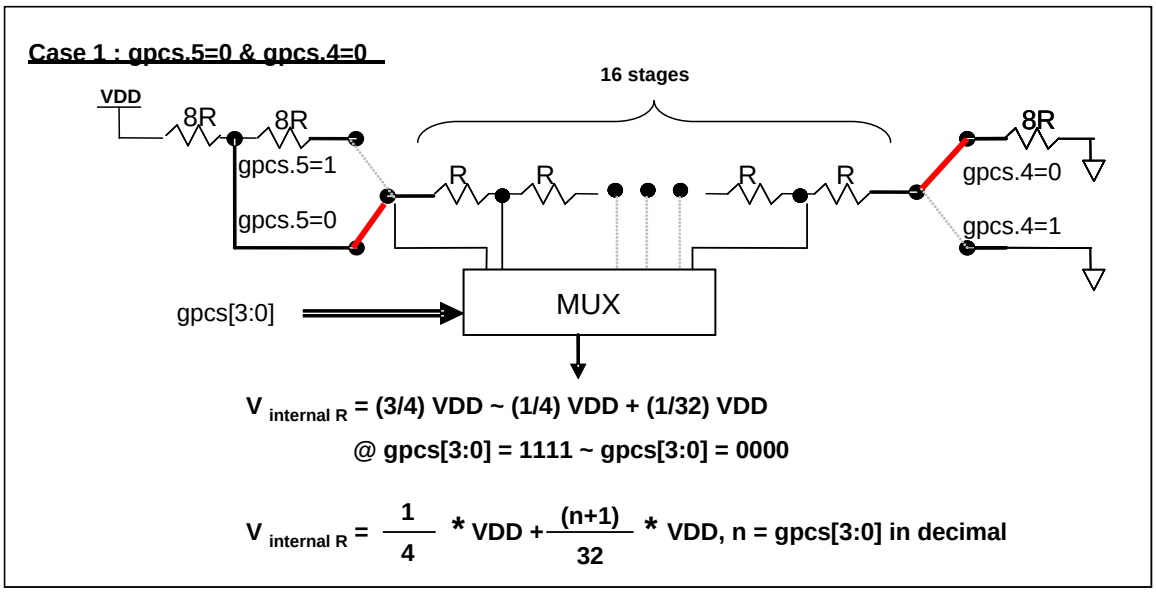


Fig.5: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=0

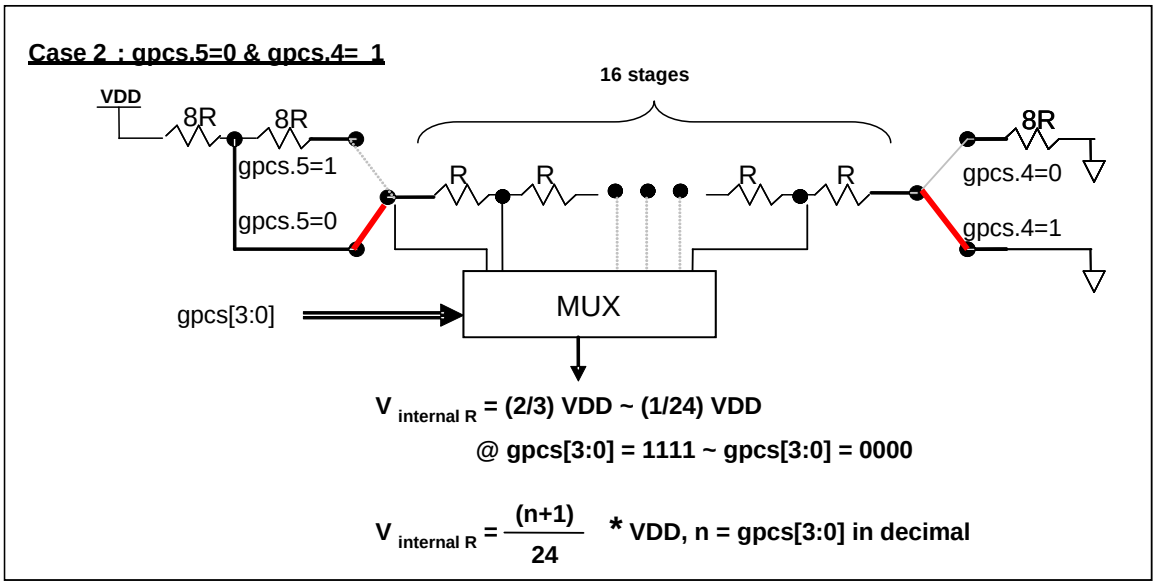


Fig.6: $V_{internal\ R}$ hardware connection if gpcs.5=0 and gpcs.4=1

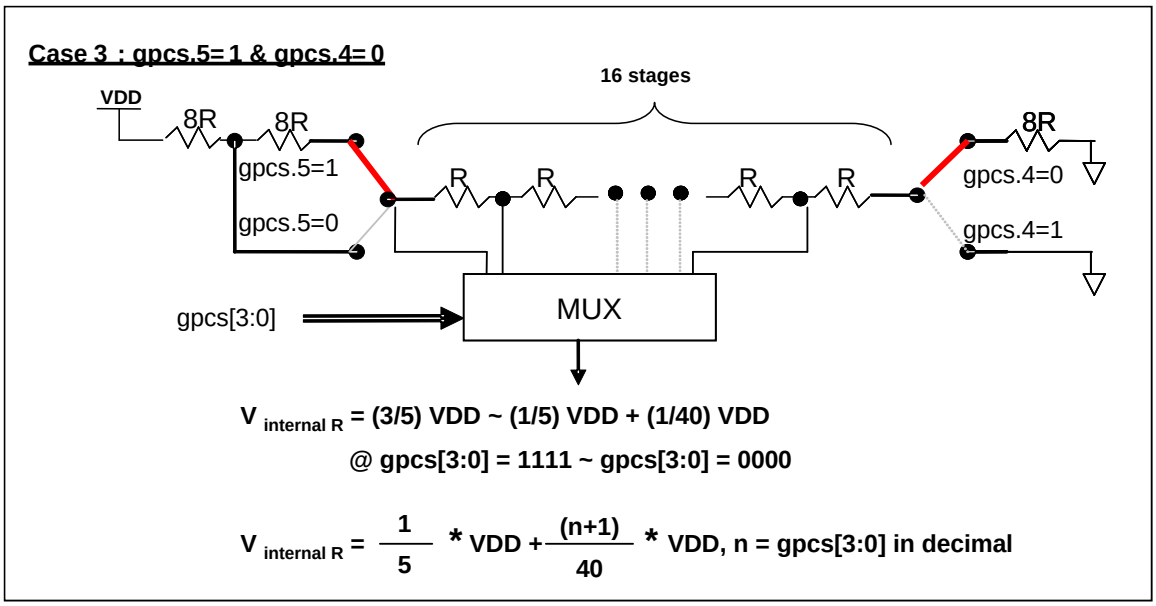


Fig.7: $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=0

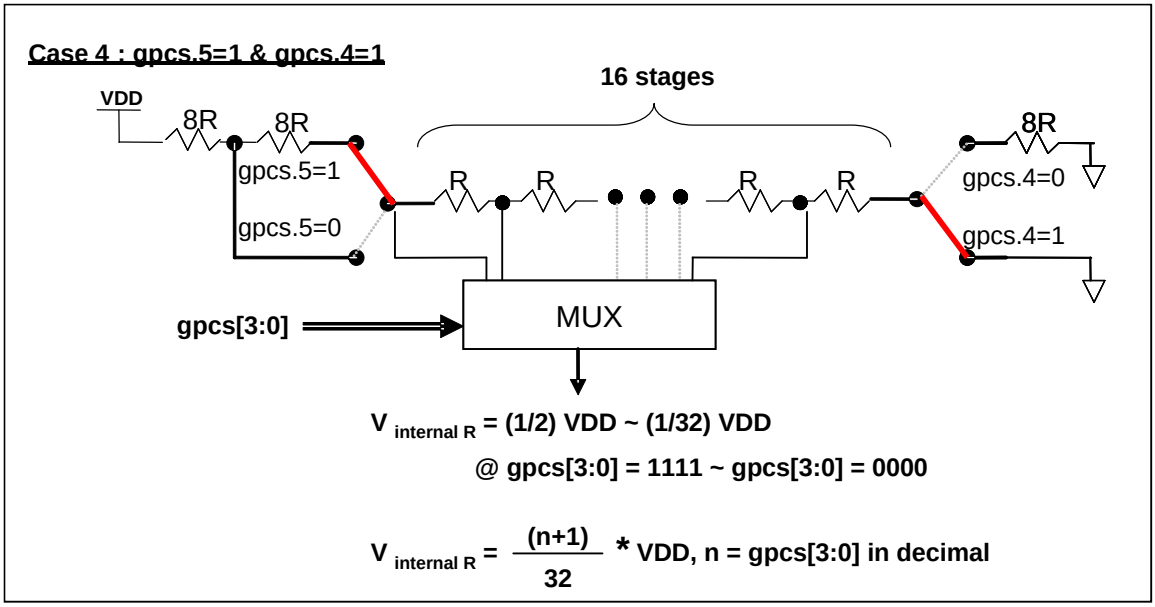


Fig.8: $V_{internal\ R}$ hardware connection if gpcs.5=1 and gpcs.4=1

5.6.2. Using the comparator

Case 1:

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

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

or

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

Case 2:

Choosing $V_{internal\ R}$ 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. $V_{internal\ R}$ is configured as the above Figure "gpcs[5:4] = 2b'10" and gpcs [3:0] = 4b'1101 (n=13) to have $V_{internal\ R}$ = (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,V_internal R = V_DD*(22/40)
gpcc    = 0b1_0_0_1_011_1;      // Inverse output, - input: V_internal R, + input: PA4
padier  = 0bxxx_0_xxxx;         // disable PA4 digital input to prevent leakage current
```

or

```
$ GPCS      V_DD*22/40;
$ GPCC      Enable,   Inverse, N_R, P_PA4;      // - input: N_R(V_internal 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 VDD 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. VDD/2 LCD Bias Voltage Generator

This function can be enabled by misc.4 and code option LCD2. There are two sets of pins can be selected as the LCD COM ports, four pins in each set. By selecting PB0_A034 for LCD2, PB0, PA0, PA4 and PA3 are defined to output VDD/2 voltage during input mode, and be used as COM function for LCD applications. By selecting PB1256 of LCD2, PB1, PB2, PB5 and PB6 are defined as COM ports.

If user wants to output VDD, VDD/2, GND three levels voltage, enabling VDD/2 bias voltage (by set misc.4=1), then set to output-high for VDD, with input mode for VDD/2, and output-low for GND correspondingly, Fig.9 shows how to use this function.

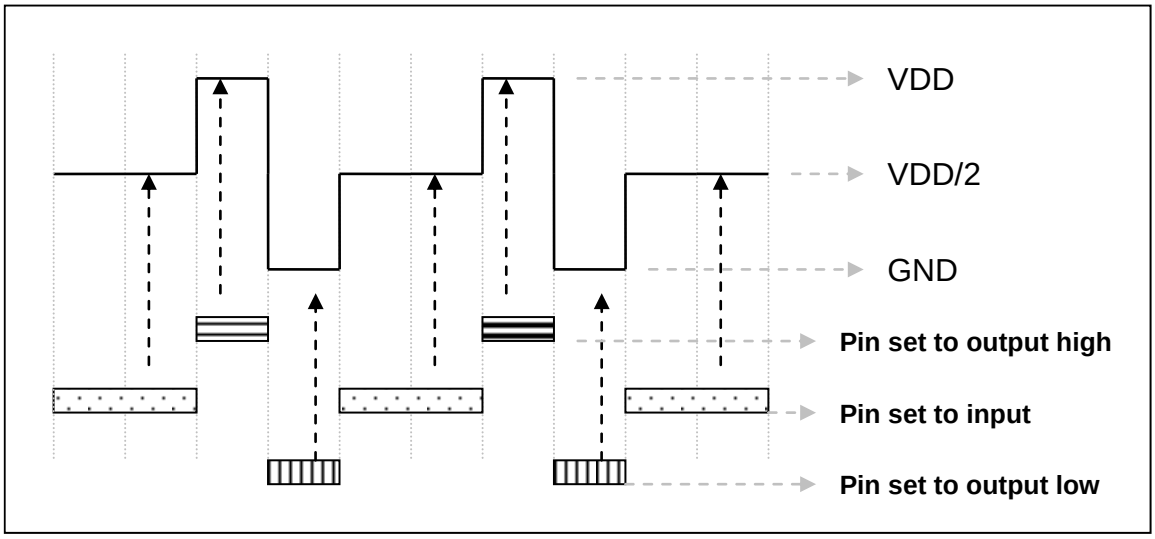


Fig. 9: Using VDD/2 LCD bias voltage generator

5.8. 16-bit Timer (Timer16)

A 16-bit hardware timer (Timer16) is implemented in the PGS134, 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.10. 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 (IO address 0x0C).

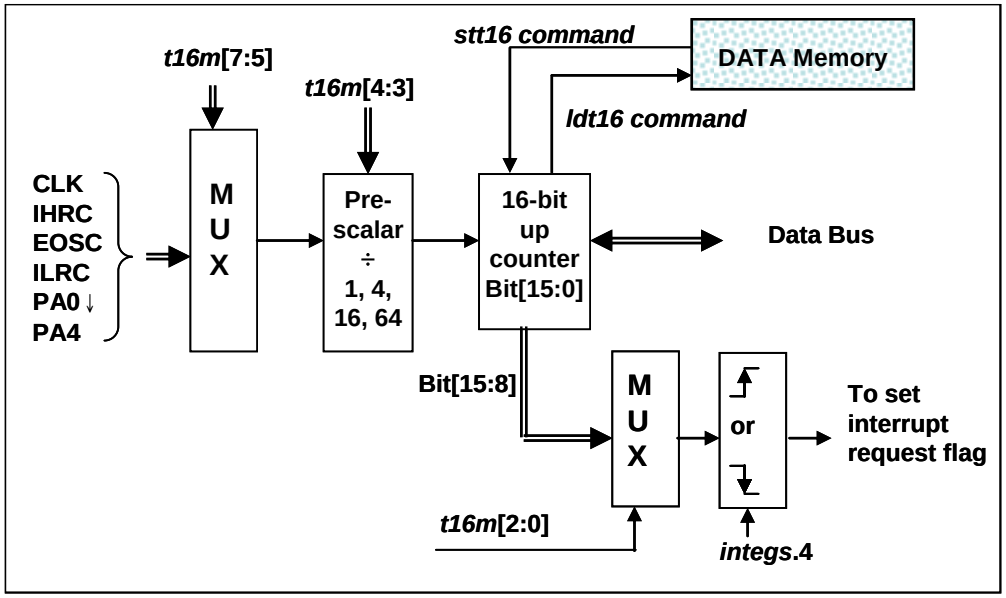


Fig.10: Hardware diagram of Timer16

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 524 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 n^{th} bit selected to request interrupt service, for example: n=10 if bit 10 is selected.

5.9. 8-bit Timer (Timer2/Timer3) with PWM generation

Two 8-bit hardware timers (Timer2 and Timer3) with PWM generation are implemented in the PGS134. The following descriptions thereinafter are for Timer2 only. It is because Timer3 have same structure with Timer2. Please refer to Fig.11 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, PB0, PA4 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 PB2, PA3 or PB4, 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, 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 to 8-bit PWM resolution, Fig.12 shows the timing diagram of Timer2 for both period mode and PWM mode.

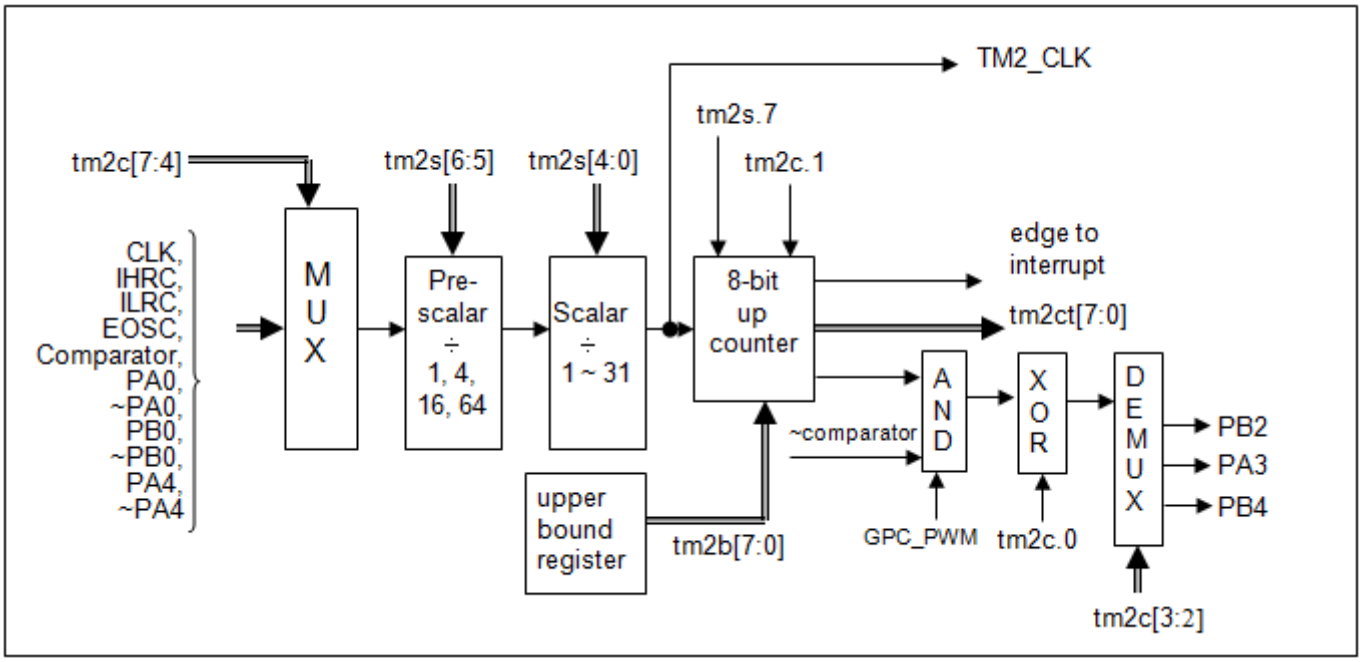


Fig.11: Timer2 hardware diagram

The output of Timer3 can be sent to pin PB5, PB6 or PB7.

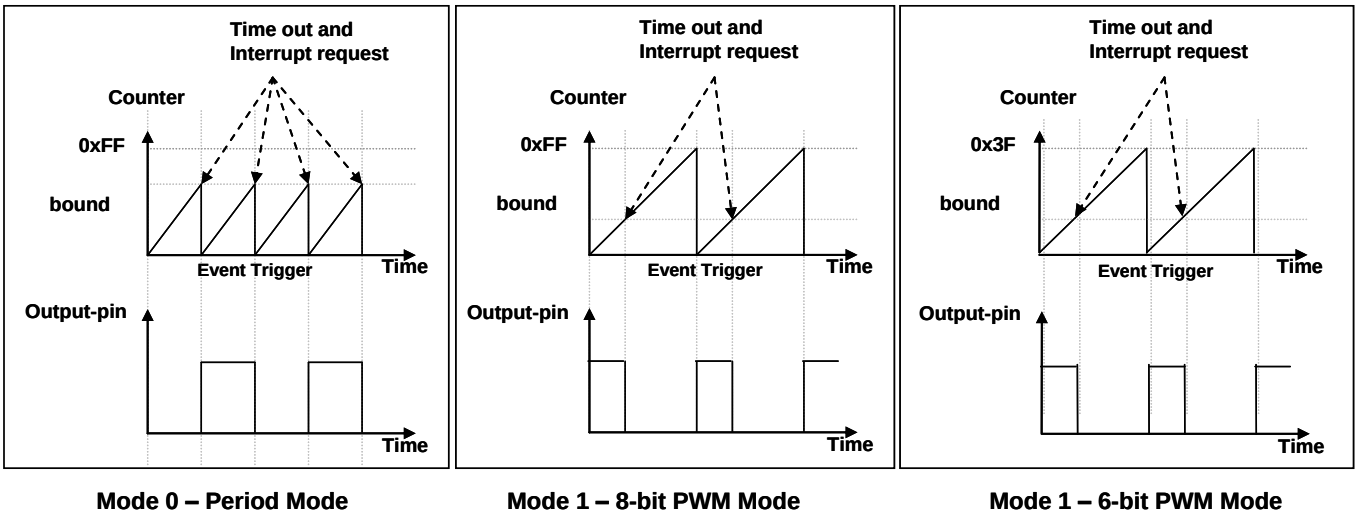


Fig.12: Timing diagram of Timer2 in period mode and PWM mode (tm2c.1=1)

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. 13.

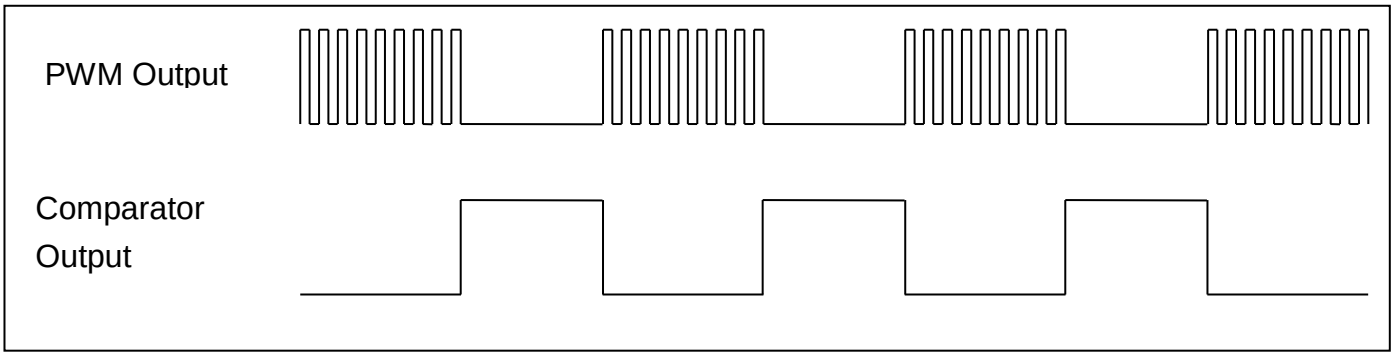


Fig.13：Comparator controls the output of PWM waveform

5.9.1. 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 = 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 = 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 = 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 = 0x0;
    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;
    }
}
```

5.9.2. 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)

Example 1:

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%

Example 2:

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%

Example 3:

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%

Example 4:

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
    tm2ct = 0x0;
    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;
    }
}
```

5.9.3. 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:

Frequency of Output = Y ÷ [64 × S1 × (S2+1)]

Duty of Output = [(K＋1) ÷ 64] × 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 ***TMx_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%

5.10.11-bit PWM Generator

Three 11-bit hardware PWM generators (PWMG0, PWMG1 & PWMG2) are implemented in the PGS134. The following descriptions there in after are for PWMG0 only. It is because PWMG1 & PWMG2 have the same structures and functions with PWMG0.

Their individual outputs are listed as below:

- PWMG0 – PA0, PB4, PB5, PC2
- PWMG1 – PA4, PB6, PB7, PC3
- PWMG2 – PA3, PB2, PB3, PC0

5.10.1. 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 ($f_{PWM} = 1/T_{Period}$).

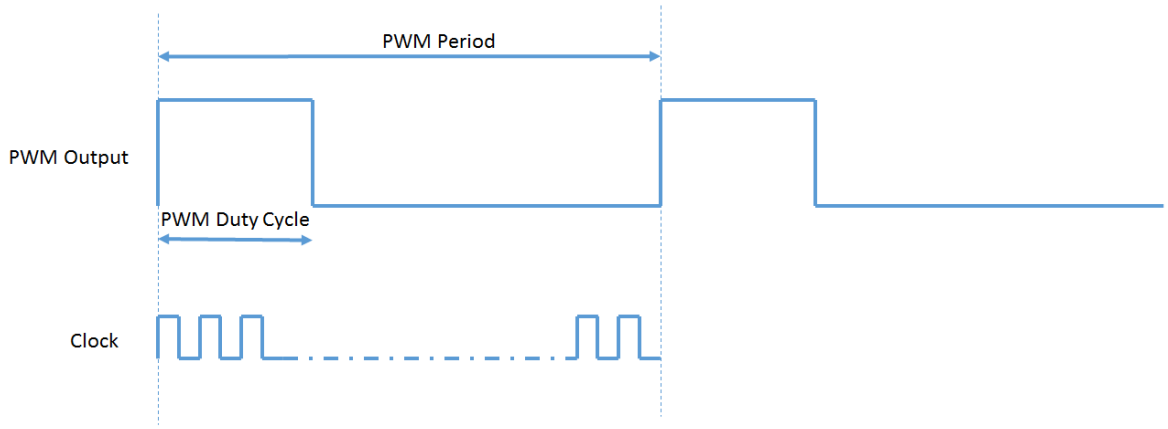


Fig.14: PWM Output Waveform

5.10.2. Hardware and Timing Diagram

Three 11-bit hardware PWM generators are built inside the PGS134; Fig.14 shows the hardware diagram PWMG0 as an example. The clock source can be IHRC or system clock and output pin can be selected via ***pwmc*** register selection. The period of PWM waveform is defined in the PWM upper bond high and low registers, the duty cycle of PWM waveform is defined in the PWM duty high and low registers. Comparator output can control the PWM waveform by selecting GPC_PWM code option.

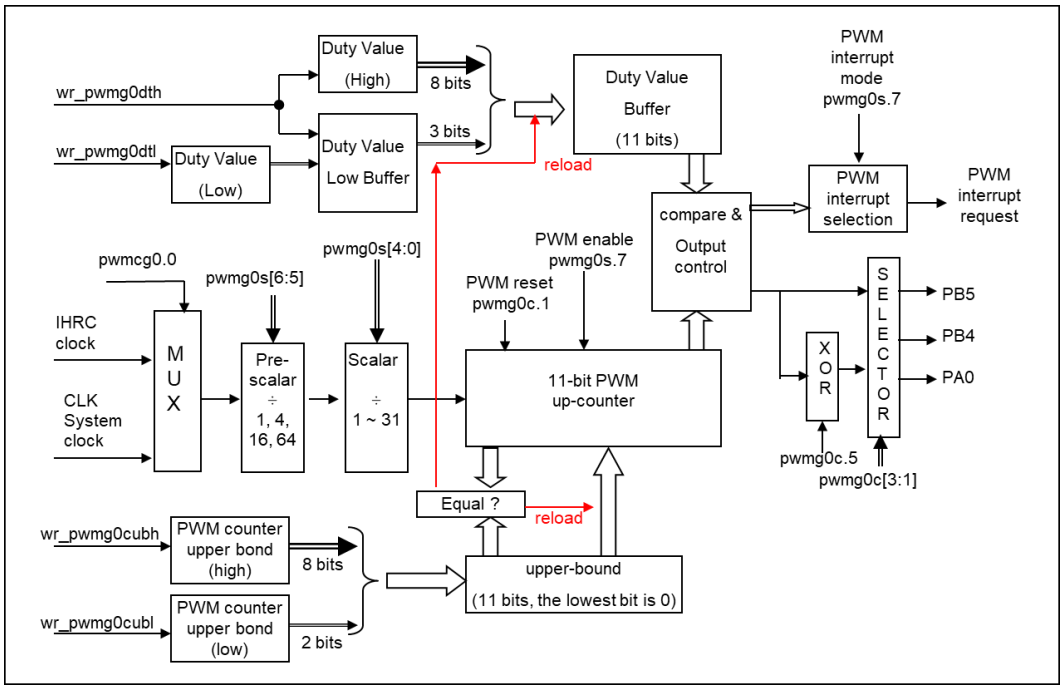


Fig.15: Hardware Diagram of 11-bit PWM Generator

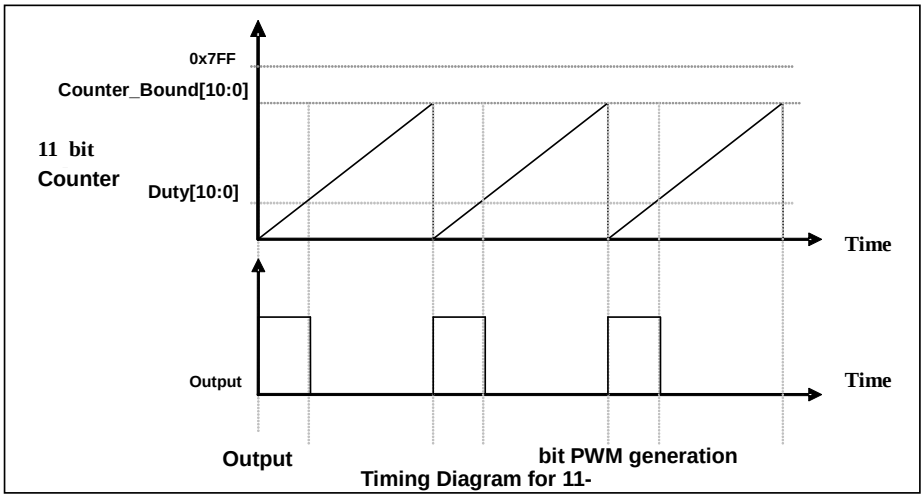


Fig.16: Output Timing Diagram of 11-bit PWM Generator

5.10.3. Equations for 11-bit PWM Generator

PWM Frequency F_{PWM} = $F_{clock\ source}$ ÷ [P × (K + 1) × (CB10_1 + 1)]

PWM Duty(in time) = (1 / F_{PWM}) × (DB10_1 + DB0 × 0.5 + 0.5) ÷ (CB10_1 + 1)

PWM Duty(in percentage) = (DB10_1 + DB0 × 0.5 + 0.5) ÷ (CB10_1 + 1) × 100%

Where, **P =** *PWMGxS* [6:5] : pre-scalar (**P** = 1, 4, 16, 64)

K = *PWMGxS* [4:0] : scalar in decimal (**K** =0 ~ 31)

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

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

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

5.11. 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, PGS134 will be reset to restart the program execution. The relative timing diagram of watchdog timer is shown as Fig.16.

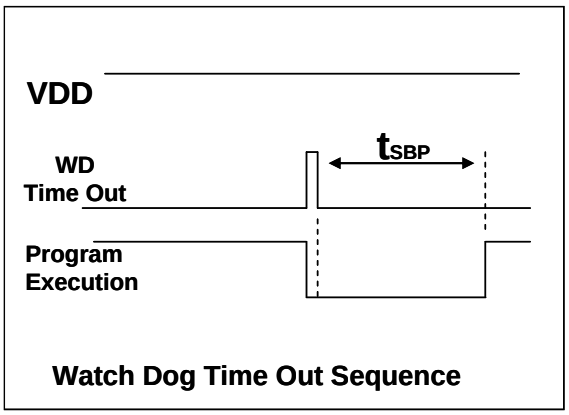


Fig.17: Sequence of Watch Dog Time Out

5.12. Interrupt

There are eight interrupt lines for PGS134:

- External interrupt PA0/PB5/PA2/PA7
- External interrupt PB0/PA4/PB6/PA3
- ADC interrupt
- Timer16 interrupt
- GPC/ PWMG1 interrupt
- PWMG0 or EEW_Done interrupt
- Timer2 interrupt
- Timer3/ PWMG2 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.17. 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.

Note: the external interrupt source can be switched through Interrupt Src0 or Interrupt Src1 in the Code Option.

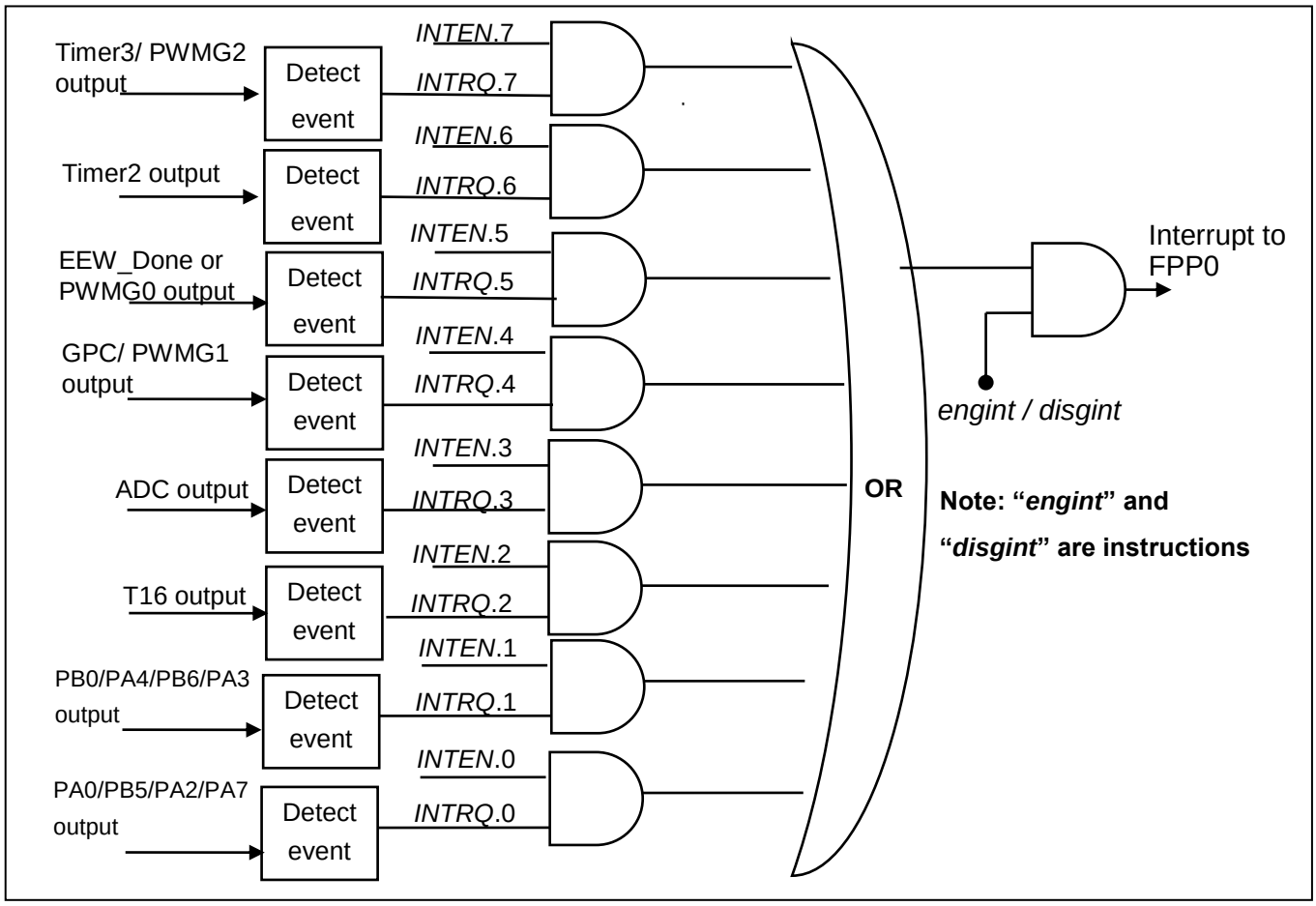


Fig.18: Hardware diagram of interrupt controller

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. For interrupt operation, the following sample program shows how to handle the interrupt, noticing that it needs four bytes stack memory to handle 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
}
```

5.13. 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 5 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 5: Differences in oscillator modules between STOPSYS and STOPEXE

5.13.1. 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 wakeup by comparator when setting GPCC.7=1 and GPCS.6=1 to enable the comparator wakeup 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, TM3, PWMG0, PWMG1, PWMG2.)
- Wake-up sources:
  a. IO toggle wake-up: IO toggling in digital input mode (*PxC* bit is 1 and *PxDIER* 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. Please note: the internal 1.20V bandgap reference voltage is not suitable for the comparator wake-up function.

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

```
$ T16M      ILRC, /1, BIT8          // Timer16 setting
$ INTEGS    BIT_R, xxx;             // BITx 0 to 1 will trigger (default)

...
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.

5.13.2. 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 PGS134 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 (PxDIER 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
```

5.13.3. Wake-up

After entering the Power-Down or Power-Save modes, the PGS134 can be resumed to normal operation by toggling IO pins, Timer interrupt is available for Power-Save mode ONLY. Table 6 shows the differences in wake-up sources between STOPSYS and STOPEXE.

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

Table 6: Differences in wake-up sources between Power-Save mode and Power-Down mode

When using the IO pins to wake-up the PGS134, 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

Please notice that when Fast boot-up is selected, 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.

5.14. IO Pins

All the pins can be independently set into two states output or input by configuring the data registers (***pa***, ***pb, pc***), control registers (***pac***, ***pbc, pcc***) and pull-high registers (***paph***, ***pbph, pcph***) or pull-low registers (***papl, pbpl, pcpl***). 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.19.

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

Table 7: PA0 Configuration Table

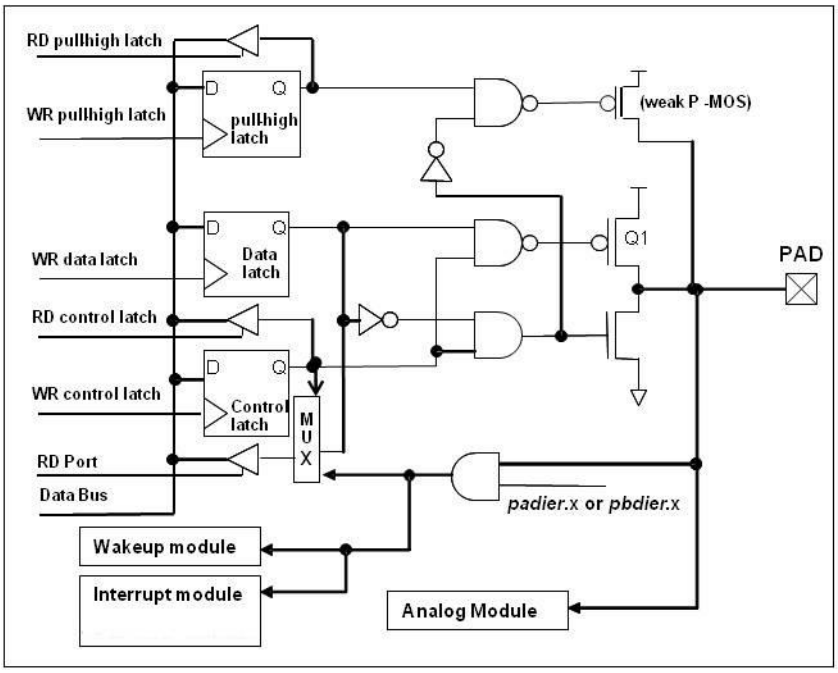


Fig. 19: Hardware diagram of IO buffer

PB4 and PB7 can adjust their drive and sink current by code option ***PB4_PB7_Drive.***

Other than PA5, all the IO pins have the same structure; PA5 can be open-drain ONLY when setting to output mode (without Q1). The corresponding bits in registers ***padier*** / ***pbdier*** / ***pcdier*** should be set to low to prevent leakage current for those pins are selected to be analog function. When PGS134 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 ***pxdier*** to high***.*** The same reason, ***padier***.0 should be set high when PA0 is used as external interrupt pin, and so for other external interrupt pins: PB0, PA4, PB5, PB6, PA2, PA3 and PA7.

5.15. Reset and LVR

5.15.1. Reset

There are many causes to reset the PGS134, once reset is asserted, most of all the registers in PGS134 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.

5.15.2. LVR reset

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

5.16. Analog-to-Digital Conversion (ADC) module

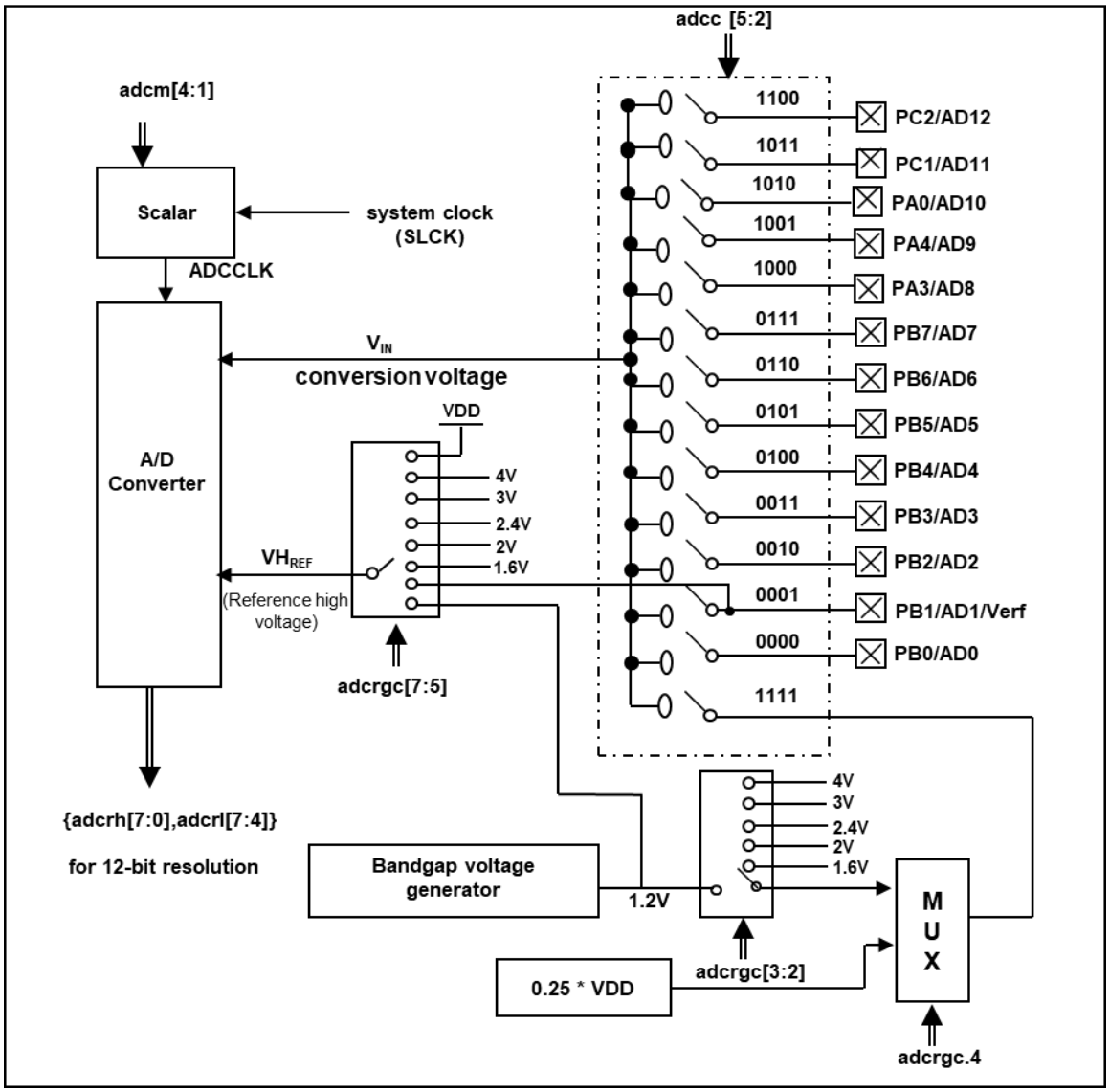


Fig.20: ADC Block Diagram

There are seven registers when using the ADC module, which are:

- ADC Control Register (***adcc***)
- ADC Regulator Control Register (***adcrgc***)
- ADC Mode Register (***adcm***)
- ADC Result High/Low Register (***adcrh***, ***adcrl***)
- Port A/B/C Digital Input Enable Register (***padier***, ***pbdier***, ***pcdier***)

The following steps are required to do the ADC conversion procedure:

(1) Configure the voltage reference high by ***adcrgc*** register

(2) Configure the AD conversion clock by ***adcm*** register

(3) Configure the pin as analog input by ***padier***, ***pbdier*** register

(4) Select the ADC input channel by ***adcc*** register

(5) Enable the ADC module by ***adcc*** register

(6) Delay a certain amount of time after enabling the ADC module

Condition1: Using bandgap 1.2V/1.6V/2.4V or 2V/3V/4V related circuit, either it is used as an internal reference high voltage or an AD Input channel, it must delay more than 1ms. Or it must delay 200 AD clocks when the time of 200 AD clocks is larger than 1ms. When internal BG/2V/3V/4V is enabled as reference high voltage, IHRC must be opened.

Condition 2: Without using any bandgap 1.2V/1.6V/2.4V or 2V/3V/4V related circuit, it needs to delay 200 AD clocks only.

Note: The 200 AD clocks in the above two conditions, which refer to the ADC conversion clock rather than the system clock (SYSCLK) after configured by the ADCM register.

(7) Execute the AD conversion and check if ADC data is ready
Set '1' to ***addc.6*** to start the conversion and check whether ***addc.6*** is '1'

(8) Read the ADC result registers:
First read the ***adcrh*** register and then read the ***adcrl*** register.

If user power down the ADC and enable the ADC again, or switch ADC reference voltage and input channel, be sure to go to step 6 to confirm the ADC becomes ready before the conversion.

5.16.1. The input requirement for AD conversion

For the AD conversion to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the voltage reference high level and discharge to the voltage reference low level. The analog input model is shown as Fig.20, the signal driving source impedance (Rs) and the internal sampling switch impedance (Rss) will affect the required time to charge the capacitor C_{HOLD} directly. The internal sampling switch impedance may vary with ADC supply voltage; the signal driving source impedance will affect accuracy of analog input signal. User must ensure the measured signal is stable before sampling; therefore, the maximum signal driving source impedance is highly dependent on the frequency of signal to be measured. The recommended maximum impedance for analog driving source is about 10KΩ under 500KHz input frequency.

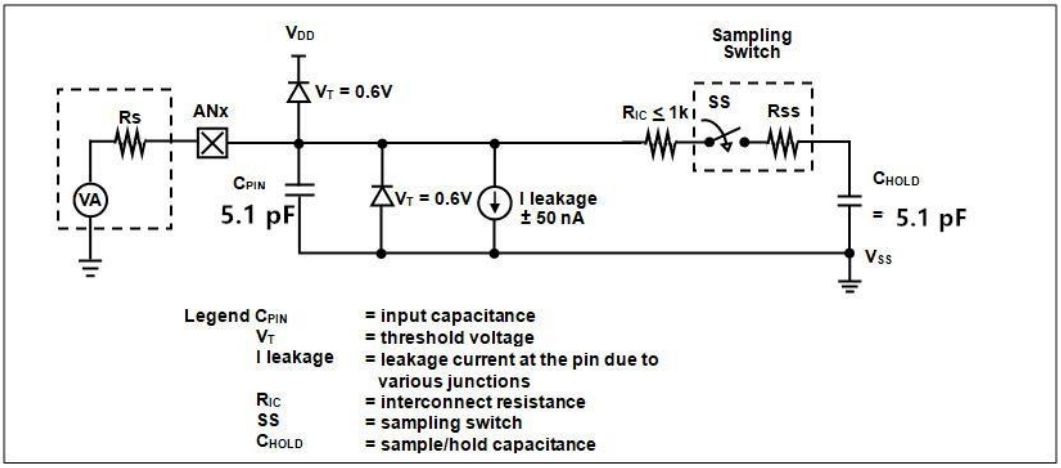


Fig.21: Analog Input Model

Before starting the AD conversion, the minimum signal acquisition time should be met for the selected analog input signal, the selection of ADCLK must be met the minimum signal acquisition time.

5.16.2. Select the reference high voltage

The ADC reference high voltage can be selected via bit[7:5] of register ***adcrgc*** and its option can be V_{DD}, 4V, 3V, 2V, 1.2V/1.6V/2.4V bandgap reference voltage or PB1 from external pin.

5.16.3. ADC clock selection

The clock of ADC module (ADCLK) can be selected by ***adcm*** register; there are 8 possible options for ADCLK from CLK÷1 to CLK÷128 (CLK is the system clock). Due to the signal acquisition time T_{ACQ} is one clock period of ADCLK, the ADCLK must meet that requirement. The recommended ADC clock is to operate at 2us.

5.16.4. Configure the analog pins

There are 14 analog signals can be selected for AD conversion, 13 analog input signals come from external pins and one is from internal bandgap reference voltage or 0.25*V_{DD}. There are 6 voltage levels selectable for the internal bandgap reference, they are 1.2V, 1.6V, 2V, 2.4V, 3V and 4V. For external pins, the analog signals are shared with Port A[0], Port A[3], Port A[4], and Port B[7:0]. To avoid leakage current at the digital circuit, those pins defined for analog input should disable the digital input function (set the corresponding bit of ***padier / pbdier / pcder*** register to be 0).

The measurement signals of ADC belong to small signal; it should avoid the measured signal to be interfered during the measurement period, the selected pin should (1) be set to input mode (2) turn off weak pull-high resistor (3) set the corresponding pin to analog input by port A/B/C digital input disable register (***padier / pbdier / pcdier***).

Note：when PC1 or PC2 is used as the AD input channel, the assignment command is different, and the emulation port will be different form the actual situation. Please refer to Table 8 for specific differences. The other channels are the same.

AD channel during actual situation	instruction	AD channel during emulation
PC1	$ ADCC enable, PC1;	PC1
PC2	$ ADCC enable, PC2;	PA1

Table 8: the difference between PC1 and PC2 in emulation and the actual situation

5.16.5. Using the ADC

The following example shows how to use ADC with PB0~PB3.

First, defining the selected pins:

```
PBC     =   0B_XXXX_0000;   //  PB0 ~ PB3 as Input
PBPH    =   0B_XXXX_0000;   //  PB0 ~ PB3 without pull-high
PBDIER  =   0B_XXXX_0000;   //  PB0 ~ PB3 digital input is disabled
```

Next, setting ***ADCC*** register, example as below:

```
$   ADCC   Enable, PB3;     //  set PB3 as ADC input
$   ADCC   Enable, PB2;     //  set PB2 as ADC input
$   ADCC   Enable, PB0;     //  set PB0 as ADC input
// Note: Only one input channel can be seleceted for each AD conversion
```

Next, setting ***ADCM*** and ***ADCRGC***register, example as below:

```
$   ADCM   12BIT, /16;      //  recommend /16 @System Clock=8MHz
$   ADCM   12BIT, /8;       //  recommend /8 @System Clock=4MHz
$   ADCRGC VDD;
```

Next, delay 400us(ADCLK=500KHz, 200*ADCLK=400us), example as below:

```
.Delay 8*400;               // System Clock=8MHz
.Delay 4*400;               // System Clock=4MHz
```

Note: If using internal reference high voltage such as bandgap 1.2V or 2V/3V/4V, the delay time must be more than 1ms.

```
$   ASDCRGC   3V;           //  AD reference voltage is 3V
.Delay 4*1010;              //  if the system clock=4MHz
                            //  the delay time must be more than 1ms
```

Please Note: If using bandgap 1.2V or 2V/3V/4V as ADC input channel, the delay time must be more than 1ms.

```
$   ADCC ADC
$   ADCRGC   VDD   ADC_BG   BG_2V   //  reference voltage is VDD
                                    //  input channel is BG_2V
.Delay 4*1010;                      //  if the system clock=4MHz
                                    //  the delay time must be more than 1ms
```

Then, start the ADC conversion:

```
AD_START =      1;          // start ADC conversion
while(!AD_DONE) NULL;       // wait ADC conversion result
```

Finally, it can read ADC result when AD_DONE is high:

```
WORD        Data;           //  two bytes result: ADCRH and ADCRL
Data$1  =   ADCRH
Data$0  =   ADCRL;
Data    =   Data >> 4;
```

The ADC can be disabled by using the following method:

```
$ ADCC Disable;
```

or

```
ADCC = 0;
```

5.17. Multiplier

There is an 8x8 multiplier on-chip to enhance hardware capability in arithmetic function, its multiplication is an 8x8 unsigned operation and can be finished in one clock cycle. Before issuing the ***mul*** command, both multiplicand and multiplicator must be put on ACC and register ***mulop*** (0x08); After ***mul*** command, the high byte result will be put on register ***mulrh*** (0x09) and low byte result on ACC. The hardware diagram of this multiplier is shown as Fig.21.

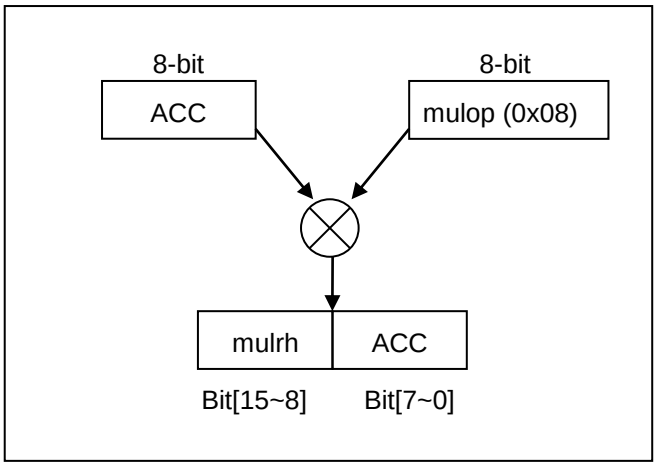


Fig.22: Block diagram of hardware multiplier

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. Please notice that bit 0 should be kept 0 due to program counter is 16 bits.

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	Enable interrupt from Timer3. 0 / 1: disable / enable.
6	0	R/W	Enable interrupt from Timer2. 0 / 1: disable / enable.
5	0	R/W	Enable interrupt from PWMG0 or EEW_Done. 0 / 1: disable / enable.
4	0	R/W	Enable interrupt from comparator. 0 / 1: disable / enable.
3	0	R/W	Enable interrupt from ADC. 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/PA4/PA3/PA6. 0 / 1: disable / enable.
0	0	R/W	Enable interrupt from PA0/PB5/PA2/PA7. 0 / 1: disable / enable.

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

Bit	Reset	R/W	Description
7	-	R/W	Interrupt Request from Timer3, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
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 PWMG0 or EEW_Done, 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 ADC, 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/PA4/PA3/PB6, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0/PB5/PA2/PA7, 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. Multiplier Operand Register (*mulop*), IO address = 0x08

Bit	Reset	R/W	Description
7 - 0	-	R/W	Operand for hardware multiplication operation.

6.8. Multiplier Result High Byte Register (*mulrh*), IO address = 0x09

Bit	Reset	R/W	Description
7 - 0	-	RO	High byte result of multiplication operation (read only).

6.9. 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	00	WO	External crystal oscillator selection. 00 : reserved 01 : Low driving capability, for lower frequency, ex: 32KHz crystal oscillator 10 : Middle driving capability, for middle frequency, ex: 1MHz crystal oscillator 11 : High driving capability, for higher frequency, ex: 4MHz crystal oscillator
4 - 1	-	-	Reserved. Please keep 0 for future compatibility.
0	0	WO	Power-down the Bandgap and LVR hardware modules. 0 / 1: normal / power-down. Note: If bandgap be disabled, there will only ILRC/T16/TM2/TM3 and I/O function can be used.

6.10. 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/PA4/PA3/PB6 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.
1 - 0	00	WO	PA0/PB5/PA2/PA7 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.

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

Bit	Reset	R/W	Description
7 - 0	0xFF	WO	Enable Port A digital input to prevent leakage when the pin is assigned for AD input. When disable is selected, the wakeup function from this pin is also disabled. 0 / 1 : disable / enable

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

Bit	Reset	R/W	Description
7 - 0	0xFF	WO	Enable Port B digital input to prevent leakage when the pin is assigned for AD input. When disable is selected, the wakeup function from this pin is also disabled. 0 / 1 : disable / enable

6.13. Port C Digital Input Enable Register (*pcdier*), IO address = 0x0f

Bit	Reset	R/W	Description
7 - 0	0xFF	WO	Enable Port C digital input to prevent leakage when the pin is assigned for AD input. When disable is selected, the wakeup function from this pin is also disabled. 0 / 1 : disable / enable

Note: Detail settings please refer to Section 9.2.

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

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Data register for Port A.

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

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Port A control registers. This register is used to define input mode or output mode for each corresponding pin of port A. 0 / 1: input / output Please note that PA5 can be INPUT or OUTPUT.

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

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Port A pull-high register. 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

6.17. Port B Data Register (*pb*), IO address = 0x13

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Data register for Port B.

6.18. Port B Control Register (*pbc*), IO address = 0x14

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

6.19. Port B Pull-High Register (*pbph*), IO address = 0x15

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

6.20. Port C Data Register (*pc*), IO address = 0x16

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Data register for Port C.

6.21. Port C Control Register (*pcc*), IO address = 0x17

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Port C control register. This register is used to define input mode or output mode for each corresponding pin of port B. 0 / 1: input / output

6.22. Port C Pull-High Register (*pcph*), IO address = 0x18

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

6.23. Port A Pull-Low Register (*papl*), IO address = 0x19

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Port A pull-low register. 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

6.24. Port B Pull-Low Register (*pbpl*), IO address = 0x1A

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

6.25. Port C Pull-Low Register (*pcpl*), IO address = 0x1B

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

6.26. ADC Control Register (*adcc*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable ADC function. 0/1: Disable/Enable.
6	0	R/W	ADC process control bit. Write “1” to start conversion Read “1” to indicate the ADC is ready or end of conversion.
5 - 2	0000	R/W	Channel selector. These four bits are used to select input signal for AD conversion. 0000: PB0/AD0, 0001: PB1/AD1, 0010: PB2/AD2, 0011: PB3/AD3, 0100: PB4/AD4, 0101: PB5/AD5, 0110: PB6/AD6, 0111: PB7/AD7, 1000: PA3/AD8, 1001: PA4/AD9, 1010: PA0/AD10, 1011: PC1/AD11, (PC1 or PA1 for ICE at this address, please refer to section 5.15.4) 1100: PC2/AD12, (PC1 or PA1 for ICE at this address, please refer to section 5.15.4) 1111: (Channel F) Bandgap reference voltage or 0.25*V_{DD} Others: reserved
0 - 1	-	-	Reserved. (keep 0 for future compatibility)

6.27. ADC Mode Register (*adcm*), IO address = 0x21

Bit	Reset	R/W	Description
7 - 4	-	-	Reserved. (keep 0 for future compatibility)
3 - 1	000	R/W	ADC clock source selection. 000: CLK (system clock) ÷ 1, 001: CLK (system clock) ÷ 2, 010: CLK (system clock) ÷ 4, 011: CLK (system clock) ÷ 8, 100: CLK (system clock) ÷ 16, 101: CLK (system clock) ÷ 32, 110: CLK (system clock) ÷ 64, 111: CLK (system clock) ÷ 128,
0	-	-	Reserved.

6.28. ADC Regulator Control Register (*adcrgc*), IO address = 0x24

Bit	Reset	R/W	Description
7 - 5	000	WO	These three bits are used to select input signal for ADC reference high voltage. 000: V_{DD}, 001: 2V, 010: 3V, 011: 4V, 100: PB1, 101: Bandgap 1.20 volt reference voltage 110: Bandgap 1.60 volt reference voltage 111: Bandgap 2.40 volt reference voltage
4	0	WO	ADC channel F selector: 0: Bandgap reference voltage 1: 0.25*V_{DD}. The deviation is within ±0.01*V_{DD} mostly.
3 - 1	00	WO	Bandgap reference voltage selector for ADC channel F: (ICE is fixed to 1.2V) 000: 1.2V 010: 2V 100: 3V 110: 4V 001: 1.6V 011: 2.4V
1 - 0	-	-	Reserved. Please keep 0.

6.29. ADC Result High Register (*adcrh*), IO address = 0x22

Bit	Reset	R/W	Description
7 - 0	-	RO	These eight read-only bits will be the bit [11:4] of AD conversion result. The bit 7 of this register is the MSB of ADC result for any resolution.

6.30. ADC Result Low Register (*adcrl*), IO address = 0x23

Bit	Reset	R/W	Description
7 - 4	-	RO	These four bits will be the bit [3:0] of AD conversion result.
3 - 0	-	-	Reserved.

6.31. MISC Register (*misc*), IO address = 0x26

Bit	Reset	R/W	Description
7 - 6	-	-	Reserved. (keep 0 for future compatibility)
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	0	WO	Enable VDD/2 LCD bias voltage generator. 0 / 1 : Disable / Enable (ICE cannot be dynamically switched) If Code Option selects LCD output, but MISC.4 does not set to 1, then the VDD/2 bias cannot be output on the IC. However, the emulator is always OK. Two above phenomena are different.
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

6.32. Comparator Control Register (*gpcc*), IO address = 0x2b

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(not suitable for the comparator wake-up function) 011 : $V_{internal\ R}$ 100 : PB6 101 : PB7 11X: reserved
0	0	R/W	Selection the plus input (+) of comparator. 0 : $V_{internal\ R}$ 1 : PA4

6.33. Comparator Selection Register (*gpcs*), IO address = 0x2c

Bit	Reset	R/W	Description
7	0	WO	Comparator output enable (to PA0). 0 / 1 : disable / enable
6	-	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)

6.34. Timer2 Control Register (*tm2c*), IO address = 0x30

Bit	Reset	R/W	Description
7 - 4	0000	R/W	Timer2 clock selection. 0000 : disable 0001 : CLK 0010 : IHRC or IHRC *2 (by code option TMx_source) 0011 : EOSC 0100 : ILRC 0101 : comparator output (ICE does NOT Support.) 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) Others: reserved **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 : PB2 10 : PA3 11 : PB4
1	0	R/W	Timer2 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer2 output. 0 / 1: disable / enable

6.35. Timer2 Counter Register (*tm2ct*), IO address = 0x31

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

6.36. Timer2 Scalar Register (*tm2s*), IO address = 0x32

Bit	Reset	R/W	Description
7	0	WO	PWM resolution selection. 0 : 8-bit 1 : 6-bit or 7-bit (by code option TMx_bit)
6 - 5	00	WO	Timer2 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4 - 0	00000	W	Timer2 clock scalar.

6.37. Timer2 Bound Register (*tm2b*), IO address = 0x33

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

6.38. Timer3 Control Register (*tm3c*), IO address = 0x34

Bit	Reset	R/W	Description
7 - 4	0000	R/W	Timer3 clock selection. 0000 : disable 0001 : CLK 0010 : IHRC or IHRC *2 (by code option TMx_source) 0011 : EOSC 0100 : ILRC 0101 : comparator output 1000 : PA0 (rising edge) 1001 : ~PA0 (falling edge) 1010 : PB0 (rising edge) 1011 : ~PB0 (falling edge) 1100 : PA4 (rising edge) 1101 : ~PA4 (falling edge) Others: reserved Notice: In ICE mode and IHRC is selected for Timer3 clock, the clock sent to Timer3 does NOT be stopped, Timer3 will keep counting when ICE is in halt state.
3 - 2	00	R/W	Timer3 output selection. 00 : disable 01 : PB5 10 : PB6 11 : PB7
1	0	R/W	Timer3 mode selection. 0 / 1 : period mode / PWM mode
0	0	R/W	Enable to inverse the polarity of Timer3 output. 0 / 1: disable / enable

6.39. Timer3 Counter Register (*tm3ct*), IO address = 0x35

Bit	Reset	R/W	Description
7 - 0	0x00	R/W	Bit [7:0] of Timer3 counter register.

6.40. Timer3 Scalar Register (*tm3s*), IO address = 0x36

Bit	Reset	R/W	Description
7	0	WO	PWM resolution selection. 0 : 8-bit 1 : 6-bit or 7-bit (by code option TMx_bit)
6 - 5	00	WO	Timer3 clock pre-scalar. 00 : ÷ 1 01 : ÷ 4 10 : ÷ 16 11 : ÷ 64
4 - 0	00000	WO	Timer3 clock scalar.

6.41. Timer3 Bound Register (*tm3b*), IO address = 0x37

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Timer3 bound register.

6.42. EEPROM Data Low Register (*eerl*), IO address = 0x3C

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

6.43. EEPROM Control Registers (*eermc*), IO address = 0x3D

Bit	Reset	R/W	Description
7	0x00	R	Reserved
6			EEPROM Busy flag. 0 / 1 : Done / Busy
5-0			Reserved
7-0	0x00	W	Write 0x5A before each STEER to Enable EEPROM Byte Read / Writer
			Write 0xA5 before each STEER to Enable EEPROM Erase All

6.44. PWMG0 control Register (*pwmg0c*), IO address = 0x40

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG0 generator. 0 / 1 : diable / enable.
6	-	RO	Output status of PWMG0 generator.
5	0	R/W	Enable to inverse the polarity of PWMG0 generator output. 0 / 1 : disable / enable.
4	0	R/W	PWMG0 counter reset. Writing "1" to clear PWMG0 counter and this bit will be self-clear to 0 after counter reset.
3 - 1	000	R/W	Select PWM output pin for PWMG0. 000: none 001: PB5 010: PC2 011: PA0 100: PB4 Others: reserved
0	0	R/W	Clock source of PWMG0 generator. 0: CLK, 1: IHRC or IHRC*2 (by code option PWM_source)

6.45. PWMG0 Scalar Register (*pwmg0s*), IO address = 0x41

Bit	Reset	R/W	Description
7	0	WO	PWMG0 interrupt mode. 0: Generate interrupt when counter matches the duty value 1: Generate interrupt when counter is 0.
6 - 5	00	WO	PWMG0 clock pre-scalar. 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 - 0	00000	WO	PWMG0 clock divider.

6.46. PWMG0 Duty Value High Register (*pwmg0dth*), IO address = 0x42

Bit	Reset	R/W	Description
7 - 0	-	WO	Duty values bit[10:3] of PWMG0.

6.47. PWMG0 Duty Value Low Register (*pwmg0dtl*), IO address = 0x43

Bit	Reset	R/W	Description
7 - 5	-	WO	Duty values bit [2:0] of PWMG0.
4 - 0	-	-	Reserved.

Note: It's necessary to write PWMG0 Duty_Value Low Register before writing PWMG0 Duty_Value High Register.

6.48. PWMG0 Counter Upper Bound High Register (*pwmg0cubh*), IO address = 0x44

Bit	Reset	R/W	Description
7 - 0	-	WO	Bit [10:3] of PWMG0 counter upper bound.

6.49. PWMG0 Counter Upper Bound Low Register (*pwmg0cubl*), IO address = 0x45

Bit	Reset	R/W	Description
7 - 6	-	WO	Bit [2:1] of PWMG0 counter upper bound.
5 - 0	-	-	Reserved.

6.50. PWMG1 control Register (*pwmg1c*), IO address = 0x46

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG1. 0 / 1: disable / enable.
6	-	RO	Output of PWMG1.
5	0	R/W	Enable to inverse the polarity of PWMG1 output. 0 / 1 : disable / enable.
4	0	R/W	PWMG1 counter reset. Writing "1" to clear PWMG1 counter. After clearing out the PWMG1 count, this bit will automatically return to 0.
3 - 1	000	R/W	Select PWMG1 output pin. 000: none 001: PB6 010: PC3 011: PA4 100: PB7 Others: reserved
0	0	R/W	Clock source of PWMG1 generator. 0: CLK 1: IHRC or IHRC*2 (by code option PWM_source)

6.51. PWMG1 Scalar Register (*pwmg1s*), IO address = 0x47

Bit	Reset	R/W	Description
7	0	WO	PWMG1 interrupt mode. 0: Generate interrupt when counter matches the duty value. 1: Generate interrupt when counter is 0.
6 - 5	00	WO	PWMG1 clock pre-scalar. 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 - 0	00000	WO	PWMG1 clock divider.

6.52. PWMG1 Duty Value High Register (*pwmg1dth*), IO address = 0x48

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Duty values bit [10:3] of PWMG1.

6.53. PWMG1 Duty Value Low Register (*pwmg1dtl*), IO address = 0x49

Bit	Reset	R/W	Description
7 - 5	000	W	Duty values bit [2:0] of PWMG1.
4 - 0	-	-	Reserved.

Note: It's necessary to write PWMG1 Duty_Value Low Register before writing PWMG1 Duty_Value High Register.

6.54. PWMG1 Counter Upper Bound High Register (*pwmg1cubh*), IO address = 0x4a

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Bit[10:3] of PWMG1 counter upper bound.

6.55. PWMG1 Counter Upper Bound Low Register (*pwmg1cubl*), IO address = 0x04b

Bit	Reset	R/W	Description
7 - 6	000	WO	Bit[2:1] of PWMG1 counter upper bound.
5 - 0	-	-	Reserved

6.56. PWMG2 control Register (*pwmg2c*), IO address = 0x4C

Bit	Reset	R/W	Description
7	0	R/W	Enable PWMG2. 0 / 1 : disable / enable.
6	-	RO	Output of PWMG2.
5	0	R/W	Enable to inverse the polarity of PWMG2 output. 0 / 1 : disable / enable.
4	0	R/W	PWMG2 counter reset. Writing "1" to clear PWMG2 counter. After clearing out the PWMG1 count, this bit will automatically return to 0.
3 - 1	000	R/W	Select PWMG2 output pin. 000: disable 001: PB3 010: PC0 011: PA3 100: PB2 Others: reserved
0	0	R/W	Clock source of PWMG2 generator. 0: CLK, 1: IHRC or IHRC*2 (by code option PWM_source)

6.57. PWMG2 Scalar Register (*pwmg2s*), IO address = 0x4D

Bit	Reset	R/W	Description
7	0	WO	PWMG2 interrupt mode. 0: Generate interrupt when counter matches the duty value. 1: Generate interrupt when counter is 0.
6 - 5	00	WO	PWMG2 clock pre-scalar. 00 : ÷1 01 : ÷4 10 : ÷16 11 : ÷64
4 - 0	00000	WO	PWMG2 clock divider.

6.58. PWMG2 Duty Value High Register (*pwmg2dth*), IO address = 0x4E

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Duty values bit [10:3] of PWMG2.

6.59. PWMG2 Duty Value Low Register (*pwmg2dtl*), IO address = 0x4F

Bit	Reset	R/W	Description
7 - 5	000	WO	Duty values bit [2:0] of PWMG2.
4 - 0	-	-	Reserved.

Note: It's necessary to write PWMG2 Duty_Value Low Register before writing PWMG2 Duty_Value High Register.

6.60. PWMG2 Counter Upper Bound High Register (*pwmg2cubh*), IO address = 0x50

Bit	Reset	R/W	Description
7 - 0	0x00	WO	Bit [10:3] of PWMG2 counter upper bound.

6.61. PWMG2 Counter Upper Bound Low Register (*pwmg2cubl*), IO address = 0x51

Bit	Reset	R/W	Description
7 - 6	000	WO	Bit [2:1] of PWMG2 counter upper bound.
5 - 0	-	-	Reserved.

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 0x00~0x7F (0~127) 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* pb, a; Result: pb ← a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nmov M, a	Take the negative logic (2's complement) of ACC to put on memory Example: *mov* MEM, a; Result: MEM ← 〒a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- *mov* a, 0xf5 ; // ACC is 0xf5 *nmov* ram9, a; // ram9 is 0x0b, ACC is 0xf5 ---
nmov a, M	Take the negative logic (2's complement) of memory to put on 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 Application Example: --- *mov* a, 0xf5 ; *mov* ram9, a ; // ram9 is 0xf5 *nmov* a, ram9 ; // ram9 is 0xf5, ACC is 0x0b ---

ldtabh index	Load high byte data in MTP program memory to ACC by using index as MTP address. It needs 2T to execute this instruction. Example: *ldtabh index;* Result: a ← {bit 15~8 of MTP [index]}; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: ---

```
    word        ROMptr ;            // declare a pointer of ROM in RAM
    ...
    mov         a, la@TableA ;   // assign pointer to ROM TableA (LSB)
    mov         lb@ROMptr, a ;   // save pointer to RAM (LSB)
    mov         a, ha@TableA ;   // assign pointer to ROM TableA (MSB)
    mov         hb@ROMptr, a ;   // save pointer to RAM (MSB)
    ...
    ldtabh      ROMptr ;            // load TableA MSB to ACC (ACC=0X02)
    ...
TableA :    dc      0x0234, 0x0042, 0x0024, 0x0018 ;
```

---

ldtabl index

Load low byte data in MTP to ACC by using index as MTP address. It needs 2T to execute this instruction.

Example: *ldtabl index;*

Result: a ← {bit7~0 of MTP [index]};

Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Application Example:

```
    word        ROMptr ;            // declare a pointer of ROM in RAM
    ...
    mov         a, la@TableA ; // assign pointer to ROM TableA (LSB)
    mov         lb@ROMptr, a ; // save pointer to RAM (LSB)
    mov         a, ha@TableA ; // assign pointer to ROM TableA (MSB)
    mov         hb@ROMptr, a ; // save pointer to RAM (MSB)
    ...
    ldtabl      ROMptr ;            // load TableA LSB to ACC (ACC=0x34)
    ...
TableA :    dc      0x0234, 0x0042, 0x0024, 0x0018 ;
```

---

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

Application Example:

```
    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
Application Example:

```
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: word ← 16-bit timer
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 ; // move 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
mul	Multiplication operation, 8x8 unsigned multiplications will be executed. Example: *mul* ; Result: {MulRH,ACC} ← ACC * MulOp Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example : --- … mov a, 0x5a ; mov mulop, a ; mov a, 0xa5 ; mul // 0x5A * 0xA5 = 3A02 (mulrh + ACC) mov ram0, a ; // LSB, ram0=0x02 mov a, mulrh ; // MSB, ACC=0X3A … ---

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
swap M	Swap the high nibble and low nibble of memory Example: *swap* MEM ; Result: MEM (b3,b2,b1,b0,b7,b6,b5,b4) ← MEM (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 a, IO	Perform logic XOR on ACC and IO register, then put result into ACC Example: *xor a, pa ;* Result: a ← a ^ pa ; // pa is the data register of port A 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, I	Compare ACC with immediate data Example: *comp* a, 0x55; Result: Flag will be changed by regarding as (a - 0x55) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV Application Example: --- *mov* a, 0x38 ;

	comp a, 0x38 ; // Z flag is set as 1 *comp* a, 0x42 ; // C flag is set as 1 *comp* a, 0x24 ; // C, Z flags are clear *comp* a, 0x6a ; // C, AC flags are set as 1 ---
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* pb.5 ; Result: set bit 5 of port B to high Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
tog IO.n	Toggle bit state of bit n of IO port Example: *tog* pa.5 ; Result: toggle bit 5 of port A Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
swapc IO.n	Swap the nth bit 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
ceqsn M, a	Compare ACC with memory and skip next instruction if both are equal. Example: *ceqsn* MEM, a; 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
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 CPU 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
wdreset	Reset Watchdog timer. Example: *wdreset* ; Result: Reset Watchdog timer. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

7.8. Summary of Instructions Execution Cycle

2T		*goto, call, idxm, pcadd, ret, reti, ldtabl , ldtabh*
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	-	-	-	-	*nmov* M, a	-	-	-	-
nmov a, M	Y	-	-	-	*ldtabh* index	-	-	-	-	*ldtabl* index	-	-	-	-
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	-	-	-	-
swap M	-	-	-	-	*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* *a, IO*	Y	-	-	-
xor IO, a	-	-	-	-	*xor* a, M	Y	-	-	-	*xor* M, a	Y	-	-	-
not a	Y	-	-	-	*not* M	Y	-	-	-	*neg* a	Y	-	-	-
neg M	Y	-	-	-	*comp* a, I	Y	Y	Y	Y	*comp* a, M	Y	Y	Y	Y
comp M, a	Y	Y	Y	Y	*set0* IO.n	-	-	-	-	*set1* IO.n	-	-	-	-
tog IO.n	-	-	-	-	*set0* M.n	-	-	-	-	*set1* M.n	-	-	-	-
swapc IO.n	-	Y	-	-	*ceqsn* a, I	Y	Y	Y	Y	*ceqsn* a, M	Y	Y	Y	Y
ceqsn M, a	Y	Y	Y	Y	*cneqsn* *a, I*	Y	Y	Y	Y	*cneqsn 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 a, M	Y	Y	Y	Y	*nadd* M, a	Y	Y	Y	Y	*steer* index	-	-	-	-
ldeer index	-	-	-	-	*mul*	-	-	-	-					

7.10. BIT definition

Bit access of RAM is only available for address from 0x00 to 0x7F.

8. Code Options

Option	Selection	Description
Security	**Enable**	**Security Enable**
	Disable	Security Disable
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**
LCD (please refer to MISC.4)	PB0_A034	VDD/2 LCD bias voltage generator enabled, PB0 PA[0,3,4] are VDD/2 if input mode
	PB_1256	VDD/2 LCD bias voltage generator enabled, PB[1,2,5,6] are VDD/2 if input mode
Interrupt Src0	**PA.0**	**INTEN/ INTRQ.Bit0 is from PA.0**
	PB.5	INTEN/ INTRQ.Bit0 is from PB.5
	PA.2	INTEN/ INTRQ.Bit0 is from PA.2
	PA.7	INTEN/ INTRQ.Bit0 is from PA.7
Interrupt Src1	**PB.0**	**INTEN/ INTRQ.Bit1 is from PB.0**
	PA.4	INTEN/ INTRQ.Bit1 is from PA.4
	PA.3	INTEN/ INTRQ.Bit1 is from PA.3
	PB.6	INTEN/ INTRQ.Bit1 is from PB.6
PB4_PB7_Drive	Normal	PB4 & PB7 Drive/ Sink Current is Normal
	Strong	**PB4 & PB7 Drive/ Sink Current is Strong**

Option	Selection	Description
Comparator Edge	**All_Edge**	**The comparator will trigger an interrupt on both 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
GPC_PWM	**Disable**	**Comparator does not control all PWM outputs**
	Enable	Comparator controls all PWM outputs (ICE does NOT Support.)
PWM_Source	**16MHZ**	**When pwmg0c.0= 1, PWMG0 clock source = IHRC = 16MHZ** **When pwmg1c.0= 1, PWMG1 clock source = IHRC = 16MHZ** **When pwmg2c.0= 1, PWMG2 clock source = IHRC = 16MHZ**
	32MHZ	When pwmg0c.0= 1, PWMG0 clock source = IHRC*2 = 32MHZ When pwmg1c.0= 1, PWMG1 clock source = IHRC*2 = 32MHZ When pwmg2c.0= 1, PWMG2 clock source = IHRC*2 = 32MHZ (ICE does NOT Support.)
TMx_Source	**16MHZ**	**When tm2c[7:4]= 0010, TM2 clock source = IHRC = 16MHZ** **When tm3c[7:4]= 0010, TM3 clock source = IHRC = 16MHZ**
	32MHZ	When tm2c[7:4]= 0010, TM2 clock source = IHRC*2 = 32MHZ When tm3c[7:4]= 0010, TM3 clock source = IHRC*2 = 32MHZ (ICE does NOT Support.)
TMx_Bit	**6 Bit**	**When tm2s.7=1, TM2 PWM resolution is 6 Bit** **When tm3s.7=1, TM3 PWM resolution is 6 Bit**
	7 Bit	When tm2s.7=1, TM2 PWM resolution is 7 Bit When tm3s.7=1, TM3 PWM resolution is 7 Bit (ICE does NOT Support.)
PAC5EnB	Enable	When PA5 is set to 0utput, on-board self-powered ISP Mode is supported, but not recommended (On-Board Writing Mode) When PA5 is set to Input, on-board self-powered ISP Mode is supported (On-Board Writing Mode) Except for an empty film and doing the first programming
	Disable	**Support on-board self-powered ISP Mode (On-Board Writing Mode)** **(PA5 can only be used as Input, the PA5 output is invalid at this time)**
ICE_Lib_Code	**Normal**	**Only for ICE: Normal Simulation**
	DownSize	Only for ICE: ROM DownSize, not supported Watch not supported

Note: The **Bolded** options are the default options.

9. Special Notes

This chapter is to remind user who use PGS134 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) If IO pin is set to be digital input and enable wake-up function

- ◆ Configure IO pin as input.
- ◆ Set corresponding bit to “1” in PXDIER.

(3) PA5 is set to be output pin

- ◆ Set PAC5EnB to Enable in CodeOption, and when PA5 is set to Input, the on-board self-powered ISP Mode is supported.
- ◆ Set PAC5EnB to Enable in CodeOption. When PA5 is set to Output, although the on-board self-powered ISP Mode is supported, the PA5 output is low, and writing may damage the IC.

(4) PA5 is set to be PRSTB input pin

- ◆ Configure PA5 as input.
- ◆ Set CLKMD.0=1 to enable PA5 as PRST# input pin.

(5) 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.
- ◆ The PA5 input voltage (including high-frequency surge voltage) must be lower than VDD+0.6V (V_{PA5} < VDD+0.6V), otherwise the PA4/PA6 IO output will stop.

(6) 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
- ◆ EOSCR register bit [6:5] selects corresponding crystal oscillator frequency :
  - ✧ 01 : for lower frequency, ex : 32KHz
  - ✧ 10 : for middle frequency, ex : 455KHz, 1MHz
  - ✧ 11 : for higher frequency, ex : 4MHz
- ◆ Program EOSCR.7 =1 to enable crystal oscillator
- ◆ 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.

(3) There are two sets of external IO pin interrupt source. Every set is decided by code option Interrupt Src0 and Interrupt Src1 corresponding to the unique interrupt pin. Please comply with the ***inten / intrq / integs*** register when selecting IO pin.

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 is open by default, but when the program executes ADJUST_IC, the watchdog will be closed. To use the watchdog, you need to reconfigure the open.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	≧ 2.2V	≧ 2.2V
4MHz	≧ 2.5V	≧ 2.5V
8MHz	≧ 3.5V	≧ 3.5V

Table 9: 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(B) to program. 3S-P-002 or older versions do not support programming PGS134.

- There are 5 pins for using the writer to program:, PA4, 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 5.5V.
  (2) V_{DD} may be higher than 5.5V, 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.

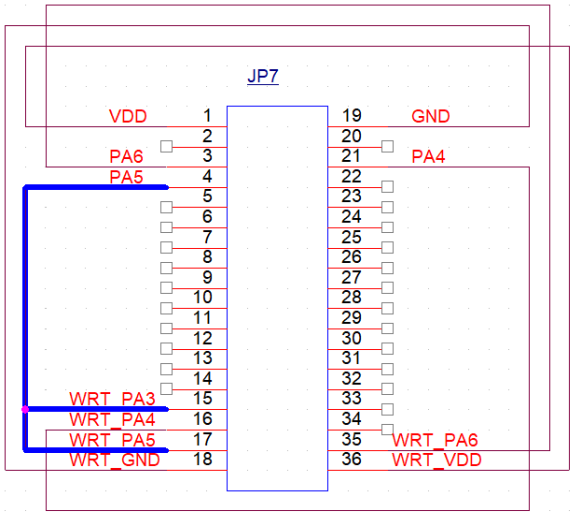
Please select the following program mode according to the actual situation.

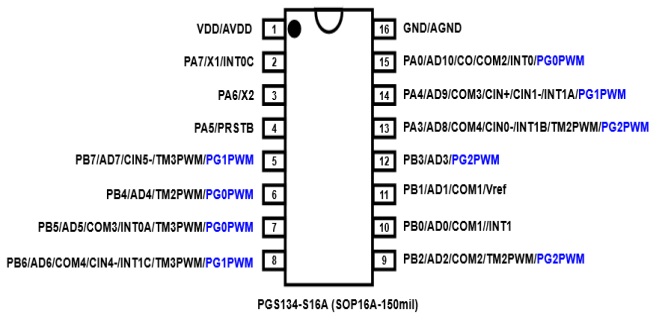
Normal Programming Mode

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(B) to program.

NOTE: In addition of ICVPP(PA5) shorten with IC's PA5 on the JP7 jumper board, the ICVPP2(PA3) also MUST be shorten with IC's PA5.

For example, the signal connect diagram of the JP7 jumper of PGS134-SOP16A on the writer is as the following:





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 are used for On-Board Writing: ICPCK(PA4), ICPDA(PA6), VDD, GND and ICVPP(PA5).

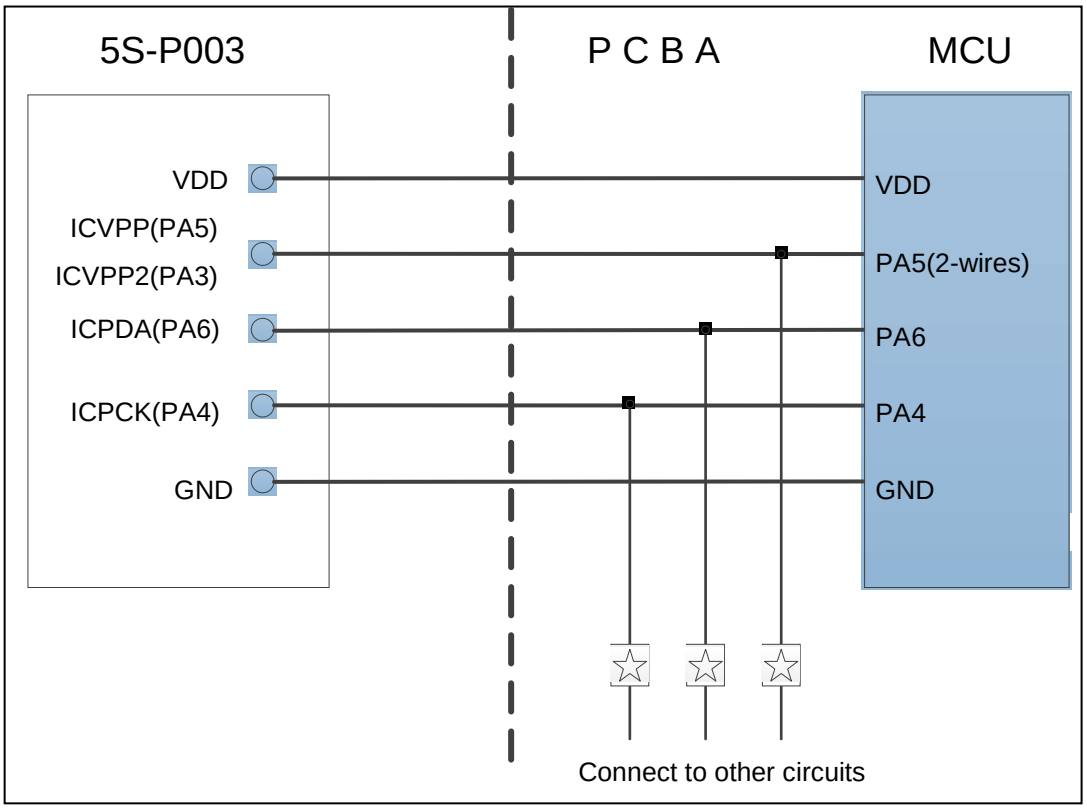


Fig. 23: Schematic Diagram of On-Board Wiring

The symbol ☆ on Fig. 23 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.

On-Board Writing with self-power mode

The PGS134 also supports on-board programming powered by the customer's board. To use this function, you need to set "On-board Program w/ Self-Power" in the programming file, which can be set in the menu of the burner software To Package. Or add the package parameter settings in the original code as a command, please refer to the burner instruction manual for detailed operation.

The burner will detect the VDD voltage on the customer's board before flashing. Therefore, please make sure that the VDD voltage on the board is higher than 2.5V, and if the VDD voltage on the application board is lower than 2.5V, the burner will display the error "**Find a diff. IC.**". The rest is the same as the burner-powered on-board programming mode.

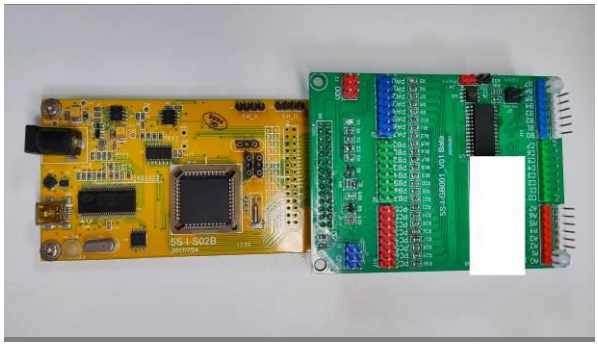
Converter PDK to Package:

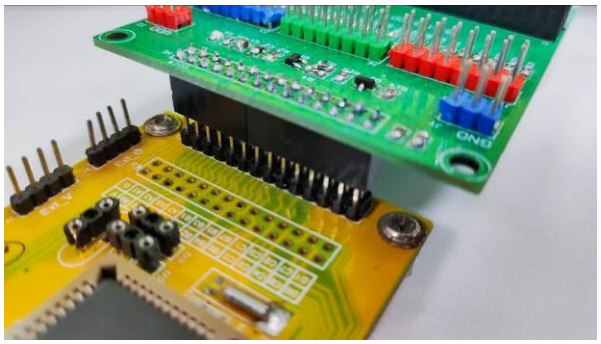
When the self-powered ISP option is checked, if CodeOption PAC.5 is set to Enable, writing may damage the IC when the PA5 output is low.

Therefore, when PA5 is set to 0utput, it can support on-board self-powered ISP Mode, but it is not recommended.

9.2. Using ICE

Emulate PGS134 Please use the 5S-I-S01/2(B) plug-in emulation board 5S-I-GB001, refer to the following picture:





Note the following when simulating:

(1) Simulation timing

- When simulating, especially when simulating EEPROM and PxPL-related functions, the 5S-I-S01/2(B) communicates with the emulation board 5S-I-GB001, so the simulation may be slightly slower than the actual IC, mainly affecting the following register configuration EERL/EERMC/STEER/LDEER/MISC2/INTRQ/INTRQ2/PAPL/PBPL/PCPL/LPWMGxC/LPWMGxDTH /LPWMGxDTL /LPWMGCLK/LPWMBCUBH/LPWMCUBL etc.

(2) Simulation voltages

- During simulation, due to the working voltage of the simulation board, it is recommended that the simulation voltage ≥ 2.5V

(3) Other precautions

- 5S-I-S01/2(B) doesn't support the instruction NMOV/SWAP/NADD/COMP with RAM.
- 5S-I-S01/2(B) doesn't support SYSCLK=ILRC/16.
- 5S-I-S01/2(B) doesn't support the dynamic setting of function ***misc.4*** (Only fix to 0 or 1)
- 5S-I-S01/2(B) doesn't support the function ***Tm2.gpcrs***/***Tm3.gpcrs***.
- 5S-I-S01/2(B) doesn't support bandgap reference voltage for ADC channel F of ADCRGC [3:2]. Only 1.2V exists and is fixed.
- 5S-I-S01/2(B) has different setting of PC2 and PC1 in ***adcc***.
- 5S-I-S01/2(B) doesn't support the code options: GPC_PWM, TMx_source, PWM_Source and TMx_bit.

- 5S-I-S01/2(B) only has 240 bytes RAM for data memory.
- 5S-I-S01/2(B) only has 0xF00 program memory.
- The PCDIER register of the 5S-I-S01/2(B) emulator is different from the real chip. The PCDIER[0] of the 5S-I-S01/2(B) is used to set PC0~PC3 to be digital input whereas PCDIER[1] is used to set PC4~ PC7 to be digital input. It is recommended not to set PCDIER.
- When 5S-I-S01/2 (B), PC7 needs to be set to output high.
- When using PB1 in ADCRGC, PA1 must float.
- When using GPCC output, PA3 will be influenced.
- When simulating PWM waveform, please check the waveform during program running. When the ICE is suspended or single-step running, its waveform may be inconsistent with the reality.
- The ILRC frequency of the 5S-I-S01/2(B) simulator is different from the actual IC and is uncalibrated, with a frequency range of about 34K~38KHz.
- When using 5S-I-S01/2(B) for simulation, changing the value of tm2ct/tm3ct will affect the duty during timer2/timer3 period mode. But it will not be affected for the actual IC.
- Fast Wakeup time is different from 5S-I-S01/2(B): 128 SysClk, PGS134: 45 ILRC
- Watch dog time out period is different from 5S-I-S01/2(B):

WDT period	**5S-I-S01/2**(B)	**PGS134**
misc[1:0]=00	2048 * T_{ILRC}	8192 * T_{ILRC}
misc[1:0]=01	4096 * T_{ILRC}	16384 * T_{ILRC}
misc[1:0]=10	16384 * T_{ILRC}	65536 * T_{ILRC}
misc[1:0]=11	256 * T_{ILRC}	262144 * T_{ILRC}