十速科技股份有限公司
tenx technology inc.

Advance Information

TM89 Series MCU User Manual

Tenx reserves the right to change or discontinue this product without notice.

tenx technology, inc.

CONTENTS

Chapter 1 TM89 Series Internal System Architecture

1-1. Power Supply

The TM89 Series MCU uses the Mask Option method to provide two operating voltage ranges: 1.5V (1.2~1.8V) and 3V (2.2~3.6V). Three options are also available for supplying the driver voltage required for the LCD driver: 1/3Bias, 1/4Bias and 1/5Bias.

1-1-1. System Power Supply

VBAT is the positive power pin while the BAK pin provides the operating voltage required for the frequency generator circuit (including external pins) and the MCU's internal logic circuits.

Where the 3V (BCF=0: BAK=VL1) power mode is chosen, the voltage level on the BAK pin will vary with changes in the backup modes. An external 0.1 uF capacitor must therefore be connected to regulate the operating voltage. Additionally, when using other power modes the BAK pin must be connected directly to the VBAT pin during operation as well.

The TM89 Series MCU also offers the option for the LCD driver voltage to be provided by an external voltage regulation circuit. The MCU's external circuitry can then be used to generate a regulated voltage for direct use by the VL1 or VL2 pins. When using this option, the output voltage from the LCD driver is supplied by an external voltage regulator while other IO pins will still operate in the voltage range between VBAT and GND.

The BAK, VL1 and VL2 connection methods corresponding to the Mask Options described above are as follows:

Power Mode	Mask Option 1	Mask Option 2	BAK Connection	VL1 Connection	VL2 Connection
1.5V	-	-	VBAT	VBAT or External voltage regulator	Capacitor
3V	BCF=0 : BAK=VL1	No voltage regulation	Capacitor	Capacitor	VBAT
		VL1 external voltage regulator	Capacitor	External voltage regulator	Capacitor
		VL2 external voltage regulator	Capacitor	Capacitor	External voltage regulator
	BCF=0 : BAK=VBAT	No voltage regulator	VBAT	Capacitor	VBAT
		VL1 external voltage regulator	VBAT	External voltage regulator	Capacitor
		VL2 external voltage regulator	VBAT	Capacitor	External voltage regulator

Notes:

1. When connecting a capacitor, use a capacitor rated at 0.1uF or above and connect the other end to GND.
2. When using the mask options for 3V (BCF=0 : BAK=VL1) and powering the LCD driver output pin from an external voltage regulation circuit, make sure that the VL1 voltage does not go below 1.1V regardless of whether the external stabilized voltage is used by VL1 or VL2. If VL1 drops below 1.1V it may lead to abnormal operation in the oscillation circuit or the MCU's internal circuitry.
3. Mask Option 1: Power source's mask option
4. Mask Option 2: Mask option for external regulator with LCD

1-1-2. LCD Driver's Power Source

The TM89 Series MCU uses the Charge Pump method to generate the voltage levels (VL1 ~ VL5) needed by the LCD Driver. The maximum voltage is determined by the mask option for LCD Bias. For 1/3, 1/4 and 1/5 Bias, the maximum voltages are VL3, VL4 and VL5 respectively.

The TM89 Series MCU allows the user to select the Charge Pump frequency using three types of mask options: PH3, PH4 and PH5. The user can use the actual load of the LCD panel, the LCD frame frequency (see 3-1 for more details) and LCD Bias to adjust the Charge Pump frequency. Generally speaking the Charge Pump frequency should be set to double the LCD frame frequency to get a better pumping efficiency. In practice however the actual setting should depend on the display results from a test with the EV chip (TM8999) in order to properly balance the MCU power consumption against the LCD panel load.

For larger LCD panel loads, choose a faster charge pump frequency to get a more stable voltage output from the LCD driver; if the LCD panel load is smaller then choosing PH5 helps to reduce the MCU power consumption.

For a larger area LCD panels, the Charge Pump architecture means that the higher power consumption from the LCD panel itself will make it difficult for the LCD driver's maximum voltage output to reach a stable value. This problem can be rectified using one of the methods listed below:

1. Where the charge pump circuit can charge the VL1 ~ VL5 capacitors to the expected voltage values, increase the capacitance used for the capacitor between the CUP pins and the VL1 ~ VL5 ground capacitors.
2. Where the charge pump circuit can charge the VL1 ~ VL5 capacitors to the expected voltage values, choose a faster Charge Pump frequency.
3. Some TM89 Series MCUs also offer for a Mask Option for 1/3 and 1/4 bias similar to the one offered for 1/5 bias. A capacitor can be connected between CUPN and CUP0 as well as CUP1 and CUP2 to increase the effective charge pump efficiency.

When the LCD driver's output voltage is higher than 6V, the production process used means that the reverse-bias current for the MCU as a whole will increase noticeably, unnecessarily increasing the MCU's power consumption. If the maximum output voltage of the LCD driver is over 6V we recommend using an external voltage regulator to provide a lower voltage to the LCD driver. This will help keep the LCD driver's maximum output voltage under 6V.

1-1-2-1. Using the Voltage Level at VL1 for the Charge Pump Circuit's Base Voltage

When using the 1.5V or 3V power mode with the VL1 voltage supplied by an external voltage regulator, VL1 will supply the base voltage for the Charge Pump circuit.

The VL2 ~ VL5 voltage levels generated by the charge pump circuit will then be as follows:

VL2 = 2 * VL1

VL3 = 3 * VL1

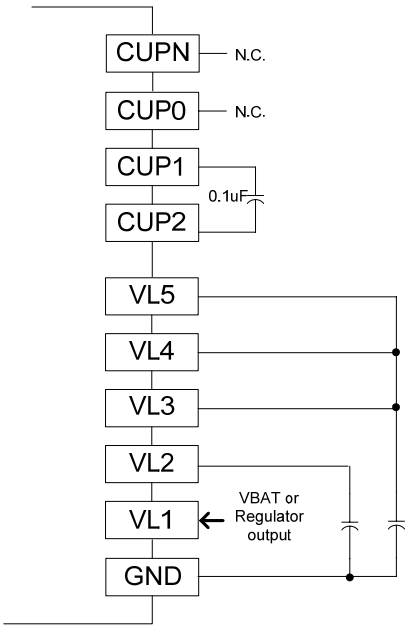
VL4 = 4 * VL1

VL5 = 5 * VL1

Below we will describe the system power application circuits used by the MCU for different LCD bias options when the VL1 voltage level is used as the charge pump circuit's base voltage.

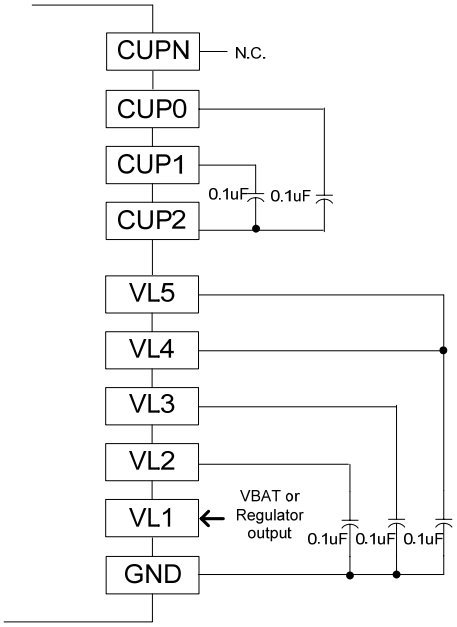
1-1-2-1-1. LCD Panel using 1/3 Bias in 1.5V power mode (standard size LCD panel)

LCD Panel's Vop = VL3 = 3 * VL1 (VL1 = VBAT or supplied by external voltage regulator)



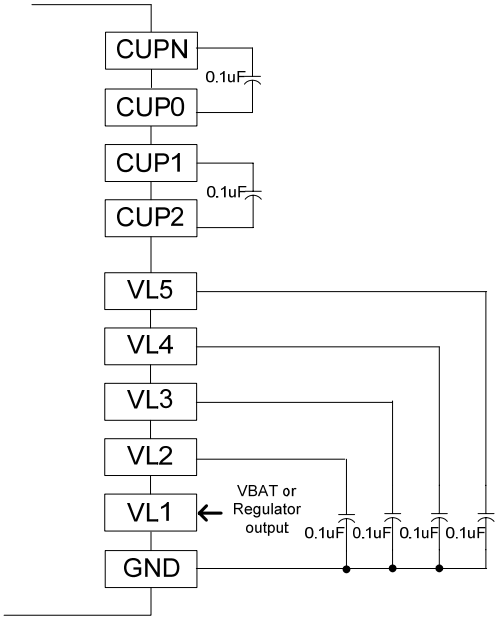
1-1-2-1-2. LCD Panel using 1/4 Bias in 1.5V power mode (standard size LCD panel)

LCD Panel's Vop = VL4 = 4 * VL1 (VL1 = VBAT or supplied by external voltage regulator)



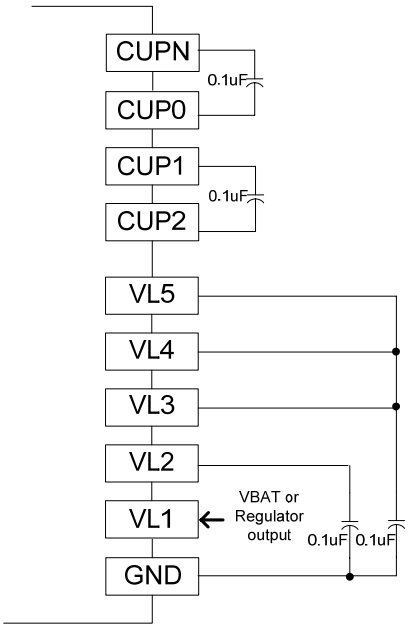
1-1-2-1-3. LCD Panel using 1/5 Bias in 1.5V power mode

LCD Panel's Vop = VL5 = 5 * VL1 (VL1 = VBAT or supplied by external voltage regulator)



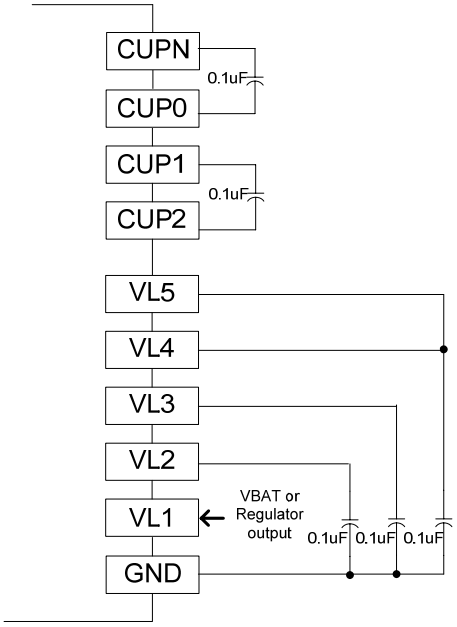
1-1-2-1-4. LCD Panel using 1/3 Bias in 1.5V power mode (Large LCD panel)

LCD Panel's Vop = VL3 = 3 * VL1 (VL1 = VBAT or supplied by external voltage regulator)



1-1-2-1-5. LCD Panel using 1/4 Bias in 1.5V power mode (Large LCD panel)

LCD Panel's Vop = VL4 = 4 * VL1 (VL1 = VBAT or supplied by external voltage regulator)



1-1-2-2. Using the Voltage Level at VL2 for the Charge Pump Circuit's Base Voltage

In 3V power mode where the VL2 is connected directly to VBAT or the voltage is supplied by an external voltage regulator, the charge pump circuit's base voltage is supplied directly by VL2.

The VL1, VL3 ~ VL5 voltage levels generated by the charge pump circuit will then be as follows:

VL1 = 1/2 * VL2

VL3 = 3/2 * VL2

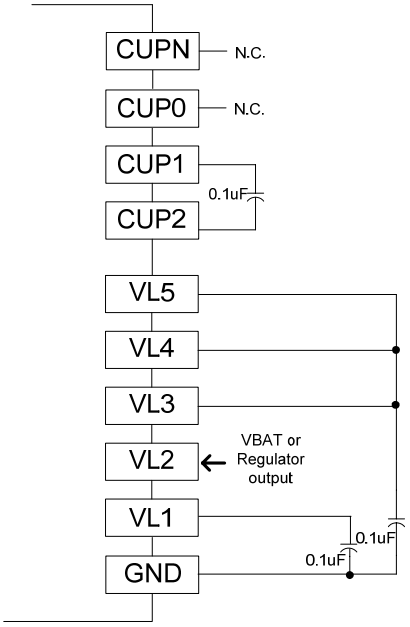
VL4 = 2 * VL2

VL5 = 5/2 * VL2

Below we will describe the system power application circuits used by the MCU for different LCD bias options when the VL2 voltage level is used as the charge pump circuit's base voltage.

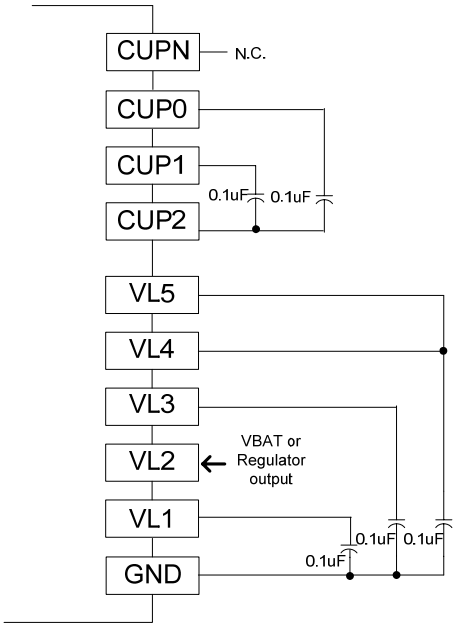
1-1-2-2-1. LCD Panel using 1/3 Bias in 3V power mode (standard size LCD panel)

LCD Panel's Vop = VL3 = 3/2 * VL2 (VL2 = VBAT or supplied by external voltage regulator)



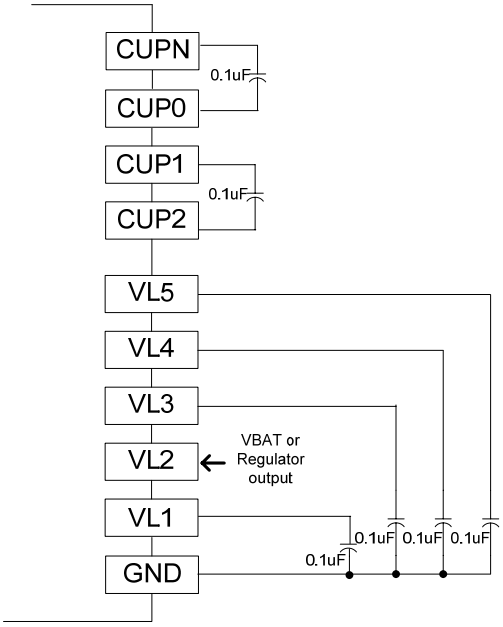
1-1-2-2-2. LCD Panel using 1/4 Bias in 3V power mode (standard size LCD panel)

LCD Panel's Vop = VL4 = 2 * VL2 (VL2 = VBAT or supplied by external voltage regulator)



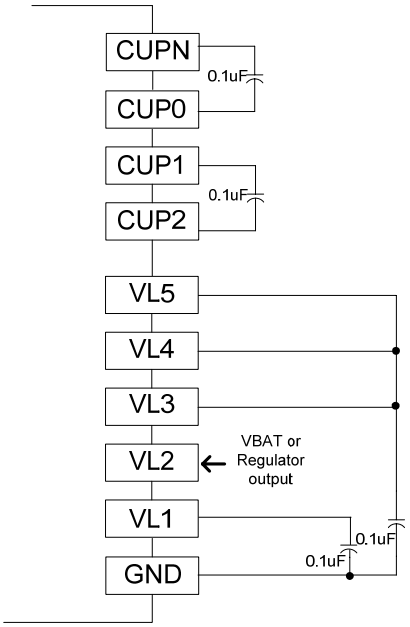
1-1-2-2-3. LCD Panel using 1/5 Bias in 3V power mode

LCD Panel's Vop = VL5 = 5/2 * VL2 (VL2 = VBAT or supplied by external voltage regulator)



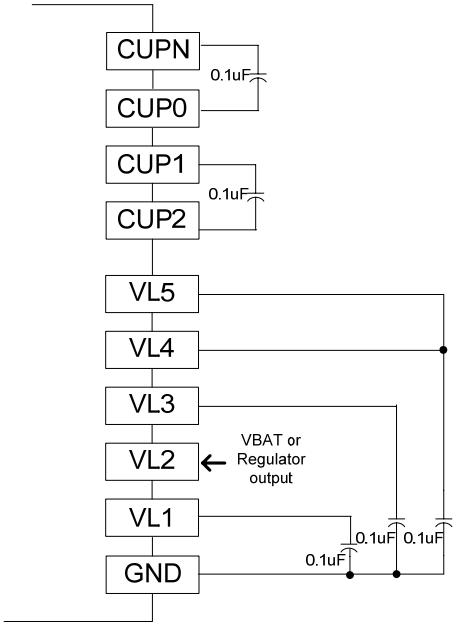
1-1-2-2-4. LCD Panel using 1/3 Bias in 3V power mode (Large LCD panel)

LCD Panel's Vop = VL3 = 3/2 * VL2 (VL2 = VBAT or supplied by external voltage regulator)



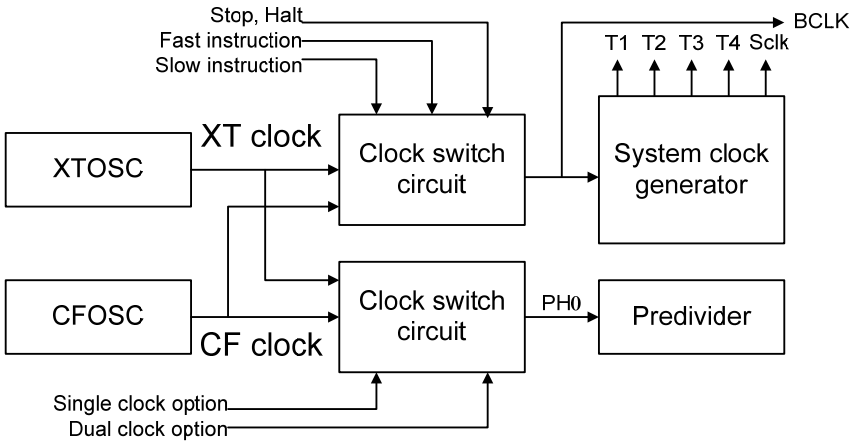
1-1-2-2-5. LCD Panel using 1/4 Bias in 3V power mode (Large LCD panel)

LCD Panel's Vop = VL4 = 2 * VL2 (VL2 = VBAT or supplied by external voltage regulator)



1-2. System Timing Control Unit

The System Timing Control Unit is made up of the slow oscillator (XTOSC), fast oscillator (CFOSC), the system clock generator, the pre-divider and the clock switch circuit. The System Timing Control Unit provides the different clock sources needed by the MCU to execute instructions and other peripheral devices. Its block diagram is as follows:



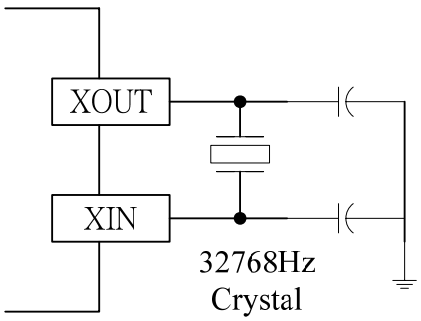
1-2-1. Slow Oscillator (XTOSC)

The XTOSC can provide a low-speed clock for the use of the system clock generator, pre-divider, timer, the IO port's chattering prevention function and LCD driver. When the MCU is powered on or a STOP release is generated by the MCU, the XTOSC begins operating and only stops after the STOP instruction is executed. If the "fast clock only option" mask option is selected for the MCU then the XTOSC will be disabled.

The TM89 Series MCU only offers two types of XTOSC – the external 32.768KHz Crystal Oscillator and the external RC oscillator. The user can use the mask option to select the required type.

1-2-1-1. External 32.768KHz Crystal Oscillator

Pictured below is the application circuit for the External 32.768KHz Crystal Oscillator:



(1) X'tal

On the PCB, the 32.768KHz Crystal component must be placed as close to the MCU IC as possible to avoid external noise from interfering with the operation of the oscillator.

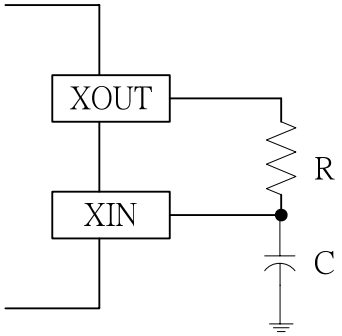
Setting the backup flag (BCF) to 1 will enable the crystal oscillator to output a more stable clock signal to against the power noise but this comes at the cost of increased MCU power consumption. Unless specifically required, please set the BCF flag to 0.

The table below shows the power consumption of the crystal oscillator in different power modes and scenarios.

	1.5V Power Mode	3V Power Mode
BCF=1	Power Intensive	Power Intensive
BCF=0	Power-Saving	Power-Saving
Initial reset	Power Intensive	Power Intensive
After reset	Power-Saving	Power-Saving

1-2-1-2. External RC oscillator

The diagram below is the application circuit for the external RC oscillator:



(2) RC

1-2-2. Fast Oscillator (CFOSC)

The CFOSC provides fast clock (CF clock) for use by the MCU and can be used in different ways under different operating modes. Described below are the 3 operating modes for CFOSC:

1. Fast Clock Only (MCU uses only CFOSC) mode:
   In this mode, the CF clock is provided for the use of the system clock generator, pre-divider, timer, IO port chattering prevention and LCD driver. When the MCU is powered on or a STOP release is generated by the MCU, the CFOSC begins operating and only stops after the STOP instruction is executed.

2. Dual Clock mode:
   In this mode, the CF clock is only provided for the system clock generator's use and only activates after the execution of the FAST instruction. After executing the FAST instruction, the system clock generator switches the clock source (BCLK) from XT clock to CF clock and goes to a fast instruction execution cycle.

   The CFOSC stops after the MCU enters HALT mode, STOP mode or executes the SLOW instruction. After the CFOSC shuts down, the MCU will automatically switch the clock source of the system clock generator from CF clock to XT clock.

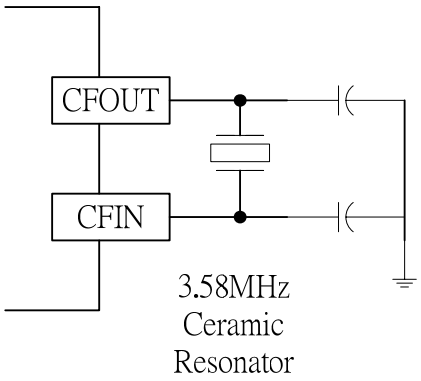
3. Slow Clock Only (MCU uses only XTOSC) mode:
   In this mode the CFOSC is always inactive.

When using the 3V power mode with the BCF flag cleared, the voltage at the BAK pin will be equal to the voltage at the VL1 pin. Since the VL1 is limited in how much current it can provide, to avoid problems with the MCU operation we recommend not executing program instructions at CF Clock speed as the BAK voltage will be unstable.

The TM89 Series MCU only offers three types of CFOSC – the external Ceramic Resonator, the RC oscillator with external resistor and the RC oscillator with internal resistor. The user can use the mask option to select the required type.

1-2-2-1. External Ceramic Resonator oscillator

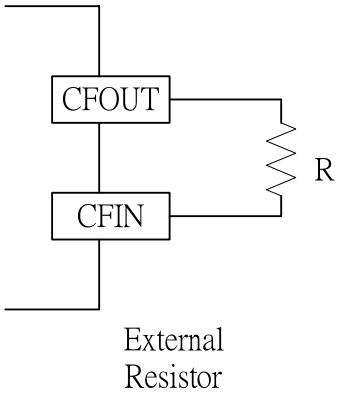
The diagram below shows how to use the external Ceramic Resonator oscillator:



On the PCB, the Ceramic Resonator component must be placed as close to the MCU IC as possible to avoid external noise from interfering with the operation of the oscillator.

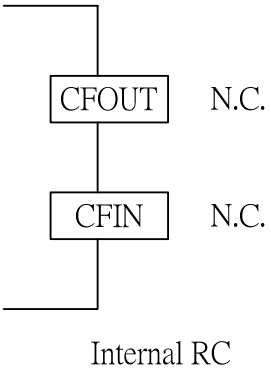
1-2-2-2. RC oscillator with External Resistor

The diagram below shows how to use the RC oscillator with external resistor:



1-2-2-3. RC oscillator with internal RC

The diagram below shows how to use the RC oscillator with internal resistor:



The RC oscillator with internal resistor can provide two different output frequencies: 250KHz and 500KHz. (The two frequencies are as measured at the BAK pin's voltage is 3V). The user can choose the frequency required in their design by using the mask option.

1-2-3. System Timing Control Unit's Operating Modes

The System Timing Control Unit can generate the clock signals for different timing of MCU operating modes. Two operating modes can be selected with the mask option: dual clock mode and single clock mode.

Each mode has its own state machine, all of them containing the five followings:

- RESET state: When power is turned on or the MCU receives a reset related signal, the MCU enters the RESET state.
- FAST state: Active both XTOSC and CFOSC (dual clock mode) or only activate the CFOSC (fast clock only mode).
- SLOW state: Activate only the XTOSC.
- HALT state: MCU enters HALT mode.
- STOP state: MCU enters STOP mode.

A detailed description of each operating mode's state machine is provided below.

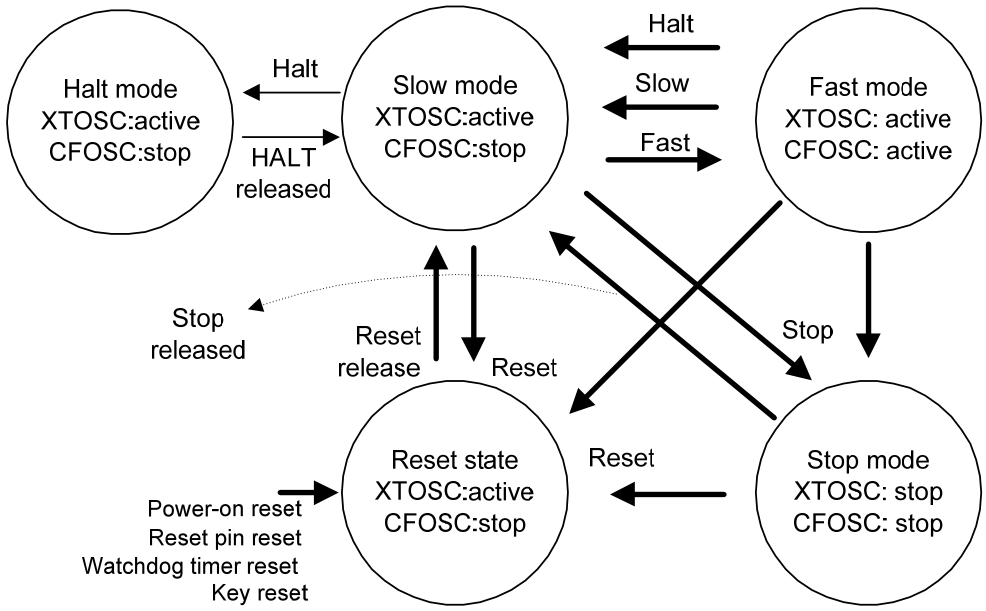
1-2-3-1. DUAL CLOCK Mode

In this mode, both the XTOSC and CFOSC can be activated or stopped. The system clock generator's clock source can choose between the XT clock or CF clock by using the clock switch control circuit.

XT clock can be used to provide the clock source needed by the system clock generator, pre-divider, timer, IO part chattering prevention and LCD driver. The CF clock can only be used to provide the clock source needed by the system generator.

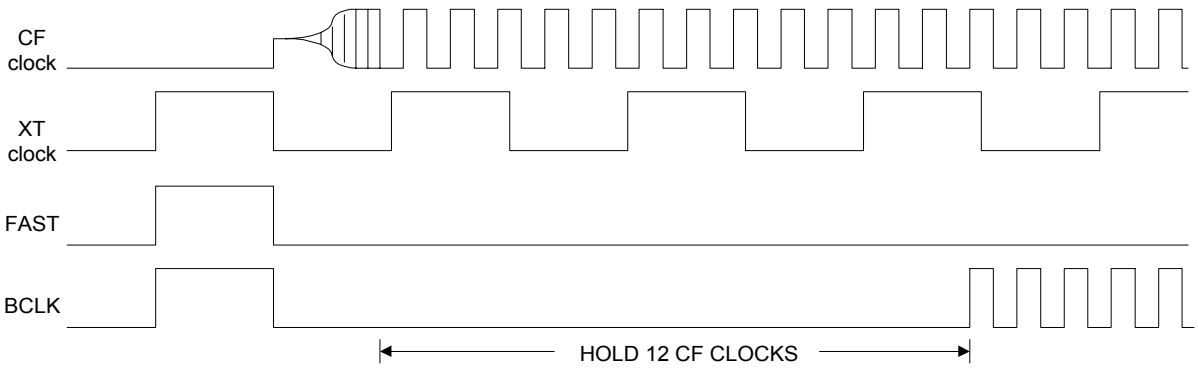
When the MCU generates a HALT release or STOP release, the system clock generator's clock source (BCLK) will automatically switch to the XT clock and stops the CFOSC. When the MCU enters the STOP mode, the backup flag (BCF) is automatically set to 1. This allows the MCU to successfully activate the XTOSC when generating a STOP release.

The diagram below sets out the state machine for the Dual Clock mode:



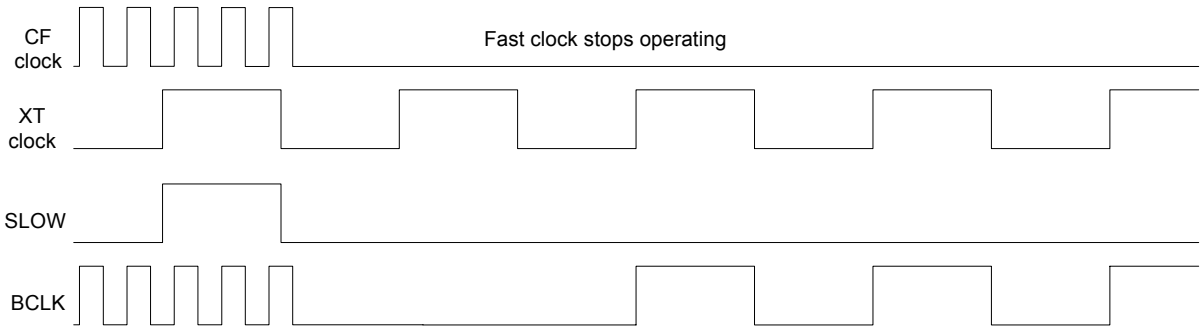
When the program issues the FAST instruction, the CFOSC becomes active but the system clock generator will wait until the CFOSC sends the 12 CF clock before switching the clock source (BCLK) form XT clock to CF clock. This will avoid any problems during the MCU switches the operating speed.

The diagram below explains how the BCLK switches from XT clock to CF clock:



When the program issues the SLOW instruction, the CFOSC stops but the system clock generator will wait until the XTOSC sends the 2nd XT clock before switching the clock source (BCLK) form CF clock to XT clock. This will avoid any problems during the MCU switches the operating speed.

The diagram below explains how the BCLK switches from CF clock to XT clock:



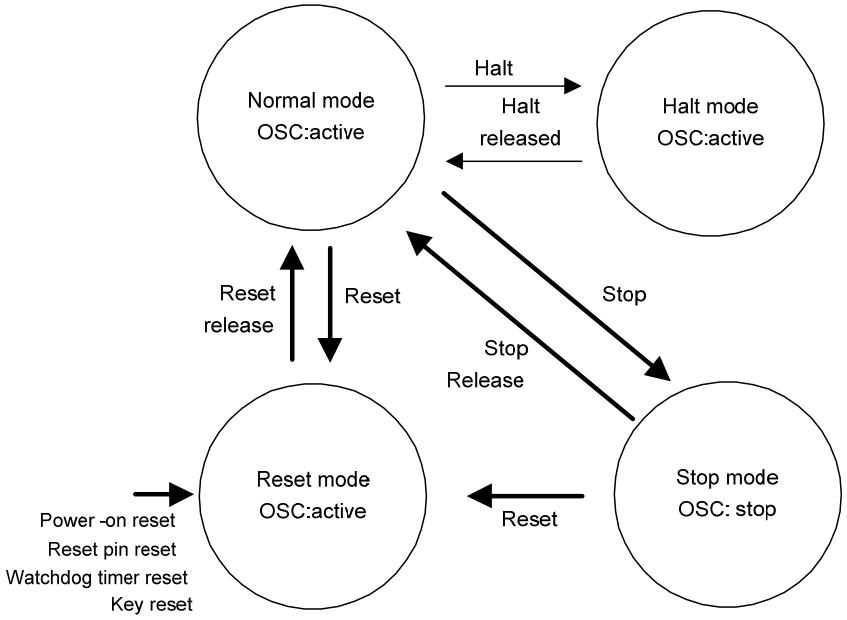
1-2-3-2. Single Clock Mode (Fast Clock Only or Slow Clock Only)

Slow Clock Only (XT clock) and Fast Clock Only (CF Clock) both come under the Single Clock Mode. The main difference is in the frequency ranges available to the pre-divider's clock source (PH0).

Executing either the FAST or SLOW instruction in this mode does not cause the MCU to switch its operating speed.

When the MCU enters the STOP mode, the backup flag (BCF) is automatically set to 1. This allows the MCU to successfully activate the clock oscillator when generating a STOP release.

The diagram below sets out the state machine for the Single Clock mode:



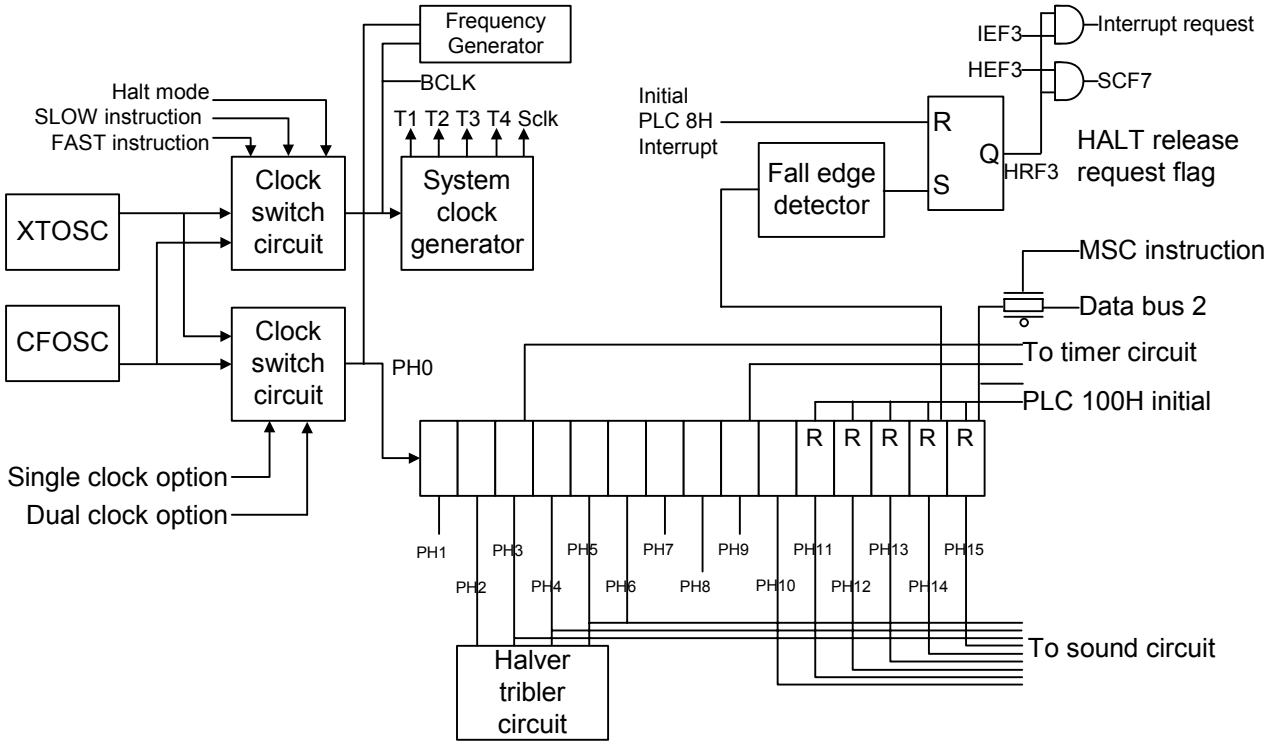
Transition Diagram of Single Clock Operation

1-2-4. PRE-DIVIDER

The Pre-divider is a 15-bit counter. If given an input signal with frequency N from the clock source (PH0), the pre-divider can generate at its output a signal with frequencies ranging between N/2(PH1) ~ N/2^15(PH15). Each output signal will change its signal level on the falling edge of the PH0 signal. These different clock frequencies can be provided for the

use of peripheral devices such as the charge-pump circuitry, LCD driver, frequency generator, timers, interrupt control circuit, HALT released control and the IO port chattering prevention function.

The diagram below shows the layout of the pre-divider and its associated peripheral devices:



The last 5 stages of the Pre-divider's output signals (PH11 ~ PH15) will be reset to 0 when the MCU enters the RESET mode or executes the PLC 100H instruction.

Each falling edge from the PH14 output signal will set the HALT Release Request Flag 3 (HRF3) to 1. Only when the MCU enters the RESET mode, executes the PLC $8 instruction or there is an interrupt, does the Halt Released Request Flag (HRF3) gets cleared.

If the pre-divider Interrupt Enable Flag 3 (IEF3) was already set to 1, when the HRF3 is set to 1 then the pre-divider will generate an interrupt request. When the MCU accepts the interrupt request it will execute the corresponding interrupt service; if the Halt Release Enable Flag 3 (HEF3) was already set to 1, the pre-divider will also send a Halt Release Request signal to the MCU to make the MCU generate a HALT release. At the same time, the Start Condition Flag 7 (SCF7) in the Status Register 3 (STS3) will be set to 1.

When the pre-divider's clock source (PH0) frequency is 32768HZ, PH14 will set the HRF3 flag to 1 every 0.5 seconds. This can be used as the reference signal for the real-time clock.

When the Fast Clock Only mode is selected, the pre-divider might not be able to generate the lower frequency signal needed by the LCD driver because the CF Clock frequency is too high. In this case the MCU should perform frequency division on the BCLK signal (CF Clock) frequency before supplying it to the pre-divider's clock source (PH0). The mask option can be used to choose between the divided frequencies:

The available frequency divisions are listed in the table below:

Mask Option name	Selected item
PH0 <-> BCLK FOR FAST ONLY	(1) PH0 = BCLK
PH0 <-> BCLK FOR FAST ONLY	(2) PH0 = BCLK/4
PH0 <-> BCLK FOR FAST ONLY	(3) PH0 = BCLK/8
PH0 <-> BCLK FOR FAST ONLY	(4) PH0 = BCLK/16
PH0 <-> BCLK FOR FAST ONLY	(5) PH0 = BCLK/32
PH0 <-> BCLK FOR FAST ONLY	(6) PH0 = BCLK/64
PH0 <-> BCLK FOR FAST ONLY	(7) PH0 = BCLK/128
PH0 <-> BCLK FOR FAST ONLY	(8) PH0 = BCLK/256

The table below shows the clock sources available to the pre-divider's clock source (PH0) under different conditions:

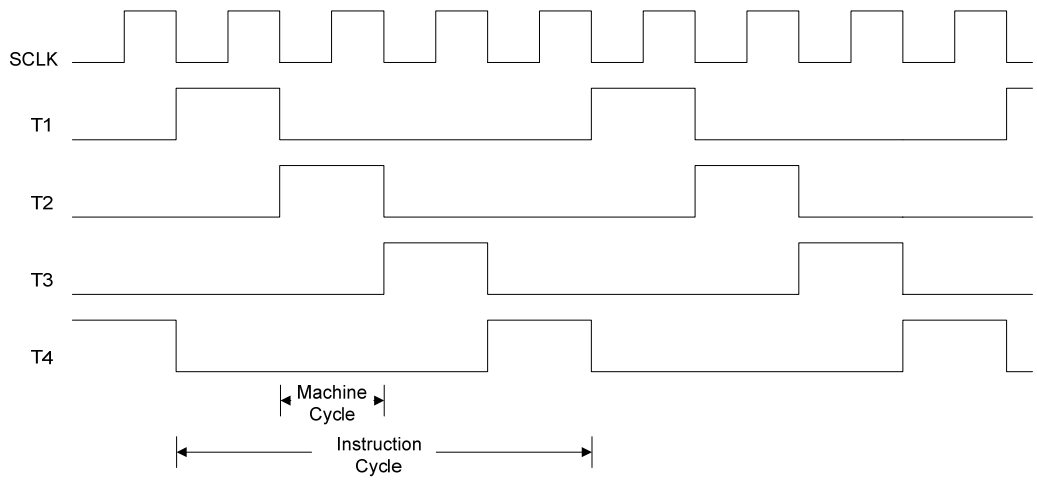
Condition	PH0
Slow clock only option	XT clock
Fast clock only option	CF clock
RESET state (dual clock option)	XT clock
Halt mode (dual clock option)	XT clock
Slow mode (dual clock option)	XT clock
Fast mode (dual clock option)	XT clock

1-2-5. System Clock Generator

The System Clock Generator can generate the different timings needed by the MCU for executing instructions. For the TM89 Series MCUs, most instruction cycles can be completed with 4 machine cycles though some instructions require 8 machine cycles to complete.

Each machine cycle corresponds to the clock cycle of the XT clock (Slow Clock Mode) or the CF clock (Fast Clock Mode).

The diagram below shows the basic instruction execution timing:



When the program executes the FAST or SLOW instruction, it switches the clock source of the system clock generator between the XT clock and CF clock, changing the operating speed. The table below shows the clock source used by the system clock generator under different conditions.

Condition	BCLK
Slow clock only option	XT clock
Fast clock only option	CF clock
RESET state(dual clock option)	XT clock
Halt mode(dual clock option)	XT clock
Slow mode(dual clock option)	XT clock
Fast mode(dual clock option)	CF clock

1-3. PROGRAM COUNTER (PC)

The Program Counter is made up of a 16-bit binary counter (smaller counters might be used by MCU with a different specification). It can be used to address the Program Memory (ROM) for addresses up to 62K (PC0 ~ PC15). The higher significant 5 bits (PC15 ~ PC11) serve as the bank register of the program memory to indicate the program's bank location. The rest of bits (PC10 ~ PC0) are used to indicate the address of the program in the bank itself.

The Program Counter operates in the following manner:

1. When an one-word instruction finishes execution, it automatically increments the current PC value by 1.

   PC ← PC + 1

2. When a two-word instruction finishes execution, it automatically increments the current PC value by 2.

   PC ← PC + 2

3. When the program executes a JUMP instruction, CALL instruction, Interrupt Servicing Routine or the MCU enters the RESET state, it loads the corresponding specified addresses in Table 1-1 into the Program Counter.
   If the loaded address differs from the current bank location, the TM89 ICE compiler program will automatically insert a SPBK instruction in front of that instruction to change the bank location to that of loaded address.

   PC ← specified address shows in table 1-1

4. When the program executes the JAC instruction, the address specified by the AC value is loaded into the Program Counter; if the AC value exceeds the defined value X (X is the operand in the JAC instruction), load the address of the next instruction into the Program Counter.

   If(AC<=X) PC ← Address specified in instruction (PC+AC+1)

   else PC ← Address of next instruction (PC+X+2; X is the operand in the JAC instruction)

5. After executing the RTS instruction, load the content of the STACK pointed to by the stack pointer to the Program Counter.

   (a). Stack pointer ← stack pointer – 1

   (b). PC ← The content of the STACK pointed to by the stack pointer

6. When the program executes the CALL instruction, increment the content of the Program Counter by 1 then store it to the STACK.

   (a). Current STACK location pointed to by the Stack pointer ← PC + 1

   (b). Stack pointer ← stack pointer + 1

7. When the MCU accepts an interrupt request, store the current content of the program counter directly to STACK.

   (a). Current STACK location pointed to by the Stack pointer ← PC

   (b). Stack pointer ← stack pointer + 1

8. When the program executes the CAC instruction, increment the content of the Program Counter by X+2 (X is the CAC instruction's operand) then store it to STACK. The specified address indicated by the AC value is then loaded into the Program Counter; if the AC value is greater than X, the address which comes after CAC instruction is loaded into the Program Counter.

   If (AC <=X)

   (a). Current STACK location pointed to by the Stack pointer ← PC + X + 2

   (b). Stack pointer ← stack pointer + 1

   (c). PC ← Address specified in instruction (PC+AC+1)

   Else if (AC >X)

   PC ← Address of next instruction (PC+X+2; X is the CAC instruction's operand)

table 1- 1

	PC15	PC14	PC13	PC12	PC11	PC	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
Initial reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Interrupt 2																
(INT pin)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Interrupt 0																
(input port A, C or D)	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
Interrupt 1																
(timer 1 interrupt)	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
Interrupt 3																
(pre-divider interrupt)	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
Interrupt 4																
(timer 2 interrupt)	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Interrupt 5																
(Key Scanning interrupt)	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
Interrupt 6																
(RFC counter interrupt)	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
Interrupt 7																
(timer3 counter interrupt)	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0
JAC,CAC instruction	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
CALL& others Jump instruction	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Notes:

1. D15 ~ D0: Can be set directly using the "SETDAT" instruction.
2. P15 ~ P0: Can be set directly using the instruction operand.

1-3-1. Using the JAC Instruction

The JAC instruction allows the program to choose from up to 16 different destination addresses. The actual destination address chosen depends on the current value of AC.

The JAC instruction is a multi-word instruction that may require 4 (AC>X) or 8 (AC<=X) machine cycles to complete. The syntax for JAC is as follows:

```
Address     OPcode      Operand
PC          JAC           X          ; X=0~15
PC+1        SETDAT      label_0      ; if AC=0, label_0 is destination
PC+2        SETDAT      label_1      ; if AC=1, label_1 is destination
-------------------------------------------------------------
PC+X+1      SETDAT      label_x      ; if AC=X, label_x is destination
PC+X+2      (Next Instruction)       ; if AC>X
```

The instruction‘s operand X indicates that the JAC instruction offers X+1 destination addresses to choose from.

When AC<=X, the MCU will load the destination address corresponding to the current AC value to the Program Counter. In this case the program will need 8 machine cycles to complete the JAC instruction. During these 8 machine cycles the MCU will automatically suspend all interrupt requests.

When AC>X, there is no destination address corresponding to the AC value to choose from. The MCU will then load the address which comes after JAC instruction (Current PC value + X + 2) into the Program Counter. In this situation, the program will only need 4 machine cycles to complete the JAC instruction. During these 4 machine cycles the MCU will automatically suspend all interrupt requests.

Example:

```
LDS         $20,$0
loop:       JAC     5
SETDAT      label_0         ; jump to label_0 if AC=0
SETDAT      label_1         ; jump to label_1 if AC=1
SETDAT      label_2         ; jump to label_2 if AC=2
SETDAT      label_3         ; jump to label_3 if AC=3
SETDAT      label_4         ; jump to label_4 if AC=4
SETDAT      label_5         ; jump to label_5 if AC=5
INC*        $20             ; if AC > 5, AC = AC+1
JMP         loop            ;
```

1-3-2. Using the CAC Instruction

The CAC instruction allows the program to choose from up to 16 different routine addresses. The actual vector called depends on the current value of AC.

The CAC instruction is a multi-word instruction that may require 4 (AC>X) or 8 (AC<=X) machine cycles to complete. The syntax for CAC is as follows:

```
Address     OPcode      Operand
PC          CAC           X          ; X=0~15
PC+1        SETDAT        label_0    ; if AC=0, label_0 subroutine is called
PC+2        SETDAT        label_1    ; if AC=1, label_1 subroutine is called
-------------------------------------------------------------
PC+X+1      SETDAT        label_x    ; if AC=X, label_x subroutine is called
PC+X+2      (Next Instruction)       ; return or if AC>X
```

The instruction‘s operand X indicates that the CAC instruction offers X+1 vectors to call from.

When AC<=X, the MCU will first store the address which comes after the CAC instruction (current PC value + X + 2) to Stack then load the vector corresponding to the current AC value to the Program Counter. In this situation, the program will need 8 machine cycles to complete the CAC instruction. During these 8 machine cycles the MCU will automatically suspend all interrupt requests.

When AC>X, there is no vector corresponding to the AC value to choose from. The MCU will then load the address which comes after the CAC instruction (Current PC value + X + 2) into the Program Counter. In this situation, the program will only need 4 machine cycles to complete the CAC instruction. During these 4 machine cycles the MCU will automatically suspend all interrupt requests.

Example:

```
LDS         $20,$0          ; initial AC=0
loop:       CAC       5
SETDAT      label_0         ; jump to label_0 if AC=0, and return to label_x
SETDAT      label_1         ; jump to label_1 if AC=1, and return to label_x
SETDAT      label_2         ; jump to label_2 if AC=2, and return to label_x
SETDAT      label_3         ; jump to label_3 if AC=3, and return to label_x
SETDAT      label_4         ; jump to label_4 if AC=4, and return to label_x
SETDAT      label_5         ; jump to label_5 if AC=5, and return to label_x
label_x INC*        $20     ; return or if AC > 5, AC = AC+1
JMP         loop            ;
```

1-4. PROGRAM MEMORY AND TABLE ROM

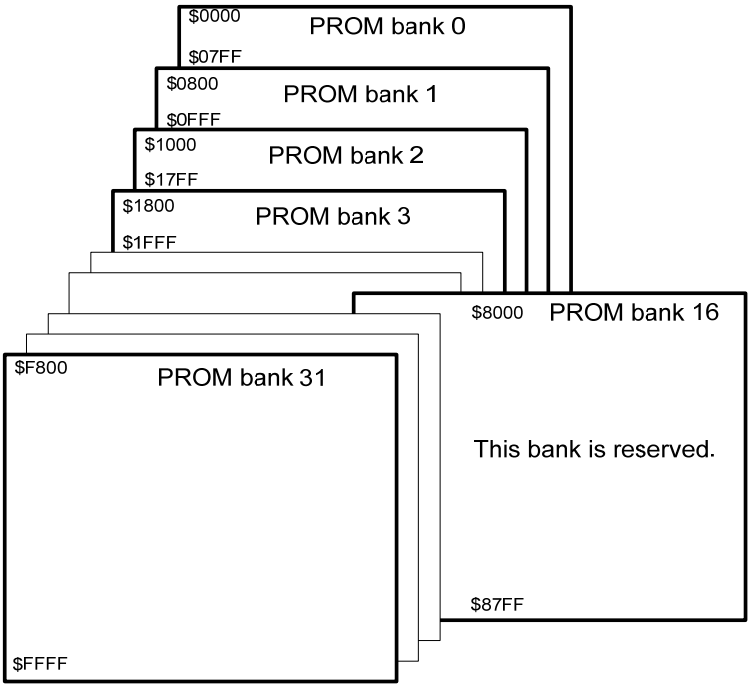
The TM89 Series MCU's Program memory can be as large as 64K x 16 bits. It is a Read Only Memory (ROM) used for storing program instructions and the look-up table. The 64K program memory is partitioned into 32 banks with each bank containing 2048 addresses.

The TM89 Series MCU also supports the look up table function, referred to as the table ROM. The table ROM actually shares the same physical memory as the program memory. Up to 60K x 8 bits can be defined for the table ROM. If the MCU allocates some of the program memory to the table ROM, this results in a corresponding decrease in the amount of program memory available to the MCU.

The relative sizes memory used by the table ROM and program memory varies according to MCU specifications. Please refer to each MCU's data sheet to determine the amount of usable memory.

1-4-1. PROGRAM MEMORY

The diagram below shows the paging method for each bank of the program memory:



The program memory's 16th bank is reserved and not available for use by programs.

If the address of a program's instruction in the program memory is at the last address of a particular bank, once the instruction finishes executing, the program counter will automatically point to the first address of the next bank. The user does not have to worry about the problem of their program spanning different banks.

Among the jump or subroutine call instructions, only the JAC and CAC instructions can directly address the entire memory space so no bank adjustments are needed. Apart from the above, the bit length restriction on the Opcode of other jump or call instructions means

that they can only directly specify addresses that are in the same bank of the program memory.

If these instructions need to directly address the entire program memory space, an absolute address or label must be used to specify the destination address or vector. In this case, when the compiler is translating the source code it will check if the destination or vector is in the same bank as the current instruction address. If not, the compiler will insert a SPBK instruction to adjust the program counter's bank value.

The MCU will suspend all interrupt requests while executing the SPBK instruction and the instruction comes after SPBK to avoid setting the bank incorrectly.

For example:

The source file is as below:

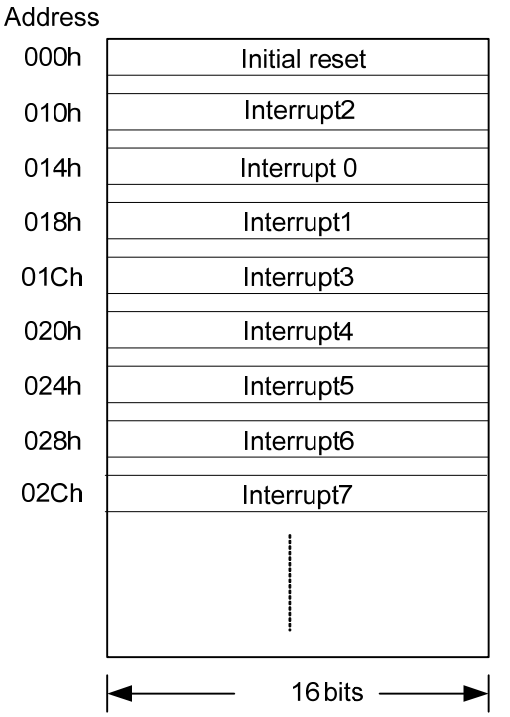
```
ORG         $0130           ; bank 0
CALL        label1          ; bank 0
…
…
ORG         $0F45           ; bank 1
Label1:
DAA                         ; bank 1
```

The sources file after compiling:

```
SPBK        1               ; set bank 1 by compiler
CALL        0F45h           ; in bank 0
…
…
DAA                         ; bank 1, PC= 0F45h
```

In bank 0 of the program memory, certain addresses are reserved for use by the MCU's interrupt service routines such as: reset address (000H), interrupt 0 address (014H), interrupt 1 address (018H), interrupt 2 address (010H), interrupt 3 address (01CH), interrupt 4 address (020H), interrupt 5 address (024H), interrupt 6 address (028H), and interrupt 7 address (02CH).

The diagram below sets out the relative addresses of these interrupt handlers.



Instruction ROM (PROM) organization

1-4-2. TABLE MEMORY (TROM)

Table ROM can be used to store some constants or look up tables used by the programs. All Table ROM data can only be accessed using the HL index register (@HL) for indexed addressing.

Table ROM offers data access in 3 data lengths: 4 bits, 8 bits and 16 bits. These read operations can all be completed within 4 machine cycles.

- LDL(*), LDH(*) related instructions can read 4 bits from the Table ROM
- LCD related instructions can read 8 bits from the Table ROM
- LCDH related instructions can read 16 bits from the Table ROM

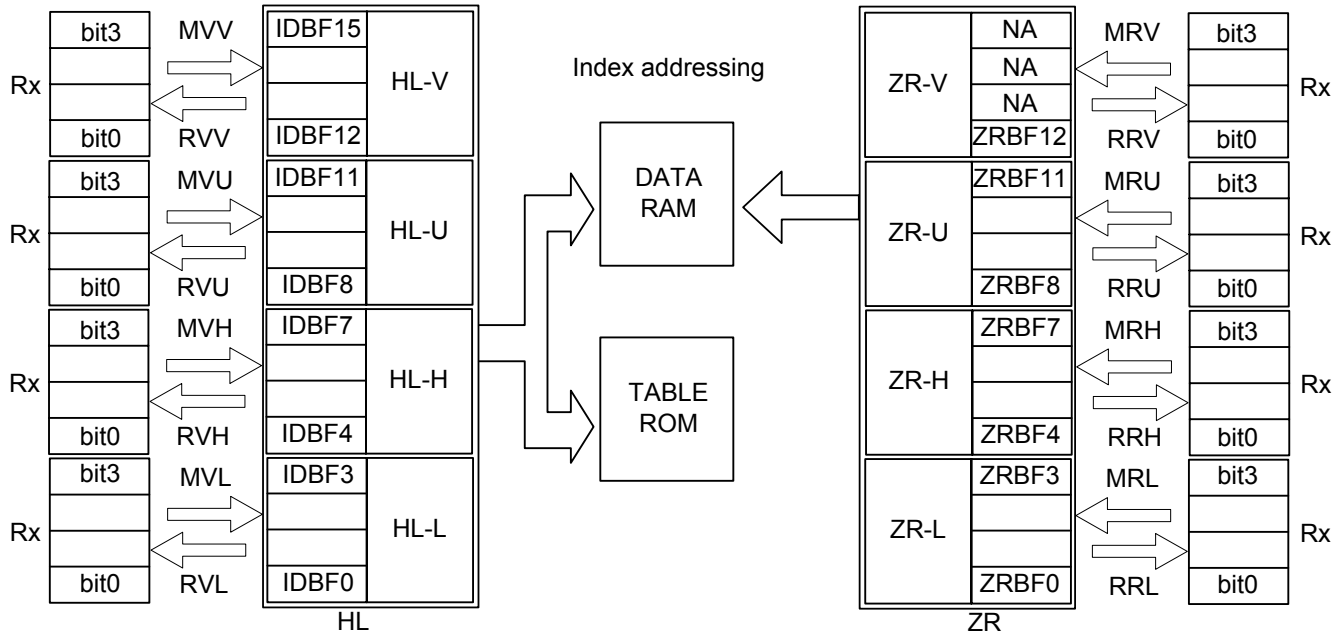
1-5. INDEX REGISTER (HL and ZR)

The TM89 Series MCU offers two types of index registers (HL and ZR) for indexed addressing of the data RAM and Table ROM.

The HL index register is a 16-bit register (may be smaller than 16-bit in some MCU specifications) used for indexed addressing (@HL) of the data RAM and table ROM. The HL index register can be further broken down into four sub-registers: HL-V, HL-U, HL-H, HL-L. IDBF15 ~ IDBF0 is then used to represent each bit in the HL index register.

The ZR index register is a 13-bit register (may be smaller than 13-bit in some MCU specifications) that can only be used for indexed addressing (@ZR) of the data RAM. The ZR index register can be further broken down into four sub-registers: ZR-V, ZR-U, ZR-H, ZR-L. ZRBF12 ~ ZRBF0 is then used to represent each bit in the ZR index register.

The diagram below shows the relationship between the two index registers and their sub-registers:



1-5-1. HL INDEX REGISTER

The HL index register is a 16-bit register (may be smaller than 16-bit in some MCU specifications) made up of four sub-registers: HL-V, HL-U, HL-H and HL-L.

The contents of the HL index register can be updated or copied using three different types of access instructions – instructions that write directly to the register as immediate data, instructions that use the content of data RAM for data access, and instructions that automatically increment the register by 1/2/4 after execution.

1-5-1-1. Updating the content of HL using IMMEDIATE DATA

The SHLX instruction operand's immediate data can be used to simultaneously update all 16 bits of the HL index register.

1-5-1-2. Updating or Backing Up the content of HL using DATA RAM

The contents of the HL index register can be updated or backed up 4 bits or 16 bits at a time using the contents of data RAM.

When using 4 bits data update or backup, executing instructions such as MVV, MVU, MVH and MVL writes the content of the data RAM to the corresponding HL-V, HL-U, HL-H and HL-L sub-registers.

Executing instructions such as RVV, RVU, RVH and RVL writes the contents of sub-registers such as HL-V, HL-U, HL-H and HL-L back to the specified data RAM address.

The table below explains the relationship between the sub-registers and the data RAM bits:

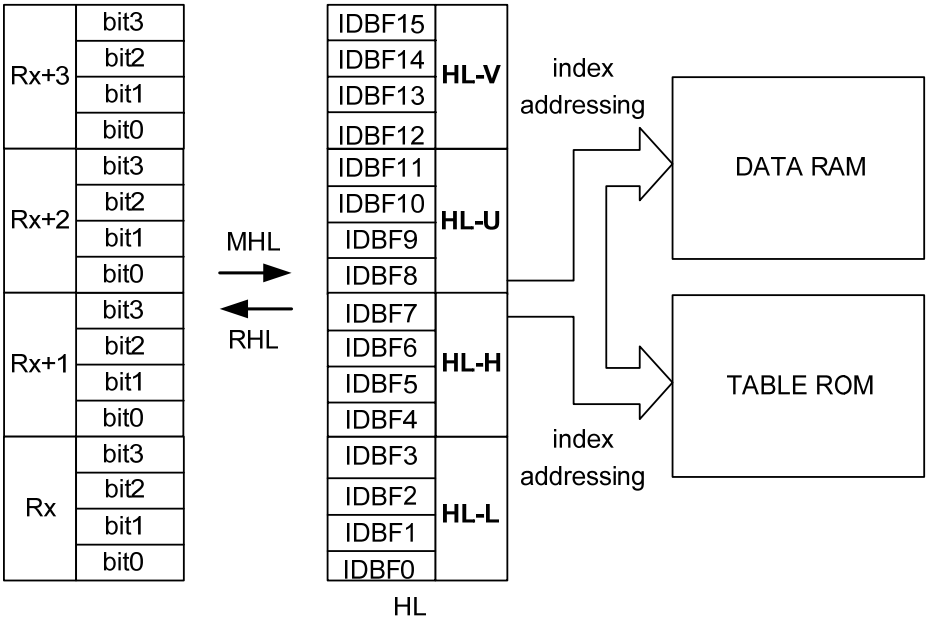
MVV Rx or RVV Rx				MVU Rx or RVU Rx				MVH Rx or RVH Rx				MVL Rx or RVL Rx			
(Rx)3	(Rx)2	(Rx)1	(Rx)0	(Rx)3	(Rx)2	(Rx)1	(Rx)0	(Rx)3	(Rx)2	(Rx)1	(Rx)0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
HL-V register				L-U register				HL-H register				HL-L register			
IDBF15	IDBF14	IDBF13	IDBF12	IDBF11	IDBF10	IDBF9	IDBF8	IDBF7	IDBF6	IDBF5	IDBF4	IDBF3	IDBF2	IDBF1	IDBF0

When using 16-bit data update or backup, the TM89 Series MCU will allocate 64 specific data RAM addresses (0080 ~ 00BFH) as the HL index register's 16-bit data backup area. This provides 16 sets of data backup units with 4 contiguous addresses per sets. The MHL X or RHL X instructions (X=0~FH) can be used to transfer all 16 bits data between a set of data backup unit and HL index register in 4 machine cycles.

Executing the MHL instruction updates the contents of 4 contiguous addresses from the data RAM to the 16-bit HL index register. Executing the RHL instruction simultaneously backups all 16-bits of the HL index register's content to 4 contiguous addresses in the data RAM.

When using the MHL and RHL instructions, the operand X will point to the 4 contiguous data RAM addresses. See 1-7-2 for how they correspond to each other.

The table below sets out the relationship between the bits for 16-bit data transfers between the HL index and data RAM:



1-5-1-3. Automatically increment HL after executing instruction

The index addressing mode instructions can be used to simultaneously access the contents of 1, 2 or 4 contiguous addresses from the data RAM and table ROM. Once these instructions finish executing, some will automatically increment the value of the HL index register by 1, 2 or 4.

1-5-1-4. Using the HL index register with Compare-then-Mask instructions

Apart from index addressing, the HL index register can also be used for condition testing by "Compare-then-Mask" instructions such as CPHL and CPHLH.

The CPHL X instruction can specify an 8-bit immediate data (X) and upon execution, the MCU will compare the content of HL-H and HL-L with X to check if they are equal. If equal, during the next instruction cycle regardless of what instruction comes after CPHL was, the MCU would execute the NOP instruction instead(*Note*); if they were not equal, the MCU will execute the instruction comes after CPHL normally in the next instruction cycle.

Note: The number of NOP instruction cycle relative to each instruction is different.

1. *Instructions with single instruction cycle will generate one NOP instruction cycle.*
2. *Instructions with double instruction cycles will generate two NOP instruction cycles.*
3. *CAC X, JAC X instructions will generate one NOP instruction cycle. After this NOP instruction, the value of PC will jump to PC+X+2 address directly.*
4. *ERX, ERY, ELZ and CLPG instructions will not generate NOP instruction cycle, but will still execute the original instruction.*

The MCU will automatically suspend all interrupt requests while executing CPHL and the next instruction until the two instruction cycles have been completed.

The table below explains the relationship between X and HL register for comparison:

CPHL X	X7	X6	X5	X4	X3	X2	X1	X0
Content of HL	IDBF7	IDBF6	IDBF5	IDBF4	IDBF3	IDBF2	IDBF1	IDBF0
Sub-register	@HL-H				@HL-L			

Example:

```
        ……                  ; @HL = 30h
        CPHL        $30h
        JMP         label1  ; NOP instead of this jump instruction
        JMP         label2  ; this instruction will be executed and then jump to label2
        ……
label1:
        ……..
label2:
```

CPHLH is a two-word instruction that takes 8 machine cycles to complete. This instruction can define a 16-bit immediate data (X) and upon execution, the MCU will compare the value of HL index register with X to check if they are equal. If equal, during the next instruction cycle regardless of what instruction comes after CPHLH was, the MCU will

execute the NOP instruction instead(refer to the note in 1-5-1-4); if they were not equal, the MCU will execute the instruction comes after CPHLH normally in the next instruction cycle.

The MCU will automatically suspend all interrupt requests while executing CPHLH and the next instruction until both instruction cycles have been completed.

The table below explains the relationship between X and HL registers for comparison:

CPHLH	X15 ~ X12	X11 ~ X8	X7 ~ X4	X3 ~ X0
HL	IDBF15 ~ IDBF12	IDBF11 ~ IDBF8	IDBF7 ~ IDBF4	IDBF3 ~IDBF0

Example:

```
        ……                  ; @HL = 1234h
        CPHLH
        SETDAT    $1234h
        JMP       label1    ; NOP instead of this jump instruction
        JMP       label2    ; this instruction will be executed and then jump to label2
        ……
label1:
        ……
label2:
```

1-5-2. ZR INDEX REGISTER

The ZR index register is a 13-bit register (may be smaller than 13-bit in some MCU specifications) made up of four sub-registers: ZR-V, ZR-U, ZR-H and ZR-L.

The contents of the ZR index register can be updated or copied using three different types of access instructions – instruction that writes directly to the register as immediate data, instruction that uses the content of data RAM for data access, and instruction that automatically increments the register by 1/2/4 after execution.

1-5-2-1. Updating the content of ZR using IMMEDIATE DATA

The SZRX instruction operand's immediate data can be used to simultaneously update all 13 bits of the ZR index register.

1-5-2-2. Updating or Backing Up the content of ZR using DATA RAM

The contents of the ZR index register can be updated or backed up 4 bits or 13 bits at a time using the contents of data RAM.

When using 4 bits data update or backup, executing instructions such as MRV, MRU, MRH and MRL writes the content of the data RAM to the corresponding ZR-V, ZR-U, ZR-H and ZR-L sub-registers.

Executing instructions such as RRV, RRU, RRH and RRL writes the contents of sub-registers such as ZR-V, ZR-U, ZR-H and ZR-L back to the specified data RAM address.

The table below explains the relationship between the sub-registers and the data RAM bits:

MRV Rx or RRV Rx				MRU Rx or RRU Rx				MRH Rx or RRH Rx				MRL Rx or RRL Rx			
(Rx)3	(Rx)2	(Rx)1	Rx0	(Rx)3	(Rx)2	(Rx)1	(Rx)0	(Rx)3	(Rx)2	(Rx)1	(Rx)0	(Rx)3	(Rx)2	(Rx)1	(Rx)0

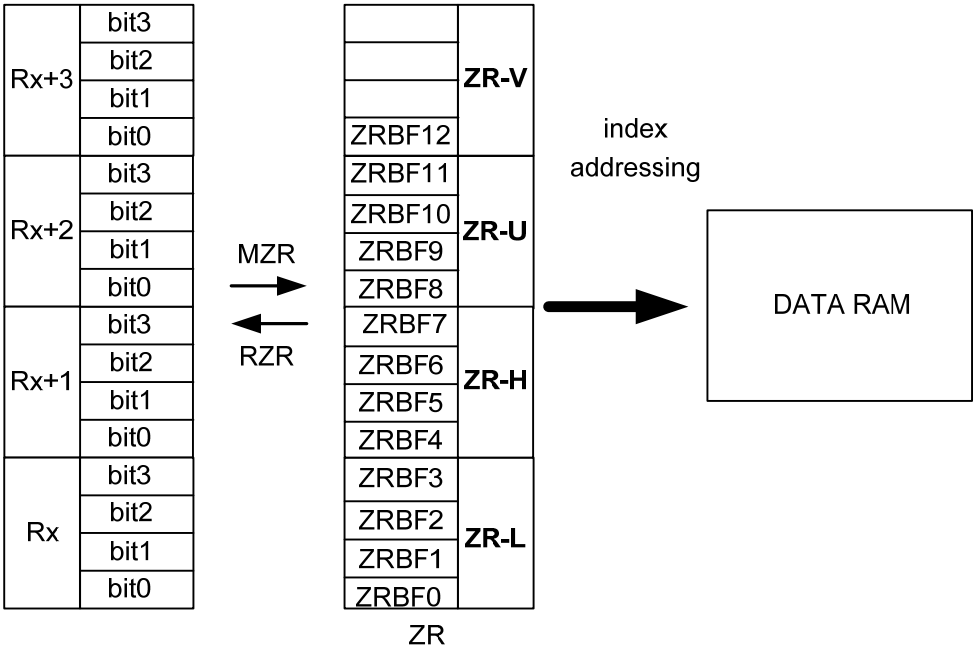
ZR-V register				ZR-U register				ZR-H register				ZR-L register			
NA	NA	NA	ZRBF12	ZRBF11	ZRBF10	ZRBF9	ZRBF8	ZRBF7	ZRBF6	ZRBF5	ZRBF4	ZRBF3	ZRBF2	ZRBF1	ZRBF0

When using 13-bit data update or backup, the TM89 Series MCU will allocate 64 specific data RAM addresses (00C0 ~ 00FFH) as the ZR index register's 13-bit data backup area. This provides 16 sets of data backup units with 4 contiguous addresses per sets. The MZR X or RZR X instructions (X=0~FH) can be used to transfer all 13 bits data between a set of data backup unit and ZR index register in 4 machine cycles.

Executing the MZR instruction writes the contents of 4 contiguous addresses from the data RAM to the 13-bit ZR index register. Executing the RZR instruction simultaneously writes all 13-bits of the ZR index register's content to 4 contiguous addresses in the data RAM.

When using the MZR and RZR instructions, the operand X will point to the 4 contiguous data RAM addresses. *See 1-7-2 for how they correspond to each other*.

The table below sets out the relationship between the bits for 13-bit data transfers between the ZR index and data RAM:



1-5-2-3. Automatically increment ZR after executing instruction

The index addressing mode instructions can be used to simultaneously access the contents of 1, 2 or 4 contiguous addresses from the data RAM. Once these instructions finish executing, some will automatically increment the value of the ZR index register by 1, 2 or 4.

1-5-2-4. Using the ZR index register with Compare-then-Mask instructions

Apart from index addressing, the ZR index register can also be used for condition testing by "Compare-then-Mask" instructions such as CPZR and CPZRH.

The CPZR X instruction can specify an 8-bit immediate data (X) and upon execution, the MCU will compare the contents of ZR-H and ZR-L with X to check if they are equal. If equal, during the next instruction cycle regardless of what instruction comes after CPZR

was, the MCU would execute the NOP instruction instead(refer to the note in 1-5-1-4); if they were not equal, the MCU will execute the instruction comes after CPZR normally in the next instruction cycle.

The MCU will automatically suspend all interrupt requests while executing CPZR and the next instruction until the two instruction cycles have been completed.

The table below explains the relationship between X and ZR registers for comparison:

CPZR X	X7	X6	X5	X4	X3	X2	X1	X0
Content of ZR	ZRBF7	ZRBF6	ZRBF5	ZRBF4	ZRBF3	ZRBF2	ZRBF1	ZRBF0
Sub-register	ZR-H				ZR-L			

Example:

```
        ……                  ; @ZR = 30h
        CPZR        $30h
        JMP         label1    ; NOP instead of this jump instruction
        JMP         label2    ; this instruction will be executed and then jump to label2
        ……
label1:
        ……..
label2:
```

CPZRH is a two-word instruction that takes 8 machine cycles to complete. This instruction can specify a 13-bit immediate data (X) and upon execution, the MCU will compare the value of ZR index register with X to check if they are equal. If equal, during the next instruction cycle regardless of what instruction comes after CPZRH was, the MCU will execute the NOP instruction instead(refer to the note in 1-5-1-4); if they were not equal, the MCU will execute the instruction comes after CPZRH normally in the next instruction cycle.

The MCU will automatically suspend all interrupt requests while executing CPZRH and the next instruction until the two instruction cycles have been completed.

The table below explains the relationship between X and ZR registers for comparison:

CPZRH	X12	X11 ~ X8	X7 ~ X4	X3 ~ X0
@ZR	ZRBF12	ZRBF11~ZRBF8	ZRBF7~ZRBF4	ZRBF3~ZRBF0

Example:

```
        ……                  ; @ZR = 1234h
        CPZRH
        SETDAT      $1234h
        JMP         label1    ; NOP will instead of this jump instruction
        JMP         label2    ; this instruction will be executed and then jump to label2
        ……
```

label1:

……

label2:

1-6. Stack Register (Stack) and Stack Pointer (SP)

The Stack Register stores the address comes after the subroutine call instruction or the address before program entering the interrupt service routine using a First-in, Last-out scheme. This allows the program to return to its original program address after executing the RTS instructions.

The TM89 Series MCU's Stack Register is allocated 16 levels * 16 bits to store program addresses. The stack register also shares 64 memory addresses (0200h~023Fh) with data RAM. Each level of the stack register uses 4 contiguous addresses from the data RAM to store a 16-bit program address. All contents of the stack register can be accessed using data RAM related instructions as well.

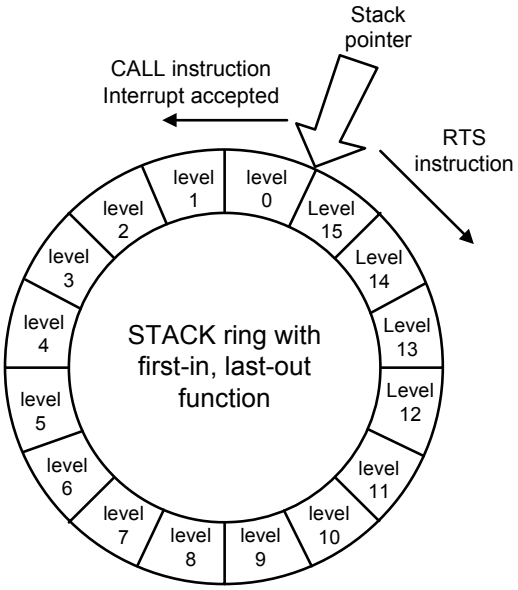
The Stack Pointer is a 4-bit up/down counter used to indicate the current usage of the stack register. The value of the Stack Pointer can be stored to data RAM and AC using the LSP instruction. When the MCU enters the RESET state the stack pointer is cleared so its initial value becomes 0.

When the MCU executes a subroutine call instruction or interrupt service routine, it will first store the address of the next instruction comes after the subroutine call instruction or the address of the instruction being executed before the interrupt being accepted to the stack register pointed to by the current stack pointer then increment the stack pointer by 1. If the value of current stack pointer is already at 0, incrementing the stack pointer by 1 at this time will cause an overflow and its value will become 1.

When the MCU executes the RTS instruction, it first decrements the current stack pointer by 1 then loads the contents of the stack register pointed to by the new stack pointer back to the program counter. If the value of current stack pointer is already at 15, decrementing the stack pointer by 1 at this time will cause an underflow and its value will become 0.

The MCU does not offer an overflow or underflow flag for the stack pointer. When a stack overflow happen the address data stored at stack register 0 will be overwritten by the new program address. In the event of a stack underflow, the address data that had been stored at stack register 15 will be loaded to the program counter.

The diagram below shows the structure of the stack register:



1-7. DATA RAM

The TM89 Series MCU's DATA RAM is composed of Random Access Memory (RAM) with a maximum addressable size of 8K addresses x 4 bits and used by programs to store data. The Data RAM is a multi-purpose memory that is used for the stack register, LCD display memory, working register (Ry), data memory (Rx) and the index register's 16-bit data backup area.

The data in the Data RAM can be accessed using two addressing modes: direct addressing and index addressing. To improve MCU's data access speed, the TM89 Series MCU also offers access modes for three types of data length: 4-bit, 8-bit and 16-bit. *Please refer to the Instructions manual for more details.*

As direct addressing can cover all data RAM addresses, the TM89 Series MCU provides page mode access for all three memory functions: LCD display memory, working register (Ry) and data memory (Rx).

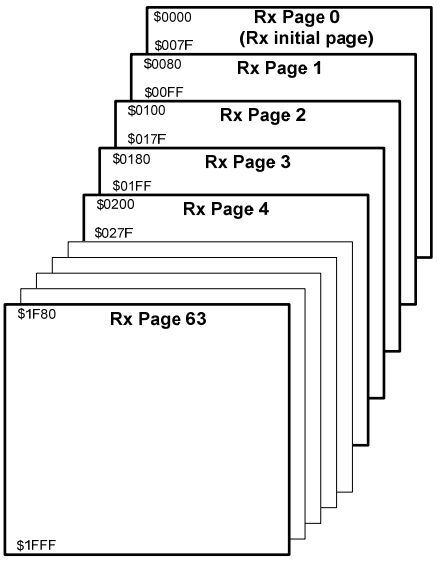
When accessing data from the data RAM using direct addressing, only 4-bit of data can be accessed in each instruction cycle. If using index addressing to access data, 4-bit, 8-bit or 16-bit data can be simultaneously accessed in each instruction cycle. A detailed description of the data RAM functions, their usage and page mode is provided below.

1-7-1. DATA MEMORY (Rx)

All Data RAM addresses (8K x 4 bits) can be used by data memory for storing general program data. Programs can access the data in data memory using two methods: direct addressing and index addressing. If using direct addressing to access data in the data memory, only Rx page mode can cover all data RAM addresses. Index addressing however can be used to directly access all data RAM addresses.

1-7-1-1. Direct Addressing with DATA Memory and Rx page Mode

Rx page mode partitions the 8K x 4 bits data memory into 64 Rx pages (page 0 ~ page 63). Each Rx page contains 128 addresses x 4 bits and with Rx page 0 designated the Rx initial page. The diagram below explains the relationship between the Rx page mode and data memory:



When the program uses direct addressing with the Rx initial page, no Rx page value needs to be set. If using direct addressing with other Rx pages however, the instruction to set the Rx page value (SRX or ERX) must be executed first to access the correct data.

The TM89 Series MCU offers two ways of setting the Rx page. It can be set automatically by the TM89 ICE's compiler program or manually by the user.

1. Automatic setting of Rx page by TM89 ICE's compiler program

   The address of the data memory in the program instruction must be defined using the absolute address. During the translation process the compiler program will automatically check to see if that address locates within the range of the Rx initial page or not. If the address is not within the Rx initial page range, the compiler will automatically adjust the Rx page by inserting an SRX instruction in front of that instruction.

   The MCU will suspend all interrupt requests while executing the SRX instruction and the instruction comes after SRX to avoid setting the page incorrectly.

   Example:

   Source file before compiling:

```
        ……
        LDS     $0100     ; Rx memory in page 2
        LDS     $0181     ; Rx memory in page 3
……
```

   After compiling:

```
……
        SRX     $2        ; set Rx memory page 2, inserted by compiler
        LDS     $0100
        SRX     $3        ; set Rx memory page 3, inserted by compiler
        LDS     $0181
……
```

2. Manual setting of Rx page by user

   Whenever the program wishes to use direct addressing on data RAM outside of the Rx initial page the SRX instruction always is inserted by compiler automatically. If this access happens too frequently it will not only increase the program size but also impact the program's execution speed. We therefore recommend the use of index addressing or set the Rx page with ERX instruction manually.

   After executing the ERX instruction, all direct addressing instructions in the program will only access the data in the specified Rx page. The compiler program will not automatically insert the SRX instruction into the program either. If the user needs to change the Rx page, simply execute the ERX instruction again.

   The program can release the ERX setting by executing the CLPG instruction (x0=1). When the compiler program encounters the CLPG (X0=1) instruction during translation, it will re-enable the automatic insertion of the SRX instruction.

   After executing any ERY, ERX or ELZ instruction, the MCU will inhibit all interrupt services. Only when all Ry, Rx and Lz page settings have been released, the MCU will enable all interrupt services again.

Example:

```
ERX   $2      ; set Rx memory in page 2
LDS   $0100   ; SRX didn't insert
LDS   $0101   ; SRX didn't insert
ERX   $3      ; update Rx memory to page 3
LDS   $0180   ; SRX didn't insert
LDS   $0181   ; SRX didn't insert
CLPG  $1
```

When the user manually sets the Rx page, they must comply with the two following guidelines when coding their program:

Guideline-1: In the program, any program address segment covered by a memory page setting instruction (ERX, ERY, ELZ) is defined as a memory page constrained segment.

During programming, do not let any JMP/CALL related instructions outside of the memory page constrained segment have a destination address that locates within a memory page constrained segment. All JMP/CALL related instructions within the memory page constrained segment should not have a destination address that locates outside of the address of the nearest ERY, ERZ, ELZ or CLPG instruction before and after that instruction itself.

This will help avoid problems with execution of JMP/CALL related instructions causing the program to end up in a different memory page constrained segment and accessing the wrong data.

Example:

```
        NOP             ; label1, 2 can't be set here!!
        ERX     $4
        JMP     label1
        NOP
label1: NOP
        NOP             ; label2 can't be set here!!
        ERX     $5
        NOP             ; label1, 2 can't be set here!!
        ERY     $27
        NOP             ; label1 can't be set here!!
label2: NOP
        NOP
        JMP     label2
        ELZ     $5
        NOP             ; label1, 2 can't be set here!!
        CLPG    $7
        NOP             ; label1, 2 can't be set here!!
```

Guideline-2: The ERX and CLPG 1 instruction can not be executed right after instructions like CPHL, CPZR, CPHLH and CPZRH in the program. This is due to the fact when a comparison of the HL/ZR content with the specified data matches, these instructions will mask the next instruction and execute NOP instead.

1-7-1-2. Using Indexed Addressing with DATA Memory

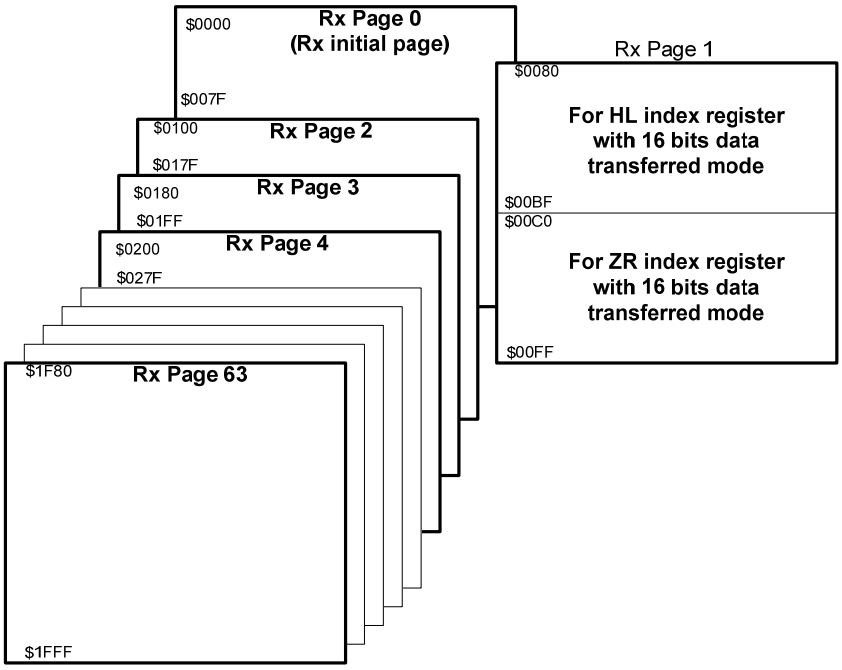
The HL and ZR index registers can be used for index addressing of the data in the Data memory. Index addressing with these two index registers can cover the entire data memory address space.

With index addressing, program run-time efficiency can be improved as some instructions can transfer 4-bit, 8-bit or 16-bit of data in 4 machine cycles. Please refer to the Instructions manual for details.

1-7-2. The INDEX REGISTER's 16-bit Data Backup Area

The MCU allocates 128 addresses (0080h ~ 00FFh) in the Data RAM to the HL and ZR index registers for use as their 16-bit data backup areas. Of these, the addresses from 0080h ~ 00BFH are only usable by the HL index register for updating/backing up data (MHL and RHL) while the addresses from 00C0h ~ 00FFh are only usable by the ZR index register for updating/backing up data (MZR and RZR).

The diagram below explains how the index registers' 16-bit data backup area addresses relates to the data RAM.



When the program uses an index register's 16-bit data backup area, the contents of the HL or ZR index register can be backed up simultaneously to four contiguous data RAM addresses within 4 machine cycles. Alternatively, the contents of 4 contiguous data RAM addresses can be updated to the HL or ZR register.

The table below explains how the index registers relates to the bits of the 4 contiguous data RAM addresses.

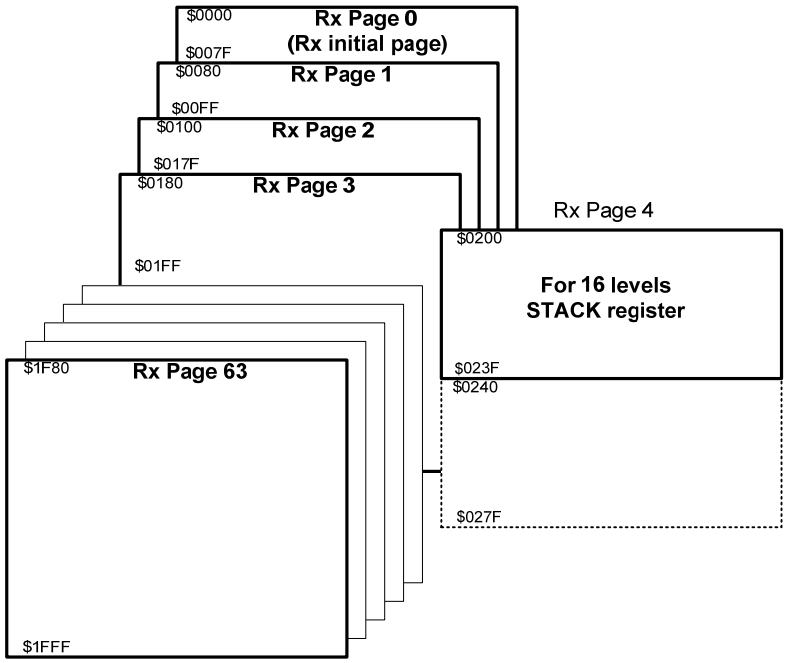
HL backup unit	HL index register(IDBF15 ~ IDBF0)			
Operand X	IDBF15~12	IDBF11~8	IDBF7~4	IDBF3~0
0	$0083	$0082	$0081	$0080
1	$0087	$0086	$0085	$0084
2	$008B	$008A	$0089	$0088
3	$008F	$008E	$008D	$008C
4	$0093	$0092	$0091	$0090
5	$0097	$0096	$0095	$0094
6	$009B	$009A	$0099	$0098
7	$009F	$009E	$009D	$009C
8	$00A3	$00A2	$00A1	$00A0
9	$00A7	$00A6	$00A5	$00A4
A	$00AB	$00AA	$00A9	$00A8
B	$00AF	$00AE	$00AD	$00AC
C	$00B3	$00B2	$00B1	$00B0
D	$00B7	$00B6	$00B5	$00B4
E	$00BB	$00BA	$00B9	$00B8
F	$00BF	$00BE	$00BD	$00BC

ZR backup unit	ZR index register(ZRBF12 ~ ZRBF0)			
Operand X	ZRBF12	ZRBF11~8	ZRBF7~4	ZRBF3~0
0	$00C3	$00C2	$00C1	$00C0
1	$00C7	$00C6	$00C5	$00C4
2	$00CB	$00CA	$00C9	$00C8
3	$00CF	$00CE	$00CD	$00CC
4	$00D3	$00D2	$00D1	$00D0
5	$00D7	$00D6	$00D5	$00D4
6	$00DB	$00DA	$00D9	$00D8
7	$00DF	$00DE	$00DD	$00DC
8	$00E3	$00E2	$00E1	$00E0
9	$00E7	$00E6	$00E5	$00E4
A	$00EB	$00EA	$00E9	$00E8
B	$00EF	$00EE	$00ED	$00EC
C	$00F3	$00F2	$00F1	$00F0
D	$00F7	$00F6	$00F5	$00F4
E	$00FB	$00FA	$00F9	$00F8
F	$00FF	$00FE	$00FD	$00FC

1-7-3. STACK REGISTER

The MCU allocates 64 addresses (0200h ~ 023Fh) in the Data RAM to the Stack Register. Each level of the stack register takes up 4 contiguous data RAM addresses. For details on how to use the stack register, please refer to 1-6.

The diagram below explains how the stack register addresses relates to the data RAM.

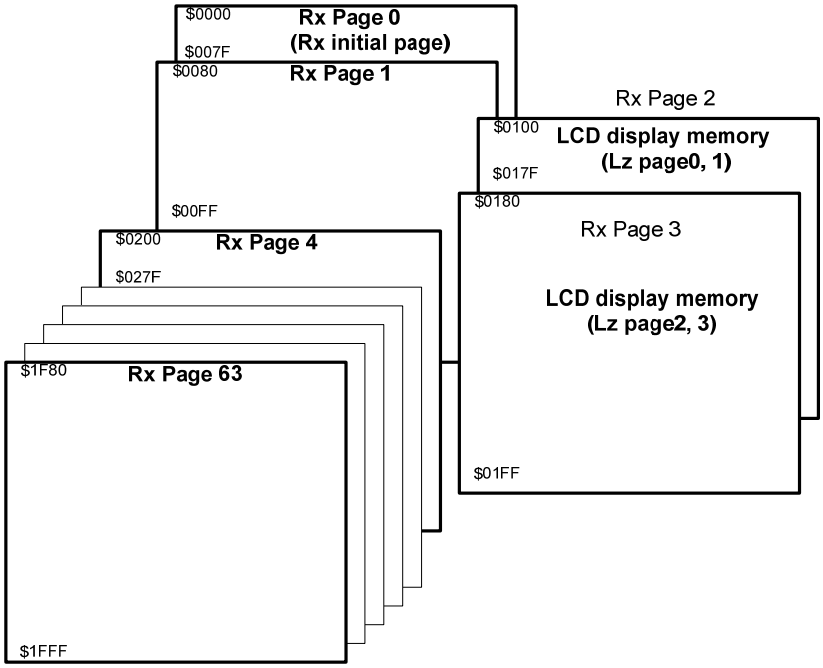


The table below explains how the PC value stored in each level of the stack level relates to the contents of 4 contiguous addresses in the data RAM.

STACK pointer	PC15~PC12	PC11~PC9	PC8~PC4	PC3~PC0
0	$0203	$0202	$0201	$0200
1	$0207	$0206	$0205	$0204
2	$020B	$020A	$0209	$0208
3	$020F	$021E	$021D	$020C
4	$0213	$0212	$0211	$0210
5	$0217	$0216	$0215	$0214
6	$021B	$021A	$0219	$0218
7	$021F	$021E	$021D	$021C
8	$0223	$0222	$0221	$0220
9	$0227	$0226	$0225	$0224
A	$022B	$022A	$0229	$0228
B	$022F	$022E	$022D	$022C
C	$0233	$0232	$0231	$0230
D	$0237	$0236	$0235	$0234
E	$023B	$023A	$0239	$0238
F	$023F	$023E	$023D	$023C

1-7-4. LCD DISPLAY MEMORY and Lz PAGE Mode

The MCU allocates 256 addresses (0100h ~ 01FFh) in the Data RAM to LCD display memory. The diagram below explains how the LCD display memory addresses relate to the data RAM.



1-7-4-1. The same accessing method as Data Memory (Rx)

Like other data RAM, LCD display memory can be treated as a part of data memory (Rx) or working register (Ry) for accessing data. Data RAM access instructions can therefore be used to access the LCD display memory as well.

1-7-4-2. Using Lz Direct Addressing to access LCD Display Memory

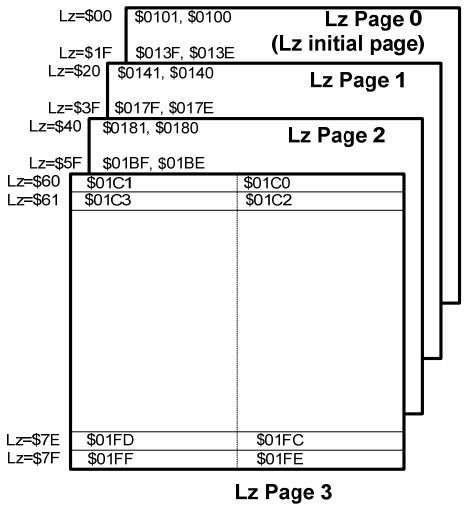
To maintain compatibility with the TM87 Series MCU's LCD instructions in the TM89 Series MCU, Lz direct addressing (PSTB) and simultaneous writing of 8-bit data (DBUS) to the LCD display memory has been retained to access the LCD display memory. For Lz direct addressing then the LCD display memory is defined as 128 Lz addresses x 8 bits, with each LCD display memory's Lz address made up of 2 contiguous data RAM addresses.

Lz direct addressing is limited to the 256 addresses of the LCD display memory however and can't address other data RAM addresses. The instructions LCT, LCB, LCP, LCD and LCDH can write data to all data RAM addresses by using the @ZR index addressing method. Additionally, when LCD related instructions make use of Lz direct addressing they can only write to the LCD display memory. Lz direct addressing can't be used to read the data from the LCD display memory.

1-7-4-3. Lz Page Mode

Lz Page mode partitions the LCD display memory into 4 Lz pages (page 0 ~ page 3), with each page containing 32 addresses x 8 bits. Lz page 0 is defined as the Lz initial page.

The diagram below explains the LCD display memory structure and Lz paging method. In the diagram, The Lz addresses outside of the page block represent the LCD display memory's Lz addresses, while the addresses inside the page block represents each Lz address's corresponding actual data RAM address.



The table below explains how the LCD display memory's bit data relates to each bit in 2 contiguous data RAM addresses:

Content in Lz address	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Content in Data memory	Bit3	Bit2	Bit1	Bit0	Bit3	Bit2	Bit1	Bit0
Address corresponding to data memory	(Lz * 2) + 101h				(Lz * 2) + 100h			

When the program uses direct addressing with the Lz initial page, no Lz page value needs to be set. If using direct addressing with other Lz pages, however, the instruction to set the Lz page value (SLZ or ELZ) must be executed first to access the correct data.

The TM89 Series MCU offers two ways of setting the Lz page. It can be set automatically by the TM89 ICE's compiler program or manually by the user.

1. Automatic setting of Lz page by TM89 ICE's compiler program

   The Lz address of the LCD display memory in the program instruction must be defined using the absolute address. During the translation process, the compiler program will automatically check to see if that address locates within the range of the Lz initial page or not. If the Lz address is not within the Lz initial page range, the compiler will automatically adjust the Lz page by inserting a SLZ instruction in front of that instruction.

   The MCU will suspend all interrupt requests while executing the SLZ instruction and the instruction comes after SLZ to avoid setting the page incorrectly.

For example,

Source file before compiling:

```
        ……
        LCB     $23, @HL        ; Rx memory in page 1
        LCB     $51, @HL        ; Rx memory in page 2
……
```

After compiling:

```
……
        SLZ     $1              ; set Rx memory page 1,
                                ; inserted by compiler.
        LCB     $23, @HL
        SLZ     $2              ; set Rx memory page 2, insert by compiler.
        LCB     $51, @HL
……
```

2. Manual setting of Lz page by user

Whenever the program wishes to use direct addressing on LCD display memory outside of the Lz initial page the SLZ instruction is always inserted automatically by compiler. If this access happens too frequently it will not only increase the program size but also impact on the program's execution speed. We therefore recommend the use of indexed addressing or set the Lz page with ELZ instruction manually.

After executing the ELZ instruction, all direct addressing instructions in the program will only access the data in the specified Lz page. The compiler program will not automatically insert the SLZ instruction into the program either. If the user needs to change the Lz page, simply execute the ELZ instruction again.

The program can release the ELZ setting by executing the CLPG instruction (X2=1). When the compiler program encounters the CLPG (X2=1) instruction during translation, it will re-enable the automatic insertion of the SLZ instruction.

After executing any ERY, ERX or ELZ instruction, the MCU will inhibit all interrupt services. Only when all Ry, Rx and Lz page settings have been released, the MCU will enable all interrupt services again.

For example:

```
ELZ     $1              ; set Lz memory in page 1
LCB     $23, @HL        ; SLZ didn't insert
LCB     $31, @HL        ; SLZ didn't insert
ELZ     $2              ; update Lz memory to page 2
LCB     $41, @HL        ; SLZ didn't insert
LCB     $52, @HL        ; SLZ didn't insert
CLPG    $4
```

When the user manually sets the Lz page, they must comply with the two following guidelines when coding their program:

Guideline-1: In the program, any program address segment covered by a memory page setting instruction (ERX, ERY, ELZ) is defined as a memory page constrained segment.

During programming, do not let any JMP/CALL related instructions outside of the memory page constrained segment have a destination address that locates within a memory page constrained segment. All JMP/CALL related instructions within the memory page constrained segment should not have a destination address that locates outside of address of the nearest ERY, ERZ, ELZ or CLPG instruction before and after that instruction itself.

This will help avoid problems with execution of JMP/CALL related instructions causing the program to end up in a different memory page constrained segment and accessing the wrong data.

For example,

```
        NOP                 ; label1, 2 can't be set here!!
        ERX     $4
        JMP     label1
        NOP
label1: NOP
        NOP                 ; label2 can't be set here!!
        ERX     $5
        NOP                 ; label1, 2 can't be set here!!
        ERY     $27
        NOP                 ; label1 can't be set here!!

label2: NOP
        NOP
        JMP     label2
        ELZ     $5
        NOP                 ; label1, 2 can't be set here!!
        CLPG    $7
        NOP                 ; label1, 2 can't be set here!!
```

Guideline-2: The ELZ and CLPG 4 instruction can not be executed right after instructions like CPHL, CPZR, CPHLH and CPZRH in the program. This is due to the fact when a comparison of the HL/ZR content with the specified data matches, these instructions will mask the next instruction and execute NOP instead.

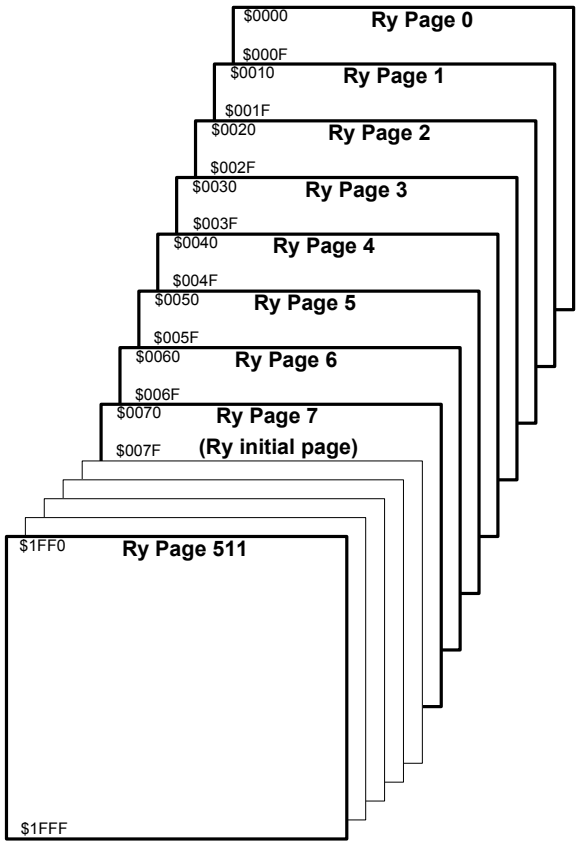
1-7-5. WORKING REGISTER (Ry) and Ry Page's Paging Mode

When the program needs to transfer data between two data RAM addresses or store the immediate data to the data RAM, it must use the working register to specify a particular memory address.

All Data RAM addresses (8K x 4 bits) can be used for the working register and the data in the working register can only be accessed through direct addressing. When using direct addressing to access the working register, Ry page mode must be used to access the data in all data RAM addresses.

The Ry page mode partitions the working register into 512 Ry pages (page 0 ~ page 511). Each page contains 16 addresses x 4 bits with Ry Page 7 being the Ry initial page.

The diagram below explains the relationship between the Ry paging mode and the working register:



When the program uses direct addressing with the Ry initial page, no Ry page value needs to be set. If using direct addressing with other Ry pages however, the instruction to set the Ry page value (SRY or ERY) must be executed first to access the correct data.

The TM89 Series MCU offers two ways of setting the Ry page. It can be set automatically by the TM89 ICE's compiler program or manually by the user.

1. Automatic setting of Ry page by TM89 ICE's compiler program

   The address of working register in the program instruction must be defined using the absolute address. During the translation process, the compiler program will automatically check to see if that address locates within the range of the Ry initial page. If the address is not within the Ry initial page range, the compiler will automatically adjust the Ry page by inserting a SRY instruction in front of that instruction.

   The MCU will suspend all interrupt requests while executing the SRY instruction and the instruction comes after SRY to avoid setting the page incorrectly.

   For example,

   Source file before compiling:

```
        ……
        LCT     @ZR, $0070      ; Rx memory in page 7
        LCT     @ZR, $0020      ; Rx memory in page 2
……
```

   After compiling:

```
……
        LCT     @ZR, $0070      ; Ry initial page
        SRY     $2              ; set Ry memory page 2, insert by compiler
        LCT     @ZR, $0020
……
```

2. Manual setting of Ry page by user

Whenever the program wishes to use direct addressing on working register outside of the Ry initial page, the SRY instruction is always inserted automatically by compiler. If this access happens too frequently it will not only increase the program size but also impact on the program's execution speed. We therefore recommend the use of indexed addressing or set the Ry page with ERY instruction manually.

After executing the ERY instruction, all direct addressing instructions in the program will only access the data in the specified RY page. The compiler program will not automatically insert the SRY instruction into the program either. If the user needs to change the Ry page, simply execute the ERY instruction again.

The program can release the ERY setting by executing the CLPG instruction (X1=1). When the compiler program encounters the CLPG (X1=1) instruction during translation, it will re-enable the automatic insertion of the SRY instruction.

After executing any ERY, ERX or ELZ instruction, the MCU will inhibit all interrupt services. Only when all Ry, Rx and Lz page settings have been released will enable all interrupt services again.

For example

```
ERY     $2              ; set Ry memory in page 2
LCT     @ZR, $0020      ; SRY didn't insert
LCT     @ZR, $002A      ; SRY didn't insert
ERY     $3              ; update Ry memory to page 3
LCT     @ZR, $0039      ; SRY didn't insert
LCT     @ZR, $003D      ; SRY didn't insert
CLPG    $2
```

When the user manually sets the Ry page, they must comply with the two following guidelines when coding their program:

Guideline-1: In the program, any program address segment covered by a memory page setting instruction (ERX, ERY, ELZ) is defined as a memory page constrained segment.

During programming, do not let any JMP/CALL related instructions outside of the memory page constrained segment have a destination address that locates within a memory page constrained segment. All JMP/CALL related instructions within the memory page constrained segment should not have a destination address that locates outside of address of the nearest ERY, ERZ, ELZ or CLPG instruction before and after that instruction itself.

This will help avoid problems with execution of JMP/CALL related instructions causing the program to end up in a different memory page constrained segment and accessing the wrong data.

For example,

```
            NOP                 ; label1, 2 can't be set here!!
            ERX     $4
            JMP     label1
            NOP
label1:NOP
            NOP                 ; label2 can't be set here!!
            ERX     $5
            NOP                 ; label1, 2 can't be set here!!
            ERY     $27
            NOP                 ; label1 can't be set here!!
label2:     NOP
            NOP
            JMP     label2
            ELZ     $5
            NOP                 ; label1, 2 can't be set here!!
            CLPG    $7
            NOP                 ; label1, 2 can't be set here!!
```

Guideline-2: the ERY and CLPG (X1=1) instruction can not be executed right after instructions like CPHL, CPZR and CPZRH in the program. This is due to the fact when a comparison of the HL/ZR content with the specified data matches, these instructions will mask the next instruction and execute NOP instead.

1-8. ACCUMULATOR (AC)

The Accumulator (AC) is an important register used for connecting the ALU with other registers or the data RAM. All data transfers must also pass through this register.

1-9. Arithmetic and Logic Unit (ALU)

The ALU is responsible for arithmetic and logic operations. The TM89 Series MCU's ALU can execute the following operations:

Binary/Decimal Addition/Subtraction	(INC, DEC, ADC(M), SBC(M), ADD(M), SUB(M), ADN, ADCI, SBCI, ADDI, SUBI, ADNI)
Logic Operation	(AND, EOR, OR, ANDI, EORI, ORI)
Shift Operation	(SR0, SR1, SL0, SL1)
Rotate Operation	(RRC, RLC)
Condition Test	(JB0, JB1, JB2, JB3, JC, JNC, JZ, JNZ, CAC, and JAC)
Binary-to-Decimal Conversion	(DAA, DAS)
Binary/Decimal Multiplication	(MULH, MULD)

When performing binary or decimal arithmetic operations in the program, make sure that all values are converted to the same number system first or the results will be wrong.

1-9-1. Converting Binary Arithmetic Results into decimals (BCD)

If the program needs to use binary arithmetic instructions with decimal values, the binary arithmetic results must be converted into decimals to get the correct results.

1-9-1-1. Binary to Decimal Conversion for Results from Binary Addition

Instructions such as DAA, DAA* and DAA# can convert the results of binary addition into decimals.

The table below explains how to convert the results of binary addition into decimals:

AC data before DAA execution	CF data before DAA execution	AC data after DAA execution	CF data after DAA execution
0 ≤ AC ≤ 9	CF = 0	no change	no change
A ≤ AC ≤ F	CF = 0	AC= AC+ 6	CF = 1
0 ≤ AC ≤ 3	CF = 1	AC= AC+ 6	no change

Example: Using the DAA* instruction to convert a binary addition operation's result into decimals.

```
LDS     $0010, 9    ; Write "9" to data memory address 10h.
LDS     $0011, 1    ; Write "1" to data memory address 11h and AC.
ADD*    $0010       ; Take the value of data memory address 10h and the value of
                    ; the AC perform the addition operation, and then write result
                    ; "A" back to data memory address 10h and AC.
```

```
DAA*    $0010    ; Convert the value of data memory address 10h into decimals,
                 ; write conversion result “0” back to data memory address 10h,
                 ; CF carries to “1”.
```

1-9-1-2. Binary to Decimal Conversion for Results from Binary Subtraction

Instructions such as DAS, DAS* and DAS# can convert the results of binary subtraction into decimals.

The table below explains how to convert the results of binary subtraction into decimals:

AC data before DAS execution	CF data before DAS execution	AC data after DAS execution	CF data after DAS execution
0 ≤ AC ≤ 9	CF = 1	no change	no change
6 ≤ AC ≤ F	CF = 0	AC= AC+ A	no change

Example: Using the DAS* instruction to convert a binary subtraction operation’s result into decimals.

```
LDS     $0010, 1    ; Write “1” to data memory address 10h.
LDS     $0011, 2    ; Write “2” to data memory address 11h and AC.
SF      1           ; Set CF to 1, indicating no borrow for subtraction.
SUB*    $0010       ; Take the value of data memory address 10h and the
                    ; value of the AC perform the subtraction operation, then
                    ; write result “F” back to data memory address 10h and
                    ; AC, CF set to 0 (there is borrow).
DAS*    $0010       ; Convert the value of data memory address 10h into
                    ; decimals, write conversion result “9” back to data
                    ; memory address 10h, CF carries to “0”.
```

1-9-2. Decimal Arithmetic Operations

The TM89 Series MCU also offers decimal arithmetic operations for carrying out decimal numerical calculations. The results come out as directly as a decimal.

1-9-2-1. Decimal Addition

Decimal addition instructions such as ADC (<m*,#) @HL/ZR, DA, ADD (M,*,#) @HL/ZR, DA can be used to produce calculation results in decimals.

Example: Using a decimal addition instruction to generate a decimal result directly:

```
SHLX
setdat  $0010    ; Write 10h to HL index register.
LDS     @HL, 9   ; Write “9” to data memory address 10h.
LDS     $0011, 1 ; Write “1” to data memory address 11h and AC.
ADD*    @HL,DA   ; Take the value of data memory address 10h and the
                 ; value of the AC perform the addition operation, then
                 ; write result “0” back to data memory address 10h,
                 ; CF carries to “1”.
```

1-9-2-1. Decimal Subtraction

Decimal subtraction instructions such as ADC (<m*,#) @HL/ZR, DA, ADD (M,*,#) @HL/ZR, DA can be used to produce calculation results in decimals.

Example: Using a decimal subtraction instruction to generate a decimal result directly:

```
SZRX
setdat    $0010       ; Write 10h to ZR index register.
LDS       @ZR, 1      ; Write "1" to data memory address 10h.
LDS       $0011, 2    ; Write "2" to data memory address 11h and AC.
SF        1           ; Set CF to 1, indicating no borrow for subtraction.
SUB*      @ZR,DA      ; Take the value of data memory address 10h and the
                      ; value of the AC perform the subtraction operation, then
                      ; write result "9" back to data memory address 10h,
                      ; CF carries to "0".
```

1-9-3. Multiplication

The TM89 Series MCU offers a 4 bits * 4 bits = 8 bits multiplication operation function. All multiplication instructions can be completed within one instruction cycle.

The following multiplication instructions can handle decimal multiplication directly:

MULD Rx, MULD @HL, MULD# @HL, MULD @ZR and MULD# @ZR

The following multiplication instructions can handle binary multiplication:

MULH Rx, MULH @HL, MULH# @HL, MULH @ZR and MULH# @ZR

During a multiplication operation, the multiplicand must be stored in the data RAM while the multiplier must be placed in the MUI register. The upper 4 bits of the multiplication results will be stored in the MU register and the lower 4 bits in the AC.

(data memory) x (MUI) = (MU, AC)

The program can use the SMUI instruction to store the data in the data RAM to the MUI register (multiplier); the MMH instruction can store the content of the MU register (product) to the data RAM and AC.

To make multiplication more efficient for programs, some arithmetic operations can operate directly on the multiplication results (the value of MU register) – e.g. ADCM, ADDM, SBCM and SUBM.

Example 1: (9 * 6 Binary Multiplication)

```
LDS     $0010, $9     ; write "9" to data memory address 10h.
LDS     $0011, $6     ; write "6" to data memory address 11h.
SMUI    $0010         ; write value of data memory address 10h to MUI.
MULH $0011            ; execute binary multiplication then write result to MU=3h and
                      ; AC=6h.
MMH     $0012         ; write the value of MU "3" to data memory address 12h.
```

Example 2: (9 * 6 Decimal Multiplication)

```
LDS     $0010, $9       ; write “9” to data memory address 10h.
LDS     $0011, $6       ; write “6” to data memory address 11h.
SMUI    $0010           ; write value of data memory address 10h to MUI.
MULD $0011              ; execute decimal multiplication then write result to MU=5
                        ; and AC=4.
MMH     $0012           ; write the value of MU “5” to data memory address 12h.
```

Example 3: (89 * 67 = 5963 Decimal Multiplication)

In this example Rx represents the data memory address. First, let us perform the multiplication manually to illustrate the multiplication procedure.

```
        89              ; multiplicand    -> store at Rx = 11h, 10h
x       67              ; multiplier      -> store at Rx = 21h, 20h
---------------
      0000              ; Step0: Multiplication result stored at Rx = 33h~30h,
                        ;        and set default to 0.
+       63
---------------
        63              ; Step1
+      56
---------------
       623              ; Step2
+      54
---------------
      1163              ; Step3
+     48
---------------
      5963              ; Step4: This is the final product of the multiplication operation.
```

A sample program for multiplication is provided below:

```
SHLX
setdat  $0010                   ; HL=10h
RHL             $0              ; store the value of HL (10h) to the HL backup unit
                                ; 0=83h~80h, HL backup unit 0 represents the address
                                ; where the multiplier in the current operation is stored.
SZRX
setdat  $0020                   ; ZR=20h
LDS8    @HL,    $89             ; store multiplicand 89 to Rx=11h, 10h
LDS8    @ZR,    $67             ; store multiplier 67 to Rx=21h, 20h
SMUI# @ZR                       ; write the multiplier’s ones’ place value (7) to the MUI,
                                ; ZR=21h
RZR             $0              ; store the value of ZR(21h) to the ZR backup unit
                                ; 0=C3h~C0h, ZR backup unit 0 represents the
                                ; address where the multiplier in the next operation
                                ; is stored.
```

```
        SZRX
        setdat  $0030           ; ZR=30h, specify Rx:33h~30h as the product.
        LDSH    @ZR
        setdat  $0000           ; write 0 to Rx:33h~30h (corresponds to
                                ; manual calculation's Step0)
        MULD# @HL               ; Rx:10h(9)xMUI(7)=MU=6, AC=3; HL=11h.
        RZR             $1      ; store the value of ZR (30h) to the ZR backup unit
                                ; 1=C7h~C4h, ZR backup unit 1 represents the
                                ; storage location of the completed product.
        ADD*# @ZR,DA            ; Rx:30h(0)+AC(3) --> CF=0, Rx:30h=3; ZR=31h.
        RZR             $2      ; store the value of ZR (31) to the ZR backup unit
                                ; 2=CB~C8h ZR backup unit 2 represents the storage
                                ; location of the next product to be calculated.
        ADCM*# @ZR,DA           ; Rx:31h(0)+MU(6)+CF(0) --> CF=0, Rx:31h=6;
                                ; ZR=32h
        JNC     lab1a           ; Jump to "lab1a" (corresponds to manual
                                ; calculation Step1)
        INC*#   @ZR
…..

lab1a:  MULD# @HL               ; Rx:11h(8) x MUI(7) --> MU=5, AC=6; HL=12h.
        MZR             $2      ; Load the value at ZR backup unit 2=CBh~C8h
                                ; (0031H) into ZR
        ADD*# @ZR,DA            ; Rx:31h(6) + AC(6) --> CF=1,Rx:31h=2; ZR =32h.
        RZR             $2      ; store the value of ZR (32h) to the ZR backup unit
                                ; 2 = CB~C8h
        ADCM*# @ZR,DA           ; Rx:32h(0)+MU(5)+CF(1) --> CF=0, Rx:32h=6; ZR =33h.
        JNC     lab2a           ; Jump to "lab2a" (corresponds to manual
                                ; calculation Step2)
        INC*#   @ZR
…..

lab2a: MHL              $0      ; load value at HL backup unit 0=83h~80h (10H) into HL
        MZR             $0      ; load the value at ZR backup unit 0=C3h~C0h (0021H)
                                ; into ZR
        SMUI# @ZR               ; write the multiplier's tens' place value (6) to the MUI,
                                ; ZR = 22h
        RZR             $0      ; store the value of ZR (22h) to the ZR backup unit
                                ; 0 = C3h~C0h,
        MULD# @HL               ; MUI(6) x Rx:10h(9) --> MU=5,AC=4; HL=11h.
        MZR             $1      ; load the value at ZR backup unit 1=C7h~C4h (30h) into ZR
        IDC%                    ; ZR+1, ZR=31h.
        RZR             $1      ; store the value of ZR (31h) to the ZR backup unit
                                ; 1 = C7h~C4h,
        ADD*# @ZR,DA            ; Rx:31h(2) + AC(4) --> CF=0, Rx:31h=6, ZR=32h.
        RZR             $2      ; store the value of ZR (32h) to the ZR backup unit
                                ; 2 = CB~C8h
        ADCM*# @ZR,DA           ; Rx:32h(6) + MU(5) + CF(0) --> CF=1,Rx:32h=1;
```

```
                                    ; ZR=33h.
        JNC     lab1b               ; No jump occur.
        INC*#   @ZR                 ; Rx:33h(0)+1 --> Rx:33h=1.
        JNC     lab1b               ; Jump to “lab1b” (corresponds to manual calculation
                                    ; Step3)
        INC*#   @ZR
…..

lab1b: MULD# @HL                    ; Rx:11h(8) x MUI(6) --> MU=4, AC=8; HL=12h
        MZR             $2          ; Load the value at ZR backup unit 2=CBh~C8h (32h)
                                    ; into ZR
        ADD*# @ZR,DA                ; Rx:32h(1) + AC(8) --> CF=0, Rx:32h=9; ZR=33h.
        RZR             $2          ; store the value of ZR (33h) to the ZR backup unit
                                    ; 2 = CB~C8h
        ADCM*# @ZR,DA               ; Rx:33h(1)+ MU(4)+CF(0) -->CF=0, Rx:33h=5; ZR=34h.
        JNC     lab2b               ; Jump to “lab2b” (corresponds to manual
                                    ; calculation Step4)
        INC*#   @ZR
…..

lab2b:  LDA     $0033               ; AC=5.
        LDA     $0032               ; AC=9.
        LDA     $0031               ; AC=6.
        LDA     $0030               ; AC=3.
```

1-10. TIMERS

The TM89 Series MCU has three sets of built-in 6-bit timers (TMR1, TMR2 and TMR3). A value can be preset for each timer then counted down to 0. The value of the timer can also be read at any time and stored to the data RAM.

The 3 timers can also be linked together to form a 12-bit or 18-bit long timer. The timers operate the same way whether they are 6-bit, 12-bit or 18-bit long. Of these, TMR1 can only be used as an independent 6-bit timer; the TMR2 can be used as an independent 6-bit timer, combined with TMR1 to form a 12-bit timer or combined with TMR1 and TMR3 to form a 18-bit timer; TMR3 can be used as an independent 6-bit timer or combined with TMR1 to form a 12-bit timer.

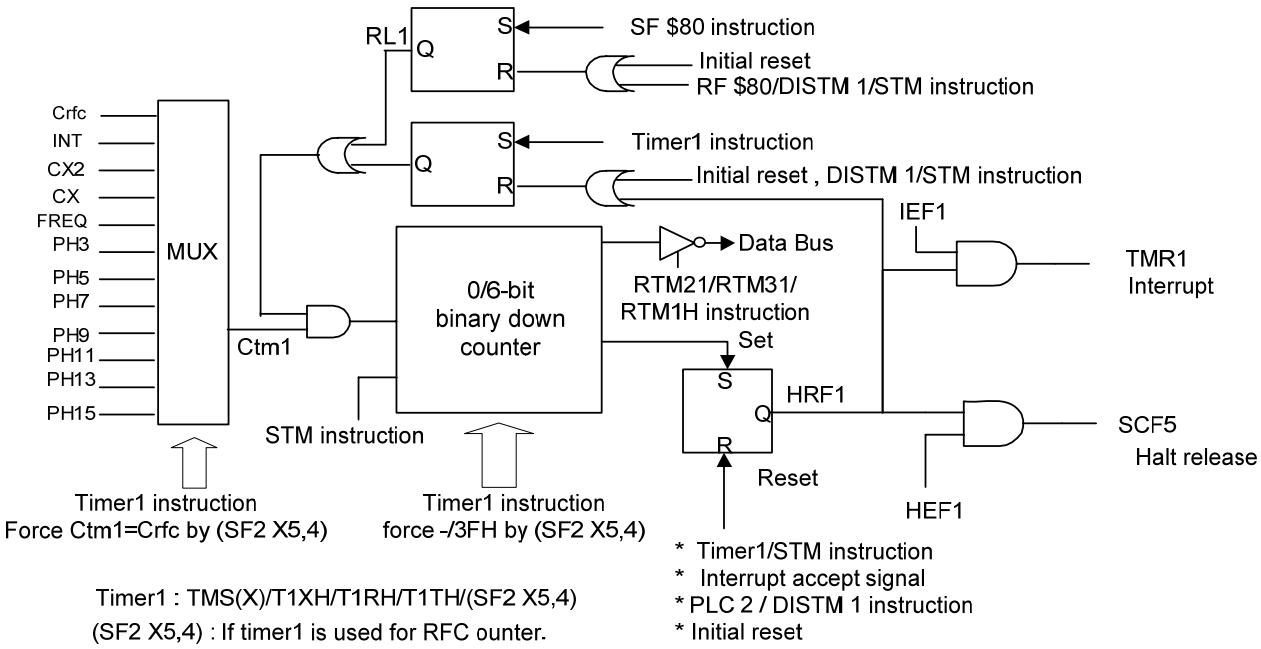
The timers can be used in the following modes: the basic one-time-only countdown (normal), automatically re-loading countdown (re-load) or used as a counter by the RFC function. TMR2 can also be used to enable or disable the RFC counter's operation.

1-10-1. 6-BIT TIMER

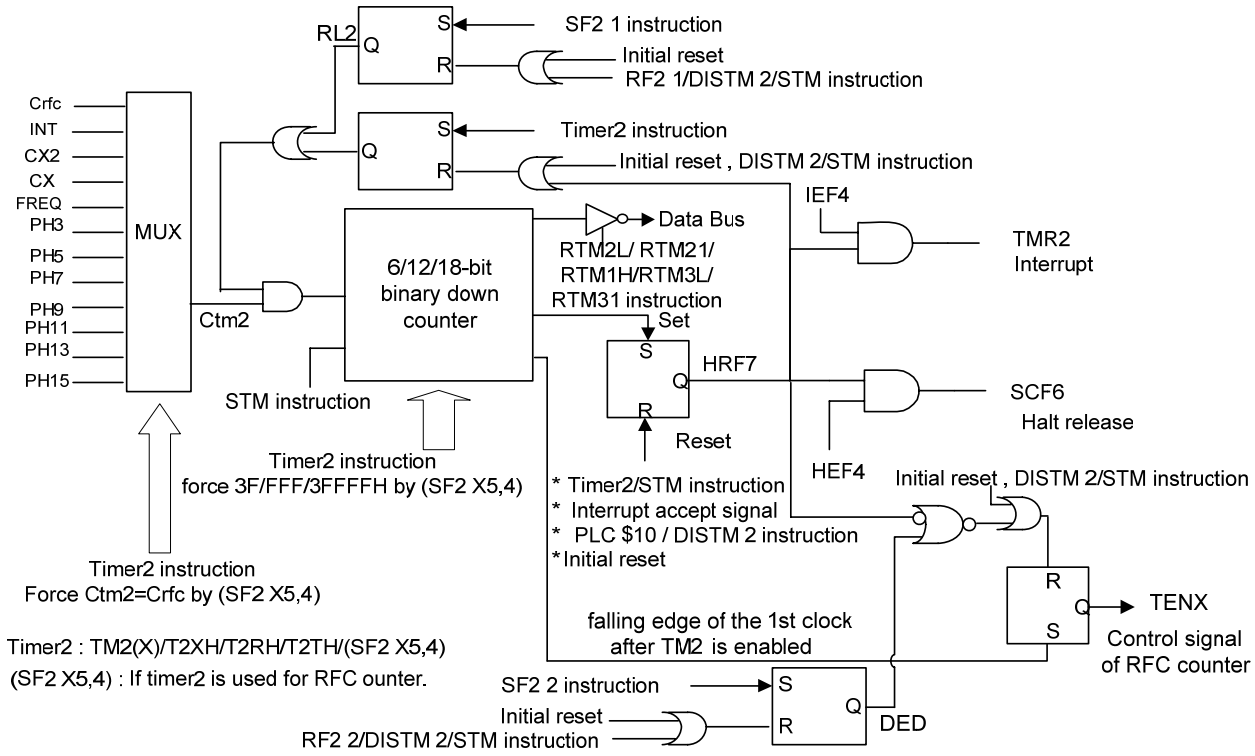
TMR1, TMR2 and TMR3 can all be used as individual 6-bit timers and all work the same way.

The three following diagrams set out the circuit structure of TMR1, TMR2 and TMR3.

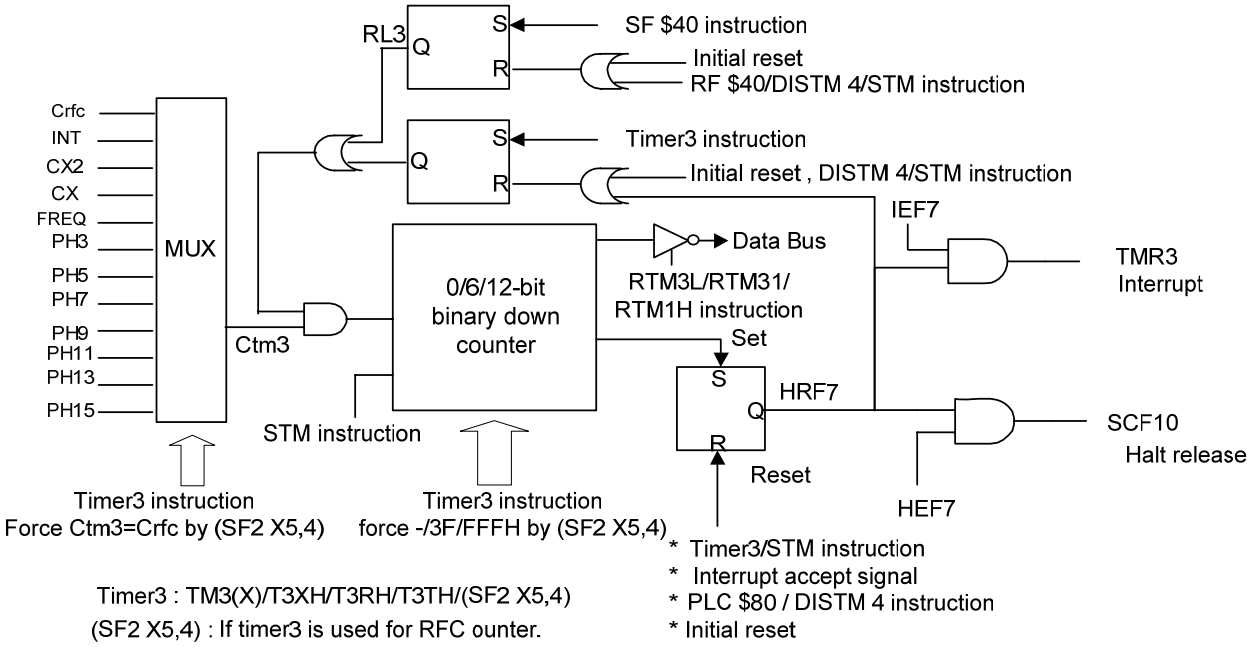
TMR1's Circuit Structure:



TMR2's Circuit Structure:



TMR3's Circuit Structure:



1-10-1-1. One-time-only Countdown Function

This is the timer's basic function. The setting of the timer's preset data, the selection of clock source and the activation of the timer can all be configured in one instruction. The timer will also stop counting after an underflow or the program executes the DISTM instruction.

After the program executes the DISTM instruction to stop the timer, the timer's content at the time of deactivation will be retained.

The table below explains the instructions for setting the 6-bit timer's preset data, selecting the clock source and activating the timer:

TMRn	OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT5~0)					
TMR1	T1XH SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6
	T1RH SETDAT RX	(RX)15	(RX)14	(RX)13	(RX)12	(RX)11	(RX)10	(RX)9	(RX)8	(RX)7	(RX)6
	T1TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6
	TMSX X	0	X8	X7	X6	X5	X4	X3	X2	X1	X0
	TMS Rx	0	0	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
	TMS @HL	0	0	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
TMR2	T2XH SETDAT SD	SD15	SD14	SD13	SD12	SD5	SD4	SD3	SD2	SD1	SD0
	T2RH SETDAT RX	(RX)15	(RX)14	(RX)13	(RX)12	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
	T2TH @HL	TD15	TD14	TD13	TD12	TD5	TD4	TD3	TD2	TD1	TD0
	TM2X X	0	X8	X7	X6	X5	X4	X3	X2	X1	X0
	TM2 Rx	0	0	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
	TM2 @HL	0	0	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
TMR3	T3XH SETDAT SD	SD15	SD14	SD13	SD12	SD5	SD4	SD3	SD2	SD1	SD0
	T3RH SETDAT RX	(RX)15	(RX)14	(RX)13	(RX)12	(RX)5	(RX)4	(RX)3	(RX)2	(RX)1	(RX)0
	T3TH @HL	TD15	TD14	TD13	TD12	TD5	TD4	TD3	TD2	TD1	TD0
	TM3X X	0	X8	X7	X6	X5	X4	X3	X2	X1	X0
	TM3 Rx	0	0	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
	TM3 @HL	0	0	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

When the timer counts down from the preset data to 0 it generates a timer underflow signal. This signal will set the timer halt release request flag (HRFn) to 1 and disable the timer. Executing the PLC instruction clears the HRFn flag.

If the timer interrupt enable flag (IEFn) is already set to 1, when the HRFn flag is set to 1 the MCU will accept the interrupt request delivered by the timer.

If the timer halt release enable flag (HEFn) is already set to 1, when the HRFn flag is set to 1 the MCU will generate a HALT release. At the same time, it will also set the start condition flag (SCFn) of status register 3 (STS3) to 1.

The table below lists each timer's corresponding flag and associated configuration instructions:

Timer	Configurable timer halt release request flag (HRFn)	PLC Instruction to clear HRFn Flag	Corresponding timer interrupt enable flag (IEFn)	Corresponding timer halt release enable flag (HEFn)	Corresponding start condition flag (SCFn)
TMR1	HRF1	PLC $02	IEF1	HEF1	SCF5
TMR2	HRF4	PLC $10	IEF4	HEF4	SCF6
TMR3	HRF7	PLC $80	IEF7	HEF7	SCF10

Please note that when the timer is counting down and the re-load function is not enabled, executing the PLC instruction will stop the timer; if the timer is counting down and the re-load function is enabled, executing the PLC instruction will not stop the timer.

The table below explains how to set the clock source:

Bit setting in instruction				clock source of each timer
CS3	CS2	CS1	CS0	
0	0	0	0	~PH9
0	0	0	1	~PH3
0	0	1	0	~PH15
0	0	1	1	FREQ
0	1	0	0	~PH5
0	1	0	1	~PH7
0	1	1	0	~PH11
0	1	1	1	~PH13
1	0	0	0	CX
1	0	0	1	CX2
1	0	1	0	INT

Notes:

1. When the TIMER clock is PH3
TIMER set time = (Set value + error) * 8 * 1/fosc (KHz) (ms)

2. When the TIMER clock is PH9
TIMER set time = (Set value + error) * 512 * 1/fosc (KHz) (ms)

3. When the TIMER clock is PH15
TIMER set time = (Set value + error) * 32768 * 1/fosc (KHz) (ms)

4. When the TIMER clock is PH5
TIMER set time = (Set value + error) * 32 * 1/fosc (KHz) (ms)

5. When the timer clock is PH7
TIMER set time = (Set value + error) * 128 * 1/fosc (KHz) (ms)

6. When the TIMER clock is PH11
TIMER set time = (Set value + error) * 2048 * 1/fosc (KHz) (ms)

7. When the TIMER clock is PH13
TIMER set time = (Set value + error) * 8192 * 1/fosc (KHz) (ms)

8. When the TIMER clock is FREQ

TIMER set time = (Set value + error) * 1/FREQ (KHz) (ms). *Refer to section 2-3-4 for the detail of FREQ.*

9. When the TIMER clock is CX

TIMER set time = (Set value + error) * 1/CX (KHz) (ms).

10. When the TIMER clock is CX2

TIMER set time = (Set value + error) * 1/CX2 (KHz) (ms).

11. When the TIMER clock is INT

TIMER set time = (Set value + error) * 1/INT (KHz) (ms).

Set value: Decimal number of timer set value

error: the tolerance of set value, 0 < error <1.

fosc: clock source of the predivider

PH3~PH15: The 3rd~15th stage output of the predivider

1-10-1-2. Automatic Timer Re-load Function

If the timer's automatic countdown re-load function is enabled, when an underflow occurs at the timer it does not stop counting down. It will instead begin counting down from 3Fh again. The timer will not stop until the program disable the re-load function.

The table below sets out the timer re-load function's control instructions:

Instruction	TMR1 re-load function	TMR2 re-load function	TMR3 re-load function
SF $80	Enabled	NA	NA
SF2 $01	NA	enabled	NA
SF $40	NA	NA	enabled
RF $80	Disabled	NA	NA
DISTM 1	Disabled	NA	NA
RF2 $01	NA	Disabled	NA
DISTM 2	NA	Disabled	NA
RF $40	NA	NA	Disabled
DISTM 4	NA	NA	Disabled
STM	Disabled	Disabled	Disabled

Once the program enables the Timer's re-load function the timer will begin counting down. The program should therefore only enable the re-load function after the timer's preset data has been set or the timer will begin counting down from a random initial value.

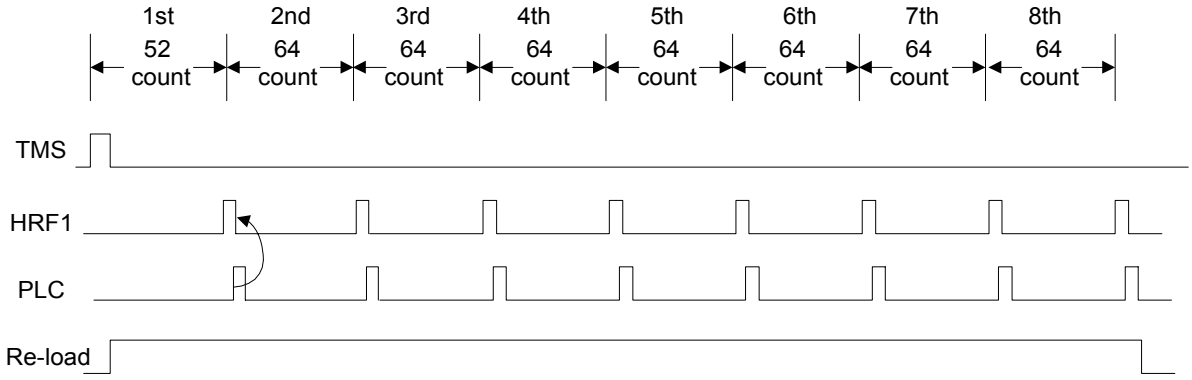
After enabling the timer's re-load function, whenever the timer counts down to 0 a timer underflow occurs. At this point the timer will begin counting down from 3Fh again though the timer underflow will still set the timer halt release request flag (HRFn) to 1. The timer halt release request flag (HRFn) must therefore be used to determine how many timer underflow events had occurred already in order to calculate the length of time correctly.

Before the re-load function is disabled, the timer can be preset at any time. Once a new preset data has been set for the timer it will begin counting down from the new value immediately.

When the re-load function is disabled, the timer will stop running immediately. The user should therefore disable the timer's re-load function only after the last timer underflow to avoid affecting the normal operation of the timer.

An example of how the timer's re-load function can be used is provided below:

If the program wishes the timer to count down from 500, it can translate this into once the countdown from 52 and 7 countdowns from 64 (64*7 + 52). In this example, the program enters HALT mode to wait for the underflow generated by TMR1.




```
        LDS     0, 0            ; use the value of data memory address 0 as TMR1's
                                ; underflow frequency counter and reset it to 0.
        PLC     2               ; clear HRF1
        SHE     2               ; allow the MCU to generate HALT release on TMR1 underflow
        TMSX    $34             ; set TMR1's preset data (52), choose PH9 as clock source,
                                ; then start TMR1
        SF      $80             ; enable TMR1's re-load function
RE_LOAD:
        HALT
        INC*    0               ; increment TMR1's underflow counter by 1
        PLC     2               ; clear HRF1
        JB3     END_TM1         ; check if TMR1 underflow frequency equals 8
        JMP     RE_LOAD
END_TM1:
        RF      $80             ; disable TMR1's re-load function
```

1-10-1-3. Using TIMER as RFC COUNTER

Please see 2-8-2-2 for more details.

1-10-1-4. Using TMR2 to Control RFC COUNTER

Please see 2-8-2-4 for more details.

1-10-1-5. Reading out the Content of TIMER

There are two methods for reading out the current content of the three timers and storing it to the data memory. One method is to read out each of the three timer's content directly 4 bits at a time; the other method is to simultaneously read out and store the content of the 3 timers (18-bit) to a specified register then read the data of the 18-bit register 4 bits at a time.

Reading the timers' content doesn't affect their normal operation. However, as the read time can't be synchronized with the timers' state changes, this means that it's possible to read the content of the timer while it's changing state.

As all timers are based on one countdown timer architecture and the designers took into consideration of the fact that the timer's content will mostly be used as a counter, this led to the readout of timer being the complement of the current timer's content.

For example if the timer began counting down from 3F, after counting down for 6 clocks the timer changes to 39h. Reading out the timer at this point gives a value of 6, indicating that the timer has counted 6 clocks.

The table below illustrates how the timer's content and the read out data are complementary to one another:

Content of timer	Read out data
3Fh	00h
3Eh	01h
…	…
01h	3Eh
00h	3Fh

Below are detailed explanations of how to use the two readout methods:

1. Directly read out the content of the 3 timers 4 bits at a time (default MCU setting):

   The table below lists the instructions that can be used for reading out the contents of the 3 timers directly 4 bits at a time.

Inst. for fetching timer	Content of TMR3						Content of TMR2						Content of TMR1					
	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RTM2L									(Rx)3	(Rx)2	(Rx)1	(Rx)0						
RTM21							(Rx)1	(Rx)0									(Rx)3	(Rx)2
RTM1H													(Rx)3	(Rx)2	(Rx)1	(Rx)0		
RTM3L			(Rx)3	(Rx)2	(Rx)1	(Rx)0												
RTM31	(Rx)1	(Rx)0															(Rx)3	(Rx)2

   In the above table, after executing these instructions the 4-bit content of the timers will be written directly to the data memory and AC.

   Note: (Rx)3 ~ (Rx)0 represents bit3~bit0 of data memory value.

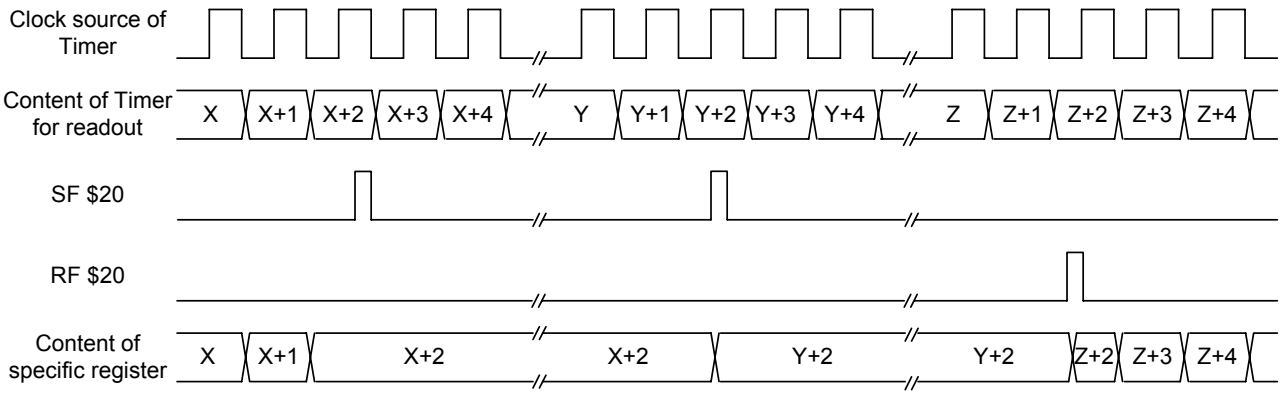
2. Simultaneously read the contents of all 3 timers (18-bit):

   Executing SF $20 simultaneously reads out the contents of the 3 timers (18-bit) then stores them to a specific register. The instructions in the above table can then store the 18-bit data 4 bits at a time to the data memory and AC.

   After executing the SF $20 instruction, the data in the specific register will not be changed even if the timer continues to count. The data in the specific register will only be updated when the SF $20 instruction is executed again.

After the program executes the RF $20 instruction, the timer's readout mode will revert back to read each of the 3 timers separately 4 bits at a time.

Shown below is the timing diagram for simultaneously reading out of the 3 timers' contents (18-bit):



1-10-2. 12-BIT TIMER

The 12-bit timer is made up of two 6-bit timers. TMR1 and TMR2 can be combined to form a 12-bit TMR2 or TMR1 and TMR3 combined to form a 12-bit TMR3. Under the 12-bit TMR2 or TMR3 structure the function of 6-bit TMR1 will be disabled.

The table below lists the instructions for merging timers to make a 12-bit timer:

instruction to merge timers	TMR1	TMR2	TMR3
STM 1	●	● (dominate)	
STM 2	●		● (dominate)

The 12-bit TMR2 functions the same way as the 6-bit TMR2 while the 12-bit TMR3 functions the same way as the 6-bit TMR3.

If the 12-bit timer's re-load function is enabled, when an underflow occurs at the timer it does not stop counting down. It will instead begin counting down from FFFh again. The timer will not stop until the program disables the re-load function.

The table below lists the instructions for setting and activating the 12-bit timers:

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT11~0)											
T2XH SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
T2RH SETDAT RX	(Rx)15	(Rx)14	(Rx)13	(Rx)12	(Rx)11	(Rx)10	(Rx)9	(Rx)8	(Rx)7	(Rx)6	(Rx)5	(Rx)4	(Rx)3	(Rx)2	(Rx)1	(Rx)0
T2TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
T3XH SETDAT SD	SD15	SD14	SD13	SD12	SD11	SD10	SD9	SD8	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0
T3RH SETDAT RX	(Rx)15	(Rx)14	(Rx)13	(Rx)12	(Rx)11	(Rx)10	(Rx)9	(Rx)8	(Rx)7	(Rx)6	(Rx)5	(Rx)4	(Rx)3	(Rx)2	(Rx)1	(Rx)0
T3TH @HL	TD15	TD14	TD13	TD12	TD11	TD10	TD9	TD8	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0

Please note that when using timers in 12-bit timer mode, do not use 6-bit TMR2 or TMR3 related instructions for configuration as this will lead to incorrect settings.

The table below shows how to read out the contents of the 12-bit TMR2 and 6-bit TMR3:

timer read-out inst.	Content of new 12-bit TMR2												Content of TMR3					
	Bit-11	Bit-10	Bit-9	Bit-8	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RTM2L									(Rx)3	(Rx)2	(Rx)1	(Rx)0						
RTM21					(Rx)3	(Rx)2	(Rx)1	(Rx)0										
RTM1H	(Rx)3	(Rx)2	(Rx)1	(Rx)0														
RTM3L															(Rx)3	(Rx)2	(Rx)1	(Rx)0
RTM31					(Rx)3	(Rx)2							(Rx)1	(Rx)0				

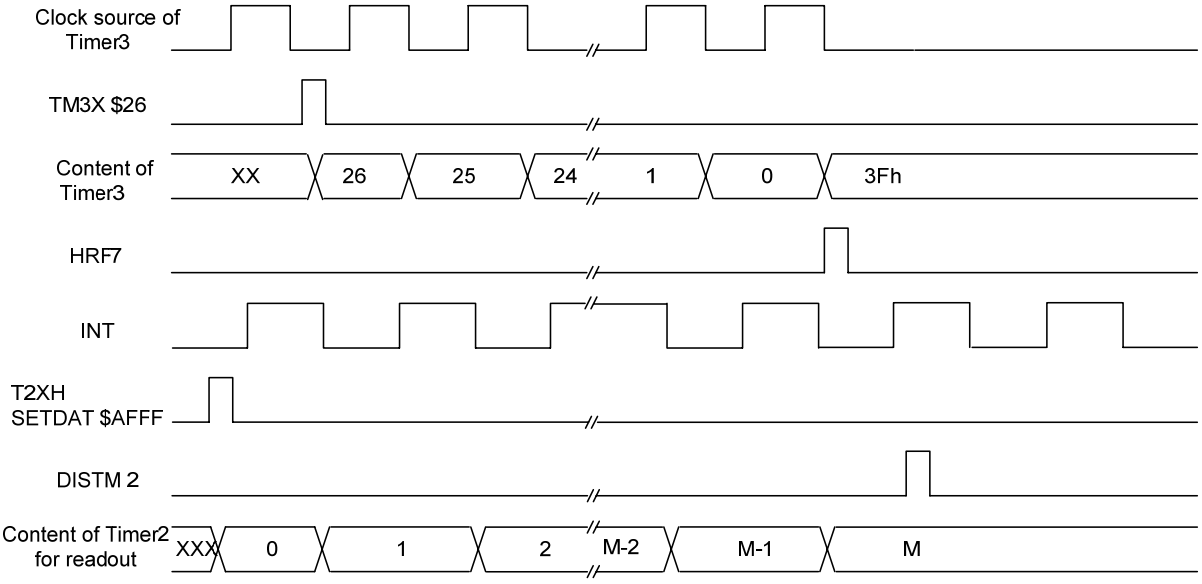
The table below shows how to read out the contents of the 12-bit TMR3 and 6-bit TMR2:

timer read-out inst.	Content of new 12-bit TMR3												Content of TMR2					
	Bit-11	Bit-10	Bit-9	Bit-8	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RTM2L															(Rx)3	(Rx)2	(Rx)1	(Rx)0
RTM21					(Rx)3	(Rx)2							(Rx)1	(Rx)0				
RTM1H	(Rx)3	(Rx)2	(Rx)1	(Rx)0														
RTM3L									(Rx)3	(Rx)2	(Rx)1	(Rx)0						
RTM31					(Rx)3	(Rx)2	(Rx)1	(Rx)0										

The example below shows how to use the 12-bit timer 2 (TMR2) and 6-bit timer 3 (TMR3) to calculate the clock frequency received on INT pin.

```
        STM     $1              ; set TMR2 to 12-bit timer
        T2XH                    ; set INT as TMR2' clock source with preset data of
                                ; FFFh
        setdat  $AFFF
        TM3X $026               ; set PH9 as TMR3' clock source with preset data of 26h
        SHE     $90             ; allow TMR2 and TMR3 to generate HALT release
        HALT
        DISTM$2                 ; disable TMR2
        MCX     $6f             ; check if TMR2 underflow
        JB1     tm2_ov
        PLC     $070            ; clear HRF7 (TMR3).
        RTM2L $0010             ; read out the content of 12-bit TMR2
        RTM21 $0011             ; this represents the clock frequency on INT pin
        RTM1H $0012             ;
……..
tm2_ov:NOP                      ; underflow on TMR2
```

The diagram below shows the timing for the signals used in this example:



1-10-3. 18-BIT TIMER

The 18-bit timer is made up of 3 6-bit timers. Only one 18-bit TMR2 can be created. Under the 18-bit timer structure, the function of 6-bit TMR1 and TMR3 will be disabled.

The table below lists the instructions for merging timers to make a 18-bit timer:

instruction to merge timers	TMR1	TMR2	TMR3
STM 3	●	● (dominate)	●

The 18-bit TMR2 functions the same way as the 6-bit TMR2.

If the 18-bit timer's re-load function is enabled, when an underflow occurs at the timer it does not stop counting down. It will instead begin counting down from 3FFFFh again. The timer will not stop until the program disables the re-load function.

The table below lists the instructions for setting and activating the 18-bit timers:

OPCODE	Select clock(CS3~0)				Pre-set data of timer(CT17~0)						
T2M3X X SETDAT SD	SD15	SD14	SD13	SD12	X5	X4	X3	X2	X1	X0	CT17~12
					SD11	SD10	SD9	SD8	SD7	SD6	CT11~6
					SD5	SD4	SD3	SD2	SD1	SD0	CT5~0

Please note that when the TMR2 is set to 18-bit timer mode, do not use 6-bit TMR2 or 12-bit TMR2 related instructions for configuration as this will lead to incorrect settings.

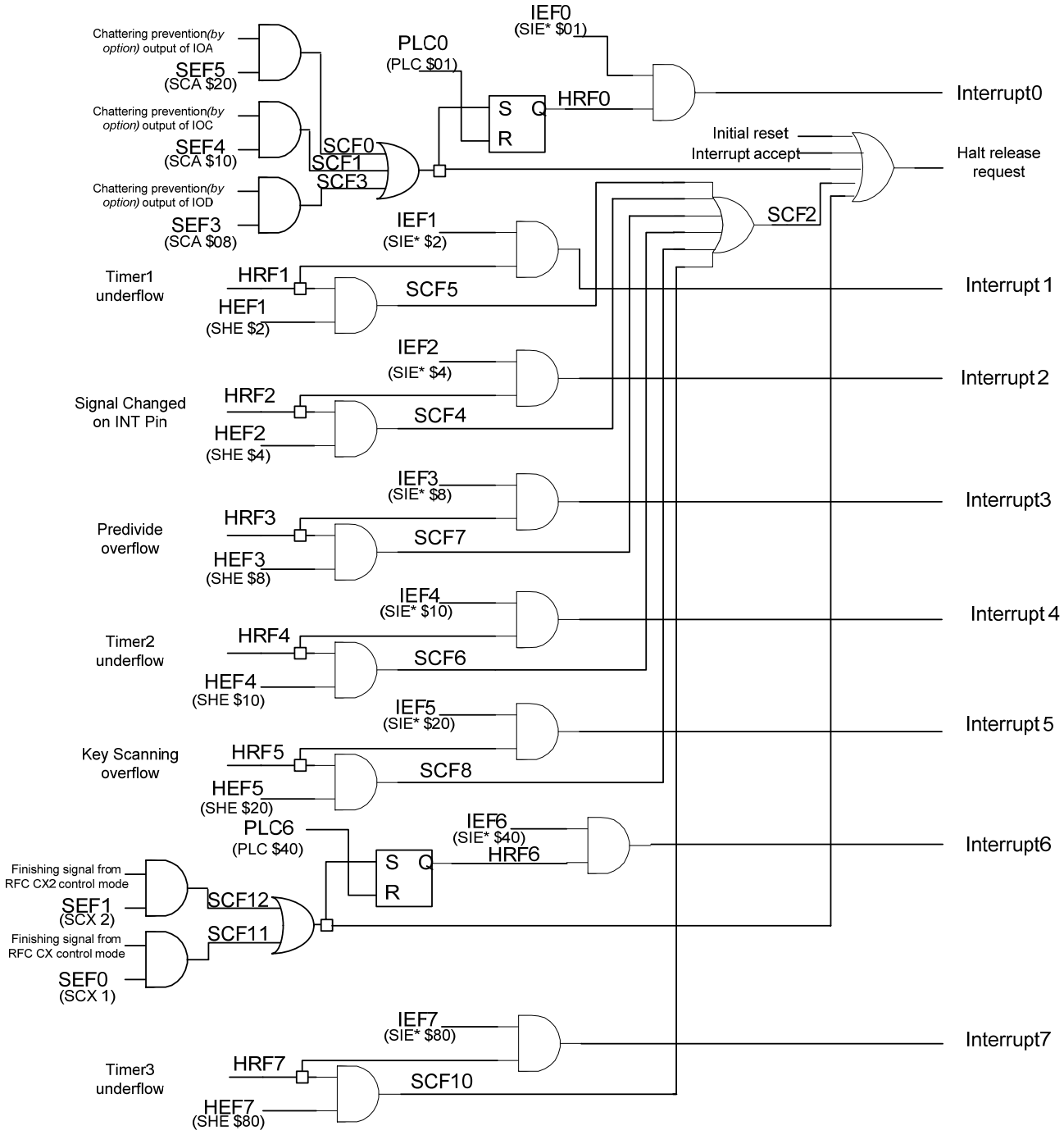
The table below explains how to read out the content of the 18-bit TMR2:

timer read-out inst.	Content of 18-bit TMR2																	
	Bit-17	Bit-16	Bit-15	Bit-14	Bit-13	Bit-12	Bit-11	Bit-10	Bit-9	Bit-8	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
RTM2L															(Rx)3	(Rx)2	(Rx)1	(Rx)0
RTM21											(Rx)3	(Rx)2	(Rx)1	(Rx)0				
RTM1H							(Rx)3	(Rx)2	(Rx)1	(Rx)0								
RTM3L			(Rx)3	(Rx)2	(Rx)1	(Rx)0												
RTM31	(Rx)1	(Rx)0									(Rx)3	(Rx)2						

1-11. STATUS REGISTER (STS)

The TM89 Series MCU has 7 different status registers (STS): STS1, STS2, STS3, STS3X, STS4, STS4X and STS5. There are mainly used for recording the various start condition flags that make the MCU generate a HALT release and other important signal flags. Each status register is made up of 4 bits of data and can be read and written to the data memory using the relevant instructions.

The diagram below shows the structure of the start condition flags in the TM89 Series MCU:



1-11-1. STATUS REGISTER 1 (STS1)

STS1 records the start condition flag 11 and 12 (SCF11, 12). These two are factors that can make the MCU generate a HALT release.

STS1 is composed of 4 flags:

1. Carry flag (CF)
Carry flag records the carry and borrow results during arithmetic operations.

2. Zero flag (ZF)
Zero flag indicates the content status of the Accumulator (AC).
When the AC value is 0, the zero flag is set to 1;
When the AC value is not equal to 0, the zero flag is cleared.

3. Start condition flag 12 (SCF12)
If the RFC counter is enabled or disabled by the input signal on CX2 pin, when the program executes the SCX instruction to set the SEF1 flag to 1, the SCF12 flag will be set to 1 after the control signal on the CX2 pin finishes.
When the program executes the SCX instruction again it will automatically clear SCF12.

4. Start condition flag 11 (SCF11)
If the RFC counter is enabled or disabled by the input signal on CX pin, when the program executes the SCX instruction to set the SEF0 flag to 1, the SCF11 flag will be set to 1 after the control signal on the CX pin finishes.
When the program executes the SCX instruction again it will automatically clear SCF11.

Executing the MAF instruction stores the content of STS1 to AC and data memory.

Executing the MRA instruction stores the content of the data memory to the STS1 carry flag.

The table below explains how the flags in status register 1 (STS1) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Carry flag (CF)	Zero flag(Z)	SCF12	SCF11
Read / write	Read only	Read only	Read only

1-11-2. STATUS REGISTER 2 (STS2)

STS2 records the start condition flag 1 and 2 (SCF1, 2, 3). These three are factors that can make the MCU generate a HALT release.

STS2 is composed of four flags: the three start condition flags 1, 2, 3 (SCF1, 2, 3) and the backup flag (BCF):

1. Start condition flag 3 (SCF3)
If the SCA instruction has set signal changes on the IOD port input pin to make the MCU generate a HALT release, SCF3 is then set to 1 when there's a change in these input signals.
When the program executes the SCA instruction again it will automatically clear SCF3.

2. Start condition flag 1 (SCF1)
If the SCA instruction has set signal changes on the IOC port input pin to make the MCU generate a HALT release, SCF1 is then set to 1 when there's a change in these input signals.
When the program executes the SCA instruction again it will automatically clear SCF1.

3. Start condition flag 2 (SCF2)
When any one of the factors capable of making the MCU generate a HALT release such as flags SCF4, 5, 6, 7, 9 and 10 occurs, SCF2 is set to 1. Only when all of these flags are cleared will SCF2 be cleared as well.
As long as any start condition flag is set to 1 the MCU can't enter HALT mode.

4. Backup flag (BCF)
The BCF flag indicates if the MCU is in back up mode.
Executing the SF 2h instruction lets the MCU enter the back up mode and sets the BCF flag to 1; executing the RF 2h instruction lets the MCU exit the back up mode and clears the BCF flag.

Executing the MSB instruction stores the content of STS2 to the AC and data memory. STS2 is itself a read-only register.

The table below explains how the flags in status register 2 (STS2) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Start condition flag 3 (SCF3)	Start condition flag 2 (SCF2)	Start condition flag 1 (SCF1)	Backup flag (BCF)
Halt release caused by the IOD port	Halt release caused by SCF4,5,6,7,9,10	Halt release caused by the IOC port	The back up mode status
Read only	Read only	Read only	Read only

1-11-3. STATUS REGISTER 3 (STS3)

STS3 records the start condition flags 4, 5, 7 (SCF4, 5, 7). These three are factors that can make the MCU generate a HALT release.

STS3 is composed of four start condition flags:

1. Start condition flag 4 (SCF4)
When the halt release enable flag 2 (HEF2) is set to 1, the halt release flag 2 (HRF2) triggered by a rising or falling edge on the INT pin (set by mask option) will set the start condition flag 4 (SCF4) to 1.
Executing the PLC instruction to clear the halt release request flag 2 (HRF2) or executing the SHE instruction to clear halt release enable flag 2 (HEF2) will both clear the start condition flag 4 (SCF4).

2. Start condition flag 5 (SCF5)
When the halt release enable flag 1 (HEF1) is set to 1, the halt release request flag 1 (HRF1) generated by TMR1 after underflow will set the start condition flag 5 (SCF5) to 1.
Executing the PLC instruction to clear the halt release request flag 1 (HRF1) or executing the SHE instruction to clear halt release enable flag 1 (HEF1) will both clear the start condition flag 5 (SCF5).

3. Start condition flag 7 (SCF7)
When the halt release enable flag 3 (HEF3) is set to 1, the halt release request flag 3 (HRF3) generated by the pre-divider after overflow will set the start condition flag 7 (SCF7) to 1.
Executing the PLC instruction to clear the halt release request flag 3 (HRF3) or executing the SHE instruction to clear halt release enable flag 3 (HEF3) will both also clear the start condition flag 7 (SCF7).

4. Pre-divider's 15^{th} stage Output Signal (PH15)

Executing the MSC instruction stores the content of STS3 to the AC and data memory. STS3 is itself a read-only register.

The table below explains how the flags in status register 3 (STS3) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Start condition flag 7 (SCF7)	Content of 15th stage of the pre-divider	Start condition flag 5 (SCF5)	Start condition flag 4 (SCF4)
Halt release caused by pre-divider overflow		Halt release caused by TMR1 underflow	Halt release caused by INT pin
Read only	Read only	Read only	Read only

1-11-4. STATUS REGISTER 3X (STS3X)

STS3X records the start condition flags 0, 6, 8 (SCF0, 6, 8). These three are factors that can make the MCU generate a HALT release.

STS3X is composed of three start condition flags:

1. Start condition flag 8 (SCF8)
When the halt release enable flag 5 (HEF5) is set to 1, at the end of each key matrix scanning cycle or when the key matrix scanning circuit detects a high voltage level input signal (set by the corresponding instructions for the key matrix scanning function) on pins KI1~4, the halt release request flag 5 (HRF5) generated will set the start condition flag 8 (SCF8) to 1.
Executing the PLC instruction to clear the halt release request flag 5 (HRF5) or executing the SHE instruction to clear halt release enable flag 5 (HEF5) will both clear the start condition flag 8 (SCF8).

2. Start condition flag 6 (SCF6)
When the halt release enable flag 4 (HEF4) is set to 1, the halt release request flag 4 (HRF4) generated by TMR2 after underflow will set the start condition flag 6 (SCF6) to 1.
Executing the PLC instruction to clear the halt release request flag 4 (HRF4) or executing the SHE instruction to clear halt release enable flag 4 (HEF4) will both clear the start condition flag 6 (SCF6).

3. Start condition flag 0 (SCF0)
If the SCA instruction has set signal changes on the IOA port input pin to make the MCU generate a HALT release, SCF0 is then set to 1 when there's a change in these input signals.
When the program executes the SCA instruction again it will automatically clear SCF0.

Executing the MSX instruction stores the content of STS3X to the AC and data memory. STS3X is itself a read-only register.

The table below explains how the flags in status register 3X (STS3X) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Start condition flag 0 (SCF0)	Start condition flag 6 (SCF6)	Start condition flag 8 (SCF8)
	Halt release caused by the IOA port	Halt release caused by TMR2 underflow	Halt release caused by key matrix scanning function
Read only	Read only	Read only	Read only

1-11-5. STATUS REGISTER 4 (STS4)

STS4 is composed of 4 flags:

1. System clock selection flag (CSF)
The CSF flag indicates the current clock source used by the system clock generator. When the CSF flag is set to 1 it means that the slow clock (XT clock) is currently in use. When the CSF flag is cleared it means the fast clock (CF clock) is currently in use.

2. Watch dog timer enable flag (WDF)
The WDF flag indicates whether the watch dog timer function is currently enabled. When the WDF flag is set to 1 it means the watch dog timer function is enabled. When the WDF flag is cleared it means the watch dog timer function is currently disabled.

3. Overflow flag of 16-bit RFC counter (RFOVF)
When an overflow happens at the 16-bit RFC counter the RFOVF flag is set to 1. Executing the SF2(X6 and X5) instruction clears the RFOVF flag.

Executing the MSD instruction stores the content of STS4 to the AC and data memory. STS4 is itself a read-only register.

The table below explains how the flags in status register 4 (STS4) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Reserved	The overflow flag of 16-bit RFC counter (RFVOF)	Watch dog timer Enable flag (WDF)	System clock selection flag (CSF)
Read only	Read only	Read only	Read only

1-11-6. STATUS REGISTER 4X (STS4X)

STS4 records the start condition flag 10 (SCF10). This is a factor that can make the MCU generate a HALT release.

STS4X is made up of start condition flag 10 (SCF10) and the input signal states on three input pins (CX, CX2 and INT):

1. Input Signal State on CX Input Pin
The CX pin is an input pin for the RFC function. The MCU can use the STS4X register to read its logic level.

2. Input Signal State on CX2 Input Pin
The CX2 pin is another input pin for the RFC function. The MCU can use the STS4X register to read its logic level.

3. Input Signal State on INT Input Pin
The INT pin is an external interrupt signal source. The MCU can use the STS4X register to read its logic level.

4. Start condition flag 10 (SCF10)
When the halt release enable flag 7 (HEF7) is set to 1, the halt release request flag 7 (HRF7) generated by TMR3 after underflow will set the start condition flag 10 (SCF10) to 1.
Executing the PLC instruction to clear the halt release request flag 7 (HRF7) or executing the SHE instruction to clear halt release enable flag 7 (HEF7) will both clear the start condition flag 10 (SCF10).

Executing the MDX instruction stores the content of STS4X to the AC and data memory. STS4X is itself a read-only register.

The table below explains how the flags in status register 4X (STS4X) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Start condition flag 10 (SCF10)	Logic state on INT pin	Logic state on CX2 pin	Logic state on CX pin
Halt release caused by TMR1 underflow			
Read only	Read only	Read only	Read only

1-11-7. STATUS REGISTER 5 (STS5)

STS5 is composed of the logic states from four key matrix scanning function's input pins (KI1 ~ KI4):

Executing the MKI instruction stores the content of STS5 to the AC and data memory. STS5 is itself a read-only register.

The table below explains how the flags in status register 5 (STS5) relate to the data memory contents:

Bit 3	Bit 2	Bit 1	Bit 0
Logic state on KI4 key-scanning input pin	Logic state on KI3 key-scanning input pin	Logic state on KI2 key-scanning input pin	Logic state on KI1 key-scanning input pin
Read only	Read only	Read only	Read only
Bit 3	Bit 2	Bit 1	Bit 0

1-12. CONTROL REGISTER (CTL)

The TM89 Series has a total of 5 different control registers (CTL): CTL1, CTL2, CTL3, CTL4 and CTL5. The MCU must set to the corresponding control register 1 in order to generate a HALT release when start condition flags occur.

1-12-1. CONTROL REGISTER 1 (CTL1)

CTL1 is made up of SEF3, SEF3 and SEF5. Executing the SCA instruction can set or clear these flags. Control register 1 (CTL) is a read-only register.

1. Switch enable flag 5 (SEF5)

When the switch enable flag 5 (SEF5) is set to 1, any effective signal change on the IOA port's input pin causes the MCU to generate a HALT release.

2. Switch enable flag 4 (SEF4)

When the switch enable flag 4 (SEF4) is set to 1, any effective signal change on the IOC port's input pin causes the MCU to generate a HALT release.

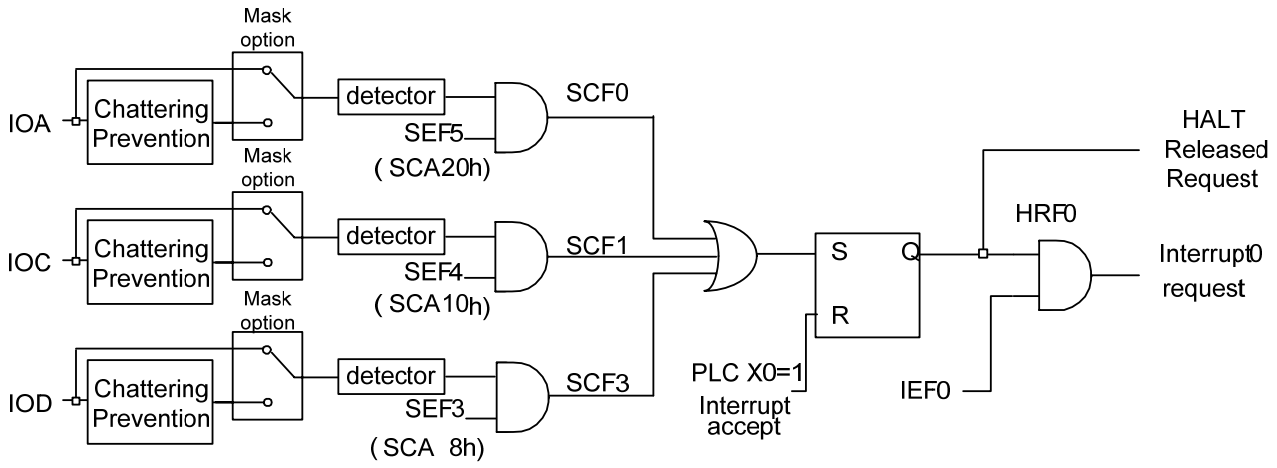
3. Switch enable flag 3 (SEF3)

When the switch enable flag 3 (SEF3) is set to 1, any effective signal change on the IOD port's input pin causes the MCU to generate a HALT release.

The table below explains the function of each flag in control register 1 (CTL1):

Bit5	Bit 4	Bit3
Switch enable flag 5 (SEF5)	Switch enable flag 4 (SEF4)	Switch enable flag 3 (SEF3)
Enables the halt release caused by the signal change on IOA port (HRF0)	Enables the halt release caused by the signal change on IOC port (HRF0)	Enables the halt release caused by the signal change on IOD port (HRF0)
Write only	Write only	Write only

The diagram below explains how the control register 1 (CTL1) relates to other signals:



In the above diagram, the detector represents the high voltage level detection function. A detailed explanation of how CTL1 is used in different states is provided below.

1-12-1-1. HALT release setup

When the control register 1 (CTL1) flags are set to 1, the MCU will generate a HALT release if there is any change in signal after all the signals from the pins of IO ports such as IOA, IOC and IOD are passed through the OR logic gate (with chattering prevention is enabled) or any of the input signals changes to a high voltage level (when high level detection is enabled). The start condition flags 0, 1, 3 (SCF0, SCF1, SCF3) are also set to 1.

When the high voltage level detection function is used by the IO port to set the start condition flag, only when all signals on the IO port input pins are at the low voltage level can make the MCU enter HALT mode. If any signal on an input pin changes to a high voltage level the corresponding start condition flag will be set to 1.

When the chattering prevention function on the IO port is used to set the start condition flag, the MCU can enter the HALT mode with any kind of input signal's states on the IO port's input pins. As long as there is any signal change in the "OR" result for the input pin signals then the corresponding start condition flag will be set to 1. Please refer to the diagrams of IO port architecture in 2-5-1, 2-5-3 and 2-5-4.

1-12-1-2. STOP release setup

The conditions for the MCU to generate a STOP release will vary according to the method used by the IO port to set the start condition flag. The MCU can only enter the STOP mode when all input signals from the IO port are at the low voltage level.

When high voltage level detection function is used for the IO port to set the start condition flag, if control register 1 (SEF5, SEF4, SEF3) is set to 1 then any change to a high voltage level by an input signal on an IOA, IOC and IOD input pin will cause the MCU to generate a STOP release and set the corresponding start condition flag.

When the chattering prevention function is used for the IO port to set the start condition flag, if control register 1 (SEF5, SEF4, SEF3) and control register 4 (SRF6, SRF4, SRF3) are all set to 1 at the same time then any change to a high voltage level by a input signal on an IOA, IOC or IOD port input pin will cause the MCU to generate a STOP release. After exiting STOP mode, the MCU will enter HALT mode first. The high voltage level signal on the IO port input pin must be maintained for a certain amount of time (oscillator start-up time plus chattering prevention timing cycle) until the chattering prevention function confirms the high voltage level signal and set the corresponding start condition flag.

If this high voltage level signal reverts back to a low voltage level before the start condition flag is set, the MCU will return immediately to STOP mode.

1-12-1-3. Interrupt Service setup

Setting the start condition flags 0, 1, 3 (SCF9, SCF1, SCF3) to 1 will set the halt release request flag (HRF0) to 1.

In this situation, if the SIE* instruction had already set the interrupt enable flag 0 (IEF0) to 1 then the MCU will accept this interrupt request. The interrupt request flag 0 (interrupt 0) will then be set to 1 and the program will enter the IO port's interrupt subroutine.

Please note that after accepting an interrupt request from IOA, C or D port the MCU will automatically clear control register (SEF5, SEF4 and SEF3). The user must therefore setup the CTL1 register by executing the SCA instruction again before using it again.

1-12-2. CONTROL REGISTER 2 (CTL2)

Control register 2 (CTL2) is made up of the halt release enable flags 1~5, 7 (HEF1~5, 7). Executing the SHE instruction can set or clear these flags.

The table below explains the function of each flag in control register 2 (CTL2):

Halt release enable flag	HEF7		HEF5	HEF4
Halt release condition	Enable the halt release caused by TMR3 underflow (HRF7)		Enable the halt release caused by key matrix scanning function(HRF5)	Enable the halt release caused by TMR2 underflow (HRF4)
Halt release enable flag	HEF3	HEF2	HEF1	
Halt release condition	Enable the halt release caused by pre-divider overflow (HRF3)	Enable the halt release caused by INT pin (HRF2)	Enable the halt release caused by TM1 underflow (HRF1)	

1-12-2-1. HALT release setup

If the halt release enable flags 1, 4, 7 (HEF1, 4, 7) are set to 1, when an underflow occurs at TMR1, 2 and 3 will cause the MCU to generate a HALT release.

In the same manner, if the halt release enable flags 2, 3 (HEF2, 3) are set to 1, while the INT pin input signal changes with rising/falling edge or an overflow occurs at the pre-divider will cause the MCU to generate a HALT release.

When the halt release enable flag 5 (HEF5) is set to 1, at the end of each key-matrix scan cycle or when the key-matrix scanning circuit on pins KI1~4 detect a high voltage level input signal (set by the corresponding instructions for the key matrix scanning function) will cause the MCU to generate a HALT release.

1-12-2-2. Set up MCU to generate STOP release

When the MCU enters STOP mode, only HEF2 and HEF5 2 flags offer the way of making the MCU generate a STOP release:

1. After the halt release enable flag 2 (HEF2) is set to 1, any halt release request flag 2 (HRF2) generated by any change in signal on the INT input pin will set the start condition flag (SCF4) to 1. This will cause the MCU to generate a STOP release.
2. After the halt release enable flag 5 (HEF5) is set to 1, if the key matrix scanning function's scanning circuit on the KI1~4 pins detect a high voltage level input signal will generate a halt release request flag 5 (HRF5) that sets the start condition flag 8 (SCF8) to 1. This will cause the MCU to generate a STOP release.

   The key matrix scanning function can't use the scanning cycle signal synchronized with the LCD waveform to generate STOP release because the LCD driver does not send a scanning signal in STOP mode. In this situation it must be set to Normal key scanning mode to generate a STOP release.

1-12-3. CONTROL REGISTER 3 (CTL3)

Control register 3 (CTL3) is made up of eight interrupt enable flags 0~7 (IEF0~7) that can be used to enable or disable the interrupt requests. Executing the SIE* function sets or clears these interrupt enable flags.

The table below explains the function of each flag in control register 3 (CTL3):

Interrupt enable flag	IEF7	IEF6	IEF5	IEF4
Interrupt request flag	Enable the interrupt request caused by TMR3 underflow (HRF7)	Enable the interrupt request caused by RFC counter is controlled by CX/CX2 pin (HRF6)	Enable the interrupt request caused by key matrix scanning function (HRF5)	Enable the interrupt request caused by TMR2 underflow (HRF4)
Interrupt flag		Interrupt 6	Interrupt 4	Interrupt 4
Interrupt enable flag	IEF3	IEF2	IEF1	IEF0
Interrupt request flag	Enable the interrupt request caused by predivider overflow (HRF3)	Enable the interrupt request caused by INT pin (HRF2)	Enable the interrupt request caused by TM1 underflow (HRF1)	Enable the interrupt request caused by IOA, IOC or IOD port signal changed (HRF0)
Interrupt flag	Interrupt 3	Interrupt 2	Interrupt 1	Interrupt 0

When the program sets an interrupt enable flag, once an interrupt factor (HRFn) listed in the above table is generated, an interrupt request will be sent to the MCU.

When the MCU accepts the interrupt request and enters the interrupt service routine, it will automatically clear the associated interrupt factor (HRFn) and interrupt enable flag (IEFn). The interrupt enable flag (IEFn) must be set again before the program finishes its interrupt service routine in preparation for the next interrupt factor.

Even if no halt release enable flag (HEFn) was set the MCU can still generate HALT release and enter the interrupt service routine after accepting the interrupt request. Some interrupt factors however still need their corresponding Switch enable flags (SEFn) to be set in order to generate a HALT release and enter the interrupt service routine.

Only the three types of interrupt factors listed below can cause a MCU to generate a STOP release and enter the interrupt service routine.

1. After the interrupt enable flag 0 (IEF0) is set to 1, as long as the signal on the IOA, IOC, IOD port input pins changes to a high voltage level and the start condition flags 0, 1, 3 (SCF0, SCF1, SCF3) can be set to 1 then the MCU will generate a STOP release.

2. After the interrupt enable flag 2 (IEF2) is set to 1, any change to the input signal on the INT pin will cause the MCU to generate a STOP release.

3. After the interrupt enable flag 5 (IEF5) is set to 1, when the key matrix scanning function's scanning circuit detects a high voltage level signal on the KI1~4 pins will cause the MCU to generate a STOP release.

1-12-4. CONTROL REGISTER 4 (CTL4)

Control register 4 (CTL4) is made up of the 3 stop release enable flags 3, 4, 6 (SRF3, 4, 6). Executing the SRE instruction can set or clear these stop release enable flags.

The table below explains the function of each flag in control register 4 (CTL4):

Stop release enable flag	SRF6	SRF4	SRF3
Stop release request	Enable the stop release caused by signal changed on IOA when chattering prevention option is enabled	Enable the stop release caused by signal changed on IOC when chattering prevention option is enabled	Enable the stop release caused by signal changed on IOD when chattering prevention option is enabled

When the chattering prevention function is used for the IOA, IOC and IOD ports to set the start condition flag, the Control register 4 (CTL4) must be set in order to let the STOP release can be generated.

If control register 1 (SEF5, SEF4, SEF3) and control register 4 (SRF6, SRF4, SRF3) are all set to 1 at the same time then any change to a high voltage level by a input signal on an IOA, IOC or IOD port input pin will cause the MCU to generate a STOP release.

Example:

This example sets the IO port, KI1~4 and INT pins to factors that can cause the MCU to generate a STOP release. IOC and IOD are both set to input mode, chattering prevention is enabled for IOC but not for IOD.

```
PLC     $25     ; clear HRF0, HRF2, HRF5
SHE     $24     ; set HEF2, HEF5
SCA     $18     ; set SEF4, SEF3
SRE     $10     ; set SRF4 because IOC is using Chattering Prevention
STOP            ; Enters the stop mode.

……              ; STOP release

MSC     $10     ; check if STOP release generated due to INT pin
MSB     $11     ; check if STOP release generated due to IOC or IOD port
MCX     $12     ; check if STOP release generated due to key matrix scanning
                ; function
```

1-12-5. CONTROL REGISTER 5 (CTL5)

Control register 5 (CTL5) is made up of the switch enable flags 0 and 1 (SEF 0, 1). Executing the SCX instruction can set or clear these stop switch enable flags.

The table below explains the function of each flag in control register 5 (CTL5):

Bit 1	Bit 0
Switch enable flag 1 (SEF1)	Switch enable flag 0 (SEF0)
Enables the halt release caused by the end of the control signal when RFC counter is controlled by CX2 (SCF12)	Enables the halt release caused by the end of the control signal when RFC counter is controlled by CX (SCF11)
Write only	Write only

If the RFC counter is enabled or disabled by the input signal on CX2 pin, when the program executes the SCX instruction to set the switch enable flag 1 (SEF1) to 1, the start condition flag 12 (SCF12) will be set to 1 after the control period on the CX2 pin ends, causing the MCU to generate a HALT release.

If the RFC counter is enabled or disabled by the input signal on CX pin, when the program executes the SCX instruction to set the switch enable flag 0 (SEF0) to 1, the start condition flag 11 (SCF11) will be set to 1 after the control signal on the CX pin ends, causing the MCU to generate a HALT release.

1-13. HALT MODE

When the MCU enters HALT mode, the program execution is stopped but all other functions will continue to operate normally. This mode will partially reduce the MCU's power consumption.

After the program executes the HALT instruction, if there are no factors that can cause the MCU to generate a HALT release active (e.g. SCF0, SCF1, SCF3, SCF11, SCF12, HRF1~5, HRF7), the MCU will enter HALT mode.

Any of the four following situations can make the MCU generate a HALT release:

1. An interrupt event

When there is an interrupt event the MCU will automatically generate a HALT release. Once the program finishes running the interrupt service routine then executes the RTS instruction, the MCU will return to HALT mode.

If the MCU generated a HALT release due to an interrupt, the corresponding interrupt factor (halt release signal, HRFn) will be automatically cleared.

2. After executing the SCA instruction to set flags SEF5, 4, 3 to 1, the MCU will generate a HALT release if there is any change in signal after all the signals from the input pins of IO ports such as IOA, IOC and IOD are passed through the OR logic gate (with chattering prevention enabled) or any of the input signals changes to a high voltage level (when high voltage level detection is enabled).

When the high voltage level detection function is used by the IO port to set the start condition flag, only when all signals on the IO port input pins are at the low voltage level can the MCU enter HALT mode. Any change to a high voltage level by an input pin signal will cause the MCU to generate a HALT release.

When the chattering prevention function on the IO port is used to set the start condition flag, the MCU can enter HALT mode with any kind of signal state on the IO port's input pin. If there is any signal change in the "OR" result for the input pin signals then the MCU will generate a HALT release.

3. After executing the SCX instruction to set the flags SEF 1, 0 to 1, if the RFC counter is enabled or disabled by the input signal on the CX2 or CX pin then the MCU will generate a HALT release when the control period on the CX2 or CX pin ends.

4. When the SHE instruction's settings allow factors(HRF1~HRF5, HRF7) that causes the MCU to generate a HALT release, the MCU will generate a HALT release.

If the MCU did not generate a HALT release due to an interrupt event, the program must use instructions like MAF, MSB, MSC, MCX and MDX to check the halt release flag (HRFn) out that caused the MCU to generate a HALT release. It must then execute the PLC instruction to clear these halt release flags (HRFn) so the MCU can enter HALT mode next time.

Before the program clears all these halt release flags (HRFn) the MCU can't go into HALT mode.

1-14. STOP MODE

When the MCU enters STOP mode all functions stops operating. In this mode the MCU itself consumes no power. Under the STOP mode, all SEGn and COMn output pins will output a GND level signal. They will only return to their normal output waveform when the MCU generates a STOP release.

After the MCU enters the STOP mode it will automatically set the BCF flag to 1 and activate backup mode to allow the oscillator to be successfully started up after the MCU exits the STOP mode. To reduce the MCU's power consumption, we recommend turning off the backup mode in the program after the MCU generates a STOP release if appropriate.

Before the MCU can go into the STOP mode, the program must complete the following tasks (set only if necessary):

1. If chattering prevention function is enabled for IOA, IOC and IOD, execute the SRE instruction to set the associated SRF6, 4, 3 flags.

2. Execute the SCA instruction to set the flags SEF5, 4, 3. This will allow the IOA, IOC and IOD ports to generate a STOP release.

3. Set the interrupt enable flags (IEF0, 2, 5) for the INT, Key matrix scanning function, IOA, IOC and IOD as the factors to generate a STOP release. Also make it that only external interrupt factors can generate a STOP release.

4. Confirm all of the signals on the IOA, IOC and IOD input pins that can cause the MCU to generate a STOP release at the low voltage level.

5. Clear the stop release signals (HRF0, HRF5, HRF2).

The following situations can cause the MCU to generate a STOP release:

A. The signal on any IOA, IOC or IOD port input pin changes to a high voltage level and maintains this level until the MCU generates a HALT release.

If chattering prevention is enabled for IOA, IOC or IOD ports, the MCU will only generate a STOP release when the signal on the IO port specified by the SRE instruction changes to a high voltage level.

B. When there is a change in the signal on the INT pin.

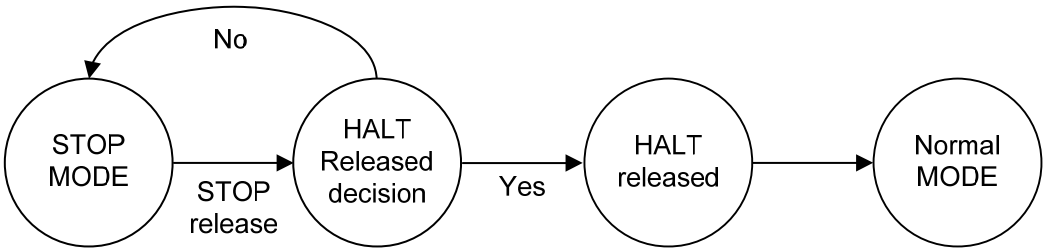
C. When the key matrix scanning function detects a high voltage level input signal on pins KI1~4.

When the MCU generates a STOP release it will first enter HALT mode then determine whether it can generate a HALT release to go into normal mode. If the high voltage level signal on the IOA, IOC or IOD port input pins can't be maintained until the MCU sets the SCF0, 1, 3 flags to 1, the MCU will return to the STOP mode when the input pins change to a low voltage signal.

After the MCU goes into normal mode the program can use the MSB, MSC or MCX instructions to identify the halt release flags (HRF0, 2, 5) that caused the MCU to generate a STOP release. The PLC instruction can then be executed to clear these halt release flags (HRF0, 2, 5) so the MCU can enter the STOP mode next time. Before the program clears these halt release flags (HRF0, 2, 5) the MCU can't go into STOP mode.

When an external interrupt occurs, after the MCU accepts the interrupt request it will generate a STOP release and then execute the interrupt service routine directly. When the MCU generates a STOP release it will clear all halt release signals (HRFn). After executing the interrupt service routine's RTS instruction the MCU will return to STOP mode.

The diagram below shows how the MCU goes into normal mode after generating a STOP release.



1-15. BACK UP MODE

When the power-intensive functions on the system are activated this causes a major fluctuation in the system's power supply voltage. The TM89 Series MCU's backup mode allows the MCU to continue operating normally in this situation. Turning on the backup mode increases the MCU's power consumption however so our recommendation to users is to keep it turned off unless necessary to optimize power conservation.

The back up flag (BCF) can show whether the MCU's backup mode is currently turned on. When the BCF flag is set to 1 this indicates that the MCU's backup mode is active. Executing the SF instruction enables the backup mode while executing the RF instruction disables the backup mode.

After enabling the backup mode some of the MCU's functions will automatically make adjustments or change their nature. These are described below:

1. If the 32.768KHz Crystal Oscillator was selected by the MCU as its slow clock oscillator, once backup mode is enabled the oscillator will increase its driving capability to avoid interference from the power supply interfering with its output frequency or even stopping the oscillator altogether. This however comes as the cost of increased MCU power consumption.

2. When the MCU enters STOP mode it will automatically enable backup mode and set the BCF flag to 1.

3. Under the 3V power mode, after backup mode is enabled the MCU's internal functions will operate in the VBAT power supply voltage; after disabling the backup mode, if the mask option is set for "BAK=VL1 when BCF=0" then the MCU's internal functions will work under the lower operating voltage (VL1) for the sake of power conservation.

In this situation please note that the output frequency from the RC oscillator will vary due to the change in internal operating voltage. Additionally, if the VL1 voltage is provided by an external voltage regulator then the supplied voltage must be higher than the MCU's lowest operating voltage to ensure the normal operation of the MCU.

As the MCU can only operate with higher operating voltage in FAST mode, do not stay in or enter the FAST mode after disabling the backup mode to ensure the correct operation of the MCU.

The MCU will automatically set different backup modes for different power supply modes. These are listed in the table below:

1.5V power mode or 3V power mode (BCF=0 => BAK=VBAT):

TM89 Series status	BCF flag status
Initial reset cycle	BCF = 1 (hardware controlled)
After initial reset cycle	BCF = 1 (hardware controlled)
Executing SF 2h instruction	BCF = 1
Executing RF 2h instruction	BCF = 0
HALT mode	Previous state
STOP mode	BCF = 1 (hardware controlled)

MCU Function	BCF = 0	BCF = 1
32.768KHz Crystal Oscillator	Small Driver	Large Driver
BAK Pin Voltage (Supplied to the clock oscillator and internal functions)	VBAT	VBAT

3V power mode (BCF=0 => BAK=VL1):

TM89 Series status	BCF flag status
Initial reset cycle	BCF = 1 (hardware controlled)
After initial reset cycle	BCF = 1 (hardware controlled)
Executing SF 2h instruction	BCF = 1
Executing RF 2h instruction	BCF = 0
HALT mode	Previous state
STOP mode	BCF = 1 (hardware controlled)

MCU Function	BCF = 0	BCF = 1
32.768KHz Crystal Oscillator	Small Driver	Large Driver
BAK Pin Voltage (Supplied to the clock oscillator and internal functions)	VL1	VBAT

Chapter 2 Control Function

2-1. INTERRUPT FUNCTION

The TM89 Series MCU has eight interrupt factors in total: 3 external interrupt factors and 5 internal interrupt f actors.

When the MCU receives an interrupt request it will automatically suspend the current program then jump to the corresponding interrupt address to execute the interrupt service routine.

The table below lists the various interrupt sources, vectors and associated flags:

Interrupt source	INT pin	IOA,C,D port	TMR1 underflow	Pre-divider overflow	TMR2 underflow	Key matrix Scanning	End of RFC counter controlled signal	TMR3 under-flow
Interrupt vector	010h	014h	018h	01Ch	020h	024h	028h	02Ch
Interrupt enable flag	IEF2	IEF0	IEF1	IEF3	IEF4	IEF5	IEF6	IEF7
Interrupt request flag	HRF 2	HRF 0	HRF 1	HRF 3	HRF 4	HRF 5	HRF 6	HRF 7

Please note that in the following situations the MCU will temporarily stop servicing all interrupt requests.

1. When executing all 8 machine cycle instructions.

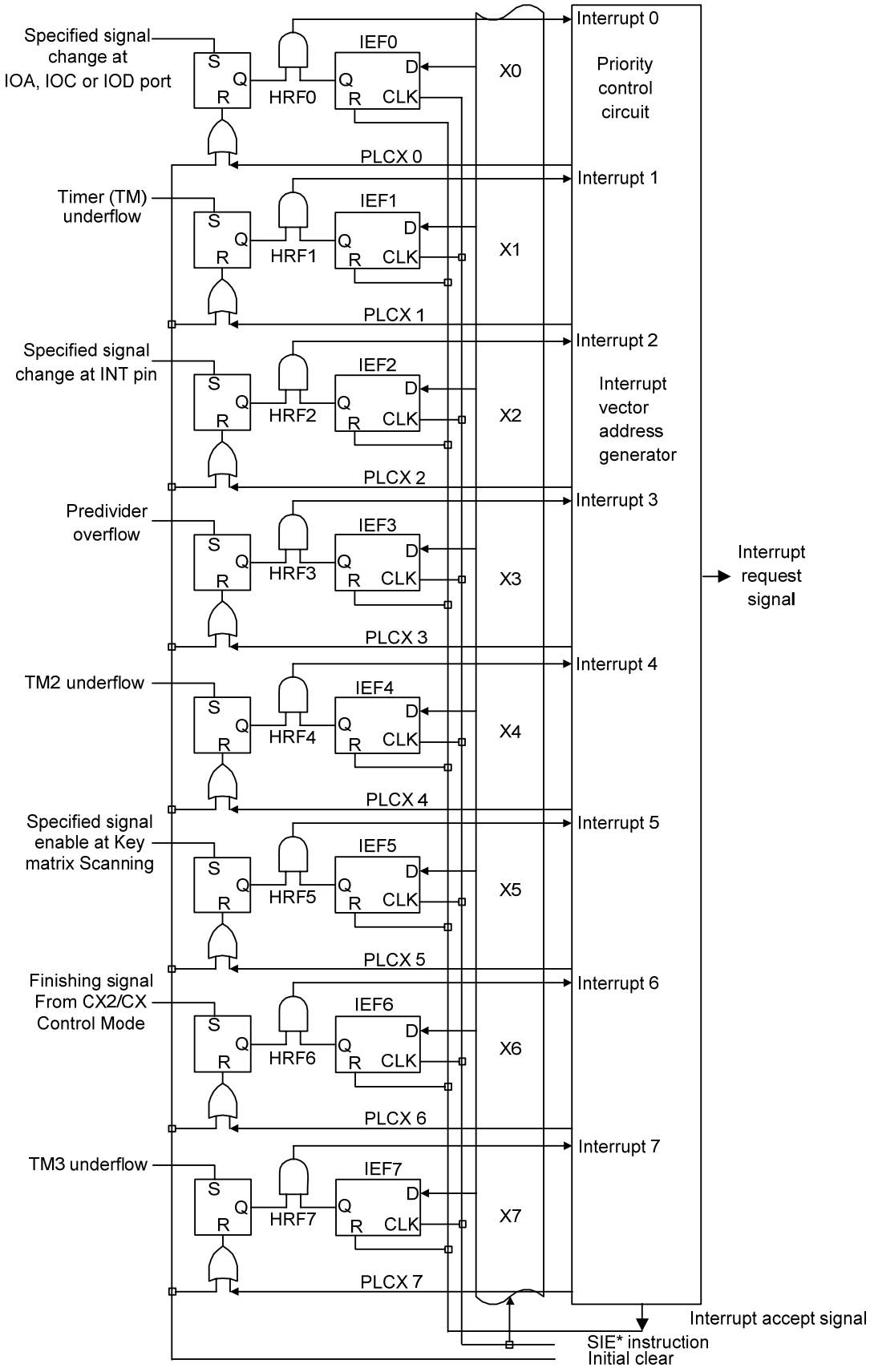
2. When executing the SRX, SRY, SLZ and SPBK instructions as well as the instruction comes after these instructions.

3. When executing the CAC and JAC instructions with 8 machine cycle.

4. When executing the CPHL, CPHLH, CPZR and CPZRH instructions as well as the instruction comes after these instructions.

5. When executing SIE, RTS instructions.

The diagram below shows the control circuitry for the interrupt function:



2-1-1. Interrupt Requests and Interrupt Servicing

2-1-1-1. External Interrupt Factor

The external interrupt factors are: INT pin, IOA, IOC or IOD ports and the Key matrix scanning function's input pin.

1. INT input pin's interrupt request
   The user can use a mask option to select whether an interrupt is generated for the MCU on the rising or falling edge of the INT pin's input signal.

   After the interrupt enable flag 2 (IEF2) is set to 1, a change in the signal on the INT input pin will set the HRF2 flag to 1 and send an interrupt request (interrupt 2) to the MCU. After the MCU accepts the interrupt request it will jump to the vector 10h to execute the interrupt service routine for interrupt 2.

   The change in signal on the INT input pin must be maintained for more than one machine cycle to be sure of generating an interrupt.

2. IOA, C, D ports' interrupt request
   After executing the SCA instruction to set control register 1 (SEF5, 4, 3), the MCU will set the interrupt request signal (HRF0) to 1 if there is any signal change in the "OR" result for the input pin signals of IO ports such as IOA, IOC and IOD (with chattering prevention function is enabled) or any of the input signals changes to a high voltage level (when high voltage level detection function is enabled).

   If the SIE* instruction had already set the interrupt enable flag 0 (IEF0) to 1 then the MCU will accept this interrupt request (interrupt 0). After the MCU accepts this interrupt request it will jump to the vector 14h to execute the interrupt service routine for interrupt 0.

3. Key matrix scanning function's interrupt request
   When the halt release enable flag 5 (HEF5) is set to 1, at the end of each key matrix scanning cycle or when the key matrix scanning circuit on pins KI1~4 detect a high voltage level input signal (set by the corresponding instructions for the key matrix scanning function) this causes the MCU to set the interrupt request signal (HRF5) to 1.

   If the SIE* instruction had already set the interrupt enable flag 5 (IEF5) to 1 then the MCU will accept this interrupt request (interrupt 5). After the MCU accepts the interrupt request it will jump to the vector 24h to execute the interrupt service routine for interrupt 5.

2-1-1-2. Internal Interrupt Factor

The internal interrupt factors are: timer 1 (TMR1), timer 2 (TMR2), timer 3 (TMR3), RFC counter and pre-divider.

1. Timers' (TMR1, 2, 3) interrupt request
   When an underflow occurs at TMR 1, 2 or 3 the interrupt request signal (HRF1, 4, 7) is set to 1.

   If the SIE* instruction had already set the interrupt enable flag 1, 4, 7 (IEF1, 4, 7) to 1 then the MCU will accept these interrupt requests (interrupt 1, 4, 7). After the MCU accepts the interrupt requests it will jump to the vectors 18h, 20h or 2Ch to execute the interrupt service routine for interrupt 1, 4 or 7.

2. Pre-divider's interrupt request
When an overflow occurs at the pre-divider the interrupt request signal (HRF3) is set to 1.

If the SIE* instruction had already set the interrupt enable flag 3 (IEF3) to 1 then the MCU will accept this interrupt request (interrupt 3). After the MCU accepts the interrupt request it will jump to the vector 1Ch to execute the interrupt service routine for interrupt 3.

3. RFC counter's (in CX/CX2 control mode) interrupt request
When the RFC counter is enabled/disabled by CX or CX2 pins, when the control period on CX or CX2 ends the interrupt request signal (HRF6) is set to 1.

If the SIE* instruction had already set the interrupt enable flag 6 (IEF6) to 1 then the MCU will accept this interrupt request (interrupt 6). After the MCU accepts the interrupt request it will jump to the vector 28h to execute the interrupt service routine for interrupt 6.

2-1-2. Interrupt Priority

When interrupt requests happen simultaneously, the MCU will process the interrupt request with the highest priority (pre-divider) first and temporarily ignore the other interrupt requests.

The table below lists the priority of each interrupt factor:

Interrupt factor	INT pin	IOA,C,D port	TMR1 underflow	Pre-divider Overflow	TMR2 Underflow	Key matrix Scanning	End of RFC counter controlled signal	TMR3 underflow
Interrupt priority	7th	6th	2nd	1st	3rd	8th	5th	4th

The follow example assumes that all interrupt requests occurred at the same time. All IEF flags have been set to 1 and all IOC ports have been set to input mode as well.

```
PLC     $FF         ; clear all HRFn flag

SCA     $10         ; enable IOC port's switch enable flag

SIE*    $FF         ; set all interrupt enable flags
;……                ; all interrupt requests happen at same time
                    ; MCU accepts interrupt request from predivider overflow,
                    ; completes interrupt service routine.

SIE*    $F7         ; set all interrupt enable flags (except pre-divider)
                    ; MCU accepts interrupt request from TMR1 underflow,
                    ; completes interrupt service routine.

SIE*    $F5         ; set all interrupt enable flags (except pre-divider, TMR1)
                    ; MCU accepts interrupt request from TMR2 underflow,
                    ; completes interrupt service routine.
```

```
SIE*    $E5         ; set all interrupt enable flags (except pre-divider, TMR1, TMR2)
                    ; MCU accepts interrupt request from TMR3 underflow,
                    ; completes interrupt service routine.

SIE*    $65         ; set all interrupt enable flags (except pre-divider, TMR1, TMR2, TMR3)
                    ; MCU accepts interrupt request from CX/CX2 controlled RFC
                    ; counter, completes interrupt service routine.

SIE*    $25         ; set all interrupt enable flags (except pre-divider, TMR1, TMR2,
                    ; TMR3, RFC counter)
                    ; MCU accepts interrupt request from IOC port, then completes
                    ; interrupt service routine.

SIE*    $24         ; set all interrupt enable flags (except pre-divider, TMR1, TMR2,
                    ; TMR3, RFC counter, IOC port)
                    ; MCU accepts interrupt request from INT port, then completes
                    ; interrupt servicing.

SIE*    $20         ; set all interrupt enable flags (except pre-divider, TMR1, TMR2,
                    ; TMR3, RFC counter, IOC port, INT pin)
                    ; MCU accepts interrupt request from Key matrix scanning function,
                    ; completes interrupt service routine.
                    ; at this point all interrupt requests have been processed.
```

2-1-3. INTERRUPT SERVICING

When the MCU accepts an interrupt request it will suspend the program being currently executed then jump to the interrupt address to execute the interrupt service routine. This involves the MCU carrying out the following actions:

1. Write the address of the instruction being executed at the time of the interrupt request to the stack register (STACK).

2. Load the interrupt address corresponding to the interrupt request into the program counter (PC).

3. Clear the interrupt request's corresponding interrupt request flag (HRFn) and also clear all interrupt enable flags (IEFn).

After the MCU accepts an interrupt request the program will follow the procedure below:

```
Instruction 1       ; when MCU receives interrupt request while executing this instruction,
                    ; store the address of the current instruction to STACK.
NOP                 ; load the corresponding interrupt address into PC and execute a NOP
                    ; instruction.
Instruction A       ; program jumps to interrupt service routine.
Instruction B
Instruction C
```

```
.............
RTS                 ; interrupt service routine finished.
Instruction 1*      ; MCU re-executes this instruction it was executing at the time of
interrupt
                    ; request.
Instruction 2
```

When the interrupt service routine begins executing, all interrupt enable flags (IEF0~IEF7) are cleared. If the program needs to service other interrupt requests as well then the SIE* instruction must be re-executed inside the interrupt service routine to set these flags.

2-2. RESET FUNCTION

The MCU will enter the RESET state upon receiving a reset signal generated by one of these four sources: MCU's power on reset, RESETB pin, external key reset, watch dog timer overflow.

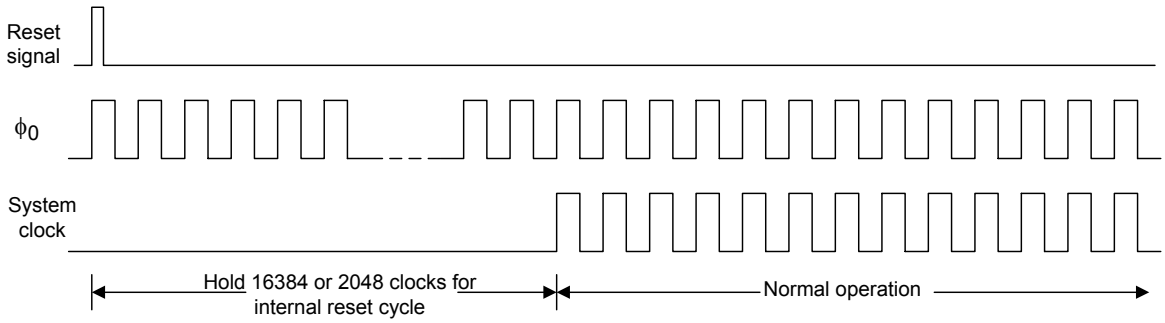
The table below lists the initial state of flags and functions when MCU enters RESET state.

Program counter	(PC)	Address 0000H
Start condition flags	(SCF1~8, 10~12)	0
Backup flag	(BCF)	1
Stop release enable flags	(SRF3,4,6)	0
Switch enable flags	(SEF0,1,3,4,5)	0
Halt release request flags	(HRF 0~7)	0
Halt release enable flags	(HEF1-5,7)	0
Interrupt enable flags	(IEF0-7)	0
BZ,BZB pin output		DC 0
Pull-low RC on I/OA~E port		Turn On
Input/output ports	(PORT I/OA, I/OB, I/OC, I/OD, I/OE)	Input mode
I/OA,C,D port chattering clock	Cch	PH10*
EL panel driver pumping clock source and duty cycle	Celp	PH0, duty cycle is 1/4
EL panel driver clearing clock source and duty cycle	Celc	PH8, duty cycle is 1/4
EL plant driver output pins	ELC, ELP	DC0
Frequency generator clock source and duty cycle	Cfq	PH0, duty cycle is 1/4, stopped
Resistor frequency converter	(RFC)	Stopped, RFC0~5 output 0
LCD driver output waveform		All OFF waveform
Key matrix scanning function		Stopped, KO1~16=1
Timer 1/2/3		Stopped
Watch dog timer	(WDT)	Stopped
Watch dog timer enable flag	(WDF)	WDF = 0
System clock's clock source (Dual Clock mode)	(BCLK)	XT clock

Note: PH10: 10^{th} stage output signal from pre-divider.

After the MCU receives the reset signal it will enter the RESET state then begin counting the RESET interval. When the RESET interval count is complete the MCU will enter normal operation.

Mask option can be used to choose between two RESET interval counts. One is the 16384 PH0 clock cycles (PH15/2) and the other is 2048 PH0 clock cycles (PH12/2).



A detailed explanation of the reset sources is provided below:

2-2-1. MCU's POWER ON RESET

When an external power supply reaches the MCU's VBAT power supply pin, or when the power supply voltage drops to below 0.6V then rises back up to the normal operating voltage, this causes the MCU to generate a power on reset signal. The MCU then enters the RESET state and begins the RESET interval count.

When the RESET interval count finishes the MCU will enter normal operation.

Mask option can be used to enable/disable the power-on reset function.

2-2-2. RESET PIN

When a low-voltage level input signal arrives at the RESETB input pin this causes the MCU to enter the RESET state.

The RESETB input pin has a built-in pull-up resistor. If a 0.1uF capacitor is connected between the RESETB pin and the power supply GND, this RC circuit can be used to generate a power on reset signal on the RESETB pin.

MCU can handle the input signal on the RESETB in two ways: level reset or pulse reset. The user can choose between the two methods using the mask option. An explanation is provided below:

2-2-2-1. LEVEL RESET

After selecting this method, when a low voltage level signal arrives at the RESETB pin the MCU will enter the RESET state but does not begin the RESET interval count immediately. The count will only begin after the signal on the RESETB pin changes to a high voltage level.

When the RESET interval count finishes the MCU will enter normal operation.

2-2-2-2. PULSE RESET

After selecting this method, when a low voltage level signal arrives at the RESETB pin the MCU will enter the RESET state and begin the RESET interval count immediately.

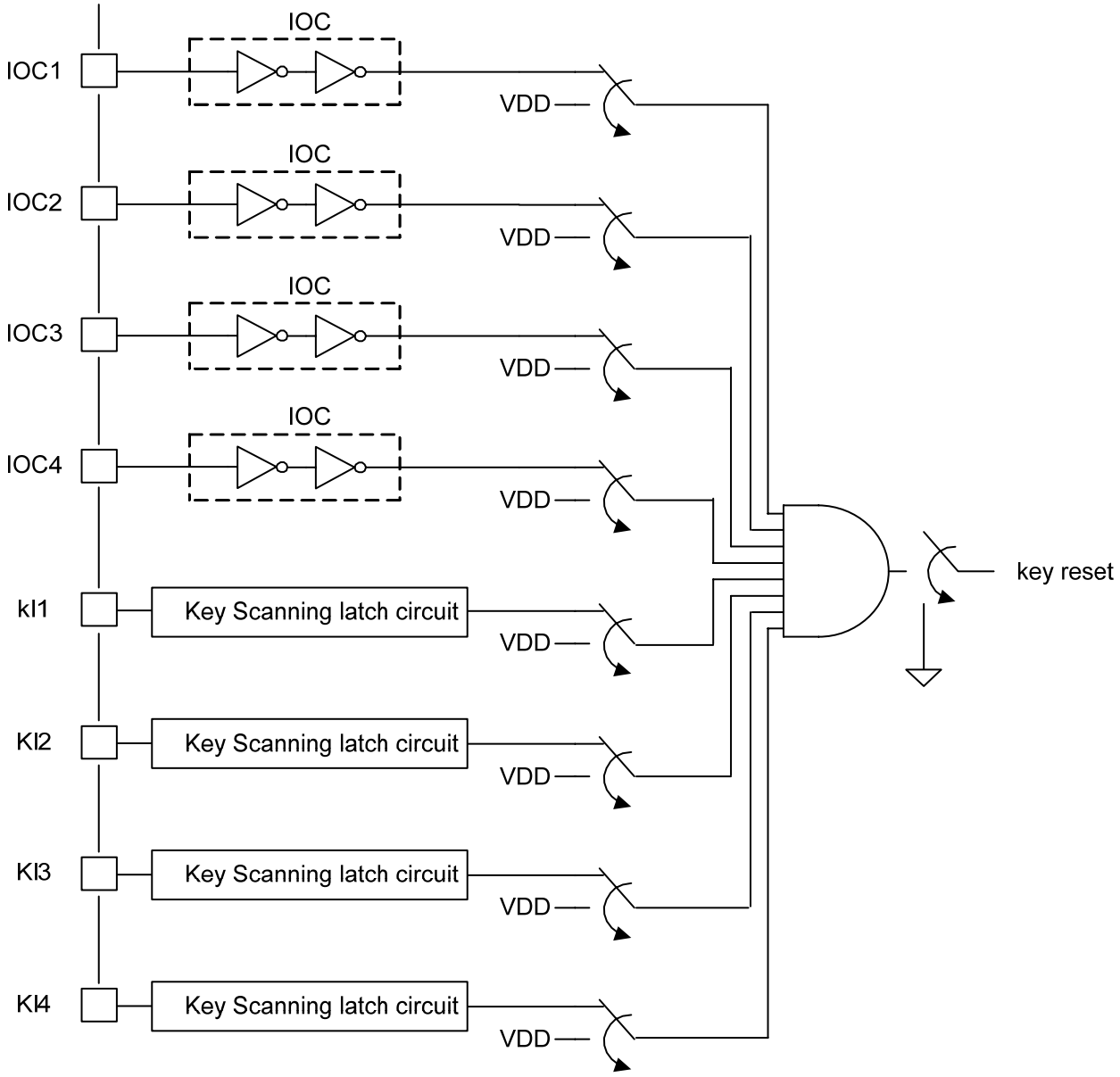
It does not matter whether the signal on the RESETB changes to a high voltage level or not. Once the RESET interval count is complete the MCU enters normal operation.

2-2-3. EXTERNAL KEY RESET

After selecting this method, having high voltage level signals appear simultaneously on several specific input pins will generate a MCU reset signal. These input pins are selected using the mask option from certain IOC port input pins or the key matrix scanning function's input pins (KI1~KI4).

When a high voltage level signal input appears simultaneously on these chosen input pins (selected IOC input pins and KI input pins), the MCU will enter RESET state. When one of the input signals changes to a low voltage level the MCU will begin the RESET interval count. When the RESET interval count finishes the MCU will enter normal operation.

The diagram below sets out the circuit architecture for this function:

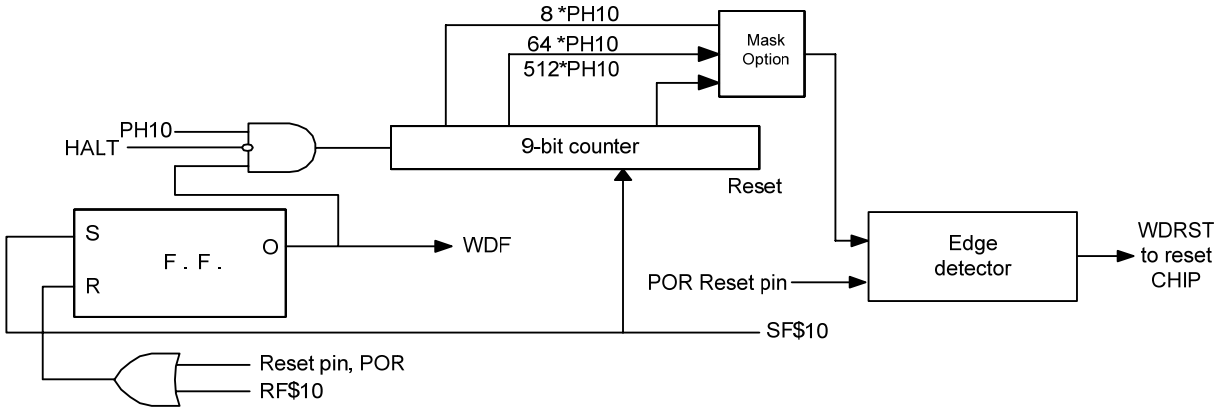


2-2-4. WATCH DOG TIMER OVERFLOW

The watch dog timer is a 9-bit counter that uses the 10th stage output signal (PH10) from the pre-divider as its clock source.

When an overflow occurs at the watch dog timer, the MCU will immediately enter RESET state and begin the RESET interval count. When the RESET interval count finishes the MCU will enter normal operation.

The diagram below sets out the circuit architecture for the watch dog timer:



If the MCU enters RESET state due to power on reset, reset pin or external key reset, the MCU will disable the watch dog timer and clear the watch dog timer enable flag (WDF).

If the MCU enters RESET state due to a watch dog timer overflow, the watch dog timer will continue to run. The watch dog timer enable flag (WDF) will not be cleared and the pre-divider's first 10 stages of output signals (PH0~PH10) will not be reset to zero either.

Executing the SF $10 instruction will reset and re-start the watch dog timer and also set the watch dog timer enable flag (WDF) to 1. Once the watch dog timer is started, the program must execute the SF 10h instruction again to reset the timer to zero before an overflow occurs.

When the watch dog timer is running, if the MCU enters HALT or STOP mode the watch dog timer will temporarily stop counting. The timer will only resume counting when the MCU generates a HALT release or STOP release. We recommend executing the SF 10h instruction once before the MCU enters HALT or STOP mode to set the timer to zero. This avoids a watch dog timer overflow happening right after the program leaves HALT or STOP mode.

Executing the RF $10 instruction disables the watch dog timer and clears the watch dog timer enable flag (WDF).

Three types of overflow intervals can be set for the watch dog timer: 8 PH10 clock cycles, 64 PH10 clock cycles or 512 PH10 clock cycles. These can be selected using the mask option.

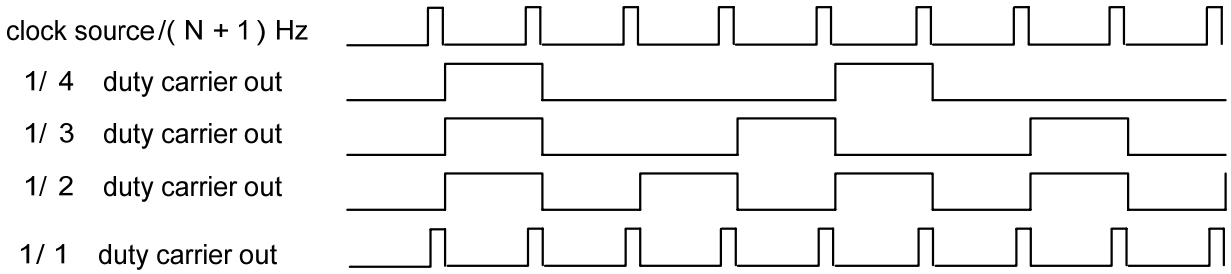
2-3. FREQUENCY GENERATOR

The frequency generator can be set to generate output signals of different frequencies or duty cycles using different instruction operands. These can be used as the clock source of the buzzer output, EL plant driver, TMR1, TMR2, TMR3 and RFC counter.

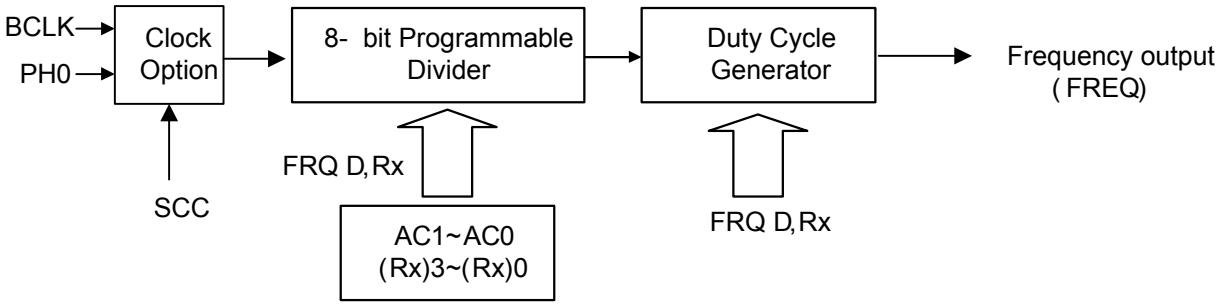
The frequency generator can select BCLK or PH10 as its clock source. The clock source signal is then passed through the pre-scaler to divide the signal frequency by N+1 (N is the preset divisor). The generated waveform will be composed of one high voltage level and N low voltage levels based on the clock source signal period. When the preset divisor (N) is set to 0, the signal waveform generated by the pre-scaler will be identical to the clock source waveform itself.

After the signal generated by the pre-scaler is passed through a duty cycle generator this produces the required signal (FREQ).

The diagram below lists the different duty cycle waveforms produced by the frequency generator:



The frequency generator's function block diagram is shown below:



Executing the SCC instruction chooses between BCLK or PH0 as the clock source of the frequency generator.

Executing FRQ related instructions sets the preset divisor (N) and duty cycle (D) values for the pre-scaler to produce the desired output signal. The frequency of the output signal can be calculated using the formula below:

Frequency Generator's Output Frequency = (Clock source frequency) / ((N+1) * X) Hz.

N: The preset divisor set by the FRQ related instructions

X: 1~4, represents 1/1, 1/2, 1/3 or 1/4 duty

In the instruction operand, the value of preset divisor (N) can come from immediate data, table ROM or both of AC and data memory.

The table below lists the relationship between bits of the operand and the preset divisor (N):

	The bit pattern of preset data N							
FRQ related instructions	bit7	Bit6	bit 5	bit 4	bit 3	Bit 2	bit 1	bit 0
FRQ D,Rx	AC3	AC2	AC1	AC0	(Rx)3	(Rx)2	(Rx)1	(Rx)0
FRQ D,@HL	TD7	TD6	TD5	TD4	TD3	TD2	TD1	TD0
FRQX D,X	X7	X6	X5	X4	X3	X2	X1	X0

Notes:

1. TD0 ~ TD7 represents the data of table ROM.
2. X0 ~ X7 is the immediate data in operand

The table below lists the relationship between the bits of operand and the set duty cycle value (D):

Preset Letter D		Duty Cycle
D1	D0	
0	0	1/4 duty
0	1	1/3 duty
1	0	1/2 duty
1	1	1/1 duty

Setting the frequency generator does not immediately activate it and it is not ordinarily active either. Only after the Timer, RFC, EL panel driver or ALM functions are enabled and using FREQ as their signal input or after executing the SBZ instruction (X1=1 or X0=1) then the frequency generator's clock source (Cfq) will be enabled to let the FREQ signal output. When these functions are disabled or turned off, the FREQ output signal ceases as well.

Special care must be taken when using the FREQ output signal as the base frequency for a timer. When a timer sets FREQ as its clock source, if there's a timer underflow and there are no other function mentioned above is enabled to keep the frequency generator active, the FREQ will stop outputting clocks. Frequency generator will only resume to output clocks in the next time the timer is started. This affects the time interval of timer underflow every time, making the time interval inaccurate.

If this function is truly required, the continued operation of the frequency generator must be maintained by enabling other related functions.

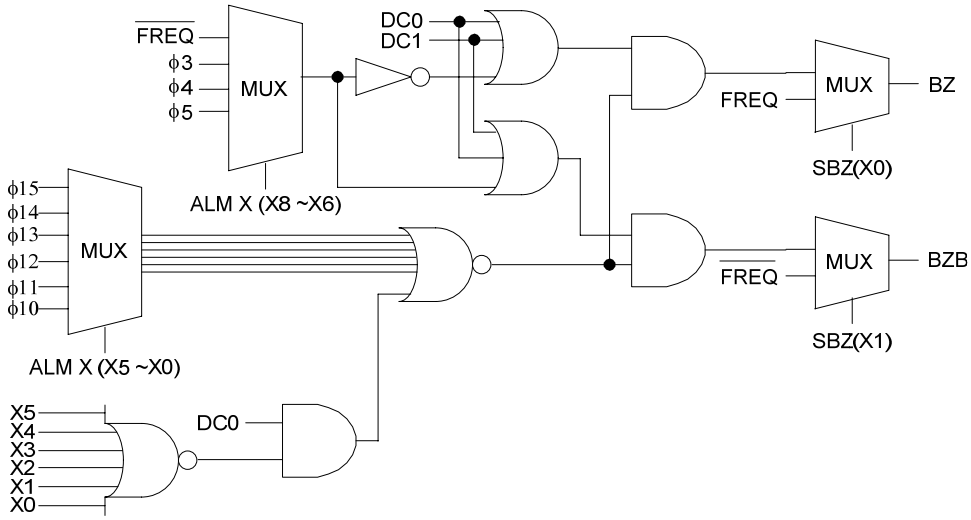
2-4. BUZZER OUTPUT

The Buzzer Output function uses the two output pins BZB and BZ to drive the buzzer components and make sound. The BZB and BZ output signals are opposites of each other.

Executing the SBZ instruction can pass the output signal of frequency generator or a modulated signal set by the ALM instruction to the BZ and BZB pins.

If the buzzer output function's signal source is the frequency generator, it can generate the carrier outputs used by the remote controller or single tone melodies. If the signal source is the modulated signal set by the ALM instruction, it can generate a sound effect function.

A detailed description of these functions' usage is provided below.



This figure shows the organization of the buzzer output.

2-4-1. SOUND EFFECT

The Sound Effect function can generate a modulated signal created by multiplexing high-frequency signals (carrier signal) such as FREQ (output signal from frequency generator), PH3 (1024Hz), PH4 (2048Hz) or PH5 (1024Hz) and low-frequency signals (envelope signal) such as PH10 (32Hz), PH11 (16Hz), PH12 (8Hz), PH13 (4Hz), PH14 (2Hz) and PH15 (1Hz).

Executing the ALM instruction can set the combinations for these signals. Please note the following items when setting the multiplexed modulated signal:

1. Only one signal among PH3, PH4, PH5 or FREQ can be set as the carrier signal; any PH10 ~ PH15 signals can be used for the envelope signal (multiple signals can be selected at the same time).

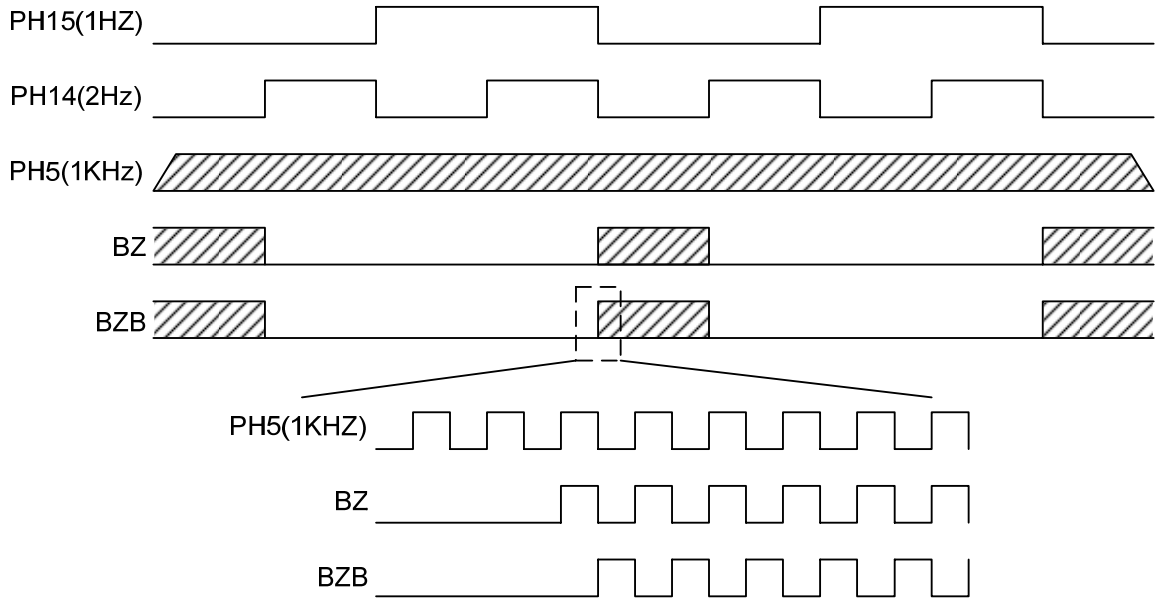
2. The frequencies in parentheses above are calculated using PH0 = 32768 Hz as the base.

3. When the MCU enters RESET state, BZ and BZB will simultaneously output a low level signal (DC0).

The example below is of a modulated signal output by the buzzer function made up of (PH5) carrier) and PH15, PH14 (envelope).

```
.....
ALM    $070          ; Output the waveform.
.....
```

The diagram below explains the modulated signal output by the example:



2-4-2. CARRIER OUTPUT used by Remote Controller

The Buzzer output function can be used in conjunction with the timer and frequency generator to produce the carrier signal needed by the remote controller.

In infra-red remote controls, the preset divisor value of the frequency generator must be greater than 3. If the frequency generator's output signal frequency is set too high, BZ or BZB's output signal can't be set in time to DC0 after a timer underflow. This makes BZ or BZB send out several extra carrier signals.

Additionally, the ALM instruction must be executed right after FRQ related instructions so the output signal (FREQ) from the frequency generator can be output by the BZ or BZB pin for use as the infra-red remote control's carrier signal.

Example:

```
SHE       1          ; set TMR1's halt release enable flag.
TMSX      $3F        ; TMR1's preset data is 3FH, clock source set to PH9, start TMR1
SCC       $40        ; select BCLK as frequency generator's clock source
FRQX      2, 3       ; set frequency generator's preset divisor to 3, duty cycle is 1/2
ALM       $1C0       ; output frequency generator's output signal (FREQ) to BZ pin
HALT                 ; wait until TMR1 generates HALT release request
.............

ALM       0          ; BZ pin outputs DC0 signal
```

2-4-3. SINGLE TONE MELODY

If the frequency generator is set to a specific preset divisor (N) and 1/2 duty cycle, this can be used with the buzzer output function to generate a single frequency tone.

The table below lists the preset divisor (N) required for each tone:

Tone	N	FREQ	Ideal	%	Tone	N	FREQ	Ideal	%
C2	249	65.536	65.4064	0.19	C4	62	260.063	261.626	-0.6
#C2	235	69.4237	69.2957	0.18	#C4	58	277.695	277.183	0.18
D2	222	73.4709	73.4162	0.07	D4	55	292.571	293.665	-0.37
#D2	210	77.6493	77.7817	-0.17	#D4	52	309.132	311.127	-0.64
E2	198	82.3317	82.4069	-0.09	E4	49	327.68	329.628	-0.59
F2	187	87.1489	87.3071	-0.18	F4	46	348.596	349.228	-0.18
#F2	176	92.565	92.4986	0.07	#F4	43	372.364	369.994	0.64
G2	166	98.1078	97.9989	0.11	G4	41	390.095	391.995	-0.48
#G2	157	103.696	103.826	-0.13	#G4	38	420.103	415.305	1.16
A2	148	109.96	110	-0.04	A4	36	442.811	440	0.64
#A2	140	116.199	116.541	-0.29	#A4	34	468.114	466.164	0.42
B2	132	123.188	123.471	-0.23	B4	32	496.485	493.883	0.53
C3	124	131.072	130.813	0.2	C5	30	528.516	523.251	1.01
#C3	117	138.847	138.591	0.19	#C5	29	546.133	554.365	-1.48
D3	111	146.286	146.832	-0.37	D5	27	585.143	587.33	-0.37
#D3	104	156.038	155.563	0.31	#D5	25	630.154	622.254	1.27
E3	98	165.495	164.814	0.41	E5	24	655.36	659.255	-0.59
F3	93	174.298	174.614	-0.18	F5	22	712.348	698.456	1.99
#F3	88	184.09	184.997	-0.49	#F5	21	744.727	739.989	0.64
G3	83	195.048	195.998	-0.48	G5	20	780.19	783.991	-0.48
#G3	78	207.392	207.652	-0.13	#G5	19	819.2	830.609	-1.37
A3	73	221.405	220	0.64	A5	18	862.316	880	-2.01
#A3	69	234.057	233.082	0.42	#A5	17	910.222	932.328	-2.37
B3	65	248.242	246.942	0.53	B5	16	963.765	987.767	-2.43

Notes:

1. Above variation does not include X'tal variation.

2. If PH0 = 65536Hz, C3 - B5 may have more accurate frequency.

3. The clock source is PH0, i.e. 32,768 Hz.

4. The duty cycle is 1/2 Duty (D=2).

5. “FREQ” is the output frequency.

6. “ideal” is the ideal tone frequency.

7. “%” is the frequency deviation.

2-5. IO PORTS

The TM89 Series MCU provides five IO ports: IOA, IOB, IOC, IOD and IOE. Each IO port has four IO pins.

If an IO port shares the pins with other functions, when mask option allocates a shared pin to another function then function of that IO port pin will be automatically disabled.

2-5-1. IOA PORT

IOA port's four pins can be set independently to input or output mode using the SPA instruction. However, when the MCU exits RESET state all IOA port pins will be automatically set to input mode.

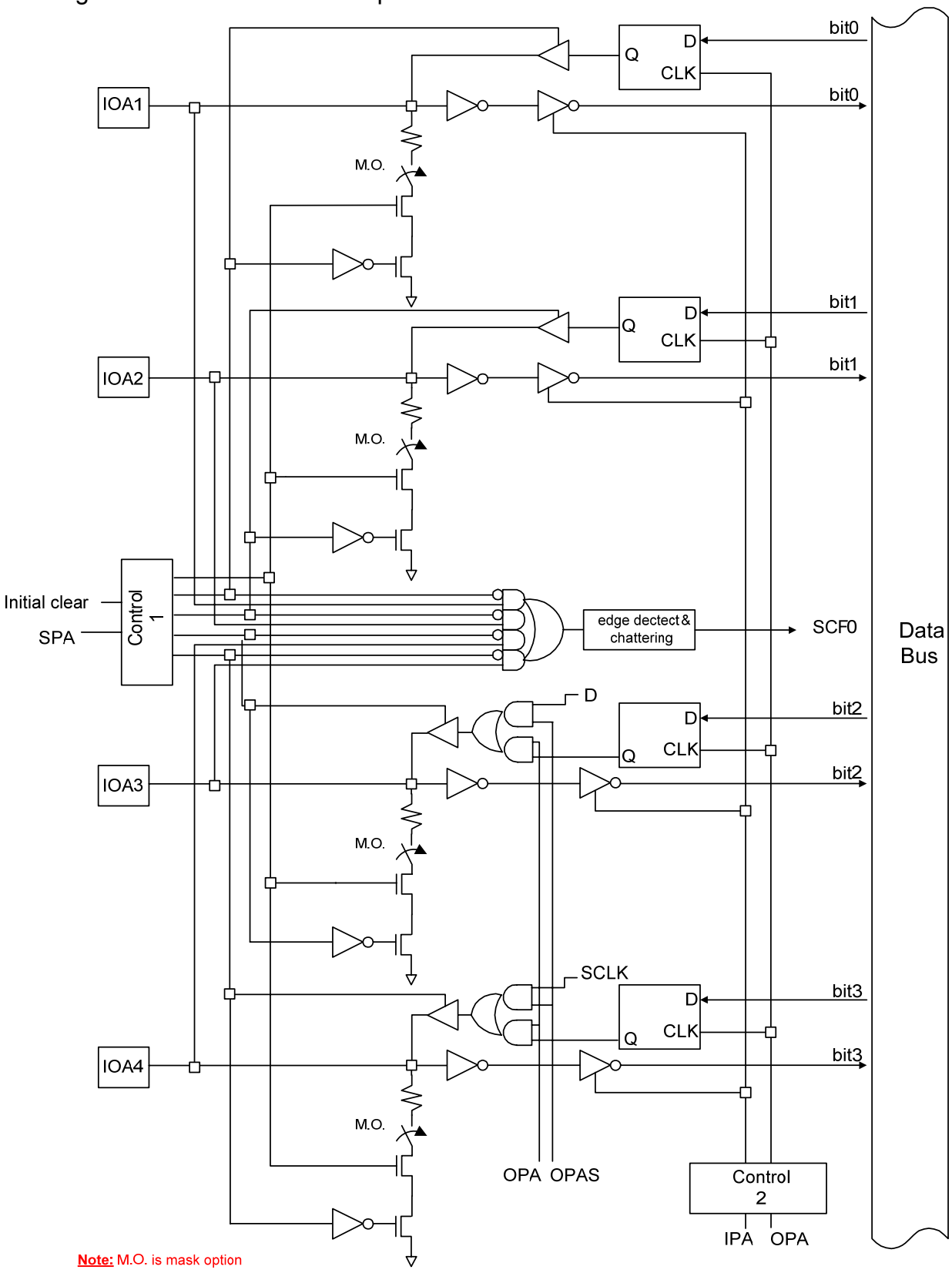
Executing the OPA instruction writes the contents of the data memory to the IOA port's output register. If the IOA port is already set to output mode then the contents of the output register will be output to the IOA pins.

Executing the IPA instruction stores the signal on the IOA port pins to data memory. If the IOA port is already set to output mode, executing the IPA instruction stores the contents of the output register to data memory.

Before the program can switch the IOA port to output mode it must first execute the OPA instruction to write the proposed output signal to the output register. This prevents the IOA port pins from outputting any unnecessary signals after being switched to output mode.

Under input mode all IOA port pins have a built-in pull-down resistor to protect against input signal floating. This pull-down resistor can be enabled or disabled using mask option. In input mode, signal floating on the input pin will cause an unexpected current to flow through the input buffer.

The diagram below shows the IOA port architecture:



Note: M.O. is mask option

2-5-1-1. PSEUDO SERIAL OUTPUT

Executing the OPAS instruction simulates a serial output mode (pseudo serial output mode) with the IOA port output function. The IOA port must be set to output mode before executing this instruction.

When using the IOA port's pseudo serial output function, the output signal on each output pin is as shown below:

(1). IOA1 and IOA2 will output bit0 and bit1 of data in data memory.

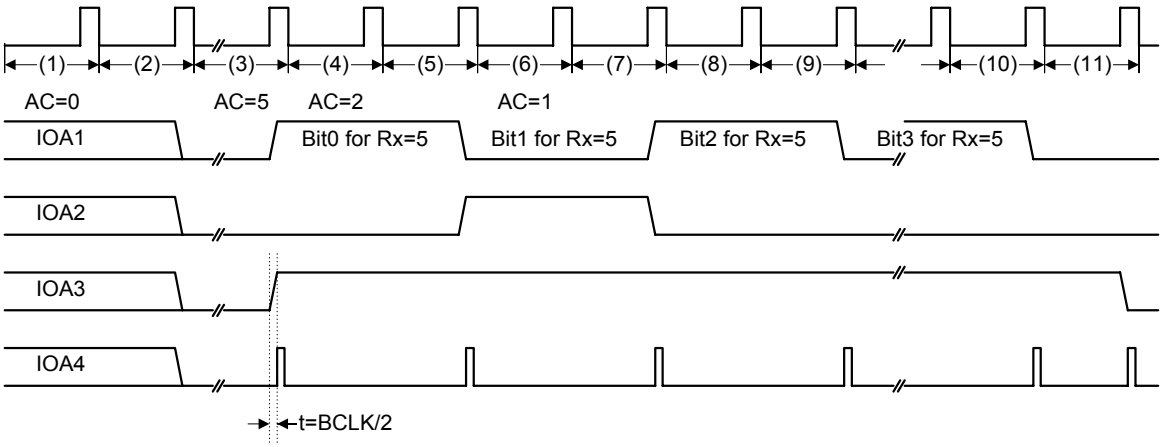
(2). IOA3 will output the immediate data (D) in the OPAS instruction operand.

(3). IOA4 will send a pulse signal with a width half that of the system clock cycle (BCLK/2) before the end of the OPAS instruction cycle.

An example of the OPAS instruction is provided below:

```
(1)     LDS    $0A, 0
(2)     OPA    $0A
        SPA    $0F
          :
          :
        LDS    1,5          ; output 0101b to IOA1, IOA3 as the enabled signal of
                            ; shift operation
(3)     OPAS   1,1          ; Bit 0 output, shift operation is enabled
(4)     SR0    1            ; Shifts bit 1 to bit 0
(5)     OPAS   1,1          ; Bit 1 output
(6)     SR0    1            ; Shifts bit2 to bit 0
(7)     OPAS   1,1          ; Bit 2 output
(8)     SR0    1            ; Shifts bit 3 to bit 0
(9)     OPAS   1,1          ; Bit 3 output
          :
          :
(10)    OPAS   1,1          ; Last data
(11)    OPAS   1,0          ; Shift operation is disabled
```

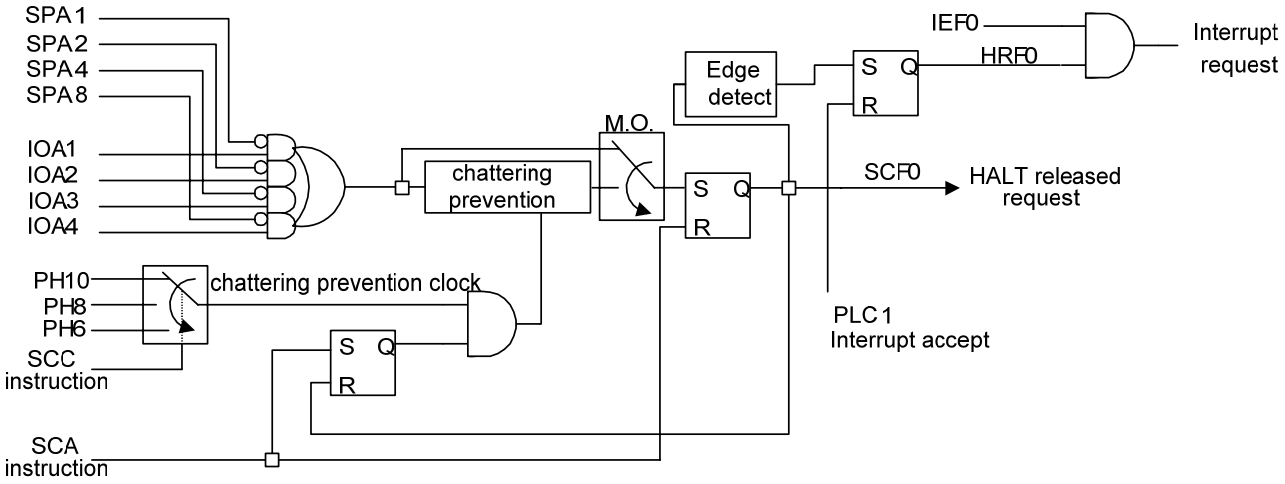
The diagram below explains the relationship between the output signals on each IOA output pin in the above example:



2-5-1-2. Setting the START CONDITION FLAG 0 (SCF0)

There are two ways for the IOA port to set the start condition flag 0 (SCF0). One is direct setting by external signal (also known as high voltage level detection) and the other is to pass the external signal through the chattering prevention circuit (also known as chattering prevention). The user can choose between the two methods using the mask option.

The diagram below shows the IOA port's associated control circuitry and flag:



Note: The default prevention clock is PH10

As the IOA port's four signal inputs must pass through the OR logic gate before it can be tested by the high voltage level detector or chattering prevention function, only when three IOA port input signals are simultaneously at a low voltage level and one input signal changes will there be a change in the output signal from the OR logic gate.

2-5-1-2-1. Direct setting by external signal

When mask option selects the use of the high voltage level detector, if the SCA instruction has already set the switch enable flag 5 (SEF5) to 1, a change from low to high voltage level in the IOA port input signal will set the start condition flag 0 (SCF0) to 1.

Once SCF0 is set to 1 the MCU will be unable to detect any further changes in the IOA port signal. All IOA port input pin signals must therefore be set to 0 and the SCA instruction executed again to clear SCF0. This allows the SCF0 to be set again the next time there's a change in the IOA port signal.

2-5-1-2-2. External signal through chattering prevention

The chattering prevention circuit has two properties: it can remove the noise from the input signal and detect changes in the input signal.

1. Removing noise from the input signal

If the noise on the input signal is smaller than a set interval, the chattering prevention circuit will remove it. (Except if the noise cycle is in synch with the clock source of chattering prevention function.)

There are three clock frequencies used for removing signal nose: PH10 (32ms), PH8 (8ms) or PH6 (2ms). These can be set using the SCC instruction and by default the MCU uses PH10.

2. Detect changes in input signal
After the chattering prevention circuit removes the noise, the rising or falling edge of this input signal will cause this circuit to set the start condition flag 0 (SCF0) to 1.

If mask option enabled the chattering prevention function, when the SCA instruction has set the switch enable flag 5 (SEF5) to 1, regardless of whether the IOA port input signal changes from a low to high voltage level or from a high to low voltage level the chattering function will always detect the change in signal state and set SCF0 to 1.

Once SCF0 is set to 1 the MCU will be unable to detect any further changes in the IOA port signal. The SCA instruction must therefore be executed again to clear SCF0. This allows the SCF0 to be set again the next time there's a change in the IOA port signal.

2-5-1-2-3. Notice for executing SCA instruction

MCU will clear all of SCF0, SCF1 and SCF3 flags in the same time while executing the SCA instruction. For this reason, the program has to check SCF1 and SCF3 flags before executing SCA instruction to avoid losing the signal changes on IOB and IOD port.

2-5-1-3. HALT RELEASE, STOP RELEASE and Interrupt Service

2-5-1-3-1. HALT RELEASE

Start condition flag 0 (SCF0) is the IOA port's halt release request signal. After SCF0 is set to 1 the halt release request flag 0 (HRF0) will be set to 1 as well, causing the MCU to generate a HALT release.

The conditions for the MCU to generate a HALT release will vary according to the method used by the IOA port to set SCF0.

1. When the IOA port chooses to use high voltage level detection to set SCF0, once SEF5 is set to 1 if any input signal on the IOA port's input pins changes to a high voltage level this will cause the MCU to generate a HALT release.
The MCU however can only enter the HALT mode when all input signals from the IO port are at the low voltage level.

2. When the IOA port chooses to use chattering prevention for setting SCF0, once SEF5 is set to 1 then any change in the "OR" result from the input signals on the IOA port input pins will cause the MCU to generate a HALT release.
When using the chattering prevention function the MCU can enter HALT mode regardless of what the IOA port's input signals are.

2-5-1-3-2. Interrupt Service

Start condition flag 0 (SCF0) is also the IOA port's interrupt request signal. If the IOA port's interrupt enable flag 0 (IEF0) is already set to 1, when the SCF0 is set to 1 it will also set the halt release request flag 0 (HRF0) to 1, causing the MCU to accept the interrupt request from the IOA port.

2-5-1-3-3. STOP RELEASE

The conditions for the MCU to generate a STOP release will vary according to the method used by the IOA port to set SCF0. The MCU can only enter the STOP mode when all input signals from the IOA port are at the low voltage level.

1. When the IOA port chooses to use high voltage level detection function to set SCF0 and SEF5 is also set to 1, if any input signal on the IOA port's input pins changes to a high voltage level this will cause the MCU to generate a STOP release.

2. When the IOA port chooses to use chattering prevention function to set start condition flag 0, only when both SEF5 and SRF6 are set to 1 will any input signal on the IOA port's input pins' change to a high voltage level cause the MCU to generate a STOP release.

After the MCU generates a STOP release it will first enter HALT mode. The high voltage level signal on the IOA port input pin must be maintained for a set time until the chattering prevention function validates this high voltage level signal and sets the start condition flag 0 to 1. Only then can it return to the low voltage level.

If this high voltage level signal reverts back to a low voltage level before SCF0 is set, the MCU will return immediately to STOP mode.

2-5-2. IOB PORT

IOB port's four pins can be set independently to input or output mode using the SPB instruction. However, when the MCU exits RESET state all IOB port pins will be automatically set to input mode.

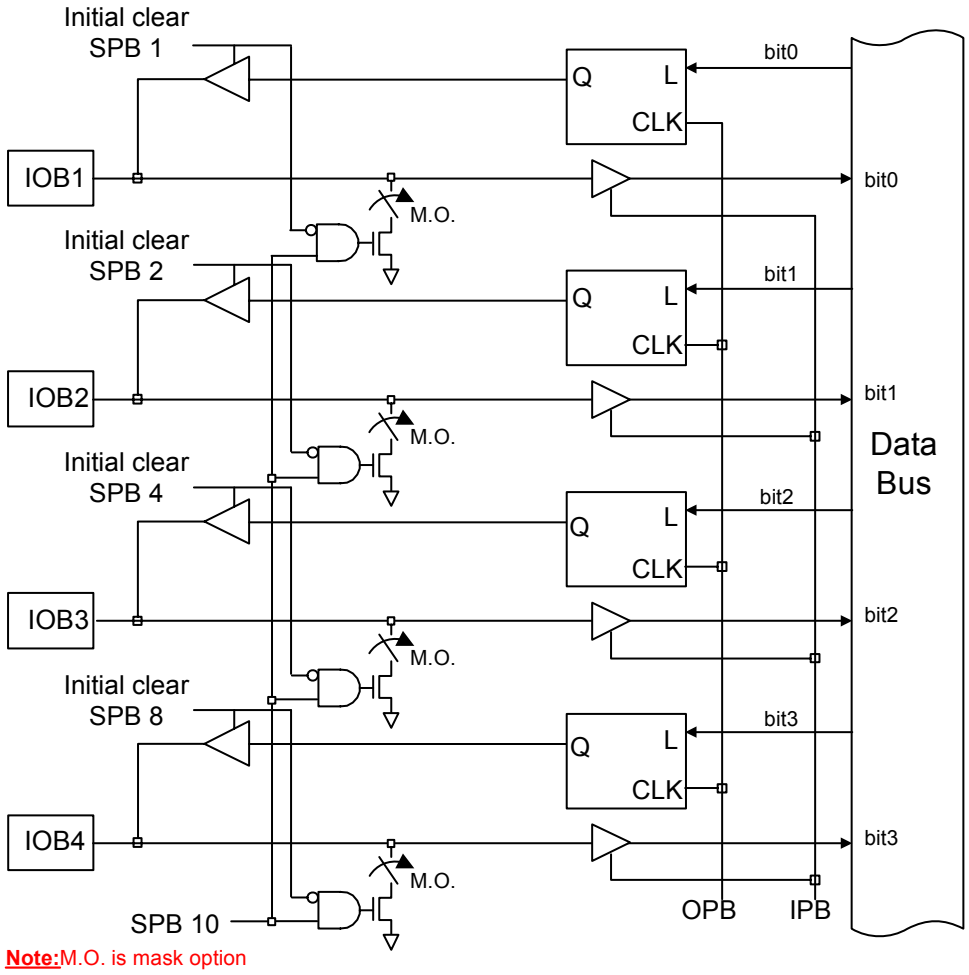
Executing the OPB instruction writes the contents of the data memory to the IOB port's output register. If the IOB port is already set to output mode then the contents of the output register will be output to the IOB pins.

Executing the IPB instruction stores the signal on the IOB port pins to data memory. If the IOB port is already set to output mode, executing the IPB instruction stores the contents of the output register to data memory.

Before the program can switch the IOB port to output mode it must first execute the OPB instruction to write the proposed output signal to the output register. This prevents the IOB port pins from outputting any unnecessary signals after being switched to output mode.

In input mode, all IOB port pins have a built-in pull-down resistor to protect against input signal floating. This pull-down resistor can be enabled or disabled using mask option. In input mode, signal floating on the input pin will cause an unexpected current to flow through the input buffer.

The diagram below shows the IOB port architecture:



2-5-3. IOC PORT

IOC port's four pins can be set independently to input or output mode using the SPC instruction. However, when the MCU exits RESET state all IOC port pins will be automatically set to input mode.

Executing the OPC instruction writes the contents of the data memory to the IOC port's output register. If the IOC port is already set to output mode then the contents of the output register will be output to the IOC pins.

Executing the IPC instruction stores the signal on the IOC port pins to data memory. If the IOC port is already set to output mode, executing the IPC instruction stores the contents of the output register to data memory.

Before the program can switch the IOC port to output mode it must first execute the OPC instruction to write the proposed output signal to the output register. This prevents the IOC port pins from outputting any unnecessary signals after being switched to output mode.

In input mode all IOC port pins have a built-in pull-down resistor and a low-level hold component to protect against input signal floating. These two components can be enabled or disabled using mask option.

The pull-down resistor can be enabled by mask option individually but the low-level hold component has to be used in conjunction with the pull-down resistor. In input mode, signal floating on the input pin will cause an unexpected current to flow through the input buffer.

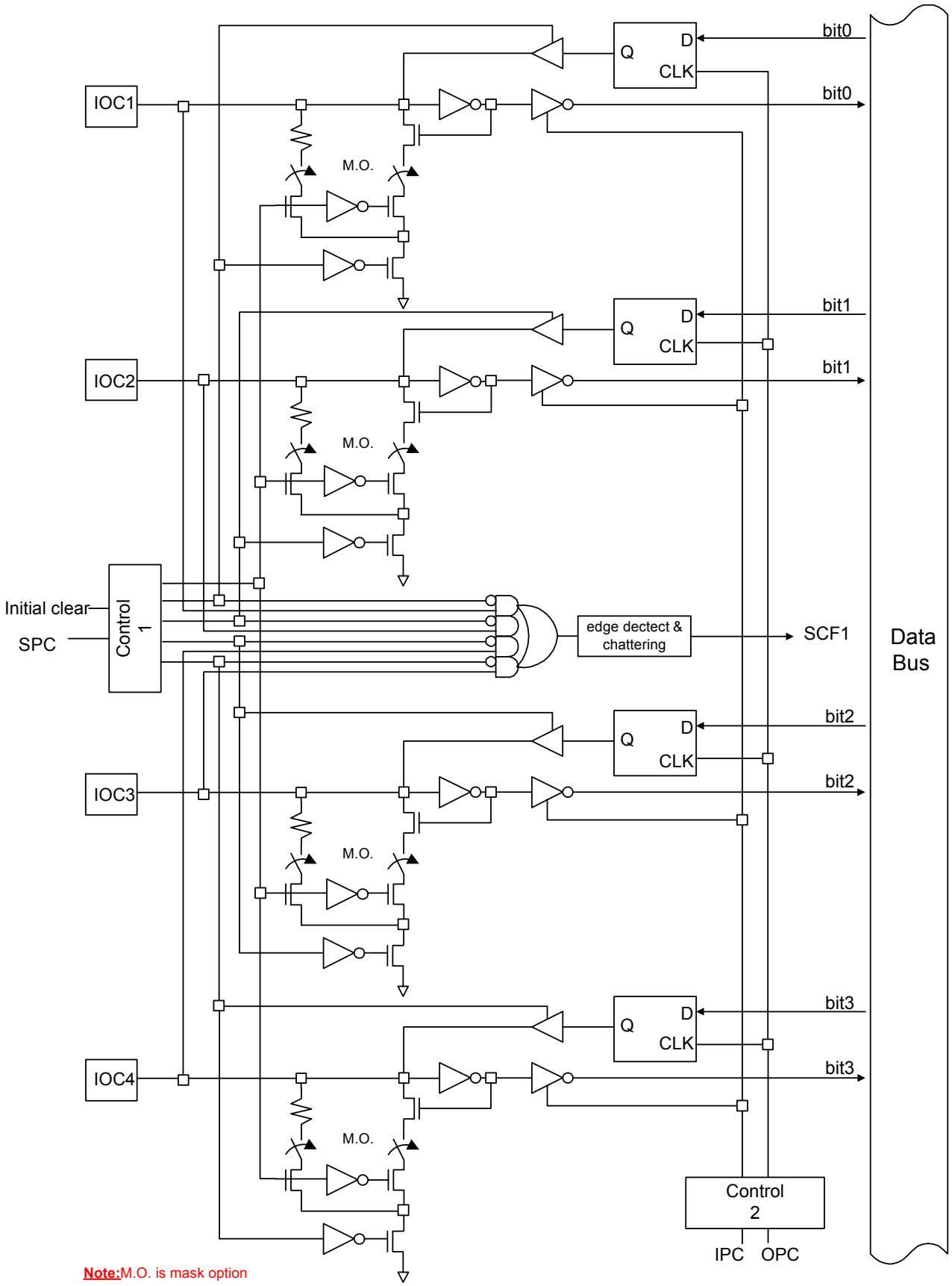
When the mask option enables both the pull-down resistor and low-level hold component at the same time, executing the SPC 10h instruction enables the pull-down resistor and disables the low-level hold component. Executing the SPC 0h instruction disables the pull-down resistor and enables the low-level hold component.

The purpose of low-level hold is to use the bonding option on the application circuit as a way of selecting the program application. On one hand, this means the PCB will only need to provide a bonding option pad for the VBAT voltage level and leave out a GND bonding option pad. On the other hand this kind of bonding option allows power conservation to still be achieved.

The usage of bonding option is as follows:

1. After the MCU exist the RESET state, first enable the pull-down resistor on the bonding option pad. If the pad is floating then use the pull-down resistor to pull it down to the GND voltage level. If pad is connected to VBAT voltage level using bonding option, pad will stay at VBAT voltage level, but there is a DC current flows through the pull-down resistor.
2. Next to execute the SPC 0h instruction (X4=0) to disable the pull-down resistor. This immediately enables the low-level hold function. If the pad is floating then the low-level hold will keep the pad at GND voltage level. If pad is connected to the VBAT voltage level using bonding option then there is no power consumption because of the pull-down resistor is disabled.

The diagram below shows the IOC port architecture:

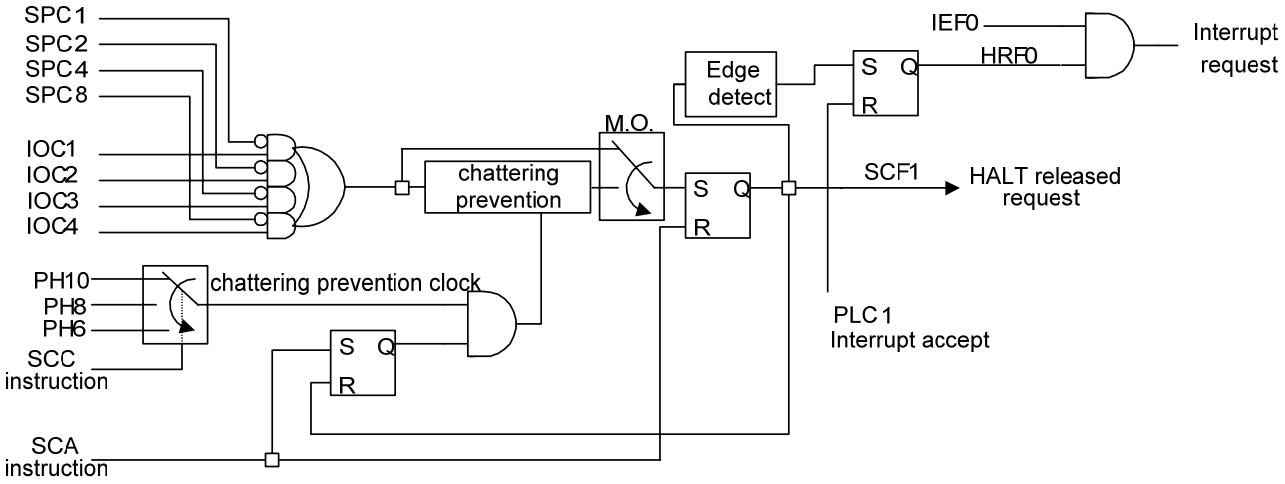


Note:M.O. is mask option

2-5-3-1. Setting the START CONDITION FLAG 1 (SCF1)

There are two ways for the IOC port to set the start condition flag 1 (SCF1). One is direct setting by external signal (also known as high voltage level detection) and the other is to pass the external signal through the chattering prevention circuit (also known as chattering prevention). The user can choose between the two methods using the mask option.

The diagram below shows the IOC port's associated control circuitry and flag:



Note: The default prevention clock is PH10

As the IOC port's four signal inputs must pass through the OR logic gate before it can be tested by the high voltage level detector or chattering prevention function, only when three IOC port input signals are simultaneously at a low voltage level and one input signal changes will there be a change in the output signal from the OR logic gate.

2-5-3-1-1. Direct setting by external signal

When mask option selects the use of the high voltage level detector, if the SCA instruction has already set the switch enable flag 4 (SEF4) to 1, a change from low to high voltage level in the IOC port input signal will set the start condition flag 1 (SCF1) to 1.

Once SCF1 is set to 1 the MCU will be unable to detect any further changes in the IOC port signal. All IOC port input pin signals must therefore be set to 0 and the SCA instruction executed again to clear SCF1. This allows the SCF1 to be set again the next time there's a change in the IOC port signal.

2-5-3-1-2. External signal through chattering prevention

The chattering prevention circuit has two properties: it can remove the noise from the input signal and detect changes in the input signal.

1. Removing noise from the input signal

If the noise on the input signal is smaller than a set interval, the chattering prevention circuit will remove it. (Except if the noise cycle is in synch with the clock source of chattering prevention function.)

There are three clock frequencies used for removing signal nose: PH10 (32ms), PH8 (8ms) or PH6 (2ms). These can be set using the SCC instruction and by default the MCU uses PH10.

2. Detect changes in input signal
After the chattering prevention circuit removes the noise, the rising or falling edge of this input signal will cause this circuit to set the start condition flag 1 (SCF1) to 1.

If mask option enabled the chattering prevention function, when the SCA instruction has set the switch enable flag 4 (SEF4) to 1, regardless of whether the IOC port input signal changes from a low to high voltage level or from a high to low voltage level the chattering function will always detect the change in signal state and set SCF1 to 1.

Once SCF1 is set to 1 the MCU will be unable to detect any further changes in the IOC port signal. The SCA instruction must therefore be executed again to clear SCF1. This allows the SCF1 to be set again the next time there's a change in the IOC port signal.

2-5-3-1-3. Notice for executing SCA instruction

MCU will clear all of SCF0, SCF1 and SCF3 flags in the same time while executing the SCA instruction. For this reason, the program has to check SCF0 and SCF3 flags before executing SCA instruction to avoid losing the signal changes on IOA and IOD port.

2-5-3-2. HALT RELEASE, STOP RELEASE and Interrupt Service

2-5-3-2-1. HALT release

Start condition flag 1 (SCF1) is the IOC port's halt release request signal. After SCF1 is set to 1 the halt release request flag 0 (HRF0) will be set to 1 as well, causing the MCU to generate a HALT release.

The conditions for the MCU to generate a HALT release will vary according to the method used by the IOC port to set SCF1.

1. When the IOC port chooses to use high voltage level detection to set start condition flag (SCF1), once SEF4 is set to 1 if any input signal on the IOC port's input pins changes to a high voltage level this will cause the MCU to generate a HALT release.
The MCU however can only enter the HALT mode when all input signals from the IOC port are at the low voltage level.

2. When the IOC port chooses to use chattering prevention for setting start condition flag 1 (SCF1), once SEF4 is set to 1 then any change in the "OR" result from the input signals on the IOC port input pins will cause the MCU to generate a HALT release.
When using the chattering prevention function the MCU can enter HALT mode regardless of what the IOC port's input signals are.

2-5-3-2-2. Interrupt Service

Start condition flag 1 (SCF1) is also the IOC port's interrupt request signal. If the IOC port's interrupt enable flag 0 (IEF0) is already set to 1, when the SCF1 is set to 1 it will also set the halt release request flag 0 (HRF0) to 1, causing the MCU to accept the interrupt request from the IOC port.

2-5-3-2-3. STOP release

The conditions for the MCU to generate a STOP release will vary according to the method used by the IOC port to set SCF1. The MCU can only enter the STOP mode when all input signals from the IOC port are at the low voltage level.

1. When the IOC port chooses to use high voltage level detection function to set start condition flag 1 (SCF1) and SEF4 is also set to 1, if any input signal on the IOC port's input pins changes to a high voltage level this will cause the MCU to generate a STOP release.
2. When the IOC port chooses to use chattering prevention function to set start condition flag 1, only when both SEF4 and SRF4 are set to 1 will any input signal on the IOC port's input pins' change to a high voltage level cause the MCU to generate a STOP release.

   After the MCU generates a STOP release it will first enter HALT mode. The high voltage level signal on the IOC port input pin must be maintained for a set time until the chattering prevention function validates this high voltage level signal and sets the start condition flag 1 to 1. Only then can it return to the low voltage level.

   If this high voltage level signal reverts back to a low voltage level before start condition flag 1 (SCF1) is set, the MCU will return immediately to STOP mode.

2-5-4. IOD PORT

IOD port's four pins can be set independently to input or output mode using the SPD instruction. However, when the MCU exits RESET state all IOD port pins will be automatically set to input mode.

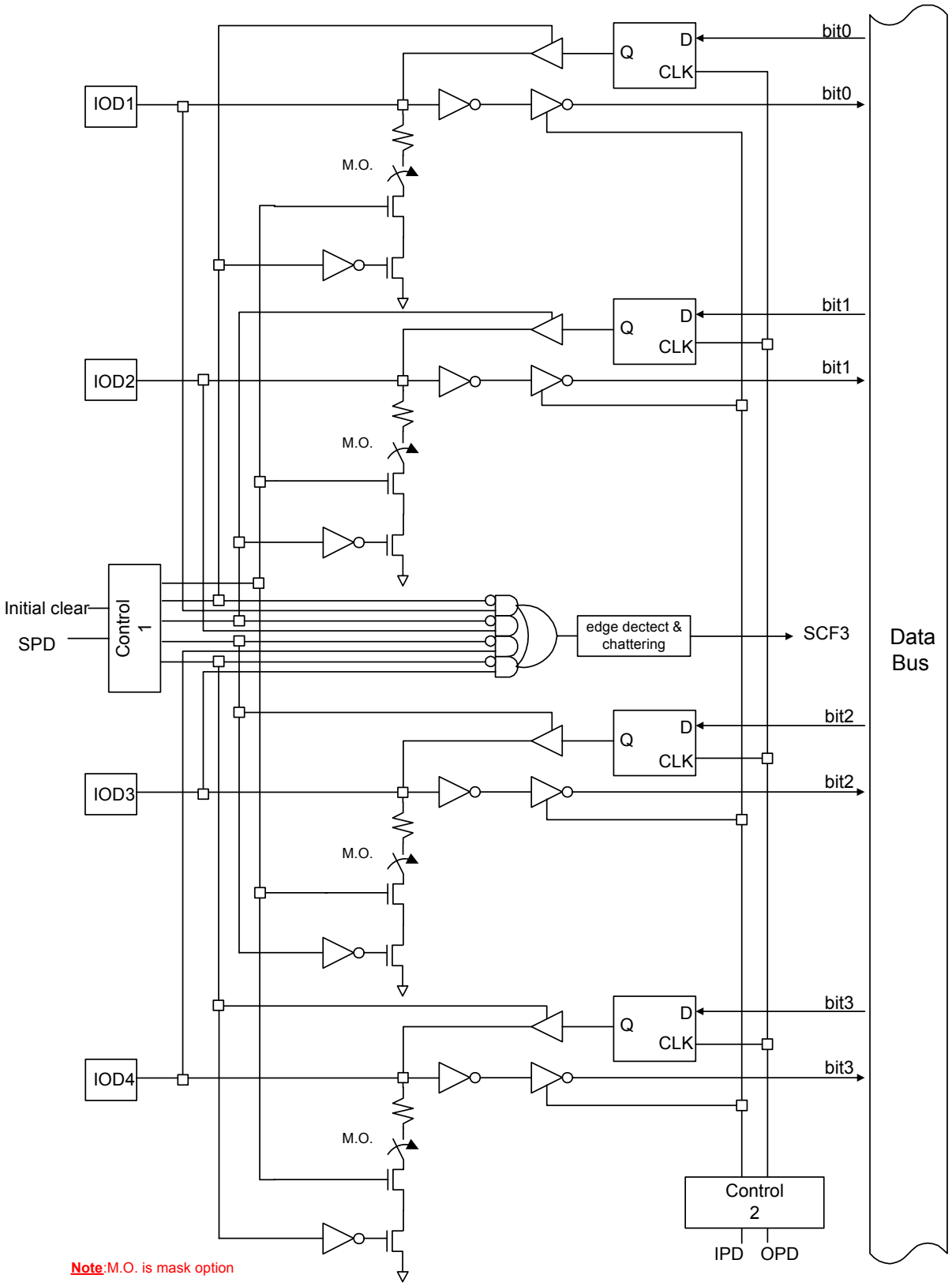
Executing the OPD instruction writes the contents of the data memory to the IOD port's output register. If the IOD port is already set to output mode then the contents of the output register will be output to the IOD pins.

Executing the IPD instruction stores the signal on the IOD port pins to data memory. If the IOD port is already set to output mode, executing the IPD instruction stores the contents of the output register to data memory.

Before the program can switch the IOD port to output mode it must first execute the OPD instruction to write the proposed output signal to the output register. This prevents the IOD port pins from outputting any unnecessary signals after being switched to output mode.

In input mode all IOD port pins have a built-in pull-down resistor to protect against input signal floating. This pull-down resistor can be enabled or disabled using mask option. In input mode, signal floating on the input pin will cause an unexpected current to flow through the input buffer.

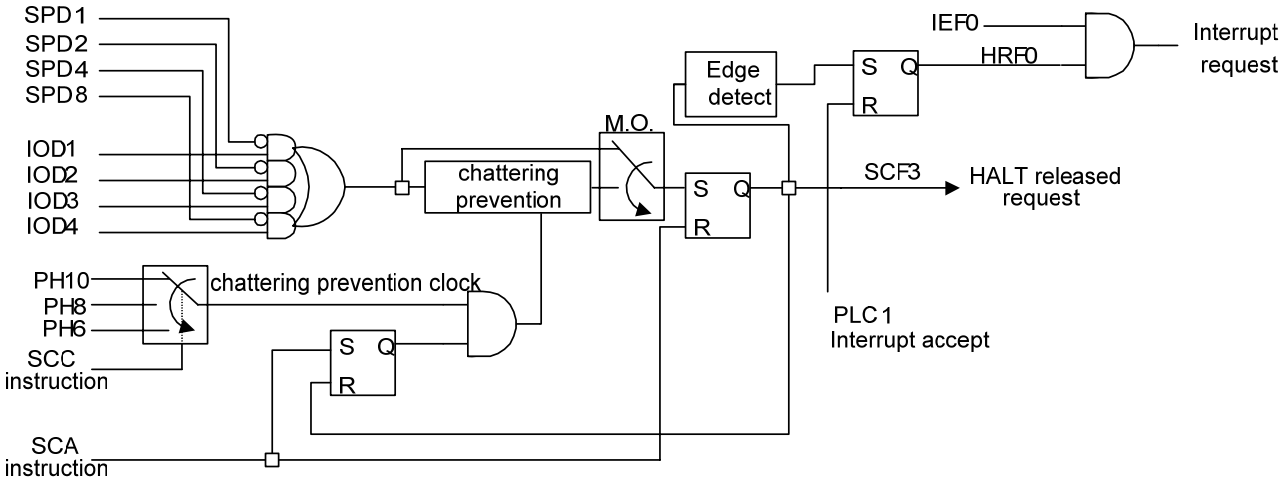
The diagram below shows the IOD port architecture:


bit0
D
Q
CLK
bit0
IOD1
M.O.
bit1
D
Q
CLK
bit1
IOD2
M.O.
Initial clear
SPD
Control 1
edge dectect & chattering
SCF3
Data Bus
bit2
D
Q
CLK
bit2
IOD3
M.O.
bit3
D
Q
CLK
bit3
IOD4
M.O.
Control 2
IPD
OPD
Note:M.O. is mask option

2-5-4-1. Setting the START CONDITION FLAG 3 (SCF3)

There are two ways for the IOD port to set the start condition flag 3 (SCF3). One is direct setting by external signal (also known as high voltage level detection) and the other is to pass the external signal through the chattering prevention circuit (also known as chattering prevention). The user can choose between the two methods using the mask option.

The diagram below shows the IOD port's associated control circuitry and flag:



Note: The default prevention clock is PH10

As the IOD port's four signal inputs must pass through the OR logic gate before it can be tested by the high voltage level detector or chattering prevention function, only when three IOD port input signals are simultaneously at a low voltage level and one input signal changes will there be a change in the output signal from the OR logic gate.

2-5-4-1-1. Direct setting by external signal

When mask option selects the use of the high voltage level detector, if the SCA instruction has already set the switch enable flag 3 (SEF3) to 1, a change from low to high voltage level in the IOD port input signal will set the start condition flag 3 (SCF3) to 1.

Once SCF3 is set to 1 the MCU will be unable to detect any further changes in the IOD port signal. All IOD port input pin signals must therefore be set to 0 and the SCA instruction executed again to clear SCF3. This allows the SCF3 to be set again the next time there's a change in the IOD port signal.

2-5-4-1-2. External signal through chattering prevention

The chattering prevention circuit has two properties: it can remove the noise from the input signal and detect changes in the input signal.

1. Removing noise from the input signal

If the noise on the input signal is smaller than a set interval, the chattering prevention circuit will remove it. (Except if the noise cycle is in synch with the clock source of chattering prevention function.)

There are three clock frequencies used for removing signal nose: PH10 (32ms), PH8 (8ms) or PH6 (2ms). These can be set using the SCC instruction and by default the MCU uses PH10.

2. Detect changes in input signal
After the chattering prevention circuit removes the noise, the rising or falling edge of this input signal will cause this circuit to set the start condition flag 3 (SCF3) to 1.

If mask option enabled the chattering prevention function, when the SCA instruction has set the switch enable flag 3 (SEF3) to 1, regardless of whether the IOD port input signal changes from a low to high voltage level or from a high to low voltage level the chattering function will always detect the change in signal state and set SCF3 to 1.

Once SCF3 is set to 1 the MCU will be unable to detect any further changes in the IOD port signal. The SCA instruction must therefore be executed again to clear SCF3. This allows the SCF3 to be set again the next time there's a change in the IOD port signal.

2-5-4-1-3. Notice for executing SCA instruction

MCU will clear all of SCF0, SCF1 and SCF3 flags in the same time while executing the SCA instruction. For this reason, the program has to check SCF0 and SCF1 flags before executing SCA instruction to avoid losing the signal changes on IOA and IOC port.

2-5-4-2. HALT RELEASE, STOP RELEASE and Interrupt Service

2-5-4-2-1. HALT release

Start condition flag 3 (SCF3) is the IOD port's halt release request signal. After SCF3 is set to 1 the halt release request flag 0 (HRF0) will be set to 1 as well, causing the MCU to generate a HALT release.

The conditions for the MCU's to generate a HALT release will vary according to the method used by the IOD port to set start condition flag 3 (SCF3).

1. When the IOD port chooses to use high voltage level detection to set start condition flag 3 (SCF3), once SEF3 is set to 1 if any input signal on the IOD port's input pins changes to a high voltage level this will cause the MCU to generate a HALT release.
The MCU however can only enter the HALT mode when all input signals from the IOD port are at the low voltage level.

2. When the IOD port chooses to use chattering prevention for setting start condition flag 3 (SCF3), once SEF3 is set to 1 then any change in the "OR" result from the input signals on the IOD port input pins will cause the MCU to generate a HALT release.
When using the chattering prevention function the MCU can enter HALT mode regardless of what the IOD port's input signals are.

2-5-4-2-2. Interrupt Service

Start condition flag 3 (SCF3) is also the IOD port's interrupt request signal. If the IOD port's interrupt enable flag 0 (IEF0) is already set to 1, when the SCF3 is set to 1 it will also set the halt release request flag 0 (HRF0) to 1, causing the MCU to accept the interrupt request from the IOD port.

2-5-4-2-3. STOP release

The conditions for the MCU to generate a STOP release will vary according to the method used by the IOD port to set SCF3. The MCU can only enter the STOP mode when all input signals from the IOD port are at the low voltage level.

1. When the IOD port chooses to use high voltage level detection function to set start condition flag 3 (SCF3) and SEF3 is also set to 1, if any input signal on the IOD port's input pins changes to a high voltage level this will cause the MCU to generate a STOP release.
2. When the IOD port chooses to use chattering prevention function to set start condition flag 3, only when both SEF3 and SRF3 are set to 1 will any input signal on the IOD port's input pins' change to a high voltage level cause the MCU to generate a STOP release.

   After the MCU generates a STOP release it will first enter HALT mode. The high voltage level signal on the IOD port input pin must be maintained for a set time until the chattering prevention function validates this high voltage level signal and sets the start condition flag 3 to 1. Only then can it return to the low voltage level.

   If this high voltage level signal reverts back to a low voltage level before start condition flag 3 is set, the MCU will return immediately to STOP mode.

2-5-5. IOE PORT

IOE port's four pins can be set independently to input or output mode using the SPE instruction. However, when the MCU exits RESET state all IOE port pins will be automatically set to input mode.

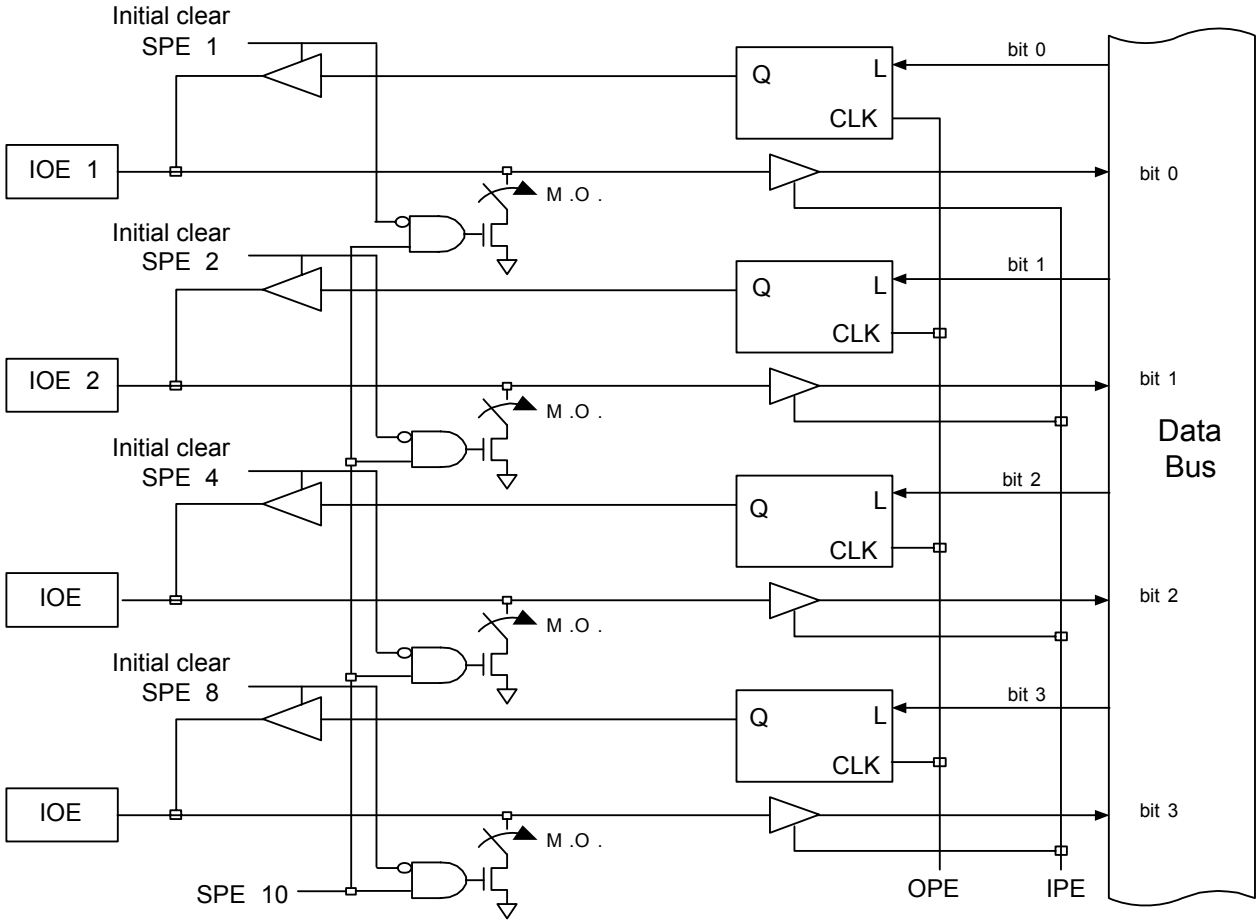
Executing the OPE instruction writes the contents of the data memory to the IOE port's output register. If the IOE port is already set to output mode then the contents of the output register will be output to the IOE pins.

Executing the IPE instruction stores the signal on the IOE port pins to data memory. If the IOE port is already set to output mode, executing the IPE instruction stores the contents of the output register to data memory.

Before the program can switch the IOE port to output mode it must first execute the OPE instruction to write the proposed output signal to the output register. This prevents the IOE port pins from outputting any unnecessary signals after being switched to output mode.

In input mode, all IOE port pins have a built-in pull-down resistor to protect against input signal floating. This pull-down resistor can be enabled or disabled using mask option. In input mode, signal floating on the input pin will cause an unexpected current to flow through the input buffer.

The diagram below shows the IOE port architecture:

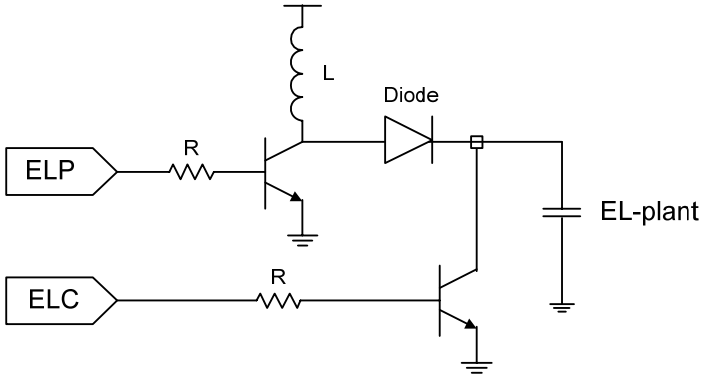


Note: M.O. is mask option

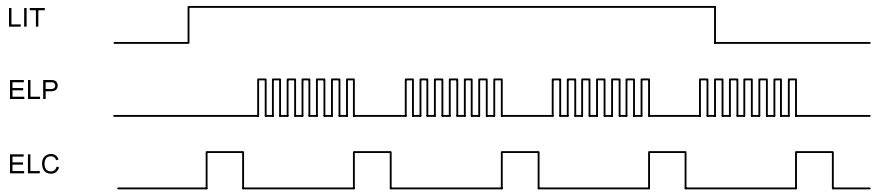
2-6. EL-PLANT DRIVER

The TM89 Series MCU provides an EL-plant driver (ELP and ELC output pins) that only needs a few external parts to be added to directly drive the EL plant and generate the light source needed by the system. The ELP pin can output the control signal needed for the charge-pump circuit while the ELC pin can output the control signal needed for the discharge circuit.

The diagram below is the application circuit for the EL-plant driver:



The diagram below shows control signal from the EL-plant driver (LIT: internal control signal)



Executing the SF instruction enables the EL-plant driver function (LIT signal is high voltage level). Executing the RF signal disables the EL-plant driver function (LIT signal is low voltage level). Before enabling the EL-plant driver function the ELC instruction must be executed first to set the ELP and ELC pins' output signal frequency and duty cycle.

When the EL-plant driver function is enabled, before the ELP pin begins outputting the charge-pump signal the ELC pin will send a pulse signal to clear the residual charge on the EL-plant component. This avoids any damage to the EL-plant component from the subsequent charge-pumping procedure.

Conversely, after the EL-plant driver is disabled, after the ELP pin sends its last set of charage-pump signal the ELC pin will send a pulse signal afterwards to clear the residual charge on the El-plant driver component.

The table below shows how the ELP output signal frequency and duty cycle can be set using the ELC instruction:

(X8,X7,X6)	Pumping clock frequency	(X9,X5,X4)	Duty cycle
0	PH0	000	1/4 duty
11	FREQB	001	1/3 duty
100	BCLK	X10	1/2 duty
101	BCLK/2	X11	1/1 duty
110	BCLK/4	100	3/4 duty
111	BCLK/8	101	2/3 duty

The table below shows how the ELC output signal frequency and duty cycle can be set using the ELC instruction:

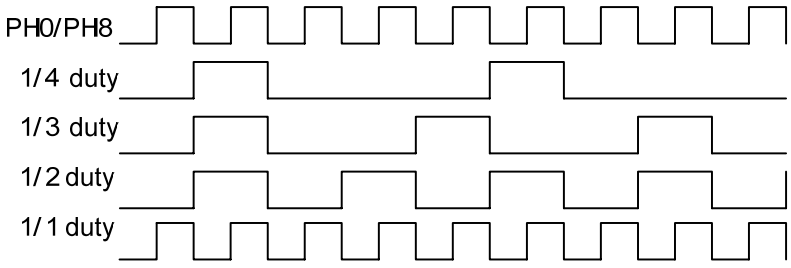
(X3,X2)	Discharge pulse frequency	(X1,X0)	Duty cycle
00	PH8	00	1/4 duty
01	PH7	01	1/3 duty
10	PH6	10	1/2 duty
11	PH5	11	1/1 duty

The initial settings for the ELP and ELC output signals after MCU enters RESET state are as follows:

ELP pin output signal: Frequency is PH0, 1/4 duty cycle

ELC pin output signal: Frequency is PH8, 1/4 duty cycle

The following diagram shows the relationship between the output signal frequency and duty cycle:



The example below shows how to set and enable the EL-pant driver

```
ELC     $110        ; set ELP output signal frequency to BCLK and 1/3 duty cycle,
                    ; set ELC output signal frequency to PH8 and 1/4 duty cycle,
SF      $C          : start EL-plant driver then cause MCU to enter HALT mode.
……..
RF      $4          ; disable EL-plant driver
```

2-7. EXTERNAL INT PIN

INT is the input pin used by the MCU to receive external interrupt signal. The logic level on the pin can be stored to the data memory by executing the MDX instruction.

The INT input pin has three selectable properties: built-in pull-low resistor, built-in pull-high resistor and high impedance input. The three input properties can be selected using the mask option.

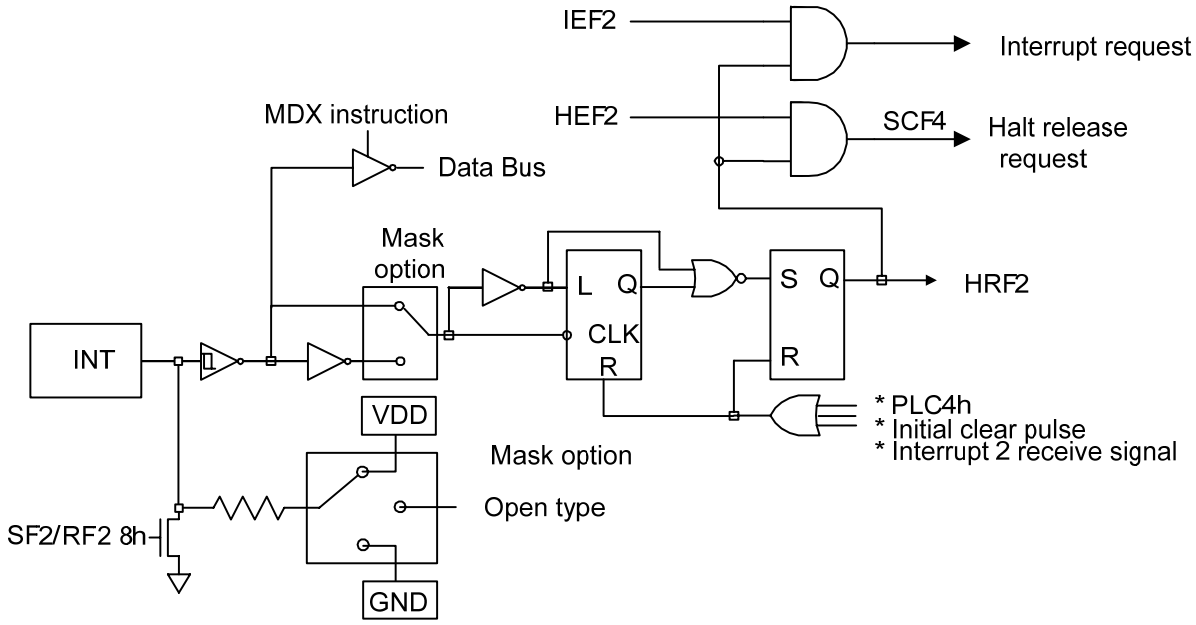
Additionally, the INT pin also has a low-resistance pull-low component that can be enabled or disabled using the instructions SF2 and RF2. The resistance of this component is far lower than the resistance of the pull-high or pull-low resistors.

The input signal on the INT pin will only generate an interrupt signal when it changes state. Mask option can be used by the user to select whether an interrupt request is delivered to the MCU on the rising or falling edge of the input signal.

When the INT pin detects a change in the input signal, it sets the halt release request flag 2 (HRF2) to 1. In this situation, if the halt release enable flag (HEF2) is already set to 1, the MCU will generate a HALT release or STOP release and set the start condition flag 2 (SCF2) to 1.

If the interrupt enable flag 2 (IEF2) is already set to 1, once the halt release request flag 2 (HRF2) is set to 1 the MCU will accept the interrupt request.

The INT pin circuit diagram is shown below:

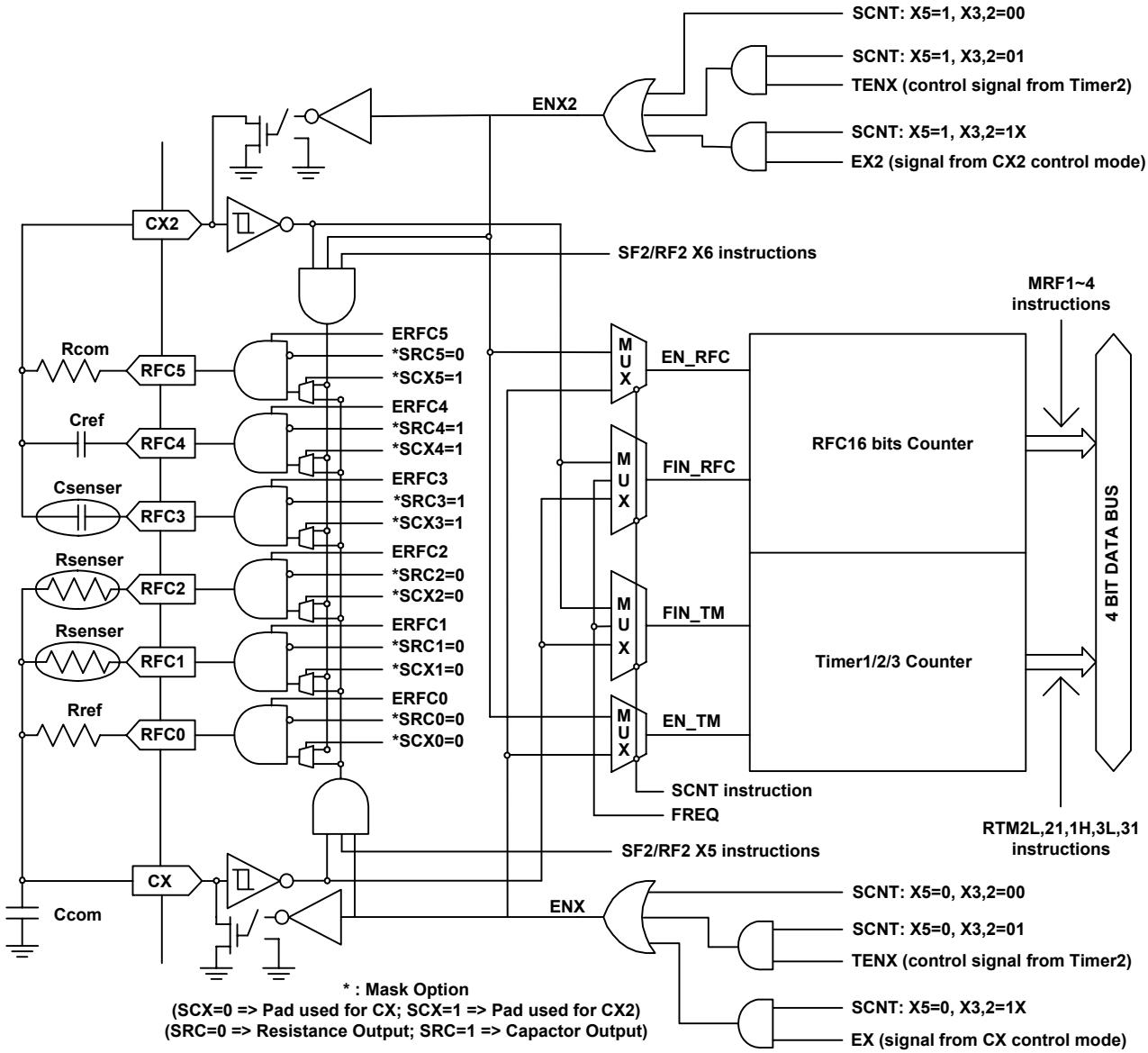


2-8. RESISTANCE TO FREQUENCY CONVERTER (RFC)

The Resistance to Frequency Converter (RFC) is made up of the RC oscillator and the RFC counter. The resistor and capacitor in the RC oscillator come from external parts in the application circuit. The RC oscillator can convert the resistance or capacitance of an external component into its corresponding frequency. The RFC counter then converts this frequency into a numerical value for the program operations to use.

The entire RFC function can only be enabled after executing the SF2 instruction. The RF2 instruction must then be executed first before disabling the RFC function.

The TM89 Series MCU offers two sets of RFC functions. The diagram below shows the RFC's basic architecture and application circuit:



2-8-1. RC Oscillator Setup and Startup

The TM89 Series MCU offers two different RFC architectures. One architecture uses a resistance sensor (e.g. RFC0~RFC2 in the figure of 2-8 “RFC Basic Architecture and Application Circuit”) while the other architecture is used as a capacitance sensor (e.g. RFC3~RC5 in the figure of 2-8 “RFC Basic Architecture and Application Circuit”).

In the RFC used for the resistance sensor, one input pin (CX or CX2) and one output pin set to connector to a resistor (RFC0~RFC5) can be used in conjunction with an external resistor and capacitor to form a RC oscillator. In this framework, after enable any output pin of RFC0~5, the capacitor can be charged/discharged through the resistor connected to the output pin. This charge/discharge signal can be outputted to the output pin of RFC0~5 through the input pin of CX/CX2 and form a RC oscillator.

In the RFC used for the capacitance sensor, one input pin (CX or CX2) and a RFCn (RFC0~RFC5) set to connect to a resistor and a RFCm (RFC0~RFC5) set to connect to a capacitor must be used in conjunction with an external resistor and capacitor to form a RC oscillator. In this framework, one of the output pin of RFCm with a connected capacitor must be set up as GND and then enable the output pin of RFCn. By doing so, the output signal of RFCn can charge or discharge the capacitor which is connected to RFCm pin through the resistor connected to the RFCn pin. This charge or discharge signal can be outputted to the output pin of RFCn through the input pin of CX and form a RC oscillator.

Before using the RC oscillator of RFC, mask option must be used to set how RFC0~RFC5 will function on the MCU hardware. An explanation is provided below:

1. Use mask option to set each RFC0~RFC5 output pin to correspond to the CX or CX2 pin to form a RC oscillators.

2. Use mask option to set each RFC0~RFC5 output pin to connect to a resistor or capacitor in the application circuit.

If the RFCn output pin is set to connect to a resistor, RFCn pin could output both the 0V and VBAT voltage level.

If the RFC output pin is set to connect to a capacitor, RFCn pin will only output a voltage of 0V.

3. Use mask option to set whether the CX or CX2 input pin should be the high impedance input mode or low impedance input mode when oscillation is stopped (enable a pull low component on the CX or CX2 input pin).

Choosing the low impedance input mode allows the voltage level on the CX or CX2 pin to be quickly discharged to 0V when the RC oscillator stops. This prevents the RFC’s counting result from being affected by the discharge time of capacitor.

If the CX or CX2 input signal is an external control signal in the actual application, do not use the low-impedance input mode or this will result in conflicting input signals.

Executing the SRF instruction sets the RFC0 ~ RFC5 pins specified in the operand as output mode or high-impedance mode (tri-state). When the RFC0~RFC5 pin is set to output mode the RC oscillator will not start immediately until the SF2 instruction is executed to enable the corresponding CX or CX2 input function, and then the RC oscillator will start up.

When the program executes the RF2 instruction to disable the corresponding CX or CX2 input function or set the corresponding RFC0~RFC5 pin to high-impedance mode, the RC oscillator will stop.

We recommend setting the program to execute SRF then SF2 when enabling the RC oscillator for a more accurate counter value.

Only one RC oscillator can be enabled at one of the CX or CX2 input pin at a time or this will lead to errors in the oscillation frequency. Additionally, while the two input pins (CX and CX2) can correspond to two different RC oscillators, we do not recommend enabling both at the same time as the power noise will affect the RC oscillators' output frequencies.

The clock generated by the active RC oscillator will be transmitted through the CX or CX2 pin to the RFC counter for counting.

2-8-2. RFC Counter's Counting Function and Control Method

In the RFC function, the RFC counter uses the CX or CX2 pin to receive the clock signals with different frequencies generated by the RC oscillator then counts the number of clocks received within a set amount of time.

The RFC counter has four different types and control methods. These can all be set by executing the SCNT instruction.

The four RFC counter types are: 16-bit RFC counter, TMR1, TMR2 and TMR3.

The four RFC counter control methods are: controlled by program instruction, controlled by TMR2, controlled by one-cycle signal on CX or CX2 input pin and controlled by the high voltage level signal on CX or CX2 input pin.

Though the SCNT instruction can set a RFC counter to be controlled simultaneously by CX and CX2 pin, or setting the RFC counter's clock source to simultaneously receive the clock signal from CX and CX2 pin, when both sets of control signals or clock sources are active the RFC counter's operation and received signal sources will become very complicated. We therefore recommend against activating both control modes when using this setup method.

When the program finishes setting up the RC oscillator, the RFC counter must be set up first before executing the SF2 instruction to activate the RFC function. When the RFC counter finishes counting, the program can use the MRF1~4 instruction to read the counted value of the 16-bit RFC counter or use instructions associated with reading the timer value to read the counted value from TMR1~3.

Regardless of the method used to control the RFC counter, when the program executes the instruction SF2 to activate the RFC function this automatically resets the RFC counter 0.

2-8-2-1. 16-BIT RFC COUNTER

This is a counter designed specifically for the use of the RFC counter. If the 16-bit RFC counter experiences an overflow while counting, the 16-bit RFC counter overflow flag (RFOVF) will be set to 1. Executing the MSD instruction will store the RFOVF flag to the data memory. When the program executes the instruction SF2 this will clear the RFOVF flag.

Mask option can be used to set whether the 16-bit RFC counter should restart its count or stop immediately after an overflow. If set to stop immediately, the counter value will remain at 0000h. If set to automatic restart, RFOVF will remain as 1 and only cleared at the next overflow. RFOVF can therefore be considered the 17th stage output signal of the 16-bit RFC counter.

The program can use the instructions MRF1~4 to read the counted value of the 16-bit RFC counter and store it to the data memory.

2-8-2-2. TIMER as RFC COUNTER

Executing the SCNT instruction can set TMR1, TMR2 or TMR3 as the counter for the RFC function. The CX or CX2 pin can also be set to become the signal input pin for the RFC counter. See the instruction manual for details on setting SCNT.

All kinds of timer's configuration such as 6-bit, 12-bit or 18-bit can be used as the RFC counter. The program can also have two RFC functions active at the same time, with one using the standard 16-bit RFC counter while the other uses a timer as the RFC counter.

When a timer is used as the RFC counter, the timer's clock source and preset data is automatically set by the RFC hardware circuit, it can only count down from the max. data (3Fh, FFFh, 3FFFFh) to 0 as well.

Only the method for setting the timer's clock source and initial value is different from standard timer operation, everything else works the same way. For example, the timer will stop on underflow and set the timer halt release request flag (HRFn), the re-load function can be set and the contents of the timer can be read etc.

Please note that when the timer is used for the RFC counter do not try to set or control the timer with the standard timer instructions as this will impact on the RFC counter's operation. Do not execute standard timer instructions before the RFC counter's contents are read either, or the counter value will be changed.

2-8-2-3. Controlling the RFC Counter with Program Instructions

With this control mode, the RFC counter's clock source must come from the signal on the CX or CX2 pin then the instructions SF2 or RF2 used to enable and disable the counter.

Upon executing SF2 20h (X5=1) or SF2 40h (X6=1), the RFC counter will be reset to zero then begin counting the signal from CX or CX2 no matter what counter type it is. Upon executing RF2 20h (X5=1) or RF2 40h (X6=1) the RFC counter will stop counting.

Example:

This example will count the number of clocks from the CX pin during a set interval. The SF2 instruction is used to enable the counting function with the counting interval defined using TMR1.

When the counting interval finishes the program will check for an overflow at the 16-bit RFC counter. If no overflow occurred it reads the content of the 16-bit RFC counter. If an overflow occurred it will shorten the counting interval. This example will use the RFC0 and CX pins to form a RC oscillator.

```
                        ; Using TMR1 to set counting interval
LDS     0, 0            ; TMR1's clock source is PH9
LDS     1, 3            ; TMR1's preset data is 3F
LDS     2, $0F
```

```
        SHE     2               ; set TMR1 underflow to generate halt release
        SCNT    $00             ; set CX pin, program instruction controlled method, 16-bit
                                ; RFC counter
        SRF     $01             ; set RFC0 to output mode
RE_CNT:
        LDA     0
        OR*     1               ; generate TMR1' preset data
        TMS     2               ; enable TMR1
        SF2     $20             ; enable CX pin's RFC function
        HALT
        RF2     $20             ; at TMR1 underflow stop the CX pin's RFC function
        MRF1    $10             ; read 16-bit counter
        MRF2    $11
        MRF3    $12
        MRF4    $13
        MSD     $20
        JB2     CNT1_OF         ; check if RFC counter overflow or not
        JMP     DATA_ACCEPT

CNT1_OF:
        DEC*    2               ; shorten the interval set for TMR1
        LDS     $20, 0          ; set AC=0
        SBC*    1
        JZ      CHG_CLK_RANGE   ; change TMR1's clock source
        PLC     1               ; clear TMR1's halt release request flag
        JMP     RE_CNT
```

2-8-2-4. Controlling RFC COUNTER with TMR2

The RFC counter types that TMR2 can control are: 16-bit RFC counter, TMR1 or TMR3. It can't control TMR2 itself.

TMR2 (6-bit, 12-bit or 18-bit) is the only timer that can be used to enable or disable the RFC counter. The program can use the SCNT instruction to set this control function.

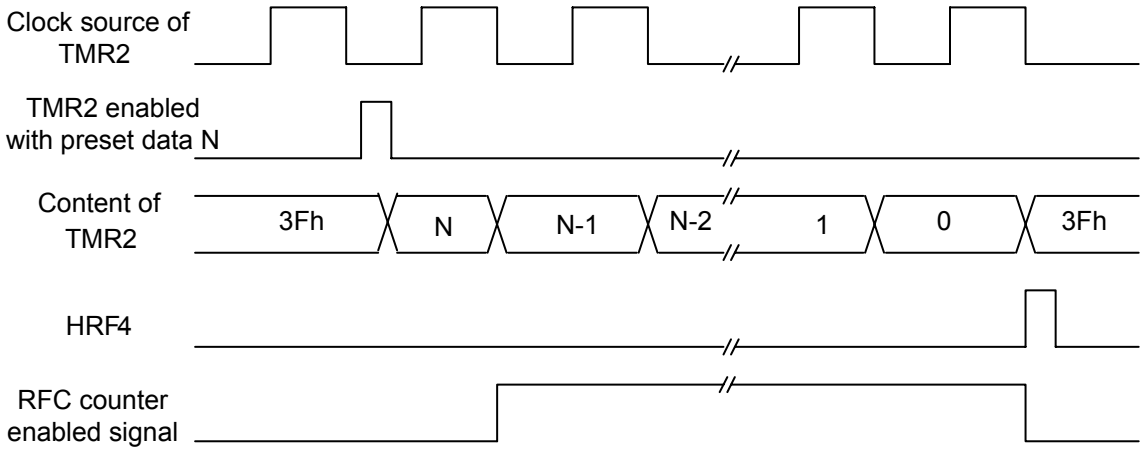
When this control mode is selected for the RFC function, the RFC counter will not start counting immediately when the instruction SF2 is executed to enable the RFC function. Only when TMR2 is enabled and the first falling edge signal appears in the TMR2 clock source will the RFC counter begin counting. The RFC counter will only stop counting when there is a TMR2 underflow. In this way TMR2 offers a very precise counter timing for control of the RFC counter. This avoids the timer to generate a counting error which smaller than a clock source signal cycle.

In this control mode, if the clock source of RFC counter is set to CX or CX2 pin, the RC oscillator formed by the CX or CX2 pin will work in the same manner as the RFC counter. The RC oscillator is controlled by TMR2 for enabling/disabling after the program executes SF2 instruction to enable the RFC function.

If the RFC counter's clock source is set to the frequency generator output (FREQ) then the RC oscillator formed by the CX or CX2 pin will be enabled or disabled directly by the SF2

or RF2 instruction. This means the RC oscillator will start-up without waiting for the first falling edge signal in the TMR2 clock source.

The timing diagram for the TMR2's control of the RFC counter is shown below:



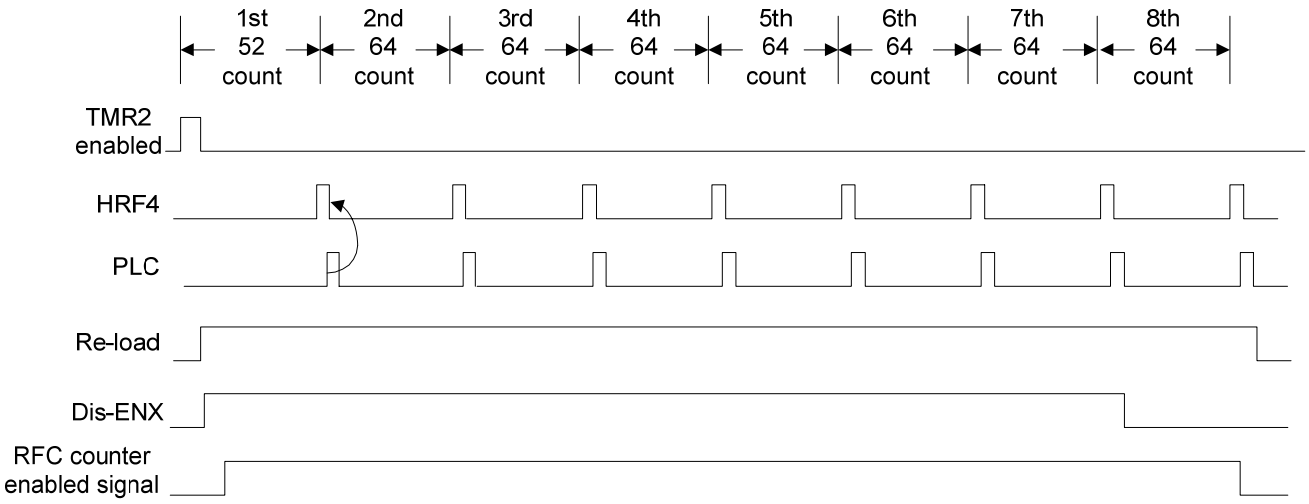
TMR2 can also use the re-load function to control the RFC counter. Since the RFC counter control signal (the "RFC counter enable signal" in the above diagram) is cleared when underflow occurs at TMR2. For this reason, another Dis-ENX flag must be set to ensure that the RFC counter enable signal won't be cleared by the HRF4 flag and the RFC counter can continue to countdown again.

Once the program enables the TMR2's re-load function it must immediately execute the SF2 2h instruction to set the Dis-ENX flag to 1. In this way, the halt release request flag (HRF4) generated by TMR2 underflow won't affect the operation of the RFC counter.

When disable the TMR2 re-load function, the program must execute the RF2 2h instruction before the last underflow at TMR2 to clear the Dis-ENX flag. The "RFC counter enabled signal" will then be cleared when the TMR2 generates its last halt release request flag 4 (HRF4) and the RFC counter will stop at the time.

The example below explains how to use the TMR2 re-load function to control the 16-bit RFC counter:

If the program wishes for the TMR2 to count down from 500, it can translate this into one countdown from 52 and 7 countdowns from 64 (64*7 + 52). In this example, the program enters HALT mode to wait for the underflow generated by TMR2.

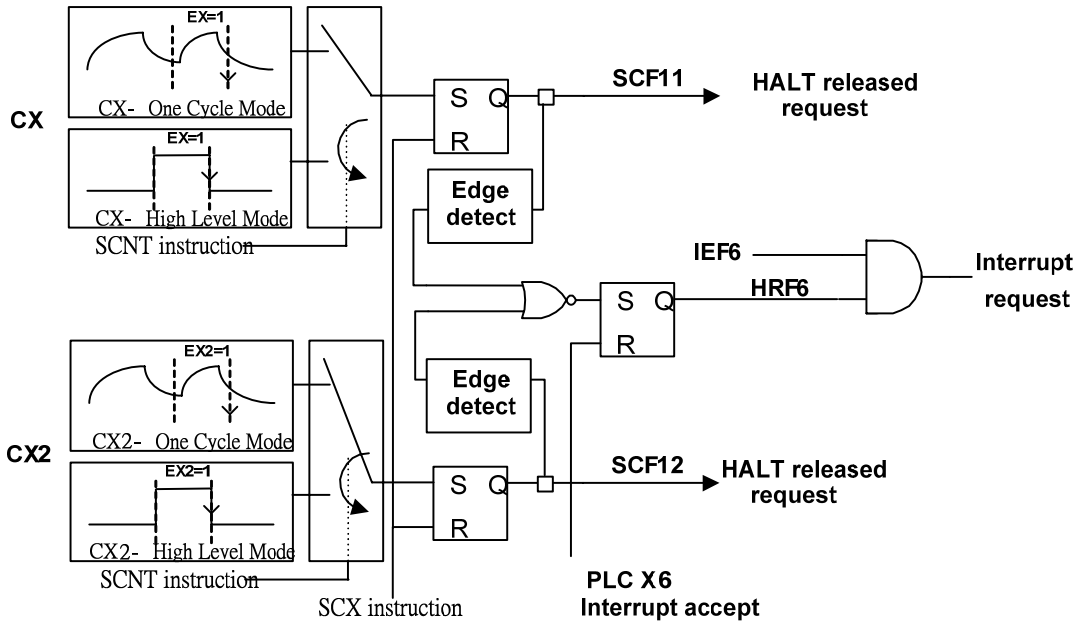



```
        LDS     0,0             ; use the value of data memory address 0 to count TMR2's
                                ; underflow times and initiate it to 0.
        PLC     $10             ; clear HRF4
        SHE     $10             ; allow the MCU to generate HALT release on TMR2 underflow
        SCNT    $04             ; let TMR2 control 16-bit RFC counter,
                                ; and set RFC counter clock source to CX.
        SF2     $20             ; activate the RFC function on CX pin
        TM2X    $34             ; set TMR2's preset data (52), choose PH9 as clock source,
                                ; start TMR2
        SF2     3               ; start TMR2' re-load function and set Dis-ENX flag to 1

RE_LOAD:
        HALT
        INC*    0               ; increment TMR2's underflow count by 1
        PLC     $10             ; clear HRF4
        LDS     $20, 7
        SUB     0               ; when the 7th TMR2 underflow occurs, clear Dis-ENX flag
                                ;
        JNZ     NOT_RESET_DED
        RF2     2               ; clear Dis-ENX flag

NOT_RESET_DED:
        LDA     0               ; write TMR2 underflow times to AC
        JB3     END_TM1         ; check if TMR2 underflow times equal 8
        JMP     RE_LOAD

END_TM1:
        RF2     1               ; Disable TMR2's re-load function
```

2-8-2-5. Using One Cycle Signal on CX or CX2 Input Pin to control RFC COUNTER

In this control method the external signal received at CX or CX2 is used to enable or disable the RFC counter. The counter can only count the frequency of the signal from the frequency generator output (FREQ).

The diagram below shows the architecture and flag using the one cycle signal or high voltage level signal on the CX / CX2 input pin to control the RFC counter:

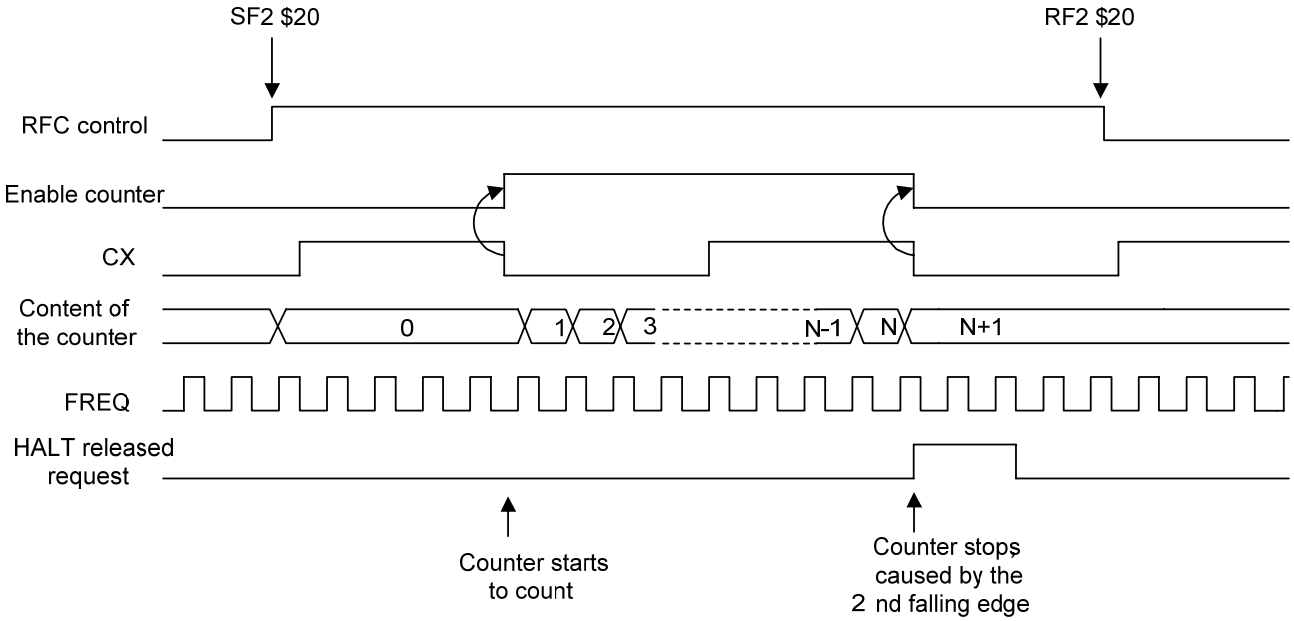


After executing the SF2 instruction to enable the RFC function, the RFC counter does not start counting immediately. It will only start counting when the first falling edge signal arrives at the CX or CX2 pin. When the second falling edge signal arrives at the CX or CX2 pin the RFC counter will stop counting. The program has to execute the RF2 instruction to disable the entire RFC function.

When the program executes the SCX instruction to set switch enable flag 0, 1 (SEF0, 1) to 1, when the second falling edge signal arrives at the CX or CX2 pin this sets the start condition flag 11, 12 (SCF11, 12) to 1. At the same time, the halt release request flag 6 (HRF6) to 1. Executing the SCX instruction will clear SCF11 and SCF12 as well.

In the above situation, if the interrupt enable flag 6 (IEF6) is already set to 1, once the halt release request flag 6 (HRF6) is set to 1 the MCU will accept the interrupt request.

The diagram below shows the timing for using the CX/CX2 falling edge signal to control the RFC counter:



Example:

```
SCC         $0          ; set frequency generator's clock source to PH0
FRQX        1, 5        ; set frequency generator's output (FREQ) = (PH0/6)/3
STM         3           ; set TMR2 to 18-bit
SHE         $10         ; set TMR2 underflow to generate halt release
SCX         1           ; clear SCF11, 12 and set SEF0 flag
SCNT        $0a         ; using one cycle signal on CX pin to control RFC counter,
                        ; TMR2=RFC counter,
                        ; clock source=FREQ
SRF         $04         ; set RFC2 to output mode
SF2         $20         ; start CX's RFC function
HALT
MCX         $6f         ; read SCF6
JB1         TM2_UF      ; check if TMR2 underflow
MAF         $7f         ; read SCF11
ANDI        $7f,1       ;
JZ          NOT_CX      ; check if CX control signal ended
SCX         1           ; clear SCF11, 12 and set SEF0 flag
PLC         50h         ; clear HRF6
RTM2L $10               ; read TMR2 content
RTM21 $11
```

```
        RTM1H $12
        RTM3L $13
        RTM31 $14
…….
TM2_UF:                         ; Timer 2 underflow occurred
```

2-8-2-6. Using High Voltage Level Signal on CX or CX2 Input Pin to control RFC COUNTER

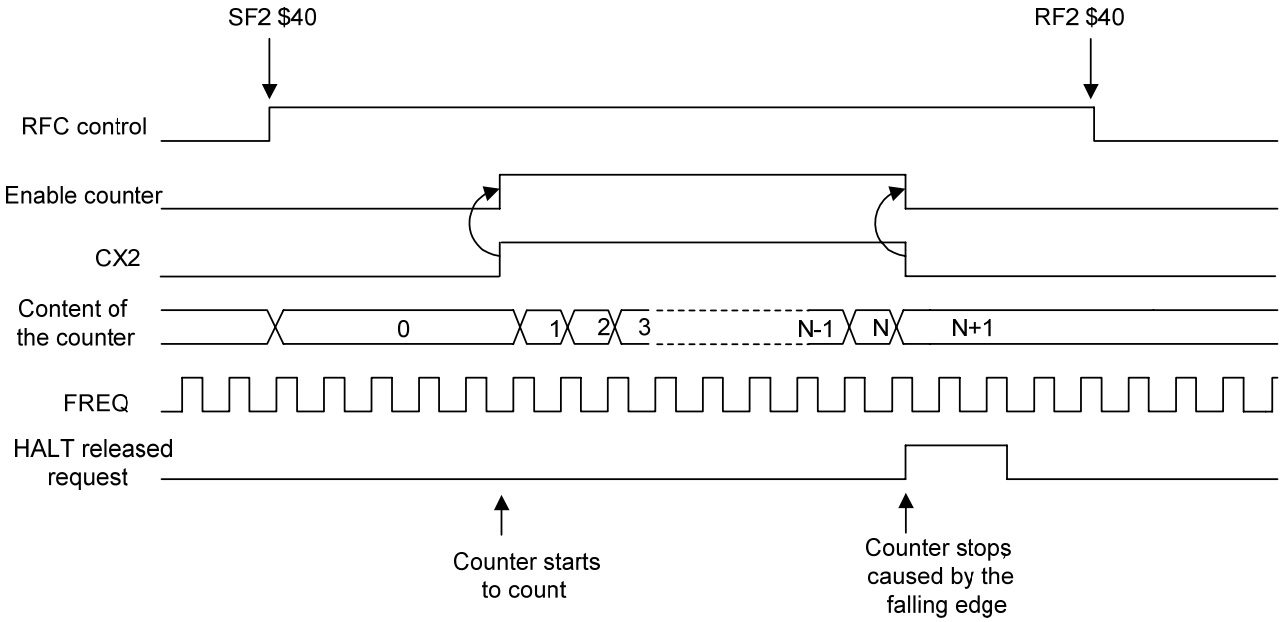
In this control method the external signal received at CX or CX2 is used to enable (high voltage level) or disable (low voltage level) the RFC counter. The counter can only count the frequency of the signal from the frequency generator (FREQ). Please refer to the diagram in 2-8-2-4 for an explanation.

After executing the SF2 instruction to start the RFC function, the RFC counter does not start counting immediately. It will only start counting when the high level signal arrives at the CX or CX2 pin. When the signal on CX or CX2 pin becomes low voltage level the RFC counter will stop counting.

When the program executes the SCX instruction to set switch enable flag 0, 1 (SEF0, 1) to 1, when the signal at the CX or CX2 pin changes from high voltage level to low voltage level this sets the start condition flag 11, 12 (SCF11, 12) to 1. At the same time, the halt release request flag 6 (HRF6) to 1. Executing the SCX instruction will clear SCF11 and SCF12 as well.

In the above situation, if the interrupt enable flag 6 (IEF6) is already set to 1, once the halt release request flag 6 (HRF6) is set to 1 the MCU will accept the interrupt request.

The diagram below shows the timing for using the CX/CX2 high voltage level signal to control the RFC counter:



Example:

```
SCC     $0      ; set frequency generator's clock source to PH0
FRQX    3, 0    ; set frequency generator's output frequency (FREQ) = PH0
STM     2       ; set TMR3 to 12-bit
SHE     $80     ; set TMR3 underflow to generate halt release
SCX     2       ; clear SCF11, 12 and set SEF1 flag
SCNT    $2f     ; using the high voltage level signal on CX2 to control RFC
                ; counter, TMR3=RFC counter, clock source=FREQ.
SRF     $08     ; set RFC3 to output mode
```

```
        SF2         $40         ; start CX2's RFC function
        HALT
        MDX         $6f         ; read SCF10
        JB3         TM3_UF      ; check if TMR3 underflow
        MAF         $7f         ; read SCF12
        ANDI        $7f,2       ;
        JZ          NOT_CX      ; check if CX2 control signal ended
        SCX         2           ; clear SCF11, 12 and set SEF1 flag
        PLC         50h         ; clear HRF6
        RTM3L       $10         ; read TMR3 content
        RTM31       $11
        RTM1H       $12

……

TM3_UF:                         ; Timer 3 underflow occurred
```

2-8-2-7. Using Multi-Cycle Signal on CX or CX2 Input Pin to control the RFC COUNTER

The SCNT instruction can set the RFC counter to be controlled by TMR2 and the RFC counter's clock source as the frequency generator output (FREQ). If TMR2's clock source is then set to the CX or CX2 pin, the CX or CX2 pin can be used to control the RFC counter indirectly through TMR2. This allows the CX/CX2 one-cycle control mode to be turned into a multi-cycle control mode.

Example: (Calculating the CX input signal frequency: Use PH0 as the time base to calculate the time interval that 100 input clocks on CX pin takes)

```
        SCC         $0          ; set frequency generator's clock source to PH0
        FRQX        3, 0        ; set frequency generator's output frequency (FREQ) = PH0
        SCNT        $14         ; sets RFC function with CX pin:  Signal source = FREQ,
                                ; control method = TMR2, counter type= 16-bit RFC counter.
        SRF         $20         ; set RFC5 to output mode
        STM         1           ; TMR2= 12 bit timer
        SF2         $20         ; enable CX's RFC function (start oscillation on CX)
        T2XH
        setdat      $8064       ; TMR2 clock source = CX, preset data = 100, enable TMR2
        SHE         $10         ; allow TMR2 to generate HALT release
        HALT
        RF2         $20         ; disable CX's RFC function (this stops oscillation on CX)
        MSD         $6F
        JB2         CNT1_OF     ; check 16-bit RFC counter for overflow
        PLC         $10         ; clear TMR2's halt release request flag
        MRF1        $10         ; read the content of 16-bit RFC counter
        MRF2        $11
        MRF3        $12
        MRF4        $13

……

CNT1_OF:                        ; 16-bit RFC counter overflow occurred
```

2-8-2-8. Examples for RFC

Example 1: (TMR2 controls two RFC counters simultaneously)

Note: When the program activates two RC oscillator units in RFC function at the same time this generates more power noise and impacts on the RC oscillator's output frequency. We therefore recommend against activating two RC oscillator units at the same time.

```
SCNT    $04     ; sets CX's RFC unit: clock source=CX, control method=TM2,
                ; counter=16-bit RFC counter.
SCNT    $24     ; sets CX2's RFC unit: clock source=CX2, control method =TM2,
                ; counter=TMR3.
STM     2       ; TMR3=12 bit timer
SRF     $11     ; set RFC0 and RFC4 to output mode
SF2     $60     ; activate the two RFC units CX and CX2 simultaneously.
SHE     $90     ; allow TMR2 and TMR3 to generate HALT release
TM2X    $1df    ; set TMR2's preset data=1Fh, clock source = PH13,
                ; enable TMR2.
HALT
MDX     $6f     ; read SCF10 (TM3 underflow flag).
JB3     TM3_UF  ; check if TMR3 underflow
MSD     $6f
JB2     CNT1_OF ; check 16-bit RFC counter for overflow
PLC     $10     ; clear TMR2's halt release request flag
MRF1    $10     ; read the content of 16-bit RFC counter
MRF2    $11
MRF3    $12
MRF4    $13
RTM3L   $20     ; read value of 12-bit TMR3
RTM31   $21
RTM1H   $22

.....

CNT1_OF:        ; 16-bit RFC counter overflow occurred

.....

TM3_UF:         ; Timer 3 underflow occurred
```

Example 2: (CX and CX2's input signals to control two RFC counters)

```
SCC     $0      ; set frequency generator's clock source to PH0
FRQX    3, 0    ; set frequency generator's output frequency(FREQ)=PH0
SCNT    $1d     ; sets CX's RFC unit: Signal source=FREQ,
                ; control method=High-Level on CX, counter=TMR1.
SCNT    $28     ; sets CX2's RFC unit: Signal source=FREQ,
                ; control method=One-Cycle on CX2,
                ; counter=16-bit RFC counter
SRF     $21     ; set RFC0 and RFC5 to output mode
SCX     3       ; clear SCF11, 12 and set SEF 1, 0.
SF2     $60     ; activate the two RFC units CX and CX2 simultaneously.
```

```
        SIE*        $40         ; set RFC function's interrupt release request flag.
        HALT
……
.org $0028                      ; RFC interrupt address。
        MAF         $60         ; read SCF11. SCF12
        JB0         CX_RLS      ; check CX for interrupt.
        JB1         CX2_RLS     ; check CX2 for interrupt.
……
CX_RLS:
        SCX         3           ; clear SCF11, 12 and set SEF 1, 0.
……
        SRF         $22         ; set RFC1 and RFC5 to output mode
        SF2         $60         ; activate the two RFC units CX and CX2 simultaneously.
        LDA         $60         ; check SCF12.
        JB1         CX2_RLS     ; check CX2 for interrupt.
        SIE*        $40         ; set RFC function's interrupt release request flag.
        RTS
```

Example 3: (RFC function for capacitance sensor application)

This example bases on the application circuit shown in 2-8 "RFC Basic Architecture and Application Circuit". The control method is using program instructions to control the RFC counter.

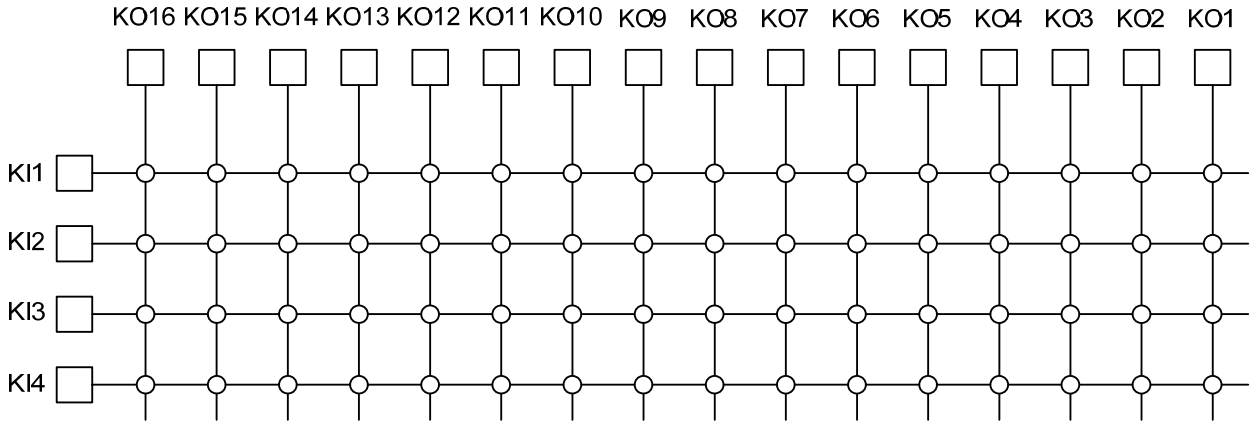
```
        SHE         $2          ; set TMR1 underflow to generate halt release
        SCNT        $20         ; set CX2's RFC unit, control method, 16-bit RFC counter
        SRF         $30         ; set RFC4, RFC5 to output mode and set up a RC
                                ; oscillator with a capacitance sensor.
                                ; in mask option, RFC5 pin must be set for connecting to a
                                ; resistor component, and RFC4 pin is set for connecting to
                                ; a capacitor component
        SF2         $40         ; enable CX2's RFC unit
        TMSX        $7F         ; TMR1's clock source is PH3, preset data is $3F.
        HALT
        RF2         $40         ; stop the CX2 pin's RFC function while TMR1 underflow
        MRF1        $10         ; read 16-bit counter
        MRF2        $11
        MRF3        $12
        MRF4        $13
        MSD         $14
        JB2         CNT1_OF     ; check RFC counter for overflow
        JMP         DATA_ACCEPT
CNT1_OF:
…
…
DATA_ACCEPT:
…
…
```

2-9. KEY MATRIX Scanning Function

The key matrix scanning function is made up of the four input pins KI1 ~ KI4, 16 output pins (shared with the LCD output pins SEG1 ~ SEG16. For ease of explanation, these will be referred to as KO1~KO16 in the rest of the document) and the external matrix keyboard.

The scanning signal on KO1 ~ KO16 will at a fixed interval set by PH6 and steal a small period from the SEG1 ~ SEG16 output waveform and transmit it (see 3-4-1). The KI~K4 input pins will read the scanned signal using the same interval as well to determine if a keyboard key was pressed.

The diagram below shows the standard application circuit of the Key matrix scanning function:

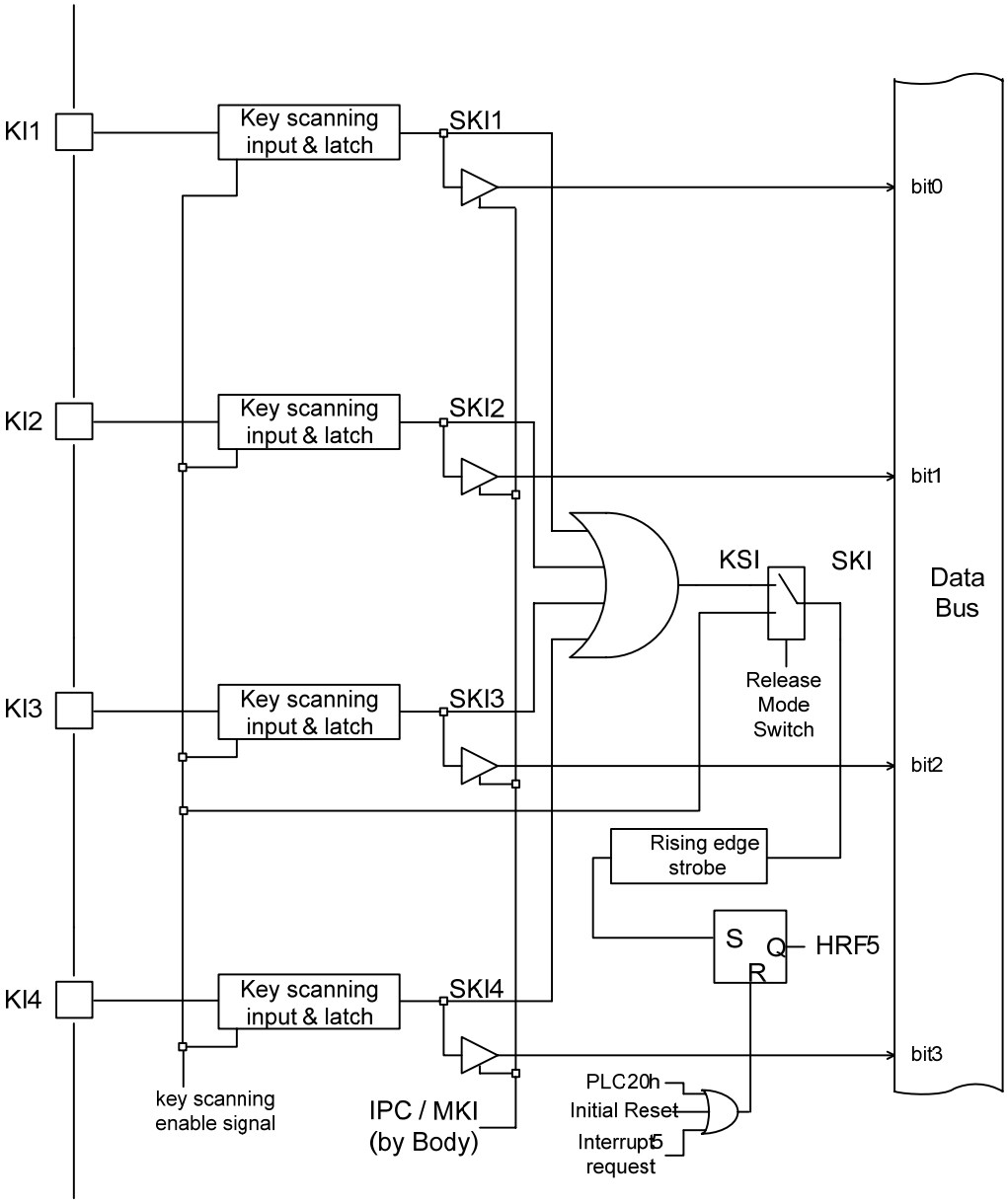


Note: KO16 = SEG16, KO15 = SEG15, and so on.

Executing instructions such as SPKXH D (2-word instruction), SPKRH D (2-word instruction), SPKTH(#) D, SPKX X, SPK Rx and SPK(#) @HL will set different scanning modes and ways of generating the halt release request flag 5 (HRF5).

Please refer to the Instructions manual for details.

The diagram below shows the control circuit and flag for the key matrix scanning function's input portion:



All TM89 Series MCUs use the PH6 signal as the scanning frequency of the key matrix scanning function and this is not user-configurable.

TM89 Series MCU also provide two mask option choices for the scanning signal width used for KO1~KO16 output, one is the PH0 period and the other is the PH1 period (please refer to the 3-4-2 Non-Overlap function). When using a shorter scanning signal width however take care to ensure that the MCU can correctly read the scanning signal even the signal delay cause by the parasitic RC loading on the key matrix scanning circuit.

2-9-1. Reading the KI1 ~ KI4 input signal

The key matrix scanning function will read the input signal on KI1~KI4 in synch with the scanning signal. The extracted signal states will then be written to the SKI1~SKI4 register. At this point the program can execute the MKI instruction to read the contents of SKI1~SKI4 then store it to the data memory in order to determine which key was pressed.

2-9-2. KEY MATRIX Scanning Function's HALT RELEASE REQUEST and INTERRUPT

If one or more high voltage level signal appears among the SKI1~SKI4 register, or if a scanning output signal in synch with the LCD waveform is generated, all these set the halt release request flag 5 (HRF5) to 1. The key matrix scanning function related instructions can be used to choose between these two methods. Executing the instruction PLC 20h clears HRF5.

Once the halt release request flag 5 (HRF5) is set to 1, if the halt release enable flag 5 (HEF5) is already set to 1, the MCU will generate a HALT release and set the start condition flag 8 (SCF8) to 1.

If the interrupt enable flag 5 (IEF5) was already set to 1 in this situation, the MCU will accept the interrupt request from the key matrix scanning function.

2-9-3. Using KI1 ~ KI4 as the MCU's RESET Pins

If this reset function was selected with mask option for the MCU, as long as KI1~KI4 (now set to the KI input pins) all detect a scanning signal at the same time, the MCU will enter RESET state and begin counting the RESET interval.

2-9-4. Program Example

Example:

```
        SPKXH 0                 ; set it so HRF5 is only set when there is a key press
        setdat      $FFFF       ; scan the entire keyboard within one scanning cycle
        PLC         $20         ; clear HRF5
        SHE         $20         ; set HEF5
        HALT                    ; wait for HRF5 to make the MCU generate a HALT
                                ; release
        MCX         $10         ; check SCF8 (SKI)
        JB0         ski_release
……..
……
ski_release:
        MKI         $10         ; read contents of SKI1 ~ SKI4
        JB0         ki1_release
        JB1         ki2_release
        JB2         ki3_release
        JB3         ki4_release
.
```

```
.
ki1_release:
        SPKXH 1                         ; check key on KI1 pin and switch to scanning cycle
        setdat      $0001               ; release mode.
        PLC         $20                 ; clear HRF5 to avoid abnormal HALT release
        CALL        wait_scan_again
                                        ; wait for next scanning signal. Wait time must be larger
                                        ; than scanning cycle.
        MKI         $10                 ; read content of SKI1
        JB0         ki1_seg1
…….
…….
        SPKXH 1                         ; send only the KO16 scanning signal
        setdat      $8000
        PLC         $20                 ; clear HRF5 to avoid abnormal HALT release
        CALL        wait_scan_again
                                        ; wait for next scanning signal. Waiting time must be larger
                                        ; than scanning cycle.
        MKI         $10                 ; read content of SKI1
        JB0         kil_seg16
…….
…….
wait_scan_again:
        HALT
        PLC     20h
        RTS
```

Chapter 3 LCD DRIVER

The T89 Series MCU has a built-in dot matrix LCD driver capable of displaying up to 1024 dots (64 SEGs x 16 COMs). The data of the content displayed by the LCD panel are all stored in the LCD display memory. The ON/OFF data for each dot is stored in a specific bit in the LCD display memory. The output waveform on the SEG pin changes according to data in each bit of the LCD display memory. The COM output pin of the LCD driver can be set to DC or P_open Drain using mask option.

When the MCU enters RESET state, the output waveform from the LCD driver will turn off all LCD dots. This LCD waveform for RESET state will continue until the first LCD related instruction is executed.

After the MCU enters normal mode, executing the instruction SF2 4h will temporarily turn off the entire LCD display regardless of the current data stored in the LCD display memory. Executing RF2 4h will recover all the pattern and display on the LCD panel again.

3-1. LCD LIGHTING SYSTEM

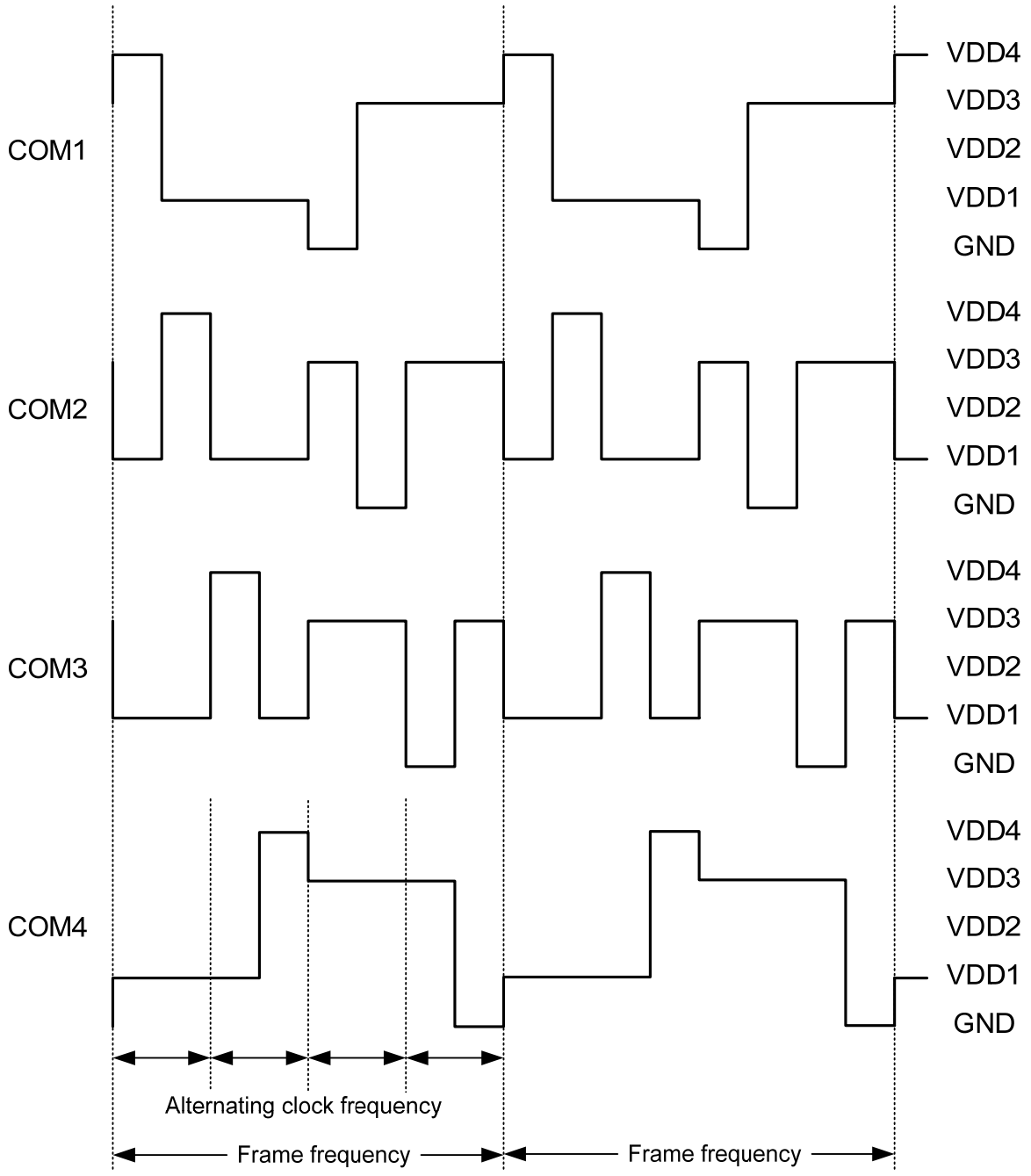
The LCD lighting system is made up of the LCD bias and LCD duty cycle, both selected using the mask option.

The following LCD duty cycles can be selected: 1/4 duty, 1/5 duty, 1/6 duty, 1/8 duty, 1/9 duty, 1/10 duty, 1/11 duty, 1/12 duty, 1/13 duty, 1/14 duty, 1/15 duty, 1/16 duty.

The following LCD bias can be selected: 1/3 bias, 1/4 bias, 1/5 bias.

The frequency of any repeating waveform output on the COM pin can be used to represent the LCD frame frequency. Every LCD lighting system has its own LCD frame frequency and this can be calculated using the LCD's alternating clock frequency and duty cycle. The user can use the mask option to select the output signal (PH3~PH10) from the pre-divider as the LCD alternating clock.

The diagram below explains the waveform's relationship between the frame frequency and alternating clock frequency:



All of the LCD frame frequencies were calculated based on the reference frequency from PH0 (32768Hz). The formula is as follows:

LCD frame frequency = LCD alternating clock frequency * LCD duty cycle

For example, if the desired LCD frame frequency is 51Hz using the 1/5 duty cycle, the equation for the LCD alternating clock frequency is as shown below:

51Hz (frame frequency)=alternating clock frequency x 1/5 duty

The alternating clock frequency is then 255Hz. This frequency approximates most closely to the pre-divider output signal PH7 (256Hz).

The table below shows the relationship between LCD frame frequency (number in the middle), LCD duty cycle and pre-divider output signal (alternating clock):

DUTY	PH3	PH4	PH5	PH6	PH7	PH8	PH9	PH10
1/4	1024	512	256	128	64	32	16	8
1/5	819	410	205	102	51	26	13	6
1/6	683	341	171	85	43	21	11	5
1/7	585	293	146	73	37	18	9	5
1/8	512	256	128	64	32	16	8	4
1/9	455	228	114	57	28	14	7	4
1/10	410	205	102	51	26	13	6	3
1/11	372	186	93	47	23	12	6	3
1/12	341	171	85	43	21	11	5	3
1/13	315	158	79	39	20	10	5	2
1/14	293	146	73	37	18	9	5	2
1/15	273	137	68	34	17	9	4	2
1/16	256	128	64	32	16	8	4	2

When choosing the LCD frame frequency, it is recommended to choose one higher than 24Hz or the LCD screen will show obvious flickering.

3-2. LCD DISPLAY MEMORY

256x4 bit (0100h~10FFh) of data memory is allocated by the MCU to the LCD display memory. These store the pattern data for 1024 LCD dots.

Each LCD dot has its own corresponding COM and SEG output pin. The table below shows how each LCD corresponds to a specific bit in the LCD display memory:

Reference Table showing how LCD display memory corresponds to COM1~COM8 and SEG1~SEG64:

ADDRESS		BIT3	BIT2	BIT1	BIT0	ADDRESS		BIT3	BIT2	BIT1	BIT0
		COM4	COM3	COM2	COM1			COM8	COM7	COM6	COM5
Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA	Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA
Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE	Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE
0100H	0	SEG1	SEG1	SEG1	SEG1	0140H	20	SEG1	SEG1	SEG1	SEG1
0101H		SEG2	SEG2	SEG2	SEG2	0141H		SEG2	SEG2	SEG2	SEG2
0102H	1	SEG3	SEG3	SEG3	SEG3	0142H	21	SEG3	SEG3	SEG3	SEG3
0103H		SEG4	SEG4	SEG4	SEG4	0143H		SEG4	SEG4	SEG4	SEG4
0104H	2	SEG5	SEG5	SEG5	SEG5	0144H	22	SEG5	SEG5	SEG5	SEG5
0105H		SEG6	SEG6	SEG6	SEG6	0145H		SEG6	SEG6	SEG6	SEG6
0106H	3	SEG7	SEG7	SEG7	SEG7	0146H	23	SEG7	SEG7	SEG7	SEG7
0107H		SEG8	SEG8	SEG8	SEG8	0147H		SEG8	SEG8	SEG8	SEG8
0108H	4	SEG9	SEG9	SEG9	SEG9	0148H	24	SEG9	SEG9	SEG9	SEG9
0109H		SEG10	SEG10	SEG10	SEG10	0149H		SEG10	SEG10	SEG10	SEG10
010AH	5	SEG11	SEG11	SEG11	SEG11	014AH	25	SEG11	SEG11	SEG11	SEG11
010BH		SEG12	SEG12	SEG12	SEG12	014BH		SEG12	SEG12	SEG12	SEG12
010CH	6	SEG13	SEG13	SEG13	SEG13	014CH	26	SEG13	SEG13	SEG13	SEG13
010DH		SEG14	SEG14	SEG14	SEG14	014DH		SEG14	SEG14	SEG14	SEG14
010EH	7	SEG15	SEG15	SEG15	SEG15	014EH	27	SEG15	SEG15	SEG15	SEG15
010FH		SEG16	SEG16	SEG16	SEG16	014FH		SEG16	SEG16	SEG16	SEG16
0110H	8	SEG17	SEG17	SEG17	SEG17	0150H	28	SEG17	SEG17	SEG17	SEG17
0111H		SEG18	SEG18	SEG18	SEG18	0151H		SEG18	SEG18	SEG18	SEG18
0112H	9	SEG19	SEG19	SEG19	SEG19	0152H	29	SEG19	SEG19	SEG19	SEG19
0113H		SEG20	SEG20	SEG20	SEG20	0153H		SEG20	SEG20	SEG20	SEG20
0114H	0A	SEG21	SEG21	SEG21	SEG21	0154H	2A	SEG21	SEG21	SEG21	SEG21
0115H		SEG22	SEG22	SEG22	SEG22	0155H		SEG22	SEG22	SEG22	SEG22
0116H	0B	SEG23	SEG23	SEG23	SEG23	0156H	2B	SEG23	SEG23	SEG23	SEG23
0117H		SEG24	SEG24	SEG24	SEG24	0157H		SEG24	SEG24	SEG24	SEG24
0118H	0C	SEG25	SEG25	SEG25	SEG25	0158H	2C	SEG25	SEG25	SEG25	SEG25
0119H		SEG26	SEG26	SEG26	SEG26	0159H		SEG26	SEG26	SEG26	SEG26
011AH	0D	SEG27	SEG27	SEG27	SEG27	015AH	2D	SEG27	SEG27	SEG27	SEG27
011BH		SEG28	SEG28	SEG28	SEG28	015BH		SEG28	SEG28	SEG28	SEG28
011CH	0E	SEG29	SEG29	SEG29	SEG29	015CH	2E	SEG29	SEG29	SEG29	SEG29
011DH		SEG30	SEG30	SEG30	SEG30	015DH		SEG30	SEG30	SEG30	SEG30
011EH	0F	SEG31	SEG31	SEG31	SEG31	015EH	2F	SEG31	SEG31	SEG31	SEG31

ADDRESS		BIT3	BIT2	BIT1	BIT0	ADDRESS		BIT3	BIT2	BIT1	BIT0
		COM4	COM3	COM2	COM1			COM8	COM7	COM6	COM5
Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA	Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA
Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE	Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE
011FH		SEG32	SEG32	SEG32	SEG32	015FH		SEG32	SEG32	SEG32	SEG32
0120H	10	SEG33	SEG33	SEG33	SEG33	0160H	30	SEG33	SEG33	SEG33	SEG33
0121H		SEG34	SEG34	SEG34	SEG34	0161H		SEG34	SEG34	SEG34	SEG34
0122H	11	SEG35	SEG35	SEG35	SEG35	0162H	31	SEG35	SEG35	SEG35	SEG35
0123H		SEG36	SEG36	SEG36	SEG36	0163H		SEG36	SEG36	SEG36	SEG36
0124H	12	SEG37	SEG37	SEG37	SEG37	0164H	32	SEG37	SEG37	SEG37	SEG37
0125H		SEG38	SEG38	SEG38	SEG38	0165H		SEG38	SEG38	SEG38	SEG38
0126H	13	SEG39	SEG39	SEG39	SEG39	0166H	33	SEG39	SEG39	SEG39	SEG39
0127H		SEG40	SEG40	SEG40	SEG40	0167H		SEG40	SEG40	SEG40	SEG40
0128H	14	SEG41	SEG41	SEG41	SEG41	0168H	34	SEG41	SEG41	SEG41	SEG41
0129H		SEG42	SEG42	SEG42	SEG42	0169H		SEG42	SEG42	SEG42	SEG42
012AH	15	SEG43	SEG43	SEG43	SEG43	016AH	35	SEG43	SEG43	SEG43	SEG43
012BH		SEG44	SEG44	SEG44	SEG44	016BH		SEG44	SEG44	SEG44	SEG44
012CH	16	SEG45	SEG45	SEG45	SEG45	016CH	36	SEG45	SEG45	SEG45	SEG45
012DH		SEG46	SEG46	SEG46	SEG46	016DH		SEG46	SEG46	SEG46	SEG46
012EH	17	SEG47	SEG47	SEG47	SEG47	016EH	37	SEG47	SEG47	SEG47	SEG47
012FH		SEG48	SEG48	SEG48	SEG48	016FH		SEG48	SEG48	SEG48	SEG48
0130H	18	SEG49	SEG49	SEG49	SEG49	0170H	38	SEG49	SEG49	SEG49	SEG49
0131H		SEG50	SEG50	SEG50	SEG50	0171H		SEG50	SEG50	SEG50	SEG50
0132H	19	SEG51	SEG51	SEG51	SEG51	0172H	39	SEG51	SEG51	SEG51	SEG51
0133H		SEG52	SEG52	SEG52	SEG52	0173H		SEG52	SEG52	SEG52	SEG52
0134H	1A	SEG53	SEG53	SEG53	SEG53	0174H	3A	SEG53	SEG53	SEG53	SEG53
0135H		SEG54	SEG54	SEG54	SEG54	0175H		SEG54	SEG54	SEG54	SEG54
0136H	1B	SEG55	SEG55	SEG55	SEG55	0176H	3B	SEG55	SEG55	SEG55	SEG55
0137H		SEG56	SEG56	SEG56	SEG56	0177H		SEG56	SEG56	SEG56	SEG56
0138H	1C	SEG57	SEG57	SEG57	SEG57	0178H	3C	SEG57	SEG57	SEG57	SEG57
0139H		SEG58	SEG58	SEG58	SEG58	0179H		SEG58	SEG58	SEG58	SEG58
013AH	1D	SEG59	SEG59	SEG59	SEG59	017AH	3D	SEG59	SEG59	SEG59	SEG59
013BH		SEG60	SEG60	SEG60	SEG60	017BH		SEG60	SEG60	SEG60	SEG60
013CH	1E	SEG61	SEG61	SEG61	SEG61	017CH	3E	SEG61	SEG61	SEG61	SEG61
013DH		SEG62	SEG62	SEG62	SEG62	017DH		SEG62	SEG62	SEG62	SEG62
013EH	1F	SEG63	SEG63	SEG63	SEG63	017EH	3F	SEG63	SEG63	SEG63	SEG63
013FH		SEG64	SEG64	SEG64	SEG64	017FH		SEG64 C_DC8 C_OD8	SEG64 C_DC7 C_OD7	SEG64 C_DC6 C_OD6	SEG64 C_DC5 C_OD5

Reference Table showing how LCD display memory corresponds to COM9~COM16 and SEG1~SEG64:

ADDRESS		BIT3	BIT2	BIT1	BIT0	ADDRESS		BIT3	BIT2	BIT1	BIT0
		COM12	COM11	COM10	COM9			COM16	COM15	COM14	COM13
Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA	Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA
Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE	Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE
0180H	40	SEG1	SEG1	SEG1	SEG1	01C0H	60	SEG1	SEG1	SEG1	SEG1
0181H		SEG2	SEG2	SEG2	SEG2	01C1H		SEG2	SEG2	SEG2	SEG2
0182H	41	SEG3	SEG3	SEG3	SEG3	01C2H	61	SEG3	SEG3	SEG3	SEG3
0183H		SEG4	SEG4	SEG4	SEG4	01C3H		SEG4	SEG4	SEG4	SEG4
0184H	42	SEG5	SEG5	SEG5	SEG5	01C4H	62	SEG5	SEG5	SEG5	SEG5
0185H		SEG6	SEG6	SEG6	SEG6	01C5H		SEG6	SEG6	SEG6	SEG6
0186H	43	SEG7	SEG7	SEG7	SEG7	01C6H	63	SEG7	SEG7	SEG7	SEG7
0187H		SEG8	SEG8	SEG8	SEG8	01C7H		SEG8	SEG8	SEG8	SEG8
0188H	44	SEG9	SEG9	SEG9	SEG9	01C8H	64	SEG9	SEG9	SEG9	SEG9
0189H		SEG10	SEG10	SEG10	SEG10	01C9H		SEG10	SEG10	SEG10	SEG10
018AH	45	SEG11	SEG11	SEG11	SEG11	01CAH	65	SEG11	SEG11	SEG11	SEG11
018BH		SEG12	SEG12	SEG12	SEG12	01CBH		SEG12	SEG12	SEG12	SEG12
018CH	46	SEG13	SEG13	SEG13	SEG13	01CCH	66	SEG13	SEG13	SEG13	SEG13
018DH		SEG14	SEG14	SEG14	SEG14	01CDH		SEG14	SEG14	SEG14	SEG14
018EH	47	SEG15	SEG15	SEG15	SEG15	01CEH	67	SEG15	SEG15	SEG15	SEG15
018FH		SEG16	SEG16	SEG16	SEG16	01CFH		SEG16	SEG16	SEG16	SEG16
0190H	48	SEG17	SEG17	SEG17	SEG17	01D0H	68	SEG17	SEG17	SEG17	SEG17
0191H		SEG18	SEG18	SEG18	SEG18	01D1H		SEG18	SEG18	SEG18	SEG18
0192H	49	SEG19	SEG19	SEG19	SEG19	01D2H	69	SEG19	SEG19	SEG19	SEG19
0193H		SEG20	SEG20	SEG20	SEG20	01D3H		SEG20	SEG20	SEG20	SEG20
0194H	4A	SEG21	SEG21	SEG21	SEG21	01D4H	6A	SEG21	SEG21	SEG21	SEG21
0195H		SEG22	SEG22	SEG22	SEG22	01D5H		SEG22	SEG22	SEG22	SEG22
0196H	4B	SEG23	SEG23	SEG23	SEG23	01D6H	6B	SEG23	SEG23	SEG23	SEG23
0197H		SEG24	SEG24	SEG24	SEG24	01D7H		SEG24	SEG24	SEG24	SEG24
0198H	4C	SEG25	SEG25	SEG25	SEG25	01D8H	6C	SEG25	SEG25	SEG25	SEG25
0199H		SEG26	SEG26	SEG26	SEG26	01D9H		SEG26	SEG26	SEG26	SEG26
019AH	4D	SEG27	SEG27	SEG27	SEG27	01DAH	6D	SEG27	SEG27	SEG27	SEG27
019BH		SEG28	SEG28	SEG28	SEG28	01DBH		SEG28	SEG28	SEG28	SEG28
019CH	4E	SEG29	SEG29	SEG29	SEG29	01DCH	6E	SEG29	SEG29	SEG29	SEG29
019DH		SEG30	SEG30	SEG30	SEG30	01DDH		SEG30	SEG30	SEG30	SEG30
019EH	4F	SEG31	SEG31	SEG31	SEG31	01DEH	6F	SEG31	SEG31	SEG31	SEG31
019FH		SEG32	SEG32	SEG32	SEG32	01DFH		SEG32	SEG32	SEG32	SEG32
01A0H	50	SEG33	SEG33	SEG33	SEG33	01E0H	70	SEG33	SEG33	SEG33	SEG33
01A1H		SEG34	SEG34	SEG34	SEG34	01E1H		SEG34	SEG34	SEG34	SEG34
01A2H	51	SEG35	SEG35	SEG35	SEG35	01E2H	71	SEG35	SEG35	SEG35	SEG35
01A3H		SEG36	SEG36	SEG36	SEG36	01E3H		SEG36	SEG36	SEG36	SEG36
01A4H	52	SEG37	SEG37	SEG37	SEG37	01E4H	72	SEG37	SEG37	SEG37	SEG37
01A5H		SEG38	SEG38	SEG38	SEG38	01E5H		SEG38	SEG38	SEG38	SEG38
01A6H	53	SEG39	SEG39	SEG39	SEG39	01E6H	73	SEG39	SEG39	SEG39	SEG39
01A7H		SEG40	SEG40	SEG40	SEG40	01E7H		SEG40	SEG40	SEG40	SEG40
01A8H	54	SEG41	SEG41	SEG41	SEG41	01E8H	74	SEG41	SEG41	SEG41	SEG41
01A9H		SEG42	SEG42	SEG42	SEG42	01E9H		SEG42	SEG42	SEG42	SEG42
01AAH	55	SEG43	SEG43	SEG43	SEG43	01EAH	75	SEG43	SEG43	SEG43	SEG43
01ABH		SEG44	SEG44	SEG44	SEG44	01EBH		SEG44	SEG44	SEG44	SEG44

ADDRESS		BIT3	BIT2	BIT1	BIT0	ADDRESS		BIT3	BIT2	BIT1	BIT0
		COM12	COM11	COM10	COM9			COM16	COM15	COM14	COM13
Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA	Rx(even)	Lz	DBUSD	DBUSC	DBUSB	DBUSA
Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE	Rx(odd)		DBUSH	DBUSG	DBUSF	DBUSE
01ACH	56	SEG45	SEG45	SEG45	SEG45	01ECH	76	SEG45	SEG45	SEG45	SEG45
01ADH		SEG46	SEG46	SEG46	SEG46	01EDH		SEG46	SEG46	SEG46	SEG46
01AEH	57	SEG47	SEG47	SEG47	SEG47	01EEH	77	SEG47	SEG47	SEG47	SEG47
01AFH		SEG48	SEG48	SEG48	SEG48	01EFH		SEG48	SEG48	SEG48	SEG48
01B0H	58	SEG49	SEG49	SEG49	SEG49	01F0H	78	SEG49	SEG49	SEG49	SEG49
01B1H		SEG50	SEG50	SEG50	SEG50	01F1H		SEG50	SEG50	SEG50	SEG50
01B2H	59	SEG51	SEG51	SEG51	SEG51	01F2H	79	SEG51	SEG51	SEG51	SEG51
01B3H		SEG52	SEG52	SEG52	SEG52	01F3H		SEG52	SEG52	SEG52	SEG52
01B4H	5A	SEG53	SEG53	SEG53	SEG53	01F4H	7A	SEG53	SEG53	SEG53	SEG53
01B5H		SEG54	SEG54	SEG54	SEG54	01F5H		SEG54	SEG54	SEG54	SEG54
01B6H	5B	SEG55	SEG55	SEG55	SEG55	01F6H	7B	SEG55	SEG55	SEG55	SEG55
01B7H		SEG56	SEG56	SEG56	SEG56	01F7H		SEG56	SEG56	SEG56	SEG56
01B8H	5C	SEG57	SEG57	SEG57	SEG57	01F8H	7C	SEG57	SEG57	SEG57	SEG57
01B9H		SEG58	SEG58	SEG58	SEG58	01F9H		SEG58	SEG58	SEG58	SEG58
01BAH	5D	SEG59	SEG59	SEG59	SEG59	01FAH	7D	SEG59	SEG59	SEG59	SEG59
01BBH		SEG60	SEG60	SEG60	SEG60	01FBH		SEG60	SEG60	SEG60	SEG60
01BCH	5E	SEG61	SEG61	SEG61	SEG61	01FCH	7E	SEG61	SEG61	SEG61	SEG61
01BDH		SEG62	SEG62	SEG62	SEG62	01FDH		SEG62	SEG62	SEG62	SEG62
01BEH	5F	SEG63	SEG63	SEG63	SEG63	01FEH	7F	SEG63	SEG63	SEG63	SEG63
01BFH		SEG64 C_DC12 C_OD12	SEG64 C_DC11 C_OD11	SEG64 C_DC10 C_OD10	SEG64 C_DC9 C_OD9	01FFH		SEG64 C_DC16 C_OD16	SEG64 C_DC15 C_OD15	SEG64 CDC14 COD14	SEG64 C_DC13 C_OD13

3-3. Using COM Pin as Normal OUTPUT PIN

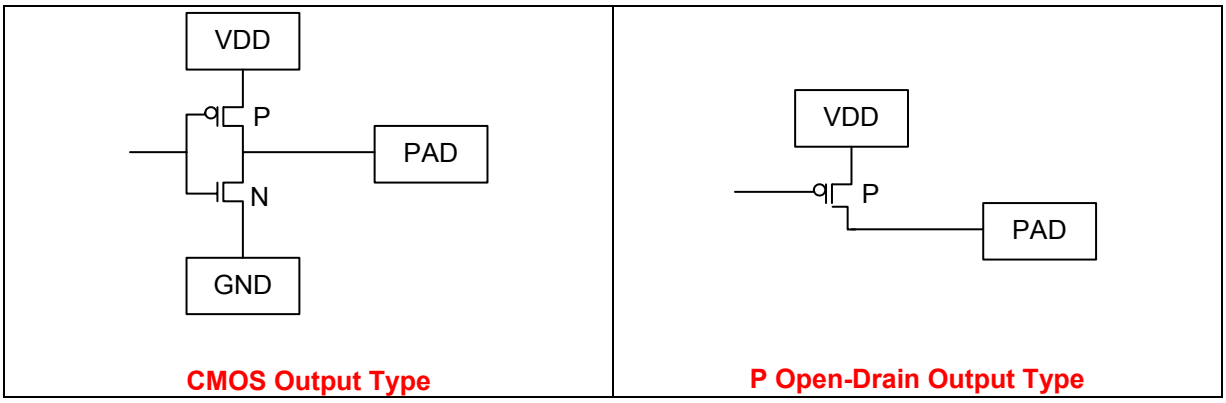
The chosen LCD duty cycle option setting may lead to some COM output pins being not used for the LCD driver. These COM pins can then be set through a mask option as CMOS type DC output or P open-drain output pins without affecting other COM output pins used by the LCD driver.

For the TM89 Series MCU only COM5~COM16 can be used as output pins. The output data for these output pins are all stored in their corresponding bit within the LCD display memory. To change the output signal from these output pins, simply change the data in the corresponding bit within the LCD display memory.

The reference table in 3-2 shows that the data for the COM5~COM8 output pins are stored at address 017Fh in the data RAM; the data for the COM9~COM12 output pins are stored at the address 01BFh in the data RAM; the data for the COM13~COM16 output pins are stored at address 01FFh in the data RAM.

After a COM output pin is altered to output pin, its output signal will not be affected if the program enters STOP mode or if the LCD display is turned off by executing the instruction SF2 4h.

The structure of the CMOS output type and P open-drain type is shown below:



Only unused COM pins can be set as output pins. When setting COM pins as output pins, the sequence of the pin order of the COM pins must be maintained as well.

For example: When the LCD is using 1/5 duty cycle COM1~COM5 must be reserved for the LCD driver. Only COM6~COM16 can then be used for DC output. You cannot assign COM1~COM4 and COM6 as the LCD driver then set COM5 as the DC output.

3-4. Non-Overlap Signal Output on the COM and SEG Pins

The LCD waveforms output from the COM and SEG pins can be inserted a non-overlap signal, using the mask option can choose to insert the non-overlap signal into the LCD waveform or not and set the width of the non-overlap signal. The non-overlap signal serves two purposes. One is to slightly reduce the LCD driver's own power consumption and the other is to generate the Key matrix function's scanning signal.

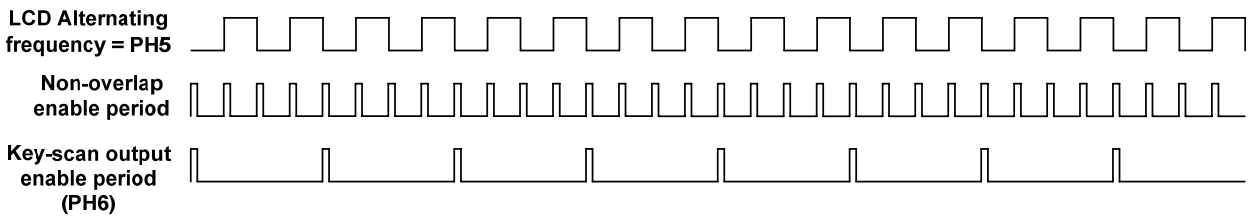
The non-overlap signal is outputted at certain times with the LCD waveform on the COM and SEG pins. In normal condition the non-overlap signal will be outputted on the rising and falling edge of the LCD alternating clock but there are some exceptions. *Please refer to 3-4-1 for more details.*

If the MCU does not need to use the non-overlap signal, this can be turned off using the mask option.

3-4-1. Using the Non-Overlap Signal as the Key Matrix Scanning Signal

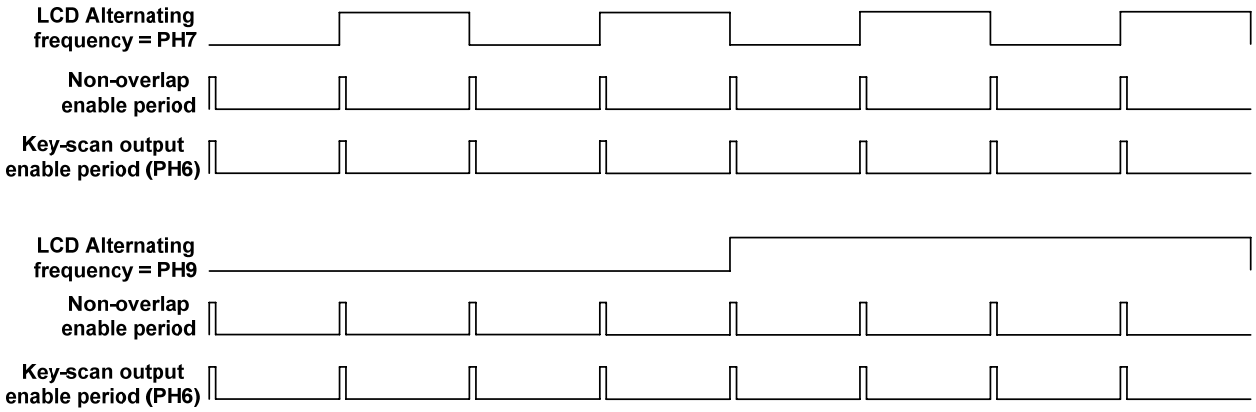
When using the key matrix scanning function the MCU must use mask option to set the non-overlap signal's output function to "all used".

If the LCD alternating clock frequency is higher than PH6, the LCD output signal on the COM and SEG pins will be altered to Hi-Z state (non-overlap signal) on the rising and falling edge of the LCD alternating clock. SEG1~SEG16 will however alter the non-overlap signal to the scanning signal on the falling edge of PH6 clock. This is shown in the diagram below:



If the LCD alternating clock frequency is lower than PH6, the LCD output signal on the COM and SEG pins will be altered to Hi-Z state (non-overlap signal) on the falling edge of PH6 clock. SEG1~SEG16 will also alter the non-overlap signal to scanning signal at the same time.

In this case, if the LCD alternating clock frequency is lower than the PH6's frequency, the output of the non-overlap signal will trail behind the falling edge of PH6 clock and doesn't change with the LCD alternating clock's frequency. This is shown in the diagram below:



When the MCU enters RESET state, the non-overlap signal output function begin outputting the GND level scanning signal on the SEG1~SEG16 pins. Once the program executes an instruction like SPKTH, SPKXH, SPKRH and SPK, only the non-overlap signals on the SEG pins specified by these instructions will output the GND level scanning signal, the non-overlap signal output from the other SEG pins will all output the Hi-Z state.

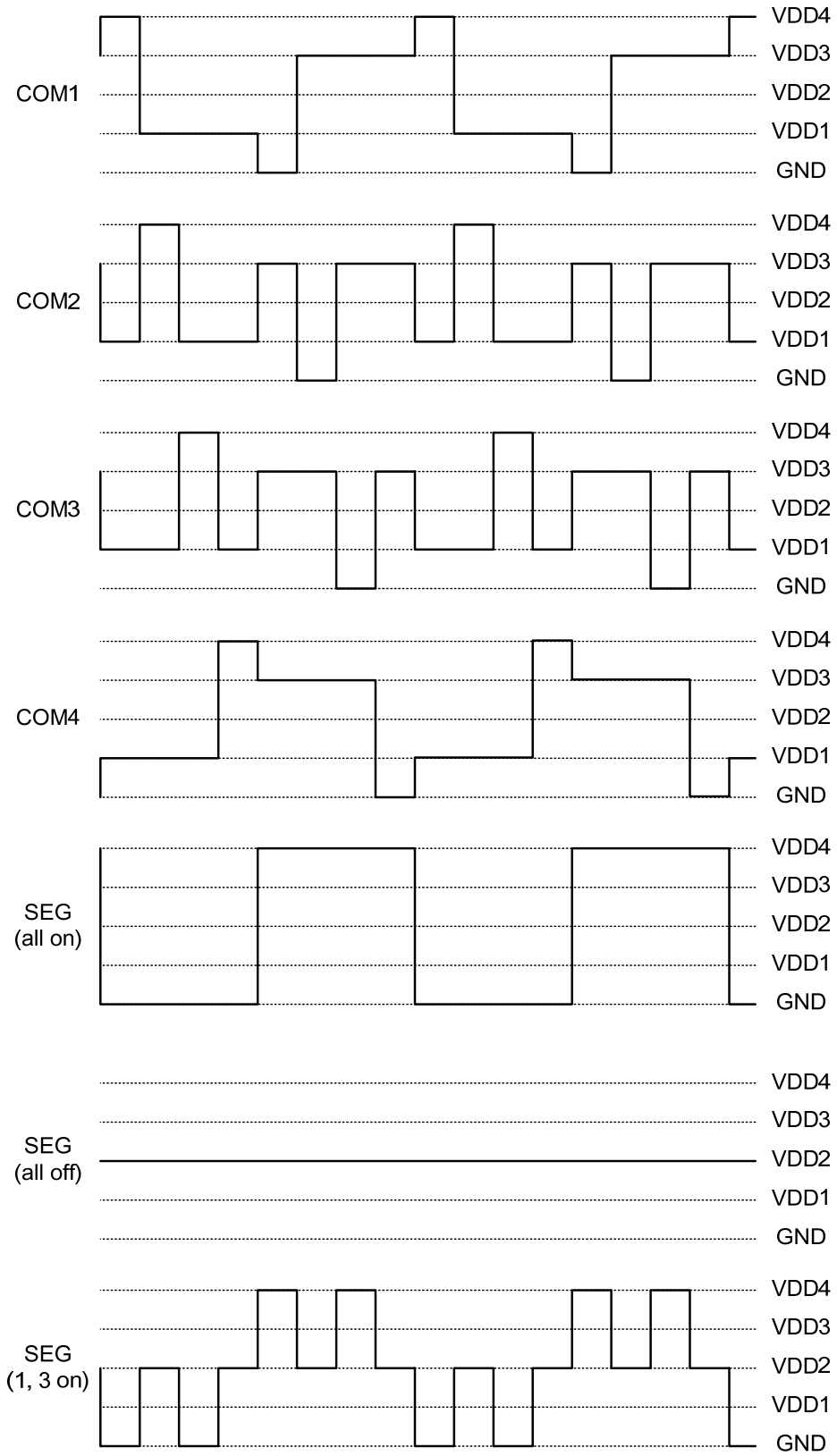
3-4-2. Using the Non-Overlap Signal to Reduce the LCD Driver's Power Consumption

If the user wishes to use the non-overlap signal to reduce the LCD driver's power consumption, mask option can be used to set the non-overlap signal output function to "al Hi-Z".

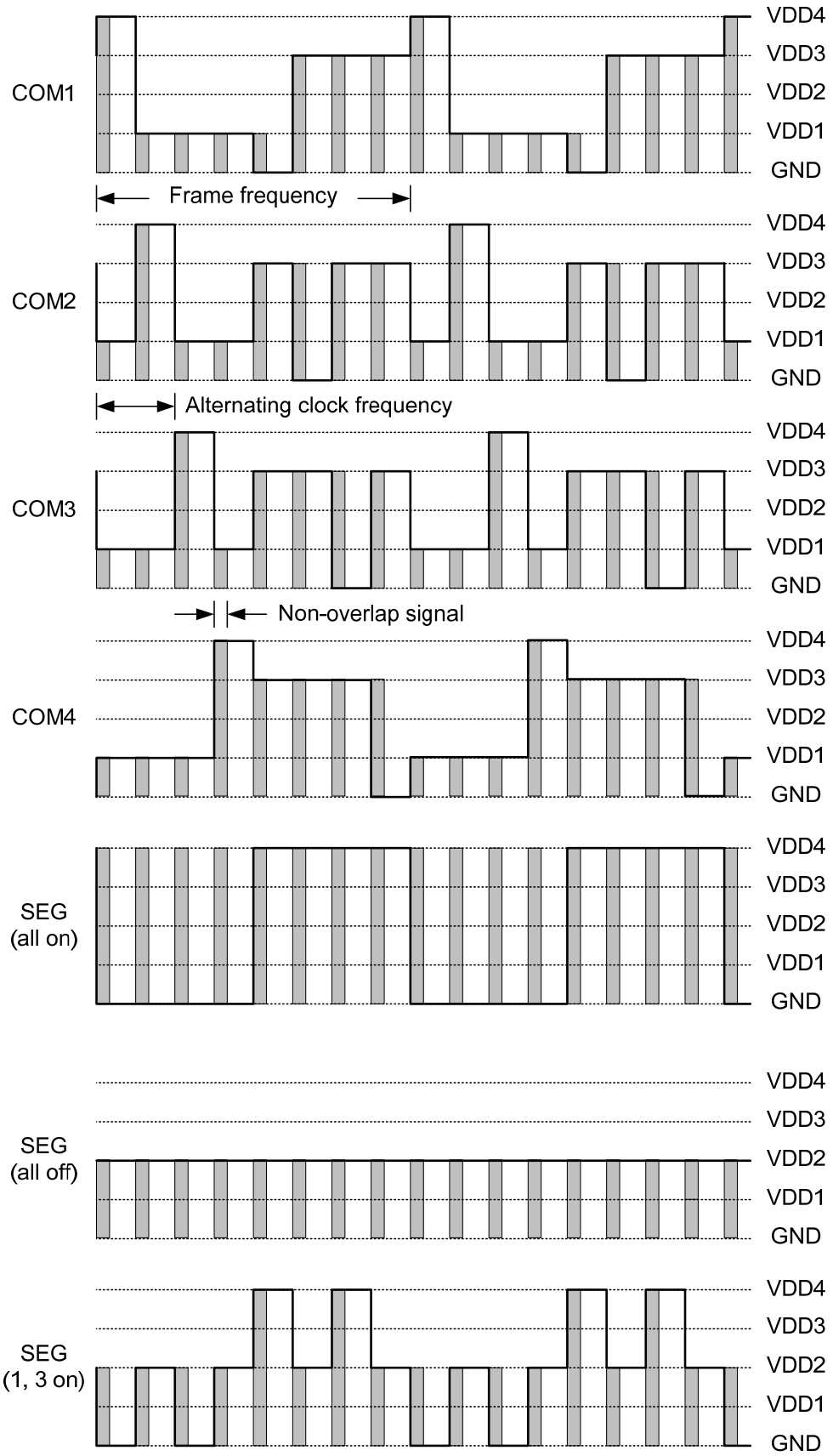
When the non-overlap signal output function is enabled, all COM and SEG output signals will alter the non-overlap signal that trailing behind the LCD alternating clock signal's rising or falling edge to Hi-Z state regardless of the LCD bias or duty cycle used.

In the diagram below, the example LCD waveform is set to 1/4 bias, 1/4 duty and alternating clock is PH7. This shows the difference in the COM and SEG output pins when the non-overlap signal output function is enabled or disabled.

Below is the COM and SEG output waveforms when the non-overlap signal output function is disabled:



The COM and SEG output waveforms after the non-overlap signal output function is enabled is shown below. The non-overlap signal is in gray bar.


COM1
VDD4
VDD3
VDD2
VDD1
GND
Frame frequency
COM2
VDD4
VDD3
VDD2
VDD1
GND
Alternating clock frequency
COM3
VDD4
VDD3
VDD2
VDD1
GND
Non-overlap signal
COM4
VDD4
VDD3
VDD2
VDD1
GND
SEG
(all on)
VDD4
VDD3
VDD2
VDD1
GND
SEG
(all off)
VDD4
VDD3
VDD2
VDD1
GND
SEG
(1, 3 on)
VDD4
VDD3
VDD2
VDD1
GND

3-4-3. Setting the Non-Overlap Signal width

Once the non-overlap signal output function is enabled, the MCU can use another mask option to set the non-overlap signal width. This is achieved either by setting the PH0 signal period or the PH1 signal period.

When the non-overlap signal is being used as the key matrix scanning signal, please note that using the PH0 period can reduce the difference in brightness between SEG1~16 and other segment pins of the LCD display. However, if the MCU isn't able to correctly read the scanning signal due to heavy RC loading from external key matrix scanning circuitry, we recommend using PH1's period instead.

Appendix Mask Option Table for the TM89 Series MCU

Appendix content:

An explanation of each MCU's mask option purpose and usage with the TM8959 for example.

A-1. Power

A-1-1. POWER SOURCE

This mask option selects the power supply voltage (VBAT) used by the MCU and the operating voltage (BAK) used by the MCU's internal logic after exiting backup mode (BCF=0).

The mask options are:

(1) 3V BATTERY(BAK=VBAT for BCF=0)

Explanation: MCU to use 3V power supply. After exiting backup mode (BCF=0) MCU's internal logic will use operating voltage same as VBAT.

(2) 3V BATTERY(BAK=VL1 for BCF=0 , VL1 > 1.10V)

Explanation: MCU to use 3V power supply. After exiting backup mode (BCF=0) MCU's internal logic will use operating voltage same as VL1.

(3) 1.5V BATTERY

Explanation: MCU to use 1.5V power supply.

A-1-2. EXTERNAL LCD REGULATOR for 3V POWER MODE

This mask option chooses whether the charge-pump base voltage used by the MCU's LCD driver is to be provided by an external voltage regulator.

After this option is set to VL1 or VL2, when BCF=0 (BAK =VL1) the 32KHz oscillator's driving capability will be increased (equivalent to the driving capability used for BCF=1 under 1.5V BATTERY mode). This will avoid the 32KHz oscillator to stop oscillation when using the lower voltage of VL1 or VL2 with external voltage regulation.

The mask options are:

(1) VL1

Explanation: When MCU uses the 3V power supply voltage, the charge-pump's base voltage is provided by VL1. Here the VL1 voltage is supplied by an external voltage regulator.

(2) VL2

Explanation: When MCU uses the 3V power supply voltage, the charge-pump's base voltage is provided by VL2.

(3) NO USE

Explanation: The charge-pump's base voltage will be automatically selected according to the mask option of A-1-1.

When power source is set to "1.5V BATTERY", the charge-pump's base voltage is supplied by VL1.

When power source is set to "3V BATTERY", the charge-pump's base voltage is supplied by VL2.

A-2. O.S.C.

A-2-1. CLOCK SOURCE

This mask option selects the operating mode of the system clock.

The mask options are:

(1) FAST ONLY

Explanation: The MCU can only use the CF clock as the system clock.

(2) SLOW ONLY

Explanation: The MCU can only use the XT clock as the system clock.

(3) DUAL

Explanation: The MCU can use both XT clock and CF clock as the system clock.

A-2-2. PH0 <-> BCLK FOR FAST ONLY

This mask option is only usable when the A-2-1 mask option is set to “Fast Only”, and can perform frequency division on CF Clock frequency before supplying it to PH0.

The mask options are:

(1) PH0=BCLK

(2) PH0=BCLK/4

(3) PH0=BCLK/8

(4) PH0=BCLK/16

(5) PH0=BCLK/32

(6) PH0=BCLK/64

(7) PH0=BCLK/128

(8) PH0=BCLK/256

A-2-3. SLOW CLOCK TYPE FOR SLOW ONLY OR DUAL

This mask option selects the circuit types of XT OSC.

The mask options are:

(1) X'tal

Explanation: 32KHz Crystal oscillator. This option is only usable when the A-2-1 mask option is set to Slow Only or Dual.

(2) RC

Explanation: RC oscillator. This option is only usable when the A-2-1 mask option is set to Slow Only or Dual.

(3) No Use for FAST ONLY

Explanation: None of slow clock is used. This option is only usable when the A-2-1 mask option is set to Fast Only.

A-2-4. FAST CLOCK TYPE FOR FAST ONLY OR DUAL

This mask option selects the circuit types of XT OSC.

The mask options are:

(1) INTERNAL RESISTOR FOR 250KHz

Explanation: Built-in 250KHz RC oscillator. This option is only usable when the A-2-1 mask option is set to “Fast Only” or “Dual”.

(2) INTERNAL RESISTOR FOR 500KHz

Explanation: Built-in 500KHz RC oscillator. This option is only usable when the A-2-1 mask option is set to “Fast Only” or “Dual”.

(3) EXTERNAL RESISTOR

Explanation: RC oscillator with external resistor. This option is only usable when the A-2-1 mask option is set to “Fast Only” or “Dual”.

(4) 3.58MHz CERAMIC RESONATOR

Explanation: 3.58MHz Crystal/ceramic resonator oscillator. This option is only usable when the A-2-1 mask option is set to “Fast Only” or “Dual”. The A-1-1 mask option must also be set to “3V battery”.

(5) No Use for SLOW ONLY

Explanation: None of fast clock is used. This option is only usable when the A-2-1 mask option is set to “Slow Only”.

A-3. RESET

A-3-1. POWER ON RESET

This mask options enables/disables the power on reset function.

The mask options are:

(1) USE

Explanation: This option enables the MCU's internal power on reset function.

(2) NO USE

Explanation: This option disables the power on reset function. The MCU can't generate its own power on reset signal and can only reset by sending a power on reset signal generated by an external component to the reset pin.

A-3-2. WATCH DOG TIMER OVERFLOW TIME INTERVAL

This mask option enables/disables the watch dog timer function and sets the timer overflow interval.

The mask options are:

(1) 8 X PH10

Explanation: The overflow interval of watch dog timer is 8 time periods of PH10 clock.

(2) 64 X PH10

Explanation: The overflow interval of watch dog timer is 64 time periods of PH10 clock.

(3) 512 X PH10

Explanation: The overflow interval of watch dog timer is 512 time periods of PH10 clock.

(4) NO USE

Explanation: This option disables the watch dog timer function.

A-3-3. RESET TIME

This mask option is used to set the MCU's RESET time interval.

The mask options are:

(1) PH15/2

Explanation: MCU's RESET time interval is 1/2 period of PH15 clock.

(2) PH12/2

Explanation: MCU's RESET time interval is 1/2 period of PH12 clock.

A-3-4. RESET PIN TYPE

This mask option is used to set how the input signal on the RESET pin will be handled. Please refer to 2-2-2.

The mask options are:

(1) LEVEL

(2) PULSE

A-3-5. IOC1 FOR KEY RESET

This mask option sets whether the IOC1 pin can be used as the input pin for key reset.

The mask options are:

(1) USE

(2) NO USE

A-3-6. IOC2 FOR KEY RESET

This mask option sets whether the IOC2 pin can be used as the input pin for key reset.

The mask options are:

(1) USE

(2) NO USE

A-3-7. IOC3 FOR KEY RESET

This mask option sets whether the IOC3 pin can be used as the input pin for key reset.

The mask options are:

(1) USE

(2) NO USE

A-3-8. IOC4 FOR KEY RESET

This mask option sets whether the IOC4 pin can be used as the input pin for key reset.

The mask options are:

(1) USE

(2) NO USE

A-3-9. KI1 FOR KEY RESET

This mask option sets whether the KI1 pin can be used as the input pin for key reset. This option is only usable when the A-6-3 mask option is set to KI1.

The mask options are:

(1) USE

(2) NO USE

A-3-10. KI2 FOR KEY RESET

This mask option sets whether the KI2 pin can be used as the input pin for key reset. This option is only usable when the A-6-4 mask option is set to KI2.

The mask options are:

(1) USE

(2) NO USE

A-3-11. KI3 FOR KEY RESET

This mask option sets whether the KI3 pin can be used as the input pin for key reset. This option is only usable when the A-6-5 mask option is set to KI3.

The mask options are:

(1) USE

(2) NO USE

A-3-12. KI4 FOR KEY RESET

This mask option sets whether the KI4 pin can be used as the input pin for key reset. This option is only usable when the A-6-6 mask option is set to KI4.

The mask options are:

(1) USE

(2) NO USE

A-4. LCD

A-4-1. BIAS

This mask option sets the bias used by the LCD driver.

The mask options are:

(1) 1/3 BIAS (ONE CAPACTOR FOR CUP1-2)

Explanation: To use this option connect a capacitor between the CUP1 and CUP2 pins.

(2) 1/4 BIAS (TWO CAPACTOR FOR CUP1-2, CUP0-2)

Explanation: To use this option connect a capacitor between the CUP1 and CUP2 pins as well as the CUP0 and CUP2 pins.

(3) 1/5 BIAS (TWO CAPACTOR FOR CUP1-2, CUPN-0)

Explanation: To use this option, the charge-pump's base voltage must be supplied by an external voltage regulator (the VL5 pin's output voltage must be limited less than 6V). The mask option for the regulator must be set to VL1 or VL2.

(4) 1/3 BIAS (TWO CAPACTOR FOR CUP1-2, CUPN-0)

Explanation: To use this option, two capacitors must be connected between the CUP1/CUP2 and CUPN/CUP0 pins. This should only be used if the LCD loading is heavy.

(5) 1/4 BIAS (TWO CAPACTOR FOR CUP1-2, CUPN-0)

Explanation: To use this option, two capacitors must be connected between the CUP1/CUP2 and CUPN/CUP0 pins. This should only be used if the LCD loading is heavy.

A-4-2. LCD DUTY CYCLE

This mask option sets the duty cycle used by the LCD driver.

The mask options are:

(1) 1/4 DUTY

(2) 1/5 DUTY

(3) 1/6 DUTY

(4) 1/7 DUTY

(5) 1/8 DUTY

(6) 1/9 DUTY

(7) 1/10 DUTY

(8) 1/11 DUTY

(9) 1/12 DUTY

(a) 1/13 DUTY

(b) 1/14 DUTY

(c) 1/15 DUTY

(d) 1/16 DUTY

A-4-3. LCD ALTERNATING CLOCK FREQUENCY

This mask option sets the LCD Alternating Clock used by the output waveform when the COM and SEG pins are used by the LCD driver.

The mask options are:

(1) PH3

(2) PH4

(3) PH5

(4) PH6

(5) PH7

(6) PH8

(7) PH9

(8) PH10

A-4-4. CHARGE PUMP CYCLE TIME

This mask option sets the clock source used by the charge-pump circuit.

The mask options are:

(1) PH3

(2) PH4

(3) PH5

A-4-5. F SEGMENT FOR DISPLAY "7"

This mask option sets whether the “f” segment in the output from the 7-segment decoder for the digit 7 will be shown.

The mask options are:

(1) ON

Explanation: If using this option the f segment will appear in the pattern of 7-segment.

(2) OFF

Explanation: If using this option the f segment will not appear in the pattern of 7-segment.

A-4-6. NON-OVERLAP FOR COM&SEG OUTPUT

This mask option enables/disables the non-overlap signal output function for the COM and SEG pins. It can also set the characteristics of the non-overlap output signal.

The mask options are:

(1) ALL USE (for Key-Scanning Use)

Explanation: This option must be used if at least one of the KI1~4 input function is enabled by the mask options between A-6-3 ~ A-6-6. The non-overlap signal output function is enabled with the non-overlap signal output from SEG1 to SEG16 used as the scanning signal for the key matrix scanning function.

(2) ALL HI-Z (for Key-Scanning No Use)

Explanation: To use this option, all mask options between A-6-3 ~ A-6-6 CANNOT be set to KI1~4. This option enables the non-overlap output signal function but the output signal will be in the Hi-Z state, reducing the LCD driver's power consumption and power noise.

(3) ALL NO USE (for Key-Scanning No Use)

Explanation: To use this option, all mask options between A-6-3 ~ A-6-6 CANNOT be set to KI1~4. This option disables the non-overlap output function.

A-4-7. NON-OVERLAP WIDTH

This mask option set the width of the output signal when the non-overlap function is enabled.

The mask options are:

(1) SHORT(PH0)

(2) NORMAL(PH1)

A-5. RFC

A-5-1. RFC0 PAD TYPE FOR EXTERNAL COMPONENT

This mask option sets the type of the external component to be driven by the RFC0 pin.

The mask options are:

(1) RESISTANCE

(2) CAPACITANCE

A-5-2. RFC1 PAD TYPE FOR EXTERNAL COMPONENT

This mask option sets the type of the external component to be driven by the RFC1 pin.

The mask options are:

(1) RESISTANCE

(2) CAPACITANCE

A-5-3. RFC2 PAD TYPE FOR EXTERNAL COMPONENT

This mask option sets the type of the external component to be driven by the RFC2 pin.

The mask options are:

(1) RESISTANCE

(2) CAPACITANCE

A-5-4. RFC3 PAD TYPE FOR EXTERNAL COMPONENT

This mask option sets the type of the external component to be driven by the RFC3 pin.

The mask options are:

(1) RESISTANCE

(2) CAPACITANCE

A-5-5. RFC4 PAD TYPE FOR EXTERNAL COMPONENT

This mask option sets the type of the external component to be driven by the RFC4 pin.

The mask options are:

(1) RESISTANCE

(2) CAPACITANCE

A-5-6. RFC5 PAD TYPE FOR EXTERNAL COMPONENT

This mask option sets the type of the external component to be driven by the RFC5 pin.

The mask options are:

(1) RESISTANCE

(2) CAPACITANCE

A-5-7. RFC0~5 DISTRIBUTE BETWEEN CX AND CX2 GROUPS

This mask option is used to set whether the CX and CX2 pins should be used to form RC oscillators with the RFC0~5 pins, or used independently as the input pin for an external clock source.

The mask options are:

(1) RFC0~5 -> CX, CX2 USE

Explanation: If this option is used, the CX pin forms 6 separate RC oscillators with RFC0~5 while the CX2 pin used by itself as the input pin for an external clock signal.

(2) RFC0~4 -> CX, RFC5 -> CX2

Explanation: If this option is used, the CX pin forms 5 separate RC oscillators with RFC0~4 while the CX2 pin and RFC5 forms one RC oscillator.

(3) RFC0~3 -> CX, RFC4~5 -> CX2

Explanation: If this option is used, the CX pin forms 4 separate RC oscillators with RFC0~3 while the CX2 pin and RFC4~5 forms 2 separate RC oscillators.

(4) RFC0~2 -> CX, RFC3~5 -> CX2

Explanation: If this option is used, the CX pin forms 3 separate RC oscillators with RFC0~2 while the CX2 pin and RFC3~5 forms 3 separate RC oscillators.

(5) RFC0~1 -> CX, RFC2~5 -> CX2

Explanation: If this option is used, the CX pin forms 2 separate RC oscillators with RFC0~1 while the CX2 pin and RFC2~5 forms 4 separate RC oscillators.

(6) RFC0 -> CX, RFC1~5 -> CX2

Explanation: If this option is used, the CX pin forms one RC oscillator with RFC0 while the CX2 pin and RFC1~5 forms 5 separate RC oscillator.

(7) CX USE, RFC0~5 -> CX2

Explanation: If this option is used, the CX pin is used by itself as the input pin for an external clock signal while the CX2 pin and RFC0~5 forms 6 separate RC oscillators.

(8) RFC0~5 -> CX, CX2 NO USE

Explanation: If this option is used, the CX pin and RFC0~5 forms 6 separate RC oscillators while the input function on the CX2 pin is disabled.

(9) CX NO USE, RFC0~5 -> CX2

Explanation: If this option is used, the input function on the CX pin is disabled while the CX2 pin and RFC0~5 forms 6 separate RC oscillators.

(A) CX & CX2 NO USE

Explanation: This option will disable the input function on both the CX and CX2 pin.

A-5-8. CX OUTPUT LOW TO DISCHARGE FOR NO ACTIVE

This mask options sets whether to use the high-impedance input mode or low-impedance input mode when the CX input pin becomes inactive. This option can only be used if mask option A-5-7 doesn't disable the CX pin's input function.

The mask options are:

(1) USE

Explanation: Using this option selects the low-impedance input mode by activating the pull-low component.

(2) NO USE

Explanation: Using this option selects the high-impedance input mode by disabling the pull-low component.

A-5-9. CX2 OUTPUT LOW TO DISCHARGE FOR NO ACTIVE

This mask options sets whether to use the high-impedance input mode or low-impedance input mode when the CX2 input pin becomes inactive. This option can only be used if mask option A-5-7 doesn't disable the CX2 pin's input function.

The mask options are:

(1) USE

Explanation: Using this option selects the low-impedance input mode by activating the pull-low component.

(2) NO USE

Explanation: Using this option selects the high-impedance input mode by disabling the pull-low component.

A-5-10. RFC OVERFLOW DISABLE 16 BITS COUNTER

This mask option sets whether the 16-bit RFC counter should stop or restart its count after an overflow.

The mask options are:

(1) USE

Explanation: This option sets the counter to stop immediately.

(2) NO USE

Explanation: This option sets the counter to restart its count automatically.

A-6. PAD

A-6-1. INT PIN INTERNAL RESISTOR

This mask option sets the INT pin's mode.

The mask options are:

(1) PULL HIGH

Explanation: This option sets INT pin to pull-high input mode.

(2) PULL LOW

Explanation: This option sets INT pin to pull-low input mode.

(3) OPEN TYPE

Explanation: This option sets the INT input pin to high-impedance input mode.

A-6-2. INT PIN TRIGGER MODE

This mask option sets the signal format used by the INT input pin when generating an external interrupt request.

The mask options are:

(1) RISING EDGE

(2) FALLING EDGE

A-6-3. IOB1/KI1

This mask option sets the function of the IOB1/KI1 shared pin.

The mask options are:

(1) IOB1

(2) KI1

A-6-4. IOB2/KI2

This mask option sets the function of the IOB2/KI2 shared pin.

The mask options are:

(1) IOB2

(2) KI2

A-6-5. IOB3/KI3

This mask option sets the function of the IOB3/KI3 shared pin.
The mask options are:

(1) IOB3

(2) KI3

A-6-6. IOB4/KI4

This mask option sets the function of the IOB4/KI4 shared pin.
The mask options are:

(1) IOB4

(2) KI4

A-6-7. IOA PULL LOW RESISTOR

This mask option sets whether to enable the pull-low resistor on the IOA port's 4 input pins.
The mask options are:

(1) USE

(2) NO USE

A-6-8. IOB PULL LOW RESISTOR

This mask option sets whether to enable the pull-low resistor on the IOB port's 4 input pins.
The mask options are:

(1) USE

(2) NO USE

A-6-9. IOC PULL LOW RESISTOR

This mask option sets whether to enable the pull-low resistor on the IOC port's 4 input pins.
The mask options are:

(1) USE

(2) NO USE

A-6-10. IOD PULL LOW RESISTOR

This mask option sets whether to enable the pull-low resistor on the IOD port's 4 input pins.
The mask options are:

(1) USE

(2) NO USE

A-6-11. IOE PULL LOW RESISTOR

This mask option sets whether to enable the pull-low resistor on the IOE port's 4 input pins.

The mask options are:

(1) USE

(2) NO USE

A-6-12. C PORT LOW LEVEL HOLD

This mask option sets whether to enable the low-level hold function on the IOC port's 4 input pins. This option is only usable when the IOC port's pull-low resistor option is enabled.

The mask options are:

(1) USE

(2) NO USE

A-6-13. IOA CHATTERING PREVENTION

This mask options sets whether to enable the chattering prevention function on the IOA port.

The mask options are:

(1) USE

(2) NO USE

A-6-14. IOC CHATTERING PREVENTION

This mask options sets whether to enable the chattering prevention function on the IOC port.

The mask options are:

(1) USE

(2) NO USE

A-6-15. IOD CHATTERING PREVENTION

This mask options sets whether to enable the chattering prevention function on the IOD port.

The mask options are:

(1) USE

(2) NO USE

A-6-16. COM5/C_DC5/C_OD5

This mask option sets whether the COM5 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM5

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/5 duty ~ 1/16 duty.

(2) C_DC5

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 duty.

(3) C_OD5

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 duty.

A-6-17. COM6/C_DC6/C_OD6

This mask option sets whether the COM6 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM6

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/6 duty ~ 1/16 duty.

(2) C_DC6

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 or 1/5 duty.

(3) C_OD6

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 or 1/5 duty.

A-6-18. COM7/C_DC7/C_OD7

This mask option sets whether the COM7 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM7

Explanation: If this option is used, COM7 is assigned to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/7 duty ~ 1/16 duty.

(2) C_DC7

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/6 duty.

(3) C_OD7

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/6 duty.

A-6-19. COM8/C_DC8/C_OD8

This mask option sets whether the COM8 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM8

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/8 duty ~ 1/16 duty.

(2) C_DC8

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/7 duty.

(3) C_OD8

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/7 duty.

A-6-20. COM9/C_DC9/C_OD9

This mask option sets whether the COM9 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM9

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/9 duty ~ 1/16 duty.

(2) C_DC9

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/8 duty.

(3) C_OD9

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/8 duty.

A-6-21. COM10/C_DC10/C_OD10

This mask option sets whether the COM10 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM10

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/10 duty ~ 1/16 duty.

(2) C_DC10

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/9 duty.

(3) C_OD10

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/9 duty.

A-6-22. COM11/C_DC11/C_OD11

This mask option sets whether the COM11 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM11

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/11 duty ~ 1/16 duty.

(2) C_DC11

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/10 duty.

(3) C_OD11

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/10 duty.

A-6-23. COM12/C_DC12/C_OD12

This mask option sets whether the COM12 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM12

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/12 duty ~ 1/16 duty.

(2) C_DC12

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/11 duty.

(3) C_OD12

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/11 duty.

A-6-24. COM13/C_DC13/C_OD13

This mask option sets whether the COM13 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM13

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/13 duty ~ 1/16 duty.

(2) C_DC13

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/12 duty.

(3) C_OD13

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/12 duty.

A-6-25. COM14/C_DC14/C_OD14

This mask option sets whether the COM14 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM14

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/14 duty ~ 1/16 duty.

(2) C_DC14

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/13 duty.

(3) C_OD14

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/13 duty.

A-6-26. COM15/C_DC15/C_OD15

This mask option sets whether the COM15 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM15

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/15 duty ~ 1/16 duty.

(2) C_DC15

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/14 duty.

(3) C_OD15

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/14 duty.

A-6-27. COM16/C_DC16/C_OD16

This mask option sets whether the COM16 output pin is to be used by the LCD driver or as a normal output pin.

The mask options are:

(1) COM16

Explanation: If this option is used, assign to the LCD driver. This option is only usable when the A-4-2 mask option is set to 1/16 duty.

(2) C_DC16

Explanation: If this option is used, set as COMS type DC output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/15 duty.

(3) C_OD16

Explanation: If this option is used, set as P-type open-drain output pin. This option is only usable when the A-4-2 mask option is set to 1/4 ~ 1/15 duty.

A-7. UTILITY

A-7-1. INSTRUCTION ROM(K Word) <-> TABLE ROM(K Byte)

This mask option sets how much address space is allocated in program memory for storing program instructions and table ROM.

The mask options are:

(1) 16.0 <-> 32
(2) 16.5 <-> 31
(3) 17.0 <-> 30
(4) 17.5 <-> 29
(5) 18.0 <-> 28
(6) 18.5 <-> 27
(7) 19.0 <-> 26
(8) 19.5 <-> 25
(9) 20.0 <-> 24
(A) 20.5 <-> 23
(B) 21.0 <-> 22
(C) 21.5 <-> 21
(D) 22.0 <-> 20
(E) 22.5 <-> 19
(F) 23.0 <-> 18
(G) 23.5 <-> 17
(H) 24.0 <-> 16
(I) 24.5 <-> 15
(J) 25.0 <-> 14
(K) 25.5 <-> 13
(L) 26.0 <-> 12
(M) 26.5 <-> 11
(N) 27.0 <-> 10
(O) 27.5 <-> 9
(P) 28.0 <-> 8
(Q) 28.5 <-> 7
(R) 29.0 <-> 6
(S) 29.5 <-> 5
(T) 30.0 <-> 4
(U) 30.5 <-> 3
(V) 31.0 <-> 2
(W) 31.5 <-> 1
(X) 32.0 <-> 0